helloQt ist die Qt-Entwicklungseinheit
der Bergx2 GmbH

Fürstenstr. 15
80333 München
Deutschland

info@helloqt.de
www.helloqt.de

Implementierung einer timerbasierten Aktion in QML mit Qt

11. Februar 2025 von Lucas Moreira de Oliveira

In unserer Qt-Anwendung verwenden wir die QML-Timer-Komponente, um Digital Signage zu verwalten und die Cursoraktivierung basierend auf der Benutzeraktivität zu steuern. Diese Komponente ermöglicht es Entwicklern, Aktionen nach einer Verzögerung oder in regelmäßigen Abständen zu planen.

In diesem Blogbeitrag zeigen wir, wie man die Timer-Komponente verwendet, um den Cursor in einer Digital Signage-Anwendung, die wir bei der Begx2 GmbH entwickeln, automatisch auszublenden. Das Ziel besteht darin, den Cursor nach 15 Sekunden Inaktivität auszublenden, sodass der Bildschirm sauber und frei von Ablenkungen bleibt.

Grundlegendes zum QML-Timer

Das Timer-Element in QML bietet eine einfache Möglichkeit, zeitgesteuerte Ereignisse zu verarbeiten. Es kann so konfiguriert werden, dass es einmal nach einer Verzögerung oder wiederholt in festgelegten Intervallen ausgelöst wird. Hier ist ein kurzer Blick auf seine wichtigsten Eigenschaften:

- Intervall: Gibt die Dauer in Millisekunden an, bevor der Timer ausgelöst wird.
- Läuft: Ein Boolescher Wert, der bestimmt, ob der Timer aktiv ist.
- Wiederholung: Wenn wahr, läuft der Timer kontinuierlich im angegebenen Intervall.
- beiAusgelöst: Der Event-Handler, der ausgeführt wird, wenn der Timer ausgelöst wird.

Implementieren der Cursor-Deaktivierung bei Inaktivität

Für unseren Digital Signage-Anwendungsfall benötigen wir einen Timer, der jedes Mal zurückgesetzt wird, wenn der Benutzer mit dem System interagiert. Wenn 15 Sekunden lang keine Aktivität erkannt wird, verschwindet der Cursor. Unten sehen Sie eine Implementierung mit QML und der MouseArea-Komponente von Qt:

```
import QtQuick
import QtQuick.Layouts
import ui as UI

Window {
    id: _window
    property real globalScale: Math.min(_window.width / 1920, _window.height /
1080)
```

```
    visible: true
    visibility: UI.AppController.isAppAlwaysFullscreen ? Window.FullScreen :
Window.Windowed
    title: qsTr("ScreenWay Media Player")
    width: Screen.width
    height: Screen.height
    color: "black"

    // 1) Properties for hiding/showing cursor
    property bool mouseHidden: true   // We want the cursor hidden by default
    property real lastMouseX: 0
    property real lastMouseY: 0

    // 2) Timer for inactivity (15 seconds)
    Timer {
        id: inactivityTimer
        interval: 15000  // 15 seconds
        repeat: false
        running: false
        onTriggered: {
            // Hide the cursor after 15s
            mouseHidden = true
        }
    }

    // 3) Full-screen MouseArea that tracks movement but does NOT accept
clicks
    //    so that the existing MouseArea for settings popup still works.
    MouseArea {
        id: fullScreenTracker
        anchors.fill: parent
        hoverEnabled: true
        acceptedButtons: Qt.NoButton
        z: 0

        // Switch between blank and arrow cursor
        cursorShape: mouseHidden ? Qt.BlankCursor : Qt.ArrowCursor

        onPositionChanged: function(event) {
            // If user moves mouse ≥ 10 pixels in any direction, show the
cursor
            var dx = event.x - lastMouseX
            var dy = event.y - lastMouseY
            if (Math.sqrt(dx * dx + dy * dy) >= 10) {
                if (mouseHidden) {
                    mouseHidden = false
                }
                // Restart the 15-second timer every time we detect
significant movement
```

```
                inactivityTimer.restart()
            }
            lastMouseX = event.x
            lastMouseY = event.y
        }
    }

    MouseArea {
        x: parent.width - width
        z: 1
        width: UI.Constants.settingsOverlayOpenAreaSize
        height: UI.Constants.settingsOverlayOpenAreaSize

        scale: _window.globalScale

        hoverEnabled: true
        cursorShape: containsMouse ? Qt.PointingHandCursor : Qt.ArrowCursor

        onClicked: {
            _settings.open()
        }
    }

    Item {
        id: _rootItem
        rotation: UI.AppController.orientation
        anchors.fill: parent

        StackLayout {
            id: _screensLayout
            anchors.fill: parent
            currentIndex: UI.AppController.currentScreen

            UI.InitialScreen { }
            UI.PairingScreen { }
            UI.LoadScreen { }
            UI.PlayerScreen { }
            UI.WaitingScreen { }
        }

        UI.SettingsPopup {
            id: _settings
            x: (_window.width - width) * 0.5
            y: (_window.height - height) * 0.5
            z: 2
            scale: _window.globalScale
            onClosed: {
                _window.mouseHidden = false
            }
```

```
        }
    }

    Component.onCompleted: {
        UI.AppController.init();
        mouseHidden = true
        inactivityTimer.start()
    }
}
```

So funktioniert es

1. Eine MouseArea-Komponente erkennt Benutzerinteraktionen wie Klicken und Drücken.

2. Jede Interaktion setzt den Timer mithilfe der Funktion resetTimer() zurück.

3. Wenn 15 Sekunden lang keine Interaktion stattfindet, ruft der onTriggered-Handler hideCursor() auf, wodurch der Cursor unsichtbar wird.

4. Jede nachfolgende Interaktion startet den Timer neu und aktiviert den Cursor erneut.

5. Kleinere Bewegungen (aufgrund von Umgebungsfaktoren) werden mithilfe der Funktion checkMovement() ignoriert, die sicherstellt, dass nur signifikante Bewegungen den Timer zurücksetzen.

Verbesserungen und Überlegungen

- Tastaturaktivität: Um diese Funktion weiter zu verfeinern, sollten Sie das Hinzufügen von Tastenereignis-Listenern in Erwägung ziehen, um den Timer bei Tastatureingaben zurückzusetzen.
- Visuelle Anzeige: Bei Bedarf kann ein subtiler Ausblendeffekt hinzugefügt werden, bevor der Cursor ausgeblendet wird.
- Benutzerkonfiguration: Ermöglichen Sie Benutzern, die Dauer der Zeitüberschreitung in den Anwendungseinstellungen zu konfigurieren.

Abschluss

Mithilfe der Timer-Komponente in QML haben wir erfolgreich einen automatischen Cursor-Deaktivierungsmechanismus implementiert, der die Benutzerfreundlichkeit einer Digital Signage-Anwendung verbessert. Dieser Ansatz kann auf verschiedene andere Anwendungsfälle ausgeweitet werden, die eine automatische Inaktivitätserkennung erfordern.

Durch die Nutzung des robusten QML-Frameworks von Qt können Sie zeitgesteuerte Ereignisse effizient verarbeiten und die Benutzerinteraktionen innerhalb Ihrer Anwendungen verbessern. Das Qt-Team von helloQt hilft Ihnen gerne bei der Implementierung Ihres Qt-Projekts.