ELEC-A7150 - C++ Programming
Q-learning 3
Miisa Martio, Kuura Parkkola, Toni Pellinen, Nuutti Sten

C++ project documentation

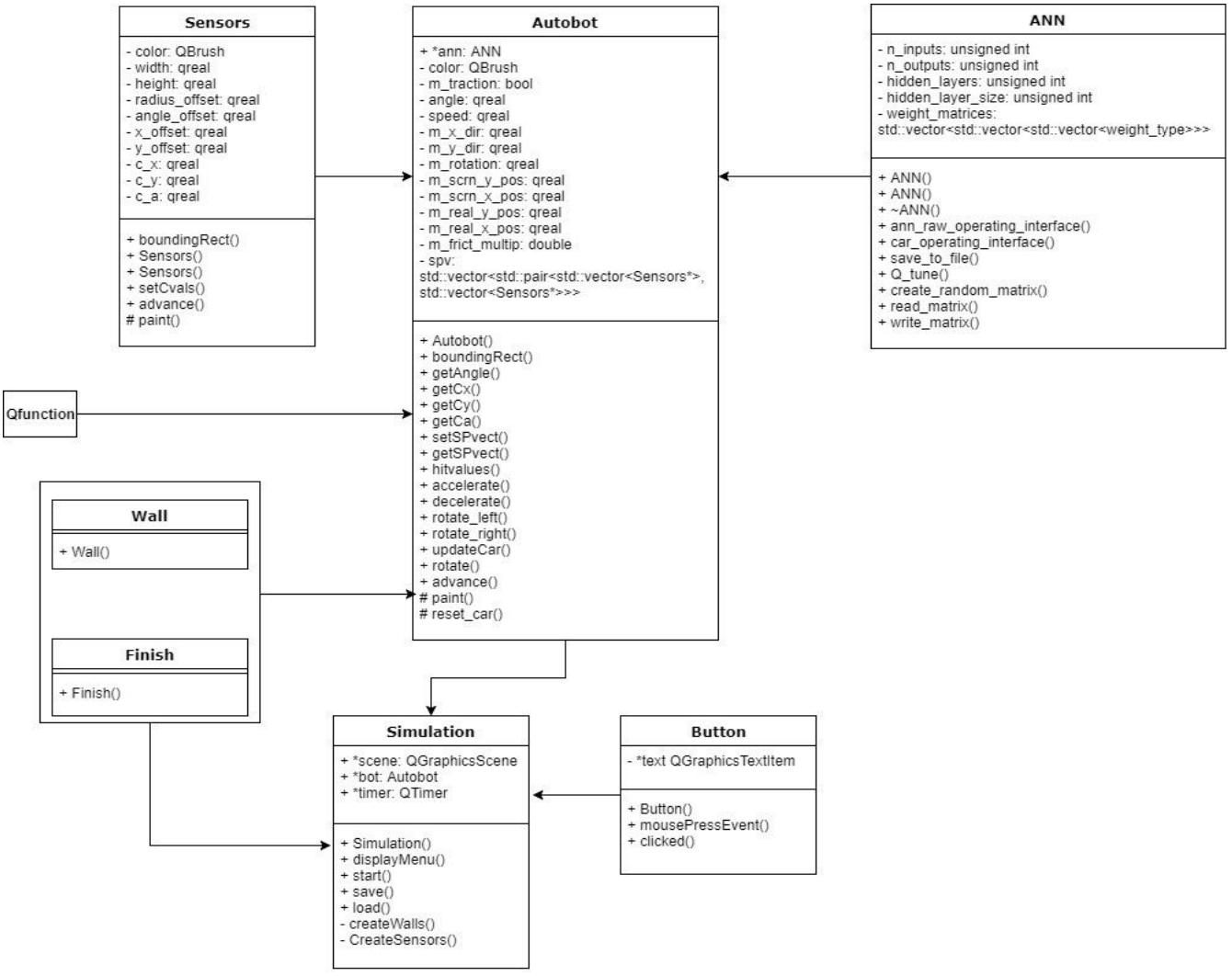
Overview

The goal of our project was to teach a car to ride using a q-learning algorithm and an artificial neural network. As claimed in our project plan, originally we were supposed to reach at least the minimum requirements for the project.

The main parts of the software are the q-learning algorithm which is teaching the neural network, graphics, physics and sensors for observing the surroundings of the car. Now, at the end of this project we have these implemented and the program is working as hoped for. The car is driving on its own and it’s learning to drive without crashing into walls or driving too fast. We’ve also added a possibility to save the car’s progress to a file.

Software structure

The program consists of seven classes and the relations between these classes are described in the diagram below. The main function calls the Simulation class which creates the car, the sensors and the walls.



We used Qt libraries for implementing the car, sensors and walls. The Qt versions we used are 5.9.3 and 5.10.0. In the above diagram you can see how these classes are related to each other. In addition, the classes *Button*, *Sensors* are inherited from

a class called QGraphicsObject, the class *Simulation* is inherited from *QGraphicsView* and the classes *Button*, *Wall* and *Finish* are inherited from classes *QObject* and *QGraphicsRectItem*. *QObject* is in module *QtCore* and *QGraphicsObject*, *QGraphicsRectItem* and *QGraphicsView* are in module *QtWidgets*. Along with these modules we also used the module *QtGui*.

Instructions for building and using

1. Compiling the program
    - Instructions here.

2. Using the software
    - After the program has been compiled, it should run on its own. No actions from the user are required except clicking the start button. So, after compiling, using the program should be quite easy.

Testing

Since we all had our own responsibilities, the program has been tested in small fractions. The q-learning algorithm and the neural network were tested own their own and so were the graphics and the car. The sensors and the physics were added after the graphics and the car were done and they were also both tested separately. We used trial and error for most of the testing and after all, ended up with a working program.

We tested the program on Windows, Mac OS X and Linux. We also used different IDEs for testing the program.

Work log

Miisa:

Originally, I was supposed to implement the car and its sensors. The first week of the project was busy for me, so I didn't have much time for the project.

On the second week, after I had survived the first week, I began learning about q-learning and how the sensors could be implemented. This continued also to the third week because it was quite difficult to start implementing the sensors without the car being implemented. Since we ended up implementing the sensors with Qt, it turned out that many of the things concerning the sensors I'd found so far were quite useless. Under the second and third weeks I used roughly ten hours on the project.

On the last week of the project, I had a lot of time issues since the program wasn't the only thing taking up my time. However, I started to plan the implementation for the sensors again, so that it would fit for the car and the environment created with Qt. I also had to learn more about Qt and finally ended up trying an awfully slow algorithm. After more learning and thinking I succeeded to come up with a somewhat reasonable structure for the sensors. I also tried my best to help other group members with their work and figure out possible problems. At the very end of the last week I wrote most of this documentation and created for example the diagram. Also, after all, I didn't end up implementing the car since I think Toni was more qualified with using Qt and it was probably easier to implement along with the graphics. In the fourth week I used up to about 30 hours with the project.

Toni:
My responsibility was originally to implement the graphics. First week was very busy, so I didn't have time for the project back then.

On the second week I began to get familiar with the Qt library and to learn how it actually works to create a Qt based program with C++. At the end of second week I was quite familiar with the basics, and I also realized that Qt is going to be so strongly connected with the rest of the program, that I created a simple window with a simple car in it that used to drive with keyboard mouse buttons. That could be called as a first version of our program, though it was more like a draft. There is not much original code left from that version, because the car class have been improved and reorganized dozens of times.

At the third week I created the walls and special finish-line-wall for the car. I also made the car interact with the walls when car collided with them. During the third week I helped a lot my group members to implement their part of the code to the "main part". There was some conflicts because neural network was developed separately from the rest of the program, but nothing we couldn't overcome.

Fourth week was the busiest of them all. Our main function looked so awful, that I removed everything from there and created a separate "simulation" class for that stuff. I also implemented four simple buttons for the main window which all had their own functionalities. Rest of the last week was figuring out with my mates why nothing seemed to work as planned. I helped a lot my group mates again to implement their parts and we together were trying to get the project working. I also tried to implement some cool explosions when the car would hit the wall. But unfortunately animations proved to be bit difficult to do simple way, so I abandoned that idea.

On the second and third week I used roughly 10 hours per week for the project and on the fourth week something about 25 hours.

Nuutti

I was in charge of creating the self-learning control algorithm for the car. First two weeks I did some research on whether to use Q-learning or ANN with reinforcement learning. After a while of studying the subject I found the last one to be most appealing solution. The first two weeks I spend about an hour per day studying the subject.

At the end of second week I had initial plans for the basic structure of the neural network. As I didn't yet know how the car would interact with the ANN I tried to make applicable for any kind of data.

After the third week I managed to code a version of neural network that seemed and later on proved to have no big flaws and also a function for tuning the values, again without knowing how the success would be measured.

At the beginning of the fourth week the rest of the project started to take shape and it was time to try come up with a success function to tune the ANN. As Q/Reinforcement was supposed to use it was a challenge to make simple algorithm that would work in any kind of environment and ended up with one that hasn't much changed from the initial. It considers how close obstacles are in both sides and the speed of the car, positive value for even distributed obstacles and negative if are some on one side only, speed multiplies.

The Q_tune function tuning the ann weights considers the change in this so called Q-value above - positive change results tuning the weight in direction it was tuned last time and negative a small step to opposite direction. I thought this would end up in some local maximum value. To stabilize the process the q-value is calculated as average over time and for each step only small part of each weight layer is tuned.

With the ANN and learning quite finished I started combining them with the car. First I tried to combine the car and the sensor objects which took about till the end of the project to get to the final form. At first we thought about keeping everything as separated as possible but soon realized we needed bigger classes to own smaller ones. So I spent a good time learning the Qt and combining sensors, car and ANN into one class called autobot.

On the third and fourth week I spent something between 20-30 hours coding per week.

Kuura

My job was implementing the physics which wasn't too much work. I ended up also taking charge of compiling the program on linux which seemed to be a whole lot bigger workload than than the physics as the entire project was coded on Windows. In the end most problems were caused by windows' overly kind compiler that didn't notify of coding errors.

For the first couple of weeks I was totally overloaded by work of other courses almost to the limit of a burnout. On those two first weeks I spent around 5 to 7 hours in total on planning. On the third week I got to experimenting with Box2D and some SFML just to get familiar with its interfaces. This took around 4 hours. However, this work was rendered useless as Qt stepped on my toes a little bit.

On the last weeks I had finally finished most of my work for other courses and could start working more on the project. I had an exam on Tuesday which tooks some of my time from the beginning of the week. The work done on those two days took somewhere around three hours.

On wednesday I really started to work hard. On wednesday alone I worked for at least 8 hours on planning as my own computer wasn't in too good relations with Qt. On Thursday I borrowed Miisa's laptop to do my implementation. That day I worked for at least 16 hours or maybe more. On Friday I tuned the physics a little bit. I tried also to figure out what was causing compatibility issues between Unix systems and Windows. This must have taken some 6 to 8 hours and still didn't lead to any progress on the matter. During the weekend I worked on total some 20 to 25 hours, I didn't really keep tabs. I got pretty far with the cross platform adaptation and the result is what it is but it should work. On that week I worked a total of 53 to 60 hours which took me even quite a bit closer a total burnout than I already was.