# Why chess programs find good moves, but barely understand chess after all.

**By Thomas Hall** **in June of 2008**

**Contents**

## 1.0 Introduction

Actually, the success of today's chess programs is undeniable: In 2006 Fritz beat world champion Kramnik in a match with six games by 4:2 and only recently Grandmaster Ehlvest lost in a rapid-chess match against Rybka 2.5:5.5, despite the initial advantage of a surplus pawn for the human opponent.

Every user however, who lets a strong chess engine play against itself in analysis mode, will notice that the engine's assumptions don't often come true, especially when pursuing an opening variant from the start. Moves estimated as quite good initially may turn out to be bad later. Why is this?

The program has fallen victim to the so-called "horizon effect" which affects all engines to the present day. To explain this it is worth examining the search methods of chess programs more closely.

### 2.0 The Minimax-Method

As early as 1950 Claude Shannon published the minimax algorithm [3], which is based on the fact that chess is a game with complete information - chance does not play a role - and both players know all the information necessary to play the game from the start to finish. The clause from game-theory that in such two-person games the sum of the gains (i.e. advantages) and losses (i.e. disadvantages) is always zero was already published in 1928 by John v. Neumann [1]. So a good move means that the opponent has got only worse moves, all of them in all variants leading to favorable positions which are therefore unfavorable for the opponent. Now we "only" have to define a favorable position in chess, and there is already a method to simply try out the moves more or less exhaustively on an internal board in the following way:

Suppose it's White's turn to move. Then the white moves are successively executed. Each of these moves leads to a position where it's Black's turn to move. If we execute the first move of Black's choices after White's first move we have completed 2 plies of the first variation. A ply in chess is a half-move; a complete chess move consists of the moves of White and Black. The following diagram shows a 2-ply game tree:

```
Maximum        W
             / | \
            /  |  \
Minimum    B   B   B
          /|\   \   \
         / | \   \   \
        W  W  W   W   W
        4 -3  2   5  -1
```

As you recognize, White has 3 choices to move in the start position. After White's first move Black has 3 choices, otherwise Black only has one choice. Typical chess positions naturally have many more, approx. 35 choices to move on average, but for now it's only about principle here. So this diagram presents a game tree that starts with the root node - that is the start position - and ends in five possible leaf nodes - the end positions.

For each of the possible leaf nodes, the chess engine's so-called evaluation function calculates a number indicating the merit of the respective position, so e.g. positive values for positions in favor of White and negative values for those in favor of Black.

According to the Minimax principle, White, when striving for the "best" move, must assume that Black will also select the "best" move, i.e. Black minimizes its positional values while White maximizes its own. In the above diagram Black would pick the move leading to the position with value -3 (i.e. the minimum) in the first subtree, and would pick the moves with the values 5 and -1 in the middle and right subtree, respectively. However White selects the maximum of these values, so from -3, 5, and -1 the maximum would be 5 and the decision would be made in favor of the middle subtree. A chess program would therefore assign a value +5 to the main variant.

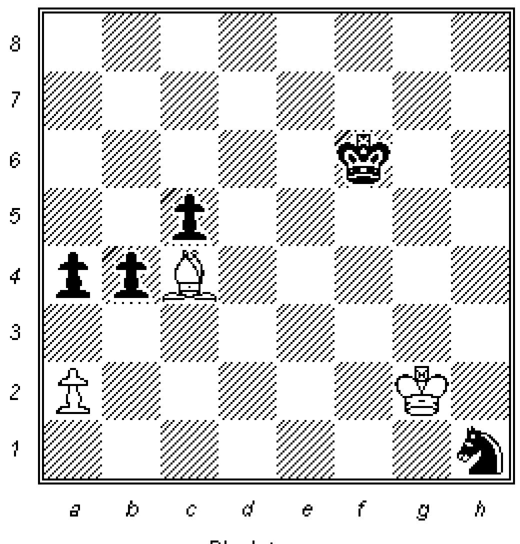
In chess the number of leaf nodes in the game tree quickly grows very large, and explodes exponentially. With n plies this amount would result in $n^{35}$ leaf nodes. To make the process more manageable in practice, it was crucial to discover a possibility to cut off whole subtrees in order to be able to evaluate far fewer leaf nodes. Prerequisite for this possible pruning is move sorting by likely quality (e.g. first exchange moves then moves attacking opponent’s pieces etc.). The procedure that really does the task is called Alpha-beta pruning. It is described more precisely in a Wikipedia article (http://en.wikipedia.org/wiki/Alpha-beta_pruning), among other references. For this discussion it is important that this method doesn't alter the essence of the Minimax algorithm.

The quality of today's chess programs is mainly determined by the type of evaluation functions used because it is only here that the program makes decisions about the assessment of good or bad positions. Early implementations only accounted for a few criteria in this context, particularly counting the material and recognition of mate and stalemate.  But it was discovered very quickly that this was nowhere near adequate. Today evaluation function factors also consider factors like king's security, pawn structures and space advantage, to name just a few.

## 3.0 The horizon effect

Now the phenomenon of the above mentioned "horizon effect" should become clear. The leaf nodes of the game tree are forming the horizon beyond which the chess program only rarely knows what will subsequently develop on the board. One of these exceptions is the search for so called quiescent positions: When pieces are captured in a leaf node, this variation will always be extended by pursuing all exchange moves until a quiescent position without captures is reached. The clear intentention here is to avoid material loss beyond the horizon.

As an illustration of the horizon effect consider the following position taken from the documentation of Salomon Chess [2]:



Black to move

FEN[1]: 8/8/5k2/2p5/ppB5/8/P5K1/7n b - - 0 1

As you may notice upon closer examination Black's best move is 1… Ke5 with the intention of advancing the king to his own queenside pawns in order to clear a path for pawn promotion. If you however set the play level of the engine to fixed search depth of 4 plies, each program using the Minimax method (and these make up the overwhelming majority) makes the bad move 1... b3. What happened?  Among the variants detected is White's capture of the knight and White's capture of the pawn. According to the Minimax algorithm, at this moment taking the knight is better for White because the engine stops any further computation and doesn't notice the consequences. After 4 plies the program is left in inky blackness; therefore this mistake occurred enabling White to ultimately reach a draw.

In principle, a chess program behaves like a blind man who is feeling his way along the curb with his cane and only notices afterwards that something crucial has happened, e.g. the curb ended and he arrived at a crossing. When starting from the opening move,  it is nearly impossible to reach a good middle game position without additional information due to the tremendous number of moves. Therefore all successful chess programs use so-called opening books where, according to current opening theory, playable variations for both sides are stored.

For the same reason, endgame tables are used. In these tables, all moves for all positions with (very) few men on the board are stored as results of preceding month-long computations. In these tables an engine can directly look up whether a certain move wins or not.  If the move wins, it can also look up how many moves are needed to reach checkmate.

[1] The FEN (Forsyth-Edwards Notation) is a machine-recognizable representation of any chess position,  compare e.g.  http://en.wikipedia.org/wiki/FEN

A chess program with no such table to consult for a given move will inevitably examine a plethora of unreasonable move combinations. Though an apparent sheer waste of time, this does imply that tactical maneuvers within the game tree's scope (i.e. on the near side of the horizon) are not overlooked. This fact decisively contributes to the considerable success of chess programs over human opponents.

During this quite aimless search the same positions can occur more than once. In order to avoid evaluating these positions twice, so called hashtables are employed internally in order to store intermediate results of previous evaluations. The user can take advantage of these tables when analysing games, more of this later.

## 4.0 The evaluation function and its limits

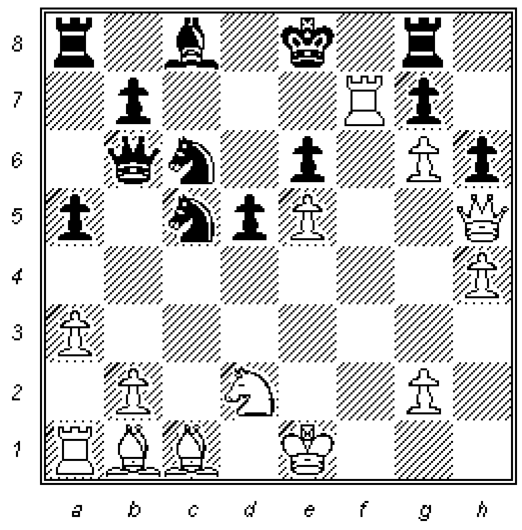
The quality of a chess program working with the Minimax method can thus be traced to the quality of its evaluation function. Here programmers are confronted with a considerable challenge: On the basis of the visible board position, they have to estimate which side is better for all game stages, even with equal material on the board. Often it's more difficult to assess an advantage if the material is not balanced. Here it is necessary to map all characteristics  to a unique numerical value.

The pure material values of the men on the board can be accumulated easily, but then positive values for positional advantages like open rook files and knight outposts or points of negative values for backward pawns and unprotected kings must be assigned. With the issue of king security alone a clear dilemma emerges: Particularly in the middle game the king should be preferably protected behind a line of pawns, but this piece becomes an active piece in the endgame and then it is disadvantageous for the king to remain at the edge of the board. This raises the question of where exactly the endgame begins, and if this can be defined by the remaining material alone. There are more matters of conscience for the programmers.  For in order to eventually get the number of the position value, dynamic elements like piece mobility have to be brought in line with static elements like material values.

## 4.1 Sacrifices

In the course of time, programs with sophisticated positional evaluation functions turned out to fare better. This became particularly apparent with the emergence of Rybka. Nevertheless, it's questionable whether this is the last word on the subject. For example, consider the position emerging on the board after the following moves of the Tarrasch variation of the French defense (taken from an analysis by the author):

1. e4 e6 2. d4 d5 3. Nd2 Nf6 4. e5 Nfd7 5. f4 c5 6. c3 cxd4 7. cxd4 Nc6 8. Ngf3 f6 9. Bd3 fxe5 10. dxe5 Nc5 11. Bb1 Be7 12. a3 a5 13. h4 Qb6 14. Ng5 Kf8 15. Qh5 Bxg5 16. fxg5 Ke7 17. g6 h6 18. Rf1 Rg8 19. Rf7+ Ke8



White to move

FEN: r1b1k1r1/1p3Rp1/1qn1p1Pp/p1npP2Q/7P/P7/1P1N2P1/RBB1K3 w Q - 0 20

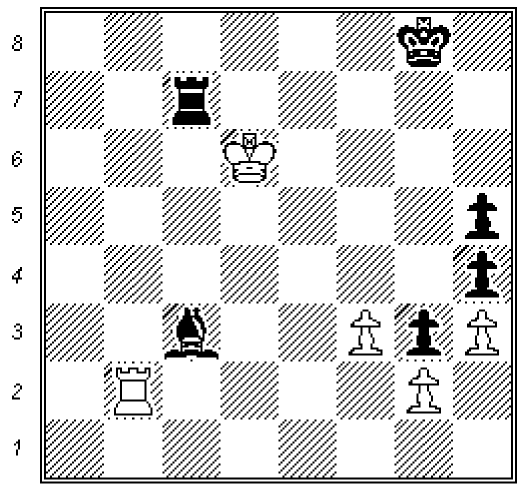
Modern engines in particular believe that Black is a bit better off here and they typically want to play 20. Qe2?. White however has a clear winning line after 20. Rxg7!, for example:

20. Rxg7 Rxg7 21. Qxh6 Rg8 22. Qh7 Ne7 23. Nf3 Kd7 24. Bg5 Re8 25. g7 Qd8 26. Bg6 Rg8 27. Rc1 b6 28. b4 axb4 29. axb4 Na6 30. Bd3 Nc7 31. Bxe7 Kxe7 32. Rxc7+ Qxc7 33. Qxg8 Qc1+ 34. Kf2 Qf4 35. Qh7 Ra2+ 36. Be2 Qf7 37. g8=Q 1-0

Granted, White's advantage after 20. Rxg7 becomes apparent after only 5 or 6 moves, but modern engines reach this search depth very fast. So what happens here? Because positional factors control the evaluation function, sacrificial moves are inserted at the end of the list of moves to be examined. The search proceeds incrementally, increasing the search depth slowly, that is ply after ply, in order to gain hints for the new sorting sequence in the current search depth from the preceding examination. Since the value of the sacrifice manifests itself after a relatively long period of time it's quite possible that the opportunity to play Rxg7 is missed by a cut-off operation of Alpha-beta pruning. So gearing the evaluation function towards predominantly positional factors would have precluded the examination of a move that makes no sense at first glance. Interestingly enough, several older programs such as Fritz 9 and even Fritz 10 still find the rook sacrifice, whereas Fritz 11 as well as Rybka don't locate this move.

## 4.2 End games

Another interesting position which most notably highlights the programs' weaknesses in the endgame is the following one:



Black to move

FEN: 6k1/2r5/3K4/7p/7p/2b2PpP/1R4P1/8 b - - 0 1

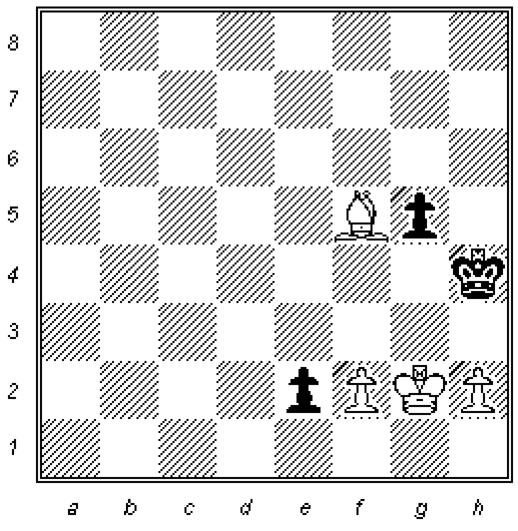
The chess playing human recognizes relatively fast that taking the rook on b2 causes an exchange of rooks which bestows a surplus bishop on Black but leaves Black rather hapless. For how shall the dark squared bishop take one of the pawns residing on light squares? Only the black king could accomplish this, but he is barred by the white king from getting to g2, the Achilles' heel of White's configuration. Therefore the correct plan to win is forgoing the rook exchange and then bringing the bishop to f2 in order to cut pawn g2 off from the rook’s protection.

Most engines however, promptly take the rook on b3 ending up in a draw after 2. Kxc7. The short term material advantage prevails over the long-term winning route which can be only detected by certain computations in the evaluation function. For the Alpha-beta procedure must have noticed success or failure in the first place and this can take a long time with the number and length of variants to be excluded. By the way, Zappa Mexico II seems to invest more effort here than others because this chess program found the key move 1… Rc4 rather quickly but needed a longer time in order to find the victorious maneuver 1... Rc4 2. Re2 Kf7 3. Ra2 Rd4+ 4. Kc5 Rd1 5. Kc4 Be1 6. f4 Bf2.

## 4.3 Fortresses

Something that is completely beyond the abilities of today's chess programs is the discovery of fortresses. A position containing a fortress is characterized by a configuration that cannot be "broken up" by the opponent even with superior material, i.e. the opponent would have to relinquish its material advantage in order to make any progress in the game. But then the player with the fortress could draw or even win the game. This means that a position with a fortress is at worst a draw.

Example:



White to move

FEN: 8/8/8/5Bp1/7k/8/4pPKP/8 w - - 0 1

After 1. Bg4 e1=Q 2. h3 the black king is incarcerated and the black queen alone cannot bring about any change, because it's completely sufficient to have the bishop shuttle between f3 and g4. Humans would immediately offer a draw but a chess program would notice the draw only after 50 moves (by intervention of the 50-move rule).

## 5.0 Conclusions

Here the biggest weakness of all current chess programs becomes evident: It is the desultory trial and error while searching for moves always with the hope that every now and then something to be exploited will be discovered. For the most part nonsensical series of moves are examined. This computation is unnecessarily costly and after all only feasible due to the speed of current processors. In spite of this computing power, very long variants absolutely cannot be tackled, although in the endgame extremely long lines to checkmate can be found, as can be gleaned from the endgame tables. An engine which wants to avoid trying out move variants in a hit-or-miss fashion, needs far more chess knowledge than that utilised in today's programs. Current programs cannot explain why a certain move is either good or bad because causal chains are not followed, thus the move and countermove of a variant is only the indirect result of a position value return to the game tree's root. Yet more is required in order to master chess. It's actually amazing that these inadequate methods generally still compute useable moves.

In closing we return to the hashtables mentioned above. Because these are not automatically deleted when a user steps through an already computed variant using the graphical frontend in analysis mode, the horizon effect can actually be mitigated in this way: If the program detects hashtable values of recurring positions that were already computed down the line, it can use them to find other, hopefully better, moves.

But overall that's small comfort. When it comes to the truth in chess, one can currently only rely on the endgame table bases. Ultimate certainty concerning the quality of chess moves can, in general, not be gained by the Alpha-beta algorithm.

## 6.0 Literature

[1] Von Neumann, John: Zur Theorie der Gesellschaftsspiele Math. Annalen. 100 (1928) 295-320

[2] Rehatschek, Herwig: Graz, Techn. Univ., Inst. f. Informationsverarbeitung, Diplomarbeit. v. 1993

[3] Shannon, Claude: Philosophical Magazine, Ser.7, Vol. 41, No. 314 – März 1950

Special thanks to Dr. Roland Schregle and Kevin Currun for helping me with the translation from German to English