Project Plan

Scope of the Project

First we aim to build a model as simple as possible with the minimum requirements, due to the tight course schedule. Our initial aim is to create four buildings with two streets (road vertexes) combining and a traffic light separating the streets (shown in Figure 1). We will program cars to travel between the buildings with exponentially distributed exit intervals. In this initial model we will assume the behaviour of drivers to be ideal (no collisions, speeding or ignoring traffic rules). The drivers can also detect other cars and take into consideration other cars that it might encounter (avoids collisions). Primitive graphics (e.g. SFML library) need to be implemented in order to test the simulation.

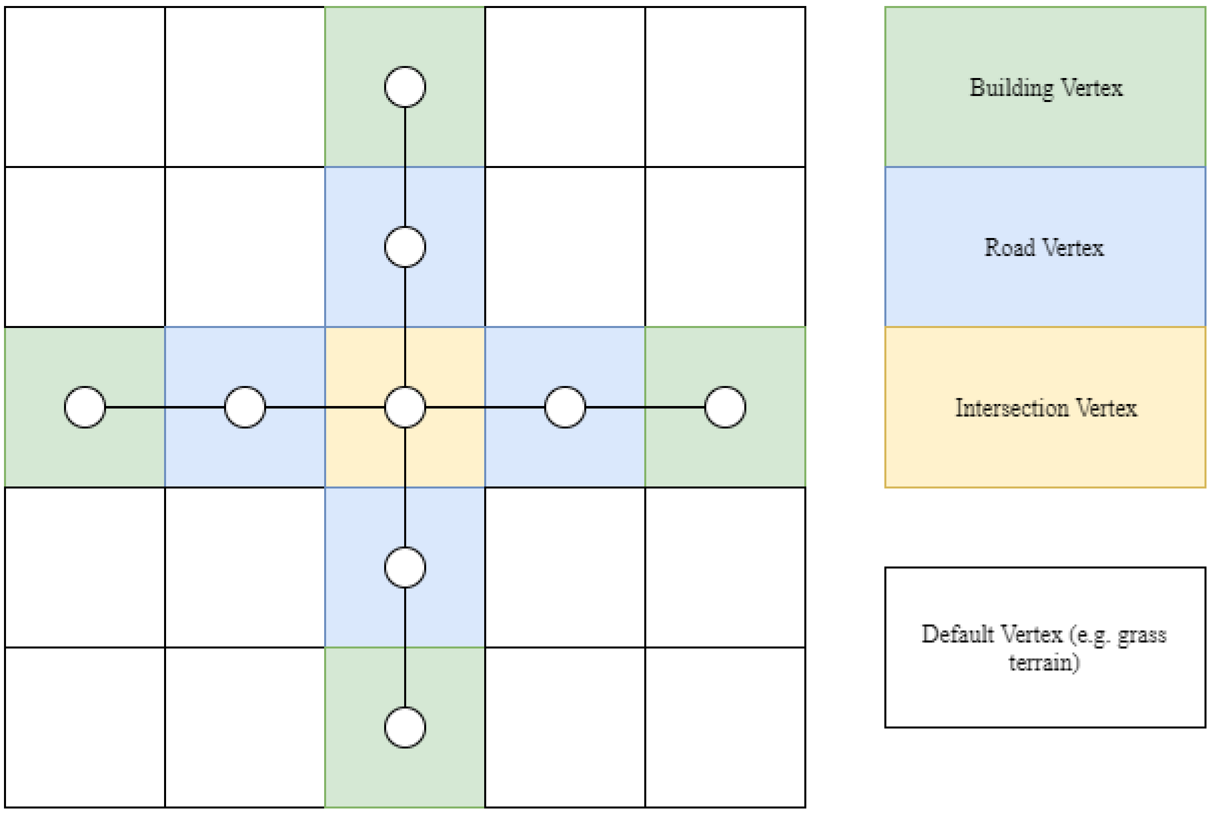

Figure 1: A simple example of 5×5 size map with basic graph structure. The intersection vertex contains traffic light, which guides the flow of the traffic.

Once we have achieved this basic level of functionality, additional functionalities will be created in the following priority order:

1) Adding custom vertexes (buildings, streets and traffic lights) into the predetermined n × m sized vertex matrix grid.
2) Creating the tools for interactive addition and removal of objects
3) Adding more pleasing graphics and creating different types of building vertices
4) Adding a fast forward tool
5) Creating a more complex example system

Major architectural decisions

Our architecture is divided into modules, to clarify the high-level class structure. Figure 2 will also clarify the relationships between the modules, which are listed below:

- Main (World)
    - Time-simulation, tick
    - Read user input from user interface and control program
    - Call calculation with graph and vehicles
- (Directed) Graph
    - Vertex (forced to XY-coordinates for UI)
        - Intersection
        - Spawner/despawner
    - Edge
    - Vehicle
    - Traffic light (possibly a subclass of Vertex)
- User interface:
    - Add and remove edges (roads) and vertices (intersections, spawners)
    - Spawner: Change frequency (rate of spawning vehicles)
    - Ability to fast forward

Currently we plan to use SFML external library to help with the graphics. We will also utilize a pathfinding algorithm (most likely A* or Dijkstra's algorithm) for vehicle pathfinding.

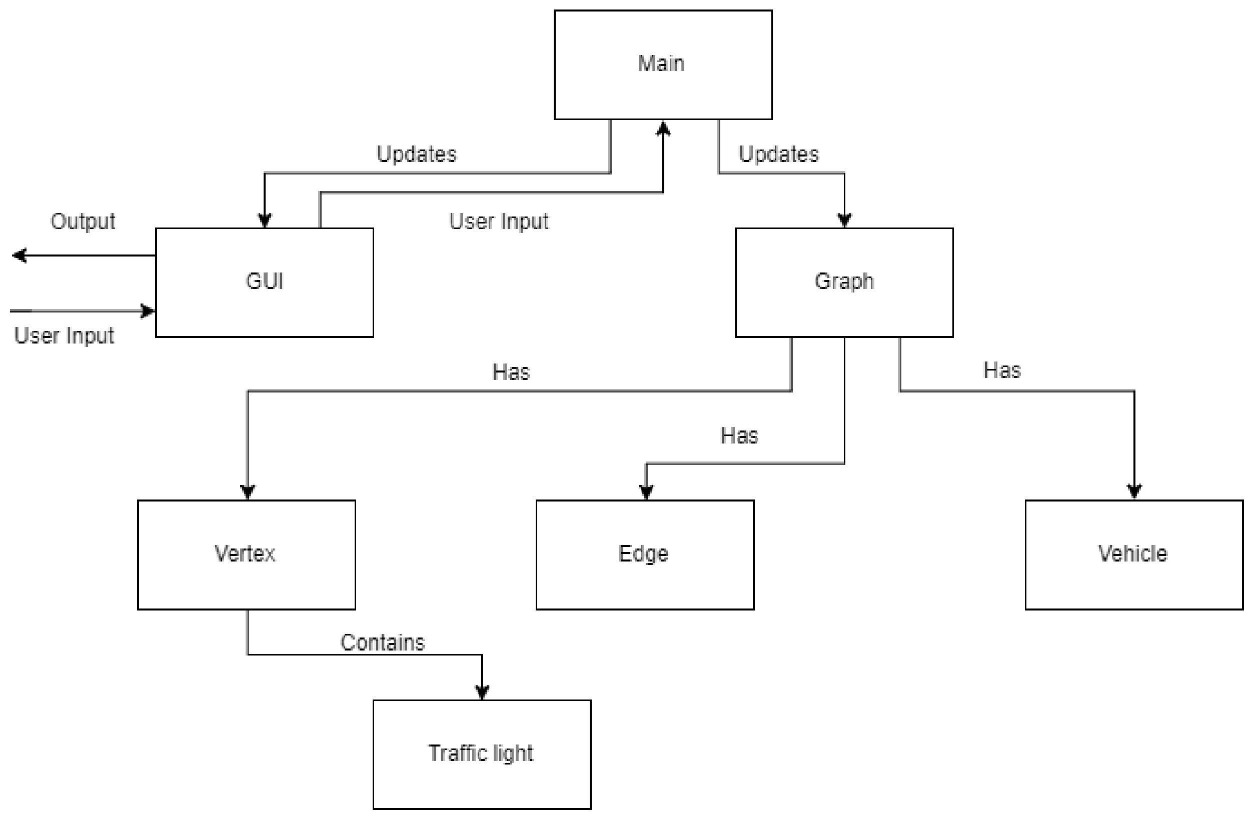

Figure 2: First sketch of our major architectural decision

Preliminary schedule

Our rough schedule (with dates) has been listed below.

- *Project plan and architecture, (17.11.)*
- Rough prototype:
    - ability to draw something on the screen, **(23.11.)**
    - working primitive graphs, **(30.11.)**
    - vehicles moving in the graph, **(7.12.)**
    - traffic lights, **(7.12.)**
    - working user interface, **(7.12.)**
- Finished software, **(8.12.)**
    - Must be able to compile using Aalto Linux computers
- Demo, (at the earliest: 11.12.)

To ensure that we are able to develop our project under strict schedule, we have weekly meeting on Thursdays to check the progress of our traffic simulation project. Milestones have been put to the corresponding dates, with the exception of the finished software and demonstration.

Distribution of roles in the group

During the planning phase, we noticed that our project has little to gain from a rigid job distribution. Naturally we will divide different portions of planned work for different individuals, but as we aren't very familiar with each other nor our skill level differences we feel it is best to let specializations emerge on their own as everybody starts finding their strengths and weaknesses compared to the others. However, this is how we initially planned to divide the workload.

- Tianzhong: Project coordinator
- Harti: Responsible for documentation
- Ilari: Responsible for graphics
- Risto: Lead Programmer

Design rationale

We initially divided the program into 7 modules, that each have their own functionality. We decided it would be a good idea to have a main module that controls all the other modules in the program.

The users interact with the main module through the user interface. The user interface is therefore its own module, that interacts with the main and the user. It also draws a picture on the screen and takes commands from buttons drawn on the screen.

All the actual elements needed for running the user interface are contained in the graph module. It houses the vertices and edges needed for the graph, the vehicles on the graph, the calculations of paths in the graph and the positions of all the elements.

The vertices, edges and vehicles are all their own separate classes/modules. This helps to make the code simpler and more intuitive, improves encapsulation and makes it easier to divide work between team members. The exact nature of the traffic light class/module is still quite unclear, but we expect to solve this problem while doing the actual programming.