Comp424 Saboteur AI Report

Yichao Yang 260764629

Weige Qian 260763075

April 2020

1 Introduction

The main goal of this project is to design an AI player for the game called Saboteur where there are two players participate. To win the game, each player aims to build a path based on tile cards to reach the goal position "nugget" on the board first. During a Saboteur game, each player plays consecutively. There are 4 different types of cards and each card will have different effects on the current board state. The AI agent will receive the information of the current board state, and choose his move based on this information.
This report reflects on how a Saboteur AI player was designed and constructed from scratch, demonstrates the algorithms that power the agent. It focuses on the discussion about his advantages and disadvantages, and possible improvements that can be done to make the AI behave more smartly and efficiently.

2 Approach and Motivation

2.1 Overview

The basic algorithm we choose to design our AI agent is the Best First Search. The best first search is a traversal technique that decides which node is to be visited next by checking which node is the most promising one and then check it. For this, it uses an evaluation function to decide the traversal.
During the Saboteur game, in each turn, we traverse through all legal moves and greedily choose the best move based on their approximate value of heuristics (the move with the smallest heuristic). Thus, the evaluation function is the most important part of the algorithm. Since each of 4 types of cards has different effects on both opponent and our AI player, we evaluate the cards separately.

2.2 Heuristic Evaluation

2.2.1 Preliminary

In order to clearly state the evaluation functions, we first introduce some concepts and variables:

1. **target**: if the agent has known the position of nugget, then the target is the nugget, otherwise, it is set as hidden cards in the middle by default. Note that if you have revealed two hidden cards and neither of which is the nugget, then the remaining one is nugget (target).
2. **D**: the manhattan distance from starting point to the target position.
3. **dist**: the smallest manhattan distance from the current cards on the board to the target point.

2.2.2 Tiles

Tile cards represent tunnels that are created in the mine. Our goal is to use these cards to build a path connecting the starting point and the nugget point. We first divide tiles into two groups: tiles with unblocked path and tiles with a dead end.
For tiles with an unblocked path, The evaluation function is:

$$0.6 \cdot (0.25 \cdot w1 + 0.75 \cdot w2) \cdot dist, \{w1, w2\} \in \{1, 2\}$$

they have two weights: $w1$ and $w2$, where $w1$ results from a test that checks for the orientation of the path: for example, if the selected position is on the left side of our target while tile has a left orientation, then it will lead to an opposite direction, and vice versa. $w2$ represents whether placing the tile at a specific position will lead to a connected path from the origin.
For tiles with a dead end, we simply calculate $1.7 \cdot dist$ so that the agent will have a very small chance to choose such a tile card. Since it is not quite useful to use these cards. From the rules of this game we can see that when our opponent is building a path towards the goal, he is actually helping us to build the path. If we block the path built by our opponent, we are actually blocking our path as well. It is a game of both cooperation and competition.

2.2.3 Malus and Bonus

The evaluation function for Malus is:

$$0.7 \cdot dist - 0.01 \cdot \frac{dist}{D}$$

where $0.1 \cdot \frac{dist}{D}$ is the biased term to adjust its value, useful when there is no appropriate tile cards to choose.
Malus is a very useful card to limit the moves of your opponent, and it may be more useful when you are close to the nugget. Since near the end of the

game, there is less chance for your opponent to get a Bonus card to escape this situation, even if he has a Bonus card, you can still be one step faster than your opponent.
If an AI agent is in the "Malus" situation and he has the Bonus card, use it without any doubt. If AI is not in "Malus" , then we simply do not use it and keep it for further plan.

2.2.4 Map

The choice for Map is straightforward. If AI has not found the nugget, then as long as there is a Map in hand, we use it. In this case, we set its evaluation function as $-100 \cdot dist$ so that we give priority to this option. Note that we have found nugget if nugget is revealed or there is only one hidden card remains with the other two revealed as a cross path.
The Map card is one of the most important cards in this game, in particular, it is even the most valuable card at the very beginning of a game. Since it is important to know which direction we need to approach. For example, if we just blindly build a path and go down, while the final destination is just a normal cross path and the nugget is in the other two, then we may need at least one more step to reach the goal, which may be the reason for losing this game.
The value of Map will reduce as two players approaching the goal, it becomes not very useful once one of three hidden cards is reached. If you have reached one of the hidden cards, then you just need to go right or left in one or two steps.

2.2.5 Drop

The evaluation function for Drop is:

$$1.5 \cdot w \cdot dist, w \in \{1, 2\}$$

where w denotes the weight. It represents whether this card is useful. During our evaluation, we can drop the tiles with a dead end. Also, if the agent has more than 2 Bonus cards, we can keep 2 Bonus in the hand and drop the rest ones.
However, we do not expect our agent to choose "Drop" option when there are appropriate tile cards or map cards to play, so the base value of its heuristic is $1.5 \cdot dist$, which is bigger than heuristic of tiles (with unblocked path).

2.2.6 Destroy

The evaluation function for Destroy is:

$$1.3 \cdot w \cdot dist, w \in \{1, 2\}$$

where w denotes the weight representing whether this card should be destroyed ($w = 1$: can be destroyed, $w = 2$: should not be destroyed). In this game, tiles with a dead end are the cards we destroy. During our evaluation, tiles with a dead end are not quite useful. Moreover, sometimes these dead ends can make

the agent difficult to build a path towards the target. In this case, destroy the tile with a dead end on the board is necessary.
So the base value is $1.3 \cdot dist$, which is bigger than the heuristic of tiles (with unblocked path) and smaller than the Drop option.

3 Results and Analysis

The performance when playing against a random player is 7 wins out of 15 games on average. The highest rate of beating the random player is 9 games out of 15.

3.1 Advantage

- Each time we traverse all legal moves and compare their values of heuristics, and greedily choose the local optimal move.
- By controlling the orientation of the path, we can build the path towards the target position in the correct direction.

3.2 Disadvantage

- The greedy algorithm cannot always give a globally optimal solution. Thus the result of a game can be DRAW instead of WIN.
- The Random player still has a small chance to beat our AI agent, because we did not consider many options to limit the opponent's move. When we are close to the target, the opponent may take advantage of the path we build and reach the target first.

4 Other Approaches

We also tried to use the Minimax algorithm to do simulations and output the best next moves, instead of greedily choosing the (local) best move at each step. However, since the game will give us partially observable state space: we can not know which (new) card will be given in the next turn, we have to enumerate all possible cards and form a belief space, and the expanded Minimax tree will be too large to traverse. Even if using alpha-beta pruning and evaluation functions, the belief space for each node is quite large. Besides that, the Minimax algorithm makes an assumption that our rival will choose the optimum move, this is difficult in this game since our opponent is a random player. Therefore, we did not choose this algorithm.

5 Possible Improvements

- Although it is expensive to expand a Minimax tree deeply, we can still grow the tree for just two or four layers and use evaluation functions to estimate the performance for each branch. However, since we can not use the clone() method, we have to copy the entire board each time and evaluate the board state.

- Use machine learning techniques (logistics regression would be enough) to improve weights assignments in the evaluation functions.
- If we know the total number of every type of card, we could count how many cards are left for each type of card. Then we use the probability model and take account of the chance of getting different cards when choosing a move.