Israel Bond

Program Architecture

My implementation of Mini Chess was constructed using C++ with some C language components for server management. To build a running version of my program a Makefile has been included which will compile the most optimal for of my program into an executable named Minichess. Using this with the proper commands needed for the IMCS server will populate a game on the server with a version of my program which performs Iterative Deepening, Alpha-Beta Pruned Negamax search to evaluate piece moves.

Feature set

Most of my implementation relies on the standard set of required features. I Implemented a piece list which stores a structure of moves for the specific piece. This limited the number of pieces stored in memory from 20 to only 7, reducing memory costs as well as redundant structures for identical pieces. Also used the Do-Undo method for making & unmaking moves on the standard board representation which uses a 2D array of characters.

If I had two more weeks to produce code for this project I would have attempted to implement a transposition table where alpha-beta pruned negamax search results are stored for future use. I would attempt to use Zorbrist hashing method to manipulate the table, keeping track of mine & my opponents moves and doing only incremental updates as needed.

Performance

My mini chess player currently plays better than a human can yet is only considered average as my implementation consists of the standard methods required for this course. My player can perform time limited Iterative Deepening, Alpha-Beta Pruned Negamax search with 2 distinct times to perform Iterative Deepening. These two values allow programed player to save a little time during the first portion of the game and allow more time spent generating board states towards the end of the game. This allows my player to find winning and losing states earlier, thereby allowing my player to define these states and attempt to steer towards a winning game or towards a draw game as losses are undesirable.

Reliability

Throughout the term I have been working on this project in pieces, having two previous assignments which consisted of having a Nagamax search for the game Haxapawn. Piece moves and positions were evaluated. Negamax would perform a search on possible move position combinations in a Depth First Search Algorithm, changing the player on turn until a winner was defined.

Testing was done to insure:

- Correct game play
- Following the rules
  - Board
  - Piece

Tests consisted of:

1. Board Boundaries
    a. Valid board positions
    b. Invalid/out of bounds positions
2. Board State changes
    a. Move
    b. Un-move
3. Piece positioning
    a. No Attacks on friendly pieces
    b. Possible moves by piece
4. Legal Moves
    a. Attack on opponent
    b. Change in board position
        i. Blocked path
        ii. Unblocked path

This test set was used with the mini chess player to provide the player with valid board states and move sets.

Testing that My program could get onto the sever and play games across the network was to simply do just that. I used the implemented C code provided to interact with the server. Testing consisted of:

- Being able to Accept and Offer a game playing as either color, Black or White.
- Reading and interpreting information passed over the network.
    - Opponent move
    - Side on Turn
    - Determine Player color

These tests validate the ability to communicate over the network, playing our program against an opponent.

Known Defects

No specifically recognized defects other than weak move heuristics. I have only specified moves based on points the move is worth (piece being removed) and an offensive King (attecks before other pieces unless that attack is on the opponents King).