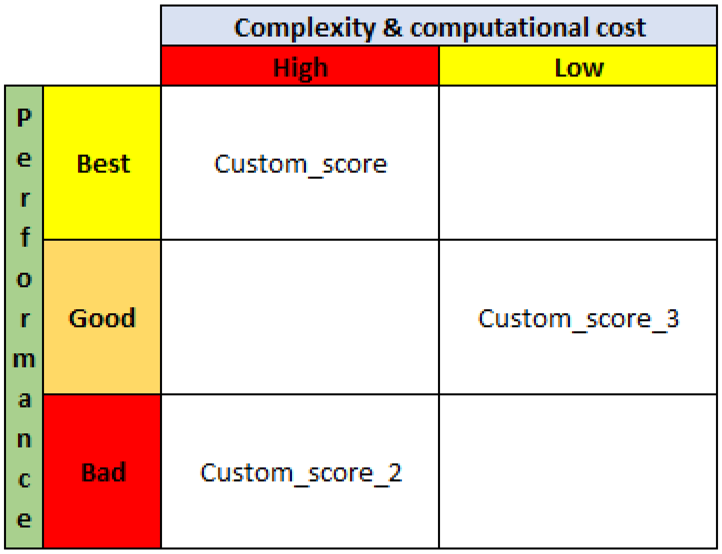
Heuristic Analysis of AIND Isolation project

Author: Naren Karthik

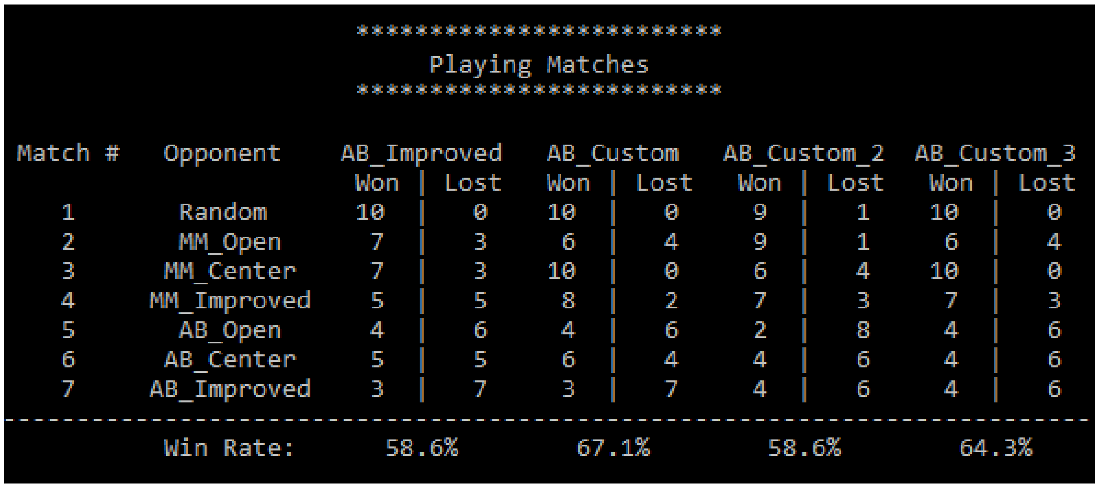
Based on the instructions of the project I had implemented 3 custom evaluation functions and the results of the tournament are as shown below.

Complexity, computational cost & performance matrix of evaluation functions



		Complexity & computational cost	
		High	Low
Performance	Best	Custom_score	
	Good		Custom_score_3
	Bad	Custom_score_2	

Tournament 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       10  |   0     10  |   0      9  |   1     10  |   0
    2         MM_Open       7  |   3      6  |   4      9  |   1      6  |   4
    3        MM_Center      7  |   3     10  |   0      6  |   4     10  |   0
    4       MM_Improved     5  |   5      8  |   2      7  |   3      7  |   3
    5         AB_Open       4  |   6      4  |   6      2  |   8      4  |   6
    6        AB_Center      5  |   5      6  |   4      4  |   6      4  |   6
    7       AB_Improved     3  |   7      3  |   7      4  |   6      4  |   6
--------------------------------------------------------------------------------
             Win Rate:      58.6%         67.1%         58.6%         64.3%
```

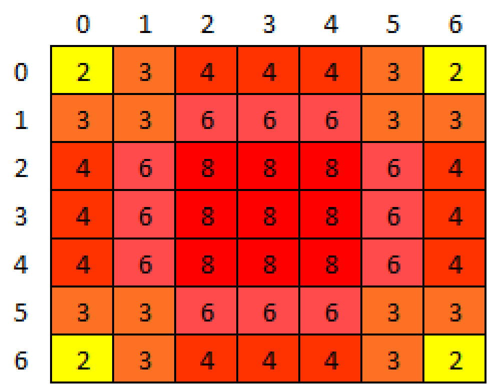
Custom_score:

A function which evaluates the both the players position from the center of the board and then captures the difference between the score of game agent built by us against the score opponent player. Thus rewarding our player higher for near center of board positions.

Analysis:

As shown above the "Custom_score" function performs better than all the other functions and I would recommend this function for use. This function is performing better than the rest because this rewards the player based on the player's current position to the center of the board. This game being a perfect information game where all the outcomes based on player's action can be predetermined with 100% accuracy. We can construct a heat map of the board with each position populated with the number of possible legal moves from that position for a player as shown below.

Heat map of the number of moves possible from a give position



	0	1	2	3	4	5	6
0	2	3	4	4	4	3	2
1	3	3	6	6	6	3	3
2	4	6	8	8	8	6	4
3	4	6	8	8	8	6	4
4	4	6	8	8	8	6	4
5	3	3	6	6	6	3	3
6	2	3	4	4	4	3	2

We can see from the heat map that being in or around the center gives the player more moves to choose from. So even as the game progresses and certain boxes become unavailable still the boxes with the most possible moves may have more moves left than the other boxes, this is the reason "Custom_score" which is based on the distance from the center provides us the best results of all functions even though this is the most computationally expensive function of all 3 functions.

Custom_score_2:

A function which evaluates which calculates the difference in number of moves available for our player and moves available for opponent player. Based on the difference it score higher for larges differences and then in decremented steps for lower difference.

Analysis:

This function performs the worst of all 3 functions because even though we are rewarding the player for having more legal moves than the opponent based on the difference the problem is this is very much dependent on the constants used to differentiate these difference levels which is highlighted in yellow below.

```
if((len(my_moves)-len(opp_moves)) > 3):
    score += 10*(len(my_moves)-len(opp_moves))
elif(((len(my_moves)-len(opp_moves)) > 0) and ((len(my_moves)-len(opp_moves)) < 3)):
    score += 6*(len(my_moves)-len(opp_moves))
else:
    score += 2*(len(my_moves)-len(opp_moves))
```

Adjusting these constant through trial and error can impact the results a greatly but that would be equivalent of a human finding the best weights of a neural network to find the best possible weights to land on the maxima by manually adjusting and readjusting the weights in the network. So this function due to the use of arbitrary constants is not an efficient function to use.

Lesson learnt: Avoid using arbitrary values in your evaluation functions, rather use the values derived from the game itself in a smart way.

Custom_score_3:

A standard function which calculates the difference in number of moves available for our player and moves available for opponent player.

Analysis:

This is a standard baseline function I used to compare against the other 2 functions to see if they perform better or worse that this function. This gives the difference in legal moves available for our player and the opponent. Based on the above results we can see this performed better than a more complex Custom_score_2 function. This is because an evaluation function being complex doesn’t mean it will provide the best estimation of the depth and end state for available moves rather it may as well be due to the complexity that Custom_score_2 was not able to evaluate more nodes of the game than Custom_score_3 function in the same timeout duration. That may be the reason for Custom_score_3 function to provide better results than Custom_score_2.