Heuristic Analysis

Custom Score 3:

Implementation:

```
def custom_score_3(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)
```

Analysis:

This score function is based on the improved score function by adding an additional constant factor to multiply `opp_moves`. That would make the agent more likely to select moves which minimize the legal moves available to the opponent. If the opponent has many legal moves available, it will be penalized by this score function. The performance of this is listed down below:

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

 Match #   Opponent    AB_Improved     AB_Custom_3
                        Won | Lost     Won | Lost
    1       Random      14  |   6      14  |   6
    2       MM_Open     12  |   8      12  |   8
    3      MM_Center    15  |   5      18  |   2
    4     MM_Improved   10  |  10      13  |   7
    5       AB_Open      9  |  11      12  |   8
    6      AB_Center    10  |  10      12  |   8
    7     AB_Improved    9  |  11      10  |  10
--------------------------------------------------------------------------
           Win Rate:      56.4%          65.0%
```

It is significantly better than the AB_Improved function on average.

Custom Score 2:

Implementation:

```
def custom_score_2(game, player):
    if game.is_loser(player):
        return float("-inf")

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

    best_first_move = (int(game.height/2), int(game.width/2))
    if best_first_move == game.get_player_location(player):
        return float("inf")

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

    if own_moves_num == opp_moves_num:
        own_pos = game.get_player_location(player)
        opp_pos = game.get_player_location(game.get_opponent(player))
        own_dis = abs(own_pos[0] - best_first_move[0]) + abs(own_pos[1] -
best_first_move[1])
        opp_dis = abs(opp_pos[0] - best_first_move[0]) + abs(opp_pos[1] -
best_first_move[1])
        return float(own_dis - opp_dis) / 10.0
    else:
        return float(own_moves_num - 2 * opp_moves_num)
```

Analysis:

This score function combines the Improved and Center score functions. The distance is calculated by not only considering the player's position but also the opponent's position. That is similar to the Improved Score function. Result:

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

 Match #   Opponent    AB_Improved   AB_Custom_2
                        Won | Lost   Won | Lost
    1       Random      15  |   5    15  |   5
    2       MM_Open     13  |   7    15  |   5
    3      MM_Center    13  |   7    18  |   2
    4     MM_Improved   15  |   5    14  |   6
    5       AB_Open     12  |   8    11  |   9
    6      AB_Center    11  |   9    11  |   9
    7     AB_Improved    9  |  11    11  |   9
----------------------------------------------------------------------
           Win Rate:      62.9%        67.9%
```

A slightly better than the improved score function.

Custom Score:

Implementation:

```
def custom_score(game, player):
    if game.is_loser(player):
        return float("-inf")

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

    h, w = int(game.height/2), int(game.width/2)
    if (h, w) == game.get_player_location(player):
        return float("inf")

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

    if own_moves == opp_moves:
        y, x = game.get_player_location(player)
        return float(abs(h-y)+abs(w-x))/10.0
    else:
        return float(own_moves - opp_moves)
```

Analysis:

This score function just combines the Center and Improved score functions. If the value of own_moves equals to the opp_moves, the distance between current position of the player and the center of the board will be returned. Since the improved score function works better than the center score function. We divided the distance by 10.0, so this distance value is less likely to have a large impact when comparing to a value calculated by improved function. In addition, we choose to use Manhattan distance rather than Euclidean distance since it works better after a few experiments. Results:

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

 Match #   Opponent    AB_Improved     AB_Custom
                        Won | Lost      Won | Lost
    1       Random      15  |   5       19  |   1
    2       MM_Open     12  |   8       14  |   6
    3      MM_Center    15  |   5       17  |   3
    4     MM_Improved   13  |   7       12  |   8
    5       AB_Open     13  |   7       13  |   7
    6      AB_Center    15  |   5       12  |   8
    7     AB_Improved    9  |  11       10  |  10
----------------------------------------------------------------------
           Win Rate:      65.7%           69.3%
```

Recommendation:

I would recommend the **custom_score** function as the final evaluation function for three reasons:

- It performs best among my three heuristic functions (~70% winning rate).
- After a large amount of experimental runs, the other two score functions may perform worse than the **AB_Improved** function in some cases. However, the **custom_score** function gives an reasonable good result constantly.
- This function is very simple but effective. By using the Manhattan distance, it only takes a very little amount of time to compute which means it is able to search much deeper and probably give a better result than other complex evaluation functions.