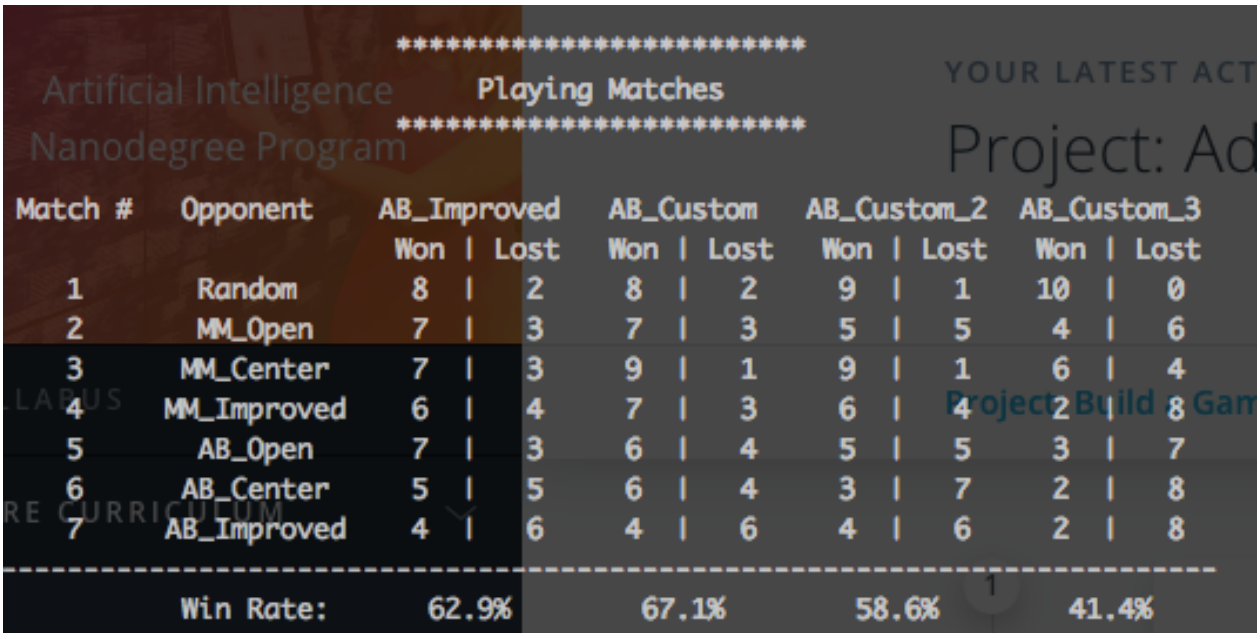
Heuristic Analysis

By: Davis Martin

Goal

To implement a game playing agent for the game Isolation where players move like knights in chess. To do this I developed three heuristics to use in conjunction with the Minimax Algorithm with Alpha-Beta Pruning to compare against the Improved heuristic included in the repo.

Results



```
**************************
Playing Matches
**************************

Match #   Opponent        AB_Improved      AB_Custom      AB_Custom_2    AB_Custom_3
                          Won | Lost       Won | Lost     Won | Lost     Won | Lost
    1     Random            8 |    2         8 |    2       9 |    1      10 |    0
    2     MM_Open           7 |    3         7 |    3       5 |    5       4 |    6
    3     MM_Center         7 |    3         9 |    1       9 |    1       6 |    4
    4     MM_Improved       6 |    4         7 |    3       6 |    4       2 |    8
    5     AB_Open           7 |    3         6 |    4       5 |    5       3 |    7
    6     AB_Center         5 |    5         6 |    4       3 |    7       2 |    8
    7     AB_Improved       4 |    6         4 |    6       4 |    6       2 |    8
--------------------------------------------------------------------------------
          Win Rate:      62.9%            67.1%          58.6%          41.4%
```

My Heuristics

1. **AB_Custom** : This was my most successful heuristic normally winning an average of approx. 65-70% of the games in the tournament. The main thinking behind this heuristic is that while the most important thing is the different between the num of moves the players still have one should take into consideration how close to the middle their opponent is since one can normally move more freely from there. To compensate for this I subtracted the opponents center position from each player respectively
2. **AB_Custom_2**: For this heuristic I tried to take into account the number of moves left. When the game first starts out I multiply the opponents moves by two to incentives staying close to them. As the space dwindle the calculation changes so that it just subtracts the difference in remaining moves. Also I added in a factor that takes into account the distance between the players. So that the closer they are the higher the score is.
3. **AB_Custom_3**: For this heuristic it is again the difference in remaining moves but I weight the opponents moves more because it should cause a more aggressive style of play.

Results

I ran `tournament.py` to test the Heuristics described above as well as the benchmark `ID_Improved` Heuristic to see how it performed against several types of player strategies in `sample\_players.py`. There was a question whether tweaking the Improved Heuristic just a little bit with a weight so that it played close to the opponent would improve the score a lot. The data illustrated that this was not the case and **AB_Custom_3** consistently had a Win Rate less than the benchmark.

I had high hopes for **AB_Custom_2** as well as intuitively I thought playing more aggressively as the game progresses would be better. However as seen in the image above this turned out not to be the case and the Win Rate was ~4% lower than the benchmark. Though I think with some more tweaking and testing this evaluation function could be improved.

Recommended Heuristic

After evaluating all 3 Heuristics I would recommend that one use the **AB_Custom** evaluation function when playing Isolation. There are three main reasons I would go with this evaluation function.

1. It is not overly complex it depends only on the current state of the game. It isn't resource intensive in that you don't have to keep track of previous moves or store them in some way.
2. By comparing to the Improved benchmark we test against. The `ID_Improved` benchmark is already a pretty high performing, and by taking into account that the center is a more powerful position we improve on this benchmark.
3. Finally, this evaluation function consistently had a Win Rate between 65-70% consistency higher than all the other evaluation functions I tested including the benchmark `ID_Improved` by somewhere between 5-10%.