April 2020

Udacity
Artificial Intelligence
Nanodegree

Project 3:

Build an Adversarial Game Playing Agent

Akshit Sarin

Introduction:

In this project, I have chosen Option 1: Developing a custom heuristic. The performance / effectiveness of the custom heuristic has also been evaluated by comparing it with a baseline heuristic. I have also combined minimax search with alpha-beta pruning and iterative deepening as explained in the lectures.

Custom Heuristic:

The final custom heuristic is actually a combination of the baseline heuristic and another custom heuristic. They are explained as follows:

- ***Baseline Heuristic*:** This heuristic is based upon a simple formula, which is - *the difference between 'our moves left' and 'opponent moves left'.*

```
def baseline(self, state):
  return len(state.liberties(state.locs[self.player_id])) - \
         len(state.liberties(state.locs[1-self.player_id]))
```

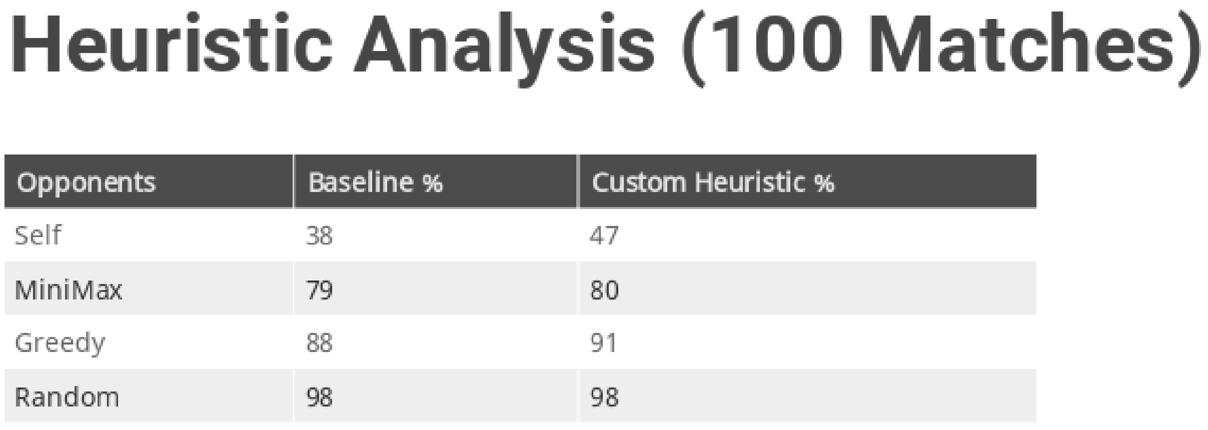
- ***Central Heuristic (intermediate)*:** This heuristic takes into account the sum of straight line (euclidean) distance between current locations of each player and the center-most cell of the board [5, 4]. This is based on the assumption that as closer to the center of the board the player is, the more are their chances of winning, since they have much more free cells to explore in more directions, as opposed to edges, where one side is completely blocked off.

```
def central(self, state):
  # get positions of player_1 and player_2
  pos_1 = state.locs[self.player_id]
  pos_2 = state.locs[1-self.player_id]

  x1, y1 = (pos_1%(11+2), pos_1//(9+2))
  x2, y2 = (pos_2%(11+2), pos_2//(9+2))

      # center coordinates: [(11-1)/2, (9-1)/2] = [5, 4]
 return - (x1-5)**2 - (y1-4)**2 + (x2-5)**2 + (y2-4)**2
```

- ***Combined Heuristic (final heuristic)*****:** This heuristic takes into account both the above mentioned heuristics, by assigning certain weights to them individually and adding them up together.
  The value obtained from baseline heuristic is multiplied by 0.5, and the value obtained by the central heuristic is multiplied by 1.5, and they are added up together to get a new heuristic. This heuristic will be in a sense better than the previous 2 since it derives its values from and is dependent on both of them.
  The values 0.5 and 1.5 were chosen by initially selecting them as 1 and then gradually changing them randomly to check which values give a higher success rate.

```
def combined(self, state):
 return (0.5 * self.baseline(state)) + (1.5 * self.central(state))
```

The chart analysing the heuristic performance is as follows:



Heuristic Analysis (100 Matches)

Opponents	Baseline %	Custom Heuristic %
Self	38	47
MiniMax	79	80
Greedy	88	91
Random	98	98

Questions:

1. ***What features of the game does your heuristic incorporate, and why do you think those features matter in evaluating states during search?***

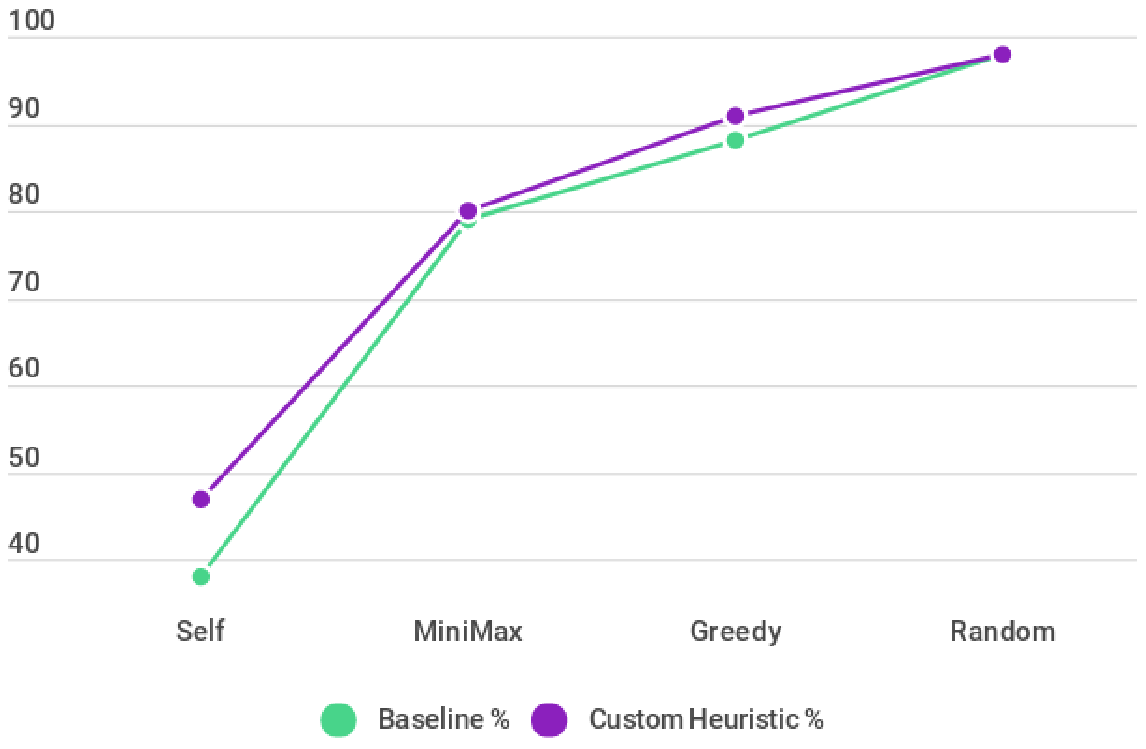
   The heuristic incorporates liberties available to each player at every individual turn, as well as their distance from the center-most cell of the board.

   To calculate the value of the final combined heuristic, we multiply each of the sub-heuristics by a constant value (0.5 and 1.5 respectively), to increase its performance. These features matter in a combined fashion - the *baseline heuristic*, by reducing the number of available liberties at each iteration, and *central heuristic* by giving the sum of straight line (euclidean) distance between both cells (player's and opponent's) to the center of the board, since the players have much more open cells to explore from the center, as opposed to the edges, where one side is completely blocked off. Their combination (final heuristic) is even more sophisticated, since it takes into account both their calculated values.

2. ***Analyze the search depth your agent achieves using your custom heuristic. Does search speed matter more or less than accuracy to the performance of your heuristic?***

   Upon analysing the depth of the agent, I found out that the maximum depth achieved was 18, and the minimum being 8.

   Upon reducing the time limit to 100ms, the agent's success rate was the same as compared to 150ms - 90%, running for greedy with fair_matches enabled for 20 games. But upon increasing it to 500ms, the success rate increased to 95%. Which, according to me, implies that the accuracy of the heuristic is dependent on it's speed as well as time limits specified. Since the heuristic has given such higher success rates, I also tried to increase the number of games to 100, to check the new success rates. The graph analysing the same is as follows:



It can be clearly observed that the custom heuristic, which is a combination of baseline and an intermediate custom heuristic, outperforms the baseline heuristic.