Heuristic Analysis
AIND

M.Shokry

Custom Score - 2*My_moves - Opp_moves + abs(my center)/4

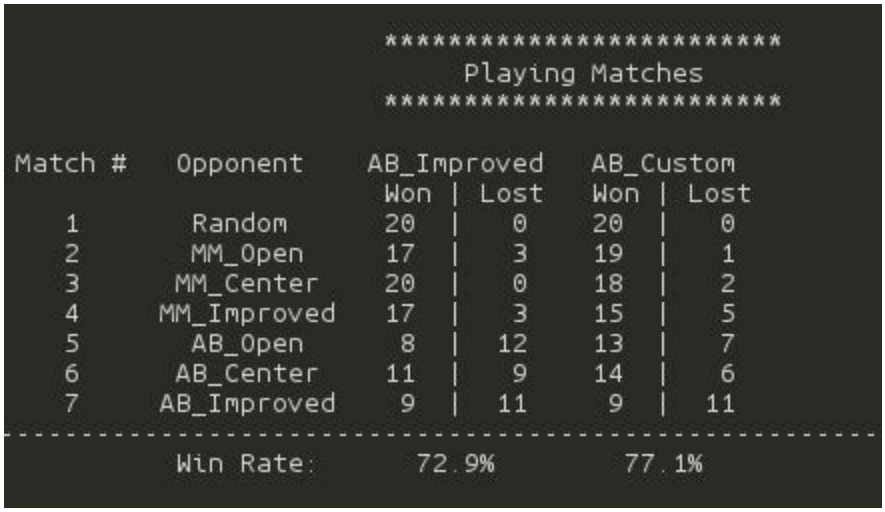
The idea behind this scoring is to rank the move that gives my player a more moves and leads to decrease my opponent's moves without pushing my player to the corner, As the corner leads to lose a lot of advantage and leads to final loss, It's a weighted defending attacking strategy.

Implementation

```
# Evaluating the the move will reduce the Opp player and twords the center
if game.is_loser(player):
    return float("-inf")

if game.is_winner(player):
    return float("inf")
My_moves = float(len(game.get_legal_moves(player)))
Opp_moves = float(len(game.get_legal_moves(game.get_opponent(player))))
diff = float(My_moves - Opp_moves)
w, h = game.width / 2., game.height / 2.
y, x = game.get_player_location(player)
center = float((h - y)**2 + (w - x)**2)
return float(2*My_moves - Opp_moves + abs(center)/4)
```

Results



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

Match #   Opponent    AB_Improved   AB_Custom
                       Won | Lost   Won | Lost
   1       Random       20  |   0    20  |   0
   2       MM_Open      17  |   3    19  |   1
   3      MM_Center     20  |   0    18  |   2
   4     MM_Improved    17  |   3    15  |   5
   5       AB_Open       8  |  12    13  |   7
   6      AB_Center     11  |   9    14  |   6
   7     AB_Improved     9  |  11     9  |  11
----------------------------------------------------------
          Win Rate:      72.9%        77.1%
```

This Custom score gives a better results than the Improved score,

- What to consider next to improve the results :
    - Testing more weights to get the optimal values.
    - Maybe subtracting the opponent's moves weight from the center will give higher results.

Custom Score 2 - (Total movies available + Next available moves)/Blank*100

The Idea Behind this scoring it to increase the score of the move that increases the dominating factor for my player - the move that gives the player more moves to be in relative to the free points to move to.

Implementation

```
# The Move leads to the next biggest moves
if game.is_loser(player):
    return float("-inf")

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

My_moves = np.array(game.get_legal_moves())
Blank = np.array(game.get_blank_spaces())
next_moves = My_moves
for move in My_moves:
    next_moves = np.vstack((next_moves, np.array(get_moves(move))))
Total = len(np.array([x for x in set(tuple(x) for x in next_moves) & set(tuple(x) for x in Blank)]))
return (float(Total)/len(Blank)*100)
```

Results

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

Match #   Opponent    AB_Improved  AB_Custom_2
                       Won | Lost   Won | Lost
    1       Random      19  |   1    20  |   0
    2       MM_Open     17  |   3    13  |   7
    3      MM_Center    19  |   1    19  |   1
    4     MM_Improved   14  |   6    12  |   8
    5       AB_Open     11  |   9     7  |  13
    6      AB_Center    13  |   7     7  |  13
    7     AB_Improved   11  |   9     9  |  11
-----------------------------------------------------
          Win Rate:      74.3%        62.1%
```

It's intensive computing score to be used, I think this is the worst thing to do in scoring as I lose the power of discovering a better solutions and waste the power to calculate the score, and I think the score will be worse if I have a lower computational power.

- What to consider to improve the results :

- May be subtracting the opponent's moves in the current moves and the next move will give higher results - but It will increase the time for score computing.

Custom Score 3 - My_moves - Opp_moves || My_moves - 3 * Opp_moves

The Idea Behind this scoring it to branch my score for the board with less than 10 moves consider the score where i can increase my moves and decrease opp. Moves 1:1 for higher than 10 search for the moves increases my moves and decrease opp. Moves with the ratio of 1:3.

Implementation

```
# Branching values
if game.is_loser(player):
    return float("-inf")

if game.is_winner(player):
    return float("inf")
My_moves = float(len(game.get_legal_moves(player)))
Opp_moves = float(len(game.get_legal_moves(game.get_opponent(player))))
Blank = np.array(game.get_blank_spaces())
if len(Blank) < 10 :
    return float(My_moves - Opp_moves)
else:
    return float(My_moves - 3 * Opp_moves)
```

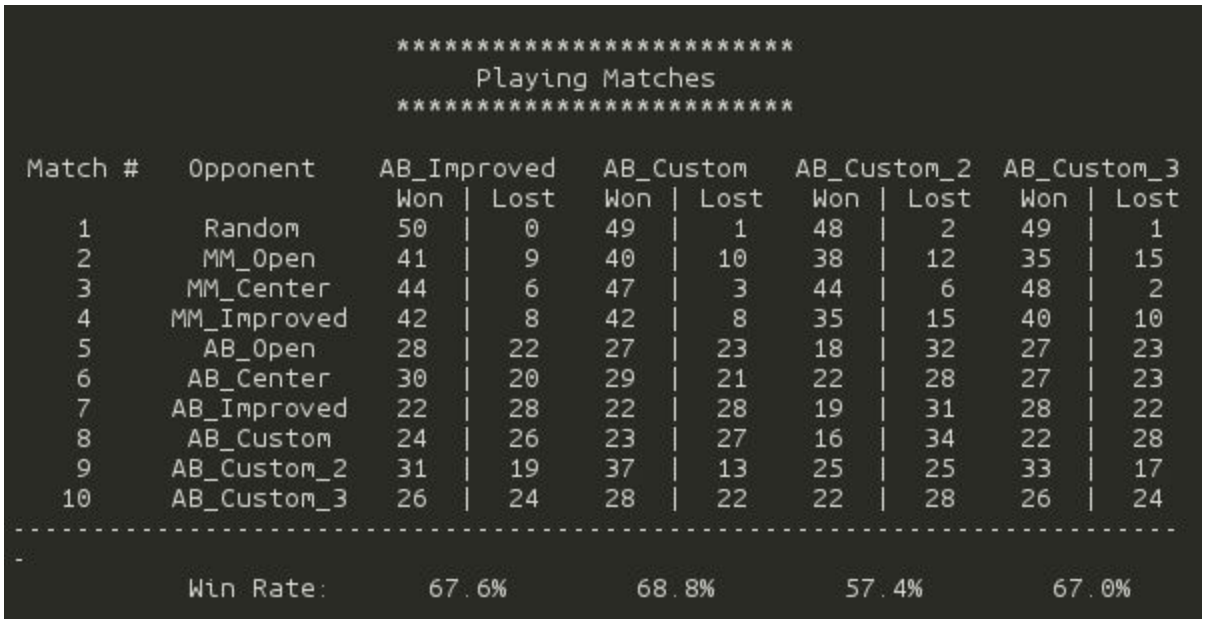
Results

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

 Match #    Opponent    AB_Improved  AB_Custom_3
                         Won | Lost   Won | Lost
    1         Random      20  |   0    20  |   0
    2         MM_Open     17  |   3    17  |   3
    3        MM_Center    19  |   1    20  |   0
    4       MM_Improved   16  |   4    14  |   6
    5         AB_Open     11  |   9    12  |   8
    6        AB_Center    12  |   8    12  |   8
    7       AB_Improved   11  |   9    12  |   8
-----------------------------------------------------
            Win Rate:      75.7%        76.4%
```

- What to do to improve the results
    - Testing more weights to get the optimal values.

Conclusion



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

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random       50  |   0    49  |   1    48  |   2    49  |   1
    2       MM_Open      41  |   9    40  |  10    38  |  12    35  |  15
    3      MM_Center     44  |   6    47  |   3    44  |   6    48  |   2
    4     MM_Improved    42  |   8    42  |   8    35  |  15    40  |  10
    5       AB_Open      28  |  22    27  |  23    18  |  32    27  |  23
    6      AB_Center     30  |  20    29  |  21    22  |  28    27  |  23
    7     AB_Improved    22  |  28    22  |  28    19  |  31    28  |  22
    8      AB_Custom     24  |  26    23  |  27    16  |  34    22  |  28
    9     AB_Custom_2    31  |  19    37  |  13    25  |  25    33  |  17
   10     AB_Custom_3    26  |  24    28  |  22    22  |  28    26  |  24
--------------------------------------------------------------------------
-
           Win Rate:      67.6%        68.8%        57.4%        67.0%
```

All the implemented playing with each other result..

As a result I recommend to use the first **heuristic function** as it performs better than other functions with different techniques.