

```

1  Attribute VB_Name = "BitBoard32"
2  Option Explicit
3
4  ' bitboard 32 bit with 2 long variables
5  Public Declare Sub CopyMemory _
6      Lib "kernel32" _
7      Alias "RtlMoveMemory" (ByVal Destination As Long, _
8          ByVal Source As Long, _
9          ByVal Length As Long)
10
11
12
13  Public Const MIN_INTEGER As Integer = -32768
14  Public Const MAX_INTEGER As Integer = 32767
15
16  Public Const BitL_0 As Long = &H1&
17  Public Const BitL_1 As Long = &H2&
18  Public Const BitL_2 As Long = &H4&
19  Public Const BitL_3 As Long = &H8&
20  Public Const BitL_4 As Long = &H10&
21  Public Const BitL_5 As Long = &H20&
22  Public Const BitL_6 As Long = &H40&
23  Public Const BitL_7 As Long = &H80&
24  Public Const BitL_8 As Long = &H100&
25  Public Const BitL_9 As Long = &H200&
26  Public Const BitL_10 As Long = &H400&
27  Public Const BitL_11 As Long = &H800&
28  Public Const BitL_12 As Long = &H1000&
29  Public Const BitL_13 As Long = &H2000&
30  Public Const BitL_14 As Long = &H4000&
31  Public Const BitL_15 As Long = &H8000&
32  Public Const BitL_16 As Long = &H10000
33  Public Const BitL_17 As Long = &H20000
34  Public Const BitL_18 As Long = &H40000
35  Public Const BitL_19 As Long = &H80000
36  Public Const BitL_20 As Long = &H100000
37  Public Const BitL_21 As Long = &H200000
38  Public Const BitL_22 As Long = &H400000
39  Public Const BitL_23 As Long = &H800000
40  Public Const BitL_24 As Long = &H1000000
41  Public Const BitL_25 As Long = &H2000000
42  Public Const BitL_26 As Long = &H4000000
43  Public Const BitL_27 As Long = &H8000000
44  Public Const BitL_28 As Long = &H10000000
45  Public Const BitL_29 As Long = &H20000000
46  Public Const BitL_30 As Long = &H40000000
47  Public Const BitL_31 As Long = &H80000000
48
49  Public Const RANK1_L = BitL_0 Or BitL_1 Or BitL_2 Or BitL_3 Or BitL_4 Or BitL_5 Or
    BitL_6 Or BitL_7
50  Public Const RANK2_L = BitL_8 Or BitL_9 Or BitL_10 Or BitL_11 Or BitL_12 Or BitL_13 Or
    BitL_14 Or BitL_15
51  Public Const RANK3_L = BitL_16 Or BitL_17 Or BitL_18 Or BitL_19 Or BitL_20 Or BitL_21
    Or BitL_22 Or BitL_23
52  Public Const RANK4_L = BitL_24 Or BitL_25 Or BitL_26 Or BitL_27 Or BitL_28 Or BitL_29
    Or BitL_30 Or BitL_31
53
54
55  Public Type TBit64 ' emulate 64 bit, use 4x16 bit (positive values only)
56      i0 As Long
57      i1 As Long
58  End Type
59
60  Public Type TInt16x2
61      i0 As Integer
62      i1 As Integer
63  End Type
64

```

```

65 Public Type TInt16x4
66     i0 As Integer
67     i1 As Integer
68     i2 As Integer
69     i3 As Integer
70 End Type
71
72 Public Type TByte8x8
73     i0 As Byte
74     i1 As Byte
75     i2 As Byte
76     i3 As Byte
77     i4 As Byte
78     i5 As Byte
79     i6 As Byte
80     i7 As Byte
81 End Type
82
83 '-----
84
85 Public FILEA_BB As TBit64, FILEB_BB As TBit64, FILEC_BB As TBit64, FILED_BB As TBit64,
86     FILEE_BB As TBit64, FILEF_BB As TBit64, FILEG_BB As TBit64, FILEH_BB As TBit64
87 Public RANK1_BB As TBit64, RANK2_BB As TBit64, RANK3_BB As TBit64, RANK4_BB As TBit64,
88     RANK5_BB As TBit64, RANK6_BB As TBit64, RANK7_BB As TBit64, RANK8_BB As TBit64
89
90 Public Bit32Pos(31) As Long
91 Public Pop16Cnt(MIN_INTEGER To MAX_INTEGER) As Byte ' Max -int to +int
92 Public Int16x4 As TInt16x4
93 Public Int16x2 As TInt16x2
94 Public Byte8x8 As TByte8x8
95
96 Public Bit8Pos(7) As Integer
97
98 Public PiecesBB(WCOL, PT_ALL_PIECES) As TBit64, AllPiecesBB As TBit64, PiecesByPtBB(
99     PT_ALL_PIECES) As TBit64, ColBB(WCOL) As TBit64, AttackedByBB(WCOL, PT_ALL_PIECES) As
100     TBit64, AttackedBy2BB(WCOL) As TBit64
101 Public SquareBB(MAX_BOARD) As TBit64
102 Public EmptyBB As TBit64
103 Public FileBB(8) As TBit64
104 Public RankBB(8) As TBit64
105 Public AdjacentFilesBB(8) As TBit64
106 Public ForwardRanksBB(WCOL, 8) As TBit64
107 Public ForwardFileBB(WCOL, MAX_BOARD) As TBit64
108 Public PawnAttackSpanBB(WCOL, MAX_BOARD) As TBit64
109 Public PawnAttackSpanAllBB(WCOL) As TBit64
110 Public PassedPawnMaskBB(WCOL, MAX_BOARD) As TBit64
111 Public OutpostRanksBB(WCOL) As TBit64
112 Public SqToBit(MAX_BOARD) As Long
113 Public BitToSq(63) As Long
114 Public PawnAttacksFromSqBB(WCOL, SQ_H8) As TBit64
115 Public PseudoAttacksFromSqBB(PIECE_TYPE_NB, MAX_BOARD) As TBit64
116 Public AttackFromToBB(MAX_BOARD, MAX_BOARD) As TBit64
117 Public BetweenBB(MAX_BOARD, MAX_BOARD) As TBit64
118 Public KingRingBB(WCOL) As TBit64
119 Public LowRanksBB(WCOL) As TBit64
120 Public CampBB(WCOL) As TBit64
121 Public CenterBB As TBit64
122 Public CenterFilesBB As TBit64
123 Public DarkSquaresBB As TBit64
124 Public LSB16(MIN_INTEGER To MAX_INTEGER) As Integer
125 Public RSB16(MIN_INTEGER To MAX_INTEGER) As Integer
126
127 Public BB0 As TBit64, BB1 As TBit64
128
129 '-----
130
131 Public Sub Init32BitBoards()

```

```

129 Dim i As Long, j As Long, k As Long, SqBB As Long
130
131 For i = 0 To 7: Bit8Pos(i) = 2 ^ i: Next
132 For i = 0 To 31: Bit32Pos(i) = BitMask32(i): Next
133
134 For j = MIN_INTEGER To MAX_INTEGER
135     Pop16Cnt(j) = Pop16CountFkt(j)
136     LSB16(j) = -1
137     For i = 0 To 15
138         If CBool(j And Bit32Pos(i)) Then LSB16(j) = i: Exit For
139     Next
140     RSB16(j) = -1
141     For i = 15 To 0 Step -1
142         If CBool(j And Bit32Pos(i)) Then RSB16(j) = i: Exit For
143     Next
144 Next
145
146 SqBB = 0
147 For i = 0 To 119
148     SqToBit(i) = -1
149     If Board(i) <> FRAME Then
150         SqToBit(i) = SqBB
151         BitToSq(SqBB) = i
152
153         '--- set ranks
154         Select Case Rank(i)
155             Case 1: SetBit64 RANK1_BB, SqBB
156             Case 2: SetBit64 RANK2_BB, SqBB
157             Case 3: SetBit64 RANK3_BB, SqBB
158             Case 4: SetBit64 RANK4_BB, SqBB
159             Case 5: SetBit64 RANK5_BB, SqBB
160             Case 6: SetBit64 RANK6_BB, SqBB
161             Case 7: SetBit64 RANK7_BB, SqBB
162             Case 8: SetBit64 RANK8_BB, SqBB
163         End Select
164
165         '--- set Files
166         Select Case File(i)
167             Case 1: SetBit64 FILEA_BB, SqBB
168             Case 2: SetBit64 FILEB_BB, SqBB
169             Case 3: SetBit64 FILEC_BB, SqBB
170             Case 4: SetBit64 FILED_BB, SqBB
171             Case 5: SetBit64 FILEE_BB, SqBB
172             Case 6: SetBit64 FILEF_BB, SqBB
173             Case 7: SetBit64 FILEG_BB, SqBB
174             Case 8: SetBit64 FILEH_BB, SqBB
175         End Select
176
177         SetBit64 SquareBB(i), SqBB
178         If ColorSq(i) = BCOL Then SetBit64 DarkSquaresBB, SqBB
179     End If
180     SqBB = SqBB + 1
181 End If
182 Next i
183
184 FileBB(FILE_A) = FILEA_BB
185 FileBB(FILE_B) = FILEB_BB
186 FileBB(FILE_C) = FILEC_BB
187 FileBB(FILE_D) = FILED_BB
188 FileBB(FILE_E) = FILEE_BB
189 FileBB(FILE_F) = FILEF_BB
190 FileBB(FILE_G) = FILEG_BB
191 FileBB(FILE_H) = FILEH_BB
192
193 RankBB(1) = RANK1_BB
194 RankBB(2) = RANK2_BB
195 RankBB(3) = RANK3_BB
196 RankBB(4) = RANK4_BB

```

```

197 RankBB(5) = RANK5_BB
198 RankBB(6) = RANK6_BB
199 RankBB(7) = RANK7_BB
200 RankBB(8) = RANK8_BB
201
202 OR64 LowRanksBB(WCOL), RANK2_BB, RANK3_BB
203 OR64 LowRanksBB(BCOL), RANK6_BB, RANK7_BB
204 OR64 CenterFilesBB, FILED_BB, FILEE_BB
205 OR64 BB0, RANK4_BB, RANK5_BB
206 AND64 CenterBB, CenterFilesBB, BB0
207 OR64 OutpostRanksBB(WCOL), RANK4_BB, RANK5_BB: OR64 OutpostRanksBB(WCOL),
OutpostRanksBB(WCOL), RANK6_BB
208 OR64 OutpostRanksBB(BCOL), RANK3_BB, RANK4_BB: OR64 OutpostRanksBB(BCOL),
OutpostRanksBB(BCOL), RANK5_BB
209
210 AdjacentFilesBB(FILE_A) = FILEB_BB
211 OR64 AdjacentFilesBB(FILE_B), FILEA_BB, FILEC_BB
212 OR64 AdjacentFilesBB(FILE_C), FILEA_BB, FILED_BB
213 OR64 AdjacentFilesBB(FILE_D), FILEA_BB, FILEE_BB
214 OR64 AdjacentFilesBB(FILE_E), FILEA_BB, FILEF_BB
215 OR64 AdjacentFilesBB(FILE_F), FILEA_BB, FILEG_BB
216 OR64 AdjacentFilesBB(FILE_G), FILEA_BB, FILEH_BB
217 AdjacentFilesBB(FILE_H) = FILEG_BB
218
219 ForwardRanksBB(WCOL, 7) = RANK8_BB
220 OR64 ForwardRanksBB(WCOL, 6), ForwardRanksBB(WCOL, 7), RANK7_BB
221 OR64 ForwardRanksBB(WCOL, 5), ForwardRanksBB(WCOL, 6), RANK6_BB
222 OR64 ForwardRanksBB(WCOL, 4), ForwardRanksBB(WCOL, 5), RANK5_BB
223 OR64 ForwardRanksBB(WCOL, 3), ForwardRanksBB(WCOL, 4), RANK4_BB
224 OR64 ForwardRanksBB(WCOL, 2), ForwardRanksBB(WCOL, 3), RANK3_BB
225 OR64 ForwardRanksBB(WCOL, 1), ForwardRanksBB(WCOL, 2), RANK2_BB
226
227 ForwardRanksBB(BCOL, 2) = RANK1_BB
228 OR64 ForwardRanksBB(BCOL, 3), ForwardRanksBB(BCOL, 2), RANK2_BB
229 OR64 ForwardRanksBB(BCOL, 4), ForwardRanksBB(BCOL, 3), RANK3_BB
230 OR64 ForwardRanksBB(BCOL, 5), ForwardRanksBB(BCOL, 4), RANK4_BB
231 OR64 ForwardRanksBB(BCOL, 6), ForwardRanksBB(BCOL, 5), RANK5_BB
232 OR64 ForwardRanksBB(BCOL, 7), ForwardRanksBB(BCOL, 6), RANK6_BB
233 OR64 ForwardRanksBB(BCOL, 8), ForwardRanksBB(BCOL, 7), RANK7_BB
234
235 CampBB(WCOL) = ForwardRanksBB(BCOL, 6)
236 CampBB(BCOL) = ForwardRanksBB(WCOL, 3)
237
238 ' Init SqFrom attacks
239 Dim d As Long, Col As Long, Offset As Long
240
241 For Col = BCOL To WCOL
242 For i = SQ_A1 To SQ_H8
243 If Board(i) <> FRAME Then
244 ' Pawn attacks
245 If Col = WCOL Then j = i + 9 Else j = i - 9
246 If Board(j) <> FRAME Then
247 SetBit64 PawnAttacksFromSqBB(Col, i), SqToBit(j)
248 End If
249 If Col = WCOL Then j = i + 11 Else j = i - 11
250 If Board(j) <> FRAME Then
251 SetBit64 PawnAttacksFromSqBB(Col, i), SqToBit(j)
252 End If
253
254 AND64 ForwardFileBB(Col, i), ForwardRanksBB(Col, Rank(i)), FileBB(File(i))
255 AND64 PawnAttackSpanBB(Col, i), ForwardRanksBB(Col, Rank(i)), AdjacentFilesBB(
File(i))
256 OR64 PassedPawnMaskBB(Col, i), ForwardFileBB(Col, i), PawnAttackSpanBB(Col, i)
257
258 If Col = WCOL Then ' same for black
259 ' King/Knight attacks
260 For d = 0 To 7
261 Offset = QueenOffsets(d)

```

```

262     j = i + Offset
263     If Board(j) <> FRAME Then
264         SetBit64 PseudoAttacksFromSqBB(PT_KING, i), SqToBit(j) 'King
265     '
266
267     Do While Board(j) <> FRAME
268         SetBit64 PseudoAttacksFromSqBB(PT_QUEEN, i), SqToBit(j) 'Queen
269
270         If Board(j) <> FRAME Then
271             If j <> i + Offset Then
272                 SetBit64 BetweenBB(i, j), SqToBit(j - Offset) 'between 2 squares,
                current square
273                 SetOR64 BetweenBB(i, j), BetweenBB(i, j - Offset) 'previous squares
                in line
274             End If
275             AttackFromToBB(i, j) = BetweenBB(i, j): SetBit64 AttackFromToBB(i, j
                ), SqToBit(j) 'includes target square
276         End If
277
278         If d < 4 Then
279             SetBit64 PseudoAttacksFromSqBB(PT_ROOK, i), SqToBit(j) 'Rook
280         Else
281             SetBit64 PseudoAttacksFromSqBB(PT_BISHOP, i), SqToBit(j) 'Bishop
282         End If
283         j = j + Offset
284     Loop
285 End If
286
287 '---
288 j = i + KnightOffsets(d)
289 If Board(j) <> FRAME Then
290     SetBit64 PseudoAttacksFromSqBB(PT_KNIGHT, i), SqToBit(j) 'Knight
291 End If
292 Next d
293 End If
294 End If
295 Next i
296 Next Col
297 End Sub
298
299 Function BitMask32(ByVal BitPos As Long) As Long '32 bit
300     'If BitPos < 0 Or BitPos > 31 Then Err.Raise 6 'overflow
301     If BitPos < 31 Then
302         BitMask32 = 2 ^ BitPos
303     Else
304         BitMask32 = BitL_31
305     End If
306 End Function
307
308 Public Function Pop16CountFkt(ByVal x As Long) As Long
309     'for positive values only
310     Pop16CountFkt = 0: If x = 0 Then Exit Function
311     If x < 0 Then Pop16CountFkt = Pop16CountFkt + 1: x = x And Not &H8000
312     Do While x > 0
313         Pop16CountFkt = Pop16CountFkt + 1: x = x And (x - 1)
314     Loop
315 End Function
316
317 Public Sub AND64(Result As TBit64, Op1 As TBit64, Op2 As TBit64)
318     Result.i0 = Op1.i0 And Op2.i0: Result.i1 = Op1.i1 And Op2.i1
319 End Sub
320
321 Public Sub SetAND64(Op1 As TBit64, Op2 As TBit64) 'returns Op1
322     Op1.i0 = Op1.i0 And Op2.i0: Op1.i1 = Op1.i1 And Op2.i1
323 End Sub
324
325 Public Sub SetANDNOT64(Op1 As TBit64, Op2 As TBit64) 'returns Op1
326     Op1.i0 = Op1.i0 And Not Op2.i0: Op1.i1 = Op1.i1 And Not Op2.i1

```

```

327 End Sub
328
329 Public Sub OR64(Result As TBit64, Op1 As TBit64, Op2 As TBit64)
330     Result.i0 = Op1.i0 Or Op2.i0: Result.i1 = Op1.i1 Or Op2.i1
331 End Sub
332
333 Public Sub SetOR64(Op1 As TBit64, Op2 As TBit64) 'returns Op1
334     Op1.i0 = Op1.i0 Or Op2.i0: Op1.i1 = Op1.i1 Or Op2.i1
335 End Sub
336
337 Public Sub XOr64(Result As TBit64, Op1 As TBit64, Op2 As TBit64)
338     Result.i0 = Op1.i0 Xor Op2.i0: Result.i1 = Op1.i1 Xor Op2.i1
339 End Sub
340
341 Public Sub ANDNOT64(Result As TBit64, Op1 As TBit64, Op2 As TBit64)
342     Result.i0 = Op1.i0 And Not Op2.i0: Result.i1 = Op1.i1 And Not Op2.i1
343 End Sub
344
345 Public Sub SetNOT64(Result As TBit64, Op1 As TBit64)
346     Result.i0 = Not Op1.i0: Result.i1 = Not Op1.i1
347 End Sub
348
349 Public Sub Set64(Result As TBit64, Op1 As TBit64)
350     Result.i0 = Op1.i0: Result.i1 = Op1.i1 ' much faster then Result=Op1 !!!!
351 End Sub
352
353 Public Function EQUAL64(Op1 As TBit64, Op2 As TBit64) As Boolean
354     If Op1.i0 = Op2.i0 Then
355         If Op1.i1 = Op2.i1 Then EQUAL64 = True Else EQUAL64 = False
356     Else
357         EQUAL64 = False
358     End If
359 End Function
360
361 Public Sub Clear64(Op1 As TBit64)
362     Op1.i0 = 0: Op1.i1 = 0
363 End Sub
364
365 Public Function ShiftDown64(Op1 As TBit64) As TBit64
366     ' shift right 8 bits
367     ' LSet Byte8x8 = Op1
368     ' Byte8x8.i0 = Byte8x8.i1
369     ' Byte8x8.i1 = Byte8x8.i2
370     ' Byte8x8.i2 = Byte8x8.i3
371     ' Byte8x8.i3 = Byte8x8.i4
372     ' Byte8x8.i4 = Byte8x8.i5
373     ' Byte8x8.i5 = Byte8x8.i6
374     ' Byte8x8.i6 = Byte8x8.i7
375     ' Byte8x8.i7 = 0
376     ' LSet ShiftDown64 = Byte8x8
377
378     ' i1
379     If Op1.i1 And BitL_31 Then
380         ShiftDown64.i1 = (((Op1.i1 And Not BitL_31) \ &H100&) Or BitL_23) And Not RANK4_L
381         ' shift 8 bits down (= &H100& ), remove rank 4 and add sign bit 31 as bit 23
382     Else
383         ShiftDown64.i1 = (Op1.i1 \ &H100&) And Not RANK4_L
384     End If
385
386     ' Copy RANK5 to RANK4 > copy to i0 bits 0-6 (= &H7F&) of rank1 and shift 24 bits (= &H1000000) up
387     If Op1.i1 And BitL_7 Then
388         ShiftDown64.i0 = ((Op1.i1 And &H7F&) * &H1000000) Or BitL_31 ' copy bit 7 from i1 to i0
389         sign bit
390     Else
391         ShiftDown64.i0 = (Op1.i1 And &H7F&) * &H1000000
392     End If
393
394     ' i0

```

```

393     If Op1.i0 And BitL_31 Then
394         ShiftDown64.i0 = ShiftDown64.i0 Or (((Op1.i0 And Not BitL_31) \ &H100&) Or
          BitL_23) And Not RANK4_L) ' shift 8 bits down (= &H100& ), remove rank 4 and add sign bit 31 as bit 23
395     Else
396         ShiftDown64.i0 = ShiftDown64.i0 Or ((Op1.i0 \ &H100&) And Not RANK4_L)
397     End If
398     'ShowLBB ShiftDown64
399 End Function
400
401 Public Function ShiftUp64(Op1 As TBit64) As TBit64
402     ' shift left 8 bits
403     ' LSet Byte8x8 = Op1
404     ' Byte8x8.i7 = Byte8x8.i6
405     ' Byte8x8.i6 = Byte8x8.i5
406     ' Byte8x8.i5 = Byte8x8.i4
407     ' Byte8x8.i4 = Byte8x8.i3
408     ' Byte8x8.i3 = Byte8x8.i2
409     ' Byte8x8.i2 = Byte8x8.i1
410     ' Byte8x8.i1 = Byte8x8.i0
411     ' Byte8x8.i0 = 0
412     ' LSet ShiftUp64 = Byte8x8
413
414     'i0
415     If Op1.i0 And BitL_23 Then
416         ShiftUp64.i0 = ((Op1.i0 And Not RANK4_L And Not BitL_23) * &H100& Or BitL_31)
417     Else
418         ShiftUp64.i0 = (Op1.i0 And Not RANK4_L) * &H100&
419     End If
420     'i1
421     If Op1.i1 And BitL_23 Then
422         ShiftUp64.i1 = ((Op1.i1 And Not RANK4_L And Not BitL_23) * &H100&) Or BitL_31
423     Else
424         ShiftUp64.i1 = (Op1.i1 And Not RANK4_L) * &H100&
425     End If
426     ' Copy RANK5 to RANK4 > copy to i1 bits 24-30 (= &H7F000000) of rank4 and shift 24 bits (= &H1000000) down
427     If Op1.i0 And BitL_31 Then
428         ShiftUp64.i1 = ShiftUp64.i1 Or ((Op1.i0 And &H7F000000) \ &H1000000) Or BitL_7 '
          copy sign bit 31 from I0 to I1 bit 7
429     Else
430         ShiftUp64.i1 = ShiftUp64.i1 Or ((Op1.i0 And &H7F000000) \ &H1000000)
431     End If
432 End Function
433
434 Public Function ShiftLeft64(Op1 As TBit64) As TBit64
435     ShiftLeft64.i0 = Op1.i0 And Not FILEA_BB.i0: ShiftLeft64.i1 = Op1.i1 And Not
          FILEA_BB.i1 ' remove file A
436     ShiftLeft64.i0 = ShiftLeft64.i0 \ &H2& And Not BitL_31
437     ShiftLeft64.i1 = ShiftLeft64.i1 \ &H2& And Not BitL_31
438 End Function
439
440 Public Function ShiftRight64(Op1 As TBit64) As TBit64
441     ShiftRight64.i0 = Op1.i0 And Not FILEH_BB.i0: ShiftRight64.i1 = Op1.i1 And Not
          FILEH_BB.i1 ' remove file H
442     If ShiftRight64.i0 And BitL_30 Then
443         ShiftRight64.i0 = ((ShiftRight64.i0 And Not BitL_30) * &H2&) Or BitL_31 ' move bit30
          to bit31 else overflow
444     Else
445         ShiftRight64.i0 = (ShiftRight64.i0 And &HFFFFFFF&) * &H2&
446     End If
447     If ShiftRight64.i1 And BitL_30 Then
448         ShiftRight64.i1 = ((ShiftRight64.i1 And Not BitL_30) * &H2&) Or BitL_31
449     Else
450         ShiftRight64.i1 = (ShiftRight64.i1 And &HFFFFFFF&) * &H2&
451     End If
452 End Function
453
454 Public Function ShiftUpOrDown64(ByVal UpDown As Long, Op1 As TBit64) As TBit64
455     If UpDown = SQ_UP Then

```

```

456     ShiftUpOrDown64 = ShiftUp64(Op1)
457     ElseIf UpDown = SQ_DOWN Then
458         ShiftUpOrDown64 = ShiftDown64(Op1)
459     End If
460 End Function
461
462
463 Public Sub SetBit64(Op1 As TBit64, ByVal BitPos As Long)
464     ' bitPos 0 to 63
465     Debug.Assert BitPos >= 0 And BitPos < 64
466     'If BitPos < 0 Then Exit Sub
467     'If BitPos > 63 Then Exit Sub
468
469     If BitPos < 32 Then
470         If BitPos = 31 Then Op1.i0 = Op1.i0 Or BitL_31 Else Op1.i0 = Op1.i0 Or Bit32Pos(
            BitPos)
471     Else
472         BitPos = BitPos - 32
473         If BitPos = 31 Then Op1.i1 = Op1.i1 Or BitL_31 Else Op1.i1 = Op1.i1 Or Bit32Pos(
            BitPos)
474     End If
475 End Sub
476
477 Public Function IsSetBit64(Op1 As TBit64, ByVal BitPos As Long) As Boolean
478     ' bitPos 0 to 63
479     Debug.Assert BitPos >= 0 And BitPos < 64
480     'If BitPos < 0 Then Exit Sub
481     'If BitPos > 63 Then Exit Sub
482
483     If BitPos < 32 Then
484         IsSetBit64 = CBool(Op1.i0 And Bit32Pos(BitPos))
485     Else
486         IsSetBit64 = CBool(Op1.i1 And Bit32Pos(BitPos - 32))
487     End If
488 End Function
489
490 Public Function IsSet64(Op1 As TBit64) As Boolean
491     If Op1.i0 <> 0 Then IsSet64 = True: Exit Function
492     If Op1.i1 <> 0 Then IsSet64 = True: Exit Function
493     IsSet64 = False
494 End Function
495
496
497 Public Sub ClearBit64(Op1 As TBit64, ByVal BitPos As Long)
498     ' bitPos 0 to 63
499     Debug.Assert BitPos < 64
500     If BitPos < 32 Then
501         Op1.i0 = Op1.i0 And Not Bit32Pos(BitPos)
502     Else
503         Op1.i1 = Op1.i1 And Not Bit32Pos(BitPos - 32)
504     End If
505 End Sub
506
507
508 Public Function PopCnt64(Op1 As TBit64) As Long
509     LSet Int16x4 = Op1
510     PopCnt64 = Pop16Cnt(Int16x4.i0) + Pop16Cnt(Int16x4.i1) + Pop16Cnt(Int16x4.i2) +
        Pop16Cnt(Int16x4.i3)
511 End Function
512
513 Public Sub ShowBB(bb As TBit64)
514     Dim i As Long, s As String
515     Debug.Print
516     Debug.Print " -----"
517     s = ""
518     For i = 63 To 0 Step -1
519         If (i + 1) Mod 8 = 0 And s <> "" Then
520             Debug.Print CStr((i + 9) \ 8) & "|" & s & "|"

```



```

521     s = ""
522     End If
523     If IsSetBit64(bb, ByVal i) Then s = " X" & s Else s = " ." & s
524 Next
525 Debug.Print CStr((i + 9) \ 8) & "|" & s & "|"
526 Debug.Print " -----"
527 Debug.Print "   A B C D E F G H"
528 Debug.Print
529
530 End Sub
531
532 Public Function Lsb64(Op1 As TBit64) As Long
533     'returns position of first bit set
534     Lsb64 = -1
535     If Op1.i0 <> 0 Then
536         LSet Int16x4 = Op1
537         Lsb64 = LSB16(Int16x4.i0): If Lsb64 >= 0 Then Exit Function
538         Lsb64 = LSB16(Int16x4.i1): If Lsb64 >= 0 Then Lsb64 = Lsb64 + 16: Exit Function
539     ElseIf Op1.i1 <> 0 Then
540         LSet Int16x4 = Op1
541         Lsb64 = LSB16(Int16x4.i2): If Lsb64 >= 0 Then Lsb64 = Lsb64 + 32: Exit Function
542         Lsb64 = LSB16(Int16x4.i3): If Lsb64 >= 0 Then Lsb64 = Lsb64 + 48
543     End If
544 End Function
545
546
547 Public Function Rsb64(Op1 As TBit64) As Long
548     'returns position of last bit set
549     Rsb64 = -1
550     If Op1.i1 <> 0 Then
551         LSet Int16x4 = Op1
552         Rsb64 = RSB16(Int16x4.i3): If Rsb64 >= 0 Then Rsb64 = Rsb64 + 48: Exit Function
553         Rsb64 = RSB16(Int16x4.i2): If Rsb64 >= 0 Then Rsb64 = Rsb64 + 32: Exit Function
554     ElseIf Op1.i0 <> 0 Then
555         LSet Int16x4 = Op1
556         Rsb64 = RSB16(Int16x4.i1): If Rsb64 >= 0 Then Rsb64 = Rsb64 + 16: Exit Function
557         Rsb64 = RSB16(Int16x4.i0)
558     End If
559 End Function
560
561 Public Function PopLsb64(Op1 As TBit64) As Long
562     PopLsb64 = Lsb64(Op1)
563     If PopLsb64 >= 0 Then ClearBit64 Op1, PopLsb64
564 End Function
565
566
567 Public Function PawnAttacksBB(ByRef Col As enumColor, Op1 As TBit64) As TBit64
568     If Col = WCOL Then
569         PawnAttacksBB = ShiftUp64(Op1)
570         OR64 PawnAttacksBB, ShiftLeft64(PawnAttacksBB), ShiftRight64(PawnAttacksBB)
571     ElseIf Col = BCOL Then
572         PawnAttacksBB = ShiftDown64(Op1)
573         OR64 PawnAttacksBB, ShiftLeft64(PawnAttacksBB), ShiftRight64(PawnAttacksBB)
574     End If
575 End Function
576
577 Public Function AttacksBoardBB(ByVal pt As enumPieceType, ByVal sq As Long) As TBit64
578     Dim LastTarget As Long, Target As Long, Offset As Long, d As Long, DirStart As
579     Long, DirEnd As Long
580     AttacksBoardBB = EmptyBB
581
582     Select Case pt
583     Case PT_ROOK: DirStart = 0: DirEnd = 3
584     Case PT_BISHOP: DirStart = 4: DirEnd = 7
585     Case PT_QUEEN: DirStart = 0: DirEnd = 7
586     Case Else
587         Exit Function
588     End Select

```

```

588
589     For d = DirStart To DirEnd
590         Offset = QueenOffsets(d): Target = sq + Offset: LastTarget = sq
591         Do While Board(Target) <> FRAME
592             LastTarget = Target
593             If Board(Target) >= NO_PIECE Then Exit Do
594             Target = Target + Offset
595         Loop
596         If sq <> LastTarget Then
597             If MaxDistance(sq, LastTarget) = 1 Then
598                 SetBit64 AttacksBoardBB, SqToBit(LastTarget)
599             Else
600                 '--- add bitboards for direction
601                 AttacksBoardBB.i0 = AttacksBoardBB.i0 Or AttackFromToBB(sq, LastTarget).i0
602                 : AttacksBoardBB.i1 = AttacksBoardBB.i1 Or AttackFromToBB(sq, LastTarget).
603                 i1
604             End If
605         End If
606     Next
607 End Function
608
609 Public Function AttacksBB(ByVal pt As enumPieceType, ByVal sq As Long, occupied As
610 TBit64) As TBit64
611     Dim LastTarget As Long, Target As Long, Offset As Long, d As Long, DirStart As
612     Long, DirEnd As Long
613
614     AttacksBB = EmptyBB
615
616     Select Case pt
617     Case PT_ROOK: DirStart = 0: DirEnd = 3
618     Case PT_BISHOP: DirStart = 4: DirEnd = 7
619     Case PT_QUEEN: DirStart = 0: DirEnd = 7
620     Case Else
621         Exit Function
622     End Select
623
624     For d = DirStart To DirEnd
625         Offset = QueenOffsets(d): Target = sq + Offset: LastTarget = sq
626
627         Do While Board(Target) <> FRAME
628             LastTarget = Target: If IsSetBit64(occupied, SqToBit(Target)) Then Exit Do
629             Target = Target + Offset
630         Loop
631         If sq <> LastTarget Then SetOR64 AttacksBB, AttackFromToBB(sq, LastTarget) '---
632         add bitboards for direction
633     Next
634 End Function
635
636 Public Function MoreThanOne(op1BB As TBit64) As Boolean
637     MoreThanOne = CBool(PopCnt64(op1BB) > 1)
638 End Function
639
640 Public Function FrontMostSq(Us As enumColor, Op1 As TBit64) As Long
641     'returns first square board position relative for color
642     If Us = WCOL Then FrontMostSq = Rsb64(Op1) Else FrontMostSq = Lsb64(Op1)
643     If FrontMostSq >= 0 Then FrontMostSq = BitToSq(FrontMostSq) Else FrontMostSq = 0
644 End Function
645
646 Public Function BackMostSq(Us As enumColor, Op1 As TBit64) As Long
647     'returns first square board position relative for color
648     If Us = WCOL Then BackMostSq = Lsb64(Op1) Else BackMostSq = Rsb64(Op1)
649     If BackMostSq >= 0 Then BackMostSq = BitToSq(BackMostSq) Else BackMostSq = 0
650 End Function
651
652 Public Sub Or64To(Op1 As TBit64, Op2 As TBit64, Result As TBit64)
653     Result.i0 = Op1.i0 Or Op2.i0: Result.i1 = Op1.i1 Or Op2.i1
654 End Sub

```

```

651
652
653 '-----
654 Public Sub SetMove(m1 As TMOVE, m2 As TMOVE)
655     With m1
656         .Captured = m2.Captured
657         .CapturedNumber = m2.CapturedNumber
658         .Castle = m2.Castle
659         .EnPassant = m2.EnPassant
660         .From = m2.From
661         .IsChecking = m2.IsChecking
662         .IsLegal = m2.IsLegal
663         .OrderValue = m2.OrderValue
664         .Piece = m2.Piece
665         .Promoted = m2.Promoted
666         .SeeValue = m2.SeeValue
667         .Target = m2.Target
668     End With
669 End Sub
670
671 Public Sub SwapMove(m1 As TMOVE, m2 As TMOVE)
672     Dim t As TMOVE
673     With t
674         .Captured = m2.Captured: m2.Captured = m1.Captured: m1.Captured = .Captured
675         .CapturedNumber = m2.CapturedNumber: m2.CapturedNumber = m1.CapturedNumber:
676         m1.CapturedNumber = .CapturedNumber
677         .Castle = m2.Castle: m2.Castle = m1.Castle: m1.Castle = .Castle
678         .EnPassant = m2.EnPassant: m2.EnPassant = m1.EnPassant: m1.EnPassant = .EnPassant
679         .From = m2.From: m2.From = m1.From: m1.From = .From
680         .IsChecking = m2.IsChecking: m2.IsChecking = m1.IsChecking: m1.IsChecking = .
681         IsChecking
682         .IsLegal = m2.IsLegal: m2.IsLegal = m1.IsLegal: m1.IsLegal = .IsLegal
683         .OrderValue = m2.OrderValue: m2.OrderValue = m1.OrderValue: m1.OrderValue = .
684         OrderValue
685         .Piece = m2.Piece: m2.Piece = m1.Piece: m1.Piece = .Piece
686         .Promoted = m2.Promoted: m2.Promoted = m1.Promoted: m1.Promoted = .Promoted
687         .SeeValue = m2.SeeValue: m2.SeeValue = m1.SeeValue: m1.SeeValue = .SeeValue
688         .Target = m2.Target: m2.Target = m1.Target: m1.Target = .Target
689     End With
690 End Sub
691
692 Public Sub ClearMove(m1 As TMOVE)
693     With m1
694         .From = 0: .Target = 0: .Piece = NO_PIECE: .Castle = NO_CASTLE: .Promoted = 0: .
695         Captured = NO_PIECE: .CapturedNumber = 0
696         .EnPassant = 0: .IsChecking = False: .IsLegal = False: .OrderValue = 0: .SeeValue
697         = UNKNOWN_SCORE
698     End With
699 End Sub
700
701 Public Function Test64()
702     Dim b As TBit64, bb As TBit64, i As Long, t As TBit64, x As Long
703     InitEngine
704     Init32BitBoards
705     '----
706     Dim StartTime As Single, EndTime As Single, y As Long, z As Long, sq As Long, j As
707     Long
708     Dim m1 As TMOVE, m2 As TMOVE, m3 As TMOVE
709     StartTime = Timer
710
711     b = EmptyBB: x = Len(m1)
712     t.i0 = 123

```

```

713 m1.From = 2: m2.From = 12: m3.Target = 34
714 For i = 1 To 50000000
715
716     SetMove m1, EmptyMove ' 2x schneller
717     SetMove m2, m3
718     SetMove m3, m1
719
720     ' m1 = EmptyMove
721     ' m2 = m3
722     ' m3 = m1
723
724     '1. ---
725     'x = 23 + i Mod 2
726     'SetBit64 b, x
727     'If x < 32& Then If x = 31 Then b.i0 = b.i0 Or BitL_31 Else b.i0 = b.i0 Or Bit32Pos(x) Else If x = 63 Then b.i1 = b.i1
       Or BitL_31 Else b.i1 = b.i1 Or Bit32Pos(x - 32)
728
729     'x = x + 20
730     'SetBit64 b, x
731     'If x < 32& Then If x = 31 Then b.i0 = b.i0 Or BitL_31 Else b.i0 = b.i0 Or Bit32Pos(x) Else If x = 63 Then b.i1 = b.i1
       Or BitL_31 Else b.i1 = b.i1 Or Bit32Pos(x - 32)
732
733
734
735     'For j = 1 To 7
736     ' bb.i0 = AttackedByBB(1, j).i0: bb.i1 = AttackedByBB(1, j).i1
737     ' b.i0 = bb.i0: b.i1 = bb.i1
738     'b.i0 = bb.i0 Or t.i0: b.i1 = bb.i1 Or t.i1
739
740
741     'bb = AttackedByBB(1, j)
742     'Set64 b, bb
743     'b = bb
744     'OR64 b, bb, t
745     'Next
746     '-----
747     'z = 31 + i Mod 2
748     '2.---
749     'AttackedByBB(1, PT_PAWN).i0 = AttackedByBB(1, PT_PAWN).i0 Or SquareBB(z).i0: AttackedByBB(1,
       PT_PAWN).i1 = AttackedByBB(1, PT_PAWN).i1 Or SquareBB(z).i1
750     'SetOR64 AttackedByBB(1, PT_PAWN), SquareBB(z)
751
752     '3.---
753     'bb.i0 = b.i0 Or SquareBB(z).i0: bb.i1 = b.i1 Or SquareBB(z).i1
754     ' bb = Or64(b, SquareBB(z)) ' 24,6
755     'Or64To b, SquareBB(z), bb ' 3,7
756     'Or64ToP b, SquareBB(z) ' 2,6
757
758
759 Next
760
761 EndTime = Timer
762 Debug.Print Format$(EndTime - StartTime, "0.000")
763 Debug.Print y
764 MsgBox Format$(EndTime - StartTime, "0.000") & " " & bb.i0 & bb.i1 & x &
       m1.Target & m2.Target & b.i1 & t.i1 & b.i0 & AttackedByBB(1, PT_PAWN).i0
765
766 End Function
767 Attribute VB_Name = "basBoard"
768 '=====
769 '= basBoard:
770 '= Board structure and move generation
771 '=====
772 Option Explicit
773 ' Index in array Board(119): A1=21, A8=28, H1=91, H8=98
774 ' frame needed for move generation (max knight move distance = 2+1 squares)
775 ' 110 -- -- -- -- -- 119
776 ' 100 -- -- -- -- -- 109

```

```

777 ' 90 -- A8 B8 C8 D8 E8 F8 G8 H8 -- 99
778 ' 80 -- A7 B7 C7 D7 E7 F7 G7 H7 -- 89
779 ' 70 -- A6 B6 C6 D6 E6 F6 G6 H6 -- 79
780 ' 60 -- A5 B5 C5 D5 E5 F5 G5 H5 -- 69
781 ' 50 -- A4 B4 C4 D4 E4 F4 G4 H4 -- 59
782 ' 40 -- A3 B3 C3 D3 E3 F3 G3 H3 -- 49
783 ' 30 -- A2 B2 C2 D2 E2 F2 G2 H2 -- 39
784 ' 20 -- A1 B1 C1 D1 E1 F1 G1 H1 -- 29
785 ' 10 -- - - - - - - - - - - 19
786 ' 0 -- - - - - - - - - - - 9
787 '
788 Public Board(MAX_BOARD) As Long ' Game board for all moves
789 Public NumPieces As Long '--- Current number of pieces at ply 0
in Pieces list
790 Public Pieces(32) As Long '--- List of pieces: pointer to board
position (Captured pieces are set to zero during search)
791 Public Squares(MAX_BOARD) As Long '--- Squares on board: pointer to
pieces list (Captured pieces are set to zero during search)
792 Public ColorSq(MAX_BOARD) As Long '--- Squares color: COL_WHITE or
COL_BLACK
793 Public PieceCnt(16) As Long ' number of pieces per piece type
and color
794 Public SameXRay(MAX_BOARD, MAX_BOARD) As Boolean ' are two squares on same rank or
file or diagonal?
795 'Public SameRookRay(MAX_BOARD, MAX_BOARD) As Boolean ' are two squares on same rank or file or
diagonal?
796 'Public SameBishopRay(MAX_BOARD, MAX_BOARD) As Boolean ' are two squares on same rank or file or
diagonal?
797 Public DirOffset(MAX_BOARD, MAX_BOARD) As Integer ' direction offset from sq1 to sq 2
798 Public bWhiteToMove As Boolean '--- side to move , false if black
to move, often used
799 Public bCompIsWhite As Boolean
800 Public CastleFlag As enumCastleFlag
801 Public WhiteCastled As enumCastleFlag
802 Public BlackCastled As enumCastleFlag
803 Public WPromotions(5) As Long '--- list of promotion pieces
804 Public BPromotions(5) As Long
805 Public WKingLoc As Long ' white king location
806 Public BKingLoc As Long ' black king location
807 Public PieceType(16) As Long ' sample: maps black pawn and
white pawn pieces to PT_PAWN
808 Public PieceColor(16) As Long ' white / Black
809 Public Ply As Long ' current ply
810 Public Fifty As Long ' counter for fifty move draw rule :
100 half moves
811 Public arFiftyMove(499) As Long ' fifty counter for ply
812 Public Rank(MAX_BOARD) As Long ' Rank from white view
813 Public RankB(MAX_BOARD) As Long ' Rank from black view 1 - 8
814 Public RelativeSq(COL_WHITE, MAX_BOARD) As Long ' sq from black view 1 - 8
815 Public File(MAX_BOARD) As Long ' file on board 1 - 8
816 Public SqBetween(MAX_BOARD, MAX_BOARD, MAX_BOARD) As Boolean ' (sq,sq1,sq2) is sq between sq1
and sq2?
817 '--- For faster move generation
818 Public WhitePiecesStart As Long ' used for access to PieceList
819 Public WhitePiecesEnd As Long ' used for access to PieceList
820 Public BlackPiecesStart As Long ' used for access to PieceList
821 Public BlackPiecesEnd As Long ' used for access to PieceList
822 Public WNonPawnPieces As Long ' counts pieces
823 Public BNonPawnPieces As Long ' counts pieces
824 '--- SEE data ( static exchange evaluation )
825 Dim PieceList(0 To 32) As Long, Cnt As Long
826 Dim SwapList(0 To 32) As Long, slIndex As Long
827 Dim Blocker(1 To 32) As Long, Block As Long
828 '-----
829 Public StartupBoard(MAX_BOARD) As Long ' Start Position used for copy to
current board
830 Public Moved(MAX_BOARD) As Long ' Track for moved pieces (castle
checks + eval)

```

```

831 Public KingCheckW(MAX_BOARD) As Integer ' for fast checking moves
      detection, integer for faster ERASE
832 Public KingCheckB(MAX_BOARD) As Integer ' for fast checking moves detection
833 ' Offsets of directions - for move generation
834 Public DirectionOffset(7) As Long
835 Public KnightOffsets(7) As Long
836 Public BishopOffsets(3) As Long
837 Public RookOffsets(3) As Long
838 Public OppositeDir(-11 To 11) As Long
839 Public EpPosArr(0 To MAX_DEPTH) As Long
840 Public MaxDistance(0 To SQ_H8, 0 To SQ_H8) As Long ' max distance between two fields
841 Private bGenCapturesOnly As Boolean ' generate QSearch -captures only
842 Private bGenQsChecks As Boolean ' generate QSearch checks
843 '-----
844
845 '-----
846 ' GenerateMoves()
847 ' =====
848 ' Generates all Pseudo-legal move for a position. Check for legal moves later with CheckLegal
849 ' if bCapturesOnly then only captures and promotions are generated.
850 ' if MovePickerDat(Ply).GenerateQsChecksCnt then checks are generated too. For QSearch first ply only.
851 '-----
852 Public Function GenerateMoves(ByVal Ply As Long, _
853                               ByVal bCapturesOnly As Boolean, _
854                               NumMoves As Long) As Long
855     Dim From As Long, i As Long
856     '--- Init special board with king checking positions for fast detection of checking moves
857     If bWhiteToMove Then FillKingCheckB Else FillKingCheckW
858
859     bGenCapturesOnly = bCapturesOnly: NumMoves = 0
860     bGenQsChecks = (MovePickerDat(Ply).GenerateQsChecksCnt > 0)
861     If bWhiteToMove Then
862
863         For i = WhitePiecesStart To WhitePiecesEnd
864             From = Pieces(i)
865             Debug.Assert (From >= SQ_A1 And From <= SQ_H8) Or From = 0 ' from=0 if piece was
               captured during search
866
867             Select Case Board(From)
868                 Case NO_PIECE, FRAME
869                 Case WPAWN
870                     ' note: FRAME has Bit 1 not set - like BCOL: PieceColor() cannot be used here, returns NO_COL for
                       EP piece
871                     If ((Board(From + 11) And 1) = BCOL) Then If Board(From + 11) <> FRAME Then
                       TryMoveWPawn Ply, NumMoves, From, From + 11 ' capture right side
872                     If ((Board(From + 9) And 1) = BCOL) Then If Board(From + 9) <> FRAME Then
                       TryMoveWPawn Ply, NumMoves, From, From + 9 ' capture left side
873                     If Board(From + 10) = NO_PIECE Then ' one row up
874                         If Rank(From) = 2 Then If Board(From + 20) = NO_PIECE Then TryMoveWPawn
                       Ply, NumMoves, From, From + 20 ' two rows up
875                         TryMoveWPawn Ply, NumMoves, From, From + 10 ' one row up
876                     End If
877                 Case WKNIGHT
878                     TryMoveListKnight Ply, NumMoves, From
879                 Case WBISHOP
880                     TryMoveSliderList Ply, NumMoves, From, PT_BISHOP
881                 Case WROOK
882                     TryMoveSliderList Ply, NumMoves, From, PT_ROOK
883                 Case WQUEEN
884                     TryMoveSliderList Ply, NumMoves, From, PT_QUEEN
885                 Case WKING
886                     TryMoveListKing Ply, NumMoves, From
887                     ' Check castling
888                     If From = WKING_START Then
889                         If Moved(WKING_START) = 0 Then
890                             'o-o
891                             If Moved(SQ_H1) = 0 And Board(SQ_H1) = WROOK Then
892                                 If Board(SQ_F1) = NO_PIECE And Board(SQ_G1) = NO_PIECE Then

```

```

893         CastleFlag = WHITEOO
894         TryCastleMove Ply, NumMoves, From, From + 2
895         CastleFlag = NO_CASTLE
896     End If
897 End If
898 'O-O-O
899 If Moved(SQ_A1) = 0 And Board(SQ_A1) = WROOK Then
900     If Board(SQ_D1) = NO_PIECE And Board(SQ_C1) = NO_PIECE And Board(SQ_B1
901         ) = NO_PIECE Then
902         CastleFlag = WHITEOOO
903         TryCastleMove Ply, NumMoves, From, From - 2
904         CastleFlag = NO_CASTLE
905     End If
906 End If
907 End If
908 End Select
909
910 Next
911
912 Else
913
914     For i = BlackPiecesStart To BlackPiecesEnd
915         From = Pieces(i)
916         Debug.Assert (From >= SQ_A1 And From <= SQ_H8) Or From = 0
917
918         Select Case Board(From)
919             Case NO_PIECE, FRAME
920             Case BPAWN
921                 ' note: NO_PIECE has Bit 1 set like WCOL
922                 If ((Board(From - 11) And 1) = WCOL) And Board(From - 11) <> NO_PIECE Then
923                     TryMoveBPawn Ply, NumMoves, From, From - 11
924                 If ((Board(From - 9) And 1) = WCOL) And Board(From - 9) <> NO_PIECE Then
925                     TryMoveBPawn Ply, NumMoves, From, From - 9
926                 If Board(From - 10) = NO_PIECE Then
927                     If Rank(From) = 7 Then If Board(From - 20) = NO_PIECE Then TryMoveBPawn
928                         Ply, NumMoves, From, From - 20
929                     TryMoveBPawn Ply, NumMoves, From, From - 10
930                 End If
931             Case BKNIGHT
932                 TryMoveListKnight Ply, NumMoves, From
933             Case BBISHOP
934                 TryMoveSliderList Ply, NumMoves, From, PT_BISHOP
935             Case BROOK
936                 TryMoveSliderList Ply, NumMoves, From, PT_ROOK
937             Case BQUEEN
938                 TryMoveSliderList Ply, NumMoves, From, PT_QUEEN
939             Case BKING
940                 TryMoveListKing Ply, NumMoves, From
941                 ' Check castling
942                 If From = BKING_START Then
943                     If Moved(BKING_START) = 0 Then
944                         'O-O
945                         If Moved(SQ_H8) = 0 And Board(SQ_H8) = BROOK Then
946                             If Board(SQ_F8) = NO_PIECE And Board(SQ_G8) = NO_PIECE Then
947                                 CastleFlag = BLACKOO
948                                 TryCastleMove Ply, NumMoves, From, From + 2
949                                 CastleFlag = NO_CASTLE
950                             End If
951                         End If
952                         'O-O-O
953                         If Moved(SQ_A8) = 0 And Board(SQ_A8) = BROOK Then
954                             If Board(SQ_D8) = NO_PIECE And Board(SQ_C8) = NO_PIECE And Board(SQ_B8
955                                 ) = NO_PIECE Then
956                                 CastleFlag = BLACKOOO
957                                 TryCastleMove Ply, NumMoves, From, From - 2
958                                 CastleFlag = NO_CASTLE
959                             End If

```



```

956         End If
957     End If
958 End If
959 End Select
960
961 Next
962
963 End If
964 GenerateMoves = NumMoves ' return move count
965 End Function
966
967 Private Function TryMoveWPawn(ByVal Ply As Long, _
968                               NumMoves As Long, _
969                               ByVal From As Long, _
970                               ByVal Target As Long) As Boolean
971     If Board(Target) = FRAME Then Exit Function
972     Dim PieceFrom As Long, PieceTarget As Long, bDoCheckMove As Boolean
973     PieceFrom = Board(From): PieceTarget = Board(Target)
974     Debug.Assert PieceTarget <> FRAME
975
976     If Rank(From) = 7 Then
977         ' White Promotion
978         Dim PromotePiece As Long
979         For PromotePiece = 1 To 4 ' for each promotion piece type
980             With Moves(Ply, NumMoves)
981                 .From = From: .Target = Target: .Captured = PieceTarget: .EnPassant = 0: .
982                 Castle = NO_CASTLE: .Promoted = WPromotions(PromotePiece): .Piece = .Promoted
983                 : .IsChecking = False: .IsLegal = False: .SeeValue = VALUE_NONE: .OrderValue
984                 = 0
985             End With
986             NumMoves = NumMoves + 1
987         Next
988     Else
989         With Moves(Ply, NumMoves)
990             Select Case PieceTarget
991             Case BEP_PIECE
992                 .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
993                 IsChecking = False: .Castle = NO_CASTLE: .Captured = PieceTarget: .
994                 CapturedNumber = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .OrderValue = 0
995                 .EnPassant = ENPASSANT_CAPTURE: NumMoves = NumMoves + 1
996             Case NO_PIECE, WEP_PIECE ' WEP_PIECE should not appear
997                 '--- Normal move, not a capture, promotion ---
998                 bDoCheckMove = False
999                 '--- in QSearch: Generate checking moves only for first QSearch ply
1000                 If bGenCapturesOnly And bGenQsChecks Then If IsCheckingMove(PieceFrom, From,
1001                                     Target, 0, 0) Then bDoCheckMove = True
1002                 If Not bGenCapturesOnly Or bDoCheckMove Then
1003                     '---Normal move, not generated in QSearch (exception: when in check)
1004                     .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
1005                     EnPassant = 0: .Castle = NO_CASTLE: .Captured = PieceTarget: .CapturedNumber
1006                     = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .OrderValue = 0
1007                     If Target - From = 20 Then .EnPassant = ENPASSANT_WMOVE
1008                     .IsChecking = bDoCheckMove: NumMoves = NumMoves + 1
1009                 End If
1010             Case FRAME
1011             Case Else
1012                 ' Normal capture.
1013                 .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
1014                 IsChecking = False: .EnPassant = 0: .Castle = NO_CASTLE: .Captured =
1015                 PieceTarget: .CapturedNumber = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .
1016                 OrderValue = 0
1017                 NumMoves = NumMoves + 1
1018             End Select
1019         End With
1020     End If
1021 End Function
1022

```



```

1013 Private Function TryMoveBPawn(ByVal Ply As Long, _
1014                               NumMoves As Long, _
1015                               ByVal From As Long, _
1016                               ByVal Target As Long) As Boolean
1017 If Board(Target) = FRAME Then Exit Function
1018 Dim PieceFrom As Long, PieceTarget As Long
1019 PieceFrom = Board(From): PieceTarget = Board(Target)
1020 Debug.Assert PieceTarget <> FRAME
1021
1022 If Rank(From) = 2 Then
1023     ' Black Promotion
1024     Dim PromotePiece As Long
1025     For PromotePiece = 1 To 4
1026         With Moves(Ply, NumMoves)
1027             .From = From: .Target = Target: .Captured = PieceTarget: .EnPassant = 0: .
1028             Castle = NO_CASTLE: .Promoted = BPromotions(PromotePiece): .Piece = .Promoted
1029             : .IsChecking = False: .IsLegal = False: .SeeValue = VALUE_NONE: .OrderValue
1030             = 0
1031         End With
1032         NumMoves = NumMoves + 1
1033     Next
1034 Else
1035     With Moves(Ply, NumMoves)
1036         Select Case PieceTarget
1037         Case WEP_PIECE
1038             .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
1039             IsChecking = False: .Castle = NO_CASTLE: .Captured = PieceTarget: .
1040             CapturedNumber = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .OrderValue = 0
1041             .EnPassant = ENPASSANT_CAPTURE: NumMoves = NumMoves + 1
1042         Case NO_PIECE, BEP_PIECE ' BEP_PIECE should not appear
1043             '--- Normal move, not a capture, promotion ---
1044             Dim bDoCheckMove As Boolean
1045             bDoCheckMove = False
1046             '--- in QSearch: Generate checking moves only for first QSearch ply
1047             If bGenCapturesOnly And bGenQsChecks Then If IsCheckingMove(PieceFrom, From,
1048             Target, 0, 0) Then bDoCheckMove = True
1049             If Not bGenCapturesOnly Or bDoCheckMove Then
1050                 '---Normal move, not generated in QSearch (exception: when in check)
1051                 .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
1052                 EnPassant = 0: .Castle = NO_CASTLE: .Captured = PieceTarget: .CapturedNumber
1053                 = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .OrderValue = 0
1054                 If Target - From = -20 Then .EnPassant = ENPASSANT_BMOVE
1055                 .IsChecking = bDoCheckMove: NumMoves = NumMoves + 1
1056             End If
1057         Case FRAME
1058         Case Else
1059             ' Normal capture.
1060             .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
1061             IsChecking = False: .EnPassant = 0: .Castle = NO_CASTLE: .Captured =
1062             PieceTarget: .CapturedNumber = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .
1063             OrderValue = 0
1064             NumMoves = NumMoves + 1
1065         End Select
1066     End With
1067 End If
1068 End Function
1069
1070 Private Function TryMoveListKnight(ByVal Ply As Long, _
1071                                   NumMoves As Long, _
1072                                   ByVal From As Long) As Boolean
1073
1074 '--- Knights only moves
1075 Dim Target As Long, ActDir As Long, PieceFrom As Long, PieceTarget As Long,
1076 bDoCheckMove As Boolean, PieceCol As Long
1077 PieceFrom = Board(From): PieceCol = (PieceFrom And 1)
1078
1079 For ActDir = 0 To 7
1080     Target = From + KnightOffsets(ActDir): PieceTarget = Board(Target)
1081     Select Case PieceTarget

```

```

1069 Case NO_PIECE, WEP_PIECE, BEP_PIECE
1070 '--- Normal move, not a capture, castle, promotion ---
1071 '--- in QSearch: Generate checking moves only for first QSearch ply
1072 If bGenCapturesOnly And bGenQsChecks Then bDoCheckMove = IsCheckingMove(
PieceFrom, From, Target, 0, 0)
1073 If Not bGenCapturesOnly Or bDoCheckMove Then
1074 '---Normal move, not generated in QSearch (exception: when in check)
1075 With Moves(Ply, NumMoves)
1076 .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
IsChecking = bDoCheckMove: .EnPassant = 0: .Castle = NO_CASTLE: .Captured =
PieceTarget: .CapturedNumber = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .
OrderValue = 0
1077 End With
1078 NumMoves = NumMoves + 1
1079 End If
1080 Case FRAME 'go on with next direction
1081 Case Else
1082 'Captures
1083 If PieceCol <> (PieceTarget And 1) Then 'Capture of own piece not allowed
1084 With Moves(Ply, NumMoves)
1085 .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
IsChecking = False: .EnPassant = 0: .Castle = NO_CASTLE: .Captured =
PieceTarget: .CapturedNumber = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .
OrderValue = 0
1086 End With
1087 NumMoves = NumMoves + 1
1088 End If
1089 End Select
1090 Next ActDir
1091
1092 End Function
1093
1094 Private Function TryMoveListKing(ByVal Ply As Long, _
1095 NumMoves As Long, _
1096 ByVal From As Long) As Boolean
1097 '--- Kings only
1098 Dim Target As Long, ActDir As Long, PieceFrom As Long, PieceTarget As Long, PieceCol
As Long
1099
1100 PieceFrom = Board(From): PieceCol = (PieceFrom And 1)
1101
1102 For ActDir = 0 To 7
1103 Target = From + DirectionOffset(ActDir): PieceTarget = Board(Target)
1104 Select Case PieceTarget
1105 Case NO_PIECE, WEP_PIECE, BEP_PIECE
1106 '--- Normal move, not a capture, castle, not generated in QSearch (exception: when in check)---
1107 If Not bGenCapturesOnly Then
1108 '---Normal move, not generated in QSearch (exception: when in check)
1109 With Moves(Ply, NumMoves)
1110 .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
IsChecking = False: .EnPassant = 0: .Castle = NO_CASTLE: .Captured =
PieceTarget: .CapturedNumber = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .
OrderValue = 0
1111 End With
1112 NumMoves = NumMoves + 1
1113 End If
1114 Case FRAME 'go on with next direction
1115 Case Else
1116 'Captures
1117 If PieceCol <> (PieceTarget And 1) Then 'Capture of own piece not allowed
1118 With Moves(Ply, NumMoves)
1119 .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
IsChecking = False: .EnPassant = 0: .Castle = NO_CASTLE: .Captured =
PieceTarget: .CapturedNumber = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .
OrderValue = 0
1120 End With
1121 NumMoves = NumMoves + 1
1122 End If

```

```

1123     End Select
1124 Next ActDir
1125
1126 End Function
1127
1128 Private Function TryCastleMove(ByVal Ply As Long, _
1129                               NumMoves As Long, _
1130                               ByVal From As Long, _
1131                               ByVal Target As Long) As Boolean
1132 If Board(Target) = FRAME Then Exit Function
1133 Dim CurrentMove As TMOVE, PieceFrom As Long, PieceTarget As Long
1134 PieceFrom = Board(From): PieceTarget = Board(Target): TryCastleMove = False
1135 If CastleFlag <> NO_CASTLE Then
1136     If Not bGenCapturesOnly Then
1137         CurrentMove.From = From
1138         CurrentMove.Target = Target
1139         CurrentMove.Piece = PieceFrom
1140         CurrentMove.Captured = PieceTarget
1141         CurrentMove.EnPassant = 0
1142         CurrentMove.Castle = CastleFlag
1143         CurrentMove.Promoted = 0: CurrentMove.IsChecking = False
1144         CurrentMove.SeeValue = VALUE_NONE
1145         CastleFlag = NO_CASTLE
1146         SetMove Moves(Ply, NumMoves), CurrentMove
1147         NumMoves = NumMoves + 1
1148         TryCastleMove = True
1149     End If
1150 End If
1151 End Function
1152
1153 Private Sub TryMoveSliderList(ByVal Ply As Long, _
1154                              NumMoves As Long, _
1155                              ByVal From As Long, _
1156                              ByVal PieceType As Long)
1157 Dim Target As Long, ActDir As Long, Offset As Long
1158 Dim PieceFrom As Long, PieceTarget As Long, bDoCheckMove As Boolean, DirStart As
1159 Long, DirEnd As Long, PieceCol As Long
1160
1161 PieceFrom = Board(From): PieceCol = (PieceFrom And 1)
1162
1163 Select Case PieceType 'get move directions
1164 Case PT_ROOK: DirStart = 0: DirEnd = 3 'Rook
1165 Case PT_BISHOP: DirStart = 4: DirEnd = 7 'Bishop
1166 Case Else: DirStart = 0: DirEnd = 7 'Queen
1167 End Select
1168
1169 For ActDir = DirStart To DirEnd 'for all possible directions
1170 Offset = DirectionOffset(ActDir): Target = From + Offset
1171 Do While Board(Target) <> FRAME '--- Slide loop
1172     PieceTarget = Board(Target)
1173     If PieceTarget < NO_PIECE Then 'Captures, not EnPassant
1174         If PieceCol <> (PieceTarget And 1) Then 'Capture of own piece not allowed, color in last
1175             bit of piece (even/uneven)
1176             ' Capture: add move to list
1177             With Moves(Ply, NumMoves)
1178                 .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
1179                 IsChecking = False: .EnPassant = 0: .Castle = NO_CASTLE: .Captured =
1180                 PieceTarget: .CapturedNumber = 0: .Promoted = 0: .SeeValue = VALUE_NONE:
1181                 .OrderValue = 0
1182             End With
1183             NumMoves = NumMoves + 1
1184         End If
1185     Exit Do '<<< end for this direction
1186 End If
1187 '--- Normal move, not a capture, castle, promotion ---
1188 '--- in QSearch: Generate checking moves only for first QSearch ply
1189 If bGenCapturesOnly And bGenQsChecks Then bDoCheckMove = IsCheckingMove(
1190 PieceFrom, From, Target, 0, 0) Else bDoCheckMove = False

```

```

1185     If Not bGenCapturesOnly Or bDoCheckMove Then '---Normal move, not generated in QSearch
1186         (exception: when in check)
1187         With Moves(Ply, NumMoves) ' add move to list
1188             .From = From: .Target = Target: .Piece = PieceFrom: .IsLegal = False: .
1189             IsChecking = bDoCheckMove: .EnPassant = 0: .Castle = NO_CASTLE: .Captured
1190             = NO_PIECE: .CapturedNumber = 0: .Promoted = 0: .SeeValue = VALUE_NONE: .
1191             OrderValue = 0
1192         End With
1193         NumMoves = NumMoves + 1
1194     End If
1195     Target = Target + Offset
1196 Loop
1197 Next ActDir
1198 End Sub
1199
1200 Public Function CheckLegalNotInCheck(mMove As TMOVE) As Boolean
1201     ' fast check for legal move: not for castling and when in check
1202     Dim Offset As Long, Target As Long, Piece As Long
1203     CheckLegalNotInCheck = False
1204     If mMove.From < SQ_A1 Then Exit Function
1205
1206     If bWhiteToMove Then
1207         If mMove.Piece = BKING Then
1208             If IsAttackedByW(mMove.Target) Then Exit Function
1209         Else
1210             If Not SameXRay(mMove.From, BKingLoc) Then CheckLegalNotInCheck = True: Exit
1211             Function
1212             If SqBetween(mMove.From, BKingLoc, mMove.Target) Then CheckLegalNotInCheck =
1213             True: Exit Function
1214
1215             Offset = DirOffset(BKingLoc, mMove.From): Target = mMove.From + Offset: Piece =
1216             Board(Target)
1217             Do While Piece <> FRAME
1218                 If Piece < NO_PIECE Then
1219                     Select Case Abs(Offset)
1220                     Case 1, 10:
1221                         If Piece = WROOK Or Piece = WQUEEN Then
1222                             Exit Do ' still to check other direction
1223                         Else
1224                             CheckLegalNotInCheck = True: Exit Function
1225                         End If
1226                     Case 9, 11:
1227                         If Piece = WBISHOP Or Piece = WQUEEN Then
1228                             Exit Do ' still to check other direction
1229                         Else
1230                             CheckLegalNotInCheck = True: Exit Function
1231                         End If
1232                     Case Else
1233                         CheckLegalNotInCheck = True: Exit Function
1234                     End Select
1235                 End If
1236                 Target = Target + Offset: Piece = Board(Target)
1237             Loop
1238
1239             If Piece <> FRAME Then
1240                 '--- possible pinner found. check if there are other piece in direction to king
1241                 Offset = -Offset: Target = mMove.From + Offset: Piece = Board(Target)
1242                 Do While Piece <> FRAME
1243                     If Piece < NO_PIECE Then
1244                         If Piece = BKING Then
1245                             CheckLegalNotInCheck = False: Exit Function
1246                         Else
1247                             CheckLegalNotInCheck = True: Exit Function
1248                         End If
1249                     End If
1250                     Target = Target + Offset: Piece = Board(Target)
1251                 Loop
1252             End If

```

```

1246         CheckLegalNotInCheck = True: Exit Function
1247
1248     End If
1249 Else
1250     If mMove.Piece = WKING Then
1251         If IsAttackedByB(mMove.Target) Then Exit Function
1252     Else
1253         If Not SameXRay(mMove.From, WKingLoc) Then CheckLegalNotInCheck = True: Exit
Function
1254         If SqBetween(mMove.From, WKingLoc, mMove.Target) Then CheckLegalNotInCheck =
True: Exit Function
1255
1256         Offset = DirOffset(WKingLoc, mMove.From): Target = mMove.From + Offset: Piece =
Board(Target)
1257         Do While Piece <> FRAME
1258             If Piece < NO_PIECE Then
1259                 Select Case Abs(Offset)
1260                     Case 1, 10:
1261                         If Piece = BROOK Or Piece = BQUEEN Then
1262                             Exit Do 'still to check other direction
1263                         Else
1264                             CheckLegalNotInCheck = True: Exit Function
1265                         End If
1266                     Case 9, 11:
1267                         If Piece = BBISHOP Or Piece = BQUEEN Then
1268                             Exit Do 'still to check other direction
1269                         Else
1270                             CheckLegalNotInCheck = True: Exit Function
1271                         End If
1272                     Case Else
1273                         CheckLegalNotInCheck = True: Exit Function
1274                     End Select
1275                 End If
1276                 Target = Target + Offset: Piece = Board(Target)
1277             Loop
1278
1279         If Piece <> FRAME Then
1280             '--- possible pinner found. check if there are other piece in direction to king
1281             Offset = -Offset: Target = mMove.From + Offset: Piece = Board(Target)
1282             Do While Piece <> FRAME
1283                 If Piece < NO_PIECE Then
1284                     If Piece = WKING Then
1285                         CheckLegalNotInCheck = False: Exit Function
1286                     Else
1287                         CheckLegalNotInCheck = True: Exit Function
1288                     End If
1289                 End If
1290                 Target = Target + Offset: Piece = Board(Target)
1291             Loop
1292         End If
1293         CheckLegalNotInCheck = True: Exit Function
1294     End If
1295 End If
1296 CheckLegalNotInCheck = CheckLegal(mMove)
1297 End Function
1298
1299 '-----
1300 '- CheckLegal() - Legal move?
1301 '-----
1302 Public Function CheckLegal(mMove As TMOVE) As Boolean
1303     CheckLegal = False
1304     If mMove.From < SQ_A1 Then Exit Function
1305     If mMove.Castle = NO_CASTLE Then
1306         If bWhiteToMove Then
1307             If IsAttackedByW(BKingLoc) Then Exit Function 'BKing mate?
1308         Else
1309             If IsAttackedByB(WKingLoc) Then Exit Function 'WKing mate?
1310         End If

```

```

1311 Else
1312
1313     ' Castling
1314     Select Case mMove.Castle
1315         Case WHITEOO:
1316             If IsAttackedByB(WKING_START) Then Exit Function
1317             If IsAttackedByB(WKING_START + 1) Then Exit Function
1318             If IsAttackedByB(WKING_START + 2) Then Exit Function
1319         Case WHITEOOO:
1320             If IsAttackedByB(WKING_START) Then Exit Function
1321             If IsAttackedByB(WKING_START - 1) Then Exit Function
1322             If IsAttackedByB(WKING_START - 2) Then Exit Function
1323         Case BLACKOO:
1324             If IsAttackedByW(BKING_START) Then Exit Function
1325             If IsAttackedByW(BKING_START + 1) Then Exit Function
1326             If IsAttackedByW(BKING_START + 2) Then Exit Function
1327         Case BLACKOOO:
1328             If IsAttackedByW(BKING_START) Then Exit Function
1329             If IsAttackedByW(BKING_START - 1) Then Exit Function
1330             If IsAttackedByW(BKING_START - 2) Then Exit Function
1331     End Select
1332
1333 End If
1334 CheckLegal = True
1335 End Function
1336
1337 '-----
1338 '- CheckEvasionLegal() - Legal move? in check before
1339 '-----
1340 Public Function CheckEvasionLegal() As Boolean
1341     If bWhiteToMove Then
1342         CheckEvasionLegal = Not IsAttackedByW(BKingLoc) ' Black king mate?
1343     Else
1344         CheckEvasionLegal = Not IsAttackedByB(WKingLoc) ' White king mate?
1345     End If
1346 End Function
1347
1348 '-----
1349 '- IsAttacked() - piece attacked? Also used for checking legal move
1350 '-----
1351 'Public Function IsAttacked(ByVal Location As Long, _
1352 '    ByVal AttackByColor As enumColor) As Boolean
1353 '    If AttackByColor = COL_WHITE Then
1354 '        IsAttacked = IsAttackedByW(Location)
1355 '    Else
1356 '        IsAttacked = IsAttackedByB(Location)
1357 '    End If
1358 'End Function
1359
1360 '-----
1361 '- IsAttackedByW() - square attacked by white ? Also used for checking legal move
1362 '-----
1363 Public Function IsAttackedByW(ByVal Location As Long) As Boolean
1364     Dim i As Long, Target As Long, Offset As Long, Piece As Long
1365     Dim OppQRCnt As Long, OppQBCnt As Long
1366     IsAttackedByW = True
1367     OppQRCnt = PieceCnt(WQUEEN) + PieceCnt(WROOK): OppQBCnt = PieceCnt(WQUEEN) +
        PieceCnt(WBISHOP)
1368
1369     ' vertical+horizontal: Queen, Rook, King
1370     For i = 0 To 3
1371         Offset = DirectionOffset(i): Target = Location + Offset: Piece = Board(Target)
1372         If Piece <> FRAME Then
1373             If Piece = WKING Then Exit Function
1374             If OppQRCnt > 0 Then
1375
1376                 Do While Piece <> FRAME
1377                     If Piece < NO_PIECE Then If Piece = WROOK Or Piece = WQUEEN Then Exit

```

```

1378         Function Else Exit Do
1379             Target = Target + Offset: Piece = Board(Target)
1380         Loop
1381     End If
1382 End If
1383 Next
1384
1385 'diagonal: Queen, Bishop, Pawn, King
1386 For i = 4 To 7
1387     Offset = DirectionOffset(i): Target = Location + Offset: Piece = Board(Target)
1388     If Piece <> FRAME Then
1389         If Piece = WPAWN Then
1390             If ((i = 5) Or (i = 7)) Then Exit Function
1391             ElseIf Piece = WKING Then Exit Function
1392             ElseIf OppQBCnt <> 0 Then
1393
1394                 Do While Piece <> FRAME
1395                     If Piece < NO_PIECE Then If Piece = WBISHOP Or Piece = WQUEEN Then Exit
1396                     Function Else Exit Do
1397                     Target = Target + Offset: Piece = Board(Target)
1398                 Loop
1399             End If
1400         End If
1401     Next
1402
1403     If PieceCnt(WKNIGHT) > 0 Then
1404         For i = 0 To 7
1405             If Board(Location + KnightOffsets(i)) = WKNIGHT Then Exit Function 'Knight
1406         Next
1407     End If
1408     IsAttackedByW = False
1409 End Function
1410
1411 '-----
1412 '- IsAttackedByB() - square attacked by black ? Also used for checking legal move
1413 '-----
1414 Public Function IsAttackedByB(ByVal Location As Long) As Boolean
1415     Dim i As Long, Target As Long, Offset As Long, Piece As Long
1416     Dim OppQRCnt As Long, OppQBCnt As Long
1417     IsAttackedByB = True
1418     OppQRCnt = PieceCnt(BQUEEN) + PieceCnt(BROOK): OppQBCnt = PieceCnt(BQUEEN) +
1419     PieceCnt(BBISHOP)
1420
1421 'vertical+horizontal: Queen, Rook, King
1422 For i = 0 To 3
1423     Offset = DirectionOffset(i): Target = Location + Offset: Piece = Board(Target)
1424     If Piece <> FRAME Then
1425         If Piece = BKING Then Exit Function
1426         If OppQRCnt > 0 Then
1427
1428             Do While Piece <> FRAME
1429                 If Piece < NO_PIECE Then If Piece = BROOK Or Piece = BQUEEN Then Exit
1430                 Function Else Exit Do
1431                 Target = Target + Offset: Piece = Board(Target)
1432             Loop
1433         End If
1434     End If
1435 Next
1436
1437 'diagonal: Queen, Bishop, Pawn, King
1438 For i = 4 To 7
1439     Offset = DirectionOffset(i): Target = Location + Offset: Piece = Board(Target)
1440     If Piece <> FRAME Then
1441         If Piece = BPAWN Then
1442             If ((i = 4) Or (i = 6)) Then Exit Function

```

```

1442     ElseIf Piece = BKING Then Exit Function
1443     ElseIf OppQBCnt <> 0 Then
1444
1445         Do While Piece <> FRAME
1446             If Piece < NO_PIECE Then If Piece = BBISHOP Or Piece = BQUEEN Then Exit
1447             Function Else Exit Do
1448             Target = Target + Offset: Piece = Board(Target)
1449         Loop
1450     End If
1451 End If
1452 Next
1453
1454 If PieceCnt(BKNIGHT) > 0 Then
1455     For i = 0 To 7
1456         If Board(Location + KnightOffsets(i)) = BKNIGHT Then Exit Function 'Knight
1457     Next
1458 End If
1459 IsAttackedByB = False
1460 End Function
1461
1462 Public Sub PlayMove(mMove As TMOVE)
1463     '--- Play move in game
1464     Dim From As Long, Target As Long
1465     Dim EnPassant As Long, Castle As Long, PromoteTo As Long
1466     Dim i As Long
1467
1468     With mMove
1469         From = .From
1470         Target = .Target
1471         EnPassant = .EnPassant
1472         Castle = .Castle
1473         PromoteTo = .Promoted
1474     End With
1475
1476     ' Init EnPassant fields
1477     For i = SQ_A3 To SQ_A6
1478         If (Board(i) = WEP_PIECE) Then Board(i) = NO_PIECE
1479     Next
1480
1481     For i = SQ_A6 To SQ_H6
1482         If (Board(i) = BEP_PIECE) Then Board(i) = NO_PIECE
1483     Next
1484
1485     ' 50 move draw rule
1486     If Board(From) = WPAWN Or Board(From) = BPAWN Or Board(Target) < NO_PIECE Or
1487     PromoteTo <> 0 Then
1488         Fifty = 0
1489     Else
1490         Fifty = Fifty + 1
1491     End If
1492     PliesFromNull = PliesFromNull + 1
1493     bWhiteToMove = Not bWhiteToMove
1494
1495     Select Case Castle
1496     Case NO_CASTLE
1497     Case WHITEOO
1498         Board(Target) = Board(From)
1499         Board(From) = NO_PIECE
1500         Board(SQ_H1) = NO_PIECE
1501         Board(SQ_F1) = WROOK
1502         Moved(Target) = Moved(Target) + 1
1503         Moved(From) = Moved(From) + 1
1504         Moved(SQ_H1) = Moved(SQ_H1) + 1
1505         Moved(SQ_F1) = Moved(SQ_F1) + 1
1506         WhiteCastled = WHITEOO
1507         WKingLoc = Target
1508         InitPieceSquares

```



```

1508     Exit Sub
1509 Case WHITEOOO
1510     Board(Target) = Board(From)
1511     Board(From) = NO_PIECE
1512     Board(SQ_A1) = NO_PIECE
1513     Board(SQ_D1) = WROOK
1514     Moved(Target) = Moved(Target) + 1
1515     Moved(From) = Moved(From) + 1
1516     Moved(SQ_A1) = Moved(SQ_A1) + 1
1517     Moved(SQ_D1) = Moved(SQ_D1) + 1
1518     WhiteCastled = WHITEOOO
1519     WKingLoc = Target
1520     InitPieceSquares
1521     Exit Sub
1522 Case BLACKOO
1523     Board(Target) = Board(From)
1524     Board(From) = NO_PIECE
1525     Board(SQ_H8) = NO_PIECE
1526     Board(SQ_F8) = BROOK
1527     Moved(Target) = Moved(Target) + 1
1528     Moved(From) = Moved(From) + 1
1529     Moved(SQ_H8) = Moved(SQ_H8) + 1
1530     Moved(SQ_F8) = Moved(SQ_F8) + 1
1531     BlackCastled = BLACKOO
1532     BKingLoc = Target
1533     InitPieceSquares
1534     Exit Sub
1535 Case BLACKOOO
1536     Board(Target) = Board(From)
1537     Board(From) = NO_PIECE
1538     Board(SQ_A8) = NO_PIECE
1539     Board(SQ_D8) = BROOK
1540     Moved(Target) = Moved(Target) + 1
1541     Moved(From) = Moved(From) + 1
1542     Moved(SQ_A8) = Moved(SQ_A8) + 1
1543     Moved(SQ_D8) = Moved(SQ_D8) + 1
1544     BlackCastled = BLACKOOO
1545     BKingLoc = Target
1546     InitPieceSquares
1547     Exit Sub
1548 End Select
1549
1550 'en passant
1551 If EnPassant = ENPASSANT_CAPTURE And (Board(From) And 1) <> 0 Then
1552     Board(Target) = Board(From)
1553     Board(From) = NO_PIECE
1554     Board(Target - 10) = NO_PIECE
1555     Moved(Target) = Moved(Target) + 1
1556     Moved(From) = Moved(From) + 1
1557     Moved(Target - 10) = Moved(Target - 10) + 1
1558     InitPieceSquares
1559     Exit Sub
1560 End If
1561 If EnPassant = ENPASSANT_CAPTURE Then
1562     Board(Target) = Board(From)
1563     Board(From) = NO_PIECE
1564     Board(Target + 10) = NO_PIECE
1565     Moved(Target) = Moved(Target) + 1
1566     Moved(From) = Moved(From) + 1
1567     Moved(Target + 10) = Moved(Target + 10) + 1
1568     InitPieceSquares
1569     Exit Sub
1570 End If
1571 If Board(From) = BPAWN And Rank(From) = 7 And Target = From - 20 Then
1572     Board(Target) = Board(From)
1573     Board(From) = NO_PIECE
1574     Board(From - 10) = BEP_PIECE
1575     Moved(Target) = Moved(Target) + 1

```

```

1576     Moved(From) = Moved(From) + 1
1577     InitPieceSquares
1578     Exit Sub
1579 End If
1580 If Board(From) = BPAWN And Board(Target) = WEP_PIECE Then
1581     Board(Target) = Board(From)
1582     Board(From) = NO_PIECE
1583     Board(Target + 10) = NO_PIECE
1584     Moved(Target) = Moved(Target) + 1
1585     Moved(From) = Moved(From) + 1
1586     Moved(Target + 10) = Moved(Target + 10) + 1
1587     InitPieceSquares
1588     Exit Sub
1589 End If
1590 If Board(From) = WPAWN And Rank(From) = 2 And Target = From + 20 Then
1591     Board(Target) = Board(From)
1592     Board(From) = NO_PIECE
1593     Board(From + 10) = WEP_PIECE
1594     Moved(Target) = Moved(Target) + 1
1595     Moved(From) = Moved(From) + 1
1596     InitPieceSquares
1597     Exit Sub
1598 End If
1599 If Board(From) = WPAWN And Board(Target) = BEP_PIECE Then
1600     Board(Target) = Board(From)
1601     Board(From) = NO_PIECE
1602     Board(Target - 10) = NO_PIECE
1603     Moved(Target) = Moved(Target) + 1
1604     Moved(From) = Moved(From) + 1
1605     Moved(Target - 10) = Moved(Target - 10) + 1
1606     InitPieceSquares
1607     Exit Sub
1608 End If
1609 'Promotion
1610 If PromoteTo <> 0 Then
1611     Board(Target) = PromoteTo
1612     Board(From) = NO_PIECE
1613     Moved(Target) = Moved(Target) + 1
1614     Moved(From) = Moved(From) + 1
1615     InitPieceSquares
1616     Exit Sub
1617 End If
1618 'Normal move
1619 If Board(From) = WKING Then
1620     WKingLoc = Target
1621 ElseIf Board(From) = BKING Then
1622     BKingLoc = Target
1623 End If
1624 Board(Target) = Board(From)
1625 Board(From) = NO_PIECE
1626 Moved(Target) = Moved(Target) + 1
1627 Moved(From) = Moved(From) + 1
1628 InitPieceSquares
1629 End Sub
1630
1631 Public Sub MakeMove(mMove As TMOVE)
1632     '--- Do move on board
1633     Dim From As Long, Target As Long
1634     Dim Captured As Long, EnPassant As Long
1635     Dim Promoted As Long, Castle As Long
1636     Dim PieceFrom As Long
1637
1638     With mMove
1639         From = .From: Target = .Target: Captured = .Captured: EnPassant = .EnPassant:
1640         Promoted = .Promoted: Castle = .Castle
1641     End With
1642     PieceFrom = Board(From)

```

```

1643 Board(From) = NO_PIECE: Moved(From) = Moved(From) + 1
1644 mMove.CapturedNumber = Squares(Target)
1645 Pieces(Squares(From)) = Target: Pieces(Squares(Target)) = 0
1646 Squares(Target) = Squares(From): Squares(From) = 0
1647 arFiftyMove(Ply) = Fifty: PliesFromNull = PliesFromNull + 1
1648 If PieceFrom = WPAWN Or PieceFrom = BPAWN Or Board(Target) < NO_PIECE Or Promoted <>
    0 Then Fifty = 0 Else Fifty = Fifty + 1
1649
1650 ' En Passant
1651 EpPosArr(Ply + 1) = 0
1652 If EnPassant <> 0 Then
1653     If EnPassant = ENPASSANT_WMOVE Then
1654         Board(From + 10) = WEP_PIECE
1655         EpPosArr(Ply + 1) = From + 10
1656     ElseIf EnPassant = ENPASSANT_BMOVE Then
1657         Board(From - 10) = BEP_PIECE
1658         EpPosArr(Ply + 1) = From - 10
1659     End If
1660     If EnPassant = ENPASSANT_CAPTURE Then '--- EP capture move
1661         If PieceFrom = WPAWN Then
1662             Board(Target) = PieceFrom
1663             Board(Target - 10) = NO_PIECE: PieceCntMinus BPAWN
1664             mMove.CapturedNumber = Squares(Target - 10)
1665             Pieces(Squares(Target - 10)) = 0: Squares(Target - 10) = 0
1666         ElseIf PieceFrom = BPAWN Then
1667             Board(Target) = PieceFrom
1668             Board(Target + 10) = NO_PIECE: PieceCntMinus WPAWN
1669             mMove.CapturedNumber = Squares(Target + 10)
1670             Pieces(Squares(Target + 10)) = 0: Squares(Target + 10) = 0
1671         End If
1672         GoTo lblExit
1673     End If
1674 End If
1675 'Castle: additional rook move here, King later as normal move
1676 If Castle <> NO_CASTLE Then
1677
1678     Select Case Castle
1679     Case WHITEOO
1680         WhiteCastled = WHITEOO
1681         Board(SQ_H1) = NO_PIECE: Moved(SQ_H1) = Moved(SQ_H1) + 1
1682         Board(SQ_F1) = WROOK: Moved(SQ_F1) = Moved(SQ_F1) + 1
1683         Pieces(Squares(SQ_H1)) = SQ_F1: Squares(SQ_F1) = Squares(SQ_H1): Squares(SQ_H1
        ) = 0
1684         Board(SQ_G1) = WKING: Moved(SQ_G1) = Moved(SQ_G1) + 1: WKingLoc = SQ_G1
1685         GoTo lblExit
1686     Case WHITEOOO
1687         WhiteCastled = WHITEOOO
1688         Board(SQ_A1) = NO_PIECE: Moved(SQ_A1) = Moved(SQ_A1) + 1
1689         Board(SQ_D1) = WROOK: Moved(SQ_D1) = Moved(SQ_D1) + 1
1690         Pieces(Squares(SQ_A1)) = SQ_D1: Squares(SQ_D1) = Squares(SQ_A1): Squares(SQ_A1
        ) = 0
1691         Board(SQ_C1) = WKING: Moved(SQ_C1) = Moved(SQ_C1) + 1: WKingLoc = SQ_C1
1692         GoTo lblExit
1693     Case BLACKOO
1694         BlackCastled = BLACKOO
1695         Board(SQ_H8) = NO_PIECE: Moved(SQ_H8) = Moved(SQ_H8) + 1
1696         Board(SQ_F8) = BROOK: Moved(SQ_F8) = Moved(SQ_F8) + 1
1697         Pieces(Squares(SQ_H8)) = SQ_F8: Squares(SQ_F8) = Squares(SQ_H8): Squares(SQ_H8
        ) = 0
1698         Board(SQ_G8) = BKING: Moved(SQ_G8) = Moved(SQ_G8) + 1: BKingLoc = SQ_G8
1699         GoTo lblExit
1700     Case BLACKOOO
1701         BlackCastled = BLACKOOO
1702         Board(SQ_A8) = NO_PIECE: Moved(SQ_A8) = Moved(SQ_A8) + 1
1703         Board(SQ_D8) = BROOK: Moved(SQ_D8) = Moved(SQ_D8) + 1
1704         Pieces(Squares(SQ_A8)) = SQ_D8: Squares(SQ_D8) = Squares(SQ_A8): Squares(SQ_A8
        ) = 0
1705         Board(SQ_C8) = BKING: Moved(SQ_C8) = Moved(SQ_C8) + 1: BKingLoc = SQ_C8

```

```

1706         GoTo lblExit
1707     End Select
1708
1709 End If
1710 If Promoted <> 0 Then
1711     PieceCntPlus Promoted
1712     Board(Target) = Promoted
1713     PieceCntMinus PieceFrom
1714     Moved(Target) = Moved(Target) + 1
1715 Else
1716
1717     '--- normal move
1718     Select Case PieceFrom
1719         Case WKING: WKingLoc = Target
1720         Case BKING: BKingLoc = Target
1721     End Select
1722
1723     Board(Target) = PieceFrom: Moved(Target) = Moved(Target) + 1
1724 End If
1725 If Captured > 0 Then If Captured < NO_PIECE Then PieceCntMinus Captured
1726 lblExit:
1727     bWhiteToMove = Not bWhiteToMove
1728 End Sub
1729
1730 Public Sub UnmakeMove(mMove As TMOVE)
1731     'take back this move on board
1732     Dim From As Long, Target As Long
1733     Dim Captured As Long, EnPassant As Long, CapturedNumber As Long
1734     Dim Promoted As Long, Castle As Long, PieceTarget As Long
1735
1736     With mMove
1737         From = .From: Target = .Target: Captured = .Captured
1738         EnPassant = .EnPassant: Promoted = .Promoted: Castle = .Castle: CapturedNumber = .
            CapturedNumber
1739     End With
1740
1741     PieceTarget = Board(Target)
1742     Squares(From) = Squares(Target): Squares(Target) = CapturedNumber
1743     Pieces(Squares(Target)) = Target: Pieces(Squares(From)) = From
1744     Fifty = arFiftyMove(Ply)
1745     If Castle <> NO_CASTLE Then
1746
1747         Select Case Castle
1748             Case WHITEOO
1749                 WhiteCastled = NO_CASTLE
1750                 Board(SQ_F1) = NO_PIECE: Moved(SQ_F1) = Moved(SQ_F1) - 1
1751                 Board(SQ_H1) = WROOK: Moved(SQ_H1) = Moved(SQ_H1) - 1
1752                 Squares(SQ_H1) = Squares(SQ_F1): Squares(SQ_F1) = 0: Pieces(Squares(SQ_H1)) =
                    SQ_H1
1753                 Board(SQ_E1) = WKING: Moved(SQ_E1) = Moved(SQ_E1) - 1: WKingLoc = SQ_E1
1754                 Board(SQ_G1) = NO_PIECE: Moved(SQ_G1) = Moved(SQ_G1) - 1
1755                 GoTo lblExit
1756             Case WHITEOOO
1757                 WhiteCastled = NO_CASTLE
1758                 Board(SQ_D1) = NO_PIECE: Moved(SQ_D1) = Moved(SQ_D1) - 1
1759                 Board(SQ_A1) = WROOK: Moved(SQ_A1) = Moved(SQ_A1) - 1
1760                 Squares(SQ_A1) = Squares(SQ_D1): Squares(SQ_D1) = 0: Pieces(Squares(SQ_A1)) =
                    SQ_A1
1761                 Board(SQ_E1) = WKING: Moved(SQ_E1) = Moved(SQ_E1) - 1: WKingLoc = SQ_E1
1762                 Board(SQ_C1) = NO_PIECE: Moved(SQ_C1) = Moved(SQ_C1) - 1
1763                 GoTo lblExit
1764             Case BLACKOO
1765                 BlackCastled = NO_CASTLE
1766                 Board(SQ_F8) = NO_PIECE: Moved(SQ_F8) = Moved(SQ_F8) - 1
1767                 Board(SQ_H8) = BROOK: Moved(SQ_H8) = Moved(SQ_H8) - 1
1768                 Squares(SQ_H8) = Squares(SQ_F8): Squares(SQ_F8) = 0: Pieces(Squares(SQ_H8)) =
                    SQ_H8
1769                 Board(SQ_E8) = BKING: Moved(SQ_E8) = Moved(SQ_E8) - 1: BKingLoc = SQ_E8

```

```

1770         Board(SQ_G8) = NO_PIECE: Moved(SQ_G8) = Moved(SQ_G8) - 1
1771         GoTo lblExit
1772     Case BLACK000
1773         BlackCastled = NO_CASTLE
1774         Board(SQ_D8) = NO_PIECE: Moved(SQ_D8) = Moved(SQ_D8) - 1
1775         Board(SQ_A8) = BROOK: Moved(SQ_A8) = Moved(SQ_A8) - 1
1776         Squares(SQ_A8) = Squares(SQ_D8): Squares(SQ_D8) = 0: Pieces(Squares(SQ_A8)) =
            SQ_A8
1777         Board(SQ_E8) = BKING: Moved(SQ_E8) = Moved(SQ_E8) - 1: BKingLoc = SQ_E8
1778         Board(SQ_C8) = NO_PIECE: Moved(SQ_C8) = Moved(SQ_C8) - 1
1779         GoTo lblExit
1780     End Select
1781
1782 End If
1783 If EnPassant <> 0 Then
1784     If EnPassant = ENPASSANT_WMOVE Then
1785         Board(From + 10) = NO_PIECE
1786     ElseIf EnPassant = ENPASSANT_BMOVE Then
1787         Board(From - 10) = NO_PIECE
1788     End If
1789     If EnPassant = ENPASSANT_CAPTURE Then
1790         If PieceTarget = WPAWN Then
1791             Board(From) = PieceTarget
1792             Board(Target) = NO_PIECE
1793             Board(Target - 10) = BPAWN: PieceCntPlus BPAWN
1794             Squares(Target - 10) = CapturedNumber
1795             Pieces(CapturedNumber) = Target - 10
1796             Squares(Target) = 0
1797         ElseIf PieceTarget = BPAWN Then
1798             Board(From) = PieceTarget
1799             Board(Target) = NO_PIECE
1800             Board(Target + 10) = WPAWN: PieceCntPlus WPAWN
1801             Squares(Target + 10) = CapturedNumber
1802             Pieces(CapturedNumber) = Target + 10
1803             Squares(Target) = 0
1804         End If
1805         Moved(From) = Moved(From) - 1
1806         GoTo lblExit
1807     End If
1808 End If
1809 If Promoted <> 0 Then
1810     If (Promoted And 1) = WCOL Then
1811         Board(From) = WPAWN: PieceCntPlus WPAWN
1812         PieceCntMinus Board(Target)
1813         Board(Target) = Captured
1814         Moved(From) = Moved(From) - 1
1815         Moved(Target) = Moved(Target) - 1
1816     Else
1817         Board(From) = BPAWN: PieceCntPlus BPAWN
1818         PieceCntMinus Board(Target)
1819         Board(Target) = Captured
1820         Moved(From) = Moved(From) - 1
1821         Moved(Target) = Moved(Target) - 1
1822     End If
1823 Else
1824
1825     '--- normal move
1826     Select Case PieceTarget
1827     Case WKING: WKingLoc = From
1828     Case BKING: BKingLoc = From
1829     End Select
1830
1831     Board(From) = PieceTarget: Moved(From) = Moved(From) - 1
1832     Board(Target) = Captured: Moved(Target) = Moved(Target) - 1
1833 End If
1834 If Captured > 0 Then If Captured < NO_PIECE Then PieceCntPlus Captured
1835 lblExit:
1836 PliesFromNull = PliesFromNull - 1

```

```

1837
1838     bWhiteToMove = Not bWhiteToMove ' switch side to move
1839
1840 End Sub
1841
1842 '-----
1843 ' InitPieceSquares: Init tables for pieces and squares
1844 ' Squares(board location) points to piece in Pieces() list
1845 ' Pieces(piece num) points to board location
1846 '-----
1847 Public Sub InitPieceSquares()
1848     Dim i As Long, PT As Long
1849     NumPieces = 0
1850     Pieces(0) = 0
1851     Erase PieceCnt()
1852     Erase Squares()
1853     Erase Pieces()
1854     WNonPawnPieces = 0: BNonPawnPieces = 0
1855     '--- White ---
1856     WhitePiecesStart = 1
1857
1858     For PT = PT_PAWN To PT_KING ' sort by piece type
1859         For i = SQ_A1 To SQ_H8
1860             If (Board(i) <> FRAME And Board(i) < NO_PIECE And (Board(i) And 1) = WCOL) And
1861                 PieceType(Board(i)) = PT Then
1862                 NumPieces = NumPieces + 1: Pieces(NumPieces) = i: Squares(i) = NumPieces
1863                 PieceCntPlus Board(i)
1864
1865                 Select Case Board(i)
1866                     Case WKING: WKingLoc = i
1867                 End Select
1868             End If
1869         Next
1870     Next
1871
1872     WhitePiecesEnd = NumPieces
1873     '--- Black ---
1874     BlackPiecesStart = NumPieces + 1
1875
1876     For PT = PT_PAWN To PT_KING
1877         For i = SQ_A1 To SQ_H8
1878             If (Board(i) <> FRAME And Board(i) < NO_PIECE And (Board(i) And 1) = BCOL) And
1879                 PieceType(Board(i)) = PT Then
1880                 NumPieces = NumPieces + 1: Pieces(NumPieces) = i: Squares(i) = NumPieces
1881                 PieceCntPlus Board(i)
1882
1883                 Select Case Board(i)
1884                     Case BKING: BKingLoc = i
1885                 End Select
1886             End If
1887         Next
1888     Next
1889
1890     BlackPiecesEnd = NumPieces
1891     ResetMaterial
1892 End Sub
1893
1894 Public Sub PieceCntPlus(ByVal Piece As Long)
1895     If Piece > FRAME And Piece < NO_PIECE Then
1896         PieceCnt(Piece) = PieceCnt(Piece) + 1
1897         If Piece > BPAWN And Piece < WKING Then ' King not counted
1898             If CBool(Piece And 1) Then WNonPawnPieces = WNonPawnPieces + 1 Else
1899                 BNonPawnPieces = BNonPawnPieces + 1
1900             End If
1901         End If
1902     End Sub

```

```

1902
1903 Public Sub PieceCntMinus(ByVal Piece As Long)
1904     If Piece > FRAME And Piece < NO_PIECE Then
1905         PieceCnt(Piece) = PieceCnt(Piece) - 1
1906     If Piece > BPAWN And Piece < WKING Then
1907         If CBool(Piece And 1) Then WNonPawnPieces = WNonPawnPieces - 1 Else
1908             BNonPawnPieces = BNonPawnPieces - 1
1909     End If
1910 End If
1911 Debug.Assert PieceCnt(Piece) >= 0
1912 End Sub
1913
1914 '-----
1915 'InCheck() Color to move in check?
1916 '-----
1917 Public Function InCheck() As Boolean
1918     If bWhiteToMove Then
1919         InCheck = IsAttackedByB(WKingLoc)
1920     Else
1921         InCheck = IsAttackedByW(BKingLoc)
1922     End If
1923 End Function
1924
1925 'Public Function OppInCheck() As Boolean
1926 ' If Not bWhiteToMove Then
1927 '     OppInCheck = IsAttackedByB(WKingLoc)
1928 ' Else
1929 '     OppInCheck = IsAttackedByW(BKingLoc)
1930 ' End If
1931 'End Function
1932
1933 Public Function LocCoord(ByVal Square As Long) As String
1934     LocCoord = UCase$(Chr$(File(Square) + 96) & Rank(Square))
1935 End Function
1936
1937 '-----
1938 ' Board File character to number A => 1
1939 '-----
1940 Public Function FileRev(ByVal sFile As String) As Long
1941     FileRev = Asc(LCase$(sFile)) - 96
1942 End Function
1943
1944 '-----
1945 'RankRev() - Board Rank number to square number Rank 2 = 30
1946 '-----
1947 Public Function RankRev(ByVal sRank As String) As Long
1948     RankRev = (Val(sRank) + 1) * 10
1949 End Function
1950
1951 Public Function RelativeRank(ByVal Col As enumColor, ByVal sq As Long) As Long
1952     If Col = COL_WHITE Then
1953         RelativeRank = Rank(sq)
1954     Else
1955         RelativeRank = (9 - Rank(sq))
1956     End If
1957 End Function
1958
1959 '-----
1960 'CompToCoord(): Convert internal move to text output
1961 '-----
1962 Public Function CompToCoord(CompMove As TMOVE) As String
1963     Dim sCoordMove As String
1964     If CompMove.From = 0 Then CompToCoord = "": Exit Function
1965     sCoordMove = Chr$(File(CompMove.From) + 96) & Rank(CompMove.From) & Chr$(File(
1966         CompMove.Target) + 96) & Rank(CompMove.Target)
1967     If CompMove.Promoted <> 0 Then
1968         Select Case CompMove.Promoted

```

```

1968         Case WKNIGHT, BKNIGHT
1969             sCoordMove = sCoordMove & "n"
1970         Case WROOK, BROOK
1971             sCoordMove = sCoordMove & "r"
1972         Case WBISHOP, BBISHOP
1973             sCoordMove = sCoordMove & "b"
1974         Case WQUEEN, BQUEEN
1975             sCoordMove = sCoordMove & "q"
1976     End Select
1977
1978 End If
1979 CompToCoord = sCoordMove
1980 End Function
1981
1982 Public Function TextToMove(ByVal sMoveText As String) As TMOVE
1983     'format "b7b8q"
1984     TextToMove = EmptyMove
1985     sMoveText = Trim(Replace(sMoveText, "-", ""))
1986     TextToMove.From = CoordToLoc(Left$(sMoveText, 2))
1987     TextToMove.Piece = Board(TextToMove.From)
1988     TextToMove.Target = CoordToLoc(Mid$(sMoveText, 3, 2))
1989     TextToMove.Captured = Board(TextToMove.Target)
1990
1991     Select Case LCase(Mid$(sMoveText, 5, 1))
1992     Case "q":
1993         If PieceColor(TextToMove.Piece) = COL_WHITE Then TextToMove.Promoted = WQUEEN
1994         Else TextToMove.Promoted = BQUEEN
1995     Case "r":
1996         If PieceColor(TextToMove.Piece) = COL_WHITE Then TextToMove.Promoted = WROOK
1997         Else TextToMove.Promoted = BROOK
1998     Case "b":
1999         If PieceColor(TextToMove.Piece) = COL_WHITE Then TextToMove.Promoted = WBISHOP
2000         Else TextToMove.Promoted = BBISHOP
2001     Case "n":
2002         If PieceColor(TextToMove.Piece) = COL_WHITE Then TextToMove.Promoted = WKNIGHT
2003         Else TextToMove.Promoted = BKNIGHT
2004     Case Else
2005         TextToMove.Promoted = 0
2006     End Select
2007 End Function
2008
2009 Public Sub RemoveEpPiece()
2010     ' Remove EP from Previous Move
2011     If EpPosArr(Ply) > 0 Then Board(EpPosArr(Ply)) = NO_PIECE
2012 End Sub
2013
2014 Public Sub ResetEpPiece()
2015     ' Reset EP from Previous Move
2016     If EpPosArr(Ply) > 0 Then
2017         Select Case Rank(EpPosArr(Ply))
2018         Case 3
2019             Board(EpPosArr(Ply)) = WEP_PIECE
2020         Case 6
2021             Board(EpPosArr(Ply)) = BEP_PIECE
2022         End Select
2023     End If
2024 End Sub
2025
2026 Public Sub CleanEpPieces()
2027     Dim i As Long
2028
2029     For i = SQ_A1 To SQ_H8
2030         If Board(i) = WEP_PIECE Or Board(i) = BEP_PIECE Then Board(i) = NO_PIECE
2031     Next
2032 End Sub

```



```

2032
2033 'Public Function Alpha2Piece(ByVal sPiece As String, ByVal bWhiteToMove As Boolean) As Long
2034 ' Dim a As Long
2035 '
2036 ' Select Case LCase(sPiece)
2037 '   Case "n"
2038 '     a = WKNIGHT
2039 '   Case "b"
2040 '     a = WBISHOP
2041 '   Case "r"
2042 '     a = WROOK
2043 '   Case "q"
2044 '     a = WQUEEN
2045 ' End Select
2046 '
2047 ' If a > 0 And Not bWhiteToMove Then a = a + 1
2048 ' Alpha2Piece = a
2049 'End Function
2050
2051 Public Function Piece2Alpha(ByVal iPiece As Long) As String
2052
2053     Select Case iPiece
2054     Case WPAWN
2055         Piece2Alpha = "P"
2056     Case BPAWN
2057         Piece2Alpha = "p"
2058     Case WKNIGHT
2059         Piece2Alpha = "N"
2060     Case BKNIGHT
2061         Piece2Alpha = "n"
2062     Case WBISHOP
2063         Piece2Alpha = "B"
2064     Case BBISHOP
2065         Piece2Alpha = "b"
2066     Case WROOK
2067         Piece2Alpha = "R"
2068     Case BROOK
2069         Piece2Alpha = "r"
2070     Case WQUEEN
2071         Piece2Alpha = "Q"
2072     Case BQUEEN
2073         Piece2Alpha = "q"
2074     Case WKING
2075         Piece2Alpha = "K"
2076     Case BKING
2077         Piece2Alpha = "k"
2078     Case Else
2079         Piece2Alpha = "."
2080     End Select
2081
2082 End Function
2083
2084 '-----
2085 'PrintPos() - board position in ASCII table
2086 '-----
2087 Public Function PrintPos() As String
2088     Dim a As Long, b As Long, c As Long
2089     Dim sBoard As String
2090     sBoard = vbCrLf
2091     If True Then ' Not bComplsWhite Then 'punto di vista del B (engine e' N)
2092         sBoard = sBoard & " -----" & vbCrLf
2093         For a = 1 To 8
2094             sBoard = sBoard & (9 - a) & "| "
2095
2096             For b = 1 To 8
2097                 c = 100 - (a * 10) + b
2098                 sBoard = sBoard & Piece2Alpha(Board(c)) & " "
2099             Next

```

```

2100         sBoard = sBoard & "|" & vbCrLf
2101     Next
2102
2103 Else
2104
2105     For a = 1 To 8
2106         sBoard = sBoard & a & vbTab
2107
2108         For b = 1 To 8
2109             c = 10 + (a * 10) - b
2110             sBoard = sBoard & Piece2Alpha(Board(c)) & " "
2111         Next
2112
2113         sBoard = sBoard & vbCrLf
2114     Next
2115
2116 End If
2117 sBoard = sBoard & " -----" & vbCrLf
2118 sBoard = sBoard & " " & vbTab & " A B C D E F G H" & vbCrLf
2119 PrintPos = sBoard
2120 End Function
2121
2122 Public Function MoveText(CompMove As TMOVE) As String
2123     ' Returns move string for data type TMove
2124     ' Sample: CompMove.from= 22: CompMove.target=24: MsgBox CompMove > "a2a4"
2125     Dim sCoordMove As String
2126     If CompMove.From = 0 Then MoveText = "": Exit Function
2127     sCoordMove = Chr$(File(CompMove.From) + 96) & Rank(CompMove.From)
2128     If CompMove.Captured <> NO_PIECE And CompMove.Captured > 0 Then sCoordMove =
2129         sCoordMove & "x"
2130     sCoordMove = sCoordMove & Chr$(File(CompMove.Target) + 96) & Rank(CompMove.Target)
2131     If CompMove.IsChecking Then sCoordMove = sCoordMove & "+"
2132     If CompMove.Promoted <> 0 Then
2133
2134         Select Case CompMove.Promoted
2135             Case WKNIGHT, BKNIGHT
2136                 sCoordMove = sCoordMove & "n"
2137             Case WROOK, BROOK
2138                 sCoordMove = sCoordMove & "r"
2139             Case WBISHOP, BBISHOP
2140                 sCoordMove = sCoordMove & "b"
2141             Case WQUEEN, BQUEEN
2142                 sCoordMove = sCoordMove & "q"
2143         End Select
2144
2145     End If
2146     MoveText = sCoordMove
2147 End Function
2148
2149 Public Function GUIMoveText(CompMove As TMOVE) As String
2150     If UCIMode Or bWbPvInUciFormat Then
2151         GUIMoveText = UCIMoveText(CompMove)
2152     Else
2153         GUIMoveText = MoveText(CompMove)
2154     End If
2155 End Function
2156
2157 Public Function UCIMoveText(CompMove As TMOVE) As String
2158     ' UCI: no x for capture or + for check
2159     ' Returns move string for data type TMove
2160     ' Sample: CompMove.from= 22: CompMove.target=24: MsgBox CompMove > "a2a4"
2161     Dim sCoordMove As String
2162     If CompMove.From = 0 Then UCIMoveText = "": Exit Function
2163     sCoordMove = Chr$(File(CompMove.From) + 96) & Rank(CompMove.From)
2164     sCoordMove = sCoordMove & Chr$(File(CompMove.Target) + 96) & Rank(CompMove.Target)
2165     If CompMove.Promoted <> 0 Then

```

```

2167     Select Case CompMove.Promoted
2168     Case WKNIGHT, BKNIGHT
2169         sCoordMove = sCoordMove & "n"
2170     Case WROOK, BROOK
2171         sCoordMove = sCoordMove & "r"
2172     Case WBISHOP, BBISHOP
2173         sCoordMove = sCoordMove & "b"
2174     Case WQUEEN, BQUEEN
2175         sCoordMove = sCoordMove & "q"
2176     End Select
2177
2178 End If
2179 UCIMoveText = sCoordMove
2180 End Function
2181
2182 Public Function PSQT64(pDestW() As TScore, pDestB() As TScore, ParamArray pSrc())
2183     ' Read piece square table as parameter list into array
2184     ' SF tables are symmetric so file A-D is flipped to E-F
2185     Dim i As Long, sq As Long, x As Long, y As Long, x2 As Long, y2 As Long, MG As Long,
2186         EG As Long
2187     Erase pDestW(): Erase pDestB()
2188
2189     ' Source table is for file A-D, rank 1-8 > Flip for E-F
2190     For i = 0 To 31
2191         MG = pSrc(i * 2): EG = pSrc(i * 2 + 1)
2192         ' White
2193         x = i Mod 4: y = i \ 4: sq = 21 + x + y * 10
2194         pDestW(sq).MG = MG: pDestW(sq).EG = EG
2195         ' Debug.Print x, y, sq, pDestW(sq).MG, pDestW(sq).EG
2196         ' flip to E-F
2197         x2 = 7 - x: y2 = y: sq = 21 + x2 + y2 * 10
2198         pDestW(sq).MG = MG: pDestW(sq).EG = EG
2199         ' Debug.Print x2, y2, sq, pDestW(sq).MG, pDestW(sq).EG
2200         ' Black
2201         x2 = x: y2 = 7 - y: sq = 21 + x2 + y2 * 10
2202         pDestB(sq).MG = MG: pDestB(sq).EG = EG
2203         ' Debug.Print x2, y2, sq, pDestB(sq).MG, pDestB(sq).EG
2204         x2 = 7 - x: y2 = 7 - y: sq = 21 + x2 + y2 * 10
2205         pDestB(sq).MG = MG: pDestB(sq).EG = EG
2206         ' Debug.Print x2, y2, sq, pDestB(sq).MG, pDestB(sq).EG
2207     Next
2208 End Function
2209
2210 Public Sub InitRankFile()
2211     Dim i As Long
2212
2213     For i = 1 To MAX_BOARD
2214         Rank(i) = (i \ 10) - 1
2215         RankB(i) = 9 - Rank(i)
2216         File(i) = i Mod 10
2217         RelativeSq(COL_WHITE, i) = i
2218         RelativeSq(COL_BLACK, i) = SQ_A1 - 1 + File(i) + (8 - Rank(i)) * 10
2219     Next
2220
2221 End Sub
2222
2223 '-----
2224 ' AttackedCnt() - ROOK+QUEEN , BISHOP+QUEEN added
2225 ' AttackedCnt attacks + DEFENDER
2226 '-----
2227 'Public Function AttackedCnt(ByVal Location As Long, ByVal Color As enumColor) As Long
2228 ' Dim i As Long, Target As Long
2229 ' AttackedCnt = 0
2230 '
2231 ' ' Orthogonal = index 0-3
2232 ' For i = 0 To 3
2233 '     Target = Location + DirectionOffset(i)

```

```

2234 ' If Color = COL_BLACK Then
2235 '   If Board(Target) = BKING Then
2236 '     AttackedCnt = AttackedCnt + 1
2237 '   Else
2238 '
2239 '     Do While Board(Target) <> FRAME
2240 '       If Board(Target) = BROOK Or Board(Target) = BQUEEN Then
2241 '         AttackedCnt = AttackedCnt + 1
2242 '       ElseIf Board(Target) = WROOK Or Board(Target) = WQUEEN Then
2243 '         AttackedCnt = AttackedCnt - 1
2244 '       ElseIf Board(Target) < NO_PIECE Then ' other pieces
2245 '         Exit Do
2246 '       End If
2247 '       Target = Target + DirectionOffset(i)
2248 '     Loop
2249 '
2250 '   End If
2251 ' Else
2252 '   If Board(Target) = WKING Then
2253 '     AttackedCnt = AttackedCnt + 1
2254 '   Else
2255 '
2256 '     Do While Board(Target) <> FRAME
2257 '       If Board(Target) = WROOK Or Board(Target) = WQUEEN Then
2258 '         AttackedCnt = AttackedCnt + 1
2259 '       ElseIf Board(Target) = BROOK Or Board(Target) = BQUEEN Then
2260 '         AttackedCnt = AttackedCnt - 1
2261 '       ElseIf Board(Target) < NO_PIECE Then ' other pieces
2262 '         Exit Do
2263 '       End If
2264 '       Target = Target + DirectionOffset(i)
2265 '     Loop
2266 '
2267 '   End If
2268 ' End If
2269 ' Next
2270 '
2271 ' ' Diagonal = index 4 to 7
2272 ' For i = 4 To 7
2273 '   Target = Location + DirectionOffset(i)
2274 '   If Color = COL_BLACK Then
2275 '     If Board(Target) = BKING Then
2276 '       AttackedCnt = AttackedCnt + 1
2277 '     Else
2278 '       If Board(Target) = BPAWN And ((i = 4) Or (i = 6)) Then
2279 '         AttackedCnt = AttackedCnt + 1
2280 '       Target = Location + DirectionOffset(i)
2281 '     End If
2282 '
2283 '     Do While Board(Target) <> FRAME
2284 '       If Board(Target) = BBISHOP Or Board(Target) = BQUEEN Then
2285 '         AttackedCnt = AttackedCnt + 1
2286 '       ElseIf Board(Target) = WBISHOP Or Board(Target) = WQUEEN Then
2287 '         AttackedCnt = AttackedCnt - 1
2288 '       ElseIf Board(Target) < NO_PIECE Then
2289 '         Exit Do
2290 '       End If
2291 '       Target = Target + DirectionOffset(i)
2292 '     Loop
2293 '
2294 '   End If
2295 ' Else
2296 '   If Board(Target) = WKING Then
2297 '     AttackedCnt = AttackedCnt + 1
2298 '   Else
2299 '     If Board(Target) = WPAWN And ((i = 5) Or (i = 7)) Then
2300 '       AttackedCnt = AttackedCnt + 1
2301 '     Target = Location + DirectionOffset(i)

```

```

2302 ' End If
2303 '
2304 ' Do While Board(Target) <> FRAME
2305 ' If Board(Target) = WBISHOP Or Board(Target) = WQUEEN Then
2306 '     AttackedCnt = AttackedCnt + 1
2307 ' Elseif Board(Target) = BBISHOP Or Board(Target) = BQUEEN Then
2308 '     AttackedCnt = AttackedCnt - 1
2309 ' Elseif Board(Target) < NO_PIECE Then
2310 '     Exit Do
2311 ' End If
2312 ' Target = Target + DirectionOffset(i)
2313 ' Loop
2314 '
2315 ' End If
2316 ' End If
2317 ' Next
2318 '
2319 ' ' Knight moves
2320 ' For i = 0 To 7
2321 '     Target = Location + KnightOffsets(i)
2322 '     If Color = COL_BLACK Then
2323 '         If Board(Target) = BKNIGHT Then AttackedCnt = AttackedCnt + 1
2324 '         If Board(Target) = WKNIGHT Then AttackedCnt = AttackedCnt - 1
2325 '     Else
2326 '         If Board(Target) = WKNIGHT Then AttackedCnt = AttackedCnt + 1
2327 '         If Board(Target) = BKNIGHT Then AttackedCnt = AttackedCnt - 1
2328 '     End If
2329 ' Next
2330 '
2331 'End Function
2332
2333 Public Sub InitMaxDistance()
2334     ' Max distance x or y
2335     Dim i As Long, j As Long
2336     Dim d As Long, v As Long
2337
2338     For i = SQ_A1 To SQ_H8
2339         For j = SQ_A1 To SQ_H8
2340             v = Abs(Rank(i) - Rank(j))
2341             d = Abs(File(i) - File(j))
2342             If d > v Then v = d
2343             MaxDistance(i, j) = v
2344         Next j
2345     Next i
2346
2347 End Sub
2348
2349 Public Sub InitSqBetween()
2350     ' InitSqBetween(sq,Sq1,Sq2) : sq between sq1 and sq2
2351     Dim i As Long, dir1 As Long, Dir2 As Long, sq As Long, sq1 As Long, sq2 As Long
2352
2353     For sq = SQ_A1 To SQ_H8
2354         If File(sq) >= 1 And File(sq) <= 8 And Rank(sq) >= 1 And Rank(sq) <= 8 Then
2355
2356             For i = 0 To 7
2357                 dir1 = DirectionOffset(i)
2358                 Dir2 = OppositeDir(dir1)
2359                 sq1 = sq + dir1
2360
2361                 Do While File(sq1) >= 1 And File(sq1) <= 8 And Rank(sq1) >= 1 And Rank(sq1) <=
2362                     8
2363                     sq2 = sq + Dir2
2364
2365                     Do While File(sq2) >= 1 And File(sq2) <= 8 And Rank(sq2) >= 1 And Rank(sq2)
2366                         <= 8
2367                         SqBetween(sq, sq1, sq2) = True
2368                         sq2 = sq2 + Dir2
2369                     Loop

```

```

2368
2369         sql = sql + dir1
2370     Loop
2371
2372     Next i
2373
2374 End If
2375 Next sq
2376
2377 End Sub
2378
2379 'Public Function TotalPieceValue() As Long
2380 ' Dim i As Long
2381 ' TotalPieceValue = 0
2382 '
2383 ' For i = 1 To NumPieces
2384 '     TotalPieceValue = TotalPieceValue + PieceAbsValue(Board(Pieces(i)))
2385 ' Next
2386 '
2387 'End Function
2388
2389 Public Function ResetMaterial() As Long
2390     Dim i As Long
2391     ResetMaterial = 0
2392
2393     For i = 1 To NumPieces
2394         Material = Material + PieceScore(Board(Pieces(i)))
2395     Next
2396
2397 End Function
2398
2399 Public Sub FillKingCheckW()
2400     '--- Fill special board to speed up detection of checking moves in OrderMoves
2401     '--- direction to white king is set for queen directions and knights
2402     Dim i As Long, Target As Long, Offset As Long
2403     Erase KingCheckW()
2404
2405     For i = 0 To 7
2406         Offset = DirectionOffset(i): Target = WKingLoc + Offset
2407
2408         Do While Board(Target) <> FRAME ' - not color critical: Opp piece can be captured, own piece can
            move away
2409             KingCheckW(Target) = Offset: If Board(Target) < NO_PIECE Then Exit Do Else
                Target = Target + Offset
2410         Loop
2411
2412         Target = WKingLoc + KnightOffsets(i): If Board(Target) <> FRAME Then KingCheckW(
            Target) = KnightOffsets(i)
2413     Next
2414
2415 End Sub
2416
2417 Public Sub FillKingCheckB()
2418     '--- Fill special board to speed up detection of checking moves in OrderMoves
2419     '--- direction to black king is set for queen directions and knights
2420     Dim i As Long, Target As Long, Offset As Long
2421     Erase KingCheckB()
2422
2423     For i = 0 To 7
2424         Offset = DirectionOffset(i): Target = BKingLoc + Offset
2425
2426         Do While Board(Target) <> FRAME
2427             KingCheckB(Target) = Offset: If Board(Target) < NO_PIECE Then Exit Do Else
                Target = Target + Offset
2428         Loop
2429
2430         Target = BKingLoc + KnightOffsets(i): If Board(Target) <> FRAME Then KingCheckB(
            Target) = KnightOffsets(i)

```

```

2431     Next
2432
2433 End Sub
2434
2435 Public Function IsBlockingMove(ThreatM As TMOVE, BlockM As TMOVE) As Boolean
2436     ' BlockM blocks TreatM ?
2437     IsBlockingMove = False
2438     If MaxDistance(ThreatM.From, ThreatM.Target) <= 1 Then Exit Function
2439     If ThreatM.Piece = WKNIGHT Or ThreatM.Piece = BKNIGHT Then Exit Function
2440     If BlockM.Piece = WKING Or BlockM.Piece = BKING Then Exit Function
2441     If SqBetween(BlockM.Target, ThreatM.From, ThreatM.Target) Then IsBlockingMove = True
2442 End Function
2443
2444 'Public Function SeeSign(Move As TMOVE) As Long
2445 ' SeeSign = 0
2446 ' ' Early return if SEE cannot be negative because captured piece value
2447 ' ' is not less then capturing one. Note that king moves always return
2448 ' ' here
2449 ' If Move.Castle > 0 Or Move.Target = 0 Or Move.Piece = NO_PIECE Or Board(Move.Target) = FRAME Then Exit
Function
2450 ' If PieceType(Move.Piece) = PT_KING Then SeeSign = VALUE_KNOWN_WIN: Exit Function ' King move always
good because legal checked before
2451 ' If Move.SeeValue = VALUE_NONE Then
2452 ' If PieceAbsValue(Move.Piece) + MAX_SEE_DIFF <= PieceAbsValue(Move.Captured) Then SeeSign =
VALUE_KNOWN_WIN: Exit Function ' winning or equal move
2453 ' ' Calculate exchange score
2454 ' Move.SeeValue = GetSEE(Move) ' Returned for future use
2455 ' End If
2456 ' SeeSign = Move.SeeValue
2457 'End Function
2458 '
2459 'Public Function BadSEEMove(Move As TMOVE) As Boolean
2460 ' BadSEEMove = False
2461 ' If Move.Castle > 0 Or Move.Target = 0 Or Move.Piece = NO_PIECE Or Board(Move.Target) = FRAME Then Exit
Function
2462 ' If PieceType(Move.Piece) = PT_KING Then Exit Function ' King move always good because legal checked before
2463 ' If Move.SeeValue = VALUE_NONE Then
2464 ' If PieceAbsValue(Move.Piece) + MAX_SEE_DIFF <= PieceAbsValue(Move.Captured) Then Exit Function '
winning or equal move
2465 ' Move.SeeValue = GetSEE(Move) ' Returned for future use
2466 ' End If
2467 ' BadSEEMove = (Move.SeeValue < -MAX_SEE_DIFF)
2468 'End Function
2469
2470 'Public Function GoodSEEMove(Move As TMOVE) As Boolean
2471 ' GoodSEEMove = True
2472 ' If Move.Castle > 0 Or Move.Target = 0 Or Move.Piece = NO_PIECE Or Board(Move.Target) = FRAME Then Exit
Function
2473 ' If PieceType(Move.Piece) = PT_KING Then Exit Function ' King move always good because legal checked before
2474 ' If Move.SeeValue = VALUE_NONE Then
2475 ' If PieceAbsValue(Move.Piece) + MAX_SEE_DIFF <= PieceAbsValue(Move.Captured) Then Exit Function '
winning or equal move
2476 ' Move.SeeValue = GetSEE(Move) ' Returned for future use
2477 ' End If
2478 ' GoodSEEMove = (Move.SeeValue >= -MAX_SEE_DIFF)
2479 'End Function
2480
2481 Public Function SEEGreaterOrEqual(Move As TMOVE, ByVal Score As Long) As Boolean
2482     '--- Optimized call of Static Exchange Evaluation (SEE): True if SEE greater or equal given Score
2483     SEEGreaterOrEqual = True
2484     If Move.Castle > 0 Or Move.Target = 0 Or Move.Piece = NO_PIECE Or Board(Move.Target)
= FRAME Then Exit Function
2485     If PieceAbsValue(Move.Captured) < Score Then SEEGreaterOrEqual = False: Exit
Function ' only for positice 'score' values
2486     If PieceType(Move.Piece) = PT_KING Then Exit Function ' King move always good because
legal checked before
2487     If Move.SeeValue = VALUE_NONE Then
2488         If PieceAbsValue(Move.Captured) - PieceAbsValue(Move.Piece) >= Score -

```

```

MAX_SEE_DIFF Then Exit Function 'winning or equal move
2489 Move.SeeValue = GetSEE(Move) 'Returned for future use
2490 End If
2491 SEEGreaterOrEqual = (Move.SeeValue >= Score - MAX_SEE_DIFF) 'MAX_SEE_DIFF to handle
bishop equal knight
2492 End Function
2493
2494 Public Function GetSEE(Move As TMOVE) As Long
2495 'Returns piece score win for AttackColor ( positive for white or black).
2496 Dim i As Long, From As Long, MoveTo As Long, Target As Long
2497 Dim CapturedVal As Long, PieceMoved As Boolean
2498 Dim SideToMove As enumColor, SideNotToMove As enumColor
2499 Dim NumAttackers(2) As Long, CurrSgn As Long, MinValIndex As Long, Piece As Long,
Offset As Long
2500 '----
2501 GetSEE = 0
2502 If PieceType(Move.Piece) = PT_KING Then GetSEE = PieceAbsValue(Move.Captured): Exit
Function
2503 If Move.Castle <> NO_CASTLE Then Exit Function
2504 From = Move.From: MoveTo = Move.Target: PieceMoved = CBool(Board(From) = NO_PIECE)
2505 If Not PieceMoved Then
2506 'If PinnedPieceDir(From, MoveTo, PieceColor(PieceMoved)) <> 0 Then GetSEE = -100000: Exit Function
2507 Piece = Board(From): Board(From) = NO_PIECE 'Remove piece to open sliding xrays
2508 If Move.EnPassant = ENPASSANT_CAPTURE Then 'remove captured pawn not on target field
2509 If Piece = WPAWN Then Board(MoveTo + SQ_DOWN) = NO_PIECE Else Board(MoveTo +
SQ_UP) = NO_PIECE
2510 End If
2511 Else
2512 Piece = Board(MoveTo)
2513 End If
2514 Cnt = 0 'Counter for PieceList array of attackers (both sides)
2515 Erase Blocker 'Array to manage blocker of sliding pieces: -1: is blocked, >0: is blocking,index of blocked
piece, 0:not blocked/blocking
2516
2517 'Find attackers
2518 For i = 0 To 3 'horizontal+vertical
2519 Block = 0: Offset = DirectionOffset(i): Target = MoveTo + Offset
2520 If Board(Target) = BKING Or Board(Target) = WKING Then
2521 Cnt = Cnt + 1: PieceList(Cnt) = PieceScore(Board(Target))
2522 Else
2523
2524 Do While Board(Target) <> FRAME
2525 Select Case Board(Target)
2526 Case BROOK, BQUEEN, WROOK, WQUEEN
2527 Cnt = Cnt + 1: PieceList(Cnt) = PieceScore(Board(Target))
2528 If Block > 0 Then Blocker(Block) = Cnt: Blocker(Cnt) = -1 '1. point to blocked
piece index; 2. -1 = blocked
2529 Block = Cnt
2530 Case NO_PIECE, WEP_PIECE, BEP_PIECE
2531 '-- Continue
2532 Case Else
2533 Exit Do 'other piece
2534 End Select
2535
2536 Target = Target + Offset
2537 Loop
2538
2539 End If
2540 Next
2541
2542 For i = 4 To 7 'diagonal
2543 Block = 0: Offset = DirectionOffset(i): Target = MoveTo + Offset
2544
2545 Select Case Board(Target)
2546 Case BKING, WKING
2547 Cnt = Cnt + 1: PieceList(Cnt) = PieceScore(Board(Target))
2548 GoTo lblContinue
2549 Case WPAWN

```



```

2550         If i = 5 Or i = 7 Then Cnt = Cnt + 1: PieceList(Cnt) = PieceScore(Board(Target
2551         )): Block = Cnt: Target = Target + Offset
2552     Case BPAWN
2553         If i = 4 Or i = 6 Then Cnt = Cnt + 1: PieceList(Cnt) = PieceScore(Board(Target
2554         )): Block = Cnt: Target = Target + Offset
2555 End Select
2556
2557 Do While Board(Target) <> FRAME
2558     Select Case Board(Target)
2559     Case BBISHOP, BQUEEN, WBISHOP, WQUEEN
2560         Cnt = Cnt + 1: PieceList(Cnt) = PieceScore(Board(Target))
2561         If Block > 0 Then Blocker(Block) = Cnt: Blocker(Cnt) = -1 ' 1. point to blocked
2562         piece index; 2. -1 = blocked
2563         Block = Cnt
2564     Case NO_PIECE, WEP_PIECE, BEP_PIECE
2565         '-- Continue
2566     Case Else
2567         Exit Do 'other piece
2568     End Select
2569     Target = Target + Offset
2570 Loop
2571
2572 lblContinue:
2573 Next
2574
2575 ' Knights
2576 If PieceCnt(WKNIGHT) + PieceCnt(BKNIGHT) > 0 Then
2577     For i = 0 To 7
2578         Select Case Board(MoveTo + KnightOffsets(i))
2579         Case WKNIGHT, BKNIGHT: Cnt = Cnt + 1: PieceList(Cnt) = PieceScore(Board(MoveTo
2580         + KnightOffsets(i)))
2581         End Select
2582     Next
2583 End If
2584
2585 '---<<< End of collecting attackers ---
2586 ' Count Attackers for each color (non blocked only)
2587 For i = 1 To Cnt
2588     If PieceList(i) > 0 And Blocker(i) >= 0 Then NumAttackers(COL_WHITE) =
2589     NumAttackers(COL_WHITE) + 1 Else NumAttackers(COL_BLACK) = NumAttackers(COL_BLACK)
2590     + 1
2591 Next
2592
2593 ' Init swap list
2594 SwapList(0) = PieceAbsValue(Move.Captured)
2595 slIndex = 1
2596 SideToMove = PieceColor(Move.Piece)
2597 ' Switch side
2598 SideNotToMove = SideToMove: If SideToMove = COL_WHITE Then SideToMove = COL_BLACK
2599 Else SideToMove = COL_WHITE
2600 ' If the opponent has no attackers we are finished
2601 If NumAttackers(SideToMove) = 0 Then
2602     GoTo lblEndSEE
2603 End If
2604 If SideToMove = COL_WHITE Then CurrSgn = 1 Else CurrSgn = -1
2605 '--- CALCULATE SEE ---
2606 CapturedVal = PieceAbsValue(Move.Piece)
2607
2608 Do
2609     SwapList(slIndex) = -SwapList(slIndex - 1) + CapturedVal
2610     ' find least valuable attacker (min value)
2611     CapturedVal = 99999
2612     MinValIndex = -1
2613
2614     For i = 1 To Cnt
2615         If PieceList(i) <> 0 Then If Sgn(PieceList(i)) = CurrSgn Then If Blocker(i) >= 0
2616         Then If Abs(PieceList(i)) < CapturedVal Then CapturedVal = Abs(PieceList(i)):
2617         MinValIndex = i

```

```

2609     Next
2610
2611     If MinValIndex > 0 Then
2612         If Blocker(MinValIndex) > 0 Then ' unblock other sliding piece?
2613             Blocker(Blocker(MinValIndex)) = 0
2614             'Increase attack number
2615             If PieceList(Blocker(MinValIndex)) > 0 Then NumAttackers(COL_WHITE) =
                NumAttackers(COL_WHITE) + 1 Else NumAttackers(COL_BLACK) = NumAttackers(
                COL_BLACK) + 1
2616         End If
2617         PieceList(MinValIndex) = 0 ' Remove from list by setting piece value to zero
2618     End If
2619     If CapturedVal = 5000 Then ' King
2620         If NumAttackers(SideNotToMove) = 0 Then slIndex = slIndex + 1
2621         Exit Do ' King
2622     End If
2623     If CapturedVal = 99999 Then Exit Do
2624     NumAttackers(SideToMove) = NumAttackers(SideToMove) - 1
2625     CurrSgn = -CurrSgn: SideNotToMove = SideToMove: If SideToMove = COL_WHITE Then
        SideToMove = COL_BLACK Else SideToMove = COL_WHITE
2626     slIndex = slIndex + 1
2627     Loop While NumAttackers(SideToMove) > 0
2628
2629     '// Having built the swap list, we negamax through it to find the best
2630     '// achievable score from the point of view of the side to move.
2631     slIndex = slIndex - 1
2632
2633     Do While slIndex > 0
2634         'SwapList(slIndex - 1) = GetMin(-SwapList(slIndex), SwapList(slIndex - 1))
2635         If -SwapList(slIndex) < SwapList(slIndex - 1) Then SwapList(slIndex - 1) = -
            SwapList(slIndex)
2636         slIndex = slIndex - 1
2637     Loop
2638
2639 lblEndSEE:
2640     If Not PieceMoved Then
2641         Board(From) = Piece
2642         If Move.EnPassant = ENPASSANT_CAPTURE Then ' restore captured pawn not on target field
2643             If Piece = WPAWN Then Board(MoveTo + SQ_DOWN) = BPAWN Else Board(MoveTo + SQ_UP)
                = WPAWN
2644             End If
2645         End If
2646         GetSEE = SwapList(0)
2647     End Function
2648
2649 Public Sub InitPieceColor()
2650     Dim Piece As Long, PieceCol As Long
2651
2652     For Piece = 0 To 16
2653         If Piece < 1 Or Piece >= NO_PIECE Then
2654             PieceCol = COL_NOPIECE ' NO_PIECE, or EP-PIECE or FRAME
2655         Else
2656             If (Piece And 1) = WCOL Then PieceCol = COL_WHITE Else PieceCol = COL_BLACK
2657             End If
2658             PieceColor(Piece) = PieceCol
2659         Next
2660
2661     End Sub
2662
2663     'Public Function SwitchColor(Color As enumColor) As enumColor
2664     ' If Color = COL_WHITE Then SwitchColor = COL_BLACK Else SwitchColor = COL_WHITE
2665     'End Function
2666
2667 Public Sub InitSameXRay()
2668     Dim i As Long, j As Long
2669
2670     For i = SQ_A1 To SQ_H8
2671         If File(i) >= 1 And File(i) <= 8 And Rank(i) >= 1 And Rank(i) <= 8 Then

```

```

2672     DirOffset(i, j) = 0
2673     For j = SQ_A1 To SQ_H8
2674         If File(j) >= 1 And File(j) <= 8 And Rank(j) >= 1 And Rank(j) <= 8 Then
2675             If File(i) = File(j) Then
2676                 SameXRay(i, j) = True
2677                 If i < j Then DirOffset(i, j) = 10 Else If i > j Then DirOffset(i, j) = -
2678                     10
2679             ElseIf Rank(i) = Rank(j) Then
2680                 SameXRay(i, j) = True
2681                 If i < j Then DirOffset(i, j) = 1 Else If i > j Then DirOffset(i, j) = -1
2682             ElseIf Abs(File(i) - File(j)) = Abs(Rank(i) - Rank(j)) Then
2683                 SameXRay(i, j) = True
2684                 If Abs(j - i) Mod 9 = 0 Then
2685                     If i < j Then DirOffset(i, j) = 9 Else If i > j Then DirOffset(i, j) = -
2686                         9
2687                 Else
2688                     If i < j Then DirOffset(i, j) = 11 Else If i > j Then DirOffset(i, j) = -
2689                         11
2690                 End If
2691             Else
2692                 SameXRay(i, j) = False
2693             End If
2694         End If
2695     Next
2696
2697     End If
2698 Next
2699
2700 End Sub
2701
2702 Public Function IsCheckingMove(ByVal PieceFrom As Long, _
2703     ByVal From As Long, _
2704     ByVal Target As Long, _
2705     ByVal Promoted As Long, ByVal EnPassant As Long) As
2706     Boolean
2707
2708     ' is this a checking move?
2709     ' array KingCheckW/B must be set before with function FillKingCheckW / FillKingCheckB (fast detection logic)
2710     Dim bFound As Boolean, Offset As Long, SlidePos As Long, EpSquare As Long
2711     bFound = False: EpSquare = 0
2712
2713     '----- White piece moves -----
2714     If (PieceFrom And 1) = WCOL Then
2715         If Promoted > 0 Then
2716             PieceFrom = Promoted: If SqBetween(From, BKingLoc, Target) Then Target = From
2717             '--- to get KingCheck array offset
2718         ElseIf EnPassant = ENPASSANT_CAPTURE Then
2719             EpSquare = Target + SQ_DOWN
2720         ElseIf PieceFrom = WKING Then
2721             ' Castling check?
2722             If From = WKING_START Then
2723                 If Target = SQ_G1 Then '00
2724                     Target = SQ_F1: PieceFrom = WROOK
2725                 ElseIf Target = SQ_C1 Then '000
2726                     Target = SQ_D1: PieceFrom = WROOK
2727             End If
2728         End If
2729     End If
2730
2731     If KingCheckB(From) = 0 Then If KingCheckB(Target) = 0 Then If KingCheckB(EpSquare) = 0 Then IsCheckingMove = False: Exit Function
2732
2733     Select Case KingCheckB(Target)
2734     Case 0: ' ignore
2735     Case -9, -11:
2736         If PieceFrom = WPAWN Then

```

```

2734         If MaxDistance(Target, BKingLoc) = 1 Then bFound = True
2735     ElseIf PieceFrom = WQUEEN Or PieceFrom = WBISHOP Then
2736         bFound = True
2737     End If
2738     Case 9, 11: If PieceFrom = WQUEEN Or PieceFrom = WBISHOP Then bFound = True
2739     Case 1, -1, 10, -10: If PieceFrom = WQUEEN Or PieceFrom = WROOK Then bFound =
        True
2740     Case 8, -8, 12, -12, 19, -19, 21, -21: If PieceFrom = WKNIGHT Then bFound = True
2741 End Select
2742
2743 If Not bFound Then
2744     '--- Sliding Check? also continue loop for EnPassant square
2745     Offset = KingCheckB(From): SlidePos = From
2746     Do
2747         Select Case Abs(Offset)
2748             Case 0, 8, 12, 19, 21: 'empty or Knight> ignore
2749             Case Else
2750                 If SqBetween(SlidePos, BKingLoc, Target) Then '--- ignore if move in same direction
                    towards king
2751                     ' ignore
2752                 ElseIf SqBetween(Target, BKingLoc, SlidePos) Then '--- ignore if move in same
                    direction towards king
2753                     ' ignore
2754                 Else
2755                     Select Case Abs(Offset) 'check needed?
2756                         Case 1, 10: If PieceCnt(WROOK) + PieceCnt(WQUEEN) + Promoted = 0 Then
                            GoTo lblNextWSq
2757                         Case 9, 11: If PieceCnt(WBISHOP) + PieceCnt(WQUEEN) + Promoted = 0
                            Then GoTo lblNextWSq
2758                     End Select
2759                     Do 'search for piece or border
2760                         SlidePos = SlidePos + Offset
2761                         Select Case Board(SlidePos)
2762                             Case NO_PIECE, WEP_PIECE, BEP_PIECE: '- go on
2763                             Case FRAME: Exit Do
2764                             Case WQUEEN: bFound = True
2765                                 Exit Do
2766                             Case WROOK: If Abs(Offset) = 10 Or Abs(Offset) = 1 Then bFound =
                                    True
2767                                 Exit Do
2768                             Case WBISHOP: If Abs(Offset) = 9 Or Abs(Offset) = 11 Then bFound =
                                    True
2769                                 Exit Do
2770                             Case Else
2771                                 Exit Do
2772                         End Select
2773                     Loop
2774                     End If
2775                 End Select
2776                 If bFound Then Exit Do
2777             lblNextWSq:
2778                 If EpSquare = 0 Then Exit Do
2779                 '--- additional EP - Check
2780                 If SqBetween(EpSquare, BKingLoc, From) Then 'King, EpSquare, attacker in same row
                    'Fix for position "8/8/3kPpR1/8/8/4K3 w - f6 0 1" Enpassant e5xf6ep/ changed2023' debug.print
                    printpos, LocCoord(from), LocCoord(target), LocCoord(EpSquare), KingCheckB(EpSquare)
2781                 Offset = KingCheckB(EpSquare): If Offset <> 0 Then SlidePos = From
2782                 ElseIf SqBetween(From, BKingLoc, EpSquare) Then
                    'Fix for position : 2. case "8/8/1k1pP1R1/8/8/4K3 w - d6 0 1" Enpassant d5xc6ep/ changed2023
2783                 Offset = KingCheckB(From): If Offset <> 0 Then SlidePos = EpSquare
2784                 ElseIf KingCheckB(EpSquare) = 0 Then
                    Exit Do
2785                 Else
2786                     Offset = KingCheckB(EpSquare): If Offset <> 0 Then SlidePos = EpSquare
2787                     Else Exit Do 'do a second loop behind EpSquare
2788                 End If
2789                 EpSquare = 0
2790             Loop '----- search for slider check

```

```

2793
2794     End If
2795
2796 '----- Black piece moves -----
2797 ElseIf (PieceFrom And 1) = BCOL Then
2798     If Promoted > 0 Then
2799         PieceFrom = Promoted: If SqBetween(From, WKingLoc, Target) Then Target = From
2800         '--- to get KingCheck array offset
2801     ElseIf EnPassant = ENPASSANT_CAPTURE Then
2802         EpSquare = Target + SQ_UP
2803     ElseIf PieceFrom = BKING Then
2804         ' Castling check?
2805         If From = BKING_START Then
2806             If Target = SQ_G8 Then '00
2807                 Target = SQ_F8: PieceFrom = BROOK
2808             ElseIf Target = SQ_C8 Then '000
2809                 Target = SQ_D8: PieceFrom = BROOK
2810             End If
2811         End If
2812     End If
2813     If KingCheckW(From) = 0 Then If KingCheckW(Target) = 0 Then If KingCheckW(EpSquare
2814 ) = 0 Then IsCheckingMove = False: Exit Function
2815
2816 Select Case KingCheckW(Target)
2817     Case 0: 'ignore
2818     Case 9, 11:
2819         If PieceFrom = BPAWN Then
2820             If MaxDistance(Target, WKingLoc) = 1 Then bFound = True
2821             ElseIf PieceFrom = BQUEEN Or PieceFrom = BBISHOP Then
2822                 bFound = True
2823             End If
2824         Case -9, -11: If PieceFrom = BQUEEN Or PieceFrom = BBISHOP Then bFound = True
2825         Case 1, -1, 10, -10: If PieceFrom = BQUEEN Or PieceFrom = BROOK Then bFound =
2826 True
2827         Case 8, -8, 12, -12, 19, -19, 21, -21: If PieceFrom = BKNIGHT Then bFound = True
2828 End Select
2829
2830 If Not bFound Then
2831     '--- Sliding Check? also continue loop for EnPassant square
2832     Offset = KingCheckW(From): SlidePos = From
2833     Do
2834         Select Case Abs(Offset)
2835             Case 0, 8, 12, 19, 21: 'empty or Knight> ignore
2836             Case Else
2837                 If SqBetween(SlidePos, WKingLoc, Target) Then '--- ignore if move in same direction
2838                 towards king
2839                 ' ignore
2840             ElseIf SqBetween(Target, WKingLoc, SlidePos) Then '--- ignore if move in same
2841             direction towards king
2842             ' ignore
2843             Else
2844
2845                 Select Case Abs(Offset) 'check needed?
2846                     Case 1, 10: If PieceCnt(BROOK) + PieceCnt(BQUEEN) + Promoted = 0 Then
2847                     GoTo lblNextBSq
2848                     Case 9, 11: If PieceCnt(BBISHOP) + PieceCnt(BQUEEN) + Promoted = 0
2849                     Then GoTo lblNextBSq
2850                 End Select
2851
2852                 Do
2853                     SlidePos = SlidePos + Offset
2854                     Select Case Board(SlidePos)
2855                         Case NO_PIECE, WEP_PIECE, BEP_PIECE: '- go on
2856                         Case FRAME: Exit Do
2857                         Case BQUEEN: bFound = True
2858                     Exit Do
2859                     Case BROOK: If Abs(Offset) = 10 Or Abs(Offset) = 1 Then bFound =
2860 True

```

```

2853             Exit Do
2854         Case BBISHOP: If Abs(Offset) = 9 Or Abs(Offset) = 11 Then bFound =
                True
2855             Exit Do
2856         Case Else
2857             Exit Do
2858         End Select
2859     Loop
2860 End If
2861 End Select
2862 If bFound Then Exit Do
2863 lblNextBSq:
2864     If EpSquare = 0 Then Exit Do
2865     '--- additional EP - Check
2866     If SqBetween(EpSquare, WKingLoc, From) Then ' King, EpSquare, attacker in same row
2867         Offset = KingCheckW(EpSquare): If Offset <> 0 Then SlidePos = From
2868     ElseIf SqBetween(From, WKingLoc, EpSquare) Then
2869         Offset = KingCheckW(From): If Offset <> 0 Then SlidePos = EpSquare
2870     ElseIf KingCheckW(EpSquare) = 0 Then
2871         Exit Do
2872     Else
2873         Offset = KingCheckW(EpSquare): If Offset <> 0 Then SlidePos = EpSquare
2874         Else Exit Do ' do a second loop behind EpSquare
2875     End If
2876     EpSquare = 0
2877     Loop '----- search for slider check
2878 End If
2879 IsCheckingMove = bFound
2880 'If bFound And EnPassant = ENPASSANT_CAPTURE And Target <> 74 Then Stop
2881 End Function
2882
2883 Public Sub InitBoardColors()
2884     Dim x As Long, y As Long, ColSq As Long, IsWhite As Boolean
2885
2886     For y = 1 To 8
2887         IsWhite = CBool((y And 1) = 0)
2888
2889         For x = 1 To 8
2890             If IsWhite Then ColSq = COL_WHITE Else ColSq = COL_BLACK
2891             ColorSq(20 + x + (y - 1) * 10) = ColSq
2892             IsWhite = Not IsWhite
2893         Next
2894     Next
2895
2896 End Sub
2897
2898 Public Function CoordToLoc(ByVal isCoord As String) As Long
2899     ' "A1" => 21 ( board array index )
2900     If Len(isCoord) = 2 Then
2901         CoordToLoc = 10 + Asc(Left$(LCase$(isCoord), 1)) - 96 + Val(Mid$(isCoord, 2)) * 10
2902     Else
2903         CoordToLoc = 0
2904     End If
2905 End Function
2906
2907 Public Function MovesEqual(m1 As TMOVE, m2 As TMOVE) As Boolean
2908     MovesEqual = False 'same moves?
2909     If m1.From = m2.From Then If m1.Target = m2.Target Then If m1.Piece = m2.Piece Then
2910         If m1.Promoted = m2.Promoted Then MovesEqual = True
2911     End If
2912 End Function
2913
2914 Public Function WCanCastleOO() As Boolean
2915     ' not checked for attacked squares
2916     WCanCastleOO = False
2917     If Moved(WKING_START) = 0 Then If Moved(SQ_H1) = 0 Then If Board(SQ_H1) = WROOK Then
2918         If Board(SQ_F1) = NO_PIECE And Board(SQ_G1) = NO_PIECE Then WCanCastleOO = True
2919     End If
2920 End Function

```

```

2917
2918 Public Function WCanCastle000() As Boolean
2919     'not checked for attacked squares
2920     WCanCastle000 = False
2921     If Moved(WKING_START) = 0 Then If Moved(SQ_A1) = 0 Then If Board(SQ_A1) = WROOK Then
2922         If Board(SQ_B1) = NO_PIECE And Board(SQ_C1) = NO_PIECE And Board(SQ_D1) = NO_PIECE
2923         Then WCanCastle000 = True
2924     End Function
2925
2926 Public Function BCanCastle00() As Boolean
2927     'not checked for attacked squares
2928     BCanCastle00 = False
2929     If Moved(BKING_START) = 0 Then If Moved(SQ_H8) = 0 Then If Board(SQ_H8) = BROOK Then
2930         If Board(SQ_F8) = NO_PIECE And Board(SQ_G8) = NO_PIECE Then BCanCastle00 = True
2931     End Function
2932
2933 Public Function BCanCastle000() As Boolean
2934     'not checked for attacked squares
2935     BCanCastle000 = False
2936     If Moved(BKING_START) = 0 Then If Moved(SQ_A8) = 0 Then If Board(SQ_A8) = BROOK Then
2937         If Board(SQ_B8) = NO_PIECE And Board(SQ_C8) = NO_PIECE And Board(SQ_D8) = NO_PIECE
2938         Then BCanCastle000 = True
2939     End Function
2940
2941 Public Function GetMoveFromSAN(ByVal isSAN As String) As TMOVE
2942     'read Standard Algebraic Notation like "Rexd1"
2943     Dim SANMove As TMOVE, FileFrom As Long, RankFrom As Long
2944     GetMoveFromSAN = EmptyMove
2945     isSAN = Trim$(isSAN)
2946     isSAN = Replace(isSAN, "x", "")
2947     isSAN = Replace(isSAN, "+", "")
2948     isSAN = Replace(isSAN, "-", "")
2949     isSAN = Replace(isSAN, "=", "")
2950     isSAN = Replace(isSAN, "#", "")
2951     isSAN = Replace(isSAN, "e.p.", "")
2952     If isSAN = "" Then Exit Function
2953     SANMove = EmptyMove
2954
2955     'Get piece type
2956     Select Case Left$(isSAN, 1)
2957     Case "K": isSAN = Mid$(isSAN, 2): If bWhiteToMove Then SANMove.Piece = WKING Else
2958         SANMove.Piece = BKING
2959     Case "O", "o": 'Castle
2960         isSAN = Mid$(isSAN, 2): If bWhiteToMove Then SANMove.Piece = WKING Else
2961             SANMove.Piece = BKING
2962         If UCase$(Left$(isSAN, 2)) = "OO" Then
2963             If bWhiteToMove Then
2964                 SANMove.From = SQ_E1: SANMove.Target = SQ_G1: SANMove.Castle = WHITEOO
2965             Else
2966                 SANMove.From = SQ_E8: SANMove.Target = SQ_G8: SANMove.Castle = BLACKOO
2967             End If
2968         ElseIf UCase$(Left$(isSAN, 3)) = "OOO" Then
2969             If bWhiteToMove Then
2970                 SANMove.From = SQ_E8: SANMove.Target = SQ_C8: SANMove.Castle = WHITEOOO
2971             Else
2972                 SANMove.From = SQ_E8: SANMove.Target = SQ_G8: SANMove.Castle = BLACKOOO
2973             End If
2974         Else
2975             Exit Function
2976         End If
2977     Case "B": isSAN = Mid$(isSAN, 2): If bWhiteToMove Then SANMove.Piece = WBISHOP
2978     Else SANMove.Piece = BBISHOP
2979     Case "N": isSAN = Mid$(isSAN, 2): If bWhiteToMove Then SANMove.Piece = WKNIGHT
2980     Else SANMove.Piece = BKNIGHT
2981     Case "R": isSAN = Mid$(isSAN, 2): If bWhiteToMove Then SANMove.Piece = WROOK Else
2982         SANMove.Piece = BROOK

```



```

2975     Case "Q": isSAN = Mid$(isSAN, 2): If bWhiteToMove Then SANMove.Piece = WQUEEN Else
        SANMove.Piece = BQUEEN
2976     Case "a" To "h": If bWhiteToMove Then SANMove.Piece = WPAWN Else SANMove.Piece =
        BPAWN
2977     Case Else
2978         Exit Function
2979 End Select
2980
2981 'd5 or ed5 or 1d5 or d8d5
2982 FileFrom = 0: RankFrom = 0
2983 If IsNumeric(Mid$(isSAN, 4, 1)) And IsNumeric(Mid$(isSAN, 2, 1)) Then
2984     'd8d5
2985     SANMove.From = FileRev(Left$(isSAN, 1)) + RankRev(Mid$(isSAN, 2, 1))
2986     isSAN = Mid$(isSAN, 3)
2987 ElseIf IsNumeric(Mid$(isSAN, 3, 1)) Then
2988     'ed5 or 1d5
2989     If IsNumeric(Left$(isSAN, 1)) Then
2990         RankFrom = RankRev(Left$(isSAN, 1))
2991     Else
2992         FileFrom = FileRev(Left$(isSAN, 1))
2993     End If
2994     isSAN = Mid$(isSAN, 2)
2995 End If
2996 'Get target square
2997 SANMove.Target = FileRev(Left$(isSAN, 1)) + RankRev(Mid$(isSAN, 2, 1))
2998 isSAN = Trim$(Mid$(isSAN, 3))
2999 If isSAN <> "" Then 'Promote: e8=Q
3000
3001     Select Case Left$(isSAN, 1)
3002         Case "B": If bWhiteToMove Then SANMove.Promoted = WBISHOP Else SANMove.Promoted
            = BBISHOP
3003         Case "N": If bWhiteToMove Then SANMove.Promoted = WKnight Else SANMove.Promoted
            = BKNIGHT
3004         Case "R": If bWhiteToMove Then SANMove.Promoted = WROOK Else SANMove.Promoted =
            BROOK
3005         Case "Q": If bWhiteToMove Then SANMove.Promoted = WQUEEN Else SANMove.Promoted =
            BQUEEN
3006     End Select
3007
3008     If SANMove.Promoted > 0 Then SANMove.Piece = SANMove.Promoted
3009 End If
3010 lblTestMoves:
3011 Dim iNumMoves As Long, i As Long, bLegalInput As Boolean
3012 GenerateMoves Ply, False, iNumMoves
3013
3014 'find move
3015 For i = 0 To iNumMoves - 1
3016     If SANMove.Piece = Moves(Ply, i).Piece And SANMove.Target = Moves(Ply, i).Target
        Then
3017         If SANMove.From > 0 Then If SANMove.From <> Moves(Ply, i).From Then GoTo
            lblNextMove
3018         If FileFrom > 0 Then If FileFrom <> File(Moves(Ply, i).From) Then GoTo
            lblNextMove
3019         If RankFrom > 0 Then If RankFrom <> Rank(Moves(Ply, i).From) Then GoTo
            lblNextMove
3020         If SANMove.Promoted > 0 Then If SANMove.Promoted <> Moves(Ply, i).Promoted Then
            GoTo lblNextMove
3021         'Ok, check if legal move
3022         RemoveEpPiece
3023         MakeMove Moves(Ply, i)
3024         If CheckLegal(Moves(Ply, i)) Then
3025             bLegalInput = True
3026             GetMoveFromSAN = Moves(Ply, i) 'found!
3027         End If
3028         UnmakeMove Moves(Ply, i)
3029         ResetEpPiece
3030     End If
3031 lblNextMove:

```



```

3032         If bLegalInput Then Exit For
3033     Next
3034
3035 End Function
3036
3037
3038 "--- Bit functions ---
3039 " many lines of codes, but very fast
3040 'Public Function BitsShiftLeft(ByVal Value As Long, ByVal ShiftCount As Long) As Long
3041 '
3042 ' '- Shifts the bits to the left the specified number of positions and returns the new value.
3043 ' '- Bits "falling off" the left edge do not wrap around. Fill bits coming in from right are 0.
3044 ' '- A shift left is effectively a multiplication by 2. Some common languages like C/C++ or Java have an operator for
3045 ' this job: "<<".
3046 ' Select Case ShiftCount
3047 '     Case 0&
3048 '         BitsShiftLeft = Value
3049 '     Case 1&
3050 '         If Value And &H40000000 Then
3051 '             BitsShiftLeft = (Value And &H3FFFFFFF) * &H2& Or &H80000000
3052 '         Else
3053 '             BitsShiftLeft = (Value And &H3FFFFFFF) * &H2&
3054 '         End If
3055 '     Case 2&
3056 '         If Value And &H20000000 Then
3057 '             BitsShiftLeft = (Value And &H1FFFFFFF) * &H4& Or &H80000000
3058 '         Else
3059 '             BitsShiftLeft = (Value And &H1FFFFFFF) * &H4&
3060 '         End If
3061 '     Case 3&
3062 '         If Value And &H10000000 Then
3063 '             BitsShiftLeft = (Value And &HFFFFFFF) * &H8& Or &H80000000
3064 '         Else
3065 '             BitsShiftLeft = (Value And &HFFFFFFF) * &H8&
3066 '         End If
3067 '     Case 4&
3068 '         If Value And &H8000000 Then
3069 '             BitsShiftLeft = (Value And &H7FFFFFFF) * &H10& Or &H80000000
3070 '         Else
3071 '             BitsShiftLeft = (Value And &H7FFFFFFF) * &H10&
3072 '         End If
3073 '     Case 5&
3074 '         If Value And &H4000000 Then
3075 '             BitsShiftLeft = (Value And &H3FFFFFFF) * &H20& Or &H80000000
3076 '         Else
3077 '             BitsShiftLeft = (Value And &H3FFFFFFF) * &H20&
3078 '         End If
3079 '     Case 6&
3080 '         If Value And &H2000000 Then
3081 '             BitsShiftLeft = (Value And &H1FFFFFFF) * &H40& Or &H80000000
3082 '         Else
3083 '             BitsShiftLeft = (Value And &H1FFFFFFF) * &H40&
3084 '         End If
3085 '     Case 7&
3086 '         If Value And &H1000000 Then
3087 '             BitsShiftLeft = (Value And &HFFFFFFF) * &H80& Or &H80000000
3088 '         Else
3089 '             BitsShiftLeft = (Value And &HFFFFFFF) * &H80&
3090 '         End If
3091 '     Case 8&
3092 '         If Value And &H800000 Then
3093 '             BitsShiftLeft = (Value And &H7FFFFFFF) * &H100& Or &H80000000
3094 '         Else
3095 '             BitsShiftLeft = (Value And &H7FFFFFFF) * &H100&
3096 '         End If
3097 '     Case 9&
3098 '         If Value And &H400000 Then
3099 '             BitsShiftLeft = (Value And &H3FFFFFFF) * &H200& Or &H80000000

```

```

3099 ' Else
3100 '     BitsShiftLeft = (Value And &H3FFFFFF) * &H200&
3101 ' End If
3102 ' Case 10&
3103 ' If Value And &H200000 Then
3104 '     BitsShiftLeft = (Value And &H1FFFFFF) * &H400& Or &H80000000
3105 ' Else
3106 '     BitsShiftLeft = (Value And &H1FFFFFF) * &H400&
3107 ' End If
3108 ' Case 11&
3109 ' If Value And &H100000 Then
3110 '     BitsShiftLeft = (Value And &HFFFFFF) * &H800& Or &H80000000
3111 ' Else
3112 '     BitsShiftLeft = (Value And &HFFFFFF) * &H800&
3113 ' End If
3114 ' Case 12&
3115 ' If Value And &H80000 Then
3116 '     BitsShiftLeft = (Value And &H7FFFF) * &H1000& Or &H80000000
3117 ' Else
3118 '     BitsShiftLeft = (Value And &H7FFFF) * &H1000&
3119 ' End If
3120 ' Case 13&
3121 ' If Value And &H40000 Then
3122 '     BitsShiftLeft = (Value And &H3FFFF) * &H2000& Or &H80000000
3123 ' Else
3124 '     BitsShiftLeft = (Value And &H3FFFF) * &H2000&
3125 ' End If
3126 ' Case 14&
3127 ' If Value And &H20000 Then
3128 '     BitsShiftLeft = (Value And &H1FFFF) * &H4000& Or &H80000000
3129 ' Else
3130 '     BitsShiftLeft = (Value And &H1FFFF) * &H4000&
3131 ' End If
3132 ' Case 15&
3133 ' If Value And &H10000 Then
3134 '     BitsShiftLeft = (Value And &HFFFF&) * &H8000& Or &H80000000
3135 ' Else
3136 '     BitsShiftLeft = (Value And &HFFFF&) * &H8000&
3137 ' End If
3138 ' Case 16&
3139 ' If Value And &H8000& Then
3140 '     BitsShiftLeft = (Value And &H7FFF&) * &H10000 Or &H80000000
3141 ' Else
3142 '     BitsShiftLeft = (Value And &H7FFF&) * &H10000
3143 ' End If
3144 ' Case 17&
3145 ' If Value And &H4000& Then
3146 '     BitsShiftLeft = (Value And &H3FFF&) * &H20000 Or &H80000000
3147 ' Else
3148 '     BitsShiftLeft = (Value And &H3FFF&) * &H20000
3149 ' End If
3150 ' Case 18&
3151 ' If Value And &H2000& Then
3152 '     BitsShiftLeft = (Value And &H1FFF&) * &H40000 Or &H80000000
3153 ' Else
3154 '     BitsShiftLeft = (Value And &H1FFF&) * &H40000
3155 ' End If
3156 ' Case 19&
3157 ' If Value And &H1000& Then
3158 '     BitsShiftLeft = (Value And &HFFF&) * &H80000 Or &H80000000
3159 ' Else
3160 '     BitsShiftLeft = (Value And &HFFF&) * &H80000
3161 ' End If
3162 ' Case 20&
3163 ' If Value And &H800& Then
3164 '     BitsShiftLeft = (Value And &H7FF&) * &H100000 Or &H80000000
3165 ' Else
3166 '     BitsShiftLeft = (Value And &H7FF&) * &H100000

```

```

3167 ' End If
3168 ' Case 21&
3169 ' If Value And &H400& Then
3170 '     BitsShiftLeft = (Value And &H3FF&) * &H200000 Or &H80000000
3171 ' Else
3172 '     BitsShiftLeft = (Value And &H3FF&) * &H200000
3173 ' End If
3174 ' Case 22&
3175 ' If Value And &H200& Then
3176 '     BitsShiftLeft = (Value And &H1FF&) * &H400000 Or &H80000000
3177 ' Else
3178 '     BitsShiftLeft = (Value And &H1FF&) * &H400000
3179 ' End If
3180 ' Case 23&
3181 ' If Value And &H100& Then
3182 '     BitsShiftLeft = (Value And &HFF&) * &H800000 Or &H80000000
3183 ' Else
3184 '     BitsShiftLeft = (Value And &HFF&) * &H800000
3185 ' End If
3186 ' Case 24&
3187 ' If Value And &H80& Then
3188 '     BitsShiftLeft = (Value And &H7F&) * &H1000000 Or &H80000000
3189 ' Else
3190 '     BitsShiftLeft = (Value And &H7F&) * &H1000000
3191 ' End If
3192 ' Case 25&
3193 ' If Value And &H40& Then
3194 '     BitsShiftLeft = (Value And &H3F&) * &H2000000 Or &H80000000
3195 ' Else
3196 '     BitsShiftLeft = (Value And &H3F&) * &H2000000
3197 ' End If
3198 ' Case 26&
3199 ' If Value And &H20& Then
3200 '     BitsShiftLeft = (Value And &H1F&) * &H4000000 Or &H80000000
3201 ' Else
3202 '     BitsShiftLeft = (Value And &H1F&) * &H4000000
3203 ' End If
3204 ' Case 27&
3205 ' If Value And &H10& Then
3206 '     BitsShiftLeft = (Value And &HF&) * &H8000000 Or &H80000000
3207 ' Else
3208 '     BitsShiftLeft = (Value And &HF&) * &H8000000
3209 ' End If
3210 ' Case 28&
3211 ' If Value And &H8& Then
3212 '     BitsShiftLeft = (Value And &H7&) * &H10000000 Or &H80000000
3213 ' Else
3214 '     BitsShiftLeft = (Value And &H7&) * &H10000000
3215 ' End If
3216 ' Case 29&
3217 ' If Value And &H4& Then
3218 '     BitsShiftLeft = (Value And &H3&) * &H20000000 Or &H80000000
3219 ' Else
3220 '     BitsShiftLeft = (Value And &H3&) * &H20000000
3221 ' End If
3222 ' Case 30&
3223 ' If Value And &H2& Then
3224 '     BitsShiftLeft = (Value And &H1&) * &H40000000 Or &H80000000
3225 ' Else
3226 '     BitsShiftLeft = (Value And &H1&) * &H40000000
3227 ' End If
3228 ' Case 31&
3229 ' If Value And &H1& Then
3230 '     BitsShiftLeft = &H80000000
3231 ' Else
3232 '     BitsShiftLeft = &H0&
3233 ' End If
3234 ' End Select

```

```

3235 '
3236 'End Function
3237 '
3238 'Public Function BitsShiftRight(ByVal Value As Long, ByVal ShiftCount As Long) As Long
3239 '
3240 ' ' Shifts the bits to the right the specified number of positions and returns the new value.
3241 ' ' Bits "falling off" the right edge do not wrap around. Fill bits coming in from left match bit 31 (the sign bit): if bit 31 is
3242 ' ' 1 the fill bits will be 1 (see ShiftRightZ for the alternative zero-fill-in version).
3242 ' ' A shift right is effectively a division by 2 (rounding downward, see Examples). Some common languages like
3242 ' C/C++ or Java have an operator for this job: ">>"
3243 ' Select Case ShiftCount
3244 ' Case 0&: BitsShiftRight = Value
3245 ' Case 1&: BitsShiftRight = (Value And &HFFFFFFFE) \ &H2&
3246 ' Case 2&: BitsShiftRight = (Value And &HFFFFFFFC) \ &H4&
3247 ' Case 3&: BitsShiftRight = (Value And &HFFFFFFF8) \ &H8&
3248 ' Case 4&: BitsShiftRight = (Value And &HFFFFFFF0) \ &H10&
3249 ' Case 5&: BitsShiftRight = (Value And &HFFFFFFE0) \ &H20&
3250 ' Case 6&: BitsShiftRight = (Value And &HFFFFFFC0) \ &H40&
3251 ' Case 7&: BitsShiftRight = (Value And &HFFFFFF80) \ &H80&
3252 ' Case 8&: BitsShiftRight = (Value And &HFFFFFF00) \ &H100&
3253 ' Case 9&: BitsShiftRight = (Value And &HFFFFFFE0) \ &H200&
3254 ' Case 10&: BitsShiftRight = (Value And &HFFFFFFC0) \ &H400&
3255 ' Case 11&: BitsShiftRight = (Value And &HFFFFFF80) \ &H800&
3256 ' Case 12&: BitsShiftRight = (Value And &HFFFFFF00) \ &H1000&
3257 ' Case 13&: BitsShiftRight = (Value And &HFFFFE000) \ &H2000&
3258 ' Case 14&: BitsShiftRight = (Value And &HFFFFC000) \ &H4000&
3259 ' Case 15&: BitsShiftRight = (Value And &HFFFF8000) \ &H8000&
3260 ' Case 16&: BitsShiftRight = (Value And &HFFFF0000) \ &H10000
3261 ' Case 17&: BitsShiftRight = (Value And &HFFFE0000) \ &H20000
3262 ' Case 18&: BitsShiftRight = (Value And &HFFFC0000) \ &H40000
3263 ' Case 19&: BitsShiftRight = (Value And &HFFF80000) \ &H80000
3264 ' Case 20&: BitsShiftRight = (Value And &HFFF00000) \ &H100000
3265 ' Case 21&: BitsShiftRight = (Value And &HFFE00000) \ &H200000
3266 ' Case 22&: BitsShiftRight = (Value And &HFFC00000) \ &H400000
3267 ' Case 23&: BitsShiftRight = (Value And &HFF800000) \ &H800000
3268 ' Case 24&: BitsShiftRight = (Value And &HFF000000) \ &H1000000
3269 ' Case 25&: BitsShiftRight = (Value And &HFE000000) \ &H2000000
3270 ' Case 26&: BitsShiftRight = (Value And &HFC000000) \ &H4000000
3271 ' Case 27&: BitsShiftRight = (Value And &HF8000000) \ &H8000000
3272 ' Case 28&: BitsShiftRight = (Value And &HF0000000) \ &H10000000
3273 ' Case 29&: BitsShiftRight = (Value And &HE0000000) \ &H20000000
3274 ' Case 30&: BitsShiftRight = (Value And &HC0000000) \ &H40000000
3275 ' Case 31&: BitsShiftRight = CBool(Value And &H80000000)
3276 ' End Select
3277 '
3278 'End Function
3279 '
3280 'Public Function BitsShiftRightZ(ByVal Value As Long, ByVal ShiftCount As Long) As Long
3281 ' ' - Shifts the bits to the right the specified number of positions and returns the new value.
3282 ' ' - Bits "falling off" the right edge do not wrap around. Fill bits coming in from left are 0 (zero, hence "ShiftRightZ",
3282 ' see ShiftRight for the alternative signbit-fill-in version)
3283 ' If Value And &H80000000 Then
3284 '
3285 ' Select Case ShiftCount
3286 ' Case 0&: BitsShiftRightZ = Value
3287 ' Case 1&: BitsShiftRightZ = &H40000000 Or (Value And &H7FFFFFFF) \ &H2&
3288 ' Case 2&: BitsShiftRightZ = &H20000000 Or (Value And &H7FFFFFFF) \ &H4&
3289 ' Case 3&: BitsShiftRightZ = &H10000000 Or (Value And &H7FFFFFFF) \ &H8&
3290 ' Case 4&: BitsShiftRightZ = &H8000000 Or (Value And &H7FFFFFFF) \ &H10&
3291 ' Case 5&: BitsShiftRightZ = &H4000000 Or (Value And &H7FFFFFFF) \ &H20&
3292 ' Case 6&: BitsShiftRightZ = &H2000000 Or (Value And &H7FFFFFFF) \ &H40&
3293 ' Case 7&: BitsShiftRightZ = &H1000000 Or (Value And &H7FFFFFFF) \ &H80&
3294 ' Case 8&: BitsShiftRightZ = &H800000 Or (Value And &H7FFFFFFF) \ &H100&
3295 ' Case 9&: BitsShiftRightZ = &H400000 Or (Value And &H7FFFFFFF) \ &H200&
3296 ' Case 10&: BitsShiftRightZ = &H200000 Or (Value And &H7FFFFFFF) \ &H400&
3297 ' Case 11&: BitsShiftRightZ = &H100000 Or (Value And &H7FFFFFFF) \ &H800&
3298 ' Case 12&: BitsShiftRightZ = &H80000 Or (Value And &H7FFFFFFF) \ &H1000&
3299 ' Case 13&: BitsShiftRightZ = &H40000 Or (Value And &H7FFFFFFF) \ &H2000&

```

```

3300 ' Case 14&: BitsShiftRightZ = &H20000 Or (Value And &H7FFFFFFF) \ &H4000&
3301 ' Case 15&: BitsShiftRightZ = &H10000 Or (Value And &H7FFFFFFF) \ &H8000&
3302 ' Case 16&: BitsShiftRightZ = &H8000& Or (Value And &H7FFFFFFF) \ &H10000
3303 ' Case 17&: BitsShiftRightZ = &H4000& Or (Value And &H7FFFFFFF) \ &H20000
3304 ' Case 18&: BitsShiftRightZ = &H2000& Or (Value And &H7FFFFFFF) \ &H40000
3305 ' Case 19&: BitsShiftRightZ = &H1000& Or (Value And &H7FFFFFFF) \ &H80000
3306 ' Case 20&: BitsShiftRightZ = &H800& Or (Value And &H7FFFFFFF) \ &H100000
3307 ' Case 21&: BitsShiftRightZ = &H400& Or (Value And &H7FFFFFFF) \ &H200000
3308 ' Case 22&: BitsShiftRightZ = &H200& Or (Value And &H7FFFFFFF) \ &H400000
3309 ' Case 23&: BitsShiftRightZ = &H100& Or (Value And &H7FFFFFFF) \ &H800000
3310 ' Case 24&: BitsShiftRightZ = &H80& Or (Value And &H7FFFFFFF) \ &H1000000
3311 ' Case 25&: BitsShiftRightZ = &H40& Or (Value And &H7FFFFFFF) \ &H2000000
3312 ' Case 26&: BitsShiftRightZ = &H20& Or (Value And &H7FFFFFFF) \ &H4000000
3313 ' Case 27&: BitsShiftRightZ = &H10& Or (Value And &H7FFFFFFF) \ &H8000000
3314 ' Case 28&: BitsShiftRightZ = &H8& Or (Value And &H7FFFFFFF) \ &H10000000
3315 ' Case 29&: BitsShiftRightZ = &H4& Or (Value And &H7FFFFFFF) \ &H20000000
3316 ' Case 30&: BitsShiftRightZ = &H2& Or (Value And &H7FFFFFFF) \ &H40000000
3317 ' Case 31&: BitsShiftRightZ = &H1&
3318 ' End Select
3319 '
3320 ' Else
3321 '
3322 ' Select Case ShiftCount
3323 ' Case 0&: BitsShiftRightZ = Value
3324 ' Case 1&: BitsShiftRightZ = Value \ &H2&
3325 ' Case 2&: BitsShiftRightZ = Value \ &H4&
3326 ' Case 3&: BitsShiftRightZ = Value \ &H8&
3327 ' Case 4&: BitsShiftRightZ = Value \ &H10&
3328 ' Case 5&: BitsShiftRightZ = Value \ &H20&
3329 ' Case 6&: BitsShiftRightZ = Value \ &H40&
3330 ' Case 7&: BitsShiftRightZ = Value \ &H80&
3331 ' Case 8&: BitsShiftRightZ = Value \ &H100&
3332 ' Case 9&: BitsShiftRightZ = Value \ &H200&
3333 ' Case 10&: BitsShiftRightZ = Value \ &H400&
3334 ' Case 11&: BitsShiftRightZ = Value \ &H800&
3335 ' Case 12&: BitsShiftRightZ = Value \ &H1000&
3336 ' Case 13&: BitsShiftRightZ = Value \ &H2000&
3337 ' Case 14&: BitsShiftRightZ = Value \ &H4000&
3338 ' Case 15&: BitsShiftRightZ = Value \ &H8000&
3339 ' Case 16&: BitsShiftRightZ = Value \ &H10000
3340 ' Case 17&: BitsShiftRightZ = Value \ &H20000
3341 ' Case 18&: BitsShiftRightZ = Value \ &H40000
3342 ' Case 19&: BitsShiftRightZ = Value \ &H80000
3343 ' Case 20&: BitsShiftRightZ = Value \ &H100000
3344 ' Case 21&: BitsShiftRightZ = Value \ &H200000
3345 ' Case 22&: BitsShiftRightZ = Value \ &H400000
3346 ' Case 23&: BitsShiftRightZ = Value \ &H800000
3347 ' Case 24&: BitsShiftRightZ = Value \ &H1000000
3348 ' Case 25&: BitsShiftRightZ = Value \ &H2000000
3349 ' Case 26&: BitsShiftRightZ = Value \ &H4000000
3350 ' Case 27&: BitsShiftRightZ = Value \ &H8000000
3351 ' Case 28&: BitsShiftRightZ = Value \ &H10000000
3352 ' Case 29&: BitsShiftRightZ = Value \ &H20000000
3353 ' Case 30&: BitsShiftRightZ = Value \ &H40000000
3354 ' Case 31&: BitsShiftRightZ = &H0&
3355 ' End Select
3356 '
3357 ' End If
3358 'End Function
3359
3360
3361
3362
3363 'Public Function PopCount(ByVal x As Long) As Long
3364 ' ' for positive values only
3365 ' Debug.Assert x >= 0
3366 '
3367 ' PopCount = 0

```

```

3368 ' Do While x > 0
3369 '   PopCount = PopCount + 1: x = x And (x - 1)
3370 ' Loop
3371 'End Function
3372 '
3373 'Public Function And64(Op1 As TBit64, Op2 As TBit64) As TBit64
3374 ' And64.i0 = Op1.i0 And Op2.i0
3375 ' And64.i1 = Op1.i1 And Op2.i1
3376 ' And64.i2 = Op1.i2 And Op2.i2
3377 ' And64.i3 = Op1.i3 And Op2.i3
3378 'End Function
3379 '
3380 'Public Function Or64(Op1 As TBit64, Op2 As TBit64) As TBit64
3381 ' Or64.i0 = Op1.i0 Or Op2.i0
3382 ' Or64.i1 = Op1.i1 Or Op2.i1
3383 ' Or64.i2 = Op1.i2 Or Op2.i2
3384 ' Or64.i3 = Op1.i3 Or Op2.i3
3385 'End Function
3386 '
3387 'Public Function Xor64(Op1 As TBit64, Op2 As TBit64) As TBit64
3388 ' Xor64.i0 = Op1.i0 Xor Op2.i0
3389 ' Xor64.i1 = Op1.i1 Xor Op2.i1
3390 ' Xor64.i2 = Op1.i2 Xor Op2.i2
3391 ' Xor64.i3 = Op1.i3 Xor Op2.i3
3392 'End Function
3393 '
3394 'Public Sub Clear64(Op1 As TBit64)
3395 ' Op1.i0 = 0
3396 ' Op1.i1 = 0
3397 ' Op1.i2 = 0
3398 ' Op1.i3 = 0
3399 'End Sub
3400 '
3401 'Public Function PopCnt64(Op1 As TBit64) As Long
3402 ' PopCnt64 = PopCount(Op1.i0) + PopCount(Op1.i1) + PopCount(Op1.i2) + PopCount(Op1.i3)
3403 'End Function
3404 '
3405 Public Sub SetMove(m1 As TMOVE, m2 As TMOVE)
3406 ' assign m2 to m1. 3x faster than Move1 = Move 2 !
3407 With m1
3408   .Captured = m2.Captured: .CapturedNumber = m2.CapturedNumber: .Castle = m2.Castle: .
   EnPassant = m2.EnPassant
3409   .From = m2.From: .IsChecking = m2.IsChecking: .IsLegal = m2.IsLegal: .OrderValue =
   m2.OrderValue: .Piece = m2.Piece
3410   .Promoted = m2.Promoted: .SeeValue = m2.SeeValue: .Target = m2.Target
3411 End With
3412 End Sub
3413
3414 Public Sub SwapMove(m1 As TMOVE, m2 As TMOVE)
3415   Dim l As Long, b As Boolean
3416   With m2
3417     l = .Captured: .Captured = m1.Captured: m1.Captured = l
3418     l = .CapturedNumber: .CapturedNumber = m1.CapturedNumber: m1.CapturedNumber = l
3419     l = .Castle: .Castle = m1.Castle: m1.Castle = l
3420     l = .EnPassant: .EnPassant = m1.EnPassant: m1.EnPassant = l
3421     l = .From: .From = m1.From: m1.From = l
3422     b = .IsChecking: .IsChecking = m1.IsChecking: m1.IsChecking = b
3423     b = .IsLegal: .IsLegal = m1.IsLegal: m1.IsLegal = b
3424     l = .OrderValue: .OrderValue = m1.OrderValue: m1.OrderValue = l
3425     l = .Piece: .Piece = m1.Piece: m1.Piece = l
3426     l = .Promoted: .Promoted = m1.Promoted: m1.Promoted = l
3427     l = .SeeValue: .SeeValue = m1.SeeValue: m1.SeeValue = l
3428     l = .Target: .Target = m1.Target: m1.Target = l
3429   End With
3430 End Sub
3431
3432 Public Sub ClearMove(m1 As TMOVE)
3433   ' 2x faster than Move1 = EmptyMove !

```



```

3434     With m1
3435         .From = 0: .Target = 0: .Piece = NO_PIECE: .Castle = NO_CASTLE: .Promoted = 0: .
Captured = NO_PIECE: .CapturedNumber = 0
3436         .EnPassant = 0: .IsChecking = False: .IsLegal = False: .OrderValue = 0: .SeeValue
= VALUE_NONE
3437     End With
3438 End Sub
3439
3440 'Public Function WCastlingRight() As Long
3441 ' If Moved(WKingLoc) = 0 Then
3442 '     If Moved(SQ_H1) = 0 Then WCastlingRight = 1
3443 '     If Moved(SQ_A1) = 0 Then WCastlingRight = WCastlingRight Or 2
3444 ' Else
3445 '     WCastlingRight = 0
3446 ' End If
3447 'End Function
3448
3449 'Public Function BCastlingRight() As Long
3450 ' If Moved(BKingLoc) = 0 Then
3451 '     If Moved(SQ_H8) = 0 Then BCastlingRight = 1
3452 '     If Moved(SQ_A8) = 0 Then BCastlingRight = BCastlingRight Or 2
3453 ' Else
3454 '     BCastlingRight = 0
3455 ' End If
3456 'End Function
3457
3458
3459 Attribute VB_Name = "basBook"
3460 '=====
3461 '= basBook:
3462 '= chess opening book functions
3463 '=====
3464 Option Explicit
3465 Public bUseBook           As Boolean
3466
3467 Public UCIBook() As String
3468 Public UCIBookMax As Long, UCIBookCnt As Long
3469 Public BookMovePossible As Boolean
3470
3471 '-----
3472 'ChooseBookMove()
3473 '-----
3474 Public Function ChooseBookMove() As TMOVE
3475     ' game has to be started from startup position, FEN/EPD loaded position not supported
3476     Dim i           As Long
3477     Dim sPossibleMove As String, sCoordMove As String
3478     Dim iNumMoves    As Long
3479
3480     SetMove ChooseBookMove, EmptyMove
3481
3482     sPossibleMove = GetUCIBookMove()
3483
3484     ' check for legal move
3485     Ply = 1
3486     GenerateMoves Ply, False, iNumMoves
3487
3488     For i = 0 To iNumMoves - 1
3489         sCoordMove = CompToCoord(Moves(Ply, i)) ' format "e4d5"
3490         If sCoordMove = sPossibleMove Then
3491             SetMove ChooseBookMove, Moves(Ply, i)
3492             Exit Function
3493         End If
3494     Next
3495
3496 End Function
3497
3498
3499 '-----

```

```

3500 'InitBook()
3501 '-----
3502 Public Function InitBook() As Boolean
3503     Static bInitBookDone As Boolean
3504     Static bUseBookOk As Boolean
3505     Dim sBookFile As String
3506
3507     If bInitBookDone Then ' read only once
3508         InitBook = bUseBookOk
3509         Exit Function
3510     End If
3511
3512     If pbMSEExcelRunning Then ' set in SetVBAPathes
3513         InitBook = ReadExcelBook()
3514     End If
3515     If Not InitBook Then
3516         sBookFile = ReadINISetting(USE_BOOK_KEY, "CB_BOOK.TXT")
3517         If pbIsOfficeMode And Trim(sBookFile) = "" Then
3518             ' Always use default book if not set in INI file
3519             sBookFile = "CB_BOOK.TXT"
3520         End If
3521         InitBook = ReadUCIBook(sBookFile)
3522     End If
3523     bUseBookOk = InitBook
3524     bInitBookDone = True
3525 End Function
3526
3527 '-----
3528 ' MS Excel: read book from internal worksheet
3529 '-----
3530 Public Function ReadExcelBook() As Boolean
3531     On Error GoTo lblError
3532
3533     #If VBA_MODE = 1 Then
3534         ' read opening book lines from Excel worksheet CB_BOOK
3535         Dim Sheet As Object, lNum As Long, i As Long, sInp As String
3536
3537         Set Sheet = ActiveWorkbook.Sheets("CB_BOOK")
3538
3539         ReDim UCIBook(0)
3540         UCIBookMax = 0: UCIBookCnt = 0
3541
3542         With Sheet
3543             lNum = .Cells(.Rows.Count, 1).End(xlUp).Row
3544             For i = 1 To lNum
3545                 sInp = Trim$(.Cells(i, 1))
3546                 If Left(sInp, 1) <> "#" And sInp <> "" Then '#: comment line
3547                     UCIBookCnt = UCIBookCnt + 1: If UCIBookCnt > UCIBookMax Then UCIBookMax =
3548                         UCIBookMax + 1000: ReDim Preserve UCIBook(UCIBookMax)
3549                     UCIBook(UCIBookCnt) = sInp
3550                 End If
3551             Next i
3552         End With 'sheet
3553
3554         ReadExcelBook = (UCIBookCnt > 0)
3555         If ReadExcelBook Then
3556             SendCommand "opening book found in Excel sheet CB_BOOK. Lines found:" &
3557                 UCIBookCnt
3558         End If
3559     #End If
3560     lblError:
3561     ReadExcelBook = False
3562 End Function
3563
3564 Public Function GetUCIGameLine() As String
3565     Dim i As Long, h As Long, s As String, MoveCnt As Long, Cnt As Long

```



```

3566
3567 GetUCIGameLine = ""
3568
3569 Cnt = GameMovesCnt
3570 If Cnt = 0 Then Exit Function
3571 s = "": MoveCnt = 0
3572
3573 For i = 1 To Cnt Step 2
3574     MoveCnt = MoveCnt + 1
3575     s = s & CompToCoord(arGameMoves(i))
3576     If i + 1 <= Cnt Then s = s & " " & CompToCoord(arGameMoves(i + 1)) & " "
3577 Next i
3578 GetUCIGameLine = Trim$(s)
3579
3580 End Function
3581
3582 Public Function ReadUCIBook(isFile As String) As Boolean
3583 'Read PGN File
3584 Dim h As Long, sInp As String, sBookFile As String
3585
3586 ReadUCIBook = False
3587
3588 h = 10 'FreeFile()
3589 ReDim UCIBook(0)
3590 UCIBookMax = 0: UCIBookCnt = 0
3591
3592 sBookFile = psEnginePath & "\" & isFile
3593
3594 On Error GoTo lblError
3595 If Dir(sBookFile) = "" Or isFile = "" Then
3596     Dim sDefault As String
3597     If pbIsOfficeMode Then sDefault = "1" Else sDefault = "0"
3598
3599     If ReadINISetting("USE_INTERNAL_BOOK", sDefault) = "1" Then
3600         InitInternalBook
3601         ReadUCIBook = True
3602         If pbIsOfficeMode Then
3603             SendCommand "internal opening book active"
3604         ElseIf UCIMode Then
3605             SendCommand "info string internal opening book active"
3606         End If
3607     End If
3608     Exit Function
3609 End If
3610
3611 Open sBookFile For Input As #h
3612
3613 Do Until EOF(h)
3614     Line Input #h, sInp: sInp = Trim(sInp)
3615     If Left(sInp, 1) <> "#" And sInp <> "" Then '#: comment line
3616         UCIBookCnt = UCIBookCnt + 1: If UCIBookCnt > UCIBookMax Then UCIBookMax =
3617             UCIBookMax + 1000: ReDim Preserve UCIBook(UCIBookMax)
3618         UCIBook(UCIBookCnt) = sInp
3619     End If
3620 Loop
3621 ReadUCIBook = (UCIBookCnt > 0)
3622 If ReadUCIBook Then
3623     If pbIsOfficeMode Then
3624         SendCommand "opening book found: " & isFile
3625     ElseIf UCIMode Then
3626         SendCommand "info string opening book found: " & isFile
3627     End If
3628 End If
3629
3630 Close #h
3631 Exit Function
3632 lblError:
3633 ReadUCIBook = False

```

```

3633 End Function
3634
3635 Public Function GetUCIBookMove() As String
3636 '--- input file ist sorted, lowercase, UCi format e4d5 (not e4xd5 or Bxd5)
3637 '---- create book file from PGN
3638 'pgn-extract.exe -Wuci --notags --noresults -C -N -V --output book.txt test.pgn
3639 'sort out.txt /o book.txt
3640
3641 Dim sUCIGame As String, sBookLine As String, r As Double
3642 Dim i As Long, lStart As Long, lEnd As Long, lUCILen As Long, x As Long
3643
3644 GetUCIBookMove = ""
3645 sUCIGame = GetUCIGameLine()
3646
3647 lUCILen = Len(sUCIGame)
3648
3649 If lUCILen >= 4 Then
3650     lStart = 0
3651     For i = 1 To UCIBookCnt
3652         If Left$(UCIBook(i), lUCILen) = sUCIGame Then
3653             lEnd = i: If lStart = 0 Then lStart = i
3654         End If
3655     Next
3656 Else
3657     ' first game move
3658     lStart = 1: lEnd = UCIBookCnt
3659 End If
3660
3661 ' get a random move in the range found
3662 Randomize
3663 r = Rnd
3664 If lEnd > lStart Then lStart = lStart + Int(((lEnd - lStart + 1) * r))
3665 sBookLine = Trim$(Mid$(UCIBook(lStart), lUCILen + 1))
3666 If Len(sBookLine) >= 4 Then
3667     sBookLine = Trim$(Left$(sBookLine, 4)) ' no promotion moves supported
3668     If Len(sBookLine) = 4 Then GetUCIBookMove = sBookLine
3669 End If
3670
3671 'Debug.Print lStart, lEnd; r, GetUCIBookMove
3672 End Function
3673
3674 Public Function InitInternalBook()
3675     ' Read internal book, just for fun - if external book is missing
3676     Dim BookArr As Variant ' extra array because Variant type needed for ARRAY()
3677     Dim i As Long
3678     BookArr = Array("a2a3 g7g6 g2g3 f8g7 f1g2", "a2a3 g8f6 g1f3 d7d5 d2d4", "b1c3 c7c5
3679         d2d4 c5d4 d1d4", "b1c3 c7c5 e2e3 g7g6 d2d4", "b1c3 d7d5 e2e4 d5d4 c3e2", _
3680         "b2b3 d7d5 c1b2 c8g4 g2g3", "b2b3 e7e5 c1b2 b8c6 e2e3", "c2c4 b8c6
3681         g2g3 e7e5 f1g2", "c2c4 c7c5 b1c3 b7b6 e2e3", "c2c4 c7c5 b1c3 b7b6
3682         e2e4", _
3683         "c2c4 e7e5 b1c3 b8c6 g2g3", "c2c4 e7e6 g1f3 g8f6 b2b3", "c2c4 e7e6
3684         g2g3 g8f6 f1g2", "c2c4 f7f5 b1c3 g8f6 d2d3", "c2c4 f7f5 b1c3 g8f6
3685         d2d4", _
3686         "c2c4 g8f6 b1c3 e7e6 g1f3", "d2d4 d7d5 c2c4 c7c6 b1c3", "d2d4 d7d5
3687         g1f3 g8f6 c2c4", "d2d4 d7d5 g1f3 g8f6 e2e3", "d2d4 d7d6 c1g5 b8d7
3688         e2e4", _
3689         "d2d4 d7d6 c1g5 f7f6 g5h4", "d2d4 d7d6 c1g5 g7g6 c2c4", "d2d4 d7d6
3690         c2c3 g8f6 c1g5", "d2d4 d7d6 c2c4 e7e5 b1c3", "d2d4 d7d6 c2c4 f7f5
3691         g2g3", _
3692         "d2d4 d7d6 c2c4 g7g6 b1c3", "d2d4 d7d6 e2e4 c7c5 d4d5", "d2d4 d7d6
3693         e2e4 e7e5 g1f3", "d2d4 d7d6 e2e4 g7g6 b1c3", "d2d4 d7d6 e2e4 g7g6
3694         c2c4", _
3695         "d2d4 d7d6 e2e4 g8f6 b1c3", "d2d4 g8f6 c2c4 e7e6 g1f3", "d2d4 g8f6
3696         g1f3 g7g6 g2g3", "e2e4 b7b6 g2g3 c8b7 f1g2", "e2e4 b8c6 b1c3 e7e5
3697         f1c4", _
3698         "e2e4 b8c6 d2d4 e7e5 g1f3", "e2e4 b8c6 f1b5 g8f6 d2d3", "e2e4 b8c6
3699         g1f3 d7d6 d2d4", "e2e4 c7c5 b1c3 a7a6 g2g4", "e2e4 c7c5 b1c3 b8c6
3700         d2d3", _

```

```

3686         "e2e4 c7c5 c2c3 e7e6 d2d4", "e2e4 c7c5 f2f4 d7d5 d2d3", "e2e4 c7c5
3687         f2f4 d7d5 e4d5", "e2e4 c7c5 g1f3 a7a6 b1c3", "e2e4 c7c5 g1f3 d8c7
3688         d2d4", _
3689         "e2e4 c7c5 g1f3 e7e6 b1c3", "e2e4 c7c6 g1f3 d7d5 e4d5", "e2e4 d7d6
3690         d2d4 g8f6 b1c3", "f2f4 b7b6 g1f3 c8b7 e2e3", "f2f4 c7c5 b2b3 g8f6
3691         c1b2", _
3692         "f2f4 d7d5 e2e3 g8f6 g1f3", "g1f3 c7c5 c2c3 g8f6 g2g3", "g1f3 c7c5
3693         c2c4 b7b6 b1c3", "g1f3 d7d5 c2c4 c7c6 g2g3", "g1f3 d7d5 c2c4 d5c4
3694         b1a3", _
3695         "g2g3 d7d5 f1g2 c7c6 g1f3", "g2g3 d7d5 f1g2 e7e5 c2c3", "g2g3 e7e5
3696         f1g2 d7d5 d2d3", "g2g3 g7g6 f1g2 f8g7 c2c4", "g2g3 g8f6 f1g2 e7e5
3697         d2d3")
3698
3699     UCIBookCnt = UBound(BookArr) + 1
3700     ReDim UCIBook(UCIBookCnt)
3701     For i = 1 To UCIBookCnt: UCIBook(i) = BookArr(i - 1): Next
3702
3703 End Function
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744

```

**Attribute** VB\_Name = "basChessBrainVB"  
 '=====  
 ' = basChessBrainVB:  
 ' = main program  
 ' =  
 ' = ChessBrainVB V4.00:  
 ' = by Roger Zuehlisdrorf (Copyright 2023)  
 ' = based on LarsenVB by Luca Dormio (<http://xoomer.virgilio.it/ludormio/download.htm>) and Faile by Adrien M.  
 Regimbald  
 ' = and Stockfish by Marco Costalba, Joona Kiiski, Gary Linscott, Tord Romstad  
 ' = start of program  
 ' = init engine  
 '=====  
**Option Explicit**  
 'DEBUGMODE: console input via VB form. Else: Winbord interface  
**Public** DebugMode **As** **Boolean**  
 'simulate standard input  
 'set in frmDebugMain.cmdFakeInput\_Click  
**Public** FakeInputState **As** **Boolean**  
**Public** FakeInput **As** **String**  
**Public** MatchInfo **As** TMatchInfo  
**Public** bXBoardMode **As** **Boolean**  
**Public** iXBoardProtoVer **As** **Long** ' winboard protocol version  
**Public** bForceMode **As** **Boolean**  
**Public** bPostMode **As** **Boolean**  
**Public** bAnalyzeMode **As** **Boolean**  
**Public** bExitReceived **As** **Boolean**  
**Public** bAllowPonder **As** **Boolean**  
**Public** ThisApp **As** **Object**  
**Public** psAppName **As** **String**  
**Public** Moves(MAX\_DEPTH, MAX\_MOVES) **As** TMOVE ' Generated moves [ply,Move]  
**Public** QuietsSearched(MAX\_DEPTH, 65) **As** TMOVE ' Quiet moves for pruning conditions  
**Public** CapturesSearched(MAX\_DEPTH, 32) **As** TMOVE ' Quiet moves for pruning conditions  
**Public** MovePickerDat(MAX\_DEPTH) **As** TMovePicker  
**Public** GameMovesCnt **As** **Long**  
**Public** arGameMoves(MAX\_GAME\_MOVES) **As** TMOVE  
**Public** GamePosHash(MAX\_GAME\_MOVES) **As** THashKey  
**Public** GUICheckIntervalNodes **As** **Long**  
**Public** MemoryMB **As** **Long** ' memory command  
**Public** UCIMode **As** **Boolean**  
**Public** pbMSEExcelRunning **As** **Boolean**

'-----  
 ' Main: Start of program ChessBrainVB -  
 '-----  
**Sub** Main()

```

3745 Dim sCmdList() As String
3746 Dim i As Long
3747
3748 '--- VBA_MODE constant is set in Excel/Word in VBAChessBrain project properties for conditional compiling
3749 #If VBA_MODE = 1 Then
3750     '--- MS-OFFICE VBA ---
3751     pbIsOfficeMode = True
3752     GUICheckIntervalNodes = 1000 ' nodes until next check for GUI commands
3753     SetVBAPathes
3754 #Else
3755     '--- VB6 ---
3756     pbIsOfficeMode = False
3757     pbMSExcelRunning = False
3758     GUICheckIntervalNodes = 5000
3759     psEnginePath = App.Path
3760     psAppName = App.EXENAME
3761 #End If
3762 DebugMode = CBool(ReadINISetting("DEBUGMODE", "0") <> "0")
3763 bWinboardTrace = CBool(ReadINISetting("COMMANDTRACE", "0") <> "0")
3764 bThreadTrace = CBool(ReadINISetting("THREADTRACE", "0") <> "0")
3765 bTimeTrace = CBool(ReadINISetting("TIMETRACE", "0") <> "0")
3766 bEGTbBaseTrace = CBool(ReadINISetting("TBBASE_TRACE", "0") <> "0")
3767 bWbPvInUciFormat = CBool(ReadINISetting("WB_PV_IN_UCI", "0") <> "0")
3768 InitTranslate
3769 'set main threadnum=-1
3770 SetThreads 1
3771 '
3772 '--- command line options
3773 '
3774 If Command$ <> "" Then
3775     sCmdList = Split(LCase(Command$))
3776
3777     For i = 0 To UBound(sCmdList)
3778         If bWinboardTrace Then WriteTrace "Command: " & sCmdList(i) & " " & Now()
3779         If Left$(Trim$(sCmdList(i)), 6) = "thread" Then
3780             #If VBA_MODE = 0 Then
3781                 ' Parameter for helper threads : "threat1" .. "threat8"
3782                 ThreadNum = Val("0" & Trim$(Mid$((Trim$(sCmdList(i))), 7)))
3783                 ThreadNum = GetMax(1, ThreadNum): NoOfThreads = ThreadNum + 1
3784                 If bThreadTrace Then WriteTrace "Command: ThreadNum = " & ThreadNum & " / "
3785                 App.Title = "ChessBrainVB_T" & Trim$(CStr(ThreadNum))
3786             #End If
3787         Else
3788
3789             Select Case Trim$(sCmdList(i))
3790                 Case "xboard", "/xboard", "-xboard"
3791                     bXBoardMode = True
3792                 Case "log", "/log", "-log"
3793                     bLogMode = True
3794                     bLogPV = CBool(Val(ReadINISetting(LOG_PV_KEY, "0")))
3795                 Case "/?", "-?", "?"
3796                     MsgBox "arguments: -xboard , -log"
3797                 Case ""
3798                 Case Else
3799                     MsgBox "Wrong argument " & vbLf & Command$, vbExclamation
3800             End Select
3801
3802         End If
3803     Next
3804 End If
3805
3806 If ThreadNum <= 0 Then
3807     OpenCommHandles 'enable GUI communication > main thread
3808     SendCommand "ChessBrainVB by Roger Zuehlisdrorf"
3809 End If
3810
3811 #If VBA_MODE <> 0 Then

```

```

3812     InitEngine
3813     frmChessX.Show
3814     Exit Sub
3815 #End If
3816 #If DEBUG_MODE <> 0 Then
3817     ' Simulate Xboard using input of debug form
3818     bXBoardMode = True
3819     InitEngine
3820     If ThreadNum <= 0 Then
3821         frmDebugMain.Show '--- Show debug form
3822     End If
3823
3824     '-----
3825     MainLoop '--- Wait for winboard commands from debug form
3826     '-----
3827
3828     Exit Sub
3829 #End If
3830 #If DEBUG_MODE = 0 And VBA_MODE = 0 Then
3831     If Not bXBoardMode And Trim(ReadINISetting("WINBOARD", "")) = "" Then
3832         bXBoardMode = CBool(Trim(ReadINISetting("XBOARD_MODE", "1")) = "1")
3833     End If
3834     If bXBoardMode Then
3835         '-----
3836         '--- normal winboard/uci mode without form
3837         '-----
3838         InitEngine
3839         '----->>> loop for new external commands <<<-----
3840         MainLoop '--- Wait for winboard/ uci commands
3841         '-----<<< loop for new external commands <<<-----
3842         '<<<
3843     Else
3844         ' init winboard path
3845         frmMain.Show '--- Show main form
3846     End If
3847 #End If
3848 End Sub
3849
3850 '-----
3851 ' InitEngine()
3852 '-----
3853 Public Sub InitEngine()
3854     iXBoardProtoVer = 1
3855     '-----
3856     '--- init arrays
3857     '-----
3858     Erase PVLength()
3859     Erase PV()
3860     Erase History()
3861     Erase CaptureHistory()
3862     Erase CounterMove()
3863     Erase ContinuationHistory()
3864     'InitContHist ' if filled with specific start values
3865
3866     Erase Pieces()
3867     Erase Squares()
3868     Erase Killer()
3869     Erase Board()
3870     Erase Moved()
3871     Erase MovesList()
3872     Erase arGameMoves()
3873     Erase GamePosHash()
3874
3875     InitPieceColor
3876
3877     '-----
3878     '--- move offsets ---
3879     '-----

```

```

3880 '0-3: Orthogonal (Queen+Rook), 4-7=diagonal (Queen+Bishop)
3881 ReadIntArr DirectionOffset(), 10, -10, 1, -1, 11, -11, 9, -9
3882 ReadIntArr KnightOffsets(), 8, 19, 21, 12, -8, -19, -21, -12
3883 ReadIntArr BishopOffsets(), 9, 11, -9, -11
3884 ReadIntArr RookOffsets(), 1, -1, 10, -10
3885 OppositeDir(1) = -1: OppositeDir(-1) = 1: OppositeDir(10) = -10: OppositeDir(-10) =
10
3886 OppositeDir(11) = -11: OppositeDir(-11) = 11: OppositeDir(9) = -9: OppositeDir(-9) =
9
3887
3888 ReadIntArr WPromotions(), 0, WQUEEN, WROOK, WKNIGHT, WBISHOP
3889 ReadIntArr BPromotions(), 0, BQUEEN, BROOK, BKNIGHT, BBISHOP
3890 ReadIntArr PieceType, 0, PT_PAWN, PT_PAWN, PT_KNIGHT, PT_KNIGHT, PT_BISHOP,
PT_BISHOP, PT_ROOK, PT_ROOK, PT_QUEEN, PT_QUEEN, PT_KING, PT_KING, NO_PIECE_TYPE,
PT_PAWN, PT_PAWN
3891 InitRankFile ' must be before InitMaxDistance
3892 InitBoardColors
3893 InitMaxDistance
3894 InitSqBetween
3895 InitSameXRay
3896 InitAttackBitCnt
3897 bAllowPonder = False
3898
3899 ' setup empty move
3900 With EmptyMove
3901 .From = 0: .Target = 0: .Piece = NO_PIECE: .Castle = NO_CASTLE: .Promoted = 0: .
Captured = NO_PIECE: .CapturedNumber = 0
3902 .EnPassant = 0: .IsChecking = False: .IsLegal = False: .OrderValue = 0: .SeeValue
= VALUE_NONE
3903 End With
3904
3905 '-----
3906 '--- startup board
3907 '-----
3908 ReadIntArr StartupBoard(), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, WROOK, WKNIGHT, WBISHOP, WQUEEN, WKING, WBISHOP, WKNIGHT, WROOK, 0, 0, WPAWN,
WPAWN, WPAWN, WPAWN, WPAWN, WPAWN, WPAWN, WPAWN, 0, 0, 13, 13, 13, 13, 13, 13, 13,
13, 0, 0, 13, 13, 13, 13, 13, 13, 13, 13, 0, 0, 13, 13, 13, 13, 13, 13, 13, 13, 0, 0
, 13, 13, 13, 13, 13, 13, 13, 13, 0, 0, BPAWN, BPAWN, BPAWN, BPAWN, BPAWN, BPAWN,
BPAWN, BPAWN, 0, 0, BROOK, BKNIGHT, BBISHOP, BQUEEN, BKING, BBISHOP, BKNIGHT, BROOK,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
3909
3910 '-----
3911 '--- Piece square table: bonus for piece position on board ---
3912 '-----
3913 '( FILE A-D: Pairs MG,EG : A(MG,EG),B(MG,EG),...
3914 '--- Pawn piece square table
3915 PSQT64 PsqtWP, PsqtBP, 0, 0, 0, 0, 0, 0, 0, 0, -11, 7, 6, -4, 7, 8, 3, -2, -18, -4,
-2, -5, 19, 5, 24, 4, -17, 3, -9, 3, 20, -8, 35, -3, -6, 8, 5, 9, 3, 7, 21, -6, -6,
8, -8, -5, -6, 2, -2, 4, -4, 3, 20, -9, -8, 1, -4, 18, 0, 0, 0, 0, 0, 0, 0, 0
3916 '--- Knight piece square table
3917 PSQT64 PsqtWN, PsqtBN, -161, -105, -96, -82, -80, -46, -73, -14, -83, -69, -43, -54,
-21, -17, -10, 9, -71, -50, -22, -39, 0, -7, 9, 28, -25, -41, 18, -25, 43, 6, 47,
38, -26, -46, 16, -25, 38, 3, 50, 40, -11, -54, 37, -38, 56, -7, 65, 27, -63, -65, -
19, -50, 5, -24, 14, 13, -195, -109, -67, -89, -42, -50, -29, -13
3918 '--- Bishop piece square table
3919 PSQT64 PsqtWB, PsqtBB, -44, -58, -13, -31, -25, -37, -34, -19, -20, -34, 20, -9, 12,
-14, 1, 4, -9, -23, 27, 0, 21, -3, 11, 16, -11, -26, 28, -3, 21, -5, 10, 16, -11, -
26, 27, -4, 16, -7, 9, 14, -17, -24, 16, -2, 12, 0, 2, 13, -23, -34, 17, -10, 6, -12
, -2, 6, -35, -55, -11, -32, -19, -36, -29, -17
3920 '--- Rook piece square table
3921 PSQT64 PsqtWR, PsqtBR, -25, 0, -16, 0, -16, 0, -9, 0, -21, 0, -8, 0, -3, 0, 0, 0, -
21, 0, -9, 0, -4, 0, 2, 0, -22, 0, -6, 0, -1, 0, 2, 0, -22, 0, -7, 0, 0, 0, 1, 0, -
21, 0, -7, 0, 0, 0, 2, 0, -12, 0, 4, 0, 8, 0, 12, 0, -23, 0, -15, 0, -11, 0, -5, 0
3922 '--- Queen piece square table
3923 PSQT64 PsqtWQ, PsqtBQ, 0, -71, -4, -56, -3, -42, -1, -29, -4, -56, 6, -30, 9, -21, 8
, -5, -2, -39, 6, -17, 9, -8, 9, 5, -1, -29, 8, -5, 10, 9, 7, 19, -3, -27, 9, -5, 8,
10, 7, 21, -2, -40, 6, -16, 8, -10, 10, 3, -2, -55, 7, -30, 7, -21, 6, -6, -1, -74,

```

```

-4, -55, -1, -43, 0, -30
3924 '--- King piece square table
3925 PSQT64 PsqtWK, PsqtBK, 267, 0, 320, 48, 270, 75, 195, 84, 264, 43, 304, 92, 238, 143
, 180, 132, 200, 83, 245, 138, 176, 167, 110, 165, 177, 106, 185, 169, 148, 169, 110
, 179, 149, 108, 177, 163, 115, 200, 66, 203, 118, 95, 159, 155, 84, 176, 41, 174,
87, 50, 128, 99, 63, 122, 20, 139, 63, 9, 88, 55, 47, 80, 0, 90
3926 FillPieceSquareVal
3927 '--- Mobility bonus for number of attacked squares not occupied by friendly pieces (pairs: MG,EG, MG,EG)
3928 ' Knights
3929 ReadScoreArr MobilityN, -75, -76, -56, -54, -9, -26, -2, -10, 6, 5, 15, 11, 22, 26,
30, 28, 36, 29
3930 ' Bishops
3931 ReadScoreArr MobilityB, -48, -58, -21, -19, 16, -2, 26, 12, 37, 22, 51, 42, 54, 54,
63, 58, 65, 63, 71, 70, 79, 74, 81, 86, 92, 90, 97, 94
3932 ' Rooks
3933 ReadScoreArr MobilityR, -56, -78, -25, -18, -11, 26, -5, 55, -4, 70, -1, 81, 8, 109,
14, 120, 21, 128, 23, 143, 31, 154, 32, 160, 43, 165, 49, 168, 59, 169
3934 ' Queens
3935 ReadScoreArr MobilityQ, -40, -35, -25, -12, 2, 7, 4, 19, 14, 37, 24, 55, 25, 62, 40,
76, 43, 79, 47, 87, 54, 94, 56, 102, 60, 111, 70, 116, 72, 118, 73, 122, 75, 128,
77, 130, 85, 133, 94, 136, 99, 140, 108, 157, 112, 158, 113, 161, 118, 174, 119, 177
, 123, 191, 128, 199
3936 'SF6: Threat by pawn (pairs MG/EG: NOPIECE,PAWN,KNIGHT (176,139), BISHOP, ROOK, QUEEN
3937 'SF6: Outpost (Pair MG/EG)[0, 1=supported by pawn]
3938 ReadScoreArr ReachableOutpostKnight, 22, 6, 36, 12
3939 ReadScoreArr ReachableOutpostBishop, 9, 2, 15, 5
3940 ReadScoreArr OutpostBonusKnight, 44, 12, 66, 18
3941 ReadScoreArr OutpostBonusBishop, 18, 4, 28, 8
3942 'SF6: King Attack Weights by attacker { 0, 0, 7, 5, 4, 1 } NO_PIECE_TYPE, PAWN, KNIGHT, BISHOP, ROOK,
QUEEN, KING,
3943 ' SF values not clear: why queen is 1 and knight is 7 !? More attack fields in total for queen?
3944 KingAttackWeights(PT_PAWN) = 5: KingAttackWeights(PT_KNIGHT) = 80: KingAttackWeights
(PT_BISHOP) = 56: KingAttackWeights(PT_ROOK) = 45: KingAttackWeights(PT_QUEEN) = 12
3945 ' Pawn eval
3946 ' Isolated pawn penalty by opposed flag
3947 ReadScoreArr IsolatedPenalty(), 27, 30, 13, 18
3948 ReadScoreArr BackwardPenalty(), 40, 26, 24, 12 ' not opposed / opposed
3949 SetScoreVal DoubledPenalty, 18, 38
3950 ReadScoreArr LeverBonus(), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17, 16, 33, 32, 0, 0, 0, 0
3951 ReadIntArr PassedDanger(), 0, 0, 0, 0, 3, 6, 12, 21
3952 ReadScoreArr PassedPawnRankBonus(), 0, 0, 0, 0, 7, 10, -12, 26, 3, 31, 42, 63, 178,
167, 279, 244
3953 ReadScoreArr PassedPawnFileBonus(), 0, 0, 17, 3, 0, 10, 1, -23, -16, -20, -
-17, -8, 3, -1, -8, 4, 17, 9
3954
3955 ReadScoreArr KingProtector(), 0, 0, 0, 0, -3, -5, -4, -3, -3, 0, -1, 1 ' for N,B,R,Q
3956 ReadIntArr QueenMinorsImbalance(), 31, -8, -15, -25, -5
3957 ReadIntArr CaptPruneMargin(), 0, -238, -262, -244, -252, -241, -228
3958 ' King safety eval
3959 ' Weakness of our pawn shelter in front of the king by [distance from edge][rank]
3960 ReadIntArr2 ShelterWeakness(), 1, 0, 100, 10, 46, 82, 87, 86, 98 ' 1 = ArrIndex, 0: fill
Array(0)
3961 ReadIntArr2 ShelterWeakness(), 2, 0, 116, 4, 28, 87, 94, 108, 104
3962 ReadIntArr2 ShelterWeakness(), 3, 0, 109, 1, 59, 87, 62, 91, 116
3963 ReadIntArr2 ShelterWeakness(), 4, 0, 75, 12, 43, 59, 90, 84, 112
3964 ' Danger of enemy pawns moving toward our king by [type][distance from edge][rank]
3965 ' BlockedByKing
3966 ReadIntArr3 StormDanger(), 1, 1, 0, 0, -290, -274, 57, 41
3967 ReadIntArr3 StormDanger(), 1, 2, 0, 0, 60, 144, 39, 13
3968 ReadIntArr3 StormDanger(), 1, 3, 0, 0, 65, 141, 41, 34
3969 ReadIntArr3 StormDanger(), 1, 4, 0, 0, 53, 127, 56, 14
3970 ' Unopposed
3971 ReadIntArr3 StormDanger(), 2, 1, 0, 4, 73, 132, 46, 31
3972 ReadIntArr3 StormDanger(), 2, 2, 0, 1, 64, 143, 26, 13
3973 ReadIntArr3 StormDanger(), 2, 3, 0, 1, 47, 110, 44, 24
3974 ReadIntArr3 StormDanger(), 2, 4, 0, 0, 72, 127, 50, 31
3975 ' BlockedByPawn
3976 ReadIntArr3 StormDanger(), 3, 1, 0, 0, 0, 79, 23, 1
3977 ReadIntArr3 StormDanger(), 3, 2, 0, 0, 0, 148, 27, 2

```



[illegible]



```

4041         End If
4042     End If
4043     DoEvents
4044     If ThreadNum > 0 Then CheckThreadTermination True
4045 Loop
4046
4047 End Sub
4048
4049 '-----
4050 ' ParseCommand() - parse winboard input
4051 '
4052 ' a command list like "xboard\nnew\nrandom\nlevel 40 5 0\nhard" is splitted
4053 '-----
4054 Public Sub ParseCommand(ByVal sCommand As String)
4055     Dim bLegalInput As Boolean
4056     Dim i As Long, c As Long, x As Long, s As String, sSearch As String
4057     Dim PlayerMove As TMOVE, sCoordMove As String
4058     Dim iNumMoves As Long
4059     Dim sCurrentCmd As String
4060     Dim sCmdList() As String
4061     Dim sInput() As String
4062     Dim Hashkey As THashKey
4063     If Trim$(sCommand) = "" Then Exit Sub
4064     sCommand = Replace(sCommand, vbCr, vbLf) 'Fix per DDInterfaceEngine:
4065     If Right$(sCommand, 1) <> vbLf Then sCommand = sCommand & vbLf
4066     sCmdList = Split(sCommand, vbLf)
4067
4068     For c = 0 To UBound(sCmdList) - 1 'ignore vbLf
4069         sCurrentCmd = sCmdList(c)
4070         If sCurrentCmd = "" Then GoTo NextCmd
4071         If bWinboardTrace Then WriteTrace "Command: " & sCurrentCmd & " " & Now()
4072         If Trim$(sCurrentCmd) = "uci" Then
4073             '--- send UCI options
4074             UCIMode = True
4075             #If VBA_MODE = 1 Then
4076                 SendCommand "id name ChessBrainVB" ' App object not defined
4077             #Else
4078                 SendCommand "id name ChessBrainVB V" & Trim(App.Major) & "." & Trim(App.Minor)
4079                 & Trim(App.Revision)
4080             #End If
4081             SendCommand ConvertID()
4082             SendCommand "option name Threads type spin default 1 min 1 max " & CStr(
4083                 MAX_THREADS)
4084             SendCommand "option name Hash type spin default 128 min 1 max " & CStr(
4085                 MAX_HASHSIZE MB)
4086             SendCommand "option name Clear Hash type button"
4087             ' SendCommand "option name SyzygyPieceSet type spin default 5 min 0 max 6"
4088             ' SendCommand "option name SyzygyPath type string default <empty>"
4089             ' SendCommand "option name SyzygyMaxPly type spin default 3 min 1 max 6"
4090             SendCommand "uciok"
4091             UCISyzygyPath = ""
4092             UCISyzygyMaxPieceSet = -1
4093             UCISyzygyMaxPly = -1
4094             GoTo NextCmd
4095         End If
4096         If UCIMode Then
4097             '--- get UCI command
4098             sCurrentCmd = Trim$(sCurrentCmd)
4099             If sCurrentCmd = "ucinewgame" Or sCurrentCmd = "position startpos" Then
4100                 If bWinboardTrace Then WriteTrace "UCI: " & sCurrentCmd & " " & Now()
4101                 InitGame
4102                 GoTo NextCmd
4103             ElseIf sCurrentCmd = "stop" Or sCurrentCmd = "ponderhit" Then
4104                 bForceMode = False
4105                 bTimeExit = True
4106                 GoTo NextCmd
4107             ElseIf sCurrentCmd = "quit" Then
4108                 ExitProgram

```

```

4106         End
4107     End If
4108     sSearch = "setoption name Hash value"
4109     If Left$(sCurrentCmd, Len(sSearch)) = sSearch Then
4110         ' UCI hash memory size
4111         MemoryMB = Val("0" & Val(Mid$(sCurrentCmd, Len(sSearch) + 1)))
4112         If bWinboardTrace Then WriteTrace "UCI: hash memory size: " & sCurrentCmd & "
4113             " & Now()
4114         GoTo NextCmd
4115     End If
4116     sSearch = "setoption name Threads value"
4117     If Left$(sCurrentCmd, Len(sSearch)) = sSearch Then
4118         ' number of threads/cores
4119         If Not pbIsOfficeMode Then
4120             If CBool(ReadINISetting("THREADS_IGNORE_GUI", "0") = "0") Then
4121                 x = Val("0" & Val(Mid$(sCurrentCmd, Len(sSearch) + 1)))
4122                 SetThreads x
4123                 If bThreadTrace Then WriteTrace "Command:" & LCase(Command$)
4124             End If
4125         End If
4126         If bWinboardTrace Then WriteTrace "UCI: Threads: " & sCurrentCmd & " " & Now()
4127         GoTo NextCmd
4128     End If
4129     sSearch = "setoption name Contempt value"
4130     If Left$(sCurrentCmd, Len(sSearch)) = sSearch Then
4131         ' contempt score in centi pawns for draw
4132         x = Val("0" & Val(Mid$(sCurrentCmd, Len(sSearch) + 1)))
4133         GoTo NextCmd
4134     End If
4135     sSearch = "setoption name Clear Hash"
4136     If Left$(sCurrentCmd, Len(sSearch)) = sSearch Then
4137         If NoOfThreads < 2 Then InitHash
4138         GoTo NextCmd
4139     End If
4140     sSearch = "setoption name SyzygyPieceSet value"
4141     If Left$(sCurrentCmd, Len(sSearch)) = sSearch Then
4142         x = Val("0" & Val(Mid$(sCurrentCmd, Len(sSearch) + 1)))
4143         UCISyzygyMaxPieceSet = x
4144         If bEGTbBaseTrace Then WriteTrace "UCI SyzygyPieceSet= " & x
4145         GoTo NextCmd
4146     End If
4147     sSearch = "setoption name SyzygyPath value"
4148     If Left$(sCurrentCmd, Len(sSearch)) = sSearch Then
4149         s = Trim$(Mid$(sCurrentCmd, Len(sSearch) + 1))
4150         If Right$(s, 1) = "\" Then s = Left$(s, Len(s) - 1) ' Remove right \
4151         UCISyzygyPath = s
4152         If bEGTbBaseTrace Then WriteTrace "UCI SyzygyPath= " & s
4153         InitTableBases
4154         If EGTBasesEnabled Then SendCommand "info string Tablebases found"
4155         GoTo NextCmd
4156     End If
4157     sSearch = "setoption name SyzygyMaxPly value"
4158     If Left$(sCurrentCmd, Len(sSearch)) = sSearch Then
4159         x = Val("0" & Val(Mid$(sCurrentCmd, Len(sSearch) + 1)))
4160         UCISyzygyMaxPly = x
4161         If bEGTbBaseTrace Then WriteTrace "UCI UCISyzygyMaxPly= " & x
4162         GoTo NextCmd
4163     End If
4164     sSearch = "setoption name Ponder"
4165     If Left$(sCurrentCmd, Len(sSearch)) = sSearch Then
4166         ' Ponder: ignore until implemented
4167         GoTo NextCmd
4168     End If
4169     If Left$(sCurrentCmd, Len("isready")) = "isready" Then
4170         'DoEvents
4171         If CBool(ReadINISetting("THREADS_IGNORE_GUI", "0") = "0") Then
4172             SendCommand "info string " & CStr(NoOfThreads) & IIf(NoOfThreads = 1, "
4173                 core", " cores")

```

```

4172 Else
4173     SendCommand "info string " & CStr(NoOfThreads) & IIf(NoOfThreads = 1, "
        core", " cores (set in INI file)")
4174 End If
4175 If bWinboardTrace Then WriteTrace "UCI: " & sCurrentCmd & " " & Now()
4176 SendCommand "readyok"
4177 GoTo NextCmd
4178 End If
4179 If Left$(sCurrentCmd, Len("position")) = "position" Then
4180     ' position setup > this command is called by ARENA for every move in game
4181     ' a) position startpos moves <move1> <move2>...
4182     ' b) position fen <FEN> moves <move1> <move2>...
4183     UCIPositionSetup sCurrentCmd
4184     GoTo NextCmd
4185 End If
4186 If Left$(sCurrentCmd, Len("go")) = "go" Then
4187     ' go command
4188     ' go <time settings>
4189     ' sample: go wtime 120000 btime 120000 winc 0 binc 0 movestogo 32
4190     If bWinboardTrace Then WriteTrace "UCI: " & sCurrentCmd & " " & Now()
4191     bCompIsWhite = bWhiteToMove
4192     bPostMode = True
4193     UCISetTimeControl Trim$(Mid$(sCurrentCmd, 4))
4194     ' Start thinking!!!
4195     GoTo NextCmd
4196 End If
4197 End If '<<< UCIMode
4198 If sCurrentCmd = "." Then ' Show analyze info
4199     bExitReceived = False
4200     If bAnalyzeMode Then
4201         SendAnalyzeInfo
4202     End If
4203     GoTo NextCmd
4204 End If
4205 ' check first 4 characters: is this a move?
4206 ReDim sInput(4) ' also for special commands like "level"
4207 sInput(0) = Mid$(sCurrentCmd, 1, 1)
4208 sInput(1) = Mid$(sCurrentCmd, 2, 1)
4209 sInput(2) = Mid$(sCurrentCmd, 3, 1)
4210 sInput(3) = Mid$(sCurrentCmd, 4, 1)
4211 sInput(4) = Mid$(sCurrentCmd, 5, 1)
4212 '--- normal move like with 4 char: e2e4 ---
4213 If Not IsNumeric(sInput(0)) And IsNumeric(sInput(1)) And Not IsNumeric(sInput(2))
    And IsNumeric(sInput(3)) Then
4214     Ply = 0
4215     GenerateMoves Ply, False, iNumMoves
4216     PlayerMove.From = FileRev(sInput(0)) + RankRev(sInput(1))
4217     PlayerMove.Target = FileRev(sInput(2)) + RankRev(sInput(3))
4218
4219     ' legal move?
4220     For i = 0 To iNumMoves - 1
4221         sCoordMove = CompToCoord(Moves(Ply, i))
4222         If Trim(sCurrentCmd) = sCoordMove Then
4223             RemoveEpPiece
4224             MakeMove Moves(Ply, i)
4225             If CheckLegal(Moves(Ply, i)) Then
4226                 bLegalInput = True
4227                 PlayerMove.Captured = Moves(Ply, i).Captured
4228                 PlayerMove.Piece = Moves(Ply, i).Piece
4229                 PlayerMove.Promoted = Moves(Ply, i).Promoted
4230                 PlayerMove.EnPassant = Moves(Ply, i).EnPassant
4231                 PlayerMove.Castle = Moves(Ply, i).Castle
4232                 PlayerMove.CapturedNumber = Moves(Ply, i).CapturedNumber
4233             End If
4234             UnmakeMove Moves(Ply, i)
4235             ResetEpPiece
4236             If bLegalInput Then Exit For
4237         End If

```

```

4238     Next
4239
4240     If Not bLegalInput Then
4241         SendCommand "Illegal move: " & sCurrentCmd
4242         If bWinboardTrace Then LogWrite "Illegal move: " & sCoordMove
4243     Else
4244         'do game move
4245         PlayMove PlayerMove
4246         HashBoard Hashkey, EmptyMove
4247         If Is3xDraw(Hashkey, GameMovesCnt, 0) Then
4248             'Result = DRAW3REP_RESULT
4249             If bWinboardTrace Then LogWrite "ParseCommand: Return Draw3Rep"
4250             SendCommand "1/2-1/2 {Draw by repetition}"
4251         End If
4252         GameMovesAdd PlayerMove
4253         'LogWrite "move: " & sCoordMove
4254     End If
4255     GoTo NextCmd
4256 End If
4257 '--- not supported commands
4258 If sCurrentCmd = "xboard" Then GoTo NextCmd
4259 If sCurrentCmd = "random" Then GoTo NextCmd
4260 If Left$(sCurrentCmd, 4) = "name" Then
4261     MatchInfo.Opponent = Mid$(sCurrentCmd, 6)
4262     GoTo NextCmd
4263 End If
4264 If Left$(sCurrentCmd, 6) = "rating" Then
4265     MatchInfo.EngRating = Val(Mid$(sCurrentCmd, 8, 4))
4266     MatchInfo.OppRating = Val(Mid$(sCurrentCmd, 13, 4))
4267     GoTo NextCmd
4268 End If
4269 If sCurrentCmd = "computer" Then
4270     MatchInfo.OppComputer = True
4271     GoTo NextCmd
4272 End If
4273 If sCurrentCmd = "allseeks" Then
4274     SendCommand "tellics seek " & ReadINISetting("Seek1", "5 0 f")
4275     SendCommand "tellics seek " & ReadINISetting("Seek2", "15 5 f")
4276     GoTo NextCmd
4277 End If
4278 If sCurrentCmd = "hard" Or sCurrentCmd = "ponder" Then
4279     bAllowPonder = True
4280     If bWinboardTrace Then WriteTrace "ParseCommand: " & sCurrentCmd & " =>PonderOn"
4281     GoTo NextCmd
4282 End If
4283 If sCurrentCmd = "easy" Then
4284     If bWinboardTrace Then WriteTrace "ParseCommand: " & sCurrentCmd & "
=>PonderOff"
4285     bAllowPonder = False
4286     GoTo NextCmd
4287 End If
4288 If sCurrentCmd = "?" Then ' Stop Analyze
4289     bTimeExit = True
4290     bPostMode = False
4291     'bAnalyzeMode = False
4292     GoTo NextCmd
4293 End If
4294 '--- protocol xboard version 2 ---
4295 If Left$(sCurrentCmd, 8) = "protover" Then
4296     iXBoardProtoVer = Val(Mid$(sCurrentCmd, 10))
4297     If iXBoardProtoVer = 2 Then
4298         SendCommand "feature variants=""normal"" ping=1 setboard=1 analyze=1 smp=1
memory=1 myname=""ChessBrainVB"" done=1 "
4299     End If
4300     GoTo NextCmd
4301 End If
4302 If Left$(sCurrentCmd, 5) = "ping " Then
4303     SendCommand "pong " & Mid$(sCurrentCmd, 6)

```

```

4304         GoTo NextCmd
4305     End If
4306     If sCurrentCmd = "post" Then ' post PV
4307         bPostMode = True
4308         GoTo NextCmd
4309     End If
4310     If sCurrentCmd = "nopost" Then
4311         bPostMode = False
4312         GoTo NextCmd
4313     End If
4314     ' winboard time commands ( i.e. send from ARENA GUI )
4315     If Left$(sCurrentCmd, 4) = "time" Then ' time left for computer in 1/100 sec
4316         TimeLeft = Val(Mid$(sCurrentCmd, 5))
4317         TimeLeft = TimeLeft / 100#
4318         GoTo NextCmd
4319     End If
4320     If Left$(sCurrentCmd, 4) = "otim" Then ' time left for opponent
4321         OpponentTime = Val(Mid$(sCurrentCmd, 5))
4322         OpponentTime = OpponentTime / 100#
4323         GoTo NextCmd
4324     End If
4325     If Left$(sCurrentCmd, 5) = "level" Then
4326         ' time control
4327         ' level 0 2 12 : Game in 2 min + 12 sec/move
4328         ' level 40 0:30 0 : 40 moves in 30 min, final 0 = clock mode
4329         Erase sInput
4330         sInput = Split(sCurrentCmd)
4331         LevelMovesToTC = Val(sInput(1))
4332         MovesToTC = LevelMovesToTC - (GameMovesCnt + 1) \ 2
4333         i = InStr(1, sInput(2), ":")
4334         If i = 0 Then
4335             SecondsPerGame = Val(sInput(2)) * 60
4336         Else
4337             SecondsPerGame = Val(Left$(sInput(2), i - 1)) * 60
4338             SecondsPerGame = SecondsPerGame + Val(Right$(sInput(2), Len(sInput(2)) - i))
4339         End If
4340         TimeIncrement = Val(sInput(3))
4341         FixedTime = SecondsPerGame
4342         OpponentTime = TimeLeft
4343         FixedDepth = NO_FIXED_DEPTH
4344         FixedTime = 0
4345         GoTo NextCmd
4346     End If
4347     If Left$(sCurrentCmd, 3) = "st " Then
4348         ' fixed time for move
4349         MovesToTC = 1
4350         SecondsPerGame = Val(Mid$(sCurrentCmd, 3))
4351         FixedTime = SecondsPerGame
4352         TimeIncrement = 0
4353         TimeLeft = SecondsPerGame
4354         OpponentTime = TimeLeft
4355         FixedDepth = NO_FIXED_DEPTH
4356         GoTo NextCmd
4357     End If
4358     If Left$(sCurrentCmd, 3) = "sd " Then
4359         ' fixed depth (RootDepth)
4360         MovesToTC = 0
4361         SecondsPerGame = 0
4362         TimeIncrement = 0
4363         FixedTime = 0
4364         TimeLeft = SecondsPerGame
4365         OpponentTime = TimeLeft
4366         FixedDepth = Val(Mid$(sCurrentCmd, 3))
4367         GoTo NextCmd
4368     End If
4369     If Left$(sCurrentCmd, 6) = "cores " Then
4370         If bThreadTrace Then WriteTrace "Command:" & LCase(Command$)
4371         If Not pbIsOfficeMode Then

```

```

4372         If CBool(ReadINISetting("THREADS IGNORE_GUI", "0") = "0") Then
4373             x = Val("0" & Val(Mid$(sCurrentCmd, 7)))
4374             SetThreads x
4375         End If
4376     End If
4377 End If
4378 If Left$(sCurrentCmd, 7) = "memory " Then
4379     MemoryMB = Val("0" & Val(Mid$(sCurrentCmd, 8)))
4380 End If
4381 '
4382 '--- critical commands if pondering
4383 '
4384 If Left$(sCurrentCmd, 8) = "setboard" Then
4385     ReadEPD Mid$(sCurrentCmd, 10)
4386     If DebugMode Then
4387         SendCommand PrintPos
4388     End If
4389 End If
4390 If sCurrentCmd = "new" Then
4391     InitGame
4392     bExitReceived = False
4393     If ThreadNum = 0 Then InitThreads
4394     'LogWrite String(20, "=")
4395     'LogWrite "New Game", True
4396     GoTo NextCmd
4397 End If
4398 If sCurrentCmd = "white" Then
4399     bExitReceived = False
4400     bWhiteToMove = True
4401     bCompIsWhite = False
4402     GoTo NextCmd
4403 End If
4404 If sCurrentCmd = "black" Then
4405     bExitReceived = False
4406     bWhiteToMove = False
4407     bCompIsWhite = True
4408     GoTo NextCmd
4409 End If
4410 If sCurrentCmd = "force" Then
4411     bExitReceived = True
4412     bForceMode = True
4413     bTimeExit = True
4414     GoTo NextCmd
4415 End If
4416 If sCurrentCmd = "go" Then
4417     bCompIsWhite = bWhiteToMove ' Fix for winboard - "black" not sent before first move after book
4418     ' bComplsWhite = Not bComplsWhite
4419     bExitReceived = False
4420     bForceMode = False
4421     GoTo NextCmd
4422 End If
4423 If sCurrentCmd = "undo" Then
4424     GameMovesTakeBack 1
4425     GoTo NextCmd
4426 End If
4427 If sCurrentCmd = "remove" Then
4428     GameMovesTakeBack 2
4429     GoTo NextCmd
4430 End If
4431 If sCurrentCmd = "draw" Then
4432     SendCommand "tellics decline"
4433     ' If iXBoardProtoVer > 1 Then
4434     '     SendCommand "tellopponent Sorry, this program does not accept draws yet."
4435     ' Else
4436     '     SendCommand "tellics say Sorry, this program does not accept draws yet."
4437     ' End If
4438     GoTo NextCmd
4439 End If

```

```

4440 If sCurrentCmd = "analyze" Then
4441 ' start analyze of position / command "?" or "exit" to stop analyze
4442 bAnalyzeMode = True
4443 bPostMode = True
4444 bExitReceived = False
4445 bForceMode = False
4446 bTimeExit = False
4447 MovesToTC = 0
4448 SecondsPerGame = 0
4449 TimeIncrement = 0
4450 FixedTime = 0
4451 TimeLeft = SecondsPerGame
4452 OpponentTime = TimeLeft
4453 FixedDepth = NO_FIXED_DEPTH
4454 bCompIsWhite = Not bCompIsWhite
4455 GoTo NextCmd
4456 End If
4457 If sCurrentCmd = "exit" Then
4458 'bAnalyzeMode = False
4459 bForceMode = False
4460 bTimeExit = True
4461 GoTo NextCmd
4462 End If
4463 If Left$(sCurrentCmd, 6) = "result" Then
4464 SendCommand Mid$(sCurrentCmd, 8)
4465 bForceMode = False
4466 bTimeExit = True
4467 bExitReceived = True
4468 'LogWrite sCurrentCmd
4469 'LogWrite MatchInfo.Opponent & " (" & MatchInfo.OppRating & ") " & MatchInfo.OppComputer
4470 GoTo NextCmd
4471 End If
4472 If sCurrentCmd = "quit" Then ExitProgram
4473 ' Debug commands
4474 If Left(UCase(sCommand), 4) = "EVAL" Then
4475 bEvalTrace = True
4476 bCompIsWhite = Not bCompIsWhite
4477 StartEngine
4478 bEvalTrace = False
4479 GoTo NextCmd
4480 End If
4481 'If DebugMode Then
4482 If sCurrentCmd = "writeepd" Then SendCommand WriteEPD
4483 If sCurrentCmd = "display" Then SendCommand PrintPos
4484 If sCurrentCmd = "list" Then
4485 GenerateMoves Ply, False, iNumMoves
4486 SendCommand DEBUBPrintMoveList(Moves)
4487 End If
4488 If Left$(sCurrentCmd, 5) = "perft" Then
4489 If IsNumeric(Right$(sCurrentCmd, 1)) Then SendCommand DEBUGPerfTest(Val(Right$(
4490 sCurrentCmd, 1)))
4491 End If
4492 If Left$(sCurrentCmd, 5) = "bench" Then
4493 If IsNumeric(Right$(sCurrentCmd, 1)) Then DEBUBench Val(Mid$(sCurrentCmd, 6, 3
4494 ))
4495 End If
4496 NextCmd:
4497 Next
4498 End Sub
4499 '-----
4500 '- InitGame()
4501 '- init all values for a new game
4502 '-----
4503 Public Sub InitGame()
4504 ' Init start position
4505 CopyIntArr StartupBoard, Board

```



```

4506     BookMovePossible = bUseBook
4507     Erase Moved()
4508     GameMovesCnt = 0: Erase arGameMoves()
4509     HintMove = EmptyMove
4510     PrevGameMoveScore = VALUE_NONE
4511
4512     InitHash
4513     InitPieceSquares
4514     MoreTimeForFirstMove = True
4515     Erase arFiftyMove()
4516     Fifty = 0
4517     Nodes = 0
4518     QNodes = 0
4519     Result = NO_MATE
4520     bWhiteToMove = True
4521     bCompIsWhite = False
4522     WKingLoc = WKING_START
4523     BKingLoc = BKING_START
4524     WhiteCastled = NO_CASTLE
4525     BlackCastled = NO_CASTLE
4526     bPostMode = False
4527     bAnalyzeMode = False
4528     MovesToTC = 0
4529     TimeIncrement = 0
4530     TimeLeft = 300
4531     OpponentTime = 300
4532     FixedDepth = NO_FIXED_DEPTH
4533     ClearEasyMove
4534     bForceMode = False
4535
4536     Erase PVLength()
4537     Erase PV()
4538     Erase History
4539     Erase CounterMove()
4540     Erase ContinuationHistory()
4541     'InitContHist
4542     Erase CaptureHistory()
4543     Erase GamePosHash()
4544     Erase arGameMoves()
4545
4546     MatchInfo.EngRating = 0
4547     MatchInfo.Opponent = ""
4548     MatchInfo.OppRating = 0
4549     MatchInfo.OppComputer = False
4550     MoveOverhead = CSng(Val("0" & Trim$(ReadINISetting("MOVEOVERHEAD", "500")))) / 1000
4551     # ' Move Overhead in milliseconds
4552 End Sub
4553
4554 'Public Sub InitContHist()
4555 '    Dim j As Long, k As Long
4556 '    For j = 0 To 15 * MAX_BOARD
4557 '        For k = 0 To 15 * MAX_BOARD
4558 '            ContinuationHistory(j, k) = -140
4559 '        Next
4560 '    Next
4561 End Sub
4562
4563 Public Sub InitUCIStartPos()
4564     ' Init start position for new UCI move, keep history and hash
4565     CopyIntArr StartupBoard, Board
4566     BookMovePossible = bUseBook
4567     Erase Moved()
4568     GameMovesCnt = 0
4569     InitPieceSquares
4570     Fifty = 0
4571     Result = NO_MATE
4572     bWhiteToMove = True
4573     bCompIsWhite = False

```



```

4573 WKingLoc = WKING_START
4574 BKingLoc = BKING_START
4575 WhiteCastled = NO_CASTLE
4576 BlackCastled = NO_CASTLE
4577 bPostMode = False
4578 bAnalyzeMode = False
4579 MovesToTC = 0
4580 TimeIncrement = 0
4581 TimeLeft = 300
4582 OpponentTime = 300
4583 FixedDepth = NO_FIXED_DEPTH
4584 bForceMode = False
4585 End Sub
4586
4587 Public Sub GameMovesAdd(mMove As TMOVE)
4588     GameMovesCnt = GameMovesCnt + 1
4589     arGameMoves(GameMovesCnt) = mMove
4590     If mMove.EnPassant = ENPASSANT_WMOVE Then
4591         Board(mMove.From + 10) = WEP_PIECE
4592         EpPosArr(1) = mMove.From + 10
4593     ElseIf mMove.EnPassant = ENPASSANT_BMOVE Then
4594         Board(mMove.From - 10) = BEP_PIECE
4595         EpPosArr(1) = mMove.From - 10
4596     Else
4597         EpPosArr(1) = 0
4598     End If
4599     ClearEasyMove
4600     HashBoard GamePosHash(GameMovesCnt), EmptyMove ' for 3x repetition draw
4601 End Sub
4602
4603 Public Sub InitEpArr()
4604     ' init Enpassant array
4605     Dim i As Long
4606     EpPosArr(1) = 0
4607     For i = SQ_A1 To SQ_H8
4608         If Board(i) = WEP_PIECE Or Board(i) = BEP_PIECE Then EpPosArr(1) = i
4609     Next
4610
4611 End Sub
4612
4613 Public Sub GameMovesTakeBack(ByVal iPlies As Long)
4614     Dim i As Long
4615     Dim iUpper As Long
4616     Dim iRealFifty As Long
4617     iUpper = GameMovesCnt
4618     If iUpper >= iPlies Then
4619
4620         For i = iUpper To iUpper - (iPlies - 1) Step -1
4621             iRealFifty = Fifty
4622             UnmakeMove arGameMoves(i)
4623             CleanEpPieces
4624             If iRealFifty > 0 Then Fifty = iRealFifty - 1
4625         Next
4626
4627         GameMovesCnt = GameMovesCnt - iPlies
4628         PliesFromNull = GameMovesCnt
4629         InitPieceSquares
4630         ClearEasyMove
4631         Result = NO_MATE
4632     End If
4633 End Sub
4634
4635 Public Sub ExitProgram()
4636     ' Exit program
4637     On Error Resume Next
4638     CloseCommChannels
4639     ' END program -----
4640 End

```

```

4641 End Sub
4642 '
4643 '---- Utility functions ----
4644 '
4645 '-----
4646 'RndInt: Returns random value between iMin and IMax
4647 '-----
4648 Public Function RndInt(ByVal iMin As Long, ByVal IMax As Long) As Long
4649     Randomize
4650     RndInt = Int((IMax - iMin + 1) * Rnd + iMin)
4651 End Function
4652
4653 Public Function GetMin(ByVal X1 As Long, ByVal x2 As Long) As Long
4654     If X1 <= x2 Then GetMin = X1 Else GetMin = x2
4655 End Function
4656
4657 Public Function GetMax(ByVal X1 As Long, ByVal x2 As Long) As Long
4658     If X1 >= x2 Then GetMax = X1 Else GetMax = x2
4659 End Function
4660
4661 Public Function GetMinSingle(ByVal X1 As Single, ByVal x2 As Single) As Single
4662     If X1 <= x2 Then GetMinSingle = X1 Else GetMinSingle = x2
4663 End Function
4664
4665 Public Function GetMaxSingle(ByVal X1 As Single, ByVal x2 As Single) As Single
4666     If X1 >= x2 Then GetMaxSingle = X1 Else GetMaxSingle = x2
4667 End Function
4668
4669 Public Function GetMaxDb1(ByVal X1 As Double, ByVal x2 As Double) As Double
4670     If X1 >= x2 Then GetMaxDb1 = X1 Else GetMaxDb1 = x2
4671 End Function
4672
4673 Public Function ReadScoreArr(pDest() As TScore, ParamArray pSrc())
4674     ' Read parameter list into array of type TScore ( MG / EG )
4675     Dim i As Long
4676
4677     For i = 0 To (UBound(pSrc) - 1) \ 2
4678         pDest(i).MG = pSrc(2 * i): pDest(i).EG = pSrc(2 * i + 1)
4679     Next
4680
4681 End Function
4682
4683 Public Function ReadScoreArr2(pDest() As TScore, i1 As Long, ParamArray pSrc())
4684     ' Read parameter list into array of type TScore ( MG / EG )
4685     Dim i As Long
4686
4687     For i = 0 To (UBound(pSrc) - 1) \ 2
4688         pDest(i1, i).MG = pSrc(2 * i): pDest(i1, i).EG = pSrc(2 * i + 1)
4689     Next
4690
4691 End Function
4692
4693 Public Function ReadLngArr(pDest() As Long, ParamArray pSrc())
4694     ' Read parameter list into array of type Long
4695     Dim i As Long
4696
4697     For i = 0 To UBound(pSrc): pDest(i) = pSrc(i): Next
4698 End Function
4699
4700 Public Function ReadIntArr(pDest() As Long, ParamArray pSrc())
4701     ' Read parameter list into array of type Integer
4702     Dim i As Long
4703
4704     For i = 0 To UBound(pSrc): pDest(i) = pSrc(i): Next
4705 End Function
4706
4707 Public Function ReadIntArr2(pDest() As Long, i1 As Long, ParamArray pSrc())
4708     ' Read Integer array of 2-dimensional array: I1= dimension 1

```

```

4709     Dim i As Long
4710
4711     For i = 0 To UBound(pSrc): pDest(i1, i) = pSrc(i): Next
4712 End Function
4713
4714 Public Function ReadIntArr3(pDest() As Long, i1 As Long, i2 As Long, ParamArray pSrc
4715 ())
4716     ' Read Integer array of 3-dimensional array: l1= dimension 1, l2= dimension 2
4717     Dim i As Long
4718
4719     For i = 0 To UBound(pSrc): pDest(i1, i2, i) = pSrc(i): Next
4720 End Function
4721
4722 Public Sub CopyIntArr(SourceArr() As Long, DestArr() As Long)
4723     Dim i As Long
4724
4725     For i = LBound(SourceArr) To UBound(SourceArr) - 1: DestArr(i) = SourceArr(i): Next
4726 End Sub
4727
4728 Public Function ConvertID() As String
4729
4730     Dim Rvalue As Long
4731     Dim a As Long
4732     Dim b As Long
4733
4734     Static r As Long
4735     Static m As Long
4736     Static N As Long
4737     Const BigNum As Long = 32768
4738     Dim i As Long, c As Long, d As Long
4739
4740     Dim isText As String
4741
4742     Rvalue = 24568
4743     a = 23467
4744     b = 21333
4745
4746     isText = "hP H6Cvxr qClic v@WynxZnFm, 2FxTmQE"
4747     r = Rvalue
4748     m = (a * 4 + 1) Mod BigNum
4749     N = (b * 2 + 1) Mod BigNum
4750
4751     For i = 1 To Len(isText)
4752         c = Asc(Mid(isText, i, 1))
4753         Select Case c
4754             Case 48 To 57
4755                 d = c - 48
4756             Case 63 To 90
4757                 d = c - 53
4758             Case 97 To 122
4759                 d = c - 59
4760             Case Else
4761                 d = -1
4762         End Select
4763         If d >= 0 Then
4764             r = (r * m + N) Mod BigNum
4765             d = (r And 63) Xor d
4766             Select Case d
4767                 Case 0 To 9
4768                     c = d + 48
4769                 Case 10 To 37
4770                     c = d + 53
4771                 Case 38 To 63
4772                     c = d + 59
4773             End Select
4774             Mid(isText, i, 1) = Chr(c)
4775         End If
4776     Next i

```

```

4776
4777     ConvertID = isText
4778 End Function
4779
4780
4781 'for Office-VBA
4782 Public Sub auto_open() 'Excel
4783     Main
4784 End Sub
4785
4786 'Public Sub Word_Auto_Open() 'Word ; normal auto open creates problems with AVAST virus scanner: false positive
altert
4787 ' Main
4788 'End Sub
4789 Public Sub UCIPositionSetup(ByVal sCommand As String)
4790     ' a) position startpos moves <move1> <move2>...
4791     '   position startpos moves c2c4 e7e6 d2d4
4792     ' b) position fen <FEN> moves <move1> <move2>... used by ARENA for every move
4793     '   position fen 1r1q1n2/2p2ppk/p2p3p/P1b1p3/2P1P3/3P1N1P/1R1B1PP1/1Q4K1 b - - 0 1
4794     '   position fen 1r1q1n2/2p2ppk/p2p3p/P1b1p3/2P1P3/3P1N1P/1R1B1PP1/1Q4K1 b - - 0 1 moves b8b2 b1b2 d8a8
4795     Dim sMovesList As String, sFEN As String, p As Long
4796     InitUCIStartPos
4797     sCommand = Trim(sCommand)
4798     '--- get optional move list
4799     p = InStr(sCommand, "moves")
4800     If p = 0 Then
4801         sMovesList = ""
4802     Else
4803         sMovesList = Trim$(Mid$(sCommand, p + Len("Moves") + 1))
4804         sCommand = Left$(sCommand, GetMax(0, p - 1))
4805     End If
4806     If Left$(sCommand, Len("position startpos")) = "position startpos" Then
4807         ' InitGame already done by "ucinewgame" command
4808     ElseIf Left$(sCommand, Len("position fen")) = "position fen" Then
4809         ' FEN string
4810         sFEN = Trim$(Mid$(sCommand, Len("position fen") + 1))
4811         ReadEPD sFEN
4812     End If
4813
4814     ' moves done after the start position
4815     If sMovesList <> "" Then
4816         UCIMoves sMovesList
4817     End If
4818 End Sub
4819
4820 Public Sub TestUCIPos()
4821     ' UCIPositionSetup "position fen 1r1q1n2/2p2ppk/p2p3p/P1b1p3/2P1P3/3P1N1P/1R1B1PP1/1Q4K1 b - - 0 1
moves b8b2 b1b2 d8a8"
4822     UCIPositionSetup "position startpos moves e2e4 d7d5"
4823     Debug.Print PrintPos
4824 End Sub
4825
4826 Public Sub UCIMoves(ByVal isMoves As String)
4827     Dim i As Long
4828     Dim asList() As String
4829     asList = Split(Trim$(isMoves))
4830     For i = 0 To UBound(asList)
4831         If Not CheckLegalRootMove(Trim$(asList(i))) Then
4832             WriteTrace "UCI position setup: illegal move " & Trim$(asList(i))
4833             Exit For
4834         End If
4835     Next
4836
4837 End Sub
4838
4839 Public Function CheckLegalRootMove(ByVal isMove As String) As Boolean
4840     Dim PlayerMove As TMOVE, i As Long, iNumMoves As Long, sCoordMove As String,
sActMove As String, bLegalInput As Boolean

```

```

4841 Dim Hashkey As THashKey, sInput(4) As String
4842 CheckLegalRootMove = False
4843 If Len(Trim$(isMove)) < 4 Then Exit Function
4844
4845 For i = 0 To 4
4846     sInput(i) = Mid$(isMove, i + 1, 1)
4847 Next
4848
4849 sActMove = Trim$(isMove)
4850 bLegalInput = False
4851 '--- normal move like with 4 char: e2e4 ---
4852 If Not IsNumeric(sInput(0)) And IsNumeric(sInput(1)) And Not IsNumeric(sInput(2))
And IsNumeric(sInput(3)) Then
4853     Ply = 0
4854     GenerateMoves Ply, False, iNumMoves
4855     PlayerMove.From = FileRev(sInput(0)) + RankRev(sInput(1))
4856     PlayerMove.Target = FileRev(sInput(2)) + RankRev(sInput(3))
4857
4858     ' legal move?
4859     For i = 0 To iNumMoves - 1
4860         sCoordMove = CompToCoord(Moves(Ply, i))
4861         If Trim(sActMove) = sCoordMove Then
4862             RemoveEpPiece
4863             MakeMove Moves(Ply, i)
4864             If CheckLegal(Moves(Ply, i)) Then
4865                 bLegalInput = True
4866                 PlayerMove.Captured = Moves(Ply, i).Captured
4867                 PlayerMove.Piece = Moves(Ply, i).Piece
4868                 PlayerMove.Promoted = Moves(Ply, i).Promoted
4869                 PlayerMove.EnPassant = Moves(Ply, i).EnPassant
4870                 PlayerMove.Castle = Moves(Ply, i).Castle
4871                 PlayerMove.CapturedNumber = Moves(Ply, i).CapturedNumber
4872             End If
4873             UnmakeMove Moves(Ply, i)
4874             ResetEpPiece
4875             If bLegalInput Then Exit For
4876         End If
4877     Next
4878
4879     If Not bLegalInput Then
4880         If bWinboardTrace Then LogWrite "Illegal move: " & sCoordMove
4881     Else
4882         ' do game move
4883         PlayMove PlayerMove
4884         HashBoard Hashkey, EmptyMove
4885         If Is3xDraw(Hashkey, GameMovesCnt, 0) Then
4886             ' Result = DRAW3REP_RESULT
4887             If bWinboardTrace Then LogWrite "ParseCommand: Return Draw3Rep"
4888             'SendCommand "1/2-1/2 {Draw by repetition}"
4889         End If
4890         GameMovesAdd PlayerMove
4891         'LogWrite "move: " & sCoordMove
4892     End If
4893 End If
4894 CheckLegalRootMove = bLegalInput
4895 End Function
4896
4897 Public Sub UCISetTimeControl(ByVal isTimeControl As String)
4898     ' sample: wtime 120000 btime 120000 winc 0 binc 0 movestogo 32
4899     Dim asList() As String, p As Long, i As Long, t As Long, WTime As Long, BTime As
Long
4900     LevelMovesToTC = 0: MovesToTC = 0: TimeIncrement = 0: TimeLeft = 0: OpponentTime = 0
: SecondsPerGame = 0
4901     FixedDepth = NO_FIXED_DEPTH: FixedTime = 0
4902     asList = Split(Trim$(isTimeControl))
4903     If bTimeTrace Then WriteTrace ">> UCISetTimeControl: " & isTimeControl
4904     WTime = -1: BTime = -1: MovesToTC = 0
4905

```

```

4906 For i = 0 To UBound(asList) Step 2
4907     If asList(i) = "infinite" Then
4908         bAnalyzeMode = True
4909         bPostMode = True
4910         bExitReceived = False
4911         bForceMode = False
4912         bTimeExit = False
4913         MovesToTC = 0
4914         SecondsPerGame = 0
4915         TimeIncrement = 0
4916         FixedTime = 0
4917         TimeLeft = 999
4918         OpponentTime = TimeLeft
4919         FixedDepth = NO_FIXED_DEPTH
4920         bCompIsWhite = Not bWhiteToMove
4921     Exit For
4922 End If
4923 If i = UBound(asList) Then Exit For
4924
4925 Select Case asList(i)
4926     Case "wtime"
4927         WTime = Val("0" & Trim(asList(i + 1)))
4928     Case "btime"
4929         BTime = Val("0" & Trim(asList(i + 1)))
4930     Case "winc", "binc" ' should be equal
4931         t = Val("0" & Trim(asList(i + 1)))
4932         TimeIncrement = t / 1000#
4933         If bTimeTrace Then WriteTrace ">> UCISetTimeControl: TimeIncrement=" & asList(
            i) & " " & TimeIncrement
4934     Case "movestogo"
4935         t = Val("0" & Trim(asList(i + 1)))
4936         MovesToTC = t: LevelMovesToTC = MovesToTC
4937         If bTimeTrace Then WriteTrace ">> UCISetTimeControl: MoveToTC=" & MovesToTC
4938     Case "movetime"
4939         t = Val("0" & Trim(asList(i + 1)))
4940         FixedTime = t \ 1000#
4941         TimeLeft = FixedTime
4942         MovesToTC = 0: WTime = 0: BTime = 0: LevelMovesToTC = 0
4943         If bTimeTrace Then WriteTrace ">> UCISetTimeControl: FixedTime=" & FixedTime
4944     Case "depth"
4945         t = Val("0" & Trim(asList(i + 1)))
4946         FixedDepth = t
4947         MovesToTC = 0: WTime = 0: BTime = 0: LevelMovesToTC = 0
4948         If bTimeTrace Then WriteTrace ">> UCISetTimeControl: FixedDepth=" & FixedDepth
4949 End Select
4950
4951 Next
4952
4953 ' some GUI send one time only
4954 If WTime = -1 Then WTime = GetMax(0, BTime \ 2)
4955 If BTime = -1 Then BTime = GetMax(0, WTime \ 2)
4956
4957 If bTimeTrace Then WriteTrace ">> UCISetTimeControl: WTime=" & WTime & ", BTime=" &
    BTime & ", bWhiteToMove=" & bWhiteToMove & ", CompIsWhite=" & bCompIsWhite
4958
4959 If bCompIsWhite Then
4960     TimeLeft = WTime / 1000#
4961     If bTimeTrace Then WriteTrace ">> UCISetTimeControl: Comp=W TimeLeft=" & TimeLeft
4962     OpponentTime = BTime / 1000#
4963     If bTimeTrace Then WriteTrace ">> UCISetTimeControl: OpponentTime=" & OpponentTime
4964 Else
4965     TimeLeft = BTime / 1000#
4966     If bTimeTrace Then WriteTrace ">> UCISetTimeControl: Comp=B TimeLeft=" & TimeLeft
4967     OpponentTime = WTime / 1000#
4968     If bTimeTrace Then WriteTrace ">> UCISetTimeControl: OpponentTime=" & OpponentTime
4969 End If
4970
4971 End Sub

```

```

4972 VERSION 1.0 CLASS
4973 BEGIN
4974     MultiUse = -1 'True
4975 END
4976 Attribute VB_Name = "clsBoardField"
4977 Attribute VB_GlobalNameSpace = False
4978 Attribute VB_Creatable = False
4979 Attribute VB_PredeclaredId = False
4980 Attribute VB_Exposed = False
4981 '--- Catch Events for board square image controls (VBA has no support for control arrays like VB6)
4982 Public WithEvents ImageEvents As MSForms.Image
4983 Attribute ImageEvents.VB_VarHelpID = -1
4984 Public Name As String
4985
4986 Public Sub SetBoardField(ctl As MSForms.Image)
4987     Set ImageEvents = ctl
4988 End Sub
4989
4990 Private Sub ImageEvents_Click()
4991     psLastFieldClick = Me.Name
4992     DoFieldClicked
4993     DoEvents
4994 End Sub
4995
4996
4997 Private Sub ImageEvents_MouseDown(ByVal Button As Integer, ByVal Shift As Integer,
4998     ByVal x As Single, ByVal y As Single)
4999     psLastFieldMouseDown = Me.Name
5000 End Sub
5001
5002 Attribute VB_Name = "basCmdOutput"
5003 '=====
5004 '= basCmdOutput:
5005 '= pipe communication with external GUI ( i.e. ARENA)
5006 '=====
5007 Option Explicit
5008 ' Joacim Andersson, Brixoft Software
5009 ' http://www.brixoft.net
5010 '=====
5011 ' STARTUPINFO flags
5012 Private Const STARTF_USESHOWWINDOW = &H1
5013 Private Const STARTF_USESTDHANDLES = &H100
5014 ' ShowWindow flags
5015 Private Const SW_HIDE = 0
5016 ' DuplicateHandle flags
5017 Private Const DUPLICATE_CLOSE_SOURCE = &H1
5018 Private Const DUPLICATE_SAME_ACCESS = &H2
5019 ' Error codes
5020 Private Const ERROR_BROKEN_PIPE = 109
5021
5022 Private Type SECURITY_ATTRIBUTES
5023     nLength As Long
5024     lpSecurityDescriptor As Long
5025     bInheritHandle As Long
5026 End Type
5027
5028 Private Type STARTUPINFO
5029     cb As Long
5030     lpReserved As String
5031     lpDesktop As String
5032     lpTitle As String
5033     dwX As Long
5034     dwY As Long
5035     dwXSize As Long
5036     dwYSize As Long
5037     dwXCountChars As Long
5038     dwYCountChars As Long

```



```

5039     dwFillAttribute As Long
5040     dwFlags As Long
5041     wShowWindow As Integer
5042     cbReserved2 As Integer
5043     lpReserved2 As Long
5044     hStdInput As Long
5045     hStdOutput As Long
5046     hStdError As Long
5047 End Type
5048
5049 Private Type PROCESS_INFORMATION
5050     hProcess As Long
5051     hThread As Long
5052     dwProcessID As Long
5053     dwThreadId As Long
5054 End Type
5055
5056 Private Declare Function CreatePipe _
5057     Lib "kernel32" (phReadPipe As Long, _
5058         phWritePipe As Long, _
5059         lpPipeAttributes As Any, _
5060         ByVal nSize As Long) As Long
5061 Private Declare Function ReadFile _
5062     Lib "kernel32" (ByVal hFile As Long, _
5063         lpBuffer As Any, _
5064         ByVal nNumberOfBytesToRead As Long, _
5065         lpNumberOfBytesRead As Long, _
5066         lpOverlapped As Any) As Long
5067 Private Declare Function CreateProcess _
5068     Lib "kernel32" _
5069     Alias "CreateProcessA" (ByVal lpApplicationName As String, _
5070         ByVal lpCommandLine As String, _
5071         lpProcessAttributes As Any, _
5072         lpThreadAttributes As Any, _
5073         ByVal bInheritHandles As Long, _
5074         ByVal dwCreationFlags As Long, _
5075         lpEnvironment As Any, _
5076         ByVal lpCurrentDirectory As String, _
5077         lpStartupInfo As STARTUPINFO, _
5078         lpProcessInformation As PROCESS_INFORMATION)
5079         As Long
5079 Private Declare Function GetCurrentProcess Lib "kernel32" () As Long
5080 Private Declare Function DuplicateHandle _
5081     Lib "kernel32" (ByVal hSourceProcessHandle As Long, _
5082         ByVal hSourceHandle As Long, _
5083         ByVal hTargetProcessHandle As Long, _
5084         lpTargetHandle As Long, _
5085         ByVal dwDesiredAccess As Long, _
5086         ByVal bInheritHandle As Long, _
5087         ByVal dwOptions As Long) As Long
5088 Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long
5089 Private Declare Function OemToCharBuff _
5090     Lib "user32" _
5091     Alias "OemToCharBuffA" (lpszSrc As Any, _
5092         ByVal lpszDst As String, _
5093         ByVal cchDstLength As Long) As Long
5094
5095 ' Function GetCommandOutput
5096 '
5097 ' sCommandLine: [in] Command line to launch
5098 ' blnStdOut     [in,opt] True (default) to capture output to STDOUT
5099 ' blnStdErr     [in,opt] True to capture output to STDERR. False is default.
5100 ' blnOEMConvert: [in,opt] True (default) to convert DOS characters to Windows, False to skip conversion
5101 '
5102 ' Returns:      String with STDOUT and/or STDERR output
5103 '
5104 Public Function GetCommandOutput(sCommandLine As String, _
5105     Optional blnStdOut As Boolean = True, _

```

```

5106         Optional blnStdErr As Boolean = False, _
5107         Optional blnOEMConvert As Boolean = True) As String
5108 Dim hPipeRead As Long, hPipeWrite1 As Long, hPipeWrite2 As Long
5109 Dim hCurProcess As Long
5110 Dim sa As SECURITY_ATTRIBUTES
5111 Dim si As STARTUPINFO
5112 Dim pi As PROCESS_INFORMATION
5113 Dim baOutput() As Byte
5114 Dim sNewOutput As String
5115 Dim lBytesRead As Long
5116 Dim fTwoHandles As Boolean
5117 Dim lRet As Long
5118 Const BUFSIZE = 1024 ' pipe buffer size
5119 ' At least one of them should be True, otherwise there's no point in calling the function
5120 If (Not blnStdOut) And (Not blnStdErr) Then
5121     Err.Raise 5 ' Invalid Procedure call or Argument
5122 End If
5123 ' If both are true, we need two write handles. If not, one is enough.
5124 fTwoHandles = blnStdOut And blnStdErr
5125 ReDim baOutput(BUFSIZE - 1) As Byte
5126
5127 With sa
5128     .nLength = Len(sa)
5129     .bInheritHandle = 1 ' get inheritable pipe handles
5130 End With
5131
5132 If CreatePipe(hPipeRead, hPipeWrite1, sa, BUFSIZE) = 0 Then
5133     Exit Function
5134 End If
5135 hCurProcess = GetCurrentProcess()
5136 ' Replace our inheritable read handle with a non-inheritable. Not that it
5137 ' seems to be necessary in this case, but the docs say we should.
5138 Call DuplicateHandle(hCurProcess, hPipeRead, hCurProcess, hPipeRead, 0, 0,
DUPLICATE_SAME_ACCESS Or DUPLICATE_CLOSE_SOURCE)
5139 ' If both STDOUT and STDERR should be redirected, get an extra handle.
5140 If fTwoHandles Then
5141     Call DuplicateHandle(hCurProcess, hPipeWrite1, hCurProcess, hPipeWrite2, 0, 1,
DUPLICATE_SAME_ACCESS)
5142 End If
5143
5144 With si
5145     .cb = Len(si)
5146     .dwFlags = STARTF_USESHOWWINDOW Or STARTF_USESTDHANDLES
5147     .wShowWindow = SW_HIDE ' hide the window
5148     If fTwoHandles Then
5149         .hStdOutput = hPipeWrite1
5150         .hStdError = hPipeWrite2
5151     ElseIf blnStdOut Then
5152         .hStdOutput = hPipeWrite1
5153     Else
5154         .hStdError = hPipeWrite1
5155     End If
5156 End With
5157
5158 If CreateProcess(vbNullString, sCommandLine, ByVal 0, ByVal 0, 1, 0, ByVal 0,
vbNullString, si, pi) Then
5159     ' Close thread handle - we don't need it
5160     Call CloseHandle(pi.hThread)
5161     ' Also close our handle(s) to the write end of the pipe. This is important, since
5162     ' ReadFile will *not* return until all write handles are closed or the buffer is full.
5163     Call CloseHandle(hPipeWrite1)
5164     hPipeWrite1 = 0
5165     If hPipeWrite2 Then
5166         Call CloseHandle(hPipeWrite2)
5167         hPipeWrite2 = 0
5168     End If
5169
5170 Do

```

```

5171 ' Add a DoEvents to allow more data to be written to the buffer for each call.
5172 ' This results in fewer, larger chunks to be read.
5173 'DoEvents
5174 If ReadFile(hPipeRead, baOutput(0), BUFSIZE, lBytesRead, ByVal 0&) = 0 Then
5175     Exit Do
5176 End If
5177 If blnOEMConvert Then
5178     ' convert from "DOS" to "Windows" characters
5179     sNewOutput = String$(lBytesRead, 0)
5180     Call OemToCharBuff(baOutput(0), sNewOutput, lBytesRead)
5181 Else
5182     ' perform no conversion (except to Unicode)
5183     sNewOutput = Left$(StrConv(baOutput(), vbUnicode), lBytesRead)
5184 End If
5185 GetCommandOutput = GetCommandOutput & sNewOutput
5186 ' If you are executing an application that outputs data during a long time,
5187 ' and don't want to lock up your application, it might be a better idea to
5188 ' wrap this code in a class module in an ActiveX EXE and execute it asynchronously.
5189 ' Then you can raise an event here each time more data is available.
5190 'RaiseEvent OutputAvailabele(sNewOutput)
5191 Loop
5192
5193 ' When the process terminates successfully, Err.LastDllError will be
5194 ' ERROR_BROKEN_PIPE (109). Other values indicates an error.
5195 Call CloseHandle(pi.hProcess)
5196 Else
5197     GetCommandOutput = "Failed to create process, check the path of the command line."
5198 End If
5199 ' clean up
5200 Call CloseHandle(hPipeRead)
5201 If hPipeWrite1 Then
5202     Call CloseHandle(hPipeWrite1)
5203 End If
5204 If hPipeWrite2 Then
5205     Call CloseHandle(hPipeWrite2)
5206 End If
5207 End Function
5208
5209 Public Function ExecuteCommand(ByVal CommandLine As String, _
5210                               Optional bShowWindow As Boolean = False, _
5211                               Optional sCurrentDir As String) As String
5212     Dim proc As PROCESS_INFORMATION 'Process info filled by CreateProcessA
5213     Dim ret As Long 'long variable for get the return value of the
5214     'API functions
5215     Dim start As STARTUPINFO 'StartUp Info passed to the CreateProceeeeA
5216     'function
5217     Dim sa As SECURITY_ATTRIBUTES 'Security Attributes passeed to the
5218     'CreateProcessA function
5219     Dim hReadPipe As Long 'Read Pipe handle created by CreatePipe
5220     Dim hWritePipe As Long 'Write Pite handle created by CreatePipe
5221     Dim lngBytesRead As Long 'Amount of byte read from the Read Pipe handle
5222     Dim strBuff As String * 256 'String buffer reading the Pipe
5223     Dim mCommand As String, mOutputs As String
5224     'if the parameter is not empty update the CommandLine property
5225     If Len(CommandLine) > 0 Then
5226         mCommand = CommandLine
5227     End If
5228     'if the command line is empty then exit whit a error message
5229     If Len(mCommand) = 0 Then
5230         ' msgbox "command line empty"
5231         Exit Function
5232     End If
5233     'Create the Pipe
5234     sa.nLength = Len(sa)
5235     sa.bInheritHandle = 1&
5236     sa.lpSecurityDescriptor = 0&
5237     ret = CreatePipe(hReadPipe, hWritePipe, sa, 0)
5238     If ret = 0 Then

```

```

5239     'If an error occur during the Pipe creation exit
5240     Debug.Print "CreatePipe failed. Error: " & Err.LastDllError
5241     Exit Function
5242 End If
5243 'Launch the command line application
5244 start.cb = Len(start)
5245 start.dwFlags = STARTF_USESTDHANDLES Or STARTF_USESHOWWINDOW
5246 'set the StdOutput and the StdError output to the same Write Pipe handle
5247 start.hStdOutput = hWritePipe
5248 start.hStdError = hWritePipe
5249 ' start.hStdInput = hInReadPipe
5250 ' If bShowWindow Then
5251 ' start.wShowWindow = SW_SHOWNORMAL
5252 ' Else
5253 start.wShowWindow = SW_HIDE
5254 ' End If
5255 'Execute the command
5256 If Len(sCurrentDir) = 0 Then
5257     ret& = CreateProcess(vbNullString, mCommand, sa, sa, 1, 0&, ByVal 0&, vbNullString
5258         , start, proc)
5259 Else
5259     ret& = CreateProcess(0&, mCommand, sa, sa, 1&, 0&, 0&, sCurrentDir, start, proc)
5260 End If
5261 If ret <> 1 Then
5262     'if the command is not found ....
5263     Debug.Print "File or command not found in procedure ExecuteCommand"
5264     Exit Function
5265 End If
5266 'Now We can ... must close the hWritePipe
5267 ret = CloseHandle(hWritePipe)
5268 ' ret = CloseHandle(hInReadPipe)
5269 mOutputs = vbNullString
5270
5271 'Read the ReadPipe handle
5272 Do
5273     ret = ReadFile(hReadPipe, strBuff, 256, lngBytesRead, 0&)
5274     mOutputs = mOutputs & Left$(strBuff, lngBytesRead)
5275     'Send data to the object via ReceiveOutputs event
5276 Loop While ret <> 0
5277
5278 'Close the opened handles
5279 Call CloseHandle(proc.hProcess)
5280 Call CloseHandle(proc.hThread)
5281 Call CloseHandle(hReadPipe)
5282 'Return the Outputs property with the entire DOS output
5283 ExecuteCommand = mOutputs
5284 End Function
5285 Attribute VB_Name = "basConst"
5286 =====
5287 '= basConst:
5288 '= definition of constants
5289 =====
5290 Option Explicit
5291 '-----
5292 ' INI file
5293 '-----
5294 Public Const INI_FILE = "ChessBrainVB.ini"
5295 Public Const CONTEMPT_KEY = "CONTEMPT"
5296 Public Const LOG_PV_KEY = "LOGPV"
5297 Public Const USE_BOOK_KEY = "OPENING_BOOK"
5298 '-----
5299 'Piece definition
5300 '-----
5301 'White pieces "(Board(x) AND 1) = WCOL" WCOL = 1
5302 'Black pieces "(Board(x) AND 1) = BCOL" BCOL = 0
5303 Public Const FRAME As Long = 0 ' frame of board array
5304 Public Const WPAWN As Long = 1 ' piece numbers for each piece and color
5305 Public Const BPAWN As Long = 2 ' white pawn

```

```

5306 Public Const WKNIGHT As Long = 3 ' black pawn
5307 Public Const BKNIGHT As Long = 4
5308 Public Const WBISHOP As Long = 5
5309 Public Const BBISHOP As Long = 6
5310 Public Const WROOK As Long = 7
5311 Public Const BROOK As Long = 8
5312 Public Const WQUEEN As Long = 9
5313 Public Const BQUEEN As Long = 10
5314 Public Const WKING As Long = 11
5315 Public Const BKING As Long = 12
5316 Public Const NO_PIECE As Long = 13 ' empty field
5317 ' skip 14, WEP-Piece needs bit 1 set for white color logic
5318 Public Const WEP_PIECE As Long = 15 ' en passant dummy piece white
5319 Public Const BEP_PIECE As Long = 16 ' en passant dummy piece black
5320
5321 Public Const ENPASSANT_WMOVE As Long = 1 ' white pawn moves 2 rows > creates
WEP_PIECE
5322 Public Const ENPASSANT_BMOVE As Long = 2 ' black pawn moves 2 rows > creates
BEP_PIECE
5323 Public Const ENPASSANT_CAPTURE As Long = 3 ' en passant captures dummy piece
WEP_PIECE or BEP_PIECE
5324
5325 '--- start positions
5326 Public Const WKING_START As Long = 25
5327 Public Const BKING_START As Long = 95
5328 '--- Piece color ( (piece AND 1 )= WCOL => bit 1 set = White)
5329 Public Const WCOL As Long = 1
5330 Public Const BCOL As Long = 0
5331 '--- Squares on board
5332 Public Const SQ_A1 As Long = 21, SQ_B1 As Long = 22, SQ_C1 As Long =
23, SQ_D1 As Long = 24, SQ_E1 As Long = 25, SQ_F1 As Long = 26, SQ_G1 As Long = 27,
SQ_H1 As Long = 28
5333 Public Const SQ_A2 As Long = 31, SQ_B2 As Long = 32, SQ_C2 As Long =
33, SQ_D2 As Long = 34, SQ_E2 As Long = 35, SQ_F2 As Long = 36, SQ_G2 As Long = 37,
SQ_H2 As Long = 38
5334 Public Const SQ_A3 As Long = 41, SQ_B3 As Long = 42, SQ_C3 As Long =
43, SQ_D3 As Long = 44, SQ_E3 As Long = 45, SQ_F3 As Long = 46, SQ_G3 As Long = 47,
SQ_H3 As Long = 48
5335 Public Const SQ_A4 As Long = 51, SQ_B4 As Long = 52, SQ_C4 As Long =
53, SQ_D4 As Long = 54, SQ_E4 As Long = 55, SQ_F4 As Long = 56, SQ_G4 As Long = 57,
SQ_H4 As Long = 58
5336 Public Const SQ_A5 As Long = 61, SQ_B5 As Long = 62, SQ_C5 As Long =
63, SQ_D5 As Long = 64, SQ_E5 As Long = 65, SQ_F5 As Long = 66, SQ_G5 As Long = 67,
SQ_H5 As Long = 68
5337 Public Const SQ_A6 As Long = 71, SQ_B6 As Long = 72, SQ_C6 As Long =
73, SQ_D6 As Long = 74, SQ_E6 As Long = 75, SQ_F6 As Long = 76, SQ_G6 As Long = 77,
SQ_H6 As Long = 78
5338 Public Const SQ_A7 As Long = 81, SQ_B7 As Long = 82, SQ_C7 As Long =
83, SQ_D7 As Long = 84, SQ_E7 As Long = 85, SQ_F7 As Long = 86, SQ_G7 As Long = 87,
SQ_H7 As Long = 88
5339 Public Const SQ_A8 As Long = 91, SQ_B8 As Long = 92, SQ_C8 As Long =
93, SQ_D8 As Long = 94, SQ_E8 As Long = 95, SQ_F8 As Long = 96, SQ_G8 As Long = 97,
SQ_H8 As Long = 98
5340 '--- Move directions
5341 Public Const SQ_UP As Long = 10
5342 Public Const SQ_DOWN As Long = -10
5343 Public Const SQ_RIGHT As Long = 1
5344 Public Const SQ_LEFT As Long = -1
5345 Public Const SQ_UP_RIGHT As Long = 11
5346 Public Const SQ_UP_LEFT As Long = 9
5347 Public Const SQ_DOWN_RIGHT As Long = -9
5348 Public Const SQ_DOWN_LEFT As Long = -11
5349 '--- Files A-H
5350 Public Const FILE_A As Long = 1, FILE_B As Long = 2, FILE_C As Long =
3, FILE_D As Long = 4, FILE_E As Long = 5, FILE_F As Long = 6, FILE_G As Long = 7,
FILE_H As Long = 8
5351
5352 '--- Score values

```

```

5353 Public Const MATE0 As Long = 100000
5354 Public Const MATE_IN_MAX_PLY As Long = 100000 - 1000
5355 Public Const VALUE_INFINITE As Long = 111111
5356 Public Const VALUE_NONE As Long = 333333
5357 Public Const VALUE_KNOWN_WIN As Long = 10000
5358
5359 '-----
5360 ' Array dimensions
5361 '-----
5362 Public Const MAX_BOARD As Long = 119
5363 Public Const MAX_MOVES As Long = 250 ' max moves for a position
5364 Public Const MAX_GAME_MOVES As Long = 999
5365 Public Const MAX_PV As Long = 255
5366 Public Const LIGHTNING_DEPTH As Long = 3
5367 Public Const MAX_DEPTH As Long = 150
5368 Public Const NO_FIXED_DEPTH As Long = 1000
5369 Public Const PV_NODE As Boolean = True
5370 Public Const NON_PV_NODE As Boolean = False
5371 Public Const QS_CHECKS As Boolean = True
5372 Public Const QS_NO_CHECKS As Boolean = False
5373 Public Const GENERATE_ALL_MOVES As Boolean = False
5374
5375 '-- Depth constants
5376 Public Const DEPTH_ZERO As Long = 0
5377 Public Const DEPTH_QS_CHECKS As Long = 0
5378 Public Const DEPTH_QS_NO_CHECKS As Long = -1
5379 Public Const DEPTH_QS_RECAPTURES As Long = -5
5380 Public Const DEPTH_NONE As Long = -6
5381 Public Const DEPTH_MAX As Long = 50
5382
5383 '--- Move ordering value groups
5384 Public Const MOVE_ORDER_QUIETS As Long = -30000
5385 Public Const MOVE_ORDER_BAD_CAPTURES As Long = -60000
5386
5387 '-----
5388 ' Structure types
5389 '-----
5390 Public Type TMOVE
5391     From As Integer
5392     Target As Integer
5393     Piece As Integer
5394     Captured As Integer
5395     EnPassant As Integer
5396     Promoted As Integer
5397     Castle As Integer ' enumCastleFlag
5398     CapturedNumber As Integer
5399     OrderValue As Long
5400     SeeValue As Long
5401     IsLegal As Boolean
5402     IsChecking As Boolean
5403 End Type
5404
5405 Public Type TMatchInfo ' for future use in GUI
5406     EngRating As Long
5407     Opponent As String
5408     OppRating As Long
5409     OppComputer As Boolean
5410 End Type
5411
5412 Public Enum enumColor
5413     COL_WHITE = 1
5414     COL_BLACK = 0
5415     COL_NOPIECE = -1
5416 End Enum
5417
5418 Public Enum enumPieceType
5419     NO_PIECE_TYPE = 0
5420     PT_PAWN = 1

```

```

5421     PT_KNIGHT = 2
5422     PT_BISHOP = 3
5423     PT_ROOK = 4
5424     PT_QUEEN = 5
5425     PT_KING = 6
5426     ALL_PIECES = 7
5427     PIECE_TYPE_NB = 8
5428 End Enum
5429
5430 Public Type TMovePicker ' data fields for move picker logic
5431     CurrMoveNum As Long
5432     EndMoves As Long
5433     BestMove As TMOVE
5434     bBestMoveChecked As Boolean
5435     bBestMoveDone As Boolean ' Moves are not generated before BestMove was tried
5436     PrevMove As TMOVE
5437     ThreatMove As TMOVE
5438     LegalMovesOutOfCheck As Long
5439     bMovesGenerated As Boolean
5440     bCapturesOnly As Boolean ' for QSearch
5441     GenerateQSChecksCnt As Long ' number of ply in QSearch where checks are generated
5442 End Type
5443
5444 Public Type TScore ' final score = mg+eg scaled by game phase
5445     MG As Long ' Midgame score
5446     EG As Long ' Endgame score
5447 End Type
5448
5449 Public Enum enumCastleFlag
5450     NO_CASTLE = 0
5451     WHITEOO = 1
5452     WHITEOOO = 2
5453     BLACKOO = 3
5454     BLACKOOO = 4
5455 End Enum
5456
5457 Public Enum enumEndOfGame ' Game result
5458     NO_MATE = 0
5459     WHITE_WON = 1
5460     BLACK_WON = 2
5461     DRAW_RESULT = 3
5462     DRAW3REP_RESULT = 4
5463 End Enum
5464
5465
5466
5467 Attribute VB_Name = "basDebug"
5468 Option Explicit
5469 '=====
5470 '= basDebug:
5471 '= Debug functions
5472 '=====
5473
5474 Public TestStart As Long, TestEnd As Long
5475
5476 Public Function DEGUBPrintMoveList (MoveList () As TMOVE) As String
5477     Dim i As Long
5478     Dim strMoves As String
5479
5480     Do While Not MoveList(i).From = 0
5481         strMoves = strMoves & vbTab & MoveText (MoveList(i))
5482         i = i + 1
5483         If i Mod 3 = 0 Then strMoves = strMoves & vbCrLf
5484     Loop
5485
5486     DEGUBPrintMoveList = strMoves
5487 End Function
5488

```



```

5489 Public Sub DEBUGPerfTestSearch(ByVal iDepth As Long)
5490     Dim NumMoves As Long
5491     Dim i As Long
5492     If iDepth = 0 Then Exit Sub
5493     Ply = Ply + 1
5494     GenerateMoves Ply, False, NumMoves
5495
5496     For i = 0 To NumMoves - 1
5497         MakeMove Moves(Ply, i)
5498         If CheckLegal(Moves(Ply, i)) Then
5499             Nodes = Nodes + 1
5500             DEBUGPerfTestSearch iDepth - 1
5501         End If
5502         UnmakeMove Moves(Ply, i)
5503     Next
5504
5505     Ply = Ply - 1
5506 End Sub
5507
5508 Public Function DEBUGPerfTest(ByVal iDepth As Long) As String
5509     Dim strResult As String, StartTime As Single, EndTime As Single
5510     InitGame
5511     Ply = 1
5512     bWhiteToMove = True
5513     Nodes = 0
5514     StartTime = Timer
5515     DEBUGPerfTestSearch iDepth
5516     EndTime = Timer
5517     ' time for move generation
5518     strResult = "time: " & Format$(EndTime - StartTime, "0.00") & " nodes: "
5519
5520     ' show correct move counts until depth 5
5521     Select Case iDepth
5522     Case 1
5523         strResult = strResult & Nodes & " (expected: 20)"
5524     Case 2
5525         strResult = strResult & Nodes - 20 & " (expected: 400)"
5526     Case 3
5527         strResult = strResult & Nodes - 400 - 20 & " (expected: 8902)"
5528     Case 4
5529         strResult = strResult & Nodes - 8902 - 400 - 20 & " (expected: 197281)"
5530     Case 5
5531         strResult = strResult & Nodes - 197281 - 8902 - 400 - 20 & " (expected:
5532         4865609)"
5533     End Select
5534
5535     DEBUGPerfTest = strResult
5536 End Function
5537
5538 Public Sub DEBUGBench(ByVal iDepth As Long)
5539     ' ORIGINAL
5540     Dim i As Long, StartTime As Single, EndTime As Single, x As Long, c As Long,
5541     s As String
5542     Dim arTime(2) As Single, EPD(20) As String
5543     '--- Test positions -----
5544     'EPD(1) = "r1b1kb1r/pppp1ppp/2n1pq2/8/2PP4/P1P2N2/4PPPP/R1BQKB1R w KQkq - 1 7 " ' SF6 problem: Too
5545     high eval until ply 7
5546     ' EPD(1) = "rn1q4/pbp2kp1/1p1ppn2/8/1PP5/P5Q1/3PPP1r/R1B1KBR1 b Q b3 0 11" ' too high KSafety eval
5547     'EPD(1) = "3r2k1/p1q1r2p/bppb2p1/6Qn/2NPp3/1PN1Pp1P/PB3PP1/2R3RK b - - 3 27 " ' King attack eval too high
5548     <<<
5549     ' EPD(1) = "r3k3/p2nbpp1/bpp1p3/3nP3/2NP3P/1PB4P/P1Q2PBq/R3RK2 w q - 1 20 " ' KS eval
5550     'EPD(1) = "r4r2/p1q1n1kp/2n1ppp1/8/3P2N1/3BPP2/2Q2P1P/R3K2R w KQ - 0 19 " ' Trapped knight h3/h4
5551     'EPD(1) = "r4rk1/1p2ppbp/pq1p2p1/3P4/1nP3n1/2N2N2/PP2QPPP/R1B2RK1 b - - 0 18 " ' Trapped knight a5
5552     'EPD(1) = "rnbq1rk1/ppp2pp1/8/2np2Q/1P6/8/P1PN1PPP/R1B2RK1 b - b3 0 11"
5553     'EPD(1) = "rnbq1r2/ppp2ppk/8/2np2Q/8/8/PPPN1PPP/R1B2RK1 b - - 1 10 "
5554     'EPD(1) = "3r1r1k/1b2b1p1/1p5p/2p1Pp2/q1B2P2/4P2P/1BR1Q2K/6R1 b - - 0 1 " ' Eval BEnch
5555     'EPD(1) = "8/p6p/4k1p1/3p4/2p4P/Pr3PK1/R5P1/8 b - - 1 41 " ' Passed Pawn eval
5556     'EPD(1) = "r2qr1k1/p3bPPP/bpn2n2/2pp4/3P1B2/1PN2NP1/P3PPBP/2RQ1RK1 w - - 8 1 " ' SEE problem

```

5553 'EPD(1) = "r1b1k2r/1pp1q2p/p1n3p1/3QPp2/8/1BP3B1/P5PP/3R1RK1 w kq - 0 1 " ' WAC133  
 5554 'EPD(1) = "r3kbr1/1p3p1b/pq4Pp/3pp1n1/3PP1N1/PQ4pP/1P3P1B/R3KBR1 w Qq - 0 1 " ' Eval Test symmetric 2  
 5555  
 5556 'EPD(1) = "rnbqkbnr/1pp2pp1/p6p/3pp3/3PP3/P6P/1PP2PP1/RNBQKBNR w KQkq - 0 1 " ' Eval Test symmetric 1  
 5557 'EPD(1) = "8/5K2/8/3N4/8/8/7k/8 w - - 0 4 " ' endgame test  
 5558 'EPD(1) = "8/6R1/8/4k1K1/8/8/3r4/8 w - - 3 3 " ' draw test  
 5559 'EPD(1) = "r1bq3r/ppppR1p1/5n1k/3P4/6pP/3Q4/PP1N1PP1/5K1R w - - 0 1 " ' WAC138  
 5560 'EPD(1) = "8/7p/7k/8/1PK5/8/8/8 w - - 0 1 " ' endgame pawn promote  
 5561 'EPD(1) = "8/8/8/6p1/6Pp/5k2/7K w - - 2 95 " ' bug hanging movepicker => one legal move out of check  
 5562 'EPD(1) = "r2qk2r/pp1n1ppp/2p1p3/5b2/P2Pn3/BBP1P3/3N1PPP/R3QRK1 w kq - 0 14 " ' Eval ?  
 5563 'EPD(1) = "2r5/7K/k5P1/8/8/1p6/8/8 b - - 0 1 " ' Passed pawn test  
 5564 'EPD(1) = "3R4/p6r/8/1P2k3/2B5/8/4K3/8 w - - 50 103 " ' endgame king to pawn1p2PP2  
 5565 'EPD(1) = "r1b2rk1/p4ppp/1p1Qp3/4P2N/1P6/8/P3qPPP/3R1RK1 w - - 0 1 " ' WAC 288  
 5566 'EPD(1) = "8/8/8/Q7/8/2K3k1/7r/8 w - - 0 1 " ' KQKR  
 5567 'EPD(1) = "8/8/8/Q7/8/2K3k1/7p/8 w - - 0 1 " ' KQKP  
 5568 'EPD(1) = "8/8/8/5pk1/8/2KR4/8/8 w - - 0 1 " ' KRKP  
 5569 'EPD(1) = "2qrr1n1/3b1kp1/2pBpn1p/p2P4/1BP5/P3Q1PP/4RRK1 w - - 0 1 " ' ; e2h5 "BWTC.0031"  
 5570 ' EPD(1) = "5rk1/1pp3bp/3p2p1/2PPp3/1P2P3/2Q1B3/4q1PP/R5K1 b - - bm Bh6; id WAC.169"  
 5571 'EPD(1) = "8/7p/1R4pk/8/6PK/7P/1p6/1r6 b - - 3 1 " ' Passed pawn attacked by rook SF6: mg:1.14 eg:2.24 cp  
 5572 'EPD(1) = "8/7p/1R4pk/8/6PK/7P/1pr5/8 b - - 0 1 " ' Passed pawn attacked by rook, blocked by own rook SF6: 1.38 2.36  
 5573 'EPD(1) = "8/7p/3R2pk/8/1r4PK/7P/1p6/8 w - - 0 1 " ' Passed pawn defended by rook SF6: 2.54 3.97  
 5574 ' EPD(1) = "r3r1k1/pbq2p2/4p2p/1p1nP2Q/2pR4/2P5/PPB2PPP/4R1K1 w - - 0 20 " ' Defend  
 5575 'EPD(1) = "r3r1k1/pbq2pp1/4p2B/1p1nP2Q/2pR4/2P5/PPB2PPP/4R1K1 b - - 0 19 " ' Attack f7f5 (g7xh6 bad)  
 5576 'EPD(1) = "r1bqkbnr/ppp2ppp/2np4/4p3/2B1P3/5N2/PPPP1PPP/RNBQK2R w KQkq - 2 4 " ' KSafety/Castle eval  
 5577 'EPD(1) = "rnbq1rkr/pppp1p1p/5n2/2b1p3/4P3/2NP4/PPP2PPP/R1BQKBNR b KQ - 2 1 " ' KSafety/Castle eval-Black  
 5578 ' EPD(1) = "6k1/6p1/8/8/8/4P2P/6K1 b - - " ' Test Endgame Tablebase acces in search for root  
 5579 'EPD(1) = "8/6k1/6p1/8/7r/3P1KP1/8/8 w - - 0 1 " ' Test Endgame Tablebase acces in search for ply=1  
 5580 ' EPD(1) = "r3k2r/pb3pbb/2p1p3/1q2p3/2p5/6P1/1PQ1PPBP/R1BR2K1 w kq - 0 2 "  
 5581 'EPD(1) = "2r1r1k1/4bp1p/p2pp1pP/q3n1P1/Np1Nb3/1P2B3/P1PQ4/1K2RBR1 b - - 1 21 " ' e5f3 not found  
 5582 ' EPD(1) = "2r1r1k1/4bp1p/p2pp1pP/q5P1/Np2b3/1P2BN2/P1PQ4/1K2RBR1 b - - 0 21 "  
 5583 ' EPD(1) = "4r1k1/4bp1p/p2pp1pP/q5P1/Np2b3/1P2BN2/P1rQ4/1K2RBR1 w - - 0 22 " ' d2xc2 ok, d2d4 >Rc2c4 illegal move, IsCHecking no detected  
 5584 'EPD(1) = "r2r2k1/pb3p1p/1qn1p2Q/5p2/1p1P4/1NPB4/P4PPP/2R1R1K1 b - - 0 22 " ' KSafety test  
 5585 ' EPD(1) = "8/5pk1/1p4Pp/q6P/Q7/1P6/8/6K1 b - - 0 1 " ' ShelterStorm test  
 5586 'EPD(1) = "5k2/6b1/8/4N3/8/8/3P1K2/8 w - - 3 1 " ' Scale factor 1 pawn test  
 5587 ' EPD(1) = "r1b2r1k/p5pp/2nq4/Ppp1pp2/2Bn1N1Q/2B1R3/2P2PPP/R5K1 w - b6 0 2 " ' EnPassant test  
 5588 ' EPD(1) = "r1b2r1k/pp4pp/2nq4/P1p1pp2/2Bn1N1Q/2B1R3/2P2PPP/R5K1 b - - 1 1 " ' EnPassant test2 move b7b5  
 5589 ' EPD(1) = "6k1/4Q1p1/7p/8/nn6/1p3R2/5PPP/6K1 w - - 1 1 " ' mate threat  
 5590 ' EPD(1) = "8/2pp4/3kPKP1/3P4/8/8/8/8 w - - 0 1 "  
 5591 'EPD(1) = "8/8/2k5/8/5K2/3R4/8/3qR3 w - - 0 1" ' EGTB  
 5592 'EPD(1) = "8/3PK3/8/5p1k/8/8/8/8 b - - 0 w " ' EGTB test promotion  
 5593 'EPD(1) = "8/5PK1/8/2Q5/4P1k1/8/8/8 b - - 0 14 "  
 5594 'EPD(1) = "8/4k3/8/8/5P2/5K2/8/8 b - - 4 3 " ' EGTB KPK  
 5595 'EPD(1) = "8/3k3K/7P/1r6/5p2/8/8/8 b - - 0 1 "  
 5596 'EPD(1) = "8/8/3R4/p3npk1/P3p2p/4P3/3K1PP1/r2B4 w - - 8 39 " ' EP capture mate bug  
 5597 'EPD(1) = "8/2b5/8/4kN2/1r4K1/6N1/8/8 w - - 0 1" ' endgame scale factor no pawns  
 5598 'EPD(1) = "8/8/7k/p1P4p/P6P/7K/8/8 w - - 0 1" ' passed pawn test 1 rank 5  
 5599 'EPD(1) = "8/8/7k/p1P4p/P6P/7K/8/2R5 w - - 0 1" ' passed pawn test 2 defended from behind  
 5600 'EPD(1) = "8/8/7k/p1P4p/P6P/7K/8/2r5 w - - 0 1" ' passed pawn test 3 attacked from behind  
 5601 'EPD(1) = "8/7r/7k/p1P4p/P6P/7K/8/2R5 w - - 0 1" ' passed pawn test 4 defended from behind + attacked path  
 5602 'EPD(1) = "8/8/2P4k/p6p/P6P/7K/8/2R5 w - - 0 1" ' passed pawn test 5 defended from behind rank 6  
 5603 'EPD(1) = "7k/5K1p/7P/8/8/8/8/8 b - - 1 1 " ' no move draw  
 5604 ' EPD(1) = "r5k1/pp4pp/2pb3r/3p2q1/P1PP1nB1/1PB1P1PP/7K/R2Q2R1 b - - 0 27" ' KSafety  
 5605  
 5606 'EPD(1) = "5rk1/pp4pp/2pb3r/3p2q1/P1PP4/1PB1P1PB/7K/R4QR1 b - - 2 29" '  
 5607 'EPD(1) = "6k1/4b1p1/5p1Q/1p2pP2/4P3/1P6/6PP/6rK w - - 0 41 " 'only one legalmove  
 5608 'EPD(1) = " /4b1p1/8/1p2p3/8/7p/5p2/6bK w - - 0 1 " 'no legal move  
 5609  
 5610 'EPD(1) = "8/8/7k/8/8/8/6PP/3r3K w - - 1 1 " ' mated result : bestmove (none)  
 5611  
 5612 'EPD(1) = "r5k1/pp4pp/2pb3r/3p2q1/P1PP4/1PB1P1PB/7K/R2Q2R1 b - - 0 28 "  
 5613 'EPD(1) = "6r1/2pq2pk/1p3p1p/1P1Pp2P/Q3P1P1/p1R3K1/P7/8 w - - 98 109 " ' fifty  
 5614 ' EPD(1) = "k7/8/P7/1K6/8/8/8/8 w - - 12 1 " ' endgame kpk"  
 5615 ' EPD(1) = "r1bq3r/1p1nbpk1/p2p1np1/P1pPp3/4P2p/2NBB2P/1PPQNPP1/R4RK1 b - - 1 14 " ' Tactic  
 5616  
 5617 ' EPD(1) = "4kb1r/1pqb1ppp/p3p3/3pP3/2r2P2/2NQB3/PPP3PP/R4RK1 w k - 6 14" ' a2a3 lost

```

5618 ' EPD(1) = "4kb1r/1pqb1ppp/p3p3/3pP3/2r2P2/P1NQB3/1PP3PP/R4RK1 b k - 0 14 " ' d5d4 wins
5619 ' EPD(1) = "4kb1r/1pqb1ppp/p3p3/4P3/2rB1P2/P1NQ4/1PP3PP/R4RK1 b k - 0 15 " ' Txd4 wins
5620 ' EPD(1) = "4kb1r/1pqb1ppp/p3p3/4P3/3r1P2/P1NQ4/1PP3PP/R4RK1 w k - 0 16 " ' d3d4 lost
5621 ' EPD(1) = "4kb1r/1pqb1ppp/p3p3/4P3/2rp1P2/P1NQB3/1PP3PP/R4RK1 w k - 0 15 " ' e3xd4 lost
5622
5623 ' EPD(1) = "8/6k1/8/5P1P/6PK/3n4/8/8 w - - 0 81 "
5624 ' EPD(1) = "r3k2r/p1ppqpb1/bn2pnp1/3PN3/1p2P3/2N2Q1p/PPPBPPPP/R3K2R w KQkq - 0 1" 'Qsearch test
5625 ' EPD(1) = "1r4Bk/PPPPPPp1P/8/7R/R6K/8/ppppppQ1/6B1 w - - 0 1 " ' QS
5626 '----- normal test ----
5627 EPD(1) = "1b5k/7P/p1p2np1/2P2p2/PP3P2/4RQ1R/q2r3P/6K1 w - - 0 1"
5628 ' EPD(1) = "4k3/6KP/8/8/6r1/8/7p/8 w - - " ' Endgame tablesbase test
5629 ' EPD(1) = "4k2K/7P/8/8/6r1/8/7p/8 b - - 1 1" 'EGTB
5630
5631 'EPD(1) = "r3k2r/pp2pp1p/8/q2Pb3/2P5/4p3/B1Q2PPP/2R2RK1 w kq - bm c5;" ' Bug depth 100
5632 ' EPD(1) = "8/7p/5P1k/1p5P/5p2/2p1p3/P1P1P1P1/1K3Nb1 w - - bm Ng3" ' Bug Depth
5633 'EPD(1) = "3kB3/5K2/7p/3p4/3pn3/4NN2/8/1b4B1 w - - " ' crash max_ply
5634 ' EPD(1) = "8/8/8/4k3/3r2p1/3P4/3K4/8 b - - 0 7" ' EGTB
5635
5636 'EPD(1) = "1r1r4/1p4k1/p1b2R2/P1N2p1p/1P3QpP/3p2P1/5P1K/8 b - - 0 67" ' Draw bug
5637 'EPD(1) = "1r1r4/1p6/p1b4Q/P1N1kp1p/1P4pP/3p2P1/5P1K/8 w - - 6 71 ' Draw fehler vor Fehlzug h6f4 -149305158
-1177636898"
5638 ' EPD(1) = "1r1r4/1p6/p1b5/P1N1kp1p/1P3QpP/3p2P1/5P1K/8 b - - 7 71" 'nach Fehlzug h6f4 -1641286943
1195148230
5639 'EPD(1) = "1r1r4/1p6/p1b2k2/P1N2p1p/1P3QpP/3p2P1/5P1K/8 w - - 8 72" ' Draw fehler endstellung 3x
5640 ' EPD(1) = "8/8/8/4kp2/1R6/P2q1PPK/8 w - - 0 1"
5641 ' EPD(1) = "1r3k2/5pbQ/3q4/3PpBP1/5P2/p1B5/P1p5/1KR5 w - - 0 41" ' Kb1a1 bad
5642
5643 'EPD(1) = "r1b3k1/p2p1nP1/2pqr3/1p2p1QP/2B1Pn2/1P6/P1PP4/1K4R1 w - - 2 3" ' g5d8
5644 '#####
5645
5646 'EPD(2) = "1rb2rk1/p3nppp/1p1qp3/3n2N1/2pP4/2P3P1/PP3PBP/R1BQR11K w - - " 'TEST 2
5647 'EPD(2) = "2k4B/bpp1qp2/p1b5/7p/1PN1n1p1/2Pr4/P5PP/R3QR1K b - - " ' g3g4 ; Ng3; e1d3not h2xg3! <<<<<<
5648 EPD(2) = "r1bqk1r1/lp1p1n2/p1n2pN1/2p1b2Q/2P1Pp2/1PN5/PB4PP/R4RK1 w q - - " ' f1xf4
SF-Eval-0.5
5649 EPD(3) = "r1b2rk1/p2nq1p1/lpp1p2p/5p2/2PPp3/2Q1P3/PP1N1PPP/2R1KB1R w K - 0 13" '---
quiet
5650 EPD(4) = "6k1/p1r5/4b1p1/R1pprp1p/7P/1P1BP3/P1P3P1/4R1K1 w - - 4 25" ' no advantage
5651 EPD(5) = "8/8/2R5/1p2qp1k/1P2r3/2PQ2P1/5K2/8 w - - 0 1" ' Endgame
5652 EPD(6) = "r7/pbk5/1pp5/4n1q1/2P5/1P6/P4BBQ/4R1K1 b - - 0 33" '
5653 EPD(7) = "r1bqk2r/p2p1pp1/lp2pn1p/nlpP2B1/1bP5/2N2N2/PPQ1PPPP/R3KB1R w KQkq - 0 9"
'<<<<< AKT
5654 EPD(8) = "r3qbk1k/1b4p1/p2pr2p/3n4/Pnp1N1N1/6RP/1B3PP1/1B1QR1K1 w - - 0 1" ' Nxh6
SF-Eval +1,4
5655 '-----
5656 DebugMode = True
5657
5658 'iDepth = 8
5659 ' ReadGame "Drawbug2.txt"
5660 'bForceMode = False
5661
5662
5663 " setup UCI game string, see ARENA GUI protocol trace window
5664 "---- 3x draw problem g8g7 d8f6
5665 "UCIPositionSetup "position fen r1bqk2r/pp1nbppp/2p1p3/3n4/4N3/3P1NP1/PPP1QPBP/R1B1K2R w KQkq - 0 1
moves e1g1 e8g8 c2c4 d5f6 e4c3 e6e5 f3e5 d7e5 e2e5 f8e4 d3d4 e7b4 e5f4 b4c3 b2c3 c8e6 f1e1 e6c4 e1e8 d8e8
c1d2 e8d7 a2a4 f6d5 f4e4 a8e8 e4c2 d5f6 c2b2 g7g6 a4a5 f6g4 g2f3 g4f6 b2b4 d7f5 f3g2 f5d3 d2e1 d3e2 h2h3
a7a6 b4b1 e8e7 b1d1 e2e6 d1d2 g8g7 d2f4 c4d5 f2f3 e6c8 g3g4 d5b3 f4d6 f6d5 d6a3 b3c4 a3c5 e7e2 g2f1 c8e8
f1e2 e8e2 c5d6 e2f3 g4g5 f3f1 g1h2 h7h6 g5h6 g7h7 d6e5 f7f6 e5e4 d5f4 e4e7 h7h6 e7f8 h6h7 f8e7 h7g8 e7d8 g8f7
d8d7 f7f8 d7d8 f8g7 d8e7 c4f7 e7f6 g7g8 f6d8 g8g7 d8f6 g7g8 f6d8"
5666 "UCIPositionSetup "position fen r1bq1rk1/ppp2ppp/5n2/2bp4/2NPP3/2P5/PP3PPP/RNBQK2R w KQ - 0 1 moves
c4e3 c5e7 e4e5 f6e4 f2f3 e4g5 f3f4 g5e4 b1d2 b7b6 f4f5 c8a6 d1f3 a6b7 d2e4 d5e4 f3g4 g8h8 e1f2 f8e8 h1d1 a7a5
f2g1 a5a4 a2a3 e7f8 g4f4 b6b5 a1b1 f7f6 e5e6 d8d6 f4g4 a8d8 b2b3 a4b3 b1b3 d6a6 b3b1 b7a8 g4e2 d8b8 e2f2
e8d8 e3c2 a6c6 c1d2 c6c4 c2e3 c4a4 d2e1 a4a3 f2e2 a3a6 e1g3 a6a5 b1a1 a5b6 g3f4 a8c6 d1b1 b8a8 a1a8 d8a8
h2h4 f8e7 h4h5 h7h6 e2c2 a8a7 c2d2 b6b8 c3c4 b5b4 c4c5 b8d8 d2b4 a7b7 b4b7 c6b7 b1b7 d8d4 b7c7 e7c5 e6e7
c5e7 c7e7 h8h7 e7c7 d4d2 g1f1 d2d3 f1e1 d3b5 e1f2 b5b2 c7c2 b2d4 f2g3 d4b4 f4c7 b4e1 g3f4 e1h4 g2g4 h4h1
c2f2 h1g1 f2g2 g1a1 c7b8 a1b1 b8d6 b1d3 d6c7 d3b1 g2c2 b1h1"
5667 "UCIPositionSetup "position fen r1bq1rk1/ppp2ppp/5n2/2bp4/2NPP3/2P5/PP3PPP/RNBQK2R w KQ - 0 1 moves
c4e3 c5e7 e4e5 f6e4 f2f3 e4g5 f3f4 g5e4 b1d2 b7b6 f4f5 c8a6 d1f3 a6b7 d2e4 d5e4 f3g4 g8h8 e1f2 f8e8 h1d1 a7a5

```

```

f2g1 a5a4 a2a3 e7f8 g4f4 b6b5 a1b1 f7f6 e5e6 d8d6 f4g4 a8d8 b2b3 a4b3 b1b3 d6a6 b3b1 b7a8 g4e2 d8b8 e2f2
e8d8 e3c2 a6c6 c1d2 c6c4 c2e3 c4a4 d2e1 a4a3 f2e2 a3a6 e1g3 a6a5 b1a1 a5b6 g3f4 a8c6 d1b1 b8a8 a1a8 d8a8
h2h4 f8e7 h4h5 h7h6 e2c2 a8a7 c2d2 b6b8 c3c4 b5b4 c4c5 b8d8 d2b4 a7b7 b4b7 c6b7 b1b7 d8d4 b7c7 e7c5 e6e7
c5e7 c7e7 h8h7 e7c7 d4d2 g1f1 d2d3 f1e1 d3b5 e1f2 b5b2 c7c2 b2d4 f2g3 d4b4 f4c7 b4e1 g3f4 e1h4 g2g4 h4h1
c2f2 h1g1 f2g2 g1a1 c7b8 a1b1 b8d6 b1d3 d6c7 d3b1 g2c2 b1h1 c2f2 h1g1 f2g2 g1b1 g2h2 b1a1"
5668 ' FixedDepth = 15: MovesToTC = 0: TimeLeft = 20: TimeIncrement = 10: bPostMode = True: bComplsWhite =
bWhiteToMove
5669 ' '--- start computing -----
5670 ' StartEngine
5671 ' Stop
5672 ' End
5673
5674 '
5675 For x = TestStart To TestEnd
5676
5677 For i = 0 To 0 ' number of time measure runs > 1x
5678 'For i = 0 To 2 ' number of time measure runs > 3x
5679 InitGame ' Reset FixedDepth , Hash, History...
5680 ReadEPD EPD(x) ' Reset FixedDepth
5681
5682 #####
5683
5684 'ParseCommand "g7f6"
5685 'Debug.Print Fifty
5686 'ParseCommand "f4g5"
5687 'Debug.Print Fifty
5688 'ParseCommand "f6e5"
5689 'Debug.Print Fifty
5690
5691 'ParseCommand "g5f4"
5692 'Debug.Print Fifty
5693 'ParseCommand "e5f6"
5694 'Debug.Print Fifty
5695
5696 'ParseCommand "f4h6"
5697 'ParseCommand "f6e5"
5698 '
5699 'ParseCommand "h6f4"
5700 'ParseCommand "e5f6"
5701
5702
5703 If True Then
5704 If x = 3 Or x = 4 Or x = 5 Or x = 7 Then
5705 FixedDepth = iDepth + 1
5706 Else
5707 FixedDepth = iDepth
5708 End If
5709 ' Else
5710 ' FixedTime = 4
5711 MovesToTC = 0: TimeLeft = 20: TimeIncrement = 10
5712 End If
5713 If InStr(EPD(x), " w") > 0 Then
5714 bCompIsWhite = True 'False:
5715 bWhiteToMove = True '---False
5716 Else
5717 bCompIsWhite = False 'True 'False:
5718 bWhiteToMove = False 'True '---False
5719 End If
5720 ' ParseCommand "b7b5"
5721 bPostMode = True
5722 ' bPostMode = False
5723 'SendCommand PrintPos
5724 If False Then ' Time based end of thinking
5725 FixedDepth = NO_FIXED_DEPTH
5726 LevelMovesToTC = 40
5727 MovesToTC = 0
5728 TimeLeft = 120
5729 TimeIncrement = 0

```

```

5730         GameMovesCnt = 119 ' plies, /2 for MoveCnt
5731     End If
5732
5733     StartTime = Timer
5734     '--- start computing -----
5735     StartEngine
5736     '
5737     EndTime = Timer
5738
5739     arTime(i) = EndTime - StartTime
5740     If arTime(i) = 0 Then arTime(i) = 1
5741     bPostMode = True
5742     SendCommand vbCrLf & "time: " & Format$(arTime(i), "0.000") & " nod: " & Nodes &
        " qn: " & QNodes & "(DMax:" & QSDepthMax & ")" & " ev:" & EvalCnt & " sc: " &
        FinalScore & " EGTB:" & EGTBasesHitsCnt & " Ply:" & MaxPly & " " & s & vbCrLf
5743
5744     ' Test Counter
5745     s = ""
5746     For c = 1 To 19
5747         If TestCnt(c) <> 0 Then s = s & CStr(c) & ":" & TestCnt(c) & ","
5748     Next c
5749     If s <> "" Then SendCommand "Counter: " & s
5750 Next
5751
5752 If arTime(0) < arTime(1) Then
5753     If arTime(0) < arTime(2) Then
5754         i = 0
5755     Else
5756         i = 2
5757     End If
5758 ElseIf arTime(1) < arTime(2) Then
5759     i = 1
5760 Else
5761     i = 2
5762 End If
5763 'count 3x
5764 If arTime(i) > 0 Then
5765     SendCommand "best time: " & Format$(arTime(i), "0.000") & " nps: " & Int(Nodes /
        arTime(i))
5766 Else
5767     SendCommand "best time: " & Format$(0, "0.000") & " nps: " & Nodes
5768 End If
5769 SendCommand "Hash usage:" & Format((Cdbl(HashUsage) / Cdbl(HashSize)) * 100#,
    "0.00")
5770 SendCommand "-----"
5771 Next x
5772
5773 End Sub
5774
5775 Public Sub WriteDebug(s As String)
5776     Debug.Print s
5777 End Sub
5778
5779 Public Sub DMoves()
5780     ' Debug: print current move line
5781     Dim i As Long, s As String
5782     s = CStr(RootDepth) & "/" & CStr(Ply) & ">"
5783
5784     For i = 1 To Ply - 1
5785         s = s & CStr(i) & ":" & MoveText(MovesList(i)) & "/"
5786     Next
5787
5788     Debug.Print s
5789     DoEvents
5790 End Sub
5791
5792 Public Function SearchMovesList() As String
5793     ' print current move line

```

```

5794 Dim i As Long, s As String
5795 s = ""
5796
5797 For i = 1 To Ply - 1
5798     s = s & MoveText(MovesList(i)) & " "
5799 Next
5800 s = Trim(s)
5801 SearchMovesList = s
5802 End Function
5803
5804
5805 Public Sub DEBUGLoadGame(ByVal iDepth As Long)
5806     'ORIGINAL
5807     Dim i As Long, StartTime As Single, EndTime As Single, x As Long, c As Long,
5808         s As String
5809     Dim arTime(2) As Single
5810     iDepth = 8
5811     DEBUGReadGame "bug001.txt"
5812     bForceMode = False
5813
5814     For i = 0 To 0 ' number of time measure runs > 1x
5815         FixedDepth = iDepth
5816         bCompIsWhite = False 'True 'False:
5817         bWhiteToMove = False 'True '---False
5818         bPostMode = True
5819         StartTime = Timer
5820         StartEngine
5821         EndTime = Timer
5822         arTime(i) = EndTime - StartTime
5823         If arTime(i) = 0 Then arTime(i) = 1
5824         bPostMode = True
5825         SendCommand vbCrLf & "time: " & Format$(arTime(i), "0.000") & " nod: " & Nodes &
5826             " qn: " & QNodes & " ev:" & EvalCnt & " sc: " & EvalSFTto100(FinalScore) & " Ply:"
5827             & MaxPly & " " & s & vbCrLf
5828     Next
5829
5830     SendCommand "-----"
5831 End Sub
5832
5833 Public Sub DEBUGReadGame(sFile As String)
5834     'Read PGN File
5835     Dim h As Long, s As String, m As Long, sInp As String, m1 As String, m2
5836         As String
5837     InitGame
5838     bForceMode = True
5839     h = 10 'FreeFile()
5840     Open sFile For Input As #h
5841
5842     Do Until EOF(h)
5843         Line Input #h, sInp
5844         sInp = Trim(sInp) & " "
5845         s = Trim(sInp)
5846         'Debug.Print s
5847         m1 = Trim(Left(s, 4))
5848         If Len(m1) = 4 Then
5849             ParseCommand m1 & vbCrLf
5850         End If
5851     Loop
5852
5853     Close #h
5854 End Sub
5855
5856 VERSION 5.00
5857 Begin VB.Form frmDebugMain
5858     Caption = "ChessBrainVB debug console"
5859     ClientHeight = 9960
5860     ClientLeft = 60
5861     ClientTop = 345
5862     ClientWidth = 14715

```



```

5858 Icon = "DebugMain.frx":0000
5859 LinkTopic = "Form1"
5860 ScaleHeight = 9960
5861 ScaleWidth = 14715
5862 StartUpPosition = 2 'CenterScreen
5863 Begin VB.CommandButton cmdRunUCI
5864     Caption = "Calc UCI-Pos"
5865     Height = 330
5866     Left = 4200
5867     TabIndex = 10
5868     Top = 600
5869     Width = 1305
5870 End
5871 Begin VB.TextBox txtUciPosition
5872     Height = 285
5873     Left = 5640
5874     TabIndex = 9
5875     Text = $"DebugMain.frx":0442
5876     Top = 600
5877     Width = 8535
5878 End
5879 Begin VB.CommandButton cmdThink
5880     Caption = "Think"
5881     Height = 330
5882     Left = 1800
5883     TabIndex = 8
5884     Top = 600
5885     Width = 1425
5886 End
5887 Begin VB.CommandButton cmdNewgame
5888     Caption = "New game"
5889     Height = 330
5890     Left = 120
5891     TabIndex = 7
5892     Top = 600
5893     Width = 1425
5894 End
5895 Begin VB.CommandButton cmdTx
5896     Caption = "flxf4"
5897     Height = 330
5898     Left = 13200
5899     TabIndex = 6
5900     Top = 240
5901     Width = 945
5902 End
5903 Begin VB.CommandButton cmdT2
5904     Caption = "g4xh6"
5905     Height = 330
5906     Left = 11880
5907     TabIndex = 5
5908     Top = 240
5909     Width = 945
5910 End
5911 Begin VB.CommandButton cmdTest1
5912     Caption = "e3e8+ M8"
5913     Height = 330
5914     Left = 10440
5915     TabIndex = 4
5916     Top = 240
5917     Width = 1065
5918 End
5919 Begin VB.CommandButton cmdFakeInput
5920     Caption = "Send"
5921     Height = 330
5922     Left = 9000
5923     TabIndex = 1
5924     Top = 240
5925     Width = 1065

```



```

5926 End
5927 Begin VB.ComboBox cboFakeInput
5928     Height           = 315
5929     Left              = 105
5930     TabIndex         = 0
5931     Top               = 240
5932     Width             = 8796
5933 End
5934 Begin VB.TextBox txtIO
5935     BackColor         = &H00E0E0E0&
5936     BeginProperty Font
5937         Name           = "Courier New"
5938         Size           = 8.25
5939         Charset         = 0
5940         Weight          = 400
5941         Underline       = 0 'False
5942         Italic          = 0 'False
5943         Strikethrough    = 0 'False
5944     EndProperty
5945     ForeColor          = &H00FF0000&
5946     Height             = 8892
5947     Left               = 120
5948     Locked              = -1 'True
5949     MultiLine          = -1 'True
5950     ScrollBars          = 3 'Both
5951     TabIndex           = 2
5952     TabStop             = 0 'False
5953     Top                = 960
5954     Width              = 14310
5955 End
5956 Begin VB.Label lblDescr
5957     BackStyle          = 0 'Transparent
5958     Caption             = "Input"
5959     Height              = 195
5960     Index               = 2
5961     Left                = 120
5962     TabIndex            = 3
5963     Top                 = 0
5964     Width               = 1335
5965 End
5966 End
5967 Attribute VB_Name = "frmDebugMain"
5968 Attribute VB_GlobalNameSpace = False
5969 Attribute VB_Creatable = False
5970 Attribute VB_PredeclaredId = True
5971 Attribute VB_Exposed = False
5972 '=====
5973 '= frmDebugMain:
5974 '= debug form
5975 '=====
5976 Option Explicit
5977
5978
5979 Private Sub cboFakeInput_KeyPress(KeyAscii As Integer)
5980     If KeyAscii = 13 Then cmdFakeInput_Click
5981 End Sub
5982
5983 Private Sub cmdFakeInput_Click()
5984     FakeInput = cboFakeInput.Text & vbCrLf
5985     FakeInputState = True
5986     cboFakeInput.SelStart = 0
5987     cboFakeInput.SelLength = Len(cboFakeInput.Text)
5988     cboFakeInput.SetFocus
5989
5990     UCIMode = False
5991     pbIsOfficeMode = True 'TEst
5992 End Sub
5993

```

```

5994 Private Sub cmdWb_Click()
5995     bPostMode = True
5996     ParseCommand "setboard r1b2rk1/pp1n2pp/2p1p3/2Pp4/1q1Pp3/4P1PN/PP2QPBP/2R2RK1 w - -
5997     ParseCommand "sd 10" & vbLf
5998     ParseCommand "go" & vbLf
5999
6000 End Sub
6001
6002 Private Sub cmdNewgame_Click()
6003     UCIMode = True
6004     cboFakeInput.Text = "ucinewgame"
6005     cmdFakeInput_Click
6006 End Sub
6007
6008 Private Sub cmdRunUCI_Click()
6009     UCIPositionSetup "position fen r1bqk2r/pp1nbppp/2p1p3/3n4/4N3/3P1NP1/PPP1QPBP/R1B1K2R
w KQkq - 0 1 moves e1g1 e8g8 c2c4 d5f6 e4c3 e6e5 f3e5 d7e5 e2e5 f8e8 d3d4 e7b4 e5f4
b4c3 b2c3 c8e6 f1e1 e6c4 e1e8 d8e8 c1d2 e8d7 a2a4 f6d5 f4e4 a8e8 e4c2 d5f6 c2b2 g7g6
a4a5 f6g4 g2f3 g4f6 b2b4 d7f5 f3g2 f5d3 d2e1 d3e2 h2h3 a7a6 b4b1 e8e7 b1d1 e2e6 d1d2
g8g7 d2f4 c4d5 f2f3 e6c8 g3g4 d5b3 f4d6 f6d5 d6a3 b3c4 a3c5 e7e2 g2f1 c8e8 f1e2 e8e2
c5d6 e2f3 g4g5 f3f1 g1h2 h7h6 g5h6 g7h7 d6e5 f7f6 e5e4 d5f4 e4e7 h7h6 e7f8 h6h7 f8e7
h7g8 e7d8 g8f7 d8d7 f7f8 d7d8 f8g7 d8e7 c4f7 e7f6 g7g8 f6d8 g8g7 d8f6 g7g8 f6d8"
6010 FixedDepth = 15: MovesToTC = 0: TimeLeft = 20: TimeIncrement = 10: bPostMode = True
6011 '--- start computing -----
6012 StartEngine
6013
6014 End Sub
6015
6016 Private Sub cmdT2_Click()
6017     cboFakeInput.Text = "bench 21"
6018     TestStart = 8
6019     TestEnd = 8
6020 End Sub
6021
6022 Private Sub cmdTest1_Click()
6023     cboFakeInput.Text = "bench 23"
6024     TestStart = 1
6025     TestEnd = 1
6026 End Sub
6027
6028
6029 Private Sub cmdThink_Click()
6030     cboFakeInput.Text = "go"
6031     cmdFakeInput_Click
6032 End Sub
6033
6034 Private Sub cmdTx_Click()
6035     cboFakeInput.Text = "bench 21"
6036     TestStart = 2
6037     TestEnd = 2
6038 End Sub
6039
6040 Private Sub Form_Load()
6041     txtIO = "*" STDIN HANDLE: " & hStdIn & vbTab & "STDOUT HANDLE: " & hStdOut & " *" & vbCrLf
6042     txtIO = ""
6043     cboFakeInput = "bench 14"
6044     cboFakeInput.AddItem "analyze"
6045     cboFakeInput.AddItem "eval" 'input in Immediate window and Tracexxx.txt
6046     cboFakeInput.AddItem "bench 6"
6047     cboFakeInput.AddItem "writeepd"
6048     cboFakeInput.AddItem "display"
6049     cboFakeInput.AddItem "list"
6050     cboFakeInput.AddItem "new"
6051     cboFakeInput.AddItem "setboard 1b5k/7P/p1p2np1/2P2p2/PP3P2/4RQ1R/q2r3P/6K1 w - - 0
1"
6052     cboFakeInput.AddItem "setboard
r1b2rk1/pp1nq1p1/2p1p2p/3p1p2/2PPn3/2NBPN2/PPQ2PPP/2R2RK1 b - -"

```

```

6053     cboFakeInput.AddItem "setboard 2br2k1/ppp2p1p/4p1p1/4P2q/2P1Bn2/2Q5/PP3P1P/4R1RK b
        - -"
6054     cboFakeInput.AddItem "setboard 8/8/R3k3/1R6/8/8/8/2K5 b - -"
6055     cboFakeInput.AddItem "setboard 2k4r/1pr1n3/p1p1q2p/5pp1/3P1P2/P1P1P3/1R2Q1PP/1RB3K1
        w KQkq -"
6056     cboFakeInput.AddItem "setboard 6k1/1b1nqpbp/pp4p1/5P2/1PN5/4Q3/P5PP/1B2B1K1 b - -"
6057     cboFakeInput.AddItem "perft 3"
6058     cboFakeInput.AddItem "xboard" & vbCrLf & "new" & vbCrLf & "random" & vbCrLf & "level 40 5
        0" & vbCrLf & "post"
6059     cboFakeInput.AddItem "xboard" & vbCrLf & "new" & vbCrLf & "random" & vbCrLf & "sd 4" &
        vbCrLf & "post"
6060     cboFakeInput.AddItem "time 30000" & vbCrLf & "otim 30000" & vbCrLf & "e2e4"
6061     cboFakeInput.AddItem "force" & vbCrLf & "quit"
6062     cboFakeInput.AddItem "setboard rnbqkbnr/ppp2ppp/4p3/3pP3/3P4/8/PPP2PPP/RNBQKBNR b
        KQkq -"
6063     cboFakeInput.AddItem "setboard 8/p1b1k1p1/Pp4p1/1Pp2pPp/2P2P1P/3B1K2/8/8 w - -"
6064     cboFakeInput.AddItem "setboard 8/2R5/1r3kp1/2p4p/2P2P2/p3K1P1/P6P/8 w - -"
6065     cboFakeInput.AddItem "setboard 7k/p7/6K1/5Q2/8/8/8/8 w - -"
6066     cboFakeInput.AddItem "debug1"
6067     DebugMode = True
6068     cmdTest1_Click
6069     UCIMode = True
6070 End Sub
6071
6072 Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
6073     ExitProgram
6074 End Sub
6075
6076 Private Sub Form_Resize()
6077     On Local Error Resume Next
6078
6079     ' With txtIO
6080     ' .Move .Left, .Top, Me.ScaleWidth - (.Left * 2), Me.ScaleHeight - 800
6081     ' End With
6082     '
6083     ' cboFakeInput.Width = txtIO.Width - cmdFakeInput.Width - 100
6084     ' cmdFakeInput.Left = cboFakeInput.Left + cboFakeInput.Width + 100
6085     ' On Local Error GoTo 0
6086 End Sub
6087
6088 Attribute VB_Name = "basEPD"
6089 '=====
6090 '= basEPD:
6091 '= EPD file format handling
6092 '=====
6093 Option Explicit
6094 ' Table for board indexes
6095 Private EPDTable(63) As Long
6096
6097 '-----
6098 ' ReadEPD()
6099 ' "ucinewgame" command earlier> calls INITGAME
6100 '-----
6101 Public Function ReadEPD(ByVal sEpdString As String) As Boolean
6102     Dim NumSquares As Long, i As Long
6103     Dim sChar As String
6104     Dim arCmdList() As String, p As Long
6105
6106     Fifty = 0: GameMovesCnt = 0
6107     BookMovePossible = False
6108     HintMove = EmptyMove
6109     arCmdList = Split(sEpdString)
6110     If UBound(arCmdList) < 3 Then
6111         ReadEPD = False
6112         Exit Function
6113     End If
6114
6115     For i = 0 To 63 ' Clear board

```

```

6116     Board(EPDTable(i)) = NO_PIECE
6117 Next
6118
6119 For i = 0 To MAX_BOARD: Moved(0) = 1: Next ' set unknown moved status
6120
6121 ' Part 1: Set pieces on board
6122 For i = 1 To Len(arCmdList(0))
6123     sChar = Mid$(arCmdList(0), i, 1)
6124
6125     Select Case sChar
6126     Case "P"
6127         Board(EPDTable(NumSquares)) = WPAWN
6128         NumSquares = NumSquares + 1
6129     Case "p"
6130         Board(EPDTable(NumSquares)) = BPAWN
6131         NumSquares = NumSquares + 1
6132     Case "N"
6133         Board(EPDTable(NumSquares)) = WKNIGHT
6134         NumSquares = NumSquares + 1
6135     Case "n"
6136         Board(EPDTable(NumSquares)) = BKNIGHT
6137         NumSquares = NumSquares + 1
6138     Case "K"
6139         WKingLoc = EPDTable(NumSquares)
6140         Board(WKingLoc) = WKING
6141         NumSquares = NumSquares + 1
6142     Case "k"
6143         BKingLoc = EPDTable(NumSquares)
6144         Board(BKingLoc) = BKING
6145         NumSquares = NumSquares + 1
6146     Case "R"
6147         Board(EPDTable(NumSquares)) = WROOK
6148         NumSquares = NumSquares + 1
6149     Case "r"
6150         Board(EPDTable(NumSquares)) = BROOK
6151         NumSquares = NumSquares + 1
6152     Case "Q"
6153         Board(EPDTable(NumSquares)) = WQUEEN
6154         NumSquares = NumSquares + 1
6155     Case "q"
6156         Board(EPDTable(NumSquares)) = BQUEEN
6157         NumSquares = NumSquares + 1
6158     Case "B"
6159         Board(EPDTable(NumSquares)) = WBISHOP
6160         NumSquares = NumSquares + 1
6161     Case "b"
6162         Board(EPDTable(NumSquares)) = BBISHOP
6163         NumSquares = NumSquares + 1
6164     Case "/"
6165     Case Else
6166         NumSquares = NumSquares + Val(sChar)
6167     End Select
6168
6169 Next
6170
6171 ' part 2: color to move
6172 sChar = arCmdList(1)
6173 If LCase(sChar) = "w" Then
6174     bWhiteToMove = True
6175 ElseIf LCase(sChar) = "b" Then
6176     bWhiteToMove = False
6177 Else
6178     Exit Function
6179 End If
6180 bCompIsWhite = Not bWhiteToMove
6181
6182 'Part 3: castling
6183 Moved(WKING_START) = 1: Moved(SQ_A8) = 1: Moved(SQ_A1) = 1

```

```

6184 Moved(BKING_START) = 1: Moved(SQ_H8) = 1: Moved(SQ_A8) = 1
6185
6186 For i = 1 To Len(arCmdList(2))
6187     sChar = Mid$(arCmdList(2), i, 1)
6188
6189     Select Case sChar
6190         Case "K"
6191             Moved(WKING_START) = 0
6192             Moved(SQ_H1) = 0
6193         Case "Q"
6194             Moved(WKING_START) = 0
6195             Moved(SQ_A1) = 0
6196         Case "k"
6197             Moved(BKING_START) = 0
6198             Moved(SQ_H8) = 0
6199         Case "q"
6200             Moved(BKING_START) = 0
6201             Moved(SQ_A8) = 0
6202         Case "-"
6203             Exit For
6204     End Select
6205
6206 Next
6207
6208 'Part4 : EnPassant
6209 sChar = arCmdList(3)
6210 If sChar <> "-" Then
6211     p = FileRev(Left$(sChar, 1)) + RankRev(Right$(sChar, 1))
6212     If bWhiteToMove Then
6213         If Right$(sChar, 1) = "6" Then
6214             Board(p) = BEP_PIECE: EpPosArr(1) = p
6215         End If
6216     Else
6217         If Right$(sChar, 1) = "3" Then
6218             Board(p) = WEP_PIECE: EpPosArr(1) = p
6219         End If
6220     End If
6221 End If
6222
6223 'Part5 : Fifty move half move count
6224 If UBound(arCmdList) >= 4 Then
6225     sChar = arCmdList(4)
6226     If sChar <> "" Then
6227         If Val("0" & sChar) > 0 Then Fifty = Val(sChar)
6228     End If
6229 End If
6230
6231 'Part5 : full move count: 1 before first move, 2 after first black move
6232 GameMovesCnt = 0
6233 If UBound(arCmdList) >= 5 Then
6234     sChar = arCmdList(5)
6235     If sChar <> "" Then
6236         If Val("0" & sChar) > 0 Then
6237             GameMovesCnt = GetMax(0, (Val(sChar) - 1) * 2)
6238             If Not bWhiteToMove Then GameMovesCnt = GameMovesCnt + 1
6239         End If
6240     End If
6241 End If
6242
6243 InitPieceSquares
6244 HashBoard GamePosHash(GameMovesCnt), EmptyMove ' for 3x repetition draw
6245 ReadEPD = True
6246 End Function
6247
6248 '-----
6249 'WriteEPD() -
6250 '-----
6251 Public Function WriteEPD() As String

```

```

6252 Dim i As Long
6253 Dim iPiece As Long, iEmptySquares As Long
6254 Dim sEPD As String, sRow As String
6255 Dim sEPPiece As String, sCastle As String
6256 sEPPiece = "-"
6257
6258 For i = 0 To 63
6259     If i Mod 8 = 0 And i > 0 Then
6260         sEPD = sEPD & "/" & sRow & Format$(iEmptySquares, "#")
6261         iEmptySquares = 0
6262         sRow = ""
6263     End If
6264     iPiece = Board(EPDTable(i))
6265
6266     Select Case iPiece
6267     Case NO_PIECE
6268         iEmptySquares = iEmptySquares + 1
6269     Case WEP_PIECE, BEP_PIECE
6270         sEPPiece = Chr$(File(EPDTable(i)) + 96) & Rank(EPDTable(i))
6271         iEmptySquares = iEmptySquares + 1
6272     Case Else
6273         sRow = sRow & Format$(iEmptySquares, "#") & Piece2Alpha(iPiece)
6274         iEmptySquares = 0
6275     End Select
6276
6277 Next
6278
6279 sEPD = sEPD & "/" & sRow & Format$(iEmptySquares, "#")
6280 sEPD = Right$(sEPD, Len(sEPD) - 1)
6281 If bWhiteToMove Then
6282     sEPD = sEPD & " w"
6283 Else
6284     sEPD = sEPD & " b"
6285 End If
6286 If Moved(WKING_START) = 0 Then
6287     If Moved(SQ_H1) = 0 Then sCastle = "K"
6288     If Moved(SQ_A1) = 0 Then sCastle = sCastle & "Q"
6289 End If
6290 If Moved(BKING_START) = 0 Then
6291     If Moved(SQ_H8) = 0 Then sCastle = sCastle & "k"
6292     If Moved(SQ_A8) = 0 Then sCastle = sCastle & "q"
6293 End If
6294 If sCastle = "" Then sCastle = "-"
6295 sEPD = sEPD & " " & sCastle & " " & sEPPiece
6296 sEPD = sEPD & " " & CStr(Fifty)
6297 sEPD = sEPD & " " & CStr(GameMovesCnt \ 2 + 1)
6298 WriteEPD = sEPD
6299 End Function
6300
6301 Public Sub InitEPDTable()
6302     EPDTable(0) = 91: EPDTable(1) = 92: EPDTable(2) = 93: EPDTable(3) = 94
6303     EPDTable(4) = 95: EPDTable(5) = 96: EPDTable(6) = 97: EPDTable(7) = 98
6304     EPDTable(8) = 81: EPDTable(9) = 82: EPDTable(10) = 83: EPDTable(11) = 84
6305     EPDTable(12) = 85: EPDTable(13) = 86: EPDTable(14) = 87: EPDTable(15) = 88
6306     EPDTable(16) = 71: EPDTable(17) = 72: EPDTable(18) = 73: EPDTable(19) = 74
6307     EPDTable(20) = 75: EPDTable(21) = 76: EPDTable(22) = 77: EPDTable(23) = 78
6308     EPDTable(24) = 61: EPDTable(25) = 62: EPDTable(26) = 63: EPDTable(27) = 64
6309     EPDTable(28) = 65: EPDTable(29) = 66: EPDTable(30) = 67: EPDTable(31) = 68
6310     EPDTable(32) = 51: EPDTable(33) = 52: EPDTable(34) = 53: EPDTable(35) = 54
6311     EPDTable(36) = 55: EPDTable(37) = 56: EPDTable(38) = 57: EPDTable(39) = 58
6312     EPDTable(40) = 41: EPDTable(41) = 42: EPDTable(42) = 43: EPDTable(43) = 44
6313     EPDTable(44) = 45: EPDTable(45) = 46: EPDTable(46) = 47: EPDTable(47) = 48
6314     EPDTable(48) = 31: EPDTable(49) = 32: EPDTable(50) = 33: EPDTable(51) = 34
6315     EPDTable(52) = 35: EPDTable(53) = 36: EPDTable(54) = 37: EPDTable(55) = 38
6316     EPDTable(56) = 21: EPDTable(57) = 22: EPDTable(58) = 23: EPDTable(59) = 24
6317     EPDTable(60) = 25: EPDTable(61) = 26: EPDTable(62) = 27: EPDTable(63) = 28
6318 End Sub
6319 Attribute VB_Name = "basEval"

```

```

6320 '=====
6321 '= basEval:
6322 '= EVAL function : Evaluation of board position =
6323 '=====
6324 Option Explicit
6325 '--- Game phase
6326 Const PHASE_MIDGAME As Long = 128
6327 Const PHASE_ENDGAME As Long = 0
6328 Public Const MAX_SEE_DIFF As Long = 80 ' greater than value bishop minus value Knight
6329 'Public Const TEMPO_BONUS As Long = 23 ' 20 bonus for side to move
6330 Public Const SPACE_THRESHOLD As Long = 12222 ' compute space eval for opening phase only
6331
6332 '--- Endgame eval scale factors
6333 Const SCALE_FACTOR_DRAW = 0
6334 Const SCALE_FACTOR_ONEPAWN = 48
6335 Const SCALE_FACTOR_NORMAL = 64
6336 Const SCALE_FACTOR_MAX = 128
6337 Const SCALE_FACTOR_NONE = 255
6338
6339 '--- Penalties for enemy's safe checks
6340 Const QueenCheck As Long = 780
6341 Const RookCheck As Long = 880
6342 Const BishopCheck As Long = 435
6343 Const KnightCheck As Long = 790
6344 '--- evaluation parameters
6345 Public IsolatedPenalty(1) As TScore
6346 Public BackwardPenalty(1) As TScore
6347 Public DoubledPenalty As TScore
6348 Public ConnectedBonus(1, 1, 2, 8) As TScore
6349 Public LeverBonus(8) As TScore
6350 Public ShelterWeakness(4, 8) As Long
6351 Public StormDanger(4, 4, 8) As Long
6352 Public ThreatenedByHangingPawn As TScore
6353 Public ThreatByRank As TScore
6354 Public WeakUnopposedPawn As TScore
6355 Public Hanging As TScore
6356 Public Overload As TScore
6357 Public SafeCheck As TScore
6358 Public OtherCheck As TScore
6359 Public PawnlessFlank As TScore
6360 Public ScorePawn As TScore
6361 Public ScoreKnight As TScore
6362 Public ScoreBishop As TScore
6363 Public ScoreRook As TScore
6364 Public ScoreQueen As TScore
6365 Public PieceScore(17) As Long
6366 Public PieceAbsValue(17) As Long
6367 Public PieceTypeValue(6) As Long
6368 Public WMaterial As Long
6369 Public WNonPawnMaterial As Long
6370 Public BMaterial As Long
6371 Public BNonPawnMaterial As Long
6372 Public Material As Long
6373 Public NonPawnMaterial As Long
6374 Public DrawContempt As Long
6375 Dim WAttack(MAX_BOARD) As Integer '- Fields around king: count attacks ' public+Erase is
6376 ' 2x faster than local in Eval function !
6377 Dim BAttack(MAX_BOARD) As Integer '- Fields around king: count attacks
6378 Dim WThreat As TScore, BThreat As TScore
6379 Public PiecePosScaleFactor As Long ' set in INI file
6380 Public CompKingDefScaleFactor As Long ' set in INI file
6381 Public OppKingAttScaleFactor As Long ' set in INI file
6382 Public PawnStructScaleFactor As Long ' set in INI file
6383 Public PassedPawnsScaleFactor As Long ' set in INI file
6384 Public MobilityScaleFactor As Long ' set in INI file
6385 Public ThreatsScaleFactor As Long ' set in INI file
6386 Public WKingScaleFactor As Long, BKingScaleFactor As Long
6387 Public PawnsWMax(9) As Long '--- Pawn max rank (2-7) for file A-H

```



```

6387 Public PawnsWMin(9) As Long '--- Pawn min rank (2-7) for file A-H
6388 Public WPawns(9) As Long '--- number of pawns for file A-H
6389 Public PawnsBMax(9) As Long
6390 Public PawnsBMin(9) As Long
6391 Public BPawns(9) As Long
6392 Public RootMove As TMOVE
6393 Public LastNodesCnt As Long
6394 Public LastThreadCheckNodesCnt As Long
6395 Public StaticEvalArr(MAX_PV) As Long ' Eval history
6396 Public TestCnt(20) As Long '--- Counter for special debug cases
6397 Public MidGameLimit As Long
6398 Public EndgameLimit As Long
6399 '-----
6400 '--- Piece square tables: value for piece type on specific board position
6401 '-----
6402 Public PsqtWP(MAX_BOARD) As TScore
6403 Public PsqtBP(MAX_BOARD) As TScore
6404 Public PsqtWB(MAX_BOARD) As TScore
6405 Public PsqtBB(MAX_BOARD) As TScore
6406 Public PsqtWN(MAX_BOARD) As TScore
6407 Public PsqtBN(MAX_BOARD) As TScore
6408 Public PsqtWQ(MAX_BOARD) As TScore
6409 Public PsqtBQ(MAX_BOARD) As TScore
6410 Public PsqtWR(MAX_BOARD) As TScore
6411 Public PsqtBR(MAX_BOARD) As TScore
6412 Public PsqtWK(MAX_BOARD) As TScore
6413 Public PsqtBK(MAX_BOARD) As TScore
6414 Public PsqVal(1, 16, MAX_BOARD) As Long ' piece square score for piece: (endgame,piece,square)
6415 '-----
6416 '--- Mobility values for pieces
6417 '-----
6418 Public MobilityN(9) As TScore
6419 Public MobilityB(15) As TScore
6420 Public MobilityR(15) As TScore
6421 Public MobilityQ(29) As TScore
6422 '-----
6423 Public ZeroScore As TScore
6424 Public ThreatBySafePawn(5) As TScore
6425 Public OutpostBonusKnight(1) As TScore
6426 Public OutpostBonusBishop(1) As TScore
6427 Public ReachableOutpostKnight(1) As TScore
6428 Public ReachableOutpostBishop(1) As TScore
6429 Public KingAttackWeights(6) As Long
6430 Public QueenMinorsImbalance(12) As Long
6431 Public WBestPawnVal As Long, BBestPawnVal As Long, WBestPawn As Long,
    BBestPawn As Long
6432 Public GamePhase As Long
6433 Public WKingAttackersWeight As Long, WKingAttackersCount As Long,
    BKingAttackersWeight As Long, BKingAttackersCount As Long
6434 Public bEvalTrace As Boolean
6435 Public bTimeTrace As Boolean
6436 Public bHashTrace As Boolean
6437 Public bWinboardTrace As Boolean
6438 Public bWbPvInUciFormat As Boolean
6439 Public bThreadTrace As Boolean
6440 Dim PassedPawns(16) As Long ' List of passed pawns (Square)
6441 Dim PassedPawnsCnt As Long
6442 Dim WPassedPawnAttack As Long, BPassedPawnAttack As Long
6443 Public PushClose(8) As Long
6444 Public PushAway(8) As Long
6445 Public PushToEdges(MAX_BOARD) As Long
6446 Public WOutpostSq(MAX_BOARD) As Boolean
6447 Public BOutpostSq(MAX_BOARD) As Boolean
6448 'endgame
6449 Public KRPPKRP_SFactor(8) As Long
6450
6451 '--- Threat list
6452 Dim ThreatCnt As Long

```

```

6453 Public Type TThreatList
6454     HangCol           As enumColor
6455     HangPieceType     As Long
6456     AttackerPieceType As Long
6457     AttackerSquare    As Long
6458     AttackedSquare    As Long
6459 End Type
6460 Dim ThreatList(32) As TThreatList
6461 ' Pawn Eval
6462 Dim Passed As Boolean, Opposed As Boolean, Backward As Boolean
6463 Dim Neighbours As Boolean, Doubled As Boolean, Lever As Long,
Supported As Long, Phalanx As Long, LeverPush As Long
6464 Public PassedPawnFileBonus(8) As TScore
6465 Public PassedPawnRankBonus(8) As TScore
6466 Public PassedDanger(8) As Long
6467 Private OwnAttCnt As Long
6468 ' Threats
6469 Public ThreatByMinor(6) As TScore ' Attacker is defended minor (B/N)
6470 Public ThreatByRook(6) As TScore
6471 Public ThreatByAttackOnQueen As TScore
6472 Public KingOnOneBonus As TScore
6473 Public KingOnManyBonus As TScore
6474 ' King protection
6475 Public KingProtector(5) As TScore
6476 ' Material imbalance (SF6)
6477 Public QuadraticOurs(5, 5) As Long
6478 Public QuadraticTheirs(5, 5) As Long
6479 Public PawnSet(8) As Long
6480 Public ImbPieceCount(COL_WHITE, 5) As Long
6481 Private bWIsland As Boolean, bBIsland As Boolean
6482 Private PieceSqlList(15, 10) As Integer ' <Piece type> <list number> Square List of pieces for
multiple runs thorough piece list
6483 Private PieceSqlListCnt(15) As Integer ' counter for PieceLoc
6484 ' temp
6485 Private bIniReadDone As Boolean
6486 'Public RootSimpleEval As Long
6487
6488
6489 '-----
6490 'InitEval(ThreatMove) Set piece values and piece square tables
6491 '-----
6492 Public Sub InitEval()
6493     Dim Score As Long, bSaveEvalTrace As Boolean
6494     ZeroScore.MG = 0: ZeroScore.EG = 0
6495     '--- Limit high eval values ( VERY important for playing style!)
6496     If Not bIniReadDone Then
6497         bIniReadDone = True
6498         '--- Default used if INI file is missing
6499         PiecePosScaleFactor = Val(ReadINISetting("POSITION_FACTOR", "100"))
6500         MobilityScaleFactor = Val(ReadINISetting("MOBILITY_FACTOR", "100"))
6501         PawnStructScaleFactor = Val(ReadINISetting("PAWNSTRUCT_FACTOR", "100"))
6502         PassedPawnsScaleFactor = Val(ReadINISetting("PASSEDPAWNS_FACTOR", "120"))
6503         ThreatsScaleFactor = Val(ReadINISetting("THREATS_FACTOR", "100"))
6504         OppKingAttScaleFactor = Val(ReadINISetting("OPPKINGATT_FACTOR", "100"))
6505         CompKingDefScaleFactor = Val(ReadINISetting("COMPKINGDEF_FACTOR", "100"))
6506     '
6507     '--- Piece values MG=midgame / EG=endgame58
6508     '--- SF6 values ( scale to centipawns: 1256 )
6509     '
6510     ScorePawn.MG = Val(ReadINISetting("PAWN_VAL_MG", "142"))
6511     ScorePawn.EG = Val(ReadINISetting("PAWN_VAL_EG", "207"))
6512     ScoreKnight.MG = Val(ReadINISetting("KNIGHT_VAL_MG", "784"))
6513     ScoreKnight.EG = Val(ReadINISetting("KNIGHT_VAL_EG", "868"))
6514     ScoreBishop.MG = Val(ReadINISetting("BISHOP_VAL_MG", "828"))
6515     ScoreBishop.EG = Val(ReadINISetting("BISHOP_VAL_EG", "916"))
6516     ScoreRook.MG = Val(ReadINISetting("ROOK_VAL_MG", "1286"))
6517     ScoreRook.EG = Val(ReadINISetting("ROOK_VAL_EG", "1378"))
6518     ScoreQueen.MG = Val(ReadINISetting("QUEEN_VAL_MG", "2528"))

```

```

6519     ScoreQueen.EG = Val(ReadINISetting("QUEEN_VAL_EG", "2698"))
6520     MidGameLimit = Val(ReadINISetting("MIDGAME_LIMIT", "15258")) ' for game phase
6521     EndgameLimit = Val(ReadINISetting("ENDGAME_LIMIT", "3915")) ' for game phase
6522     ' Draw contempt in centipawns > scale to SF (needs ScorePawn.EG set)
6523     DrawContempt = Val(ReadINISetting(CONTEMPT_KEY, "1"))
6524     DrawContempt = Eval100ToSF(DrawContempt) ' in centipawns
6525 End If
6526 '--- Detect endgame stage ---
6527 bSaveEvalTrace = bEvalTrace: bEvalTrace = False ' Save trace setting, trace not needed here
        before init done
6528 Score = Eval() ' Set material,NonPawnMaterial for GamePhase calculation
6529 bEvalTrace = bSaveEvalTrace
6530 SetGamePhase NonPawnMaterial ' Set GamePhase, PieceValues, bEndGame
6531 InitPieceValue
6532 InitReductionArray
6533 InitConnectedPawns
6534 InitOutpostSq
6535 End Sub
6536
6537 Public Sub InitPieceValue()
6538     '--- Piece values, always absolut, positive value
6539     PieceAbsValue(FRAME) = 0
6540     PieceAbsValue(WPAWN) = ScorePawn.MG: PieceAbsValue(BPAWN) = ScorePawn.MG
6541     PieceAbsValue(WKNIGHT) = ScoreKnight.MG: PieceAbsValue(BKNIGHT) = ScoreKnight.MG
6542     PieceAbsValue(WBISHOP) = ScoreBishop.MG: PieceAbsValue(BBISHOP) = ScoreBishop.MG
6543     PieceAbsValue(WROOK) = ScoreRook.MG: PieceAbsValue(BROOK) = ScoreRook.MG
6544     PieceAbsValue(WQUEEN) = ScoreQueen.MG: PieceAbsValue(BQUEEN) = ScoreQueen.MG
6545     PieceAbsValue(WKING) = 5000: PieceAbsValue(BKING) = 5000
6546     PieceAbsValue(13) = 0: PieceAbsValue(14) = 0
6547     PieceAbsValue(WEP_PIECE) = ScorePawn.MG: PieceAbsValue(BEP_PIECE) = ScorePawn.MG
6548     '--- Piece SCore: positive for White, negative for Black
6549     PieceScore(FRAME) = 0
6550     PieceScore(WPAWN) = ScorePawn.MG: PieceScore(BPAWN) = -ScorePawn.MG
6551     PieceScore(WKNIGHT) = ScoreKnight.MG: PieceScore(BKNIGHT) = -ScoreKnight.MG
6552     PieceScore(WBISHOP) = ScoreBishop.MG: PieceScore(BBISHOP) = -ScoreBishop.MG
6553     PieceScore(WROOK) = ScoreRook.MG: PieceScore(BROOK) = -ScoreRook.MG
6554     PieceScore(WQUEEN) = ScoreQueen.MG: PieceScore(BQUEEN) = -ScoreQueen.MG
6555     PieceScore(WKING) = 5000: PieceScore(BKING) = -PieceScore(WKING)
6556     PieceScore(13) = 0: PieceScore(14) = 0
6557     PieceScore(WEP_PIECE) = ScorePawn.MG: PieceScore(BEP_PIECE) = -ScorePawn.MG
6558     PieceTypeValue(PT_PAWN) = ScorePawn.MG
6559     PieceTypeValue(PT_KNIGHT) = ScoreKnight.MG
6560     PieceTypeValue(PT_BISHOP) = ScoreBishop.MG
6561     PieceTypeValue(PT_ROOK) = ScoreRook.MG
6562     PieceTypeValue(PT_QUEEN) = ScoreQueen.MG
6563     PieceTypeValue(PT_KING) = PieceScore(WKING)
6564 End Sub
6565
6566 Public Function SetGamePhase(ByVal NonPawnMaterial As Long) As Long
6567     Debug.Assert NonPawnMaterial >= 0
6568     NonPawnMaterial = GetMax(EndgameLimit, GetMin(NonPawnMaterial, MidGameLimit))
6569     GamePhase = (((NonPawnMaterial - EndgameLimit) * PHASE_MIDGAME) / (MidGameLimit -
        EndgameLimit))
6570     bEndgame = (GamePhase <= PHASE_ENDGAME)
6571 End Function
6572
6573 '-----
6574 '--- Eval() - Evaluation of position
6575 '--- Returns value from view of side to move (positive if black to move and black is better)
6576 '--- Value scaled to stockfish pawn endgame value (258 = 1 pawn)
6577 '---
6578 '--- Steps:
6579 '--- Init: inits attacks arrays, pawn arrays, material values for pieces
6580 '--- Check material draw or special endgame positions
6581 '--- STEPS:
6582 '--- 1. Loop over all pieces to fill pawn structure array and pawn threats
6583 '--- 2. Loop over all pieces types: evaluate each piece except kings.
6584 '--- do a move generation to calculate mobility, attackers, defenders. fill attack array with piece bitcode

```

```

6585 '--- 3. Pass for pawn push (locate here because full attack info needed)
6586 '--- 4. Calculate king safety ( shelter, pawn storm, check attacks ), king distance to best pawn
6587 '--- 5. Calculate threats
6588 '--- 6. Calculate trapped bishops, passed pawns, center control, pawn islands
6589 '--- 7. Calculate total material values and endgame scale factors
6590 '--- 8. Calculate weights and total eval
6591 '--- Add all evaluation terms weighted by variables set in INI file:
6592 '--- Material + Position(general) + PawnStructure + PassedPawns + Mobility +
6593 '--- KingSafetyComputer + KingSafetyOpponent + Threats
6594 '--- 9. Invert score for black to move
6595 '--- 10. Add tempo value for side to move
6596 '-----
6597 Public Function Eval() As Long
6598 Dim a As Long, i As Long, Square As Long, Target As Long,
Offset As Long, MobCnt As Long, r As Long, rr As Long, AttackBit As Long, k As Long,
ForkCnt As Long, SC As TScore
6599 Dim WPos As TScore, BPos As TScore, WPassed As TScore, BPassed As
TScore, WMobility As TScore, BMobility As TScore
6600 Dim WPawnStruct As TScore, BPawnStruct As TScore, Piece As Long,
WPawnCnt As Long, BPawnCnt As Long
6601 Dim WKSafety As TScore, BKSafety As TScore, bDoWKSafety As Boolean,
bDoBKSafety As Boolean
6602 Dim WKingAdjacentZoneAttCnt As Long, BKingAdjacentZoneAttCnt As Long, WKingAttPieces
As Long, BKingAttPieces As Long
6603 Dim KingDanger As Long, Undefended As Long, RankNum As Long, RelRank As
Long, QueenWeak As Boolean
6604 Dim FileNum As Long, MinWKingPawnDistance As Long,
MinBKingPawnDistance As Long, KingSidePawns As Long, QueenSidePawns As Long
6605 Dim DefByPawn As Long, AttByPawn As Long, bAllDefended As Boolean,
BlockSqDefended As Boolean, WPinnedCnt As Long, BPinnedCnt As Long, WKDefender As
Long, BKDefender As Long
6606 Dim RankPath As Long, sq As Long, WSemiOpenFiles As Long, BSemiOpenFiles
As Long
6607 Dim BlockSq As Long, MBonus As Long, EBonus As Long, UnsafeCnt As
Long, PieceAttackBit As Long
6608 Dim OwnCol As Long, OppCol As Long, MoveUp As Long, OwnKingLoc As
Long, OppKingLoc As Long, BlockSqUnsafe As Boolean
6609 Dim WBishopsOnBlackSq As Long, WBishopsOnWhiteSq As Long, BBishopsOnBlackSq As
Long, BBishopsOnWhiteSq As Long, WCenterPawnsBlocked As Long, BCenterPawnsBlocked
As Long
6610 Dim WPawnCntOnWhiteSq As Long, BPawnCntOnWhiteSq As Long, WWeakUnopposedCnt As
Long, BWeakUnopposedCnt As Long
6611 Dim WKingFile As Long, BKingFile As Long, WFrontMostPassedPawnRank As
Long, BFrontMostPassedPawnRank As Long, ScaleFactor As Long
6612 Dim WChecksCounted As Long, BChecksCounted As Long, WUnsafeChecks As Long,
BUnsafeChecks As Long, KingLevers As Long
6613 'Dim bLazy As Boolean, SimpleEval As Long
6614 '
6615 '----- Init Eval
6616 '
6617
6618 If bEvalTrace Then WriteTrace "----- Start Eval -----"
6619 EvalCnt = EvalCnt + 1
6620 Eval = 0
6621 WPawnCnt = PieceCnt(WPAWN): BPawnCnt = PieceCnt(BPAWN)
6622 WKingFile = File(WKingLoc): BKingFile = File(BKingLoc)
6623 WNonPawnMaterial = PieceCnt(WQUEEN) * ScoreQueen.MG + PieceCnt(WROOK) * ScoreRook.MG
+ PieceCnt(WBISHOP) * ScoreBishop.MG + PieceCnt(WKNIGHT) * ScoreKnight.MG
6624 WMaterial = WNonPawnMaterial + WPawnCnt * ScorePawn.MG
6625 BNonPawnMaterial = PieceCnt(BQUEEN) * ScoreQueen.MG + PieceCnt(BROOK) * ScoreRook.MG
+ PieceCnt(BBISHOP) * ScoreBishop.MG + PieceCnt(BKNIGHT) * ScoreKnight.MG
6626 BMaterial = BNonPawnMaterial + BPawnCnt * ScorePawn.MG
6627 NonPawnMaterial = WNonPawnMaterial + BNonPawnMaterial
6628 Material = WMaterial - BMaterial
6629 SetGamePhase NonPawnMaterial
6630
6631 ' Lazy eval?
6632 ' SimpleEval = ScorePawn.EG * (PieceCnt(WPAWN) - PieceCnt(BPAWN)) + (WNonPawnMaterial -

```

```

6633 BNonPawnMaterial)
6634 '
6635 ' bLazy = (Abs(SimpleEval) >= ScoreRook.EG + ScoreBishop.EG + Abs(FinalScore) + Abs(RootSimpleEval))
6636 ' If bLazy Then
6637 '   SimpleEval = SimpleEval + (Nodes And 7) - 3
6638 '   If Not bWhiteToMove Then SimpleEval = -SimpleEval
6639 '   Eval = SimpleEval
6640 '   TestCnt(1) = TestCnt(1) + 1
6641 '   Exit Function
6642 ' End If
6643
6644 'Debug.Assert PieceSqlListCnt(WPAWN) = PieceCnt(WPAWN)
6645 'Debug.Assert PieceSqlListCnt(BPAWN) = PieceCnt(BPAWN)
6646 '
6647 '--- Endgame function available?
6648 '
6649 Select Case WPawnCnt + BPawnCnt
6650 Case 0 ' no pawns
6651   ' KQKR
6652   If (WMaterial = ScoreQueen.MG And BMaterial = ScoreRook.MG) Or (BMaterial =
6653     ScoreQueen.MG And WMaterial = ScoreRook.MG) Then
6654     Eval = Eval_KQKR(): GoTo lblEndEval
6655   End If
6656   '--- Insufficient material draw?
6657   If IsMaterialDraw() Then
6658     Eval = 0: Exit Function '- Endgame draw: not sufficient material for mate
6659   End If
6660 Case 1 ' one pawn
6661   If (WMaterial = ScoreRook.MG And BMaterial = ScorePawn.MG) Or (BMaterial =
6662     ScoreRook.MG And WMaterial = ScorePawn.MG) Then
6663     Eval = Eval_KRKP(): GoTo lblEndEval ' KRKP
6664   ElseIf (WMaterial = ScoreQueen.MG And BMaterial = ScorePawn.MG) Or (BMaterial =
6665     ScoreQueen.MG And WMaterial = ScorePawn.MG) Then
6666     Eval = Eval_KQKP(): GoTo lblEndEval ' KQKP
6667   End If
6668 End Select
6669
6670 '----- Init Eval -----
6671 WBestPawnVal = VALUE_NONE: WBestPawn = 0
6672 BBestPawnVal = VALUE_NONE: BBestPawn = 0
6673 WPassedPawnAttack = 0: BPassedPawnAttack = 0
6674 ThreatCnt = 0: WThreat = ZeroScore: BThreat = ZeroScore
6675
6676 '--- Fill Pawn Arrays: number of pawns in file
6677 Erase WPawns: Erase BPawns: Erase PawnsWMax: Erase PawnsBMax
6678 For a = 0 To 9
6679   PawnsWMin(a) = 9: PawnsBMin(a) = 9
6680 Next
6681
6682 WPawns(0) = -1: BPawns(0) = -1
6683 WPawns(9) = -1: BPawns(9) = -1
6684 PassedPawnsCnt = 0
6685 Erase WAttack(): Erase BAttack() 'Init attack arrays (fast)
6686 Erase PieceSqlListCnt()
6687 MinWKingPawnDistance = 9: MinBKingPawnDistance = 9
6688
6689 '--- Step 1. loop over pieces: count pieces for material totals and game phase calculation. add piece square table
6690 '--- score.
6691 '----- calc pawn min/max rank positions per file; pawn attacks(for mobility used later)
6692
6693 For a = 1 To NumPieces
6694   Square = Pieces(a): If Square = 0 Or Board(Square) >= NO_PIECE Then GoTo
6695     lblNextPieceCnt
6696   r = Board(Square): PieceSqlListCnt(r) = PieceSqlListCnt(r) + 1: PieceSqlList(r,
6697     PieceSqlListCnt(r)) = Square ' fill piece list
6698
6699   Select Case r

```

```

6694 Case WPAWN
6695 WAttack(Square + SQ_UP_LEFT) = WAttack(Square + SQ_UP_LEFT) Or PLAttackBit:
WAttack(Square + SQ_UP_RIGHT) = WAttack(Square + SQ_UP_RIGHT) Or PRAttackBit
' Set pawn attack here for use in pieces eval
6696 FileNum = File(Square): RankNum = Rank(Square): WPawns(FileNum) = WPawns(
FileNum) + 1
6697 If RankNum < PawnsWMin(FileNum) Then PawnsWMin(FileNum) = RankNum
6698 If RankNum > PawnsWMax(FileNum) Then PawnsWMax(FileNum) = RankNum
6699 If MaxDistance(WKingLoc, Square) < MinWKingPawnDistance Then
MinWKingPawnDistance = MaxDistance(WKingLoc, Square)
6700 If ColorSq(Square) = COL_WHITE Then WPawnCntOnWhiteSq = WPawnCntOnWhiteSq + 1
' for Bishop eval
6701 ' If FileNum < FILE_E Then QueenSidePawns = QueenSidePawns + 1 Else KingSidePawns =
KingSidePawns + 1
6702 Case BPAWN
6703 BAttack(Square + SQ_DOWN_LEFT) = BAttack(Square + SQ_DOWN_LEFT) Or PLAttackBit
: BAttack(Square + SQ_DOWN_RIGHT) = BAttack(Square + SQ_DOWN_RIGHT) Or
PRAttackBit
6704 FileNum = File(Square): RankNum = Rank(Square): BPawns(FileNum) = BPawns(
FileNum) + 1
6705 If RankNum < PawnsBMin(FileNum) Then PawnsBMin(FileNum) = RankNum
6706 If RankNum > PawnsBMax(FileNum) Then PawnsBMax(FileNum) = RankNum
6707 If MaxDistance(BKingLoc, Square) < MinBKingPawnDistance Then
MinBKingPawnDistance = MaxDistance(BKingLoc, Square)
6708 If ColorSq(Square) = COL_WHITE Then BPawnCntOnWhiteSq = BPawnCntOnWhiteSq + 1
' for Bishop eval
6709 ' If FileNum < FILE_E Then QueenSidePawns = QueenSidePawns + 1 Else KingSidePawns =
KingSidePawns + 1
6710 End Select
6711
6712 lblNextPieceCnt:
6713 Next
6714
6715 '--- KPK endgame: Eval if promoted pawn cannot be captured
6716 If NonPawnMaterial = 0 And (WPawnCnt + BPawnCnt = 1) Then
6717 If WPawnCnt = 1 Then
6718 sq = PieceSqList(WPAWN, 1)
6719 If File(sq) = FILE_A Or File(sq) = FILE_H Then
6720 If File(BKingLoc) = File(sq) And Rank(BKingLoc) > Rank(sq) Then Eval = 0:
GoTo lblEndEval
6721 End If
6722
6723 If bWhiteToMove Then
6724 If Rank(sq) = 7 Then
6725 If sq + SQ_UP <> WKingLoc Then ' own king not at promote square
6726 If MaxDistance(BKingLoc, sq + SQ_UP) > 1 Or MaxDistance(WKingLoc, sq +
SQ_UP) = 1 Then
6727 Eval = VALUE_KNOWN_WIN: GoTo lblEndEval
6728 End If
6729 End If
6730 End If
6731 '--- Draw if opp king 2 rows in front of pawn (not at rank 8) and own king behind
6732 If Rank(BKingLoc) <> 8 Then
6733 If BKingLoc >= sq + SQ_UP + SQ_UP_LEFT And BKingLoc <= sq + SQ_UP +
SQ_UP_RIGHT Then
6734 If WKingLoc >= sq + SQ_DOWN_LEFT And WKingLoc <= sq + SQ_DOWN_RIGHT Then
6735 Eval = 0: GoTo lblEndEval
6736 End If
6737 End If
6738 End If
6739 Else
6740 sq = PieceSqList(BPAWN, 1)
6741 If File(sq) = FILE_A Or File(sq) = FILE_H Then
6742 If File(WKingLoc) = File(sq) And Rank(WKingLoc) < Rank(sq) Then Eval = 0:
GoTo lblEndEval
6743 End If
6744

```



```

6745     If Not bWhiteToMove Then
6746         If Rank(sq) = 2 Then
6747             If sq + SQ_DOWN <> BKingLoc Then ' own king not at promote square
6748                 If MaxDistance(WKingLoc, sq + SQ_DOWN) > 1 Or MaxDistance(BKingLoc, sq +
SQ_DOWN) = 1 Then
6749                     Eval = -VALUE_KNOWN_WIN: GoTo lblEndEval
6750                 End If
6751             End If
6752         End If
6753         '--- Draw if opp king in front of pawn (not at rank 1) and own king behind
6754         If Rank(WKingLoc) <> 1 Then
6755             If WKingLoc >= sq + SQ_DOWN + SQ_DOWN_LEFT And WKingLoc <= sq + SQ_DOWN +
SQ_DOWN_RIGHT Then
6756                 If BKingLoc >= sq + SQ_UP_LEFT And BKingLoc <= sq + SQ_UP_RIGHT Then Eval
= 0: GoTo lblEndEval
6757             End If
6758         End If
6759     End If
6760 End If
6761 End If
6762 '
6763 '--- King safety needed?
6764 '
6765 bDoWKSafety = CBool(BNonPawnMaterial >= ScoreQueen.MG)
6766 bDoBKSafety = CBool(WNonPawnMaterial >= ScoreQueen.MG)
6767 WKingAttackersCount = 0: WKingAttackersWeight = 0: BKingAttackersCount = 0:
BKingAttackersWeight = 0
6768 '--- King Position
6769 WKSafety = ZeroScore: BKSafety = ZeroScore
6770 If WNonPawnMaterial > 0 And BMaterial = 0 Then
6771     WPos.EG = WPos.EG + (7 - MaxDistance(BKingLoc, WKingLoc)) * 12 ' follow opp king to edge
for mate (KRK, KQK)
6772     BPos.EG = BPos.EG + PsqtBK(BKingLoc).EG
6773 ElseIf BNonPawnMaterial > 0 And WMaterial = 0 Then
6774     BPos.EG = BPos.EG + (7 - MaxDistance(WKingLoc, BKingLoc)) * 12
6775     WPos.EG = WPos.EG + PsqtWK(WKingLoc).EG
6776 Else
6777     AddScore WPos, PsqtWK(WKingLoc)
6778     AddScore BPos, PsqtBK(BKingLoc)
6779 End If
6780
6781 '-----
6782 '--- Step 2: EVAL Loop over pieces -----
6783 '-----
6784 '
6785 '-----
6786 '--- WHITE PAWNS -----
6787 '-----
6788 For a = 1 To PieceSqListCnt(WPAWN)
6789     Square = PieceSqList(WPAWN, a): FileNum = File(Square): RankNum = Rank(Square):
RelRank = RankNum: SC.MG = 0: SC.EG = 0
6790     WPos.MG = WPos.MG + PsqtWP(Square).MG: WPos.EG = WPos.EG + PsqtWP(Square).EG
6791     DefByPawn = AttackBitCnt(WAttack(Square) And PAttackBit) ' counts 1 or 2 pawns
6792     AttByPawn = AttackBitCnt(BAttack(Square) And PAttackBit) ' counts 1 or 2 pawns
6793
6794     If bEndgame And RankNum > 4 Then If MaxDistance(Square, BKingLoc) = 1 Then SC.EG =
SC.EG + 10 ' advanced pawn supported by king
6795     'If BPawns(FileNum) = 0 Then WSemiOpenFiles = WSemiOpenFiles + 12 \ WPawns(FileNum) ' only count
once per file, so 12 \ WPawns(FileNum) works for 1,2,3,4 pawns
6796     Opposed = (BPawns(FileNum) > 0) And RankNum < PawnsBMax(FileNum)
6797     Lever = AttByPawn
6798     Supported = DefByPawn
6799     LeverPush = AttackBitCnt(BAttack(Square + SQ_UP) And PAttackBit)
6800     Doubled = (Board(Square + SQ_DOWN) = WPAWN) 'not SQ_UP!
6801     Neighbours = (WPawns(FileNum + 1) > 0 Or WPawns(FileNum - 1) > 0)
6802     Phalanx = AttackBitCnt(WAttack(Square + SQ_UP) And PAttackBit)
6803     '
6804     If Not Neighbours Or Lever Or RelRank >= 5 Then

```



```

6805     Backward = False
6806 Else
6807     r = GetMin(PawnsWMin(FileNum - 1), PawnsWMin(FileNum + 1))
6808     If r <= RankNum Then
6809         Backward = False
6810     Else
6811         Backward = True
6812         If r = RankNum + 1 Then ' can safely advance to not backward rank?
6813             If LeverPush = 0 Then If Board(Square + SQ_UP) <> BPAWN Then Backward =
6814                 False
6815             End If
6816         End If
6817     End If
6818 ' Blocked pawn on center files? Needed for bishop eval
6819 If FileNum >= FILE_C Then If FileNum <= FILE_F Then If Board(Square + SQ_UP) <
6820     NO_PIECE Then WCenterPawnsBlocked = WCenterPawnsBlocked + 1
6821 '
6822 '----- Passed pawn?
6823 '
6824 Passed = False
6825 If Doubled Then GoTo lblEndWPassed
6826
6827 ' Stopper two or more ranks in front?
6828 For k = -1 To 1
6829     If PawnsBMax(FileNum + k) > RankNum + 1 Then GoTo lblEndWPassed
6830 Next k
6831 If Board(Square + SQ_UP) = BPAWN Then
6832     ' phalanx neighbour can capture block opp pawn and became a passer
6833     If Phalanx > LeverPush Then If Supported >= Lever And RankNum >= 5 And
6834         bWhiteToMove Then Passed = True: GoTo lblEndWPassed
6835 Else
6836     If AttByPawn = 0 Then
6837         Passed = True: GoTo lblEndWPassed
6838     ElseIf Phalanx >= LeverPush Then
6839         ' debug.print printpos, LocCoord(square)
6840         If Supported >= Lever Then Passed = True: GoTo lblEndWPassed
6841     End If
6842 End If
6843 '
6844 If Not Passed And Supported > 0 And RankNum >= 5 Then ' sacrifice supporter pawn to create
6845     passer?
6846     If PawnsBMax(FileNum) = RankNum + 1 Then ' blocker pawn
6847         If PawnsBMax(FileNum - 1) < RankNum Then ' no other stopper left side
6848             If CBool(WAttack(Square) And PRAttackBit) Then ' left side supporter pawn (attacks to
6849                 right)
6850                 If Board(Square + SQ_LEFT) >= NO_PIECE Then ' can move forward to attack stopper
6851                     If Not CBool(BAttack(Square + SQ_LEFT) And PRAttackBit) Then ' no second left
6852                         to right attacker from file-2
6853                     Passed = True: GoTo lblEndWPassed
6854                 End If
6855             End If
6856         End If
6857     End If
6858     If Not Passed Then
6859         If PawnsBMax(FileNum + 1) < RankNum Then
6860             If CBool(WAttack(Square) And PLAttackBit) Then ' right side supporter pawn (attacks
6861                 from left)
6862                 If Board(Square + SQ_RIGHT) >= NO_PIECE Then ' can move forward to attack
6863                     stopper
6864                     If Not CBool(BAttack(Square + SQ_RIGHT) And PLAttackBit) Then ' no
6865                         second right to left attacker
6866                     Passed = True: GoTo lblEndWPassed
6867                 End If
6868             End If
6869         End If
6870     End If

```

```

6864         End If
6865     End If
6866 End If
6867 lblEndWPassed:
6868
6869 '--- pawn score
6870 If Lever Then AddScore SC, LeverBonus(RelRank)
6871 If Supported Or Phalanx Then 'Connected
6872     AddScore SC, ConnectedBonus(Abs(Opposed), Abs(Phalanx <> 0), DefByPawn, RelRank)
6873 ElseIf Not Neighbours Then
6874     MinusScore SC, IsolatedPenalty(Abs(Opposed))
6875     If Not Opposed Then WWeakUnopposedCnt = WWeakUnopposedCnt + 1
6876 ElseIf Backward Then
6877     MinusScore SC, BackwardPenalty(Abs(Opposed))
6878 End If
6879 If Doubled And Supported = 0 Then MinusScore SC, DoubledPenalty
6880 '-----
6881 If bEndgame Then
6882     If FileNum = 1 Or FileNum = 8 Then AddScore SC, PsqtWP(Square)
6883     If WPawnCnt = 1 Then SC.EG = SC.EG + 2 * RelRank * RelRank
6884     If SC.EG + PsqtWP(Square).EG > WBestPawnVal Then
6885         WBestPawnVal = SC.EG + PsqtWP(Square).EG: WBestPawn = Square
6886     ElseIf SC.EG = WBestPawnVal Then
6887         If WBestPawn = 0 Or MaxDistance(Square, WKingLoc) < MaxDistance(WBestPawn,
        WKingLoc) Then
6888             WBestPawnVal = SC.EG: WBestPawn = Square
6889         End If
6890     End If
6891 End If
6892 ' Passed : eval later when full attack is available
6893 If Passed Then
6894     PassedPawnsCnt = PassedPawnsCnt + 1: PassedPawns(PassedPawnsCnt) = Square
6895     If RankNum > 4 Then If Abs(FileNum - BKingFile) <= 2 Then WPassedPawnAttack =
        WPassedPawnAttack + 1
6896 End If
6897 '
6898 AddScore WPawnStruct, SC
6899 If bEvalTrace Then WriteTrace "WPawn: " & LocCoord(Square) & ">" & SC.MG & ", " &
        SC.EG
6900 Next a
6901
6902 '-----
6903 '--- BLACK PAWNS -----
6904 '-----
6905 For a = 1 To PieceSqlListCnt(BPAWN)
6906     Square = PieceSqlList(BPAWN, a): FileNum = File(Square): RankNum = Rank(Square):
        RelRank = (9 - RankNum): SC.MG = 0: SC.EG = 0
6907     'Debug.Assert Board(Square) = BPAWN
6908     BPos.MG = BPos.MG + PsqtBP(Square).MG: BPos.EG = BPos.EG + PsqtBP(Square).EG
6909     DefByPawn = AttackBitCnt(BAttack(Square) And PAttackBit) 'counts 1 or 2 pawns
6910     AttByPawn = AttackBitCnt(WAttack(Square) And PAttackBit) 'counts 1 or 2 pawns
6911
6912     If bEndgame And RelRank > 4 Then If MaxDistance(Square, WKingLoc) = 1 Then SC.EG =
        SC.EG + 10 'advanced pawn supported by king
6913     'If WPawns(FileNum) = 0 Then BSemiOpenFiles = BSemiOpenFiles + 12 \ BPawns(FileNum)
6914     Opposed = RankNum > PawnsWMin(FileNum) ' PawnsWMin=9 if no pawn
6915     Lever = AttByPawn
6916     Supported = DefByPawn
6917     LeverPush = AttackBitCnt(WAttack(Square + SQ_DOWN) And PAttackBit)
6918     Doubled = Abs(Board(Square + SQ_UP) = BPAWN)
6919     Neighbours = (BPawns(FileNum + 1) > 0 Or BPawns(FileNum - 1) > 0)
6920     Phalanx = AttackBitCnt(BAttack(Square + SQ_DOWN) And PAttackBit)
6921
6922     If Not Neighbours Or Lever Or RelRank >= 5 Then
6923         Backward = False
6924     Else
6925         r = GetMax(PawnsBMax(FileNum - 1), PawnsBMax(FileNum + 1))
6926         If r >= RankNum Then

```

```

6927         Backward = False
6928     Else
6929         Backward = True
6930         If r = RankNum - 1 Then ' can safely advance to not backward rank?
6931             If LeverPush = 0 Then If Board(Square + SQ_DOWN) <> WPAWN Then Backward =
                False
6932             End If
6933         End If
6934     End If
6935
6936     ' Blocked pawn on center files? Needed for bishop eval
6937     If FileNum >= FILE_C Then If FileNum <= FILE_F Then If Board(Square + SQ_DOWN) <
        NO_PIECE Then BCenterPawnsBlocked = BCenterPawnsBlocked + 1
6938
6939     '
6940     '----- Passed pawn?
6941     '
6942     Passed = False
6943     If Doubled Then GoTo lblEndBPassed
6944
6945     ' Stopper two or more ranks in front
6946     For k = -1 To 1
6947         If PawnsWMin(FileNum + k) < RankNum - 1 Then GoTo lblEndBPassed
6948     Next k
6949     If Board(Square - SQ_UP) = WPAWN Then
6950         If Phalanx > LeverPush Then If Supported >= Lever And RankNum <= 4 And Not
            bWhiteToMove Then Passed = True: GoTo lblEndBPassed
6951     Else
6952         If AttByPawn = 0 Then
6953             Passed = True: GoTo lblEndBPassed
6954         ElseIf Phalanx >= LeverPush Then
6955             If Supported >= Lever Then Passed = True: GoTo lblEndBPassed
6956         End If
6957     End If
6958
6959     If Not Passed And Supported And RankNum <= 4 Then ' sacrifice supporter pawn to create passer?
6960     If PawnsWMin(FileNum) = RankNum - 1 Then
6961         If PawnsWMin(FileNum - 1) > RankNum Then ' no other stopper left side (PawnsWMin=9 if no
            pawn)
6962             If CBool(BAttack(Square) And PRAttackBit) Then ' left side supporter pawn
6963                 If Board(Square + SQ_LEFT) >= NO_PIECE Then ' can move forward to attack stopper
6964                     If Not CBool(WAttack(Square + SQ_LEFT) And PRAttackBit) Then ' no second left
                        to right attacker from file-2
6965                     Passed = True: GoTo lblEndBPassed
6966                 End If
6967             End If
6968         End If
6969     End If
6970     If Not Passed Then
6971         If PawnsWMin(FileNum + 1) > RankNum Then
6972             If CBool(BAttack(Square) And PLAttackBit) Then ' right side supporter pawn
6973                 If Board(Square + SQ_RIGHT) >= NO_PIECE Then ' can move forward to attack
                    stopper
6974                 If Not CBool(WAttack(Square + SQ_RIGHT) And PLAttackBit) Then ' no
                    second right to left attacker
6975                 Passed = True: GoTo lblEndBPassed
6976             End If
6977         End If
6978     End If
6979     End If
6980     End If
6981     End If
6982     End If
6983 lblEndBPassed:
6984
6985     '--- pawn score
6986     If Lever Then AddScore SC, LeverBonus(RelRank)
6987     If Supported Or Phalanx Then ' Connected

```

```

6988     AddScore SC, ConnectedBonus(Abs(Opposed), Abs(Phalanx <> 0), DefByPawn, RelRank)
6989 ElseIf Not Neighbours Then
6990     MinusScore SC, IsolatedPenalty(Abs(Opposed))
6991     If Not Opposed Then BWeakUnopposedCnt = BWeakUnopposedCnt + 1
6992 ElseIf Backward Then
6993     MinusScore SC, BackwardPenalty(Abs(Opposed))
6994 End If
6995 If Doubled And Supported = 0 Then MinusScore SC, DoubledPenalty
6996 '-----
6997 If bEndgame Then
6998     If FileNum = 1 Or FileNum = 8 Then AddScore SC, PsqtBP(Square)
6999     If BPawnCnt = 1 Then SC.EG = SC.EG + 2 * RelRank * RelRank
7000     If SC.EG + PsqtBP(Square).EG > BBestPawnVal Then
7001         BBestPawnVal = SC.EG + PsqtBP(Square).EG: BBestPawn = Square
7002     ElseIf SC.EG = BBestPawnVal Then
7003         If BBestPawn = 0 Or MaxDistance(Square, BKingLoc) < MaxDistance(BBestPawn,
7004             BKingLoc) Then
7005             BBestPawnVal = SC.EG: BBestPawn = Square
7006         End If
7007     End If
7008 End If
7009 ' Passed : eval later when full attack is available
7010 If Passed And Not Doubled Then
7011     PassedPawnsCnt = PassedPawnsCnt + 1: PassedPawns(PassedPawnsCnt) = Square
7012     If RelRank > 4 Then If Abs(FileNum - WKingFile) <= 2 Then BPassedPawnAttack =
7013         BPassedPawnAttack + 1
7014 End If
7015 '
7016 AddScore BPawnStruct, SC
7017 If bEvalTrace Then WriteTrace "BPawn: " & LocCoord(Square) & ">" & SC.MG & ", " &
7018     SC.EG
7019 Next a
7020 '-----
7021 '--- WHITE KNIGHTS -----
7022 '-----
7023 For a = 1 To PieceSqlListCnt(WKNIGHT)
7024     Square = PieceSqlList(WKNIGHT, a): FileNum = File(Square): RankNum = Rank(Square):
7025     RelRank = RankNum: SC.MG = 0: SC.EG = 0
7026     WPos.MG = WPos.MG + PsqtWN(Square).MG: WPos.EG = WPos.EG + PsqtWN(Square).EG: r =
7027     0
7028     ' Outpost bonus
7029     If WOutpostSq(Square) Then
7030         If Not CBool(BAttack(Square) And PAttackBit) Then ' not attacked by pawn
7031             ' Defended by pawn?
7032             AddScore SC, OutpostBonusKnight(Abs(CBool(WAttack(Square) And PAttackBit))): r
7033             = 3 ' ignore ReachableOutpost
7034             If bEvalTrace Then WriteTrace "WKight: " & LocCoord(Square) & "> Outpost:" &
7035                 OutpostBonusKnight(Abs(CBool(WAttack(Square) And PAttackBit))).MG
7036         End If
7037     End If
7038 End If
7039 '--- Mobility
7040 If Moved(Square) = 0 Then If RankNum = 1 Then SC.MG = SC.MG - 45 ' develop knight
7041 ForkCnt = 0: MobCnt = 0
7042 If a = 1 Then PieceAttackBit = N1AttackBit Else PieceAttackBit = N2AttackBit
7043 For i = 0 To 7
7044     Offset = KnightOffsets(i): Target = Square + Offset
7045     If Board(Target) <> FRAME Then
7046         WAttack(Target) = WAttack(Target) Or PieceAttackBit
7047     End If
7048     Select Case Board(Target)
7049     Case NO_PIECE:
7050         If (Not CBool(BAttack(Target) And PAttackBit)) Then MobCnt = MobCnt + 1:
7051             SC.MG = SC.MG + 3
7052     Case WPAWN: SC.MG = SC.MG + 2: SC.EG = SC.EG + 2
7053         If RankNum > 3 Then If Board(Target + SQ_UP) >= NO_PIECE Then If Not CBool
7054             (BAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1

```

```

7047     Case BPAWN: SC.MG = SC.MG + 7: SC.EG = SC.EG + 7: If Rank(Target) >= 6 Then
SC.MG = SC.MG + 4
7048     If (Not CBool(BAttack(Target) And PAttackBit)) Then MobCnt = MobCnt + 1:
AddThreat COL_BLACK, PT_PAWN, PT_KNIGHT, Square, Target
7049     Case BKNIGHT, BBISHOP: If (Not CBool(BAttack(Target) And PAttackBit)) Then
MobCnt = MobCnt + 1
7050     AddThreat COL_BLACK, PieceType(Board(Target)), PT_KNIGHT, Square, Target
'-- no Score for WKnight: total is zero
7051     Case BROOK, BQUEEN: If (Not CBool(BAttack(Target) And PAttackBit)) Then
MobCnt = MobCnt + 1
7052     AddThreat COL_BLACK, PieceType(Board(Target)), PT_KNIGHT, Square, Target:
ForkCnt = ForkCnt + 1
7053     Case WKING, WQUEEN: 'ignore
7054     Case BKING: MobCnt = MobCnt + 1: ForkCnt = ForkCnt + 1
7055     Case WEP_PIECE, BEP_PIECE:
7056     If (Not CBool(BAttack(Target) And PAttackBit)) Then MobCnt = MobCnt + 1:
SC.MG = SC.MG + 3
7057     Case Else: If (Not CBool(BAttack(Target) And PAttackBit)) Then MobCnt =
MobCnt + 1
7058     End Select
7059
7060     If r < 2 Then 'choose best square only
7061     If WOutpostSq(Target) Then 'Empty or opp piece: square can be occupied.
7062     'not attacked by opp pawn? Else if not blocked by own piece
7063     If Not CBool(BAttack(Target) And PAttackBit) Then
7064     r = 2: rr = 1 + Abs(CBool(WAttack(Target) And PAttackBit)) 'supported by
own pawn? Factor 2
7065     Else
7066     If r = 0 Then If PieceColor(Board(Target)) <> COL_WHITE Then r = 1: rr =
1 + Abs(CBool(WAttack(Target) And PAttackBit)) 'supported by own pawn?
Factor 2
7067     End If
7068     End If
7069     End If
7070     End If
7071     Next
7072
7073     If ForkCnt > 1 Then AddScoreVal SC, 7 * ForkCnt * ForkCnt, 5 * ForkCnt * ForkCnt:
If bWhiteToMove Then AddScoreVal SC, 35, 35
7074     AddScore WMobility, MobilityN(MobCnt)
7075     'Minor behind pawn bonus
7076     If RelRank < 5 Then
7077     If PieceType(Board(Square + SQ_UP)) = PT_PAWN Then SC.MG = SC.MG + 16: If
bEvalTrace Then WriteTrace "WKnight: " & LocCoord(Square) & "> Behind pawn 16"
7078     End If
7079     If r > 0 And r < 3 Then AddScoreWithFactor SC, ReachableOutpostKnight(r - 1), rr
7080     If CBool(BAttack(Square) And PAttackBit) Then AddPawnThreat BThreat, COL_WHITE,
PieceType(Board(Square)), Square
7081     AddScoreWithFactor SC, KingProtector(PT_KNIGHT), MaxDistance(Square, WKingLoc) '
defends king?
7082     AddScore WPos, SC
7083     If bEvalTrace Then WriteTrace "WKnight: " & LocCoord(Square) & ">" & SC.MG & ", "
& SC.EG & " / " & WPos.MG & ", " & WPos.EG
7084     Next a
7085
7086     '-----
7087     '---- BLACK KNIGHTs -----
7088     '-----
7089     For a = 1 To PieceSqlListCnt(BKNIGHT)
7090     Square = PieceSqlList(BKNIGHT, a): FileNum = File(Square): RankNum = Rank(Square):
RelRank = (9 - RankNum): SC.MG = 0: SC.EG = 0
7091     BPos.MG = BPos.MG + PsqtBN(Square).MG: BPos.EG = BPos.EG + PsqtBN(Square).EG: r =
0
7092     'Outpost bonus
7093     If BOutpostSq(Square) Then
7094     If Not CBool(WAttack(Square) And PAttackBit) Then 'not attacked by pawn
7095     'Defended by pawn?
7096     AddScore SC, OutpostBonusKnight(Abs(CBool(BAttack(Square) And PAttackBit))): r

```

```

        = 3 'ignore ReachableOutpost
7097 If bEvalTrace Then WriteTrace "BKnight: " & LocCoord(Square) & "> Outpost:" &
    OutpostBonusKnight(Abs(CBool(BAttack(Square) And PAttackBit))).MG
7098 End If
7099 End If
7100 If Moved(Square) = 0 Then If RankNum = 8 Then SC.MG = SC.MG - 45
7101 '--- Mobility
7102 ForkCnt = 0: MobCnt = 0
7103 If a = 1 Then PieceAttackBit = N1AttackBit Else PieceAttackBit = N2AttackBit
7104
7105 For i = 0 To 7
7106     Offset = KnightOffsets(i)
7107     Target = Square + Offset
7108     If Board(Target) <> FRAME Then
7109         BAttack(Target) = BAttack(Target) Or PieceAttackBit
7110         Select Case Board(Target)
7111             Case NO_PIECE:
7112                 If (Not CBool(WAttack(Target) And PAttackBit)) Then MobCnt = MobCnt + 1:
                    SC.MG = SC.MG + 3
7113             Case BPAWN: SC.MG = SC.MG + 2: SC.EG = SC.EG + 2
7114                 If RankNum < 6 Then If Board(Target + SQ_DOWN) >= NO_PIECE Then If Not
                    CBool(WAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1
7115             Case WPAWN: SC.MG = SC.MG + 7: SC.EG = SC.EG + 7: If Rank(Target) <= 3 Then
                    SC.MG = SC.MG + 4
7116                 If (Not CBool(WAttack(Target) And PAttackBit)) Then MobCnt = MobCnt + 1
7117                 AddThreat COL_WHITE, PT_PAWN, PT_KNIGHT, Square, Target
7118             Case WKNIGHT, WBISHOP: If (Not CBool(WAttack(Target) And PAttackBit)) Then
                    MobCnt = MobCnt + 1
7119                 AddThreat COL_WHITE, PieceType(Board(Target)), PT_KNIGHT, Square, Target
7120                 '-- no Score for WKnight: total is zero
7121             Case WROOK, WQUEEN: If (Not CBool(WAttack(Target) And PAttackBit)) Then
                    MobCnt = MobCnt + 1
7122                 AddThreat COL_WHITE, PieceType(Board(Target)), PT_KNIGHT, Square, Target:
                    ForkCnt = ForkCnt + 1
7123             Case BKING, BQUEEN: 'Ignore
7124             Case WKING: If (Not CBool(WAttack(Target) And PAttackBit)) Then MobCnt =
                    MobCnt + 1
7125                 If (Not CBool(WAttack(Target) And PAttackBit)) Then ForkCnt = ForkCnt + 1
7126             Case WEP_PIECE, BEP_PIECE:
                    If (Not CBool(WAttack(Target) And PAttackBit)) Then MobCnt = MobCnt + 1:
                        SC.MG = SC.MG + 3
7127             Case Else: If (Not CBool(WAttack(Target) And PAttackBit)) Then MobCnt =
                    MobCnt + 1
7128         End Select
7129
7130     If r < 2 Then
7131         If BOutpostSq(Target) Then ' Empty or opp piece: square can be occupied
7132             ' not attacked by opp pawn? Else if not blocked by own piece
7133             If Not CBool(WAttack(Target) And PAttackBit) Then
7134                 r = 2: rr = 1 + Abs(CBool(BAttack(Target) And PAttackBit)) 'supported by
                    own pawn? Factor 2
7135             Else
7136                 If r = 0 Then If PieceColor(Board(Target)) <> COL_BLACK Then r = 1: rr =
                    1 + Abs(CBool(BAttack(Target) And PAttackBit)) 'supported by own pawn?
                    Factor 2
7137             End If
7138         End If
7139     End If
7140 End If
7141 Next
7142
7143 If ForkCnt > 1 Then AddScoreVal SC, 7 * ForkCnt * ForkCnt, 5 * ForkCnt * ForkCnt:
    If Not bWhiteToMove Then AddScoreVal SC, 35, 35
7144 AddScore BMobility, MobilityN(MobCnt)
7145 ' Minor behind pawn bonus
7146 If RelRank < 5 Then
7147     If PieceType(Board(Square + SQ_DOWN)) = PT_PAWN Then SC.MG = SC.MG + 16: If
        bEvalTrace Then WriteTrace "BKnight: " & LocCoord(Square) & "> Behind pawn 16"

```



```

7148 End If
7149 If r > 0 And r < 3 Then AddScoreWithFactor SC, ReachableOutpostKnight(r - 1), rr
7150 If CBool(WAttack(Square) And PAttackBit) Then AddPawnThreat WThreat, COL_BLACK,
PieceType(Board(Square)), Square
7151 AddScoreWithFactor SC, KingProtector(PT_KNIGHT), MaxDistance(Square, BKingLoc) '
defends king?
7152 AddScore BPos, SC
7153 If bEvalTrace Then WriteTrace "BKnight: " & LocCoord(Square) & ">" & SC.MG & ", "
& SC.EG & " / " & BPos.MG & ", " & BPos.EG
7154 Next a
7155
7156 '-----
7157 '--- WHITE BISHOPs ---
7158 '-----
7159 For a = 1 To PieceSqListCnt(WBISHOP)
7160 Square = PieceSqList(WBISHOP, a): FileNum = File(Square): RankNum = Rank(Square):
RelRank = RankNum: SC.MG = 0: SC.EG = 0
7161 If ColorSq(Square) = COL_WHITE Then WBishopsOnWhiteSq = WBishopsOnWhiteSq + 1 Else
WBishopsOnBlackSq = WBishopsOnBlackSq + 1
7162 WPos.MG = WPos.MG + PsqtWB(Square).MG: WPos.EG = WPos.EG + PsqtWB(Square).EG: r =
0
7163 ' Outpost bonus
7164 If WOutpostSq(Square) Then
7165 If Not CBool(BAttack(Square) And PAttackBit) Then ' not attacked by pawn
7166 ' Defended by pawn?
7167 AddScore SC, OutpostBonusBishop(Abs(CBool(WAttack(Square) And PAttackBit))): r
= 3 ' ignore ReachableOutpost
7168 If bEvalTrace Then WriteTrace "WBishop: " & LocCoord(Square) & "> Outpost:" &
OutpostBonusBishop(Abs(CBool(WAttack(Square) And PAttackBit))).MG
7169 End If
7170 End If
7171 '--- Mobility
7172 MobCnt = 0
7173 If a = 1 Then PieceAttackBit = B1AttackBit Else PieceAttackBit = B2AttackBit
7174
7175 For i = 4 To 7
7176 Offset = DirectionOffset(i): Target = Square + Offset: AttackBit =
PieceAttackBit
7177
7178 Do While Board(Target) <> FRAME
7179 WAttack(Target) = WAttack(Target) Or AttackBit
7180
7181 Select Case Board(Target)
7182 Case NO_PIECE:
7183 If Not CBool(BAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1: If
Offset > 0 Then SC.MG = SC.MG + 2
7184 Case WPAWN: SC.MG = SC.MG + 2: SC.EG = SC.EG + 3:
7185 If RankNum > 3 Then If Board(Target + SQ_UP) >= NO_PIECE Then If Not CBool
(BAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1
7186 If Offset > 0 Then WAttack(Target + Offset) = WAttack(Target + Offset) Or
BXrayAttackBit
7187 Exit Do
7188 Case BPAWN: If Not CBool(BAttack(Target) And PAttackBit) Then MobCnt =
MobCnt + 1
7189 If AttackBit = PieceAttackBit Then AddThreat COL_BLACK, PT_PAWN, PT_BISHOP
, Square, Target: SC.MG = SC.MG + 7: SC.EG = SC.EG + 7
7190 Exit Do
7191 Case BKNIGHT, BBISHOP, BROOK, BQUEEN: If Not CBool(BAttack(Target) And
PAttackBit) Then MobCnt = MobCnt + 1
7192 If AttackBit = PieceAttackBit Then AddThreat COL_BLACK, PieceType(Board(
Target)), PT_BISHOP, Square, Target ' Reattack: no SC because x-x=0
7193 Exit Do
7194 Case WKING: Exit Do 'ignore
7195 Case BKing: MobCnt = MobCnt + 1
7196 Exit Do
7197 Case WQUEEN: AttackBit = BXrayAttackBit '--- Continue xray
7198 Case WEP_PIECE, BEP_PIECE:
7199 If Not CBool(BAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1: If

```



```

        Offset > 0 Then SC.MG = SC.MG + 2
7200     Case Else: If Not CBool(BAttack(Target) And PAttackBit) Then MobCnt = MobCnt
        + 1
7201     Exit Do ' own bishop or knight
7202 End Select
7203
7204     If r < 2 Then
7205         If WOutpostSq(Target) Then ' Empty or opp piece: square can be occupied
7206             ' not attacked by opp pawn? Else if not blocked by own piece
7207             If Not CBool(BAttack(Target) And PAttackBit) Then
7208                 r = 2: rr = 1 + Abs(CBool(WAttack(Target) And PAttackBit)) ' supported by
                    own pawn? Factor 2
7209             Else
7210                 If r = 0 Then If PieceColor(Board(Target)) <> COL_WHITE Then r = 1: rr =
                    1 + Abs(CBool(WAttack(Target) And PAttackBit)) ' supported by own pawn?
                    Factor 2
7211                 End If
7212             End If
7213         End If
7214         Target = Target + Offset
7215     Loop
7216
7217 Next
7218
7219 AddScore WMobility, MobilityB(MobCnt)
7220 If bEvalTrace Then WriteTrace "WBishop: " & LocCoord(Square) & ">" & SC.MG & ", "
    & SC.EG & " / " & WPos.MG & ", " & WPos.EG
7221 ' Minor behind pawn bonus
7222 If RelRank < 5 Then
7223     If PieceType(Board(Square + SQ_UP)) = PT_PAWN Then SC.MG = SC.MG + 16: If
        bEvalTrace Then WriteTrace "WBishop: " & LocCoord(Square) & "> Behind pawn 16"
7224 End If
7225 If r > 0 And r < 3 Then AddScoreWithFactor SC, ReachableOutpostBishop(r - 1), rr
7226 If CBool(BAttack(Square) And PAttackBit) Then AddPawnThreat BThreat, COL_WHITE,
    PieceType(Board(Square)), Square
7227 AddScoreWithFactor SC, KingProtector(PT_BISHOP), MaxDistance(Square, WKingLoc) '
    defends king?
7228 AddScore WPos, SC
7229 Next a
7230
7231 '-----
7232 '--- BLACK BISHOPS -----
7233 '----- '
7234 For a = 1 To PieceSqlListCnt(BBISHOP)
7235     Square = PieceSqlList(BBISHOP, a): FileNum = File(Square): RankNum = Rank(Square):
        RelRank = (9 - RankNum): SC.MG = 0: SC.EG = 0
7236     If ColorSq(Square) = COL_WHITE Then BBishopsOnWhiteSq = BBishopsOnWhiteSq + 1 Else
        BBishopsOnBlackSq = BBishopsOnBlackSq + 1
7237     BPos.MG = BPos.MG + PsqtBB(Square).MG: BPos.EG = BPos.EG + PsqtBB(Square).EG: r =
        0
7238     ' Outpost bonus
7239     If BOutpostSq(Square) Then
7240         If Not CBool(WAttack(Square) And PAttackBit) Then ' not attacked by pawn
7241             ' Defended by pawn?
7242             AddScore SC, OutpostBonusBishop(Abs(CBool(BAttack(Square) And PAttackBit))): r
                = 3 ' ignore ReachableOutpost
7243             If bEvalTrace Then WriteTrace "BBishop: " & LocCoord(Square) & "> Outpost:" &
                OutpostBonusBishop(Abs(CBool(BAttack(Square) And PAttackBit))).MG
7244             End If
7245         End If
7246     End If
7247     '--- Mobility
7248     MobCnt = 0
7249     If a = 1 Then PieceAttackBit = B1AttackBit Else PieceAttackBit = B2AttackBit
7250
7251     For i = 4 To 7
7252         Offset = DirectionOffset(i): Target = Square + Offset: AttackBit =
            PieceAttackBit

```

```

7253 Do While Board(Target) <> FRAME
7254     BAttack(Target) = BAttack(Target) Or AttackBit
7255
7256 Select Case Board(Target)
7257     Case NO_PIECE:
7258         If Not CBool(WAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1: If
            Offset < 0 Then SC.MG = SC.MG + 2
7259     Case BPAWN: SC.MG = SC.MG + 2: SC.EG = SC.EG + 3
7260         If RankNum < 6 Then If Board(Target + SQ_DOWN) >= NO_PIECE Then If Not
            CBool(WAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1
7261         If Offset < 0 Then BAttack(Target + Offset) = BAttack(Target + Offset) Or
            BXrayAttackBit
7262         Exit Do
7263     Case WPAWN: If Not CBool(WAttack(Target) And PAttackBit) Then MobCnt =
            MobCnt + 1
7264         If AttackBit = PieceAttackBit Then AddThreat COL_WHITE, PT_PAWN, PT_BISHOP
            , Square, Target: SC.MG = SC.MG + 7: SC.EG = SC.EG + 7
7265         Exit Do
7266     Case WKNIIGHT, WBISHOP, WROOK, WQUEEN: If Not CBool(WAttack(Target) And
            PAttackBit) Then MobCnt = MobCnt + 1
7267         If AttackBit = PieceAttackBit Then AddThreat COL_WHITE, PieceType(Board(
            Target)), PT_BISHOP, Square, Target
7268         Exit Do
7269     Case BKING: Exit Do 'Ignore
7270     Case WKING: MobCnt = MobCnt + 1
7271     Exit Do
7272     Case BQUEEN: AttackBit = BXrayAttackBit '--- Continue xray
7273     Case WEP_PIECE, BEP_PIECE:
7274         If Not CBool(WAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1: If
            Offset < 0 Then SC.MG = SC.MG + 2
7275     Case Else: If Not CBool(WAttack(Target) And PAttackBit) Then MobCnt = MobCnt
            + 1
7276         Exit Do 'own bishop or knight
7277 End Select
7278
7279 If r < 2 Then
7280     If BOutpostSq(Target) Then ' Empty or opp piece: square can be occupied
7281         ' not attacked by opp pawn? Else if not blocked by own piece
7282         If Not CBool(WAttack(Target) And PAttackBit) Then
7283             r = 2: rr = 1 + Abs(CBool(BAttack(Target) And PAttackBit)) 'supported by
                own pawn? Factor 2
7284         Else
7285             If r = 0 Then If PieceColor(Board(Target)) <> COL_BLACK Then r = 1: rr =
                1 + Abs(CBool(BAttack(Target) And PAttackBit)) 'supported by own pawn?
                Factor 2
7286             End If
7287         End If
7288     End If
7289     Target = Target + Offset
7290 Loop
7291
7292 Next
7293
7294 AddScore BMobility, MobilityB(MobCnt)
7295 If bEvalTrace Then WriteTrace "BBishop: " & LocCoord(Square) & ">" & SC.MG & ", "
    & SC.EG & " / " & BPos.MG & ", " & BPos.EG
7296 ' Minor behind pawn bonus
7297 If RelRank < 5 Then
7298     If PieceType(Board(Square + SQ_DOWN)) = PT_PAWN Then SC.MG = SC.MG + 16: If
        bEvalTrace Then WriteTrace "BBishop: " & LocCoord(Square) & "> Behind pawn 16"
7299 End If
7300 If r > 0 And r < 3 Then AddScoreWithFactor SC, ReachableOutpostBishop(r - 1), rr
7301 If CBool(WAttack(Square) And PAttackBit) Then AddPawnThreat WThreat, COL_BLACK,
    PieceType(Board(Square)), Square
7302 AddScoreWithFactor SC, KingProtector(PT_BISHOP), MaxDistance(Square, BKingLoc) '
    defends king?
7303 AddScore BPos, SC
7304 Next a

```

```

7305 '-----
7306 '--- WHITE ROOKS -----
7307 '-----
7308
7309 For a = 1 To PieceSqlListCnt(WROOK)
7310     Square = PieceSqlList(WROOK, a): FileNum = File(Square): RankNum = Rank(Square):
7311     RelRank = RankNum: SC.MG = 0: SC.EG = 0
7312     WPos.MG = WPos.MG + PsqtWR(Square).MG: WPos.EG = WPos.EG + PsqtWR(Square).EG
7313     If WPawns(FileNum) = 0 Then
7314         If BPawns(FileNum) = 0 Then
7315             SC.MG = SC.MG + 45: SC.EG = SC.EG + 20
7316         Else
7317             SC.MG = SC.MG + 20: SC.EG = SC.EG + 7
7318         End If
7319     End If
7320     '--- Mobility
7321     MobCnt = 0
7322     If a = 1 Then PieceAttackBit = R1AttackBit Else PieceAttackBit = R2AttackBit
7323
7324     For i = 0 To 3
7325         Offset = DirectionOffset(i): Target = Square + Offset: AttackBit =
7326             PieceAttackBit
7327
7328         Do While Board(Target) <> FRAME
7329             WAttack(Target) = WAttack(Target) Or AttackBit
7330
7331             Select Case Board(Target)
7332             Case NO_PIECE:
7333                 If Not CBool(BAttack(Target) And PBNAttackBit) Then MobCnt = MobCnt + 1:
7334                 If Abs(Offset) = 10 Then SC.MG = SC.MG + 7
7335             Case WPAWN: SC.MG = SC.MG + 2: SC.EG = SC.EG + 5:
7336                 If RankNum > 3 Then If Board(Target + SQ_UP) >= NO_PIECE Then If Not CBool
7337                     (BAttack(Target) And PBNAttackBit) Then MobCnt = MobCnt + 1
7338                 Exit Do
7339             Case BPAWN:
7340                 SC.MG = SC.MG + 7: SC.EG = SC.EG + 10 '--- no reattack possible
7341                 If Not CBool(BAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1
7342                 If AttackBit = PieceAttackBit Then AddThreat COL_BLACK, PT_PAWN, PT_ROOK,
7343                     Square, Target
7344                 If RankNum >= 5 Then SC.MG = SC.MG + 8: SC.EG = SC.EG + 25 'aligned pawns
7345                 Exit Do
7346             Case BKNIGHT, BBISHOP:
7347                 If Not CBool(BAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1
7348                 If AttackBit = PieceAttackBit Then AddThreat COL_BLACK, PieceType(Board(
7349                     Target)), PT_ROOK, Square, Target '--- no reattack possible
7350                 Exit Do
7351             Case BROOK: If AttackBit = PieceAttackBit Then AddThreat COL_BLACK, PT_ROOK
7352                 , PT_ROOK, Square, Target
7353                 MobCnt = MobCnt + 1
7354                 Exit Do 'equal exchange, ok for mobility
7355             Case WKING: Exit Do 'ignore
7356             Case BKING: MobCnt = MobCnt + 1
7357                 Exit Do
7358             Case BQUEEN: MobCnt = MobCnt + 1: If AttackBit = PieceAttackBit Then
7359                 AddThreat COL_BLACK, PT_QUEEN, PT_ROOK, Square, Target
7360                 Exit Do
7361             Case WROOK, WQUEEN:
7362                 If Offset = 10 Then
7363                     If WPawns(FileNum) = 0 Then SC.MG = SC.MG + 12: If BPawns(FileNum) = 0
7364                         Then SC.MG = SC.MG + 15
7365                     End If
7366                 If Board(Target) = WROOK Then If Not CBool(BAttack(Target) And
7367                     PBNAttackBit) Then MobCnt = MobCnt + 1
7368                 If a = 1 Then AttackBit = R1XrayAttackBit Else AttackBit = R2XrayAttackBit
7369                 '--- double lines , continue xray
7370             Case WEP_PIECE, BEP_PIECE:
7371                 If Not CBool(BAttack(Target) And PBNAttackBit) Then MobCnt = MobCnt + 1:
7372                 If Abs(Offset) = 10 Then SC.MG = SC.MG + 7

```

```

7361         Case Else: If Not CBool(BAttack(Target) And PBNAttackBit) Then MobCnt =
              MobCnt + 1
7362         Exit Do ' own bishop or knight
7363     End Select
7364
7365     Target = Target + Offset
7366 Loop
7367
7368 Next
7369
7370 AddScore WMobility, MobilityR(MobCnt)
7371 ' Trapped rook by king : worse when cannot castle
7372 If Not bEndgame Then
7373     If MobCnt <= 3 Then
7374         If WPawns(FileNum) > 0 Then
7375             If RankNum = Rank(WKingLoc) Or Rank(WKingLoc) = 1 Then
7376                 r = 0
7377                 If WKingFile < FILE_E Then
7378                     If FileNum < WKingFile Then r = -1
7379                 Else
7380                     If FileNum > WKingFile Then r = 1
7381                 End If
7382                 If r <> 0 Then
7383
7384                     For k = WKingFile + r To FileNum - r Step r ' own blocking pawns on files between
                        king an rook
7385                         If WPawns(k) = 0 Then
7386                             r = 0: Exit For
7387                         ElseIf PawnsWMin(k) > RankNum + 2 Then
7388                             r = 0: Exit For
7389                         End If
7390                     Next
7391
7392                     If r <> 0 Then SC.MG = SC.MG - (92 - MobCnt * 22) * (1 + Abs(Rank(
                        WKingLoc) = 1 And (Moved(WKING_START) > 0 Or (Moved(Square) > 0 And
                        RankNum = 1))))
7393                 End If
7394             End If
7395         End If
7396     End If
7397 Else
7398     If WPawns(FileNum) > 0 And BPawns(FileNum) = 0 And PawnsWMin(FileNum) >= 5 Then
7399         SC.MG = SC.MG + (PawnsWMin(FileNum)): SC.EG = SC.EG + 5 * PawnsWMin(FileNum)
7400     End If
7401 End If
7402 If CBool(BAttack(Square) And PAttackBit) Then AddPawnThreat BThreat, COL_WHITE,
PieceType(Board(Square)), Square
7403 AddScoreWithFactor SC, KingProtector(PT_ROOK), MaxDistance(Square, WKingLoc) '
defends king?
7404 AddScore WPos, SC
7405 If bEvalTrace Then WriteTrace "Wrook: " & LocCoord(Square) & ">" & SC.MG & ", " &
SC.EG & " / " & WPos.MG & ", " & WPos.EG
7406 Next a
7407
7408 '-----
7409 '---- BLACK ROOKs -----
7410 '----- '
7411 For a = 1 To PieceSqlListCnt(BROOK)
7412     Square = PieceSqlList(BROOK, a): FileNum = File(Square): RankNum = Rank(Square):
RelRank = (9 - RankNum): SC.MG = 0: SC.EG = 0
7413     BPos.MG = BPos.MG + PsqtBR(Square).MG: BPos.EG = BPos.EG + PsqtBR(Square).EG
7414     If BPawns(FileNum) = 0 Then
7415         If WPawns(FileNum) = 0 Then
7416             SC.MG = SC.MG + 45: SC.EG = SC.EG + 20
7417         Else
7418             SC.MG = SC.MG + 20: SC.EG = SC.EG + 7
7419         End If
7420     End If

```

```

7421 '--- Mobility
7422 MobCnt = 0
7423 If a = 1 Then PieceAttackBit = R1AttackBit Else PieceAttackBit = R2AttackBit
7424
7425 For i = 0 To 3
7426     Offset = DirectionOffset(i): Target = Square + Offset: AttackBit =
7427         PieceAttackBit
7428
7429     Do While Board(Target) <> FRAME
7430         BAttack(Target) = BAttack(Target) Or AttackBit
7431
7432         Select Case Board(Target)
7433             Case NO_PIECE:
7434                 If Not CBool(WAttack(Target) And PBNAttackBit) Then MobCnt = MobCnt + 1:
7435                     If Abs(Offset) = 10 Then SC.MG = SC.MG + 7
7436             Case BPAWN: SC.MG = SC.MG + 2: SC.EG = SC.EG + 5
7437                 If RankNum < 6 Then If Board(Target + SQ_DOWN) >= NO_PIECE Then If Not
7438                     CBool(WAttack(Target) And PBNAttackBit) Then MobCnt = MobCnt + 1
7439                 Exit Do
7440             Case WPAWN:
7441                 SC.MG = SC.MG + 7: SC.EG = SC.EG + 10 '--- no reattack possible
7442                 If Not CBool(WAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1
7443                 If AttackBit = PieceAttackBit Then AddThreat COL_WHITE, PT_PAWN, PT_ROOK,
7444                     Square, Target
7445                 If RankNum <= 4 Then SC.MG = SC.MG + 8: SC.EG = SC.EG + 25 'aligned pawns
7446                 Exit Do
7447             Case WKNIIGHT, WBISHOP:
7448                 If Not CBool(WAttack(Target) And PAttackBit) Then MobCnt = MobCnt + 1
7449                 If AttackBit = PieceAttackBit Then AddThreat COL_WHITE, PieceType(Board(
7450                     Target)), PT_ROOK, Square, Target
7451                 Exit Do '--- no reattack possible
7452             Case WROOK: If AttackBit = PieceAttackBit Then AddThreat COL_WHITE, PT_ROOK
7453                 , PT_ROOK, Square, Target
7454                 MobCnt = MobCnt + 1
7455                 Exit Do 'equal exchange ok for mobility
7456             Case BKING: Exit Do 'Ignore
7457             Case WKING: MobCnt = MobCnt + 1
7458                 Exit Do
7459             Case WQUEEN: MobCnt = MobCnt + 1: If AttackBit = PieceAttackBit Then
7460                 AddThreat COL_WHITE, PT_QUEEN, PT_ROOK, Square, Target
7461                 Exit Do
7462             Case BROOK, BQUEEN:
7463                 If Offset = -10 Then
7464                     If BPawns(FileNum) = 0 Then SC.MG = SC.MG + 12: If WPawns(FileNum) = 0
7465                         Then SC.MG = SC.MG + 15
7466                 End If
7467                 If Board(Target) = BROOK Then If Not CBool(WAttack(Target) And
7468                     PBNAttackBit) Then MobCnt = MobCnt + 1
7469                 If a = 1 Then AttackBit = R1XrayAttackBit Else AttackBit = R2XrayAttackBit
7470                 '--- double lines , continue xray
7471             Case WEP_PIECE, BEP_PIECE:
7472                 If Not CBool(WAttack(Target) And PBNAttackBit) Then MobCnt = MobCnt + 1:
7473                     If Abs(Offset) = 10 Then SC.MG = SC.MG + 7
7474                 Case Else: If Not CBool(WAttack(Target) And PBNAttackBit) Then MobCnt =
7475                     MobCnt + 1
7476                 Exit Do 'own bishop or knight
7477         End Select
7478
7479         Target = Target + Offset
7480     Loop
7481
7482 Next
7483
7484 AddScore BMobility, MobilityR(MobCnt)
7485 ' Trapped rook by king : worse when cannot castle
7486 If Not bEndgame Then
7487     If MobCnt <= 3 Then
7488         If BPawns(FileNum) > 0 Then

```

```

7477     If RankNum = Rank(BKingLoc) Or Rank(BKingLoc) = 1 Then
7478         r = 0
7479     If BKingFile < FILE_E Then
7480         If FileNum < BKingFile Then r = -1
7481     Else
7482         If FileNum > BKingFile Then r = 1
7483     End If
7484     If r <> 0 Then
7485
7486         For k = BKingFile + r To FileNum - r Step r ' own blocking pawns on files between
king an rook
7487             If BPawns(k) = 0 Then
7488                 r = 0: Exit For
7489             ElseIf PawnsBMax(k) < RankNum - 2 Then
7490                 r = 0: Exit For
7491             End If
7492         Next
7493
7494         If r <> 0 Then SC.MG = SC.MG - (92 - MobCnt * 22) * (1 + Abs(Rank(
BKingLoc) = 8 And (Moved(BKING_START) > 0 Or (Moved(Square) > 0 And
RankNum = 8))))
7495     End If
7496 End If
7497 End If
7498 End If
7499 Else
7500     If BPawns(FileNum) > 0 And WPawns(FileNum) = 0 And PawnsBMax(FileNum) <= 4 Then
7501         SC.MG = SC.MG + (9 - PawnsBMin(FileNum)): SC.EG = SC.EG + 5 * (9 - PawnsBMin(
FileNum))
7502     End If
7503 End If
7504 If CBool(WAttack(Square) And PAttackBit) Then AddPawnThreat WThreat, COL_BLACK,
PieceType(Board(Square)), Square
7505 AddScoreWithFactor SC, KingProtector(PT_ROOK), MaxDistance(Square, BKingLoc) '
defends king?
7506 AddScore BPos, SC
7507 If bEvalTrace Then WriteTrace "BROOK: " & LocCoord(Square) & ">" & SC.MG & ", " &
SC.EG & " / " & BPos.MG & ", " & BPos.EG
7508 Next a
7509
7510 '-----
7511 '---- WHITE QUEENS ( last - full attack info needed for mobility ) -
7512 '-----
7513 For a = 1 To PieceSqlListCnt(WQUEEN)
7514     Square = PieceSqlList(WQUEEN, a): FileNum = File(Square): RankNum = Rank(Square):
RelRank = RankNum: SC.MG = 0: SC.EG = 0: QueenWeak = False
7515     WPos.MG = WPos.MG + PsqtWQ(Square).MG: WPos.EG = WPos.EG + PsqtWQ(Square).EG
7516     '--- Mobility
7517     MobCnt = 0
7518
7519     For i = 0 To 7
7520         Offset = DirectionOffset(i): Target = Square + Offset: AttackBit = QAttackBit
7521
7522         Do While Board(Target) <> FRAME
7523             WAttack(Target) = WAttack(Target) Or AttackBit
7524
7525             Select Case Board(Target)
7526             Case NO_PIECE: If Not CBool(BAttack(Target) And PNBRAAttackBit) Then MobCnt =
MobCnt + 1
7527             Case WPAWN: SC.MG = SC.MG + 2: SC.EG = SC.EG + 2
7528                 If RankNum > 3 Then If Board(Target + SQ_UP) >= NO_PIECE Then If Not CBool
(BAttack(Target) And PNBRAAttackBit) Then MobCnt = MobCnt + 1
7529                 If Offset = SQ_UP_LEFT Or Offset = SQ_UP_RIGHT Then WAttack(Target +
Offset) = WAttack(Target + Offset) Or QXrayAttackBit
7530                 If CBool(BAttack(Target) And RBAttackBit) Then CheckWQueenWeek Target,
Offset, i, QueenWeak ' pin oder discovered attack?
7531                 Exit Do 'Defends pawn
7532             Case BPAWN:

```

```

7533     If Not CBool(BAttack(Target) And PNBRAAttackBit) Then
7534         MobCnt = MobCnt + 1: If AttackBit = QAttackBit Then AddThreat COL_BLACK,
            PT_PAWN, PT_QUEEN, Square, Target
7535     Else
7536         If CBool(BAttack(Target) And RBAttackBit) Then CheckWQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7537     End If
7538     SC.MG = SC.MG + 7: SC.EG = SC.EG + 7
7539     Exit Do 'Attack pawn
7540 Case BKNIGHT:
7541     If Not CBool(BAttack(Target) And PNBRAAttackBit) Then
7542         MobCnt = MobCnt + 1: If AttackBit = QAttackBit Then AddThreat COL_BLACK,
            PT_KNIGHT, PT_QUEEN, Square, Target
7543     Else
7544         If CBool(BAttack(Target) And RBAttackBit) Then CheckWQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7545     End If
7546     If AttackBit = QAttackBit Then AddThreat COL_BLACK, PieceType(Board(Target
    )), PT_QUEEN, Square, Target
7547     Exit Do
7548 Case BBISHOP:
7549     If Not CBool(BAttack(Target) And PNBRAAttackBit) Then
7550         MobCnt = MobCnt + 1: If AttackBit = QAttackBit Then AddThreat COL_BLACK,
            PT_BISHOP, PT_QUEEN, Square, Target
7551     Else
7552         If CBool(BAttack(Target) And RBAttackBit) Then CheckWQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7553     End If
7554     If AttackBit = QAttackBit Then AddThreat COL_BLACK, PieceType(Board(Target
    )), PT_QUEEN, Square, Target
7555     If CBool(BAttack(Target) And RBAttackBit) Then CheckWQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7556     Exit Do
7557 Case BROOK:
7558     If Not CBool(BAttack(Target) And PNBRAAttackBit) Then
7559         MobCnt = MobCnt + 1: If AttackBit = QAttackBit Then AddThreat COL_BLACK,
            PT_ROOK, PT_QUEEN, Square, Target
7560     Else
7561         If CBool(BAttack(Target) And RBAttackBit) Then CheckWQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7562     End If
7563     If AttackBit = QAttackBit Then AddThreat COL_BLACK, PieceType(Board(Target
    )), PT_QUEEN, Square, Target
7564     If CBool(BAttack(Target) And RBAttackBit) Then CheckWQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7565     Exit Do
7566 Case WKING: Exit Do 'ignore
7567 Case BKING: MobCnt = MobCnt + 1
7568     Exit Do
7569 Case BQUEEN: If AttackBit = QAttackBit Then AddThreat COL_BLACK, PT_QUEEN,
    PT_QUEEN, Square, Target: MobCnt = MobCnt + 1
7570     Exit Do
7571 Case WBISHOP:
7572     If Not CBool(BAttack(Target) And PNBRAAttackBit) Then
7573         MobCnt = MobCnt + 1: SC.MG = SC.MG + 4: SC.EG = SC.EG + 2
7574     Else
7575         If CBool(BAttack(Target) And RBAttackBit) Then CheckWQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7576     End If
7577     If i > 3 Then AttackBit = QXrayAttackBit Else Exit Do
7578 Case WKNIGHT:
7579     If Not CBool(BAttack(Target) And PNBRAAttackBit) Then
7580         MobCnt = MobCnt + 1
7581     Else
7582         If CBool(BAttack(Target) And RBAttackBit) Then CheckWQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7583     End If
7584     Exit Do

```



```

7585     Case WROOK:
7586         If Not CBool(BAttack(Target) And PNBRAAttackBit) Then
7587             If Offset = 10 Then
7588                 If WPawns(FileNum) = 0 Then
7589                     SC.MG = SC.MG + 10: SC.EG = SC.EG + 5
7590                 ElseIf BPawns(FileNum) = 0 Then
7591                     SC.MG = SC.MG + 15: SC.EG = SC.EG + 5
7592                 End If
7593             End If
7594             MobCnt = MobCnt + 1 '--- double lines
7595         Else
7596             If CBool(BAttack(Target) And RBAAttackBit) Then CheckWQueenWeek Target,
              Offset, i, QueenWeak 'pin oder discovered attack?
7597         End If
7598         If i < 4 Then AttackBit = QXrayAttackBit Else Exit Do
7599     Case WEP_PIECE, BEP_PIECE: If Not CBool(BAttack(Target) And PNBRAAttackBit)
              Then MobCnt = MobCnt + 1
7600     Case Else:
7601         Exit Do
7602     End Select
7603
7604     Target = Target + Offset
7605 Loop
7606
7607 Next
7608
7609 AddScore WMobility, MobilityQ(MobCnt)
7610 If CBool(BAttack(Square) And PAttackBit) Then AddPawnThreat BThreat, COL_WHITE,
              PieceType(Board(Square)), Square
7611 AddScoreWithFactor SC, KingProtector(PT_QUEEN), MaxDistance(Square, WKingLoc) '
              defends king?
7612 If QueenWeak Then SC.MG = SC.MG - 50: SC.EG = SC.EG - 10
7613 AddScore WPos, SC
7614 If bEvalTrace Then WriteTrace "WQueen: " & LocCoord(Square) & ">" & SC.MG & ", " &
              SC.EG & " / " & WPos.MG & ", " & WPos.EG
7615 Next a
7616
7617 '-----
7618 '---- BLACK QUEENS ( last - full attack info needed for mobility ) --
7619 '-----
7620 For a = 1 To PieceSqlListCnt(BQUEEN)
7621     Square = PieceSqlList(BQUEEN, a): FileNum = File(Square): RankNum = Rank(Square):
              RelRank = (9 - RankNum): SC.MG = 0: SC.EG = 0: QueenWeak = False
7622     BPos.MG = BPos.MG + PsqtBQ(Square).MG: BPos.EG = BPos.EG + PsqtBQ(Square).EG
7623     '--- Mobility
7624     MobCnt = 0
7625
7626     For i = 0 To 7
7627         Offset = DirectionOffset(i): Target = Square + Offset: AttackBit = QAttackBit
7628
7629         Do While Board(Target) <> FRAME
7630             BAttack(Target) = BAttack(Target) Or AttackBit
7631
7632             Select Case Board(Target)
7633             Case NO_PIECE: If Not CBool(WAttack(Target) And PNBRAAttackBit) Then MobCnt =
              MobCnt + 1
7634             Case BPAWN: SC.MG = SC.MG + 2: SC.EG = SC.EG + 2
7635                 If RankNum < 6 Then If Board(Target + SQ_DOWN) >= NO_PIECE Then If Not
              CBool(WAttack(Target) And PNBRAAttackBit) Then MobCnt = MobCnt + 1
7636                 If Offset = SQ_DOWN_LEFT Or Offset = SQ_DOWN_RIGHT Then BAttack(Target +
              Offset) = BAttack(Target + Offset) Or QXrayAttackBit
7637                 If CBool(WAttack(Target) And RBAAttackBit) Then CheckBQueenWeek Target,
              Offset, i, QueenWeak 'pin oder discovered attack?
7638                 Exit Do 'Defends pawn
7639             Case WPAWN:
7640                 If Not CBool(WAttack(Target) And PNBRAAttackBit) Then
7641                     MobCnt = MobCnt + 1: If AttackBit = QAttackBit Then AddThreat COL_WHITE,
              PT_PAWN, PT_QUEEN, Square, Target

```

```

7642 Else
7643     If CBool(WAttack(Target) And RAttackBit) Then CheckBQueenWeek Target,
        Offset, i, QueenWeak ' pin oder discovered attack?
7644 End If
7645 SC.MG = SC.MG + 7: SC.EG = SC.EG + 7
7646 Exit Do 'Attack pawn
7647 Case WKNIGHT:
7648     If Not CBool(WAttack(Target) And PNBAttackBit) Then
7649         MobCnt = MobCnt + 1
7650     Else
7651         If CBool(WAttack(Target) And RAttackBit) Then CheckBQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7652         End If
7653         If AttackBit = QAttackBit Then AddThreat COL_WHITE, PieceType(Board(Target
            )), PT_QUEEN, Square, Target
7654         Exit Do
7655 Case WBISHOP:
7656     If Not CBool(WAttack(Target) And PNBAttackBit) Then
7657         MobCnt = MobCnt + 1
7658     Else
7659         If CBool(WAttack(Target) And RAttackBit) Then CheckBQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7660         End If
7661         If AttackBit = QAttackBit Then AddThreat COL_WHITE, PieceType(Board(Target
            )), PT_QUEEN, Square, Target
7662         Exit Do
7663 Case WROOK:
7664     If Not CBool(WAttack(Target) And PNBAttackBit) Then
7665         MobCnt = MobCnt + 1
7666     Else
7667         If CBool(WAttack(Target) And RAttackBit) Then CheckBQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7668         End If
7669         If AttackBit = QAttackBit Then AddThreat COL_WHITE, PieceType(Board(Target
            )), PT_QUEEN, Square, Target
7670         Exit Do
7671 Case BKING: Exit Do 'Ignore
7672 Case WKING: MobCnt = MobCnt + 1
7673 Exit Do
7674 Case WQUEEN: If AttackBit = QAttackBit Then AddThreat COL_WHITE, PT_QUEEN,
PT_QUEEN, Square, Target: MobCnt = MobCnt + 1
7675 Exit Do
7676 Case BBISHOP:
7677     If Not CBool(WAttack(Target) And PNBAttackBit) Then
7678         MobCnt = MobCnt + 1: SC.MG = SC.MG + 4: SC.EG = SC.EG + 2
7679     Else
7680         If CBool(WAttack(Target) And RAttackBit) Then CheckBQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7681         End If
7682         If i > 3 Then AttackBit = QXrayAttackBit Else Exit Do
7683 Case BKNIGHT:
7684     If Not CBool(WAttack(Target) And PNBAttackBit) Then
7685         MobCnt = MobCnt + 1
7686     Else
7687         If CBool(WAttack(Target) And RAttackBit) Then CheckBQueenWeek Target,
            Offset, i, QueenWeak ' pin oder discovered attack?
7688         End If
7689         Exit Do
7690 Case BROOK:
7691     If Not CBool(WAttack(Target) And PNBAttackBit) Then
7692         If Offset = -10 Then
7693             If BPawns(FileNum) = 0 Then
7694                 SC.MG = SC.MG + 10: SC.EG = SC.EG + 5
7695             ElseIf WPawns(FileNum) = 0 Then
7696                 SC.MG = SC.MG + 15: SC.EG = SC.EG + 5
7697             End If
7698         End If
7699         MobCnt = MobCnt + 1

```

```

7700         Else
7701             If CBool(WAttack(Target) And RAttackBit) Then CheckBQueenWeek Target,
                Offset, i, QueenWeak ' pin oder discovered attack?
7702         End If
7703         If i < 4 Then AttackBit = QXrayAttackBit Else Exit Do
7704         Case WEP_PIECE, BEP_PIECE: If Not CBool(WAttack(Target) And PNBRAAttackBit)
                Then MobCnt = MobCnt + 1
7705         Case Else:
7706             Exit Do
7707         End Select
7708         Target = Target + Offset
7709     Loop
7710 Next
7711
7712 AddScore BMobility, MobilityQ(MobCnt)
7713 If CBool(WAttack(Square) And PAttackBit) Then AddPawnThreat WThreat, COL_BLACK,
    PieceType(Board(Square)), Square
7714 AddScoreWithFactor SC, KingProtector(PT_QUEEN), MaxDistance(Square, BKingLoc) '
    defends king?
7715 If QueenWeak Then SC.MG = SC.MG - 50: SC.EG = SC.EG - 10
7716 AddScore BPos, SC
7717 If bEvalTrace Then WriteTrace "BQueen: " & LocCoord(Square) & ">" & SC.MG & ", " &
    SC.EG & " / " & BPos.MG & ", " & BPos.EG
7718 Next a
7719
7720 '-----
7721 '---- Step 3.: Pass for pawn push ( full attack info needed for mobility )
7722 '-----
7723 SC = ZeroScore
7724
7725 For a = 1 To PieceSqListCnt(WPAWN)
7726     Square = PieceSqList(WPAWN, a): RelRank = Rank(Square)
7727
7728     ' bonus if safe pawn push attacks an enemy piece
7729     For rr = 1 To 1 + Abs(RelRank = 2)
7730         Target = Square + SQ_UP * rr
7731         If Board(Target) >= NO_PIECE Then ' empty or ep-dummy piece
7732             SC.MG = SC.MG + 8: SC.EG = SC.EG + 8 ' pawn mobility
7733             ' Safe pawn push: push field not attacked by opp pawn AND defend by own piece or not attacked by opp
7734             If BAttack(Target) = 0 Or WAttack(Target) > 0 Then
7735                 If Not (rr = 2 And CBool(BAttack(Square + SQ_UP) And PAttackBit)) Then '
                    check EnPassant capture
7736
7737                 For i = 9 To 11 Step 2
7738                     r = Board(Target + i)
7739                     If PieceColor(r) = COL_BLACK And r <> BPAWN Then
7740                         If Not CBool(WAttack(Target + i) And PAttackBit) Then ' already attacked by
                            own pawn?
7741                             SC.MG = SC.MG + 38: SC.EG = SC.EG + 22 ' pawn threats non pawn enemy
7742                         End If
7743                     End If
7744                 Next i
7745
7746             End If
7747         End If
7748     Else
7749         Exit For
7750     End If
7751 Next
7752 Next a
7753
7754 If SC.MG > 0 Then AddScore WPos, SC
7755 SC = ZeroScore
7756
7757 For a = 1 To PieceSqListCnt(BPAWN)
7758     Square = PieceSqList(BPAWN, a): RelRank = (9 - Rank(Square))
7759
7760     ' bonus if safe pawn push attacks an enemy piece

```

```

7761 For rr = 1 To 1 + Abs(RelRank = 2)
7762     Target = Square + SQ_DOWN * rr
7763     If Board(Target) >= NO_PIECE Then
7764         SC.MG = SC.MG + 8: SC.EG = SC.EG + 8 ' pawn mobility
7765         ' Safe pawn push: push field not attacked by opp pawn AND defend by own piece and not attacked by opp
7766         If WAttack(Target) = 0 Or BAttack(Target) > 0 Then
7767             If Not (rr = 2 And CBool(WAttack(Square + SQ_DOWN) And PAttackBit)) Then '
7768                 check EnPassant capture
7769
7770                 For i = 9 To 11 Step 2
7771                     r = Board(Target - i)
7772                     If PieceColor(r) = COL_WHITE And r <> WPAWN Then
7773                         If Not CBool(BAttack(Target - i) And PAttackBit) Then ' already attacked by
7774                             own pawn?
7775                             SC.MG = SC.MG + 38: SC.EG = SC.EG + 22 ' pawn threats non pawn enemy
7776                             End If
7777                         End If
7778                     Next i
7779                 End If
7780             Else
7781                 Exit For
7782             End If
7783         Next rr
7784     Next a
7785
7786     If SC.MG > 0 Then AddScore BPos, SC
7787     '--- End pass for pawn push <<<<
7788
7789
7790     '-----
7791     '--- Step 4: King Safety -----
7792     '-----
7793     If bEndgame Then
7794         WKSafety = ZeroScore: BKSafety = ZeroScore
7795     Else
7796         Dim Bonus As Long
7797         Dim KingOnlyDefended As Long, bSafe As Boolean, Tropism As Long
7798         '-----
7799         '--- White King Safety Eval -----
7800         '-----
7801         RankNum = Rank(WKingLoc): FileNum = WKingFile: Bonus = 0
7802         If (PieceCnt(BQUEEN) * 2 + PieceCnt(BROOK)) > 1 Then
7803             KingDanger = 0
7804             If WPawnCnt = 0 Then MinWKingPawnDistance = 0 Else MinWKingPawnDistance =
7805                 MinWKingPawnDistance - 1
7806             If RankNum > 4 Then
7807                 WKSafety.EG = WKSafety.EG - 16 * MinWKingPawnDistance
7808             Else
7809                 Bonus = WKingShelterStorm(WKingLoc)
7810                 If WhiteCastled = NO_CASTLE Then
7811                     If WKingLoc = SQ_E1 Then
7812                         If WPawns(7) > 0 And PawnsWMin(7) < 4 Then
7813                             If WCanCastleOO() Then
7814                                 Bonus = GetMax(Bonus, WKingShelterStorm(SQ_G1))
7815                             End If
7816                         End If
7817                     If (WPawns(3) > 0 And PawnsWMin(3) < 4) Or (WPawns(2) > 0 And PawnsWMin(2)
7818                         < 4) Then
7819                         If WCanCastleOOO() Then
7820                             Bonus = GetMax(Bonus, WKingShelterStorm(SQ_C1))
7821                         End If
7822                     End If
7823                 End If
7824                 AddScoreVal WKSafety, Bonus, -16 * MinWKingPawnDistance
7825             End If

```

```

7825 If bDoWKSafety Then
7826
7827 ' King tropism: firstly, find squares that opponent attacks in our king flank
7828 ' Secondly, add the squares which are attacked twice in that flank
7829 GetKingFlankFiles WKingLoc, r, rr: Tropism = 0
7830 For k = SQ_A1 - 1 To SQ_A1 - 1 + 40 Step 10 ' start square - 1 of rank 1-5 (camp)
7831     For Square = k + r To k + rr ' files king flank
7832         If BAttack(Square) <> 0 Then
7833             Tropism = Tropism + 1: If AttackBitCnt(BAttack(Square)) > 1 Then
7834                 Tropism = Tropism + 1 ' Attacked twice?
7835         End If
7836     Next
7837 Next
7838
7839 ' Pawnless king flank penalty
7840 k = 0
7841 For i = r To rr
7842     If WPawns(i) + BPawns(i) > 0 Then k = 1: Exit For
7843 Next
7844 If k = 0 Then MinusScore WKSafety, PawnlessFlank
7845
7846 '--- Check threats at king ring
7847 Undefended = 0: KingOnlyDefended = 0: WKingAttPieces = 0: KingLevers = 0
7848 ' add the 2 or 3 squares in front of king ring: king G1 => F3+G3+H3
7849 If RankNum = 1 Then
7850     For Target = WKingLoc + 19 To WKingLoc + 21
7851         If Board(Target) <> FRAME Then
7852             If BAttack(Target) <> 0 Then
7853                 If WAttack(Target) = 0 Or WAttack(Target) = QAttackBit Then
7854                     Undefended = Undefended + 1
7855                     ' exclude double pawn defended squares
7856                     If AttackBitCnt(WAttack(Target) And PAttackBit) < 2 Then
7857                         WKingAttPieces = WKingAttPieces Or BAttack(Target)
7858                         If Board(Target) = WPAWN Then
7859                             If CBool(BAttack(Target) And PAttackBit) Then KingLevers =
7860                                 KingLevers + 1
7861                         End If
7862                     End If
7863                 End If
7864             End If
7865         End If
7866     Next
7867 End If
7868
7869 For i = 0 To 7 ' for all directions from king square
7870     Offset = DirectionOffset(i): Target = WKingLoc + Offset
7871     If Board(Target) <> FRAME Then
7872         If BAttack(Target) <> 0 Then
7873             ' King attacks are added later in attack array, so distance=1 and WAttack=0 is equal to king
7874             attack only
7875             If WAttack(Target) = 0 Then KingOnlyDefended = KingOnlyDefended + 1
7876             WKingAdjacentZoneAttCnt = WKingAdjacentZoneAttCnt + AttackBitCnt(
7877                 BAttack(Target) And Not PAttackBit)
7878             ' exclude double pawn defended squares
7879             If AttackBitCnt(WAttack(Target) And PAttackBit) < 2 Then
7880                 WKingAttPieces = WKingAttPieces Or BAttack(Target)
7881                 If Board(Target) = WPAWN Then
7882                     If CBool(BAttack(Target) And PAttackBit) Then KingLevers =
7883                         KingLevers + 1
7884                     End If
7885                 End If
7886             End If
7887             WKDefender = WKDefender Or WAttack(Target)
7888             rr = 1 ' rr=Distance to King
7889
7890             Do ' loop for a direction
7891                 r = BAttack(Target)
7892                 If CBool(r And QRBAttackBit) Then
7893                     bSafe = False ' Safe attack square?
7894                     If PieceColor(Board(Target)) <> BCOL Then

```

```

7885     If WAttack(Target) = 0 Then
7886         If rr = 1 Then
7887             If AttackBitCnt(BAttack(Target)) > 1 Then bSafe = True
7888         Else
7889             bSafe = True
7890         End If
7891     End If
7892 End If
7893 ' Queen safe checks
7894 If bSafe Then
7895     If CBool(r And QAttackBit) Then
7896         If Not CBool(WChecksCounted And QAttackBit) Then
7897             KingDanger = KingDanger + QueenCheck
7898             WChecksCounted = (WChecksCounted Or QAttackBit)
7899         End If
7900     End If
7901 End If
7902 If CBool(r And RBoRxrayAttackBit) Then
7903     If Not bSafe And rr > 1 Then ' not defended by king
7904         ' For minors and rooks, also consider the square as safe if attacked twice,
7905         ' and only defended by our queen.
7906         If CBool(WAttack(Target) = QAttackBit) Then
7907             If AttackBitCnt(BAttack(Target)) > 1 Then
7908                 If Not (AttackBitCnt(WAttack(Target)) > 1 Or PieceColor(
7909                     Board(Target)) = BCOL) Then
7910                     bSafe = True
7911                 End If
7912             End If
7913         End If
7914     End If
7915     ' (i=0-3: orthogonal offset, 4-7:diagonal)
7916     ' Rook checks
7917     If i < 4 Then
7918         If CBool(r And RORxrayAttackBit) Then ' R1Attackbit or R2Attackbit set, if 2
7919             ' look for both rooks, different to SF
7920             If CBool(r And R1OrXrayAttackBit) Then
7921                 If Not CBool(WChecksCounted And R1AttackBit) Then ' count only
7922                     ' once per square!
7923                     If CBool(r And R1XrayAttackBit) Then
7924                         If SqBetween(Target, PieceSqList(BROOK, 1), WKingLoc)
7925                             Then ' xray attack only if in direct line to opp king
7926                             KingDanger = KingDanger + RookCheck \ 3:
7927                             WChecksCounted = (WChecksCounted Or R1AttackBit)
7928                         Else
7929                             KingDanger = KingDanger + 20 ' may be an attack plan
7930                         End If
7931                     Else
7932                         KingDanger = KingDanger + RookCheck: WChecksCounted = (
7933                             WChecksCounted Or R1AttackBit)
7934                     End If
7935                 End If
7936             End If
7937         If CBool(r And R2OrXrayAttackBit) Then
7938             If Not CBool(WChecksCounted And R2AttackBit) Then ' count only
7939                 ' once per square!
7940                 If CBool(r And R2XrayAttackBit) Then
7941                     If SqBetween(Target, PieceSqList(BROOK, 2), WKingLoc)
7942                         Then ' xray attack only if in direct line to opp king
7943                         KingDanger = KingDanger + RookCheck \ 3:
7944                         WChecksCounted = (WChecksCounted Or R2AttackBit)
7945                     Else
7946                         KingDanger = KingDanger + 20 ' may be an attack plan
7947                     End If
7948                 Else
7949                     KingDanger = KingDanger + RookCheck: WChecksCounted = (
7950                         WChecksCounted Or R2AttackBit)
7951                 End If
7952             End If
7953         End If
7954     End If
7955 End If

```





```

7999         ' if pinned pawn then add bonus for attacker
8000         If Board(Target) = WPAWN Then AddScoreVal BPos,
            ThreatByRank.MG \ 2 * Rank(Target), ThreatByRank.EG \
            2 * Rank(Target)
8001         End If
8002     End If
8003     Exit For
8004 Else
8005     If Not (CBool(BAttack(sq) And QAttackBit)) Then Exit For
8006 End If
8007 Next k
8008 End If
8009 Else ' i>4 diagonal
8010     If CBool(BAttack(Target) And QAttackBit) Then ' bishop or queen,
        direction not clear
8011
8012     For k = 1 To 7
8013         sq = Target + Offset * k: Piece = Board(sq)
8014         If Piece = FRAME Then Exit For
8015         If Piece < NO_PIECE Then
8016             If Piece = BQUEEN Or Piece = BBISHOP Then
8017                 If (PieceType(Piece) <> PieceType(Board(Target))) Then
8018                     WPinnedCnt = WPinnedCnt + 1
8019                     If Piece = BBISHOP And Board(Target) = WQUEEN Then
8020                         If bWhiteToMove Then
8021                             AddScoreVal BThreat, 50, 70
8022                             If BAttack(sq) <> 0 And WAttack(sq) = QAttackBit
                                Then
8023                                 AddScoreVal BThreat, 100, 130 ' attacker defended?
                                    less because may be blocker move?
8024                                 If MaxDistance(Target, sq) = 1 Then AddScoreVal
                                    BThreat, 400, 500 ' no blocker option
8025                             End If
8026                         Else
8027                             AddScoreVal BThreat, 1300, 1500
8028                         End If
8029                     End If
8030                     ' if pinned pawn then add bonus for attacker (if pawn cannot capture
                        attacker = distance>1)
8031                     If Board(Target) = WPAWN Then If MaxDistance(Target,
                        sq) > 1 Or Offset < 0 Then AddScoreVal BPos,
                        ThreatByRank.MG \ 2 * Rank(Target), ThreatByRank.EG \
                        2 * Rank(Target)
8032                     End If
8033                 End If
8034             Exit For
8035         Else
8036             If Not (CBool(BAttack(sq) And QAttackBit)) Then Exit For
8037         End If
8038     Next k
8039
8040     End If
8041 End If
8042 End If
8043 ' --- Piece found - exit direction loop
8044     If Board(Target) <> WQUEEN Then Exit Do ' threat Q+K
8045 End If
8046     Target = Target + Offset: rr = rr + 1
8047 Loop While Board(Target) <> FRAME
8048
8049 End If ' <<< Board(Target) <> FRAME
8050
8051 ' Knight Check
8052 If PieceCnt(BKNIGHT) > 0 Then
8053     Target = WKingLoc + KnightOffsets(i)
8054     If Board(Target) <> FRAME Then
8055         If CBool(BAttack(Target) And NAttackBit) Then
8056             bSafe = False ' Safe attack square?

```

```

8057     If PieceColor(Board(Target)) <> BCOL Then If WAttack(Target) = 0
8058     Then bSafe = True
8059     If Not bSafe Then
8060         If CBool(WAttack(Target) = QAttackBit) Then
8061             If AttackBitCnt(BAttack(Target)) > 1 Then
8062                 If Not (AttackBitCnt(WAttack(Target)) > 1 Or PieceColor(Board(
8063                     Target)) = BCOL) Then bSafe = True
8064             End If
8065         End If
8066     End If
8067     If Not CBool(WChecksCounted And N1AttackBit) Then ' count only once per
8068     square!
8069     If bSafe Then
8070         KingDanger = KingDanger + KnightCheck: WChecksCounted = (
8071             WChecksCounted Or N1AttackBit) ' only one knight check expected, two are
8072             very rare
8073         Else
8074             WUnsafeChecks = WUnsafeChecks + 1
8075         End If
8076     End If
8077     ' Knight check fork?
8078     If WAttack(Target) = 0 Or (WAttack(Target) = QAttackBit And (BAttack
8079     (Target) <> NAttackBit)) Then ' no attack
8080
8081     If PieceCnt(WQUEEN) + PieceCnt(WROOK) > 0 Then
8082         For k = 0 To 7
8083             Select Case Board(Target + KnightOffsets(k))
8084             Case WQUEEN: AddScoreVal BThreat, 25, 35
8085             Case WROOK: AddScoreVal BThreat, 15, 20
8086             End Select
8087         Next
8088     End If
8089     End If ' <<< CBool(BAttack(Target) And NAttackBit)
8090     End If ' <<< Board(Target) <> FRAME
8091     End If ' <<< PieceCnt(BKNIGHT) > 0
8092 Next i ' <<< direction
8093
8094 If WKingAttPieces <> 0 Then AddWKingAttackers WKingAttPieces
8095
8096 If WKingAttackersCount > 1 - PieceCnt(BQUEEN) Then
8097     ' total KingDanger
8098     KingDanger = KingDanger + WKingAttackersCount * WKingAttackersWeight _
8099     + 65 * WKingAdjacentZoneAttCnt + Abs(KingLevers > 0) * 64 _
8100     + 190 * (KingOnlyDefended + Undefended) _
8101     - 100 * Abs(CBool(WKDefender And NAttackBit)) _
8102     - 40 * Abs(CBool(WKDefender And BAttackBit)) _
8103     + 152 * (WPinnedCnt + WUnsafeChecks) _
8104     - 885 * Abs(PieceCnt(BQUEEN) = 0) _
8105     - 6 * Bonus \ 8 _
8106     + 5 * Tropism * Tropism \ 16 _
8107     + (BMobility.MG - WMobility.MG) - 10
8108
8109     ' Penalty for king on open or semi-open file
8110     If NonPawnMaterial > 9000 And WPawns(FileNum) = 0 And WKingLoc <>
8111     WKING_START Then
8112         If BPawns(FileNum) = 0 Then KingDanger = KingDanger + 18 Else KingDanger
8113         = KingDanger + 9
8114     End If
8115     r = KingDanger + BPassedPawnAttack * 8 ' passed pawn attacking king?
8116     If r > 100 Then
8117         WKSafety.MG = WKSafety.MG - (r * r) \ 4096
8118         WKSafety.EG = WKSafety.EG - r \ 16
8119     End If
8120 End If

```

```

8117
8118     End If
8119
8120
8121     ' Bonus for a dangerous pawn in the center near the opponent king, for instance pawn e5 against king g8.
8122     If FileNum >= 4 Then If Board(SQ_E4) = BPAWN Then WKSafety.MG = WKSafety.MG -
8123     5
8124     If FileNum <= 5 Then If Board(SQ_D4) = BPAWN Then WKSafety.MG = WKSafety.MG -
8125     5 ' both possible if king centered
8126
8127     ' King tropism bonus, to anticipate slow motion attacks on our king
8128     WKSafety.MG = WKSafety.MG - 7 * Tropism ' closeEnemies
8129
8130 End If
8131
8132 '-----
8133 '--- Black King Safety Eval -----
8134 '-----
8135 RankNum = Rank(BKingLoc): RelRank = (9 - RankNum): FileNum = BKingFile: Bonus = 0:
8136 KingLevers = 0
8137 If (PieceCnt(WQUEEN) * 2 + PieceCnt(WROOK)) > 1 Then
8138     KingDanger = 0
8139     If BPawnCnt = 0 Then MinBKingPawnDistance = 0 Else MinBKingPawnDistance =
8140     MinBKingPawnDistance - 1
8141     If RelRank > 4 Then
8142         BKSafety.EG = BKSafety.EG - 16 * MinBKingPawnDistance
8143     Else
8144         Bonus = BKingShelterStorm(BKingLoc)
8145         If BlackCastled = NO_CASTLE Then
8146             If BKingLoc = SQ_E8 Then
8147                 If BPawns(7) > 0 And PawnsBMax(7) > 5 Then
8148                     If BCanCastleOO() Then
8149                         Bonus = GetMax(Bonus, BKingShelterStorm(SQ_G8))
8150                     End If
8151                 End If
8152             End If
8153             If (BPawns(3) > 0 And PawnsBMax(3) > 5) Or (BPawns(2) > 0 And PawnsBMax(2)
8154             > 5) Then
8155                 If BCanCastleOOO() Then
8156                     Bonus = GetMax(Bonus, BKingShelterStorm(SQ_C8))
8157                 End If
8158             End If
8159         End If
8160     End If
8161     AddScoreVal BKSafety, Bonus, -16 * MinBKingPawnDistance
8162 End If
8163 If bDoBKSafety Then
8164
8165     ' King tropism: firstly, find squares that opponent attacks in our king flank
8166     ' Secondly, add the squares which are attacked twice in that flank
8167     GetKingFlankFiles BKingLoc, r, rr: Tropism = 0
8168     For k = SQ_A1 - 1 + 30 To SQ_A1 - 1 + 70 Step 10 ' start square - 1 of rank 5-8
8169         For Square = k + r To k + rr ' files king flank
8170             If WAttack(Square) <> 0 Then
8171                 Tropism = Tropism + 1: If AttackBitCnt(WAttack(Square)) > 1 Then
8172                     Tropism = Tropism + 1 ' Attacked twice?
8173             End If
8174         Next
8175     Next
8176
8177     ' Pawnless king flank penalty
8178     k = 0
8179     For i = r To rr
8180         If WPawns(i) + BPawns(i) > 0 Then k = 1: Exit For
8181     Next
8182     If k = 0 Then MinusScore BKSafety, PawnlessFlank
8183

```

```

8179 '--- Check threats at king ring
8180 Undefended = 0: KingOnlyDefended = 0: BKingAttPieces = 0
8181 ' add the 2 or 3 squares in front of king ring: king G8 => F6+G6+H6
8182 If RankNum = 8 Then
8183
8184     For Target = BKingLoc - 21 To BKingLoc - 19
8185         If Board(Target) <> FRAME Then
8186             If WAttack(Target) <> 0 Then
8187                 If BAttack(Target) = 0 Or BAttack(Target) = QAttackBit Then
8188                     Undefended = Undefended + 1
8189                     ' exclude double pawn defended squares
8190                     If AttackBitCnt(BAttack(Target) And PAttackBit) < 2 Then
8191                         BKingAttPieces = BKingAttPieces Or WAttack(Target)
8192                         If Board(Target) = BPAWN Then
8193                             If CBool(WAttack(Target) And PAttackBit) Then KingLevers =
8194                                 KingLevers + 1
8195                         End If
8196                     End If
8197                 End If
8198             End If
8199         End If
8200     Next
8201 End If
8202
8203 For i = 0 To 7
8204     Offset = DirectionOffset(i): Target = BKingLoc + Offset
8205     If Board(Target) <> FRAME Then
8206         If WAttack(Target) <> 0 Then
8207             ' King attacks are added later in attack array, so distance=1 and BAttack=0 is equal to king
8208             attack only
8209             If BAttack(Target) = 0 Then KingOnlyDefended = KingOnlyDefended + 1
8210             BKingAdjacentZoneAttCnt = BKingAdjacentZoneAttCnt + AttackBitCnt(
8211                 WAttack(Target) And Not PAttackBit)
8212             ' exclude double pawn defended squares
8213             If AttackBitCnt(BAttack(Target) And PAttackBit) < 2 Then
8214                 BKingAttPieces = BKingAttPieces Or WAttack(Target)
8215                 If Board(Target) = BPAWN Then
8216                     If CBool(WAttack(Target) And PAttackBit) Then KingLevers =
8217                         KingLevers + 1
8218                 End If
8219             End If
8220         End If
8221         BKDefender = BKDefender Or BAttack(Target)
8222         rr = 1 ' rr=Distance to King
8223
8224         Do ' loop for a direction
8225             r = WAttack(Target)
8226             If CBool(r And QRBAttackBit) Then
8227                 bSafe = False ' Safe attack square?
8228                 If PieceColor(Board(Target)) <> WCOL Then
8229                     If BAttack(Target) = 0 Then
8230                         If rr = 1 Then
8231                             If AttackBitCnt(WAttack(Target)) > 1 Then bSafe = True
8232                         Else
8233                             bSafe = True
8234                         End If
8235                     End If
8236                 End If
8237             End If
8238             ' Queen safe checks
8239             If bSafe Then
8240                 If CBool(r And QAttackBit) Then
8241                     If Not CBool(BChecksCounted And QAttackBit) Then
8242                         KingDanger = KingDanger + QueenCheck
8243                         BChecksCounted = (BChecksCounted Or QAttackBit)
8244                     End If
8245                 End If
8246             End If
8247             If CBool(r And RBoRxrayAttackBit) Then
8248                 If Not bSafe And rr > 1 Then ' not defended by king
8249                     ' For minors and rooks, also consider the square as safe if attacked twice,

```

```

8240 ' and only defended by our queen.
8241 If CBool(BAttack(Target) = QAttackBit) Then
8242     If AttackBitCnt(WAttack(Target)) > 1 Then
8243         If Not (AttackBitCnt(BAttack(Target)) > 1 Or PieceColor(
8244             Board(Target)) = WCOL) Then
8245             bSafe = True
8246         End If
8247     End If
8248 End If
8249 '(i=0-3: orthogonal offset, 4-7:diagonal)
8250 ' Rook checks
8251 If i < 4 Then
8252     If CBool(r And ROrXrayAttackBit) Then ' R1Attackbit or R2Attackbit set, if 2
8253         rooks only one is counted per square
8254     If bSafe Then
8255         ' look for both rooks, different to SF
8256     If CBool(r And R1OrXrayAttackBit) Then
8257         If Not CBool(BChecksCounted And R1AttackBit) Then ' count only
8258             once per square!
8259         If CBool(r And R1XrayAttackBit) Then
8260             If SqBetween(Target, PieceSqList(WROOK, 1), BKingLoc)
8261             Then ' xray attack only if in direct line to opp king
8262                 KingDanger = KingDanger + RookCheck \ 3:
8263                 BChecksCounted = (BChecksCounted Or R1AttackBit)
8264             Else
8265                 KingDanger = KingDanger + 20 ' may be an attack plan
8266             End If
8267         Else
8268             KingDanger = KingDanger + RookCheck: BChecksCounted = (
8269                 BChecksCounted Or R1AttackBit)
8270         End If
8271     End If
8272 End If
8273 '
8274 If CBool(r And R2OrXrayAttackBit) Then
8275     If Not CBool(BChecksCounted And R2AttackBit) Then ' count only
8276         once per square!
8277     If CBool(r And R2XrayAttackBit) Then
8278         If SqBetween(Target, PieceSqList(WROOK, 2), BKingLoc)
8279         Then ' xray attack only if in direct line to opp king
8280             KingDanger = KingDanger + RookCheck \ 3:
8281             BChecksCounted = (BChecksCounted Or R2AttackBit)
8282         Else
8283             KingDanger = KingDanger + 20 ' may be an attack plan
8284         End If
8285     Else
8286         KingDanger = KingDanger + RookCheck: BChecksCounted = (
8287             BChecksCounted Or R2AttackBit)
8288     End If
8289 End If
8290 End If
8291 Else ' i >= 4
8292     ' Bishop checks
8293     If CBool(r And BXrayAttackBit) Then ' B1Attackbit or B2Attackbit set, if 2
8294         rooks only one is counted
8295     If Not CBool(BChecksCounted And B1AttackBit) Then ' count only
8296         once! only one bishop has same color as king
8297     If bSafe Then
8298         If CBool(r And BXrayAttackBit) Then
8299             If SqBetween(Target, PieceSqList(WBISHOP, 1), BKingLoc)
8300             Or
8301             SqBetween(Target, PieceSqList(WBISHOP, 2), BKingLoc)
8302             Then ' xray attack only if in direct line to opp king

```

```

8294         ' do not count xray if through a blocked pawn
8295         If Board(Target + Offset) <> WPAWN Or Board(Target +
            Offset + SQ_UP) >= NO_PIECE Then KingDanger =
            KingDanger + BishopCheck \ 3: BChecksCounted = (
            BChecksCounted Or B1AttackBit)
8296     Else
8297         KingDanger = KingDanger + 10 ' may be an attack plan
8298     End If
8299     Else
8300         KingDanger = KingDanger + BishopCheck: BChecksCounted =
            (BChecksCounted Or B1AttackBit)
8301     End If
8302     Else
8303         BUnsafeChecks = BUnsafeChecks + 1
8304     End If
8305     End If
8306     End If
8307     End If
8308     End If
8309 End If ' r and QRBAttackbit
8310 If Board(Target) < NO_PIECE Then ' Piece found
8311     '
8312     '--- Check for pinned pieces
8313     '
8314     If (Board(Target) And 1) = BCOL Then ' own piece
8315         If i < 4 Then ' orthogonal
8316             If CBool(WAttack(Target) And QRAttackBit) Then ' rook or queen,
            direction not clear
8317                 For k = 1 To 7
8318                     sq = Target + Offset * k: Piece = Board(sq)
8319                     If Piece = FRAME Then Exit For
8320                     If Piece < NO_PIECE Then
8321                         If Piece = WQUEEN Or Piece = WROOK Then
8322                             If (PieceType(Piece) <> PieceType(Board(Target))) Then
8323                                 If Piece = WROOK And Board(Target) = BQUEEN Then
8324                                     If Not bWhiteToMove Then
8325                                         AddScoreVal WThreat, 30, 50
8326                                         If WAttack(sq) <> 0 And BAttack(sq) = QAttackBit
8327                                         Then
8328                                             AddScoreVal WThreat, 75, 100 ' attacker defended? less
8329                                             because may be blocker move?
8330                                             If MaxDistance(Target, sq) = 1 Then AddScoreVal
8331                                             WThreat, 400, 500 ' no blocker option
8332                                         End If
8333                                     Else
8334                                         AddScoreVal WThreat, 1200, 1400
8335                                     End If
8336                                 End If
8337                                 BPinnedCnt = BPinnedCnt + 1
8338                                 ' if pinned pawn then add bonus for attacker
8339                                 If Board(Target) = BPAWN Then AddScoreVal WPos,
8340                                 ThreatByRank.MG \ 2 * (9 - Rank(Target)),
8341                                 ThreatByRank.EG \ 2 * (9 - Rank(Target))
8342                             End If
8343                         End If
8344                     Exit For
8345                 Else
8346                     If Not CBool(WAttack(sq) And QRAttackBit) Then Exit For
8347                 End If
8348             Next k
8349         End If
8350     Else ' i>4 diagonal
8351         If CBool(WAttack(Target) And QBAttackBit) Then ' bishop or queen,
            direction not clear
8352             For k = 1 To 7
8353                 sq = Target + Offset * k: Piece = Board(sq)
8354                 If Piece = FRAME Then Exit For

```

```

8351         If Piece < NO_PIECE Then
8352             If Piece = WQUEEN Or Piece = WBISHOP Then
8353                 If (PieceType(Piece) <> PieceType(Board(Target))) Then
8354                     BPinnedCnt = BPinnedCnt + 1
8355                     If Piece = WBISHOP And Board(Target) = BQUEEN Then
8356                         If Not bWhiteToMove Then
8357                             AddScoreVal WThreat, 50, 70
8358                             If WAttack(sq) <> 0 And BAttack(sq) = QAttackBit
8359                                 Then
8360                                     AddScoreVal WThreat, 100, 130 'attacker defended?
8361                                     less because may be blocker move?
8362                                     If MaxDistance(Target, sq) = 1 Then AddScoreVal
8363                                         WThreat, 400, 500 'no blocker option
8364                                     End If
8365                                 Else
8366                                     AddScoreVal WThreat, 1300, 1500
8367                                 End If
8368                             End If
8369                             ' if pinned pawn then add bonus for attacker (if pawn cannot capture
8370                             attacker = distance>1)
8371                             If Board(Target) = BPAWN Then If MaxDistance(Target,
8372                                 sq) > 1 Or Offset > 0 Then AddScoreVal WPos,
8373                                 ThreatByRank.MG \ 2 * (9 - Rank(Target)),
8374                                 ThreatByRank.EG \ 2 * (9 - Rank(Target))
8375                             End If
8376                         End If
8377                     Exit For
8378                 Else
8379                     If Not CBool(WAttack(sq) And QBAttackBit) Then Exit For
8380                 End If
8381             Next k
8382
8383         End If
8384     End If
8385     ' --- Piece found - exit direction loop
8386     If Board(Target) <> BQUEEN Then Exit Do
8387 End If
8388 Target = Target + Offset: rr = rr + 1
8389 Loop While Board(Target) <> FRAME
8390
8391 End If ' <<< Board(Target) <> FRAME
8392 ' Knight Check
8393 If PieceCnt(WKNIGHT) > 0 Then
8394     Target = BKingLoc + KnightOffsets(i)
8395     If Board(Target) <> FRAME Then
8396         If CBool(WAttack(Target) And NAttackBit) Then
8397             bSafe = False ' Safe attack square?
8398             If PieceColor(Board(Target)) <> WCOL Then If BAttack(Target) = 0
8399                 Then bSafe = True
8400             If Not bSafe Then
8401                 If CBool(BAttack(Target) = QAttackBit) Then
8402                     If AttackBitCnt(WAttack(Target)) > 1 Then
8403                         If Not (AttackBitCnt(BAttack(Target)) > 1 Or PieceColor(Board(
8404                             Target)) = WCOL) Then
8405                             bSafe = True
8406                         End If
8407                     End If
8408                 End If
8409             End If
8410         End If
8411     End If
8412     If Not CBool(BChecksCounted And N1AttackBit) Then ' count only once per
8413     square!
8414         If bSafe Then
8415             KingDanger = KingDanger + KnightCheck: BChecksCounted = (
8416                 BChecksCounted Or N1AttackBit) ' only one knight check expected, two are
8417                 very rare
8418         Else
8419             BUnsafeChecks = BUnsafeChecks + 1
8420         End If
8421     End If
8422 End Sub

```



```

8407         End If
8408     End If
8409     ' Knight check fork?
8410     If BAttack(Target) = 0 Or (BAttack(Target) = QAttackBit And (WAttack
(Target) <> NAttackBit)) Then ' field not defended or by queen only but other
attacker

8411
8412     If PieceCnt(BQUEEN) + PieceCnt(BROOK) > 0 Then
8413         For k = 0 To 7
8414
8415             Select Case Board(Target + KnightOffsets(k))
8416                 Case BQUEEN: AddScoreVal WThreat, 25, 35
8417                 Case BROOK: AddScoreVal WThreat, 15, 20
8418             End Select
8419
8420         Next
8421     End If
8422
8423     End If
8424     End If '<<< CBool(WAttack(Target) And NAttackBit)
8425     End If '<<< Board(Target) <> FRAME
8426     End If '<<< PieceCnt(WKNIGHT) > 0
8427     Next i '<<< direction
8428
8429     If BKingAttPieces <> 0 Then AddBKingAttackers BKingAttPieces
8430
8431     If BKingAttackersCount > 1 - PieceCnt(WQUEEN) Then
8432
8433
8434     ' total KingDanger
8435     KingDanger = KingDanger + BKingAttackersCount * BKingAttackersWeight _
8436         + 65 * BKingAdjacentZoneAttCnt + Abs(KingLevers > 0) * 64 _
8437         + 190 * (KingOnlyDefended + Undefended) _
8438         - 100 * Abs(CBool(BKDefender And NAttackBit)) _
8439         - 40 * Abs(CBool(BKDefender And BAttackBit)) _
8440         + 152 * (BPinnedCnt + BUnsafeChecks) _
8441         - 885 * Abs(PieceCnt(WQUEEN) = 0) _
8442         - 6 * Bonus \ 8 _
8443         + 5 * Tropism * Tropism \ 16 _
8444         + (WMobility.MG - BMobility.MG) - 10
8445
8446     ' Penalty for king on open or semi-open file
8447     If NonPawnMaterial > 9000 And BPawns(FileNum) = 0 And BKingLoc <>
BKING_START Then
8448         If WPawns(FileNum) = 0 Then KingDanger = KingDanger + 18 Else KingDanger
= KingDanger + 9
8449     End If
8450     r = KingDanger + WPassedPawnAttack * 8 ' passed pawn attacking king?
8451     If r > 100 Then
8452         BKSafety.MG = BKSafety.MG - (r * r) \ 4096
8453         BKSafety.EG = BKSafety.EG - r \ 16
8454     End If
8455 End If
8456
8457 End If
8458
8459 ' Bonus for a dangerous pawn in the center near the opponent king, for instance pawn e5 against king g8.
8460 If FileNum >= 4 Then If Board(SQ_E5) = WPAWN Then BKSafety.MG = BKSafety.MG -
5
8461 If FileNum <= 5 Then If Board(SQ_D5) = WPAWN Then BKSafety.MG = BKSafety.MG -
5 ' both possible if king centered
8462
8463
8464 ' King tropism bonus, to anticipate slow motion attacks on our king
8465 BKSafety.MG = BKSafety.MG - 7 * Tropism ' closeEnemies
8466
8467 End If
8468 End If ' Endgame

```

```

8469
8470
8471 '--- Endgame King distance to best pawn. Not in PawnHash because "Fifty" may be different
8472 If bEndgame Or (WPawnCnt + BPawnCnt <= 8) Then
8473     If WBestPawn > 0 Then
8474         i = MaxDistance(WBestPawn, WKingLoc)
8475         AddScoreVal WPos, 0, (7 - i) * (7 - i) * 6
8476         If Rank(WBestPawn) >= 5 Then AddScoreVal WPos, 0, ((Rank(WBestPawn - 4) * Rank(
WBestPawn - 4)) * (Fifty + 1)) \ 3 * 2 '--- bonus for move pawn in endgame
8477     ElseIf BBestPawn > 0 Then
8478         i = MaxDistance(BBestPawn, WKingLoc) '--- Close to Opp Pawn
8479         AddScoreVal WPos, 0, (7 - i) * (7 - i)
8480     End If
8481     If BBestPawn > 0 Then
8482         i = MaxDistance(BBestPawn, BKingLoc)
8483         If i > 2 Then AddScoreVal BPos, 0, (7 - i) * (7 - i) * 6
8484         If RankB(BBestPawn) >= 5 Then AddScoreVal BPos, 0, ((RankB(BBestPawn - 4) *
RankB(BBestPawn - 4)) * (Fifty + 1)) \ 3 * 2 '--- bonus for move pawn in endgame
8485     ElseIf WBestPawn > 0 Then
8486         i = MaxDistance(WBestPawn, BKingLoc) '--- Close to Opp Pawn
8487         AddScoreVal BPos, 0, (7 - i) * (7 - i)
8488     End If
8489 Else
8490     '--- Midgame
8491 End If
8492 'add kings to attack array
8493 r = 0: rr = 0
8494
8495 For i = 0 To 7
8496     Offset = DirectionOffset(i)
8497     Target = WKingLoc + Offset
8498     If Board(Target) <> FRAME Then
8499         WAttack(Target) = WAttack(Target) Or KAttackBit
8500         If PieceColor(Board(Target)) = COL_BLACK Then
8501             If AttackBitCnt(WAttack(Target)) > AttackBitCnt(BAttack(Target)) Then r = r +
1 ' King attacks unprotected piece
8502         End If
8503     End If
8504     Target = BKingLoc + Offset
8505     If Board(Target) <> FRAME Then
8506         BAttack(Target) = BAttack(Target) Or KAttackBit
8507         If PieceColor(Board(Target)) = COL_WHITE Then
8508             If AttackBitCnt(BAttack(Target)) > AttackBitCnt(WAttack(Target)) Then rr = rr
+ 1 ' King attacks unprotected piece
8509         End If
8510     End If
8511 Next
8512
8513 If r > 0 Then
8514     If r = 1 Then AddScore WThreat, KingOnOneBonus Else AddScore WThreat,
KingOnManyBonus
8515 End If
8516 If rr > 0 Then
8517     If rr = 1 Then AddScore BThreat, KingOnOneBonus Else AddScore BThreat,
KingOnManyBonus
8518 End If
8519
8520 '-----
8521 '--- Step 6: Eval threats -----
8522 '-----
8523 CalcThreats 'in WThreat and BThreat
8524
8525 If WWeakUnopposedCnt > 0 Then
8526     If PieceCnt(BQUEEN) + PieceCnt(BROOK) > 0 Then AddScoreWithFactor BThreat,
WeakUnopposedPawn, WWeakUnopposedCnt
8527 End If
8528 If BWeakUnopposedCnt > 0 Then
8529     If PieceCnt(WQUEEN) + PieceCnt(WROOK) > 0 Then AddScoreWithFactor WThreat,

```

```

WeakUnopposedPawn, BWeakUnopposedCnt
8530 End If
8531
8532 ' Trapped bishops at a7/h7, a2/h2
8533 If PieceCnt(WBISHOP) > 0 Then
8534     ' white bishop not defended trapped at A7 by black pawn B6 (or if pawn can move to B6)
8535     If Board(SQ_A7) = WBISHOP Then
8536         If BAttack(SQ_B6) > 0 And WAttack(SQ_A7) = 0 Then
8537             If Board(SQ_B6) = BPAWN Or (Not bWhiteToMove And Board(SQ_B6) >= NO_PIECE And
Board(SQ_B7) = BPAWN) Then
8538                 AddScoreVal BThreat, ScoreBishop.MG \ 3, ScoreBishop.MG \ 4
8539             End If
8540         End If
8541     End If
8542     If Board(SQ_H7) = WBISHOP Then
8543         If BAttack(SQ_G6) > 0 And WAttack(SQ_H7) = 0 Then
8544             If Board(SQ_G6) = BPAWN Or (Not bWhiteToMove And Board(SQ_G6) >= NO_PIECE And
Board(SQ_G7) = BPAWN) Then
8545                 AddScoreVal BThreat, ScoreBishop.MG \ 3, ScoreBishop.MG \ 4
8546             End If
8547         End If
8548     End If
8549 End If
8550 If PieceCnt(BBISHOP) > 0 Then
8551     If Board(SQ_A2) = BBISHOP Then
8552         If WAttack(SQ_B3) > 0 And BAttack(SQ_A2) = 0 Then
8553             If Board(SQ_B3) = WPAWN Or (bWhiteToMove And Board(SQ_B3) >= NO_PIECE And
Board(SQ_B2) = WPAWN) Then
8554                 AddScoreVal WThreat, ScoreBishop.MG \ 3, ScoreBishop.MG \ 4
8555             End If
8556         End If
8557     End If
8558     If Board(SQ_H2) = BBISHOP Then
8559         If WAttack(SQ_G3) > 0 And BAttack(SQ_H2) = 0 Then
8560             If Board(SQ_G3) = WPAWN Or (bWhiteToMove And Board(SQ_G3) >= NO_PIECE And
Board(SQ_G2) = WPAWN) Then
8561                 AddScoreVal WThreat, ScoreBishop.MG \ 3, ScoreBishop.MG \ 4
8562             End If
8563         End If
8564     End If
8565 End If
8566 '
8567 '--- Passed pawns (white and black). done here because full attack info is needed
8568 '
8569 WFrontMostPassedPawnRank = 0: BFrontMostPassedPawnRank = 0
8570
8571 For a = 1 To PassedPawnsCnt
8572     Dim AttackedFromBehind As Long, DefendedFromBehind As Long
8573     Square = PassedPawns(a): FileNum = File(Square): RankNum = Rank(Square)
8574     MBonus = 0: EBonus = 0: UnsafeCnt = 0
8575     If PieceColor(Board(Square)) = COL_WHITE Then
8576         ' White piece
8577         OwnCol = COL_WHITE: OppCol = COL_BLACK: MoveUp = SQ_UP
8578         RelRank = RankNum: OwnKingLoc = WKingLoc: OppKingLoc = BKingLoc
8579         ' Attack Opp King?
8580         If RelRank > WFrontMostPassedPawnRank Then WFrontMostPassedPawnRank = RankNum
8581         If PieceCnt(WBISHOP) > 0 Then ' Bishop with same color as promote square? ( not SF logic )
8582             sq = SQ_A1 + FileNum - 1 + 7 * MoveUp
8583             If ColorSq(sq) = COL_WHITE Then
8584                 r = Sgn(Sgn(WBishopsOnWhiteSq) - Sgn(BBishopsOnWhiteSq)) ' 0 if both sides have
same bishop color, else +1 or -1
8585                 If r <> 0 Then MBonus = MBonus + r * (10 + (RelRank - 1) * 3): EBonus =
EBonus + r * (20 + (RelRank - 1) * (RelRank - 1))
8586             Else
8587                 r = Sgn(Sgn(WBishopsOnBlackSq) - Sgn(BBishopsOnBlackSq)) ' 0 if both sides have
same bishop color, else +1 or -1
8588                 If r <> 0 Then MBonus = MBonus + r * (10 + (RelRank - 1) * 3): EBonus =
EBonus + r * (20 + (RelRank - 1) * (RelRank - 1))

```

```

8589         End If
8590     End If
8591 Else
8592     OwnCol = COL_BLACK: OppCol = COL_WHITE: MoveUp = SQ_DOWN
8593     ' Black piece
8594     RelRank = (9 - RankNum): OwnKingLoc = BKingLoc: OppKingLoc = WKingLoc
8595     If RelRank > BFrontMostPassedPawnRank Then BFrontMostPassedPawnRank = RelRank
8596     If PieceCnt(BBISHOP) > 0 Then ' Bishop with same color as promote square?
8597         sq = SQ_A1 + FileNum - 1
8598         If ColorSq(sq) = COL_WHITE Then
8599             r = Sgn(Sgn(BBishopsOnWhiteSq) - Sgn(WBishopsOnWhiteSq)) ' 0 if both sides have
            same bishop color, else +1 or -1
8600             If r <> 0 Then MBonus = MBonus + r * (10 + (RelRank - 1) * 3): EBonus =
            EBonus + r * (20 + (RelRank - 1) * (RelRank - 1))
8601         Else
8602             r = Sgn(Sgn(BBishopsOnBlackSq) - Sgn(WBishopsOnBlackSq)) ' 0 if both sides have
            same bishop color, else +1 or -1
8603             If r <> 0 Then MBonus = MBonus + r * (10 + (RelRank - 1) * 3): EBonus =
            EBonus + r * (20 + (RelRank - 1) * (RelRank - 1))
8604         End If
8605     End If
8606 End If
8607 '
8608 '--- Path to promote square blocked? => penalty
8609 '
8610 r = RelRank
8611 rr = PassedDanger(RelRank)
8612 MBonus = MBonus + PassedPawnRankBonus(r).MG: EBonus = EBonus + PassedPawnRankBonus
(r).EG
8613 ' Bonus based on rank ' SF9
8614 If rr <> 0 Then
8615     BlockSq = Square + MoveUp
8616     If Board(BlockSq) <> FRAME Then
8617         ' Adjust bonus based on the king's proximity
8618         AttackedFromBehind = 0: DefendedFromBehind = 0
8619         EBonus = EBonus + (GetMin(5, MaxDistance(BlockSq, OppKingLoc)) * 5 - GetMin(5,
            MaxDistance(BlockSq, OwnKingLoc)) * 2) * rr
8620         'If blockSq is not the queening square then consider also a second push
8621         If RelRank <> 7 Then EBonus = EBonus - MaxDistance(BlockSq + MoveUp,
            OwnKingLoc) * rr
8622         'If the pawn is free to advance, then increase the bonus
8623         If Board(BlockSq) >= NO_PIECE Then
8624             k = 0: bAllDefended = True: BlockSqDefended = True: BlockSqUnsafe = False
8625             ' Rook or Queen attacking/defending from behind?
8626             If CBool(BAttack(Square) And QRAttackBit) Or CBool(WAttack(Square) And
            QRAttackBit) Then
8627                 sq = Square
8628                 For RankPath = RelRank - 1 To 1 Step -1
8629                     sq = sq - MoveUp ' move down to rank 1
8630                     Select Case Board(sq)
8631                     Case NO_PIECE:
8632                     Case BROOK, BQUEEN:
8633                         If OwnCol = COL_WHITE Then
8634                             BlockSqUnsafe = True: AttackedFromBehind = 1
8635                         Else
8636                             DefendedFromBehind = 1:
8637                         End If
8638                     Exit For
8639                     Case WROOK, WQUEEN:
8640                         If OwnCol = COL_BLACK Then
8641                             BlockSqUnsafe = True: AttackedFromBehind = 1
8642                         Else
8643                             DefendedFromBehind = 1
8644                         End If
8645                     Exit For
8646                     Case Else:
8647                         Exit For
8648                 End Select

```

```

8649         Next
8650     End If
8651
8652     sq = Square
8653     For RankPath = RelRank + 1 To 8
8654         sq = sq + MoveUp
8655         OwnAttCnt = AttackBitCnt(AttackByCol(OwnCol, sq)) + DefendedFromBehind
8656         If OwnAttCnt = 0 And sq <> OwnKingLoc Then
8657             bAllDefended = False: If sq = BlockSq Then BlockSqDefended = False
8658         End If
8659         If PieceColor(Board(sq)) = OppCol Then
8660             If sq = BlockSq Then BlockSqUnsafe = True Else UnsafeCnt = UnsafeCnt + 1
8661             ElseIf AttackBitCnt(AttackByCol(OppCol, sq)) + AttackedFromBehind > 0 Then
8662                 If CBool(AttackByCol(OwnCol, sq) And PAttackBit) And Not CBool(
8663                     AttackByCol(OppCol, sq) And PAttackBit) Then
8664                     ' Own pawn support but no enemy pawn attack: square is safe ( NOT SF LOGIC )
8665                 Else
8666                     If sq = BlockSq Then BlockSqUnsafe = True Else UnsafeCnt = UnsafeCnt +
8667                         1
8668                 End If
8669             End If
8670         End If
8671     Next RankPath
8672
8673     If BlockSqUnsafe Then UnsafeCnt = UnsafeCnt + 1
8674     If UnsafeCnt = 0 Then
8675         k = 20
8676     ElseIf Not BlockSqUnsafe Then
8677         k = 9 '- UnsafeCnt
8678     Else
8679         k = 0
8680     End If
8681     If bAllDefended Then
8682         k = k + 6 '- UnsafeCnt \ 2
8683     ElseIf BlockSqDefended Then
8684         k = k + 4 '- UnsafeCnt \ 2
8685     End If
8686     ' If protected by more than one rook or queen, assign extra bonus
8687     If k > 0 Then
8688         If OwnCol = COL_WHITE Then
8689             If AttackBitCnt((WAttack(Square) And QOrXrayROrXrayAttackBit)) > 1 Then
8690                 k = k + 2
8691             Else
8692                 If AttackBitCnt((BAttack(Square) And QOrXrayROrXrayAttackBit)) > 1 Then
8693                     k = k + 2
8694                 End If
8695             End If
8696         End If
8697         '-- add bonus
8698         If k <> 0 Then MBonus = MBonus + k * rr: EBonus = EBonus + k * rr
8699     Else
8700         If PieceColor(Board(BlockSq)) = OwnCol Then MBonus = MBonus + rr + (r - 1) *
8701             2: EBonus = EBonus + rr + (r - 1) * 2 ' r-1 because rank r is 0 based in C
8702         End If
8703     End If
8704     End If 'rr>0
8705 '
8706 If UnsafeCnt > 0 Then MBonus = MBonus - UnsafeCnt * 8: EBonus = EBonus - UnsafeCnt
8707 ' hindered passed pawn
8708 '
8709 If OwnCol = COL_WHITE Then
8710     If WPawnCnt > BPawnCnt Then EBonus = EBonus + EBonus \ 4
8711     MBonus = MBonus + PassedPawnFileBonus(FileNum).MG: EBonus = EBonus +
8712         PassedPawnFileBonus(FileNum).EG
8713     If BNonPawnMaterial = 0 Then EBonus = EBonus + 20
8714     If Board(Square + SQ_UP) = BPAWN Then MBonus = MBonus \ 2: EBonus = EBonus \ 2 '
8715         supporter sacrify needed
8716     If bWhiteToMove Then MBonus = (MBonus * 105) \ 100: EBonus = (EBonus * 105) \
8717         100
8718     AddScoreVal WPassed, MBonus, EBonus

```

```

8708     If 1000 + EBonus > WBestPawnVal Then WBestPawn = Square: WBestPawnVal = 1000 +
EBonus 'new best pawn
8709     If bEvalTrace Then WriteTrace "WPassed: " & LocCoord(Square) & ">" & MBonus &
", " & EBonus
8710     ElseIf OwnCol = COL_BLACK Then
8711         If BPawnCnt > WPawnCnt Then EBonus = EBonus + EBonus \ 4
8712         MBonus = MBonus + PassedPawnFileBonus(FileNum).MG: EBonus = EBonus +
PassedPawnFileBonus(FileNum).EG
8713         If WNonPawnMaterial = 0 Then EBonus = EBonus + 20
8714         If Board(Square + SQ_DOWN) = WPAWN Then MBonus = MBonus \ 2: EBonus = EBonus \ 2
' supporter sacrifice needed
8715         If Not bWhiteToMove Then MBonus = (MBonus * 105) \ 100: EBonus = (EBonus * 105
) \ 100
8716         AddScoreVal BPassed, MBonus, EBonus
8717         If 1000 + EBonus > BBestPawnVal Then BBestPawn = Square: BBestPawnVal = 1000 +
EBonus 'new best pawn
8718         If bEvalTrace Then WriteTrace "BPassed: " & LocCoord(Square) & ">" & MBonus &
", " & EBonus
8719     End If
8720 Next a
8721 '
8722 '---<<< end Passed pawn
8723 '
8724 '--- If both sides have only pawns, score for potential unstoppable pawns
8725 If WNonPawnMaterial + BNonPawnMaterial = 0 Then
8726     If WFrontMostPassedPawnRank > 0 Then AddScoreVal WPassed, 0,
WFrontMostPassedPawnRank * 20
8727     If BFrontMostPassedPawnRank > 0 Then AddScoreVal BPassed, 0,
BFrontMostPassedPawnRank * 20 ' RelRank is used, so >0 is correct
8728 End If
8729 '
8730 '--- Penalty for pawns on same color square of bishop
8731 '
8732 If PieceCnt(WBISHOP) > 0 Then
8733     r = WPawnCntOnWhiteSq * WBishopsOnWhiteSq + (WPawnCnt - WPawnCntOnWhiteSq) *
WBishopsOnBlackSq
8734     If r <> 0 Then
8735         r = r * (1 + WCenterPawnsBlocked)
8736         AddScoreVal WPos, -3 * r, -5 * r
8737     End If
8738     ' Bonus for bishop on a long diagonal if it can "see" both center squares and no pawns
8739     If WBishopsOnWhiteSq > 0 And Not bEndgame Then
8740         If CBool(WAttack(SQ_E4) And BAttackBit) Then
8741             If PieceType(Board(SQ_E4)) <> PT_PAWN Then
8742                 If CBool(WAttack(SQ_D5) And BAttackBit) Then If PieceType(Board(SQ_D5)) <>
PT_PAWN Then WPos.MG = WPos.MG + 22
8743             End If
8744         End If
8745     End If
8746     If WBishopsOnBlackSq > 0 Then
8747         If CBool(WAttack(SQ_D4) And BAttackBit) Then
8748             If PieceType(Board(SQ_D4)) <> PT_PAWN Then
8749                 If CBool(WAttack(SQ_E5) And BAttackBit) Then If PieceType(Board(SQ_E5)) <>
PT_PAWN Then WPos.MG = WPos.MG + 22
8750             End If
8751         End If
8752     End If
8753 End If
8754 If PieceCnt(BBISHOP) > 0 Then
8755     r = BPawnCntOnWhiteSq * BBishopsOnWhiteSq + (BPawnCnt - BPawnCntOnWhiteSq) *
BBishopsOnBlackSq
8756     If r <> 0 Then
8757         r = r * (1 + BCenterPawnsBlocked)
8758         AddScoreVal BPos, -3 * r, -5 * r
8759     End If
8760     ' Bonus for bishop on a long diagonal if it can "see" both center squares and no pawns
8761     If BBishopsOnWhiteSq > 0 And Not bEndgame Then
8762         If CBool(BAttack(SQ_D5) And BAttackBit) Then

```

```

8763         If PieceType(Board(SQ_D5)) <> PT_PAWN Then
8764             If CBool(BAttack(SQ_E4) And BAttackBit) Then If PieceType(Board(SQ_E4)) <>
                PT_PAWN Then BPos.MG = BPos.MG + 22
8765         End If
8766     End If
8767 End If
8768 If BBishopsOnBlackSq > 0 Then
8769     If CBool(BAttack(SQ_E5) And BAttackBit) Then
8770         If PieceType(Board(SQ_E5)) <> PT_PAWN Then
8771             If CBool(BAttack(SQ_D4) And BAttackBit) Then If PieceType(Board(SQ_D4)) <>
                PT_PAWN Then BPos.MG = BPos.MG + 22
8772         End If
8773     End If
8774 End If
8775 End If
8776 '
8777 '--->>> Pawn Islands (groups of pawns) ---
8778 '
8779 r = 0: bWIsland = False 'r: white islands
8780 rr = 0: bBIsland = False 'rr: black islands
8781
8782 For FileNum = 1 To 9
8783     If WPawns(FileNum) <= 0 Then ' File WPawns(9) = -1
8784         bWIsland = True
8785     ElseIf bWIsland Then
8786         r = r + 1: bWIsland = False ' empty file and pawn onleft side > island
8787     End If
8788     If BPawns(FileNum) <= 0 Then ' File BPawns(9) = -1
8789         bBIsland = True
8790     ElseIf bBIsland Then
8791         rr = rr + 1: bBIsland = False ' empty file and pawn onleft side > island
8792     End If
8793 Next
8794
8795 If r > 0 Then AddScoreVal WPawnStruct, -15 * r, -25 * r ' Penalty for each island
8796 If rr > 0 Then AddScoreVal BPawnStruct, -15 * rr, -25 * rr
8797 '---<<< end Pawn Islands ---
8798 '
8799 '-----
8800 '--- Step 7: Calculate total material values and endgame scale factors ---
8801 '-----
8802 '
8803 ' Piece values were set in SetGamePhase
8804 Dim AllTotal As TScore, MatEval As Long
8805 AllTotal.MG = Material ' Based on MG, no need to recalc ' (PieceCnt(WQUEEN) - PieceCnt(BQUEEN)) *
ScoreQueen.MG + (PieceCnt(WROOK) - PieceCnt(BROOK)) * ScoreRook.MG + (PieceCnt(WBISHOP) -
PieceCnt(BBISHOP)) * ScoreBishop.MG + (PieceCnt(WKNIGHT) - PieceCnt(BKNIGHT)) * ScoreKnight.MG +
(WPawnCnt - BPawnCnt) * ScorePawn.MG
8806 AllTotal.EG = (PieceCnt(WQUEEN) - PieceCnt(BQUEEN)) * ScoreQueen.EG + (PieceCnt(
WROOK) - PieceCnt(BROOK)) * ScoreRook.EG + (PieceCnt(WBISHOP) - PieceCnt(BBISHOP)) *
ScoreBishop.EG + (PieceCnt(WKNIGHT) - PieceCnt(BKNIGHT)) * ScoreKnight.EG + (
WPawnCnt - BPawnCnt) * ScorePawn.EG
8807 MatEval = ScaleScore(AllTotal)
8808 If bEvalTrace Then
8809     Debug.Print "Material: " & EvalSFTto100(AllTotal.MG) & ", " & EvalSFTto100(
AllTotal.EG)
8810 End If
8811 '
8812 '
8813 '--- Calculate SPACE in opening phase for safe squares in center (files 3-6, ranks 2-4)
8814 '
8815 If NonPawnMaterial > SPACE_THRESHOLD Then
8816     r = 0: rr = 0
8817     For k = 3 To 6 ' files 3-6
8818         '--- White space
8819         Offset = PawnsWMin(k): Target = k + 20
8820         For RankNum = 2 To 4 ' WHITE
8821             Target = Target + 10

```



```

8822     If Board(Target) <> WPAWN Then
8823         If Not CBool(BAttack(Target) And PAttackBit) Then
8824             r = r + 1: If RankNum < Offset Then If RankNum >= Offset - 3 Then r = r +
                1 ' extra bonus if at most three squares behind some friendly pawn
8825         End If
8826     End If
8827 Next
8828
8829 '--- Black space
8830 Offset = PawnsBMin(k): Target = k + 50
8831 For RankNum = 5 To 7 '
8832     Target = Target + 10
8833     If Board(Target) <> BPAWN Then
8834         If Not CBool(WAttack(Target) And PAttackBit) Then
8835             rr = rr + 1: If RankNum <= Offset + 3 And RankNum > Offset Then rr = rr +
                1 ' extra bonus if at most three squares behind some friendly pawn
8836         End If
8837     End If
8838 Next
8839 Next
8840
8841 If r + rr <> 0 Then
8842     ' weight for space
8843     k = 0
8844     For i = 1 To 8 ' count open files
8845         If WPawns(i) = 0 Then If BPawns(i) = 0 Then k = k + 1
8846     Next
8847     If r > 0 Then
8848         a = WNonPawnPieces + 1 + WPawnCnt - 2 * k
8849         WPos.MG = WPos.MG + r * a * a \ 16
8850     End If
8851     If rr > 0 Then
8852         a = BNonPawnPieces + 1 + BPawnCnt - 2 * k
8853         BPos.MG = BPos.MG + rr * a * a \ 16
8854     End If
8855 End If
8856 End If ' <<< Space
8857
8858
8859 '-----
8860 '--- Step 8: Calculate weights and total eval -
8861 '-----
8862 '
8863 '--- evaluate_initiative() /Complexity computes the initiative correction value for the
8864 '--- position, i.e., second order bonus/malus based on the known attacking/defending status of the players.
8865 '--- Semiopenfiles \12 because tricky counting to avoid count duplicate pawns per file
8866 '----- REMOVED: slightly better result without this logic. More complex better because no EGTB?
8867 ' k = 12 * (Abs(WKingFile - BKingFile) - Abs(Rank(WKingLoc) - Rank(BKingLoc))) _
8868 '   + 8 * (Abs(WSemiOpenFiles + BSemiOpenFiles) \ 12 + PassedPawnsCnt) _
8869 '   + 12 * (WPawnCnt + BPawnCnt) _
8870 '   + 16 * Abs(KingSidePawns > 0 And QueenSidePawns > 0) _
8871 '   + 48 * Abs(NonPawnMaterial = 0) _
8872 '   - 136
8873 '
8874 ' rr = MatEval + (WPos.EG - BPos.EG) + (WPassed.EG - BPassed.EG) ' strong side?
8875 ' If rr > 0 Then
8876 '     WPos.EG = WPos.EG + GetMax(k, -Abs(rr))
8877 ' Elseif rr < 0 Then
8878 '     BPos.EG = BPos.EG + GetMax(k, -Abs(rr))
8879 ' End If
8880
8881 '
8882 '--- Material Imbalance / Score trades
8883 '
8884 Dim TradeEval As Long
8885 If MatEval = 0 Then
8886     TradeEval = 0
8887 Else

```

```

8888     TradeEval = Imbalance() 'SF6
8889     AllTotal.MG = AllTotal.MG + TradeEval: AllTotal.EG = AllTotal.EG + TradeEval
8890 End If
8891 AllTotal.MG = AllTotal.MG + ((WPos.MG - BPos.MG) * PiecePosScaleFactor) \ 100&
8892 AllTotal.EG = AllTotal.EG + ((WPos.EG - BPos.EG) * PiecePosScaleFactor) \ 100&
8893 AllTotal.MG = AllTotal.MG + ((WPawnStruct.MG - BPawnStruct.MG) *
PawnStructScaleFactor) \ 100&
8894 AllTotal.EG = AllTotal.EG + ((WPawnStruct.EG - BPawnStruct.EG) *
PawnStructScaleFactor) \ 100&
8895 AllTotal.MG = AllTotal.MG + ((WPassed.MG - BPassed.MG) * PassedPawnsScaleFactor) \
100&
8896 AllTotal.EG = AllTotal.EG + ((WPassed.EG - BPassed.EG) * PassedPawnsScaleFactor) \
100&
8897 AllTotal.MG = AllTotal.MG + ((WMobility.MG - BMobility.MG) * MobilityScaleFactor) \
100&
8898 AllTotal.EG = AllTotal.EG + ((WMobility.EG - BMobility.EG) * MobilityScaleFactor) \
100&
8899 '
8900 'different weights for defending computer king / attacking opp king
8901 If bCompIsWhite Then
8902     WKingScaleFactor = CompKingDefScaleFactor: BKingScaleFactor =
OppKingAttScaleFactor
8903 Else
8904     BKingScaleFactor = CompKingDefScaleFactor: WKingScaleFactor =
OppKingAttScaleFactor
8905 End If
8906 If bWhiteToMove Then
8907     WKingScaleFactor = WKingScaleFactor + 5
8908 Else
8909     BKingScaleFactor = BKingScaleFactor + 5
8910 End If
8911
8912 AllTotal.MG = AllTotal.MG + (WKSafety.MG * WKingScaleFactor) \ 100&
8913 AllTotal.EG = AllTotal.EG + (WKSafety.EG * WKingScaleFactor) \ 100&
8914 AllTotal.MG = AllTotal.MG - (BKSafety.MG * BKingScaleFactor) \ 100&
8915 AllTotal.EG = AllTotal.EG - (BKSafety.EG * BKingScaleFactor) \ 100&
8916 AllTotal.MG = AllTotal.MG + ((WThreat.MG - BThreat.MG) * ThreatsScaleFactor) \ 100&
8917 AllTotal.EG = AllTotal.EG + ((WThreat.EG - BThreat.EG) * ThreatsScaleFactor) \ 100&
8918 '-----
8919 '
8920 '--- Scale Factor ---
8921 '
8922 ScaleFactor = SCALE_FACTOR_NORMAL 'Normal ScaleFactor, scales EG value only
8923 If GamePhase < PHASE_MIDGAME Then
8924     'KRPPKRP endgame
8925     'if the defending king is actively placed and not passed pawn for strong side, the position is drawish
8926     If WNonPawnMaterial = ScoreRook.MG And BNonPawnMaterial = ScoreRook.MG Then
8927         If WPawnCnt = 2 And BPawnCnt = 1 Then 'white is strong side
8928             If WFrontMostPassedPawnRank = 0 Then 'no passed pawn for strong side
8929                 Square = PieceSqList(WPAWN, 1) '1. pawn
8930                 If Rank(BKingLoc) > Rank(Square) Then 'Opp king in front
8931                     If Abs(File(Square) - File(BKingLoc)) <= 1 Then 'and nearby file
8932                         r = Rank(Square): Square = PieceSqList(WPAWN, 2) '2. pawn
8933                     If Rank(BKingLoc) > Rank(Square) Then 'Opp king in front
8934                         If Abs(File(Square) - File(BKingLoc)) <= 1 Then 'and nearby file
8935                             ScaleFactor = KRPPKRP_SFactor(GetMax(r, Rank(Square))): GoTo
8936                             lblEndScaleFactor
8937                         End If
8938                     End If
8939                 End If
8940             End If
8941         End If
8942     ElseIf BPawnCnt = 2 And WPawnCnt = 1 Then 'black is strong side
8943         If BFrontMostPassedPawnRank = 0 Then
8944             Square = PieceSqList(BPAWN, 1) '1. pawn
8945             If RelativeRank(COL_BLACK, WKingLoc) > RelativeRank(COL_BLACK, Square) Then
' Opp king in front

```

```

8946     If Abs(File(Square) - File(WKingLoc)) <= 1 Then ' and nearby file
8947         r = RelativeRank(COL_BLACK, Square)
8948         Square = PieceSqList(BPAWN, 2) ' 2. pawn
8949         If RelativeRank(COL_BLACK, WKingLoc) > RelativeRank(COL_BLACK, Square)
            Then ' Opp king in front
8950             If Abs(File(Square) - File(WKingLoc)) <= 1 Then ' and nearby file
8951                 ScaleFactor = KRPPKRP_SFactor(GetMax(r, RelativeRank(COL_BLACK,
                    Square))): GoTo lblEndScaleFactor
8952             End If
8953         End If
8954     End If
8955 End If
8956 End If
8957 End If
8958 End If
8959
8960 '- zero or just one pawn makes it difficult to win
8961 If AllTotal.EG > 0 Then ' white stronger
8962     If NonPawnMat() = 0 Then If WPawnCnt > BPawnCnt Then ScaleFactor = 96 ' A small
            advantage is typically decisive in a pure pawn endings
8963     Select Case WPawnCnt
8964     Case 0:
8965         If WNonPawnMaterial - BNonPawnMaterial <= ScoreBishop.MG Then
8966             If WNonPawnMaterial < ScoreRook.MG Then
8967                 ScaleFactor = SCALE_FACTOR_DRAW
8968             Else
8969                 If BNonPawnMaterial <= ScoreBishop.MG Then ScaleFactor = 4 Else
                    ScaleFactor = 44
8970             End If
8971         End If
8972     Case 1: If WNonPawnMaterial - BNonPawnMaterial <= ScoreBishop.MG Then
            ScaleFactor = SCALE_FACTOR_ONEPAWN
8973     End Select
8974 ElseIf AllTotal.EG < 0 Then
8975     If NonPawnMat = 0 Then If BPawnCnt > WPawnCnt Then ScaleFactor = 96 ' A small
            advantage is typically decisive in a pure pawn endings
8976     Select Case BPawnCnt
8977     Case 0:
8978         If BNonPawnMaterial - WNonPawnMaterial <= ScoreBishop.MG Then
8979             If BNonPawnMaterial < ScoreRook.MG Then
8980                 ScaleFactor = SCALE_FACTOR_DRAW
8981             Else
8982                 If WNonPawnMaterial <= ScoreBishop.MG Then ScaleFactor = 4 Else
                    ScaleFactor = 44
8983             End If
8984         End If
8985     Case 1: If BNonPawnMaterial - WNonPawnMaterial <= ScoreBishop.MG Then
            ScaleFactor = SCALE_FACTOR_ONEPAWN
8986     End Select
8987 End If
8988
8989 '
8990 '- Endgame with opposite-colored bishops and no other pieces (ignoring pawns)
8991 '- is almost a draw, in case of KBP vs KB, it is even more a draw.
8992 If PieceCnt(WBISHOP) = 1 And PieceCnt(BBISHOP) = 1 And WBishopsOnWhiteSq =
            BBishopsOnBlackSq Then ' opposite-colored bishops
8993     If (WNonPawnMaterial = ScoreBishop.MG Or WNonPawnMaterial = ScoreBishop.MG +
            ScoreQueen.MG) And BNonPawnMaterial = WNonPawnMaterial Then
8994         If WPawnCnt + BPawnCnt > 1 Then ScaleFactor = 31 Else ScaleFactor = 9
8995     Else
8996         ' Endgame with opposite-colored bishops, but also other pieces. Still
8997         ' a bit drawish, but not as drawish as with only the two bishops.
8998         If PieceCnt(WQUEEN) + PieceCnt(BQUEEN) = 0 Then ScaleFactor = 46
8999     End If
9000 Else
9001     If WNonPawnMaterial + BNonPawnMaterial = 0 Then
9002         ' KPsK: K and two or more pawns vs K. There is just a single rule here: If all pawns
9003         ' are on the same rook file and are blocked by the defending king, it's a draw.

```

```

9004     If WPawnCnt >= 2 And BPawnCnt = 0 Then
9005         If File(PieceSqlList(WPAWN, 1)) = 1 Or File(PieceSqlList(WPAWN, 1)) = 8 Then
9006             r = 0
9007
9008         For a = 1 To PieceSqlListCnt(WPAWN)
9009             If File(PieceSqlList(WPAWN, a)) <> File(PieceSqlList(WPAWN, 1)) Then r = 1
9010             : Exit For
9011             If Abs(File(PieceSqlList(WPAWN, a)) - File(BKingLoc)) > 1 Then r = 1:
9012             Exit For
9013             If Rank(PieceSqlList(WPAWN, a)) >= Rank(BKingLoc) Then r = 1: Exit For
9014         Next
9015
9016         If r = 0 Then ScaleFactor = 0 'Draw
9017     End If
9018 ElseIf BPawnCnt >= 2 And WPawnCnt = 0 Then
9019     If File(PieceSqlList(BPAWN, 1)) = 1 Or File(PieceSqlList(BPAWN, 1)) = 8 Then
9020         r = 0
9021
9022         For a = 1 To PieceSqlListCnt(BPAWN)
9023             If File(PieceSqlList(BPAWN, a)) <> File(PieceSqlList(BPAWN, 1)) Then r = 1
9024             : Exit For
9025             If Abs(File(PieceSqlList(BPAWN, a)) - File(WKingLoc)) > 1 Then r = 1:
9026             Exit For
9027             If Rank(PieceSqlList(BPAWN, a)) <= Rank(WKingLoc) Then r = 1: Exit For
9028         Next
9029
9030         If r = 0 Then ScaleFactor = 0 'Draw
9031     End If
9032 End If
9033 End If
9034 End If
9035 End If
9036 End If
9037 End If
9038 End If
9039 End If
9040
9041 lblEndScaleFactor:
9042
9043 '
9044 '--- Added all to eval score (SF based scaling: Eval*100/SFPawnEndGameValue= 100 centipawns =1 pawn)
9045 '--- Example: Eval=240 => 1.00 pawn
9046 Eval = AllTotal.MG * GamePhase + AllTotal.EG * CLng(PHASE_MIDGAME - GamePhase) *
9047 ScaleFactor \ SCALE_FACTOR_NORMAL
9048 Eval = Eval \ PHASE_MIDGAME
9049 ' Initiative: Keep more pawns when attacking
9050 Bonus = (50 * (14 - (WPawnCnt + BPawnCnt))) \ 14
9051 If Eval > 0 Then
9052     Eval = GetMax(Eval - Bonus, Eval \ 2)
9053 ElseIf Eval < 0 Then
9054     Eval = GetMin(Eval + Bonus, Eval \ 2)
9055 End If
9056 lblEndEval:
9057 If bEvalTrace Then
9058     bEvalTrace = False
9059     WriteTrace "---- EVAL TRACE : " & Now()
9060     WriteTrace PrintPos
9061     WriteTrace "Material: " & EvalSFTto100(MatEval)
9062     WriteTrace "Trades : " & EvalSFTto100(TradeEval)
9063     WriteTrace "Position: " & ShowScoreDiff100(WPos, BPos) & " => W" & ShowScore(WPos
9064 ) & ", B" & ShowScore(BPos)
9065     WriteTrace "PawnStru: " & ShowScoreDiff100(WPawnStruct, BPawnStruct) & " => W" &
9066 ShowScore(WPawnStruct) & ", B" & ShowScore(BPawnStruct)

```

```

9065 WriteTrace "PassedPw: " & ShowScoreDiff100(WPassed, BPassed) & " => W" & ShowScore
      (WPassed) & ", B" & ShowScore(BPassed)
9066 WriteTrace "Mobility: " & ShowScoreDiff100(WMobility, BMobility) & " => W(" &
      ShowScore(WMobility) & ", B" & ShowScore(BMobility)
9067 WriteTrace "KSafety : " & ShowScoreDiff100(WKSafety, BKSafety) & " => W(" &
      ShowScore(WKSafety) & ", B" & ShowScore(BKSafety)
9068 WriteTrace "Threats : " & ShowScoreDiff100(WThreat, BThreat) & " => W(" &
      ShowScore(WThreat) & ", B" & ShowScore(BThreat)
9069 WriteTrace "Eval      : " & Eval & " (" & EvalSFTo100(Eval) & "cp)"
9070 WriteTrace "-----"
9071 bTimeExit = True
9072 End If
9073
9074 '-----
9075 '--- Step 9: Invert score for black to move ---
9076 '-----
9077 If Not bWhiteToMove Then Eval = -Eval '--- Invert for black
9078
9079 '-----
9080 '--- Step 10: Add tempo value for side to move ---
9081 '-----
9082 'Eval = Eval + TEMPO_BONUS ' Tempo for side to move
9083 Eval = Eval + (16 + NonPawnMaterial \ ScoreKnight.MG \ 2) ' use dynamic tempo, more during
      opening
9084
9085 ' draw value?
9086 If Eval = DrawContempt Then
9087     Eval = Eval + 1 ' if not a real draw then make a difference
9088 End If
9089
9090 ' If Eval = DrawContempt And bWhiteToMove Then
9091 '     Eval = Eval + 1 ' if not a real draw then make a difference
9092 ' Elseif Eval = -DrawContempt And Not bWhiteToMove Then
9093 '     Eval = Eval - 1 ' if not a real draw for black then make a difference
9094 ' Else '--- other logics: tested, results bad
9095 '     Eval = (Eval \ 16) * 16 ' add granularity
9096 '
9097 '     ' dump eval when shuffling
9098 '     If Fifty > 14 Then ' slows down engine !?!
9099 '         If Abs(Eval) > 5 Then
9100 '             'Eval = (Eval * (230& - Fifty)) \ 200& ' dump eval when shuffling
9101 '             Eval = (Eval * (200& - Fifty)) \ 214& ' SF
9102 '         End If
9103 '     End If
9104 ' End If
9105
9106 '
9107 End Function
9108
9109 '-----
9110 '----- END OF EVAL -----
9111 '-----
9112
9113 Private Function Eval__EndOfEval_DUMMY()
9114     ' for faster navigation in source
9115 End Function
9116
9117 Private Function IsMaterialDraw() As Boolean
9118     '( Protector logic )
9119     IsMaterialDraw = False
9120     If PieceCnt(WPAWN) + PieceCnt(BPAWN) = 0 Then ' no pawns
9121         '--- no heavies Q/R */
9122         If PieceCnt(WROOK) = 0 And PieceCnt(BROOK) = 0 And PieceCnt(WQUEEN) = 0 And
            PieceCnt(BQUEEN) = 0 Then
9123             If PieceCnt(BBISHOP) = 0 And PieceCnt(WBISHOP) = 0 Then
9124                 '--- only knights */
9125                 '--- it pretty safe to say this is a draw */
9126                 If PieceCnt(WKNIGHT) < 3 And PieceCnt(BKNIGHT) < 3 Then IsMaterialDraw = True:

```

```

Exit Function
9127 ElseIf PieceCnt(WKNIGHT) = 0 And PieceCnt(BKNIGHT) = 0 Then
9128 '--- only bishops */
9129 '--- not a draw if one side two other side zero
9130 '--- else its always a draw */
9131 If Abs(PieceCnt(WBISHOP) - PieceCnt(BBISHOP)) < 2 Then IsMaterialDraw = True:
Exit Function
9132 ElseIf (PieceCnt(WKNIGHT) < 3 And PieceCnt(WBISHOP) = 0) Or (PieceCnt(WBISHOP) =
1 And PieceCnt(WKNIGHT) = 0) Then
9133 '--- we cant win, but can black? */
9134 If (PieceCnt(BKNIGHT) < 3 And PieceCnt(BBISHOP) = 0) Or (PieceCnt(BBISHOP) = 1
And PieceCnt(BKNIGHT) = 0) Then IsMaterialDraw = True: Exit Function '--- guess
not */
9135 End If
9136 ElseIf PieceCnt(WQUEEN) = 0 And PieceCnt(BQUEEN) = 0 Then
9137 If PieceCnt(WROOK) = 1 And PieceCnt(BROOK) = 1 Then
9138 '--- rooks equal */
9139 '--- one minor difference max: a draw too usually */
9140 If (PieceCnt(WKNIGHT) + PieceCnt(WBISHOP)) < 2 And (PieceCnt(BKNIGHT) +
PieceCnt(BBISHOP)) < 2 Then IsMaterialDraw = True: Exit Function
9141 ElseIf (PieceCnt(WROOK) = 1 And PieceCnt(BROOK) = 0) Then
9142 '--- one rook */
9143 '--- draw if no minors to support AND minors to defend */
9144 If (PieceCnt(WKNIGHT) + PieceCnt(WBISHOP) = 0) And ((PieceCnt(BKNIGHT) +
PieceCnt(BBISHOP) = 1) Or (PieceCnt(BKNIGHT) + PieceCnt(BBISHOP) = 2)) Then
IsMaterialDraw = True: Exit Function
9145 ElseIf PieceCnt(BROOK) = 1 And PieceCnt(WROOK) = 0 Then
9146 '--- one rook */
9147 '--- draw if no minors to support AND minors to defend */
9148 If (PieceCnt(BKNIGHT) + PieceCnt(BBISHOP) = 0) And ((PieceCnt(WKNIGHT) +
PieceCnt(WBISHOP) = 1) Or (PieceCnt(WKNIGHT) + PieceCnt(WBISHOP) = 2)) Then
IsMaterialDraw = True: Exit Function
9149 End If
9150 End If
9151 End If
9152 End Function
9153
9154 Public Function AdvancedPawnPush(ByVal Piece As Long, ByVal Target As Long) As Boolean
9155 AdvancedPawnPush = False
9156 If Piece = WPAWN Then
9157
9158 Select Case Rank(Target)
9159 Case 7, 8: AdvancedPawnPush = True
9160 Case 6:
9161 '--- if no enemy in front and no enemy pawns left or right
9162 If (Board(Target + SQ_UP) >= NO_PIECE Or (Board(Target + SQ_UP) And 1) = WCOL)
Then If Board(Target + SQ_UP_LEFT) <> BPAWN And Board(Target + SQ_UP_RIGHT)
<> BPAWN Then AdvancedPawnPush = True
9163 End Select
9164
9165 ElseIf Piece = BPAWN Then
9166
9167 Select Case Rank(Target)
9168 Case 1, 2: AdvancedPawnPush = True
9169 Case 3:
9170 If (Board(Target + SQ_DOWN) >= NO_PIECE Or (Board(Target + SQ_DOWN) And 1) =
BCOL) Then If Board(Target + SQ_DOWN_LEFT) <> WPAWN And Board(Target +
SQ_DOWN_RIGHT) <> WPAWN Then AdvancedPawnPush = True
9171 End Select
9172
9173 End If
9174 End Function
9175
9176 Public Function AdvancedPassedPawnPush(ByVal Piece As Long, ByVal Target As Long) As
Boolean
9177 AdvancedPassedPawnPush = False
9178 If Piece = WPAWN Then
9179

```

```

9180     Select Case Rank(Target)
9181         Case 7, 8: AdvancedPassedPawnPush = True
9182         Case 6:
9183             '--- if no enemy pawn in front and no enemy pawns left or right
9184             If Board(Target + SQ_UP) = BPAWN Then Exit Function
9185             If Board(Target + SQ_UP_LEFT) = BPAWN Then Exit Function
9186             If Board(Target + SQ_UP_RIGHT) = BPAWN Then Exit Function
9187             AdvancedPassedPawnPush = True
9188         End Select
9189
9190     ElseIf Piece = BPAWN Then
9191
9192         Select Case Rank(Target)
9193             Case 1, 2: AdvancedPassedPawnPush = True
9194             Case 3:
9195                 '--- if no enemy pawn in front and no enemy pawns left or right
9196                 If Board(Target + SQ_DOWN) = WPAWN Then Exit Function
9197                 If Board(Target + SQ_DOWN_LEFT) = WPAWN Then Exit Function
9198                 If Board(Target + SQ_DOWN_RIGHT) = WPAWN Then Exit Function
9199                 AdvancedPassedPawnPush = True
9200             End Select
9201
9202     End If
9203 End Function
9204
9205 Public Function PieceSquareVal(ByVal Piece As Long, ByVal Square As Long) As Long
9206     '--- Piece value for a square
9207     PieceSquareVal = 0
9208     If bEndgame Then
9209
9210         Select Case Piece
9211             Case NO_PIECE
9212             Case WPAWN
9213                 PieceSquareVal = PsqtWP(Square).EG
9214             Case BPAWN
9215                 PieceSquareVal = PsqtBP(Square).EG
9216             Case WKNIGHT
9217                 PieceSquareVal = PsqtWN(Square).EG
9218             Case BKNIGHT
9219                 PieceSquareVal = PsqtBN(Square).EG
9220             Case WBISHOP
9221                 PieceSquareVal = PsqtWB(Square).EG
9222             Case BBISHOP
9223                 PieceSquareVal = PsqtBB(Square).EG
9224             Case WROOK
9225                 PieceSquareVal = PsqtWR(Square).EG
9226             Case BROOK
9227                 PieceSquareVal = PsqtBR(Square).EG
9228             Case WQUEEN
9229                 PieceSquareVal = PsqtWQ(Square).EG
9230             Case BQUEEN
9231                 PieceSquareVal = PsqtBQ(Square).EG
9232             Case WKING
9233                 PieceSquareVal = PsqtWK(Square).EG
9234             Case BKING
9235                 PieceSquareVal = PsqtBK(Square).EG
9236             End Select
9237
9238     Else
9239
9240         Select Case Piece
9241             Case NO_PIECE
9242             Case WPAWN
9243                 PieceSquareVal = PsqtWP(Square).MG
9244             Case BPAWN
9245                 PieceSquareVal = PsqtBP(Square).MG
9246             Case WKNIGHT
9247                 PieceSquareVal = PsqtWN(Square).MG

```



```

9248     Case BKNIGHT
9249         PieceSquareVal = PsqtBN(Square).MG
9250     Case WBISHOP
9251         PieceSquareVal = PsqtWB(Square).MG
9252     Case BBISHOP
9253         PieceSquareVal = PsqtBB(Square).MG
9254     Case WROOK
9255         PieceSquareVal = PsqtWR(Square).MG
9256     Case BROOK
9257         PieceSquareVal = PsqtBR(Square).MG
9258     Case WQUEEN
9259         PieceSquareVal = PsqtWQ(Square).MG
9260     Case BQUEEN
9261         PieceSquareVal = PsqtBQ(Square).MG
9262     Case WKING
9263         PieceSquareVal = PsqtWK(Square).MG
9264     Case BKING
9265         PieceSquareVal = PsqtBK(Square).MG
9266     End Select
9267
9268     End If
9269 End Function
9270
9271 Public Sub FillPieceSquareVal()
9272     Dim Piece As Long, Target As Long
9273
9274     For Piece = 1 To 16
9275         For Target = SQ_A1 To SQ_H8
9276             bEndgame = False
9277             PsqVal(0, Piece, Target) = PieceSquareVal(Piece, Target)
9278             bEndgame = True
9279             PsqVal(1, Piece, Target) = PieceSquareVal(Piece, Target)
9280         Next
9281     Next
9282
9283 End Sub
9284
9285 Private Function AttackByCol(Col As Long, Square As Long) As Long
9286     If Col = COL_WHITE Then AttackByCol = WAttack(Square) Else AttackByCol = BAttack(
9287         Square)
9288 End Function
9289
9290 Public Sub AddPawnThreat(Score As TScore, _
9291     ByVal HangCol As enumColor, _
9292     ByVal PieceType As enumPieceType, _
9293     ByVal Square As Long)
9294     'SF6: const Score ThreatBySafePawn[PIECE_TYPE_NB] = {
9295     '    S(0, 0), S(0, 0), S(107, 138), S(84, 122), S(114, 203), S(121, 217)
9296     '    const Score ThreatenedByHangingPawn = S(71, 61);
9297     '--- attack by black pawn?
9298     If HangCol = COL_WHITE Then
9299         If Board(Square + SQ_UP_LEFT) = BPAWN Then
9300             If Board(Square + SQ_UP_LEFT + SQ_UP_LEFT) = BPAWN Or Board(Square + SQ_UP_LEFT
9301                 + SQ_UP_RIGHT) = BPAWN Then
9302                 AddScore Score, ThreatBySafePawn(PieceType)
9303             Else
9304                 AddScore Score, ThreatenedByHangingPawn
9305             End If
9306         ElseIf Board(Square + SQ_UP_RIGHT) = BPAWN Then
9307             If Board(Square + SQ_UP_RIGHT + SQ_UP_LEFT) = BPAWN Or Board(Square +
9308                 SQ_UP_RIGHT + SQ_UP_RIGHT) = BPAWN Then
9309                 AddScore Score, ThreatBySafePawn(PieceType)
9310             Else
9311                 AddScore Score, ThreatenedByHangingPawn
9312             End If
9313         Else ' attack by white pawn?
9314             If Board(Square + SQ_DOWN_LEFT) = WPAWN Then

```

```

9313     If Board(Square + SQ_DOWN_LEFT + SQ_DOWN_LEFT) = WPAWN Or Board(Square +
9314         SQ_DOWN_LEFT + SQ_DOWN_RIGHT) = WPAWN Then
9315         AddScore Score, ThreatBySafePawn(PieceType)
9316     Else
9317         AddScore Score, ThreatenedByHangingPawn
9318     End If
9319 ElseIf Board(Square + SQ_DOWN_RIGHT) = WPAWN Then
9320     If Board(Square + SQ_DOWN_RIGHT + SQ_DOWN_LEFT) = WPAWN Or Board(Square +
9321         SQ_DOWN_RIGHT + SQ_DOWN_RIGHT) = WPAWN Then
9322         AddScore Score, ThreatBySafePawn(PieceType)
9323     Else
9324         AddScore Score, ThreatenedByHangingPawn
9325     End If
9326 End If
9327 End If
9328 End Sub
9329
9330 Public Sub AddThreat(ByVal HangCol As enumColor, _
9331     ByVal HangPieceType As enumPieceType, _
9332     ByVal AttackerPieceType As enumPieceType, _
9333     ByVal AttackerSquare As Long, _
9334     ByVal AttackedSquare As Long)
9335     ' Add threat to threat list. calculate score later when full attack array data is available
9336     ThreatCnt = ThreatCnt + 1
9337
9338     With ThreatList(ThreatCnt)
9339         .HangCol = HangCol
9340         .HangPieceType = HangPieceType
9341         .AttackerPieceType = AttackerPieceType
9342         .AttackerSquare = AttackerSquare
9343         .AttackedSquare = AttackedSquare
9344     End With
9345 End Sub
9346
9347 Public Sub CalcThreats()
9348     If ThreatCnt = 0 Then Exit Sub
9349     Dim i As Long, Defended As Boolean, StronglyProtected As Boolean, Weak As Boolean
9350     Dim UsAttackCnt As Long, ThemAttackCnt As Long, RelRank As Long, PawnProtected As
9351     Boolean, Score As TScore
9352
9353     For i = 1 To ThreatCnt
9354         With ThreatList(i)
9355             ' Add a bonus according to the kind of attacking pieces
9356
9357             Score = ZeroScore
9358             If .HangCol = COL_WHITE Then ' view from attacker side = us, attacked = them
9359                 UsAttackCnt = AttackBitCnt(BAttack(.AttackedSquare)): ThemAttackCnt =
9360                 AttackBitCnt(WAttack(.AttackedSquare))
9361                 PawnProtected = CBool(WAttack(.AttackedSquare) And PAttackBit): RelRank = Rank
9362                 (.AttackedSquare)
9363             Else ' Black
9364                 UsAttackCnt = AttackBitCnt(WAttack(.AttackedSquare)): ThemAttackCnt =
9365                 AttackBitCnt(BAttack(.AttackedSquare))
9366                 PawnProtected = CBool(BAttack(.AttackedSquare) And PAttackBit): RelRank = 9 -
9367                 Rank(.AttackedSquare)
9368             End If
9369             ' StronglyProtected: by pawn or by more defenders then attackers
9370             StronglyProtected = PawnProtected Or (ThemAttackCnt > UsAttackCnt)
9371             ' Non-pawn enemies strongly defended
9372             Defended = .HangPieceType <> PT_PAWN And StronglyProtected
9373             ' Enemies not strongly defended and under our attack
9374             Weak = Not StronglyProtected
9375             If Defended Or Weak Then
9376                 If .AttackerPieceType = PT_BISHOP Or .AttackerPieceType = PT_KNIGHT Then

```

```

9374         AddScore Score, ThreatByMinor(.HangPieceType)
9375         If .HangPieceType <> PT_PAWN Then
9376             AddScoreVal Score, ThreatByRank.MG * RelRank, ThreatByRank.EG * RelRank
9377         End If
9378     End If
9379     If Weak Then If ThemAttackCnt = 0 Then AddScore Score, Hanging 'hanging
9380     If .HangPieceType <> PT_PAWN Then 'Overload: attacked and defended only once
9381         If ThemAttackCnt = 1 Then AddScore Score, Overload
9382     End If
9383 End If
9384 If (.HangPieceType = PT_QUEEN Or Weak) And .AttackerPieceType = PT_ROOK Then
9385     AddScore Score, ThreatByRook(.HangPieceType)
9386     If .HangPieceType <> PT_PAWN Then
9387         AddScoreVal Score, ThreatByRank.MG * RelRank, ThreatByRank.EG * RelRank
9388     End If
9389 End If
9390 If Score.MG <> 0 Or Score.EG <> 0 Then If .HangCol = COL_WHITE Then AddScore
    BThreat, Score Else AddScore WThreat, Score
9391 End With
9392
9393 lblNext:
9394     Next
9395
9396 End Sub
9397
9398 Public Sub AddWKingAttackers(ByVal AttackBit As Long)
9399     If AttackBit And PLAttackBit Then AddWKingAttack PT_PAWN
9400     If AttackBit And PRAttackBit Then AddWKingAttack PT_PAWN
9401     If AttackBit And N1AttackBit Then AddWKingAttack PT_KNIGHT
9402     If AttackBit And N2AttackBit Then AddWKingAttack PT_KNIGHT
9403     If AttackBit And B1AttackBit Then AddWKingAttack PT_BISHOP
9404     If AttackBit And B2AttackBit Then AddWKingAttack PT_BISHOP
9405     If AttackBit And BXrayAttackBit Then If Not (AttackBit And (B1AttackBit Or
    B2AttackBit)) Then WKingAttackersCount = WKingAttackersCount + 1
9406     If AttackBit And (R1AttackBit Or R1XrayAttackBit) Then AddWKingAttack PT_ROOK
9407     If AttackBit And (R2AttackBit Or R2XrayAttackBit) Then AddWKingAttack PT_ROOK
9408     If AttackBit And (QAttackBit Or QXrayAttackBit) Then AddWKingAttack PT_QUEEN
9409 End Sub
9410
9411 Public Sub AddBKingAttackers(ByVal AttackBit As Long)
9412     If AttackBit And PLAttackBit Then AddBKingAttack PT_PAWN
9413     If AttackBit And PRAttackBit Then AddBKingAttack PT_PAWN
9414     If AttackBit And N1AttackBit Then AddBKingAttack PT_KNIGHT
9415     If AttackBit And N2AttackBit Then AddBKingAttack PT_KNIGHT
9416     If AttackBit And B1AttackBit Then AddBKingAttack PT_BISHOP
9417     If AttackBit And B2AttackBit Then AddBKingAttack PT_BISHOP
9418     If AttackBit And BXrayAttackBit Then If Not (AttackBit And (B1AttackBit Or
    B2AttackBit)) Then BKingAttackersCount = BKingAttackersCount + 1
9419     If AttackBit And (R1AttackBit Or R1XrayAttackBit) Then AddBKingAttack PT_ROOK
9420     If AttackBit And (R2AttackBit Or R2XrayAttackBit) Then AddBKingAttack PT_ROOK
9421     If AttackBit And (QAttackBit Or QXrayAttackBit) Then AddBKingAttack PT_QUEEN
9422 End Sub
9423
9424 Public Sub AddWKingAttack(PT As enumPieceType)
9425     WKingAttackersCount = WKingAttackersCount + 1
9426     WKingAttackersWeight = WKingAttackersWeight + KingAttackWeights(PT)
9427 End Sub
9428
9429 Public Sub AddBKingAttack(PT As enumPieceType)
9430     BKingAttackersCount = BKingAttackersCount + 1
9431     BKingAttackersWeight = BKingAttackersWeight + KingAttackWeights(PT)
9432 End Sub
9433
9434 Public Function InitConnectedPawns()
9435     'SF6
9436     Dim Seed(8) As Long, Opposed As Long, Phalanx As Long, Support As Long, r As Long, v
    As Long, x As Long
9437     ReadLngArr Seed(), 0, 0, 13, 24, 18, 76, 100, 175, 330

```

```

9438
9439 For Opposed = 0 To 1
9440 For Phalanx = 0 To 1
9441 For Support = 0 To 2
9442 For r = 2 To 7
9443 If Phalanx > 0 Then x = (Seed(r + 1) - Seed(r)) / 2 Else x = 0
9444 v = 17 * Support
9445 v = v + Seed(r)
9446 If Phalanx > 0 Then v = v + (Seed(r + 1) - Seed(r)) \ 2
9447 If Opposed > 0 Then v = v / 2 '>> operator for opposed in VB: /2
9448 ConnectedBonus(Opposed, Phalanx, Support, r).MG = v
9449 ConnectedBonus(Opposed, Phalanx, Support, r).EG = v * ((r - 1) - 2) \ 4 'rank
    r ist zero based in C, so (r-1)
9450 Next
9451 Next
9452 Next
9453 Next
9454
9455 End Function
9456
9457 Public Sub InitImbalance() 'SF6
9458 '// pair pawn knight bishop rook queen OUR PIECES
9459 ReadIntArr2 QuadraticOurs(), 0, 1667 'Bishop pair
9460 ReadIntArr2 QuadraticOurs(), PT_PAWN, 40, 0 'Pawn
9461 ReadIntArr2 QuadraticOurs(), PT_KNIGHT, 32, 255, -3 'Knight
9462 ReadIntArr2 QuadraticOurs(), PT_BISHOP, 0, 104, 4, 0 'Bishop
9463 ReadIntArr2 QuadraticOurs(), PT_ROOK, -26, -2, 47, 105, -149 'Rook
9464 ReadIntArr2 QuadraticOurs(), PT_QUEEN, -185, 24, 122, 137, -134, 0 'Queen
9465 '// pair pawn knight bishop rook queen THEIR PIECES
9466 ReadIntArr2 QuadraticTheirs(), 0, 0 'Bishop pair
9467 ReadIntArr2 QuadraticTheirs(), PT_PAWN, 36, 0 'Pawn
9468 ReadIntArr2 QuadraticTheirs(), PT_KNIGHT, 9, 63, 0 'Knight
9469 ReadIntArr2 QuadraticTheirs(), PT_BISHOP, 59, 65, 42, 0 'Bishop
9470 ReadIntArr2 QuadraticTheirs(), PT_ROOK, 46, 39, 24, -24, 0 'Rook
9471 ReadIntArr2 QuadraticTheirs(), PT_QUEEN, 101, 100, -37, 141, 268, 0 'Queen
9472 '// PawnSet[pawn count] contains a bonus/malus indexed by number of pawns
9473 ReadIntArr PawnSet(), 24, -32, 107, -51, 117, -9, -126, -21, 31
9474 End Sub
9475
9476 Public Function Imbalance() As Long 'SF
9477 Dim v As Long, Key As Long
9478 Key = CalcMaterialKey()
9479 Imbalance = ProbeMaterialHash(Key)
9480 If Imbalance <> VALUE_NONE Then Exit Function
9481 ImbPieceCount(COL_WHITE, 0) = Abs(PieceCnt(WBISHOP) > 1) 'index 0 used for bishop pair
9482 ImbPieceCount(COL_BLACK, 0) = Abs(PieceCnt(BBISHOP) > 1) 'index 0 used for bishop pair
9483 ImbPieceCount(COL_WHITE, PT_PAWN) = PieceCnt(WPAWN)
9484 ImbPieceCount(COL_BLACK, PT_PAWN) = PieceCnt(BPAWN)
9485 ImbPieceCount(COL_WHITE, PT_KNIGHT) = PieceCnt(WKNIGHT)
9486 ImbPieceCount(COL_BLACK, PT_KNIGHT) = PieceCnt(BKNIGHT)
9487 ImbPieceCount(COL_WHITE, PT_BISHOP) = PieceCnt(WBISHOP)
9488 ImbPieceCount(COL_BLACK, PT_BISHOP) = PieceCnt(BBISHOP)
9489 ImbPieceCount(COL_WHITE, PT_ROOK) = PieceCnt(WROOK)
9490 ImbPieceCount(COL_BLACK, PT_ROOK) = PieceCnt(BROOK)
9491 ImbPieceCount(COL_WHITE, PT_QUEEN) = PieceCnt(WQUEEN)
9492 ImbPieceCount(COL_BLACK, PT_QUEEN) = PieceCnt(BQUEEN)
9493 v = (ColImbalance(COL_WHITE) - ColImbalance(COL_BLACK)) \ 16
9494 'If Imbalance <> VALUE_NONE And Imbalance <> v Then MsgBox "Diff"
9495 Imbalance = v
9496 SaveMaterialHash Key, Imbalance
9497 End Function
9498
9499 Public Function ColImbalance(ByVal Col As enumColor) As Long
9500 Dim Bonus As Long, pt1 As Long, pt2 As Long, Us As Long, Them As Long, v As Long
9501 If Col = COL_WHITE Then
9502 Us = COL_WHITE: Them = COL_BLACK: Bonus = PawnSet(PieceCnt(WPAWN))
9503 If PieceCnt(WQUEEN) = 1 Then If PieceCnt(BQUEEN) = 0 Then Bonus = Bonus +
    QueenMinorsImbalance(PieceCnt(BKNIGHT) + PieceCnt(BBISHOP))

```

```

9504 Else
9505     Us = COL_BLACK: Them = COL_WHITE: Bonus = PawnSet(PieceCnt(BPAWN))
9506     If PieceCnt(BQUEEN) = 1 Then If PieceCnt(WQUEEN) = 0 Then Bonus = Bonus +
        QueenMinorsImbalance(PieceCnt(WKNIGHT) + PieceCnt(WBISHOP))
9507 End If
9508
9509 For pt1 = 0 To PT_QUEEN
9510     If ImbPieceCount(Us, pt1) > 0 Then
9511         v = 0
9512
9513         For pt2 = 0 To pt1
9514             v = v + QuadraticOurs(pt1, pt2) * ImbPieceCount(Us, pt2) + QuadraticTheirs(pt1
                , pt2) * ImbPieceCount(Them, pt2)
9515         Next pt2
9516
9517         Bonus = Bonus + ImbPieceCount(Us, pt1) * v
9518     End If
9519 Next pt1
9520
9521 ColImbalance = Bonus
9522 End Function
9523
9524 Public Sub AddScore(ScoreTotal As TScore, ScoreAdd As TScore)
9525     ScoreTotal.MG = ScoreTotal.MG + ScoreAdd.MG: ScoreTotal.EG = ScoreTotal.EG +
        ScoreAdd.EG
9526 End Sub
9527
9528 Public Sub AddScoreWithFactor(ScoreTotal As TScore, ScoreAdd As TScore, Factor As Long
)
9529     ScoreTotal.MG = ScoreTotal.MG + ScoreAdd.MG * Factor: ScoreTotal.EG = ScoreTotal.EG
        + ScoreAdd.EG * Factor
9530 End Sub
9531
9532 'Public Sub AddScore100(ScoreTotal As TScore, ScoreAdd As TScore)
9533 ' ' Score 100 centipawns based: scale to SF pawn value
9534 ' ScoreTotal.MG = ScoreTotal.MG + (ScoreAdd.MG * ScorePawn.EG) \ 100&: ScoreTotal.EG = ScoreTotal.EG +
        (ScoreAdd.EG * ScorePawn.EG) \ 100&
9535 'End Sub
9536
9537 Public Sub AddScoreVal(ScoreTotal As TScore, ByVal MGScore As Long, ByVal EGScore As
Long)
9538     ScoreTotal.MG = ScoreTotal.MG + MGScore: ScoreTotal.EG = ScoreTotal.EG + EGScore
9539 End Sub
9540
9541 Public Sub SetScoreVal(ScoreSet As TScore, ByVal MGScore As Long, ByVal EGScore As
Long)
9542     ScoreSet.MG = MGScore: ScoreSet.EG = EGScore
9543 End Sub
9544
9545 Public Function EvalSFTto100(ByVal Eval As Long) As Long
9546     If Abs(Eval) < MATE_IN_MAX_PLY Then EvalSFTto100 = (Eval * 100&) / CLng(ScorePawn.EG)
        Else EvalSFTto100 = Eval
9547 End Function
9548
9549 Public Function Eval100ToSF(ByVal Eval As Long) As Long
9550     Eval100ToSF = (Eval * CLng(ScorePawn.EG)) / 100&
9551 End Function
9552
9553 Public Sub MinusScore(ScoreTotal As TScore, ScoreMinus As TScore)
9554     ScoreTotal.MG = ScoreTotal.MG - ScoreMinus.MG: ScoreTotal.EG = ScoreTotal.EG -
        ScoreMinus.EG
9555 End Sub
9556
9557 Public Function ScaleScore(Score As TScore) As Long
9558     ' Calculate score for game phase
9559     ScaleScore = Score.MG * GamePhase + Score.EG * CLng(PHASE_MIDGAME - GamePhase) ' *
        SF6 / SCALE_FACTOR_NORMAL
9560     ScaleScore = ScaleScore \ PHASE_MIDGAME

```

```

9561 End Function
9562
9563 'Public Function ScaleScore100(Score As TScore, ByVal ScaleVal As Long) As TScore
9564 ' ScaleScore100.MG = (Score.MG * ScaleVal) \ 100&: ScaleScore100.EG = (Score.EG * ScaleVal) \ 100&
9565 'End Function
9566
9567 Public Function ShowScore(Score As TScore) As String
9568 ' show MG, EG Score as text
9569 ShowScore = "(" & CStr(Score.MG) & ", " & CStr(Score.EG) & ")=" & ScaleScore(Score)
9570 End Function
9571
9572 Public Function ShowScoreDiff100(Score1 As TScore, Score2 As TScore) As String
9573 ' show MG, EG Score as text
9574 Dim Diff As TScore
9575 Diff.MG = Score1.MG - Score2.MG: Diff.EG = Score1.EG - Score2.EG
9576 ShowScoreDiff100 = ShowScore(Diff)
9577 End Function
9578
9579 Public Function PieceSQ(ByVal Side As enumColor, _
9580                         ByVal SearchPieceType As enumPieceType) As Long
9581 Dim a As Long, p As Long
9582
9583 For a = 1 To NumPieces
9584 p = Board(Pieces(a)): If PieceType(p) = SearchPieceType And PieceColor(p) = Side
9585 Then PieceSQ = Pieces(a): Exit Function
9586
9587 Next
9588
9589 End Function
9590
9591 Public Function Eval_KRKP() As Long
9592 Dim WKSq As Long, BKSq As Long, RookSq As Long, PawnSq As Long, StrongSide
9593 As enumColor, WeakSide As enumColor
9594 Dim StrongKingLoc As Long, WeakKingLoc As Long, QueeningSq As Long, Result As Long,
9595 SideToMove As enumColor
9596 If WMaterial > BMaterial Then
9597 StrongSide = COL_WHITE: WeakSide = COL_BLACK: StrongKingLoc = WKingLoc:
9598 WeakKingLoc = BKingLoc
9599 Else
9600 StrongSide = COL_BLACK: WeakSide = COL_WHITE: StrongKingLoc = BKingLoc:
9601 WeakKingLoc = WKingLoc
9602 End If
9603 If bWhiteToMove Then SideToMove = COL_WHITE Else SideToMove = COL_BLACK
9604 WKSq = RelativeSq(StrongSide, StrongKingLoc)
9605 BKSq = RelativeSq(StrongSide, WeakKingLoc)
9606 RookSq = RelativeSq(StrongSide, PieceSQ(StrongSide, PT_ROOK))
9607 PawnSq = RelativeSq(WeakSide, PieceSQ(WeakSide, PT_PAWN))
9608 QueeningSq = SQ_A1 + File(PawnSq) - 1 + 7 * SQ_UP
9609 '-- If the stronger side's king is in front of the pawn, it's a win
9610 If WKSq < PawnSq And File(WKSq) = File(PawnSq) Then
9611 Result = ScoreRook.EG - MaxDistance(WKSq, PawnSq)
9612 '-- If the weaker side's king is too far from the pawn and the rook, it's a win.
9613 ElseIf MaxDistance(BKSq, PawnSq) >= (3 + Abs(SideToMove = WeakSide)) And MaxDistance
9614 (BKSq, RookSq) >= 3 Then
9615 Result = ScoreRook.EG - MaxDistance(WKSq, PawnSq)
9616 '-- If the pawn is far advanced and supported by the defending king, the position is drawish
9617 ElseIf Rank(BKSq) <= 3 And MaxDistance(BKSq, PawnSq) = 1 And Rank(WKSq) >= 4 And
9618 MaxDistance(WKSq, PawnSq) > (2 + Abs(SideToMove = StrongSide)) Then
9619 Result = 80 - 8 * MaxDistance(WKSq, PawnSq)
9620 Else
9621 Result = 200 - 8 * (MaxDistance(WKSq, PawnSq + SQ_DOWN) - MaxDistance(BKSq, PawnSq
9622 + SQ_DOWN) - MaxDistance(PawnSq, QueeningSq))
9623 End If
9624 If StrongSide = SideToMove Then Eval_KRKP = Result Else Eval_KRKP = -Result
9625 If Not bWhiteToMove Then Eval_KRKP = -Eval_KRKP
9626 End Function
9627
9628 Public Function Eval_KQKP() As Long
9629 ' KQ vs KP. In general, this is a win for the stronger side, but there are a

```



```

9621 'few important exceptions. A pawn on 7th rank and on the A,C,F or H files
9622 'with a king positioned next to it can be a draw, so in that case, we only
9623 'use the distance between the kings.
9624 Dim WinnerKSq As Long, LoserKSq As Long, PawnSq As Long, StrongSide As enumColor,
WeakSide As enumColor
9625 Dim Result As Long, SideToMove As enumColor
9626 If WMaterial > BMaterial Then
9627     StrongSide = COL_WHITE: WeakSide = COL_BLACK: WinnerKSq = WKingLoc: LoserKSq =
BKingLoc
9628 Else
9629     StrongSide = COL_BLACK: WeakSide = COL_WHITE: WinnerKSq = BKingLoc: LoserKSq =
WKingLoc
9630 End If
9631 PawnSq = PieceSQ(WeakSide, PT_PAWN)
9632 If bWhiteToMove Then SideToMove = COL_WHITE Else SideToMove = COL_BLACK
9633 Result = PushClose(MaxDistance(WinnerKSq, LoserKSq))
9634 If RelativeRank(WeakSide, PawnSq) <> 7 Or MaxDistance(LoserKSq, PawnSq) <> 1 Then
9635     Result = Result + ScoreQueen.EG - ScorePawn.EG
9636 Else
9637
9638     Select Case File(PawnSq) ' For File A,C,F,H
9639         Case 2, 4, 5, 7: Result = Result + ScoreQueen.EG - ScorePawn.EG
9640     End Select
9641
9642 End If
9643 If StrongSide = SideToMove Then Eval_KQKP = Result Else Eval_KQKP = -Result
9644 If Not bWhiteToMove Then Eval_KQKP = -Eval_KQKP
9645 End Function
9646
9647 Public Function Eval_KQKR() As Long
9648     Dim WinnerKSq As Long, LoserKSq As Long, StrongSide As enumColor, WeakSide As
enumColor
9649     Dim Result As Long, SideToMove As enumColor
9650     If WMaterial > BMaterial Then
9651         StrongSide = COL_WHITE: WeakSide = COL_BLACK: WinnerKSq = WKingLoc: LoserKSq =
BKingLoc
9652     Else
9653         StrongSide = COL_BLACK: WeakSide = COL_WHITE: WinnerKSq = BKingLoc: LoserKSq =
WKingLoc
9654     End If
9655     If bWhiteToMove Then SideToMove = COL_WHITE Else SideToMove = COL_BLACK
9656     Result = ScoreQueen.EG - ScoreRook.EG + PushToEdges(LoserKSq) + PushClose(
MaxDistance(WinnerKSq, LoserKSq))
9657     If StrongSide = SideToMove Then Eval_KQKR = Result Else Eval_KQKR = -Result
9658     If Not bWhiteToMove Then Eval_KQKR = -Eval_KQKR
9659 End Function
9660
9661 Private Function WKingShelterStorm(ShelterKingLoc As Long) As Long
9662     Dim Center As Long, k As Long, r As Long, RelFile As Long, Safety As Long, RankUs As
Long, RankThem As Long, RankNum As Long
9663     Safety = 258
9664     ' Opp pawn rank A/H protects king
9665     If File(WKingLoc) = 1 Or File(WKingLoc) = 8 Then
9666         If Rank(WKingLoc) <= 2 Then If Board(WKingLoc + SQ_UP) = BPAWN Then Safety = 350
9667     End If
9668
9669     '--- Pawn shelter
9670     Center = GetMax(2, GetMin(7, File(ShelterKingLoc))): RankNum = Rank(ShelterKingLoc)
9671     ' File A=>B, File H=>G
9672
9673     For k = Center - 1 To Center + 1
9674         ' Pawn shelter/storm
9675         RankUs = 1
9676         If WPawns(k) > 0 Then If PawnsWMin(k) >= RankNum Then RankUs = PawnsWMin(k)
9677         RankThem = 1
9678         If BPawns(k) > 0 Then If PawnsBMin(k) >= RankNum Then RankThem = PawnsBMin(k)
9679         If RankThem = RankNum + 1 And k = File(ShelterKingLoc) Then
9680             r = 1 ' BlockedByKing

```



```

9680     ElseIf RankUs = 1 Then
9681         r = 2 'NoFriendlyPawn
9682     ElseIf RankThem = RankUs + 1 Then
9683         r = 3 'BlockedByPawn
9684     Else
9685         r = 4 'Unblocked
9686     End If
9687     RelFile = GetMin(k, 9 - k)
9688     Safety = Safety - (ShelterWeakness(RelFile, RankUs) + StormDanger(r, RelFile,
9689                                     RankThem))
9689 Next
9690
9691 If Center >= 6 Then
9692     If Board(SQ_H3) = BPAWN Then
9693         If Board(SQ_H2) = WPAWN Then If Board(SQ_G3) = WPAWN Then If Board(SQ_F2) =
9694             WPAWN Then Safety = Safety + 300
9695     End If
9696     If Board(SQ_F3) = BPAWN Then
9697         If Board(SQ_H2) = WPAWN Then If Board(SQ_G3) = WPAWN Then If Board(SQ_F2) =
9698             WPAWN Then Safety = Safety + 300
9699     End If
9700     ElseIf Center <= 3 Then
9701         If Board(SQ_A3) = BPAWN Then
9702             If Board(SQ_A2) = WPAWN Then If Board(SQ_B3) = WPAWN Then If Board(SQ_C2) =
9703                 WPAWN Then Safety = Safety + 300
9704         End If
9705         If Board(SQ_C3) = BPAWN Then
9706             If Board(SQ_A2) = WPAWN Then If Board(SQ_B3) = WPAWN Then If Board(SQ_C2) =
9707                 WPAWN Then Safety = Safety + 300
9708         End If
9709     End If
9710     WKingShelterStorm = Safety
9711 End Function
9712
9713 Private Function BKingShelterStorm(ByVal ShelterKingLoc As Long) As Long
9714     Dim Center As Long, k As Long, r As Long, RelFile As Long, Safety As Long, RankUs As
9715         Long, RankThem As Long, RankNum As Long
9716     Safety = 258
9717     'Opp pawn rank A/H protects king
9718     If File(BKingLoc) = 1 Or File(BKingLoc) = 8 Then
9719         If Rank(BKingLoc) >= 7 Then If Board(BKingLoc + SQ_DOWN) = WPAWN Then Safety = 350
9720     End If '--- Pawn shelter
9721     Center = GetMax(2, GetMin(7, File(ShelterKingLoc))): RankNum = 9 - Rank(
9722         ShelterKingLoc) 'File A=>B, File H=>G
9723
9724 For k = Center - 1 To Center + 1
9725     'Pawn shelter/storm
9726     RankUs = 1
9727     If BPawns(k) > 0 Then If 9 - PawnsBMax(k) >= RankNum Then RankUs = (9 - PawnsBMax(
9728         k))
9729     RankThem = 1
9730     If WPawns(k) > 0 Then If 9 - PawnsWMax(k) >= RankNum Then RankThem = (9 -
9731         PawnsWMax(k))
9732     If RankThem = RankNum + 1 And k = File(ShelterKingLoc) Then
9733         r = 1 'BlockedByKing
9734     ElseIf RankUs = 1 Then
9735         r = 2 'NoFriendlyPawn
9736     ElseIf RankThem = RankUs + 1 Then
9737         r = 3 'BlockedByPawn
9738     Else
9739         r = 4 'Unblocked
9740     End If
9741     RelFile = GetMin(k, 9 - k)
9742     Safety = Safety - (ShelterWeakness(RelFile, RankUs) + StormDanger(r, RelFile,
9743         RankThem))
9744 Next
9745 If Center >= 6 Then

```

```

9738     If Board(SQ_H6) = WPAWN Then
9739         If Board(SQ_H7) = BPAWN Then If Board(SQ_G6) = BPAWN Then If Board(SQ_F7) =
            BPAWN Then Safety = Safety + 250
9740     End If
9741     If Board(SQ_F6) = WPAWN Then
9742         If Board(SQ_H7) = BPAWN Then If Board(SQ_G6) = BPAWN Then If Board(SQ_F7) =
            BPAWN Then Safety = Safety + 150
9743     End If
9744     ElseIf Center <= 3 Then
9745         If Board(SQ_A6) = WPAWN Then
9746             If Board(SQ_A7) = BPAWN Then If Board(SQ_B6) = BPAWN Then If Board(SQ_C7) =
                BPAWN Then Safety = Safety + 250
9747         End If
9748         If Board(SQ_C6) = WPAWN Then
9749             If Board(SQ_A7) = BPAWN Then If Board(SQ_B6) = BPAWN Then If Board(SQ_C7) =
                BPAWN Then Safety = Safety + 150
9750         End If
9751     End If
9752
9753     BKingShelterStorm = Safety
9754 End Function
9755
9756 Private Sub GetKingFlankFiles(ByVal KingLoc As Long, FileFrom As Long, FileTo As Long)
9757
9758     Select Case File(KingLoc)
9759         Case 1 To 3: FileFrom = FILE_A: FileTo = FILE_D ' File A-C> A-D
9760         Case 4 To 5: FileFrom = FILE_C: FileTo = FILE_F ' File D-E> C-F
9761         Case 6 To 8: FileFrom = FILE_E: FileTo = FILE_H ' File F-H> E-H
9762     End Select
9763
9764 End Sub
9765
9766 'Public Function PinnedPieceDir(ByVal PinnedLoc As Long, ByVal MoveTarget As Long, PieceCol As enumColor) As
Long
9767 ' ' check if a piece is pinned to king and returns direction offset from piece to king, if not pinned = 0
9768 ' PinnedPieceDir = 0
9769 ' If PieceCol = COL_WHITE Then
9770 '     PinnedPieceDir = WPinnedPieceDir(PinnedLoc)
9771 '     If PinnedPieceDir <> 0 Then
9772 '         If SameXRay(MoveTarget, WKingLoc) Then PinnedPieceDir = 0 ' move in pinned direction Ok
9773 '     End If
9774 ' ElseIf PieceCol = COL_BLACK Then
9775 '     PinnedPieceDir = BPinnedPieceDir(PinnedLoc)
9776 '     If PinnedPieceDir <> 0 Then
9777 '         If SameXRay(MoveTarget, BKingLoc) Then PinnedPieceDir = 0 ' move in pinned direction Ok
9778 '     End If
9779 ' End If
9780 'If PinnedPieceDir <> 0 Then Stop
9781 'End Function
9782
9783 Public Function WPinnedPieceDir(ByVal PinnedLoc As Long) As Long
9784 ' -- check if a piece is pinned to king and returns direction offset from piece to king, if not pinned = 0
9785 Dim k As Long, sq As Long, Offset As Long, Piece As Long
9786 WPinnedPieceDir = 0
9787 If PinnedLoc = WKingLoc Then Exit Function
9788 Offset = DirOffset(PinnedLoc, WKingLoc) ' Find direction to king
9789 If Offset = 0 Then Exit Function
9790
9791 ' no other piece between piece and own king?
9792 sq = PinnedLoc
9793 For k = 1 To 7
9794     sq = sq + Offset: If sq = WKingLoc Then Exit For ' pinned possible
9795     Piece = Board(sq) ' If Piece = FRAME Then Exit For ' should not happen
9796     If Piece < NO_PIECE Then Exit Function ' other piece found > not pinned
9797 Next k
9798
9799 ' check other direction for attacker
9800 sq = PinnedLoc

```

```

9801 For k = 1 To 7
9802     sq = sq - Offset
9803     Piece = Board(sq): If Piece = FRAME Then Exit For
9804     If Piece < NO_PIECE Then
9805         Select Case Piece
9806             Case BQUEEN:
9807                 WPinnedPieceDir = Offset: Exit Function ' pinned by queen
9808             Case BROOK:
9809                 If Abs(Offset) = 10 Or Abs(Offset) = 1 Then WPinnedPieceDir = Offset: Exit
9810                     Function ' pinned by rook
9811             Case BBISHOP:
9812                 If Abs(Offset) = 9 Or Abs(Offset) = 11 Then WPinnedPieceDir = Offset: Exit
9813                     Function ' pinned by bishop
9814             End Select
9815             Exit Function ' other piece found
9816         End If
9817     Next k
9818     ' not pinned here
9819 End Function
9820
9821 Public Function BPinnedPieceDir(ByVal PinnedLoc As Long) As Long
9822     '-- check if a piece is pinned to king and returns direction offset from piece to king, if not pinned = 0
9823     Dim k As Long, sq As Long, Offset As Long, Piece As Long
9824     BPinnedPieceDir = 0
9825     If PinnedLoc = BKingLoc Then Exit Function
9826     Offset = DirOffset(PinnedLoc, BKingLoc) ' Find direction to king
9827     If Offset = 0 Then Exit Function
9828     ' no other piece between piece and own king?
9829     sq = PinnedLoc
9830     For k = 1 To 7
9831         sq = sq + Offset: If sq = BKingLoc Then Exit For ' pinned possible
9832         Piece = Board(sq) ' If Piece = FRAME Then Exit For ' should not happen
9833         If Piece < NO_PIECE Then Exit Function ' other piece found > not pinned
9834     Next k
9835     ' check other direction for attacker
9836     sq = PinnedLoc
9837     For k = 1 To 7
9838         sq = sq - Offset
9839         Piece = Board(sq): If Piece = FRAME Then Exit For
9840         If Piece < NO_PIECE Then
9841             Select Case Piece
9842                 Case WQUEEN:
9843                     BPinnedPieceDir = Offset: Exit Function ' pinned by queen
9844                 Case WROOK:
9845                     If Abs(Offset) = 10 Or Abs(Offset) = 1 Then BPinnedPieceDir = Offset: Exit
9846                         Function ' pinned by rook
9847                 Case WBISHOP:
9848                     If Abs(Offset) = 9 Or Abs(Offset) = 11 Then BPinnedPieceDir = Offset: Exit
9849                         Function ' pinned by bishop
9850             End Select
9851             Exit Function ' other piece found
9852         End If
9853     Next k
9854     ' not pinned here
9855 End Function
9856
9857 'Public Function PinnedPieceW(ByVal PinnedLoc As Long, ByVal Direction As Long) As Boolean
9858 ' ' white pieces it threatend by pinned pieces and slider attack?
9859 ' Dim k As Long, sq As Long, Offset As Long, AttackBit As Long, Piece As Long
9860 ' PinnedPieceW = False
9861 ' If Direction < 4 Then ' Queen or rook orthogonal
9862 '     If Not CBool(BAttack(PinnedLoc) And QRAAttackBit) Then Exit Function
9863 '     AttackBit = QRAAttackBit
9864 ' Else ' Queen or bishop diagonal
9865 '     If Not CBool(BAttack(PinnedLoc) And QBAttackBit) Then Exit Function
9866 '     AttackBit = QBAttackBit

```

```

9865 ' End If
9866 ' Offset = DirectionOffset(Direction)
9867 '
9868 ' For k = 1 To 8
9869 '   sq = PinnedLoc + Offset * k: Piece = Board(sq)
9870 '   If Piece = FRAME Then Exit For
9871 '   If Piece < NO_PIECE Then
9872 '     If Piece = BQUEEN Then PinnedPieceW = True: Exit Function
9873 '     If Piece = BROOK Then If Direction < 4 Then PinnedPieceW = True: Exit Function
9874 '     If Piece = BBISHOP Then If Direction >= 4 Then PinnedPieceW = True: Exit Function
9875 '     Exit For
9876 '   Else
9877 '     If Not (CBool(BAttack(sq) And AttackBit)) Then Exit For
9878 '   End If
9879 ' Next k
9880 '
9881 'End Function
9882 '
9883 'Public Function PinnedPieceB(ByVal PinnedLoc As Long, ByVal Direction As Long) As Boolean
9884 ' ' black pieces it threatend by pinned pieces and slider attack?
9885 ' Dim k As Long, sq As Long, Offset As Long, AttackBit As Long, Piece As Long
9886 ' PinnedPieceB = False
9887 ' If Direction < 4 Then ' Queen or rook orthogonal
9888 '   If Not CBool(WAttack(PinnedLoc) And QRAttackBit) Then Exit Function
9889 '   AttackBit = QRAttackBit
9890 ' Else ' Queen or bishop diagonal
9891 '   If Not CBool(WAttack(PinnedLoc) And QBAttackBit) Then Exit Function
9892 '   AttackBit = QBAttackBit
9893 ' End If
9894 ' Offset = DirectionOffset(Direction)
9895 '
9896 ' For k = 1 To 8
9897 '   sq = PinnedLoc + Offset * k: Piece = Board(sq)
9898 '   If Piece = FRAME Then Exit For
9899 '   If Piece < NO_PIECE Then
9900 '     If Piece = WQUEEN Then PinnedPieceB = True: Exit Function
9901 '     If Piece = WROOK Then If Direction < 4 Then PinnedPieceB = True: Exit Function
9902 '     If Piece = WBISHOP Then If Direction >= 4 Then PinnedPieceB = True: Exit Function
9903 '     Exit For
9904 '   Else
9905 '     If Not (CBool(WAttack(sq) And AttackBit)) Then Exit For
9906 '   End If
9907 ' Next k
9908 '
9909 'End Function
9910
9911 Public Sub InitOutpostSq()
9912   Dim sq As Long
9913
9914   For sq = SQ_A1 To SQ_H8
9915     If Rank(sq) >= 4 And Rank(sq) <= 6 Then WOutpostSq(sq) = True
9916     If Rank(sq) >= 3 And Rank(sq) <= 5 Then BOutpostSq(sq) = True
9917   Next sq
9918
9919 End Sub
9920
9921 'Public Function NonPawnMatForSide(ByVal UseColOfSideToMove As Boolean) As Long
9922 ' If UseColOfSideToMove Then
9923 '   If bWhiteToMove Then
9924 '     NonPawnMatForSide = PieceCnt(WQUEEN) * ScoreQueen.MG + PieceCnt(WROOK) * ScoreRook.MG +
9925 '     PieceCnt(WBISHOP) * ScoreBishop.MG + PieceCnt(WKNIGHT) * ScoreKnight.MG
9926 '   Else
9927 '     NonPawnMatForSide = PieceCnt(BQUEEN) * ScoreQueen.MG + PieceCnt(BROOK) * ScoreRook.MG +
9928 '     PieceCnt(BBISHOP) * ScoreBishop.MG + PieceCnt(BKNIGHT) * ScoreKnight.MG
9929 '   End If
9930 ' Else
9931 '   If Not bWhiteToMove Then
9932 '     NonPawnMatForSide = PieceCnt(WQUEEN) * ScoreQueen.MG + PieceCnt(WROOK) * ScoreRook.MG +

```

```

9931 PieceCnt(WBISHOP) * ScoreBishop.MG + PieceCnt(WKNIGHT) * ScoreKnight.MG
9932 ' Else
9933 '   NonPawnMatForSide = PieceCnt(BQUEEN) * ScoreQueen.MG + PieceCnt(BROOK) * ScoreRook.MG +
9934 PieceCnt(BBISHOP) * ScoreBishop.MG + PieceCnt(BKNIGHT) * ScoreKnight.MG
9935 ' End If
9936 ' End If
9937 'End Function
9938
9939 Public Function NonPawnMat() As Long
9940     NonPawnMat = (PieceCnt(WQUEEN) + PieceCnt(BQUEEN)) * ScoreQueen.MG + (PieceCnt(WROOK)
9941         + PieceCnt(BROOK)) * ScoreRook.MG + (PieceCnt(WBISHOP) + PieceCnt(BBISHOP)) *
9942         ScoreBishop.MG + (PieceCnt(WKNIGHT) + PieceCnt(BKNIGHT)) * ScoreKnight.MG
9943 End Function
9944
9945 'Public Function MaterialTotal() As Long
9946 ' ' from view of white
9947 ' MaterialTotal = (PieceCnt(WQUEEN) - PieceCnt(BQUEEN)) * ScoreQueen.MG + (PieceCnt(WROOK) -
9948 PieceCnt(BROOK)) * ScoreRook.MG + (PieceCnt(WBISHOP) - PieceCnt(BBISHOP)) * ScoreBishop.MG +
9949 (PieceCnt(WKNIGHT) - PieceCnt(BKNIGHT)) * ScoreKnight.MG + (PieceCnt(WPAWN) - PieceCnt(BPAWN)) *
9950 ScorePawn.MG
9951 'End Function
9952
9953 'Public Function PositionalDiff(ByVal Score As Long) As Long
9954 ' ' from view of white: absolute difference material piece values / Total score = Positional score
9955 ' If Score = VALUE_NONE Then PositionalDiff = 0: Exit Function
9956 ' Dim MatScore As Long
9957 ' MatScore = (PieceCnt(WQUEEN) - PieceCnt(BQUEEN)) * ScoreQueen.MG + (PieceCnt(WROOK) -
9958 PieceCnt(BROOK)) * ScoreRook.MG + (PieceCnt(WBISHOP) - PieceCnt(BBISHOP)) * ScoreBishop.MG +
9959 (PieceCnt(WKNIGHT) - PieceCnt(BKNIGHT)) * ScoreKnight.MG + (PieceCnt(WPAWN) - PieceCnt(BPAWN)) *
9960 ScorePawn.MG
9961 ' PositionalDiff = Abs(Score - MatScore)
9962 'End Function
9963
9964 Public Sub CheckWQueenWeek(ByVal sq As Long, ByVal Offset As Long, ByVal Direction As
9965 Long, ByRef Result As Boolean)
9966     ' Queen pinned or discovered threat possible
9967     If Result Then Exit Sub ' count only once
9968     Dim r As Long
9969     Result = False: r = sq + Offset ' next sq in same direction
9970     Select Case Board(r)
9971     Case BROOK: If Direction < 4 Then Result = True
9972     Case BBISHOP: If Direction > 3 Then Result = True
9973     Case NO_PIECE:
9974         If Direction < 4 Then ' Rook
9975             ' 2nd part: compare both attackbits, may be from different rooks: R1Attackbit or R2Attackbit
9976             If CBool(BAttack(sq) And RAttackBit) Then If (BAttack(r) And RAttackBit) = (
9977                 BAttack(sq) And RAttackBit) Then Result = True
9978         Else ' Bishop?
9979             If CBool(BAttack(sq) And BAttackBit) Then If (BAttack(r) And BAttackBit) = (
9980                 BAttack(sq) And BAttackBit) Then Result = True
9981         End If
9982     End Select
9983 End Sub
9984
9985 Public Sub CheckBQueenWeek(ByVal sq As Long, ByVal Offset As Long, ByVal Direction As
9986 Long, ByRef Result As Boolean)
9987     ' Queen pinned or discovered threat possible
9988     If Result Then Exit Sub ' count only once
9989     Dim r As Long
9990     Result = False: r = sq + Offset ' next sq in same direction
9991     Select Case Board(r)
9992     Case WROOK: If Direction < 4 Then Result = True
9993     Case WBISHOP: If Direction > 3 Then Result = True
9994     Case NO_PIECE:
9995         If Direction < 4 Then ' Rook
9996             ' 2nd part: compare both attackbits, may be from different rooks: R1Attackbit or R2Attackbit
9997             If CBool(WAttack(sq) And RAttackBit) Then If (WAttack(r) And RAttackBit) = (
9998                 WAttack(sq) And RAttackBit) Then Result = True

```

```

9984     Else ' Bishop?
9985         If CBool(WAttack(sq) And BAttackBit) Then If (WAttack(r) And BAttackBit) = (
           WAttack(sq) And BAttackBit) Then Result = True
9986     End If
9987 End Select
9988 End Sub
9989
9990 'Public Function DistToOppKing(Piece As Integer, Target As Integer) As Long
9991 ' If PieceColor(Piece) = COL_WHITE Then
9992 '     DistToOppKing = MaxDistance(BKingLoc, Target)
9993 ' Else
9994 '     DistToOppKing = MaxDistance(WKingLoc, Target)
9995 ' End If
9996 'End Function
9997
9998 'Public Function CalcSimpleEval() As Long
9999 ' Dim WNonPawnMaterial As Long, BNonPawnMaterial As Long
10000 ' WNonPawnMaterial = PieceCnt(WQUEEN) * ScoreQueen.MG + PieceCnt(WROOK) * ScoreRook.MG +
PieceCnt(WBISHOP) * ScoreBishop.MG + PieceCnt(WKNIGHT) * ScoreKnight.MG
10001 ' BNonPawnMaterial = PieceCnt(BQUEEN) * ScoreQueen.MG + PieceCnt(BROOK) * ScoreRook.MG +
PieceCnt(BBISHOP) * ScoreBishop.MG + PieceCnt(BKNIGHT) * ScoreKnight.MG
10002 '
10003 ' CalcSimpleEval = ScorePawn.EG * (PieceCnt(WPAWN) - PieceCnt(BPAWN)) + (WNonPawnMaterial -
BNonPawnMaterial)
10004 ' If Not bWhiteToMove Then CalcSimpleEval = -CalcSimpleEval
10005 'End Function
10006 VERSION 5.00
10007 Begin {C62A69F0-16DC-11CE-9E98-00AA00574A4F} frmChessX
10008     Caption           = "ChessBrainVBA 3.03"
10009     ClientHeight       = 10500
10010     ClientLeft        = 45
10011     ClientTop         = 375
10012     ClientWidth       = 15915
10013     OleObjectBlob     = "frmChessX.frx":0000
10014     ShowModal         = 0 'False
10015     StartUpPosition   = 3 'Windows-Standard
10016 End
10017 Attribute VB_Name = "frmChessX"
10018 Attribute VB_GlobalNameSpace = False
10019 Attribute VB_Creatable = False
10020 Attribute VB_PredeclaredId = True
10021 Attribute VB_Exposed = False
10022 '=====
10023 '= VBACHessBrainX, a chess playing winboard engine by Roger Zuehlisdorf (Copyright 2015)
10024 '= and is based on LarsenVb by Luca Dormio(http://xoomer.virgilio.it/ludormio/download.htm)
10025 '=
10026 '= VBACHessBrainX is free software: you can redistribute it and/or modify
10027 '= it under the terms of the GNU General Public License as published by
10028 '= the Free Software Foundation, either version 3 of the License, or
10029 '= (at your option) any later version.
10030 '=
10031 '= VBACHessBrainX is distributed in the hope that it will be useful,
10032 '= but WITHOUT ANY WARRANTY; without even the implied warranty of
10033 '= MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
10034 '= GNU General Public License for more details.
10035 '=
10036 '= You should have received a copy of the GNU General Public License
10037 '= along with this program. If not, see <http://www.gnu.org/licenses/>.
10038 '=====
10039
10040 Option Explicit
10041
10042
10043 ' GUI controls
10044 Dim oField(1 To 64) As Control
10045 Dim oFieldEvents(1 To 64) As clsBoardField
10046 Dim oLabelsX(1 To 8) As Control
10047 Dim oLabelsX2(1 To 8) As Control

```

```

10048 Dim oLabelsY(1 To 8) As Control
10049 Dim oLabelsY2(1 To 8) As Control
10050 Dim oPiecePics(1 To 12) As Control
10051 Dim oPieceCnt(1 To 6) As Control
10052
10053 Dim i As Long
10054
10055
10056
10057
10058
10059
10060 Private Sub chkFlipBoard_Change()
10061     If chkFlipBoard.Value = True Then
10062         FlipBoard False
10063     Else
10064         FlipBoard True
10065     End If
10066 End Sub
10067
10068 Private Sub chkShowThinking_Change()
10069     txtIO.Visible = chkShowThinking
10070 End Sub
10071
10072
10073 Private Sub chkTableBases_Click()
10074     If chkTableBases.Value = True Then
10075         TableBasesRootEnabled = True
10076         WriteINISetting "TB_ROOT_ENABLED", "1"
10077         TableBasesSearchEnabled = True
10078         WriteINISetting "TB_SEARCH_ENABLED", "1"
10079         optSecondsPerMove.Value = 1
10080         cboSecondsPerMove.Value = "30" ' Min 20 sec fix for EGTB Init needed
10081     Else
10082         TableBasesRootEnabled = False
10083         WriteINISetting "TB_ROOT_ENABLED", "0"
10084         TableBasesSearchEnabled = False
10085         WriteINISetting "TB_SEARCH_ENABLED", "0"
10086     End If
10087 End Sub
10088
10089 Private Sub cmdClearBoard_Click()
10090     Dim i As Integer
10091     For i = SQ_A1 To SQ_H8
10092         If Board(i) <> FRAME Then Board(i) = NO_PIECE
10093     Next
10094     ShowBoard
10095 End Sub
10096
10097 Private Sub cmdClearCommand_Click()
10098     cboFakeInput = ""
10099 End Sub
10100
10101
10102 Private Sub SelectPiece(PieceType As Integer)
10103     Dim i As Integer
10104     For i = 1 To 12: Me.Controls("Piece" & CStr(i)).SpecialEffect = 0: Next
10105     SetupPiece = PieceType
10106     Me.Controls("Piece" & CStr(PieceType)).SpecialEffect = 3
10107 End Sub
10108
10109 Private Sub cmdEndSetup_Click()
10110     Dim i As Integer, WKCnt As Integer, BKCnt As Integer, bPosLegal As Boolean
10111
10112     ' Is position legal?
10113     bPosLegal = True: WKCnt = 0: BKCnt = 0
10114     For i = SQ_A1 To SQ_H8
10115         Select Case Board(i)

```



```

10116     Case WKING: WKCnt = WKCnt + 1: If WKCnt > 1 Then bPosLegal = False: MsgBox
10117     Translate("Illegal position: only one White King allowed!")
10117     Case BKING: BKCnt = BKCnt + 1: If BKCnt > 1 Then bPosLegal = False: MsgBox
10118     Translate("Illegal position: only one Black King allowed!")
10118     Case WPAWN, BPAWN: If Rank(i) = 1 Or Rank(i) = 8 Then bPosLegal = False:: MsgBox
10119     Translate("Illegal position: Pawn rank must between 2 and 7!")
10119     End Select
10120 Next
10121 If WKCnt = 0 Then bPosLegal = False: MsgBox Translate("Illegal position: White
10122 King needed!")
10122 If BKCnt = 0 Then bPosLegal = False: MsgBox Translate("Illegal position: Black
10123 King needed!")
10124 If Not bPosLegal Then Exit Sub
10125
10125 SetupBoardMode = False
10126 cmdClearBoard.Visible = False
10127 cmdEndSetup.Visible = False
10128 chkWOO.Visible = False
10129 chkWOOO.Visible = False
10130 chkBOO.Visible = False
10131 chkBOOO.Visible = False
10132 lblSelectPiece.Visible = False
10133 cmdSetup.Visible = True
10134
10135 ' Init data
10136 Erase arFiftyMove()
10137 Fifty = 0
10138 Erase Moved()
10139
10140 OpeningHistory = " "
10141 BookPly = BOOK_MAX_PLY + 1 ' no book
10142
10143 ' Castling
10144 WhiteCastled = NO_CASTLE
10145 BlackCastled = NO_CASTLE
10146 If Not chkWOO.Value Then Moved(SQ_H1) = 1 ' Rook moved flag
10147 If Not chkWOOO.Value Then Moved(SQ_A1) = 1 ' Rook moved flag
10148 If Not chkBOO.Value Then Moved(SQ_H8) = 1 ' Rook moved flag
10149 If Not chkBOOO.Value Then Moved(SQ_A8) = 1 ' Rook moved flag
10150
10151 InitPieceSquares
10152 GameMovesCnt = 0
10153 HintMove = EmptyMove
10154 GamePosHash(GameMovesCnt) = HashBoard(EmptyMove) ' for 3x repetition draw
10155 ShowMoveList
10156 ShowBoard
10157 psLastFieldClick = "": psFieldFrom = "": psFieldTarget = "": plFieldFrom = 0:
10158 plFieldTarget = 0
10159 End Sub
10160
10160 Private Sub cmdHint_Click()
10161     If HintMove.From > 0 Then
10162         If Board(HintMove.From) <> NO_PIECE Then
10163             SendCommand ">" & Translate("Hint") & ": " & MoveText(HintMove)
10164             ResetGUIFieldColors
10165             ShowMove HintMove.From, HintMove.Target
10166             DoEvents
10167             Sleep 2000
10168             ResetGUIFieldColors
10169         End If
10170     End If
10171 End Sub
10172
10173 Private Sub cmdSetup_Click()
10174     If cmdStop.Visible Then Exit Sub ' Thinking
10175     SetupBoardMode = True
10176     cmdClearBoard.Visible = True
10177     cmdEndSetup.Visible = True

```

```

10178     lblSelectPiece.Visible = True
10179     chkWOO.Visible = True: chkWOO = False
10180     chkWOOO.Visible = True: chkWOOO = False
10181     chkBOO.Visible = True: chkBOO = False
10182     chkBOOO.Visible = True: chkBOOO = False
10183     cmdSetup.Visible = False
10184     txtIO = Translate("Select piece and click at square")
10185 End Sub
10186
10187 Private Sub cmdSwitchSideToMove_Click()
10188     If cmdStop.Visible = True Then Exit Sub
10189     bWhiteToMove = Not bWhiteToMove
10190     ShowColToMove
10191 End Sub
10192
10193
10194
10195 Private Sub cmdTestPos1_Click()
10196     ' Read from INI or use default
10197     optSecondsPerMove.Value = 1
10198     If cboSecondsPerMove.Value < 8 Then cboSecondsPerMove.Value = "8"
10199     cboFakeInput.Text = "setboard " & ReadINISetting("TEST_POSITION1",
10200         "1b5k/7P/p1p2np1/2P2p2/PP3P2/4RQ1R/q2r3P/6K1 w - - bm Re8+; id WAC.250;Mate in 8;")
10201     cmdFakeInput_Click
10202 End Sub
10203
10204 Private Sub cmdTestPos2_Click()
10205     optSecondsPerMove.Value = 1
10206     If cboSecondsPerMove.Value < 10 Then cboSecondsPerMove.Value = "10"
10207     cboFakeInput.Text = "setboard " & ReadINISetting("TEST_POSITION2",
10208         "2k4B/bpp1qp2/p1b5/7p/1PN1n1p1/2Pr4/P5PP/R3QR1K b - - bm Ng3+; id WAC.273;")
10209     cmdFakeInput_Click
10210 End Sub
10211
10212 Private Sub cmdTestPos3_Click()
10213     optSecondsPerMove.Value = 1
10214     If cboSecondsPerMove.Value < 10 Then cboSecondsPerMove.Value = "10"
10215     cboFakeInput.Text = "setboard " & ReadINISetting("TEST_POSITION3",
10216         "r3q2r/2p1k1p1/p5p1/1p2Nb2/1P2nB2/P7/2PNQbPP/R2R3K b - - bm Rxh2+; id WAC.266;")
10217     'cboFakeInput.Text = "setboard 8/5P2/8/4K3/2k5/8/8 w - - " ' Promote test
10218     cmdFakeInput_Click
10219 End Sub
10220
10221 Private Sub cmdTestPos4_Click()
10222     cboFakeInput.Text = "setboard " & ReadINISetting("TEST_POSITION4",
10223         "8/6k1/6p1/8/7r/3P1KP1/8/8 w - - 0 1 ; Tablebase test;")
10224     optSecondsPerMove.Value = 1
10225     cboSecondsPerMove.Value = "30" ' Min 20 sec fix EGTB Init needed
10226     cmdFakeInput_Click
10227 End Sub
10228
10229 Private Sub cmdWriteFEN_Click()
10230     Dim s As String, r As String
10231     s = WriteEPD()
10232     r = InputBox(Translate("please copy"), Translate("EPD position string"), s)
10233 End Sub
10234
10235
10236 Private Sub cmdZoomMinus_Click()
10237     If Me.Zoom > 30 Then
10238         Me.Zoom = Me.Zoom - 5
10239         Me.Width = Me.Width * 0.95
10240         Me.Height = Me.Height * 0.95
10241     End If
10242 End Sub
10243
10244 Private Sub cmdZoomPlus_Click()
10245     Me.Zoom = Me.Zoom + 5
10246     Me.Width = Me.Width * 1.05

```

```

10242     Me.Height = Me.Height * 1.05
10243 End Sub
10244
10245
10246
10247
10248
10249
10250
10251 Private Sub imgLangDE_Click()
10252     ' Translate to german
10253     WriteINISetting "LANGUAGE", "DE"
10254     InitTranslate
10255     TranslateForm
10256     ShowBoard
10257 End Sub
10258
10259 Private Sub imgLangEN_Click()
10260     ' Translate to english
10261     WriteINISetting "LANGUAGE", "EN"
10262     InitTranslate
10263     TranslateForm
10264     ShowBoard
10265     MsgBox "Please restart for english"
10266 End Sub
10267
10268 Private Sub UserForm_Initialize()
10269     ' GUI Start: Init
10270     ' Application.Workbooks.Parent.Visible = False ' Don't show EXCEL
10271     SetVBAPathes
10272     ReadColors
10273     CreateBoard
10274     LoadPiecesPics
10275     InitTimes
10276     InitTestSets
10277
10278     InitEngine
10279     InitGame
10280     TranslateForm
10281     ShowBoard
10282     chkTableBases.Value = TableBasesRootEnabled
10283     Me.Show
10284 End Sub
10285
10286 Public Sub cmdThink_Click()
10287     '
10288     '--- Start thinking for computer move
10289     '
10290     Static bThinking As Boolean
10291     If bThinking Or SetupBoardMode Then Exit Sub
10292     bThinking = True
10293     txtIO = ""
10294
10295     SetTimeControl
10296
10297     bPostMode = True
10298     bForceMode = False
10299     Result = NO_MATE
10300
10301     If bWhiteToMove And optBlack = False Then optBlack = True
10302
10303     If bWhiteToMove And optBlack = True Then
10304         optWhite = True
10305         SendToEngine "white"
10306     ElseIf Not bWhiteToMove And optWhite = True Then
10307         optBlack = True
10308         SendToEngine "black"
10309     End If

```

```

10310     If optWhite Then bCompIsWhite = True Else bCompIsWhite = False
10311
10312     DoEvents
10313     cmdThink.Caption = Translate("Thinking") & "...
10314     cmdThink.Enabled = False
10315     cmdStop.Visible = True
10316     DoEvents
10317
10318     SendToEngine "go"
10319
10320     If optWhite Then bCompIsWhite = True Else bCompIsWhite = False
10321
10322     '--- Start chess engine -----
10323     StartEngine
10324     '--- End thinking
10325
10326     '--- Human to move
10327     cmdThink.Caption = Translate("Think") & " !"
10328     cmdThink.Enabled = True
10329     cmdStop.Visible = False
10330
10331     bThinking = False
10332     ShowBoard
10333     ShowLastMoveAtBoard
10334     ShowMoveList
10335     Me.Show
10336 End Sub
10337
10338
10339
10340 Private Sub cmdFakeInput_Click()
10341     '--- parse command input
10342     FakeInput = cboFakeInput.Text & vbLf
10343     FakeInputState = True
10344     cboFakeInput.SelStart = 0
10345     cboFakeInput.SelLength = Len(cboFakeInput.Text)
10346     cboFakeInput.SetFocus
10347     SetupBoardMode = False
10348
10349     If InStr(FakeInput, "setboard") > 0 Then
10350         InitGame
10351         txtMoveList = ""
10352         Erase arGameMoves()
10353         GameMovesCnt = 0
10354         Result = NO_MATE
10355     End If
10356
10357     ParseCommand FakeInput
10358     ShowBoard
10359
10360     If bWhiteToMove Then
10361         optWhite.Value = True
10362     Else
10363         optBlack.Value = True
10364     End If
10365     ShowColToMove
10366     psLastFieldClick = "": plFieldFrom = 0: plFieldTarget = 0
10367 End Sub
10368
10369 Public Sub ShowBoard()
10370     Dim x As Long, y As Long, Pos As Long, piece As Long
10371
10372     For x = 1 To 8
10373         For y = 1 To 8
10374             Pos = x + (y - 1) * 8
10375             piece = Board(SQ_A1 + x - 1 + (y - 1) * 10)
10376             If piece = NO_PIECE Then
10377                 Set oField(Pos).Picture = Nothing

```

```

10378         ElseIf piece >= 1 And piece <= 12 Then
10379             Set oField(Pos).Picture = oPiecePics(piece).Picture
10380         End If
10381     Next
10382 Next
10383 ResetGUIFieldColors
10384
10385 ' Show piece counts for white; call Eval to get counts
10386 InitEval
10387 x = Eval()
10388 oPieceCnt(PieceDisplayOrder(WPAWN) + 1).Caption = CStr(PieceCnt(WPAWN) - PieceCnt(
BPawn))
10389 oPieceCnt(PieceDisplayOrder(WKNIGHT) + 1).Caption = CStr(PieceCnt(WKNIGHT) -
PieceCnt(BKNIGHT))
10390 oPieceCnt(PieceDisplayOrder(WBISHOP) + 1).Caption = CStr(PieceCnt(WBISHOP) -
PieceCnt(BBISHOP))
10391 oPieceCnt(PieceDisplayOrder(WROOK) + 1).Caption = CStr(PieceCnt(WROOK) - PieceCnt(
BROOK))
10392 oPieceCnt(PieceDisplayOrder(WQUEEN) + 1).Caption = CStr(PieceCnt(WQUEEN) - PieceCnt(
BQUEEN))
10393
10394 ' instead of king count show total sum
10395 oPieceCnt(PieceDisplayOrder(WKING) + 1).Caption = CStr(PieceCnt(WPAWN) - PieceCnt(
BPawn) + (PieceCnt(WKNIGHT) - PieceCnt(BKNIGHT)) * 3 + _
10396                                     (PieceCnt(WBISHOP) - PieceCnt(
BBISHOP)) * 3 + (PieceCnt(WROOK) -
PieceCnt(BROOK)) * 5 + (PieceCnt(
WQUEEN) - PieceCnt(BQUEEN)) * 9)
10397
10398 Me.Repaint
10399 ShowColToMove
10400 End Sub
10401
10402 Private Sub CreateBoard()
10403     '--- Create Square Images and Labels
10404     Dim lFieldWidth As Long, lFrameWidth As Long
10405     Dim x As Long, y As Long, i As Long, bBackColorIsWhite As Boolean
10406
10407     bBackColorIsWhite = False
10408     lFieldWidth = Me.fraBoard.Width \ 9 ' 8 + 1xFrame
10409     lFrameWidth = lFieldWidth / 2
10410
10411     For y = 1 To 8
10412         '--- Label board with A - H
10413         Set oLabelsX(y) = Me.fraBoard.Controls.Add("Forms.Label.1", "LabelX")
10414         With oLabelsX(y)
10415             .Width = lFieldWidth: .Height = lFrameWidth: .FontSize = 12: .TextAlign = 2: .
Font.Bold = True
10416             .Left = lFrameWidth + (y - 1) * lFieldWidth: .Top = 8 * lFieldWidth + lFrameWidth
10417             .BackStyle = 0: .ForeColor = WhiteSqCol: .Caption = Chr$(Asc("A") - 1 + y): .
BackColor = WhiteSqCol
10418         End With
10419
10420         Set oLabelsX2(y) = Me.fraBoard.Controls.Add("Forms.Label.1", "LabelX2")
10421         With oLabelsX2(y)
10422             .Width = lFieldWidth: .Height = lFrameWidth: .FontSize = 12: .TextAlign = 2: .
Font.Bold = True
10423             .Left = lFrameWidth + (y - 1) * lFieldWidth: .Top = 2 ' 1 * lFieldWidth
10424             .BackStyle = 0: .ForeColor = WhiteSqCol: .Caption = Chr$(Asc("A") - 1 + y): .
BackColor = WhiteSqCol
10425         End With
10426
10427
10428         '--- Label board with 1 - 8
10429         Set oLabelsY(y) = Me.fraBoard.Controls.Add("Forms.Label.1", "LabelY")
10430         With oLabelsY(y)
10431             .Width = lFrameWidth: .Height = lFieldWidth: .FontSize = 12: .TextAlign = 2: .
Font.Bold = True

```

```

10432     .Left = 0: .Top = (8 - y) * lFieldWidth + lFrameWidth + lFieldWidth \ 3
10433     .BackStyle = 0: .ForeColor = WhiteSqCol: .Caption = CStr(y): .BackColor =
WhiteSqCol
10434 End With
10435
10436 Set oLabelsY2(y) = Me.fraBoard.Controls.Add("Forms.Label.1", "LabelY2")
10437 With oLabelsY2(y)
10438     .Width = lFrameWidth: .Height = lFieldWidth: .FontSize = 12: .TextAlign = 2: .
Font.Bold = True
10439     .Left = lFrameWidth + (9 - 1) * lFieldWidth: .Top = (8 - y) * lFieldWidth +
lFrameWidth + lFieldWidth \ 3
10440     .BackStyle = 0: .ForeColor = WhiteSqCol: .Caption = CStr(y): .BackColor =
WhiteSqCol
10441 End With
10442
10443
10444 '--- set square images
10445 For x = 1 To 8
10446     i = x + (y - 1) * 8
10447     Set oField(i) = Me.fraBoard.Controls.Add("Forms.Image.1", "Square" & i)
10448
10449     Set oFieldEvents(i) = New clsBoardField: oFieldEvents(i).SetBoardField oField(i) '
To catch click events
10450     oFieldEvents(i).Name = "Square" & i
10451
10452     With oField(i)
10453         .Width = lFieldWidth: .Height = lFieldWidth: .PictureSizeMode =
fmPictureSizeModeZoom
10454         .Left = lFrameWidth + (x - 1) * lFieldWidth: .Top = lFrameWidth + (8 - y) *
lFieldWidth
10455         .Tag = 20 + x + (y - 1) * 10 '--- Engine field number
10456         If bBackColorIsWhite Then .BackColor = WhiteSqCol Else .BackColor = BlackSqCol
10457         bBackColorIsWhite = Not bBackColorIsWhite
10458     End With
10459 Next x
10460 bBackColorIsWhite = Not bBackColorIsWhite
10461 Next y
10462 End Sub
10463
10464 Private Sub FlipBoard(bWhiteAtBottom As Boolean)
10465     '--- Create Square Images and Labels
10466     Dim lFieldWidth As Long, lFrameWidth As Long
10467     Dim x As Long, y As Long, i As Long
10468
10469     lFieldWidth = Me.fraBoard.Width \ 9 '8 + 1xFrame
10470     lFrameWidth = lFieldWidth / 2
10471
10472 For y = 1 To 8
10473     '--- Label board with A - H
10474     With oLabelsX(y)
10475         If bWhiteAtBottom Then
10476             .Left = lFrameWidth + (y - 1) * lFieldWidth
10477         Else
10478             .Left = 8 * lFieldWidth - (lFrameWidth + (y - 1) * lFieldWidth)
10479         End If
10480     End With
10481
10482     With oLabelsX2(y)
10483         If bWhiteAtBottom Then
10484             .Left = lFrameWidth + (y - 1) * lFieldWidth
10485         Else
10486             .Left = 8 * lFieldWidth - (lFrameWidth + (y - 1) * lFieldWidth)
10487         End If
10488     End With
10489
10490 '--- Label board with 1 - 8
10491 With oLabelsY(y)
10492     If bWhiteAtBottom Then

```

```

10493         .Top = (8 - y) * lFieldWidth + lFrameWidth + lFieldWidth \ 3
10494     Else
10495         .Top = (y - 1) * lFieldWidth + lFrameWidth + lFieldWidth \ 3
10496     End If
10497 End With
10498
10499 With oLabelsY2(y)
10500     If bWhiteAtBottom Then
10501         .Top = (8 - y) * lFieldWidth + lFrameWidth + lFieldWidth \ 3
10502     Else
10503         .Top = (y - 1) * lFieldWidth + lFrameWidth + lFieldWidth \ 3
10504     End If
10505 End With
10506
10507 '--- set square images
10508 For x = 1 To 8
10509     i = x + (y - 1) * 8
10510     With oField(i)
10511         If bWhiteAtBottom Then
10512             .Left = lFrameWidth + (x - 1) * lFieldWidth: .Top = lFrameWidth + (8 - y) *
lFieldWidth
10513         Else
10514             .Left = 8 * lFieldWidth - (lFrameWidth + (x - 1) * lFieldWidth): .Top = 8 *
lFieldWidth - (lFrameWidth + (8 - y) * lFieldWidth)
10515         End If
10516     End With
10517 Next x
10518 Next y
10519 End Sub
10520
10521
10522 Private Sub LoadPiecesPics()
10523 Dim PicExtension As String
10524 Dim sFile As String
10525 Dim i As Long, lFieldWidth As Long
10526
10527 PicExtension = "cur"
10528
10529 sFile = Dir(psDocumentPath & "\WhitePawn.*") '--- Get image extension
10530 If Trim(sFile) <> "" Then PicExtension = Right(sFile, 3) "cur"
10531
10532 lFieldWidth = Me.fraPieces.Width \ 6
10533
10534 ' Init piece count fields
10535 For i = 1 To 6
10536     Set oPieceCnt(i) = Me.fraPieceCnt.Controls.Add("Forms.Label.1", "PieceCnt")
10537     With oPieceCnt(i)
10538         .Width = lFieldWidth: .Height = lFieldWidth \ 2: .FontSize = 10: .TextAlign = 2: .
Font.Bold = True
10539         .Left = (i - 1) * (lFieldWidth - 2): .Top = 0
10540         .BackStyle = 0: .ForeColor = &H80000012: .Caption = " "
10541     End With
10542 Next i
10543
10544 '--- Init piece pictures
10545 For i = 1 To 12
10546     Set oPiecePics(i) = Me.Controls("Piece" & CStr(i)) ' Preloaded images
10547
10548 ' Load piece images dynamical
10549 If False Then
10550     Set oPiecePics(i) = Me.fraPieces.Controls.Add("Forms.Image.1", "Pieces")
10551
10552     Select Case i
10553     Case 1
10554         Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\WhitePawn." &
PicExtension)
10555     Case 2
10556         Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\BlackPawn." &

```



```

        PicExtension)
10557 Case 3
10558     Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\\WhiteKnight." &
        PicExtension)
10559 Case 4
10560     Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\\BlackKnight." &
        PicExtension)
10561 Case 5
10562     Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\\WhiteKing." &
        PicExtension)
10563 Case 6
10564     Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\\BlackKing." &
        PicExtension)
10565 Case 7
10566     Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\\WhiteRook." &
        PicExtension)
10567 Case 8
10568     Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\\BlackRook." &
        PicExtension)
10569 Case 9
10570     Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\\WhiteQueen." &
        PicExtension)
10571 Case 10
10572     Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\\BlackQueen." &
        PicExtension)
10573 Case 11
10574     Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\\WhiteBishop." &
        PicExtension)
10575 Case 12
10576     Set oPiecePics(i).Picture = LoadPicture(psDocumentPath & "\\BlackBishop." &
        PicExtension)
10577 End Select
10578
10579 With oPiecePics(i)
10580     If i Mod 2 = 0 Then
10581         .Top = lFieldWidth: .Left = PieceDisplayOrder(i) * lFieldWidth
10582     Else
10583         .Top = 0: .Left = PieceDisplayOrder(i) * lFieldWidth
10584     End If
10585     .Width = lFieldWidth: .Height = lFieldWidth
10586 End With
10587 End If
10588
10589 Next
10590 End Sub
10591
10592 Private Function PieceDisplayOrder(piece As Long) As Integer
10593     Select Case piece
10594     Case WPAWN, BPAWN: PieceDisplayOrder = 0
10595     Case WKNIGHT, BKNIGHT: PieceDisplayOrder = 1
10596     Case WBISHOP, BBISHOP: PieceDisplayOrder = 2
10597     Case WROOK, BROOK: PieceDisplayOrder = 3
10598     Case WQUEEN, BQUEEN: PieceDisplayOrder = 4
10599     Case WKING, BKING: PieceDisplayOrder = 5
10600     Case Else: PieceDisplayOrder = 0
10601     End Select
10602 End Function
10603
10604 Private Sub cmdForward_Click()
10605     'TODO
10606 End Sub
10607
10608 Private Sub cmdLoadFEN_Click()
10609     Dim sFEN As String
10610     sFEN = InputBox(Translate("Enter FEN position:"), Translate("FEN position"))
10611     If Trim(sFEN) <> "" Then
10612         cboFakeInput = "setboard " & sFEN
10613         cmdFakeInput_Click

```

```

10614     End If
10615 End Sub
10616
10617 Private Sub cmdNewGame_Click()
10618     If cmdStop.Visible = True Then Exit Sub 'Thinking
10619     SendToEngine "new"
10620     txtIO = ""
10621     txtMoveList = ""
10622     Result = NO_MATE
10623     ShowBoard
10624 End Sub
10625
10626 Private Sub cmdSave_Click()
10627     Dim sFile As String
10628     If psGameFile = "" Then psGameFile = "Game1.pgn"
10629     sFile = InputBox(Translate("Enter file name to save:"), "", psGameFile)
10630     sFile = psDocumentPath & "\" & sFile
10631
10632     'Write Game File
10633     WriteGame sFile
10634
10635 End Sub
10636
10637
10638 Private Sub cmdLoad_Click()
10639     Dim sFile As String
10640     If psGameFile = "" Then psGameFile = "Game1.pgn"
10641     sFile = InputBox(Translate("Enter file name to load:"), "", psGameFile)
10642     sFile = psDocumentPath & "\" & sFile
10643
10644     If Dir(sFile) = "" Then MsgBox Translate("File not found!"): Exit Sub
10645     'Write Game File
10646     cmdNewGame_Click
10647
10648     ReadGame sFile
10649     ShowBoard
10650 End Sub
10651
10652 Private Sub cmdStop_Click()
10653     If SetupBoardMode Then Exit Sub
10654     bTimeExit = True
10655     SendCommand "---" & Translate("Stopped") & "!---"
10656 End Sub
10657
10658
10659 Private Sub SetTimeControl()
10660     Dim lMin1 As Integer, lSec1 As Integer, lSec2 As Integer, lDepth As Long, sLevel As
String
10661
10662     'SendToEngine "sd 2":Exit Sub ' Test with fixed depth
10663     If optSecondsPerMove.Value = True Then
10664         lSec1 = CLng("0" & cboSecondsPerMove.Value): If lSec1 < 1 Then lSec1 = 2 '- max
Seconds per Move
10665         SendToEngine "st " & CStr(lSec1)
10666     ElseIf optMinutesPerGame.Value = True Then
10667         lMin1 = CLng("0" & cboMinutesPerGame.Value): If lMin1 < 1 Then lMin1 = 2
10668         SendToEngine "level 0 " & CStr(lMin1) & " 0" '- max Minutes per Game
10669     ElseIf optFixedDepth.Value = True Then
10670         lDepth = CLng("0" & cboFixedDepth.Value): If lDepth < 1 Then lDepth = 5
10671         SendToEngine "sd 0 " & CStr(lDepth) ' Fixed depth
10672     ElseIf optBlitz.Value = True Then
10673         lMin1 = CLng("0" & cboBlitzMin1.Value): If lMin1 < 0 Then lMin1 = 0 '- Minutes per Game
10674         sLevel = CStr(lMin1)
10675         lSec1 = CLng("0" & cboBlitzSec1.Value): If lSec1 < 0 Then lSec1 = 0 '- Seconds per
Game
10676         If lSec1 > 0 Then sLevel = sLevel & ":" & CStr(lSec1)
10677         lSec2 = CLng("0" & cboBlitzSec2.Value): If lSec2 < 0 Then lSec2 = 0 '- Increment per
move

```

```

10678         sLevel = sLevel & " " & CStr(lSec2)
10679         SendToEngine "level " & sLevel
10680     End If
10681 End Sub
10682
10683 Private Sub SendToEngine(isCommand As String)
10684     ParseCommand isCommand & vbCrLf
10685 End Sub
10686
10687 Private Sub TranslateForm()
10688     Dim ctrl As Control, sText As String, sTextEN As String
10689
10690     If LangCnt = 0 Then Exit Sub
10691
10692     For Each ctrl In Me.Controls
10693         Select Case TypeName(ctrl)
10694             Case "CommandButton", "Label", "OptionButton", "CheckBox", "Frame"
10695                 sTextEN = ctrl.Caption
10696                 sText = Translate(sTextEN)
10697                 If sText <> sTextEN Then ctrl.Caption = sText
10698             End Select
10699         Next ctrl
10700 End Sub
10701
10702 Private Sub cmdUndo_Click()
10703     SendToEngine "undo"
10704     ShowBoard
10705     HintMove = EmptyMove
10706     ShowLastMoveAtBoard
10707     ShowMoveList
10708 End Sub
10709
10710
10711 Private Sub fraBoard_Click()
10712     ' board/square clicks are handled in class clsBoardField: ImageEvents_Click
10713 End Sub
10714
10715
10716 Private Sub InitTestSets()
10717     'txtIO = "*" STDIN HANDLE: " & hStdIn & vbTab & "STDOUT HANDLE: " & hStdOut & " " & vbCrLf
10718     txtIO = ""
10719     cboFakeInput = "setboard lb5k/7P/plp2np1/2P2p2/PP3P2/4RQ1R/q2r3P/6K1 w - - 0 1 ;e3e8
Mate in 8"
10720     'Aggiungiamo alcuni comandi di debug
10721     cboFakeInput.AddItem "setboard lb5k/7P/plp2np1/2P2p2/PP3P2/4RQ1R/q2r3P/6K1 w - - 0 1
;e3e8 Mate in 8"
10722     cboFakeInput.AddItem "setboard r4rk1/pbq2pp1/lppbpnlp/8/2PP4/1P1Q1N2/PBB2PPP/R3R1K1 w
- - 0 1; WAC249 c4c4,d4d5 "
10723     cboFakeInput.AddItem "eval" ' Show evaluation of position in thinking window and writes in Trace file
10724     cboFakeInput.AddItem "bench 3"
10725     'cboFakeInput.AddItem "bench 5"
10726     'cboFakeInput.AddItem "debug1 "
10727     'cboFakeInput.AddItem "setboard r1b2rk1/pp1nq1p1/2p1p2p/3p1p2/2PPn3/2NBPn2/PPQ2PPP/2R2RK1 b - -"
10728     'cboFakeInput.AddItem "setboard 2br2k1/ppp2p1p/4p1p1/4P2q/2P1Bn2/2Q5/PP3P1P/4R1RK b - -"
10729     'cboFakeInput.AddItem "setboard 8/8/R3k3/1R6/8/8/2K5 b - -"
10730     'cboFakeInput.AddItem "setboard 2k4r/1pr1n3/p1p1q2p/5pp1/3P1P2/P1P1P3/1R2Q1PP/1RB3K1 w KQkq -"
10731     'cboFakeInput.AddItem "setboard 6k1/1b1nqbbp/pp4p1/5P2/1PN5/4Q3/P5PP/1B2B1K1 b - -"
10732     'cboFakeInput.AddItem "display"
10733     'cboFakeInput.AddItem "xboard" & vbCrLf & "new" & vbCrLf & "random" & vbCrLf & "level 40 5 0" & vbCrLf & "post"
10734     'cboFakeInput.AddItem "xboard" & vbCrLf & "new" & vbCrLf & "random" & vbCrLf & "sd 4" & vbCrLf & "post"
10735     'cboFakeInput.AddItem "time 30000" & vbCrLf & "otim 30000" & vbCrLf & "e2e4"
10736     'cboFakeInput.AddItem "force" & vbCrLf & "quit"
10737
10738
10739     'cboFakeInput.AddItem "setboard rnbqkbnr/ppp2ppp/4p3/3pP3/3P4/8/PPP2PPP/RNBQKBNR b KQkq -"
10740     'cboFakeInput.AddItem "setboard 8/p1b1k1p1/Pp4p1/1Pp2pPp/2P2P1P/3B1K2/8/8 w - -"
10741     'cboFakeInput.AddItem "setboard 8/2R5/1r3kp1/2p4p/2P2P2/p3K1P1/P6P/8 w - -"
10742     'cboFakeInput.AddItem "setboard 7k/p7/6K1/5Q2/8/8/8 w - -"

```

```

10743 'cboFakeInput.AddItem "writepd"
10744 'cboFakeInput.AddItem "display"
10745 'cboFakeInput.AddItem "debug1"
10747 End Sub
10748
10749 Public Sub InitTimes()
10750 Dim i As Integer
10751 With cboSecondsPerMove
10752     .AddItem "1": .AddItem "2": .AddItem "3": .AddItem "5": .AddItem "8": .AddItem
10753     "10": .AddItem "15": .AddItem "20": .AddItem "30": .AddItem "60"
10754 End With
10755 With cboMinutesPerGame
10756     .AddItem "1": .AddItem "2": .AddItem "3": .AddItem "5": .AddItem "8": .AddItem
10757     "10": .AddItem "15": .AddItem "20": .AddItem "30": .AddItem "60"
10758 End With
10759 With cboFixedDepth
10760     For i = 1 To 15
10761         .AddItem CStr(i)
10762     Next
10763 End With
10764
10765 With cboBlitzMin1
10766     .AddItem "0": .AddItem "1": .AddItem "2": .AddItem "3": .AddItem "5": .AddItem "8"
10767     : .AddItem "10": .AddItem "15": .AddItem "20": .AddItem "30": .AddItem "60"
10768 End With
10769 With cboBlitzSec1
10770     .AddItem "0": .AddItem "15": .AddItem "30": .AddItem "30": .AddItem "45"
10771 End With
10772
10773 With cboBlitzSec2
10774     .AddItem "0": .AddItem "1": .AddItem "2": .AddItem "3": .AddItem "5": .AddItem "8"
10775     : .AddItem "10": .AddItem "15": .AddItem "20": .AddItem "30": .AddItem "60"
10776 End With
10777 End Sub
10778
10779 Public Sub ReadColors()
10780     WhiteSqCol = Val(ReadINISetting("WHITE_SQ_COLOR", "&HC0FFFF"))
10781     BlackSqCol = Val(ReadINISetting("BLACK_SQ_COLOR", "&H80FF&"))
10782     BoardFrameCol = Val(ReadINISetting("BOARD_FRAME_COLOR", "&H000040C0&"))
10783     fraBoard.BackColor = BoardFrameCol
10784 End Sub
10785
10786
10787 Public Sub ShowMoveList()
10788 Dim i As Integer
10789
10790 txtMoveList = ""
10791 If GameMovesCnt = 0 Then Exit Sub
10792 If arGameMoves(1).piece Mod 2 = 0 Then txtMoveList = " "
10793 For i = 1 To GameMovesCnt
10794     If Len(txtMoveList) > 32000 Then txtMoveList = ""
10795
10796     If arGameMoves(i).piece Mod 2 = 1 Then
10797         If arGameMoves(i).From > 0 Or arGameMoves(i + 1).From > 0 Then
10798             txtMoveList = txtMoveList & Left(MoveText(arGameMoves(i)) & Space(6), 6)
10799         End If
10800     Else
10801         If arGameMoves(i).From > 0 Then txtMoveList = txtMoveList & " - " & MoveText(
10802             arGameMoves(i)) & vbCrLf
10803     End If
10804 Next i
10805 txtMoveList.SetFocus: txtMoveList.SelStart = Len(txtMoveList): txtMoveList.SelLength
10806 = 0

```

```

10805     DoEvents
10806 End Sub
10807
10808 Private Sub Piece1_Click()
10809     SelectPiece 1
10810 End Sub
10811 Private Sub Piece2_Click()
10812     SelectPiece 2
10813 End Sub
10814 Private Sub Piece3_Click()
10815     SelectPiece 3
10816 End Sub
10817 Private Sub Piece4_Click()
10818     SelectPiece 4
10819 End Sub
10820 Private Sub Piece5_Click()
10821     SelectPiece 5
10822 End Sub
10823 Private Sub Piece6_Click()
10824     SelectPiece 6
10825 End Sub
10826 Private Sub Piece7_Click()
10827     SelectPiece 7
10828 End Sub
10829 Private Sub Piece8_Click()
10830     SelectPiece 8
10831 End Sub
10832 Private Sub Piece9_Click()
10833     SelectPiece 9
10834 End Sub
10835 Private Sub Piece10_Click()
10836     SelectPiece 10
10837 End Sub
10838 Private Sub Piece11_Click()
10839     SelectPiece 11
10840 End Sub
10841 Private Sub Piece12_Click()
10842     SelectPiece 12
10843 End Sub
10844 Attribute VB_Name = "basHash"
10845 '=====
10846 '= basHash:
10847 '= Hash functions for transposition table
10848 '=====
10849 Option Explicit
10850
10851 Public Const MAX_THREADS As Long = 64
10852 Public Const MAX_HASHSIZE_MB As Long = 1400 ' limit by 32 bit around 1500Mb / also long datatype
10853 'The style of the hash table rows
10854 Public Const TT_NO_BOUND As Byte = 0
10855 Public Const TT_UPPER_BOUND As Byte = 1
10856 Public Const TT_LOWER_BOUND As Byte = 2
10857 Public Const TT_EXACT As Byte = 3 ' = TT_UPPER_BOUND or TT_LOWER_BOUND !
10858 Public Const HASH_CLUSTER As Long = 4
10859 Public Const TT_TB_BASE_DEPTH As Long = 222
10860 Public Const MATERIAL_HASHSIZE As Long = 8192
10861
10862 Public Const HASH_SIZE_FACTOR As Long = 34000 ' entries per MB hash
10863
10864 Public Type THashKey
10865     ' 2x 32 bit
10866     HashKey1 As Long
10867     Hashkey2 As Long
10868 End Type
10869
10870 Public ZobristHash1 As Long ' for calculation of hash key
10871 Public ZobristHash2 As Long

```

```

10872
10873 Public HashWhiteToMove As Long ' hashkey to add for white to move
10874 Public HashWhiteToMove2 As Long
10875
10876 Public HashWCanCastle As Long
10877 Public HashWCanCastle2 As Long
10878
10879 Public HashBCanCastle As Long
10880 Public HashBCanCastle2 As Long
10881
10882 Public HashExcluded As Long
10883 Public InHashCnt As Long
10884 Public HashAccessCnt As Long
10885 Public HashUsage As Long
10886 Private bHashUsed As Boolean
10887 Public bHashVerify As Boolean
10888 Public HashGeneration As Long
10889 Public EmptyHash As THashKey
10890
10891 Private Type HashTableEntry
10892     Position1 As Long ' 2x32 bit position hash key
10893     Position2 As Long
10894     Depth As Integer ' not Byte, negative values possible for QSearch
10895     Generation As Byte
10896     IsChecking As Boolean
10897     MoveFrom As Byte
10898     MoveTarget As Byte
10899     MovePromoted As Byte
10900     EvalType As Byte
10901     Eval As Long
10902     StaticEval As Long
10903     PvHit As Boolean
10904     ThreadNum As Byte ' used for thread hit cnt => for testing only
10905 End Type
10906
10907 Private moHashMap As clsHashMap
10908 Public HashSizeMB As Long
10909 Public HashSizeMax As Long
10910 Public HashSize As Long ' in bytes
10911 Public bHashSizeIgnoreGUI As Boolean ' HASHSIZE_IGNORE_GUI
10912 Dim ZobristTable(SQ_A1 To SQ_H8, 0 To 16) As Long ' key for each piece type and board
10913     position
10914 Dim ZobristTable2(SQ_A1 To SQ_H8, 0 To 16) As Long
10915 'Dim FiftyZobristTable(0 To 100) As Long ' fifty move draw: make different hash when fifty increases> not
10916     better
10917 'Dim FiftyZobristTable2(0 To 100) As Long
10918 Dim MatZobristTable(0 To 10, 0 To 12) As Long
10919 'The main array to hold the hash table
10920 Private HashTable() As HashTableEntry
10921 Private HashCluster(0 To HASH_CLUSTER - 1) As HashTableEntry
10922 ' Pointer to multi-Thread map data
10923 Public NoOfThreads As Long
10924 Public ThreadNum As Long ' 0 = Main Thread
10925 Public MainThreadStatus As Long, LastThreadStatus As Long ' 1 =
10926     start, 0 = stop, -1 = Exit
10927 Public ThreadCommand As String
10928
10929 Public HashMapEnd As Long
10930 Public HashMapHashSizePtr As Long
10931 Public HashMapThreadStatusPtr(MAX_THREADS - 1) As Long
10932 Public HashMapBestPVPtr(MAX_THREADS - 1) As Long ' Best pv for 10 moves
10933 Public HashMapBoardPtr As Long
10934 Public HashMapMovedPtr As Long
10935 Public HashMapWhiteToMovePtr As Long
10936 Public HashMapGameMovesCntPtr As Long
10937 Public HashMapGameMovesPtr As Long
10938 Public HashMapGamePosHashPtr As Long
10939 Public HashMapSearchPtr As Long

```

```

10937
10938 Public HashRecLen As Long
10939 Public HashClusterLen As Long
10940 Private BestPV(10) As TMOVE
10941 Public SingleThreadStatus(MAX_THREADS - 1) As Long ' 1 = start, 0 = stop, -1 = Stopped
10942 Private HashMapFile As String
10943 Public bTraceHashCollision As Boolean
10944
10945 Public HashFoundFromOtherThread As Long
10946 Private Type TMaterialHashEntry
10947     Hashkey As Long
10948     Score As Long
10949 End Type
10950
10951 Public MaterialHash(MATERIAL_HASHSIZE) As TMaterialHashEntry
10952
10953 Public Sub InitHash()
10954     'Initialize the hash-table
10955     ' Use maximum hash size form INI file and memory command
10956     Dim NewHashSize As Long
10957     bHashTrace = CBool(ReadINISetting("HASHTRACE", "0") <> "0")
10958     HashSizeMB = GetMin(MAX_HASHSIZE_MB, Val(ReadINISetting("HASHSIZE", "64"))) ' 2 GB for
10959     32 bit ( max 1.5 GB?)
10960     If CBool(ReadINISetting("HASHSIZE_IGNORE_GUI", "0") = "0") Then
10961         HashSizeMB = GetMax(HashSizeMB, MemoryMB) ' memory command value from GUI
10962     End If
10963     HashSizeMB = GetMin(MAX_HASHSIZE_MB, HashSizeMB) ' in 1 core: vb array MB, in IDE max around
10964     350MB, EXE 1.5 GB
10965     If InIDE Then HashSizeMB = GetMin(128, HashSizeMB) ' Limited in IDE, depends on local memory
10966     usage
10967
10968     'HashSizeMB = 1400
10969     'NoOfThreads = 2
10970     'ThreadNum = 0 ' TEST
10971
10972 lblHashSize:
10973     If bHashTrace Then WriteTrace "Init hash size start " & HashSizeMB & "MB " & Now()
10974     If ThreadNum <= 0 Then ' for helper threads if hash size was changed
10975         If Not pbMSEExcelRunning Then
10976             WriteINISetting "HASH_USED", CStr(HashSizeMB)
10977         End If
10978     Else
10979         HashSizeMB = Val(ReadINISetting("HASH_USED", "64")) ' read from main thread
10980     End If
10981     HashSize = HashSizeMB * HASH_SIZE_FACTOR ' in Bytes, seems to fit...? hash len = 31
10982     HashUsage = 0
10983     bHashUsed = False
10984     #If VBA_MODE = 0 Then ' Find unique file name if more than one version is CB are
10985     running
10986         HashMapFile = ReadINISetting("HASH_MAP_FILE", "CBVBHash" & Trim(App.Major) & Trim(
10987         App.Minor) & Trim(App.Revision) & "_" & GetAppTimeString() & ".DAT") ' Change in INI
10988         to run 2x CB engine
10989     #End If
10990
10991     bHashVerify = CBool(ReadINISetting("HASH_VERIFY", "0") <> "0") ' verify hash read/write to
10992     avoid collisions for many cores
10993     If NoOfThreads < 2 Then bHashVerify = False
10994     bTraceHashCollision = bHashVerify And CBool(ReadINISetting("HASH_COLL_TRACE", "0")
10995     <> "0") ' trace hash read/write collisions for > 1 core
10996     HashRecLen = LenB(HashCluster(0)): HashClusterLen = HashRecLen * HASH_CLUSTER
10997
10998     If bHashTrace Then WriteTrace "InitHash: HashSize:" & HashSize & ", Threads:" &
10999     NoOfThreads
11000     If NoOfThreads <= 1 Then
11001         If bHashTrace Then WriteTrace "InitHash: Redim HashTable(0) done " & Now()
11002         If HashSize > HashSizeMax Then
11003             ReDim HashTable(HashSize + HASH_CLUSTER) ' may be OutOfMemory Error here
11004             If bHashTrace Then WriteTrace "InitHash: Redim done HashTable Size= " & HashSize

```



```

        & " entries " & Now()
HashSizeMax = HashSize
10996
10997 Else
10998     If bHashTrace Then WriteTrace "InitHash: Keep HashTable Size= " & HashSize & "
        entries " & Now()
10999     ' REDIM HashTable > creates random error: Out of memory / needs unfragmented memory fo rrequested
        size
11000     Dim j As Long
11001     For j = 1 To HashSize: HashTable(j).Position1 = 0: Next
11002 End If
11003 'MsgBox "Hashtable " & NoOfThreads & "/" & ThreadNum
11004 ElseIf NoOfThreads > 1 Then
11005     ' Structure for game data
11006     ' ThreadStatus as long ' 1 = start, 0 = stop, -1 = Exit
11007     ReDim HashTable(0) ' internal hash not needed
11008     If bHashTrace Then WriteTrace "InitHash: Init hash map " & HashSize & " Bytes " &
        Now()
11009
11010     ' HashMapEnd value starts a 0, every part of memory added will increase the value to address the next one
11011     HashMapEnd = 0
11012     If bHashTrace Then WriteTrace "HashMap: " & NoOfThreads & "/" & ThreadNum & ",
        HashMapEnd:" & HashMapEnd & " MB:" & HashSizeMB
11013     Dim i As Long
11014
11015     For i = 0 To MAX_THREADS - 1
11016         HashMapThreadStatusPtr(i) = HashMapEnd: HashMapEnd = HashMapEnd + LenB(
            MainThreadStatus)
11017         'If bHashTrace Then WriteTrace "InitHash:HashMapThreadStatusPtr:" & i & ":" & HashMapThreadStatusPtr(i)
11018     Next
11019
11020     For i = 0 To MAX_THREADS - 1
11021         HashMapBestPVPtr(i) = HashMapEnd: HashMapEnd = HashMapEnd + LenB(PV(0, 0)) * 10
11022     Next
11023
11024     HashMapBoardPtr = HashMapEnd: HashMapEnd = HashMapEnd + LenB(Board(0)) * MAX_BOARD
11025     HashMapMovedPtr = HashMapEnd: HashMapEnd = HashMapEnd + LenB(Moved(0)) * MAX_BOARD
11026     HashMapWhiteToMovePtr = HashMapEnd: HashMapEnd = HashMapEnd + LenB(bWhiteToMove)
11027     HashMapGameMovesCntPtr = HashMapEnd: HashMapEnd = HashMapEnd + LenB(GameMovesCnt)
11028     HashMapGameMovesPtr = HashMapEnd: HashMapEnd = HashMapEnd + LenB(arGameMoves(0)) *
        MAX_GAME_MOVES
11029     HashMapGamePosHashPtr = HashMapEnd: HashMapEnd = HashMapEnd + LenB(GamePosHash(0))
        * MAX_GAME_MOVES
11030
11031     ' the real hash for search is allocated now:
11032     HashMapSearchPtr = HashMapEnd
11033     HashMapEnd = HashMapEnd + HashRecLen * (HashSize + HASH_CLUSTER)
11034     ' allocate hash map file for multiple threads
11035     If ThreadNum >= 0 Then
11036         If bHashTrace Then WriteTrace "InitHash:OpenHashMap: HashMapEnd " & HashMapEnd
11037         NewHashSize = HashMapEnd
11038         OpenHashMap NewHashSize
11039         If NewHashSize <> HashMapEnd Then
11040             HashSizeMB = NewHashSize \ 1024# \ 1024# ' use reduced hash size
11041             WriteTrace "InitHash: New HashSize: " & HashSizeMB & " / " & Now()
11042             GoTo lblHashSize
11043         End If
11044     End If
11045 End If
11046 If bHashTrace Then WriteTrace "Init hash size done " & HashSize & " entries " & Now
    ()
11047 End Sub
11048
11049 Public Sub HashBoard(HashKeyOut As THashKey, ExcludedMove As TMOVE)
11050     Dim i As Long, sq As Long
11051     ZobristHash1 = 0: ZobristHash2 = 0
11052
11053     For i = 1 To NumPieces
11054         sq = Pieces(i): If sq <> 0 Then ZobristHash1 = ZobristHash1 Xor ZobristTable(sq,

```

```

Board(sq)): ZobristHash2 = ZobristHash2 Xor ZobristTable2(sq, Board(sq))
11055 Next
11056
11057 If EpPosArr(Ply) > 0 Then HashSetPiece EpPosArr(Ply), Board(EpPosArr(Ply))
11058 If bWhiteToMove Then
11059     ZobristHash1 = ZobristHash1 Xor HashWhiteToMove: ZobristHash2 = ZobristHash2 Xor
        HashWhiteToMove2
11060 End If
11061 If Moved(WKING_START) = 0 Then ' white can castle?
11062     If Moved(SQ_H1) = 0 Then ZobristHash1 = ZobristHash1 Xor HashWCanCastle
11063     If Moved(SQ_A1) = 0 Then ZobristHash2 = ZobristHash2 Xor HashWCanCastle2
11064 End If
11065 If Moved(BKING_START) = 0 Then ' black can castle?
11066     If Moved(SQ_H8) = 0 Then ZobristHash1 = ZobristHash1 Xor HashBCanCastle
11067     If Moved(SQ_A8) = 0 Then ZobristHash2 = ZobristHash2 Xor HashBCanCastle2
11068 End If
11069 If ExcludedMove.From > 0 Then ' use from/target sq to be different to normal position
11070     ZobristHash1 = ZobristHash1 Xor ZobristTable(ExcludedMove.From, ExcludedMove.Piece
        ): ZobristHash2 = ZobristHash2 Xor ZobristTable2(ExcludedMove.Target,
        ExcludedMove.Piece)
11071 End If
11072
11073 HashKeyOut.HashKey1 = ZobristHash1: HashKeyOut.Hashkey2 = ZobristHash2
11074 End Sub
11075
11076 Public Function HashGetKey() As THashKey
11077     HashGetKey.HashKey1 = ZobristHash1
11078     HashGetKey.Hashkey2 = ZobristHash2
11079 End Function
11080
11081 Public Sub NextHashGeneration()
11082     HashGeneration = GetMin(255, GameMovesCnt \ 2 + 1)
11083 End Sub
11084
11085 Public Sub HashSetKey(ByRef Hashkey As THashKey)
11086     ZobristHash1 = Hashkey.HashKey1
11087     ZobristHash2 = Hashkey.Hashkey2
11088 End Sub
11089
11090
11091 Public Function HashTableSave(Hashkey As THashKey, _
11092     Depth As Long, _
11093     HashMove As TMOVE, _
11094     EvalType As Long, _
11095     EvalScore As Long, _
11096     StaticEval As Long, _
11097     PvHit As Boolean)
11098     'Dim FiftyHash As THashKey
11099     'If Fifty >= 4 Then ' fifty move draw: make different hash when fifty increases every 8 moves > problern with 3x
        draw detection using hash
11100     ' FiftyHash.HashKey1 = Hashkey.HashKey1 Xor FiftyZobristTable(Fifty \ 8): FiftyHash.Hashkey2 =
        Hashkey.Hashkey2 Xor FiftyZobristTable2(Fifty \ 8)
11101     'Else
11102     ' FiftyHash.HashKey1 = Hashkey.HashKey1: FiftyHash.Hashkey2 = Hashkey.Hashkey2
11103     'End If
11104
11105     If ThreadNum < 0 Then ' single core using internal VB array
11106         InsertIntoHashTable Hashkey, Depth, HashMove, EvalType, EvalScore, StaticEval,
            PvHit
11107     Else ' multiple cores using global hash map
11108         InsertIntoHashMap Hashkey, Depth, HashMove, EvalType, EvalScore, StaticEval, PvHit
11109     End If
11110 End Function
11111
11112
11113
11114 Public Function InsertIntoHashTable(Hashkey As THashKey, _
11115     ByVal Depth As Long, _

```

```

11116         HashMove As TMOVE, _
11117         ByVal EvalType As Long, _
11118         ByVal EvalScore As Long, _
11119         ByVal StaticEval As Long, _
11120         ByVal PvHit As Boolean)
11121 '--- Insert hash entry into hash array if only one thread (faster than access to global mapped memory)
11122 Dim ClusterIndex As Long, NewHashMove As TMOVE, i As Long, ReplaceIndex As Long,
MaxReplaceValue As Long, ReplaceValue As Long, bPosFound As Boolean
11123 Debug.Assert HashMove.From = 0 Or (HashMove.Piece <> NO_PIECE And Board(
HashMove.From) <> NO_PIECE)
11124 If bTimeExit Then Exit Function 'score not exact
11125 SetMove NewHashMove, HashMove ' Don't overwrite move of caller function
11126 bHashUsed = True: bPosFound = False
11127 MaxReplaceValue = 9999
11128 '--- Compute hash key
11129 ZobristHash1 = Hashkey.HashKey1: ZobristHash2 = Hashkey.Hashkey2
11130 ClusterIndex = HashKeyCompute() * HASH_CLUSTER
11131 ReplaceIndex = ClusterIndex
11132 If HashAccessCnt < 2100000000 Then HashAccessCnt = HashAccessCnt + 1
11133
11134 For i = 0 To HASH_CLUSTER - 1
11135     With HashTable(ClusterIndex + i)
11136         If .Position1 = 0 Then ReplaceIndex = ClusterIndex + i: Exit For 'use empty entry
11137         If HashGeneration = .Generation Then If HashUsage < 2100000000 Then HashUsage =
HashUsage + 1
11138         ' Don't overwrite more valuable entry
11139         If (.Position1 = ZobristHash1 And .Position2 = ZobristHash2) Then
11140             ' Position found: Preserve hash move if no new move
11141             If .MoveFrom > 0 And NewHashMove.From = 0 Then 'old hash move exists
11142                 NewHashMove.From = .MoveFrom: NewHashMove.Target = .MoveTarget:
NewHashMove.Promoted = .MovePromoted: NewHashMove.IsChecking = .IsChecking
11143             End If
11144             ReplaceIndex = ClusterIndex + i: bPosFound = True
11145             Exit For
11146         Else
11147             ' Other position found. Overwrite?
11148             ReplaceValue = .Depth - 8 * (HashGeneration - .Generation)
11149             If ReplaceValue < MaxReplaceValue Then
11150                 MaxReplaceValue = ReplaceValue: ReplaceIndex = ClusterIndex + i
11151             End If
11152         End If
11153     End With
11154 Next
11155
11156 With HashTable(ReplaceIndex)
11157     '--- Save hash data, preserve hash move if no new move
11158     If Not bPosFound Or EvalType = TT_EXACT Or Depth > .Depth - 4 Then
11159         .Position1 = ZobristHash1: .Position2 = ZobristHash2
11160         .MoveFrom = NewHashMove.From: .MoveTarget = NewHashMove.Target: .MovePromoted =
NewHashMove.Promoted
11161         .EvalType = EvalType: .Eval = ScoreToHash(EvalScore)
11162         .StaticEval = StaticEval: .Depth = Depth
11163         .Generation = HashGeneration
11164         .IsChecking = NewHashMove.IsChecking
11165         .PvHit = PvHit
11166         Debug.Assert .MoveFrom = 0 Or Board(.MoveFrom) <> NO_PIECE
11167     End If
11168 End With
11169
11170
11171 End Function
11172
11173 Public Function HashTableRead(Hashkey As THashKey, _
11174     ByRef HashDepth As Long, _
11175     HashMove As TMOVE, _
11176     ByRef EvalType As Long, _
11177     ByRef EvalScore As Long, _
11178     ByRef StaticEval As Long, _

```

```

11179         ByRef PvHit As Boolean, ByRef HashThreadNum As Long) As
11180         Boolean
11181     ' Dim FiftyHash As THashKey
11182     ' If Fifty >= 4 Then ' fifty move draw: make different hash when fifty increases every 8 moves > problem with 3x draw
11183     '     detection using hash
11184     '     FiftyHash.HashKey1 = Hashkey.HashKey1 Xor FiftyZobristTable(Fifty \ 8): FiftyHash.Hashkey2 =
11185     '     Hashkey.Hashkey2 Xor FiftyZobristTable2(Fifty \ 8)
11186     ' Else
11187     '     FiftyHash.HashKey1 = Hashkey.HashKey1: FiftyHash.Hashkey2 = Hashkey.Hashkey2
11188     ' End If
11189
11190     If ThreadNum < 0 Then ' single core using internal VB array
11191         HashTableRead = IsInHashTable(Hashkey, HashDepth, HashMove, EvalType, EvalScore,
11192             StaticEval, PvHit)
11193         HashThreadNum = -1
11194     Else ' multiple cores using global hash map
11195         HashTableRead = IsInHashMap(Hashkey, HashDepth, HashMove, EvalType, EvalScore,
11196             StaticEval, PvHit, HashThreadNum)
11197     End If
11198 End Function
11199
11200 Public Function IsInHashTable(Hashkey As THashKey, _
11201     ByRef HashDepth As Long, _
11202     HashMove As TMOVE, _
11203     ByRef EvalType As Long, _
11204     ByRef EvalScore As Long, _
11205     ByRef StaticEval As Long, _
11206     ByRef PvHit As Boolean) As Boolean
11207     '--- Search for hash entry into hash array if one thread
11208     Dim IndexKey As Long, i As Long
11209
11210     IsInHashTable = False: ClearMove HashMove: EvalType = TT_NO_BOUND: EvalScore =
11211     VALUE_NONE: StaticEval = VALUE_NONE: HashDepth = -MAX_GAME_MOVES
11212     ZobristHash1 = Hashkey.HashKey1: ZobristHash2 = Hashkey.Hashkey2
11213     IndexKey = HashKeyCompute() * HASH_CLUSTER
11214
11215     For i = 0 To HASH_CLUSTER - 1
11216         If HashTable(IndexKey + i).Position1 = 0 Then If ZobristHash1 <> 0 Then Exit
11217         Function '--- empty entry, not found
11218         With HashTable(IndexKey + i)
11219             If ZobristHash1 = .Position1 And ZobristHash2 = .Position2 Then
11220                 If .Depth > HashDepth Then
11221                     ' entry found
11222                     IsInHashTable = True: PvHit = False
11223                     If InHashCnt < 2000000 Then InHashCnt = InHashCnt + 1
11224                     '--- Read hash data
11225                     If .MoveFrom > 0 Then
11226                         HashMove.From = .MoveFrom: HashMove.Target = .MoveTarget:
11227                         HashMove.IsChecking = .IsChecking
11228                         If Board(.MoveTarget) <= NO_PIECE Then HashMove.Captured = Board(.
11229                         MoveTarget)
11230                         HashMove.Piece = Board(.MoveFrom): HashMove.CapturedNumber = Squares(.
11231                         MoveTarget)
11232                         HashMove.Promoted = .MovePromoted: If HashMove.Promoted <> 0 Then
11233                         HashMove.Piece = HashMove.Promoted
11234                         Debug.Assert HashMove.Piece <> NO_PIECE
11235                         HashMove.IsLegal = True
11236
11237                     'If Not MovePossible(HashMove) Then Stop
11238                     Select Case HashMove.Piece
11239                     Case WPAWN
11240                         If .MoveTarget - .MoveFrom = 20 Then
11241                             HashMove.EnPassant = ENPASSANT_WMOVE
11242                         ElseIf Board(.MoveTarget) = BEP_PIECE Then
11243                             HashMove.EnPassant = ENPASSANT_CAPTURE
11244                             HashMove.Captured = BEP_PIECE
11245                         End If
11246                     Case BPAWN

```

```

11236         If .MoveFrom - .MoveTarget = 20 Then
11237             HashMove.EnPassant = ENPASSANT_BMOVE
11238         ElseIf Board(.MoveTarget) = WEP_PIECE Then
11239             HashMove.EnPassant = ENPASSANT_CAPTURE
11240             HashMove.Captured = WEP_PIECE
11241         End If
11242     Case WKING
11243         If .MoveFrom = SQ_E1 Then
11244             If .MoveTarget = SQ_G1 Then
11245                 HashMove.Castle = WHITEOO
11246             ElseIf .MoveTarget = SQ_C1 Then
11247                 HashMove.Castle = WHITEOOO
11248             End If
11249         End If
11250     Case BKING
11251         If .MoveFrom = SQ_E8 Then
11252             If .MoveTarget = SQ_G8 Then
11253                 HashMove.Castle = BLACKOO
11254             ElseIf .MoveTarget = SQ_C8 Then
11255                 HashMove.Castle = BLACKOOO
11256             End If
11257         End If
11258     End Select
11259 End If
11260 EvalType = .EvalType: EvalScore = HashToScore(.Eval): StaticEval = .
    StaticEval
11261 HashDepth = .Depth
11262 PvHit = .PvHit
11263 .Generation = HashGeneration ' Update generation> still valid in this game
11264 Exit For
11265 End If
11266 End If
11267 End With
11268 Next
11269
11270 End Function
11271
11272 Public Function LimitDouble(ByVal d As Double) As Long
11273     ' Prevent overflow by looping off anything beyond 31 bits
11274     Const MaxNumber As Double = 2 ^ 31
11275     LimitDouble = CLng(d - (Fix(d / MaxNumber) * MaxNumber))
11276 End Function
11277
11278 Public Sub InitZobrist()
11279     ' init values for hash calculation. 2x32 bit for 64 bit key
11280     Static bDone As Boolean
11281     Dim p As Long, s As Long
11282     If bDone Then Exit Sub
11283     bDone = True
11284     ZobristHash1 = 0: ZobristHash2 = 0
11285     Randomize 1001 ' init random generator with fix value
11286
11287     ' create hash value for each piece type and each board position
11288     For s = SQ_A1 To SQ_H8
11289         For p = 0 To 16
11290             ZobristTable(s, p) = CalcUniqueKey(): ZobristTable2(s, p) = CalcUniqueKey()
11291         Next
11292     Next
11293
11294     HashWhiteToMove = CalcUniqueKey(): HashWhiteToMove2 = CalcUniqueKey()
11295     HashWCanCastle = CalcUniqueKey(): HashWCanCastle2 = CalcUniqueKey()
11296     HashBCanCastle = CalcUniqueKey(): HashBCanCastle2 = CalcUniqueKey()
11297
11298     ' ' for rule: draw after fifty quiet moves , make a different hash key when fifty counter increases
11299     ' For s = 1 To 100
11300     '     FiftyZobristTable(s) = CalcUniqueKey(): FiftyZobristTable2(s) = CalcUniqueKey()
11301     ' Next
11302

```

```

11303 ' keys for material values total
11304 For s = 0 To 10 ' Material hash: Piece cnt
11305     For p = 0 To 12 ' Piece
11306         MatZobristTable(s, p) = CalcUniqueKey()
11307     Next
11308 Next
11309
11310 End Sub
11311
11312 Public Function CalcMaterialKey() As Long
11313     CalcMaterialKey = MatZobristTable(PieceCnt(WQUEEN), WQUEEN) Xor MatZobristTable(
11314         PieceCnt(BQUEEN), BQUEEN) Xor MatZobristTable(PieceCnt(WROOK), WROOK) Xor
11315         MatZobristTable(PieceCnt(BROOK), BROOK) Xor MatZobristTable(PieceCnt(WBISHOP),
11316         WBISHOP) Xor MatZobristTable(PieceCnt(BBISHOP), BBISHOP) Xor MatZobristTable(
11317         PieceCnt(WKNIGHT), WKNIGHT) Xor MatZobristTable(PieceCnt(BKNIGHT), BKNIGHT) Xor
11318         MatZobristTable(PieceCnt(WPAWN), WPAWN) Xor MatZobristTable(PieceCnt(BPAWN), BPAWN)
11319 End Function
11320
11321 Private Function CalcUniqueKey() As Long
11322     Static KeyList((SQ_H8 + 1) * 17 * 2 + 8) As Long
11323     Static ListCnt As Long
11324     Dim l As Long, i As Long
11325 NextTry:
11326     l = 65536 * (Int(Rnd * 65536) - 32768) Or Int(Rnd * 65536)
11327
11328     For i = 1 To ListCnt
11329         If KeyList(i) = l Then GoTo NextTry
11330     Next
11331
11332     ListCnt = ListCnt + 1: KeyList(ListCnt) = l
11333     CalcUniqueKey = l
11334 End Function
11335
11336 Public Sub HashSetPiece(ByVal Position As Long, ByVal Piece As Long)
11337     If Piece = FRAME Or Piece = NO_PIECE Then Exit Sub
11338     ZobristHash1 = ZobristHash1 Xor ZobristTable(Position, Piece)
11339     ZobristHash2 = ZobristHash2 Xor ZobristTable2(Position, Piece)
11340 End Sub
11341
11342 Public Sub HashDelPiece(ByVal Position As Long, ByVal Piece As Long)
11343     If Piece = FRAME Or Piece = NO_PIECE Then Exit Sub
11344     ZobristHash1 = ZobristHash1 Xor ZobristTable(Position, Piece)
11345     ZobristHash2 = ZobristHash2 Xor ZobristTable2(Position, Piece)
11346 End Sub
11347
11348 Public Sub HashMovePiece(ByVal From As Long, Target As Long, ByVal Piece As Long)
11349     ZobristHash1 = ZobristHash1 Xor ZobristTable(From, Piece) Xor ZobristTable(Target,
11350         Piece)
11351     ZobristHash2 = ZobristHash2 Xor ZobristTable(From, Piece) Xor ZobristTable2(Target,
11352         Piece)
11353 End Sub
11354
11355 Public Function HashKeyCompute() As Long
11356     HashKeyCompute = ZobristHash1 Xor ZobristHash2
11357     If HashKeyCompute = -2147483648# Then HashKeyCompute = HashKeyCompute + 1
11358     HashKeyCompute = Abs(HashKeyCompute) Mod (HashSize \ HASH_CLUSTER)
11359 End Function
11360
11361 Public Function HashKeyComputeMap() As Long
11362     HashKeyComputeMap = ZobristHash1 Xor ZobristHash2
11363     If HashKeyComputeMap = -2147483648# Then HashKeyComputeMap = HashKeyComputeMap + 1
11364     HashKeyComputeMap = Abs(HashKeyComputeMap) Mod (HashSize \ HASH_CLUSTER)
11365 End Function
11366
11367 Public Sub SetHashToMove()
11368     If bWhiteToMove Then
11369         ZobristHash1 = ZobristHash1 Xor HashWhiteToMove: ZobristHash2 = ZobristHash2 Xor
11370         HashWhiteToMove2

```

```

11363     End If
11364 End Sub
11365
11366 Public Sub HashSetCastle()
11367     If WhiteCastled = NO_CASTLE Then ZobristHash1 = ZobristHash1 Xor HashWCanCastle:
ZobristHash2 = ZobristHash2 Xor HashWCanCastle2
11368     If BlackCastled = NO_CASTLE Then ZobristHash1 = ZobristHash1 Xor HashBCanCastle:
ZobristHash2 = ZobristHash2 Xor HashBCanCastle2
11369 End Sub
11370
11371 Public Function ScoreToHash(ByVal Score As Long) As Long
11372     If Score = VALUE_NONE Then
11373         ScoreToHash = Score
11374     ElseIf Score >= MATE_IN_MAX_PLY Then
11375         ScoreToHash = Score + Ply
11376     ElseIf Score <= -MATE_IN_MAX_PLY Then
11377         ScoreToHash = Score - Ply
11378     Else
11379         ScoreToHash = Score
11380     End If
11381 End Function
11382
11383 Public Function HashToScore(ByVal Score As Long) As Long
11384     If Score = VALUE_NONE Then
11385         HashToScore = Score
11386     ElseIf Score >= MATE_IN_MAX_PLY Then
11387         HashToScore = Score - Ply
11388     ElseIf Score <= -MATE_IN_MAX_PLY Then
11389         HashToScore = Score + Ply
11390     Else
11391         HashToScore = Score
11392     End If
11393 End Function
11394
11395 Public Function HashUsagePerc() As String
11396     If HashSize = 0 Then
11397         HashUsagePerc = ""
11398     Else
11399         If HashUsage > HashSize Then HashUsage = HashSize
11400         HashUsagePerc = Format$(CDBl(HashUsage) * 100 & / HashSize, "0.0")
11401     End If
11402 End Function
11403
11404 Public Function HashUsageUCI() As Long
11405     Dim x As Single
11406     If HashSize = 0 Or HashUsage <= 0 Then
11407         HashUsageUCI = 0
11408     Else
11409         x = HashUsage: x = x * CSng(1000) / CSng(1 + HashAccessCnt)
11410         HashUsageUCI = GetMin(1000, CLng(x))
11411     End If
11412 End Function
11413
11414 Public Function OpenHashMap(ByRef TotalSize As Long) As Long
11415     '--- init global mapped memory if more then one thread, used by all threads!
11416     Static OldHashSize As Long
11417     If OldHashSize = 0 Then
11418         Set moHashMap = New clsHashMap
11419     End If
11420     If OldHashSize = 0 Or OldHashSize <> TotalSize Then
11421         If ThreadNum = 0 Then
11422             If OldHashSize = 0 Then
11423                 Set moHashMap = New clsHashMap
11424                 If bThreadTrace Then WriteTrace "OpenHashMap: New clsHashMap: " & TotalSize
11425             Else
11426                 If bThreadTrace Then WriteTrace "OpenHashMap: CloseMap"
11427                 moHashMap.CloseMap
11428             End If

```



```

11429     moHashMap.CreateMap HashMapFile, TotalSize ' TotalSize may be reduced if not enough memory
11430     !!!
11431     If bThreadTrace Then WriteTrace "OpenHashMap: CreateMap: Size " & TotalSize
11432     ElseIf ThreadNum > 0 Then
11433         moHashMap.OpenMap HashMapFile, TotalSize
11434         If bThreadTrace Then WriteTrace "OpenHashMap: OpenMap: Size " & TotalSize
11435     End If
11436     OldHashSize = TotalSize
11437 Else
11438     If ThreadNum = 0 Then moHashMap.ClearMap TotalSize
11439 End If
11440 End Function
11441
11442 Public Function CloseHashMap() As Long
11443     moHashMap.CloseMap
11444 End Function
11445
11446 Public Function InsertIntoHashMap(Hashkey As THashKey, _
11447     ByVal Depth As Long, _
11448     HashMove As TMOVE, _
11449     ByVal EvalType As Long, _
11450     ByVal Eval As Long, _
11451     ByVal StaticEval As Long, _
11452     ByVal PvHit As Boolean)
11453     '--- Insert hash entry into global mapped memory if more then one thread, used by all threads!
11454     Dim ClusterIndex As Long, NewHashMove As TMOVE, i As Long, ReplaceIndex As Long,
11455     MaxReplaceValue As Long, ReplaceValue As Long, bPosFound As Boolean
11456     Debug.Assert HashMove.From = 0 Or HashMove.Piece <> NO_PIECE
11457     Debug.Assert NoOfThreads > 1
11458     If bTimeExit Then Exit Function ' score not exact
11459
11460     'If ThreadNum > 0 Then Exit Function '#####TESTc2
11461
11462     SetMove NewHashMove, HashMove ' Don't overwrite
11463     bHashUsed = True: bPosFound = False
11464     MaxReplaceValue = 9999
11465     '--- Compute hash key
11466     ZobristHash1 = Hashkey.HashKey1: ZobristHash2 = Hashkey.Hashkey2
11467     ClusterIndex = HashKeyComputeMap() * HASH_CLUSTER
11468     ReplaceIndex = 0
11469     moHashMap.ReadMapHashCluster ClusterIndex, VarPtr(HashCluster(0)), HashClusterLen '
11470     read this cluster only
11471     If HashAccessCnt < 2100000000 Then HashAccessCnt = HashAccessCnt + 1
11472
11473     For i = 0 To HASH_CLUSTER - 1
11474         With HashCluster(i) ' search in retrieved cluster
11475             If .Position1 = 0 Then ReplaceIndex = ClusterIndex + i: Exit For ' empty entry found
11476             If HashGeneration = .Generation Then If HashUsage < 2100000000 Then HashUsage =
11477             HashUsage + 1
11478             ' Don't overwrite more valuable entry
11479             If (.Position1 = ZobristHash1 And .Position2 = ZobristHash2) Then
11480                 ' Position found: Preserve hash move if no new move
11481                 If NewHashMove.From = 0 And .MoveFrom > 0 Then
11482                     NewHashMove.From = .MoveFrom: NewHashMove.Target = .MoveTarget:
11483                     NewHashMove.Promoted = .MovePromoted: NewHashMove.IsChecking = .IsChecking
11484                 End If
11485                 ReplaceIndex = ClusterIndex + i: bPosFound = True
11486                 Exit For
11487             Else
11488                 ' Other position found. Find least valuable entry
11489                 ReplaceValue = .Depth - 8 * (HashGeneration - .Generation)
11490                 If ReplaceValue < MaxReplaceValue Then
11491                     MaxReplaceValue = ReplaceValue: ReplaceIndex = ClusterIndex + i
11492                 End If
11493             End If
11494         End With
11495     Next

```

```

11492 With HashCluster(ReplaceIndex - ClusterIndex)
11493 '--- Save hash data, preserve hash move if no new move
11494 If Not bPosFound Or EvalType = TT_EXACT Or Depth > .Depth - 4 Then
11495     .Position1 = ZobristHash1: .Position2 = ZobristHash2
11496     .MoveFrom = NewHashMove.From: .MoveTarget = NewHashMove.Target: .MovePromoted =
NewHashMove.Promoted
11497     .EvalType = EvalType: .Eval = ScoreToHash(Eval)
11498     .StaticEval = StaticEval: .Depth = Depth
11499     .Generation = HashGeneration
11500     .IsChecking = NewHashMove.IsChecking
11501     .PvHit = PvHit
11502     If ThreadNum >= 0 Then .ThreadNum = ThreadNum
11503     '--- Write Hash Map: replace index in Cluster only
11504     moHashMap.WriteMapHashEntry ReplaceIndex, VarPtr(HashCluster(ReplaceIndex -
ClusterIndex))
11505     Debug.Assert .MoveFrom = 0 Or Board(.MoveFrom) <> NO_PIECE
11506 End If
11507 End With
11508
11509 End Function
11510
11511 Public Function IsInHashMap(Hashkey As THashKey, _
11512     ByRef HashDepth As Long, _
11513     HashMove As TMOVE, _
11514     ByRef EvalType As Long, _
11515     ByRef Eval As Long, _
11516     ByRef StaticEval As Long, _
11517     ByRef PvHit As Boolean, ByRef HashThreadNum As Long) As
Boolean
11518 '--- search for hash entry in global mapped memory if more then one thread
11519 Dim IndexKey As Long, i As Long
11520 Debug.Assert NoOfThreads > 1
11521 IsInHashMap = False: ClearMove HashMove: EvalType = TT_NO_BOUND: Eval = VALUE_NONE:
StaticEval = VALUE_NONE: HashDepth = -MAX_GAME_MOVES
11522 ZobristHash1 = Hashkey.HashKey1: ZobristHash2 = Hashkey.HashKey2
11523 IndexKey = HashKeyComputeMap() * HASH_CLUSTER
11524 moHashMap.ReadMapHashCluster IndexKey, VarPtr(HashCluster(0)), HashClusterLen
11525
11526 For i = 0 To HASH_CLUSTER - 1
11527
11528     With HashCluster(i)
11529         If .Position1 = 0 Then If ZobristHash1 <> 0 Then Exit Function '--- empty entry, not
found
11530         If ZobristHash1 = .Position1 And ZobristHash2 = .Position2 Then
11531             If .Depth > HashDepth Then
11532                 'entry found
11533                 IsInHashMap = True: PvHit = False
11534                 If InHashCnt < 2000000 Then InHashCnt = InHashCnt + 1
11535                 '--- Read hash data
11536                 If .MoveFrom > 0 Then
11537                     HashMove.From = .MoveFrom: HashMove.Target = .MoveTarget:
HashMove.IsChecking = .IsChecking
11538                     If Board(.MoveTarget) <= NO_PIECE Then HashMove.Captured = Board(.
MoveTarget)
11539                     HashMove.Piece = Board(.MoveFrom): HashMove.CapturedNumber = Squares(.
MoveTarget)
11540                     HashMove.Promoted = .MovePromoted: If HashMove.Promoted <> 0 Then
HashMove.Piece = HashMove.Promoted
11541                     Debug.Assert HashMove.Piece <> NO_PIECE
11542                     HashMove.IsLegal = True
11543
11544                     'If Not MovePossible(HashMove) Then Stop
11545                     Select Case HashMove.Piece
11546                     Case WPAWN
11547                         If .MoveTarget - .MoveFrom = 20 Then
11548                             HashMove.EnPassant = ENPASSANT_WMOVE
11549                         ElseIf Board(.MoveTarget) = BEP_PIECE Then
11550                             HashMove.EnPassant = ENPASSANT_CAPTURE

```

```

11551         HashMove.Captured = BEP_PIECE
11552     End If
11553     Case BPAWN
11554         If .MoveFrom - .MoveTarget = 20 Then
11555             HashMove.EnPassant = ENPASSANT_BMOVE
11556         ElseIf Board(.MoveTarget) = WEP_PIECE Then
11557             HashMove.EnPassant = ENPASSANT_CAPTURE
11558             HashMove.Captured = WEP_PIECE
11559         End If
11560     Case WKING
11561         If .MoveFrom = SQ_E1 Then
11562             If .MoveTarget = SQ_G1 Then
11563                 HashMove.Castle = WHITEOO
11564             ElseIf .MoveTarget = SQ_C1 Then
11565                 HashMove.Castle = WHITEOOO
11566             End If
11567         End If
11568     Case BKING
11569         If .MoveFrom = SQ_E8 Then
11570             If .MoveTarget = SQ_G8 Then
11571                 HashMove.Castle = BLACKOO
11572             ElseIf .MoveTarget = SQ_C8 Then
11573                 HashMove.Castle = BLACKOOO
11574             End If
11575         End If
11576     End Select
11577 End If
11578 ' Read values for entry
11579 EvalType = .EvalType: Eval = HashToScore(.Eval): StaticEval = .StaticEval
11580 HashDepth = .Depth
11581 PvHit = .PvHit
11582 HashThreadNum = .ThreadNum
11583 If .Generation <> HashGeneration Then
11584     .Generation = HashGeneration ' Update generation, each game move is a new generation
11585     '--- Write Hash Map: replace index in Cluster only
11586     moHashMap.WriteMapHashEntry IndexKey + i, VarPtr(HashCluster(i))
11587 End If
11588 'If ThreadNum >= 0 Then If .ThreadNum <> GetMax(0, ThreadNum) Then
11589     HashFoundFromOtherThread = HashFoundFromOtherThread + 1
11590 Exit For
11591 End If
11592 End If
11593 End With
11594 Next i
11595 End Function
11596
11597 Public Function InitThreads()
11598     Static bInitDone As Boolean
11599     Dim i As Long
11600     DoEvents
11601     #If VBA_MODE = 0 Then
11602         If Not bInitDone And NoOfThreads > 1 Then
11603             If CreateAppLockFile() Then ' Already started?
11604                 If bThreadTrace Then WriteTrace "InitThreads: NoOfThreads=" & NoOfThreads
11605                 MainThreadStatus = 0: WriteMainThreadStatus 0 ' idle
11606             ' Dim tStart As Single, tEnd As Single
11607             ' tStart = Timer
11608             ' Dim sCmd As String
11609             For i = 2 To NoOfThreads
11610                 StartProcess App.Path & "\ChessBrainVB.exe thread" & Trim$(CStr(i - 1)) '
11611                 Much faster
11612             Next
11613             '---Shell App.Path & "\ChessBrainVB.exe thread" & Trim$(CStr(i - 1)), vbMinimizedNoFocus ' SHELL is
11614             MUCH slower ( 1 sec per call!?)
11615             ' tEnd = Timer()
11616             ' WriteTrace "Threads started:" & ", Time:" & Format$(tEnd - tStart, "0.00000")

```

```

11616         Sleep 500
11617     End If
11618 End If
11619 #End If
11620 bInitDone = True
11621 End Function
11622
11623
11624 Public Function CreateAppLockFile() As Boolean
11625     'for main thread: create a locked file that gets unlocked when main thread end/crashed
11626     'this file is checked by the helper threads: if file is unlocked also exit helper threads
11627     Static lLOCK_FILEHANDLE As Long
11628     Sleep 200 ' wait for end of previous exe run
11629     #If VBA_MODE = 0 Then
11630         Debug.Assert NoOfThreads > 1
11631         lLOCK_FILEHANDLE = FreeFile()
11632         On Error GoTo lblLockErr
11633         Open App.Path & "\CB_THREAD0.TXT" For Append Access Write Lock Write As
11634             #lLOCK_FILEHANDLE
11635         Print #lLOCK_FILEHANDLE, "Temporary lock file. Main thread started:" & Now()
11636         CreateAppLockFile = True
11637     #End If
11638     lblExit:
11639     Exit Function
11640     lblLockErr:
11641     CreateAppLockFile = False
11642     WriteTrace "Already started? Cannot open Application lock file: CB_THREAD0.TXT " &
11643         Now()
11644     Resume lblExit
11645 End Function
11646
11647 Public Function CheckAppLockFile() As Boolean
11648     'this file is checked is used by the helper threads: returns true if file is unlocked > also exit helper threads
11649     Dim lLOCK_FILEHANDLE2 As Long
11650     On Error GoTo lblErr
11651     CheckAppLockFile = False
11652     #If VBA_MODE = 0 Then
11653         lLOCK_FILEHANDLE2 = FreeFile()
11654         Open App.Path & "\CB_THREAD0.TXT" For Append Access Write Lock Write As
11655             #lLOCK_FILEHANDLE2
11656         CheckAppLockFile = False ' File unlocked-> main thread was terminated-> exit helper threads too
11657         Close #lLOCK_FILEHANDLE2
11658     #End If
11659     Exit Function
11660     lblErr:
11661     CheckAppLockFile = True
11662 End Function
11663
11664 Public Function WriteLog(isLine As String) As Boolean
11665     'write debug log
11666     Dim lLOCK_FILEHANDLE3 As Long
11667     #If VBA_MODE = 0 Then
11668         lLOCK_FILEHANDLE3 = FreeFile()
11669         Open psEnginePath & "\DEBUG_LOG.TXT" For Append As #lLOCK_FILEHANDLE3
11670         Print #lLOCK_FILEHANDLE3, isLine
11671         Close #lLOCK_FILEHANDLE3
11672     #End If
11673 End Function
11674
11675 Public Sub CheckThreadTermination(ByVal bCheckAlways As Boolean)
11676     Debug.Assert NoOfThreads > 1
11677     If ThreadNum >= 1 Then
11678         If bCheckAlways Or (Nodes > LastThreadCheckNodesCnt + (GUICheckIntervalNodes * 50
11679             )) Then
11680             LastThreadCheckNodesCnt = Nodes
11681             If Not CheckAppLockFile() Then

```

```

11680         '>>> END of program here because main thread was terminated
11681     CloseHashMap
11682     If bThreadTrace Then WriteTrace "!!! Main Thread terminated: Stop helper
        thread! " & Now()
11683     End '<<<<
11684 End If
11685 End If
11686 End If
11687 End Sub
11688
11689 Public Function WriteMainThreadStatus(ByVal ilNewThreadStatus As Long) As Long
11690     Debug.Assert NoOfThreads > 1
11691     SingleThreadStatus(0) = ilNewThreadStatus
11692     moHashMap.WriteMapPos HashMapThreadStatusPtr(0), VarPtr(ilNewThreadStatus), CLng(
        LenB(ilNewThreadStatus))
11693     If bThreadTrace Then WriteTrace "WriteMainThreadStatus: " & HashMapThreadStatusPtr(0
        )
11694 End Function
11695
11696 Public Function ReadMainThreadStatus() As Long
11697     Static LastRead As Long
11698     Dim MainThreadStatus As Long
11699     Debug.Assert NoOfThreads > 1
11700     moHashMap.ReadMapPos HashMapThreadStatusPtr(0), VarPtr(MainThreadStatus), CLng(LenB(
        MainThreadStatus))
11701     SingleThreadStatus(0) = MainThreadStatus
11702     ReadMainThreadStatus = MainThreadStatus
11703     If bThreadTrace Then If LastRead <> ReadMainThreadStatus Then WriteTrace
        "ReadMainThreadStatus:Threadnum=" & ThreadNum & ", Ptr:" & HashMapThreadStatusPtr(0)
        & ", MainStatus:" & ReadMainThreadStatus & " / " & Now()
11704     LastRead = ReadMainThreadStatus
11705 End Function
11706
11707 Public Function WriteHelperThreadStatus(ByVal ilThreadNum As Long, _
        ByVal ilNewThreadStatus As Long) As Long
11708     ' Write run status for current thread
11709     Debug.Assert NoOfThreads > 1 And ilThreadNum > 0
11710     SingleThreadStatus(ilThreadNum) = ilNewThreadStatus
11711     moHashMap.WriteMapPos HashMapThreadStatusPtr(ilThreadNum), VarPtr(ilNewThreadStatus
        ), CLng(LenB(ilNewThreadStatus))
11712 End Function
11713
11714
11715 Public Function ReadHelperThreadStatus(ByVal ilThreadNum As Long) As Long
11716     ' Write run status for current thread
11717     Dim HelperThreadStatus As Long
11718     Debug.Assert NoOfThreads > 1 And ilThreadNum > 0
11719     moHashMap.ReadMapPos HashMapThreadStatusPtr(ilThreadNum), VarPtr(HelperThreadStatus
        ), CLng(LenB(HelperThreadStatus))
11720     SingleThreadStatus(ilThreadNum) = HelperThreadStatus
11721     ReadHelperThreadStatus = HelperThreadStatus
11722 End Function
11723
11724 Public Function WriteMapGameData() As Long
11725     ' Write game moves to map for other threads
11726     Debug.Assert NoOfThreads > 1
11727     moHashMap.WriteMapPos HashMapBoardPtr, VarPtr(Board(0)), CLng(LenB(Board(0)) *
        MAX_BOARD)
11728     moHashMap.WriteMapPos HashMapMovedPtr, VarPtr(Moved(0)), CLng(LenB(Moved(0)) *
        MAX_BOARD)
11729     moHashMap.WriteMapPos HashMapWhiteToMovePtr, VarPtr(bWhiteToMove), CLng(LenB(
        bWhiteToMove))
11730     moHashMap.WriteMapPos HashMapGameMovesCntPtr, VarPtr(GameMovesCnt), CLng(LenB(
        GameMovesCnt))
11731     arGameMoves(MAX_GAME_MOVES - 1).Target = Fifty ' tricky fix to avoid new map size
11732     moHashMap.WriteMapPos HashMapGameMovesPtr, VarPtr(arGameMoves(0)), CLng(LenB(
        arGameMoves(0)) * MAX_GAME_MOVES)
11733     moHashMap.WriteMapPos HashMapGamePosHashPtr, VarPtr(GamePosHash(0)), CLng(LenB(
        GamePosHash(0)) * MAX_GAME_MOVES)

```

```

11734 End Function
11735
11736 Public Function ReadMapGameData() As Long
11737     ' Read game moves to map for other threads
11738     Dim bToMove As Boolean
11739     Debug.Assert NoOfThreads > 1
11740     moHashMap.ReadMapPos HashMapBoardPtr, VarPtr(Board(0)), CLng(LenB(Board(0)) *
MAX_BOARD)
11741     InitEpArr
11742     moHashMap.ReadMapPos HashMapMovedPtr, VarPtr(Moved(0)), CLng(LenB(Moved(0)) *
MAX_BOARD)
11743     moHashMap.ReadMapPos HashMapWhiteToMovePtr, VarPtr(bToMove), CLng(LenB(bToMove))
11744     bWhiteToMove = bToMove: bCompIsWhite = bWhiteToMove
11745     moHashMap.ReadMapPos HashMapGameMovesCntPtr, VarPtr(GameMovesCnt), CLng(LenB(
GameMovesCnt))
11746     moHashMap.ReadMapPos HashMapGameMovesPtr, VarPtr(arGameMoves(0)), CLng(LenB(
arGameMoves(0)) * MAX_GAME_MOVES)
11747     Fifty = arGameMoves(MAX_GAME_MOVES - 1).Target ' tricky fix to avoid new map size
11748     moHashMap.ReadMapPos HashMapGamePosHashPtr, VarPtr(GamePosHash(0)), CLng(LenB(
GamePosHash(0)) * MAX_GAME_MOVES)
11749     InitPieceSquares
11750 End Function
11751
11752 Public Function ClearMapBestPVforThread() As Long
11753     Dim th As Long
11754     Erase BestPV()
11755
11756     For th = 0 To MAX_THREADS - 1
11757         moHashMap.WriteMapPos HashMapBestPVPtr(th), VarPtr(BestPV(0)), CLng(LenB(BestPV(0
)) * 10)
11758     Next
11759
11760 End Function
11761
11762 Public Function WriteMapBestPVforThread(ByVal CompletedDepth As Long, _
ByVal BestScore As Long, _
BestMove As TMOVE) As Long
11763
11764     ' Write PV from helper thread for main thread
11765     Dim i As Long
11766     Debug.Assert NoOfThreads > 1
11767     Debug.Assert HashMapBestPVPtr(ThreadNum) + CLng(LenB(PV(0, 0)) * 10) <
HashMapBoardPtr
11768     ' Use PV0 to store some values... not nice...
11769     Erase BestPV
11770     If CompletedDepth > 0 Then
11771         For i = 0 To GetMin(9, PVLength(1)): BestPV(i) = PV(1, i): Next
11772         If BestPV(1).From = 0 Then
11773             ' use BestMove instead
11774             BestPV(1) = BestMove: BestPV(0).From = 1
11775         End If
11776     End If
11777     BestPV(0).Target = CompletedDepth: BestPV(0).SeeValue = BestScore: BestPV(0).From =
GetMin(9, PVLength(1)): BestPV(0).OrderValue = Nodes
11778     If bThreadTrace Then WriteTrace "WriteMapBestPVforThread: D:" & CompletedDepth & ",
PV:" & MoveText(BestPV(1)) & " / " & Now()
11779     moHashMap.WriteMapPos HashMapBestPVPtr(ThreadNum), VarPtr(BestPV(0)), CLng(LenB(
BestPV(0)) * 10)
11780 End Function
11781
11782 Public Function ReadMapBestPVforThread(ByVal SelThread As Long, _
ByRef CompletedDepth As Long, _
ByRef BestScore As Long, _
ByRef BestPVLength As Long, _
ByRef HelperNodes As Long, _
BestPV() As TMOVE) As Boolean
11783
11784     ' Write PV from helper thread for main thread
11785     Debug.Assert NoOfThreads > 1

```

```

11792 Debug.Assert HashMapBestPVPtr(SelThread) + CLng(LenB(BestPV(0)) * 10) <
HashMapBoardPtr
11793 ReadMapBestPVforThread = False
11794 Erase BestPV
11795 ' Use PV0 to get some values... not nice...
11796 moHashMap.ReadMapPos HashMapBestPVPtr(SelThread), VarPtr(BestPV(0)), CLng(LenB(
BestPV(0)) * 10)
11797 CompletedDepth = BestPV(0).Target: BestScore = BestPV(0).SeeValue: BestPVLength =
BestPV(0).From: HelperNodes = BestPV(0).OrderValue
11798 If BestPV(1).From = 0 Or BestPV(1).Target = 0 Then
11799     If bThreadTrace Then WriteTrace "!!!!??ReadMapBestPVforThread:PV Empty Thread:" &
SelThread & ", Completed Depth:" & CompletedDepth
11800 End If
11801 If bThreadTrace Then WriteTrace "ReadMapBestPVforThread: PV:" & MoveText(BestPV(1))
& " / " & Now()
11802 ReadMapBestPVforThread = (BestPVLength > 0)
11803 End Function
11804
11805 Public Function SetThreads(ByVal iMaxThreads As Long)
11806     'set thread numbers: 1-4
11807     NoOfThreads = GetMax(1, Val("0" & Trim$(ReadINISetting("THREADS", "1"))))
11808     NoOfThreads = GetMax(NoOfThreads, iMaxThreads)
11809     NoOfThreads = GetMin(NoOfThreads, MAX_THREADS)
11810     'NoOfThreads = 2 '#####testc2
11811     If NoOfThreads <= 1 Then
11812         ThreadNum = -1 ' Single core mode
11813     Else
11814         ThreadNum = 0 ' main thread
11815     End If
11816     'WriteTrace "SetThreads=" & NoOfThreads & " / " & Now()
11817 End Function
11818
11819 Public Function MaterialHashCompute(ByVal Key As Long) As Long
11820     If Key = -2147483648# Then Key = Key + 1
11821     MaterialHashCompute = Abs(Key) Mod MATERIAL_HASHSIZE
11822 End Function
11823
11824 Public Function SaveMaterialHash(ByVal Key As Long, ByVal Score As Long)
11825     Dim Index As Long
11826     Index = MaterialHashCompute(Key)
11827
11828     With MaterialHash(Index)
11829         .Hashkey = Key
11830         .Score = Score
11831     End With
11832
11833 End Function
11834
11835 Public Function ProbeMaterialHash(ByVal Key As Long) As Long
11836     Dim Index As Long
11837     Index = MaterialHashCompute(Key)
11838
11839     With MaterialHash(Index)
11840         If .Hashkey = Key Then
11841             ProbeMaterialHash = .Score
11842         Else
11843             ProbeMaterialHash = VALUE_NONE
11844         End If
11845     End With
11846
11847 End Function
11848
11849 Public Function InIDE() As Boolean
11850     'running IDE ( VB development environment) ? if compiled EXE returns false
11851     Static i As Byte
11852     i = i + 1
11853     If i = 1 Then Debug.Assert Not InIDE()
11854     InIDE = (i = 0)

```



```

11855         i = 0
11856     End Function
11857
11858     Public Function GetAppTimeString() As String
11859         'returns exe filedatetime with digits only
11860         Dim p As Long, s As String
11861         GetAppTimeString = ""
11862         s = Now()
11863         #If VBA_MODE = 0 Then
11864             If Dir(App.EXENAME & ".exe") <> "" Then
11865                 s = FileDateTime(App.EXENAME & ".exe")
11866             End If
11867         #End If
11868         For p = 1 To Len(s)
11869             If IsNumeric(Mid$(s, p, 1)) Then GetAppTimeString = GetAppTimeString & Mid$(s, p,
11870                 1)
11871         Next
11872     End Function
11873
11874
11875
11876
11877     VERSION 1.0 CLASS
11878     BEGIN
11879         MultiUse = -1 'True
11880         Persistable = 0 'NotPersistable
11881         DataBindingBehavior = 0 'vbNone
11882         DataSourceBehavior = 0 'vbNone
11883         MTSTransactionMode = 0 'NotAnMTSObject
11884     END
11885     Attribute VB_Name = "clsHashMap"
11886     Attribute VB_GlobalNameSpace = False
11887     Attribute VB_Creatable = True
11888     Attribute VB_PredeclaredId = False
11889     Attribute VB_Exposed = False
11890     Option Explicit
11891     '
11892     '=====
11893     ' HashMap =
11894     '=====
11895     '
11896     ' - class for sharing a read/write memory-mapped file
11897     ' - backed by the Windows paging file rather than a specific
11898     ' - disk file between processes running under the same User account
11899     ' - on the same system.
11900     '
11901     ' - The Name values can optionally be prefixed "Global" or "Local"
11902     ' - (see the documentation) and the rest can consist of any
11903     ' - characters except the "\" character.
11904     '
11905     ' - After we obtain a handle to the object, we'll create a single
11906     ' - "view" containing the entire object as one BLOB.
11907     '
11908     ' - When all handles to the mapped object have been closed, it disappears.
11909     '
11910     Private Const API_NULL As Long = 0
11911     Private Const API_FALSE As Long = 0
11912     Private Const INVALID_HANDLE_VALUE As Long = -1
11913     Private Const PAGE_READWRITE As Long = 4
11914     Private Const SECTION_MAP_WRITE = &H2
11915     Private Const FILE_MAP_WRITE = SECTION_MAP_WRITE
11916     Private Const ERROR_ALREADY_EXISTS As Long = 183
11917
11918     Private Type SECURITY_ATTRIBUTES
11919         nLength As Long
11920         lpSecurityDescriptor As Long
11921         bInheritHandle As Long

```

```

11922 End Type
11923
11924 Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long
11925 Private Declare Sub CopyMemory _
11926     Lib "kernel32" _
11927     Alias "RtlMoveMemory" (ByVal Destination As Long, _
11928         ByVal Source As Long, _
11929         ByVal Length As Long)
11930 Private Declare Function RtlCompareMemory _
11931     Lib "ntdll" (ByRef Source1 As Any, _
11932         ByRef Source2 As Any, _
11933         ByVal Length As Long) As Long
11934
11935 Private Declare Function CreateFileMapping _
11936     Lib "kernel32" _
11937     Alias "CreateFileMappingA" (ByVal hFile As Long, _
11938         ByVal lpFileMappingAttributes As Long, _
11939         ByVal flProtect As Long, _
11940         ByVal dwMaximumSizeHigh As Long, _
11941         ByVal dwMaximumSizeLow As Long, _
11942         ByVal lpName As String) As Long
11943 Private Declare Function MapViewOfFile _
11944     Lib "kernel32" (ByVal hFileMappingObject As Long, _
11945         ByVal dwDesiredAccess As Long, _
11946         ByVal dwFileOffsetHigh As Long, _
11947         ByVal dwFileOffsetLow As Long, _
11948         ByVal dwNumberOfBytesToMap As Long) As Long
11949 Private Declare Function OpenFileMapping _
11950     Lib "kernel32" _
11951     Alias "OpenFileMappingA" (ByVal dwDesiredAccess As Long, _
11952         ByVal bInheritHandle As Long, _
11953         ByVal lpName As String) As Long
11954 Private Declare Function UnmapViewOfFile _
11955     Lib "kernel32" (ByVal lpBaseAddress As Long) As Long
11956 Private Declare Sub ZeroMemory2 _
11957     Lib "kernel32.dll" _
11958     Alias "RtlZeroMemory" (Destination As Any, _
11959         ByVal Length As Long)
11960 Private hObject As Long
11961 Private lpMap As Long
11962 Private mSize As Long
11963 Private Ptr As Long
11964 Private CheckData(128) As Byte, CheckDataPtr As Long
11965 Private VerifyArr(500) As Byte
11966
11967
11968 Public Sub CloseMap()
11969     UnmapViewOfFile lpMap
11970     CloseHandle hObject
11971     hObject = 0
11972 End Sub
11973
11974 Public Function CreateMap(ByVal Name As String, ByRef Size As Long) As Boolean
11975     'Returns True if the memory mapped file already exists. If so CAUTION,
11976     'the size will be its previously-created size.
11977     'Size may be reduced if not enough memory - and returned to caller !!!
11978     Dim i As Long, ErrCode As Long
11979     If Size < 1 Then Err.Raise 5, TypeName(Me), "Size must be at least 1 byte"
11980     For i = 1 To 10
11981         DoEvents
11982         Err.Clear
11983         hObject = CreateFileMapping(INVALID_HANDLE_VALUE, API_NULL, PAGE_READWRITE, 0, Size,
11984             Name)
11985         ErrCode = Err.LastDllError
11986         If hObject = API_NULL Then
11987             Err.Raise &H80049300, TypeName(Me), "CreateFileMapping system error " & CStr(
11988                 Err.LastDllError)
11989         End If

```

```

11988 CreateMap = (Err.LastDllError = ERROR_ALREADY_EXISTS)
11989
11990 DoEvents
11991 lpMap = MapViewOfFile(hObj, FILE_MAP_WRITE, 0, 0, 0)
11992 If lpMap = API_NULL Then
11993     Me.CloseMap
11994     If i = 10 Then
11995         Err.Raise &H80049302, TypeName(Me), "MapViewOfFile system error " & CStr(
11996             Err.LastDllError)
11997         Exit Function
11998     ElseIf i >= 3 Then 'try reducing size
11999         Size = (Size / 3) * 2
12000     End If
12001
12002 WriteTrace "***error> CreateMap: " & i & " / " & Name & " / Size= " & Size & "
12003 / Err:" & CStr(Err.LastDllError) & " / " & Now()
12004 'Err.Raise &H80049302, TypeName(Me), "MapViewOfFile system error " & CStr(Err.LastDllError)
12005 Sleep 100
12006 Else
12007     Exit For
12008 End If
12009 Next
12010 ZeroMemory2 ByVal lpMap, Size
12011 If bThreadTrace Then WriteTrace "---Creat map:ZERO_MAP / " & Now()
12012 mSize = Size
12013 End Function
12014
12015 Public Sub ClearMap(ByRef Size As Long)
12016     ZeroMemory2 ByVal lpMap, Size
12017     If bThreadTrace Then WriteTrace "--- Clear_MAP / " & Now()
12018 End Sub
12019
12020 Public Sub OpenMap(ByVal Name As String, ByVal Size As Long)
12021     Dim i As Long
12022     If Size < 1 Then Err.Raise 5, TypeName(Me), "Size must be at least 1 byte"
12023     DoEvents
12024     hObj = OpenFileMapping(FILE_MAP_WRITE, API_FALSE, Name)
12025     If hObj = API_NULL Then
12026         Err.Raise &H80049304, TypeName(Me), "OpenFileMapping system error " & CStr(
12027             Err.LastDllError)
12028     End If
12029     For i = 1 To 5
12030         DoEvents
12031         lpMap = MapViewOfFile(hObj, FILE_MAP_WRITE, 0, 0, Size)
12032         If lpMap = API_NULL Then
12033             Err.Raise &H80049306, TypeName(Me), "MapViewOfFile system error " & CStr(
12034                 Err.LastDllError)
12035         Else
12036             Exit For
12037         End If
12038     Next
12039     mSize = Size
12040     CheckDataPtr = VarPtr(CheckData(0))
12041 End Sub
12042
12043 Public Sub ReadMapHashCluster(ByVal Index As Long, _
12044                               ByVal lpData As Long, _
12045                               ByVal Size As Long)
12046     'Pass a pointer lpData and a length in bytes Size.
12047     Dim i As Long
12048     If Index * HashRecLen + Size > mSize Then Err.Raise 5, TypeName(Me), "Size must not
12049         exceed mapped size"
12050     If hObj = API_NULL Then Err.Raise &H80049308, TypeName(Me), "ReadMap: Map not open"
12051
12052     Ptr = lpMap + HashMapSearchPtr + Index * HashRecLen
12053     'If bHashTrace Then WriteTrace "ReadMapHashCluster: " & Index & "/" & Index & "Ptr:" & Ptr & "/ Nodes:" &
12054         Nodes & "/" & Now()

```

```

12050 For i = 1 To 3 'about 1 hash collision for 1.000.000.000 endgame nodes measured
12051 CopyMemory ByVal lpData, ByVal Ptr, ByVal Size
12052 If bHashVerify Then
12053 ' If bHashTrace Then WriteTrace "ReadMapHashCluster:Verify :" & VarPtr(VerifyArr(0)) & "Ptr:" & Ptr & Now()
12054 CopyMemory ByVal VarPtr(VerifyArr(0)), ByVal Ptr, ByVal Size
12055 ' If bHashTrace Then WriteTrace "ReadMapHashCluster: Compare " & VarPtr(VerifyArr(0)) & "Ptr:" & Ptr &
Now()
12056 If RtlCompareMemory(ByVal Ptr, ByVal VarPtr(VerifyArr(0)), ByVal Size) <> Size
Then
12057 ' Difference found => try again
12058 If bTraceHashCollision Then WriteTrace "HashMapDifference: Read " & Index &
"/" & i & "/" & Nodes:" & Nodes & " / " & Now()
12059 Else
12060 Exit For
12061 End If
12062 Else
12063 Exit For
12064 End If
12065 Next
12066 'If bHashTrace Then WriteTrace "ReadMapHashCluster:End "
12067 End Sub
12068
12069 Public Sub WriteMapHashEntry(ByVal ReplaceIndex As Long, ByVal lpData As Long)
12070 'Pass a pointer lpData and a length in bytes Size.
12071 Dim i As Long
12072 If (ReplaceIndex + 1) * HashRecLen > mSize Then Err.Raise 5, TypeName(Me), "Size
must not exceed mapped size"
12073 If hObj = API_NULL Then Err.Raise &H8004930A, TypeName(Me), "WriteMap: Map not open"
12074 Ptr = lpMap + HashMapSearchPtr + ReplaceIndex * HashRecLen
12075
12076 For i = 1 To 3 'about 1 hash collision for 1.000.000.000 endgame nodes measured
12077 CopyMemory ByVal Ptr, ByVal lpData, ByVal HashRecLen
12078 '--- Reread the written entry to verify that there was no parallel write from other thread that mixed up the data
12079 '--- Try max 3 times
12080 If RtlCompareMemory(ByVal Ptr, ByVal lpData, ByVal HashRecLen) <> HashRecLen Then
12081 ' Difference found => try again
12082 If bTraceHashCollision Then WriteTrace "HashMapDifference: Write " &
ReplaceIndex & "/" & i & "/" & Nodes:" & Nodes & " / " & Now()
12083 Else
12084 Exit For
12085 End If
12086 Next
12087
12088 End Sub
12089
12090 Public Sub WriteMapPos(ByVal StartPos As Long, ByVal lpData As Long, ByVal Size As
Long)
12091 'Pass a pointer lpData and a length in bytes Size.
12092 If StartPos + Size > mSize Then Err.Raise 5, TypeName(Me), "Size must not exceed
mapped size"
12093 If hObj = API_NULL Then Err.Raise &H8004930A, TypeName(Me), "WriteMap: Map not open"
12094 Ptr = lpMap + StartPos
12095 CopyMemory ByVal Ptr, ByVal lpData, ByVal Size
12096 End Sub
12097
12098 Public Sub ReadMapPos(ByVal StartPos As Long, ByVal lpData As Long, ByVal Size As Long
)
12099 'Pass a pointer lpData and a length in bytes Size.
12100 If StartPos + Size > mSize Then Err.Raise 5, TypeName(Me), "Size must not exceed
mapped size"
12101 If hObj = API_NULL Then Err.Raise &H80049308, TypeName(Me), "ReadMap: Map not open"
12102 Ptr = lpMap + StartPos
12103 CopyMemory ByVal lpData, ByVal Ptr, ByVal Size
12104 End Sub
12105
12106 Private Sub Class_Terminate()
12107 If hObj <> 0 Then CloseMap
12108 End Sub

```

```

12109 Attribute VB_Name = "basIO"
12110 '=====
12111 '= IOBas:
12112 '= Winboard / UCI communication / output of think results
12113 '=====
12114 Option Explicit
12115 '--- Win32 API functions
12116 Declare Function GetStdHandle Lib "kernel32" (ByVal nStdHandle As Long) As Long
12117 Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long
12118 Declare Function PeekNamedPipe _
12119     Lib "kernel32" (ByVal hNamedPipe As Long, _
12120         lpBuffer As Any, _
12121         ByVal nBufferSize As Long, _
12122         lpBytesRead As Long, _
12123         lpTotalBytesAvail As Long, _
12124         lpBytesLeftThisMessage As Long) As Long
12125 Declare Function ReadFile _
12126     Lib "kernel32" (ByVal hFile As Long, _
12127         lpBuffer As Any, _
12128         ByVal nNumberOfBytesToRead As Long, _
12129         lpNumberOfBytesRead As Long, _
12130         lpOverlapped As Any) As Long
12131 Declare Function WriteFile _
12132     Lib "kernel32" (ByVal hFile As Long, _
12133         ByVal lpBuffer As String, _
12134         ByVal nNumberOfBytesToWrite As Long, _
12135         lpNumberOfBytesWritten As Long, _
12136         lpOverlapped As Any) As Long
12137 Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
12138 Declare Function GetPrivateProfileString _
12139     Lib "kernel32" _
12140     Alias "GetPrivateProfileStringA" (ByVal lpApplicationName As String, _
12141         ByVal lpKeyName As Any, _
12142         ByVal lpDefault As String, _
12143         ByVal lpReturnedString As String, _
12144         ByVal nSize As Long, _
12145         ByVal lpFileName As String) As Long
12146 Declare Function WritePrivateProfileString _
12147     Lib "kernel32" _
12148     Alias "WritePrivateProfileStringA" (ByVal lpApplicationName As String, _
12149         ByVal lpKeyName As Any, _
12150         ByVal lpString As Any, _
12151         ByVal lpFileName As String) As Long
12152 Public Declare Sub ZeroMemory2 _
12153     Lib "kernel32.dll" _
12154     Alias "RtlZeroMemory" (Destination As Any, _
12155         ByVal Length As Long)
12156 Public hStdIn As Long ' Handle Standard Input
12157 Public hStdOut As Long ' Handle Standard Output
12158 Public Const STD_INPUT_HANDLE = -10&
12159 Public Const STD_OUTPUT_HANDLE = -11&
12160 Public psEnginePath As String ' path of engine directory (init different VB6 / Office)
12161 Public psDocumentPath As String ' path of office document
12162 Public pbIsOfficeMode As Boolean
12163 Public pllLastPostNodes As Long ' to avoid duplicate outputs
12164 Public EGTBasesEnabled As Boolean
12165 Public EGTBasesMaxPieces As Long ' 3,4,5,6 piece set
12166 Public EGTBasesMaxPly As Long ' max ply using EGTB in search
12167 Public EGTBasesPath As String ' SYZYGY EGTB files path
12168 Private oProxy As Object ' for online tablebases
12169 Public bEGTBBaseTrace As Boolean
12170 Public EGTBasesHitsCnt As Long ' count for GUI output
12171 Public EGTBRootProbeDone As Boolean
12172 Public EGTBRootResultScore As Long
12173 Public EGTBBestMoveStr As String, EGTBBestMoveListStr As String
12174 Public EGTBMoveListCnt(MAX_PV) As Long, EGTBMoveList(MAX_PV, 199) As String
12175 Public UCISyzygyPath As String
12176 Public UCISyzygyMaxPieceSet As Long

```

```

12177 Public UCISzygyMaxPly As Long
12178 '-----
12179 ' Log file
12180 '-----
12181 Public bLogPV As Boolean ' log PV in post mode
12182 Public bLogMode As Boolean
12183 Public LogFile As Long
12184 Public LastFullPV As String
12185 Public LanguageENArr(200) As String
12186 Public LanguageArr(200) As String
12187 Public LangCnt As Long
12188 Public psLanguage As String
12189
12190 '-----
12191 Public Sub OpenCommHandles()
12192 ' Open IO channels to Winboard
12193 hStdIn = GetStdHandle(STD_INPUT_HANDLE)
12194 hStdOut = GetStdHandle(STD_OUTPUT_HANDLE)
12195 End Sub
12196
12197 Public Sub CloseCommChannels()
12198 ' Close IO channels to Winboard
12199 CloseHandle hStdIn
12200 CloseHandle hStdOut
12201 If EGTBasesEnabled And Not DebugMode Then
12202 ' wait to avoid windows error when programs exits in AREAN after tablesbase access in Win7 ( ok for Win10)
12203 Dim i As Long
12204
12205 For i = 1 To 15
12206 Sleep 500
12207 DoEvents
12208 Next
12209
12210 End If
12211 End Sub
12212
12213 '-----
12214 'PollCommand() - check standard input
12215 '
12216 ' returns TRUE if data found
12217 '-----
12218 Function PollCommand() As Boolean
12219 If ThreadNum <= 0 Then
12220 #If DEBUG_MODE <> 0 Then
12221 ' from Debug form
12222 PollCommand = FakeInputState
12223 #Else
12224 ' winboard input
12225 Dim sBuff As String
12226 Dim lBytesRead As Long
12227 Dim lTotalBytes As Long
12228 Dim lAvailBytes As Long
12229 Dim rc As Long
12230
12231 sBuff = String(4096, Chr$(0))
12232 rc = PeekNamedPipe(hStdIn, ByVal sBuff, 4096, lBytesRead, lTotalBytes, lAvailBytes)
12233 PollCommand = CBool(rc And lBytesRead > 0)
12234 #End If
12235 Else
12236 '--- Multi-thread mode: helper threads get commands from main thread
12237
12238 MainThreadStatus = ReadMainThreadStatus()
12239
12240 'If bThreadTrace Then WriteTrace "PollCommand: ThreadStatusCheck:" & MainThreadStatus & " " &
LastThreadStatus & " / " & Now()
12241 Select Case MainThreadStatus
12242 Case 1

```

```

12243         If LastThreadStatus <> MainThreadStatus Then
12244             ThreadCommand = "go" & vbLf: PollCommand = True
12245             If bThreadTrace Then WriteTrace "PollCommand: MainThreadStatus = 1" & " / "
                & Now()
12246         End If
12247         Case 0
12248             If LastThreadStatus <> MainThreadStatus Then
12249                 ThreadCommand = "exit" & vbLf: PollCommand = True: bTimeExit = True
12250                 If bThreadTrace Then WriteTrace "PollCommand: MainThreadStatus = 0" & " / "
                    & Now()
12251             Else
12252                 Sleep 25
12253             End If
12254         End Select
12255
12256         LastThreadStatus = MainThreadStatus
12257     End If
12258 End Function
12259
12260 '-----
12261 'ReadCommand()
12262 '-----
12263 Function ReadCommand() As String
12264     If ThreadNum > 0 Then
12265         If bThreadTrace Then WriteTrace "ReadCommand: ThreadCommand = " & ThreadCommand &
            " / " & Now()
12266         ReadCommand = ThreadCommand
12267         ThreadCommand = ""
12268         Exit Function
12269     End If
12270     #If DEBUG_MODE <> 0 Then
12271         ReadCommand = FakeInput 'from Debug form
12272         FakeInputState = False
12273         FakeInput = ""
12274     #Else
12275         Dim sBuff As String
12276         Dim lBytesRead As Long
12277         Dim rc As Long
12278         sBuff = String$(4096, Chr$(0))
12279         rc = ReadFile(hStdIn, ByVal sBuff, 4096, lBytesRead, ByVal 0&)
12280         ReadCommand = Left$(sBuff, lBytesRead)
12281     #End If
12282 End Function
12283
12284 '-----
12285 'SendCommand()
12286 '-----
12287
12288 Function SendCommand(ByVal sCommand As String) As String
12289     Dim p As Long, s As String, sOut As String
12290     #If VBA_MODE = 1 Then
12291
12292         'OFFICE VBA
12293         With frmChessX
12294             If .txtIO.Visible Then
12295                 If Len(.txtIO) > 64000 Then .txtIO = ""
12296                 If .txtIO <> "" Then .txtIO = .txtIO & vbCrLf
12297                 If Len(sCommand) > 120 Then
12298                     s = sCommand: sOut = ""
12299                     Do While Len(s) > 120
12300                         p = InStrRev(Left$(s, 120), " ")
12301                         sOut = sOut & Left$(s, p)
12302                         s = Trim$(Mid$(s, p + 1))
12303                         If s <> "" Then sOut = sOut & vbCrLf & Space(14)
12304                     Loop
12305                     sOut = sOut & s
12306                     .txtIO = .txtIO & sOut
12307                 Else

```



```

12308         .txtIO = .txtIO & sCommand
12309     End If
12310     .txtIO.SetFocus
12311     .txtIO.SelStart = Len(.txtIO)
12312     .txtIO.SelLength = 0
12313     DoEvents
12314 End If
12315 End With
12316
12317 #End If
12318 #If DEBUG_MODE <> 0 Then
12319
12320     'VB DEBUG FORM
12321     With frmDebugMain
12322         If Len(.txtIO) > 32000 Then .txtIO = Left$(.txtIO, 8000)
12323         .txtIO = .txtIO & vbCrLf & sCommand
12324         .txtIO.SelStart = Len(.txtIO)
12325         .txtIO.SelLength = 0
12326         .Refresh
12327     End With
12328
12329 #End If
12330 #If DEBUG_MODE = 0 And VBA_MODE = 0 Then
12331     'WINBOARD STDOUT channel
12332     Dim lBytesWritten As Long
12333     Dim lBytes As Long
12334     Dim rc As Long
12335     sCommand = vbCrLf & sCommand & vbCrLf
12336     lBytes = Len(sCommand)
12337     rc = WriteFile(hStdOut, ByVal sCommand, lBytes, lBytesWritten, ByVal 0&)
12338 #End If
12339     SendCommand = sCommand
12340 End Function
12341
12342 Public Sub WriteGame(sFile As String)
12343     '--- Write file for game un UCI format
12344     '
12345     ' Format:
12346     '[Event "F/S Return Match"]
12347     '[Site "Belgrade, Serbia Yugoslavia|JUG"]
12348     '[Date "1992.11.04"]
12349     '[Round "29"]
12350     '[White "Fischer, Robert J."]
12351     '[Black "Spassky, Boris V."]
12352     '[Result "1/2-1/2"]
12353     ' 1. e2e4 e7e5 2. c2c4 f8e7 3. d2d4 e5d4 4. b1c3 d4c3
12354     Dim i As Long, h As Long, s As String, MoveCnt As Long, Cnt As Long
12355     Cnt = GameMovesCnt
12356     If Cnt = 0 Then Exit Sub
12357     s = "": MoveCnt = 0
12358
12359     For i = 1 To Cnt Step 2
12360         MoveCnt = MoveCnt + 1
12361         s = s & CStr(MoveCnt) & ". " & CompToCoord(arGameMoves(i))
12362         If i + 1 <= Cnt Then s = s & " " & CompToCoord(arGameMoves(i + 1)) & " "
12363     Next i
12364
12365     If s <> "" Then
12366         h = FreeFile()
12367         Open sFile For Append Lock Write As #h
12368         Print #h, "[Date " & Chr$(34) & Format(Now(), "YYYY.MM.DD HH:NN") & Chr$(34) & "]"
12369         Print #h, "[White " & Chr$(34) & "?" & Chr$(34) & "]"
12370         Print #h, "[Black " & Chr$(34) & "?" & Chr$(34) & "]"
12371         Print #h, "[Result " & Chr$(34) & "?" & Chr$(34) & "]"
12372         Print #h, s
12373         Close #h
12374     End If
12375 End Sub

```

```

12376
12377 Public Sub ReadGame(sFile As String)
12378     ' Read PGN File
12379     Dim h As Long, s As String, m As Long, sInp As String, m1 As String, m2
        As String
12380     Dim asMoveList() As String
12381     InitGame
12382     BookMovePossible = False
12383     bForceMode = True
12384     h = 10 'FreeFile()
12385     Open sFile For Input As #h
12386
12387     Do Until EOF(h)
12388         Line Input #h, sInp
12389         sInp = Trim(sInp) & " "
12390         If Left(sInp, 1) <> "[" Then '--- Ignore Header Tags
12391             asMoveList = Split(sInp, ".") ' split at move number dot
12392
12393             For m = 0 To UBound(asMoveList)
12394                 s = asMoveList(m)
12395                 s = Replace(s, "-", "")
12396                 s = Replace(s, "x", "")
12397                 s = Replace(s, "+", "")
12398                 s = Left(s, 10)
12399                 If Left(s, 1) = " " Then ' behind move number
12400                     s = Trim(s)
12401                     'Debug.Print s
12402                     m1 = Trim(Left(s, 4))
12403                     If Len(m1) = 4 Then
12404                         'Debug.Print m1, asMoveList(m)
12405                         ParseCommand m1 & vbLf
12406                     End If
12407                     If Len(s) > 8 Then
12408                         m2 = Trim$(Mid(s, 6, 4))
12409                         If Len(m2) >= 4 Then
12410                             'Debug.Print m2, asMoveList(m)
12411                             ParseCommand m2 & vbLf
12412                         End If
12413                     End If
12414                 End If
12415             Next
12416
12417         End If
12418     Loop
12419
12420     Close #h
12421 End Sub
12422
12423 Public Sub SendThinkInfo(Elapsed As Single, ActDepth As Long, CurrentScore As Long,
        Alpha As Long, Beta As Long)
12424     Static FinalMoveForHint As TMOVE
12425     Static sLastInfo As String
12426     Dim sPost As String, j As Long, sPostPV As String
12427     'pbIsOfficeMode = False 'Test
12428     If pbIsOfficeMode Then
12429         '--- MS OFFICE
12430         sPost = " " & Translate("Depth") & ":" & ActDepth & "/" & MaxPly & " " & Translate(
            "Score") & ":" & FormatScore(EvalSFTo100(CurrentScore)) & " " & Translate("Nodes"
            ) & ":" & Format("0.000", CalcNodes()) & " " & Translate("Sec") & ":" & Format(
            Elapsed, "0.00")
12431         If pLastPostNodes <> CalcNodes() Then
12432             SendCommand sPost
12433             pLastPostNodes = CalcNodes()
12434             sPostPV = " " & Translate("Line") & ":" "
12435
12436             For j = 1 To PVLength(1) - 1
12437                 sPostPV = sPostPV & " " & GUIMoveText(PV(1, j))
12438                 ' Save Hint move

```

```

12439     If j = 1 And Not MovesEqual(FinalMoveForHint, PV(1, 1)) Then HintMove =
EmptyMove ' for case that 1. ply as hash move only
12440     If j = 2 Then
12441         If PV(1, j).From > 0 Then HintMove = PV(1, j): FinalMoveForHint = PV(1, 1)
12442     End If
12443 Next
12444
12445     If sPost <> sLastInfo Then
12446         SendCommand sPostPV
12447         sLastInfo = sPost
12448         ShowMoveInfo MoveText(FinalMove), ActDepth, MaxPly, EvalSFTto100(CurrentScore),
Elapsed ' VBA mode only
12449     End If
12450 End If
12451 Else
12452     '--- VB6
12453     If UCIMode Then
12454         ' format: info depth 1 seldepth 1 multipv 1 score cp 417 nodes 51 nps 25500 tbhits 0 time 2 pv e8g8
12455         sPost = "info depth " & ActDepth & " seldepth " & MaxPly & " multipv 1 score " &
UciGUIScore(CurrentScore, Alpha, Beta)
12456         If Nodes > 1000 Then sPost = sPost & " hashfull " & HashUsageUCI()
12457         sPost = sPost & " nodes " & CalcNodes() & " nps " & CalcNPS(Elapsed) & " tbhits
" & EGTBasesHitsCnt & " time " & Int(Elapsed * 1000#) & " pv"
12458     Else
12459         sPost = ActDepth & " " & EvalSFTto100(CurrentScore) & " " & (Int(Elapsed) * 100)
& " " & CalcNodes()
12460     End If
12461
12462     sPostPV = ""
12463     For j = 1 To GetMax(1, PVLength(1) - 1)
12464         If PV(1, j).From <> 0 Then sPostPV = sPostPV & " " & GUIMoveText(PV(1, j))
12465     Next
12466
12467     Dim bLastFullPVUsed As Boolean
12468     bLastFullPVUsed = False
12469     If Len(Trim(sPostPV)) > 12 Then ' more than 2 moves
12470         If Len(Trim(sPostPV)) < Len(Trim>LastFullPV)) Then
12471             If Left(Trim>LastFullPV), Len(Trim(sPostPV))) = Trim(sPostPV) Then
12472                 sPostPV = LastFullPV
12473                 bLastFullPVUsed = True
12474             End If
12475         End If
12476         If Not bLastFullPVUsed Then
12477             LastFullPV = sPostPV
12478             LastFullPVLen = PVLength(1)
12479             For j = 1 To PVLength(1): SetMove LastFullPVArr(j), PV(1, j): Next
12480         End If
12481     Else
12482         If Left(Trim(sPostPV), 5) = Left(Trim>LastFullPV), 5) Then
12483             If Len(Trim(sPostPV)) < Len(Trim>LastFullPV)) Then
12484                 sPostPV = LastFullPV
12485             End If
12486         End If
12487     End If
12488     sPost = sPost & sPostPV
12489     If Not UCIMode And Not bWbPvInUciFormat Then sPost = sPost & "(" & MaxPly & "/" &
HashUsagePerc & ")"
12490     If Not GotExitCommand() Then
12491         If sPost <> sLastInfo Then
12492             SendCommand sPost
12493             sLastInfo = sPost
12494         End If
12495     End If
12496 End If
12497 End Sub
12498
12499 'Public Sub SendRootInfo(Elapsed As Single, ActDepth As Long, CurrentScore As Long, Alpha As Long, Beta As
Long)

```

```

12500 ' Dim sPost As String, j As Long, sPV As String
12501 ' 'CurrentScore = ScaleScoreByEGTB(CurrentScore)
12502 ' If pbIsOfficeMode Then
12503 ' '--- MS OFFICE
12504 ' sPost = " " & Translate("Depth") & ":" & ActDepth & "/" & MaxPly & " " & Translate("Score") & ":" &
FormatScore(EvalSFTTo100(CurrentScore)) & " " & Translate("Nodes") & ":" & Format("0.000", CalcNodes()) & " " &
Translate("Sec") & ":" & Format(Elapsed, "0.00")
12505 ' If plLastPostNodes <> Nodes Or Nodes = 0 Then
12506 ' SendCommand sPost
12507 ' plLastPostNodes = Nodes
12508 ' sPost = " >Line: "
12509 '
12510 ' For j = 1 To PVLength(1) - 1
12511 ' sPost = sPost & " " & MoveText(PV(1, j))
12512 ' Next
12513 '
12514 ' SendCommand sPost
12515 ' ShowMoveInfo MoveText(FinalMove), ActDepth, MaxPly, EvalSFTTo100(CurrentScore), Elapsed
12516 ' End If
12517 ' Else
12518 ' 'VB6
12519 ' If UCIMode Then
12520 ' ' format: info depth 1 seldepth 1 multipv 1 score cp 417 nodes 51 nps 25500 tbhits 0 time 2 pv e8g8
12521 ' sPost = "info depth " & ActDepth & " seldepth " & MaxPly & " multipv 1 score " & UciGUIScore(CurrentScore,
Alpha, Beta) & " nodes " & CalcNodes() & " nps " & CalcNPS(Elapsed) & " tbhits " & EGTBasesHitsCnt & " time " &
Int(Elapsed * 1000#) & " pv"
12522 ' Else
12523 ' sPost = ActDepth & " " & EvalSFTTo100(CurrentScore) & " " & (Int(Elapsed) * 100) & " " & CalcNodes()
12524 ' End If
12525 ' sPV = ""
12526 '
12527 ' For j = 1 To PVLength(1) - 1
12528 ' If PV(1, j).From <> 0 Then sPV = sPV & " " & GUIMoveText(PV(1, j))
12529 ' Next
12530 '
12531 ' If Len(Trim(sPV)) > 8 Then
12532 ' LastFullPV = sPV
12533 ' Else
12534 ' If Trim(Left(sPV, 5)) = Trim(Left(LastFullPV, 5)) Then
12535 ' sPV = LastFullPV
12536 ' End If
12537 ' End If
12538 ' sPost = sPost & sPV
12539 ' If Not GotExitCommand() Then
12540 ' SendCommand sPost
12541 ' End If
12542 ' End If
12543 ' If bWinboardTrace Then If bLogPV Then LogWrite Space(6) & sPost
12544 'End Sub
12545
12546 Public Function GotExitCommand() As Boolean
12547 Dim sInput As String
12548 GotExitCommand = False
12549 If PollCommand Then
12550 sInput = ReadCommand
12551 If Left$(sInput, 1) = "." Then
12552 SendAnalyzeInfo
12553 Else
12554 If sInput <> "" Then
12555 ParseCommand sInput
12556 GotExitCommand = bExitReceived
12557 End If
12558 End If
12559 End If
12560 End Function
12561
12562 Public Function FormatScore(ByVal lScore As Long) As String
12563 If lScore < -MATE_IN_MAX_PLY And lScore >= -MATE0 Then

```

```

12564     FormatScore = "-M" & CStr((Abs(MATE0) - Abs(lScore)) \ 2)
12565 ElseIf lScore > MATE_IN_MAX_PLY And lScore <= MATE0 Then
12566     FormatScore = "+M" & (MATE0 - lScore) \ 2
12567 ElseIf lScore = VALUE_NONE Then
12568     FormatScore = "?"
12569 Else
12570     FormatScore = Format$(lScore / 100#, "+0.00;-0.00")
12571 End If
12572 End Function
12573
12574 Public Sub SendAnalyzeInfo()
12575     Dim sPost As String, Elapsed As Single
12576     Elapsed = TimeElapsed
12577     sPost = "stat01: " & Int(Elapsed) & " " & CalcNodes() & " " & RootDepth & " " & "1
12578     If Not GotExitCommand() Then
12579         SendCommand sPost
12580     End If
12581 End Sub
12582
12583 Public Sub WriteTrace(s As String)
12584     Dim h As Long
12585     On Error Resume Next
12586     'Debug.Print s
12587     If s <> "" Then
12588         h = FreeFile()
12589         If ThreadNum <= 0 Then
12590             Open psEnginePath & "\Trace_" & Format(Date, "YYMMDD") & ".txt" For Append Lock
12591             Write As #h
12592         Else
12593             Open psEnginePath & "\Trace_" & Format(Date, "YYMMDD") & "_T" & Trim(CStr(GetMax
12594                 (0, ThreadNum))) & ".txt" For Append Lock Write As #h
12595         End If
12596         Print #h, s
12597         Close #h
12598     End If
12599     If pbIsOfficeMode Then SendCommand s
12600 End Sub
12601
12602 '-----
12603 'ReadINISetting: Read values fromm INI file
12604 '-----
12605 Function ReadINISetting(ByVal sSetting As String, ByVal sDefault As String) As String
12606     Dim sBuffer As String
12607     Dim lBufferLen As Long
12608     sBuffer = Space(260)
12609     lBufferLen = GetPrivateProfileString("Engine", sSetting, sDefault, sBuffer, 260,
12610         psEnginePath & "\" & INI_FILE)
12611     If lBufferLen > 0 Then
12612         ReadINISetting = Left$(sBuffer, lBufferLen)
12613     Else
12614         'LogWrite "Error retrieving setting: " & sSetting, True, True
12615     End If
12616 End Function
12617
12618 '-----
12619 ' WriteINISetting: write values to INI file
12620 '-----
12621 Function WriteINISetting(ByVal sSetting As String, ByVal sValue As String) As Boolean
12622     Dim lBufferLen As Long
12623     lBufferLen = WritePrivateProfileString("Engine", sSetting, sValue, psEnginePath &
12624         "\" & INI_FILE)
12625     If lBufferLen > 0 Then
12626         WriteINISetting = True
12627     Else
12628         LogWrite "Error writing setting: " & sSetting & "=" & sValue, True
12629         WriteINISetting = False
12630     End If

```

```

12627 End Function
12628
12629 '-----
12630 'LogWrite: Write log file
12631 'bTime adds the time
12632 '-----
12633 Public Sub LogWrite(sLogString As String, Optional ByVal BTime As Boolean)
12634     Dim sStr As String
12635     LogFile = FreeFile
12636     sStr = sLogString
12637     If BTime Then sStr = Now & " - " & sStr
12638     Open psEnginePath & "\" & LCase(psAppName) & ".log" For Append Lock Write As
12639     #LogFile
12640     Print #LogFile, sStr
12641     'Debug.Print sStr
12642     Close #LogFile
12643 End Sub
12644
12645 Public Sub ShowMoveInfo(ByVal sMove As String, _
12646     ByVal lDepth As Long, _
12647     ByVal lMaxPly As Long, _
12648     ByVal lScore As Long, _
12649     ByVal lTime As Single)
12650
12651     #If VBA_MODE Then
12652
12653         With frmChessX
12654             If InStr(sMove, "x") = 0 Then
12655                 .lblMove = Translate("Move") & ": " & UCase(Left$(sMove, 2)) & "-" & UCase$(
12656                     Mid$(sMove, 3))
12657             Else
12658                 .lblMove = Translate("Move") & ": " & UCase(Left$(sMove, 2)) & "x" & UCase$(
12659                     Mid$(sMove, 4))
12660             End If
12661             .lblDepth = Translate("Depth") & ": " & CStr(lDepth) & "/" & CStr(lMaxPly) & ":"
12662             & CStr(RootMoveCnt)
12663             .lblScore = Translate("Score") & " : " & FormatScore(lScore)
12664             .lblTime = Translate("Time") & ": " & Format(lTime, "0.00") & "s"
12665             DoEvents
12666         End With
12667     #End If
12668 End Sub
12669
12670 Public Function FieldNumToCoord(ByVal ilFieldNum As Long) As String
12671     FieldNumToCoord = Chr$(Asc("a") + ((ilFieldNum - 1) Mod 8)) & Chr$(Asc("1") + ((
12672         ilFieldNum - 1) \ 8))
12673 End Function
12674
12675 '
12676 '--- Translate functions ---
12677 '
12678 Public Sub ReadLangFile(ByVal isLanguage As String)
12679     '--- sample: isLanguage = "DE"
12680     Dim sLine As String
12681     Dim i As Long
12682     Dim sFile As String
12683     Dim f As Long
12684     Dim c As String
12685     Dim sTextEN As String
12686     Dim sText As String
12687     sFile = psEnginePath & "\ChessBrainVB_Language_" & isLanguage & ".txt"
12688     LangCnt = 0
12689     If Dir(sFile) <> "" Then
12690         f = FreeFile()
12691         Open sFile For Input As #f
12692
12693         Do While Not EOF(f)
12694             Line Input #f, sLine

```

```

12690     sLine = Trim$(sLine) 'Input
12691     If Not sLine = "" Then
12692         'Debug.Print sLine
12693         c = Left$(LTrim$(sLine), 1)
12694         If c <> ";" Then
12695             If StringSplit(sLine, sTextEN, sText) Then
12696                 LangCnt = LangCnt + 1
12697                 LanguageENArr(LangCnt) = sTextEN
12698                 LanguageArr(LangCnt) = sText
12699             End If
12700         End If
12701     End If
12702 Loop
12703
12704     Close #f
12705 End If ' File Exists
12706 End Sub
12707
12708 Public Sub InitTranslate()
12709
12710     If pbMSEExcelRunning Then
12711         psLanguage = "EN"
12712         #If VBA_MODE = 1 Then
12713             InitTranslateExcel
12714         #End If
12715     Else ' other VBA Office
12716         psLanguage = "EN"
12717         ReadLangFile "DE"
12718     End If
12719 End Sub
12720
12721 Public Function Translate(ByVal isTextEN As String) As String
12722     Dim i As Long
12723     If pbIsOfficeMode Then
12724
12725         For i = 1 To LangCnt
12726             If LanguageENArr(i) = isTextEN Then Translate = LanguageArr(i): Exit Function
12727         Next
12728
12729     End If
12730     Translate = isTextEN
12731 End Function
12732
12733 Public Function StringSplit(sInput As String, _
12734                             ByRef sTextEN As String, _
12735                             ByRef sText As String) As Boolean
12736     'Split String from Format "english#languageX#"
12737     Dim v As Variant
12738     v = Split(sInput, "#", -1, vbBinaryCompare)
12739     If Not UBound(v) = 2 Then
12740         StringSplit = False
12741         Exit Function
12742     End If
12743     sTextEN = v(0): sText = v(1): StringSplit = True
12744 End Function
12745
12746 Public Function InitTableBases() As Boolean
12747     On Error GoTo lblErr
12748     EGTBasesEnabled = CBool(Trim(ReadINISetting("EGTB_ENABLED", "0")) = "1") Or
12749     TableBasesRootEnabled
12750     If Not EGTBasesEnabled Then InitTableBases = False: Exit Function
12751     'pbIsOfficeMode = True 'TEST
12752     If pbIsOfficeMode Then ' for VBA-GUI only
12753         ' Online endgame tablebases
12754         ' see: https://github.com/lichess-org/lila-tablebase
12755         EGTBasesMaxPieces = 7
12756         EGTBasesMaxPly = 1
12757         InitTableBases = True

```



```

12757 Else
12758     'winboard / UCI mode: using SYZYGY endgame tablebases
12759     EGTBasesPath = Trim(ReadINISetting("TB_SYZYGY_PATH", psEnginePath))
12760     If UCIMode And Trim$(UCISyzygyPath) <> "" Then
12761         EGTBasesPath = UCISyzygyPath
12762     End If
12763     EGTBasesMaxPieces = Val("0" & ReadINISetting("TB_SYZYGY_MAX_PIECES", "0"))
12764     If UCIMode And UCISyzygyMaxPieceSet > 0 Then
12765         EGTBasesMaxPieces = UCISyzygyMaxPieceSet
12766     End If
12767     'probe for first x plies only
12768     EGTBasesMaxPly = Val("0" & ReadINISetting("TB_SYZYGY_MAX_PLY", "1")) 'ply 1=root
12769     InitTableBases = (EGTBasesMaxPieces > 2 And EGTBasesPath <> "")
12770     If UCIMode And UCISyzygyMaxPly > 0 Then
12771         EGTBasesMaxPly = UCISyzygyMaxPly
12772     End If
12773     If Trim$(EGTBasesPath) = "" Then EGTBasesEnabled = False: Exit Function
12774     '
12775     EGTBasesHitsCnt = 0
12776     If InitTableBases Then
12777         Dim ResultScore As Long, BestMove As String, MoveListStr As String, MoveCnt As
12778         Long
12779         InitTableBases = ProbeEGTB("8/8/8/3k4/5P2/5K2/8/8 b - - 0 1", ResultScore, True,
12780             BestMove, MoveListStr)
12781         If UCIMode Then
12782             If InitTableBases Then
12783                 SendCommand "info string tablebases found"
12784             Else
12785                 SendCommand "info string tablebases not found at:" & EGTBasesPath
12786             End If
12787         End If
12788         If bEGTbBaseTrace Then WriteTrace "InitTableBases: Path:" & EGTBasesPath & "
12789             PieceSet:" & EGTBasesMaxPieces & " > " & InitTableBases
12790     End If
12791     If bEGTbBaseTrace Then WriteTrace "Init endgame tablebase OK! "
12792 lblExit:
12793     Exit Function
12794 lblErr:
12795     If bEGTbBaseTrace Then WriteTrace "Init endgame tablebase:ERROR! "
12796     InitTableBases = False
12797     EGTBasesEnabled = False
12798     Resume lblExit
12799 End Function
12800
12801 Public Function IsTimeForEGTbBaseProbe() As Boolean
12802     If Not pbIsOfficeMode Then
12803         IsTimeForEGTbBaseProbe = False
12804         If FixedDepth <> NO_FIXED_DEPTH Then IsTimeForEGTbBaseProbe = True: Exit Function
12805         'If Ply < GetMax(3, RootDepth \ 3) Then
12806         If CBool(TimeLeft > 1.5) Then
12807             IsTimeForEGTbBaseProbe = True
12808         End If
12809         'End If
12810     Else
12811         ' max 20 sec for initial online TB call needed, expect refresh after 30 min pause
12812         IsTimeForEGTbBaseProbe = CBool(TimeLeft > 20 Or FixedDepth <> NO_FIXED_DEPTH)
12813     End If
12814     If bEGTbBaseTrace And Not IsTimeForEGTbBaseProbe Then WriteTrace "No time for
12815         endgame tablebase access: " & TimeLeft
12816 End Function
12817
12818 Public Function IsEGTbBasePosition() As Boolean
12819     Dim ActPieceCnt As Long
12820     ActPieceCnt = 2 + WNonPawnPieces + PieceCnt(WPAWN) + BNonPawnPieces + PieceCnt(BPAWN
12821 )
12822     IsEGTbBasePosition = CBool(ActPieceCnt <= EGTBasesMaxPieces)
12823 End Function

```

```

12820
12821 Public Sub TestTableBase()
12822     Dim sFEN As String, GameResultScore As Long, BestMove As String, BestMovesList As
        String
12823     Dim i As Long
12824     pbIsOfficeMode = True
12825     TableBasesRootEnabled = True
12826     InitTableBases
12827
12828     For i = 1 To 1
12829         If i Mod 2 = BCOL Then
12830             sFEN = "6k1/6p1/8/8/8/8/4P2P/6K1 b - -"
12831         Else
12832             sFEN = "7k/4P3/6K1/8/8/8/8/8 w - -"
12833             'sFEN = "R7/P4k2/8/8/8/r7/6K1 w - -"
12834         End If
12835         sFEN = "8/6k1/6p1/8/7r/3P1KP1/8/8 w - - 0 1"
12836         sFEN = "2Q1k3/8/4K3/8/8/3P4/8/8 b - - 0 11"
12837         sFEN = "8/k7/2K5/8/3P4/1Q6/8/8 b - - 0 11"
12838
12839         If ProbeTablebases(sFEN, GameResultScore, True, BestMove, BestMovesList) Then
12840             Debug.Print sFEN & " / Score: " & GameResultScore & " > " & BestMove & " / " &
                Left(BestMovesList, 80)
12841             DoEvents
12842         Else
12843             Debug.Print "Error"
12844         End If
12845     Next
12846
12847 End Sub
12848
12849 Public Function ProbeTablebases(ByVal sFEN As String, _
12850                                 ByRef GameResultScore As Long, _
12851                                 ByVal bShowBestMoves As Boolean, _
12852                                 ByRef BestMove As String, _
12853                                 ByRef BestMovesList As String) As Boolean
12854     If pbIsOfficeMode Then
12855         ProbeTablebases = ProbeOnlineEGTB(sFEN, GameResultScore, BestMove, BestMovesList)
12856     Else
12857         ProbeTablebases = ProbeEGTB(sFEN, GameResultScore, bShowBestMoves, BestMove,
            BestMovesList)
12858     End If
12859 End Function
12860
12861 'Public Function ProbeOnlineEGTB(ByVal sFEN As String, _
12862 '                                ByRef GameResultScore As Long, _
12863 '                                ByVal bShowBestMoves As Boolean, _
12864 '                                ByRef BestMove As String, _
12865 '                                ByRef BestMovesList As String) As Boolean
12866 ' ' Online Web Access needed !
12867 ' ' Documentation: http://www.lokasoft.nl/tbapi.aspx
12868 ' ' Comsvcs.dll needed
12869 ' ' function returns false if no result
12870 ' Static blnitDone As Boolean
12871 ' Static blnitOk As Boolean
12872 ' Dim sResult As String, sCommand As String
12873 ' GameResultScore = VALUE_NONE: BestMove = "": BestMovesList = "": ProbeOnlineEGTB = False
12874 ' If Not blnitDone Then
12875 '     blnitOk = InitTableBases()
12876 '     blnitDone = True
12877 ' End If
12878 ' If Not blnitOk Then ProbeOnlineEGTB = False: Exit Function
12879 ' On Error GoTo lblErr
12880 ' ' The score is given as distance to mat, or 0 when the position is a draw.
12881 ' ' An error response is returned when position is invalid or not in database. '
12882 ' ' e.g. M5 = color to move gives mate in 5 , -M3 = color to move gets mated in 5 moves.
12883 ' sCommand = "curl http://tablebase.lichess.ovh/standard/mainline?fen=4k3/8/8/8/4K3/8/8 w - -"
12884 ' sResult = GetCommandOutput(sCommand)

```

```

12885 '
12886 ' If Trim$(sResult) = "" Then Exit Function ' TODO Trim$(oProxy.ProbePosition(sFEN))
12887 ' If sResult = "0" Then
12888 '   GameResultScore = 0
12889 ' Elseif Left$(sResult, 1) = "M" Then
12890 '   GameResultScore = MATE0 - 2 * Val("0" & Mid$(sResult, 2))
12891 ' Elseif Left$(sResult, 2) = "-M" Then
12892 '   GameResultScore = -MATE0 + 2 * Val("0" & Mid$(sResult, 3))
12893 ' End If
12894 ' Shows list of best move
12895 ' If GameResultScore <> VALUE_NONE Then
12896 '   ProbeOnlineEGTB = True
12897 '   If bShowBestMoves Then
12898 '     BestMovesList = "" ' TODO
12899 '   End If
12900 ' End If
12901 ' If bEGTBBaseTrace Then WriteTrace "endgame tablebase move: " & BestMove & " / Score: " & GameResultScore
12902 ' & " " & Now() & vbCrLf & PrintPos()
12903 'lblExit:
12904 ' Exit Function
12905 'lblErr:
12906 ' blnitDone = False
12907 ' ProbeOnlineEGTB = False
12908 ' Resume lblExit
12909 'End Function
12910
12911 Public Function ExtractFirstTbMove(ByVal sMoveList As String) As String
12912     Dim sMove As String, p As Long, c As String
12913
12914     For p = 1 To Len(sMoveList)
12915         c = Mid$(sMoveList, p, 1)
12916         If (c >= "a" And c <= "h") Or (c >= "0" And c <= "9") Then
12917             If Len(sMove) <= 4 Then sMove = sMove & c
12918             ElseIf InStr("QRNB", c) > 0 Then
12919                 ' Promote piece
12920                 If Len(sMove) = 4 Then sMove = sMove & c
12921                 ElseIf c = " " Or c = Chr$(10) Then
12922                     Exit For
12923                 End If
12924             Next
12925
12926             If Len(sMove) = 4 Or Len(sMove) = 5 Then
12927                 ExtractFirstTbMove = sMove
12928             Else
12929                 ExtractFirstTbMove = ""
12930             End If
12931     End Function
12932
12933 Public Function ProbeEGTB(ByVal sFEN As String, _
12934     ByRef GameResultScore As Long, _
12935     ByVal bShowBestMoves As Boolean, _
12936     ByRef BestMove As String, _
12937     ByRef BestMovesListStr As String) As Boolean
12938 '
12939 '--- Use Fathom.exe to access Syzygy Endgame Tabelebases
12940 '--- Output string is parsed for result and bestmove
12941 '
12942 Dim sCommand As String, sRet As String, p As Long, p2 As Long, i As Long, sResult As
12943 String, sSearch As String, sOut As String, MoveList() As String, TmpMove As TMOVE,
12944 MoveCnt As Long, DTZ As Long
12945 GameResultScore = VALUE_NONE: BestMove = "": BestMovesListStr = "": ProbeEGTB =
12946 False: EGTBMoveListCnt(Ply) = 0: DTZ = 0
12947 On Error GoTo lblErr
12948 '
12949 '--- Call Fathom.exe and return output
12950 '
12951 sCommand = psEnginePath & "\Fathom.exe --path=" & Chr$(34) & EGTBasesPath & Chr$(34)
12952 & " " & Chr$(34) & sFEN & Chr$(34)

```

```

12948 sOut = GetCommandOutput(sCommand)
12949 If Trim$(sOut) = "" Then Exit Function
12950 sOut = Replace(sOut, Chr$(34), "") 'Remove "
12951 'search for DTZ (distance to zero for fifty counter): [DTZ 11]
12952 sRet = Trim$(sOut)
12953 p = InStr(sRet, "[DTZ")
12954 If p > 0 Then
12955     sRet = Mid$(sRet, p + Len("[DTZ") + 1)
12956     p = InStr(sRet, "]"): If p = 0 Then Exit Function
12957     sRet = Trim$(Left$(sRet, GetMax(p - 1, 0)))
12958     DTZ = Val("0" & Trim$(sRet))
12959 End If
12960 sRet = Trim$(sOut)
12961 'Debug.Print sOut
12962 'search for result: [WDL "Win"]
12963 p = InStr(sRet, "[WDL "): If p = 0 Then Exit Function
12964 sRet = Mid$(sRet, p + 5)
12965 p = InStr(sRet, "]"): If p = 0 Then Exit Function
12966 sResult = Left$(sRet, p - 1)
12967
12968 Select Case sResult
12969     Case "Win"
12970         sSearch = "[WinningMoves"
12971         GameResultScore = ScorePawn.EG * 20# - 3 * (Ply + DTZ): ProbeEGTB = True
12972     Case "Draw", "CursedWin", "BlessedLoss" 'CursedWin/BlessedLoss: 50 move draw avoids loss/win
12973         sSearch = "[DrawingMoves"
12974         GameResultScore = 0: ProbeEGTB = True
12975     Case "Loss"
12976         sSearch = "[LosingMoves"
12977         GameResultScore = -(ScorePawn.EG * 20# - 3 * (Ply + DTZ)): ProbeEGTB = True
12978     Case Else
12979         sSearch = "?????"
12980     Exit Function
12981 End Select
12982
12983 EGTBasesHitsCnt = EGTBasesHitsCnt + 1
12984 'search for moves: [WinningMoves "Rexd1, Re6, Rdx1, Rc3"]
12985 p = InStr(sRet, sSearch): If p = 0 Then Exit Function
12986 sRet = Mid$(sRet, p + Len(sSearch) + 1)
12987 p = InStr(sRet, "]"): If p = 0 Then Exit Function
12988 sRet = Trim$(Left$(sRet, GetMax(p - 1, 0)))
12989 Dim s As String, CaptureVal As Long, BestCaptureVal As Long, tmp As String
12990 If sRet <> "" Then
12991     'Convert best move to internal move (Rexd1 => e1d1), generate moves and find matching move
12992     MoveList = Split(sRet, " ")
12993     CaptureVal = -99999
12994
12995     For i = 0 To UBound(MoveList())
12996         s = Trim$(MoveList(i))
12997         If s <> "" And InStr(s, ".") = 0 Then 'ignore move cnt '1.'
12998             If InStr(s, "-") = 0 Then 'ignore result '1-0'
12999                 EGTBMoveListCnt(Ply) = EGTBMoveListCnt(Ply) + 1
13000                 EGTBMoveList(Ply, EGTBMoveListCnt(Ply)) = CompToCoord(GetMoveFromSAN(s))
13001                 If EGTBMoveListCnt(Ply) = 1 Then
13002                     BestMove = EGTBMoveList(Ply, 1)
13003                     'Debug.Print MoveText(BestMove)
13004                 End If
13005                 tmp = EGTBMoveList(Ply, EGTBMoveListCnt(Ply))
13006                 TmpMove = TextToMove(tmp)
13007                 If InStr(s, "x") > 0 Or Len(tmp) = 5 Then 'prefer captures/promotions
13008                     If Len(tmp) = 5 Then
13009                         CaptureVal = PieceAbsValue(TmpMove.Promoted) - PieceAbsValue(
13010                             TmpMove.Piece) 'promotion
13011                     Else
13012                         CaptureVal = GetSEE(TmpMove) 'try best capture
13013                     End If
13014                 Else
13015                     CaptureVal = (PsqVal(1, TmpMove.Piece, TmpMove.Target) - PsqVal(1,

```

```

        TmpMove.Piece, TmpMove.From))
13015 End If
13016 If CaptureVal > BestCaptureVal Then
13017     BestCaptureVal = CaptureVal
13018     BestMove = EGTBMoveList(Ply, EGTBMoveListCnt(Ply))
13019 End If
13020 'Debug.Print MoveCnt & ">:" & s
13021 End If
13022 End If
13023 Next
13024
13025 ' If sResult = "Loss" Then ' do not return move filter
13026 ' EGTBMoveListCnt = 0
13027 ' End If
13028 End If
13029 ' Find first move of best line " 1. d8=Q Kg4 2. Ke6 Kf4
13030 If bShowBestMoves Then
13031     BestMovesListStr = Mid$(sOut, InStrRev(sOut, "]") + 5) ' find last ] from [LosingMoves..]
13032 End If
13033 sRet = Trim$(Replace(BestMovesListStr, "...", ".")) & " " ' black to move : "1..."
13034 MoveCnt = 0
13035 MoveList = Split(sRet, " ")
13036
13037 For i = 0 To UBound(MoveList())
13038     s = Trim$(MoveList(i))
13039     If s <> "" And InStr(s, ".") = 0 Then ' ignore move cnt '1.'
13040         If InStr(s, "-") = 0 Then ' ignore result '1-0'
13041             MoveCnt = MoveCnt + 1
13042             ' If MoveCnt = 1 Then
13043             '     BestMove = CompToCoord(GetMoveFromSAN(s))
13044             ' Debug.Print MoveText(BestMove)
13045             ' End If
13046             ' Debug.Print MoveCnt & ">:" & s
13047         End If
13048     End If
13049 Next
13050
13051 'If MoveCnt > 0 Then
13052 ' Select Case sResult
13053 ' Case "Win"
13054 '     If BestCaptureVal > 150 Then MoveCnt = MoveCnt \ 2
13055 '     GameResultScore = ScorePawn.EG * 20# - 3 * MoveCnt
13056 ' Case "Loss"
13057 '     If BestCaptureVal > 150 Then MoveCnt = MoveCnt + 200 ' prefer good captures
13058 '     GameResultScore = -(ScorePawn.EG * 20# - 6 * MoveCnt)
13059 ' Case Else
13060 '     ' keep 0
13061 ' End Select
13062 'End If
13063 lblExit:
13064 Exit Function
13065 lblErr:
13066 ProbeEGTB = False
13067 Resume lblExit
13068 End Function
13069
13070 Public Function CalcNodes() As Long
13071 Dim TotalNodes As Double
13072 If NoOfThreads > 1 Then TotalNodes = Cdbl(NoOfThreads) * Cdbl(Nodes) Else TotalNodes
    = Nodes
13073 If TotalNodes > 2147483647# Then CalcNodes = 9999999 Else CalcNodes = TotalNodes
13074 End Function
13075
13076 Public Function CalcNPS(ByVal ElapsedTime As Single) As Long
13077 Dim TotalNodes As Double
13078 If NoOfThreads > 1 Then TotalNodes = Cdbl(NoOfThreads) * Cdbl(Nodes) Else TotalNodes
    = Nodes
13079 CalcNPS = Cdbl(TotalNodes) / GetMaxSingle(0.01, ElapsedTime)

```

```

13080 End Function
13081
13082 Public Function ScaleScoreByEGTB(Score As Long) As Long
13083     'If Ply > 1 Then Stop
13084     If EGTBRootResultScore = VALUE_NONE Or Abs(Score) > MATE_IN_MAX_PLY Or Ply > 1 Then
13085         ScaleScoreByEGTB = Score
13086     ElseIf EGTBRootResultScore > 0 Then
13087         ScaleScoreByEGTB = ScorePawn.EG * 20 + Score
13088     ElseIf EGTBRootResultScore < 0 Then
13089         ScaleScoreByEGTB = -ScorePawn.EG * 20 + Abs(Score)
13090     ElseIf EGTBRootResultScore = 0 Then
13091         ScaleScoreByEGTB = Score \ 10
13092     End If
13093 End Function
13094
13095 Public Function UciGUIScore(ByVal UciScore As Long, ByVal Alpha As Long, ByVal Beta As
Long) As String
13096     If UciScore <= -MATE_IN_MAX_PLY Then
13097         UciGUIScore = "mate -" & CStr((MATE0 - Abs(UciScore)) \ 2)
13098     ElseIf UciScore >= MATE_IN_MAX_PLY Then
13099         UciGUIScore = "mate " & CStr((MATE0 - UciScore) \ 2)
13100     Else
13101         UciGUIScore = "cp " & EvalSFTto100(UciScore)
13102         If UciScore <= Alpha Then
13103             UciGUIScore = UciGUIScore & " upperbound"
13104         ElseIf UciScore >= Beta Then
13105             UciGUIScore = UciGUIScore & " lowerbound"
13106         End If
13107     End If
13108 End Function
13109
13110 Public Function TestEGTB() As String
13111
13112     ' see: https://github.com/lichess-org/lila-tablebase
13113
13114     'Public Function TestEGTB(GameResult As enumEndOfGame) As String
13115     '--- curl http://tablebase.lichess.ovh/standard/mainline?fen=4k3/6KP/8/8/6r1/8/7p/8\_w\_-\_
13116     'curl http://tablebase.lichess.ovh/standard/mainline?fen=4k3/8/8/8/8/4K3/8/8\_w\_-\_
13117     '{"mainline":[],"winner":null,"dtz":0,"precise_dtz":0}'
13118     Dim sInp As String, i As Long, sWinner As String, sCommand As String
13119     Dim sTBMoves As String
13120     Dim EGTBArr() As String
13121     Dim GameResult As enumEndOfGame
13122
13123
13124     TestEGTB = ""
13125
13126     sCommand = "curl
http://tablebase.lichess.ovh/standard/mainline?fen=4k3/P7/8/8/8/4K3/p7/8\_w\_-\_
"
13127     sInp = GetCommandOutput(sCommand)
13128
13129
13130     If sInp = "too many pieces" Then GameResult = NO_MATE: Exit Function
13131
13132     'sInp =
13133     '{"mainline":[{"uci":"","g7h8","san":"","Kh8","dtz":3,"precise_dtz":3},{"uci":"","g4a4","san":"","Ra4","dtz":-2,"pr
13134     ecise_dtz":-2},{"uci":"","h8g7","san":"","Kg7","dtz":1,"precise_dtz":1},{"uci":"","h2h1q","san":"","h1=Q","dtz":-2,"
13135     "precise_dtz":-2},{"uci":"","g7f6","san":"","Kf6","dtz":1,"precise_dtz":1},{"uci":"","h1h7","san":"","Qxh7","dtz":-5,"
13136     "precise_dtz":-5},{"uci":"","f6e5","san":"","Ke5","dtz":4,"precise_dtz":4},{"uci":"","h7g6","san":"","Qg6","dtz":-3,"
13137     "precise_dtz":-3},{"uci":"","e5d5","san":"","Kd5","dtz":2,"precise_dtz":2},{"uci":"","g6d6","san":"","Qd6+","dtz":-1
13138     ","precise_dtz":-1},{"uci":"","d5d6","san":"","Kxd6","dtz":21,"precise_dtz":21},{"uci":"","a4a5","san":"","Ra5","dtz
13139     "":-20,"precise_dtz":-20},{"uci":"","d6e6","san":"","Ke6","dtz":19,"precise_dtz":19},{"uci":"","a5h5","san":"","Rh5"
13140     ","dtz":-18,"precise_dtz":-18},{"uci":"","e6d6","san":"","Kd6"
13141     "
13142     "
13143     'sInp = sInp & " "winner":"","b","dtz":-4,"precise_dtz":-4}'
13144     'sInp = ""mainline":[],"winner":null,"dtz":0,"precise_dtz":0}'
13145
13146
13147     EGTBArr() = Split(sInp, "uci":"","")

```

```

13138 sTBMoves = ""
13139 For i = 1 To UBound(EGTBArr)
13140     sTBMoves = sTBMoves & Left$(EGTBArr(i), 4) & " "
13141 Next i
13142 sTBMoves = Trim(sTBMoves)
13143
13144 i = InStr(sInp, "winner")
13145 sWinner = ""
13146 If i > 0 Then sWinner = Mid$(sInp, i + 9, 1) ' w =white, b =black, u =Null(Draw)
13147 Select Case sWinner
13148 Case "w"
13149     GameResult = WHITE_WON
13150 Case "b"
13151     GameResult = BLACK_WON
13152 Case "u"
13153     GameResult = DRAW_RESULT
13154 Case Else
13155     GameResult = NO_MATE
13156 End Select
13157
13158 ' Public Enum enumEndOfGame ' Game result
13159 ' NO_MATE = 0
13160 ' WHITE_WON = 1
13161 ' BLACK_WON = 2
13162 ' DRAW_RESULT = 3
13163 ' DRAW3REP_RESULT = 4
13164 ' End Enum
13165
13166 Debug.Print sTBMoves, sWinner
13167 End Function
13168
13169 Public Function ProbeOnlineEGTB(ByVal sFEN As String, _
13170     ByRef GameResultScore As Long, _
13171     ByRef BestMove As String, _
13172     ByRef BestMovesList As String) As Boolean
13173     ' Online Web Access needed ! Uses Windows program curl.exe (comes with Windows)
13174     ' Documentation: see: https://github.com/lichess-org/lila-tablebase
13175     ' sample call : curl.exe http://tablebase.lichess.ovh/standard/mainline?fen=4k3/6KP/8/8/6r1/8/7p/8\_w\_-\_-
13176     ' function returns false if no result
13177     ' sampel string returned:
13178     ' sResult =
13179     '{"mainline":[{"uci":"g7h8","san":"Kh8","dtz":3,"precise_dtz":3},{"uci":"g4a4","san":"Ra4","dtz":-2,"precise_dtz":-2},{"uci":"h8g7","san":"Kg7","dtz":1,"precise_dtz":1},{"uci":"h2h1q","san":"h1=Q","dtz":-2,"precise_dtz":-2},{"uci":"g7f6","san":"Kf6","dtz":1,"precise_dtz":1},{"uci":"h1h7","san":"Qxh7","dtz":-5,"precise_dtz":-5},{"uci":"f6e5","san":"Ke5","dtz":4,"precise_dtz":4},{"uci":"h7g6","san":"Qg6","dtz":-3,"precise_dtz":-3},{"uci":"e5d5","san":"Kd5","dtz":2,"precise_dtz":2},{"uci":"g6d6","san":"Qd6+","dtz":-1,"precise_dtz":-1},{"uci":"d5d6","san":"Kxd6","dtz":21,"precise_dtz":21},{"uci":"a4a5","san":"Ra5","dtz":-20,"precise_dtz":-20},{"uci":"d6e6","san":"Ke6","dtz":19,"precise_dtz":19},{"uci":"a5h5","san":"Rh5","dtz":-18,"precise_dtz":-18},{"uci":"e6d6","san":"Kd6"}]}'
13180     ' sInp = ""mainline":[],"winner":null,"dtz":0,"precise_dtz":0}'
13181     ' Test FEN/EPD: 8/8/1P6/5pr1/8/4R3/7k/2K5 w - -
13182
13183     Static bInitDone As Boolean
13184     Static bInitOk As Boolean
13185     Dim sResult As String, sCommand As String
13186     Dim i As Long, sWinner As String
13187     Dim lDTM As Long ' Distance to mate
13188     Dim sTBMove As String, bMate As Boolean
13189     Dim GameResult As enumEndOfGame
13190
13191     GameResultScore = VALUE_NONE: BestMove = "": BestMovesList = "": GameResult = NO_MATE: ProbeOnlineEGTB = False
13192     If Not bInitDone Then
13193         bInitOk = InitTableBases()
13194         bInitDone = True
13195     End If

```



```

13197 If Not bInitOk Then ProbeOnlineEGTB = False: Exit Function
13198 On Error GoTo lblErr
13199 ' The score is given as distance to mat, or 0 when the position is a draw.
13200 ' An error response is returned when position is invalid or not in database. '
13201 ' e.g. M5 = color to move gives mate in 5 , -M3 = color to move gets mated in 5 moves.
13202
13203 sCommand = "curl http://tablebase.lichess.ovh/standard?fen=" & Replace(sFEN, " ",
13204 " ")
13205 sResult = Trim(GetCommandOutput(sCommand))
13206
13207 ' sResult =
13208 ""checkmate"":false,""stalemate"":false,""variant_win"":false,""variant_loss"":false,""insufficient_material"":false,""dtz"":
13209 -4,""precise_dtz"":-4,""dtm"":-10,""category"":""loss"",""moves"":{""uci"":""g7h8"",""san"":""Kh8"",""zeroing"":false,""chec
13210 kmate"":false,""stalemate"":false,""variant_win"":false,""variant_loss"":false,""insufficient_material"":false,""dtz"":3,""pre
13211 cise_dtz"":3,""dtm"":9,""category"":""win""}"
13212
13213 'sCommand = "curl http://tablebase.lichess.ovh/standard/mainline?fen=" & Replace(sFEN, " ", " ")
13214 'sResult =
13215 "{"mainline":[{"uci":"","g7h8","san":"","Kh8","dtz":3,"precise_dtz":3},{"uci":"","g4a4","san":"","Ra4","dtz":-2,""
13216 precise_dtz":-2},{"uci":"","h8g7","san":"","Kg7","dtz":1,"precise_dtz":1},{"uci":"","h2h1q","san":"","h1=Q","dtz"
13217 ":-2,""precise_dtz":-2},{"uci":"","g7f6","san":"","Kf6","dtz":1,"precise_dtz":1},{"uci":"","h1h7","san":"","Qxh7",""
13218 dtz":-5,""precise_dtz":-5},{"uci":"","f6e5","san":"","Ke5","dtz":4,"precise_dtz":4},{"uci":"","h7g6","san":"","Qg6"
13219 ",""dtz":-3,""precise_dtz":-3},{"uci":"","e5d5","san":"","Kd5","dtz":2,"precise_dtz":2},{"uci":"","g6d6","san":"","Qd
13220 6+",""dtz":-1,""precise_dtz":-1},{"uci":"","d5d6","san":"","Kxd6","dtz":21,"precise_dtz":21},{"uci":"","a4a5","sa
13221 n":"","Ra5","dtz":-20,""precise_dtz":-20},{"uci":"","d6e6","san":"","Ke6","dtz":19,"precise_dtz":19},{"uci":"","a5
13222 h5","san":"","Rh5","dtz":-18,""precise_dtz":-18},{"uci":"","e6d6","san":"","Kd6"
13223 ""winner"":""b","dtz":-4,""precise_dtz":-4}"
13224
13225 ' more than 7 pieces?
13226 If sResult = "" Or sResult = "too many pieces" Then ProbeOnlineEGTB = False: Exit
13227 Function
13228
13229 '--- search for UCI moves in result string
13230 sResult = Replace(sResult, "","", " ")
13231
13232 bMate = False
13233 If InStr(Left$(sResult, 90), "checkmate :true") > 0 Then
13234     lDTM = 0
13235     bMate = True
13236 Else
13237     i = InStr(sResult, "uci :")
13238     sTBMove = Trim$(Mid$(sResult, i + 6, 5)) ' 5th character for promotion: qrbn
13239     sResult = Mid$(sResult, i)
13240     i = InStr(sResult, "dtm :")
13241     If Mid$(sResult, i + 5, 4) = "null" Then
13242         lDTM = -1 ' NO_MATE
13243     Else
13244         lDTM = Val(Trim$(Mid$(sResult, i + 5, 5)))
13245     End If
13246 End If
13247
13248 If lDTM < 0 Then
13249     If bWhiteToMove Then
13250         GameResult = WHITE_WON: GameResultScore = MATE0 - lDTM
13251     Else
13252         GameResult = BLACK_WON: GameResultScore = -MATE0 + lDTM
13253     End If
13254 ElseIf lDTM > 0 Then
13255     If Not bWhiteToMove Then
13256         GameResult = WHITE_WON: GameResultScore = MATE0 - Abs(lDTM)
13257     Else
13258         GameResult = BLACK_WON: GameResultScore = -MATE0 + Abs(lDTM)
13259     End If
13260 ElseIf lDTM = 0 Then
13261     If bMate Then ' Mate
13262         If bWhiteToMove Then
13263             GameResult = WHITE_WON: GameResultScore = MATE0

```

```

13250         Else
13251             GameResult = BLACK_WON: GameResultScore = -MATE0
13252         End If
13253     Else 'draw
13254         GameResult = DRAW_RESULT: GameResultScore = 0
13255     End If
13256 Else
13257     GameResult = NO_MATE
13258     GameResultScore = VALUE_NONE
13259 End If
13260
13261 ' Shows list of best move
13262 If GameResult <> NO_MATE Then
13263     ProbeOnlineEGTB = True
13264     BestMove = Trim$(Left$(sTBMove, 5))
13265     BestMovesList = BestMove
13266 End If
13267
13268 If bEGTbBaseTrace Then WriteTrace "endgame tablebase move: " & BestMove & " /
Score: " & GameResultScore & " " & Now() & vbCrLf & PrintPos()
13269 lblExit:
13270 Exit Function
13271 lblErr:
13272 bInitDone = False
13273 ProbeOnlineEGTB = False
13274 Resume lblExit
13275 End Function
13276
13277
13278
13279
13280
13281
13282
13283
13284 VERSION 5.00
13285 Begin VB.Form frmMain
13286     BorderStyle      = 3 'Fixed Dialog
13287     Caption          = "ChessBrain VB"
13288     ClientHeight      = 3435
13289     ClientLeft       = 2580
13290     ClientTop        = 1920
13291     ClientWidth      = 5250
13292     Icon             = "Main.frx":0000
13293     LinkTopic        = "Form1"
13294     MaxButton        = 0 'False
13295     MinButton        = 0 'False
13296     ScaleHeight      = 3435
13297     ScaleWidth       = 5250
13298     StartUpPosition = 2 'CenterScreen
13299     Begin VB.Label lblDescr
13300         AutoSize      = -1 'True
13301         BackStyle     = 0 'Transparent
13302         Caption       = "In option General->Commandline parameters please add
-xboard"
13303         BeginProperty Font
13304             Name      = "MS Sans Serif"
13305             Size      = 9.75
13306             Charset   = 0
13307             Weight    = 400
13308             Underline = 0 'False
13309             Italic    = 0 'False
13310             Strikethrough = 0 'False
13311         EndProperty
13312         Height       = 945
13313         Index       = 2
13314         Left        = 1800
13315         TabIndex    = 6

```

```

13316     ToolTipText      =    "GNU General Public License"
13317     Top                =    1920
13318     UseMnemonic        =    0    'False
13319     Width               =    2850
13320     WordWrap            =    -1    'True
13321 End
13322 Begin VB.Label lblDescr
13323     Alignment           =    1    'Right Justify
13324     AutoSize            =    -1    'True
13325     BackStyle           =    0    'Transparent
13326     Caption             =    "based on engines: LarsenVB (by Luca Dormio) and Faile (by
Adrien M. Regimbald) / Stockfish"
13327     Height              =    390
13328     Index               =    4
13329     Left                =    795
13330     TabIndex            =    5
13331     ToolTipText         =    "GNU General Public License"
13332     Top                 =    840
13333     UseMnemonic         =    0    'False
13334     Width               =    3525
13335     WordWrap            =    -1    'True
13336 End
13337 Begin VB.Label lblDescr
13338     BackStyle           =    0    'Transparent
13339     Caption             =    "ChessBrainVB 4.00 by Roger Zuehlisdorf 2023"
13340     BeginProperty Font
13341         Name              =    "MS Sans Serif"
13342         Size              =    13.5
13343         Charset           =    0
13344         Weight            =    400
13345         Underline         =    0    'False
13346         Italic            =    0    'False
13347         Strikethrough     =    0    'False
13348     EndProperty
13349     Height              =    855
13350     Index               =    0
13351     Left                =    960
13352     TabIndex            =    4
13353     Top                 =    120
13354     UseMnemonic         =    0    'False
13355     Width               =    3405
13356 End
13357 Begin VB.Label lblDescr
13358     AutoSize            =    -1    'True
13359     BackStyle           =    0    'Transparent
13360     Caption             =    "Please use a winboard chess GUI (i.e. ARENA)"
13361     BeginProperty Font
13362         Name              =    "MS Sans Serif"
13363         Size              =    9.75
13364         Charset           =    0
13365         Weight            =    400
13366         Underline         =    0    'False
13367         Italic            =    0    'False
13368         Strikethrough     =    0    'False
13369     EndProperty
13370     Height              =    480
13371     Index               =    3
13372     Left                =    1692
13373     TabIndex            =    3
13374     ToolTipText         =    "GNU General Public License"
13375     Top                 =    1332
13376     UseMnemonic         =    0    'False
13377     Width               =    2256
13378     WordWrap            =    -1    'True
13379 End
13380 Begin VB.Label lblCmd
13381     AutoSize            =    -1    'True
13382     BackStyle           =    0    'Transparent

```

```

13383         Caption           =      "Quit"
13384     BeginProperty Font
13385         Name                 =      "MS Sans Serif"
13386         Size                 =      12
13387         Charset              =      0
13388         Weight               =      400
13389         Underline            =      0 'False
13390         Italic               =      0 'False
13391         Strikethrough        =      0 'False
13392     EndProperty
13393     Height                  =      300
13394     Index                  =      2
13395     Left                   =      252
13396     MousePointer           =      99 'Custom
13397     TabIndex               =      2
13398     Top                   =      2700
13399     Width                  =      432
13400 End
13401 Begin VB.Label lblCmd
13402     AutoSize               =      -1 'True
13403     BackStyle              =      0 'Transparent
13404     Caption                =      "Play game "
13405     BeginProperty Font
13406         Name                 =      "MS Sans Serif"
13407         Size                 =      12
13408         Charset              =      0
13409         Weight               =      400
13410         Underline            =      0 'False
13411         Italic               =      0 'False
13412         Strikethrough        =      0 'False
13413     EndProperty
13414     Height                  =      300
13415     Index                  =      0
13416     Left                   =      216
13417     MousePointer           =      99 'Custom
13418     TabIndex               =      1
13419     Top                   =      1332
13420     Width                  =      1224
13421 End
13422 Begin VB.Label lblDescr
13423     Alignment              =      1 'Right Justify
13424     AutoSize               =      -1 'True
13425     BackStyle              =      0 'Transparent
13426     Caption                =      "Copyright: GNU GENERAL PUBLIC LICENSE V3"
13427     Height                 =      330
13428     Index                  =      1
13429     Left                   =      1440
13430     TabIndex               =      0
13431     ToolTipText            =      "GNU General Public License"
13432     Top                   =      3120
13433     UseMnemonic            =      0 'False
13434     Width                  =      3600
13435 End
13436 Begin VB.Image imgIco
13437     Height                 =      480
13438     Left                   =      105
13439     Top                   =      105
13440     Width                  =      480
13441 End
13442 Begin VB.Image imgPointer
13443     Height                 =      480
13444     Left                   =      735
13445     Picture                =      "Main.frx":0442
13446     Top                   =      105
13447     Visible                =      0 'False
13448     Width                  =      480
13449 End
13450 End

```

```

13451 Attribute VB_Name = "frmMain"
13452 Attribute VB_GlobalNameSpace = False
13453 Attribute VB_Creatable = False
13454 Attribute VB_PredeclaredId = True
13455 Attribute VB_Exposed = False
13456 '=====
13457 '= frmMain:
13458 '= Main form ( not shown under winboard)
13459 '=====
13460 Option Explicit
13461 Private sWBPath As String 'path winboard.exe
13462
13463 Private Function BrowseForFolders() As String
13464     BrowseForFolders = InputBox("Enter path of Winboard.exe (or edit INI file):")
13465     ' WinAPI removed to avoid problems with missing reference
13466     '###WIN32 sTitle = StrConv("Select location of winboard.exe (or use ARENA GUI) :", vbFromUnicode)
13467     '###WIN32 BInfo.hwndOwner = Me.hWnd
13468     '###WIN32 BInfo.lpszTitle = StrPtr(sTitle)
13469     '###WIN32 BInfo.ulFlags = BIF_RETURNONLYFSDIRS
13470     '###WIN32 lpIdList = SHBrowseForFolder(BInfo)
13471     '###WIN32 If lpIdList Then
13472         '###WIN32 sFolderName = String$(260, 0)
13473         '###WIN32 SHGetPathFromIDList lpIdList, sFolderName
13474         '###WIN32 sFolderName = Left$(sFolderName, InStr(sFolderName, Chr(0)) - 1)
13475         '###WIN32 CoTaskMemFree lpIdList
13476         '###WIN32 End If
13477         '###WIN32 BrowseForFolders = sFolderName
13478 End Function
13479
13480 '-----
13481 'GetCmdLine() - pass command line to ChessBrainVB
13482 '
13483 '-----
13484 Private Function GetCmdLine() As String
13485     ' GetCmdLine = "-cp -fcp ""ChessBrainVB -xboard"" -fd "" & psEnginePath & "" -scp ""ChessBrainVB -xboard""
13486     ' -sd "" & psEnginePath & """"
13487 End Function
13488
13489 Private Sub SetWBPath()
13490     sWBPath = ReadINISetting("WINBOARD", "")
13491     If sWBPath = "" Then
13492         sWBPath = BrowseForFolders
13493     Else
13494         On Local Error Resume Next
13495         If Dir$(sWBPath & "\winboard.exe") = "" Then
13496             sWBPath = BrowseForFolders
13497         End If
13498         On Local Error GoTo 0
13499     End If
13500 End Sub
13501
13502 Private Sub Form_Load()
13503     Dim i As Long
13504     imgIco.Picture = Me.Icon
13505     Set Me.Icon = Nothing
13506
13507     With App
13508         Me.Caption = Me.Caption & " ver. " & .Major & "." & Format(.Minor, "00") & "." &
13509         Format(.Revision, "0000")
13510         'lblDescr(1) = .LegalCopyright
13511     End With
13512
13513     For i = 0 To lblCmd.UBound
13514         lblCmd(i).MouseIcon = imgPointer.Picture
13515     Next
13516
13517     'lblDescr(0) = LoadResString(resMainTitle)
13518     'lblDescr(2) = LoadResString(resMainOption)

```

```

13517     lblCmd(0) = LoadResString(resMainPlay)
13518     lblCmd(1) = LoadResString(resMainBookEd)
13519     lblCmd(2) = LoadResString(resMainQuit)
13520 End Sub
13521
13522 Private Sub Form_MouseMove(Button As Integer, Shift As Integer, x As Single, y As
Single)
13523     Dim i As Long
13524
13525     For i = 0 To lblCmd.UBound
13526         lblCmd(i).Font.Underline = False
13527         lblCmd(i).Font.Bold = False
13528     Next
13529
13530 End Sub
13531
13532 Private Sub lblCmd_Click(Index As Integer)
13533     Dim sBookName As String
13534
13535     Select Case Index
13536     Case 0         'Winboard
13537         SetWBPath
13538         On Local Error GoTo CmdError:
13539         Shell sWBPath & "\winboard.exe" & GetCmdLine, vbNormalFocus
13540         WriteINISetting "WINBOARD", sWBPath
13541         End
13542
13543     Case 2         'Quit
13544         End
13545     End Select
13546
13547     On Local Error GoTo 0
13548     Exit Sub
13549 CmdError:
13550
13551     Select Case Err.Number
13552     Case 53
13553
13554         Select Case Index
13555         Case 0
13556             MsgBox "Cannot find Winboard", vbCritical
13557         Case 1
13558             MsgBox "Cannot find BookEdit", vbCritical
13559         End Select
13560     End Select
13561
13562     Screen.MousePointer = vbDefault
13563 End Sub
13564
13565 Private Sub lblCmd_MouseMove(Index As Integer, _
13566                               Button As Integer, _
13567                               Shift As Integer, _
13568                               x As Single, _
13569                               y As Single)
13570
13571     Dim i As Long
13572     Static LastIndex As Long
13573     If Index <> LastIndex Then
13574
13575         For i = 0 To lblCmd.UBound
13576             lblCmd(i).Font.Underline = False
13577             lblCmd(i).Font.Bold = False
13578         Next
13579
13580         LastIndex = Index
13581     End If
13582     lblCmd(Index).Font.Underline = True
13583     lblCmd(Index).Font.Bold = True
13584 End Sub

```

```

13584 Attribute VB_Name = "basProcess"
13585 '=====
13586 '= basProcess:
13587 '= start processes for multi core case
13588 '=====
13589
13590 Option Explicit
13591
13592 Private Type STARTUPINFO
13593     cb As Long
13594     lpReserved As String
13595     lpDesktop As String
13596     lpTitle As String
13597     dwX As Long
13598     dwY As Long
13599     dwXSize As Long
13600     dwYSize As Long
13601     dwXCountChars As Long
13602     dwYCountChars As Long
13603     dwFillAttribute As Long
13604     dwFlags As Long
13605     wShowWindow As Integer
13606     cbReserved2 As Integer
13607     lpReserved2 As Long
13608     hStdInput As Long
13609     hStdOutput As Long
13610     hStdError As Long
13611 End Type
13612
13613 Private Type PROCESS_INFORMATION
13614     hProcess As Long
13615     hThread As Long
13616     dwProcessID As Long
13617     dwThreadId As Long
13618 End Type
13619
13620 Const STARTF_USESHOWWINDOW = &H1&
13621 Const NORMAL_PRIORITY_CLASS = &H20&
13622 Const SW_HIDE = 3
13623
13624 Private Declare Function CreateProcess Lib "kernel32" Alias "CreateProcessA" (ByVal
lpApplicationName As String, ByVal lpCommandLine As String, lpProcessAttributes As Any
, lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As
Long, lpEnvironment As Any, ByVal lpCurrentDirectory As String, lpStartupInfo As
STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION) As Long
13625 Private Declare Function WaitForSingleObject Lib "kernel32" (ByVal hHandle As Long,
ByVal dwMilliseconds As Long) As Long
13626 Private Declare Function GetExitCodeProcess Lib "kernel32" (ByVal hProcess As Long,
lpExitCode As Long) As Long
13627 Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long
13628
13629 '--- Shells the passed command line and waits for the process to finish
13630 '--- Returns the exit code of the shelled process
13631 Function StartProcess(strCmdLine As String) As Long
13632     Dim udtProc As PROCESS_INFORMATION, udtStart As STARTUPINFO
13633
13634     'initialize the STARTUPINFO structure
13635     udtStart.cb = Len(udtStart) 'size
13636     udtStart.dwFlags = STARTF_USESHOWWINDOW 'uses show window command
13637     udtStart.wShowWindow = SW_HIDE 'the hide window command
13638
13639     'Launch the application
13640     CreateProcess vbNullString, strCmdLine, ByVal 0&, ByVal 0&, 0, NORMAL_PRIORITY_CLASS
, ByVal 0&, vbNullString, udtStart, udtProc
13641
13642 End Function
13643
13644

```



```

13645
13646 Attribute VB_Name = "basSearch"
13647 Option Explicit
13648 '=====
13649
13649 '= basSearch:                                     =
13650 '=                                                         =
13651 '= Search functions: Think->SearchRoot->Search->QSearch>Eval
13652 '= Think.....: Init search and call "SearchRoot" with increasing iterative depth 1,2,3... until time is over
13653 '= SearchRoot: create root moves at ply 1 and call "Search" starting with ply 2
13654 '= Search.....: search for best move by recursive calls to itself down to iterative depth or time is over
13655 '=         when iterative depth reached, calls QSearch
13656 '= QSearch....: quiescence search calculates all captures and checks (first QS-ply only) by recursive calls to itself
13657 '=         When all captures are done, the final position evaluation is returned
13658 '=====
13659
13660 Public Result As enumEndOfGame ' game result
13661 win/draw
13661 Public RootDepth As Long ' start search depth of root
13662 Public Nodes As Long ' counter for calls of SEARCH
13663 function
13663 Public QNodes As Long ' counter for calls of
13664 QSEARCH function
13664 Public QSDepthMax As Long ' max QS search depth
13665 reached
13665 Public EvalCnt As Long ' counter for calls of EVAL
13666 function
13666 Public RootDelta As Long ' delta of alpha beta at root
13667 Public bEndgame As Boolean ' switch for endgame logic
13668 Public PlyScore(MAX_DEPTH) As Long ' score for current search ply
13669 Public MaxPly As Long ' may ply reached in Search
13670 Public PV(MAX_PV, MAX_PV) As TMOVE '--- principal variation(PV):
13671 best path of moves in current search tree
13671 Public LastFullPVArr(MAX_PV) As TMOVE ' list of moves in search
13672 Public LastFullPVLen As Long
13673 Public PVLength(MAX_PV) As Long
13674 Private bSearchingPV As Boolean '--- often used for special
13675 handling (more exact search)
13675 Public HintMove As TMOVE ' user hint move for GUI
13676 Public MovesList(MAX_PV) As TMOVE '--- currently searched move
13677 path
13677 Public CntRootMoves As Long ' number of root moves: zero
13678 = draw
13678 Public PliesFromNull As Long '--- number of moves since
13679 last null move : for 3x draw detection
13679 Public FinalMove As TMOVE, FinalScore As Long
13680 '--- Final move selected
13680 Public PieceCntRoot As Long ' number of pieces on board
13681 at root
13681 Private bOnlyMove As Boolean ' direct response if only
13682 one move
13682 Private RootStartScore As Long ' Eval score at root from view
13683 of side to move
13683 Public PrevGameMoveScore As Long ' Eval score at root from view
13684 of side to move
13684 Private RootMatScore As Long ' Material score at root from
13685 view of side to move
13685 Public RootMoveCnt As Long ' current root move for GUI
13686 Public LastFinalScore As Long ' Final move score
13687 Public bFailedLowAtRoot As Boolean ' bad root move > needs
13688 more time
13688 Public DoubleExtensions(MAX_PV) As Long ' counts search extensions to
13689 avoid search explosion
13689 Public CutOffCnt(MAX_PV) As Long ' cutoff
13690 Public ttPVArr(MAX_PV) As Boolean
13691
13692 '--- Search performance: move ordering, cuts of search tree ---

```

```

13693 Public History(COL_WHITE, MAX_BOARD, MAX_BOARD) As Long ' move history From square
-> To square for color
13694 Public CaptureHistory(BEP_PIECE, MAX_BOARD, BEP_PIECE) As Long ' capture history moving
piece -> To square > captured Piece type
13695 Public StatScore(MAX_PV + 3) As Long ' statistics score per search
ply
13696 Public CounterMove(15, MAX_BOARD) As TMOVE ' Good move against
previous move
13697 Public ContinuationHistory(15 * MAX_BOARD, 15 * MAX_BOARD) As Integer ' statistics for follow
up moves ; Integer for less memory
13698 Public CmhPtr(MAX_PV) As Long ' Pointer to first move of
ContinuationHistory

13699
13700 Public Type TKiller
13701     Killer1 As TMOVE 'killer moves: good moves for better move ordering
13702     Killer2 As TMOVE
13703     Killer3 As TMOVE
13704 End Type

13705
13706 Public Killer(MAX_PV) As TKiller
13707 Public Killer0 As TKiller
13708 Public Killer2 As TKiller
13709 Public Reductions(63) As Long '[moveNumber]
13710 Public BestMovePlay(MAX_PV) As TMOVE
13711 Public EmptyMove As TMOVE
13712 Public CaptPruneMargin(6) As Long

13713
13714 '--- piece bit constants for attack arrays, used for evaluation
13715 Public Const PLAttackBit As Long = 1 ' Pawn attack to left side (from white view)
13716 Public Const PRAttackBit As Long = 2 ' Pawn attack to right side (from white view) (to count multiple
pawn attacks)

13717 Public Const N1AttackBit As Long = 4 ' for 1. knight
13718 Public Const N2AttackBit As Long = 8 ' for 2. knight
13719 Public Const B1AttackBit As Long = 16
13720 Public Const B2AttackBit As Long = 32
13721 Public Const R1AttackBit As Long = 64
13722 Public Const R2AttackBit As Long = 128
13723 Public Const QAttackBit As Long = 256
13724 Public Const KAttackBit As Long = 512
13725 Public Const BXrayAttackBit As Long = 1024 ' Xray attack through own bishop/queen, one xray enough
because different square colors
13726 Public Const R1XrayAttackBit As Long = 2048 ' Xray attack through own rook/queen
13727 Public Const R2XrayAttackBit As Long = 4096 ' to count multiple rook attacks, not needed for bishop
and queens (promotion needed)
13728 Public Const QXrayAttackBit As Long = 8192 ' Xray attack through own bishop/rook/queen
13729 '--- combined attack bits
13730 Public Const PAttackBit As Long = PLAttackBit Or PRAttackBit
13731 Public Const NAttackBit As Long = N1AttackBit Or N2AttackBit
13732 Public Const BAttackBit As Long = B1AttackBit Or B2AttackBit
13733 Public Const BOrXrayAttackBit As Long = B1AttackBit Or B2AttackBit Or BXrayAttackBit
13734 Public Const RAttackBit As Long = R1AttackBit Or R2AttackBit
13735 Public Const R1OrXrayAttackBit As Long = R1AttackBit Or R1XrayAttackBit
13736 Public Const R2OrXrayAttackBit As Long = R2AttackBit Or R2XrayAttackBit
13737 Public Const ROrXrayAttackBit As Long = R1AttackBit Or R2AttackBit Or R1XrayAttackBit
Or R2XrayAttackBit
13738 Public Const PBNAttackBit As Long = PAttackBit Or NAttackBit Or BAttackBit
13739 Public Const RBAttackBit As Long = RAttackBit Or BAttackBit
13740 Public Const RBOOrXrayAttackBit As Long = ROrXrayAttackBit Or BOrXrayAttackBit
13741 Public Const QOrXrayAttackBit As Long = QAttackBit Or QXrayAttackBit
13742 Public Const QOrXrayROrXrayAttackBit As Long = QOrXrayAttackBit Or ROrXrayAttackBit
13743 Public Const QBAttackBit As Long = QAttackBit Or BAttackBit
13744 Public Const QRAttackBit As Long = QAttackBit Or RAttackBit
13745 Public Const QRBAttackBit As Long = QAttackBit Or RAttackBit Or BAttackBit ' slider
attacks, detect pinned pieces
13746 Public Const QRBOrXrayAttackBit As Long = QAttackBit Or QXrayAttackBit Or
ROrXrayAttackBit Or BOrXrayAttackBit ' slider attacks, detect pinned pieces
13747 Public Const QRBNAttackBit As Long = QAttackBit Or RAttackBit Or BAttackBit Or
NAttackBit

```

```

13748 Public Const PNBRAAttackBit As Long = PAttackBit Or NAttackBit Or BAttackBit Or
RAttackBit
13749 '---
13750 Public AttackBitCnt(QXrayAttackBit * 2) As Long ' Returns number of attack bits set
13751 Public EasyMove As TMOVE
13752 Public EasyMovePV(3) As TMOVE
13753 Public EasyMoveStableCnt As Long
13754 Public bEasyMovePlayed As Boolean
13755 Public QSDepth As Long
13756 Private TmpMove As TMOVE
13757 Public bFirstRootMove As Boolean
13758 'Public bEvalBench As Boolean
13759 Public LegalRootMovesOutOfCheck As Long
13760 Public IsTBScore As Boolean
13761 Public SkipSize(20) As Long 'multi core search: sizes and phases of the skip-blocks, used
for distributing search depths across the threads
13762 Public SkipPhase(20) As Long
13763 Public DepthInWork As Long ' multi core search: For decision if better thread
13764
13765 Public FinalCompletedDepth As Long ' root depth completed
13766 Private NullMovePly As Long ' search depth for null move verification
13767
13768 Public TableBasesRootEnabled As Boolean
13769 Public TableBasesSearchEnabled As Boolean
13770
13771 '--- end if declarations -----
13772 '-----
13773 'StartEngine: starts the chess engine to return a move
13774 '-----
13775 Public Sub StartEngine ()
13776 Dim CompMove As TMOVE
13777 Dim sCoordMove As String
13778 Dim bOldEvalTrace As Boolean
13779 Dim i As Long
13780 '--- in winboard FORCE mode return, also check side to move
13781 'Debug.Print bComplsWhite, bWhiteToMove, bForceMode, Result
13782 If bAnalyzeMode Then bCompIsWhite = bWhiteToMove
13783
13784 '--- for main loop exit here if opponent has to move
13785 If bCompIsWhite <> bWhiteToMove Or bForceMode Or Result <> NO_MATE Then Exit Sub
13786
13787 '--- for single core is ThreadNum=-1, fot multi core main thread is ThreadNum=0, core 2 is ThreadNum=1
13788 If NoOfThreads > 1 And ThreadNum = 0 Then
13789 InitThreads
13790 End If
13791
13792 '--- Init Search data
13793 QNodes = 0
13794 QSDepthMax = 0
13795 Nodes = 0
13796 Ply = 1
13797 Result = NO_MATE
13798 TimeStart = Timer
13799 bOldEvalTrace = bEvalTrace
13800
13801 ' If DebugMode And ThreadNum = 0 Then
13802 ' DEBUGReadGame "bug001game.txt"
13803 ' FixedTime = 30
13804 ' End If
13805
13806 '--- Multi core search: init thread data
13807 If ThreadNum = 0 Then
13808 If bThreadTrace Then WriteTrace "StartEngine: WriteMainThreadStatus 1 " & " / " &
Now()
ClearMapBestPVforThread
WriteMapGameData
MainThreadStatus = 1: WriteMainThreadStatus 1 ' start helper threads
ElseIf ThreadNum > 0 Then

```

[illegible]

```

13880         End If
13881     Else
13882         If UCIMode Then
13883             SendCommand "bestmove (none)" ' ??? try same as Stockfish
13884         End If
13885     End If
13886 Case DRAW3REP_RESULT
13887     'Draw?
13888     PlayMove CompMove
13889     GameMovesAdd CompMove
13890     If UCIMode Then
13891         SendCommand "bestmove" & " " & sCoordMove
13892     Else
13893         SendCommand Translate("move") & " " & sCoordMove
13894         SendCommand "1/2-1/2 {" & Translate("Draw by repetition") & "}"
13895     End If
13896 Case DRAW_RESULT:
13897     If UCIMode Then
13898         SendCommand "bestmove (none)"
13899     Else
13900         SendCommand "1/2-1/2 {" & Translate("Draw no move") & "}"
13901     End If
13902 Case Else
13903     '
13904     '--- Send move to GUI
13905     '
13906     If CompMove.From <> 0 Then
13907         PlayMove CompMove
13908         GameMovesAdd CompMove
13909         If UCIMode Then
13910             SendCommand "bestmove" & " " & sCoordMove
13911         Else
13912             SendCommand Translate("move") & " " & sCoordMove
13913         End If
13914         '--- Draw?
13915         If Fifty >= 100 Then
13916             SendCommand "1/2-1/2 {" & Translate("50 Move Rule") & "}"
13917         Else '--- no move
13918             SendCommand "1/2-1/2 {" & Translate("Draw") & "}"
13919         End If
13920     End If
13921 End Select
13922
13923 'WriteTrace "move: " & CompMove & vbCrLf ' & "(t:" & Format(SearchTime, "###0.00") & " s:" & FinalScore ' & "
13924 'n:" & Nodes & " qn:" & QNodes & " q%:" & "%)"
13925 End Sub
13926
13927 '=====
13928 ' THINK: Start of Search with iterative deepening      =
13929 '   aspiration windows used in 3 steps                  =
13930 '   called by: STARTENGINE, calls: SEARCH              =
13931 '=====
13932 Public Function Think() As TMOVE
13933     Dim Elapsed As Single
13934     Dim CompMove As TMOVE, LastMove As TMOVE
13935     Dim IMax As Long, i As Long, j As Long, k As Long
13936     Dim BoardTmp(MAX_BOARD) As Long
13937     Dim GoodMoves As Long
13938     Dim RootAlpha As Long
13939     Dim RootBeta As Long
13940     Dim OldScore As Long, Delta As Long
13941     Dim bOldEvalTrace As Boolean
13942     Dim Hashkey As THashKey
13943     Dim AdjustedDepth As Long, FailedHighCnt As Long
13944
13945     '--- Thread management
13946     Dim bHelperMove As Boolean, HelperCompletedDepth As Long, HelperBestScore As

```

```

13947     Long, HelperNodes As Long, HelperPvLength As Long, HelperPV(11) As TMOVE
13948 '-----
13948 ClearMove CompMove
13949 ResetMaterial
13950 'init counters
13951 Nodes = 0
13952 QNodes = 0
13953 EvalCnt = 0
13954 HashUsage = 0
13955 HashAccessCnt = 0
13956 InitEval
13957 bEvalTrace = bEvalTrace Or CBool(ReadINISetting("EVALTRACE", "0") <> "0") 'after InitEval
13958 bOldEvalTrace = bEvalTrace
13959 MaxPly = 0
13960 EGTBasesHitsCnt = 0
13961 LastNodesCnt = 0: RootMoveCnt = 0: LastThreadCheckNodesCnt = 0
13962 plLastPostNodes = 0: IsTBScore = False
13963 NextHashGeneration 'set next generation for hash entries
13964 LastFullPV = ""
13965 Erase LastFullPVArr: LastFullPVLen = 0
13966 'HashFoundFromOtherThread = 0
13967 FinalCompletedDepth = 0: DepthInWork = 0
13968 'init easy move
13969 EasyMove = GetEasyMove() 'get easy move from previous Think call
13970 If bTimeTrace Then WriteTrace "Think: Easymove: " & MoveText(EasyMove) & " " & Now()
13971 ClearEasyMove
13972 bEasyMovePlayed = False
13973 BestMoveChanges = 0
13974 SetMove FinalMove, EmptyMove
13975
13976 'Tracing
13977 bTimeTrace = CBool(ReadINISetting("TIMETRACE", "0") <> "0")
13978 If bTimeTrace Then
13979     WriteTrace " "
13980     WriteTrace "----- Start thinking, GAME MOVE >>>: " & GameMovesCnt \ 2 & " <<<"
13981 ElseIf bLogPV Then
13982     If bWinboardTrace Then LogWrite Space(6) & "----- Start thinking, GAME MOVE >>>: "
13983     & GameMovesCnt \ 2 & " <<<"
13984 End If
13985
13986 'reset move lists
13987 For i = 0 To 99: PlyScore(i) = 0: MovesList(i).From = 0: MovesList(i).Target = 0:
13988 Next i
13989
13990 'reset debug counter
13991 For i = 0 To 20: TestCnt(i) = 0: Next
13992
13993 bTimeExit = False '--- Used for stop search, currently searched line result is not valid!!
13994
13995 '=====
13996 '= Opening book move ? =
13997 '=====
13998 If BookMovePossible Then
13999     CompMove = ChooseBookMove
14000     If CompMove.From <> 0 Then
14001         FinalScore = 0
14002         If UCIMode Then
14003             SendCommand "info string book move: " & CompToCoord(CompMove)
14004         Else
14005             SendCommand "0 0 0 0 (Book Move)"
14006         End If
14007         Think = CompMove
14008         Exit Function '<<<< EXIT with book move
14009     End If
14010     BookMovePossible = False
14011 End If

```

```

14012 '--- Init search scores ---
14013 FinalScore = -MATE0
14014 RootStartScore = Eval() ' Output for EvalTrace, sets EvalTrace=false
14015 If bOldEvalTrace Then ClearMove Think: Exit Function ' Exit if we only want an EVAL trace
14016 'LogWrite "Start Think "
14017
14018 '--- Init timer ---
14019 TimeStart = Timer
14020 AllocateTime
14021 'Debug.Print "OptTime=" & OptimalTime & ", MaxTime=" & MaximumTime
14022
14023 '--- init hash map for multi core search
14024 If ThreadNum > 0 Then InitHash ' check new hash size
14025
14026 HashBoard Hashkey, EmptyMove
14027 InHashCnt = 0
14028 IMax = MAX_DEPTH
14029 If bThreadTrace Then WriteTrace "Think: Threadnum=" & ThreadNum & " " & Now() &
vbCrLf & " start board=" & vbCrLf & PrintPos
14030 If ThreadNum > 0 Then WriteHelperThreadStatus ThreadNum, 1
14031
14032 ' copy current board before start of search to restore it later
14033 CopyIntArr Board, BoardTmp
14034
14035 ' - not better ?
14036 '--- Init search data--
14037 " Erase History()
14038 " Erase ContinuationHistory()
14039 '--- Rescale history ???? not better, same results with 32, 64, 128
14040 ' For j = SQ_A1 To SQ_H8
14041 ' For k = SQ_A1 To SQ_H8
14042 ' For i = COL_WHITE To COL_BLACK
14043 ' History(i, j, k) = History(i, j, k) \ 32
14044 ' Next
14045 ' ContinuationHistory(i, j) = ContinuationHistory(j, k) \ 32
14046 ' Next
14047 ' Next
14048 'Erase CounterMove()
14049
14050 '==> Keep old data in History arrays!
14051 Erase Killer()
14052 Erase PV()
14053 If ThreadNum > 0 Then WriteMapBestPVforThread 0, VALUE_NONE, EmptyMove
14054 Erase MovesList()
14055 CntRootMoves = 0
14056 LastChangeMove = ""
14057 FinalScore = -VALUE_INFINITE
14058 Result = NO_MATE
14059
14060 EGTBMoveListCnt(1) = 0: EGTBRootResultScore = VALUE_NONE: EGTBRootProbeDone =
False
14061
14062 '-----
14063 '--- Iterative deepening ----
14064 '-----
14065 For RootDepth = 1 To IMax
14066
14067 '--- Distribute search depths across the threads
14068 If ThreadNum > 0 Then
14069 Dim th As Long
14070 th = (ThreadNum - 1) Mod 20
14071 If ((RootDepth + SkipPhase(th)) / GetMax(1, SkipSize(th))) Mod 2 <> 0 And
RootDepth > 1 Then
14072 If RootDepth > 1 Then PlyScore(RootDepth) = PlyScore(RootDepth - 1)
14073 GoTo lblNextRootDepth
14074 Else
14075 If bThreadTrace Then WriteTrace "Think: RootDepth=" & RootDepth & " / " &
Now()

```



[illegible]

```

14141 ' SendCommand "D:" & RootDepth & ">>> Search A:" & RootAlpha & ", B:" & RootBeta & " => SC: " &
FinalScore
14142 ' End If
14143 #End If
14144 Debug.Assert Abs(FinalScore) <= Abs(VALUE_NONE)
14145 Debug.Assert Abs(RootAlpha) <= Abs(VALUE_NONE)
14146 Debug.Assert Abs(RootBeta) <= Abs(VALUE_NONE)
14147
14148 '--LastMove.From = 0 no move => draw
14149 If bTimeExit Or IsTBScore Or LastMove.From = 0 Or (bOnlyMove And RootDepth = 1
) Then Exit Do
14150
14151 '
14152 '-- Research: if no move found in Alpha-Beta window
14153 '
14154 bSearchingPV = True: GoodMoves = 0
14155
14156 ' GUI info
14157 If (RootDepth > 1 Or IsTBScore) And bPostMode And PVLength(1) >= 1 Then
Elapsed = TimeElapsed()
14158 If Not bExitReceived Then SendThinkInfo Elapsed, RootDepth, FinalScore,
RootAlpha, RootBeta ' Output to GUI
14159 End If
14160
14161 If FinalScore <= RootAlpha Then '<<< search failed low
14162 #If DEBUG_MODE Then
14163 If RootDepth > 5 Then
14164 SendCommand " Research " & " SC:" & FinalScore & " <= A:" &
RootAlpha
14165 End If
14166 #End If
14167 RootBeta = (RootAlpha + RootBeta) \ 2
14168 RootAlpha = GetMax(FinalScore - Delta, -MATE0)
14169
14170 If ThreadNum <= 0 Then FailedHighCnt = 0
14171 bResearching = True
14172 If ThreadNum <= 0 Then bFailedLowAtRoot = True
14173
14174 ElseIf FinalScore >= RootBeta Then '<<<< search failed high
14175 #If DEBUG_MODE Then
14176 If RootDepth > 5 Then
14177 SendCommand " Research " & " SC:" & FinalScore & " B:" & RootBeta
14178 End If
14179 #End If
14180 If ThreadNum <= 0 Then FailedHighCnt = FailedHighCnt + 1
14181 RootBeta = GetMin(FinalScore + Delta, MATE0)
14182 bResearching = True
14183 Else
14184 Exit Do '<<< search result in alpha/beta window: finish this search depth
14185 End If
14186
14187 ' mate search?
14188 If FinalScore > 2 * ScoreQueen.EG And FinalScore <> MATE0 Then
RootBeta = MATE0
14189 ElseIf FinalScore < -2 * ScoreQueen.EG And FinalScore <> -MATE0 Then
RootAlpha = -MATE0
14190 End If
14191
14192 ' set new delta for research
14193 Debug.Assert Abs(Delta) <= 200000
14194 If Abs(Delta) < MATE_IN_MAX_PLY Then Delta = Delta + (Delta \ 4 + 5)
14195 Debug.Assert Abs(Delta) <= 200000
14196
14197 DoEvents
14198 Loop
14199
14200 '=====
14201

```

```

14204 '--- Search result for current iteration ---
14205 '=====
14206
14207 If (bOnlyMove And RootDepth = 1) Then FinalScore = LastFinalScore Else
LastFinalScore = FinalScore
14208
14209 If FinalScore <> VALUE_NONE And FinalScore <> -VALUE_INFINITE Then
14210     If Not bTimeExit Then
14211         If FinalMove.From > 0 Then FinalCompletedDepth = AdjustedDepth
14212     End If
14213     If ThreadNum > 0 And Trim(MoveText(PV(1, 1))) = "" Then
14214         If bThreadTrace Then WriteTrace "!!!???Think:PV Empty "
14215     Else
14216         If ThreadNum > 0 And PVLength(1) > 1 Then
14217             WriteMapBestPVforThread FinalCompletedDepth, FinalScore, FinalMove
14218         Else
14219             If bThreadTrace Then WriteTrace "Think: else PVLen<2" & PVLength(1)
14220         End If
14221     End If
14222     CompMove = FinalMove
14223     PlyScore(RootDepth) = FinalScore
14224     If bPostMode And PVLength(1) >= 1 Then
14225         Elapsed = TimeElapsed()
14226         If Not bExitReceived Then SendThinkInfo Elapsed, RootDepth, FinalScore,
RootAlpha, RootBeta ' Output to GUI
14227     End If
14228 End If
14229
14230 CopyIntArr BoardTmp, Board ' copy old position to main board / just to be sure
14231
14232 If bOnlyMove Or IsTBScore Then
14233     bOnlyMove = False: Exit For
14234 End If
14235 If RootDepth > 2 And FinalScore > MATE0 - RootDepth Then
14236     Exit For
14237 End If
14238 If bTimeExit Or IsTBScore Or (RootDepth = 1 And LastMove.From = 0) Then GoTo
lblIterationsExit
14239
14240 'easy move?
14241 If RootDepth >= 7 - 3 * Abs(pbIsOfficeMode) And EasyMove.From > 0 And Not
FixedDepthMode And Not FixedTime > 0 Then
14242     If bTimeTrace Then WriteTrace "Easy check PV (IT:" & RootDepth & "): EM:" &
MoveText(EasyMove) & ": PV1:" & MoveText(PV(1, 1))
14243     If MovesEqual(PV(1, 1), EasyMove) Then
14244         If bTimeTrace Then WriteTrace "Easy check2 bestmove: " & Format(
BestMoveChanges, "0.000")
14245         If BestMoveChanges < 0.03 Then
14246             Elapsed = TimeElapsed()
14247             If bTimeTrace Then WriteTrace "Easy check3 Elapsed: " & Format$(Elapsed,
"0.00") & Format$(OptimalTime * 5# / 42#, "0.00")
14248             If Elapsed > OptimalTime * 5# / 44# Then
14249                 bEasyMovePlayed = True
14250                 bTimeExit = True
14251                 If bTimeTrace Then
14252                     WriteTrace "Easy move played: " & MoveText(EasyMove) & " Elapsed:" &
Format$(Elapsed, "0.00") & ", Opt:" & Format$(OptimalTime, "0.00")
& ", Max:" & Format$(MaximumTime, "0.00") & ", Left:" & Format$(
TimeLeft, "0.00")
14253                 End If
14254             End If
14255         End If
14256     End If
14257 End If
14258
14259 If RootDepth > 15 Then 'emergency exit or mate found?
14260     If RootDepth > 80 Or (Abs(FinalScore) > MATE0 - 6 And Abs(FinalScore) < MATE0)
Then bTimeExit = True

```

[illegible]



```

14380                                     GoodMoves As Long) As TMOVE
14381 Dim RootScore As Long, CurrMove As Long
14382 Dim BestRootScore As Long
14383 Dim BestRootMove As TMOVE, CurrentMove As TMOVE, HashMove As TMOVE
14384 Dim LegalMoveCnt As Long, bCheckBest As Boolean, QuietMoves As Long,
CaptureMoves As Long
14385 Dim Elapsed As Single, lExtension As Long
14386 Dim PrevMove As TMOVE
14387 Dim CutNode As Boolean, r As Long, Factor As Long, s As String
14388 Dim NewDepth As Long, Depth1 As Long, bCaptureOrPromotion As Boolean
14389 Dim Hashkey As THashKey, EgCnt As Long, i As Long, bLegal As Boolean
14390 Dim EGTBBestRootMoveRootStr As String, EGTBBestRootMoveListRootStr As String
14391 Dim Improving As Long
14392 Dim ss As Long ' Search stack pointer
14393 Dim BestValueCnt As Long
14394
14395 Dim bHashFound As Boolean, ttHit As Boolean, HashEvalType As Long, HashScore As Long
, HashStaticEval As Long, HashDepth As Long
14396 Dim ttMove As TMOVE, ttValue As Long, HashPvHit As Boolean
14397
14398 '-----
14399 ss = 1 ' reset search stack
14400 Ply = 1 ' start with ply 1 for root
14401
14402 InitPieceSquares '-- also sets WKINGLOC and BKINGLOC needed for InCheck-Function later!
14403 InitEpArr
14404
14405 EGTBRootResultScore = VALUE_NONE
14406 RootStartScore = Eval()
14407 PieceCntRoot = 2 + PieceCnt(WPAWN) + PieceCnt(WKNIGHT) + PieceCnt(WBISHOP) +
PieceCnt(WROOK) + PieceCnt(WQUEEN) + PieceCnt(BPAWN) + PieceCnt(BKNIGHT) + PieceCnt(
BBISHOP) + PieceCnt(BROOK) + PieceCnt(BQUEEN) ' For TableBases
14408 ' PlyMatScore (1) = WMaterial - BMaterial
14409 RootMatScore = WMaterial - BMaterial: If Not bWhiteToMove Then RootMatScore = -
RootMatScore
14410 'RootSimpleEval = CalcSimpleEval()
14411 StaticEvalArr(0) = RootStartScore
14412 StaticEvalArr(ss + 1) = VALUE_NONE
14413
14414 CutNode = False: QSDepth = 0
14415 bOnlyMove = False
14416 GoodMoves = 0: RootMoveCnt = 0
14417 ClearMove PrevMove
14418 BestRootScore = -MATE0
14419 ClearMove BestRootMove
14420 PliesFromNull = GameMovesCnt
14421 ClearMove BestMovePly(ss): ClearMove BestMovePly(ss + 1)
14422 If GameMovesCnt > 0 Then PrevMove = arGameMoves(GameMovesCnt)
14423 PrevMove.IsChecking = InCheck()
14424 Improving = Abs(Not PrevMove.IsChecking)
14425 StatScore(0) = 0
14426 If PrevMove.From > 0 Then StatScore(0) = History(PieceColor(PrevMove.Piece),
PrevMove.From, PrevMove.Target) - 4000
14427
14428 ' init history values
14429 CmhPtr(ss) = 0
14430 NullMovePly = 0
14431 RootDelta = 0
14432
14433 StatScore(ss) = 0
14434
14435 With Killer(ss + 2)
14436 ClearMove .Killer1: ClearMove .Killer2: ClearMove .Killer3
14437 End With
14438
14439 ' ---Test time needed for evaluation function
14440 ' Debug.Print "-----"
14441 ' If bEvalBench Then

```

[illegible]



```

OrderMoves 1, CntRootMoves, PrevMove, EmptyMove, EmptyMove, False,
LegalRootMovesOutOfCheck
FinalMove = SearchRoot: FinalScore = EGTBRootResultScore: BestRootScore = 
FinalScore: PV(1, 1) = SearchRoot: PVLenght(1) = 2
'Debug.Print "RootPos: "; CompToCoord(Moves(1, CurrMove)), FinalScore
Elapsed = TimeElapsed()
bTimeExit = True 'no more search
LegalMoveCnt = 1
If pbIsOfficeMode Then
    If EGTBRootResultScore = 0 Then
        s = "DRAW"
    ElseIf EGTBRootResultScore > 0 Then
        If EGTBRootResultScore = 100000 Then
            s = "White mates!"
        Else
            s = "White wins in " & Abs(100000 - EGTBRootResultScore - 1) \ 2 &
                " moves"
        End If
    ElseIf EGTBRootResultScore < 0 Then
        If EGTBRootResultScore = -100000 Then
            s = "Black mates!"
        Else
            s = "Black wins in " & Abs(100000 + EGTBRootResultScore + 1) \ 2 &
                " moves"
        End If
    End If
End If
SendCommand s
End If
GoTo lblEndRootMoves '<<<<<<<<<<<< NO MORE SEARCH NEEDED for tablebase move
End If
Next
End If
End If
End If '<<< Endgame Tablebase check

Elapsed = TimeElapsed()
BestValueCnt = 0

'=====
'= loop for root moves           =
'=====
For CurrMove = 0 To CntRootMoves - 1
    CurrentMove = Moves(1, CurrMove)
    MovePickerDat(ss).CurrMoveNum = CurrMove
    ' WriteTrace "SearchRoot RootDepth=" & RootDepth & " " & CurrMove & " " & MoveText(CurrentMove) & "
    Cnt=" & EGTBMovelistCnt(ss)
    ' Debug.Print MoveText(CurrentMove)
    RootScore = -VALUE_INFINITE
    If EGTBMovelistCnt(1) > 0 Then
        ' Filter for endgame tablebase move: Ignore losingmoves if draw or win from tablebases
        For EgCnt = 1 To EGTBMovelistCnt(1)
            If CompToCoord(CurrentMove) = EGTBMovelist(1, EgCnt) Then GoTo lblEGMoveOK
        Next
        GoTo lblNextRootMove
    End If
lblEGMoveOK:
    CmHPtr(ss) = CurrentMove.Piece * MAX_BOARD + CurrentMove.Target ' set pointer for history
move statistics
    ' WriteTrace "SearchRoot RootDepth=" & RootDepth & " " & CurrMove & " OK "

    '-----
    '--- Make root move -
    '-----

RemoveEpPiece
MakeMove CurrentMove: Ply = Ply + 1: bCheckBest = False: bLegal = False

If CheckLegal(CurrentMove) Then

```

```

14565 Nodes = Nodes + 1: bLegal = True: LegalMoveCnt = LegalMoveCnt + 1: RootMoveCnt =
    LegalMoveCnt
14566 bCaptureOrPromotion = CurrentMove.Captured <> NO_PIECE Or CurrentMove.Promoted
    <> 0
14567 HashBoard Hashkey, EmptyMove
14568 If pbIsOfficeMode And RootDepth > 3 Then ' Show move cnt
14569     ShowMoveInfo MoveText(FinalMove), RootDepth, MaxPly, EvalSFTto100(FinalScore),
        Elapsed
14570 End If
14571 If UCIMode Then
14572     If TimeElapsed() > 3 Then
14573         SendCommand "info depth " & RootDepth & " currmove " & UCIMoveText(
            CurrentMove) & " currmove number " & LegalMoveCnt
14574     End If
14575 End If
14576 bFirstRootMove = CBool(LegalMoveCnt = 1)
14577 SetMove MovesList(ss), CurrentMove
14578 StaticEvalArr(ss) = RootStartScore
14579 RootMove = CurrentMove
14580 '-----
14581 'WriteTrace "Root:" & RootDepth & ": " & MoveText(CurrentMove) & " Score:" & FinalScore
14582 r = 0
14583 lExtension = 0
14584 '
14585 '--- Check extension ---
14586 '
14587 If (CurrentMove.IsChecking) Then
14588     If SEEGreaterOrEqual(CurrentMove, 0) Then
14589         lExtension = 1: GoTo lblEndExtensions
14590     End If
14591 End If
14592 ' Castling extension
14593 If CurrentMove.Castle <> NO_CASTLE Then
14594     lExtension = 1: GoTo lblEndExtensions
14595 End If
14596 ' Passed pawn move extension
14597 If PieceType(CurrentMove.Captured) = PT_PAWN Then
14598     If AdvancedPassedPawnPush(CurrentMove.Piece, CurrentMove.Target) Then
14599         lExtension = 1: GoTo lblEndExtensions
14600     End If
14601 End If
14602
14603 lblEndExtensions:
14604
14605 '--- new search depth
14606 NewDepth = GetMax(0, Depth + lExtension - 1)
14607 '
14608
14609 '--- Step 16. Reduced depth search (LMR). If the move fails high it will be re-searched at full depth.
14610 '
14611 r = Reduction(Improving, Depth, LegalMoveCnt, (Beta - Alpha), RootDelta)
14612 r = r - 1 ' is Pv
14613
14614 If Not bCaptureOrPromotion Then
14615     '--- Decrease reduction for moves that escape a capture
14616     If CurrentMove.Castle = NO_CASTLE Then
14617         TmpMove.From = CurrentMove.Target: TmpMove.Target = CurrentMove.From:
            TmpMove.Piece = CurrentMove.Piece: TmpMove.Captured = NO_PIECE:
            TmpMove.SeeValue = VALUE_NONE
14618         ' Move back to old square, were we in danger there?
14619         If Not SEEGreaterOrEqual(TmpMove, -MAX_SEE_DIFF) Then r = r - 2 ' old square
            was dangerous
14620     End If
14621 End If
14622
14623 StatScore(ss) = History(PieceColor(CurrentMove.Piece), CurrentMove.From,
    CurrentMove.Target) - 4000 ' fill here if needed in next ply
14624 Dim CmH As Long

```

[illegible]

```

14685         GoodMoves = GoodMoves + 1
14686         DepthInWork = RootDepth ' For decision if better thread
14687     End If
14688     '-----
14689     '--- Save final move -
14690     '-----
14691
14692     ' Store PV: best moves
14693     UpdatePV ss, FinalMove
14694     If PVLength(1) = 2 Then
14695         ' try to get 2nd move from hash
14696         HashMove = GetHashMove(Hashkey)
14697         If HashMove.From > 0 Then
14698             PV(1, 2) = HashMove: PVLength(1) = 3
14699         Else
14700             ClearMove PV(1, 2)
14701         End If
14702         If LastFullPVLen > 2 Then
14703             If MovesEqual(PV(1, 1), LastFullPVArr(1)) Then
14704                 For r = 1 To LastFullPVLen: SetMove PV(1, r), LastFullPVArr(r): Next
14705                 PVLength(1) = LastFullPVLen
14706             End If
14707         End If
14708     ElseIf PVLength(1) > 2 Then
14709         For r = 1 To PVLength(1): SetMove LastFullPVArr(r), PV(1, r): Next
14710         LastFullPVLen = PVLength(1)
14711     End If
14712     If PV(1, 1).From > 0 Then ' helper thread writes result for main thread 0
14713         If ThreadNum > 0 Then WriteMapBestPVforThread FinalCompletedDepth, FinalScore,
            FinalMove
14714     End If
14715     LastChangeDepth = RootDepth
14716     LastChangeMove = MoveText(PV(1, 1))
14717 End If
14718 '
14719 '----- normal alpha beta check -----
14720 '
14721 If RootScore > BestRootScore Then
14722     BestRootScore = RootScore
14723
14724     If RootScore > Alpha Then
14725         BestRootMove = BestRootMove
14726
14727         If RootScore >= Beta Then
14728             Exit For ' fail high
14729         Else
14730             If Depth > 2 And Depth < 12 And Beta < 14000 And RootScore > -12000 Then
14731                 Depth = Depth - 2
14732                 Alpha = RootScore
14733             End If
14734             ElseIf BestRootMove.From = 0 Then
14735                 BestValueCnt = BestValueCnt + 1
14736                 If BestValueCnt >= 3 Then Exit For
14737             End If
14738         End If
14739 '--- Add Quiet move, used for pruning and history update
14740 '
14741 If Not MovesEqual(BestRootMove, CurrentMove) Then
14742     If CurrentMove.Captured = NO_PIECE And CurrentMove.Promoted = 0 And QuietMoves <
        64 Then
14743         QuietMoves = QuietMoves + 1: QuietsSearched(ss, QuietMoves) = CurrentMove
14744     ElseIf CurrentMove.Captured <> NO_PIECE And CaptureMoves < 32 Then
14745         CaptureMoves = CaptureMoves + 1: CapturesSearched(ss, CaptureMoves) =
            CurrentMove
14746     End If
14747 End If
14748

```

```

14749 'If bTimeTrace Then WriteTrace "SearchRoot: FixedTime: " & FixedTime & " " & FixedDepthMode & ", TimeDiff:"
14750 & TimeElapsed()
14751 If Not FixedDepthMode And GoodMoves > 0 And Not bAnalyzeMode Then
14752     If FixedTime > 0 Then
14753         If TimeElapsed() >= FixedTime - 0.1 Then
14754             bTimeExit = True
14755         End If
14756     ElseIf (RootDepth > LIGHTNING_DEPTH) Then ' Time for next move?
14757         If Not CheckTime() Then
14758             SearchTime = TimeElapsed()
14759             If bTimeTrace Then WriteTrace "Exit SearchRoot3: Used:" & Format$(SearchTime
14760 , "0.00") & " OptimalTime:" & Format$(OptimalTime, "0.00")
14761             bTimeExit = True
14762         End If
14763     End If
14764 End If
14765
14766 If (bTimeExit And LegalMoveCnt > 0) Or RootScore = MATE0 - 1 Then Exit For
14767
14768 If pbIsOfficeMode Then
14769     If bTimeExit Then
14770         SearchTime = TimeElapsed()
14771         'Debug.Print Nodes, SearchTime
14772     End If
14773     #If VBA_MODE = 1 Then
14774         '-- Office sometimes lost focus for Powerpoint
14775         If Application.Name = "Microsoft PowerPoint" Then
14776             If RootDepth > 4 Then frmChessX.cmdStop.SetFocus
14777         End If
14778     #End If
14779     If RootDepth > 2 Then DoEvents
14780 Else
14781     If RootDepth > 6 Then DoEvents
14782 End If
14783 If bTimeExit Then Exit For
14784 '
14785 lblNextRootMove:
14786 Next CurrMove
14787
14788 '---<<< End of root moves loop -----
14789
14790 lblEndRootMoves:
14791 '-----
14792 '--- End of game? -
14793 '-----
14794 If LegalMoveCnt = 0 Then ' no move
14795     If InCheck Then ' Mate
14796         If bWhiteToMove Then
14797             Result = BLACK_WON
14798         Else
14799             Result = WHITE_WON
14800         End If
14801     Else 'draw
14802         Result = DRAW_RESULT: FinalScore = 0
14803         SetMove FinalMove, EmptyMove
14804     End If
14805     GoodMoves = -1
14806 Else
14807     If (LegalMoveCnt = 1 And RootDepth = 1) And Not bTimeExit Then bOnlyMove = True:
14808     RootScore = 0: FinalScore = 0 ' single move only?
14809     If RootScore = MATE0 - 2 Then ' Mate
14810         If bWhiteToMove Then
14811             Result = WHITE_WON
14812         Else
14813             Result = BLACK_WON
14814         End If
14815     Else
14816         If Fifty > 99 Then ' Draw 50 moves rule

```

```

14814         Result = DRAW_RESULT
14815     End If
14816 End If
14817 End If
14818
14819 If FinalMove.From > 0 And Not bTimeExit Then
14820     UpdateStats ss, FinalMove, BestRootScore, Beta, QuietMoves, CaptureMoves,
14821     EmptyMove, RootDepth ' update statistics
14822     '-----
14823     '--->>> Save hash for root
14824     '-----
14825     HashBoard Hashkey, EmptyMove ' was changed above
14826
14827     If FinalScore >= Beta Then
14828         HashEvalType = TT_LOWER_BOUND
14829     ElseIf FinalMove.From >= SQ_A1 Then
14830         HashEvalType = TT_EXACT
14831     Else
14832         HashEvalType = TT_UPPER_BOUND
14833     End If
14834
14835     HashBoard Hashkey, EmptyMove ' changed before
14836     HashTableSave Hashkey, Depth, FinalMove, HashEvalType, FinalScore, StaticEvalArr(0
), True
14837     ' WriteTrace "SearchRoot SAVE TT:" & ThreadNum & ". " & RootDepth & " > " & MoveText(FinalMove) & " < "
& FinalScore
14838     '-----
14839     '<<< Save hash for root
14840     '-----
14841
14842 End If ' FinalMove.From
14843
14844 '-----
14845 ' Return final move -
14846 '-----
14847 SearchRoot = FinalMove
14848
14849 'WriteDebug "Root: " & RootDepth & " Best:" & MoveText(SearchRoot) & " Sc:" & BestRootScore & " M:" &
GoodMoves
14850 End Function
14851
14852
14853 '=====
14854 '= Search: Search moves from ply=2 to x. =
14855 '= called by SEARCHROOT, calls SEARCH recursively, then QSEARCH. =
14856 '= Returns eval score for a position with a specific search depth =
14857 '=====
14858 Private Function Search(ByVal ss As Long, _
14859     ByVal PVNode As Boolean, _
14860     ByVal Alpha As Long, _
14861     ByVal Beta As Long, _
14862     ByVal Depth As Long, _
14863     InPrevMove As TMOVE, _
14864     ExcludedMove As TMOVE, _
14865     ByVal CutNode As Boolean, ByVal PrevMoveExtension As Long) As
Long
14866 '-----
14867 Dim CurrentMove As TMOVE, Score As Long, bNoMoves As Boolean, bLegalMove As
Boolean, LegalMovesOutOfCheck As Long
14868 Dim NullScore As Long, PrevMove As TMOVE, QuietMoves As Long, CaptureMoves
As Long, rBeta As Long, rDepth As Long
14869 Dim StaticEval As Long, GoodMoves As Long, NewDepth As Long, LegalMoveCnt As
Long, MoveCnt As Long
14870 Dim lExtension As Long, lPlyExtension As Long, bTTMoveIsSingular As Boolean
14871 Dim bMoveCountPruning As Boolean, bKillerMove As Boolean, bTTCapture As Boolean,
lSingularExtension As Long
14872 Dim r As Long, Improving As Long, bCaptureOrPromotion As Boolean,

```

```

14873 LmrDepth As Long, Depth1 As Long
14874 Dim BestValue As Long, bIsNullMove As Boolean, ThreatMove As TMOVE,
TryBestMove As TMOVE
14875 Dim bHashFound As Boolean, ttHit As Boolean, HashEvalType As Long, HashScore
As Long, HashStaticEval As Long, HashDepth As Long, HashThreadNum As Long
14876 Dim EvalScore As Long, Hashkey As THashKey, HashMove As TMOVE, ttMove As
TMOVE, ttValue As Long, HashPvHit As Boolean
14877 Dim BestMove As TMOVE, sInput As String, MoveStr As String, Factor As Long,
HistoryVal As Long
14878 Dim CmH As Long, Fmh1 As Long, FMh3 As Long, HistVal As Long, CurrPtr
As Long, Cm_Ok As Boolean
14879 Dim IsEGTbPos As Boolean, bSingularExtensionNode As Boolean, ttPv As Boolean
, bSkipQuiets As Boolean
14880 Dim bSingularQuietLMR As Boolean, bLikelyFailLow As Boolean, Bonus As Long,
bAlmostFutilPruned As Boolean
14881 '-----
14882 Debug.Assert Not (PVNode And CutNode)
14883 Debug.Assert (PVNode Or (Alpha = Beta - 1))
14884 Debug.Assert (¬VALUE_INFINITE <= Alpha And Alpha < Beta And Beta <= VALUE_INFINITE)
14885 Debug.Assert ss = Ply
14886 '-----
14887 '--- Step 1. Initialize node for search -
14888 '-----
14889 SetMove PrevMove, InPrevMove '--- bug fix: make copy to avoid changes in parameter use
14890 BestValue = -VALUE_INFINITE: ClearMove BestMove: ClearMove BestMovePly(ss):
ClearMove BestMovePly(ss + 1)
14891 EvalScore = VALUE_NONE
14892
14893 StaticEvalArr(ss + 1) = VALUE_NONE
14894 If ExcludedMove.From = 0 Then
14895     StaticEval = VALUE_NONE: StaticEvalArr(ss) = VALUE_NONE
14896 Else
14897     StaticEval = StaticEvalArr(ss)
14898 End If
14899
14900 If bSearchingPV Then PVNode = True: CutNode = False ' searching main line is always principle
variation
14901
14902 If Ply > MaxPly Then MaxPly = Ply '--- Max depth reached in normal search
14903
14904 '---- QSEARCH ?----
14905 If Depth <= 0 Or Ply >= MAX_DEPTH - 5 Then
14906     Search = QSearch(ss, PVNode, Alpha, Beta, MAX_DEPTH, PrevMove, QS_CHECKS)
14907     Exit Function '<<<<<< RETURN >>>>>>'
14908 End If
14909
14910 ClearMove ThreatMove: bTTMoveIsSingular = False
14911 bIsNullMove = (PrevMove.From < SQ_A1)
14912 EGTBMoveListCnt(ss) = 0
14913 '--- Debug ---
14914 'dmoves 'list search moves in debug window
14915 'If Ply = 2 And Left$(MoveText(PrevMove),4) = "c6d6" Then Stop ' Left needed for checking +
14916 'If RootDepth = 3 And Ply = 2 Then Debug.Print PrintPos, Movetext(PrevMove): Stop
14917 'If Nodes = 1127 Then Stop
14918 'If Ply > 70 Then Stop
14919 'If SearchMovesList = "h2c2 a1h1" Then Stop
14920 ' If Ply = 2 And Left$(MoveText(PrevMove), 4) = "g5d8" Then Stop ' Left needed for checking +
14921
14922 bAlmostFutilPruned = False
14923 StatScore(ss) = 0
14924 CmHPtr(ss) = 0
14925 DoubleExtensions(ss) = DoubleExtensions(ss - 1)
14926 With Killer(ss + 2)
14927     ClearMove .Killer1: ClearMove .Killer2: ClearMove .Killer3
14928 End With
14929 CutOffCnt(ss + 2) = 0
14930 ttPv = PVNode: ttPVArr(ss) = ttPv

```



```

14931 '
14932 '
14933 '--- Step 2. Check for aborted search and immediate draw
14934 '
14935 HashBoard Hashkey, ExcludedMove ' Save current position hash keys for insert later
14936 GamePosHash(GameMovesCnt + Ply - 1) = Hashkey
14937
14938
14939 ' Step 2. Check immediate draw
14940 If Fifty > 99 Then ' 50 moves rule draw ?
14941     If CompToMove() Then Search = DrawContempt Else Search = -DrawContempt
14942     PVLength(ss) = 0
14943     Exit Function
14944 End If
14945
14946 If Not bIsNullMove Then
14947     '--- 3x repeated position draw?
14948     If Fifty >= 3 And PliesFromNull >= 3 Then
14949         If Is3xDRAW(Hashkey, GameMovesCnt, Ply) Then
14950             If CompToMove() Then Search = DrawContempt Else Search = -DrawContempt
14951             PVLength(ss) = 0
14952             Exit Function
14953         End If
14954     End If
14955 End If
14956
14957 ' Endgame tablebase position?
14958 IsEGTbPos = False
14959 If EGTBasesEnabled And Ply <= EGTBasesMaxPly Then
14960     ' For first plies only because TB access is very slow for this implementation
14961     ' If EGTBRootResultScore = VALUE_NONE And PrevMove.Captured <> NO_PIECE Then ' not a TB position
14962     ' at root
14963     ' If Ply <= EGTBasesMaxPly And PrevMove.Captured <> NO_PIECE Then ' captured because else TB access
14964     ' in previous ply
14965     If IsEGTbBasePosition() Then
14966         If IsTimeForEGTbBaseProbe() Then
14967             IsEGTbPos = True
14968         End If
14969     End If
14970 End If
14971
14972 '--- Step 3.: Mate distance pruning
14973 '
14974 Alpha = GetMax(-MATE0 + Ply, Alpha)
14975 Beta = GetMin(MATE0 - Ply + 1, Beta)
14976 If Alpha >= Beta Then Search = Alpha: Exit Function
14977
14978 If Alpha < DrawContempt And Fifty >= 3 And PliesFromNull >= 3 Then
14979 ' If Alpha < -DrawContemptForSide() And Fifty >= 3 Then
14980     If CyclingMoves(ss) Then
14981         Alpha = DrawContempt
14982         If Alpha >= Beta Then Search = Alpha: Exit Function
14983     End If
14984 End If
14985
14986 '
14987 '--- Step 4. Transposition hash table lookup
14988 '
14989 NullScore = VALUE_NONE
14990 bHashFound = False: ttHit = False: ClearMove HashMove
14991 ttHit = False: ClearMove ttMove: ttValue = VALUE_NONE: bTTCapture = False
14992
14993 If Depth >= 0 Then
14994     ttHit = HashTableRead(Hashkey, HashDepth, HashMove, HashEvalType, HashScore,
14995     HashStaticEval, HashPvHit, HashThreadNum)
14996     If ttHit Then

```

```

14996 SetMove ttMove, HashMove: ttValue = HashScore
14997 If HashMove.From <> 0 Then
14998     SetMove BestMovePly(ss), HashMove
14999     bTTCapture = (ttMove.Captured <> NO_PIECE Or ttMove.Promoted <> 0)
15000 End If
15001 If ExcludedMove.From = 0 Then ttPv = ttPv Or HashPvHit: ttPVArr(ss) = ttPv '
    ttPv=PvNode earlier
15002 End If
15003
15004 Dim bDoTT As Boolean
15005
15006 If ThreadNum <= 0 Then ' single core / main thread / different to Stockfish logic HashDepth> Depth
15007     bDoTT = (Not PVNode Or HashDepth = TT_TB_BASE_DEPTH) And HashDepth >= Depth
        And ttHit And ttValue <> VALUE_NONE And ExcludedMove.From = 0
15008 Else ' multi core helper threads: different logic
15009     bDoTT = (Not PVNode Or HashDepth = TT_TB_BASE_DEPTH) And (HashDepth >= Depth -
        Abs(HashEvalType = TT_EXACT)) And ttHit And ttValue <> VALUE_NONE And
        ExcludedMove.From = 0
15010 End If
15011 If bDoTT Then
15012     If ttValue >= Beta Then
15013         bHashFound = CBool(HashEvalType And TT_LOWER_BOUND) ' bit wise compare eq:
            (HashEvalType = TT_LOWER_BOUND Or HashEvalType = TT_EXACT)
15014     Else
15015         bHashFound = CBool(HashEvalType And TT_UPPER_BOUND) ' bit wise compare eq:
            ((HashEvalType = TT_UPPER_BOUND Or HashEvalType = TT_EXACT)
15016     End If
15017 If bHashFound Then
15018     If IsEGTbPos And HashDepth <> TT_TB_BASE_DEPTH Then
15019         ' Ignore Hash and continue with TableBase query
15020     Else
15021         If ttMove.From >= SQ_A1 Then
15022             If ttValue >= Beta Then
15023                 If Not bTTCapture Then
15024                     '--- Update statistics
15025                     UpdQuietStats ss, ttMove, PrevMove, StatBonus(Depth)
15026                 End If
15027
15028                 ' Extra penalty for a quiet TT move in previous ply when it gets refuted
15029                 If PrevMove.Captured = NO_PIECE Then
15030                     If PrevMove.From > 0 Then
15031                         If MovePickerDat(ss - 1).CurrMoveNum < 2 Or MovesEqual(PrevMove,
                            Killer(ss - 1).Killer1) Then
15032                             UpdateContHistStats ss - 1, PrevMove.Piece, PrevMove.Target, -
                                StatBonus(Depth + 1)
15033                         End If
15034                     End If
15035                 End If
15036             ElseIf Not bTTCapture Then
15037                 ' Penalty for a quiet ttMove that fails low
15038                 Bonus = -StatBonus(Depth)
15039                 UpdHistory ttMove.Piece, ttMove.From, ttMove.Target, Bonus
15040                 UpdateContHistStats ss, ttMove.Piece, ttMove.Target, Bonus
15041             End If ' ttValue >= Beta
15042         End If ' ttMove.From >= SQ_A1
15043
15044         If Fifty < 90 Then
15045             Search = ttValue
15046             BestMovePly(ss) = ttMove
15047             Exit Function ' <<<< exit with TT move
15048         End If
15049     End If
15050 End If
15051 End If
15052 End If '--- End Hash
15053
15054 If Ply + Depth > MAX_DEPTH Then Depth = MAX_DEPTH - Ply - 2
15055 StaticEval = StaticEvalArr(ss)

```

```

15056 bNoMoves = True
15057 ClearMove BestMovePly(ss)
15058
15059 '--- Check Time ---
15060 If Not FixedDepthMode Or ThreadNum > 0 Then
15061     '-- Fix:Nodes Mod 1000 > not working because nodes are incremented in QSearch too
15062     If (Nodes > LastNodesCnt + (GUICheckIntervalNodes * 2 \ (1 + Abs(bEndgame)))) And
15063         (RootDepth > LIGHTNING_DEPTH Or Ply = 2) Then
15064         #If DEBUG_MODE <> 0 Then
15065             DoEvents
15066         #End If
15067         '--- Check new commands from GUI (i.e. analyze stop)
15068         If PollCommand Then
15069             If bThreadTrace Then WriteTrace "Search PollCommand: ThreadCommand =" &
15070                 ThreadCommand & " / " & Now()
15071             sInput = ReadCommand
15072             If Left$(sInput, 1) = "." Then
15073                 SendAnalyzeInfo
15074             Else
15075                 If sInput <> "" Then
15076                     ParseCommand sInput
15077                 End If
15078             End If
15079         End If
15080         If ThreadNum > 0 Then CheckThreadTermination False ' <<< program my end here
15081         LastNodesCnt = Nodes
15082         If bTimeExit Then Search = 0: Exit Function
15083         If FixedTime > 0 Then
15084             If Not bAnalyzeMode And TimeElapsed() >= FixedTime - 0.1 Then bTimeExit = True
15085             : Exit Function
15086         ElseIf Not bAnalyzeMode Then
15087             If TimeElapsed() > MaximumTime Then
15088                 If bTimeTrace Then WriteTrace "Exit Search: TimeElapsed: " & Format$(
15089                     TimeElapsed()) & ", Maximum: " & Format$(MaximumTime, "0.00")
15090                 bTimeExit = True: Search = 0: Exit Function
15091             End If
15092         End If
15093     End If
15094 End If
15095
15096 '
15097 '--- / Step 5. Tablebase (endgame) - not active any more because too slow with external calls
15098 '
15099 ' Tablebase access / too slow in live tests
15100 ' If IsEGTbPos And HashDepth <> TT_TB_BASE_DEPTH Then ' Position already done and saved in hash?
15101 ' Dim sTbFEN As String, IEGTBResultScore As Long, sEGTBBestMoveStr As String, sEGTBBestMoveListStr As
15102 String
15103 ' sTbFEN = WriteEPD()
15104 ' If bEGTbBaseTrace Then WriteTrace "TB-Search: check move " & MoveText(PrevMove) & ", ply=" & Ply
15105 ' If ProbeTablebases(sTbFEN, IEGTBResultScore, True, sEGTBBestMoveStr, sEGTBBestMoveListStr) Then
15106 '     BestMove = TextToMove(sEGTBBestMoveStr)
15107 '     StaticEval = Eval(): IEGTBResultScore = IEGTBResultScore + StaticEval
15108 '     If bEGTbBaseTrace Then WriteTrace "TB-Search: Move " & sEGTBBestMoveStr & " " & IEGTBResultScore & "
15109 ply=" & Ply
15110 '     Search = IEGTBResultScore
15111 '     HashTableSave HashKey, TT_TB_BASE_DEPTH, EmptyMove, TT_EXACT, IEGTBResultScore,
15112 IEGTBResultScore, ttPv
15113 '     SetMove ttMove, BestMove
15114 ' End If
15115 ' End If
15116
15117 '--- / Step 6. Evaluate the position statically
15118 If PrevMove.IsChecking Then
15119     StaticEval = VALUE_NONE: StaticEvalArr(ss) = VALUE_NONE: EvalScore = VALUE_NONE:
15120     Improving = 0
15121     GoTo lblSkipEarlyPruning ' lblMovesLoop worse
15122 ElseIf ExcludedMove.From <> 0 Then

```

```

15116     StaticEval = StaticEvalArr(ss)
15117     EvalScore = StaticEval
15118 ElseIf ttHit Then
15119     If HashStaticEval = VALUE_NONE Then StaticEval = Eval() Else StaticEval =
HashStaticEval
15120     EvalScore = StaticEval
15121     If ttValue <> VALUE_NONE Then
15122         If ttValue > EvalScore Then
15123             If CBool(HashEvalType And TT_LOWER_BOUND) Then EvalScore = ttValue
15124             Else
15125                 If CBool(HashEvalType And TT_UPPER_BOUND) Then EvalScore = ttValue
15126             End If
15127         End If
15128     Else
15129         If StaticEval = VALUE_NONE Then
15130             StaticEval = Eval() ' <<< evaluate position
15131         End If
15132         HashTableSave Hashkey, DEPTH_NONE, EmptyMove, TT_NO_BOUND, VALUE_NONE, StaticEval,
ttPv ' Save TT
15133         EvalScore = StaticEval
15134     End If
15135     StaticEvalArr(ss) = StaticEval
15136
15137 '--- Improving ?
15138 Improving = 1
15139 If StaticEvalArr(ss - 2) <> VALUE_NONE Then
15140     Improving = Abs(StaticEval > StaticEvalArr(ss - 2))
15141 ElseIf StaticEvalArr(ss - 4) <> VALUE_NONE Then
15142     Improving = Abs(StaticEval > StaticEvalArr(ss - 4))
15143 End If
15144
15145 If RootDepth <= 4 Then GoTo lblMovesLoop
15146
15147 If (bWhiteToMove And CBool(WNonPawnMaterial = 0)) Or (Not bWhiteToMove And CBool(
BNonPawnMaterial = 0)) Then GoTo lblMovesLoop
15148 '
15149 '--- Step 7. Razoring (skipped when in check)
15150 '
15151 If EvalScore < Alpha - 450 - 250 * Depth * Depth Then
15152     Score = QSearch(ss, NON_PV_NODE, Alpha - 1, Alpha, MAX_DEPTH, PrevMove, QS_CHECKS)
15153     If Score < Alpha Then
15154         CutOffCnt(ss) = CutOffCnt(ss) - 1
15155         Search = Score
15156         Exit Function
15157     End If
15158 End If
15159 '
15160 '--- Step 8. Futility pruning: child node (skipped when in check)
15161 '
15162 If Not PVNode And Depth < 9 And EvalScore > Beta And EvalScore < VALUE_KNOWN_WIN + 1
Then ' >=beta bad? Different to SF
15163     If EvalScore - FutilityMargin(Depth, Improving) - StatScore(ss - 1) \ 280 >= Beta
Then
15164         Search = EvalScore
15165         Exit Function
15166     End If
15167 End If
15168
15169 '
15170 '--- Step 9. NULL MOVE -----
15171 '
15172 If Not PVNode And PrevMove.From > 0 And PrevMoveExtension = 0 And EvalScore >= Beta
And EvalScore >= StaticEval Then
15173     If Not bIsNullMove And StatScore(ss - 1) < 18755 And ExcludedMove.From = 0 Then
15174         If Fifty < 80 And Abs(Beta) < VALUE_KNOWN_WIN And Abs(StaticEval) < 2 *
VALUE_KNOWN_WIN And Alpha <> DrawContempt - 1 Then
15175             If (StaticEval >= Beta - (35 * Depth) + 222) Then
15176                 If (bWhiteToMove And WNonPawnPieces > 0) Or (Not bWhiteToMove And

```

```

BNonPawnPieces > 0) Then
15177 If Ply >= NullMovePly Then
15178 '--- Do NULLMOVE ---
15179 Dim bOldToMove As Boolean, OldPliesFromNull As Long
15180 bOldToMove = bWhiteToMove
15181 OldPliesFromNull = PliesFromNull: PliesFromNull = 0
15182 bWhiteToMove = Not bWhiteToMove 'MakeNullMove
15183 ClearMove BestMovePly(ss + 1): CmhPtr(ss) = 0: RemoveEpPiece: ClearMove
    MovesList(ss)
15184 Ply = Ply + 1: EpPosArr(Ply) = 0: Fifty = Fifty + 1: ClearMove CurrentMove:
    MovePickerDat(ss).CurrMoveNum = 0
15185 Debug.Assert EvalScore - Beta >= 0
15186
15187 '--- Stockfish
15188 r = GetMin((EvalScore - Beta) \ 168, 6) + Depth \ 3 + 4
15189 If Depth - r <= 0 Then
15190     NullScore = -QSearch(ss + 1, NON_PV_NODE, -Beta, -Beta + 1, MAX_DEPTH,
        CurrentMove, QS_CHECKS)
15191 Else
15192     NullScore = -Search(ss + 1, NON_PV_NODE, -Beta, -Beta + 1, Depth - r,
        CurrentMove, EmptyMove, Not CutNode, 0)
15193 End If
15194 Call RemoveEpPiece: Ply = Ply - 1: ResetEpPiece: Fifty = Fifty - 1: CmhPtr(
    ss) = 0: PliesFromNull = OldPliesFromNull
15195
15196 ' UnMake NullMove
15197 bWhiteToMove = bOldToMove
15198 If bTimeExit Then Search = 0: Exit Function
15199
15200 If NullScore < -MATE_IN_MAX_PLY Then ' Mate threat : own extra logic
15201     SetMove ThreatMove, BestMovePly(ss + 1)
15202     lPlyExtension = 1: GoTo lblMovesLoop
15203 End If
15204
15205 If NullScore >= Beta Then
15206     If NullScore >= MATE_IN_MAX_PLY Then NullScore = Beta
15207
15208     If NullMovePly <> 0 Or (Abs(Beta) < VALUE_KNOWN_WIN And Depth < 12) Then
15209         Search = NullScore
15210         Exit Function
15211     End If
15212
15213     '
15214     ' Do verification search at high depths
15215     '
15216     NullMovePly = Ply + 3 * (Depth - r) \ 4 ' search depth for verification
15217     If Depth - r <= 0 Then
15218         Score = QSearch(ss, NON_PV_NODE, Beta - 1, Beta, MAX_DEPTH, PrevMove,
            QS_CHECKS)
15219     Else
15220         Score = Search(ss, NON_PV_NODE, Beta - 1, Beta, Depth - r, PrevMove,
            EmptyMove, False, 0)
15221     End If
15222     NullMovePly = 0
15223     If Score >= Beta Then
15224         Search = NullScore
15225         Exit Function '--- Return Null Score, not Score!
15226     End If
15227
15228 End If
15229
15230 '--- Capture Threat? ( not SF logic )
15231 If BestMovePly(ss + 1).From <> 0 Then
15232     If (BestMovePly(ss + 1).Captured <> NO_PIECE Or NullScore < -
        MATE_IN_MAX_PLY) Then
15233         If Board(BestMovePly(ss + 1).Target) = BestMovePly(ss + 1).Captured Then
15234             ' not changed by previous move
            SetMove ThreatMove, BestMovePly(ss + 1)

```

```

15235         End If
15236     End If
15237 End If
15238
15239     End If ' Ply >= NullMovePly
15240 End If
15241 End If
15242 End If
15243 End If
15244 End If
15245 '
15246 '--- Step 10. ProbCut (skipped when in check)
15247 '
15248 ' If we have a very good capture (i.e. SEE > seeValues[captured_piece_type])
15249 ' and a reduced search returns a value much above beta, we can (almost) safely prune the previous move.
15250 If Not PVNode And Depth > 4 And PrevMove.Target > 0 Then
15251
15252     If Abs(Beta) < MATE_IN_MAX_PLY And Abs(StaticEval) < 2 * VALUE_KNOWN_WIN Then
15253         rBeta = GetMin(Beta + 186 - 54 * Improving, MATE0)
15254
15255         If Not (ttHit And HashDepth >= Depth - 3 And ttValue <> VALUE_NONE And ttValue <
15256             rBeta) Then '+++2023+++
15257
15258             Debug.Assert PrevMove.Target > 0
15259             MovePickerInit ss, ttMove, PrevMove, ThreatMove, True, False,
15260                 GENERATE_ALL_MOVES
15261
15262             Do While MovePicker(ss, CurrentMove, LegalMovesOutOfCheck)
15263                 If CurrentMove.Captured <> NO_PIECE Or CurrentMove.Promoted > 0 Then
15264                     If ExcludedMove.From <> 0 Then If MovesEqual(ExcludedMove, CurrentMove)
15265                         Then GoTo lblNextProbCut
15266                     rDepth = Depth - 4
15267                     Debug.Assert rDepth >= 1
15268                     '--- do the current move on the board -----
15269                     CmhPtr(ss) = CurrentMove.Piece * MAX_BOARD + CurrentMove.Target
15270                     Call RemoveEpPiece: MakeMove CurrentMove: Ply = Ply + 1
15271                     bLegalMove = False
15272                     If CheckLegal(CurrentMove) Then
15273                         bLegalMove = True: SetMove MovesList(ss), CurrentMove
15274                         ' Perform a preliminary qsearch to verify that the move holds
15275                         Score = -QSearch(ss + 1, NON_PV_NODE, -rBeta, -rBeta + 1, MAX_DEPTH,
15276                             CurrentMove, QS_CHECKS)
15277                         ' If the qsearch held perform the regular search
15278                         If Score >= rBeta Then
15279                             Score = -Search(ss + 1, NON_PV_NODE, -rBeta, -rBeta + 1, rDepth,
15280                                 CurrentMove, EmptyMove, Not CutNode, 0)
15281                         End If
15282                     End If
15283                     '--- Undo move -----
15284                     Call RemoveEpPiece: Ply = Ply - 1: UnmakeMove CurrentMove: ResetEpPiece
15285
15286                     If Score >= rBeta And bLegalMove Then
15287                         HashTableSave Hashkey, Depth - 3, CurrentMove, TT_LOWER_BOUND, Score,
15288                             StaticEval, ttPv
15289                         SetMove BestMovePly(ss), CurrentMove
15290                         Search = Score
15291                         Exit Function '---<<< Return
15292                     End If
15293                 End If
15294             End If
15295             lblNextProbCut:
15296             Loop ' While MovePicker
15297
15298         End If
15299     End If
15300 End If
15301
15302 lblSkipEarlyPruning:
15303 '

```

```

15297 ' Step 11. If the position is not in TT, decrease depth by 3.
15298 '
15299 ' Use qsearch if depth is equal or below zero (~9 Elo)
15300 If PVNode And ttMove.From = 0 Then
15301     Depth = Depth - (2 + 2 * Abs(ttHit And HashDepth >= Depth))
15302     If Depth <= 0 Then
15303         Search = QSearch(ss, PVNode, Alpha, Beta, MAX_DEPTH, PrevMove, QS_CHECKS)
15304         Exit Function '<<<<<<< R E T U R N >>>>>>>'
15305     End If
15306 End If
15307
15308 If CutNode And Depth >= 7 And ttMove.From = 0 Then
15309     Depth = Depth - 2 'never zero
15310 End If
15311 '
15312 '--- Moves Loop -----
15313 '
15314 lblMovesLoop:
15315
15316 ' Probcut idea
15317 rBeta = Beta + 391
15318 If PrevMove.IsChecking And Not PVNode And Depth >= 2 Then
15319     If bTTCapture And CBool(HashEvalType And TT_LOWER_BOUND) And ttValue >= rBeta And
15320     HashDepth >= Depth - 3 Then
15321         If Abs(ttValue) <= VALUE_KNOWN_WIN And Abs(Beta) <= VALUE_KNOWN_WIN Then
15322             Search = rBeta
15323             Exit Function '<<<<<<< R E T U R N >>>>>>>'
15324         End If
15325     End If
15326 End If
15327 '-----
15328
15329 Dim DrawMoveBonus As Long
15330 DrawMoveBonus = DrawValueForSide(bWhiteToMove)
15331 bSkipQuiets = False
15332
15333 '
15334 '---- Singular extension search.
15335 '
15336 bTTMoveIsSingular = False
15337 lSingularExtension = 0
15338 If ttMove.From > 0 And ExcludedMove.From = 0 And HashDepth >= Depth - 3 Then
15339     bSingularExtensionNode = (Ply < RootDepth * 2) And (Depth >= 4 - Abs(RootDepth - 1
15340     > 20) + 2 * Abs(PVNode And ttPv)) _
15341     And Abs(ttValue) < VALUE_KNOWN_WIN And CBool(HashEvalType
15342     And TT_LOWER_BOUND)
15343 Else
15344     bSingularExtensionNode = False
15345 End If
15346
15347 '--- SF logic (but moved before moves loop too avoid recursive call problems)
15348 If bSingularExtensionNode Then
15349     If MovePossible(ttMove) Then
15350         '--- Current move excluded
15351         '--- Make move -
15352         Call RemoveEpPiece: MakeMove ttMove: Ply = Ply + 1
15353         bLegalMove = CheckLegal(ttMove)
15354         '--- Undo move -----
15355         Call RemoveEpPiece: Ply = Ply - 1: UnmakeMove ttMove: ResetEpPiece
15356
15357         If bLegalMove Then
15358             rBeta = GetMax(ttValue - ((3 + 2 * Abs(ttPv And Not PVNode)) * Depth) \ 2, -
15359             MATE0)
15360             'rBeta = GetMax(ttValue - ((82 + 65 * Abs(ttPv And Not PVNode)) * Depth) \ 64, -MATE0)
15361
15362             Score = Search(ss, NON_PV_NODE, rBeta - 1, rBeta, (Depth - 1) \ 2, PrevMove,

```



```

ttMove, CutNode, 0)
DoubleExtensions(ss) = DoubleExtensions(ss - 1)

If Score < rBeta Then
    bTTMoveIsSingular = True
    If Not bTTCapture And Not bIsNullMove Then
        CounterMove(PrevMove.Piece, PrevMove.Target) = ttMove
    End If
    lSingularExtension = 1
    bSingularQuietLMR = Not bTTCapture

    '(better for tactic but worse in game???) '+++SING2
    ' If Not PVNode And Score < rBeta - 25 And DoubleExtensions(ss) <= 10 And DoubleExtensions(ss) <=
    1 + (RootDepth \ 12) Then ' Avoid search explosion
    If Not PVNode And Score < rBeta - 25 And DoubleExtensions(ss) <= 10 Then '
        Avoid search explosion
        lSingularExtension = 2
        If Depth < 13 Then Depth = Depth + 1
    End If

ElseIf rBeta >= Beta Then
    Search = rBeta
    BestMovePly(ss) = ttMove
    Exit Function
ElseIf ttValue >= Beta Then
    lSingularExtension = -2 - Abs(Not PVNode)

' If Depth + lSingularExtension < HashDepth And Not PVNode Then
' Search = ttValue
' Exit Function
' End If

ElseIf CutNode Then
    If Depth < 17 Then lSingularExtension = -3 Else lSingularExtension = -1
    ElseIf ttValue <= Score Then
        lSingularExtension = -1
    End If
End If ' bLegalMove
End If ' MovePossible
End If ' bSingularExtensionNode

'-----
'--- Capture Threat? ( not SF logic )
If ThreatMove.From = 0 Then
    If BestMovePly(ss + 1).From <> 0 Then
        If (BestMovePly(ss + 1).Captured <> NO_PIECE) Then
            If Board(BestMovePly(ss + 1).Target) = BestMovePly(ss + 1).Captured Then 'not
            changed by previous move
                If Board(BestMovePly(ss + 1).From) = BestMovePly(ss + 1).Piece Then
                    bWhiteToMove = Not bWhiteToMove
                    If MovePossible(BestMovePly(ss + 1)) Then
                        SetMove ThreatMove, BestMovePly(ss + 1)
                    End If
                End If
            End If
        End If
    End If
End If

'-----
'---- Step 12. Loop through moves -----
'-----
PVLength(ss) = ss
LegalMoveCnt = 0: QuietMoves = 0: CaptureMoves = 0: MoveCnt = 0
If ttMove.From > 0 Then SetMove TryBestMove, ttMove Else ClearMove TryBestMove
'

```

```

15425 ' Init MovePicker -----
15426 '
15427 MovePickerInit ss, TryBestMove, PrevMove, ThreatMove, False, False,
GENERATE_ALL_MOVES
15428 Score = BestValue
15429 ' Set move history pointer
15430 CmH = CmHPtr(ss - 1): Cm_Ok = (MovesList(ss - 1).From > 0)
15431 Fmh1 = 0: Fmh3 = 0 ' follow up moves
15432 If ss > 2 Then Fmh1 = CmHPtr(ss - 2): If ss > 4 Then Fmh3 = CmHPtr(ss - 4)
15433
15434 bMoveCountPruning = False
15435 bSingularQuietLMR = False
15436 bLikelyFailLow = (PVNode And ttMove.From <> 0 And CBool(HashEvalType And
TT_UPPER_BOUND) And HashDepth >= Depth)
15437 '
15438 '--- Loop over moves -----
15439 '
15440 Do While MovePicker(ss, CurrentMove, LegalMovesOutOfCheck)
15441   If ExcludedMove.From > 0 Then If MovesEqual(CurrentMove, ExcludedMove) Then GoTo
lblNextMove ' skip excluded move
15442   If PrevMove.IsChecking Then If Not CurrentMove.IsLegal Then GoTo lblNextMove '---
Legality for checks already tested in Ordermoves!
15443   bLegalMove = False: MoveCnt = MoveCnt + 1
15444   'Debug.Print "Search:" & RootDepth & ", ss:" & ss & " " & MoveText(CurrentMove)
15445
15446   If EGTBMoveListCnt(ss) > 0 Then '--- move from tablebases?
15447     ' Filter for endgame tablebase move: Ignore losing moves if draw or win from tablebases
15448     MoveStr = CompToCoord(CurrentMove)
15449     For r = 1 To EGTBMoveListCnt(ss)
15450       If MoveStr = EGTBMoveList(ss, r) Then GoTo lblEGMoveOK
15451     Next
15452     GoTo lblNextMove
15453   End If
15454   lblEGMoveOK:
15455
15456   '--- set pointer to history statistics
15457   CurrPtr = CurrentMove.Piece * MAX_BOARD + CurrentMove.Target
15458   CmHPtr(ss) = CurrPtr
15459
15460   '--- move count pruning / specifix login for ChessBrainVB: examine more moves if draw score
15461   bMoveCountPruning = Depth < 15 And MoveCnt >= FutilityMoveCnt(Improving, Depth) +
Abs(Abs(BestValue) = DrawMoveBonus And BestValue > StaticEval) * 10
15462   bCaptureOrPromotion = (CurrentMove.Captured <> NO_PIECE Or CurrentMove.Promoted <>
0)
15463   bKillerMove = IsKiller1Move(ss, CurrentMove)
15464   lExtension = 0
15465   NewDepth = Depth - 1
15466   '
15467   '--- Step 14. Pruning at shallow depth -----
15468   '
15469   r = Reduction(Improving, Depth, LegalMoveCnt, (Beta - Alpha), RootDelta) ' depth
reduction depending on depth and move counter
15470
15471   '--- Step 14. Pruning at shallow depth
15472   If BestValue > -MATE_IN_MAX_PLY Then
15473     ' reduce depth for next Late Move Reduction search
15474     LmrDepth = GetMax(NewDepth - r, 0)
15475
15476     If bCaptureOrPromotion Or CurrentMove.IsChecking Or AdvancedPawnPush(
CurrentMove.Piece, CurrentMove.Target) Then
15477       ' Capture or check
15478       If Not CurrentMove.IsChecking And Not PrevMove.IsChecking And Not PVNode And
LmrDepth < 7 Then
15479         If StaticEval + 182 + 230 * LmrDepth + PieceAbsValue(CurrentMove.Captured) +
CaptureHistory(CurrentMove.Piece, CurrentMove.Target, CurrentMove.Captured)
\ 7 < Alpha Then
15480           GoTo lblNextMove
15481         End If

```

```

15482 End If
15483 If Not SEEGreaterOrEqual(CurrentMove, -206 * Depth) Then GoTo lblNextMove '
    piece can be captured?
15484
15485 Else '--- not a capture > quiet move -----
15486
15487     If Not bKillerMove And bMoveCountPruning Then
15488     '
15489     ' Threat move logic specific to ChessBrainVB
15490     '
15491     With BestMovePly(ss + 1) ' new threat move?
15492     If .From > 0 And .Captured <> NO_PIECE Then
15493     If ThreatMove.From <> .From And ThreatMove.Target <> .Target Then
15494     If Board(.Target) = .Captured Then
15495     If BestMovePly(ss).From <> 0 And BestMovePly(ss).Target <> .Target
15496     And BestMovePly(ss).Target <> .From Then ' not changed by previous move
15497     SetMove ThreatMove, BestMovePly(ss + 1) ' new threat move
15498     End If
15499     End If
15500     End If
15501     End With
15502     If ThreatMove.From > 0 Then ' try to avoid threat move
15503     ' don't skip threat escape
15504     If CurrentMove.From <> ThreatMove.Target Then ' threat escape?
15505     ' blocking threat move makes sense only with less or equal valuable piece
15506     If (PieceAbsValue(CurrentMove.Piece) - 80 < PieceAbsValue(
15507     ThreatMove.Piece)) Then
15508     If IsBlockingMove(ThreatMove, CurrentMove) Then
15509     ' blocking move - so do NOT skip this move
15510     'Debug.Print PrintPos, MoveText(ThreatMove), MoveText(CurrentMove) : Stop
15511     Else
15512     bSkipQuiets = True
15513     GoTo lblNextMove ' skip this move, not a threat move defeat
15514     End If
15515     End If
15516     End If
15517     Else
15518     bSkipQuiets = True
15519     GoTo lblNextMove ' not a threat move
15520     End If ' ThreatMove.From
15521 End If ' Not bKillerMove
15522
15523 '--- ContinuationHistory based pruning
15524 HistoryVal = 0
15525 If CmH > 0 Then HistoryVal = HistoryVal + ContinuationHistory(CmH, CurrPtr)
15526 If Fmh1 > 0 Then HistoryVal = HistoryVal + ContinuationHistory(Fmh1, CurrPtr)
15527 If FMh3 > 0 Then HistoryVal = HistoryVal + ContinuationHistory(FMh3, CurrPtr)
15528
15529 If LmrDepth < 5 And HistoryVal < -4405 * (Depth - 1) Then GoTo lblNextMove
15530
15531 HistoryVal = HistoryVal + 2 * History(PieceColor(CurrentMove.Piece),
15532 CurrentMove.From, CurrentMove.Target)
15533 LmrDepth = LmrDepth + HistoryVal \ 7278
15534 LmrDepth = GetMax(LmrDepth, -2)
15535
15536 Dim FutilVal As Long
15537 FutilVal = StaticEval + 104 + 145 * LmrDepth + HistoryVal \ 52
15538 If Not PrevMove.IsChecking And LmrDepth < 13 Then
15539     If FutilVal <= Alpha Then
15540     GoTo lblNextMove
15541     ElseIf FutilVal <= Alpha + 20 Then
15542     bAlmostFutilPruned = True
15543     End If
15544 End If
15545 LmrDepth = GetMax(LmrDepth, 0)

```

```

15546      '--- SEE based LMP
15547      If Not SEEGreaterOrEqual(CurrentMove, -24 * LmrDepth * LmrDepth - 16 *
      LmrDepth) Then GoTo lblNextMove
15548      End If ' bCaptureOrPromotion
15549
15550  End If ' BestValue
15551
15552  '
15553  '--- Step 13. Extensions
15554  '
15555  DoubleExtensions(ss) = DoubleExtensions(ss - 1) ' may be overwritten in searches before
15556
15557  'if We take care to not overdo to avoid search getting stuck.
15558  If Ply + 1 < RootDepth * 2 Then
15559
15560      '- Singular move extent first , extension may be > 1 or < 0
15561      If lSingularExtension <> 0 And MoveCnt = 1 Then
15562          If MovesEqual(CurrentMove, ttMove) Then lExtension = lSingularExtension: GoTo
          lblEndExtensions
15563      End If
15564
15565      '- Mate threat extent
15566      If lPlyExtension > 0 Then lExtension = 1: GoTo lblEndExtensions
15567
15568      '- Single move check escape extent
15569      If (PrevMove.IsChecking) Then
15570          If LegalMovesOutOfCheck <= 1 Then lExtension = 1: GoTo lblEndExtensions
15571      End If
15572
15573      '- Checking extension ---
15574      If (CurrentMove.IsChecking) Then
15575          If Depth > 10 And Abs(StaticEval) > 88 Then
15576              lExtension = 1: GoTo lblEndExtensions
15577          End If
15578      End If
15579
15580      '- Queen exchange extent
15581      If Depth < 12 Then
15582          If PieceType(CurrentMove.Captured) = PT_QUEEN Then
15583              If PieceType(CurrentMove.Piece) = PT_QUEEN Then lExtension = 1: GoTo
              lblEndExtensions
15584          End If
15585      End If
15586
15587      '- Castling extent
15588      If CurrentMove.Castle <> NO_CASTLE Then
15589          lExtension = 1: GoTo lblEndExtensions
15590      End If
15591
15592      '- Good killer move extent
15593      If PVNode And bKillerMove Then
15594          If CmH > 0 And ttMove.From > 0 Then
15595              If MovesEqual(CurrentMove, ttMove) Then
15596                  If ContinuationHistory(CmH, CurrPtr) > 5705 Then lExtension = 1: GoTo
                  lblEndExtensions
15597              End If
15598          End If
15599      End If
15600
15601      '- Passed pawn move extent
15602      If PieceType(CurrentMove.Captured) = PT_PAWN Then
15603          If AdvancedPassedPawnPush(CurrentMove.Piece, CurrentMove.Target) Then
15604              lExtension = 1: GoTo lblEndExtensions
15605          End If
15606      End If ' Ply < RootDepth * 2
15607  lblEndExtensions:
15608

```

```

15609 '- Add extensions to new depth for this move
15610 NewDepth = GetMax(0, NewDepth + lExtension)
15611 DoubleExtensions(ss) = DoubleExtensions(ss - 1) + Abs(lExtension >= 2)
15612
15613 '-----
15614 '--- Step 15. Make move -
15615 '-----
15616 Call RemoveEpPiece: MakeMove CurrentMove: Ply = Ply + 1
15617 If Not PrevMove.IsChecking And CurrentMove.Castle = NO_CASTLE Then
15618     CurrentMove.IsLegal = CheckLegalNotInCheck(CurrentMove)
15619 ' If CurrentMove.IsLegal Then ' verify correctness
15620 '     If Not CheckLegal(CurrentMove) Then WriteTrace PrintPos & MoveText(PrevMove) & " " &
MoveText(CurrentMove): MsgBox "C1": Stop: End
15621 ' Else
15622 '     If CheckLegal(CurrentMove) Then WriteTrace PrintPos: MsgBox "C2": Stop: End
15623 ' End If
15624 ElseIf Not CurrentMove.IsLegal Then
15625     CurrentMove.IsLegal = CheckLegal(CurrentMove)
15626 End If
15627
15628 '- move is legal
15629 If CurrentMove.IsLegal Then
15630     Nodes = Nodes + 1: LegalMoveCnt = LegalMoveCnt + 1
15631
15632     #If DEBUG_MODE <> 0 Then
15633         If (Nodes \ 1000) Mod 5 = 0 Then DoEvents ' allow break in debug mode
15634     #End If
15635
15636     bNoMoves = False: bLegalMove = True
15637     SetMove MovesList(ss), CurrentMove
15638
15639     '--- Step 16. Reduced depth search (LMR). If the move fails high it will be re-searched at full depth.
15640
15641     r = Reduction(Improving, Depth, LegalMoveCnt, (Beta - Alpha), RootDelta)
15642
15643     If ttPv And Not bLikelyFailLow Then
15644         r = r - 2
15645     Else
15646         If bAlmostFutilPruned Then r = r - 1
15647     End If
15648     If MovePickerDat(ss - 1).CurrMoveNum > 7 Then r = r - 1 'Decrease reduction if
opponent's move count is high
15649     If CutNode Then
15650         r = r + 2
15651     ElseIf CurrentMove.Castle = NO_CASTLE Then
15652         '--- Decrease reduction for moves that escape a capture
15653         TmpMove.From = CurrentMove.Target: TmpMove.Target = CurrentMove.From:
15654         TmpMove.Piece = CurrentMove.Piece: TmpMove.Captured = NO_PIECE:
15655         TmpMove.SeeValue = VALUE_NONE
15656         ' Move back to old square, were we in danger there?
15657         If Not SEEGreaterOrEqual(TmpMove, -MAX_SEE_DIFF) Then r = r - 2 ' old square was
dangerous
15658     End If
15659     If bTTCapture Then r = r + 1 ' If TTMove was a capture, quiets rarely are better
15660     If PVNode Then If Depth > 0 Then r = r - (1 + 12 \ (3 + Depth)) ' PV node deeper
15661     If bSingularQuietLMR Then r = r - 1 ' quiet singular move
15662     If CutOffCnt(ss + 1) > 3 Then r = r + 1 ' many cutoffs for next ply
15663     If ttMove.From <> 0 Then If MovesEqual(CurrentMove, ttMove) Then r = r - 1 ' TT
move
15664     If bKillerMove And CmH > 0 Then ' Good killer move
15665         If ContinuationHistory(CmH, CurrPtr) >= 3722 Then r = r - 1
15666     End If
15667
15668 ' If ss > 2 And Fifty > 3 Then
15669 '     If CurrentMove.From = MovesList(ss - 2).Target Then
15670 '         If CurrentMove.Target = MovesList(ss - 2).From Then
15671 '             r = r + 1
15672 '         If ss > 4 Then If MovesEqual(CurrentMove, MovesList(ss - 4)) Then r = r + 1

```

```

15671 ' End If
15672 ' End If
15673 ' End If
15674
15675 ' If ss > 4 And Fifty > 4 Then ' repeated move
15676 '   If MovesEqual(CurrentMove, MovesList(ss - 4)) Then r = r + 2 ': TestCnt(1) = TestCnt(1) + 1
15677 ' End If
15678 ' If CutOffCnt(ss + 1) > 3 Then
15679 '   r = r + 1 ' many cutoffs
15680 ' Elseif ttMove.From <> 0 Then
15681 '   If MovesEqual(CurrentMove, ttMove) Then r = r - 1 ' TT move
15682 ' End If
15683
15684 '
15685 HistVal = 2 * History(PieceColor(CurrentMove.Piece), CurrentMove.From,
15686 CurrentMove.Target)
15687 If CmH > 0 Then HistVal = HistVal + ContinuationHistory(CmH, CurrPtr)
15688 If Fmh1 > 0 Then HistVal = HistVal + ContinuationHistory(Fmh1, CurrPtr)
15689 If FMh3 > 0 Then HistVal = HistVal + ContinuationHistory(FMh3, CurrPtr)
15690 StatScore(ss) = HistVal - 4082
15691
15692 '--- Decrease/increase reduction by comparing opponent's stat score
15693 If StatScore(ss) >= 0 And StatScore(ss - 1) < 0 Then
15694   r = r - 1
15695   If StatScore(ss) > StatScore(ss - 1) + 5000 Then r = r - 1
15696 ElseIf StatScore(ss - 1) >= 0 And StatScore(ss) < 0 Then
15697   r = r + 1
15698   If StatScore(ss) < StatScore(ss - 1) - 5000 Then r = r + 1
15699 End If
15700
15701 '--- Decrease/increase reduction for moves with a good/bad history
15702 Factor = 11111 + 4700 * Abs(Depth > 5 And Depth < 22)
15703 r = r - StatScore(ss) \ Factor
15704 If r < 0 Then r = 0 '?! if r<0 search explosions
15705 lblNoMoreReductions:
15706 '----- Step 17. Late moves reduction / extension
15707 If Depth >= 2 And LegalMoveCnt > 1 + Abs(PVNode) And _
15708   (Not ttPv Or Not bCaptureOrPromotion Or (CutNode And MovePickerDat(ss - 1).
15709     CurrMoveNum >= 1)) Then
15710   Depth1 = NewDepth - r
15711   If Depth1 < 1 Then Depth1 = 1 Else If Depth1 > NewDepth + 1 Then Depth1 =
15712     NewDepth + 1
15713   'rBeta = Abs(StaticEval >= Alpha - 81 * Depth)
15714   'If Depth1 < rBeta Then Depth1 = rBeta Else If Depth1 > NewDepth + 1 Then Depth1 = NewDepth + 1
15715
15716 '--- Reduced SEARCH -----
15717 Score = -Search(ss + 1, NON_PV_NODE, -(Alpha + 1), -Alpha, Depth1, CurrentMove
15718 , EmptyMove, True, lExtension)
15719 If (Score > Alpha And Depth1 < NewDepth) Then
15720   Dim bDoDeeperSearch As Boolean, bDoEvenDeeperSearch As Boolean,
15721   bDoShallowerSearch As Boolean
15722   bDoDeeperSearch = (Score > (Alpha + 58 + 12 * (NewDepth - Depth1)))
15723   bDoEvenDeeperSearch = (Score > Alpha + 588 And DoubleExtensions(ss) <= 5)
15724   bDoShallowerSearch = (Score < BestValue + NewDepth)
15725
15726   DoubleExtensions(ss) = DoubleExtensions(ss) + Abs(bDoEvenDeeperSearch)
15727
15728   NewDepth = NewDepth + Abs(bDoDeeperSearch) - Abs(bDoShallowerSearch) + Abs(
15729     bDoEvenDeeperSearch)
15730   If NewDepth > Depth1 Then
15731     Score = -Search(ss + 1, NON_PV_NODE, -(Alpha + 1), -Alpha, NewDepth,
15732       CurrentMove, EmptyMove, Not CutNode, lExtension)
15733   End If
15734   If Score <= Alpha Then
15735     Bonus = -StatBonus(Depth) ' better than NewDepth?
15736   ElseIf Score >= Beta Then
15737     Bonus = StatBonus(Depth)
15738   Else

```

```

15732         Bonus = 0
15733     End If
15734     UpdateContHistStats ss, CurrentMove.Piece, CurrentMove.Target, Bonus
15735 End If 'Score
15736
15737 ElseIf (Not PVNode Or LegalMoveCnt > 1) Then
15738     If ttMove.From = 0 And CutNode Then r = r + 2
15739     If NewDepth - Abs(r > 4) <= 0 Then
15740         Score = -QSearch(ss + 1, NON_PV_NODE, -(Alpha + 1), -Alpha, MAX_DEPTH,
            CurrentMove, QS_CHECKS)
15741     Else
15742         Score = -Search(ss + 1, NON_PV_NODE, -(Alpha + 1), -Alpha, NewDepth - Abs(
            r > 4), CurrentMove, EmptyMove, Not CutNode, lExtension)
15743     End If
15744 End If ' Depth >= 3 ...
15745
15746
15747 '-----
15748 '--->>> RECURSIVE MAIN SEARCH <<<----
15749 '-----
15750
15751 ' For PV nodes only, do a full PV search on the first move or after a fail
15752 ' high (in the latter case search only if value < beta), otherwise let the
15753 ' parent node fail low with value <= alpha and to try another move.
15754 If (PVNode And (LegalMoveCnt = 1 Or (Score > Alpha And Score < Beta))) And Not
    bTimeExit Then
15755     If NewDepth <= 0 Or (Ply + NewDepth >= MAX_DEPTH) Then
15756         Score = -QSearch(ss + 1, PV_NODE, -Beta, -Alpha, MAX_DEPTH, CurrentMove,
            QS_CHECKS)
15757     Else
15758         Score = -Search(ss + 1, PV_NODE, -Beta, -Alpha, NewDepth, CurrentMove,
            EmptyMove, False, lExtension)
15759     End If 'NewDepth
15760 End If 'PVNode
15761
15762 lblSkipMove:
15763 End If '--- CheckLegal
15764
15765 '-----
15766 '--- Step 18. Undo move --
15767 '-----
15768 Call RemoveEpPiece: Ply = Ply - 1: UnmakeMove CurrentMove: ResetEpPiece
15769
15770 If bTimeExit Then Search = 0: Exit Function
15771
15772 '-----
15773 '--- Step 19. Check for a new best move --
15774 '-----
15775 If Score > BestValue And bLegalMove Then
15776     BestValue = Score
15777
15778     If (Score > Alpha) Then
15779         GoodMoves = GoodMoves + 1
15780         SetMove BestMove, CurrentMove
15781         If PVNode Then UpdatePV ss, CurrentMove '--- Save PV ---
15782         If PVNode And Score < Beta Then
15783             If Depth > 1 And Depth < 6 And Beta < 10500 And Score > -10500 Then Depth =
                Depth - 1
15784             Alpha = Score
15785             Debug.Assert Depth > 0
15786         Else
15787             '--- Fail High ---
15788             CutOffCnt(ss) = CutOffCnt(ss) + 1
15789             If StatScore(ss) < 0 Then StatScore(ss) = 0
15790             Exit Do
15791         End If
15792     End If
15793 End If

```



```

15794
15795     If bLegalMove Then
15796         '--- Add Quiet move, used for pruning and history update
15797         If Not MovesEqual(BestMove, CurrentMove) Then
15798             If Not bCaptureOrPromotion And QuietMoves < 64 Then
15799                 QuietMoves = QuietMoves + 1: SetMove QuietsSearched(ss, QuietMoves),
                    CurrentMove
15800             ElseIf CurrentMove.Captured <> NO_PIECE And CaptureMoves < 32 Then
15801                 If Not MovesEqual(BestMove, CurrentMove) Then CaptureMoves = CaptureMoves + 1
                    : CapturesSearched(ss, CaptureMoves) = CurrentMove
15802             End If
15803         End If
15804     Else
15805         MoveCnt = MoveCnt - 1 ' not legal
15806     End If
15807 lblNextMove:
15808     Loop
15809     '-----
15810     '--- next move in search ---
15811     '-----
15812
15813     '-----
15814     '--- Step 20. Check for mate and stalemate ---
15815     '-----
15816     If bNoMoves Then
15817         Debug.Assert LegalMovesOutOfCheck = 0 Or ExcludedMove.From > 0
15818         If ExcludedMove.From > 0 Then
15819             BestValue = Alpha
15820         ElseIf InCheck() Then '-- mate - do check again to be sure
15821             BestValue = -MATE0 + Ply ' mate in N plies
15822         Else 'draw
15823             If CompToMove() Then BestValue = DrawContempt Else BestValue = -DrawContempt
15824         End If
15825     ElseIf BestMove.From > 0 Then
15826         '--- New best move
15827         SetMove BestMovePly(ss), BestMove
15828         UpdateStats ss, BestMove, BestValue, Beta, QuietMoves, CaptureMoves, PrevMove,
                    Depth + Abs((Not PVNode And Not CutNode) Or (BestValue > Beta + ScorePawn.MG))
15829
15830         '--- Extra penalty for a quiet TT move in previous ply when it gets refuted
15831         If PrevMove.Captured = NO_PIECE Then
15832             If PrevMove.From > 0 And ss > 2 And CmH > 0 Then
15833                 If MovePickerDat(ss - 1).CurrMoveNum = 0 Or IsKiller1Move(ss - 1, CurrentMove)
                    Then
15834                     UpdateContHistStats ss - 1, PrevMove.Piece, PrevMove.Target, -StatBonus(
                        Depth + 1)
15835                 End If
15836             End If
15837         End If
15838     Else
15839         '--- failed low - no best move
15840         ClearMove BestMovePly(ss)
15841         ' Bonus for prior countermove that caused the fail low
15842         If Depth >= 3 Or PVNode Then
15843             If PrevMove.Captured = NO_PIECE Then
15844                 If Cm_Ok And ss > 2 Then
15845                     r = Abs(Depth > 5) + Abs(PVNode Or CutNode) + Abs(BestValue < Alpha - 97 *
                        Depth) + Abs(MovePickerDat(ss - 1).CurrMoveNum > 10)
15846                     UpdateContHistStats ss - 1, PrevMove.Piece, PrevMove.Target, StatBonus(Depth
                        ) * r
15847                     'UpdHistory PrevMove.Piece, PrevMove.From, PrevMove.Target, StatBonus(Depth) * r * 3 \ 5
15848                 End If
15849             End If
15850         End If
15851     End If
15852 End If
15853
15854 If Fifty > 99 Then ' Draw 50 moves rule ?
    If CompToMove() Then BestValue = DrawContempt Else BestValue = -DrawContempt

```

```

15855 End If
15856
15857 If BestValue <= Alpha Then ' add to pv?
15858     ttPv = ttPv Or (ttPVArr(ss - 1) And Depth > 3): ttPVArr(ss) = ttPv
15859 End If
15860
15861 If ExcludedMove.From = 0 Then
15862     '-----
15863     '--- Save hash values for best move ---
15864     '-----
15865     If BestValue >= Beta Then
15866         HashEvalType = TT_LOWER_BOUND
15867     ElseIf PVNode And BestMove.From >= SQ_A1 Then
15868         HashEvalType = TT_EXACT
15869     Else
15870         HashEvalType = TT_UPPER_BOUND
15871     End If
15872
15873     If BestValue = DrawMoveBonus Then Depth1 = GetMin(4, Depth) Else Depth1 = Depth
15874     HashTableSave Hashkey, Depth1, BestMove, HashEvalType, BestValue, StaticEval, ttPv
15875     'Save eval in hash table
15876 End If
15877
15878 Search = BestValue ' return best score for search. Best move is saved in BestMovePly(ss) and PV.
15879
15880 End Function
15881 '=====
15882 '=====
15883
15884
15885 '=====
15886 '=====
15887
15888 '= QSearch (Quiescence Search): search for quiet position until no more capture possible, =
15889 '= finally calls position evaluation =
15890 '= called by SEARCH, calls QSEARCH recursively , then EVAL =
15891 '=====
15892 '=====
15893
15894 Private Function QSearch(ByVal ss As Long, _
15895     ByVal PVNode As Boolean, _
15896     ByVal Alpha As Long, _
15897     ByVal Beta As Long, _
15898     ByVal Depth As Long, _
15899     InPrevMove As TMOVE, _
15900     ByVal GenerateQSChecks As Boolean) As Long
15901
15902     Dim PrevMove As TMOVE, Hashkey As THashKey, HashMove As TMOVE, bHashBoardDone As
15903     Boolean, ttDepth As Long, MoveCnt As Long, LegalMovesOutOfCheck As Long
15904     Dim bHashFound As Boolean, ttHit As Boolean, HashEvalType As Long, HashScore As
15905     Long, HashStaticEval As Long, HashDepth As Long, HashPvHit As Boolean, ttPv As
15906     Boolean, HashThreadNum As Long
15907     If ss > MAX_DEPTH Then MsgBox "SS overflow:" & ss
15908
15909     QSDepth = QSDepth + 1: If QSDepth > QSDepthMax Then QSDepthMax = QSDepth
15910     ClearMove BestMovePly(ss)
15911
15912     If Not PVNode Then GenerateQSChecks = False ' QSChecks for PVNodes in first QS ply only because
15913     slow
15914
15915     SetMove PrevMove, InPrevMove: HashScore = VALUE_NONE
15916     bHashFound = False: ttHit = False: ClearMove HashMove: bHashBoardDone = False
15917     If Fifty > 99 Then ' Draw ?
15918         If CompToMove() Then QSearch = DrawContempt Else QSearch = -DrawContempt
15919         QSDepth = QSDepth - 1
15920     Exit Function
15921 End If

```

```

15914
15915 If Fifty >= 3 And PliesFromNull >= 3 Then
15916     HashBoard Hashkey, EmptyMove: bHashBoardDone = True ' Save current keys for insert later
15917     If Is3xDraw(Hashkey, GameMovesCnt, Ply) Then
15918         If CompToMove() Then QSearch = DrawContempt Else QSearch = -DrawContempt
15919         QSDepth = QSDepth - 1
15920         Exit Function '-- Exit
15921     End If
15922 End If
15923
15924 If (Depth <= 0 Or Ply >= MAX_DEPTH) Then
15925     QSearch = Eval(): QSDepth = QSDepth - 1
15926     Exit Function '-- Exit
15927 End If
15928
15929 '--- Mate distance pruning
15930 ' Alpha = GetMax(-MATE0 + Ply, Alpha)
15931 ' Beta = GetMin(MATE0 - Ply, Beta)
15932 ' If Alpha >= Beta Then QSearch = Alpha: Exit Function
15933
15934 '--- Check Hash -----
15935 If Not bHashBoardDone Then HashBoard Hashkey, EmptyMove ' Save current keys for insert later
15936 GamePosHash(GameMovesCnt + Ply - 1) = Hashkey
15937
15938 If PrevMove.IsChecking Or GenerateQSChecks Then
15939     ttDepth = DEPTH_QS_CHECKS ' = 0
15940 Else
15941     ttDepth = DEPTH_QS_NO_CHECKS ' = -1
15942 End If
15943 ttHit = HashTableRead(Hashkey, HashDepth, HashMove, HashEvalType, HashScore,
15944 HashStaticEval, HashPvHit, HashThreadNum)
15945 ttPv = ttHit And HashPvHit
15946 If Not PVNode And ttHit Then
15947     If HashScore <> VALUE_NONE And HashDepth >= ttDepth Then
15948         If HashScore >= Beta Then
15949             bHashFound = (HashEvalType And TT_LOWER_BOUND)
15950         Else
15951             bHashFound = (HashEvalType And TT_UPPER_BOUND)
15952         End If
15953         If bHashFound Then
15954             SetMove BestMovePly(ss), HashMove
15955             QSearch = HashScore: QSDepth = QSDepth - 1
15956             Exit Function '-- Exit
15957         End If
15958     End If
15959 End If
15960
15961 '-----
15962 Dim CurrentMove As TMOVE, bNoMoves As Boolean, Score As Long, BestMove As TMOVE
15963 Dim bLegalMove As Boolean, FutilBase As Long, FutilScore As Long, StaticEval As
15964 Long, BestValue As Long
15965 Dim bCapturesOnly As Boolean
15966
15967 BestValue = -VALUE_INFINITE: StaticEval = VALUE_NONE
15968 If ttHit And HashMove.From > 0 Then SetMove BestMovePly(ss), HashMove Else ClearMove
15969 BestMovePly(ss)
15970 '-----
15971 If PrevMove.IsChecking Then
15972     FutilBase = -VALUE_INFINITE
15973     bCapturesOnly = False ' search all moves to prove mate
15974 Else
15975     '--- SEARCH CAPTURES ONLY ----
15976     If ttHit Then
15977         If HashStaticEval = VALUE_NONE Then
15978             StaticEval = Eval()
15979         Else
15980             StaticEval = HashStaticEval
15981         End If

```

```

15979     BestValue = StaticEval
15980     If HashScore <> VALUE_NONE Then
15981         If HashScore > BestValue Then
15982             If CBool(HashEvalType And TT_LOWER_BOUND) Then BestValue = HashScore
15983             Else
15984                 If CBool(HashEvalType And TT_UPPER_BOUND) Then BestValue = HashScore
15985             End If
15986         End If
15987     Else
15988         StaticEval = Eval()
15989         BestValue = StaticEval
15990     End If
15991     '--- Stand pat. Return immediately if static value is at least beta
15992     If BestValue >= Beta Then
15993         If Not ttHit Then
15994             HashTableSave Hashkey, DEPTH_NONE, EmptyMove, TT_LOWER_BOUND, BestValue,
15995                 StaticEval, False
15996             End If
15997             QSearch = BestValue: QSDepth = QSDepth - 1
15998             Exit Function '-- exit
15999         End If
16000         If PVNode And BestValue > Alpha Then Alpha = BestValue
16001         FutilBase = StaticEval + 200
16002         bCapturesOnly = True ' Captures only
16003     End If ' PrevMove.IsChecking
16004     StaticEvalArr(ss) = StaticEval
16005
16006     PVLength(ss) = ss: bNoMoves = True
16007     Dim QuietCheckEvasions As Long
16008     QuietCheckEvasions = 0
16009
16010     '
16011     '---- QSearch moves loop -----
16012     '
16013     ' New: Always use hash move
16014     If HashMove.From > 0 Then ' Hash move is capture or check ?
16015         If GenerateQSChecks And HashMove.IsChecking Then
16016             ' keep Hash move
16017         ElseIf bCapturesOnly And HashMove.Captured <> NO_PIECE Then
16018             ' keep Hash move
16019         Else
16020             ClearMove HashMove
16021         End If
16022     End If
16023
16024     Dim CmH As Long, Fmh As Long, CurrPtr As Long
16025     CmH = PrevMove.Piece * MAX_BOARD + PrevMove.Target
16026     If Ply > 2 Then Fmh = CmHPtr(Ply - 2) Else Fmh = 0
16027
16028     MovePickerInit ss, HashMove, PrevMove, EmptyMove, bCapturesOnly, False,
16029     GenerateQSChecks
16030
16031     Do While MovePicker(ss, CurrentMove, LegalMovesOutOfCheck)
16032         ' Debug.Print "QS:" & ss, MoveText(CurrentMove)
16033         MoveCnt = MoveCnt + 1
16034         If PrevMove.IsChecking Then
16035             If LegalMovesOutOfCheck = 0 Then
16036                 '--- Mate
16037                 QSearch = -MATE0 + Ply: QSDepth = QSDepth - 1
16038                 Exit Function
16039             Else
16040                 If Not CurrentMove.IsLegal Then GoTo lblNext
16041             End If
16042         ElseIf QSDepth > 6 Then ' recaptures only after 5 QS calls (starts with 1)
16043             If CurrentMove.Target <> PrevMove.Target Then GoTo lblNext
16044         End If

```

```

16045 '-----
16046 '--- Futil Pruning -
16047 '-----
16048 'If BestValue > -MATE_IN_MAX_PLY And ((bWhiteToMove And CBool(WNonPawnMaterial <> 0)) Or (Not
bWhiteToMove And CBool(BNonPawnMaterial <> 0))) Then
16049 If BestValue > -MATE_IN_MAX_PLY Then
16050     If Not CurrentMove.IsChecking And CurrentMove.Target <> PrevMove.Target And
FutilBase > -VALUE_KNOWN_WIN And CurrentMove.Promoted = 0 Then
16051         If MoveCnt > 2 Then GoTo lblNext
16052         FutilScore = FutilBase
16053         If CurrentMove.Captured <> NO_PIECE Then FutilScore = FutilScore +
PieceAbsValue(CurrentMove.Captured)
16054
16055         If FutilScore <= Alpha Then
16056             If FutilScore > BestValue Then BestValue = FutilScore
16057             GoTo lblNext
16058         End If
16059
16060         If FutilBase <= Alpha Then
16061             If Not SEEGreaterOrEqual(CurrentMove, 1) Then
16062                 If FutilBase > BestValue Then BestValue = FutilBase
16063                 GoTo lblNext
16064             End If
16065         End If
16066
16067         If FutilBase > Alpha Then
16068             If Not SEEGreaterOrEqual(CurrentMove, (Alpha - FutilBase) * 4) Then
16069                 BestValue = Alpha
16070                 GoTo lblNext
16071             End If
16072         End If
16073
16074     End If ' Not CurrentMove.IsChecking
16075     If QuietCheckEvasions > 1 Then Exit Do
16076
16077     ' Continuation history based pruning
16078     If CurrentMove.Captured = NO_PIECE Then
16079         If CmH > 0 Then
16080             CurrPtr = CurrentMove.Piece * MAX_BOARD + CurrentMove.Target
16081             If ContinuationHistory(CmH, CurrPtr) < 0 Then
16082                 If Fmh > 0 Then
16083                     If ContinuationHistory(Fmh, CurrPtr) < 0 Then
16084                         GoTo lblNext
16085                     End If
16086                 End If
16087             End If
16088         End If
16089     End If
16090
16091     ' Don't search moves with negative SEE values
16092     If Not SEEGreaterOrEqual(CurrentMove, -110) Then GoTo lblNext
16093 End If ' BestValue
16094
16095 If PrevMove.IsChecking Then If CurrentMove.Captured = NO_PIECE Then
QuietCheckEvasions = QuietCheckEvasions + 1
16096
16097 '-----
16098 '--- Do QS move -
16099 '-----
16100 CmHPtr(ss) = CurrentMove.Piece * MAX_BOARD + CurrentMove.Target
16101 Call RemoveEpPiece: MakeMove CurrentMove: Ply = Ply + 1: bLegalMove = False
16102
16103 If Not PrevMove.IsChecking And CurrentMove.Castle = NO_CASTLE Then
CurrentMove.IsLegal = CheckLegalNotInCheck(CurrentMove)
16104
' If CurrentMove.IsLegal Then ' verify correctness
' If Not CheckLegal(CurrentMove) Then WriteTrace PrintPos & MoveText(PrevMove) & " " &
MoveText(CurrentMove): MsgBox "C3": Stop: End
16107 ' Else

```

```

16108 ' If CheckLegal(CurrentMove) Then WriteTrace PrintPos: MsgBox "C4": Stop: End
16109 ' End If
16110 ElseIf Not CurrentMove.IsLegal Then
16111     CurrentMove.IsLegal = CheckLegal(CurrentMove)
16112 End If
16113
16114 If CurrentMove.IsLegal Then
16115     Nodes = Nodes + 1: QNodes = QNodes + 1: bLegalMove = True: bNoMoves = False
16116     SetMove MovesList(ss), CurrentMove
16117     '-----
16118     '--- QSearch recursive -----
16119     '-----
16120     Score = -QSearch(ss + 1, PVNode, -Beta, -Alpha, Depth - 1, CurrentMove,
16121         QS_NO_CHECKS)
16122 End If
16123
16124 '-----
16125 '--- Undo QS move -
16126 '-----
16127 Call RemoveEpPiece: Ply = Ply - 1: UnmakeMove CurrentMove: ResetEpPiece
16128
16129 ' check for best move
16130 If (Score > BestValue) And bLegalMove Then
16131     BestValue = Score
16132
16133     If Score > Alpha Then
16134         SetMove BestMove, CurrentMove
16135         SetMove BestMovePly(ss), CurrentMove
16136         'If bSearchingPV And PVNode Then UpdatePV ss, CurrentMove
16137         If Score < Beta Then
16138             Alpha = Score
16139         Else
16140             'If CutOffCnt(ss + 1) > 1 Then CutOffCnt(ss) = CutOffCnt(ss) + 1
16141             Exit Do '--- Fail high: >= Beta
16142         End If
16143     End If
16144 End If
16145 End If
16146 lblNext:
16147 Loop '--- QS moves
16148
16149 '--- Mate?
16150 If PrevMove.IsChecking And bNoMoves Then
16151     If InCheck() Then
16152         QSearch = -MATE0 + Ply ' mate in N plies, check again to be sure
16153         QSDepth = QSDepth - 1
16154         Exit Function
16155     End If
16156 End If
16157
16158 '--- Save Hash values ---
16159 If BestValue >= Beta Then HashEvalType = TT_LOWER_BOUND Else HashEvalType =
16160 TT_UPPER_BOUND
16161 HashTableSave Hashkey, ttDepth, BestMove, HashEvalType, BestValue, StaticEval, ttPv
16162 ' save eval in hash table
16163
16164 QSDepth = QSDepth - 1
16165 SetMove BestMovePly(ss), BestMove ' return QS best move
16166 QSearch = BestValue ' return QS score
16167 End Function
16168
16169 '=====
16170 '= OrderMoves() =
16171 '= Assign an order value to the generated moves =
16172 '=====
16173 Private Sub OrderMoves(ByVal Ply As Long, _
16174     ByVal NumMoves As Long, _
16175     PrevMove As TMOVE, _
16176     BestMove As TMOVE, _

```

```

16173             ThreatMove As TMOVE, _
16174             ByVal bCapturesOnly As Boolean, _
16175             LegalMovesOutOfCheck As Long)
16176
16177 Dim i As Long, From As Long, Target As Long, Promoted As Long,
Captured As Long, lValue As Long, Piece As Long, EnPassant As Long
16178 Dim bSearchingPVNew As Boolean, BestValue As Long, BestIndex As Long, WhiteMoves As
Boolean, CmH As Long
16179 Dim bLegalsOnly As Boolean, TmpVal As Long, PieceVal As Long, CounterMoveTmp As
TMOVE, KingLoc As Long, v As Long
16180 Dim Fm1 As Long, Fm2 As Long, Fm3 As Long, Fm5 As Long, CurrPtr As Long,
bIsChecking As Boolean
16181 '-----
16182 LegalMovesOutOfCheck = 0
16183 If NumMoves = 0 Then Exit Sub
16184 bSearchingPVNew = False
16185 BestValue = -9999999: BestIndex = -1 '--- save highest score
16186 WhiteMoves = CBool((Board(Moves(Ply, 0).From) And 1) = 1) 'to be sure to have correct side ...
16187 'set killer moves
16188 Killer0 = Killer(Ply)
16189 If Ply > 2 Then
16190     Killer2 = Killer(Ply - 2)
16191 Else
16192     ClearMove Killer2.Killer1: ClearMove Killer2.Killer2: ClearMove Killer2.Killer3
16193 End If
16194
16195 bLegalsOnly = PrevMove.IsChecking And Not bCapturesOnly 'Count legal moves in normal search
(not in QSearch)
16196 If bWhiteToMove Then KingLoc = WKingLoc Else KingLoc = BKingLoc
16197
16198 '--- set pointer to history statistics
16199 CmH = PrevMove.Piece * MAX_BOARD + PrevMove.Target
16200 If Ply > 2 Then Fm1 = CmHPtr(Ply - 2) Else Fm1 = 0
16201 If Ply > 3 Then Fm2 = CmHPtr(Ply - 3) Else Fm2 = 0
16202 If Ply > 4 Then Fm3 = CmHPtr(Ply - 4) Else Fm3 = 0
16203 If Ply > 6 Then Fm5 = CmHPtr(Ply - 6) Else Fm5 = 0
16204 SetMove CounterMoveTmp, CounterMove(PrevMove.Piece, PrevMove.Target)
16205 '-----
16206 '--- Moves loop -
16207 '-----
16208 For i = 0 To NumMoves - 1
16209     With Moves(Ply, i) 'assign move fields for speed reasons
16210         From = .From: Target = .Target: Promoted = .Promoted: Captured = .Captured:
Piece = .Piece: EnPassant = .EnPassant: bIsChecking = .IsChecking
16211         .IsLegal = False: .SeeValue = VALUE_NONE
16212     End With
16213
16214     lValue = 0
16215     '--- Count legal moves if in check
16216     If bLegalsOnly Then
16217         If Moves(Ply, i).Castle = NO_CASTLE Then 'castling not allowed in check
16218             'Avoid costly legal proof for moves with cannot be a check evasion, EnPassant bug fixed here(wrong
mate score if ep Capture is only legal move)
16219             If From <> KingLoc And PieceType(Captured) <> PT_KNIGHT And Not SameXRay(From,
KingLoc) And Not SameXRay(Target, KingLoc) And EpPosArr(Ply) = 0 Then
16220                 'ignore this move because it cannot be a check evasion
16221             Else
16222                 'Do move and test for legal
16223                 RemoveEpPiece
16224                 MakeMove Moves(Ply, i)
16225                 If CheckEvasionLegal() Then Moves(Ply, i).IsLegal = True:
LegalMovesOutOfCheck = LegalMovesOutOfCheck + 1
16226                 'Undo move
16227                 UnmakeMove Moves(Ply, i)
16228                 ResetEpPiece
16229             End If
16230         End If
16231         If Moves(Ply, i).IsLegal Then

```



```

16232         lValue = lValue + 3 * MATE0 ' Out of check moves have top order value
16233     Else
16234         lValue = -999999 ' not a legal evasion
16235         GoTo lblIgnoreMove
16236     End If
16237 End If
16238
16239 PieceVal = PieceAbsValue(Piece)
16240
16241 '--- Is Move checking ?
16242 If Not bIsChecking Then bIsChecking = IsCheckingMove(Piece, From, Target, Promoted
, EnPassant)
16243 If bIsChecking Then
16244     If Not bCapturesOnly Then
16245         If Captured = NO_PIECE Then lValue = lValue + 9000
16246     Else
16247         lValue = lValue + 800 ' in QSearch search captures first??
16248     End If
16249     lValue = lValue + PieceVal \ 6
16250     If Ply > 2 Then
16251         If MovesList(Ply - 2).IsChecking Then lValue = lValue + 500 ' Repeated check
16252     End If
16253     Moves(Ply, i).IsChecking = True
16254 End If
16255 '--- bonus for main line
16256 If bSearchingPV Then
16257     If From = PV(1, Ply).From And Target = PV(1, Ply).Target And Promoted = PV(1,
Ply).Promoted Then
16258         bSearchingPVNew = True: lValue = lValue + 2 * MATE0 ' Highest score
16259         GoTo lblNextMove
16260     End If
16261 End If
16262 '--- bonus for threat move
16263 If ThreatMove.From <> 0 Then
16264     If Target = ThreatMove.From Then
16265         lValue = lValue + 600 ' Try capture, additional bonus later for captures
16266     End If
16267     If From = ThreatMove.Target Then ' Try escape capture
16268         If PieceVal > PieceAbsValue(Board(ThreatMove.From)) + 80 Then
16269             lValue = lValue + 4000 + (PieceVal - PieceAbsValue(Board(ThreatMove.From)))
\ 2
16270         Else
16271             lValue = lValue + 2000 + PieceVal \ 4
16272         End If
16273     Else
16274         ' blocking move?
16275         If (PieceVal - 80 < PieceAbsValue(ThreatMove.Piece)) Then ' blocking makes sense only with less or equal
valuable piece
16276         If IsBlockingMove(ThreatMove, Moves(Ply, i)) Then lValue = lValue + 300 +
PieceAbsValue(ThreatMove.Captured) \ 4
16277         End If
16278     End If
16279 End If
16280 '--- Capture bonus
16281 If Captured <> NO_PIECE Then
16282     '-- Captures
16283     If Not bEndgame Then
16284         If bWhiteToMove Then lValue = lValue - 100 * Rank(Target) Else lValue = lValue
- 100 * (9 - Rank(Target))
16285     End If
16286     If Piece = WKING Or Piece = BKING Then
16287         TmpVal = PieceAbsValue(Captured) ' cannot be defended because legal move
16288     Else
16289         TmpVal = PieceAbsValue(Captured) - PieceVal
16290     End If
16291     v = CaptureHistory(Piece, Target, Captured) \ 150
16292     If TmpVal > MAX_SEE_DIFF Then
16293         '--- Winning capture

```

```

16294         lValue = lValue + TmpVal * 5 + 6000 + v
16295     ElseIf TmpVal > -MAX_SEE_DIFF Then
16296         '--- Equal capture
16297         lValue = lValue + PieceAbsValue(Captured) - PieceVal \ 2 + 800 + v
16298     Else
16299         '--- Loosing capture? Check with SEE later in MovePicker
16300         lValue = lValue + PieceAbsValue(Captured) \ 2 - PieceVal + v
16301     End If
16302     If Target = PrevMove.Target Then lValue = lValue + 250 'Recapture
16303     '-- King attack?
16304     If WhiteMoves Then
16305         If Piece <> WPAWN Then If MaxDistance(Target, BKingLoc) <= 2 And Target <>
            BKingLoc Then lValue = lValue + (PieceVal \ 2 + 400) \ MaxDistance(Target,
            BKingLoc)
16306     Else
16307         If Piece <> BPAWN Then If MaxDistance(Target, WKingLoc) <= 2 And Target <>
            WKingLoc Then lValue = lValue + (PieceVal \ 2 + 400) \ MaxDistance(Target,
            WKingLoc)
16308     End If
16309     Else
16310         '
16311         '--- Not a Capture, subtract 30000 to select captures first
16312         '
16313         If Not bCapturesOnly Then lValue = lValue + MOVE_ORDER_QUIETS 'negative value for
            MOVE_ORDER_QUIETS > set to -30000
16314         'bonus per killer move:
16315         If From = Killer0.Killer1.From Then If Target = Killer0.Killer1.Target Then
            lValue = lValue + 3000: GoTo lblKillerDone
16316         If From = Killer0.Killer2.From Then If Target = Killer0.Killer2.Target Then
            lValue = lValue + 2500: GoTo lblKillerDone
16317         If From = Killer0.Killer3.From Then If Target = Killer0.Killer3.Target Then
            lValue = lValue + 2200: GoTo lblKillerDone
16318
16319         If Ply > 2 Then '--- killer bonus for previous move of same color
16320             If From = Killer2.Killer1.From Then If Target = Killer2.Killer1.Target Then
            lValue = lValue + 2700: GoTo lblKillerDone
16321             If From = Killer2.Killer2.From Then If Target = Killer2.Killer2.Target Then
            lValue = lValue + 200
16322             ' Killer3 not better
16323         End If
16324         If PrevMove.Target <> 0 Then
16325             If CounterMoveTmp.Target = Target Then
16326                 lValue = lValue + 250 ' Bonus for Countermove
16327                 If CounterMoveTmp.Piece = Piece Then lValue = lValue + 250 - PieceVal \ 20
16328             End If
16329         End If
16330     End If
16331
16332     '--- value for piece square table difference of move
16333     lValue = lValue + PieceAbsValue(Promoted) \ 2 + (PsqVal(Abs(bEndgame), Piece,
            Target) - PsqVal(Abs(bEndgame), Piece, From)) * 2
16334
16335     '--- Attacked by pawn or pawn push?
16336     If WhiteMoves Then
16337         If Piece = WPAWN Then
16338             If Rank(Target) >= 6 Then If AdvancedPawnPush(Piece, Target) Then lValue =
            lValue + 250
16339         Else
16340             If Board(Target + 9) = BPAWN Then lValue = lValue - PieceVal \ 4 Else If Board
            (Target + 11) = BPAWN Then lValue = lValue - PieceVal \ 4 '--- Attacked by Pawn
16341             If Board(Target - 9) = WPAWN Then lValue = lValue + 50 + PieceVal \ 8 Else If
            Board(Target - 11) = WPAWN Then lValue = lValue + 50 + PieceVal \ 8 '---
            Defended by Pawn
16342             TmpVal = MaxDistance(Target, BKingLoc): lValue = lValue - TmpVal * TmpVal '
            closer to opp king
16343         End If
16344     Else
16345         If Piece = BPAWN Then

```

```

16346     If Rank(Target) <= 3 Then If AdvancedPawnPush(Piece, Target) Then lValue =
        lValue + 250
16347 Else
16348     If Board(Target - 9) = WPAWN Then lValue = lValue - PieceVal \ 4 Else If Board
        (Target - 11) = WPAWN Then lValue = lValue - PieceVal \ 4 '--- Attacked by Pawn
16349     If Board(Target + 9) = BPAWN Then lValue = lValue + 50 + PieceVal \ 8 Else If
        Board(Target + 11) = BPAWN Then lValue = lValue + 50 + PieceVal \ 8 '---
        Defended by Pawn
16350     TmpVal = MaxDistance(Target, WKingLoc): lValue = lValue - TmpVal * TmpVal '
        closer to opp king
16351 End If
16352 End If
16353 lblKillerDone:
16354 'Check evasions
16355 If PrevMove.IsChecking Then
16356     If Piece = WKING Or Piece = BKING Then lValue = lValue + 200 'King check escape
        move?
16357     If Target = PrevMove.Target Then lValue = lValue + 200 'Capture checking piece?
        'If PrevMove.Target > 0 Then lValue = lValue + History(PieceColor(Piece), From, Target) \ 6
16358 Else 'not in check
16359     'ContinuationHistory
16360     If Captured = NO_PIECE And Promoted = 0 Then
16361         v = 2& * History(PieceColor(Piece), From, Target) '2& data type to avoid overflow
16362     If PrevMove.Target > 0 Then
16363         CurrPtr = Piece * MAX_BOARD + Target
16364         '2& = LONG data type to avoid overflow
16365         v = v + (2& * ContinuationHistory(CmH, CurrPtr) + ContinuationHistory(Fm1,
            CurrPtr) + ContinuationHistory(Fm2, CurrPtr) + ContinuationHistory(Fm3,
            CurrPtr) + ContinuationHistory(Fm5, CurrPtr))
16366         v = v \ 12 'bonus per history heuristic: Caution: big effects! +++order
16367     End If
16368     'If v < TestCnt(1) Then TestCnt(1) = v
16369     'If v > TestCnt(2) Then TestCnt(2) = v
16370     lValue = lValue + v
16371 End If
16372 End If 'PrevMove.IsChecking
16373
16374
16375 lblNextMove:
16376 '--- Hashmove
16377 If BestMove.From = From Then If BestMove.Target = Target Then lValue = lValue +
        MATE0 \ 2: GoTo lblCheckBest
16378 '--- Move from Internal Iterative Deepening
16379 If BestMovePly(Ply).From = From Then If BestMovePly(Ply).Target = Target Then
        lValue = lValue + MATE0 \ 2
16380
16381 lblCheckBest:
16382 If lValue > BestValue Then BestValue = lValue: BestIndex = i 'save best for first move
16383
16384 lblIgnoreMove:
16385 'Set order value for move picker
16386 Moves(Ply, i).OrderValue = lValue
16387 Next '---- Move
16388
16389 bSearchingPV = bSearchingPVNew
16390 'Debug: for i=0 to nummoves-1: Debug.Print i,Moves(ply,i).ordervalue, MoveText(Moves(ply,i)):next
16391
16392 If BestIndex > 0 Then
16393     'Swap best move to top
16394     SwapMove Moves(Ply, 0), Moves(Ply, BestIndex)
16395 End If
16396 End Sub
16397
16398 '-----
16399 'BestMoveAtFirst: get best move from generated move list, scored by OrderMoves.
16400 'Faster than SortMoves if alpha/beta cut in the first moves
16401 '-----
16402
16403 Public Sub BestMoveAtFirst(ByVal Ply As Long, _
        ByVal StartIndex As Long, _
        ByVal NumMoves As Long)
    Dim i As Long, MaxScore As Long, MaxPtr As Long, ActScore As Long

```

```

16404     MaxScore = -9999999
16405     MaxPtr = StartIndex
16406     For i = StartIndex To NumMoves
16407         ActScore = Moves(Ply, i).OrderValue: If ActScore > MaxScore Then MaxScore =
            ActScore: MaxPtr = i
16408     Next i
16409     If MaxPtr > StartIndex Then
16410         SwapMove Moves(Ply, StartIndex), Moves(Ply, MaxPtr)
16411     End If
16412     ' For i = StartIndex To NumMoves '--- check for correct order
16413     ' If Moves(Ply, StartIndex - 1).OrderValue < Moves(Ply, i - 1).OrderValue Then Stop
16414     ' Next
16415 End Sub

16416
16417 ' Stable sort: order of equal values is not changed
16418 Private Sub SortMovesStable(ByVal Ply As Long, ByVal iStart As Long, ByVal iEnd As
Long)
16419     Dim i As Long, j As Long, iMin As Long, IMax As Long
16420     iMin = iStart + 1: IMax = iEnd
16421     i = iMin: j = i + 1
16422
16423     Do While i <= IMax
16424         If Moves(Ply, i).OrderValue > Moves(Ply, i - 1).OrderValue Then
16425             SwapMove Moves(Ply, i), Moves(Ply, i - 1)
16426             If i > iMin Then i = i - 1
16427         Else
16428             i = j: j = j + 1
16429         End If
16430     Loop
16431
16432     ' For i = iStart To iEnd - 1 ' Check sort order
16433     ' If Moves(Ply, i).OrderValue < Moves(Ply, i + 1).OrderValue Then Stop
16434     ' Next
16435 End Sub
16436
16437
16438 '-----
16439 '--- init move picker list -
16440 '-----
16441 Public Function MovePickerInit(ByVal ActPly As Long, _
16442     BestMove As TMOVE, _
16443     PrevMove As TMOVE, _
16444     ThreatMove As TMOVE, _
16445     ByVal bCapturesOnly As Boolean, _
16446     ByVal bMovesGenerated As Boolean, _
16447     ByVal bGenerateQSChecks As Boolean)
16448
16449     With MovePickerDat(ActPly)
16450         .CurrMoveNum = 0
16451         .EndMoves = 0
16452         SetMove .BestMove, BestMove
16453         .bBestMoveChecked = False
16454         .bBestMoveDone = False
16455         SetMove .PrevMove, PrevMove
16456         SetMove .ThreatMove, ThreatMove
16457         .bCapturesOnly = bCapturesOnly
16458         .bMovesGenerated = bMovesGenerated
16459         .LegalMovesOutOfCheck = -1
16460         If bGenerateQSChecks Then .GenerateQSChecksCnt = 1 Else .GenerateQSChecksCnt = 0
16461     End With
16462
16463 End Function
16464
16465 '-----
16466 '- Move picker
16467 '- Returns next move in "Move"
16468 '- or function returns false if no more moves
16469 '-----

```

```

16470 Public Function MovePicker(ByVal ActPly As Long, _
16471                             Move As TMOVE, _
16472                             LegalMovesOutOfCheck As Long) As Boolean
16473 Dim SeeVal As Long, NumMovesPly As Long, BestMove As TMOVE
16474 MovePicker = False: LegalMovesOutOfCheck = 0
16475
16476 With MovePickerDat(ActPly)
16477     ' First: try BestMove. If Cutoff then no move generation needed.
16478     If Not .bBestMoveChecked Then
16479         .bBestMoveChecked = True
16480         If .BestMove.From <> 0 Then
16481             SetMove BestMove, .BestMove
16482             If Not .PrevMove.IsChecking Then ' Check: First generate all out of check moves,
                LegalMovesOutOfCheck needed
16483                 If MovePossible(BestMove) Then
16484                     SetMove Move, BestMove: .bBestMoveDone = True: MovePicker = True:
                        Move.OrderValue = 5 * MATE0
16485
16486                     If bSearchingPV Then
16487                         If Move.From = PV(1, ActPly).From And Move.Target = PV(1, ActPly).Target
                            And Move.Promoted = PV(1, ActPly).Promoted Then
16488                             ' keep SearchingPV
16489                         Else
16490                             bSearchingPV = False
16491                         End If
16492                     End If
16493                     Exit Function '--- return best move before move generation
16494                 End If
16495             End If
16496         End If
16497     End If
16498
16499     If Not .bMovesGenerated Then
16500         ' Generate all moves
16501         GenerateMoves ActPly, .bCapturesOnly, .EndMoves
16502         ' Order moves
16503         OrderMoves ActPly, .EndMoves, .PrevMove, .BestMove, .ThreatMove, .bCapturesOnly,
            .LegalMovesOutOfCheck
16504         .bMovesGenerated = True: .GenerateQSChecksCnt = 0: .CurrMoveNum = 0
16505     End If
16506     LegalMovesOutOfCheck = .LegalMovesOutOfCheck
16507     .CurrMoveNum = .CurrMoveNum + 1 ' array index starts at 0 = nummoves-1
16508     ' ignore Hash move, already done
16509     If .bBestMoveDone And BestMove.From <> 0 Then
16510         If MovesEqual(BestMove, Moves(ActPly, .CurrMoveNum - 1)) Then
16511             .CurrMoveNum = .CurrMoveNum + 1
16512         End If
16513     End If
16514     NumMovesPly = .EndMoves
16515     If NumMovesPly <= 0 Or .CurrMoveNum > NumMovesPly Then ClearMove Move: Exit
        Function
16516     If .CurrMoveNum > 1 Then ' First move is already sorted to top in OrderMoves
16517         ' sort best move to top of remaining list
16518         BestMoveAtFirst ActPly, .CurrMoveNum - 1, NumMovesPly - 1
16519     End If
16520     '---
16521     Do
16522         SetMove Move, Moves(ActPly, .CurrMoveNum - 1)
16523         If Not Move.IsChecking And Move.Captured = NO_PIECE Then MovePicker = True: Exit
            Function ' Quiet move
16524         If Move.OrderValue < MOVE_ORDER_BAD_CAPTURES + 5000 Then MovePicker = True: Exit
            Function ' Bad Capture
16525         If .CurrMoveNum >= NumMovesPly Then MovePicker = True: Exit Function ' Last move
16526         If Move.OrderValue > 1000 Then MovePicker = True: Exit Function ' Good Capture or
            killer
16527         '--- examine capture: good or bad?
16528         If PieceAbsValue(Move.Captured) - PieceAbsValue(Move.Piece) < -MAX_SEE_DIFF Then
16529             '-- Bad capture?

```

```

16530 SeeVal = GetSEE(Move) : Move.SeeValue = SeeVal ' Slow! Delay the costly SEE until this move
is needed - may be not needed if cutoffs earlier
16531 Moves(ActPly, .CurrMoveNum - 1).SeeValue = SeeVal ' Save for later use
16532 If SeeVal >= -MAX_SEE_DIFF Then
16533     MovePicker = True: Exit Function
16534 Else
16535     Move.OrderValue = MOVE_ORDER_BAD_CAPTURES + SeeVal * 5 ' negative See! - Set to fit
condition above < -15000
16536     '- to avoid new list sort: append this bad move to the end of the move list (add new record), skip current
list entry
16537     'Moves(ActPly, .CurrMoveNum - 1).From = 0 ' Delete move in list, not needed ??
16538     NumMovesPly = NumMovesPly + 1: MovePickerDat(ActPly).EndMoves = NumMovesPly:
MovePickerDat(ActPly, NumMovesPly - 1) = Move
16539 End If
16540 Else
16541     MovePicker = True: Exit Function ' good captures
16542 End If
16543 .CurrMoveNum = .CurrMoveNum + 1 ' skip bad capture
16544 Loop
16545
16546 End With
16547
16548 End Function
16549
16550 Public Function CompToMove() As Boolean
16551     If bCompIsWhite Then CompToMove = bWhiteToMove Else CompToMove = Not bWhiteToMove
16552 End Function
16553
16554 Private Function FixedDepthMode() As Boolean
16555     '--- if no time limit use depth limit
16556     FixedDepthMode = CBool(FixedDepth <> NO_FIXED_DEPTH)
16557 End Function
16558
16559 Public Function IsAnyLegalMove(ByVal NumMoves As Long) As Boolean
16560     ' Count legal moves
16561     Dim i As Long
16562     IsAnyLegalMove = False
16563
16564     For i = 0 To NumMoves - 1
16565         RemoveEpPiece
16566         MakeMove Moves(Ply, i)
16567         If CheckLegal(Moves(Ply, i)) Then IsAnyLegalMove = True
16568         UnmakeMove Moves(Ply, i)
16569         ResetEpPiece
16570         If IsAnyLegalMove = True Then Exit Function
16571     Next i
16572
16573 End Function
16574
16575 '-----
16576 '--- Check 3xRepetition Draw in current moves
16577 '-----
16578 Public Function Is3xDraw(Hashkey As THashKey, _
16579     ByVal GameMoves As Long, _
16580     ByVal SearchPly As Long) As Boolean
16581     Dim i As Long, Repeats As Long, EndPos As Long, StartPos As Long, PlyDiff As Long,
Key1 As Long
16582     Is3xDraw = False
16583
16584     If CompToMove Then
16585         PlyDiff = Fifty: If PliesFromNull < Fifty Then PlyDiff = PliesFromNull
16586     Else
16587         PlyDiff = Fifty - 1: If PliesFromNull - 1 < Fifty - 1 Then PlyDiff = PliesFromNull
- 1
16588     End If
16589     If PlyDiff < 4 Then Exit Function
16590     If SearchPly > 1 Then SearchPly = SearchPly - 1
16591     StartPos = GameMoves + SearchPly - 1: If StartPos < 0 Then StartPos = 0

```

```

16592 EndPos = GameMoves + SearchPly - PlyDiff: If EndPos < 0 Then EndPos = 0
16593 If StartPos - EndPos < 2 Then Exit Function
16594
16595 Repeats = 0: Key1 = Hashkey.HashKey1
16596 If Key1 = 0 Then Exit Function
16597 For i = StartPos - 1 To EndPos Step -2
16598     If Key1 = GamePosHash(i).HashKey1 Then
16599         If Hashkey.Hashkey2 = GamePosHash(i).Hashkey2 Then
16600             '2 repeats=3 equal positions. 1 repeated position in search=>Draw; or 1 in game plus 1 in search(except
             root) = 2 => draw
16601             Repeats = Repeats + 1
16602             If Repeats + Abs(i > GameMoves) >= 2 Then
16603                 Is3xDraw = True: Exit Function
16604             End If
16605         End If
16606     End If
16607 Next i
16608 End Function
16609
16610 Public Function CyclingMoves(ByVal ActPly As Long) As Boolean
16611     '--- repeated move ? i.e. "Ra1-a4 <opp move> Ra4-a1
16612     CyclingMoves = False
16613
16614     If ActPly > 3 Then
16615         If Fifty >= 3 And PliesFromNull >= 3 Then
16616             If MovesList(ActPly - 3).From = MovesList(ActPly - 1).Target Then
16617                 If MovesList(ActPly - 3).Target = MovesList(ActPly - 1).From Then
16618                     If MovesList(ActPly - 2).Castle = NO_CASTLE And MovesList(ActPly - 1).Castle
                        = NO_CASTLE Then
16619                         If Not SqBetween(MovesList(ActPly - 1).Target, MovesList(ActPly - 2).From,
                            MovesList(ActPly - 2).Target) Then
16620                             CyclingMoves = True
16621                         End If
16622                     End If
16623                 End If
16624             End If
16625         End If
16626     ElseIf ActPly = 2 Then
16627         If GameMovesCnt > 1 Then
16628             If arGameMoves(GameMovesCnt - 1).From = MovesList(ActPly - 1).Target Then
16629                 If arGameMoves(GameMovesCnt - 1).Target = MovesList(ActPly - 1).From Then
16630                     If arGameMoves(GameMovesCnt).Castle = NO_CASTLE And MovesList(ActPly - 1).
                        Castle = NO_CASTLE Then
16631                         If Not SqBetween(MovesList(ActPly - 1).Target, arGameMoves(GameMovesCnt).
                            From, arGameMoves(GameMovesCnt - 1).Target) Then
16632                             CyclingMoves = True
16633                         End If
16634                     End If
16635                 End If
16636             End If
16637         End If
16638     End If
16639 End Function
16640
16641 Private Function IsKillerMove(ByVal ActPly As Long, Move As TMOVE) As Boolean
16642     ' If Move.From = 0 Then IsKillerMove = False: Exit Function
16643     ' IsKillerMove = True
16644     ' With Killer(ActPly)
16645     '     If Move.From = .Killer1.From Then If Move.Target = .Killer1.Target Then Exit Function
16646     '     If Move.From = .Killer2.From Then If Move.Target = .Killer2.Target Then Exit Function
16647     '     If Move.From = .Killer3.From Then If Move.Target = .Killer3.Target Then Exit Function
16648     ' End With
16649     '
16650     ' IsKillerMove = False
16651 End Function
16652 '
16653 Private Function IsKiller1Move(ByVal ActPly As Long, Move As TMOVE) As Boolean 'first
    killer first?

```



```

16654     If Move.From = 0 Then IsKiller1Move = False: Exit Function
16655     IsKiller1Move = False
16656     With Killer(ActPly).Killer1
16657         If Move.From = .From Then If Move.Target = .Target Then If Move.Piece = .Piece
16658             Then IsKiller1Move = True
16659     End With
16660 End Function

16661 Public Function FutilityMoveCnt(ilImproving As Long, ilDepth As Long) As Long
16662     If ilImproving <> 0 Then FutilityMoveCnt = (3 + ilDepth * ilDepth) Else
16663         FutilityMoveCnt = (3 + ilDepth * ilDepth) \ 2
16664 End Function

16665
16666 Public Function FutilityMargin(ByVal iDepth As Long, ByVal Improving As Long) As Long
16667     FutilityMargin = (154 & * (iDepth - Improving))
16668 End Function
16669
16670
16671 Public Sub InitReductionArray()
16672     ' Init reductions array
16673     Dim mc As Long
16674     Debug.Assert NoOfThreads > 0
16675
16676     For mc = 1 To 63
16677         Reductions(mc) = CLng(19.47 + Log(CDb1(NoOfThreads)) \ 2) * Log(CDb1(mc))
16678         'Debug.Print mc, Reductions(mc)
16679     Next mc
16680
16681 End Sub

16682
16683 '-----
16684 '- Returns depth reduction
16685 '-----
16686 Public Function Reduction(ByVal Improving As Long, _
16687     ByVal Depth As Long, _
16688     ByVal MoveNumber As Long, ByVal Delta As Long, ByVal
16689     RootDelta As Long) As Long
16690
16691     Dim r As Long
16692     If MoveNumber > 63 Then MoveNumber = 63
16693     r = Reductions(Depth) * Reductions(MoveNumber)
16694     Reduction = (r + 1372 - ((Delta * 1037) \ RootDelta)) \ 1024
16695     Debug.Assert Reduction >= 0
16696     If Improving = 0 Then If r > 936 Then Reduction = Reduction + 1
16697 End Function
16698
16699 '-----
16700 '- Updates statistics
16701 '-----
16702 Private Function UpdateStats(ByVal ActPly As Long, _
16703     BestMove As TMOVE, _
16704     ByVal BestScore As Long, _
16705     ByVal Beta As Long, _
16706     ByVal QuietMovesSearched As Long, _
16707     ByVal CaptureMovesSearched As Long, _
16708     PrevMove As TMOVE, _
16709     ByVal Depth As Long)
16710
16711     '
16712     '--- Update Killer moves and History-Score
16713     '
16714     Dim j As Long, Bonus1 As Long
16715     Debug.Assert BestMove.Piece > FRAME And BestMove.Piece < NO_PIECE
16716
16717     Bonus1 = StatBonus(Depth + 1)
16718
16719     ' if NOT a capture
16720     If BestMove.From >= SQ_A1 And BestMove.Captured = NO_PIECE Then
16721         Dim Bonus2 As Long

```

```

16719 'If BestScore > Beta + 145 Then
16720 'If BestScore > Beta + 150 And BestScore > 0 Then ###beta###
16721 '--- not clear why * 150 instead of + 150 works much better here
16722 If BestScore > Beta * 150 Then Bonus2 = Bonus1 Else Bonus2 = StatBonus(Depth)
16723
16724 ' Increase stats for the best move in case it was a quiet move
16725 UpdQuietStats ActPly, BestMove, PrevMove, Bonus2
16726
16727 '--- Decrease History for previous tried quiet moves that did not cut off
16728 For j = 1 To QuietMovesSearched
16729     With QuietsSearched(ActPly, j)
16730         If .From = BestMove.From And .Target = BestMove.Target And .Piece =
16731             BestMove.Piece Then
16732             ' ignore
16733         Else
16734             UpdHistory .Piece, .From, .Target, -Bonus2
16735             If PrevMove.Target > 0 Then UpdateContHistStats ActPly, .Piece, .Target, -
16736                 Bonus2
16737         End If
16738     End With
16739 Next j
16740
16741 Else ' a Capture
16742     ' Increase stats for the best move in case it was a capture move
16743     UpdCaptureHistory BestMove.Piece, BestMove.Target, BestMove.Captured, Bonus1
16744 End If
16745
16746 ' << Extra penalty for a quiet TT move in previous ply when it gets refuted > in Search Code
16747
16748 ' Decrease stats for all non-best capture moves
16749 For j = 1 To CaptureMovesSearched
16750     With CapturesSearched(ActPly, j)
16751         If .From = BestMove.From And .Target = BestMove.Target And .Piece =
16752             BestMove.Piece Then
16753             ' ignore
16754         Else
16755             UpdCaptureHistory .Piece, .Target, .Captured, -Bonus1
16756         End If
16757     End With
16758 Next j
16759
16760 End Function
16761
16762 Public Sub UpdQuietStats(ByVal ActPly As Long, _
16763     CurrentMove As TMOVE, _
16764     PrevMove As TMOVE, _
16765     ByVal Bonus As Long)
16766     '--- update killer moves
16767     With Killer(ActPly)
16768         If CurrentMove.Target <> PrevMove.From Then ' not if opp moved attacked piece away > not a killer
16769             for other moves
16770             SetMove .Killer3, .Killer2: SetMove .Killer2, .Killer1: SetMove .Killer1,
16771                 CurrentMove
16772         End If
16773     End With
16774
16775 UpdHistory CurrentMove.Piece, CurrentMove.From, CurrentMove.Target, Bonus
16776 UpdateContHistStats ActPly, CurrentMove.Piece, CurrentMove.Target, Bonus
16777
16778 If PrevMove.From >= SQ_A1 And PrevMove.Captured = NO_PIECE Then
16779     '--- CounterMove:
16780     SetMove CounterMove(PrevMove.Piece, PrevMove.Target), CurrentMove
16781 End If
16782
16783 End Sub
16784
16785 Public Sub UpdHistory(ByVal Piece As Long, _
16786     ByVal From As Long, _

```

```

16782         ByVal Target As Long, _
16783         ByVal ScoreVal As Long)
16784 'range +/- 10692
16785 Debug.Assert Piece > FRAME And Piece < NO_PIECE
16786 History(PieceColor(Piece), From, Target) = History(PieceColor(Piece), From, Target)
+ ScoreVal - (History(PieceColor(Piece), From, Target) * Abs(ScoreVal) \ 7183)
16787 'Debug.Assert Abs(History(PieceColor(Piece), From, Target)) <= 7183
16788 End Sub
16789
16790 Public Sub UpdCaptureHistory(ByVal Piece As Long, _
16791         ByVal Target As Long, _
16792         ByVal CapturedPiece As Long, _
16793         ByVal ScoreVal As Long)
16794 Debug.Assert Piece > FRAME And Piece < NO_PIECE
16795 CaptureHistory(Piece, Target, CapturedPiece) = CaptureHistory(Piece, Target,
CapturedPiece) + ScoreVal - (CaptureHistory(Piece, Target, CapturedPiece) * Abs(
ScoreVal) \ 10692)
16796 'Debug.Assert Abs(CaptureHistory(Piece, Target, CapturedPiece)) <= 10692
16797 End Sub
16798
16799 Public Sub UpdateContHistStats(ByVal ActPly As Long, _
16800         ByVal Piece As Long, _
16801         ByVal Square As Long, _
16802         ByVal Bonus As Long)
16803 Debug.Assert Piece > FRAME And Piece < NO_PIECE
16804 If ActPly > 1 Then
16805     If MovesList(ActPly - 1).From > 0 Then
16806         ContHistVal MovesList(ActPly - 1).Piece, MovesList(ActPly - 1).Target, Piece,
Square, Bonus
16807     End If
16808     If ActPly > 2 Then
16809         If MovesList(ActPly - 2).From > 0 Then
16810             ContHistVal MovesList(ActPly - 2).Piece, MovesList(ActPly - 2).Target, Piece,
Square, Bonus
16811         End If
16812     ' If ActPly > 3 Then
16813     '     If MovesList(ActPly - 3).From > 0 Then
16814     '         ContHistVal MovesList(ActPly - 3).Piece, MovesList(ActPly - 3).Target, Piece, Square, Bonus \ 4
16815     '     End If
16816     '     If ActPly > 4 And Not MovesList(ActPly - 1).IsChecking Then 'no more when in check
16817     '         If MovesList(ActPly - 4).From > 0 Then
16818     '             ContHistVal MovesList(ActPly - 4).Piece, MovesList(ActPly - 4).Target,
Piece, Square, Bonus
16819     '         End If
16820     '         If ActPly > 6 Then
16821     '             If MovesList(ActPly - 6).From > 0 Then
16822     '                 ContHistVal MovesList(ActPly - 6).Piece, MovesList(ActPly - 6).Target,
Piece, Square, Bonus
16823     '             End If
16824     '         End If '6
16825     '     End If '4
16826     ' End If '3
16827     ' End If '2
16828     ' End If '1
16829 End Sub
16830
16831 Public Sub ContHistVal(ByVal PrevPiece As Long, _
16832         ByVal PrevSquare As Long, _
16833         ByVal Piece As Long, _
16834         ByVal Square As Long, _
16835         ByVal ScoreVal As Long)
16836 ' Range +/-29952
16837 Debug.Assert Piece > FRAME And Piece < NO_PIECE
16838 Dim PrevPtr As Long, CurrPtr As Long
16839 PrevPtr = PrevPiece * MAX_BOARD + PrevSquare: CurrPtr = Piece * MAX_BOARD + Square
16840 ContinuationHistory(PrevPtr, CurrPtr) = ContinuationHistory(PrevPtr, CurrPtr) +
ScoreVal - (ContinuationHistory(PrevPtr, CurrPtr) * Abs(ScoreVal) \ 29952)
16841 'Debug.Assert Abs(ContinuationHistory(PrevPtr, CurrPtr)) <= 29952

```

```

16842 End Sub
16843
16844 '-----
16845 '- update moves for current line
16846 '-----
16847 Public Sub UpdatePV(ByVal ActPly As Long, Move As TMOVE)
16848     Dim j As Long
16849     SetMove PV(ActPly, ActPly), Move
16850     If PVLength(ActPly + 1) > 0 Then
16851
16852         For j = ActPly + 1 To PVLength(ActPly + 1) - 1
16853             SetMove PV(ActPly, j), PV(ActPly + 1, j)
16854         Next
16855
16856         PVLength(ActPly) = PVLength(ActPly + 1)
16857     End If
16858 End Sub
16859
16860 Public Function MovePossible(Move As TMOVE) As Boolean
16861     ' for test of HashMove before move generation if this move is possible. This may avoid move generation
16862     Dim Offset As Long, sq As Long, Diff As Long, AbsDiff As Long, OldPiece As Long
16863     MovePossible = False
16864     OldPiece = Move.Piece: If Move.Promoted > 0 Then OldPiece = Board(Move.From)
16865     If Move.From < SQ_A1 Or Move.From > SQ_H8 Or OldPiece < 1 Or Move.From = Move.Target
16866         Or OldPiece = NO_PIECE Then Exit Function
16867     If Board(Move.Target) = FRAME Then Exit Function
16868     If Board(Move.From) <> OldPiece Then Exit Function
16869     If Move.Captured < NO_PIECE Then If Board(Move.Target) <> Move.Captured Then Exit
16870     Function
16871     If bWhiteToMove Then
16872         If (OldPiece And 1) <> 1 Then Exit Function
16873     Else
16874         If (OldPiece And 1) <> 0 Then Exit Function
16875     End If
16876     If Board(Move.Target) <> NO_PIECE Then
16877         If (Board(Move.Target) And 1) = (OldPiece And 1) Then Exit Function ' same color
16878     End If
16879     Diff = Move.Target - Move.From: AbsDiff = Abs(Diff)
16880     If PieceType(OldPiece) = PT_PAWN Then
16881         If (AbsDiff = 9 Or AbsDiff = 11) And Board(Move.Target) = NO_PIECE Then Exit
16882         Function
16883         If AbsDiff = 10 And Board(Move.Target) <> NO_PIECE Then Exit Function
16884         If AbsDiff = 20 Then
16885             If Board(Move.From + 10 * Sgn(Diff)) <> NO_PIECE Then Exit Function
16886             If Board(Move.Target) <> NO_PIECE Then Exit Function
16887         End If
16888         MovePossible = True
16889         Exit Function
16890     ElseIf OldPiece = WKNIGHT Or OldPiece = BKNIGHT Then
16891
16892         ' Knight
16893         Select Case AbsDiff
16894             Case 8, 12, 19, 21
16895                 MovePossible = True ' OK
16896         End Select
16897
16898         Exit Function
16899     ElseIf OldPiece = WKING Then
16900         ' WKing: Castling
16901         If AbsDiff = 2 Then
16902             If Move.From <> WKING_START Or Moved(WKING_START) > 0 Then Exit Function
16903             If Diff = 2 Then
16904                 If Board(Move.From + 1) <> NO_PIECE Or Board(Move.From + 2) <> NO_PIECE Or
16905                     Board(Move.From + 3) <> WROOK Then Exit Function
16906             ElseIf Diff = -2 Then
16907                 If Board(Move.From - 1) <> NO_PIECE Or Board(Move.From - 2) <> NO_PIECE Or
16908                     Board(Move.From - 3) <> NO_PIECE Or Board(Move.From - 4) <> WROOK Then Exit
16909                 Function

```

```

16904         End If
16905     End If
16906     MovePossible = True
16907     Exit Function
16908 ElseIf OldPiece = BKING Then
16909     'BKing: Castling
16910     If AbsDiff = 2 Then
16911         If Move.From <> BKING_START Or Moved(BKING_START) > 0 Then Exit Function
16912         If Diff = 2 Then
16913             If Board(Move.From + 1) <> NO_PIECE Or Board(Move.From + 2) <> NO_PIECE Or
16914                 Board(Move.From + 3) <> BROOK Then Exit Function
16915             ElseIf Diff = -2 Then
16916                 If Board(Move.From - 1) <> NO_PIECE Or Board(Move.From - 2) <> NO_PIECE Or
16917                     Board(Move.From - 3) <> NO_PIECE Or Board(Move.From - 4) <> BROOK Then Exit
16918                     Function
16919             End If
16920         End If
16921     End If
16922     MovePossible = True
16923     Exit Function
16924 End If
16925 '--- Sliding piece blocked?
16926 If MaxDistance(Move.From, Move.Target) > 1 Then
16927     If AbsDiff Mod 9 = 0 Then
16928         Offset = Sgn(Diff) * 9
16929     ElseIf AbsDiff Mod 11 = 0 Then
16930         Offset = Sgn(Diff) * 11
16931     ElseIf AbsDiff Mod 10 = 0 Then
16932         Offset = Sgn(Diff) * 10
16933     Else
16934         Offset = Sgn(Diff) * 1
16935     End If
16936
16937     Select Case OldPiece
16938     Case WROOK, BROOK:
16939         If Abs(Offset) <> 1 And Abs(Offset) <> 10 Then Exit Function
16940     Case WBISHOP, BBISHOP:
16941         If Abs(Offset) <> 9 And Abs(Offset) <> 11 Then Exit Function
16942     Case WQUEEN, BQUEEN:
16943         If Abs(Offset) <> 1 And Abs(Offset) <> 10 And Abs(Offset) <> 9 And Abs(Offset)
16944             <> 11 Then Exit Function
16945     End Select
16946
16947     For sq = Move.From + Offset To Move.Target - Offset Step Offset
16948         If Board(sq) < NO_PIECE Then Exit Function
16949     Next
16950
16951 End If
16952 MovePossible = True
16953 End Function
16954
16955 Public Function PawnOnRank7() As Boolean
16956     'check if side to move has a pawn on relative rank 7
16957     Dim i As Long
16958     If bWhiteToMove Then
16959         For i = SQ_A7 To SQ_H7
16960             If Board(i) = WPAWN Then PawnOnRank7 = True: Exit Function
16961         Next
16962     Else
16963         For i = SQ_A2 To SQ_H2
16964             If Board(i) = BPAWN Then PawnOnRank7 = True: Exit Function
16965         Next
16966     End If
16967     PawnOnRank7 = False
16968 End Function
16969
16970 Public Sub ClearEasyMove()
16971     EasyMovePV(1) = EmptyMove: EasyMovePV(2) = EmptyMove: EasyMovePV(3) = EmptyMove
16972     EasyMoveStableCnt = 0

```

```

16968 End Sub
16969
16970 Public Sub UpdateEasyMove()
16971     Dim i As Long, bDoUpdate As Boolean
16972     If MovesEqual(PV(1, 3), EasyMovePV(3)) Then
16973         EasyMoveStableCnt = EasyMoveStableCnt + 1
16974     Else
16975         EasyMoveStableCnt = 0
16976     End If
16977     bDoUpdate = False
16978
16979     For i = 1 To 3
16980         If PV(1, i).From > 0 Then If Not MovesEqual(EasyMovePV(i), PV(1, i)) Then
16981             bDoUpdate = True
16982         Next
16983
16984     If bDoUpdate Then
16985         For i = 1 To 3: EasyMovePV(i) = PV(1, i): Next
16986         'If bTimeTrace Then WriteTrace "UpdateEasyMove: " & MoveText(PV(1, 1)) & " " & MoveText(PV(1, 2)) & " " &
16987         MoveText(PV(1, 3))
16988     End If
16989 End Sub
16990
16991 Public Function GetEasyMove() As TMOVE
16992     ' Return Easy move if previous moves are as expected
16993     SetMove GetEasyMove, EmptyMove
16994     If GameMovesCnt >= 2 And EasyMovePV(3).From > 0 Then
16995         If bTimeTrace Then WriteTrace "GetEasyMove: EM3" & MoveText(EasyMovePV(3)) & " (
16996         EM1:" & MoveText(EasyMovePV(1)) & " = GM1:" & MoveText(arGameMoves(GameMovesCnt -
16997         1)) & " / EM2:" & MoveText(EasyMovePV(1)) & " = GM2:" & MoveText(arGameMoves(
16998         GameMovesCnt))
16999         If MovesEqual(EasyMovePV(1), arGameMoves(GameMovesCnt - 1)) And MovesEqual(
17000         EasyMovePV(2), arGameMoves(GameMovesCnt)) Then
17001             SetMove GetEasyMove, EasyMovePV(3)
17002         End If
17003     End If
17004 End Function
17005
17006 Public Sub InitAttackBitCnt()
17007     ' fill array with attacking pieces count for attack bits set
17008     Dim i As Long, Cnt As Long
17009
17010     For i = 1 To QXrayAttackBit * 2
17011         Cnt = 0
17012         If i And PLAttackBit Then Cnt = Cnt + 1
17013         If i And PRAttackBit Then Cnt = Cnt + 1
17014         If i And N1AttackBit Then Cnt = Cnt + 1
17015         If i And N2AttackBit Then Cnt = Cnt + 1
17016         If i And B1AttackBit Then Cnt = Cnt + 1
17017         If i And B2AttackBit Then Cnt = Cnt + 1
17018         If i And (R1AttackBit Or R1XrayAttackBit) Then Cnt = Cnt + 1
17019         If i And (R2AttackBit Or R2XrayAttackBit) Then Cnt = Cnt + 1
17020         If i And QAttackBit Then Cnt = Cnt + 1
17021         If i And KAttackBit Then Cnt = Cnt + 1
17022         If i And BXrayAttackBit Then Cnt = Cnt + 1 ' for multiple bishops
17023         If i And QXrayAttackBit Then Cnt = Cnt + 1 ' for multiple queens
17024         AttackBitCnt(i) = Cnt
17025     Next
17026 End Sub
17027
17028 Public Function StatBonus(ByVal Depth As Long) As Long
17029     ' StatBonus = Depth * Depth + 2 * Depth - 2
17030     StatBonus = 340 * Depth - 470: If StatBonus > 1710 Then StatBonus = 1710
17031 End Function
17032
17033 Public Function GetHashMove(Hashkey As THashKey) As TMOVE

```

```

17030 'get best move for hint at root
17031 Dim ttHit As Boolean, HashEvalType As Long, HashScore As Long, HashStaticEval As
Long, HashDepth As Long, HashMove As TMOVE, HashPvHit As Boolean, HashThreadNum As
Long
17032 ClearMove GetHashMove
17033 ttHit = HashTableRead(Hashkey, HashDepth, HashMove, HashEvalType, HashScore,
HashStaticEval, HashPvHit, HashThreadNum)
17034 If ttHit Then
17035     If HashMove.From <> 0 Then SetMove GetHashMove, HashMove
17036 End If
17037 End Function
17038
17039 Public Function MoveInMoveList(ByVal ActPly As Long, _
17040                               ByVal StartIndex As Long, _
17041                               ByVal EndIndex As Long, _
17042                               CheckMove As TMOVE) As Boolean
17043 ' Check if the move is in the generate move list, and copies missing attribute ( IsChecking,...)
17044 Dim i As Long, tmp As TMOVE
17045 MoveInMoveList = False
17046 If CheckMove.From = 0 Then Exit Function
17047
17048 For i = StartIndex To EndIndex
17049     'Debug.Print MoveText(Moves(ActPly, i))
17050     tmp = Moves(ActPly, i)
17051     If CheckMove.From <> tmp.From Then GoTo lblNext
17052     If CheckMove.Target <> tmp.Target Then GoTo lblNext
17053     If CheckMove.Promoted <> tmp.Promoted Then GoTo lblNext
17054     If CheckMove.Captured <> tmp.Captured Then GoTo lblNext
17055     ' Found
17056     SetMove CheckMove, tmp ' return all attributes of the move
17057     MoveInMoveList = True
17058     Exit Function
17059 lblNext:
17060 Next
17061
17062 End Function
17063
17064 Public Function DrawValueForSide(bSideToMoveIsWhite As Boolean) As Long
17065 If bCompIsWhite Then
17066     If bSideToMoveIsWhite Then DrawValueForSide = DrawContempt Else DrawValueForSide =
-DrawContempt
17067 Else
17068     If Not bSideToMoveIsWhite Then DrawValueForSide = DrawContempt Else
DrawValueForSide = -DrawContempt
17069 End If
17070 End Function
17071
17072
17073
17074
17075
17076
17077
17078
17079
17080 Attribute VB_Name = "basTime"
17081 Option Explicit
17082 '=====
17083 '= basTime:      =
17084 '= Time management  =
17085 '=====
17086 Public bTimeExit As Boolean
17087 Public TimeStart As Single
17088 Public SearchStart As Single
17089 Public SearchTime As Single
17090 Public ExtraTimeForMove As Single
17091 Public TimeLeft As Single
17092 Public OpponentTime As Single

```



```

17093 Public TimeIncrement As Long
17094 Public LevelMovesToTC As Long
17095 Public MovesToTC As Long
17096 Public SecondsPerGame As Long
17097 Public FixedDepth As Long '=NO_FIXED_DEPTH if time limit is used
17098 Public FixedTime As Single
17099 Public LastChangeDepth As Long, LastChangeMove As String
17100 Public bResearching As Boolean '--- out of aspiration windows: more time
17101 Public BestMoveChanges As Single ' More time if best move changes often
17102 Public MaximumTime As Single
17103 Public OptimalTime As Single
17104 Public MoveOverhead As Single
17105 Public MoreTimeForFirstMove As Boolean ' fill Hash table
17106
17107
17108 '-----
17109 '- AllocateTime()
17110 '-----
17111
17112 Public Sub AllocateTime()
17113 Dim GameMovesDone As Long
17114
17115 If bTimeTrace Then
17116 WriteTrace " -----"
17117 WriteTrace ">> Start AllocateTime MTOC:" & MovesToTC & ", MoveCnt=" & CStr(
GameMovesCnt) & ", Left:" & Format$(TimeLeft, "0.00")
17118 End If
17119
17120 GameMovesDone = (GameMovesCnt + 1) \ 2 ' Full move = 2* Half move
17121
17122 If Not UCIMode And LevelMovesToTC > 0 Then
17123 MovesToTC = LevelMovesToTC - (GameMovesDone Mod LevelMovesToTC)
17124 If bTimeTrace Then
17125 WriteTrace "CalcTime WB: LevelMovesToTC=" & LevelMovesToTC & ", MovesToTC=" &
MovesToTC & ", MovesDone:" & GameMovesDone
17126 End If
17127 End If
17128 '
17129 OptimalTime = CalcTime(MovesToTC, TimeIncrement, TimeLeft, MoveOverhead + 0.05 * (
NoOfThreads - 1), True)
17130 MaximumTime = CalcTime(MovesToTC, TimeIncrement, TimeLeft, MoveOverhead + 0.05 * (
NoOfThreads - 1), False)
17131 '
17132 MaximumTime = GetMinSingle(MaximumTime, TimeLeft / 2#)
17133 OptimalTime = GetMinSingle(MaximumTime, OptimalTime)
17134
17135 If OptimalTime < 0.2 Then
17136 OptimalTime = GetMinSingle(0.2 + 0.05 * NoOfThreads, 1#): MaximumTime =
OptimalTime
17137 End If
17138 MoreTimeForFirstMove = False
17139 If bTimeTrace Then
17140 WriteTrace ">>>> Time allocated Opt: " & Format$(OptimalTime, "0.00") & " / Max:" &
Format$(MaximumTime, "0.00") & " MTOC:" & MovesToTC & " MoveCnt=" & CStr(
GameMovesCnt) & ", Left:" & Format$(TimeLeft, "0.00")
17141 End If
17142 End Sub
17143
17144 '
17145 '--- Calculate time for move
17146 '
17147 Public Function CalcTime(ByVal MovesToTC As Long, _
17148 ByVal TimeIncr As Single, _
17149 ByVal MyTime As Single, _
17150 ByVal MoveOverhead As Single, _
17151 ByVal TimeTypeIsOptimum As Boolean) As Single
17152 Dim Ratio As Single, Inc As Single, k As Single, SafetyMargin As Single
17153 Dim GameMovesDone As Long

```

```

17154
17155 GameMovesDone = (GameMovesCnt + 1) \ 2 ' Full move = 2* Half move
17156
17157 If MyTime <= 0 Then CalcTime = 0: Exit Function
17158
17159 Inc = TimeIncr * GetMaxSingle(60#, 125# - 0.1 * CSng((GameMovesDone - 23) *
    (GameMovesDone - 23)))
17160 SafetyMargin = 1.5
17161
17162 If MovesToTC > 0 Then
17163     If TimeTypeIsOptimum Then
17164         Ratio = 1#
17165     Else
17166         If MovesToTC <= 10 Then Ratio = 2.5 Else Ratio = 4.5
17167     End If
17168     Ratio = Ratio / CSng(GetMin(45, GetMax(1, MovesToTC)))
17169
17170     If GameMovesDone <= 40 Then
17171         Ratio = Ratio * (1.3 - 0.001 * CSng((GameMovesDone - 23) * (GameMovesDone - 23)))
17172     Else
17173         Ratio = Ratio * 1.45
17174     End If
17175     Ratio = Ratio * (1# + Inc / (MyTime * 8.2))
17176     If MovesToTC <= 3 Then SafetyMargin = 3#
17177 Else
17178     k = 1# + 21# * CSng(GameMovesDone) / CSng(500 + GameMovesDone)
17179     If TimeTypeIsOptimum Then Ratio = 0.021 Else Ratio = 0.075
17180     Ratio = Ratio * (k + Inc / MyTime)
17181 End If
17182 If MoreTimeForFirstMove Then Ratio = Ratio * 1.5
17183 '
17184 CalcTime = GetMinSingle(1#, Ratio) * GetMaxSingle(0.01, MyTime - MoveOverhead -
    SafetyMargin - TimeIncr / 10#)
17185 End Function
17186
17187
17188 Public Function TimerDiff(ByVal StartTime As Single, ByVal EndTime As Single) As
    Single
17189     If StartTime - 0.1 > EndTime Then ' Timer resets to 0 ad midnight > EndTime > Startime
17190         EndTime = EndTime + CSng(60& * 60& * 24&)
17191     End If
17192     TimerDiff = EndTime - StartTime
17193     If TimerDiff < 0 Then TimerDiff = 0.1
17194 End Function
17195
17196 Public Function TimeElapsed() As Single
17197     TimeElapsed = TimerDiff(TimeStart, Timer())
17198 End Function
17199 '
17200 '--- Check for time exceeded
17201 '
17202 Public Function CheckTime() As Boolean
17203     Dim Elapsed As Single, Improve As Single, Optimum2 As Single, NewScore As Long,
        PrevScore As Long
17204     CheckTime = True
17205
17206     Elapsed = TimeElapsed()
17207     If FinalScore = VALUE_NONE Then NewScore = 0 Else NewScore = FinalScore
17208     If PrevGameMoveScore = VALUE_NONE Then PrevScore = FinalScore - 80 Else PrevScore =
        PrevGameMoveScore
17209
17210     Improve = GetMaxSingle(229#, GetMinSingle(715#, 357# + 119# * Abs(bFailedLowAtRoot)
        - 5# * CSng(NewScore - PrevScore)))
17211     ' if score jumps up, extra time to make sure it holds
17212     If Abs(NewScore) < ScorePawn.MG * 5 Then
17213         If (NewScore - PrevScore) > ScorePawn.MG * 2 \ 3 Then
17214             Improve = Improve * 2
17215             If (NewScore - PrevScore) > ScorePawn.MG * 2 Then Improve = Improve * 2

```

```

17216         End If
17217     End If
17218     Optimum2 = (OptimalTime * (1# + BestMoveChanges) * Improve) / 640#
17219
17220     If Elapsed >= GetMinSingle(MaximumTime, Optimum2) Then
17221         CheckTime = False
17222         If bTimeTrace Then
17223             WriteTrace "CheckTime D" & RootDepth & ": Elapsed:" & Format$(Elapsed, "0.00")
17224                 & ", Opt2:" & Format$(Optimum2, "0.00") & ", Opt:" & Format$(OptimalTime,
17225                 "0.00") & ", Max:" & Format$(MaximumTime, "0.00")
17226         End If
17227     End If
17228 End Function
17229 Attribute VB_Name = "UtilVBAbas"
17230 '=====
17231 '= UtilVBAbas:
17232 '= functions for VBA GUI ( VBA= Visual Basic for Application in MS-Office)
17233 '=====
17234 Option Explicit
17235 Public Const TEST_MODE As Boolean = True
17236 Public ThisApp As Object ' Office object: Excel, Word,...
17237 Public psGameFile As String
17238 Public LastInfoNodes As Long
17239
17240 Public psLastFieldClick As String
17241 Public psLastFieldMouseDown As String
17242 Public psLastFieldMouseUp As String
17243
17244 Public SetupBoardMode As Boolean ' manual board setup using GUI
17245 Public SetupPiece As Long
17246
17247 ' GUI colors
17248 Public WhiteSqCol As Long
17249 Public BlackSqCol As Long
17250 Public BoardFrameCol As Long
17251
17252 Public plFieldFrom As Long, plFieldTarget As Long
17253 Public psFieldFrom As String, psFieldTarget As String
17254 Dim plFieldFromColor As Long, plFieldTargetColor As Long
17255 Dim psMove As String
17256
17257 Sub run_ChessBrainX()
17258     Main
17259 End Sub
17260
17261 Public Sub SetVBAPathes()
17262     pbIsOfficeMode = True
17263     Set ThisApp = Application
17264     Select Case ThisApp.Name
17265         Case "Microsoft Excel"
17266             psDocumentPath = ThisApp.ActiveWorkbook.Path
17267
17268         Case "Microsoft Word"
17269             psDocumentPath = ThisApp.ActiveDocument.Path
17270
17271         ' Case "Microsoft Powerpoint"
17272         ' psDocumentPath = ActivePresentation.Path
17273
17274         Case Else
17275             psDocumentPath = ThisApp.ActiveWorkbook.Path
17276     End Select
17277     psAppName = "ChessBrainX"
17278     psEnginePath = psDocumentPath
17279 End Sub
17280
17281 Public Sub DoFieldClicked()
17282     ' square click handling: 1. click: select FROM square, 2. click: select TARGET square => do move

```

```

17282 Dim bIsLegal As Boolean, NumLegalMoves As Long, FieldPos As Long, FieldTarget As
17283 Long
17284 Dim sPromotePiece As String, lResult As Long
17285 '--- Setup board mode: if square not empty: 1 click: white piece, 2. click: black piece, 3. click: clear field
17286 If SetupBoardMode Then
17287     If Trim(psLastFieldClick) <> "" Then
17288         If SetupPiece > 0 Then
17289             psFieldFrom = psLastFieldClick
17290             plFieldFrom = Val("0" & Mid(psLastFieldClick, Len("Square") + 1))
17291             FieldPos = FieldNumToBoardPos(plFieldFrom)
17292
17293             If Board(FieldPos) = NO_PIECE Or (PieceType(Board(FieldPos)) <> PieceType(
17294                 SetupPiece)) Then
17295                 Board(FieldPos) = SetupPiece
17296             ElseIf PieceColor(Board(FieldPos)) = COL_WHITE Then
17297                 If PieceColor(SetupPiece) = COL_WHITE Then
17298                     Board(FieldPos) = SetupPiece + 1 ' Black piece, same type
17299                 Else
17300                     Board(FieldPos) = NO_PIECE
17301                 End If
17302             ElseIf PieceColor(Board(FieldPos)) = COL_BLACK Then
17303                 If PieceColor(SetupPiece) = COL_BLACK Then
17304                     Board(FieldPos) = SetupPiece - 1 ' white piece, same type
17305                 Else
17306                     Board(FieldPos) = NO_PIECE
17307                 End If
17308             Else
17309                 ' Clear
17310                 Board(FieldPos) = NO_PIECE
17311             End If
17312             frmChessX.ShowBoard
17313             DoEvents
17314         End If
17315     End If
17316 Exit Sub
17317 End If
17318 ' Move input
17319 If Trim(psLastFieldClick) <> "" Then
17320     If plFieldFrom = 0 Then
17321
17322         '--- First click: Field from
17323         psFieldFrom = psLastFieldClick
17324         plFieldFrom = Val("0" & Mid(psLastFieldClick, Len("Square") + 1))
17325         FieldPos = FieldNumToBoardPos(plFieldFrom)
17326         If Board(FieldPos) < NO_PIECE Then
17327             '-- check color to move
17328             If bWhiteToMove And Board(FieldPos) Mod 2 <> 1 Or _
17329                 Not bWhiteToMove And Board(FieldPos) Mod 2 <> 0 Then
17330                 '--- wrong color
17331                 SendCommand "Wrong color! "
17332                 plFieldFrom = 0
17333                 ResetGUIFieldColors
17334             Else
17335                 frmChessX.Controls(psLastFieldClick).BackColor = &HFF8080
17336                 ShowLegalMovesForPiece FieldNumToCoord(plFieldFrom)
17337             End If
17338         Else
17339             ' ignore empty field
17340             plFieldFrom = 0
17341             ResetGUIFieldColors
17342         End If
17343     Else
17344
17345         '--- Second click: Field target
17346         If psLastFieldClick = psFieldFrom Then

```

```

17348         ResetGUIFieldColors
17349         DoEvents
17350         plFieldFrom = 0
17351     Else
17352         psFieldTarget = psLastFieldClick
17353         plFieldTarget = Val("0" & Mid(psLastFieldClick, Len("Square") + 1))
17354         frmChessX.Controls(psLastFieldClick).BackColor = &HC0FFC0
17355         DoEvents
17356         Sleep 250
17357         '--- Check player move
17358         bIsLegal = CheckGUIMoveIsLegal(FieldNumToCoord(plFieldFrom), FieldNumToCoord(
plFieldTarget), NumLegalMoves)
17359         If bIsLegal Then
17360             'Promotion?
17361             sPromotePiece = "": FieldPos = FieldNumToBoardPos(plFieldFrom): FieldTarget
= FieldNumToBoardPos(plFieldTarget)
17362             If (Board(FieldPos) = WPAWN And Rank(FieldTarget) = 8) Or (Board(FieldPos) =
BPAWN And Rank(FieldTarget) = 1) Then
17363                 lResult = MsgBox(Translate("Promote to queen?"), vbYesNo) 'or Knight
17364                 If lResult = vbYes Then sPromotePiece = "q" Else sPromotePiece = "n"
17365             End If
17366             '--- Send move to Engine
17367             psMove = FieldNumToCoord(plFieldFrom) & FieldNumToCoord(plFieldTarget) &
sPromotePiece & vbLf
17368             ParseCommand psMove
17369             frmChessX.ShowMoveList
17370             frmChessX.ShowBoard
17371         Else
17372             If NumLegalMoves = 0 Then
17373                 If InCheck() Then
17374                     SendCommand "Mate!"
17375                 Else
17376                     SendCommand "No legal move -> Draw!!!"
17377                 End If
17378             Else
17379                 SendCommand "Illegal move: " & FieldNumToCoord(plFieldFrom) &
FieldNumToCoord(plFieldTarget) & " !!!"
17380             End If
17381         End If
17382
17383         'Reset
17384         plFieldFrom = 0: plFieldTarget = 0
17385         ResetGUIFieldColors
17386
17387         If bIsLegal And frmChessX.chkAutoThink = True Then
17388             DoEvents
17389             frmChessX.cmdThink_Click
17390             DoEvents
17391         End If
17392     End If
17393 End If
17394 Else
17395     ResetGUIFieldColors
17396 End If
17397 DoEvents
17398 End Sub
17399
17400
17401 Public Function FieldNumToBoardPos(ByVal ilFieldNum As Long) As Long
17402     Dim s As String
17403     s = FieldNumToCoord(ilFieldNum)
17404     FieldNumToBoardPos = FileRev(Left(s, 1)) + RankRev(Mid(s, 2, 1))
17405 End Function
17406
17407
17408 Public Function CheckGUIMoveIsLegal(MoveFromText, MoveTargetText, oLegalMoves As Long)
As Boolean
17409     ' Input: "e2", "e4", Output: oLegalMoves: Number of Legal Moves

```

```

17410 Dim a As Long, NumMoves As Long, From As Long, Target As Long
17411 CheckGUIMoveIsLegal = False
17412
17413 Ply = 0
17414 oLegalMoves = GenerateLegalMoves(NumMoves)
17415 If oLegalMoves > 0 Then
17416     From = FileRev(Left(MoveFromText, 1)) + RankRev(Mid(MoveFromText, 2, 1))
17417     Target = FileRev(Left(MoveTargetText, 1)) + RankRev(Mid(MoveTargetText, 2, 1))
17418
17419     For a = 0 To NumMoves - 1
17420         If Moves(0, a).From = From And Moves(0, a).Target = Target Then
17421             CheckGUIMoveIsLegal = Moves(0, a).IsLegal: Exit For
17422         End If
17423     Next a
17424 End If
17425 End Function
17426
17427 Public Sub ShowLegalMovesForPiece(MoveFromText)
17428     'Input: square as text "e2"
17429     Dim a As Long, NumMoves As Long, From As Long, Target As Long
17430     Dim NumLegalMoves As Long, ctrl As Control, bFound As Boolean
17431
17432     Ply = 0: bFound = False
17433     NumLegalMoves = GenerateLegalMoves(NumMoves)
17434     From = FileRev(Left(MoveFromText, 1)) + RankRev(Mid(MoveFromText, 2, 1))
17435     If NumLegalMoves = 0 Then
17436         SendCommand "No legal moves!"
17437     Else
17438         For Each ctrl In frmChessX.Controls
17439             Target = Val("0" & ctrl.Tag)
17440             If Target > 0 Then
17441                 For a = 0 To NumMoves - 1
17442                     If Moves(0, a).From = From And Moves(0, a).Target = Target And Moves(0, a).
17443                         IsLegal Then
17444                         ctrl.BackColor = &HCOFFC0
17445                         bFound = True
17446                     End If
17447                 Next a
17448             End If
17449         Next ctrl
17450         If Not bFound Then
17451             SendCommand "No legal move for this piece!"
17452         End If
17453     End If
17454 End Sub
17455
17456 Public Sub ResetGUIFieldColors()
17457     Dim x As Long, y As Long, bBackColorIsWhite As Boolean, i As Long
17458
17459     bBackColorIsWhite = False
17460
17461     For y = 1 To 8
17462         For x = 1 To 8
17463             i = x + (y - 1) * 8
17464             With frmChessX.fraBoard.Controls("Square" & i)
17465                 If bBackColorIsWhite Then
17466                     If .BackColor <> WhiteSqCol Then .BackColor = WhiteSqCol
17467                 Else
17468                     If .BackColor <> BlackSqCol Then .BackColor = BlackSqCol
17469                 End If
17470             End With
17471             bBackColorIsWhite = Not bBackColorIsWhite
17472         Next x
17473     bBackColorIsWhite = Not bBackColorIsWhite
17474 Next y
17475 End Sub
17476

```

```

17477
17478
17479 Public Function GenerateLegalMoves(olTotalMoves As Long) As Long
17480 ' Returns all moves in Moves(ply). Moves(x).IsLegal=true for legal moves
17481 Dim LegalMoves As Long, lLegalMoves As Long, i As Long, NumMoves As Long
17482
17483 GenerateMoves Ply, False, NumMoves
17484 Ply = 0: lLegalMoves = 0
17485
17486 For i = 0 To NumMoves - 1
17487     RemoveEpPiece
17488     MakeMove Moves(Ply, i)
17489     If CheckLegal(Moves(Ply, i)) Then
17490         Moves(Ply, i).IsLegal = True: lLegalMoves = lLegalMoves + 1
17491         Debug.Print MoveText(Moves(Ply, i))
17492     End If
17493     UnmakeMove Moves(Ply, i)
17494     ResetEpPiece
17495     'Debug.Print MoveText(Moves(0, i)), Moves(Ply, i).IsLegal
17496 Next
17497 olTotalMoves = NumMoves
17498 GenerateLegalMoves = lLegalMoves
17499 End Function
17500
17501 Public Sub ShowColToMove()
17502 With frmChessX.lblColToMove
17503     If bWhiteToMove Then
17504         .BackColor = vbWhite
17505         .ForeColor = vbBlack
17506         .Caption = Translate("White to move")
17507     Else
17508         .BackColor = vbBlack
17509         .ForeColor = vbWhite
17510         .Caption = Translate("Black to move")
17511     End If
17512 End With
17513 End Sub
17514
17515 Public Sub ShowLastMoveAtBoard()
17516 If GameMovesCnt = 0 Then Exit Sub
17517 ShowMove arGameMoves(GameMovesCnt).From, arGameMoves(GameMovesCnt).Target
17518 End Sub
17519
17520 Public Sub ShowMove(From As Long, Target As Long)
17521 ' show move on board with different bgcolor
17522 Dim Pos As Long, ctrl As Control
17523
17524 If From > 0 Then
17525     For Each ctrl In frmChessX.Controls
17526         Pos = Val("0" & ctrl.Tag)
17527         If Pos = From Then ctrl.BackColor = &HCOFFC0
17528     Next ctrl
17529 End If
17530
17531 If Target > 0 Then
17532     For Each ctrl In frmChessX.Controls
17533         Pos = Val("0" & ctrl.Tag)
17534         If Pos = Target Then ctrl.BackColor = &HCOFFC0
17535     Next ctrl
17536 End If
17537 End Sub
17538 SUB
17539
17540 ;=====
17541 ;=====
17542 ; =====
17543 ; chess engine ChessBrainVB V3 for winboard and UCI interfaces like Arena GUI. by

```



```

Roger Zuehltsdorf (2018)
17544 ; based on LarsenVB by Luca Dormio (http://xoomer.virgilio.it/ludormio/download.htm)
and Faile
17545 ; by Adrien M. Regimbald
17546 ; and ideas from Stockfish, Protector and other engines.
17547 ; Author: Roger Zuehltsdorf (2018)
17548 ;=====
=====

17549
17550 ;=====
=====

17551 ;--- Settings for chess engine
17552 ;=====
=====

17553 [Engine]
17554
17555 ; if not winboard path set: for ARENA GUI use XBOARD Mode
17556 XBOARD_MODE=1
17557 ; in winbord mode show PV line without extra character like x !
17558 WB_PV_IN_UCI=0
17559
17560 ; Opening book, empty entry for no book or rename file ( use ARENA main book for
better results )
17561 OPENING_BOOK=
17562 ;OPENING_BOOK=
17563 ;OPENING_BOOK=CB_BOOK.TXT
17564 ; if no book above is found then use a small dummy book for fun (1=active, 0 = off,
empty: OfficeMode=1, UCI/WB=0)
17565 USE_INTERNAL_BOOK=
17566 ;USE_INTERNAL_BOOK=1
17567 ;USE_INTERNAL_BOOK=0
17568
17569 ; set number of threads (max 8); 0 or 1: normal single thread, 2-8: multiple processes
of EXE running
17570 ; Winboard "cores" command overrides this setting if cores value > THREADS (UCI
setting overrides too )
17571 THREADS=1
17572 ; Threads send from GUI ignored if set
17573 THREADS_IGNORE_GUI=0
17574 ;THREADS_IGNORE_GUI=1
17575
17576 ; Hash size in MB. (31 bytes per entry) (UCI/WB settings overrides if higher)
17577 ; max 1400 MB
17578 HASHSIZE=64
17579 ; Hash used value will be overwritten by main task to communicate with helper threads
17580 HASH_USED=64
17581 ; Hash size send from GUI ignored if set
17582 HASHSIZE_IGNORE_GUI=0
17583 ;HASHSIZE_IGNORE_GUI=1
17584
17585 ; Verify Hash reads to handle hash collisions, slows down engine 5%, but avoids some
bad moves
17586 HASH_VERIFY=1
17587
17588 ; Overhead for move communication to GUI and start/stop thinking in milliseconds (more
threads may need more? 1000ms)
17589 ; recommended for ARENA: 800ms, for ChessGUI 1000ms, for SMP 1500ms
17590 MOVEOVERHEAD=1000
17591
17592 ; Contempt value: draw score in centipawns(100= 1Pawn) from engine view. Against
better engine set positive value
17593 CONTEMPT=1
17594
17595 ; UCI/WB-Mode:SYZYGY ; Office(EXCEL): Online probe ): Enabled endgame tablebase access
: 0= disabled, 1 = enabled.
17596 EGTB_ENABLED=1
17597 ;EGTB_ENABLED=0
17598

```

```

17599 ; Syzygy endgame tablebases
17600 ; PATH = path for Syzygy files ( if path is empty then programs sets EGTB_ENABLED=0 )
17601 ; MAX_PIECES = max piece count for table base probe (3,4,5,6)
17602 ; MAX_PLY = tb access first first x plies only (slow access): 1=root
17603 ; UCI GUI may overwrite this settings; winboard uses this settings here
17604
17605 ;TB_SYZYGY_PATH="C:\Chess\TB\Syzygy\Syzygy 3-4-5"
17606 TB_SYZYGY_PATH=""
17607 TB_SYZYGY_MAX_PIECES=5
17608 TB_SYZYGY_MAX_PLY=3
17609
17610 ;--- online tablebase access used for Office VBA GUI only
17611 ; Endgame table base online Web service.
17612 ; First call needs about 15 seconds to init connection.
17613 ; Conditions for use of TB: 5 pieces or less on board + minimum 20 seconds time
    remaining for engine
17614 ; Used for PLY=1 only because too slow for deep searches
17615 ; Lokasoft Web service is used, 5 pieces
17616 TB_ONL_URL="http://www.lokasoft.nl/tbweb/tbapi.wsdl"
17617
17618
17619 ;=====
    =====
17620 ;--- Evaluation of position (factor in percent: 100 = unchanged score, 0 = zero score,
    200 = double score)
17621 ;=====
    =====
17622 ; Position: piece position evaluation values(i.e. piece square tables)
17623 POSITION_FACTOR=100
17624
17625 ; Mobility: mobility of pieces
17626 MOBILITY_FACTOR=100
17627
17628 ; Pawn structure: pawn value depending on supported, isolated, backwards,...
17629 PAWNSTRUCT_FACTOR=100
17630
17631 ; Passed pawns: passed pawns value depending on rank, safe advance to promote square
    ,...
17632 PASSEDPAWNS_FACTOR=120
17633 ;PASSEDPAWNS_FACTOR=100
17634
17635 ; Threats: bonus for threats at opponent pieces depending on piece types
17636 THREATS_FACTOR=100
17637 ;THREATS_FACTOR=100
17638
17639 ; Opponent king attack: bonus for safe king shelter, penalty for attack options
17640 OPPKINGATT_FACTOR=100
17641 ;OPPKINGATT_FACTOR=100
17642
17643 ; Computer king defense: bonus for safe king shelter, penalty for attack options
17644 COMPKINGDEF_FACTOR=100
17645 ;COMPKINGDEF_FACTOR=100
17646
17647 ;=====
    =====
17648 ;--- Piece values (MG: Midgame, EG: Endgame for scaling using game phases) based on
    Stockfish6 ---
17649 ;--- This values are default in engine if entries are missing here
17650 ;=====
    =====
17651 PAWN_VAL_MG=142
17652 PAWN_VAL_EG=207
17653
17654 KNIGHT_VAL_MG=784
17655 KNIGHT_VAL_EG=868
17656
17657 BISHOP_VAL_MG=828
17658 BISHOP_VAL_EG=916

```

```
17659
17660 ROOK_VAL_MG=1286
17661 ROOK_VAL_EG=1378
17662
17663 QUEEN_VAL_MG=2528
17664 QUEEN_VAL_EG=2698
17665
17666 ; for game phase calculation
17667 MIDGAME_LIMIT=15258
17668 ENDGAME_LIMIT=3915
17669
17670
17671 ;=====
17672 ;--- Debug settings
17673 ;=====
17674 ;enable PV log = 1 ; disable PV log =0 , same as command parameter "-log".
17675 LogPV=0
17676
17677 ;Trace file settings: 0 / 1
17678 EVALTRACE=0
17679 TIMETRACE=0
17680 HASHTRACE=0
17681 HASH_COLL_TRACE=0
17682 COMMANDTRACE=0
17683 TBBASE_TRACE=0
17684 THREADTRACE=0
17685
17686 ;=====
17687 ;--- MS OFFICE GUI settings (not used for winboard engine)
17688 ;=====
17689
17690 ; Translate for language: DE => ChessBrainVB_Language_DE.txt
17691 LANGUAGE=EN
17692
17693 ; Color for GUI board squares
17694 ;WHITE_SQ_COLOR = "&H00C0FFFF&"
17695 WHITE_SQ_COLOR = "&HC0FFFF"
17696 ;BLACK_SQ_COLOR = "&H0080C0FF&"
17697 BLACK_SQ_COLOR = "&H80FF&"
17698
17699 BOARD_FRAME_COLOR = &H000040C0&
17700
17701 ;--- Chess test positions (EXCEL GUI) in FEN (EPD) format (from WAC (Win At Chess))
17702 test set)
17703 TEST_POSITION1 = "1b5k/7P/p1p2np1/2P2p2/PP3P2/4RQ1R/q2r3P/6K1 w - - bm Re8+; id
17704 WAC.250;Mate in 8;"
17705 TEST_POSITION2 = "2k4B/bpp1qp2/p1b5/7p/1PN1n1p1/2Pr4/P5PP/R3QR1K b - - bm Ng3+; id
17706 WAC.273;"
17707 TEST_POSITION3 = "r3q2r/2p1k1p1/p5p1/1p2Nb2/1P2nB2/P7/2PNQbPP/R2R3K b - - bm Rxh2+;
17708 id WAC.266;"
17709 TEST_POSITION4 ="8/6k1/6p1/8/7r/3P1KP1/8/8 w - - 0 1; Tablebase test"
17710 ;=====
17711
17712 TB_ROOT_ENABLED=0
17713 TB_SEARCH_ENABLED=0
```