CHESS PIECING TOGETHER

John White discusses the main developments in computerising the game of chess, and suggests how to employ these techniques when you set about writing your first chess program.

SPEAKING IN 1949, the mathematician Shannon pointed out that an average chess game lasted 40 moves and that there are an average 30 move possibilities during each of those moves. There are, therefore, at least 10^{120} possible games of chess. To search these at the ridiculously high rate of 1,000,000 games per second would require a search time of 10^{108} years to exhaust all possibilities.

When constructing a program to play chess, it is certainly not possible to carry out an exhaustive search. Instead, a search of a few moves is considered and the position which arises is scored. The means of doing this is known as the evaluation function — EF — and the strength of a chess program depends very much on how effective the EF is.

Contary to popular belief, it is actually relatively easy to write a chess program. The real difficulty lies in constructing a program which meets the following requirements:

- Plays strong chess.
- Makes its move within a reasonable time — certainly not more than five minutes.
- Uses little memory — less than 32K at the outside.

Skilled chess programmers earn high fees, with good reason. It is generally true that good programs are written by highly-experienced machine-code programmers, rather than by good chess players. The mainframe world computer chess champion, Belle, was written by a non-chess player, and relies on brute strength to find its move rather than by making complicated positional assessments. However, some appreciation of good chess play is necessary to write a good program for a micro.

Before embarking on writing a chess program, you should clearly define what you expect from it. The amateur, working in his spare time, is unlikely to be able to produce a program capable of beating one written by a team of full-time, salaried professionals. Nor will the amateur have access to dedicated chess units for his program, and will have to rely on his domestic microcomputer which cannot be expected to run so fast as the dedicated unit would.

On the other hand, writing chess programs will give much pleasure and will also improve your programming skill. With practice, you will soon find that writing programs becomes much easier; you will not, for example, need to keep rewriting your move generator.

There are also many discoveries waiting to be made in improving evaluation and search procedures. Computer chess is essentially only 30 years old, and the alpha-beta pruning method only half that. The obvious "chopper" pruning mechanism entered commercial chess computers as late as 1981.

As a final inducement, I should mention that many of the chess games available on cassette for domestic micros play a feeble game which could easily be improved — think of the market for the Vic and the BBC Micro.

Searching for checks

Many modern chess computers search to a depth of four-ply at their higher levels, searching deeper for checks and captures. A ply of search is equivalent to one half move, that is, one move by either side. This dictates the choice of programming language. If a machine can choose its best move at one-ply in one second, then it will take 30 seconds to examine its opponent's reply to each of its moves, 900 seconds to examine its own responses to each of the possible opponent moves, and 27,000 seconds — 7.5 hours — to examine the opponent counter responses at the fourth ply. If the program takes 10 seconds to find its best one-ply move, than it will take 312 days to search to a depth of four-ply.

These figures assume a full search of the tree of moves which is constructed by considering all possible permutations of moves to a depth of four-ply. In fact, powerful pruning methods exist to reduce the size of the tree, and a program which can select its best one-ply move in one second can be made to search to a depth of four-ply in about two to three minutes.

To reduce the time spent selecting a move at one ply to 10 seconds or less, machine code or assembly is essential. Out of curiosity, I wrote two chess programs, one in interpreted Basic and one in compiled Fortran, to see how long they would take to run with a one-ply search. The Basic version took three minutes per move, the Fortran version five seconds, and both played ghastly chess due to a minimal EF. A good machine-code program should find its one-ply move within one second.

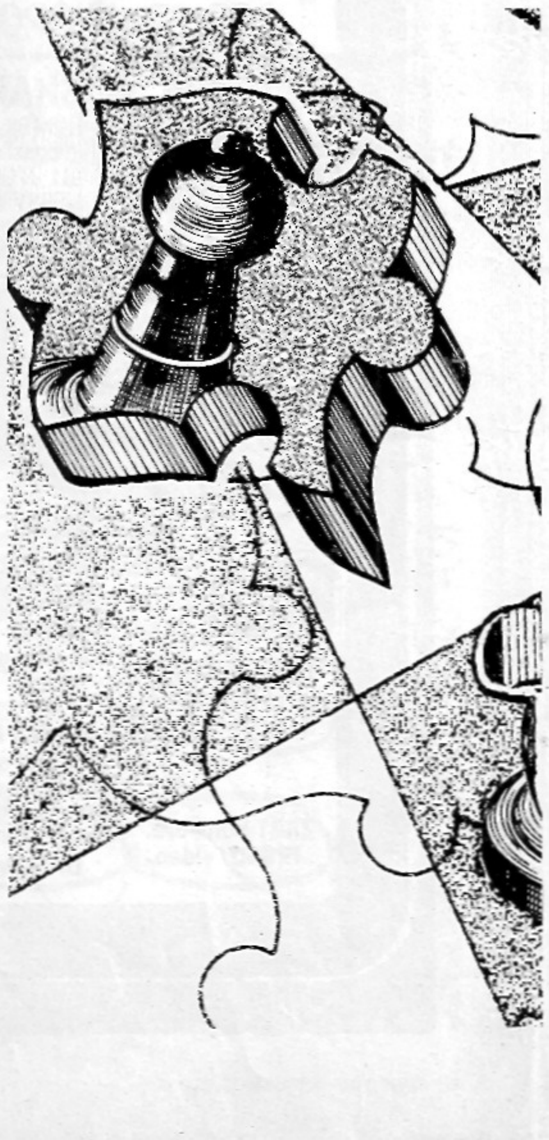
Another good reason for programming in machine code is the inability of many other languages, including Basic, to perform recursion. I do not know any way of enabling a Basic program to call itself more than once, since the Return statement bears no label. It is desirable to use, say, a move generator at each level of search, rather than to have to write a fresh generator for each and every level.

Finally, machine-code programs are more economical on space than other languages, and it is possible to write a reasonable program using less than 4K of RAM.

A variety of methods may be used to set up the chessboard. Simplest is a two-dimensional array, where different positive values are assigned to the machine pieces, and negative values to the opponent's. The values may be equated with the nominal value of a piece, so a queen could be assigned a value of 90, a bishop 32 and a knight 29. Loss of the king is fatal, so these are assigned very high values, say, about 5,000.

All moves to a position will then be subject to the constraint that X * (9−X) and Y * (9−Y) — where X and Y are the new co-ordinates on the board — must both be greater than zero.

It is common practice to put the 0th and ninth lines of the array to a number which is distinguishable from the pieces. This marks



THE BEST MOVES

the rim of the board and saves checking whether a project move has gone off the board.

Picture a rook moving down a file. It can either feel its way down cautiously or it can thunder down until it bounces off the rim. A second rim can also be added to check the legality of knight moves, which may hop over the first rim. Remember that the rim may have to change sign, in some implementations, according to which side is moving.

Separate table

It is also possible to devise chessboards in a one-dimensional array and even to use two two-dimensional arrays, so that all the pieces will appear to be moving in the same direction.

The position of major pieces can additionally be kept in a separate piece table, and this enables attacks on enemy pieces to be found very quickly. This method is used by all the major chess manufacturers.

The move generator simply calculates all the moves for the chess men. A queen is composed of a rook and a bishop. The generator can be written very easily, but remember to test for move legality. A piece can move on to a vacant square, on to an enemy-occupied square but not on to a friendly-occupied square. A piece cannot move through another piece, unless it is a knight.

En passant is also reasonably easy to cater for. A flag is set whenever a pawn makes a move enabling *en passant* by the opponent. Much more difficult is castling where the castling rook must not have been moved, the king must not have been moved, the king must not be in check and the king must not pass through check or settle in check.

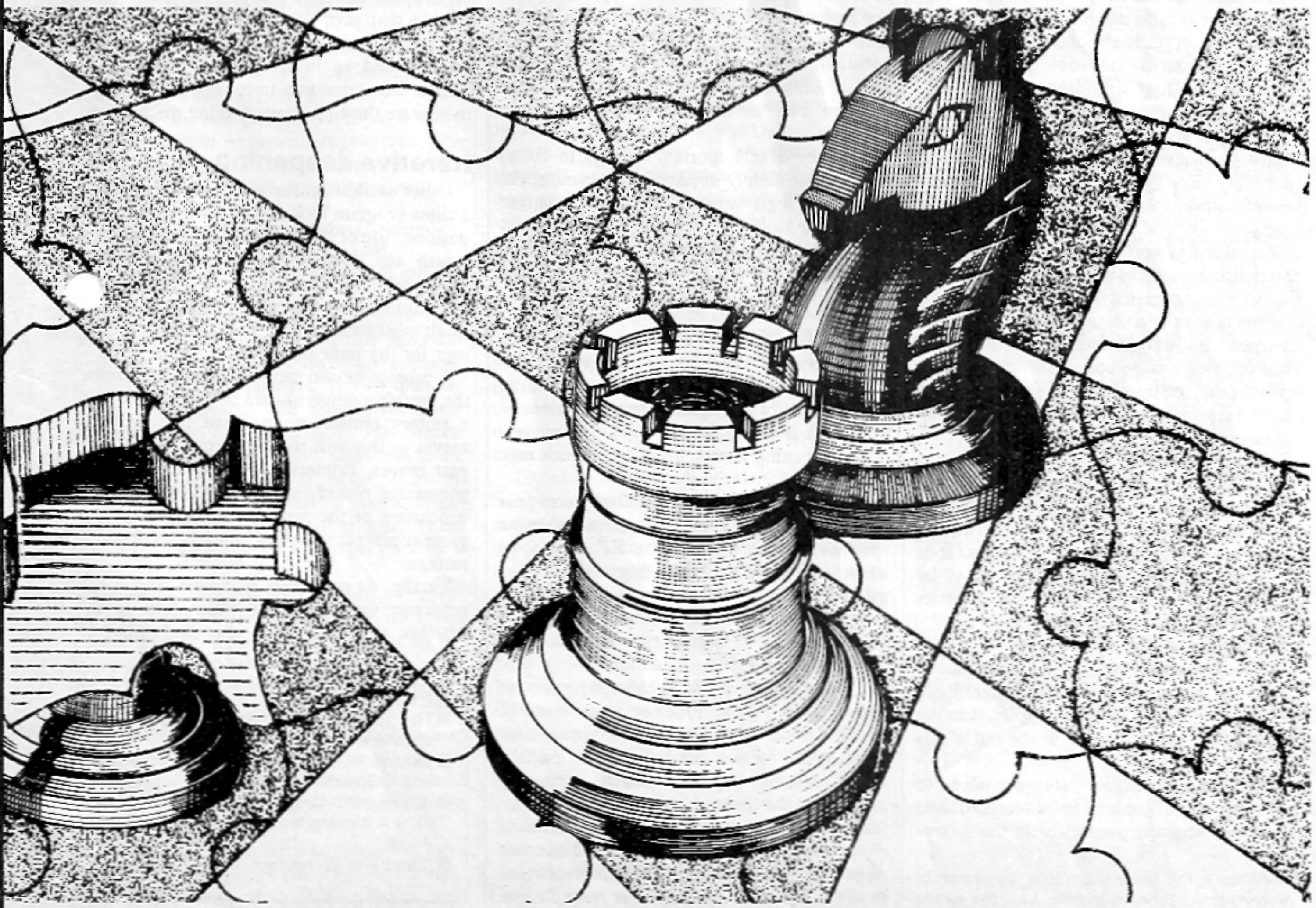
A test to see if the king is in check is, therefore, essential and this can also be used to give priority to king protection. The test is done by making legal rook, bishop, pawn and knight moves away from the king, and testing to see whether the appropriate enemy piece is encountered. Testing to see if the moving piece is giving check is inadequate, since it may miss a discovered check. Incidentally, do not forget to allow for the possibility of double check.

Evaluation function

Two methods are available for working out which move a computer should make. The first is the EF, which assesses and scores a position arising after a piece has been moved. The second is the look ahead — search in depth — which considers responses to machine moves, counter responses, counter-counter responses and so on.

There is considerable debate in computing circles as to whether a chess program should use a sophisticated EF combined with a shallow search — but searching some forcing lines such as captures in greater depth — or

(continued on next page)



(continued from previous page)
should use a minimal EF with as deep a search as possible. Bearing in mind that the EF is called after every potential move, it must be kept as short — that is, as fast — as possible, particularly for a deep search, and will rarely contain more than 20 elements.

Devotees of the first method point out that a detailed search of only part of the tree, selected by a sophisticated EF, most closely mimics human chess play. David Levy is a believer in this approach, and his company's Philidor program uses special, still secret methods to attain a strong EF capable of considering even strategic factors. Another example is the German Shach computer which plays a respectable game with a look ahead of only one move, but with a very powerful EF.

Adherents of the second approach observe that programmers should concentrate on the computer's greatest strength — its ability to calculate rapidly. The Belle program makes its moves by calculation deep into the chess tree. It is also common to give great consideration to the EF at the first and second ply, but to reduce the EF at all subsequent levels, so as to spend less time searching.

Search routines

By convention, the score from an EF is taken as positive if favourable for the program, negative if unfavourable. An essential feature of any good EF is an evaluation of the number and quality of pieces bearing on any square, particularly in the centre. This is done as described for the search for check on the king, and the same routine can be used for both purposes. However, it is important to remember that for square control one piece hidden behind another may still exert an effect.

For example, a queen on the same diagonal as a bishop — with no intervening piece — will exert its own pressure on the same squares as the bishop. The bishop will exert the greater pressure, since it is more expendable than the queen.

Other features worth including in the EF are material count, attacks on king, queen or lesser pieces, pins, presence of doubled pawns, development of pieces, whether castling has occurred and advancement of pawns. Yet other features can be added, limited only by available time or imagination. Some examples I have seen include fianchettoing the bishop and doubling rooks along a file or row.

So far we have been considering only the evaluation of positions. An alternative method is to evaluate each move as it is made. This is significantly faster than evaluating positions, but unfortunately gives weaker results. It is not suited to chess programs, but could be used for draughts or other games where tactics are more important than strategy.

The ability to search moves in depth is a subject which would require as much space as all the rest of this article put together. Basic principles, with excellent examples, can be found in the references given at the end of this article.

Numerous techniques are available to reduce the size of the tree to be searched, and you should acquaint yourself with the following:

- Minimax is the name given to a full search of every permutation of moves, i.e., the whole tree, where the opponent tries to minimise the machine's score while the program tries to maximise it. This is the slowest type of search.
- Alpha-beta search is a method of pruning which gives the same result as minimax but in less time. The principle is that if any one response to a program move can be found that makes the move weaker than one previously considered, then the program need not waste time calculating other responses to that move.
- Hard pruning can be effected simply by eliminating all potential moves which fail to achieve a certain minimum score.
- Razoring makes use of the assumption that the opponent can always find a move that will make things better for him than if he had made no move.
- Chopper: sometimes the program has only one legal move. This can be made at once without need to calculate all the possible responses and counter responses.
- Killer heuristic makes use of the assumption that any response which cuts off part of the tree with alpha-beta pruning will also cut off another part at the same level.

The efficiency of many of these pruning methods, especially alpha-beta, can be greatly increased by sorting the moves into decreasing score order. This must not be done too often, or the time spent sorting exceeds the time saved, but can be very effective if carried out after each ply of search.

This leads us to iterative deepening which is used on virtually all the better chess machines now. All the moves are found at the first level. These are sorted into score order and searched to the second ply. These are sorted and searched to third ply and so on. By this means, the best move yet found is always available at each level of search, and this is normally constantly displayed. If a timer is employed, the thinking can be interrupted at any time and the best move becomes the machine's choice.

Mobility is an important concept in chess, and is most simply obtained by summing the number of legal moves made by the program and by the calculated responses to each of its moves. Pruning must not be carried out at the first and second ply or this method will not work.

Normal pruning

Mobility may also be assessed in a sufficiently sophisticated EF — for example, by modification of the square control routines — but slows the program noticeably. However, normal pruning is now permitted, which may compensate.

When the total material count of pieces falls below a minimum level, then extra end-game routines can be called. The EF should be adapted to make the king more active and to make the advance of pawns — especially as chains — more favourable. The depth of search can also be safety increased since less material is available to be moved.

As a guide, the powerful Morphy program enters its end-game routines when material equivalent to two kings, two rooks, two knights and seven pawns remains on the board, and the depth of search is roughly doubled at the higher levels.

Book openings are very useful for games of chess, enabling the program to avoid opening traps and permitting some non-obvious strategic moves to be made. For example, the black move C7-C5 is thematic in many queen's pawn openings, yet I know of no program which does this early in the game except as part of a book opening.

The only limit on the book is that of memory space, which is unlikely to trouble most owners of micros. The book should be held in an array, matched against move number, and not as part of an opening tree which will take a long time to search.

Random selection between moves of nearly equal merit is a very useful feature, making all games different, and is most simply done by adding a small random number in the EF to the score from each evaluation.

Counter moves

After completing its search, the program will come up with a series of moves and a series of counter moves. These can be stored, which is very expensive on memory but will enable the program — on request — to reject the best move in favour of the next best and so on. This facility is available on several of the programs from the software company Philidor — such as Pet Chess, Intelligent Chess and Che[illegible] Champion Mk V.

More commonly, the scores are compared with a store which is initially set at minus infinity, say, −10,000. If the score exceeds the store, the store level is set to the score and the moves creating the score are also stored. Thus the store is constantly upgraded until only the best move remains in the move storage area. For opponent responses, a second store is preset to plus infinity — say, +10,000 — and scores that are less than the store are exchanged with the score until the lowest-scoring, and so best, opponent response is stored. The same two stores can also be used to operate the alpha-beta pruning mechanism.

Iterative deepening

Other useful features which can be added to a chess program include the ability to set up a position, use of real-time clocks to record play length and a move counter. For programs using iterative deepening, the clocks can [illegible] used to interrupt the machine's thinking, a[illegible] a halt button can also be made available for the user for the same purpose.

A prompt button can be used to reveal what the machine thinks should be your move, and a restore button can be used to take back moves — this will require memory storage of past moves. Printers can be interfaced for a permanent record, which may be had at the conclusion of the game on request, or as the game is played; the latter requires no memory storage.

Finally, do not forget to couple your move generator with a routine which tests to see whether the opponent's proposed input is legal.

REFERENCES

- The following are essential reading for anyone writing a chess program:
- *Sargon — a computer chess program,* D and K Spracklen, Hayden.
- *Advances in Computer Chess* pages 89 to 97, J A Birmingham and P Kent, Pergamon Press.
- *Chess and Computers,* D Levy, Batsford.