Choosing a Heuristic to an Isolation Game-Playing Agent

Uirá Caiado

The role of games in artificial intelligence is intriguing. In an article published by [1], the author argued that there are several reasons for this connection. For example, many researchers use games as training grounds for real world problems. Others, observing that different games require different cognitive skills, think games can help them understand how the problem of intelligence may be broken down into smaller, more manageable chunks. Others still, building on these two observations, think games can help them develop a proper theory of artificial intelligence.

In this project, we also engaged in the game-playing domain. I implemented an adversarial search agent to play a game called "Isolation". As stated by [3], Isolation is a deterministic, two-player game of perfect information in which the players alternate turns moving a single piece from one cell to another on a board. Whenever either player occupies a cell, that cell becomes blocked for the remainder of the game. The first player with no remaining legal moves loses, and the opponent is declared the winner.

To solve this game, I have implemented two strategies: minimax algorithm and alpha-beta pruning. As explained by [2], the first one performs a complete depth-first exploration of the game tree and its time cost in real games can be impractical. The second one, alpha-beta pruning, ignores a portion of the search tree that makes no difference to the final choice. According to [2], when this technique is applied to standard minimax tree, it returns the same move as minimax would.

However, alpha-beta still has to search all the way to terminal states for at least a portion of the search space. To cut off the search earlier and allow the agent makes decisions in a reasonable amount of time, I implemented three different versions of a heuristic evaluation function, inspired in the *improved_score* and *open_move_score* evolution functions, already implemented by Udacity. Evaluation functions estimate the expected utility of the game from a given position and are applied to states in the search, turning non-terminal nodes into terminal leaves.

The first function implemented, called *custom_score3*, rewards the agent simply by the number of legal moves still available, as the *open_move_score* function, and also gives an additional reward to the agent if it is in the center of the board. The idea is to motivate the agent to dominate the center when it is possible.

The second function implemented, *custom_score2*, is a weighted linear function, where is given to the opponent's moves two times more relevance than the agent's moves. The motivation of this approach is to influence the agent to prioritize moves that minimize the moves from the opponent before maximize its moves.

Finally, the *custom_score* uses the same approach that the last function, giving more emphasis to the opponents' moves. However, like the first function, this one also gives an additional reward when the agent moves to the center of the board. Thus, this function motivates the agent to minimize the moves from the opponent and also to dominate the center of the board.

agent / heuristic	AB_Improved	AB_Custom	AB_Custom_2	AB_Custom_3
Random	95%	100%	90%	90%
MM_Open	75%	80%	65%	60%
MM_Center	75%	90%	90%	75%
MM_Improved	70%	65%	75%	80%
AB_Open	45%	60%	60%	50%
AB_Center	50%	65%	50%	40%
AB_Improved	40%	45%	55%	45%

Table 1: Output example from *tournament.py*

To test these functions, I performed ten tests with ten matches each, using *tournament.py*. The table 1 presents an example of the result from a tournament. Then, I compared the performance of the agent using each heuristic created to the performance of an agent using the heuristic *improved_score* . To do so, I used a one-sided Welch's unequal variances t-test[1] for the null hypothesis that the agent using custom heuristic has the expected win rate greater than the performance of an agent using the benchmark heuristic. As the implementation of the t-test in the Scipy[2] assumes a two-sided t-test, to perform the one-sided test, we have to divide the p-value by 2 to compare to a critical value of 0.05 and require that the t-value be greater than zero.

There was not a significant difference between the performances of the agent using the custom heuristics. The best one was the performance of the *AB_Custom* (t-value ≈ 0.97; p-value < 0.173). Even though this agent was able to beat the benchmark in 7 of the 10 tests performed, its performance was not significantly superior to the agent *AB_Improved*, as suggested by the hypothesis test.

The agent *AB_Custom_2* performed worst (t-value ≈ 0.29; p-value < 0.386), beating the benchmark in 5 of the 10 tests performed and *AB_Custom_3* was significantly worst than the agent *AB_Improved* (t-value ≈ -3.04; p-value < 0.004).

Based on these results, I recommend to use the function *custom_score*. Even though the agent that have used this function was not able to play much better than the *AB_Improved*, it still won 70% of the tournaments.

References

[1] The Economist. Shall we play a game?, 2017.

[2] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach, 2003.*

[3] Udacity. Build a game-playing agent. `https://goo.gl/ND8wPY`, 2017.

[1]Source: `https://goo.gl/Je2ZLP`
[2]Source: `https://goo.gl/gs222c`