COMP 424 Final Report: Artificial Intelligence on Game Saboteur

Leyao Zhang
260761305
Jiayao Zhang
260767074
leyao.zhang@mail.mcgill.ca,
jiayao.zhang@mail.mcgill.ca

1 Introduction

Our goal of this project is to beat a random player on a board game Saboteur and create a path from initial position to golden nugget. Operations that players can do include placing a tile to create a path connected or disconnected, stopping opponent to play, destroying a tile and revealing a hidden objective. We can also drop any card in hand or flip a tile card. The game is non-deterministic and partially observable. State size of the game can be very large and complicated to analyze as the game goes on.

We first considered Monte Carlo Tree Search [1] as it learns from randomly generated samples and does not require us to explicitly define rules. As Saboteur has different types of card, we decided to apply MCT search on placing tunnel tiles only.

Besides the Monte Carlo Tree search, the player follows explicit rules when it holds cards other than tiles (i.e, map, bonus, destroy, etc.).

Rules adopted by StudentPlayer:

1. If opponent uses malus, the player uses bonus to against the malus if it holds a bonus. If the player does not hold a bonus, it is free to choose moves including
   a. map if nugget is not found,
   b. drop a tunnel tile with broken path,
   c. drop a map if nugget is found,
   d. or destroy a tunnel tile with invalid path on the board.
2. If gold is not found, use map to reveal a hidden tile.
3. Or use malus to stop opponent from playing if path is found or player does not hold a map.
4. If player's current hand does not satisfy any of the above situations, player drops tunnel tile that has a broken path or destroys tunnel tile on board that results in a broken path.

(a) (b) (c)

Figure 1: Example of tile with broken path.

The player randomly chooses a move from list of moves generated by one of the rules above according to different situation. If the player cannot decide the best move according to above rules, MCT search algorithm helps it to place a tunnel tile on the board.
In general, the winning rate of this player is approximately 45%.

2 Theoretical Basis

The MCT search we implemented follows Selection and Expansion of MCT search algorithm introduced in class. The changes and details will be discussed below. The following process is conducted until 900 milliseconds have been taken, which is a hand selected upper bound to avoid time out.

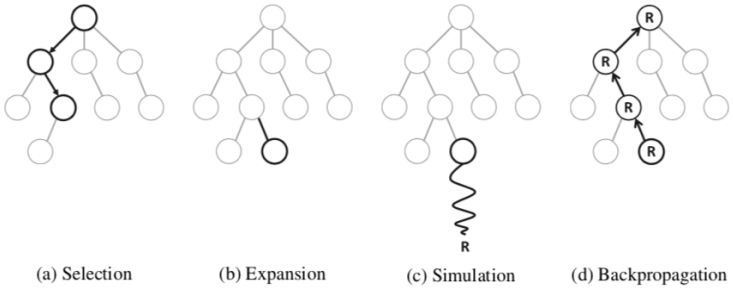

Figure 2: Basic idea of Monte Carlo Tree Search.

2.1 Selection

The player adopts tree policy to select a promising leaf node. At each node, if all children has been visited, each child is evaluated by calculating the upper confidence tree (UCT) value:

$$UCT(m) = v + c * \sqrt{\frac{\log N}{n}},$$

where v is the value (heuristic) of choosing move m in boardstate s, $c = 1$ is a scaling constant, N is defined as the number of times we have visited s in simulations, n is defined as the number of times we have chosen move m in boardstate s. The child with the highest UCT value is chosen, until a node with unvisited children has been reached.

2.2 Expansion

In this step, the program loops through list of moves of the leaf node to find a move that has not been taken. The move is processed on a copy of the current state and a new child corresponding to the move and the updated state is added to the leaf.

2.3 Simulation

From the newly added node, the game is simulated using the default policy until a path is built from the entrance tile to the nugget or 40 turns have been taken. The state is then evaluated by a heuristic function.

Heuristic function is defined as:

$$h(n) = \begin{cases} 1, & \text{if dist(G)=1, and player=StudentPlayer} \\ -1, & \text{if dist(G)=1, and player=Opponent} \\ \frac{10}{dist(G)}, & \text{otherwise} \end{cases}$$

where $dist(G)$ is defined as the distance from node n to hidden gold.

2.4 Backpropagation

The value and times of visited are updated for each node on the selected path.

3 Other Approaches

We have considered Minimax search over the complete state. Since the game states are stochastic and not fully observable, we have tried 2 approaches to represent the states. First is to enumerate all possible states given the current board, and after each move, all states generated is placed in a belief state object. This approach does a complete search in the state space theoretically. However, since the state space is large, Minimax search with alpha-beta pruning could only search 2 steps ahead in given time, and the performance of the agent was not satisfying. The other approach is to randomly select a state from the belief state and continue the searching from there. However, this is not an accurate approximation of the current state and thus the agent performed poorly. In addition, since there are many parameters to consider in a full state and they might not be independent from each other, writing a reasonable heuristic function is difficult, so we chose to only evaluate the board itself and encode other parameters by hand coded rules.
We also developed a player according to rules described in Introduction section(greedy player). Those rules are our strategy of playing Saboteur. It chooses a move that decreases the distance from lowest tile on board to the nugget. It performs well only when its opponent plays well and can be very unstable when it plays against random player.

4 Discussion

The MCT player performs steadily when against random player(45% winning), unlike the player with human-learned rules(20% winning). We suppose that it could learn implicit rules that are hard to observe and conclude by human, such as how to stop opponent from winning, and benefit from that.
The MCT player performs not as good if it plays secondly(3 win and 6 lose in 10 games against a greedy player). We do not know the reason of why it performs worse, but we suggest the reason can be that the greedy player is not optimal so it chooses a bad move first and results in loses of MCT player. Also, the reason can be that the MCT player is not able to reach total randomness and is not able to find the real best move that leads to winning, due to time limit.

5 Conclusion

Improvements can be made by analyzing the full state in the heuristic function. By using the full state in MCT search, the agent might be able to choose the optimal move in general for the

situation instead of acting according to hard-coded rules. Our agent is coded to randomly select a move from the dropping and destroying moves. If we let those moves be part of moves considered in MCT, the agent may be able to strategically destroy or drop a card. However, if the full state is used, after each move, several states would be generated. To select possible states in the belief state, it is helpful to introduce probabilistic models. For example, Bayesian networks can be used to calculate probabilities of each possible state given the tiles on the board and current player's hand. With those probabilities, the simulation carried out would be a better approximation of the remaining game.

References

[1] Steven James, George Konidaris, and Benjamin Rosman. An analysis of monte carlo tree search. In *AAAI*, 2017.