McGill | School of Computer Science

FACULTY OF SCIENCE

COMP 424 - ARTIFICIAL INTELLIGENCE

Project Report: Saboteur

Authors:

- Haoran Du (ID: 260776911)
- Cameron Cherif (ID: 260784819)

April 25, 2020

Contents

1 Introduction

The project is based on a on a specific version of a card game called Saboteur, which was originally introduced back in 2004. In this version, there are 5 types of cards available, namely Tile, Malus, Bonus, Destroy, and Map, on a restricted board of size (15,15). The goal of the project is to implement a computer agent with artificial intelligence called `StudentPlayer` defined in `public class StudentPlayer extends SaboteurPlayer`. The AI agent is supposed to beat another predefined agent whose move in the game is set to be random.

There are many constrains to our project, both from the scope of the project itself and several external factors. In the end, the result we obtained was satisfactory. We had a success rate of 33.33% from our testing rounds of play and we have satisfied all the requirements from the project specification document.

2 Technical Approach

2.1 Description

Our strategy is a modified version of the Local Optimization Search to head towards the gold nugget objectives, which is based on the knowledge of Hill Climbing and Simulated Annealing we learned in class. In short, our general strategy is to use the Map cards to find the location of the goal, drop cards that are deemed discardable along the way (to maximize the chances of having a favourable hand), and build a path with the best Tile card in hand, determined by the Local Optimization search, towards the goal.

There is also an idea of critical region which was integrated in the approach. The critical region in the game is defined to be the area below row 8. Inside the region, since it is close to the target, a special aggressive approach was chosen as using the Malus card whenever possible to reduce the winning possibility of the enemy. Thus, the setup of the critical region was not only to ensure that there are enough cards to build a path to the gold nugget with the help of the cards from the enemy in the beginning, but also to steer the enemy away from winning on the path we built or blocked the path.

The details of our approach is listed below in the same order as the logic of our `StudentPlayer` class:

1. Check if the enemy has played a Malus upon the player. If yes, use the the Bonus card in hand if possible.
2. If the location of gold nugget is still unknown, use the Map card in in hand if possible.
3. Drop the dead-end Tile cards in hand if possible to maximize the chances of having a hand of more utility. The dead-end Tile cards are defined as in
   `public static final ArrayList<String> deadEndTileNames`
   in the class constructor. The reason for this discarding move is based on the fact the enemy is completely in random. Hence, we believe it is better to discard those cards in exchange of a potential better hand rather than use those to block the path of the other, since the latter also increases the chance polluting the potential path down to the goal. The over 300 runs of the game we ran proved our prediction.
4. Use the Destroy cards in hand if possible to destroy all dead-end Tile cards currently on the board below the entrance. This is to maximize the chance of successfully reaching down to row 12.
5. Check the previous conditions. If there is still no viable option available, do as in Listing 1:

```
for (SaboteurMove mov : moves) {
  if (mov.getPosPlayed()[0] >= 5) {
    if (mov.getPosPlayed()[1] <= 7 && (mov.getPosPlayed()[1] >= 3))
      return moves.indexOf(mov);
  }
}
```

Listing 1: A simple greedy approach when the gold location is unknown.

6. Repeat previous steps until the gold nugget location is revealed.
7. Once the gold location is known, discard the Map cards in hand.
8. Check if the cards has reached down to the critical region (below row 8). If not, perform simple greedy approach in Listing 1.
9. If reached the critical region, play the Malus card in hand at once. As stated before, this is a special aggressive approach to reduce the winning possibility of

the enemy and to reduce the chance that the planned path is being blocked a dead-end Tile.

10. Next, perform enhanced greedy algorithm in Listing 2 where playing the cross-shape and vertical-stroke shape Tile cards has higher preference in the move.

```
1       // the + and | tiles in the critical area have priority
2       boolean hasTileVer = false;
3       for (SaboteurCard card : cards){
4           if (card.getName().equals("Tile:8")
5           card.getName().equals("Tile:0")
6           card.getName().equals("Tile:6")
7           card.getName().equals("Tile:6_flip")){
8               hasTileVer = true;
9               break;
10          }
11      }
12      if (!hasTileVer) {
13          for (SaboteurMove mov : moves) {
14              if (mov.getPosPlayed()[0] > 8) {
15                  if (Math.abs(mov.getPosPlayed()[1] - goldCoord[1]) <= 1)
16                      return moves.indexOf(mov);
17              }
18          }}
19      else{
20          for (SaboteurMove mov : moves){
21              if (mov.getPosPlayed()[0] > 8 && mov.getPosPlayed()[0] < 12
22              && Math.abs(mov.getPosPlayed()[1] - goldCoord[1]) <= 1
23              && (mov.getCardPlayed().getName().equals("Tile:8")
24              mov.getCardPlayed().getName().equals("Tile:0")
25              mov.getCardPlayed().getName().equals("Tile:6")
26              mov.getCardPlayed().getName().equals("Tile:6_flip") ))
27                  return moves.indexOf(mov);
28          }}
```

Listing 2: Enhanced greedy approach

2.2 Motivation

There were several other methods that were discussed before the writing of the code, including turning the game into a Constraint Satisfaction Problem (CSP), similarly to the 4 queens problem we studied. However, due to the complexity of the Saboteur game, we believed that the time required for the CSP approach's implementation outweighed its potential performance.

There was also an attempted approach by Haoran Du of forming the game into a Minimax tree. In the end, the team decided to switch to the Local Search Optimization approach in the end based on the following reasons:

1. There is a total of 56 cards distributed among the players, both starting with 7 cards. Thus, there are only 42 turns, or 21 rounds to discard cards that do not fit one's strategy. Furthermore, only two Malus cards are present in the deck, for twice as many Bonus cards. It increases the chances of countering a Malus, which is unfavourable for an aggressive strategy. However, 6 Map cards are in the deck, thus composing almost a ninth of the deck, favouring a greedy strategy. Three Destroy cards can also compose the deck, which allow the destruction of a tile placed on the board, favourable for path building when one of the 9 dead-end tiles available in the deck are placed. The remaining cards are tunnel tiles permitting route building in various directions.

2. We believe that in order to beat a random agent, the best move would be perform a Local Optimization each time after the enemy has played instead of planning a path from the beginning. We don't suppose a very complex approach or would have a lot more advantages than a greedy algorithm when facing something random.

3. The enemy is completely random rather than rational. Hence, it is almost impossible to make predictions to the next move of the enemy based on which card was played in the last round. This would make several approaches very hard, for example calculating heuristics of the move, or deciding the min/max agent in $\alpha - \beta$ pruning, if also considering the complexity of the moves or tiles.

Further, another notable path that has been sought, although ultimately abandoned due to the AI's unsatisfactory behaviour following its implementation, was Cameron Cherif's attempt at applying simulated annealing.
The goal was to make a function that would decrease from round 1 to 42, representing the probability of discarding a card versus placing a tile. A similar function for the probability of placing a tile has been made.

For f(x) and g(x) being the probability of discarding a card and of placing tile respectively:
$f(x) = -x^2/1764 + 1$
$g(x) = x^2/1764$
The implementation of simulated annealing has been done as follows:

```
1    boolean canDiscard = boardState.getTurnNumber() < 42;
2    if(canDiscard) {
3      double probabilityOfDiscarding
4      = -Math.sqrt(boardState.getTurnNumber()
5      + numberOfCardsDiscarded - numberOfTilesPlaced
6      - discardInHand.size()) / 1764 + 1;
7      double probabilityOfPlacing
8      = Math.sqrt(boardState.getTurnNumber()
9      - numberOfTilesPlaced + numberOfCardsDiscarded + usefulTiles.size()) / 1764;
10     //choose whether to discard an undesired card probabilistically
11     if ((probabilityOfDiscarding
12      >= probabilityOfPlacing && !justDiscarded
13      && !discardInHand.isEmpty())  usefulTiles.isEmpty()) {
14         justDiscarded = true;
```

Listing 3: Simulated Annealing

2.3 Theoretical Basis

2.3.1 Hill Climbing

Hill climbing is a very powerful method using greedy algorithm to determine the local optimization of a problem. It basically compares the outcome of all its available neighbors and choose the best among them. It has the problem of stuck in a local maximum. In this project, all the cards in hand of the agent is considered neighbors. We used the local greedy search to choose the one that has the best chance to reaching down to the (potential) gold location. The details can be seen in Listing 1 and 2.

2.3.2 Simulated Annealing

Simulated Annealing is an improved version of Hill Climbing. It allows random non-greedy moves to escape the local maximum. In out problem, the default return move is chosen randomly from the list of all moves available. Also, the malus card is only used once inside the critical region, which is a form let the random player doing the process of Simulated Annealing.

2.3.3 Divide and Conquer

A divide-and-conquer algorithm works by recursively breaking down a problem into sub-problems, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem. In this project, the use of the critical region divides the game into two sub-problems. The first one is that when being outside the critical region, we allow bad moves and simply going down using Listing 1. The second one is that when being inside the critical region, we use a more dedicated approach to move as close and as fast to the gold tile as possible. This idea of Divide and Conquer makes the solution easier to approach.

3 Summary of Results

The result we obtained was satisfactory. We achieved a success rate of 33.33% from our over 300 rounds of testing. we have satisfied all the requirements from the project specification document. The RAM we used is lower than 500 MB. The calculating time of our algorithm is smaller than 2 seconds. We did not use File I/O in out program.

4 Reflections and Future Improvements

The implementation of this local optimization search approach permits a rapid progression and a fast recon of the coordinates of the goal. Furthermore, the different scenarios, depending on whether the goal is known and whether the critical region has been reached, allow the AI agent to choose among the different aspects of the greedy and aggressive strategy accordingly. Using a modified greedy algorithm as well as multiple conditions for analysis of the board and its development, speed and goal-achievement

are favoured simultaneously. Additionally, discarding inconvenient cards quickly greatly increases one's chance of having a good hand early on in the game.

However, the program does not fully take into account the Destroy cards, which can be extremely advantageous when the opponent's strategy is to sabotage our path progression. Moreover, grid analysis for road finding is not used, thus giving the disadvantage of not being able to predict a path towards the goal, rather only building a tunnel using the legal moves available in direction of its general location. The rudiment nature of the greedy algorithm, although bringing the advantage of speed and best move selection for the current turn, makes the possibility of a long-term approach unlikely to be opted for. For example, if the all the openings are blocked by the adversary in the first rounds, the AI agent does not seek an alternative path from the side, rather building away from the goal. Ultimately, the biggest disadvantage of the program is the likeliness of never reaching the goal, regardless of whether its location has been revealed, due to the implementation of tile placement.

Accordingly, if time permits, we believe there are improvements could be done to drastically improve the performance. We could improve our greedy algorithm by adding more response based on the opponent's move. For example, if the enemy blocks a planned path leading to the gold nugget, a destroy card could be applied. Another issue could be improved is that, inside the critical region, we can add more calculation on best path leading to the tile based on the feature of the Tile cards. In that way, there would less erroneous placed cards in our path.

Above all, a remarkably advantageous implementation in the future would be behaviour analysis of the opponent's gameplay, which would permit carrying out an adaptive strategy. As aforesaid, multiple types of strategies exist, and can be combined in different ways to obtain distinct behaviours. A human player would adapt their tactical approach in response to the adversary's, for instance using "Destroy" cards and putting dead-end tiles in a "rusher"'s path, or playing defensively against an aggressive player, etc. Thus, mimicking this evolutionary trait that is plasticity, and incorporating it in the AI, shall be a tremendous step towards enhancing its performance.

Students' contributions

Both of student partners worked together to understand the problem and write the code in this project.