Assignment 5: A Graphical User Interface for the Traffic Simulator

Submission date: April 30th, 2018, 09:00am

Objective: Model-View-Controller design patters, Graphical User Interface with Swing.

1. Overview

In this assignment, we will build a graphical user interface for the traffic simulator. We will do so by first modifying the traffic simulator to use the MVC design pattern, and then build a swing GUI that allows interacting with the simulator.

IMPORTANT: some parts are optional, all are marked with the word **OPTIONAL**.

2. Traffic Simulator with MVC

We will first change the traffic simulator to support observers, which will be able to register to be notified whenever the state of the simulator changes.

The simulator should notify (at least) when the following occurs:

- `REGISTERED`. When an observer registers, that specific observer, and only that one, should receive a corresponding notification with the current state of the simulator.
- `RESET`. When the simulator is reset, all observers should receive a corresponding notification with the current state of the simulator.
- `NEW_EVENT`. When new events are added to the simulator, all observers should receive a corresponding notification with the current state of the simulator.
- `ADVANCED`. When the simulator advances the time, all observers should receive a corresponding notification with the current state of the simulator.
- `ERROR`. When an error occurs, all observers should receive a corresponding notification with the current state of the simulator and the corresponding error.

3. Graphical User Interface

Build a Swing GUI for the traffic simulator. The GUI should look similar, not necessarily identical, to the one depicted in Figure 1. Below we provide more details on the different components.

3.1. Events Editor

The events editor is the area where the user can edit events. It is simply a text editor, using `JTextArea` for example. The interface should provide the user with the following functionality:

- *load* a file into the text editor.
- *save* the current content of the text editor to a file
- *delete* the current content of the text editor.

Note that we provide you with an example of a text editor in Swing. You should adapt it to your needs.

3.2. Events Queue

The events queue lists the current events in the simulators queue. Each line should include the scheduled time and a description of the corresponding event. Use a `JTable` or `JList` to implement this component.

3.3. Reports Area

The reports area is simply a text-area where reports are written. We have the following ways for generating reports:

- `Generate all reports on demand`. The GUI should provide a way to generate all reports of the current state (e.g., by pressing a button) and write them into the reports area.
- `Selected reports on demand` (**OPTIONAL**). The GUI should provide a way to select vehicles, roads, and junctions and generate reports for them. Selecting the simulated objects for which reports should be generated can be done using one of the following (1) a dialog window like the one of Figure 3; (2) selecting them from the corresponding tables using `getSelectionModel()` of `JTable`; (3) any other reasonable option you can think of. We provide you an example of how to make a dialog window for this purpose. **If you implement this option do not implement the previous one**.
- `Reports generated by the simulator` (**OPTIONAL**). The output of the simulator (i.e., what it writes to its output stream) should be redirected to the corresponding text-area. To achieve this functionality, you should implement an `OutputStream` that simply writes to the corresponding `JTextArea` when its `write` methods are called, and ask the simulator to use it as an output stream. The user should also have the possibility to disable redirecting the report events output to the text area (see the simulator menu Figure. 2) – this can be done simply

by setting the simulator's output stream to `null`. **If you do not implement this part, then set the output stream of the traffic simulator to `null` right from the beginning**.

The GUI should also allow:

- Saving the current content of the reports area to a file (**OPTIONAL**);
- Deleting the current content of the reports area.

3.4. Simulated Object Tables

The GUI should include tables, using `JTable` with corresponding `TableModel`, for vehicles, roads, and junctions. The information to be included is similar to those of the reports. You have freedom here, you can add whatever information you feel necessary and use any syntax that is easily understandable.

3.5. Road Map

A swing component that draws the road map. We provide you a similar component for drawing a graph (with dots on edges, like vehicles). All you need is to replace the `Graph` data structure by a `RoadMap`.

3.6. Status Bar)

The GUI must include a status bar (or an information area in general), where corresponding informative messages are shown when the user interacts with the GUI. In Figure 1, the status bar is at the bottom.

3.7. Interacting with the Simulator

The GUI should provide the following 3 operations for interacting with the simulator:

- `Check-in events`. It adds the current events available in the events editor the simulator. Note that you can create an `InputStream` from a string `s` as follows: `new ByteArrayInputStream(s.getBytes())`.
- `Run` it runs the simulator for some time units (the user should have the possibility to specify the time unit, e.g., using a `JSpinner` as in Figure 1)
- Reset the simulator.

3.8. Tool and Menu Bars

The toolbar buttons and menu options basically allow the same operations to be performed, and you only need to implement *one* of the two corresponding bars. We recommend that you implement *both* bars, but doing so is considered (**OPTIONAL**).

Let us explain the toolbar elements from left to right: load events file, save events file, clear events area, check-in events, run the simulator, reset the simulator, the number of time units to advance when running the simulator, a label indicating the current time of the simulator, generate reports, clear reports area, save reports, and quit.

3.9. Event Templates (OPTIONAL)

To facilitate the way the user type events in the events editor, add a popup menu (activated with mouse right-click in the events area) that includes a list of events, and when the user selects an event a template is added to the event area at the position of the cursor (see Figure 4). We provide you with an example of a popup menu similar to what you have to implement.

4. The Main Class

You should modify the main class to support starting the program in a batch-mode (assignment 4) or gui-mode. We will add an argument `-m` that can take values '`batch`' and '`gui`' and starts the application in the corresponding mode. If the default value of the option, i.e., if it is not provided, is '`batch`'.

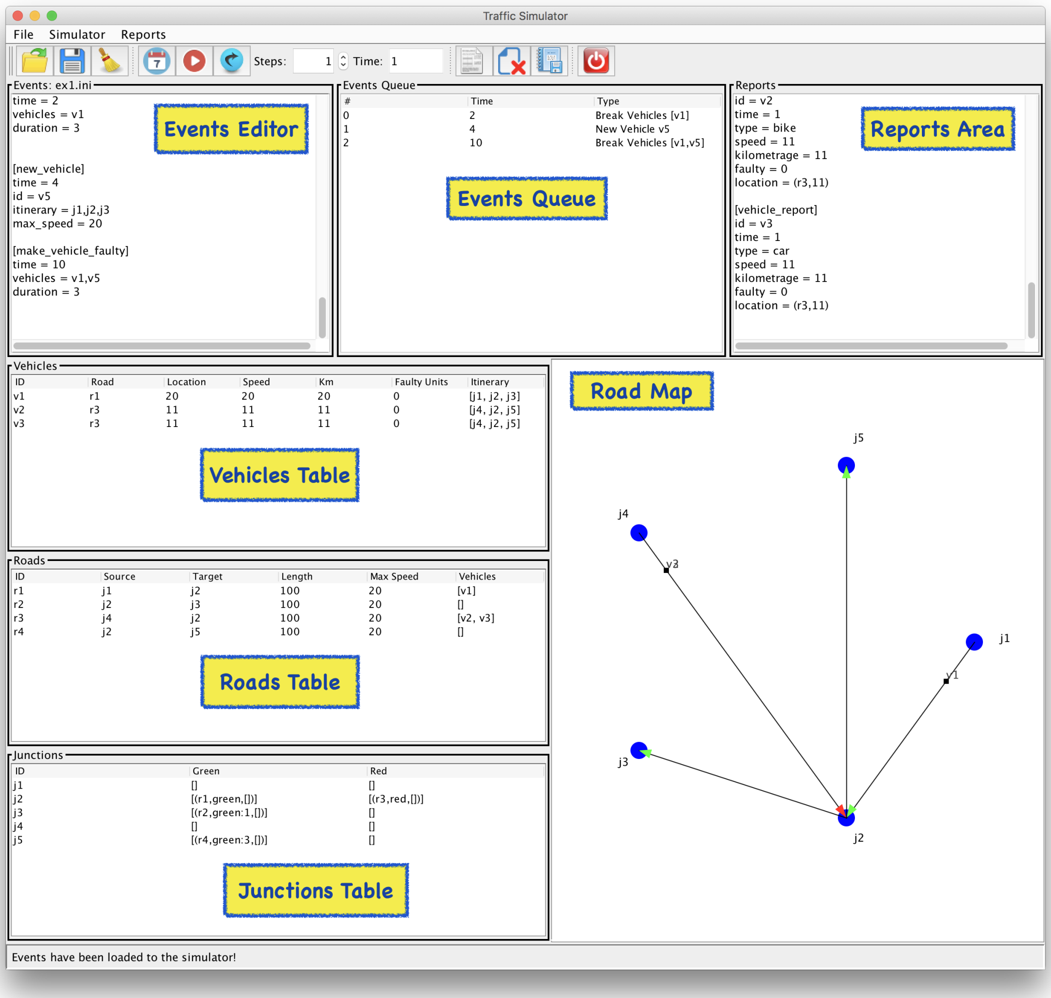
```
> java Main -h
usage: Main [-h] [-i <arg>] [-m <arg>] [-o <arg>] [-t <arg>]
 -h,--help            Print this message
 -i,--input <arg>     Events input file
 -m,--mode <arg>      'batch' for batch mode and 'gui' for GUI mode
                      (default value is 'batch')
 -o,--output <arg>    Output file, where reports are written.
 -t,--time <arg>      Time units to execute the simulator's main loop
                      (default value is 10).
```

In `batch` mode, the behaviors should be exactly as in assignment 4. In `gui` mode, the following should be supported:

- If an input events file is provided, using the `-i` option, it should be automatically loaded to the events-area when the GUI is created;
- Options `-o` and `-t` are ignored.

5. Further Comments and Requirements

- **It is strictly forbidden to use tools for building GUIs, you should write all code in Java.**
- We provide example code for: (1) text editor; (2) dialog window; (3) graph layout; and (4) popup menu.



Events: ex1.ini

```
time = 2
vehicles = v1
duration = 3


[new_vehicle]
time = 4
id = v5
itinerary = j1,j2,j3
max_speed = 20

[make_vehicle_faulty]
time = 10
vehicles = v1,v5
duration = 3
```

Events Queue

#	Time	Type
0	2	Break Vehicles [v1]
1	4	New Vehicle v5
2	10	Break Vehicles [v1,v5]

Reports

```
id = v2
time = 1
type = bike
speed = 11
kilometrage = 11
faulty = 0
location = (r3,11)

[vehicle_report]
id = v3
time = 1
type = car
speed = 11
kilometrage = 11
faulty = 0
location = (r3,11)
```

Vehicles

ID	Road	Location	Speed	Km	Faulty Units	Itinerary
v1	r1	20	20	20	0	[j1, j2, j3]
v2	r3	11	11	11	0	[j4, j2, j5]
v3	r3	11	11	11	0	[j4, j2, j5]

Roads

ID	Source	Target	Length	Max Speed	Vehicles
r1	j1	j2	100	20	[v1]
r2	j2	j3	100	20	[]
r3	j4	j2	100	20	[v2, v3]
r4	j2	j5	100	20	[]

Junctions

ID	Green	Red
j1	[]	[]
j2	[(r1,green,[])]	[(r3,red,[])]
j3	[(r2,green:1,[])]	[]
j4	[]	[]
j5	[(r4,green:3,[])]	[]

Events have been loaded to the simulator!

Figure 1: The Graphical User Interface. Yellow boxes are annotations, not part of the GUI.

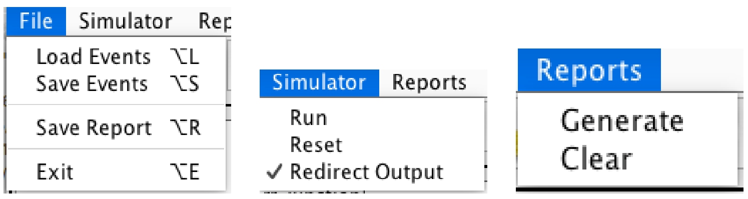
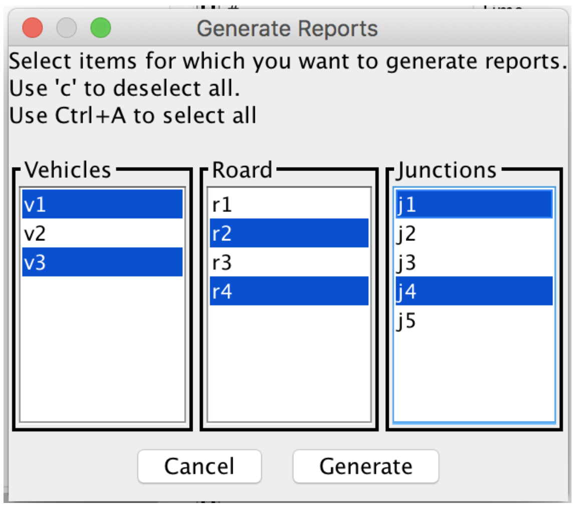


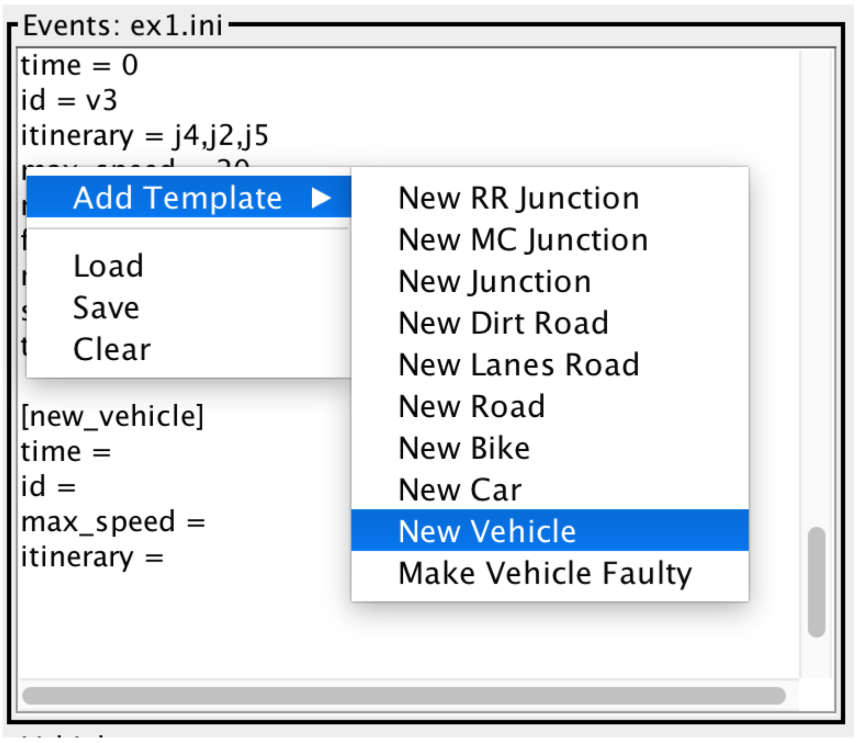
Figure 2: The different menus in the menu bar.



Select items for which you want to generate reports.
Use 'c' to deselect all.
Use Ctrl+A to select all

Vehicles	Roard	Junctions
v1	r1	j1
v2	r2	j2
v3	r3	j3
	r4	j4
		j5

Figure 3: A Dialog for selecting simulated objects.



```
time = 0
id = v3
itinerary = j4,j2,j5

[new_vehicle]
time =
id =
max_speed =
itinerary =
```

Figure 4: Popup menu for adding event templates.