CHECKMATE

A computer program for playing a chess ending

PCW *Readers, look out for an article on how to approach chess programming. It's by International Chess master David Levy, and will appear in the next issue.* **PCW**

Francis Best

Introduction

I am 18 years old and started computing when I entered the sixth form at school. A thriving interest in chess caused the obvious link to some kind of chess playing program but the aim of this article is not just to explain how my program works. I also hope to indicate some of the problems encountered and how I overcame them. In short, I would like to paint a more human picture of how an idea is conceived and translated into something which is meaningful in a computer. This, I think, is important; especially since I was relatively inexperienced in programming when I started 'Checkmate' (as I decided to call it), and was not entirely sure how I was to set about writing the program.

1. Early considerations

The first thing to be considered was the basic structure of the program. One method would be to devise a set of rules for determining each move. This would have the disadvantage of only being applicable to the ending of king and rook versus king, which I had chosen. It would also be a rather inflexible approach, since it would be difficult to modify rules in the light of experience.

Another method would be to scan ahead each position a number of moves and choose the line which seems to drive the opposing king towards the edge of the board. This system is the closest to how computers are currently programmed to play a whole game of chess, but most of these programs do not play very well, so that left me with one alternative.

The final method involves considering each move and scoring it according to its various merits. Parameters could be used to adjust the comparative importance of each of the heuristics.

Although the third approach was eventually adopted, and produced a reasonable level of play, it would be much more difficult to use the same technique for a whole game, when many more factors than I used would have to be taken into account.

There were a few other minor considerations. For the input and output of moves, descriptive chess notation was used. This was possibly unwise as there was little string handling available in the version of BASIC which I used. Consequently, I stored all possible strings so that they could be compared with the input. Clearly this would be impossible on a small system. For input of the starting position, I used coordinates (file then rank).

2. Move generation

Given that we have stored the starting position in the computer, it is a relatively simple matter to generate a list of all legal moves by performing simple additions on the coordinates of each piece. For the rook, we start in the square on the same file, on rank 1, then we add one to the second (rank) coordinate. If the original position of the rook is encountered, that move is eliminated and if the position of the white king is encountered, move generation along that file is stopped. The process is then repeated for the moves along the rank on which the rook is standing.

This system assumes that the white king is not between the square of the rook and the square, on which move generation is started. If this situation does arise, the method is similar, except that generation occurs *in front of* the (white) king to the last rank on that file. This is made clearer by a diagram:

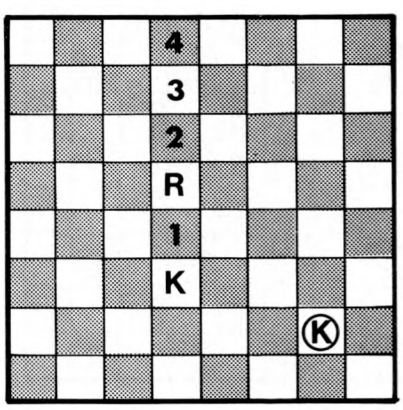

Fig. 1 Ⓚ = **black king**
Numbers indicate order in which rook moves are generated along file.

The king moves present little problem. Here, 1, 9, 10 and 11 are added

and subtracted to the original position and a subroutine is called into play to check the legality of the move. The move is stored, if legal; and rejected otherwise.

2. The legality subroutine

This subroutine has a number of uses. It checks that the coordinates of the pieces are neither less than 1 nor greater than 8, and that no two pieces are on the same square. Finally, if the rook is attacking the black king, then the position is counted as illegal. The variable L is set to 1 or 0 according to whether legality is verified or not.

The immediate usefulness of the subroutine is in checking the original position input. It is also useful in generating the moves of the white king as indicated above. Another use is to see if black is in check. If the king is actually attacked, it depends on whose move it is whether a position is really legal or not.

4. The heuristics

(i) This heuristic encourages reducing the number of squares that the black king can move to. The score given to each move is kx(8 - s), where s is the number of moves available to the black king, and k is a weighting factor which will be discussed later. The only problem with this heuristic is that it would encourage stalemate. Therefore, the position is examined to see if black is in check (this is after s has been found to be 0). If he is, then that move will be made (since it is checkmate), otherwise the score for that move is taken to be -1000, thus assuring that that move will not be made. Incidentally, a score of -1000 is also given to a position where the white rook can be taken.

An example of the usefulness of this heuristic can be seen from the following diagram:

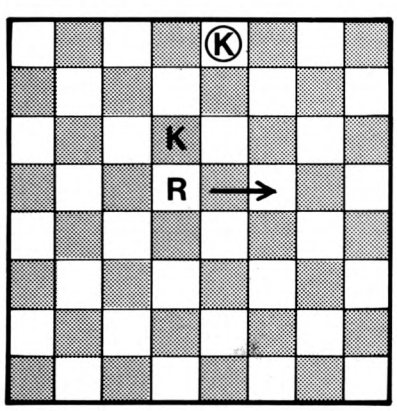

Fig. 2

Here, the move which allows black only one move would be encouraged by heuristic 1. Black would then be forced to play K–Q1 and R–KB8 is checkmate.

(ii), (iii) & (iv) These heuristics are all linked, so it is better to treat them as one unit. They are governed by the following expressions:

Heuristic	*Expression*
2	k_1 (14-distance of white and black kings)
3	k_2 (14-distance of w. rook and b. king)
4	k_3 (14-distance of white rook and king)

The distance being simply defined as the sum of the differences of the corresponding coordinates. The general idea is for the program to keep the pieces close together, the weightings (k_1) providing some way of introducing priorities. The following diagram illustrates how the parameters work.

The three moves indicated would all be quite possible: heuristic (i) would encourage move 1. But if R–Q7 is played, black can reply . . . K–K3, and white will have to move his rook away from danger. With suitable weightings for heuristics (ii) – (iv), though move 2 or 3 can be encouraged, either being better than move 1.

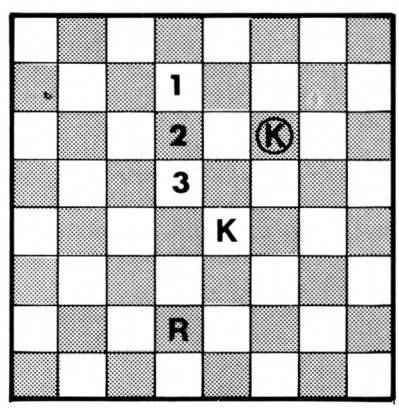

Fig. 3

(v) This heuristic is relatively unimportant. The idea is that the zone marked on the diagram is deemed less preferable for the rook to be on than the rest of the board. Similarly for the white king.

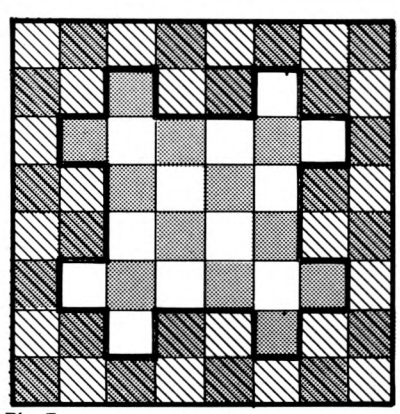

Fig. 5
Zone marked is less preferable for white king

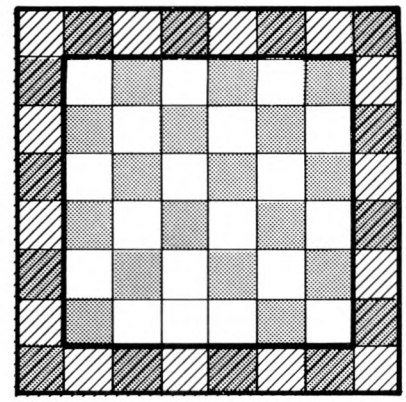

Fig. 4
Zone marked is less preferable for rook

The underlying idea of this heuristic is that the white pieces should avoid moving to squares on the edge of the board.

The preceding heuristics were those I arrived at somewhat informally at the outset. They were based on a simplified analysis of my own thoughts on how the ending should be played. After a little experimentation I decided to add another heuristic:–

(vi) This heuristic turned out to be useful in several unexpected situations. Basically, its aim is to encourage white to place the rook so that it is more or less the same distance from both of the kings. This is accomplished by dividing the number of moves that the black king would require to reach the rook by the corresponding number for the white king. This is then fed into a quadratic whose maximum is 1¼. The reason for having the maximum just greater than 1 is that, generally, it is better for the rook to be slightly closer to the white king than the black.

For calculating the shortest number of moves that a king requires to reach a certain square, it is only necessary to take the maximum of the two differences in corresponding co-ordinates (see diagram).

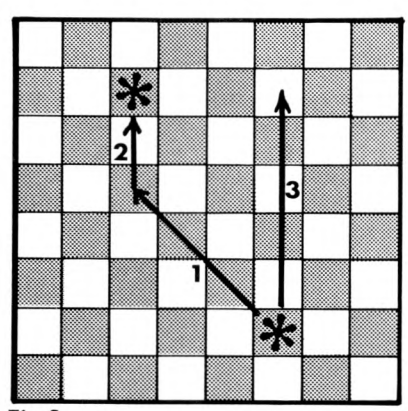

Fig. 6
Shortest route between the 2 squares marked with an asterisk is indicated by arrows 1 and 2. Route is same length as arrow 3, which is largest difference between co-ords.

The following position illustrates the usefulness of the heuristic (see Fig. 7). Here, white does not really want to do anything for a move. The moves indicated are examples of reasonable ways in which this can be done. Move 3 has the disadvantage that the black king can escape towards the centre of the board. If, however, we weight heuristic (iv) so that it is more important than (ii), that move will be eliminated. Move 2 is slightly inferior to 1 because the movement of the rook is more limited. This is where heuristic (vi) comes in. The actual quadratic used was: $2\frac{1}{2}X-X^2$, where X is the fraction of the two distances. Now the value of the expression is ½ more for move 1 than 2, so if we weight the heuristics appropriately, our aim will have been achieved.

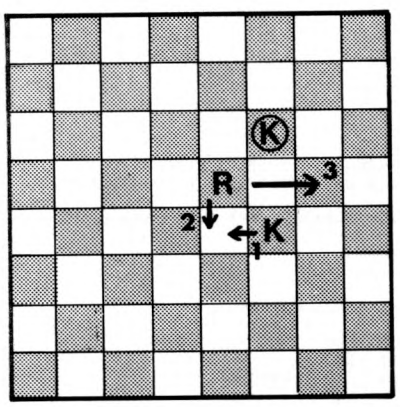

Fig. 7

5. Problems encountered

The weightings for the heuristics was the most difficult problem with this program. I first tried likely looking values and played various positions out to see the kind of move which it was found difficult to deal with. Having done this, it was then possible to set up a number of inequalities by comparing two high-scoring moves. This is best illustrated by an example.

Consider the following position:

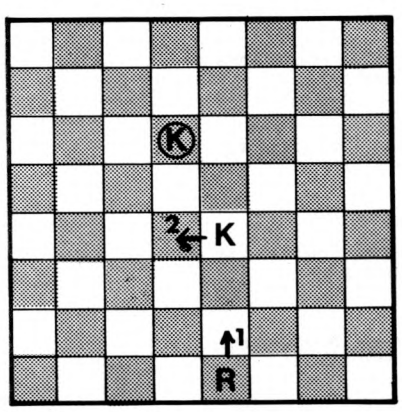

Fig. 8

Surprisingly, it was found that move 1 scored fairly highly. Move 2 is easily the best here, though. Taking each parameter in turn, we can compare them for both positions.

(i) After move 2, black has three moves. After 1, he has six.
(ii) Here the white king is one square closer after move 2 than 1.
(iii) This is better for move 1; the rook would be one square closer than for move 2.
(iv) This also favours 1; after move one, the rook and white king are 2 squares closer together than after 2 (remembering that it is not the actual distance but the difference in co-ordinates).
(v) This favours move 1 also. The king stays out of the low-scoring zone for both moves but the rook is on the edge of the board for 2 and not for 1.
(vi) This is in favour of move 2.
After move 2, we have:

$$(5/3).2\frac{1}{2}-(5/3)^2$$

After move 1, we have:

$$(4/2) \cdot 2\frac{1}{2}-(4/2)^2$$

Therefore, the overall advantage for 2 is (25/18)−1 or (7/18).

Taking all these factors into account, we require

$$k_5' + 3k_1 + k_2 + (7/18)k_6 > k_3 + 2k_4 + k_5''$$

where k_1 is the weighting parameter for heuristic number i, k_5' is the score added on when the rook is on the edge of the board and k_5'' the score added on when it is not on the edge.

Thus, it is possible to go about obtaining the best parameters in a methodical way. It also revealed rather a serious problem. In calculating the inequalities, I reached several requirements which could not be fulfilled together. So the heuristics were not sufficient on their own.

There were two ways round this problem. Either more heuristics could have been introduced, or a provision could have been made for changing the parameters according to the position. The first method, I felt, would have quite a high chance of failing for reasons similar to those for the failure of the original heuristics. Also, the second method seemed more true to life; a human chess-player adjusts the priorities of a position as the game progresses.

There was one type of position which seemed to contradict the tendencies of the heuristics (see Fig. 9). If the rook moves away as indicated, black has to play . . . K–K1 and is then mated with R–QB8. The heuristics, though would almost certainly give the highest score to K–Q6, which is a much slower ein. In fact, a similar move is probably best even when the king is not confined to the back rank. Such a move is therefore played automatically in positions of this type (as is the follow up of the rook checking the black king). This may seem a bit of a cheat but I maintain that this, too, is how a human chess player works: he knows a few general principles together with a limited number of exceptions.

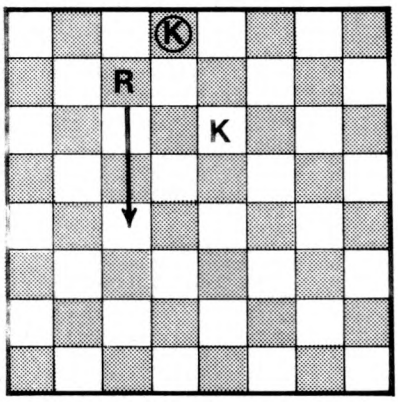

Fig. 9

Besides these special moves, a modification was introduced to recognize three classes of position and use a different set of parameters for each class. First, a check is made to see if the corresponding coordinates of the two kings differ by more than 2. If either set does, then parameters are chosen to encourage moving the white king closer to the black king. The other sets of parameters correspond to positions where the black king can be forced to an unfavourable square by a rook check and, finally to positions which most commonly occur i.e., those where white has no immediate win but must improve his position according to the general principles built into the program.

6. Success

At this point, it was possible to adjust the three sets of parameters so that a fairly good game could be played. I hasten to add that the program does not always checkmate in the most efficient manner, but the worst I have had so far is taking 23 moves to win. The average seems to be around 14 moves.

I enclose a listing of the program with a few sample runs. I leave the reader to form his own conclusions, but whatever one thinks of 'Checkmate', it was fun, and that's the main thing!

Final comments

The program was run on an ICL 1903 machine, where there was no shortage of storage. On a small system, though, an alternative approach would be to utilize a certain amount of look-ahead, thus reducing the need for so many heuristics. With care, look-ahead need not involve storing a large number of positions if moves are analysed and assessed fully before going on to consider another one. The actual process of look-ahead, being an essentially repetitive procedure, would not take up too much storage.

CHECKMATE

```
  10 REM  ROOK & KING V. KING CHESS ENDING (MK II)
  30 FILES MOVES
  40 DIM T$(440),U$(440),V$(100),W$(100)
  50 DIM R(14),D(14),F(14)
  60 DIM Y(30)
  70 REM  READ IN DATA FOR CHESS NOTATION.
  80 RESTORE #1
  90 FOR  A = 1 TO 8
 100   FOR  A1 = 1 ,TO 8
 110     READ #1,T$(10*A1+A)
 120   NEXT A1
 130 NEXT A
 140 FOR  A2 = 1 TO 8
 150   FOR  A3 = 1 TO 8
 160     READ #1,U$(10*A3+A2)
 170   NEXT A3
 180 NEXT A2
 190 READ Z(0),Z(1),Z(2),Z(3),Z(4),Z(5),Z(6),Z(7)
 200 REM  READ PARAMETERS FOR SCORING POSITIONS.
 210 READ A(0),A(1),A(2),A(3),A(4),A(5),A(6),A(7),A(8)
 220 READ B(0),B(1),B(2),B(3),B(4),B(5),B(6),B(7),B(8)
 230 READ C(0),C(1),C(2),C(3),C(4),C(5),C(6),C(7),C(8)
 240 PRINT TAB(20);"?";
 250 INPUT A$
 260 IF  A$ = "END"  THEN 9999
 270 IF  A$ = "NEW"  THEN 770
 280 IF  A$ = "INFO"  THEN 520
 290 IF  A$ = "OFF"  THEN 540
 300 IF  A$ = "C1"  THEN 560
 310 IF  A$ = "C2"  THEN 590
 320 IF  A$ = "C3"  THEN 610
 330 IF  A$ = "SHOW"  THEN 630
 340 FOR  A4 = 1 TO 8
 350   FOR  A5 = 1 TO 8
 360     A = 10*A5+A4
 370     IF  A$ = U$(A)  THEN 420
 380   NEXT A5
 390 NEXT A4
 400 PRINT "INVALID INPUT";
 410 GO TO 240
 420 H = 9-A+20*INT(A/10)
 430 A = ABS(H-H1)
 440 IF  (A-1)*(A-10)*(A-11)*(A-9) = 0  THEN 470
 450 PRINT "ILLEGAL MOVE";
 453 H = H1
 454 H2 = INT(H/10)
 455 H3 = H-10*H2
 460 GO TO 240
 470 H2 = INT(H/10)
 475 H3 = H-10*H2
 476 GO SUB 3290
 480 IF  L = 0  THEN 450
 485 H1 = H
 490 W$(I) = A$
 500 I = I+1
 510 GO TO 950
 520 N3 = 1
 530 GO TO 240
 540 N3 = 0
 550 GOTO 240
 560 PRINT "?";
 570 INPUT A(0),A(1),A(2),A(3),A(4),A(5),A(6),A(7),A(8)
 580 GO TO 240
 590 PRINT "?";
 595 INPUT B(0),B(1),B(2),B(3),B(4),B(5),B(6),B(7),B(8)
 600 GO TO 240
 610 PRINT "?";
 615 INPUT C(0),C(1),C(2),C(3),C(4),C(5),C(6),C(7),C(8)
 620 GO TO 240
 630 PRINT
 640 PRINT
 650 PRINT
 660 PRINT TAB(6);"WHITE","BLACK"
 670 FOR  A6 = 1 TO I
 680   IF  Y(A6) = 1  THEN 710
 690   PRINT A6;TAB(6);V$(A6),W$(A6)
 700   GO TO 720
 710   PRINT A6;TAB(6);V$(A6);"+",W$(A6)
 720 NEXT A6
 730 PRINT
 740 PRINT
 750 PRINT
 760 GO TO 240
 770 PRINT "TYPE STARTING POSITION";
 780 INPUT F,G,H
 790 PRINT
 800 I = 1
 810 F1 = F
 820 G1 = G
 830 H1 = H
 840 F2 = INT(F/10)
 850 F3 = F-10*F2
 860 G2 = INT(G/10)
 870 G3 = G-10*G2
 880 H2 = INT(H/10)
 890 H3 = H-10*H2
 900 REM  CHECK THAT POSITION IS LEGAL.
 910 GO SUB 3240
 920 IF  L = 1  THEN 950
 930 PRINT "ILLEGAL POSITION";
 940 GO TO 240
 950 REM  DECIDE ON SET OF PARAMETERS
 960 P = ABS(G2-H2)
 970 P1 = ABS(G3-H3)
 980 IF  (1-SGN(P-2))*(1-SGN(P1-2)) = 0  THEN 1250
 990 P2 = ABS(F2-H2)
1000 P3 = ABS(F3-H3)
1010 P4 = ABS(G2-H2)
1020 P5 = ABS(H3-G3)
1030 P6 = ABS(F2-G2)
1040 P7 = ABS(F3-G3)
1050 IF  ABS(P2-1)+ABS(P4-1)+ABS(P6-2) = 0  THEN 1140
1060 IF  ABS(P3-1)+ABS(P5-1)+ABS(P7-2) = 0  THEN 1180
1070 IF  ABS(G2-H2)+ABS(1-SGN(P2-2)) = 0  THEN 3620
1080 IF  ABS(G3-H3)+ABS(1-SGN(P3-2)) = 0  THEN 3650
1090 IF  (H2-1)*(H2-8)*(H3-1)*(H3-8) <> 0  THEN 1290
1100 IF  (H-11)*(H-18)*(H-81)*(H-88) = 0  THEN 1330
1110 IF  (H-12)*(H-21)*(H-71)*(H-82) = 0  THEN 1330
1120 IF  (H-17)*(H-28)*(H-87)*(H-78) = 0  THEN 1330
1130 GO TO 1290
1140 F = 10*F2+2-SGN(F3-1)
1150 IF  H3 > G3  THEN 1210
1160 F = 10*F2+7-SGN(F3-8)
1170 GO TO 1210
1180 F = F3+10*(2+SGN(1-F2))
1190 IF  G2 < H2  THEN 1210
1200 F = F3+10*(7-SGN(F2-8))
1210 F2 = INT(F/10)
1215 F1 = F
1220 F3 = F-10*F2
1230 GO TO 2580
1250 FOR  B5 = 0 TO 8
1260   X(B5) = A(B5)
1270 NEXT B5
1280 GO TO 1360
1290 FOR  B6 = 0 TO 8
1300 X(B6) = B(B6)
1310 NEXT B6
1320 GO TO 1360
1330 IF  ABS(P-2)+ABS(P1-2) = 0  THEN 1333
1331 IF  (G2-1)*(G2-8)*(G3-1)*(G3-8) = 0  THEN 1250
1332 GO TO 1339
1333 IF  (H-11)*(H-18)*(H-81)*(H-88) = 0  THEN 1339
1335 GO TO 1290
1339 FOR  B7 = 0 TO 8
1340   X(B7) = C(B7)
1350 NEXT B7
1360 REM  GENERATE ALL LEGAL MOVES
1365 M1 = M2 = 1
1370 IF  F2 = G2  THEN 1500
1380 IF  F3 = G3  THEN 1680
1390 FOR  A7 = (10*F2+1) TO (10*F2+8)
1400   IF  A7 = F  THEN 1430
1410   R(M1) = A7
1420   M1 = M1+1
1430 NEXT A7
1440 FOR  A8 = (F3+10) TO (F3+80) STEP 10
1450   IF A8 = F  THEN 1480
1460   R(M1) = A8
1470   M1 = M1+1
1480 NEXT A8
1490 GO TO 1860
1500 IF  F > G  THEN 1620
1510 FOR  A9 = (10*F2+1) TO (G-1)
1520   IF  F = A9  THEN 1550
1530   R(M1) = A9
1540   M1 = M1+1
1550 NEXT A9
1560 FOR  A0 = (F3+10) TO (F3+80) STEP 10
1570   IF  A0 = F  THEN 1600
1580   R(M1) = A0
1590   M1 = M1+1
1600 NEXT A0
1610 GO TO 1860
1620 FOR  B = (G+1) TO (10*F2+8)
1630   IF  B = F  THEN 1660
1640   R(M1) = B
1650   M1 = M1+1
1660 NEXT B
1670 GO TO 1560
1680 IF  F > G  THEN 1800
1690 FOR  B1 = (F3+10) TO (G-10) STEP 10
1700   IF  B1 = F  THEN 1730
1710   R(M1) = B1
1720   M1 = M1+1
1730 NEXT B1
1740 FOR  B2 = (10*F2+1) TO (10*F2+8)
1750   IF B2 = F  THEN 1780
1760   R(M1) = B2
1770   M1 = M1+1
1780 NEXT B2
1790 GO TO 1860
1800 FOR  B3 = (G+10) TO (F3+80) STEP 10
1810   IF  B3 = F  THEN 1840
1820   R(M1) = B3
1830   M1 = M1+1
1840 NEXT B3
1850 FOR  N9 = (10*F2+1) TO (10*F2+8)
1851   IF  N9 = F  THEN 1858
1852   R(M1) = N9
1855   M1 = M1+1
1858 NEXT N9
1860 M2 = 1
1870 FOR  B4 = 0 TO 7
1880   G = G1+Z(B4)
1890   G2 = INT(G/10)
1900   G3 = G-10*G2
1910   GO SUB 3530
```

```
1920   IF  L = 0  THEN 1950
1930   K(M2) = G
1940   M2 = M2+1
1950 NEXT B4
1960 REM  EVERY LEGAL MOVE HAS NOW BEEN GENERATED
1970 G = G1
1980 G2 = INT(G/10)
1990 G3 = G-10*G2
2000 M1 = M1-1
2010 M2 = M2-1
2020 REM  NOW SCORING OF POSITIONS CAN START
2025 I5 = 0
2030 IF  N3 =,0  THEN 2050
2040 PRINT "ROOK"
2050 FOR  B8 = 1 TO M1
2060   F = R(B8)
2070   F2 = INT(F/10)
2080   F3 = F-10*F2
2090   GO SUB 3340
2100   IF  L = 0  THEN 2170
2110   F(B8) = 0
2120   GO SUB 2750
2125   IF  E0 = 1  THEN 2742
2130   IF  N3 = 0  THEN 2150
2140   PRINT F;S
2150   D(B8) = S
2160   GO TO 2190
2170   F(B8) = 10
2180   GO TO 2120
2190 NEXT B8
2200 F = F1
2210 F2 = INT(F/10)
2220 F3 = F-10*F2
2230 IF  N3 = 0  THEN 2245
2240 PRINT "KING"
2245 I5 = 1
2250 FOR  B9 = 1 TO M2
2260   G = K(B9)
2270   G2 = INT(G/10)
2280   G3 = G-10*G2
2290   GO SUB 2750
2300   IF  N3 = 0  THEN 2320
2310   PRINT G;S
2320   E(B9) = S
2330 NEXT B9
2340 G = G1
2350 G2 = INT(G/10)
2360 G3 = G-10*G2
2370 REM  FIND MOVE WITH BEST SCORE
2380 J9 = D(1)
2390 R0 = 1
2400 FOR  B0 = 2 TO M1
2410   IF  D(B0) <= J9  THEN 2440
2420   J9 = D(B0)
2430   R0 = B0
2440 NEXT B0
2450 K9 = E(1)
2460 S0 = 1
2470 FOR  C = 2 TO M2
2480 IF  E(C) <= K9  THEN 2510
2490   K9 = E(C)
2500   S0 = C
2510 NEXT C
2515 REM  PRINT COMPUTER MOVE
2520 Y(I) = 0
2530 IF  K9 > J9  THEN 2640
2540 F = R(R0)
2545 F1 = F
2550 F2 = INT(F/10)
2560 F3 = F-10*F2
2570 IF  F(R0) = 10  THEN 2610
2580 PRINT T$(F);
2590 V$(I) = T$(F)
2600 GO TO 240
2610 PRINTT$(F);"CHECK";
2620 Y(I) = 1
2630 GO TO 2590
2640 G = K(S0)
2645 G1 = G
2650 G2 = INT(G/10)
2660 G3 = G-10*G2
2670 GO SUB 3340
2680 IF  L = 0  THEN 2720
2690 PRINT U$(G)
2700 V$(I) = U$(G)
2710 GO TO 240
2720 Y(I) = 1
2730 PRINT U$(G);"CHECK";
2740 GO TO 2700
2742 PRINT T$(F);"CHECKMATE";
2743 V$(I) = T$(F)
2744 W$(I) = "MATE"
2745 GO TO 240
2750 REM  SUBROUTINE FOR SCORING MOVES
2755 E0 = 0
2760 Q = ABS(F-H)
2770 R = ABS(F-G)
2780 IF  (Q-1)*(Q-9)*(Q-10)*(Q-11) <> 0  THEN 2820
2790 IF (R-1)*(R-9)*(R-10)*(R-11) = 0  THEN 2820
2800 S = -1000
2810 GO TO 3230
2820 S = 0
2830 FOR  C1 = 0 TO 7
2840   H = H1+Z(C1)
2850   H2 = INT(H/10)
2860   H3 = H-10*H2
2870   GO SUB 3270
2880   IF  L = 0  THEN 2900
2890   S = S+1
2900 NEXT C1
2910 H = H1
2920 H2 = INT(H/10)
2930 H3 = H-10*H2
2940 IF  S > 0  THEN 3000
2950 IF  I5 = 1  THEN 2800
2960 IF  F(B8) = 10  THEN 2980
2970 GO TO 2900
2980 E0 = 1
2990 GO TO 3230
3000 S = X(0)*(8-S)
3010 S = S+X(1)*(14-ABS(G3-H3)-ABS(G2-H2))
3020 S = S+X(2)*(14-ABS(F2-H2)-ABS(F3-H3))
3030 S = S+X(3)*(14-ABS(F2-G2)-ABS(F3-G3))
3040 IF  (F2-1)*(F2-8)*(F3-1)*(F3-8) = 0  THEN 3070
3050 S = S+X(4)
3060 GO TO 3075
3070 S = S+X(5)
3075 IF (G-32)*(G-23)*(G-26)*(G-37) = 0  THEN 3100
3076 IF (G-67)*(G-76)*(G-73)*(G-62) = 0  THEN 3100
3080 IF (G2-1)*(G2-2)*(G2-8)*(G2-7) = 0  THEN 3120
3090 IF (G3-1)*(G3-2)*(G3-8)*(G3-7) = 0  THEN 3120
3100 S = S+X(6)
3110 GO TO 3130
3120 S = S+X(7)
3130 Y0 = ABS(F3-H3)
3140 W = ABS(F2-H2)
3150 IF  Y0 >= W  THEN 3170
3160 Y0 = W
3170 X0 = ABS(F2-G2)
3180 W1 = ABS(F3-G3)
3190 IF  X0 >= W1  THEN 3210
3200 X0 = W1
3210 X0 = Y0/X0
3220 S = S+X(8)*X0*(2.5-X0)
3230 RETURN
3240 REM  SUBROUTINE FOR CHECKING LEGALITY
3250 IF  (1+SGN(F2-1))*(1-SGN(F2-8)) = 0  THEN 3470
3260 IF  (1+SGN(F3-1))*(1-SGN(F3-8)) = 0  THEN 3470
3270 IF  (1+SGN(G2-1))*(1-SGN(G2-8)) = 0  THEN 3470
3280 IF  (1+SGN(G3-1))*(1-SGN(G3-8)) = 0  THEN 3470
3290 IF  (1+SGN(H2-1))*(1-SGN(H2-8)) = 0  THEN 3470
3300 IF  (1+SGN(H3-1))*(1-SGN(H3-8)) = 0  THEN 3470
3310 IF (F-G)*(G-H)*(F-H) = 0  THEN 3470
3320 M = ABS(G-H)
3330 IF  (M-1)*(M-9)*(M-10)*(M-11) = 0  THEN 3470
3340 IF  F2 = H2  THEN 3400
3350 IF  F3 <> H3  THEN 3380
3360 IF  F3 = G3  THEN 3400
3370 GO TO 3470
3380 L = 1
3390 GO TO 3480
3400 IF  F > H  THEN 3440
3410 IF  G < F  THEN 3470
3420 IF  G > H  THEN 3470
3430 GO TO 3380
3440 IF  G > F  THEN 3470
3450 IF G < H  THEN 3470
3460 GO TO 3380
3470 L = 0
3480 RETURN
3490 DATA 1,-1,10,-10,9,-9,11,-11
3500 DATA 5,20.1,1.7,7,2,1,14,1,10
3510 DATA 9,11.5,3,11,1.1,1,6.6,1,102
3520 DATA 5,5,5.9,-2,1,16.1,2,1,50
3530 REM  SUBROUTINE FOR LEGALITY OF KING MOVES
3535 IF  F = G  THEN 3600
3540 IF  (1+SGN(G2-1))*(1-SGN(G2-8)) = 0  THEN 3600
3550 IF  (1+SGN(G3-1))*(1-SGN(G3-8)) = 0  THEN 3600
3560 M = ABS(G-H)
3570 IF  (M-1)*(M-10)*(M-9)*(M-11) = 0  THEN 3600
3580 L = 1
3590 GO TO 3610
3600 L = 0
3610 RETURN
3620 F = 10*F2+H3
3621 F2 = INT(F/10)
3622 F3 = F-10*F2
3623 F1 = F
3630 IF  (H3-8)*(H3-1) = 0  THEN 2742
3640 GO TO 2610
3650 F = 10*H2+F3
3651 F2 = INT(F/10)
3652 F3 = F-10*F2
3653 F1 = F
3660 IF  (H2-1)*(H2-8) = 0  THEN 2742
3670 GO TO 2610
9999 END
```

ERROR MESSAGE

In my article "Drawpic" in the February issue of PCW, there are some errors:

Here they are:—

Line 5000 (C61) should be (C—1)

Line 8000 a greater than sign is missing i.e.

```
:IF (F(N)+1) >127 THEN 8020
```

Line 1030 IF A$="L" THEN X=X—1: SET(X,Y):
RESET (X+1,Y):
IF Q=2 THEN SET (X+1,Y)

A. O. Ellefsen,
121 The Furlongs, Ingatestone, Essex CM4 0AL