James Braun
CMPT 310
Spring 2020
Assignment 5

Reversi with Monte Carlo Tree Search

For this assignment, I created a program in C++ for playing games of Reversi. In my Reversi program, a human player can play against an AI opponent, or two AI opponents can play against each other. The AI players can make moves in different ways.

First, the AI can select its next move totally at random. It generates a list of all the valid moves for the given game state and then picks one randomly.

Second, the AI can select moves using pure Monte Carlo tree search. This means for every valid move, it does up to 250 random playouts and records the result (win, loss, or draw) after each one. There is a time limit of 5 seconds for making each move. The AI meets this time limit by dividing 5 seconds by the number of valid moves, and then it spends at most that amount of time performing random playouts for each valid move. For example, if there are 10 valid moves, it spends at most 0.5 seconds on each one before moving on to the next. It will also go on to the next move after 250 playouts if less than 0.5 seconds have elapsed. The time limit usually only comes into play during the beginning of the game as that is when playouts take longer. After each playout, the playout score is determined. Wins are worth 1 point, draws are worth 0.5 points, and losses are worth 0 points. After the playouts have been performed for all valid moves, the AI picks the move that results in the greatest score. Note that if fewer than 250 playouts were done for any move, then that move's score is adjusted to as if 250 playouts were done. For example, if only 100 random playouts were done for a move and the total score was 50, then the score would be adjusted to 50 * (250 / 100) = 125. This prevents the AI from simply picking the move that had the most playouts done on it. Lastly, if multiple moves result in the same score, one is chosen randomly.

Third, the AI can select moves using a weighted combination of various popular heuristics of Reversi. The first heuristic is based on piece position. In Reversi, it is better to have your pieces on certain squares compared to others. For example, it is really good to have a piece in a corner since it can't be flipped by your opponent, but really bad to have a piece diagonally adjacent to a corner. This AI gives preference to moves that would result in the greatest difference between the summed positional worth of all its pieces and the summed positional worth of all the opponent's pieces. The second heuristic is based on the number of moves the opponent can make after you place a piece down. The fewer options your opponent has, the greater the likelihood they will have to make a bad move. This AI gives preference to moves that result in the opponent having fewer valid moves to choose from on their next turn. The third heuristic is a bit counter-intuitive. You actually want to make moves that flip over the least number of your opponent's pieces – at least in the beginning of the game. The reason why this works relates to the second heuristic. If you have fewer pieces on the board, then your opponent has fewer moves that they can make, which again can force them into making a bad move since they always have to move if they are able. In the combined-heuristic move selection process, this heuristic is used until there are 10 or fewer empty spaces left on the board. After that, this AI switches over and uses the fourth heuristic, which is simply selecting the move that flips the most pieces. It does this for two reasons. First, the overall goal of Reversi is still to control the most pieces at the end of the game. Second, I found from testing that this switch at this point made this AI perform slightly better.

Overall, I weighted the heuristics in the following way:

More than 10 empty spaces left:

$$\text{total heuristic score} = \text{positional score} + \frac{30}{\text{opponents moves} + 1} + \frac{80}{\text{flipped pieces} + 1}$$

10 or fewer empty spaces left:

$$\text{total heuristic score} = \frac{\text{positional score}}{10} + \frac{60}{\text{opponents moves} + 1} + 400 * \text{flipped pieces}$$

Opponents moves and flipped pieces are in the denominator when I want fewer moves or pieces to result in a higher score, and I added the +1's to avoid any issues with dividing by zero. I adjusted all of the weights of the heuristics by hand by simulating hundreds of games and seeing which weights resulted in the highest win percentage.

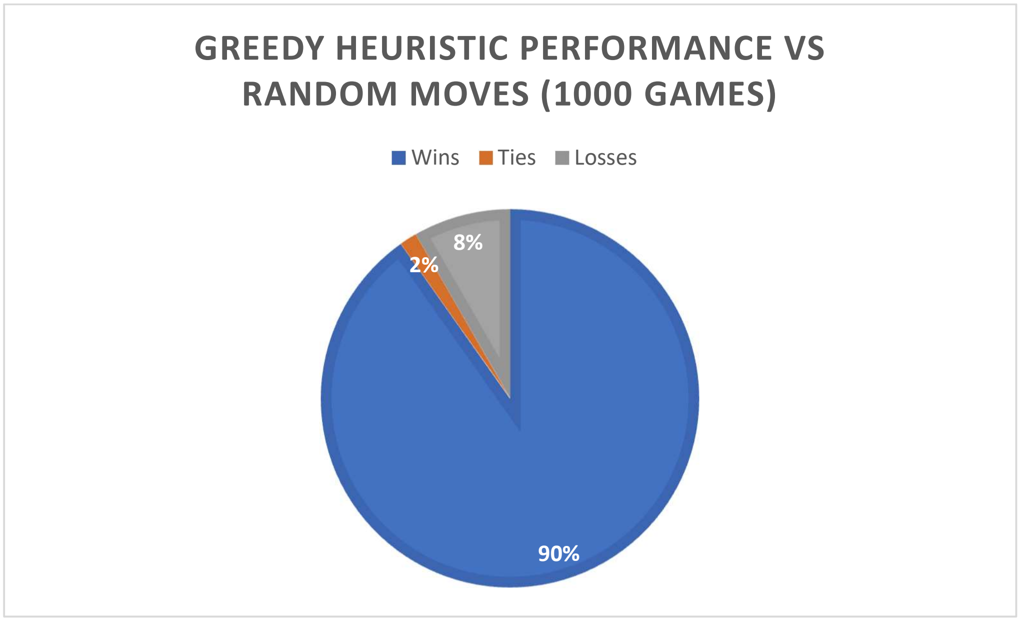
These weights resulted in an AI that won roughly 90% of the time against another AI opponent that made totally random moves (the first AI mentioned above). There was no real difference in winning percentage depending on who went first or second.

Here is a summary table of the results of 1,000 games:

Move Order	**Wins**	**Ties**	**Losses**
Second	457	8	35
First	445	7	48
Total	*902*	*15*	*83*

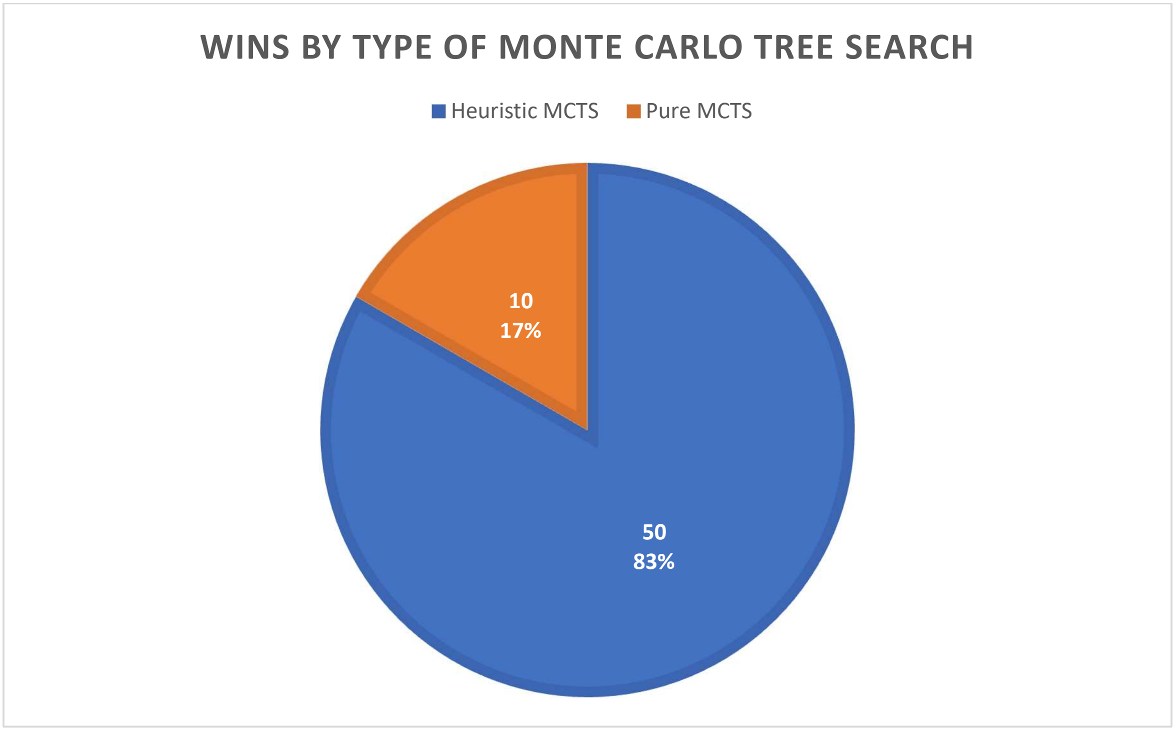
Where turn order represents if the heuristic selection AI went first or second.

Here are the results in pie-chart form:



I also modified the pure Monte Carlo tree search AI to create an AI that utilizes this heuristic move selection process during the playouts. To really see if this was better than random playouts, I had my pure MCTS program and my heuristic MCTS play against each other 60 times. They each went first 30 times to avoid any bias of move order.

Here is a pie chart of the results:



For the final piece counts of every game, please see the last page of this report. In games 1-30, the pure MCTS program went first, and in games 31-60, the heuristic MCTS program went first.

My heuristic MCTS won 83% of the time, lost 17%, and there were no ties. There was no difference in the results based on who went first. The heuristic MCTS won 25 times went it went first, and won 25 times went it went second. I feel like the sample size of 60 and wide margin of wins are enough to comfortably conclude that my heuristic version is better.

Just for fun, I decided to see what would happen if the two MCTS programs played against an opponent that just moved randomly, and both won every single time for dozens of games (again alternating going first and second). This is not to say that my MCTS programs play perfectly, but they are certainly a lot better than randomly moving. In addition, they both beat me whenever I played against them, although admittedly I'm not very good at Reversi.

Overall, I feel like I created a program that combines Monte Carlo tree search and sensible move selection heuristics to play Reversi quite well.

Game #	Heuristic Pieces	Pure Pieces	Winner
1	40	24	Heuristic
2	47	17	Heuristic
3	40	24	Heuristic
4	47	17	Heuristic
5	44	20	Heuristic
6	39	25	Heuristic
7	44	20	Heuristic
8	37	27	Heuristic
9	19	45	Pure
10	40	24	Heuristic
11	44	20	Heuristic
12	40	24	Heuristic
13	40	24	Heuristic
14	40	24	Heuristic
15	36	28	Heuristic
16	42	22	Heuristic
17	34	30	Heuristic
18	22	42	Pure
19	38	26	Heuristic
20	43	21	Heuristic
21	41	23	Heuristic
22	30	34	Pure
23	21	43	Pure
24	40	24	Heuristic
25	23	41	Pure
26	48	16	Heuristic
27	45	19	Heuristic
28	36	28	Heuristic
29	34	30	Heuristic
30	44	20	Heuristic
31	50	13	Heuristic
32	19	45	Pure
33	40	24	Heuristic
34	45	19	Heuristic
35	41	23	Heuristic
36	42	22	Heuristic
37	18	46	Pure
38	41	23	Heuristic
39	46	18	Heuristic
40	45	19	Heuristic
41	49	15	Heuristic
42	42	22	Heuristic
43	39	25	Heuristic
44	43	21	Heuristic
45	35	29	Heuristic
46	36	28	Heuristic
47	28	36	Pure
48	49	15	Heuristic
49	41	23	Heuristic
50	34	30	Heuristic
51	44	20	Heuristic
52	49	15	Heuristic
53	47	17	Heuristic
54	25	39	Pure
55	38	26	Heuristic
56	41	23	Heuristic
57	40	24	Heuristic
58	36	28	Heuristic
59	12	52	Pure
60	42	22	Heuristic