Build an agent using advanced search techniques (for example: killer heuristic, principle variation search (not in lecture), or monte carlo tree search (not in lecture))

- 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)
- The effectiveness of alpha-beta search is as follows (from 10 observations of different play length):
  - Minimax: 70.0% - 90%
  - Random: 90% - 100%
  - Greedy: 80.0% - 100%

- Use `run_search.py` to evaluate the effectiveness of your agent using your own custom search techniques
- The effectiveness of Monte Carlo Tree Search is as follows (from 10 observations of different play length):
  - Minimax: 65.0% - 85%
  - Random: : 90% - 100%
  - Greedy: 85.0% - 100%

- You must decide whether to test with or without "fair" matches enabled--justify your choice in your report
- I chose to test with “fair” match enabled as in many cases the victory depends upon choosing the first winning move. Fair matches option balances that advantage and give more realistic results.

1. How much performance difference does your agent show compared to the baseline?
   The algorithm performs nearly as well as the baseline with fair match enabled. The algorithm tends to outperform the baseline with fair match disabled.

2. Why do you think the technique you chose was more (or less) effective than the baseline?
   The Monte Carlo Tree Search tends to look further in the solution space when searching for solution which helps it to make better decision. The discount factor value ‘c’ (chosen at 1/sqrt(2) in this project) governs how far particular branch will be considered for taking decision.