Heuristic Analysis

I used the following 3 heuristics for the game playing agent.

Heuristic 1

```
# TODO: finish this function!
if game.is_loser(player):
    return float("-inf")

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

my_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
return float(my_moves - opp_moves)
```

Here I was keeping the score as my_moves – opponent_moves to ensure I win the game based on number of moves.

Heuristic 2

```
# TODO: finish this function!
if game.is_loser(player):
    return float("-inf")

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

my_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
return float(my_moves - 2*opp_moves)
```

Here I was keeping the score as my_moves – 2 * opponent_moves to ensure I win the game based on number of moves. This was a more aggressive scoring function to win in fewer number of moves.

Heuristic 3

```
# TODO: finish this function!
if game.is_loser(player):
    return float("-inf")

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

my_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
return float(my_moves - 3*opp_moves)
```

Here I was keeping the score as my_moves – 3 * opponent_moves to ensure a more aggressive win strategy by beating the opponent in fewer moves than the previous heuristic functions.

Outcome

```
(aind) C:\Users\Rohan\ai-nd\AIND-Isolation>python tournament.py

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       10  |   0    10  |   0    10  |   0     9  |   1
    2       MM_Open       8  |   2     8  |   2    10  |   0     9  |   1
    3      MM_Center      7  |   3     9  |   1     9  |   1     9  |   1
    4     MM_Improved     6  |   4     6  |   4     7  |   3     9  |   1
    5       AB_Open       6  |   4     7  |   3     5  |   5     5  |   5
    6      AB_Center      6  |   4     8  |   2     9  |   1     6  |   4
    7     AB_Improved     4  |   6     4  |   6     3  |   7     3  |   7
--------------------------------------------------------------------------
          Win Rate:      67.1%        74.3%        75.7%        71.4%
```

Looking at the outcomes, my custom_score_2 outperformed all other heuristic functions.

- AB_Custom_3 under performed even though the function was more aggressive (my_moves – 3 * opp_moves) as the search may have timed out and therefore returned sub-optimal results.

- AB_Custom_2 (my_moves – 2* opp_moves) outperformed the general algorithm as it had a balanced heuristic that ensured the opponent was beaten in fewer moves while ensuring that the computation speed was within the timeout period defined to return optimal results in each step of the search.
- AB_Custom (my_moves – opp_moves) performed much better than AB_Custom_3 even though it was less aggressive in trying to win in fewer moves. This was because the computation power required to go deeper during each iteration was less and therefore returned a more optimal value.