Rajesh Sakhamuru
Rohan Subramaniam
CS5100

10/27/19

Chess AI Final Project Proposal

The first part of this project will be to create a working 2-player chess board, which allows for all conventional chess moves and after each one, will check whether a winning or drawing move has been made.
Winning/Losing scenario would be:

- **Checkmate**: This is when a player's king is in check (threatened by capture), and there is no way to remove that check. The player whose king is threatened loses.

The drawing scenarios would be:

- **Stalemate**: a player has no legal moves, but is not in check.
- **Insufficient Material**: no checkmate possible with the pieces left (no pawns and one, Knight or Bishop, or less)
- **Threefold repetition**: "If the exact same position occurs three times, with the same player to move, then either player can claim a draw by threefold repetition. The game doesn't end in a draw automatically, so you can carry on if you want to, but if either player claims the draw, then the game ends immediately even if the other player wants to carry on."
- **50-Move Rule**: After 50 moves have passed without a capture or a pawn advancing, you can claim a draw.

The chess code which we will be implementing our game logic into will be from this source: https://repl.it/@f9we/chess . The code here is implemented in Python 2, but with some changes it works in python 3. The chess board as is has no logic for win/lose/draw or even certain moves like en passant and castling implemented. Capturing pieces and alternating turns is functioning.

https://www.chessstrategyonline.com/content/tutorials/how-to-play-chess-draws
https://en.m.wikipedia.org/wiki/Checkmate

Once the working chessboard has been implemented, the chess AI for one side (or both) can be added. The Chess AI project that we are planning to implement will use the following 6 methods/algorithms:

- **Minimax search**
  - Adversarial search using the heuristically scored evaluation of each board state in the search tree, with the AI player as the maximizer and the opponent as the minimizer.
  - The move that is chosen will be based on the assumption that the minimizer will also play their optimal move given the opportunity.
- **alpha/beta pruning + cutting of search**
  - Pruning options that are known to be less optimal.

    - Cutting off search at a certain depth to control runtime with generating smaller search trees.
- **Iterative depth first search**
    - Execute a shallower search first and then use the resulting alpha/beta cutoff values as start values for a deeper search.
    - This will further shrink the search tree and allow for much less optimal moves to be prevented from wasting search resources.
- **Forward pruning/beam search**
    - Will cut off search branches that are “probably” bad
    - Downside is that this method has a chance to cut off future states which may turn out to be beneficial in the future.
    - Will be integrated into the iterative depth first search.
- **Secondary search**
    - Will be used to avoid the horizon effect of the Minimax Search
    - Minimax algorithm in certain scenarios will “waste time” to push unfavorable moves past the search horizon that it can reach and interpret it as having eliminated those unfavorable moves.
    - A secondary search further than the depth of the specific move returned through Minimax is a simple way to check if that is what is happening.
    - A secondary search may only be needed at depths where terminal states begin to become more common.
- **Genetic Algorithm**
    - Can play AIs with different heuristic values generated by the genetic algorithm against each other and can determine the best values based on the best performing heuristic.
    - The top performing heuristic values can “breed” to form the consecutive populations forcing the overall population to evolve to use the best possible heuristic values for that implementation of the chess-board state evaluation function.

In order for the Minimax algorithm to work without searching to the max depth, we will be implementing a heuristic evaluation function to analyze the chess board state for intermediate, non endgame configurations. The heuristic scores of the boards will be used for comparing states and choosing the optimal move. The evaluation will account for multiple variables and weight them appropriately:

- **Total pieces**
    - This is the easiest to account for, but pawns, knights, bishops, rooks, and queens should be valued at different weights
- **Space control**
    - This should weigh how much of the board each player has pieces controlling. Pawns are a good indicator of space control.
    - Center row control will be weighed more heavily than other rows.

- **King safety**
    - How well protected or threatened the king is. The king's potential moves also contribute, such as the option to castle
- **Pawn position**
    - The structure of the pawn frontier. How well pawns that are protected either by other pawns or more powerful pieces will be taken into account.
    - En passant and passed-pawn scenarios will be favored, as well as potentially promoting a pawn.
- **Minor piece advantage**
    - Evaluates the bishops and knights for each side and scores based on their ability to threaten. It has to account for imbalance in these pieces, and which spaces the remaining pieces can access.