Exercises Lecture MeeGo 2 – Developing Qt Apps for MeeGo

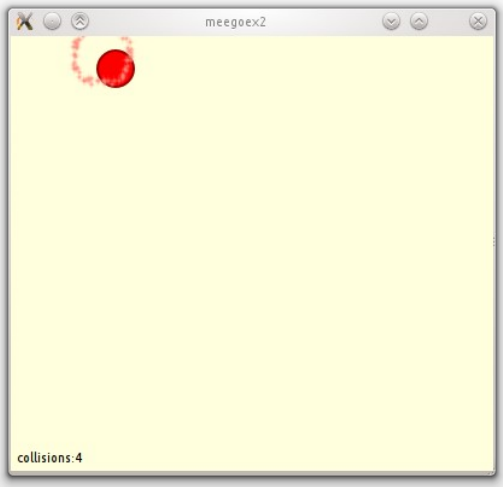
Aim: In this exercise you will create a simple Qt application, debug and deploy it to different MeeGo environments.

Duration: 1h

Introduction

The goal of this exercise is to learn how to build and install applications for MeeGo using features offered by Qt Creator. You will create a simple application based on Qt Quick and debug it using the desktop platform, Qt Simulator and a MeeGo deployment environment.

Step 1 – Creating a simple Qt Quick application



As part of this exercise, create a simple application displaying a ball bouncing off the borders of a window. Start by creating a new Qt Quick project using “Qt Quick Application” template. When asked about targets, make sure “Simulator”, “Desktop” and “MeeGo” are checked.

Just to get you started – the ball is a Rectangle element with perfectly rounded corners. Remember to give your item an id so that you can reference it later from other parts of the code. To have some fun with the application, you can make it emit particles when the ball changes direction (you can use particle images from the “Same Game” demo that comes with Qt).

In addition this this, the QML code has to emit a signal upon each collision. Connect this signal to the C++ code using a custom slot. In that slot, count the collisions and update the counter shown in the bottom left corner of the window. Store the number of collisions as a custom property of the ball item and bind it to the text displayed at the bottom of the window.

If you don't feel comfortable with Qt Quick, you can implement the application in pure C++. Ask your teacher before doing so.

> You can animate the ball using a timer, either from QML or C++. Upon each tick, update item position by a predefined step according to direction of movement of the ball. When the ball hits a border, reverse the appropriate velocity component.

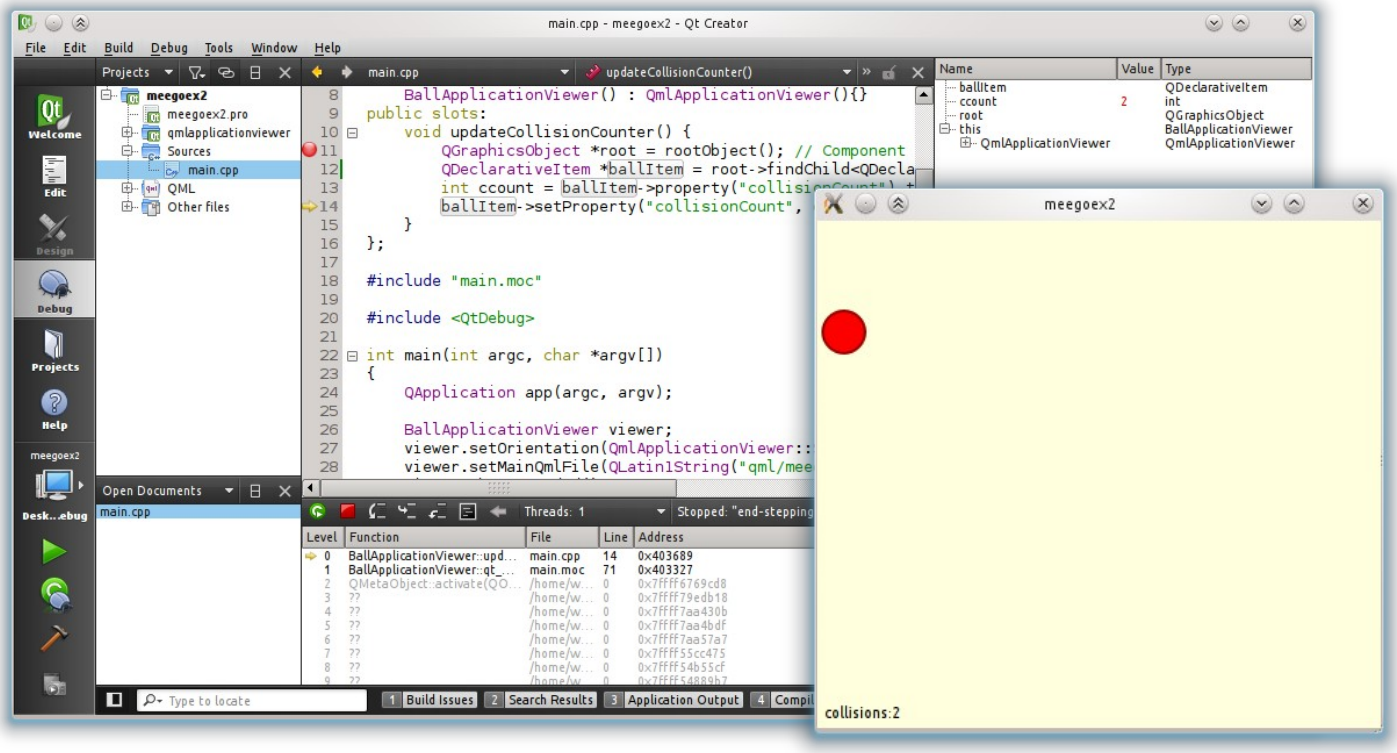
The slot in C++ should update the collision count. This can be done like this:

```
void BallApplicationViewer::updateCollisionCounter() {
    QGraphicsObject *root = rootObject(); // Component
    QObject *ballItem = root->findChild<QObject*>(“movingItem”);
    int ccount = ballItem->property(“collisionCount”).toInt();
    ballItem->setProperty(“collisionCount”, ccount+1);
}
```

If you use `QObject::findChild()` to retrieve the item pointer, make sure to set the `objectName` property in QML for the item, otherwise `findChild()` will return a null pointer.

Step 2 – Debugging the application on desktop

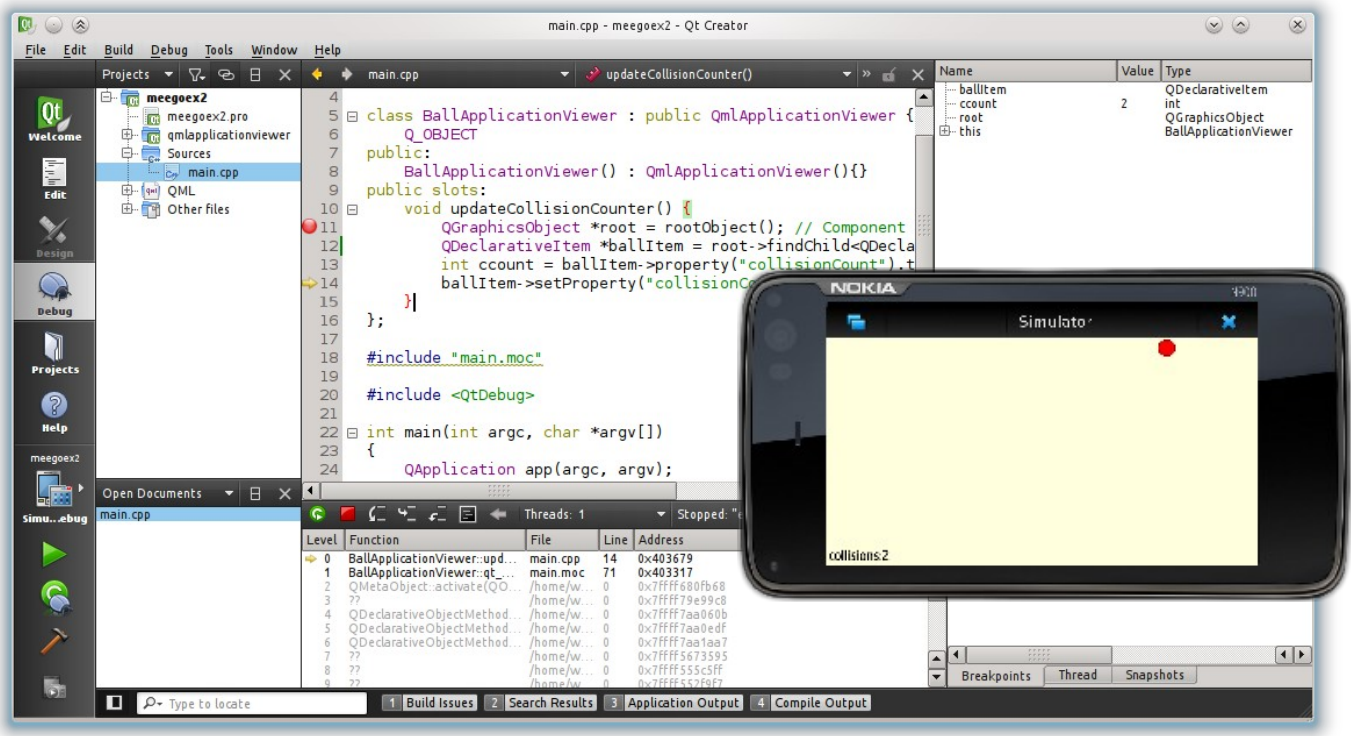
Build the application in debug mode for your desktop platform. Set a breakpoint in the C++ slot. Run the application in debug mode and observe the value of `ccount` as the program is trapped in the C++ slot.



This will be your benchmark debug session.

Step 3 – Debugging the application in the simulator

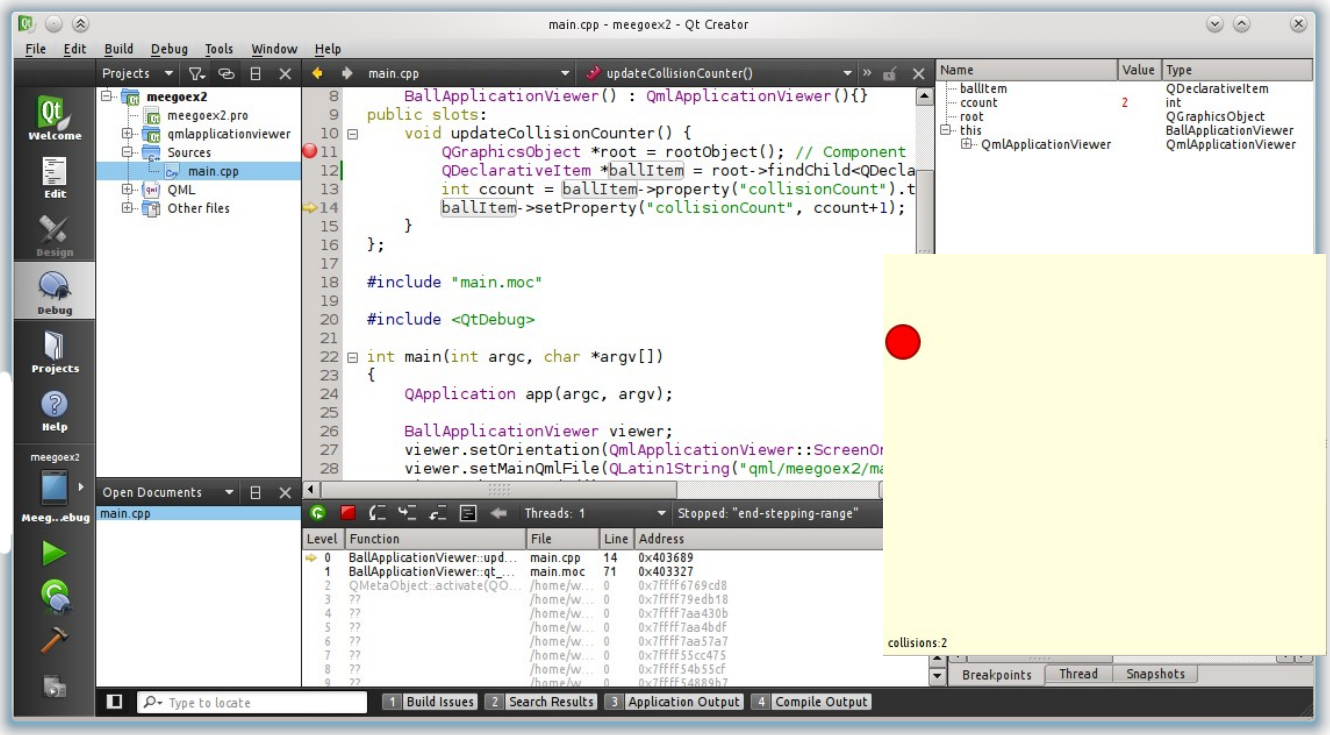
Now, repeat the process from step 2 but set the target to Qt Simulator.



Compare the results and working methods to your benchmark run.

Step 4 – Debugging the application in virtual environment

Repeat the process for the virtual device environment. Switch the target to the virtual device, make sure the virtual machine is working and run the application. If you have problems communicating with the device, you can test the connection in MeeGo Device Configurations Options panel.



Make notes of any differences in procedure and results compared to your benchmark run.

Solution Tips

Step 1

Add the following properties to the declaration of the object:

```
    property int xVel: 4 // horizontal velocity
    property int yVel: 4 // vertical velocity
    property int collisionCount: 0
```

Declare the signal as:

```
    signal collided
```

and emit it from within code executed on the timeout signal from the timer (i.e. “onTimeout” in QML).