QML Drag-and-drop

This article explains how to implement drag and drop in a QML GridView element.

16 Aug 2011

Introduction

In computer graphical user interfaces, drag-and-drop is the action of (or support for the action of) selecting a virtual object by "grabbing" it and dragging it to a different location or onto another virtual object. Since QML permits to developer to interact only with simple elements such as Rectangle, Text and so on, even more complicated QML as ListView and GridView don't support such feature at all. So, this article as been written in order to show you how to add this basic feature to your QML ItemView Elements.

The final result

In the video below, you can see the final result of this article. The video shows how an user can interact with GridView items. Items can be placed on top of other ones, which become invisible. The media player is loading...

Code

Here is the QML code shown in the video. it basically defines a model with 9 items that fill a 3x3 grid. The code is pretty simple to read and it doesn't require any further explanation, except for the container item. In fact during dragging the dragged item is re-parented to the container item in order to be translated correctly.

```
import QtQuick 1.0

Rectangle {
    width: 640
    height: 480
    color: "#222222"


    ListModel {
        id: widgetmodel
        ListElement {
            iColor: "red"
             visible: true
        }
        ListElement {
            iColor: "blue"
             iVisible: true
        }
        ListElement {
            iColor: "green"
             iVisible: true
        }
        ListElement {
             iColor: "yellow"
              iVisible: true
        }
        ListElement {
             iColor: "pink"
              iVisible: true
        }
        ListElement {
             iColor: "orange"
             iVisible: true
        }
        ListElement {
             iColor: "brown"
              iVisible: true
        }
        ListElement {
             iColor: "gray"
              iVisible: true
        }
        ListElement {
             iColor: "white"
              iVisible: true
        }
    }

    Component {
        id: widgetdelegate
        Item {
            width: grid.cellWidth;
            height: grid.cellHeight

            Rectangle {
                id: im
                state: "inactive"
                //anchors.centerIn: parent
                width: grid.cellWidth - 10;
                height: grid.cellHeight - 10

                color: iColor
                visible: iVisible

                border.color: "white"
                border.width: 0

                states: [
                    State {
                        name: "inactive";
                        when: (grid.firstIndexDrag == -1) || (grid.firstIndexDrag == index)
                        PropertyChanges { target: im; border.width: 0}
```


(C) Copyright Nokia 2013. All rights reserved.

```
                }
            ]
        }

        states: [
            State {
                name: "inDrag"
                when: index == grid.firstIndexDrag

                PropertyChanges { target: im; border.width: 5 }
                PropertyChanges { target: im; parent: container }

                PropertyChanges { target: im; anchors.centerIn: undefined }
                PropertyChanges { target: im; x: coords.mouseX - im.width/2 }
                PropertyChanges { target: im; y: coords.mouseY - im.height/2 }
            }
        ]
    }
}

GridView {
    property int firstIndexDrag: -1
    id: grid
    interactive: false // no flickable
    anchors.fill: parent
    cellWidth: parent.width / 3;
    cellHeight: parent.height /3;

    model: widgetmodel
    delegate: widgetdelegate

    //
    Item {
        id: container
        anchors.fill: parent
    }

    MouseArea {
        id: coords
        anchors.fill: parent
        onReleased: {
            var newIndex = grid.indexAt(mouseX, mouseY);
            var oldIndex = grid.firstIndexDrag;

            if (grid.firstIndexDrag != -1 && newIndex != -1 && newIndex != oldIndex){
                if (newIndex > oldIndex){
                    widgetmodel.remove(newIndex);
                    widgetmodel.move(oldIndex, newIndex -1, 1);
                    widgetmodel.insert(oldIndex, {"iColor": "cyan", "iVisible": false});
                } else {
                    widgetmodel.move(oldIndex, newIndex, 1);
                    widgetmodel.remove(newIndex+1);
                    widgetmodel.insert(oldIndex, {"iColor": "cyan", "iVisible": false});
                }
            }
            grid.firstIndexDrag = -1
        }
        onPressed: {
            grid.firstIndexDrag = grid.indexAt(mouseX, mouseY)
        }
    }
}
}
```

Conclusion

So far there are no so much documentations about this argument in internet. Googling it's possible to find a drag-n-drop implementation but with more effects and features which hide the core implementation. For this article I didn't add any of these fancy stuff to the code in order to keep it minimal and simple to read and understand. Initially I wrote this code as proof of concept for creating a Chess game in QML. As you can see with a simple logic anyone could implement such kinda games with this technology. So, I'm looking forward to see your chess or a board game like that in the OVI store soon. :D

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.