Udacity Artificial Intelligence Nanodegree Assignment 2

Game Playing Agent Evaluation Heuristic Analysis

Richard C. Stilborn

Introduction

This section of the nanodegree was about game playing agents and the different search strategies and heuristics that can be used to find the optimal move for any given scenario. The chosen game for the examples and the assignment was Isolation (see wikipedia). In the variant of the game we implemented pieces could make L-shaped moves like knights in chess. For the assignment we had to:

1. Implement minimax search, alpha beta pruning, and iterative deepening.
2. Implement and evaluate 3 different evaluation heuristics.

Benchmarking

In order to evaluate the heuristics we were provided with a comparison heuristic called "ID_Improved" and a set of opponent agents that implemented different search strategies as well as different heuristics. Tournaments were run with our heuristic playing each opponent and the benchmark heuristic playing each opponent. Each player took turns starting and every game was initialized with one random move for each player.

Features of evaluation function

The features I chose to investigate as part of the evaluation function were:

1. Number of moves available to self
2. Number of moves available to the opponent
3. Distance from center

I also decided to split the game into first half and second half as it seemed reasonable to refine strategies as the board filled up.

I wrote three heuristic functions and started to test them against the provided benchmark function called "ID_Improved". I quickly realized that there were many different combinations of the features that might prove successful and that a more structured data driven approach would be better.

Parameterized version

In order to test many different combinations I wrote the class ParameterizedEvaluationFunction that uses six provided weights. The evaluation function is coded as:
In the first half of the game return:

(w_0 * my mobility) - (w_1 * opponent mobility) - (w_2 * Euclidian distance from center)

In the second half of the game return:

(w_3 * my mobility) - (w_4 * opponent mobility) - (w_5 * Euclidian distance from center)

My first plan was to evaluate the following ranges:

w_0	0 .. 3
w_1	0 .. 3
w_2	-2 .. 2
w_3	0 .. 3
w_4	0 .. 3
w_6	-2 .. 2

However that would have been 6400 different combinations which at 1000 games per round (to compensate for the random start positions) would take approximately 10^8 seconds on my home computer! Because each move is limited to 150ms the system is not limited by cpu speed. However tournaments could be run in parallel on multi-core architectures so I re-wrote tournament.py as tournament_mp.py to take advantage of this.

Even running on an AWS c4.4xlarge with 16 vCPUs this would still take nearly 3 months to run. I also ran into a problem with timeouts happening when I ran one process per core. This is because the timer is based on elapsed time not CPU time so if the OS switches out the process at the wrong point it can cause the agent to lose the whole game. The solution to this was to run less processes - I found running 3 on a four core machine or 14 on a 16 core machine to work well.

Test run 1

For the first run I chose the following weight ranges:

w_0	1, 2
w_1	1, 2
w_2	0, -1
w_3	1, 2
w_4	1, 2
w_6	0, -1

Including the benchmark function this produced 65 agents. At 200 games per round and 14 sub-processes (running on an AWS c4.4xlarge with 16 vCPUs) this took 3 hours 45 minutes. Here are the results for the agents that beat the benchmark agent:

1st Half			2nd Half			
w_0	w_1	w_2	w_3	w_4	w_5	Score
1	2	-1	1	2	0	75.75
1	1	0	2	1	-1	75.67
2	1	-1	1	2	-1	75.08
1	1	0	2	2	-1	74.67

1	2	-1	2	1	0	74.67
1	1	-1	1	1	0	74.42
2	1	-1	2	2	-1	74.25
1	1	-1	2	2	-1	74.00
1	1	-1	2	1	0	73.92
1	2	-1	1	1	-1	73.75
1	1	-1	2	2	0	73.67
1	2	-1	2	2	-1	73.58
1	2	-1	2	1	-1	73.42
2	1	0	1	2	-1	73.42
1	1	-1	1	1	-1	73.33
2	1	-1	1	2	0	73.33
2	1	-1	2	1	-1	73.33
2	2	0	2	1	0	73.33
1	1	-1	1	2	-1	73.17
1	1	0	1	1	-1	73.17
2	2	0	1	2	-1	73.08
2	1	0	2	1	0	72.92
1	1	0	1	2	-1	72.83
2	2	-1	2	2	-1	72.83
ID_Improved						72.75

Results from the first test run

The yellow row is the benchmark row and the red row is the best heuristic from this run.

Test run 2

For the second run I decided to increase the range for some of the weights and also increase to 500 games per round.

w_0	1, 2
w_1	0, 1, 2
w_2	-1, 0, 1
w_3	1, 2
w_4	0, 1, 2
w_6	0

In order to run in a reasonable time I decided to ignore the distance from the center feature for the second half of the game. This test took 18 hours to complete. Here are the results highlights:

First half			Second half			
w_0	w_1	w_2	w_3	w_4	w_5	**Score**
1	1	1	2	2	0	72.57
2	1	-1	2	1	0	72.49
1	2	-1	1	2	0	72.43
1	2	-1	2	2	0	72.23
2	1	0	2	2	0	71.89
1	1	0	2	1	0	71.6
2	1	1	1	1	0	71.54
1	2	1	1	2	0	71.46
1	1	-1	2	1	0	71.46
2	1	0	1	1	0	71.43
1	2	-1	2	1	0	71.4
1	2	1	2	2	0	71.4
2	2	1	2	1	0	71.31
2	1	0	1	2	0	71.29
2	0	1	2	1	0	71.23
2	0	0	2	1	0	71.2
2	1	1	1	2	0	71.17
1	1	-1	2	2	0	71.17
1	2	1	2	1	0	71.17
1	2	-1	1	1	0	71.14
1	1	1	1	2	0	71
2	2	0	1	2	0	70.97
2	1	0	2	1	0	70.97
1	0	1	2	1	0	70.94
1	1	0	1	2	0	70.91
1	1	0	2	2	0	70.91
1	2	0	2	2	0	70.86
1	0	1	2	2	0	70.8
2	2	-1	1	1	0	70.74
2	0	-1	2	1	0	70.74
1	2	1	1	1	0	70.71
2	0	-1	2	2	0	70.71
2	1	1	2	2	0	70.71
1	1	1	1	1	0	70.69
2	1	-1	2	2	0	70.69
1	1	0	1	1	0	70.66
2	2	-1	2	1	0	70.66

2	2	0	2	1	0	70.63
1	2	0	1	2	0	70.6
2	1	1	2	1	0	70.6
2	2	0	2	2	0	70.6
1	2	0	2	1	0	70.6
1	0	-1	1	2	0	70.46
2	2	1	1	2	0	70.4
1	0	-1	2	1	0	70.37
1	0	0	1	2	0	70.34
1	0	-1	2	2	0	70.34
2	1	-1	1	2	0	70.17
1	0	0	2	2	0	70.17
2	0	1	1	1	0	70.11
1	0	-1	1	1	0	70.11
2	0	1	1	2	0	70.09
1	1	-1	1	2	0	70.09
1	0	1	1	2	0	70.03
2	0	0	1	2	0	69.97
1	1	1	2	1	0	69.97
1	1	-1	1	1	0	69.91
2	1	-1	1	1	0	69.89
2	2	-1	1	2	0	69.83
1	0	1	1	1	0	69.8
2	0	0	2	2	0	69.71
ID_Improved						69.69

Results from the second test run

The yellow row is the benchmark row and the red row is the best heuristic from the last run.

Test run 3

Because the first two tests did not produce a single conclusive winner I decided to pick one of the top performing heuristics and run a longer test head-to-head with the benchmark function. This was run in a single process for 1000 games per round and took seven and a half hours.

1st Half			2nd Half			
w_0	w_1	w_2	w_3	w_4	w_5	Score
1	2	-1	1	2	0	73.27
ID_Improved						71.93

Results from the third test run

The yellow row is the benchmark row and the red row is the best heuristic from the previous runs.

Further Study

After analyzing the results from the first two test runs there was no clear pattern as to which features were more important for the game. It is possible that by expanding the weight ranges a discernable pattern may appear. There are other features that could be explored - longest path and neighborhood sparsity to name two. As always a balance has to be found between accuracy and performance. Every millisecond the evaluation function uses up is a millisecond the search function doesn't have to explore another branch.

Conclusion

The heuristic that I choose to submit is [1,2,-1,1,2,0]. I think that this is the best heuristic because it:

1. Demonstrated better results in all three tests.
2. Seeks to minimize the opponent's available moves more than maximizing its own moves but still takes both metrics into account.
3. Stays away from the center of the board for the first half of the game but shows no preference for the second half of the game.
4. Only adds two additional calculations - game stage and distance from center. This means the heuristic is almost as fast as ID_Improved so search depth is not overly compromised.

This is implemented as __heuristic3__ in game_agent.py