Scotland Yard Project Report

Du Guoyuan kx18412
Zhu Yuqing fm19121

1. Introduction

CW-Model passed all tests with high readability and reusability.
CW-AI based on Min-Max DFS, Alpha-beta pruning and Dijkstra algorithms.

2. Content

2.1 Using Lambda, Stream and Collectors

For all lists and sets iteration, boxing and unboxing, we used Lambda, Stream and Collectors.

```
1. nextRemaining.addAll(detectives.stream()
2.         .filter(this::isTicketExists)
3.         .map(Player::piece)
4.         .collect(Collectors.toList()));
```

2.2 Using Immutable Builder

```
1. everyone = new ImmutableList.Builder<Player>()
2.                              .add(mrX).addAll(detectives).build();
```

2.3 Correct way of using Optional class

Using “orElse()” or “orElseGet()” instead of using “get()”

```
var destination = destinations.get(destinationIndex);
return moves.asList().stream().filter(destination::equals)
                              .findFirst().orElse(defaultMove);
```

3. AI Implementation

3.1 Dijkstra algorithm

We use Dijkstra to calculate distance (i.e. shortest path) for any two points in the map and save them in two-dimensional array. Dijkstra is done ONLY ONCE for all points at the first run due to our singleton (i.e. single instance) design pattern for MRX AI class.

3.2 Min-Max DFS

We introduce Min-Max to iterative search tree with a strategy of finding maximum value for MRX and minimum value for detectives. Min-Max is a pessimistic algorithm, we always assume that detectives could do the best decision in their perspective i.e. minimize value in MRX perspective.

3.3 Optimization for DFS

3.3.1 Alpha-beta pruning

Pruning by setting upper bound “alpha” in maximum (MRX) level and lower bound “beta” in minimum (detectives) level. If found alpha is greater than beta, do prune.

3.3.2 Feasible pruning

For MRX, do pruning if distance between MRX and any one detective is less than or equal to two.

For detectives, filter some obviously unwise decisions.

3.4 For more effective DFS

Ticket consuming system is added in DFS in order to make the simulation of Min-Max algorithm more realistic and effective. This is done by backtracking and our customized class “Player” and “Destination”. Customizing class makes code more readable and reusable in developing AI.

3.5 Value evaluation for DFS

We considered two dimensions of value evaluation:

1. shortest distance between MRX and all detectives and weighted by proximity.
2. Number of optional destinations for MRX

```
public int evaluate(int source){
    AtomicInteger weight = new AtomicInteger(detectives.size());
    List<Integer> distances = detectives.stream()
```

```
4.              .map(x->distance[source][x.position])
5.              .sorted(Comparator.naturalOrder())
6.              .map(x->{
7.                  if(x < 3) return 0;
8.                  return x * 100 * weight.getAndDecrement();
9.              })
10.             .collect(Collectors.toList());
11.     double choiceNum = getOptionalDestinationsForMrX(SINGLE_MOVE).si
    ze() / 2.0;
12.     return (int)(distances.stream().mapToInt(x -> x).sum() * choiceN
    um);
13. }
```

3.6 Timing of using DOUBLE and SECRET tickets

Since we are using Min-Max algorithm, we actually don't care about the timing of using SECRET tickets to trick detective players, SECRET tickets are only used for Ferry.

For DOUBLE tickets, we evaluate value at the beginning of every round, if value is less than a threshold value, double move destinations will add to MRX's optional destination list, thus double move will be considered by our Min-Max algorithm.

4. Critical reflection

In this project, I tried some concept in functional programming e.g. Optional class, lambda, map, filter, boxing and unboxing, and used them in practice.

I also got familiar and tried many OOP design pattern such as Visitor, Observer and Singleton pattern, and knew when and how to use them flexibly.

For AI algorithms, it is not the first time I write a game AI, I am already familiar with DFS, pruning and Dijkstra.

I learnt a lot by reading other part of the source code when I was completing Model and AI. For instance, I used to use Interface for callback function or anonymous internal class when I wrote Java or Android programs, now I can use them more flexible.