TRAFFIC SIMULATOR PROJECT PLAN

Group name: traffic_simulator_henrik_toikka_4

Aape Hartikainen, aape.hartikainen@aalto.fi
Jiri Simell, jiri.simell@aalto.fi
Jaakko Karhu, jaakko.karhu@aalto.fi
Artur Brander, artur.brander@aalto.fi

SCOPE OF THE WORK

The main purpose of this project is to create a configurable traffic simulator that is used to simulate and analyze the traffic patterns in a city. This could be useful for predicting traffic jams, estimating the effect of new roads or analyzing the patterns of certain citizens. The simulator should be able to analyze and process large cities.

The simulator is developed with C++ and it is compiled and used with a compatible environment. It was also mentioned that the program should be able to run on Aalto Linux machines, using the readily installed libraries there. Make or Cmake is used for building and compiling the project and it should be able to adapt to different environments.

A graphical user interface could be a great way to visualize the simulation. The large cities could be problematic for the GUI, since it requires a lot of performance to visualize larger cities. The implementation of the GUI should be decoupled from the simulation itself. The early stage development involves making the software work without the GUI, meaning that the simulator provides only the analysis tools and results. After these basic functionalities, the GUI should be coupled with the simulator.

Analysis tools are a way of obtaining desired output from the simulation. The user should be able to:

- print out data to terminal
- analyze specific roads
- obtain a histogram of the amount of cars on the road in respect to hour of the day
- obtain a .csv file
- analyze multiple roads
- obtain average over multiple days to get more reliable results

The simulated city should contain roads with two lanes to opposite directions, intersections, cars, buildings, persons, speed limits and some traffic management (traffic lights etc.) These are discussed with more detail in the **Classes** section.

The simulator should be able to read user input. The user input could include:

- starting or pausing the simulation
- initialization
- retrieving analysis data
- adding cars
- loading a different city
- etc.

The loading of different cities should be possible from a JSON file into the simulator. The file contains all the needed information to initialize and start the simulation. The file format should include the data needed for the first stage of the simulation.

Randomness should be taken into account to avoid repetition of the same results. This provides more realistic circumstances and the simulation can be repeated to obtain different results. For instance, the cars should have randomness in their destinations and the time intervals should be randomized within a realistic scope, meaning that they are taken from realistic values (e.g. 15 - 30 minutes.)

EXTERNAL LIBRARIES

JSON library for parsing the loaded file json

GUI: SFML, QT

Maybe need an external library for parallel threads? QT

DIVISION OF WORK AND RESPONSIBILITIES BETWEEN THE GROUP MEMBERS

- Following the deadline schedule should be obeyed with the features and functionalities
    - If one deadline is achieved early, the next one should be started with regard to the amount of hours dedicated to that particular week. For instance, if the first deadline is achieved with 5 hours of workload and there is still 10 hours left, the leftover hours can be dedicated to the next features/deadline.
- The development of classes divided evenly
    - Have to think about the workload of particular functionalities; some may need more time than others

- Libraries should be studied independently, since they have features that are new to the group
    - Depending on the interests of the group members, the study of the libraries is going to be divided evenly, so that some libraries are studied briefly by the other members and other libraries are studied more profoundly
- Additional features are considered in this project plan, but these features are going to be ignored if some of the crucial features are not functioning correctly during final deadlines
- All of the group members are going to work on the main functionalities of the simulator. Some of the topics/classes are divided into:

- Artur

  Road, Analysis tools, (GUI), testing

- Jiri

  Traffic light, Intersection, testing

- Aape

  Building, Person, testing

- Jaakko

  Car, Lane, testing

Preliminary project schedule:

Date of completion	Action
5.11	Classes for buildings, roads, vehicles. Testing of classes and their functions.
12.11	Classes for intersections and persons. Function implementations, analysis tools
19.11	Loading from file, testing of simulator's main functionalities
26.11	GUI development and continuation of tests
3.12	Traffic management (Traffic lights, speed limits etc.)

6.12	Building profiles and person profiles
7-9.12	Final testing
10.12	Final version

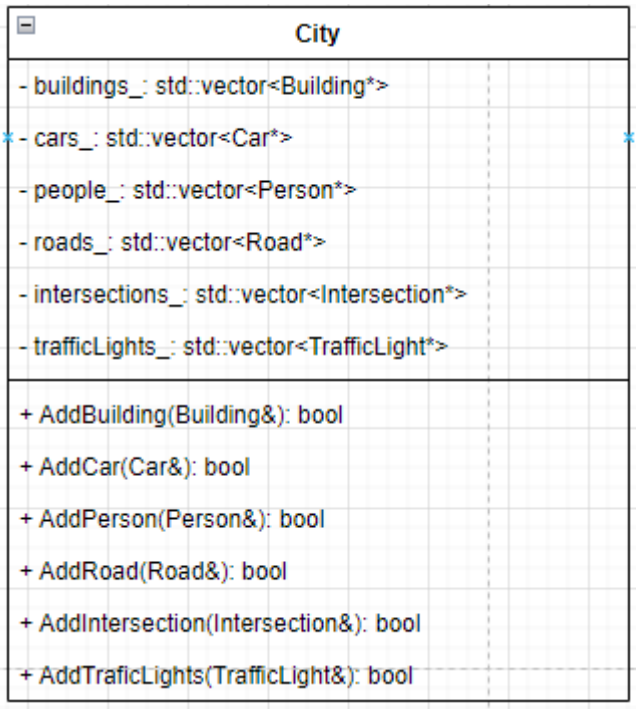
Classes:

(picture was too large for this pdf; included in a separate link)
Picture of UML Class Diagram

Classes in the UML-graph don't contain constructors or destructors because of the limited space.

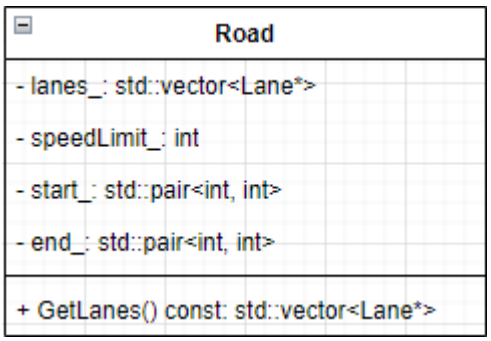
City:

- Contains all the roads, buildings, cars etc. A city object can be loaded from a JSON-file.



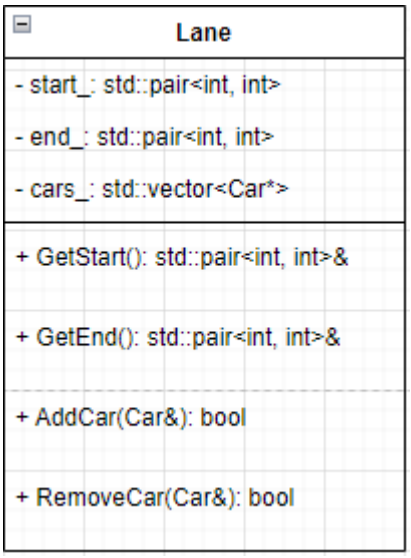
Road:

- Each road object contains two lanes, one for each direction. Road objects have a speed limit.



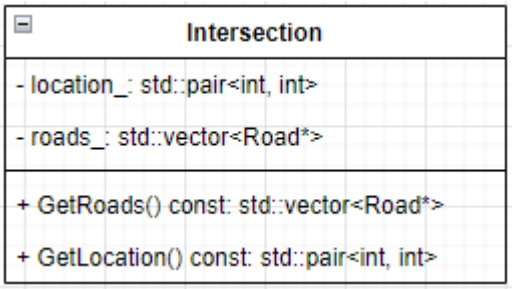
Lane:

- Each lane object has a starting point and an ending point. Lane objects contain a certain number of cars which cannot exceed the maximum number.



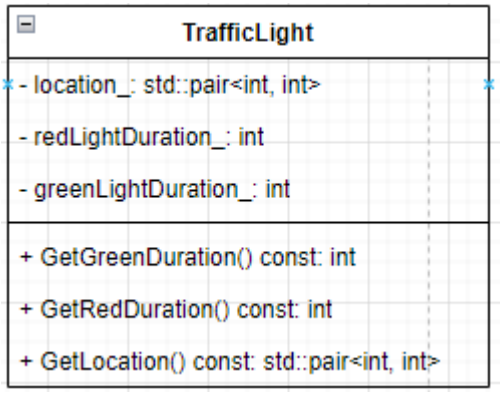
Intersection:

- Intersection objects connect three or more roads. Intersection object contains a location and a list of roads it connects.



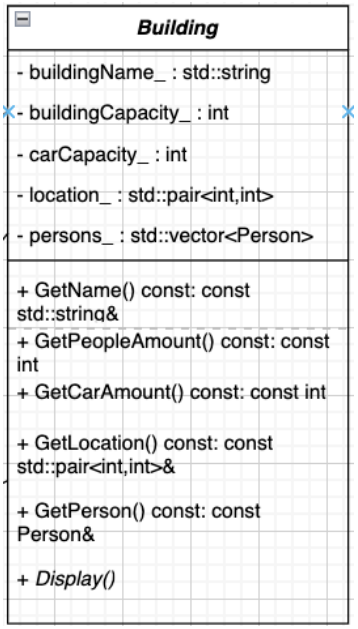
Traffic light:

- Intersections contain traffic lights which have certain times for red and green light durations.



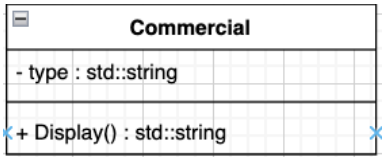
Building:

- Abstract class which is inherited by three classes: Commercial, Industrial, and Residential. People work in industrial buildings, shop/go to gym/eat in commercial buildings and live in residential buildings. People travel between these buildings according to their schedule, with some random humane deviation in respect to time. These buildings have capacity for people and cars which can't be exceeded. Location gives the coordinates of one building. Persons is the list of persons inside.



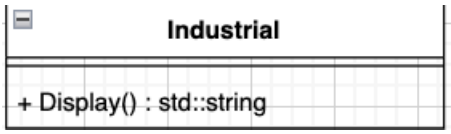
Commercial:

- Inherited from Building. Has a type (store, gym or restaurant).



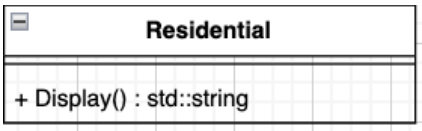
Industrial:

- Class for workplace, inherited from Building.



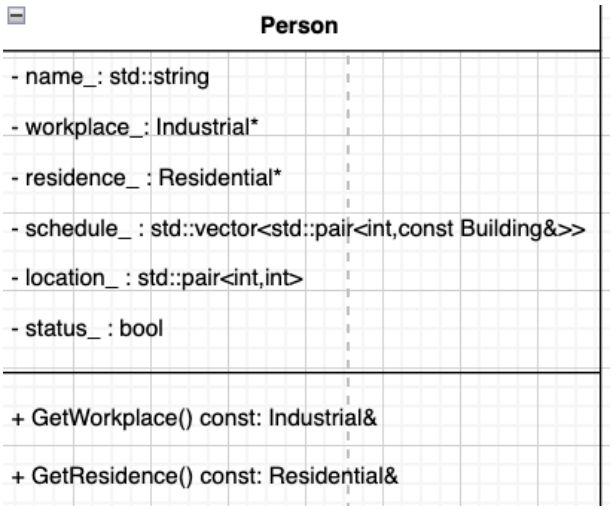
Residential:

- Class for residence, inherited from Building.



Person:

- Each person has a name, schedule for working and other activities, which is a vector of pairs, where each pair contains an hour and the activity (the building where it needs to go), home, and a workplace. Persons travel between the buildings according to their time schedules. Location gives the coordinates of the person and status gives if the person is busy at the moment.



Car:

- Each car has a register number, size, location, velocity, persons, starting building and destination building.