Cole Falxa-Sturken
Connor Lowne

Experiment 3 - Time Constraints

In this experiment, we found that when using various time limits, the vanilla MCTS bot generates a much larger tree than our modified MCTS bot. The data for the one-second constraint is presented below in Fig. 1.

```
1   Vanilla:
2   Mean: 4930.973236009732
3   Median: 3247
4   Min: 2073
5   Max: 75735
6
7   Modified:
8   Mean: 978.4821428571429
9   Median: 280
10  Min: 152
11  Max: 51366
```

Figure 1

The median shows that the difference between the tree sizes was typically about one order of magnitude. With a much larger tree, the vanilla bot defeated the modified bot almost every time, with a record of (W-T-L) 19-0-1. This result is surprising because previously the modified bot was trouncing the vanilla bot with a win rate of 80% over 25 matches. This was tested on a different, more powerful CPU, but that does not seem to account for the differences. It is consistent with the findings of experiment 2

Experiment 4 - Parallelization

After reading about MCTS parallelization, it appeared that the most effective methods were tree parallelization and root parallelization. In tree parallelization, you use one shared tree that all threads access via a global mutex or local mutexes. In the root method, you instead use separate trees and run the entire algorithm in parallel. After the threads have completed their search, they return their root nodes and then the visit and win counts are aggregated into one tree which is used to select the best action.

I chose to split the node limit based on the number of threads used, to keep comparisons more balanced. However, when using the entire node limit per thread it

outperformed the vanilla and modified bots. I only made a parallelized version of the vanilla bot. The data showing the differences in wallclock time when using 1000 nodes and 5000 nodes are shown below in Fig. 2 and Fig. 3.

```
Player 1 (mcts_vanilla) stats:
Mean turn time: 0.2906028990242836
Median turn time: 0.3065108999981021
Min turn time: 0.012002600000414532
Max turn time: 0.4536521999980323

Player 2 (mcts_parallel) stats:
Mean turn time: 0.7415770965001866
Median turn time: 0.7426956999988761
Min turn time: 0.6346593000016583
Max turn time: 0.9159368000000541
```

Figure 2

```
Player 1 (mcts_vanilla) stats:
Mean turn time: 1.4655814035716048
Median turn time: 1.5049211500008823
Min turn time: 0.08511639999778708
Max turn time: 2.355503500002669

Player 2 (mcts_parallel) stats:
Mean turn time: 1.2513029037382974
Median turn time: 1.2762101000007533
Min turn time: 0.8975695999979507
Max turn time: 1.5934266000003845
```

Figure 3

I utilized Python's multiprocessing module instead of utilizing threads directly. I could likely improve the overhead and reduce turn time if I were to handle the threads manually. This was run using 8 threads on my 8-core Ryzen 7 5800x3D. The CPU has 16 logical processors so it could be run with 16 threads, but my preliminary testing showed that the overhead was not worth it unless using far more nodes or a longer search time.