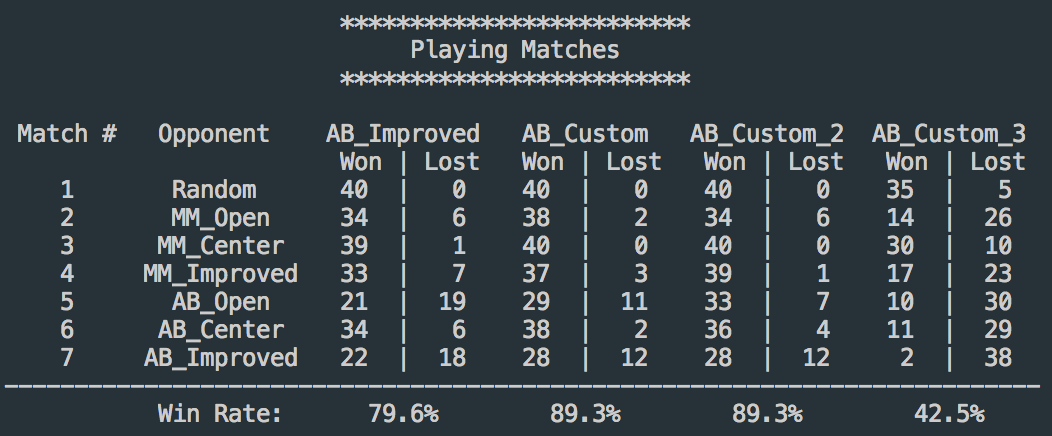
Isolation Heuristic Analysis

Evaluation functions are in integral part to any AI agent. My intent is to keep them fairly simple without adding any unnecessary complexity for complexity’s sake.

During my trials I tried to establish a control metric that would return a random valuation for a board state. I wanted to make certain my measures were better than random. Initially I wasn’t getting the results I had expected. My random function which would just return random.random(), was winning a majority of its games. It wasn’t until someone on the slack channel had suggested I was reaching end-game, so It didn’t matter what the valuation function returned as the is\_winner/is\_loser methods were being called.

For these evaluations I bumped the board size to 13 x 13 to give my valuation functions some playing time, along with increasing the matches count to provide a more continuous calculation to the win rate and was rather surprised as my previous best heuristic turned out to be inferior to my simplest one.

These runs take a considerable amount of time so for brevity I ran this extended run once. For the submission I will be swapping the AB\_Custom & AB\_Custom\_2 functions.



Worthy of noting is I do have some ideas on how to improve my win rates, at this juncture it’s time to call it good enough for the project as there was a lot of time spent wading through the weeds to get it even working.

AB\_Custom: 71.4% Win rate

def custom\_score(game, player):

if game.is\_loser(player):

return float("-inf")

if game.is\_winner(player):

return float("inf")

c = int(game.width / 2) + 1

board\_center = (c, c)

pl\_moves = len(game.get\_legal\_moves(player))

op\_moves = len(game.get\_legal\_moves(game.get\_opponent(player)))

avail\_moves = game.get\_legal\_moves(player)

total\_moves = len(avail\_moves)

total\_distance\_from\_ctr = 0

for move in avail\_moves:

mx, my = move

cx, cy = board\_center[0], board\_center[1]

total\_distance\_from\_ctr += np.sqrt((mx-cx)\*\*2 + (my-cy)\*\*2)

# I want to penalize board states that have a lot of moves around the perimeter

# So I'm multiplying the inverse of the mean distance to the center by the number

# of moves we have.

if total\_moves > 0 and total\_distance\_from\_ctr > 0:

return (1. / float(total\_distance\_from\_ctr / total\_moves)) \* float(pl\_moves - op\_moves)

else:

return float("-inf")

This one didn’t perform as good as I had hoped. I was trying a two-pronged approach to this one. I wanted to score a board states based on not only the number of moves my agent has available but want to weight the states that have center moves more favorably.  
  
It works better than just looking at center moves entirely, but not as well as just the total number of moves in my customer\_score\_2.

I don’t have the coding ability (yet) or time to test this; however, I hypothesize this function is discounting those game states where my agent can possibly win but most of the center moves have been taken.

AB\_Custom\_2: 79.3% Win rate

def custom\_score\_2(game, player):

if game.is\_loser(player):

return float("-inf")

if game.is\_winner(player):

return float("inf")

pl\_moves = len(game.get\_legal\_moves(player))

op\_moves = len(game.get\_legal\_moves(game.get\_opponent(player)))

return float(pl\_moves - op\_moves)

This performed the best out of the functions I put together and tested myself. It performs better than my expected best metric against the AB\_Center sample player. Because of its simplicity, it’s not under-valuing some scenarios.

AB\_Custom\_3 Win Rate 45%

def custom\_score\_3(game, player):

if game.is\_loser(player):

return float("-inf")

if game.is\_winner(player):

return float("inf")

c = int(game.width / 2) + 1

board\_center = (c, c)

avail\_moves = game.get\_legal\_moves(player)

total\_moves = len(avail\_moves)

total\_distance\_from\_ctr = 0

for move in avail\_moves:

mx, my = move

cx, cy = board\_center[0], board\_center[1]

total\_distance\_from\_ctr += np.sqrt((mx-cx)\*\*2 + (my-cy)\*\*2)

if total\_moves > 0 and total\_distance\_from\_ctr > 0:

return float(total\_distance\_from\_ctr / total\_moves)

else:

return float("-inf")

For the third one I wanted to see if a heuristic that doesn’t use a variant on total moves would be viable. Unfortunately, this one suffers the same short fall as my first custom\_score.