Implementing a Rotary Dialer with QML

This article shows **how to implement a Rotary Dialer in QML**.

23 Oct 2011

The Rotary Dialer

Rotary Dialers were commonly used in old telephones to send electric pulses through rotation of the dialer itself. The dialer has **one hole for each number**, and can be rotated until the **user's finger reaches the dialer stop**. When the dialer is released, it **automatically returns to its initial position**.

A Rotary Dialer can be schematized as:

- a lower static layer that shows the **numbers**
- a middle layer that represents the **rotating dialer**
- an upper layer that shows the **dialer stop position**

The base structure can be written, in an empty Dialer.qml file, as follows:

```
Rectangle {
  id: dialerView

  width: 340
  height: 340
  anchors.centerIn: parent
  color: "transparent"

  // the numbers layer

  Rectangle {
 id: numbers

 anchors.fill: parent
 color: "transparent"
 anchors.centerIn: parent
  }

  // the rotating dialer layer

  Image {
 id: dialer

 anchors.fill: parent
 source: "pics/rotary_cover.png"
  }

  // the dialer stop layer

  Image {
 id: dialerStop

 anchors.bottom: parent.bottom
 anchors.right: parent.right
 source: "pics/rotary_stop.png"
  }
}
```

So, both the **rotating dialer and the dialer stop layers will be represented as images**, while the numbers layer is just a container element, where numbers will be dynamically placed.

The numbers layer

The numbers layer must be populated with the dialer numbers. Let's create an empty RotaryDialer.js JavaScript file, and start defining a variable that holds the **angle between the number holes**:

```
// angle between numbers
var digitAngle = (3 * Math.PI / 2) / 9;
```

Now, **import the RotaryDialer.js file** in RotaryDialer.qml:

```
import "RotaryDialer.js" as Dialer
```

Done that, the number elements can be easily created by using a Repeater, as shown below:

```
Rectangle {
  id: numbers
  anchors.fill: parent
  color: "transparent"

  Repeater {
 model: 10
 delegate: Rectangle {
     width: 30
     height: 30
     color: "transparent"
     x: dialerView.width / 2 + 130 * Math.cos(index * Dialer.digitAngle) - 15
     y: dialerView.height / 2 - 130 * Math.sin(index * Dialer.digitAngle) - 15

     Text {
  color: "black"
  font.pointSize: 20
  anchors.centerIn: parent
  text: (index + 1) % 10
     }
 }
  }

  anchors.centerIn: parent
}
```

The dialer layer

The dialer layer is the part that **rotates when the user selects a number and performs a rotation** movement with his finger. When the user releases his finger, the dialer layer rotates back to its starting position.

What is needed to perform the correct dialer rotation, is:

- when the user press his finger on the dialer, **identify the angle relative to the dialer center**
- when the user moves his finger on the dialer, calculate the **difference between the new angle and the starting angle** calculated above
- when the user releases his finger, **rotate back the dialer to its starting position**

So, basically, the dialer layer must be able to intercept and handle three kinds of mouse events: onPressed, onReleased and onPositionChanged. To do this, the following MouseArea is added to the root dialer's element:

```
import QtQuick 1.0

import "RotaryDialer.js" as Dialer

Rectangle {
  id: dialerView

  [...]

  MouseArea {
 anchors.fill: parent
 onPositionChanged: Dialer.dialerMoved(mouse)
 onReleased: Dialer.dialerReleased(mouse)
 onPressed: Dialer.dialerPressed(mouse)
  }
}
```

Let's take back the RotaryDialer.js file, and define two variables that hold the **x and y coordinates of the dialer center**. This values are needed in order to calculate the angles of the user's finger positions.

```
// dialer center X
var centerX = null;
// dialer center Y
var centerY = null;

function initialize()
{
  centerX = dialer.x + dialer.width / 2;
  centerY = dialer.y + dialer.height / 2;
}
```

With the variables defined above, it is possible to calculate an angle as shown by the function below. The argument passed to this function is a MouseEvent object: this object is always passed when mouse events are generated, and so can be used to calculate the needed angles. The **getEventAngle()** function perform a normalization of the angle, to always have positive values.

```
function getEventAngle(event)
```

```
{
  var angle = Math.atan2(event.y - centerY, event.x - centerX);

  if(angle < 0)
 angle += 2 * Math.PI;

  return angle;
}
```

The dialer pressed event

When the dialer is pressed, the following operations must be performed:

- check if any **valid number is pressed** (so, if the finger ideally "fits" one of the available number holes)
- evaluate the **starting angle**
- evaluate the **maximum allowed dialer rotation**: depending on the hole that was used, the finger must be able to perform different rotations. For instance, if the user uses the hole for number "1" he will be able to perform a smaller rotation that the one for the hole "9", due to the presence of the dialer stop.

Before defining the dialerPressed function, a utility function is defined, that returns the **index of the hole corresponding to a specified angle**, or null if no hole corresponds to that angle:

```
function holeFromAngle(angle)
{
  var hole = Math.round(angle / digitAngle) % 12;

  if(hole < 10)
  {
 return hole;
  }
  else
  {
 return null;
  }
}
```

The following **dialerPressed()** function performs the operations described above:

```
function dialerPressed(event)
{
  var newAngle = getEventAngle(event);

  var hole = holeFromAngle(2 * Math.PI - newAngle);

  // check the angle corresponds to a hole
  if(hole != null)
  {
 // hold the starting angle value
 startAngle = newAngle;

 // evaluate the maximum rotation angle
 maxAngle = hole * digitAngle + dialerStopAngle;
  }
}
```

The dialer rotation

Once the user has pressed his finger on the dialer, and start moving it, the dialer must be accordingly rotated. Specifically, for each movement of the user's finger, the following operations must be performed:

- evaluate the **difference between the current angle and the start angle** (calculated by the dialerPressed event handler)
- **normalize the angle difference** within the allowed boundaries: zero (the dialer starting angle) and maxAngle (the maximum angle before the finger reaches the dialer stop)
- **apply the rotation** to the dialer layer

The following utility function returns the normalized difference between an angle and the startAngle defined above:

```
function getAngleDiff(angle)
{
  while(angle < startAngle)
 angle += 2 * Math.PI;

  return (angle - startAngle);
}
```

The **dialerMoved()** event handler can be defined as follows:

```
function dialerMoved(event)
{
  if(startAngle != null)
  {
 var newAngle = getEventAngle(event);

 var angleDiff = getAngleDiff(newAngle);

 if(angleDiff < 0)
     angleDiff = 0;
 else if(angleDiff > maxAngle)
     angleDiff = maxAngle;

 dialer.rotation = angleDiff * 180 / Math.PI;
  }
}
```

Releasing the dialer

When the user releases the dialer, the following operations must be performed:

- check **which number is dialed**: this can be different from the number corresponding to the used hole, since the user could have performed an incomplete rotation (so, his finger may have not reached the dialer stop)
- if a valid number is dialed, **send an appropriate signal**
- **rotate back the dialer** to its starting position

So, first let's define a signal that will be used to notify about the dialed numbers:

```
Rectangle {
  id: dialerView

  signal numberEntered(int number)

  [...]
}
```

Then, a RotationAnimation element is added to the dialer, in order to perform the rotation to the starting position:

```
Rectangle {
  id: dialerView

  [...]

  RotationAnimation {
 id: rotaryReleaseAnimation
 duration : 1000
 target: dialer
 property: "rotation"
 easing.type: Easing.OutBounce
 direction: RotationAnimation.Counterclockwise
  }
}
```

Now, the **dialerReleased()** event handler can be written as follows:

```
function dialerReleased(event)
{
  if(startAngle != null)
  {
 var angleDiff = getAngleDiff(getEventAngle(event));

 var hole = holeFromAngle(Math.min(angleDiff, maxAngle) - dialerStopAngle);

 if(hole != null)
 {
     var digit = (hole + 1) % 10;

     numberEntered(digit);
 }

 rotaryReleaseAnimation.to = 0;
 rotaryReleaseAnimation.running = true;

 startAngle = null;
  }
}
```

How to use the Rotary Dialer

Here's a video of the Rotary Dialer in action on a Nokia N8 device:

The media player is loading...

The Rotary Dialer can be easily used as shown in the example below:

```
Rectangle {
  width: 360
  height: 640

  Rectangle {
 border.width: 1
 border.color: "black"
 anchors.top: parent.top
 anchors.left: parent.left
 anchors.right: parent.right
 anchors.topMargin: 60
 anchors.margins: 10
 height: dialerOutput.height

 Text {
     id: dialerOutput
     text: ""
     font.pointSize: 15
     elide: Text.ElideLeft
 }
  }

  Dialer {
 onNumberEntered: dialerOutput.text += number
  }
}
```

Related Content

The Qt Creator project containing the code presented in this article is available here: File:QMLRotaryDialer.zip