I've implemented 3 different heuristic functions:

- custom_score: improved the improved_score by adding the weighted coefficients to the number of legal move for player 1 and player 2 – score = w1*player1_moves – w2*player2_moves. Where w1 = 3 and w2 = 4, so it makes perfect sense that if the score is positive, the player 1 is in much better situation comparing to player 2. Easy to implement.
- custom_score_2: attempt to make a partitioning of the board where the result is the difference of the number of empty spaces available for player's 1 part of the board to the number of empty spaces available for player's 2 part of the board.
- custom_score_3: getting the difference between the squared centered distance for player 2 and squared centered distance for player 1. In this case we assume that if player 1 is closer to the center, than player 2, it has higher chances to win. Easy to implement.

I've made 3 test runs. Following are the results.

```
            **************************
                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    8 |   2    10 |   0    10 |   0
   2      MM_Open     9 |   1    7 |   3     5 |   5     6 |   4
   3     MM_Center    9 |   1    8 |   2     7 |   3     5 |   5
   4    MM_Improved   5 |   5    6 |   4     6 |   4     6 |   4
   5     AB_Open      5 |   5    6 |   4     5 |   5     4 |   6
   6     AB_Center    5 |   5    8 |   2     4 |   6     6 |   4
   7    AB_Improved   6 |   4    6 |   4     4 |   6     6 |   4
----------------------------------------------------------------------
       Win Rate:      64.3%       70.0%       58.6%       61.4%


            **************************
                Playing Matches
            **************************

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                 Won | Lost   Won | Lost   Won | Lost   Won | Lost
   1      Random      8 |   2    9 |   1     7 |   3     8 |   2
   2      MM_Open     5 |   5    7 |   3     5 |   5     8 |   2
   3     MM_Center    6 |   4    7 |   3     5 |   5     6 |   4
   4    MM_Improved   6 |   4    6 |   4     6 |   4     6 |   4
   5     AB_Open      4 |   6    6 |   4     4 |   6     4 |   6
```

```
  6     AB_Center    8  |   2    5  |   5    6  |   4    4  |   6
  7    AB_Improved   3  |   7    5  |   5    5  |   5    6  |   4
--------------------------------------------------------------------------
       Win Rate:     57.1%       64.3%       54.3%       60.0%

            *************************
                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    9  |   1    8  |   2    6  |   4
   2      MM_Open     5  |   5    8  |   2    9  |   1    4  |   6
   3     MM_Center    5  |   5    7  |   3    7  |   3    8  |   2
   4    MM_Improved   8  |   2    8  |   2    8  |   2    5  |   5
   5      AB_Open     7  |   3    5  |   5    6  |   4    5  |   5
   6     AB_Center    5  |   5    6  |   4    4  |   6    4  |   6
   7    AB_Improved   4  |   6    3  |   7    4  |   6    5  |   5
--------------------------------------------------------------------------
       Win Rate:     61.4%       65.7%       65.7%       52.9%
```

I would recommend the custom_score heuristics because of the following reasons:

- Based on the results above, custom_score heuristics constantly outperforms the improved_score based on the assumptions described above (an average win ratio for custom_score is 66.66%, for improved_score is 60.93%).
- It's easy to implement and even though its complexity is O(n) (2 calls of isolation.get_legal_moves() which takes O(n) – cycles + random.shuffle), in our case n == 8 (considering that we have only 8 moves max if the board is not in initial state), so on that small amount of data this should perform pretty fast. If we consider the collection of legal moves to be of a much bigger size, I would think of a different heuristics, e.g. custom_score_3 just for performance sake...
- It predicts the final outcome of the game pretty accurately which has been shown by the empirical results.
- It doesn't traverse the game tree, it just collects moves, so, basically it's pretty simple and performs well.