Chess Programming Part VI: Evaluation Functions

by François Dominic Laramée

It's been six months, and I know sometimes it must have felt like I would never shut up, but here we are: the sixth and last installment of my chess programming series. Better yet: my Java chess program, primitive though it may be, is now complete, and I shipped it to Gamedev along with this, which proves that I know (a little bit of) what I've been talking about.

This month's topic, the evaluation function, is unique in a very real sense: while search techniques are pretty much universal and move generation can be deducted from a game's rules and no more, evaluation requires a deep and thorough analysis of strategy. As you can well imagine, it is impossible to assess a position's relative strengths if we don't know what features are likely to lead to victory for one player or the other. Therefore, many of the concepts discussed here may apply to other games in very different fashion, or not at all; it is your job as programmer to know your game and decide what the evaluator should take into consideration.

Without further delay, let us take a look at some board evaluation metrics and at how they can be used to evaluate a chess position.

Material Balance

To put it simply, material balance is an account of which pieces are on the board for each side. According to chess literature, a queen may be worth 900 points, a rook 500, a bishop 325, a knight 300 and a pawn 100; the king has infinite value. Computing material balance is therefore a simple matter: a side's material value is equal to

$$MB = Sum(Np * Vp)$$

where Np is the number of pieces of a certain type on the board and Vp is that piece's value. If you have more material on the board than your opponent, you are in good shape.

Sounds simple, doesn't it? Yet, it is by far the overwhelming factor in any chess board evaluation function. CHESS 4.5's creators estimate that an enormous advantage in position, mobility and safety is worth less than 1.5 pawns. In fact, it is quite possible to play decent chess without considering anything else!

Sure, there are positions where you should sacrifice a piece (sometimes even your queen) in exchange for an advantage in momentum. These, however, are best discovered through search: if a queen sacrifice leads to mate in 3 moves, your search function will find the mate (provided it looks deep enough) without requiring special code. Think of the nightmares if you were forced to write special-case code in your evaluator to determine when a sacrifice is worth the trouble!

Few programs use a material evaluation function as primitive as the one I indicated earlier. Since the computation is so easy, it is tempting to add a few more features into it, and most people do it all the time. For example, it is a well-known fact that once you are ahead on material, exchanging pieces of equal value is advantageous. Exchanging a single pawn is often a good idea, because it opens up the board for your rooks, but you would still like to keep most of your pawns on the board until the endgame to provide defense and an opportunity for queening. Finally, you don't want your program to panic if it plays a gambit (i.e., sacrifices a pawn) from its opening book, and therefore you may want to build a "contempt factor" into the material balance evaluation; this allows your program to think it's ahead even though it is behind by 150 points of material or more, for example.

Note that, while material balance is highly valuable in chess and in checkers, it is deceiving in Othello. Sure, you must control more squares than the opponent at the end of the game to win, but it is often better to limit his options by having as few pieces on the board as possible during the middlegame. And in other games, like Go-Moku and all other Connect-N variations, material balance is irrelevant because no pieces are ever captured.

Mobility and Board Control

One of the characteristics of checkmate is that the victim has no legal moves left. Intuitively, it also seems better to have a lot of options available: a player is more likely to be able to find a good line of play if he has 30 legal moves to choose from than if he is limited to 3.

In chess, mobility is easily assessed: count the number of legal moves for each side given the position, and you are done. However, this statistic has proven to be of little value. Why? For one thing, many chess moves are pointless. It may be legal to make your rook patrol the back rank one square at a time, but it is rarely productive. For another, trying to limit the opponent's mobility at all costs may lead the program to destroy its own defensive position in search of "pointless checks": since there are rarely more than 3-4 legal ways to evade check in any given position, a mobility-oriented program would be likely to make incautious moves to put the opponent in check, and after a while, it may discover that it has accomplished nothing and has dispersed its forces all over the board.

Still, a few sophisticated mobility evaluation features are always a good thing. My program penalizes "bad bishops", i.e., bishops whose movement is hampered by a large number of pawns on squares of the same color, as well as knights sitting too close to the edges of the board. As another example, rooks are more valuable on open and semi-open files, i.e., files where there are no pawns (or at least no friendly ones).

A close relative of mobility is board control. In chess, a side controls a square if it has more pieces attacking it than the opponent. It is usually safe to move to a controlled square, and hazardous to move to one controlled by the opponent. (There are exceptions: moving your queen to a square attacked by an enemy pawn is rarely a good idea, no matter how many ways you can capture that pawn once it has done its damage. You may also voluntarily put a piece in harm's way to lead a defender away from an even more valuable spot.) In chess, control of the center is a fundamental goal of the opening. However, control is somewhat difficult to compute: it requires maintaining a database of all squares attacked by all pieces on the board at any given time. Many programs do it; mine doesn't.

While mobility is less important than it seems to the chess programmer, it is extremely important in Othello (where the side with the fewest legal moves in the endgame is usually in deep trouble). As for board control, it is the victory condition of Go.

Development

An age-old maxim of chess playing is that minor pieces (bishops and knights) should be brought into the battle as quickly as possible, that the King should castle early and that rooks and queens should stay quiet until it is time for a decisive attack. There are many reasons for this: knights and bishops (and pawns) can take control of the center, support the queen's attacks, and moving them out of the way frees the back rank for the more potent rooks. Later on in the game, a rook running amok on the seventh rank (i.e., the base of operations for the opponent's pawns) can cause a tremendous amount of damage.

My program uses several factors to measure development. First, it penalizes any position in which the King's and Queen's pawns have not moved at all. It also penalizes knights and bishops located on the back rank where they hinder rook movement, tries to prevent the queen from moving until all other pieces have, and gives a big bonus to positions where the king has safely castled (and smaller bonuses to cases where it hasn't castled yet but hasn't lost the right to do so) when the opponent has a queen on the board. As you can see, the development factor is important in the opening but quickly loses much of its relevance; after 10 moves or so, just about everything that can be measured here has already happened.

Note that favoring development can be dangerous in a game like Checkers. In fact, the first player to vacate one of the squares on his back rank is usually in trouble; avoiding development of these important defensive pieces is a very good idea.

Pawn Formations

Chess grandmasters often say that pawns are the soul of the game. While this is far from obvious to the neophyte, the fact that great players often resign over the loss of a single pawn clearly indicates that they mean it!

Chess literature mentions several types of pawn features, some valuable, some negative. My program looks at the following:

- Doubled or tripled pawns. Two or more pawns of the same color on the same file are usually bad, because they hinder each other's movement.
- Pawn rams. Two opposing pawns "butting heads" and blocking each other's forward movement constitute an annoying obstacle.
- Passed pawns. Pawns which have advanced so far that they can no longer be attacked or rammed by enemy pawns are very strong, because they threaten to reach the back rank and achieve promotion.
- Isolated pawns. A pawn which has no friendly pawns on either side is vulnerable to attack and should seek protection.
- Eight pawns. Having too many pawns on the board restricts mobility; opening at least one file for rook movement is a good idea.

A final note on pawn formations: a passed pawn is extremely dangerous if it has a rook standing behind it, because any piece that would capture the pawn is dead meat. My program therefore scores a passed pawn as even more valuable if there is a rook on the same file and behind the pawn.

King Safety and Tropism

We have already touched on king safety earlier: in the opening and middle game, protecting the king is paramount, and castling is the best way to do it.

However, in the endgame, most of the pieces on both sides are gone, and the king suddenly becomes one of your most effective offensive assets! Leaving him behind a wall of pawns is a waste of resources.

As for "tropism", it is a measure of how easy it is for a piece to attack the opposing king, and is usually measured in terms of distance. The exact rules used to compute tropism vary by piece type, but they all amount to this: the closer you are to the opposing king, the more pressure you put on it.

Picking the Right Weights

Now that we have identified the features we would like to measure, how do we assign relative weights to each?

That, my friends, is a million-dollar question. People can (and do) spend years fiddling with the linear combination of features in their evaluation functions, sometimes giving a little more importance to mobility, sometimes de-emphasizing safety, etc. I wish there were an absolute solution, but there isn't. This is still very much a matter for trial and error. If your program plays well enough, great. If not, try something else, and play it against the old version; if it wins most of the time, your new function is an improvement.

Three things you may want to keep in mind:

- It is very difficult to refine an evaluation function enough to gain as much performance as you would from an extra ply of search. When in doubt, keep the evaluator simple and leave as much processing power as possible to alphabeta.
- Unless you are after the World Championship, your evaluator does not need to be all-powerful!
- If your game plays really fast, you may want to try evolving an appropriate evaluator using a genetic algorithm. For chess, though, this would likely require thousands of years!

Next Month

Well, there ain't no next month. This is it.

If I wanted to drag this series even longer, I could write about opening books, endgame libraries, specialized chess hardware and a zillion other things. I could, I could. But I won't. Some of these topics I reserve for the book chapter I will be writing on this very topic later this Fall. Others I just don't know enough about to contribute anything useful. And mostly, I'm just too lazy to bother.

Still, I hope you enjoyed reading this stuff, and that you learned a useful thing or two or three. If you did, look me up next year at the GDC or at E3, and praise me to whomever grants freelance design and programming contracts in your company, will ya?

Cheers!

François Dominic Laramée, October 2000

Discuss this article in the forums

Date this article was posted to GameDev.net: **10/8/2000**
(Note that this date does not necessarily correspond to the date the article was written)

See Also:
Gaming