RESULTS

Here is how the three custom heuristics that I created for this project peformed against the other agents:

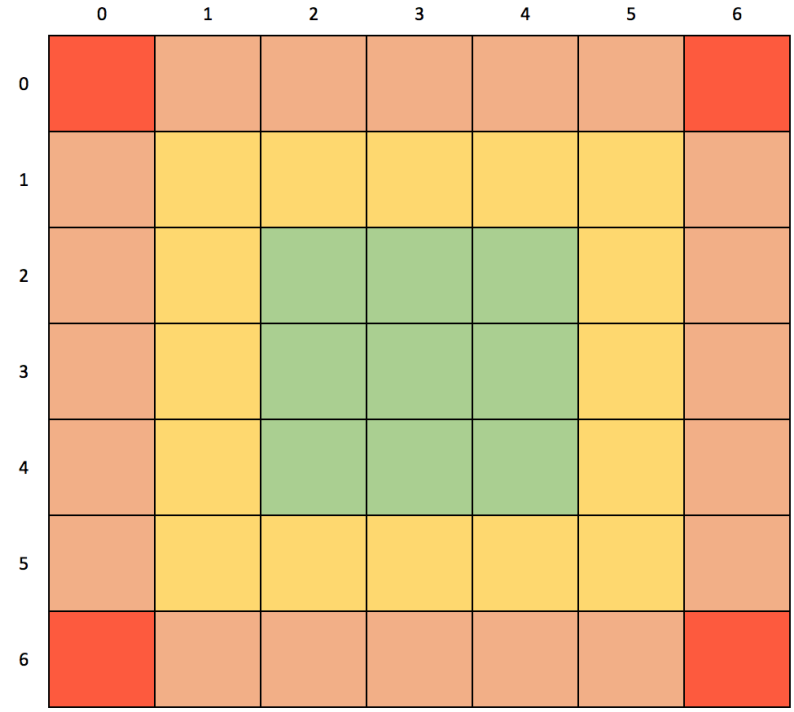
```
                         **************************
                              Playing Matches
                         **************************

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                       Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      93  |   7    95  |   5    96  |   4    98  |   2
    2       MM_Open     73  |  27    74  |  26    82  |  18    71  |  29
    3      MM_Center    91  |   9    90  |  10    86  |  14    84  |  16
    4     MM_Improved   73  |  27    84  |  16    66  |  34    66  |  34
    5       AB_Open     54  |  46    56  |  44    54  |  46    47  |  53
    6      AB_Center    59  |  41    62  |  38    61  |  39    53  |  47
    7     AB_Improved   45  |  55    52  |  48    48  |  52    44  |  56
------------------------------------------------------------------------
          Win Rate:      69.7%        73.3%        70.4%        66.1%
```

AB_Custom_3

(link to code)

This was my original exploration of heuristics that exploit board position. This function rewards moves toward the center of the board, since these squares have more legal moves available to them. Meanwhile, the function penalizes moves along the edges and corners, since these squares have fewer moves available to them.

To achieve this, the board is divided into four sections (see diagram below).



The scoring logic can be summarized as follows:

- Moves in the center portion of the board (green) receive a higher score of 5 throughout the game, and a score of 10 at the beginning of the game (first seven moves)
- Moves in the second ring of the board (yellow) receive a score of 3
- Moves along the edges receive a score of 1
- Moves in the corners receive a score of 0

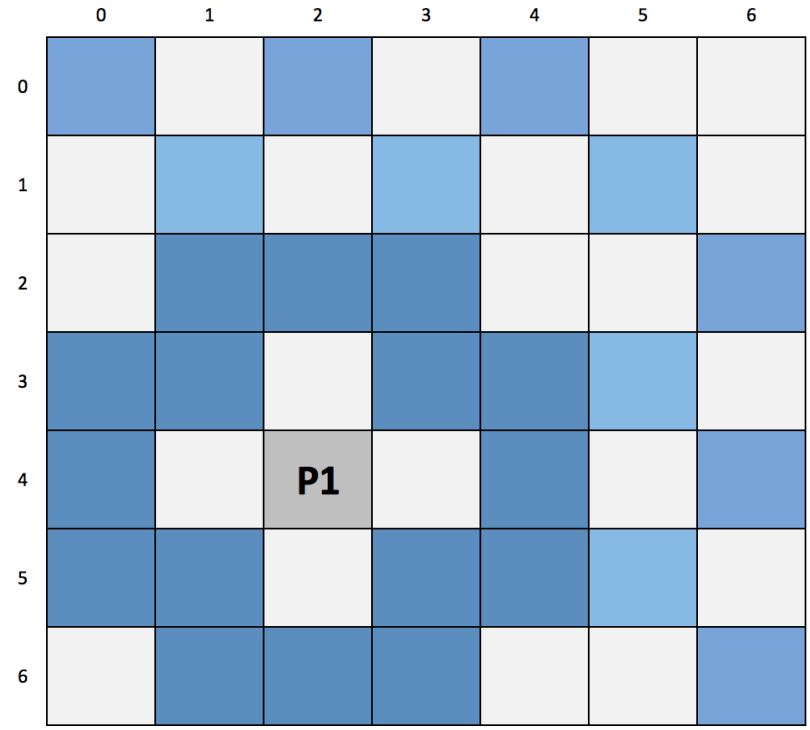
This heuristic does not outperform AB_Improved. However, it still managed a respectable 44% winning percentage in direct competion with AB_Improved. And, it does much better than the Random and Minimax agents, winning 80% of the time. Given these results, I continued iterating on this 'center is better' strategy and utilized elements of it when developing the subsequent function.

AB_Custom_2

(link to code)

This was my second attempt to exploit board position, but this time I tried to do it more systematically. The goal of this heurisitic is to reduce the mobility of the opponent — while simultaneously improving the active player's mobility. This would naturally favor positions in the middle of the board, but I wanted to get away from hard coding this into the strategy as I did previously in AB_Custom_3.

After many iterations on this concept, I eventually found a simple tactic that performed well. The final heuristic function calculates the number of available squares a player could reach within two moves (see diagram below). It then rewards moves that: (a) increase the number of reachable squares for the active player, and (b) decrease the number of squares reachable by the opponent.



This heuristic peforms as well as AB_Improved, maybe slightly better. However, after much testing, I wasn't able to consistently beat the AB_Improved heuristic by a significant margin. So, I concluded that since the game of Isolation is won by the player who makes the most moves, that focusing on the number of moves remaining (not the number of open spaces) would be the best avenue to pursue if I wanted get over the hump.

AB_Custom

(link to code)

If you can't beat 'em, join 'em. After many different permutations, my final AB_Custom heuristic is essentially an improved version of AB_Improved, but there is one big difference. The AB_Custom heuristic consistently adjusts its tactics throughout the match.

In the beginning of the match, the function is more aggresive at reducing the number of opponent moves. But, as the game goes on, it becomes less concerned with minimizing opponent moves, and becomes increasingly aggressive at maximizing the active player's moves. To me this made intuitive sense, since you want to limit your opponent's options as much as possible in the beggining of the match, but at some point you have to focus on creating the longest string of moves possible for yourself.

You can see in the code snippet below that this is accomplished using a weight that is inversely proportional to the number of the moves in the match.

```
    # get current move count
    move_count = game.move_count

    # count number of moves available
    own_moves = len(game.get_legal_moves(player))
    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

    # calculate weight
    w = 10 / (move_count + 1)

    # return weighted delta of available moves
    return float(own_moves - (w * opp_moves))
```

Recommendation:

AB_Custom is the recommended evaluation function for the following reasons:

1. It outperformed all of the other heuristics, winning 73.3% of its matches, which is almost 3% higher than the runner-up (AB_Custom_2).
2. It outperformed the benchmark heuristic (AB_Improved) in its head-to-head matchup, winning 52 matches vs 48.
3. It has great breadth of performance. AB_Custom had either the first or second highest score in matches against all other minimax and alpha-beta heuristics.
4. By adjusting its strategy throughout the course of the match, it demonstrates a greater level of game awareness . The tactics of all other heuristics are relatively static.