Aalto University
ELEC-A7151 Object oriented programming with C++
Year 2020

Project plan

Traffic Simulator-1

Date: 17.7.2020

Team members
Gyeha Lim
Tung Nguyen
Mahmoud Gadalla
Lac Truong (Sophie Truong)

Project Supervisor
Emma Turkulainen

Starting date
07.07.2020

Submitted date
28.08.2020

Outline

I) Scope

This project aims at simulating and post-processing traffic flows of city roads through a set of analytic functionalities. In particular, the output software is expected to operate on a map spatial data while identifying building blocks and connecting roads and intersections. Then, it simulates a traffic flow of cars holding passengers who move between buildings according to prescribed daily routines, although a bit of randomness is acknowledged. A final deliverable of this software is expected to support a Graphical User Interface (GUI) to enable direct interaction and manipulation of the data by users.

As a general note, the project seeks simplicity (KISS principle), robustness, data normalization (DRY principle), and modularity in the code design, such that a user could operate on an independent and interchangeable set of implementation details via an interface without impacting the downstream core, while a developer should easily read and improve the source code in its minimal version. In addition, since the project is a teamwork-based, consistency in the programming style and implementation is taken into account through following single-code standards. Finally, continuous integration and automated testing are planned to provide proper coverage for the source code statements, hence a maintainable and verified code is achieved.

II) High-Level Structure

1. Overall architecture

The application shall be constructed of 3 main parts:

a) Graphical User Interface(GUI):

The GUI will be the main interface where users interact with the software. We expect the interface to have an Import option where users can upload a map (in JSON format), and various options that help users to create traffic objects (for example: car, pedestrian). The software will perform a traffic simulator, while acknowledging randomness, which will be shown on the GUI.

b) External map imported by the user:

This map contains stationary objects like road, building, traffic light and traffic sign, bus stop and many more (depending on the utilized map)

c) Dynamic object:

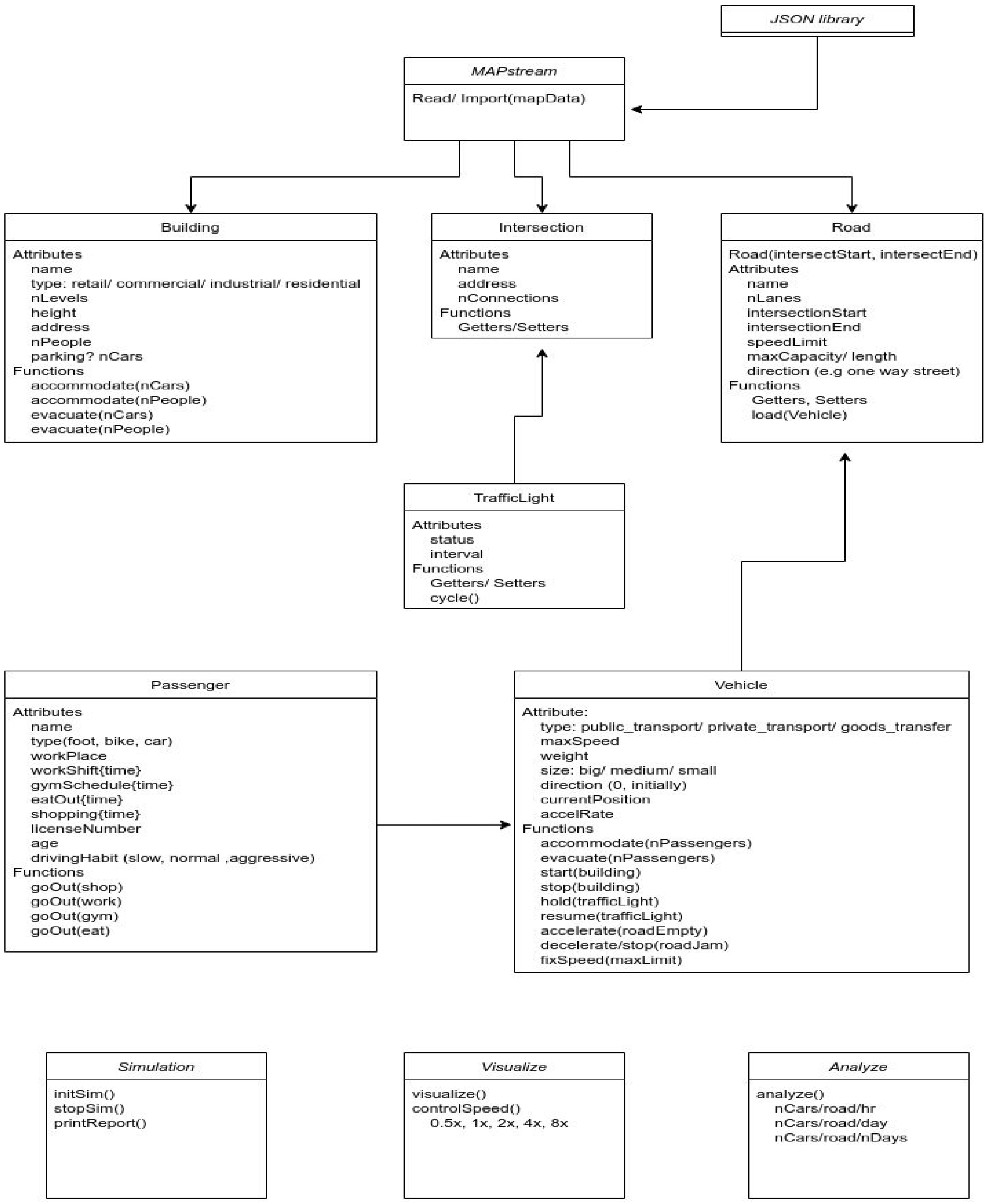
The dynamic object is the object that moves under the control of a randomized algorithm. This includes cars, pedestrians, and bicycles. They shall be created and placed on the map by the user using built-in functions.

The followings shall not be implemented for the simplicity of the app and time constraints:

- More specific types of vehicles, for example, motorbike, bikes etc.
- Collisions between traffic objects.

2. Classes

The program shall have these classes:



3. Relationship between classes

   The general framework is shown in the previous diagram. Detailed information considering specific interactions between class object will be determined during the implementation stage.

III) Instructions for building and using the software

1. Libraries
   a) Nlohmann JSON library

      An external library that allows the use of JSON objects and its functionalities in c++, including parsing. This is heavily unit-tested and covers 100% of the code, including all exceptional behavior. It can be easily imported with a single header file and thus it is easy to use. It may not offer the fastest performance, but its strength lies in its simplicity and safety.

   b) GUI libraries: We plan to use either SFML, FLTK or Wxwidgets.
   c) More will be determined during the development process.
2. Resources

   Simulation game - CIties: skylines :

   https://store.steampowered.com/app/255710/Cities_Skylines/

   www.openstreetmap.org
3. How to use (Makefile or CMake)

We plan to compile the software through a Make/CMake file, and also to provide one tutorial with documentation on how to use.

IV) Documentation and Testing

Documentation will be based on Doxygen, in addition to an overall customized document in pdf format. Unit tests are planned to be based on the Boost Test Library.

V) Division of work and responsibilities between the group

1. Method of management

We follow a combination of Waterfall and Agile Development method, where the team is divided into:

- Product owner: Tung Nguyen, assisted by Sophie Truong
- Scrum master: Mahmoud Gadalla, assisted by Gyeha Lim
- Development team: all members

We use Trello application to organize tasks between developers and keep tracking on individual roles.

2. Roles of members in the development

- Sophie: Graphical User Interface
- Mahmoud Gadalla and Gyeha Lim: Build classes for the various objects in the traffic simulation, as shown in the previous diagram.
- Tung Nguyen: Fetch the map from an external source and process the map into C++ class objects.

VI) Work Breakdown Structure

1) Work management
   a) All commits will be pushed to the Git repository, while following recommendations for commit info and network/graph smoothness.
   b) Task assignments will be communicated via Trello
2) Main modules for Buildings, Roads, Cars, Intersections, Passengers
   a) Modules for Cars, Passengers that can be added onto the map
   b) Modules for Traffic lights and signs
   c) Modules for Buildings, Roads, and Intersections that import their attributes from the map
   d) Define special properties for roads (e.g. maxSpeed) for cars to follow. Same goes for cars special properties for passengers to follow.
   e) Define a lane-changing behavior (If we allow more than 2 lanes per road)
3) IO Stream, city loading, and Analysis tools
   a) Find a JSON format map.
   b) Understand the structure of the JSON file in depth.
   c) Give the app the ability to import the external map(JSON format) and store it into the temporary memory.
   d) Create the Building and Road object from the fetched map.
4) GUI
   a) Find a good library with a clear tutorial for creating GUI
   b) Create a general template with layouts, buttons, etc.
   c) Integrate with the code
5) Each developer is responsible for providing respective unit tests and documentation for the implemented module.

Milestones and Timeline:

<u>1- Familiarize</u>

- Week 1 (18-25 July):

Sophie: Find a good library with a clear tutorial for creating GUI. Create a simple GUI layout

Gyeha & Mahmoud: Establish the detail relationship between classes (inheritance, the interaction between classes)

Gyeha & Mahmoud: Create modules for Car, Passengers, Traffic Lights, and Signs.

Mahmoud: Create template for documentation in Doxygen.

Tung: Find a map with sufficient information about the road, building, and intersection that is written in JSON format.

2- Main interface prior to temporal integration

- Week 2 (26 July - 1 August):

  Sophie: Finish GUI layout with the Importing and Exporting functionalities, and add a hook for simulation

  Tung: Able to parse the JSON map and convert it into class objects data type

  Gyeha & Mahmoud: Define behaviors of vehicles
- Week 3 (2-8 August) :

  Sophie: Finish GUI layout with the Importing functionality and add a hook for simulation

  Tung: Integrate the importing ability to the GUI.

  Gyeha & Mahmoud: Create modules for Road, intersections, and Buildings

- Week 4 (9-15 August):

  The team: Start integrating the JSON processor with the classes, and write a randomized algorithm to simulate the behavior of the objects.

3- Traffic simulation and post-processing

- Week 5 (16- 22 August):

  The team: Integrate the back-end of the program to the GUI, documentation and unit test

- Week 6 (23-28 August):

  The team: polish the GUI and finish the backlog of previous weeks.

  24 - 28th August: project demonstration

  28th August: deadline for project documentation and source code

END