# Heuristic Analysis of AI Isolation

## Summary

After implementing the MiniMax and AlphaBeta algorithms, we implemented 3 custom scoring heuristics which, when called, provide a score for the current game state. A higher score implies the current state is more likely to lead to a win for the current player.

## Detail

By way of example we were provided with several example score functions, the best performing of which(**improved_score**), simply returns $moves_{own} \ - \ moves_{opp}$. The idea is that the player with more remaining moves has an advantage, and so a higher positive score for player 1 results in an advantage.

For our version of the scorer, we enhanced the **improved_score** version:

- **custom_score_3:** We have added a metric about the game state, "empty_board", which contains the number of spaces left unblocked in the game. We divide the **improved_score** by this number. The idea behind this is that when the game begins, no particular move is very valuable, as we have plenty of room to move still

  $$\frac{moves\_own - moves\_opp}{empty\_board}$$

- **custom_score_2:** Similar to the prior scorer, we have added another metric, "moves_left", which is the total number of moves left to all players. While simply substituting this in where empty_board was works, it provided no change in game outcome. instead we combined the two metrics, as a deeper indicator of how much game remains. In addition, we found that squaring the number of player moves led to an improved score.

  $$\frac{moves\_own^2 - moves\_opp^2}{moves\_left + empty\_board}$$

- **custom_score:** In the final version of the scorer, we used the idea that during the early portion of the game, the distance from the center will pay an important role. To this end during the first half of the game, we modify the prior formula from custom score_2 by adding their "center score" to the number of moves of each player. "center_score" is generated using the code from the example scorers, and is the square of the distance from the center of the board to the position of the player. During the final half of the game, we procede as in custom_score_2.

  $$\frac{(moves\_own + \frac{center\_score}{moves\_left + empty\_board})^2 - (moves\_opp)^2}{moves\_left + empty\_board}$$

  Additionally, I implemented a scoring model that outputs a probability that a specific game state leads to a win for a specific player, and then multiplied the improved score by this probability, etc. The model used ExtraTreesClassifier from scikit-learn, and generated game state and win/loss by saving the output of approximately 50,000 games. Details are available in score_model.py in the git repository. This model proved to be to slow computationally and lost due to reaching the timeout condition in all games, and so is unviable for the current testing scenario.

## Conclusion

It's difficult to beat the given included scorer, **improved_score**. While most of the time the custom scorers described above are able to perform as well as the given, in some runs they perform worse. In tournaments with high(50+) NUM_MATCHES, it was seen that a consistent improvement was possible of at least 1%. In some runs, we observed a 10% improvement. When we compare best runs between the given and our custom scorer, over all tests, we saw a 6% improvement in best score. Overall, **custom score_3** seems to perform most consistently better than **improved_score**, and **custom_score** achieves the highest overall scores. As written, I would recommend **custom score_3** for further game play. It is the simplest method, and closest to the original **improved_score**. It is also the most consistent in maintaining a winning score across all my tests, even though it did not acheive the highest score seen in testing. As seen in the two runs below, it generally maintained a score betwen of 67-71%, with average slightly over 68%. The **improved_score** maintained an avg of slightly under 68%.

Further improvements could be made by an in-depth analysis of game play, and creating scoring heuristics that credited good/bad strategy appropriatly.

*Example tournament runs:*

```
 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      95  |   5    95  |   5    91  |   9    95  |   5
    2       MM_Open     67  |  33    81  |  19    76  |  24    66  |  34
    3      MM_Center    92  |   8    92  |   8    90  |  10    86  |  14
    4     MM_Improved   72  |  28    67  |  33    64  |  36    75  |  25
    5       AB_Open     53  |  47    52  |  48    39  |  61    51  |  49
    6      AB_Center    56  |  44    61  |  39    60  |  40    63  |  37
    7     AB_Improved   50  |  50    47  |  53    52  |  48    45  |  55
--------------------------------------------------------------------------
           Win Rate:      69.3%        70.7%        67.4%        68.7%
```

```
 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      91  |   9    94  |   6    91  |   9    96  |   4
    2       MM_Open     74  |  26    75  |  25    71  |  29    79  |  21
    3      MM_Center    84  |  16    92  |   8    90  |  10    91  |   9
    4     MM_Improved   69  |  31    72  |  28    72  |  28    70  |  30
    5       AB_Open     45  |  55    43  |  57    52  |  48    56  |  44
    6      AB_Center    59  |  41    55  |  45    63  |  37    56  |  44
    7     AB_Improved   49  |  51    43  |  57    50  |  50    45  |  55
--------------------------------------------------------------------------
           Win Rate:      67.3%        67.7%        69.9%        70.4%
```

see https://github.com/llathrop/AIND-Isolation (https://github.com/llathrop/AIND-Isolation) for full implementation code, etc