Self-review form for project traffic-simulator-2019-1

**Name of project to be reviewed:

- Traffic Simulator 2019 - Group 1

**Names of reviewers:

- Jonathan Leinola - 477329
- Niko Uski - 51961T
- Artur Gynter - 569499
- Hien Cao - 716718

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)**

The implementation covers almost all of the basic features, additional features, and advanced features. So the project was wide enough to accommodate work for everyone. There were four unofficial goals we set for the application: the program to be scalable, use automated memory management and proper containers, make consistent/clean code and make it stable. We think we accomplished these goals during the project and we are very satisfied with the end result.

*** 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)**

Inheritance was used to store different types of Tiles in the same array and also it helped to minimize the amount of duplicate code. Also inheritance made program more scalable for future development. Only needed functions were public.

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

External libraries we used were SFML, ImGui and SFML + Imgui v2.1. ImGui was chosen for the project because of our intention to implement GUI and SFML was

chosen simply because it allows GPU rendering with little to no knowledge of OpenGL. GPU rendering was necessary for performance because of a huge amount of vertices like cars and roads. SFML + ImGui provided easy bindings to draw GUI with ImGui.

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)**

There are about 243 commits that are logical and they are spread smoothly along with the project schedule. Commit logs are pretty descriptive and always written in the same language in English. All the commits were tested before pushing to the project's repository.

*** 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)**

From the final project documentation, it's easy to see that the work is distributed and organized well and logically. Everybody had their own area of responsibility which matched their skills and competences. Project was started early and we had a big part of the program done before the first meeting.

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

Quality assurance is ensured through testing by team members. The testing is based on the features indicated in the project plan. The bugs were taken note and discussed with other members via group chat channel and physical meeting weekly. In addition, some use cases were generated and tested to reduce the bugs happening during the usage.

3. Implementation aspects (0-8p)

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

In our project documentation, there are simple instructions on how to build and run the program. There are only five steps users need to take to execute the

program. Our system uses CMake. Also our repository contains prebuilt dependency libraries so that user/tester doesn't have to worry about installing dependencies. This made execution possible on Linux as well as Mac OS. machines. The course assistant was able to compile and run the program successfully with the first try.

*** 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)**
From the beginning of the project, we decided to use smart pointers and avoid using any new statements in our code. The reason for using smart pointers was to keep memory management automated. We accomplished the project without any new statement and our memory management worked as we planned.

*** 3.3: C++ standard library is used where appropriate. For example, containers are used instead of own solutions where it makes sense. (2 p)**
Smart pointers, containers such as vector and array are utilized well and there is a reason for each container why we used exactly it. Also lambdas used with standard library for example: std::sort and std::remove_if. Also we decided to go with C++ 14 because it introduced new things to C++ like std::make:unique.

*** 3.4: Implementation works robustly also in exceptional situations. E.g., functions can survive invalid inputs and exception handling is used where appropriate. (2p)**
Error handling is implemented for some part of the project such as loading and saving the map, and input file's name.

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)**

As fulfilling all the minimal requirements, the additional and advanced features are also implemented. The detail for the additional and advanced can be found in the project documentation. Biggest feature of the project was MapBuilder. It

accounts for roughly 25% of the whole code. It has a lot of “quality of life” features that makes map building a lot easier and quicker: intersection templates, slide addition and deletion with hotkeys, traffic light network visualization and simple GUI to operate the whole thing. In addition, the application was added custom features such as A* and DFS pathfinder algorithm, heat map and mini map.