Margery Carlos
Udacity: Artificial Intelligence Nanodegree
Building an Adversarial Game Playing Agent

Isolation Game Playing Report

Data

Heuristic:	**Baseline**	**Custom 1**	**Custom 2**	**Custom 3**	**Custom 4.1**	**Custom 4.2**	**Custom 5**
Test Agent:					*Weights*: lib_weight=0.5, center_weight=1.5	*Weights*: lib_weight=1.5, center_weight=0.5	*Weights*: lib_weight=1.5, center_weight=0.5
Greedy	65.0%	50.0%	72.5%	80.0%	62.5%	85.0%	80.0%
Random	95.0%	85.0%	90.0%	90.0%	90.0%	90.0%	92.5%
Minimax	47.5%	27.5%	57.5%	42.5%	62.5%	42.5%	57.5%
Self	47.5%	45.0%	40.0%	57.5%	52.5%	57.5%	65.0%

Commands Used to Gather Data:

```
$ python run_match.py -f -r 10 -o GREEDY -p 4
$ python run_match.py -f -r 10 -o RANDOM -p 4
$ python run_match.py -f -r 10 -o MINIMAX -p 4
$ python run_match.py -f -r 10 -o SELF -p 4
```

General Summary

I built five different custom heuristics aside from the baseline for assigning values to the next move. They all use minimax with alpha-beta pruning. The baseline heuristic is just a simple difference of the player's possible moves at that spot and the opponent's possible moves. The first custom heuristic only calculates the player's distance to the center. The second custom heuristic takes the player's possible moves and subtracts a weighted player distance to the center. The third custom heuristic subtracts the opponent's moves from the player's moves but also subtracts the weighted distance of that spot from the center. The fourth custom heuristic weights the opponent's moves and weights the player's distance to center. I tried giving more weight to the opponent's moves, then tried giving more weight to the player's distance to the center, but the heuristic was still not doing better than the baseline against the game agents. Finally, I decided to take a ratio of opponent and player possible moves instead of the difference of the two. I weighted that ratio and also subtracted a weighted player distance to the center, and the agent did better than the baseline agent against every agent. Custom 5 is my final heuristic that I use for analysis below.

Heuristics

Baseline: own_loc – opp_loc
Custom 1: distanceToCenter(own_loc)

Custom 2: len(own_liberties) - (center_weight * distanceToCenter(own_loc))
Custom 3: len(own_liberties) - (len(opp_liberties)) - (center_weight * distanceToCenter(own_loc))
Custom 4: len(own_liberties) - (lib_weight * len(opp_liberties)) - (center_weight * distanceToCenter(own_loc))
4.1 Weights: lib_weight=0.5, lib_weight=1.5
4.2 Weights: lib_weight=1.5, lib_weight=0.5
Custom 5: (lib_weight * (len(own_liberties) / len(opp_liberties)) - (center_weight * distanceToCenter(own_loc)))
5 Weights: lib_weight=1.5, lib_weight=0.5

Questions

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

My heuristic takes into account the moves available to both the player and the opponent, finding a weighted ratio between the two. It also calculates the distance of the player to the center, opting for moves that stay closer to the middle square. I had researched for other heuristics playing Isolation and it seemed one good tactic was to pick moves closer to the center because the opponent can more easily block the player in near the edges of the board. Thus, I used the player's distance to the center as part of my heuristic. In all the heuristics I read about, it was a good idea to play both offensive by tracking the player's available moves and play defensive by tracking the opponent's available moves. Thus, I used the ratio between them both to get a balanced value that would rate how both defensive and offensive a move would be together. I tried a lot of different custom heuristics, trying to balance the player's vs the opponent's moves, but the best one seemed to be the weighted ratio between them with the less weighted value of the player's distance to center.

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?*

My game playing agent uses a search depth of 3. With a depth less than 3, it couldn't find the most accurate moves, and with a depth more than 3 or 4 it did not have time to return optimal moves before the search time was up. The accuracy of the heuristic is probably more important when there are default time constraints, like in the beginning and middle of the game. As the number of free spaces on the board decrease, the search speed probably matters more because the accuracy of the heuristic is going to have less effect. With less spaces to choose from, there is more room to randomly pick the same optimal move as an accurate heuristic.