**Assignment 5: Connect 4**

**Information about project**

How to compile: g++ -o *executable\_name* Main.cpp Node.h Node.cpp Game.h Game.cpp Board.h Board.cpp coord.h

How to run after compiling: ./*executable\_name*

Time limit can be changed in play\_game() function by variable time\_limit  
(Main.cpp line 31)

Play game takes an integer as a parameter.

1. 1 = Player vs Pure Monte Carlo Tree Search
2. 2 = player vs Second Monte Carlo Tree Search with Heuristic and other adjustments
3. 3 = Pure Monte Carlo vs Second Monte Carlo

**Note** I ran my project on ubuntu in the csil computers and time seems to be in microseconds vs on my own pc which was milliseconds. If results start coming out very quickly, change *X*000 to *X*000000

Sorry for the inconvenience.

**Implementation**

The game has been broken up into several classes. The Board class is responsible for representing the board and perform actions on the board. These actions consist of keeping track of pieces on the board, applying moves by the player or Monte Carlo Tree Search, providing the height and weight of the board and printing the board. Retrieving pieces can be done by using the Coord class that is responsible for holding an x and y position of a piece on the board. To ensure that moves made adhere to the rules of connect four, the Games class is used.

The Games class is responsible for checking if the game is over and who won, placing pieces in the appropriate position on the board, and has a list of next possible moves which is updated after a move is made. After each turn, a check to see whether the game is won or not is done. This is done by checking the four directions (vertical, horizontal, and the two diagonals) of the last move made. If a series of 4 of the same characters are found the game is over.

To move between states of the game after a move is made, the Node class is used. The Node class hold a state of the game and has a variable to retain wins, ties, and losses for Monte Carlo Tree Search and keeps track of which player’s turn it is. When performing random playouts in Monte Carlo Tree Search, when the game is won, the win, loss and tie count is updated depending on who won.

The Play game function in Main.cpp is responsible for driving the game. It will determine which player should move and which version of Monte Carlo to use. My implementation of Monte Carlo decided wins based on a win percentage instead of absolute max wins. This was an attempt to try to make Monte Carlo Tree Search avoid choosing an option had short losing games. My version also checked for sequences of 3 continuous opponent characters to check if their next move would end the game and attempts to block that move.

**Performance**

When performing Monte Carlo Tree Search with a 5 second time limit. The time limit is divided by the number of next possible moves (7 unless a column has been filled). 50 – 110 full playouts are performed for each of the next possible moves. As columns fill up, more playouts are performed due to fewer options for next moves being available. Win rate between the two versions of Monte Carlo was about equal favouring the player who played first slightly. Playing against myself, I was unable to beat either of the versions of Monte Carlo.

Number of Playouts per time limit

|  |  |  |
| --- | --- | --- |
| Seconds | Playouts per possible next move | Total Playouts |
| 1 | ~10 | ~60 |
| 2 | ~20 | ~120 |
| 3 | ~30 | ~180 |
| 4 | ~40 | ~240 |
| 5 | ~50 | ~300 |

Playing the two versions of Monte Carlo against each other, the win rates between the two are very similar. My version did win a couple of games more when going first vs pure MCTS, but possibly within the margin of error. The version that went first usually won as stated in the assignment description.

Win rates over 25 games

|  |  |  |
| --- | --- | --- |
| Time limit per move | Pure MCTS goes first | My MCTS goes first |
| 1 | 20 | 23 |
| 2 | 22 | 21 |
| 3 | 21 | 24 |
| 4 | 22 | 25 |
| 5 | 23 | 23 |