Programación de Videojuegos

Android nos proporciona ciertos mecanismos para poder realizar animaciones y trabajar con gráficos 2d y 3d. Obviamente, es mucho mejor desarrollar videojuegos con otras herramientas más especializadas como Unity; no obstante, todo esto se puede utilizar de forma razonablemente sencilla en nuestras App para hacer pequeñas animaciones: logos, iconos que giran, etc.

Android nos ofrece:

- **Canvas**: Un canvas, en inglés lienzo, nos permite dibujar cualquier objeto (líneas, rectángulos, óvalos…) en una superficie (Surface) que se volcará después en la pantalla de nuestro dispositivo.

- **Animadores (Animators)**: Permiten animar prácticamente cualquier objeto con el uso de propiedades y estilos preprogramados.

- **Drawable Animation**: Permite visualizar recursos Drawable como, por ejemplo, mostrando una imagen tras otra como si fuera un rollo de película.

- **Los Dibujables (Drawables)**: Android además dispone de una librería 2D para dibujar formas e imágenes.

- **Open GL**: Es una librería de funciones de alto rendimiento para el tratamiento de gráficos en 3D. Es una librería estándar y multiplataforma que permite dibujar escenas tridimensionales utilizando geometría de puntos, líneas y triángulos.

El Canvas en Android

Cuando queremos generar gráficos de forma dinámica, una forma es utilizando un objeto "lienzo" o "canvas". El canvas tiene la particularidad de que se repinta regularmente, de forma que podemos aprovechar ese *repintado* para hacer una animación.

El objeto Canvas tiene asociado un **bitmap**, que es lo que *se pinta* en la pantalla. Para mostrar un Canvas necesitamos:

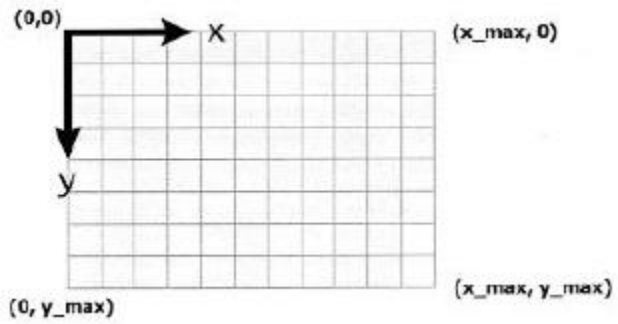
- Las coordenadas (X, Y) donde se va a *pintar* el objeto.

- Qué dibujar, por ejemplo, Rect (rectángulo), Text (texto), Bitmap (gráfico)…

- Un objeto Paint para describir los colores y estilo para el dibujo.

Para crear un Canvas se emplea el método **onDraw ().**

El sistema de **coordenadas** de Android está pensado para ser independientemente del tamaño de pantalla que se esté utilizando.



Por tanto, se hace necesario conocer los valores de **x_max** e **y_max**, que son dependientes de la pantalla del dispositivo. Para hacerlo, nos basamos en el objeto **Display**.

```
Display display = getWindowManager().getDefaultDisplay();
Point point = new Point();
display.getSize(point);
int maxX = point.x;
int maxY = point.y;
```

Canvas dispone de muchos métodos para **dibujar** elementos. A continuación, mostramos tan sólo unos pocos de todos los disponibles:

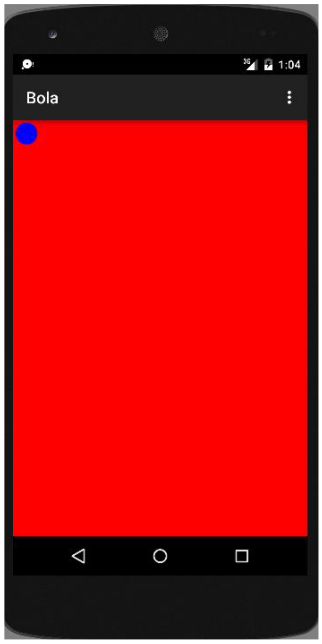
- drawCircle (): Dibuja un círculo de determinado radio.
- drawLine (): Dibuja una línea.
- drawText (): Dibuja un texto.
- …

Para dar colores y estilos utilizamos un **Paint**. Algunos de los métodos disponibles son:

- setColor(Color.BLACK): Para asignar el negro por defecto.
- setColor(Color.argb(alfa, rojo, verde, azul): Tu propio color.
- Usar el recurso colors.xml para crear tu propio color

  <color name="Azul">#500000FF</color>
- setTextAlign (): Para alinear texto.
- setTypeFace (): Tipo de Fuente.

Un ejemplo de Canvas: Creamos un objeto View de forma dinámica, donde dibujamos una esfera azul sobre un fondo rojo.




```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate( savedInstanceState );

        // En lugar de asignarle un Layout...
        // setContentView( R.layout.activity_main );

        // ... nos lo creamos dinámicamente...
        BolaView bolaView = new BolaView( context: this);
        setContentView (bolaView);
    }
}
```

```
public class BolaView extends View {

    public BolaView( Context context ) {
        super( context );
    }

    protected void onDraw (Canvas canvas ) {
        super.onDraw( canvas );

        Paint myPaint = new Paint();
        canvas.drawColor( Color.RED );
        myPaint.setColor( Color.BLUE );
        canvas.drawCircle( cx: 50, cy: 50, radius: 40, myPaint );
    }
}
```

Otro ejemplo: Creamos un Logo con texto para nuestra App.


```
protected void onDraw (Canvas canvas ) {
    super.onDraw( canvas );

    Path trazo = new Path();
    trazo.addCircle ( x: 400 , y: 400 , radius: 250 ,Path.Direction.CCW);
    canvas.drawColor (Color.WHITE);
    Paint pincel = new Paint ();
    pincel.setColor (Color.BLUE);
    pincel.setStrokeWidth(8);
    pincel.setStyle (Paint.Style.STROKE);
    canvas.drawPath (trazo, pincel);

    pincel.setStrokeWidth(1);
    pincel.setStyle (Paint.Style.FILL);
    pincel.setTextSize (40);
    pincel.setTypeface (Typeface.SANS_SERIF);
    canvas.drawTextOnPath ( text: "Desarrollo de aplicaciones para móviles con Android ",
            trazo, hOffset: 20 , vOffset: 50 , pincel);
    canvas.drawTextOnPath ( text: "Desarrollo de aplicaciones para móviles con Android por dentro",
            trazo, hOffset: 0, vOffset: -10 , pincel);
}
```

El Drawable

Literalmente, “Drawable” significa cualquier cosa que se puede dibujar. Usaremos un Drawable cuando queramos usar una imagen guardad en un fichero de nuestra App. Android es capaz de manejar ficheros del tipo .png, .jpg y .gif, los cuales deberán estar cargados en la carpeta de recursos **drawable**. Estos elementos pueden ser referenciados en recursos/drawable normalmente mediante **R.drawable.nombre_fichero**, o si se desean usar en un Layout mediante **@drawable/nombre_fichero**.

Para usar un Drawable, podríamos utilizar un **bitmap** y cargarlo en un canvas mediante la siguiente instrucción:

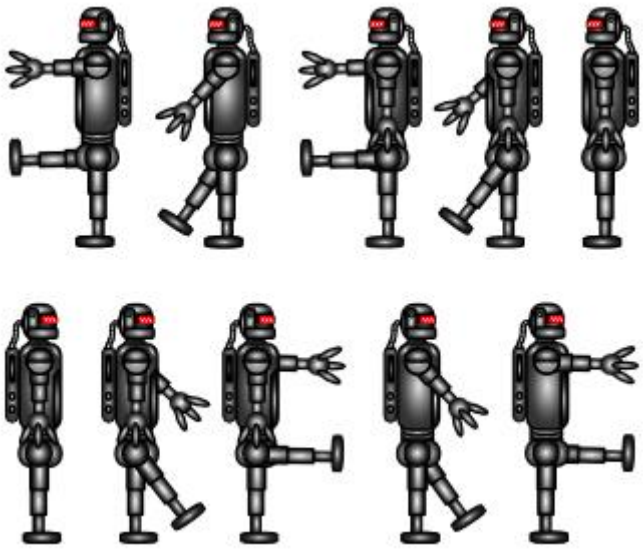
```
bitmap = BitmapFactory.decodeResource (getResources(), R.drawable.imagen);
canvas.drawBitmap (bitmap, 500, 500, null);
```

Sprites

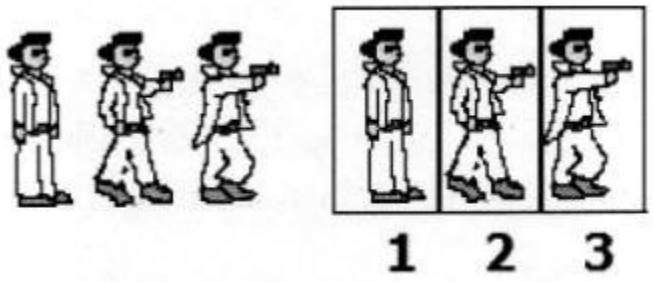
Los Sprites son imágenes con fondo transparente que se suelen utilizar para integrarlos con un fondo o escena. Para poder crear un Sprite podría ser necesario usar un programa de edición de imágenes tipo Gimp. En la siguiente dirección podríamos descargarnos muchos bitmaps y Sprites libres para nuestras App:

http://opengameart.org

Un ejemplo de uso de los Sprites podría ser una **animación**. Por ejemplo, utilizando como plantilla la siguiente imagen de robots en movimiento, podríamos ir intercalando diferentes imágenes para conseguir el efecto de movimiento.



Otra forma de hacer una animación sería tomar una única imagen con todas las posturas y cargarlas de una vez. Después, se puede ir mostrando a trozos mediante la instrucción:




```
canvas.drawBitmap (bitmap , source , destino , null );
```

Donde bitmap es la imagen con todas las posturas, source un objeto de tipo rectángulo con las coordenadas del recorte, y destino son las coordenadas donde se va a dibujar el bitmap.

Tipos de Drawables

Los diferentes tipos de Drawables son:

- BitmapDrawable: Son imágenes png o jpg que están en la carpeta drawable.
- GradienDrawable: Son degradados de color que se suelen usar de fondo.
- TransitionDrawable: Utilizados para pasar de una imagen a otra.
- ShapeDrawable: Permiten dibujar gráficos a partir de formas básicas.
- AnimationDrawable: Animación frame a frame a partir de una serie de otros objetos de tipo Drawable.

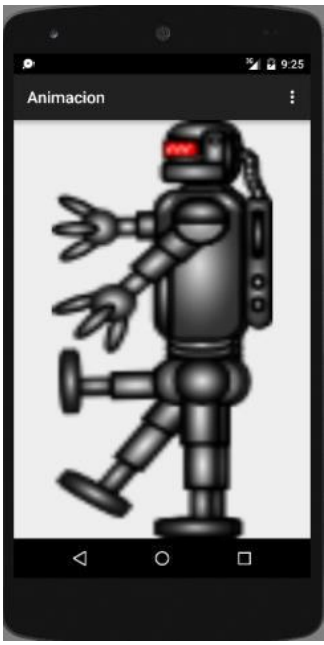
Un ejemplo de Drawable: GradienDrawable. Creamos el fichero **degrada.xml** en Drawable, con el siguiente contenido:

```
<shape xmlns:android =="http://schemas.android.com/apk/res/android">
<gradient
    android:startColor="#FFFFF"
    android:endColor="#000000"
    android:angle="270"/>
</shape>
```

Para mostrar el degradado podríamos:

- En el onCreate de un Activity: setBackgroundResource(R.drawable.degrada);
- En el layout: android:background="@drawable/degrada"

Otro ejemplo de Drawable: TransitionDrawable



res/drawable/transicion.xml

```
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable = "@drawable/robot1"/>
    <item android:drawable = "@drawable/robot2"/>
</transition>
```

Las imágenes **robot1** y **robot2** se cargan en **res/drawable**

Ahora, sólo hay que cargar la transición:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate( savedInstanceState );
    setContentView( R.layout.activity_main );

    ImageView imageView = new ImageView( context: this);
    setContentView (imageView);

    TransitionDrawable transitionDrawable = (TransitionDrawable) getDrawable (R.drawable.transition);
    imageView.setImageDrawable(transitionDrawable);
    transitionDrawable.startTransition( durationMillis: 2000);
}
```

Animations

Android incorpora un potente sistema de animaciones que permiten realizar cualquier tipo de animaciones automáticas sencillo. Para crear animaciones, utilizamos la clase **Animation**, que representa una animación que se puede mostrar sobre Vistas, Superficies u otros objetos. Además, se puede utilizar la clase AnimationSet para gestionar un conjunto de animaciones que serán visualizadas al mismo tiempo o unas detrás de otras.

Tenemos varios sistemas para crear animaciones:

- PropertyAnimations
- ViewAnimations
- DrawableAnimations

PropertyAnimations

Introducidas en el API nivel 11, nos permiten animar las propiedades de cualquier objeto. Por ejemplo, podemos cambiar el tamaño de un botón, rotarlo o incluso crear traslaciones por toda la pantalla. Un ejemplo de traslación de un botón que emerge lentamente (fade in) podría ser:

```
public void AnimacionBoton(){
      AnimatorSet animadorBoton = new AnimatorSet();

       //1ª animación, trasladar desde la izquierda
      //(800 pixeles menos hasta la posición inicial (0)
      ObjectAnimator trasladar=
          ObjectAnimator.ofFloat(botonJuego,"translationX",-800,0);
      trasladar.setDuration(5000); //duración 5 segundos

      //2ª Animación fade in de 8 segundos
      ObjectAnimator fade=
          ObjectAnimator.ofFloat(botonJuego, "alpha", 0f, 1f);
      fade.setDuration(8000);

       //se visualizan las dos animaciones a la vez
      animadorBoton.play(trasladar).with(fade);

       //comenzar animación
      animadorBoton.start();
}
```

ViewAnimations

Permiten dotar de animación a las Vistas. Las animaciones se calculan con información del punto de inicio, punto de fin, tamaño, rotación y otros parámetros. Se pueden crear dos tipos principales de efectos:

- **Animadores**: Realizan cambios en una vista para producir una animación.

RotateAnimation	Una animación que controla la rotación de un objeto
ScaleAnimation	Una animación que controla la escala de tamaño objeto
Transformation	Define la transformación que se aplicará a un objeto en algún punto de la animación
TranslateAnimation	Una animación que controla la posición del objeto
AlphaAnimation	Una animación que controla el nivel alfa (transparencia) de un objeto

- **Interpoladores:** Controlan cómo se producen los cambios en la animación.

AccelerateDecelerateInterpolator	Un interpolador donde el ratio de cambio comienza y acaba lentamente, pero acelera en el medio
AccelerateInterpolator	Un interpolador donde el ratio de cambio comienza lentamente y luego acelera
BounceInterpolator	Interpolador donde el cambio bota al final
CycleInterpolator	Interpolador que repite la animación un determinado número de veces
DecelerateInterpolator	Un interpolador que cambia rápidamente al principio y luego decelera
LinearInterpolator	Un interpolador donde el ratio de cambio es lineal

Estos efectos se pueden combinar a través de AnimationSet

Un ejemplo: mover una imagen de izquierda a derecha y de derecha a izquierda de la pantalla.

```
ImageView imagen = (ImageView) findViewById(R.id.imgPlaneta);

TranslateAnimation animation =
    new TranslateAnimation(-400.0f, 800.0f, 80.0f, 400.0f);
animation.setDuration(10000);  // duración de la animación
animation.setRepeatCount(5);  // repetir 5 veces
animation.setRepeatMode(2);   // repetir de izda a dcha y de dcha a izquierda

imagen.startAnimation(animation);  // comenzar animación
```

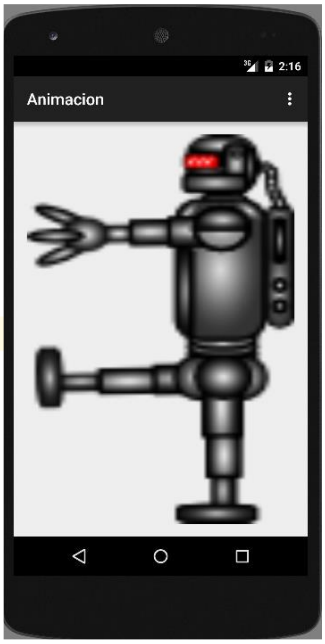
Drawable Animations

Con este tipo de animaciones se pueden encadenar varios bitmaps uno detrás de otro, de forma que se cree una animación fotograma a fotograma como si se tratase de una película. Para crear uno, bastaría con definir los bitmaps en un XML (en res/drawable) mediante la etiqueta **<animation-list>**. Cada uno de los elementos de la lista será una de las imágenes que se irán intercalando en la animación, que permanecerán visibles tantos milisegundos como indiquemos en la propiedad **android:duration**. Añadiendo el atributo **android:oneshot="false"** podemos hacer que la animación se repita continuamente.

Por ejemplo: para animar un robot en un ImageView:

Animación_robot.xml

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/robot1" android:duration="200" />
    <item android:drawable="@drawable/robot2" android:duration="200" />
    <item android:drawable="@drawable/robot3" android:duration="200" />
    <item android:drawable="@drawable/robot4" android:duration="200" />
    <item android:drawable="@drawable/robot5" android:duration="200" />
    <item android:drawable="@drawable/robot6" android:duration="200" />
    <item android:drawable="@drawable/robot7" android:duration="200" />
    <item android:drawable="@drawable/robot8" android:duration="200" />
    <item android:drawable="@drawable/robot9" android:duration="200" />
    <item android:drawable="@drawable/robot10" android:duration="200" />
</animation-list>
```



En el Activity:

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    super.onCreate (savedInstanceState);
    setContentView R.layout.activity_main;

    ImageView imageView = (ImageView ) findViewById (R.id.imgRobot);
    imageView.setBackgroundResource(R.drawable.animationDrawable);
    AnimationDrawable animationDrawable = (AnimationDrawable) imageView.getBackground();
}
```