CMPT 310

Final Project

Reversi with Monte-Carlo Tree Search

John (Junseong) Kim

301262540

[Jka157@sfu.ca](mailto:Jka157@sfu.ca)

# 1. Introduction

The purpose of the project was writing a program that can play Reversi on an 8x8 board with pure Monte-Carlo Tree search and random playouts. As multiple simulations were needed to pick a preferred move, a language more efficient than Python was used. My program was written in C++ for this reason was able to simulate faster than Python was able to with the tic-tac-toe example. Interacting with my program is very simple. The user is given an option to pick 1 or 2 as can be seen in Figure 1. Entering the value 1 will start a game with the user and the computer implemented with a program that selects moves at random from its possible selection of moves. On the other hand, entering the value 2 will start the program with the Monte-Carlo Tree search with modified heuristics playing against the random playouts from the first option.

A screenshot of a cell phone

Description automatically generated

Figure 1: Initial UI for the Program

# 2. Player vs Random Playouts

By selecting 1, the program lets the player play the game against the computer that selects the moves randomly from its possible selection of moves. The player will play the black stone and the computer will play the white stone and this UI can be seen in Figure 2. The board state is from 0 – 63. If you select the wrong move, the program will show the possible moves highlighted with the circles as can be seen in Figure 3. If the input is valid, the computer will make a random move from the moves possible. This will continue until there is a deadlock or the board is filled out. At this point the game will output the result of who won with how many of their stones are on the board. This method of random selection of move is not particularly effective against a smart player. A player with the knowledge of special tricks will play much better than a computer plays its move randomly. This computer’s performance will be compared to the modified computer algorithm with additional heuristics and random playouts.

A close up of a logo

Description automatically generated

Figure 2: UI for Option 1 (Player vs Random Selection)

A close up of a mans face

Description automatically generated

Figure 3: Option 1 Incorrect Input

# 3. Modified Monte-Carlo Tree vs Random Playouts

Selecting option 2, the modified Monte-Carlo Tree search will play as the black stone and the computer with random selection of moves will play as the white stone. From here on forth I will refer to the program that plays as the black stone as the tree search method and the program that plays as the white stone the random method. The tree search method will initially search which plays it can make. From the initial state, the possible moves are the same as that from option 1 and can be seen in Figure 3: board positions 20, 29, 34 and 43. The tree search method will select the possible board positions in order and will make random playouts. This simulation is done 10 times (total of 40 simulations before making the move), and the wins/losses/draws are recorded. The move that resulted in the greatest number of wins will be chosen and the turn will go over to the computer with the random method. The weights for the wins were given 10, losses given -10 and the draws were given 3. This makes sure that the simulation doesn’t choose the move that leads to the most amount of losses. The game is played out until the board is filled or neither of the players can make a move. On top of the multiple simulations, the tree search method was also implemented with the heuristic that favors corners. In Reversi, it is a common strategy to place your stone on the corners if possible. This should, in theory, help with the win rate of the program. The UI of this playout can be seen in Figure 4 and Figure 5.

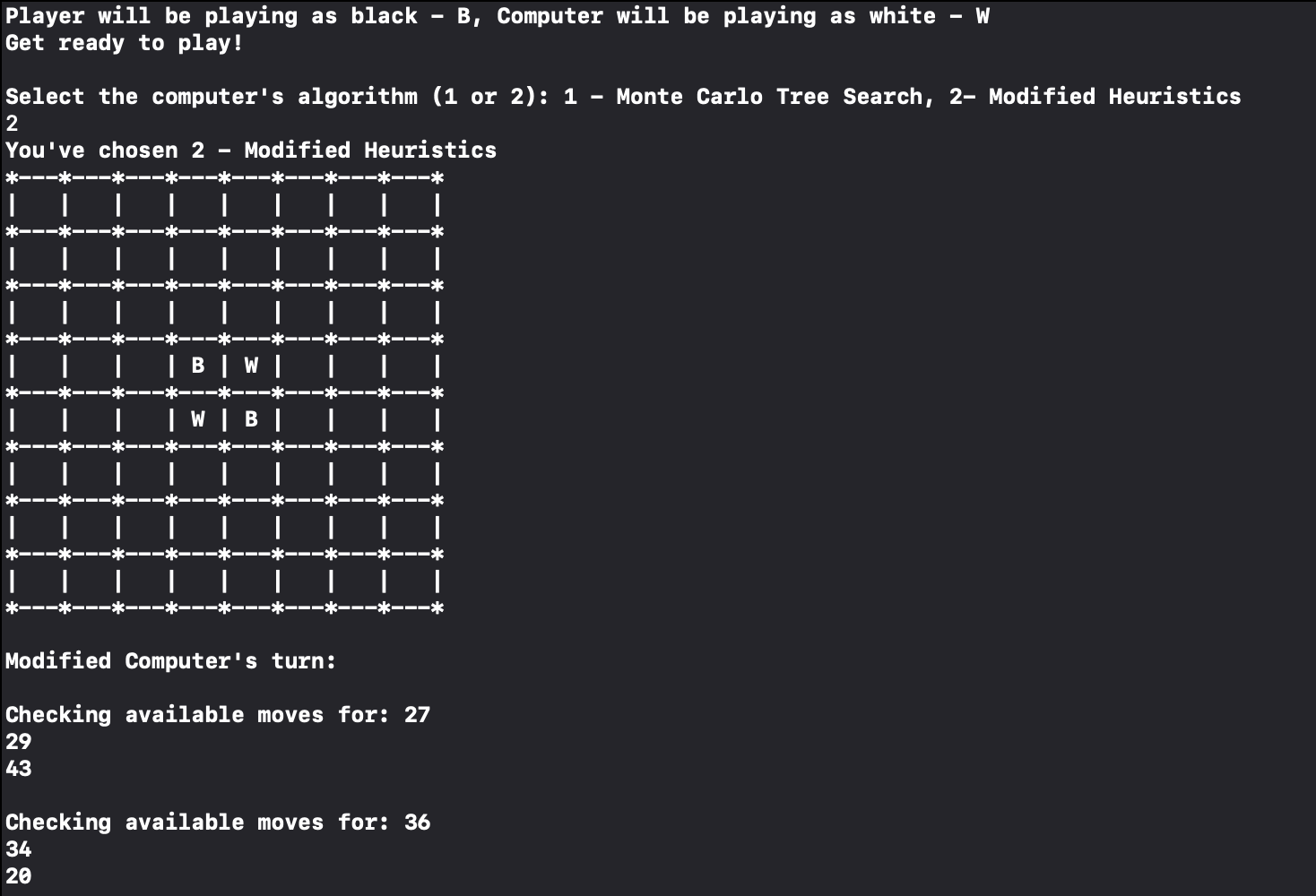


Figure 4: Option 2 UI

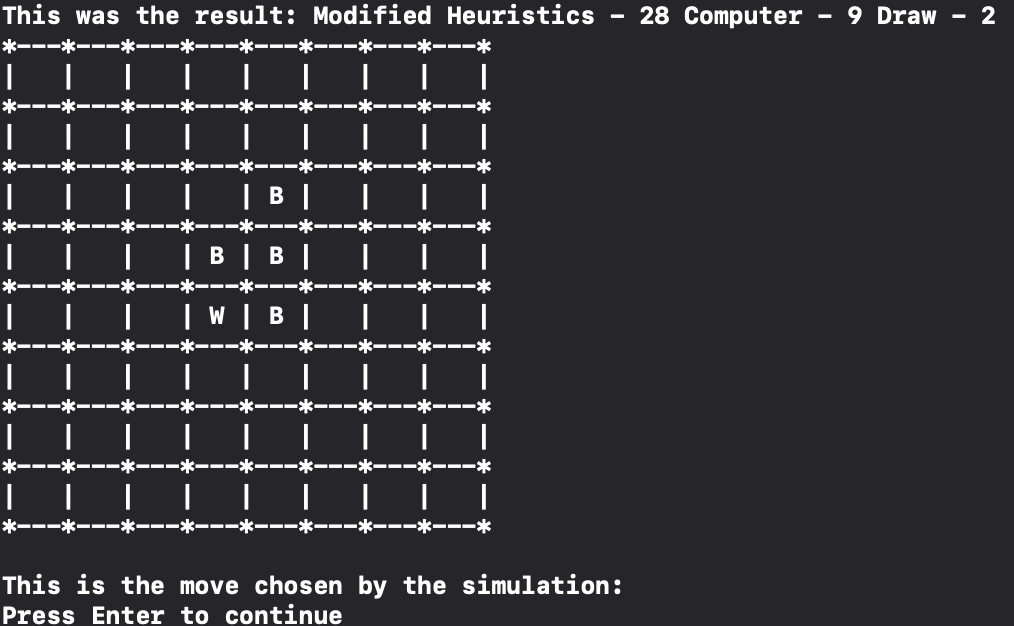


Figure 5: Move Made after Simulation

# 4. Results

Comparing the performance of a “smart” player, random playouts and the modified Monte-Carlo Tree search method, it is hard to say either of the programmed algorithm can confidently beat a knowledgeable “smart” player in Reversi. Reversi is a much more complicated game to play with many different possible moves that can be made compared to the game tic-tac-toe. For example, tic-tac-toe game’s possible moves decrease from 9 (assuming you play first) down to 7, 5 and so on. On the other hand, it is hard to even gauge how many different moves are available after certain number of plays has been made Reversi. Due to this fact, it is very difficult to simulate every possible playout before making a move. In addition, the reason why the weights were given to the number of wins/losses/draws compared to choosing the move that flips the greatest number of stones is because the number of stones that a certain move flips does not matter as much until the very end. When a corner move is made, a very large number of stones can flip at once, essentially flipping the whole game around. With all this complications and applications in mind, the result I had playing the tree search method against the random method can be seen in Figure 6. As the results show, the wins are not consistent with the tree search method against the random method even though it has been programmed with more knowledge. This is mainly due to the aforementioned issue of not being able to simulate all possible moves before making a decision and the general complexity of the game. It should also be noted that the ratio of the stones flipped is random and does not show a noticeable pattern.

Figure 6: Table of the Result for Tree Search Method and Random Playouts Algorithms