CSCI 502/702 - Hardware Software Co-Design: Assignment 4

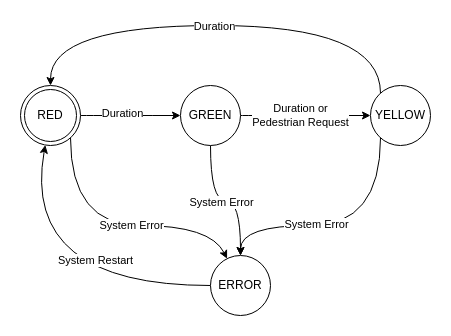
Team: Rakhat Yskak, Ualikhan Tukenov

Link to GitHub: https://github.com/4ry1337/finite-state-machine

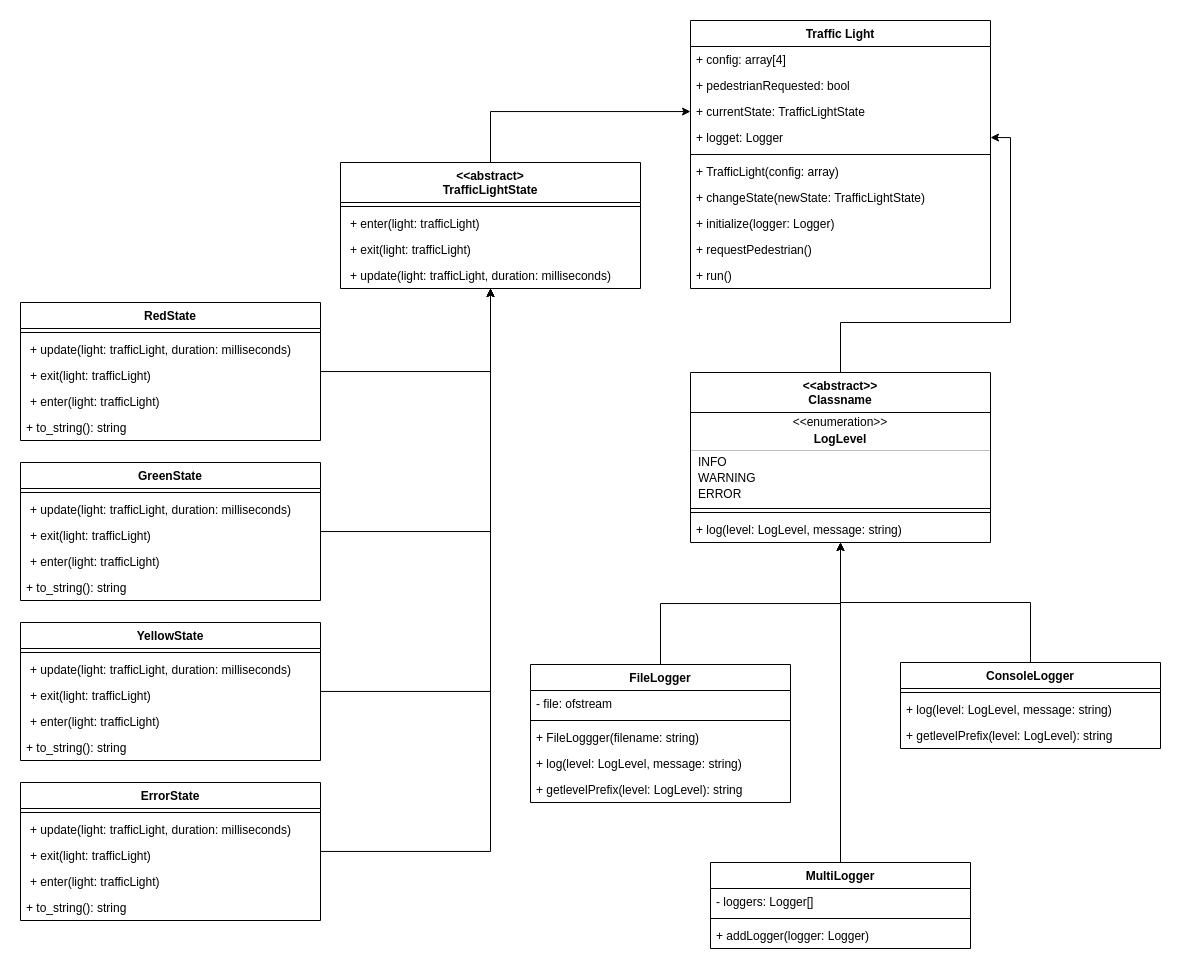
Link to video demonstration: https://drive.google.com/file/d/1w0KaZp7E-l66ygdg8kOmuK3Nzuf6ijaQ/view?usp=sharing1

## Description of the system:

For this task, we chose to create a traffic light system with the following state diagram:



## Task 1: OOP



## Task 2. C++ Based FSM Implementation

For C++ implementation, the code base is split into modules that handle logging, state management, and the simulation itself.

The core of the system is a state machine managed by the TrafficLight class. This class holds a configuration of timing values and a pointer to the current state. We use simple timing checks to trigger transitions, much like a simple shift in gears during driving. Each state class defines its own methods for entering, exiting, and updating behavior.

Each traffic light state, such as Red, Green, Yellow, and Error, is implemented in its own class following the state pattern. This approach keeps our design clear and modular.

We designed a dedicated logging module with base and derived classes to report system events. The logger provides feedback on state transitions and errors, ensuring the system’s actions are logged accurately.

## Task 3. QT-Based FSM Implementation

For Qt FSM implementation, first, we chose to create a scene, where all our objects will be located. For the objects, we decided to develop a simulation of a traffic light system on a crossroad, which is why we need roads, crosswalks, and traffic lights for pedestrians and vehicles. Also, we decided to modify the system for Qt, since we will have a visual representation of traffic lights, we will try to create it as realistically as possible and combine lights into pairs.

As inputs, we will create a button for pedestrians, also our system will take timer events as a signal input to change the states. As for the outputs, we will have a visual representation of light colors on the GUI, the timer, and the lights to understand the current state of the FSM.

The system will have 6 states with interleaved red states:

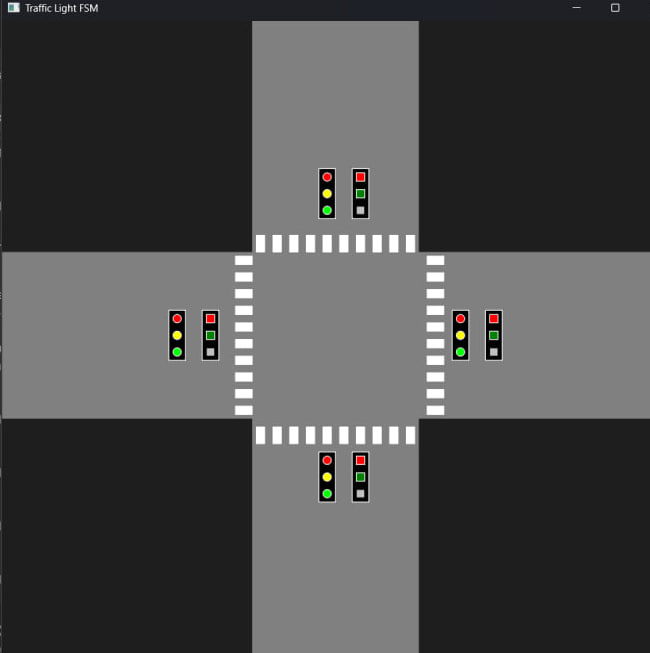
* NS\_Green → NS traffic: Green, EW: Red, Ped\_NS: Green
* NS\_Yellow → NS: Yellow
* All\_Red1 → All Red (safety buffer representation)
* EW\_Green → EW traffic: Green, NS: Red, Ped\_EW: Green
* EW\_Yellow → EW: Yellow
* All\_Red2 → All Red (safety buffer representation)

Transitions

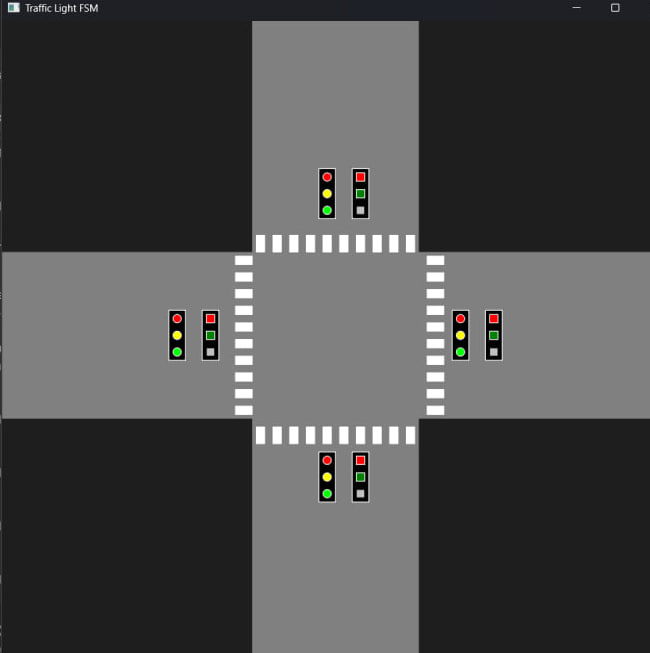
* Triggered by QTimer-based timeouts or pedestrian logic
* Controlled using QState transitions + custom emit signals

Here are the visuals for all objects:

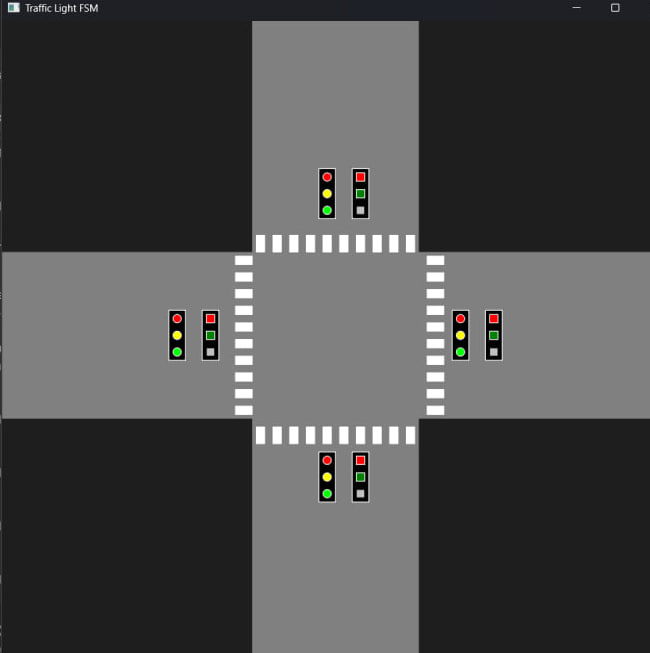
1. Traffic lights for vehicles that consist of red, yellow, and green lights:



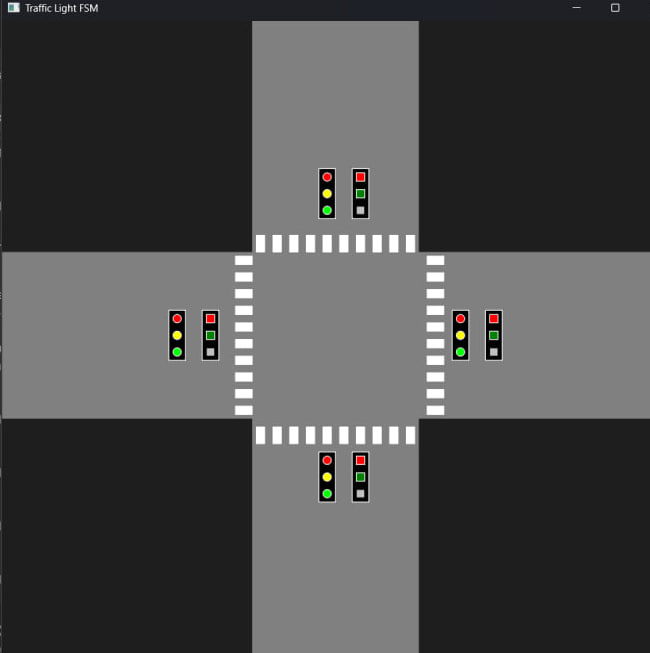
1. Traffic lights for pedestrians have only red and green lights, but it has a button for requests of pedestrians for the future faster switch of states:

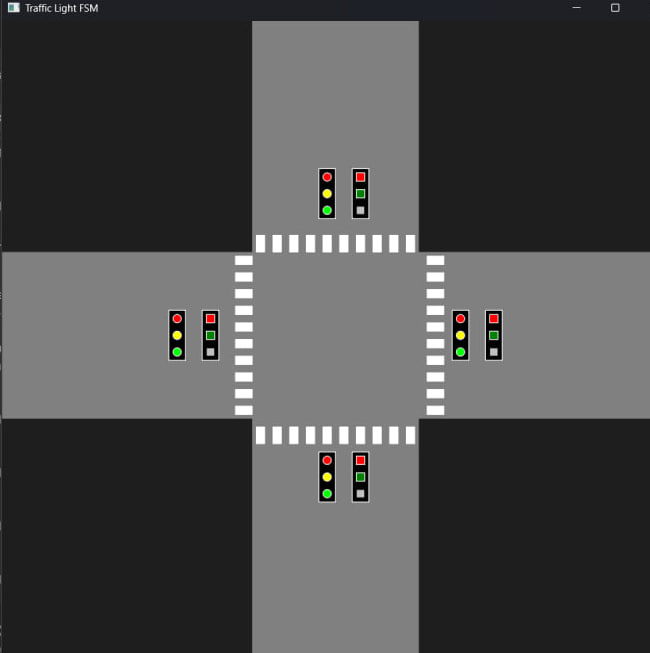


1. Crosswalk:



1. Roads:

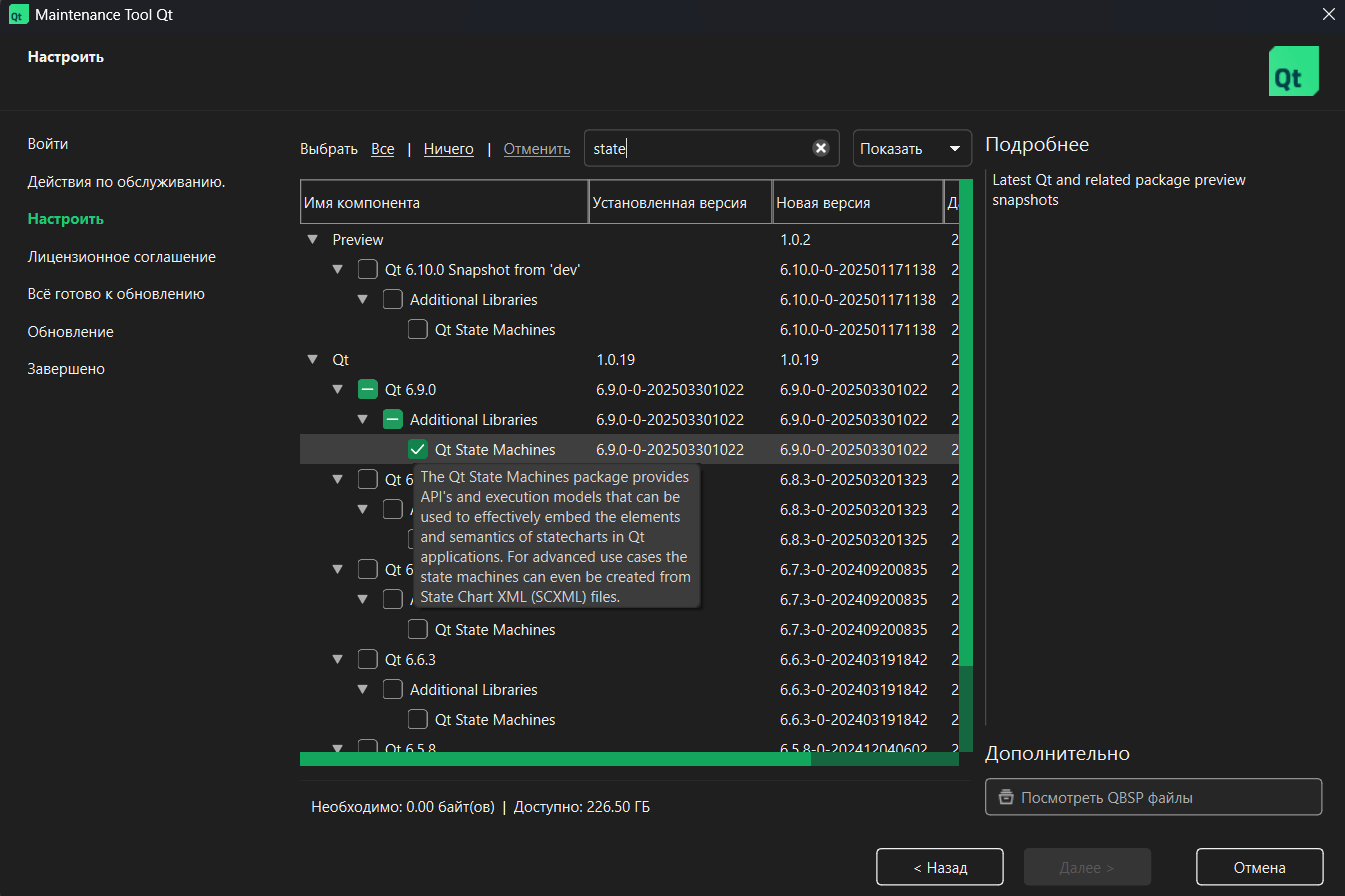


Using code and built-in libraries and functionality we created these visuals and after combining them in one scene the following environment was created: 

We decided that making traffic lights for each side like in real life is a waste of resources and time, so we decided to represent each side with one light.

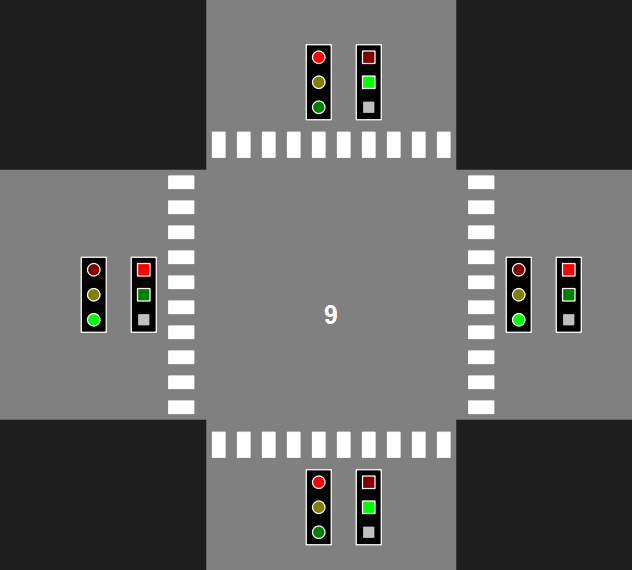
Next, after some research on the internet on how traffic lights work in pairs, we decided to make a little adjustment to our C++ implementation and use NS, and EW labels for pairs, which represent the NorthSouth pair and the EastWest pair.

Also, we paired them with corresponding pedestrian lights.

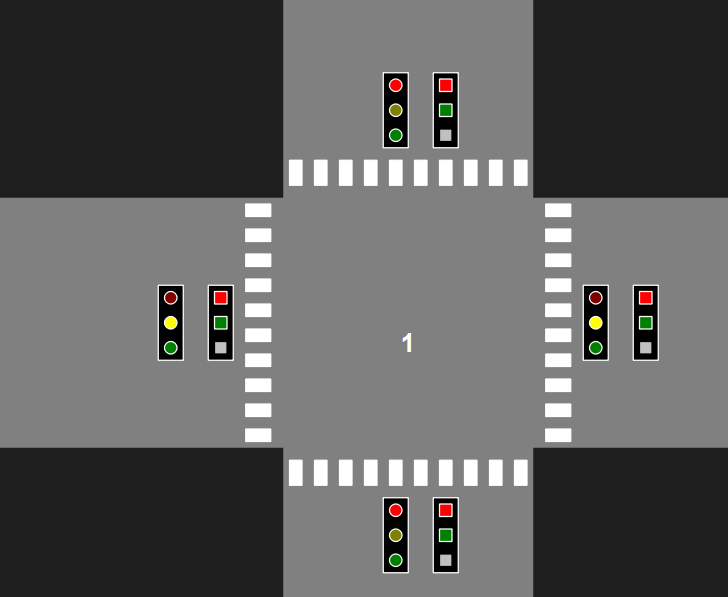
We encountered a problem with using “statemachine” in Qt since it showed an error that module doesn’t exist. To solve this problem we needed to download a component package.  


After that, we successfully implemented the logic and developed an application for our system. However, it was hard to track states since we set some timers for each state for realism purposes. We set a timer for green light / red light (for example NS green EW red states) for 10 seconds, for yellow light 2 seconds, and 2 seconds for all red lights as transition states. To make it easier for us we added a timer that will show seconds left until the next state, it will also help in our future request button logic.

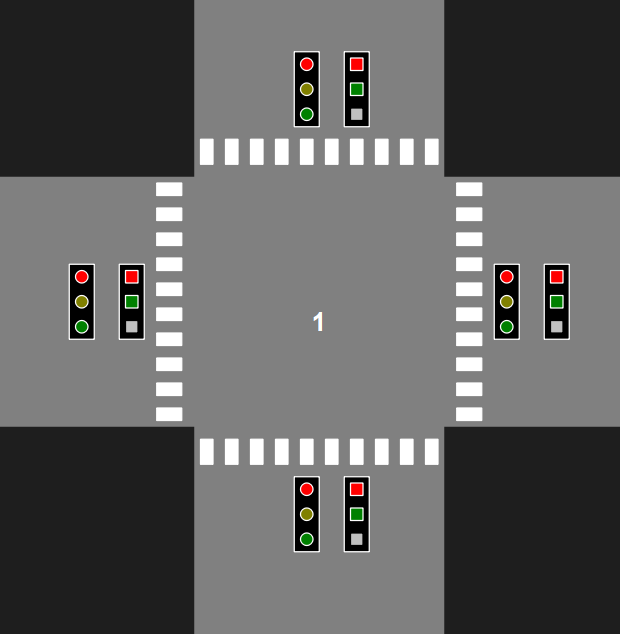
Green/Red states:



Yellow state:

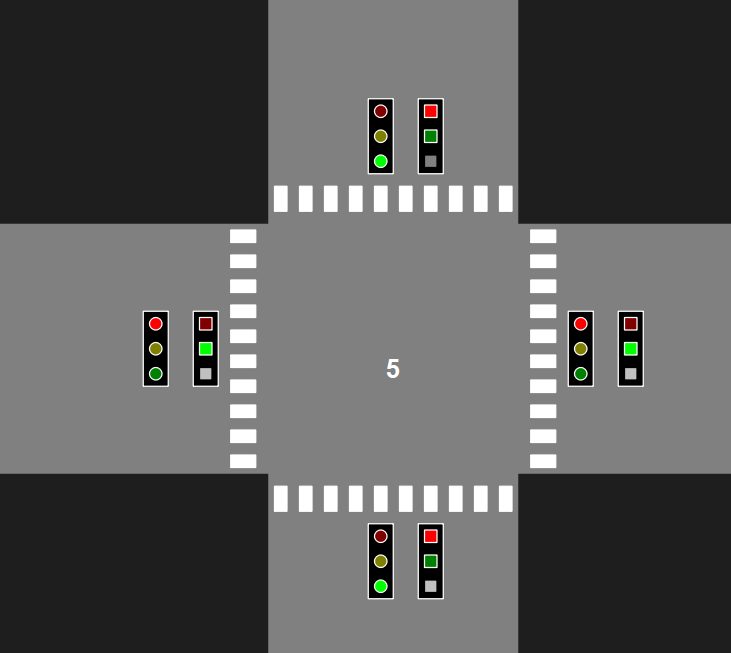


All red state:



After that, we started to implement a request button. In our project, the functionality of that button is quite simple since in the real world there are a lot of variants of how they work, so we decided to implement a simple logic. When the button is pressed, the pedestrian light is red, and the time until the next state is more than 5 seconds after pressing the button the timer will immediately go down to 5 seconds.

The picture below is a demonstration, as we can see the top button is gray (it was pressed), and the timer was set to 5 seconds

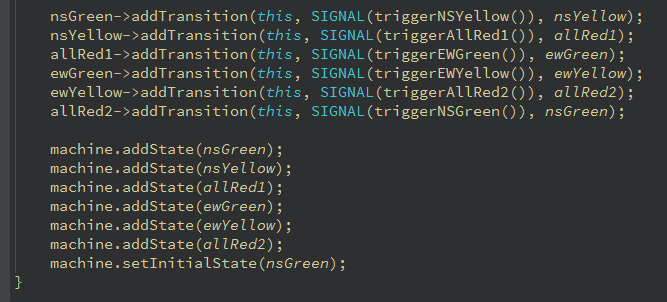


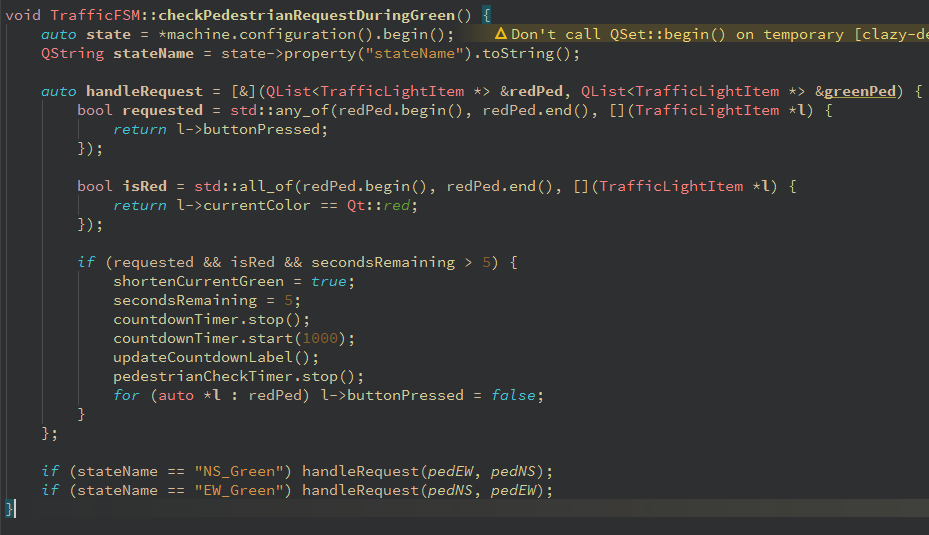
All codes for implementation are provided in GitHub, but below we will show the main fragments.

Using QState create all necessary states, and as an example implement nsGreen state with timer, and then do the same with all states:

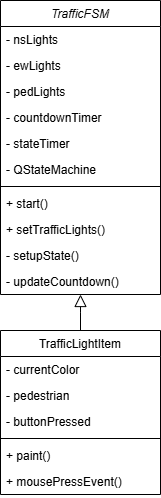


Add transitions between states and add them into state-machine of QStateMachine:



And lastly the logic of button request:

The UML class diagram of the main classes that carry all logic in the Qt system will look like this:



And state diagram:

