# Project Programmation 2 : Report part 1

Adrien Bardes, Enguerrand Dezerces

March 6, 2018

## 1 The game

When the game starts, towns are generated at random positions and with a random number of citizen. The player is in charge of a train company and he can do the following actions :

— Build a station in a town (there can not be more than one station per town)

— Build a rail between two towns having a station

— Build a train in a town having a station

Each of these actions cost money The player can also click on a town or a rail in order to get extra information about it The player has access to the list of trains and can click on one of the elements of the list to get extra info about the train

## 2 Architecture of the project

The project is divided into 4 packages :

— engine : The core of the game where the logic is executed

— interface : All the classes used for the graphical user interface

— link : Utility classes in order to make the link between the engine and the interface

— utils : Utility classes that can be used everywhere in the project

### 2.1 Engine

The towns graph is contained in the object *World*, each node is an instance of the abstract class *Town*, and each edge is an instance of the class *Rail*. The game progresses by tick, at each tick, the world state is updated and the changes are sent to the interface. The world contains instance of the class *Company*. The rails and the trains belongs to the company. For now there is only one company, owned by the player.

When a town is updated, it generates a percentage of passengers which are sent to the neighbouring towns, according to the number of available trains. The town contains an instance of the abstract class *Station*. The station manages the available trains. When there is no train available for a specific destination, the station keeps in memory the number of waiting passengers for this destination. When a station tries to send passenger to a destination, it takes the first train available for this destination and it loads it with the passengers. When a train is updated, it checks its destination and checks if it is arrived. If so, it calls the method unload of the class station, on itself. If not, it moves toward its destination.

### 2.2 Interface

The interface uses the *ScalaFX* library. It is divided in objects, each object manages a specific part of the interface. *GUI* is the main object, it is also the main class, indeed it extends *JFXApp* (a scalaFX class) therefore it is launched first. The other components are the following, they all inherit from the class *GUIComponent* and all have a method make that allowed building them

— *ItemsButtonsBar* : Manages the buttons used to select an item to build (Station, rail, train)
— *AllTrainsInformationPanel* : The list of all the trains in the map, each train is represented by a button
— *GlobalInformationPanel* : Global information about the world
— *LocalInformationPanel* : Information about an element (A rail or a town) of the world is displayed there when the user clicks on it.
— *OneTrainInformationPanel* : Information about one specific train when the user clicks on it directly on the map or on the list of trains

### 2.3 Link

We use the observer, observable design pattern. In the link package are all the classes used to make the link between the engine and the interface. Elements of the engine are observables and elements of the interface are observers. When a change occurs in the engine, it is putted in a list of changed. At each tick, the interface is notified about the changes.

### 2.4 Utils

All utilitarian classes

## 3 Shortcomings

— For now there is only one type of train, all trains have the same characteristics.
— There is also one type of city, station and rail.
— We can set a destination to a train with the interface but it actually doesn't do anything.