Project Plan

Traffic Simulator, Group 1

Nikita Tarutin, Emma Rauhala, Alexey Serous, Lauri Sivuoja, Nikolai Semin
Aalto University, 2021

Table of Contents

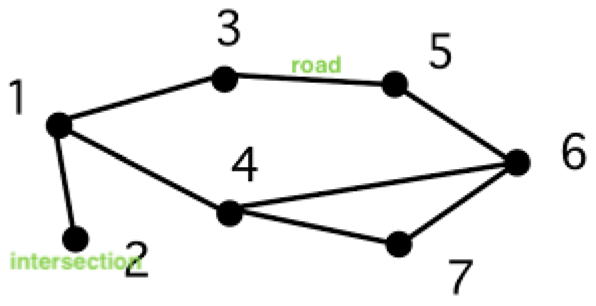
1. Scope of the work

The team will implement traffic simulator software that is capable of simulating traffic flow in relatively large cities taking into account citizens' demands and wants: going to work and earning money, becoming hungry, buying food and spending free time.

Input information about a city will be passed in the form of a .JSON file. Roads for analysis can be specified and the data will be represented in .csv format or as a histogram.

The software will heavily rely on object-orientated paradigm of programming, therefore, implementing classes for such objects as people, living in a city, vehicles they use and buildings they use. Possible implementation is displayed in Figure 1.

Since the software is a traffic simulator, it seems clever to treat the whole city as a graph network, where roads are edges of a graph and intersections are its nodes. Moreover, vehicles need to move around the city from one point to another, and pathfinding is a backbone of vehicles travelling: for that purpose the team is looking to implement either Dijsktra or A* algorithms.



Additionally, there is an intention to use a simulation class (e.g. City class) that will store parsed information; this decision will also allow not to read input file multiple times if several simulations are needed for the city.

The other problem is yet to be tackled is simulation of time. Clearly, the task is to create software's own representation of time, hence, while loop might be used. Each increment will represent a change of a time unit, for example, one minute. A user may define the desired time of simulation, otherwise it will be set to 24 hours by default. Based on the time within the simulation , certain actions will happen: cars will travel distance based on their speed, people will leave buildings when they need to.

Because each person leaves home or work at random time, constructor of the Person class will take responsibility for assigning random leaving and coming times. Besides, realisation of other needs and factors that influence behavior of people in the city will bring randomness. Hunger, tiredness and money are among these factors.

Taking into account additional and advanced features that the team would like to add on the later stages of the project, current program design is intended to be scalable: with extensive use of abstract classes and desire to make the software modular. Modularity will be implemented as three main modules of the program: parser of an input file, main body, where simulation is happening and output file is formed, and finally GUI. Since the user interface is not a critical part of the software, it is desired for the whole program to be fully functional without it and, therefore, GUI should be an additional module that will be connected to the main modules through developed interfaces.

It is important to remember that software engineering is not only about coding, but it is also about processes used to manage development of the product. The team has decided to follow an agile-like approach with fast communications through a chat, hold weekly meetings and be always ready to assist each other. Additionally, following Google code style, DRY principle and committing often should help keeping codebase clean, avoid merge conflicts and maintain code readable for people outside the team.

Key points of the work are following:

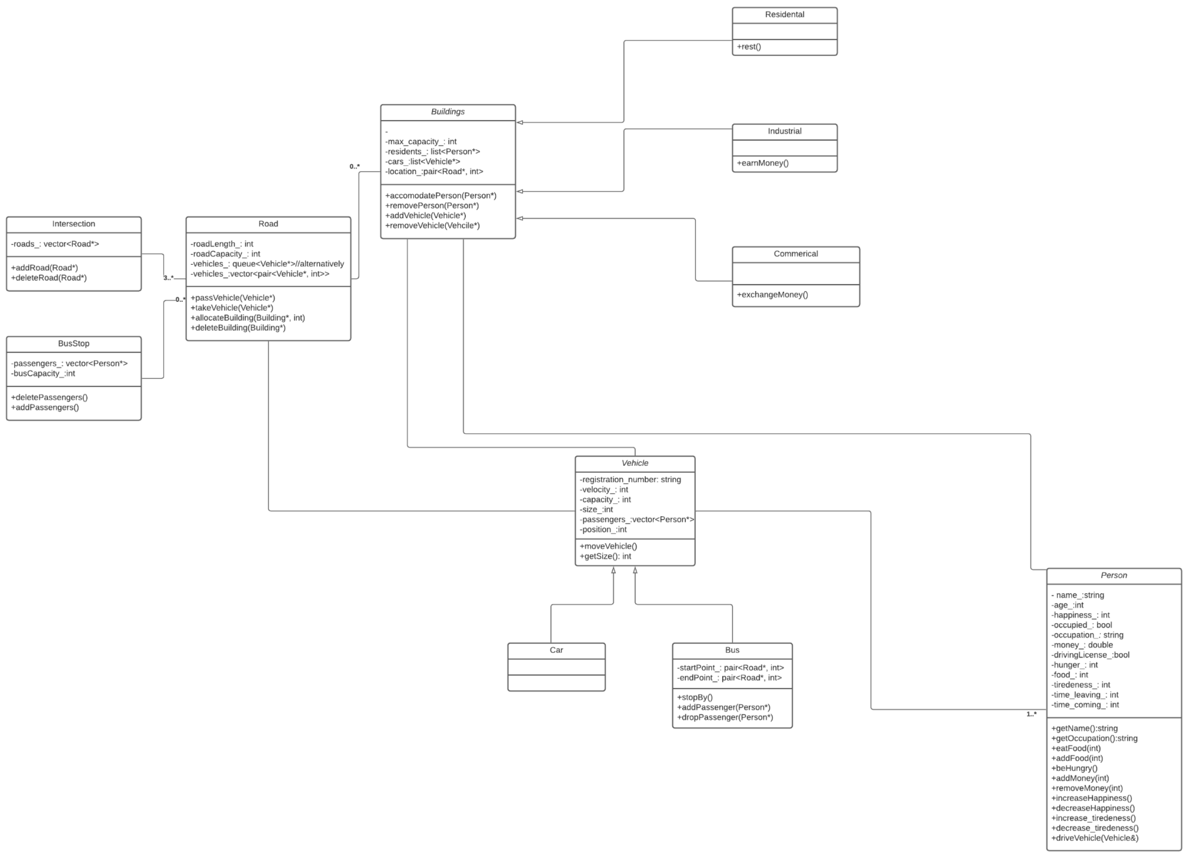
- Following discussed practices
- Implement and test base classes
- Develop pathfinding
- Implement .JSON file parser
- Implement additional features
- Add GUI

2. The high-level structure of the software

As described above, a simulated city, its structure and objects inside the city will be encapsulated into the simulation class. The class will potentially have interfaces to be connected with GUI and will be responsible for initialising parsed objects.

A class that represents a city's inhabitant is a Person class. Persons have names, time when they should leave and come home and potentially a car that they will ride. Hunger, tiredness and money are attributes of the class that influence behaviour of a person. One cannot go to work when he is tired (waits at home till tiredness is full) or hungry. These parameters' values are changed during time flow: by default tiredness will minimally decrease while a person is outside his home and then increase when one is resting at home. Being at work will have an even more negative effect on one’s tiredness. Similarly, money can be exchanged for food and hunger will be fulfilled.

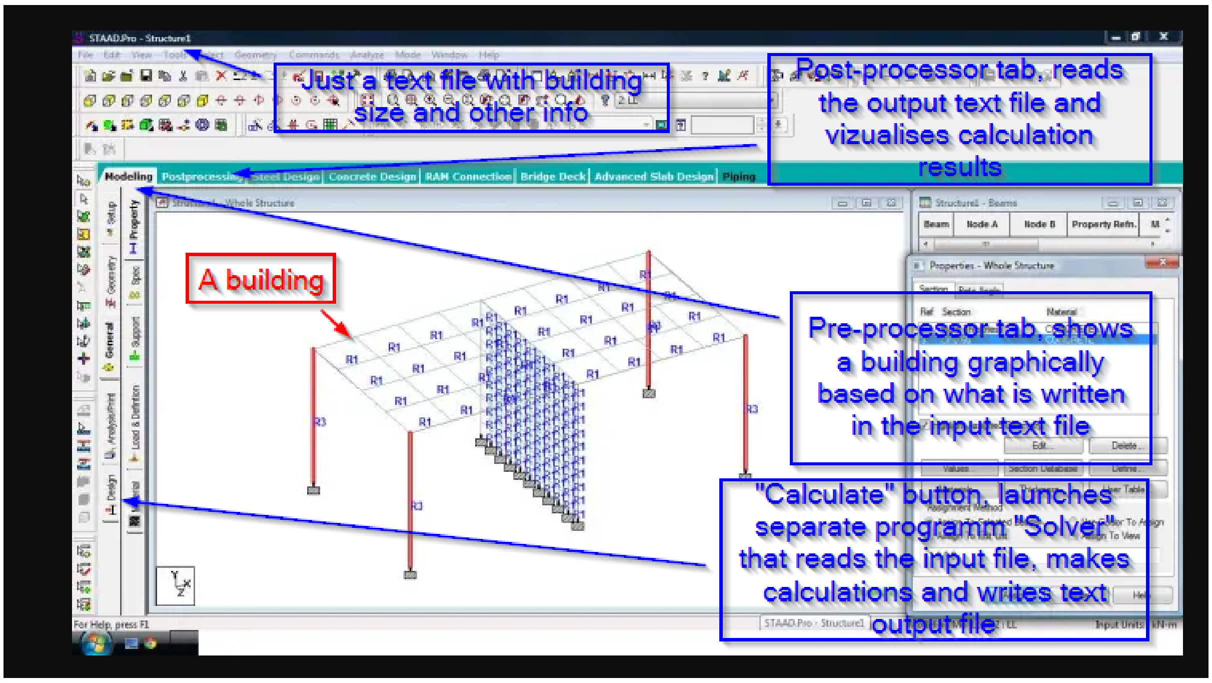
Each person should own a car to travel from home to work and other buildings and back; it is not yet decided how car ownership should be defined, but it is an easy thing to tackle and, therefore, will be solved when implementing the Person class starts. Cars travel on roads using their speed to define what length they should move per unit of simulated time. Because each road stores a pointer to the intersection(s) it belongs to, it would make sense to treat the city's road system like a graph. To define the position of a car, a road will be imagined as an axis. Therefore, each car has its own place on the road that is occupied only by it. Alternatively queue might be used, but its use is not that clear for now.



In the UML diagram below, some drafting is already made for scalability and prospective implementation of advanced and additional features.

3. Software Modular Approach Description

The software will use the pre-processor -> solver -> post-processor concept. In practise this means that a simple input data file exists (for example, plain text or xml) with all the necessary information for simulation. This is to be filled by a user. This file is possible to be viewed and modified graphically by an independent program “Pre-processor”, where all the text is represented visually as real objects. After the input information is ready, an independent program (normally without any user interface at all) “Solver” starts and performs the simulation. The simulation results are written to an output file (normally, same format as the input one). Finally, the third program “Post processor” reads the output file and visualizes the simulation results in graphics. Very often, pre- and post-processor are the same program for convenience. Please see an example of Bentley Staad, a software used to model and calculate buildings:


STAAD.Pro - Structure1
Just a text file with building size and other info
Post-processor tab, reads the output text file and vizualises calculation results
Modeling
Postprocessing
Steel Design
Concrete Design
RAM Connection
Bridge Deck
Advanced Slab Design
Piping
A building
Pre-processor tab, shows a building graphically based on what is written in the input text file
"Calculate" button, launches separate programm "Solver" that reads the input file, makes calculations and writes text output file

The following approach allows for a modular structure. A solver can be written and tested independently, without the necessity to develop a GUI meanwhile.

4. Planned third party tools and libraries

- Build system: Cmake
- Graphical interface libraries: Qt5
- Compiler: gcc
- Google Test

5. Division of work and responsibilities

Task	**Subtasks**	**Description**	**Person(s) in charge**
Planning	Project plan	Defining the scope of the work and designing the high-level structure of the software	Everyone
	UML	Defining base-classes, and their functionalities	Nikolai
Class-implementations	City/Simulation	Writing the respective class-files	Alexey
	Building		Emma
	Vehicle		Nikita
	Person		Nikolai
	Road		Alexey
	Intersection		Lauri
Pre- and postprocessing	Pre	Planning the structure of, implementing and parsing input files	Alexey
	Post	Designing the output format and implementing data extraction	Emma
Build-tools		Setting up Cmake	Nikita
Git-managing			Nikita
Unit-tests			Nikolai
Documentation	Source code documentation	Writing well-commented and self-explanatory code	Everyone!
	(Final) Project Document	Responsibility for the overall contents and structure of the final report	Emma
Meeting secretary		Taking and uploading weekly meeting notes	Lauri
(GUI-implementation)	*To be subtasked and determined*		

6. Planned schedule and milestones

Milestone	**Details**	**Deadline**
Project plan ready		Fri 05.11.21 23:59
Main-function for testing		Wed 10.11.21
Class-sketches	Header-files and some compilable implementations ready	Wed 10.11.21
Base classes ready		Fri 12.11.21
Simulation function implementation		To be confirmed
Path-finding function implementation		To be confirmed
Hard-coded simulation	Sample city hardcoded, no input file, output to cout	After the classes and functions are completed
Input/output from/to file		After the classes and functions are completed
GUI	Tentative	In case the base program is ready
Clean repo		Fri 10.12.21
Project document	Keep updating on the way	Fri 10.12.21
Final commit to git		Sun 12.12.21 23:59