Part B - Metro Madness

SWEN30006, Semester 1 2017

Overview

Continuing your work as software contractors, you have been contacted by the head of the Melbourne Metro Rail Redevelopment Project to help them out of a tricky spot. Melbourne Metro had previously hired the *Shoddy Software Development Company* to build a simulation of their train network to aid in the decisions around where new stations are build and how to scale existing infrastructure to meet Melbourne's growing transport needs.

Unfortunately *Shoddy Software Development Company* has done a rather... *Shoddy* job. They have provided a poorly designed system which, although it functions as a metro train simulation, is not flexible or easily modifiable. The developers did not follow good design principles when they implemented their simulation, nor did they consider the future requirements of the simulation.

Your task is to first provide a detailed analysis of the existing software package, highlighting inappropriate design decisions and providing documentation of the existing Train state simulation. Having completed this design, you will then be required to redesign the existing package applying good design principles and patterns to improve the flexibility of the simulation platform to cope with Melbourne Metro's changing needs. Finally, having improved the design, you are to extend the design to support some additional functionality and modify the existing simulation to match your modified design.

The Existing Simulation

The current simulation provides the following features:

- Reading Train Maps from XML to setup simulation.
- Simulating Multiple Lines with Multiple Stations and Trains.
- Simulating Trains travelling on the network.
- Simulating Trains entering and leaving Stations.
- Simulating Single and Double tracks (with locks on access).
- Simulating Passengers entering and exiting the network at Active Stations.
- Simulating Passengers travelling on a single line journey.
- Detailed log output describing behaviour of Trains and Passengers
- 2D render of the train network at any given point in time (see details later).

This is good enough for Metro Melbourne's current needs, however it unfortunately cannot model some constraints on trains stopping at stations. Further, the existing simulation only allows one of two types of trains:

- A train that can carry 10 passengers.
- A train that can carry 80 passengers.

This does not allow Metro Melbourne to vary the train sizes to simulate behaviour of passengers on the network with varying train sizes.

Controls

The current simulation provides camera controls to navigate around the train network. Specifically, it supports the following desktop controls:

- Arrow keys to pan around (up, down, left and right).
- Zooming in and out using Q and A respectively.
- Exit using E.

LibGDX

The previous developers have built the existing simulation using the LibGDX game development framework. They did this for two reasons: first LibGDX allows for easy visualisation of the train network for demonstration purposes, and second it allows for compilation on Android and iOS so that Metro representatives can demonstrate on the go.

LibGDX as a framework is mainly used for gaming purposes, but in this case, it is only being used for its OpenGL rendering support. The simulation **does not** use any of the networking, controller or audio libraries. As already noted, the current simulation provides a 2D render of the train network at any given point in time.

Gradle

The provided simulation makes use of Gradle as a build and dependency management tool. Using Gradle avoids the need to manually link any of the libraries or dependencies within the project, and it is much easier to migrate between development environments and IDEs. In order to use Gradle within Eclipse, you must do two things:

- Install the Eclipse Gradle integration by following the installation guide at listed here.
- Import the Gradle project into Eclipse using the following LibGDX Guide.

Once imported you should see three projects: `Metro Madness`, `Metro Madness-core` and `Metro Madness-desktop`. These three projects combine to make the one simulation. `Metro Madness` contains the Gradle build scripts for the entire simulation. `Metro Madness-core` contains the core simulation logic within 6 packages. Finally, `Metro Madness-desktop` contains the Desktop launch configuration along with the fonts and maps required for the simulation.

Your first task should be to import the project and build the application to ensure that you have the existing simulation running without issue. When you first run the simulation, you may need to adjust the view: try using the `A` key to zoom out, and then the arrow keys to pan or the `Q` key to zoom in.

The Task

There are three main components to your task, first an analysis of the existing design, second a new improved and extended design for the simulation system, and third an implementation of that new design. We will now break these components down in detail.

Analysis of Existing Simulation

You are required to complete two tasks as part of your existing design analysis.

First, the design documents provided by *Shoddy Software Development Company* unfortunately lack a state diagram for the Train class. This makes it quite difficult to understand the behaviour of this Train class and therefore more difficult to modify. Metro Melbourne would like you to do is to provide a State Machine diagram that reflects the current implementation of the Train class so that they may more accurately understand the provided implementation. This must reflect all states of the train, and all transitions the train can take.

Your second analysis task is a design analysis report on the provided simulation. You should focus on the design patterns used (if any) along with general object oriented code quality. You should use the analysis

exercises from workshop 4 as a guideline and focus on the GRASP patterns, on how well the provided code follows these patterns, and on where it does not.

In this report you should be critical of the existing software, but ensure that you backup your claims and statements with reasoned arguments about the design. Simply stating something is bad is not sufficient for this task. Neither is saying that your modified code (see below) is better. You should aim for between 1200 and 1500 words for this report.

Updated Design

Having now critically analysed the provided package, your next task is to provide a refactored software design that addresses the concerns listed in your report. Further to this, Metro Melbourne have asked you implement an additional feature, based on issues with passengers traveling with significant amounts of cargo. The system needs to be able to support a new kind of train, a CargoTrain. These trains will only be able to stop at cargo stations (which have the required facilities), and will pick up passengers up their passenger capacity (as usual), and accompanied cargo up to their cargo capacity: SmallCargoTrains have the passenger capacity of a SmallPassengerTrain with a max cargo capacity (set to 200kg for testing purposes) and LargeCargoTrains have the passenger capacity of a LargePassengerTrain with a max cargo capacity (set to 1000kg for testing purposes). Passengers at cargo stations will have a random amount of cargo up to 50kg, and will have a cargo station as their destination. Passengers at non-cargo stations will not have any cargo.

As part of your updated design you are to complete this new feature. This new feature (including relevant methods and data structures) should be included in your new design documents. You will be provided with sample log output to further illustrate the required changes; you need to ensure that the log output for your revised system is consistent with the sample log output. You are welcome to modify the visualisation to represent cargo trains, however this is optional.

You will also need to modify the XML map (`Metro Madness\desktop\assets\maps\melbourne.xml`) to change the type of some stations to "cargo" for testing purposes. The XML map already includes a cargo train, as this was a planned feature.

Your should provide, as part of your new design, the following diagrams:

- A design class diagram of all components, complete with visibility modifiers, associations and all methods.
- A design sequence diagram showing the behaviour of a train entering and leaving a station, covering both the case where it stops and the new case where it doesn't stop because the station is not a cargo station.

These diagrams should be consistent with each other, and with the updated implementation (see below).

Implementation

Finally, you must provide an updated Simulation that corresponds to your new design. This new implementation must be entirely consistent with your new design, and will be marked on code quality, so should include all appropriate comments, visibility modifiers and functional decomposition.

Building and Running Your Program

Your program must be importable using the same instructions we have provided you to run the project. We will be testing your application using the desktop environment.

> **Note** Your program **must** run on the University lab computers. It is **your responsibility** to ensure you have tested in this environment before your submit your project.

Submission Checklist

1. State Machine Diagram demonstrating the operation of the Train class.
2. Design Analysis Report.
3. New Class Diagram reflecting new design for the Train simulation.
4. New Sequence Diagrams reflecting the new design for the Train simulation.
5. An updated source code package reflecting your new design.

Marking Criteria

This project will account for 10 marks out of the total 100 available for this subject. These will be broken down as follows:

Criterion	Mark
Correct State Machine Diagram for the provided Train class	1 mark
Design Analysis Report for Provided Simulation	4 mark
Updated and Improved Design Documentation including new cargo feature	3 marks
Implementation of new design	2 marks

We also reserve the right to award or deduct marks for clever or poor implementations on a case by case basis outside of the prescribed marking scheme.

Further, we expect to see good variable names, well commented functions, inline comments for complicated code. We also expect good object oriented design principles and functional decomposition.

Finally, you are expected to follow UML 2.1+ syntax. You will lose marks for incorrect UML diagram notation.

If we find any significant issues with code quality we may deduct further marks.

On Plagiarism

We take plagiarism very seriously in this subject. You are not permitted to submit the work of others under your own name. This is a **group** project. More information can be found here: (https://academichonesty.unimelb.edu.au/advice.html).

Submission

You should submit one zip file containing both the updated code package and the report and state machine diagram. Your reports should be submitted as a PDF. You must include your group number in all of your pdf submissions, and as a comment in all source code files provided as part of your project.

Only one member from your group should submit your project. Detailed submission instructions will be provided separately.

Submission Date

This project is due at **11:59 p.m. on Tue 25th of April.** Any late submissions will incur a 1 mark penalty per day unless you have supporting documents. If you have any issues with submission, please email William at tiow@unimelb.edu.au, before the submission deadline.