Udacity Artificial Intelligence Nanodegree

Isolation game - Heuristics analysis

Federico Bogado, December 20, 2017

In this project, the objective was to implement the minimax with alpha-beta pruning algorithm, using iterative deepening, to play a modified version of the game "Isolation".

The agent uses a heuristic function to score different game states and uses that information in the search algorithm.

Three different functions were implemented and tested against the already implemented agents in a tournament. A study of multiple tournaments results was made.

Heuristic functions:

- ### custom_score

For the current game state, check the available moves for both players, and how many of them are shared by both players.

If the agent has the next move, the number of shared moves adds to the final score. If the opponent has the next turn, that number subtracts from the final score. The final score is the number of available moves for the agent, minus the number of moves for the opponent, and then add or subtract the number of shared moves.

The idea behind this function is making the agent take into account game states where he can block opponents move, and avoid being blocked.

- ### custom_score2 and custom_score3

These two functions have similar mechanics, but their approach to the game is different, in the way that one is more defensive and the other is more offensive. The score of each is calculated in the same way:

```
score = agent_moves * defense - opponent_moves * offense
```

The idea is that we can make one agent prioritize game states where it has a lot of available moves, without caring that much about the opponents available moves. The other agent does the opposite, it prioritizes game states where the

opponent has a low quantity of moves without caring that much about its own available moves.

The values used for each agent are:
Offensive agent: `defense = 0,5 ; offence >= 1`
Defensive agent: `offence = 0,5 ; defence >= 1`

To obtain the best values for offense and defense, in each agent, an experiment was done, where the agent plays 400 matches against the AB_Improved CPU agent, changing the values by a little amount. Then with the four best values of each, the agent played 1000 matches against the AB_Improved CPU agent. The agents with the best scores are used to test the heuristics in the tournaments against all the CPU agents.

This are the four best scores of each agent:

	Defensive value	Offensive value	Win rate
AB_custom_3	0,5	1,75	**0,485**
	0,5	2	**0,519**
	0,5	2,5	**0,507**
	0,5	2,75	**0,468**
AB_custom_2	1	0,5	**0,505**
	1,5	0,5	**0,513**
	2,75	0,5	**0,497**
	3	0,5	**0,507**
AB_custom	-	-	**0,513**

The code and results of this analysis can be found here.

The final values are:
Offensive agent: `defense = 0,5 ; offence = 2`
Defensive agent: `offence = 0,5 ; defence = 1,5`

Tournaments

The three agents were tested against all the CPU agents in 12 different tournaments, with this results:

Tournament	AB_improved	AB_custom	AB_custom_2	AB_custom_3
1	68,57	70,00	67,14	72,86
2	77,14	75,71	68,57	72,86
3	61,43	68,57	74,29	67,14
4	78,57	68,57	65,71	77,14
5	80,00	68,57	70,00	67,14
6	72,86	74,29	62,86	72,86
7	74,29	77,14	72,86	65,71
8	74,29	72,86	75,71	70,00
9	68,57	68,57	70,00	72,86
10	75,71	68,57	74,29	67,14
11	62,86	65,71	67,14	64,32
12	71,43	65,71	75,71	72,90
Averages	**72,14**	**70,36**	**70,36**	**70,24**

Conclusions

The selected function to be the primary custom score, is the first one mentioned in this document. But the other two are pretty close, and they could also have been chosen. The three of them won the 1000 match challenge against the AB_Improved CPU agent, but by a little margin.

Future work

Some more work can be done with the defensive and offensive values. Testing for different values is very slow, and only one value was changed at a time during the experiment. If a way to test more quickly is found, maybe changin both at the same time in different ranges could help find some better pair of values, for each strategy.