Saboteur Project Report

Raphaelle Tseng and Miya Keilin

April 2020

1 Introduction

Saboteur is a card game that was first published in 2004. It pitches 'miners' against 'saboteurs', where the 'miners' seek to reach a golden nugget, hidden under one of three cards at the opposite end of the board. The 'saboteurs' search to block the 'miners' from ever reaching their goal.
In our modified version of the game, we remove the 'saboteurs'. Instead, we have only two players, playing against each other, each seeking to find the golden nugget before the other. The players each hold a hand of seven cards. There are five different types of card in the game.

1. Tiles represent tunnel paths that can lead from the entrance tile to the objective
2. Malus stops your opponent from playing Tile cards until they can play a Bonus
3. Bonus heals from a malus
4. Destroy cards destroy existing tiles on the board
5. Map cards allow a player to look at a hidden objective.

During a turn, a player may play a card in their hand or drop a card. We have used a board size of 15 x 15 for this project, instantiated with an entrance tile and three hidden objectives, underneath one which is hidden the golden nugget.

In this project, we sought to implement an AI agent to play our modified version of Saboteur against other players. We chose to implement pruning to traverse the game state tree, before applying a heuristic function to determine the best course of action for our agent. We used a heuristic that, for tile cards, considers the distance in number of moves between the agent's current place on the board and the three possible locations of the golden nugget. For the other four cards, we assigned a heuristic based on situations and how value we perceived the card to be.

2 Motivation

In recent years, a boom in AI has resulted in a lot of research being conducted on the topic of games and game playing agents. Studies on turn-based games such as checkers and Go show that AI agents are most successful when using some variation of alpha-beta search or a Monte Carlo Search Tree. Although these are games where both opponents have perfect information, the popularity and success of these techniques inspired us to try adapting them for a similarly deterministic game like Saboteur. The number of possible cards and combinations on our board lend the game to a large branching factor as well. However Saboteur is a game with imperfect information; we cannot see the hand of the opposing player, nor do we initially know where the golden nugget might be located.

Saboteur's relatively recent inception also makes it less conventional than traditional games like chess, checkers and Go. This therefore limits the resources and works currently available, specifically related to this game. However we did find one paper by a group at the Universitas Pelita Harapan, Indonesia [1] who had modelled a decision system for Saboteur using heuristics and means-ends analysis. Their implementation had both miners and saboteurs and the agent worked well as either player. They suggested exploring Monte Carlo tree search and deep learning algorithms in the future. This paper was also a helpful starting point for us and motivated us to explore using effective heuristic functions.

3 Implementation

Using the information we had gathered we decided to pursue a pruning method instead of using Monte Carlo Tree Search because MCTS converges slowly. We began by assigning all the cards with different values, to rank the best moves depending on the board status. Some assignments could be made clearly: in the case when the opposition plays a malus, the bonus card in our hand should become the best valued card, with the highest priority to be played so that we can continue placing tile cards as soon as possible. By extension, a bonus card when no malus has been played should not be worth much as it has no value when played. Map cards are interesting too; if we don't know where the golden nugget is situated, a map card is helpful in narrowing down potential objective options. However, when the location of the golden nugget is known, map cards become effectively useless to us, and their value can be dropped. At most, we will only ever need two map cards to determine where the golden nugget is located. Any more than that becomes useless to us.

We decided the heuristic value for tile cards based on a tiles distance from where our current estimate of the nugget is and the tile itself. The distance is measured by how many tiles it would take to build a continuous path from the tile played to the nugget. Tile cards such as 0 and 6, which have paths exiting from the bottom of the tile, have a shorter distance than tiles cards such as 7

and 10, which have no path exiting from the bottom, if they were played at the same position on the board. The heuristic for tile cards with continuous paths is the distance from the nugget calculated in this manner.

Furthermore, we decided that tile cards with discontinuous paths, such as 4 and 13, are detrimental to our effort to win, so we wanted to try and keep them as far away from our path as possible. We used the same method as described above to calculate the distance from a position on the board to the nugget; however, the heuristic was 100-distance instead of just the distance, since the further away from the nugget—and the path we are attempting to build toward it–we could place the tile cards with discontinuous paths, the better off our agent is.

These were situations we wanted to prune from the search tree to narrow the options of cards we might have to play.

In the cases where we were unable to prune any branches or find anywhere to play a tile card, the agent was told to drop a card. This was the final option for our agent if nothing else could be done.

We decided to avoid thinking about destroy cards and instead prioritise finding the golden nugget rather than preventing our opponent from reaching it first.

4 Theory

We applied theories taught in class and taken from the textbook [2]. Our game state tree was implemented so that the root represented the current board state and the children were all the possible states resulting from all the possible legal card moves that could be played, much like in any minimax and alpha-beta pruning state tree.

Alpha-Beta pruning is a standard pruning technique that allows us to reduce the size of the game tree by 'pruning' or removing branches that look less promising than what we currently have. It allows us to save computational resources, and helps us return the best possible move available to us.

The heuristic in game playing programs represent an approximation of the win/loss value of the game. For it to be effective and useful, it should return something that accurately represents the current game state, and allows different moves and game states to be compared to one another, from the perspective of the current agent. From this, we can then determine what state and move is more favourable for our agent.

It should be noted that there is always a trade off between the complexity of any heuristic and the search depth. Generally, it seems that a simpler function with deeper search returns better results than a computationally-heavy one with restricted search depth. Hence our choice of a simpler heuristic to evaluate the current game state.

5 Advantages and Disadvantages

One advantage of our strategy is that the agent played aggressively. We valued knowing where to build our path (finding out where the nugget is quickly) and preventing the other player from making progress, so we gave good heuristic values to map and malus cards. In the same vein, we prioritized playing bonus cards if we had a malus card played against us since we cannot win if we cannot build and we cannot build if we have a malus card played against us. Secondly, since our strategy does not require simulation of play to decide the best move, the program had relatively little to compute making it fast and space efficient.

A disadvantage of our strategy is that we assumed our opponent is focused on building path and winning in the same way that we are (it is playing optimally); that is, we didn't check for continuous path before we played a tile. This left us susceptible to playing a tile on paths that could have been disrupted by a destroy card or were simply built with tile cards with discontinuous tiles. Our agent is also limited as it cannot search and evaluate the entire search tree and thus may be finding the best solution 'locally' but not overall.

Additionally, the use of a simpler algorithm means a lower level of sophistication that may not always work effectively in more niche cases.

6 Alternative Approaches

We attempted to implement a more concrete version of the alpha-beta pruning algorithm [2]. This was done by implementing the start of a minimax tree that would be called upon by our alpha-beta pruning function. minimaxDec, which returned a Move and took in an array list of SaboteurMoves as well as a Saboteur BoardState looked to do an iterative deepening search to a max depth of three. It would search the game search tree across these three levels to return the most promising node by making moves on a cloned board. This function also called highestMove, which iterated through an array for the maximum value present. If we had been able to successfully implement alpha - beta pruning, our agent may have been more successful as it would've been able to search at a greater depth and with a faster search time.

A clear alternative route would also have been to implement Monte Carlo tree search.

7 Future Improvements

In the future, improvements could be made to address some of the disadvantages of our strategy, such as the case where we place a tile where there is no continuous path to the nugget. To do so, we would have to find if there existed a continuous path from the entrance to the position where we want to place our tile card and consider this boolean in our calculations of the heuristic value for a given position and tile. This would increasing the computation time required for every turn, but it would improve our heuristic function. We could expand

on this idea, too; it is one thing to identify that a path is discontinuous and therefore avoid building it, but it is another to try and fix that path so that we can continue to build it. Identifying broken paths that are promising and being able to fix the path would improve the performance of our agent.

Fixing the alpha - beta pruning functions would also improve our player.

This project could also be expanded to integrate machine learning techniques by using neural networks to learn from a larger data set of game states and outcomes. A Cornell University Artificial Intelligence lecture by B. Selman [3] introduces the use of neural networks to develop a 'winning moves database', with the potential to be expanded by playing a much larger number of games and storing moves which commonly lead to a win.

From this, we could develop a weighted heuristic function which considers more complex game features than the ones currently evident. In addition, a heuristic function which also takes into account current search depth and the average outcome of a branch (rather than simply picking the best move currently available) might help the agent determine which moves will eventually lead to more successful outcomes and potential wins at the end of the game. Given that we don't know what the opposing player has in their hand and we cannot necessarily predict their moves, this could be worth exploring.

References

[1] D. Krisnadi, S. Lukas, L. Logan, N. Bachtiar, and L. Lohanda. Decision system modelling for "saboteur" using heuristics and means-ends analysis, 2019.

[2] S.J. Russell and P. Norvig. *Artificial Intelligence: A modern approach (third edition).* 2009.

[3] B. Selman. Intelligent machines: From turing to deep blue to watson and beyond, 2018.