# CMPT310 assignment 5 report

I wrote this assignment in C ++ because it is more efficient than python. I first conceived the structure of the entire file, and decided to write a class for the Reversi, which contains many rules of Reversi. I created a header file for the class and wrote the public functions in the corresponding cpp file. The attribute of the Reversi is very simple, contains only a two-dimensional array for the chessboard, a counter for black, a counter for white and a board size, which is 8. The function I wrote is for the rules of the game, such as isvaildmove(int x, int y, int tile) is used to judge whether each move of the user is correct. checkWin(int tail) is used to check whether the winner has been decided, and if yes, who is the winner.

For pure Monte Carlo Tree Search and Monte Carlo Tree Search with heuristics, I wrote two functions for them in the corresponding header file, the method I used for pure Monte Carlo Tree Search is exactly the same as what I did in assignment 3: firstly I got the valid move index for the computer, and for each valid move index, I let computer play with itself 400 times, for each step, the computer choose a position from the valid move index randomly. Because the rule of Reversi is that only if both sides have no chess to play, a victory can be determined, so I set a variable of type int to control the order in which the two players play. If the computer wins, If the computer wins, a point will be added to the corresponding valid index, if the player wins, a point will be deducted in the corresponding index, else if it’s a tie, no change.

For Monte Carlo Tree Search with heuristics, I used two other methods to share the weight of the score, one is the weight matrix and the other one is number of opponent pieces flipped in each move. According to my search on the Internet, some positions are relatively good, and some position are relatively bad(e.g. Positions in the four corners are more likely to win than positions in the middle) and we hope AI can play on position with larger weight as much as possible. So that is a helpful heuristics. The number of opponent pieces flipped in each move was came out by myself, because I think the larger the number of opponent pieces flipped in each step, there’s more likely to win. I added these three scoring methods together, the weight matrix weight 30%, the number of opponent pieces flipped in each move weight 20% and the random play weight 50%. Because I still use the random move when the computer play against itself, the running speed has not been greatly improved, but the win rate of the Monte Carlo Tree Search with heuristics has improved significantly. And below are the graph and table of the win rate and playouts per second.

**NB: way to run the code: directly run the code and follow the instructions in the terminal!**

Winning rate table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1st test | | 2nd test | | 3rd test | |
| winner | Round | winner | Round | winner | Round |
| AI(heuristic) | 25 | AI(heuristic) | 24 | AI(heuristic) | 27 |
| AI(no heuristic) | 3 | AI(no heuristic) | 6 | AI(no heuristic) | 2 |
| Draw | 2 | Draw | 0 | Draw | 1 |
|  | Total:30 |  | Total:30 |  | Total:30 |

Performance table (As the game progresses, the playouts per second will increase)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Performance(playouts/sec) | | | | | | | | | | | | | | | | | | | |
| AI with heuristic | | | 203 | 214 | 232 | | 256 | | 278 | | 283 | | | 304 | | 341 | | 368 | |
| AI no heuristic | | | 236 | 247 | 251 | | 283 | | 298 | | 323 | | | 334 | | 358 | | 376 | |
| Performance(playouts/sec) | | | | | | | | | | | | | | | | | | | |
| 379 | 455 | 536 | 651 | 740 | 896 | 1025 | | 1348 | | 1564 | | 1645 | 2096 | | 3582 | | 6959 | | 10041 |
| 410 | 453 | 512 | 663 | 792 | 840 | 1115 | | 1286 | | 1468 | | 1673 | 2140 | | 3694 | | 7724 | | 13290 |