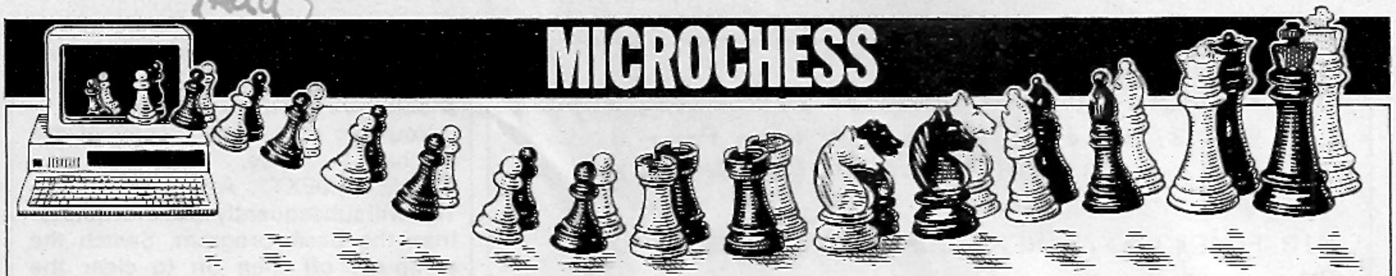
JULY 1984 approx (AUG)

MICROCHESS



Booked to win!

Tony Harrington is often asked for his advice on chess programming. This month he recommends one chess publication which should be of interest to beginners.

Welcome to Micro Chess

Micro Chess covers all the news and events in the busy world of computer chess. With new chess programs and new chess computers appearing all the time, we evaluate their strengths and weaknesses as they become available. We shall be presenting profiles of programmers, both amateurs and professionals, which will cover their methods and their interest in chess programming, and we shall be talking to suppliers and looking at their plans.

Computer Chess affects computer enthusiasts in two different ways. For some, the fact that they can now play chess against either their home computer or a dedicated chess computer has opened up the delights of the game. For others, the real interest is not so much in playing chess as in trying to build a chess program. Micro Chess aims to meet the interests of both.

Chess is a game that can be as exciting for the beginner as it is for the grand master. So if you haven't played before, get yourself a good introduction to the game — there are dozens in the bookshops — and get to it.

The Chess Computer Handbook (Batsford £4.95) written by David Levy is a very good place to begin to understand such abstruse matters as evaluation functions, killer heuristics and the alpha-beta algorithm. (Sorry if these terms put you off the idea of ever taking up chess programming.)

Levy's approach throughout the book is to take the layman firmly, though with some kindness, through the neat logical manoeuvres that make up the received wisdom of chess programming. Not all of what he says is easily grasped. There are mathematics professors who madden their students by saying things like: 'Now it can easily be seen that . . .' when students feel themselves groping about in near total darkness. You might find this text has, on occasion, a similar effect.

This is largely because it is an extremely compact work. The entire subject of chess programming is disposed of in four chapters, totalling 62 pages. The remaining four chapters, which make up the rest of the book's 131 pages, are on topics of general interest, and have titles like 'What to look for in a chess computer', 'How to play against chess programs' and 'How strong can chess computers become?'

The first four chapters are an account of the concepts needed in chess programming; they are not a guide to writing programs in machine code or Basic or anything else. There is not a line of code in the whole book (though Levy reckons that anyone who knows Basic should be able to put the concepts to work without much difficulty.) This absence of code makes it more approachable reading for all those non-programmers who might have wondered how computers can be made to play chess.

This involves two rather different kinds of problem. The first problem, stated by Levy in the opening line of the book, is: How do you tell a computer what chess is? As Levy reminds us: 'It is one thing for a human to gaze at a chessboard, see where the pieces are located and understand the relationships between them, but a computer is merely a device that can store and manipulate numbers.' The answer to this problem takes you into the fundamentals of a chess move generator.

That is the easy bit. The next and far more difficult problem, once the computer can generate chess moves, involves teaching it to recognise worthwhile moves from bad ones. This brings us to the evaluation function. In order for the evaluation function to work well (and computer chess programming still has a long way to go here), all the subtleties of chess have to be reduced to terms the computer can understand — that is, everything has to come down to numbers.

Take the move generator first. To start with, you need a way of telling the computer precisely which piece is on what square. This is achieved by assigning the pieces numbers, positive for white, negative for black. Even in so simple a step there are subtleties to be taken into account. Kings and rooks that have not moved (and which therefore still keep their castling rights) have to be earmarked in some way. Pawns that start with a double move (such as e2-e4) have to be noted so that the *en passant* capture rule can be applied.

This information gives the program all it needs to know about the location and identity of each piece. It still doesn't know what they are. Remember that computers are about numbers. Defining a knight, for example, for a computer is not a pictorial affair. There is no way to tell it that a knight, by historical convention, in the Staunton set, is a horse's head and neck on a pedestal. Tell it all the possible moves a knight can make in any position and you have told it all it needs to know.

Levy explains three ways of generating lists of all the possible legal moves for every piece in any chess position as follows:

(1) Move generation by square offset;
(2) Table driven move generation; and
(3) Incremental updating of move lists.

If you've never thought about this subject before, you should come away from this section with a fair idea of how all three work.

There are some lovely little gems of logic that chess programmers have come up with in even this rather basic area. But first, for the beginner, here, in summary form, is Levy's account of the first method, move generation by square offset. Assign each of the 64 squares on the chessboard a two-digit number, so:

18	28	38	48	58	68	78	88
17	27	37	47	57	67	77	78
16	26	36	46	56	66	76	86
15	25	35	45	55	65	75	85
14	24	34	44	54	64	74	84
13	23	33	43	53	63	73	83
12	22	32	42	52	62	72	82
11	21	31	41	51	61	71	81

If you place a king on square number 54, the rules of chess allow the king to move to any adjacent square not already occupied by his own pieces or not under direct attack by an opponent's pieces (he may, of course, move onto a square occupied by an opponent's piece — captures by the king are not unheard of in chess . . .). A series of arithmetical operations will generate the addresses of all the squares adjacent to square 54. The same operations will hold good for generating the addresses of legal moves for the king from any square on the board (the border squares excepted, for the moment).

As Levy points out, if you call the square the king is on, square k, then, in general form, the squares it can move to (rules permitting) are:

k−9, k+9, k−1, k+1, k−11, k−10, k+10.

Try it and see. If you do your arithmetic properly you should come up with the squares, 45, 63, 53, 55, 43, 65, 44 and 64.

When you've got this far, and defined the move sequences for all the pieces, you'll have an interesting little problem. What do you do about moves which would take a piece off the edge of the board? It might sound trivial, but try solving it.

The first step is to allocate an 'out of bounds' number to offsets without addresses on the board. A more sophisticated solution is to treat the board as a 10x12 cylinder.

Why 10? — knights can leap two squares off the board before hitting an 'out of bounds' square. Why 12? — because, as Levy puts it: '. . . the file which is two files to the left of the a-file actually occupies the same place in the computer's memory as the file which is two files to the right of the h-file.' Got it?

If you get through the first chapter, you probably have a taste for the joys of logical analysis. In which case, Chapter Two, which tackles the basics of building an evaluation mechanism, will be pure delight. Levy proposes ways of giving numerical weights to such factors as material, mobility, centre control, piece development, king attacks, pawn structures, piece attacks and piece defences.

Chapter Three, 'Tree Searches' tells you enough about minimax to get you started on your own program. It also provides a clear account of why the alpha-beta algorithm can cut down the number of positions a computer needs to evaluate by 99.5% — without the least danger of missing a good move.

The fourth and last chapter on this theme deals with search strategies. How dows the computer know when it can profitably spend time searching a position deeply? How does it decide when to terminate a search? This is well-trodden ground, but fascinating for newcomers.

Although I've concentrated on the conceptual side of the book, there is much of interest in the second, more anecdotal, half. All in all, a neat little volume and a useful book for chess programmers.

Games section

White: David Levy. Black: Cray Blitz. Notes by Dr John Nunn.

I had been favourably impressed by some of the games played by Cray Blitz in the world computer championships and other events, so the result of David Levy's match with this program came as rather a surprise to me. Not that David winning 4-0 was a surprise, but the manner in which he won was. Quite simply David made Cray Blitz look like a very weak club player. He relentlessly exploited all the weaknesses of computer programs, taking the machine out of its opening book at the first chance, never allowing complications to start, and utilising Cray's reluctance to repeat the position to induce inferior moves. It is apparently a very different matter for a program to play against other programs than it is to play against the adaptable mind of a knowledgeable human opponent.

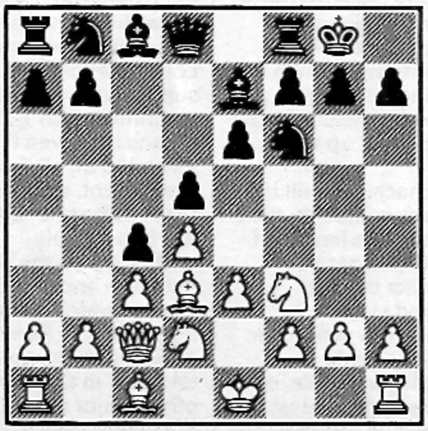
Here is the most interesting game of the match.

★ ★ ★

1 d2-d4 Ng8-f6
c2-c3

(An unusual move designed to take Black out of its opening book.)

2 ... e7-e6
3 Ng1-f3 c7-c5
4 e2-e3 Bf8-e7
5 Qdl-c2 0-0



Position after 7 . . . c5-c4

6 Bf1-d3 d7-d5
7 Nb1-d2 c5-c4?

(A serious positional error. Black is tempted by the prospect of forcing the bishop to retreat, but he forfeits the pressure against White's centre afforded by the attack of the c5 pawn on the one at d4. Without this pressure White has a completely free hand in the centre. To be fair to Cray Blitz, this type of mistake was often made by top human players in the 1890s, when the problems of central play were not well understood. The correct plan was to exchange White's most dangerous piece, the d3 bishop, by 7 . . . b7-b6 followed by 8 . . . Bc8-a6.

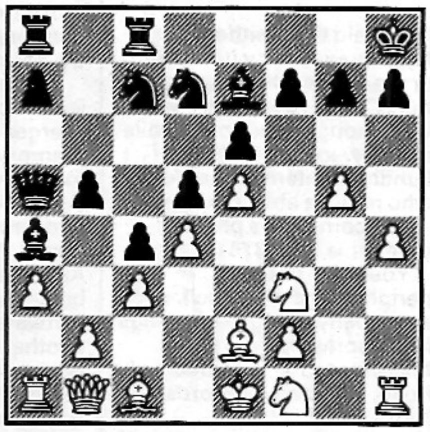
8 Bd3-e2 Qd8-a5?

(Black's only hope for counterplay was to advance his queenside pawns to a5 and b4. On a5 the queen obstructs the execution of this plan.)

9 e3-e4 Bc8-d7
10 e4-e5 Nf6-e8
11 Nd2-f1 Ne8-c7
12 g2-g4

(It is a matter of personal taste how White conducts the attack. 12 h2-h4 followed by Nf3-g5 might have been even stronger.)

12 . . . Bd7-a4
13 Qc2-b1 Nb8-d7
14 g4-g5 Rf8-c8
15 h2-h4 b7-b5
16 a2-a3 Kg8-h8



Position after 16 . . . kg8-h8

(Even Cray Blitz can make typical pointless computer moves.)

17 Nf1-h2 Ba4-b3
18 Nf3-d2!

(Black threatened 18 . . . Qa5-a4 followed by . . . Bb3-c2. Now the knight can eliminate the bishop if Black plays his queen to a4.

18 . . . Nd7-b6
19 Nh2-g4 Bb3-a4
20 Nd2-f3 Rc8-g8
21 h4-h5 Ba4-b3
22 Nf3-d2 Qa5-a4
23 Nd2xb3 c4xb3
24 g5-g6 f7xg6
25 h5xg6 h7-h6
26 Qb1-d3

(Now Black cannot prevent a lethal sacrifice at h6.)

26 Be7-h4
27 Rh1xh4 Resigns

END