Playing whacking game with Accelerometer Sensor

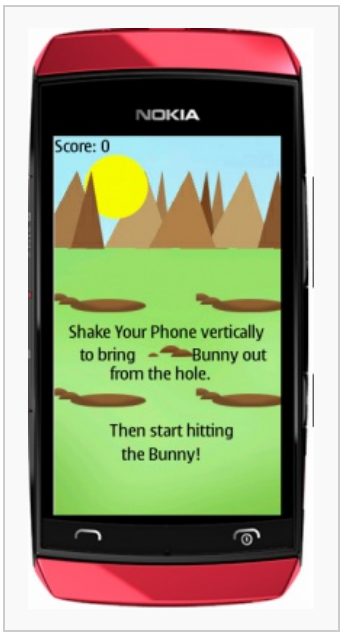
Note: This is an entry in the Asha Touch Competition 2012Q3

16 Sep 2012

This article explains how to write a simple and fun game using the Sensor API and Gesture API.

Introduction

This article demonstrates the Gesture API and Sensor API using the context of a simple game called Whack-A-Bunny, that runs on Series 40 Full Touch devices. To play the game the users shakes the phone vertically (up/down). After shaking the phone a bunny will appear from a random hole - tap the bunny before it disappears to get points. The main game screens are shown below.



Startup screen

Playing the game (after a vertical shake)

The article assumes that readers have a working knowledge of Java ME covering: basics, threads and timers, and graphics and Canvas.

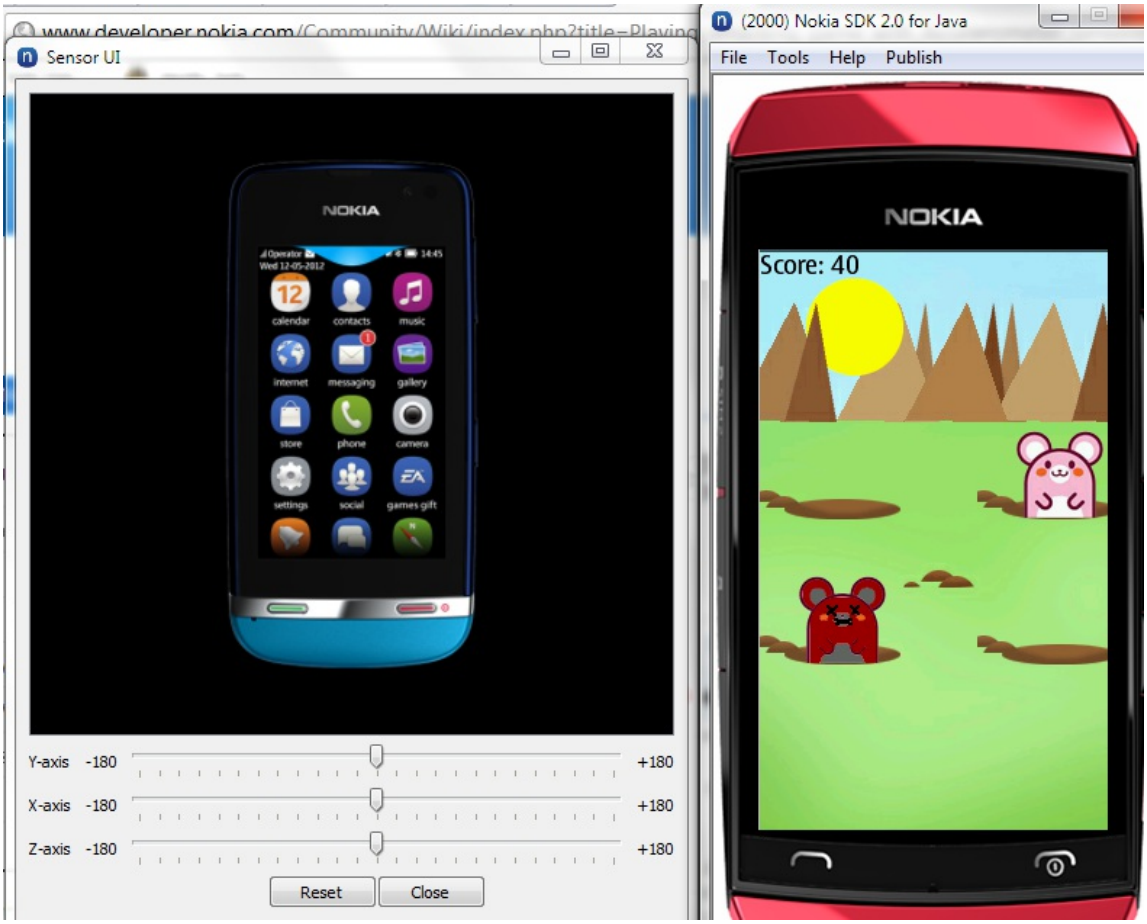
Testing the game

The application can be downloaded and run on a touch device (source code and install files are here).

You can also use the sensor simulation from the SDK phone simulator to simulate the shaking of the phone. For this app, you could move the y-axis bar of the sensor simulation from left to right or vice versa, to simulate of flipping the phone vertically. To open, just click the following from the phone Emulator menu: **Tools -> Sensor Simulation**

Below is a screenshot of Sensor Simulation display, and hitting one of the bunnies using the phone simulator.

(C) Copyright Nokia 2013. All rights reserved.



Source code

The key classes in this example are listed below: **WhackaMainMidlet.java**, **WhackaMainMidlet.class**, **WhackGame.java**, **WhackGame.class**

1. WhackaMainMidlet Midlet:

This class contains the code implementation for the "acceleration" movement of the phone with the use of accelerometer sensor.
The **WhackaMainMidlet** implements the `DataListener` interface to implement the Sensor API. The `DataListener` interface has only one method to be implemented, which we will explain it later. See DataListener interface for more details. Here's the part of the code in the `startApp()` method where we instantiate the `WhackGame` class (a `GameCanvas` object). This is set as a current display. Also in this code we open the acceleration sensor and set the `DataListener` object. Since this is a simple app, and we don't need a large data buffering because the app is not doing any critical performance and to conserve memory space also, then we just set The BUFFERSIZE to 1. You may adjust the BUFFERSIZE up to 256 for handling a larger data buffering. For more info about data buffering, see: SensorConnection .

```
import java.io.IOException;
import javax.microedition.io.Connector;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.sensor.*;
/**
 * @author NoelAnonas
 */
public class WhackaMainMidlet extends MIDlet implements DataListener {
    static int currentEvent = -1;
    private static final int BUFFERSIZE = 1;
    private SensorConnection sensor;

    public void startApp() {
        WhackGame wg = new WhackGame(this);
        Display.getDisplay(this).setCurrent(wg);
        sensor = openAccelerometerSensor();
  if (sensor==null) return;
  sensor.setDataListener(this, BUFFERSIZE);
```

The application implements the `DataListener Interface` and registers itself with the `setDataListener`method, the mode of retrieval of data from the sensor is asynchronous. See SensorConnection Interface for more details about the mode of retrieval of data. Below is the source code for the `openAccelerometerSensor()` method. Since the application would use the accelerator sensor of the device, we would use the "acceleration" as the quantity value of the findSensors method of SensorManager. For more information about `SensorManager class` you may visit this link: SensorManager Class. For other possible quantity value for finding sensors aside from our "acceleration" value to refer to Accelerometer Sensor, you may visit this link: Finding Sensors.

```
    private SensorConnection openAccelerometerSensor(){
   SensorInfo infos[] = SensorManager.findSensors("acceleration", null);
   if (infos.length == 0) return null;
   try{
    return (SensorConnection)Connector.open(infos[0].getUrl());
   }catch(Exception e){
    e.printStackTrace();
    System.out.println("Problem in opening the accelerometer Sensor. "+ infos[0].getUrl());
```


(C) Copyright Nokia 2013. All rights reserved.

```
    return null;
  }
 }
```

Once the `DataListener` bound to the target object (midlet), the `dataReceived()` method will be called for each acceleration action done on the phone device. The `dataReceived()` method is the implementation of the abstract method of the `DataListener` interface.

The x,y and z data coordinates of the phone acceleration can be determined using the array of `Data` type argument of `dataReceived()` method. We are only looking for vertical acceleration movement, i.e. the up and down action. See Data Interface for more details. To easily determine it, we convert the data to an `int` key action (`convertToActionEvent()` method).

```
public void dataReceived(SensorConnection sensor, Data[] data, boolean isDataLost) {
      int[] events = convertToActionEvent(data);
      for (int i=0; i<BUFFERSIZE; i++){
  if (events[i] == currentEvent )
   continue;
  currentEvent = events[i];
 }
   }
```

For the `data[0].getChannelInfo()` method in `convertToActionEvent()` method, we used the value 0 for array of data to indicate the application uses the information of the first channel of the sensor. This method returns a `ChannelInfo` object. See the ChannelInfo. To get the y-coordinate value of the accelerometer sensor, we check first the return value of `getDataType()` method of the `ChannelInfo` object to see what particular data type would this `ChannelInfo` object produces . If the return data is int, then we use the data[1].getIntValues() else data[1].getDoubleValues() code will be used if the return data is double. The assigned index 1 of data-array variable refers to the y-axis direction of the accelerometer sensor. After getting this result data value, We use the `getActionKey()` method to convert the value to the desired int value, based on the vertical shake movement of the phone or the phone UI simulation, that is, if it is shaken upward and downward.

Code for the `convertToActionEvent()` method:

```
private static int[] convertToActionEvent(Data[] data){
  ChannelInfo cInfo = data[0].getChannelInfo();
  boolean isInt = false;
                if (cInfo.getDataType() == ChannelInfo.TYPE_INT) isInt= true; else isInt= false;
  int[] events = new int[BUFFERSIZE];

  if (isInt){
     int []  vint = data[1].getIntValues();  //index=1, getting the y-coordinate values only;
   for (int i=0; i<BUFFERSIZE; i++){
    events[i] = getActionKey(vint[i]);
   }
   return events;
  }
                double [] vdouble = new double[BUFFERSIZE];
                vdouble = data[1].getDoubleValues();
                for (int i=0; i<BUFFERSIZE; i++){
   events[i] = getActionKey(vdouble[i]);
  }
  return events;
 }
```

Below is the code for `getActionKey()` method. Based on this code, if the ycoord value is less than 0, it means the phone moved in upward position, and the return value of the getActionKey() method is the int value of Canvas.UP constant. If the ycoord value is zero or greater than 0 then the return int value is Canvas.DOWN constant. See Canvas.UP and Canvas.DOWN.

```
private static int getActionKey(double ycoord){
  // ycoord: UP or DOWN only
                        boolean isUp = ycoord<0;
                        if (isUp) return Canvas.UP;
                        return Canvas.DOWN;
        }
```

2. WhackGame Canvas:

This canvas contains the necessary code to display the game graphics, to capture the "acceleration" movement of the phone and to capture the gesture tap made by the user.
The "WhackGame" canvas class implements the `Runnable` interface for a thread object and the `GestureListener` interface for Gesture Tap. We need this Runnable in our application because we need another thread object that will monitor the data independently from the accelerometer sensor. For more info about Runnable interface, see: Runnable interface . For more info about Thread, see: Thread class .

To work the gesture tap to the game, we need the following procedures:

1. Implement the `gestureAction()` method of the `com.nokia.mid.ui.gestures.GestureListener` interface. all codes relating to the tap activities/actions will be placed here.
2. Instantiate the `com.nokia.mid.ui.gestures.GestureInteractiveZone` class with an argument of `GestureInteractiveZone.GESTURE_TAP` because we would just want to monitor the gesture tap only. The gesture tap is the same as pressing the touch-screen and releasing it quickly.
3. Register the `GestureListener` object and the `GestureInteractiveZone` object to the `GestureRegistrationManager`. Use the the static `register()` method of `GestureRegistrationManager` to bind those two objects to the `GestureRegistrationManager`. This is use to register a gesture interactive zone to a container, which is our `WhackGame` object.
4. And finally, use the `GestureRegistrationManager.setListener(this, this)` to add the listener object to the container. The reason why the value of the `setListener()` method is set to two "this" method is because the `WhackGame` object extends the `GameCanvas` class (container), and implements the `GestureListener` interface (listener).

For more info about registering the gesture, see GestureRegistrationManager .

```
import com.nokia.mid.ui.gestures.GestureEvent;
import com.nokia.mid.ui.gestures.GestureInteractiveZone;
import com.nokia.mid.ui.gestures.GestureListener;
import com.nokia.mid.ui.gestures.GestureRegistrationManager;
import java.io.IOException;
import java.util.Random;
import javax.microedition.lcdui.Canvas;
```


(C) Copyright Nokia 2013. All rights reserved.

```
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.GameCanvas;


/**
 * Title: Whack-A-Bunny
 * @version 1.0
 * @author NoelAnonas
 */
public class WhackGame extends GameCanvas implements Runnable, GestureListener {

    WhackaMainMidlet bm1;
    String msg = "";
    Image bgd, hit, b1;
    Image hole1;
    int vscore, vspeed = 500, vkey = 0;
    boolean bhit1, bhit2, bhit3, bhit4;
    boolean unSplash;
    int vrateappear=180;
    boolean pullup, pulldown, show1, show2, show3, show4;
    long hitdelay = 0, hitdelay2 = 0, hitdelay3 = 0, hitdelay4 = 0, timestart = 0;

    public WhackGame(WhackaMainMidlet bm) {
        super(true);
        bm1 = bm;
//Set the game in fullscreen mode
        setFullScreenMode(true);
        try {
            bgd = Image.createImage("/bg.jpg"); //background image
            hit = Image.createImage("/hit1.png"); //whacked bunny image
            hole1 = Image.createImage("/hole1.png"); //hole image
            b1 = Image.createImage("/char1.png");
            msg = "Score: ";
        } catch (IOException ex) {
            ex.printStackTrace();
        }

        GestureInteractiveZone giz = new GestureInteractiveZone(GestureInteractiveZone.GESTURE_TAP);
        GestureRegistrationManager.register(this, giz);
        GestureRegistrationManager.setListener(this, this);
//Create a thread object and run it.
        Thread t = new Thread(this);
        t.start();
    }
```

We override the abstract method of `GestureListener` interface named `GestureAction` to accept the Gesture Tap data. In this code, we check whether the Gesture Tap is within the designated area.

```
    public void gestureAction(Object container, GestureInteractiveZone gZone, GestureEvent gEvent) {
        switch (gEvent.getType()) {
            case GestureInteractiveZone.GESTURE_TAP:
//Check if tap is within the area of the top-left bunny
                boolean isInsideX11 = gEvent.getStartX() > 36 & gEvent.getStartX() < 90;
                boolean isInsideY11 = gEvent.getStartY() > 136 & gEvent.getStartY() < 185;

//Check if tap is within the area of the top-right bunny
                boolean isInsideX12 = gEvent.getStartX() > 187 & gEvent.getStartX() < 230;
                boolean isInsideY12 = gEvent.getStartY() > 136 & gEvent.getStartY() < 185;

//Check if tap is within the area of the bottom-right bunny
                boolean isInsideX21 = gEvent.getStartX() > 36 & gEvent.getStartX() < 80;
                boolean isInsideY21 = gEvent.getStartY() > 230 & gEvent.getStartY() < 280;

//Check if tap is within the area of the bottom-left bunny
                boolean isInsideX22 = gEvent.getStartX() > 187 & gEvent.getStartX() < 230;
                boolean isInsideY22 = gEvent.getStartY() > 230 & gEvent.getStartY() < 280;

//Bunny is hit at the top-left bunny
                if (isInsideX11 & isInsideY11 & show1) {
                    vscore += 10;
                    show1 = false;
                    bhit1 = true;
                    hitdelay = System.currentTimeMillis();
                }

//Bunny is hit at the top-right bunny
                if (isInsideX12 & isInsideY12 & show2) {
                    vscore += 10;
                    show2 = false;
                    bhit2 = true;
                    hitdelay2 = System.currentTimeMillis();
                }

//Bunny is hit at the bottom-left bunny
                if (isInsideX21 & isInsideY21 & show3) {
                    vscore += 10;
                    show3 = false;
                    bhit3 = true;
                    hitdelay3 = System.currentTimeMillis();
                }

//Bunny is hit at the bottom-right bunny
                if (isInsideX22 & isInsideY22 & show4) {
                    vscore += 10;
                    show4 = false;
                    bhit4 = true;
                    hitdelay4 = System.currentTimeMillis();
                }

//draw all the images again
                repaint();
                break;
        }
    }
```

We also override the run abstract method of `Runnable` interface to run the other thread. This thread is used to monitor the current "acceleration" event data from the WhackaMainMidlet (`WhackaMainMidlet.currentEvent`) , and to display the Bunny image on the screen accordingly. It randomly generates a value for the appearance speed of the bunny. The pullup and pulldown variables are set to true if the user shakes the phone vertically. There's a random generator to determine what particular bunny to be displayed in a certain location. Using nextInt(2) of Random object, it randomly generates an int value between 0 (inclusive) and 1. The initial speed rate of appearing and disappearing of the bunny is 500 milliseconds. This will be changed after the initial use. The next speed rate will be determined by this formula: *vspeed = 5 * (vrateappear + r.nextInt(120))* . This formula guaranteed that the speed rate to

 (C) Copyright Nokia 2013. All rights reserved.

display a bunny is in the range of 900 and 1495 inclusively. See Random class for more info.

```
 public void run() {
        while (true) {
            msg = "Score: " + vscore;

 //Getting the value from the accelerometer
 //the vkey will be used to check the shake direction of the phone (up/down).
            vkey = WhackaMainMidlet.currentEvent;
            if (timestart != 0) {
//Get the current time
                long timenow = System.currentTimeMillis();
//Get the time difference between the time when the bunny appeared and the current time.
                long diff = timenow - timestart;
//Check the time difference between the time when the bunny appeared and the given random time.
                if (diff > vspeed) {
                    pullup = false;
                    pulldown = false;
                    show1 = false;
                    show2 = false; show3=false; show4=false;
                    timestart = 0;
                    Random r = new Random();
                    vspeed = 5 * (vrateappear + r.nextInt(120));
                }
            }

// Remove the whacked bunny if any.
            removeHitImage();

//Enable pullup and pulldown variables if the phone is shaken upward or downward respectively.
            switch (vkey) {
                case Canvas.UP:
                    pullup = true;
                    break;
                case Canvas.DOWN:
                    pulldown = true;
                    break;
            }

 // if the user shake the phone from down to up or vice versa, it will
 // randomly display the bunny image to the screen.
            if (pullup & pulldown) {
                unSplash = true;
                pullup = false;
                pulldown = false;
                Random r = new Random();
            //if the random number is equal to 1 then display the bunny.
                if (r.nextInt(2)==1) show1 = true;
                if (r.nextInt(2)==1) show2 = true;
                if (r.nextInt(2)==1) show3 = true;
                if (r.nextInt(2)==1) show4 = true;
             //reset the time for timestart
                timestart = System.currentTimeMillis();
            }
//Refresh the drawing area.
            repaint();
        }
    }
```

Code for the `removeHitImage()` method mentioned from the `run()` method. This is used to clean the bunny that was hit. The vpause variable is used to provide the time delay to display the whacked bunny. In this example, the whacked bunny will be displayed for about 800 millisecond before it disappears.

```
//removing the hit bunny image after 800 millisecond
    public void removeHitImage() {
        int vpause =800;
//Remove the whacked bunny at the top-left corner
        if (bhit1) {
            if (System.currentTimeMillis() - hitdelay > vpause) {
                bhit1 = false;
                hitdelay = 0;
            }
        }
//Remove the whacked bunny at the top-right corner
        if (bhit2) {
            if (System.currentTimeMillis() - hitdelay2 > vpause) {
                bhit2 = false;
                hitdelay2 = 0;
            }
        }
//Remove the whacked bunny at the bottom-left corner
        if (bhit3) {
            if (System.currentTimeMillis() - hitdelay3 > vpause) {
                bhit3 = false;
                hitdelay3 = 0;
            }
        }
//Remove the whacked bunny at the bottom-right corner
        if (bhit4) {
            if (System.currentTimeMillis() - hitdelay4 > vpause) {
                bhit4 = false;
                hitdelay4 = 0;
            }
        }
    }
```

To complete the code for this application, Here's the code of the `paint()` method. This is the area where the images and text are drawn. The `drawString()` method of the `Graphics` object is used to draw the text in a canvas, The reason why there's a space value of the string arguments of the `drawString()` method for the welcome screen is because we would like to put the text away from some dark image background and to put in a proper location. See the comments below for more code info.

```
  public void paint(Graphics g) {
//Draw the background image.
        g.drawImage(bgd, 0, 0, Graphics.LEFT | Graphics.TOP);
//Draw the hole image at the top-left corner
        g.drawImage(hole1, 0, 160, Graphics.LEFT | Graphics.TOP);
//Draw the hole image at the top-right corner
        g.drawImage(hole1, 150, 160, Graphics.LEFT | Graphics.TOP);
```


(C) Copyright Nokia 2013. All rights reserved.

```
 //Draw the hole image at the bottom-left corner
        g.drawImage(hole1, 0, 260, Graphics.LEFT | Graphics.TOP);
 //Draw the hole image at the bottom-right corner
        g.drawImage(hole1, 150, 260, Graphics.LEFT | Graphics.TOP);
  //display the bunny at top-left corner if the user shakes the phone vertically
        if (show1) {
            g.drawImage(b1, 10, 112, Graphics.LEFT | Graphics.TOP);
        }
  //if bunny was hit at top-left corner.
        if (!show1 & bhit1) {
            g.drawImage(hit, 10, 112, Graphics.LEFT | Graphics.TOP);
        }
  //display the bunny at top-right corner if the user shakes the phone vertically
        if (show2) {
            g.drawImage(b1, 160, 111, Graphics.LEFT | Graphics.TOP);
        }
  //if bunny was hit at top-right corner.
        if (!show2 & bhit2) {
            g.drawImage(hit, 160, 111, Graphics.LEFT | Graphics.TOP);
        }
  //display the bunny at bottom-left corner if the user shakes the phone vertically
        if (show3) {
            g.drawImage(b1, 10, 211, Graphics.LEFT | Graphics.TOP);
        }
  //if bunny was hit at bottom-left corner.
        if (!show3 & bhit3) {
            g.drawImage(hit, 10, 211, Graphics.LEFT | Graphics.TOP);
        }
  //display the bunny at bottom-right corner corner if the user shakes the phone vertically
        if (show4) {
            g.drawImage(b1, 160, 211, Graphics.LEFT | Graphics.TOP);
        }
  //if bunny was hit at bottom-right corner.
        if (!show4 & bhit4) {
            g.drawImage(hit, 160, 211, Graphics.LEFT | Graphics.TOP);
        }
        g.drawString(msg, 0, 0, Graphics.LEFT | Graphics.TOP);

 //This is the Welcome Screen.
       if (!unSplash)  {
                                g.drawString("Shake Your Phone vertically ", 15, 196, Graphics.TOP|Graphics.LEFT  );
                                g.drawString("   to bring Bunny out  ", 15, 220, Graphics.TOP|Graphics.LEFT  );
                                g.drawString("           from the hole.  ", 15, 240, Graphics.TOP|Graphics.LEFT  );
                                g.drawString("           Then start hitting   ", 15, 300, Graphics.LEFT | Graphics.TOP);
                                g.drawString("              the Bunny!   ", 15, 325, Graphics.LEFT | Graphics.TOP);
      }
 }
```

That's it! Have fun in playing this game.

Downloads

- Media:WhackaBunnySourceCode.zip - Full source code for the Whack-A-Bunny game
- Media:WhackaBunny.zip - Installation files for the device (zip contains .jad and .jar files)

Summary

Using the Gesture API and Sensor API is simple to implement once you know the basics of using it.

(C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.