Tronity: Alexander Hunter, Scott Wong, Stavya Datta

COMP30024 Project Part B Design Report

Abstract
Our task was to create a game-playing module for the game RoPaSci360, which can complete games within 60 seconds using 100 MB of memory or less. To do this, we developed heuristics to decide which pieces will be the best to move, and evaluate the value of board states in order to continuously attack enemy pieces.

Our approach
To handle the simultaneous-play nature of RoPaSci360, we decided to analyse a set number of "good" moves from each side for each turn. At each turn, our module will create a tree of possible states from the current state, at each stage selecting moves from those considered to be stronger at that stage using a pruning function. Then each leaf state node will be evaluated by a heuristic function, and pass the value to the primary move node leading to that state from the root node (current state). Once all of the following moves have been evaluated for a primary move node, the primary move node will be evaluated to have the average (mean) value of all its immediate children. This will allow us to choose the move which most consistently brings the game-playing module closest to winning.

Representing the board
We chose to represent the state of the game as a graph, and used three different classes of nodes to represent different pieces of information; the state node, the primary move node and the secondary move node.
The state node contains the information required to represent a board state. This information includes a list of pieces and their positions for both upper and lower players, a list of board tiles with pieces on each tile, the number of pieces thrown by each player, the team our game-play module is controlling and the depth that the current state node is at. In addition, each state node contains a link to the previous secondary move and a list of the possible primary move nodes.
The primary move node represents a valid move our game-playing module could make. Each primary move node holds information including the move the node represents, the sum of heuristic scores and the average heuristic score for making the move. In addition each primary move node contains a link to the parent state node and possible secondary move nodes.
The secondary move node represents a valid move our opponent could make. Each secondary move node contains the move it represents. In addition each secondary move node contains a link to the parent primary move node and resulting state node.
Our representation of the state of the game has several benefits. By representing player and opponent moves as primary and secondary move nodes rather than just storing state nodes with the corresponding upper and lower moves that resulted in that state, it becomes much easier to see which states can result from any of our agent's possible moves. Storing all the information needed to represent a state inside a single object (state node) makes keeping track of this information significantly easier.

Outcomes of testing

We discerned from experience testing heuristics and examining the rules of RoPASci360 that a successful board evaluation heuristic should value the absolute number of pieces left for each side higher than any board state, except for the presence of invincible pieces. In addition, we found that the ratio for number of pieces is a very good indicator of relative strength, as each piece lost results in less options available and a higher vulnerability to facing an opposing invincible piece. Due to the rules of the game enforcing a set upper bound on turns and time, we found it more effective to not even consider known weak options from either team by preemptively pruning them, as this would take considerable time to explore moves a competent opponent would never play anyway. We found playing aggressively to be a more effective use of resources than defending our own pieces, due to the available time not being sufficient to examine both offensive and defensive moves. This is because it is possible to win a game of RoPaSci360 with only 1-2 pieces, while playing defensively is not even guaranteed to save player pieces once they reach the end of the board. Thus, as long as a player can maintain a piece advantage over their opponent, trading pieces is good (provided this doesn't give the opponent an invincible piece), and all-out aggression is a quick path to victory.

Our Heuristics

Our game playing algorithm makes two different types of decisions based on heuristics; evaluating game states and evaluating which pieces are best to move. By evaluating which of our pieces are best to move, and which opposing pieces are best to move, we can limit the search space to a set number of pieces. This lets us prune the list of moves to search through significantly, especially in the middle of the game, when several times more pieces are on the board compared to later in the game. By pruning the pieces considered worst to move, we minimise the loss in game performance from lowering the number of moves we consider. This has an additional benefit in more accurately predicting a skilled opponent, since a skilled opponent would not make weak moves anyway. Based on our testing, we found that certain moves were generally stronger than other moves, pieces closer to capturing opposing pieces (or those close to being captured) are generally better to move than those further away, and throws which are not directly onto an opposing piece (to capture it) or into the furthest row possible (to try and pressure the opposing pieces on board to move) are vastly inferior. Evaluating game states is necessary in order to select the best move out of those we search through. The information used to evaluate the game state affects the "strategy" our module will use. When evaluating the board state, we gave high weighting to capturing an enemy piece, a lower weighing for pieces close to capturing an enemy piece and a coefficient of the ratio of remaining player pieces to remaining enemy pieces (including remaining throws in both cases). The ratio of pieces coefficient keeps our module from throwing away too many pieces carelessly while still valuing trading pieces while ahead. When the opponent has 0 pieces, in order to avoid a dividing by 0 error, yet still give value to an absolute victory without severely distorting the heuristic valuation, we made this ratio factor equal to twice the number of player pieces left. Our board evaluation heuristic and piece pruning encourage our module to play very aggressively, since our testing showed that aggressive play is just as effective and quicker than a more balanced approach with the amount of resources available.

Fine-tuning

We made several adjustments to the decision-making process in order to save time by not considering inferior options. For the first four turns of the game, our module throws into the closest row to the enemy possible. By doing this, we ensure that we increase our throw range close to the “action” of the game as soon as possible, which is important as this prevents the opponent from picking off our pieces without risking their pieces getting captured by reinforcements. In addition, our module will try to hold two throws in reserve until the opponent has used all of their throws. This allows our module to counter an enemy piece on the board in the “endgame”, preventing our opponent from obtaining an invincible piece early. We also improved the pruning of moves by categorising moves into “tiers” based on general strength/difficulty to counter and determining the number of moves we can explore without running out of time. We then add moves from each tier of moves from strongest to weakest tier until we hit our maximum analysable move limit, so that we never consider a weaker move over a stronger one, but still utilise all of the resources available to us for each move. Finally, in edge cases where the difference between best and worst move is minimal and there is no clear great move (for example, if the opponent has no pieces on the board and must throw), we take a random move, in order to save time for more impactful decisions later.

Playing to draw

In cases where winning is impossible (opponent has an invincible piece), but the game is not over, we implemented a simple stalling strategy in an attempt to force a draw. We try to maximise the distance between our pieces and their predators by moving the player piece closest to a piece that can defeat it as far away as possible, delaying the enemy from being able to defeat an allied piece for as long as possible. While this strategy may not be able to stall a vastly superior opponent for very long, against an opponent which has not considered this possibility, we can almost guarantee a draw via either move limit or repeated states.

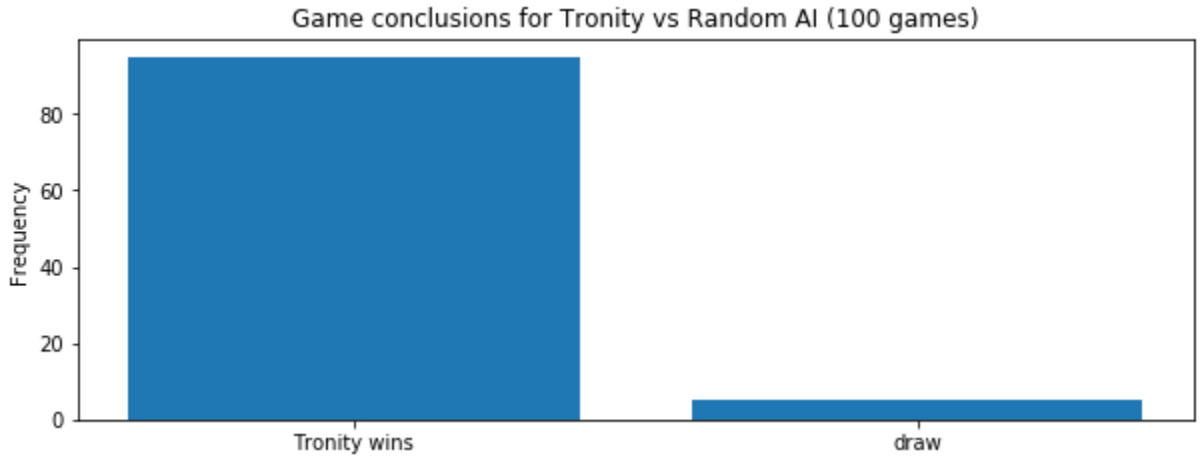
Performance

Our approach has several strengths. Because we only consider moves we consider generally strong from the opponent, even if our opponent does not make a move we expect, the outcome will likely not be disastrous, as could happen if we considered every enemy move, including the generally poor moves, equally probable. In addition, the focus of our heuristics on aggression and capturing pieces precludes our module from getting baited into a lengthy defensive period or being overly cautious, wasting precious turns and time. This is important because our goal is to win, not to avoid losing, and the only ways to win involve defeating at least 8 opposing pieces (As per game ending conditions 1 and 3). Our approach is quick in both moves and time whilst still being effective compared to countering strategies which require significantly more time to correctly identify and implement due to the need to balance defence and offence.

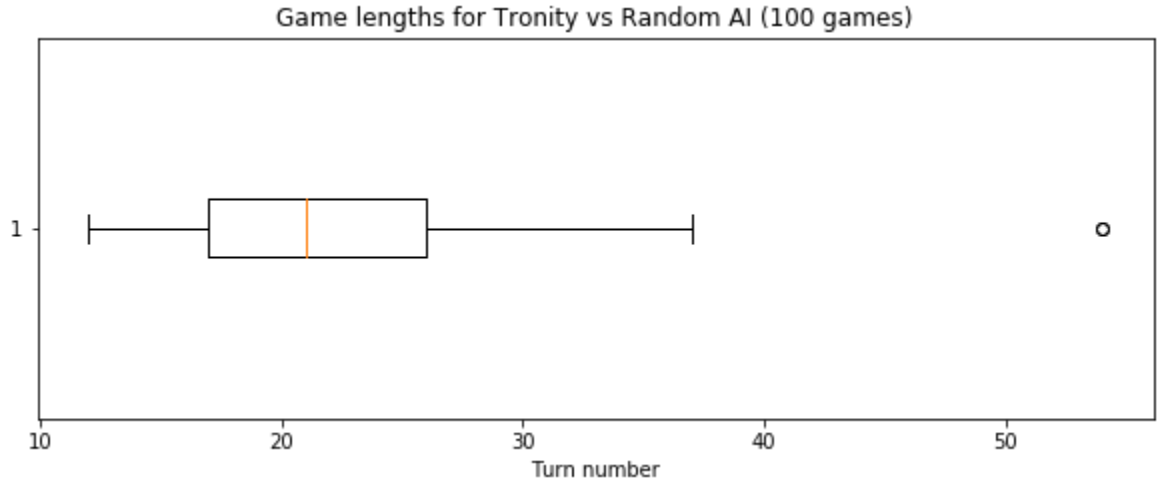
The main weaknesses of our approach is that because we can only look ahead a few turns at a time, we overlook plays that will not pay off until much later. In addition, because we only consider moves from the pieces with the highest current average utility, we may overlook a piece with one great move and many bad moves. However, due to careful selection of the moves that are chosen to be considered, our module makes consistently aggressive plays,

mitigating most of the disadvantage of these weaknesses, as opponents will have to make difficult decisions to either sacrifice pieces or waste their own moves defending their pieces.

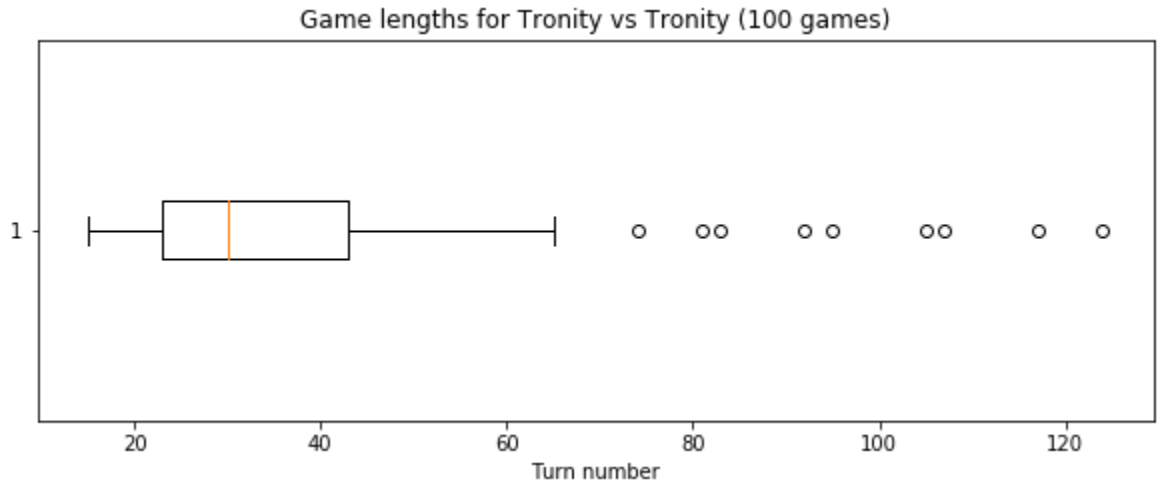
We tested the performance of our module by playing against different opponents. Our module played 100 games against a random opponent equally likely to make any valid move and won 98 games while only drawing 2 games.



These games ended quickly, with the mean game length being around 20 turns.

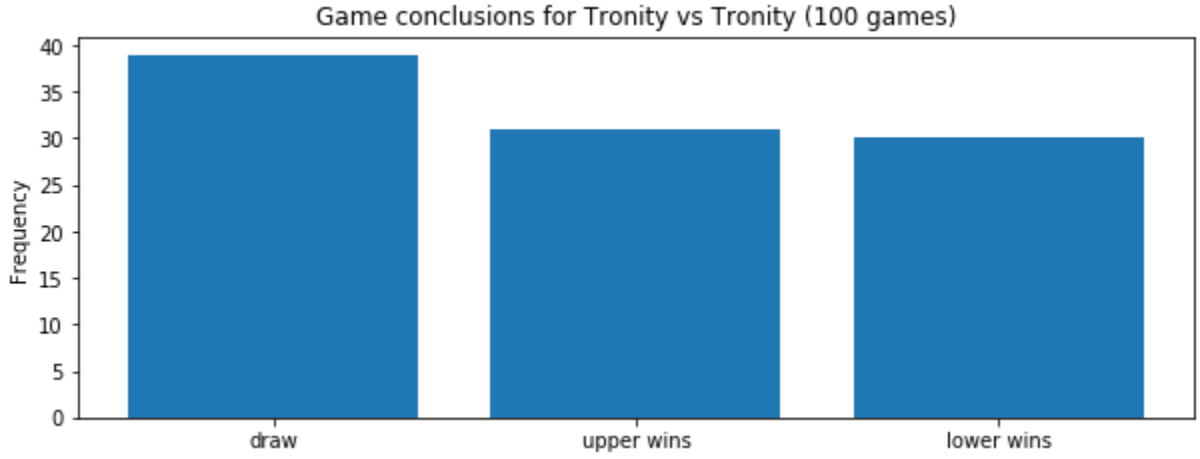


This test set a baseline expectation that our module can easily beat any unoptimised opponent. We also made identical copies of our agent play 100 games against each other, tracking the number of turns each game took (shown below).

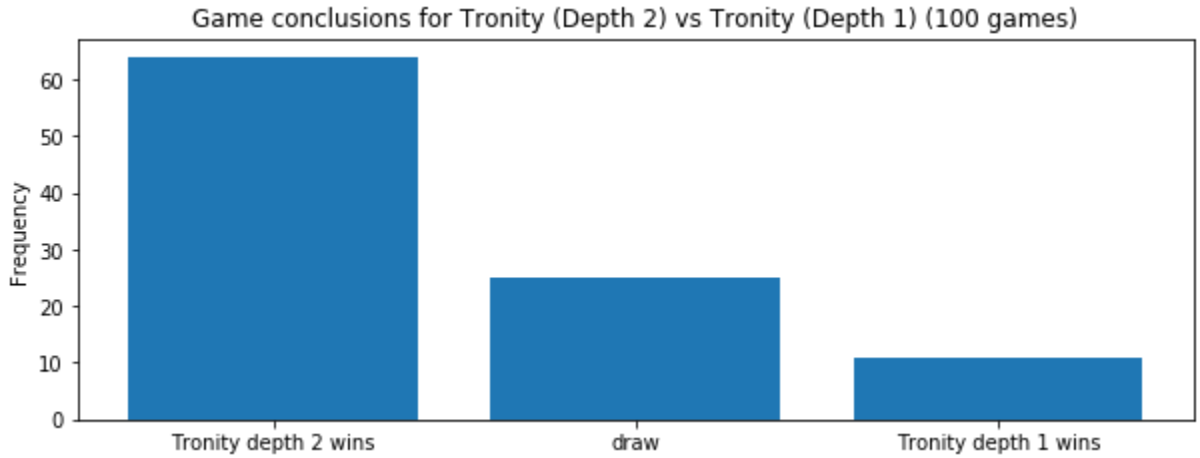


No game lasted longer than 130 turns, and the mean turns per game was under 40 turns. These results show that our agent consistently favours aggressive strategies, and that these aggressive strategies result in quick games.

It was determined that our agent has no bias in terms of which team it plays on as it won roughly an equal number of times when playing on both teams.



Finally, we conducted a test by playing 100 games between two copies of our module where one side would search to a depth of 2 and the other would only search to a depth of 1. The agent searching to a depth of 2 won over 60 out of 100 games, with over 20 of the remaining games resulting in draws.



This test conclusively proves that looking further into the future provides a decisive advantage.

Comparison to other approaches

Our approach has a lot in common with deterministic approaches such as minimax, as we assume the enemy will make moves from a set of good moves, thereby creating a defined search space. In addition, we also assume that our opponent will follow a strategy that we consider “optimal”, much like many deterministic approaches. However, our approach is not entirely deterministic, as we have introduced some randomness in order to save time. For example, if a tier of move contains too many moves to be entirely explored due to time constraints, instead of either ignoring all of them, or sorting them by some criteria (which takes even more time away from exploring moves), we randomly select moves to try and explore as many as possible. Our approach is different from machine learning approaches in that we rely on human analysis to construct generalised opinions about the value of board states, whereas machine learning approaches will generally use training data to construct probabilities of winning based on actions taken, without human input beyond assembling the training data.

Conclusion

We implemented an approach which explores a tree of the most generally strong moves by both sides in order to determine the best move for our module to make. Our heuristics heavily favour offensive play, in order to attempt to end the game quickly and efficiently. We used randomness in cases where the increase in game performance would be minimal or too costly, in order to save time for important decisions.