Review form for project XXX

Name of project to be reviewed:

Own Project: traffic-simulation-2019-2

Names of reviewers:

Tommi Gröhn, Kaisa Kärkkäinen, Akseli Oinaanoja, Stas Tatun

Provide short comments (2-4 sentences) for each item below.

1. Overall design and functionality (0-6p)

** 1.1: The implementation corresponds to the selected topic and scope. The extent of project is large enough to accommodate work for everyone (2p)*

Our implementation corresponded well the selected scope. First of all, we used real-world maps which we analyzed. Even if the maps we used were complicated and realistic, we managed to create all the basic features and many from the additional features. The project had enough to do for us: read an openstreet-map, compress the data into a graph, create a logic for traffic simulation, add some randomness to create realistic traffics and of course implement GUI.

** 1.2: The software structure is appropriate, clear and well documented. e.g. class structure is justified, inheritance used where appropriate, information hiding is implemented as appropriate. (2p)*

Our group thinks the software structure works well. Class structure is simple: essentially the code works with classes mainwindow, city, road, car and node. However, there were some parts in the code we did something too complicated than needed. Some functions could have been defined as const. We tried to add a lot of comments before every function to make sure an outside reader would have a realistic chance to understand our implementation.

** 1.3: Use of external libraries is justified and well documented. (2p)*

We used Qt5 in creating the GUI for the simulation and rapidXML library for parsing the XML-files that are exported from OpenStreetMap. The use of Qt5 was justified, because it was a familiar library for GUI developers (Akseli and Kaisa). RapidXML was found to be efficient and easy-to-use in parsing the map files. The use of these libraries is mentioned in the project documentation, and do not require much, if anything, from the user.

2. Working practices (0-6p)

** 2.1: Git is used appropriately (e.g., commits are logical and frequent enough, commit logs are descriptive). (2 p)*

Git was used properly during the whole project. The commits started coming later in the schedule, but this was something we agreed upon. The idea was to go through the thought process first together during the first weeks and then start committing first pseudocode, then moving onto the actual compiling and running code. Commit logs are not always as descriptive as they could be. However, the commits were always discussed together and we often confirmed that everyone was on the same page.

** 2.2: Work is distributed and organised well. Everyone contributes to the project and has a relevant role that matches his/her skills. The distribution of roles is described well enough. (2p)*

The work distribution was very well handled, especially in the sense of matching everyone's individual skillset. Stas had the most experience in algorithms and data structures, therefore being interested in the backbone of the program, parsing XML-files and implementing network algorithms that determine e.g. the routes of cars from home to work. Tommi, as mathematically oriented student, was interested in implementing randomness and stochastics to the simulation, because obviously we didn't have real world data of e.g. traffic during the day vs. during the night. Akseli and Kaisa were interested in creating the GUI and its features as they had some previous experiences from other courses. Qt was also somewhat familiar to them through python courses and PyQt.

** 2.3: Quality assurance is appropriate. Implementation is tested comprehensively and those testing principles are well documented. (2p)*

The program was run a lot during development. Especially when GUI features started adding complexity to the behavior of the simulation and bending rules, (e.g. setting the time arbitrarily) testing was crucial. We tested thoroughly the interplay between additional features (e.g. Set Time and Change File) as soon as the baseline simulation was found to work sufficiently on its own. The testing should have been documented more comprehensively when considering non-project members looking at the documentation, which was a clear mistake from our side. However for us, everything about the testing was clear and routinely organized.

3. Implementation aspects (0-8p)

** 3.1: Building the software is easy and well documented. CMake or such tool is highly recommended. (2p)*

In read.me there are reasonable explanations how to run our program. We had one test person who could use build our software well, which gave us confidence the documentation cannot be too bad.

** 3.2: Memory management is robust, well-organised and coherent. E.g., smart pointers are used where appropriate or RO3/5 is followed. The memory management practices should be documented. (2p)*

Memory management is something we could have done better. Mostly the reason is that we concentrated mostly on the logic and GUI and we started focusing on memory later on the project. However, we had some clever memory management: for example mainwindow creates a new city while changing a file.

** 3.3: C++ standard library is used where appropriate. For example, containers are used instead of own solutions where it makes sense. (2 p)*

We tried to use the standard library as much as we could. However, we could maybe have tried to find more other alternatives outside standard library to make our code more efficient.

** 3.4: Implementation works robustly also in exceptional situations. E.g., functions can survive invalid inputs and exception handling is used where appropriate. (2p)*

The program is designed to handle exceptional situations, to a sensible degree. For example, the GUI implementation includes quite a few try-catch expressions to guard the program from crashing when the user provides a bad input. Instead, the program almost always announces this and tries to describe the fault in the user's actions e.g. through error message boxes. However, these catches are designed to grab the most common and humane errors but not everything. In our opinion, this is justified because we wanted to keep the code as clean as possible but also provide instructions for the most common mistakes that can be made by the user.

4. Project extensiveness (0-10p)

** Project contains features beyond the minimal requirements: Most of the projects list additional features which can be implemented for more points. Teams can also suggest their own custom features, though they have to be in the scope of the project and approved by the course assistant who is overseeing the project. (0-10p)*

Naturally, we implemented all the basic features. We also implemented the following additional features which were listed on the topic description:

1. GUI: our GUI visualized complicated road structures based on real-world data. We also had multiple features in GUI: pause and continue, shift time, change map, choose road to analyze and draw histogram of the total traffic.

2. Passenger profiles: In our simulation was considered people’s usual working times as well their free time preferences. For example we took care of the fact people usually go to work in the morning and sleep at nights.

Next, we would like to emphasize other features or implementations in our project that were not included in the basic features:

1. Our program creates a simulation from real-world data (openstreetmap).

2. Our program has a lot of realistic randomness in it because we used Poisson distribution for creating new drives in the city. The activity of people is easily changeable. By going on the field and getting an approximation for people’s activity would enable getting quite realistic results for the simulation.

3. Our program figures out the optimal routes using Dijkstra's algorithm.