ECE 469 Project #1 Report – Colin Hwang

**How to run and compile my code:**

Compile and run my code the same way you would do for my DSA1 and DSA2 assignments.

What I do (in Cygwin):

**To compile:**

g++ playCheckers.cpp

**To run:**

./a.exe

**Project Writeup:**

Before starting the game:

On startup, the program will prompt the user for various options:

1. Does the user want to start a new game or load a saved game?
2. Does the user want the game mode to be AI vs AI or User vs AI?
3. What time limit does the user want to set for the AI (3s – 60s)?
4. Which player is starting the game?

For option 1, the time limit and starting player should be given in the saved game file. Therefore, the program will not prompt the user for these things. The loaded board will follow the same format as the provided saved game files. 1 represents player 1’s counter, 2 represents player 2’s counter, 3 represents player 1’s king, 4 represents player 2’s king, 0 is an empty space. The starting player and time limit are under the board in the text file.

For option 4, it is important to note that for the User vs AI game mode, player 1 is the user and player 2 is the AI. Player 1 will use red counters and player 2 will use white counters.

Checkerboard Display:

After deciding between the various pre-game options, a checkerboard will be displayed. The checkerboard consists of black and white tiles on an 8x8 grid, and pieces are placed on the black tiles. Each row and column will be labeled with green numerical values from 0 to 7.

Player 1 will use red counters represented the letter ‘a’, and player 2 will use white counters represented by the letter ‘z’. Player 1 kings will be displayed as ‘A’ and player 2 kings will be displayed as ‘Z’. All of this information will be displayed on the “counter key” at the top of the board.

A list of legal moves will be displayed for both the AI and the user, and each of the legal moves will have a numerical value associated with it. To select a move, the user will input the associated numerical value. Displayed legal moves will show the starting position and end position, with the format: [row, column] –> [new row, new column].

After the AI is done selecting its move, the program will display the move the AI chose, the maximum depth searched to completion, the depth the AI stopped searching in, and the amount of time taken to select the move. The AI will never go over the time limit. Jumps will be displayed appropriately, displaying intermediate steps.

Minimax search

The AI searches for ideal moves with an implementation of minimax search with alpha beta pruning and iterative deepening.

When it is the AI’s turn to move, iterative deepening is called. Iterative deepening will continuously run the minimax search to successive depths (starting at 0, then 1, then 2, etc). Iterative deepening will stop once the time limit has been reached. If a depth is searched and there is less than half of the time limit remaining, the AI won’t bother to progress to the next depth. If only one possible move is available, there is no need to do a minimax search, and the move will be chosen.

The minimax search starts by making sure that the time limit has not been passed. If minimax is at the maximum depth that can be searched or if the current player cannot move, the output of the heuristic function will be returned. The minimax search may continue to search to a further depth until there are no more jumps to make. This is because trading in checkers may result in a better move for the AI. Therefore, it is possible that the heuristic function will run after a piece is given up.

For every possible move, minimax will make a copy of the board, make the move and run the minimax search on the copy.

Heuristic Function

The heuristic function evaluates the game in two different stages: “early” and “late” game.

The heuristic function for “early game” considers:

1. Number of pawns each player has
2. Number of kings each player has (kings > pawns)
3. Pieces in each player’s home rows (first and last rows)
4. Pieces in the middle box of the board
5. Pieces in the middle rows, but not the middle box

The heuristic function for “late game” considers:

1. Number of pieces on board
2. Number of pawns each player has
3. Number of kings each player has (the king’s value is increased)
4. Trading pieces (player that has more pieces should be trying to force trades)
5. The number of pieces that have yet to be kinged (this will cause the opponent’s home rows to be more valuable)