Group 3 - Traffic Simulator
Project Plan

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

November 1, 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 Features and Functionalities

The final outcome of the project is a traffic simulation featuring a grid-based map with different types of tiles, such as roads, land, and buildings. The simulation will include people living in residential buildings who use their cars to drive on roads, either to reach industrial buildings where they work or commercial buildings where they shop for goods or services. Each individual will have a unique schedule with random variations. The system will also incorporate traffic control mechanisms like lights and signs, which cars will strictly obey. Each road will have a limit on the number of cars allowed. Additionally, there will be a tool to plot a histogram showing the number of cars on a specific road at different times of the day. The plot can be exported as a csv file.

The city will include customizable statistics that users can set, such as area, population, number of roads, and buildings. These settings can be exported as a JSON file, and cities can also be created by importing a JSON file. Furthermore, the project will feature a GUI that enables users to zoom out for a macro view or zoom in for a micro view of the city. The GUI will also allow users to customize various aspects of their city.

3 High-level structure of the software

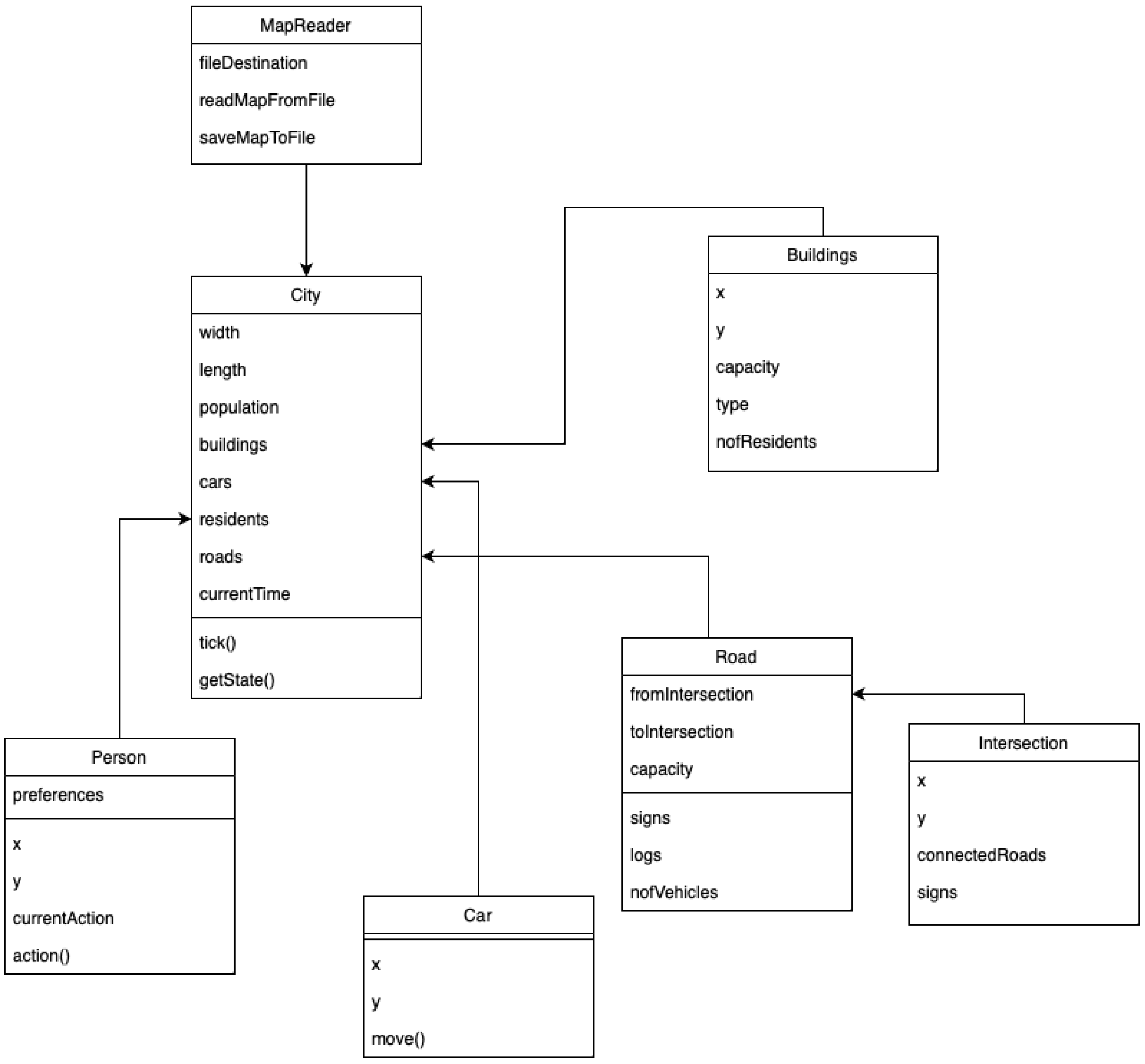

Figure 1: Software Structure

The general relationships between the program's classes is demonstrated in the figure above. An object of class MapReader will be responsible for reading maps written in dedicated format, JSON in this case, and return a new instance of City.

City: Act as the main object/Map in the program. It has width and length for parameters, which helps identify the limits in which other objects can be placed. The city also contains multiple lists of roads, buildings, residents, cars, which are objects in the city participating in the simulation activity. The startTime and currentTime variable will store the start time and current time of the simulations, which helps determine the simulation state as well as calculating the next state of the city.

The method tick() increases the current time of the city and also causes objects such as person, car, building in the city to update to the next state e.g. Cars moving forward, people doing actions. method getState() returns the current state of the city.

Intersection, Car, Person, Building classes share common x and y attributes, representing the current position of the object. It is likely that the position of Building and Intersection will stay the same while the x,y coordinates of Cars and People will change occasionally.

Intersection: Positions where two or more roads connect. It has the connectedRoads array to keep track of the roads meeting at the intersection. The intersection also has an array of traffic signs.

Road: Acts as the path for cars and people to commute. It has multiple arrays containing the intersections and buildings where it is connected to.

Each road has its own capacity, representing the maximum number of vehicles that can travel on it. nofVehicles store the current number of vehicles traveling on the road, which helps to calculate the same quantity in the next state of the city. The road also stores its past statistics in order to perform analysis.

Building: Acts as the start place/ end destinations of vehicles and residents. The building has a type attribute showing the purpose of the building, such workplace, residential, or commercial buildings. Capacity and nofVehicles denotes the maximum and current number of residents in the building, respectively.

Person: Aside from the current location, each person has a preference , which is the list of places that they travel occasionally at certain times. The current action of a person is recorded in currentAction, which helps to determine the next action/state of the person. Action() method changes the current state/action of a person according to the last one.

Car: a method of communication of people. The move() method is similar to the action() of a person, which updates the position of a car every time it is called.

4 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.

Data Analysis: **Boost.Histogram, Boost.Accumulators** - The first one is specialized in histogram data collection and manipulation. The second one is useful for calculating statistics like averages and histograms, especially to analyze traffic trends over time.

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.

Simulation: **Eigen** - An efficient linear algebra library, particularly for matrix operations, useful in any spatial or physics-related calculations. **Random** provides powerful tools for generating random distributions for simulating slight variations in car speeds, departure times, etc.

5 Sprint Planning and Management

5.1 Sprint Overview and Timeline

- **Sprint 1: Core Framework and Basic Features (1/11 - 7/11)**

  High-level Goals: Set up the project structure and repository; Implement city loading from a file and basic parsing for buildings and roads; Create initial classes and data structures for buildings, roads, intersections, and cars; Basic functionality to load city data and initialize basic objects in memory.

- **Sprint 2: Traffic Simulation Mechanics (8/11 - 14/11)**

  High-level Goals: Implement basic traffic flow mechanics, for example car movement; Introduce randomness to simulate real-world variability; Develop passenger behavior to simulate trips between residential, commercial, and industrial buildings.

- **Sprint 3: Data Analysis and Basic GUI (15/11 - 25/11)**

  High-level Goals: Develop analysis tools, such as a histogram; Export analysis data to CSV for further examination; Implement a simple GUI for visualizing; Building and improving tests for analysis and GUI functionality.

- **Sprint 4: Advanced Features and Final Testing (25/11 - 12/12)**

  High-level Goals: Add optional advanced features; Improve and fine-tune the GUI with additional visualization and basic user interactions; Conduct thorough testing; Prepare final documentation and ensure that all project goals are met.

5.2 Sprint Meetings and Progress Tracking

To maintain steady progress and ensure the team is aligned, we will hold **weekly sprint meetings**. Each meeting will last about **30 minutes** and will cover updates, challenges, and next steps. Meeting Schedule: Every **Monday at 8 PM**. At the end of each sprint, we will hold a sprint review and retrospective session. This session will focus on evaluating completed work, discussing improvements, and planning adjustments for the next sprint.

6 Tasks Division

Name	**Responsible Parts**
Bach Tue Dinh	Analysis tools, Basic Main Classes
Duc Tung Nguyen	GUI, Testing
Vu Minh Nguyen	IO modules, Object Features
Nam Khanh Thieu	GUI, Interaction Between Classes

Table 1: Project Responsibilities by Team Member