AIND Isolation Heuristic Analysis

Author: Balint Toth, 2017 February Cohort

Overview

My approach to custom isolation heuristics was to make the best default scoring algorithm called *Improved* even better. In this algorithm the score of a game state is the sum of the possible moves by our player minus the opponent's number of possible moves. The heuristics I came up with all use this same scoring as a base, but if multiple game states have the same score there are further logic to decide which one to go with. In that sense they are tie breakers on top of the original algorithm. My score variations were:

1. *Middle*: if base scores are equal the extra value is bigger if the player ends up closer to the middle/center of the board.
2. *Edge*: if base scores are equal the extra value is bigger if the player ends up further away from the middle/center of the board.
3. *Attack*: if base scores are equal the extra value is bigger if the player ends up closer to the opponent.
4. *Avoid*: if base scores are equal the extra value is bigger if the player ends up further away from the opponent.

Example calculation:

```
avoid_score = improved_score * 1000 + min(999, square_distance_between players)
```

Measurements

I have modified the provided tournament Python module in two ways:

1. Each tournament is played for 100 matches.
2. All the CPU and my agents were using the same search algorithm in one tournament. I did this to make sure search algorithm differences have no impact on the outcome.

Altogether I have played six such tournaments, two with each of these search algorithms:

1. Minimax with depth of 3.
2. Minimax with depth of 5.
3. Incremental deepening alpha-beta with 150 ms round time.

I have chosen to play two from each to be able to see the variance between the same type of tournaments. This variance stems from the randomness of starting moves and choosing moves from the ones with same score during the game play, so it is independent from the actual algorithms' performance.

Results

Here is the outcome of one tournament of 100 games.

```
                    *************************

                        MiniMax depth = 5

                    *************************

Match #    Opponent  MM_Improved    MM_Middle    MM_Edge      MM_Avoid      MM_Attack

                     Won | Lost    Won | Lost    Won | Lost    Won | Lost    Won | Lost

1       Random       97  |   3     93  |   7     94  |   6     96  |   4     95  |   5

2       MM_Open      49  |  51     62  |  38     53  |  47     52  |  48     53  |  47

3       MM_Center    69  |  31     73  |  27     79  |  21     84  |  16     76  |  24

4       MM_Improved   52  |  48    48  |  52     43  |  57     48  |  52     51  |  49

-------------------------------------------------------------------------------------

        Win Rate:       66.8%        69.0%        67.2%        70.0%        68.8%
```

In the six tournament altogether the strategies has played 600 matches. These are the six win rate results:

	Default Improved	Improved + Middle	Improved + Edge	Improved + Avoid	Improved + Attack
Minimax depth 5 run 1	70.50%	70.00%	70.80%	69.50%	72.80%
Minimax depth 5 run 2	66.80%	69.00%	67.20%	70.00%	68.80%
Minimax depth 3 run 1	70.50%	69.50%	72.80%	71.50%	67.00%
Minimax depth 3 run 2	70.00%	69.80%	67.50%	73.80%	68.20%
AB Iterative Deepening run 1	57.50%	68.50%	65.80%	68.20%	67.20%
AB Iterative Deepening run 2	56.50%	63.20%	61.50%	67.80%	64.50%

Based on this the average winrate and standard deviation of win rate of each strategy is:

	Default Improved	Improved + Middle	Improved + Edge	Improved + Avoid	Improved + Attack
Average winrate	65.30%	68.33%	67.60%	70.13%	68.08%
Standard deviation of runs	6.01%	2.35%	3.60%	2.04%	2.50%

As we can see a couple of hundred games still leaves a few percent of random difference in the results. The trend we see is that having tie-breaker logic lowers the variance of the default strategy and also improves the win rate.

To get more decisive results I made each strategy play 2000 matches against the default *Improved* logic using Alpha-Beta Iterative Deepening search.

	Win	Lose	Win Rate
Default Improved	1016	984	50.80%
Improved + Avoid	1392	608	69.60%
Improved + Middle	1290	710	64.50%
Improved + Attack	1390	610	69.50%
Improved + Edge	1312	688	65.60%

Evaluation of the Results

From the 2000 matches we see nicely that the *Improved* strategy has close 50% win rate against itself. Also the *Avoid* and *Attack* strategies emerge as clear winners over the others.

Based on my experiments I have chosen the *Avoid* strategy as my best scoring algorithm, because:

1. It has the highest win rate when using different search algorithms.
2. It ties as the highest win rate when playing a lots of games with the same conditions.
3. It has a low variance so can be used together with a vide variety of searching algorithms.