HEURISTIC ANALYSIS

CUSTOM HEURISTIC #1

```
(own_moves - opp_moves) + (opp_distance_to_center -
distance_to_center)*0.634/(game.move_count))
```

My intuition behind developing this heuristic was to push the opponent to the walls, since that's where I observed most losses occur. I did this by building on the AB_Improved heuristic. The heuristic maximizes the distance from the center for the opponent and minimizes the distance from the center for my player. However, moves close to the center are rare towards the end of the game and center moves might not always be the right move towards the end of the game. Hence, I am reducing the weight of this distance parameter as more moves are played. 0.634 is a constant that I found gave best results when I played this heuristic iteratively against AB_Improved.

Result

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

 Match #   Opponent    AB_Improved   AB_Custom
                        Won | Lost   Won | Lost
    1       Random      40  |   0    40  |   0
    2       MM_Open     32  |   8    31  |   9
    3      MM_Center    36  |   4    37  |   3
    4     MM_Improved   31  |   9    31  |   9
    5       AB_Open     24  |  16    21  |  19
    6      AB_Center    25  |  15    22  |  18
    7     AB_Improved   19  |  21    24  |  16
--------------------------------------------------------------------------
          Win Rate:      73.9%        73.6%
```

Analysis

This heuristic performs adequately. The win rate isn’t better than AB_Improved when it plays 40 games, but it comes quite close. This heuristic is quick to compute and involves additional information about the state of the board. How important the distance to the center is at any point in the game is controlled by the varying weight. The constant in this weight was obtained by random sampling in iterative plays against AB_Improved. The value of this constant could be improved to achieve better results.

Implementation

```
def custom_score(game, player):
    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    own_moves = len(game.get_legal_moves(player))
    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

    w, h = game.width / 2., game.height / 2.
    y, x = game.get_player_location(player)
    y2, x2 = game.get_player_location(game.get_opponent(player))

    distance_to_center = float((h - y)**2 + (w - x)**2)
    opp_distance_to_center = float((h - y2)**2 + (w - x2)**2)
    return float((own_moves - opp_moves) + (opp_distance_to_center -
distance_to_center)*0.634/(game.move_count))
```

CUSTOM HEURISTIC #2

```
if percent_game_completed(0, 10, game):
     return 2*own_moves - 0.5*number_of_boxes_to_center
elif percent_game_completed(10, 40, game):
     return float(3*own_moves - opp_moves - 0.5*number_of_boxes_to_center)
else:
     return 2*own_moves - opp_moves
```

The intuition behind this heuristic was to stay in the center and use Manhattan distance to compute distance to the center instead of the Euclidean distance. During the initial 10% of the game, the agent tries to aggressively capture the center positions. For the rest of the game, the agent maximizes its own moves.

Result

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

 Match #   Opponent    AB_Improved  AB_Custom_2
                        Won | Lost   Won | Lost
    1       Random      40  |   0    38  |   2
    2       MM_Open     31  |   9    31  |   9
    3      MM_Center    39  |   1    35  |   5
    4     MM_Improved   35  |   5    31  |   9
    5       AB_Open     19  |  21    19  |  21
    6      AB_Center    24  |  16    25  |  15
    7     AB_Improved   23  |  17    18  |  22
--------------------------------------------------------------------------
           Win Rate:      75.4%        70.4%
```

Analysis

This heuristic performs adequately, but is clearly much worse than the simple AB_Improved. I think the Manhattan distance formula to compute the distance to the center might be performing better than the Euclidean distance formula since it is more relevant to the game player and is also faster to compute. The idea of switching strategies based on a stage in the game is also a good one, but the time to switch the strategy and the optimal strategy to use during a game stage is difficult to find. Towards the end of the game, we're not looking at the distance to the center because most moves will be away from the center and towards the walls anyway.

Implementation

```
def custom_score_2(game, player):
    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    own_moves = len(game.get_legal_moves(player))
    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

    pos_y, pos_x = game.get_player_location(player)
    number_of_boxes_to_center = abs(pos_x - math.ceil(game.width/2)) + \
        abs(pos_y - math.ceil(game.height/2)) - 1

    if percent_game_completed(0, 10, game):
        return 2*own_moves - 0.5*number_of_boxes_to_center
    elif percent_game_completed(10, 40, game):
        return float(3*own_moves - opp_moves - 0.5*number_of_boxes_to_center)
    else:
        return 2*own_moves - opp_moves
```

CUSTOM HEURISTIC #3

```
if percent_game_completed(0, 40, game):
     return float(own_moves - opp_moves - distance_to_center + quality_of_move +
     penalty)
else:
     return own_moves - 2*opp_moves
```

The intuition behind this heuristic was to penalize moves that are on the walls of the board and assess the quality of a move by the distance to the center for each of the future moves. Lesser the distance, better the quality of the move. Towards, the end, the agent aggressively tries to minimize the moves of the opponent.

Result

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

 Match #   Opponent    AB_Improved  AB_Custom_3
                        Won | Lost   Won | Lost
    1       Random      38  |   2    39  |   1
    2       MM_Open     36  |   4    28  |  12
    3      MM_Center    33  |   7    35  |   5
    4     MM_Improved   31  |   9    25  |  15
    5       AB_Open     27  |  13    12  |  28
    6      AB_Center    23  |  17    20  |  20
    7     AB_Improved   17  |  23    15  |  25
--------------------------------------------------------------------------
           Win Rate:      73.2%        62.1%
```

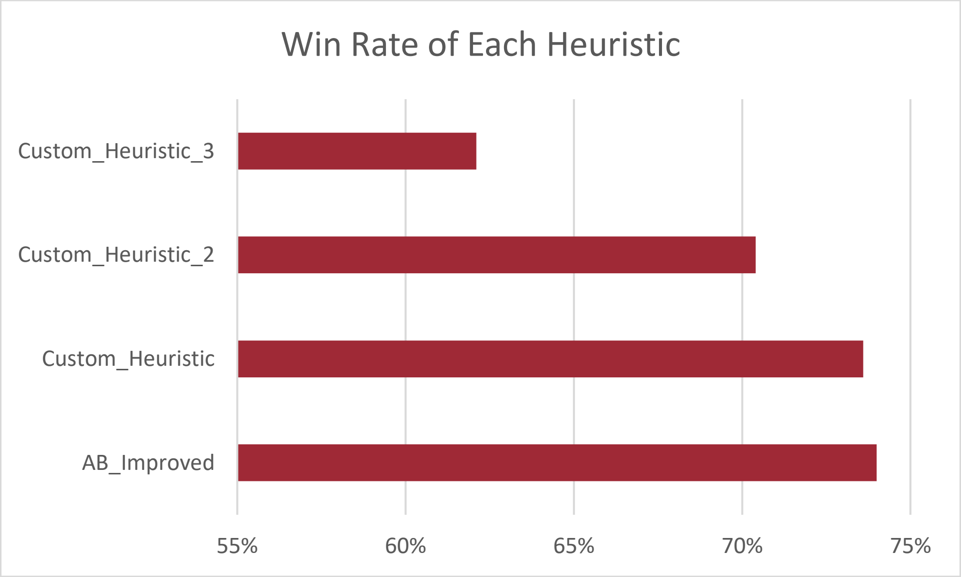
Analysis

Clearly this heuristic performs significantly worse than AB_Improved. Even though this heuristic takes multiple inputs to assess the quality of the board state, just a summation is clearly not enough. Adding weights to the inputs might help in improving the win rate. This heuristic is also more expensive to compute since we must go through future moves and compute the distance to the center for each. Again, switching strategies based on the stage in the game might be a good idea, but it is difficult to predict when the switch strategies.

Implementation

```
def custom_score_3(game, player):
    if game.is_loser(player):
        return float("-inf")
    if game.is_winner(player):
        return float("inf")

    my_moves = game.get_legal_moves(player)
    opponent_moves = game.get_legal_moves(game.get_opponent(player))
    own_moves = len(my_moves)
    opp_moves = len(opponent_moves)
    moves_so_far = 0
    for box in game._board_state:
        if box == 1:
            moves_so_far += 1

    w, h = game.get_player_location(game.get_opponent(player))
    y, x = game.get_player_location(player)
    distance_to_center = float((h - y)**2 + (w - x)**2)

    wall_boxes = [(x, y) for x in (0, game.width-1) for y in range(game.width)] +
\
    [(x, y) for y in (0, game.height-1) for x in range(game.height)]

    penalty = 0
    if (x, y) in wall_boxes:
        penalty -= 1

    quality_of_move = 0
    for move in my_moves:
        y, x = move
        dist = float((h - y)**2 + (w - x)**2)
        if dist == 0:
            quality_of_move += 1
        else:
            quality_of_move += 1/dist
        if move in opponent_moves:
            quality_of_move -= 1

    if percent_game_completed(0, 40, game):
        return float(own_moves - opp_moves - distance_to_center + quality_of_move
+ penalty)
    else:
        return own_moves - 2*opp_moves
```

OVERALL RESULTS



As you can see from the above visualization, the custom heuristic #1 and AB_Improved perform comparably. It is possible that after tuning the constant portion of the weight, custom heuristic #1 can achieve better results than AB_Improved. Custom heuristic #1 is also computationally least expensive. By using the distance to the center for the opponent and the player, it captures more information about the board state when compared to AB_Improved. Hence, I recommend this heuristic over the others. The remaining two evaluation functions have good ideas like switching strategies and penalization. But these ideas require further research before they can achieve better results than AB_Improved.