ELEC-A7150 C++ Programming
Autumn 2017

Project plan

Micro Machines

Miika Karsimus 479437
Martin Vidjeskog 432539
Vili Karilas 552794
Hannes Pitkänen 475127

Table of contents

General **2**
Project topic 2
Project goals 2
Schedule and division of labor 3
Methods of work and communication 3
Program structure **4**
Major architectural decisions 4
Class diagram 4
Files 5
Coding style 5
External libraries **6**
SFML 6
Building and testing **6**
Building 6
Unit tests 6
What else 6

General

Project topic

"Micro Machines is a legendary driving game from the 90s in which you race with little toy cars in everyday-like environments such as tabletops, floors or bathtubs. Your task is create a micro-machines-like game or similar top-down driving game."

This was the course description for the Micro Machines, an old driving game from the last millennium. Our version of the game however, will differ quite a lot from the original 90s gem. It will be more like a reduced clone, which will focus more on the basics and puts less effort to the eye-catching graphics. In addition to the reduced graphics, it will most likely lack some of the gameplay features from the original game. More about the goals is discussed in the next chapter.

Micro Machines (http://games4win.com/games/micro-machines-2/)

Project goals

As we mentioned in the previous chapter, the purpose of the project is to develop a clone version of the Micro Machines driving game. We intend that the final program meets the requirements/goals of the list below:

- Fun to play and has no major bugs
- Never crashes
- All the minimal requirements and some extra features, including weapons, sounds, different vehicles and possibly a simple artificial intelligence
- Random track generator
- Filehandler (loading/saving tracks)

Schedule and division of labor

The schedule will be very strict. There will not be room for huge errors. In practice, the basic features should be implemented about two weeks before deadline. After that, we can try to implement some extra features if there is still time remaining. We will start by implementing one car and a simple track with a few obstacles. When this is done, we can implement more cars, more complex tracks and other features.

1. Some kind of menu (and window)
2. Vehicles (drawing and physics)
3. Tracks (Random generator)
4. AI (opponents and their logic)
5. Weapons
6. Game modes
7. FileHandler (save and load tracks)
8. Fancy details

Distribution of roles is the following (Not final, probably will change later):

- Miika: Vehicle physics, vehicle controls, camera and view control

- Martin: Maps, AI

- Hannes: Weapons, file handling, drawing graphics

- Vili: Sound effects, main menu

Methods of work and communication

The main communication channels which we are going to use, are the telegram and slack applications. Most of the work is meant to be done remotely. Even if most of the work is done remotely, we will try to organize weekly meetings.

Git version control system will be the most important tool used in this project. Also, we will probably use Doxygen to document the code but this is not topical at the beginning.

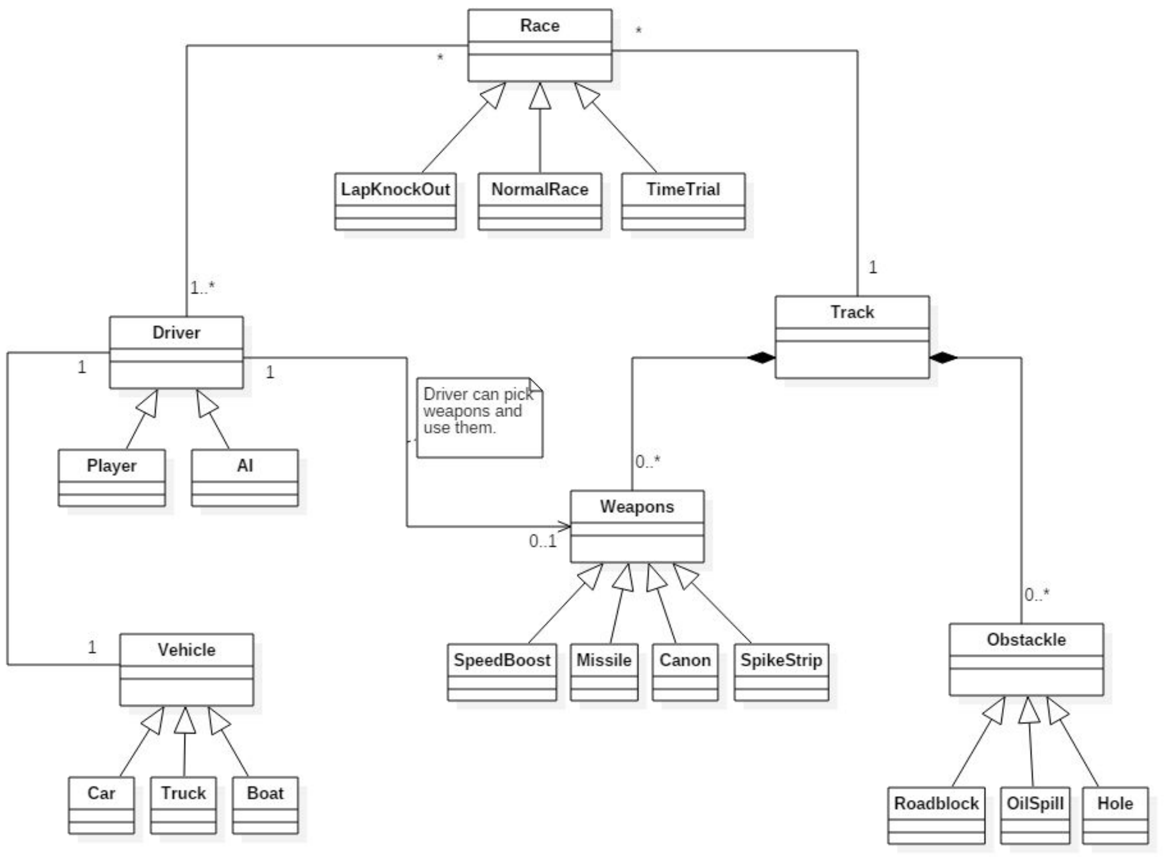
Program structure

Major architectural decisions

Class structure should be natural, meaning that a single entity, for example, car, map, or driver, is represented in a single class. Classes, which have similar features, should inherit the base class. For example, class Vehicle is the base class for classes Car, Truck and Boat.

Class diagram

The first version of the class diagram is presented below. The diagram has been kept very minimal, thus no attributes or methods are marked. Also, the functionality of the game is not included in the diagram. It only shows the most important classes for the program and how they are linked together. In addition to the classes in the diagram, there will be more classes related to GUI, for example, main menu.



Files

The maps will be saved in files. We have not decided the file format yet. However, it is probably XML, JSON or CSV, because files with these formats are easy to handle. In the first phase, one or two maps can be raw-coded and later we will start to implement a file parser.

Coding style

To keep the program code coherent and clear we have decided to follow five basic guidelines. First of all, the code should be well commented, so that there will not be any mystery parts. This is especially important, because there are four people working on this project. This does not mean that every line should be commented, but at least every method should. The function of the method and possible input and output parameters should be briefly explained. Of course, if there are lines that are hard to follow, they should be commented. Also, on top of every class in header file, there should be a short description what the class does and what methods it has.

The second rule, which we have agreed to follow, is the single responsibility principle. This principle states that every class should have a responsibility over a single part of the functionality provided by the game.

The third rule concerns the designation of variables. The chosen practise is camel case. This means that the words in names of the variables, functions, methods or classes are written together and the first letter of each word is capitalized. For classes we use upper camel case (initial upper case letter) and for everything else, lower camel case (initial lower case). Of course, encapsulation will be applied, meaning that a single attribute or function is private if there are is not good reason to make it public. In practice, most part of functions will be public, while the most part of member variables will be private.

According to the fourth guideline, we do not use statements like “using namespace std;” or “using namespace sf”. This makes it easier to follow in which library the class belongs to or whether it is self-made.

The final rule applies to classes. Each class should be divided into two files (cpp and hpp). The hpp-files should include definitions of classes and methods. The cpp-files should include the implementations for the methods defined in hpp-files. The classes might be wrapped into namespaces.

In addition to these five rules, there are many other principles that we will try to follow. We try to avoid allocate memory with “new”-keyword. Instead smart-pointers should be used whenever possible. Also, we prefer to use references instead of pointers, if possible (again).

External libraries

SFML

We are going to draw the graphics by using SFML graphics library. SFML is simple and well-suited for 2D game development and it was recommended by course staff. It is also documented well and good tutorials can be found from their site.

We are probably not going to use Box2D for driving physics. If it turns out to be too difficult to implement physics by ourselves, we consider to start using Box2D.

We will possibly use some XML-parser library if we end up to use XML as file format. Yet, this is not sure and it is also possible that we implement the file handler by ourselves.

Building and testing

Building

The program is build using CMake and Make. It should be possible to compile the program with Linux, Mac OS and Windows, because CMake is platform-independent. Object files, binary files and other build files are located in a separate build directory which must be created before building the program. This ensures that the file structure remains clear, and build files are separated from source files. Note that this build directory is not under version control. Everyone who compiles the program on their own machine, has a build directory with different content, depending on the OS and building tools which they use.

Unit tests

We will write a couple of unit tests with Google Test Framework. Valgrind will be used to avoid memory leaks. Of course, the best way to avoid memory leaks, is to use smart pointers instead of manually allocated dynamic memory.

What else

At different stages of development, we strive to ensure that the program works with Aalto's linux machines. In addition, we try to make sure it works on windows machines as well.