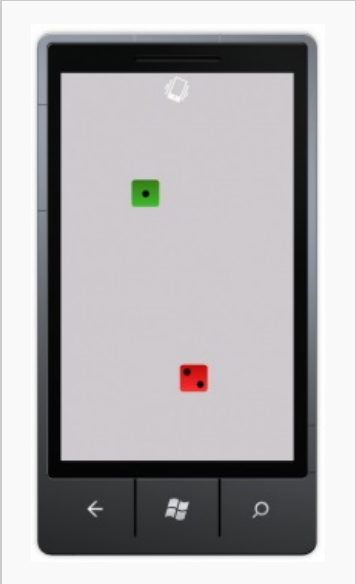
Using the Accelerometer in windows phone for shake app interaction

This article explains how to properly use the accelerometer for effects and user interaction for apps.
This is a windows phone version of my Using the Accelerometer for cool apps in JavaME which shows the same principle

Introduction

User interaction with an application is usually constraints to the Touch screen, However thinking and implementing other user interaction methods can be both fun and useful for different apps or actions within the app. For this article i will demonstrate how to use the accelerometer in a functional way to detect shake and interact with the app.



App main

After shaking

The code

The application code & samples are all written in XNA+SL and it should run on all WP phones. The code is split into 4 classes

- `Dice` - a domain class this represent a single die, it has the x,y position as well as the current state of the die (moving,rolling) it also has all the strips of the die sides.
- `DirectionSensor` - this utility class is responsible to listen to the accelerometer x,y,z axis and calculate the difference between the current and former x,y,z state and if the difference is large enough it send a message to `ShakeListener`
- `GamePage` - This class is a `PhoneApplicationPage` class which is the only SL page of the application.
- `DiceScreen` - The class that is responsible for initialization of the dice, their drawing and movements update.

I'll post below parts of the code for the `DirectionSensor` class and `GamePage` class which both relates to the topic of this article which is the use of the accelerometer, for the full source code of all the class you can downloaded the attached source project.

`DirectionSensor` class - 3 important methods. `dataReceived(Object con, SensorReadingEventArgs<AccelerometerReading> e)` - this method is needed in order to implements the `SensorReadingEventArgs<AccelerometerReading>` for the `EventHandler` it calls `activate(ShakeListener listener)` Activates the accelerometer and register the proper handler to it.

```
public void dataReceived(Object con, SensorReadingEventArgs<AccelerometerReading> e)
{
        Vector3 a = e.SensorReading.Acceleration;
        double[] temp = { a.Y * 0.5, a.X * 0.5, a.Z * 0.5 };
  if (!receivedFirstData)
  {
   receivedFirstData = true;
   acceleration[0] = (float) temp[1];
```


(C) Copyright Nokia 2014. All rights reserved.

```
    acceleration[1] = (float) temp[0];
    acceleration[2] = (float) temp[2];
    return;
   }
   else
   {
    if (Math.Abs(vectorMagnitude(new float[]{(float) temp[1],(float) temp[0],(float)
temp[2]})-vectorMagnitude(acceleration))>DIRECTION_DELTA)
    {
     acceleration[0] = (float) temp[1];
     acceleration[1] = (float) temp[0];
     acceleration[2] = (float) temp[2];
     listener.shakeStarted();
     return;
    }
    else
    {
     acceleration[0] = (float) temp[1];
     acceleration[1] = (float) temp[0];
     acceleration[2] = (float) temp[2];
     return;
    }
   }
}
```

`vectorMagnitude(float[] arr)` - this is a helper function to calculate a 3d vector's magnitude - the x,y,z values of the accelerometer can be think of as 3d vector and you can calculate its magnitude and compare it to another 3d vector magnitude and if the difference between the two magnitudes are bigger than delta then the movement was big.

```
private static float vectorMagnitude(float[] arr)
{
 return (float) Math.Abs(Math.Sqrt(arr[0]*arr[0]+arr[1]*arr[1]+arr[2]*arr[2]));
}
```

`activate(ShakeListener listener)` - this function is called when you want to start to register to accelerometer shake events.

```
public void activate(ShakeListener listener)
{
 this.listener = listener;
 receivedFirstData = false;
        handler = new
EventHandler<SensorReadingEventArgs<AccelerometerReading>>(dataReceived);
        tiltsensor.CurrentValueChanged += handler;
        tiltsensor.Start();
}
```

`GamePage` class - Its important parts are within the `OnNavigatedTo` method where we initialize the `DiceScreen` and register for the shake events and in the `OnNavigatedFrom` we make sure to unregister to the shake events, this class also implements the `ShakeListener` interface which consist of just 1 method shakeStarted().

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
            // Set the sharing mode of the graphics device to turn on XNA rendering
            SharedGraphicsDeviceManager.Current.GraphicsDevice.SetSharingMode(true);
```


(C) Copyright Nokia 2014. All rights reserved.

```
            width = SharedGraphicsDeviceManager.Current.GraphicsDevice.Viewport.Width;
            height = SharedGraphicsDeviceManager.Current.GraphicsDevice.Viewport.Height;
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new
SpriteBatch(SharedGraphicsDeviceManager.Current.GraphicsDevice);
            gameScreen = new DiceScreen(contentManager);
            gameScreen.setDimension(width, height);
            sensor.activate((ShakeListener)this);
            // Start the timer
            timer.Start();
            base.OnNavigatedTo(e);
}
// the shake interface implementation
public void shakeStarted()
{
            gameScreen.startRolling();
}
```

Important Note

When implementing a new kind of user interaction you should always make sure the user is aware of how this user interaction works and to what it is used. If you will see the complete source attached to this article you'll notice that in the main screen I've added an image of a "phone shake" action at the top of the screen to indicate the user he should shake the phone for action.

Summary

This specific type of user interaction can be used for a variety of apps and situation such as Form data reset, Paint app clear screen, Snow globe application and my Dice shaking example but always remember my note from above and make sure the user will know of this option properly. I hope you enjoyed this article and download the source and test out the app yourself.

(C) Copyright Nokia 2014. All rights reserved.