Heuristic Analysis

Heuristic Function 1

```
def _unused_heuristic_function_for_submission_2(game, pla
yer):
    # ~53% of games won in tournament
    # Return inf or -inf if the player has won/lost w/ le
ast comparisions
    win_or_lose = game.utility(player)
    if win_or_lose != 0:
        return win_or_lose

    opponent = game.get_opponent(player)
    player_move_count = len(game.get_legal_moves(player))
    opponent_move_count = len(game.get_legal_moves(oppone
nt))

    return float(player_move_count - 2 * opponent_move_co
unt)
```

Originally, this function only caculated the number of moves available to the current agent. With experimentation, it was found that subtracting the number of the moves available to the opponent was useful when scoring the game state. It was decided to weight the negative value of the opponent's moves twice as much as the agent's

moves.

Heuristic Function 2

```
def _unused_heuristic_function_for_submission_1(game, pla
yer):
    # ~59% of games won in tournament
    # Return inf or -inf if the player has won/lost w/ le
ast comparisions
    win_or_lose = game.utility(player)
    if win_or_lose != 0:
        return win_or_lose

    player_move_count = len(game.get_legal_moves(player))

    # if less than 50% of the game has been played, we us
e a less computationally expensive
    # heuristic function because the branching factor is
larger on average
    if game.move_count/float(game.width * game.height) <
0.5:
        return player_move_count

    opponent = game.get_opponent(player)
    opponent_move_count = len(game.get_legal_moves(oppone
nt))

    return float(player_move_count - 2 * opponent_move_co
```

```
unt)
```

This function builds upon the last, changing the evaluation at the 50% mark of the game. If less than 50% of the game has been played, we use only use the current agent's move count as it is less computationally expensive. This is so we can search relatively deeper as opposed to later on in the game, when we want to search "smarter".

Heuristic Function 3

```
def custom_score(game, player):
    # ~70% of games won in tournament
    # Return inf or -inf if the player has won/lost w/ le
ast comparisions
    win_or_lose = game.utility(player)
    if win_or_lose != 0:
        return win_or_lose

    opponent = game.get_opponent(player)
    opponent_move_count = len(game.get_legal_moves(oppone
nt))

    # if less than 50% of the game has been played, we us
e a less computationally expensive
    # heuristic function because the branching factor is
larger on average
    if game.move_count/float(game.width * game.height) <
```

```
0.5:
        return -opponent_move_count

    player_move_count = len(game.get_legal_moves(player))

    return float(player_move_count - 2 * opponent_move_co
unt)
```

This function builds upon function 2, however uses the opponent's move count before the 50% mark as opposed to the current players moves. This function showed the most promising results.

Note: When simply returning the opponent's move count, we do not need to give it a weight of 2 as it is a constant factor. Also boards with less than 50% of moves will not be compared with boards with over 50% of moves.

The tournament.py result with the last function is shown below:

```
**************************
 Evaluating: ID_Improved
**************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 17 to 3
  Match 2: ID_Improved vs   MM_Null     Result: 10 to 10
```

```
  Match 3: ID_Improved vs   MM_Open     Result: 11 to 9
  Match 4: ID_Improved vs MM_Improved   Result: 12 to 8
  Match 5: ID_Improved vs   AB_Null     Result: 15 to 5
  Match 6: ID_Improved vs   AB_Open     Result: 13 to 7
  Match 7: ID_Improved vs AB_Improved   Result: 8 to 12


Results:
----------
ID_Improved         61.42%

*************************
   Evaluating: Student
*************************

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 18 to 2
  Match 2:   Student   vs   MM_Null     Result: 15 to 5
  Match 3:   Student   vs   MM_Open     Result: 10 to 10
  Match 4:   Student   vs MM_Improved   Result: 13 to 7
  Match 5:   Student   vs   AB_Null     Result: 15 to 5
  Match 6:   Student   vs   AB_Open     Result: 12 to 8
  Match 7:   Student   vs AB_Improved   Result: 13 to 7


Results:
----------
```

```
Student                 68.57%
```

Comparisions of the three functions are summarized in a table below:

Function	Games Won	Games Lost	Percent of games won
1	74	66	52.85%
2	83	57	59.28%
3	96	44	68.57%

Overall, the obvious recommendation is the third heuristic function. The function was able to outperform other functions by 10% in terms of the percent of games won. The function was also able to leverage the lower branching factor near the end of the game to search “smarter” and take the own agent’s moves into account. Also, during the first half of the game this function values only limiting the opponents moves and during the second half of the game it values limiting the opponents moves twice as much as the weight of out own moves. This puts the agent on the offensive as it is always trying to corner or box-in the opponent, which is the primary objective of this game.