Heuristic Analysis – Luca Fiaschi

For this project, I decided to focus on trying to beat the **AB_Improved** agent which was given as a template. This agent uses a scoring function which counts the number of moves available and subtract the number of opponent moves available: hence it tries to balance between an aggressive and defensive behavior.

My heuristics were a variation of the baseline AB_improved scoring.

The first heuristic adds a centrality term to the improved score. It is a **defensive heuristic** as it tries to keep the player as close to the center of the board (where more moves are available) as possible

```
# Implement number of my moves - opponent moves plus a centrality term that
# tries to keep the player at the centre of the board

return improved_score(game, player) + center_score(game, player)
```

The second heuristic is an **aggressive player:**

```
# (My moves - 2 Opponent moves) * run towards the opponent ; aggressive player

position_of_player = game.get_player_location(player)
position_of_opponent = game.get_player_location(game.get_opponent(player))
manhattan_distance = abs(position_of_player[0] - position_of_opponent[0]) + abs(
    position_of_player[1] - position_of_opponent[1])

my_moves = float(len(game.get_legal_moves(player)))
opponent_moves = float(len(game.get_legal_moves(game.get_opponent(player))))

return (my_moves - 2 * opponent_moves) / manhattan_distance + game.utility(player)
```

Here not only the negative weight of opponent moves is higher than in the improved score, but also that term is multiplied by the inverse of the distance between the two players. It simulates a player tries who tries to run towards the opponent and putting her in a situation where she has no more moves available

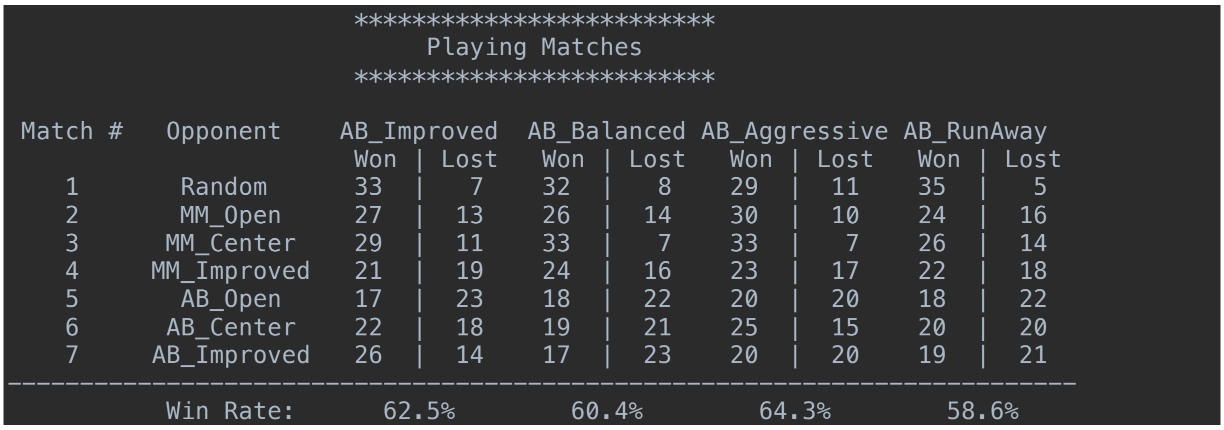
The last heuristic is a **defensive player**:

```
# Run away from the opponent (defensive player) keeps as far away as possible
# while maintaining the number of moves available high

position_of_player = game.get_player_location(player)
position_of_opponent = game.get_player_location(game.get_opponent(player))
manhattan_distance = abs(position_of_player[0] - position_of_opponent[0]) + abs(
    position_of_player[1] - position_of_opponent[1])

my_moves = float(len(game.get_legal_moves(player)))

return my_moves + manhattan_distance + game.utility(player)
```

She tries to keep the number of open moves as large as possible while at the same time keeping the player as far as possible from the opponent.

Results

Playing the tournament resulted in the following results:



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

 Match #   Opponent    AB_Improved  AB_Balanced AB_Aggressive AB_RunAway
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      33  |   7    32  |   8    29  |  11    35  |   5
    2       MM_Open     27  |  13    26  |  14    30  |  10    24  |  16
    3      MM_Center    29  |  11    33  |   7    33  |   7    26  |  14
    4     MM_Improved   21  |  19    24  |  16    23  |  17    22  |  18
    5       AB_Open     17  |  23    18  |  22    20  |  20    18  |  22
    6      AB_Center    22  |  18    19  |  21    25  |  15    20  |  20
    7     AB_Improved   26  |  14    17  |  23    20  |  20    19  |  21
--------------------------------------------------------------------------
          Win Rate:      62.5%        60.4%        64.3%        58.6%
```

I believe the actual rates should be taken with a pinch of salt, indeed the variability associated with the randomness in the game opening makes the analysis more difficult. As an example note that the **AB_improved** player, playing against herself seems to have a winning rate of 65%, while in theory she should result in a 50% winning rate. This fact suggests that few percent difference in winning rate probability are not significant, and that more accurate analysis should use a statistical model of uncertainty for rates such as the beta binomial.

That said, qualitatively we can see that the 4 heuristics (AB_Improved, AB_Balanced, AB_Aggressive, AB_RunAway) perform similarly with the AB players clearly outperforming Minimax players thanks to a faster exploration of the search tree.

On average, the aggressive strategy seems to perform slightly better than the others as it wins more games against a wider range of different players.