# Udacity AI Nanodegree

# Project 3 – Adversarial Game Playing Agent

Stephen Blystone

## Introduction

I chose the 3rd option for this project: Build an agent using advanced search techniques.

I wrote code for several search techniques including Negascout, Principal Variation Search (PVS), an iterative version of PVS, PVS with Zero Window Search, and Monte Carlo Tree Search.

I initially tried Negascout and PVS but due to poor performance I continued trying different search techniques before settling on Monte Carlo Tree Search (MCTS).

While I tested playing against a random agent and my own agent (self), for all the test results below I played against the MiniMax Agent.

## Performance Baseline

I wrote an Alpha Beta with Iterative Deepening search agent to use as my baseline against my MCTS agent. I configured a depth limit of 5 for the iterative deepening.

## Fairness

"Fair" matches are defined as repeating every game that is played but the agents switch initiative and use their opponent's opening move. This is intended to balance out the advantage of picking "perfect openings".

Prior to running any games my hypothesis was that running "fair" games would improve the win percentage due to some opening positions possibly being better than others. If the opponent started in a great opening position during a game, then they could have an unfair advantage and result in a loss for me. But by introducing "fairness" then my agent would have a chance to be in the same opening position, which could result in a win for me.

I tried running my agent and the baseline with "fair" and "unfair" games and found varied results. When I had an optimal value of c (0.25 or 0.5) then "fair" provided an advantage. When I had an unoptimal value of c (0, 0.75, 1, 100) then "fair" and "unfair" didn't really make a difference.

## Search Time

I tried increasing the search time (the maximum allowed time for my Agent's get_action() method to respond) and found that the results were generally improved. Some games didn't appear to be impacted by the increased search time.

## MCTS Exploration vs Exploitation

Part of the power of the MCTS algorithm comes from the ability to tune the amount of exploration vs exploitation in the Best Child method. Exploration is when the agent will randomly explore in hopes of finding a good move. Exploitation is using the existing knowledge to make moves with known outcomes.

The equation is "exploitation + c * exploration", where c is the tunable parameter. I tested values of 0, 0.25, 0.5, 0.75, 1, and 100. When c is 0 there should be no exploration, when c is 1 then exploration and exploitation have equal importance, and when c is 100 then the exploration component should dominate over the exploitation.

I found the best values occurred with c = 0.25 and c = 0.5 indicating that a balance of exploration and exploitation is critical for success. If I were to extend this code further, I might explore using a variable parameter that starts off allowing more exploration and reduces the amount later in the game.

# Required Questions

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

| Algorithm | # Rounds | Tunable Parameter for Exploration | Time Limit | Fairness? | % Wins |
|---|---|---|---|---|---|
| MCTS | 100 | 0.5 | 25000 | Fair | 74.80% |
| Alpha Beta Iterative Deepening | 100 | N/A | 25000 | Fair | 66.20% |

*Figure 1 - % Wins MCTS vs Baseline*

My MCTS Agent's best run was with the exploration tunable parameter set to 0.5, timer set to 25000ms, and running "fair" resulting in 74.8% wins. When using the baseline Alpha-Beta with Iterative Deepening with the same parameters it won 66.2%. My agent had an 8.6% higher win percentage over the baseline.

I modified my code to display the number of nodes searched for both the baseline and MCTS agent. The MCTS generally simulated more nodes than the baseline Alpha Beta agent was able to search. But looking at the number of nodes selected and expanded (which equals the number of nodes seen in backpropagation), there are significantly fewer nodes than the baseline. See appendix for counts.

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

I think my MCTS agent was more effective than the baseline because of the exploration/exploitation component of the algorithm. While the baseline Alpha-Beta with Iterative Deepening was limited to a certain depth, the MCTS agent was able to go deeper and simulate all the way to terminal state. As we saw in lecture, predicting a score at a fixed depth can vary as you go deeper one level. By being able to simulate further down the tree MCTS was able to make better move selections.

# Appendix – Full Data Tables

## Comparing % Wins for Various Parameter Settings

| Algorithm | # Rounds | Tunable Parameter for Exploration | Time Limit | Fairness? | % Wins |
|---|---|---|---|---|---|
| MCTS | 100 | 0 | Default | Not Fair | 63.00% |
| MCTS | 100 | 0 | Default | Fair | 62.50% |
| MCTS | 100 | 0 | 25000 | Not Fair | 65.00% |
| MCTS | 100 | 0 | 25000 | Fair | 68.00% |
| MCTS | 100 | 0.25 | Default | Not Fair | 67.00% |
| MCTS | 100 | 0.25 | Default | Fair | 73.20% |
| MCTS | 100 | 0.25 | 25000 | Not Fair | 67.00% |
| MCTS | 100 | 0.25 | 25000 | Fair | 71.80% |
| MCTS | 100 | 0.5 | Default | Not Fair | 68.50% |
| MCTS | 100 | 0.5 | Default | Fair | 67.00% |
| MCTS | 100 | 0.5 | 25000 | Not Fair | 70.50% |
| MCTS | 100 | 0.5 | 25000 | Fair | 74.80% |
| MCTS | 100 | 0.75 | Default | Not Fair | 63.00% |
| MCTS | 100 | 0.75 | Default | Fair | 61.80% |
| MCTS | 100 | 0.75 | 25000 | Not Fair | 61.00% |
| MCTS | 100 | 0.75 | 25000 | Fair | 66.20% |
| MCTS | 100 | 1 | Default | Not Fair | 60.00% |
| MCTS | 100 | 1 | Default | Fair | 63.50% |
| MCTS | 100 | 1 | 25000 | Not Fair | 66.50% |
| MCTS | 100 | 1 | 25000 | Fair | 62.20% |
| MCTS | 100 | 100 | Default | Not Fair | 17.50% |
| MCTS | 100 | 100 | Default | Fair | 14.00% |
| MCTS | 100 | 100 | 25000 | Not Fair | 18.50% |
| MCTS | 100 | 100 | 25000 | Fair | 13.20% |
| Alpha Beta Iterative Deepening | 100 | N/A | Default | Not Fair | Not able to get results; Timeouts occurred and run_match.py stopped playing games. |
| Alpha Beta Iterative Deepening | 100 | N/A | Default | Fair | Not able to get results; Timeouts occurred and run_match.py stopped playing games. |
| Alpha Beta Iterative Deepening | 100 | N/A | 25000 | Not Fair | 66.50% |
| Alpha Beta Iterative Deepening | 100 | N/A | 25000 | Fair | 66.20% |

## Comparing Number Nodes Searched

| Game Move | Monte Carlo Tree Search (MCTS) | | Alpha Beta |
|---|---|---|---|
| | MCTS Selected & Expanded | MCTS Simulated | Nodes Searched |
| 1 | 416 | 4242 | 2232 |
| 2 | 397 | 4541 | 4865 |
| 3 | 531 | 3935 | 2398 |
| 4 | 393 | 3842 | 2935 |
| 5 | 428 | 3463 | 786 |
| 6 | 368 | 3384 | 5310 |
| 7 | 461 | 3043 | 1636 |
| 8 | 409 | 2642 | 785 |
| 9 | 428 | 2377 | 3855 |
| 10 | 416 | 2488 | 3008 |
| 11 | 431 | 2095 | 1221 |
| 12 | 613 | 2183 | 1732 |
| 13 | 573 | 2092 | 2373 |
| 14 | 509 | 2011 | 3713 |
| 15 | 467 | 1969 | 2179 |
| 16 | 528 | 2005 | 1107 |
| 17 | 554 | 1727 | 875 |
| 18 | 434 | 1751 | 1678 |
| 19 | 486 | 1486 | 554 |
| 20 | 541 | 1497 | 559 |
| 21 | 502 | 1482 | 507 |
| 22 | 440 | 1038 | 135 |
| 23 | 612 | 1020 | 211 |
| 24 | 540 | 908 | 456 |
| 25 | 523 | 840 | 396 |
| 26 | 885 | 679 | 275 |
| 27 | 915 | 357 | 197 |
| 28 | 782 | 175 | 263 |
| 29 | 574 | 56 | 161 |
| 30 | 406 | 8 | 110 |
| 31 | | | 226 |
| 32 | | | 112 |
| 33 | | | 61 |
| 34 | | | 24 |