Heuristic Analysis

Eduardo Carrasco Jr

My first approach to the analysis was to try and work it out on paper for a few moves that a player could make. This was not an easy task but it helped me to further my understanding on how heuristic would play a role in the sample_player.py After dissecting how the sample_player.py worked, I had descent understanding about how to approach the heuristic.

As far as my understanding of chess is concerned, whoever controls the center of the board is most likely controlling the flow of the game. With that in mind there are methods two possible stances a player can do. Either an aggressive offensive position or an aggressive defense position. The three created heuristic functions were focused on a combination of distance from center and how many moves the player has. This is where I decided to create a function called centrality which calculates a player's distance to the center. Similar to the Manhattan distance.

The following three heuristics were tested and used to play the chess game.

Heuristic 1

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

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

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

# Use the distant formula to calculate the distance of the current location from the center
# of both the player and opponent
my_distance_from_center = centrality(game, player)
opponent_distance_from_center = centrality(game, game.get_opponent(player))

# return the difference in distance from the center plus the difference of number of moves
# available for each player
return float(my_distance_from_center - opponent_distance_from_center + (my_moves - opponent_moves))
```

In this heuristic method I calculate the distance from the center of the board and the closer the player is to the center it means they control more of the board space, therefore have a higher score value.

Heuristic Analysis

Eduardo Carrasco Jr

Heuristic 2

.

```
if game.is_loser(player):
    return float('-inf')
if game.is_winner(player):
    return float('inf')

# Grab list of moves available for player and opponent
my_moves = game.get_legal_moves(player)
opponent_moves = game.get_legal_moves(game.get_opponent(player))

# check how many moves they currently share in common as possible moves
# iterate through opponents moves and check if they are in player's moves
# if it is then increment counter by 1
similar_moves = 0
for move in opponent_moves:
    if move in my_moves:
        similar_moves += 1

# return difference of moves available plus the number of moves they share as possible moves
return float(len(my_moves) - len(opponent_moves) + similar_moves)
```

In this heuristic method I try to find the similar moves each player has and compute a difference to see how unique each players move set was. The more options a player has the more likely they are dominating the board and winning

Heuristic 3

```
play_moves = game.get_legal_moves(player)
opp_moves = game.get_legal_moves(game.get_opponent(player))

common_moves = opp_moves and play_moves

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

factor = 1.0 / (game.move_count + 1)

inv_factor = 1 / factor

return float(len(common_moves) * factor + inv_factor * len(game.get_legal_moves()))
```

Heuristic Analysis

Eduardo Carrasco Jr

In the last heuristic was a mix scoring of defense and offence that the player was using against the opponent. As the game comes to a closes the player has very little move set to work with their score goes down but rises as they play more better.

Below are my results for running tournament.py

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

Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                      Won | Lost   Won | Lost   Won | Lost   Won | Lost
   1       Random      9  |  1     10  |  0     7  |  3     7  |  3
   2       MM_Open     6  |  4      6  |  4     6  |  4     6  |  4
   3      MM_Center    7  |  3      6  |  4     8  |  2     8  |  2
   4     MM_Improved   6  |  4      7  |  3     5  |  5     4  |  6
   5       AB_Open     8  |  2      8  |  2     5  |  5     3  |  7
   6      AB_Center    4  |  6      5  |  5     6  |  4     5  |  5
   7     AB_Improved   5  |  5      2  |  8     4  |  6     5  |  5
--------------------------------------------------------------------
          Win Rate:      64.3%        62.9%        58.6%        54.3%
```

It appears that the last two customs scores didn't play drastically well, but the AB improved score was the best. The first custom score from the center was a second runner up for best score. The slowest computations were by far the last two scoring.

Overall I would recommend the AB_Custom. The score did the best overall in term of winning rate. The computation time is faster because it has less operations to do then the other two. The depth search also seems to the faster on average as well. The AB_Custom was also the easiest metric to come up with and implement. So my best recommendation for the best heuristic is AB_Custom.