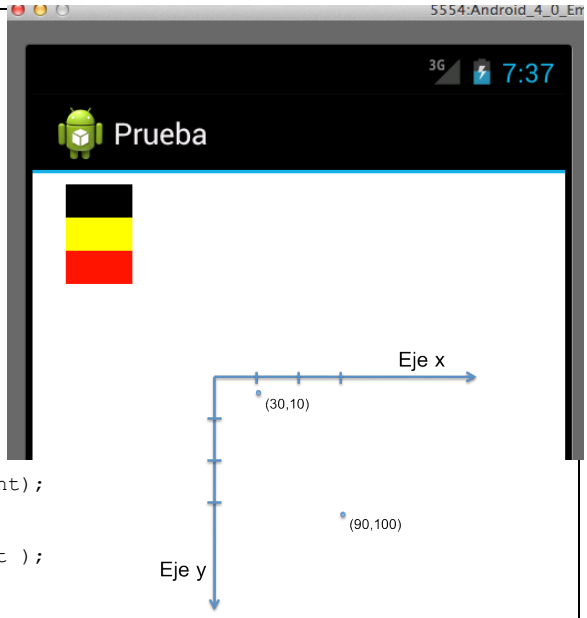
10. Gráficos en dos dimensiones

10.1 El método onDraw()

Los botones, como el resto de vistas, poseen su propia implementación del método `onDraw()`. Este método se ejecuta automáticamente cuando se representa la vista. Veamos un sencillo ejemplo que dibuja una bandera belga:

`/src/MyButton.java`

```
package com.programming.android.prueba;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.widget.Button;

public class MyButton extends Button {
    Paint paint = new Paint();

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

    public void onDraw(Canvas canvas) {

        paint.setColor(Color.BLACK);
        canvas.drawRect(30, 10, 90, 100, paint);

        paint.setColor(Color.YELLOW);
        canvas.drawRect(30, 40, 90, 70, paint );

        paint.setColor(Color.RED);
        canvas.drawRect(30, 70, 90, 100, paint );
    }
}
```



Comencemos el estudio de la clase `MyButton` por los colores.

Los colores en Android vienen representados como enteros en forma de 4 bytes: alfa, rojo, verde y azul (ARGB). Cada componente está comprendida entre 0 y 255. La primera mide la transparencia: 0 es totalmente transparente y 255 totalmente opaco. Para las demás, 0 significa que la componente no contribuye al color y 255 que contribuye al 100%. Por ejemplo, el azul opaco al 100% es `0xFF0000FF`, y el verde `0xFF00FF00`.

Para utilizar un color básico puedes recurrir a una de las constantes de la clase `Color`:

```
int color = Color.WHITE;
```

Sin embargo, es muy recomendable crear recursos de color en el archivo `colors.xml`, ya que pueden ser actualizados cómodamente:

/res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="background_color">#F0F0F0</color>
</resources>
```

Para acceder al recurso de color desde Java es necesario hacer lo siguiente:

```
int color = getResources().getColor(R.color.background_color);
```

El método `onDraw()` tiene un parámetro de tipo `Canvas`, que es el marco en el que se dibuja. Para dibujar es necesario instanciar un objeto de tipo `Paint` donde se especifica, entre otras cosas, el color de la pintura y el ancho de la brocha:

```
Paint paint = new Paint();
paint.setColor(Color.BLACK);
```

La clase `Canvas` dispone de métodos para dibujar líneas, rectángulos, círculos, etc. El prototipo de `drawRect()` es el siguiente:

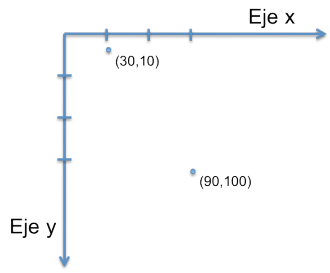
```
void drawRect (float left, float top, float right, float bottom, Paint paint);
```

donde `(left, top)` son las coordenadas del extremo superior izquierdo y `(right, bottom)` las del extremo inferior derecho del rectángulo.

Por ejemplo, la siguiente instrucción

```
canvas.drawRect(30, 10, 90, 100, paint);
```

dibuja un rectángulo cuyo vértice superior izquierdo está en (30, 10) y el inferior derecho en (90, 100):



El método `onDraw()` dibuja dos rectángulos más sin contornos: uno amarillo con vértices (30, 40) y (90, 70), y otro rojo con vértices (30, 70) y (90, 100):

```
paint.setColor(Color.YELLOW);
canvas.drawRect(30, 40, 90, 70, paint );
paint.setColor(Color.RED);
canvas.drawRect(30, 70, 90, 100, paint );
```

La pintura de estos rectángulos se superpone a la negra del rectángulo original, creando así una bandera como se puede ver en la figura del principio.

La clase `Main` se encarga de instanciar el objeto `but` de tipo `MyButton`, poner el fondo del botón de color blanco e inflar la vista:

/src/Main.java

```
package com.programming.android.prueba;

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

public class Main extends Activity {
    MyButton but;

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

        but = new MyButton(this);
        but.setBackgroundColor(Color.WHITE);
        setContentView(but);
    }
}
```

El lienzo (`canvas`) dispone de muchos otros métodos que se ilustran en el siguiente ejemplo:

- `setAntiAlias(true)` hace que el trazo sea más suave.
- `drawCircle(140,50,20,paint)` dibuja un círculo de radio 20 centrado en (x=140, y=50), y con el pincel `paint`.
- `setStyle(Style.STROKE)` fija el estilo del pincel como solo contorno . Otras posibilidades son `Style.FILL` (por defecto) y `Style.FILL_AND_STROKE`.
- `setStrokeCap(Cap.ROUND)` hace que el trazo se extienda en forma de semicírculo. Las otras posibilidades son `Cap.SQUARE` y `Cap.BUTT` (el trazo no se extiende de ninguna manera, es la opción por defecto).
- `drawLine(205,30,205,70,paint)` traza una recta desde (205,30) a (205, 70), es decir, una recta vertical con el pincel `paint`.
- `drawLines(pts,paint)` traza las líneas especificadas en el array `pts` en grupos de 4. Por ejemplo, la primera línea será la que va de (pts[0],pts[1]) a (pts[2],pts[3]); la segunda de (pts[4],pts[5]) a (pts[6],pts[7]), etc.
- `drawRoundRect(rf,10,10,paint)` dibuja un rectángulo redondeado a partir del objeto `RectF` pasado como primer argumento, que especifica las fronteras. El segundo y tercer argumentos especifican el radio en el eje x y en el eje y, respectivamente, de redondeo de las esquinas.
- `drawBitmap(bitmap,30,180,paint)` dibuja el objeto de tipo `Bitmap` pasado como primer argumento, colocando su esquina superior izquierda en el punto de coordenadas (x=30, y=180).

Los gradientes

Un gradiente de color permite sombrear las figuras entre dos colores. Existen varios tipos de gradientes: `LinearGradient`, `RadialGradient` y `SweepGradient`. En el ejemplo que sigue se utiliza uno radial:

```
RadialGradient grad = new RadialGradient (0, 0, 40,
                          Color.RED, Color.BLUE, Shader.TileMode.REPEAT);
```

Los cambios comienzan en un punto y se esparcen radialmente en círculo. El centro de este círculo de gradiente corresponde a los dos primeros argumentos: (x=0,y=0). El tercero es el radio del círculo para el gradiente de color. El cuarto y

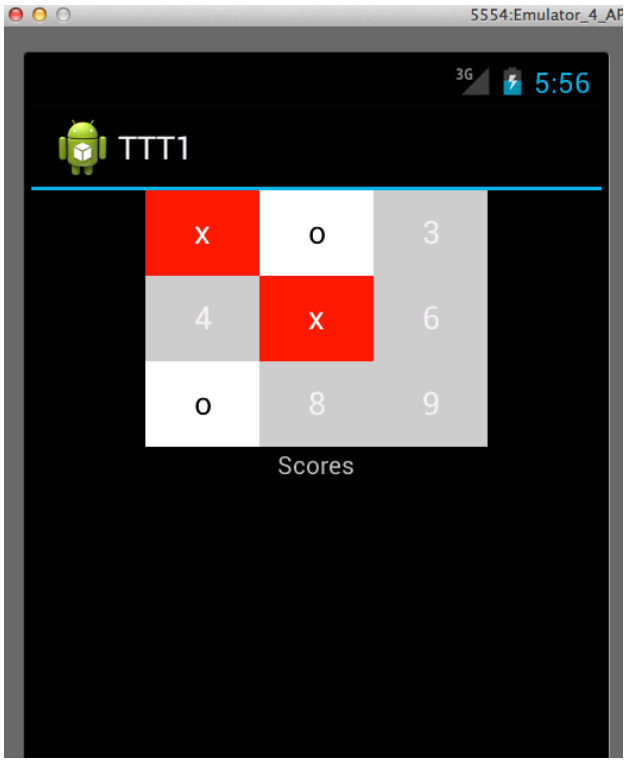
quinto argumentos son, respectivamente, el color del centro (`Color.RED`) y del extremo del círculo de gradiente (`Color.BLUE`). El último argumento es el modo de sombreado: las posibilidades son: `CLAMP`, `MIRROR` y `REPEAT`.

`/src/Main.java`


```
package com.example.drawing;

import android.os.Bundle;
...
import android.widget.Button;

class MyButton extends Button {
    Paint paint = new Paint();
    public MyButton(Context context) {
        super(context);
    }
    public void onDraw(Canvas canvas) {
        paint.setColor(Color.BLACK);
        paint.setStrokeWidth(3);
        canvas.drawCircle(40, 50, 20, paint);

        paint.setAntiAlias(true);
        canvas.drawCircle(90, 50, 20, paint);

        paint.setStyle(Style.STROKE);
        canvas.drawCircle(140, 50, 20, paint);

        canvas.drawPoint(190, 50, paint);

        paint.setStrokeCap(Cap.ROUND);
        canvas.drawLine(205, 30, 205, 70, paint);
        canvas.drawLine(205, 70, 255, 70, paint);

        float[] pts = new float[8];
        pts[0]=205; pts[1]=30; pts[2]=205; pts[3]=70;
        pts[0]=205; pts[1]=70; pts[2]=255; pts[3]=70;
        canvas.drawLines(pts, paint);

        Rect r = new Rect(20, 90, 80, 120);
        canvas.drawRect(r, paint);

        RectF rf = new RectF(90, 90, 150, 120);
        canvas.drawRoundRect(rf, 10, 10, paint);

        RadialGradient grad = new RadialGradient (0, 0, 40,
                              Color.RED, Color.BLUE, Shader.TileMode.REPEAT);
        paint.setShader(grad);
        paint.setStyle(Style.FILL);
        canvas.drawCircle(190, 120, 30, paint);

        Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.imagen);
        canvas.drawBitmap(bitmap, 30, 180, paint);
    }
}
public class Main extends Activity {
   protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        MyButton button = new MyButton(this);
        setContentView(button);
    }
}
```

10.2 La clase TTTButton

Nuestra clase `TTTButton` plantea el siguiente problema: al cambiar el color de fondo del botón remplazamos los dibujables (drawable) con que la clase `Button` viene por defecto. Por defecto, el botón se dibuja como un rectángulo centrado en la vista de fondo pero ligeramente más pequeño que la vista. En la unidad 9, al cambiar el color de fondo del botón, eliminamos ese ligero padding, y como resultado los botones del tablero aparecían pegados unos a otros como se puede ver en la siguiente figura.



El botón `TTTButton` de la unidad 9 utilizaba la implementación por defecto de `onDraw()`. En esta unidad, vamos a sobrecargar el método `onDraw()` como sigue:

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

    float width = getWidth();
    float height = getHeight();
    float padding = 10;

    Paint backgroundPaint = new Paint();
    backgroundPaint.setColor(color);
    canvas.drawRect(padding, padding, width-padding, height-padding, backgroundPaint);

    float x = 0.5f * width;
    float y = 0.5f * height;

    Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG);
    paint.setColor(textColor);
    paint.setStrokeWidth(2);
    paint.setTextAlign(Align.CENTER);
    paint.setTextSize(20);
    canvas.drawText(symbol, x, y, paint);
}
```

Al comienzo de `onDraw()` se guarda la anchura y la altura del botón en `width` y `height`, respectivamente:

```
float width = getWidth();
float height = getHeight();
```

A continuación, instanciamos un objeto de tipo `Paint` y le asignamos el color representado por el miembro `color` (echa un vistazo a la clase `TTTButton.java` completa más adelante):

```
Paint backgroundPaint = new Paint();
backgroundPaint.setColor(color);
```

Seguidamente, se dibuja un rectángulo con la brocha `backgroundPaint` en el marco `canvas`:

```
canvas.drawRect(padding, padding, width-padding, height-padding, backgroundPaint);
```

Este rectángulo tiene como vértice superior izquierdo (`10`, `10`) y como vértice inferior derecho (`width-10`, `height-10`).

Para escribir el símbolo del jugador actual dentro del rectángulo utilizamos el método `drawText()` de `Canvas`, que escribe la cadena pasada como primer argumento con la brocha pasada como último argumento:

```
canvas.drawText(symbol, x, y, paint);
```

donde `x` e `y` son las coordenadas del punto medio del rectángulo:

```
float x = 0.5f * width;
float y = 0.5f * height;
```

En la unidad anterior los símbolos se dibujaban mediante llamadas a `drawX()` y `drawO()`. En nuestra nueva versión, el método `onDraw()` se ejecutará automáticamente cuando el botón necesite dibujarse. Esto ocurrirá después de las llamadas a los nuevos `drawX()` y `drawO()`, que alteran el color del texto y del fondo, así como el símbolo dibujado.

Por ejemplo, `drawX()` queda de la siguiente manera:

```
public void drawX(){
    color = xBackgrColor;
    symbol = x;
    textColor = xColor;
    this.setClickable(false);
}
```

donde `xBackgrColor`, `x` y `xColor` son miembros estáticos constantes de `TTTButton`:

```
private String symbol = noSymbol;
private int color = voidColor;
private int textColor = voidColor;

private static String x = "x";
private static int xColor = Color.WHITE;
private static int xBackgrColor = Color.RED;
```

La clase `TTTButton` al completo queda como sigue:

src/TTTButton.java

```
package com.programming.android.ttt.ttt2;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Paint.Align;
import android.util.AttributeSet;
import android.widget.Button;

public class TTTButton extends Button {
      private String symbol = noSymbol;
      private int color = voidColor;
      private int textColor = voidColor;

      private static String noSymbol = "";
      private static String x = "x";
      private static String o = "o";

      private static int voidColor = Color.BLACK;
      private static int xColor = Color.WHITE;
      private static int oColor = Color.BLACK;

      private static int voidBackgrColor = Color.LTGRAY;
      private static int xBackgrColor = Color.RED;
      private static int oBackgrColor = Color.WHITE;

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

          float width = getWidth();
          float height = getHeight();
          float padding = 10;

          Paint backgroundPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
          backgroundPaint.setColor(color);
          canvas.drawRect(padding, padding, width-padding, height-padding,
                          backgroundPaint);

          float x = 0.5f * width;
          float y = 0.5f * height;

          Paint paint = new Paint();
          paint.setColor(textColor);
          paint.setStrokeWidth(2);
          paint.setTextAlign(Align.CENTER);
          paint.setTextSize(20);
          canvas.drawText(symbol, x, y, paint);
      }
      public TTTButton(Context context, AttributeSet attrs){
          super(context, attrs);
          reset();
      }
      public void drawX(){
          color = xBackgrColor;
          symbol = x;
          textColor = xColor;
          this.setClickable(false);
      }
      public void drawO(){
          color = oBackgrColor;
          symbol = o;
          textColor = oColor;
          this.setClickable(false);
      }
      public void reset(){
          color = voidBackgrColor;
          symbol = noSymbol;
          textColor = voidColor;
          this.setClickable(true);
      }
}
```

10.3 La clase Main.java

Esta clase no cambia con respecto a la de la unidad 9. La incluimos aquí por completitud:

src/Main.java

```
package com.programming.android.ttt.ttt2;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class Main extends Activity implements View.OnClickListener{
    private Game game;
    private TextView textView;
    private int ids [] = {R.id.button1, R.id.button2, R.id.button3,
                          R.id.button4, R.id.button5, R.id.button6,
                          R.id.button7, R.id.button8, R.id.button9};

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        registerListeners();
        textView = (TextView) findViewById(R.id.textView);
        game = new Game();
    }

    private int fromIdToPosition (int id){
        for (int i=0; i<9; i++)
            if (id == ids[i])
                return i+1;
        return -1;
    }

    public void onClick(View v) {
        TTTButton button = (TTTButton) v;

        if (!game.isGameActive())
            return;

        if (game.isCurrentSymbolX())
            button.drawX();
        else
            button.drawO();

        game.play(fromIdToPosition(button.getId()));

        if (game.isWon())
            textView.setText("Player " + game.getCurrentPlayer() + " is the winner");
        else if (game.isDrawn())
            textView.setText("Game drawn");

        game.switchCurrentSymbolAndPlayer();
    }

    private void registerListeners(){
        TTTButton button;
        for (int i=0; i<9; i++) {
                button = (TTTButton) findViewById(ids[i]);
                button.setText(Integer.toString(i+1));
                button.setOnClickListener(this);
        }
    }
}
```

El fichero de diseño utiliza ahora los nuevos botones, como es natural:

res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center_horizontal" >
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >
        <com.programming.android.ttt.ttt2.TTTButton
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="1"/>
        <com.programming.android.ttt.ttt2.TTTButton
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="2"/>
        <com.programming.android.ttt.ttt2.TTTButton
            android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="3"/>
    </LinearLayout>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <com.programming.android.ttt.ttt2.TTTButton
            android:id="@+id/button4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="4"/>
        ...
        <com.programming.android.ttt.ttt2.TTTButton
            android:id="@+id/button9"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="9"/>
    </LinearLayout>

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Scores" />

</LinearLayout>
```

Ejercicio 10.1

En este ejercicio puedes reutilizar las clases `Main.java` y `Game.java` del ejercicio 9.2. El único cambio tendrá lugar en la clase `TTTButton`, en la que vas a adaptar los métodos `on()` y `off()` para que ajusten el valor de `currentColor` a verde y gris, respectivamente, en lugar de invocar `setText()`. También tendrás que sobrescribir el método `onDraw()` para dibujar un círculo de radio 15 centrado en el centro del rectángulo del botón. El método `drawCircle()` de `Canvas` tiene el siguiente prototipo:

```
void drawCircle (float x, float y, float radio, Paint paint);
```

donde `(x,y)` es el centro del círculo, `radio` es el radio del círculo y `paint` es la brocha. El color del círculo será `currentColor`, de tal forma que aparezca verde cuando el botón está on y gris cuando está off. También puedes utilizar un bitmap diferente para cada estado del botón.

10.3 Animaciones

La plataforma Android proporciona cuatro tipos de animaciones:

- Imágenes GIF animadas. Los GIF animados son ficheros gráficos que contienen varios fotogramas (frames).
- Animaciones fotograma a fotograma. Mediante la clase `AnimationDrawable`, el programador suministra los fotogramas y las transiciones entre ellos.
- Animaciones de interpolación (tweening). Estas animaciones pueden resolverse con código XML y se aplican a cualquier vista.
- Animaciones con la biblioteca OPEN GL ES.

Animaciones de interpolación

Estos son los cuatro tipos de animaciones de interpolación:

- Animación `alpha` para cambiar la transparencia de una vista.
- Animación `rotate` para rotar una vista un cierto ángulo alrededor de un eje o punto de pivote.
- Animación `scale` para agrandar o disminuir una vista según el eje X e Y..
- Animación `translate` para desplazar una vista a lo largo del eje X e Y.

Las animaciones de interpolación se pueden definir tanto en XML como en Java. Por ejemplo, para crear una animación `alpha` en XML, añadiremos una carpeta de nombre `anim` a la carpeta res de nuestro proyecto. Dentro de esta carpeta, añadiremos un archivo de nombre `alpha.xml`. En el ejemplo siguiente, el atributo `android:fromAlpha` especifica el valor inicial de la trasparencia y

`android:toAlpha` el valor final. El atributo `android:duration` especifica la duración de la animación:

`res/anim/alpha.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromAlpha="1.0"
    android:toAlpha="0.0"
    android:duration="5000">
</alpha>
```

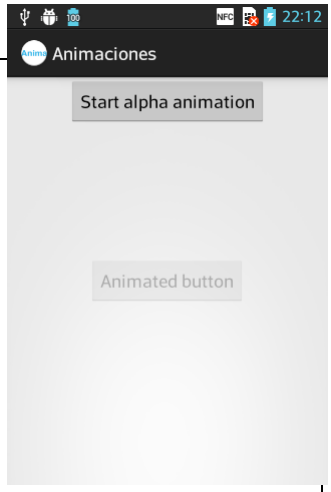
El fichero de diseño de este ejemplo especifica dos botones: uno que arranca la animación y otro en el centro del dispositivo que es el botón animado:

`res/layout/main.xml`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Main" >

    <Button
        android:id="@+id/button_start"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="Start alpha animation" />

    <Button
        android:id="@+id/button_animated"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="Animated button" />

</RelativeLayout>
```

El proceso que asocia una animación definida en un fichero XML a una vista concreta se denomina cargar (load) la animación. Para ello se utiliza la clase de ayuda `AnimationUtils`. El código que sigue carga la animación definida en `alpha.xml` y la asocia con el botón de identificador `button_animated`:

src/Main.java




```
package com.example.animaciones;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.Toast;

public class Main extends Activity {

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

      Button button_start = (Button) findViewById(R.id.button_start);
      final Button button_animated = (Button) findViewById(R.id.button_animated);

      button_start.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            // TODO Auto-generated method stub
            Animation animation = AnimationUtils.loadAnimation(Main.this, R.anim.alpha);
            button_animated.startAnimation(animation);
         }
      });
   }
}
```

Las animaciones se pueden combinar en XML mediante un elemento `<set>`. En el siguiente fichero se especifica una animación de escala seguida por una de rotación. La segunda empieza cuando acaba la primera pues el atributo `android:startOffSet` de la segunda animación se iguala a la duración de la primera (5000 milisegundos). La primera animación dobla el tamaño de la vista según el eje X y lo triplica según el eje Y. La segunda animación rota la vista alrededor de su punto medio (`android:pivotY="50%"`):

res/anim/alpha.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <scale
         android:fromXScale="1.0"
         android:toXScale="2.0"
         android:fromYScale="1.0"
         android:toYScale="3.0"
         android:pivotX="50%"
         android:pivotY="50%"
         android:duration="5000"/>
    <rotate
         android:fromDegrees="0"
         android:toDegrees="360"
         android:pivotX="50%"
         android:pivotY="50%"
         android:duration="4000"
         android:startOffset="5000"/>
</set>
```

El atributo `android:fillAfter` permite especificar si se quiere que la vista vuelva o no a su estado inicial. Si le asignamos el valor `true`, la vista no volverá a su estado inicial.

Ejercicio 10.2

Utiliza una animación para hacer más atractivo el juego.