Heuristic Analysis

Introduction

I evaluated the performance of my agent using three custom-implemented evaluation functions. These were tested alongside a given 'improved_score' evaluation function which returned the difference between the number of moves available to the current player vs the number of moves available to the opponent.

These functions were implemented as follows: custom_score implemented a "Weighted Linear Combination" heuristic; custom_score_2 implemented a "Nearness to Center" heuristic; and custom_score_3 implemented a "Nearness to Opponent" heuristic

Overall Results of tournament.py

tournament.py returned each of the following results:

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

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random       7  |   3     6  |   4     8  |   2     9  |   1
    2       MM_Open      4  |   6     5  |   5     5  |   5     3  |   7
    3      MM_Center     4  |   6     6  |   4     5  |   5     5  |   5
    4     MM_Improved    4  |   6     5  |   5     4  |   6     1  |   9
    5       AB_Open      7  |   3     3  |   7     6  |   4     6  |   4
    6      AB_Center     4  |   6     6  |   4     2  |   8     4  |   6
    7     AB_Improved    5  |   5     6  |   4     4  |   6     5  |   5
--------------------------------------------------------------------------
           Win Rate:      50.0%        52.9%        48.6%        47.1%
```

From this output, we consider the results of match #7, which was against the most sophisticated opponent. Here we can see that our custom_score or "Weighted Linear Combination" heuristic performed best, followed by custom_3, or "Nearness to Opponent", and then followed by custom_2, or "Nearness to Center"

Summary of "Weighted Linear Combination" heuristic

A visualization of the performance of this heuristic is given in the "Overall Results" above. It appears to have outperformed the "improved" heuristic because it gave a heigher weight to limiting the moves of the opponents vs

keeping "own" moves count large. This may have incentivized the agent to find a partition earlier in the search, helping to explain its improved performance

Summary of "Nearness to Center" heuristic

A visualization of the performance of this heuristic is given in the "Overall Results" table above. It appears to have underperformed the other heuristics. One possible reason for this could be that its not optimal to try to keep moves open in the center of the board if the opponent can't be fooled into creating a partition. This may explain why this heuristic (AB_Custom_2 in the table above) performed relatively well against a random agent but poorly against the "AB_Improved" agent.

Summary of "Nearness to Opponent" heuristic

This heuristic rewarded an agent for placing himself nearer to the opponent. It is thought that this heuristic may have rewarded creating a partition, and it appears to have down so. However, without the intelligence of scoring the number of positions that remained available to the opponent after "moving towards" him, it merely drew a tie against the "AB_Improved" agent. It appears this heuristic may have similarity to the "improved" heuristic, seen by the lack of any meaningful improvement vs this heuristic.

Recommended Heuristic

My recommended heuristic is the Weighted Linear Combination Heuristic.

I can see three distinct reasons to support the choice of using the weighted linear combination heuristic:

1) It is relatively simple to implement, meaning that others can modify the code in the future without needing a great deal of training or introducing a great risk of breakage. This is not the case in the other two heuristics, which rely on a more subtle "sum of squared errors" approach for evaluation.

2) It is an heuritstic which would support a grid-based search (using e.g., scikit-learn) to find an optimal coefficients, i.e., it can take the form ax + by + z, and each of a, b, and z coefficients can be determined via grid search. It has only been shown empirically that -1.5 is a good value for the b coefficient, but it is likely that better choices exist

3) It allows us to find partitions early, especially those that "box opponents" into a corner due to its tendency to prefer moves that significantly decrease that ratio of opponent moves to own available moves. This is posited to help the algorithm reach an endgame earlier in simulation, allowing for an

increased efficiency in timed iterative deepening searches compared to the other heuristics.