**Heuristic Analysis**

**AIND-Isolation**

# Heuristic 1

**Improved Score**

**own\_moves** = Moves of first player  
**opp\_moves** = Moves of opponent 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)))*

*return float(own\_moves - opp\_moves)*

This is the heuristic given in the lectures. It outputs "a score equal to the difference in the number of moves available to the two players". As long as the number returned is positive, we will have a better chance of winning.

**Heuristic 2**

**Custom Score 1**

**opponent** = opponent player  
**my\_moves** = moves of the first player  
**opponent\_moves** = moves of the opponent player

*opponent = game.get\_opponent(player)*

*my\_moves = len(game.get\_legal\_moves(player))*

*opponent\_moves = len(game.get\_legal\_moves(opponent))*

*h1 = float(my\_moves - 2\*opponent\_moves)*

This heuristic is the one mentioned in the lectures. It is meant to be a more aggressive strategy, by giving more weight to the opponent moves, so we are always chasing the opponent.

# Heuristic 3

**Custom Score 2**

**opponent** = opponent player  
**my\_moves** = moves of the first player  
**opponent\_moves** = moves of the opponent player  
**agg\_factor** = factor that will help us lower or increase aggressiveness

*opponent = game.get\_opponent(player)*

*my\_moves = len(game.get\_legal\_moves(player))*

*opponent\_moves = len(game.get\_legal\_moves(opponent))*

*agg\_factor = 1*

*if my\_moves-opponent\_moves<=0:*

*agg\_factor = 3*

*else:*

*agg\_factor = 1*

*h2 = float(my\_moves - agg\_factor\*opponent\_moves)*

This heuristic is meant to balance aggressiveness. If we are loosing, we will be more aggressive by incrementing opponent moves by 3. If we are winning, we will keep playing normally

# Heuristic 4

**Custom Score 3**

**opponent** = opponent player  
**my\_moves** = moves of the first player  
**opponent\_moves** = moves of the opponent player  
**height** = center position  
**width** = center position  
**posx** = player location on x axis  
**posy** = player location on y axis  
**center** = center of the first player  
**center\_opp** = center of the opponent player

*opponent = game.get\_opponent(player)*

*height = game.height/2*

*width = game.width/2*

*posx, posy = game.get\_player\_location(player)*

*posx\_o, posy\_o = game.get\_player\_location(opponent)*

*center = abs(height-posy)+abs(width-posx)*

*center\_opp = abs(height-posy\_o)+abs(width-posx\_o)*

*my\_moves = len(game.get\_legal\_moves(player))*

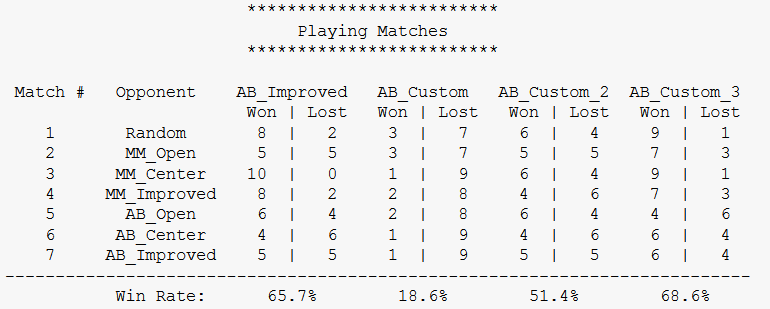
*opponent\_moves = len(game.get\_legal\_moves(opponent))*

*h3 = float((center\*my\_moves)-(center\_opp\*opponent\_moves))*

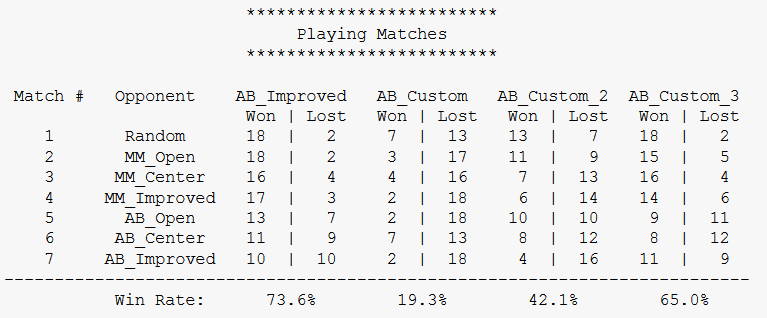
This heuristic is taking into account how far each of the players are from the center. If we are near the center, we will play aggressively

# Results

First run:



Second run:



Comparing each of the custom heuristics to the AB\_Improved, we got:

- Being aggressive the whole time doesn't benefit us at all, we lose most of the time.

- Being aggressive just in certain parts of the game, is a better approach, but it still doesn't get us to the result of just calculating remaining moves from AB\_Improved

- The third approach, and the one recommended, is the one that takes into account how far are the players in the board, playing near the center get us a much better result, as shown by the tests, we got close to the AB\_Improved and even got a better result in the first run.