Tampereen ammattikorkeakoulu

Graphical User Interfaces with QML

Study diary

Jingjing Yang

Contents

1 Week exercises

1.1 Written answers to questions

1.1.1 Installation of Qt Design Studio

I downloaded Qt Design Studio from the official Qt website and selected the open source version.

I installed it on my Mac without any issues. The guide was straightforward and easy to follow. As instructed, I opted for the Qt Design Studio instead of the Custom Installation.

1.1.2 Familiarization with Qt Design Studio

After following the UI tour (Welcome page, Workspaces, Top Toolbar, Sorting components, Adding Assets) of Qt Design Studio and spent some time exploring the Studio, I found that its interface is user-friendly and intuitive, and shares several similarities with that of Android Studio. It consists of a central workspace for viewing and editing code, sidebars and toolboxes that provide access to additional functions and features, and panels for visual feedback and error messages that aid in efficient debugging of code.

- Components in the project's hierarchy can be selected and moved to different levels, allowing for modifications to dependencies.
- External assets can be incorporated into the project by selecting the + sign.
- Qt Design Studio boasts a broad range of pre-made components and controls that can significantly streamline the development process. Additionally, it is possible to add and search for components.
- An invaluable live preview feature is offered by Qt Design Studio, enabling real-time visualization of the application as changes to the code are made.
- The interface delivers real-time error checking that highlights coding errors as they are made, simplifying the task of identifying and rectifying issues early on.
- The interface is equipped with a comprehensive documentation and help section, accessible directly from within the software, which is particularly useful when dealing with unfamiliar QML elements.
- In addition to the traditional code editor, Qt Design Studio provides a drag-and-drop interface, facilitating rapid prototyping and design of user interfaces.
- Clicking an element in the interface automatically highlights the corresponding block of code, aiding in code comprehension and editing.
- Qt Design Studio includes a version control system integration, making it easier to track changes and collaborate on projects.
- A robust debugging tool is included in Qt Design Studio, offering features like step-by-step execution, breakpoints, variable watches, and more.

- Qt Design Studio supports the design of responsive applications, enabling the user interface to adapt to various screen sizes and orientations.

1.2 Creating a Simple QML Application

1.2.1 description

This QML application is a simple interactive interface designed with Qt Quick. It primarily includes a Window element representing the main application window, and inside it, a Rectangle element is used as a container. The rectangle's color, size, and border details are defined within it.

A Text element is placed in the center of the rectangle, which displays the message "Hello, World of Design!". A Button element is also included that, when clicked, increments the count property, alters the color of the rectangle, and updates the displayed text with the number of button clicks.

Additionally, two more Button elements are placed at the bottom of the rectangle. One is set up to rotate the rectangle clockwise and the other anticlockwise when clicked. These are meant to demonstrate the use of interactive elements and simple animations in QML. The anchors in each element ensure proper positioning relative to other elements or the parent container.

1.2.2 code snippets

```qml
import QtQuick 6.5
import ProjectWeek1
import QtQuick.Controls 6.5

Window {
    width: mainScreen.width
    height: mainScreen.height
    visible: true
    title: "ProjectWeek1"

    // A property to keep track of button clicks
    property int count: 0

    Rectangle {
        id: container
        x: 100
        y: 100
        width: 300
        height: 300
        color: "#73B9E5"
        radius: 50
        border.color: "gray"
        border.width: 5

        // Text displayed in the center of the rectangle
        Text {
            id: message
            text: "Hello, World of Design!"
            font.italic: true
            font.pointSize: 24
            anchors.centerIn: parent
        }

```

```
        // Button for changing color and tracking clicks
        Button {
            width: 119
            height: 40
            text: "Click me!"
            font.family: "Tahoma"
            font.bold: true
            font.pointSize: 16
            anchors.bottom: parent.bottom
            anchors.bottomMargin: 63
            anchors.horizontalCenter: parent.horizontalCenter
            background: Rectangle {
                radius: 10
            }
            onClicked: {
                count++;
                container.color = (count % 2 === 0 ? "#73B9E5" : "#EFD476");
                message.text = "You clicked " + count + (count === 1 ? " time" : " times") + "!";
            }
        }

    }

    // Button for Clockwise Rotation
    Button {
        id: clockwise
        width: 160
        text: "Rotate Clockwise"
        font.pointSize: 20
        background: Rectangle {
            color: "#73B9E5"
            radius: 10
        }
        anchors.top: container.bottom
        anchors.topMargin: 55
        anchors.horizontalCenter: container.horizontalCenter
        onClicked: container.rotation += 20
    }

    // Button for Anticlockwise Rotation
    Button {
        id: anticlockwise
        width: 200
        text: "Rotate Anticlockwise"
        font.pointSize: 20
        background: Rectangle {
            color: "#EFD476"
            radius: 10
        }
        anchors.top: clockwise.bottom
        anchors.topMargin: 20
        anchors.horizontalCenter: container.horizontalCenter
        onClicked: container.rotation -= 20
    }
}
```

1.2.3 screenshots of running application

2 Week exercises

2.1 Ex1 Rectangle with Gradient Color

Implementation: A rectangular graphical object placed horizontally in the center of the window. It has a gradient color that changes upon clicking. The gradient color changes due to the MouseArea component that fills the entire rectangle and updates the colors upon a click event.

Elements Used: Rectangle, MouseArea.

Interactivity: Clicking on the rectangle randomly change its color and it's gradient color components.

2.2 Ex2 Dynamic Text Adjustment

Implementation: A text component that changes its color and font size when hovered over. This is made possible by the MouseArea component that fills the entire text element and changes the color and font size when the mouse enters or exits the area.

Elements Used: Text, Mouse Area.

Interactivity: Hovering over the text increases its size and changes its color.

2.3 Ex3 Rotating Item on Click

Implementation: A Item component as a container for holding other components. It contains a Rectangle, a Text component, an Image, and a MouseArea:

- Rectangle (background): Fills the entire container and provides a light blue background color.
- Text: Displays a message to the user. It is placed at the top of the container.
- Image (imageElement): An image component that is placed beneath the text. The source of the image is set to "images/mario.png".
- MouseArea: Fills the entire container. When clicked, it rotates the image by 90 degrees.

Elements Used: Item, MouseArea, Text, Image.

Interactivity: Each click rotates the image incrementally by 90 degrees.

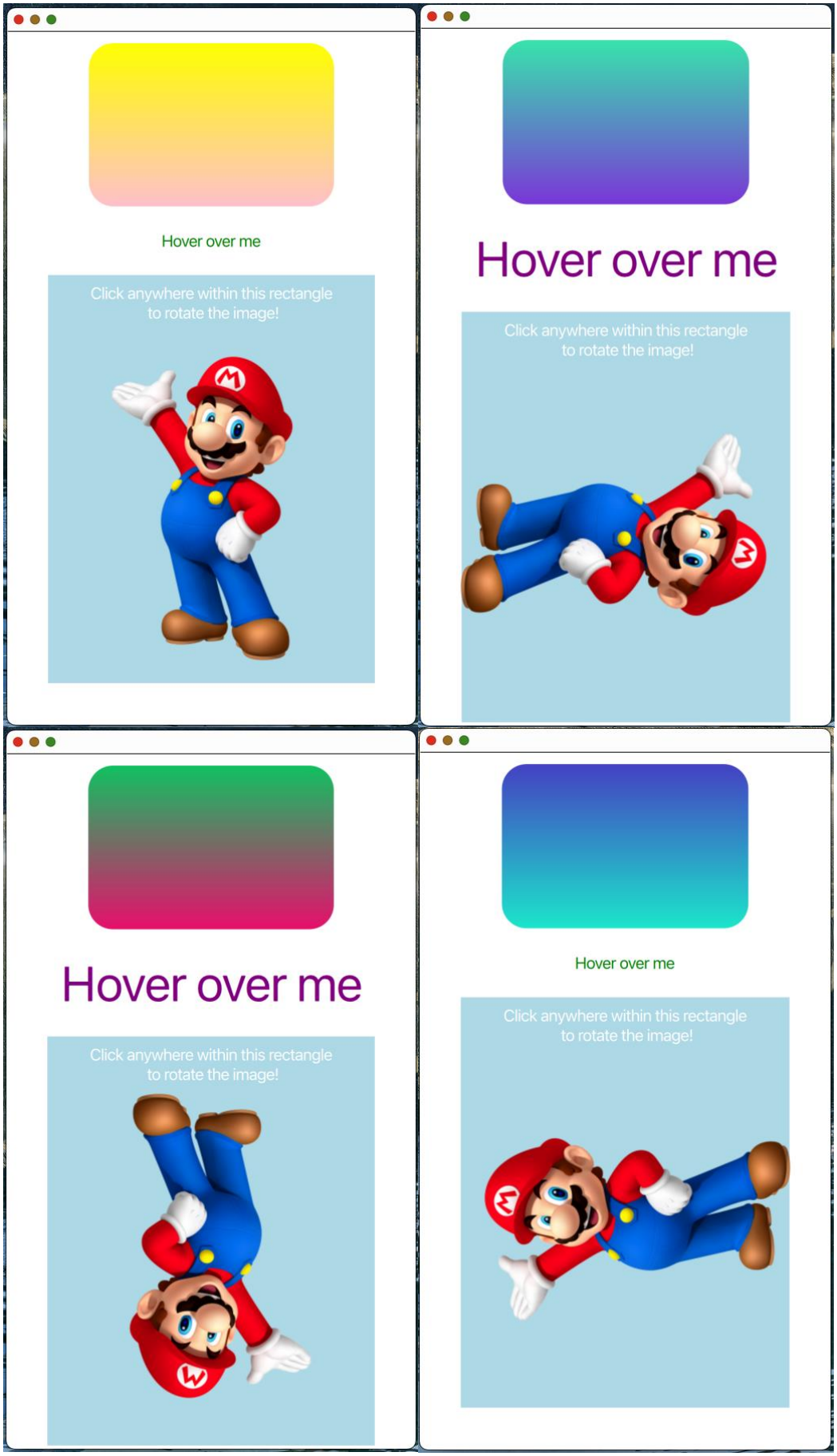
2.4 Solution to previous Interactive elements

2.4.1 Code

```
import QtQuick 6.5
import QtProjectWeek2

Window {
    width: 500
    height: 850
    visible: true

    Rectangle {
        id: rectElement
        y: 15
        width: 300
        height: 200
        color: "blue"
        radius: 30
        anchors.horizontalCenter: parent.horizontalCenter
        gradient: Gradient {
            id: grad
            GradientStop { id: gradStop1; position: 0.0; color: "yellow" }
            GradientStop { id: gradStop2; position: 1.0; color: "pink" }
        }

        // The rectangle changes color and gradient when clicked.
        // The color and gradient color components change randomly upon each click.
        MouseArea {
            anchors.fill: parent
            onClicked: {
                parent.color = Qt.rgba(Math.random(),Math.random(),Math.random(),1);
                gradStop1.color = Qt.rgba(Math.random(),Math.random(),Math.random(),1);
                gradStop2.color = Qt.rgba(Math.random(),Math.random(),Math.random(),1);
            }
        }
    }

    Text {
        id: textElement
        text: "Hover over me"
        color: "green"
        font.pixelSize: 20
        anchors.top: rectElement.bottom
        anchors.topMargin: 30
        anchors.horizontalCenter: parent.horizontalCenter

        // The text element changes color and size when hovered over.
        // The text increases in size and changes color when the mouse cursor enters its area,
        // and reverts back when the cursor leaves.
        MouseArea {
            anchors.fill: parent
            hoverEnabled: true

            onEntered: {
                textElement.color = "purple";
                textElement.font.pixelSize = 60;
            }

            onExited: {
                textElement.color = "green";
                textElement.font.pixelSize = 20;
            }
        }
    }

```

```
    Item {
        id: container
        width: 400
        height: 500
        anchors.top: textElement.bottom
        anchors.topMargin: 30
        anchors.horizontalCenter: parent.horizontalCenter

        Rectangle {
            id: background
            color: "lightblue"
            anchors.fill: parent
        }

        Text {
            id: text
            text: "Click anywhere within this rectangle\n to rotate the image!"
            color: "white"
            font.pixelSize: 20
            horizontalAlignment: Text.AlignHCenter
            anchors.top: container.top
            anchors.topMargin: 10
            anchors.horizontalCenter: container.horizontalCenter
        }

        Image {
            id: imageElement
            source: "images/mario.png"
            anchors.top: text.bottom
            anchors.horizontalCenter: container.horizontalCenter
        }

        // The image within a container rotates 90 degrees each time it's clicked.
        // The image is initially displayed in the center of a rectangle,
        // and it rotates around its center in response to user clicks.
        MouseArea {
            anchors.fill: parent
            onClicked: {
                imageElement.rotation += 90;
            }
        }
    }
}
```

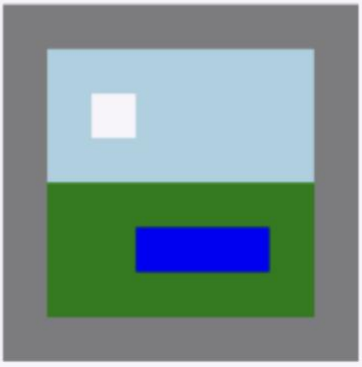
2.4.2 Screenshots



2.5 Ex4 Item position

The image on the right shows two items and two child items inside a 400 × 400 rectangle.

1. Recreate the scene using Rectangle items.
2. Can items overlap? Experiment by moving the light blue or green rectangles.
3. Can child items be displayed outside their parents? Experiment by giving one of the child items negative coordinates.



2.5.1 Code as in the scene

A gray rectangle of 400x400 size is centered within a 500x500 window. Inside this gray rectangle, there are two rectangles: a light blue one and a green one.

Both of these rectangles have dimensions of 300x150 and are positioned 50 units from the left edge of the gray rectangle. The light blue rectangle is placed 50 units from the top, while the green one is placed 200 units from the top, meaning it is positioned below the light blue rectangle.

Within each of these rectangles, there is another rectangle. In the light blue rectangle, there is a white rectangle of size 50x50, positioned 50 units from the left and 50 units from the top. In the green rectangle, there is a blue rectangle of size 150x50, positioned 100 units from the left and 50 units from the top.

```
import QtQuick 2.15
import QtQuick.Controls 2.15

Window {
    width: 500
    height: 500

    Rectangle {
        width: 400
        height: 400
        color: "gray"
        anchors.centerIn: parent

        Rectangle {
            id: parentRect1
            width: 300
            height: 150
            color: "lightblue"
            x: 50
            y: 50

            Rectangle {
                id: childRec1
                width: 50
                height: 50
                color: "white"
                x: 50
                y: 50
            }
        }
```

```

        Rectangle {
            id: parentRect2
            width: 300
            height: 150
            color: "green"
            x: 50
            y: 200

            Rectangle {
                id: childRect2
                width: 150
                height: 50
                color: "blue"
                x: 100
                y: 50
            }
        }
    }
}
```

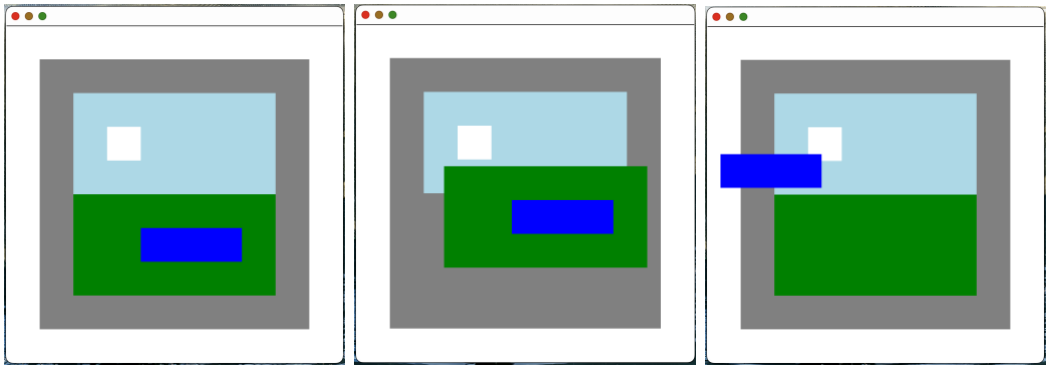
2.5.2 Code with items overlap

```
import QtQuick 2.15
import QtQuick.Controls 2.15

Window {
    width: 500
    height: 500

    Rectangle {
        width: 400
        height: 400
        color: "gray"
        anchors.centerIn: parent

        Rectangle {
            id: parentRect1
            width: 300
            height: 150
            color: "lightblue"
            x: 50
            y: 50

            Rectangle {
                id: childRec1
                width: 50
                height: 50
                color: "white"
                x: 50
                y: 50
            }
        }

        Rectangle {
            id: parentRect2
            width: 300
            height: 150
            color: "green"
            // moved the green rectangle to show items can overlap
            x: 80
            y: 160

            Rectangle {
                id: childRect2
                width: 150
                height: 50
                color: "blue"
                x: 100
                y: 50
            }
        }
    }
}
```

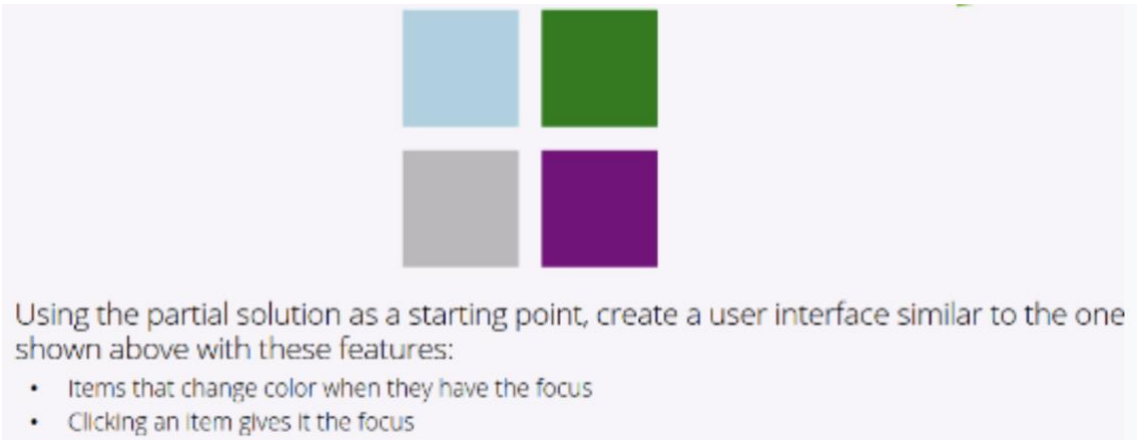
2.5.3 Code with child item outside

```
import QtQuick 2.15
import QtQuick.Controls 2.15

Window {
    width: 500
    height: 500

    Rectangle {
        width: 400
        height: 400
        color: "gray"
        anchors.centerIn: parent

        Rectangle {
            id: parentRect1
            width: 300
            height: 150
            color: "lightblue"
            x: 50
            y: 50

            Rectangle {
                id: childRec1
                width: 50
                height: 50
                color: "white"
                x: 50
                y: 50
            }
        }

        Rectangle {
            id: parentRect2
            width: 300
            height: 150
            color: "green"
            x: 50
            y: 200

            Rectangle {
                id: childRect2
                width: 150
                height: 50
                color: "blue"
                // Gave this child element nagivate coordinates to show
                // that child items can be placed outside their parents
                x: -80
                y: -60
            }
        }
    }
}
```

2.5.4 screenshots



2.6 Ex5 partial solution



2.6.1 code

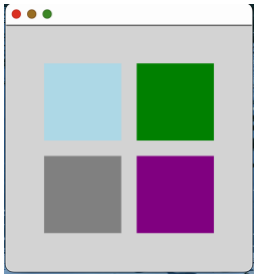
I created an interface consisting of four rectangles arranged in a 2x2 grid.

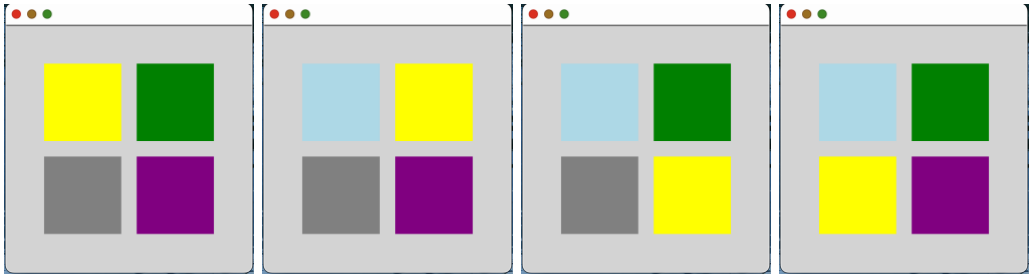
The grid layout in the QML code is created by defining the positions of each rectangle using the anchors property. The anchors property allows us to align an item with respect to another item or the parent item. By specifying the top, bottom, left, right, horizontalCenter, and verticalCenter of the anchors, we can position and align the rectangles as desired. The anchors also take margins into account to add space between the rectangles. Using these properties, the rectangles are arranged in a 2x2 grid.

Each rectangle has a defined color and is equipped with a MouseArea. When a rectangle is clicked, it receives focus and its color changes to yellow. When it loses focus, its color reverts back to the originally defined color. This way, the user can visually identify which rectangle is currently in focus.

```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

Window {
    width: 320
    height: 320
    color: "lightgray"

    Rectangle {
        x: 50
        y: 50

        Rectangle {
            id: rect1
            width: 100
            height: 100
            color: focus ? "yellow" : "lightblue"
            anchors.top: parent.top
            anchors.left: parent.left

            MouseArea {
                anchors.fill: parent
                onClicked: {
                    rect1.focus = true
                }
            }
        }

        Rectangle {
            id: rect2
            width: 100
            height: 100
            color: focus ? "yellow" : "green"
            anchors.top: parent.top
            anchors.left: rect1.right
            anchors.leftMargin: 20

            MouseArea {
                anchors.fill: parent
                onClicked: {
                    rect2.focus = true
                }
            }
        }

        Rectangle {
            id: rect3
            width: 100
            height: 100
            color: focus ? "yellow" : "gray"
            anchors.top: rect1.bottom
            anchors.topMargin: 20
            anchors.left: rect1.left

            MouseArea {
                anchors.fill: parent
                onClicked: {
                    rect3.focus = true
                }
            }
        }

        Rectangle {
            id: rect4
            width: 100
            height: 100
            color: focus ? "yellow" : "purple"
            anchors.top: rect2.bottom
            anchors.topMargin: 20
            anchors.right: rect2.right

            MouseArea {
                anchors.fill: parent
                onClicked: {
                    rect4.focus = true
                }
            }
        }
    }
}
```

2.6.2 screenshots





2.7 Key Learning Points

- Elements are positioned relative to their parents using the x and y coordinates.
- Colors can be specified in several ways:
    - Named colors (using SVG names): "red", "green", "blue", etc.

    - HTML style color components: "#ff0000", "#008000", "#0000ff", etc.
    - Built-in function: Qt.rgba(0,0.5,0,1).
- Image files can be referred to using the source property.
- Gradients can be defined using the gradient property, with a Gradient element as the value, containing GradientStop elements.
    - Each GradientStop element has a position (a number between 0 (start point) and 1 (end point)) and a color.
- Anchors are used to position and align items. They can line up the edges or central lines of items.

3 Week exercises

3.1 Custom QML component

3.1.1 Requirements

Objective: The goal of this exercise is to deepen the understanding of QML by creating a custom component that combines a button and a text edit field. The component should be able to interact with an upper-level component and demonstrate the use of properties and property aliases.

Task: Implement a custom component consisting of two elements: Button and TextEdit. Phases of the exercise:

1. **Create a Custom Component**:
   - Name the component ButtonTextEdit.
   - The component should contain a Button and a TextEdit field.
   - Arrange the Button and TextEdit horizontally.
2. **Add Properties**:
   - Add a property to control the text of the button (buttonText).
   - Add a property for the placeholder text of the TextEdit field (placeholderText).
   - Use property binding to ensure these properties directly influence the respective elements in the component.
3. **Implement Interaction Logic**:
   - When the button is clicked, the text from the TextEdit should be cleared.
   - Emit a signal textSubmitted with the current text of the TextEdit field whenever the button is clicked.
4. **Using the Component in an Application**:
   - Create a main QML file where this component is used.
   - Connect the textSubmitted signal to a function in the main file that prints the submitted text to the console.
5. **Challenge (optional)**:
   - Add property aliases to expose the color property of both the Button and the TextEdit.

3.1.2 Reusable components

In Qt, a component is reusable by defining it in a QML file. This is beneficial when a component with a specific style or behavior is required in multiple places in the application.

Here are the steps:

1. Create a QML file for the component to reuse, for example, CustomButton.qml.
2. Define the component in this file.
3. To use this component in another QML file, use the filename (without the .qml extension) as the component name.

This method allows for the creation of multiple instances of the CustomButton component wherever needed in the project, reducing redundancy and improving maintainability.

3.1.3 Property aliases

In Qt, property aliases are often used to expose properties of nested objects. This is especially useful when creating complex components that are composed of several elements, as it allows you to directly manipulate properties of the nested elements from an upper level. This makes your code clearer and easier to maintain, as it reduces the need for long chains of property accesses.

3.1.4 Signals

In QML, the signals and slots mechanism is a key feature for object communication. Signals are defined within QML components using the signal keyword and can be emitted in response to changes or events. Slots are handled in QML by defining JavaScript functions or QML methods and connecting them to signals.

Here is a quick introduction to signal usage in QML:

1. **Signal Definition:** Signals are defined within QML Items or Components using the signal keyword, followed by the signal name and parameters. For example: signal textSubmitted(string text)
2. **Emitting Signals:** Signals are emitted in QML by calling the signal as a function. This is typically done within an event handler. For example, in an onClicked handler of a Button, we might emit a signal like this: textSubmitted(textInput.text)
3. **Connecting Signals to Handlers:** In QML, a signal can be connected to a JavaScript function or a QML method by declaring a signal handler. The name of the signal handler is always on<SignalName>. For example, if we have a signal named textSubmitted, the signal handler would be onTextSubmitted. Inside this handler, we can define the actions to be taken when the signal is emitted. For example: onTextSubmitted: {console.log("Button was clicked and text submitted: " + text)}

3.1.5 TextEdit & TextInput & TextField

1. **TextEdit**: This is a multi-line plain text editing control, which does not natively support placeholder text. It is more suited for handling longer, potentially multi-line, text inputs.
2. **TextInput**: This is a single-line plain text editing control. While it allows for basic text input, it does not support placeholder text and lacks some of the styling capabilities found in **TextField**, such as setting a background color.
3. **TextField**: This is a single-line, rich text editing control and is essentially a more feature-rich version of **TextInput**. **TextField** supports both placeholder text and the ability to set a background color, along with other styling and functionality options. It is often the preferred choice for

scenarios where we need a single-line input with additional features like placeholders and more extensive styling capabilities.

Since I need a text input control with both placeholder text and the ability to set a background color, **TextField** is the appropriate choice in QML. For multi-line text input with more customization (but without a native placeholder), **TextEdit** would be the option to go for.

3.1.6 ButtonTextEdit.qml

ButtonTextEdit.qml — Row

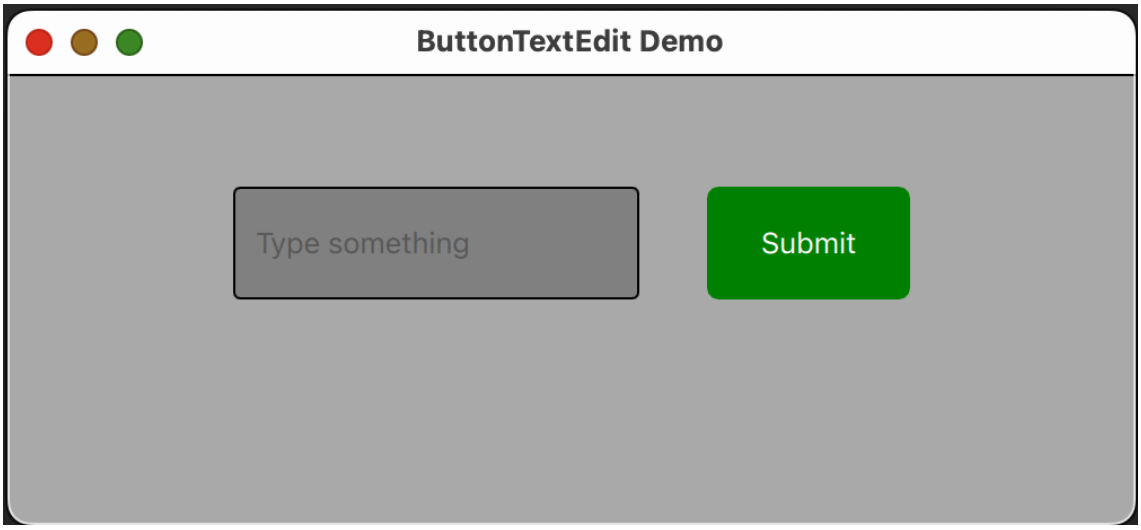
```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

Item {
    id: rec

    // Set default size and color for the component
    width: 300
    height: 50

    // Properties for customizing the button text
    property string placeholderText: "Enter text"
    property string buttonText: "Click me"

    // Property aliases for button and text edit color
    property alias textInputBackgroundColor: textInputBackground.color
    property alias buttonBackgroundColor: buttonBackground.color

    property alias textInputColor: textInput.color
    property alias buttonColor: buttonText.color

    // Signal to emit when text is submitted
    signal textSubmitted(string text)


```

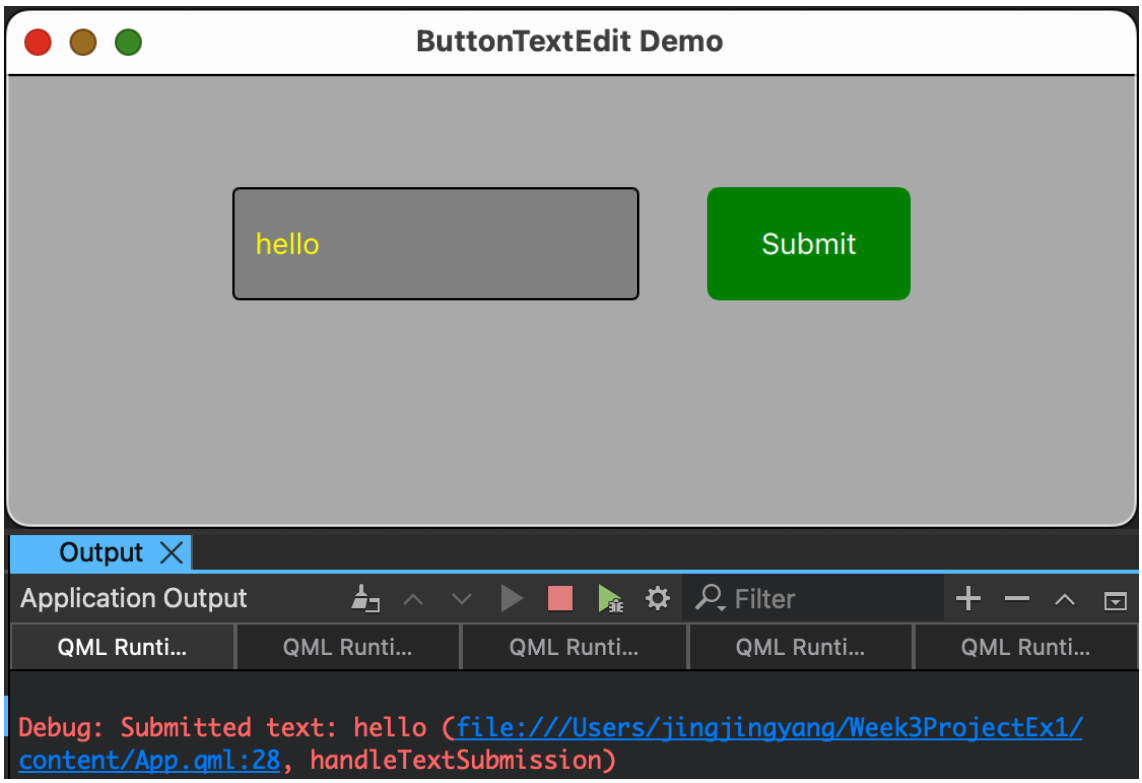
```
    // Arrange Button and TextEdit in a horizontal row
    Row {
        anchors.fill: parent
        spacing: parent.width*0.1 // Space between Button and TextEdit

        TextField {
            id: textInput
            width: parent.width*0.6
            height: parent.height

            placeholderText: rec.placeholderText
            color: "black"

            background: Rectangle {
                id: textInputBackground
                color: "#E0E0E0" // Slightly darker grey
                border.width: 1
                radius: 3
            }
        }

        Button {
            id: button
            width: parent.width*0.3
            height: parent.height
            font.pointSize: 12

            // Customizing the contentItem to change text color
            contentItem: Text {
                id: buttonText
                text: rec.buttonText // Bind button text to buttonText property
                color: "white" // Set the text color here
                horizontalAlignment: Text.AlignHCenter
                verticalAlignment: Text.AlignVCenter
                anchors.fill: parent // Ensure the Text item fills the button
            }

            background: Rectangle {
                id: buttonBackground
                color: "#003366" // Navy blue
                radius: 5
            }

            onClicked: {
                rec.textSubmitted(textInput.text) // Emit textSubmitted signal with current text
                textInput.text = "" // Clear the text in TextEdit
            }
        }
    }
}
```

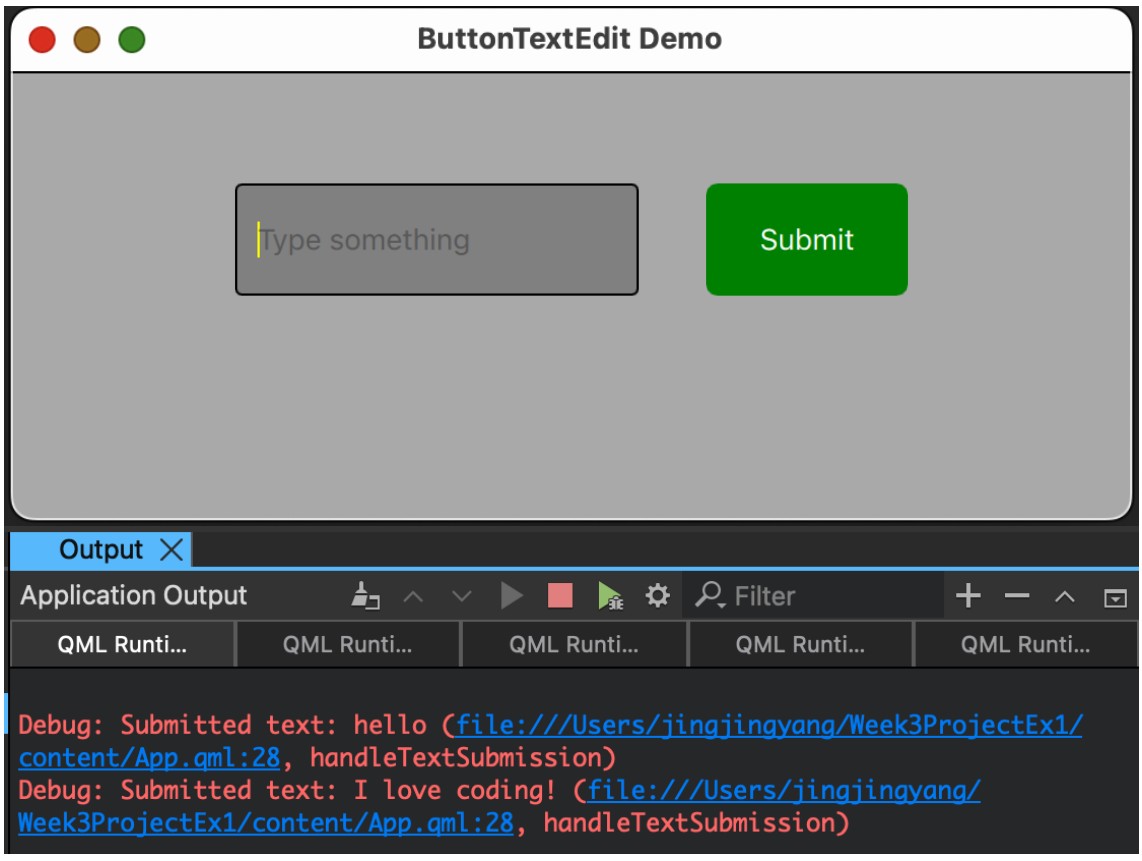
3.1.7 Use of component ButtonTextEdit

App.qml — visible

```qml
import QtQuick 6.5
import QtQuick.Controls 6.5
import Week3ProjectEx1

Window {
    visible: true
    width: 500
    height: 200
    title: "ButtonTextEdit Demo"
    color: "darkgray" // Blue Gray

    ButtonTextEdit {
        x: 100
        y: 50

        // Customizing the ButtonTextEdit component
        buttonText: "Submit"
        placeholderText: "Type something"

        textInputBackgroundColor: "gray"  // Setting text edit color
        buttonBackgroundColor: "green" // Setting button color

        textInputColor: "yellow" // Setting text edit text color
        buttonColor: "white" // Setting button text color

        // Function to handle text submission
        function handleTextSubmission(submittedText) {
            console.log("Submitted text:", submittedText);
        }

        // Connecting the function to the textSubmitted signal
        onTextSubmitted: handleTextSubmission(text)
    }
}
```




ButtonTextEdit Demo
hello
Submit
Output
Application Output
Filter
QML Runti...
QML Runti...
QML Runti...
QML Runti...
QML Runti...
Debug: Submitted text: hello (file:///Users/jingjingyang/Week3ProjectEx1/content/App.qml:28, handleTextSubmission)


ButtonTextEdit Demo
Type something
Submit
Output
Application Output
Filter
QML Runti...
QML Runti...
QML Runti...
QML Runti...
QML Runti...
Debug: Submitted text: hello (file:///Users/jingjingyang/Week3ProjectEx1/content/App.qml:28, handleTextSubmission)
Debug: Submitted text: I love coding! (file:///Users/jingjingyang/Week3ProjectEx1/content/App.qml:28, handleTextSubmission)

3.2 Create a simple REST based app with QML

3.2.1 Requirements

Objective: Develop a simple weather application (or other REST based app of your choice) using QML, which demonstrates the use of layouts, fetching data from a remote server (like OpenWeatherMap API), and dynamically updating the UI. This exercise will help students understand network operations in QML, the use of row and column layouts, and basic UI elements like buttons and text displays.

1. **Setup and API Integration**:
    - Create a new QML project for the weather application.
    - Register and obtain an API key from OpenWeatherMap (or any similar weather API or other REST service of your choice).
    - Write a function in QML/JavaScript to fetch weather data from the API.
2. **Design the User Interface**:
    - Use Column and Row layouts to organize the UI elements.
    - Display key weather information such as temperature, weather conditions, and wind speed.
    - Each piece of information should be in its separate Text element within the layout.
    - Use a gradient rectangle as a background to reflect the current weather with it's colour (choose your own colours), e.g.
        - Sunny: lightblue/yellow
        - Cloudy: blue/gray
        - Rain: gray/white
        - ...
3. **Implement the Refresh Button**:
    - Add a Button to the interface.
    - Set its text to "Refresh" or similar.
    - Implement the onClicked event to trigger the weather data fetch function.
4. **Data Handling and Display**:
    - Parse the JSON response from the weather API.
    - Update the UI elements with the new weather data upon each fetch.
5. **Error Handling**:
    - Implement basic error handling for network requests and data parsing.
6. **Challenge (optional)**:
    - To your GUI, add a weather icon to reflect the current weather (sunny.png, clouds.png etc.)

3.2.2 Icon list

Icon list

Day icon	Night icon	Description
01d.png	01n.png	clear sky
02d.png	02n.png	few clouds
03d.png	03n.png	scattered clouds
04d.png	04n.png	broken clouds
09d.png	09n.png	shower rain
10d.png	10n.png	rain
11d.png	11n.png	thunderstorm
13d.png	13n.png	snow
50d.png	50n.png	mist

3.2.3 App.qml

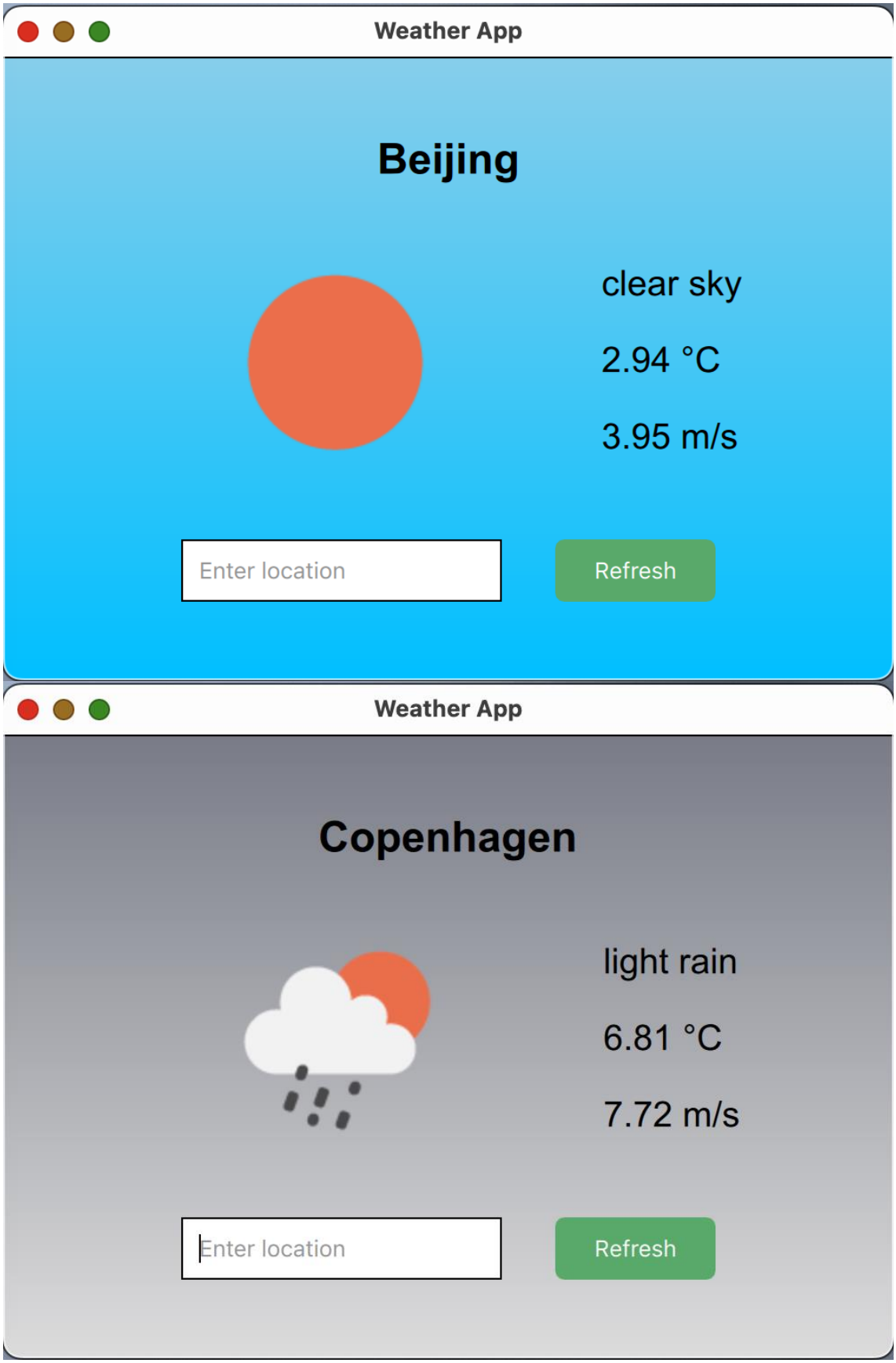
App.qml — Window

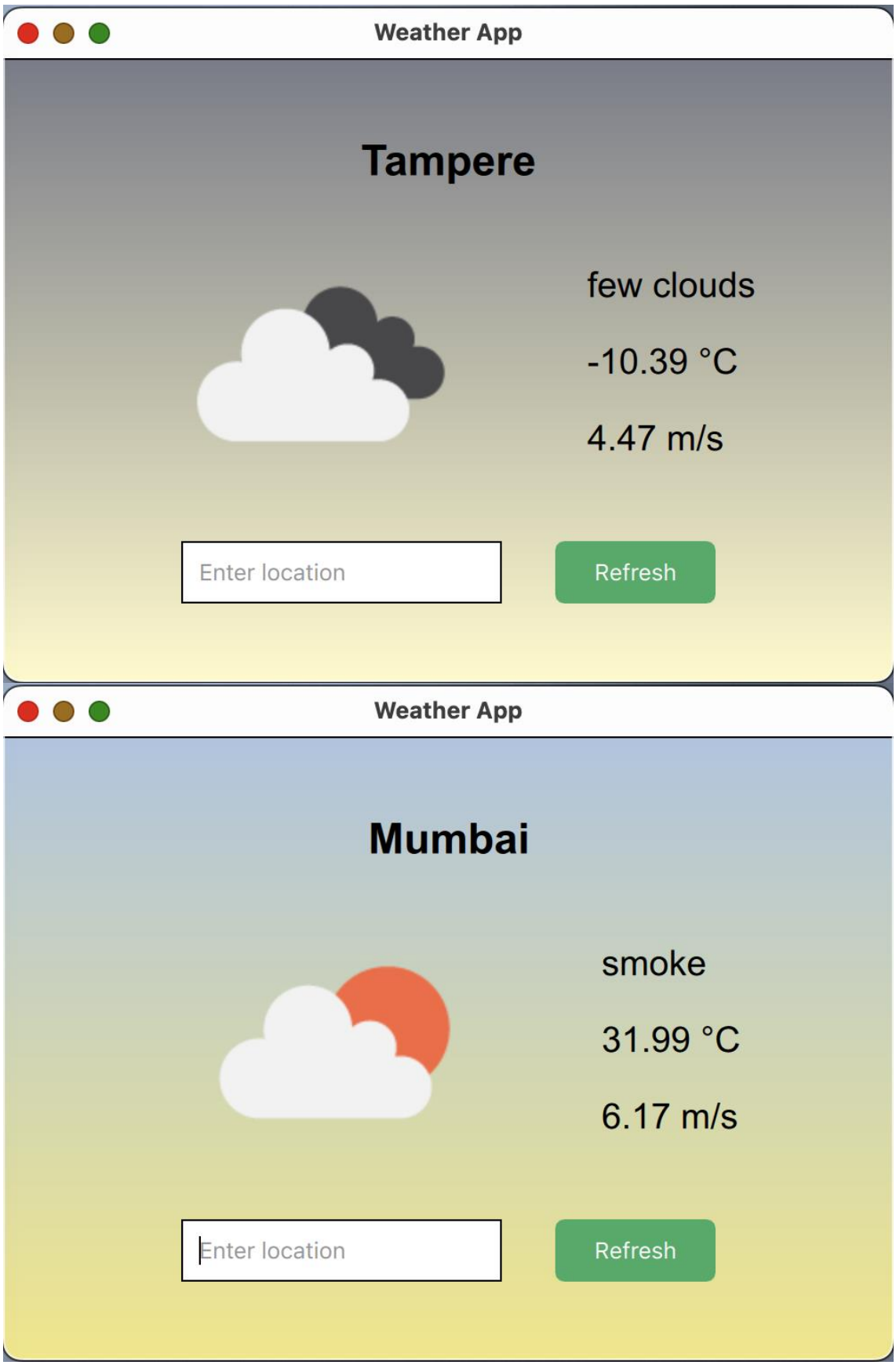
```
import QtQuick 6.5
import Week3ex2

Window {
    visible: true
    width: 500
    height: 350
    title: "Weather App"

    Component.onCompleted: {
        // Fetch weather data for a default location when the app starts
        fetchWeatherData("Tampere");
    }

    Rectangle {
        id: background
        anchors.fill: parent
        gradient: Gradient {
            GradientStop { id: stop1; position: 0.0; color: "lightblue" } // Default gradient
            GradientStop { id: stop2; position: 1.0; color: "yellow" }
        }

        Column {
            anchors.centerIn: parent
            width: parent.width

            Text {
                id: location;
                anchors.horizontalCenter: parent.horizontalCenter
                text: "Location"
                font {
                    pixelSize: 24
                    family: "Arial"
                    bold: true
                }
            }

            Row {
                anchors.horizontalCenter: parent.horizontalCenter
                spacing: 50

                Image {
                    id: weatherIcon
                    source: "image/default.png"
                    height: 200
                    width: 200
                    onStatusChanged: {
                        if (weatherIcon.status == Image.Error) {
                            console.log("Failed to load image: " + weatherIcon.errorString);
                        }
                    }
                }

```

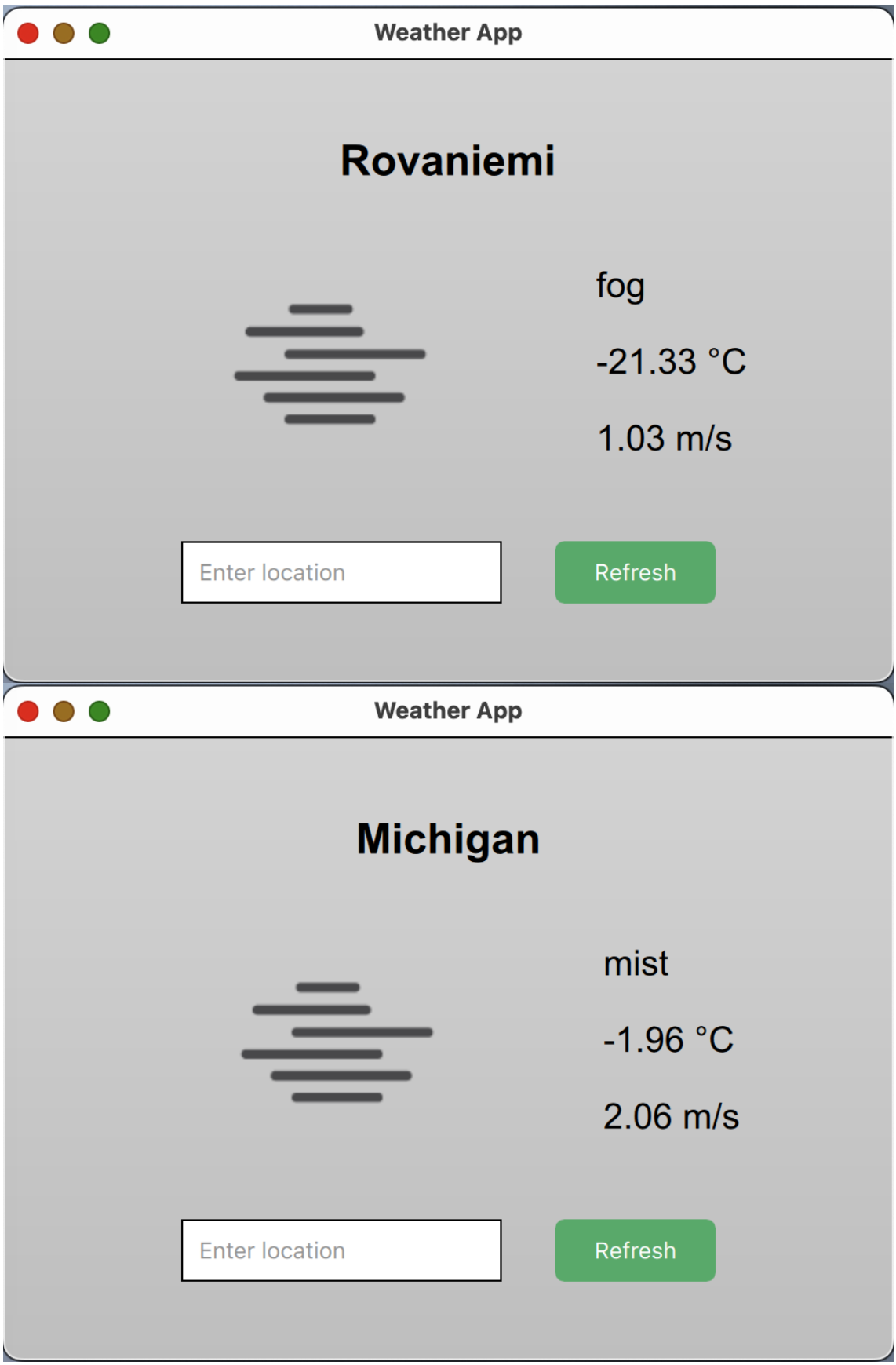
```
54                 Column {
55                     anchors.verticalCenter: parent.verticalCenter
56                     spacing: 20
57
58                     Text {
59                         id: weatherCondition;
60                         text: "Weather"
61                         font {
62                             pixelSize: 20
63                             family: "Arial"
64                         }
65                     }
66                     Text {
67                         id: temperature;
68                         text: "Temperature"
69                         font {
70                             pixelSize: 20
71                             family: "Arial"
72                         }
73                     }
74                     Text {
75                         id: windSpeed;
76                         text: "Wind Speed"
77                         font {
78                             pixelSize: 20
79                             family: "Arial"
80                         }
81                     }
82                 }
83             }
84
85             ButtonTextEdit {
86                 anchors.horizontalCenter: parent.horizontalCenter
87                 width: 300
88                 height: 35
89                 buttonText: "Refresh"
90                 placeholderText: "Enter location"
91                 textInputBackgroundColor: "white"
92                 buttonBackgroundColor: "#59A96A"
93                 textInputColor: "black"
94                 buttonColor: "white"
95                 onTextSubmitted: fetchWeatherData(text);
96             }
97         }
98     }
99
```

```
    function fetchWeatherData(location) {
        var xhr = new XMLHttpRequest();
        const API_KEY="395dc02446c77c0ac922cb465d9a395b";

        xhr.open("GET", `https://api.openweathermap.org/data/2.5/weather?q=${location}
            &units=metric&appid=${API_KEY}`, true);

        xhr.onreadystatechange = function () {
            if (xhr.readyState === XMLHttpRequest.DONE) {
                if (xhr.status === 200) {
                    var response = xhr.responseText;

                    processData(response);
                } else {
                    console.log("Error fetching weather data");
                }
            }
        }
        xhr.send();
    }

    function processData(response){
        var jsonData = JSON.parse(response);

        location.text = jsonData.name;
        temperature.text = jsonData .main.temp + " °C";
        weatherCondition.text = jsonData.weather[0].description;
        windSpeed.text = jsonData.wind.speed + " m/s";

        var icon = jsonData.weather[0].icon;
        weatherIcon.source = `https://openweathermap.org/img/wn/${icon}@4x.png`;

        // Update background based on weather condition
        updateBackground(jsonData.weather[0].description);
    }

    function updateBackground(condition) {
        // Change background gradient based on weather condition
        if (condition.includes("cloud")) {
            stop1.color = "#797C87";
            stop2.color = "#FFFACD";
        } else
            if (condition.includes("clear")) {
            stop1.color = "#87CEEB";
            stop2.color = "#00BFFF";
        } else if (condition.includes("rain")) {
            stop2.color = "#DCDCDC";
        }else if (condition.includes("snow") || condition.includes("sleet")) {
            stop1.color = "#F0F8FF";
            stop2.color = "#F8F8FF";
        }else if (condition.includes("fog") || condition.includes("mist")) {
            stop1.color = "#D3D3D3";
            stop2.color = "#BEBEBE";
        }else if (condition.includes("thunder")) {
            stop1.color = "#A0522D";
            stop2.color = "#2E2E2E";
        } else {
            // Default gradient for other conditions
            stop1.color = "#B0C4DE";
            stop2.color = "#F0E68C";
        }
    }
}
```

3.2.4 App testing




Weather App
Tampere
few clouds
-10.39 °C
4.47 m/s
Enter location
Refresh
Weather App
Mumbai
smoke
31.99 °C
6.17 m/s
Enter location
Refresh


Weather App
Rovaniemi
fog
-21.33 °C
1.03 m/s
Enter location
Refresh
Weather App
Michigan
mist
-1.96 °C
2.06 m/s
Enter location
Refresh

4 Week exercises

4.1 Defining Traffic Light States

4.1.1 Requirements

Objective: Create the initial setup of the traffic light with three distinct states representing the red, yellow, and green lights.

1. **Setup the Project**
   - Create a new QML file and set up a basic QML application structure with an ApplicationWindow.
2. **Create the Traffic Light UI**
   - Inside the ApplicationWindow, add a parent Rectangle that will serve as the traffic light's background.
   - Add three Rectangle elements inside the parent Rectangle, each representing the traffic lights (red, yellow, green). Set their initial colors to represent lights that are turned off (e.g., dark red, dark yellow, dark green).
3. **Define States**
   - For each color (red, yellow, green), define a state that changes the color of the respective Rectangle to a "lit" color (e.g., bright red for the red light).
   - Use the State element within the parent Rectangle to define these states, including PropertyChanges for each light's color property.

Deliverable: A QML application that displays a traffic light. The lights don't change yet, as this step focuses on setting up the visual elements and defining the states.

4.1.2 TrafficLight.qml

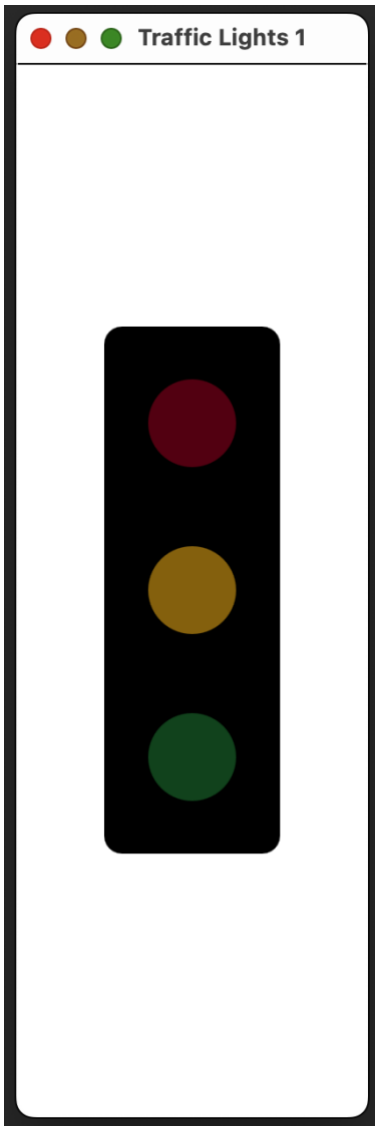
An ApplicationWindow is created and a Rectangle is added to serve as the background for the traffic light. Inside this Rectangle, three more Rectangle elements are added to represent the traffic lights. The radius: width / 2 attribute transforms the rectangles into perfect circles.

The state of each light is defined using the State element, with a "stop" state that lights up the red light, a "go" state that lights up the green light, and a "wait" state that lights up the yellow light. The colors of the lights change according to their respective states.

TrafficLight.qml — states

```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 200
    height: 600
    title: "Traffic Lights 1"

    Rectangle {
        id: trafficLight
        width: 100
        height: 300
        radius: 10
        anchors.centerIn: parent
        color: "black"

        Rectangle {
            id: redLight
            width: 50
            height: 50
            color: "#520011" // dark red
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.top: parent.top
            anchors.topMargin: 30
            radius: width / 2
        }

        Rectangle {
            id: yellowLight
            width: 50
            height: 50
            color: "#84600D" // dark yellow
            anchors.centerIn: parent
            radius: width / 2
        }

        Rectangle {
            id: greenLight
            width: 50
            height: 50
            color: "#11421C" // dark green
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.bottom: parent.bottom
            anchors.bottomMargin: 30
            radius: width / 2
        }

        // State Definitions
        states: [
            State {
                name: "stop"
                // Bright the red light, Dim the green light and yellow light
                PropertyChanges { target: redLight; color: "red" }
                PropertyChanges { target: greenLight; color: "#11421C" }
                PropertyChanges { target: yellowLight; color: "#84600D" }
            },
            State {
                name: "go"
                // Bright the green light, Dim the red light and yellow light
                PropertyChanges { target: greenLight; color: "#1AE802" }
                PropertyChanges { target: redLight; color: "#520011" }
                PropertyChanges { target: yellowLight; color: "#84600D" }
            },
            State {
                name: "wait"
                // Bright the yellow light, Dim the red light and green light
                PropertyChanges { target: yellowLight; color: "yellow" }
                PropertyChanges { target: redLight; color: "#520011" }
                PropertyChanges { target: greenLight; color: "#11421C" }
            }
        ]
    }
}
```

4.1.3 Live Preview



4.2 Adding Timer and Transitions

4.2.1 Requirements

Objective: Implement a timer to cycle through the traffic light states automatically, with transitions to smoothly animate the color changes.

1. **Implement a Timer**
   - Add a Timer component within the parent Rectangle. Configure it to trigger at regular intervals, each time changing the state of the traffic light to the next in the sequence (red → yellow → green → red).
2. **Define Transitions**
   - Use the Transition element to define animations that occur when changing between states. For example, a ColorAnimation can be used to smoothly transition the light colors.

Deliverable: The traffic light now cycles through red, yellow, and green states automatically, with smooth transitions between colors.

4.2.2 TrafficLightTransition.qml

The application consists of a single traffic light with three states: "stop", "go", and "wait". Each state corresponds to a specific color: red for "stop", green for "go", and yellow for "wait".

The traffic light cycles through these states automatically, with smooth transitions (red → green → yellow → red) between colors, which is achieved through the use of a Timer component. The duration of each state varies ("stop" lasts for 3 seconds, "wait" for 1.5 seconds, and "go" for 5 seconds,). The traffic light is displayed within an ApplicationWindow, making it a standalone application.

In a real-world setting, typically, the order of traffic light signals is red → green → yellow → red. Red signals drivers to stop, green signals drivers to go, and yellow acts as a warning that the light is about to change from green to red.

Moreover, the typical duration of each state of a traffic light might look something like this:

- "Stop" (red light): about 55 seconds, allowing cross traffic to move.
- "Wait" (yellow light): about 5 seconds, serving as a warning that the light is about to change to red.
- "Go" (green light): about 60 seconds, allowing traffic to move in the direction of the green light.

Having a smooth transition where one light dims as the next one brightens can be aesthetically pleasing due to the smoothness of the transition. However, there could indeed be a brief period where neither light is at full brightness, which might be confusing or less visible to drivers.

In real-world traffic light systems, the transition between colors is usually immediate without any dimming or brightening effects to avoid any confusion. Each light is either fully on or fully off, and the change from one color to the next is instantaneous.

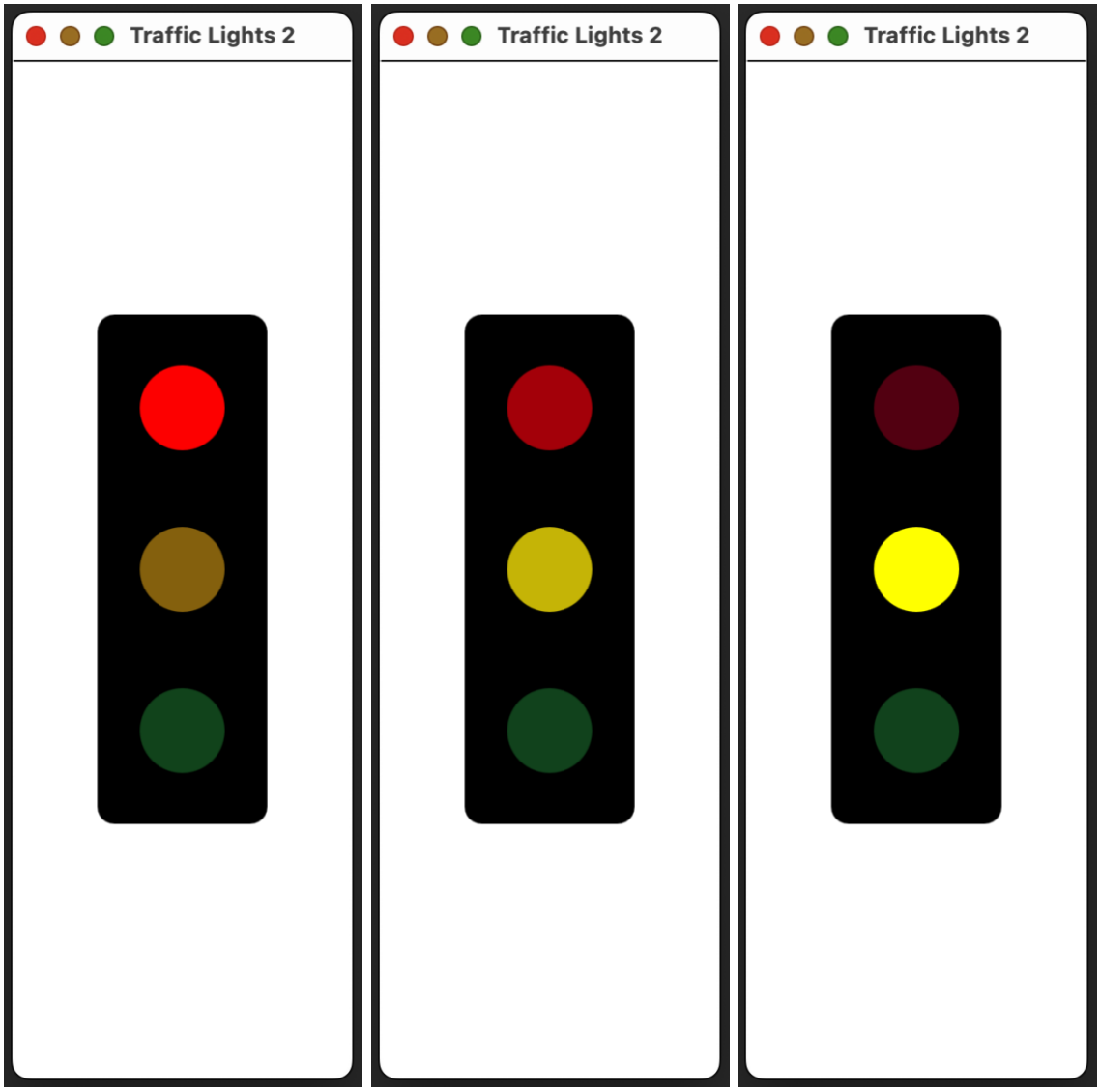
So, while a smooth transition might be visually appealing in a simulated environment or in certain artistic contexts, it's typically not used in actual traffic light systems due to safety and visibility concerns.

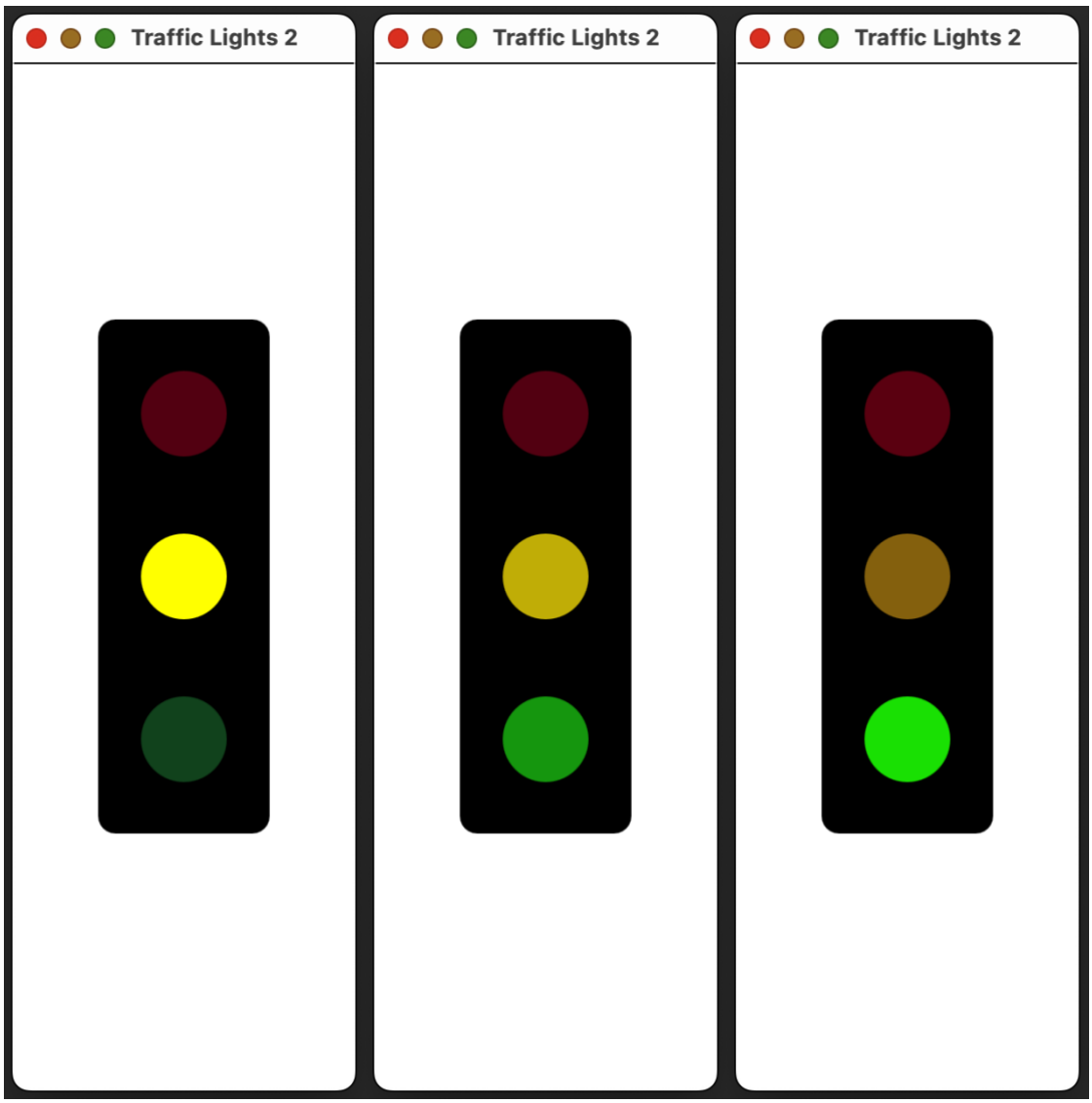
TrafficLightTransition.qml — radius

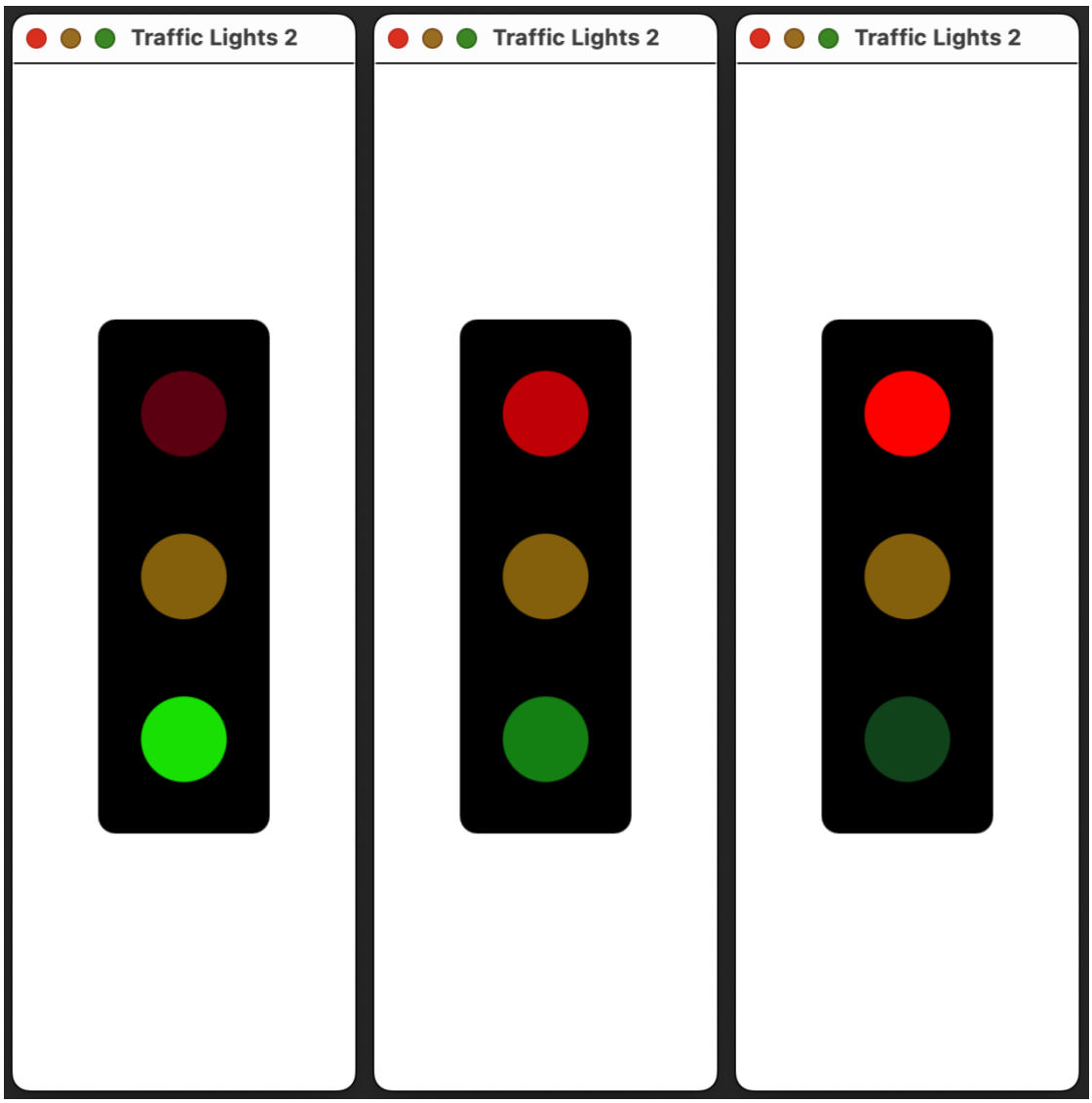
```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 200
    height: 600
    title: "Traffic Lights 2"

    Rectangle {
        id: trafficLight
        width: 100
        height: 300
        radius: 10
        anchors.centerIn: parent
        color: "black"

        state: "stop"

        Rectangle {
            id: redLight
            width: 50
            height: 50
            color: "#520011" // dark red
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.top: parent.top
            anchors.topMargin: 30
            radius: width / 2
        }

        Rectangle {
            id: yellowLight
            width: 50
            height: 50
            color: "#84600D" // dark yellow
            anchors.centerIn: parent
            radius: width / 2
        }

        Rectangle {
            id: greenLight
            width: 50
            height: 50
            color: "#11421C" // dark green
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.bottom: parent.bottom
            anchors.bottomMargin: 30
            radius: width / 2
        }

```

```
        // State Definitions
        states: [
            State {
                name: "stop"
                // Bright the red light, Dim the green light and yellow light
                PropertyChanges { target: redLight; color: "red" }
                PropertyChanges { target: greenLight; color: "#11421C" }
                PropertyChanges { target: yellowLight; color: "#84600D" }
            },
            State {
                name: "go"
                // Bright the green light, Dim the red light and yellow light
                PropertyChanges { target: greenLight; color: "#1AE802" }
                PropertyChanges { target: redLight; color: "#520011" }
                PropertyChanges { target: yellowLight; color: "#84600D" }
            },
            State {
                name: "wait"
                // Bright the yellow light, Dim the red light and green light
                PropertyChanges { target: yellowLight; color: "yellow" }
                PropertyChanges { target: redLight; color: "#520011" }
                PropertyChanges { target: greenLight; color: "#11421C" }
            }
        ]

            // Timer Definition
        Timer {
            interval: 1000 // 1 second
            running: true
            repeat: true // Timer will automatically reset after each interval
            onTriggered: {
                // Logic to change states based on current state
                if (trafficLight.state === "go") {
                    interval = 3000 // red light 3 seconds
                    trafficLight.state = "stop" // Change state to 'stop'
                } else if (trafficLight.state === "stop") {
                    interval = 1500 // yellow light 3 seconds
                    trafficLight.state = "wait" // Change state to 'wait'
                } else if (trafficLight.state === "wait") {
                    trafficLight.state = "go" // green light 3 seconds
                    interval = 5000 // Set interval to 5 seconds in 'go' state
                }
            }
        }

        // Transition Definition
        transitions: [
            Transition {
                from: "*" // Applies from any state
                to: "*" // Applies to any state
                ColorAnimation { duration: 1000 } // Animates color changes over 1 second
            }
        ]
    }
}
```

4.2.3 Live Preview




Traffic Lights 2
Traffic Lights 2
Traffic Lights 2



4.3 Implementing Blinking Yellow Light and Control Buttons

4.3.1 Requirements

Objective: Add functionality to switch the traffic light on and off. When off, the yellow light should blink. Also, add buttons to control these functionalities.

1. **Blinking Yellow Light**
   - Define an additional state or use a conditional animation for the yellow light that causes it to blink when the traffic light is "off". This can involve changing the opacity of the yellow light in a loop.
   - Ensure this blinking state is only active when the traffic light system is turned off.
2. **Add Control Buttons**
   - Add two Button elements below the traffic light: "Traffic Lights On" and "Traffic Lights Off".
   - Implement the onClicked event for each button to control the traffic light's operation. Turning the lights on should resume the normal operation (cycling through red, yellow, green), and turning them off should activate the blinking yellow light.

3. **Adjust the Timer and State Logic**
    - Modify the timer's behavior to stop the regular state cycle when the traffic light is turned off and resume it when turned on again.
    - Ensure the logic correctly handles the transition between the blinking yellow state and the normal operational states based on the button clicks.

Deliverable: A complete traffic light simulation with operational control. Users can start and stop the traffic light sequence, with a blinking yellow light indicating the "off" state.

4.3.2 TrafficLightBlink.qml

The updated QML code includes a blinking yellow light feature and control buttons for a traffic light application. The blinking light is implemented by adding a new state and using an OpacityAnimator to animate the yellow light's opacity when the trafficLight.state is "blink".

Two control buttons have been added at the bottom of the ApplicationWindow. When the "OFF" button is clicked, the state is set to "blink", and the timer is stopped, causing the yellow light to blink. On clicking the "ON" button, the state is reset to "stop" and the timer starts running again, stopping the blinking and resuming the traffic light's normal cycle through the stop, go, wait states.

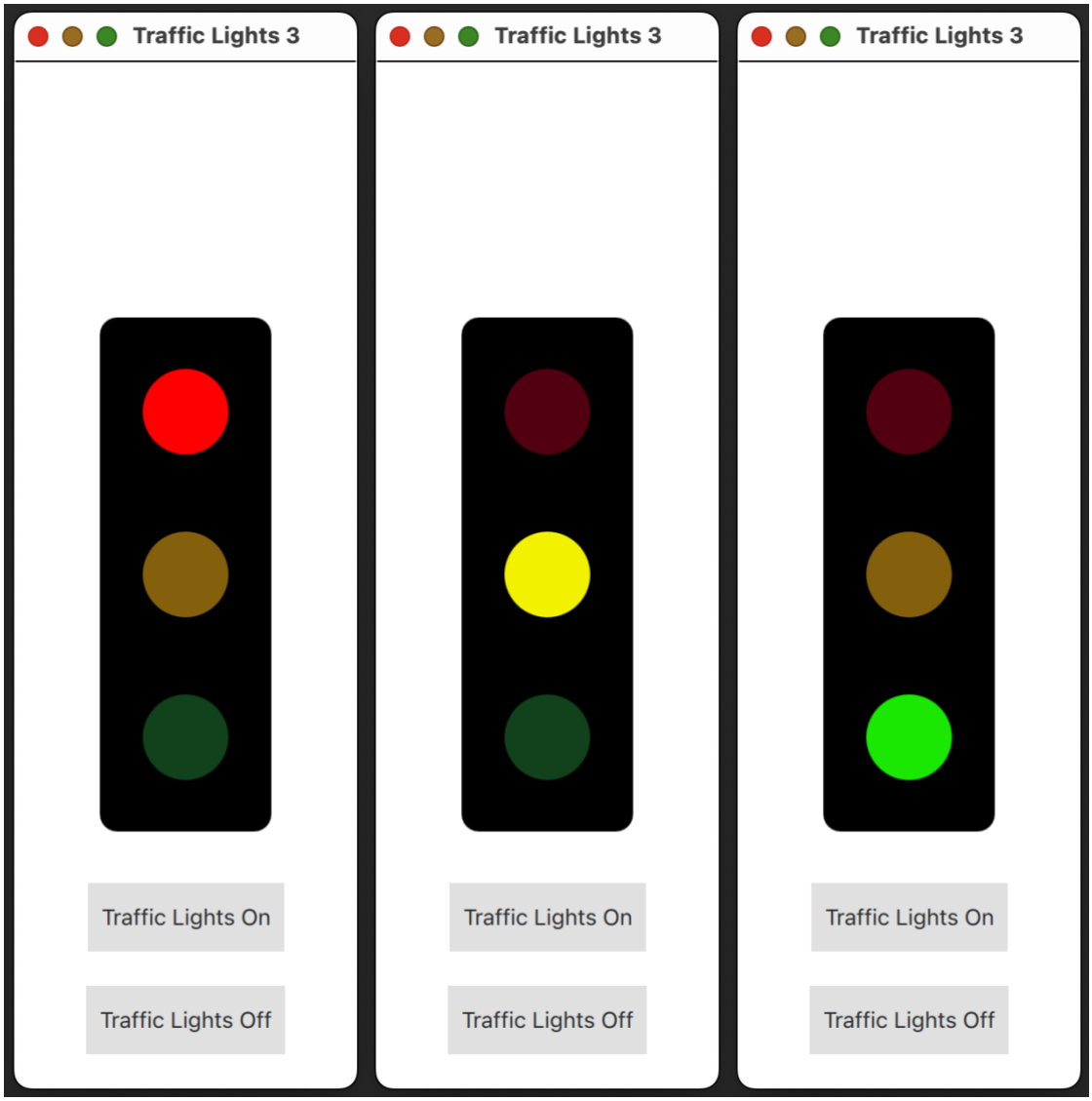
TrafficLightBlink.qml

```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 200
    height: 600
    title: "Traffic Lights 3"

    Rectangle {
        id: trafficLight
        width: 100
        height: 300
        radius: 10
        anchors.centerIn: parent
        color: "black"

        state: "stop"

        Rectangle {
            id: redLight
            width: 50
            height: 50
            color: "#520011"
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.top: parent.top
            anchors.topMargin: 30
            radius: width / 2
        }

        Rectangle {
            id: yellowLight
            width: 50
            height: 50
            color: "#84600D" // dark yellow
            anchors.centerIn: parent
            radius: width / 2

            OpacityAnimator {
                target: yellowLight
                id: opacityAnimator
                running: trafficLight.state === "blink"
                loops: Animation.Infinite
                from: 1
                to: 0
                duration: 500
            }
        }

        Rectangle {
            id: greenLight
            width: 50
            height: 50
            color: "#11421C" // dark green
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.bottom: parent.bottom
            anchors.bottomMargin: 30
            radius: width / 2
        }

```

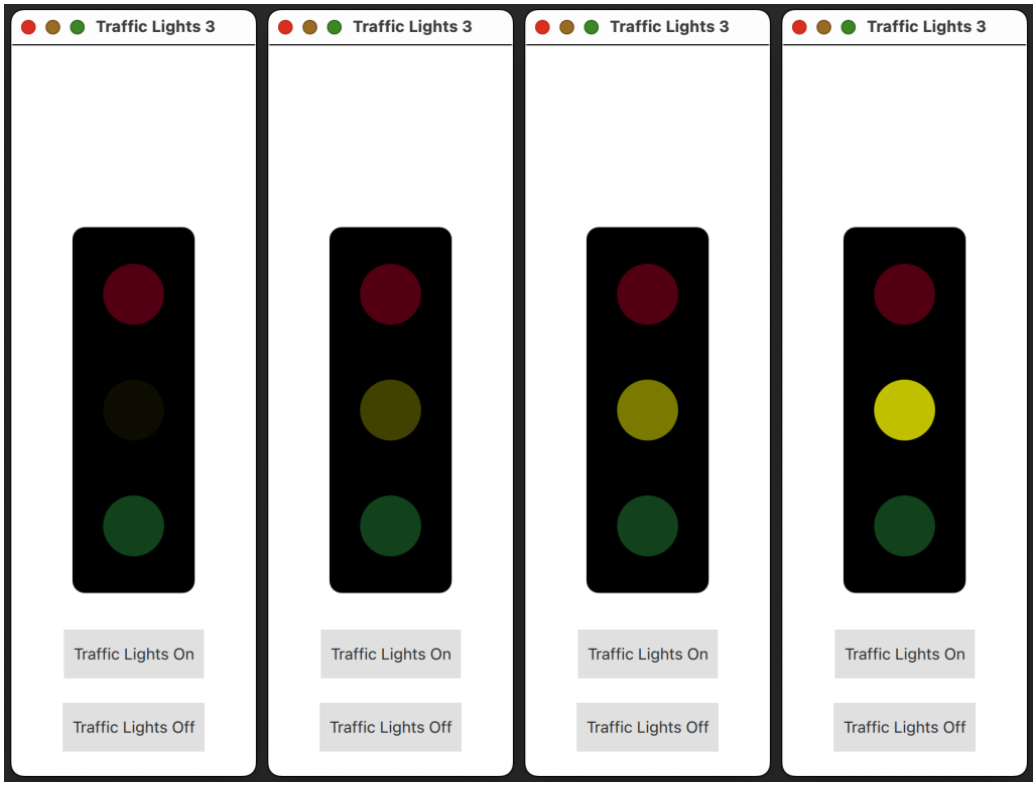
```
        // State Definitions
        states: [
            State {
                name: "stop"
                PropertyChanges { target: redLight; color: "red" }
                PropertyChanges { target: greenLight; color: "#11421C" }
                PropertyChanges { target: yellowLight; color: "#84600D" }
            },
            State {
                name: "go"
                PropertyChanges { target: greenLight; color: "#1AE802" }
                PropertyChanges { target: redLight; color: "#520011" }
                PropertyChanges { target: yellowLight; color: "#84600D" }
            },
            State {
                name: "wait"
                PropertyChanges { target: yellowLight; color: "yellow" }
                PropertyChanges { target: redLight; color: "#520011" }
                PropertyChanges { target: greenLight; color: "#11421C" }
            },
            State {
                name: "blink"
                PropertyChanges { target: yellowLight; color: "yellow" }
                PropertyChanges { target: redLight; color: "#520011" }
                PropertyChanges { target: greenLight; color: "#11421C" }
            }
        ]

        // Timer Definition
        Timer {
            interval: 1000 // 1 second
            running: true
            repeat: true // Timer will automatically reset after each interval
            onTriggered: {
                // Logic to change states based on current state
                if (trafficLight.state === "go") {
                    interval = 3000 // red light 3 seconds
                    trafficLight.state = "stop" // Change state to 'stop'
                } else if (trafficLight.state === "stop") {
                    interval = 1500 // yellow light 3 seconds
                    trafficLight.state = "wait" // Change state to 'wait'
                } else if (trafficLight.state === "wait") {
                    trafficLight.state = "go" // green light 3 seconds
                    interval = 5000 // Set interval to 5 seconds in 'go' state
                }
            }
        }

        // Transition Definition
        transitions: [
            Transition {
                from: "*" // Applies from any state
                to: "*" // Applies to any state
                ColorAnimation { duration: 1000 } // Animates color changes over 1 second
            }
        ]
    }

    Button {
        text: "Traffic Lights On"
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottom: parent.bottom
        anchors.bottomMargin: 80
        onClicked: {
            trafficLight.state = "stop"
            timer.start()
        }
    }

    Button {
        text: "Traffic Lights Off"
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottom: parent.bottom
        anchors.bottomMargin: 20
        onClicked: {
            trafficLight.state = "blink"
            timer.stop()
        }
    }
}
```

4.3.3 Live Preview

When traffic lights on:



When traffic lights off:



4.4 Simple speedometer dashboard for a vehicle with engine on/off functionality

4.4.1 Requirements

Objective: Create the speedometer display with engine on/off control. This involves setting up the graphical representation of the speedometer and implementing the logic to turn the engine on and off.

1. **Prepare the Assets:**
   - Ensure you have PNG files for the speedometer base and the needle (you can use the ones from the course moodle page). Place these in your project's assets folder.
2. **Create the Speedometer Display:**
   - Use an Image item to display the speedometer base.
   - Add another Image item for the needle, which will rotate to indicate speed. Use the rotation property to adjust its position.
3. **Implement Engine On/Off Logic:**
   - Add a Button for turning the engine on and off. This will control whether the vehicle can accelerate or brake.
   - Define a boolean property (e.g., engineOn) that changes when the button is clicked, representing the engine's state.
   - Use some cold gradient rectangle colour as the dashboard background when the engine is on. Change the colour to indicate the engine on feature.
4. **Set Up Initial States and Transitions:**

- Define states for the engine being on and off. When the engine is off, the needle should point to the minimum speed, and acceleration/braking should be disabled.
- Implement a transition for the needle to smoothly move to the minimum speed position when the engine is turned off.

By the end of this exercise you should have a speedometer with a needle that reacts to the engine's state. The engine on/off button toggles the engineOn property, but accelerating and braking do not yet affect the speed.

4.4.2 SpeedometerEngine.qml

The application is a simple simulation of a speedometer in a vehicle. The speedometer has two states, represented by the engineOn boolean: ON and OFF. This state is toggled by clicking the EngineButton.

When the engine is ON, the background color of the application changes from a gradient of light grey and dark grey (#7C7C7C) to a gradient of sky blue and dark blue (#03395C) to visually indicate the change in state. When the engine is OFF, the needle (which indicates speed) smoothly moves back to its minimum speed position (-137 degrees).

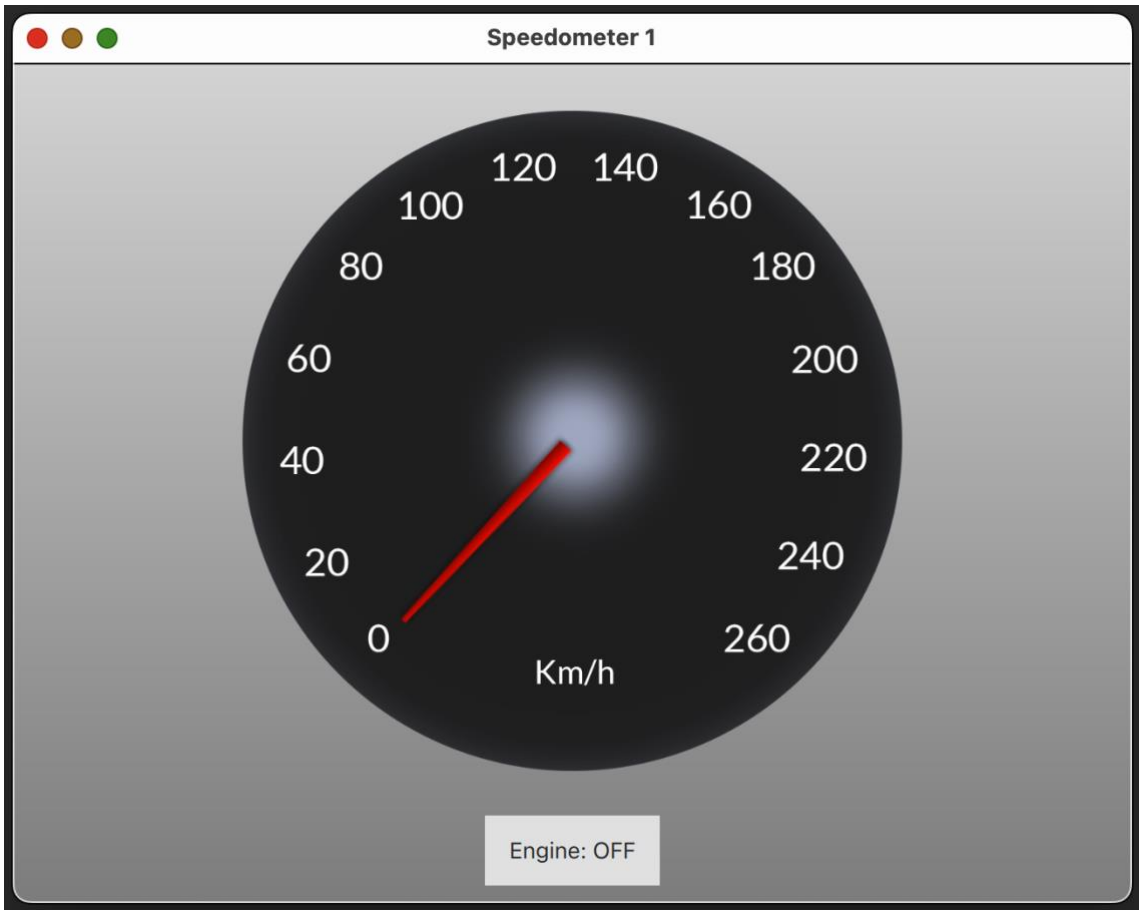
The speedometer and needle are represented by two image files (speedo.png and needlered.png). The needle's rotation is controlled by the rotation property, and it rotates around its bottom point, which is set as the transformOrigin.

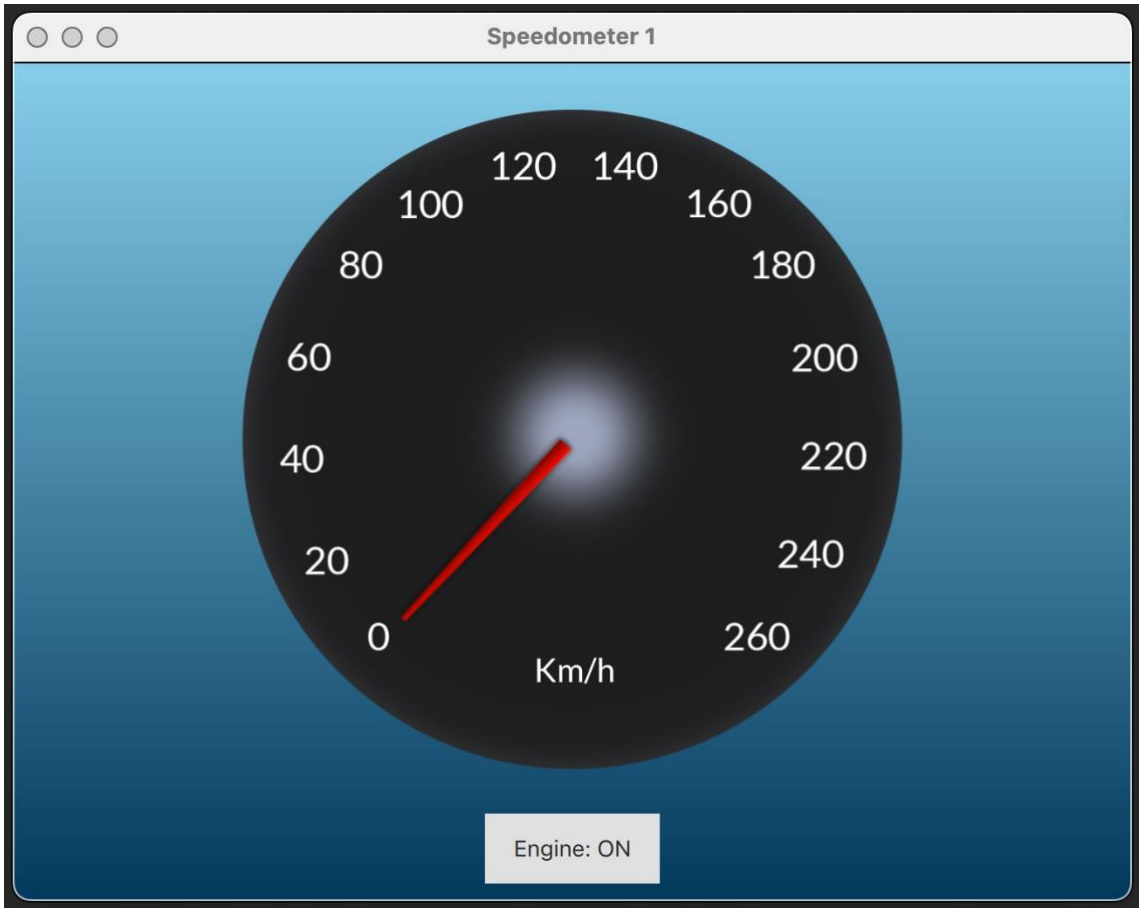
The transitions between the engine ON and OFF states are animated. When turning off the engine, the needle smoothly moves to the minimum speed position over the course of 1 second.

SpeedometerEngine.qml — onClicked

```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 640
    height: 480
    title: "Speedometer 1"

    property bool engineOn: false

    Rectangle {
        id: background
        anchors.fill: parent
        gradient: Gradient {
            GradientStop { position: 0.0; color: engineOn ? "skyblue" : "lightgrey" }
            GradientStop { position: 1.0; color: engineOn ? "#03395C" : "#7C7C7C" }
        }

        states: [
            State {
                name: "engineOff"
                when: !engineOn
                PropertyChanges {
                    target: needle
                    rotation: -137 // minimum speed position
                }
            },
            State {
                name: "engineOn"
                when: engineOn
            }
        ]

        transitions: [
            Transition {
                from: "engineOn"
                to: "engineOff"
                NumberAnimation {
                    target: needle
                    property: "rotation"
                    duration: 1000 // Smooth transition duration
                }
            }
        ]

        Image {
            id: speedo
            height: parent.height * 0.9
            width: height
            fillMode: Image.PreserveAspectFit // Maintains aspect ratio
            source: "images/speedo.png"
            anchors.horizontalCenter: parent.horizontalCenter
        }

        Image {
            id: needle
            source: "images/needlered.png"
            height: parent.height * 0.3
            anchors.bottom: speedo.verticalCenter
            anchors.horizontalCenter: speedo.horizontalCenter
            transformOrigin: Image.Bottom
            rotation: -137
        }

        Button {
            id: engineButton
            text: engineOn ? "Engine: ON" : "Engine: OFF"
            anchors.horizontalCenter: parent.horizontalCenter
            anchors.bottom: parent.bottom
            anchors.bottomMargin: 10
            onClicked: {
                engineOn = !engineOn
            }
        }
    }
}
```

4.4.3 Live Preview





4.5 Implementing Acceleration, Braking, and Dynamic Needle Movement

4.5.1 Requirements

Objective: Extend the speedometer functionality to include acceleration and braking, with the needle providing visual feedback on the current speed. This requires adding controls for speed changes and implementing the logic to update the needle's position based on the vehicle's speed.

1. **Add Acceleration and Braking Controls:**
   - Introduce Button elements for acceleration and braking. These buttons should only be functional (enabled) when the engine is on.
2. **Implement Speed Control:**
   - Create a numeric property (e.g., currentSpeed) to track the vehicle's speed. Accelerating should increase this value, while braking decreases it, within defined limits (e.g., 0 to 100).
   - Use property bindings or change handlers to ensure the needle's rotation reflects the current speed.
3. **Animate Needle Movements:**
   - When the speed changes due to acceleration or braking, use NumberAnimation on the needle's rotation property to animate its movement. The animation speed can be adjusted to simulate the responsiveness of the vehicle.
4. **Enhance User Interaction:**
   - Optionally, implement logic to gradually increase or decrease the speed if an acceleration or brake button is held down, mimicking more realistic vehicle control.
5. **Introduce Transitions for Engine State Changes:**
   - Further refine the experience by adding smooth transitions for the needle when the engine is turned on or off, ensuring a visually appealing start-up and shut-down sequence for the speedometer.

The speedometer is finalized by making the acceleration and braking buttons functional, with the needle's movement accurately reflecting changes in speed. The application will visually and interactively simulate a vehicle's speedometer, complete with engine control and speed adjustments.

4.5.2 SpeedometerMove.qml

Features:

- **Engine Toggle:** Start or stop the engine with a simple button click, changing the dashboard color to reflect the engine's state – sky blue for on, and light grey for off.
- **Real-Time Speed Display:** Watch the speed change in real time on a stylish speedometer. The current speed is dynamically displayed in red text.
- **Accelerate and Brake Controls:** Control the speed with "Accelerate" and "Brake" buttons. The speed increments or decrements in real-time, simulating actual car behavior.

Please note, on a Mac laptop, you can simulate a press by holding down the button on the trackpad and releasing when you want to simulate the release of the button.

- **Coasting Functionality:** Experience how a car coasts when the engine is turned off. The speed gradually decreases, mimicking the natural deceleration of a vehicle.
- **Smooth Needle Movement:** The speedometer needle moves smoothly with a visually pleasing animation that responds to speed changes, providing an authentic dashboard experience.
- **Responsive Design:** Designed to fit a standard application window of 640x480, ensuring compatibility with various devices and screen sizes.

How It Works:

- The dashboard background color changes based on the engine's state.
- Press the "Engine: ON/OFF" button to toggle the engine state.
- Use the "Accelerate" button to increase speed, and the "Brake" button to decrease speed.
- Observe the speedometer needle and digital speed display respond to your actions.
- When the engine is turned off while in motion, the car coasts to a stop with a realistic deceleration.

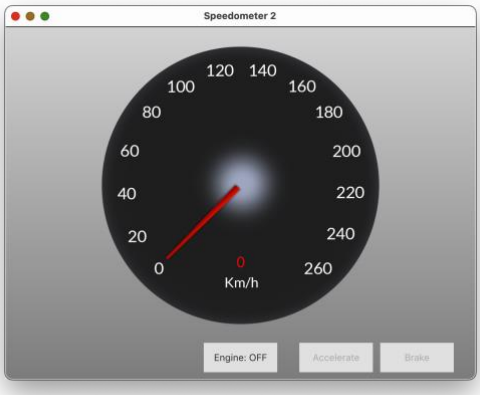
SpeedometerMove.qml — Gradient

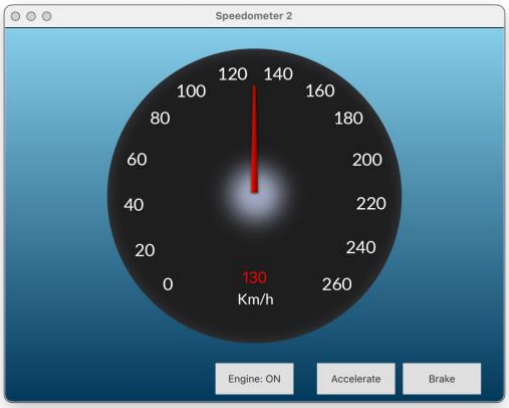
```
1  import QtQuick 2.15
2  import QtQuick.Controls 2.15
3
4  ApplicationWindow {
5      visible: true
6      width: 640
7      height: 480
8      title: "Speedometer 2"
9
10     property bool engineOn: false
11     property int currentSpeed: 0
12
13     Rectangle {
14         id: background
15         anchors.fill: parent
16         gradient: Gradient {
17             GradientStop { position: 0.0; color: engineOn ? "skyblue" : "lightgrey" }
18             GradientStop { position: 1.0; color: engineOn ? "#03395C" : "#7C7C7C" }
19         }
20
21         Image {
22             id: speedo
23             height: parent.height * 0.9
24             width: height
25             fillMode: Image.PreserveAspectFit
26             source: "images/speedo.png"
27             anchors.horizontalCenter: parent.horizontalCenter
28         }
29
30         Text {
31             id: speedDisplay
32             text: currentSpeed
33             color: "red"
34             font.pixelSize: 20
35             anchors.horizontalCenter: speedo.horizontalCenter
36             anchors.bottom: speedo.bottom
37             anchors.bottomMargin: 100
38
39
40         }
```

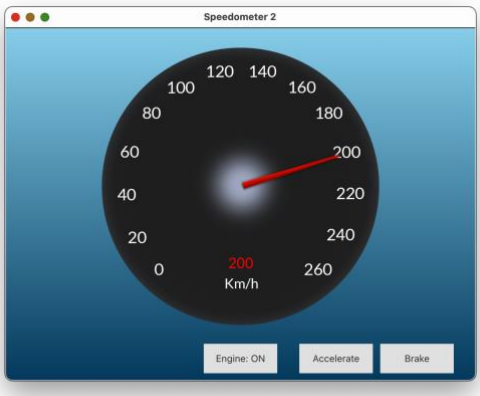
```
41
42          Image {
43              id: needle
44              source: "images/needlered.png"
45              height: parent.height * 0.3
46              anchors.bottom: speedo.verticalCenter
47              anchors.horizontalCenter: speedo.horizontalCenter
48              transformOrigin: Image.Bottom
49              rotation: -135 + (135/130.0 * currentSpeed)
50
51              NumberAnimation {
52                  target: needle
53                  property: "rotation"
54                  duration: 500
55                  easing.type: Easing.InOutQuad
56              }
57          }
58
59          Button {
60              id: engineButton
61              text: engineOn ? "Engine: ON" : "Engine: OFF"
62              anchors.horizontalCenter: parent.horizontalCenter
63              anchors.bottom: parent.bottom
64              anchors.bottomMargin: 10
65              onClicked: {
66                  engineOn = !engineOn
67                  if (!engineOn) {
68                      coastTimer.start();
69                  } else {
70                      coastTimer.stop();
71                  }
72              }
73          }
74
75          Button {
76              id: accelerateButton
77              text: "Accelerate"
78              anchors.bottom: engineButton.bottom
79              anchors.left: engineButton.right
80              anchors.leftMargin: 30
81              enabled: engineOn
82              onPressed: {
83                  accelerateTimer.start()
84              }
85              onReleased: {
86                  accelerateTimer.stop()
87              }
88          }
89
90          Button {
91              id: brakeButton
92              text: "Brake"
93              anchors.bottom: engineButton.bottom
94              anchors.left: accelerateButton.right
95              anchors.leftMargin: 10
96              enabled: engineOn
97              onPressed: {
98                  brakeTimer.start()
99              }
100             onReleased: {
101                 brakeTimer.stop()
102             }
103         }
```

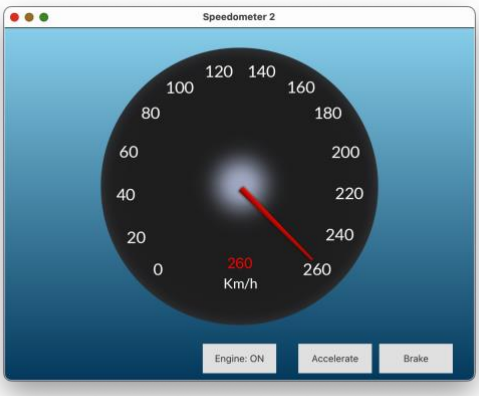
```
104
105         // Acceleration: Increment speed by 2 every 100 milliseconds
106         Timer {
107             id: accelerateTimer
108             interval: 100
109             repeat: true
110             onTriggered: {
111                 if (engineOn && currentSpeed < 260) {
112                     currentSpeed += 2
113                 }
114             }
115         }
116
117         // Braking: Decrement speed by 3 every 50 milliseconds
118         Timer {
119             id: brakeTimer
120             interval: 50
121             repeat: true
122             onTriggered: {
123                 if (engineOn && currentSpeed > 0) {
124                     currentSpeed -= 3
125                 }
126             }
127         }
128
129         // Coasting: Decrement speed by 1 every 150 milliseconds
130         Timer {
131             id: coastTimer
132             interval: 150
133             repeat: true
134             onTriggered: {
135                 if (currentSpeed > 0) {
136                     currentSpeed -= 1
137                 }
138             }
139         }
140     }
141 }
```

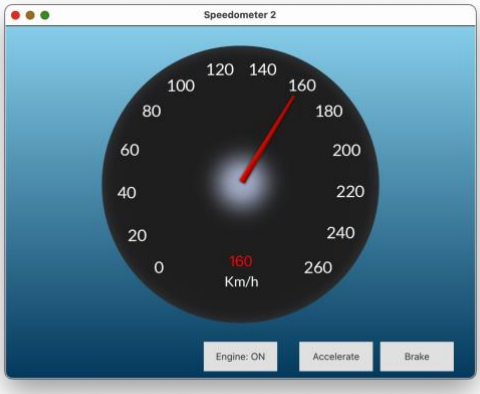
4.5.3 Live Preview

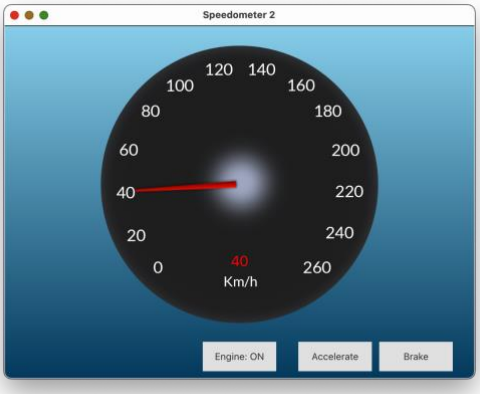












5 Week exercises

5.1 Setting Up a Basic ListView

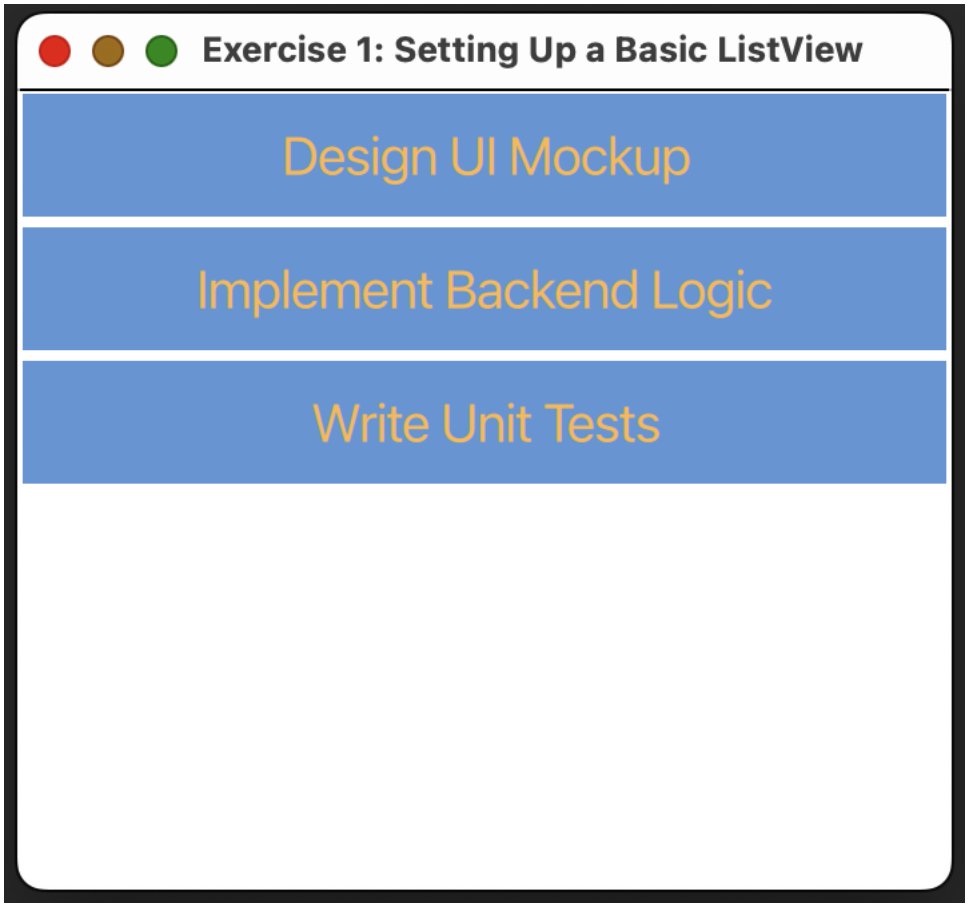
Objective: Create a QML application that displays a static list of items using ListView and ListModel.

- Set up an ApplicationWindow.
- Inside, create a ListModel with at least three ListElement items, each with a property name.
- Add a ListView that displays the names of these items using a simple delegate with a Text element.

BasicListView.qml

```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 350
    height: 300
    title: "Exercise 1: Setting Up a Basic ListView"

    ListModel {
        id: nameModel
        ListElement { name: "Design UI Mockup" }
        ListElement { name: "Implement Backend Logic" }
        ListElement { name: "Write Unit Tests" }
    }

    ListView {
        id: listView
        anchors.fill: parent
        model: nameModel
        delegate: Rectangle {
            width: listView.width
            height: 50
            color: '#6895D2'

            border.color: 'white'
            border.width: 2

            Text {
                anchors.centerIn: parent
                text: model.name
                color: '#F3B95F'
                font.pointSize: 20
            }
        }
        clip: true
    }
}
```

This is simple application displays a list of tasks in a window. The tasks listed are "Design UI Mockup", "Implement Backend Logic", and "Write Unit Tests". Each task is displayed in a colored rectangle, with the text centered and displayed in a contrasting color.



5.2 Adding Dynamic Interaction

Objective: Extend the basic ListView by adding functionality to dynamically add items to the list through user input.

- Add a TextField where users can type in the name of a new item.
- Implement a Button labeled "Add Item". When clicked, a new item should be added to the ListModel using the text from the TextField.
- Ensure the TextField is cleared after adding an item.

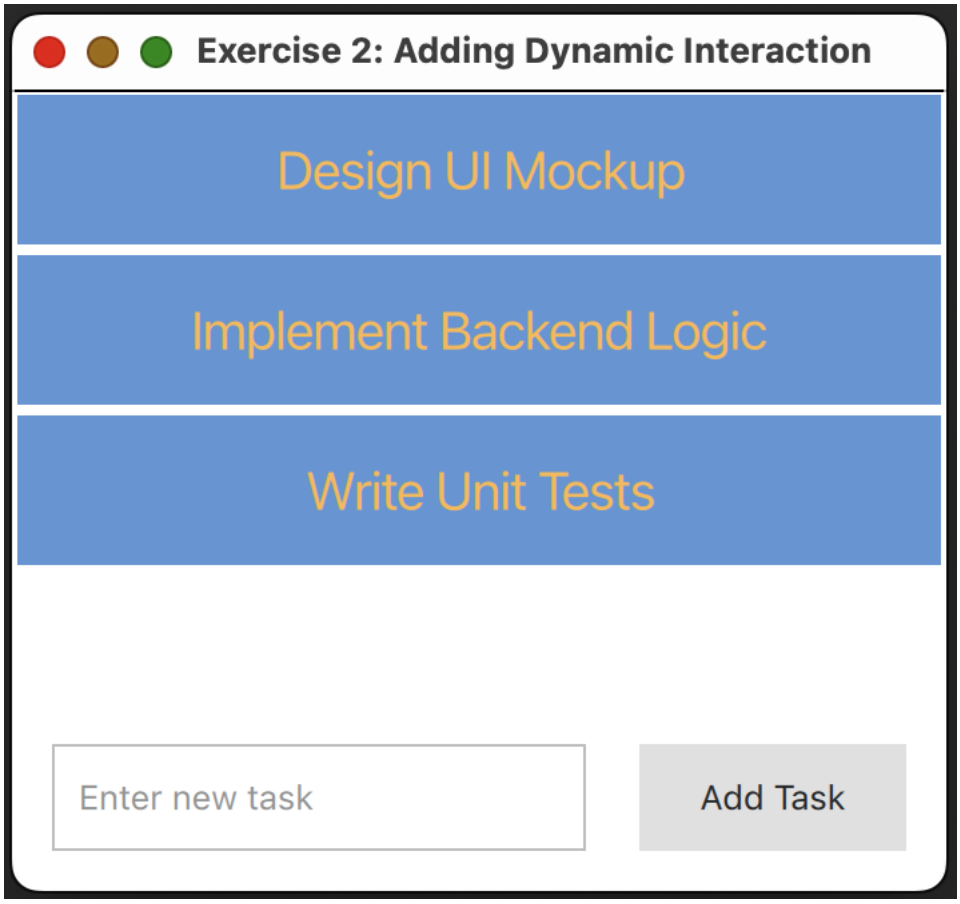
ExtendedListView.qml

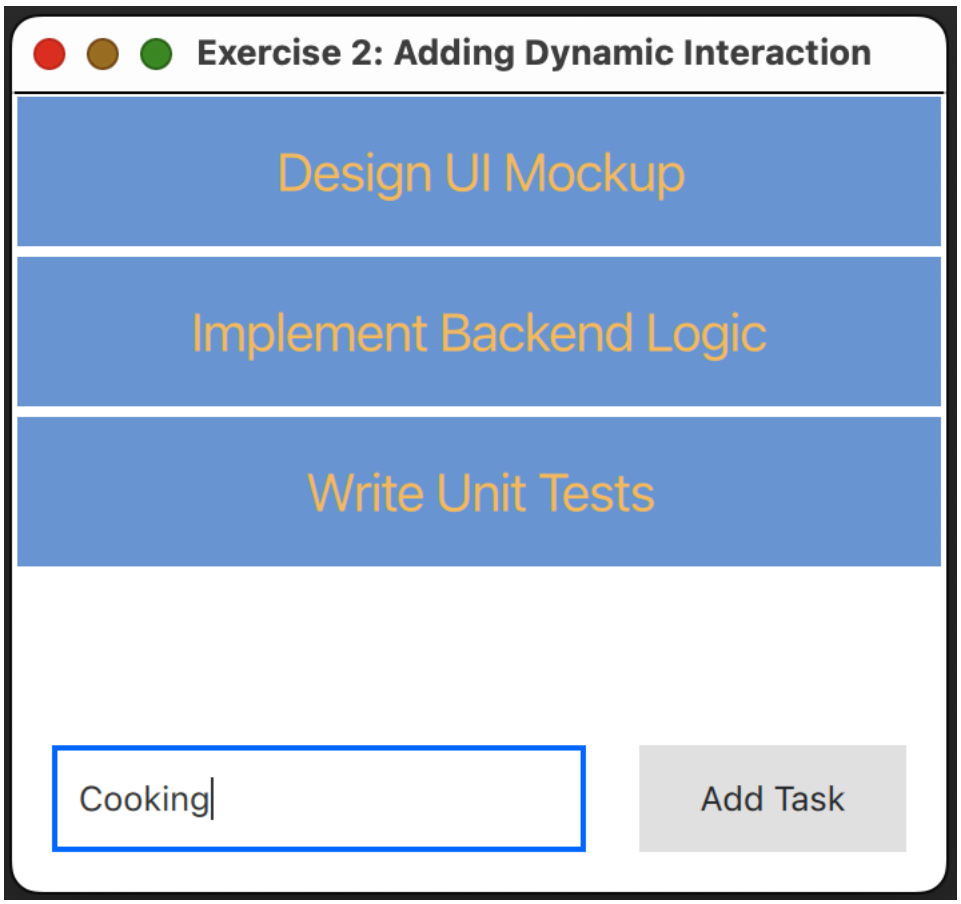
```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 350
    height: 300
    title: "Exercise 2: Adding Dynamic Interaction"

    ListModel {
        id: nameModel
        ListElement { name: "Design UI Mockup" }
        ListElement { name: "Implement Backend Logic" }
        ListElement { name: "Write Unit Tests" }
    }

    ListView {
        id: listView
        anchors.fill: parent
        model: nameModel
        delegate: Rectangle {
            width: listView.width
            height: 60
            color: '#6895D2'

            border.color: 'white'
            border.width: 2

            Text {
                anchors.centerIn: parent
                text: model.name
                color: '#F3B95F'
                font.pointSize: 20
            }
        }
        clip: true
    }

    Row {
        spacing: 20
        anchors.bottom: parent.bottom
        anchors.left: parent.left
        anchors.right: parent.right
        anchors.margins: 15

        TextField {
            id: textField
            placeholderText: "Enter new task"
            width: 200
        }

        Button {
            id: addButton
            text: "Add Task"
            width: 100

            onClicked: {
                if (textField.text.length > 0) {
                    nameModel.append({name: textField.text});
                    textField.text = "";
                }
            }
        }
    }
}
```

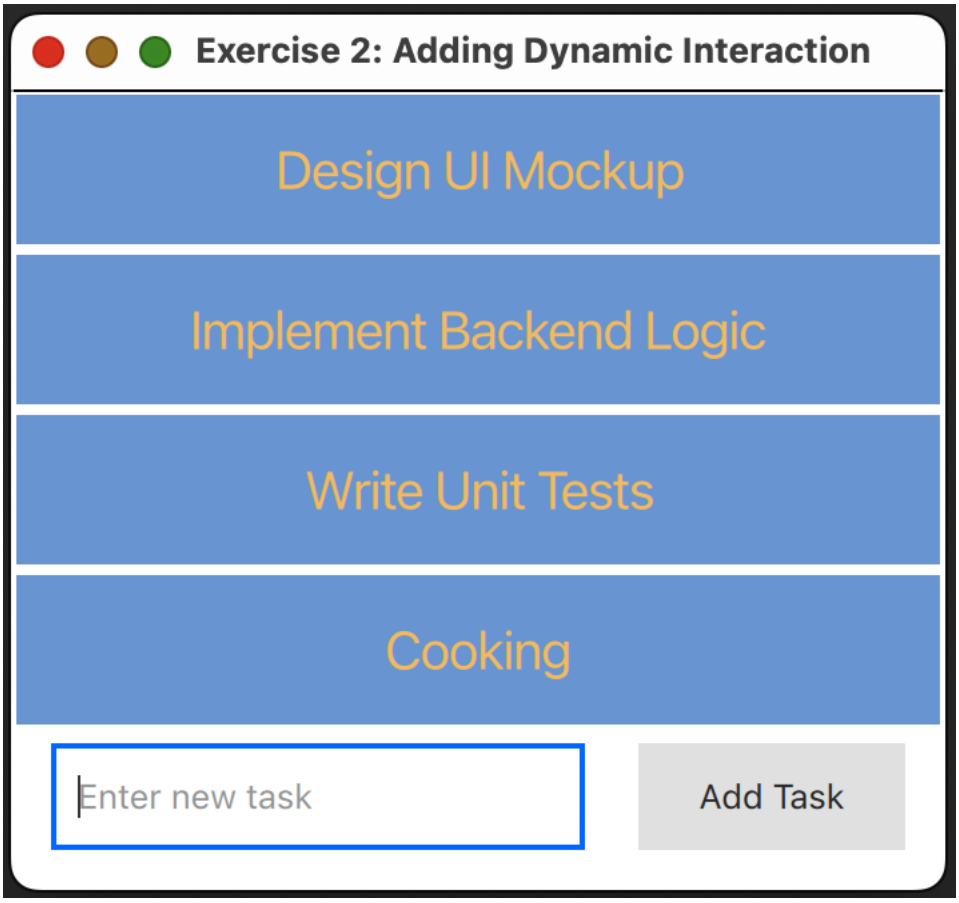
The ListModel is used to store the tasks. Each task is a ListElement with a name property. The tasks "Design UI Mockup", "Implement Backend Logic", and "Write Unit Tests" are initialized in the ListModel.

The ListView is used to display the tasks. The delegate property of the ListView is set to a Rectangle with specific width and height. Inside each rectangle, a Text element displays the name of the task. The model property of the ListView is set to the ListModel.

The user interaction feature is implemented through a TextField and a Button. The TextField is used for users to input new tasks. The Button is labeled "Add Task", when clicked, it triggers the onClicked event. Inside this event, a condition checks if the text field is not empty, and if true, a new task is appended to the ListModel using the text from the TextField and the TextField is then cleared.




Exercise 2: Adding Dynamic Interaction
Design UI Mockup
Implement Backend Logic
Write Unit Tests
Cooking
Add Task


Exercise 2: Adding Dynamic Interaction
Design UI Mockup
Implement Backend Logic
Write Unit Tests
Cooking
Enter new task
Add Task

5.3 Drawing with Canvas

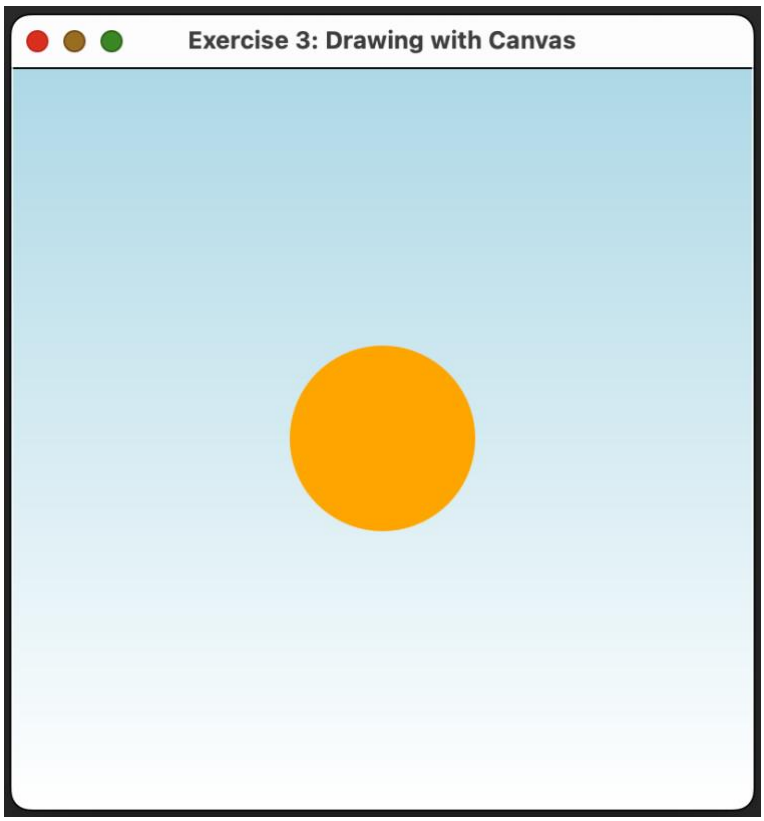
Objective: Use the Canvas element to draw a static ball.

- Set up an ApplicationWindow.
- Add a Canvas element that fills the window. Use some background image or just draw a gradient rectangle in the background.
- Implement the onPaint event to draw a circle at the center of the canvas, representing a ball.

StaticBall.qml — ApplicationWindow

```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 400
    height: 400
    title: "Exercise 3: Drawing with Canvas"

    // Background gradient
    Canvas {
        anchors.fill: parent
        onPaint: {
            var ctx = getContext("2d")
            var gradient = ctx.createLinearGradient(0, 0, 0, height)
            gradient.addColorStop(0, "lightblue")
            gradient.addColorStop(1, "white")
            ctx.fillStyle = gradient
            ctx.fillRect(0, 0, width, height)
        }
    }

    // Ball created using a Rectangle
    Rectangle {
        width: 100
        height: 100
        color: "orange"
        radius: width/2
        anchors.centerIn: parent
    }

}
```

This application displays a window with a gradient background, ranging from light blue at the top to white at the bottom, created using a Canvas element. In the center of the window, there is a static orange ball, represented by a Rectangle with a radius set to half its width, which creates a circular shape.



5.4 Implementing Animation

Objective: Animate the ball to move and bounce within the canvas boundaries.

- Define properties for the ball's position (ballPosition), velocity (dx, dy), and radius (ballRadius).
- Add a Timer that updates the ball's position based on its velocity.
- Implement boundary collision detection: reverse the ball's direction when it hits the canvas edges.
- Use requestPaint() to update the canvas drawing based on the ball's new position.

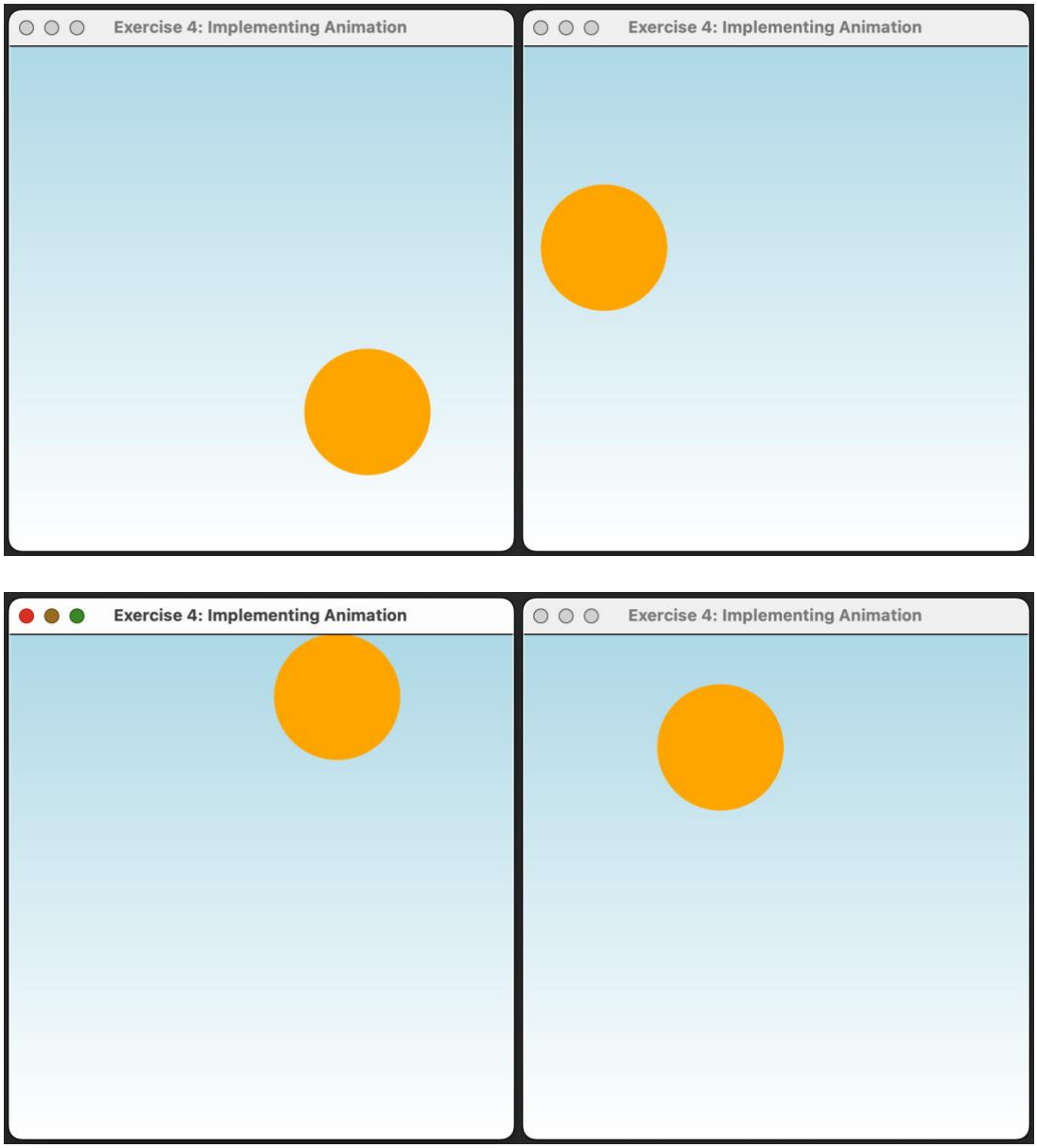
BouncingBall.qml — ApplicationWindow

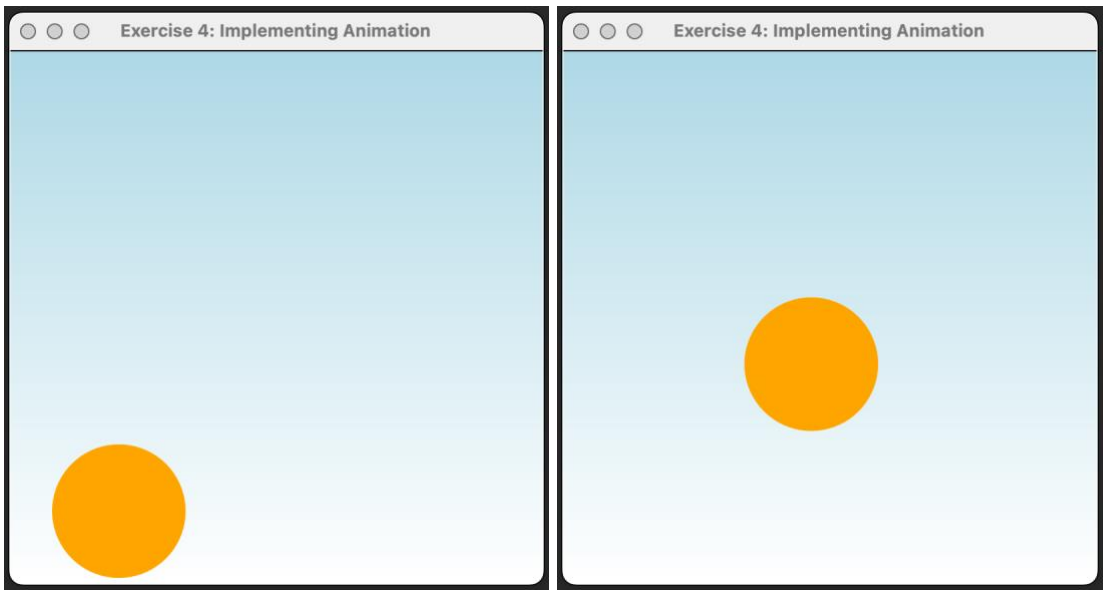
```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 400
    height: 400
    title: "Exercise 4: Implementing Animation"

    property real ballRadius: 50
    property real dx: 2
    property real dy: 5

    // Background gradient
    Canvas {
        id: canvas
        anchors.fill: parent
        onPaint: {
            var ctx = getContext("2d")
            var gradient = ctx.createLinearGradient(0, 0, 0, height)
            gradient.addColorStop(0, "lightblue")
            gradient.addColorStop(1, "white")
            ctx.fillStyle = gradient
            ctx.fillRect(0, 0, width, height)
        }
        z: -1  // Ensure the canvas is in the background
    }

    // Ball created using a Rectangle
    Rectangle {
        id: ball
        width: ballRadius * 2
        height: ballRadius * 2
        color: "orange"
        radius: ballRadius
        x: (parent.width - width) / 2  // Center the ball initially
        y: (parent.height - height) / 2
    }

    Timer {
        interval: 16  // Approximately 60 frames per second
        repeat: true
        running: true

        onTriggered: {
            // Update ball position
            ball.x += dx
            ball.y += dy

            // Collision detection with canvas boundaries
            if ((ball.x + ball.width >= canvas.width) || (ball.x <= 0)) {
                dx *= -1
            }
            if ((ball.y + ball.height >= canvas.height) || (ball.y <= 0)) {
                dy *= -1
            }
        }
    }
}
```

This interactive QML application creates an animation of a bouncing ball. The ball, represented by an orange Rectangle, moves across a Canvas with a gradient

background. The movement is controlled by a Timer that regularly updates the ball's position according to its velocity (dx, dy). Collision detection is implemented to reverse the ball's direction when it hits the edges of the Canvas, creating a bouncing effect.

The bouncing ball's movement might seem random at first, but it's actually determined by the initial velocity and the bounce mechanics. The ball's direction changes whenever it hits the edges of the Canvas, giving the illusion of randomness.





5.5 Qt/QML Architecture

Objective: To understand the basic principles on Qt architectrure. Just short written questions and answers of very basic principles (no deep dive).

- Provided you have your own Linux (or custom OS) based device and you need to provide a fluid GUI for the users. How can you get Qt on your device and how can it be ported in your device (give just and overall principle of Qt stack).
- How can you connect your C++ code and QML code together (this is quite normal case as QML is used for GUI and C/C++ world is typically under the hood.
- How can you make your Qt/QML app to work on Android?

5.5.1 Porting Qt to Linux or Custom OS Devices

To get Qt on a Linux or custom OS device, typically, the Qt SDK needs to be downloaded and installed. It is available for various platforms. Qt applications can be ported to the device by cross-compiling the Qt libraries for the target device architecture, deploying the application to the device, and running it there.

The Qt stack, based on the "write once, compile anywhere" principle, allows developers to create cross-platform C++ applications without changing the source code. It's divided into modules like QtCore, QtGui, QtWidgets, and QtQuick, each offering various features. QtCore manages non-graphical functions, QtGui handles system integration and graphics, and QtWidgets offers UI elements for desktop interfaces.

QtQuick is essential for QML development, facilitating seamless C++ and QML integration for flexible UI creation. The stack also includes the Qt

Creator IDE, internationalization utilities, a help system, and more, creating a comprehensive developer ecosystem.

5.5.2 Connecting C++ Code and QML Code

To connect C++ code and QML code together, the Qt Meta-Object Language (QML) can be used. It allows seamless integration of GUI designs with backend logic. C++ objects can be exposed to QML, C++ methods can be called from QML, signals and slots can be handled, and data can be shared between the two.

5.5.3 Making a Qt/QML App Work on Android

To make a Qt/QML app work on Android, the Qt for Android port, which includes libraries, tools, and a plugin for Qt Creator that allows the design, build, and deployment of Qt apps for Android devices, could be used. The application would then need to be compiled for the Android platform, packaged as an APK file, and installed on the device.

6 Project - Coffee Machine Interface

6.1 Development Plan

Screen Navigation: The application is planned to have three screens for users to navigate through - a home screen displaying all the available drinks, a customization screen where users can adjust the temperature and size of their drink in milliliters, and a waiting screen for the preparation of the drink.

Favorite Selection: Users will have the ability to mark one of the drinks as a favorite. They can choose the size, strength, and milk-to-coffee ratio of this drink and save it for future orders.

6.2 Development Resources

Image Sources: The icons used in this application were sourced from Flaticon.

Image Conversion: Various tools were used to convert the images to PNG format for use in the application.

6.3 Challenges

The most time-consuming parts of the development were finding proper coffee icons and deciding on the color combination. Additionally, learning new things like navigation, arc animation, radio buttons, sliders, and GridView presented challenges. However, after some dedicated study and practice, they became manageable and doable.

6.4 Application overview

This application is a user-friendly coffee selection tool designed with Qt Quick. The application offers a variety of features to create a personalized coffee experience.

Users can choose their preferred coffee from a wide array of options. The application further allows customization of the coffee's strength, temperature, and size to suit individual taste preferences. Users can also save and access their favorite coffee selections for quick and easy ordering. The application features intuitive navigation, enabling smooth transitions between different screens.

In addition to these features, the application also enhances the user experience with visually pleasing graphics and animations, and an organized, aesthetically appealing layout. With its straightforward usage, users can easily select their coffee, customize it to their liking, and navigate through the application with ease.

6.5 Features

- **Coffee Selection:** Users can choose from a variety of coffee types.
- **Customization:** Options to customize coffee strength, temperature, and size.
- **Favorites:** Users can save and access their favorite coffee selections.
- **Intuitive Navigation:** Easy-to-use navigation between different screens.

6.6 Implementation Details

6.6.1 Custom Coffee Component

- **Interactive Coffee Tiles:** Each coffee option in the GridView is represented by a custom Coffee component. This component combines an image (icon) and a text description, creating an engaging and informative selection tile.
    - **User Interaction:** The component is designed to emit a coffeeSelected signal when clicked, allowing the app to respond to user selections seamlessly.

6.6.2 GridView and ListModel

- **Coffee Selection Grid:** The app features a GridView to display various coffee options. Each grid cell represents a different coffee type, such as Espresso, Americano, Cappuccino, etc.
    - **Dynamic Data Handling:** A ListModel is used to define the coffee types, where each ListElement contains the name and icon of the coffee, facilitating an adaptable and scalable coffee menu.

6.6.3 Layout Design

- **Organized UI Layout:** The app's layout is meticulously designed using Column and Row elements to arrange the UI components in a structured and aesthetically pleasing manner.

6.6.4 User Preferences and Customization

- **Temperature and Size Selection:** Utilizing Sliders and RadioButtons, the app offers users the ability to customize their coffee temperature and size, enhancing the personalization of the coffee experience.

6.6.5 Canvas for Dynamic Graphics

- **Visual Enhancements and Animations:** In the Processing Screen of the app, a Canvas element is used for rendering custom graphics, enriching the user interface with visually pleasing elements.
    - **Dynamic Graphics Rendering:** The Canvas is designed to draw a background circle and a progress arc that animates over time. It calculates the center coordinates and radius of the circle and uses

them to draw the circle and arc. The canvas is cleared before each drawing, ensuring that the progress arc is refreshed each time.

6.6.6 Animation and Timer

- **Arc Animation:** A NumberAnimation is used on the angle property of the Canvas to animate the progress arc. The animation starts from 0 and ends a little more than 360 for a complete circle, with a duration of 20 seconds. The animation runs immediately when the processing screen is viewed.
- **Timed Actions:** In the processing screen, a timer is set to trigger after 20 seconds (same duration as the progress arc animation), at which point it pops the top item off the stack, thus transitioning away from the processing screen.

6.6.7 Signals and Navigation

- **Signals:** Extensively used for component communication, especially for triggering actions like selecting a coffee or adjusting preferences.
- **StackView for Screen Management:** The StackView is employed to manage navigation between various screens like the main menu, coffee details, and customization screens. This approach allows for smooth transitions and a clear navigation flow.

6.7 Usage

- **Select a Coffee:** From the main menu, click on your desired coffee type.
- **Customize Your Coffee:** Use sliders and radio buttons to customize your drink.
- **Favorite Selection:** Access and customize your favorite drinks.
- **Navigate:** Use 'Back' and 'Confirm' buttons to navigate through the app.

6.8 Code

6.8.1 App.qml

App.qml — StackView

```
import QtQuick 2.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 640
    height: 350
    title: "Coffee App"

    // StackView for managing the navigation between screens
    StackView {
        id: stackView
        anchors.fill: parent

        // Set the initial item (screen) to be displayed in the StackView
        initialItem: MenuScreen {
            // Define a function to be called when a coffee type is selected
            onCoffeeSelected: function(coffeeName) {
                // Check if the selected coffee type is 'Favorite'
                if (coffeeName === "Favorite") {
                    // Push the FavoriteScreen onto the stack with stackView reference
                    stackView.push(Qt.resolvedUrl("FavoriteScreen.qml"), {"stackView": stackView});
                } else {
                    // Push the DetailsScreen onto the stack with the selected coffee name and stackView reference
                    stackView.push(Qt.resolvedUrl("DetailsScreen.qml"), {"coffeeName": coffeeName, "stackView": stackView});
                }
            }
        }
    }
}
```

6.8.2 Coffee.qml

Coffee.qml — MouseArea

```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

// Define a custom rectangle component
Rectangle {
    width: 110
    height: 140
    radius: 20

    color: "#BD8769"

    // Declare properties that can be bound to external values
    property alias imageSource: clickableImage.source
    property alias description: description.text

    // Declare a custom signal that will be emitted when the image is clicked
    signal imageClicked(string coffeeName)

    // Column layout for the image and text
    Column {
        width: parent.width*0.8
        height: parent.height*0.8
        anchors.centerIn: parent

        Image {
            id: clickableImage
            source: "images/espresso.png"
            width: parent.width
            fillMode: Image.PreserveAspectFit
        }

        Text {
            id: description
            text: "Espresso"
            color: "white"
            font.pixelSize: 20
            width: parent.width
            horizontalAlignment: Text.AlignHCenter
        }
    }

    // Mouse area to detect clicks
    MouseArea {
        anchors.fill: parent
        onClicked: {
            console.log("Image of " + description.text + " clicked!")

            // Emit the custom signal with the description text
            parent.imageClicked(description.text);
        }
    }
}
```

6.8.3 MenuScreen.qml

MenuScreen.qml | GridView

```
Item {
    visible: true
    width: 640
    height: 350

    // Custom signal to notify when a coffee type is selected
    signal coffeeSelected(string coffeeName)

    // Background rectangle with gradient
    Rectangle {
        anchors.fill: parent
        gradient: Gradient {
            GradientStop { position: 0.0; color: "#f5e0c3" } // Light creamy color
            GradientStop { position: 0.5; color: "#b08968" } // Medium coffee color
            GradientStop { position: 1.0; color: "#3b2b1e" } // Dark espresso color
        }
    }

    // GridView to display coffee options
    GridView {
        width: cellWidth * 4
        height: cellHeight * 2
        cellWidth: 110 + 20
        cellHeight: 140 + 20
        model: coffeeModel
        anchors.centerIn: parent

        delegate: Item {
            width: 130
            height: 160

            // Custom coffee component
            Coffee {
                id: coffeeComponent
                description: name // Bind the description to the coffee name
                imageSource: icon // Bind the image source to the coffee icon

                anchors.centerIn: parent // Center the coffee component in the delegate item

                // Handler for when an image in the coffee component is clicked
                onImageClicked: {
                    coffeeSelected(name) // Emit the coffeeSelected signal with the name of the coffee
                }
            }
        }
    }

    // Model to hold data about different coffee types
    ListModel {
        id: coffeeModel
        // Define each coffee type with a name and icon
        ListElement { name: "Espresso"; icon: "images/espresso.png" }
        ListElement { name: "Americano"; icon: "images/americano.png" }
        ListElement { name: "Cappuccino"; icon: "images/cappuccino.png" }
        ListElement { name: "Latte"; icon: "images/latte.png" }
        ListElement { name: "Mocha"; icon: "images/mocha.png" }
        ListElement { name: "Chocolate"; icon: "images/chocolate.png" }
        ListElement { name: "Milk"; icon: "images/milk.png" }
        ListElement { name: "Favorite"; icon: "images/favorite.png" }
    }
}
```

6.8.4 DetailsScreen.qml

DetailsScreen.qml — Slider

```qml
import QtQuick 2.15
import QtQuick.Controls 2.15
import QtQuick.Layouts

Item {
    visible: true
    width: 640
    height: 350

    property var stackView: null

    Image {
        id: backgroundImage
        source: "images/coffee_break.png"
        width: parent.width
        fillMode: Image.PreserveAspectFit // Keep the aspect ratio of the image
    }

    // Main column layout for the interface elements
    Column {
        spacing: 20
        anchors.verticalCenter: parent.verticalCenter
        anchors.left: parent.left
        anchors.leftMargin: 50

        // Sub-column for coffee temperature selection
        Column {
            // Label displaying the selected temperature
            Label {
                id: temperatureLabel
                font.pixelSize: 20
                text: "Temperature: " + temperature.value + "°C"
                color: "#353637"
                anchors.left: parent.left
                anchors.leftMargin: 5
            }

            // Slider for selecting the coffee temperature
            Slider {
                id: temperature
                width: 200 // Set width
                from: 0 // Minimum temperature value
                to: 100 // Maximum temperature value
                stepSize: 1 // Increment step
                value: 95  // Set initial value
                onValueChanged: {
                    temperatureLabel.text = "Temperature: " + value + "°C" // Update label text on value change
                }
            }
        }


        // Size selection using Radio Buttons
        Column {
            RadioButton { id: smallSize; text: "Small (240 ml)"; font.pixelSize: 20; checked: true }
            RadioButton { id: mediumSize; text: "Medium (355 ml)"; font.pixelSize: 20 }
            RadioButton { id: largeSize; text: "Large (475 ml)"; font.pixelSize: 20 }
        }

```

DetailsScreen.qml* | Row

```
60      Row {
61          spacing: 20
62          anchors.left: parent.left
63          anchors.leftMargin: 10
64
65          // Back button to return to the previous screen
66          Button {
67              id: backButton
68              text: "Back"
69              font.pixelSize: 22
70
71              onClicked: {
72                  stackView.pop(); // Pop the current view from the stack view
73              }
74
75              // Customizing the button appearance
76              background: Rectangle {
77                  color: "#79B82B"
78                  radius: 10 // Corner radius
79              }
80          }
81
82          // Confirm button to finalize the selection
83          Button {
84              text: "Confirm"
85              font.pixelSize: 22
86
87              onClicked: {
88                  // Logic to handle the selection
89                  console.log(temperatureLabel.text)
90
91                  var selectedSize = "";
92                  if (smallSize.checked) {
93                      selectedSize = "Small";
94                  } else if (mediumSize.checked) {
95                      selectedSize = "Medium";
96                  } else if (largeSize.checked) {
97                      selectedSize = "Large";
98                  }
99
100                 console.log("Selected size: " + selectedSize);
101
102                 // Push ProcessingScreen and pass the stackView
103                 stackView.push(Qt.resolvedUrl("ProcessingScreen.qml"), {"stackView": stackView});
104             }
105
106             // Customizing the button appearance
107             background: Rectangle {
108                 color: "#79B82B"
109                 radius: 10 // Corner radius
110             }
111         }
112     }
113 }
114 }
```

6.8.5 FavoriteScreen.qml

FavoriteScreen.qml — Gradient

```qml
import QtQuick 2.15
import QtQuick.Controls 2.15

Item {
    width: 640
    height: 350

    property var stackView: null // Property to hold a reference to the stackView

    // Background rectangle with a gradient
    Rectangle {
        anchors.fill: parent
        gradient: Gradient {
            GradientStop { position: 0.0; color: "#f5e0c3" } // Light creamy color
            GradientStop { position: 0.5; color: "#b08968" } // Medium coffee color
            GradientStop { position: 1.0; color: "#3b2b1e" } // Dark espresso color
        }
    }

    // Column layout to organize the controls vertically
    Column {
        spacing: 20
        anchors.verticalCenter: parent.verticalCenter
        anchors.left: parent.left
        anchors.leftMargin: 50

        Column {
            // Coffee Strength Selection
            Label {
                id: strengthLabel
                font.pixelSize: 20
                text: "Milk to coffee ratio: " + strength.value
                color: "#353637"
                anchors.left: parent.left
                anchors.leftMargin: 5
            }
            Slider {
                id: strength
                width: 200
                from: 0
                to: 4
                stepSize: 0.5
                value: 0.5  // Set initial value
                onValueChanged: {
                    // Update the label text when the slider value changes
                    strengthLabel.text = "Milk to coffee ratio: " + value
                }
            }
        }

```

FavoriteScreen.qml — Label

```
51      Column {
52          // Coffee Temperature Selection
53          Label {
54              id: temperatureLabel
55              font.pixelSize: 20
56              text: "Temperature: " + temperature.value + "°C"
57              color: "#353637"
58              anchors.left: parent.left
59              anchors.leftMargin: 5
60          }
61          Slider {
62              id: temperature
63              width: 200
64              from: 0
65              to: 100
66              stepSize: 1
67              value: 95  // Set initial value
68              onValueChanged: {
69                  // Update the label text when the slider value changes
70                  temperatureLabel.text = "Temperature: " + value + "°C"
71              }
72          }
73      }
74
75      Column {
76          // Coffee Size Selection
77          Label {
78              id: sizeLabel
79              font.pixelSize: 20
80              text: "Size: " + size.value + " ml"
81              color: "#353637"
82              anchors.left: parent.left
83              anchors.leftMargin: 5
84          }
85          Slider {
86              id: size
87              width: 200
88              from: 200
89              to: 500
90              stepSize: 1
91              value: 355  // Set initial value
92              onValueChanged: {
93                  // Update the label text when the slider value changes
94                  sizeLabel.text = "Size: " + value + " ml"
95              }
96          }
97      }
98
```

FavoriteScreen.qml — onClicked

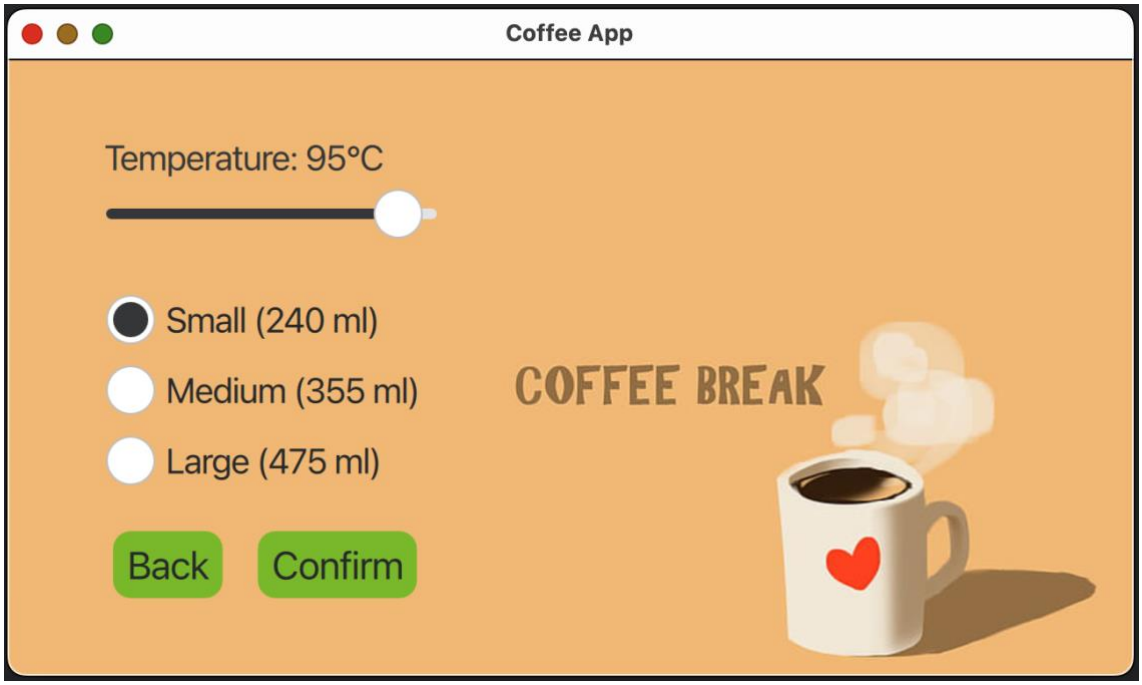
```
99          // Row layout for buttons
100         Row {
101             spacing: 20 // Space between each button
102             anchors.left: parent.left
103             anchors.leftMargin: 10
104
105             // Back Button
106             Button {
107                 id: backButton
108                 text: "Back"
109                 font.pixelSize: 22
110
111                 onClicked: {
112                     // Pop the current view from the stackView
113                     stackView.pop();
114                 }
115
116                 // Customizing the button appearance
117                 background: Rectangle {
118                     color: "#79B82B"
119                     radius: 10 // Corner radius
120                 }
121             }
122
123             // Confirm Button
124             Button {
125                 text: "Confirm"
126                 font.pixelSize: 22
127
128                 onClicked: {
129                     // Log the selected options and push a new screen
130                     console.log(strengthLabel.text)
131                     console.log(temperatureLabel.text)
132                     console.log(sizeLabel.text)
133
134                     // Push ProcessingScreen and pass the stackView
135                     stackView.push(Qt.resolvedUrl("ProcessingScreen.qml"), {"stackView": stackView});
136                 }
137
138                 // Customizing the button appearance
139                 background: Rectangle {
140                     color: "#79B82B"
141                     radius: 10 // Corner radius
142                 }
143             }
144         }
145     }
146
147
148     Image {
149         id: clickableImage
150         source: "images/heart.png"
151         height: parent.height*0.8
152         fillMode: Image.PreserveAspectFit
153
154         anchors.right: parent.right
155         anchors.rightMargin: 20
156         anchors.verticalCenter: parent.verticalCenter
157     }
158 }
```

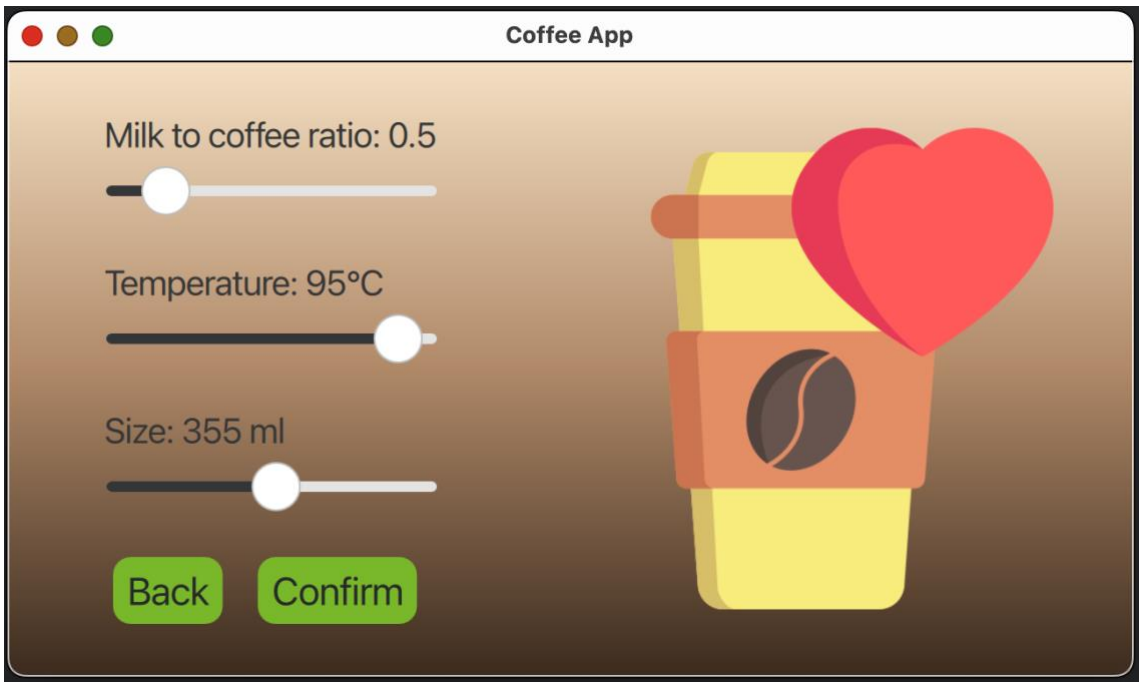
6.8.6 ProcessingScreen.qml

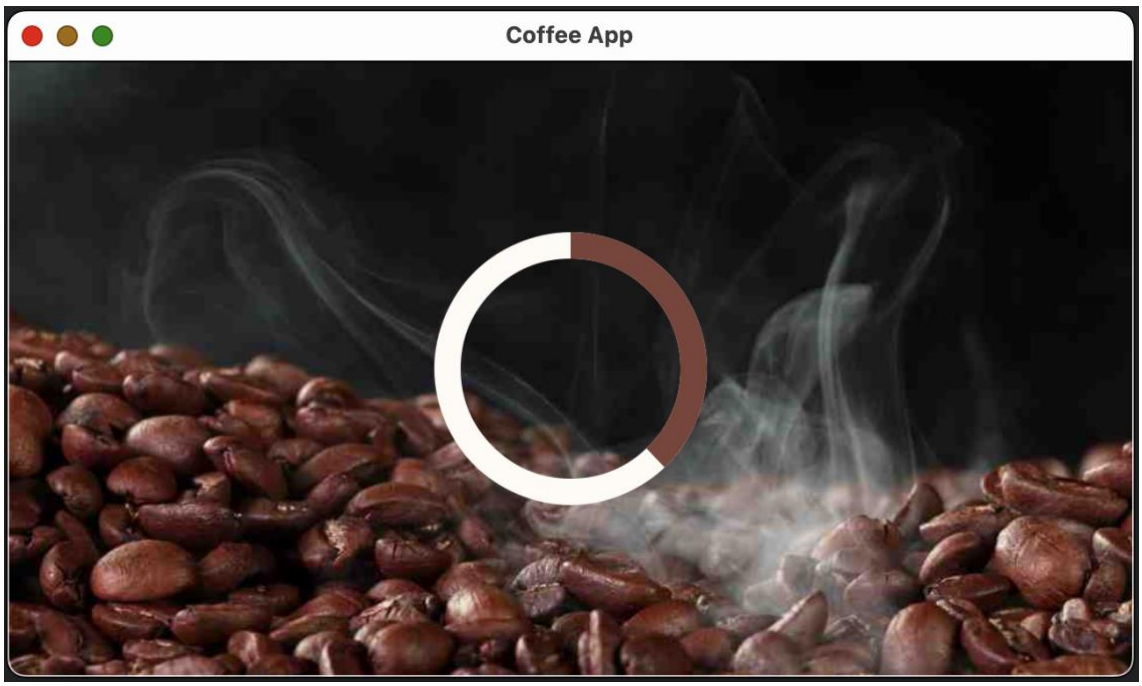
ProcessingScreen.qml — onAngleChanged

```qml
import QtQuick 2.15

Item {
    width: 640
    height: 350

    property var stackView: null

    Image {
        id: clickableImage
        source: "images/beans.png"
        width: parent.width
        fillMode: Image.PreserveAspectFit // Keep the aspect ratio of the image
    }

    Canvas {
        id: canvas
        width: 160
        height: 160
        anchors.centerIn: parent

        property real angle: 0 // Property to keep track of the angle for the arc

        onAngleChanged: requestPaint() // Request to repaint whenever angle changes

        onPaint: {
            var ctx = getContext("2d");
            var x = width / 2; // Calculate the center x-coordinate
            var y = height / 2; // Calculate the center y-coordinate
            var radius = Math.min(x, y) - 10; // Calculate the radius of the circle
            var startAngle = -Math.PI / 2; // Starting angle for the arc (top of the circle)
            var endAngle = startAngle + angle * Math.PI / 180; // Ending angle for the arc

            // Clear the canvas before drawing
            ctx.clearRect(0, 0, width, height);

            // Draw background circle
            ctx.beginPath();
            ctx.arc(x, y, radius, 0, 2 * Math.PI, false);
            ctx.lineWidth = 15;
            ctx.strokeStyle = "#FEFBF6"; // Light color for the background circle
            ctx.stroke();

            // Draw progress arc
            ctx.beginPath();
            ctx.arc(x, y, radius, startAngle, endAngle, false);
            ctx.lineWidth = 15;
            ctx.strokeStyle = "#76453B"; // Dark color for the progress arc
            ctx.stroke();
        }

        // Arc animation
        NumberAnimation on angle {
            from: 0 // Start angle
            to: 360 + 10 // End angle (a little more than 360 for a complete circle)
            duration: 20000 // Duration of the animation: 20 seconds
            running: true // Start the animation immediately
        }
    }

    Timer {
        id: timer
        interval: 20000 // 20 seconds
        running: true
        repeat: false
        onTriggered: {
            stackView.pop(); // Pop the top item off the stack when the timer triggers
        }
    }
}
```

6.9 Screens of design







6.10 Ideas for further development

- Incorporate additional customization features, such as options for adding flavored syrups or choosing different milk types (skim, soy, almond, etc.).
- Enhance the user interface with user feedback mechanisms to improve the overall user experience.
- Incorporate user reviews and ratings for different coffee types.
- The processing time should vary depending on the coffee options, temperature, and size.
- Make the design more classic.
- Integrate a feature that tracks the user's coffee consumption history to provide valuable data for improving the app and personalizing user recommendations.

Sources used with exercises

https://www.qt.io/download-open-source
https://doc.qt.io/qtdesignstudio/qtdesignstudio-loginui1-example.html
https://doc.qt.io/qtdesignstudio/studio-terms.html

https://doc.qt.io/qt-5/qml-qtgraphicaleffects-lineargradient.html
https://doc.qt.io/qt-6/qgradient.html
https://doc.qt.io/qt-6/qtquick-positioning-anchors.html
https://doc.qt.io/qt-6/qtquick-positioning-topic.html

https://doc.qt.io/qt-6/qtquick-positioning-layouts.html
https://doc.qt.io/qt-6/qml-qtquick-textedit.html
https://doc.qt.io/qt-6/qtqml-syntax-objectattributes.html
https://doc.qt.io/qt-6/qtnetwork-http-example.html
https://doc.qt.io/qtdesignstudio/creator-how-to-view-output.html
https://openweathermap.org/weather-conditions

https://doc.qt.io/qt-6/qtquick-statesanimations-states.html
https://doc.qt.io/qt-6/qml-qtqml-timer.html
https://doc.qt.io/qt-6/qml-qtquick-transition.html

https://doc.qt.io/qt-6/qml-qtquick-listview.html
https://doc.qt.io/qt-6/qml-qtquick-canvas.html
https://www.qt.io/product/mobile-app-development/
https://www.qt.io/deploy

Coffee Machine Example
https://doc.qt.io/qt-5/qtdoc-demos-coffee-cupform-ui-qml.html
Qt Quick Controls 2 - Coffee machine demo
https://youtube.com/shorts/a56XtlGySVM?si=kY_5xZtxGRHIirLK
Saeco Xelsis Connected Coffee Machine | QT GUI
https://www.youtube.com/watch?v=HqRXZEkqOb4
Witekio's Touchscreen Coffee Machine UI Built with Qt Quick {showcase}
https://www.youtube.com/watch?v=9Ied3Dpb_Kg
Built with Qt: Qt Quick Coffee machine demo, QtWS17
https://www.youtube.com/watch?v=8xm9t7SaZSI&list=PL661F1B899D92A4E4&index=38
https://doc.qt.io/qt-6/qml-qtquick-gridview.html
https://doc.qt.io/qt-6/qml-qtquick-controls-radiobutton.html
https://doc.qt.io/qt-5/qml-qtquick-controls2-stackview.html
AI contributions
https://chat.openai.com/share/fe02532f-881b-4067-90aa-7bb66e39ea13
https://chat.openai.com/share/999f9142-2f71-432d-8860-7d701b0c8de5
https://chat.openai.com/share/5726f2f8-9a5b-4ba2-8c16-8ff57ccfe18a
https://chat.openai.com/share/095d3ba1-0f1b-4038-a81e-cbbd57a744b3
https://chat.openai.com/share/f140cc3b-c3f0-47e8-a4c2-2822ae65efac