COSC 3P71 Term Project

By: Drayton Williams & Cameron Hammel

Creating an artificially intelligent chess opponent is a common endeavor in the field of computer science. Many people grow up playing chess and learning various strategies to apply new logic and to design new plans for victory. Chess is an excellent education tool of sorts which helps develop and train the brain, especially for youth, as the experience gained by playing continuously can contribute greatly to improving overall performance in the long run. When developing AI, the learning aspect is critical to push the limits as to what it can do and how it can improve itself. With a complex, logic-heavy game such as chess, one can see why it's such a common focus of the industry.

This program utilizes a heuristics-based AI solution to analyze and rank possible board positions and make movements based on the resulting positions. The program searches through a game tree using alpha-beta pruning for the optimal move at that point in time. Since searching all possible moves would be extremely exhausting for the algorithm and computationally expensive, the user is to provide a tree depth value to decide how deep the search will go (our recommended value is 4). This search only utilizes the moves that are legal, as individual "generateMovesX" (where the 'X' is the first letter of either Pawn, Knight, Bishop, Rook, Queen, or King) methods determine which moves are legal for each piece in their current state. Throughout the algorithm, a method called flipBoard() is called. This method allows for the AI to evaluate the board and perform its move search without needing to duplicate every move generation method which had previously only been coded for white pieces. For example, after the player first makes their move, the method "flipBoard" is called. This is done so that when it is the AI's turn to evaluate the board, it is looking at the board as if it was the player controlling the white pieces and that the move previously performed by the other player was done on the black pieces. The "flipBoard" method splits up the board into two halves, with the first half being the board squares indexed from 0 to 31 and the second being indexed from 32 to 63. As the method is ran, it will look at the first index of the side currently being accessed and the last index of the opposite. As pieces are iterated over, the method will change the value of the piece occupying that space to the colour and the piece exactly opposite of the board, effectively rotating the board and allowing anyone who is playing to be evaluated as the 'white piece' player. Additionally, the "safeKingMove" method for the AI determines which moves result in a check against the AI and attempts to avoid them.

The user-controlled side also utilizes the generateMovesX methods in order to provide the user with all of their options, for the sake of simplicity and convenience. Possible positions produced by the generateMovesX methods are formatted as (x1,y1,x2,y2,captured piece), where x1 and y1 represent the current piece location, x2 and y2 represent the new location, and captured piece represents the piece that would be captured by committing to the move (or a 'P' if a pawn would be promoted by committing to the move). The user then commits to a move by inputting the desired move using the same (x1, y1, x2, y2, captured piece) format, with an updated board position printed in the console for visual assistance. If either the user or the AI are in a board

state such that the king cannot escape check, (checkmate) the game will end, and the victor will be displayed in the console.

Inside the user interface, the two sides of the board are represented by black and white, and are printed out based on uppercase and lowercase lettering (AI's king would be inspected in the code by colour and seen as "K", whereas the human player's king would be seen as "k"). The board itself is simply represented by a 2-dimensional array of BoardTile objects, which represent each tile on the board and are assigned colour values (i.e. black and white) based on their individual position in each row. When moves are generated, the board is iterated searching for pieces of a target colour and possible moves for each piece found are recorded as they're discovered (for 6 possible types of pieces). There also exists a NullPiece with its own special piece colour and name to help avoid possible null pointers when scanning the board. The board is printed out based on the lowercase/uppercase lettering for pieces, dashes "-" for empty tiles (NullPiece), and other formatting characters for visual clarity such as brackets for each row "[", "]", and vertical bars between tiles "|". A sample output looks like this:

```
[R|N|B|Q|K|B|N|R]
[P|P|P|P|P|P|P|P]
[-|-|-|-|-|-|-|-]
[-|-|-|-|-|-|-|-]
[-|-|-|-|-|-|-|-]
[-|-|-|-|-|-|-|-]
[p|p|p|p|p|p|p|p]
[r|n|b|q|k|b|n|r]
```

Our implementation of a board evaluation can be broken down into four parts which contribute to the final rating that is used to determine the AI's next move on the board. The board is evaluated based on an attack score, a material score, a movability score, and a positional score. The attack score consists of evaluating the current chess board to determine which move is preferable based on the piece that is possible to capture. For example, the Queen is one of the higher rated values because it is one of the more valuable pieces to take out on the board. This is in contrast to the pawn, which is assigned the lowest value for simply being a pawn; there are less of them and they are less advantageous to the player if captured (in general cases).

The material score can be produced by evaluating the player's current pieces active on the chess board. This evaluation method looks at the entire board to find the player's pieces and uses centipawn piece values to help build a score. The method contains a special case value that is assigned based on how many bishops are present on the board. A pawn may have been promoted to a bishop, putting that player in a more advantageous position, therefore a greater value is assigned. This plays in contrast to when the player possesses only one pawn on the board, giving them less than desirable odds and assigning a lower value that reflects that.

The movability score is simply calculated by taking the length of the list of total possible moves for the player. If the move list being evaluated is at a length of 0, this means there are no moves available for the player to make and they are either in a position for checkmate or stalemate. The scores assigned to being in either of these positions suggest that it is less preferable to be in the position of checkmate.

The positional score is generated based on the position of the player's own pieces on the board. Using positional values obtained from a simplified evaluation function online, this aspect of the board evaluation will look at each piece and, if finding that piece on a specific point on the board, will add the positional value to the score. The purpose of this positional evaluation is to help encourage certain pieces to occupy specific spots or areas on the chess board. For example, a knight is preferably moved to the center of the board and this is reflected by the value table:

```
// knight
-50,-40,-30,-30,-30,-30,-40,-50,
-40,-20,  0,  0,  0,  0,-20,-40,
-30,  0, 10, 15, 15, 10,  0,-30,
-30,  5, 15, 20, 20, 15,  5,-30,
-30,  0, 15, 20, 20, 15,  0,-30,
-30,  5, 10, 15, 15, 10,  5,-30,
-40,-20,  0,  5,  5,  0,-20,-40,
-50,-40,-30,-30,-30,-30,-40,-50,
```

The positive numbers that increase in value as they get closer to the center make it so a better overall score is produced when testing possible knight moves to that area. For proof of the board evaluation effectiveness, below are board layouts that showcase the AI's move selection before a score system is in place (each evaluation returning a score of zero) and after (all four evaluation techniques being used to create a score):

```
[R|N|B|Q|K|B|N|R]
[P|P|P|P|P|P|P|-]
[-|-|-|-|-|-|-|P]
[-|-|-|-|-|-|-|-]
[-|-|-|-|-|-|-|-]
[p|-|-|-|-|-|-|-]
[-|p|p|p|p|p|p|p]
[r|n|b|q|k|b|n|r]
```

```
[R|N|B|Q|K|B|-|R]
[P|P|P|P|P|P|P|P]
[-|-|-|-|-|N|-|-]
[-|-|-|-|-|-|-|-]
[-|-|-|-|-|-|-|-]
[-|-|-|-|-|-|-|p]
[p|p|p|p|p|p|p|-]
[r|n|b|q|k|b|n|r]
```

This change in evaluation score demonstrates that instead of choosing the first available move and going with that, the AI (represented as capital letters on the board) actually evaluates various factors before making its move. The AI was ultimately able to produce a common first move amongst chess players that can lead to strategic chess game openings like the "Ruy Lopez"[6] or "Giuoco Piano"[6].

While some of the values used within these functions are a culmination of general guidelines used by many others when building chess engines (eg. Piece values using centipawns[4]), many of the values have the ability to be tweaked to an infinite number of combinations to possibly allow for a better board evaluation score.

The chess game itself will repeatedly ask the user to input their choice of move from a list of all possible generated moves. After the player has entered their move, the AI will automatically respond and will display on the console that the "AI is thinking…". During this process, the AI will be going through its alpha beta minimax algorithm trying to find the optimal move in response. A total amount of configurations tested is kept track of while the AI performs this search to help show some of the computing power exhausted behind each move generation. With even a small increase in search depth, the AI can take much longer to produce a move and will have many more board configurations tested. However, this increase in search depth will make for a (in theory) much smarter and more prepared opponent. The game terminates based on two cases. Firstly, the game may terminate if when producing the players move set, the total possible moves is empty, the player is in stalemate and will lose. On the other hand, after the players have successfully made their moves, if the board is absent of either player's king piece a checkmate has occurred.

In conclusion, while far from perfect, our AI chess engine is able to effectively make use of the pieces given to it to come up with moves and play a game of chess. Using various methods of board evaluation and an alpha-beta minimax search algorithm, the AI has been giving the ability to (essentially) think on its own and make logical decisions when playing.

References

1. http://www.learnchessrules.com/ - Used for piece movement rules

2. http://kidchess.com/learn-chess/ - Used for initial board configuration

3. https://medium.freecodecamp.org/simple-chess-ai-step-by-step-1d55a9266977 - Used for possible minimax chess piece weights

4. https://chess.stackexchange.com/questions/2409/how-many-points-is-each-chess-piece-worth - Used for chess piece values (using centipawns)

5. http://chessprogramming.wikispaces.com/Simplified+evaluation+function - Used for chess piece positional ratings

6. https://www.dwheeler.com/chess-openings/ - Used for strategic chess game openings