# Heuristic analysis

***Author: Alexis Alulema***

## Introduction

We saw in class that one of the better initial positions for the player is taking the central position of the board, it is supposed to offer most moves than occupying other positions in the board. Because of this it is important to consider a "betweenness centrality", which will indicate how good a move can be. In this case, the nodes will be board positions.

I'm going to try to simplify the algorithm for "betweenness centrality" (https://en.wikipedia.org/wiki/Centrality) taking the highest degree centrality in each axe, and adapt it for Isolation Game.

$$central_x, central_y = \frac{x}{2}, \frac{y}{2}$$

$$Central\ Factor = (x - central_x)^2 + (y - central_y)^2 + (move_x - central_x)^2 + (move_y + central_y)^2$$

## custom_score

```
if game.is_loser(player) or game.is_winner(player):
    return game.utility(player)

all_moves, _, opp_moves = get_moves(game, player)
center_factor = get_center_factor(game, game.get_player_location(player))

return float(len(all_moves) - len(opp_moves) + center_factor)
```

This method adds the number of moves available plus this player's "central factor" value for the potential move. It focuses on the importance on the possible move and emphasizing its importance as it is closer to the center of the board.

## custom_score_2

```
all_moves, _, opp_moves = get_moves(game, player)

if not opp_moves:
    return float("inf")
if not all_moves:
    return float("-inf")

possible_moves = len(all_moves) - len(opp_moves)
all_common = common_moves(game, player)

return float(possible_moves + sum(get_center_factor(game, m) for m in all_moves) + all_common)
```

This method considers the available moves for both the player and the opponent, a sum of the center factor of the player considering all moves, and the number of common moves. And, for balancing the result, it considers possible moves for player and opponent.

## custom_score_3

```
    all_moves, _, opp_moves = get_moves(game, player)
    intersecting_moves = opp_moves and all_moves

    if not opp_moves:
        return float("inf")
    if not all_moves:
        return float("-inf")

    factor = 1 / (game.move_count + 1)
    inverse_factor = 1 / factor
    return float(len(intersecting_moves) * factor + inverse_factor * len(all_moves))
```

This method considers the possible moves for both player and opponent, but experimenting with the concept of variable importance of the parameters as the game progresses. Towards the end of the game, a player is expected to have 2 or 3 moves available at best, so the number of moves available should decrease in importance, while the number of common moves should increase in importance.

***Results with custom_score***

| Match # | Opponent | AB_Improved | | AB_Custom | | AB_Custom_2 | | AB_Custom_3 | |
|---|---|---|---|---|---|---|---|---|---|
| | | Won | Lost | Won | Lost | Won | Lost | Won | Lost |
| 1 | Random | 6 | 4 | 9 | 1 | 8 | 2 | 8 | 2 |
| 2 | MM_Open | 5 | 5 | 5 | 5 | 3 | 7 | 3 | 7 |
| 3 | MM_Center | 7 | 3 | 8 | 2 | 7 | 3 | 5 | 5 |
| 4 | MM_Improved | 4 | 6 | 5 | 5 | 3 | 7 | 4 | 6 |
| 5 | AB_Open | 4 | 6 | 3 | 7 | 3 | 7 | 4 | 6 |
| 6 | AB_Center | 5 | 5 | 5 | 5 | 3 | 7 | 4 | 6 |
| 7 | AB_Improved | 7 | 3 | 9 | 1 | 3 | 7 | 3 | 7 |
| | Win Rate: | 54.3% | | 62.9% | | 42.9% | | 44.3% | |

***Results with custom_score_2***

| Match # | Opponent | AB_Improved | | AB_Custom | | AB_Custom_2 | | AB_Custom_3 | |
|---|---|---|---|---|---|---|---|---|---|
| | | Won | Lost | Won | Lost | Won | Lost | Won | Lost |
| 1 | Random | 10 | 0 | 9 | 1 | 10 | 0 | 8 | 2 |
| 2 | MM_Open | 6 | 4 | 6 | 4 | 6 | 4 | 5 | 5 |
| 3 | MM_Center | 7 | 3 | 9 | 1 | 6 | 4 | 6 | 4 |
| 4 | MM_Improved | 6 | 4 | 6 | 4 | 4 | 6 | 5 | 5 |
| 5 | AB_Open | 6 | 4 | 5 | 5 | 4 | 6 | 6 | 4 |
| 6 | AB_Center | 3 | 7 | 5 | 5 | 6 | 4 | 3 | 7 |
| 7 | AB_Improved | 4 | 6 | 4 | 6 | 4 | 6 | 4 | 6 |
| | **Win Rate:** | **60.0%** | | **62.9%** | | **57.1%** | | **52.9%** | |

***Results with custom_score_3***

| 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 | 8 | 2 | 9 | 1 |
| 2 | MM_Open | 7 | 3 | 7 | 3 | 3 | 7 | 5 | 5 |
| 3 | MM_Center | 7 | 3 | 6 | 4 | 5 | 5 | 7 | 3 |
| 4 | MM_Improved | 3 | 7 | 7 | 3 | 5 | 5 | 1 | 9 |
| 5 | AB_Open | 7 | 3 | 5 | 5 | 5 | 5 | 2 | 8 |
| 6 | AB_Center | 3 | 7 | 4 | 6 | 6 | 4 | 2 | 8 |
| 7 | AB_Improved | 5 | 5 | 5 | 5 | 1 | 9 | 3 | 7 |
| | Win Rate: | 57.1% | | 61.4% | | 47.1% | | 41.4% | |

## Conclusion

As we can see, we obtain the better results by using custom_score_2, and it makes sense because this method considers a better-balanced point of view, taking in account player's and opponent's moves. However, this improvement comes with a cost in performance, as this method is more consuming time than the other two methods.