Introduction

The project this year entails development of a simple tycoon game in the Scala programming language. The object of the game will be to manage and expand a transport company (much like games such as the Transport Tycoon or Railroad Tycoon series). The game will be played on a world represented by a graph (nodes are towns and edges are routes connecting them). Apart from the aspects of transport network management, the game will contain advanced simulation of economy, AI competitors, save and load features, and possibility to display various performance statistics. Various aspects of the game will motivate you to proper use of OOP.

1 Requirements

For the first part of your project you will be developing a graphical interface for the game alongside some of the basic functionality, such as creating a new train. The program should satisfy the following requirements:

- Simple and intuitive GUI including the following:
  - Ability to start a new game and quit the program.
  - Visualisation of the graph representing the game world, including extra information (name, population, waiting passengers etc.) displayed when a node ("town") is selected, and extra information (endpoints, length etc.) when an edge is selected.
  - List of trains owned by the player, plus the ability to select one of the trains for extra information (current location/movement, engine and carriages etc.).
  - Ability to create a new train (for now, it is enough to have one type of engine and one type of passenger carriage).
  - Ability to plan a route for the new train (for now, it is enough to connect just two towns).
  - A display of current in-game time and money owned by the player.

- The game should progress automatically, by ticks. The game status is updated at every tick.

- The graph of the world should contain at least 10 nodes.

- Each town should generate passengers based on its size.

- Trains should generate income based on the number of passengers they are carrying (so far it is enough to receive flat rate for passenger/distance).

- Routing for trains should try to find the shortest route.

Try to use scalable object-oriented code where possible. This will make it easier for you to include future improvements.

2 Evaluation Criteria

2.1 Report and Presentation

You will have to submit a short (2-3 pages) written report (ideally \LaTeX-generated PDF). In the report you should explain the implementation challenges you encountered and describe your solutions. In case your program has any defects or shortcomings, list them in the report explaining the reason for their existence. Each group will deliver a 10 minute presentation of their project at the end of phase 1.
2.2 Functionality

Naturally, the functionality of your program will play a major role in its evaluation. Bonus features on top of the ones requested in this document may yield bonus points. The quality of graphical user interface (GUI) will also be taken into account. Of course, this is not a UX design course, so it will be evaluated from the functional perspective, however, intuitiveness and ease of use will be taken into account.

2.3 Code Organisation

Your project must be organised hierarchically, e.g. code split into files, classes and methods. Try to separate functionality as much as possible. It is especially advised to separate the data layer (business logic) and the user interface.

2.4 Code Quality

Proper use of OOP and the Scala language are important criteria for evaluation. Pay attention to proper use of inheritance. Avoid code duplication. Prefer functional constructs over nested loops/if statements. Stick to the naming conventions of Scala, if at all possible. Try to name your variables/methods/classes etc. with names that well describe their role in the program. In case you use a construction whose purpose is not obvious from the code, please document this in a comment.

3 Important Dates

- Deadline for submitting the code and the report: **Tuesday 06/03/2018**.
- The presentation of phase 1: **Friday 09/03/2018**.