Project 2 RoPaSci360 - Report

COMP30024 - Artificial Intelligence - 2021 semester 1

Authors: Olivia Ryan (1082223), Lu Liu (1068336)

This report summarises and discusses the techniques used in the RoPaSci360 game playing agent developed by Lu Liu and Olivia Ryan: “Luv.” It covers the strategies created and the algorithms those strategies use, the game insights used to enhance these strategies, how the time and space constraints were addressed and how performance was assessed and used to identify weaknesses in the strategies.

Strategies and Algorithms

Our approach was to develop various strategies of increasing difficulty assessing their performance as we go. Ultimately we developed four main strategies:

- Random
- Minimax with alpha-beta pruning
- Equilibrium payoff
- Monte Carlo tree search

Random

We started off with a simple player that computes all the possible legal moves and chooses one randomly. This was very straightforward and with this strategy our player was able to play a valid game.

Minimax with Alpha-Beta Pruning

The next strategy we developed used the minimax tree search algorithm as discussed in lectures. Our minimax algorithm treated RoPaSci360 as a turn-based game with our opponent playing second. This biased the search in the opponent’s favour as they have

more information when making a minimising decision in the tree search. This strategy was very slow and a player that used this strategy was certainly outside the 60 second time limit. We addressed this by implementing alpha-beta pruning which reduced the time though the player was still using more than the provided 60 seconds to complete a game. For some parts of this strategy, we used and adapted the source code from github of aima-python function alpha-beta search (1).

Equilibrium Payoff

We started using a payoff matrix in order to consider the actions of two players simultaneously. Using this matrix, we created a simple strategy that would consider a limited set of actions for each player and compute a payoff matrix that assessed the entire board state given a set of action combinations.

The limited set of actions were chosen based on a crude evaluation function that only considered the distance between the token in question and its nearest target and the board evaluation function was simplistic in that it only considered the average distance from each of our tokens to its nearest target.

We initially had intentions to develop more complex and accurate evaluation functions and to combine the minimax and equilibrium payoff strategies to create a hybrid that would be able to complete a minimax tree search and consider actions as having a simultaneous effect. We didn't end up following this plan as our research into the Monte Carlo tree search algorithm led us to create an entirely new strategy.

Monte Carlo Tree Search (MCTS)

The logic of MCTS is based on the paper "A Survey of Monte Carlo Tree Search Methods" (2) from which we used the UCT algorithm (p.9, algorithm 2). The Monte Carlo tree search algorithm consists of four stages: selection, expansion, simulation and backpropagation.

To assess the desirability of an action, the algorithm runs simple simulations where the succeeding actions are chosen at random until a goal state is reached. Then the utility

of the goal state is propagated back to the root node. We construct the tree, expand the tree and run simulations, until we can return a “best action.”
We improved the speed by limiting the actions considered and by implementing a calculation time so as to restrict the growth of the tree (MCTS is convenient for setting a maximum search time and returning the best result obtained in that search time). For more details on our implementation and the algorithm itself, view the “Monte Carlo Tree Search - Research” section of this report.

This algorithm is highly useful for RoPaSci360 which has a very large number of action combinations as it reduces the search depth and provides a decent method for evaluating actions. The win rate against the provided greedy player improved drastically and the time taken reduced significantly though the time constraint still posed a problem as discussed in the “Constraints” section of this report.

Game insights

First Move

The first move of the game has fairly little impact on our chances of success since we have no information about the other player. For this reason, we decided to use a random strategy for the first move rather than wasting computation time on completing a monte carlo tree search (or other algorithm) that will be of little use.

Limiting Actions

The list of all possible actions for a given game state can be very very large. Since two of our four strategies use tree search algorithms that are very costly, we concluded that the set of actions we will consider can be whittled down using token distances before applying costly algorithms to determine the utility of an action. We don’t want our player to spend precious time considering swinging a token that has no targets on the board or one that is as far from its target as possible. Instead we can rule these out and only consider actions that are “likely” to be useful. Then we can complete a comprehensive

assessment of the action in the context of the entire game state using an adversarial game search strategy.

360 Moves?

The maximum number of moves a player can make is 360 but this does not mean the average game will come close to this number. We could have applied a time constraint (in relation to MCTS) so that each move would be given an equal share of the time (time to compute best action = 60/360) but this would result in the player not taking advantage of the available time in games where the player makes fewer than 360 moves. For this reason, the time given to compute the best action adjusts as the game is played. Earlier turns are given much more time than later turns as later turns have a much lower probability of being used in the game.

Constraints

Time

Our MCTS strategy would often win against a greedy opponent within 60 seconds however there were occasional games that exceeded this limit. To guarantee that our player will never use more than 60 seconds, we implemented an adjustable maximum computation time variable that could be passed to the tree search so that it would terminate the search if the time taken had exceeded the given max. The maximum computation time is recalculated every turn, taking into account the amount of time used so far and the number of turns used so far. The ratio of these is adjusted using a constant (“RATE”). This approach prioritizes earlier moves by giving them more time than later moves so that we utilize as much time as possible while still leaving enough time to calculate up to 360 moves.

Space

None of our four strategies used an excessive amount of space so we attempted a trade-off by applying alterations that would increase the space requirements but decrease the time requirements.

Performance

To assess our player's performance we ran various games against the random and greedy players provided in the battleground. We were able to check that our player always made valid moves and we were able to adjust time-related variables based on the average time used in each game.
Having developed the MCTS strategy, we tested it against the greedy player with decent results. Our success rate was approximately 80%.

We then played against other students with the intent of revealing any weaknesses that were not exploited by the greedy player. One weakness this revealed was the excessive use of throws in the early stages of the game.
Since our strategy could only look a few moves ahead before completing the simulation stage, it failed to recognise the utility of saving throws for later in the game. To address this, we limited the throw actions that would be considered. Specifically, we only wanted to consider throws when a throw would be necessary to ultimately win (the opponent has a token that cannot be defeated by any of our tokens currently on the board) or when a token could be thrown directly onto a target to defeat it.
This adjustment improved our performance against the greedy player but we ran out of time to test it against other players.

Monte Carlo Tree Search - Research

Selection uses an evaluation function to select the “most promising child”. The evaluation function we used is upper confidence bound for trees (UCT) based on part (p.7, section 3.3.1). A child is selected to maximise the following formula:

$$UCT = \bar{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

In the UCT formula, n is the number of visits for its parent board, $\bar{X}_j = \frac{Q(v)}{N(v)}$ is set as the approximation of evaluation for this node, where Q(v) is the number of wins in the roll out process and N(v)= nj is the total visit time of this node. C_p is used as a factor to trade off between exploration and exploitation. During selection, we set C_p to $\frac{1}{\sqrt{2}}$ to focus on exploration. When it comes to the final decision, we set C_p to 0 to focus on the success rate.

This page provides a more comprehensive explanation of the MCTS algorithm: https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/

References

1. D. Bacon, P. Norvig, A. Maronikolakis, et al, (2020), Artificial Intelligence: A Modern Approach (AIMA), Github, https://github.com/aimacode/aima-python/blob/master/games.py
2. C. Browne, E.Powley, D. Whitehouse, et al, (2014), A Survey of Monte Carlo Tree Search Methods, p.5 section 2.4.2, Research Gate, https://www.researchgate.net/publication/235985858_A_Survey_of_Monte_Carlo_Tree_Search_Methods