Heuristic Analysis

Below are my heuristic functions of the isolation game:

- The first heuristic calculates the square of the difference of the number of the player's legal moves and 2 times the number of opponent's legal moves.

```
def custom_score(game, player):

    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)**2
```

- The second heuristic first calculates two scores equal to square of the distance from the center of the board to the position of the active player and the opponent player respectively. It then returns the difference of these two scores.

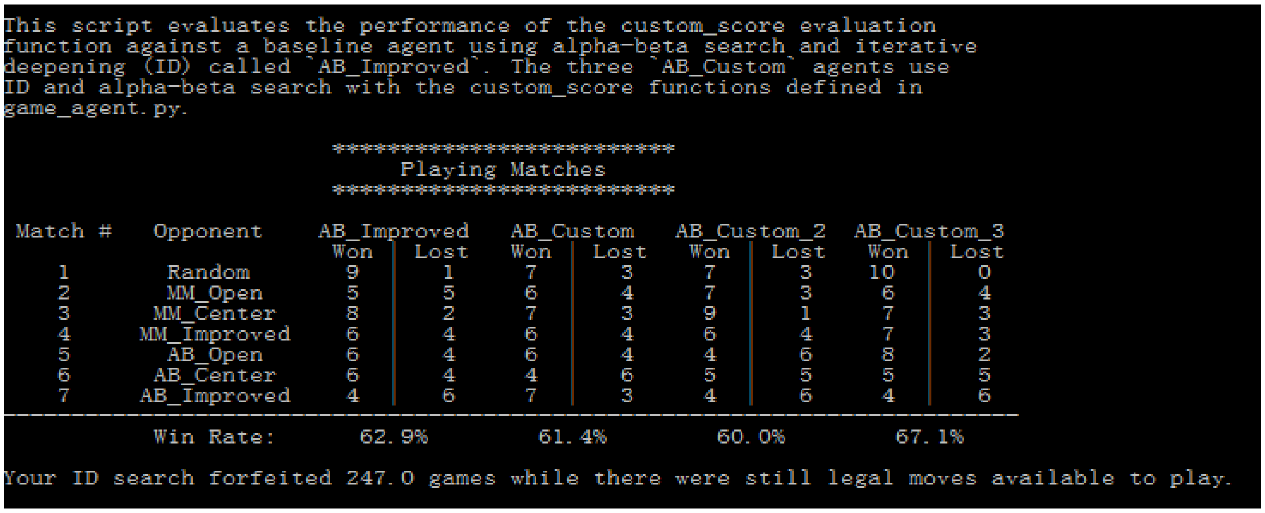
```
def custom_score_2(game, player):

    if game.is_loser(player):
        return float("-inf")

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

    w, h = game.width / 2., game.height / 2.
    y1, x1 = game.get_player_location(player)
    y2, x2 = game.get_player_location(game.get_opponent(player))
    player_1_score = float((h - y1)**2 + (w - x1)**2)
    player_2_score = float((h - y2)**2 + (w - x2)**2)

    return float(player_1_score - player_2_score)
```

- The third heuristic combines the above two heuristic functions. It calculates the sum of the scores in heuristic one and heuristic two.

```
def custom_score_3(game, player):
```

```python
    if game.is_loser(player):
        return float("-inf")

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

    w, h = game.width / 2., game.height / 2.
    y1, x1 = game.get_player_location(player)
    y2, x2 = game.get_player_location(game.get_opponent(player))
    player_1_score = float((h - y1)**2 + (w - x1)**2)
    player_2_score = float((h - y2)**2 + (w - x2)**2)

    own_moves = len(game.get_legal_moves(player))
    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

    return float(10 * (own_moves - 2 * opp_moves) + (player_1_score
- player_2_score))
```

The performance of the heuristic functions are as follows:



```
This script evaluates the performance of the custom_score evaluation
function against a baseline agent using alpha-beta search and iterative
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     7  |   3     7  |   3    10  |   0
    2        MM_Open      5  |   5     6  |   4     7  |   3     6  |   4
    3       MM_Center     8  |   2     7  |   3     9  |   1     7  |   3
    4      MM_Improved    6  |   4     6  |   4     6  |   4     7  |   3
    5        AB_Open      6  |   4     6  |   4     4  |   6     8  |   2
    6       AB_Center     6  |   4     4  |   6     5  |   5     5  |   5
    7      AB_Improved    4  |   6     7  |   3     4  |   6     4  |   6
--------------------------------------------------------------------------
           Win Rate:      62.9%        61.4%        60.0%        67.1%

Your ID search forfeited 247.0 games while there were still legal moves available to play.
```

All the heuristic functions have winning rate of more than 50%. Heuristic 3 is the function to be chosen for the game agent because it has the highest win rate of 67.1%. It has perfect win rate against random opponent. It also performed better than all MinimaxPlayer agents. It only lost to AB_Improved opponent with 4 wins and 6 losses. This heuristic is simple to be calculated. A fast and less computation overhead heuristic function can give the agent advantage to search deeper in the game tree and increase the chance of winning.