TEMA 3 (Parte 1): GRÁFICOS Y MENÚS EN ANDROID

Contenido

1. GRÁFICOS EN ANDROID

La API gráfica de Android nos proporciona funciones para manipular imágenes, gráficos vectoriales, animaciones, trabajo con texto y gráficos en 3D.

1.1. Elementos gráficos en Android

Las clases principales que necesitaremos para incluir gráficos en nuestras aplicaciones serán Canvas y Drawable.

1.1.1.Canvas

Esta clase representa una superficie donde podemos dibujar. Nos ofrece métodos para dibujar líneas, círculos, texto, etc. Para ello tendremos que utilizar un pincel (Paint) donde se definirá el color, grosor del trazo, transparencia, etc. de lo que vayamos a dibujar.

Algunos de los métodos más importantes de la clase Canvas se enumeran a continuación:

Para dibujar figuras geométricas:
drawCircle(float cx, float cy, float radio, Paint pincel);
drawOval(RectF ovalo, Paint pincel);
drawRect(RectF recta, Paint pincel);
drawPoint(float x, float y, Paint pincel);

Para dibujar líneas y arcos:
drawLine(float iniX, float iniY, float finX, float finY, Paint pincel);
drawLines(float [] puntos, Paint pincel);
drawArc(RectF ovalo, float iniAngulo, float angulo, Boolean usarCentro, Paint pincel);
drawPath(Path trazo, Paint pincel);

Para dibujar texto:
drawText(String texto, float x, float y, Paint pincel);
drawTextOnPath(String texto, Path trazo, float desplazHoriz, float desplazVert, Paint pincel);
drawPosText(String texto, float[] posicion, Paint pincel);

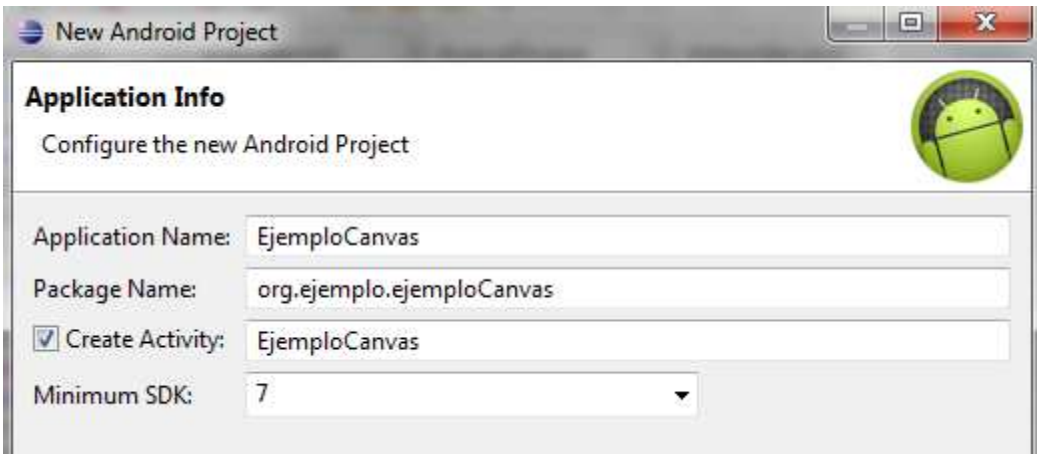
Para rellenar todo el Canvas:
drawColor(int color);
drawARGB(int alfa, int rojo, int verde, int azul);
drawPaint(Paint pincel);

Para dibujar imágenes: drawBitmap(Bitmap bitmap, Matrix matriz, Paint pincel);

Para averiguar el tamaño del Canvas:
int getHeight();
int getWidth();

En el siguiente ejemplo podemos apreciar el trabajo con Canvas.

Creamos un proyecto con los siguientes datos:



El código de la actividad que se genera lo reemplazamos por el siguiente:

```
package org.ejemplo.ejemploCanvas;

import android.app.Activity;

public class EjemploCanvas extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new EjemploView(this));
    }

    public class EjemploView extends View {
        public EjemploView(Context contexto) {
            super(contexto);
        }
        @Override
        protected void onDraw(Canvas canvas) {
            //Dentro de este método utilizamos los métodos para dibujar

        }
    }
}
```

Casi todos los métodos para dibujar en el Canvas utilizan un objeto de la clase Paint. Esta clase nos permite definir el color, estilo o grosor de un gráfico vectorial.

Como ejemplo podemos escribir el siguiente código dentro de OnDraw para conseguir dibujar un círculo en el Canvas:

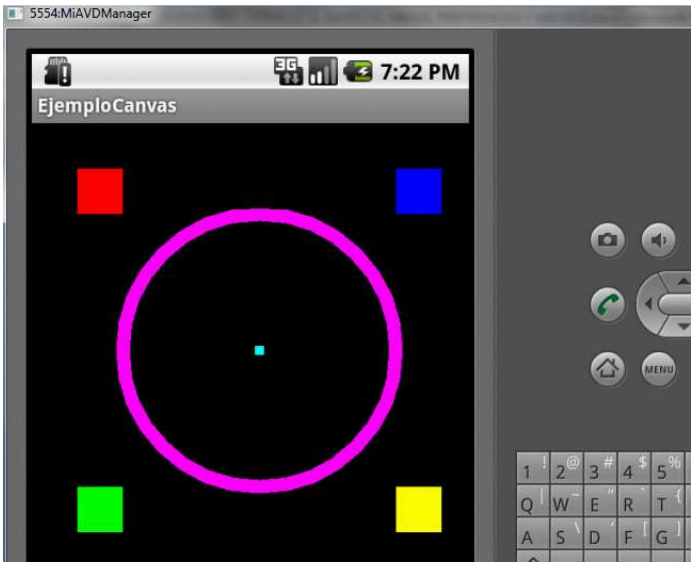
```
protected void onDraw(Canvas canvas) {
    //Dentro de este método utilizamos los métodos para dibujar

    //Creamos un pincel con el que elegir color, trazo, estilo, etc.
    Paint pincel = new Paint();
    //Seleccionamos el color azul para el pincel
    pincel.setColor(Color.BLUE);
    //Establecemos el grosor del pincel
    pincel.setStrokeWidth(15);
    //Establecemos el estilo del trazo
    pincel.setStyle(Style.STROKE);
    canvas.drawCircle(150, 150, 80, pincel);
}
```

1.1.1.1. Ejercicio 1

Realiza el siguiente dibujo probando para ello con otros métodos de dibujo como drawPoint(), drawCircle(), drawRect(), etc.

Enviar una captura de pantalla como solución al ejercicio.



1.1.2.Drawable

Esta clase representa "algo que puede ser dibujado". Es una clase abstracta que se extiende para definir objetos gráficos más específicos. Muchos de ellos se pueden definir como recursos en ficheros XML.

Entre las subclases más importantes de Drawable nos encontramos con las siguientes:

BitmapDrawable	Imagen basada en un fichero gráfico (PNG o JPG). Etiqueta XML <bitmap>.
ShapeDrawable	Permite realizar un gráfico a partir de primitivas vectoriales, como formas básicas (círculos, cuadrados, …) o trazados (Path).
LayerDrawable	Contiene un array de Drawable que son visualizados según el orden del array. Etiqueta XML <layer-list>.
StateListDrawable	Contiene una lista de objetos Drawable, y usando una máscara de bits podemos seleccionar los objetos visibles. Etiqueta XML <selector>.
GradienDrawable	Degradado de color que puede ser usado en botones o fondos.
TransitionDrawable	Una extensión de LayerDrawables que permite una transición entre la primera y la segunda capa. Para iniciar la transición hay que llamar a startTransition() y para visualizar la primera capa hay que llamar a resetTransition(). Etiqueta XML <transition>.
AnimationDrawable	Permite crear animaciones frame a frame a partir de una serie de objetos Drawable. Etiqueta XML <animation-list>.

La clase Drawable ofrece una serie de métodos para indicar cómo ha de dibujarse. Algunos de los más importantes serían los siguientes:

setBounds(Rect)	Permite indicar el rectángulo donde ha de dibujarse el Drawable.
setState(int[])	Permite indicar al Drawable en qué estado ha de ser dibujado, por ejemplo "con foco", "seleccionado", etc. Algunos Drawable cambian su representación según este estado.
setLevel(int)	Permite implementar un controlador sencillo para indicar cómo se representará el Drawable. Por ejemplo, un nivel puede ser interpretado como una batería de niveles o un nivel de progreso. Algunos Drawable modifican la imagen basándose en el nivel.

1.1.2.1. BitmapDrawable

Para añadir gráficos a una aplicación lo más sencillo es incluir el gráfico en la carpeta res/drawable del proyeto.

Los formatos soportados son PNG, JPEG y GIF. Se aconseja PNG y se desaconseja GIF.

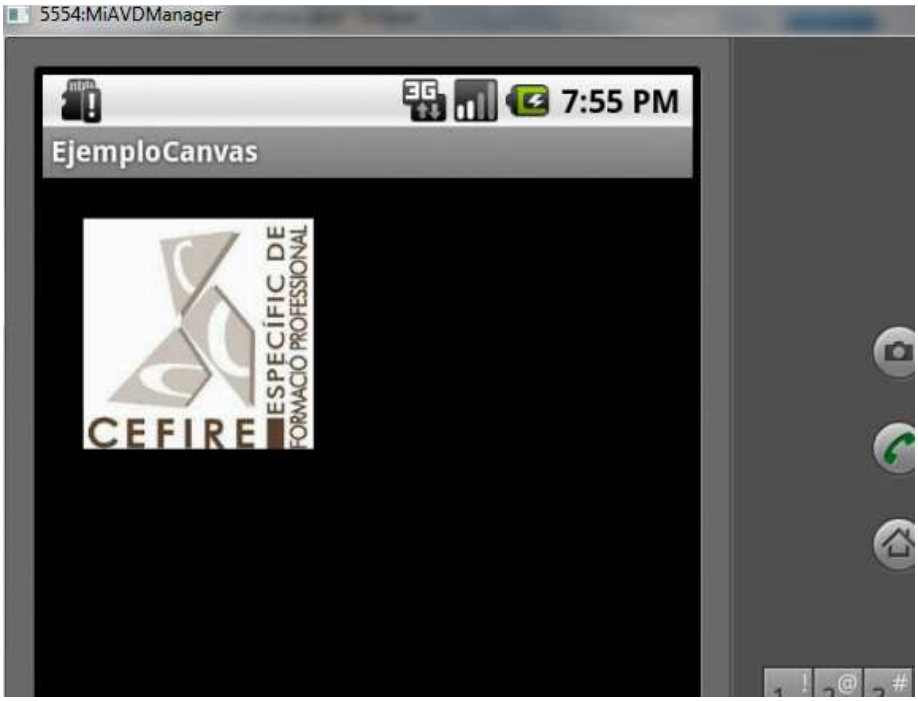
Podemos incluir una imagen en la carpeta res/drawable y ver el funcionamiento incluyendo las siguientes líneas en el constructor de la clase EjemploView creada anteriormente:

```
public class EjemploCanvas extends Activity {
    private BitmapDrawable miImagen;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new EjemploView(this));
    }

    public class EjemploView extends View {
        public EjemploView(Context contexto) {
            super(contexto);
            Resources res = contexto.getResources();
            miImagen = (BitmapDrawable) res.getDrawable(R.drawable.logo_cefire);
            miImagen.setBounds(new Rect(30,30,200,200));
        }
        @Override
        protected void onDraw(Canvas canvas) {
            //Dentro de este método utilizamos los métodos para dibujar
            //BitmapDrawable
            miImagen.draw(canvas);
```

Obtendremos algo parecido a esto:



También podemos asignar una imagen a un objeto de la clase ImageView definiéndolo en XML de la siguiente forma:

```
<ImageView
    android:id="@+id/imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/logo_cefire" />
```

1.1.2.2. GradienDrawable

Podemos definir otro tipo de Drawable creando el siguiente fichero en *res/drawable/degradado.xml.* Se trata de un gradiente, o sea, el paso de un color a otro con un determinado ángulo, obteniendo diferentes diseños según el ángulo especificado.

```
<!-- El atributo ángulo debe ser múltiplo de 45 -->
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <gradient
        android:startColor="#FFFFFF"
        android:endColor="#0022AA"
        android:angle="225"/>
</shape>
```

Ahora podemos utilizar este nuevo recurso para cambiar el fondo de una vista. Tenemos dos formas de hacerlo:

1. Introducir la siguiente línea en el constructor de una vista:

```
//Cambiamos el fondo de una vista
setBackgroundResource(R.drawable.degradado);
```

2. O bien, definir el siguiente parámetro en la definición del Layout en XML.

```
android:background="@drawable/degradado"
```

1.1.2.2.1. Ejercicio 2

Modificar el fondo de la aplicación SOLOBICI siguiendo los siguientes pasos:

- Copiar el recurso degradado.xml visto en el anterior punto a la aplicación SOLOBICI.
- Asignar al fondo del Layout definido en main.xml el recurso que acabas de copiar.
- Como el título de la aplicación no se verá demasiado bien, cambiar el color del texto a este título.

Enviar una captura de pantalla del resultado obtenido.

1.1.2.3. TransitionDrawable

Al igual que hemos creado un recurso de gradiente mediante GradienDrawable, podemos crear un recurso TransitionDrawable que nos permite pasar de una imagen a otra pasados X segundos.

En el siguiente ejemplo (Aplicación EjemploTransicion) podemos ver el funcionamiento.

Los pasos seguidos son:

1. Copiar en las carpetas res/drawable la imagen inicial y la imagen final sobre las que queremos crear una transición. (En el proyecto que os paso podéis encontrar las dos imágenes que he utilizado. Sí, es mi mano :) la que aparece).
2. Crear el fichero res/drawable/transicion.xml e incluir el siguiente código en el mismo:

```
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/imagentransicion1" />
    <item android:drawable="@drawable/imagentransicion2" />
</transition>
```

Donde "imagentransicion1" e "imagentransicion2" son los nombres de las dos imágenes copiadas en el anterior paso.

3. Reemplazar el método onCreate() en el fichero Java por el siguiente:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ImageView imagen = new ImageView(this);
    setContentView(imagen);
    TransitionDrawable mi_transicion = (TransitionDrawable)
            getResources().getDrawable(R.drawable.transicion);
    imagen.setImageDrawable(mi_transicion);
    mi_transicion.startTransition(2000);
}
```

Como se puede comprobar en primer lugar se crea un objeto ImageView que contendrá cada una de las imágenes de la transición. Después se carga el recurso con la definición de la transición utilizando "R.drawable.transicion". Por último se inicia la transición dando como parámetro el número de milisegundos (2000, o sea, 2 segundos) que debe transcurrir en la transición.

1.1.2.4. ShapeDrawable

La clase ShapeDrawable permite dibujar gráficos a partir de formas primitivas. Utilizaremos un objeto ShapeDrawable para crear una vista a medida. Tenéis el ejemplo en el proyecto denominado EjemploShapeDawable.

```
public class EjemploShapeDrawable extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new VistaAMedida(this));
    }

    public class VistaAMedida extends View {
        private ShapeDrawable miDrawable;
        public VistaAMedida(Context contexto) {
            super(contexto);
            int x=10, y=10;
            int ancho=300, alto=50;
            miDrawable = new ShapeDrawable(new OvalShape());
            miDrawable.getPaint().setColor(0xff0000ff);
            miDrawable.setBounds(x, y, x + ancho, y + alto);
        }

        protected void onDraw(Canvas canvas) {
            miDrawable.draw(canvas);
        }
    }
}
```

En este caso creamos una elipse que pasamos al constructor de ShapeDrawable. Después le ponemos un color y le asignamos mediante el método “setBounds” su posición y su tamaño.

Las vistas pueden ser reutilizadas. De esta forma, si queremos dibujar un óvalo, incluiremos el siguiente código en el Layout correspondiente.

```
<org.ejemplo.ejemploshapedrawable.VistaAMedida
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
/>
```

1.1.2.5. AnimationDrawable

La clase AnimationDrawable nos permite crear una animación a partir de una serie de fotogramas.

Para ello crearemos un fichero XML que defina esta animación y lo guardaremos en res/anim/animación.xml.

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
                android:oneshot="false">
    <item android:drawable="@drawable/fotograma1"
        android:duration="500" />
    <item android:drawable="@drawable/fotograma2"
        android:duration="500" />
    <item android:drawable="@drawable/fotograma3"
        android:duration="500" />
</animation-list>
```

Disponéis del código fuente de este ejemplo de animación en el proyecto denominado EjemploAnimationDrawable cuyo código fuente analizamos a continuación.

```
package org.ejemplo.aguilar;


import android.app.Activity;
import android.graphics.Color;
import android.graphics.drawable.AnimationDrawable;
import android.os.Bundle;
import android.view.MotionEvent;
import android.widget.ImageView;

public class EjemploAnimationDrawable extends Activity {
    AnimationDrawable animacion;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Obtenemos el recurso creado en animacion.xml
        animacion = (AnimationDrawable) getResources().getDrawable(R.anim.animacion);
        //Creamos una vista que contendrá una imagen
        ImageView imagen = new ImageView(this);
        //Le ponemos color de fondo
        imagen.setBackgroundColor(Color.WHITE);
        //Cargamos en lugar de una imagen, una animación.
        imagen.setImageDrawable(animacion);
        setContentView(imagen);
    }
    public boolean onTouchEvent(MotionEvent evento) {
        //Al realizar una pulsación en pantalla
        if (evento.getAction() == MotionEvent.ACTION_DOWN) {
            //Ponemos en marcha la animación
            animacion.start();
            return true;
        }
        return super.onTouchEvent(evento);
    }
}
```

En el anterior ejemplo, hemos aprovechado para introducir un escuchador de evento que se activa al realizar una pulsación en la pantalla (MotionEvent.ACTION_DOWN) poniendo en marcha la animación (animación.start()).

1.2. Ejercicio Guiado 1

En este ejercicio vamos a crear la actividad principal de la aplicación “Solobici” que vamos a ir desarrollando a lo largo del curso. Esta pantalla será en la que aparezcan los elementos del juego, o sea, la bicicleta que controlaremos y los coches que estarán a nuestro alrededor.

1.2.1.Layout para la pantalla del juego

Para la pantalla del juego utilizaremos un Layout que crearemos en el archivo res/layout/juego.xml y tendrá el siguiente código:

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <juego.solobici.VistaJuego
        android:id="@+id/VistaJuego"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:focusable="true"
        android:background="@drawable/fondo" />
</LinearLayout>
```

Como se puede apreciar hemos utilizado una vista creada por nosotros, y que hemos denominado juego.solobici.VistaJuego. Como hemos utilizado el parámetro android:background asociado al recurso @drawable/fondo, debemos poner el recurso fondo.png (que tenéis en el fichero de códigos fuente que os he pasado) en las carpetas drawable correspondientes. También copiaremos los recursos bici.png, coche.png y rueda.png a las carpetas res/drawable.

1.2.2.Creación de la actividad principal (Juego.java)

Creamos en primer lugar la actividad principal que denominaremos Juego.java e incluiremos en el paquete (juego.solobici) de fuentes Java de la aplicación. Esta actividad tendrá el siguiente código:

```
package juego.solobici;

import android.app.Activity;
import android.os.Bundle;

public class Juego extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.juego);
    }
}
```

1.2.3.Creación de la clase Gráfico (Grafico.java)

Crearemos una clase (Grafico.java) que representará cualquier gráfico a desplazar por pantalla (bici, coches, ruedas, …). El código de esta clase será el siguiente:

```
package juego.solobici;

import android.graphics.Canvas;
import android.graphics.drawable.Drawable;
import android.view.View;

public class Grafico {
    private Drawable drawable; // Imagen que dibujaremos
    private double posX, posY; // Posición en la pantalla
    private double incX, incY; // Velocidad de desplazamiento
    private int angulo, rotacion;// Ángulo y velocidad rotación
    private int ancho, alto; // Dimensiones de la imagen
    private int radioColision; // Para determinar si chocamos con algún objeto
    // Vista donde dibujamos el gráfico
    private View view;
    // Para determinar el espacio a borrar
    public static final int MAX_VELOCIDAD = 20;

    //Inicializamos los atributos de esta clase
    public Grafico(View view, Drawable drawable) {
        this.view = view;
        this.drawable = drawable;
        ancho = drawable.getIntrinsicWidth();
        alto = drawable.getIntrinsicHeight();
        radioColision = (alto + ancho) / 4;
    }

    //Dibujamos el gráfico en su posición actual
    public void dibujaGrafico(Canvas canvas) {
        canvas.save();
        int x = (int) (posX + ancho / 2);
        int y = (int) (posY + alto / 2);
        canvas.rotate((float) angulo, (float) x, (float) y);
        drawable.setBounds((int) posX, (int) posY, (int) posX + ancho,
                (int) posY + alto);
        drawable.draw(canvas);
        canvas.restore();
        //Calculamos el área donde no podrán solaparse/chocar
        //otros gráficos con este
        int rInval = (int) distanciaE(0, 0, ancho, alto) / 2 + MAX_VELOCIDAD;
        view.invalidate(x - rInval, y - rInval, x + rInval, y + rInval);
    };
```

```
//Correcciones de posición en caso de que el gráfico se salga de la pantalla
//En estos casos aparecemos por el otro lado de la pantalla
public void incrementaPos() {
    posX += incX;
    // Si salimos de la pantalla, corregimos posición
    if (posX < -ancho / 2) {
        posX = view.getWidth() - ancho / 2;
    }
    if (posX > view.getWidth() - ancho / 2) {
        posX = -ancho / 2;
    }
    posY += incY;
    // Si salimos de la pantalla, corregimos posición
    if (posY < -alto / 2) {
        posY = view.getHeight() - alto / 2;
    }
    if (posY > view.getHeight() - alto / 2) {
        posY = -alto / 2;
    }
    angulo += rotacion; // Actualizamos ángulo
}
//Nos devuelve la distancia entre dos objetos Grafico
public double distancia(Grafico g) {
    return distanciaE(posX, posY, g.posX, g.posY);
}

//Nos devuelve si se produce o no colisión entre dos objetos
public boolean verificaColision(Grafico g) {
    return (distancia(g) < (radioColision + g.radioColision));
}

public static double distanciaE(double x, double y, double x2, double y2) {
    return Math.sqrt((x - x2) * (x - x2) + (y - y2) * (y - y2));
}
```

Como debemos acceder a los atributos privados de esta clase, crearemos los métodos getters y setters. Para ello ponemos el cursor al final de la clase (justo antes de la última llave) y pulsamos con el botón derecho seleccionando en el menú desplegable Source->Generate Getters and Setters.

1.2.4.Creación de la vista VistaJuego (VistaJuego.java)

En esta vista será donde dibujaremos todos los gráficos necesarios en la pantalla del juego.

```
package juego.solobici;

import java.util.Vector;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.drawable.Drawable;
import android.util.AttributeSet;
import android.view.View;

public class VistaJuego extends View {
    //  COCHES  //
    private Vector<Grafico> Coches; //Vector con los Coches
    private int numCoches = 5;      //Número inicial de Coches
    private int numMotos = 3;       //Fragmentos/Motos en que se dividirá un Coche

    public VistaJuego(Context contexto, AttributeSet atributos) {
        super(contexto, atributos);
        Drawable graficoBici, graficoCoche, graficoRueda;
        //Obtenemos la imagen/recurso del coche
        graficoCoche = contexto.getResources().getDrawable(R.drawable.coche);

        //Creamos un vector para contener todos los coches que irán por la pantalla
        //y lo rellenamos con gráficos de coches
        // con valores aleatorios para su velocidad, dirección y rotación.
        Coches = new Vector<Grafico>();
        for (int i=0; i<numCoches; i++) {
            Grafico coche = new Grafico(this, graficoCoche);
            coche.setIncX(Math.random() * 4 - 2);
            coche.setIncY(Math.random() * 4 - 2);
            coche.setAngulo((int) (Math.random() * 360));
            coche.setRotacion((int) (Math.random() * 8 - 4));
            Coches.add(coche);
        }
    }

    //Al comenzar y dibujar por primera vez la pantalla del juego
    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        super.onSizeChanged(w, h, oldw, oldh);

        //Dibujamos los coches en posiciones aleatorias
        for (Grafico coche: Coches) {
            coche.setPosX(Math.random()*(w-coche.getAncho()));
            coche.setPosY(Math.random()*(h-coche.getAlto()));
        }
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        //Dibujamos cada uno de los coches
        for (Grafico coche: Coches) {
            coche.dibujaGrafico(canvas);
        }
    }

}
```

1.2.5.Añadir la actividad a AndroidManifest.xml

Como ya sabemos todas las actividades (pantallas) que hagamos en nuestras aplicaciones debemos añadirlas al fichero AndroidManifest.xml.

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
    <activity
        android:name=".Solobici"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".AcercaDe"
        android:label="Acerca de..."
        android:theme="@android:style/Theme.Dialog">
    </activity>
    <activity android:name=".Juego
        android:label="Juego..." >
    </activity>
</application>
```

1.2.6.Crear botón y escuchador para la pantalla Juego

El botón y el escuchador para poder activar la pantalla Juego debemos incluirlos en el archivo “Solobici.java”.

```
package juego.solobici;

import android.app.Activity;

public class Solobici extends Activity {

    private Button bAcercaDe;
    private Button bJuego;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //Botón y escuchador para la pantalla "Juego"
        //Al hacer click en este botón llamamos al método lanzarJuego()
        bJuego = (Button) findViewById(R.id.Boton01);
        bJuego.setOnClickListener(new OnClickListener(){
            public void onClick(View view) {
                lanzarJuego();
            }
        });


        //Botón y escuchador para la pantalla "Acerca de"
        //Al hacer click en este botón llamamos al método lanzarAcercaDe()
        bAcercaDe = (Button) findViewById(R.id.Boton03);
        bAcercaDe.setOnClickListener(new OnClickListener(){
            public void onClick(View view) {
                lanzarAcercaDe();
            }
        });
    }

    //Método que activa la pantalla Juego
    public void lanzarJuego(){
        Intent i = new Intent(this, Juego.class);
        startActivity(i);
    }

    //Método que activa la pantalla AcercaDe
    public void lanzarAcercaDe(){
        Intent i = new Intent(this, AcercaDe.class);
        startActivity(i);
    }
}
```

1.2.7.Dibujar la bici

Para dibujar la bici declararemos en primer lugar las siguientes variables/atributos al comienzo de la clase VistaJuego:

```
// BICI //
private Grafico bici;
private int giroBici;           //Incremento en la dirección de la bici
private float aceleracionBici;  //Aumento de velocidad en la bici
private static final int PASO_GIRO_BICI = 5;
private static final float PASO_ACELERACION_BICI = 0.5f;
```

1.2.7.1. Ejercicio 3

Modificar los tres métodos de la clase VistaJuego para que la bici sea correctamente inicializada, posicionada en el centro de la pantalla y dibujada.

Enviar una captura de pantalla de la pantalla del juego con la bici, los coches y el fondo.

1.2.8.Distribuir Coches por la pantalla

Puesto que los coches se distribuyen por la pantalla de una forma aleatoria, puede suceder que alguno de ellos coincida en posición con la nave. Para que esto no ocurra introduciremos un código que puede corregir esta situación:

```
//Al comenzar y dibujar por primera vez la pantalla del juego
@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    super.onSizeChanged(w, h, oldw, oldh);

    //Dibujamos los coches en posiciones aleatorias
    for (Grafico coche : coches) {
        do {
            coche.setPosX(Math.random()*(w-coche.getAncho()));
            coche.setPosY(Math.random()*(h-coche.getAlto()));
        } while (coche.distancia(bici) < (w+h)/5);
    }
```

1.2.9.Dotar de movimiento a los coches y a la bici

De momento haremos que los coches y la bici se muevan de forma autónoma. Más adelante realizaremos los cambios oportunos para controlar la bici tanto con los cursores como con la pantalla táctil.

Para esto declaramos las siguientes variables/atributos al principio de la clase VistaJuego:

```
// THREAD Y TIEMPO //
//Hilo encargado de procesar el tiempo
private HiloJuego hiloJuego;
//Tiempo que debe transcurrir para procesar cambios (ms)
private static int PERIODO_PROCESO = 50;
//Momento en el que se realizó el último proceso
private long ultimoProceso = 0;
```

Creamos el siguiente método en la clase VistaJuego.

```
protected synchronized void actualizaMovimiento() {
    long ahora = System.currentTimeMillis();
    // No hacemos nada si el período de proceso no se ha cumplido.
    if (ultimoProceso + PERIODO_PROCESO > ahora) {
        return;
    }
    // Para una ejecución en tiempo real calculamos retardo
    double retardo = (ahora - ultimoProceso) / PERIODO_PROCESO;
    // Actualizamos la posición de la bici
    bici.setAngulo((int) (bici.getAngulo() + giroBici * retardo));
    double nIncX = bici.getIncX() + aceleracionBici
            * Math.cos(Math.toRadians(bici.getAngulo())) * retardo;
    double nIncY = bici.getIncY() + aceleracionBici
            * Math.sin(Math.toRadians(bici.getAngulo())) * retardo;
    if (Grafico.distanciaE(0, 0, nIncX, nIncY) <= Grafico.getMaxVelocidad()) {
        bici.setIncX(nIncX);
        bici.setIncY(nIncY);
    }
    bici.incrementaPos();

    //Movemos los coches
    for (Grafico coche : Coches) {
        coche.incrementaPos();
    }
    ultimoProceso = ahora;
}
```

Introducimos la siguiente clase en VistaJuego:

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

Introducimos las siguientes líneas al final del método onSizeChanged() de VistaJuego:

```
//HILO QUE CONTROLA EL JUEGO
hiloJuego = new HiloJuego();
hiloJuego.start();
```

2. MENÚS EN ANDROID

Android permite asignar menús contextuales a las aplicaciones que se despliegan pulsando el botón *menú* del teléfono.

2.1. Menús y submenús básicos

La primera posibilidad que tenemos para crear un menú es mediante XML. Los ficheros XML de menú se deben colocar en la carpeta "res/menu" del proyecto y tendrán la siguiente estructura:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">

<item android:id="@+id/MnuOpc1" android:title="Opcion1"
      android:icon="@drawable/iconoTag"></item>
<item android:id="@+id/MnuOpc2" android:title="Opcion2"
      android:icon="@drawable/iconoFilter"></item>
<item android:id="@+id/MnuOpc3" android:title="Opcion3"
      android:icon="@drawable/iconoChart">
    <menu>
        <item android:id="@+id/SubMnuOpc1"
              android:title="Opcion 3.1" />
        <item android:id="@+id/SubMnuOpc2"
              android:title="Opcion 3.2" />
    </menu>
</item>

</menu>
```

Tenemos un elemento <menu> que contendrá una serie de elementos <ítem> que se corresponderán con las distintas opciones a mostrar en el menú. En un elemento <ítem> podemos incluir otro elemento <menu> de forma que actuará como submenú.

Una vez definido el menú en el fichero XML, tenemos que definir el evento onCreateOptionsMenu() de la actividad que queremos que lo muestre según la siguiente estructura:

```
public boolean onCreateOptionsMenu(Menu menu) {

    //Alternativa 1

    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_principal, menu);


    return true;
}
```

Para implementar cada una de las opciones se incluirá en el evento onOptionsItemSelected() de la actividad que mostrará el menú. En el siguiente ejemplo, lo que haremos será modificar el texto de una etiqueta (lblMensaje) colocada en la pantalla principal de la aplicación:

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.MnuOpc1:
        lblMensaje.setText("Opcion 1 pulsada!");
        return true;
    case R.id.MnuOpc2:
        lblMensaje.setText("Opcion 2 pulsada!");;
        return true;
    case R.id.MnuOpc3:
        lblMensaje.setText("Opcion 3 pulsada!");;
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}
```

Al ejecutar el proyecto que os paso “EjemploMenu” comprobaremos cómo al pulsar el botón de “menú” del teléfono aparece el menú que hemos definido y que al pulsar cada opción se muestra el mensaje de ejemplo.

En caso de querer iconos para mostrar en los menús podéis acudir a los siguientes enlaces donde encontraréis algunos gratuitos:

- http://www.androidicons.com/freebies.php
- http://www.glyfx.com/products/free_android2.html

2.2. Ejercicio 4 (Menús en Solobici)

Debemos introducir un menú en nuestra aplicación didáctica Solobici. El menú resultante debe ser parecido al siguiente:

Enviar una captura de pantalla de la pantalla del juego con la bici, los coches y el fondo.