stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 21/74 SEPTEMBER

COEN ZUIDEMA

CHESS, HOW TO PROGRAM THE EXCEPTIONS?

2e boerhaavestraat 49 amsterdam

AMS (MOS) subject classification scheme (1970): 68A45.

ABSTRACT

After nearly 25 years of chess programming it may be stated that the original claims have not been fulfilled and that the problem has been strongly underrated. In the present paper we are concerned with the bottlenecks.

A subproblem is defined and solved, with the intention of simulating the main problem - programming chess - on a convenient scale. The exceptions to general rules turn out to be mainly accountable for a poor level of play. A continuous effort to improve this level will jam in an endless stream of details.

Another serious source of troubles is formed by the way the *tree of moves* has to be *pruned*. Also on that issue, still one of the main bottlenecks, the troubles arise from the numerous exceptions.

Other aspects of chess, as pattern recognition and experience, are broadly dealt with. The most important problems from the point of view of an experienced chess player are discussed. The prospects of having a computer play master chess within a foreseeable future, look anything but favourable.

The author is an international chess master and was champion of the Netherlands in 1972.

KEYWORDS & PHRASES: *Artificial intelligence, Chess programming, Evaluation functions, Pattern recognition.*

CONTENTS

0. STRUCTURE OF THE PAPER

Chapter 1 gives both an introduction to and a motivation for our investigations. This motivation leads to a subproblem: the mating of the deprived king by king and rook. Some results are given in advance.

The basic algorithm for this endgame has been supplied in the chapters 2 and 3. In chapter 2 we sketch the main lines. In chapter 3 we go more into detail. That chapter is concluded with an evaluation of program play.

As we were not satisfied with the level of program play, a new version was made. Chapter 4 provides a general description of the changes and a discussion of a good many of the trouble spots in the primitive version. The examples given are not really shocking, but they are characteristic of the kind of difficulties one is confronted with in programming chess. Improvement of play is compared to programming effort and program length. Finally we give some examples of program play by both versions.

Chapter 5 is split up in two parts. Both are not directly important to our goals, but are linked with programming endgames. Firstly, we speak about application of our method to other elementary endgames. Secondly, related work of HUBERMAN and TAN is discussed.

In chapter 6 we come to our conclusions. We dwell on the claims that have been made since the early fifties and the present state of the art. Next, conclusions of our subproblem are made. Discussing strategy, the ideas of BOTWINNIK - to us one of the better ways of attacking the problem - are brought in. Furthermore we deal with several more aspects of chess, as pattern recognition and experience. The most important problems from the point of view of a chess player are described. These problems have scarcely been mentioned in the literature. In the final section we are not very optimistic about progress in the future.

The reader will find the program for the rook endgame in chapter 7, together with some clarifying remarks.

We have tried to enable both information scientists and chess-players to read this paper. We assume the reader is familiar with the rules of chess. Chess-players not familiar with programming are recommended not to bother too much about some technical terms.

1. INTRODUCTION

Current chess programs are, with respect to the tree of moves, based on the ideas of SHANNON, stated in 1950:

1. Use a numerical scoring function. This function includes factors such as material balance, centre control, pawn structure and open files for the rook.
2. Limit the size of the tree (pruning).

For a historical survey see D.N. LEVY [7] and P.G. RUSHTON & T.A. MARSLAND [9].

Recent publications show some criticism of these methods. The programs are extremely weak in the endgame. They are tactically sound, but lack direction of play or planning.

One of the participants in the annual ACM Computer Chess Championships is the program TECH, written by GILLOGLY. The author states that the program is a useful benchmark for other programs. This is because TECH spends at most 5 percent of its CPU-time on chess heuristics, viz. for sorting the moves in the first ply, and then scans the tree by brute force. The end-positions are evaluated with material as only criterion [5]. TECH's second prize in 1971 throws doubt upon more sophisticated methods.

There is consensus over the need of strategic play, the significance of goals and relevant plans. However, new methods are not given, (D.N. LEVY [7], B. MITTMAN [8] and P.G. RUSHTON & T.A. MARSLAND [9]). That means that after two decades of practice in this field the problem of how to program chess is basically unsolved. Moreover, we feel that the mere implementation of goals, however difficult, will not suffice. The question is whether it is possible to give numerical weights to a long list of chess heuristics and immaterial factors as mentioned above, which are meaningful under different circumstances (M.M. BOTWINNIK [1]). Related to this is the question whether implementation of general rules does not lead to blundering moves, in view of the many exceptions of the rules.

We would like to illustrate this by an existing program. However, documentation usually consists of a description in general terms. Access to the programs is hampered by their great bulk. And, how can one determine

the effect of a single parameter ? *)

To get around this, we state a very simple subproblem as a casestudy: mating the deprived king by king and rook.

It must be strongly emphasized that the solution of this problem is no goal in itself. In fact, this ending has been programmed often enough. For this reason we reject the method of exhaustive enumeration: a prescription for the situations with the black king on the eighth rank, the seventh, the sixth and the fifth, further subdivision of these cases, etc. On the contrary, the same means are used as current in chess programming, e.g. generation of moves and pruning the tree. These means are adapted to our problem.

Besides this, the program is based on a good strategy. As said before, strategy is suggested to be the next essential step to be made in chess programming. Moreover, a good strategy can be given for our subproblem far more easier than for complicated situations.

Our algorithm is of course not a perfect solution. The rules given are never complete in chess. Otherwise, there would be no problem.

Each position will be evaluated on its own merits. Most legal moves will be generated. The move that is best in the light of this strategy, is performed.

Some moves are rejected beforehand on plausible grounds (pruning). We have chosen to limit the depth of the tree to one move. This is a severe restriction, but given the simplicity of the problem, easy to overcome.

We strive after a well-playing program. The number of moves must not exceed the minimum needed for mate too far. The moves themselves must be coherent, and, as a whole, must show a direction of play.

Our final purpose is to keep track of the following:

. level of program play

. effect of new criteria which refine the algorithm

---

*) Varying the value of one parameter and fixing the others is an impossible task, because the number of parameters is too large. See also A.L. SAMUEL [10], page 602.

. exception of rules
. improvement of play in new versions versus programming effort and program length.

Because of the clarity of the problem we expect to get a better insight in where the difficulties in chess programming lie.

It will appear that a global strategy provides a solution of the problem from every starting position well within the required 50 moves. Level of play is not high.

For any rule given in the program one is able to construct exceptional situations. A more refined strategy is needed to elevate level of play. A small improvement, however, entails a great deal of expense in programming effort and program length. The new rules will have their exceptions too. Exceptions that will not even be noticed by human players.

This is the kernel of the problem. A trivial exercise as our endgame gives rise to a burden of small problems that have to be overcome. What about an overall-strategy for chess ?

ACKNOWLEDGEMENTS

I am indebted to J.W. de Bakker, D. Grune and J. Wolleswinkel for carefully reading and commenting on the manuscript. Furthermore, T.H. Gunsing who edited it, C.J. Klein Velderman-Los who did a fine typing job, and D.Zwarst, J. Suiker and J. Schipper for printing.

My special gratitude goes to A. van Wijngaarden who, as the director of the Mathematical Centre, supplied an ideal atmosphere in which it was possible to pursue and combine both my professional activities in computer programming and my interests in chess.

2. THE ROOK ENDGAME

Abbreviation

KW: the white king
KB: the black king
R : the (white) rook

Notation

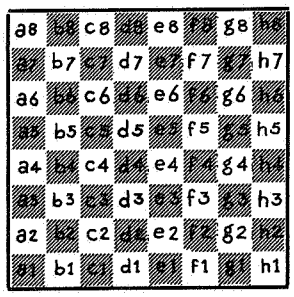

fig. 1

The algebraic notation, used everywhere except in Spanish or English speaking countries, gives the first letter of the moving piece and the square to which this piece is moved, according to fig. 1. There is no difference in notation between black and white.

Mating positions

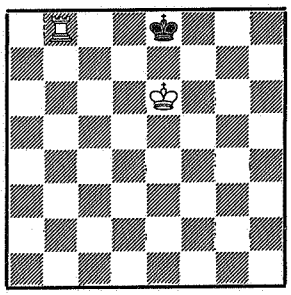

fig. 2

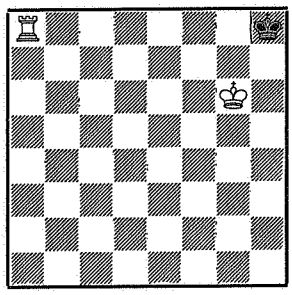

fig. 3

The computer plays for white, and has a king and a rook. Its charge is mating the black king. The starting position and the moves of black are input via a terminal by the human opponent.

In order to deliver mate KB must be driven to the edge, fig. 2 and 3. Although mating is possible with KB on any square of the edge, the corner is of vital importance, since mate cannot be forced on a board with the ranks unlimited on both sides.

KB can be driven row by row to the edge, regardless of the row where KB stands. However, this is not a fine strategy, because it does not

utilize the power of R in vertical direction.

Program scheme

A symbolic program may help to clarify the following discussion (ALGOL-like language):

```
begin initialization;
start: input position or opponents move;
       make your move;
       goto start
end
```

In the routine *make your move* the program computes its next move. A scheme follows:

```
begin transposition;
      while not last move do
      begin generate next move;
            calculate room;
            calculate measure;
            compare with candidate
      end;
      get candidate;
      inverse transposition;
      output
end
```

The algorithm

The leading idea of the algorithm is to minimize *room*, the number of free squares KB can reach in one or more moves.

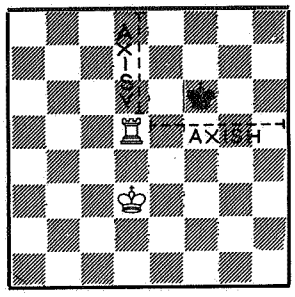

fig. 4

The rook divides the board in four quadrants. KB stands in one of them. If R cannot be driven away - such a position of R is called *strong* - KB will never escape from its quadrant. Its room is the number of squares in the quadrant:

$$room = axisv \times axish$$

In fig. 4 the position of R is strong and

$$room = 3 \times 4 = 12.$$

If R does not hold a *strong* position and KB attacks it, R can retreat over a file or over a rank, choosing the case with the smaller quadrant. Even when R is not under direct attack, *room* is calculated as if the rook were placed at the beginning of this file or rank:

$$room = 7 \times min(axisv, axish).$$

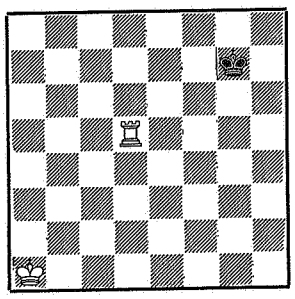

fig. 5

Example:
KB is able to attack R in two moves,
KW is far away. R may move to a5 or d1.

$$room = 7 \times min(4, 3) = 21.$$

When KW is in the same quadrant as its opponent, it controls a number of squares. This is accounted for -*tax*- and *room* is diminished accordingly.

The program computes the *room* for most allowed moves, and takes the moves which minimizes the *room*. So the depth of the tree is one. Some moves are excluded beforehand. These are moves that increase the distance between the kings, and the moves of the rook to the second, third and fourth row, or the second and third file.

When we look upon the row where KB is to be mated as the eighth row, then first row, second row etc. are also defined. For this the position

of KB is decisive. The representation in the program is simplified by an internal transposition, such that KB stands in the triangle e5-e8-h8 (see chapter 3).

Calculation of room seems superfluous if the distance between the two kings -*measure*- is large, e.g. four rows. In such a case a king move is always played, unless R is under attack.

The black king being in the middle of the board, calculation of *room* does not need to be too accurate. When the process proceeds, *room* must correspond exactly to the number of free squares that KB can reach, including its own square.

If more than one possible move yields the same minimal room, more criteria are needed. The first is a minimal *measure*. A final decision may come from the order of move generation. Therefore, this order is not arbitrary.

It is well-known that a check does not bring matters any further. It is also inconvenient from the point of view of the algorithm: It allows the defending king to choose the best of two quadrants. For this reason a check is only considered if it forces KB to give up a row.

Mate could be a logical consequence of this case if *room* is zero. The mating positions, however, are readily characterized. As a good policy, mating moves are considered before anything else.

Stalemate, on the other hand, follows directly from the relation

$$room = 1 \qquad (*)$$

because the situations in which a check is made leave the king more than one refuge ($room > 1$). A move which makes relation (*) true is rejected.

With the means described mate is not always brought about. For example, the anticipation of a stalemate may result in a repetition of moves (see fig. 14). Therefore, some specific cases are distinguished. Further, a mechanism is built in to exclude repetition of moves. History of the game is recorded in an array *history* of fifty elements (the well-known drawing limit). Moves leading to recorded positions are rejected.

We state that a program, based on these principles, solves the problem from every starting position, and does so well within fifty moves.

3. FURTHER ELABORATION. THE PRIMITIVE VERSION

Firstly we recall our task: solving a subproblem with means reflecting those of normal chess programming. We are of course free to adapt our tools to this specific problem. In fact things like representation of pieces, move generation and the algorithm itself are not apt to generalization.

Program scheme

An extension of the program scheme from the preceding chapter for the body of *make your move* is given. In this routine the program computes its next move. The discussion closely follows the scheme:

```
begin transposition;
      if not immediate move then
      begin prune;
            while not last move do
            begin generate next move;
                  calculate room;
                  compare with candidate
            end;
            get candidate
      end;
      inverse transposition;
      output
end
```

Transposition

Every position can be transposed such that the KB stays in the triangle e5-e8-h8. The distance of the king to the edge is an indication for the progress of the mating process. The main reason for the transposition is that the program knows the eighth row for the nearest edge. In most cases the king will be mated there. First row, second row etc. are also defined. For example, the rook on the first file is far away from the opposed king. This affects the order of move generation. The transposition

is not yet completely described. With the defending king on the diagonal a1-h8 its colleague can be placed in the triangle a1-h8-h1. When KW also stands on this diagonal, R is placed in the triangle a1-a8-h8. These choices are justified looking upon the range a8-h8 as the nearest edge. As a consequence, the program makes some distinctions between horizontal and vertical.

The program works move by move. Transposition and inverse transposition take place before and after computation of every move. For a person trying to understand a sequence of moves, this may give some difficulties. When, for example, KB goes from e6 to f5 or from c4 to d3, the meaning of horizontal and vertical is interchanged.

There is an interesting side effect. The array *history* records the position after transposition. Moves leading to previous positions are rejected. But these are not the only ones. All moves leading to positions that have the same transposition will be rejected as well.

Representation

As said earlier, representation of pieces is adapted to the problem. Each of the three pieces has two coordinates. It is quite natural to declare three vectors of two components each. However, in ALGOL 60, six integers are more efficient: *kwh*, *kwv*, *kbh*, *kbv*, *rh* and *rv* for the horizontal of KW, the vertical of KW etc.

Mate and mate in three

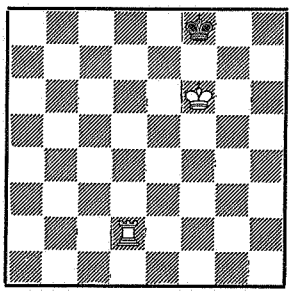

fig. 6

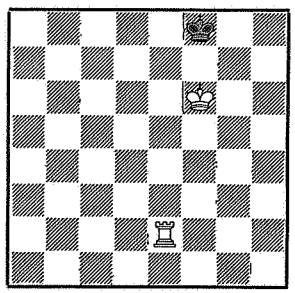

fig. 7

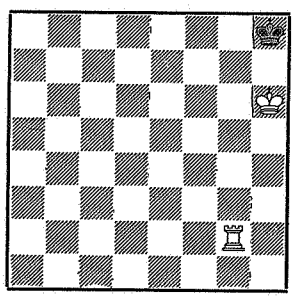

fig. 8

The mating positions are characterized as follows:

KB on the eighth rank, KW in opposition on the sixth, or on knight-jump distance if KB is in the corner, and R on minimally two files distance of KB (fig. 6: 1.Rd8 mate).

If the last condition is not fulfilled, it is mate in at most three moves. Having detected this situation all the same, the program utilizes it (boolean *mate 2*). The next move is automatically carried out:
fig. 7: 1.Re1 Kg8 2.Rh1 Kf8 3.Rh8 mate.

The exception is fig. 8: 1.Rg1 is stalemate. So 1.Rf2 Kg8 2.Rf1 Kh8 3.Rf8 mate.

This is the only situation in which information is passed on to the next move.

Measure

Measure is the distance between the two kings, computed as the sum of squares:

$$(kbh-kwh)^2 + (kbv-kwv)^2.$$

More usual is a distance of the number of king moves between two fields. This seems to be more natural. But as a chess player I feel that the distance between say f2 and b6 is larger than that between f2 and f6. One way to illustrate this is to argue that there is only one path for the king to get in four moves from f2 to b6, but several ways lead from f2 to f6. So the latter case offers a more flexible approach. This is expressed in our *measure*. A more sophisticated *measure* might have been possible, but for our problem we are not interested in the consequences of a distance of more than four lines or rows.

A *measure* smaller than three indicates an irregular situation. If *measure* > *15* (a distance of more than three rows or exactly three squares along a diagonal) there seems to be no use in calculating *room*. A king move is needed anyhow, and is played immediately. *Measure* also acts a part in pruning and as a second criterion after *room*. Out of the moves with the same *room*, the move with minimal *measure* is chosen. However, there is no reason to prefer kings in opposition (*measure* = *4*) over kings on knight-jump distance (*measure* = *5*). A value of 5 is accordingly changed in 4. (In the queen-ending it is possible to give such a preference. This leads

to the imaginary value 3).

Coding of moves

A move is put into effect by changing the coordinates of the playing piece. To be able to discuss a specific move, the boolean array *move* [*1:3,1:8*] is declared. This array contains all the legal white moves. It is initialized true. Illegal or rejected moves have the value false. Computation is only done for moves with a value true. The coordinates of the pieces are changed as soon as the move is decided upon. A set of shadow variables, *kwh1*, *kwv1* and so on, reflects the situation corresponding to a move under consideration.

An array element

move[*type,field*]

represents a move of the following type:

type = *1*, a rook move along a horizontal. *Field* indicates the new vertical coordinate,

rv1 := *field*

type = *2*, a rook move along a vertical. *Field* indicates the new horizontal coordinate,

rh1 := *field*

type = *3*, a king move. There are eight of these. The meaning of *field* is given in the diagram:

6	7	8
4	KW	5
1	2	3

diagram for the moves of the king

field = *1*, KW goes bottom left
field = *2*, KW goes bottom middle
and so on.

Some moves are ruled out:
Rook moves to their own square (*field* is the value of *rh* or *rv* respectively) and rook moves jumping over the white king are illegal. The same holds for king moves out of the board or into the range of the opponent king.

The order of moves is now fixed:

rook moves along a line, rook moves along a file, king moves. Within each type *field* steps from 1 to 8. Move order becomes important if *room* and *measure* break even.

The order of king moves is experimental. Considerations of *room* and *measure* force the attacking king in the direction of his opponent. The reverse effect of the order of moves keeps it off the edge, where it is in the way of the black king.

The order of rook moves reflects the long range of the rook. If it makes no difference for the value of *room*, the move with *field* = *1* comes first, that is, R gets far away from KB. So R has an effective retreat when it is under attack.

Pruning and Tempo

Some moves are excluded beforehand.

1. Rook moves to the second, third and fourth rank (*move*[2,2] [2,3] and [2,4]) and to the b- and c-file. Since KB is in the upper part of the board, these moves serve no purpose. The rook move to the first row (file) must remain for selection, among others as a tempo move. If black would have no obligation to move, white could not force checkmate. This demonstrates the importance of the tempo move. If, however, the rook is already on the first row (file), the move to the second row (file) must take over, and is indeed not excluded.
2. King moves that enlarge *measure* with a value greater than one. Such moves are obviously aimless.

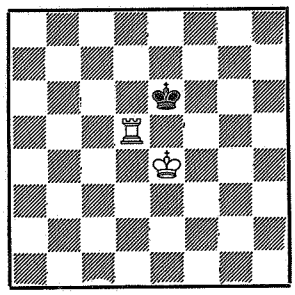

fig. 9

The eventuality of making a tempo move is not excluded by this rule.
In fig. 9 white has to make a tempo move, Kd4. *Measure* is increased by one.

Role of the king

In the procedure *room* the king plays a secondary role. The rook determines the quadrant, according to the term

axisv × *axish* (fig. 5)

Dependent of the position of KW, this term is decreased by a value, *tax*. Globally, there are four disjunct cases:

1. Check: is only considered if the kings are in opposition (*measure* = *4*): KB is driven back over one line.
2. The rook can be chased away by black (rook not *strong*). No *tax*.
3. KW is outside the quadrant. No *tax*.
4. KW is inside the quadrant. *Tax* equals the number of rows white has penetrated into the quadrant times a *factor*.

This *factor* is initialized to three, corresponding to the influence of the king on a line upon its own square, one on the left and one on the right. *Factor* is decreased by one if the king is on an edge and again by one if rook and king are on a neighbouring file.

Examples:

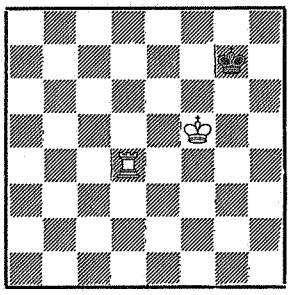

fig. 10

quadrant 4*4
tax 2*3
room 16-1 = 10

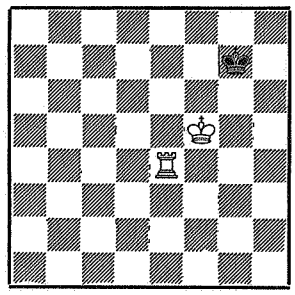

fig. 11

quadrant 3*4
tax 2*2
room 12-4 = 8

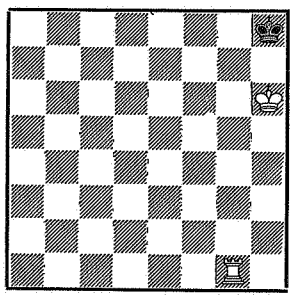

fig. 12

quadrant 7*1
tax 6*1
room = 1

An example of stalemate !

An exception is made for the case where black can leave his area in the shadow of the white king:

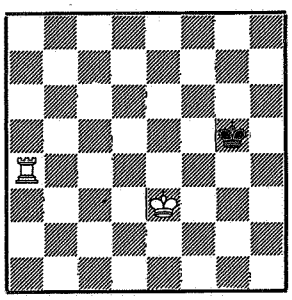

fig. 13

The move 1.Ke4 is rejected because of the answer 1...Kg4.

An exception to this exception is the already mentioned position of fig. 14.

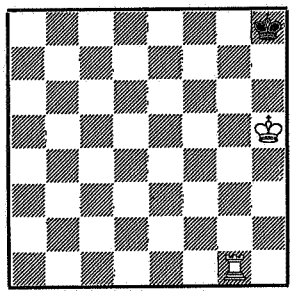

fig. 14

1.Kh6 results in stalemate. 1.Rg2 gives no no solution at all:
1.Rg2 Kh7 2.Rg1 Kh8 etc.
The move 1.Kg6, interrupting the influence of the rook over g8 gets a *room* of value 2. This is the same value as the move 1.Rg2 has, but a smaller *measure* decides in favour of Kg6.

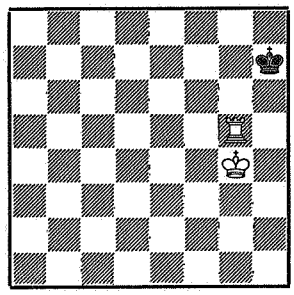

fig. 15

So the next position results in the forced variation:
1.Kh5! Kh8 2.Kg6 Kg8 3.Rf5 Kh8 4.Rf8 mate.

Evaluation

A good impression of program play is given at fig. 28 (see examples of program play).

It is not always easy to start off. A pathological case is fig. 29.

The algorithm has a limited look ahead of one move. This is a rather severe restriction. Of course the problem stated is so easy in comparison with general chess positions, that the program should be able to overcome this. Yet the program play often lacks a clear direction. *Room* is not decreased by every step. Occasionally the given criteria lead to an attempt to restore a previous position. The mechanism for excluding repetition of moves is essential to the solution of the problem.

Nevertheless, as the ultimate trend is the decrease of *room*, the process will converge to mate.

4. A NEW VERSION

The algorithm is good enough to solve the problem well within the prescribed 50 moves. Yet the moves themselves are not very impressive. There are too many rook moves, indicating that the long range effect of R is not very well employed. In several cases too many erratic moves are made before the ultimate strategy is found. It is known, theoretically, that from any starting position mate can be achieved within 18 moves. Unfortunately, an example was readily constructed for which the program needed 27 moves (fig. 29). So the algorithm had to be adjusted. This was an essential part of our case study.

Apart from a more general description, this chapter deals with a number of positions that induced specific changes. An important issue for our investigations is the effort needed for raising level of play by one class. We adhere to a tree depth of one move and do the same pruning as in the primitive version.

Room

Most of our reconstruction was applied to this procedure. The influence of the king is substantially increased. The primitive version is less accurate for central positions of KB. The choice of a move becomes indistinct and program play has no clear direction.

In the new version a number of distinctions are made on the basis of the position of KW. If KW is not inside the quadrant, the situation is not altered, no *tax*. As a consequence, R has less opportunity to penetrate between the two kings. If KW is inside the quadrant, but the position of R is not *strong*, *tax* is taken into account all the same. Allowance has been made for the weak rook position in the calculation of the quadrant (see fig. 5), and by increasing *room* by one. On the other hand, the rook can get secured in one move. Therefore, its safety must not have too many consequences for the weight of the king position.

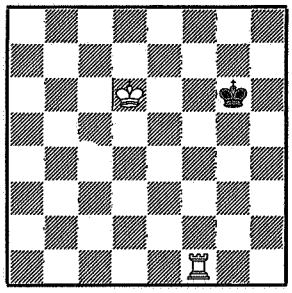

fig. 16

Both after 1.Ke7 and 1.Ke5 the quadrant is 7*2. No *tax*. However, after 1. Ke7 the rook is not strong and *room* is increased by one. This causes a natural preference to Ke5.
Unfortunately, 1.Rf2 is played. Compare to fig. 23.

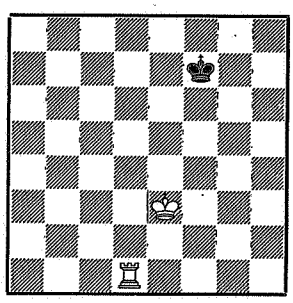

fig. 17

1.Rd6, no good, rook not *strong*.
1.Rd5 is a better move. Black can no more cross the fifth rank.
1.Kf4, another move which gives white control over the fifth rank.
Room is computed as if the rook was on d5(4*3). A smaller *measure* decides in favour of 1.Kf4.

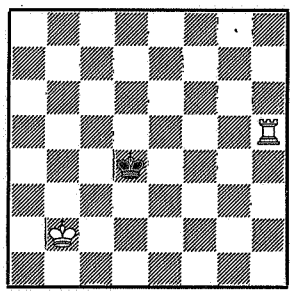

fig. 18

The primitive version plays 1.Ra5, moving R to the only *strong* square of the fifth row (...Kc4 2.Ka3). This makes no difference for the calculation of the quadrant (7*4). For the *tax* it does. If R is *strong*, *tax* is 2*3, yielding a *room* of 22.
The new version plays 1.Kb3. The rook is not *strong*, but *tax* is accounted for. *Quadrant* 7*4+1, *tax* 2*4, *room* 21.
Note that 1.Kb3 is forceful, e.g. ...Ke4 2.Kc3 or ...Kd3 2.Rh4 etc.

The algorithm was modified since a number of cases were not satisfactory. The changes, however, may have an adverse effect on other situations. The decision taken appears then to be too specific. This gives rise to more changes. These must be integrated in the framework of the algorithm.

The weight of the position of KW is not readily expressed in a number. Via a number of stages the new version has been reached. We shall not be able to discuss all of them.

Here an important issue for documentation turns up. The why and wherefore of each individual decision soon gets lost, for the programmer as well. A large number of considerations, processed in the program, is not transferable, especially those of chess-technical nature.

In the procedure *room* not less than fifteen cases are distinguished. These originate from the relation of the position of the three pieces on the board: which piece is in the middle, in horizontal and in vertical direction. If R is in one direction in the middle, then KW is outside the quadrant, no *tax*. If KB is in both directions in the middle, R is not *strong*, but *tax* is accounted for. KB may be in one direction in the middle, and KW in the other, or on the same line as R, etc. It is checked which case applies. The calculation of *room* is then readily done.

Distance rook - black king

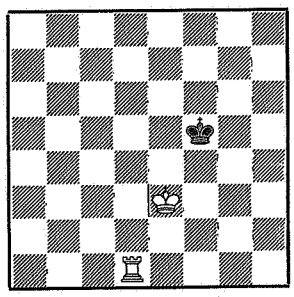

fig. 19

In the new version 1.Kf3 and 1.Rd4 get the same value of *room* (compare with fig. 17). Also *measure* is the same. The nice move is 1.Kf3; after 1.Rd4 white has to retreat a step: 1.Rd4 Ke5 2.Kd3.
The long range effect of R has already been mentioned before. A new criterion is introduced, *distance*, the distance between the black king and the rook. A maximal *distance* follows in priority after a minimal *room* and a minimal *measure*. This criterion having been programmed, the game proceeds: 1.Kf3 Ke5 2.Rd2 (tempo) Kf5 3.Re2 etc.
The last move is preferred over 3.Rd5 check, owing to this same criterion *distance*.

A well-known pattern

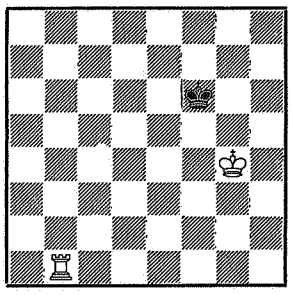

fig. 20

Often a well-known pattern pops up, which is readily programmed. Best is 1.Re1. Black must abandon a row, 1...Kf7 2.Kg5 or 1...Kg6 2.Re6†. The situation can easily be characterized (*measure* = *5*, etc.). The *room* is calculated as if R was already on e6, and is then increased by 2.
Here, *room* = *2*3+2*.

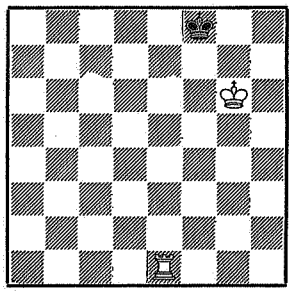

fig. 21

Fig. 21. Mate in one move: 1...Kg8 2.Re8.
room = *2*0+2* = *2* (one more than stalemate).

One may claim that the program in this pattern looks further than one move, viz. three half moves (of white, black and white).

Excluded moves

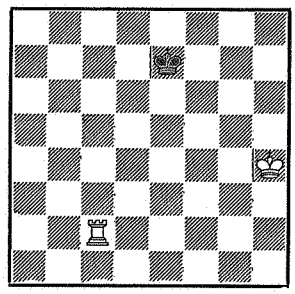

fig. 22

In the primitive version the first moves are:
1.Kg5 Ke6 2.Rc5 Kd6 3.Rb5 Kc6 4.Ra5 Kb6
5.Rd5 Kc6 6.Rd1.
Why such a despairing start?
The first two moves are okay. The rest does not seem appropriate. 3.Rf5 looks alright. Indeed, this move should have got a smaller *room*. But this move is pruned, being a move to the third file (transpose to Kb5, Rf5, Ke6, the move under consideration is then 3.Rc5, *move*[*1,3*]). We have excluded these moves, as being useless because KB is in the upper part of the board.
The point is that in this example *move*[*1,1*] is blocked by the position of the white king: 3.Rh5 is illegal in the non-transposed situation. 3.Ra5 looks better all the same, having equal values for *room* and *measure* as 3.Rb5. But now the order of moves is decisive, generating Rb5 (*move*[*1,7*]) before Ra5 (*move*[*1,8*]).

(In the new version the new criterion *distance* turns the tables). 4.Rf5 and 4.Re5 are excluded for similar reason (*moves*[2,3] and [2,4] after transposition to Ke2, Re7, Kf6). On the sixth move the rook has no more moves over the fifth row, 6.Ra5 being forbidden by history. From sheer necessity it takes the d-file, and white proceeds pretty well to check-mate.

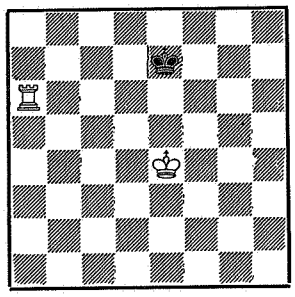

fig. 23

1.Rb6 is played.
Here an intrinsic weakness of the algorithm is shown. The necessity to minimize *room* leads to trivialities. 1.Rc6 would be better for that, but is again excluded (move to third file). 1.Kd5 would have been a good move.

Blocking the edge

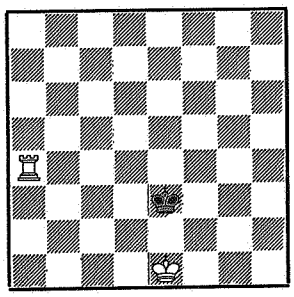

fig. 24

The attacking king is fixed by its opponent on the edge, the very place where black is to be mated. Prospects for an early mate are no good: 1.Ra3, check, spoils *room*, 1.Kd1 is followed by 1...Kd3. If it is black's turn, the ban is broken: 1...Kd3 2.Kf2 or 1...Kf3 2.Kd2.
So if it is black's turn, opposition is favourable for white, knight-jump (*measure* = 5) not. The last case gets no *tax*, the former does. In the diagram position 1.Kd1 and 1.Kf1 are rejected and a tempomove is made: 1.Rb4.

Order of king moves

The initial order of king moves aimed at keeping the white king off the edge. This care is not needed in the new version. Experimentally it appears to be better to reverse the order, as expressed in the next diagram.

4	2	1
5	KW	3
8	7	6

diagram for the
moves of the king

field = *1*, KW goes top right,
field = *2*, KW goes top middle,
and so on.

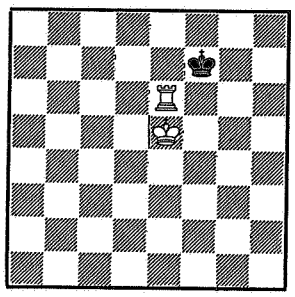

fig. 25

KW prefers the direction of the corner h8. In an earlier stage *3* and *4* were interchanged. The present order is understood, because *3* aims more at the square h8 than *4*. This is expressed in fig. 25.
1.Kf5 is better than 1.Kd6 (*room* and *measure* breaking even, the order of moves is decisive).

Blind obedience

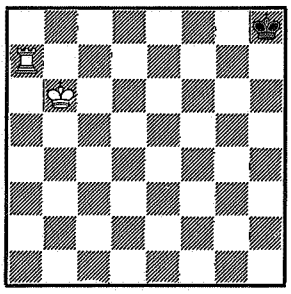

fig. 26

If *measure* > *15*, a king move is played. This move is only based on *measure*. In some instances this goes at the expense of *room*.
In the figure: 1.Kc7 Kg7 2.Kd7 Kf6 3.Ra5,
in stead of 1.Kc6 etc.
This is not altered in the new version.

Check

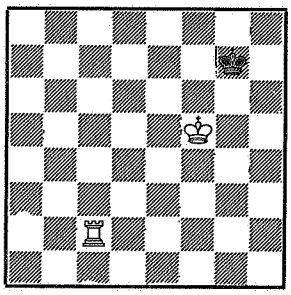

fig. 27

Being careful with checks has its drawbacks as well. The best move here is 1.Rc7 check, forcing black backwards, for 1...Kh6 2.Ra7 ! Kh5 3.Rh7 is mate.

What are the pro's and the con's of programming this pattern? It goes without saying that we appreciate any increase of program play. On the other

hand our starting point was a program calculating moves rather than going through a long list of all possible situations (exhaustive enumeration).

Brought on the level of general chess programming, what is the limit to the number of patterns a program may contain ?

Evaluation

The performance of the program has been improved in the new version. Pretty good examples are at fig. 30 and 31. There are still some starting problems, but they demand only a few moves, see fig. 32. However, many indications remain of a brain without human flexibility. Refer to fig. 23 and 26. A class-C player would do better, I guess.

The most important change was made in the procedure room. The following figures give a rough impression of the increase in program length (in lines of ALGOL-text) and in object code (in 60-bits words):

	ALGOL-text		object code	
	prim.	new	prim.	new
total program	275	390	2000	2900
I-Ø	80	80	715	715
procedure room	45	135	280	1070

A small improvement entails a great deal of effort. The conclusion forces itself that refining the algorithm and exceptions of rules give rise to an overburdened program, at least for more complicated chess problems. Note that runtime is not increased in that way. Only a part of the object code is executed for each position. The new criterion distance, however, does not ask for much program text, but, on the other hand, it takes some runtime.

Examples of programplay

primitive version

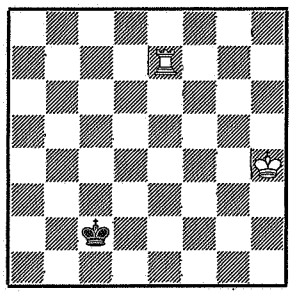

fig. 28

1.Kg3 Kd3 2.Kf4 Kd4 3.Re5 Kd3 4.Re4 Kd2
5.Re3 Kc2 6.Ke4 Kd2 7.Kd4 Rc2 8.Rd3 Kc1
9.Kc3 Kb1 10.Rd2 Kc1 11.Rd8 Kb1 12.Ra8 Kc1
13.Ra1 mate.

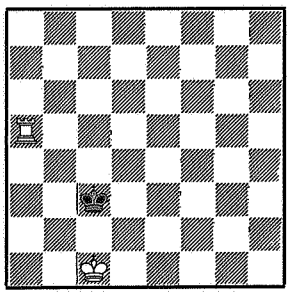

fig. 29

1.Ra4 Kb3 2.Rh4 Kc3 3.Rg4 Kd3 4.Rh4 Kc3
5.Kd1 Kd3 6.Rg4 Ke3 7.Ra4 Kd3 8.Rh4 Ke3
9.Rg4 Kd3 10.Rb4 Kc3 11.Ra4 Kd3 12.Ke1 Ke3
13.Rh4 Kf3 14.Kd2 Kg3 15.Ra4 Kf3 16.Rb4 Kf2
17.Rf4† Kg2 18.Ke3 Kg3 19.Ke4 Kg2 20.Rf3 Kh2
21.Kf4 Kg2 22.Kg4 Kh2 23.Rf2† Kg1 24.Kf3 Kh1
25.Kg3 Kg1 26.Rf8 Kh1 27.Rf1 mate.

new version

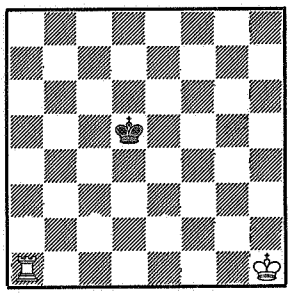

fig. 30

1.Kg2 Ke5 2.Rd1 Ke4 3.Kf2 Kf4 4.Re1 Kg4
5.Kg2 Kf4 6.Re2 Kg4 7.Rf2 Kh4 8.Rf4† Kh5
9.Kh3 Kg5 10.Rf1 Kh5 11.Rg1 Kh6 12.Kh4 Kh7
13.Kh5 Kh8 14.Kg6 Kg8 15.Rf1 Kh8 16.Rf8 mate.

Putting the rook in the starting position on a8 instead of a1, the game proceeds
(fig. 30A: Kh1, Ra8, Kd5):
1.Kg2 Ke4 2.Rd8 Ke3 3.Kf1 Kf3 4.Re8 Kf4 (or 4...Kg3 5.Rf8 Kh3 6.Rg8 etc.) 5.Kg2 Kf5 6.Kg3 Kf6 7.Kg4 Kf7 8.Re5 Kf6 9.Re1 Kg6 (or 9...Kf7 10.Kg5 etc.) 10.Rf1 Kh6 11.Rf6† Kh7 etc. (17 moves).

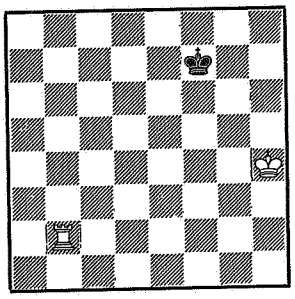

fig. 31

1.Re2 Kf6 2.Kg4 Kg6 3.Rf2 Kh6 4.Rf6† Kg7
5. Kg5 Kh7 6.Rf7† Kh8 7.Kh6 Kg8 8.Rf1 Kh8
9. Rf8 mate.

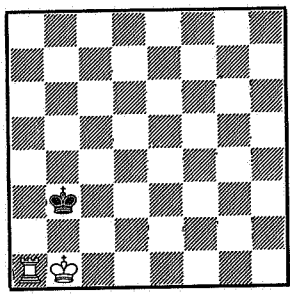

fig. 32

1.Ra5 Kb4 2.Rd5 Kc4 3.Rd2 Kb3 4.Rc2 Ka3
5.Rb2 Ka4 6.Ka2 etc. (12 moves).

5. EXTENSION OF OUR METHOD. OTHER PROGRAMS

Not important to our goals but still interesting is the question whether our method can be applied to the other elementary endgames, king and queen, king and two bishops, king and bishop and knight. The queen ending has been programmed, the other endgames have not. The chapter is concluded with a discussion of the endgames of HUBERMAN [6] and TAN [11].

The queen ending

This ending is much easier to program than king and rook. Therefore, we give only a short description of the algorithm.

The queen is more powerful than the rook. The program has to exploit this. When KW is positioned at one side of KB and the queen at the other side, black has a narrow path in which he can walk. If the queen follows, proceeding from the center, black is easily driven up and checkmated. The narrow path KB has available implies that black has only a few moves at his disposal. This is exploited in the program. The principle idea is to minimize the number of legal moves black has. This number is computed for every white move. The move leaving a minimal choice to black is decided upon. This leads automatically to the desired situation: the white king at one side and the queen at the other side of the black monarch. A minor criterion gives preference to central queen positions over positions near the edge. So the queen does not block the edge.

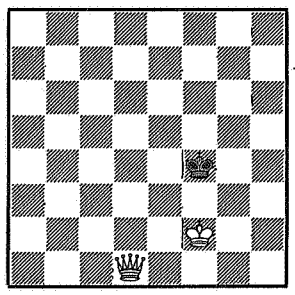

fig. 33

A fine example is shown by fig. 33.
Mate is reached in four moves, the depth of the tree being one !
1.Qd5 Kg4 2.Qe5 Kh4 (...Kh3 3.Qh5 mate)
3.Kf3 Kh3 4.Qh5 mate.

Special cases, as in the rook endgame, are exceptional. Yet the array history is of vital importance. Some problems arise because of the many stalemate positions. However, we may conclude that this ending is not appropriate for a case study. It is just too simple.

Two Bishops

This ending can be programmed with the same means as the rook ending. The notion of room is essentially the same. Here the two bishops make up the area. The program has to surmount more problems. First, coordination of three pieces is more difficult than coordination of two. Secondly, the mating proces takes more moves, and mate can only be forced in the corner. Thirdly, when black is driven into the edge, things do not go off of their own accord. The numerous stalemate positions hinder a straightforward strategy. Tempo moves are a natural outcome. In fact, the anticipation of stalemate asks for a tree depth greater than one.

Bishop and Knight

This is an order of magnitude more difficult than all the previous endgames. I conjecture that the best way to program it is using the method given by CHÉRON for human players [2]. The problem is split up in four stages by means of three triangles. They border the area to which the black king is confined successively.

Assume the bishop occupies the white coloured squares.

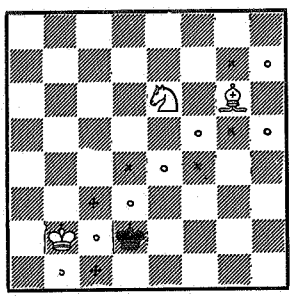

fig. 34

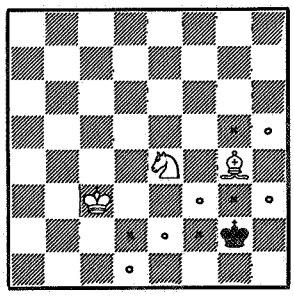

fig. 35

The large triangle, b1-h7-h1, is seen in fig. 34. Black never comes out because the neighbouring black squares are controlled by knight and king. Especially the knight, occupying a square of the same colour as the bishop, can be of great help. The medium triangle, d1-h5-h1, given in fig. 35, is made up by bishop and knight alone. The small triangle is formed by f1-h3-h1.

The first step is forcing black into the large triangle. Then KW will

drive KB backwards, so that white can take up a smaller triangle. The bishop may support its king along the hypothenuse. Eventually confined to the small triangle, black is mated in a few moves. CHÉRON demonstrates succesfully that each separate step is readily made.

In programming, a notion of *room* may be supplied for all stages. However, the immediate goal is switching to the next stage. Here a map for moving the knight to an arbitrary selected square can be put to good use. This map is introduced by BOTWINNIK (next chapter). The transition to this next stage is more easily computed by inspection of the map than by just scanning the tree for a number of single moves. The depth of the tree should of course be greater than one.

The endgames of HUBERMAN

As mentioned before the endgame of king and rook has often been programmed. Most chess programs are able to mate with rook and king. It is more remarkable that so many are not. However, the endgame is rarely documented. A reason could be that this is only interesting if the program serves a special purpose.

This applies for the study of HUBERMAN [6], one out of the two documents about programs of chess endgames I know of. The study is concerned with the process of translating book descriptions of problem solving methods into program heuristics. The chess endgames provide a good area for this research. The book method used is that of REUBEN FINE [4].

Two functions, *better* and *worse* , are supplied to compare positions. The program will search the tree for positions that are *better* than the starting position. The tree is pruned at positions that are *worse* than the starting position. The functions *better* and *worse* are built up out of information derived from the description in the book.
An example of program play demonstrates that the program closely follows an example given by the book. An informal proof of program correctness shows that the program reaches mate from every starting position.
It would be still interesting to know how the program would behave in some

special situations. Unfortunately, no more examples of program play are given.

Three endgames are dealt with by this method: king and rook, king and two bishops, and king and bishop and knight. The two-bishops ending is rather laborious, but essentially not more difficult than king and rook. The strength of the method is demonstrated by coping with king and bishop and knight, indeed one of the most difficult endings.

From the point of view of chessprograms it is a disadvantage that FINE is not concerned with the best method, but only with a simple one. E.g. rather curiously a rule is supplied in the program, which is nearly the opposite of a rule in our method. This rule prefers positions in which the *distance* between the white king and its rook is minimal (here *distance* equals the minimum numbers of king moves). By this rule the rook is hampered in its movements. Compare to our rule for maximizing *distance* between the black king and the rook (fig. 19).
A combination of both didactis and speed is given by EUWE [3].

Comparing our rook ending with the program of HUBERMAN the way of pruning the tree is worth mentioning. HUBERMAN splits up the program in stages, according to the method of the book. The program computes the transition to the next stage. If the tree becomes too long, additions are made to the functions *better* and *worse*. *Better* is responsible for a smaller tree-depth. Maximum tree-depth is five double moves (for white and black). Because of this length, the width of the tree must be reduced considerably: this is the responsability of *worse*.

In our program the width of the tree is not so important. So we could restrict ourselves to pruning some plausibly weak moves. That our pruning is not too bad may also be deduced from the comparison of program play. We think our program better, that is to say it is reaching checkmate in fewer moves.

Of course the two programs serve a different purpose. The comparison, therefore, has no great significance.

The endgames of TAN

When the endgames with a queen, a rook, two bishops and bishop and knight have been programmed succesfully, the attention may be turned to endings with a single pawn. This endings are investigated by Dr.S.T.TAN [11].

His main purpose, however, is the representation, organization and use of knowledge, and the program supplies a specific form of representation of knowledge. As such, it does not give us new starting points for discussion. But it does strengthen our opinion that basic work like these endings are worth programming, in order to get a better perception of what chess programming really is.

6. REFLECTIONS OF A CHESS PLAYER

Introduction

Chess is a game of exploiting the coincidences. A rule merely serves as a guideline. Experienced players know to use it at the right moment. Here weak or inexperienced players will fail.

This obvious observation has not discouraged the workers in the field of chess programming, who, it should be put clearly, have done invaluable work. Nevertheless, it is the main cause of troubles in nearly all aspects of the field. Therefore, it seems incredible that this point hardly has got any attention in the literature.

There would be less reason for surpise if chess programmers only would have the intention to have the machine play chess on a level not much higher than that of the average player. But their goal is just the opposite. There is no discussion about expert level*) (class-A, a high level indeed), even not about master level, but the claim is to beat the world champion. This claim, stated in the early fifties, has not vanished. A recent example is found in the title of Prof. MITTMAN's article: *Can a Computer Beat Bobby Fisher?* [8].

Firstly, this clearly indicates that the difference in strength between an expert and a master, subtle and hard to describe, is not at all understood, let alone the elusive difference between master and grand-master level. The result of twenty years programming is that any expert can easily beat a computer. This is not a depreciation of the work done. It only shows that the expectations have been put too high.

Various aspects of chess programming will be discussed in the rest of this chapter. Some of them have been built in into existing programs, some have not. The topics chosen cover a larger field than is strictly justified by the scope of our subproblem, the rook endgame. For the sake of clarity, note that the results of this will be applied only to the topics of pruning and strategy. As for the other topics, I did not want to let pass the opportunity to discuss them. There has always been great scepticism

*) technical term.

about programming chess amongst strong chess players. However, their opinions are rarely voiced - nor solicited, for that matter.

Pruning and Evaluating

The first thing in programming chess is limiting the size of the tree. This can be done by rejecting the non-plausible moves. One can only hope that the best moves are not thrown out in this way.

A different method starts by determining the state the program is in. Then a list of possible goals is derived from that state, and moves are generated accordingly. This method is better, but it does not get around the problem: Will not the best moves be thrown away ? How to program the states? What to do with the exceptions ? If there is a class of exceptions, what to do with the exceptions to that class ? How to choose, in general, goals according to a given state ? How to decide, in general, which moves are conducive to these goals and which are not ? Again, what about the exceptions ?

The same holds for the components of the evaluation function, the value of open lines, attacking chances, passed pawns, double pawns etc. Chess heuristics may seem plausible in many situations, however, there is just no reason to assume that this is the road to master level.

Here a recently made remark of TAN applies. Referring to evaluation functions in the SHANNON-TURING-way he states that chess programs in the now traditional sense do not seem to have any interest from the point of view of artificial intelligence any more [11].

Strategy

As said in the introduction, there is consensus over the need for strategy play. The problems involved are similar to those above. How to describe a strategy in a program ? What strategy for what class of positions ?

Our rook endgame has a start at two points.

1. The problem can be solved, it is evident that white wins, and

it is anything but complicated.

2. A fairly good strategy is given.

All the same, numerous small problems pop up. Improving the play entails a burden of programtext.

So I have come to the conclusion that, given a reasonable strategy in a complicated game like chess, a continuous attempt to improve level of play will jam in proliferating details, long before master level will have been reached.

The strategy of BOTWINNIK

BOTWINNIK's strategy, developed in his book *Computers, Chess and Long Range Planning* [1], is based on two principles.

First: The goal for both sides in chess is material gain. This results in attack ("assertion") and defence ("negation"), and in preventing the defense ("negation of negation").*)

Secondly: The problem has to be limited. Therefore, a horizon must be established, and the program will only deal with those attacks that fall within the horizon. If there are many attacks, the horizon lies nearby. If there are only few, the horizon recedes.

As a consequence, we may expect a deeper and more straightforward analysis in the endgame. Pawn promotion is of course a form of material gain. As for elementary endings, if gain of space is programmed as a form of material gain, there is no need for separately programming this endings.**)

For each attack there must be an attacking piece, an object of attack and an attacking path. Regarding the squares of this path, BOTWINNIK makes a distinction between

1) α-squares, the squares on which the attacking pieces come to rest
and
2) β-squares, the squares over which pieces pass.

(E.g., there are no β-squares for the moves of a knight).

The attacking path must be safe, that means that α-squares have to be under control of the attacking side. Here other pieces can give help.

---

*) In the sequel we shall only speak about attack.

**) BOTWINNIK does not directly hint at this. But he points out that the part of the theory covering positional play is not finished and must be refined by a good many experiments. The gain of space may be seen as an example.

It is very interesting that BOTWINNIK is able to distinguish between active and passive style within this framework. When choosing between making unsafe the α-squares of the other side or reinforcing his own attacking path, the passive player prefers the former approach, and the active player the latter.

This concept may also shed light on the way BOTWINNIK plays chess.

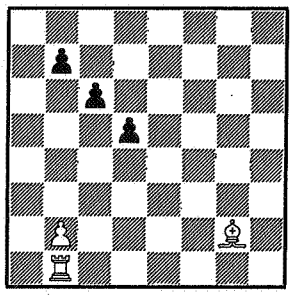

fig. 36

fig. 36:
White plays b4 and b5, undermining black's pawn chain, possibly followed by Rc1 (pattern).

Putting the bishop on the long diagonal and striving to break open an opponent's pawn chain at the end of the diagonal is only a trivial example of this strategy.

The representation of pieces and generation of moves is a logical consequence of this scheme for BOTWINNIK. For each piece a coding table is constructed, containing, for every square on the board, the minimal number of moves in which this piece can move to that square. It goes without saying that this technique holds a great advantage over a step generation of moves: "It is the way the master sees the board".

It is remarkable that the strength of this approach is not fully recognized. RUSHTON and MARSLAND even assert that this method does not give an indication of the direction of the game and thus will fail [9]. The play for annihilation of pieces, however, is evidently a strategy. Any move that does not contribute to this within a given horizon is rejected. When an attack has been decided upon, pieces are coordinated for that attack. This results in a very effective pruning of the tree. I would even venture to say that a better indication of the direction of the game has not yet been found.

In my opinion, pruning is still the main problem of chess programming. Programs based on the principles of BOTWINNIK will make significant progress in this area. But, nevertheless, master level will not be reached.

In this approach the danger of pruning the best move has been reduced considerably, but not completely. A choice for the best attack will not always be easy. What about attacking a square (instead of: a piece) ? Then, at each move again, a choice must be made between all possible attacks. This is what is criticized as: no direction of the game. It is the weak point of chess programming as a whole: In chess one has to change his lines of attack continuously.

Learning and advice taking

The idea of self teaching programs seems very promising. Such programs learn from their own games as well as from book games. But it must first be programmed before the problems will turn up. How can a program decide which was the bad move, causing the loss of the game ? What are the consequences if a self-teaching program fails to realize that a won-position was only lost by a blunder ?

For the game of checkers classic work was done by SAMUEL [10]. The problem of getting the program to generate its own parameters is still unsolved. A fundamental problem is lack of time, as in nearly all aspects of programming a game. A basic question, concerning both learning and advice taking, will be discussed later on: How to apply a lesson to other situations, as a human is able to do (not always succesfully!) ?

An advice taking program has been written by ZOBRIST and CARLSON. In the Scientific American they discuss the problem whether chess ideas can be expressed in words, either in common language or in special purpose language [12]. Until such a special purpose language will be defined, one can but guess.

But their prospect that in their approach players like Fisher could record their chess techniques for posterity seems to me an idea characteristic of a chess amateur. Fisher's technique is recorded in his games. There is no better way.

Pattern Recognition

A new prospect is provided by pattern recognition. Certainly, implementation of patterns will be helpful, for example the fork of fig. 37 and the notorious mate of fig. 38.

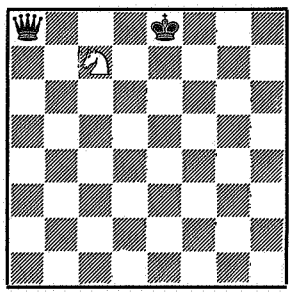

fig. 37

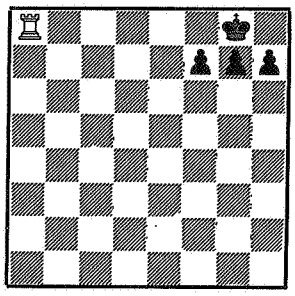

fig. 38

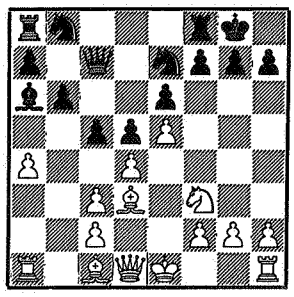

fig. 39

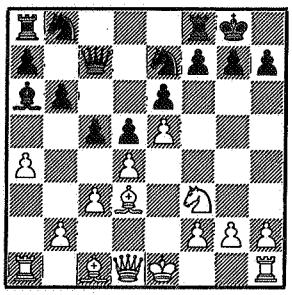

fig. 40

More complex patterns reveal some problems. Every expert knows the bishop sacrifice of fig. 39: 1.B×h7† K×h7 2.Ng5† Kg8 3.Qh5, threatening mate on h7. Branches are at move 2...Kg6 and at move 3...Rd8, but in this case the attack is winning. A useful pattern indeed.
The primary constituants of the pattern are,
for black: the king and three pawns: Kg8, pawns f7, g7, h7;
for white: Bd3, Nf3 and Qd1. More precisely, for white a bishop aiming at h7, a knight aiming at g5, and a queen aiming at h5 - possibly interrupted by a knight on f3. (Again, note the relevance of the maps of BOTWINNIK).

However, replacing the pawn on c2 by a pawn on b2, makes the sacrifice incorrect, fig. 40: 1.B×h7† K×h7 2.Ng5† Kg8 3.Qh5 Bd3, covering square h7. We conclude that this type of pattern must not be used automatically. It

only suggests a series of moves that has to be checked separately for each instance of the pattern.

The differences rather than the similarities count in a third form of patterns. Suppose a player is confronted with a new move in a well-known opening, a so called novelty. How does he proceed ?
There is a well-known pattern on the board, viz. the position deriving from the usual move except at most two pieces occupying different squares. Suppose the direction of play and the features of the old position are known. Then our player has to look for the differences in the two positions, trying to indicate a drawback of the new move. Therefore, a combination of knowledge, experience and creativity is required. Omitting the subjects knowledge and creativity, we come to our final topic.

Experience

Suppose two players are analysing a chess position, discussing moves, counter moves, starting again etc. At some moment the more experienced player bruskly interrupts his younger colleague, puts the king on g2 and declares: "In this sort of positions the king should stand on g2, not on g1". He cannot explain why. He is also not able to sketch what is meant by "this sort of positions". There is no typical pattern. He is even not willing to discuss it: He *knows*.
In the best case the younger player has some chance to grasp his meaning. Perhaps he believes it is true. He applies it in some games, say one time not in the right situation, the other times with more success. Now he *knows* too. But, this sort of positions not turning up frequently, some years have passed.
I cannot imagine this younger player being a computer.

Look ahead

Measured by the world champion, current chess programs have to go a long, long way. Issues like strategy, learning, pattern recognition and experience are scarcely well-thought out. The ideas of BOTWINNIK may improve standard of play, but the theory is not finished and the whole

system is waiting embarrassingly for implementation. In this situation the question arises: *Can a Computer Beat Bobby Fisher?*
Prof. MITTMAN states: *Many computer scientists would answer this question with: "May be". However, not many of them are willing to answer the next logical question: "When"?* [8].
In may opinion, FISHER and KARPOV (curiously enough a disciple of this same BOTWINNIK) never will have to fear any real danger from the side of chess programs during their lives. A significant step would be made if the level of an expert would be achieved in the next ten years.

7. THE PROGRAM

The program for the rook endgame has been written in ALGOL 60. A justification of this choice goes by saying that clarity of program was far more important than efficiency.

The program has been designed for a display terminal of a CYBER-machine of Control Data. The I-Ø procedures of Control Data-ALGOL are based on the Knuth-proposal. I-Ø is not used before line 310. Input is given over channel 60, the standard input channel, which should be connected with the terminal. Output is given over channel 61 and channel 1. Channel 61 is the standard output channel, which should also be connected with the terminal. Channel 1 is connected with a file that the user has to define. This file will contain a listing of the games played.

To facilitate the understanding of the program, the symbolic program schemes of chapter 4 are repeated. First the program scheme:

```
begin initialization;
start: input position or opponents move;
       make your move;
       goto start
end
```

input position or opponents move (symbolic for lines 361-387) is complicated by the necessity for dealing with moves or positions that are not correct. According to the preceding input-situation, the program has to branch to *next move* or to *next game* (see *procedure amiss*, line 349).

In the scheme for *make your move* here below, each line is provided with at most four numbers, which connect a symbolic line with the affected program lines.

begin transposition;	201-206, 14-23
if not immediate move then	208-258
begin prune;	259-279
while not last move do	282,283
begin generate next move;	284-289, 43-62
calculate room;	290, 63-199
compare with candidate	291-301
end;	302
get candidate	303-308
end;	
inverse transposition;	309, 14-29
output	310-322
end	323

```
        ALGOL-60    VERSION 4.0        LEVEL 0013

 1    ROOK ENDING , VERSION 2 . COEN ZUIDEMA.                                    1
 2    "BEGIN" "INTEGER" KWV,KWH,RV,RH,KBV,KBH,KBMINKV,KBMINKH,                   2
 3      KBMINRV,KBMINRH,KWMINRV,KWMINRH,MEASURE,A,B,C,D,KBV1,KBH1,               3
 4      COUNT,POS,PRESENT;                                                       4
 5    "INTEGER""ARRAY" HIST[1:50];                                               5
 6    "BOOLEAN"COUNTERM,MATE,MATE2,DANGER,STRONG,CHECK;                          6
 7                                                                               7
 8    "PROCEDURE" MAKE YOUR MOVE;                                                8
 9    "BEGIN" "INTEGER" KWV1,KWH1,RV1,RH1,AXIS,FIELD,TYPE,FIELDM,TYPEM,MAX1,     9
10      MAX2,KBMINKV1,KBMINKH1,KBMINRV1,KBMINRH1,KWMINRV1,KWMINRH1,MEASURE1,     10
11      DISTANCE,MIN;                                                            11
12      "BOOLEAN" "ARRAY" MOVE[1:3,1:8];                                         12
13                                                                               13
14      "PROCEDURE"TRANSPOSE (AXIS);"VALUE" AXIS;"INTEGER" AXIS;                 14
15      "BEGIN" "SWITCH" S:= HOR,VERT,DIAG;                                      15
16        "GOTO" S[AXIS];                                                        16
17      HOR: KWH:=9-KWH; KBH:=9-KBH;RH:=9-RH;     "GOTO" END;                    17
18      VERT:KWV:=9-KWV; KBV:=9-KBV;RV:=9-RV;     "GOTO" END;                    18
19      DIAG:A:=KWH; KWH:=KWV; KWV:=A;                                           19
20           A:=KBH; KBH:=KBV; KBV:=A;                                           20
21           A:=RH ; RH:= RV ; RV :=A;                                           21
22      END:                                                                     22
23      "END";                                                                   23
24                                                                               24
25      "PROCEDURE" REVERSE;                                                     25
26      "BEGIN" "IF"AXIS>3 "THEN""BEGIN"AXIS:=AXIS-4;TRANSPOSE(3)"END";          26
27              "IF"AXIS>1 "THEN""BEGIN"AXIS:=AXIS-2;TRANSPOSE(2)"END";          27
28              "IF"AXIS>0 "THEN" TRANSPOSE(1)                                   28
29      "END";                                                                   29
30                                                                               30
31      "BOOLEAN" "PROCEDURE" IRREGULAR;                                         31
32      IRREGULAR:=MEASURE<4                                                     32
33        "OR" KWMINRH=0 "AND" KWMINRV=0                                         33
34        "OR"KBMINRV=0"AND""NOT"(KWMINRV=0"AND"SIGN(KBMINKH)=SIGN(KWMINRH))    34
35        "OR"KBMINRH=0"AND""NOT"(KWMINRH=0"AND"SIGN(KBMINKV)=SIGN(KWMINRV));   35
36                                                                               36
37      "BOOLEAN" "PROCEDURE" QUESTION (CHECK); "BOOLEAN"CHECK;                  37
38        QUESTION := CHECK:= KBMINRV1=0 "AND""NOT" (KWMINRV1=0 "AND"            38
39                                          SIGN(KBMINKH1)=SIGN(KWMINRH1))       39
40              "OR" KBMINRH1=0 "AND""NOT" (KWMINRH1=0 "AND"                     40
41                                          SIGN(KBMINKV1)=SIGN(KWMINRV1));      41
42                                                                               42
43      "PROCEDURE" GENERATE;                                                    43
44      "BEGIN" KWV1:=KWV; KWH1:=KWH; RV1:=RV; RH1:=RH;                          44
45        "IF"TYPE=3 "THEN"                                                      45
46        "BEGIN"                                                                46
47          "IF"FIELD=1"OR"FIELD=3"OR"FIELD=6"THEN" KWV1:=KWV+1 "ELSE"           47
48          "IF"FIELD=4"OR"FIELD=5"OR"FIELD=8"THEN" KWV1:=KWV-1;                 48
49          "IF"FIELD<3"OR"FIELD=4 "THEN" KWH1:=KWH+1 "ELSE"                     49
50          "IF"FIELD>5 "THEN" KWH1:=KWH-1;                                      50
51          KBMINKV1:=KBV-KWV1; KBMINKH1:=KBH-KWH1;                              51
52          MEASURE1 := KBMINKV1**2 + KBMINKH1**2;                               52
53          KBMINRV1:= KBMINRV; KBMINRH1:= KBMINRH                               53
54        "END" TYPE=3 "ELSE"                                                    54
55        "BEGIN" "IF" TYPE=1"THEN"RH1:=FIELD "ELSE"RV1:=FIELD;                  55
56          KBMINKV1:=KBMINKV; KBMINKH1:=KBMINKH;                                56
57          MEASURE1:=MEASURE;                                                   57
58          KBMINRV1:=KBV-RV1;KBMINRH1:=KBH-RH1                                  58
59        "END" TYPE ;                                                           59
60        KWMINRV1:=KWV1-RV1; KWMINRH1:=KWH1-RH1                                 60
61      "END" GENERATE ;                                                         61
62                                                                               62
63      "INTEGER" "PROCEDURE" ROOM;                                              63
```

```
ALGOL-60   VERSION 4.0       LEVEL 0013

"BEGIN" "INTEGER" RANGEV,RANGEH,TAX; "BOOLEAN"VERT;

"PROCEDURE" A1;"COMMENT" CHECK: IS ONLY GOOD, IF KB MUST GIVE UP
        A ROW;
    "IF" KBMINRV1=0 "THEN"
    "BEGIN" "IF" MEASURE1=4 "AND" KBMINKH1=0
      "OR" MEASURE1=5 "AND" KBH=8"AND" KBMINKH1=1 "THEN"
      "BEGIN" RANGEV:="IF"KBMINKV1>0"THEN"8-RV1"ELSE"RV1-1;
        "IF" RANGEH>5 "THEN" TAX:=RANGEH-5
      "END" "ELSE" TAX := - 20
    "END" VERTICAL CHECK "ELSE"
    "IF" MEASURE1=4 "AND" KBMINKV1=0 "THEN"
    "BEGIN" RANGEH:="IF"KBMINKH1>0"THEN"8-RH1"ELSE"RH1-1;
      "IF" RANGEV>5 "THEN" RANGEV:=5
    "END" "ELSE" TAX := - 20;
"PROCEDURE" A2;"COMMENT" KW IN THE MIDDLE,IN BOTH DIRECTIONS.
    ROOK IS STRONG;
    TAX:=(ABS(KWMINRV1)+1) * (ABS(KWMINRH1)+1);
"PROCEDURE" A3;"COMMENT" KW VERT.IN THE MIDDLE, HOR.EQUAL WITH R .
    EXCLUDE THE CASE THAT KB WALKS BEHIND KW: ABS(KBMINRV)=1 ;
    TAX :="IF" ABS(KBMINRH1)=1"THEN"-20"ELSE"ABS(KWMINRV1)+1;
"PROCEDURE" A4;"COMMENT" COMPLEMENT OF A3. A SPECIAL CASE IS
    KG6,R ON G-FILE,KH8 ;
    TAX := "IF" ABS(KBMINRV1)=1 "THEN"
           ("IF"KBV=8"AND"KWH1=6"THEN" RANGEH-2 "ELSE" -20)
          "ELSE" ABS(KWMINRH1)+1;
"PROCEDURE" A5;"COMMENT" KW VERT. IN THE MIDDLE (OR EQUAL WITH ROOK),
    HOR.EQUAL WITH KB.ROOK IS STRONG;
    RANGEV:=RANGEV - ABS(KWMINRV1) - 1;
"PROCEDURE" A6;"COMMENT" COMPLEMENT OF A5 ;
    RANGEH := RANGEH - ABS(KWMINRH1) - 1;
"PROCEDURE" A7;"COMMENT" KW VERT.IN THE MIDDLE,KB HOR. ROOK MAY
    BE NOT STRONG. IF KW CAN BE CUT OF,TAX IS ZERO.THE IDEAL
    SITUATION: KB IS ON KNIGHT JUMP DISTANCE FROM KW AND HAS TO
    GIVE UP A LINE, GETS EXTRA TAX. A GENERAL CASE REMAINS ;
    "IF" MEASURE1 = 5 "THEN"
    "BEGIN" "IF" "NOT"VERT "AND"KBMINKH1=-2 "THEN" TAX:=0 "ELSE"
      "IF" ABS(KWMINRH1)=2 "THEN" TAX:=ABS(KBMINRV1)*RANGEH-2
      "ELSE" A7A8
    "END" "ELSE" A7A8;
"PROCEDURE" A7A8;"IF"VERT"THEN"RANGEH:=
                          ("IF"KBMINKH1<0"THEN"KWH1-1"ELSE"8-KWH1)
                        + ("IF"ABS(KBMINKH1)=1"THEN"1"ELSE"0)
            "ELSE"RANGEV:=("IF"KBMINKV1<0"THEN"KWV1-1"ELSE"8-KWV1)
                        + ("IF"ABS(KBMINKV1)=1"THEN"1"ELSE"0) ;
"PROCEDURE" A8;"COMMENT" COMPLEMENT OF A7 ;
    "IF" MEASURE1=5 "THEN"
    "BEGIN" "IF" VERT "AND" KBMINKV1=-2 "THEN" TAX:=0 "ELSE"
      "IF" ABS(KWMINRV1)=2 "THEN" TAX:=ABS(KBMINRH1)*RANGEV-2
      "ELSE" A7A8
    "END" "ELSE" A7A8 ;
"PROCEDURE" A9;"COMMENT" KB IN BOTH DIRECTIONS IN THE MIDDLE:A
    COMBINATION OF A7 AND A8.ROOK NOT STRONG.TAKE APART TWO CASES;
    "IF" MEASURE1=5 "THEN"
    "BEGIN" "IF" ("IF"VERT"THEN"KBMINKV1=-2"ELSE"KBMINKH1=-2)
             "THEN" TAX := 0 "ELSE"
      "IF" ABS(KWMINRH1)=2"THEN"
      "BEGIN" RANGEV:="IF"KBMINKV1<0 "THEN" KWV1-3 "ELSE" 6-KWV1;
              TAX := - 2
      "END" "ELSE" "IF" ABS(KWMINRV1)=2 "THEN"
      "BEGIN" RANGEH:="IF" KBMINKH1<0 "THEN" KWH1-3 "ELSE" 6-KWH1;
              TAX := - 2
      "END" "ELSE" A7A8
```

```
ALGOL-60   VERSION 4.0       LEVEL 0013

127       "END" "ELSE" A7A8;                                                    127
128     "PROCEDURE" A10;"COMMENT" KB VERT.IN THE MIDDLE,HOR.EQUAL WITH KW.ROOK  128
129       NOT STRONG. NOTE THAT KBH>4( BY TRANSPOSITION);                       129
130       "IF" VERT "THEN" RANGEH:=KBH "ELSE"                                   130
131       RANGEV := "IF"KBMINKV1>0"THEN"7-KWV1"ELSE"KWV1-2;                     131
132     "PROCEDURE" A11;"COMMENT" COMPLEMENT OF A10;                            132
133       "IF" "NOT" VERT "THEN" RANGEV := KBH "ELSE"                           133
134       RANGEH := "IF" KBMINKH1>0 "THEN"7-KWH1"ELSE"KWH1-2;                   134
135     "PROCEDURE" A12;"COMMENT" KB VERT.IN THE MIDDLE,HOR.KW AND R EQUAL.     135
136       APART OF THE IDEAL POSITION THE SITUATION IS A7A8 ;                   136
137       "IF" MEASURE1=5 "AND" ABS(KWMINRV1)=2 "THEN" TAX:=2*RANGEV-2          137
138                       "ELSE" A7A8;                                          138
139     "PROCEDURE" A13;"COMMENT" COMPLEMENT OF A12;                            139
140       "IF" MEASURE1=5 "AND" ABS(KWMINRH1)=2 "THEN" TAX:=2*RANGEH-2          140
141                       "ELSE" A7A8;                                          141
142     "PROCEDURE" A14;"COMMENT" ROOK VERT.IN THE MIDDLE. KB NOT ON THE EDGE,  142
143       NO TAX, DUMMY STATEMENT; ;                                            143
144     "PROCEDURE" A15;"COMMENT" ROOK HOR.IN THE MIDDLE. TAX IF KB ON THE      144
145           EDGE AND R STANDS WELL. ROOM MAY NOT BE 1 ( STALEMATE! );         145
146       "IF" KBH=8 "THEN"                                                     146
147       "BEGIN" "IF" KWH1=6 "AND"SIGN(KBMINKV1)=SIGN(KWMINRV1)                147
148               "THEN" TAX:= ABS(KWMINRV1);                                   148
149         "IF" KBV=8"THEN"""BEGIN""IF"MEASURE1=5"THEN"TAX:=TAX-1"END"         149
150       "END" ;                                                               150
151                                                                             151
152     A:=ABS(KBMINRV1);B:=ABS(KBMINRH1);DISTANCE:=A**2+B**2;                  152
153     "IF" A<B "THEN" A:=B;                                                   153
154     C:=ABS(KWMINRV1);D:=ABS(KWMINRH1); "IF" C<D "THEN" C:=D;                154
155     "IF" A=1"AND"C>1 "THEN" ROOM := 50 "ELSE"                               155
156     "BEGIN" RANGEV:= "IF" KBMINRV1>0 "THEN"8-RV1"ELSE"RV1-1;                156
157       RANGEH:= "IF"KBMINRH1>0"THEN"8-RH1"ELSE"RH1-1;                        157
158       STRONG := "FALSE";                                                    158
159       "IF" A >= C "THEN"                                                    159
160       "BEGIN""IF""NOT"(SIGN(KBMINKV1)=-SIGN(KBMINRV1)"AND"                  160
161                        ABS(KBMINKH1)<2 ) "THEN"                             161
162         "BEGIN""IF""NOT"(SIGN(KBMINKH1)=-SIGN(KBMINRH1) "AND"               162
163                        ABS(KBMINKV1)<2 ) "THEN" STRONG:="TRUE"              163
164         "END"                                                               164
165       "END";                                                                165
166       VERT := RANGEV <= RANGEH;  TAX := 0;                                  166
167       "IF" "NOT" STRONG "THEN"                                              167
168       "BEGIN" "IF" VERT "THEN" RANGEH:=7 "ELSE" RANGEV:=7 "END";            168
169                                                                             169
170       "COMMENT" NOW A CALL IS MADE TO ONE OUT OF SEVERAL PROCEDURES,        170
171         BASED ON THE POSITION OF KW, KB AND R WITH RESPECT TO ONE           171
172         ANOTHER.IN THE PROCEDURE THE VALUES OF TAX,AXISV AND AXISH          172
173         ARE COMPUTED;                                                       173
174       "IF" QUESTION(CHECK) "THEN" A1 "ELSE"                                 174
175         "IF" SIGN(KBMINRV1)=-SIGN(KWMINRV1) "THEN" A14 "ELSE"               175
176         "IF" SIGN(KBMINRH1)=-SIGN(KWMINRH1) "THEN" A15 "ELSE"               176
177         "IF" SIGN(KBMINKV1)= SIGN(KWMINRV1)                                 177
178         "AND" SIGN(KBMINKH1)=SIGN(KWMINRH1) "THEN" A2 "ELSE"                178
179         "IF" SIGN(KBMINKV1)=-SIGN(KBMINRV1)                                 179
180         "AND"SIGN(KBMINKH1)=-SIGN(KBMINRH1) "THEN" A9 "ELSE"                180
181         "IF" SIGN(KBMINKV1)=SIGN(KWMINRV1) "THEN"                           181
182         "BEGIN""IF" KWMINRH1=0 "THEN" A3 "ELSE"                             182
183           "IF" KBMINKH1=0 "THEN" A5 "ELSE" A7                               183
184         "END" "ELSE"                                                        184
185         "IF" SIGN(KBMINKH1)=SIGN(KWMINRH1) "THEN"                           185
186         "BEGIN""IF" KWMINRV1=0 "THEN" A4 "ELSE"                             186
187           "IF" KBMINKV1=0 "THEN" A6 "ELSE" A8                               187
188         "END" "ELSE"                                                        188
189         "IF" SIGN(KBMINKV1)=-SIGN(KBMINRV1) "THEN"                          189
```

```
ALGOL-60   VERSION 4.0       LEVEL 0013

190           "BEGIN""IF" KBMINKH1=0 "THEN" A10 "ELSE" A12 "END""ELSE"          190
191           "IF" SIGN(KBMINKH1)=-SIGN(KBMINRH1) "THEN"                        191
192           "BEGIN""IF" KBMINKV1=0 "THEN" A11 "ELSE" A13 "END""ELSE"          192
193           "IF" KBMINKH1=0 "THEN" A5 "ELSE" A6 ;                             193
194                                                                             194
195         "IF""NOT" STRONG "THEN" TAX := TAX - 1;                             195
196         ROOM := RANGEV * RANGEH - TAX                                       196
197       "END"                                                                 197
198     "END" ROOM ;                                                            198
199                                                                             199
200                                                                             200
201     AXIS:=0;                                                                201
202     "IF" KBH<5 "THEN" "BEGIN" TRANSPOSE(1);AXIS:=AXIS+1"END";               202
203     "IF" KBV<5 "THEN" "BEGIN" TRANSPOSE(2);AXIS:=AXIS+2"END";               203
204     "IF" KBH<KBV                                                            204
205     "OR" KBH=KBV "AND"(KWV<KWH "OR" KWV=KWH"AND"RH<RV)                      205
206       "THEN" "BEGIN" TRANSPOSE(3); AXIS:=AXIS+4 "END";                      206
207                                                                             207
208     KBMINKV := KBV-KWV; KBMINKH :=KBH-KWH;                                  208
209     KBMINRV := KBV- RV; KBMINRH := KBH-RH;                                  209
210     KWMINRV := KWV -RV; KWMINRH := KWH-RH;                                  210
211     MEASURE := KBMINKV**2 + KBMINKH**2;                                     211
212     "IF" IRREGULAR "THEN"                                                   212
213     "BEGIN" REVERSE;                                                        213
214       "IF" COUNTERM "THEN""BEGIN" KBV:=C; KBH:=D"END";                      214
215       AMISS                                                                 215
216     "END" IRREGULAR;                                                        216
217                                                                             217
218     "COMMENT" NOW MATING POSITIONS ARE CHECKED;                             218
219     "IF" MEASURE=4 "OR" MEASURE=5"AND"KBV=8 "THEN"                          219
220     "BEGIN" "IF" KBH=8 "THEN"                                               220
221       "BEGIN" "IF" KWH=6 "THEN"                                             221
222         "BEGIN" "IF" ABS(KBMINRV)>1 "THEN"                                  222
223           "BEGIN" MATE := "TRUE"; RH:=8; TYPE:=1 "END" "ELSE"               223
224           "IF" KBV=8 "THEN"                                                 224
225           "BEGIN" RV := 6;  TYPE:=2 "END" "ELSE"                            225
226           "IF" KBMINRV=0 "THEN"                                             226
227           "BEGIN" RV := RV-1; MATE2:= "TRUE"; TYPE:=2 "END" "ELSE"          227
228           "IF" RH=7 "THEN"                                                  228
229           "BEGIN" RH:=1; TYPE:=1; MATE2:="TRUE" "END" "ELSE"                229
230           "BEGIN" KWV:= KWV+ SIGN(KWMINRV); "COMMENT" MATE IN 1 ;           230
231             TYPE:=3                                                         231
232           "END";                                                            232
233           "GOTO" READY2                                                     233
234         "END"                                                               234
235       "END"                                                                 235
236     "END";                                                                  236
237                                                                             237
238     "IF" MATE2 "THEN"                                                       238
239     "BEGIN" A := KBV + SIGN(KBMINKV);                                       239
240       "IF"RV=A "THEN" RH := "IF"RH=1"THEN"2"ELSE"1                          240
241                 "ELSE" RV := A;                                             241
242       "COMMENT" MOVE FIXED, MATE IN 1 ;                                     242
243       MATE2 := "FALSE"; TYPE:=1; "GOTO" READY2                              243
244     "END" MATE2;                                                            244
245                                                                             245
246     "COMMENT" IF MEASURE>15 A KING MOVE IS MADE IMMEDIATEDLY,               246
247       UNLESS R CAN BE CAPTURED (DANGER);                                    247
248     DANGER := KBMINRH**2+KBMINRV**2 <3 "AND" KWMINRV**2+KWMINRH**2 >3;      248
249     "IF" "NOT" DANGER "AND" MEASURE > 15 "THEN"                             249
250     "BEGIN" A:=KWV +SIGN(KBMINKV);  B:=KWH +SIGN(KBMINKH);                  250
251       "IF" A=RV "AND" B=RH "THEN"                                           251
252       "BEGIN" "IF" ABS(KBMINKV) > ABS(KBMINKH) "THEN"                       252
```

```
ALGOL-60   VERSION 4.0       LEVEL 0013

253                KWV := A "ELSE" KWH := B                                  253
254          "END" "ELSE"                                                    254
255          "BEGIN" KWV:=A; KWH:=B "END";                                   255
256          TYPE:=3; "GOTO" READY2                                          256
257        "END" MEASURE > 15 ;                                              257
258                                                                          258
259        "COMMENT" INITIALISATION.ILLEGAL MOVES AND SOME OBVIOUSLY BAD     259
260          MOVES ARE SET FALSE;                                            260
261        MAX1:=MAX2:=100;                                                  261
262        "FOR" TYPE:=1,2,3"DO""FOR"FIELD:=1"STEP"1"UNTIL"8"DO"             262
263          MOVE [TYPE,FIELD] := "TRUE" ;                                   263
264        MOVE[2,RV]:=MOVE[1,RH]:=MOVE[2,3]:=MOVE[1,3]:=MOVE[1,4]:="FALSE";264
265        "IF" "NOT" RV = 1 "THEN" MOVE[2,2]:= "FALSE";                     265
266        "IF" "NOT" RH = 1 "THEN" MOVE[1,2]:= "FALSE";                     266
267        "IF" RV = KWV "THEN"                                              267
268        "BEGIN" "IF" KWH>RH "THEN"                                        268
269          "BEGIN""FOR"FIELD:=KWH"STEP"1"UNTIL"8"DO"MOVE[1,FIELD]:="FALSE""END"269
270          "ELSE" "FOR"FIELD:=1"STEP"1"UNTIL"KWH"DO"MOVE[1,FIELD]:="FALSE"  270
271        "END" "ELSE"                                                      271
272        "IF" RH = KWH "THEN"                                              272
273        "BEGIN" "IF" KWV>RV "THEN"                                        273
274          "BEGIN""FOR"FIELD:=KWV"STEP"1"UNTIL"8"DO"MOVE[2,FIELD]:="FALSE""END"274
275          "ELSE" "FOR"FIELD:=1"STEP"1"UNTIL"KWV"DO"MOVE[2,FIELD]:="FALSE"  275
276        "END";                                                            276
277        "IF" KWH=8 "THEN" MOVE[3,1]:=MOVE[3,2]:=MOVE[3,4]:= "FALSE";      277
278        "IF" KWV=8 "THEN" MOVE[3,1]:=MOVE[3;3]:=MOVE[3,6]:= "FALSE";      278
279                                                                          279
280        "COMMENT" MOVES ARE GENERATED. SOME BAD MOVES ARE REJECTED. THEN ROOM 280
281          AND MEASURE ARE COMPUTED. THE BEST MOVE BECOMES CANDIDATE;      281
282        "FOR"TYPE:=1,2,3"DO""FOR"FIELD:=1"STEP"1"UNTIL"8"DO"              282
283            "IF"MOVE[TYPE,FIELD] "THEN"                                   283
284        "BEGIN"  GENERATE ;                                               284
285          "IF" MEASURE1-MEASURE>1 "OR" MEASURE1<3                         285
286          "OR" KWMINRV1=0"AND"KWMINRH1=0 "THEN" "GOTO" REJECT;            286
287          POS := ((((KBV*8+KBH)*8+RV1)*8+RH1)*8+KWV1)*8+KWH1;             287
288          "FOR" A := 1"STEP"1"UNTIL" COUNT - 1 "DO"                       288
289            "IF" POS = HIST[A] "THEN" "GOTO" REJECT;                      289
290          B := ROOM;                                                      290
291          "COMMENT" 4 CRITERIA: MINIMALIZE ROOM,THEN MEASURE1,            291
292            MAXIMALIZE DISTANCE, NOW THE ORDER OF MOVES IS DECISIVE.      292
293            MEASURE1 OF 4 AND 5 ARE EQUIVALENT ;                          293
294          "IF" MEASURE1=5 "THEN"MEASURE1:=4;                              294
295          A:= "IF" B = 1 "THEN" 0 "ELSE"                                  295
296               SIGN(MAX1-B)*4 +SIGN(MAX2-MEASURE1)*2 -SIGN(MIN-DISTANCE); 296
297          "IF"A<=0 "THEN" "GOTO" REJECT;                                  297
298        CANDIDATE:                                                        298
299          TYPEM:=TYPE; FIELDM:=FIELD; PRESENT:=POS;                       299
300          MAX1:=B; MAX2:=MEASURE1; MIN:=DISTANCE;                         300
301        REJECT:                                                           301
302        "END" "FOR" TYPE ;                                                302
303                                                                          303
304        TYPE := TYPEM; FIELD := FIELDM;                                   304
305        "COMMENT" THE MOVE HAS BEEN COMPUTED AND IS NOW MADE;             305
306                                                                          306
307      READY: GENERATE; KWV:=KWV1;KWH:=KWH1;RV:=RV1;RH:=RH1;               307
308        QUESTION (CHECK); HIST[COUNT]:=PRESENT;                           308
309      READY2: REVERSE;                                                    309
310        "IF"TYPE = 3 "THEN"                                               310
311        "BEGIN" OUTPUTS("("ZD"(". K")")",COUNT);                          311
312          OUTSYMBOL(KWV);                                                 312
313          OUTPUTS("("D5B")",KWH)                                          313
314        "END""ELSE"                                                       314
315        "BEGIN" OUTPUTS("("ZD"(". R")")",COUNT);                          315
```

```
        ALGOL-60    VERSION 4.0        LEVEL 0013

316       OUTSYMBOL(RV);                                                    316
317       OUTPUTS("("D")",RH);                                              317
318       "IF" MATE "THEN" OUTPUTS("("""("* MATE")"Z//")",0) "ELSE"         318
319       "IF" CHECK "THEN" OUTPUTS("("""("*")"3BZ")",0)                    319
320                    "ELSE" OUTPUTS("("4BZ")",0)                          320
321     "END" TYPE ;                                                        321
322     OUTPUTS("("Z/")",0)                                                 322
323   "END" MAKE YOUR MOVE;                                                 323
324                                                                         324
325                                                                         325
326   "PROCEDURE" OUTPUTS(S,EL);"VALUE"EL;"INTEGER"EL;"STRING"S;            326
327     "BEGIN" OUTPUT(61,S,EL);                                            327
328             OUTPUT( 1,S,EL)                                             328
329     "END";                                                              329
330   "PROCEDURE" OUTSYMBOL(NR); "VALUE"NR;"INTEGER"NR;                     330
331     "BEGIN" OUTCHARACTER(61,"("ABCDEFGH")",NR);                         331
332             OUTCHARACTER( 1,"("ABCDEFGH")",NR)                          332
333     "END";                                                              333
334   "PROCEDURE" LETTER(CHAR);"INTEGER"CHAR;                               334
335     "BEGIN" "BOOLEAN" FIRST; FIRST:="TRUE";                             335
336     READ:INCHARACTER(60 ,"("ABCDEFGH0 ")",CHAR);                        336
337         "IF" CHAR=-1 "THEN" "GOTO" READ;                                337
338         "IF" FIRST "THEN" "BEGIN" "IF" CHAR=10 "THEN" "GOTO" READ;      338
339                                   FIRST := "FALSE";                     339
340                                   "IF" CHAR=0"THEN""GOTO"READ           340
341                           "END";                                        341
342         "IF" CHAR=9 "THEN" "GOTO" "IF"COUNTERM"THEN"RESIGNS"ELSE"STOP;  342
343         "IF"CHAR=0 "OR" CHAR=10 "THEN" AMISS                            343
344     "END";                                                              344
345   "PROCEDURE" DIGIT(NR);"INTEGER"NR;                                    345
346     "BEGIN" INCHARACTER (60,"("12345678")",NR);                         346
347             "IF" NR=0 "THEN" AMISS                                      347
348     "END";                                                              348
349   "PROCEDURE" AMISS;                                                    349
350     "BEGIN" OUTPUTS("("""("NOT CORRECT. TRY AGAIN")"Z/")", 0);          350
351       INPUT(60,"("/")");                                                351
352      "GOTO" "IF" COUNTERM "THEN" NEXT MOVE "ELSE" TYPE IN               352
353     "END";                                                              353
354                                                                         354
355   OUTPUT(61,"("//,"("PLEASE GIVE THE POSITION IN ALGEBRAIC NOTATION.EACH  355
356   ")"/,"("PIECE BENEATH THE CORRESPONDING COLUMN:WHITE KING, ROOK, BLACK  356
357   ")"/,"("KING. FOR EXAMPLE: KA1   RH1   KD5                            357
358   ")"/,"("YOU ARE PLAYING WITH BLACK.TYPE YOUR MOVE WHEN WHITES MOVE IS   358
359   ")"/,"("GIVEN. TYPING ZERO MEANS RESIGNATION ")"//,                     359
360        "("KING  ROOK  KING ")"/")");                                      360
361                                                                         361
362   START: COUNT := 1;  COUNTERM :="FALSE";                               362
363     MATE:=MATE2:="FALSE";                                               363
364     "FOR" A:=1"STEP"1"UNTIL"50"DO" HIST[A]:= 0;                         364
365   TYPE IN: LETTER(KWV); DIGIT(KWH);                                     365
366            LETTER( RV); DIGIT( RH);                                     366
367            LETTER(KBV); DIGIT(KBH);                                     367
368     "FOR" A:=1,61 "DO"                                                  368
369       OUTPUT(A,"("//"("POSITION:  K")"A,D,5B"("R")"A,D,5B"("K")"A,D/")",  369
370              KWV*2**42,KWH,RV*2**42,RH,KBV*2**42,KBH);                  370
371                                                                         371
372   PLAY:  MAKE YOUR MOVE;                                                372
373                                                                         373
374     "IF"MATE "THEN" "GOTO" NEXT GAME;                                   374
375     COUNT:=COUNT+1: COUNTERM:="TRUE": INPUT(60,"("/")");                375
376                                                                         376
377   NEXT MOVE: LETTER(KBV1); DIGIT(KBH1);                                 377
378     "IF" ABS(KBV1-KBV)>1 "OR" ABS(KBH1-KBH)>1                           378
```

```
        ALGOL-60   VERSION 4.0       LEVEL 0013

379       "OR" KBV1=KBV "AND" KBH1=KBH       "THEN" AMISS;                                379
380       C:=KBV; D:=KBH; KBV:=KBV1; KBH:=KBH1;                                           380
381       OUTPUT(1,"("""("K")"A,D/")",KBV*2**42,KBH);                                     381
382       "GOTO" PLAY;                                                                    382
383                                                                                       383
384     RESIGNS: OUTPUT(1,"("""("BLACK RESIGNS")""")");                                   384
385     NEXT GAME:OUTPUT(61,"("//"("ANOTHER GAME? THEN GIVE POSITION, ELSE TYPE           385
386     ")","("ZERO ")",//,"("KING  ROOK  KING")",/")");                                  386
387           INPUT(60,"("/")");                                                          387
388           "GOTO" START;                                                               388
389     STOP:                                                                             389
390     "END"                                                                             390
```

REFERENCES

[1] M.M. BOTWINNIK, *Computers, Chess and Long Range Planning*, Springer Verlag, 1970.

[2] A. CHÉRON, *Lehr- und Handbuch der Eindspiele, Teil II*, Engelhardt Verlag, Berlin (in German).

[3] M. EUWE & J. DEN HERTOG, *Praktische Schaaklessen I en II*, Van Goor, The Hague (in Dutch).

[4] R. FINE, *Basic Chess Endings*, David McKay Cie, New York.

[5] J.J. GILLOGLY, *The Technology Chess Program*, Nov. 1971, Rep., CMU-CS-17-109, Dep. of Computer Sc., Carnegie Mellon University.

[6] BARBARA J. HUBERMAN, *A Program to Play Chess End Games*, Tech. Rep. CS-106, August 1968, Comp. Sc. Dep., Stanford University.

[7] D.N. LEVY, *Computer Chess*, A Case Study on the CDC 6600, Machine Intelligence, vol. 6, 1971, page 151-163.

[8] B. MITTMAN, *Can a Computer Beat Bobby Fisher*, Datamation, June 1973, page 84-87.

[9] P.G. RUSHTON & T.A. MARSLAND, *Current Chess Programs*, A Summary of their Potential and Limitations, INFOR, vol. 11 no 1, Febr. 1973, page 13-20.

[10] A.L. SAMUEL, *Some Studies in Machine Learning Using the Game of Checkers*, IBM, J.Res.Dev., July 1959, page 210-229 and November 1967, page 601-617.

[11] S.T. TAN, *Representation of knowledge for very simple pawn endings in Chess*, MIP-R-98, November 1972; *Kings, Pawn and Bishop*, MIP-R-108, May 1974, School of Artificial Intelligence, University of Edinburgh.

[12] A.L. ZOBRIST & F.R. CARLSON JR., *An Advice Taking Chess Computer*, Scientific American, June 1973, page 92-105.