An AI was created to play Othello. This was done using a min max search with Alpha Beta pruning. The game allows the use to choose between human vs. human, human vs. computer, or computer vs. computer play. Additionally, a file representing the state of the game can be uploaded to start from a random playing spot. The syntax for that file is 8 lines of 0, 1, or 2 separated by spaces to represent no players, the first player, or the second player in each position respectively. The next line is the number for which player it is (1 or 2) and the last line is the number of seconds the computer has to calculate it's move. Here's a sample file:

0 2 2 1 2 2 2 2

1 1 1 1 2 1 0 2

1 1 1 2 1 1 1 2

1 1 1 1 1 1 2 0

0 1 1 1 1 2 2 2

1 1 1 1 1 2 2 2

1 2 2 2 2 2 2 2

1 1 1 1 1 0 2 1

1

10

The AI searches the game space up to a depth of 1, then 2, then 3, etc. While it does this at the top of each recursive call to the AlphaBeta function, it checks the time and if it only has less than .01 seconds left it changes a global Boolean indicating that the level was not completed and returns. The program then uses the deepest level that it was able to completely search.

At the final level of each search, if the game is not yet over a heuristic is used to determine how good the position of the "max" player is. This heuristic is zero-sum meaning that for heuristic(player1) == heuristic(player2). A number of different types of heuristics were tested to make a powerful heuristic function. The methods include: counting coins; weighing the corners positively; weighing the sides positively; weighing the sides right next to the corner and the subcorner negatively; weighing the next to corner pieces negatively only when the corner is not yet taken; determining the number of stable (and semi-stable) pieces positively; using the difference between the number of moves max vs. min player has; and using a static weighing for each position on the board. In the end a weighed combination of some of these were used. When it was determined that the game was over, a number greater than the value for the heuristics was returned plus the coins count (so that it would be motivated to try to win with more points).

There was a Board class which was used to represent the state of a game including the board, who's turn, if there's been a skipped turn, etc. The board was passed down the alpha Beta search and for each move a new board was created "it's child" and then the test move was done in it. To save time on memory allocation, it was attempted to pre-allocate a list of boards and then just move the boardPointer for each new board. This only ended up saving about 10% of the time, which due to the exponential factor of searching the game space isn't very significant. Because it delayed the game start a few seconds (which was annoying for the user), this wasn't used. Additionally, it was considered to use multi-processing to try to add more computational power. But without too many cores, this too isn't worth the complexity.