Group 3 - Traffic Simulator Project Documentation

Vu Minh Nguyen, Duc Tung Nguyen, Bach Tue Dinh, Nam Khanh Thieu

December 13, 2024

1 Introduction

Among all the proposed projects for the final project in the course **ELEC-A7151 Object Oriented Programming with C++**, our group has chosen the Traffic Simulator. Simulating and analyzing urban traffic can be useful for predicting traffic congestion and assessing the impact of new highways. Several factors, including traffic volume, road layout, population density, and the density of factories and companies, contribute to traffic's highly nonlinear and chaotic nature. Therefore, it is useful for authorities, road planners, and urban planners to have this tool that allows them to observe and analyze the traffic situation, enabling them to optimize traffic flow by improving road layouts, redistributing industrial buildings, and developing strategies to manage peak traffic hours more effectively.

2 Project Overview and Features

2.1 Importing and Exporting City from JSON

The city data is loaded from a JSON file, which is updated at the end of the program. The file, located in `src/data`, contains information about roads, people, intersections, and buildings.

2.2 Buildings with Different Profiles

There are five distinct types of buildings: residential, factory, office, two variations of multipurpose malls, and a special building type—bus stations. (Figure 1)

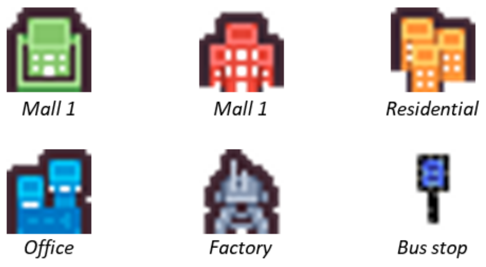

Figure 1: Different Types of Buildings

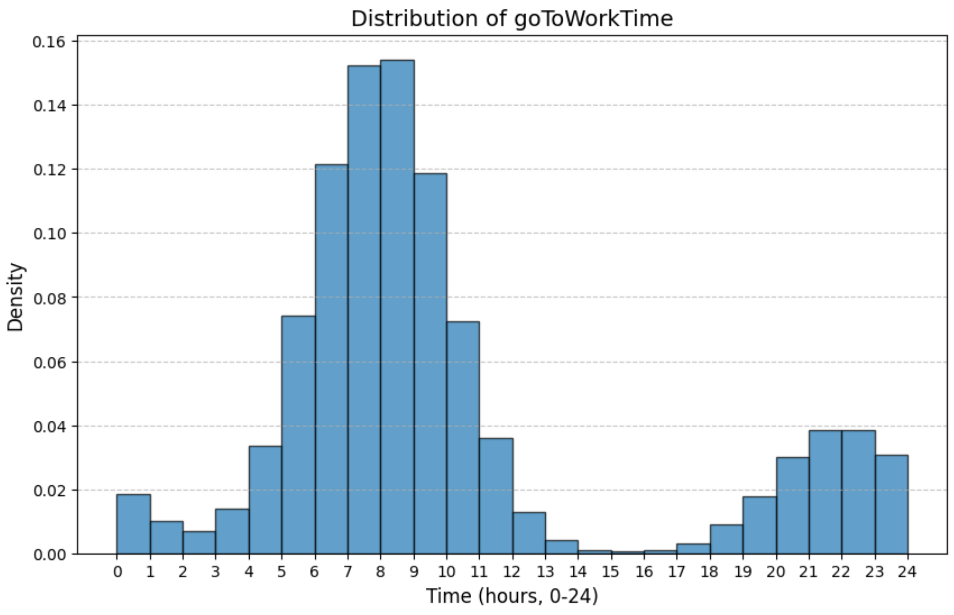

Figure 2: *Distribution of goToWorkTime drawn from 100,000 samples*

2.3 Residents with Randomized Activity Patterns

In this simulation, the behavior of residents is modeled with random schedules, incorporating various probability distributions to introduce natural variability. The following distributions are used to generate daily schedules for citizens:

- **Go to work time:** Residents determine their time to leave for work based on a weighted combination of two normal distributions, one peaks around 8:00 AM (day shift) and the other peaks around 10:00 PM (night shift), both have standard deviation of 2 hours. A weighting factor of 0.2 determines the probability of a person having night shift. (Figure 2)

- **Work Duration:** Once at work, residents spend a random amount of time there, determined by a uniform distribution between 6 and 8 hours.

- **Interval Between Work and Leisure:** After work, the time residents spend before transitioning to leisure activities is also randomized. This duration is sampled from a uniform distribution between 4 and 7 hours.

- **Leisure Duration:** The time spent at leisure locations, such as their favorite buildings, follows another uniform distribution ranging from 4 to 7 hours.

- **Minute Remainder:** To further randomize the departure times, a uniform distribution between 0 and 59 is used, adding minute-level precision to the schedule. For example, if `minuteRemainder` is 3, `goToWorkTime` is 7 and `workDuration` is 8, then `homeFromWorkTime` is 15:03.

2.4 Cars with Path Finding Algorithms

This path finding algorithm is a local, heuristic-based approach that enables cars to navigate efficiently in a grid-like city layout. At each intersection, the car will decide among all connected roads which road to take next, based on a prioritized heuristic that minimizes the distance to the destination. Importantly, the car will not choose the road it just exited. By prioritizing directions based on proximity to the destination and handling transitions between roads and intersections dynamically, the algorithm balances simplicity and realism for urban traffic simulation.

2.5 Public Transport

The city has a bus service that runs continuously throughout the day and night, providing transportation 24 hours a day. The bus follows a circular route, stopping at each station along the way. At every station, it waits for about one minute in simulation time.

2.6 Analysis Tool

The simulation includes an interactive road analysis tool that provides detailed insights into traffic patterns on individual roads. Users can right-click on a road to view its logs a bar chart highlighting the daily average traffic in different time of the day. (Figure 3)

The logs of each road will be exported to a file in CSV format after the user closes the program.

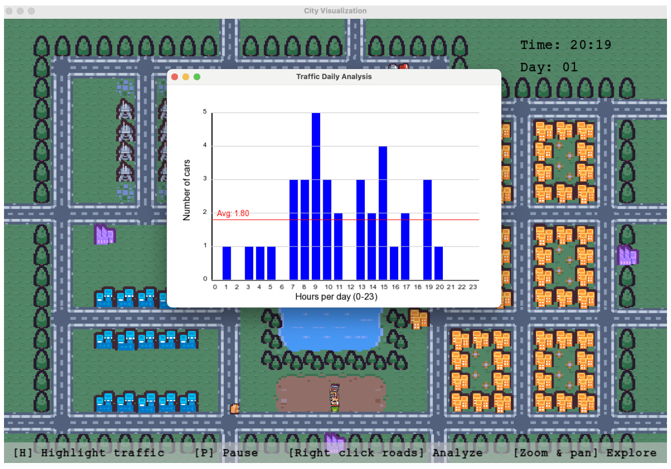

Figure 3: *Traffic Analysis*

2.7 Congestion Highlight

In *Highlight Mode*, which can be activated or deactivated by pressing `H`, roads are color based on their current traffic density, with green indicating free-flowing traffic and red signaling heavy congestion. (Figure 4)

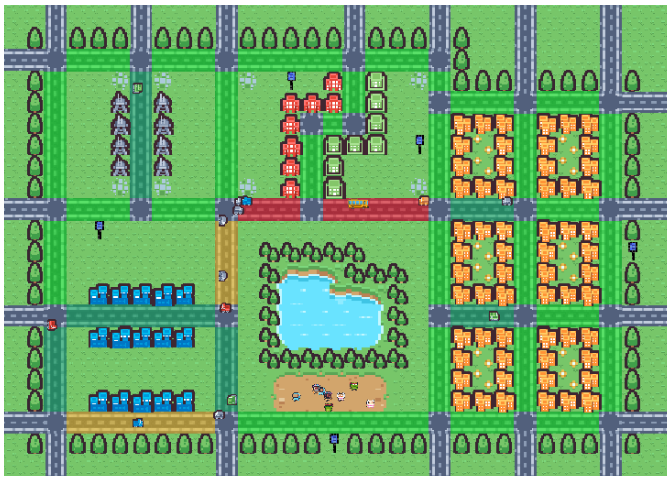

Figure 4: *Congestion Highlight Mode*

3 Software Structure

3.1 Class Structure

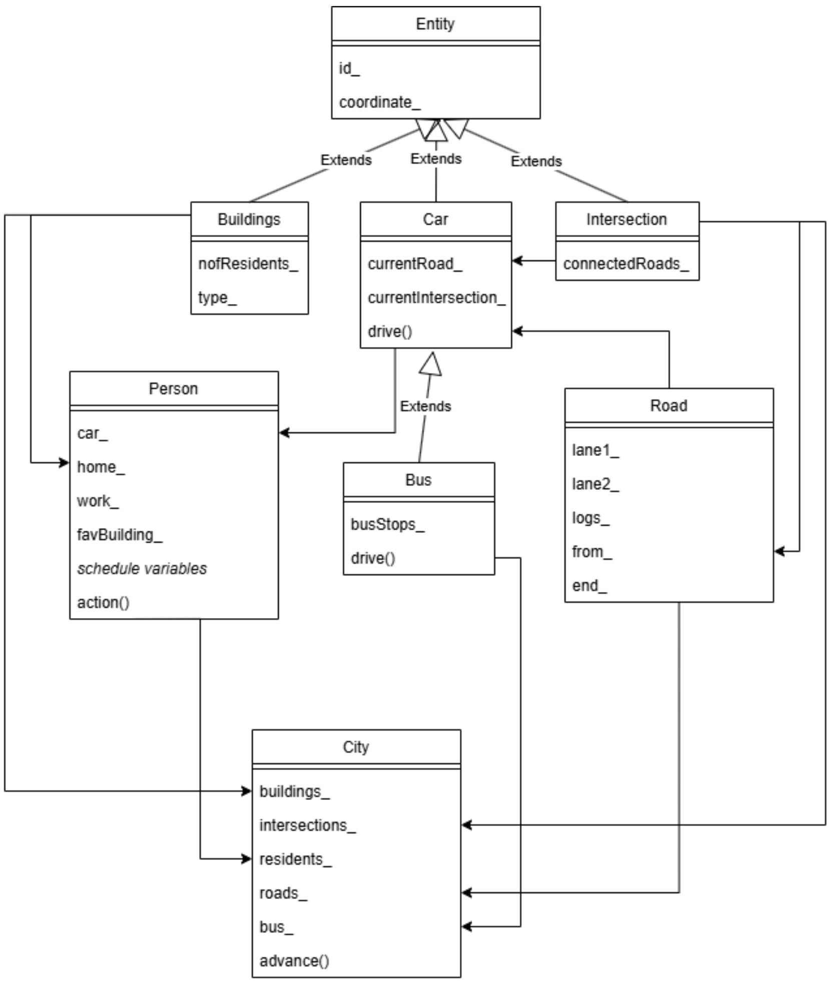

Figure 5: *Software Structure*

At the top is the **Entity** class, which serves as the base class with attributes `id_` and `coordinate_`. Three classes **Buildings**, **Car**, and **Intersection**, inherit from this base class.

The **City** class is the center of the simulation, contains all classes with `buildings_`, `intersections_`, `residents_`, and `roads_`. It manages the simulation's progression through its `advance()` method.

The **Person** class represents residents in the simulation. Their behaviors, such as traveling and scheduling, are governed by the `action()` method.

The **Car** class models vehicles, tracking their current `currentRoad_` and `currentIntersection_`, and facilitates movement through the `drive()` method. Cars interact with the **Road** class, which defines connections between intersections using attributes like `lane1_`, `lane2_`, and `logs_`. Roads also have start and end points (`from_` and `end_`).
Since we are using **Doxygen** for code documentation, it is possible to check out the documentation, it is different for each operating systems in the command to run the **html** file but for example in **macOS**, you can execute the following commands:

```
cd doxygen-docs
cd html
open index.html
```

For other operating system to run this HTML file, you can install **Live Server** extension, get to the **doxygen-docs/html**, right click on **index.html** and browse Code Documentation, you can see the code documentation

3.2 Libraries Usage

File Parsing (City Loading from File): **nlohmann/json** - An easy-to-use header-only JSON library. It allows parsing and serializing JSON objects without much boilerplate. Used for parsing the user's input and saving the created city layout.

Visualizations (GUI): **SFML (Simple and Fast Multimedia Library)** - A good choice for 2D graphics and is well-suited for visualizing buildings, roads, cars, intersections, etc. It can also be used for rendering shapes.

Code Documentation: **Doxygen** - It helps in code documentation, ensuring clear and comprehensive descriptions of the functionality and structure of the simulation codebase.

Testing: **Google Test (gtest)** - It helps to write unit tests to validate simulation's functionality, ensuring that components like file parsing, car movements, traffic analysis, and more are working as expected.

4 Instructions for building and using the software

4.1 Install Libraries

- **Install CMake:**
  - Since we utilized CMake as our build system generator, it is essential to have CMake installed to run the project. If CMake is not installed, please refer to the documentation provided below.
  - Refer to the CMake documentation for installation instructions for Windows, macOS, and Linux.
- **Install SFML:**
  - Refer to SFML official website and follow the download instructions for Windows, macOS, and Linux.
  - For example, on Linux, you can execute the following command:

    ```
    sudo apt update
    sudo apt-get install libsfml-dev
    ```
- **Install GoogleTest:**
  - Refer to the GoogleTest documentation for installation steps. for Windows, macOS, and Linux.
  - For example, on Linux, you can execute the following command:

    ```
    sudo apt update
    sudo apt install libgtest-dev
    ```

4.2 Compiling

To build and use the software, start by navigating to the root directory of the project. From there, run the following commands to generate the necessary build files in `build` directory:

```
mkdir build
cd build
cmake ..
```

Next run the following command to compile the program:

```
make
```

Once the build process is complete, launch the application by running:

```
./TrafficSimulator
```

Alternatively, we created a Bash script file to run this faster, run the following commands to grant execute permission and run the software:

```
./run-app.sh
```

4.3 User Guide

When the program is running, you can interact with the interface using several intuitive controls:

- Use pinch or scroll gestures to zoom in and out of the map, allowing you to view the simulation from different levels of detail.
- To explore different areas of the map, click and drag to span across the city.
- If you wish to temporarily halt the simulation, press the `P` key to pause.
- Additionally, you can press the `H` key to toggle *Highlight Mode.*
- Last but not least, you can right click a road to analyze its traffic logs.

These controls provide a comprehensive way to interact with and analyze the simulated city environment.

5 Testing and instructions for testing the software

The project includes automated unit tests built with Google Test (gtest) for the Road, Intersection, Building, Car, and Person classes, covering nearly all functions within them, more specific details you can check out the `readme.md` in `tests` directory of our project. Functions in the main file, such as draw methods, analysis tools, and congestion tools, are tested manually using sample data and then real data from the software.

To run the unit tests, start by navigating to the root directory of the project. From there, run the following commands to generate the necessary build files in `build` directory:

```
mkdir build
cd build
cmake ..
```

Next run the following command to compile the program:

```
make
```

Once the build process is complete, run the unit tests by running:

```
./TrafficSimulatorTests
```

Alternatively, we created a Bash script file to run this faster, run the following commands to grant execute permission and run the unit tests:

```
./run-test.sh
```

6 Working Log

Below is the division of work between the members of our group.

- Duc Tung Nguyen:

Date	Description of Work	Hours Spent
Week 0	Meeting to plan the project. Complete the project plan	3.5
Week 1	Implementing simple logic for class Person	4
Week 2	Resolving logic for class Road. Update meeting notes	4
Week 3	Complete 80% of the logic for classes in the project	8
Week 4	Complete the basic logic and integration for classes in the project	8
Week 5	Fix the logic for driving and create a new map for project, completed building and person profiles	8
Week 6	Complete the logic for basic and additional requirements	8
Week 7	Complete the analysis tool, unit tests, Doxygen documentation, testing documentation, and documentation	19

- Vu Minh Nguyen:

Date	Description of Work	Hours Spent
Week 0	Meeting to plan the project. Complete the project plan	3.5
Week 1	Implementing simple logic for class City, Building	2.5
Week 2	Resolving logic for class Car	3
Week 3	Complete 80% of the logic for classes in the project	9.5
Week 4	Complete the basic logic and integration for classes in the project	8
Week 5	Fix the logic for driving and create a new map for project, completed building and person profiles	9
Week 6	Complete the logic for basic and additional requirements	7
Week 7	Complete the analysis tool, congestion tool, documentation, and refactor the code	18

- Nam Khanh Thieu:

Date	Description of Work	Hours Spent
Week 0	Meeting to plan the project. Complete the project plan	3.5
Week 1	Implementing simple logic for class Intersection, Road	4
Week 2	Resolving logic for class Intersection	4
Week 3	Research JSON library, create a demo to load data from files	9
Week 4	Complete the basic logic and integration for classes in the project	8
Week 5	Working on the city loading from file functionality	8
Week 6	Complete the import city from file, export logs to CSV, and debugging	12
Week 7	Complete the documentation and QA testing the code	12

- Bach Tue Dinh:

Date	Description of Work	Hours Spent
Week 0	Meeting to plan the project. Complete the project plan	3.5
Week 1	Implementing simple logic for class Car	4
Week 2	Resolving logic for class Car	4
Week 3	Create a demo GUI with FSML, create build file for project	8
Week 4	Create a new GUI for the project	8
Week 5	Working on a new GUI for the project	8
Week 6	Complete the GUI for the project	8
Week 7	Complete the GUI with enhanced visualization	17