Elli Rappaport
Othello – AI
November 1, 2015

For this project, I implemented Othello in C++ using ASCII color coding for the display. The AI game-space search was performed using an iterative deepening algorithm and alpha-beta pruning.

At the start of the program, the user is allowed to choose whether Player 1 (the Black player) or Player 2 (the White player) will be human players or computer players; the user can choose for both to be humans, both to be computer or any combination. The user can then choose whether to load a specific starting board configuration from file or to play with the standard initial setup. Finally, if the there is at least one computer player and we did not load the board from file, the user can choose an integer number of seconds to limit the computer's turn. The higher the number, the longer the computer can search the board-space for.

The board is displayed in the terminal as an 8x8 board. A blank space shows up as green, a space occupied by White shows as white, and a space occupied by Black shows as blue (the black color displayed weirdly). Additionally, on each green (unoccupied) space, there may be an indicator indicating if a legal move is available. If only White has a legal move, the space will have a “ww” written inside; if only Black has a legal move, the space will have “bb” printed inside; if both White and Black could legally take the space, an “aa” will be printed. The board will display all legal moved for both players no matter whose turn it is in order to give a better feel and for the available moves to each player.

As mentioned above, this project was implemented in C++. The project has a Board class where the board's occupants and the legal moves are stored. The Board object also holds whose turn it is and a few other game state pieces. There is then a Player object, which contains a PlayerMover object. The PlayerMover object can be either a PersonPlayerMover or a ComputerPlayerMover, where each one holds the necessary code for the move selection – either parsing the user entry or performing the search.

The bulk of the project's work was to implement the board-space search along with the pruning and the heuristic. The search is an iterative deepening search, primarily limited by the user selected time limit. Over a 5 second duration, the application can usually search around 9 plies deep. If it cannot finish the full search for the depth limit on that duration, the AI will use the previous best move based on that last iteration with a complete search. At the end of each iteration, the AI uses a heuristic based off of several factors. These factors include, the ratio of coins the player holds, the ratio of corners the player holds, a penalty for being near unoccupied corners (risking giving it away to the opponent), and the number of moves available.

With more time, this program can be improved in a few ways. Firstly, I would try to clean up a lot of the memory allocation during the search. The memory allocation and some of the wasteful work done during the search may be limiting the depth that the AI can actually get to over the set time limit. If the AI is allowed to search further, better decisions can be made and we would dampen the horizon affect. Also, with more time and work, I would have liked to compare several different weightings of the heuristics in a tournament style between different players. I would have liked to set each player with a slightly different heuristic function and played multiple times to gather better statistics on the different heursitics. The heuristic is very important to the AI game play and better adjustments of that would result in a better game play.