Heuristic Analysis

Chyld Medford

Artificial Intelligence Nanodegree/May Cohort

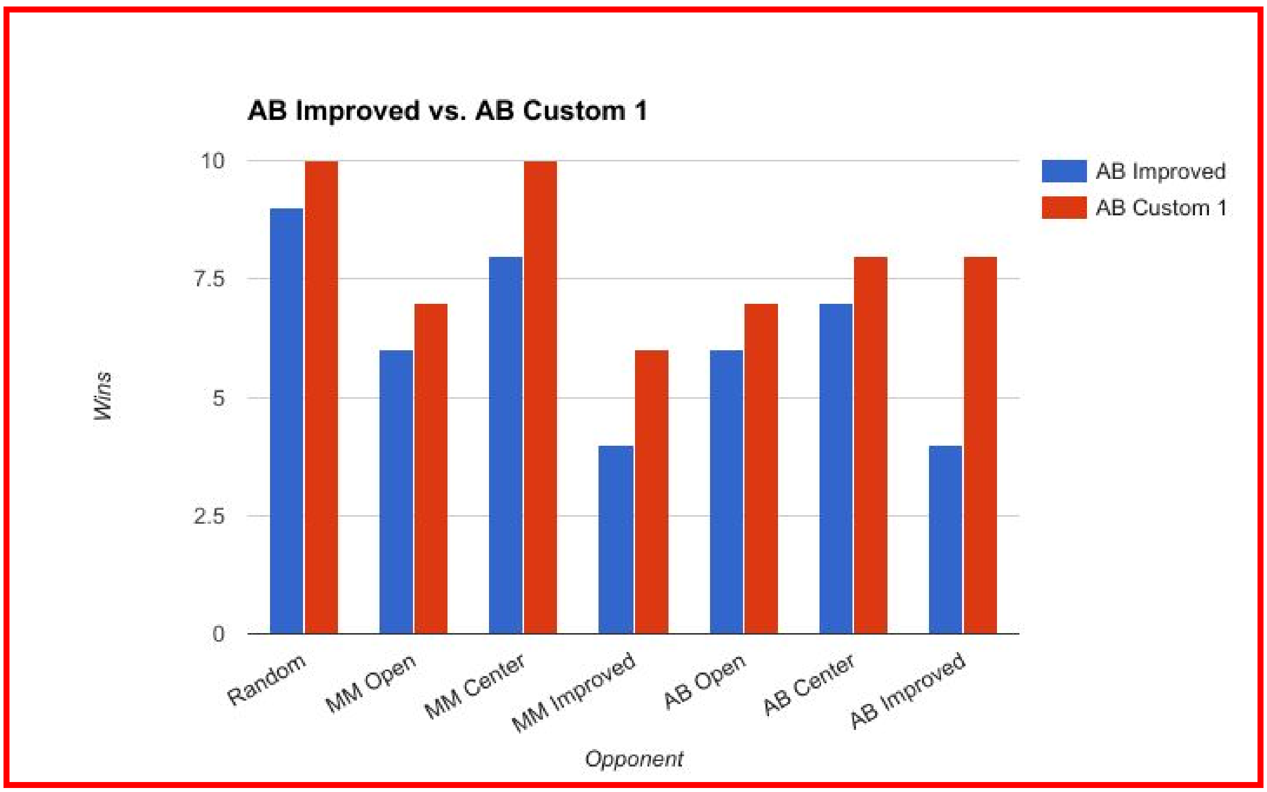
Heuristic 1: Random

For this heuristic, I simply chose a random value. I wanted to see, mainly as an experiment, how well it would perform against other agents. Turns out, that it has been outperforming all other agents that it has competed against.

```python
if game.is_loser(player):
    return float("-inf")

if game.is_winner(player):
    return float("inf")

return random.random()
```



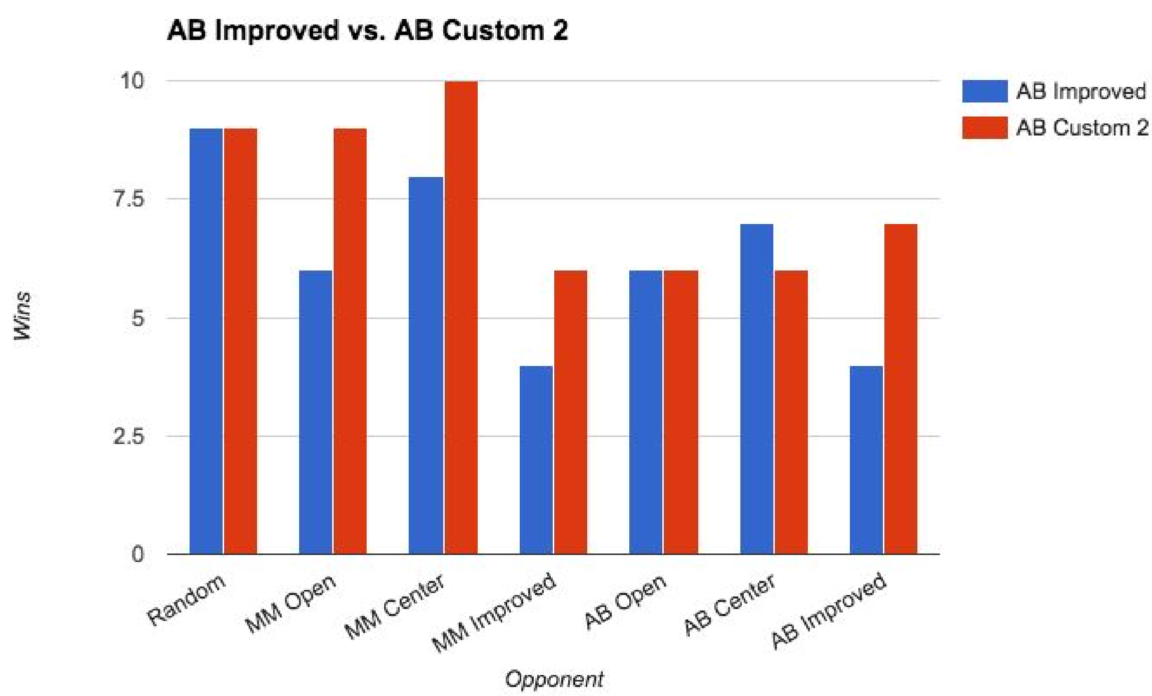
Heuristic 2: Ratio

For this algorithm, I wanted the score to be based on a ratio of “my moves” over “total moves”. It would seem reasonable, that higher scores would lead to better overall results. So far, the testing shows this heuristic is promising - as it is beating the “Improved” model fairly consistently.

```python
if game.is_loser(player):
    return float("-inf")

if game.is_winner(player):
    return float("inf")

own_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
return float(own_moves / (own_moves + opp_moves))
```



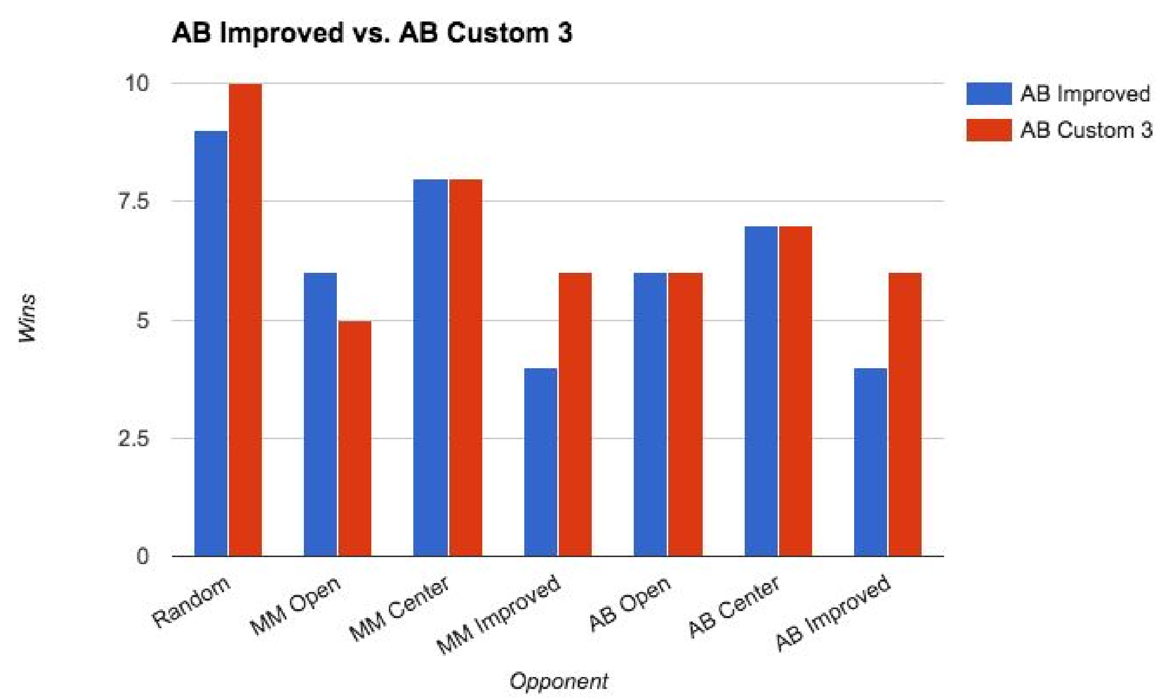
Heuristic 3: Distance

For my final heuristic, I wanted to know how the distance between the two opposing players would contribute or correlate to a win or loss. My thought was the further away, the more "mobile" a player could be. However, my results show that it essentially ties with the "Improved" model.

```python
if game.is_loser(player):
    return float("-inf")

if game.is_winner(player):
    return float("inf")

y1, x1 = game.get_player_location(player)
y2, x2 = game.get_player_location(game.get_opponent(player))
d = (((y2 - y1) ** 2) + ((x2 - x1) ** 2)) ** 0.5
return float(d)
```



Performance Benchmark

Below are the benchmarks for my three heuristics. The random comes out first, followed by the ratio and finally the distance metric is third. However, it's important to note that all three heuristics are a considerable improvement over the "Improved" model. Overall, I'm happy with these initial results. In the future, I'd like to think about combining some type of ensemble (aggregate) model that could use the best characteristics of each.

```
This script evaluates the performance of the custom_score evaluation
function against a baseline agent using alpha-beta search and iterative
deepening (ID) called `AB_Improved`. The three `AB_Custom` agents use
ID and alpha-beta search with the custom_score functions defined in
game_agent.py.

                        *************************
                             Playing Matches
                        *************************

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random       9  |   1    10  |   0     9  |   1    10  |   0
    2       MM_Open      6  |   4     7  |   3     9  |   1     5  |   5
    3      MM_Center     8  |   2    10  |   0    10  |   0     8  |   2
    4     MM_Improved    4  |   6     6  |   4     6  |   4     6  |   4
    5       AB_Open      6  |   4     7  |   3     6  |   4     6  |   4
    6      AB_Center     7  |   3     8  |   2     6  |   4     7  |   3
    7     AB_Improved    4  |   6     8  |   2     7  |   3     6  |   4
--------------------------------------------------------------------------
           Win Rate:      62.9%        80.0%        75.7%        68.6%
```

Heuristic Recommendation

After creating and then running my evaluation functions, I have arrived at "Custom 1" also know as my "Random" function. I choose it for a number of reasons.

1. It's easy to understand. It just returns a random value as the evaluation function.
2. It's quick to execute. The quicker the evaluation function is, the deeper the search can go. This definitely helps out near the end of the game when the search function can see the bottom of the tree, and therefore direct the player to the optimal move.
3. Although "randomness" could be bad, I think near the beginning of the game when the number of branches is so high and you can't get to the bottom, a complicated evaluation function is probably just as useful, in the end, as a random move.
4. Finally, if you look at the data, and the chart above, you will see that the "Random" evaluation function does way better than "AB Improved", beating it 8 out of 10 times. And it's consistently gets the highest score of all three evaluation functions.