CONTENIDO

1. ENTRADAS DE DATOS

Veremos los eventos de teclado, la pantalla táctil y por último los sensores. Después aplicaremos estos tres mecanismos de interacción al manejo de la bici en la aplicación didáctica/juego que estamos realizando.

1.1. Manejo de eventos de usuario

Los eventos de usuario se recogen en Android mediante escuchadores de eventos o mediante manejadores de eventos.

1.1.1. Escuchadores de eventos

Los escuchadores de eventos de los que disponemos son los siguientes:

onClick()	Método de la interfaz View.onClickListener. Se llama cuando el usuario selecciona un elemento.
onLongClick()	Método de la interfaz View.onLongClickListener. Se llama cuando el usuario selecciona un elemento durante más de un segundo.
onFocusChange()	Método de la interfaz View.onFocusChangeListener. Se llama cuando el usuario navega dentro o fuera de un elemento.
onKey()	Método de la interfaz View.onKeyListener. Se llama cuando se pulsa o se suelta una tecla del dispositivo.
onTouch()	Método de la interfaz View.onTouchListener. Se llama cuando se pulsa o se suelta o se desplaza en la pantalla táctil.
onCreateContextMenu()	Método de la interfaz View.onCreateContextMenuListener. Se llama cuando se crea un menú de contexto.

Una forma de crear un escuchador de evento para un botón sería:

```
public class Ejemplo extends Activity implements OnClickListener {

    protected void onCreate(Bundle savedValues) {
        //...
        Button boton = (Button)findViewById(R.id.Button01);
        boton.setOnClickListener(this);
    }

    public void onClick(View v) {
        //Acciones a realizar
    }
    //...
}
```

1.1.2.Manejadores de eventos

En una clase que herede de la clase View podemos utilizar directamente los siguientes manejadores de evento.

onKeyDown(int keyCode, KeyEvent e)	Llamado cuando una tecla es pulsada.
onKeyUp(int keyCode, keyEvent e)	Llamado cuando una tecla deja de ser pulsada.
onTrackballEvent(MotionEvent me)	Llamado cuando se mueve el trackball.
onTouchEvent(MotionEvent me)	Llamado cuando se utilice la pantalla táctil.
onFocusChanged(boolean obtengoFoco, int direccion, Rect prevRectanguloFoco)	Llamado cuando cambia el foco.

1.2. El Teclado

1.2.1.Ejercicio Guiado 1

Como ejemplo proponemos un manejador de eventos de teclado para manejar la bici de la aplicación Solobici Didáctica que debemos incluir en la clase VistaJuego:

```
@Override
public boolean onKeyDown(int codigoTecla, KeyEvent evento) {
    super.onKeyDown(codigoTecla, evento);
    //Procesamos la pulsación
    boolean pulsacion=true;
    switch (codigoTecla) {
        case KeyEvent.KEYCODE_DPAD_UP:
            aceleracionBici=+PASO_ACELERACION_BICI;
            break;
        case KeyEvent.KEYCODE_DPAD_LEFT:
            giroBici=-PASO_GIRO_BICI;
            break;
        case KeyEvent.KEYCODE_DPAD_RIGHT:
            giroBici=+PASO_GIRO_BICI;
            break;
        case KeyEvent.KEYCODE_DPAD_CENTER:
        case KeyEvent.KEYCODE_ENTER:
            //lanzarRueda();
            break;
        default:
            //Si estamos aquí no hemos pulsado nada que nos interese
            pulsacion=false;
            break;
    }
    return pulsacion;
}
```

1.2.2.Ejercicio 1

El manejador de eventos onKeyDown sólo se activa cuando se pulsa una tecla, pero no cuando se suelta. Trata de escribir el manejador de eventos onKeyUp para que la bici atienda a las órdenes de forma correcta y tenga un movimiento menos caótico.

PISTA: Podéis probar haciendo que al soltar la tecla cursor “arriba” la aceleración se haga 0 y que al soltar las teclas cursor “izquierda” y “derecha” el giro se haga 0.

1.3. La Pantalla Táctil

La pantalla táctil se puede manejar mediante el método onTouchEvent en una clase View (o implementando la interfaz onTouchListener en otras clases). Este método nos devolverá en un parámetro, un objeto de la clase MotionEvent.

Los métodos más interesantes de la clase MotionEvent son:

getAction()	Tipo de acción realizada: ACTION_DOWN, ACTION_MOVE, ACTION_UP o ACTION_CANCEL.
getX(), getY()	Posición de la pulsación.
getDownTime()	Tiempo en ms en que el usuario presionó por primera vez en una cadena de eventos de posición.
getEventTime()	Tiempo en ms del evento actual.
getPression()	Estima la presión de la pulsación. El valor 0 es el mínimo, el valor 1 representa una pulsación normal.
getSize()	Valor entre 0 y 1 que estima el grosor de la pulsación.

1.3.1. Ejercicio Guiado 2

Utilizaremos el siguiente código para utilizar la pantalla táctil en el juego didáctico que venimos realizando:

```
// PANTALLA TÁCTIL //
// Las variables mX y mY se utilizarán para recordar
// las coordenadas del último evento.
private float mX=0, mY=0;
private boolean disparo=false;

@Override
public boolean onTouchEvent(MotionEvent evento) {
    super.onTouchEvent(evento);
    //Obtenemos la posición de la pulsación
    float x=evento.getX();
    float y=evento.getY();
    switch (evento.getAction()) {
    //Si comienza una pulsación (ACTION_DOWN) activamos la variable disparo
    case MotionEvent.ACTION_DOWN:
        disparo=true;
        break;
    //Comprobamos si la pulsación es continuada con un desplazamiento horizontal o vertical.
    //En caso de ser asi, desactivamos disparo porque se tratará de un movimiento
    //en lugar de un disparo.
    case MotionEvent.ACTION_MOVE:
        float dx=Math.abs(x-mX);
        float dy=Math.abs(y-mY);
        if (dy<6 && dx>6)   //Un desplazamiento del dedo horizontal hace girar la bici.
        {
            giroBici = Math.round((x-mX)/2);
            disparo = false;
        } else if (dx<6 && dy>6)    //Un desplazamiento vertical produce una aceleración.
                {
                    aceleracionBici = Math.round((mY-y)/25);
                    disparo = false;
                }
        break;
```

```
    //Si se levanta el dedo (ACTION_UP) sin haberse producido desplazamiento horizontal o vertical
    //disparo estará activado y lo que hacemos es disparar
    case MotionEvent.ACTION_UP:
        giroBici = 0;
        aceleracionBici = 0;
        if (disparo){
            //ActivarRueda();
        }
        break;
    }
    mX=x; mY=y;
    return true;
}
```

1.4. Los sensores

Android permite acceder a los sensores internos del dispositivo a través de las clases Sensor, SensorEvent, SensorManager y la interfaz SensorEventListener, del paquete android.hardware.

La clase Sensor acepta 8 tipos de sensores, aunque los sensores disponibles dependerán del dispositivo que estemos utilizando.

TYPE_ACCELEROMETER	Acelerómetro.
TYPE_GYROSCOPE	Giroscopio.
TYPE_LIGHT	Sensor de luz.
TYPE_MAGNETIC_FIELD	Brújula.
TYPE_ORIENTATION	Sensor de orientación.
TYPE_PRESSURE	Sensor de presión.
TYPE_PROXIMITY	Sensor de proximidad.
TYPE_TEMPERATURE	Sensor de temperatura.

1.4.1.Conocer los sensores del dispositivo

Utilizando la clase SensorManager podemos conocer los sensores de los que dispone nuestro dispositivo.

Podemos crear una aplicación ***(EjemploSensor)*** con el siguiente código:

```
package org.ejemplo.aguilar;

import java.util.List;

public class EjemploSensor extends Activity {
    private TextView salida;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //Campo de texto para mostrar la salida
        salida = (TextView)findViewById(R.id.salida);
        //Objeto SensorManager que nos permitirá ver la lista de sensores del dispositivo
        SensorManager miSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
        List<Sensor> listaSensores = miSensorManager.getSensorList(Sensor.TYPE_ALL);
        //Vamos mostrando los sensores uno a uno
        for(Sensor sensor:listaSensores) {
            mostrar(sensor.getName());
        }
    }

    private void mostrar(String cadena) {
        salida.append(cadena + "\n");
    }
}
```

Añadiendo la siguiente propiedad al TextView de res/layout/main.xml:

```
android:id="@+id/salida"
```

1.4.2.Acceder a los datos del sensor

Para acceder a los datos del sensor se lo debemos indicar al SensorManager mediante el método registerListener().

Para ello podemos usar el siguiente código que debemos introducir en el método onCreate().

```
//Registramos los sensores para tener acceso a ellos.
//Debemos registrar cada tipo de sensor por separado para poder obtener información de él.
//En primer lugar registramos el sensor de orientación
listaSensores = miSensorManager.getSensorList(Sensor.TYPE_ORIENTATION);
Sensor sensorOrientacion = listaSensores.get(0);
miSensorManager.registerListener(this, sensorOrientacion, SensorManager.SENSOR_DELAY_UI);
//Después registramos el acelerómetro
listaSensores = miSensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
Sensor sensorAcelerómetro = listaSensores.get(0);
miSensorManager.registerListener(this, sensorAcelerómetro, SensorManager.SENSOR_DELAY_UI);
//Después registramos la brújula
listaSensores = miSensorManager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
Sensor sensorBrújula = listaSensores.get(0);
miSensorManager.registerListener(this, sensorBrújula, SensorManager.SENSOR_DELAY_UI);
//Por último registramos el sensor de temperatura
listaSensores = miSensorManager.getSensorList(Sensor.TYPE_TEMPERATURE);
Sensor sensorTemperatura = listaSensores.get(0);
miSensorManager.registerListener(this, sensorTemperatura, SensorManager.SENSOR_DELAY_UI);
```

En este caso hemos registrado los 4 sensores de los que se nos informaba que disponía el dispositivo con el que trabajamos. En nuestras aplicaciones solamente es necesario registrar aquel o aquellos sensores con los que vayamos a trabajar.

El funcionamiento del método registerListener() es el siguiente:

1. El primer parámetro es la instancia de la clase que implementa el SensorEventListener. En el siguiente apartado vemos cómo implementar esta interfaz.
2. El segundo parámetro es el sensor que estamos registrando.
3. El tercer parámetro indica al sistema con qué frecuencia nos gustaría recibir actualizaciones del sensor. Los valores posibles de menor a mayor frecuencia son: SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_GAME y SENSOR_DELAY_FASTEST.

1.4.3. Obtener datos de los sensores

Para poder recibir los datos de los sensores debemos implementar dos métodos (onAccuracyChanged y onSensorChanged) de la interfaz SensorEventListener.

Tras implementar dicha interfaz en la clase:

```
public class EjemploSensor extends Activity implements SensorEventListener{
```

Implementamos los 2 métodos necesarios:

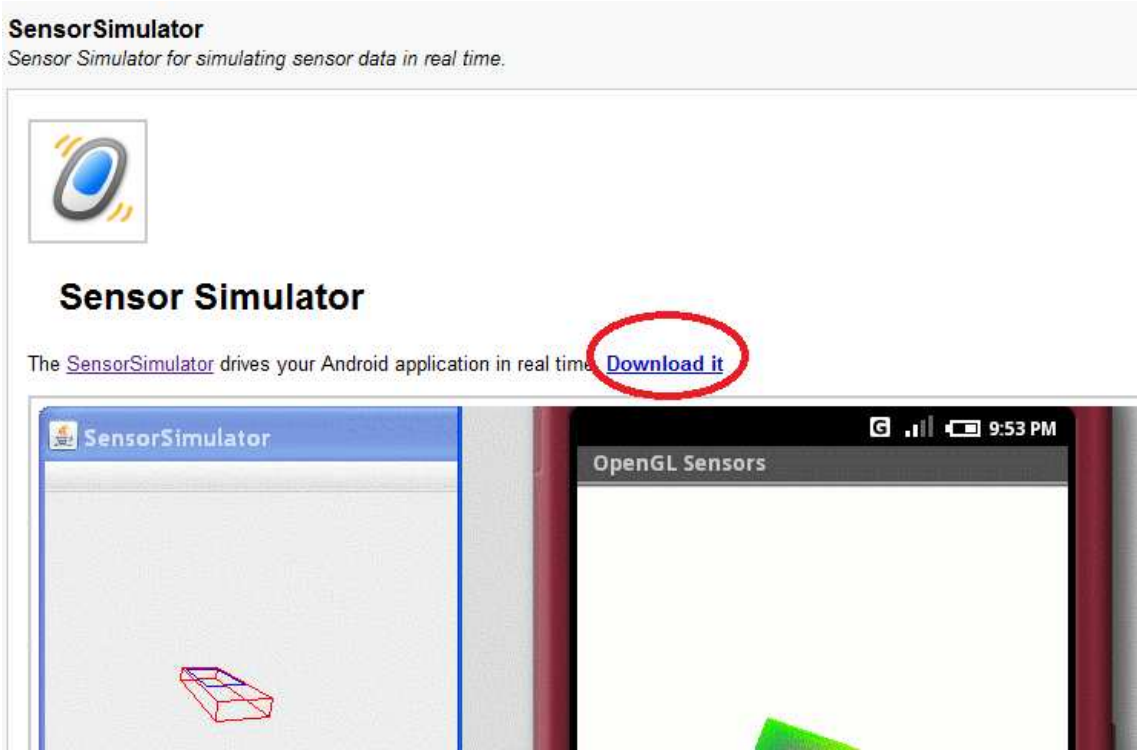
```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
    // TODO Auto-generated method stub

}

@Override
public void onSensorChanged(SensorEvent evento) {
    //Cada sensor puede provocar que un hilo pase por aquí
    //así que sincronizamos el acceso.
    synchronized (this) {
        switch (evento.sensor.getType()) {
        case Sensor.TYPE_ORIENTATION:
            for (int i=0 ; i<3 ; i++) {
                mostrar("Orientación " + i + ": " + evento.values[i]);
            }
            break;
        case Sensor.TYPE_ACCELEROMETER:
            for (int i=0 ; i<3 ; i++) {
                mostrar("Acelerómetro " + i + ": " + evento.values[i]);
            }
            break;
        case Sensor.TYPE_MAGNETIC_FIELD:
            for (int i=0 ; i<3 ; i++) {
                mostrar("Brújula " + i + ": " + evento.values[i]);
            }
            break;
        default:
            for (int i=0 ; i<evento.values.length ; i++) {
                mostrar("Temperatura " + i + ": " + evento.values[i]);
            }
            break;
        }
    }
```

En el momento en el que el evento se dispara en el método onSensorChanged comprobamos qué sensor lo ha causado y leemos los datos de dicho sensor.

1.4.4.Emulación de sensores para realizar pruebas

Para utilizar los sensores en nuestras aplicaciones y poder realizar las pruebas tenemos dos alternativas. En primer lugar, si disponemos de un terminal físico podemos utilizarlo para hacer las pruebas. En caso de no disponer de dicho terminal físico, lo que podemos hacer es instalar un software de emulación que permita realizar las pruebas sobre el emulador.

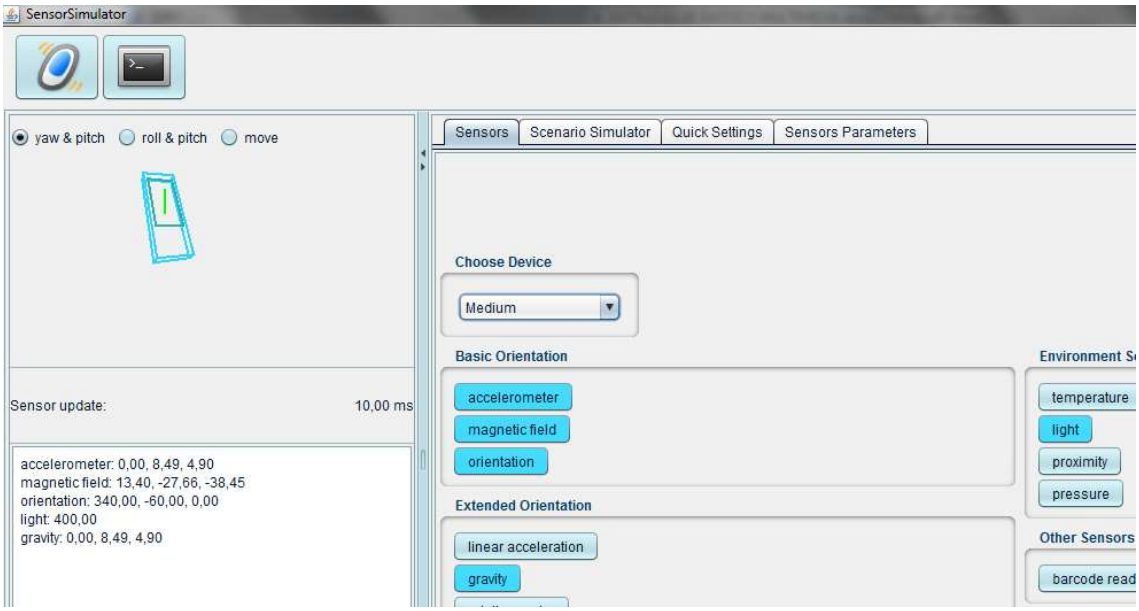
Este software de emulación lo podemos encontrar en http://code.google.com/p/openintents/wiki/SensorSimulator. En primer lugar descargamos el software de dicha página.



La versión que hemos descargado en esta ocasión es la sensorsimulator-2.0-rc1.zip.

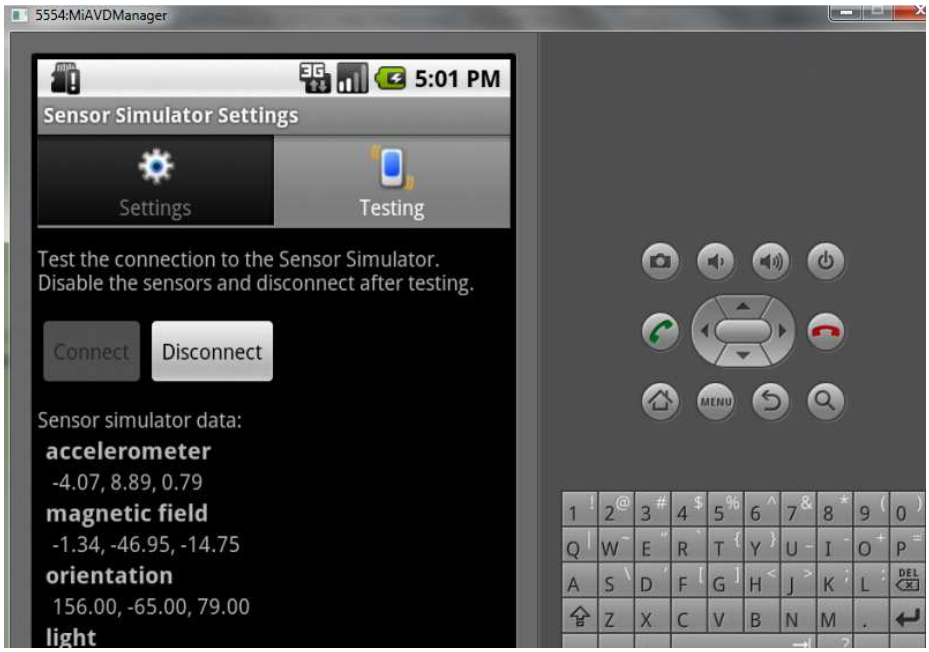
1.4.5.Pasos para instalar el simulador

1. Descomprimimos el archivo. Ejecutamos bin/sensorsimulator-2.0-rc1.jar. Obtendremos una pantalla parecida a esta.



2. Instalamos la aplicación en el emulador (antes debemos iniciar el emulador).
    a. Para instalar la aplicación en el emulador usamos la aplicación “adb” que se encuentra en “android-sdk/platform-tools/adb”.
    b. Primero listamos los dispositivos mediante el comando: adb devices
    c. Después lo instalamos mediante el comando: “adb -s <device> install sensorsimulator-1.1.1/bin/SensorSimulatorSettings-1.1.1.apk” (Debemos reemplazar <device> por el dispositivo mostrado en el paso anterior).
    d. Ahora en el emulador buscamos la aplicación instalada en el paso anterior: Sensor Simulator Settings y la iniciamos.

e. Introducimos IP y Socket (debe ser la misma que aparece en SensorSimulator que fue el programa abierto en el paso 1). Vamos a la pestaña Testing y le damos a Connect.



f. Si movemos la imagen del teléfono en la aplicación SensorSimulator, veremos cómo se actualizan los valores en el emulador de Eclipse. En este momento ya tenemos instalado y funcionando el simulador de sensores.

g. Ahora veremos cómo podemos utilizarlo en una aplicación realizada por nosotros:

   i. Lo primero que tenemos que hacer es crear un directorio en la raíz del proyecto de nuestra aplicación y copiar sensorsimulator-1.1.1/lib/sensorsimulator-lib-x.x.x.jar dentro del mismo. Después, en Eclipse, le damos con el botón derecho sobre el archivo que acabamos de copiar y seleccionamos la opción “Build Path” y luego “Add to Build Path”.

   ii. Comentamos los imports originales y agregamos los de las clases del package sensorsimulator:

```
/*
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorManager;
import android.hardware.SensorListener;
*/
import org.openintents.sensorsimulator.hardware.Sensor;
import org.openintents.sensorsimulator.hardware.SensorEvent;
import org.openintents.sensorsimulator.hardware.SensorEventListener;
import org.openintents.sensorsimulator.hardware.SensorManagerSimulator;
```

iii. En el método onCreate() se reemplaza el código:

```
//mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
```

por el código:

```
mSensorManager = SensorManagerSimulator.getSystemService(this, SENSOR_SERVICE);
mSensorManager.connectSimulator();
```

iv. Para registrar/desregistrar los sensores:

```
@Override
protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
            SensorManager.SENSOR_DELAY_FASTEST);
    mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
            SensorManager.SENSOR_DELAY_FASTEST);
    mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
            SensorManager.SENSOR_DELAY_FASTEST);
    mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensor.TYPE_TEMPERATURE),
            SensorManager.SENSOR_DELAY_FASTEST);
}

@Override
protected void onStop() {
mSensorManager.unregisterListener(this);
super.onStop();
}
```

v. Finalmente implementamos SensorEventListener:

```
public class Sensors extends Activity implements SensorEventListener{
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // TODO Auto-generated method stub

    }

        @Override
        public void onSensorChanged(SensorEvent event) {
            int sensor = event.type;
            float[] values = event.values;
            // Hacer algo con los datos del sensor
        }
        }
```

vi. Por último, en el archivo AndroidManifest.xml debemos agregar la siguiente línea que irá antes de la etiqueta <application>:

```
<uses-permission android:name="android.permission.INTERNET" />
```

NOTA IMPORTANTE: Una vez que tengamos la aplicación terminada y lista para pasar al teléfono hay que volver a cambiar los imports por los originales, comentar las líneas agregadas y descomentar las líneas originales. De otra forma la aplicación fallará.

1.4.6. Utilización de sensores en la aplicación Solobici Didáctica

A continuación se propone un código que puede ser utilizado para manejar la nave utilizando el sensor de orientación.

En primer lugar debemos implementar la interfaz SensorEventListener:

```
public class VistaJuego extends View implements SensorEventListener {
```

En segundo lugar registramos en el constructor el sensor:

```
//REGISTRO DE SENSORES
SensorManager miSensorManager = (SensorManager)getContext().getSystemService(Context.SENSOR_SERVICE);
List<Sensor> listaSensores = miSensorManager.getSensorList(Sensor.TYPE_ORIENTATION);
if (!listaSensores.isEmpty()) {
    Sensor sensorOrientacion = listaSensores.get(0);
    miSensorManager.registerListener(this, sensorOrientacion, SensorManager.SENSOR_DELAY_UI);
}
```

En tercer lugar añadimos los dos métodos que implementan la interfaz SensorEventListener:

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
    // TODO Auto-generated methos stub
}

private boolean hayValorInicial = false;
private float valorInicial;

@Override
public void onSensorChanged(SensorEvent evento) {
    float valor = evento.values[1];
    if (!hayValorInicial) {
        valorInicial = valor;
        hayValorInicial = true;
    }
    giroBici = (int) (valor-valorInicial)/3;
}
```

1.5. Lanzando ruedas en Solobici Didáctico

1.5.1. Ejercicio guiado 3

En este apartado vamos a introducir en el juego la posibilidad de que la bicicleta lance ruedas para intentar destruir coches. Para ello, añadimos las siguientes variables en la clase VistaJuego:

```
// RUEDA //
private Grafico rueda;
private static int VELOCIDAD_RUEDA = 12;
private boolean ruedaActiva;
private int distanciaRueda;
```

Creamos la variable drawableRueda mediante un gráfico vectorial introduciendo el siguiente código en el constructor:

```
// RUEDA
/*DIBUJO RUEDA
ShapeDrawable dRueda = new ShapeDrawable(new RectShape());
dRueda.getPaint().setColor(Color.WHITE);
dRueda.getPaint().setStyle(Style.STROKE);
dRueda.setIntrinsicWidth(15);
dRueda.setIntrinsicHeight(3);
graficoRueda = dRueda;
*/
graficoRueda = contexto.getResources().getDrawable(R.drawable.rueda);

rueda = new Grafico(this, graficoRueda);
ruedaActiva = false;
```

En el método onDraw() dibujamos la rueda en caso de que así lo indique la variable ruedaActiva.

```
//Dibujamos la rueda si lo indica la variable ruedaActiva
if (ruedaActiva)
    rueda.dibujaGrafico(canvas);
```

En el método actualizaMovimiento() añadimos las siguientes líneas:

```
//Movemos la rueda
if (ruedaActiva) {
    rueda.incrementaPos();
    distanciaRueda--;
    if (distanciaRueda<0) {
        ruedaActiva = false;
    } else {
        for (int i=0; i<Coches.size(); i++) {
            if (rueda.verificaColision(Coches.elementAt(i))) {
                destruyeCoche();
            }
        }
    }
}
```

Añadimos los siguientes dos métodos:

```
private void destruyeCoche(int i) {
    Coches.remove(i);
    ruedaActiva = false;
}

private void lanzarRueda() {
    rueda.setPosX(bici.getPosX() + bici.getAncho()/2 - rueda.getAncho()/2);
    rueda.setPosY(bici.getPosY() + bici.getAlto()/2 - rueda.getAlto()/2);
    rueda.setAngulo(bici.getAngulo());
    rueda.setIncX(Math.cos(Math.toRadians(rueda.getAngulo())) + VELOCIDAD_RUEDA);
    rueda.setIncY(Math.sin(Math.toRadians(rueda.getAngulo())) + VELOCIDAD_RUEDA);
    distanciaRueda = (int)Math.min(
            this.getWidth() / Math.abs(rueda.getIncX()),
            this.getHeight() / Math.abs(rueda.getIncY())) - 2;
    ruedaActiva = true;
}
```

Por último descomentamos las llamadas al método LanzarRueda() que dejamos en el momento de programar los eventos de teclado.

2. CICLO DE VIDA DE UNA APLICACIÓN ANDROID

2.1. Estados y eventos del ciclo de vida

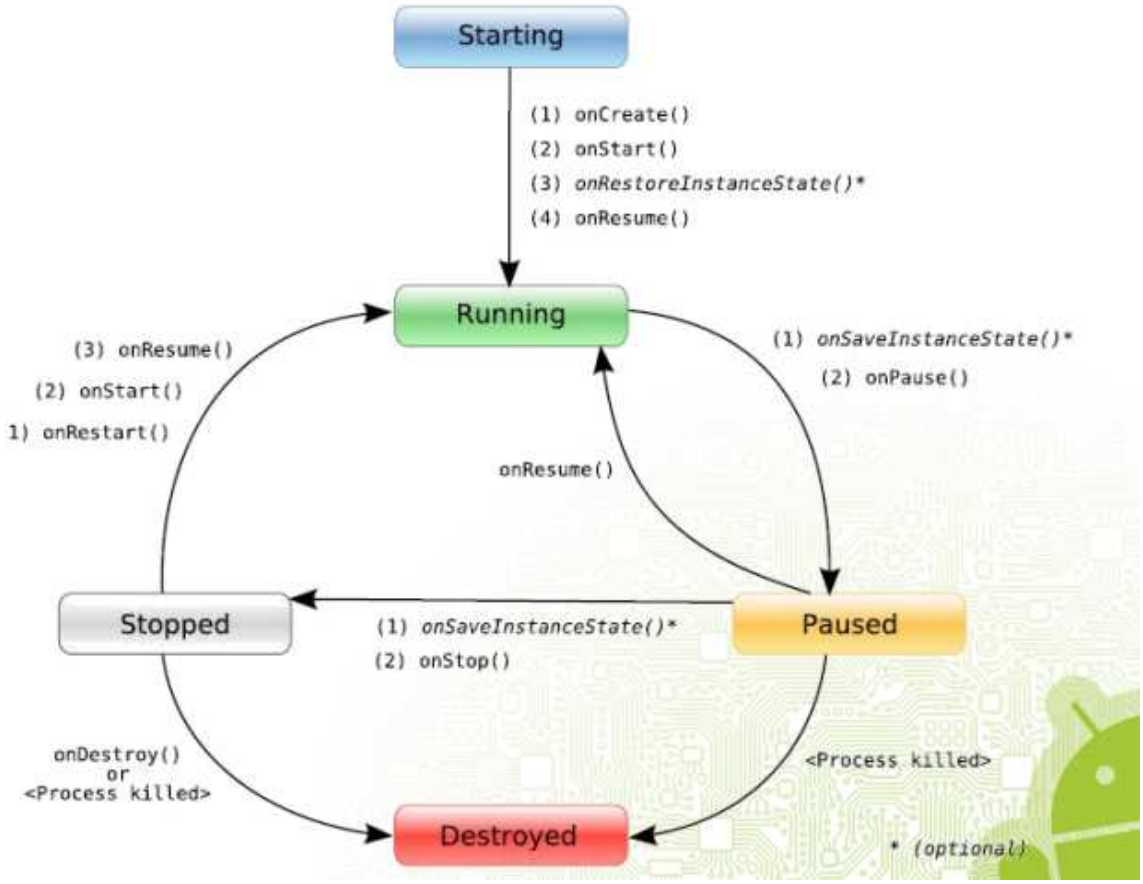
El ciclo de vida de una aplicación Android es bastante diferente al ciclo de vida de una aplicación en otros S.O., como Windows. La mayor diferencia es que, en Android el ciclo de vida es controlado completamente por el sistema, en lugar de ser controlado directamente por el usuario.

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, conocidos como actividades. Además de varias actividades una aplicación también puede contener servicios. Son las actividades las que realmente controlan el ciclo de vida de una aplicación, dado que el usuario no cambia de aplicación, sino de actividad. El sistema va a mantener una pila con las actividades previamente visualizadas, de forma que el usuario va a poder regresar a la actividad anterior pulsando la tecla “atrás”.

Una actividad en Android puede estar en uno de estos cuatro estados:

- **Activa (Running):** La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.
- **Visible (Paused):** La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada completamente, pasa a estar parada.
- **Parada (Stopped):** Cuando la actividad no es visible, se recomienda guardar el estado de la interfaz de usuario, preferencias, etc.
- **Destruida (Destroyed):** Cuando la actividad termina o es matada por el sistema Android, sale de la pila de actividades.

Cuando una actividad cambia de estado se producen eventos que pueden ser capturados por ciertos métodos de la actividad.



- **onCreate(Bundle)**
  - Se invoca cuando la Actividad se arranca por primera vez.
  - Se utiliza para tareas de inicialización como crear la interfaz de usuario de la Actividad.
  - Su parámetro es null o información de estado guardada previamente por onSaveInstanceState().
- **onStart()**
  - Se invoca cuando la Actividad va a ser mostrada al usuario.
- **onResume()**
  - Se invoca cuando la actividad va a empezar a interactuar con el usuario.
  - **Es un buen lugar para lanzar las animaciones y la música.**
- **onPause()**
  - Se invoca cuando la Actividad va a pasar a segundo plano porque otra Actividad ha sido lanzada para ponerse delante.
  - Es el lugar adecuado para almacenar los datos que estaban en edición.
- **onStop()**
  - Se invoca cuando la actividad va a dejar de ser visible y no se necesitará durante un tiempo.
  - Si hay escasez de recursos en el sistema, este método podría no llegar a ser invocado y la actividad ser destruida directamente.

- **onRestart()**
  - Se invoca cuando una actividad parada pasa a estar activa.
- **onDestroy()**
  - Se invoca cuando la Actividad va ser destruida.
  - Si hay escasez de recursos en el sistema, este método podría no llegar a ser invocado y la actividad ser destruida directamente.

En algunas ocasiones necesitaremos guardar el estado de una actividad. Por ejemplo, si hemos estado utilizando una actividad y cambiamos a otras, puede ser que el sistema elimine el proceso que ejecutaba la primera y al volver hayamos perdido la información.

En caso de tratarse de información extremadamente sensible, se recomienda el uso del método onPause() para guardar el estado y onResume() para recuperarlos.

Otra posibilidad que no tiene una fiabilidad tan grande pero resulta mucho más sencilla consiste en utilizar los siguientes dos métodos:

- **onSaveInstanceState(Bundle)**
  - Se invoca para permitir a la actividad guardar su estado.
  - Si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.
- **onRestoreInstanceState(Bundle)**
  - Se invoca para recuperar el estado guardado por onSaveInstanceState.

El siguiente ejemplo muestra cómo guardar la información de una variable tipo cadena de caracteres y entero.

```
String cadena;
int pos;

protected void onSaveInstanceState(Bundle guardarEstado) {
    super.onSaveInstanceState();
    guardarEstado.putString("variable", cadena);
    guardarEstado.putInt("posicion", pos);
}

protected void onRestoreInstanceState(Bundle recuperaEstado) {
    super.onRestoreInstanceState(recuperaEstado);
    cadena = recuperaEstado.getString("variable");
    pos = recuperaEstado.getInt("posicion");
}
```

Como ventaja de utilizar estos métodos tenemos que no se debe buscar ningún método de almacenamiento persistente ya que será el sistema quien lo haga. Como desventaja tenemos que el sistema no garantiza que estos métodos sean llamados.

2.2. Eventos del ciclo de vida en la aplicación Solobici Didáctico

Si pasamos a segundo plano nuestra aplicación el teléfono sobre el que se ejecuta puede funcionar más lentamente ya que el thread que se encarga de mover los objetos sigue funcionando.

2.2.1.Ejercicio Guiado 4

Para evitar este funcionamiento más lento podemos incluir el siguiente código en nuestra aplicación.

1. Definir al principio de la clase VistaJuego las siguientes variables globales:

```
//Controlar si la aplicación está en segundo plano
private boolean corriendo = false;
//Controlar si la aplicación está en pausa
private boolean pausa;
```

2. Añadir en el constructor de la clase VistaJuego:

```
//CONTROL DEL HILO DEL JUEGO
corriendo = true;
```

3. Modificar la clase HiloJuego para que contenga el siguiente código:

```
private class HiloJuego extends Thread{
    @Override
    public void run() {
        while (corriendo) {
            actualizaMovimiento();
        }
    }
}
```

4. Añadir los siguientes métodos en la clase VistaJuego:

```
public HiloJuego getHilo() {
    return hiloJuego;
}

public void setCorriendo(boolean corriendo) {
    this.corriendo = corriendo;
}

public void setPausa(boolean pausa) {
    this.pausa = pausa;
}
```

5. Añadir el siguiente atributo en la clase Juego.java:

```
private VistaJuego vistaJuego;
```

6. Añadir o modificar los siguientes métodos en la clase Juego.java:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.juego);
    vistaJuego = (VistaJuego) findViewById(R.id.VistaJuego);
}

@Override
protected void onDestroy() {
    //Al poner la variable corriendo a false permitimos que el thread pueda acabar.
    vistaJuego.setCorriendo(false);
    super.onDestroy();
}

@Override
protected void onPause() {
    super.onPause();
    //Ponemos el thread en suspensión.
    vistaJuego.setPausa(true);
}
@Override
protected void onResume() {
    super.onResume();
    //Continuamos ejecutando el thread.
    vistaJuego.setPausa(false);
}
```

Con este código por lo tanto evitamos que el terminal funcione más lento en caso de dejar nuestra aplicación en segundo plano para atender o ejecutar otra aplicación. Esto es así ya que pasa a estados de pausa en el momento de pasar a segundo plano y vuelve a activarse al pasar a primer plano.

3. MULTIMEDIA

En Android podemos reproducir audio y vídeo desde diferentes orígenes:

1. Desde un fichero almacenado en el dispositivo.
2. Desde un recurso que esté dentro del paquete de la aplicación.
3. Desde un stream que se lea desde una conexión de red.

Las clases de Android que permiten acceder a los servicios multimedia son:

MediaPlayer	Reproducción de audio/vídeo desde ficheros o streams.
MediaControler	Visualiza controles estándar para mediaPlayer.
VideoView	Vista que permite la reproducción de vídeo.
SoundPool	Maneja y reproduce una colección de recursos de audio.
AsyncPlayer	Reproduce lista de audios desde un thread distinto al nuestro.
MediaRecorder	Permite grabar audio y vídeo.
FaceDetector	Identifica la cara de la gente en un bitmap.
AudioManager	Gestiona varias propiedades del sistema (volumen, tonos, …)
AudioTrack	Reproduce un búfer de audio PCM directamente por hardware.

3.1. La clase VideoView para reproducir un vídeo

Con la clase VideoView podemos incluir un vídeo en una aplicación creando una vista de tipo VideoView. Lo podemos ver en el siguiente ejemplo (Aplicación proporcionada con el código fuente **EjemploVideo**):

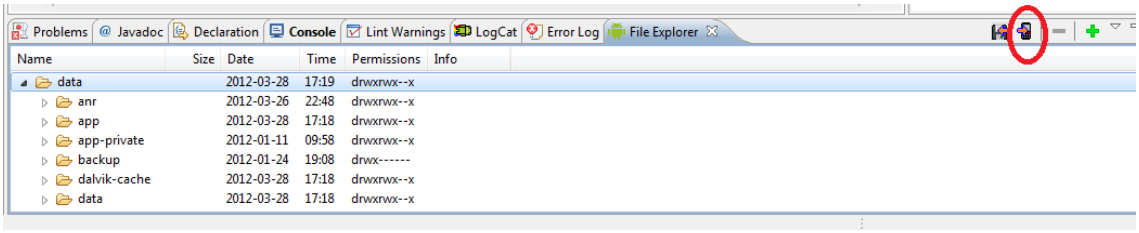
1. Creamos una nueva aplicación de nombre EjemploVideo.
2. Reemplazamos el fichero "res/layout/main.xml" por el siguiente código donde introducimos el elemento VideoView en el layout que se encargará de mostrar el vídeo que deseamos:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <VideoView
        android:layout_width="320px"
        android:layout_height="240px"
        android:id="@+id/vistaVideo" />

</LinearLayout>
```

3. Reemplazar el fichero “EjemploVideo.java” por el siguiente código:

```
package org.ejemplo.EjemploVideo;

import android.app.Activity;

public class EjemploVideo extends Activity {
    private VideoView vistaVideo;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        vistaVideo = (VideoView)findViewById(R.id.vistaVideo);
        //De forma alternativa si queremos un streaming debemos usar:
        //vistaVideo = setVideoURI(Uri.parse(cadenaConURL);
        /*
        vistaVideo.setVideoURI(Uri.parse(
                "http://www.oles3gp.com/kylie3gp/stereo/" +
                "Kylie%20Minogue%20-%202%20Hearts%20%5bOFFICIAL%20VIDEO%5d-stereo.3gp"));
        */
        //En el siguiente método estamos indocando un fichero local.
        //Para ello debemos almacenarlo en el simulador utilizando la opción
        //del menú Window->Show View->Others...->Android->File Explorer
        //Podéis probar con la siguiente URL
        vistaVideo.setVideoPath("/data/video.mp4");
        vistaVideo.setMediaController(new MediaController(this));
        vistaVideo.start();
        vistaVideo.requestFocus();
    }
}
```

4. Almacenamos un fichero mp4 dentro de la carpeta “/data” con el nombre video.mp4 utilizando el botón para cargar ficheros en el dispositivo. **Para ver el sistema de ficheros del dispositivo, el emulador debe estar funcionando**.



5. Ejecutamos la aplicación y veremos el vídeo. Si el vídeo quedase parado en un fotograma y pudiésemos oír el sonido del vídeo se trata de un problema del emulador. Podemos intentar probar con otro emulador como por ejemplo “Google APIs (Google Inc) para plataforma 2.1”. También podéis probar a pasar la aplicación a un terminal real si queréis comprobar el funcionamiento correcto.

3.2. La clase MediaPlayer

La clase MediaPlayer es la que se encarga de la reproducción multimedia en Android.

Un objeto MediaPlayer puede pasar por varios estados:

initialized	Inicializados sus recursos.
preparing	Preparando la reproducción.
prepared	Preparado para reproducir.
started	Reproduciendo.
paused	En pausa.
stopped	Parado.
playback completed	Reproducción completada.
end	Finalizado.
error	Con error.

3.2.1.Audio de fondo

En el caso en el que nos interese reproducir un audio siempre en nuestra aplicación es interesante incluirlo en el paquete (.apk) de la aplicación y tratarlo como un recurso más.

Para poder hacer esto (en la aplicación que estamos desarrollando "Solobici Didáctico) los pasos a seguir serían los siguientes:

1. Abrir la aplicación Solobici Didáctico.
2. Crear una nueva carpeta con nombre "raw" dentro de la carpeta "res".
3. Arrastrar a su interior el fichero "audio.mid".
4. Abrir el fichero "gen/R.java y comprobar que se ha creado una nueva referencia de recurso con nombre audio.
5. Añadir las siguientes líneas al final del método onCreate() de la clase Juego.java:

```
//Música de fondo
MediaPlayer miMediaPlayer = MediaPlayer.create(this, R.raw.audio);
miMediaPlayer.start();
```

6. Comprobar que funciona correctamente.

3.2.2.Ejercicio 2

Al salir de la actividad, la música sigue sonando un cierto tiempo. Intentar evitar este problema utilizando los métodos onResume() y onPause(), por ejemplo poniendo la música en pausa cuando la actividad no esté activa.

3.2.3.Ejercicio 3

Incorporar efectos de audio al videojuego reproduciendo el fichero explosión.mp3 cuando alcancemos un coche con una rueda.

3.2.4.Ejercicio 4 (Opcional)

Cuando cambiamos la inclinación del teléfono, el audio comienza a reproducirse desde el principio. Esto es debido a que la actividad es destruida y vuelta a construir, generando un evento onCreate(). Trata que el audio no se reinicie utilizando los métodos onSaveInstanceState y onRestoreInstanceState(Bundle).