Heuristic # 1:

Heuristic that calculates the log score of the probability that our player will win. Probability is calculated by finding the number of legal moves that our player can make in this turn and dividing by the total number of moves available in that turn which is the sum of the legal moves our player and the opponent can make. Taking the natural logarithm of this probability gives a higher result when our player can make higher number of moves in the current turn with respect to the number of moves the opponent can make.

Analysis:

This heuristic has allowed our player to consistently win over 50% of the games sometimes winning as many as 60% of all games against the opponent in tournament.py script. Its simple and quick to calculate without much of computing overhead which was an important consideration given that we have a limited amount of time to make a move, failure to do so would have resulted in our player forfeiting the turn and losing the game.

Heuristic # 2:

In my experience of manually playing Isolation against human opponents, if I try to remain in the centre of the board (filling in the centre row or column) and force my opponent to stay on the sides, I can ultimately block of my opponent from moving to any other areas thus overall giving him less moves and winning the game at the end. This strategy allows me to make a move to any side of the board if the empty squares in the centre row and columns have all been filled. First the Manhattan distance is calculated from our current position to the centre of the board then for each move in the legal moves for that turn the move with the smallest distance to the centre is returned thereby allowing us to move as close to the centre of the board as possible giving us much more chance of winning the game.

Analysis:

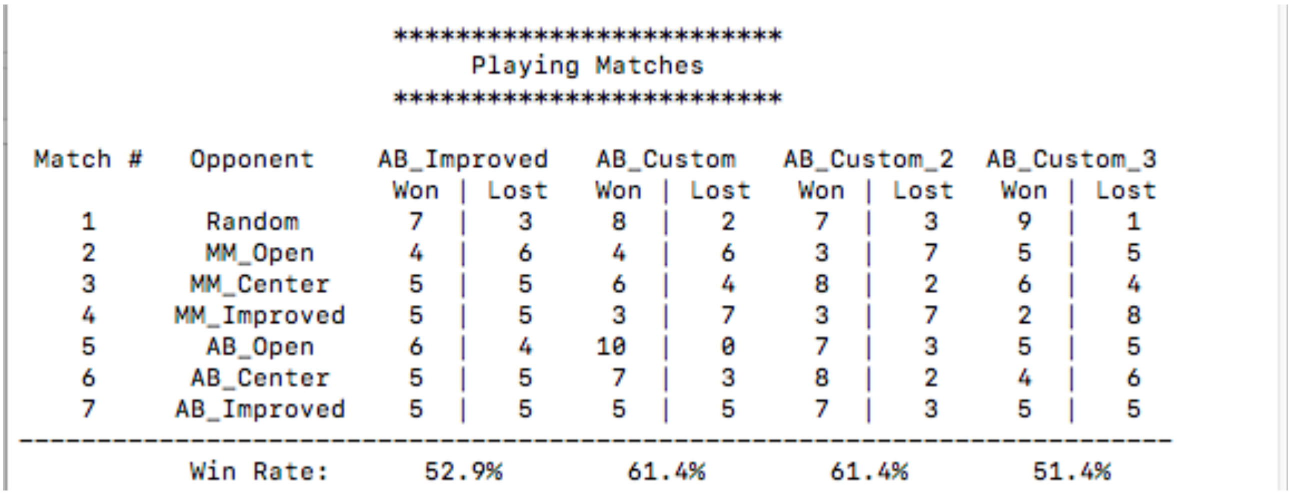
This heuristic has slightly more computing overhead then the first heuristic as the distance for all possible moves to the centre have to be calculated but it performed well overall consistently winning more than 50% of all games against its opponent in tournament.py.

Heuristic # 3:

This heuristic takes into consideration the number of moves our player and the opponent have available in this turn but also the number of moves they both have in the next turn. Dividing by the number of moves the opponent has by our moves forces us to take moves where the number of the opponents moves are less. Adding moves one ply deep also takes into consideration the moves in the next turn. Subtraction forces this heuristic to aggressively cut down on the moves that the opponent can make in the next turn.

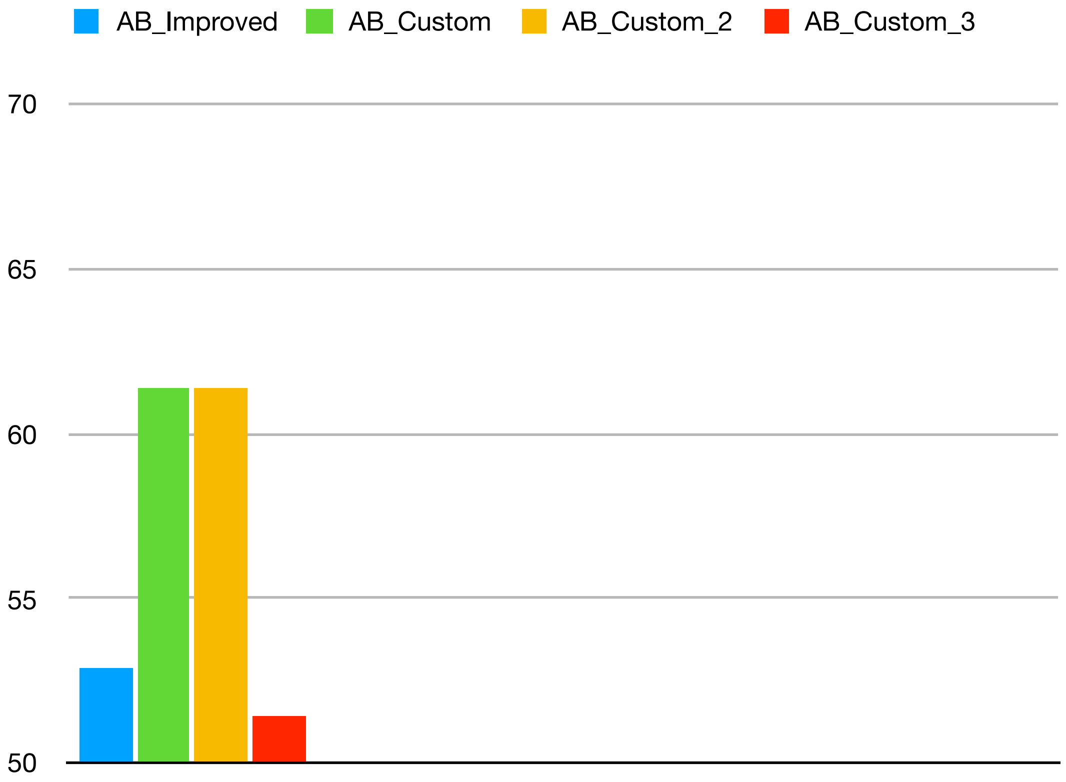
Analysis:

This is the weakest of the heuristic amongst all three as it barely crosses 50% win rate against its opponent in tournament.py but on the plus side its fast.



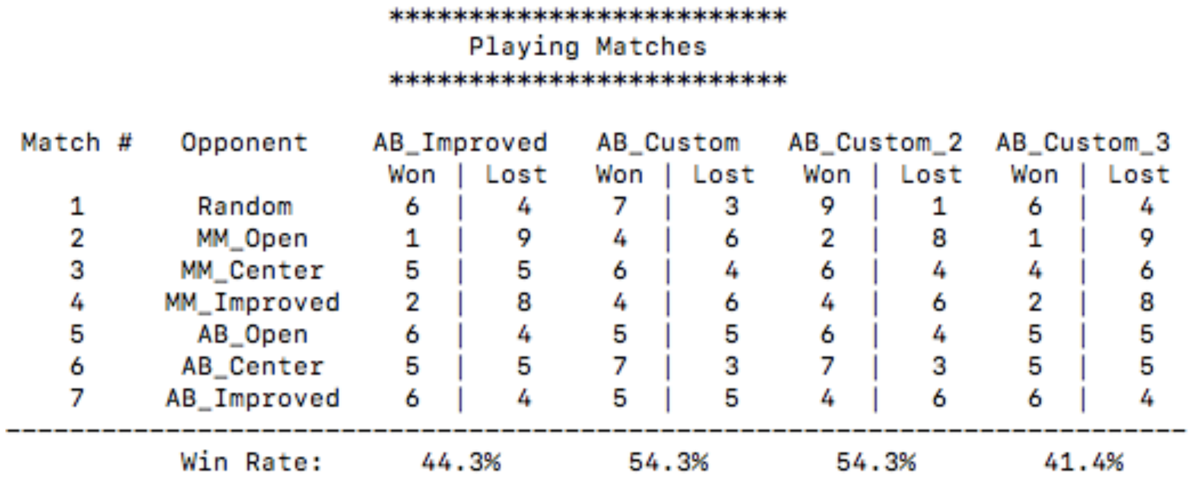
```
                          *************************
                               Playing Matches
                          *************************

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random       7  |   3     8  |   2     7  |   3     9  |   1
    2       MM_Open      4  |   6     4  |   6     3  |   7     5  |   5
    3      MM_Center     5  |   5     6  |   4     8  |   2     6  |   4
    4     MM_Improved    5  |   5     3  |   7     3  |   7     2  |   8
    5       AB_Open      6  |   4    10  |   0     7  |   3     5  |   5
    6      AB_Center     5  |   5     7  |   3     8  |   2     4  |   6
    7     AB_Improved    5  |   5     5  |   5     7  |   3     5  |   5
--------------------------------------------------------------------------
           Win Rate:      52.9%        61.4%        61.4%        51.4%
```



Final Thoughts:

The first heuristic has proven to be the best amongst all three heuristics described in this analysis. Its fast to compute, provides solid results giving consistent win rates. Also takes into account the most important feature that we have to keep in mind when playing a game of Isolation i.e. for every move we make, we must have more moves available going forward than what the opponent has available. This heuristic encapsulates this behaviour in a simple probability function and calculates the log of this probability using the built-in log function in Python.



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

 Match #     Opponent      AB_Improved   AB_Custom    AB_Custom_2  AB_Custom_3
                            Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1         Random         6  |   4     7  |   3     9  |   1     6  |   4
    2         MM_Open        1  |   9     4  |   6     2  |   8     1  |   9
    3        MM_Center       5  |   5     6  |   4     6  |   4     4  |   6
    4       MM_Improved      2  |   8     4  |   6     4  |   6     2  |   8
    5         AB_Open        6  |   4     5  |   5     6  |   4     5  |   5
    6        AB_Center       5  |   5     7  |   3     7  |   3     5  |   5
    7       AB_Improved      6  |   4     5  |   5     4  |   6     6  |   4
--------------------------------------------------------------------------
           Win Rate:       44.3%        54.3%        54.3%        41.4%
```

Another example where Heuristic # 1 provided win rates above 50%, and did not lose by a huge margin unlike the other two for example in Match # 2 heuristic 2 lost 2-8 to its opponent and heuristic 3 (in this example this was simply my_moves-2*opp_moves) lost 1-9 to its opponent.

A combination of the first and second heuristic would be even more powerful as it would choose moves where our player has more freedom to move around the board and it would also take into consideration the moves which allow us to keep as close to centre of the board as possible. But this would be quite an expensive (computationally) heuristic the result of which would be that the same processing device would not be able to go deeper into the search tree to find better moves in the same amount of time.