```
COMMENT awit.alg -- version of Awit.alg.orig for AW2C ;

@NOCHECK
@DEBUG,0
COMMENT
        Copyright 1978, T.A. Marsland
        University of Alberta, Edmonton, Canada.
        This version of the chess program Awit (alias WITA)
        has been under development since 1971.
        It is made available for reference purposes only,
        and is not to be reproduced or redistributed
        without the permission of the author.
;
COMMENT
$R *SORT PAR=SORT=CH:A:1:8 INPUT=AWIT.DIR:F:100:100 OUTPUT=*PRINT*
;

COMMENT                 MTS dependent interface
       SUBROUTINE DEPEND(IND,HIST)
       INTEGER IND, INDEX, HIST(35)
       INTEGER*2 LENGTH/140/, DUMMY
       INDEX = IABS(IND*1000)
       IF (IND .GT. 0) CALL WRITE(HIST, LENGTH, 16386, INDEX, 8)
       IF (IND .LT. 0) CALL READ (HIST,  DUMMY, 16386, INDEX, 8)
       IF(IND.NE.0 .AND. IND.GT.-9000) RETURN
       CALL CUINFO('ECHOOFF ', 1)
       IF(IND .EQ. -9999)
     * CALL MTSCMD('$RES 9=WITA:GAMES(LAST+10)@TRIM ',32)
       IF (IND .GT. -10000) GO TO 15
       INDEX = MOD(IABS(IND),10)
       CALL CLOSFL(INDEX)
       CALL UNLK(INDEX)
15     IF (IND .NE. 0) GO TO 20
       CALL MTSCMD('$RES 3=WITA:CHESS.TABLE ', 24)
       CALL MTSCMD('$RES SCARDS=*SOURCE*@UC ',24)
       CALL GUINFO('SIGNONID', HIST(35))
C           Check to see if unit 8 is assigned.
       CALL GDINF(8, HIST(23), &30)
C           Yes it is, make sure it exists.
       CALL WRITE(HIST, 0, 2, -99999999, 8)
C           MTS will prompt for valid file name.
       GO TO 25
30     CALL MTSCMD('$RES 8=-CHESS#(LAST) ',21)
25     CALL GDINF(2, HIST(23),&10)
C            Unit 2 is already attached.
       GO TO 20
10     CALL MTSCMD('$RES 2=WITA:BOOK@UC ',20)
20     CALL CUINFO('ECHOOFF ', 0)
       RETURN
       END
C
       SUBROUTINE TRANSF(B, L, M, N, U, RW)
       INTEGER B(64), L, M, N, U, RW
       INTEGER LL
       INTEGER*2 HALF(2)
       EQUIVALENCE (HALF,LL)
       LL = L
       HALF(1) = 0
       IF (RW .GT. 0) CALL WRITE(B, HALF(2), M, N, U)
       IF (RW .LT. 0) CALL READ (B, HALF(2), M, N, U, &10)
C         MTS version 4 returns a length of zero upon EOF read     C
       IF (RW .EQ. 0) CALL GETLST(U, N, &15)
20     L = LL
       RETURN
15     N = 0
10     LL = -1
       GO TO 20
       END
;
@TITLE,"Awit-WITA CHESS PROGRAM"
```

```
BEGIN
  COMMENT Dummy MTS procedures;
  PROCEDURE GET (INTEGER VALUE UNIT) ; ;
  PROCEDURE PUT (INTEGER VALUE UNIT) ; ;

  COMMENT a tracing procedure is entered if the first symbol typed
    is "?".  Different actions are taken depending on the characters
    which follow.  Explicitly mention squares, e.g., piece/squarefrom
    * piece/squareto. The command STOP terminates the program ;
  COMMENT *****************declarations************* ;
  INTEGER MAXTREE, MAXCHAR, MAXLEV, LIM, BAD, TOTAL,TSIZE, WSIZE,
    WIDTH, ELIMIT, RLIMIT, ZEROTIME, CONLIM, HOLE, MAXPLY, W_LIM,
    BRAN, MAX_PLIES, UPPER, T_SIZE, HALFP, GOODCASTLE, WEAKPIN,
    T_NUMB, TIME_LEFT,MOVES_LEFT, SECONDARY_MOVES, DIR, DISCHECK,
    M4, PREDICT, DRAW, PAWNVALUE, THRESHOLD, EPSILON, NOSCR, TWOCHECK,
    NOD, DUP, REC, SAL, BASE,  DYN, TOT, ROOK, KNIGHT, BISHOP,
    QUEEN, KING, PAWN, ENPRIS, RANK, FILE, LDIAG, RDIAG, QSIDE,
    KSIDE, KLIST, ENDPLAY, PROMOTE, EMPTY, PFTWO, DELTA, NIL, KTEST,
    CAP, RTMV, P_NUMB, BACKROW, MOVE, CLIST, K, NUM, ONUM, CNUM,
    SLIST, INDEX, OCON, CCON, NUMB, PROMCAPT, MOVECOUNT, CAPT_T,
    CHARCOUNT, FILL, KINGVAL, INITIALCNT, N, WITA, CHECKING, NINES,
    EIGHTS, SEVENS, SIXES, FIVES, TWOROWS, M1, M2, M3, M5, M6,
    M7, M8, M9, M10, M11, M12, M14, KSQ, MAXPIECE, MINPIECE, LOSSES,
    SPARE0, SENDTIME, CPU, OLDCPU, ELAPSEDTIME, COMPUTERTIME, USERTIME,
    USER, S_SIZE, CHECKSQ, DATUM, COUNT, DIFFVAL, REALVAL, ONEVAL,
    TWOVAL, MOVETIME, AGITATE, SCALE, PVAL, BRDWIDTH, EDGE, WQRSQ,
    BKRSQ, POINT, SKTEST, CTIME, HASH, BOOK, BOOKMOVE, PONENT,
    P, REALCHECK, LENGTH, LEVEL, SECONDARY_TIME, MAXNODES, PLEX,
    REPLY, ORIGIN, LASTORIGIN, MAXL, SAVEL, PATH,  OLDSCR, PREVSCR,
    ASCORE,DSCORE, PP_STATUS, TTYOUT, NEW, LIMBS, PSCORE, BRAN_S,
    REC_S, DUP_S, TOT_S, MODE, PART, A_SCR, MAX_WID, CAPT_WID,
    OLD_DIR, OLD_DEST, LSAVE, ESAVE, KNIGHTVALUE, NBP, SPARE5,
    DEPTH_LIMIT, WIDTH_LIMIT ;
  LOGICAL REVERSIBLE, PPIN, ENDGAME, OPENING, LOG1, LOG2, MONITOR,
    MON, FLAGS, CASTLING, KINGUSED, FORCEDREPLY, PRUNE, DUMMY6,
    ECHO, NEWGAME, EXAMINE, REMEMBER, LEGAL, EARLY, WHO, BOTH,
    DEBUG, QUICK, TERMINAL, GARDE, TEST, NULLMOVE, DUMPP, CBOTH,
    WS, LOOKAHEAD, TOURNAMENT, MAKEMV, SCORES, INCHECK, DBLCHECK,
    PINNED, ENGLISH, COKO, GIVINGCH, MATE, DOUBLECH,
    STALEM, CAPTURE_TREE, LATE_END, ALTERNATIVE, INERR, IND_ATTA,
    PAB, THINKING, DUMMY4, DUMMY5 ;

COMMENT   in order to emphasize the three levels of this chess
program, the arrays that are associated with each level have been
separated.  The three levels are :: - conversational, occuring
whenever the user can input data - position generation, whenever
a new node of the search tree is created - and move scoring, in
which many moves from a fixed position (node) are studied. ;

COMMENT *****************conversational*************: ;

  STRING (12) DATE, SKILL ;
  STRING (4) USERID ;
  STRING (3) BLANK ;
  STRING (1) SPACE ;
  STRING (5) ARRAY COLOR (-1::1) ;
  STRING (1) ARRAY CHESSMAN (0::11) ;
  STRING (1) ARRAY CHESSPIECE (0::8) ;
  STRING (1) ARRAY U (0::120) ;
COMMENT string (1) array u (0::maxchar);
  STRING (140) BUFFER, HISTORY ;
  STRING (80) INPUT, OUTPUT ;
  STRING (3) ARRAY MAN (-8::8) ;
  STRING (1) ARRAY MC (0::23) ;
  STRING (1) ARRAY DIGIT (0::9) ;
  STRING (8) ARRAY REASON (-7::5) ;
  STRING (34) MESSAGE1, PREV_MESSAGE1 ;
  STRING (9) MESSAGE2, PREV_MESSAGE2 ;
  INTEGER ARRAY FSQ, TSQ (0::1) ;
  INTEGER ARRAY REW, FMAN, TMAN (0::2) ;
```

```
141   INTEGER ARRAY NUMBER, SPAN, KINGSQ (-1::1) ;
142
143 COMMENT ****************positional******************* ;
144
145   INTEGER ARRAY S (0::8) ;
146 COMMENT integer array try (0::wize) ;
147   COMMENT rlimit >= wsize ;
148   INTEGER ARRAY TRY (0::20) ;
149   INTEGER ARRAY VALUES (0::12) ;
150   INTEGER ARRAY RLIST, TLIST (0::15) ;
151 COMMENT integer array rlist, tlist (0::rlimit);
152   INTEGER ARRAY ATTASQRS (0::10) ;
153   INTEGER ARRAY POWER (-1::10) ;
154   INTEGER ARRAY SQR (-5::10) ;
155   INTEGER ARRAY PRIMES (-14::14) ;
156   INTEGER ARRAY RANKS, PAWNVAL (1::8) ;
157   INTEGER ARRAY REPEATS (0::100) ;
158   INTEGER ARRAY TWO, THREE, K_SQRS, FILES, QRSQ, KRSQ (-1::1) ;
159   INTEGER ARRAY CHECKS, PAWNS (11::88) ;
160   INTEGER ARRAY COLS, ROWS (10::89) ;
161   INTEGER ARRAY KSQRS, PIECES (-8::8) ;
162   INTEGER ARRAY CHECKLIST, OLDLIST (0::29) ;
163 COMMENT integer array checklist, oldlist (0::lim);
164   INTEGER ARRAY MOVEFROM, MOVETO, SCORE, REWARD (-80::80) ;
165 COMMENT integer array movefrom, moveto, score, reward (-total::total);
166 COMMENT integer array merit (-total::total) ;
167   INTEGER ARRAY POSITION (11::88,0::29) ;
168 COMMENT integer array position (11::88,0::lim);
169   INTEGER ARRAY POS (11::88) ;
170   INTEGER ARRAY CONTROL (11::88,-8::8) ;
171 COMMENT integer array control (11::88,-conlim::conlim);
172   INTEGER ARRAY CON (-1::1,11::88) ;
173   INTEGER ARRAY BRD (-11::109) ;
174 COMMENT integer array brd (hole::109);
175   INTEGER ARRAY SQUARE, LOSS (0::31) ;
176 COMMENT integer array square, loss (0::elimit);
177   LOGICAL ARRAY CASTLE (-3::3) ;
178   INTEGER ARRAY NEAR, FAR (0::7,0::7) ;
179   INTEGER ARRAY CLEARPAWN(-1::1) ;
180   STRING (6) KBUFF ;
181   STRING (2) KEY ;
182   INTEGER ARRAY FLAG(1::1) ;
183   STRING(1) BELL ;
184   STRING(2) ARRAY BOX(10::88) ;
185   INTEGER ARRAY OFFSET(-10::109) ;
186   INTEGER ARRAY BOTV(1::8, -112::112) ;
187   INTEGER ARRAY MATE_P, MATE_T(-1::1) ;
188   INTEGER ARRAY MEN(-1::1) ;
189   LOGICAL ARRAY OPEN(-1::1) ;
190 COMMENT  This first attempt at a chess playing program is very
191 unsophisticated.  Its level of play is that of advanced beginner.
192 It was the aim of this program to play reasonable middle game chess.
193 At any stage in the game the seven best moves are selected on a
194 basis of mobility, modified by material gain and checking threats.
195 For each of these moves the attacking and defensive potential is
196 computed and the overall score modified again. Move with highest
197 score is selected. The basic book of opening moves are the Najdorf,
198 Gruenfeld and Two Knights Defence. Another book (from K. Thompson)
199 was incorporated during the summer of 1974, since the scheme to
200 inhibit early queen moves was no longer adequate.  There is no special
201 section on endgames. Such additions would certainly be advantageous.
202 Until the incorporation of chess features and determination of their
203 relative merit is complete, it is not thought to be wise to construct
204 trees greater than the current default of 100 positions.  It is
205 claimed that by constructing a program which plays as well as possible
206 using a small tree, one can ensure that a minimal number of irrelevant
207 branches will be searched in any larger tree. ;
208
209   INTEGER ARRAY A_VAL(0::12) ;
210 COMMENT integer array store (-4*(2*total+1)::4*(2*total+1)) ;
```

```
211    INTEGER ARRAY STORE (-644::644) ;
212    INTEGER ARRAY M_TIMES(0::10) ;
213    INTEGER ARRAY FKS(11::88) ;
214    INTEGER ARRAY BACK_ROW(-1::1, 1::8) ;
215    INTEGER ARRAY CH_ATTA(-11::109) ;
216    LOGICAL ARRAY CAPTURE_CH(-11::109) ;
217    INTEGER ARRAY CHECK_S(-1::1, -11::109) ;
218    INTEGER ARRAY FORKS(-1::1, 11::88) ;
219    INTEGER ARRAY DOUBLE_KNIGHT(0::49) ;
220    INTEGER ARRAY VARS(1::10, 1::10) ;
221    COMMENT 10 ply ;
222    INTEGER ARRAY USE(0::80) ;
223    COMMENT integer array use(0::total);
224    LOGICAL ARRAY CHEC(0::5, 0::11) ;
225    INTEGER ARRAY SECRET(11::88, -8::8) ;
226    INTEGER ARRAY SEC(-1::1, 11::88) ;
227    LOGICAL ARRAY KING_HELD(-1::1);
228    INTEGER ARRAY DECOYS (-11::109);
229
230    LOGICAL ARRAY QUIES(-80::80);
231    STRING (4) ARRAY NEGA(0::100);
232  @TITLE "INITIALIZE"
233    PROCEDURE INITIALIZE ;
234    BEGIN
235      MMTTS(0,HISTORY) ;
236      USERID := HISTORY(136|4) ;
237      COLOR(-1) := "BLACK" ;
238      COLOR(0) := "WHOSE" ;
239      COLOR(1) := "WHITE" ;
240      BELL := "^G" ;
241     COMMENT bell char. is hex 2F ;
242      P := 99999971 ;
243      HALFP := (P-1) DIV 2 ;
244      ROOK := 5 ;
245      KNIGHT := 3 ;
246      BISHOP := 4 ;
247      QUEEN := 7 ;
248      KING := 8 ;
249      PAWN := 1 ;
250      ENPRIS := 11 ;
251      RANK := 1 ;
252      FILE := 10 ;
253      LDIAG := 9 ;
254      RDIAG := 11 ;
255      QSIDE := 9 ;
256      KSIDE := 10 ;
257      BRDWIDTH := 10 ;
258      QRSQ(-1) := 81 ;
259      QRSQ(1) := WQRSQ := 11 ;
260      KRSQ(1) := 18 ;
261      KRSQ(-1) := BKRSQ := 88 ;
262      EDGE := 6 ;
263      PFTWO := 12 ;
264      EMPTY := NIL  := 0 ;
265      PROMOTE := 10 ;
266      BLANK :=  "   " ;
267      SPACE := " " ;
268      CHECKING := 20 ;
269      REALCHECK := 2*CHECKING ;
270      DISCHECK := 3*CHECKING ;
271      TWOCHECK := 4*CHECKING ;
272      NINES := 88888 ;
273      EIGHTS := 99999 ;
274      SEVENS := 77777 ;
275      SIXES := 66666 ;
276      FIVES := 55555 ;
277      TWOROWS := 2*FILE ;
278      DATUM := 500 ;
279      SCALE := 150 ;
280      PVAL := 6 ;
```

```
281       PAWNVALUE := PVAL*SCALE ;
282     KNIGHTVALUE := 3*PAWNVALUE;
283       THRESHOLD := PAWNVALUE DIV 3 ;
284       EPSILON := THRESHOLD DIV 3 ;
285       DRAW :=  + 4*SCALE ;
286     COMMENT 1/2 PAWN ;
287       FOR I := 0 UNTIL 23
288       DO MC(I) := SPACE ;
289     COMMENT the values of the pieces are given below ;
290       FOR K := -1,1
291       DO BEGIN
292         TWO(K) := 2*K ;
293         THREE(K) := 3*K ;
294         K_SQRS(K) := 0 ;
295         FILES(K) := FILE*K ;
296       END ;
297       FOR FILL := 1  UNTIL 8
298       DO RANKS(FILL) := CASE FILL OF (-1,-1,-1,-1,1,1,1,1) ;
299       FOR FILL := 1 UNTIL 9
300       DO S(FILL-1) := CASE FILL OF (0,12,19,-12,-19,21,8,-21,-8) ;
301       FOR FILL := 1 UNTIL 10
302       DO DIGIT (FILL-1) := CASE FILL OF ( "0", "1",  "2", "3", "4",
303         "5", "6", "7", "8", "9") ;
304       FOR FILL := -KING UNTIL KING
305       DO MAN(FILL) := CASE (FILL + 1 + KING) OF (" Kb", " Qb",
306         "    ", " Rb", " Bb", " Nb", "    ", " Pb", "    ", " Pw",
307         "    ", " Nw", " Bw", " Rw", "###"," QW"," KW") ;
308       FOR FILL := 1 UNTIL 13
309       DO A_VAL(FILL-1) := CASE FILL  OF (0, PVAL, 0, 18, 20, 25,
310         0, 30, PVAL, 5, DATUM, 4, SCALE) ;
311       FOR FILL := 1 UNTIL 13
312       DO VALUES(FILL-1) := CASE FILL OF ( 0, PVAL, 0, 18, 20, 30,
313         0, 57, 60, 5, DATUM, 4,SCALE) ;
314       FOR FILL := 1 UNTIL 12
315       DO CHESSMAN (FILL-1) := CASE FILL OF ( " ", "P",  "#", "N",
316         "B", "R", "@", "Q", "K", "@", " ", "*") ;
317       FOR FILL := 1 UNTIL 9
318       DO CHESSPIECE(FILL-1) := CASE FILL OF ( " ", "R", "N", "B",
319         "Q", "K", "B", "N", "R") ;
320       FOR FILL := 1 UNTIL 13
321       DO REASON(FILL-8) := CASE FILL OF ("NODES   ", "DEPTH   ",
322         "BETTER  ", "ANTICIP ", "NO PROS ", "GREAT   ",
323         "REVERSE ", "LIMIT   ", "RESIGN  ", "STALE M.",
324         " DRAW   ", " MATE   ", "KING CAP") ;
325   COMMENT  King and passed pawn opposition table.
326
327              FAR  (ENDGAME)             NEAR  (~ENDGAME)
328
329        7 -  5  4  3  2  1  0  0  0     0 0 0 0 0 0 0 0
330        6 -  7  6  5  4  3  2  1  0     0 0 0 0 0 0 0 0
331        5 -  9  8  7  6  5  4  2  0     0 0 0 0 0 0 0 0
332        4 - 11 10  9  8  7  5  3  1     2 1 0 0 0 0 0 0
333        3 - 13 12 11 10  8  6  4  2     4 3 2 0 0 0 0 0
334        2 - 15 14 13 11  9  7  5  3     6 5 4 2 0 0 0 0
335        1 - 17 16 14 12 10  8  6  4     8 7 5 3 1 0 0 0
336        0 - 19 17 15 13 11  9  7  5    10 8 6 4 2 0 0 0
337
338              0  1  2  3  4  5  6  7
339     ;
340       FOR FILL := 1 UNTIL 8
341       DO BEGIN
342         PAWNVAL(FILL) := CASE FILL OF (4, 4, 2, 1, 1, 2, 4, 4) ;
343         BACK_ROW(-1,FILL) := 80+FILL ;
344         BACK_ROW(1,FILL) := 10+FILL ;
345       END ;
346       COLS(10) := 0 ;
347       FOR I := 1 UNTIL 8
348       DO FOR J := 0 UNTIL 9
349          DO BEGIN
350            FILL := BRDWIDTH*I +J ;
```

```
          ROWS(FILL) := I ;
          COLS(FILL) := J ;
    END ;
COMMENT Pawn location and control table
        especially for pawns on 7th and in centre

       RANKS    PAWNS                                ROWS PAWNVAL

        PAWN     0   0   0   0   0   0   0   0      8        8
        PAWN     1   1   1   1   1   1   1   1      7    4   7
        PAWN     2   4   6   6   6   6   4   2      6    2   6
        PAWN     2   4   6   8   8   6   4   2      5    1   5
       -PAWN     2   4   6   8   8   6   4   2      4    1   5
       -PAWN     2   4   6   6   6   6   4   2      3    2   6
       -PAWN    -1  -1  -1  -1  -1  -1  -1  -1      2    4   7
       -PAWN     0   0   0   0   0   0   0   0      1        8

        COLS     a   b   c   d   e   f   g   h
;
    GET(3) ;
    IOCONTROL(1) ;
    FOR I := -10 UNTIL 109
    DO READON(OFFSET(I)) ;
    FOR K := 3 UNTIL 8
    DO BEGIN
      IOCONTROL(1) ;
      FOR I := -112 UNTIL 112
      DO READON(BOTV(K, I)) ;
    END ;
    IOCONTROL(1) ;
    FOR I := 0 UNTIL 7
    DO FOR J := 0 UNTIL 7
    DO READON(FAR(I,J)) ;
    IOCONTROL(1) ;
    FOR I := 0 UNTIL 7
    DO FOR J := 0 UNTIL 7
    DO READON(NEAR(I,J)) ;
    IOCONTROL(1) ;
    FOR I := 11 UNTIL 88
    DO READON(PAWNS(I)) ;
    FOR K := 1,2
    DO BEGIN
      IOCONTROL(1) ;
      FOR I := -112 UNTIL 112
      DO READON(BOTV(K,I)) ;
    END ;
    IOCONTROL(1) ;
    FOR I := 10 UNTIL 88
    DO READON(BOX(I)) ;
    IOCONTROL(1) ;
    FOR I := 0 UNTIL 49
    DO READON(DOUBLE_KNIGHT(I)) ;
    FOR I := 1 UNTIL 10
    DO BEGIN
      IOCONTROL(1) ;
      FOR J := 1 UNTIL 10
      DO READON(VARS(I,J)) ;
    END ;
    IOCONTROL(1) ;
    FOR I := 0 UNTIL 5
    DO FOR J := 0 UNTIL 11
    DO READON(CHEC(I,J)) ;
    GET(5) ;
    KINGVAL := VALUES(KING) ;
    MOVECOUNT := NUM := 0 ;
    BAD := - 66667 ;
    NOSCR := BAD +1 ;
    COMMENT DELTA := ABS(VALUES(BISHOP) - VALUES(KNIGHT)) + 1 ;
    DELTA := 3;
    ECHO := DUMPP := DEBUG := MONITOR := TEST := COKO := THINKING :=
```

```
421       ENGLISH := CAPTURE_TREE := LOOKAHEAD := EXAMINE := FALSE ;
422     WS := TRUE;
423   ORIGIN := -1;
424     DATER(5,0,DATE) ;
425     WRITE(DATE) ;
426     PAB := QUICK  := TRUE ;
427     TOURNAMENT := MAX_WID = WSIZE -1 ;
428     WRITE("Awit, a chess program, ") ;
429     WRITE("Enter ?8 for control options, STOP to terminate.") ;
430     MMTTS(-10003, HISTORY) ;
431     TRANSFER(HISTORY, NUM, 0, COUNT, 8, 0) ;
432     COUNT := COUNT DIV 1000 ;
433   COMMENT  normally COUNT is 0, but initialized here
434            through indexed read on last record of history file;
435     FOR K := -1, 1
436     DO FOR SQ := 11 UNTIL 88
437     DO  CHECK_S(K,SQ) := FORKS(K,SQ) := 0 ;
438   END OFINITIALIZE ;
439
440 @TITLE,"ALGEBRAIC"
441 PROCEDURE ALGEBRAIC (STRING(1) ARRAY G(*) ;
442 INTEGER VALUE PTR) ;
443 BEGIN
444   COMMENT Algebraic input routines developed for T.A.M. by E. Culham,
445     under NRC grant A7902, July 1977 ;
446
447 INTEGER ROW, COL, TEMPPC, TCOL, OFFSET;
448   STRING(1) T ;
449   LOGICAL PAWN_CAPT ;
450
451   LOGICAL PROCEDURE GETOK(LOGICAL VALUE YOU_WANT_THE_ROW) ;
452   BEGIN
453     LOGICAL RESLT ;
454     RESLT := FALSE ;
455     IF YOU_WANT_THE_ROW
456     THEN BEGIN
457       IF (T >= "1") AND (T <= "9")
458       THEN BEGIN
459         ROW := DECODE(T) - 240 ;
460         RESLT := TRUE ;
461       END ;
462     END ELSE
463     BEGIN
464       IF (T >= "A") AND (T <= "H")
465       THEN BEGIN
466         COL := DECODE(T) - 192 ;
467         RESLT := TRUE ;
468       END ;
469     END ;
470     RESLT
471   END OFGETOK ;
472
473   LOGICAL PROCEDURE NEXT(STRING(1) RESULT T) ;
474   BEGIN
475     LOGICAL RESLT ;
476     PTR := PTR - 1 ;
477     IF  PTR < 0
478     THEN RESLT := FALSE
479     ELSE BEGIN
480       T := G(PTR) ;
481       RESLT := TRUE ;
482     END ;
483     RESLT
484   END OFNEXT ;
485
486   PROCEDURE GETPC ;
487   BEGIN
488     FOR I := KNIGHT,BISHOP,ROOK,QUEEN,KING, PAWN
489     DO IF T = CHESSMAN(I)
490        THEN BEGIN
```

```
491           FMAN(2) := I ;
492           IF NEXT(T)
493           THEN BEGIN
494             MESSAGE1 := "TOO MUCH MOVE TO DECODE" ;
495             GO TO ERROR ;
496           END ;
497           GO TO EXIT_FROM_FORI ;
498       END OFFORI ;
499   EXIT_FROM_FORI:
500     END OFGETPC ;
501
502   @TITLE,"ALGEBRAIC"
503     TSQ(1) := FSQ(1) := -NINES ;
504     IF NEXT(T)
505     THEN IF G(0) ~= "?"
506          THEN BEGIN
507            COMMENT  strip off characters from the book ;
508            IF T = "?" OR T = "!"
509            THEN IF ~NEXT(T)
510                 THEN GO TO ERROR ;
511
512   COMMENT test for draw offer ;
513           IF T = "W"
514           THEN BEGIN
515             PTR := PTR - 2 ;
516             IF NEXT(T) AND T = "D"
517             THEN IF (SCORE(RTMV) <  + PAWNVALUE)
518                  THEN BEGIN
519                    WRITE("DRAW ACCEPTED") ;
520                    GO TO STARTS ;
521             END ELSE MESSAGE1 := "DRAW NOT ACCEPTED AT THIS TIME" ;
522             IF ~NEXT(T)
523             THEN GO TO ERROR ;
524           END ;
525
526   COMMENT test for a check ;
527           IF T = "+"
528           THEN BEGIN
529             REW(0) := REW(0) + CHECKING ;
530             IF ~NEXT(T)
531             THEN GO TO ERROR ;
532           END ;
533
534   COMMENT test for enpassant capture ;
535           IF T = "P"
536           THEN BEGIN
537             IF ~NEXT(T) OR T ~= "E" OR ~NEXT(T)
538             THEN GO TO ERROR
539             ELSE TMAN(2) := ENPRIS
540           END ELSE IF PTR >= 3
541                    THEN BEGIN
542                      COMMENT test if it was a promotion ;
543                      FOR J := KNIGHT,BISHOP,ROOK,QUEEN
544                      DO IF  T = CHESSMAN(J)
545                         THEN IF ~NEXT(T) OR T ~= "=" OR ~NEXT(T)
546                              THEN BEGIN
547                                MESSAGE1 := "ILLEGAL PROMOTION TYPE" ;
548                                GO TO ERROR ;
549                      END ELSE
550                      BEGIN
551                        REW(2) := J + PROMOTE ;
552                        GO TO LEAVE_FOR_J ;
553                      END OFFORJ ;
554   LEAVE_FOR_J:
555           END ;
556
557   COMMENT only simple moves and castling remain ;
558   @TITLE "DESCRIBEMOVE"
559           IF GETOK(TRUE)
560           THEN BEGIN
```

```
561             IF NEXT(T) AND GETOK(FALSE)
562             THEN BEGIN
563               TSQ(0) := BRDWIDTH * ROW + COL ;
564               TMAN(2) := ABS(BRD(TSQ(0))) ;
565               IF ~NEXT(T)
566               THEN FMAN(2) := PAWN
567               ELSE BEGIN
568                 IF (T = "X") OR (T = ":") OR (T = "*")
569                 THEN IF ~NEXT(T)
570                      THEN GO TO ERROR ;
571                 GETPC ;
572                 IF FMAN(2) = 0
573                 THEN BEGIN
574                   IF (T = "-")
575                   THEN IF ~NEXT(T)
576                        THEN GO TO ERROR
577                        ELSE IF ~GETOK(FALSE) AND ~GETOK(TRUE)
578                             THEN GOTO ERROR ;
579                   IF GETOK(FALSE)
580                   THEN BEGIN
581                     COMMENT potential pawn capture ;
582                     ROW := ROW -K ;
583                     IF NEXT(T)
584                     THEN GETPC
585                     ELSE FMAN(2) := PAWN ;
586                     FSQ(0) := BRDWIDTH*ROW + COL ;
587                     IF FMAN(2) ~= ABS(BRD(FSQ(0)))
588                     THEN GOTO ERROR ;
589                     PAWN_CAPT := TRUE ;
590                   END ELSE PAWN_CAPT := FALSE ;
591                   IF GETOK(TRUE) OR PAWN_CAPT
592                   THEN BEGIN
593                     IF PAWN_CAPT OR NEXT(T) AND GETOK(FALSE)
594                     THEN BEGIN
595                       FSQ(0) := BRDWIDTH * ROW + COL ;
596                       FMAN(2) := ABS(BRD(FSQ(0))) ;
597                       IF FMAN(2) = EMPTY
598                       THEN BEGIN
599                         MESSAGE1 := "ATTEMPT TO MOVE AN EMPTY SQUARE" ;
600                         GO TO ERROR ;
601                       END ;
602                       IF NEXT(T)
603                       THEN BEGIN
604                         TEMPPC := FMAN(2) ;
605                         GETPC ;
606                         IF FMAN(2) ~= TEMPPC
607                         THEN BEGIN
608                           MESSAGE1 :=
609                             "SQUARE DOES NOT CONTAIN THAT PIECE" ;
610                           GO TO ERROR ;
611                         END ;
612                       END ;
613                     END ELSE GO TO ERROR ;
614                   END ELSE GO TO ERROR ;
615                 END ;
616               END ;
617             END ;
618           END ELSE IF (T = "0") OR (T = "O")
619                    THEN BEGIN
620                      WHILE NEXT(T)
621                      DO IF (T = "0") OR (T = "O")
622                         THEN REW(1) := REW(1) + 1 ;
623                      IF (REW(1) > 0) AND (REW(1) < 3)
624                      THEN REW(1) := IF (REW(1) = 2)
625                                     THEN QSIDE
626                                     ELSE KSIDE
627                      ELSE BEGIN
628                        MESSAGE1 := "INVALID CASTLING" ;
629                        GO TO ERROR ;
630                      END ;
```

```
631           END ELSE IF GETOK(FALSE)
632                    THEN BEGIN
633                      COMMENT try for simple pawn capture of the
634                        form ed ;
635                      TCOL := COL ;
636                      IF NEXT(T) AND GETOK(FALSE)
637                      THEN BEGIN
638                        OFFSET := TCOL - COL + 10*K ;
639                        FMAN(2) := PAWN ;
640                        FOR I := BRDWIDTH +COL STEP 10 UNTIL BKRSQ
641                        DO IF BRD(I) = K*PAWN
642                           THEN IF K*BRD(I+OFFSET) < 0
643                                THEN BEGIN
644                                  TSQ(0) := I + OFFSET ;
645                                  TMAN(2) := ABS(BRD(TSQ(0))) ;
646                                  FSQ(0) := I ;
647                        END ;
648                        IF NEXT(T) AND T ~= "P" OR TSQ(0) = 0
649                        THEN GOTO ERROR ;
650                      END ELSE GOTO ERROR ;
651           END ELSE GOTO ERROR ;
652           GO TO QUIT ;
653     END ;
654   ERROR:
655     NULLMOVE := TRUE ;
656   QUIT:
657   END OFALGEBRAIC ;
658
659   PROCEDURE DESCRIBEMOVE(INTEGER VALUE N ;
660   STRING(1) ARRAY MC(*)) ;
661   COMMENT internal move description is converted to chess notation ;
662   BEGIN
663     INTEGER M,M2,M4,M44,M5,M6,M3,KK, I, J ;
664     STRING(24) OUT ;
665     STRING(4) ARRAY CHK(0::5) ;
666
667     STRING(3) PROCEDURE CHESSBOARD(INTEGER VALUE SQ) ;
668     BEGIN
669       INTEGER ROW, COL ;
670       STRING(3) SQUARE ;
671       ROW := ROWS(SQ) ;
672       COL := COLS(SQ) ;
673       SQUARE(0|1) := IF COL < 4
674                      THEN "Q"
675                      ELSE IF COL > 5
676                           THEN "K"
677                           ELSE SPACE ;
678       SQUARE(1|1) := CHESSPIECE(COL) ;
679       SQUARE(2|1) := IF KK > 0
680                      THEN DIGIT(ROW)
681                      ELSE DIGIT(9-ROW) ;
682       SQUARE
683     END OFCHESSBOARD ;
684
685     M2  := ABS (REWARD(N)) ;
686     M := M2 DIV CHECKING ;
687     M2 := M2 REM CHECKING ;
688     IF (M2 = 2)
689     THEN M2 := 0 ;
690     CHK(0) := "    " ;
691     CHK(5) := "MATE" ;
692     CHK(2) := " CH." ;
693     CHK(4) := " DBL" ;
694     CHK(3) := " DIS" ;
695     CHK(1) := " STA" ;
696     J := 5 ;
697     IF (N = 0)
698     THEN M := 5
699     ELSE IF (ENGLISH OR ~COKO) AND (M2 = KSIDE OR M2 = QSIDE)
700          THEN OUT(0|5) := IF (M2 = QSIDE)
```

```
                        THEN "O-O-O"
                        ELSE "O-O  "
  ELSE BEGIN
    M3  := MOVEFROM(N) ;
    M4  := BRD (M3) ;
    M5  := MOVETO(N) ;
    M6 := BRD(M5) ;
    IF (ABS(M6) = ENPRIS)
    THEN M6 := NIL ;
    IF M4 = 0
    THEN BEGIN
      M4 := M6 ;
      M6 := -M2*(IF M4 > 0
                 THEN 1
                 ELSE -1) ;
    END ;
    IF ABS(M6) = PFTWO OR M4*M6 > 0
    THEN M6 := 0 ;
    KK := IF (M4 > 0)
          THEN 1
          ELSE IF M4 < 0
               THEN -1
               ELSE -K ;
    M44 := ABS(M4) ;
    OUT(0|1) := CHESSMAN(M44) ;
    IF ~ENGLISH
    THEN BEGIN
      J := IF OUT(0|1) = "P" OR COKO
           THEN 0
           ELSE 1 ;
      IF (PIECES(M4) > 1) AND (M44 ~= BISHOP) AND ((M44 ~= PAWN)
        OR (M2 = ENPRIS) OR (M6 ~= 0)) OR COKO
      THEN BEGIN
        OUT(J|2) := BOX(M3) ;
        OUT(J+2|1) := IF (M6 ~= 0) OR ((M2 = ENPRIS) AND (M44 = PAWN))
                      THEN "x"
                      ELSE "-" ;
        J := J + 3 ;
        IF COKO
        THEN J := J -1 ;
      END ELSE IF (M6 ~= 0) OR ((M2 = ENPRIS) AND (M44 = PAWN))
                THEN BEGIN
                  IF J = 0
                  THEN J := 1 ;
                  OUT(J|1) := "x" ;
                  J := J + 1 ;
      END ;
      OUT(J|2) := BOX(M5) ;
      J := J + 2 ;
    END ELSE
    BEGIN
      IF (PIECES(M4) > 1) AND (M44 ~= BISHOP) AND ((M44 ~= PAWN)
        OR (M2 > 0) AND (M2 < PFTWO)) OR COKO
      THEN BEGIN
        OUT(1|1) := "/" ;
        OUT(2|3) := CHESSBOARD(M3) ;
      END ELSE J := 1 ;
      IF (M2 = ENPRIS) AND (M44 = PAWN)
      THEN BEGIN
        OUT(J|3) := "xP/" ;
        OUT(J+3|3) := CHESSBOARD(M5-K*FILE) ;
        IF COKO
        THEN J := J + 4 ;
        OUT(J+2 | 3) := " ep" ;
        J := J+5 ;
      END ELSE
      BEGIN
        IF (M6 ~= 0)
        THEN BEGIN
          OUT(J|3) := "x /" ;
```

```
771           OUT(J+1|1) := CHESSMAN(ABS(M6)) ;
772           J := J + 2 ;
773         END ELSE OUT(J|1) := "-" ;
774         IF (M6 = 0) OR (PIECES(M6) > 1) OR COKO
775         THEN BEGIN
776           OUT(J+1|3) := CHESSBOARD(M5) ;
777           J := J + 4 ;
778         END ;
779       END ;
780     END ;
781     IF (M2 > PFTWO)
782     THEN BEGIN
783       OUT(J|2) := " =" ;
784       OUT(J+2|1) := CHESSMAN(M2-PROMOTE) ;
785       J := J + 3 ;
786     END ;
787   END ;
788   OUT(J|4) := CHK(M) ;
789   IF (M >1) AND (~ENGLISH OR COKO)
790   THEN OUT(J|4) := " +  " ;
791   IF ~ENGLISH
792   THEN IF M >= 4
793        THEN OUT(J+2|1) := IF M=4
794                           THEN "-"
795                           ELSE "+" ;
796   IF ENGLISH OR ~COKO
797   THEN J := J + 4 ;
798   FOR I := 0 UNTIL J
799   DO MC(I) := OUT(I|1) ;
800   FOR I := J UNTIL 23
801   DO MC(I) := SPACE ;
802 END OFDESCRIBEMOVE ;
803
804 @TITLE DUMPER"
805 PROCEDURE DUMPER(INTEGER VALUE X, Y) ;
806 BEGIN
807   INTEGER T ;
808   STRING(4) MARK ;
809   MARK := " #  " ;
810   T := X ;
811   IF (T > Y)
812   THEN X := Y ;
813   IF (T > Y)
814   THEN Y := T ;
815   IF (X < 1)
816   THEN X := 1 ;
817   IF (Y > 8)
818   THEN Y := 8 ;
819   I_W := 2 ;
820   FOR L := X*BRDWIDTH UNTIL (Y+1)*BRDWIDTH
821   DO IF (BRD(L) ~= EDGE)
822      THEN BEGIN
823        WRITE(L) ;
824        FOR J :=  1 STEP  1 UNTIL POS(L)
825        DO WRITEON(POSITION(L,J)) ;
826        WRITEON(MARK) ;
827        FOR J := POSITION(L,LIM) UNTIL LIM-1
828        DO WRITEON(POSITION(L,J)) ;
829        WRITEON(MARK) ;
830        FOR J := CON(-1,L) UNTIL CON(1,L)
831        DO WRITEON(CONTROL(L,J)) ;
832        WRITEON(MARK) ;
833        WRITEON(CON(0,L), CON(1,L)) ;
834   END ;
835   I_W := 4 ;
836 END OFDUMPER ;
837
838 PROCEDURE FORKING( INTEGER VALUE KSQ, SQ, PC, KK) ;
839 BEGIN
840   INTEGER DIR, SQT, PCT, VAL ;
```

```
841   IF PC ~= KNIGHT
842   THEN BEGIN
843     DIR := BOTV(EDGE, OFFSET(KSQ) -OFFSET(SQ)) ;
844     SQT := SQ ;
845     IF DIR ~= 0
846     THEN WHILE BRD(SQT) = 0
847     DO SQT := SQT + DIR ;
848     PCT := BRD(SQT) ;
849 COMMENT WRITE("FORK", KSQ, SQ, SQT, PCT, PC, KK, CON(KK,SQT));
850     IF PCT ~= EDGE AND PCT*KK > 0 AND (CON(KK,SQT) = 0 OR ABS(PCT)
851       > PC)
852     THEN FORKS(KK,SQ) := -PC*KK ;
853   END ELSE FOR J := 1 UNTIL 8
854            DO BEGIN
855              SQT := SQ + S(J) ;
856              PCT := BRD(SQT) ;
857 COMMENT WRITE("FORK", KSQ, SQ, SQT, PCT, PC, KK, CON(KK,SQT));
858              IF SQT ~= KSQ AND PCT ~= EDGE AND PCT*KK
859                > 0 AND (CON(KK,SQT) = 0 OR ABS(PCT) > PC)
860              THEN FORKS(KK,SQ) := -PC*KK ;
861   END ;
862 END OFFORKING ;
863
864 @TITLE,"GETMAN(STRING(1) ARRAY PCDESC(*) "
865 PROCEDURE GETMAN(STRING(1) ARRAY PCDESC(*) ;
866 INTEGER ARRAY MAN(*) ;
867 INTEGER VALUE L) ;
868 COMMENT determines internal code for chess man mentioned ;
869 BEGIN
870   FOR I := 0, 1, 2
871   DO BEGIN
872     MAN (I)  := 0 ;
873     FOR J  := PAWN UNTIL KING
874     DO IF (PCDESC (L) = CHESSMAN (J))
875        THEN BEGIN
876          MAN (I)  := J ;
877          GO TO NEXTI
878     END ;
879 NEXTI:
880     L := L + 1 ;
881   END FORI ;
882   IF (MAN (1) = 0)
883   THEN BEGIN
884     MAN (2)  := MAN (0) ;
885     MAN (0)  := 0 ;
886   END ELSE IF (MAN (2) = 0)
887            THEN BEGIN
888              MAN (2)  := MAN (1) ;
889              MAN (1)  := MAN (0) ;
890              MAN (0)  := 0 ;
891   END ;
892   IF (MAN(2) = 0)
893   THEN MESSAGE1 := "IMPROPER PIECE" ;
894 END OFGETMAN ;
895
896 @TITLE,"GETSQ(STRING(1) ARRAY SQDESC(*) "
897 PROCEDURE GETSQ(STRING(1) ARRAY SQDESC(*) ;
898 INTEGER ARRAY SQ(*) ;
899 INTEGER VALUE D) ;
900 COMMENT determines internal number for chess square ;
901 BEGIN
902   INTEGER FIRST, NEXT, ROW,COL ;
903   STRING(1) SIDE ;
904   FIRST := D + 2 ;
905   NEXT := D + 1 ;
906   SIDE := SQDESC (D) ;
907   SQ(0) := NINES ;
908   SQ(1) := -NINES ;
909   FOR J := 1,2,3
910   DO BEGIN
```

```
911      FOR I  := 1 UNTIL 8
912      DO IF (SQDESC (FIRST) = DIGIT (I))
913         THEN BEGIN
914           ROW := IF (K < 0)
915                  THEN 9 - I
916                  ELSE I ;
917           GO TO AA
918      END ;
919      FIRST := NEXT ;
920      NEXT  := D ;
921      SIDE  := "0" ;
922    END OFWHILELOOP ;
923    GO TO QUIT ;
924  AA:
925    IF (SIDE = "K")
926    THEN COL := IF SQDESC(NEXT) = "R"
927                THEN 8
928                ELSE IF SQDESC(NEXT) = "B"
929                     THEN 6
930                     ELSE IF SQDESC(NEXT) = "N"
931                          THEN 7
932                          ELSE IF SQDESC(NEXT) = "K"
933                               THEN 5
934                               ELSE NINES
935                               ELSE IF (SIDE = "Q")
936                                    THEN COL := IF SQDESC(NEXT) = "N"
937                         THEN 2
938                         ELSE IF SQDESC(NEXT) = "B"
939                              THEN 3
940                              ELSE IF SQDESC(NEXT) = "R"
941                                   THEN 1
942                                   ELSE IF (SQDESC(NEXT) = "Q")
943                     THEN 4
944                     ELSE NINES
945    ELSE BEGIN
946      IF (SIDE ~= "0")
947      THEN GO TO QUIT
948      ELSE IF (SQDESC(NEXT) = "K") OR (SQDESC(NEXT) = "Q")
949           THEN SIDE := SQDESC(NEXT)
950      ELSE BEGIN
951        SQ(1) := NINES ;
952        SIDE := "Q" ;
953      END ;
954      GO TO AA ;
955    END ;
956    IF (COL < BRDWIDTH)
957    THEN BEGIN
958      SQ(0) := BRDWIDTH*ROW + COL ;
959      IF (SQ(1) = NINES)
960      THEN SQ(1) := (ROW+1) * BRDWIDTH - COL - 1 ;
961    END ;
962  QUIT:
963    IF (SQ(0) = NINES)
964    THEN MESSAGE1 :=  "IMPROPER SQ" ;
965  END OFGETSQ ;
966
967  @TITLE,"PRINTLINE"
968  PROCEDURE PRINTLINE(INTEGER VALUE N,PLY) ;
969  BEGIN
970    INTEGER L,  SCR ;
971    DESCRIBEMOVE(N,MC) ;
972    IOCONTROL(2) ;
973    FOR I := 1 UNTIL PLY
974    DO WRITEON("  ") ;
975    L := (8*N -(IF (N > 0)
976                THEN 4
977                ELSE -4)) ;
978     COMMENT 4*(2*N -K) ;
979    SCR := SCORE(N) ;
980    IF DUMPP AND SCR ~= BAD AND SCR ~= NOSCR
```

```
   THEN BEGIN
     I_W := 3 ;
     WRITEON(N) ;
     FOR I := 0,4,5
     DO WRITEON(STORE(L+I)) ;
     FOR I := 6,7
     DO WRITEON(STORE(L+I)) ;
     I_W := 5 ;
     FOR I := 1,2,3
     DO WRITEON(STORE(I+L)) ;
   END ;
   WRITEON(IF N = TRY(TRY(0))
           THEN "#"
           ELSE " ") ;
   FOR I := 0 UNTIL 10
   DO WRITEON(MC(I)) ;
   I_W := 3 ;
   WRITEON(REWARD(N)) ;
   I_W := 4 ;
   WRITEON(SCR) ;
END OFPRINTLINE ;

INTEGER PROCEDURE CONVERTS( STRING(1) ARRAY U(*) ;
LOGICAL RESULT ERR) ;
BEGIN
   INTEGER CCOUNT, NUM, TEMP ;
   LOGICAL NEGATIVE ;
   NEGATIVE := ERR := FALSE ;
   NUM := CCOUNT := 0 ;
   WHILE U(CCOUNT) = " "
   DO IF CCOUNT < CHARCOUNT
      THEN CCOUNT := CCOUNT+1
      ELSE GO TO ERRTERM ;
   IF U(CCOUNT) = "-"
   THEN BEGIN
     NEGATIVE := TRUE ;
     CCOUNT := CCOUNT + 1 ;
   END ELSE IF U(CCOUNT) = "+"
            THEN  CCOUNT := CCOUNT+1 ;
   WHILE U(CCOUNT) >= "0" AND U(CCOUNT) <= "9" AND NUM <= 214748363
   DO BEGIN
     TEMP := DECODE(U(CCOUNT)) - 240 ;
     NUM := (NUM * 10) + TEMP ;
     CCOUNT := CCOUNT+1 ;
   END ;
   IF U(CCOUNT) ~= " " OR U(CCOUNT-1) = "+" OR U(CCOUNT-1) = "-"
   THEN GO TO ERRTERM
   ELSE BEGIN
     FOR I := 0 UNTIL CCOUNT-1
     DO U(I) := " " ;
     GO TO TERM ;
   END ;
ERRTERM:
   ERR := TRUE ;
   NUM := 0 ;
TERM:
   IF NEGATIVE
   THEN -NUM
   ELSE NUM
END OFCONVERTS ;

PROCEDURE PRINTSHORTBRD (INTEGER VALUE S) ;
BEGIN
   INTEGER PC, LEFT, RIGHT ;
    COMMENT print boarder as well for now ;
   LEFT := IF (S = 1)
           THEN 0
           ELSE 9 ;
   RIGHT := IF (S = 1)
            THEN 9
```

```
1051            ELSE 0 ;
1052    FOR I := LEFT STEP S UNTIL RIGHT
1053    DO BEGIN
1054      IOCONTROL(2) ;
1055      FOR J := RIGHT STEP -S UNTIL LEFT
1056      DO BEGIN
1057        PC := BRD(BRDWIDTH*I+J) ;
1058        IF PC = EMPTY
1059        THEN WRITEON(IF (I+J) REM 2 = 0
1060                     THEN " ::"
1061                     ELSE "   ")
1062                     ELSE IF DEBUG OR (PC ~= EDGE)
1063                          THEN IF ABS(PC) = ENPRIS
1064                               THEN WRITEON(" EP")
1065                               ELSE WRITEON(MAN(PC)) ;
1066      END ;
1067    END ;
1068    I_W := 2 ;
1069    S_W := 0 ;
1070    WRITE(DATE(0|8), "AP1.0 ", COLOR(K),"'S MOVE ", MOVECOUNT,". ") ;
1071    I_W := 4 ;
1072    S_W := 1 ;
1073  END OFPRINTSHORTBRD ;
1074
1075  @TITLE,"PUNCHBRD (LOGICAL VALUE FLAG) "
1076  PROCEDURE PUNCHBRD (LOGICAL VALUE FLAG ;
1077  STRING(140) VALUE RESULT HISTORY) ;
1078  BEGIN
1079    INTEGER L, M, T ;
1080    M := -1 ;
1081    HISTORY(0|78) := " " ;
1082    FOR KK := 1, -1
1083    DO BEGIN
1084      FOR I := 1 UNTIL 8
1085      DO BEGIN
1086        T := 0 ;
1087        L := I * BRDWIDTH ;
1088        FOR J := 1 UNTIL 8
1089        DO BEGIN
1090          L := L+1 ;
1091          IF (KK*BRD(L) <= 0)
1092          THEN T := T + 1
1093          ELSE BEGIN
1094            M := M+2 ;
1095            IF (T ~= 0)
1096            THEN HISTORY(M-1|1) := DIGIT(T)
1097            ELSE M := M-1 ;
1098            T := 0 ;
1099            HISTORY(M|1) := CHESSMAN(ABS(BRD(L))) ;
1100          END ;
1101        END FORJ ;
1102        IF (T ~= 0)
1103        THEN BEGIN
1104          M := M+1 ;
1105          HISTORY(M|1) := DIGIT(T) ;
1106        END ;
1107      END FORI ;
1108      M := M+1 ;
1109      HISTORY(M|1) := IF K=1
1110                      THEN "+"
1111                      ELSE "-" ;
1112    END ;
1113    FOR I := -3 UNTIL 3
1114    DO HISTORY(I+81|1) := IF CASTLE(I)
1115                          THEN "T"
1116                          ELSE "F" ;
1117    M := MOVECOUNT REM 100 ;
1118    HISTORY(111|1) := IF (MOVECOUNT < 10)
1119                      THEN " "
1120                      ELSE CODE((M DIV 10) + 240) ;
```

```
1121   HISTORY(112|1) := CODE((M REM 10) + 240) ;
1122 END OFPUNCHBRD ;
1123
1124 @TITLE,"READU"
1125 LOGICAL PROCEDURE READU ;
1126 COMMENT reads teletype ;
1127 BEGIN
1128   LOGICAL TRAC ;
1129   ENDFILE := EXCEPTION(FALSE, 1, 1, FALSE, "UNEXPECTED EOF") ;
1130 STARTS:
1131   TRAC := FALSE ;
1132   IOCONTROL(2) ;
1133    COMMENT read tty buffer and place characters into
1134    array u(s::f). LAST points to the end of the actual input ;
1135   CHARCOUNT := -1 ;
1136   WHILE (CHARCOUNT < 0)
1137   DO BEGIN
1138     CHARCOUNT := MAXCHAR-1 ;
1139     READCARD(BUFFER) ;
1140     IF XCPNOTED(ENDFILE)
1141     THEN GO TO STARTS ;
1142 COMMENT strip trailing blanks;
1143     WHILE (CHARCOUNT >= 0) AND (BUFFER(CHARCOUNT|1) = BLANK)
1144     DO CHARCOUNT := CHARCOUNT - 1 ;
1145   IF ECHO THEN BEGIN
1146      FOR I := 0 UNTIL CHARCOUNT
1147      DO WRITEON(BUFFER(I|1)) ;
1148    IOCONTROL(2);
1149 END;
1150     IF (BUFFER(0|1) = ".")
1151     THEN CHARCOUNT := -1 ;
1152     FOR I := 0 UNTIL CHARCOUNT+2
1153     DO U(I) := BUFFER(I|1) ;
1154   END ;
1155   IF (U (0) = "?") OR (U(0) = "*")
1156   THEN TRAC := TRUE
1157   ELSE IF (U(0) = "S") AND (U(1) = "T")
1158        THEN GO TO STOP ;
1159    COMMENT    IF COKO THEN GET(5) ;
1160    COMMENT restore input unit ;
1161   TRAC
1162 END OFREADU ;
1163
1164 @TITLE,"SCORER(STRING(1) VALUE S, S1, S2) "
1165 PROCEDURE SCORER(STRING(1) VALUE S, S1, S2) ;
1166 BEGIN
1167   INTEGER J, F, T ;
1168   F := DECODE(S1) - 240 ;
1169   T := DECODE(S2) - 240 ;
1170   IF (F < 0)
1171   THEN F := 0 ;
1172   IF (T < 0)
1173   THEN T := 0 ;
1174   J := IF (S = "-")
1175        THEN -1
1176        ELSE 1 ;
1177   F := J*(F*6)+J ;
1178   T := J*(T+1)*6 ;
1179   IF ABS(T) = 60 OR (ABS(T) > ABS(NUMBER(J)))
1180   THEN T := NUMBER(J) ;
1181   FOR N := F STEP J UNTIL T
1182   DO PRINTLINE(USE(ABS(N)),0) ;
1183 END OFSCORER ;
1184
1185 PROCEDURE SETBOARD (STRING(140) VALUE BUFFER) ;
1186 COMMENT fills nsq(*) to describe board position read in ;
1187 BEGIN
1188   INTEGER I, J,  PC, KP, TT, SBK ;
1189   STRING(1) T ;
1190   BRD(HOLE) := EMPTY ;
```

```
1191    FOR I := WQRSQ UNTIL BKRSQ
1192    DO BRD(I) := EMPTY ;
1193    FOR I := 10 STEP BRDWIDTH UNTIL 80
1194    DO BRD(I) := BRD(I+9) := EDGE ;
1195    FOR I := -10 UNTIL 9
1196    DO BRD(I) := BRD(I+100) := EDGE ;
1197    I := WQRSQ - RANK ;
1198    J  := 0 ;
1199    SBK := 1 ;
1200    WHILE (BUFFER(J|1) = SPACE)
1201    DO J := J+1 ;
1202  NEXTSYMBOL:
1203    T := BUFFER(J|1) ;
1204    J := J + 1 ;
1205    IF (COLS(I) = 8)
1206    THEN I := I + 2 ;
1207    TT := DECODE(T) - 240 ;
1208     COMMENT is TT a digit ;
1209    IF TT > 8 OR TT < 1
1210    THEN BEGIN
1211      FOR KP := PAWN,KNIGHT,BISHOP,ROOK,QUEEN,KING,ENPRIS
1212      DO BEGIN
1213        PC := KP ;
1214        IF (T = CHESSMAN (PC))
1215        THEN GO TO PIECE ;
1216      END ;
1217      SBK := -SBK ;
1218      I := WQRSQ - RANK ;
1219      IF (T ~= "+" AND T ~= "-" AND T ~= ".")
1220      THEN GOTO ERROR ;
1221      IF (SBK = -1)
1222      THEN BEGIN
1223        K := IF T = "-"
1224             THEN -1
1225             ELSE 1 ;
1226        GO TO NEXTSYMBOL ;
1227      END ELSE GO TO QUIT ;
1228  PIECE:
1229      I := I+RANK ;
1230      IF BRD(I) ~= EMPTY
1231      THEN GOTO ERROR ;
1232      BRD(I) := SBK*PC ;
1233    END ELSE IF COLS(I) + TT > 8
1234             THEN GOTO ERROR
1235             ELSE I := I + TT ;
1236    GO TO NEXTSYMBOL ;
1237  ERROR:
1238    WRITE("ERROR: SETBOARD") ;
1239    PRINTSHORTBRD(-1) ;
1240  QUIT:
1241    IF (BUFFER(111|2) ~= "  ")
1242    THEN BEGIN
1243      T := BUFFER(111|1) ;
1244      MOVECOUNT := IF (T = " ")
1245                   THEN 0
1246                   ELSE DECODE(T)-240 ;
1247      IF MOVECOUNT = 0 AND T ~= " "
1248      THEN MOVECOUNT := 10 ;
1249      T := BUFFER(112|1) ;
1250      MOVECOUNT := IF (T = " ")
1251                   THEN 1
1252                   ELSE MOVECOUNT*10+DECODE(T)-240 ;
1253    END ;
1254    IF (MOVECOUNT <= 0)
1255    THEN MOVECOUNT := 1 ;
1256    IF (BUFFER(81|1) = " ")
1257    THEN BEGIN
1258      FOR SBK := -1, 1
1259      DO BEGIN
1260        IF (BRD(50-35*SBK) = KING * SBK)
```

```
1261        THEN BEGIN
1262          CASTLE(SBK) := IF (BRD(KRSQ(SBK)) = ROOK * SBK)
1263                         THEN TRUE
1264                         ELSE FALSE ;
1265          CASTLE(TWO(SBK)) := IF (BRD(QRSQ(SBK)) = ROOK * SBK)
1266                              THEN TRUE
1267                              ELSE FALSE ;
1268        END ELSE CASTLE(SBK) := CASTLE(TWO(SBK)) := FALSE ;
1269        CASTLE(THREE(SBK)) := ~ (CASTLE(TWO(SBK)) OR CASTLE(SBK)) ;
1270      END ;
1271      CASTLE(0) := IF CASTLE(-3) AND CASTLE(3)
1272                   THEN FALSE
1273                   ELSE IF CASTLE(-3) OR CASTLE(3)
1274                        THEN TRUE
1275                        ELSE FALSE ;
1276    END ELSE FOR I := -3 UNTIL 3
1277    DO CASTLE(I) := IF (BUFFER(I+81|1) = "T")
1278                    THEN TRUE
1279                    ELSE FALSE ;
1280  END OFSETBOARD ;
1281  @TITLE,"STARTTHISPROBLEM "
1282  LOGICAL PROCEDURE STARTTHISPROBLEM (LOGICAL VALUE SEEK) ;
1283  COMMENT sets parameters to start a chess problem ;
1284  BEGIN
1285    INTEGER J, PC ;
1286    LOGICAL NEEDKING ;
1287    MAKEMV := FALSE ;
1288    NEEDKING := FALSE ;
1289    KINGSQ (1) := KINGSQ (-1) := DIFFVAL := MEN(1) := MEN(-1) := 0 ;
1290    FOR I := -8 UNTIL 8
1291    DO PIECES (I) := 0 ;
1292    FOR SQ :=  WQRSQ UNTIL BKRSQ
1293    DO BEGIN
1294      PC := BRD(SQ) ;
1295      IF (PC ~= EDGE) AND (PC ~= EMPTY)
1296      THEN IF (ABS(PC) <= KING)
1297           THEN BEGIN
1298             J := IF (PC > 0)
1299                  THEN 1
1300                  ELSE -1 ;
1301             PIECES(0) := PIECES(0) + 1 ;
1302             MEN(J) := MEN(J) + 1 ;
1303             DIFFVAL := DIFFVAL + J*VALUES(ABS(PC)) ;
1304             PIECES(PC) := PIECES(PC) + 1 ;
1305             IF (ABS(PC) = KING)
1306             THEN CLEARPAWN(J) := KINGSQ(J)  := SQ ;
1307      END ;
1308      POS(SQ) := POSITION(SQ,0) :=  CON(-1,SQ) := CON(0,SQ) := CON(1,SQ)
1309        := CONTROL(SQ,0) := 0 ;
1310      SEC(-1,SQ) := SEC(0,SQ) := SEC(1,SQ) := SECRET(SQ,0) := 0 ;
1311      POSITION(SQ,LIM) := LIM ;
1312      CAPTURE_CH(SQ) := FALSE ;
1313    END FORSQ ;
1314    IF PIECES(KING) = 1 AND PIECES(-KING) = 1
1315    THEN BEGIN
1316      SQUARE(ELIMIT-1) := WQRSQ ;
1317      FOR SQ := WQRSQ UNTIL BKRSQ
1318      DO BEGIN
1319        PC := BRD(SQ) ;
1320        IF (PC ~= EMPTY) AND (PC ~= EDGE)
1321        THEN BEGIN
1322          LOSS(ELIMIT) := ELIMIT ;
1323          REMAKE(SQ,SQ) ;
1324          IF SEEK AND ABS(PC) = PAWN
1325          THEN BEGIN
1326            J := IF PC > 0
1327                 THEN 1
1328                 ELSE -1 ;
1329                 COMMENT note most advanced passed pawn;
1330            IF PASSEDPAWN(SQ, J) > 0
```

```
1331            THEN IF J > 0 OR CLEARPAWN(J) = KINGSQ(J)
1332               OR ROWS(CLEARPAWN(J)) = ROWS(SQ)
1333                 THEN CLEARPAWN(J) := SQ ;
1334          END ;
1335        END ;
1336      END ;
1337      REMAKE(KINGSQ(1),KINGSQ(1)) ;
1338      REMAKE(KINGSQ(-1),KINGSQ(-1)) ;
1339      KINGCONTROL(1) ;
1340      KINGCONTROL(-1) ;
1341      IF LEVEL = 0 AND CON(K,KINGSQ(-K)) ~= 0
1342      THEN NEEDKING := TRUE ;
1343      ENPRISE(-K) ;
1344      J := ELIMIT ;
1345      TWOVAL := 0 ;
1346      FOR I  := SQUARE(0) STEP -1 UNTIL 1
1347      DO BEGIN
1348        J := J-1 ;
1349        SQUARE(J) := SQUARE(I) ;
1350        LOSS(J) := LOSS(I) ;
1351        IF (LOSS(J) > TWOVAL)
1352        THEN TWOVAL := LOSS(J) ;
1353      END ;
1354      LOSS(ELIMIT) := SQUARE(ELIMIT) := J ;
1355      IF SEEK
1356      THEN ENPRISE(K) ;
1357    END ELSE NEEDKING := TRUE ;
1358    IF NEEDKING
1359    THEN WRITE("KING PROBLEMS") ;
1360    ~NEEDKING
1361  END OFSTARTTHISPROBLEM ;
1362
1363  @TITLE,"CHECK(INTEGER VALUE SQ, KK "
1364  LOGICAL PROCEDURE CHECK(INTEGER VALUE SQ, KK ;
1365  INTEGER RESULT NT) ;
1366  COMMENT  how many times (nt) is square sq "attacked" by pieces
1367  of colour -kk? ;
1368  BEGIN
1369    INTEGER NN, HD, HN,  SQU, VAL, PCU, L, SQ3, PC, J, LL, SQP, K_FILE,
1370      I ;
1371    LOGICAL NOTKING ;
1372    NN := HN := 0 ;
1373    HD := 10 ;
1374    K_FILE := FILES(K) ;
1375    KINGUSED := FALSE ;
1376    PC := ABS(BRD(SQ)) ;
1377    NOTKING := IF (PC = KING)
1378               THEN FALSE
1379               ELSE TRUE ;
1380    LL := CON(-KK,SQ) ;
1381    IF (PC = ENPRIS)
1382    THEN BEGIN
1383      COMMENT enpassant pawn may block attack/defence along the file ;
1384      SQP := SQ + K_FILE ;
1385      FOR L := CON(-KK,SQP) STEP KK UNTIL -KK
1386      DO BEGIN
1387        SQU := CONTROL(SQP,L) ;
1388        PCU := ABS(BRD(SQU)) ;
1389        IF (PCU = ROOK) OR (PCU = QUEEN)
1390        THEN IF (BOTV(EDGE,OFFSET(SQP)-OFFSET(SQU)) = K_FILE)
1391             THEN BEGIN
1392               LL := LL -KK ;
1393               CONTROL(SQ,LL) := SQU ;
1394        END ;
1395      END ;
1396    END ;
1397    FOR L := LL STEP KK UNTIL -KK
1398    DO BEGIN
1399      SQU := CONTROL(SQ,L) ;
1400      PCU := ABS(BRD(SQU)) ;
```

```
1401      IF (PCU < PAWN) OR (PCU > KING)
1402      THEN BEGIN
1403        WRITE("CHECK",SQ,SQU,PCU,L,K,LEVEL) ;
1404        GO TO STARTS ;
1405      END ;
1406      I := NN ;
1407      WHILE I > 0 AND PCU < ABS(BRD(SQR(I)))
1408      DO BEGIN
1409        SQR(I+1) := SQR(I) ;
1410        I := I -1 ;
1411      END ;
1412      SQR(I+1) := SQU ;
1413      NN := NN + 1 ;
1414      IF (PCU = KING)
1415      THEN KINGUSED := TRUE ;
1416      IF (PCU ~= KNIGHT)
1417      THEN IF (NOTKING)
1418           THEN WHILE HIDDEN(SQ, SQU, SQ3, 0, FALSE)
1419                DO BEGIN
1420                  COMMENT hidden attack or defence ;
1421                  IF KK*BRD(SQ3) > 0
1422                  THEN J := HD := HD -1
1423                  ELSE IF KINGUSED
1424                       THEN J := HN -1
1425                       ELSE J := HN := HN -1 ;
1426           COMMENT can have hidden defender "through" King ;
1427                  SQU := SQR(J) := SQ3 ;
1428      END ;
1429    END ;
1430    NT := NN + ABS(HN) ;
1431    SQR(10) := HD ;
1432    L := IF KINGUSED
1433         THEN NN-1
1434         ELSE NN ;
1435    FOR I := -1 STEP -1 UNTIL HN
1436    DO BEGIN
1437      COMMENT include hidden pieces. ;
1438      L := L +1 ;
1439      SQR(L) := SQR(I) ;
1440    END ;
1441     COMMENT   place king at end if used. ;
1442    IF (KINGUSED)
1443    THEN SQR(NT) := KINGSQ(-KK) ;
1444    POWER(0) := 0 ;
1445    IF DEBUG AND HN ~= 0
1446    THEN WRITE("check", NT) ;
1447    IF (NN > 0)
1448    THEN BEGIN
1449      FOR L := 1 UNTIL NT
1450      DO BEGIN
1451        SQU := SQR(L) ;
1452        SECRET(SQ,-KK*L) := SQU ;
1453        PCU := ABS(BRD(SQU)) ;
1454        POWER(L) := POWER(L-1) + VALUES(PCU) + (IF PCU = PAWN AND
1455          RANKS(ROWS(SQU)) = -KK
1456                        THEN PAWNVAL(ROWS(SQU))
1457                        ELSE 0) ;
1458        IF DEBUG AND HN ~= 0
1459        THEN WRITEON(POWER(L)) ;
1460      END ;
1461      MINPIECE := POWER(1) ;
1462      XRAY(SQ,-KK,NN,NT) ;
1463      MAXPIECE := IF KINGUSED
1464                  THEN VALUES(KING)
1465                  ELSE POWER(NN) - POWER(NN-1) ;
1466    END ;
1467    IF KINGUSED AND NN ~= NT
1468    THEN BEGIN
1469      SECRET(SQ, -KK*NN) := SQR(NT) ;
1470      SECRET(SQ, -KK*NT) := SQR(NN) ;
```

```
1471   END ;
1472   SEC(-KK,SQ) := -KK*NT ;
1473   POWER(0) := NN ;
1474   SQR(0) := NT ;
1475   IF (NN > 0)
1476   THEN TRUE
1477   ELSE FALSE
1478 END OFCHECK ;
1479
1480 @TITLE,"SACRIFICE(INTEGER VALUE SQ "
1481 LOGICAL PROCEDURE SACRIFICE(INTEGER VALUE SQ ;
1482 LOGICAL VALUE PRINT) ;
1483 COMMENT  determines whether the piece on square sq is being attacked ;
1484 BEGIN
1485   INTEGER MAXATTACK, MINATTACK, IA, ID, LOST, COST, PC, HID, KK,IX,
1486     I, L, TMP, SQ3, PC3, HID_DEF, SQA, SQ1, PC1, PCA, DEFN, KSQ;
1487   INTEGER ARRAY ATTACKERS(0::10) ;
1488   LOGICAL SACRIFICE,  DEFENDED, PROMCAPTURE, CHECKCAPTURE, T,WITHCHECK;
1489   SACRIFICE := FALSE ;
1490   COST := -NINES ;
1491   IX := ATTACKERS(0) := ATTASQRS(0) := ID := IA := 0 ;
1492   SEC(-1,SQ) := CON(-1,SQ); SEC(1,SQ) := CON(1,SQ);
1493   PC := BRD(SQ) ;
1494   KK := IF PC > 0
1495         THEN 1
1496         ELSE -1 ;
1497        IX := 0;
1498   IF  CHECK(SQ,KK,IA)
1499   THEN IF ~KINGUSED OR IA ~= 1 OR CON(KK,SQ) = 0
1500        THEN BEGIN
1501          PC := ABS(PC) ;
1502          MAXATTACK := MAXPIECE ;
1503          LOST := VALUES(PC) ;
1504          IF (PC = PAWN)
1505          THEN BEGIN
1506            IF (RANKS(ROWS(SQ)) = KK)
1507            THEN LOST := LOST + PAWNVAL(ROWS(SQ)) ;
1508            IF ENDGAME OR LOST >= 9
1509            THEN LOST := LOST + PASSEDPAWN(SQ,KK) ;
1510            TMP := 0 ;
1511            IF ABS(BRD(SECRET(SQ,-KK))) = PAWN AND CON(KK,SQ) = 0
1512            THEN TMP := PASSEDPAWN(SQ,-KK)
1513            ELSE BEGIN
1514              BRD(SQ) := 0 ;
1515              FOR L := -1, 0, 1
1516              DO BEGIN
1517                I := SQ + FILES(KK) + L ;
1518                WHILE (ABS(BRD(I)) ~= PAWN AND BRD(I) ~= EDGE)
1519                DO I := I + FILES(KK) ;
1520                TMP := TMP + (IF BRD(I) = -KK*PAWN
1521                              THEN PASSEDPAWN(I,-KK)
1522                              ELSE 0) ;
1523                IF TMP > 0 AND L ~= 0 AND ABS(ROWS(SQ)-ROWS(I))
1524                  = 1 AND KK*SEC(0,I-FILES(KK)) > 0
1525                THEN TMP := 0 ;
1526              END ;
1527              BRD(SQ) := KK*PAWN ;
1528            END ;
1529            IF TMP > 1 THEN TMP := TMP -1;
1530            LOST := LOST + TMP ;
1531          END ;
1532          COST := LOST ;
1533          PROMCAPTURE := IF (MINPIECE < VALUES(KNIGHT)) AND (PAWNS(SQ)
1534            = 0)
1535                         THEN TRUE
1536                         ELSE FALSE ;
1537          IF PROMCAPTURE
1538          THEN COST := COST +VALUES(ROOK) ;
1539          FOR I := 0 UNTIL IA
1540          DO BEGIN
```

```
1541            ATTASQRS(I) := SQR(I) ;
1542            ATTACKERS(I) := POWER(I) - LOST ;
1543          END ;
1544          ATTACKERS(0) := POWER(0) ;
1545          MINATTACK := POWER(1) ;
1546          HID_DEF := SQR(10) ;
1547          FOR I := SQR(10) UNTIL 10
1548          DO ATTASQRS(I) := SQR(I) ;
1549          CHECKCAPTURE := WEAKPIN < 0 ;
1550          HID := MINATTACK -MINPIECE ;
1551          DEFENDED := IF (CON(KK,SQ) = 0) AND (ATTASQRS(10) = 10)
1552                      THEN FALSE
1553                      ELSE TRUE ;
1554          IX := 1;
1555          IF ~DEFENDED OR ~CHECK(SQ, -KK, ID) AND ID > 10-ATTASQRS(10)
1556          THEN POWER(0) := SQR(0) := 0
1557          ELSE BEGIN
1558            COMMENT the following is an attempt to estimate hidden
1559              attacks ;
1560            IF DEBUG
1561            THEN IF (HID_DEF < 10 OR SQR(10) < 10)
1562                 THEN WRITE("SACR. PROB.", SQ, KK, IA, ID, SQR(SQR(10)),
1563                   SQR(10), ATTASQRS(HID_DEF)) ;
1564            IF HID_DEF = 9 AND ID = 0 AND ATTASQRS(0) = 1
1565            THEN IF VALUES(ABS(BRD(ATTASQRS(HID_DEF))))
1566              > VALUES(ABS(BRD(ATTASQRS(1)))) + DELTA
1567                 THEN HID_DEF := 10 ;
1568            IF PROMCAPTURE
1569            THEN COST := COST -VALUES(ROOK) ;
1570            L := 10 -HID_DEF ;
1571            IF L > 0
1572            THEN BEGIN
1573              L := CON(KK,SQ) ;
1574              FOR I := HID_DEF UNTIL 9
1575              DO BEGIN
1576                L := L + KK ;
1577                CONTROL(SQ,L) := ATTASQRS(I) ;
1578                ID := ID + 1 ;
1579                SQR(ID) := ATTASQRS(I) ;
1580                SECRET(SQ,KK*ID) := SQR(ID) ;
1581                POWER(ID) := POWER(ID-1) + VALUES(ABS(BRD(SQR(ID)))) ;
1582              END ;
1583            END ;
1584            L := 10 -SQR(10) ;
1585            IF L > 0
1586            THEN BEGIN
1587              L := CON(-KK,SQ) ;
1588              FOR I := SQR(10) UNTIL 9
1589              DO BEGIN
1590                L := L - KK ;
1591                CONTROL(SQ,L) := SQR(I) ;
1592                IA := IA + 1 ;
1593                ATTASQRS(IA) := SQR(I) ;
1594                SECRET(SQ,-KK*IA) := SQR(I) ;
1595                ATTACKERS(IA) := ATTACKERS(IA-1) +VALUES(ABS(BRD(SQR(I)
1596                  ))) ;
1597              END ;
1598            END ;
1599            SEC(KK,SQ) := KK*(ID) ;
1600            SEC(-KK,SQ) := -KK*(IA) ;
1601  COMMENT assume attacker makes last move;
1602            IF (IA > ID)
1603            THEN
1604            BEGIN
1605            COMMENT  e.g., qbn/b/nr, kn/p/b ;
1606              IF KINGUSED
1607              THEN ID := ID -1 ;
1608              COST := LOST;
1609          IX := 1;
1610              FOR I := 1 UNTIL ID
```

```
1611              DO BEGIN
1612                IX := I;
1613                COMMENT kbn/q/nq, kq/r/k, qbn/n/pq ;
1614                TMP := POWER(I) - ATTACKERS(I);
1615                IF TMP < -DELTA
1616                THEN BEGIN
1617                  COMMENT defender makes last move;
1618                  COMMENT e.g., nnp/p/pp, rr/b/p ;
1619                  IX := 0;
1620                  IF I = 1
1621                  THEN COST := -ATTACKERS(1)
1622                  ELSE IF I = 2
1623                       THEN COST := COST - POWER(I-1)
1624                       ELSE COST := COST - POWER(I-1) + POWER(I-2);
1625                  GOTO SWAP;
1626                END ELSE IF COST < TMP -DELTA
1627                         THEN GOTO SWAP
1628                         ELSE COST := TMP;
1629              END;
1630              IX := ID +1;
1631              GO TO SWAP ;
1632            END ELSE
1633            BEGIN
1634 COMMENT assume defender makes last move (no forkloss);
1635              IX := 0;
1636              COST := - ATTACKERS(1) ;
1637              IF (ID = 1) AND (WEAKPIN > 0)
1638              THEN COST := COST +(IF WEAKPIN > MINATTACK
1639                                  THEN MINATTACK
1640                                  ELSE WEAKPIN) ;
1641              IF (ABS(BRD(ATTASQRS(IA))) = KING)
1642              THEN BEGIN
1643                COMMENT K/Q/K, K/R/QK, KB/B/NK ;
1644                IA := IA -1 ;
1645                IF IA = 0
1646                THEN COST := -NINES ;
1647              END ;
1648    COMMENT 2 <=  IA <=  ID ;
1649              IF IA <= 1
1650              THEN BEGIN
1651                COMMENT N/P/R, P/P/N, N/P/PB, P/N/PB ;
1652                GOTO SWAP;
1653              END;
1654              FOR I := 2 UNTIL IA
1655              DO BEGIN
1656                COMMENT RB/P/NR, QR/N/PQ ;
1657                COMMENT QR/Q/BK ;
1658                TMP := POWER(I-1) - ATTACKERS(I);
1659                IF TMP < -DELTA
1660                THEN GOTO SWAP
1661                ELSE IF COST <= TMP + DELTA
1662                     THEN COST := TMP
1663                     ELSE BEGIN
1664                       COMMENT attacker makes last move;
1665                       COMMENT PP/P/RK, QN/N/RK ;
1666                       IX := I -1;
1667                       IF I = 2
1668                       THEN COST := LOST;
1669                       GOTO SWAP;
1670                     END;
1671              END;
1672              IF COST > LOST
1673              THEN   BEGIN
1674                COST := LOST;
1675                IX := 1;
1676              END;
1677            END ;
1678 SWAP:
1679            ATTASQRS(0) := IA ;
1680            SQR(0) := ID ;
```

```
1681            IF KINGUSED AND IA > ID
1682            THEN COST := COST + 1 ;
1683 COMMENT CREDIT for capture in vicinity of King ;
1684          END ;
1685          IF ~DEFENDED AND (ATTACKERS(0) = 1)
1686          THEN COST := COST -HID ;
1687          CAPTURE_CH(SQ) := FALSE ;
1688          TMP := CHECK_S(KK,SQ) ;
1689          IF TMP ~= 0 AND PC ~= KING
1690          THEN FOR I := CON(-KK,SQ) STEP KK UNTIL -KK
1691               DO BEGIN
1692                 PC := BRD(CONTROL(SQ,I)) ;
1693                 IF PC*TMP > 0
1694                 THEN IF CHEC(ABS(TMP), ABS(PC))
1695                      THEN BEGIN
1696                        IF DEBUG
1697                        THEN WRITE("CH ", SQ, PC, TMP) ;
1698                        CAPTURE_CH(SQ) := TRUE ;
1699                 END ;
1700          END ;
1701 COMMENT
1702 IF DEBUG AND TMP ~= 0 THEN WRITE("SACRIF", SQ, KK, CHECK_S(KK,SQ));
1703   END ;
1704   SEC(0,SQ) := SEC(1,SQ) + SEC(-1,SQ) ;
1705   ATTASQRS(10) := IX ;
1706      TMP := IF ID = 0
1707             THEN ATTACKERS(0)
1708             ELSE IF IX > ID
1709                  THEN IA
1710                  ELSE IX;
1711   LOSSES := COST ;
1712   IF LOSSES > -DELTA AND IX <= IA
1713   THEN BEGIN
1714     IF IX = 0 THEN IX := 1 ELSE
1715     LOSSES := LOSSES + FORKLOSS(SQ, IX, TMP) ;
1716     IF ~ CHECKCAPTURE AND IX = IA THEN
1717     FOR I := 1 UNTIL IX
1718     DO BEGIN
1719       SQA := ATTASQRS(I);
1720       PCA := BRD(SQA);
1721 IF DEBUG THEN WRITE("SAC", SQA, SQ);
1722       FOR J := K STEP K UNTIL CON(K,SQA)
1723       DO BEGIN
1724         SQ1 := CONTROL(SQA,J);
1725         PC1 := BRD(SQ1);
1726 IF DEBUG THEN WRITEON(SQ1, PC1, LOSSES);
1727         IF SQ1 ~= SQ AND BISHOP <= ABS(PC1) AND ABS(PC1) <= QUEEN
1728         THEN BEGIN
1729           T := HIDDEN(SQ1, SQA, SQ3, -K, FALSE);
1730           PC3 := BRD(SQ3);
1731           COMMENT is SQA pinned against SQ3 by SQ1;
1732           IF SQ3 ~= HOLE AND PC1*PC3 < 0 THEN BEGIN
1733             KSQ := IF PCA > 0 THEN KINGSQ(-1) ELSE KINGSQ(1);
1734             WITHCHECK := ABS(PCA) = PAWN AND ABS(KSQ-SQA-FILES(K)) = 1
1735                          OR BOTV(ABS(PCA), OFFSET(SQ)-OFFSET(KSQ)) = 1
1736                            AND CLEAR(SQ, SQ, KSQ);
1737           IF ~WITHCHECK THEN IF PC3*PCA > 0 THEN BEGIN
1738             DEFN := ABS(CON(-K,SQ3));
1739             PC3 := ABS(PC3); PCA := ABS(PCA);
1740           IF DEFN = 0 OR DEFN = 1 AND
1741              BOTV(PCA, OFFSET(SQA)-OFFSET(SQ3)) = 1 AND
1742              BOTV(PCA, OFFSET(SQ)-OFFSET(SQ3)) ~= 1
1743           THEN LOSSES := LOSSES - (IF ID = 0 THEN VALUES(PC3)
1744                                    ELSE IF VALUES(PCA) > VALUES(PC3)-DELTA
1745                                         THEN 0
1746                                         ELSE VALUES(PC3) -VALUES(PCA));
1747           END ELSE
1748             IF BOTV(ABS(PC3), OFFSET(SQ3) -OFFSET(SQ1)) = 1
1749             THEN BEGIN
1750               PC1 := ABS(PC1); PCA := ABS(PCA); PC3 := ABS(PC3);
```

```
1751                DEFN := ABS(CON(K,SQ1));
1752                IF DEFN = 0
1753                THEN LOSSES := LOSSES - VALUES(PC1) +
1754                               (IF ID = 0 THEN 0 ELSE VALUES(PCA))
1755                ELSE IF VALUES(PC1) > VALUES(PC3) - DELTA
1756                     THEN LOSSES := LOSSES - VALUES(PC1) + VALUES(PC3);
1757              END;
1758            END;
1759            END;
1760          END;
1761         END;
1762     END;
1763     IF PAWNS(SQ) = 0 AND IA <= ID AND MINPIECE < VALUES(KNIGHT)
1764     THEN LOSSES := LOSSES - VALUES(BISHOP) ;
1765     SACRIFICE := IF (LOSSES > -PVAL)
1766                  THEN TRUE
1767                  ELSE FALSE ;
1768     IF (DEBUG)  AND (PRINT) AND (IA > 0)
1769     THEN BEGIN
1770       IF SACRIFICE OR (ID ~= 0)
1771       THEN WRITE(COLOR(KK)," SACR",IA,MINATTACK,MAXATTACK, ATTACKERS(0),
1772         SQ,BRD(SQ), IX, HID,WEAKPIN, MAXPIECE, MINPIECE,ID,COST,LOSSES) ;
1773       IF (POWER(0) ~= SQR(0)) OR (ATTACKERS(0) ~= ATTASQRS(0))
1774                OR LOSSES ~= COST
1775       THEN BEGIN
1776         IA := IF (SQR(0) > ATTASQRS(0))
1777               THEN SQR(0)
1778               ELSE ATTASQRS(0) ;
1779         WRITE("HIDDEN A/D ON",SQ, LOST) ;
1780         FOR I := 0 UNTIL IA
1781         DO WRITE(SQR(I), POWER(I), ATTASQRS(I), ATTACKERS(I)) ;
1782         FOR I := SEC(-1,SQ) UNTIL SEC(1,SQ)
1783         DO WRITEON(SECRET(SQ,I)) ;
1784       END ;
1785     END ;
1786     IF ABS(LOSSES) < DELTA
1787     THEN LOSSES := 0 ;
1788     SACRIFICE
1789   END OFSACRIFICE ;
1790
1791   @TITLE," XRAY(INTEGER VALUE SQL,KK "
1792   PROCEDURE XRAY(INTEGER VALUE SQL,KK ;
1793   INTEGER VALUE RESULT NN, NT) ;
1794   COMMENT this procedure determines whether the piece (pc) on square
1795   sq is pinned against a major piece, and therefore cannot control
1796   sql. if it is, a check is made to see if the pinning piece (pca)
1797   is attacked by pc.  compromise, really want to know if pca can
1798   be captured by any piece. questionable pin if pca attacks sql ;
1799   COMMENT  SQ colour KK, SQA color -KK, SQH color KK, SQL color JK ;
1800   BEGIN
1801     INTEGER SQA, PCA, PC, SQH, PCH, SQ, I, PIN, SQD, PP, SQ3, VAL_PCL,
1802       VAL_PCH, VAL_PCA, JK, J, JJ, TMP, PCC, SQH_DEF, SAVEJ, NEWPIN ;
1803     LOGICAL T, PINNED ;
1804     I := 1 ;
1805     WEAKPIN := 0 ;
1806     VAL_PCL := VALUES(ABS(BRD(SQL))) ;
1807     JK := IF (BRD(SQL) < 0)
1808           THEN -1
1809           ELSE 1 ;
1810     WHILE (I <= NT)
1811     DO BEGIN
1812       SQ := SQR(I) ;
1813       IF (SQ < 0)
1814       THEN BEGIN
1815         SQR(I) := -SQ ;
1816         I := I+1 ;
1817       END ELSE
1818       BEGIN
1819         PINNED := FALSE ;
1820         PCC := BRD(SQ) ;
```

```
1821        PC := ABS(PCC) ;
1822        SAVEJ := 0 ;
1823        PIN := -FIVES ;
1824        FOR J := CON(-KK,SQ) STEP KK UNTIL -KK
1825        DO BEGIN
1826          SQA := CONTROL(SQ,J) ;
1827          PCA := ABS(BRD(SQA)) ;
1828              COMMENT a piece (on SQL) under attack cannot pin another ;
1829          IF (PCA >= BISHOP) AND (PCA <= QUEEN) AND (SQL ~= SQA)
1830          THEN BEGIN
1831            T := HIDDEN(SQA,SQ,SQH,-KK, FALSE) ;
1832            IF SQH ~= HOLE
1833            THEN BEGIN
1834              PCH := KK*BRD(SQH) ;
1835              SQH_DEF := CON(KK,SQH) ;
1836              VAL_PCA := VALUES(PCA) ;
1837              VAL_PCH := VALUES(ABS(PCH)) ;
1838              IF (PCH > 0) AND (VAL_PCH > VAL_PCA OR SQH_DEF = 0
1839                OR SEC(0,SQH)*KK < 0 AND BOTV(ABS(PCH), OFFSET(SQL)
1840                -OFFSET(SQH)) ~= 1)
1841              THEN BEGIN
1842                COMMENT PC is potentially pinned, can it capture PCA ;
1843                PINNED := TRUE ;
1844                FOR JJ := CON(KK,SQA) STEP -KK UNTIL KK
1845                DO BEGIN
1846                  COMMENT can pca be exchanged for a piece of nearly
1847                    equal value? ;
1848        COMMENT are sqa and sql the same, and being attacked by the
1849        "pinned" piece? ;
1850                  SQD := CONTROL(SQA,JJ) ;
1851                  IF (SQ ~= SQD) AND PCC*K < 0 AND (ABS(VAL_PCA -
1852                    VALUES(ABS(BRD(SQD)))) < DELTA)
1853                  THEN PINNED := FALSE ;
1854                END ;
1855      COMMENT does pinning piece (pca) attack sql? ;
1856              IF PINNED
1857              THEN BEGIN
1858                JJ := CON(-JK,SQL) ;
1859                IF DEBUG
1860                THEN WRITE("PIN ON", SQ,SQA,SQL,SQH,KK,JK, CON(JK,SQL),
1861                  JJ, CONTROL(SQL,JJ), SQH_DEF) ;
1862                IF KK = JK
1863                THEN IF (ABS(JJ) = 1) AND (SQA = CONTROL(SQL,JJ))
1864                     THEN PINNED := FALSE
1865                     ELSE IF ABS(SQH) ~= KING
1866                          THEN FOR JJ := JJ STEP JK UNTIL -JK
1867                          DO IF (SQA = CONTROL(SQL,JJ))
1868                             THEN IF VAL_PCL -VALUES(PC)
1869                               +(IF ABS(CON(JK,SQL)) > 1
1870                                 THEN VAL_PCA
1871                                 ELSE 0) > -VAL_PCH + (IF SQH_DEF ~= 0
1872                                           THEN VAL_PCA
1873                                           ELSE 0) + DELTA
1874                                 THEN PINNED := FALSE ;
1875                IF DEBUG
1876                THEN WRITEON(PINNED) ;
1877              END ;
1878            END ;
1879            IF PCH = KING AND PINNED
1880            THEN GO TO QUIT ;
1881            NEWPIN := VAL_PCH ;
1882            IF SQH_DEF > 1 OR SQH_DEF = 1 AND SQ ~= CONTROL(SQH,KK)
1883            THEN NEWPIN := NEWPIN - VAL_PCA ;
1884            IF PIN < NEWPIN
1885            THEN BEGIN
1886              PIN := NEWPIN ;
1887              SAVEJ := J ;
1888            END ;
1889            IF DEBUG
1890            THEN WRITE("PINS", PIN, NEWPIN, SAVEJ, J, SQA, SQH) ;
```

```
1891            END ;
1892          END ;
1893        END ;
1894  QUIT:
1895        IF ~ PINNED
1896        THEN I := I+1
1897        ELSE BEGIN
1898          IF (PCH ~= KING)
1899          THEN PINNED := FALSE ;
1900          IF ~PINNED
1901          THEN BEGIN
1902            SQA := CONTROL(SQ,SAVEJ) ;
1903            PCA := ABS(BRD(SQA)) ;
1904            PCH := KK*BRD(SQH) ;
1905            SQH_DEF := CON(KK,SQH) ;
1906            VAL_PCA := VALUES(PCA) ;
1907            VAL_PCH := VALUES(ABS(PCH)) ;
1908          END ;
1909          PIN := VAL_PCH ;
1910          IF ~PINNED
1911          THEN BEGIN
1912            IF SQH_DEF > 1 OR SQH_DEF = 1 AND SQ ~= CONTROL(SQH,KK)
1913            THEN PIN := PIN -VAL_PCA ;
1914            IF (PIN < DELTA)
1915            THEN PIN := 0 ;
1916                 COMMENT does PC on SQ give direct check on SQL capture.
1917                 What about discovered check? ;
1918            TMP := CHECK_S(KK,SQL) ;
1919            IF PCC < 0
1920            THEN TMP := -TMP ;
1921            WEAKPIN := IF TMP > 0 AND CHEC(TMP, PC)
1922                       THEN -1
1923                       ELSE PIN ;
1924            IF WEAKPIN = -1
1925            THEN WEAKPIN := IF CON(-KK,SQA) = 0  AND PCH >= BISHOP
1926              AND BOTV(PCH,OFFSET(SQH)-OFFSET(SQ)) = 1
1927                           THEN -VAL_PCA
1928                           ELSE 0 ;
1929          END ;
1930              COMMENT is there perhaps a hidden attack on SQH? ;
1931          IF (WEAKPIN = 0) AND (ABS(SQH_DEF) <= ABS(CON(-KK,SQH))
1932            +1) AND HIDDEN(SQ,SQA,SQ3,-KK, FALSE)
1933          THEN WEAKPIN := VAL_PCL + PVAL ;
1934          PP := POWER(I) - POWER(I-1) ;
1935          IF DEBUG
1936          THEN WRITEON(PIN,PP,POWER(I), TMP, WEAKPIN, PCH, I, NN, NT) ;
1937          PIN := PIN + PP ;
1938          J := I ;
1939          IF WEAKPIN >= 0
1940          THEN WHILE (PINNED AND J < NT OR J < NN AND PIN > PP)
1941               DO BEGIN
1942                 PP := POWER(J+1) - POWER(J) ;
1943                 IF DEBUG
1944                 THEN WRITEON(POWER(J), PP) ;
1945                 IF PINNED OR PIN > PP
1946                 THEN BEGIN
1947                   SQR(J) := SQR(J+1) ;
1948                   POWER(J) := POWER(J-1) + PP ;
1949                   J := J +1 ;
1950                 END ELSE POWER(J) := POWER(J-1) + VALUES(PC) ;
1951          END ;
1952          IF I = 1 AND J > 1
1953          THEN MINPIECE := POWER(1) ;
1954          SQR(J) := -SQ ;
1955          IF KK ~= JK AND (PINNED AND NN > ABS(CON(JK,SQL)) OR WEAKPIN
1956            < 0)
1957          THEN FOR II := CON(JK,SQL) STEP -JK UNTIL JK
1958          DO IF SQA = CONTROL(SQL,II)
1959             THEN BEGIN
1960               PINNED := FALSE ;
```

```
1961              IF WEAKPIN < 0
1962              THEN WEAKPIN := -VAL_PCL + VALUES(PC) -
1963                (IF ABS(CON(-JK,SQL)) > 1
1964                            THEN VAL_PCA
1965                            ELSE 1) ;
1966          END ;
1967          FOR L := J UNTIL NT
1968          DO POWER(L) := POWER(L) + WEAKPIN ;
1969          IF DEBUG
1970          THEN WRITEON(PIN, PP, PINNED, I, J, POWER(J)) ;
1971          IF PINNED
1972          THEN BEGIN
1973            NT := NT -1 ;
1974            IF (I <= NN)
1975            THEN NN := NN -1 ;
1976     COMMENT does the pinned piece stop a hidden attack ;
1977            IF (NT > NN)
1978            THEN IF HIDDEN(SQL, SQ, SQH, (IF PCC > 1
1979                                  THEN -1
1980                                  ELSE 1),FALSE)
1981            THEN FOR J := NN+1 UNTIL NT
1982            DO IF (SQH = SQR(J))
1983               THEN BEGIN
1984                 SQR(J) := SQR(NT) ;
1985                 NT := NT -1 ;
1986            END ;
1987          END ;
1988        END ;
1989      END ;
1990    END ;
1991     COMMENT this whole problem should be treated recursively, passing
1992     SQR and POWER as parameters.  Note also that in general the
1993     value of PIN, the incremental cost of moving a pinned piece,
1994     is overestimated ;
1995  END OFXRAY ;
1996
1997  @TITLE,"DELETE"
1998  PROCEDURE DELETE(INTEGER VALUE SQF, SQT, KK) ;
1999  BEGIN
2000    INTEGER T ;
2001    T := CON(KK,SQT) ;
2002    FOR L := T STEP -KK UNTIL KK
2003    DO IF (SQF = CONTROL(SQT,L))
2004       THEN BEGIN
2005         CONTROL(SQT,L) := CONTROL(SQT,T) ;
2006         CON(KK,SQT) := T - KK ;
2007         SEC(0,SQT) := CON(0,SQT) :=  CON(0,SQT) -KK ;
2008         GO TO NEXT ;
2009    END ;
2010  NEXT:
2011  END OFDELETE ;
2012
2013  INTEGER PROCEDURE DOUBLE(INTEGER VALUE SQ, DIR, K) ;
2014  BEGIN
2015    INTEGER C, Q, R, S, P;
2016    C := P := 0;
2017    Q := K*QUEEN ;
2018    R := K*ROOK ;
2019    FOR I := DIR, -DIR
2020    DO BEGIN
2021      S := SQ + I ;
2022      WHILE BRD(S) = 0
2023      DO S := S + I ;
2024      IF BRD(S) = Q OR BRD(S) = R
2025      THEN C := C + 1 ;
2026      WHILE BRD(S) ~= EDGE
2027      DO BEGIN
2028        IF ABS(BRD(S)) = PAWN
2029        THEN P := P +1;
2030        S := S + I;
```

```
2031      END;
2032    END ;
2033      IF P > 1 THEN C := 0;
2034    C
2035  END OF_DOUBLE ;
2036
2037  LOGICAL PROCEDURE DRAWS(INTEGER VALUE K) ;
2038  BEGIN
2039    IF TEST
2040    THEN WRITE ("DRAWS ", K, MEN(1), MEN(-1)) ;
2041        COMMENT does K have a mating force;
2042    MEN(K) = 1 OR MEN(K) = 2 AND PIECES(3*K)+PIECES(4*K) = 1
2043  END OFDRAWS ;
2044
2045  @TITLE,"MATE_THREAT"
2046  PROCEDURE MATE_THREAT ;
2047  BEGIN
2048    INTEGER T, TT, II, SQA, PC, SQ_K ;
2049     COMMENT back rank mate threat ;
2050    MATE_T(1) := MATE_T(-1) := BACKROW := 0 ;
2051    FOR KK := 1, -1
2052    DO IF (PIECES(KK*QUEEN) > 0 OR PIECES(KK*ROOK) > 0)
2053       THEN BEGIN
2054         SQ_K := KINGSQ(-KK) ;
2055         KING_HELD(-KK) := ROWS(SQ_K) = (IF KK = 1
2056                           THEN 8
2057                           ELSE 1) AND (POS(SQ_K) = 0 OR POS(SQ_K)
2058                             = 1 AND ABS(SQ_K-POSITION(SQ_K,1)) =
2059                             1 OR POS(SQ_K) = 2 AND ABS(POSITION(SQ_K,1)
2060                             - POSITION(SQ_K,2)) = 2);
2061  IF KING_HELD(-KK)
2062  THEN BEGIN
2063    COMMENT King is held on back rank;
2064      COMMENT look at every square on the back row, identify
2065                those which are not defended, yet are under attack;
2066    II := IF KK = 1
2067          THEN BKRSQ-7
2068          ELSE WQRSQ ;
2069    TT := 0 ;
2070    FOR SQ := II UNTIL II+7
2071    DO IF (SQ ~= SQ_K) AND CON(KK,SQ) ~= 0 AND KK*SEC(0,SQ) > 0
2072       THEN BEGIN
2073         T := 0 ;
2074    COMMENT How many major pieces bear on the back rank;
2075         FOR J := CON(KK,SQ) STEP -KK UNTIL KK
2076         DO BEGIN
2077           SQA := CONTROL(SQ,J) ;
2078           PC := ABS(BRD(SQA)) ;
2079           IF (ROWS(SQA) ~= ROWS(SQ) OR
2080                  SQA ~= SQ AND BRD(SQ) ~= 0) THEN
2081           IF PC = ROOK OR PC = QUEEN
2082           THEN BEGIN
2083             T := T + 1;
2084  COMMENT either 2(R or Q) bearing on back row, or R/Q
2085          attacker not comeing from 7th rank;
2086         IF T > ABS(CON(-KK,SQ)) AND (ABS(CON(KK,SQ)) > 1
2087                                       AND T > 1
2088            OR ABS(ROWS(SQ_K) - ROWS(SQA)) > 1)
2089         THEN TT := TT + (IF CHECK_S(-KK,SQ) ~= 0
2090                           THEN 6
2091                           ELSE IF KK = K AND M3 = SQA AND
2092                                   (COLS(BASE) ~= COLS(SQ))
2093                                THEN 3
2094                           ELSE T);
2095                       END;
2096         END ;
2097    END ;
2098    COMMENT does a back rank mate threat exist,
2099             or are we moving to the 7th rank?;
2100    IF TT > 0
```

```
2101   THEN MATE_T(KK) := MATE_T(KK) + (IF KK = K
2102                                    THEN 1
2103                                    ELSE -1)* (IF TT < 10
2104                         THEN TT
2105                         ELSE IF TT < 15
2106                              THEN TT-2
2107                              ELSE TT-6)
2108   ELSE IF (K = KK) AND (M4 = ROOK OR
2109     M4 = QUEEN) AND (ROWS(BASE) ~= ROWS(M3))
2110     AND ABS(ROWS(M3) -ROWS(SQ_K)) = 1
2111        THEN MATE_T(K) := MATE_T(K) +1 ;
2112        END ;
2113   END ;
2114 COMMENT reduce mate conter threat if giving check;
2115     IF GIVINGCH THEN MATE_T(-K) := MATE_T(-K) DIV 2;
2116   BACKROW := MATE_T(-1) + MATE_T(1) ;
2117   IF ~OPENING AND BACKROW < 0 AND LEVEL >= LENGTH
2118   THEN BACKROW := BACKROW -3 ;
2119           COMMENT discourage terminal moves with mate threats ;
2120 END OF_MATE_THREAT ;
2121
2122 COMMENT ****************fixit***************** ;
2123
2124 PROCEDURE FIXIT(INTEGER VALUE SQF, SQT, KK) ;
2125 BEGIN
2126   INTEGER M ;
2127   M := CON(KK,SQT) + KK ;
2128   CON(KK,SQT) := M ;
2129   CONTROL(SQT,M) := SQF ;
2130   SEC(0,SQT) := CON(0,SQT) := CON(0,SQT) + KK ;
2131 END OFFIXIT ;
2132
2133 @TITLE,"HIDDEN(INTEGER VALUE SQ1, SQ2 "
2134 LOGICAL PROCEDURE HIDDEN(INTEGER VALUE SQ1,SQ2 ;
2135 INTEGER RESULT SQ3 ;
2136 INTEGER VALUE KK ;
2137 LOGICAL VALUE PATH) ;
2138 COMMENT   hidden true if sq3 contains q, r or b of colour kk. ;
2139 COMMENT if no hidden piece then sq3 := hole ;
2140 BEGIN
2141   LOGICAL HIDDEN ;
2142   INTEGER SQ, PC ;
2143   SQ3 := HOLE ;
2144   HIDDEN := FALSE ;
2145   DIR := BOTV(EDGE, OFFSET(SQ1)-OFFSET(SQ2)) ;
2146   IF DIR = 0
2147   THEN BEGIN
2148     DIR := SQ2 - SQ1 ;
2149     GOTO QUIT ;
2150   END ;
2151   IF PATH AND SQ1 + DIR ~= SQ2
2152   THEN FOR SQ := SQ1+DIR STEP DIR UNTIL SQ2-DIR
2153   DO IF BRD(SQ) ~= 0
2154      THEN GO TO QUIT ;
2155   SQ := SQ2 + DIR ;
2156   WHILE (BRD(SQ) = EMPTY)
2157   DO SQ := SQ + DIR ;
2158   IF (BRD(SQ) = EDGE)
2159   THEN GO TO QUIT ;
2160   SQ3 := SQ ;
2161   PC := IF (KK = 0)
2162         THEN ABS(BRD(SQ))
2163         ELSE KK*BRD(SQ) ;
2164   IF PC = QUEEN OR (PC = ROOK OR PC = BISHOP) AND BOTV(PC, OFFSET(SQ3)
2165     -OFFSET(SQ2)) = 1
2166   THEN HIDDEN := TRUE ;
2167 QUIT:
2168   HIDDEN
2169 END OFSEARCHFORHIDDENATTACK ;
2170
```

```
@TITLE,"HIDDENATTACK"
PROCEDURE HIDATTACK ;
BEGIN
  INTEGER PC, I, SQ2, SQ3, L2 ;
  LOGICAL T ;
  L2 := LOSS(ELIMIT) ;
  FOR I := POSITION(M3,LIM) UNTIL LIM-1
  DO BEGIN
    SQ2 := POSITION(M3,I) ;
    T := HIDDEN(M3, SQ2, SQ3,0, FALSE) ;
    IF SQ3 ~= HOLE AND (K*BRD(SQ3) < 0)
    THEN BEGIN
      COMMENT Q-Q, B-B, R-R ;
         COMMENT case of Q or B inline with a P is already handled
         by existing mechanism in MAKEFINAL, since a non-losing
         exchange occurs ;
      PC := ABS(BRD(SQ2)) ;
      IF (M4 = PC) OR (((M4 = BISHOP) OR (M4 = ROOK)) AND (PC = QUEEN))
        OR M4 = QUEEN AND (PC =ROOK OR PC = BISHOP) AND BOTV(PC,
        OFFSET(SQ2) -OFFSET(M3)) = 1
      THEN BEGIN
        L2 := L2 -1 ;
        SQUARE(L2) := SQ3 ;
      END ;
    END ;
  END ;
  LOSS(ELIMIT) := L2 ;
END OFHIDATTACK ;

@TITLE,"KINGCONTROL(INTEGER VALUE K) "
COMMENT ***************kingcontrol*************** ;

PROCEDURE KINGCONTROL(INTEGER VALUE K) ;
BEGIN
  INTEGER J,KSQ,SQ ;
  J := 0 ;
  KSQ := KINGSQ(K) ;
  FOR I := -LDIAG,-FILE,-RDIAG,-RANK,LDIAG,FILE,RDIAG,RANK
  DO BEGIN
    SQ := KSQ + I ;
    IF (BRD(SQ) ~= EDGE)
    THEN BEGIN
      J := J + K ;
      KSQRS(J) := SQ ;
    END ;
  END ;
  K_SQRS(K) := J ;
END OFKINGCONTROL ;

PROCEDURE KINGSMOVE (INTEGER VALUE HOME, K) ;
BEGIN
  INTEGER ONE, M6, SQ, TWO ;
  ONE := TWO := 0 ;
  FOR M := CON(-K,HOME) STEP K UNTIL -K
  DO BEGIN
    SQ := CONTROL(HOME,M) ;
    IF (ABS(BRD(SQ)) > KNIGHT)
    THEN BEGIN
      DIR := BOTV(EDGE, OFFSET(SQ)-OFFSET(HOME)) ;
      IF (ONE = 0)
      THEN ONE := DIR
      ELSE TWO := DIR ;
    END ;
  END ;
  FOR L := LDIAG,FILE,RDIAG,-RANK,RANK,-RDIAG,-FILE,-LDIAG
  DO BEGIN
    SQ := HOME + L ;
    IF (BRD(SQ) ~= EDGE)
    THEN BEGIN
      M6 :=  K*BRD(SQ) ;
```

```
2241        IF (M6 = -ENPRIS)
2242        THEN M6 := NIL ;
2243        IF (M6 > NIL) OR (CON(-K,SQ) ~= 0) OR (L = ONE) OR (L = TWO)
2244        THEN INDEX := SLIST := SLIST -1
2245        ELSE INDEX := CLIST := CLIST +1 ;
2246        CHECKLIST(INDEX) := SQ ;
2247           COMMENT Note difficulties with the King:  It can move
2248           to squares it does not control, castling. It can attack
2249           squares to which it cannot move, those defended by opponent.
2250           In the latter case that square is added to the defence list. ;
2251      END ;
2252    END ;
2253  END OFKINGSMOVE ;
2254
2255  @TITLE,"LISTKINGMOVES  "
2256  COMMENT *****************list king moves************** ;
2257
2258  PROCEDURE LISTKINGMOVES (INTEGER VALUE SQ, K) ;
2259  BEGIN
2260    INTEGER   KSQ ;
2261    KINGSMOVE(SQ, K) ;
2262    KLIST := CLIST ;
2263    KSQ := IF (K > 0)
2264           THEN 15
2265           ELSE 85 ;
2266    IF (SQ = KSQ) AND (CON(-K,KSQ) = 0)
2267    THEN BEGIN
2268      IF CASTLE(K) AND (BRD(2+KSQ) = EMPTY) AND (BRD(1+KSQ) = EMPTY)
2269        AND K*BRD(KSQ+3) = ROOK
2270      THEN IF CON(-K,1+KSQ) = 0
2271           THEN IF CON(-K,2+KSQ) = 0
2272                THEN BEGIN
2273                  CLIST := CLIST + 1 ;
2274                  CHECKLIST(CLIST) := 2 + KSQ ;
2275      END ;
2276      IF CASTLE(TWO(K)) AND (BRD(KSQ-3) = EMPTY) AND (BRD(KSQ-2) = EMPTY)
2277        AND (BRD(KSQ-1) = EMPTY) AND K*BRD(KSQ-4) = ROOK
2278      THEN IF CON(-K,KSQ-1) = 0
2279           THEN IF CON(-K,KSQ-2) = 0
2280                THEN BEGIN
2281                  CLIST := CLIST + 1 ;
2282                  CHECKLIST(CLIST) := KSQ-2 ;
2283      END ;
2284    END ;
2285     COMMENT   order important ;
2286  END OFLISTKINGMOVES ;
2287
2288  @TITLE,"LISTMOVES "
2289  PROCEDURE LISTMOVES ;
2290  COMMENT all moves are listed including some which keep own king
2291  in check ;
2292  BEGIN
2293    INTEGER M, PC, N, SQ, R, II, LL, RR, TMP ;
2294    ENPRISE (K) ;
2295    N := 0 ;
2296    LL := 0 ;
2297    RR := TOTAL + 1 ;
2298    FOR J := (IF K=1
2299              THEN WQRSQ
2300              ELSE BKRSQ - 7) STEP FILES(K) UNTIL (IF K=1
2301                                                   THEN BKRSQ
2302                                                   ELSE WQRSQ)
2303    DO FOR I := J UNTIL J+7
2304    DO IF (K*BRD(I) > 0)
2305       THEN BEGIN
2306         M := POS(I) ;
2307         PC := ABS(BRD(I)) ;
2308         IF (PC ~= ENPRIS)
2309         THEN FOR L := 1 UNTIL M
2310              DO BEGIN
```

```
2311                SQ := POSITION(I,L) ;
2312                IF (PC ~= PAWN) OR (PAWNS(SQ) ~= 0)
2313                THEN BEGIN
2314                  N := N+K ;
2315                  QUIES(N) := FALSE;
2316                  SCORE(N) := NOSCR ;
2317                  MOVEFROM(N) := I ;
2318                  MOVETO(N) := SQ ;
2319                  R := ABS(BRD(SQ)) ;
2320                  REWARD(N) := IF (PC = PAWN)
2321                               THEN IF (ABS(I - SQ) = TWOROWS)
2322                                    THEN PFTWO
2323                                    ELSE R
2324                                    ELSE IF (R = ENPRIS)
2325                     THEN 0
2326                     ELSE IF (PC = KING)
2327                          THEN IF (ABS(SQ - I) = 2)
2328                               THEN IF (I > SQ)
2329                                    THEN QSIDE
2330                                    ELSE KSIDE
2331                               ELSE R
2332                          ELSE R ;
2333                  IF R ~= 0
2334                  THEN II := RR := RR - 1
2335                  ELSE BEGIN
2336                    TMP := K*CHECK_S(-K,SQ) ;
2337                    IF TMP > 0 AND CHEC(TMP,PC)
2338                    THEN II := RR := RR - 1
2339                    ELSE II := LL := LL + 1 ;
2340                  END ;
2341                  USE(II) := N ;
2342                END ELSE FOR L := 1 UNTIL 4
2343                         DO BEGIN
2344                           N := N +K ;
2345                           RR := RR - 1 ;
2346                           USE(RR) := N ;
2347                           REWARD(N) := PROMOTE + (CASE L OF (QUEEN,
2348                             KNIGHT, ROOK, BISHOP)) ;
2349                           MOVEFROM(N) := I ;
2350                           MOVETO(N) := SQ ;
2351                           SCORE(N) := NOSCR ;
2352                END ;
2353           END ;
2354     END ;
2355     NUMBER(K) := N ;
2356     NUM := ABS(N) ;
2357     LL := LL + 1 ;
2358     FOR I := RR UNTIL TOTAL
2359     DO USE(LL+I-RR) := USE(I) ;
2360     USE(0) := 0 ;
2361     IF NUM > 77
2362     THEN BEGIN
2363       WRITE("#MOVES =", N, LEVEL) ;
2364       PUNCHBRD(TRUE, HISTORY) ;
2365       WRITE(HISTORY(0|114)) ;
2366     END ;
2367   END OFLISTMOVES ;
2368   @TITLE,"LISTPAWNMOVES "
2369   PROCEDURE LISTPAWNMOVES (INTEGER VALUE HOME, K) ;
2370   BEGIN
2371     INTEGER INDEX, MOVE, PC, I ;
2372     I := HOME ;
2373      COMMENT  enpassant handled by move r. & move l ;
2374      COMMENT right-side capture? ;
2375     FOR J := RDIAG, LDIAG
2376     DO BEGIN
2377       MOVE := I + J*K ;
2378       PC := BRD(MOVE) ;
2379       IF (PC ~= EDGE)
2380       THEN BEGIN
```

```
2381        IF (K*PC < EMPTY)
2382        THEN INDEX := CLIST := CLIST + 1
2383        ELSE INDEX := SLIST := SLIST -1 ;
2384        CHECKLIST(INDEX) := MOVE ;
2385      END ;
2386    END ;
2387     COMMENT left-side capture? ;
2388    KLIST := CLIST ;
2389    MOVE := I + FILES(K) ;
2390    IF (BRD(MOVE) = EMPTY)
2391    THEN BEGIN
2392      IF (ABS(PAWNS(HOME)) = 1)
2393      COMMENT is pawn on second rank? ;
2394  THEN BEGIN
2395    I := MOVE + FILES(K) ;
2396    IF BRD(I) = EMPTY
2397    THEN BEGIN
2398      CLIST := CLIST + 1 ;
2399      CHECKLIST(CLIST) := I ;
2400    END ;
2401      END ;
2402        COMMENT increm. move order important ;
2403      CLIST := CLIST + 1 ;
2404      CHECKLIST(CLIST) := MOVE ;
2405    END OFPAWNFORWARD ;
2406  END OFLISTPAWNMOVES ;
2407
2408
2409  PROCEDURE LISTQUEENMOVES (INTEGER VALUE HOME, K) ;
2410  BEGIN
2411    TRYMINORMOVE(HOME, LDIAG, RDIAG, K) ;
2412    TRYMINORMOVE(HOME, FILE, RANK, K) ;
2413  END OFLISTQUEENMOVES ;
2414
2415  @TITLE,"SEEKPAWN(INTEGER VALUE SQ) "
2416  PROCEDURE SEEKPAWN(INTEGER VALUE SQ) ;
2417  BEGIN
2418    INTEGER SQP, PC ;
2419    IF (SQ > 30)
2420    THEN BEGIN
2421      SQP := SQ - FILE ;
2422      PC := BRD(SQP) ;
2423      IF (PC = PAWN)
2424      THEN REMAKE(SQP, SQ)
2425      ELSE IF (PC = NIL) AND (ROWS(SQ) = 4)
2426           THEN IF (BRD(SQP - FILE) = PAWN)
2427                THEN REMAKE(SQP-FILE,SQ) ;
2428    END ;
2429    IF (SQ < 70)
2430    THEN BEGIN
2431      SQP := SQ + FILE ;
2432      PC := BRD(SQP) ;
2433      IF (PC = -PAWN)
2434      THEN REMAKE(SQP,SQ)
2435      ELSE IF (PC = NIL) AND (ROWS(SQ) = 5)
2436           THEN IF (BRD(SQP+FILE) = -PAWN)
2437                THEN REMAKE(SQP+FILE, SQ) ;
2438    END ;
2439  END OFSEEKPAWN ;
2440
2441  @TITLE,"TRYKNIGHTMOVE (INTEGER VALUE M) "
2442  PROCEDURE TRYKNIGHTMOVE (INTEGER VALUE HOME, K) ;
2443  BEGIN
2444    INTEGER M6 ;
2445    FOR I := 1 UNTIL 8
2446    DO BEGIN
2447      M6 := BRD(HOME+S(I)) ;
2448      IF M6 = EDGE
2449      THEN GO TO QUIT ;
2450      M6  := K * M6 ;
```

```
2451      IF (ABS(M6) = ENPRIS)
2452      THEN M6 := NIL ;
2453      IF (M6 <= NIL)
2454      THEN INDEX := CLIST := CLIST +1
2455      ELSE INDEX := SLIST := SLIST -1 ;
2456      CHECKLIST(INDEX) := HOME +S(I) ;
2457  QUIT:
2458    END ;
2459  END OFTRYKNIGHTMOVE ;
2460
2461  PROCEDURE TRYMINORMOVE (INTEGER VALUE HOME, LEFT, RIGHT, K) ;
2462  BEGIN
2463    INTEGER M6, H ;
2464    FOR D := LEFT, RIGHT, -LEFT, -RIGHT
2465    DO BEGIN
2466      H := HOME ;
2467      M6 := 0 ;
2468      WHILE (M6 = 0)
2469      DO BEGIN
2470        H := H + D ;
2471        M6 := BRD(H) ;
2472        IF (M6 ~= EDGE)
2473        THEN BEGIN
2474          M6 := K*M6 ;
2475          IF (ABS(M6) = ENPRIS)
2476          THEN M6 := 0 ;
2477          IF (M6 <= 0)
2478          THEN INDEX := CLIST := CLIST + 1
2479          ELSE INDEX := SLIST := SLIST -1 ;
2480          CHECKLIST(INDEX) := H ;
2481        END ;
2482      END OFWHILELOOP ;
2483    END ;
2484  END OFTRYMINORMOVE ;
2485
2486  @TITLE,"UNMAKE"
2487  COMMENT ******************unmake************** ;
2488
2489  PROCEDURE UNMAKE(INTEGER VALUE SQ) ;
2490  BEGIN
2491    INTEGER I,KK,T,J,PC ;
2492     COMMENT CAPTURE_CH(SQ) := FALSE;
2493    PC := BRD(SQ) ;
2494     COMMENT IF DEBUG THEN WRITE("UNM",SQ,PC) ;
2495    IF (PC ~= 0)
2496    THEN BEGIN
2497      KK := IF (PC > 0)
2498            THEN 1
2499            ELSE -1 ;
2500      FOR J := 1 UNTIL POSITION(SQ,0)
2501      DO DELETE(SQ, POSITION(SQ,J), KK) ;
2502      FOR J := POSITION(SQ,LIM) UNTIL LIM-1
2503      DO DELETE(SQ, POSITION(SQ,J), KK) ;
2504      POS(SQ) := POSITION(SQ,0) := 0 ;
2505      POSITION(SQ,LIM) := LIM ;
2506      BRD(SQ) := EMPTY ;
2507      SEEKPAWN(SQ) ;
2508    END ;
2509  END OFUNMAKE ;
2510
2511  @TITLE,"OPENFILE, BACKPAWN, POTENTIAL AND PP_UN_BLOCK"
2512  INTEGER PROCEDURE OPENFILE(INTEGER VALUE SQ) ;
2513  BEGIN
2514    INTEGER L, OPEN, PC, PCD ;
2515    LOGICAL FEAR ;
2516     COMMENT compute credit for playing Rooks on open or semi- open
2517     files.  Treat semi-open files with advanced pawn as open file ;
2518     COMMENT  PC is a Rook or a Queen;
2519    OPEN := 2 ;
2520    PC := BRD(SQ) ;
```

```
2521    FEAR := FALSE ;
2522    FOR J := FILE, -FILE
2523    DO BEGIN
2524      L := IF (J < 0)
2525           THEN WQRSQ
2526           ELSE BKRSQ ;
2527      FOR I := SQ+J STEP J UNTIL L
2528      DO BEGIN
2529        COMMENT don't count half open file with R/Q behind
2530                a back pawn. Note PC*J > 0 when forward direction;
2531        PCD := BRD(I) ;
2532        IF ABS(PCD) = PAWN
2533        THEN BEGIN
2534          IF PC*PCD < 0 OR PC*J < 0 OR
2535             RANKS(ROWS(I))*PCD > 1
2536          THEN OPEN := OPEN -1
2537          ELSE OPEN := OPEN -2;
2538        END
2539        ELSE IF ABS(PCD) ~= QUEEN AND ABS(PCD) ~= ROOK AND PCD*PC
2540          > 0 AND PAWNS(I) ~= 0
2541             THEN FEAR := TRUE ;
2542      END ;
2543    END ;
2544    COMMENT don't put R/Q on same open file as a minor piece
2545            for fear of a pin;
2546    IF FEAR AND OPEN = 2
2547    THEN OPEN := 1 ;
2548    IF (OPEN < 0)
2549    THEN 0
2550    ELSE OPEN
2551  END OFOPENFILE ;
2552
2553  INTEGER PROCEDURE BACKPAWN(INTEGER VALUE SQ, D, PC) ;
2554  BEGIN
2555    INTEGER B, L ;
2556    B := 1 ;
2557    L := IF (D > 0)
2558         THEN BKRSQ
2559         ELSE WQRSQ ;
2560    FOR K := -1, 1
2561    DO FOR J := SQ  +K STEP D UNTIL L
2562    DO IF (BRD(J) = PC)
2563       THEN B := B -1 ;
2564    B
2565  END OFBACKPAWN ;
2566
2567  INTEGER PROCEDURE POTENTIAL(INTEGER VALUE SQ1, SQ2 ;
2568  LOGICAL VALUE PASS) ;
2569  BEGIN
2570    INTEGER I, J, C ;
2571    IF PASS AND ENDGAME
2572    THEN C := BOTV((IF K < 0
2573                    THEN 1
2574                    ELSE 2), OFFSET(SQ1)-OFFSET(SQ2))
2575    ELSE BEGIN
2576      I := ABS(COLS(SQ1) - COLS(SQ2)) ;
2577      J := ABS(ROWS(SQ1) - ROWS(SQ2)) ;
2578      C := IF ENDGAME
2579           THEN FAR(I,J)
2580           ELSE NEAR(I,J)
2581    END ;
2582    IF DEBUG OR C<0
2583    THEN WRITE("POTEN", K, SQ1, SQ2, C, PASS) ;
2584    C
2585  END OFPOTENTIAL ;
2586
2587  @TITLE "PP_BLOCK"
2588  INTEGER PROCEDURE PP_BLOCK(INTEGER VALUE SQ, INC) ;
2589  BEGIN
2590    INTEGER J, L, V, PC ;
```

```
2591 COMMENT are opponent's passed pawns blocked;
2592   V := 0 ;
2593   L := K*SEC(0,SQ+INC) ;
2594   IF DEBUG
2595   THEN WRITE("PP_BL", SQ, BRD(SQ), L) ;
2596   FOR I := INC, -INC
2597   DO BEGIN
2598     J := SQ + I ;
2599     WHILE BRD(J) = 0
2600     DO J := J + I ;
2601     PC := K*BRD(J) ;
2602 COMMENT is there a R or Q on the file of the pp;
2603     IF ABS(PC) = ROOK OR ABS(PC) = QUEEN
2604     THEN BEGIN
2605 COMMENT piece is behind, so change control value on
2606           the square in front of the pp;
2607       IF I = -INC
2608       THEN L := L + (IF PC > 0
2609                      THEN 1
2610                      ELSE -1) ;
2611 COMMENT if or R/Q on the file is not attacked
2612           then increment V;
2613       IF PC > 0
2614       THEN IF CON(-K,J) = 0
2615            THEN V := V + (IF K > 0
2616                           THEN ROWS(SQ)
2617                           ELSE 9 - ROWS(SQ)) DIV 2
2618            ELSE IF K*SEC(0,J) >= 0
2619                 THEN V := V + 1 ;
2620 COMMENT else if our R/Q is attacked, but safe,
2621               increment V by 1;
2622     END ;
2623     IF DEBUG
2624     THEN WRITEON(J, V) ;
2625   END ;
2626 COMMENT is the pawn safely blocked;
2627   IF BRD(SQ+INC) ~= 0
2628   THEN V := V + 1
2629   ELSE V := V + (IF L > 0
2630                  THEN 1
2631                  ELSE -(IF K > 0
2632                         THEN 9-ROWS(SQ)
2633                         ELSE ROWS(SQ)));
2634   IF DEBUG
2635   THEN WRITEON(L, V) ;
2636     IF V < 0 THEN V := 0;
2637   V
2638 END OF_PP_BLOCK ;
2639
2640 PROCEDURE CHECK_SQRS(INTEGER VALUE KK) ;
2641 BEGIN
2642   INTEGER PC, KSQ, SQC ;
2643   FOR SQ := WQRSQ UNTIL BKRSQ
2644   DO IF CHECK_S(KK,SQ) ~= 0
2645      THEN CHECK_S(KK,SQ) := FORKS(KK,SQ) := 0 ;
2646    COMMENT only really necessary when King moves ;
2647   KSQ := KINGSQ(KK) ;
2648   FOR M :=  KNIGHT, BISHOP, ROOK, PAWN
2649   DO BEGIN
2650     BRD(KSQ) := PC := KK*M ;
2651     MOVESQUARE(KSQ) ;
2652     FOR I := SLIST UNTIL LIM-1
2653     DO BEGIN
2654       KLIST := KLIST + 1 ;
2655       CHECKLIST(KLIST) := CHECKLIST(I) ;
2656     END ;
2657     FOR I := 1 UNTIL KLIST
2658     DO BEGIN
2659       SQC := CHECKLIST(I) ;
2660  COMMENT IF DEBUG AND BRD(SQC) = 0 THEN
```

```
2661 WRITE("CHE", SQC, KK, PC, PIECES(-PC), M, CON(KK,SQC));
2662       CHECK_S(KK,SQC) := -PC ;
2663       IF BRD(SQC) = 0 AND (PIECES(-PC) > 0 OR M ~= KNIGHT
2664         AND PIECES(-KK*QUEEN) > 0) AND (CON(KK,SQC) = 0 OR M = PAWN)
2665       THEN FORKING(KSQ, SQC, M, KK) ;
2666     END ;
2667   END ;
2668   BRD(KSQ) := KK*KING ;
2669 END OF_CHECK_SQRS ;
2670
2671 @TITLE,"KNIGHT_DEFENCE"
2672 LOGICAL PROCEDURE KN_DEFENCE(INTEGER VALUE SQ, SQH, K) ;
2673 BEGIN
2674   INTEGER DIST, A, B ;
2675   LOGICAL SAFE ;
2676   SAFE := FALSE ;
2677   IF BOTV(KNIGHT, OFFSET(SQ)-OFFSET(SQH)) = 2
2678   THEN BEGIN
2679     DIST := ABS(SQH - SQ) ;
2680     A := DOUBLE_KNIGHT(DIST) ;
2681 COMMENT
2682    .. .. 38 .. 40 .. 42 .. ..      .. .. 19 .. 21 .. 21 .. ..
2683    .. 27 .. 29 .. 31 .. 33 ..      .. 19 .. 21 .. 12 .. 21 ..
2684    16 .. .. .. 20 .. .. .. 24      08 .. .. .. 12 .. .. .. 12
2685    .. 07 .. 09 .. 11 .. 13 ..      .. 19 .. 21 .. 19 .. 21 ..
2686    .. .. .. .. ** .. 02 .. 04      .. .. .. .. ** .. 21 .. 12
2687                 Corrected 2 April 1986 ;
2688     IF A > 0
2689     THEN BEGIN
2690       B := DIST - A ;
2691       IF SQ > SQH
2692       THEN BEGIN
2693         A := -A ;
2694         B := -B ;
2695       END ;
2696       IF CON(K,SQ+A) = 0 AND K*BRD(SQ+A) >= 0
2697       THEN SAFE := TRUE
2698       ELSE IF A ~= B AND CON(K, SQ+B) = 0 AND K*BRD(SQ+B) >= 0
2699            THEN SAFE := TRUE;
2700     END ;
2701   END ;
2702   SAFE
2703 END OFKN_DEFENCE ;
2704
2705 INTEGER PROCEDURE ISOLATE(INTEGER VALUE COL, PAWN, FILE, K) ;
2706 BEGIN
2707   INTEGER I, C, M ;
2708   C := 0 ;
2709   M := QRSQ(K) - 1 + COL + FILE ;
2710   FOR L := 0 STEP FILE UNTIL 5*FILE
2711   DO IF BRD(L+M) = PAWN
2712      THEN C:= -1 ;
2713   IF C < 0
2714   THEN FOR L := -1, 1
2715        DO BEGIN
2716          I := M + L ;
2717          WHILE BRD(I) ~= EDGE AND BRD(I) ~= PAWN
2718          DO I := I + FILE ;
2719          IF BRD(I) = PAWN
2720          THEN C := C + 1 ;
2721   END ;
2722   IF C > 0
2723   THEN C := 0 ;
2724   IF DEBUG
2725   THEN WRITE("ISO", COL, PAWN, FILE, K, C, I) ;
2726   C
2727 END OFISOLATE ;
2728
2729 @TITLE,"ENPRISE"
2730 PROCEDURE ENPRISE (INTEGER VALUE K) ;
```

```
2731 COMMENT   determines which moving pieces are under attack. ;
2732 BEGIN
2733   INTEGER PC, M, KSQ, KK, MEN ;
2734   MEN := PIECES(0) - PIECES(PAWN) - PIECES(-PAWN) ;
2735   LATE_END := PIECES(QUEEN) = 0 AND PIECES(-QUEEN) = 0 ;
2736   IF OPEN(K)
2737   THEN BEGIN
2738     M  := IF (K > 0)
2739           THEN 15
2740           ELSE 85 ;
2741     AGITATE := 0 ;
2742     FOR I := -3,-2,1,2
2743     DO BEGIN
2744       PC := ABS(BRD(I+M)) ;
2745       IF (PC ~= EMPTY) AND (PC < ROOK)
2746       THEN AGITATE := AGITATE -1 ;
2747          COMMENT agitate backrow minor pieces ;
2748     END ;
2749     IF (LEVEL <= 1) AND ~ALTERNATIVE
2750     THEN BEGIN
2751       IF EARLY
2752       THEN IF MOVECOUNT > 8  OR AGITATE = 0
2753            THEN NEWGAME := EARLY := FALSE
2754            ELSE IF WHO AND (MOVECOUNT = 2)
2755                 THEN LOOKAHEAD := TRUE ;
2756       IF (MOVECOUNT > 16 OR MEN <= 8 OR AGITATE = 0 AND PIECES(0) <=26)
2757       THEN BEGIN
2758         WRITE("START MIDDLE GAME ",COLOR(K)) ;
2759         OPEN(K) := FALSE ;
2760         IF ~OPEN(-K)
2761         THEN MMTTS(-10002, HISTORY) ;
2762       END ;
2763     END ;
2764   END ;
2765   OPENING := OPEN(K) ;
2766   ENDGAME := IF ~OPENING AND ( MEN <= 8
2767                   OR LATE_END AND PIECES(0) <= 18 OR PIECES(0) <= 12)
2768              THEN TRUE
2769              ELSE FALSE ;
2770   LATE_END := ENDGAME AND LATE_END AND (MEN <= 6 AND PIECES(0) <= 14
2771             OR MEN < 6 AND PIECES(ROOK) = 0 AND PIECES(-ROOK) = 0) ;
2772   KK :=      IF ALTERNATIVE
2773              THEN -K
2774              ELSE K ;
2775   CHECK_SQRS(KK) ;
2776   KSQ := KINGSQ(KK) ;
2777   M := 0 ;
2778   FOR SQ := WQRSQ UNTIL BKRSQ
2779   DO BEGIN
2780     DECOYS(SQ) := 0;
2781     PC := BRD(SQ) ;
2782     IF (PC ~= EDGE) AND (K*PC > 0)
2783     THEN IF (SQ ~= KSQ)
2784          THEN IF (CON(-K,SQ) ~= 0)
2785               THEN IF (SACRIFICE(SQ, DEBUG) OR SQR(10) ~= ATTASQRS(10))
2786                    THEN BEGIN
2787                      COMMENT   attacked directly ;
2788                      M := M +1 ;
2789                      SQUARE(M) := SQ ;
2790                      LOSS(M) := LOSSES ;
2791     END ;
2792   END OFFORSQLOOP ;
2793   LOSS(0) := SQUARE(0) := M ;
2794 END OFENPRISE ;
2795
2796 @TITLE,"FORKLOSS"
2797 INTEGER PROCEDURE FORKLOSS(INTEGER VALUE SQ, FROM, UNTO) ;
2798 BEGIN
2799   INTEGER LOSS, HI, HIGH, PCD, SQD, PC, PCA, APC, LOST, K_PAWN,
2800     ASQ, KK, START ;
```

```
2801
2802   PROCEDURE FKLOSS(INTEGER VALUE RESULT LOST) ;
2803   BEGIN
2804     MOVESQUARE(SQ) ;
2805     HIGH := LOSS := 0 ;
2806     PCA := VALUES(ABS(APC)) ;
2807     FOR L := 1 UNTIL KLIST
2808     DO BEGIN
2809       SQD := CHECKLIST(L) ;
2810       PCD := ABS(BRD(SQD)) ;
2811       IF (PCD > NIL)
2812       THEN BEGIN
2813         HI := PCD := VALUES(PCD) ;
2814     COMMENT can a defender recapture on SQ;
2815         FOR J := 1 UNTIL SQR(0)
2816         DO IF SQD = SQR(J)
2817            THEN IF PCD < PCA + DELTA OR ABS(CON(KK,SQ)) =1
2818                 THEN GO TO SKIP
2819                 ELSE HI := HI - PCA ;
2820         IF PCD ~= KINGVAL AND HI = PCD
2821         THEN IF (CON(-KK,SQD) ~= 0) OR ABS(APC) ~= KNIGHT
2822           AND HIDDEN(SQ, ASQ, SQD, -KK, FALSE)
2823              THEN IF (PCA < PCD + DELTA)
2824                   THEN HI := HI - PCA
2825                   ELSE GO TO NEXT ;
2826         IF (HI > HIGH)
2827         THEN BEGIN
2828           LOSS := HIGH ;
2829           HIGH := HI ;
2830         END ELSE IF (HI > LOSS)
2831                  THEN LOSS := HI ;
2832       END ;
2833 NEXT:
2834     END ;
2835 SKIP:
2836     COMMENT   must re-org. in order to call sacrifice ;
2837     IF DEBUG
2838     THEN IF (LOSS ~= 0)
2839          THEN WRITE("FORK",SQ,PCA,LOSS,LOST,HI,HIGH,APC,ASQ,FROM,UNTO) ;
2840     IF (LOSS > LOST)
2841     THEN LOST := LOSS ;
2842   END OFFKLOSS ;
2843 @TITLE,"FORKLOSS"
2844   PC := BRD(SQ) ;
2845   LOST := 0 ;
2846   KK := IF PC > 0
2847         THEN -1
2848         ELSE 1 ;
2849   K_PAWN := KK*PAWN ;
2850    COMMENT RANK_7 := 30 +50*K ;
2851   FOR I := FROM UNTIL UNTO
2852   DO BEGIN
2853     ASQ := ATTASQRS(I) ;
2854     APC := BRD(SQ) := BRD(ASQ) ;
2855     BRD(ASQ) := 0 ;
2856     IF (APC ~= K_PAWN) OR (PAWNS(SQ) ~= 0)
2857     THEN FKLOSS(LOST)
2858     ELSE FOR PPC := KNIGHT,BISHOP,ROOK,QUEEN
2859     DO BEGIN
2860       COMMENT pawn about to promote ;
2861       APC := BRD(SQ) := PPC*KK ;
2862       FKLOSS(LOST) ;
2863       APC := K_PAWN ;
2864     END ;
2865     BRD(ASQ) := APC ;
2866   END ;
2867   BRD(SQ) := PC ;
2868   LOST
2869 END OFFORKLOSS ;
2870
```

```
2871 @TITLE,"GAMERECORD "
2872 PROCEDURE GAMERECORD (LOGICAL VALUE FINIS) ;
2873 BEGIN
2874   INTEGER F ;
2875   STRING(20) NAME ;
2876   STRING(20) SPACES ;
2877   NAME := "    Amdahl 5860/2" ;
2878   I_W := 4 ;
2879   IF FINIS
2880   THEN BEGIN
2881     COMMENT send score to WITA:GAMES ;
2882     MMTTS(-9999,HISTORY) ;
2883     PUT(9) ;
2884   END ;
2885   SPACES := " " ;
2886   F := COUNT - (MOVECOUNT-INITIALCNT)*2 ;
2887   IF (F < 0)
2888   THEN F := 0 ;
2889   HISTORY(120|1) := "." ;
2890   WHILE HISTORY(120|1) = "." AND F < COUNT
2891   DO BEGIN
2892     F := F + 1 ;
2893     MMTTS(-F, HISTORY) ;
2894   END ;
2895   DATER(4,0,DATE) ;
2896   IOCONTROL(3);
2897   WRITEON(DATE(0|8)) ;
2898   DATER(5,0,DATE) ;
2899   WRITEON("    ", DATE) ;
2900   IF (K = -1) AND ~WHO OR (K = 1) AND WHO
2901   THEN BEGIN
2902     WRITE(SPACES(0|12),"Awit","  ",SPACES,USERID) ;
2903     WRITE("REAL mins: ",COMPUTERTIME DIV 3600,SPACES, USERTIME
2904       DIV 3600) ;
2905     WRITE("CPU  secs: ",WITA DIV 60,SPACES,USER DIV 60) ;
2906     WRITE(NAME, SKILL) ;
2907   END ELSE
2908   BEGIN
2909     WRITE(SPACES(0|12),USERID,"  ",SPACES,"Awit") ;
2910     WRITE("REAL mins: ",USERTIME DIV 3600,SPACES, COMPUTERTIME
2911       DIV 3600) ;
2912     WRITE("CPU  secs: ",USER DIV 60,SPACES,WITA DIV 60) ;
2913     WRITE(SPACES, SKILL, NAME) ;
2914   END ;
2915   WRITE(" ") ;
2916   FOR I := F STEP 2 UNTIL COUNT
2917   DO BEGIN
2918     F := I ;
2919     MMTTS(-I,HISTORY) ;
2920     WRITE(HISTORY(85|54)) ;
2921   END ;
2922   IF COUNT > 0
2923   THEN MMTTS(-COUNT,HISTORY) ;
2924   IF (F < COUNT)
2925   THEN WRITE(HISTORY(85|30)) ;
2926   IF FINIS
2927   THEN PRINTSHORTBRD(-1) ;
2928   WRITE("NODES ",NOD, "TERMINAL ",TOT, "DYNAMIC ", DYN,"CAPTURE ",CAP)
2929     ;
2930   WRITE("SALVAGE ", SAL, "DUPLIC ", DUP, "RECLAIM ", REC,
2931     "BRANCH ",BRAN) ;
2932   WRITE("DUP_TREE ", DUP_S, "REC_TREE ", REC_S, "BRAN_SCORED ",
2933     BRAN_S) ;
2934   IOCONTROL(2) ;
2935   IF FINIS
2936   THEN PUT(TTYOUT) ;
2937   IF FINIS
2938   THEN MMTTS(-10009,HISTORY) ;
2939 END OFGAMERECORD ;
2940
```

```
@TITLE,"HIDDENLOSS(INTEGER VALUE N "
PROCEDURE HIDDENLOSS(INTEGER VALUE N ;
INTEGER VALUE RESULT LSAVE,ESAVE) ;
BEGIN
  INTEGER SQF, SQT, SQA, SQD, SQ3, I, K, L, PCA, PCD, SAVEB, PC3,
    PCF, E, KA, K3, REWARDN, KK ;
  LOGICAL T ;
  L := LSAVE ;
  E := ESAVE ;
  SQF := MOVEFROM(N) ;
  SQT := MOVETO(N) ;
  PCF := BRD(SQF) ;
  BRD(SQF) := EMPTY ;
  K := IF (PCF > 0)
       THEN 1
       ELSE -1 ;
  IF ~ REMEMBER AND ~ PPIN
  THEN IF (BRD(SQT) ~= NIL)
       THEN GO TO OMIT
       ELSE GO TO QUIT ;
  PINNED := TRUE ;
   COMMENT        direct pin and discovered attack ;

  FOR J := CON(-1,SQF) UNTIL CON(1,SQF)
  DO IF J ~= 0
     THEN BEGIN
       KK := IF J < 0
             THEN -1
             ELSE 1 ;
       SQA := CONTROL(SQF,J) ;
       PCA := BRD(SQA) ;
       KA := IF (PCA > 0)
             THEN 1
             ELSE -1 ;
       PCA := ABS(PCA) ;
       IF PCA >= BISHOP AND PCA <= QUEEN
       THEN BEGIN
         T := HIDDEN(SQA,SQF,SQ3,KK, FALSE) ;
         PC3 := BRD(SQ3) ;
         K3 := IF (PC3 > 0)
               THEN 1
               ELSE -1 ;
         IF (PC3 = KING*K) AND (KA ~= K3)
         THEN BEGIN
           IF ABS(DIR) = ABS(BOTV(EDGE, OFFSET(SQF)-OFFSET(SQT)))
           THEN PPIN := TRUE
           ELSE GO TO EXIT ;
               COMMENT  incremental move problem, use of ppin ensures
               that potential pin is detected even if incremental
               move is initially along the attack direction ;
         END ELSE IF (SQ3 ~= HOLE)
                  THEN IF (K3 ~= KA)
                       THEN BEGIN
                         COMMENT either sq3 contains a pinned piece,
                           or there is a discovered attack on sq3 ;
                         IF (K3 = K)
                         THEN I := L := L + 1
                         ELSE I := E := E - 1 ;
                         SQUARE(I) := SQ3 ;
                             COMMENT IF DEBUG
                             THEN WRITE("HIDD",SQF,SQT,SQ3,KA,K3) ;
         END ;
       END ;
  END ;
OMIT:
  PINNED := FALSE ;
   COMMENT     indirect pin and captured piece loss ;
  REWARDN := ABS(REWARD(N)) REM CHECKING ;
  FOR KK := -K,K
  DO BEGIN
```

```
3011      SQ3 := IF (K=KK)
3012             THEN SQF
3013             ELSE IF (REWARDN = ENPRIS)
3014                  THEN SQT - K*FILE
3015                  ELSE SQT ;
3016      IF (SQ3 ~= SQF) OR TERMINAL OR REMEMBER
3017      THEN WHILE (SQ3 ~= 0)
3018           DO BEGIN
3019             FOR J := POSITION(SQ3,LIM) UNTIL LIM-1
3020             DO BEGIN
3021               SQD := POSITION(SQ3,J) ;
3022               IF (KK*BRD(SQD) > 0) AND (CON(-KK,SQD) ~= 0)
3023               THEN IF (K=KK)
3024                    THEN BEGIN
3025                      L := L + 1 ;
3026                      SQUARE(L) := SQD ;
3027               END ELSE
3028               BEGIN
3029                 E := E - 1 ;
3030                 SQUARE(E) := SQD ;
3031               END ;
3032             END ;
3033             SQ3 := IF (K=KK) AND (ABS(SQT - SQF) = 2) AND (ABS(PCF)
3034               = KING) AND (SQ3 = SQF)
3035                    THEN IF (REWARDN = QSIDE)
3036                         THEN QRSQ(K)
3037                         ELSE IF (REWARDN = KSIDE)
3038                              THEN KRSQ(K)
3039                              ELSE 0
3040                    ELSE 0 ;
3041      END ;
3042    END ;
3043    IF ~TERMINAL AND ~REMEMBER
3044    THEN GO TO QUIT ;
3045     COMMENT       unpinning piece ;
3046    IF (ABS(PCF) > KNIGHT) AND (ABS(PCF) < KING)
3047    THEN FOR I := 1 UNTIL POS(SQF)
3048         DO BEGIN
3049           SQA := POSITION(SQF,I) ;
3050           IF (K*BRD(SQA) < 0)
3051           THEN BEGIN
3052             T := HIDDEN(SQF,SQA,SQ3,K, FALSE) ;
3053             PC3 := BRD(SQ3) ;
3054             IF (K*PC3 < 0) AND (SQT ~= SQA)
3055             THEN IF (ABS(PCF) < ABS(PC3))
3056                  THEN
3057                  COMMENT  possible unpin ;
3058                  IF DIR ~= BOTV(EDGE, OFFSET(SQT)-OFFSET(SQA))
3059                  THEN
3060                  COMMENT   definite unpin ;
3061                  FOR J := POS(SQA) STEP -1 UNTIL 1
3062                  DO BEGIN
3063                    SQD := POSITION(SQA,J) ;
3064                    IF (K*BRD(SQD) > 0)
3065                    THEN BEGIN
3066                      L := L+1 ;
3067                      SQUARE(L) := SQD ;
3068                    END ;
3069             END ;
3070           END ;
3071    END ;
3072  QUIT:
3073    LSAVE := L ;
3074    ESAVE := E ;
3075     COMMENT       blocking a defence ;
3076    FOR KK := -K,K
3077    DO BEGIN
3078      I := CON(KK,SQT) ;
3079      IF (ABS(PCF) = KNIGHT)
3080      THEN IF (ABS(I) < CONLIM)
```

```
3081            THEN BEGIN
3082              PC3 := HOLE ;
3083              DIR := BOTV(EDGE, OFFSET(SQT)-OFFSET(SQF)) ;
3084              IF DIR = 0
3085              THEN PC3 := NIL
3086              ELSE FOR J := SQT+DIR STEP DIR UNTIL SQF-DIR
3087              DO IF (BRD(J) ~= EMPTY)
3088                 THEN PC3 := NIL ;
3089              IF (PC3 ~= NIL)
3090              THEN IF HIDDEN(SQT,SQF,SQ3,-KK, FALSE)
3091                   THEN BEGIN
3092                     I := I + KK ;
3093                     CONTROL(SQT,I) := SQ3 ;
3094              END ;
3095                 COMMENT special order of incremental knight
3096                 moves is used to minimize problem associated
3097                 with the blocking of a mutual defence between
3098                 bishop and queen ;
3099       END ELSE WRITE("HIDDEN CONLIM ERROR") ;
3100       FOR I := I STEP -KK UNTIL KK
3101       DO BEGIN
3102         SQD := CONTROL(SQT,I) ;
3103         PCD := ABS(BRD(SQD)) ;
3104         IF (PCD = QUEEN) OR (PCD = ROOK) OR (PCD = BISHOP)
3105         THEN IF (PCF ~= BRD(SQD))
3106              THEN BEGIN
3107                T := HIDDEN(SQD,SQT,SQ3,-KK, FALSE) ;
3108                IF (KK*BRD(SQ3) > 0)
3109                THEN IF (K = KK)
3110                     THEN BEGIN
3111                       L := L+1 ;
3112                       SQUARE(L) := SQ3 ;
3113                END ELSE
3114                BEGIN
3115                  E := E - 1 ;
3116                  SQUARE(E) := SQ3 ;
3117                END ;
3118         END ;
3119       END ;
3120     END ;
3121     FOR I := SQUARE(0)+1 UNTIL L
3122     DO LOSS(I) := 0 ;
3123     LOSS(0) := L ;
3124     LOSS(ELIMIT) := E ;
3125   EXIT:
3126     BRD(SQF) := PCF ;
3127   END OFHIDDENLOSS ;
3128
3129   @TITLE,"MOVESQUARE(INTEGER VALUE I, DIRECT) "
3130   PROCEDURE MOVESQUARE(INTEGER VALUE SQ) ;
3131   COMMENT   lists the moves for the piece on square i ;
3132   BEGIN
3133     INTEGER K, M1 ;
3134      COMMENT   movesquare is called by many procedures ;
3135     M1 := BRD(SQ) ;
3136     K := IF (M1 < 0)
3137          THEN -1
3138          ELSE 1 ;
3139     M1 := ABS(M1) ;
3140     SLIST := LIM ;
3141     CLIST := 0 ;
3142      COMMENT for King and Pawn moves, KLIST set in LISTK...
3143              and LISTP... ;
3144     IF (M1 = KING)
3145     THEN LISTKINGMOVES(SQ, K)
3146     ELSE IF M1 = PAWN
3147          THEN LISTPAWNMOVES(SQ, K)
3148     ELSE BEGIN
3149       IF M1 = KNIGHT
3150       THEN TRYKNIGHTMOVE(SQ, K)
```

```
3151      ELSE IF (M1 = QUEEN)
3152           THEN LISTQUEENMOVES(SQ, K)
3153           ELSE IF (M1 = BISHOP)
3154                THEN TRYMINORMOVE(SQ, LDIAG, RDIAG, K)
3155                ELSE IF M1 = ROOK
3156                     THEN TRYMINORMOVE(SQ, FILE, RANK, K) ;
3157      KLIST := CLIST ;
3158    END ;
3159    CHECKLIST(0) := CLIST ;
3160    CHECKLIST(LIM) := SLIST ;
3161  END OFMOVESQUARE ;
3162
3163  @TITLE,"PASSED PAWN"
3164  INTEGER PROCEDURE PASSEDPAWN(INTEGER VALUE SQ, KK) ;
3165  BEGIN
3166    INTEGER VAL, LIMIT, INC, SQ3, A, PC, C, SQ2, DP, DK ;
3167    LOGICAL FREE, STOPPED ;
3168    VAL := 0 ;
3169    IF ~OPENING AND PAWNS(SQ) = -KK
3170    THEN GO TO QUIT ;
3171    INC := FILES(KK) ;
3172    LIMIT := IF (KK > 0)
3173             THEN BKRSQ
3174             ELSE WQRSQ ;
3175     COMMENT increase VAL if passed pawn, and ENDGAME ;
3176    FOR L := -1 UNTIL 1
3177    DO FOR I := SQ + INC + L STEP INC UNTIL LIMIT
3178    DO IF (BRD(I) = -KK)
3179       THEN GO TO QUIT ;
3180    VAL := 8 ;
3181    FREE := TRUE ;
3182    SQ2 := SQ -INC;
3183    IF BRD(SQ) ~= 0
3184    THEN SQ2 := SQ
3185    ELSE IF ABS(BRD(SQ2)) ~= PAWN
3186         THEN SQ2 := LIMIT;
3187    A := 0;
3188    IF SQ2 = LIMIT
3189    THEN SQ2 := SQ
3190    ELSE IF HIDDEN(SQ+INC, SQ2, SQ3, 0, FALSE)
3191         THEN A := IF KK*BRD(SQ3) > 0
3192                   THEN 1
3193                   ELSE -1;
3194    IF SQ2 ~= SQ AND KK*SEC(0,SQ) + A < 0
3195    THEN FREE := FALSE ;
3196    FOR I := SQ+INC STEP INC UNTIL LIMIT
3197    DO BEGIN
3198      IF FREE
3199      THEN BEGIN
3200        PC := KK*BRD(I) ;
3201        C := KK*CON(0,I) ;
3202        IF C + A < 0 OR PC < 0 AND C <= 0
3203        THEN FREE := FALSE ;
3204        IF PC < ROOK OR C <= 0
3205        THEN IF PC > 0
3206             THEN VAL := VAL -1 ;
3207      END ;
3208      VAL := VAL -1 ;
3209    END ;
3210    IF ~FREE
3211    THEN VAL := VAL -1 ;
3212    IF VAL < 4
3213    THEN VAL := VAL -1 ;
3214    SQ2 := SQ2 + INC;
3215    STOPPED := BRD(SQ2) ~= EDGE AND (BRD(SQ2)*KK < 0 OR CON(KK,SQ2) = 0
3216                                AND CON(-KK,SQ2) ~= 0);
3217    IF DEBUG
3218    THEN WRITE("PASSED", SQ, FREE, A, C, VAL, KK, SQ2, STOPPED) ;
3219  IF ~ STOPPED THEN BEGIN
3220     COMMENT  can King stop pawn?;
```

```
3221   DP := BOTV(KING, OFFSET(SQ)-OFFSET(BACK_ROW(-KK,COLS(SQ)))) ;
3222   DK := BOTV(KING,OFFSET(KINGSQ(-KK))-OFFSET(BACK_ROW(-KK,COLS(SQ)))) ;
3223   STOPPED := DP + (IF K = KK
3224                    THEN 1
3225                    ELSE 0) >= DK ;
3226 END ELSE IF VAL > 2 THEN VAL := VAL -2;
3227   IF STOPPED
3228   THEN BEGIN
3229     IF VAL > 3
3230     THEN IF CON(KK,SQ) ~= 0 OR CON(KK,SQ+INC) ~= 0
3231          THEN VAL := 1 + (VAL DIV 2)
3232          ELSE VAL := 3 ;
3233   END ELSE IF FREE
3234             THEN VAL := VAL + 3 ;
3235   IF FREE
3236   THEN BEGIN
3237     IF A = 1
3238     THEN VAL := VAL +1
3239     ELSE IF BRD(SQ) ~= 0
3240          THEN VAL := VAL +1 ;
3241   END ELSE IF BRD(SQ) ~= 0 AND VAL >= 6 AND ABS(CON(-KK,SQ+INC)) <= 1
3242             THEN IF A = 1
3243                  THEN VAL := VAL + 3 ;
3244   IF ~STOPPED AND VAL >= 10
3245   THEN VAL := VAL + 2 ;
3246   IF DEBUG
3247   THEN WRITEON(STOPPED, DP, DK, K, VAL,CON(-KK,SQ+INC)) ;
3248 QUIT:
3249   VAL
3250 END OFPASSEDPAWN ;
3251
3252 @TITLE,"REMAKE "
3253 PROCEDURE REMAKE(INTEGER VALUE SQF, SQT) ;
3254 BEGIN
3255   INTEGER PC, SQ1, SQ2, L1, L2, I1, KK ;
3256   LOGICAL FLAG ;
3257   PC := BRD(SQF) ;
3258   IF MAKEMV
3259   THEN BEGIN
3260     FOR L := 2 UNTIL NUMB
3261     DO IF (SQF = RLIST(L))
3262        THEN IF (ABS(PC) = PAWN)
3263             THEN GO TO SET
3264             ELSE IF L > T_NUMB
3265                  THEN GO TO QUIT ;
3266     NUMB := NUMB+1 ;
3267     RLIST(NUMB) := SQF ;
3268     TLIST(NUMB) := SQT ;
3269   END ;
3270 SET:
3271   KK := IF (PC > 0)
3272         THEN 1
3273         ELSE -1 ;
3274   L1 := POSITION(SQF,0) ;
3275   FOR L := 1 UNTIL L1
3276   DO OLDLIST(L) := POSITION(SQF,L) ;
3277    COMMENT pawns do not control all the squares to which they can
3278            move ;
3279    COMMENT kings cannot move to all the squares that they control,
3280            nor do they control the castling square ;
3281   FOR L := POSITION(SQF,LIM) UNTIL LIM-1
3282   DO BEGIN
3283     L1 := L1 + 1 ;
3284     OLDLIST(L1) := POSITION(SQF,L) ;
3285   END ;
3286   MOVESQUARE(SQF) ;
3287    COMMENT note use of KLIST, and not POS(SQF) ;
3288   FOR L := 1 UNTIL CLIST
3289   DO POSITION(SQF,L) := CHECKLIST(L) ;
3290   FOR L := SLIST UNTIL LIM
```

```
3291    DO POSITION(SQF,L) := CHECKLIST(L) ;
3292    POS(SQF) := CLIST ;
3293    POSITION(SQF,0) := I1  := KLIST ;
3294    FOR I := SLIST UNTIL LIM-1
3295    DO BEGIN
3296      I1 := I1 + 1 ;
3297      CHECKLIST(I1) := CHECKLIST(I) ;
3298    END ;
3299     COMMENT delete unique entries in OLDLIST, fix in CHECKLIST ;
3300    FOR L := 1 UNTIL L1
3301    DO BEGIN
3302      SQ1 := OLDLIST(L) ;
3303      FOR I := 1 UNTIL I1
3304      DO IF (CHECKLIST(I) = SQ1)
3305         THEN BEGIN
3306           CHECKLIST(I) := CHECKLIST(I1) ;
3307           I1 := I1 -1 ;
3308           GO TO NEXTL ;
3309      END ;
3310      DELETE(SQF,SQ1,KK) ;
3311  NEXTL:
3312    END ;
3313     COMMENT does not handle correctly, pawns which are about to
3314     promote with a capture.  not serious? ;
3315    L2 := LOSS(ELIMIT) ;
3316    FOR I := 1 UNTIL I1
3317    DO BEGIN
3318      SQ2 := CHECKLIST(I) ;
3319      FIXIT(SQF,SQ2,KK) ;
3320      IF REMEMBER
3321      THEN IF (K*BRD(SQ2) < 0)
3322           THEN BEGIN
3323             L2 := L2 -1 ;
3324             SQUARE(L2) := SQ2 ;
3325      END ;
3326    END ;
3327    LOSS(ELIMIT) := L2 ;
3328  QUIT:
3329  END OFREMAKE ;
3330  @TITLE,"CAPT_ORDER AND SELECTRTMV"
3331  PROCEDURE CAPT_ORDER(INTEGER RESULT BEST ;
3332  LOGICAL VALUE LOSING) ;
3333  BEGIN
3334    LOGICAL LEGAL ;
3335    INTEGER NOSC, DEFN, PC_VAL, ATTA_VAL, ATTA;
3336    INTEGER R, PC, CH, TO, SQF, SQA, SAVE_D, KSQ, FROM,DIS, TMP, VAL ;
3337    NOSC := IF LOSING
3338            THEN -99
3339            ELSE NOSCR ;
3340    KSQ := KINGSQ(K) ;
3341    BEST := QUEEN ;
3342    SAVE_D := IF INCHECK AND ~DBLCHECK AND CHECKSQ ~= HOLE
3343              THEN  BOTV(EDGE, OFFSET(KSQ)-OFFSET(CHECKSQ))
3344              ELSE 0 ;
3345    IF ABS(BRD(CHECKSQ)) = KNIGHT
3346    THEN SAVE_D := CHECKSQ -KSQ ;
3347    FOR N := NUMBER(K) STEP -K UNTIL K
3348    DO BEGIN
3349      R := REWARD(N) ;
3350      FROM := MOVEFROM(N) ;
3351      TO := MOVETO(N) ;
3352      PC := ABS(BRD(FROM)) ;
3353      IF (PC = PAWN) AND (PAWNS(TO) = 0)
3354      THEN PC := ABS(R) REM PROMOTE ;
3355      TMP := K*CHECK_S(-K,TO) ;
3356      CH := IF TMP > 0 AND CHEC(TMP, PC)
3357            THEN REALCHECK
3358            ELSE 0 ;
3359      IF HIDDEN(KINGSQ(-K), FROM, SQA, K, TRUE)
3360      THEN BEGIN
```

```
3361       DIS := DISCHECK ;
3362       IF DIR = BOTV(EDGE, OFFSET(KINGSQ(-K))-OFFSET(TO))
3363       THEN DIS := 0
3364       ELSE FOR I := KINGSQ(-K) +DIR STEP DIR UNTIL FROM -DIR
3365       DO IF BRD(I) ~= NIL
3366          THEN DIS := 0 ;
3367       CH := IF CH = 0
3368             THEN DIS
3369             ELSE IF DIS = 0
3370                  THEN CH
3371                  ELSE TWOCHECK ;
3372          COMMENT IF TEST THEN WRITE("DIS_CAP",KINGSQ(-K), FROM,TO,
3373          DIR,CH) ;
3374     END ;
3375     IF (CH ~= 0)
3376     THEN REWARD(N) := R + CH ;
3377     LEGAL := FALSE ;
3378 IF PC = KING AND CON(-K,TO) ~= 0
3379 THEN BEGIN
3380   PRINTSHORTBRD(-1);
3381   PRINTLINE(N,0);
3382 END ELSE
3383     IF (PC = KING AND R~=KSIDE AND R~=QSIDE)
3384     THEN LEGAL := TRUE
3385     ELSE IF ~DBLCHECK
3386          THEN BEGIN
3387            IF ~HIDDEN(KSQ, FROM, SQA, -K, TRUE)
3388            THEN LEGAL := TRUE
3389            ELSE BEGIN
3390              IF ABS(DIR) = ABS(BOTV(EDGE, OFFSET(FROM)-OFFSET(TO)))
3391              THEN LEGAL := TRUE ;
3392            END ;
3393            IF INCHECK AND LEGAL
3394            THEN BEGIN
3395              LEGAL := FALSE ;
3396              FOR SQ := KSQ+SAVE_D STEP SAVE_D UNTIL CHECKSQ
3397              DO IF SQ = TO
3398                 THEN LEGAL := TRUE ;
3399            END ;
3400     END ;
3401     IF ~LEGAL
3402     THEN IF (R = ENPRIS) AND (CHECKSQ = TO -K*FILE)
3403          THEN LEGAL := TRUE
3404     ELSE BEGIN
3405       R := BAD ;
3406       NUM := NUM -1 ;
3407     END ;
3408     DEFN := K*SEC(0,TO) ;
3409 COMMENT only legal replies generated ;
3410     IF LEGAL THEN BEGIN
3411       ATTA := CON(-K,TO);
3412       PC_VAL := VALUES(PC);
3413     COMMENT IF TEST
3414         THEN WRITE("CAPT", PC, R, DEFN, ATTA, PC_VAL, TO, K);
3415     IF ATTA ~= 0 THEN
3416       ATTA_VAL := VALUES(ABS(BRD(CONTROL(TO,-K))));
3417     COMMENT IF TEST THEN WRITEON(ATTA_VAL);
3418               ATTA := ABS(ATTA);
3419       IF R = 0 OR R > ENPRIS AND R < PROMOTE + QUEEN
3420                        + QUEEN
3421       THEN R := IF (CH ~= 0)
3422                 THEN IF DEFN <= 0
3423                      THEN -PC_VAL
3424                      ELSE -PC
3425                 ELSE IF INCHECK
3426                      THEN -8      +PC
3427                      ELSE IF ATTA = 1
3428                           THEN IF DEFN < 0 OR DEFN = 0 AND PC ~= PAWN
3429                                THEN -PC_VAL
3430                                ELSE IF ATTA_VAL < PC_VAL - DELTA
```

```
                                        THEN -PC_VAL + ATTA_VAL
                                        ELSE NOSC
                              ELSE NOSC
       ELSE IF R ~= 0 AND R < KING
            THEN BEGIN
              VAL := IF DEFN <= 0 OR ATTA > 1
                     THEN 0
                     ELSE IF ATTA = 1
                          THEN  ATTA_VAL
                          ELSE PC_VAL ;
              R := VALUES(R) - PC_VAL + VAL ;
            END ;
       COMMENT IF TEST AND R ~= NOSC
       THEN WRITEON(R);
     END ;
     IF PC = PAWN AND R = NOSC
     THEN DEFN := DEFN + 1 ;
     IF LOSING AND R ~= BAD AND DEFN <= 0
     THEN R := R + NOSC ;
     SCORE(N) := R ;
   END ;
 END OFCAPT_ORDER ;

 @TITLE, "MAIN_PROCEDURE"
 PROCEDURE MAIN ;
 BEGIN
   INTEGER ARRAY LASTSQ, DEBIT, CREDIT(-1::MAX_PLIES) ;
   INTEGER ARRAY STACK_HASH(0::MAX_PLIES) ;
   INTEGER ARRAY NODETO(0::MAXTREE, 0::MAX_WID) ;
   INTEGER ARRAY GETNODE, PARENT, DUPSCR(0::MAXTREE+1) ;
   INTEGER ARRAY ROOTNODE(0::MAX_PLIES) ;
   INTEGER ARRAY BACK(0::MAX_PLIES, 0::TSIZE) ;
   INTEGER ARRAY STATES(-1::MAXTREE,1::2*MAX_WID+T_SIZE+2) ;
   INTEGER ARRAY REFUT, REFCNT (1::MAX_PLIES, 0::10) ;
   INTEGER ARRAY LASTDIR (-1::MAX_PLIES) ;
   INTEGER ARRAY REFTAB(1::MAX_WID+MAX_PLIES, 1::MAX_PLIES);

 PROCEDURE COMPUTER ;
 COMMENT makes move which has largest score ;
 BEGIN
   SCORES := TRUE ;
   IF (BOOK = -1)
   THEN USEBOOK ;
   IF (BOOK = 0)
   THEN IF (LOOKAHEAD) AND (MOVECOUNT > 1)
        THEN SEARCH
        ELSE LISTLEGALMOVES (FALSE) ;
   IF (SKTEST = 3)
   THEN IF (SCORE(RTMV) > PAWNVALUE)
        THEN WRITE("CANNOT ACCEPT DRAW YET")
        ELSE KTEST := 3 ;
   IF (NUM = 0)
   THEN KTEST := IF (CON(-K,KINGSQ(K)) ~= 0)
                 THEN 4
                 ELSE 2 ;
    COMMENT if stalemate, reply = 0, do not generate response;
   IF ~EARLY AND REPLY = 0 AND BOOK = 0 AND KTEST = 0
   THEN NULLMOVE := TRUE
   ELSE IF (KTEST ~= 3)
        THEN BEGIN
          S_W := 0 ;
          WRITE(MOVECOUNT,". ") ;
          S_W := 1 ;
          DESCRIBEMOVE(RTMV, MC) ;
          IF K ~= 1
          THEN WRITEON("  ... ") ;
          FOR I := 0 UNTIL 23
          DO WRITEON(MC(I)) ;
          WRITEON(BELL) ;
          IF COKO AND FALSE
```

```
3501          THEN BEGIN
3502            COMMENT switch i-o units ;
3503            PUT(7) ;
3504          COMMENT switch o-p to unit 7 ;
3505            WRITE(" ") ;
3506            FOR I := 0 UNTIL 23
3507            DO WRITEON(MC(I)) ;
3508            IOCONTROL(2) ;
3509            PUT(6) ;
3510          COMMENT restore o-p to unit 6 ;
3511          END ;
3512          IF (SCORE(RTMV) >  + VALUES(QUEEN)*SCALE)
3513          THEN WRITE("I WILL ACCEPT YOUR RESIGNATION") ;
3514    END ELSE WRITE("ERROR.  NUM =",NUM, REASON(KTEST)) ;
3515  END OFCOMPUTER ;
3516  @TITLE,"FINALIZE"
3517  PROCEDURE FINALIZE ;
3518  BEGIN
3519    INTEGER I, NODE, D, CT ;
3520    PUNCHBRD(TRUE, HISTORY) ;
3521    I := IF (K = 1)
3522         THEN 85
3523         ELSE 115 ;
3524    FOR J := 0 UNTIL 23
3525    DO HISTORY(I+J | 1) := MC(J) ;
3526    I := 110 -15*K ;
3527    D := SCORE(RTMV) + (IF WHO THEN -200 ELSE +200) ;
3528    IF RTMV ~= 0 AND PREDICT > 0
3529    THEN IF ENDGAME AND ABS(D) <= THRESHOLD
3530         THEN BEGIN
3531           HISTORY(I+2|4) := REASON(3)(1|4) ;
3532           IF WHO AND (MOVECOUNT > 30)
3533           THEN WRITEON(".DRAW?") ;
3534    END ELSE IF (SCORE(RTMV) < -9876)
3535              THEN BEGIN
3536                HISTORY(I|6) := REASON(1)(0|6) ;
3537                IF WHO AND (QUICK OR ~TOURNAMENT)
3538                THEN KTEST := 1 ;
3539    END ;
3540    NODE := IF WHO
3541            THEN LASTORIGIN
3542            ELSE ORIGIN ;
3543    I := I + 8 ;
3544    HISTORY(I|1) := IF TRY(WSIZE) = 1
3545                    THEN "F"
3546                    ELSE IF  BOOK ~= 0
3547                         THEN "B"
3548                         ELSE IF NODETO(NODE,0) <= 0
3549                              THEN "R"
3550                              ELSE CODE((NODETO(NODE,0) REM 10) +240) ;
3551    IF HISTORY(I|1) ~= "R"
3552    THEN BEGIN
3553      HISTORY(I-2|1) := IF BOOK = 0 AND ~FORCEDREPLY AND PREDICT <= 0
3554                        THEN "#"
3555                        ELSE IF PREDICT >= 10
3556                             THEN "1"
3557                             ELSE IF NODETO(NODE,0) > 9
3558                                  THEN "."
3559                                  ELSE " " ;
3560      PREDICT := PREDICT REM 10 ;
3561      HISTORY(I-1|1) := IF ~FORCEDREPLY
3562                        THEN CODE((ABS(PREDICT) REM 10)+240)
3563                        ELSE " " ;
3564      IF ~FORCEDREPLY
3565      THEN IF WHO OR HISTORY(I-2|1) ~= "#"
3566           THEN HISTORY(I+1|1) := IF ABS(D) > 3600
3567                                  THEN IF D > 0
3568                    THEN "$"
3569                    ELSE "*"
3570                    ELSE IF ABS(D) > 2100
```

```
3571                          THEN IF D > 0
3572                               THEN ">"
3573                               ELSE "<"
3574                               ELSE IF D > 600
3575                                    THEN "+"
3576                                    ELSE IF D < -300
3577                     THEN "-"
3578                     ELSE "=" ;
3579    END ;
3580    HISTORY(I+2|1) := IF EARLY OR ENDGAME AND ~LATE_END
3581                       THEN "E"
3582                       ELSE IF OPENING
3583                            THEN "O"
3584                            ELSE IF LATE_END
3585                                 THEN "L"
3586                                 ELSE "M" ;
3587    FORCEDREPLY := WHO AND TRY(WSIZE) = 1 ;
3588    CPU := TIME(1) - OLDCPU ;
3589    ELAPSEDTIME := TIME(-1) - SENDTIME ;
3590    IF (ELAPSEDTIME < 0)
3591    THEN ELAPSEDTIME := ELAPSEDTIME + 5184000 ;
3592  COMMENT midnight correction ;
3593    CTIME := IF WHO
3594             THEN CPU
3595             ELSE ELAPSEDTIME ;
3596    CT := CTIME := (CTIME + 30) DIV 60 ;
3597  COMMENT in secs. ;
3598    I := I + 5 ;
3599    WHILE (CTIME > 0)
3600    DO BEGIN
3601      HISTORY(I|1) := CODE((CTIME REM 10) +240) ;
3602      I := I -1 ;
3603      CTIME := CTIME DIV 10 ;
3604    END ;
3605    IF ~WHO AND (BOOK = 0) AND (MOVECOUNT < 16 OR ~ LOOKAHEAD)
3606    THEN BOOK := -1 ;
3607    IF ~WHO AND (KTEST = 3)
3608    THEN KTEST := 0 ;
3609    IF (KTEST <= 1 OR KTEST = 2 OR KTEST = 4) AND (RTMV ~= 0)
3610    THEN BEGIN
3611      IF (K = -1)
3612      THEN BEGIN
3613        MOVECOUNT := MOVECOUNT + 1 ;
3614        MOVES_LEFT := MOVES_LEFT -1 ;
3615      END ;
3616      COUNT := COUNT + 1 ;
3617      MMTTS(COUNT,HISTORY) ;
3618      MMTTS(-10008, HISTORY) ;
3619      COMMENT free the history file ;
3620      MAKEMOVE(RTMV,TRUE) ;
3621      IF  ENDGAME
3622      THEN FOR I := WQRSQ UNTIL BKRSQ
3623      DO IF K*BRD(I) = PAWN
3624         THEN IF PASSEDPAWN(I, K) > 0
3625              THEN BEGIN
3626                CLEARPAWN(K) := I ;
3627                IF K < 0
3628                THEN GO TO DONE ;
3629                COMMENT find location of most advanced passed pawn ;
3630      END ;
3631  DONE:
3632      K := -K ;
3633  COMMENT  switchsides. ;
3634      IF (CON(K,KINGSQ(-K)) ~= 0)
3635      THEN BEGIN
3636        WRITE("KING IN CHECK, ILLEGAL") ;
3637        LEGAL := WHO :=  FALSE ;
3638        TRACER(9, "?DPWPBPX18") ;
3639        REINITIALIZE("0") ;
3640      END ELSE
```

```
3641      BEGIN
3642        ENDPLAY := IF ~REVERSIBLE
3643                   THEN 1
3644                   ELSE ENDPLAY +1 ;
3645        IF ENDPLAY > 100
3646        THEN BEGIN
3647          IF REPEATS(0) ~= -12345
3648          THEN REPEATS(0) := -12345
3649          ELSE KTEST := 3 ;
3650          ENDPLAY := 1 ;
3651        END ;
3652        REPEATS(ENDPLAY) :=  HASH ;
3653        NEGA(ENDPLAY) := HISTORY(IF K = 1 THEN 115 ELSE 85 | 4);
3654        D := 0 ;
3655        FOR J := 1 UNTIL ENDPLAY
3656        DO IF (HASH = REPEATS(J))
3657           THEN D := D + 1 ;
3658        IF (D > 2)
3659        THEN KTEST := 3 ;
3660        IF CASTLING
3661        THEN CASTLE(0) := ~ CASTLE(0) ;
3662        SENDTIME := SENDTIME + ELAPSEDTIME ;
3663        OLDCPU := OLDCPU + CPU ;
3664        IF WHO
3665        THEN BEGIN
3666          WITA := WITA + CPU ;
3667          WRITEON(CPU DIV 60, WITA DIV 60) ;
3668          COMPUTERTIME := COMPUTERTIME +ELAPSEDTIME ;
3669          CTIME := COMPUTERTIME DIV (MOVECOUNT*60) ;
3670          TIME_LEFT := TIME_LEFT - ELAPSEDTIME ;
3671          IF CPU = 0
3672          THEN CPU := 1 ;
3673          IF BOOK <= 0  AND SKILL > "  BEGIN" OR MONITOR
3674          THEN BEGIN
3675            WRITE("CPU ELAPSE ELA/MOVE LEFT LIMBS/SEC") ;
3676            WRITE(CT, ELAPSEDTIME DIV 60, CTIME, TIME_LEFT DIV 3600,
3677              MOVES_LEFT+1, (LIMBS*60) DIV CPU, HISTORY(135|1)) ;
3678          END ;
3679        END ELSE
3680        BEGIN
3681          USER := USER + CPU ;
3682          USERTIME := USERTIME + ELAPSEDTIME ;
3683        END ;
3684        IOCONTROL(2) ;
3685        WHO := IF CBOTH
3686               THEN TRUE
3687               ELSE IF BOTH
3688                    THEN FALSE
3689                    ELSE ~ WHO ;
3690        IF (K = 1) AND (BOOK ~= 1) AND ((MOVECOUNT-1) REM 10 = 0)
3691          AND (SKILL > "  BEGINNER")
3692        THEN GAMERECORD(TRUE) ;
3693        IF (K = 1) AND MOVES_LEFT < 0
3694        THEN MOVES_LEFT := MOVES_LEFT +20 ;
3695      END ;
3696    END ELSE WRITE("INVALID MOVE ", REASON(KTEST), RTMV, PREDICT) ;
3697  END OFFINALIZE ;
3698
3699  @TITLE,"MODOK(INTEGER ARRAY CM(*) "
3700  PROCEDURE FINDRTMV ;
3701   COMMENT finds which move on the list corresponds to the input move ;
3702  BEGIN
3703    INTEGER FROMSQ, TOSQ, FROMPC, TOPC ;
3704    STRING(140) BUFFER;
3705
3706    LOGICAL PROCEDURE MODOK(INTEGER ARRAY CM(*) ;
3707    INTEGER VALUE TM) ;
3708  COMMENT checks modifier of chess notation, eg.,  distinguishes
3709  B from BP from KBP from QBP,  from KN etc. ;
3710    BEGIN
```

```
3711      INTEGER J, COL ;
3712      LOGICAL B, NOTKQ ;
3713      COL := COLS(TM) ;
3714      B := TRUE ;
3715      IF (CM(1) = 0)
3716      THEN GO TO QUIT ;
3717      J := 1 ;
3718      NOTKQ  := FALSE ;
3719     COMMENT q?p ;
3720      IF (CM(0) ~= QUEEN)
3721      THEN IF (CM(0) = KING)
3722           THEN J := -8
3723           COMMENT k?p ;
3724           ELSE IF (CM(1) = KING)
3725                THEN BEGIN
3726                  B := FALSE ;
3727                  IF (CM(2) = PAWN)
3728                  THEN B := IF (COL = 5)
3729                            THEN TRUE
3730                            ELSE FALSE
3731                  ELSE FOR J := +8, +7, +6, 5
3732                  DO IF (COL = J)
3733                     THEN B := TRUE ;
3734                  GO TO QUIT ;
3735      END ELSE IF (CM(1) = QUEEN)
3736                THEN BEGIN
3737                  B := FALSE ;
3738                  IF (CM(2) = PAWN)
3739                  THEN B := IF (COL = 4)
3740                            THEN TRUE
3741                            ELSE FALSE
3742                  ELSE FOR J := 4,  3, 2, 1
3743                  DO
3744                  COMMENT which q? ;
3745                  IF (COL = J)
3746                  THEN B := TRUE ;
3747                  GO TO QUIT ;
3748      END ELSE NOTKQ := TRUE ;
3749                           COMMENT ?p ;
3750      J := ABS(J + (IF CM(1) = ROOK
3751                    THEN 0
3752                    ELSE IF CM(1) = KNIGHT
3753                         THEN 1
3754                         ELSE IF CM(1) = BISHOP
3755                              THEN 2
3756                              ELSE -J)) ;
3757      IF (COL = J)
3758      THEN GO TO QUIT
3759      ELSE IF NOTKQ AND (COL = 9-J)
3760           THEN GO TO QUIT
3761           ELSE B := FALSE ;
3762  QUIT:
3763      B
3764    END OFMODOK ;
3765
3766  @TITLE,"FINDRTMV"
3767    RTMV := 0 ;
3768    FOR N := NUMBER(K) STEP -K UNTIL K
3769    DO BEGIN
3770      FROMSQ := MOVEFROM(N) ;
3771      TOSQ := MOVETO(N) ;
3772      FROMPC := ABS(BRD(FROMSQ)) ;
3773      TOPC := ABS(BRD(TOSQ)) ;
3774      IF (FROMSQ ~= EMPTY)
3775      THEN BEGIN
3776        IF (FMAN(2) = 0)
3777        THEN IF (ABS(REWARD(N)) REM CHECKING = REW(1))
3778             THEN IF (REW(1) ~= 0)
3779                  THEN BEGIN
3780                    RTMV := N ;
```

```
3781                    COMMENT castling move ;
3782                   GO TO QUIT ;
3783       END ELSE GO TO NEXTN ;
3784       IF (FMAN(2) ~= FROMPC)
3785       THEN GO TO NEXTN ;
3786       COMMENT a minor modification would allow this procedure to take
3787       advantage of the fact that the moves corresponding to a single
3788       piece are in a block ;
3789       IF (FSQ(0) ~= 0)
3790       THEN IF (FROMSQ ~= FSQ(0)) AND ( FROMSQ ~= FSQ(1))
3791            THEN GO TO NEXTN ;
3792       IF (TMAN(2) ~= TOPC)
3793       COMMENT for enpassant capture must write p*p ep. ;
3794 THEN GO TO NEXTN
3795 ELSE IF (TOPC = ENPRIS)
3796      THEN BEGIN
3797        IF (RTMV ~= 0)
3798        THEN GO TO AMBIG ;
3799        RTMV := N ;
3800        GO TO NEXTN ;
3801       END ;
3802       IF (TSQ(0) ~= 0)
3803       THEN IF (TOSQ ~= TSQ(0)) AND (TOSQ ~= TSQ(1))
3804            THEN GO TO NEXTN ;
3805       IF (REW(2) ~= 0)
3806       THEN IF (REW(2) ~= ABS (REWARD(N)) REM CHECKING)
3807            THEN GO TO NEXTN ;
3808       IF (MODOK (TMAN,TOSQ))
3809       THEN IF (MODOK (FMAN,FROMSQ))
3810            THEN BEGIN
3811              COMMENT check specified? ;
3812                   PUNCHBRD(TRUE, BUFFER);
3813              BASE := MOVEFROM(N) ;
3814              REMEMBER := TRUE ;
3815              MAKEMOVE(N,FALSE) ;
3816              IF CON(-K,KINGSQ(K)) ~= 0
3817              THEN NULLMOVE := TRUE
3818              ELSE IF (REW(0) < CHECKING OR (REW(0) >= CHECKING
3819                AND CON(K,KINGSQ(-K)) ~= 0))
3820                 THEN IF (RTMV = 0)
3821                      THEN RTMV := N
3822                      ELSE NULLMOVE := TRUE ;
3823              SETBOARD(BUFFER);
3824              IF ~STARTTHISPROBLEM(TRUE)
3825              THEN GOTO STARTS;
3826              IF NULLMOVE
3827              THEN REMAKE(KINGSQ(K),KINGSQ(K)) ;
3828              IF NULLMOVE
3829              THEN GO TO AMBIG ;
3830       END ;
3831     END ;
3832 NEXTN:
3833   END ;
3834   IF (RTMV ~= 0)
3835   THEN GO TO QUIT ;
3836 AMBIG:
3837   MESSAGE2 := IF RTMV=0
3838               THEN "NOT LEGAL"
3839               ELSE "AMBIGUOUS" ;
3840   RTMV := 0 ;
3841   NULLMOVE := TRUE ;
3842 QUIT:
3843 END OFFINDRTMV ;
3844
3845 @TITLE,          "GETMOVE"
3846 LOGICAL PROCEDURE GETMOVE(STRING(1) ARRAY G(*)) ;
3847 COMMENT converts chess move from chess to internal notation ;
3848 BEGIN
3849   INTEGER   PTR, I ;
3850   STRING(1) T ;
```

```
3851    LOGICAL LEFT, SLASH, CAPT, PTRSET, PREV_ENGLISH ;
3852    MESSAGE1 := MESSAGE2 := SPACE ;
3853    PREV_ENGLISH := ENGLISH ;
3854    I := PTR := 0 ;
3855    WHILE (I <= CHARCOUNT)
3856    DO BEGIN
3857      WHILE (G(I) = " ")
3858      DO I := I + 1 ;
3859      IF (G(I) = ".")
3860      THEN CHARCOUNT := PTR
3861      ELSE BEGIN
3862        G(PTR) := G(I) ;
3863        PTR := PTR + 1 ;
3864      END ;
3865      I := I + 1 ;
3866    END ;
3867    CHARCOUNT := PTR -1 ;
3868    FOR NOTUSED := 1,2
3869    DO BEGIN
3870      FOR I := 0, 1
3871      DO REW(I) := FMAN(I) := TMAN(I) := TSQ(I) := FSQ(I) := 0 ;
3872      REW(2) := FMAN(2) := TMAN(2) := 0 ;
3873      NULLMOVE := FALSE ;
3874      PREV_MESSAGE1 := MESSAGE1 ;
3875      PREV_MESSAGE2 := MESSAGE2 ;
3876      IF ~ENGLISH
3877      THEN ALGEBRAIC(G,CHARCOUNT+1)
3878      ELSE BEGIN
3879        CAPT := FALSE ;
3880        LEFT := TRUE ;
3881        I := -1 ;
3882        PTR := 0 ;
3883  NOSLASH:
3884        SLASH := FALSE ;
3885  UNSET:
3886        PTRSET := FALSE ;
3887  BUMP:
3888        I := I + 1 ;
3889        T := G (I) ;
3890        IF (I > CHARCOUNT) OR (T = "!") OR (T = "?")
3891        THEN GO TO ENDRIGHT ;
3892        IF (T = "-")
3893        THEN GO TO ENDLEFT ;
3894        IF (T = "X")  OR (T = "*")
3895        THEN BEGIN
3896          CAPT := TRUE ;
3897          GO TO ENDLEFT ;
3898        END ;
3899        IF (T = "/")
3900        THEN BEGIN
3901          SLASH := TRUE ;
3902          IF (LEFT)
3903          THEN GETMAN (G, FMAN, PTR)
3904          ELSE GETMAN (G, TMAN, PTR) ;
3905          GO TO UNSET ;
3906        END ;
3907        IF (T = "C") OR (T = "+") OR (T = "M")
3908        THEN BEGIN
3909          REW(0) := REW(0) + CHECKING ;
3910          GO TO ENDRIGHT ;
3911        END ;
3912        IF (T = "O") OR (T = "0")
3913        THEN BEGIN
3914          FOR I := I+1 UNTIL CHARCOUNT
3915          DO IF (G(I) = "0") OR (G(I) = "O")
3916             THEN REW(1) := REW(1) + 1 ;
3917          IF (REW(1) > 0)
3918          THEN REW(1) := IF (REW(1) > 1)
3919                         THEN QSIDE
3920                         ELSE KSIDE ;
```

```
3921          GO TO QUIT ;
3922        END ;
3923        IF (T = "=") OR (T = "(")
3924        THEN BEGIN
3925          FOR J := KNIGHT,BISHOP,ROOK,QUEEN
3926          DO IF (G(I+1) = CHESSMAN(J))
3927             THEN REW(2) := J +PROMOTE ;
3928          GO TO ENDRIGHT ;
3929        END ;
3930        IF (T = "E")
3931        THEN
3932        COMMENT enpassant capture ;
3933        BEGIN
3934          TMAN(2) := ENPRIS ;
3935          GO TO QUIT ;
3936        END ;
3937        IF (T = "D") AND (G (I + 1) = "R")
3938        THEN KTEST := 3
3939        ELSE IF (T = "R") AND (G (I + 3) = "I")
3940             THEN KTEST := 1 ;
3941        IF (KTEST ~= 0)
3942        THEN BEGIN
3943          IF (KTEST = 1)
3944          THEN MESSAGE1 := "RESIGNATION ACCEPTED"
3945          ELSE IF (KTEST = 3)
3946               THEN IF (SCORE(RTMV) < PAWNVALUE)
3947                    THEN MESSAGE1 := "DRAW ACCEPTED."
3948                    ELSE NULLMOVE := TRUE ;
3949          IF ~NULLMOVE AND (LEFT OR (KTEST ~= 1))
3950          THEN GO TO STARTS ;
3951          IF LEFT
3952          THEN GO TO QUIT ;
3953          NULLMOVE := FALSE ;
3954          GO TO ENDRIGHT ;
3955        END ;
3956        IF (~ PTRSET)
3957        THEN BEGIN
3958          PTR := I ;
3959          PTRSET := TRUE ;
3960        END ;
3961        GO TO BUMP ;
3962  ENDLEFT:
3963        LEFT := FALSE ;
3964        IF SLASH
3965        THEN GETSQ(G,FSQ,PTR)
3966        ELSE GETMAN(G,FMAN,PTR) ;
3967        GO TO NOSLASH ;
3968  ENDRIGHT:
3969        IF CAPT AND ~ SLASH
3970        THEN GETMAN(G,TMAN,PTR)
3971        ELSE GETSQ(G,TSQ,PTR) ;
3972  QUIT:
3973      END ;
3974      IF ~NULLMOVE
3975      THEN FINDRTMV ;
3976      IF ~NULLMOVE
3977      THEN GO TO EXIT_THIS_LOOP ;
3978      ENGLISH := ~ENGLISH ;
3979    END OFFORLOOP ;
3980    WRITE(PREV_MESSAGE1) ;
3981    WRITE(PREV_MESSAGE2) ;
3982  EXIT_THIS_LOOP:
3983    COMMENT  don't reset the notation just because of the book ;
3984    IF WHO
3985    THEN ENGLISH := PREV_ENGLISH ;
3986    ~NULLMOVE
3987  END OFGETMOVE ;
3988  @TITLE,"HUB"
3989  PROCEDURE HUB ;
3990  COMMENT   central hub of chess program. ;
```

```
3991 BEGIN
3992   SETUP ;
3993   WHILE (KTEST = 0)
3994   DO BEGIN
3995     IF K = 1 AND (EXAMINE OR (MOVECOUNT REM 5) = 1)
3996     THEN BEGIN
3997       WRITE(" ") ;
3998       WRITE("Await = ", COMPUTERTIME DIV 3600, " MIN.  ",
3999         "USER = ",  USERTIME DIV 3600, " MIN.") ;
4000       IF TEST
4001       THEN WRITE("(CPU:  Await = ", WITA DIV 60, " SECS.",
4002         " USER = ", USER  DIV 60, " SECS.)") ;
4003       WRITE(" ") ;
4004       IF CBOTH
4005       THEN BEGIN
4006         WRITE("SHALL I CONTINUE") ;
4007         WHILE READU
4008         DO TRACER(CHARCOUNT,BUFFER(0|20)) ;
4009         IF (U(0) ~= "Y")
4010         THEN BOTH := WHO := CBOTH := FALSE ;
4011       END ;
4012     END ;
4013     VALUES(BISHOP) := VALUES(KNIGHT) + (IF ENDGAME OR
4014             PIECES(-BISHOP) < 2 AND PIECES(BISHOP) < 2
4015                     THEN 0
4016                     ELSE 2) ;
4017     PREDICT := DEBIT(0) := DEBIT(-1) := CREDIT(0) := CREDIT(-1) := 0 ;
4018     LASTSQ(-1) := LASTSQ(0) ;
4019     LASTDIR(-1) := LASTDIR(0) ;
4020     LASTSQ(0) := LASTSQ(1) ;
4021     LASTDIR(0) := LASTDIR(1) ;
4022     UPPER := EIGHTS ;
4023     LEVEL :=1 ;
4024     IF (WHO)
4025     THEN COMPUTER
4026     ELSE OPPONENT ;
4027     IF ~WHO
4028     THEN SKTEST := KTEST ;
4029     IF NULLMOVE
4030     THEN BEGIN
4031       NULLMOVE := FALSE ;
4032       WHO := TRUE ;
4033       REPLY := WSIZE ;
4034     END ELSE FINALIZE ;
4035   END ;
4036 QUIT:
4037   WRITE(REASON(KTEST)) ;
4038   PRINTSHORTBRD(1) ;
4039 END OFHUB ;
4040
4041 @TITLE,"OPPONENT "
4042 COMMENT **************opponent***************** ;
4043
4044 PROCEDURE OPPONENT ;
4045 COMMENT makes moves read from teletype ;
4046 BEGIN
4047   INTEGER NUMS, N, MF, MT, RE ;
4048   LOGICAL PREDICTOR ;
4049   IF LEGAL
4050   THEN BEGIN
4051     SCORES := IF (BOOK <= 0) AND (REPLY < 1
4052       OR (REPLY > NODETO(ORIGIN,0)))
4053               THEN TRUE
4054               ELSE FALSE ;
4055     WRITE(COLOR(K),"'S MOVE ",MOVECOUNT,BELL) ;
4056     FOR I := 1 UNTIL 5 DO WRITEON(BELL);
4057     IOCONTROL(2) ;
4058     LISTLEGALMOVES (FALSE) ;
4059   END ELSE LEGAL := TRUE ;
4060   IF (NUM ~= 0)
```

```
4061    THEN BEGIN
4062      IF COKO AND FALSE
4063      THEN  GET(3) ;
4064        COMMENT take input from unit 3 ;
4065      IF (BOOK = 1)
4066      THEN CREATEBOOK ;
4067        COMMENT coko input restored in readu ;
4068      IF MOVES_LEFT < 20
4069      THEN WRITE("Use ?Y to update internal clock and move count");
4070  THINK:
4071      IF THINKING AND ORIGIN > 0 AND NODETO(ORIGIN,0) > 0 AND KTEST = 0
4072      THEN BEGIN
4073        FLAG(1) := 0;
4074              RESTORESTATE(ORIGIN) ;
4075              RTMV := TRY(1) ;
4076              REPLY := 1 ;
4077              WRITE("Assuming ") ;
4078              PRINTLINE(RTMV,2) ;
4079          FOR I := 2 UNTIL 5 DO
4080          IF I <= TRY(WSIZE)
4081          THEN PRINTLINE(TRY(I), 4);
4082              FINALIZE ;
4083              WHO := FALSE ;
4084              COMPUTER ;
4085              FINALIZE ;
4086              WHO := FALSE ;
4087              SCORES := TRUE ;
4088              IF FLAG(1) = 0
4089              THEN WRITE("If bad prediction, enter '?R1'")
4090              ELSE BEGIN
4091                TRACER(2,"?R1") ;
4092                ORIGIN := -1 ;
4093                GO TO THINK ;
4094              END;
4095              WRITE("Type OK to accept assumed move");
4096              WRITE("Hit BREAK key to regain control");
4097      END ;
4098  LOOP:
4099      IF (BOOK < 1)
4100      THEN WHILE READU
4101      DO TRACER(CHARCOUNT,BUFFER(0|20)) ;
4102      IF THINKING AND BUFFER(0|2) = "OK"
4103      THEN GOTO THINK ELSE
4104      IF BUFFER(0|5) = "BLITZ"
4105      THEN BEGIN
4106        BLITZ ;
4107        GOTO LOOP ;
4108      END ELSE IF BUFFER(0|5) = "INTER"
4109                THEN BEGIN
4110                  INTERMEDIATE ;
4111                  GOTO LOOP ;
4112      END ELSE IF BUFFER(0|5) = "EXPER"
4113                THEN BEGIN
4114                  EXPERIMENTAL ;
4115                  GOTO LOOP ;
4116      END ELSE IF BUFFER(0|5) = "TOURN"
4117                THEN BEGIN
4118                  TOURNAMENTS ;
4119                  GOTO LOOP ;
4120      END ELSE IF BUFFER(0|4) = "SWAP"
4121                THEN BEGIN
4122                  K := -K ;
4123                  WRITE("SIDES SWAPPED. CONTINUE.") ;
4124                  IOCONTROL(2) ;
4125                  GO TO LOOP ;
4126      END ELSE IF BUFFER(0|4) = "PLAY"
4127                THEN NULLMOVE := TRUE
4128                ELSE IF BUFFER(0|3) = "PAB"
4129                     THEN BEGIN
4130                       PAB := ~PAB ;
```

```
4131                    WRITE("Palphabeta", PAB) ;
4132                    GOTO LOOP ;
4133      END ELSE
4134      BEGIN
4135        IF ~GETMOVE(U)
4136             THEN BEGIN
4137               BOOK := 0 ;
4138               WRITE(" TRY AGAIN ") ;
4139               FOR I := 0 UNTIL CHARCOUNT
4140               DO WRITEON(U(I)) ;
4141               GO TO LOOP ;
4142        END ;
4143        IF LOOKAHEAD AND BOOK = 0 OR EXAMINE OR SCORES
4144        THEN BEGIN
4145          MF := MOVEFROM(RTMV) ;
4146          MT := MOVETO(RTMV) ;
4147          RE := ABS(REWARD(RTMV)) REM CHECKING ;
4148          N := IF EXAMINE AND ~ SCORES
4149               THEN NODETO(ORIGIN, REPLY)
4150               ELSE -1 ;
4151          IF EXAMINE AND N > 0
4152               THEN RESTORE(N)
4153               ELSE IF LOOKAHEAD AND ORIGIN ~=  GETNODE(1) AND ~SCORES
4154                 AND (((REPLY > 0) AND (REPLY <= WIDTH)) AND (ORIGIN > 0))
4155                    THEN RESTORE(ORIGIN) ;
4156          IF EXAMINE
4157          THEN TRACER(2, "?DP") ;
4158          N := (MAX_WID+1) ;
4159          FOR I := 1 UNTIL TRY(WSIZE)
4160          DO IF (MF = MOVEFROM(TRY(I)))
4161             THEN IF (MT = MOVETO(TRY(I)))
4162                  THEN IF (RE <= PFTWO) OR RE = ABS(REWARD(TRY(I)))
4163                    REM CHECKING
4164                       THEN N := I ;
4165          PREDICT := IF N > MAX_WID AND TRY(WSIZE) > 1 OR LOOKAHEAD
4166            AND (N > NODETO(ORIGIN,0))
4167                     THEN -N
4168                     ELSE N ;
4169                       COMMENT don't build tree if reply is not in
4170                       window ;
4171          IF EXAMINE AND (N <= MAX_WID)
4172          THEN BEGIN
4173            FLAG(1) := 0 ;
4174            SEARCH;
4175            TRACER(1,"?D") ;
4176            FLAG(1) := 1 ;
4177            REPLY := (MAX_WID+1) ;
4178            FOR I := 1 UNTIL TRY(WSIZE)
4179            DO IF (MF = MOVEFROM(TRY(I))) AND (MT = MOVETO(TRY(I)))
4180               THEN IF (RE <= PFTWO)
4181                    THEN REPLY := I
4182                    ELSE IF RE = ABS(REWARD(TRY(I))) REM CHECKING
4183                         THEN REPLY := I ;
4184          END ELSE REPLY := N ;
4185          IF ~WS OR ~ BOTH
4186          THEN IF (REPLY <= WIDTH)
4187               THEN RTMV := TRY(REPLY)
4188          ELSE BEGIN
4189            LISTMOVES ;
4190            FINDRTMV ;
4191          END ;
4192          IF EXAMINE
4193          THEN BEGIN
4194            COMMENT have experienced severe problems with use of
4195              *g. need to restore the window array, try. As a result
4196              parts of the movelist are overlayed, but cannot guarantee
4197              that the moves in the window overlay themselves ;
4198
4199                 MF := 0 ;
4200            NUMS := NODETO(ORIGIN,0) ;
```

```
4201            IF (REPLY > NUMS)
4202            THEN MF := IF (REPLY > WIDTH)
4203                       THEN 0
4204                       ELSE -REPLY
4205            ELSE FOR I := 1 UNTIL NUMS
4206            DO IF SCORE(TRY(I)) >= SCORE(RTMV)
4207               THEN MF := MF + 1 ;
4208            WRITE(RTMV,"WAS SELECTED, IT WAS",  MF,
4209              "IN MY LIST, ORIGINALLY",REPLY,TRY(0)) ;
4210            DESCRIBEMOVE(RTMV,MC) ;
4211            IOCONTROL(2) ;
4212            FOR I := 0 UNTIL 23
4213            DO WRITEON(MC(I)) ;
4214            IF (MF <= 0)
4215            THEN REPLY := 0 ;
4216          END ;
4217        END ELSE REPLY := 0 ;
4218      END ;
4219      FOR I := 0 UNTIL 23
4220      DO MC(I) := U(I) ;
4221      FOR I := CHARCOUNT+1 UNTIL 23
4222      DO MC(I) := " " ;
4223    END ELSE GO TO STARTS ;
4224  END OFOPPONENT ;
4225
4226  @TITLE,"REINITIALIZE(STRING(1) VALUE CHAR)  "
4227  PROCEDURE REINITIALIZE(STRING(1) VALUE CHAR) ;
4228  BEGIN
4229    INTEGER M, MM ;
4230    M := DECODE(CHAR) - 240 ;
4231    IF (M<0) OR (M>9)
4232    THEN GO TO QUIT ;
4233    COUNT := COUNT -M ;
4234    IF (COUNT < 0)
4235    THEN COUNT := 1 ;
4236     COMMENT newgame := false ;
4237    MM :=  ((M + (IF (K = 1)
4238                  THEN 2
4239                  ELSE 1)) DIV 2) ;
4240    MOVECOUNT := MOVECOUNT  - MM ;
4241    MOVES_LEFT := MOVES_LEFT + MM ;
4242    IF ((M REM 2) = 0)
4243    THEN K := -K ;
4244    MMTTS(-COUNT, HISTORY) ;
4245    SETBOARD(HISTORY) ;
4246    COUNT := COUNT-1 ;
4247    WRITE(COLOR(K),"'S MOVE ",MOVECOUNT, BELL) ;
4248    IF  STARTTHISPROBLEM(TRUE)
4249    THEN LISTLEGALMOVES(FALSE)
4250    ELSE GO TO STARTS ;
4251    INIT_TREE ;
4252    HASH := NEWHASH(BRD, K, KEY) ;
4253    ENDPLAY := IF (ENDPLAY > M)
4254               THEN ENDPLAY-M-1
4255               ELSE 0 ;
4256     COMMENT if retracting over an irreversible move it is necessary
4257     to reconstruct earlier history -for proper draw detection ;
4258  QUIT:
4259  END OFREINITIALIZE ;
4260
4261  @TITLE,"SELECTRTMV"
4262  PROCEDURE SELECTRTMV (LOGICAL VALUE LOSING) ;
4263  COMMENT  choose the move with the highest score ;
4264  BEGIN
4265    INTEGER NN,J,I, JJ, SQ, BESTSCORE,TEMP,MINSCR,SCR, SUFFICE, X,
4266      Y, PAST, PRES, WID ;
4267     COMMENT   score all the moves ;
4268    WID := IF ALTERNATIVE
4269           THEN CAPT_WID
4270           ELSE MAX_WID ;
```

```
4271    IF ALTERNATIVE
4272    THEN CAPT_ORDER(BESTSCORE, LOSING)
4273    ELSE MOBILITYSCORE(BESTSCORE) ;
4274    SPAN(K) := SPAN(0) := NUM ;
4275     COMMENT   put the "width" best moves into array try. ;
4276    NN := 0 ;
4277    FOR N := K STEP K UNTIL NUMBER(K)
4278    DO BEGIN
4279      SCR := SCORE(N) ;
4280             COMMENT  mate, dbl ch., and forcing moves ;
4281      IF ~ALTERNATIVE
4282      THEN IF (ABS(SCR) > SEVENS) OR (ABS(REWARD(N)) >= TWOCHECK)
4283           THEN IF (BESTSCORE < SEVENS) AND SCR > -NINES
4284                THEN SCORE(N) := SCR :=
4285                                 BESTSCORE - (BESTSCORE -SCR) DIV 100;
4286      IF (SCR ~= BAD)
4287      THEN BEGIN
4288        FOR IX := 1 UNTIL NN
4289        DO IF (SCR > SCORE(TRY(IX)))
4290           THEN BEGIN
4291             I := IX ;
4292             FOR JJ := NN STEP -1 UNTIL I
4293             DO TRY(JJ+1) := TRY(JJ) ;
4294             GO TO FOUND ;
4295        END ;
4296        I := NN+1 ;
4297  FOUND:
4298        COMMENT IF (NN < WID) THEN NN := NN+1 ;
4299        IF (NN < MAX_WID)
4300        THEN NN := NN + 1 ;
4301        TRY(I) := N ;
4302      END ;
4303    END ;
4304    BESTSCORE := SCORE(TRY(1)) ;
4305     COMMENT   the best moves are now ordered. ;
4306    SUFFICE := IF (LEVEL < 3)
4307               THEN DRAW
4308               ELSE PAWNVALUE ;
4309    J := IF ALTERNATIVE
4310         THEN 1
4311         ELSE 2 ;
4312    TRY(NN+1) := 0 ;
4313    IF NN <= 1
4314    THEN BEGIN
4315      TRY(0) := TRY(WSIZE) := NN ;
4316      RTMV := TRY(NN) ;
4317    END ELSE
4318    BEGIN
4319      PAST := TRY(1) ;
4320      X := ABS(REWARD(PAST)) ;
4321      MINSCR := SCORE(TRY(IF NN > WID
4322                          THEN WID
4323                          ELSE NN)) ;
4324      FOR I := J UNTIL NN
4325      DO BEGIN
4326        PRES := TRY(I) ;
4327        SCR := SCORE(PRES) ;
4328        Y := ABS(REWARD(PRES)) ;
4329        IF (Y = 2) AND (X = 1)
4330        THEN Y := 0 ;
4331        IF (ABS(BESTSCORE - SCR) < SUFFICE)
4332        THEN J := I ;
4333        TEMP := SCORE(PAST) - SCR ;
4334        IF ~CAPTURE_TREE OR LEVEL > 2
4335        THEN IF (TEMP < EPSILON) AND (TEMP > DELTA)
4336             THEN IF Y > X AND (Y < DISCHECK OR Y >= TWOCHECK)
4337            THEN BEGIN
4338        COMMENT if too many disc. ch., don't re-order ;
4339              TRY(I-1) := PRES ;
4340              TRY(I) := PAST ;
```

```
4341          IF I = 2 THEN SCORE(PRES) := SCORE(PAST);
4342          IF I = 2 THEN REWARD(PRES) := -ABS(REWARD(PRES));
4343              PRES := PAST ;
4344              Y := X ;
4345        END ;
4346        PAST := PRES ;
4347        X := Y ;
4348      END ;
4349      JJ := J ;
4350      IF ~ALTERNATIVE
4351      THEN FOR I := 3 UNTIL NN
4352           DO BEGIN
4353             J := I ;
4354             WHILE(J <= NN AND MOVEFROM(TRY(J)) = MOVEFROM(TRY(I-1)))
4355             DO J := J +1 ;
4356             IF (J <= NN) AND (J > I) AND (SCORE(TRY(I)) -SCORE(TRY(J))
4357               < PAWNVALUE)
4358             THEN BEGIN
4359  COMMENT break up long sequences of moves with same piece;
4360               PRES := TRY(J) ;
4361               FOR L := J STEP -1 UNTIL I+1
4362               DO TRY(L) := TRY(L-1) ;
4363               IF (J > JJ) AND (I <= JJ)
4364               THEN JJ := JJ +1 ;
4365               TRY(I) := PRES ;
4366             END ;
4367      END ;
4368     COMMENT   used by search and tree ;
4369      NN := IF (NN > WID)
4370            THEN WID
4371            ELSE NN ;
4372      TRY(0) := IF (JJ > NN)
4373                THEN NN
4374                ELSE JJ ;
4375      J := IF (MOVECOUNT > 2) OR ~NEWGAME
4376           THEN 1
4377           ELSE (TIME(-1) REM TRY(0)) +1 ;
4378      RTMV := TRY(J) ;
4379      IF DEBUG
4380      THEN WRITE("SELE",NUM,BESTSCORE,J,RTMV,JJ,NN,WID,EARLY, OPENING,
4381        ALTERNATIVE, LATE_END, MEN(K), PIECES(0)) ;
4382      IF SCORE(TRY(NN-1)) <= MINSCR
4383      THEN SCORE(TRY(NN-1)) := SCORE(TRY(NN)) ;
4384      TRY(WSIZE) := NN ;
4385     COMMENT IF TEST AND ALTERNATIVE THEN PRINTMOVES(K) ;
4386    END ;
4387  END OFSELECTRTMV ;
4388
4389  @TITLE,"SETUP  "
4390  PROCEDURE SETUP ;
4391  BEGIN
4392    INTEGER L ;
4393    IF (MOVECOUNT > 0)
4394    THEN BEGIN
4395      WRITE("FAILURE.  ", "Either type REPEAT, or") ;
4396      IF (MOVECOUNT -INITIALCNT > 5) OR CAPTURE_TREE AND SKILL(0|1)
4397        ~= " "
4398      THEN GAMERECORD(TRUE) ;
4399    END ;
4400    BOOK := -1 ;
4401    WRITE("New game? (YES or NO)") ;
4402    WHILE READU
4403    DO TRACER(CHARCOUNT,BUFFER(0|20)) ;
4404     COMMENT delay initialization ;
4405    LEVEL := 0 ;
4406    DEBIT(-1) := CREDIT(0) := CREDIT(-1) := 0 ;
4407    KTEST := ONEVAL := TWOVAL := NUM := MOVEFROM(N) := SKTEST := N := 0 ;
4408    FOR L := -1 UNTIL 1
4409    DO SPAN(L) := NUMBER(L) := 0 ;
4410    REMEMBER := TERMINAL := LEGAL := TRUE ;
```

```
4411   LOSS(ELIMIT) := SQUARE(ELIMIT) := ELIMIT ;
4412   LOSS(0) := SQUARE(0) := 0 ;
4413   LASTSQ(0) := LASTSQ(1) := 0 ;
4414   LASTDIR(0) := LASTDIR(1) := 0 ;
4415   IF (U(0) = "R")
4416   THEN GO TO START ;
4417   BOOKMOVE := DUP := REC := NOD := SAL := LIMBS := 0 ;
4418   TOT := DYN := CAP := BRAN := BRAN_S := REC_S := DUP_S := 0 ;
4419   TRY(0) := 0 ;
4420   REPLY := WSIZE ;
4421   BOTH := CBOTH :=  ALTERNATIVE := FALSE ;
4422   K := NUMB := -1 ;
4423   USER := TIME(1) ;
4424   CPU := ELAPSEDTIME := USERTIME := COMPUTERTIME := WITA := 0 ;
4425   MOVETIME := 180 ;
4426   ENDPLAY := 1 ;
4427   MOVECOUNT := 0 ;
4428    COMMENT *P AND *3 PROTECTION ;
4429   HISTORY(115|25) := HISTORY(85|25) := "    ....            " ;
4430   HISTORY(105|10) := "     ( 0) " ;
4431   NEWGAME := IF (U(0) = "N")
4432              THEN FALSE
4433              ELSE TRUE ;
4434 AGAIN:
4435   IF NEWGAME
4436   THEN BUFFER := "RNBQKBNRPPPPPPPP+888888PPPPPPPPRNBQKBNR+"
4437   ELSE BEGIN
4438     WRITE("Enter the board (in quasi Forsythe notation)") ;
4439     WHILE READU
4440     DO TRACER(CHARCOUNT,BUFFER(0|20)) ;
4441   END STARTOLDGAME ;
4442   SETBOARD(BUFFER) ;
4443   IF ~NEWGAME AND (MOVECOUNT <= 1)
4444   THEN MOVECOUNT := 17 ;
4445   MOVES_LEFT := 40 - (MOVECOUNT REM 40) ;
4446   TIME_LEFT := 324000 ;
4447    COMMENT 90 MINS. ;
4448   INITIALCNT := MOVECOUNT ;
4449   HISTORY(0|78) := BUFFER(0|78) ;
4450   FOR I := -3 UNTIL 3
4451   DO HISTORY(I+81|1) := IF CASTLE(I)
4452                         THEN "T"
4453                         ELSE "F" ;
4454 START:
4455   COMMENT REFLECT(BRD);
4456   IF ~STARTTHISPROBLEM(TRUE)
4457   THEN GO TO AGAIN ;
4458   OPEN(1) := OPEN(-1) := EARLY := TRUE ;
4459   WHO := ENDGAME := FALSE ;
4460   IF (~NEWGAME)
4461   THEN BEGIN
4462     PRINTSHORTBRD(-1) ;
4463     BOOK := 0 ;
4464   END ;
4465   LENGTH := MAXLEV ;
4466   MODE  := IF (MAXTREE < 10)
4467            THEN 1
4468            ELSE IF (LENGTH > 7) AND (MAXTREE >= 600)
4469                 THEN 6
4470                 ELSE IF LENGTH > 5
4471                      THEN 5
4472                      ELSE IF (LENGTH > 3)
4473                           THEN 4
4474                           ELSE IF ~CAPTURE_TREE
4475                                THEN 2
4476                                ELSE 3 ;
4477   SKILL := CASE MODE OF ("   DEBUG", "   SPEED", "  BEGINNER",
4478     "INTERMEDIATE", "TOURNAMENT", "EXPERIMENTAL") ;
4479   FOR I := 1 UNTIL 10
4480   DO M_TIMES(I) := 60*(CASE I OF (1, 12, 20, 30, 90, 120, 600, 0, 0, 0))
```

```
4481      ;
4482    M_TIMES(0) := M_TIMES(MODE) DIV 3 ;
4483    INIT_TREE ;
4484    IF (SKILL = "EXPERIMENTAL")
4485    THEN QUICK := FALSE
4486    ELSE IF (SKILL ~= "INTERMEDIATE")
4487         THEN TOURNAMENT := FALSE ;
4488    WRITEON(SKILL, " mode") ;
4489    WRITE("Enter your move or type PLAY") ;
4490    SENDTIME := TIME(-1) ;
4491    OLDCPU := TIME(1) ;
4492    HASH := NEWHASH(BRD, K, KEY) ;
4493    REPEATS(ENDPLAY) := HASH ;
4494    IF (K < 0)  AND INITIALCNT = MOVECOUNT
4495    THEN BEGIN
4496      COMMENT  not repeating part of continuing game ;
4497      COUNT := COUNT +1 ;
4498      MMTTS(COUNT,HISTORY) ;
4499    END ;
4500    WHO := IF CBOTH
4501           THEN TRUE
4502           ELSE FALSE ;
4503  END OFSETUP ;
4504
4505  @TITLE,"TAKEBACKMOVE"
4506  PROCEDURE TAKEBACKMOVE(LOGICAL VALUE HALF_DONE) ;
4507  COMMENT  retracts last move made by makemove during mobilityscore. ;
4508  BEGIN
4509    INTEGER K,  LL ;
4510    LOSS(ELIMIT) := SQUARE(ELIMIT) ;
4511    K := IF (M5 > 0)
4512         THEN 1
4513         ELSE -1 ;
4514    IF HALF_DONE
4515    THEN BEGIN
4516      IF M7 ~= 0 AND (ABS(M7) < KING OR ABS(M5) = PAWN)
4517      THEN BEGIN
4518        MEN(-K) := MEN(-K) + 1 ;
4519           COMMENT PIECES(M6) := PIECES(M6) +1;
4520           COMMENT PIECES(0) := PIECES(0) +1;
4521      END ;
4522      LL := 0 ;
4523      LOSS(0) := SQUARE(0) ;
4524    END ELSE LL := 2 ;
4525    IF ABS(M5) = KING
4526    THEN KINGSQ(K) := M2 ;
4527    IF M11 > 0 AND M11 ~= M3
4528    THEN BRD(M11) := -K*ENPRIS ;
4529    IF (M9 > 0)
4530    THEN BRD(M9) := IF M8 = 0
4531                    THEN -K*PAWN
4532                    ELSE K*ROOK ;
4533    UNMAKE(M3) ;
4534    IF (M8 > 0)
4535    THEN UNMAKE(M8) ;
4536    BRD(M2) := M5 ;
4537    BRD(M3) := M7 ;
4538    IF (M9 > 0)
4539    THEN MAKE(M9) ;
4540    IF (M7 ~= 0)
4541    THEN MAKE(M3) ;
4542     COMMENT   restore captured piece. ;
4543    MAKE(M2) ;
4544        COMMENT  retract promotion. ;
4545    IF M14 ~= 0
4546    THEN IF ~STARTTHISPROBLEM(TRUE)
4547         THEN WRITE("TAKE", M8, M9, M11, M14) ;
4548     COMMENT IF TEST AND (DEBUG OR LL = 0) THEN
4549     WRITE("TAKE",LL,NUMB,MEN(-1), MEN(1),LEVEL, M2,M3,M8,M9, SEC(0,M3),
4550      M11, M14, BRD(M3),CON(-K,M2),MAKEMV) ;
```

```
4551   FOR II := NUMB STEP -1 UNTIL LL
4552   DO REMAKE(RLIST(II),TLIST(II)) ;
4553   IF (ABS(M5) = KING)
4554   THEN KINGCONTROL(K) ;
4555   IF MEN(1) + MEN(-1) ~= PIECES(0)
4556   THEN BEGIN
4557     WRITE("CONSISTENCY ", MEN(1), MEN(-1), PIECES(0)) ;
4558     IF TEST
4559     THEN TRACER(4, "?PX18") ;
4560   END ;
4561 END OFTAKEBACKMOVE ;
4562
4563 @TITLE,"TRACER "
4564 PROCEDURE TRACER(INTEGER VALUE CHARCOUNT ;
4565 STRING(20) VALUE UU) ;
4566 COMMENT initiates trace according to trace parameters ;
4567 BEGIN
4568   INTEGER I, J ;
4569   STRING(140) BUFF ;
4570   LOGICAL OLDECHO ;
4571   OLDECHO := ECHO ;
4572   I := 0 ;
4573   IF CHARCOUNT > 19
4574   THEN CHARCOUNT := 19 ;
4575   WHILE (I < CHARCOUNT)
4576   DO BEGIN
4577     I := I + 1 ;
4578     COMMENT  processing terminated if illegal character found in
4579              ~ stream ;
4580     J := DECODE(UU(I|1)) - 192 ;
4581     IF ((J < 0) OR (J > 57))
4582     THEN I := CHARCOUNT
4583     ELSE CASE J OF
4584       BEGIN
4585
4586         BOOK := - BOOK;
4587
4588
4589
4590
4591
4592       COMMENT     "a" ;
4593         FOR L := NUMBER(-1) UNTIL -1
4594         DO PRINTLINE(L,0) ;
4595       COMMENT     "b" ;
4596         COKO := ~ COKO ;
4597       COMMENT     "c" ;
4598         FOR J := 1 UNTIL TRY(WSIZE)
4599         DO PRINTLINE(TRY(J),0) ;
4600       COMMENT     "d" ;
4601         EXAMINE := ~ EXAMINE ;
4602       COMMENT     "e" ;
4603         ECHO := ~ ECHO ;
4604       COMMENT     "f" ;
4605         IF LOOKAHEAD
4606         THEN IF (LASTORIGIN > 0)  AND ORIGIN > 0
4607              THEN BEGIN
4608                J := NODETO(LASTORIGIN,0) ;
4609                IF J > 5
4610                THEN J := 5 ;
4611                PUNCHBRD(TRUE, BUFF) ;
4612                FOR REPLY := 1 UNTIL J
4613                DO BEGIN
4614                  RESTORESTATE(LASTORIGIN) ;
4615                  PRINTLINE(TRY(REPLY), 0) ;
4616                  WRITEON(LASTORIGIN, NODETO(LASTORIGIN,0),
4617                    NODETO(LASTORIGIN,REPLY),
4618                    PARENT(NODETO(LASTORIGIN,REPLY))) ;
4619                  GAMETREE(NODETO(LASTORIGIN,REPLY), 1, 2) ;
4620                END ;
```

```
4621                RESTORESTATE(ORIGIN) ;
4622                SETBOARD(BUFF) ;
4623                IF STARTTHISPROBLEM(TRUE)
4624                THEN LISTMOVES
4625                ELSE GOTO STARTS ;
4626              END ;
4627            COMMENT     "g" ;
4628              WRITE("$COPY WITA:COMMANDS(1,2) for description") ;
4629            COMMENT WRITECARD( " Summer 1973 version of WITA, a chess-playing
4630            program", " written by T.A. Marsland, Univ. of Alberta, Edmonton",
4631            " the primary aim of this program is to explore techniques",
4632            "for highly selective searches of large game trees.",
4633            "a goal of controlled search with a branching factor no",
4634            "greater than six is sought.  the 1968 version of WITA",
4635            "was written in Burroughs extended algol, for the B-5500.") ;
4636            COMMENT     "h" ;
4637              WRITE("Use algebraic or Descriptive notation for moves",
4638                " A ?OPTION may be entered at any time.",
4639                " ENTER ?6 to alter tree parameters.") ;
4640            COMMENT     "i" ;
4641              ;
4642              ;
4643              ;
4644              ;
4645              ;
4646              ;
4647              ;
4648              CAPTURE_TREE := ~CAPTURE_TREE ;
4649            COMMENT     "j" ;
4650              GO TO STARTS ;
4651            COMMENT     "k" ;
4652              BEGIN
4653                IF (LOOKAHEAD)
4654                THEN EXAMINE := LOOKAHEAD := TEST := FALSE
4655                ELSE LOOKAHEAD := TRUE
4656              END ;
4657            COMMENT     "l" ;
4658              MONITOR := ~MONITOR ;
4659            COMMENT     "m" ;
4660              BEGIN
4661                TTYOUT := IF TTYOUT = 6
4662                          THEN 4
4663                          ELSE 6 ;
4664                PUT(TTYOUT) ;
4665              END ;
4666            COMMENT     "n" ;
4667              BEGIN
4668                WRITE ("?C=", COKO,"?1=", DUMPP, "?2=", WS, "?45= ", IF CBOTH
4669                                        THEN "BOTH"
4670                                        ELSE IF BOTH
4671                       THEN "USER"
4672                       ELSE "EACH", "  ", "?L=", LOOKAHEAD, "MONITOR=",
4673                         MONITOR) ;
4674                WRITE("?7=", TEST,"?Q=", QUICK,"?T=",
4675                  TOURNAMENT,"?E=",EXAMINE, "?V=",DEBUG, "ENDGAME=",ENDGAME)
4676                  ;
4677                WRITE("CAPTURE=",CAPTURE_TREE, "OPENING=",OPEN(-1), OPEN(1))
4678                  ;
4679              END ;
4680            COMMENT     "o" ;
4681              IF (MOVECOUNT>0)
4682              THEN PRINTSHORTBRD(-1) ;
4683            COMMENT     "p" ;
4684              BEGIN
4685                IF QUICK
4686                THEN QUICK := LOOKAHEAD := TEST := FALSE
4687                ELSE QUICK := LOOKAHEAD := TRUE
4688              END ;
4689            COMMENT     "q" ;
4690              BEGIN
```

```
4691            I := I+1 ;
4692            REINITIALIZE(UU(I|1)) ;
4693          END ;
4694        COMMENT    "Ri" ;
4695          ;
4696          ;
4697          ;
4698          ;
4699          ;
4700          ;
4701          ;
4702          ;
4703          GAMERECORD (FALSE) ;
4704        COMMENT     "s" ;
4705          TOURNAMENT := ~ TOURNAMENT ;
4706        COMMENT     "t" ;
4707          BEGIN
4708            SCORER (UU(I+1|1),UU(I+2|1),UU(I+3|1)) ;
4709            I := I+3 ;
4710          END ;
4711        COMMENT     "Usij" ;
4712          DEBUG := ~ DEBUG ;
4713        COMMENT     "v" ;
4714          FOR L := NUMBER(1) STEP -1 UNTIL 1
4715          DO PRINTLINE(L,0) ;
4716        COMMENT     "w" ;
4717          BEGIN
4718            DUMPER (DECODE(UU(I+1|1)) - 240, DECODE(UU(I+2|1)) - 240) ;
4719            I := I+2 ;
4720          END ;
4721        COMMENT   "Xij" ;
4722          BEGIN
4723  AGAIN:
4724            WRITE("Invalid data, try again") ;
4725            WRITE(MOVES_LEFT, TIME_LEFT DIV 3600) ;
4726            WRITE ("ENTER MOVES_LEFT & TIME_LEFT(MINS)") ;
4727            WHILE READU
4728            DO  TRACER(CHARCOUNT,BUFFER(0|20)) ;
4729            MOVES_LEFT := CONVERTS(U,INERR) ;
4730            IF INERR
4731            THEN GO TO AGAIN ;
4732            TIME_LEFT := CONVERTS(U,INERR) ;
4733            IF INERR OR MOVES_LEFT <= 0 OR TIME_LEFT <= 0
4734            THEN  GO TO AGAIN
4735            ELSE TIME_LEFT := TIME_LEFT * 3600 ;
4736          END ;
4737        COMMENT     "y" ;
4738          IF (MOVECOUNT>0)
4739          THEN BEGIN
4740            PUNCHBRD(TRUE, HISTORY(0|140)) ;
4741            WRITECARD(HISTORY(0|114)) ;
4742          END ;
4743        COMMENT     "z" ;
4744          ;
4745          ;
4746          ;
4747          ;
4748          ;
4749          ;
4750          BEGIN
4751            WRITE("NODES ",NOD, "TERMINAL ",TOT, "DYNAMIC ", DYN,"CAPTURE ",CAP) ;
4752            WRITE("SALVAGE", SAL, "DUPLIC", DUP, "RECLAIM", REC,
4753              "BRANCH ",BRAN) ;
4754            WRITE("DUP_TREE ", DUP_S, "REC_TREE ", REC_S,
4755              "BRAN_SCORED ", BRAN_S) ;
4756            FOR LEV := 1 UNTIL MAX_PLIES
4757            DO BEGIN
4758              IF REFUT(LEV,0) > 0
4759              THEN WRITE(COLOR(IF LEV REM 2 = 1
4760                               THEN K
```

```
4761                              ELSE -K), " REFUTATIONS ") ;
4762              FOR J := 1 UNTIL REFUT(LEV,0)
4763              DO WRITEON(REFUT(LEV,J), REFCNT(LEV,J)) ;
4764            END ;
4765            IOCONTROL(2) ;
4766          END ;
4767        COMMENT    "0" ;
4768          DUMPP :=  ~ DUMPP ;
4769        COMMENT     "1" ;
4770          WS := ~ WS ;
4771        COMMENT     "2" ;
4772          THINKING := ~ THINKING;
4773        COMMENT     "3" ;
4774          BEGIN
4775            IF ~ CBOTH
4776            THEN IF (BOTH)
4777                 THEN BOTH := WHO := FALSE
4778                 ELSE BOTH := WHO := CBOTH := TRUE ;
4779          END ;
4780        COMMENT     "4" ;
4781          BEGIN
4782            IF (CBOTH)
4783            THEN CBOTH := BOTH := FALSE
4784            ELSE IF BOTH
4785                 THEN WHO := FALSE
4786                 ELSE BOTH := TRUE ;
4787          END ;
4788        COMMENT     "5" ;
4789          GO TO ENTRY ;
4790        COMMENT     "6" ;
4791          TEST := ~ TEST ;
4792        COMMENT     "7" ;
4793          WRITE("$COPY WITA:COMMANDS(3,4) for options") ;
4794        COMMENT WRITECARD("1",
4795        " ?A -Toggle BOOK, to create the opening book.",
4796        " ?B -Print Black's moves.",
4797        " ?C -Toggle COKO flag, to play another computer.",
4798        " ?D -Dump first moves of principal variations.",
4799        " ?E -Toggle EXAMINE flag, build opponent's tree.",
4800        " ?F -Toggle ECHO flag, to echo input.",
4801        " ?G -Print game tree of principal variations.",
4802        " ?H -History.",
4803        " ?I -Instructions.",
4804        " ?J -Toggle CAPTURE_TREE option.",
4805        " ?K -Kill game, restart another game.",
4806        " ?L -Toggle LOOKAHEAD flag, build game tree to depth 5 ply.",
4807        " ?M -Toggle MONITOR flag, print secondary trees.",
4808        " ?N -Next output to unit 4 or 6.",
4809        " ?O -Print current settings of flags.",
4810        " ?P -Print board from Black's viewpoint.",
4811        " ?Q -Toggle QUICK lookahead flag, game tree to depth 3 ply.",
4812        " ?Ri -Retract last i+1 half moves, 0 <= i <= 9.",
4813        " ?S -Print record of current game.",
4814        " ?T -Toggle TOURNAMENT flag, game tree to depth 7 ply.",
4815        " ?Usij -Print snapshot of White(s=+)/Black(s=-) move",
4816        "        list, from moves 6i+1 to 6(j+1), 0 <= i <= j <= 9.",
4817        " ?V -Toggle DEBUG flag, linear/nonlinear scoring function.",
4818        " ?W -Print White's moves.",
4819        " ?Xij -Print square control information, for squares on",
4820        "       rows i to j,  1 <= i <= j <= 8.",
4821        " ?Y -Set time control parameters.",
4822        " ?Z -Print board in quasi Forsythe notation.",
4823        " ?1 -Toggle DUMPP flag, to dump various information.",
4824        " ?2 -Toggle WS flag, to compute opponent's scores.",
4825        " ?3 -Print board from White's viewpoint.",
4826        " ?4 -Toggle CBOTH flag, computer will play one more side.",
4827        " ?5 -Toggle BOTH flag, user will play one more side.",
4828        " ?6 -Terminate game, restart whole program.",
4829        " ?7 -Toggle TEST flag, to provide more debugging information.",
4830        " ?8 -Print this list of control options.",
```

```
4831        " ?9 -Print King's squares, piece count, squares",
4832        "     controlled by Kings, and the en prise squares.",
4833        " STOP -Terminates the program.", "1") ;
4834        COMMENT     "8" ;
4835          BEGIN
4836            WRITE (DIFFVAL,TWOVAL,KINGSQ(-1), KINGSQ(1)) ;
4837            FOR J := K_SQRS(-1) UNTIL K_SQRS(1)
4838            DO WRITEON (KSQRS(J)) ;
4839            WRITE(LOSS(0)) ;
4840            FOR J := 0 UNTIL LOSS(0)
4841            DO WRITEON (SQUARE(J)) ;
4842            WRITE(LOSS(ELIMIT)) ;
4843            FOR J := LOSS(ELIMIT) UNTIL ELIMIT
4844            DO WRITEON(SQUARE(J)) ;
4845            WRITE(ENDPLAY) ;
4846            FOR J := -1 UNTIL 1
4847            DO WRITEON(CLEARPAWN(J)) ;
4848            FOR J := 1 UNTIL ENDPLAY
4849            DO WRITEON(REPEATS(J)) ;
4850            FOR J := WQRSQ UNTIL BKRSQ
4851            DO IF J REM BRDWIDTH = 1
4852               THEN WRITE(CHECK_S(-1,J)+CHECK_S(1,J))
4853               ELSE IF BRD(J) ~= EDGE
4854                    THEN WRITEON(CHECK_S(-1,J)+CHECK_S(1,J)) ;
4855            FOR J := WQRSQ UNTIL BKRSQ
4856            DO IF J REM BRDWIDTH = 1
4857               THEN WRITE(FORKS(-K,J))
4858               ELSE IF BRD(J) ~= EDGE
4859                    THEN WRITEON(FORKS(-K,J)) ;
4860            IOCONTROL(2) ;
4861            FOR I := -8 UNTIL 8
4862            DO WRITEON(PIECES(I)) ;
4863            WRITEON(MEN(1), MEN(-1)) ;
4864          END ;
4865        COMMENT     "9" ;
4866        END ;
4867      END ;
4868  COMMENT    NEW ;
4869      IF ECHO AND ~OLDECHO
4870      THEN WRITE(BUFFER)
4871      ELSE WRITE("CONTINUE") ;
4872    END OFTRACER ;
4873
4874  @TITLE,  "USEBOOK"
4875    PROCEDURE USEBOOK ;
4876    BEGIN
4877      STRING(140) BUFF ;
4878      INTEGER I, LEN, N, H, R, J ;
4879      INTEGER ARRAY OFFSETS(0::10) ;
4880      LOGICAL SPECIAL ;
4881      N := -HALFP ;
4882      HASH := NEWHASH(BRD,K,KEY) ;
4883      SPECIAL := FALSE ;
4884  LOOP:
4885      TRANSFER(BUFF,LEN,16384+2,HASH,2, -1) ;
4886      IF LOOKAHEAD THEN BOOK := 0;
4887      IF (LEN <= 0)
4888      THEN WRITE(HASH, KEY(0|2), " No more book")
4889      ELSE IF (BUFF(0|2) = KEY(0|2))
4890           THEN BEGIN
4891             OFFSETS(0) := J := 3 ;
4892             R := 0 ;
4893             FOR I := 7 UNTIL LEN-1
4894             DO IF BUFF(I|1) = "!"
4895                THEN BEGIN
4896                  IF SPECIAL
4897                  THEN R := R +1
4898                  ELSE R := 1 ;
4899                  SPECIAL := TRUE ;
4900                  OFFSETS(R-1) := J ;
```

```
4901            END ELSE IF ~SPECIAL AND BUFF(I|1) = "?"
4902                     THEN R := R -1
4903                     ELSE IF (BUFF(I|1) = ".")
4904                          THEN IF SPECIAL
4905                               THEN J := I +1
4906                          ELSE BEGIN
4907                            R := R + 1 ;
4908                            OFFSETS(R) := J := I + 1 ;
4909            END ;
4910            IF TEST OR R = 0
4911            THEN BEGIN
4912              FOR I := 0 UNTIL R
4913              DO WRITEON(OFFSETS(I)) ;
4914              FOR I := 3 UNTIL LEN-1
4915              DO WRITEON(BUFF(I|1)) ;
4916              WRITE("HASH IS ", HASH, " KEY IS ", KEY(0|2)) ;
4917            END ;
4918            IF R > 0
4919            THEN BEGIN
4920              COMMENT move till period ;
4921              R := IF R <= 1
4922                   THEN OFFSETS(0)
4923                   ELSE OFFSETS((TIME(-1) DIV 60) REM R) ;
4924              I := 0 ;
4925              WHILE BUFF(I + R |1) ~= "."
4926              DO BEGIN
4927                U(I) := BUFF(I + R |1) ;
4928                I := I + 1 ;
4929              END ;
4930              U(I) := "." ;
4931              CHARCOUNT := I ;
4932              LISTMOVES ;
4933              BOOK := -1 ;
4934              REPLY := 0 ;
4935              IF ~GETMOVE(U)
4936              THEN BEGIN
4937                WRITE(HASH,"BOOK AMBIGUOUS  ",BUFF(R|6), R, BUFF(0|20))
4938                  ;
4939                GO TO STARTS ;
4940              END ;
4941     END;
4942          END ELSE
4943          BEGIN
4944            COMMENT Even though the quadratic residue method was
4945              used previously, implementation of the Zobrist method,
4946              which distributes the initial indices more uniformly
4947              across the table, makes use of the simpler linear
4948              resolution ;
4949            HASH := HASH + K ;
4950            IF TEST
4951            THEN WRITEON(KEY," Book conflict ",HASH,BUFF(0|2)) ;
4952            GO TO LOOP ;
4953          END ;
4954   END OFUSEBOOK ;
4955
4956 @TITLE,"CREATEBOOK"
4957   PROCEDURE CREATEBOOK ;
4958   BEGIN
4959     INTEGER LINE, LEN, BUFLEN, COUNT, R, TLEN ;
4960     STRING(140) BUFF, BUFR ;
4961     STRING(2) TERMINATOR ;
4962    COMMENT STRING(1) ARRAY MC(0::11) ;
4963 READIN:
4964     TRANSFER(BUFF, LEN, 32, LINE, 1, -1) ;
4965     IF (LEN < 0)
4966     THEN BOOK := -BOOK
4967     ELSE BEGIN
4968       BUFF(LEN|1) := " " ;
4969       R := COUNT := 0 ;
4970       WHILE BUFF(R|1) = " "
```

```
4971        DO R := R + 1 ;
4972        IF BUFF(R|1) < "0" OR BUFF(R|1) > "9" OR R >= LEN
4973        THEN GO TO READIN ;
4974        WHILE (BUFF(R|1) ~= " ")
4975        DO BEGIN
4976          COUNT := 10 * COUNT + DECODE(BUFF(R|1)) - 240 ;
4977          R := R + 1 ;
4978        END ;
4979        COUNT := COUNT + COUNT ;
4980        IF LEN < 13 OR BUFF(8|4) ~= "    "
4981        THEN COUNT := COUNT - 1 ;
4982        WHILE (BUFF(R|1) = " ")
4983        DO R := R + 1 ;
4984        WHILE (BOOKMOVE - COUNT > 9)
4985        DO BEGIN
4986          REINITIALIZE("9") ;
4987          BOOKMOVE := BOOKMOVE - 10 ;
4988        END ;
4989        IF (BOOKMOVE >= COUNT)
4990        THEN REINITIALIZE(CODE(BOOKMOVE - COUNT + 240)) ;
4991        BOOKMOVE := COUNT ;
4992  NEWMOVE:
4993        COMMENT try to handle two moves per line ;
4994        HASH := NEWHASH(BRD, K, KEY) ;
4995        TLEN := COUNT := 0 ;
4996        WHILE (BUFF(R|1) ~= " " AND R < LEN)
4997        DO BEGIN
4998          MC(COUNT) := BUFF(R|1) ;
4999          IF (BUFF(R|1) = "!" OR BUFF(R|1) = "?" OR BUFF(R|1) = "+")
5000          THEN BEGIN
5001            TERMINATOR(TLEN|1) := BUFF(R|1) ;
5002            TLEN := TLEN + 1 ;
5003          END ;
5004          COUNT := COUNT + 1 ;
5005          R := R + 1 ;
5006        END ;
5007        MC(COUNT) := "." ;
5008        CHARCOUNT := COUNT ;
5009        LISTMOVES ;
5010        IF ~GETMOVE(MC)
5011        THEN BEGIN
5012          WRITE("Illegal move at line ",LINE, R, BUFF(0|20)) ;
5013          GO TO QUIT ;
5014        END ;
5015        FINDRTMV ;
5016        IF NULLMOVE
5017        THEN BEGIN
5018          WRITE("NULLMOVE AT LINE ",LINE) ;
5019          GO TO QUIT ;
5020        END ;
5021  RDBOOK:
5022        TRANSFER(BUFR, BUFLEN, 16386, HASH, 2, -1) ;
5023        IF (BUFLEN > 0)
5024        THEN BEGIN
5025          IF KEY(0|2) ~= BUFR(0|2)
5026          THEN BEGIN
5027            WRITE("PRIMARY CONFLICT AT HASH ", HASH) ;
5028            HASH := HASH+K ;
5029            GO TO RDBOOK ;
5030          END ;
5031        END ELSE
5032        BEGIN
5033          BUFLEN := 3 ;
5034          BUFR(0|2) := KEY(0|2) ;
5035          BUFR(2|1) := " " ;
5036        END ;
5037        IF TEST
5038        THEN BUFR(BUFLEN|20) := " " ;
5039        BUFR(BUFLEN|2) := BOX(MOVEFROM(RTMV)) ;
5040        BUFR(BUFLEN+2|2) := BOX(MOVETO(RTMV)) ;
```

```
5041        COMMENT IF TEST  THEN
5042          WRITE("HASH: ",HASH," BUFF: ",BUFR(0|60)) ;
5043        FOR I := 1 UNTIL TLEN
5044        DO BUFR(BUFLEN+3+I|1) := TERMINATOR(I-1|1) ;
5045        BUFR(BUFLEN+TLEN+4|1) := "." ;
5046        COUNT := 3 ;
5047        IF (BUFLEN > 0)
5048        THEN WHILE (COUNT < BUFLEN)
5049             DO BEGIN
5050               IF (BUFR(COUNT|4) = BUFR(BUFLEN|4))
5051               THEN BEGIN
5052                 IF (TLEN > 0 AND BUFR(COUNT+4|1) ~= BUFR(BUFLEN+4|1))
5053                 THEN WRITE("DUPLICATE AT LINE ", HASH, LINE,
5054                   ".   BUFFER IS ", BUFR(0|60)) ;
5055                 GO TO FOUNDUP ;
5056               END ;
5057               WHILE BUFR(COUNT|1) ~= "."
5058               DO COUNT := COUNT + 1 ;
5059               COUNT := COUNT + 1 ;
5060        END ;
5061  WRITE("New Move: ", HASH, BUFR(BUFLEN|5));
5062        BUFLEN := BUFLEN + TLEN + 5 ;
5063        TRANSFER(BUFR, BUFLEN, 16386, HASH, 2, 1) ;
5064  FOUNDUP:
5065        FINALIZE ;
5066        WHILE BUFF(R|1) = " " AND R < LEN
5067        DO R := R + 1 ;
5068        IF R >= LEN OR R >= 20
5069        THEN GOTO READIN ;
5070        BOOKMOVE := BOOKMOVE + 1 ;
5071        GOTO NEWMOVE ;
5072      END ;
5073  QUIT:
5074      PRINTSHORTBRD(-1) ;
5075    END OFCREATEBOOK ;
5076    PROCEDURE INIT_TREE;
5077    BEGIN
5078      OPENING := MOVECOUNT <= 16;
5079      OPEN(1) := OPEN(-1) := OPENING;
5080      EARLY := MOVECOUNT < 8;
5081      MAXL := MAXTREE;
5082      MAXNODES := 1;
5083    ORIGIN := 0;
5084      FOR L := 0 UNTIL MAXTREE
5085      DO BEGIN
5086        PARENT(L) := -1;
5087        GETNODE(L) := L;
5088        NODETO(L,0) := -FIVES;
5089      END;
5090      GETNODE(MAXTREE+1) := 0;
5091      PREVSCR := OLDSCR := FIVES;
5092      LASTORIGIN := SAVEL := 0 ;
5093      FOR LEV := 1 UNTIL MAX_PLIES
5094      DO REFUT(LEV,0) := REFCNT(LEV,0) := 0 ;
5095      M_TIMES(0) := M_TIMES(MODE) DIV 3 ;
5096    END OFINIT_TREE ;
5097
5098    PROCEDURE BLITZ ;
5099    BEGIN
5100      WRITE("BLITZ mode") ;
5101      MODE := 3 ;
5102      INIT_TREE ;
5103      MAXLEV := 3 ;
5104      MAX_PLIES := MAXLEV + MAXPLY ;
5105      MAX_WID := WIDTH := 9 ;
5106      INIT_TREE ;
5107    END BLITZ ;
5108
5109    PROCEDURE INTERMEDIATE ;
5110    BEGIN
```

```
5111      WRITE("INTERMEDIATE mode") ;
5112      MODE := 4 ;
5113      INIT_TREE ;
5114      MAX_PLIES := 5 + MAXPLY ;
5115      IF MAX_PLIES > DEPTH_LIMIT
5116      THEN MAX_PLIES := DEPTH_LIMIT ;
5117      MAXLEV := MAX_PLIES - MAXPLY ;
5118      MAX_WID := 12 ;
5119      IF MAX_WID > WIDTH_LIMIT
5120      THEN MAX_WID := WIDTH_LIMIT ;
5121      WIDTH := MAX_WID ;
5122      INIT_TREE ;
5123    END INTERMEDIATE ;
5124
5125    PROCEDURE EXPERIMENTAL ;
5126    BEGIN
5127      WRITE("EXPERIMENTAL mode") ;
5128      MODE := 6 ;
5129      INIT_TREE ;
5130      MAX_PLIES := 9 + MAXPLY ;
5131      IF MAX_PLIES > DEPTH_LIMIT
5132      THEN MAX_PLIES := DEPTH_LIMIT ;
5133      MAXLEV := MAX_PLIES - MAXPLY ;
5134      MAX_WID := 19 ;
5135      IF MAX_WID > WIDTH_LIMIT
5136      THEN MAX_WID := WIDTH_LIMIT ;
5137      WIDTH := MAX_WID ;
5138      INIT_TREE ;
5139    END EXPERIMENTAL ;
5140
5141    PROCEDURE TOURNAMENTS ;
5142    BEGIN
5143      WRITE("TOURNAMENT mode") ;
5144      MODE := 5 ;
5145      INIT_TREE ;
5146      MAX_PLIES := 7 + MAXPLY ;
5147      IF MAX_PLIES > DEPTH_LIMIT
5148      THEN MAX_PLIES := DEPTH_LIMIT ;
5149      MAXLEV := MAX_PLIES - MAXPLY ;
5150      MAX_WID := 19 ;
5151      IF MAX_WID > WIDTH_LIMIT
5152      THEN MAX_WID := WIDTH_LIMIT ;
5153      WIDTH := MAX_WID ;
5154      INIT_TREE ;
5155    END TOURNAMENTS ;
5156
5157  @TITLE,"LISTLEGALMOVES"
5158  PROCEDURE LISTLEGALMOVES (LOGICAL VALUE LOSING) ;
5159  COMMENT  only legal moves are listed ;
5160  BEGIN
5161    INTEGER NM ;
5162    KTEST := 0 ;
5163     COMMENT    enprise makes a list of pieces that are attacked
5164     and provides some incentive to move them. ;
5165     COMMENT   do not penalize q move ;
5166    NM := ABS(CON(-K,KINGSQ(K))) ;
5167    INCHECK := IF (NM=0)
5168               THEN FALSE
5169               ELSE TRUE ;
5170    CHECKSQ := IF INCHECK
5171               THEN CONTROL(KINGSQ(K),-K)
5172               ELSE HOLE ;
5173     COMMENT only used when incheck is true ;
5174    DBLCHECK := IF (NM < 2)
5175                THEN FALSE
5176                ELSE TRUE ;
5177    LISTMOVES ;
5178    IF INCHECK
5179    THEN UPPER := EIGHTS ;
5180     COMMENT   generates all potential moves. ;
```

```
5181    ONEVAL := DIFFVAL*K ;
5182    REALVAL := ONEVAL + TWOVAL ;
5183     COMMENT inc. enprise piece ;
5184    IF SCORES
5185    THEN SELECTRTMV (LOSING) ;
5186  END OFLISTLEGALMOVES ;
5187
5188  @TITLE,"MAKE(INTEGER VALUE I) "
5189  COMMENT ********************make*************** ;
5190  PROCEDURE MAKE(INTEGER VALUE I) ;
5191  BEGIN
5192    INTEGER PC, SQ, L1, KK ;
5193    PC := BRD(I) ;
5194     COMMENT CAPTURE_CH(I) := FALSE;
5195    KK := IF (PC > 0)
5196          THEN 1
5197          ELSE -1 ;
5198     COMMENT IF DEBUG THEN WRITE("MAK",I,PC) ;
5199    MOVESQUARE(I) ;
5200    POS(I) := CLIST ;
5201    FOR L := 1 UNTIL CLIST
5202    DO POSITION(I,L) := CHECKLIST(L) ;
5203    FOR L := SLIST UNTIL LIM
5204    DO POSITION(I,L) := CHECKLIST(L) ;
5205    POSITION(I,0) := KLIST ;
5206    L1 := LOSS(ELIMIT) ;
5207    FOR L := 1 UNTIL KLIST
5208    DO BEGIN
5209      SQ := CHECKLIST(L) ;
5210      FIXIT(I,SQ,KK) ;
5211      IF (K*BRD(SQ) < 0)
5212      THEN BEGIN
5213        L1 := L1 -1 ;
5214        SQUARE(L1) := SQ ;
5215           COMMENT IF DEBUG THEN WRITE("MAKE",I,PC,SQ) ;
5216      END ;
5217    END ;
5218    IF ABS(PC) = KING
5219    THEN FOR L := SLIST UNTIL LIM-1
5220    DO IF PC*BRD(CHECKLIST(L)) < 0
5221       THEN BEGIN
5222         L1 := L1 -1 ;
5223         SQUARE(L1) := CHECKLIST(L) ;
5224    END ;
5225  COMMENT compensates for fundamental differences in handling King
5226  moves.  A King is said to "defend" those squares which are attacked
5227  by opponent (even though not occupied by own piece) ;
5228    LOSS(ELIMIT) := L1 ;
5229    FOR L := SLIST UNTIL LIM-1
5230    DO  FIXIT(I,CHECKLIST(L),KK) ;
5231    SEEKPAWN(I) ;
5232  END OFMAKE ;
5233
5234  @TITLE,"MAKEFINAL "
5235  PROCEDURE MAKEMOVE(INTEGER VALUE N ;
5236  LOGICAL VALUE FINAL) ;
5237  COMMENT  moves pieces and keeps track of piece count. ;
5238  BEGIN
5239    INTEGER I, TT, M, M1, T, L, K, OPAWN, PC, K_FILE, CREDITS, DEBITS,
5240      R_SQ, L_SQ, COL, GOOD, WEAK, K_PAWN, A, B, M21, M22, M31, M33,
5241      II, KSQ, SQ_K, ATTA, SQA,  PASS_PAWN, DEFN, SQ3, NEW_DIR, NODEBIT ;
5242    LOGICAL ROOKMOVE, PP_CHANGE, DELIBERATE ;
5243    INTEGER ARRAY C, D(0::15) ;
5244     COMMENT credits/debits C(0...15) AND D(0...15) EXPLAINED IN
5245             awit.lib(15000) ;
5246
5247    PROCEDURE MAKEFINAL ;
5248    BEGIN
5249      MATE := IF TT >= 5
5250              THEN TRUE
```

```
5251              ELSE FALSE ;
5252      STALEM := IF ~GIVINGCH AND TT >= 1
5253                THEN TRUE
5254                ELSE FALSE ;
5255      HASH := -NEWHASH(BRD, K, KEY) ;
5256      STACK_HASH(LEVEL) := HASH ;
5257      VALUES(0) := 0 ;
5258      LASTSQ(LEVEL) := M3 ;
5259      LASTDIR(LEVEL) := NEW_DIR ;
5260      GARDE := EARLY ;
5261      IF GARDE AND (CON(-K,KSQ-1) = 0)
5262      THEN GARDE := FALSE ;
5263        COMMENT make early game positions, in which the queen is
5264        attacked, non-quiescent ;
5265      IF (M6 ~= EMPTY) AND ~PP_CHANGE
5266      THEN BEGIN
5267        PIECES(M6) := PIECES(M6) -1 ;
5268        PIECES(0) := PIECES(0) -1 ;
5269        MEN(-K) := MEN(-K) -1 ;
5270      END ;
5271        COMMENT capture ;
5272      FLAGS := TRUE ;
5273        COMMENT  promotion increase separate. ;
5274      IF M14 ~= 0 AND ~PP_CHANGE
5275      THEN  PIECES(K_PAWN) := PIECES(K_PAWN) -1
5276      ELSE IF CASTLE(K) OR CASTLE(TWO(K))
5277           THEN IF (M4 = KING)
5278                THEN  CASTLE(K) := CASTLE(TWO(K)) := FLAGS := FALSE
5279                ELSE IF (M4 = ROOK)
5280                     THEN IF (M2 = QRSQ(K))
5281                          THEN CASTLE(TWO(K)) := FLAGS := FALSE
5282                          COMMENT  q-rook moving. ;
5283                          ELSE IF (M2 = KRSQ(K))
5284                               THEN CASTLE(K) := FLAGS := FALSE ;
5285        COMMENT  k-rook moving. ;
5286      IF CASTLING
5287      THEN CASTLE(THREE(K)) := TRUE ;
5288      IF (ABS(M6) = ROOK)
5289      THEN IF (M3 = QRSQ(-K))
5290           THEN CASTLE(-TWO(K)) := FLAGS := FALSE
5291           ELSE IF (M3 = KRSQ(-K))
5292                THEN CASTLE(-K) := FLAGS := FALSE ;
5293      IF ~PP_CHANGE
5294      THEN DIFFVAL := DIFFVAL + (VALUES(ABS(M6)) +(IF M14 ~= 0
5295                           THEN VALUES(ABS(M14))-PVAL
5296                           ELSE 0))*K ;
5297      ONEVAL := -K*DIFFVAL ;
5298        COMMENT ONEVAL needed by tree reversal ;
5299      M := ELIMIT ;
5300      TWOVAL := 0 ;
5301      IF (ATTA ~= 0)
5302      THEN IF (SACRIFICE(M3,FALSE))
5303           THEN BEGIN
5304             M := M-1 ;
5305             SQUARE(M) := M3 ;
5306                COMMENT  moving piece can be recaptured ;
5307             IF (LOSSES > TWOVAL)
5308             THEN TWOVAL := LOSSES ;
5309             LOSS(M) := VALUES(M4) ;
5310      END ;
5311      FOR I := 1 UNTIL LOSS(0)
5312      DO IF (K*BRD(SQUARE(I)) > 0)
5313         THEN IF SACRIFICE(SQUARE(I),FALSE) OR SQR(10) ~= ATTASQRS(10)
5314              THEN BEGIN
5315                M := M-1 ;
5316                SQUARE(M) := SQUARE(I) ;
5317                LOSS(M) := IF (I > SQUARE(0))
5318                           THEN LOSSES
5319                           ELSE LOSS(I) ;
5320                IF LOSS(M) > TWOVAL
```

```
5321                THEN TWOVAL := LOSS(M) ;
5322                FOR J := M+1 UNTIL ELIMIT-1
5323                DO IF (SQUARE(M) = SQUARE(J))
5324                   THEN M := M+1 ;
5325      END ;
5326      LOSS(ELIMIT) := SQUARE(ELIMIT) := M ;
5327      IF M4 = KING
5328      THEN CHECK_SQRS(K) ;
5329    END OFMAKEFINAL ;
5330
5331  @TITLE,"PAWN_MOVE"
5332    PROCEDURE PAWN_MOVE ;
5333    BEGIN
5334      INTEGER PC, TT;
5335      DEFN := K*SEC(0,M3) ;
5336      IF EARLY
5337      THEN BEGIN
5338        II := IF (K > 0)
5339              THEN 25
5340              ELSE 75 ;
5341           COMMENT encourage appropriate pawn structure so that Bishops
5342           can move ;
5343        IF (BASE = II) AND (ABS(BRD(II+2)) = PAWN) OR (BASE = II+2)
5344          AND (ABS(BRD(II)) = PAWN) OR (BASE = II-1) AND (ABS(BRD(II-3)
5345          ) = PAWN) OR (BASE = II-3) AND (ABS(BRD(II-1)) = PAWN)
5346        THEN C(1) := + (IF (M1 = PFTWO)
5347                        THEN IF ABS(BASE-II) > 1
5348                             THEN 0
5349                             ELSE 1
5350                        ELSE 1) ;
5351      END ;
5352      IF (M7 ~= 0)
5353      THEN IF PAWNS(M3) = 0
5354           THEN PROMCAPT := ABS(M7)
5355      ELSE BEGIN
5356        FOR L := -K_FILE, K_FILE
5357        DO BEGIN
5358          I := M3 +L ;
5359          WHILE (BRD(I) = 0)
5360          DO I := I +L ;
5361          PC := BRD(I) ;
5362          IF ABS(PC) = ROOK
5363          THEN C(7) := C(7) + (IF DEFN >= 0 AND K*PC < 0
5364                               THEN 1
5365                               ELSE -1) ;
5366          IF PC = K_PAWN AND (ATTA = 0 OR DEFN >= 0 AND
5367            ABS(BRD(CONTROL(M3,ATTA))) > PAWN)
5368          THEN C(2) := C(2) -2 ;
5369                    COMMENT  double-pawn debit, if not recapturable ;
5370          I := BASE +L ;
5371          WHILE (BRD(I) = 0)
5372          DO I := I +L ;
5373          PC := BRD(I) ;
5374          IF ABS(PC) = ROOK
5375          THEN C(7) := C(7) + (IF K*PC > 0
5376                               THEN 1
5377                               ELSE -1) ;
5378          IF (PC = K_PAWN)
5379          THEN C(3) := C(3) +1 ;
5380                    COMMENT undouble-pawn credit ;
5381        END ;
5382                COMMENT capture isolates/joins a pawn? ;
5383        D(1) := 0 ;
5384        FOR L := -1 , 1
5385        DO BEGIN
5386          COL := COLS(BASE) + L ;
5387          T := ISOLATE(COL, K_PAWN, -K_FILE, -K) ;
5388          IF T ~= 0 AND COL = COLS(M3)
5389          THEN BEGIN
5390            T := -2 ;
```

```
5391            C(2) := 0 ;
5392          END ;
5393          D(1) := D(1) + T ;
5394        END ;
5395        IF C(2) = 0
5396        THEN BEGIN
5397          BRD(M3) := 0 ;
5398          BRD(BASE) := K_PAWN ;
5399          D(1) := D(1) - ISOLATE(COLS(BASE), K_PAWN, -K_FILE, -K) ;
5400          BRD(M3) := K_PAWN ;
5401          BRD(BASE) := 0 ;
5402        END ;
5403      END ;
5404      IF ENDGAME OR RANKS(ROWS(M3)) = K OR PASS_PAWN = BASE OR M7 ~= 0
5405      THEN BEGIN
5406        T := IF M14 ~= 0
5407             THEN 1
5408             ELSE PASSEDPAWN(M3,K) ;
5409  COMMENT  no credit for one square advance from pawn origin,
5410           or if sacrificing then same credit as if not moving;
5411        IF M7 = 0
5412        THEN IF PAWNS(BASE) = -K AND M1 ~= PFTWO
5413             THEN T := 0
5414             ELSE IF K*SEC(0,M3) < 0
5415                  THEN BEGIN
5416                     TT := PASSEDPAWN(M3, -K);
5417                     IF T > 0 AND TT = 0
5418                     THEN T := 1
5419                     ELSE T := TT;
5420                  END;
5421        IF T > 0
5422        THEN IF PASS_PAWN = KSQ OR (ROWS(M3) -ROWS(PASS_PAWN))*K
5423          > 0 OR ROWS(M3) = ROWS(PASS_PAWN) AND (CON(-K,M3) = 0 AND
5424          CON(-K,PASS_PAWN) ~= 0 OR CON(-K,M3+K_FILE) = 0
5425          AND CON(-K,PASS_PAWN+K_FILE) ~= 0)
5426             THEN IF ~FINAL
5427                  THEN PASS_PAWN := M3
5428                  ELSE BEGIN
5429                    IF PAWNS(M3) ~= 0
5430                    THEN CLEARPAWN(K) := M3
5431                    ELSE IF STARTTHISPROBLEM(TRUE)
5432                         THEN PP_CHANGE := TRUE ;
5433                    PASS_PAWN := CLEARPAWN(K) ;
5434        END ;
5435        IF DEBUG
5436        THEN WRITE("PASS", M3, CLEARPAWN(K), PASS_PAWN, T, FINAL) ;
5437  COMMENT PASS_PAWN only updated if new primary pawn ;
5438  COMMENT  credit for pushing passed pawn ;
5439        IF T > 1
5440        THEN IF DEFN >= 0
5441             THEN BEGIN
5442               PP_STATUS := PASSEDPAWN(BASE, K) + (IF RANKS(ROWS(M3))
5443                 = -K
5444                          THEN PAWNVAL(ROWS(M3))
5445                          ELSE PAWNVAL(ROWS(BASE))) ;
5446               T := T + PAWNVAL(ROWS(M3)) ;
5447               IF FINAL
5448               THEN T := T - PP_STATUS ;
5449               IF BOTV(KING,OFFSET(M33)-OFFSET(SQ_K)) ~= 2
5450               THEN PP_STATUS := PP_STATUS DIV 2
5451               ELSE T := T DIV 2 ;
5452        END ELSE T := T DIV 2 ;
5453  COMMENT O.K. as long as OPAWN is not pinned ;
5454        IF DEBUG
5455        THEN WRITE("P-M", M3, PASS_PAWN, PP_STATUS, T,DEFN) ;
5456        IF PASS_PAWN = M3
5457        THEN D(15) := D(15) + T
5458        ELSE D(2) := T ;
5459        IF CLEARPAWN(-K) ~= SQ_K AND CLEARPAWN(-K) ~= M3 AND M4 =
5460          PAWN AND PASSEDPAWN(CLEARPAWN(-K), -K) = 0
```

```
5461        THEN BEGIN
5462          BRD(M3) := 0 ;
5463          D(2) := D(2) + PASSEDPAWN(CLEARPAWN(-K), -K) ;
5464          BRD(M3) := M5 ;
5465        END ;
5466  COMMENT  pawn capture may deny opponent passed pawn ;
5467        IF DEBUG
5468        THEN WRITE("ELIM", CLEARPAWN(-K), SQ_K, D(2), M3) ;
5469  COMMENT don't accumulate ;
5470        IF DELIBERATE
5471        THEN NODEBIT := NODEBIT + D(2) ;
5472  COMMENT debit for creating passed pawn ;
5473        A := M22-1 ;
5474        B := M22+1 ;
5475        IF (M3 = A)
5476        THEN A := A +1
5477        ELSE IF (M3 = B)
5478             THEN B := B -1 ;
5479             FOR J := 0 STEP K_FILE UNTIL (IF (M1 = PFTWO)
5480                      THEN K_FILE
5481                      ELSE 0)
5482        DO FOR I := A+J, B+J
5483        DO IF BRD(I) = OPAWN OR BRD(I+K_FILE) = OPAWN AND BRD(I) = NIL
5484           THEN BEGIN
5485             II := I - K_FILE ;
5486             IF BRD(I) = NIL OR BRD(II) ~= NIL
5487             THEN II := I ;
5488             T := PASSEDPAWN(II, -K) ;
5489             IF DEBUG
5490             THEN WRITE("PP_CRE", I, BRD(I), II, T) ;
5491             COMMENT see if opponent can create a pp.,
5492                     either by pushing or by capturing;
5493             IF BRD(I) = NIL THEN BEGIN
5494               TT := PASSEDPAWN(M3, -K);
5495               IF T > 0 AND BRD(M3+K_FILE) ~= OPAWN
5496               THEN T := T - PASSEDPAWN(M3+K_FILE, K);
5497               IF T < TT
5498               THEN T := TT;
5499             END;
5500        COMMENT does each side get equal pp potential ;
5501        COMMENT IF RANKS(ROWS(II)) = -K THEN T := T + PAWNVAL(ROWS(II)) ;
5502             IF T < 0
5503             THEN T := 0 ;
5504             D(3) := D(3) -T ;
5505        END ;
5506      END ;
5507  COMMENT Phalanx, backpawn and hole removal;
5508  IF (M1 = PFTWO OR PAWNS(BASE) ~= -K)
5509  THEN BEGIN
5510      T := 0 ;
5511      COL := ABS(COLS(M3) - COLS(KSQ)) ;
5512  COMMENT  5. Phalanx creation credit, and hole removal ;
5513      IF (~OPENING OR COL > 1)
5514      THEN T := T + (IF BRD(M3+1) = K_PAWN
5515                     THEN IF BRD(M3-1) = K_PAWN
5516                          THEN 2
5517                          ELSE 1
5518                          ELSE IF BRD(M3-1) = K_PAWN
5519                               THEN 1
5520                               ELSE 0) ;
5521  COMMENT  4.  Push backward pawns, if no phalanx credit,
5522             if in home half of board ;
5523      IF ~OPENING AND T = 0
5524      THEN IF (COL > 1 OR ENDGAME) AND (BACKPAWN(BASE, -K_FILE, K_PAWN)
5525        > 0) AND RANKS(ROWS(BASE))*K < 0
5526           THEN T := T + 1 ;
5527        COMMENT  5a.  Hole removal ;
5528      IF COL = 1
5529      THEN IF M1 = PFTWO
5530           THEN T := T + (IF BRD(M31-1) = K_PAWN AND BRD(M31+1) = K_PAWN
```

```
                          THEN 1
                          ELSE 0)
                          ELSE IF OPENING AND BRD(M3+1) = K_PAWN
                            AND BRD(M3-1) = K_PAWN
                               THEN T := T + 1 ;
    C(4) := + T ;
END;
COMMENT   debit for creating holes ;
    T := 0 ;
    FOR I := -2,0,2
    DO IF BRD(M33+I) = OPAWN AND (I = 0 OR BRD(M33) = 0)
       THEN FOR J := -1,1
       DO IF I*J >= 0
          THEN IF BRD(M31+J) = K_PAWN AND BACKPAWN(M31+J,-K_FILE,K_PAWN)
            > 0
               THEN IF SEC(0,M3+J)*K < 0
                    THEN T := T -2;
    COMMENT  V formation debit;
    IF BRD(M3-2) = K_PAWN AND BRD(M31-1) = K_PAWN
    THEN T := T -1 ;
    IF BRD(M3+2) = K_PAWN AND BRD(M31+1) = K_PAWN
    THEN T := T - 1 ;
    D(4) := T ;
  END  OFPAWN_MOVE ;
@TITLE,"CAPTURE"
  PROCEDURE CAPTURE ;
  BEGIN
    COMMENT  have captured a pawn ;
    FOR L := -1, 0, 1
    DO BEGIN
      I := M31 + L ;
      WHILE (ABS(BRD(I)) ~= PAWN) AND (BRD(I) ~= EDGE)
      DO I := I - K_FILE ;
         COMMENT capture creates passed pawn ;
      T := IF (BRD(I) = K_PAWN)
           THEN PASSEDPAWN(I, K)
           ELSE 0 ;
      IF T >  0
      THEN BEGIN
        COMMENT don't get too excited about wing pawns ;
        IF COLS(I) = 1 OR COLS(I) = 8
        THEN T := T -1 ;
        IF FINAL AND PASS_PAWN ~= KSQ AND COLS(I) ~= COLS(PASS_PAWN)
        THEN DEBITS := DEBITS + T ;
COMMENT  carry forward credits for second pp;
        IF PASS_PAWN = KSQ OR (ROWS(I) - ROWS(PASS_PAWN))*K > 0
        THEN IF FINAL
             THEN PASS_PAWN := CLEARPAWN(K) := I
             ELSE PASS_PAWN := I ;
            COMMENT if PASS_PAWN = I then new principal passed pawn ;
        IF PASS_PAWN = I
        THEN D(13) := D(13) + T
        ELSE D(5) := D(5) + T ;
            COMMENT debit opponent for secondary PP creation ;
        IF DEBUG
        THEN WRITE("NEW_PP", PASS_PAWN, I, T, BRD(I), K_PAWN) ;
      END ;
    END ;
    IF DELIBERATE
    THEN NODEBIT := NODEBIT + D(5) ;
IF ABS(ATTA) ~= 1
THEN I := 0 ELSE
    I := IF BRD(M33-1) = OPAWN
         THEN M33-1
         ELSE IF BRD(M33+1) = OPAWN
              THEN M33+1
              ELSE 0 ;
    D(6) := 0 ;
      COMMENT did we capture an isolated pawn;
    IF I = 0 AND LEVEL > 1
```

```
5601      THEN D(6) := +ISOLATE(COLS(M3), OPAWN, K_FILE, K) ;
5602      IF I ~= 0
5603      THEN BEGIN
5604        COMMENT recapture leaves opponent with isolated pawn ;
5605        BRD(I) := 0 ;
5606        BRD(M3) := OPAWN ;
5607        D(6) := -ISOLATE(COLS(M3), OPAWN, K_FILE, K) ;
5608        BRD(I) := OPAWN ;
5609        BRD(M3) := M5 ;
5610      END ;
5611        COMMENT did the capture create an isolated pawn?;
5612      IF I = 0
5613      THEN FOR L := -1, 1
5614      DO D(6) := D(6) - ISOLATE(COLS(M3)+L, OPAWN, K_FILE, K) ;
5615      II := LASTSQ(LEVEL-1) ;
5616      I := IF M3 = II
5617           THEN M33
5618           ELSE M3 ;
5619      T := 0 ;
5620      IF (RANKS(ROWS(I)) = -K)
5621      THEN T := T + PAWNVAL(ROWS(I)) ;
5622        COMMENT SACRIFICE gives credit for capturing advanced pawn ;
5623      IF M3 = CLEARPAWN(-K)
5624      THEN BEGIN
5625        T := T + PASSEDPAWN(I, -K) + PP_BLOCK(M3, -K_FILE) ;
5626        IF FINAL
5627        THEN IF STARTTHISPROBLEM(TRUE)
5628             THEN  PP_CHANGE := TRUE ;
5629      END ELSE IF CLEARPAWN(-K) ~= KINGSQ(-K)
5630               THEN T := T + PASSEDPAWN(I,-K)
5631               ELSE IF (ENDGAME OR T >= 3) AND (LEVEL = 1 OR M3 ~= II)
5632                    THEN T := T + (IF M3 ~= II AND BRD(M31) = NIL
5633                                   THEN PASSEDPAWN(M31,-K)
5634                                   ELSE IF M4 = PAWN
5635                     THEN 0
5636                     ELSE PASSEDPAWN(I,-K)) ;
5637        COMMENT either captured principal or secondary passed pawn ,
5638                              or advanced pawn ;
5639      IF T ~= 0 AND DEBUG
5640      THEN WRITE("RID_AP", PASS_PAWN, M3, T, II, I, CLEARPAWN(-K),
5641            DELIBERATE);
5642        COMMENT don't accummulate, deliberate opponent ;
5643      D(7) := + T ;
5644      IF DELIBERATE
5645      THEN IF LEVEL = 1
5646           THEN NODEBIT := NODEBIT + T
5647           ELSE NODEBIT := NODEBIT + T ;
5648        COMMENT opponent's debit for loss of PP ;
5649      IF ABS(ATTA) <= 2
5650      THEN BEGIN
5651        COMMENT will a recapture double pawns ;
5652        IF (BRD(M33+1) = OPAWN OR BRD(M33-1) = OPAWN) AND (BRD(M31)
5653          = OPAWN OR BRD(M33) = OPAWN OR BRD(M33+K_FILE) = OPAWN)
5654        THEN C(5) := + (IF ABS(ATTA) = 1
5655                        THEN 2
5656                        ELSE 1) ;
5657      END ;
5658        COMMENT credit for backpawn capture.
5659                No special credit for capturing isolated pawn;
5660      IF ~OPENING AND BACKPAWN(M3, K_FILE, OPAWN) > 0
5661        AND ISOLATE(COLS(M3), PAWN, K_FILE, K) = 0
5662      THEN D(11) := 1 ;
5663    END OFCAPTURE ;
5664
5665  @TITLE,"SPECIAL_MOVE"
5666    PROCEDURE SPECIAL_MOVE ;
5667    BEGIN
5668      IF (M1 = QSIDE) OR (M1 = KSIDE)
5669      THEN BEGIN
5670        COMMENT   m8 is the new square of the rook ;
```

```
5671          COMMENT   m9 is the old square of the rook ;
5672        IF (M1 = QSIDE)
5673        THEN BEGIN
5674          COMMENT  castle q-side ;
5675          M8 := R_SQ ;
5676          L := L_SQ ;
5677          M9 := L_SQ-1 ;
5678              COMMENT   queen rook ;
5679        END ELSE IF (M1 = KSIDE)
5680                 THEN BEGIN
5681                   COMMENT  castle k-side. ;
5682                   M8 := L_SQ ;
5683                   L := M3 ;
5684                   M9 := R_SQ ;
5685                       COMMENT   king rook ;
5686        END ;
5687        M10 := BRD(M8) := BRD(M9) ;
5688        CASTLING := TRUE ;
5689        UNMAKE(M9) ;
5690        IF ~ENDGAME
5691        THEN BEGIN
5692          FOR J := L, L-1, L+1
5693          DO BEGIN
5694            GOOD := WEAK := 0 ;
5695            FOR I := 1,2,3
5696            DO IF (BRD(J+I*K_FILE) = K_PAWN)
5697               THEN GOOD := 1 ;
5698            FOR I := 4,5,6
5699            DO IF (BRD(J+I*K_FILE) = OPAWN)
5700               THEN WEAK := -1 ;
5701            GOODCASTLE := GOODCASTLE + ( IF (WEAK = 0)
5702                     THEN -GOOD
5703                     ELSE GOOD - WEAK) ;
5704                 COMMENT goodcastle is incremented by -1, 0, 1, or 2 ;
5705            IF (J = L) AND (GOODCASTLE = 0)
5706            THEN GOODCASTLE := -1 ;
5707          END ;
5708          GOODCASTLE := IF (GOODCASTLE < -1)
5709                        THEN -2
5710                        ELSE IF (GOODCASTLE > 3)
5711                             THEN 2
5712                             ELSE IF (GOODCASTLE > 1)
5713                                  THEN 1
5714                                  ELSE IF (GOODCASTLE < 1)
5715                     THEN -1
5716                     ELSE 0 ;
5717          C(7) := C(7) + OPENFILE(M8) ;
5718        END ;
5719      END ELSE IF ~ENDGAME AND (M4 = KING)
5720               THEN GOODCASTLE := IF CASTLE(K) ~= CASTLE(TWO(K))
5721                                  THEN -1
5722                                  ELSE IF CASTLE(K)
5723                     THEN -2
5724                     ELSE 0
5725                     ELSE IF (M1 = ENPRIS)
5726                          THEN BEGIN
5727                            M9 := M31 ;
5728                            UNMAKE(M9) ;
5729                          COMMENT m9 is the square of the enpassant
5730                          pawn. ;
5731                            M6 := M10 := OPAWN ;
5732      END ELSE IF (M1 = PFTWO)
5733               THEN BEGIN
5734                 COMMENT board edge takes care of things ;
5735                 IF (BRD(R_SQ) = OPAWN) OR (BRD(L_SQ) = OPAWN)
5736                 THEN BEGIN
5737                   M8 := M31 ;
5738                   BRD(M8) := -M12 ;
5739                   REMEMBER := TRUE ;
5740                       COMMENT m8 is a potential ep. sq ;
```

```
5741                  IF (BRD(R_SQ) = OPAWN)
5742                  THEN REMAKE(R_SQ,R_SQ) ;
5743                  IF (BRD(L_SQ) = OPAWN)
5744                  THEN REMAKE(L_SQ,L_SQ) ;
5745                END ;
5746                COL := COLS(M3) ;
5747                IF OPENING AND ABS(BRD(M33)) ~= PAWN
5748                THEN BEGIN
5749                  IF ((COL = 8) AND (CASTLE(K) OR EARLY)) OR ((COL = 1)
5750                    AND (CASTLE(TWO(K)) OR EARLY))
5751                  THEN C(9) := IF ABS(COL - COLS(KINGSQ(-K))) > 2
5752                               THEN -2
5753                               ELSE 0 ;
5754                       COMMENT  discourage P-R4 ;
5755                END ;
5756                IF ~ENDGAME
5757                THEN IF COL ~= 5 AND COL = COLS(KSQ)
5758                     THEN C(10) := -1 ;
5759    END ;
5760  END OFSPECIAL_MOVE ;
5761
5762 @TITLE "PASSED PAWN SITUATIONS"
5763  PROCEDURE OPPON_PP ;
5764  BEGIN
5765    I := PP_BLOCK(CLEARPAWN(-K),-K_FILE) ;
5766    T := PASSEDPAWN(CLEARPAWN(-K), -K) ;
5767    IF DEBUG
5768    THEN WRITE("PROMOTE", CLEARPAWN(-K), I, T) ;
5769    IF T > 7 OR T = 7 AND I <= 0
5770    THEN I := I - (IF GIVINGCH
5771                   THEN 1
5772                   ELSE 2)*(T-6) ;
5773    IF I > 0
5774    THEN T := T + 2 ;
5775   COMMENT promotion imminent ;
5776    D(14) := I - (IF GIVINGCH OR M6 ~= 0 AND M4 >= ROOK
5777                  THEN T DIV 2
5778                  ELSE T) ;
5779  END OF_OPPON_PP ;
5780
5781  PROCEDURE COMPU_PP ;
5782  BEGIN
5783      INTEGER LL;
5784    IF ROOKMOVE
5785    THEN C(14) := IF COLS(M3) = COLS(PASS_PAWN)
5786                  THEN 1
5787                  ELSE IF COLS(BASE) = COLS(PASS_PAWN)
5788                       THEN -1
5789                       ELSE 0 ;
5790    I := PASS_PAWN ;
5791    II := I + FILES(K) ;
5792    L := IF PAWNS(I) ~= 0
5793         THEN SEC(0,II)*K
5794         ELSE -1;
5795    LL := K*SEC(0,I);
5796    T := PASSEDPAWN(I, K) ;
5797    IF LL < 0 AND (~GIVINGCH OR L < 0)
5798    THEN T := T DIV 2 ;
5799    IF T > 0 AND BRD(II) = 0 AND (L >= 0 OR LL < 0)
5800    THEN C(15) := + 1 + L ;
5801    IF T = 7 AND PAWNS(I) = K AND (L >= 0 OR LL = -K AND
5802      HIDDEN(II,I, I, K, FALSE))
5803    THEN T := 9 ;
5804    IF PASS_PAWN = M3
5805    THEN IF LL >= 0
5806         THEN T := T      COMMENT was T -2;
5807         ELSE T := PASSEDPAWN(I-FILES(K), K);
5808    IF DEBUG
5809    THEN WRITE("COMPU_PP", PASS_PAWN, M3, T, L, LL);
5810    IF D(14) = 0
```

```
5811      THEN T := IF T < 3
5812                THEN T
5813                ELSE IF T > 9
5814                     THEN 10
5815                     ELSE IF T > 8
5816                          THEN 6
5817                          ELSE IF T > 7
5818                               THEN 4
5819                               ELSE 2 ;
5820      IF D(13) = 0
5821      THEN D(15) := D(15) + T ;
5822    END OF_COMPU_PP ;
5823  @TITLE,"MAKEMOVE"
5824    FOR I := 1 UNTIL 15
5825    DO C(I) := D(I) := 0 ;
5826    PP_CHANGE := FALSE ;
5827    M2 := MOVEFROM(N) ;
5828    CASTLING := FALSE ;
5829    K := IF (BRD(M2) > 0)
5830         THEN 1
5831         ELSE -1 ;
5832    PROMCAPT := M8 := M9 := M14 := M10 := 0 ;
5833    K_PAWN := K*PAWN ;
5834    OPAWN := -K_PAWN ;
5835    K_FILE := FILES(K) ;
5836    PASS_PAWN := CLEARPAWN(K) ;
5837    C(0) := -DEBIT(LEVEL-2) ;
5838    IF FINAL
5839    THEN BEGIN
5840      DEBIT(LEVEL) := DEBIT(LEVEL-1) := 0 ;
5841      PP_STATUS := 0 ;
5842      LOSS(ELIMIT) := SQUARE(ELIMIT) ;
5843      REMEMBER := TERMINAL := TRUE ;
5844      BASE := M2 ;
5845    END ;
5846    CREDIT(LEVEL) := CREDIT(LEVEL-2) ;
5847    D(0) := GOODCASTLE := BACKROW := 0 ;
5848    CREDITS := DEBITS := 0 ;
5849    MAKEMV := TRUE ;
5850    M3 := MOVETO(N) ;
5851    M5 := BRD(M2) ;
5852    M4 := ABS(M5) ;
5853    SQ_K := KINGSQ(-K) ;
5854    KSQ := IF M4 = KING
5855           THEN BASE
5856           ELSE KINGSQ(K) ;
5857    DELIBERATE := M3 = LASTSQ(LEVEL-1) ;
5858    NODEBIT := 0 ;
5859    IF (REMEMBER)
5860    THEN BEGIN
5861      T := SQUARE(0) ;
5862      L := SQUARE(ELIMIT) ;
5863      IF FINAL
5864      THEN HIDDENLOSS(N,T,L) ;
5865      M11 := 0 ;
5866      M12 := -ENPRIS*K ;
5867      L := 46 +15*K ;
5868      T := L + 7 ;
5869      FOR I := L UNTIL T
5870      DO IF (BRD(I) = M12)
5871         THEN BEGIN
5872           BRD(I) := EMPTY ;
5873           M11 := I ;
5874      END ;
5875      RLIST(0) := TLIST(0) := KSQ ;
5876      P_NUMB := 1 ;
5877      RLIST(1) := TLIST(1) := SQ_K ;
5878    END ;
5879    M6 := M7 := BRD(M3) ;
5880    NEW_DIR := IF INCHECK OR CASTLING OR M7 ~= 0 OR M4 = PAWN
```

```
5881                THEN 0
5882                ELSE ABS(BOTV(EDGE, OFFSET(BASE) -OFFSET(M3))) ;
5883    IF BASE = OLD_DEST AND NEW_DIR = OLD_DIR
5884    THEN C(13) := -1 ;
5885     COMMENT tempo loss, inline move;
5886    NUMB := T_NUMB := P_NUMB ;
5887    M1 := ABS(REWARD(N)) ;
5888    IF (M1 < CHECKING)
5889    THEN TT := 0
5890    ELSE BEGIN
5891      TT := M1 DIV CHECKING ;
5892      M1 := M1 REM CHECKING ;
5893    END ;
5894    ROOKMOVE := IF (M4 = ROOK) AND (ROWS(BASE) = ROWS(M3))
5895                THEN TRUE
5896                ELSE FALSE ;
5897    IF M4 = ROOK AND ~ROOKMOVE AND K*SEC(0,M3) > 0
5898    THEN BEGIN
5899      IF ABS(ROWS(M3) - ROWS(SQ_K)) = 1
5900      THEN D(8) := + 1 ;
5901      IF ABS(ROWS(BASE) - ROWS(SQ_K)) = 1 AND ROWS(M3) ~= ROWS(SQ_K)
5902      THEN D(8) := D(8) - 1 ;
5903    END ;
5904    UNMAKE(M2) ;
5905     COMMENT   moving piece ;
5906    IF (M7 ~= 0)
5907    THEN UNMAKE(M3) ;
5908     COMMENT   captured piece ;
5909    IF (M1 > PFTWO)
5910    THEN BEGIN
5911      M1 := M1 - PROMOTE ;
5912      BRD(M3) := M14 := (M1)*K ;
5913        COMMENT   promoted piece ;
5914      PIECES(M14) := PIECES(M14) +1 ;
5915    END ELSE BRD(M3) := M5 ;
5916    IF (M3 = M11)
5917    THEN M7 := M12 ;
5918    IF ROOKMOVE
5919    THEN C(8) :=  DOUBLE(M3, FILE, K) - DOUBLE(BASE, FILE, K)
5920    ELSE IF ENDGAME AND M4 = ROOK AND COLS(BASE) = COLS(M3)
5921         THEN C(8) := DOUBLE(M3, 1, K) - DOUBLE(BASE, 1, K) ;
5922     COMMENT GIVING UP FILE? ;
5923    M33 := M3 + K_FILE ;
5924    M22 := BASE +K_FILE ;
5925    M31 := M3 - K_FILE ;
5926    M21 := BASE -K_FILE ;
5927      COMMENT   M33
5928                M3
5929                M31
5930                     used primarily to locate blocked pawns.
5931                M22
5932                BASE
5933                M21  ;
5934    R_SQ := M3+1 ;
5935    L_SQ := M3-1 ;
5936    IF M1 > KING
5937    THEN SPECIAL_MOVE ;
5938    FOR L := CON(-1,M2) UNTIL CON(1,M2)
5939    DO IF (L ~= 0)
5940       THEN REMAKE(CONTROL(M2,L),M2) ;
5941    IF REMEMBER
5942    THEN P_NUMB := NUMB ;
5943    REMEMBER := TERMINAL ;
5944     COMMENT   only increment numb if terminal move ;
5945    FOR L := CON(-1,M3) UNTIL CON(1,M3)
5946    DO IF (L ~= 0)
5947       THEN REMAKE(CONTROL(M3,L),M3) ;
5948    IF (M10 = OPAWN) OR (CASTLING)
5949    THEN
5950    BEGIN
```

```
5951    COMMENT   enpassant pawn ;
5952      FOR L := CON(-1,M9) UNTIL CON(1,M9)
5953      DO IF (L ~= 0)
5954         THEN REMAKE(CONTROL(M9,L),M9) ;
5955      IF (CASTLING)
5956      THEN BEGIN
5957        COMMENT   ex defence of rooks square ;
5958        FOR L := CON(-1,M8) UNTIL CON(1,M8)
5959        DO IF (L ~= 0)
5960           THEN REMAKE(CONTROL(M8,L),M8) ;
5961        MAKE(M8) ;
5962      END ;
5963    END ;
5964    MAKEMV := FALSE ;
5965    MAKE(M3) ;
5966    IF ROOKMOVE
5967    THEN C(7) := C(7) - OPENFILE(BASE) + OPENFILE(M3) ;
5968  COMMENT capture of a pawn may open, or half open, a file(+1).
5969          Also, doubled rooks may exist.
5970             Note a minor piece may capture a pawn and open
5971             a file which contains a rook/queen;
5972    IF M7 = OPAWN THEN
5973    IF M4 = ROOK AND COLS(M3) = COLS(BASE)
5974    THEN C(8) := C(8) + DOUBLE(M3, FILE, K) +1
5975    ELSE FOR J := -10, 10 DO BEGIN
5976      SQ3 := M3;
5977      PC := 0;
5978      WHILE ABS(PC) ~= EDGE AND PC ~= PAWN
5979      DO BEGIN
5980        SQ3 := SQ3 + J;
5981        PC := K*BRD(SQ3);
5982        IF PC = ROOK
5983        THEN C(8) := C(8) + 1;
5984      END;
5985    END;
5986    IF M4 >= BISHOP AND M4 <= QUEEN
5987    THEN HIDATTACK ;
5988    IF (M4 = KING)
5989    THEN BEGIN
5990      IF FINAL  AND (PASS_PAWN = BASE)
5991      THEN CLEARPAWN(K) := M3 ;
5992      KINGSQ(K) := M3 ;
5993      KINGCONTROL(K) ;
5994    END ;
5995        COMMENT  REMAKE(KSQ,KSQ) ;
5996    REMAKE(SQ_K,SQ_K) ;
5997    ATTA := CON(-K,M3) ;
5998    IF (M4 = PAWN)
5999    THEN PAWN_MOVE ;
6000    IF (M7 = OPAWN) OR (ABS(M7) = ENPRIS)
6001    THEN CAPTURE ;
6002    IF OPENING
6003    THEN BEGIN
6004      COMMENT penalize early queen moves ;
6005      IF EARLY
6006      THEN IF (M4 = QUEEN) AND M1 = 0 AND (CON(-K,BASE) = 0) AND ~INCHECK
6007           THEN C(11) := -3 ;
6008           IF M4 = KING AND ~CASTLING AND ~INCHECK
6009           THEN C(11) := -1 ;
6010        COMMENT is the King forced to recapture;
6011      IF M7 ~= 0 AND CON(-K,M3) = -K AND CONTROL(M3,-K) = KINGSQ(-K)
6012      THEN C(11) := C(11) + 1 ;
6013      BACKROW := IF (M4 = BISHOP OR M4 = KNIGHT) AND ROWS(BASE) =
6014                                  (IF K > 0
6015                                   THEN 1
6016                                   ELSE 8)
6017                 THEN AGITATE +1
6018                 ELSE AGITATE ;
6019    END ;
6020      IF M4 = KING AND ~CASTLING AND ~ENDGAME
```

```
6021      THEN IF (CASTLE(K) OR CASTLE(2*K))
6022           THEN C(11) := -2;
6023     COMMENT can one threaten a checking fork?;
6024    IF M4 > PAWN AND M4 < KING
6025    THEN FOR I := 1 UNTIL POS(M3)
6026    DO IF FORKS(-K,POSITION(M3,I)) = M5
6027       THEN C(11) := C(11) + 3 ;
6028  COMMENT influence on pawn structures ;
6029  COMMENT give pawn structure credits ;
6030  COMMENT  1. Block opponent's pawn, fill hole with N, P, or B ;
6031    IF (BRD(M33) = OPAWN) AND (ATTA = 0  OR SEC(0,M3)*K >= 0)
6032    THEN BEGIN
6033      T := BACKPAWN(M33, K_FILE, OPAWN) ;
6034      IF T < 0 AND M4 ~= KNIGHT
6035      THEN T := 0 ;
6036      IF (T > 0) AND (M4 <= BISHOP)
6037      THEN T := IF M4 = PAWN AND BACKPAWN(M3, -K_FILE, K_PAWN) > 0
6038                THEN 0
6039                ELSE T ;
6040      D(9) := T ;
6041     COMMENT - fix opponent's hole;
6042      T := D(4) ;
6043      IF M4 = PAWN OR M4 = BISHOP OR M4 = QUEEN
6044      THEN FOR J := -1, 1
6045      DO IF BRD(M33+K_FILE+J) = OPAWN AND BRD(M33+J) = EMPTY
6046         THEN IF NEW_DIR ~= ABS(M33+J-M3)
6047              THEN BEGIN
6048  COMMENT can pawn advance safely;
6049                II := K*CON(0,M33+J);
6050                IF II > 1 OR
6051                   II = 1 AND ~HIDDEN(M33+J, M33+J+K_FILE, SQ3, -K, FALSE)
6052                THEN T := T + 2;
6053              END;
6054      D(4) := T ;
6055    END ;
6056  COMMENT  3.  Don't block own pawns after opening phase Don't block
6057  central pawn anytime ;
6058    IF ~INCHECK AND M1 = 0 AND BRD(M31) = K_PAWN AND (COLS(M3) =
6059      4 OR COLS(M3) = 5 OR ~EARLY AND ROWS(BASE) ~= ROWS(KRSQ(K))
6060      AND BACKPAWN(M31, -K_FILE, K_PAWN) > 0)
6061    THEN D(10) := -1 ;
6062    T := 0 ;
6063  COMMENT  6. Unblock own backward pawns after opening phase ;
6064    IF ~EARLY AND ~INCHECK AND (BRD(M21) = K_PAWN) AND
6065      (BACKPAWN(M21,-K_FILE, K_PAWN) > 0)
6066    THEN T := T + 1 ;
6067  COMMENT 2. Don't unblock opponent's backward pawns ;
6068    IF ~INCHECK AND M22 ~= M33 AND BRD(M22) = OPAWN AND
6069        K*CON(0,BASE) > 0 AND
6070      RANKS(ROWS(M22)) = K AND (BACKPAWN(M22, K_FILE, OPAWN) > 0)
6071    THEN T := T -1 ;
6072    D(11) := D(11) + T ;
6073  COMMENT penalty for giving up fix on opponent's hole;
6074    IF (BRD(M22) = OPAWN) AND (M4 = BISHOP OR M4 = QUEEN)
6075    THEN FOR J := -1, 1
6076    DO IF BRD(M22+K_FILE+J) = OPAWN AND BRD(M33+J) = EMPTY
6077       THEN IF NEW_DIR ~=  ABS(M22+J-BASE)
6078            THEN IF SEC(0,M22+J)*K = 0
6079                 THEN D(4) := D(4) - 2 ;
6080  COMMENT   moving piece ;
6081    I := ABS(CON(K,SQ_K)) ;
6082    IF (I = 0)
6083    THEN GIVINGCH := FALSE
6084    ELSE BEGIN
6085      GIVINGCH := TRUE ;
6086     COMMENT don't count hidden pieces ;
6087      DOUBLECH := IF (I = 2)
6088                  THEN TRUE
6089                  ELSE FALSE ;
6090      II := IF (I = 1) AND (M3 ~= CONTROL(SQ_K,K))
```

```
6091            THEN 3
6092            ELSE IF DOUBLECH
6093                 THEN 4
6094                 ELSE I+1 ;
6095     COMMENT  castling with check yields dis. ch. ;
6096      IF M14 ~= 0
6097      THEN M1 := M1 + PROMOTE ;
6098      REWARD(N) := M1 + (II)*CHECKING ;
6099     COMMENT dis. or dbl. check ;
6100    END ;
6101  COMMENT penalize tempo wasting moves in opening ;
6102  COMMENT discourage successive moves with same piece ;
6103    IF M14 ~= 0
6104    THEN IF DEFN < 0 AND ~HIDDEN(M3, CONTROL(M3,-K), SQ3, K, FALSE)
6105         THEN C(6) := -20
6106         ELSE D(12) := IF ATTA = 0 AND LENGTH > LEVEL
6107                       THEN LENGTH -LEVEL +1
6108                       ELSE 2 ;
6109    FOR I := 0 UNTIL 12
6110    DO DEBITS := DEBITS + D(I) ;
6111    IF FINAL
6112    THEN DEBIT(LEVEL-1) := DEBIT(LEVEL-1) + DEBITS +C(7) DIV 2 -
6113      NODEBIT ;
6114    IF CLEARPAWN(-K) ~= SQ_K AND M3 ~= CLEARPAWN(-K)
6115    THEN OPPON_PP ;
6116    IF PASS_PAWN ~= KSQ
6117    THEN COMPU_PP ;
6118    DEBITS := DEBITS + D(13) + D(14) + D(15) ;
6119    PP_STATUS := D(15) + D(2) ;
6120    FOR I := 0 UNTIL 15
6121    DO CREDITS := CREDITS + C(I) ;
6122    IF DEBUG
6123    THEN BEGIN
6124      I_W := 1 ;
6125      WRITE(N, "CRDB ", CREDITS, DEBITS, CREDIT(LEVEL)) ;
6126      FOR I := 0 UNTIL 15
6127      DO WRITEON(C(I), D(I)) ;
6128      WRITEON(PASS_PAWN, KSQ, SQ_K, CLEARPAWN(-K), PP_STATUS, NODEBIT);
6129      I_W := 4 ;
6130    END ;
6131    CREDITS := CREDITS + GOODCASTLE + DEBITS ;
6132  COMMENT IF LEVEL > 2 AND CREDITS > LEVEL THEN CREDITS := CREDITS
6133  - LEVEL DIV 2 ;
6134    CREDIT(LEVEL) := CREDIT(LEVEL) + CREDITS -(IF FINAL
6135                       THEN D(13)+D(14)+D(15)+C(15)+C(14)
6136                       ELSE 0) ;
6137  COMMENT   moving piece ;
6138    REVERSIBLE := IF (M7 ~= 0) OR (M4 = PAWN) OR CASTLING
6139                  THEN FALSE
6140                  ELSE TRUE ;
6141  COMMENT castling is also non-reversible ;
6142    IF (FINAL)
6143    THEN MAKEFINAL ;
6144    IF INCHECK
6145    THEN CHECK_SQRS(K) ;
6146    INFLUENCE(FINAL);
6147  END OFMAKEMOVE ;
6148
6149  PROCEDURE GAMETREE (INTEGER VALUE ROOT, LEVEL, WIDTH) ;
6150  BEGIN
6151    INTEGER NODE, LIMIT ;
6152    IF ROOT <= 0
6153    THEN WRITE("game", ROOT)
6154    ELSE BEGIN
6155      LIMIT := NODETO(ROOT,0) ;
6156      IF LIMIT > WIDTH
6157      THEN LIMIT := WIDTH ;
6158      IF LIMIT <= 0 OR LEVEL > 2
6159      THEN LIMIT := 1 ;
6160      FOR J := 1 UNTIL LIMIT
```

```
6161      DO BEGIN
6162        RESTORESTATE(ROOT) ;
6163        PRINTLINE (TRY(J),LEVEL) ;
6164        NODE := NODETO(ROOT,J) ;
6165        WRITEON(ROOT, NODETO(ROOT,0)) ;
6166        IF (NODE >= 1) AND (NODE <= MAXTREE)
6167        THEN BEGIN
6168          IF (ROOT ~= PARENT(NODE) REM 1000) AND
6169                PARENT(NODE) <= PARENT(ROOT)
6170          THEN WRITEON("DUPLICATE", NODE) ;
6171               WRITEON(NODE, PARENT(NODE), NODETO(NODE,0)) ;
6172               GAMETREE(NODE,LEVEL+1, WIDTH) ;
6173               RESTORESTATE(ROOT) ;
6174        END;
6175      END ;
6176    END ;
6177  END OFGAMETREE ;
6178
6179  @TITLE,"MOBILITYSCORE(INTEGER RESULT BEST) "
6180  COMMENT ************mobilityscore************ ;
6181
6182  PROCEDURE MOBILITYSCORE(INTEGER RESULT BEST) ;
6183  COMMENT  scores on mobility and defence.
6184   This whole chess program is poorly organized and inefficient.
6185    a complete re-writing is probably better than continued
6186   evolution.
6187   Too much reliance has been placed on global variables,
6188   in the interests of reduced parameter passing ;
6189  BEGIN
6190    INTEGER ARRAY EPLIST(0::ELIMIT) ;
6191    INTEGER MV, SAMV, JJ, SQ, SVMV, REWARDN, PC, JN, I, II, GAINS,
6192      EPGAIN, EP, LAST_SQ, NEXTSQ, KINGSQU, CAPT, EPL, SPAR4, SPAR5,
6193      EPSQ,  SQA, SAVEB, KINGATTACKER, SQJJ, TMP, J, ATTACKSCORE,
6194      DEFENCESCORE, N, KSQ, ENPR, MF, NN, MATER, K_FILE, K_PAWN,
6195      SSQA, PLACESCORE, SPARE3, IP, SPARE1, SPARE2 ;
6196    LOGICAL SACRIFICING, PIN, DISC_ATTA, LEGALMV, STOP, MADELEGAL,
6197      TRAPPED, KINGMOVING, FORCING, EXCHANGE, CAPT_CH, DUMMY2 ;
6198
6199  @TITLE,"CHECKER "
6200  PROCEDURE CHECKER ;
6201  BEGIN
6202    INTEGER SQ, PC, SQA, I, SAVE_D, LOST_PC, VAL_ATTA, ATTA, DEFN;
6203     COMMENT does it mate? ;
6204    MATE := TRUE ;
6205    SQA := KINGATTACKER ;
6206    VAL_ATTA := VALUES(ABS(BRD(SQA))) ;
6207    IF DOUBLECH
6208    THEN GO TO DONE ;
6209     COMMENT protect against rare mate by enpassant pawn ;
6210    IF (REWARDN = PFTWO)
6211    THEN IF (BRD(SQA-K_FILE) = ENPRIS*K)
6212         THEN SQA := SQA - K_FILE ;
6213    SAVE_D := BOTV(EDGE, OFFSET(SQA)-OFFSET(KSQ)) ;
6214    IF SAVE_D = 0
6215    THEN SAVE_D := KSQ-SQA ;
6216    ATTA := I := CON(-K,SQA) ;
6217    LOST_PC := KINGVAL ;
6218         WHILE (SQA ~= KSQ) AND MATE
6219         DO BEGIN
6220           COMMENT  is the defending or interposing piece pinned? ;
6221           DEFN := CON(K,SQA);
6222           WHILE MATE  AND (I ~= 0)
6223           DO BEGIN
6224             COMMENT try each blocker for SQ to SQA;
6225             SQ := CONTROL(SQA,I) ;
6226             PC := -K*BRD(SQ) ;
6227             IF PC > 0 THEN
6228             IF SQA = KINGATTACKER
6229             THEN MATE := FALSE
6230             ELSE IF PC < KING AND PC > PAWN
```

```
6231                  THEN IF ABS(ATTA)-1 > ABS(DEFN)
6232                       THEN MATE := FALSE
6233                       ELSE IF ABS(ATTA) = 2
6234                       THEN MATE := HIDDEN(KSQ, SQA, TMP, K, FALSE)
6235                       ELSE IF PC > KNIGHT
6236                            THEN MATE := FALSE;
6237             IF ~ MATE
6238             THEN IF (KSQ = SQ) AND (DEFN ~= 0)
6239                  THEN MATE := TRUE
6240                  ELSE IF SQA ~= KINGATTACKER
6241                       THEN BEGIN
6242                         PC := VALUES(PC) ;
6243                         IF ABS(ATTA) ~= 1
6244                         THEN PC := PC - VAL_ATTA ;
6245                         IF PC < LOST_PC
6246                         THEN LOST_PC := PC ;
6247                         MATE := TRUE ;
6248                        COMMENT probable mate ;
6249             END ;
6250             IF DEBUG
6251             THEN WRITE("MAT", MATE,LOST_PC,PC,I, KSQ, SQ, SQA, BRD(SQ),
6252               TMP, BRD(TMP),  SAVE_D,  K) ;
6253             I := I + K ;
6254          END ;
6255       COMMENT is there an interposing pawn move? ;
6256          SQA := SQA + SAVE_D ;
6257          I := CON(-K,SQA) ;
6258          PC := BRD(SQA+K_FILE) ;
6259          IF (PC = -K_PAWN) OR ((PC = NIL) AND (PAWNS(SQA+2*K_FILE)
6260            = K_PAWN))
6261          THEN BEGIN
6262            I := I - K ;
6263          COMMENT note con(k,sqa) is not changed ;
6264            CONTROL(SQA,I) := SQA + K_FILE*(IF (PC=NIL)
6265                      THEN 2
6266                      ELSE 1) ;
6267          END ;
6268    END ;
6269    IF MATE AND LOST_PC < KINGVAL
6270    THEN BEGIN
6271      MATE := FALSE ;
6272      IF LOST_PC > 0
6273      THEN BEGIN
6274        GAINS := GAINS + LOST_PC ;
6275        BACKROW := BACKROW + (IF GAINS < -REALVAL
6276                              THEN -(REALVAL +GAINS) + DELTA
6277                              ELSE 4) ;
6278      END ;
6279    END ;
6280  DONE:
6281    IF DEBUG
6282    THEN WRITE("MATE =",MATE,KSQ,KINGATTACKER, SQA,LOST_PC,GAINS) ;
6283    IF MATE
6284    THEN REWARD(N) := REWARDN + 5*CHECKING ;
6285  END OFCHECKER ;
6286
6287  @TITLE,"COLLECTDATA "
6288  PROCEDURE COLLECTDATA ;
6289  BEGIN
6290    INTEGER T ;
6291    JN := 4*(2*N - K) ;
6292    STORE(JN) := EPGAIN ;
6293    STORE(JN+1) := SAMV ;
6294    STORE(JN+2) := SVMV ;
6295    STORE(JN+3) := MV ;
6296    STORE(JN+4) := GAINS +CREDIT(LEVEL) + BACKROW ;
6297    STORE(JN+5) := ENPR ;
6298    STORE(JN+6) := ATTACKSCORE ;
6299    STORE(JN+7) := IF ~EARLY
6300                   THEN DEFENCESCORE
```

```
6301                    ELSE PLACESCORE ;
6302    IF DEBUG
6303    THEN BEGIN
6304      WRITE("COL",N,OCON,CCON,ONUM,CNUM,"#") ;
6305      FOR JJ := 1 UNTIL EPLIST(0)
6306      DO WRITEON(SQUARE(EPLIST(JJ))) ;
6307      WRITEON("#",EPSQ,"#") ;
6308      FOR JJ := EPLIST(ELIMIT) UNTIL ELIMIT-1
6309      DO WRITEON(SQUARE(EPLIST(JJ))) ;
6310    END ;
6311    T := IF (EPGAIN < 0) AND (EPGAIN > -PVAL)
6312         THEN 0
6313         ELSE EPGAIN ;
6314    TMP := IF GIVINGCH
6315           THEN T
6316           ELSE 0 ;
6317    IF GIVINGCH
6318    THEN T := 0 ;
6319    IF TEST AND EXAMINE
6320    THEN WRITE(N,OCON,CCON,ONUM,CNUM,ONEVAL,GOODCASTLE, ATTACKSCORE,
6321      DEFENCESCORE,EP,GAINS,SAMV, ENPR,T,TMP,SVMV,MV+SAMV) ;
6322  END OFCOLLECTDATA ;
6323
6324  @TITLE,"ENPRISLIST"
6325  PROCEDURE ENPRISLIST ;
6326  BEGIN
6327    INTEGER ARRAY OLDLOSS(0::ELIMIT) ;
6328    INTEGER T, TOP, SQ3, SQH, PCH, J ;
6329      TOP := LOSS(0);
6330    EPL := 1 ;
6331    EPLIST(1) := TOP ;
6332    T := LOSS(ELIMIT) ;
6333    IF PC >= BISHOP AND PC <= QUEEN
6334    THEN FOR I := POSITION(SQ,LIM) UNTIL LIM-1
6335         DO BEGIN
6336           COMMENT hidden attack on otherwise safe piece? ;
6337           SQH := POSITION(SQ,I) ;
6338           PCH := ABS(BRD(SQH)) ;
6339           IF PCH ~= QUEEN AND PCH ~= PC
6340           THEN SQ3 := HOLE
6341           ELSE IF HIDDEN(SQ, SQH, SQ3, 0, FALSE)
6342                THEN ;
6343           IF SQ3 ~= HOLE AND K*BRD(SQ3) < 0
6344           THEN BEGIN
6345             T := T -1 ;
6346             SQUARE(T) := SQ3 ;
6347           END ;
6348    END ;
6349    FOR JJ := 1 UNTIL TOP
6350    DO BEGIN
6351      OLDLOSS(JJ) := LOSS(JJ) ;
6352        COMMENT order of FOR loop is important;
6353      SQJJ := SQUARE(JJ) ;
6354        IF SQJJ < WQRSQ OR SQJJ >BKRSQ
6355        THEN WRITE("LIST", SQUARE(0), LOSS(0), I, TOP, SQJJ) ELSE
6356      IF (BRD(SQJJ) ~= EMPTY)
6357      THEN BEGIN
6358        IF ~SACRIFICE(SQJJ,DEBUG)
6359        THEN IF (LOSSES < -SEVENS) AND (SQJJ ~= KINGATTACKER)
6360             THEN GO TO NEXTJJ ;
6361           COMMENT piece unlikely to be captured ;
6362        EP := LOSSES ;
6363      END ELSE IF GIVINGCH OR CON(-K,SQJJ) = 0
6364               THEN GO TO NEXTJJ
6365      ELSE BEGIN
6366        EP := LOSS(JJ)-VALUES(ABS(BRD(SQ))) ;
6367        IF DEBUG
6368        THEN WRITE("EMPTY", SQJJ, EP, LOSS(JJ)) ;
6369                   COMMENT may be better to defend forked piece ;
6370        IF (EP < PVAL) OR (ABS(BRD(SQ)) ~= PAWN) OR SQJJ = CLEARPAWN(K)
```

```
6371        THEN GO TO NEXTJJ ;
6372        ATTASQRS(0) := 0 ;
6373      END ;
6374      FOR I := 1 UNTIL ATTASQRS(0)
6375      DO BEGIN
6376        FOR J := T UNTIL ELIMIT-1
6377        DO IF SQUARE(J) = ATTASQRS(I)
6378           THEN GOTO NEXTI ;
6379        T := T -1 ;
6380        SQUARE(T) := ATTASQRS(I) ;
6381        COMMENT consider all counter attacks ;
6382  NEXTI:
6383      END ;
6384      I := 0 ;
6385      WHILE (I < EPL)
6386      DO BEGIN
6387        I := I + 1 ;
6388        IF (EP >= LOSS(EPLIST(I)))
6389        THEN BEGIN
6390          FOR J := EPL STEP -1 UNTIL I
6391          DO EPLIST(J+1) := EPLIST(J) ;
6392          EPLIST(I) := JJ ;
6393          LOSS(JJ) := EP ;
6394          I := EPL ;
6395        END ;
6396      END ;
6397      EPL := EPL + 1 ;
6398  NEXTJJ:
6399    END ;
6400    EPLIST(0) := EPL := EPL -1 ;
6401    IF T - TOP < 5
6402    THEN BEGIN
6403      WRITE("ENPRIS LIST ERROR", BASE, SQ, T,TOP, LOSS(ELIMIT)) ;
6404      FOR J := 1 UNTIL ELIMIT
6405      DO WRITEON(SQUARE(J)) ;
6406      PUNCHBRD(TRUE, HISTORY) ;
6407      WRITE(HISTORY(0|114)) ;
6408    END ;
6409    LOSS(ELIMIT) := T ;
6410    IF (EPL = 0)
6411    THEN EPL := 1 ;
6412    ENPR := LOSS(EPLIST(1)) ;
6413    EPLIST(ELIMIT) := ELIMIT ;
6414    EPSQ := SQUARE(EPLIST(1)) ;
6415    EPGAIN := IF FALSE AND INCHECK AND (EPLIST(0) = 0)
6416              THEN 0
6417              ELSE EPSCORE(EPSQ,ENPR) ;
6418    SACRIFICING := IF (ENPR > 0)
6419                   THEN TRUE
6420                   ELSE FALSE ;
6421    IF (ENPR <= -PVAL)
6422    THEN ENPR := 0 ;
6423    IF (EPGAIN >= ENPR)
6424    THEN SACRIFICING := FALSE ;
6425    FOR JJ := 1 UNTIL TOP
6426    DO LOSS(JJ) := OLDLOSS(JJ) ;
6427  END OFENPRISLIST ;
6428
6429  PROCEDURE PARTIAL_ORDER ;
6430  BEGIN
6431    INTEGER R, II, RR, LL, TMP, PC ;
6432    LL := 0 ;
6433    RR := TOTAL + 1 ;
6434    FOR N := K STEP K UNTIL NUMBER(K)
6435    DO BEGIN
6436      R := REWARD(N) ;
6437      IF R ~= 0 AND ABS(R) ~= PFTWO
6438      THEN II := RR := RR - 1
6439      ELSE BEGIN
6440        PC := ABS(BRD(MOVEFROM(N))) ;
```

```
6441        TMP := K*CHECK_S(K,MOVETO(N)) ;
6442        IF TMP > 0 AND CHEC(TMP, PC)
6443        THEN II := RR := RR - 1
6444        ELSE II := LL := LL + 1 ;
6445      END ;
6446      USE(II) := N ;
6447    END ;
6448    LL := LL +1 ;
6449    FOR I := RR UNTIL TOTAL
6450    DO USE(LL+I-RR) := USE(I) ;
6451    USE(0) := 0 ;
6452  END OFPARTIAL_ORDER ;
6453
6454  LOGICAL PROCEDURE KINGDEFENDER(INTEGER VALUE SQ ;
6455  INTEGER VALUE RESULT EPS) ;
6456  BEGIN
6457    INTEGER SQ1, SQ3 ;
6458    LOGICAL TRAPPED ;
6459    TRAPPED := FALSE ;
6460    FOR I := POSITION(SQ,LIM) UNTIL LIM -1
6461    DO IF ~TRAPPED
6462       THEN BEGIN
6463         COMMENT does any piece rely on this defender? ;
6464         SQ1 := POSITION(SQ,I) ;
6465         IF CON(0,SQ1) = 0  AND BRD(SQ)*BRD(SQ1) > 0
6466         THEN BEGIN
6467           TRAPPED := TRUE ;
6468           FOR J := 1 UNTIL POS(SQ)
6469           DO IF TRAPPED
6470              THEN BEGIN
6471                SQ3 := POSITION(SQ,J) ;
6472                IF ABS(BOTV(KING, OFFSET(SQ1) -OFFSET(SQ3))) = 1
6473                THEN TRAPPED := FALSE ;
6474           END ;
6475           IF TRAPPED
6476           THEN EPS := VALUES(ABS(BRD(SQ1))) ;
6477         END ;
6478         IF DEBUG
6479         THEN WRITE("KINGDEF", SQ, SQ1, I, EPS, SQ3, TRAPPED) ;
6480    END ;
6481    TRAPPED
6482  END KINGDEFENDER ;
6483
6484  @TITLE,"PINTRAP AND EPSCORE"
6485  COMMENT ***********epscore***************** ;
6486
6487  INTEGER PROCEDURE EPSCORE(INTEGER VALUE RESULT EPSQR, ENPR) ;
6488  COMMENT find the second largest piece (largest if sacrificing) ;
6489  BEGIN
6490    INTEGER I, II, J, SQA, SCR,  EPSCORE, VALSQT, VALSQA, TOP, PCA,
6491      COST, EP, SQ, SAVE_D, KSQ, SQTDEF, MINDEF, TEMP, TT, SQAA,
6492      SQ1, PC1, SQ3, PC3, VALSQ3, ATTA, PC, STAT_SQ3, ENPSEC, SC, ENP,
6493      LOST, SQT,  SQD, SQJ, DECOY, SQSAVE, NA, ND, NEW, OLD, PCAA,
6494      PPCA, KK, L, SQS, SMALL_ATTA, SQT_ATT, MAX_CAPT, TMP, PCASAVE,
6495      SQA_ATT, PCT, LOSERS, FIRST, SECOND, THIRD, OFF3, SQI, PCI,
6496      G1, G2, G3, LOSS3, SEC_EP, EPSQ2 ;
6497    INTEGER ARRAY ASQRS(ELIMIT-10::ELIMIT) ;
6498    LOGICAL   T, KINGATTA, DUMMY1, NORMAL, CAPTURE, SQT_CAPT, RELEVANT,
6499      SPECIAL, CHK, SQA_CAPT, CH_CAPT, ADD, E1, E2, E3, E4, E5 ;
6500    INTEGER ARRAY SQAXSQT(20::ELIMIT-1, 1::10) ;
6501    INTEGER ARRAY SQTXSQA(1::10, 20::ELIMIT-1) ;
6502    LOGICAL PIN, VALID, FRESH, EXTRALOSS, SACR_CAPT, SAFE, PASS1,
6503      DEFEND, V_A, S_A, RECAPT_CH ;
6504    INTEGER ARRAY HOLD(-16::16) ;
6505    INTEGER SMALL_DEF, MAJOR, OFFSSQA;
6506
6507    PROCEDURE DEFENCE ;
6508    BEGIN
6509      COMMENT seek pawn defence ;
6510      I := SSQA + FILES(K) ;
```

```
6511      TMP := KK*PAWN ;
6512      IF ~ GIVINGCH
6513      THEN FOR L := I+1, I-1
6514      DO IF TRAPPED
6515         THEN BEGIN
6516           J := L ;
6517           WHILE BRD(J) = 0
6518           DO J := J + FILES(K) ;
6519           IF BRD(J) =TMP AND (ABS(J-L) = 10 OR ABS(J-L) = 20 AND PAWNS(J)
6520             = K AND BRD(L+1) ~= -TMP AND BRD(L-1) ~= -TMP)
6521           THEN BEGIN
6522             TRAPPED := FALSE ;
6523             IF DEBUG
6524             THEN WRITEON("PAWN", J) ;
6525             EPSCORE := IF ENPR < LOST
6526                        THEN LOST
6527                        ELSE IF EPSCORE > ENPR
6528                             THEN ENPR
6529                             ELSE EPSCORE ;
6530           END ;
6531      END ;
6532      L := K*SEC(0,SSQA) ;
6533      SQI := IF CON(-K,SSQA) = 0
6534             THEN HOLE
6535             ELSE CONTROL(SSQA,-K) ;
6536      OFFSSQA := OFFSET(SSQA);
6537      RELEVANT := ~GIVINGCH OR BOTV(KING,OFFSSQA-OFFSET(KSQ)) = 2 ;
6538      IF TRAPPED AND RELEVANT
6539      THEN IF L <= 1
6540           THEN FOR I := 1,-1, 10,-10, 9,-9, 11,-11
6541                DO BEGIN
6542                  COMMENT seek Q, R, B, or K defence ;
6543                  SQ := SSQA + I ;
6544                  WHILE TRAPPED AND BRD(SQ) = 0
6545  OR SQI ~= 0 AND BRD(SQ) ~= EDGE AND K*BRD(SQ) < 0 AND
6546                  BOTV(ABS(BRD(SQ)), OFFSET(SQ)-OFFSSQA) = 1
6547                  DO BEGIN
6548                    IF K*CON(K, SQ) <= 0 AND BRD(SQ) = 0
6549                    THEN FOR J := CON(-K,SQ) STEP K UNTIL -K
6550                         DO BEGIN
6551                           SQ3 := CONTROL(SQ,J) ;
6552                           PC3 := ABS(BRD(SQ3)) ;
6553  IF DEBUG THEN WRITE("DEF", SSQA, SQ, SQ3, PC3);
6554                           OFF3 := OFFSET(SQ3) - OFFSSQA ;
6555                           TMP := OFFSET(SQ) - OFFSSQA ;
6556                           IF SQ3 ~= SQI AND SSQA ~= SQ3 AND (PC3
6557                             >= BISHOP AND PC3 <= QUEEN
6558                             AND BOTV(EDGE,OFF3) ~= BOTV(EDGE,TMP)
6559                             AND ~GIVINGCH OR PC3 = KING AND SQ =
6560                             SSQA+I AND BOTV(KING,OFF3) = 2)
6561                AND (PASS1 OR MAJOR = SQ3)
6562                           THEN BEGIN
6563                             PC := IF (PC3 = ROOK OR PC3 = BISHOP)
6564                                   THEN BOTV(PC3, TMP)
6565                                   ELSE 0 ;
6566  IF DEBUG THEN WRITEON(PC, L, SMALL_ATTA, SQI);
6567                             IF PC3 = QUEEN OR PC = 1 OR PC3 = KING
6568                             THEN IF L = 1 OR VALUES(PC3)
6569                               < VALUES(SMALL_ATTA) + DELTA
6570                                  THEN IF PC3 < QUEEN OR SQI = HOLE
6571                                    OR ABS(BRD(SQI)) < QUEEN
6572                                  COMMENT  avoid having both King
6573                                    and Queen as defenders ;
6574  THEN BEGIN
6575    IF DEBUG
6576    THEN WRITE("DEFN", SSQA, SQ, SQ3, PC3, SQI, PC, ENPR, LOST) ;
6577    TRAPPED := FALSE ;
6578  COMMENT ensure this is not a decoy situation in which
6579          the piece on SQ3 is the sole defender of another
6580          piece, one which it cannot defend from SQ;
```

```
6581    FOR I := POSITION(SQ3,LIM) UNTIL LIM-1
6582    DO BEGIN
6583       COMMENT is potential defender committed elsewhere;
6584      SQ1 := POSITION(SQ3,I) ;
6585      IF BRD(SQ1) ~= 0 AND K*SEC(0,SQ1) >= 0
6586      THEN IF BOTV(PC3, OFFSET(SQ)-OFFSET(SQ1)) ~= 1
6587           THEN TRAPPED := TRUE ;
6588    END ;
6589    IF ~TRAPPED
6590    THEN EPSCORE := IF PASS1 AND ENPR < LOST
6591                    THEN LOST
6592                    ELSE IF EPSCORE > ENPR
6593                         THEN ENPR
6594                         ELSE EPSCORE ;
6595    IF ~TRAPPED
6596    THEN GOTO OK ;
6597                              END ELSE
6598                              BEGIN
6599                                IF PASS1
6600                                THEN EPSCORE := EPSCORE DIV 2 ;
6601                                DEFEND := TRUE ;
6602                                GOTO OK ;
6603                              END ;
6604                            END ;
6605                    END ;
6606                    SQ := SQ + I ;
6607                  END ;
6608      END ;
6609  OK:
6610      IF DEBUG
6611      THEN WRITEON(TRAPPED, EPSCORE, SQI, L) ;
6612  COMMENT seek knight defence ;
6613      PC3 := KK*KNIGHT ;
6614      T := PC3 ~= BRD(SSQA) ;
6615      IF TRAPPED AND ~GIVINGCH
6616      THEN IF L <= 1 AND PIECES(PC3) ~= 0
6617           THEN FOR I := 1 UNTIL 8
6618                DO BEGIN
6619                  SQ3 := SSQA + S(I) ;
6620                  IF BRD(SQ3) = 0 AND CON(K,SQ3) = 0 AND CON(-K,SQ3) ~= 0
6621                  THEN FOR J := 1 UNTIL 8
6622                  DO IF T OR SQ3+S(J) ~= SSQA
6623                     THEN IF BRD(SQ3 + S(J)) = PC3
6624                          THEN BEGIN
6625                            COMMENT  Knight must not defend itself ;
6626                            TRAPPED := FALSE ;
6627                            IF DEBUG
6628                            THEN WRITEON("KNIGHT", SQ3) ;
6629                            EPSCORE := IF ENPR < LOST
6630                    THEN LOST
6631                    ELSE IF EPSCORE > ENPR
6632                         THEN ENPR
6633                         ELSE EPSCORE ;
6634                  END ;
6635      END ;
6636    END OFDEFENCE ;
6637
6638  PROCEDURE PINTRAP ;
6639  BEGIN
6640    IF DEBUG
6641    THEN WRITE("EPSR", SCR, EPSCORE) ;
6642    PIN := FALSE ;
6643    PASS1 := SCR > EPSCORE -DELTA ;
6644    LOST := EPSCORE ;
6645  LOOP:
6646    IF ~PASS1 AND EPSQR ~= HOLE AND SSQA ~= HOLE AND SCR > -DELTA
6647      AND ( CON(-K,SSQA) = 0 OR VALUES(ABS(BRD(EPSQR)))
6648      < VALUES(ABS(BRD(SSQA))) + DELTA)
6649    THEN FOR I := CON(K,SSQA) STEP -K UNTIL K
6650    DO IF CONTROL(SSQA,I) = EPSQR
```

```
6651       THEN ENPR := 0 ;
6652    IF ~GIVINGCH OR (SSQA ~= KSQ)
6653    THEN IF GIVINGCH OR EPSCORE > SCR OR SCR > -ENPR  AND ENPR >
6654      0 OR ~CAPTURE_CH(EPSQR) AND CAPTURE_CH(SSQA)
6655         THEN BEGIN
6656           SCR := SCR + ENPR ;
6657           EPSCORE := SCR ;
6658    END ;
6659    IF DEBUG
6660    THEN WRITEON(SSQA, SCR,EPSCORE, EPSQR, ENPR) ;
6661    IF EPSQR = HOLE AND CON(K,SSQA) = 0
6662    THEN BEGIN
6663      TRAPPED := TRUE ;
6664      EPSCORE := SCR ;
6665      IF DEBUG
6666      THEN WRITEON("EH", K, SSQA) ;
6667    END ELSE IF SCR > -DELTA OR EPSCORE > 0
6668             THEN BEGIN
6669               TRAPPED := IF (SSQA ~= KSQ) AND (SCR > 0 OR POS(SSQA)
6670                 > 0 AND SCR > -DELTA)
6671                           THEN TRUE
6672                           ELSE FALSE ;
6673               PC := ABS(BRD(SSQA)) ;
6674               VALSQA := VALUES(PC) ;
6675               MAX_CAPT := 0 ;
6676               FOR I := 1 UNTIL POS(SSQA)
6677               DO IF TRAPPED
6678                  THEN BEGIN
6679                    SQ := POSITION(SSQA,I) ;
6680                    VALSQ3 := VALUES(ABS(BRD(SQ))) ;
6681                    IF MAX_CAPT < VALSQ3
6682                    THEN MAX_CAPT := VALSQ3 ;
6683                    VALID := VALSQ3 < VALUES(PC) - DELTA ;
6684                    NA := CON(-KK,SQ) ;
6685                    ND := ABS(CON(KK,SQ)) - (IF PC = PAWN AND ABS(SQ-SSQA)
6686                      REM FILE = 0
6687                       THEN 0
6688                       ELSE 1) ;
6689                    COMMENT can the piece on SSQA move to a square
6690                    which is not attacked, or to a square which is
6691                    defended but attacked by a "greater" piece? ;
6692                    TRAPPED := FALSE ;
6693                    IF VALID AND HIDDEN(SQ, SSQA, SQ3, -KK, FALSE)
6694                    THEN BEGIN
6695                      NA := NA - KK ;
6696                      CONTROL(SQ, NA) := SQ3 ;
6697                    END ;
6698                    IF VALID
6699                    THEN FOR L := NA STEP KK UNTIL -KK
6700                    DO IF VALUES(ABS(BRD(CONTROL(SQ,L)))) < VALUES(PC)
6701                      -DELTA
6702                       THEN TRAPPED := TRUE ;
6703                    NA := ABS(NA) ;
6704                    IF VALID AND ~TRAPPED AND (NA >= ND) AND (NA
6705                      >  ND + (IF HIDDEN(SQ, SSQA, SQ3, KK,FALSE)
6706                               THEN 1
6707                               ELSE 0))
6708                    THEN TRAPPED := TRUE ;
6709                    COMMENT note pawn might not defend the move-to
6710                    square ;
6711                    IF ~TRAPPED AND PC = PAWN AND PAWNS(SQ) = 0
6712                    THEN BEGIN
6713                      EPSCORE := 0 ;
6714                      SQS := SSQA ;
6715                    END ;
6716     COMMENT  no epscore of opponent can promote ;
6717               END ;
6718             COMMENT if ?trapped, guaranteed escape ;
6719             COMMENT  does not check whether SSQA can simply be
6720             defended ;
```

```
6721                IF TRAPPED
6722                THEN IF (ABS(CON(-KK,SSQA)) = 1) AND (CONTROL(SSQA, -KK)
6723                  = EPSQR) AND (ENPR < -DELTA)
6724                     THEN TRAPPED := FALSE ;
6725                  COMMENT IF DEBUG AND TRAPPED
6726                  THEN WRITE("TRAP", SSQA, SCR, EPSCORE, TRAPPED,
6727                    KK,SQ,MAX_CAPT, CON(KK,SQ), NA, ND) ;
6728              COMMENT MAX_CAPT is the largest piece SSQA can capture ;
6729              COMMENT MAX_CAPT is the largest man PC can capture ;
6730               IF TRAPPED AND VALSQA-SCR > -DELTA AND VALSQA-SCR <
6731                 MAX_CAPT
6732               THEN SCR := VALSQA - MAX_CAPT ;
6733               IF TRAPPED
6734               THEN EPSCORE := SCR -(IF GIVINGCH OR LEVEL <= 2
6735                                     THEN 0
6736                                     ELSE IF LEVEL > 4
6737                                          THEN 2
6738                                          ELSE 1) ;
6739               IF TRAPPED AND ~GIVINGCH
6740               THEN EPSCORE := EPSCORE + (IF EPSCORE <= PVAL +PAWNVAL(2)
6741                 OR PC = PAWN
6742                      THEN 0
6743                      ELSE LOSERS - (IF ENPR > 0 OR CON(K,EPSQ) =
6744                        0 OR EPSCORE < VALUES(ROOK)
6745                                     THEN 5
6746                                     ELSE 8)) ;
6747              COMMENT determine if the "largest" piece under attack
6748              is pinned, hence change EPSCORE value ;
6749               IF DEBUG
6750               THEN WRITEON(EPSCORE, LOSERS) ;
6751               SMALL_ATTA := KING ;
6752               IF (SCR > DELTA OR EPSCORE > DELTA)
6753               THEN HIDETRAP ;
6754               IF GIVINGCH AND EPSCORE > 0 AND SSQA ~= KSQ AND EPSQR
6755                 >= WQRSQ AND ~TRAPPED
6756               THEN BEGIN
6757                 TRAPPED := TRUE ;
6758                 FOR I := CON(-K,EPSQR) STEP K UNTIL -K
6759                 DO IF EPSQR = KINGATTACKER AND SSQA = CONTROL(EPSQR,I)
6760                    THEN TRAPPED := FALSE ;
6761               END ;
6762               DEFEND := FALSE ;
6763 COMMENT is there a potential defence for a trapped piece ;
6764               IF SSQA = KSQ
6765               THEN TRAPPED := TRAPPED OR KINGDEFENDER(SSQA, EPSCORE)
6766               ELSE IF ENPR <= EPSCORE AND TRAPPED AND VALUES(SMALL_ATTA)
6767                 > VALSQA - DELTA
6768                    THEN DEFENCE ;
6769               L := K*SEC(0,SSQA) ;
6770               IF ~GIVINGCH OR SSQA = KSQ
6771               THEN IF TRAPPED AND L <= 1
6772                    THEN FOR J := CON(-KK,SSQA) STEP KK UNTIL -KK
6773                    DO BEGIN
6774                      COMMENT can attack path be blocked ;
6775                      SQ3 := CONTROL(SSQA,J) ;
6776                      DIR := BOTV(EDGE, OFFSET(SSQA)-OFFSET(SQ3)) ;
6777                      IF DIR ~= 0
6778                      THEN FOR I := SSQA+DIR STEP DIR UNTIL SQ3-DIR
6779                           DO BEGIN
6780                             II := SEC(0,I)*KK ;
6781                             VALID := II >= 0 AND BRD(I-FILES(KK))
6782                               = KK*PAWN ;
6783                             IF II > 0 OR VALID
6784                             THEN IF VALID
6785                             THEN BEGIN
6786                                TRAPPED := FALSE ;
6787                                IF DEBUG
6788                                THEN WRITEON("P BLOCK", I, SQ3,EPSCORE) ;
6789                                EPSCORE := IF ENPR < LOST
6790                                           THEN LOST
```

```
6791                                                ELSE IF EPSCORE > ENPR
6792                                                     THEN ENPR
6793                                                     ELSE EPSCORE ;
6794                              END ELSE
6795                              BEGIN
6796                                COMMENT need concept of safe block ;
6797                                IF ~PIN AND ~DEFEND
6798                                THEN EPSCORE := EPSCORE DIV 2 ;
6799                              END ;
6800                         END ;
6801                    END ;
6802     END ;
6803     IF (PASS1 OR ~GIVINGCH) AND SQS ~= SSQA AND ~TRAPPED AND LOST
6804       > -DELTA AND SQS >= WQRSQ AND SQS <= BKRSQ
6805     THEN BEGIN
6806       IF DEBUG
6807       THEN WRITE("PASS", SCR, ENPR, SEC_EP, CAPTURE_CH(EPSQR),
6808         CAPTURE_CH(EPSQ2), EPSQR, EPSQ2) ;
6809       IF ~CAPTURE_CH(EPSQR) AND ~CAPTURE_CH(EPSQ2) AND SCR > ENPR
6810         + DELTA AND SCR > SEC_EP + DELTA
6811       THEN ENPR := 0
6812       ELSE IF SEC_EP >= 0
6813            THEN ENPR := SEC_EP ;
6814           MAJOR := SSQA;
6815       SSQA := SQS ;
6816       PASS1 := FALSE ;
6817       EPSQR := EPSQ2 ;
6818       SCR := EPSCORE := LOST ;
6819       GO TO LOOP ;
6820     END ;
6821     IF PASS1 AND ~TRAPPED AND ~PIN AND ENPR < EPSCORE
6822     THEN EPSCORE := ENPR ;
6823     IF CAPT_CH AND ~TRAPPED
6824     THEN EPSCORE := -1 ;
6825   IF DEBUG THEN
6826   WRITE("ENPSEC ", EXTRALOSS, ENPR, CON(K,EPSQR), EPSQR, THIRD);
6827     ENPSEC := IF ~EXTRALOSS
6828               THEN 0
6829               ELSE IF ENPR > 0 AND (CON(K,EPSQR) = 0  OR LOST+ENPR
6830                 < -DELTA)
6831                    THEN THIRD
6832                    ELSE IF EPSQR = SQUARE(EPLIST(1)) OR GIVINGCH
6833                         THEN SECOND
6834                         ELSE FIRST ;
6835     IF ENPSEC < 0
6836     THEN ENPSEC := 0 ;
6837     IF ENPSEC > 0 AND EPSQR >= WQRSQ
6838     THEN BEGIN
6839       COMMENT no secondary loss if piece involved in exchange ;
6840       SQ := IF ENPSEC = FIRST AND SQUARE(EPLIST(1)) ~= EPSQR
6841             THEN 1
6842             ELSE IF ENPSEC = SECOND AND SQUARE(EPLIST(2)) ~= EPSQR
6843                  THEN 2
6844                  ELSE 3 ;
6845       SQ := SQUARE(EPLIST(SQ)) ;
6846         COMMENT  does EPSQR defend SQ ;
6847       FOR I := CON(K,EPSQR) STEP -K UNTIL K
6848       DO IF SQ = CONTROL(EPSQR,I)
6849          THEN ENPSEC := 0 ;
6850         COMMENT does SSQA attack SQ ;
6851       IF EPSCORE > 0
6852       THEN FOR I := CON(-K,SQ) STEP K UNTIL -K
6853       DO IF ENPSEC > 0
6854          THEN BEGIN
6855            IF SSQA = CONTROL(SQ,I)
6856            THEN ENPSEC := 0
6857            ELSE FOR J := CON(-K, EPSQR) STEP K UNTIL -K
6858            DO IF CONTROL(EPSQR,J) = CONTROL(SQ,I)
6859               THEN ENPSEC := 0 ;
6860       END ;
```

```
6861    END ;
6862    IF GIVINGCH AND ENPSEC > PVAL
6863    THEN ENPSEC := ENPSEC DIV 2 ;
6864  END OFPINTRAP ;
6865
6866  PROCEDURE HIDETRAP ;
6867  BEGIN
6868    INTEGER SQTO;
6869    FOR J := CON(-KK,SSQA) STEP KK UNTIL -KK
6870    DO IF ~PIN
6871       THEN BEGIN
6872         ATTA := CONTROL(SSQA,J) ;
6873         PCA := ABS(BRD(ATTA)) ;
6874         IF SMALL_ATTA > PCA
6875         THEN SMALL_ATTA := PCA ;
6876         IF PCA >= BISHOP AND PCA <= QUEEN
6877         THEN T := HIDDEN(ATTA, SSQA, SQ3, -KK, FALSE)
6878         ELSE SQ3 := HOLE ;
6879         PC3 := KK*BRD(SQ3) ;
6880         IF (SQ3 ~= HOLE) AND (PC3 > 0)
6881         THEN BEGIN
6882           VALSQ3 := VALUES(PC3) ;
6883           STAT_SQ3 := KK*SEC(0,SQ3) ;
6884           IF BISHOP <= PC AND PC <= QUEEN AND
6885              BOTV(PC, OFFSET(SSQA)-OFFSET(SQ3)) = 1
6886           THEN STAT_SQ3 := STAT_SQ3 -1;
6887  COMMENT cannot handle xray-double pins properly.
6888          Also, not clear that inline pins handled correctly;
6889           IF ((STAT_SQ3 <= 0) OR (PCA <= PC3)) AND (ENPR < DELTA
6890             OR VALSQ3 > VALUES(ABS(BRD(EPSQR))) -DELTA)
6891           THEN BEGIN
6892             IF (PC3 = KING)
6893             THEN PIN := TRUE ;
6894             IF DEBUG
6895             THEN WRITE("PIN", PIN, ATTA, KK,J,PCA,SSQA,SQ3,PC3,SCR,
6896               EPSCORE, EPSQR,VALSQ3,ENPR, STAT_SQ3) ;
6897             IF ~PIN
6898             THEN BEGIN
6899               COMMENT is SQ3 overdefended, and hence safe ;
6900               IF STAT_SQ3 > 0
6901               THEN VALSQ3 := VALSQ3 - VALUES(PCA) ;
6902               IF VALSQ3 > VALSQA
6903               THEN BEGIN
6904                 IF SACRIFICE(SSQA, DEBUG)
6905                 THEN VALSQ3 := LOSSES ;
6906                 TRAPPED := TRUE ;
6907               END ;
6908               IF SCR > VALSQ3
6909               THEN SCR := VALSQ3 ;
6910             END ;
6911     COMMENT  piece on SSQA not attack PCA ;
6912             IF (EPSCORE < SCR)
6913             THEN EPSCORE := SCR - (IF EPSQR = ATTA
6914                                    THEN ENPR
6915                                    ELSE 0) ;
6916             IF PIN OR EPSCORE > DELTA
6917             THEN TRAPPED := TRUE ;
6918  COMMENT can movement of SSQA defend SQ3;
6919             IF TRAPPED AND ~PIN
6920             THEN IF PC = KNIGHT AND KN_DEFENCE(SSQA, SQ3, K)
6921                   OR PC = QUEEN
6922           OR PC > KNIGHT AND BOTV(PC, OFFSET(SSQA)-OFFSET(SQ3)) = 2
6923                  THEN BEGIN
6924                    IF PC = KNIGHT THEN TRAPPED := FALSE
6925                    ELSE FOR I := 1 UNTIL POS(SSQA)
6926                         DO IF TRAPPED THEN BEGIN
6927                           SQTO := POSITION(SSQA,I);
6928                           IF CON(K,SQTO) = 0
6929                              AND BOTV(PC, OFFSET(SQTO)-OFFSET(SQ3)) = 1
6930                              AND CLEAR(SSQA,SQTO, SQ3)
```

```
6931                          THEN TRAPPED := FALSE;
6932                        END;
6933                   IF ~TRAPPED THEN
6934                   EPSCORE := IF ENPR < LOST
6935                              THEN LOST
6936                              ELSE IF EPSCORE > ENPR
6937                                   THEN ENPR
6938                                   ELSE EPSCORE ;
6939            END ;
6940            IF DEBUG
6941            THEN WRITE("PPIN", TRAPPED, PIN, EPSCORE, VALSQ3, PC,
6942              POS(SSQA)) ;
6943            IF TRAPPED AND ~PIN
6944 AND PC > PAWN AND BOTV(PC, OFFSET(SSQA)-OFFSET(KINGSQ(K))) = 2
6945            THEN FOR I := 1 UNTIL POS(SSQA)
6946                 DO BEGIN
6947                   IF TRAPPED
6948                   THEN BEGIN
6949                     SQ := POSITION(SSQA,I) ;
6950                         COMMENT can we give check from a safe
6951                           square ;
6952                     IF CON(K,SQ) = 0
6953                     THEN BEGIN
6954                       TMP := KK*CHECK_S(KK,SQ) ;
6955                       IF DEBUG
6956                       THEN WRITEON(SQ, TMP) ;
6957                       IF TMP > 0 AND CHEC(TMP, PC)
6958                       OR BOTV(PC, OFFSET(SQ)-OFFSET(KINGSQ(K))) = 1
6959                          AND CLEAR(ATTA,SQ, KINGSQ(K))
6960                       THEN TRAPPED := FALSE;
6961                       IF ~TRAPPED
6962                       THEN BEGIN
6963                         EPSCORE := IF ENPR < LOST
6964                                    THEN LOST
6965                                    ELSE IF EPSCORE>ENPR
6966                                         THEN ENPR
6967                                         ELSE EPSCORE ;
6968                         IF DEBUG
6969                         THEN WRITEON("MOVE", SQ, ENPR, LOST, EPSCORE) ;
6970                       END ;
6971                     END ;
6972                   END ;
6973            END ;
6974            IF TRAPPED
6975            THEN PIN := TRUE ;
6976          END ;
6977        END ;
6978   END ;
6979   PC := ABS(BRD(SSQA)) ;
6980   IF ~CAPT_CH AND ~TRAPPED AND SCR > DELTA
6981   THEN FOR I := POSITION(SSQA,LIM) UNTIL LIM-1
6982        DO BEGIN
6983          SQ3 := POSITION(SSQA,I) ;
6984          SAFE := FALSE ;
6985          OFF3 := OFFSET(SQ3) ;
6986    COMMENT is movement of SSQA impossible, because tied to defence
6987    of SQ3? ;
6988          IF CON(-KK,SQ3) ~= 0 AND SEC(KK,SQ3) = KK AND KK*BRD(SQ3) > 0
6989          THEN BEGIN
6990            IF PC >= BISHOP
6991            THEN FOR J := 1 UNTIL POS(SSQA)
6992            DO IF ~SAFE
6993               THEN BEGIN
6994                 SQD := POSITION(SSQA,J) ;
6995                 IF CON(K,SQD) = 0 AND BOTV(PC, OFF3-OFFSET(SQD)) = 1
6996                 THEN SAFE := TRUE ;
6997            END ;
6998            VALSQ3 := VALUES(ABS(BRD(SQ3))) ;
6999            IF SAFE
7000            THEN VALSQ3 := VALSQ3 - VALUES(ABS(BRD(CONTROL(SQ3,-KK)))) ;
```

```
7001            IF VALSQ3 > 0 AND EPSQR ~= HOLE
7002            THEN BEGIN
7003              COMMENT epscore := max( min(valsq3,scr), enpr) ;
7004              V_A := BOTV(ABS(BRD(SQ3)), OFFSET(SQ3) -OFFSET(EPSQR))
7005                = 1 ;
7006              S_A := BOTV(ABS(BRD(SSQA)), OFFSET(SSQA) -OFFSET(EPSQR))
7007                = 1 ;
7008              IF V_A
7009              THEN BEGIN
7010                V_A := FALSE ;
7011                FOR I := SEC(KK,EPSQR) STEP -KK UNTIL KK
7012                DO IF SECRET(EPSQR,I) = SQ3
7013                   THEN V_A := TRUE ;
7014              END ;
7015              IF S_A
7016              THEN BEGIN
7017                S_A := FALSE ;
7018                FOR I := SEC(KK,EPSQR) STEP -KK UNTIL KK
7019                DO IF SECRET(EPSQR,I) = SSQA
7020                   THEN S_A := TRUE ;
7021              END ;
7022              IF VALSQ3 > SCR
7023              THEN BEGIN
7024                EPSCORE := SCR ;
7025                IF ENPR > 0 AND ~V_A
7026                THEN IF ~S_A OR VALSQ3 -ENPR > SCR
7027                     THEN ENPR := 0
7028                     ELSE EPSCORE := VALSQ3 ;
7029              END ELSE
7030              BEGIN
7031                COMMENT  valsq3 < scr ;
7032                EPSCORE := VALSQ3 ;
7033                IF ENPR > 0 AND ~S_A AND SCR -ENPR > -DELTA
7034                THEN IF ~V_A OR SCR -ENPR < VALSQ3
7035                     THEN ENPR := 0
7036                     ELSE EPSCORE := SCR ;
7037              END ;
7038              TRAPPED := TRUE ;
7039              IF PC = KING
7040              THEN EPSCORE := EPSCORE +2 ;
7041     COMMENT capture in vicinity of King ;
7042            END ;
7043                          COMMENT IF DEBUG
7044                          THEN WRITE("DECOY/TRAP ", PC, SSQA, SQ3,
7045                         SCR, EPSCORE, VALSQ3, SAFE, V_A, S_A, EPSQR) ;
7046          END ;
7047    END ;
7048  END ;
7049  @TITLE,"DECOY"
7050    PROCEDURE DECOYED ;
7051    BEGIN
7052      INTEGER AT3, SQM, PCM, SQ, ENP_VAL;
7053    LOGICAL T, DISCOVER;
7054      SQD := SQA ;
7055      LOST := -1 ;
7056        COMMENT is SQA pinned ;
7057      PIN := SACR_CAPT := FALSE ;
7058      SQA_ATT := CON(-KK,SQA) ;
7059      FOR II := SQA_ATT STEP KK UNTIL -KK
7060      DO IF ~PIN
7061         THEN BEGIN
7062           COMMENT is SQA pinned against King, or other inadequately
7063             defended piece ;
7064           SQ1 := CONTROL(SQA,II) ;
7065           PC1 := ABS(BRD(SQ1)) ;
7066           IF (SQ1 ~= SQT) AND (PC1 >= BISHOP) AND (PC1 <= QUEEN)
7067           THEN BEGIN
7068             IF HIDDEN(SQ1,SQA,SQ3,KK, FALSE)
7069             THEN ;
7070             PC3 := KK*BRD(SQ3) ;
```

```
7071            IF (SQ3 ~= HOLE)
7072            THEN IF (PC3 > 0)
7073                 THEN IF (PC3 = KING)
7074                      THEN PIN := TRUE
7075                 ELSE BEGIN
7076                   AT3 := K*SEC(0,SQ3) +1 ;
7077                   IF AT3 = 0 AND PCA > PAWN AND BOTV(PCA, OFFSET(SQA)
7078                     -OFFSET(SQ3)) = 1 AND BOTV(PCA, OFFSET(SQT)
7079                     -OFFSET(SQ3)) ~= 1
7080                   THEN AT3 := 1 ;
7081                     IF DEBUG
7082                     THEN WRITEON("DECO", SQ1, SQ3, AT3 );
7083                   IF AT3 = 1 AND CONTROL(SQ3,K) = SQT
7084                   THEN AT3 := 0 ;
7085                   IF (AT3 > 0) OR (PC1 < PC3)
7086                   THEN BEGIN
7087                     DECOY := VALUES(PC3) ;
7088                     ENP := VALSQT ;
7089                     TMP := K*CHECK_S(-K,SQ3);
7090                     IF DEBUG THEN WRITE("DECO ", SQ1, SQ3, TMP, K,
7091                         CHECK_S(K,SQ3), CHEC(ABS(TMP),PC1));
7092                     IF TMP > 0 AND CHEC(TMP,PC1)
7093                     THEN ENP := 0
7094                     ELSE
7095                     IF CON(K,SQT) ~= 0
7096                     THEN IF VALSQA > DECOY+DELTA OR CAPTURE_CH(SQT)
7097                          THEN BEGIN
7098                            ENP := ENP - VALSQA ;
7099                            DECOY := 0 ;
7100                     END ;
7101                               COMMENT see XRAY pin ;
7102                     IF AT3 <= 0 AND DECOY > 0
7103                     THEN DECOY := DECOY -VALUES(PC1) ;
7104 IF DEBUG THEN WRITEON(DECOY, LOST, ENP, TMP);
7105                     IF (LOST < DECOY)
7106                     THEN BEGIN
7107                       SQD := SQ3 ;
7108                       LOST := DECOY ;
7109                     END ;
7110                     IF CAPTURE_CH(SQD)
7111                     THEN LOST := LOST + 1 ;
7112                   END ;
7113            END ;
7114          END ;
7115     END ;
7116     IF PIN
7117     THEN LOST := LOSSES
7118     ELSE BEGIN
7119       RELEVANT := SQT_CAPT := TRUE ;
7120       CH_CAPT := FALSE ;
7121       TMP := IF M4 = KING AND ~INCHECK
7122              THEN 0
7123              ELSE KK*CHECK_S(K,SQT) ;
7124       IF DEBUG
7125       THEN WRITEON(TMP, ENP) ;
7126       DISCOVER := FALSE;
7127       FOR I := -K STEP -K UNTIL CON(-K,SQA)
7128       DO BEGIN
7129         COMMENT possible discovered attack on us when opponent captures;
7130         SQ1 := CONTROL(SQA,I) ;
7131         PC1 := ABS(BRD(SQ1)) ;
7132         IF PC1 >= BISHOP AND PC1 <= QUEEN
7133         THEN BEGIN
7134           T := HIDDEN(SQ1, SQA, SQ3, K, FALSE) ;
7135           COMMENT but not an inline capture SQ3 ~= SQT,
7136          and SQ3 cannot recapture on SQT;
7137           IF SQ3 ~= HOLE AND SQ3 ~= SQT
7138       AND BOTV(ABS(BRD(SQ3)), OFFSET(SQT)-OFFSET(SQ3)) ~= 1
7139           THEN IF K*BRD(SQ3) > 0 AND CON(K,SQ3) = 0
7140                THEN BEGIN
```

```
7141                  DISCOVER := TRUE;
7142                  ENP := ENP + VALUES(ABS(BRD(SQ3))) ;
7143                END;
7144            IF DEBUG
7145            THEN WRITEON(SQ1, SQ3, ENP) ;
7146          END ;
7147        END ;
7148        IF ENP <= -DELTA
7149        THEN TMP := 0
7150        ELSE IF M4 = KING AND ABS(SQT-M3) <= 11
7151             THEN FOR I := K_SQRS(-KK) STEP KK UNTIL -KK
7152             DO IF SQT = KSQRS(I)
7153                THEN BEGIN
7154                  TMP := KK*(M3-SQT) ;
7155                  TMP := IF ABS(TMP) = 10 OR ABS(TMP) = 1
7156                         THEN 5
7157                         ELSE IF TMP < 0
7158                              THEN 4
7159                              ELSE 1 ;
7160        END ;
7161        IF TMP >0 AND CHEC(TMP, PCA)
7162        THEN BEGIN
7163          COMMENT if PCA is a Q, R or B and capture of SQT yields
7164            a check then set NORMAL false and SEC <= 0 ;
7165          IF DEBUG OR ~KINGMOVING AND ~CAPTURE_CH(SQT)
7166          THEN WRITE("CHECK CAPT. ON", SQT, PCA, KK,CHECK_S(K,SQT), N);
7167          NORMAL := FALSE ;
7168          CH_CAPT := CAPTURE_CH(SQT) := TRUE ;
7169          PCA := TMP ;
7170  COMMENT determine potential safety of the capture with check;
7171          SQD := SQA ;
7172          LOST := -1 ;
7173        END ;
7174        IF (~GIVINGCH OR ~DOUBLECH AND (SQT = KINGATTACKER) OR
7175             SQTDEF = 0 AND SQA = KSQ) AND
7176          LOST < VALSQT +DELTA AND (SQTDEF = 0 OR
7177                   VALSQA < VALSQT + DELTA OR
7178             VALSQA -MINDEF < VALSQT +DELTA AND K*SEC(0,SQT) < 0)
7179          OR DISCOVER AND ENP > -DELTA
7180        THEN BEGIN
7181          IF DEBUG
7182          THEN WRITE("YECAP", MINDEF,VALSQT,VALSQA,SQTDEF,
7183            SQT_ATT,SQA_ATT, LOST, ENP, "'") ;
7184          IF ~CH_CAPT
7185          THEN BEGIN
7186  COMMENT can the major piece on SQA be decoyed from the back rank,
7187           and hence be open to a back rank mate?;
7188  IF ABS(PCA) = ROOK OR ABS(PCA) = QUEEN THEN
7189  IF ROWS(KSQ) = ROWS(SQA) AND ROWS(SQT) ~= ROWS(SQA)
7190     AND KING_HELD(-K)
7191  THEN BEGIN
7192    COMMENT King held on back rank;
7193    SQ := IF K = 1 THEN BKRSQ-7 ELSE WQRSQ;
7194    FOR I := 0 UNTIL 7
7195    DO BEGIN
7196      IF SQ ~= KSQ AND ABS(CON(-K,SQ)) <= 1
7197                   AND CON(K,SQ) ~= 0
7198      THEN FOR J := CON(K,SQ) STEP -K UNTIL K
7199      DO BEGIN
7200        SQM := CONTROL(SQ,J);
7201        PCM := ABS(BRD(SQM));
7202        IF (PCM = ROOK OR PCM = QUEEN) AND
7203            SQT ~= SQM AND ROWS(SQM) ~= ROWS(SQA)
7204        THEN BEGIN
7205          COMMENT Q or R may go to back row - do they give check?;
7206          T := CHECK_S(-K,SQ) ~= 0;
7207          IF ~T THEN BEGIN
7208            T := HIDDEN(SQ, SQA, SQ3, -K, TRUE);
7209            T := SQ3 = KSQ;
7210          END;
```

```
7211          IF DEBUG THEN
7212          WRITE("MATE?", T, SQM, SQ);
7213          IF T THEN BEGIN
7214            LOST := VALUES(ABS(BRD(SQ)));
7215            IF CON(-K,SQA) = 0
7216            THEN LOST := LOST + VALSQA
7217            ELSE IF LOST = 0 OR ABS(CON(-K,SQA)) > 1
7218                 THEN LOST := LOST + VALSQA - VALUES(ABS(PCM))
7219                 ELSE LOST := LOST + VALSQA;
7220            IF LOST > ENP
7221            THEN LOST := ENP
7222            ELSE IF LOST < 0
7223                 THEN LOST := 0;
7224            SQD := SQ;
7225            DECOYS(SQT) := LOST;
7226          END;
7227        END;
7228      END;
7229      SQ := SQ + 1;
7230    END;
7231  END;
7232            COMMENT are there decoy possibilities, in which SQA captures
7233              SQT, but leaves a piece on SQD enprise ;
7234            FOR J := POSITION(SQA,LIM) UNTIL LIM-1
7235            DO BEGIN
7236              SQJ := POSITION(SQA,J) ;
7237              IF (BRD(SQJ)*KK > NIL)
7238              THEN IF ABS(CON(KK,SQJ)) = 1 AND CON(-KK,SQJ) ~= 0
7239                AND (CONTROL(SQJ,-KK) ~= SQT OR ABS(CON(-KK,SQJ)) > 1)
7240                   THEN BEGIN
7241                     TMP := K*CHECK_S(K,SQJ) ;
7242                     IF TMP > 0
7243                     THEN FOR I := CON(K,SQJ) STEP -K UNTIL K
7244                          DO BEGIN
7245                            PC1 := K*BRD(CONTROL(SQJ,I)) ;
7246                            IF CHEC(TMP, PC1)
7247                            THEN RECAPT_CH := TRUE ;
7248                     END ;
7249                     IF DEBUG
7250                     THEN WRITEON(SQJ, BRD(SQJ), RECAPT_CH) ;
7251                     SQI := SQT ;
7252                     OFF3 := OFFSET(SQT) - OFFSET(SQJ) ;
7253                     SAVE_D := IF PCA > KNIGHT  AND BOTV(PCA,OFF3) = 1
7254                                 THEN BOTV(EDGE, OFF3)
7255                                 ELSE 0 ;
7256                     IF SAVE_D ~= 0
7257                     THEN IF PCA = KING
7258                          THEN BEGIN
7259                            IF BOTV(KING, OFF3) = 1
7260                            THEN SQI := SQJ ;
7261                     END ELSE
7262                     BEGIN
7263                       SQI := SQI + SAVE_D ;
7264                       WHILE BRD(SQI) = 0
7265                       DO SQI := SQI + SAVE_D ;
7266                       IF SQI = SQA
7267                       THEN SQI := SQJ ;
7268                       IF DEBUG
7269                       THEN WRITEON("*", PCA, SQI, SAVE_D, SQJ) ;
7270    COMMENT does PCA defend SQJ from SQT? ;
7271                     END ;
7272                     IF (SQI ~= SQJ)
7273                     THEN BEGIN
7274                       COMMENT SQA  sole defender of SQD, and SQT
7275                         not sole attacker, and SQT, SQA, SQJ not
7276                         inline ;
7277                       PC1 := ABS(BRD(SQJ)) ;
7278                       DECOY := VALUES(PC1) ;
7279                       IF  (PC1 = PAWN)
7280                       THEN BEGIN
```

```
7281                          IF ENDGAME
7282                          THEN DECOY := DECOY + PASSEDPAWN(SQJ,KK) ;
7283                          IF (RANKS(ROWS(SQJ)) = KK)
7284                          THEN DECOY := DECOY + PAWNVAL(ROWS(SQJ)) ;
7285                        END ;
7286                        IF (LOST < DECOY)
7287                        THEN BEGIN
7288                          LOST := DECOY ;
7289                          SQD := SQJ ;
7290                          SPECIAL := FALSE ;
7291                          DECOYS(SQT) := LOST;
7292                        END ;
7293                      END ;
7294              END ;
7295            END ;
7296  COMMENT if SQA x SQT then does the recapture by SQ1
7297          lead to a decoy loss on SQ3? ;
7298        ENP_VAL := 0;
7299            FOR I := K STEP K UNTIL CON(K,SQT)
7300            DO BEGIN
7301              SQ1 := CONTROL(SQT,I) ;
7302        ENP_VAL := 0;
7303              FOR J := POSITION(SQ1,LIM) UNTIL LIM-1
7304              DO BEGIN
7305                SQ3 := POSITION(SQ1,J) ;
7306                IF SQ3 ~= SQT
7307                THEN IF ABS(CON(K,SQ3)) = 1 AND CON(-K,SQ3) ~= 0
7308                     THEN BEGIN
7309                       VALSQ3 := VALUES(ABS(BRD(SQ3))) ;
7310                       IF VALSQ3 < VALSQA - DELTA
7311                       THEN ENP_VAL := ENP_VAL + VALSQ3
7312                       ELSE ENP_VAL := ENP_VAL + VALSQA ;
7313        IF DEBUG THEN WRITEON(ENP_VAL);
7314                END ;
7315              END ;
7316        IF ENP_VAL = 0 THEN GOTO OK;
7317            END ;
7318        ENP := ENP + ENP_VAL;
7319  OK:
7320            NORMAL := TRUE ;
7321            IF (LOST ~= -1)
7322            THEN IF LOST > LOSSES AND (VALSQT < LOST - DELTA)
7323                 THEN BEGIN
7324                   IF DEBUG AND ABS(SQT_ATT) = 1
7325                   THEN WRITEON(LOST, SQD) ;
7326                   SQD := SQA ;
7327                   LOST := LOSSES ;
7328                   IF ABS(SQT_ATT) = 1
7329                   THEN ENP := LOSS(EPLIST(JJ)) := 0 ;
7330            END ;
7331          END ;
7332        END ELSE
7333        BEGIN
7334          IF DEBUG
7335          THEN WRITE("NOCAP", MINDEF,VALSQT,VALSQA,SQTDEF,
7336            SQT_ATT,SQA_ATT,ENP) ;
7337          CH_CAPT := FALSE ;
7338      SQT_CAPT := FALSE;
7339          SQD := SQA ;
7340          LOST := IF LOSSES < -SEVENS OR LOSSES > -DELTA
7341                  THEN LOSSES
7342                  ELSE 0 ;
7343          IF ENP <= -PVAL AND -ENP < LOST AND VALSQT-VALSQA-ENP > -DELTA
7344          THEN BEGIN
7345            SACR_CAPT := TRUE ;
7346          SQT_CAPT := TRUE;
7347            LOST := -ENP ;
7348            ENP := 0 ;
7349          END ELSE IF (CONTROL(SQA, SQA_ATT) = SQT AND ENP > DELTA)
7350                   THEN LOST := VALSQT -VALSQA;
```

```
7351        END ;
7352        SQA := SQD ;
7353      END ;
7354      IF LOST = -1 AND ENP < -DELTA
7355      THEN LOST := LOSSES ;
7356      IF DEBUG
7357      THEN WRITEON(LOST) ;
7358    END  OFDECOYED ;
7359
7360  @TITLE "BIGGER_LOSS"
7361    PROCEDURE BIGGER_LOSS ;
7362    BEGIN
7363    LOGICAL SQ3_ATTA, SQ3_DEF;
7364      FOR LL := TT UNTIL ELIMIT-1
7365      DO BEGIN
7366        COMMENT even though SQA captures SQT, an alternative loss
7367          may exist ;
7368        L := EPLIST(LL) ;
7369        SQ3 := SQUARE(L) ;
7370        IF SQA = SQ3 OR SQT_CAPT AND SQ3 = SQSAVE
7371        THEN GOTO U ;
7372        IF (SQ3 = KSQ)
7373        THEN GO TO U ;
7374        LOSS3 := LOSS(L) ;
7375        PC3 := ABS(BRD(SQ3)) ;
7376        VALSQ3 := VALUES(PC3) ;
7377        COMMENT if SQAxSQT and recapture requred on SQT, no more loss
7378                 unless capture on SQ3 is with check;
7379        IF CON(K,SQT) ~= 0 THEN
7380        IF SQT_CAPT AND VALSQA > VALSQ3 - DELTA AND VALSQT > ENP
7381          + DELTA AND ~CAPTURE_CH(SQ3) OR CH_CAPT AND (ABS(CON(-K,SQ3))
7382          > 1 OR CONTROL(SQ3,K) ~= KINGSQ(K))
7383        THEN GO TO U ;
7384        IF (LOST < LOSS3 AND (LOST ~= -77 OR LOSS3 > 0)) OR LOST
7385          = LOSS3 AND POS(SQA) > POS(SQ3)
7386        THEN BEGIN
7387          OFF3 := OFFSET(SQT) -OFFSET(SQ3) ;
7388
7389  COMMENT Will the capture by SQI on SQT save SQ3
7390          1. By removing SQT as an attacker,
7391          2. By adding a NEW defender at SQT.
7392        In either case see if it actually makes difference;
7393
7394          SAVE_D := BOTV(EDGE, OFF3) ;
7395      COMMENT is SQT the only attacker of SQ3
7396                is SQ3 the only attacker of SQT;
7397          IF (CON(K,SQ3) = K AND CONTROL(SQ3,K) = SQT)
7398                 AND ENP > -DELTA
7399          THEN GOTO U
7400          ELSE IF ENP >= -LOSS3
7401               THEN IF CON(-K,SQT) = -K AND CONTROL(SQT,-K) = SQ3
7402                    THEN GOTO U
7403                    ELSE IF (LOST ~= -1 OR LOSSES < LOSS3 +DELTA)
7404                         THEN FOR II := CON(KK,SQT) STEP -KK UNTIL KK
7405  DO BEGIN
7406                         COMMENT consider SQI, each attacker of SQT ;
7407    SQI := CONTROL(SQT,II) ;
7408    PCI := ABS(BRD(SQI)) ;
7409
7410     COMMENT removing P attacker, or adding P, K defence;
7411    IF SAVE_D ~= 0 AND (PCI = PAWN AND ABS(SQ3 -SQT -FILES(-K)) =
7412      1 OR PCT = PAWN AND ABS(SQ3 -SQT -FILES(K)) = 1) AND (PCI =
7413      PAWN OR PCT = PAWN OR PCI = KING) AND SQT + SAVE_D = SQ3 AND
7414      VALUES(ABS(BRD(CONTROL(SQ3,K)))) >= VALSQ3 -DELTA
7415    THEN GOTO U ;
7416
7417        COMMENT adding Knight defender or
7418                removing Knight attacker;
7419    E1 := BOTV(KNIGHT, OFF3) = 1 ;
7420    IF E1 AND ASQRS(L) ~= 0
```

```
7421   THEN BEGIN
7422     IF DEBUG
7423     THEN WRITE("KNIGHT", PCI, SQI, PC3, SQ3, PCT, SQT, ASQRS(L),
7424       K, SEC(0,SQ3)) ;
7425
7426 COMMENT are we removing a Knight defender? ;
7427     IF PCI = KNIGHT AND SQI ~= SQ3
7428     THEN IF VALUES(ABS(BRD(ABS(ASQRS(L))))) > LOSS3-DELTA
7429          THEN GOTO U ;
7430
7431 COMMENT are we removing a small Knight attacker? ;
7432     IF PCT = KNIGHT
7433     THEN IF ABS(ASQRS(L)) ~= SQT
7434          THEN GOTO UUU
7435          ELSE IF K*SEC(0,SQ3) <= 0
7436               THEN GOTO U
7437               ELSE IF SECRET(SQ3,K) ~= SQT
7438                    THEN GOTO UUU
7439                    ELSE IF ABS(SEC(0,SQ3)) > 1
7440                         THEN GOTO UUU
7441                         ELSE BEGIN
7442                           FOR I := K+K STEP K UNTIL CON(K,SQ3)
7443                           DO IF VALUES(ABS(BRD(CONTROL(SQ3,I))))
7444                                 < VALSQ3 - DELTA
7445                              THEN GOTO UUU;
7446                           GOTO U;
7447                         END;
7448   END ;
7449
7450    COMMENT removing B, R, Q attacker,
7451            or adding B, R, Q, K defence;
7452     SQ3_ATTA := PCT > KNIGHT AND BOTV(PCT, OFF3) = 1;
7453     SQ3_DEF  := SQI ~= SQ3 AND PCI > KNIGHT AND BOTV(PCI, OFF3) = 1
7454       AND (CON(K,SQT) = 0 OR VALSQ3 > VALUES(PCI)+DELTA);
7455   IF SAVE_D ~= 0 AND (SQ3_ATTA OR SQ3_DEF)
7456   THEN BEGIN
7457     SQ1 := SQT ;
7458 LOOP:
7459     SQ1 := SQ1 + SAVE_D ;
7460     WHILE BRD(SQ1) = 0
7461     DO SQ1 := SQ1 + SAVE_D ;
7462     PC1 := ABS(BRD(SQ1)) ;
7463     IF DEBUG
7464     THEN WRITE("BIGG ", PCI, SQI, PCT, SQT, SQ3, PC1, SQ1, LOSS3,
7465       LOSS(I), LOSSES, CON(K,SQ3), CONTROL(SQ3,K),
7466 SQ3_ATTA, SQ3_DEF);
7467     IF SQ1 ~= SQ3 AND PC1 > KNIGHT AND PC1 < KING AND BOTV(PC1,
7468       OFFSET(SQ1)-OFFSET(SQ3)) = 1
7469     THEN GOTO LOOP ;
7470
7471    COMMENT  indirect attack/defence ;
7472     E1 := FALSE ;
7473     IF DEBUG
7474     THEN WRITEON(CON(K,SQT), CON(-K,SQ3), CONTROL(SQ3,-K)) ;
7475     IF SQ1 = SQ3 AND (~GIVINGCH OR SQT = KINGATTACKER)
7476     THEN IF CON(K,SQT) = 0 AND PC3 > KNIGHT AND BOTV(PC3,OFF3) = 1
7477          THEN BEGIN
7478            COMMENT PC3 may capture on SQT ;
7479            IF ~SQT_CAPT OR LOST >= DELTA
7480            THEN GOTO U
7481            ELSE LOST := LOSS3 := (IF LOSS3 < LOSSES
7482                                   THEN LOSS3
7483                                   ELSE LOSSES) ;
7484     END ELSE IF CON(K,SQT) = 0
7485               THEN E1 := TRUE
7486     ELSE BEGIN
7487       IF DEBUG
7488       THEN WRITE("BIG ", PCT, BOTV(PCT, OFF3), BOTV(PCI,OFF3)) ;
7489       IF CLEAR(SQT, SQT, SQ3) AND ENP > -DELTA AND
7490         (SQ3_ATTA AND (SQ3_DEF OR ABS(CON(K,SQ3)) = 1 OR
```

```
7491                      CON(-K,SQ3) ~= 0 AND CONTROL(SQ3,-K) ~= SQI)
7492         OR SQ3_DEF AND (~CLEAR(SQI, SQI, SQ3) OR
7493                           BOTV(PCI, OFFSET(SQ3)-OFFSET(SQI)) ~= 1))
7494       THEN BEGIN
7495         COMMENT capture of SQT by SQI makes SQ3 safe if
7496                 a. SQT attacks SQ3 and defence of SQ3 not reduced.
7497                 b. Defence of SQ3 increases.
7498           Otherwise reduce LOSS3 if appropriate;
7499         GOTO U ;
7500       END ELSE IF VALSQ3 - VALSQT < LOSS3 - DELTA
7501                THEN BEGIN
7502                  LOSS3 := VALSQ3 - VALSQT ;
7503       END ;
7504       COMMENT IF CON(K,SQ3) = K
7505       THEN LOSS3 := LOSS3 -VALUES(ABS(BRD(CONTROL(SQ3,K)))) ;
7506     END ;
7507     IF E1
7508     THEN IF VALSQT < VALSQ3 + DELTA OR CON(K,SQ3) = K OR VALSQT
7509       > VALSQ3 - DELTA AND K*SEC(0,SQ3) <= 1 AND (CON(-K,SQ3) ~=
7510       -K OR CONTROL(SQ3,-K) ~= SQI OR  BOTV(PCI, OFF3) = 1)
7511          THEN LOSS3 := -VALSQT ;
7512     IF DEBUG
7513     THEN WRITEON(LOSS3, E1, K*SEC(0,SQ3)) ;
7514   END ;
7515 UUU:
7516         END ;
7517
7518         SMALL_ATTA := MAXINTEGER ;
7519         E1 := FALSE ;
7520      COMMENT is there a pin or decoy relationship in effect ;
7521         IF ~GIVINGCH OR SQT = KINGATTACKER
7522         THEN IF (ENP ~= -77) AND (~SQT_CAPT OR LOSS(I) < DELTA)
7523              THEN FOR J := CON(KK,SQT) STEP -KK UNTIL KK
7524              DO IF (SQ3 = CONTROL(SQT,J))
7525                 THEN IF (CON(-KK,SQT)  = 0 OR LOSS3 < DELTA OR
7526                   VALSQ3 < VALSQT + DELTA)
7527                      THEN GO TO U
7528                      ELSE E1 := TRUE ;
7529         IF E1
7530         THEN IF SEC(0,SQT)*KK < 0
7531              THEN E1 := FALSE
7532              ELSE FOR J := CON(-KK,SQT) STEP KK UNTIL -KK
7533         DO BEGIN
7534           PC1 := VALUES(ABS(BRD(CONTROL(SQT,J)))) ;
7535           IF SMALL_ATTA > PC1
7536           THEN SMALL_ATTA := PC1 ;
7537         END ;
7538         IF E1 AND SMALL_ATTA > VALSQ3 -DELTA
7539         THEN GO TO U ;
7540         IF ENP ~= -77 AND ENP < VALSQT -DELTA
7541         THEN IF (~SQT_CAPT AND LOSS3 > VALSQ3 -DELTA OR SQT_CAPT
7542           AND VALSQA < VALSQ3 +DELTA)
7543              THEN IF -ENP >= LOSS3 AND LOST ~= -77
7544                   THEN GOTO U
7545                   ELSE IF ~CAPTURE_CH(SQ3) AND G2 < VALSQT-DELTA
7546                   AND G2 > -DELTA
7547                        THEN BEGIN
7548                          IF DEBUG THEN
7549                            WRITEON("New_enp", VALSQT, G2, ENP,
7550                   E1, SQ3, LOSS3, LOST);
7551                          ENP := VALSQT;
7552                        END;
7553 COMMENT capture SQT, since no recapture on SQ3 ;
7554         LOST := LOSS3 ;
7555         IF LOSS3 ~= LOSS(L) AND LOSS3 > -DELTA
7556         THEN ENP := 0 ;
7557         SQA := SQ3 ;
7558         PCA := ABS(BRD(SQA)) ;
7559         VALSQA := VALUES(PCA) ;
7560         SQA_CAPT := FALSE ;
```

```
7561          SQT_CAPT := IF ENP < -DELTA
7562                      THEN FALSE
7563                      ELSE SQT_CAPT OR BOTV(ABS(BRD(SQA)), OFFSET(SQA)
7564                        -OFFSET(SQT)) =1 ;
7565        END ;
7566  U:
7567      END ;
7568    END OFBIGGER_LOSS ;
7569  @TITLE,"ANALYSE"
7570    PROCEDURE ANALYSE ;
7571    WHILE (JJ < EPL)
7572    DO BEGIN
7573      JJ := JJ+1 ;
7574      SQT := SQUARE(EPLIST(JJ)) ;
7575     COMMENT SQT contains mover's piece.  If EPLIST(0) = 0 then mover
7576       not under attack.  therefore ENPR = 0 ;
7577      LOST := LOSSES ;
7578      SQA_CAPT := SQT_CAPT := FALSE ;
7579      DECOY := 0 ;
7580      SQA := SQSAVE ;
7581      PCA := PCASAVE ;
7582      VALSQA := VALUES(PCA) ;
7583      IF DEBUG
7584      THEN SQAXSQT(I,JJ) := +99 ;
7585      IF DEBUG
7586      THEN SQTXSQA(JJ,I) := -99 ;
7587      SQT_ATT := CON(KK,SQT) ;
7588      SQTDEF := CON(-KK,SQT) ;
7589      MINDEF := IF (SQTDEF = 0)
7590                THEN 0
7591                ELSE KING ;
7592      FOR J := SQTDEF STEP KK UNTIL -KK
7593      DO BEGIN
7594        PC := ABS(BRD(CONTROL(SQT,J))) ;
7595        IF (MINDEF > PC)
7596        THEN MINDEF := PC ;
7597      END ;
7598      SQTDEF := ABS(SQTDEF) ;
7599      MINDEF := VALUES(MINDEF) ;
7600      ENP := LOSS(EPLIST(JJ)) ;
7601      IF DEBUG
7602      THEN WRITE("EPS",SQA,SQT,EPSQR,ENPR,SCR,EPSCORE,LOST,ENP) ;
7603      KINGATTA := FALSE ;
7604      IF GIVINGCH AND (SQTDEF = 0)
7605      THEN FOR J := SQT_ATT STEP -KK UNTIL KK
7606      DO IF (KSQ = CONTROL(SQT,J))
7607         THEN KINGATTA := TRUE ;
7608        COMMENT can the king capture on SQT? ;
7609      IF (ENP < -SEVENS) OR GIVINGCH AND (SQA ~= KSQ) AND ((SQT ~=
7610        KINGATTACKER) OR DOUBLECH) AND ~KINGATTA
7611      THEN ENP := -77 ;
7612      PCT := ABS(BRD(SQT)) ;
7613      VALSQT := VALUES(PCT) ;
7614      NORMAL := CH_CAPT := FALSE ;
7615      IF (VALSQT < ENP)
7616      THEN VALSQT := ENP ;
7617      RELEVANT := FALSE ;
7618        COMMENT   does sqa attack sqt, if so, then SQA may be pinned, or
7619        enprise piece might be required in the current exchange, if sqt
7620        undefended, or sqt worth more than sqa, or sqt defended by a major
7621        piece ;
7622      RECAPT_CH := FALSE ;
7623      IF (ENP ~= -77) AND (SQA ~= KSQ OR SQTDEF = 0)
7624      THEN FOR J := SQT_ATT STEP -KK UNTIL KK
7625      DO IF (SQSAVE = CONTROL(SQT,J))
7626         THEN DECOYED ;
7627      IF DEBUG AND SQT_CAPT
7628      THEN BEGIN
7629        SQAXSQT(I,JJ) := IF LOST < -SEVENS
7630                         THEN -ENP
```

```
                        ELSE LOST-ENP ;
      SQTXSQA(JJ,I) := (IF CH_CAPT
                        THEN 200
                        ELSE 0) +(IF VALSQT > ENP
                                  THEN 100
                                  ELSE 0) ;
    END ;
      COMMENT if lost = -1 then sqa can successfully capture sqt ;
    IF GIVINGCH AND (LOST < -SEVENS) AND (ENP = -77) OR (PCA ~=
      PAWN AND PCT = ENPRIS)
    THEN GO TO OMIT ;
      COMMENT can the King capture on SQT? ;
      COMMENT does SQT attack SQA.  If so does the capture on SQT
      save the piece on SQA? ;
    SQA_ATT := CON(-KK,SQA) ;
    IF GIVINGCH AND ~KINGATTA AND (SQT ~= KINGATTACKER)
    THEN ENP := -77 ;
    IF LOSS(I) > DELTA OR SQA ~= SQSAVE
    THEN FOR J := SEC(-KK,SQA) STEP KK UNTIL -KK
    DO IF (SQT = CONTROL(SQA,J))
       THEN IF ~HIDDEN(KINGSQ(-KK), SQT, SQ3, KK,TRUE)
            THEN BEGIN
              COMMENT   SQ3 ~= SQA ;
              IF DEBUG
              THEN SQTXSQA(JJ,I) :=  IF ~SQT_CAPT
                  THEN LOST
                  ELSE IF CON(KK,SQA) ~= 0
                       THEN VALSQA-VALSQT
                       ELSE VALSQA ;
              IF EPLIST(0) > 0
              THEN BEGIN
                SQA_CAPT := TRUE ;
                IF ENP ~= -77 AND (NORMAL OR CH_CAPT OR ~RELEVANT)
                THEN IF LOST ~= -1
                     THEN IF DECOY~=0
                          THEN SQA_CAPT := FALSE
                          ELSE IF (ABS(SQA_ATT) = 1)
                               THEN BEGIN
                                 COMMENT sqt only attacker ;
   COMMENT SQA not safe if only a losing capture on SQT ;
                                 IF LOST > -DELTA
                                 THEN IF (ENP < 0)
                                      THEN BEGIN
                    IF -ENP < LOST
                    THEN BEGIN
                      LOST := -ENP ;
                      ENP := 0 ;
                                        END ELSE ENP := 0 ;
                                 END ELSE IF SQT = EPSQR
                     THEN LOST := -2
                     ELSE LOST := -3 ;
COMMENT when LOST is -2 it usually means that KK (opponent) has
two pieces of the same value enpris,but it is the other one (not
on EPSQR) which it considers sacrificed. No epgain should be credited ;
                END ELSE IF ABS(J) <= ABS(SQA_ATT)
                         THEN BEGIN
                           COMMENT is SQT the smallest of several
                             attackers and SQA defended? ;
                           SQ3 := ASQRS(I) ;
                           VALSQA := VALUES(ABS(BRD(SQA))) ;
                           IF SQ3 >= WQRSQ AND LOST + ENP > -DELTA
                           THEN BEGIN
                             FOR I := SEC(-1,SQA) UNTIL SEC(1,SQA)
                             DO HOLD(I) := CONTROL(SQA,I);
                             FOR I := -1 UNTIL 1
                             DO HOLD(16*I) := SEC(I,SQA);
                             CONTROL(SQA,J) := CONTROL(SQA, SQA_ATT) ;
                             CON(-KK,SQA) := SQA_ATT + KK ;
      COMMENT protect against SQA -- SQD decoy ;
                             LOST := IF SACRIFICE(SQA, DEBUG)
```

```
7701                                        THEN IF LOSSES >= 0
7702                                             THEN LOSSES
7703                                             ELSE -3
7704                                        ELSE -3 ;
7705                                   FOR I := -1 UNTIL 1
7706                                   DO SEC(I,SQA) := HOLD(16*I);
7707                                   FOR I := SEC(-1,SQA) UNTIL SEC(1,SQA)
7708                                   DO CONTROL(SQA,I) := HOLD(I);
7709                                   CON(-KK,SQA) :=  SQA_ATT ;
7710                                 END ;
7711                               COMMENT IF DEBUG
7712                               THEN WRITE("SQTXSQA", SQ3, SQT,
7713                                 SQA, CON(KK,SQA),SQA_ATT, LOST) ;
7714                     END ;
7715                   END ;
7716                   RELEVANT := TRUE ;
7717         END ;
7718                           COMMENT if (lost < 0) then capture of sqt
7719                           makes sqa safe ;
7720         IF (LOST < -3) AND ~CH_CAPT AND ~SQA_CAPT
7721         THEN LOST := -77 ;
7722         IF (LOST > ENP AND ~SQA_CAPT OR ENP = -77 OR LOST = -77 OR
7723           CH_CAPT OR SQA_CAPT AND ENP < -DELTA) AND ~GIVINGCH
7724           AND ~RECAPT_CH
7725         THEN FOR II := 1 UNTIL EPL
7726              DO BEGIN
7727                COMMENT assume we will actually make largest capture ;
7728                L := EPLIST(II) ;
7729                SQ3 := SQUARE(L) ;
7730                IF (SQ3 = SQT)
7731                THEN GO TO UU ;
7732                PC3 := ABS(BRD(SQ3)) ;
7733                VALSQ3 := VALUES(PC3) ;
7734                IF SQT_CAPT AND CON(-KK,SQ3) ~= 0
7735                THEN FOR J := CON(KK,SQ3) STEP -KK UNTIL KK
7736                DO IF (SQA = CONTROL(SQ3,J))  AND VALSQA > VALSQ3 +DELTA
7737                   THEN GO TO UU ;
7738 COMMENT when SQAxSQT will SQA then defend SQ3?;
7739           IF SQT_CAPT AND CON(K,SQ3) = 0
7740               AND BOTV(PCA, OFFSET(SQT)-OFFSET(SQ3)) = 1
7741            THEN IF (ABS(PCA) <= KNIGHT OR ABS(PCA) = KING)
7742                THEN GOTO UU;
7743 COMMENT if SQT is attacked, ensure SQ3 is not
7744             a defender of SQT;
7745                IF CON(KK,SQT) ~= 0
7746                THEN FOR J := CON(-KK,SQT) STEP KK UNTIL -KK
7747                DO IF (SQ3 = CONTROL(SQT,J))
7748                   THEN GO TO UU ;
7749       COMMENT falacy: does not account for decoy losses ;
7750      IF DECOYS(SQ3) > LOSS(L) - DELTA
7751      THEN GOTO UU;
7752                IF (ENP < LOSS(L) -DELTA)
7753                THEN BEGIN
7754                  ENP := LOSS(L) ;
7755                  SQT_CAPT := CH_CAPT ;
7756                  IF LOST = -1 AND ~CH_CAPT
7757                  THEN LOST := LOSSES ;
7758 COMMENT           WRITE("UU ", SQ3, SQT, SQA, KK, CON(KK,SQ3));
7759                  SQT := SQ3 ;
7760                  PCT := PC3 ;
7761                  FOR J := CON(KK,SQT) STEP -KK UNTIL KK
7762                  DO IF SQA = CONTROL(SQT,J)
7763                     THEN BEGIN
7764                       COMMENT RELEVANT := TRUE ;
7765                       SQT_CAPT := TRUE ;
7766                       IF VALSQA < VALSQ3 + DELTA
7767                       THEN LOST := -1 ;
7768                  END ;
7769                  VALSQT := VALSQ3 ;
7770                END ;
```

```
7771 UU:
7772     END ;
7773     IF ENP > 0
7774     THEN CH_CAPT := CAPTURE_CH(SQT) ;
7775     IF DEBUG
7776     THEN WRITE(RELEVANT, LOST,ENP, SQT_CAPT, SQA_CAPT, CON(-KK,SQT),
7777       VALSQA) ;
7778     IF ~CH_CAPT AND (~RELEVANT OR LOST < -3) OR (SQT_CAPT
7779       AND ~SACR_CAPT AND (~CH_CAPT OR CON(-KK,SQT) ~= 0 AND VALSQT
7780       <= VALSQA-DELTA) OR LOST < -1 AND SQA_CAPT AND ~SQT_CAPT)
7781     THEN BIGGER_LOSS ;
7782     IF DEBUG
7783     THEN WRITE(LOST,ENP) ;
7784     IF ~RELEVANT AND LOST > DELTA AND ENP > 0 AND VALSQT - ENP > DELTA
7785     THEN IF VALSQT -ENP >= VALSQA +DELTA
7786          THEN LOST := 0
7787          ELSE IF ~CAPTURE_CH(SQA)
7788               THEN ENP := VALSQT ;
7789       COMMENT is SQA a "primary" attacker of SQT? ;
7790     IF (LOST = -1)
7791     THEN LOST := IF ENP < VALSQT
7792                  THEN -1
7793                  ELSE -EIGHTS ;
7794     IF ~RELEVANT AND LOST = -1
7795     THEN LOST := 0 ;
7796     VALID := RELEVANT OR LOST ~= -77 OR ENP > -DELTA ;
7797     IF ENP < -DELTA AND ~GIVINGCH
7798     THEN SQT := HOLE ;
7799     IF DEBUG
7800     THEN WRITEON(LOST,ENP,SQA_CAPT, SQA, SQT, VALID) ;
7801     IF ENP = -77 OR SQT = HOLE
7802     THEN ENP := 0 ;
7803 COMMENT is there a hidden defence of SQA via a capture of SQT ;
7804     IF ENP < 0
7805     THEN IF LOST = -77 OR LOST >= 0 OR ~SQT_CAPT AND ENP <= -DELTA
7806          THEN ENP := 0
7807          ELSE IF (ENP > -DELTA) AND RELEVANT AND ~SQT_CAPT AND
7808            CON(-KK,SQT) = 0 AND (LOST > 0) AND HIDDEN(SQA, SQT,
7809            SQ3, KK, TRUE)
7810               THEN LOST := -3 ;
7811     IF SQT = HOLE
7812     THEN SQT_CAPT := FALSE ;
7813     SC := (IF LOST = -77 OR LOST < -SEVENS OR LOST < -DELTA
7814            THEN 0
7815            ELSE LOST) -ENP ;
7816     IF VALID
7817     THEN IF (SC < COST) OR ABS(SC-COST) < DELTA AND RELEVANT
7818          THEN BEGIN
7819            COMMENT update SPAR1 if capture of SQT forces a recapture ;
7820            EP := ENP ;
7821            SQAA := IF LOST > -PVAL AND SC + ENP ~= 0
7822                    THEN SQA
7823                    ELSE HOLE ;
7824            PCAA := PCA ;
7825            SQ := SQT ;
7826            T := KINGATTA ;
7827            CHK := ~NORMAL OR RELEVANT ;
7828            ADD := CH_CAPT OR LOST < -SEVENS AND ENP > 0 ;
7829            CAPTURE := SQT_CAPT ;
7830            COST := SC ;
7831            IF DEBUG
7832            THEN WRITEON(SQ,CAPTURE_CH(SQA), CAPTURE_CH(SQT)) ;
7833     END ;
7834     IF DEBUG
7835     THEN WRITEON(SQT_CAPT,SQD,NORMAL, KINGATTACKER,DECOY,EP,SC) ;
7836 OMIT:
7837   END OFANALYSE ;
7838
7839 @TITLE,"EPSCORE"
7840    COMMENT purge duplicated squares from enpris list ;
```

```
7841    TOP := LOSS(ELIMIT) ;
7842    FOR I := TOP UNTIL SQUARE(ELIMIT)-1
7843    DO BEGIN
7844      J := I + 1 ;
7845      WHILE (J < ELIMIT)
7846      DO IF (SQUARE(J) = SQUARE(I))
7847         THEN BEGIN
7848           IF (I >= ESAVE)
7849           THEN BEGIN
7850             SQUARE(I) := SQUARE(ESAVE) ;
7851             SQUARE(ESAVE) := SQUARE(TOP) ;
7852             ESAVE := ESAVE + 1 ;
7853           END ELSE SQUARE(I) := SQUARE(TOP) ;
7854           J := ELIMIT ;
7855           TOP := TOP + 1 ;
7856      END ELSE J := J + 1 ;
7857    END ;
7858    LOSS(ELIMIT) := TOP ;
7859    KK := -K ;
7860    KSQ := KINGSQ(KK) ;
7861    SQS := SSQA := SQD := HOLE ;
7862    EPSCORE := -NINES ;
7863    PASS1 := TRUE ;
7864    SCR := -NINES ;
7865    SEC_EP := ENPSEC := TEMP := LOSERS := 0 ;
7866    G1 := G2 := G3 := -NINES ;
7867    PPCA := EDGE ;
7868    JJ := LOSS(0) ;
7869    SPECIAL := TRUE ;
7870    TT := ELIMIT ;
7871    FOR I := TOP UNTIL ELIMIT-1
7872    DO BEGIN
7873      SQA := SQUARE(I) ;
7874      ASQRS(I) := 0 ;
7875      IF (KK*BRD(SQA) > 0)
7876      THEN BEGIN
7877        TT := TT -1 ;
7878        EPLIST(TT) := I ;
7879        IF (CON(-KK,SQA) ~= 0)
7880        THEN BEGIN
7881          COMMENT not just captured, not duplicated, not king square ;
7882          IF  SACRIFICE(SQA, DEBUG)
7883          THEN LOSERS := LOSERS + 1 ;
7884          IF LOSSES > G2
7885          THEN BEGIN
7886            G3 := G2 ;
7887            IF LOSSES > G1
7888            THEN BEGIN
7889              G2 := G1 ;
7890              G1 := LOSSES ;
7891            END ELSE G2 := LOSSES ;
7892          END ELSE IF LOSSES > G3
7893                    THEN G3 := LOSSES ;
7894          TEMP := ATTASQRS(0) ;
7895          SPECIAL := FALSE ;
7896          FOR L := 1 UNTIL TEMP
7897          DO BEGIN
7898            ADD := TRUE ;
7899            SQT := ATTASQRS(L) ;
7900            FOR J := 1 UNTIL EPL
7901            DO IF ADD AND SQT = SQUARE(EPLIST(J))
7902               THEN ADD := FALSE ;
7903            IF ADD
7904            THEN BEGIN
7905              EPL := EPL +1 ;
7906              JJ := JJ +1 ;
7907              SQUARE(JJ) := SQT ;
7908              EPLIST(EPL) := JJ ;
7909            END ;
7910          END ;
```

```
7911          VALSQA := VALUES(ABS(BRD(SQA))) ;
7912          SQT := IF TEMP = 0
7913                 THEN 0
7914                 ELSE -ATTASQRS(1) ;
7915          IF SQT ~= 0
7916          THEN IF (TEMP = 1) OR (VALUES(ABS(BRD(ATTASQRS(2)))) >
7917            VALSQA -DELTA)
7918               THEN SQT := -SQT ;
7919      COMMENT SQT is -ve if it is one of many small attackers of SQA;
7920          ASQRS(I) := SQT ;
7921        END ELSE LOSSES := -NINES ;
7922        J := TT + 1 ;
7923        WHILE J < ELIMIT AND LOSSES > LOSS(EPLIST(J))
7924        DO BEGIN
7925          EPLIST(J-1) := EPLIST(J) ;
7926          J := J +1 ;
7927        END ;
7928        EPLIST(J-1) := I ;
7929      END ;
7930      LOSS(I) := LOSSES ;
7931    END ;
7932    FOR L := LOSS(0)+1 UNTIL JJ
7933    DO IF SACRIFICE(SQUARE(L), DEBUG) OR TRUE
7934       THEN LOSS(L) := LOSSES ;
7935    IF SPECIAL
7936    THEN BEGIN
7937      TOP := ELIMIT -1 ;
7938      LOSS(TOP) := -NINES -1 ;
7939    END ELSE EPLIST(ELIMIT) := TT ;
7940    CAPT_CH := FALSE ;
7941    FIRST := LOSS(EPLIST(1)) ;
7942    SECOND := IF EPL > 1
7943              THEN LOSS(EPLIST(2))
7944              ELSE 0 ;
7945    THIRD := IF EPL > 2
7946             THEN LOSS(EPLIST(3))
7947             ELSE 0 ;
7948    EXTRALOSS := EPL > 1 AND SECOND > 0 AND FIRST > 0 ;
7949    IF TT < ELIMIT-1
7950    THEN SSQA := SQUARE(EPLIST(ELIMIT-2)) ;
7951      IF CON(K,SSQA) = 0
7952      THEN SSQA := HOLE;
7953    II := TT;
7954    WHILE II < ELIMIT
7955    DO BEGIN
7956      I := EPLIST(II) ;
7957      II := II + 1 ;
7958        COMMENT SQA contains opponent's piece.  If SPECIAL, no opponent's
7959        losses.  therefore EPSCORE = 0 ;
7960      SQAA := SQA := SQSAVE := SQUARE(I) ;
7961      LOSSES := LOSS(I) ;
7962      IF LOSSES = -1
7963      THEN LOSSES := 0 ;
7964        COMMENT still enprise ;
7965      PCASAVE := PCA := ABS(BRD(SQA)) ;
7966      COST := NINES ;
7967      JJ := 0 ;
7968      EP := THIRD ;
7969      IF (EP < DELTA)
7970      THEN EP := 0 ;
7971      SQ := EPSQR ;
7972      ADD := CHK := CAPTURE := T := FALSE ;
7973      ANALYSE ;
7974      SC := COST+EP ;
7975      NEW := COST ;
7976      OLD := SCR ;
7977      FRESH := SQ ~= EPSQR AND EP > 0 OR SSQA ~= SQAA AND SC > 0
7978        OR SQAA = HOLE ;
7979      IF DEBUG
7980      THEN WRITE("COST", SC, EP, ENPR, SEC_EP, SQ,COST, CAPT_CH,
```

```
7981        ADD,SQAA,SSQA,SQS,NEW,OLD,CHK,CAPTURE,PCAA) ;
7982      E1 := ~CAPTURE OR SQ ~= EPSQR OR EP <= 0 OR SCR < -SEVENS OR
7983        ABS(NEW - OLD) >= DELTA OR SQAA = HOLE ;
7984      E2 := E1 AND COST < NINES AND (~GIVINGCH OR SCR < -SEVENS OR
7985        GIVINGCH AND SQ ~= KINGATTACKER OR SQAA = KSQ OR EP < 0 OR
7986        T OR ~DOUBLECH AND SQ = KINGATTACKER) ;
7987      E3 := E2 AND ~CAPT_CH AND NEW >= OLD+DELTA AND (SCR = -NINES
7988        OR SSQA ~= SQAA OR SSQA ~= HOLE AND SQ ~=
7989        HOLE AND (SQ ~= EPSQR OR ABS(CON(KK,SQ)) <= 1 OR ABS(SEC(0,SQ))
7990        = 1 AND (ABS(BRD(CONTROL(SQ,KK))) = KING
7991        OR ABS(BRD(CONTROL(SQ,KK+KK))) = KING))) ;
7992      E4 := E2 AND FRESH AND ~CAPT_CH AND ABS(NEW-OLD) < DELTA AND
7993        (EP < ENPR OR CHK OR COST = -EIGHTS) ;
7994      E5 := E2 AND NEW <= OLD-DELTA AND (ADD OR CAPTURE AND (SSQA
7995        ~= SQAA AND SQAA ~= HOLE OR SQ ~= EPSQR AND EPSQR = HOLE
7996        AND SSQA = SQAA)) ;
7997      IF E2 AND DEBUG
7998      THEN WRITEON(E3, E4, E5) ;
7999        COMMENT ensure that better capture on SQT not already found ;
8000      IF EPSCORE = -NINES
8001      THEN SQS := SSQA ;
8002     COMMENT if (sc = -eights) then opponent has a safe recapture
8003       without loss. If scr > -nines then this is not the first exchange
8004       by the opponent. if (enpr < ep) better exchange perhaps? ;
8005      IF E1 AND E2
8006      THEN IF E3 OR E4 OR E5
8007           THEN BEGIN
8008             IF GIVINGCH OR SQAA ~= SSQA
8009             THEN BEGIN
8010               EPSCORE := SCR ;
8011               EPSQ2 := IF PASS1
8012                        THEN SQ
8013                        ELSE EPSQR ;
8014               SQS := SSQA ;
8015               IF ~PASS1
8016               THEN SEC_EP := ENPR ;
8017             END ;
8018             PASS1 := FALSE ;
8019             IF ADD
8020             THEN CAPT_CH := TRUE ;
8021             SCR := COST ;
8022             IF E5 OR SQAA ~= HOLE
8023             THEN SSQA := SQAA ;
8024             PPCA := IF CAPTURE
8025                     THEN PCAA
8026                     ELSE EDGE ;
8027             ENPR := EP ;
8028     COMMENT ENPSEC := SPAR1 ;
8029             EPSQR := SQ ;
8030     COMMENT IF DEBUG THEN WRITEON(PPCA,SCR, SPAR1) ;
8031      END ELSE IF (COST  > EPSCORE) AND FRESH
8032               THEN BEGIN
8033                 EPSCORE := COST ;
8034                 EPSQ2 := EPSQR ;
8035                 SEC_EP := EP ;
8036              IF SQAA ~= SSQA THEN
8037                 SQS := SQAA ;
8038      END ;
8039  IF DEBUG THEN WRITEON(SSQA, SQS);
8040    END ;
8041    IF SPECIAL OR SSQA = HOLE
8042    THEN EPSCORE := 0
8043    ELSE IF SSQA < WQRSQ OR SSQA > BKRSQ
8044         THEN WRITE("WHAT", SSQA, SQAA, SCR, EPSCORE, EPSQR, ENPR)
8045         ELSE PINTRAP ;
8046    TMP := IF EPSQR = HOLE
8047           THEN 0
8048           ELSE KK*CHECK_S(KK,EPSQR) ;
8049    IF ~TRAPPED AND EPSCORE > 0 AND (M4 ~= KING) AND (TMP > 0)
8050    THEN FOR I := CON(-K,EPSQR) STEP K UNTIL -K
```

```
8051          DO BEGIN
8052            PC := ABS(BRD(CONTROL(EPSQR,I))) ;
8053            IF CHEC(TMP, PC)
8054            THEN EPSCORE := EPSCORE DIV 2 ;
8055    END ;
8056  COMMENT  if material is recaptured with check, don't count epscore ;
8057    IF EPSCORE < -DELTA
8058    THEN EPSCORE := EPSCORE + ENPR ;
8059    IF (EPSCORE <= - DELTA)
8060    THEN EPSCORE := 0 ;
8061    IF EPSCORE >= PVAL OR GIVINGCH
8062    THEN EPSCORE := EPSCORE -ENPSEC ;
8063    IF DEBUG
8064    THEN WRITE(EPLIST(0)) ;
8065    IF DEBUG
8066    THEN FOR J := 1 UNTIL EPL
8067    DO WRITEON("   ", BOX(SQUARE(EPLIST(J)))) ;
8068    IF DEBUG
8069    THEN FOR II := TT UNTIL ELIMIT-1
8070          DO BEGIN
8071            IOCONTROL(2) ;
8072            I := EPLIST(II) ;
8073            WRITE("  ", BOX(SQUARE(I)), "  ") ;
8074            FOR J := 1 UNTIL EPL
8075            DO WRITEON(SQAXSQT(I,J)) ;
8076            FOR J := 1 UNTIL EPL
8077            DO WRITEON(SQTXSQA(J,I)) ;
8078    END ;
8079    IF ~TRAPPED AND ENPR > 0 AND EPSCORE >= ENPR AND G2 > DELTA
8080    THEN BEGIN
8081      TRAPPED := TRUE ;
8082      EPSCORE := IF ENPR > G2
8083                 THEN ENPR
8084                 ELSE G2 ;
8085    END ;
8086    IF EPSCORE < DELTA AND G3 > DELTA
8087    THEN EPSCORE := EPSCORE + G3 ;
8088  COMMENT really dealing with a generalized fork, in which the opponent
8089  has two or more pieces under attack. A defence is to move one piece
8090  to block the attack on another (typically to blockcheck) ;
8091  COMMENT What about enpris loss?  For example, if ENPR loss exists
8092  and EPGAIN is possible, then should make move which produces the
8093  loss and see if a subsequent loss is likely. If so mark the position
8094  as highly non-quiescent ;
8095    IF DEBUG
8096    THEN WRITE(".", EPSQR, ENPR, SQ, TRAPPED, PIN, PPCA, EXTRALOSS,
8097      ENPSEC, EPSCORE, G1, G2, G3, FIRST, SECOND, THIRD) ;
8098    EXCHANGE := FIRST > -DELTA ;
8099    FOR I := 1 UNTIL EPL
8100    DO IF ~FORCING AND LOSS(EPLIST(I)) > 0
8101       THEN FORCING := CAPTURE_CH(SQUARE(EPLIST(I))) ;
8102    IF ~FORCING AND G2 > DELTA AND FIRST > -DELTA
8103    THEN FORCING := TRUE ;
8104    EPSCORE
8105  END OFEPSCORE ;
8106
8107  @TITLE,"SCOREMOVE"
8108  PROCEDURE SCOREMOVE ;
8109  COMMENT   generates ocon, ccon, onum, cnum etc as output ;
8110  BEGIN
8111    INTEGER I,J,PC,KK,VAL, PC3, SQ3, SQA, PCA, INC, NA, ND, K1 ;
8112    INTEGER SAC_SQ, OLD_ATTA, ATTA_SQ, SQ2, PK, KSQ, K_SQ ;
8113    LOGICAL OPPOSITION;
8114
8115    INTEGER PROCEDURE MATE_POT(INTEGER VALUE KK, K, PK, SQA) ;
8116    BEGIN
8117      INTEGER S, PC, TMP, SQ3, PCA ;
8118      S := 0 ;
8119      IF PK <= 3
8120      THEN IF PIECES(K*QUEEN) > 0 OR PK < 3 AND PIECES(K*ROOK) > 0
```

```
8121            THEN BEGIN
8122              PCA := K*BRD(SQA) ;
8123              TMP := K*CHECK_S(-K,SQA) ;
8124              IF GIVINGCH AND KK = 0 AND PCA > 0
8125              THEN KK := 1 ;
8126              FOR I := K STEP K UNTIL CON(K,SQA)
8127              DO BEGIN
8128                SQ3 := CONTROL(SQA,I) ;
8129                PC := ABS(BRD(SQ3)) ;
8130  COMMENT        WRITE("MATE_P ", SQA, PCA, TMP, K, SQ3, SAC_SQ,KK);
8131                IF SQ3 ~= SAC_SQ
8132                THEN IF KK > 0 OR GIVINGCH AND KK = 0 OR PC > BISHOP
8133                        AND HIDDEN(SQA, SQ3, SQ3, K, FALSE)
8134                     THEN IF TMP > 0 AND CHEC(TMP, PC) OR
8135                             GIVINGCH AND PCA >= ROOK
8136                          THEN S := S + (IF PK = 0 AND (KK > 0 OR
8137                                           ~GIVINGCH)
8138                                         THEN 3
8139                                         ELSE IF PK <= 1
8140                                              THEN IF GIVINGCH AND KK = 0
8141                                                   THEN (IF PCA = 0
8142                                                         THEN 2
8143                                                         ELSE 1)
8144                                                   ELSE IF PC = QUEEN

8145                                                        THEN 3
8146                                                        ELSE IF PCA >= ROOK
8147                                                             THEN 4
8148                                                             ELSE 2
8149                                               ELSE IF PC = QUEEN
8150                                                    THEN 2
8151                                                    ELSE 1) ;
8152                IF DEBUG AND S > 0
8153                THEN WRITE("MATE_POT", S, PK, SQ3, SQA, TMP, PC,PCA) ;
8154              END ;
8155        END ;
8156        S
8157      END OF_MATE_POT ;
8158
8159      IF EXCHANGE AND ENPR > -DELTA AND (EPGAIN < ENPR +DELTA)
8160      THEN SAC_SQ := IF (ABS(BRD(EPSQ)) = ENPRIS)
8161                     THEN EPSQ + K_FILE
8162                     ELSE EPSQ
8163      ELSE SAC_SQ := ATTA_SQ := HOLE ;
8164      IF SAC_SQ ~= HOLE
8165      THEN BEGIN
8166        PC := KING; ATTA_SQ := HOLE;
8167        FOR I := -K STEP -K UNTIL CON(-K,SAC_SQ)
8168        DO BEGIN
8169          SQ3 := CONTROL(SAC_SQ,I);
8170          IF PC > ABS(BRD(SQ3))
8171          THEN BEGIN
8172            PC := ABS(BRD(SQ3));
8173            ATTA_SQ := SQ3;
8174          END;
8175        END;
8176      END
8177      ELSE IF TRAPPED
8178           THEN ATTA_SQ := SSQA ;
8179      ASCORE := DSCORE := PSCORE := 0 ;
8180      PONENT := IF PIECES(-K*QUEEN) = 0
8181                THEN 0
8182                ELSE 1 ;
8183      DISC_ATTA := FALSE ;
8184      IF (DEBUG)
8185      THEN WRITE("ATTA", SAC_SQ, ATTA_SQ, EPSQ, BRD(EPSQ)) ;
8186      OCON := CCON := CNUM := 1 ;
8187      ONUM := ABS(PONENT-1) ;
8188      FOR CSQ := WQRSQ UNTIL BKRSQ
8189      DO IF BRD(CSQ) ~= EDGE
```

```
8190        THEN BEGIN
8191          COMMENT in endgame, only give credit for control of contested
8192            squares ;
8193          KK := K*SEC(0,CSQ) ;
8194          IF KK ~= 0
8195          THEN KK := IF KK > 0
8196                     THEN 1
8197                     ELSE -1 ;
8198          NA := CON(K,CSQ) ;
8199          ND := CON(-K,CSQ) ;
8200          COMMENT IF ~ENDGAME
8201          THEN PSCORE := PSCORE + KK*ABS(PAWNS(CSQ)) ;
8202          IF (NA ~= 0)
8203          THEN CCON := CCON + (IF ND ~= 0
8204                               THEN IF KK > 0
8205                                    THEN 2
8206                                    ELSE 1
8207                               ELSE IF ENDGAME
8208                                    THEN 0
8209                                    ELSE 1) ;
8210            COMMENT extra credit for absolute control of contested
8211            square, no credit in endgame for uncontested square ;
8212          IF (ND ~= 0)
8213          THEN OCON := OCON + (IF NA ~= 0
8214                               THEN IF KK < 0
8215                                    THEN 2
8216                                    ELSE 1
8217                               ELSE IF ENDGAME
8218                                    THEN 0
8219                                    ELSE 1) ;
8220          PC := BRD(CSQ) ;
8221          IF (PC ~= EMPTY)
8222          THEN BEGIN
8223            OLD_ATTA := ASCORE ;
8224            KK := IF (PC > 0)
8225                  THEN 1
8226                  ELSE -1 ;
8227            IF ~ENDGAME
8228            THEN PSCORE := PSCORE + KK*ABS(PAWNS(CSQ)) ;
8229            PC := ABS(PC) ;
8230            VAL := A_VAL(PC) ;
8231            IF (PC = PAWN)
8232            THEN BEGIN
8233              IF (RANKS(ROWS(CSQ)) = KK)
8234              THEN VAL := VAL + PAWNVAL(ROWS(CSQ)) ;
8235              IF VAL > PVAL
8236              THEN VAL := VAL + PASSEDPAWN(CSQ,KK) ;
8237            END ;
8238            IF K = KK
8239            THEN BEGIN
8240              J := NA ;
8241              NA := ND ;
8242              ND := J ;
8243              IF NA ~= 0
8244              THEN IF ABS(NA)<= ABS(ND) AND CONTROL(CSQ,NA) = KINGSQ(-K)
8245                   THEN NA := NA + K
8246                   ELSE IF ABS(ND) = 1 AND ~GIVINGCH
8247                        THEN BEGIN
8248                          PC3 := BRD(CONTROL(CSQ,ND)) ;
8249                          IF (ABS(PC3) = PAWN) AND (PC3 = BRD(CSQ+K_FILE)
8250                            OR PC3 = BRD(SQ-K_FILE)) AND
8251                            VAL > A_VAL(ABS(BRD(CONTROL(CSQ,NA)))) -DELTA
8252                          THEN GAINS := GAINS - 1 ;
8253                               COMMENT penalty for leaving double
8254                               pawn potential ;
8255              END ;
8256            END ;
8257            IF ABS(NA) > ABS(ND) +1
8258            THEN NA := -ND - KK ;
8259            IF (PC ~= KING) AND (NA ~= 0)
```

```
8260          THEN DSCORE := DSCORE + (IF (ABS(ND) > ABS(NA))
8261                                   THEN -NA
8262                                   ELSE ND)*VAL ;
8263             COMMENT  no credit for defending king or un-attacked
8264             piece ;
8265          INC := IF ENDGAME
8266                 THEN LIM-POSITION(CSQ,LIM)
8267                 ELSE 0 ;
8268       COMMENT don't expose Queen to early danger;
8269       COMMENT IF EARLY AND PC = QUEEN AND KK = K AND NA ~= 0
8270       THEN PSCORE := PSCORE - K*SCALE;
8271          IF (PC ~= QUEEN)
8272          THEN IF (KK = K)
8273               THEN CNUM := CNUM + POS(CSQ) + INC
8274               ELSE BEGIN
8275                 ONUM := ONUM + INC ;
8276                 PONENT := PONENT + POS(CSQ) ;
8277               END ;
8278 COMMENT don't credit single King attack on defended piece ;
8279          IF (CSQ ~= ATTA_SQ) AND NA ~= 0
8280          THEN ASCORE := ASCORE - (IF ABS(ND) >= ABS(NA)
8281                                   THEN KK
8282                                   ELSE IF ABS(NA) > 1
8283                    THEN KK+KK
8284                    ELSE IF ND = 0 OR KINGSQ(-KK) ~= CONTROL(CSQ,NA)
8285                            OR K ~= KK AND (~INCHECK OR ~GIVINGCH)
8286                         THEN KK
8287                         ELSE 0)*VAL ;
8288          FOR J := CON(-1,CSQ) UNTIL CON(1,CSQ)
8289          DO IF J ~= 0
8290             THEN BEGIN
8291               K1 := IF J > 0
8292                     THEN 1
8293                     ELSE -1 ;
8294               IF (~GIVINGCH OR (K = K1))
8295               THEN BEGIN
8296                 COMMENT try to give credit for the attacking value
8297                   of a pin ;
8298          COMMENT no credit for hidden counter- attack when in check ;
8299                 SQA := CONTROL(CSQ,J) ;
8300                 PCA := BRD(SQA) ;
8301                 IF (ABS(PCA) > KNIGHT) AND (ABS(PCA) < KING)
8302                 THEN BEGIN
8303                   DIR := BOTV(EDGE, OFFSET(SQA)-OFFSET(CSQ)) ;
8304                   SQ3 := CSQ + DIR ;
8305                   WHILE (BRD(SQ3) = 0)
8306                   DO SQ3 := SQ3 + DIR ;
8307                   PC3 := BRD(SQ3) ;
8308                   IF (PC3 ~= EDGE)  AND SQ3 ~= ATTA_SQ AND (PC3*PCA
8309                     < 0)
8310                   THEN BEGIN
8311                     PC3 := ABS(PC3) ;
8312                     SQ2 := CSQ + FILES(KK);
8313                     IND_ATTA := PC ~= PAWN OR ABS(DIR) ~= FILE
8314                       AND BRD(SQ2) = 0 OR ABS(DIR) =
8315                       FILE AND (BRD(SQ2+1)*KK < 0 AND BRD(SQ2+1)
8316                       ~= EDGE OR BRD(SQ2-1)*KK < 0 AND BRD(SQ2-1)
8317                       ~= EDGE) ;
8318                     IF IND_ATTA AND PC3 = KING AND KK ~= K
8319                     THEN IF K1 = KK
8320                          THEN DISC_ATTA := TRUE ;
8321                     IF (PC3 = KING) AND (PC ~= QUEEN) AND (KK ~= K)
8322                     THEN PONENT := PONENT - POS(CSQ) DIV 2 ;
8323                     SQ2 := CSQ + FILES(KK) ;
8324                     ASCORE := ASCORE + K1* (IF IND_ATTA
8325                       THEN (IF PC3 = KING
8326                             THEN IF K ~= K1
8327                                  THEN IF KK ~= K1
8328                                       THEN 15
8329                                       ELSE 50
```

```
8330                                   ELSE IF KK ~= K1
8331                                        THEN 10
8332                                        ELSE 25
8333                                ELSE A_VAL(PC3) DIV 2)
8334                           ELSE IF PC3 ~= KING OR K1 = KK
8335                                THEN 0
8336                                ELSE A_VAL(PAWN)) ;
8337                     END ;
8338                   END ;
8339                   PCA := ABS(PCA);
8340                   IF (K1 ~= KK)  AND (PC ~= KING) AND (PCA ~= KING
8341                     OR ND = 0) AND (CSQ ~= SAC_SQ) AND (A_VAL(PCA) <=
8342                      VAL-DELTA OR ND=0 AND ~HIDDEN(CSQ,SQA,SQ3,KK, FALSE)
8343                     )
8344                   THEN ASCORE := ASCORE - KK*(IF ND = 0
8345                        THEN VAL DIV 2
8346                        ELSE VAL-A_VAL(PCA)) ;
8347            COMMENT give extra credit for attacking greater or undefended
8348            piece ;
8349                 END ;
8350            END ;
8351            IF (DEBUG) AND OLD_ATTA ~= ASCORE
8352            THEN WRITEON(CSQ, ASCORE, DSCORE, PSCORE) ;
8353          END ;
8354 COMMENT   THEN WRITE("OCON CCON ",CSQ, OCON, CCON, ONUM, CNUM, PONENT);
8355     END ;
8356     ONUM := ONUM + PONENT ;
8357     MATE_P(1) := MATE_P(-1) := 0 ;
8358     K_SQ := KINGSQ(K) ;
8359     PK := POS(K_SQ) ;
8360     FOR J := K_SQRS(K) STEP -K UNTIL K
8361     DO BEGIN
8362       SQA := KSQRS(J) ;
8363       KK := K*SEC(0,SQA) ;
8364       IF CON(-K,SQA) = 0
8365       THEN CCON := CCON + (IF ~ENDGAME OR BRD(SQA) = 0
8366                            THEN 1 ELSE 0)
8367       ELSE IF ENDGAME THEN BEGIN
8368          IF KK <= 0
8369          THEN CCON := CCON + (IF KK = 0 THEN 2 ELSE 1);
8370         COMMENT in endgame, make King aggressive. Encourage opposition
8371                 and use King as an attacking piece;
8372          OCON := OCON - (IF KK < 0 THEN 2 ELSE 1);
8373       END ELSE IF (KK < 0 OR INCHECK AND KK = 0)
8374                THEN CCON := CCON -1;
8375      COMMENT  above undoes OCON credits assigned earlier;
8376       IF ~GIVINGCH AND KK < 0  AND ABS(CON(K,SQA)) = 1
8377       THEN MATE_P(-K) := MATE_P(-K) - MATE_POT(-KK,-K,PK,SQA) ;
8378 COMMENT  THEN WRITE("OCON CCON ",SQA, OCON, CCON, ONUM, CNUM);
8379     END ;
8380     KSQ := KINGSQ(-K) ;
8381     PK := POS(KSQ) ;
8382     OPPOSITION := BOTV(KING, OFFSET(K_SQ)-OFFSET(KSQ)) = 2;
8383     FOR J := K_SQRS(-K) STEP K UNTIL -K
8384     DO BEGIN
8385       SQA := KSQRS(J) ;
8386       KK := K*SEC(0,SQA) ;
8387       IF CON(K,SQA) = 0
8388       THEN OCON := OCON + (IF ENDGAME AND BRD(SQA) ~= 0
8389                            THEN 0
8390                            ELSE 1)
8391       ELSE IF ENDGAME AND KK = 0
8392            THEN OCON := OCON + 1
8393            ELSE IF ~OPPOSITION OR
8394                    BOTV(KING, OFFSET(K_SQ)-OFFSET(SQA)) ~= 1
8395            THEN IF (KK > 0 OR KK = 0 AND GIVINGCH)
8396                 THEN OCON := OCON - 1;
8397         COMMENT in the above engame mode, OCON not decremented
8398                 in region of overlapping King domains (since done);
8399 COMMENT         WRITE("SCORE ", KK, PK, SQA, CON(-K,SQA),CON(K,SQA));
```

```
8400      IF ~INCHECK AND KK > 0  AND ABS(CON(-K,SQA)) = 1
8401      THEN MATE_P(K) := MATE_P(K) + MATE_POT(KK,K,PK,SQA) ;
8402  COMMENT  THEN WRITE("OCON CCON ",SQA, OCON, CCON, ONUM, CNUM);
8403    END ;
8404     COMMENT compute King opposition credit ;
8405     COMMENT  or passed pawn blockage credit ;
8406         SQ3 := IF CLEARPAWN(-K) ~= KSQ
8407                THEN IF K_SQ = CLEARPAWN(-K)
8408                     THEN K_SQ
8409                     ELSE CLEARPAWN(-K) -FILES(K)
8410               ELSE IF BASE ~= CLEARPAWN(K)
8411                    THEN CLEARPAWN(K)
8412                    ELSE KSQ;
8413    CCON := CCON + (IF ~KINGMOVING
8414                    THEN CLEARPAWN(0)
8415                    ELSE POTENTIAL(K_SQ, SQ3, K_SQ = CLEARPAWN(-K)));
8416  COMMENT  THEN WRITE("OCON CCON ",SQA, OCON, CCON, ONUM, CNUM);
8417    IF SAC_SQ ~= HOLE
8418    THEN BEGIN
8419      COMMENT no attackscore for sacrificed piece ;
8420      IF ENPR > DELTA AND (ABS(BRD(SAC_SQ)) ~= QUEEN)
8421      THEN CNUM := CNUM - POS(SAC_SQ) ;
8422      J := POS(SAC_SQ) ;
8423        COMMENT Note SAC_SQ may contain a Pawn, but not King ;
8424      FOR IX := 1 UNTIL J
8425      DO CHECKLIST(IX) := POSITION(SAC_SQ,IX) ;
8426      FOR IX := POSITION(SAC_SQ,LIM) UNTIL LIM-1
8427      DO BEGIN
8428        J := J +1 ;
8429        CHECKLIST(J) := POSITION(SAC_SQ,IX) ;
8430      END ;
8431      FOR IX := 1 UNTIL J
8432      DO BEGIN
8433        I := CHECKLIST(IX) ;
8434        IF ENPR > DELTA
8435        THEN IF (SEC(0,I) = K) AND (~ENDGAME OR (CON(-K,I) ~= 0))
8436             THEN CCON := CCON -1
8437             ELSE IF (SEC(0,I) = 0)
8438                  THEN OCON := OCON +1 ;
8439        PC := BRD(I) ;
8440        IF (PC*K < 0)
8441        THEN ASCORE := ASCORE - K*A_VAL(ABS(PC)) ;
8442           COMMENT ignore defended squares ;
8443  COMMENT  THEN WRITE("OCON CCON ",I, OCON, CCON, ONUM, CNUM);
8444      END ;
8445      ASCORE := ASCORE - (-K)*A_VAL(ABS(BRD(SAC_SQ))) ;
8446    END ;
8447  COMMENT  THEN WRITE("OCON CCON ",SAC_SQ, OCON, CCON, ONUM, CNUM);
8448    IF DISC_ATTA
8449    THEN GAINS := GAINS -2 ;
8450    IF ONUM <= 0 OR OCON <= 0 OR CNUM <= 0 OR CCON <= 0
8451    THEN BEGIN
8452      WRITE("PROBLEMS",ONUM,OCON,CNUM,CCON) ;
8453      TRACER(1, "?P") ;
8454    IF OCON < 1 THEN OCON := 1;
8455    END ;
8456  END OFSCOREMOVE ;
8457
8458  @TITLE,"TRYTHEM"
8459  PROCEDURE TRYTHEM ;
8460  BEGIN
8461    LOGICAL NOWIN ;
8462    NOWIN := LATE_END AND (MEN(K) = 1 OR MEN(K) = 2
8463       AND PIECES(KNIGHT*K) + PIECES(BISHOP*K) = 1) ;
8464    IP := NN ;
8465    WHILE IP ~= 0
8466    DO BEGIN
8467      N := USE(ABS(IP)) ;
8468      SQ := MOVETO(N) ;
8469      LEGALMV := FORCING := FALSE ;
```

```
8470      IF ~ INCHECK
8471      THEN LEGALMV := TRUE
8472      ELSE IF ~ DBLCHECK
8473           THEN FOR JJ := KINGSQU+SAVEB STEP SAVEB UNTIL CHECKSQ
8474           DO IF (SQ = JJ)
8475              THEN LEGALMV := TRUE ;
8476      MF := MOVEFROM(N) ;
8477      IF REMEMBER
8478      THEN BEGIN
8479        LSAVE := SQUARE(0) ;
8480        ESAVE := SQUARE(ELIMIT) ;
8481      END ;
8482      REWARDN := ABS( REWARD(N)) ;
8483      IF ~LEGALMV AND (REWARDN = ENPRIS)
8484      THEN IF (SQ = CHECKSQ + K_FILE)
8485           THEN LEGALMV := TRUE ;
8486      IF (TERMINAL)
8487      THEN BASE := MF
8488      ELSE MOVEFROM(N) := LAST_SQ ;
8489      NEXTSQ := MOVEFROM(USE(ABS(IP-K))) ;
8490      IF ~ INCHECK
8491      THEN TERMINAL := IF (BASE ~= NEXTSQ) OR (REWARDN ~= 0)
8492                       THEN TRUE
8493                       ELSE FALSE ;
8494      HIDDENLOSS(N, LSAVE, ESAVE) ;
8495     COMMENT add hidden pieces to enprise list ;
8496      IF (BASE ~= NEXTSQ)
8497      THEN PPIN := FALSE ;
8498      IF (LEGALMV AND ~ PINNED) OR (ABS(BRD(LAST_SQ)) = KING)
8499      THEN BEGIN
8500        COMMENT no illegal king moves are generated ;
8501        LAST_SQ := SQ ;
8502        MADELEGAL := TRUE ;
8503        MAKEMOVE(N,FALSE) ;
8504        MV := 0 ;
8505        KSQ := KINGSQ(-K) ;
8506        MATE := FALSE ;
8507        KINGATTACKER := IF ~GIVINGCH
8508                        THEN HOLE
8509                        ELSE CONTROL(KINGSQ(-K), K) ;
8510        PC := REWARDN ;
8511        IF (PC >= ENPRIS)
8512        THEN PC := PC -PROMOTE ;
8513        GAINS := IF (REWARDN > PFTWO)
8514                 THEN (IF PC = ROOK OR PC = BISHOP
8515                       THEN 0
8516                       ELSE VALUES(PC)) -PVAL - PVAL
8517                       ELSE IF (PC > NIL) AND (PC < KING)
8518                            THEN VALUES(PC)
8519                            ELSE 0 ;
8520        CAPT := GAINS ;
8521        IF (PROMCAPT > 0)
8522        THEN GAINS := GAINS + VALUES(PROMCAPT) ;
8523        PC := ABS(BRD(SQ)) ;
8524        KINGMOVING := IF (PC = KING)
8525                      THEN TRUE
8526                      ELSE FALSE ;
8527        MATE_P(1) := MATE_P(-1) := MATE_T(1) := MATE_T(-1) := 0 ;
8528        IF ~OPENING
8529        THEN MATE_THREAT ;
8530        TRAPPED := FALSE ;
8531        ENPRISLIST ;
8532        SCOREMOVE ;
8533        GAINS := GAINS + MATE_P(1) + MATE_P(-1) ;
8534        IF GIVINGCH AND (POS(KSQ) = 0)
8535        THEN CHECKER
8536        ELSE IF (PONENT = 0)
8537             THEN REWARD(N) := REWARDN + CHECKING ;
8538        IF (EPLIST(0) = 0)
8539        THEN EPSQ := HOLE ;
```

```
8540        IF (PC =PAWN) AND (PAWNS(SQ) = K)
8541        THEN CNUM := CNUM + 3*POS(SQ) ;
8542  COMMENT promotion mobility credit ;
8543        SAMV := -DATUM +  (DATUM*CNUM*CCON) DIV (ONUM*OCON) ;
8544        IF ENDGAME AND (SAMV > 450)
8545        THEN SAMV := 300 + SAMV DIV 4 ;
8546        IF (EPSQ = SQ) AND (PC = PAWN) AND (ENPR ~= PVAL)
8547        THEN ENPR := ENPR -2 ;
8548        EP := IF ENPR > 0  AND GAINS+EPGAIN < ENPR+DELTA
8549          AND LOSS(EPLIST(1)) > -DELTA
8550              THEN IF REALVAL < -DELTA
8551                   THEN IF REALVAL < -ENPR
8552                        THEN -ENPR DIV 4
8553                        ELSE REALVAL DIV 4
8554                        ELSE IF ONEVAL > DELTA
8555                             THEN IF ONEVAL > ENPR
8556                                  THEN ENPR DIV 4
8557                                  ELSE ONEVAL DIV 4
8558                             ELSE 0
8559              ELSE 0 ;
8560  COMMENT penalize a sacrifice when down,  encourage when ahead ;
8561        IF (ENPR = -1) AND ~TRAPPED AND ~GIVINGCH
8562        THEN IF (EPGAIN > 0)
8563             THEN EPGAIN := 0 ;
8564        IF (ENPR <= 0)
8565        THEN ENPR := 0; COMMENT
8566        ELSE IF ENPR = VALUES(BISHOP)  AND EPGAIN < VALUES(BISHOP)
8567             THEN ENPR := VALUES(KNIGHT) ;
8568        SVMV := IF TRAPPED OR (ENPR < PVAL) OR REWARDN > PFTWO
8569                THEN ENPR
8570                ELSE IF CAPT = 0 AND CON(K,EPSQ) = 0
8571                     THEN ENPR -1
8572                     ELSE ENPR -2 ;
8573        TMP := IF SVMV > EPGAIN + DELTA OR PIN
8574               THEN EPGAIN
8575               ELSE IF ENPR <= 0 OR EPGAIN < ENPR -DELTA AND CAPT = 0
8576                    THEN (5*EPGAIN) DIV 7
8577                    ELSE IF EPGAIN < ENPR + DELTA AND CAPT = 0
8578                         THEN (9*ENPR) DIV 10
8579                         ELSE SVMV ;
8580  COMMENT re-estimate epgain, preserve forks ;
8581        TMP := IF EPGAIN < -DELTA
8582               THEN 2*EPGAIN DIV 3
8583               ELSE IF EPGAIN < DELTA
8584                    THEN 0
8585                    ELSE IF GIVINGCH
8586                         THEN IF TRAPPED
8587                              THEN EPGAIN
8588                              ELSE (9*EPGAIN) DIV 10
8589                         ELSE IF TRAPPED
8590                              THEN IF (REWARDN > 0) AND (REWARDN
8591                                ~= PFTWO) OR (EPGAIN - ENPR
8592                                < VALUES(KNIGHT) -DELTA)
8593                     THEN EPGAIN
8594                     ELSE IF EPGAIN -ENPR > 40
8595                          THEN 30
8596                          ELSE EPGAIN - 2*PVAL DIV 3
8597                              ELSE IF (ENPR >= EPGAIN +DELTA)
8598                                   THEN TMP
8599                                   ELSE IF (ENPR ~= -1)
8600                                        THEN TMP
8601                                        ELSE IF (EPLIST(0) > 0)
8602                                             THEN EPGAIN DIV 3
8603                                             ELSE (2*EPGAIN) DIV 3 ;
8604        MATER := IF INCHECK AND (POS(KINGSQ(K)) = 0) AND (GAINS <
8605                    ENPR-DELTA)
8606                 THEN PVAL
8607                 ELSE 0 ;
8608        IF GIVINGCH AND ENPR = 0
8609        THEN IF POS(KINGSQ(-K)) <= 1
```

```
8610            THEN GAINS := GAINS +1 ;
8611 COMMENT forcing check ;
8612       IF DEBUG
8613       THEN WRITE("PARM",N,CAPT,CREDIT(LEVEL),GOODCASTLE,MV,GAINS,TMP,PC,
8614         ONEVAL, EP,PROMCAPT, SVMV,MATER,BACKROW, MATE_P(1),MATE_T(1),
8615         MATE_P(-1), MATE_T(-1), LAST_SQ, BASE, MF,NEXTSQ,SQ, DISC_ATTA)
8616         ;
8617 COMMENT attempt to equalize losses if threatening mate;
8618       IF MATE_P(K) >= 3 AND GAINS + TMP + BACKROW <= ENPR + MATER
8619           AND ~CAPT_CH
8620       THEN GAINS := SVMV + MATER - TMP -BACKROW+2 ;
8621       SVMV := MV := SCALE*(MV+ GAINS + TMP + EP - SVMV + ONEVAL
8622         -MATER +CREDIT(LEVEL) + BACKROW) ;
8623       ATTACKSCORE := K*ASCORE ;
8624       DEFENCESCORE := K*DSCORE ;
8625       PLACESCORE := K*PSCORE ;
8626       MV := MV + 4*ATTACKSCORE + 2*DEFENCESCORE  + 3*PLACESCORE ;
8627       IF (DUMPP) OR TEST
8628       THEN COLLECTDATA ;
8629       MV := SAMV + MV ;
8630       IF REVERSIBLE
8631       THEN IF INCHECK OR GIVINGCH OR ENDGAME AND (PIECES(0) < 12)
8632            THEN BEGIN
8633              HASH := -NEWHASH(BRD, K, KEY) ;
8634              TMP := 0;
8635              FOR J := 1 UNTIL ENDPLAY
8636              DO IF (HASH = REPEATS(J))
8637                 THEN TMP := TMP + 1 ;
8638              FOR J := 1 UNTIL LEVEL-1
8639              DO IF HASH = STACK_HASH(J)
8640                 THEN TMP := TMP + 1 ;
8641              IF TMP > 1 OR TMP = 1 AND MV < DRAW
8642              THEN MV := DRAW;
8643 IF TEST AND MV = DRAW THEN BEGIN
8644   WRITE("CYCLE ", TMP);
8645   FOR I := 1 UNTIL ENDPLAY
8646   DO WRITEON(REPEATS(I));
8647   FOR I := 1 UNTIL LEVEL-1
8648   DO WRITEON(STACK_HASH(I));
8649 END;
8650       END ;
8651       MV :=  IF (MV = BAD)
8652              THEN BAD+2
8653              ELSE IF MATE
8654                   THEN NINES
8655                   ELSE IF ~GIVINGCH AND (PONENT = 0)
8656                        THEN DRAW
8657                        ELSE IF (MV > SEVENS)
8658                             THEN SEVENS-7
8659                             ELSE MV ;
8660 COMMENT   if the moving piece does not capture, the concept of
8661 an incremental move can be used to reduce the needless referencing
8662 of takebackmove. ;
8663       IF (MV > BEST)
8664       THEN BEST := MV ;
8665       IF NOWIN AND MV ~= DRAW THEN BEGIN
8666         MV := MV*DRAW DIV (IF BEST < DRAW THEN DRAW ELSE BEST) ;
8667         IF LEVEL > 1
8668         THEN MV := MV +CREDIT(LEVEL)*SCALE;
8669       END;
8670       REWARDN := ABS(REWARD(N)) ;
8671       IF FALSE AND ~MATE AND ~CAPTURE_TREE
8672       THEN IF (REWARDN >= REALCHECK) AND (REWARDN < TWOCHECK)
8673            THEN IF (POS(KSQ) = 0) OR (ABS(CON(-K,EPSQ)) = 1) AND
8674              (CONTROL(EPSQ,-K) = KSQ) AND ((CAPT > 0) OR (POS(KSQ)
8675              = 1)) OR GIVINGCH AND (POS(KSQ) <= 1) AND
8676              (CON(-K,KINGATTACKER) = 0 OR (CONTROL(KINGATTACKER,-K)
8677              = KSQ AND K*SEC(0,KINGATTACKER) >= 0) OR
8678              ~SACRIFICE(KINGATTACKER, FALSE))
8679                 THEN MV := -NINES-NINES +(MV DIV 10) ;
```

```
8680
8681 COMMENT two types of king check sacrifices are now detected.
8682 (1).  King has no moves - common in back-rank mates.
8683 (2).  King is forced to recapture (only move).
8684 (3).  Cannot handle case in which loss occurs even if the King
8685       recaptures. ;
8686 COMMENT forcing King to move, ignore blocks ;
8687
8688       TMP := IF ABS(EPGAIN) > DELTA AND (LEVEL < LENGTH OR ENPR
8689         > EPGAIN -DELTA OR TRAPPED OR PIN) OR DISC_ATTA OR ABS(EPGAIN)
8690         > VALUES(BISHOP) OR GIVINGCH AND (DOUBLECH OR
8691         CON(K,KINGATTACKER) = 0 AND BOTV(KING, OFFSET(KINGSQ(-K))
8692         -OFFSET(KINGATTACKER)) = 2) OR FORCING
8693              THEN 6
8694              ELSE IF EPL > 1 AND LOSS(EPLIST(2)) > DELTA
8695                   THEN IF ENPR > -DELTA AND CAPT > DELTA
8696                        THEN 5
8697                        ELSE IF EPGAIN < -1
8698                             THEN 4
8699                             ELSE 3
8700                             ELSE IF POS(KINGSQ(K)) = 0
8701                                  THEN 2
8702                                  ELSE 0 ;
8703       IF DEBUG
8704       THEN WRITEON("*", TMP, CLEARPAWN(K), CLEARPAWN(-K), FORCING) ;
8705 COMMENT later give special attention to disc. and dbl. ch, and
8706 also to all forcing checks ;
8707       TMP := IF TMP > 4 OR ~OPENING AND BACKROW > 2 OR ABS(MATE_T(-K)
8708         +MATE_P(-K)) >= 3 OR GIVINGCH AND MATE_P(K) >= 3
8709              THEN 1
8710              ELSE 0 ;
8711       IF TMP > 0
8712       THEN REWARD(N) := IF (REWARDN = 0)
8713                         THEN -2
8714                         ELSE -REWARDN ;
8715 COMMENT set reward(n) -ve to flag non-quiescent position ;
8716     END ELSE
8717     BEGIN
8718       MV := BAD ;
8719       NUM := NUM -1 ;
8720       IF ~ MADELEGAL
8721       THEN LAST_SQ := MF ;
8722       TERMINAL := TRUE ;
8723       BASE := MF ;
8724     END ;
8725     M2 := MOVEFROM(N) := MF ;
8726     SCORE(N) := MV ;
8727 COMMENT IF DUMPP THEN PRINTLINE(N, LEVEL);
8728 COMMENT IF DUMPP THEN WRITEON(GAINS, CREDIT(LEVEL), BACKROW);
8729     STOP := ~INCHECK AND TMP = 0 AND MV -LEVEL > UPPER AND MV ~=
8730       BAD AND PP_STATUS < 10 AND M14 = 0 AND (~LATE_END OR REWARDN
8731       < CHECKING) AND 100*ABS(IP) > 5*ABS(NN) ;
8732     IF TERMINAL OR STOP
8733     THEN BEGIN
8734       COMMENT retract last legal move ;
8735       IF MADELEGAL
8736       THEN BEGIN
8737         MADELEGAL := FALSE ;
8738         TAKEBACKMOVE(FALSE) ;
8739       END ;
8740       REMEMBER := TRUE ;
8741       LAST_SQ := NEXTSQ ;
8742     END ;
8743     IP := IP - K ;
8744     IF STOP
8745     THEN BEGIN
8746       MV  := NUM -ABS(IP) ;
8747       IP := 0 ;
8748     END ;
8749   END FORN ;
```

```
8750 END OFTRYTHEM ;
8751
8752 @TITLE,"MOBILITY"
8753 I_W := 4 ;
8754    COMMENT PARTIAL_ORDER;
8755 IF INCHECK
8756 THEN BEGIN
8757   KINGSQU := KINGSQ(K) ;
8758   DIR := BOTV(EDGE, OFFSET(KINGSQU)-OFFSET(CHECKSQ)) ;
8759   SAVEB := IF (DIR = 0)
8760            THEN CHECKSQ - KINGSQU
8761            ELSE DIR ;
8762    COMMENT knight attack ;
8763   IF DEBUG
8764   THEN WRITE("MOBI",KINGSQU,CHECKSQ,BRD(CHECKSQ),DIR,DBLCHECK) ;
8765 END ;
8766 COMMENT IF ~INCHECK THEN CLEARPAWN(K) :=  KINGSQ(K) ;
8767 K_PAWN := K*PAWN ;
8768 K_FILE := FILES(K) ;
8769 KSQRS(0) := POTENTIAL(KINGSQ(K), KINGSQ(-K), FALSE) ;
8770 CLEARPAWN(0) := IF CLEARPAWN(-K) = KINGSQ(-K)
8771                 THEN KSQRS(0)
8772                 ELSE POTENTIAL(KINGSQ(K), CLEARPAWN(-K)-K_FILE,TRUE) ;
8773 MADELEGAL := PPIN := FALSE ;
8774 OLD_DIR := LASTDIR(LEVEL-2) ;
8775 OLD_DEST := LASTSQ(LEVEL-2) ;
8776 IF OLD_DIR = 0
8777 THEN OLD_DIR := -2 ;
8778 NN := NUMBER(K) ;
8779 TERMINAL := REMEMBER := TRUE ;
8780 MOVEFROM(0) := BASE := HOLE ;
8781 BEST := -NINES ;
8782 LAST_SQ := MOVEFROM(USE(ABS(NN))) ;
8783 IF TEST AND EXAMINE  AND DUMPP
8784 THEN WRITE("    N   OCON   CCON   ONUM   CNUM ONEVAL GOODCA",
8785   " ATTACK DEFENCE    EP  GAINS  SAMV    ENPR EPGAIN  KINGCH  ",
8786   "SVMV  SCORE") ;
8787 TRYTHEM ;
8788 NEW := NEW + 1 ;
8789 BRAN := BRAN + NUM ;
8790 BRAN_S := BRAN_S + (IF STOP
8791                     THEN MV
8792                     ELSE NUM) ;
8793 NOD := NOD + 1 ;
8794 IF INCHECK
8795 THEN REMAKE(KINGSQ(K),KINGSQ(K)) ;
8796 IF UPPER ~= EIGHTS AND SCORE(USE(1)) ~= NOSCR
8797 THEN UPPER := EIGHTS ;
8798 END OFMOBILITYSCORE ;
8799
8800 @TITLE "RESTOREBACK"
8801 PROCEDURE RESTOREBACK(INTEGER VALUE LEVEL ;
8802 LOGICAL VALUE FLAG) ;
8803 BEGIN
8804   INTEGER J, J1 ;
8805   IF FLAG
8806   THEN BEGIN
8807     M2 := BACK(LEVEL,0) ;
8808     M3 := BACK(LEVEL,1) ;
8809     M7 := BACK(LEVEL,2) ;
8810     M8 := BACK(LEVEL,3) ;
8811     M9 := BACK(LEVEL,4) ;
8812     J := BACK(LEVEL,5) ;
8813     M14 := ABS(J) REM 100 ;
8814     M11 := ABS(J) DIV 100 ;
8815     IF J < 0
8816     THEN M14 := -M14 ;
8817     NUMB := BACK(LEVEL,6) ;
8818     M5 := BACK(LEVEL,7) ;
8819     J := 19 + NUMB ;
```

```
8820      FOR I := 0 UNTIL NUMB
8821      DO BEGIN
8822        RLIST(I) := BACK(LEVEL,18+I) ;
8823        TLIST(I) := BACK(LEVEL,J+I) ;
8824      END ;
8825    END ;
8826    J := S_SIZE ;
8827    IF FLAG
8828    THEN FOR J1 := WQRSQ STEP FILE UNTIL BKRSQ
8829    DO FOR J2 := J1 UNTIL J1+7
8830       DO BEGIN
8831         J := J + 1 ;
8832         BRD(J2) := BACK(LEVEL,J) ;
8833    END ;
8834    J := 64 + S_SIZE ;
8835    KINGSQ(-K) := BACK(LEVEL,8) ;
8836    KINGSQ(K) := BACK(LEVEL,9) ;
8837    CLEARPAWN(-K) := BACK(LEVEL,10) ;
8838    CLEARPAWN(K) := BACK(LEVEL,11) ;
8839    DIFFVAL := BACK(LEVEL,12) ;
8840    ONEVAL := BACK(LEVEL,13) ;
8841    OLDSCR := BACK(LEVEL,14) ;
8842    PREVSCR := BACK(LEVEL,15) ;
8843    KTEST := BACK(LEVEL,16) ;
8844    NUM := BACK(LEVEL,17) ;
8845    INCHECK := IF (BACK(LEVEL,J+1) > 0)
8846               THEN TRUE
8847               ELSE FALSE ;
8848    CASTLE(K) := CASTLE(TWO(K)) := CASTLE(THREE(K)) := FALSE ;
8849    J1 := BACK(LEVEL,J+2) ;
8850    IF (J1 > 0)
8851    THEN CASE J1 OF
8852    BEGIN
8853      CASTLE(K) := TRUE ;
8854      CASTLE(TWO(K)) := TRUE ;
8855      CASTLE(TWO(K)) := CASTLE(K) := TRUE ;
8856      CASTLE(THREE(K)) := TRUE ;
8857    END ;
8858    J := J + 11 ;
8859     COMMENT j := j+2+9 ;
8860    FOR I := -8 UNTIL 8
8861    DO PIECES(I) := BACK(LEVEL,J+I) ;
8862    J := J + 9 ;
8863     COMMENT j := j+8+1 ;
8864    J1 := J + ELIMIT + 1 ;
8865    IF FLAG
8866    THEN FOR I := 0 UNTIL ELIMIT
8867         DO BEGIN
8868           SQUARE(I) := BACK(LEVEL,J+I) ;
8869           LOSS(I) := BACK(LEVEL,J1+I) ;
8870    END ;
8871  END OFRESTOREBACK ;
8872
8873  @TITLE,"RESTORESTATE(INTEGER VALUE NODE) "
8874  PROCEDURE RESTORESTATE(INTEGER VALUE NODE) ;
8875  BEGIN
8876    INTEGER J,J1 ;
8877    K := STATES(NODE,1) ;
8878    PLEX := STATES(NODE,2) ;
8879    CAPT_T := STATES(NODE,3) ;
8880    TOT_S := NODETO(NODE,4) ;
8881    PART := STATES(NODE,5) ;
8882    A_SCR := STATES(NODE,6) ;
8883    UPPER := STATES(NODE,7) ;
8884    HASH := STATES(NODE,T_SIZE) ;
8885    RESTORE(NODE) ;
8886  END OFRESTORESTATES ;
8887
8888  PROCEDURE RESTORE(INTEGER VALUE NODE) ;
8889  BEGIN
```

```
8890    INTEGER J, J1, J2, J3, J4,  N, STATE ;
8891    TRY(0) := STATES(NODE,T_SIZE+1) ;
8892    TRY(WSIZE) := STATES(NODE,T_SIZE+2) REM 1000 ;
8893    NUM := STATES(NODE,T_SIZE+2) DIV 1000 ;
8894    FOR I := 1 UNTIL TRY(WSIZE)
8895    DO TRY(I) := (TOTAL-I)*K ;
8896    J := TRY(WSIZE) ;
8897    J1 := T_SIZE + 2 ;
8898    J2 := J1 + MAX_WID ;
8899    FOR I := 1 UNTIL J
8900    DO BEGIN
8901      N := TRY(I) ;
8902      SCORE(N) := STATES(NODE,J1+I) ;
8903      STATE := STATES(NODE, J2+I) ;
8904      MOVEFROM(N) := ABS(STATE) REM 256 ;
8905      STATE := STATE DIV 256 ;
8906      MOVETO(N) := ABS(STATE) REM 256 ;
8907      STATE := STATE DIV 256 ;
8908      QUIES(N) := (ABS(STATE) REM 2) = 1;
8909      REWARD(N) := ABS(STATE) DIV 2;
8910      IF STATE < 0
8911      THEN REWARD(N) := -REWARD(N) ;
8912        COMMENT MOVEFROM(N) := STATES(NODE,J2+I) ;
8913        COMMENT MOVETO(N) := STATES(NODE,J3+I) ;
8914        COMMENT REWARD(N) := STATES(NODE,J4+I) ;
8915    END ;
8916  END OFRESTORE ;
8917
8918  @TITLE,"SAVEBACK"
8919  PROCEDURE SAVEBACK(INTEGER VALUE LEVEL ;
8920  LOGICAL VALUE FLAG) ;
8921  BEGIN
8922    INTEGER J, J1 ;
8923    IF FLAG
8924    THEN BEGIN
8925      J := 100*M11 + ABS(M14) ;
8926      IF M14 < 0
8927      THEN J := -J ;
8928      FOR I := 1 UNTIL 8
8929      DO BACK(LEVEL,I-1) := CASE I OF (M2,M3,M7,M8,M9,J,NUMB,M5) ;
8930      J := 19+NUMB ;
8931      FOR I := 0 UNTIL NUMB
8932      DO BEGIN
8933        BACK(LEVEL,18+I) := RLIST(I) ;
8934        BACK(LEVEL,J+I) := TLIST(I) ;
8935      END ;
8936    END ;
8937    J := S_SIZE ;
8938    IF FLAG
8939    THEN FOR J1 := WQRSQ STEP FILE UNTIL BKRSQ
8940    DO FOR J2 := J1 UNTIL J1+7
8941       DO BEGIN
8942         J := J + 1 ;
8943         BACK(LEVEL,J) := BRD(J2) ;
8944    END ;
8945    IF ~FLAG
8946    THEN FOR I := 8 UNTIL 17
8947    DO BACK(LEVEL,I) := CASE I-7 OF ( KINGSQ(-K),KINGSQ(K),CLEARPAWN(-K),
8948      CLEARPAWN(K), DIFFVAL, ONEVAL, OLDSCR, PREVSCR, KTEST, NUM) ;
8949    J := S_SIZE + 64 ;
8950    IF ~FLAG
8951    THEN BACK(LEVEL,J+1) := IF INCHECK
8952                            THEN 1
8953                            ELSE 0 ;
8954    IF CASTLE(THREE(K))
8955    THEN J1 := 4
8956    ELSE BEGIN
8957      J1 := 0 ;
8958      IF CASTLE(K)
8959      THEN J1 := J1 +1 ;
```

```
8960      IF CASTLE(TWO(K))
8961      THEN J1 := J1 +2 ;
8962    END ;
8963    IF ~FLAG
8964    THEN BACK(LEVEL,J+2) := J1 ;
8965    J := J + 11 ;
8966     COMMENT j := j+2+9 ;
8967    IF ~FLAG
8968    THEN FOR I := -8 UNTIL 8
8969    DO BACK(LEVEL,J+I) := PIECES(I) ;
8970    J := J + 9 ;
8971     COMMENT j := j+8+1 ;
8972    J1 := J + ELIMIT + 1 ;
8973    IF FLAG
8974    THEN FOR I := 0 UNTIL ELIMIT
8975         DO BEGIN
8976           BACK(LEVEL,J+I) := SQUARE(I) ;
8977           BACK(LEVEL,J1+I) := LOSS(I) ;
8978    END ;
8979  END OFSAVEBACK ;
8980
8981  @TITLE,"SAVESTATE(INTEGER VALUE NODE)"
8982  PROCEDURE SAVESTATE(INTEGER VALUE NODE) ;
8983  BEGIN
8984    INTEGER J,J1 ;
8985  COMMENT IF MONITOR THEN WRITE("SAVEST ", NODE, K, HASH);
8986    FOR I := 1 UNTIL T_SIZE
8987    DO STATES(NODE,I) := CASE I OF (K, PLEX, CAPT_T, TOT_S, PART,
8988      A_SCR, UPPER, HASH) ;
8989    SAVE(NODE) ;
8990  END OFSAVESTATE ;
8991
8992  PROCEDURE SAVE(INTEGER VALUE NODE) ;
8993  BEGIN
8994    INTEGER J,J1,J2,J3,J4,N, STATE ;
8995    STATES(NODE,T_SIZE+1) := TRY(0) ;
8996    STATES(NODE,T_SIZE+2) := TRY(WSIZE) + NUM*1000 ;
8997    J := TRY(WSIZE) ;
8998    J1 := T_SIZE + 2 ;
8999    J2 := J1 + MAX_WID ;
9000    FOR I := 1 UNTIL J
9001    DO BEGIN
9002      N := TRY(I) ;
9003      STATES(NODE,J1+I) := SCORE(N) ;
9004      STATE := MOVEFROM(N) + MOVETO(N)*256 + 131072*ABS(REWARD(N)) ;
9005      IF QUIES(N)
9006      THEN STATE := STATE + 65536;
9007      IF REWARD(N) < 0
9008      THEN STATE := -STATE ;
9009      STATES(NODE, J2+I) := STATE ;
9010        COMMENT STATES(NODE,J2+I) := MOVEFROM(N) ;
9011        COMMENT STATES(NODE,J3+I) := MOVETO(N) ;
9012        COMMENT STATES(NODE,J4+I) := REWARD(N) ;
9013    END ;
9014  END OFSAVE ;
9015
9016  COMMENT **********search********** ;
9017  @TITLE,"SEARCH "
9018  PROCEDURE SEARCH ;
9019  COMMENT   determines rtmv through look-ahead. ;
9020  BEGIN
9021    INTEGER BEST, T_NODES, N, SIDE, DEEP, NEWLEVEL, NEWNODES, BRANCHES,
9022      NODE, B_SCR, TOTALS, E_NUM, FINAL, EXPECT, LOST, MINDEPTH,
9023      MAXDEPTH, ID, REUSE, ALPHA, BETA ;
9024    INTEGER INCR, T_TIME, TYPE, PLY, EXPECTATION, FIRSTONE, NODECOUNT ;
9025    LOGICAL SMALL, SACR_ANAL, SECOND, REVERSE, MARK, FULLWIN, INTERIOR ;
9026    INTEGER ARRAY OLDSC(-1::MAX_PLIES) ;
9027    INTEGER ARRAY MAXSC(-1::MAX_WID) ;
9028    INTEGER ARRAY AB(-1::MAX_PLIES) ;
9029    INTEGER ARRAY ALPHABET(0::2*MAXPLY+1) ;
```

```
9030   INTEGER ARRAY MINTREE(1::MAX_PLIES) ;
9031   INTEGER ARRAY LIST(0::ELIMIT) ;
9032
9033 PROCEDURE COLLECT(INTEGER VALUE NODE,ROOT) ;
9034 COMMENT    collects nodes from discarded tree. ;
9035 BEGIN
9036   INTEGER L ;
9037   IF NODE > 0
9038   THEN IF NODE > MAXTREE
9039        THEN WRITE("ERROR COLLECT", NODE)
9040        ELSE IF           (ROOT = PARENT(NODE) REM 1000)
9041             THEN BEGIN
9042               COMMENT retrieve least significant node of most extreme
9043                 subtree ;
9044       COMMENT gametree not want PARENT(NODE) := 0 ;
9045               L := NODETO(NODE,0) ;
9046               FOR I := L STEP -1 UNTIL 1
9047               DO COLLECT(NODETO(NODE,I),NODE) ;
9048 COMMENT display retrieved nodes;
9049     IF MAXL >= MAXTREE THEN BEGIN
9050       WRITE("COLL", MAXL, NODE, L, PARENT(NODE));
9051       FOR I := 1 UNTIL 10 DO WRITEON(GETNODE(I));
9052     END ELSE BEGIN
9053               MAXL := MAXL + 1 ;
9054               GETNODE(MAXL) := NODE ;
9055               STATES(NODE,4) := -ABS(STATES(NODE,4)) ;
9056               PARENT(NODE) := 1000*(PARENT(NODE) DIV 1000) ;
9057               NODETO(NODE,0) := IF L = 0
9058                                 THEN -FIVES
9059                                 ELSE -ABS(L) ;
9060   COMMENT IF MONITOR THEN WRITEON(NODE, ROOT, PARENT(NODE)) ;
9061     END;
9062   END ;
9063 END OFCOLLECT ;
9064
9065 LOGICAL PROCEDURE MARKTREE(INTEGER VALUE NODE, ROOT, DEPTH, TAG) ;
9066 BEGIN
9067   LOGICAL MARK ;
9068   MARK := FALSE ;
9069 COMMENT check for severe error, impossible node value;
9070   IF (NODE > 0)
9071   THEN IF NODE > MAXTREE
9072        THEN WRITE("ERROR RECOVERY ", ROOT, NODE, DEPTH,TAG)
9073   ELSE BEGIN
9074 COMMENT display marked nodes;
9075     COMMENT IF MONITOR THEN WRITE("MA", NODE, PARENT(NODE), DEPTH,
9076       NODETO(NODE,0)) ;
9077 COMMENT IF MONITOR THEN FOR I := 1 UNTIL NODETO(NODE,0)
9078                 DO WRITEON(NODETO(NODE,I));
9079     IF PARENT(NODE) > DEPTH
9080     THEN BEGIN
9081 NODECOUNT := NODECOUNT + 1;
9082       IF TAG ~= 0
9083       THEN STATES(ROOT,2) := 0 ;
9084       FOR I := NODETO(NODE,0) STEP -1 UNTIL 1
9085       DO IF MARKTREE(NODETO(NODE,I), NODE, DEPTH+1000, TAG)
9086          THEN NODETO(NODE,I) := 0 ;
9087       PARENT(NODE) := ROOT  + DEPTH ;
9088     END ELSE MARK := TRUE ;
9089   END ;
9090   MARK
9091 END OFISOLATIONOFNEXTTREE ;
9092
9093 @TITLE,"DEADTREE "
9094
9095 PROCEDURE DEADTREE (INTEGER VALUE REPLY) ;
9096 BEGIN
9097   COMMENT between move garbage collection underway;
9098   COMMENT WRITE("DEADTREE", REPLY, SAVEL, ORIGIN, NODETO(ORIGIN,0),
9099     NODETO(ORIGIN,REPLY)) ;
```

```
9100    PREVSCR := OLDSCR ;
9101    IF REPLY <= 0 OR REPLY > NODETO(ORIGIN,0) OR NODETO(ORIGIN,REPLY)
9102      <= 0
9103    THEN SAVEL := 0 ;
9104    IF (SAVEL > 0)
9105    THEN BEGIN
9106      POINT := 0 ;
9107      FOR I := SAVEL UNTIL MAXL
9108      DO BEGIN
9109        POINT := POINT + 1 ;
9110        GETNODE(POINT) := GETNODE(I) ;
9111      END ;
9112        COMMENT more unused nodes ;
9113        COMMENT retrieve current origin ;
9114      MAXL := POINT + 1 ;
9115      GETNODE(MAXL) := NODE := ORIGIN ;
9116      ORIGIN := NODETO(ORIGIN,REPLY) ;
9117        COMMENT will reduce depth of tree by 1 ;
9118      NODECOUNT := 0;
9119      MARK := MARKTREE(ORIGIN, NODE, 1000, 1) ;
9120        COMMENT mark remnant of current tree, in case duplicate positions
9121        ;
9122        COMMENT retrieve discarded sub trees ;
9123      FOR I := NODETO(NODE,0) STEP -1 UNTIL 1
9124      DO IF I ~= REPLY
9125         THEN COLLECT(NODETO(NODE,I),NODE) ;
9126      GETNODE(MAXL+1) := NIL ;
9127      SAVEL := 1 ;
9128      NODECOUNT := NODECOUNT + MAXL;
9129      IF  NODECOUNT < MAXTREE THEN BEGIN
9130        WRITE("INCONSISTENT ", MAXL, NODECOUNT, MAXTREE);
9131        IF FALSE AND MONITOR THEN
9132        FOR I := 0 UNTIL MAXTREE-1
9133        DO BEGIN
9134          IF (I REM 20) = 0
9135          THEN WRITE(" ");
9136          WRITEON(GETNODE(I+1));
9137        END;
9138        IOCONTROL(2);
9139          COMMENT MARK := MARKTREE(ORIGIN, NODE, 1000, 1) ;
9140      END;
9141        COMMENT savel is no. of nodes just added to tree ;
9142      OLDSCR := SCORE(TRY(REPLY)) ;
9143    END ELSE
9144    BEGIN
9145      MAXL := MAXTREE ;
9146      FOR I := 1 UNTIL MAXL
9147      DO GETNODE(I) := I ;
9148      ORIGIN := 0;
9149      OLDSCR := FIVES ;
9150        COMMENT   unexpected move ;
9151      MINDEPTH := MAXNODES := 2 ;
9152    END ;
9153   COMMENT IF MONITOR THEN WRITE("DEAD ");
9154   COMMENT IF MONITOR THEN FOR I := 1 UNTIL 15
9155               DO WRITEON(GETNODE(I));
9156  END OFDEADTREECOLLECTION ;
9157
9158  PROCEDURE INVALIDATE(INTEGER VALUE NODE) ;
9159  BEGIN
9160    COMMENT IF MONITOR THEN WRITE("INVAL", NODE, NODETO(NODE,0),
9161      PARENT(NODETO(NODE,1))) ;
9162    IOCONTROL(2) ;
9163    IF NODE > 0 AND NODE <= MAXTREE
9164    THEN BEGIN
9165      FOR I := 1 UNTIL NODETO(NODE,0)
9166      DO IF PARENT(NODETO(NODE,I)) DIV 1000 = NODE
9167         THEN INVALIDATE(NODETO(NODE,I)) ;
9168      STATES(NODE,2) := 0 ;
9169    END ;
```

```
9170 END OFINVALIDATE ;
9171
9172 @TITLE,"TREE "
9173 INTEGER PROCEDURE TREE(INTEGER VALUE  ALPHA, BETA, T_HASH, DEEPER ;
9174 LOGICAL VALUE PRINCVAR ;
9175 INTEGER RESULT TYPE) ;
9176 COMMENT generates the decision tree ;
9177 BEGIN
9178   INTEGER IP,TARGET,ROOT,IDONE,LIMIT, PASTSCR, LASTSCR, INCREASE,
9179     EST, MIN, SCR, REWN, BETTER, MX, LX, FIRST, F_T, INC, INIT_T_NODES,
9180     CONDITION, LARGE, B_SCR, T_SCR, LOWER ;
9181   LOGICAL OLD, ANOTHER, DYNAMIC, SHAH, FORCED, PARTIAL, MATER,
9182     DISCARD, GOOD, MORE, SEEN, FINISH, AGAIN ;
9183   INTEGER HIGHER, L_SCR, CNT, NEW_HASH, TMP ;
9184   LOGICAL EXTENSION ;
9185   INTEGER ARRAY REFSAVE (1::WSIZE) ;
9186   INTEGER EXCESS, SAVE_WIDTH;
9187
9188 PROCEDURE SHIFT(INTEGER VALUE IP, F, ROOT) ;
9189 BEGIN
9190   INTEGER LX, LAST ;
9191   STRING(5) NOTE ;
9192   IF IP > F
9193   THEN BEGIN
9194     NOTE := IF F = 1
9195             THEN "MOVED"
9196             ELSE "SHIFT" ;
9197     LX := TRY(IP) ;
9198     LAST := NODETO(ROOT, IP) ;
9199     FOR I := IP-1 STEP -1 UNTIL F
9200     DO BEGIN
9201       TRY(I+1) := TRY(I) ;
9202       NODETO(ROOT, I+1) := NODETO(ROOT, I) ;
9203     END ;
9204     TRY(F) := LX ;
9205     NODETO(ROOT, F) := LAST ;
9206     SAVE(ROOT) ;
9207     IF TEST
9208     THEN WRITE(NOTE,LEVEL, IP, F, ROOT, NODETO(ROOT,0), NODETO(ROOT,F),
9209       NODETO(ROOT, F+1)) ;
9210 IF TEST AND IP > NODETO(ROOT,0) AND F <= NODETO(ROOT,0)
9211 THEN WRITEON("WOW ");
9212   END ;
9213 END OF_SHIFT ;
9214 @TITLE,"FIRSTENTRY "
9215 PROCEDURE FIRSTENTRY (INTEGER VALUE ROOT) ;
9216 BEGIN
9217   INTEGER TEMP ;
9218   EXPECT := SCORE(TRY(1)) ;
9219   TEMP :=  + SCALE*ONEVAL ;
9220 COMMENT display parameters at beginning of each iteration;
9221   IF TEST
9222   THEN WRITE("FIRST",ONEVAL, EXPECT, TEMP, MINDEPTH, MAXDEPTH,
9223     DEEP, MAXNODES, NUM, TRY(0), TRY(WSIZE), WIDTH) ;
9224   FOR I := 1 UNTIL MAX_PLIES
9225   DO MINTREE(I) := IF I < 3
9226                    THEN MAXNODES
9227                    ELSE MINTREE(I-1) DIV 2 ;
9228 END OFFIRSTENTRY ;
9229
9230 PROCEDURE REPETITION(INTEGER VALUE NODE, ROOT, LEVEL) ;
9231 BEGIN
9232   COMMENT repetition, looking for a cycle;
9233   COMMENT IF MONITOR THEN WRITE("R", NODE, ROOT, LEVEL, NODETO(NODE,0))
9234     ;
9235   IF (FINAL ~= 3)  AND LEVEL <= LENGTH AND NODE > 0
9236   THEN FOR I := 1 UNTIL NODETO(NODE,0)
9237   DO IF (NODETO(NODE,I) = ROOT)
9238      THEN FINAL := 3
9239      ELSE REPETITION(NODETO(NODE,I), ROOT, LEVEL+1) ;
```

```
9240 END OFREPETITIONTEST ;
9241
9242 PROCEDURE LOCATE(INTEGER VALUE START, NODE, ROOT, DEPTH) ;
9243 BEGIN
9244   COMMENT IF MONITOR THEN WRITE("LOCATE ", START, NODE, ROOT, DEPTH,
9245     PARENT(START), NODETO(START,0)) ;
9246   IF (START > 0) AND (START ~= ROOT) AND DEPTH < PARENT(START)
9247   THEN IF (START = NODE)
9248        THEN BEGIN
9249          REPETITION(START, ROOT,1) ;
9250          IF (FINAL ~= 3) AND (LEVEL*1000 > PARENT(NODE))
9251          THEN FINAL := 1 ;
9252            COMMENT IF MONITOR AND (FINAL = 1)
9253            THEN WRITEON(PARENT(NODE), LEVEL, NODE, ROOT) ;
9254   END ELSE FOR I := 1 UNTIL NODETO(START,0)
9255   DO IF (FINAL = 2)
9256      THEN LOCATE(NODETO(START,I), NODE, ROOT, DEPTH+1000) ;
9257 END OFLOCATE ;
9258
9259 @TITLE,"BUILD_SACR_TREE"
9260 INTEGER PROCEDURE BUILD_SACR_TREE(INTEGER VALUE ROOT, DEPTH, BOUND,
9261     SAC_ALPHA, SAC_BETA; INTEGER VALUE RESULT IDONE) ;
9262 BEGIN
9263   INTEGER BESTSCR ;
9264   INTEGER ARRAY SAVER(0::W_LIM) ;
9265   INTEGER ARRAY ALPHABET(0::DEPTH+1) ;
9266   LOGICAL MAYBEMATE, NARROW, EXCLUDE, EXTEND, FIRST;
9267
9268   INTEGER PROCEDURE SACR_TREE(INTEGER VALUE PLY, AA, BB;
9269   INTEGER ARRAY SAVER(*)) ;
9270   BEGIN
9271     INTEGER ARRAY MOVES(0::W_LIM) ;
9272     STRING(140) BUFFER ;
9273     INTEGER BESTSCR, SCR, KK, N, DONE, SC, MT, NODE, REWN, ND,
9274         TEMP_HASH;
9275     LOGICAL SAVECH, GOOD, PARTIAL ;
9276     INTEGER ARRAY SAVECL(-1::1) ;
9277     KK := IF PLY REM 2 = 0
9278           THEN -1
9279           ELSE 1 ;
9280     ALTERNATIVE := IF PLY > 1
9281                    THEN TRUE
9282                    ELSE FALSE ;
9283   TEMP_HASH := HASH;
9284     UPPER := BB;
9285     LISTLEGALMOVES (FALSE) ;
9286     PARTIAL := UPPER ~= EIGHTS ;
9287     N := TRY(1) ;
9288     IF ALTERNATIVE
9289     THEN SAVE_THE_MOVES(SAVER) ;
9290     IF (NUM = 0)
9291     THEN BEGIN
9292       IF CON(-K,KINGSQ(K)) = 0
9293       THEN BESTSCR := DRAW
9294       ELSE BESTSCR := -NINES;
9295     END ELSE BEGIN
9296       SCR := SCORE(N);
9297                 IF ~FIRST THEN IF PLY > 1 OR
9298                    SCR > AA AND SCR < BB OR
9299                         INCHECK AND SCR > AA
9300                 THEN BEGIN
9301                   REUSE := REUSE-1 ;
9302                   IF REUSE <= POINT
9303                   THEN REUSE := MAXL ;
9304                   NODE := GETNODE(REUSE) ;
9305                   IF NODE > 0 AND NODE <= MAXTREE
9306                   THEN IF POINT < MAXL
9307                        THEN BEGIN
9308                    GOOD := TRUE ;
9309                    HASH := TEMP_HASH;
```

```
9310                    SAVESTATE(NODE) ;
9311                    IF PP_STATUS > 0 AND K*CON(0,M3) < 0
9312                    THEN SCR := SCR - SCALE*PP_STATUS DIV
9313                            (IF CON(K,M3) ~= 0
9314                             THEN 2
9315                             ELSE 1) ;
9316                    STATES(NODE,5) := IF PARTIAL THEN 1 ELSE 0 ;
9317                    STATES(NODE,2) := 1 ;
9318                   NODETO(NODE,0) := 0;
9319                    STATES(NODE,4) := 1 ;
9320                    PARENT(NODE) := 1000*LENGTH ;
9321                    IF TEST
9322                    THEN WRITE("REUSE", LEVEL, POINT, REUSE, MAXL,NODE,
9323                      AA, SCR, BB, CREDIT(LEVEL), DEBIT(LEVEL-1)) ;
9324                  END ELSE NODE := 0 ;
9325                END ELSE NODE := 0 ;
9326       FIRST := FALSE;
9327             IF (PLY <= 1)
9328             THEN BEGIN
9329               ND := 1 ;
9330 LOOP:
9331               SCR := SCORE(N) ;
9332               REWN := REWARD(N) ;
9333               IF TEST
9334               THEN PRINTLINE(N, LEVEL) ;
9335               IF TEST
9336               THEN WRITEON(IF PARTIAL
9337                            THEN "%"
9338                            ELSE " ", CREDIT(LEVEL), DEBIT(LEVEL-1),
9339                              CLEARPAWN(-K), AA, BB, TEMP_HASH) ;
9340               IF ABS(REWN) >= DISCHECK
9341                 OR SCR < BB
9342                  OR INCHECK AND NUM < 5
9343           OR EXTEND
9344                    OR ABS(REWARD(N)) >= REALCHECK
9345                 AND ABS(CON(-K,KINGSQ(-K))) <= 1 AND
9346                 K*SEC(0,MOVETO(N)) > 0
9347               THEN BEGIN
9348                 GOOD := FALSE ;
9349                   PUNCHBRD(TRUE, BUFFER) ;
9350                   IF (REWN < 0 AND ABS(REWN) >= REALCHECK
9351            OR EXTEND
9352                     OR INCHECK)
9353                     AND (LEVEL <= 4) AND SCR < SEVENS
9354                   THEN BEGIN
9355                     DONE := NODETO(0,0) := 0 ;
9356                     SAVESTATE(0) ;
9357                     MAKEMOVE(N, TRUE) ;
9358                     SCR := -BUILD_SACR_TREE(0, 2, BOUND,
9359                             -BB, -AA, DONE);
9360 IF TEST THEN
9361 WRITE("EXTEND ", SCR, ALPHA, BETA);
9362                     SETBOARD(BUFFER) ;
9363                     IF ~STARTTHISPROBLEM(TRUE)
9364                     THEN GOTO STARTS ;
9365                     IF GOOD AND ABS(SCR) > SEVENS
9366                     THEN STATES(NODE, T_SIZE+2+ND) := SCR -LEVEL;
9367 COMMENT if poor score returned from 2-ply extension
9368         look at first alternative;
9369                     ND := ND + 1 ;
9370                     N := TRY(ND) ;
9371                     IF ND = 2 AND TRY(WSIZE) > 1 AND
9372                       SCR < SCORE(N)
9373                     THEN GO TO LOOP ;
9374                   END ;
9375               END ;
9376               BESTSCR := SCR ;
9377               CAP := CAP + 1 ;
9378     END ELSE
9379     BEGIN
```

```
9380        MAYBEMATE := INCHECK AND NUM < WIDTH ;
9381        BESTSCR := -EIGHTS ;
9382        PUNCHBRD(TRUE, BUFFER) ;
9383        FOR J := -1 UNTIL 1
9384        DO SAVECL(J) := CLEARPAWN(J) ;
9385        FOR I := 1 STEP 4 UNTIL SAVER(0)
9386        DO BEGIN
9387          MOVEFROM(TOTAL) := SAVER(I) ;
9388          MOVETO(TOTAL)   := MT := SAVER(I+1) ;
9389          REWARD(TOTAL)   := SAVER(I+2) ;
9390          SCORE(TOTAL) := SC := SAVER(I+3) ;
9391          IF (SC > NOSCR)
9392          THEN BEGIN
9393            IF TEST
9394            THEN PRINTLINE(TOTAL, LEVEL) ;
9395            MAKEMOVE(TOTAL, TRUE) ;
9396            IF TEST
9397            THEN WRITEON(IF PARTIAL
9398                         THEN "%"
9399                         ELSE " ", CREDIT(LEVEL), DEBIT(LEVEL-1)) ;
9400            SAVECH := INCHECK ;
9401            K := -K ;
9402            LEVEL := LEVEL +1 ;
9403            SCR := -SACR_TREE(PLY-1, -BB, -AA, MOVES) ;
9404            K := -K ;
9405            SETBOARD(BUFFER) ;
9406            IF ~STARTTHISPROBLEM(TRUE)
9407            THEN GO TO STARTS ;
9408            LEVEL := LEVEL -1 ;
9409            INCHECK := SAVECH ;
9410            FOR J := -1 UNTIL 1
9411            DO CLEARPAWN(J) := SAVECL(J) ;
9412          END ELSE SCR := -FIVES ;
9413          SAVER(I+3) := SCR ;
9414          IF SCR > BESTSCR
9415          THEN BESTSCR := SCR ;
9416          IF BESTSCR > BB
9417          THEN BEGIN
9418            IF TEST
9419            THEN WRITE("CUT-OFF", INCHECK, SC, SCR, BB,BOUND, KK, PLY);
9420            SAVER(0) := I ;
9421            GO TO PRUNE ;
9422          END ;
9423          IF BESTSCR > AA
9424          THEN AA := BESTSCR;
9425          IF NARROW AND I > 5
9426          THEN GO TO PRUNE ;
9427        END ;
9428  PRUNE:
9429      END ;
9430  END;
9431      BESTSCR
9432    END OFSACR_TREE ;
9433
9434    PROCEDURE INSERT_NEW_MOVES(INTEGER ARRAY SAVER(*)) ;
9435    BEGIN
9436      LOGICAL NEW, PROMOTION, SHUFFLE;
9437      INTEGER  MF, MT, SCR, JJ, TW, S, TT, RW, N, R, SAVED, INC,
9438        IN, TOP, NN, BEST_J, MINSCR, TEMP, J_MAX;
9439    J_MAX := IF NODETO(ROOT,0) > 4
9440             THEN 4 ELSE NODETO(ROOT,0);
9441  COMMENT IF TEST AND LEVEL <= 2
9442  THEN TRACER(1, "?D");
9443      TW := TRY(WSIZE) ;
9444  COMMENT display current movelist;
9445      IF TEST AND SAVER(0) > 0 THEN
9446          FOR J := TW STEP -1 UNTIL 1
9447          DO BEGIN
9448            N := TRY(J) ;
9449            WRITE(".....",N, MOVEFROM(N),MOVETO(N), REWARD(N),
```

```
9450              SCORE(N)) ;
9451      END;
9452      JJ := K ;
9453      TT := 0 ;
9454      FOR I := SAVER(0) STEP -4 UNTIL 1
9455      DO BEGIN
9456        MINSCR := SCORE(TRY(TW)) ;
9457        TOP := SCORE(TRY(1));
9458        BEST_J := 1;
9459        PROMOTION := FALSE ;
9460        NEW := TRUE ;
9461        S := TW+1 ;
9462        TT := TT +1 ;
9463        MF := SAVER(I) ;
9464        MT := SAVER(I+1) ;
9465        RW := SAVER(I+2) ;
9466        SCR := SAVER(I+3) ;
9467        IF ABS(SCR) ~= FIVES
9468        THEN BEGIN
9469          FOR J := TW STEP -1 UNTIL 1
9470          DO BEGIN
9471            N := TRY(J) ;
9472            IF MINSCR > SCORE(N)
9473            THEN MINSCR := SCORE(N) ;
9474           IF TOP < SCORE(N)
9475           THEN BEGIN
9476             TOP := SCORE(N);
9477             BEST_J := J;
9478           END;
9479            IF (MF = MOVEFROM(N)) AND (MT = MOVETO(N))
9480            THEN BEGIN
9481              R :=  ABS (ABS(RW) - ABS(REWARD(N))) REM CHECKING ;
9482              IF R = 0 OR R = 2 AND (RW = 0)
9483              THEN BEGIN
9484                NEW := FALSE ;
9485                IF RW > PFTWO
9486                THEN PROMOTION := TRUE ;
9487                S := J ;
9488                RW := REWARD(N) ;
9489                NN := N ;
9490              END ;
9491            END ;
9492          END ;
9493  COMMENT display sacrificial move;
9494          IF TEST
9495          THEN WRITE("COMP",MF,MT,RW,SCR, MINSCR, TOP, BEST_J) ;
9496          IF SCR > TOP AND SCR < SIXES
9497          THEN SCR := TOP + (PAWNVALUE*(SCR-TOP)) DIV (SCR-TOP
9498            + PAWNVALUE) ;
9499          IF NEW AND SCR > TOP AND SCR < SIXES
9500          THEN SCR := TOP -1 ;
9501          IF ( ~NEW OR SCR > MINSCR)
9502          THEN WHILE S > 0
9503               DO BEGIN
9504  COMMENT the moveliist is divided into two parts
9505          a. Moves searched by 3-ply tree
9506          b. Unsearched 1-ply moves.
9507       Part a moves may have lower scores than those from part b;
9508                FOR J := 1 UNTIL TW
9509                DO IF SCR >= SCORE(TRY(J)) AND (SCR > TOP OR
9510                          J > J_MAX OR J > BEST_J) OR ~NEW AND J = TW
9511                   THEN BEGIN
9512                     IN := IF S < J
9513                           THEN J-1
9514                           ELSE J ;
9515                     IF TEST AND S ~= IN
9516                     THEN WRITE("INSERT", S,IN,J,IDONE, K,MF,MT,
9517                       NEW,TT,TW, JJ, SCORE(TRY(1)), SCR, SCORE(TRY(J)));
9518                     IF NEW
9519                     THEN BEGIN
```

```
9520                        S := TW ;
9521                        IF S <= IDONE
9522                        THEN DISCARD_SUBTREES(S) ;
9523                        IF NODETO(ROOT,0) >= S THEN
9524                        COLLECT(NODETO(ROOT,S), ROOT) ;
9525                        NODETO(ROOT, S) := 0 ;
9526                        TRY(S) := JJ ;
9527                        MOVEFROM(JJ) := MF ;
9528                        MOVETO(JJ) := MT ;
9529                        REWARD(JJ) := RW ;
9530                        SCORE(JJ) := SCR ;
9531                        JJ := JJ +K ;
9532                      END ELSE SCORE(NN) := SCR ;
9533                      SHUFFLE := FALSE;
9534                      IF S < IN
9535                      THEN BEGIN
9536                        SHUFFLE := TRUE;
9537                        TEMP := S;
9538                        IF S + 5 < IN
9539                        THEN S := S + 5
9540                        ELSE S := IN;
9541                        IN := TEMP;
9542                      END;
9543                      IF S > IDONE
9544                      THEN BEGIN
9545                        IF IN <= IDONE
9546                        THEN BEGIN
9547                          DISCARD_SUBTREES(IN) ;
9548                          IDONE := IDONE + 1 ;
9549                          NODETO(ROOT, 0) := IDONE ;
9550                        END ;
9551                        NODETO(ROOT, S) := 0 ;
9552                      END ;
9553 COMMENT is the target move advancing or declining in score;
9554                      IF ~SHUFFLE
9555                      THEN SHIFT(S, IN, ROOT)
9556 ELSE BEGIN
9557   IF S <= IDONE AND IN > IDONE
9558   THEN BEGIN
9559     COLLECT(NODETO(ROOT,S), ROOT);
9560     NODETO(ROOT,S) := -1;
9561     IDONE := IDONE -1;
9562     IF MONITOR THEN WRITE("REDUCE ", ROOT, S, IDONE);
9563   END;
9564                          FOR I := IN UNTIL S-1
9565                          DO SHIFT(I+1, I, ROOT);
9566 END;
9567                      GOTO INSERTED ;
9568                END ;
9569 INSERTED:
9570                      S := 0 ;
9571         END ;
9572         TRY(WSIZE) :=  TW ;
9573       END ;
9574     END ;
9575     SAVE(ROOT) ;
9576   END OFINSERT_NEW_MOVES ;
9577
9578   PROCEDURE SAVE_THE_MOVES(INTEGER ARRAY SAVER(*)) ;
9579   BEGIN
9580     INTEGER ARRAY HOLD (1::MAX_WID) ;
9581     LOGICAL DELETE ;
9582     INTEGER J, JJ, MOVE, DONE ;
9583     DONE := NODETO(ROOT,0) ;
9584 IF DONE > WIDTH THEN DONE := WIDTH;
9585     IF EXCLUDE
9586     THEN FOR L := 1 UNTIL DONE
9587     DO HOLD(L) := ABS(STATES(ROOT, T_SIZE+2+MAX_WID+L)) REM 65536 ;
9588 IF FALSE AND TEST AND LEVEL <= 2 THEN BEGIN
9589   WRITE("SAVE_THE ", DONE);
```

```
9590    FOR L := 1 UNTIL DONE DO WRITEON(HOLD(L));
9591  END;
9592      J := -3 ;
9593      FOR I := 1 UNTIL TRY(0)
9594      DO BEGIN
9595        J := J + 4 ;
9596        JJ := TRY(I) ;
9597  IF FALSE AND TEST AND LEVEL <= 2
9598  THEN PRINTLINE(JJ, LEVEL);
9599        DELETE := FALSE ;
9600        IF EXCLUDE
9601        THEN BEGIN
9602          MOVE := MOVEFROM(JJ) + 256*MOVETO(JJ) ;
9603          FOR L := 1 UNTIL DONE
9604          DO IF ~ DELETE
9605             THEN BEGIN
9606               IF HOLD(L) = MOVE
9607               THEN DELETE := TRUE ;
9608          END ;
9609        END ;
9610        IF DELETE
9611        THEN J := J - 4
9612        ELSE BEGIN
9613          SAVER(J) := MOVEFROM(JJ) ;
9614          SAVER(J+1) := MOVETO(JJ) ;
9615          SAVER(J+2) := REWARD(JJ) ;
9616          SAVER(J+3) := SCORE(JJ) ;
9617        END ;
9618      END ;
9619      SAVER(0) := J ;
9620      EXCLUDE := FALSE;
9621    END OFSAVE_THE_MOVES ;
9622
9623    PROCEDURE DISCARD_SUBTREES(INTEGER VALUE FIRST) ;
9624    BEGIN
9625      INTEGER SAVE_P ;
9626      SAVE_P := POINT +1;
9627      FOR I := FIRST STEP -1 UNTIL 1
9628      DO MARK := MARKTREE(NODETO(ROOT,I), ROOT, LEVEL*1000, 0) ;
9629      IF SAVE_P = 0
9630      THEN POINT := MAXL
9631      ELSE BEGIN
9632        REUSE := REUSE -POINT  ;
9633        POINT := 0 ;
9634        FOR I := SAVE_P UNTIL MAXL
9635        DO BEGIN
9636          POINT := POINT + 1 ;
9637          GETNODE(POINT) := GETNODE(I) ;
9638        END ;
9639      END;
9640      IF TEST
9641      THEN WRITE("DISC", FIRST,IDONE, ROOT,LEVEL,SAVE_P, MAXL,REUSE) ;
9642        IF TEST THEN WRITEON(POINT,GETNODE(1)) ;
9643      MAXL := POINT ;
9644        POINT := 0;
9645      GETNODE(MAXL+1) := 0 ;
9646      DISCARD := TRUE ;
9647    END OFDISCARD_SUBTREES ;
9648
9649    IF DEPTH <= MAXPLY AND LEVEL > 4
9650    THEN BEGIN
9651      IF SAC_ALPHA < BOUND AND SAC_BETA > BOUND
9652      THEN SAC_BETA := -BOUND;
9653    END;
9654    IF DEPTH <= MAXPLY
9655    THEN BEGIN
9656      NARROW := LEVEL > 2 AND ~INCHECK ;
9657      LEVEL := LEVEL + 1 ;
9658      K := -K ;
9659    END ELSE
```

```
9660      IF SAC_ALPHA < BOUND AND SAC_BETA > BOUND
9661      THEN SAC_ALPHA := BOUND;
9662    EXCLUDE := LEVEL < MAXDEPTH;
9663      EXTEND := LEVEL =2;
9664        FIRST := TRUE;
9665    BESTSCR := SACR_TREE(IF EXTEND THEN 2 ELSE DEPTH,
9666           SAC_ALPHA, SAC_BETA, SAVER);
9667    IF DEPTH <= MAXPLY
9668    THEN BEGIN
9669      LEVEL := LEVEL -1 ;
9670      K := -K ;
9671    END ;
9672    STATES(ROOT,3) := CAPT_T := 1 ;
9673    RESTORESTATE(ROOT) ;
9674    IF  DEPTH > MAXPLY
9675    THEN INSERT_NEW_MOVES(SAVER)
9676    ELSE SCORE(TRY(1)) :=  BESTSCR ;
9677    ALTERNATIVE := FALSE ;
9678    SCORE(TRY(1)) -DEPTH
9679  END OFBUILD_SACR_TREE ;
9680
9681  @TITLE,"REFUTE"
9682  PROCEDURE REFUTE(INTEGER VALUE MV) ;
9683  BEGIN
9684    INTEGER I, DENIER;
9685    DENIER := I := REFUT(LEVEL,0);
9686    WHILE I > 0
9687    DO BEGIN
9688      IF REFUT(LEVEL,I) = MV
9689      THEN GOTO DELTA ;
9690      I := I - 1 ;
9691    END ;
9692    IF DENIER < 10
9693    THEN BEGIN
9694      I := REFUT(LEVEL,0) := DENIER +1;
9695      REFUT(LEVEL,I) := MV;
9696    END;
9697  DELTA:
9698    REFCNT(LEVEL,I) := REFCNT(LEVEL,I) + 1 ;
9699    IF TEST
9700    THEN WRITE("REFUT ", LEVEL, MV, I, DENIER, REFCNT(LEVEL,I));
9701  IF LEVEL > 1 THEN
9702       FOR I := LEVEL UNTIL MAX_PLIES
9703       DO REFTAB(PATH+LEVEL-2,I) := REFTAB(PATH+LEVEL-1,I);
9704  END OFREFUTE ;
9705
9706  @TITLE,"DUPLICATE(INTEGER VALUE NODE)"
9707  LOGICAL PROCEDURE DUPLICATE(INTEGER VALUE NODE) ;
9708  BEGIN
9709    INTEGER J, LL ;
9710    LOGICAL DUPLICATE ;
9711    DUPLICATE := TRUE ;
9712         LL := POINT +1;
9713    FOR I := LL UNTIL MAXL
9714    DO IF (NODE = GETNODE(I))
9715       THEN BEGIN
9716         REC := REC + 1 ;
9717         REC_S := REC_S + ABS(STATES(NODE,4)) ;
9718         J := GETNODE(I) ;
9719         GETNODE(I) := GETNODE(LL) ;
9720         GETNODE(LL) := J ;
9721         FINAL := 5 ;
9722      NODETO(NODE,0) := NODETO(NODE,1) := 0;
9723         GO TO QUIT ;
9724    END ;
9725    FINAL := 2 ;
9726    DUP := DUP +1 ;
9727  QUIT:
9728    IF DUPLICATE AND TEST
9729    THEN WRITEON(NODE, IF (FINAL = 2)
```

```
                      THEN "DUPLIC. "
                      ELSE "FLOATER ", PARENT(NODE), STATES(NODE,2),
                        STATES(NODE,T_SIZE+2+1), "&") ;
  DUPLICATE
END OFDUPLICATEPOSITIONCHECKER ;

@TITLE,"NODAL"
LOGICAL PROCEDURE NODAL ;
BEGIN
  INTEGER D ;
   COMMENT   check for duplicate position ;
  IF OLD AND NODE > 0 AND NEW_HASH = STATES(NODE, T_SIZE)
  THEN GO TO FOUNDUPLICATE ;
  FOR NODEX := 1 UNTIL MAXTREE
  DO IF NEW_HASH = STATES(NODEX,T_SIZE) AND K ~= STATES(NODEX,1)
     THEN IF OLD OR FINAL = 5 OR ROOT ~= PARENT(NODEX) REM 1000
          THEN BEGIN
            NODE := NODEX ;
            IF (DUPLICATE(NODE))
            THEN GO TO FOUNDUPLICATE ;
  END ;
  FINAL := 0 ;
  OLD := FALSE ;
FOUNDUPLICATE:
  IF REVERSIBLE AND (ENDGAME OR FINAL > 0 OR LEVEL <= 2)
  THEN BEGIN
    D := 0;
    FOR J := 1 UNTIL ENDPLAY
    DO IF (NEW_HASH = REPEATS(J))
       THEN D := D + 1;
    FOR J := 1 UNTIL LEVEL -1
    DO IF NEW_HASH = STACK_HASH(J)
       THEN D := D +1;
    IF D > 0
       THEN BEGIN
         ANOTHER := TRUE ;
         SCR := STATES(ROOT, T_SIZE+2+IP);
         IF TEST
         THEN WRITE("DRAWING",D,FINAL,IP,ROOT,SCR,NEW_HASH,OLD) ;
         IF D > 1 OR SCR <= DRAW AND SIDE = -1
         THEN FINAL := 3
         ELSE FINAL := 1;
         IF FINAL = 3 THEN
         STATES(ROOT,T_SIZE+2+IP) := -SIDE*DRAW ;
    END ;
  END ;
  IF (FINAL > 0)
  THEN BEGIN
    IF (FINAL = 2) AND (LEVEL > 2)
    THEN IF (NODE = ORIGIN)
         THEN FINAL := 3
         ELSE LOCATE(NODETO(ORIGIN,PATH), NODE, ROOT, 1000) ;
    IF (FINAL = 2)
    THEN IF (1000*LEVEL > PARENT(NODE))
         THEN FINAL := 1 ;
    IF (FINAL = 3) OR (FINAL = 1)
    THEN BEGIN
      COMMENT repetition loop detected, draw ;
      SCR := IF (FINAL = 3)
             THEN DRAW
             ELSE -STATES(NODE, T_SIZE+2+1) ;
      IF FINAL = 1
      THEN OLD := TRUE ;
         COMMENT reject ;
      IF FINAL = 3 OR SIDE = -1
      THEN ANOTHER := TRUE ;
      GO TO EXIT ;
    END ;
    D := ABS(STATES(NODE,4)) ;
    IF D = 0 AND FINAL ~= 3
```

```
9800      THEN WRITE("INCONSISTENT DRAW", FINAL, D) ;
9801      IF (FINAL = 2)
9802      THEN OLD := TRUE ;
9803      IF OLD OR FINAL = 3
9804      THEN DUP_S := DUP_S + (IF D = 0
9805                             THEN 1
9806                             ELSE D) ;
9807    END ;
9808  EXIT:
9809    ANOTHER
9810  END OFNODAL ;
9811
9812  @TITLE,"MAKE_SPACE"
9813  PROCEDURE MAKE_SPACE(INTEGER VALUE ORIGIN,  REPLY, PATH, DEPTH,EST) ;
9814  BEGIN
9815    INTEGER TEMP, LX, NOD, OLDMAXL;
9816    IF POINT = 0
9817    THEN POINT := MAXL
9818    ELSE BEGIN
9819      LX := POINT+1;
9820      REUSE := REUSE - POINT  ;
9821      POINT := 0 ;
9822      FOR I := LX UNTIL MAXL
9823      DO BEGIN
9824        POINT := POINT + 1 ;
9825        GETNODE(POINT) := GETNODE(I) ;
9826      END ;
9827      MAXL := POINT ;
9828    END;
9829    FIRST := IF ORIGIN > 0
9830             THEN NODETO(ORIGIN,0)
9831             ELSE 0 ;
9832     COMMENT mark subtree of best move.  Note cannot mark subtree
9833     of current path, since part of it may have been discarded and
9834     re-used elsewhere.  See 12 Feb. 76 Belzerowski game ;
9835    LX := IF REPLY > 0 AND REPLY ~= PATH
9836          THEN REPLY
9837          ELSE IF PATH = 0
9838               THEN 1
9839               ELSE 0 ;
9840  COMMENT node collection, about to regain space;
9841    IF TEST OR ORIGIN < 0
9842    THEN WRITE("MAKE_SPACE ", ORIGIN, REPLY, PATH, FIRST, LX,MAXL,EST,
9843        GETNODE(1));
9844    FOR I := FIRST STEP -1 UNTIL 1
9845    DO IF I ~= LX AND I ~= PATH
9846       THEN MARK := MARKTREE(NODETO(ORIGIN,I), ORIGIN, DEPTH, 0) ;
9847    IF LX > 0
9848    THEN MARK := MARKTREE(NODETO(ORIGIN,LX),ORIGIN, DEPTH, 0) ;
9849    IF PATH > 0
9850    THEN MARK := MARKTREE(NODETO(ORIGIN,PATH), ORIGIN, DEPTH, 0) ;
9851    FOR LX := FIRST STEP -1 UNTIL 1
9852    DO IF REPLY <= 0 AND PATH <= 0 AND (LX ~= 1 OR PATH < 0)
9853       THEN BEGIN
9854         COLLECT(NODETO(ORIGIN,LX), ORIGIN) ;
9855         NODETO(ORIGIN, LX) := -1 ;
9856         IF TEST
9857         THEN WRITE("FULL PURGE", ROOT, LX, MAXL) ;
9858    END ELSE IF (MAXL <= EST) AND (LX ~= REPLY) AND (LX ~= PATH)
9859              THEN BEGIN
9860                TEMP := NODETO(ORIGIN,LX) ;
9861                COMMENT IF TEMP < 0 AND ~ TOURNAMENT
9862                THEN WRITE("SUBTREE DISCARD", ORIGIN, LX, MAXL,TEMP);
9863                IF TEMP > 0
9864                THEN BEGIN
9865                     OLDMAXL := MAXL;
9866                  FOR J :=  (IF REPLY = 0 AND PATH = 0
9867                             THEN 2
9868                             ELSE 1) UNTIL NODETO(TEMP,0)
9869                  DO BEGIN
```

```
9870                  COLLECT(NODETO(TEMP, J), TEMP) ;
9871                  NODETO(TEMP, J) := -1 ;
9872                IF TEST AND MAXL > OLDMAXL
9873                THEN WRITE("SUBTREE DISCARD", LX,J,NODETO(TEMP,J),MAXL);
9874                END ;
9875              END ;
9876    END ;
9877    IF (IP <= MIN) AND (REPLY > 0) AND (MAXL <= EST) AND (LEVEL
9878      <= 2 OR LEVEL = 3 AND IP < 2 OR INCHECK)
9879    THEN IF REPLY ~= PATH
9880         THEN BEGIN
9881           TEMP := NODETO(ORIGIN,REPLY) ;
9882  COMMENT about to take some nodes from the PV;
9883           IF TEST
9884           THEN WRITE("REPLY DISCARD", TEMP) ;
9885           IF TEMP > 0
9886           THEN FOR LX := NODETO(TEMP,0) STEP -1 UNTIL 1
9887           DO IF MAXL <= EST
9888              THEN BEGIN
9889                COLLECT(NODETO(TEMP,LX),TEMP) ;
9890                NODETO(TEMP,LX) := -1 ;
9891                IF TEST
9892                THEN WRITEON(LX, MAXL) ;
9893           END ;
9894    END ELSE IF MAXL = 0
9895             THEN BEGIN
9896               NOD := NODETO(ORIGIN,1) ;
9897               IF NOD = ROOTNODE((DEPTH+1000) DIV 1000)
9898               THEN BEGIN
9899                 MAXL := 0 ;
9900                 WRITE("SERIOUS", LEVEL, DEPTH, EST, NOD) ;
9901                 FOR LX := 1 UNTIL LEVEL
9902                 DO WRITEON(ROOTNODE(LX)) ;
9903                 FOR I := 1 UNTIL NODETO(NOD,0)
9904                 DO IF NODETO(NOD,I) = ROOTNODE((DEPTH+2000) DIV 1000)
9905                    THEN MAKE_SPACE(NOD, 1, I, DEPTH+1000, EST) ;
9906               END ;
9907    END ;
9908    IF TEST
9909    THEN WRITEON("####", LEVEL, DEPTH, REPLY, PATH, EST, REUSE,MAXL,
9910      POINT, GETNODE(1)) ;
9911    POINT := 0 ;
9912    GETNODE(MAXL+1) := 0 ;
9913  END OF_MAKE_SPACE ;
9914  @TITLE,"   LOOKAHEAD"
9915  PROCEDURE LOOKAHEAD ;
9916  BEGIN
9917    IP := 0 ;
9918    B_SCR := -EIGHTS ;
9919    EXTENSION := FALSE ;
9920    WHILE (IP < LIMIT) AND (WHO OR FLAG(1) = 0)
9921    DO BEGIN
9922      IP := IP + 1 ;
9923      N := TRY(IP) ;
9924      SCR := SCORE(N) ;
9925      REWN := REWARD(N) ;
9926      IF IP = MIN AND (SCR <= SCORE(TRY(MIN+1)) OR MIN < TRY(0) AND
9927        LEVEL <= 2)
9928      THEN MIN := MIN +1 ;
9929      OLD := IF (IP > IDONE) AND ~AGAIN
9930             THEN FALSE
9931             ELSE TRUE ;
9932      IF LEVEL = 1
9933      THEN T_TIME := TIME(1) ;
9934      IF TEST
9935      THEN BEGIN
9936        PRINTLINE(N, LEVEL-1) ;
9937        WRITEON(IF PARTIAL
9938                THEN "%"
9939                ELSE " ", IF OLD
```

```
                          THEN -12345
                          ELSE PASTSCR, CLEARPAWN(-K), CLEARPAWN(K),
                            ALPHA, BETA, HASH) ;
    END ;
    IF OLD
    THEN BEGIN
      NODE := NODETO(ROOT,IP) ;
      IF (NODE < 0)
      THEN BEGIN
        OLD := FALSE ;
        FINAL := -3 ;
      END ELSE IF (PARENT(NODE) REM 1000 ~= ROOT)
                THEN NODE := NIL ;
    END ELSE NODETO(ROOT, IP) := 0 ;
  SEEN := QUIES(TRY(IP));
    ANOTHER := FALSE ;
    SHAH := IF (ABS(REWN) < REALCHECK)
            THEN FALSE
            ELSE TRUE ;
   COMMENT also expand tree if large epgain ;
    IF (LEVEL = 1)
    THEN BEGIN
      PATH := IP ;
      PLEX := 0 ;
      MX := AB(1) := -NINES ;
      MAXSC(IP) := SCR-1 ;
      INIT_T_NODES := T_NODES := 0 ;
      IF MAXDEPTH = 1
      THEN BEGIN
        REPLY := 1 ;
        FINAL := 0 ;
        MX := BEST := B_SCR := SCR ;
        IF (IDONE > 0)
        THEN GO TO QUIT ;
        NODETO(ROOT,0) := 1 ;
        NODETO(ROOT,1) := NIL ;
        GO TO QUIT ;
      END ;
    END ;
REFTAB(PATH+LEVEL-1,LEVEL) := 1000*MOVEFROM(N) + MOVETO(N);
    MAKEMOVE(N,TRUE) ;
    NEW_HASH := HASH;
    IF OLD AND NODE ~= NIL AND NEW_HASH ~= STATES(NODE, T_SIZE)
    THEN BEGIN
      OLD := FALSE ;
COMMENT The node we were pointing to was not a proper successor
        of the previous position. Should only occur deep in the
        tree, and should be infrequent;
      WRITE("DISASTER AVOIDED", LEVEL, NODE, T_HASH, NEW_HASH,
                    STATES(NODE, T_SIZE));
      NODE := NIL ;
    END ;
    IF TEST
    THEN WRITEON(CREDIT(LEVEL), DEBIT(LEVEL-1), "R=", ROOT) ;
    MORE := SHAH OR INCHECK ;
    DYNAMIC := IF INCHECK OR ~EARLY AND (SHAH OR (REWN < 0)) OR
      (SIDE = -1) AND (GARDE OR  (REWN < 0))
                THEN TRUE
                ELSE FALSE ;
    IF (CON(-K,KINGSQ(K)) ~= 0)
    THEN BEGIN
      COMMENT illegal move "impossible" ;
      SCR := EIGHTS*SIDE ;
COMMENT A King is in check and the other side has the move;
      WRITE("IMPOSSIBLE", K, KINGSQ(K), CON(-K,KINGSQ(K)), KINGSQ(-K))
        ;
      TRACER(2,"?PD") ;
      ANOTHER := TRUE ;
      FINAL := -3 ;
      GO TO EXIT ;
```

```
10010     END ;
10011     IF ~OLD OR (NODE = NIL)
10012     THEN IF (ROOT = NIL) OR NODAL
10013          THEN GO TO EXIT ;
10014   IF FINAL >= 1 THEN BEGIN
10015     IF TEST
10016     THEN WRITE("RECLAIM ", SCR, NODE, PARENT(NODE), 1000*LEVEL,
10017       STATES(NODE,2), DEEPER, STATES(NODE,3), STATES(NODE,5));
10018     NODETO(ROOT,IP) := 0;
10019   END;
10020     IF  (OLD OR FINAL=5) AND NODE > NIL AND 1000*LEVEL >= PARENT(NODE)
10021          AND STATES(NODE,2) > DEEPER AND ~INCHECK AND ~AGAIN
10022     THEN BEGIN
10023       COMMENT extension into lower subtree ;
10024       FINAL := 1 ;
10025       ANOTHER := TRUE ;
10026       PP_STATUS := 0 ;
10027       SCR := -STATES(NODE, T_SIZE+2+1) ;
10028       IF TEST
10029       THEN WRITEON("&&&",NODE, SCR, PARENT(NODE), FINAL) ;
10030       GO TO EXIT ;
10031     END ;
10032     INCREASE := 0 ;
10033   IF INCHECK OR GIVINGCH AND EXTENSION OR
10034     (DYNAMIC OR (NUM < 10) AND ~EARLY OR GIVINGCH
10035       OR SIDE < 0 AND (ALPHA < SCR + THRESHOLD OR IP = 1))
10036     THEN IF DEEPER <= 1 AND LEVEL+1 < MAXDEPTH
10037          THEN INCREASE := IF (DEEPER > 0) AND (SIDE = -1) OR
10038                           DEEPER = 0 AND SIDE = 1
10039                           THEN 1
10040                           ELSE 2 ;
10041     IF OLD AND DYNAMIC AND DEEPER = 0 AND INCREASE = 0 AND LEVEL
10042       < MAXDEPTH AND LEVEL > MINDEPTH
10043     THEN INCREASE := 1 ;
10044     IF LEVEL > LENGTH
10045     THEN INCREASE := 0
10046     ELSE IF (INCHECK OR PP_STATUS >= 10 OR M14~=0 OR LATE_END AND
10047       PP_STATUS > 0) AND LEVEL >= MINDEPTH AND INCREASE = 0 AND
10048       (LEVEL +DEEPER < MAXDEPTH OR LEVEL <= 3 AND LEVEL = MAXDEPTH
10049       AND DEEPER = 0) OR EXTENSION AND LEVEL <= MAXDEPTH
10050          THEN INCREASE :=  (IF SIDE = -1
10051                             THEN 2
10052                             ELSE 1) ;
10053     IF TEST AND (INCHECK OR INCREASE > 0)
10054     THEN WRITE("INC",INCREASE, MINDEPTH, MAXDEPTH, DEEP, LEVEL,
10055       DEEPER, OLD, FINAL, INCHECK, LENGTH, PP_STATUS, EXTENSION) ;
10056     IF (LEVEL >= MAXDEPTH) AND LEVEL > DEEP +1 AND (~INCHECK AND
10057       PP_STATUS < 10 AND M14 = 0 AND (~LATE_END OR PP_STATUS = 0)
10058       OR LEVEL > LENGTH AND SIDE = -1)
10059     THEN FINAL := -6
10060     ELSE IF ROOT = NIL
10061          THEN FINAL := -6
10062          ELSE IF EXPAND
10063               THEN GO TO LOOP ;
10064     IF FINAL = -4 OR FINAL = -5
10065     THEN GO TO QUIT ;
10066 EXIT:
10067     IF TERMINATE
10068     THEN GO TO QUIT ;
10069    COMMENT if drawing or rejecting, point to dummy node 0 and examine
10070    next move ;
10071     IF (IDONE < IP) AND ~ EXTENSION
10072     THEN BEGIN
10073       NODETO(ROOT,0) := IP ;
10074       COMMENT NODETO(ROOT,IP) := NIL ;
10075     END ;
10076    COMMENT IF SIDE*MX > -B_SCR THEN MX := -B_SCR ;
10077     IF ((AB(LEVEL) - MX)*SIDE > 0)
10078     THEN AB(LEVEL) := MX ;
10079     IF TEST
```

```
10080     THEN WRITE(REASON(FINAL),"EXPANSION ", AB(LEVEL), EXTENSION,
10081       B_SCR) ;
10082     OLDSCR := LASTSCR ;
10083     PREVSCR := PASTSCR ;
10084     IF PARTIAL AND IP < LIMIT AND SCORE(TRY(IP+1)) < BETA AND IP > 0
10085     THEN BEGIN
10086   COMMENT for some reason the beta bound has increased, so a full
10087           evaluation is not required;
10088       WRITE("EMERGENCY ", IP, SCORE(TRY(IP+1)), BETA) ;
10089       IP := 0 ;
10090       UPPER := EIGHTS ;
10091       PARTIAL := FALSE ;
10092       PART := PLEX := 0 ;
10093       LISTLEGALMOVES(FALSE) ;
10094       HASH := T_HASH;
10095       SAVESTATE(ROOT) ;
10096     END ;
10097 LOOP:
10098     IF IP > 0 AND ~EXTENSION
10099     THEN LIMITS ;
10100     IF TEST AND LEVEL = 1
10101     THEN WRITE("TIMES",IP,MIN, LIMIT, TIME(1) -OLDCPU, M_TIMES(MODE),
10102       MODE, MX,B_SCR) ;
10103   END ;
10104    COMMENT limit of variations ;
10105 QUIT:
10106 END LOOKAHEAD ;
10107
10108 @TITLE,"  EXPAND"
10109 LOGICAL PROCEDURE EXPAND ;
10110 BEGIN
10111   LOGICAL EXP, ABANDONNED;
10112   INTEGER TEMP, I, MV, GAP, NOD, LEV, FROM, TOTO, NODEX, STATE,
10113        CHANGE, BASE_SCR;
10114  LOGICAL ARRAY FULLEVAL(1::MAX_WID);
10115   EXP := FALSE ;
10116   EST := MINTREE(LEVEL) DIV 2 ;
10117   IF (MAXL -POINT < EST) AND ~OLD
10118   THEN BEGIN
10119     MAKE_SPACE(ORIGIN, 1, PATH, 1000, EST) ;
10120     LEV := 2 ;
10121     NOD := ROOTNODE(2) ;
10122     WHILE (MAXL-POINT < EST) AND (LEV < LEVEL)
10123     DO BEGIN
10124       LEV := LEV + 1 ;
10125       FOR I := 1 UNTIL NODETO(NOD,0)
10126       DO IF NODETO(NOD,I) = ROOTNODE(LEV)
10127          THEN MAKE_SPACE(NOD, 1, I, 1000*LEV, EST) ;
10128       IF TEST
10129       THEN WRITE("NO ROOM?", LEV, LEVEL, NOD, MAXL-POINT, EST) ;
10130       NOD := ROOTNODE(LEV) ;
10131     END ;
10132   END ;
10133   IF OLD OR FINAL = 5 OR SIDE = 1 OR INCREASE > 0 OR ((MAXL >= POINT)
10134     AND (FORCED AND (LEVEL > 2))) OR ((MAXL - POINT >= EST) AND
10135     ((IP <= MIN) OR (BEST <= EXPECT+EPSILON) OR LEVEL = 1 AND BEST
10136     < SCR) OR GIVINGCH)
10137   THEN BEGIN
10138     IF (DEEPER > 0 OR INCREASE > 0 OR OLD) AND (~INCHECK OR LEVEL
10139       ~= DEEP OR IP < WIDTH DIV 2 OR TIME(1) - T_TIME < M_TIMES(0))
10140     THEN BEGIN
10141       COMMENT abreviate search if excessive CPU time on this variation,
10142         and not in check ;
10143       IF ~OLD OR FINAL = 5
10144       THEN BEGIN
10145         IF SIDE = 1 AND POINT >= MAXL
10146         THEN NODE := 0
10147         ELSE BEGIN
10148                IF POINT < MAXL
10149                THEN BEGIN
```

```
10150                 POINT := POINT +1;
10151                 NODE := GETNODE(POINT) ;
10152               END ELSE NODE := NIL;
10153             END;
10154       NODETO(NODE,0) := IF FINAL = 5 AND NODETO(NODE,0) >= 0
10155                         THEN 0
10156                         ELSE -1 ;
10157     END ;
10158     IF TEST
10159     THEN WRITEON(NODE) ;
10160     PARENT(NODE) := ROOT + LEVEL*1000 ;
10161     IF (NODETO(ROOT,0) < IP)
10162     THEN NODETO(ROOT,0) := IP ;
10163     NODETO(ROOT,IP) := NODE ;
10164     T_NODES := T_NODES+1 ;
10165     PREVSCR := OLDSCR ;
10166     OLDSCR := SCR ;
10167     BASE_SCR := SCR;
10168     SAVEBACK(LEVEL, TRUE) ;
10169     K := -K ;
10170     LEVEL := LEVEL+1 ;
10171     COMMENT  switchsides. ;
10172     IF PRINCVAR  AND IP > 1 AND ~AGAIN
10173     THEN FULLWIN := FALSE ;
10174     IF ~AGAIN
10175     THEN BEGIN
10176     LOWER := IF B_SCR > ALPHA
10177              THEN B_SCR
10178              ELSE ALPHA ;
10179       HIGHER := IF FULLWIN OR IP = 1
10180                 THEN BETA
10181                 ELSE LOWER +1 ;
10182       L_SCR := HIGHER;
10183     END ;
10184     T_SCR := -TREE(-HIGHER, -LOWER, NEW_HASH, DEEPER+INCREASE-1,
10185       PRINCVAR AND FULLWIN, F_T) ;
10186     COMMENT IF TEST
10187     THEN WRITE("SCORE ", LEVEL, ROOT, STATES(ROOT, T_SIZE+2+IP),
10188     T_SCR, B_SCR, MX, F_T) ;
10189     IF TEST AND PRINCVAR
10190     THEN WRITE("PAB", IP, AGAIN, HIGHER, LOWER, B_SCR, T_SCR,
10191       FULLWIN, PRINCVAR, L_SCR, MX, BEST, EXTENSION) ;
10192     IF EXTENSION
10193     THEN EXTENSION := ANOTHER := FALSE ;
10194     IF T_SCR > B_SCR OR AGAIN AND T_SCR > L_SCR
10195     THEN BEGIN
10196       IF AGAIN
10197       THEN BEGIN
10198         IF CNT > 0
10199         THEN WRITE ("PAB error-loop ", LOWER, T_SCR, HIGHER) ;
10200         CNT := CNT + 1 ;
10201         B_SCR := IF T_SCR > L_SCR
10202                  THEN T_SCR
10203                  ELSE L_SCR ;
10204       END ELSE
10205       BEGIN
10206         CNT := 0 ;
10207         L_SCR := B_SCR ;
10208         B_SCR := T_SCR ;
10209       END ;
10210       IF PRINCVAR AND IP > 1
10211       THEN IF ~AGAIN OR T_SCR > HIGHER OR T_SCR < LOWER
10212            THEN BEGIN
10213              IF LEVEL = 1
10214              THEN WRITE("New candidate PV", FIRSTONE, IP, L_SCR,
10215                B_SCR, T_SCR) ;
10216              IOCONTROL(2) ;
10217              IF FIRSTONE = 0 AND LEVEL = 1
10218              THEN FIRSTONE := IP ;
10219              FULLWIN := AGAIN := TRUE ;
```

```
10220                HIGHER := EIGHTS ;
10221                LOWER := L_SCR -1 ;
10222                IP := IP -1 ;
10223              END ELSE AGAIN := FALSE ;
10224          END ELSE
10225          BEGIN
10226            IF AGAIN
10227            THEN B_SCR := L_SCR ;
10228            IF TEST AND AGAIN AND LEVEL = 1
10229            THEN BEGIN
10230              WRITE("Candidate Rejected", FIRSTONE, IP, L_SCR, B_SCR,
10231                T_SCR) ;
10232              IOCONTROL(2) ;
10233            END ;
10234            AGAIN := FALSE ;
10235          END ;
10236          IF AGAIN
10237          THEN EXP := TRUE
10238          ELSE BEGIN
10239            IF KTEST ~= 0
10240            THEN NODETO(ROOT,IP) := NIL ;
10241            IF KTEST = 2
10242            THEN KTEST := BACK(LEVEL,16) := 0 ;
10243            BETTER := NINES ;
10244            IF IP = 1 OR SIDE*(MX -AB(LEVEL+1)) > 0
10245            THEN MX := BETTER := AB(LEVEL) := AB(LEVEL+1) ;
10246        COMMENT increase width if first move loses;
10247            IF IP = 1 AND DEEPER > 1 AND SIDE < 0 AND SCR -MX > PAWNVALUE
10248            THEN MIN := MIN +1 ;
10249        COMMENT IF TEST
10250        THEN WRITEON(T_SCR,ALPHA,BETA, BETTER, MX,AB(LEVEL+1)) ;
10251            SCORE(TRY(IP)) := STATES(ROOT,T_SIZE+2+IP)
10252              := -SIDE*AB(LEVEL+1) ;
10253  COMMENT IF TEST
10254  THEN WRITE("TEST ", LEVEL, IP, ALPHA, B_SCR, BETA,F_T,BETTER);
10255            FINAL := 0 ; ABANDONNED := FALSE;
10256            IF (B_SCR < BETA) AND (LEVEL > 2 OR IP < 4 OR B_SCR > ALPHA)
10257            THEN EXP := TRUE
10258            ELSE BEGIN
10259              IF B_SCR > ALPHA OR LEVEL > 2 OR IP < 4
10260                  OR B_SCR >= SCORE(TRY(IP+1))
10261              THEN REFUTE(1000*M2+M3)
10262              ELSE BEGIN
10263                ABANDONNED := TRUE;
10264                IF TEST THEN
10265                   WRITE("ABANDONNED", LEVEL, IP, PLY, DEEP, ALPHA,
10266                   B_SCR, BETA, SCORE(TRY(IP+1)));
10267              END;
10268              STATES(NODETO(ROOT,IP),2) := 0 ;
10269              COMMENT unreliable subtree ;
10270              FINAL := -5 ;
10271              COMMENT   reverse prune, already found better line ;
10272              COMMENT   or abandon aspiration search early;
10273              IF TEST
10274              THEN WRITE("  REVE", LEVEL, ALPHA, B_SCR, BETA, T_SCR,
10275                MX,BEST, PRINCVAR, FULLWIN) ;
10276              COMMENT can now distinguish between no more nodes and
10277              no more depth ;
10278     CHANGE := BASE_SCR - T_SCR;
10279     FOR I := NODETO(ROOT,0) +1 UNTIL TRY(WSIZE)
10280     DO SCORE(TRY(I)) := SCORE(TRY(I)) - CHANGE;
10281     HASH := T_HASH;
10282     SAVESTATE(ROOT);
10283            END ;
10284            MV := 0 ;
10285            IF IP > 1 AND (~ABANDONNED OR T_SCR = B_SCR)
10286            THEN IF (FINAL = -5 AND B_SCR > ALPHA OR
10287              EXP AND F_T = 0 AND BETTER ~= NINES AND LEVEL > 1 OR
10288              LEVEL = 1 AND MX > BEST)
10289            THEN MV := 1
```

```
10290           ELSE BEGIN
10291             IF F_T = 0 AND PRINCVAR AND ABS(HIGHER - LOWER) = 1
10292             THEN F_T := 1 ;
10293             TEMP := -SIDE*AB(LEVEL+1) ;
10294             I := IP;
10295             WHILE I > 1 DO BEGIN
10296               I := I -1;
10297               FULLEVAL(I) := STATES(NODETO(ROOT,I),5) = 0;
10298               IF F_T ~= 0 AND FULLEVAL(I)
10299               THEN I := 0 ELSE
10300               GAP := TEMP -STATES(ROOT, T_SIZE+2+I) ;
10301             IF I > 0 THEN
10302               IF GAP < -200 THEN I := 0 ELSE
10303               IF GAP > 0 OR F_T = 0 AND ~FULLEVAL(I)
10304               THEN IF F_T = 0 OR I = 1
10305                    THEN MV := I
10306                    ELSE IF GAP > 200 OR I > 8 OR IP-5 <= I
10307                         THEN MV := I
10308                         ELSE IF IP-5 < MV
10309                              THEN IF IP-5 >= 8
10310                                   THEN MV := 8
10311                                   ELSE MV := IP-5;
10312             END ;
10313           END ;
10314           IF MV > 0
10315           THEN BEGIN
10316             IF TEST
10317             THEN WRITE("SEE", MV, IP, F_T, GAP, ROOT, TEMP,
10318                 NODETO(ROOT,MV), STATES(NODETO(ROOT,MV),5));
10319             IF TEST
10320             THEN FOR I:= 1 UNTIL IP
10321             DO WRITEON(STATES(ROOT,T_SIZE+2+I)) ;
10322             SHIFT(IP,MV,ROOT) ;
10323           END ;
10324           IF  (IP = 1 OR MV = 1)
10325           THEN IF ABS(HIGHER - LOWER) = 1
10326                THEN TYPE := -1
10327                ELSE TYPE := IF F_T = 0
10328                             THEN 0
10329                             ELSE -1 ;
10330           IF IP = 1 OR MV > 0
10331           THEN STATES(ROOT,2) := DEEPER + 1 ;
10332           IF LEVEL = 1 AND MX > BEST
10333           THEN BEGIN
10334             REPLY := PATH ;
10335             BEST := MAXSC(PATH) := MX ;
10336           END ;
10337           IF PARTIAL
10338           THEN IF EXP
10339                THEN BEGIN
10340 COMMENT previous partial evaluation insufficient,
10341          do full evaluation;
10342                IF TEST
10343                THEN WRITE("CUTOFF REVISED",LEVEL, IP, B_SCR, BETA) ;
10344                FROM := MOVEFROM(TRY(1)) ;
10345                TOTO := MOVETO(TRY(1)) ;
10346                PARTIAL := FALSE ;
10347                UPPER := EIGHTS ;
10348                LISTLEGALMOVES(FALSE) ;
10349                TYPE := PLEX := PART := 0 ;
10350                IP := IF FROM = MOVEFROM(TRY(1)) AND
10351                  TOTO = MOVETO(TRY(1))
10352                      THEN 1
10353                      ELSE 0 ;
10354                MAKE_SPACE(ROOT, IP, -1, 1000*LEVEL, MAXTREE) ;
10355                IDONE := IP ;
10356                HASH := T_HASH;
10357                SAVESTATE(ROOT) ;
10358           END ELSE IF TEST
10359                THEN WRITE("SUCCESSFUL SALVAGE", ROOT, IP) ;
```

```
10360 COMMENT "successful salvage" means that the previous partial evaluation
10361         still provides an effective cut-off;
10362         END PRINCVAR_LOOP ;
10363       IF LEVEL =1 AND (AGAIN OR IP = 1 OR T_SCR > L_SCR)
10364       THEN BEGIN
10365         IOCONTROL(2) ;
10366         NODEX := ORIGIN ;
10367         I := IF AGAIN THEN IP +1
10368              ELSE 1;
10369         WHILE NODEX > 0
10370         DO BEGIN
10371           STATE := STATES(NODEX, T_SIZE+2+MAX_WID+I) ;
10372           FROM := ABS(STATE) REM 256 ;
10373           TOTO := (ABS(STATE) REM 65536) DIV 256 ;
10374           WRITEON(BOX(FROM), BOX(TOTO), " ") ;
10375           NODEX := NODETO(NODEX,I) ;
10376           I := 1 ;
10377         END ;
10378         WRITEON(LOWER, T_SCR) ;
10379         IOCONTROL(2) ;
10380       END ;
10381         IF TEST AND LEVEL < 3
10382         THEN WRITE("TYPE", LEVEL, LOWER, HIGHER, ALPHA, BETA, F_T,TYPE,
10383           AGAIN, IP, MV, PRINCVAR) ;
10384       END ELSE FINAL := -6 ;
10385    COMMENT maximum depth ;
10386     END ELSE
10387     BEGIN
10388       IF TEST OR (IP <= 2) AND (LEVEL < 4)
10389       THEN WRITE("PRUNE", LEVEL, IP, FINAL, MAXL, POINT, EST, SIDE,
10390         FORCED, MAXNODES, T_NODES) ;
10391       IF IP > 1 AND LEVEL < MAXDEPTH
10392       THEN SCR := -44444 ;
10393       FINAL := -7 ;
10394       IF FORCED
10395       THEN SMALL := TRUE ;
10396     END ;
10397    COMMENT no more nodes for this variation ;
10398     EXP
10399   END OFEXPAND ;
10400
10401 @TITLE,"MATE_SOLUTION"
10402   PROCEDURE MATE_SOLN ;
10403   BEGIN
10404     LOGICAL PRINT;
10405     INTEGER MVT, MVF, SQ3, NEW_MOVES, VAR, KEEP, SCR, LEV, STATE,
10406       NODEX, FROM, TOTO, ALL;
10407     INTEGER JJ, IN, R, M, J, SQF, SQT, DIR, SQ, L, PC, OFFT, SQ1, D ;
10408   LOGICAL ARRAY AVAIL(1::80);
10409     INTEGER ARRAY REFLINE (1::MAX_PLIES);
10410   PRINT := TEST OR -B_SCR > SEVENS;
10411     VAR := IF LEVEL = 1 THEN REPLY ELSE PATH;
10412   IF PRINT THEN
10413     WRITE("LOSS_SOLN ",COLOR(K), LEVEL,IP,IDONE,VAR, REPLY,LASTSCR,
10414       TARGET, B_SCR, EST, ALPHA, BETA) ;
10415   IF PRINT THEN WRITE("REFUTATIONS ");
10416   FOR LEV := 1 UNTIL MAX_PLIES
10417   DO REFLINE(LEV) := 0;
10418   NODEX := ROOT;
10419   LEV := LEVEL;
10420         WHILE NODEX > 0
10421         DO BEGIN
10422           STATE := STATES(NODEX, T_SIZE+2+MAX_WID+1) ;
10423           FROM := ABS(STATE) REM 256 ;
10424           TOTO := (ABS(STATE) REM 65536) DIV 256 ;
10425           REFLINE(LEV) := 1000*FROM + TOTO;
10426           IF PRINT THEN WRITEON(REFLINE(LEV));
10427           LEV := LEV + 1;
10428           NODEX := NODETO(NODEX,1) ;
10429         END ;
```

```
10430     R := 1 ;
10431     MAKE_SPACE(ROOT, 0, 0, 1000*LEVEL, MAXTREE) ;
10432    ALL := TRY(WSIZE);
10433     IN := TOTAL*K ;
10434     JJ := -IN -K ;
10435     FOR L := 1 UNTIL ALL
10436     DO BEGIN
10437       JJ := JJ +K ;
10438       MOVEFROM(JJ) := MOVEFROM(TRY(L)) ;
10439       MOVETO(JJ) := MOVETO(TRY(L)) ;
10440       REWARD(JJ) := REWARD(TRY(L));
10441       SCORE(JJ) := SCORE(TRY(L));
10442     QUIES(JJ) := QUIES(TRY(L));
10443   COMMENT IF PRINT THEN PRINTLINE(JJ,2);
10444   COMMENT IF PRINT THEN WRITEON(L, JJ, TRY(L));
10445     END ;
10446     R := TRY(R) ;
10447     SCORE(IN) := SCORE(R) ;
10448     REWARD(IN) := REWARD(R) ;
10449     MOVEFROM(IN) := MOVEFROM(R) ;
10450     MOVETO(IN) := MOVETO(R) ;
10451     ALTERNATIVE := TRUE ;
10452     LISTLEGALMOVES (TRUE) ;
10453     ALTERNATIVE := FALSE ;
10454     NUM := NUMBER(K) ;
10455     FOR L := -IN STEP K UNTIL JJ
10456     DO FOR J := NUM STEP -K UNTIL K
10457     DO IF (MOVEFROM(J) = MOVEFROM(L) AND MOVETO(J) = MOVETO(L))
10458        THEN SCORE(J) := BAD ;
10459       FOR J := K STEP K UNTIL NUM
10460       DO AVAIL(ABS(J)) := SCORE(J) ~= BAD;
10461       NEW_MOVES := M := 1;
10462   IF PRINT THEN WRITE("REFUTATIONS ");
10463 IF PRINT THEN
10464 FOR LEV := 1 UNTIL MAX_PLIES
10465 DO WRITEON(REFTAB(PATH+LEVEL-1,LEV));
10466     FOR LEV := LEVEL+1 STEP 2 UNTIL MAX_PLIES
10467     DO BEGIN
10468       R := REFLINE(LEV);
10469     IF R > 0
10470     THEN BEGIN
10471       COMMENT get direction of refuting move SQF - SQT
10472               look for moves which
10473               1. give check or move King.
10474               2. Move pc on SQT, if present.
10475               3. Defend SQT.
10476               4. Attack SQF, if empty.
10477               5. Block attack path, SQF - SQT;
10478       SQF := R DIV 1000 ;
10479       SQT := R REM 1000 ;
10480       OFFT := OFFSET(SQT) ;
10481       DIR := BOTV(EDGE, OFFT-OFFSET(SQF)) ;
10482       COMMENT DIR = 0 means attacker is a Knight;
10483       IF PRINT
10484       THEN WRITE("MATER", SQF, SQT, M, DIR, ROOT, NODETO(ROOT,1)) ;
10485       L := SEC(0,SQT) ;
10486       IF L ~= 0
10487       THEN L := 1 ;
10488       J := NUM ;
10489 IF SQF > WQRSQ THEN
10490       WHILE J ~= 0
10491       DO BEGIN
10492         IF AVAIL(ABS(J))
10493         THEN BEGIN
10494         SQ := SQT + DIR ;
10495           JJ := L ;
10496           PC := ABS(BRD(MOVEFROM(J))) ;
10497           KEEP := 0 ;
10498           MVF := MOVEFROM(J) ;
10499           MVT := MOVETO(J) ;
```

```
10500           COMMENT accept checks or King moves;
10501           IF ABS(REWARD(J)) >= CHECKING OR PC = KING
10502           THEN KEEP := 1
10503           ELSE IF PC >= KNIGHT AND BOTV(PC, OFFT-OFFSET(MVT)) = 1
10504                   AND BOTV(PC, OFFT-OFFSET(MVF)) ~= 1
10505                THEN BEGIN
10506                  COMMENT move attacked piece, unless it is a pawn.
10507                          add a new defender to SQT;
10508                  D := BOTV(EDGE, OFFSET(MVT) -OFFT) ;
10509                  IF PC = KNIGHT
10510                  THEN D := SQT - MVT ;
10511                  IF D ~= 0 AND D ~= BOTV(EDGE, OFFSET(MVF) -OFFT)
10512                  THEN BEGIN
10513                    SQ1 := MVT + D ;
10514                    IF D ~= 0
10515                    THEN WHILE SQ1 ~= SQT AND BRD(SQ1) = 0
10516                    DO SQ1 := SQ1 + D ;
10517                    COMMENT D = 0 means that SQT = MVT;
10518                    IF SQ1 = SQT
10519                    THEN KEEP := 2 ;
10520                  END;
10521           END ;
10522             COMMENT  find pawn defence ;
10523           IF PC = PAWN
10524           THEN IF ABS(MVT -SQT +K*FILE) = 1 OR
10525                   BRD(SQF) = 0 AND ABS(MVT -SQF +K*FILE) = 1
10526           THEN KEEP := 3 ;
10527           IF KEEP=0 AND DIR ~= 0
10528           THEN WHILE KEEP=0 AND (JJ <= 1)
10529                DO BEGIN
10530                  COMMENT block the attack path;
10531                  IF BRD(SQ) ~= 0
10532                  THEN JJ := JJ +1 ;
10533                  IF SQ = MVT
10534                  THEN KEEP := 4
10535                  ELSE SQ := SQ + DIR ;
10536           END ;
10537           COMMENT does piece movement add a hidden defender;
10538           IF KEEP=0 AND BOTV(EDGE, OFFT-OFFSET(MVF)) ~= 0
10539           THEN IF CLEAR(MVF, MVF, SQT)
10540                THEN IF BOTV(PC, OFFT-OFFSET(MVT)) ~= 1
10541           THEN IF HIDDEN(SQT, MVF, SQ3, K, FALSE)
10542                THEN KEEP := 5 ;
10543              IF (KEEP=0 OR SQT = MVF) AND BRD(SQF) = 0
10544                AND BOTV(PC, OFFSET(MVT) -OFFSET(SQF)) = 1
10545              THEN BEGIN
10546                COMMENT attack SQF, so that opponent cannot move there;
10547                D := BOTV(EDGE, OFFSET(MVT) -OFFSET(SQF));
10548                IF D ~= DIR   AND
10549                   D ~= BOTV(EDGE, OFFSET(MVF) -OFFSET(SQF))
10550                THEN BEGIN
10551                  COMMENT not an inline move;
10552                  SQ1 := MVT + D;
10553                  IF D ~= 0
10554                  THEN WHILE BRD(SQ1) = 0 AND SQ1 ~= SQF
10555                       DO SQ1 := SQ1 + D;
10556                  IF PC = KNIGHT
10557                  THEN SQ1 := SQF;
10558                  IF SQ1 = SQF
10559                  THEN IF KEEP > 0
10560                       THEN SCORE(J) := SCORE(J) + 50
10561                       ELSE KEEP := 6;
10562                END;
10563              END;
10564           IF KEEP > 0
10565           THEN BEGIN
10566 IF PRINT THEN
10567 WRITE("SOLN", SQF, SQT, PC, MVF, MVT, D, DIR, SQ1, SQ, M,KEEP);
10568             COMMENT assert TW = MAX_WID;
10569             IF PRINT THEN PRINTLINE(J, 2);
```

```
10570      SCR := 0;
10571      IF KEEP = 1 AND PC = KING
10572      THEN SCR := -50
10573      ELSE IF KEEP = 5 OR KEEP = 2 AND CON(-K,MVT) = 0
10574           THEN SCR := 50;
10575      SCORE(J) := SCORE(J) + SCR;
10576        AVAIL(ABS(J)) := FALSE;
10577        NEW_MOVES := NEW_MOVES + 1;
10578              IF M <= MAX_WID THEN M := M + 1 ;
10579              TRY(M) := J ;
10580              FOR I := M STEP -1 UNTIL 2
10581              DO IF SCORE(J) > SCORE(TRY(I))
10582                 THEN BEGIN
10583                   TRY(I+1) := TRY(I) ;
10584                   TRY(I) := J ;
10585              END ;
10586            END ;
10587          END ;
10588          J := J - K ;
10589        END ;
10590      END ELSE WRITE("No killer at level ", LEV, R) ;
10591      END ;
10592        J := -IN + IP*K;
10593        JJ := NUM;
10594      FOR I := 0,1 DO BEGIN
10595        WHILE M < MAX_WID AND ABS(J) > ABS(IN) -ALL  AND ABS(JJ) < TOTAL
10596        DO BEGIN
10597  COMMENT then the moves not yet searched,
10598          followed by those just searched (if room);
10599          M := M + 1;
10600  COMMENT first the best move
10601          then the injected "potential refuters";
10602          JJ := JJ + K;
10603            IF PRINT THEN PRINTLINE(J, 2);
10604        IF PRINT THEN WRITEON(M, J, JJ);
10605          MOVEFROM(JJ) := MOVEFROM(J);
10606          MOVETO(JJ) := MOVETO(J);
10607          REWARD(JJ) := REWARD(J);
10608          SCORE(JJ) := SCORE(J);
10609          QUIES(JJ) := QUIES(J);
10610          TRY(M) := JJ;
10611          J := J + K;
10612        END;
10613        J := -IN+K;
10614      END;
10615      NUM := M ;
10616      IF NEW_MOVES > 1
10617      THEN BEGIN
10618        TRY(1) := IN ;
10619        JJ := IF NUM > WIDTH
10620              THEN WIDTH
10621              ELSE NUM ;
10622        FOR I := 1 UNTIL JJ
10623        DO BEGIN
10624          IF TEST OR LEVEL = 1
10625          THEN PRINTLINE(TRY(I), 0);
10626          IF I > 1
10627          THEN SCORE(TRY(I)) := SCALE*ONEVAL;
10628        END;
10629        IF PRINT
10630        THEN WRITEON(REPLY,NUM, JJ,K,LEVEL,ONEVAL) ;
10631        LIMIT := TRY(0) := TRY(WSIZE) := JJ ;
10632        REMAKE(KINGSQ(K),KINGSQ(K)) ;
10633        REMAKE(KINGSQ(-K),KINGSQ(-K)) ;
10634        SAVE(ROOT) ;
10635        MX := SIDE*EIGHTS ;
10636        IDONE := NODETO(ROOT,0) := 1 ;
10637        NODE := NODETO(ROOT,1) ;
10638        IP := 0 ;   COMMENT try also IP := 1   why repeat PV?;
10639        IF LEVEL = 1
```

```
10640       THEN REPLY := 1 ;
10641       MATER := FALSE ;
10642     END ELSE
10643     BEGIN
10644       WRITEON("No suitable refutation ") ;
10645       RESTORE(ROOT) ;
10646     END ;
10647     NUM := ABS(NUMBER(K)) ;
10648   END OFMATE_SOLN ;
10649
10650 @TITLE,"TERMINATE"
10651   LOGICAL PROCEDURE TERMINATE ;
10652   BEGIN
10653     INTEGER EST_SCR, DEST, LOW_SCR;
10654     INTEGER FROM, TOTO, VAR_TIME, INDEX, FIELD;
10655   LOGICAL PV;
10656   LOWER := IF B_SCR > ALPHA THEN B_SCR ELSE ALPHA;
10657     PV := LOWER < SCR AND SCR < BETA;
10658     FROM := M2 ;
10659     TOTO := M3 ;
10660     EST_SCR := SCR - THRESHOLD ;
10661     LOW_SCR := SCORE(TRY(IP+1)) ;
10662     IF IP < LIMIT AND LOW_SCR > EST_SCR
10663     THEN EST_SCR := LOW_SCR ;
10664     IF EST_SCR > BETA
10665     THEN EST_SCR := BETA ;
10666     IF  PARTIAL AND LOW_SCR = BAD
10667     THEN LOW_SCR := EST_SCR ;
10668     DYNAMIC := FALSE ;
10669     IF ABS(FINAL-2) ~= 1
10670     THEN BEGIN
10671       IF DEEPER > 0  AND (DEEPER REM 2) ~= 0
10672       THEN WRITE("QUESTION DEPTH", DEEPER, MINDEPTH, LEVEL, MAXDEPTH,
10673         NODE,OLD,FINAL) ;
10674       IF STATES(ROOT,2) ~= 0 AND (OLD OR NODETO(ROOT,0) >= IP)
10675       THEN SCR := SCR + LEVEL
10676       ELSE IF PP_STATUS > 0
10677            THEN IF (CON(-K,M3) ~= 0 OR PP_STATUS > 3) AND ABS(BRD(M3))
10678              = PAWN
10679                 THEN BEGIN
10680                   DEST := BACK_ROW(-K, COLS(M3)) ;
10681                   DYNAMIC := PP_STATUS > 3 AND CON(-K,M3) ~= 0 ;
10682                   IF ~DYNAMIC
10683                   THEN SCR := SCR - SCALE*(PP_STATUS
10684                     DIV ( IF BOTV(KING, OFFSET(KINGSQ(-K)) -OFFSET(DEST)
10685                     ) <= BOTV(KING, OFFSET(M3) -OFFSET(DEST)) +1
10686                              THEN IF K*CON(0,M3) > 0
10687                                   THEN 2
10688                                   ELSE 1
10689                                   ELSE IF CON(-K,M3) = 0
10690                    THEN IF CON(-K,M3+K*FILE) ~= 0
10691                         THEN 1
10692                         ELSE IF CON(K,M3) = 0
10693                              THEN 2
10694                              ELSE 4
10695                              ELSE IF K*SEC(0,M3) < 0
10696                                   THEN 1
10697                                   ELSE 2)) ;
10698                   IF TEST
10699                   THEN WRITEON("PP", PP_STATUS, SCR, DEST, BOTV(KING,
10700                     OFFSET(KINGSQ(-K))-OFFSET(DEST))) ;
10701                   IF SCR < BETA AND (PARTIAL OR SCR < EST_SCR)
10702                   THEN ANOTHER := TRUE ;
10703       END ;
10704     END ;
10705         IF TEST
10706         THEN WRITE("PV    ", SCR, LOWER,BETA,LEVEL,MAXDEPTH,MAXLEV,
10707       EXTENSION, DYNAMIC, LENGTH, DEEP);
10708     IF  FINAL = -6 AND (LEVEL <= MAXDEPTH OR LEVEL <= MAXLEV AND
10709       MAXDEPTH >= 5)
```

```
10710     THEN BEGIN COMMENT room for capture search;
10711       TOT := TOT +1 ;
10712 IF ~DYNAMIC THEN
10713       IF SCR > LOWER OR REWN < 0 AND ABS(CON(-K,KINGSQ(-K))) <=1
10714          OR REWN <= -CHECKING
10715          OR PP_STATUS ~= 0 OR LATE_END AND REWN ~= 0
10716       THEN DYNAMIC := (REWN < 0 OR LATE_END AND REWN ~= CHECKING
10717              OR INCHECK AND REWN ~= 0)
10718         AND LEVEL <= LENGTH  OR
10719       PV AND (LEVEL > LENGTH OR LEVEL = DEEP);
10720         IF (TEST OR EXTENSION AND IP > 1) AND PV
10721         THEN WRITE("PVPV  ", SCR, LOWER,BETA, PRINCVAR, REWN,
10722               PP_STATUS, LEVEL, DEEP, SEEN, EXTENSION, IP);
10723 IF EXTENSION  THEN EXTENSION := FALSE ELSE
10724       IF DYNAMIC AND SCR < SEVENS AND
10725         (ABS(REWN) >= CHECKING OR REWN < 0 OR
10726             REWN > 0 AND SCR > LOWER OR
10727           SKILL(0|8) = "   SPEED" OR T_NODES <= PATH*MAXNODES)
10728         AND (LEVEL = DEEP AND ~SEEN OR PV
10729          AND PRINCVAR)
10730            AND SCR > LOWER - THRESHOLD
10731     THEN BEGIN
10732       VAR_TIME := TIME(1);
10733       IF TEST THEN WRITEON(VAR_TIME, T_TIME, M_TIMES(0));
10734    IF PRINCVAR AND SCR > BEST OR VAR_TIME -T_TIME < M_TIMES(0)
10735       THEN BEGIN
10736         COMMENT  note CPU limitation on excessive variation ;
10737         DYN := DYN + 1 ;
10738         IF TEST
10739         THEN WRITEON("FLUID ");
10740         SAVEBACK(LEVEL, TRUE) ;
10741         SAVESTATE(0) ;
10742         EST := SCR + THRESHOLD ;
10743         SCR := -BUILD_SACR_TREE(0, 2, EST_SCR, -BETA,-LOWER,IDONE) ;
10744   IF ~QUIES(TRY(IP)) THEN BEGIN
10745     INDEX := T_SIZE+2+MAX_WID+IP;
10746     FIELD := STATES(ROOT, INDEX);
10747     FIELD := IF FIELD < 0
10748              THEN -(ABS(FIELD) + 65536) ELSE FIELD + 65536;
10749     STATES(ROOT,INDEX) := FIELD;
10750   END;
10751       QUIES(TRY(IP)) := TRUE;
10752         IF ~SHAH
10753         THEN IF ABS(SCR) = FIVES
10754              THEN SCR := EST - THRESHOLD
10755              ELSE IF (BETA > SEVENS OR SCR > BETA)
10756                   THEN BEGIN
10757                     EXTENSION := ANOTHER := TRUE ;
10758                     IF IP = 1
10759                     THEN B_SCR := -EIGHTS ;
10760                     IP := IP -1 ;
10761         END ELSE IF SCR > EST
10762                  THEN SCR := EST ;
10763         IF SCR < BETA AND SCR < LOW_SCR
10764         THEN IF PARTIAL
10765              THEN ANOTHER := TRUE
10766              ELSE IF -SIDE*LOW_SCR > LOWER
10767                   THEN ANOTHER := TRUE ;
10768         RESTOREBACK(LEVEL, TRUE) ;
10769         IF SCR > LOWER
10770         THEN REFUTE(1000*M2 +M3) ;
10771       END ;
10772    END;
10773     END ;
10774     RESTORESTATE(ROOT) ;
10775     RESTOREBACK(LEVEL, FALSE) ;
10776     TAKEBACKMOVE(TRUE) ;
10777     IF ~EXTENSION
10778     THEN BEGIN
10779       EST := IF FINAL = 3
```

```
10780              THEN IF ENDGAME AND ~DRAWS(K) AND DRAWS(-K)
10781                   THEN SIDE*3*DRAW
10782                   ELSE IF ENDGAME AND ~DRAWS(-K) AND DRAWS(K)
10783                        THEN -SIDE*3*DRAW
10784                        ELSE -SIDE*(DRAW -(IF EXPECT > 0
10785                                           THEN LEVEL
10786                                           ELSE -LEVEL))
10787              ELSE IF LATE_END AND SCR < DRAW AND MEN(K) <= 2
10788                        AND DRAWS(K) AND DRAWS(-K)
10789                   THEN -SIDE*(DRAW-LEVEL)
10790                   ELSE IF FINAL > 0
10791                        THEN SCR
10792                        ELSE IF ANOTHER
10793                             THEN -SIDE*SCR
10794                             ELSE IF SIDE = -1
10795               THEN SCR - LEVEL
10796               ELSE IF SCR > SEVENS
10797                    THEN SCR - LEVEL
10798                    ELSE IF IP = 1 OR FINAL >= -2 OR DEEPER = 0
10799                            AND FINAL > -7
10800                         THEN -(LASTSCR + (PASTSCR -SCR) DIV 2)
10801                         ELSE -AB(LEVEL) ;
10802                 COMMENT if even ply termination, estimate score;
10803       IF TEST
10804       THEN WRITEON(ROOT, IP, STATES(ROOT, T_SIZE+2+IP), EST) ;
10805       IF ANOTHER AND PARTIAL    AND SCR < BETA
10806       THEN BEGIN
10807         PARTIAL := FALSE ;
10808         UPPER := EIGHTS ;
10809         PLEX := PART := 0 ;
10810         LISTLEGALMOVES(FALSE) ;
10811         LOW_SCR := SCORE(TRY(IP+1)) ;
10812         IP := IF FROM = MOVEFROM(TRY(1)) AND TOTO = MOVETO(TRY(1))
10813                       AND IP = 1
10814               THEN IP
10815               ELSE 0 ;
10816 COMMENT We are at a terminal node and need a full evaluation afterall;
10817         IF TEST
10818         THEN WRITE("NO CUTOFF", LEVEL, IP, SCR, BETA, LOW_SCR) ;
10819         MAKE_SPACE(ROOT, IP, -1, 1000*LEVEL, MAXTREE) ;
10820         HASH := T_HASH;
10821         SAVESTATE(ROOT) ;
10822       END ;
10823       IF IP > 0 AND FINAL ~= -7 AND FINAL ~= 1
10824       THEN STATES(ROOT, T_SIZE+2+IP) := -SIDE*EST ;
10825     IF IP > IDONE THEN
10826       NODETO(ROOT,0) := IF PARTIAL
10827                         THEN IP-1
10828                         ELSE IP ;
10829       IF ~PARTIAL
10830       THEN STATES(ROOT,2) := 1 ;
10831       IF IP > 0
10832       THEN SCORE(TRY(IP)) := EST ;
10833       EST_SCR := -SIDE*EST ;
10834    COMMENT save incase salvaged ;
10835       GOOD := -SIDE*MX < EST ;
10836       IF GOOD
10837       THEN MX := EST_SCR ;
10838       IF TEST
10839       THEN WRITE("  EXIT",LEVEL,IP,FINAL,SIDE,LIMIT,ANOTHER,DYNAMIC,
10840         MEN(1), MEN(-1), MX, AB(LEVEL), SCR, BEST, LOWER, LOW_SCR) ;
10841       IF GOOD AND (IP > 1 OR MATER) AND (FINAL ~= -7 OR SCR ~= -44444)
10842       THEN BEGIN
10843         PLEX := 0 ;
10844         IF B_SCR < EST  OR IP = 1
10845         THEN B_SCR := EST ;
10846         IF IP > 1 AND (LEVEL > 1 OR MX > BEST)
10847         THEN BEGIN
10848           NODETO(ROOT,0) := IP ;
10849           SCORE(TRY(IP)) := -SIDE*MX ;
```

```
10850           SHIFT(IP, 1, ROOT) ;
10851           IF LEVEL = 1 AND IP > 0
10852           THEN BEGIN
10853             REPLY := 1 ;
10854             BEST := MAXSC(PATH) := MX ;
10855           END ;
10856         END ;
10857         IF TEST
10858         THEN WRITEON("#",B_SCR, BETA) ;
10859       END ;
10860       IF ALPHA < EST AND EST < BETA
10861       THEN ALPHA := EST ;
10862       IF ANOTHER
10863       THEN IF B_SCR > BETA OR B_SCR < LOWER AND (~DYNAMIC)
10864            THEN ANOTHER := FALSE ;
10865       IF LEVEL = MINDEPTH AND MINDEPTH < MAXDEPTH
10866       THEN ANOTHER := TRUE ;
10867     END ;
10868     FINISH := TRUE ;
10869     TYPE := IF PARTIAL
10870             THEN 1
10871             ELSE 0 ;
10872     IF ~ANOTHER AND IP < TRY(WSIZE)
10873     THEN IF SCR < BETA AND SCR < LOW_SCR
10874          THEN ANOTHER := TRUE ;
10875     REFTAB(PATH+LEVEL-2,LEVEL) := REFTAB(PATH+LEVEL-1,LEVEL);
10876 IF LEVEL < MAX_PLIES
10877 THEN REFTAB(PATH+LEVEL-2,LEVEL+1) := -1;
10878     ~ANOTHER  OR IP >= TRY(WSIZE)
10879   END OF_TERMINATE ;
10880
10881 @TITLE "LIMITS"
10882   PROCEDURE LIMITS ;
10883   BEGIN
10884     COMMENT auto. expansion for non-quiescent positions, or at
10885       level 1 ;
10886     IF (KTEST = 4) OR (MX*SIDE < -SEVENS)
10887     THEN SHAH := DYNAMIC := FALSE ;
10888     FINAL := 0 ;
10889     IF (MX - (IF SIDE < 0
10890               THEN SCR
10891               ELSE OLDSCR))*SIDE > 1500 AND (LEVEL <= 5 OR MIN < 3)
10892     THEN ANOTHER := TRUE ;
10893     IF SHAH AND (IP > 1)  OR ANOTHER AND IP >= MIN
10894     THEN MIN := MIN+1 ;
10895     EST := IF (PATH = 1)
10896            THEN MAXSC(1)
10897            ELSE BEST ;
10898     IF (EST -EXPECT)*SIDE > 0
10899     THEN EST := EXPECT ;
10900     IF TEST AND (ABS(MX) > SEVENS)
10901     THEN WRITE("MATE",IP,LEVEL,POINT,T_NODES, MX, EST,BEST, B_SCR,
10902       OLD, MAXL, GETNODE(POINT)) ;
10903     IF ~ANOTHER AND IP = LIMIT
10904     THEN IF (ABS(MX-EST) > LOST OR (PATH > 1) AND ABS(BEST-EST)
10905       > LOST) AND (IP < 3*VARS(LEVEL,LENGTH))
10906          THEN ANOTHER := IF (MX-EST)*SIDE > 0 AND ((BEST-EST)*SIDE
10907            > 0 OR (AB(LEVEL) -EST)*SIDE > LOST) AND (T_NODES -
10908            INIT_T_NODES < MINTREE(LEVEL) OR IP < MIN)
10909                          THEN TRUE
10910                          ELSE FALSE ;
10911     IF SHAH AND (ABS(REWN) >= TWOCHECK) OR (MX*SIDE > SEVENS)
10912     THEN ANOTHER := TRUE ;
10913     IF IP = LIMIT AND IP < WIDTH AND IP < TRY(WSIZE) AND (ANOTHER
10914       OR IP < MIN OR ALPHA < B_SCR AND B_SCR < BETA AND -SIDE*B_SCR
10915       < SCORE(TRY(IP+1)) OR (LEVEL = 1) AND (LIMIT < IDONE) AND
10916       (B_SCR -MAXSC(IP+1) < THRESHOLD))
10917     THEN LIMIT := LIMIT + 1 ;
10918     FORCED := ANOTHER OR (IP < IDONE) ;
10919     EST := (MX - EST)*SIDE ;
```

```
10920     IF TEST AND ((LEVEL < 6) OR ANOTHER OR DYNAMIC)
10921     THEN WRITE("  EXPAN",LEVEL,DYNAMIC,FORCED,IP,MIN,LIMIT, SIDE,
10922       OLDSCR, ABS(MX-EST), ABS(BEST-EST), PATH, ANOTHER, EXPECT,
10923       EST, MX, TARGET, ALPHA, B_SCR, BETA, BEST, T_NODES-INIT_T_NODES,
10924       MAXL - POINT,SCR,LOST) ;
10925     LARGE := IF BEST > MX
10926              THEN BEST
10927              ELSE MX ;
10928     CONDITION := IF ~ANOTHER AND SIDE*MX < -SEVENS
10929                  THEN 0
10930                  ELSE IF FORCED
10931                       THEN 1
10932                       ELSE IF DYNAMIC AND IP < MIN AND LEVEL < 3
10933                            THEN 2
10934                            ELSE IF EARLY AND SIDE*(LARGE-EXPECT)
10935                              > -THRESHOLD
10936                                 THEN 3
10937                                 ELSE IF IP < MIN AND (REVERSE OR
10938                                   EST > -THRESHOLD)
10939                   THEN 4
10940                   ELSE IF EST > THRESHOLD  AND LEVEL <= 5
10941                        THEN 5
10942                        ELSE IF (IF SIDE < 0
10943                                 THEN SCR-MX
10944                                 ELSE MX-OLDSCR) > LOST
10945 THEN 6
10946 ELSE IF PATH = 1 AND IP < MIN
10947      THEN 7
10948      ELSE IF SIDE = -1 AND (TOURNAMENT OR SKILL(0|1) ~= " ") AND
10949        LARGE < EXPECT AND LARGE < MAXSC(PATH) + THRESHOLD
10950           THEN 8
10951           ELSE IF LEVEL = 2 AND PATH > 1 AND MX > BEST
10952                THEN 9
10953                ELSE IF LEVEL <= 2 AND SIDE*(LARGE-EXPECT) > -EPSILON
10954                     THEN 10
10955                     ELSE IF MX < 0 AND IP < MIN
10956                          THEN 11
10957                          ELSE IF LATE_END AND IP < MIN AND LEVEL
10958                            = 1 AND LARGE - EXPECT < PAWNVALUE
10959                               THEN 12
10960                               ELSE IF EST > EPSILON AND LEVEL <= 3
10961                                    THEN 13
10962                                    ELSE 0 ;
10963     IF TEST
10964     THEN WRITEON(CONDITION, TARGET, TYPE, STATES(ROOT,2)) ;
10965     IF ~AGAIN AND IP < LIMIT
10966     THEN IF LEVEL > 1 AND (B_SCR > SEVENS OR CONDITION = 0 AND IP > MIN)
10967            OR LEVEL <= 1 AND (IP >= WIDTH DIV 2 AND TIME(1) -OLDCPU
10968            > M_TIMES(MODE) OR CONDITION = 0 AND IP >= MIN AND B_SCR
10969            > SCORE(TRY(IP+1)) - 2*PAWNVALUE OR IP > 1 AND TIME(1) -
10970            OLDCPU > 2*M_TIMES(MODE))
10971          THEN LIMIT := IP
10972          ELSE IF ~FORCED AND ~MORE AND (LEVEL > 1 OR IP >= MIN
10973                 AND BEST > EXPECT)
10974               THEN IF SIDE = -1 AND (T_NODES-INIT_T_NODES
10975                      > 3*MINTREE(LEVEL) OR MAXL-POINT < MINTREE(LEVEL)
10976                      AND IP >= MIN AND (LEVEL > 3 OR CONDITION ~= 10))
10977                      OR SIDE = 1 AND LENGTH = DEEP AND IP > WIDTH DIV 2
10978                      AND T_NODES -INIT_T_NODES > 3*MINTREE(LEVEL)
10979                    THEN BEGIN
10980                      FINAL := -7 ;
10981                      LIMIT := IP ;
10982                      IF TEST
10983                      THEN WRITE("NODES", T_NODES, MINTREE(LEVEL),
10984                        MAXL-POINT, SIDE) ;
10985     END ;
10986 COMMENT only continue if sufficient nodes ;
10987     IF (LEVEL = 1) AND ~AGAIN
10988     THEN BEGIN
10989       IF PATH < LIMIT AND MX > -SEVENS
```

```
10990       THEN EXPECT := (PATH*EXPECT + MX) DIV (PATH+1) ;
10991       MAXSC(PATH) := MX ;
10992       IF (MX > BEST)
10993       THEN BEGIN
10994         BEST := MX ;
10995         REPLY := PATH ;
10996       END ;
10997     END ;
10998     EST := IF SIDE = 1
10999            THEN - B_SCR - LASTSCR
11000            ELSE TARGET - B_SCR ;
11001     IF ALPHA < B_SCR AND B_SCR < BETA
11002        AND -SIDE*B_SCR < KNIGHTVALUE
11003     THEN IF IP = LIMIT AND (-B_SCR > SEVENS OR EST > (IF ENDGAME
11004                              THEN 1200
11005                              ELSE 2400) AND LEVEL <= 5)
11006             OR LEVEL <= 2 AND IP >= 4 AND
11007                  (PLY = DEEP OR PLY >= 5) AND
11008                  EST > (IF ENDGAME THEN 900
11009                                    ELSE 1800)
11010          THEN IF MATER
11011               THEN MATE_SOLN
11012               ELSE WRITE(COLOR(K), IF ABS(MX) < SEVENS
11013                                    THEN " LOSES"
11014                                    ELSE " MATED", " IN", LEVEL, EST,
11015                                      TARGET, B_SCR, MX) ;
11016     IF EXTENSION AND IP ~= 0
11017     THEN EXTENSION := FALSE ;
11018   END OFLIMITS ;
11019
11020 @TITLE,"TREE"
11021   ROOT := ROOTNODE(LEVEL) := NODE ;
11022   FINISH := FALSE ;
11023   SIDE := -SIDE ;
11024   MATER := TRUE ;
11025   IDONE := NODETO(ROOT,0) ;
11026   AGAIN := FALSE ;
11027   MX := AB(LEVEL) := SIDE*EIGHTS ;
11028   LASTSCR := OLDSCR ;
11029   PASTSCR := PREVSCR ;
11030   A_SCR := ALPHA;
11031   OLD := FALSE ;
11032   PARTIAL := TRUE;
11033   INCREASE := TOT_S := 0 ;
11034   INIT_T_NODES := T_NODES ;
11035   IF ROOT < 0 OR ROOT = 0 AND IDONE >= 0  OR ROOT > MAXTREE
11036   THEN WRITE("TREE TROUBLE",LEVEL, ROOT, IDONE, NUM, NUMBER(K))
11037   ELSE
11038   IF IDONE >= 0 AND ROOT > 0
11039   THEN BEGIN
11040     RESTORESTATE(ROOT) ;
11041     IF HASH ~= T_HASH
11042     THEN BEGIN WRITE("QUERY HASH", HASH, T_HASH, ROOT);
11043       TRACER(1, "?P");
11044       B_SCR := SIDE*EIGHTS;
11045       FINAL := -3;
11046       GOTO QUIT;
11047     END;
11048     IF ~STARTTHISPROBLEM(TRUE)
11049     THEN BEGIN
11050       WRITE("RESTART ERROR", ROOT, IDONE, LEVEL) ;
11051       PRINTSHORTBRD(-1) ;
11052     END ;
11053     OLD := TRUE ;
11054     IF IDONE <= 0 THEN IDONE := NODETO(ROOT,0) := 1;
11055     PARTIAL := PART = 1 ;
11056   A_SCR := SCORE(TRY(1)) ;
11057     INCHECK := CON(-K,KINGSQ(K)) ~= 0 ;
11058     IF LEVEL >= MAXDEPTH
11059     THEN STATES(ROOT,2) := 1 ;
```

```
11060      IF TEST AND PARTIAL
11061      THEN WRITE("PART", PARTIAL, PART, A_SCR, BETA, NUM, INCHECK,
11062        STATES(ROOT,5), IDONE, ROOT) ;
11063    END;
11064    IF PARTIAL
11065    THEN IF A_SCR <= BETA AND PART = 1 OR ~OLD
11066         THEN BEGIN
11067           IF OLD THEN
11068             MAKE_SPACE(ROOT, 0, -1, 1000*LEVEL, MAXTREE) ;
11069        COMMENT regain subtree from a partial node;
11070           PARTIAL := DEEPER = 0 AND LEVEL > 2 ;
11071           UPPER := IF PARTIAL
11072                    THEN BETA + LEVEL
11073                    ELSE EIGHTS ;
11074           PARTIAL := UPPER ~= EIGHTS ;
11075           LISTLEGALMOVES (FALSE) ;
11076           IF UPPER = EIGHTS
11077           THEN PARTIAL := FALSE ;
11078           OLD := FALSE ;
11079           PART := IF PARTIAL
11080                   THEN 1
11081                   ELSE 0 ;
11082     COMMENT array try already filled ;
11083           NODETO(ROOT,0) := IDONE := 0 ;
11084           CAPT_T := 0 ;
11085           HASH := T_HASH ;
11086           SAVESTATE(ROOT) ;
11087    END ;
11088    EST := EXCESS := 0 ; MIN := 1;
11089     COMMENT no injection at max. depth;
11090    IF ~PRINCVAR AND LEVEL < DEEP AND (~OLD OR LEVEL <= 3)
11091    THEN BEGIN
11092      LX := TRY(WSIZE) ;
11093      FOR I := 1 UNTIL LX
11094      DO REFSAVE(I) := 1000*MOVEFROM(TRY(I)) + MOVETO(TRY(I)) ;
11095      IF LX > 1 THEN
11096      FOR I := 1 UNTIL REFUT(LEVEL,0)
11097      DO BEGIN
11098           IF TEST AND LEVEL < 3 THEN BEGIN
11099             IF I = 1
11100             THEN WRITE("CANDIDATES ") ;
11101             WRITEON(REFUT(LEVEL,I), REFCNT(LEVEL,I)) ;
11102           END;
11103           TMP := EST - REFCNT(LEVEL,I) ;
11104           IF TMP <= 0
11105           THEN FOR J := 1 UNTIL LX
11106           DO IF REFSAVE(J) = REFUT(LEVEL,I)
11107                  AND (TMP < 0 OR J < MIN)
11108              THEN BEGIN
11109                EST := REFCNT(LEVEL,I) ;
11110                EXCESS := ABS(TMP);
11111                MIN := J ;
11112           END ;
11113      END ;
11114    END ;
11115    IF TEST AND EST > 0 AND DEEPER > 0 AND LEVEL = 2
11116    THEN BEGIN
11117      TRACER(1,"?D") ;
11118      WRITE("INJ ", REFSAVE(MIN), EST, IDONE, LENGTH) ;
11119    END ;
11120    IF MIN > 1 AND SCORE(TRY(MIN)) ~= NOSCR
11121    THEN BEGIN
11122      LX := IF MIN > WIDTH OR ABS(REWARD(TRY(1))) >= 3 AND EXCESS < 4
11123                OR SCORE(TRY(1)) - SCORE(TRY(MIN)) > 900
11124            THEN IF SCORE(TRY(2)) - SCORE(TRY(MIN)) > 900
11125                 THEN 3
11126                 ELSE 2
11127            ELSE 1 ;
11128  COMMENT Have just dynamically reordered to consider an effective
11129          killer move sooner;
```

```
11130      IF TEST
11131      THEN WRITE("INJECT ", LEVEL, EST, MIN, LX, NODETO(ROOT,0), LENGTH) ;
11132      IF LEVEL > 2
11133      THEN INCREASE := 1 ;
11134      IF IDONE < MIN AND IDONE >= LX
11135      THEN BEGIN
11136        NODETO(ROOT,MIN) := NIL ;
11137        IDONE := IDONE +1;
11138      END;
11139      SHIFT(MIN, LX, ROOT) ;
11140      IF MIN > TRY(0)
11141      THEN TRY(0) := TRY(0) + 1 ;
11142      HASH := T_HASH;
11143      SAVESTATE(ROOT) ;
11144    END ;
11145    IF ~OLD
11146    THEN STATES(ROOT,2) := 0 ;
11147    TYPE := IF PARTIAL
11148            THEN 1
11149            ELSE 0 ;
11150    SAVEBACK(LEVEL, FALSE) ;
11151    IF (NUM = 0) OR TRY(WSIZE) = 0
11152    THEN BEGIN
11153      COMMENT   rare mate or stalemate condition ;
11154      B_SCR := -(NINES -LEVEL+1) ;
11155      IF ~INCHECK
11156      THEN B_SCR := -SIDE*DRAW ;
11157      FINAL := IF INCHECK
11158               THEN 4
11159               ELSE 2 ;
11160      KTEST := FINAL ;
11161      MX := -SIDE*B_SCR ;
11162      IP := 0 ;
11163      TOT := TOT + 1 ;
11164      IF TEST
11165      THEN WRITE("(STALE)MATE", LEVEL, B_SCR, FINAL, KTEST, ROOT) ;
11166      NUM := 0 ;
11167      IF LEVEL = 1
11168      THEN GO TO QUIT ;
11169      POINT := POINT -1;
11170      NODE := ROOTNODE(LEVEL-1) ;
11171      NODETO(NODE,NODETO(NODE,0)) := NIL ;
11172      NODETO(ROOT,0) := -1 ;
11173      GO TO QUIT ;
11174    END ;
11175     COMMENT IF MONITOR THEN TRACER(1, "?D");
11176    IF LEVEL = 1
11177    THEN TARGET := EXPECTATION
11178    ELSE TARGET := IF SIDE = 1
11179                   THEN  + SCALE*ONEVAL
11180                   ELSE SCORE(TRY(1)) ;
11181    IF (FINAL ~= 5)
11182    THEN FINAL := 0 ;
11183    LIMIT := TRY(WSIZE) ;
11184    SCR := BETA;
11185    FOR I := 1 UNTIL (IF LIMIT > 8 THEN 8 ELSE LIMIT)
11186    DO IF SCR > SCORE(TRY(I))
11187       THEN SCR := SCORE(TRY(I));
11188    B_SCR := SCORE(TRY(1)) - PAWNVALUE - PAWNVALUE;
11189    IF SCR < B_SCR
11190    THEN SCR := B_SCR ;
11191    IF PRINCVAR AND SACR_ANAL AND CAPTURE_TREE AND
11192     CAPT_T = 0 AND TRY(WSIZE) > 1 AND (LEVEL < LENGTH)
11193      AND LEVEL <= 5
11194    THEN SCR := BUILD_SACR_TREE(ROOT, MAXPLY+1, SCR,
11195                     ALPHA, BETA, IDONE)
11196    ELSE DISCARD := FALSE ;
11197    IF (LEVEL = 1)
11198    THEN FIRSTENTRY(ROOT)
11199    ELSE IF (LEVEL = 2)
```

```
11200         THEN LIMIT := IF ~OPENING AND ~ENDGAME AND TRY(0) > WIDTH DIV 2
11201                       THEN TRY(0)
11202                       ELSE LIMIT
11203         ELSE LIMIT := VARS(LEVEL, LENGTH) ;
11204      SAVE_WIDTH := WIDTH;
11205 IF PRINCVAR AND LEVEL = 2 AND PATH > 1 AND LENGTH = 5
11206      AND WIDTH < MAX_WID
11207 THEN LIMIT := WIDTH := 12;
11208 COMMENT width may have been narrowed at previous iteration;
11209 IF LEVEL = 2 AND PRINCVAR AND PATH > 1 AND LENGTH = 5
11210 THEN WRITE("ITER WIDE", LIMIT, TRY(WSIZE), WIDTH);
11211   IF LIMIT = 0 OR LIMIT > TRY(WSIZE)
11212   THEN LIMIT := TRY(WSIZE) ;
11213   IF IDONE > WIDTH
11214   THEN BEGIN
11215     FOR I := IDONE STEP -1 UNTIL WIDTH+1
11216     DO IF NODETO(ROOT,I) > 0 THEN BEGIN
11217        MAKE_SPACE(NODETO(ROOT,I), 0, -1, 1000*LEVEL, MAXTREE);
11218       COLLECT(NODETO(ROOT,I), ROOT);
11219       IF MONITOR
11220       THEN WRITE("SPACE ", ROOT, I, NODETO(ROOT,0), NODETO(ROOT,I));
11221       NODETO(ROOT,I) := -1;
11222     END ;
11223    COMMENT this just leaves the main line inplace;
11224     NODETO(ROOT,0) := IDONE := WIDTH;
11225   END ;
11226   IF (IDONE > LIMIT)  AND IDONE <= WIDTH
11227   THEN LIMIT :=  IDONE ;
11228   IF LEVEL = 1 AND OLDSCR = FIVES
11229   THEN LASTSCR :=  - SCALE*ONEVAL ;
11230   IF (LEVEL < 3)
11231   THEN IF (PREVSCR = FIVES)
11232        THEN PASTSCR :=  + SCALE*ONEVAL ;
11233   IF LIMIT > WIDTH
11234   THEN LIMIT := WIDTH ;
11235   IF LEVEL > 2 AND LIMIT = 19
11236   THEN LIMIT := 9 ;
11237   MIN := IF LEVEL > 4
11238          THEN 2
11239          ELSE IF LEVEL > 2
11240               THEN IF LATE_END OR ~QUICK
11241                    THEN 3
11242                    ELSE 2
11243               ELSE IF LATE_END OR ~QUICK OR EXAMINE
11244                    THEN LIMIT
11245                    ELSE IF LEVEL > 1
11246                         THEN LIMIT DIV 2
11247                         ELSE WIDTH DIV 2 ;
11248   MIN := MIN + INCREASE ;
11249   COMMENT for move injection ;
11250   REVERSE := FORCED := FALSE ;
11251   LOOKAHEAD ;
11252     WIDTH := SAVE_WIDTH;
11253 QUIT:
11254   STATES(ROOT,6) := B_SCR ;
11255   IF (IP > TRY(0))
11256   THEN STATES(ROOT,T_SIZE+1) := TRY(0) := IP ;
11257   TOT_S := T_NODES - INIT_T_NODES +1 ;
11258   STATES(ROOT,4) := TOT_S ;
11259   PLEX       := IF PARTIAL AND FINISH OR FINAL = -5
11260                 THEN 0
11261                 ELSE DEEPER + 1 ;
11262   STATES(ROOT,2) := PLEX ;
11263   IF STATES(ROOT,5) < 1
11264   THEN STATES(ROOT,5) := TYPE ;
11265   SIDE := -SIDE ;
11266   IF TEST
11267   THEN WRITEON(REASON(FINAL), PLEX, TYPE) ;
11268   AB(LEVEL) := MX ;
11269   IF (LEVEL > 1)
```

```
11270   THEN BEGIN
11271     LEVEL := LEVEL -1 ;
11272     ROOT := ROOTNODE(LEVEL) ;
11273     RESTORESTATE(ROOT) ;
11274     RESTOREBACK(LEVEL, TRUE) ;
11275     TAKEBACKMOVE(TRUE) ;
11276     IF BETA > B_SCR AND PRINCVAR
11277     THEN BEGIN
11278       IF TYPE = 0
11279       THEN REFUTE(1000*M2 + M3) ;
11280       IF TEST
11281       THEN WRITEON("**", MX, M2, M3) ;
11282     END ;
11283   END ELSE MAXSC(IP) := MX ;
11284   ID := IP ;
11285   B_SCR
11286 END OFTREE ;
11287
11288 @TITLE,"SEARCH"
11289   LOST := VALUES(KNIGHT)*SCALE ;
11290   SCORES := TRUE ;
11291 COMMENT essential for EXAMINE mode ;
11292   IF EXAMINE OR ~CBOTH AND (WHO OR FLAG(1) = 0)
11293   THEN IF (ORIGIN > 0)
11294        THEN DEADTREE(REPLY)
11295        ELSE INIT_TREE ;
11296   IF TOURNAMENT
11297   THEN BEGIN
11298     N := TIME_LEFT DIV (MOVES_LEFT +1) ;
11299 WRITE("SEARCH MODE ", TIME_LEFT, MOVES_LEFT, M_TIMES(MODE),
11300          N, MODE, ORIGIN);
11301     IF MOVES_LEFT <= 0
11302     THEN BEGIN
11303       COMMENT extend time control to 30 in 30;
11304       MOVES_LEFT := MOVES_LEFT +30 ;
11305       TIME_LEFT := TIME_LEFT +108000 ;
11306     END ;
11307     IF N < M_TIMES(MODE) OR N > M_TIMES(MODE+1)
11308     THEN BEGIN
11309       WHILE N > M_TIMES(MODE) AND MODE < 6
11310       DO MODE := MODE +1;
11311       WHILE N < M_TIMES(MODE) AND MODE > 0
11312       DO MODE := MODE -1;
11313     IF MODE >= 1 AND MODE <= 6
11314     THEN BEGIN
11315       CASE MODE OF
11316       BEGIN
11317         BLITZ ;
11318         ;
11319         ;
11320         INTERMEDIATE ;
11321         TOURNAMENTS ;
11322         EXPERIMENTAL ;
11323       END ;
11324     END ;
11325   END;
11326   WRITEON(ORIGIN, N, MODE);
11327 END ;
11328 COMMENT collect opponent tree ;
11329   SECOND := SMALL := FALSE ;
11330   POINT := 0 ;
11331   IF (ORIGIN <= 0)
11332   THEN BEGIN
11333     ORIGIN := GETNODE(1);
11334     NODETO(ORIGIN,0) := -FIVES ;
11335     POINT := 1;
11336     ALPHA := SCALE*ONEVAL ;
11337   END ELSE
11338   BEGIN
11339     RESTORESTATE(ORIGIN) ;
```

```
11340     ALPHA := SCORE(TRY(1)) ;
11341   END ;
11342 COMMENT WRITE("SEARCH", ORIGIN, NODETO(ORIGIN,0), POINT, MAXL);
11343   DEEP   := IF EARLY
11344             THEN 5
11345             ELSE IF ~ENDGAME
11346                  THEN 7
11347                  ELSE IF TOURNAMENT AND ~QUICK
11348                       THEN 9
11349                       ELSE 7 ;
11350   IF MODE = 6
11351   THEN DEEP := IF LATE_END
11352                THEN 9
11353                ELSE IF ENDGAME
11354                     THEN 7
11355                     ELSE 5 ;
11356   IF MODE = 6
11357   THEN DEEP := MAXLEV ;
11358   IF DEEP > MAXLEV
11359   THEN DEEP := MAXLEV ;
11360   FOR I := 0 UNTIL ELIMIT
11361   DO BEGIN
11362     BACK(0,I) := SQUARE(I) ;
11363     BACK(0,I+ELIMIT+1) := LOSS(I) ;
11364   END ;
11365   E_NUM := 35 ;
11366   KTEST := 0 ;
11367   TOTALS := MAXTREE -MAXL ;
11368   LIMBS := BRAN ;
11369   B_SCR := 0 ;
11370   SAL := SAL + TOTALS ;
11371   LEVEL := IF POINT = 0 AND DEEP >= 3 AND MAX_WID ~= 19
11372            THEN 3
11373            ELSE 1 ;
11374   PLY := LEVEL ;
11375   WHILE PLY <= DEEP AND (WHO OR FLAG(1) = 0) AND B_SCR < SEVENS
11376     AND (KTEST = 0 OR REPLY > 0)
11377   DO BEGIN
11378     MAXDEPTH := LENGTH := PLY ;
11379     EXPECTATION := ALPHA ;
11380     BETA := ALPHA + THRESHOLD ;
11381     ALPHA := ALPHA - THRESHOLD ;
11382     MINDEPTH := MAXDEPTH -2 ;
11383     IF EARLY
11384     THEN MINDEPTH := 5 ;
11385     IF MINDEPTH < 3
11386     THEN MINDEPTH := 3 ;
11387     IF (E_NUM < 16) AND TOURNAMENT AND ~EARLY
11388     THEN BEGIN
11389       NEWLEVEL := MAXDEPTH ;
11390       NEWNODES := MAXL DIV 3 ;
11391     END ELSE IF (E_NUM < 31) OR TOURNAMENT
11392              THEN BEGIN
11393                NEWLEVEL := MAXDEPTH -2 ;
11394                NEWNODES := MAXL DIV (IF TOURNAMENT
11395                   THEN 3
11396                   ELSE 5) ;
11397     END ELSE
11398     BEGIN
11399       NEWLEVEL := MINDEPTH ;
11400       NEWNODES := MAXL DIV 5 ;
11401     END ;
11402     IF (MINDEPTH < NEWLEVEL)
11403     THEN MINDEPTH := NEWLEVEL ;
11404     IF (MAXNODES < NEWNODES)
11405     THEN MAXNODES := NEWNODES  +1 ;
11406     IF (MAXDEPTH > LENGTH)
11407     THEN MAXDEPTH := LENGTH ;
11408     IF (MINDEPTH > MAXDEPTH)
11409     THEN MINDEPTH := MAXDEPTH ;
```

```
11410      IF MODE = 6 AND DEEP = 5 AND ~OPENING
11411      THEN MINDEPTH := MAXDEPTH ;
11412  REPEAT:
11413      REPLY := 0 ;
11414      FULLWIN := TRUE ;
11415      REUSE := MAXL ;
11416      FOR LEV := 1 UNTIL MAX_PLIES
11417      DO BEGIN
11418         FOR I := 0 UNTIL 10
11419         DO REFUT(LEV,I) := REFCNT(LEV,I) := 0 ;
11420        FOR I := 1 UNTIL MAX_WID+MAX_PLIES
11421        DO REFTAB(I,LEV) := 0;
11422      END;
11423      IF MONITOR AND ~TEST
11424      THEN BEGIN
11425        TRACER(3, "?NPD") ;
11426        MON := TEST := TRUE ;
11427      END ELSE MON := FALSE ;
11428      NODE := ORIGIN ;
11429      SIDE := 1 ;
11430      NEW := 0 ;
11431      BRANCHES := BRAN ;
11432      BEST := -EIGHTS + 1 ;
11433      T_NODES := 0 ;
11434      LEVEL := 1 ;
11435    PATH := 0;
11436      IF PAB
11437      THEN ALPHA := -EIGHTS ;
11438      IF PAB
11439      THEN BETA := EIGHTS ;
11440      FIRSTONE := 0 ;
11441      FOR I := 1 UNTIL MAXTREE
11442      DO BEGIN
11443        COMMENT make sure individual nodes are reclaimed, not individual
11444          subtrees ;
11445        PARENT(I) := PARENT(I) + 1000 ;
11446        STATES(I, 2) := 0 ;
11447      END ;
11448      SACR_ANAL := LENGTH <= 3 AND MAX_WID ~= 19
11449                              OR MAX_WID = 19 AND LENGTH > 3;
11450      B_SCR := TREE(ALPHA, BETA, HASH, MINDEPTH-1, PAB, TYPE) ;
11451      LEVEL := 1 ;
11452      T_NODES := NODETO(ORIGIN,0) ;
11453      SAVEL := POINT + 1 ;
11454      TOTALS := TOTALS + NEW ;
11455      BRANCHES := BRAN - BRANCHES ;
11456      E_NUM := IF BRANCHES <= 0 OR NEW <= 0
11457               THEN E_NUM
11458               ELSE BRANCHES DIV NEW ;
11459      IF (TEST)
11460      THEN WRITE(LEVEL,MINDEPTH,SAVEL,MAXL, T_NODES,MAXNODES,NOD,DUP,
11461        BEST,  OLDSCR, PREVSCR, ORIGIN, NODETO(ORIGIN,REPLY),
11462  GETNODE(POINT), GETNODE(SAVEL));
11463      IOCONTROL(2) ;
11464      IF MON
11465      THEN BEGIN
11466        MON := TEST := FALSE ;
11467        TRACER(1, "?N") ;
11468      END ;
11469      SECOND := IF TOURNAMENT AND (BEST < EXPECT)
11470                THEN TRUE
11471                ELSE FALSE ;
11472      IF LENGTH > 1
11473      THEN IF SKILL > "  BEGINNER" OR EXAMINE OR TEST OR MONITOR
11474        OR FLAG(1) = 0
11475           THEN BEGIN
11476            T_NODES := ID;
11477             IF ~MONITOR AND ID > 8
11478             THEN ID := 8 ;
11479             WRITE(B_SCR,EXPECT,": ") ;
```

```
11480           FOR I := 1 UNTIL ID
11481           DO WRITEON(MAXSC(I)) ;
11482           WRITE(ALPHA, BETA, ": ") ;
11483           FOR I := 1 UNTIL ID
11484           DO WRITEON(BOX(MOVEFROM(TRY(I))), BOX(MOVETO(TRY(I))),
11485             " ") ;
11486           WRITE(BRANCHES, "SCORED ", NEW, "/", TOTALS, "POSNS.",
11487             REPLY, "/",T_NODES) ;
11488           WRITEON(" DEPTH =", LENGTH, E_NUM, (TIME(1)-OLDCPU)
11489             DIV 60) ;
11490           IOCONTROL(2) ;
11491     END ;
11492     IF FALSE AND (TEST OR MONITOR) THEN BEGIN
11493       WRITE("REFUTATIONS ");
11494       IOCONTROL(2);
11495       FOR I := 1 UNTIL ID DO BEGIN
11496         INTERIOR := TRUE;
11497         FOR LEV := 1 UNTIL MAX_PLIES
11498         DO IF INTERIOR AND REFTAB(I,LEV) > 0
11499            THEN WRITEON(REFTAB(I,LEV))
11500            ELSE INTERIOR := FALSE;
11501         IOCONTROL(2);
11502       END;
11503     END;
11504     FOR I := 0 UNTIL ELIMIT
11505     DO BEGIN
11506       SQUARE(I) := BACK(0,I) ;
11507       LOSS(I) := BACK(0,I+ELIMIT+1) ;
11508     END ;
11509     IF LENGTH > 1 AND (WHO OR FLAG(1) = 0)
11510     THEN IF B_SCR ~= BEST OR B_SCR < ALPHA OR B_SCR > BETA
11511          THEN BEGIN
11512            IF B_SCR > ALPHA AND B_SCR < BETA OR REPLY <= 0
11513            THEN WRITE("ALPHA-BETA ERROR ", B_SCR, BEST, REPLY,
11514              ALPHA, BETA)
11515            ELSE BEGIN
11516              GAMETREE(ORIGIN, 1, 1) ;
11517         COMMENT LISTMOVES;
11518            END ;
11519            IOCONTROL(2) ;
11520            IF REPLY <= 0
11521            THEN REPLY := 1 ;
11522            IF B_SCR <= ALPHA
11523            THEN BEGIN
11524              INVALIDATE(ORIGIN) ;
11525              BETA := B_SCR +1 ;
11526              ALPHA := B_SCR -THRESHOLD ;
11527              GOTO REPEAT ;
11528            END ELSE IF B_SCR >= BETA
11529                     THEN BEGIN
11530                       INVALIDATE(ORIGIN) ;
11531                       ALPHA := B_SCR -1 ;
11532                       BETA := B_SCR +THRESHOLD ;
11533                       GOTO REPEAT ;
11534            END ;
11535     END ;
11536     ALPHA := SCORE(TRY(1)) ;
11537          COMMENT retrieve nodes (if any) in width+1 ... 2*width;
11538     IF MAX_WID = 19
11539     THEN IF (~LATE_END AND PLY = 3 OR PLY = 7 OR PLY = 5 AND
11540             (~ENDGAME OR
11541              WIDTH = MAX_WID OR TIME(1)-OLDCPU > M_TIMES(MODE) DIV 3))
11542     THEN WIDTH := WIDTH DIV 2 ;
11543     IF MODE = 6
11544     THEN IF PLY >= 5 AND (E_NUM >= 30 OR TOTALS > MAXTREE)
11545      AND (TIME(-1) -SENDTIME > 7200 OR TIME(1) - OLDCPU > 5400)
11546       OR TOTALS > 2*MAXTREE
11547          THEN DEEP := 5
11548          ELSE IF PLY >= 7 AND (E_NUM >= 15 OR PIECES(0) > 12)
11549               THEN DEEP := 7 ;
```

```
11550       IF TRY(WSIZE) = 1
11551       THEN PLY := DEEP;
11552     PLY := PLY + 2 ;
11553   END ;
11554   LIMBS := BRAN - LIMBS ;
11555   WIDTH := MAX_WID ;
11556   IF  TEST
11557   THEN WRITE(TOURNAMENT,BEST,EXPECT, SMALL,REPLY,SECOND, FORCEDREPLY,
11558     MINDEPTH, MAXDEPTH) ;
11559   IOCONTROL(2) ;
11560   WRITE(" ") ;
11561   IF (REPLY = 0)
11562   THEN BEGIN
11563     REPLY := 1 ;
11564       COMMENT  mate/stalemate detected ;
11565     WRITE("ERROR IN SEARCH", TRY(1), NUM,  BEST, MAXSC(REPLY),
11566       TRY(0),TRY(WSIZE),ORIGIN) ;
11567   END ;
11568   LASTORIGIN := ORIGIN ;
11569   RTMV := TRY(1) ;
11570   IF RTMV = 0
11571   THEN WRITE("SEAR", TRY(0), TRY(1), TRY(WSIZE)) ;
11572   SCORE(RTMV) := MAXSC(REPLY) ;
11573   PREDICT := REPLY ;
11574   IF WHO OR ~EXAMINE
11575   THEN DEADTREE (1)
11576   ELSE ORIGIN := NODETO(LASTORIGIN,1) ;
11577 END OFSEARCH ;
11578
11579   WHILE TRUE
11580   DO HUB ;
11581 END OFMAIN ;
11582
11583 PROCEDURE ATTENTION(INTEGER ARRAY FLAG(*)) ;
11584 ;
11585 comment FORTRAN "ATTN#   " ;
11586
11587 PROCEDURE TRANSFER(STRING(140) VALUE RESULT B ;
11588 INTEGER VALUE RESULT L ;
11589 INTEGER VALUE M ;
11590 INTEGER VALUE RESULT N ;
11591 INTEGER VALUE U, RW) ;
11592 ;
11593 comment FORTRAN "TRANSF" ;
11594
11595 PROCEDURE DATER(INTEGER VALUE KEY,PR ;
11596 STRING(12) RESULT DATE) ;
11597 ;
11598 comment FORTRAN "TIME" ;
11599
11600 PROCEDURE MMTTS(INTEGER VALUE INDEX ;
11601 STRING (140) VALUE RESULT HISTORY) ;
11602 ;
11603 comment FORTRAN "DEPEND" ;
11604
11605 PROCEDURE INFLUENCE(LOGICAL VALUE FINAL);
11606 BEGIN
11607   INTEGER TOP, J, K_FILE, I;
11608   K_FILE := FILES(K);
11609     IF REMEMBER AND FINAL
11610     THEN BEGIN
11611       LSAVE := SQUARE(0) ;
11612       ESAVE := SQUARE(ELIMIT) ;
11613     END ;
11614       IF (ABS(REWARD(N)) = PFTWO)
11615       THEN IF (ABS(BRD(M3-K_FILE)) = ENPRIS)
11616            THEN BEGIN
11617              I := LOSS(0) := LOSS(0)+1 ;
11618              SQUARE(I) := M3-K_FILE ;
11619              LOSS(I) := 0 ;
```

```
11620       END ;
11621   TOP := LOSS(ELIMIT) ;
11622   FOR I := TOP UNTIL SQUARE(ELIMIT)-1
11623   DO BEGIN
11624     J := I + 1 ;
11625     WHILE (J < ELIMIT)
11626     DO IF (SQUARE(J) = SQUARE(I))
11627        THEN BEGIN
11628          IF (I >= ESAVE)
11629          THEN BEGIN
11630            SQUARE(I) := SQUARE(ESAVE) ;
11631            SQUARE(ESAVE) := SQUARE(TOP) ;
11632            ESAVE := ESAVE + 1 ;
11633          END ELSE SQUARE(I) := SQUARE(TOP) ;
11634          J := ELIMIT ;
11635          TOP := TOP + 1 ;
11636     END ELSE J := J + 1 ;
11637   END ;
11638   LOSS(ELIMIT) := TOP ;
11639   TOP := LOSS(0) ;
11640   FOR I := LOSS(0) STEP -1 UNTIL SQUARE(0)+1
11641   DO BEGIN
11642     COMMENT purge duplicated squares from enpris list ;
11643     J := I - 1 ;
11644     WHILE (J > 0)
11645     DO IF (SQUARE(I) = SQUARE(J))
11646        THEN BEGIN
11647          IF (I <= LSAVE)
11648          THEN BEGIN
11649            SQUARE(I) := SQUARE(LSAVE) ;
11650            SQUARE(LSAVE) := SQUARE(TOP) ;
11651            LSAVE := LSAVE -1 ;
11652          END ELSE SQUARE(I) := SQUARE(TOP) ;
11653          J := 0 ;
11654          TOP := TOP -1 ;
11655     END ELSE J := J - 1 ;
11656   END ;
11657   LOSS(0) := TOP := TOP + 1 ;
11658   SQUARE(TOP) := M3 ;
11659   LOSS(TOP) := -NINES ;
11660 END INFLUENCE;
11661
11662 LOGICAL PROCEDURE CLEAR(INTEGER VALUE SQA, SQF, SQT);
11663 BEGIN
11664   INTEGER DIR;
11665   LOGICAL FREE;
11666   FREE := FALSE;
11667   DIR := BOTV(EDGE, OFFSET(SQF)-OFFSET(SQT));
11668   IF DIR ~= 0 THEN BEGIN
11669   FREE := TRUE;
11670     IF SQA ~= SQF AND
11671        DIR = BOTV(EDGE, OFFSET(SQA)-OFFSET(SQT))
11672     THEN FREE := FALSE ;
11673     FOR SQ := SQF+DIR STEP DIR UNTIL SQT-DIR
11674     DO IF FREE AND BRD(SQ) ~= 0
11675        THEN FREE := FALSE;
11676   END;
11677   FREE
11678 END CLEAR;
11679
11680 PROCEDURE REFLECT(INTEGER ARRAY BRD(*));
11681 BEGIN
11682   INTEGER I, J, M, L, PC;
11683   LOGICAL C;
11684   FOR J := 0 UNTIL 3
11685   DO FOR I := 11 UNTIL 18
11686   DO BEGIN
11687     M := J*10 + I;
11688     L := (7-J)*10 + I;
11689     PC := BRD(M);
```

```
11690      BRD(M) := -BRD(L);
11691      BRD(L) := -PC;
11692    END;
11693    K := -K;
11694    FOR I := 1 UNTIL 3
11695    DO BEGIN
11696      C := CASTLE(I);
11697      CASTLE(I) := CASTLE(-I);
11698      CASTLE(-I) := C;
11699    END;
11700    PUT(4);
11701    PUNCHBRD(TRUE, HISTORY(0|140));
11702    WRITECARD(HISTORY(0|114));
11703    PUT(TTYOUT);
11704  END REFLECT;
11705
11706  STRING(6) PROCEDURE INTOCHAR(INTEGER VALUE NUMBER);
11707  BEGIN
11708    INTEGER X, I, NUM;
11709    STRING(6) TMP;
11710    TMP := IF NUMBER >= 0 THEN "     0" ELSE "-    0";
11711    NUM := ABS(NUMBER);
11712    I := 5;
11713    WHILE NUM > 0
11714    DO BEGIN
11715      X := NUM REM 10;
11716      NUM := NUM DIV 10;
11717      TMP(I|1) := CODE(X + 240);
11718      I := I -1;
11719    END;
11720    IF I < 0 THEN
11721     WRITE("ERROR, NUM TOO LONG ", NUMBER, TMP, I, X)
11722    ELSE BEGIN
11723      TMP(I|1) := TMP(0|1);
11724      IF I > 0 THEN TMP(0|1) := " ";
11725    END;
11726    TMP
11727  END INTOCHAR;
11728
11729  INTEGER PROCEDURE CHARTOINT (STRING(6) VALUE NUM);
11730  BEGIN
11731    INTEGER I, N;
11732    LOGICAL NEGATIVE;
11733    I := 0;
11734    WHILE NUM(I|1) = " "
11735    DO I := I + 1;
11736    NEGATIVE := NUM(I|1) = "-";
11737    IF NEGATIVE THEN I := I +1;
11738    N := 0;
11739    WHILE I < 6 DO BEGIN
11740      N := 10*N + DECODE(NUM(I|1)) - 240;
11741      I := I +1;
11742    END;
11743    IF NEGATIVE THEN N := -N;
11744    N
11745  END CHARTOINT;
11746
11747  INTEGER PROCEDURE NEWHASH(INTEGER ARRAY BRD(*) ;
11748  INTEGER VALUE K ;
11749  STRING(2) RESULT KEY) ;
11750  0 ;
11751  comment ALGOL "NEWHASH" ;
11752
11753    FLAG(1) := 1 ;
11754    ATTENTION(FLAG) ;
11755    TTYOUT := 6 ;
11756    GET(5) ;
11757    PUT(TTYOUT) ;
11758    S_W := 1 ;
11759    MAXCHAR := 118 ;
```

```
11760   ELIMIT := 31 ;
11761   LIM := 29 ;
11762   HOLE := -11 ;
11763     COMMENT one word buffer seems necessary ;
11764   BAD := -3333 ;
11765   COMMENT see initialize ;
11766   WSIZE := 20 ;
11767   RLIMIT := 15 ;
11768   CONLIM := 8 ;
11769   I_W := 4 ;
11770   ZEROTIME := TIME(-1) ;
11771   MAXTREE := 80 ;
11772   MAXPLY := 2 ;
11773   MAXLEV := 3 ;
11774   TOTAL := 80 ;
11775   WIDTH := 9 ;
11776   GO TO ENTER ;
11777   ENTRY:
11778   WRITE("Enter MAXTREE, MAXLEV, TOTAL, WIDTH") ;
11779   WRITE("If zeros then defaults of 600, 7, 80, 7 are used") ;
11780   WHILE READU
11781   DO ;
11782   MAXTREE := CONVERTS(U,INERR) ;
11783   IF INERR
11784   THEN GO TO ENTRY ;
11785   MAXLEV := CONVERTS(U,INERR) ;
11786   IF INERR
11787   THEN GO TO ENTRY ;
11788   TOTAL := CONVERTS(U,INERR) ;
11789   IF INERR
11790   THEN GO TO ENTRY ;
11791   WIDTH := CONVERTS(U,INERR) ;
11792   IF INERR
11793   THEN GO TO ENTRY ;
11794   MAXTREE := IF (MAXTREE <= 0)
11795              THEN 600
11796              ELSE IF MAXTREE < 100
11797                   THEN 100
11798                   ELSE MAXTREE ;
11799   MAXLEV := IF (MAXLEV <= 0)
11800             THEN 7
11801             ELSE IF MAXLEV < 3
11802                  THEN 3
11803                  ELSE MAXLEV ;
11804   TOTAL := IF (TOTAL < 20)
11805            THEN 80
11806            ELSE TOTAL ;
11807   WIDTH := IF (WIDTH <= 0)
11808            THEN 9
11809            ELSE IF WIDTH < 5
11810                 THEN 5
11811                 ELSE WIDTH ;
11812 ENTER:
11813   DEPTH_LIMIT := MAX_PLIES := MAXLEV + MAXPLY ;
11814   T_SIZE := 8 ;
11815   S_SIZE := 20 + 2*RLIMIT ;
11816   TSIZE := S_SIZE + 83 + 2 + 2*ELIMIT ;
11817   CAPT_WID := 14 ;
11818   WIDTH_LIMIT := MAX_WID := WIDTH ;
11819   W_LIM := 4*(CAPT_WID) ;
11820   ASSERT(WSIZE > CAPT_WID AND TOTAL <= 80 AND RLIMIT <= 15) ;
11821   ASSERT(LIM = 29 AND CONLIM = 8 AND ELIMIT = 31 AND WIDTH <= MAX_WID) ;
11822   INITIALIZE ;
11823 STARTS:
11824   LENGTH := MAXLEV ;
11825   MAIN ;
11826 STOP:
11827   COKO := FALSE ;
11828   ZEROTIME := TIME(-1) - ZEROTIME ;
11829   IF ZEROTIME < 0
```

```
11830   THEN ZEROTIME := ZEROTIME + 5184000 ;
11831   WRITE("PROCESSOR TIME = ",TIME(1) DIV 60,
11832     "SECS.  ELAPSED TIME = ", (ZEROTIME) DIV 3600,"MINS.") ;
11833   IF BOOK ~= 1
11834   THEN GAMERECORD(TRUE) ;
11835
11836   END.
11837 COMMENT                    24 Oct 1980
11838           credits and debits in MAKEMOVE
11839
11840  C(0) to C(13) accumulated for side to move: C(14) & C(15) not.
11841
11842  0  -DEBIT(LEVEL-2)
11843  1  early pawn structures to encourage Bishop moves  (1,2)
11844  2  doubled pawn, not recapturable       (-2)
11845  3  pawn capture undoubling pawn        (1)
11846  4  phalanx of pawns or back pawn push    (1,2)
11847  5  capture of pawn leads to recapture which doubles pawns (1,2)
11848  6  unsafe promotion    (-20)
11849  7  rook move on/off (half) open file (+-1,+-2)
11850     open file for rook upon castling      (1,2)
11851  8  pawn capture blocks/unblocks rook on file  (-1,1)
11852     doubling/undoubling rooks      (+-1)
11853     unless fully blocked file      (0)
11854  9  penalize p-r4 if can castle or early and
11855        opponent  not castled on that side.    (-2)
11856  10 penalize pawn forward two infront of king, except p-k4     (-1)
11857  11 penalize queen move in early opening    (-3)
11858          or moving king and losing castling priviledge      (-1)
11859              fork threat credit          (+3)
11860  12 capture backward pawn                 (1)
11861  13 tempo loss penalty      (-1)
11862  14 take rook on/off column of own passed pawn    (+-1)
11863  15 credit for being able to make safe push of passed pawn   (L+1)
11864
11865  debits D(0) to D(12) accumulated for opponent: not D(13), .. D(15)
11866
11867  0  0
11868  1  pawn capture isolates/joins pawns    (+-1)
11869  2  credit for pushing secondary passedpawn    (T)
11870  3  created passedpawn for opponent      (-T)
11871  4  hole creation debit          (-1)
11872     fix opponent hole           (2)
11873     give up fix on hole        (-2)
11874  5  capture of pawn creates a secondary passedpawn    (T)
11875  6  captured an isolated pawn       (-1)
11876  7  capture advanced, or principal/secondary passed, pawn    (T)
11877  8  moving rook onto row adjacent to king      (1)
11878     moving rook off row adjacent to king, and not giving check (-1)
11879  9  Fill hole with N, P or B             (T)
11880  10 blocking own pawn in opening, or central pawn anytime    (-1)
11881  11 unblocking backward pawn, own/opponent       (+-1)
11882       or capture of backpawn                    (1)
11883  12 safe promotion    (MAXLEV-LEVEL+1)
11884  13 capture of pawn creates new passed pawn    (T)
11885  14 opponent has passed pawn      (T)
11886  15 credit for pushing primary passed pawn        (T)
11887     credit for having a primary passed pawn      (T)
11888 ;
11889 COMMENT    definition of tree statistics symbols.
11890
11891   BRAN   - # branches in positions considered.
11892   BRAN_S - # branches actually scored,  BRAN_s <= BRAN
11893   NOD    - # positions scored, BRAN/NOD = average tree width.
11894   TOT    -  # terminal nodes
11895   DYN    - # dynamic terminal nodes    DYN <= TOT
11896   CAPT   - # positions from capture tree.
11897   SAL    - size of trees carried forward at each move.
11898   DUP    - # duplicate nodes within tree.
11899   REC    - # reclaimed positions from discarded trees.
```

```
11900    DUP_S  - # positions in duplicated subtree
11901    REC_S  - # positions that were in discarded subtree
11902    TOT_S  - counter keeping track of subtree size.
11903 ;
```