Archived:Collision detection for QWidgets

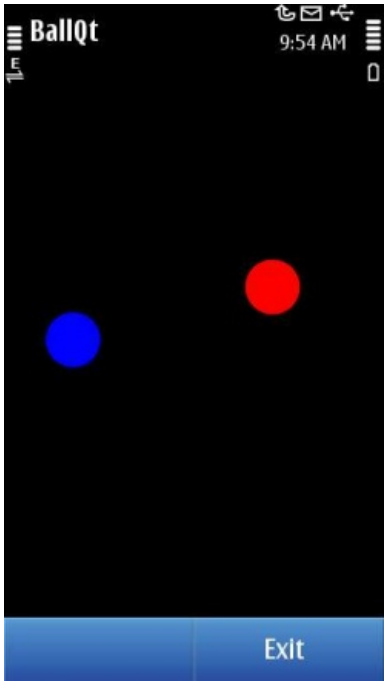
Archived: This article is **archived** because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Qt Quick should be used for all UI development on mobile devices. The approach described in this article (based on QWidget) is deprecated.

This article presents an approach with an example to illustrate *collision detection* of QWidgets. The source code of the sample application is available for download Collision detection for QWidgets.

Basic idea

We create two filled circles (`QWidgets`) moving constantly within the bounds of a window. To implement such a behavior, we override the protected method `QWidget::paintEvent()`, which would be called when the widget has to repaint itself. To draw the desired shapes (circle, in our case), we use the `QPainter` class. To paint moving circular widgets (balls) we modify their position and repaint the widgets repetitively. We can do this by using `QTimer` class. At a constant interval the timer's `timerout()` signal will be emitted and the widget's `update()` slot is called to ensure that the window repainted. Finally we find the distance between the two balls and change their position in different direction to detect collision.



Initializing Ball Properties

While starting we need to define the ball initial position and its velocity.

```
p.setX(this->width()/2 );
p.setY(this->height()/2 );
v.setX(10);
v.setY(10);
```

We consider `p` as the position of the ball and `v` is the velocity of the ball. `height()` and `width()` are the **height** and **width** of the window.

Creating Timer

We need the **timer** to repaint the balls after a specific time intervals

(C) Copyright Nokia 2014. All rights reserved.

```
    timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(update()));

    //start emitting
    timer->start(101);
```

We start the timer and it emits after a particular time interval and call the `update()` SLOT.

Balls New Positions

After every time interval we need to create new position of the ball. Here we will set the new position of the ball, and also we will check that the ball doesn't go out of the window

```
 p+=v;
if(p.x() < 0) {
p.setX(0);
v.setX(-v.x());
}
else if(p.x() > width()-D) {
p.setX(width()-D);
v.setX(-v.x());
}

if(p.y() < 0) {
p.setY(0);
v.setY(-v.y());
}
else if(p.y() > height()-D) {
p.setY(height()-D);
v.setY(-v.y());
}
```

We will create the new position for two balls.

Collision Detection

Now after we set the new properties of the balls we need to check whether the balls collide with each other or not.

```
//collision detection
xd = p1.x()-p.x();
yd = p1.y()-p.y();
Distance = sqrt(xd*xd + yd*yd);
if( 25 + 25 >= Distance)
{

    if(p.x()>p1.x())
    {
    p.setX(p.x()+(D/2));
    v.setX(-v.x());
    }

    if(p.y()>p1.y())
     {
     p.setY(p.y()+(D/2 ));
```


(C) Copyright Nokia 2014. All rights reserved.

```
        v.setY(-v.y());
        }

      if(p1.x()>p.x())
      {
          p1.setX(p1.x()+(D/2));
          v1.setX(-v1.x());
      }
      if(p1.y()>p.y())
      {
          p1.setY(p1.y()+(D/2));
          v1.setY(-v1.y());
      }


  }
```

First we find the distance between two balls and compare the distance with the ball's diameter. If its crosses the diameter then the balls collide and need to move to a different direction.

Drawing Balls

This section shows how we draw the balls after we get the properties

```
  //draw ball in its new position
  QPainter painter(this);
  QColor color(255, 0, 0);
  painter.setBrush(color);
  painter.setPen(color);
  painter.setRenderHint(QPainter::Antialiasing);
  painter.translate(p.x(),p.y());
  painter.drawEllipse(0,0,D,D);
```

We use `QPainter` to draw the balls. `QPainter` is responsible for implementing all the low-level drawing functions, including drawing different kinds of shapes, manipulating the drawing parameters and setting up matrix transformations. Here `D` represents the diameter of the balls.

Source Code

The source code presented in this article is available here File:BallQt.zip

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.