# Heuristic analysis

Runkun Miao

To have a good evaluation for isolation, it is essential to know how do we win the game when we play it. When I play the game, I like to corner my opponent or to take out my opponent's potential move. As a result, the heuristic function has been designed in this fashion.

First of all, there are two functions that needs to be addressed, centrality and common_moves. Centrality function is to calculate the distance of opponent's move to the centre of the board. Therefore, if the value is big, we can be sure that we are cornering our opponent. Common_moves function is to check how many moves we have in common with our opponent, which help us to chase our opponents.

```
def custom_score(game, player):
    """Calculate the heuristic value of a game state from the point of view
    of the given player.
    This should be the best heuristic function for your project submission.
    Note: this function should be called from within a Player instance as
    `self.score()` -- you should not need to call this function directly.
    Parameters
    ----------
    game : `isolation.Board`
        An instance of `isolation.Board` encoding the current state of the
        game (e.g., player locations and blocked cells).
    player : object
        A player instance in the current game (i.e., an object corresponding to
        one of the player objects `game.__player_1__` or `game.__player_2__`.)
    Returns
    -------
    float
        The heuristic value of the current game state to the specified player.
    """
    if game.is_winner(player) or game.is_loser(player):
        return game.utility(player)
    my_moves = len(game.get_legal_moves())
    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

    return float(my_moves - opp_moves + centrality(game, game.get_player_location(player)) + common_moves(game, player))
```

First function is straight forward, my available move – opponent's available moves help agent to take out our opponents move, plus the centrality and common_moves which help us to chase our opponents and corner them.

```
def custom_score_2(game, player):
    #get opponents
    if game.is_winner(player) or game.is_loser(player):
        return game.utility(player)
    my_moves = len(game.get_legal_moves())
    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
    return float(my_moves - opp_moves*1.3 + centrality(game, game.get_player_location(player)))
```

The second function is mainly focusing on cancelling out opponent's move, opp_moves*1.3 helps our agent plays more aggressively, centrality corners our opponent.

```
def custom_score_3(game, player):
    my_moves = float(len(game.get_legal_moves(game._active_player)))
    opp_moves = float(len(game.get_legal_moves(game._inactive_player)))

    return float(my_moves - opp_moves*1.3)
```

Function three is simple weighted evaluation function, which allow our agent plays aggressively.

```
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     9  |   1     7  |   3
   2       MM_Open      6  |   4     6  |   4     7  |   3     2  |   8
   3      MM_Center     7  |   3     8  |   2     7  |   3     6  |   4
   4     MM_Improved    8  |   2     8  |   2     8  |   2     2  |   8
   5       AB_Open      5  |   5     4  |   6     6  |   4     4  |   6
   6      AB_Center     8  |   2     6  |   4     5  |   5     6  |   4
   7     AB_Improved    5  |   5     4  |   6     5  |   5     2  |   8
--------------------------------------------------------------------------
         Win Rate:      68.6%        65.7%        67.1%        41.4%
```

| Match # | Opponent | AB_Improved | | AB_Custom | | AB_Custom_2 | | AB_Custom_3 | |
|---|---|---|---|---|---|---|---|---|---|
| | | Won | Lost | Won | Lost | Won | Lost | Won | Lost |
| 1 | Random | 9 | 1 | 9 | 1 | 10 | 0 | 5 | 5 |
| 2 | MM_Open | 8 | 2 | 4 | 6 | 6 | 4 | 3 | 7 |
| 3 | MM_Center | 7 | 3 | 8 | 2 | 8 | 2 | 2 | 8 |
| 4 | MM_Improved | 6 | 4 | 6 | 4 | 8 | 2 | 4 | 6 |
| 5 | AB_Open | 5 | 5 | 6 | 4 | 6 | 4 | 5 | 5 |
| 6 | AB_Center | 8 | 2 | 2 | 8 | 8 | 2 | 4 | 6 |
| 7 | AB_Improved | 3 | 7 | 8 | 2 | 7 | 3 | 6 | 4 |
| | Win Rate: | 65.7% | | 61.4% | | 75.7% | | 41.4% | |

| Match # | Opponent | AB_Improved | | AB_Custom | | AB_Custom_2 | | AB_Custom_3 | |
|---|---|---|---|---|---|---|---|---|---|
| | | Won | Lost | Won | Lost | Won | Lost | Won | Lost |
| 1 | Random | 9 | 1 | 9 | 1 | 6 | 4 | 8 | 2 |
| 2 | MM_Open | 3 | 7 | 5 | 5 | 8 | 2 | 4 | 6 |
| 3 | MM_Center | 8 | 2 | 8 | 2 | 7 | 3 | 6 | 4 |
| 4 | MM_Improved | 6 | 4 | 6 | 4 | 8 | 2 | 2 | 8 |
| 5 | AB_Open | 5 | 5 | 4 | 6 | 4 | 6 | 5 | 5 |
| 6 | AB_Center | 5 | 5 | 7 | 3 | 3 | 7 | 6 | 4 |
| 7 | AB_Improved | 6 | 4 | 1 | 9 | 5 | 5 | 2 | 8 |
| | Win Rate: | 60.0% | | 57.1% | | 58.6% | | 47.1% | |

From three tests, we found out that second evaluation function has the best performance, followed by the first one. We can found out that the common_move function did not boost the winning rate as much as the simple weighted evaluation function. But the weighted evaluation function along, could not provide accurate value for agent to win the game. It might due to the fact that agent only has a short time to compute the score. Fewer variables cause evaluation function has lots of same score, if the agent does not search deep enough. In this case, agent usually makes wrong decisions.