MEMO

TO: Professor Brian Smith

FROM: Section 10, Group 1, Tommy Crooks, Patrick Faley, John Paden, Cesar Parra-Castro, Matt Peine

DATE: February 20, 2019

SUBJECT: Mathematical Model

The group's design statement is as follows: Simulate a game of chess, with options for 0, 1, or 2 human players, and compare different strategies for playing the game to determine which is the most effective.

Rules of Chess

- General Rules
    - The board is an 8x8 checkerboard. Each side consists of 8 pawns, 2 rooks, 2 knights, 2 bishops, 1 queen and 1 king.
    - Players take turns moving, with white taking the first move.
    - Capturing occurs when a piece moves into a square occupied by a piece of the other team.
- Movement
    - Pawns can move forward 1 square at a time unless that pawn has not moved yet during that game, in which case it can move forward two squares. The pawn can not capture directly ahead; it can capture diagonally forward one space. The pawn cannot move backwards.
    - Rooks can move directly forward, backward or side to side as far as the user desires until it encounters another piece or the edge of the board.
    - Knights are the only piece that can jump over either their side's pieces or the opposing side's pieces. They move two spaces in one direction and one space Check occurs when king can get captured on the opponent's next move
    - A player cannot move into check
    - If a player is currently in check, they must move to get out of check if possible.
    - Checkmate occurs when a player is currently in check and no move will successfully get them out of check. This is how most games of Chess end.perpendicular to that direction.
    - Bishops move diagonally as far as the user desires until they encounter another piece or the edge of the board.

    - The queen can move forwards, backwards, or diagonally as far as the user desires until it encounters another piece of the edge of the board.
    - The king has the same movements as the queen, except its movement in any direction is limited to one square.
- Check and Checkmate
    -
- Stalemate
    - When a player is in a position where they cannot make any legal moves but are not in check, the game is a draw.
- Miscellaneous Rules
    - Castling
        - When the king and the rook are in their original positions and there are no pieces in between them, the user may castle. This involves moving the king two squares towards the rook and placing the rook immediately on the other side of the king
        - Castling cannot occur if
            - ❏ Either the king or the rook has moved during the game
            - ❏ The king is currently in check
            - ❏ The opponent is attacking any of the squares between the king and the rook
    - En Passant
        - When a player's pawn is on the 5th row relative to their own side, and their opponent elects to move their pawn 2 spaces so that it is now directly to the left or the right of the player's pawn, the player may capture that pawn as if it had only moved one space, therefore ending up 1 square forward and 1 square to the left of the right.
    - Promotion
        - When a player advance their pawn to the final row, they exchange the pawn for any other piece of their choice (except for a King).

Strategies/Algorithms

- General Strategy: Minmaxing, Fitness Functions, and Game Trees
    - One of the major strategies the team will implement is the minimax algorithm. In order to use this algorithm, a tree is constructed of all possible board states up to a certain depth. The computer will compute all possible moves it could make, then all moves its opponent can make in response to those moves. This process can be repeated as many times as necessary, constructing a deeper and deeper tree each time. Once the game tree has been constructed, each board position at the final

point is evaluated according to a fitness function. This fitness function takes in the board as input, and outputs a numeric score which corresponds to how good of a position the computer player is in. After applying a fitness function to the tree of possible board states, something similar to Figure 1 below will be generated.

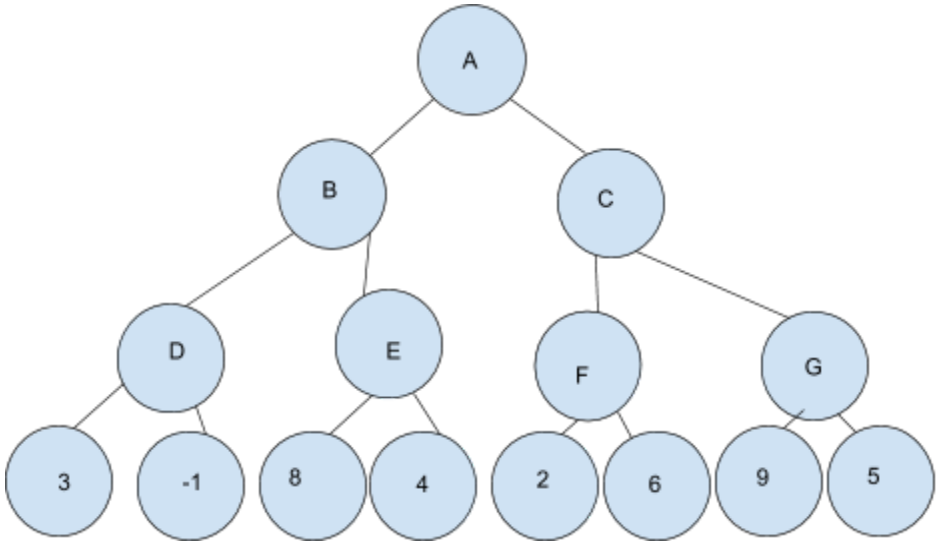

Figure 1. Game Tree Example

Each circular node in this game tree represents a possible board state. The top node represents the current board, with each successive level representing a turn. At each level, a decision is made by either the algorithm or its opponent. The goal of the algorithm is the maximization of the score (corresponding to a good board state for the algorithm), and the goal of the opponent is the minimization of the score (corresponding to a good board state for the opponent). By assuming that the opponent will take the move which will be the worst for the algorithm, the algorithm can choose a move which will guarantee it the best outcome. Here is an example of how the algorithm's thought process would work on Figure 1:

- Nodes D-G: The algorithm first examines node D. Here, it is the algorithm's choice, and it will pick the greater of 3 and -1, so node D assumes a value of 3. By the same logic, E becomes 8, F becomes 6, and G becomes 9.
- Nodes B-C: Here, it is the opponent's choice, meaning they will choose the lower of the two options. B will become 3, the smaller of 3 and 8, while C becomes 6, the smaller of 6 and 9.
- Node A: Here, after evaluating the rest of the tree, the algorithm will choose the move to actually make, choosing node C, since that will guarantee it the highest possible score assuming the opponent plays perfectly.

There are several optimizations which can be made to this strategy, including Alpha/Beta pruning and the Negamax algorithm. The Negamax algorithm is mostly the same as Minimax, except for the fact that it takes the maximum of the negative values of the scores during the evaluation of an opponent's scores. This optimization is easier because it allows for the use of the same function to evaluate both the algorithm's and its opponent's board positions with the same function, which reduces the amount of code involved while increasing computational efficiency. Alpha/Beta pruning is an adjustment to the Minimax or Negamax algorithms which keeps track of the lowest and highest possible scores, which are denoted as alpha and beta respectively. By keeping track of alpha and beta, we can skip steps and not actually generate and evaluate every node in the game graph if we find them to be irrelevant. Doing so can save considerably on running speed.

- Variations in Fitness Function
    - While all algorithms we try will depend on the same basic Minimax and game tree approach, the team will be varying the fitness functions it uses in assigning scores to each board. Here are some of the strategies we will examine:
        - Random Movement: The tree will only be generated to a depth of 1, and there will be no fitness function implemented. Instead, the algorithm will randomly select one of the possible legal moves. This approach will serve as a baseline to which other strategies will be compared.
        - Simple Point Addition: In this approach, each piece will be assigned a point value: pawns as 1, knights and bishops as 3, rooks as 5, queens as 9, and kings as 1000 (or some other arbitrarily large number). The algorithm will generate positions up to a certain depth, and each final board's score will be determined by subtracting the total value of the opponent's pieces from the total value of the algorithm's pieces. This approach should be better than the random strategy since it prioritizes moves which don't require sacrifice, and which capture enemy pieces. However, there are several nuances to Chess strategy which this approach may fail to capture.
        - Piece-Square Tables: Another approach could be to assign each piece an 8x8 matrix of weights. These weights would correspond to how good of a position said piece is in. For example, the knight is generally better off in the center of the board, so its piece-square table may have higher values in the middle and lower values everywhere else. When computing the score for a board state, the algorithm could add up the entry in each piece-square table for its own pieces, and subtract those of its opponent's pieces. This

approach, potentially in conjunction with point values, could be good for teaching the computer about strong board position.

- Combination: Another option could be to add several different features together. For example, we could add points for pieces as well as using piece-square tables, and maybe even add in factors to address things like the vulnerability of the king, control of the center, or pawn structures explicitly.

References:

https://medium.freecodecamp.org/simple-chess-ai-step-by-step-1d55a9266977
http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html