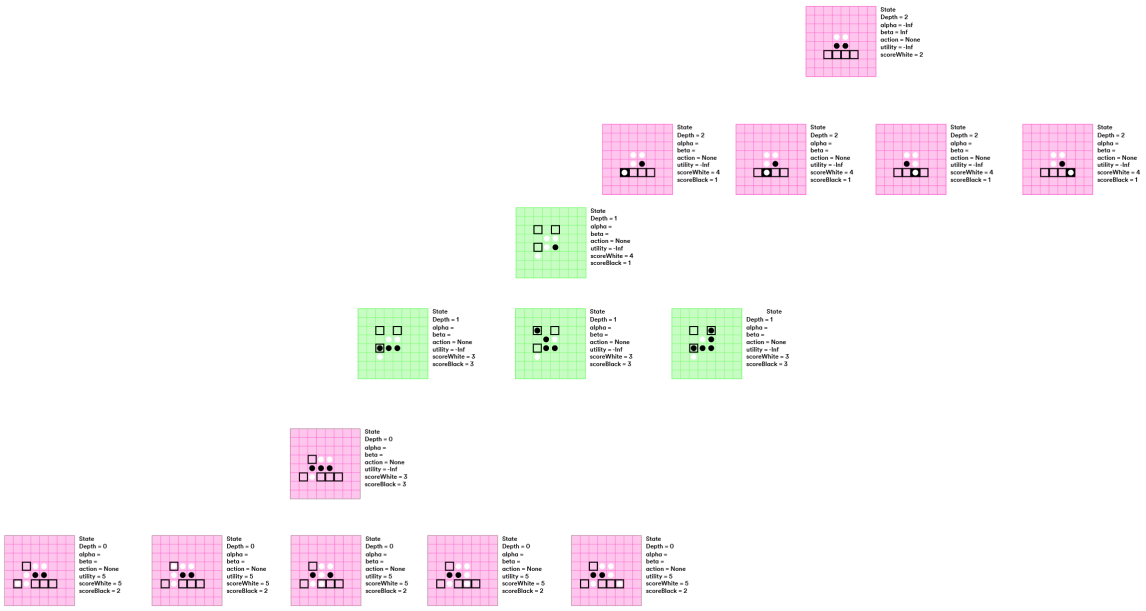
Reversi Lab
CS470

Daniel Reich
Gavin Jensen

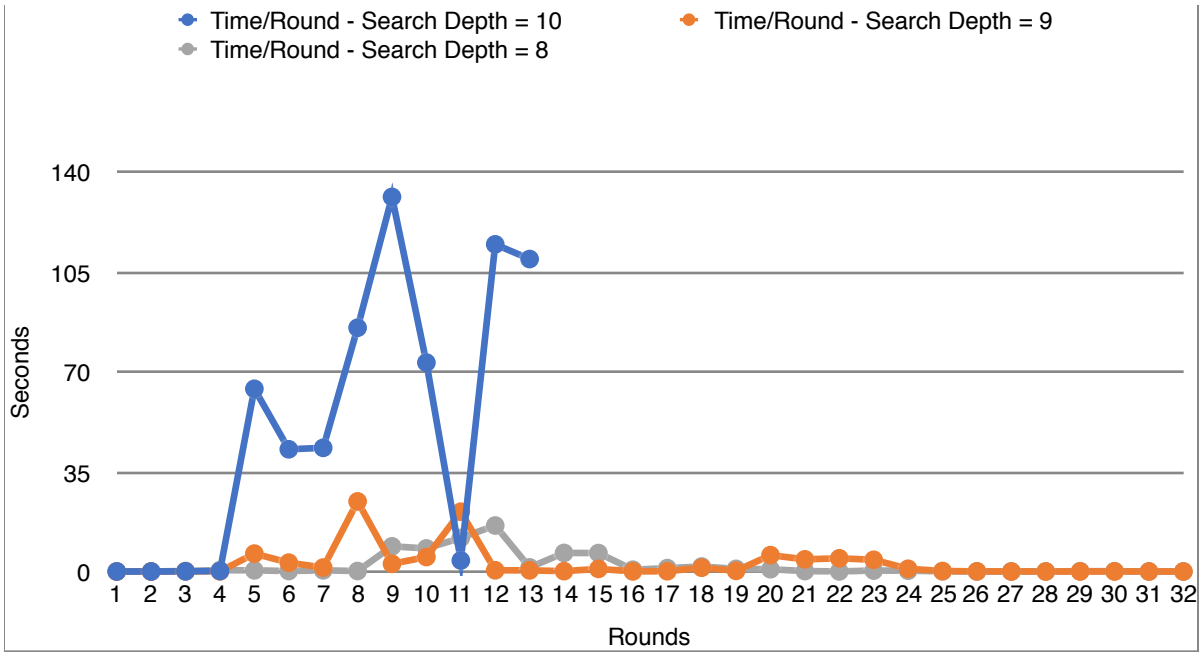
Time spent
Both partners spent approximately 9.75 hour each on this project.

Implementation:
As seen in the attached code, we successfully implemented the minimax and alpha-beta pruning algorithm found in the book. We created a diagram of game states for a tree of 3 levels to gut check that the algorithm was working as intended.



Time
When we played against the algorithm, we tried different levels of depth with the alpha-beta pruning and no heuristics except for score maximization. We wanted to see how long it would take to search the tree. The data gathered is not a consistent representation of performance since the games were different every time. The time taken between a search depth of 8 and 9 were comparable. A search depth of 10 did not finish with the longest round taking over 2 minutes. The human player had an aggressive strategy of locking in pieces and minimizing number of moves of opponent which explains why some rounds were very quick for the algorithm given that the count of valid moves was low. Graph of time shown below:

Heuristics:



When we first implemented the alpha/beta pruning, we used the score of the game as a simple heuristic. Our algorithm would simply count its pieces on each round of the game. This worked for the purposes of getting the algorithm to work, but is a bad game playing strategy since many pieces are flippable; a high score at the beginning of the game doesn't necessarily mean the player will win.

We decided each of us would come up with a more interesting heuristic, and then we would play them against each other to evaluate their strengths and weaknesses:

1. Our first heuristic used the game score as a measure, but also checked the board for any available corners or "safe edges" – pieces on an edge that couldn't be flipped. This algorithm gave the heaviest weight to corners and safe edges, it chooses them no matter what when available.
2. Our second heuristic was similar to the first; it used the score as a base, then added additional points based on how flippable each piece was. For example, it would find all of the player's pieces, then for each one check if it could be flipped by playing on the vertical, horizontal, or diagonal row that piece was in. If a piece could be flipped from 2 directions, it was given the score of 1, if it could be flipped from only 1 direction, it was given a score of 3, and if it could no longer be flipped, it was given a score of 6.

Both these algorithms chose random pieces for the first 4 moves of the game. We played the algorithms against each other 10 times, switching player 1 and player 2 after 5 rounds. Here are the results:

	Wins	Losses
Algorithm 1	4	6
Algorithm 2	6	4

Based on the results, we concluded that neither heuristic was markedly better than the other. This is most likely due to the face that both are similar, they both give heavy weight to pieces that can't be flipped, but also use the score of the game as a factor in decision making.

When playing against these algorithms ourselves, we noticed they were much better than when the heuristic was simply the score of the game. We noticed that because of the maximin nature of the algorithm, it would often make moves to prevent us from getting the corner (thereby minimizing the loss to itself). Our algorithms were still

beatable, but took considerably more effort. One of the biggest weaknesses seems to be that while the new heuristic made the algorithm guard the corners a bit more, it seemed like it often played in tiles that made it easy to take the corners after a few rounds. This would be an interesting thing to add to the heuristic on future iterations.

Other Experiments:

We implemented a second algorithm in an effort to see how it fared against the minimax search with alpha/beta pruning. This algorithm explored the game tree by assuming that its opponent was evaluating turns based on a minimax search with alpha/beta pruning. For every possible move, it would run the minimax search for its opponent, then see its available moves based on what it guessed was its opponent's move.

This algorithm allows us to assume the heuristic our opponent is using, while using a different heuristic for ourselves. This is in contrast to regular minimax search, which assumes your opponent is evaluating each game state the same way you are. The hypothesis in implementing this algorithm was that if we had a better heuristic than our opponent, and if we could approximate the heuristic they were using, the algorithm could "guess" its opponents moves and stay a step ahead in the game.

This algorithm also has some large weaknesses. It is much slower, since it is running minimax search for multiple possible moves instead of just once. Because of this, in order to be feasible, the minimax search is limited in how many levels it can search. This is a huge disadvantage, since our opponent could be evaluating 5 turns deep while we can only search 3 levels for each possible turn.

To explore this algorithm, we ran it against a minimax algorithm that used the game score as its heuristic. Our algorithm would find its possible moves, guess what move its opponent would take by running the minimax algorithm, then use its own heuristic that factored in corners and edges to choose its best move. The minimax algorithm searched the game at a depth of 5 turns, while the guessing algorithm assumed its opponent would use minimax for 3 future turns.

We ran these algorithms against each other 10 times, switching player 1 and player 2 after 5 rounds:

	Wins	Losses
Minimax algorithm	5	5
Guessing algorithm	5	5

Based on the results, the guessing algorithm did not show any advantage over alpha/beta pruning. This algorithm wasn't very good and predicting its opponent's move, mostly due to the difference in searching depths; our algorithm guessed its opponent was searching at a depth of 3, when the opponent was searching at a depth of 5.

This algorithm was only made to see if it could beat the maximin algorithm, it is not a good general algorithm and is easily beaten by a human player. Based on its poor performance, we concluded that the guessing algorithm is not very competitive.