Udacity AIND Project 3
Build an Adversarial Game Playing Agent - Report

20 August 2018

1. Pick an Experiment

For code, I have chosen 2 options:
Option 2: Develop an opening book
Option 3: Build an agent using advanced search techniques

For report, I have chosen only option 3:
I have chosen to use **monte carlo tree search (MCTS).**
The mcts code is written and placed in my_custom_player.py file.

- **Create a performance baseline using run_search.py to evaluate the effectiveness of a baseline agent (e.g., an agent using your minimax or alpha-beta search code from the classroom)**

  I have used the Alpha Beta (minimax with pruning) agent from lecture to be the baseline agent.
  The code is placed in my_custom_player.py file.

- **Use run_search.py to evaluate the effectiveness of your agent using your own custom search techniques**

  Time Limit = 150ms (Default Time)

  Running 100(200) games - fair matches **disabled**
  $ python run_match.py -r 100 -o 'MINIMAX'

Algorithm (Self)	Won VS Minimax (Opponent)
Alpha-Beta (depth=3)	38.0%
MCTS (epoch=40)	56.2%

  Running 100(200x2) games - fair matches **enabled**
  $ python run_match.py -r 100 -o 'MINIMAX' --fair_matches

Algorithm (Self)	Won VS Minimax (Opponent)
Alpha-Beta (depth=3)	35.2%
MCTS (epoch=40)	50.2%

- **You must decide whether to test with or without "fair" matches enabled--justify your choice in your report**

  The Definition of fair matches
  Run 'fair' matches to mitigate differences caused by opening position

  My Intuition
  The MCTS agent should use fair matches enabled.
  In the code, the MCTS is not applied on the opening moves, but only on game state with moves greater than 4. This is due to the vast number of moves possible at the beginning of the game state. All 9 x 11 board squares on the board are available for player 1 on the first move, and one less for player 2.

The opening moves like the centre square of the game board have an obvious advantage of 8 possible next moves, where the corners of the game board have an obvious disadvantage only 2 possible next moves.
There is a time limit of 150 mini-seconds for the self agent to make a move, it is not efficient for MCTS to search this large amount of tree node branches.

So i think using fair matches allow both game agent not reply on any opening advantages which could skew the winning probability.
This is what MCTS needs.

Prove
To prove my intuition is either correct / incorrect, i have run with --fair matches with both enabled + disabled.
When fair matches is enabled, I expected MCTS will perform better.
However, in the results, winning rate is actually lower. So my intuition is incorrect.

Notice
In my code implementation, the opening moves from 1 to 4, I have used an opening book to select a move. (Project code option 2)
The code to generate the opening book is a standalone program - opening_book.py
To run it, best use pypy3 for much faster execution time:

$pypy3 opening_book.py

Hints:

- **If the results are very close, try increasing the number of matches (e.g., >100) to increase your confidence in the results**

The results are not close, therefore no need to run more than 100 games.

- **Experiment with adding more search time--does adding time confer any advantage to your agent?**

Time Limit = 1000ms (Extra Time - Project Requirement)

Running 100(200) games - fair matches **disabled**
$ python run_match.py -r 100 -o 'MINIMAX'

Algorithm (Self)	Won VS Minimax (Opponent)	Compare to result of 150ms
Alpha-Beta (depth=3)	39.0%	1.0%
MCTS (epoch=40)	63.5%	7.3%
MCTS (epoch=80)	65.5%	N/A

Running 100(200x2) games - fair matches **enabled**
$ python run_match.py -r 100 -o 'MINIMAX' --fair_matches

Algorithm (Self)	Won VS Minimax (Opponent)	Compare to result of 150ms
Alpha-Beta (depth=3)	30.8%	-4.4%
MCTS (epoch=40)	55.2%	5.0%
MCTS (epoch=80)	57.0%	N/A

By compare the results between 1000ms to 150ms, there are less than 10% of improvement on winning rate.
In one case (Alpha-Beta (depth=3), it actually decreased the winning rate.
I think purely adding extra time will not confer any advantage to the agent MCTS.

This is because the MCTS code will also need to change to use the extra time efficiently, otherwise the MCTS algorithm will not have a strong improvement on result.

The MCTS agent (that i have implemented) is fine tuned to use epoch = 40 for 150ms, this is to allow maximum of nodes in the game tree, while avoiding timeout (in most of the cases) when running the MCTS search.
If both extra time added, and increase of epochs, then this will definitely increase the winning rate playing against opponent Minimax. (Shown in table above)

But the opponent Minimax agent can also increase the depth, from depth=3 to depth=4 or even more, if there is enough extra time, to take advantage of searching into deeper depths.

The advantage of the search agent is difficult to determine by just adding more time, unless the code is made to use the extra time automatically. At the moment, the Minimax depth and MCTS epoch have to be manually tuned.

- **Augment the code to count the number of nodes your agent searches--does your agent have an advantage compared to the baseline search algorithm you chose?**

  Observations
  After added the code to count the total number of nodes created for each MCTS search, I can see the total number of nodes created by MCTS is very consistent between each search.
  The winning rate of MCTS agent is higher than the Alpha-Beta agent.

  Summary
  For running games playing against opponent Minmax, the MCTS agent has the advantage simply because the winning rate is higher than the baseline Alpha-Beta agent.

  But I failed to see the point of counting the total number of nodes, how is it related to the advantage of the MCTS agent? I don't know.

2. Report Requirements

Your report must include a table or chart with data from an experiment to evaluate the performance of your agent as described above. Use the data from your experiment to answer the relevant questions below. (You may choose one set of questions if your agent incorporates multiple techniques.)

Option 3: Advanced Search Techniques (MCTS)

- **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?**

Using result tables above to calculate difference in performance.

Time Limit = 150ms (Default Time)

Running 100(200) games - fair matches **disabled**

Algorithm (Self)	Won VS Minimax (Opponent)
Alpha-Beta (depth=3)	38.0%
MCTS (epoch=40)	56.2%
Performance Difference	+18.2%

The winning rate of MCTS is 56.2%
The winning rate of Alpha-Beta (baseline) is 38.0%
Calculation: 56.2 – 38.0 = 18.2
The performance difference of my MCTS agent is +18.2%

Running 100(200x2) games - fair matches **enabled**

Algorithm (Self)	Won VS Minimax (Opponent)
Alpha-Beta (depth=3)	35.2%
MCTS (epoch=40)	50.2%
Performance Difference	+15.0%

The winning rate of MCTS is 50.2%
The winning rate of Alpha-Beta (baseline) is 35.2%
Calculation: 50.2 – 35.2 = 15.0
The performance difference of my MCTS agent is +15.0%

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

The MCTS is more effective than the Alpha-Beta (baseline).

According MCTS on Wikipedia, there are a number of advantages MCTS offers.

MCTS on Wikipedia
https://en.wikipedia.org/wiki/Monte_Carlo_tree_search#Advantages_and_disadvantages

1. MTCS do not need an evaluation function, it uses only a simulation with accumulated rewards of a node to see which child node is best and select the action result that child state.

2. No Heuristics / No human expert game play knowledge required. MCTS only needs the game rules / mechanics.

3. The MCTS tree search grows asymmetrically, this concentrate the best child leads to the highest probability of winning. But sometimes this feature could ‘not see’ a move that could lead to lost, which is hard to pick up by random simulations.

4. MCTS search can be stopped anytime and return a best guessed action, however I have not implemented this feature in the code. If the MCTS search timeout, the game is lost to the opponent Minimax.