Ruihan Zou

Custom_Score:

The heuristic1 is trying to restrict the opponent's moves and maximizing the player's moves:

Supposing we have
my_moves ratio = len(my next possible moves) / len(opponent next possible moves),
opponent_moves ratio = len(opponent next possible moves) / len(my next possible moves)
separately.

Then in order to make my_moves ratio - opponent_moves ratio > 0, we need to multiply
len(opponent next possible moves) * len(opponent next possible moves)

Then it can be derived to :
score = len(my next possible moves) * len(my next possible moves) - a * (opponent next possible moves) * (opponent next possible moves) where a = 1.8

Custom_Score2:

The heuristic2 is trying to remove those corner moves from my moves which means the corner moves would get negative scores. The factor a would be 4 when the spaces are less then 25%:

score = (len(my moves) - a * len(my moves in corner)) - len(opponent moves) where a = 4

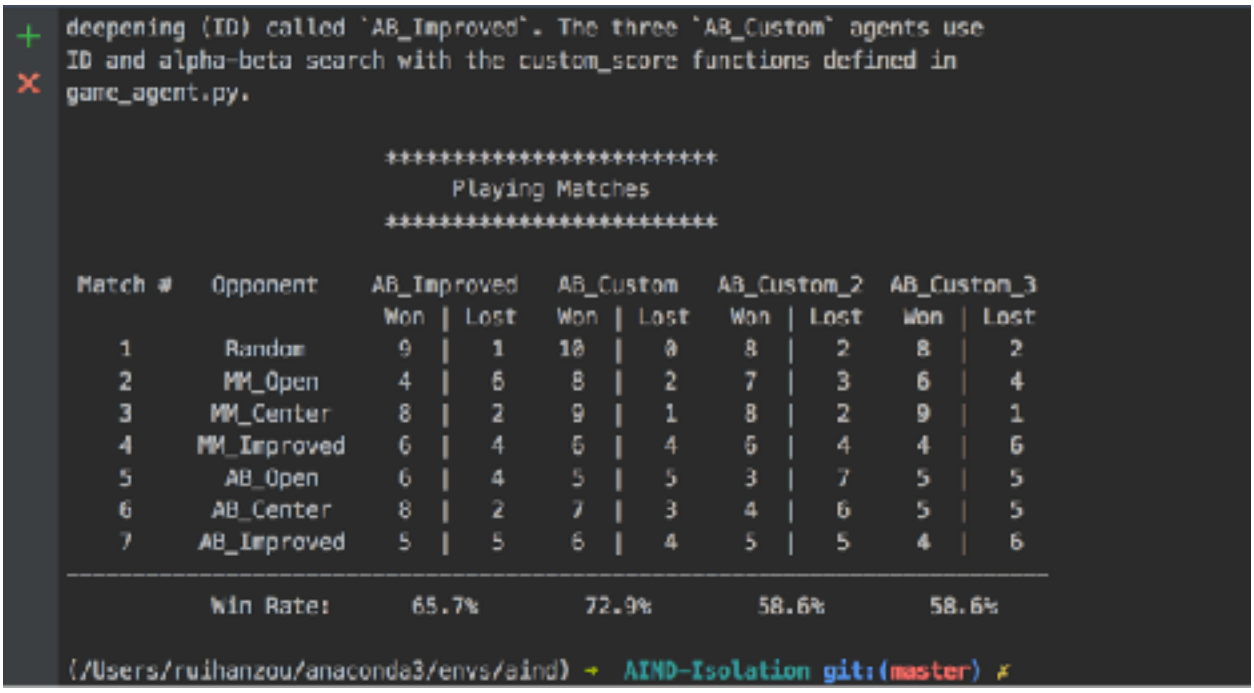
Custom_Score3:

The heuristic3 is trying to maximize player's moves:

score = a *(len(my moves) - len(opponent moves) where a = 1.8

The above factors are modified to get the best results according some experiences

Below is the test result after executing tournament.py.



```
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     8  |  2     8  |  2
    2       MM_Open     4  |  6      8  |  2     7  |  3     6  |  4
    3      MM_Center    8  |  2      9  |  1     8  |  2     9  |  1
    4     MM_Improved   6  |  4      6  |  4     6  |  4     4  |  6
    5       AB_Open     6  |  4      5  |  5     3  |  7     5  |  5
    6      AB_Center    8  |  2      7  |  3     4  |  6     5  |  5
    7     AB_Improved   5  |  5      6  |  4     5  |  5     4  |  6
--------------------------------------------------------------------
          Win Rate:      65.7%        72.9%        58.6%        58.6%

(/Users/ruihanzou/anaconda3/envs/aind) → AIND-Isolation git:(master) ✗
```

AB_Custom uses heuristic1 as evaluate function, AB_Custom2 uses heuristic2 as evaluate function and AB_Custom_3 use heuristic2 as evaluate function

We can see that heuristic1 has the best results with factor a = 1.8, the worse performance match is the game played with AB_Open which is a 50/50 win and lost.

Here is the table to show the lost times while playing against to opponent agents

win rate	AB_Improved	AB_Custom Lost	AB_Custom_2 Lost	AB_Custom_3 Lost
	65.7%	72.9%	58.6%	58.6%

Heuristic 1 has the best performance on losing the least games. It proves that our strategy works. So simply removing the corner moves in heuristic 2 and maximizing the players move are not quite efficiency.

From the experience result, we can see heuristic 1 has the better result. I think it is because It not only restricts the opponent's moves but also tries to maximize the player moves. It is like a combination strategy but not a single one.
That's why we have 72.9% win rate which is higher than other heuristics being considered. Also, heuristic 1 is really easy to be understood and explained to people though it is not a straightforward idea at the beginning. The code is easy to implement and following the lecture which is giving higher score if the player has better options.