Project 4 - Build an Adversarial Game Playing Agent

Antoine Mertz

02 January 2020

Introduction

The goal of this project is to build a smart agent to play knights isolation, using adversarial search techniques. I've chosen to implement an advanced technique not covered in the lecture but mentioned several times in the project statement: Monte-Carlo Tree Search (MCTS), corresponding to chose option 3. The statement suggests to search on the internet for other adversarial search techniques not in the lectures and see the applications on other games like chess, go or connect4. After searching a bit on the internet and implementations on Github, I've found one implementation close to what I wanted to develop (https://github.com/Alfo5123/Connect4/blob/master/game.py) and I've found MCTS explanations on Youtube videos (https://www.youtube.com/watch?v=onBYsen2_eA).

Implementation

I let reviewer take the time to look MCTS implementation in the `my_custom_player.py` file. I tried to describe well classes and functions implemented.

Results

I've plaid 100 games for each experiment.

Opposition	Fair Games	Time to explore (s)	% of victories
CUSTOM PLAYER vs RANDOM	NO	0.1	100.0
CUSTOM PLAYER vs RANDOM	YES	0.1	98.0
CUSTOM PLAYER vs GREEDY	NO	0.1	94.0
CUSTOM PLAYER vs GREEDY	YES	0.1	92.0
CUSTOM PLAYER vs MINIMAX	NO	0.1	82.0
CUSTOM PLAYER vs MINIMAX	YES	0.1	83.0
CUSTOM PLAYER vs MINIMAX	YES	0.05	72.0

As a baseline, I've chosen MINIMAX player with fair games.

Opposition	Fair Games	% of victories
MINIMAX vs RANDOM	YES	96.0
MINIMAX vs GREEDY	YES	71.0

Analysis

If we compare to Random player, our algorithm almost always win. We cannot consider the Random player as a baseline to evaluate our MCTS player because we had some intelligence in comparison to random player, and we except to always win against player that don't know what he's doing.

So as baseline I've chosen to evaluate our algorithm against MINIMAX agent implemented for us. And to evaluate performance of our player, it is good to play fair games that are not bias by opening moves because we just play random move as opening ones in our custom player.

And the result are encouraging for our player: against MINIMAX player, playing 100 fair games, we win 83% of the times.

But if we reduce time for our agent to expand and explore tree (time to explore = 0.05 second), the performance is decreasing a lot. So our algorithm is very sensitive to time to explore and expand tree. So if we don't have a lot of time to compute and play next move, maybe our agent is not the best option.

- How much performance difference does your agent show compared to the baseline?

But if we look our performances in comparison the performance of the MINIMAX, our custom player is much successful than MINIMAX player (92% vs 71% of victories against GREEDY player). So yes, I'm happy with this first development of custom player.

- Why do you think the technique you chose was more (or less) effective than the baseline?

So if we have some time to explore tree and randomly expand it, our advanced search technique is more successful than MINIMAX one. I think that one of the reason of our success is that MCTS algorithm doesn't need robust and advanced heuristic to perform well. And the space of the knights isolation game is not so big so the MCTS player can explore lot of branches and evaluate the best move it can take from a given state. And MCTS allow us to just grow and explore tree in the most promising parts of the tree and not consider all possible actions.