ECE 469: AI **Checkers Report** Miraj Patel

Prof. Sable

The checkers game I have implemented is in C++ and uses simple ASCII characters to represent and print the board, with some hex escapes codes to color pieces appropriately to easily distinguish between each player’s pieces. The UI is similar to that of the C++ checkers program Professor Sable posted, at least in the beginning: the binary mode of operation for each player is the first prompt, followed by a prompt for the time limit, in seconds, for the AI if either player is chosen to be the computer; finally, to start the game, the program asks if the user would like to specify a board/game state to start from. The input game state is modelled after the sample boards Professor Sable uploaded – 8 lines of 4 characters representing the playable spaces on the checkerboard, followed by whose turn it is, and the time limit for the AI. However, one (accidental) difference between the representation of my program’s board and Professor Sable’s board is that in my program player 1’s pieces start on the bottom of the board and his pieces work their way up to the top, whereas it is the reverse in Professor Sable’s (hence a 180 degree rotation).

Each turn of the game, the program displays all the legal moves the current player can perform, and the player can only choose one of these moves by entering the number corresponding to the move number. Internally, each move is a structure containing an integer representing the starting piece’s location in the player’s list of pieces, an end Location, an array of Locations representing the pieces that would be jumped if the move was chosen, and an integer representing exactly how many pieces would be jumped. Location refers to a structure containing two integers: row and column in the 2D board array held in the game class – this packed data structure is used throughout the program to determine in which directions the piece can move given a distance and information on the piece (who owns it, and whether it’s a king or not). The two lists of the players’ pieces help reduce the time to find all of the pieces belonging to the current player, as opposed to searching the entire checkerboard. Once a move is chosen, the starting piece’s position is updated, and all the jumped pieces are found in the opponent’s list and removed.

The AI’s heuristic function breaks up evaluation into three regions, of decreasing importance in terms of magnitude: value of the pieces, value of the positions, and value of the advantage. The value of the pieces is a sum of the each regular piece’s worth (+2 for player 1, -2 for player 2), and each king’s worth (+7, -7) since they have much more mobility than pawns. The position value applies to only the pawns, and adds 1 point for each row closer it is to becoming a king, else 9 points for remaining in the starting position to prevent the other player’s advancement to produce kings. The advantage value is specifically designed to resolve the 2 kings vs 1 king dilemma; if the 2 kings are too far away from the opponent’s king, generally the alpha beta search will not be able to see far enough in a short amount of time to realize it can guarantee a win by moving towards the king and weaseling it out of its corner. Thus, after determining of the player’s piece is in a double corner, increasing weights are added to tempt the two kings to approach the double corner – when they are within range, the heuristic’s heavy weighting for winning will kick in and the moves necessary to force it out and capture it shortly thereafter will be taken. Wins are also individually weighed based on proximity to current game state: the higher in the search tree greatly increases its evaluation.

Tweaks and optimizations made to try to increase the depth that the alpha-beta could search to include copying only the necessary components when cloning the game for each child (each possible move arising from a game state), using the given of a reasonable time limit to provide a lower bound of depth 5 for the iterative deepening search, and increasing the lower bound if the time limit is greater than the minimum (3 seconds) expected and when the total number of pieces on the board decreases, indicating a smaller branching factor, and thus less nodes. The iterative deepening search ceases to go deeper if after a completed depth less than half the remaining time remains, else it will try one more level. Inside alpha-beta, given that the search was called with more than half the time remaining, the threshold is increased to 75%: when the time elapsed exceeds 75% of the total time, the alpha-beta breaks out and the incomplete result is ignored. Furthermore, all the data structures, except for the list of pieces, were statically allocated, and are thus removed once they are out of scope – it was assumed that not relying on allocating dynamic memory and having to freeing this memory would decrease the overhead time of the AI.