Heuristic Analysis

- Atul Acharya

The aim of the project is to create a game playing agent that plays an adversarial game "Isolation". The agent incorporates well known algorithms — such as MiniMax — to determine its moves. To efficiently determine moves within time constraints, modifications such as Alphabeta Pruning (AB) and Iterative Deepening (ID) are implemented. The agent's ability to beat opponents depends on the choice of heuristics at each search level.

The submission consists of the following:

1. Game-tree searching: Minimax and Alphabeta algorithms — implemented in *game_agent.py*
2. At least three different heuristics implemented in functions *custom_score*(), *custom_score2*() and *custom_score3*() in *game_agent.py*
3. Heuristic analysis (this report)
4. AI game-playing research report

Submissions and discussion

1. Game-tree Search

Successfully implemented MiniMax (MM) and AlphaBeta (AB) iterative deepening (ID) algorithms. All tests successfully pass.

2. Heuristic Analysis

Apart from the evaluator provided heuristics (null_score(), open_move_score(), improved_score()), I implemented four heuristic functions.

favor_center_heuristic() — This heuristic favors center squares more than edges. It is calculated as the sum of absolute positions from center, for both the player and the opponent. It puts a negative coefficient on distance from center. *This heuristic yielded a win rate of 68% - 74% in tournament results, and was consistently higher than other heuristics.* The relatively high success rate is likely because center positions yield more room for player to maneuver and hence yield higher winning moves.

> score = α *(len(player moves) - len(opponent moves)) - abs(player_distance_to_center) + abs(opponent_distance_to_center), where* $\alpha \in [1, 10]$

Here α was empirically chosen to be 5.

favor_edges_heuristic() — This heuristic favors edge positions more. It is the opposite of favor_center() and puts a positive coefficient on distance from the center. *This yielded a win rate of 61% - 67%* - which is a lower rate compared to other heuristics, and was thus discarded. The low win rate is because the edge positions tend to get boxed out leading to losses.

> *score =* α *(len(player moves) - len(opponent moves)) + abs(player_distance_to_center) - abs(opponent_distance_to_center), where* $\alpha \in [1, 10]$

Here α was empirically chosen to be 5.

weighted_moves_ratio() — This heuristic is calculated as a ratio of player's moves to opponent moves, and multiplied by factor of 1.5, thus favoring player's moves more. *This yielded a win rate of 67% - 73% in different tournament runs.*

$$score = \alpha \cdot len(player\ moves) / len(opponent\ moves),\ where\ \alpha \in [1, 10]$$

Empirically, α was chosen to be 1.5.

bonus_improved_score() — This heuristic is a small variation from the canned improved_score() heuristic; it subtracts a certain bonus term (equal to the distance of player from center) from the improved_score. *It yielded roughly 63% - 72% win rate*, roughly in line with the canned heuristic. The variations in results are likely due to chance.

$$score = len(player\ moves) - len(opponent\ moves) - player\_distance\_to\_center$$

weighted_difference_heuristic() — This heuristic favors a stronger difference between player moves and opponent moves, and is another variation on improved_score().

$$score = \alpha \cdot len(player\ moves)\ **\ 2\ - len(opponent\ moves)\ **\ 2$$

Empirically, α was chosen to be 1.5
This yielded a win rate of 65% - 74%, similar in line with other heuristics so far.

Tournament Results

```
This script evaluates the performance of the custom_score evaluation
function against a baseline agent using alpha-beta search and iterative
deepening (ID) called `AB_Improved`. The three `AB_Custom` agents use
ID and alpha-beta search with the custom_score functions defined in
game_agent.py.

                    ************************
                         Playing Matches
                    ************************

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3  AB_Custom_4
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      19  |   1    20  |   0    20  |   0    19  |   1    19  |   1
    2       MM_Open     15  |   5    17  |   3    13  |   7    13  |   7    17  |   3
    3      MM_Center    18  |   2    18  |   2    17  |   3    16  |   4    16  |   4
    4     MM_Improved   13  |   7    15  |   5    14  |   6    18  |   2    13  |   7
    5       AB_Open      9  |  11     9  |  11     8  |  12    12  |   8    10  |  10
    6      AB_Center    10  |  10    10  |  10    11  |   9    13  |   7     8  |  12
    7     AB_Improved    9  |  11    14  |   6    10  |  10    10  |  10    12  |   8
--------------------------------------------------------------------------
          Win Rate:      66.4%        73.6%        66.4%        72.1%        67.9%

real    34m49.590s
user    34m40.698s
sys     0m4.359s
```

Some of the results from the tests are shown here.

The AB_Improved agent win rates were approximately 63%-72%. In general, the AB_Improved agent is rather hard to beat every time.

The best performing heuristic so far is the **favor_center_heuristic**() implemented in **custom_score_3()**. *It yields a win rate of 68-74%*, and its success can be attributed to the following:

a) It depends only on the current state and does not store the history (memory)
b) The computation complexity is approximately the same as that for improved_score()
c) It is easy to implement with just a few more operations
d) It aligns with the intuition that player with more moves towards the center has higher chances of winning (and will not get boxed out)

For these reasons, I would choose **favor_center_heuristic**() as the heuristic