Connect 4 – Multiplayer game analysis.

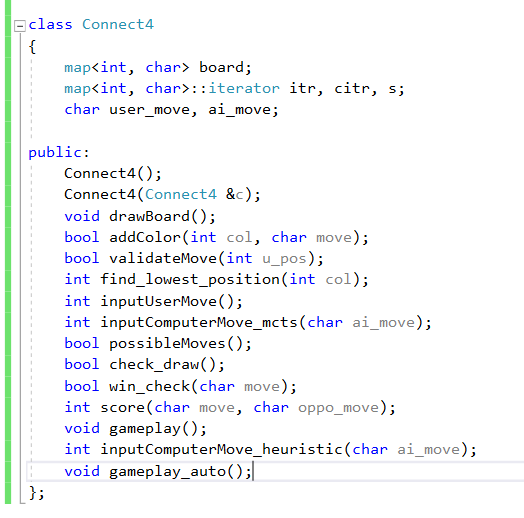
Using Monte Carlo Tree Search and Heuristics

The project is implemented in OOP with a class of ‘Connect4’ which had all the required methods and variables. There are two algorithms to play against: one is Monte Carlo Tree Search and the other is a heuristic (explained later). **This project was run on Visual Studio on Windows, timings differ when it is run on Ubuntu computers. Instructions at last.**

# Class methods and variables

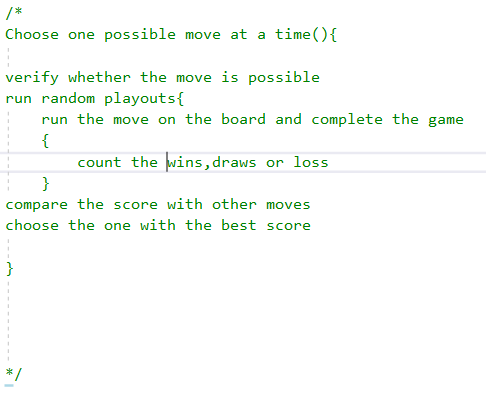
The following image is the class used in the project.

Important: Gameplay are the functions that initiate a game to happen. Their names specify what kind of algorithm you will play against. ‘Gameplay\_auto()’ is a function to let the two algorithms play against each other.



# Implementations

Pseudo-code for Monte Carlo Tree Search:



Implementing the heuristic function

* **Heuristic function calculates the score by considering all the possible lines that could lead to a win. All these lines have 4 spaces to fill. These lines could be vertical diagonal or horizontal.**
* **Depending upon the situation, some of them might be occupied by a red tile or a yellow tile or with no tiles. If all of them are filled with one particular tile then that player wins.**

**Score Calculation**

* **We give one point to each type of tile that occupies a winning line. To calculate the score with respect to one color we subtract the number of points of other color from the number of points of that color.**
* **Consider a line which has 3 red tiles and 1 yellow tile, the score with respect to red is 3-1=2, and score with respect to yellow is 1-3=-2**
* **Similarly, winning score is 4 for either of the colors.**

Pseudo- Code for heuristic

Choose one possible move {  
 Validate move;  
 run random playouts {  
 run the move on board, calculate score for every move until game ends and add  
 }

Compare the scores and return the move with maximum score

# Analyzing the Monte Carlo Tree Search

I played 20 games. For most of my games, my moves were weak. The time cutoff was set at 10 seconds and was then later changed to 8 seconds.

The MCTS usually took 6-8 seconds to come up with one move, but it was good at not losing.  
Moreover the maximum time it took to come up with a move was 8.9 seconds. It averaged around 7.5 seconds for all moves in 20 games.

Out of 20 games, it won 15 and I won 5 while 1 was drawn.

The minimum time it took to come up with a move was 6.01 seconds.

Talking about performance it averaged about 13.48 random playouts per second, and maxed out at 16.621 while the lower bound was set at 11.1

Here are the graphs for the algorithm:

# Analyzing the Heuristic function

Again, played 20 games with the heuristic function, and found it to be a bit slower and less good at the game.  
Numerically it averaged around 17.6 seconds for one move. Cutoff was put at 20.  
The maximum time it took was around 19.97 and minimum was 15.02

Out of 20 games, it won 8 games while I won 12. The moves that I made were again not that amazing.

In term of performance, it averaged around 1.4 playouts per second compared to 13.48 playouts per second of monte carlo tree search.

So it is okay to conclude that monte carlo is much better.

# Heuristic vs Monte Carlo

When I played both algorithms against each other it turned out to a quite expected solution.

Monte carlo won 19 of those 20 games, while only 1 was drawn.

So, to conclude monte carlo is faster, efficient and optimal in case of Connect 4.

INSTRUCTIONS TO RUN

1. To run on ubuntu, go to terminal, change to the file directory, run command:  
   g++ -o a5 a5.cpp  
   ./a5
2. To run on windows on visual studio, include precompile header file and run normally.