Che.ss
CS 154 Project

Nishant Kumar Singh (120050043)
Pratyaksh Sharma (120050019)

18 April 2013

1 Introduction

Chess is a two player strategy game, played on a chequered gameboard. It is one of the most popular board games, and is played by millions of players at their own leisure, in tournaments, or even online. A chess enine is a computer program capable of calculating chessboard positions and generating intelligent moves.

IBM's Deep Blue[1] was the first chess computer to defeat a reigning World Chess Champion in a match. Advances in computer hardware and algorithms have resulted in making these computers stronger each year. Current chess engines are able to overcome even the strongest human players under standard tournament conditions.

Here we attempt to manufacture a basic chess engine.

2 Approach to the problem

We break the problem into parts:

1. Interface for interacting with the human user.
2. Checking and generating legal moves.
3. Tree search based on minimax and alpha-beta pruning
4. Evaluating and comparing chessboard positions

[1]http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

3 On XBoard

For the interface, we use a free GUI called XBoard[2]. Our chess engine communicates with XBoard using the Chess Engine Communication Protocol[3]. XBoard was created to provide a standard GUI for chess engines, and most modern day chess programs are chess engines which use XBoard (or its alternatives) to interact with the user.

Further XBoard is capable of playing two chess engines one vs one on the same board.

Our program uses a two-dimensional vector for the board representation. There are various functions which can effect piece-moving on the board. The evaluator function returns a numerical value for the strength of a given board position for a given side based on several heuristics and chess strategy notions.

4 Implementing rules of Chess

This part is pretty straightforward. The function *is-legal?* will check whether the given move is legal or not, given a board position. Then there is a procedure which *generates* possible valid-moves for building the chess tree. Castling, pawn promotion and en passant rules have also been implemented.

5 Making the best move

The minimax rule is well suited for a game of chess. We aim at minimizing the possible loss for a worst case (maximum loss) scenario.

A game tree is generated, albeit implicitly (no *structs* have been used). Now, not all branches of the tree are expanded, since some are surely worse than the others. We rely on the alpha-beta pruning algorithm for this.

Looking 3 moves into the future, in other words, a tree of depth 3 (can be set to any natural number) we search for the best possible move, based on the above strategy. Chessboard positions are evaluated by the *eval-board* procedure.

[2]http://www.gnu.org/software/xboard
[3]www.gnu.org/software/xboard/engine-intf.html

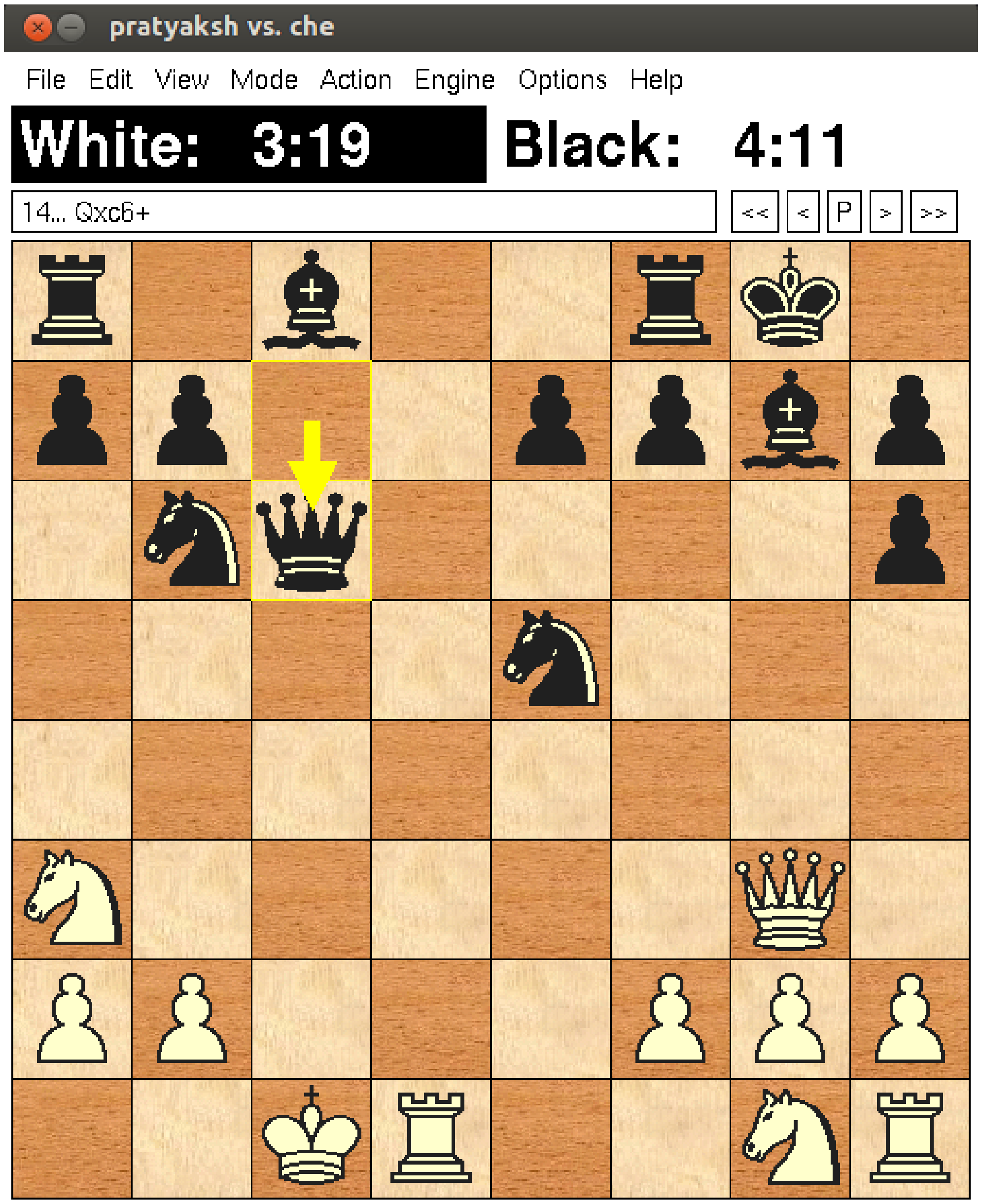

Figure 1: The computer playing black.

6 Heuristics

The basic function of evaluation function[4] is to return a number signifying the board position and from a particular player's perspective. Each of the pieces have some values attached with them which are relative to the value of the pawn. They also have attack-value. This value shows the strength of the piece's attack on other piece. The pawn has the highest attack-value.

The *eval-piece-score* function calculates the position piece by piece based on the given board and the endgamephase (a boolean). The evaluation is based on a number of factors, one of them being Piece Square Tables[5]. This table is the table of numbers which signifies the strength (or the weakness) of a piece on a particular position.

Then there are functions to calculate attack and defence on a particular piece on the given board. These are actually instances of a function called chhof which is a multi-purpose function for calculating attack on a particular piece. This also helped in reduction of number of lines of code as it also is being used for deciding whether a particular side is in check or not.

Evaluation also considers the pawn structure, doubled pawns, isolated pawns etc.

While evaluating the board the piece values are first added to the score. Then attack-values and defend-values are calculated and added and subtracted respectively in the score. While doing this the fact that who is to make move next is also taken into consideration. If the attack on any piece other than pawn is due to a pawn, then a penalty is being imposed. Again if attack value is more than defended value another penalty is imposed.

Finally while evaluating the board, the eval-piece function is being called repetitively over all the board positions and the values are added or subtracted depending on what piece is it, if it is white then the score is being subtracted else is added. The computer or the user is minimally penalised for check. Also if the king moves without castling, a penalty is imposed.

[4]en.wikipedia.org/wiki/Chess_strategy

[5]chessprogramming.wikispaces.com/Piece-Square+tables

7 Limitations and bugs

1. The search depth is currently 3. Beyond 3 the time taken isn't reasonable for timed chess. The depth can be increased significantly if we use good move ordering in tree searching.

2. A large number of bugs have been corrected, through repeated testing. Although a few more might come up.

8 Future additions

1. Instead of simple alpha-beta pruning, the MTD-f[6](uses memoization) algorithm can be implemented.

2. Iterative deepening.

3. Standard openings can be defined, so that there is no need for game tree during the first few moves.

4. Feature of pondering. Doing calculations while the human player is thinking of a move.

5. Tweaking the heuristics based on past wins/losses. Learning from defeats.

6. Taking help of endgame tablebase[7].

7. Feature of undoing moves.

[6]people.csail.mit.edu/plaat/mtdf.html

[7]en.wikipedia.org/wiki/Endgame_tablebase#Computer_chess