CMPT310
Connect 4 Game with Monte Carlo Tree Search & Two-Step Predict Heuristic

Simon Fraser University - Summer 2019

Bowen Wang

301267523

Part 1 – Pure Monte Carlo Tree Search

With the same idea from assignment 4, random dropping has been used for searching for best move in the first part of this assignment. In **a5_q1.cpp**, 15000 times of simulation for each possible move has been taken by the agent to find out whichever move could be the best. During the simulation, the program do ***check_win()*** test on each outcome state, if rather the player or computer win, assign a score based on how many blank space left on the board. After all possible move have been checked, the program choose to pick the move with highest score.

To have a closer look at the functionality in my code. I define a class named ***connect4*** with functions:

> ***display():*** to visualize the board and give a direct user interface of the current state.
>
> ***Move_first():*** to set player's position (offensive or defensive)
>
> ***player_move():*** ask for player for their move.
>
> ***check_win():*** check whether the state comes to a result of winner, return the winner.
>
>> The check have been separate into 4 categories, horizontal, vertical, diagonal (left top to right botom) and diagonal (left bottom to right top). I designed the process of checking the chips nearby last drop chip to reduce comparison and improve performance as this function are going to used after every simulation which can be called most frequently.
>
> ***legal_move():*** determine whichever column can be added new chip
>
> ***next_move():*** determine whichever move is the best move to move next.
>
>> Use pure Monte Carlo tree search based totally on random picking and 15000 times of simulation. Chose the move with highest score calculated by possible computer win states.

Part 2 – Two Step Predict Heuristic

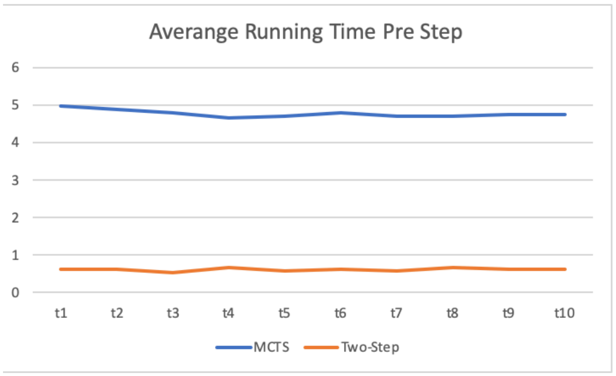
As the random pick takes too much time and have a chance of mis-predict. I designed a heuristic to minimize the mis-predict and reduce time cost. The idea is to make sure player cannot win right after current move, moreover, choose the most aggressive move that can win in further 1 to 2 steps. The priority based on how fast computer can win.

To have a closer look at the functionality in my code, I define a different class named ***Connect4*** with functions ***display()*** and ***player_move()*** same as before, the difference is I designed a class structure to have a easily access to add and change specific chip as I need to simulate step move between player and computer in 2 rounds.

Performance

As both methods have a well-designed algorithm and I'm not good at playing board game personally, I didn't win any of game during the competition with neither agent. Afterward, I designed a competition between 2 agents, and tested for 10 times, all the result comes to a draw, which indicates they have similar level of difficulty/intelligence.

But when comparing the time cost, pure Monte Carlo tree search takes 4.76 seconds in average while two-step predict heuristic takes only 0.62 second. The big gap between two algorithms on time cost indicates the second one has a significant higher efficiency.



Analysis

When I implement the pure Monte Carlo tree search, I used 15000 times of random simulation for each possible move. Although 15000 is a large number, the randomness can result in a duplicate of states which waste time and prevent the agent improve its correctness. Moreover, further steps carry less effect on how good current move is, since there are many possible moves between current states and a specific further step. But the agent spends same amount of resource to solve it, which slows down the responding speed. For two-step predict heuristic, only two steps have been predicted and since there is no more than 7*7 = 49 possible states, there is no need to simulate which saves great amount of time. Meanwhile, the closest 2 step has great weights for people to refer to, the correctness would not drop much if cut off the rest of states.

For further work, to improve the correctness, I have an idea of combine a heuristic into the Monte Carlo tree search and change the data structure of it, say, build a search tree to record states have been checked.