Alpha-beta search baseline performance data

Opponent	*Matches*	*Search Time*	*Win percentage*
Greedy	100	150 ms	55.2%
Minimax	100	150 ms	35.8%
Random	100	150 ms	80.2%

Monte Carlo Tree Search with UCT performance data

Opponent	*Matches*	*Search Time*	*Win percentage*
Greedy	100	150 ms	87.5%
Minimax	100	150 ms	74.0%
Random	100	150 ms	97.0%

The data was not tested using fair matches, as I didn't test on opening positions or with heuristics and the performance with both fair matches and without were practically the same. As you can see in the data, the Monte Carlo UCT search does a lot better compared to the alpha-beta search with the search time being at 150 milliseconds.

Alpha-beta search for 500 milliseconds search time

Opponent	*Matches*	*Search Time*	*Win percentage*
Greedy	100	500 ms	55.5%
Minimax	100	500 ms	35%
Random	100	500 ms	80%

Monte Carlo Tree Search with UTC for 500 milliseconds search time

Opponent	*Matches*	*Search Time*	*Win percentage*
Greedy	100	500 ms	89.5%
Minimax	100	500 ms	82.5%
Random	100	500 ms	98.5%

For my agent, adding more search time gives it a huge advantage over the other agents because the search doesn't have an iteration limit, moreover it achieves more iterations as the time limit is set higher because the method iterates over the given time and not an iteration limit.

Number of Nodes for Monte Carlo Tree Search and Alpha Beta Search

Matches	*Number of Nodes (MCTS)*	*Number of Nodes (Alpha-Beta)*
1	93621	54007
2	64901	95279
3	88986	84185
4	83855	96653
5	26049	83687
6	84658	93494
7	99904	79212
8	93143	63570
9	86713	94834
10	84896	84955

This table shows that the number of nodes expanded for the Monte Carlo algorithm and the Alpha Beta algorithm are largely the same. The Monte Carlo algorithm had 80672.6 nodes expanded and the Alpha-Beta algorithm had 82987.6 nodes expanded on average.

Questions

1.

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

On average for the 150 milliseconds search time, the Alpha-Beta algorithm has a 57.07% win rate against all opponents and the Monte Carlo UCT algorithm has an 85.07% win rate against all opponents. This means that there is a 28% increase in performance when using the Monte Carlo algorithm. There is also a 38.2% increase in performance when playing against the minimax algorithm. Additionally if optimizations were to be made into the Monte Carlo algorithm it would easily hit a 100% win percentage against the minimax algorithm, as it would have more iterations in the given search time and so it would be more precise.

For 500 milliseconds search time games, the Alpha-Beta algorithm still has an average win rate of 57% however the Monte Carlo algorithm shows an average win rate of 90.2% and now has a performance gain of 33.3% when using the Monte Carlo algorithm. The win percentage against the minimax agent also went up by 8.5% while the alpha-beta search algorithm stayed the same with only negligible changes in the win percentages.

2.

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

I think that the technique I chose was more effective because not only was the performance of the algorithm better as seen in the data above, but it is also a lot faster and efficient. We can see this by running the ‘unittest’ on both algorithms, where the Alpha-Beta algorithm takes 33 seconds to complete it and the Monte Carlo algorithm takes 21 second. The algorithm can also return a good solution even if it is interrupted before it ends, which makes it perfect for a problem like this where it only has 150 milliseconds to find an answer.