Aalto University
School of Electrical Engineering

ELEC-A7150 C++ PROGRAMMING

SIM-CITY/TRAFFIC SIMULATOR

Project Plan

Group 2:
Francis SHIH
Rui SILVA
João GONÇALVES
Marco PEREIRA

Supervisor:
Pasi SAROLAHTI

November 16, 2017

Contents

1 Introduction

The following report will serve to illustrate the overall planning, decision making and general scope of the project.

This project is meant to both consolidate and further develop the C++ programming skills acquired during period I of this semester. As such, the group is expected to demonstrate what it learned during that period, as well as search for new solutions needed, for example, the necessity to implement a graphical interface.

It is expected, as well, that the group work simulates an environment of actual software development. Proper techniques, tools and conventions are necessary. The use of Git, for example, is a must in software development, and it is going to be used, for the first time for many, in this project.

The group chose to implement a Sim-city/Traffic simulator game. The main objective scope of this project theme is to implement a functioning traffic simulator, with varying volumes of traffic, avoiding collisions and, after that, to add complexity to both the traffic system and to the world building capabilities. Classes like the map, roads, traffic lights and cars are the ones that will get the greatest priority in implementation, along with their interactions, to create the functioning base game.

The plans for the game are also create a sandbox like simulator, where the user can create what he wants, without money restrictions, for example. Building roads, buildings and such won't imply a cost. And so, there will exist no scoring or reward mechanics, as it is not necessary.

In appendix A there is an UML diagram, demonstrating the class relations in a more descriptive manner.

2 Libraries

As recommended, we will implement this game using the SFML graphic library, which is intalled on Aalto Linux machines.

3 Algorithms

We will use a directed graph since there will be one-way roads. To define routes for each vehicle, we plan to implement a random mechanism, as well as an algorithm that minimizes distance, such as Dijkstra.

4 Class structure

The UML diagram can be found in appendix A.

4.1 Map

The `Map` class will contain all the elements of the game, e.g. `MapItem` objects that can be:

UnsignedItem: used at the start of the game

Route: can have a `TrafficLight` that regulates the circulation. It will be used for the shortest path algorithm as it is equivalent to a directed arc.

Parking: contains vehicles traveling the map

4.2 Vehicle

The `Vehicle` class is an abstract class. It'll be the base class for `Car`, `Motorbike` and `Bus`.

4.3 Position

All objects in the map have a `Position`. This class will be used for the shortest path algorithm since it can be considered as a node.

5 Project schedule

The project deadline is **December 15th**, and we will try to add features gradually according to the following plan.

Week 1 - 20/11-26/11

- Fundamentals for the user interface
- Setting the basic behaviour of vehicles in standard roads

Week 2 - 27/11-3/12

- Map editor, able to save and load from files
- Algorithms for path definition

Week 3 - 4/12-10/12

- Buildings, parking, linking to vehicle behaviour
- Updating the traffic engine, with one-way roads and different speeds

Week 4 - 11/12-15/12

- Extra features:
  - Time warp
  - Busy road highlighter
  - Special vehicles: ambulances, firetrucks, etc.
- Documentation

6 Roles

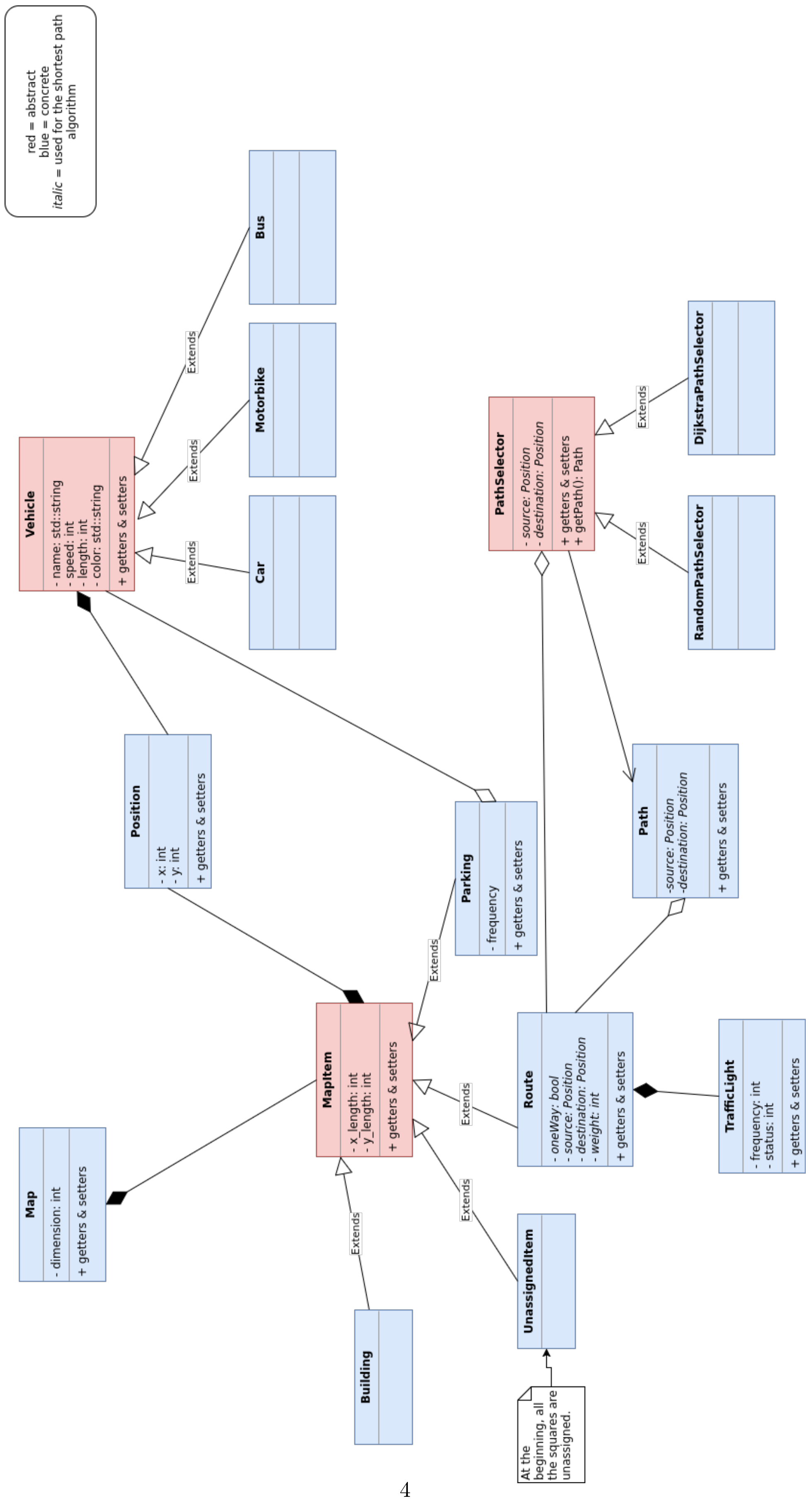
- Project manager: Francis
- Developers: Everyone

with main focus:

- Graphical user interface: Francis, João
- Map construction and path algorithms: Rui, Marco
- Vehicle dynamics: Marco, Francis
- Traffic generation: João, Rui

Documentation and testing done on the go.

A UML diagram



red = abstract
blue = concrete
italic = used for the shortest path algorithm
Vehicle
- name: std::string
- speed: int
- length: int
- color: std::string
+ getters & setters
Bus
Motorbike
Car
Extends
PathSelector
- *source: Position*
- *destination: Position*
+ getters & setters
+ getPath(): Path
DijkstraPathSelector
RandomPathSelector
Position
- x: int
- y: int
+ getters & setters
Parking
- frequency
+ getters & setters
Path
-source: Position
-destination: Position
+ getters & setters
MapItem
- x_length: int
- y_length: int
+ getters & setters
Route
- *oneWay: bool*
- *source: Position*
- *destination: Position*
- *weight: int*
+ getters & setters
TrafficLight
- frequency: int
- status: int
+ getters & setters
Map
- dimension: int
+ getters & setters
Building
UnassignedItem
At the beginning, all the squares are unassigned.