Softwarepraktikum SS 2021
Assignment 1 Report

Group 3

Jamie Anike Heikrodt	394705	anike.heikrodt@rwth-aachen.de
Thomas Pollert	406215	thomas.pollert@rwth-aachen.de
Jascha Austermann	422571	jascha.auster@rwth-aachen.de
Silyu Li	402523	silyu.li@rwth-aachen.de

Contents

Exercise 1

1.1 Map 1

This map:

- has the size of 8*8.
- contains **two** players, and each player has **two** override stones and **four** bombs with the explosion radius of **four**.
- has **one** inversion tile, **two** bonus tile, **three** expansion tile and **three** choice tile.
- has **one** special transition.

1.2 Map 2

This map:

- has the size of 10*15.
- contains **three** players, and each player has **four** override stones and **five** bombs with the explosion radius of **two**.
- has **one** inversion tile, **two** bonus tile, **two** expansion tile and **three** choice tile.
- has **one** special transition.

1.3 Map 3

This map:

- has the size of 25*25.
- contains **five** players, and each player has **three** override stones and **four** bombs with the explosion radius of **one**.
- has **two** inversion tile, **three** bonus tile, **four** expansion tile and **three** choice tile.
- has **one** special transition.

Exercise 2

2.1 Discussions

- How to represent the map, the different field and the transitions.
- How to efficiently calculate the neighbouring tiles of a tile.

2.2 Solutions

- In this task we used the **2-dimensional array** to represent the coordinate system of the map, which was the first idea came to our mind.
- Use **Hashmap** of type (x,y,direction) -> Tile(Neighbour) to store the extra transitions given by the server.
- **First thought**: Define a class **Tile** with several subclasses to represent different types of tiles, which allows good modification possibilities.
  Improvement: Use of enumeration class **TileTypes** combining with the class **Tile** with several methods and attributes.
- **First thought**: Calculate the neighbouring tiles by calculating the (x,y) coordinates of them, which would not work so well for special transitions.
  Second thought: Store for each tile its 8 neighbours, which would cause too much memory overhead.
  Improvement:Firstly check the Hashmap if there are special transitions, then calculate with (x,y) coordinates.

Exercise 3

3.1 Discussions

- When is a move valid?

3.2 Solutions

- After we found the logical requirements of a valid move, we needed to check given moves on a given map for these with an algorithm.
- We came up with a boolean type algorithm that decides if a given move is valid. **Therefore** we check if the coordinates the move is done refers to a valid tile in the game. If were in the second phase, the move is valid. If the tile the move is performed on is player-occupied already, we checked if the moving player has enough override stones. Then we check if theres a path in any direction of player-occupied Tiles to a Tile occupied by the moving player. If there is, the move is valid, otherwise it isn't because no stones are turned.

Exercise 4

4.1 Discussions

- Given a map and a valid move, how to efficiently implement a method that calculates the successor map.
- Given a map, how to efficiently calculate all the possible fields for the next stone.

4.2 Solutions

- **First thought**: Go through all the tiles and check if this tile is suitable, but this would be too inefficient.
  Improvement: There are only two cases, we place either an override stone on an occupied tile or a normal stone on an unoccupied tile, which must have at least one occupied neighbouring tile.
- Use of **Array List** to save the empty tiles with occupied neighbours and the occupied tiles, which simplifies the calculation.
- Specify bomb/override for bonus fields and player number for choice fields by adding a choice variable in the **move class**. Different characteristics make up different moves for us, which we also take into account when listing the successors.

Exercise 5

5.1 Building Phase

Heuristic: We thought that a position should be viewed as "good" if we have big diffrence in occupied Tile to the players under us and a low difference to the ones over us. Having players above you in the ranking is "bad", having them below you is"good". We thought that its "better" to play against opponents close to you in the ranking(to save your position from players trying to pass you in the ranking and to pass players above you).
Therefore we calculate a value for a given map and player(us) by:

- counting the Tiles occupied by every player.
- Now we calculate the diffrence between our number of occupied tiles and every other.
- We added a factor (distance factor) that leads to playing against opponents close to you in the ranking.The basic idea is that we factor each summand with either $1 - \frac{|difference(i,j)|}{totalDifference}$ for player j above us in the ranking and $\frac{|difference(i,j)|}{totalDifference}$ for player j above us.
- a value is calculated for every opponent: calculating the distance, calculating the distance factor, switching the sign if the player viewed is above us.
- all these values are summed

The heuristic we chose aims to get our player as high as possible in the ranking. To achieve that we try to minmize the distance to opponents above us and maximize it to the opponents below us. The player easiest to pass in the ranking is the closest one above us, thats why the distance to their stone is more important to the result value, after all we want to be as high as possible in the ranking, which is not the same as having as many stones as possible. The player most likely to pass us in the ranking is the closest one below us. Thats why the distance to them is of greater importance then the distance to opponents far away.

5.2 Elimination Phase

Heuristic: We use the same heuristic. But before calculating a move, we watch if the distance to any player is greater than

$$\text{totalbombs} \times (2{\times}\text{BombRadius})^2$$

These players are not in our reach. We ignore them when calculating the result of a game. Its not good to destroy someones tiles if we cannot reach them until the game ends.
Note: that is not really part of the heuristic, it justs defines the set of players relevant for calculating result.