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.

....

To test these functions, I performed ten tests with ten matches each, using *tournament.py*. Then, I compared the permorfance of the agent using each heuristic to the performance of an agent using the heuristic *improved_score* using a Welch's unequal variances t-test.

The table 1 presents an example of the result from an turnment.

A Welch's unequal variances t-test was conducted to compare if the PnL of the learner was greater than the PnL of a random agent. There was a significant difference between the performances (t-value ≈ 7.93; p-value < 0.000). These results suggest that the functoin that really outperformed the benchmark was ...

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*

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.