Build a Game-Playing Agent

Heuristic Analysis

custom_score_3

Based on **improved_score()**, I add **center_score()** because it is important to stay close to center. This method will increase the win rate.

```
def custom_score_3(game, player):
    ''' improved score + central score'''
    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 - opp_moves)
            - central_distance(game, game.get_player_location(player))
```

custom_score_2

The **custom_score2()** modifies **AB_Improved()**. Doubling the opponent's move means we focus more on the count of opponent's moves.

```
def custom_score_2(game, player):
    ''' improved score + common moves'''
    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 - 2 * opp_moves)
```

custom_score

I create a function **attack_moves()** because I want to occupy the location near the center, try to let the opponent stay away from center.

```
def attack_moves(game, player):
    ''' try to let the opponent stay away from central position '''
    cmoves = common_moves(game, player)
    if not cmoves:
        return 0
    return max(central_distance(game, m) for m in cmoves)
```

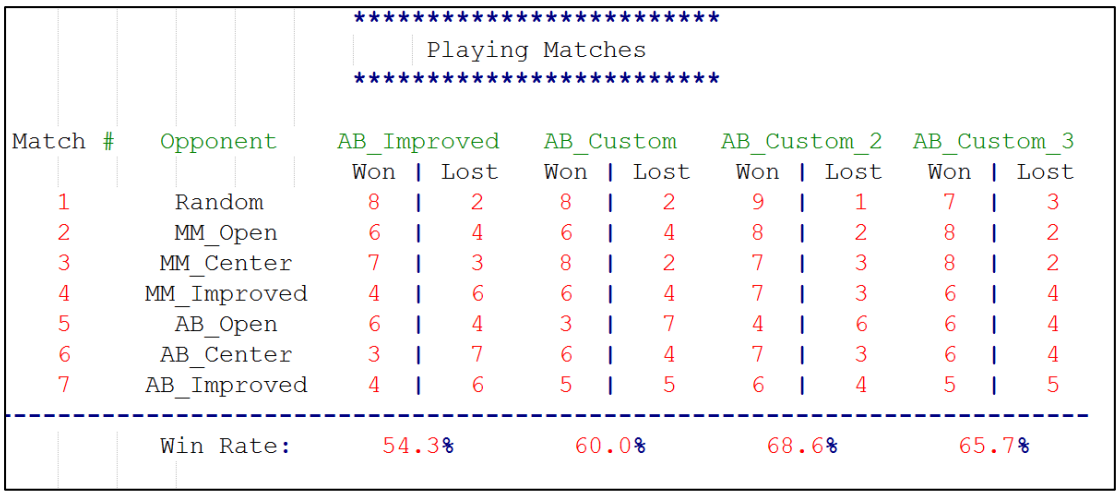
The **custom_score()** combines with **improved_score()** and **attack_moves()** as below.

```
def custom_score(game, player):
    ''' improved socre  + attack moves'''
    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 - opp_moves) - attack_moves(game, player)
```

Performance



```
*************************
    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       7  |  3
   2       MM_Open       6  |  4      6  |  4      8  |  2       8  |  2
   3      MM_Center      7  |  3      8  |  2      7  |  3       8  |  2
   4    MM_Improved      4  |  6      6  |  4      7  |  3       6  |  4
   5       AB_Open       6  |  4      3  |  7      4  |  6       6  |  4
   6      AB_Center      3  |  7      6  |  4      7  |  3       6  |  4
   7    AB_Improved      4  |  6      5  |  5      6  |  4       5  |  5
----------------------------------------------------------------------
          Win Rate:      54.3%        60.0%        68.6%        65.7%
```

AB_Custom: Improved_score() + attack_moves()

AB_Custom2: Double opponent's moves in AB_improved()

AB_Custom3: Improved_score() + center_score()

In AB_custom_2, I focus more on opponent's move. Therefore, the double weight was given. As the result, AB_custom_2 performs well in most of time. Also, AB_custom_3 performs well because it combines AB_improved() and center score calculation. Finding good position is really important to win the competition. AB_custom_3 is more stable in my result. Therefore, it is the best score function in my project.