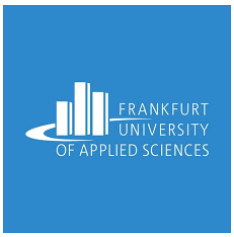
FRA-UAS
DEPARTMENT OF COMPUTER SCIENCE



REAL-TIME TRAFFIC SIMULATION WITH JAVA

SiMpFi

Instructor: Prof. Dr.-Eng. Ghadi Mahmoudi

Student 1: Kenji Bischoff - 1402720
Student 2: Nguyen Ha Khai - 1650329
Student 3: Nguyen Duy Khanh - 1648306
Student 4: To Huynh Phuc - 1651780
Student 5: Ninh Hoang Quan - 1650910

Frankfurt am Main, 11/2025

Abstract

This report provides progress, purpose, and other detailed informations about simpfi, a real-time traffic simulation app in which users can simulate real-world transportation scenarios and export useful statistics.

List of Tables

List of Figures

Contents

1 Introduction

1.1 Context

SUMO - Simulation of Urban MObility - is a traffic simulation tool used to simulate urban transportation scenarios. The application is helpful for designers, researchers who design road network and evaluate features such as CO_2 levels, traffic congestion rate, etc.

1.2 Main Issue

Although providing useful functionalities for traffic simulation, SUMO is difficult to utilize effectively. In other words, only users with programming knowledge can benefit greatly from the software.
This disadvantage can be observed by the fact that users are required to grasp a basic understanding of SUMO's configurations such as XML files and how to run command lines.

1.3 Purpose and Goals

In order to address the problem mentioned above, the team aims to create a software that leverages SUMO's features but maintains the simplicity of user interface. That is, people can perform simple operations such as drag-and-drop and clicking buttons to retrieve meaningful detailed information and represent complicated traffic networks. Consequently, more and more designs can be sketched and created quickly, which potentially speeds up one's studies or transportation planning. Furthermore, an increased number of designs means that the urban area may have better transportation plans, thereby reducing traffic congestion and CO_2 emissions.
Regarding study goals, the team plans to master in Object-Oriented Programming concepts and gain more experience with teamwork, writing documents such as javadoc and academic reports.

2 Project Overview

2.1 Description

The software **simpfi**, which stands for **Si**mulation **Ma**p Traf**fi**c, is a traffic simulation tool used to create and monitor simulated representation of real-world networks. For clarity, the application's core is based on SUMO, but the difference is that it is developed to suit a wider range of target users from non-coders to experts.

2.2 Technology Stack

There are three main components the team uses for the project: **Java**, **TraaS**, and **SUMO**. Additionally, the team uses **GitHub** to create a shared repository for members to submit codes, and **Notion** to distribute tasks.

2.3 Team Roles

The below table shows the main role of each member during the milestone 2, note that although these are specific-domain roles, the team tends to overlap the work and responsibilities in order to fully experience how to work in real-world projects:

ID	Name	Main Role
1402720	Kenji Bischoff	Implement Inspect Panel
1650329	Nguyen Ha Khai	Implement Filter Panel
1648306	Nguyen Duy Khanh	Implement Program-Lights Panel
1651780	To Huynh Phuc	Implement Map-View Panel
1650910	Ninh Hoang Quan	Implement Statistics Panel

Table 1: Team Roles

2.4 Time Plan

For the detailed time plan, readers can see our activities in this Notion page: `https://www.notion.so/2a62b60367bb80a58886c6713e08ba54?v=2a62b60367bb80db8153000caff77c02&source=copy_link`

3 Architecture Diagram

Here is the architecture diagram of the project:

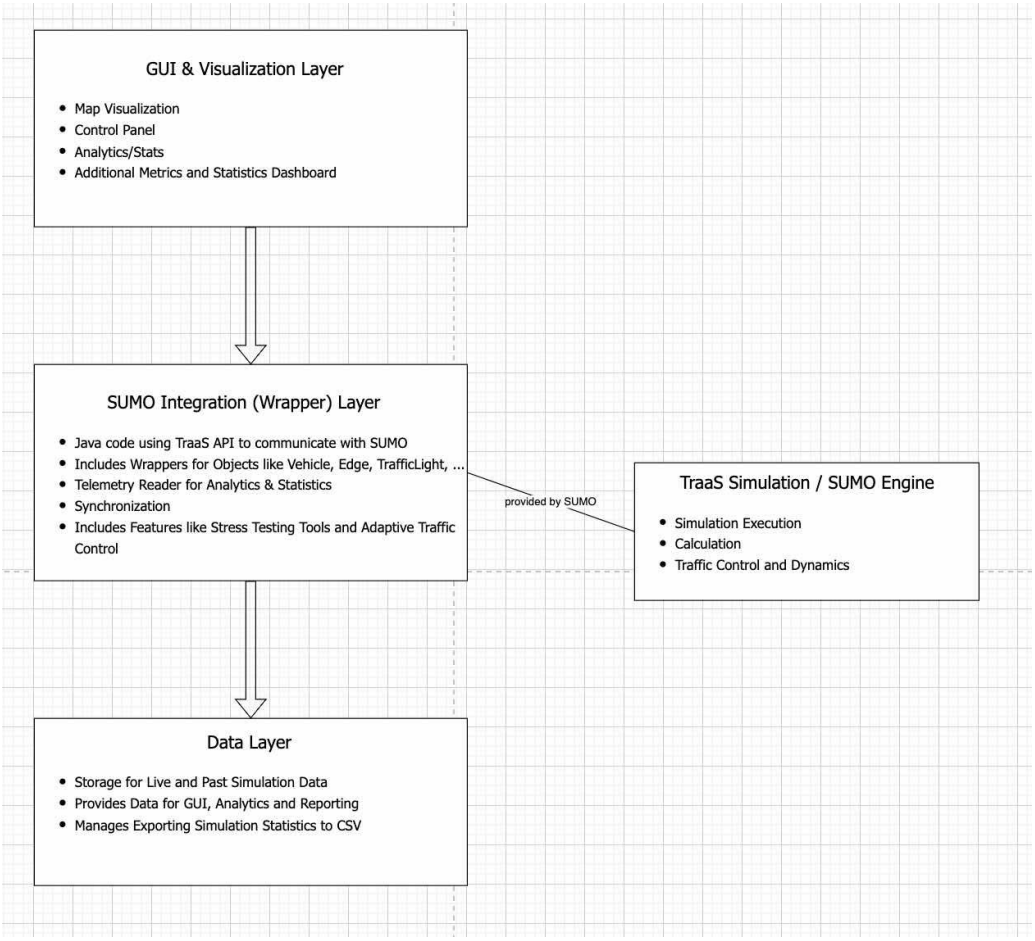


Figure 1: Architecture Diagram

From the diagram, three main layers can be observed: GUI & Visualization, SUMO Integration, and Data. Each layer provides separate methods in order to guarantee modularity and coherence in the system's design.

3.1 GUI & Visualization Layer

The main task of this layer is to present an user-friendly UI for users. In particular, it includes map visualization, multiple panels such as **Inject** so that users can add vehicles to the network. Additionally, users can move and scale the map by dragging and scrolling the mouse respectively.

3.2 SUMO Integration (Wrapper) Layer

The layer is where TraCI connection takes place. Put differently, it is the interface where SUMO and wrapper classes communicate with each other. From there, the simulation can be executed, logical operations like numerical calculations can be performed, and network controls can be handled flawlessly.

3.3 Data Layer

As its name suggests, this layer handle data-related operations such as live storage and past storage, which is essential for exporting meaningful statistics. Also, the layer is developed to ensure the consistency and integrity of data throughout different parts of the system.

4 Class Design

4.1 Overview

Besides core wrapper classes, the team decided to draw the user interface by defining and connecting classes such as buttons, panels,... As a result, the number of classes was up to 30, which would be excessively complicated and difficult to understand if the team sketch them in one single class diagram. Therefore, a wrapper class diagram was designed to ensure the conciseness and readability.

4.2 Wrapper Class Diagram

The class diagram is shown as follows:

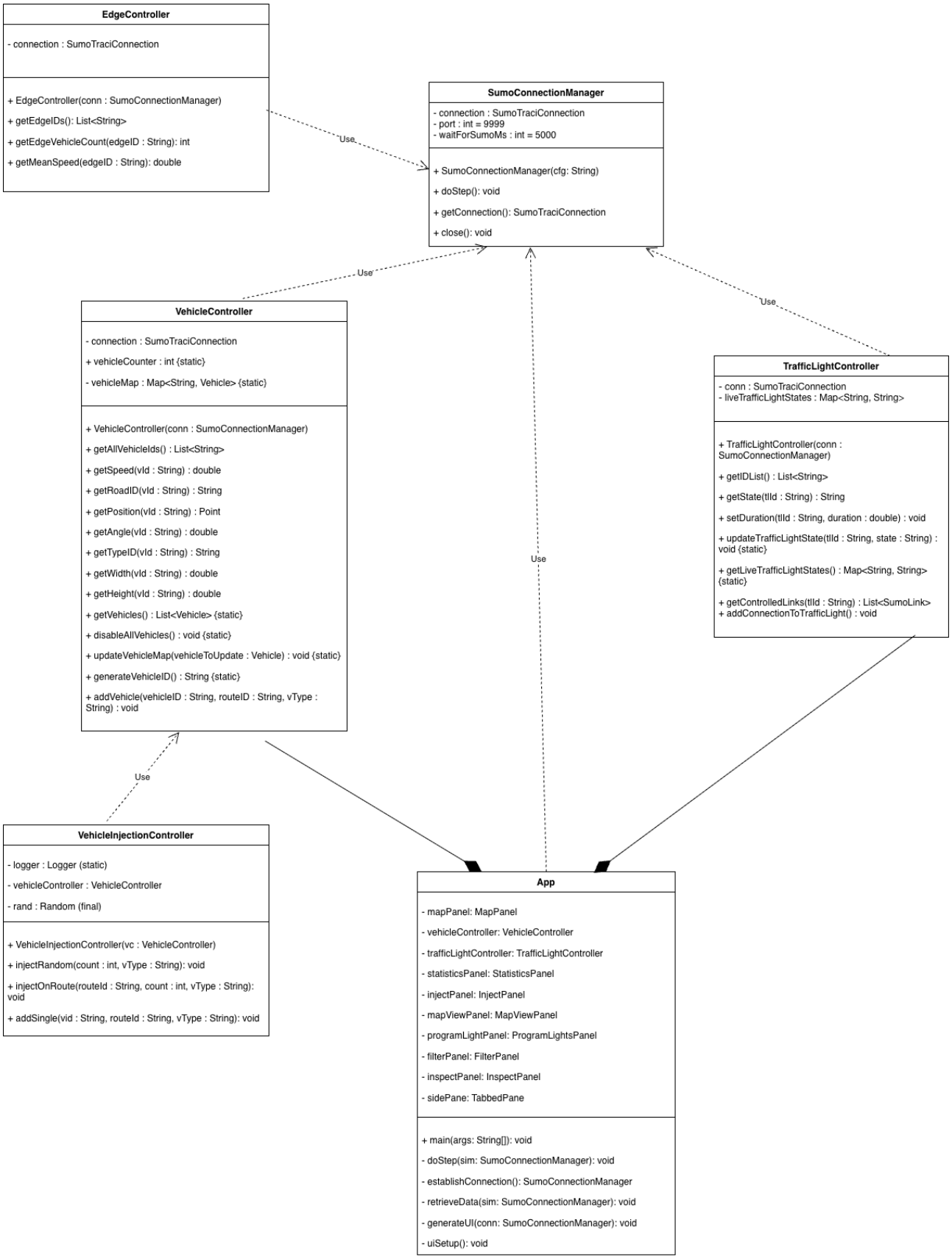


Figure 2: Wrapper Class Diagram

4.3 Class Responsibilities

- **EdgeController**: gets the information of edges and vehicles to calculate meaningful statistics such as average speed on a specific edge.
- **VehicleInjectionController**: controls different types of injections like batch, batch random or single mode.
- **VehicleController**: its main role is to process and retrieve detailed information regarding vehicles. To achieve this, the class has a `SumoTraciConnection` attribute `connection` used to actively connect to SUMO so that getters such as `getSpeed()`, `getRoadID()` can be implemented. Additionally, `vehicleCounter` and `vehicleMap` are developed to keep tracks of vehicles number and mapping. Furthermore, the class provides several useful methods like `addVehicle()` for users to perform vehicle-related operations.
- **TrafficLightController**: as its name suggests, the class deals with traffic light objects, especially in the real-time scenarios since it has a live traffic light mapping: `liveTrafficLightStates`. Similar to `VehicleController`, the class establishes a connection with SUMO using the attribute `connection` and therefore is able to implement getters such as `getIDList()`, `getState()`. Besides, `updateTrafficLightState()`, `getLiveTrafficLightStates()` are provided to maintain the logic of SUMO traffic lights in the Simpfi's map.
- **SumoConnectionManager**: is the class that creates and manages the TraCI connection between SUMO and the project's app. This is achieved by passing a XML configuration file to its constructor and then the program can initialize the connection by `getConnection()` method and close it safely later using `close()`. Moreover, it provides an *important* method `doStep()` which advances the simulation by one timestep.
- **App**: is where the main function resides. Particularly, it connects the frontend (`mapPanel`) and the backend (`vehicleController`, `trafficLightController`, ...); defines the TraCI connection; and handles the connection loop so that the software can function properly.

5 GUI Mockups

Below is the intended final user interface of the project:

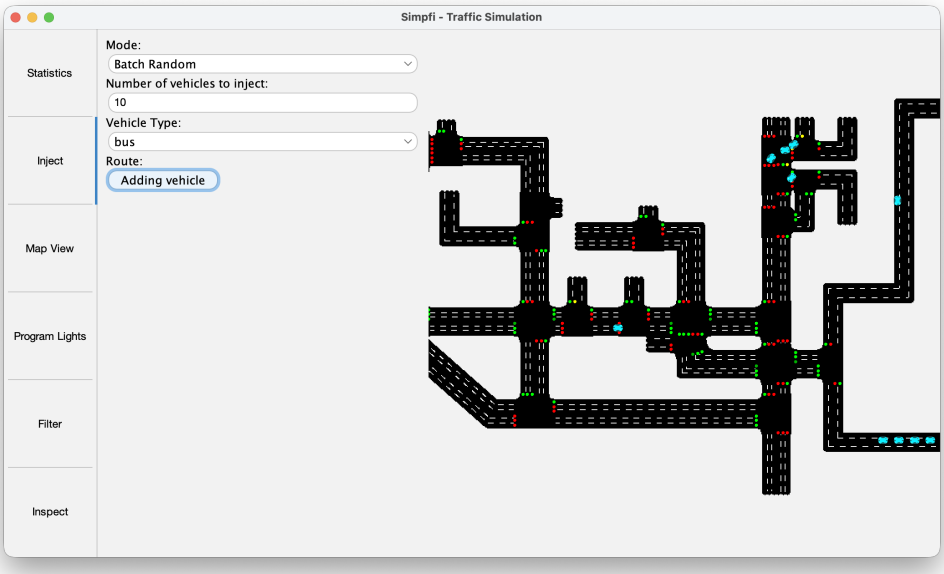


Figure 3: User Interface

6 User Guide

6.1 Introduction

6.1.1 Overview

This application provides a simple graphical interface for controlling, visualizing, and analyzing SUMO traffic simulations. Users can inject traffic, program traffic lights, and monitor simulation statistics in real time.

6.1.2 Target Audience

Our project is aimed at users working with traffic and urban mobility simulations, such as:

- Designers.
- Transportation Researchers.
- Environmental Analysts.
- Students.

6.1.3 Features

- Real Time Map Visualization.

- UI including:
  - Vehicle Injection
  - Vehicle Inspection and Filtering
  - Traffic Light Programming
  - Statistics Panel
  - Map View settings

6.2 Setup

6.2.1 Software Requirements

1. Java 17 (JDK) or higher.
2. SUMO software.
3. IDE for Java.

6.2.2 Installation Guide

Make sure that you download the versions compatible with your operating system.

1. Install Java Development Kit (JDK from oracle).
2. Install SUMO (Simulation of Urban MObility).
3. Install an IDE of your choice to run the Java project (Eclipse, VSCode, IntelliJ, ...).
4. Install `flatlaf-3.6.2.jar`, `jcommon-1.0.23`, `jfreechart-1.0.19`, `TraaS`.

6.2.3 Project Setup

The last step of the setup is to download our project by cloning it from GitHub. After that, open it up in your IDE.
For running the application, simply run the class "App.java".

6.3 User Handbook

The application window will be divided into 2 main sections: (1) Map Panel on the right side and (2) Control Panel on the left side.

Figure 4: Navigating UI

The **Map Panel** can be manipulated in 2 ways:

- Move around with arrow keys, zoom in with "Ctrl =" and zoom out with "Ctrl –" (Not recommended since arrow keys can also be used to move around in the Control Panel and differences between US keyboard layout and German keyboard layout makes zooming more difficult).
- Move around by clicking and dragging the map area, zoom in by scrolling up and zoom out by scrolling down (Recommended).

The **Control Panel** have six sub-panels with different features:

6.3.1 Statistics

Shows a variety of charts that is updated in real time with statistics from the running simulation:

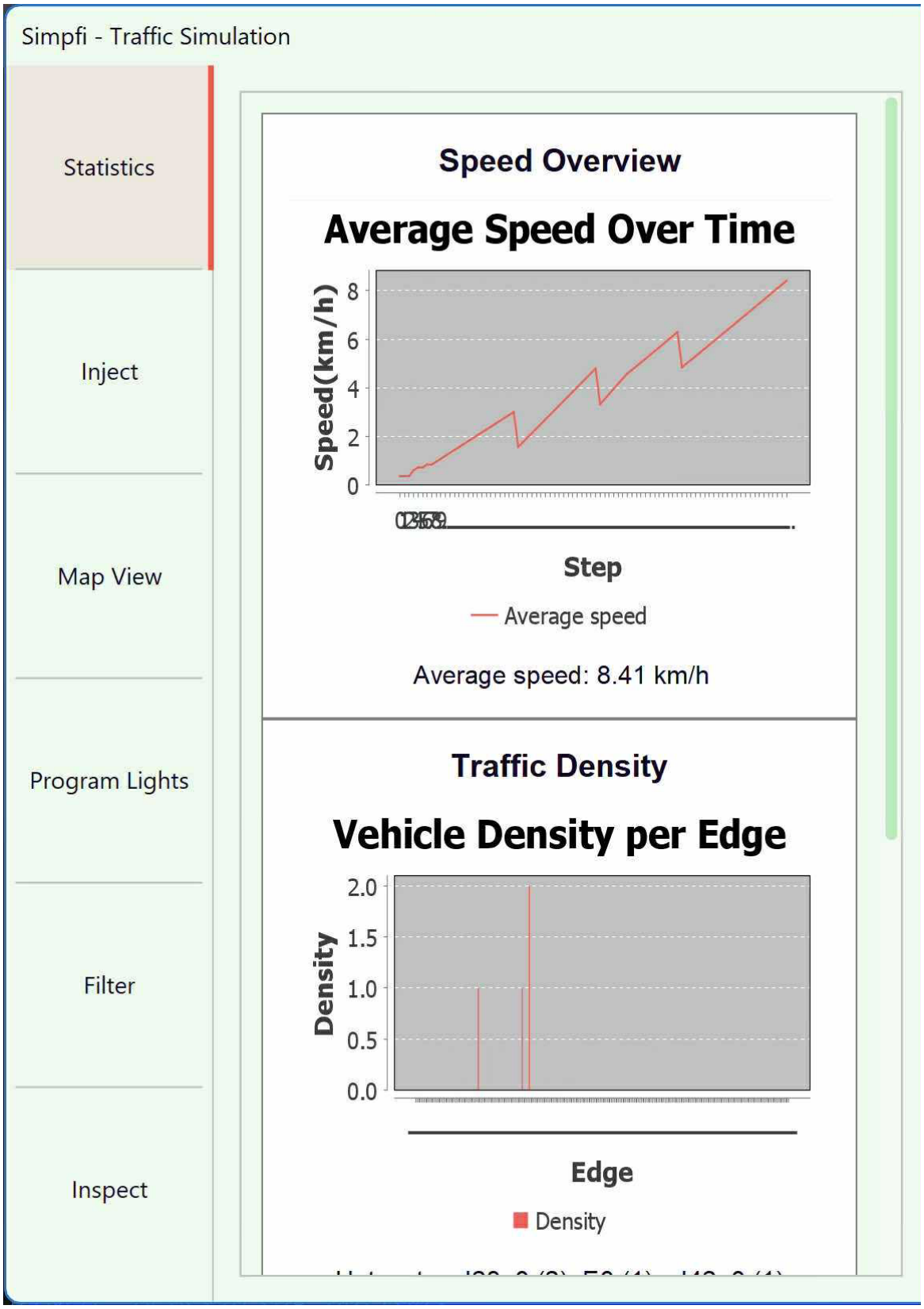


Figure 5: Stastistics Panel

6.3.2 Inject

In the Inject Panel, there are four Dropdowns and one button to add vehicle to the network:

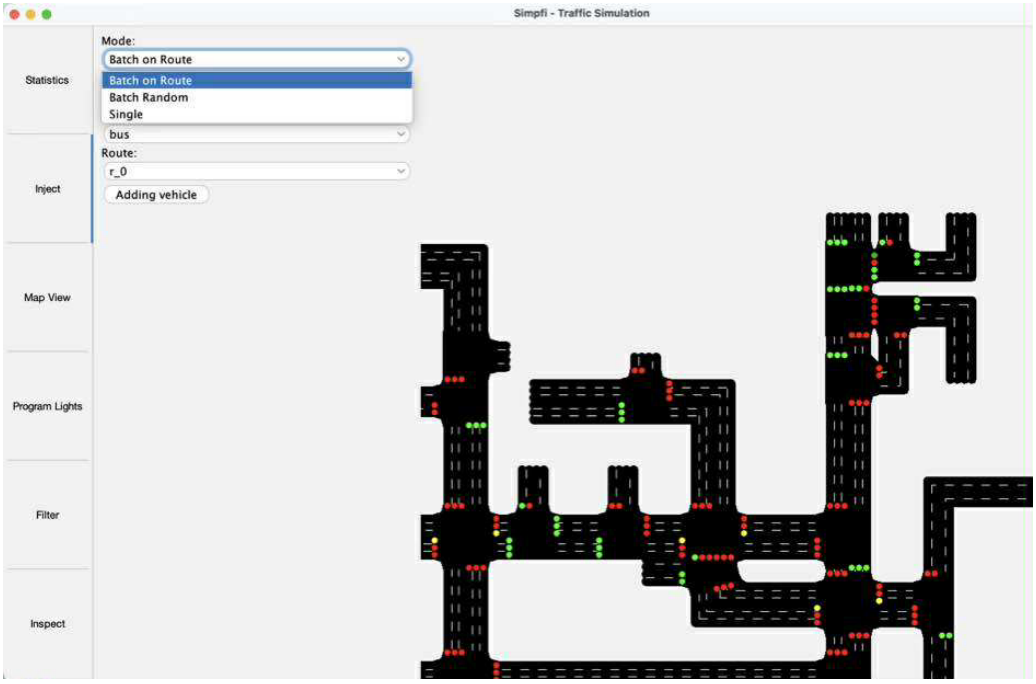


Figure 6: Inject Panel Interface

Before injection, users have to choose one of three modes: batch on route, batch random, and single. Batch On Route means that users can inject any number of vehicles they want on a specific route chosen from the Route Dropdown. For the Batch Random mode, users are allowed to add any number of vehicles to random routes. The last mode is Single mode, for which users can only inject one vehicle on one route.

After choosing the mode, users have to set a value indicating the number of vehicles that they want to inject. This Dropdown only appears when a chosen mode is Batch On Route or Batch Random. The third Dropdown is Vehicle Type, where users can choose one of five vehicle types. Each vehicle is colored differently depending on its type. Next, users have to specify which route they want their vehicles to run. This Dropdown appears only when a mode is Batch On Route or Single. Finally, users press the button "Adding vehicle" to start the process.

6.3.3 Map View

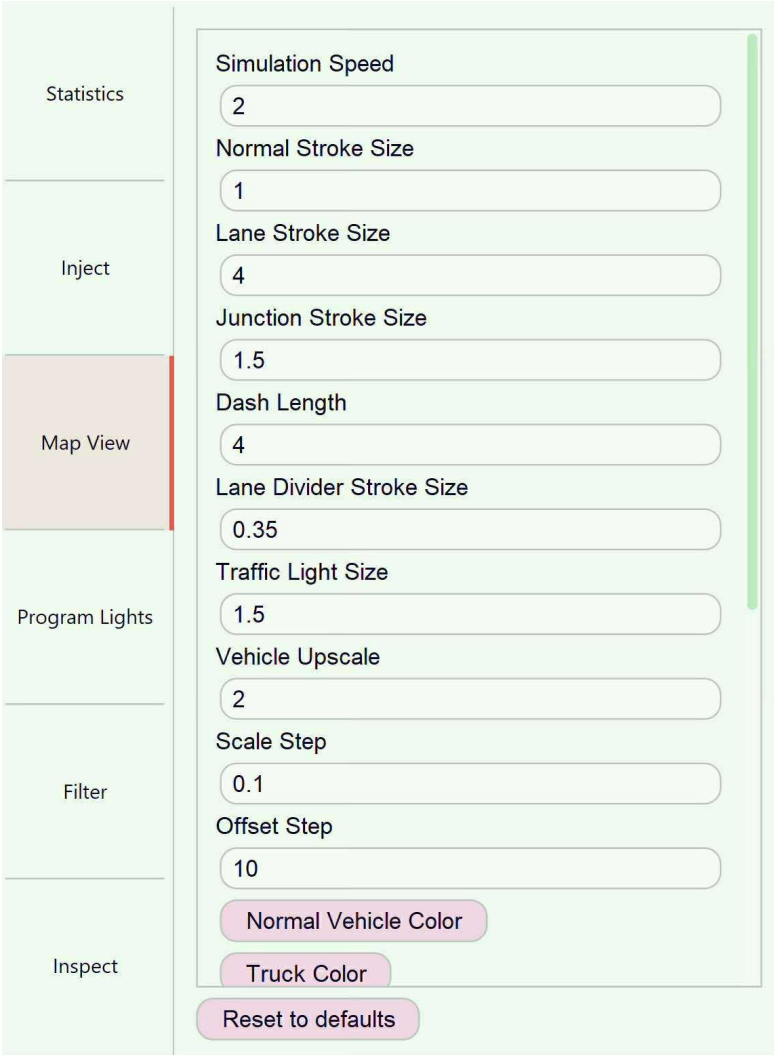


Figure 7: Map View Panel Interface

In the Map View panel, user can change many settings about the display of the map such as how big the elements are or what color is used. **Note** that set the "Simulation Speed" too high is not intended and can have unexpected consequences.

6.3.4 Program Lights

In this panel, users can interact with traffic lights in the network. To start with, users need to identify the traffic lights they want by selecting the traffic light ID in "Select intersection".

After a intersection is chosen, one of its connections is highlighted. The intersection between two lanes connection is the junction that users choose. If users want to see other connections, simply select them in "Connection". For example, another connection becomes highlighted after being selected:

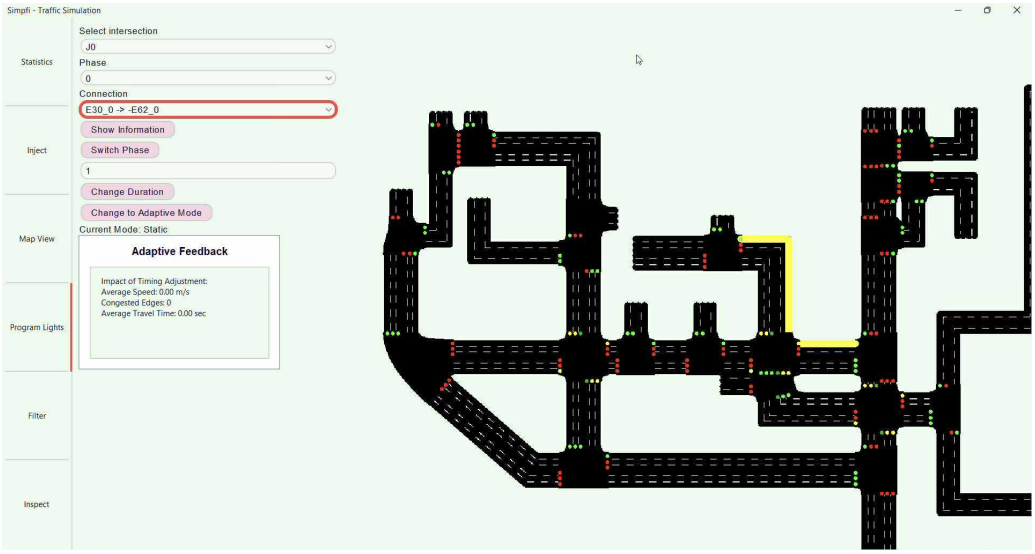


Figure 8: Choosing another connection in the same Intersection

If users want to view all the information including the state of the traffic light, the current phase, the signal of each connection, and the remaining time of the current duration, Users can click on “Show information”.

After that, a pop-up dialog will appear on the screen and update by real time in traffic light. And the signal of traffic light will represent the status light of that connection. Here “r” is red, “y” is yellow, and “g” is green. Sometimes, users will see big “G”, which represents for green light, but higher priority than “g”.

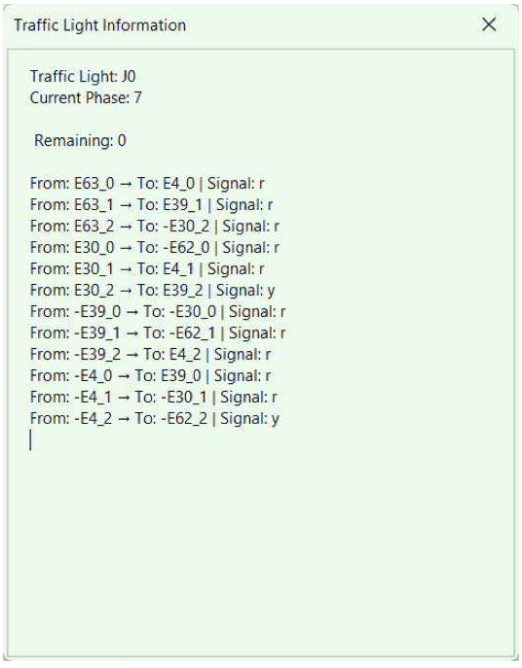


Figure 9: Information DiaLog in Program Lights Panel

If users want to switch the desired phases immediately. First, users need to choose the traffic light ID in “Select intersection” and then choose the phase number in “Phase” and click “Switch Phase” after that to trigger the event. Similarly, if users want to set new duration for specific phases of the traffic light. First, select traffic light ID in “Select intersection”. And then select the phase user want to choose in “Phase”. Then enter the time value which is the new duration for the phase user choose. And finally, click “Change Duration”. Additionally, this panel introduce two different traffic light modes: static and adaptive mode. Users can freely switch between two modes by clicking “Change to Adaptive Mode” or “Change to Static Mode”. Here “Static Mode” is when the logic or program of the traffic light is predefined from the *XML* file. So, there is no change in the definition of the program (e.g. phases, states, signals, duration, etc.). If users tend to choose the mode to “Adaptive" mode. The traffic light may change based on the congestion of the traffic. Here, the team added the logic that if any traffic light has the lane which the number of vehicles of that lane is larger than 2, then the duration of current phase of that traffic light is increased to 5 (In other words, take the default duration and plus 5). This helps users to track the traffic and have a comparison between static mode and adaptive mode.

6.3.5 Filter

This panel provides three main features:

- `vehicle-type filter`: users can see a list of checkboxes showing different types of vehicles. Vehicles whose type is not ticked will not be painted on the map.
- `road filter`: with this filter, users are able to filter out roads, which are the combinations of edges and their sub-edges(not include junction edges). Designed similarly like the `vehicle-type filter`, users check the boxes to opt in roads where vehicles are visible. The only difference is that users can enter the mouse into each checkbox to see the corresponding road. For example, `E45` is entered:

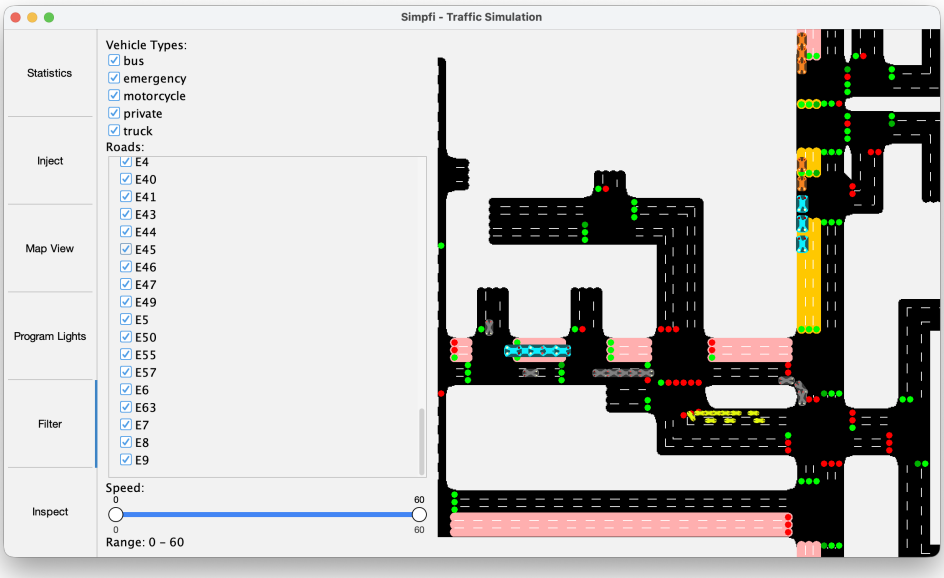


Figure 10: Map when entering the mouse into E45

Then when E45 is unticked, vehicles become invisible in that road:

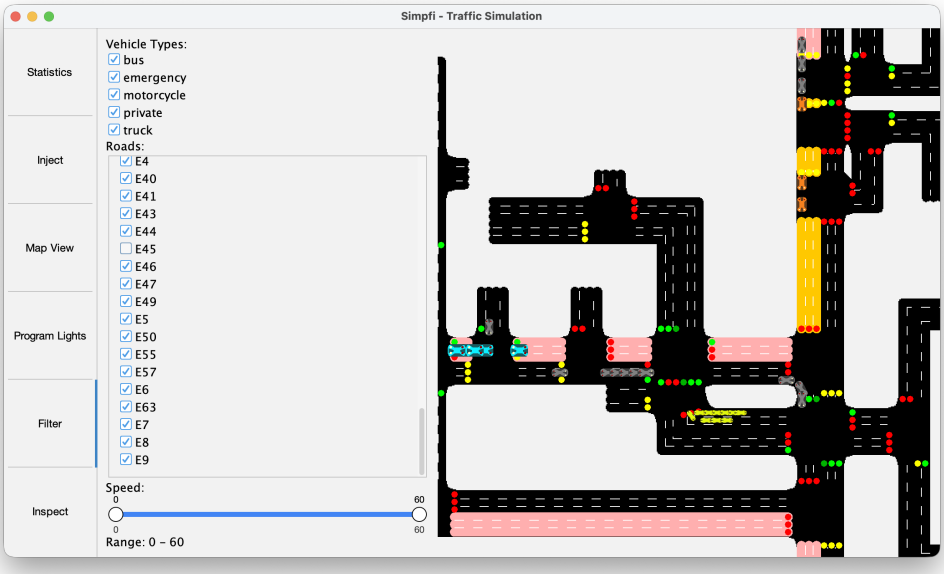


Figure 11: Map when unticking E45

- `speed filter`: Different from the two previous filters, this filter use a range slider to filter out vehicles within a specific range. The defined range is set from 0 to 60(m/s), which is derived from German speed limits. Users can drag two thumbs to set the desired speed range for visible vehicles.

6.3.6 Inspect

With the **Inspect Panel**, users can select vehicles directly from the Simulation by clicking on it.
The user can switch between two modes by pressing the **Change Mode** Button on the bottom: The **Pan mode** is used to drag the map around. After changing to the **Select mode**, the user will be able to select a vehicle on the map by clicking on it.
The list on the top left of the panel shows the currently selected Vehicles. To the right there is a **Select All button** to select all vehicles, which are currently in the Simulation. Use the **Clear button** below to clear the List.
If the user chooses a selected vehicle from the upper box, this element will be highlighted and the corresponding stats will show in the lower Box.
The list can also be sorted by using the Group By dropdown. The user can group the selected vehicles by **Vehicle Type**, **Color**, **Speed** or **Route**. To see the list without grouping, select **None**.
The stat **Vehicle Type** shows one of the following vehicles: Bus (blue), Emergency (orange), Motorcycle (pink), Private (yellow), Truck (grey).
The stat **Color** shows the color in an RGB format, representing the Color components Red, Green, Blue from 0 to 255. Example: R:255, G:0, B:0 –> maximum (255) amount of Red, 0 Green and Blue
The **Speed** is the only dynamic statistic in the list. It gets updated every 500ms. The speed is given in km/h.
Max Speed shows the maximum speed the vehicle reaches in the simulation.
Acceleration is given in m/s^2.
The **Distance Traveled** stat shows the covered distance in this simulation and is given in meters.
The **Route** stat shows the connected edges forming the vehicles route.

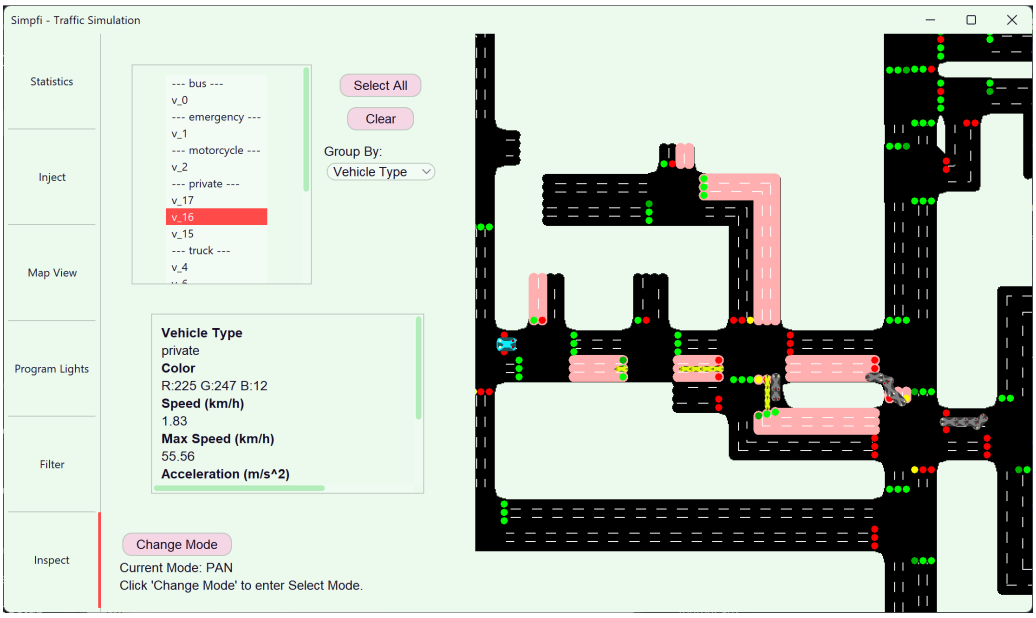


Figure 12: Inspect Panel

As you can see in the picture, the list is currently grouped by Vehicle Type. Each batch of vehicles with the same type are headed with their types name.

6.4 Technical Description

Below are the six panels with a more technical explanation:

6.4.1 Technical Description – Statistics Panel

In the Statistics feature, the average speed, vehicle density per edge, and travel time distribution are visualized by 3 live charts: line chart, bar chart, and histogram. First of all, the average speed is computed by summing all vehicles' velocities and dividing by the number of vehicles running on the network via the getAverageSpeed() method. The second data is the traffic density, which simulates the density of vehicles on each edge using a bar chart and results in congested hotspots – 3 edges with the largest number of vehicles. Particularly, the number of vehicles is collected on each edge per step by traversing all edge IDs and storing them in a hash map, which is the edgeVehicleCount. Then, this data is put into the vehicleDataset for plotting a live bar chart. Finally, the travel time data is computed by subtraction of the current step from the start step. Then, it is stored in a list travelTimes for the travel time distribution visualization.

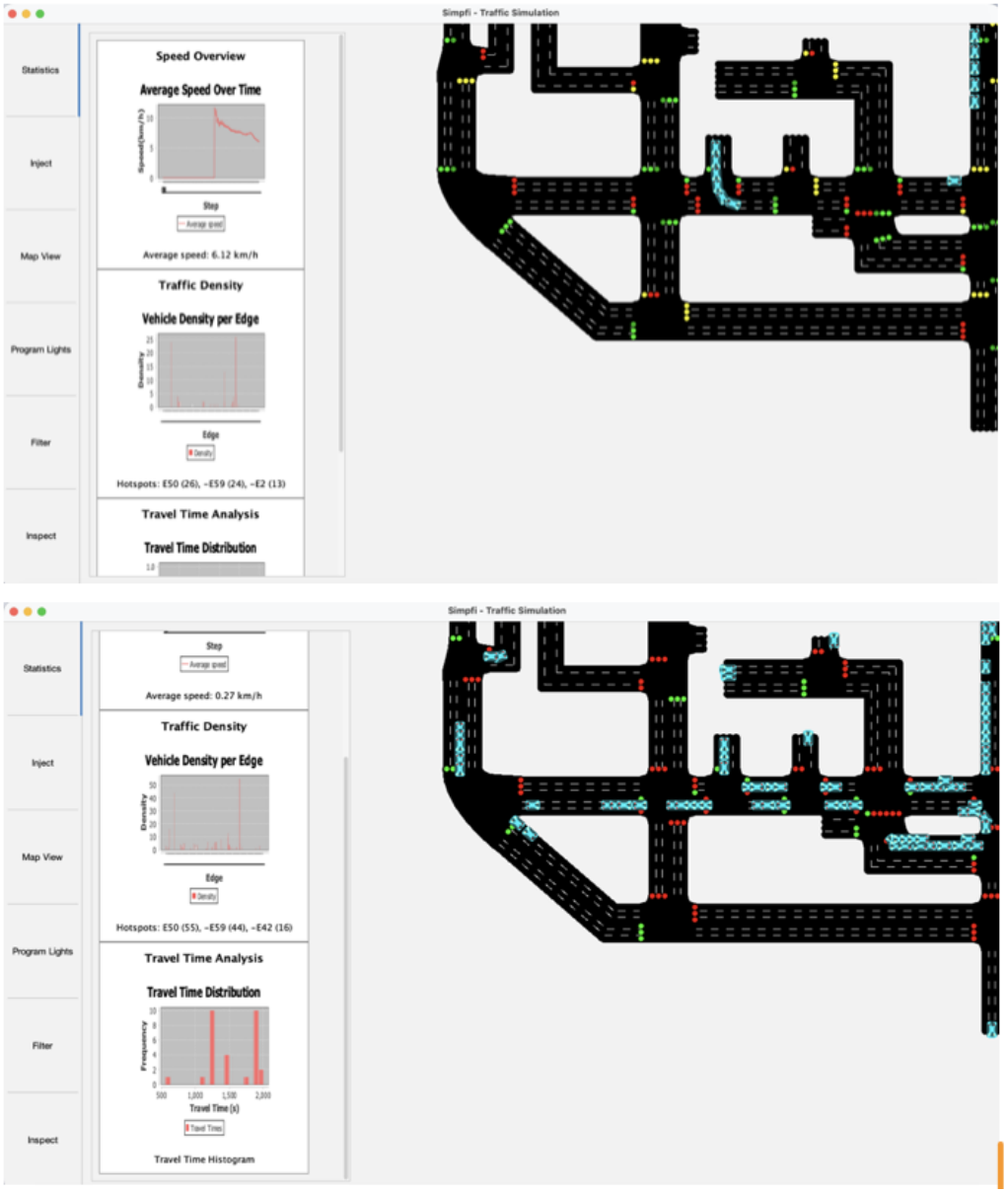


Figure 13: Statistics Panel

6.4.2 Technical Description – Inject Panel

The InjectPanel class is a Java Swing-based user interface for injecting vehicles. It inherits the Panel class and allows for selection of vehicle types, routes, injection mode, and the number of vehicles to inject. The InjectPanel allows users to dynamically inject vehicles into a network during simulation time. It supports three modes of injection:

- **Batch On Route** – inject multiple vehicles across a specific route
- **Batch Random** – inject multiple vehicles across a network
- **Single** – inject a selected vehicle across a chosen route

The image below shows 5 key components of the Inject Panel feature:

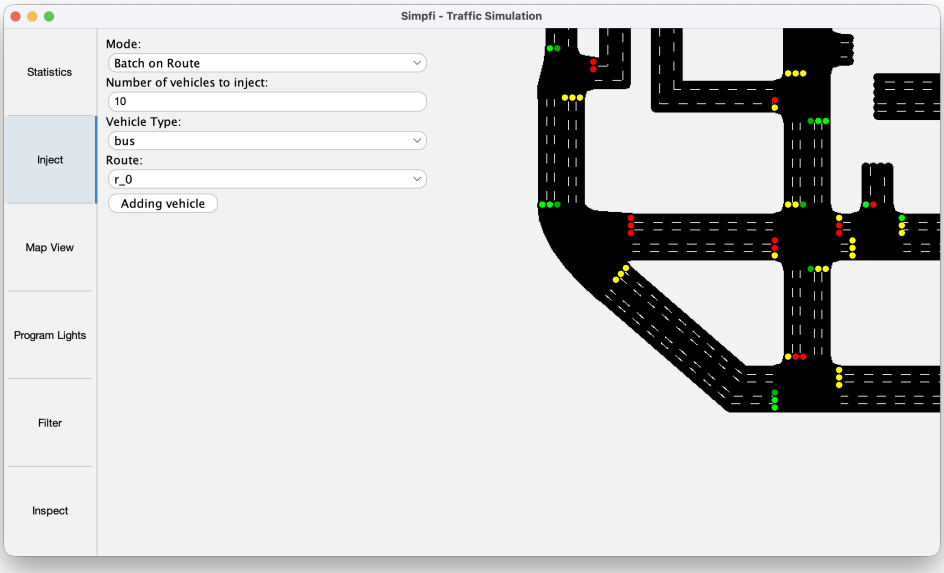


Figure 14: Inject Panel

1. **Mode Dropdown (modeDropdown)**: Provides 3 options – Batch On Route, Batch Random, and Single. It determines whether users want to inject multiple vehicles into a selected or a random route, or even a single vehicle into a specific route when clicking on the "Add vehicle" button.

2. **Count Field (countField)**: Provides a text area allowing users to type any number of vehicles they want to add to a network in batch modes. In a single mode, it is hidden dynamically.

3. **Vehicle Type Dropdown (vehicleTypeDropdown)**: allows users to select a vehicle type from a list of vehicle types obtained from Settings.network.getVehicleTypes(), which is wrapped by the getAllVehiclesTypesAsString() method.

4. **Route Dropdown (routeDropdown)**: allows users to select a route for vehicle injection from a list of route IDs achieved from Settings.network.getRoutes(), which is wrapped by the getAllRouteIdsAsStrings() method.

5. **Add Vehicle button (addVehicleBtn)**: Executes the vehicle injection process via mouse click in a separate Thread to prevent blocking the EDT from Java Swing.

The addVehicle() method handles vehicle injection and reads users' input for vehicle mode, vehicle count, vehicle type, and route. Then the method executes a thread for injection and

divides three modes into three cases using three different methods obtained from the VehicleInjectionController class.

6.4.3 Technical Description – Map View Panel

In the map view panel, many attributes of the map can be changed freely by the user. This is possible by updating an internal variable that can be accessed by all classes and functions so that the map drawing can be consistent. In order to implement the "Reset defaults" button, another list of constants are stored with the default values, and when the button is pressed, all properties of the map are set to the original default value.

6.4.4 Technical Description – Program Lights Panel

The Program Light Panel provides access to all traffic light information, including the junction ID, all lane connections, and the traffic light programs defined in SUMO. These attributes are retrieved through SUMO's junction calls. Since the drawing interacts with the SUMO API (TrasS), traffic light states can be changed dynamically by calling the SUMO traffic light interface. User interface elements such as drop-down menus, buttons, text areas, and dialogs are implemented by extending Java Swing classes. The remaining duration of the current phase of specific traffic light is computed by retrieving the next switch time from SUMO and subtracting it by the current time. The signal of each traffic light connection is parsed from the traffic light state String by splitting it into individual characters, where each character represents the signal assigned to a specific connection. Moreover, to ensure consistency and accuracy, both the remaining duration and the signal of the traffic light are updated in real-time and displayed in "Show Information". Finally, in order to implement the logic of "Adaptive mode", the map is continuously updated in real time. In this mode, the update logic adjusts the traffic light phase duration based on the current traffic flow conditions, enabling the system to adapt automatically and maintain flexible traffic signal control.

6.4.5 Technical Description – Filter Panel

For the implementation of the first two filters(vehicle-tpye and road), the team defined a `filterFlag` in `VehicleType.java` and `Road.java`. After creating the dropdowns, which extend `JComboBox`, listeners were added to those components to update the `filterFlag` when there are updates. As a result, the map will only draw vehicles whose `filterFlag` is set to `true`.

In addition to two filters above, speed filter is implemented in a more simple approach, since each vehicle already has its own `getSpeed()` method and the code just needs to check if its speed falls within the limited range before drawing. However, the difficult part is the Range Slider,

which is not supported in Java Swing. Therefore, the team developed a new `RangeSlider` object that inherits from `JComponent` to display it and let users freely choose the desired speed range. Furthermore, the maximum speed was set to 60 m/s manually because the team could not find any specific documentation about vehicles' maximum velocity in SUMO, so this figure was decided based on our research about speed limits in Germany.

6.4.6 Technical Description – Inspect Panel

The Inspect Panel allows users to inspect the vehicles in the simulation. Users can toggle between two modes:

- **Pan Mode**: Enables map navigation by dragging.
- **Select Mode**: Allows selecting a vehicle on the map with a click.

The **Select Mode** feature is realized by a Mouse Listener to get the coordinates of the selected part of the map. These coordinates are then translated to world coordinates to be utilized to find the nearest vehicle in a range of 20 units.

The top-left list displays the currently selected vehicles. Users can use the **Select All** button to select all vehicles in the simulation or the **Clear** button to clear the list. Selecting a vehicle from the list marks it and shows its statistics in the lower box. The list can be sorted using the Group By dropdown. Users can group the selected vehicles by Vehicle Type, Color, Speed, or Route. To display the list without grouping, select None. The speed headers, which appear when grouping by route are not updated live, because this would be too demanding.

Displayed vehicle statistics include:

- **Vehicle Type**: Bus, Emergency, Motorcycle, Private, Truck.
- **Color**: RGB format (0-255) representing Red, Green, and Blue components.
- **Speed**: Updated every 500 ms, in km/h.
- **Max Speed**: Maximum speed reached during the simulation.
- **Acceleration**: In m/s^2.
- **Distance Traveled**: Covered distance in meters.
- **Route**: Sequence of connected edges forming the vehicle's route.

7 Conclusion

From an objective perspective, the team is progressing at a high rate and therefore plausibly finish the project before the target date.