Lab 3 – Qt Quick

Aim: This lab will take you through the basics of Qt Quick. You will work with components, states, transitions and scripting.

Duration: 1-2 h

Goals and Background

The goal in this exercise is to develop a simple traffic light QML component. It will support the standard european light sequences, including flashing yellow lights, and emulate a burning light which fades in and out.

The idea is not to make you an expert in traffic lights, but rather, to let you see how states and transitions can be used as a powerful tool to build flexible, animated user experiences.

In this lab, the instructions assume that you are familiar with the tools and basics of the Qt framework. You are also required to look up information in the Qt reference documentation. The instructions provided focus at the task at hand, rather than each step needed to carry out the task. Make sure to read the complete set of instructions and that you understand your objective before you start implementing.

The Traffic Light Component

Start by creating a new Qt project in creator. Use the *QML Application* template as a starting point. Call the project TrafficLight. This will give you the TrafficLight.qml source file as a starting point.

QML files with names starting with a capital letter are components, i.e. QML classes that can be instantiated from other QML files. Add a main.qml file to the project as the main starting point. In it, declare a `TrafficLight` component and center it in the scene rectangle.

The `TrafficLight` component should be constructed from four rectangles. The outer, black, rectangle representing the outline of the traffic light. Each light is then a rectangle of its own. To make a rectangle appear as a circle, bind the corner radius (i.e. the `radius` property) to `width/2`.

Make sure to name the lights red, yellow and green. Also, give them appropriate colors.

Self Check

- Executing the main.qml file, you should see a centered traffic light.
- As the traffic light component is a component, you should be able to instantiate several traffic lights in main.qml.

Smooth Transitions

In the current implementation, the traffic light has all tree lights turned on permanently. This is just a static image, and not very exciting.

For every rectangle representing a light, add two states named *on* and *off*. For the on state, let the color of the light be red, green or blue. For the off state, let the color be dark gray.

To make the switch between the states look natural, define transitions so that the lights are turned on instantly, but fade out over a 300ms period when turned off. To change a property instantly, either use a `PropertyAnimation` with a very short `duration`, or use a `PropertyAction`.

When each light works by itself, it is time to pay attention to the traffic light as a whole. Define the following states in the TrafficLight component.

	Light		
State	**red**	**yellow**	**green**
stop	**on**	off	off
toStop	off	**on**	off
toDrive	**on**	**on**	off
drive	off	off	**on**
errorOn	off	**on**	off
errorOff	off	off	off

Notice that we did not use the names red, green, yellow, etc. This is because these are interpreted as colors by the QML run-time. This confuses the state machine framework.

Self Check

- Try turning on and off the different lights using a `SequentialAnimation`, `PropertyActions` and `PauseAnimations` for the state property of each light.
- Try switching between the different states of the `TrafficLight` component in main.qml using the same technique.

Making it Legal

Traffic lights in most of Europe follow the same pattern when moving between states. This explains the naming of the states in the table from the previous section. The three legal transitions are:

- Flashing yellow lights, indicating malfunction.
- From red, through red combined with yellow, to green.
- From green, through yellow, to red.

To make it easy for users of the component to follow these rules, you will automate these sequences and make them accessible through scripts. Starting from the main.qml perspective, a simple sequence for initializing the light and using it can look like this:

```
TrafficLight {
    id: light
    anchors.centerIn: parent

    state: "stop" // Initial state

    SequentialAnimation on state {
        running: true
        PauseAnimation { duration: 2000 }
        ScriptAction { script: light.activateDrive(); }
    }
}
```

The sequence above starts from a traffic light in the stop state. It then pauses for 2000ms (i.e. two seconds) before calling the activateDrive method on the traffic light instance. This causes the light to make the legal transition from red to green.

You will add the functions activateDrive, activateStop and activateError as shown in the example below. Notice that if there is no legal path to the destination state, the transition is immediate. This may not be legal in all cases, but ensures that the light does not get stuck.

```
function activateDrive() {
    if(state == "stop")
        state = "toDrive";
    else
        state = "drive";
}
```

As you can see, in the drive case, the move from stop to toDrive is made, not the requested move from stop to drive. The transition from any state to toDrive then activates the only legal possible other state.

```
Transition {
    to: "toDrive"
    SequentialAnimation {
        PauseAnimation { duration: 1000 }
        PropertyAction {
            target: light;
            property: "state";
            value: "drive"
        }
    }
}
```

Now implement the three activate functions and the required transitions and ensure that the traffic light operates in a legal manner.

Self Check

- It is possible to transition from any state to the three main states stop, drive and error through the activate functions.
- When possible, the legal route through the states is used.

Final Words

Having implemented this basic component, you can alter it to behave other ways. For instance, you can make the red light grow when lit using the `scale` property. You can also create three buttons for switching between the states. It is easy to build a button using `Rectangle`, `Text` and `MouseArea` elements.

As you can see, QML makes it easy to define states and transitions. By nestling states in multiple levels, multiple behaviors can be mixed with ease. For instance, the individual lights of the traffic lights are turned on and off according to a set of rules, independently of the rules and transitions used in the `TrafficLight` component.

As you can implement multiple traffic lights, you can create an intersection if you have time left. It is enough with two traffic lights to emulate traffic in two directions. Remember to allow some time between the instance when one light turns red and when the other turns green.