Welcome to Micro Chess

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 be 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 casting 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.
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 does 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


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
2  c2-c3
(NAn unusual move designed to take Black out of its opening book.)
2  e7-e6
3  Ng1-f3
4  c2-e3
5  Bf8-e7
6  Qd1-c2
0-0

Position 16...

(An unusual move designed to take Black out of its opening book.)
2  e7-e6
3  Ng1-f3
4  c2-e3
5  Bf8-e7
6  Qd1-c2
0-0

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

Position after 6...

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

Position 7...

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

Position after 7...

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

Position after 8...

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

Position after 9...

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

Position after 10...

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

Position after 11...

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

Position after 12...

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

Position after 13...

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

Position after 14...

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

Position after 15...

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

Position after 16...

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

Position after 17...

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

Position after 18...

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

Position after 19...

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

Position after 20...

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

Position after 21...

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

Position after 22...

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

Position after 23...

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

Position after 24...

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

Position after 25...

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

Position after 26...

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

Position after 27...

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