CMPT 310 - Assignment 5

Reversi with Monte Carlo Tree Search

Joshua Peng: 301340929

Praneer Shrestha: 301361473

Prof. Toby Donaldson

Simon Fraser University

Submission Date: August 11th, 2020

We implemented the Reversi with Monte Carlo Tree Search in C++. We used several collections from the C++ Standard Library that includes:

- <vector>
- <iostream>
- <queue>
- <cmath>
- <time.h>

The implementation included two main classes, Reversi and MCTS (Monte Carlo Tree Search), and a struct for an object holding x and y coordinates called moveCoords.

The Reversi class encapsulated all functions related to the game including returning the state of the game, making moves and placing pieces. Whereas the MCTS class had no information about the implementation of the game and only had a core function to perform random playouts and the heuristics of the game.

Helpful information about the program's implementation:

- Black pieces in the board are represented by the number 1
- White pieces in the board are represented by the number 2
- Empty pieces in the board are represented by the number 0
- The threshold for number of playouts is not the total number of playouts to perform one move, but the total number of playouts for one potential move. E.g. At the start of the game, there are 4 potential moves, and if the threshold was set to 100,000, then there would be in total 400,000 simulations of the game the AI performs.

Pure Monte Carlo Tree Search

The first implementation version of our Reversi AI program was done using pure Monte Carlo Tree Search, an algorithm in which the AI must simulate a series of playouts starting from the current state of the game, strictly based on random potential movements.

Generally, in the Monte Carlo Tree Search, like our Tic-Tac-Toe game in previous assignments, the AI usually plays out entire durations of the game. The AI simulates every permutation of moves, the AI and it's opponent can select, and these playouts record the number of wins, for each move from the current state. That way, the AI is able to select moves that lead to the most number of wins. However, this method can only be done on significantly small games, such as Tic-Tac-Toe, that have smaller boards which decrease the maximum amount of potential moves at each iteration of the simulation. Reversi, on the other hand, has a much larger board and due to the functionality of the game itself, the number of potential moves the AI and the simulated player can choose for each of the branching simulations, grows exponentially. This is why, using depth-first search, and simulating entire games is virtually impossible with our current home laptops and computers with respect to both time and space.

We instead decided to use breadth-first search through a queue to allow the AI to explore every potential move and constantly switch between different simulated game states. Additionally, we set a maximum threshold on the depth of the simulated games, then recording if the AI is currently winning at a simulated game state by a simple comparison to the number of favorable pieces to the AI against the opponent. This information is relayed back to inform the AI to pick the initial move that led to more states in which the AI was winning.

In terms of performance, the pure MCTS AI performed very well against average players like us. Considering that pure MCTS AI is trying to maximize its own score at every move, the pure MCTS AI consistently chose moves that flipped opponent tiles. However, this method only works very well when the AI is given a large threshold on the number playouts. Furthermore, although the winner of the game is ultimately decided by who has more favorable pieces on the board, there are more aspects to the strategy such as limiting the opponent's moves and trying to gain stable pieces. A set back to the pure Monte Carlo Tree Search implementation is that it takes a significant amount of time to compute the best move given a high threshold. A lower threshold will make the AI weaker in the game as it is not able to stimulate a deeper depth of the game states.

Heuristics

We used three heuristics in implementing our heuristic-based Monte Carlo Tree Search algorithm.

1. **Coin Parity Heuristic:** Captures the difference in coins between the two players.
2. **Mobility Heuristic:** Captures the relative difference of the number of potential moves between two players.
3. **Corner Capture Heuristic:** A larger reward or more incentive given to corner pieces.

We scaled each heuristic to return a score from -100 to 100 (where -100 is least desirable and 100 is most desirable).

The coin parity heuristic was trivial and determined the ratio of the coins the AI had in comparison to its opponent. This was different from the simpler reward function in the pure Monte Carlo Tree Search where a stagnant score of 50 would be given or deducted based on the AI having more coins than the opponent. The coin parity heuristic calculated not just if the AI had more favorable pieces on the board at that time, but by how many.

The mobility heuristic intends in restricting the opponent's mobility while increasing one's own mobility. The score was calculated by finding the relative difference of the AI's potential moves with that of the opponent. If the opponent had more moves than the AI, then this score would be negative.

In Reversi, the four corner pieces are highly desirable as once captured, these pieces can never be taken by the opponent. Hence, corner pieces are stable. The heuristic returned a score of the difference in the number of corner pieces obtained by the AI compared to its opponent. If a playout has more corners taken by the opponent, then the score will be negative, whereas, if the AI has more corners, the score will be more positive. Interestingly, when playing against the AI with these three heuristics, the AI did not attempt to capture corners immediately even if it had a possible move to do so. Although not capturing a corner immediately, we saw that it made moves that did not allow us (the opponent) to capture corners.

Performance

Pure Monte Carlo Tree Search vs Humans (us):

The Pure Monte Carlo Tree Search AI was surprisingly very good at playing the game. When the threshold was set to 10,000 and above, we consistently lost against the AI. Considering that we are simply average players of Reversi, our only strategy when we played against the AI was simply to choose the move that would flip the most tiles but more importantly, we tried to capture the corner spaces first. Since the Pure MCTS AI determined moves strictly on completely random playouts, we were able to capture the corners fairly easily. But although we captured the corners of the board, the AI still managed to find a way to win in the end. However, when we tested threshold values lower than 10,000, such as 1000 or 100, the AI was fairly easy to beat, especially when we captured the corner spaces. We believe this is because 1000 and 100 play outs are way too little play outs for the AI to determine a potential move. We believe the AI has not explored the necessary options to find the most optimal move, which is why the AI makes mistakes and ultimately loses.

Heuristic Monte Carlo Tree Search vs Humans (us):

The Monte Carlo tree search AI with heuristics played very well against us with a threshold given of 10,000 and 100,000 playouts. We were not able to capture the corners of the board as the AI performed moves that did not allow any openings to the corner pieces. Although, the 100,000 playouts took significantly longer, even with 10,000 playouts the AI was consistently won.

Heuristic Monte Carlo Tree Search vs Pure Monte Carlo Tree Search

Heuristics (100,000) vs Pure (100,000) :
The Monte Carlo tree search with heuristics was able to win every game against the pure Monte Carlo tree search implementation. All corner pieces were captured by the AI with heuristics and the result did not change when switching which one went first.

Heuristics (10,000) vs Pure (100,000):
The pure Monte Carlo tree search was able to beat the implementation with heuristics. Here we handicapped the implementation with heuristics to perform only 10,000 playouts for each potential move whereas the pure Monte Carlo tree search performed

100,000 playouts for each potential move. The pure tree search won by only a few chips, hence the score difference was minimal. Furthermore, the implementation with heuristics did win when it went second and not first.

Heuristics (10,000) vs Pure (10,000):
When there are 10,000 playouts for the Heuristics MCTS and the Pure MCTS, the Heuristics MCTS AI was able to beat the Pure MCTS AI consistently. Because of the vast foresight of the heuristics MCTS AI, it was actually able to play very intelligently and does a lot of setup throughout the game. The setup includes strategies such as capturing the corners. Although it looks like it is losing at the beginning, by the last move, the heuristics MCTS AI captures all of the opponent's pieces in one motion and wins the game.

Heuristics (100) vs Pure (100):
Since the number of playouts was fixed to be very small, none of the AI were given much information about the game so the game so neither of the AIs were very good. This is shown when the Heuristics MCTS AI is able to beat the Pure MCTS AI when it goes first but not when it goes second.

Evaluation

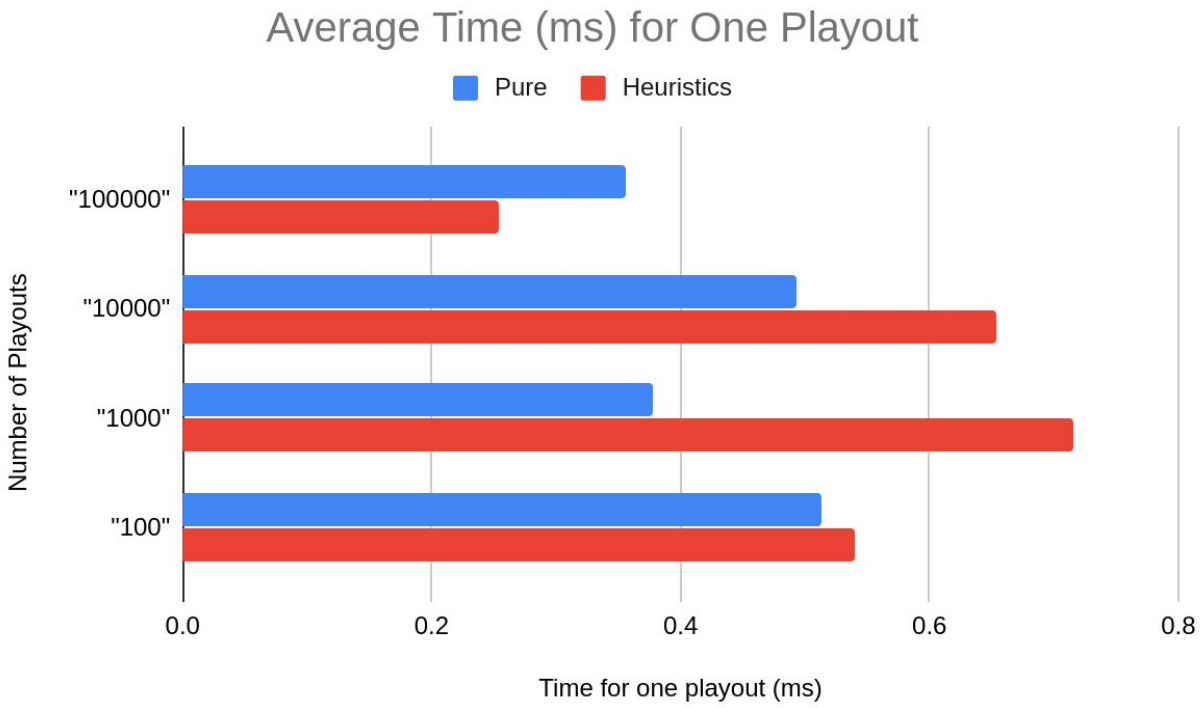
The algorithm with heuristics generally performed better, however, with a smaller threshold on the number of playouts, the pure Monte Carlo search tree performed better.

Cost

Obviously with a larger number of playout stimulations allowed, the time it took to perform moves significantly increased. Both algorithms (heuristic and pure) were quite efficient as shown below.

This graph represents the average time it took for the AI to perform a single playout in each of the games with different thresholds.

	Time for one playout (ms)	
Number of Playouts	**Pure**	**Heuristics**
"100000"	0.356	0.2548
"10000"	0.4932	0.6543
"1000"	0.3782	0.7157
"100"	0.5135	0.54



As predicted the AI with heuristics performed each single playout more slowly than the AI without heuristics, with exception to the game with 100,000 max stimulated playouts for each potential move. The heuristics taking longer makes sense as it calculates the score with three different heuristics and each of these functions can take more time to compute. However, interestingly enough, when the threshold was set to 100,000 max stimulated playouts, the AI with heuristics took on average a shorter amount of time compared to the pure Monte Carlo Tree Search.

Since the number of playouts was increased ten-fold to experiment, the time it took to perform a single move increased accordingly. Thus, we came to the conclusion that the Monte Carlo Tree Search with heuristics and a threshold of 10,000 maximum number of playouts per potential move was the optimum configuration for a fast yet intelligent Reversi playing AI.

Appendix

Heuristic (100,000) vs Pure Random (100,000)				
Move	**Total Time Taken (s)**	**Total Number of Playouts**	**Average Time (s)**	**Heuristic or Pure**
1	89.2103	400000	0.000223	Heuristic
2	49.0192	300000	0.0001634	Pure
3	125.52	500000	0.000251	Heuristic
4	56.33	300000	0.0001878	Pure
5	188.817	600000	0.0003147	Heuristic
6	66.6201	300000	0.0002221	Pure
7	211.381	600000	0.0003523	Heuristic
8	299.941	1100000	0.0002727	Pure
9	561.492	1200000	0.0004679	Heuristic
10	331.746	1100000	0.0003016	Pure
11	544.553	1100000	0.000495	Heuristic
12	271.02	900000	0.0003011	Pure
13	463.207	1000000	0.0004632	Heuristic
14	280.402	1000000	0.0002804	Pure
15	478.423	1100000	0.0004349	Heuristic
16	369.92	1200000	0.0003083	Pure
17	406.089	900000	0.0004512	Heuristic
18	406.322	1200000	0.0003386	Pure
19	497.592	1100000	0.0004524	Heuristic
20	378.696	1200000	0.0003156	Pure
21	446.018	900000	0.0004956	Heuristic
22	576.629	1400000	0.0004119	Pure
23	339.585	700000	0.0004851	Heuristic
24	599.073	1300000	0.0004608	Pure
25	510.62	1000000	0.0005106	Heuristic
26	596.563	1400000	0.0004261	Pure
27	881.522	1500000	0.0005877	Heuristic
28	649.223	1500000	0.0004328	Pure
29	567.592	1100000	0.000516	Heuristic

30	587.474	1400000	0.0004196	Pure
31	454.748	900000	0.0005053	Heuristic
32	473.727	1200000	0.0003948	Pure
33	405.977	1000000	0.000406	Heuristic
34	390.998	1100000	0.0003555	Pure
35	490.54	1100000	0.0004459	Heuristic
36	391.419	1100000	0.0003558	Pure
37	539.663	1200000	0.0004497	Heuristic
38	375.133	1100000	0.000341	Pure
39	412.376	1000000	0.0004124	Heuristic
40	350.935	1200000	0.0002924	Pure
41	578.34	1300000	0.0004449	Heuristic
42	231.959	1000000	0.000232	Pure
43	414.681	1200000	0.0003456	Heuristic
44	222.521	1000000	0.0002225	Pure
45	384.661	1200000	0.0003206	Heuristic
46	206.963	900000	0.00023	Pure
47	209.212	1000000	0.0002092	Heuristic
48	138.886	900000	0.0001543	Pure
49	136.273	1100000	0.0001239	Heuristic
50	40.6877	700000	0.0000581	Pure
51	11.2477	329930	0.0000341	Heuristic
52	1.65777	46593	0.0000356	Pure
53	0.27501	6910	0.0000398	Heuristic
54	0.057071	1505	0.0000379	Pure
55	0.015568	395	0.0000394	Heuristic
56	0.004986	114	0.0000437	Pure
57	0.001464	32	0.0000458	Heuristic
58	0.00026	6	0.0000433	Pure
59	0.0000083	2	0.0000042	Heuristic
60	0.0000034	1	0.0000034	Pure
Average Time Taken for Pure (s)				0.000356
Average Time Taken for Heuristics (s)				0.0002548

Heuristic (10,000) vs Pure Random (100,000)				
Move	**Total Time Taken (s)**	**Total Number of Playouts**	**Average Time (s)**	**Heuristic or Pure**
1	7.40627	40000	0.0001852	Heuristic
2	46.0263	300000	0.0001534	Pure
3	8.53958	40000	0.0002135	Heuristic
4	107.056	600000	0.0001784	Pure
5	11.3168	40000	0.0002829	Heuristic
6	89.5654	400000	0.0002239	Pure
7	29.6266	100000	0.0002963	Heuristic
8	145.025	600000	0.0002417	Pure
9	33.2563	100000	0.0003326	Heuristic
10	188.086	800000	0.0002351	Pure
11	32.619	90000	0.0003624	Heuristic
12	323.495	1200000	0.0002696	Pure
13	40.4232	100000	0.0004042	Heuristic
14	313.753	1000000	0.0003138	Pure
15	46.6157	1100000	0.0000424	Heuristic
16	283.414	900000	0.0003149	Pure
17	39.256	90000	0.0004362	Heuristic
18	454.744	1400000	0.0003248	Pure
19	49.5881	110000	0.0004508	Heuristic
20	482.735	1400000	0.0003448	Pure
21	63.5708	130000	0.000489	Heuristic
22	508.962	1400000	0.0003635	Pure
23	67.3959	130000	0.0005184	Heuristic
24	553.75	1500000	0.0003692	Pure
25	79.2219	1400000	0.0000566	Heuristic
26	709.906	1800000	0.0003944	Pure
27	89.3436	150000	0.0005956	Heuristic
28	710.574	1700000	0.000418	Pure
29	117.021	180000	0.0006501	Heuristic
30	830.467	1800000	0.0004614	Pure
31	125.704	190000	0.0006616	Heuristic

32	787.068	1700000	0.000463	Pure
33	125.834	200000	0.0006292	Heuristic
34	692.775	1600000	0.000433	Pure
35	124.518	200000	0.0006226	Heuristic
36	604.338	1400000	0.0004317	Pure
37	118.352	210000	0.0005636	Heuristic
38	498.955	1300000	0.0003838	Pure
39	83.2989	180000	0.0004628	Heuristic
40	393.216	1300000	0.0003025	Pure
41	85.691	180000	0.0004761	Heuristic
42	401.055	1400000	0.0002865	Pure
43	56.8587	140000	0.0004061	Heuristic
44	306.364	1300000	0.0002357	Pure
45	69.6389	150000	0.0004643	Heuristic
46	280.983	1300000	0.0002161	Pure
47	33.6603	120000	0.0002805	Heuristic
48	163.24	1200000	0.000136	Pure
49	19.2595	100000	0.0001926	Heuristic
50	87.2668	1100000	0.0000793	Pure
51	9.5704	80000	0.0001196	Heuristic
52	8.09516	257267	0.0000315	Pure
53	1.74147	53886	0.0000323	Heuristic
54	0.228504	7051	0.0000324	Pure
55	0.035575	1094	0.0000325	Heuristic
56	0.006671	199	0.0000335	Pure
57	0.001328	40	0.0000332	Heuristic
58	0.000286	9	0.0000318	Pure
59	0.0000026	2	0.0000013	Heuristic
60	0.0000026	1	0.0000026	Pure
Average Time Taken for Pure (s)				0.0002569
Average Time Taken for Heuristics (s)				0.0003432

Heuristic (10,000) vs Pure Random (10,000). Pure random goes first				
Move	**Total Time Taken (s)**	**Total Number of Playouts**	**Average Time (s)**	**Heuristic or Pure**
1	9.15413	40000	0.00022885325	Pure
2	11.1117	30000	0.00037039	Heuristic
3	10.9929	40000	0.0002748225	Pure
4	24.8924	60000	0.0004148733333	Heuristic
5	16.6493	50000	0.000332986	Pure
6	22.6207	40000	0.0005655175	Heuristic
7	22.2061	60000	0.0003701016667	Pure
8	35.0848	60000	0.0005847466667	Heuristic
9	23.6954	60000	0.0003949233333	Pure
10	39.1449	60000	0.000652415	Heuristic
11	26.8008	60000	0.00044668	Pure
12	21.4888	30000	0.0007162933333	Heuristic
13	37.326	70000	0.0005332285714	Pure
14	44.1748	60000	0.0007362466667	Heuristic
15	52.5974	90000	0.0005844155556	Pure
16	37.7558	50000	0.000755116	Heuristic
17	35.3665	70000	0.0005052357143	Pure
18	38.3381	50000	0.000766762	Heuristic
19	27.5962	60000	0.0004599366667	Pure
20	31.296	40000	0.0007824	Heuristic
21	30.5612	50000	0.000611224	Pure
22	63.0639	90000	0.00070071	Heuristic
23	24.9469	50000	0.000498938	Pure
24	67.106	90000	0.000745622222	Heuristic

			2	
25	16.8472	40000	0.00042118	Pure
26	106.287	110000	0.0009662454545	Heuristic
27	12.2334	20000	0.00061167	Pure
28	88.3902	110000	0.0008035472727	Heuristic
29	29.9897	50000	0.000599794	Pure
30	75.9359	100000	0.000759359	Heuristic
31	25.7335	40000	0.0006433375	Pure
32	79.5861	100000	0.000795861	Heuristic
33	12.5873	20000	0.000629365	Pure
34	113.054	130000	0.0008696461538	Heuristic
35	14.5083	20000	0.000725415	Pure
36	102.484	140000	0.0007320285714	Heuristic
37	27.6945	50000	0.00055389	Pure
38	88.3203	130000	0.0006793869231	Heuristic
39	24.0252	50000	0.000480504	Pure
40	87.7247	130000	0.0006748053846	Heuristic
41	15.5017	40000	0.0003875425	Pure
42	75.4937	130000	0.0005807207692	Heuristic
43	12.2216	30000	0.0004073866667	Pure
44	65.217	130000	0.0005016692308	Heuristic
45	19.2391	60000	0.0003206516667	Pure
46	41.3797	100000	0.000413797	Heuristic
47	21.5594	90000	0.0002395488889	Pure
48	27.2413	90000	0.0003026811111	Heuristic

49	11.1003	60000	0.000185005	Pure
50	18.7856	100000	0.000187856	Heuristic
51	6.04609	60000	0.0001007681667	Pure
52	1.79554	36715	0.0000489048073	Heuristic
53	0.386324	7638	0.00005057920922	Pure
54	0.04452	923	0.0000482340195	Heuristic
55	0.01216	237	0.00005130801688	Pure
56	0.001957	40	0.000048925	Heuristic
57	0.00095	16	0.000059375	Pure
58	0.000181	3	0.00006033333333	Heuristic
59	0.000154	2	0.000077	Pure
60	0.0000051	1		Heuristic
Average Time Taken for Pure (s)				0.0003929
Average Time Taken for Heuristics (s)				0.0005609

Heuristic (10,000) vs Pure Random (10,000). Heuristic goes first				
Move	**Total Time Taken (s)**	**Total Number of Playouts**	**Average Time (s)**	**Heuristic or Pure**
1	13.1601	40000	0.000329	Heuristic
2	7.70314	30000	0.0002568	Pure
3	16.0775	40000	0.0004019	Heuristic
4	17.3208	60000	0.0002887	Pure
5	22.4699	50000	0.0004494	Heuristic
6	24.9565	70000	0.0003565	Pure
7	27.0473	50000	0.0005409	Heuristic
8	27.8457	70000	0.0003978	Pure
9	42.9689	70000	0.0006138	Heuristic
10	27.1406	60000	0.0004523	Pure
11	49.3425	70000	0.0007049	Heuristic
12	37.2484	80000	0.0004656	Pure

13	36.8946	50000	0.0007379	Heuristic
14	48.362	90000	0.0005374	Pure
15	58.3003	80000	0.0007288	Heuristic
16	47.9542	90000	0.0005328	Pure
17	63.5853	80000	0.0007948	Heuristic
18	60.2206	120000	0.0005018	Pure
19	46.4851	50000	0.0009297	Heuristic
20	71.802	110000	0.0006527	Pure
21	78.6388	90000	0.0008738	Heuristic
22	52.0997	90000	0.0005789	Pure
23	101.97	110000	0.000927	Heuristic
24	78.7864	110000	0.0007162	Pure
25	166.03	160000	0.0010377	Heuristic
26	110.867	140000	0.0007919	Pure
27	166.249	150000	0.0011083	Heuristic
28	127.041	150000	0.0008469	Pure
29	146.736	150000	0.0009782	Heuristic
30	135.439	160000	0.0008465	Pure
31	149.245	130000	0.001148	Heuristic
32	148.212	170000	0.0008718	Pure
33	156.116	150000	0.0010408	Heuristic
34	143.932	160000	0.0008996	Pure
35	131.346	150000	0.0008756	Heuristic
36	126.401	150000	0.0008427	Pure
37	220.798	190000	0.0011621	Heuristic
38	132.486	160000	0.000828	Pure
39	208.07	190000	0.0010951	Heuristic
40	133.696	170000	0.0007864	Pure
41	160.033	180000	0.0008891	Heuristic
42	110.831	160000	0.0006927	Pure
43	101.613	160000	0.0006351	Heuristic
44	73.1925	130000	0.000563	Pure
45	72.9411	140000	0.000521	Heuristic
46	48.0032	110000	0.0004364	Pure

47	51.1454	120000	0.0004262	Heuristic
48	26.0132	100000	0.0002601	Pure
49	33.2879	100000	0.0003329	Heuristic
50	14.9759	90000	0.0001664	Pure
51	11.3707	80000	0.0001421	Heuristic
52	5.88403	80000	0.0000736	Pure
53	0.65003	13020	0.0000499	Heuristic
54	0.13772	2730	0.0000504	Pure
55	0.015993	286	0.0000559	Heuristic
56	0.005743	115	0.0000499	Pure
57	0.000658	14	0.000047	Heuristic
58	0.000284	6	0.0000473	Pure
59	0.000104	2	0.000052	Heuristic
60	0.0000037	1	0.0000037	Pure
Average Time Taken for Pure (s)				0.0004932
Average Time Taken for Heuristics (s)				0.0006543

Heuristic (1000) vs Pure Random (1000)				
Move	**Total Time Taken (s)**	**Total Number of Playouts**	**Average Time (s)**	**Heuristic or Pure**
1	1.12056	4000	0.0002801	Heuristic
2	0.61556	3000	0.0002052	Pure
3	1.32725	4000	0.0003318	Heuristic
4	1.4271	6000	0.0002379	Pure
5	1.21112	3000	0.0004037	Heuristic
6	1.42575	5000	0.0002852	Pure
7	3.44545	7000	0.0004922	Heuristic
8	2.03609	7000	0.0002909	Pure
9	4.55859	9000	0.0005065	Heuristic
10	2.42659	7000	0.0003467	Pure
11	6.04738	10000	0.0006047	Heuristic
12	5.14513	11000	0.0004677	Pure
13	5.12526	8000	0.0006407	Heuristic

14	2.04549	5000	0.0004091	Pure
15	4.6908	8000	0.0005864	Heuristic
16	3.82825	9000	0.0004254	Pure
17	7.58323	10000	0.0007583	Heuristic
18	4.91333	10000	0.0004913	Pure
19	10.4716	13000	0.0008055	Heuristic
20	3.38618	7000	0.0004837	Pure
21	15.2706	15000	0.001018	Heuristic
22	3.30795	7000	0.0004726	Pure
23	20.1669	16000	0.0012604	Heuristic
24	4.7983	9000	0.0005331	Pure
25	22.2937	18000	0.0012385	Heuristic
26	3.63419	8000	0.0004543	Pure
27	25.1182	18000	0.0013955	Heuristic
28	3.79624	7000	0.0005423	Pure
29	29.4898	20000	0.0014745	Heuristic
30	7.04011	10000	0.000704	Pure
31	28.6997	20000	0.001435	Heuristic
32	7.72557	11000	0.0007023	Pure
33	26.1068	20000	0.0013053	Heuristic
34	5.63626	9000	0.0006263	Pure
35	22.1762	19000	0.0011672	Heuristic
36	4.4496	8000	0.0005562	Pure
37	22.3784	18000	0.0012432	Heuristic
38	4.18283	8000	0.0005229	Pure
39	17.2243	16000	0.0010765	Heuristic
40	5.77045	10000	0.000577	Pure
41	12.3619	14000	0.000883	Heuristic
42	4.89743	10000	0.0004897	Pure
43	8.93228	12000	0.0007444	Heuristic
44	4.14304	10000	0.0004143	Pure
45	7.42119	12000	0.0006184	Heuristic
46	2.74688	8000	0.0003434	Pure
47	4.33237	10000	0.0004332	Heuristic

48	2.27259	8000	0.0002841	Pure
49	2.85342	9000	0.000317	Heuristic
50	1.69096	9000	0.0001879	Pure
51	2.18252	10000	0.0002183	Heuristic
52	0.957998	7000	0.0001369	Pure
53	0.36195	5596	0.0000647	Heuristic
54	0.190563	3854	0.0000494	Pure
55	0.038153	769	0.0000496	Heuristic
56	0.003903	84	0.0000465	Pure
57	0.001433	28	0.0000512	Heuristic
58	0.000337	6	0.0000562	Pure
59	0.000133	2	0.0000665	Heuristic
60	0.000004	1	0.000004	Pure
Average Time Taken for Pure (s)				0.0003782
Average Time Taken for Heuristics (s)				0.0007157