Adversarial Game Playing Agent

For this project, I went ahead with Option 3- Building an Agent Using Advanced Search Technique. The technique which I have implemented is Monte Carlo Tree Search as it has been proven effective in games like Go, Chess and Shogi.

Choose a baseline search algorithm for comparison (for example, alpha-beta search with iterative deepening, etc.). How much performance difference does your agent show compared to the baseline?

I evaluated the performance of the technique with Random, Greedy and Minimax search techniques available in sample agents -

- With 'fair' flag on to mitigate differences caused by opening position
- Without the 'fair' flag on to accommodate randomness in opening positions

Here are the results with percentage of games won using this technique-

Fair

Number of matches	RANDOM	GREEDY	MINIMAX	SELF
10	90%	60%	45%	45%
20	90%	65%	47.50%	45%
50	92%	64%	57%	45%
100	91%	74.50%	50.50%	49%

Not Fair				
Number of matches	RANDOM	GREEDY	MINIMAX	SELF
10	100%	90%	70%	60%
20	95%	65%	35%	35%
50	96%	64%	52%	56%
100	91%	69%	40%	59%

The algorithm performed consistently better than Random Search indicating it would perform better than any random action on the board.

Performance of the algorithm was better throughout when playing against Greedy technique. The proportion of wins dropped compared to the random search.

This algorithm doesn't seem to beat Minimax algorithm. In order to test the performance, I tried tweaking below items to improve it's performance-

- Time for simulations- I started from 100 and tried various iterations. Increasing it up to 150 helped in improving. After this point, the execution time started increasing significantly.
- Weights of choices for tree nodes
- Trial with different number of games in fair and not fair settings

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

This algorithm seem to work in par with Minimax algorithm but better than random and greedy algorithms. Some of the parameters like the ones mentioned above can be further tweaked to improve its performance. I also compared the performance of the algorithm against self with an intuition that the winning percentage should be around 50% in fair settings which eventually is the case. There are certain fluctuations in performance when playing against the same algorithm in unfair settings.

References-

- https://en.wikipedia.org/wiki/Monte_Carlo_tree_search-
- https://github.com/shambakey1/MCTS_adversial_search

-Aditya Mehta

AI Nanodegree