Final Project

Reversi with Monte-Carlo Tree Search

CMPT 310 Summer 2020
Yixu Ye (301368702)
yixuy@sfu.ca
Zirui Huang (301307482)
ziruih@sfu.ca

Introduction

In this report, we are going to explain in detail how we implemented our Reversi AI based on pure Monte Carlo Tree Search (PMCTS) algorithm. We would also like to explore and analyze the performance of PMCTS AI in terms of its efficiency and winning ratio against other AIs including Random AI and MCTS AI with heuristics. Finally, the limitation and potential improvement of our approach are mentioned.

Technical Detail

1. Theory

   Pure Monte Carlo Tree Search (PMCTS) algorithm heavily relies on randomness, it builds a game tree and uses results from rollouts to guide search; a rollout is a path that descends the tree with a randomized decision at each play until reaching a leaf [1].

   Our implementation of PMCTS can be generalized into four steps:

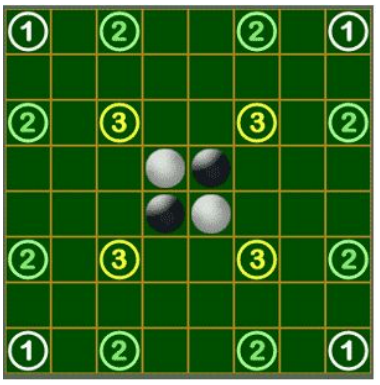
   - ***Find legal moves****:* for each turn of AI, we begin by finding all legal moves in the current game state. Then we apply following two steps repeatedly to each of these moves.
   - ***Update State****:* add one of the legal moves to the board and update the game state.
   - ***Simulation****:* (1)find all legal moves in the current game state, (2) choose one legal move randomly, then (3) advance the game state. This phase ends when we reach a state where a game is finished (two consecutive passes happened) [2]. Simulation will last in a given amount of time (5 seconds).
   - ***Selection****:* Select the move with the greatest winning count as the AI move.

2. The idea of heuristic used for PMCTS[3]:

   For the second (improved) version of AI, we added the following heuristics.

   - ***Corners first****:* With this heuristic, the AI will directly choose corner move, if any. The corners on the board are important since there is no way that the opponent can flip the corner tile. And from the statistics, after we get the corner filled by AI, its winning ratio increases dramatically, so it is decisive to the game result.
   - ***Edge more little weight***: This heuristic will give extra weight if AI's next move is at the corner on the game board. The tile on edges on the board is more valuable than the tile in the middle since there is only one direction that could change the tile's color since along with the edge.

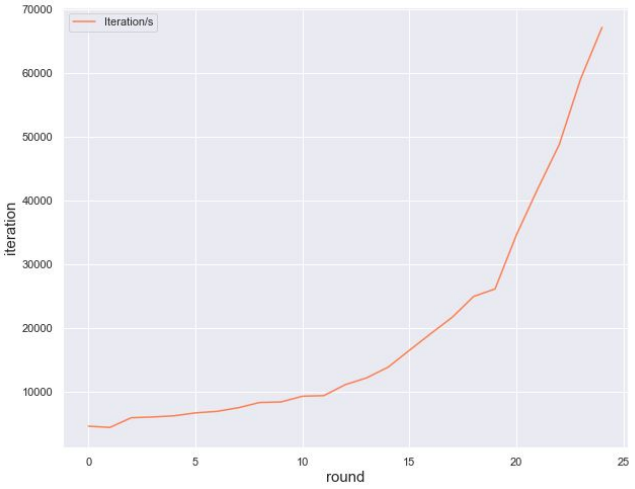
- ***Valuable spots more weight***: There are some sweet spots besides corners and edges, and we can give them extra weight on simulation. The positions of these power points are indicated on the picture.



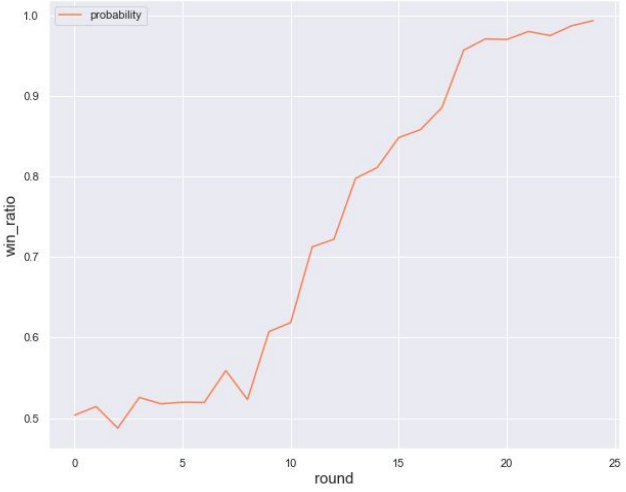
- ***More next move more weight***: We find the number of valid moves can also have an effect on the result. Therefore, we give more weight to move if it is followed by a large number of valid moves in later random playout. (E.g. Taking position A allows me to have 3 valid moves in next round, and position B allows me to have 5 valid moves in next round, I would choose B)

Statistic Analysis

1. ***Performance:*** we find the number of iterations per second increases constantly as more tiles have been covered on the board, this is because the search space becomes smaller and smaller. In the beginning, the average iteration rate is about 5000/s, this rises to 70000/s at the end.



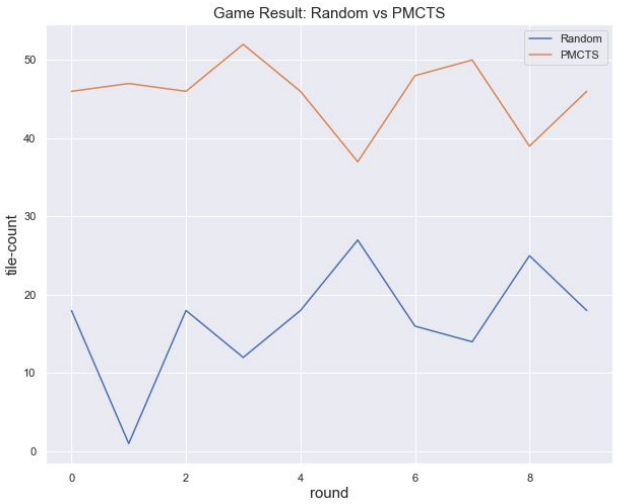
2. ***Potential Winning Ratio:*** we account *potential winning ratio* in random playout as $\frac{win\ round}{total\ round}$, this statistic allows us to have a sense of how the potential game result changed over time. No surprise, the potential winning ratio is about 50% at the beginning, and if everything is good, the ratio can increase constantly to about 100% near the end of the game.



3. *Game Result*

a. play with random AI

We have created a 'silly' opponent of our PMCTS AI which always chooses the next move randomly. Let's have a look at the game results between PMCTS AI and Random AI. From the plot, we can see PMCTS AI defeated Random AI in all 20 rounds, just as we expect.



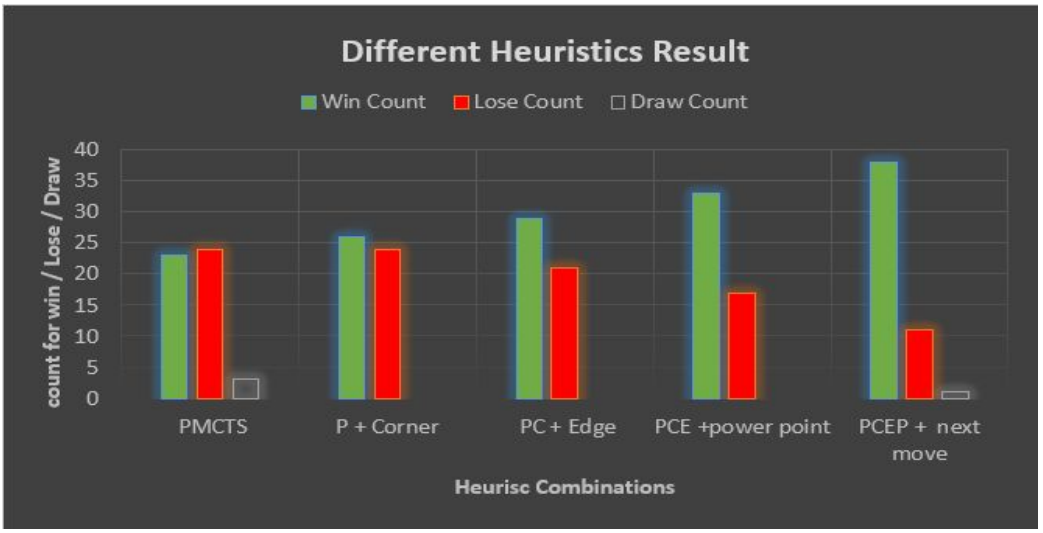
b. play with enhanced PMCTS AIs

To make stronger AI, we decide to add more herustics to the original PMCTS AI step by step. We would like to know how the enhanced AIs play against the original one. First, we believe the original PMCTS AI would have 50% to win when playing against itself. The data indicates what we think is correct.
We started by adding the corner heuristics and we are a little bit disappointed because it has no obvious effect. The reason is that the original PMCTS AI is able to occupy corners if there is one.

Therefore, we further added edge heuristic to AI such that more weight would be given to moves on the edges. Through the internet, we find there are some 'power points' on the board that can have an effect on the game result, so we give more weight to these points. We are surprised to discover this combination (PMCTS+Corner+Edge+Power Points) can result in a 15% increase in the winning rate.

Finally, as we discussed in the previous section, we give more weight to moves that end with many valid moves in following simulations. It seems this heuristic works pretty well. One difficulty we encountered is how to reasonably assign weight to this heuristics, we have experiment some numerical values and ends up adding (50 * number of valid moves) to the score.

Different Heuristics Result					
Compete with PMCTS	PMCTS	P + Corner	PC + Edge	PCE + power point	PCEP + next move
Win Count	23	26	29	33	37
Lose Count	26	24	21	17	13
Draw Count	1	0	0	0	0



Discussion and Conclusion

- ***Experiment:***
  We started the project by challenging online Reversi AI. In the beginning, we just asked our AI to move randomly and this did not work well. We then changed it to PMCTS AI with ***Most Win*** scoring mechanism, it performed much better and we found the potential winning ratio (win count / total round played) constantly increased along with each step.

  However, we get stuck at the time when we try to defeat Nintendo Switch AI (hard mode). Our AI may lose with a strict time limit on the random playout. We find the number of random playouts can have a considerable effect on the performance of PMCTS AI, and the algorithm seems to be doing better each time we increase the time limit (so the number of iterations) by an order of magnitude. We try to optimize the implementation of PMCTS algorithm so the AI can work more efficiently, but it did not help much. Therefore, adding heuristic to AI may be a reasonable alternative.

- ***Interesting Finding***

1. The control of corners can have a significant effect on the game result. We find each time our AI occupies a corner, there can be a surge of winning ratio in later simulation. Also, if we start simulation from a

corner, it is interesting to find that there tend to be more winning rounds compared to simulations from other points. Therefore, our AI is able to automatically take corner position, especially in the early and middle phase (when the average winning ratio in simulation is below 90%). After the winning ratio reaches a threshold of about 95% (so no matter where the simulation starts, 95 percent of simulations ends with a win), the AI may stop choosing a corner.

2. Being passed (no available tile to move) can have a detrimental effect on the performance of our AI. The winning ratio in later simulations can drop if this round is a pass. Also, we find it becomes hard for our AI to win the game if it has been trapped to a pass during the game.

- ***Limitation and Potential Improvement***

1. Though heuristics can improve the overall performance of our AI, it was unable to make it perfect because heuristic cannot predict the future. For instance, occupying edge points if available can usually be a good strategy, but taking edge points does not guarantee you can win the game.
2. As we have discussed, **pass** can have a considerable effect on the final game result. Each time if the player could make the other pass, it would increase the chance of winning dramatically, so from our perspective, the monte carlo tree is required for checking the next pass. In later work, we may seek to add heuristics that can trap opponents to a pass, so the performance of our AI can be further improved.

Reference:

1. *Christopher D. Rosin (2019). Nested Rollout Policy Adaptation for Monte Carlo Tree Search. Retrieved from https://www.ijcai.org/Proceedings/11/Papers/115.pdf*

2. *Muens, P. (2020, January 14). Game AIs with Minimax and Monte Carlo Tree Search. Retrieved from https://towardsdatascience.com/game-ais-with-minimax-and-monte-carlo-tree-search-af2a177361b0*

3. *How to win at Reversi. Retrieved from https://guides.net4tv.com/games/how-win-reversi#:~:text=The%20basic%20moves%20of%20Reversi,your%20stone%20in%20that%20square.*

4. *An artificial intelligence agent playing Reversi http://ceur-ws.org/Vol-1107/paper2.pdf*