



ANIMACIONES Y JUEGOS EN ANDROID

Roberto Bayón López

Índice de contenidos

1 – REALIZACIÓN DE ANIMACIONES	2
1.1 - DIBUJAR UNA IMAGEN USANDO SURFACEVIEW	2
1.2 – GAMELOOP Y ANIMACIÓN	4
1.3 – LOS SPRITES	9
1.4 – VELOCIDAD Y ANIMACIÓN DE LOS SPRITES	13
2 – DESTRIPIANDO EL JUEGO: “APLASTA EL MAL”	19
2-1 – DESCRIPCION DE LAS CLASES	19
2.2 – TRABAJANDO CON VARIOS SPRITES A LA VEZ.....	20
2.3 – EVENTOS TÁCTILES Y DETECCIÓN DE COLISIONES DE SPRITES	24
2.4 SPRITES TEMPORALES	29
2.5 – CONDICIONES DE VICTORIA Y DERROTA	34
2.6 EL MENÚ PRINCIPAL Y LAS INSTRUCCIONES	39
2.7 – IMPLANTACIÓN DE SONIDO EN EL JUEGO	43
2.8 – FONDOS EN EL JUEGO	52
2.9 – CUESTIÓN DE RENDIMIENTO	54
3 – EL JUEGO ESTÁ TERMINADO: ¿AHORA QUE?.....	55
3.1 – SITUACIÓN DEL MERCADO DE JUEGOS MÓVILES	55
3.2 – Google Play Store	56
3.3 – LAS CLAVES DEL ÉXITO.....	58
3.3.1 – UN ÉXITO INESPERADO, FLAPPY BIRD	58
4 – BIBLIOGRAFIA	60

1 – REALIZACIÓN DE ANIMACIONES

Existen varias maneras de realizar una animación en Android, a continuación voy a empezar paso por paso a explicar como se hace, para ello deberemos familiarizarnos con la clase SurfaceView, esta clase proporciona un modo más directo de renderizar los componentes en pantalla.

Se trata de una View que contiene una Surface que es un buffer crudo que es usado por el screen compositor, el cual se encarga de renderizar todo en Android.

También tenemos que saber sobre el Canvas, esta clase representa una serie de herramientas para dibujar. Dispone de una serie de métodos que nos permiten representar líneas, círculos, texto ...

1.1 - DIBUJAR UNA IMAGEN USANDO SURFACEVIEW

En la Activity principal de nuestro proyecto, en el área onCreate, veremos que ejecutamos una View (vista) a partir del layout asociado a nuestra Activity como vemos en la siguiente captura:

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

Para poder trabajar en un SurfaceView lo que tenemos que hacer es que en esa View en vez de llamar al layout se llame al SurfaceView, crearemos una clase que extienda de ella y la llamaremos en el setContentView:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(new GameView(this));
}
```

En requestWindowFeature(Window.FEATURE_NO_TITLE) lo que hacemos es decir que la View se ejecute a pantalla completa, es decir, sin la barra superior donde nos aparecería el nombre del proyecto.

En setContentView hemos llamado a la clase que extiende de SurfaceView, la hemos llamado GameView y la hemos pasado como parámetro el Context, es decir, el contexto de la Activity que ha llamado a la clase, lo cual nos permitirá acceder a los recursos de dicha Activity.

El código de la clase GameView en este momento sería el siguiente:

```
public class GameView extends SurfaceView {
```

```

private Bitmap bmp;
private SurfaceHolder holder;
public GameView(Context context) {
    super(context);
    holder = getHolder();
    holder.addCallback(new SurfaceHolder.Callback() {
        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
        }
        @Override
        public void surfaceCreated(SurfaceHolder holder) {
            Canvas c = holder.lockCanvas(null);
            onDraw(c);
            holder.unlockCanvasAndPost(c);
        }
        @Override
        public void surfaceChanged(SurfaceHolder holder, int format,
            int width, int height) {
        }
    });
    bmp = BitmapFactory.decodeResource(getResources(), R.drawable.icon);
}
@Override
protected void onDraw(Canvas canvas) {
    canvas.drawColor(Color.BLACK);
    canvas.drawBitmap(bmp, 10, 10, null);
}
}

```

Como podemos ver se ha creado un objeto Bitmap y se ha guardado el icono de proyecto android que tenemos por defecto, luego se ha sobrescrito el método onDraw, en dicho método hemos dicho que el color del “lienzo” en el que vamos a dibujar sea de color negro y que en dicho lienzo se dibuje el icono de android en las coordenadas x 10 y 10.

El método onDraw no se llama por si solo, necesitamos que el objeto Canvas sea pasado como parámetro. Se puede pensar en el Canvas como una pizarra donde podemos pintar lo que queramos.

Para tener todo listo para pintar tenemos que crear un callback listener al holder, el holder es el objeto que contiene al View. En el método surfaceCreated (que es llamado cuando el View se crea), conseguimos el Canvas del holder y después llamamos al método onDraw.

Para conseguir el Canvas, utilizamos el método lockCanvas que bloquea el Canvas para evitar que nadie más dibuje cuando otro está dibujando. Una vez que se termina de dibujar, se desbloquea el Canvas llamando al método unlockCanvasAndPost.

Hay que tener en cuenta que durante el tiempo que se tiene un recurso bloqueado, estamos perdiendo rendimiento y por ello es importante minimizar el tiempo de bloqueo.



1.2 – GAMELOOP Y ANIMACIÓN

El "game loop" es la repetición de dos actividades principales;

1. Actualización de la física; se actualizan los datos del juego, como por ejemplo la posición "x" e "y" para un caracter (posiciones de los sprites, puntuación ...)
2. Dibujo; el dibujo de la imagen que se ve en la pantalla. Cuando este método es llamado repetidamente produce la percepción de ser una película o una animación.

Vamos a ejecutar el "game loop" en un thread diferente. En un hilo, hacemos las actualizaciones y los dibujos y en el hilo principal, manejamos los eventos como hacemos en una aplicación normal. El código que tenemos a continuación nos muestra esta implementación:

```
import android.graphics.Canvas;
public class GameLoopThread extends Thread {
    private GameView view;
    private boolean running = false;
    public GameLoopThread(GameView view) {
        this.view = view;
    }
    public void setRunning(boolean run) {
        running = run;
    }
}
```

```

    }
    @Override
    public void run() {
        while (running) {
            Canvas c = null;
            try {
                c = view.getHolder().lockCanvas();
                synchronized (view.getHolder()) {
                    view.onDraw(c);
                }
            } finally {
                if (c != null) {
                    view.getHolder().unlockCanvasAndPost(c);
                }
            }
        }
    }
}

```

El campo "running" es un "flag" que hace que se pare el "game loop". Dentro de este loop, llamamos al método *onDraw*,. En este caso, por simplicidad, hacemos el update y las actividades de dibujo en el método *onDraw*. Utilizamos la sincronización para evitar que cualquier otro hilo nos produzca conflicto cuando estemos dibujando.

En la clase *GameView* ahora vamos a realizar cambios, vamos a añadir un atributo llamado *x* que se referirá a la posición en el eje de las *x* de la imagen, al ejecutar el hilo se va a ir llamando al método *onDraw*, dicho método cada vez que se llama va a incrementar la *x* en 1, por lo que la imagen se moverá por el eje de las *x* mientras no se haya llegado al margen derecho de la pantalla, la clase *GameView* quedará así:

```

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class GameView extends SurfaceView {
    private Bitmap bmp;
    private SurfaceHolder holder;
    private GameLoopThread gameLoopThread;
    private int x = 0;

    public GameView(Context context) {
        super(context);
        gameLoopThread = new GameLoopThread(this);
    }
}

```

```

holder = getHolder();
holder.addCallback(new SurfaceHolder.Callback() {

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        boolean retry = true;
        gameLoopThread.setRunning(false);
        while (retry) {
            try {
                gameLoopThread.join();
                retry = false;
            } catch (InterruptedException e) {
            }
        }
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        gameLoopThread.setRunning(true);
        gameLoopThread.start();
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format,
        int width, int height) {
    }

});
bmp = BitmapFactory.decodeResource(getResources(), R.drawable.icon);
}

@Override
protected void onDraw(Canvas canvas) {
    canvas.drawColor(Color.BLACK);
    if (x < getWidth() - bmp.getWidth()) {
        x++;
    }
    canvas.drawBitmap(bmp, x, 10, null);
}
}

```

Si ejecutamos la aplicación podremos observar que la velocidad de la animación no es constante. Esto es porque cuando el "game loop" obtiene más tiempo de la CPU, la animación es más rápida. Podemos arreglar esto definiendo cuantos FPS (frames per second; imágenes por segundo) queremos en la aplicación.

Limitamos el dibujo a 10 FPS, que es 100 ms (milisegundos). Utilizamos el método sleep para que el tiempo restante sea 100ms. Si el bucle tarda más de 100ms, lo

dormimos 10ms igualmente para evitar que la aplicación utilice demasiada CPU, GameLoopThread nos queda con los cambios así:

```
import android.graphics.Canvas;

public class GameLoopThread extends Thread {
    static final long FPS = 10;
    private GameView view;
    private boolean running = false;

    public GameLoopThread(GameView view) {
        this.view = view;
    }

    public void setRunning(boolean run) {
        running = run;
    }

    @Override
    public void run() {
        long ticksPS = 1000 / FPS;
        long startTime;
        long sleepTime;
        while (running) {
            Canvas c = null;
            startTime = System.currentTimeMillis();
            try {
                c =
view.getHolder().lockCanvas();
                synchronized (view.getHolder())
                {
                    view.onDraw(c);
                }
            } finally {
                if (c != null) {
                    view.getHolder().unlockCa
nvasAndPost(c);
                }
            }
            sleepTime = ticksPS -
(System.currentTimeMillis() - startTime);
            try {
                if (sleepTime > 0)
                    sleep(sleepTime);
                else
                    sleep(10);
            } catch (Exception e) {}
        }
    }
}
```


Seguimos con las modificaciones, en este caso cambiamos otra vez GameView para que cuando la imagen llegue a el margen derecho rebote y vaya hasta la izquierda y asi indefinidamente, para ello agregamos un atributo llamado xSeep y cambiamos el método onDraw, nos quedaría así:

```
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class GameView extends SurfaceView {
    private Bitmap bmp;
    private SurfaceHolder holder;
    private GameLoopThread gameLoopThread;
    private int x = 0;
    private int xSpeed = 1;

    public GameView(Context context) {
        super(context);
        gameLoopThread = new GameLoopThread(this);
        holder = getHolder();
        holder.addCallback(new SurfaceHolder.Callback() {

            @Override
            public void surfaceDestroyed(SurfaceHolder holder) {
                boolean retry = true;
                gameLoopThread.setRunning(false);
                while (retry) {
                    try {
                        gameLoopThread.join();
                        retry = false;
                    } catch (InterruptedException e) {
                    }
                }
            }

            @Override
            public void surfaceCreated(SurfaceHolder holder) {
                gameLoopThread.setRunning(true);
                gameLoopThread.start();
            }

            @Override
            public void surfaceChanged(SurfaceHolder holder, int format,
```

```

        int width, int height) {
    }
});
bmp = BitmapFactory.decodeResource(getResources(), R.drawable.icon);
}

@Override
protected void onDraw(Canvas canvas) {
    if (x == getWidth() - bmp.getWidth()) {
        xSpeed = -1;
    }
    if (x == 0) {
        xSpeed = 1;
    }
    x = x + xSpeed;
    canvas.drawColor(Color.BLACK);
    canvas.drawBitmap(bmp, x, 10, null);
}
}

```

1.3 - LOS SPRITES

Un sprite es una imagen parcialmente transparente que suele ponerse normalmente sobre un fondo, en el caso de un videojuego vamos a acabar teniendo varios en la pantalla y pueden representar entidades manejadas por el ordenador, el personaje protagonista e incluso efectos y entornos.

Guardaremos los archivos correspondientes a los sprites en la carpeta `/res/drawable-mdpi` ya que son imágenes de poco tamaño.

Cada sprite tendrá asociado una posición (x,y) y una velocidad (x,y), y como veremos mas tarde también sonido, a continuación podemos ver el aspecto que tiene un sprite:



Ahora vamos a crear una clase para nuestros sprites:

```

import android.graphics.Bitmap;
import android.graphics.Canvas;

public class Sprite {
    private int x = 0;

```

```

private int xSpeed = 5;
private GameView gameView;
private Bitmap bmp;

public Sprite(GameView gameView, Bitmap bmp) {
    this.gameView=gameView;
    this.bmp=bmp;
}

private void update() {
    if (x > gameView.getWidth() - bmp.getWidth() - xSpeed) {
        xSpeed = -5;
    }
    if (x + xSpeed < 0) {
        xSpeed = 5;
    }
    x = x + xSpeed;
}

public void onDraw(Canvas canvas) {
    update();
    canvas.drawBitmap(bmp, x, 10, null);
}
}

```

En el constructor le pasamos la SurfaceView sobre la que se va a dibujar y el Bitmap que es la imagen que vamos a usar, el método update nos permite actualizar la posición del sprite, podemos observar como la condición para controlar la colisión con el borde de la pantalla ha cambiado y se le ha añadido el parámetro xSpeed, esto es debido a que en este caso el movimiento se realiza de 5 píxeles en 5 píxeles y eso tiene que tenerse en cuenta.

La clase GameView también ha de alterarse para adaptarse al cambio:

```

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class GameView extends SurfaceView {
    private Bitmap bmp;
    private SurfaceHolder holder;
    private GameLoopThread gameLoopThread;
    private Sprite sprite;
}

```

```

public GameView(Context context) {
    super(context);
    gameLoopThread = new GameLoopThread(this);
    holder = getHolder();
    holder.addCallback(new SurfaceHolder.Callback() {

        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
            boolean retry = true;
            gameLoopThread.setRunning(false);
            while (retry) {
                try {
                    gameLoopThread.join();
                    retry = false;
                } catch (InterruptedException e) {
                }
            }
        }

        @Override
        public void surfaceCreated(SurfaceHolder holder) {
            gameLoopThread.setRunning(true);
            gameLoopThread.start();
        }

        @Override
        public void surfaceChanged(SurfaceHolder holder, int format,
            int width, int height) {
        }
    });
    bmp = BitmapFactory.decodeResource(getResources(), R.drawable.bad1);
    sprite = new Sprite(this, bmp);
}

@Override
protected void onDraw(Canvas canvas) {
    canvas.drawColor(Color.BLACK);
    sprite.onDraw(canvas);
}
}

```

Si ejecutamos la aplicación tendremos lo siguiente moviéndose por la pantalla:



No es exactamente esto lo que queremos, lo que queremos es ir dibujando las imágenes de dentro del sprite una por una para conseguir un efecto de animación, para empezar vamos a usar la segunda fila que tiene una animación de caminar hacia la izquierda, para ello modificamos la clase Sprite:

```
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Rect;

public class Sprite {
    private static final int BMP_ROWS = 4;
    private static final int BMP_COLUMNS = 3;
    private int x = 0;
    private int y = 0;
    private int xSpeed = 5;
    private GameView gameView;
    private Bitmap bmp;
    private int currentFrame = 0;
    private int width;
    private int height;

    public Sprite(GameView gameView, Bitmap bmp) {
        this.gameView = gameView;
        this.bmp = bmp;
        this.width = bmp.getWidth() / BMP_COLUMNS;
        this.height = bmp.getHeight() / BMP_ROWS;
    }

    private void update() {
        if (x > gameView.getWidth() - width - xSpeed) {
            xSpeed = -5;
        }
        if (x + xSpeed < 0) {
            xSpeed = 5;
        }
        x = x + xSpeed;
    }
}
```

```

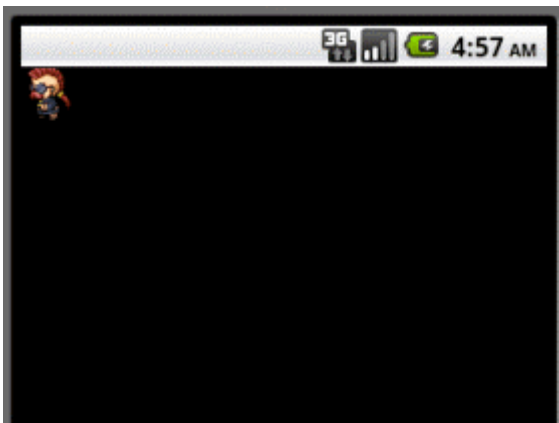
        currentFrame = ++currentFrame % BMP_COLUMNS;
    }

    public void onDraw(Canvas canvas) {
        update();
        int srcX = currentFrame * width;
        int srcY = 1 * height;
        Rect src = new Rect(srcX, srcY, srcX + width, srcY + height);
        Rect dst = new Rect(x, y, x + width, y + height);
        canvas.drawBitmap(bmp, src, dst, null);
    }
}

```

Partimos de la base de que el sprite que utilizamos tiene 3 columnas y 4 filas y lo reflejamos así en las constantes BMP_COLUMNS y BMP_ROWS, usando esas constantes es como si dividiéramos el sprite en cuadrículas, de esta manera calculamos en el constructor el alto (`this.height = bmp.getHeight() / BMP_ROWS;`) y ancho del dibujo (`this.width = bmp.getWidth() / BMP_COLUMNS;`) del personaje.

Para controlar el fotograma en el que nos encontramos usamos el atributo `currentFrame` y lo inicializamos a 0, cada vez que se ejecute el método `update` su valor cambiará y conseguiremos transición de imágenes, luego en el método `onDraw` calculamos el tamaño de un rectángulo, dentro de ese rectángulo es donde iremos dibujando los sprites, observamos que tenemos dos rectángulos, `src` y `dst`, el rectángulo `src`(origen) es el rectángulo de dentro de nuestro Bitmap, es decir, lo que vamos a dibujar, en este caso sería un rectángulo con el personaje dentro, el segundo rectángulo, `dst`(destino), es el rectángulo que dibujamos en el Canvas dentro del cual iría el rectángulo origen.



1.4 – VELOCIDAD Y ANIMACIÓN DE LOS SPRITES

Vamos a empezar dándole a nuestro sprite una velocidad `x` e `y` aleatoria, con eso conseguiremos que la dirección en la que se mueve sea aleatoria y además también vamos a hacer que cuando llegue al borde cambie de sentido:

```

import java.util.Random;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Rect;

public class Sprite {
    private static final int BMP_ROWS = 4;
    private static final int BMP_COLUMNS = 3;
    private int x = 0;
    private int y = 0;
    private int xSpeed = 5;
    private GameView gameView;
    private Bitmap bmp;
    private int currentFrame = 0;
    private int width;
    private int height;
    private int ySpeed;

    public Sprite(GameView gameView, Bitmap bmp) {
        this.gameView = gameView;
        this.bmp = bmp;
        this.width = bmp.getWidth() / BMP_COLUMNS;
        this.height = bmp.getHeight() / BMP_ROWS;
        Random rnd = new Random();
        xSpeed = rnd.nextInt(10)-5;
        ySpeed = rnd.nextInt(10)-5;
    }

    private void update() {
        if (x > gameView.getWidth() - width - xSpeed || x + xSpeed < 0) {
            xSpeed = -xSpeed;
        }
        x = x + xSpeed;
        if (y > gameView.getHeight() - height - ySpeed || y + ySpeed < 0) {
            ySpeed = -ySpeed;
        }
        y = y + ySpeed;
        currentFrame = ++currentFrame % BMP_COLUMNS;
    }

    public void onDraw(Canvas canvas) {
        update();
        int srcX = currentFrame * width;
        int srcY = 1 * height;
        Rect src = new Rect(srcX, srcY, srcX + width, srcY + height);
        Rect dst = new Rect(x, y, x + width, y + height);
        canvas.drawBitmap(bmp, src, dst, null);
    }
}

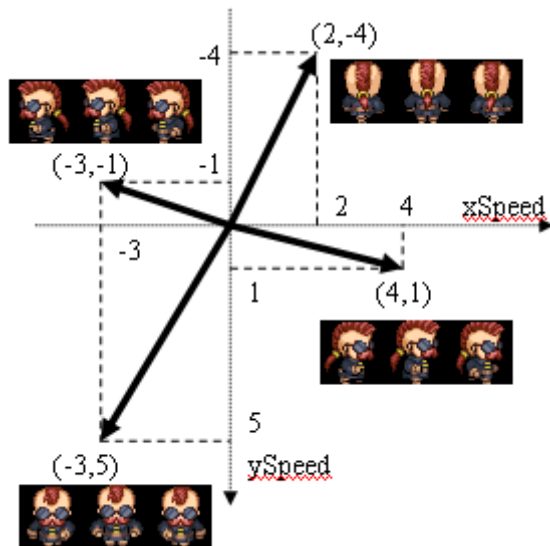
```

```
}  
}
```

La imagen de nuestro sprite contiene 4 animaciones con 3 imágenes cada una:



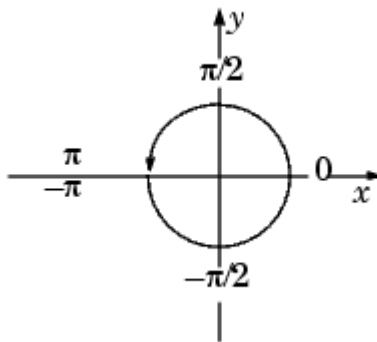
Dependiendo en que dirección vaya el sprite tenemos que enseñar una animación diferente, por ejemplo:



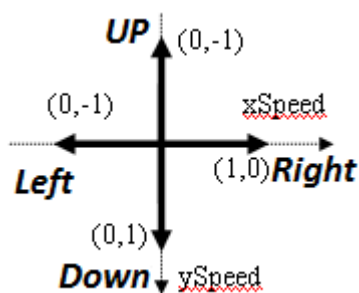
Necesitamos una función que tenga como parámetros "xSpeed" y "ySpeed" y que devuelva la fila que tenemos que utilizar para la animación [0,1,2,3]

Utilizaremos la función `Math.atan2(xSpeed, ySpeed)` para calcular la dirección del sprite en tiempo de ejecución.

`atan2(x,y)` nos da el ángulo en radianes en double desde (-PI a PI), pero necesitamos el resultado en int de 0 a 3 para saber que animación utilizar.



Observemos el gráfico que tenemos a continuación. En él escribimos las coordenadas (x,y) para cada dirección en la que tenemos una animación: Arriba (0,-1), derecha (1,0), abajo (0,1) e izquierda (0,-1):



Cuando el ángulo resultante está cerca de una de estas direcciones, queremos utilizar la animación correspondiente (up, down, left, right). Para esto utilizaremos la función `Math.round`.

Resolvemos esto con esta función y una constante array;

```
// direction = 0 up, 1 left, 2 down, 3 right,
// animation = 3 up, 1 left, 0 down, 2 right
int[] DIRECTION_TO_ANIMATION_MAP = { 3, 1, 0, 2 };

private int getAnimationRow() {
    double dirDouble = (Math.atan2(xSpeed, ySpeed) / (Math.PI / 2) + 2);
    int direction = (int) Math.round(dirDouble) % BMP_ROWS;
    return DIRECTION_TO_ANIMATION_MAP[direction];
}
```

En esta tabla muestro que he hecho en la función;

1. con `atan2(xSpeed,ySpeed)` obtengo el ángulo en radianes desde (-PI a PI)
2. Divido el ángulo por $\pi/2$ y obtengo un *double* de (-2 a 2)
3. Sumo 2 para cambiar el rango de (0 a 4)
4. Utilizo % (modulo) para reducir el rango de (0 a 3) (ver que 0 y el 4 son la misma dirección)
5. Mapeo cada dirección a la animación correcta utilizando el array { 3, 1, 0, 2 }

	x	y	atan2(x,y)	atan2(x,y)/(PI/2)	(atan2(x,y)/(PI/2)+2)%4	bmp row (from 0)
up	0	-1	PI or -PI	2 or -2	4 or 0	3
right	1	0	PI/2	1	3	2
down	0	1	0	0	2	0
left	-1	0	-PI/2	-1	1	1

Nota: aqui "x" y "y" son xSpeed y ySpeed.

A continuación lo que tenemos que hacer es implementar los cambios en la clase Sprite:

```
import java.util.Random;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Rect;

public class Sprite {
    // direction = 0 up, 1 left, 2 down, 3 right,
    // animation = 3 back, 1 left, 0 front, 2 right
    int[] DIRECTION_TO_ANIMATION_MAP = { 3, 1, 0, 2 };
    private static final int BMP_ROWS = 4;
    private static final int BMP_COLUMNS = 3;
    private int x = 0;
    private int y = 0;
    private int xSpeed = 5;
    private GameView gameView;
    private Bitmap bmp;
    private int currentFrame = 0;
    private int width;
    private int height;
    private int ySpeed;

    public Sprite(GameView gameView, Bitmap bmp) {
        this.width = bmp.getWidth() / BMP_COLUMNS;
        this.height = bmp.getHeight() / BMP_ROWS;
        this.gameView = gameView;
        this.bmp = bmp;

        Random rnd = new Random(System.currentTimeMillis());
        xSpeed = rnd.nextInt(50) - 5;
        ySpeed = rnd.nextInt(50) - 5;
    }

    private void update() {
        if (x >= gameView.getWidth() - width - xSpeed || x + xSpeed <= 0) {
```

```

        xSpeed = -xSpeed;
    }
    x = x + xSpeed;
    if (y >= gameView.getHeight() - height - ySpeed || y + ySpeed <= 0) {
        ySpeed = -ySpeed;
    }
    y = y + ySpeed;
    currentFrame = ++currentFrame % BMP_COLUMNS;
}

public void onDraw(Canvas canvas) {
    update();
    int srcX = currentFrame * width;
    int srcY = getAnimationRow() * height;
    Rect src = new Rect(srcX, srcY, srcX + width, srcY + height);
    Rect dst = new Rect(x, y, x + width, y + height);
    canvas.drawBitmap(bmp, src, dst, null);
}

// direction = 0 up, 1 left, 2 down, 3 right,
// animation = 3 back, 1 left, 0 front, 2 right
private int getAnimationRow() {
    double dirDouble = (Math.atan2(xSpeed, ySpeed) / (Math.PI / 2) + 2);
    int direction = (int) Math.round(dirDouble) % BMP_ROWS;
    return DIRECTION_TO_ANIMATION_MAP[direction];
}
}

```

2 – DESTRIPIANDO EL JUEGO: “APLASTA EL MAL”

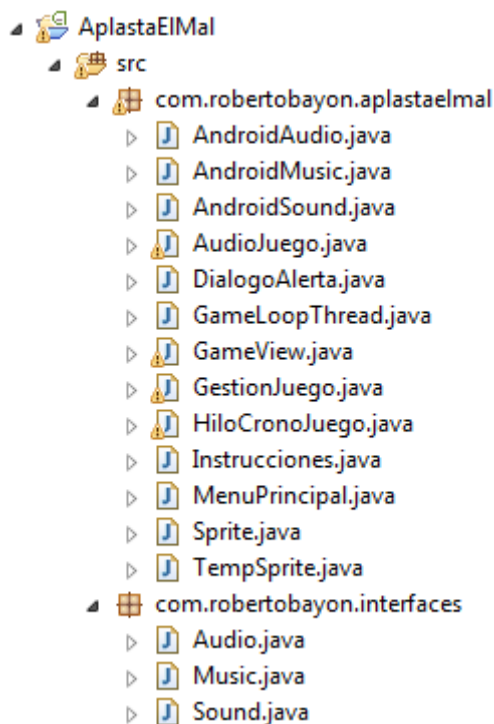
Ya hemos descubierto como realizar animaciones en android y hemos aprendido a usar sprites, ahora vamos a centrarnos en el juego protagonista de este proyecto: “Aplasta el mal”.

“Aplasta el mal” es un juego de habilidad en el cual personajes del bien y del mal se mueven por la pantalla, nuestro objetivo es acabar con los malos en el menor tiempo posible con ayuda de nuestro dedo.

En este capítulo vamos a desmenuzar el juego y a aprender como funcionan sus partes, para empezar veremos una descripción de todas las clases usadas.

2-1 – DESCRIPCION DE LAS CLASES

El proyecto consta de las siguientes clases:



1. AndroidAudio: controla el audio, sonido y música, del juego.
2. AndroidMusic: controla la música del juego.
3. AndroidSound: controla los sonidos del juego.
4. AudioJuego: clase encargada de unificar la gestión del audio del juego, música y efectos de sonido.
5. DialogoAlerta: gestiona los mensajes de alerta que vamos a usar para indicar si hemos ganado o perdido la partida.
6. GameLoopThread: hilo que controla el comportamiento de los personajes del juego.
7. GameView: es la SurfaceView, controla la superficie de dibujo y el lienzo donde dibujamos, es la encargada de representar el juego en pantalla.

8. GestionJuego: es la Activity encargada de la gestión del juego, desde aquí lanzamos el SurfaceView
9. HiloCronoJuego: es un hilo que se lanza cuando empieza el juego, cuenta los segundos y gracias a el podemos saber el tiempo que hemos tardado en completar el juego.
10. Instrucciones: Activity que nos muestra las instrucciones del juego.
11. MenuPrincipal: es la Activity principal del proyecto, se encarga de mostrar un menú con 3 opciones: Nueva partida, la cual lanzará la Activity GestionJuego, Instrucciones que nos llevará a la Activity Instrucciones y Salir que nos permitirá salir del juego.
12. Sprite: clase que controla las propiedades del sprite, posición, velocidad, imagen...
13. TempSprite: clase que controla los sprites temporales, vamos a hablar de ello mas tarde.
14. Audio: interface para implementar audio, la usamos en AndroidAudio
15. Music: interface para implementar música, la usamos en AndroidMusic
16. Sound: interface para implementar sonidos, la usamos en AndroidSound

2.2 – TRABAJANDO CON VARIOS SPRITES A LA VEZ

Nuestro juego no se va a limitar a mostrar un solo personaje pululando por nuestra pantalla, vamos a representar varios a la vez.

Esta vez para añadir sprites vamos a usar un método, lo vamos a llamar createSprites y a su vez llamará a otro método llamado createSprite al que pasamos la referencia del recurso y nos devolverá un sprite, irá añadiendo sprites a un ArrayList de sprites que habremos creado como atributo en la clase GameView, este metodo tendrá este aspecto:

```
private void createSprites() {
    sprites.add(createSprite(R.drawable.bad1));
    sprites.add(createSprite(R.drawable.bad2));
    sprites.add(createSprite(R.drawable.bad3));
    sprites.add(createSprite(R.drawable.bad4));
    sprites.add(createSprite(R.drawable.bad5));
    sprites.add(createSprite(R.drawable.bad6));
    sprites.add(createSprite(R.drawable.good1));
    sprites.add(createSprite(R.drawable.good2));
    sprites.add(createSprite(R.drawable.good3));
    sprites.add(createSprite(R.drawable.good4));
    sprites.add(createSprite(R.drawable.good5));
    sprites.add(createSprite(R.drawable.good6));
}

private Sprite createSprite(int resouce) {
    Bitmap bmp = BitmapFactory.decodeResource(getResources(), resouce);
    return new Sprite(this,bmp);
}
```

Tambien tendríamos que modificar el método onDraw:

```
for (Sprite sprite : sprites) {
    sprite.onDraw(canvas);
}
```

Ahora si ejecutamos el proyecto veriamos a todos los sprites en movimiento, pero hay algo que no queda muy bien, y es que todos los sprites salen del mismo punto, las coordenadas 0,0 que corresponden con la esquina superior izquierda de la pantalla.

Podemos solucionar este pequeño contratiempo haciendo que los sprites partan de posiciones aleatorias:

```
x = rnd.nextInt(gameView.getWidth() - width);
y = rnd.nextInt(gameView.getHeight() - height);
```

Esto lo hacemos en el constructor de la clase Sprite, después de hacer estos cambios la clase GameView quedaría así:

```
import java.util.ArrayList;
import java.util.List;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class GameView extends SurfaceView {
    private SurfaceHolder holder;
    private GameLoopThread gameLoopThread;
    private List<Sprite> sprites = new ArrayList<Sprite>();

    public GameView(Context context) {
        super(context);
        gameLoopThread = new GameLoopThread(this);
        holder = getHolder();
        holder.addCallback(new SurfaceHolder.Callback() {

            @Override
            public void surfaceDestroyed(SurfaceHolder holder) {
                boolean retry = true;
                gameLoopThread.setRunning(false);
                while (retry) {
                    try {
                        gameLoopThread.join();
                        retry = false;
                    } catch (InterruptedException e) {
```

```

        }
    }
}

@Override
public void surfaceCreated(SurfaceHolder holder) {
    createSprites();
    gameLoopThread.setRunning(true);
    gameLoopThread.start();
}

@Override
public void surfaceChanged(SurfaceHolder holder, int format,
    int width, int height) {
}

});
}

private void createSprites() {
    sprites.add(createSprite(R.drawable.bad1));
    sprites.add(createSprite(R.drawable.bad2));
    sprites.add(createSprite(R.drawable.bad3));
    sprites.add(createSprite(R.drawable.bad4));
    sprites.add(createSprite(R.drawable.bad5));
    sprites.add(createSprite(R.drawable.bad6));
    sprites.add(createSprite(R.drawable.good1));
    sprites.add(createSprite(R.drawable.good2));
    sprites.add(createSprite(R.drawable.good3));
    sprites.add(createSprite(R.drawable.good4));
    sprites.add(createSprite(R.drawable.good5));
    sprites.add(createSprite(R.drawable.good6));
}

private Sprite createSprite(int resouce) {
    Bitmap bmp = BitmapFactory.decodeResource(getResources(), resouce);
    return new Sprite(this, bmp);
}

@Override
protected void onDraw(Canvas canvas) {
    canvas.drawColor(Color.BLACK);
    for (Sprite sprite : sprites) {
        sprite.onDraw(canvas);
    }
}
}
}

```

La clase Sprite quedaría así:

```
import java.util.Random;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Rect;

public class Sprite {
    // direction = 0 up, 1 left, 2 down, 3 right,
    // animation = 3 back, 1 left, 0 front, 2 right
    int[] DIRECTION_TO_ANIMATION_MAP = { 3, 1, 0, 2 };
    private static final int BMP_ROWS = 4;
    private static final int BMP_COLUMNS = 3;
    private static final int MAX_SPEED = 5;
    private GameView gameView;
    private Bitmap bmp;
    private int x = 0;
    private int y = 0;
    private int xSpeed;
    private int ySpeed;
    private int currentFrame = 0;
    private int width;
    private int height;

    public Sprite(GameView gameView, Bitmap bmp) {
        this.width = bmp.getWidth() / BMP_COLUMNS;
        this.height = bmp.getHeight() / BMP_ROWS;
        this.gameView = gameView;
        this.bmp = bmp;

        Random rnd = new Random();
        x = rnd.nextInt(gameView.getWidth() - width);
        y = rnd.nextInt(gameView.getHeight() - height);
        xSpeed = rnd.nextInt(MAX_SPEED * 2) - MAX_SPEED;
        ySpeed = rnd.nextInt(MAX_SPEED * 2) - MAX_SPEED;
    }

    private void update() {
        if (x >= gameView.getWidth() - width - xSpeed || x + xSpeed <= 0) {
            xSpeed = -xSpeed;
        }
        x = x + xSpeed;
        if (y >= gameView.getHeight() - height - ySpeed || y + ySpeed <= 0) {
            ySpeed = -ySpeed;
        }
        y = y + ySpeed;
        currentFrame = ++currentFrame % BMP_COLUMNS;
    }
}
```



```

public void onDraw(Canvas canvas) {
    update();
    int srcX = currentFrame * width;
    int srcY = getAnimationRow() * height;
    Rect src = new Rect(srcX, srcY, srcX + width, srcY + height);
    Rect dst = new Rect(x, y, x + width, y + height);
    canvas.drawBitmap(bmp, src, dst, null);
}

private int getAnimationRow() {
    double dirDouble = (Math.atan2(xSpeed, ySpeed) / (Math.PI / 2) + 2);
    int direction = (int) Math.round(dirDouble) % BMP_ROWS;
    return DIRECTION_TO_ANIMATION_MAP[direction];
}
}

```

2.3 – EVENTOS TÁCTILES Y DETECCIÓN DE COLISIONES DE SPRITES

Nuestros sprites se desplazarán por la pantalla de forma aleatoria, pero todavía quedan cosas por hacer, tenemos que controlar las colisiones, tenemos que poder interactuar con los personajes y esto lo haremos a través de nuestro dedo.

Vamos a añadir un método a la clase GameView para controlar si en el sitio en el que hemos tocado con el dedo hay alguno de nuestros personajes, vamos a recorrer nuestro array de sprites consultando las coordenadas en las que se encuentran para saber si hay colision, en caso afirmativo retiramos el sprite. Iteramos la lista de sprites hacia atrás para evitar errores en la siguiente iteración, cuando hayamos eliminado el sprite:

```

@Override
public boolean onTouchEvent(MotionEvent event) {
    for (int i = sprites.size()-1; i >= 0; i--) {
        Sprite sprite = sprites.get(i);
        if (sprite.isCollition(event.getX(),event.getY())) {
            sprites.remove(sprite);
            break();
        }
    }
    return super.onTouchEvent(event);
}

```

El break es muy importante, si no lo tuviéramos todo seguiría funcionando pero cuando tocáramos en la pantalla todos los personajes que estuvieran en las coordenadas donde hemos puesto el dedo desaparecerían, un comportamiento mas correcto sería que solo desapareciera el primero, y eso es lo que conseguimos con el break.

La siguiente vez que el método `onDraw` dibuja la lista de sprites, los sprites eliminados de la lista, no serán dibujados. El efecto final es que el personaje desaparece debajo de nuestro dedo.

Para hacerlo funcionar, tenemos que implementar el método `isCollision` en la clase `Sprite`. Este método tiene que devolver `true` si las coordenadas (x,y) están dentro del área cubierta por el sprite:

```
public boolean isCollision(float x2, float y2) {  
    return x2 > x && x2 < x + width && y2 > y && y2 < y + height;  
}
```

Por último hacemos la sincronización. Los eventos son gestionados en un hilo diferente al del dibujo y la actualización. Esto puede producir un conflicto si actualizamos los datos que están siendo dibujados al mismo tiempo, también para evitar problemas vamos a hacer que el evento de cuando tocamos pantalla vaya mas lento ya que por defecto es muy rápido y puede hacer un efecto raro:

```
if (System.currentTimeMillis() - lastClick > 300) {  
    lastClick = System.currentTimeMillis();  
}
```

Con esto conseguimos que el tiempo minimo entre clicks sea de 300 milisegundos, con estos cambios la clase `GameView` queda así:

```
import java.util.ArrayList;  
import java.util.List;  
  
import android.content.Context;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.view.MotionEvent;  
import android.view.SurfaceHolder;  
import android.view.SurfaceHolder.Callback;  
import android.view.SurfaceView;  
  
public class GameView extends SurfaceView {  
    private Bitmap bmp;  
    private SurfaceHolder holder;  
    private GameLoopThread gameLoopThread;  
    private List<Sprite> sprites = new ArrayList<Sprite>();  
    private long lastClick;  
  
    public GameView(Context context) {  
        super(context);  
        gameLoopThread = new GameLoopThread(this);  
        holder = getHolder();  
    }  
}
```

```

holder.addCallback(new Callback() {

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        createSprites();
        gameLoopThread.setRunning(true);
        gameLoopThread.start();
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format,
        int width, int height) {
    }
});
}

private void createSprites() {
    sprites.add(createSprite(R.drawable.bad1));
    sprites.add(createSprite(R.drawable.bad2));
    sprites.add(createSprite(R.drawable.bad3));
    sprites.add(createSprite(R.drawable.bad4));
    sprites.add(createSprite(R.drawable.bad5));
    sprites.add(createSprite(R.drawable.bad6));
    sprites.add(createSprite(R.drawable.good1));
    sprites.add(createSprite(R.drawable.good2));
    sprites.add(createSprite(R.drawable.good3));
    sprites.add(createSprite(R.drawable.good4));
    sprites.add(createSprite(R.drawable.good5));
    sprites.add(createSprite(R.drawable.good6));
}

private Sprite createSprite(int resource) {
    Bitmap bmp = BitmapFactory.decodeResource(getResources(), resource);
    return new Sprite(this, bmp);
}

@Override
protected void onDraw(Canvas canvas) {
    canvas.drawColor(Color.BLACK);
    for (Sprite sprite : sprites) {
        sprite.onDraw(canvas);
    }
}

```

```

    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        if (System.currentTimeMillis() - lastClick > 500) {
            lastClick = System.currentTimeMillis();
            synchronized (getHolder()) {
                for (int i = sprites.size() - 1; i >= 0; i--) {
                    Sprite sprite = sprites.get(i);
                    if (sprite.isCollision(event.getX(), event.getY())) {
                        sprites.remove(sprite);
                        break;
                    }
                }
            }
        }
        return true;
    }
}

```

La clase Sprite queda así:

```

import java.util.Random;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Rect;

public class Sprite {
    // direction = 0 up, 1 left, 2 down, 3 right,
    // animation = 3 back, 1 left, 0 front, 2 right
    int[] DIRECTION_TO_ANIMATION_MAP = { 3, 1, 0, 2 };
    private static final int BMP_ROWS = 4;
    private static final int BMP_COLUMNS = 3;
    private static final int MAX_SPEED = 5;
    private GameView gameView;
    private Bitmap bmp;
    private int x = 0;
    private int y = 0;
    private int xSpeed;
    private int ySpeed;
    private int currentFrame = 0;
    private int width;
    private int height;

    public Sprite(GameView gameView, Bitmap bmp) {
        this.width = bmp.getWidth() / BMP_COLUMNS;
    }
}

```

```

    this.height = bmp.getHeight() / BMP_ROWS;
    this.gameView = gameView;
    this.bmp = bmp;

    Random rnd = new Random();
    x = rnd.nextInt(gameView.getWidth() - width);
    y = rnd.nextInt(gameView.getHeight() - height);
    xSpeed = rnd.nextInt(MAX_SPEED * 2) - MAX_SPEED;
    ySpeed = rnd.nextInt(MAX_SPEED * 2) - MAX_SPEED;
}

private void update() {
    if (x >= gameView.getWidth() - width - xSpeed || x + xSpeed <= 0) {
        xSpeed = -xSpeed;
    }
    x = x + xSpeed;
    if (y >= gameView.getHeight() - height - ySpeed || y + ySpeed <= 0) {
        ySpeed = -ySpeed;
    }
    y = y + ySpeed;
    currentFrame = ++currentFrame % BMP_COLUMNS;
}

public void onDraw(Canvas canvas) {
    update();
    int srcX = currentFrame * width;
    int srcY = getAnimationRow() * height;
    Rect src = new Rect(srcX, srcY, srcX + width, srcY + height);
    Rect dst = new Rect(x, y, x + width, y + height);
    canvas.drawBitmap(bmp, src, dst, null);
}

private int getAnimationRow() {
    double dirDouble = (Math.atan2(xSpeed, ySpeed) / (Math.PI / 2) + 2);
    int direction = (int) Math.round(dirDouble) % BMP_ROWS;
    return DIRECTION_TO_ANIMATION_MAP[direction];
}

public boolean isCollision(float x2, float y2) {
    return x2 > x && x2 < x + width && y2 > y && y2 < y + height;
}
}

```

Por ultimo la clase GameLoopThread queda así:

import android.graphics.Canvas;

```

public class GameLoopThread extends Thread {
    static final long FPS = 10;
    private GameView view;
    private boolean running = false;

    public GameLoopThread(GameView view) {
        this.view = view;
    }

    public void setRunning(boolean run) {
        running = run;
    }

    @Override
    public void run() {
        long ticksPS = 1000 / FPS;
        long startTime;
        long sleepTime;
        while (running) {
            Canvas c = null;
            startTime = System.currentTimeMillis();
            try {
                c = view.getHolder().lockCanvas();
                synchronized (view.getHolder()) {
                    view.onDraw(c);
                }
            } finally {
                if (c != null) {
                    view.getHolder().unlockCanvasAndPost(c);
                }
            }
            sleepTime = ticksPS - (System.currentTimeMillis() - startTime);
            try {
                if (sleepTime > 0)
                    sleep(sleepTime);
                else
                    sleep(10);
            } catch (Exception e) {}
        }
    }
}

```

2.4 SPRITES TEMPORALES

Los sprites temporales van a ser mucho mas fáciles de manejar que los sprites que hemos manejado hasta ahora, pero ¿Cuál es la diferencia?.

La diferencia radica en el uso que le vamos a dar, un sprite temporal nos va a servir para recrear diversos efectos en el juego, como por ejemplo un relámpago, una explosión...o en nuestro caso una mancha de sangre, y por supuesto van a desaparecer de la pantalla (por eso son temporales).

Para gestionarlos vamos a crear una clase nueva llamada TempSprite:

```
import java.util.List;
import android.graphics.Bitmap;
import android.graphics.Canvas;

public class TempSprite {
    private float x;
    private float y;
    private Bitmap bmp;
    private int life = 15;
    private List<TempSprite> temps;

    public TempSprite(List<TempSprite> temps, GameView gameView, float x,
        float y, Bitmap bmp) {
        this.x = Math.min(Math.max(x - bmp.getWidth() / 2, 0),
            gameView.getWidth() - bmp.getWidth());
        this.y = Math.min(Math.max(y - bmp.getHeight() / 2, 0),
            gameView.getHeight() - bmp.getHeight());
        this.bmp = bmp;
        this.temps = temps;
    }

    public void onDraw(Canvas canvas) {
        update();
        canvas.drawBitmap(bmp, x, y, null);
    }

    private void update() {
        if (--life < 1) {
            temps.remove(this);
        }
    }
}
```

Para hacer que el sprite desaparezca, introducimos un nuevo campo llamado *life*. Va a contener el número de *ticks* en los que el sprite va a estar vivo. Un *tick* es una marca para cada vez que se llama al método *update* dentro del *game loop*. Las imágenes por segundo (FPS) es una marca para cada llamada al método *onDraw* en un segundo. En nuestro juego FPS y los ticks de update están sincronizados pero no tiene porque ser así.

Cada vez que se llama al método `update`, disminuimos el valor de la vida en uno y cuando es menor que 1, eliminamos nuestro sprite de la lista de sprites temporales *temps*. La lista *temps* guarda todos los sprites temporales y es pasado como parámetro por el *view* en el constructor de *TempSprite*.

El constructor recibe además de los parámetros; una referencia al *view*, la posición (x,y) de donde hacemos click y el bitmap. La posición (x,y) se ajusta en el constructor teniendo en cuenta varios factores;

- El centro del bitmap de la sangre tiene que estar en el eje de coordenadas (x,y). Si no lo está, vamos a tener la sensación de que la sangre está abajo a la izquierda de donde hacemos click.

Conseguimos esto con: **`x - bmp.getWidth() / 2`**

- (x,y) no pueden salir de la pantalla. Si ocurre tendremos comportamientos inesperados y errores. Por lo tanto:
 1. x e y tienen que ser mayor que 0. **`Math.max(x - bmp.getWidth() / 2, 0)`**
 2. x tiene que ser menor que `gameView.getWidth()`, descontando el bitmap e y tiene que ser menor que `gameView.getHeight()` descontando la altura del bitmap. **`Math.min(... , gameView.getWidth() - bmp.getWidth());`**

```
this.x = Math.min(Math.max(x - bmp.getWidth() / 2, 0), gameView.getWidth() - bmp.getWidth());
```

```
this.y = Math.min(Math.max(y - bmp.getHeight() / 2, 0), gameView.getHeight() - bmp.getHeight());
```

El bitmap se imprime completo en la posición calculada de (x,y) en el método *onDraw*.

Todavía hacen falta unos cambios más;

1. Hay que copiar la siguiente imagen a la carpeta recursos:



2. Añadir la siguiente línea de código como última línea en el constructor de *view*.

```
bmpBlood = BitmapFactory.decodeResource(getResources(), R.drawable.blood1);
```

3. Incluir el campo *temps*

```
private List<TempSprite> temps = new ArrayList<TempSprite>();
```

4. Y añadir los dibujos de los sprites *temps* al método *onDraw*.


```

        for (int i = temps.size() - 1; i >= 0; i--) {
            temps.get(i).onDraw(canvas);
        }

```

Iteramos hacia atrás para evitar errores cuando los sprites sean eliminados. Escribimos este código antes de dibujar los otros sprites para dar la sensación de que la sangre está en un segundo plano.

5. Y por último añadimos

```

temps.add(new TempSprite(temps, this, x, y, bmpBlood));

```

justo después de que cada sprite sea eliminado en el método onTouchEventy.

Nuestra clase GameView queda de la siguiente forma:

```

import java.util.ArrayList;
import java.util.List;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class GameView extends SurfaceView {
    private GameLoopThread gameLoopThread;
    private List<Sprite> sprites = new ArrayList<Sprite>();
    private List<TempSprite> temps = new ArrayList<TempSprite>();
    private long lastClick;
    private Bitmap bmpBlood;

    public GameView(Context context) {
        super(context);
        gameLoopThread = new GameLoopThread(this);
        getHolder().addCallback(new SurfaceHolder.Callback() {

            @Override
            public void surfaceDestroyed(SurfaceHolder holder) {
                boolean retry = true;
                gameLoopThread.setRunning(false);
                while (retry) {
                    try {
                        gameLoopThread.join();
                        retry = false;
                    } catch (InterruptedException e) {}
                }
            }
        });
    }
}

```

```

    }
}

@Override
public void surfaceCreated(SurfaceHolder holder) {
    createSprites();
    gameLoopThread.setRunning(true);
    gameLoopThread.start();
}

@Override
public void surfaceChanged(SurfaceHolder holder, int format,
    int width, int height) {
}

});
bmpBlood = BitmapFactory.decodeResource(getResources(),
R.drawable.blood1);
}

private void createSprites() {
    sprites.add(createSprite(R.drawable.bad1));
    sprites.add(createSprite(R.drawable.bad2));
    sprites.add(createSprite(R.drawable.bad3));
    sprites.add(createSprite(R.drawable.bad4));
    sprites.add(createSprite(R.drawable.bad5));
    sprites.add(createSprite(R.drawable.bad6));
    sprites.add(createSprite(R.drawable.good1));
    sprites.add(createSprite(R.drawable.good2));
    sprites.add(createSprite(R.drawable.good3));
    sprites.add(createSprite(R.drawable.good4));
    sprites.add(createSprite(R.drawable.good5));
    sprites.add(createSprite(R.drawable.good6));
}

private Sprite createSprite(int resouce) {
    Bitmap bmp = BitmapFactory.decodeResource(getResources(), resouce);
    return new Sprite(this, bmp);
}

@Override
protected void onDraw(Canvas canvas) {
    canvas.drawColor(Color.BLACK);
    for (int i = temps.size() - 1; i >= 0; i--) {
        temps.get(i).onDraw(canvas);
    }
    for (Sprite sprite : sprites) {
        sprite.onDraw(canvas);
    }
}

```

```

    }
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    if (System.currentTimeMillis() - lastClick > 300) {
        lastClick = System.currentTimeMillis();
        float x = event.getX();
        float y = event.getY();
        synchronized (getHolder()) {
            for (int i = sprites.size() - 1; i >= 0; i--) {
                Sprite sprite = sprites.get(i);
                if (sprite.isCollision(x, y)) {
                    sprites.remove(sprite);
                    temps.add(new TempSprite(temps, this, x, y, bmpBlood));
                    break;
                }
            }
        }
        return true;
    }
}

```

2.5 – CONDICIONES DE VICTORIA Y DERROTA

Ya tenemos una serie de personajes correteando por nuestra pantalla, y no solo eso, si no que si tocamos sobre ellos desaparece y queda en su lugar una mancha de sangre, pero aun nos quedan mas cosas por hacer, por ejemplo establecer las condiciones de victoria y derrota.

Todos los juegos tienen determinadas unas condiciones en las cuales el jugador gana o el jugador pierde, en nuestro caso al tener personajes buenos y personajes malos y al llamarse nuestro juego “Aplasta el mal” obviamente nuestra condición de victoria será acabar con todos los malos y nuestra condición de derrota será acabar por accidente con una de las inocentes representantes del bien.

Lo primero en lo que deberíamos pensar es en como distinguimos los buenos de los malos, esto lo solucionamos poniendo un atributo boolean en la clase Sprite donde indicaremos si el personaje es bueno o malo, después en la zona donde se realiza el evento de tocar la pantalla veremos si el personaje que se encuentra bajo nuestro dedo es bueno o malo.

Debemos también contar cuantos malos hay y si al tocar con el dedo eliminamos a uno de ellos restaremos uno al total de malos, si el total de malos llega a 0 habremos

ganado y mostraremos un mensaje, sin embargo si el personaje asesinado es de los buenos supondrá nuestra descalificación fulminante.

Queda por tratar el tema del record, lo gestionamos con un SharedPreferences para almacenar y con un nuevo hilo cronometro para contar el tiempo, la primera vez que juguemos el record estará en 9999 segundos, cada vez que lo batamos se almacenará en el SharedPreferences.

Para implementar todo esto primero vamos a implementar una clase para gestionar los mensajes de alerta:

```
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;

public class DialogoAlerta extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        AlertDialog.Builder builder =
            new AlertDialog.Builder(getActivity());

        builder.setMessage("Esto es un mensaje de alerta.")
            .setTitle("Información")
            .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                }
            });

        return builder.create();
    }
}
```

En la clase GameView, concretamente en el método onTouchEvent, es donde vamos a controlar la victoria o la derrota, si ganamos el mensaje de alerta será así:

```
AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
dialog = builder.setTitle("Victoria").setMessage("!Lo has conseguido!").setPositiveButton("OK", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {

        Context con = getContext();
        ((Activity) con).finish();
    }
})
```

```

}).create();
dialog.show();
gameLoopThread.setRunning(false);

```

Cuando ganemos nos saldrá el mensaje y cuando pulsemos sobre OK nos saldremos de la Activity, conviene recordar que no estamos trabajando directamente sobre la Activity GestionJuego así que primero sacamos el Context y luego le aplicamos finish(), para evitar errores paramos los hilos con `gameLoopThread.setRunning(false)`;

Otro tanto de lo mismo con el mensaje de derrota:

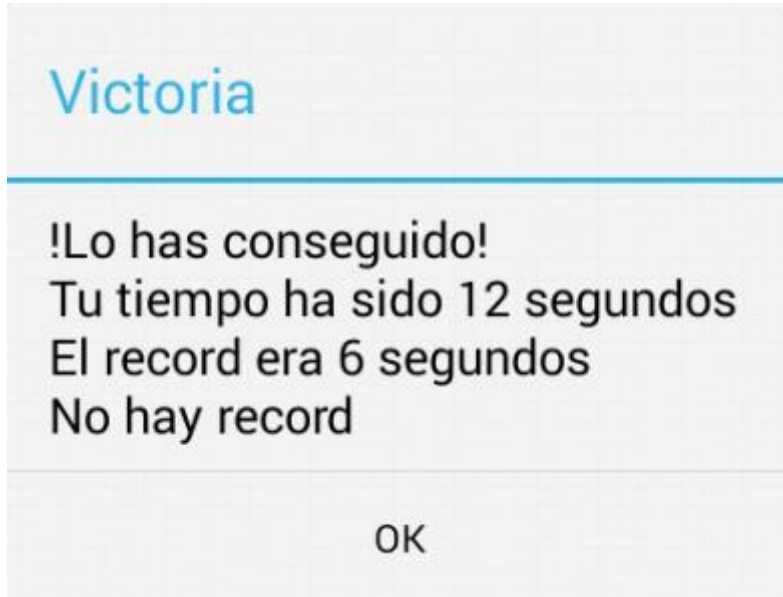
```

AlertDialog.Builder builder = new AlertDialog.Builder(getContext());
dialog = builder.setTitle("Game over").setMessage("!Has cometido un terrible
error!").setPositiveButton("OK", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {

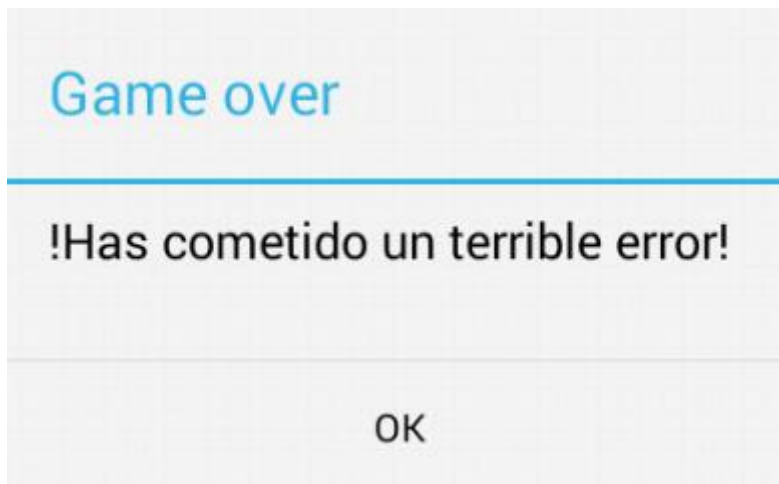
        Context con = getContext();
        ((Activity) con).finish();
    }
}).create();
dialog.show();
gameLoopThread.setRunning(false);

```

La alerta en caso de victoria seria así:



Notese que ya se han llevado a cabo los cambios que explicaremos mas abajo. A continuación veamos la alerta de derrota:



En la clase Sprite debemos añadir un nuevo atributo booleano para saber si el personaje es bueno o malo:

```
private boolean malo;
```

Con el siguiente cambio en el constructor:

```
public Sprite(GameView gameView, Bitmap bmp, boolean alineacion) {  
    this.width = bmp.getWidth() / BMP_COLUMNS;  
    this.height = bmp.getHeight() / BMP_ROWS;  
    this.gameView = gameView;  
    this.bmp = bmp;  
    this.malo = alineacion;  
}
```

Para terminar añadimos un get:

```
public boolean esMalo(){  
    return malo;  
}
```

Necesitamos un hilo cronometro para llevar la cuenta de los segundos de juego, quedaría así:

```
import static java.lang.Thread.sleep;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
class HiloCronoJuego extends Thread {  
  
    private boolean funcionar = true;  
    private int contadorSegs;  
  
    public HiloCronoJuego(){  
        int contadorSegs = 0;  
    }  
}
```

```

    }

    public void run() {

        while (funcionar) {
            contadorSegs++;
            try {
                sleep(1000);
            } catch (InterruptedException ex) {

            }
        }
    }

    public void parar(){
        funcionar = false;
    }

    public int getSegundos(){
        return contadorSegs;
    }
}

```

Poco tiene que explicar, es un hilo que tiene un bucle en el cual se va incrementando un contador cada 1000 milisegundos, es decir, que el valor del contador serán los segundos transcurridos, iniciamos el hilo cuando se crea el SurfaceView y lo paramos cuando ganemos o perdamos, con el método getSegundos obtendremos los segundos que hemos estado jugando.

Por ultimo necesitamos persistencia de datos para tratar los records, como hemos dicho esto lo hacemos con un SharedPreferences, en la Activity que gestiona el juego (GestionJuego) es donde vamos a hacerlo, lo primero va a ser poner un nuevo atributo que se llame record, y otro de tipo SharedPreferences que se llame records:

```

private static record;
private static SharedPreferences records;

```

En onCreate los inicializamos:

```

record = 9999;
records = getSharedPreferences("datos", Context.MODE_PRIVATE);

```

Luego implementamos dos métodos, el primero nos devuelve el record almacenado:

```

public static int leeRecord() {
    int recuRecord = records.getInt("record", record);
    return recuRecord;
}

```

El segundo graba un nuevo record en el caso que lo haya:

```
public static void grabaRecord(int tiempo) {  
    Editor editor = records.edit();  
    record = tiempo;  
    editor.putInt("record", record);  
    editor.commit();  
  
}
```

Estos métodos los llamamos desde el GameView, comparamos el resultado de HiloCronoJuego con el record que nos devuelve el método leeRecord, si hay nuevo record pasamos los segundos obtenidos por el hilo cronometro al método grabaRecord.

Los resultados se mostraran en las ventanas de alerta, por lo que realizaremos los siguientes cambios (en caso de victoria):

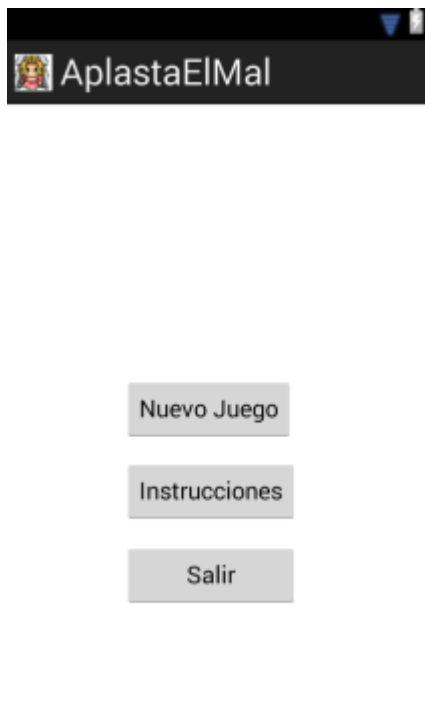
```
AlertDialog.Builder builder = new AlertDialog.Builder(getContext());  
dialog = builder.setTitle("Victoria").setMessage("!Lo has conseguido!\nTu tiempo ha  
sido " + crono.getSegundos() + " segundos\n" + "El record era " + recordAnterior + "  
segundos\n" + hayRecord).setPositiveButton("OK", new  
DialogInterface.OnClickListener() {  
    public void onClick(DialogInterface dialog, int id) {  
        GestionJuego.audioJuego.pararMusica();  
        Context con = getContext();  
        ((Activity) con).finish();  
    }  
}).create();  
dialog.show();  
gameLoopThread.setRunning(false);
```

Las variables usadas son: crono.getSegundos los segundos del crono, recordAnterior es el record que estaba almacenado en el SharedPreferences, hayRecord es una cadena que cambia dependiendo de si hay record, si hay record su valor es “!Se ha batido el record!”, si no “No se ha batido el record”.

2.6 EL MENÚ PRINCIPAL Y LAS INSTRUCCIONES

La Activity que gestiona el menú principal que nos da la bienvenida es bastante simple, en el layout pondremos 3 botones, uno para empezar un nuevo juego, otro para ver las instrucciones de juego y otro para poder salir del juego.

El aspecto del layout será algo así:



La Activity del menú principal tiene el siguiente código:

```
import java.io.IOException;
```

```
import android.os.Bundle;
```

```
import android.app.Activity;
```

```
import android.content.Intent;
```

```
import android.content.res.AssetFileDescriptor;
```

```
import android.content.res.AssetManager;
```

```
import android.view.Menu;
```

```
import android.view.View;
```

```
import android.view.Window;
```

```
public class MenuPrincipal extends Activity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        requestWindowFeature(Window.FEATURE_NO_TITLE);
```

```
        setContentView(R.layout.activity_menu_principal);
```

```
    }
```

```
    @Override
```

```
    public boolean onCreateOptionsMenu(Menu menu) {
```

```
        // Inflate the menu; this adds items to the action bar if it is present.
```

```
        getMenuInflater().inflate(R.menu.menu_principal, menu);
```

```
        return true;
```

```
    }
```

```

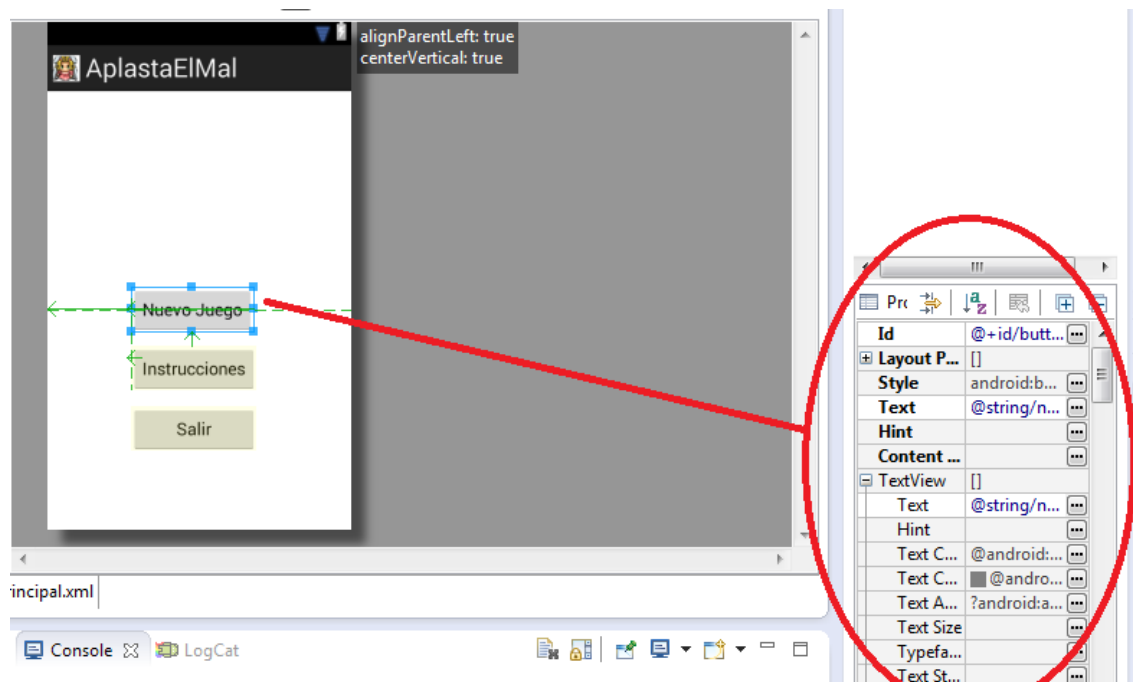
    public void jugar(View v) {
        Intent inten = new Intent(this, GestionJuego.class);
        startActivity(inten);
    }

    public void instrucciones(View v) {
        Intent inten = new Intent(this, Instrucciones.class);
        startActivity(inten);
    }

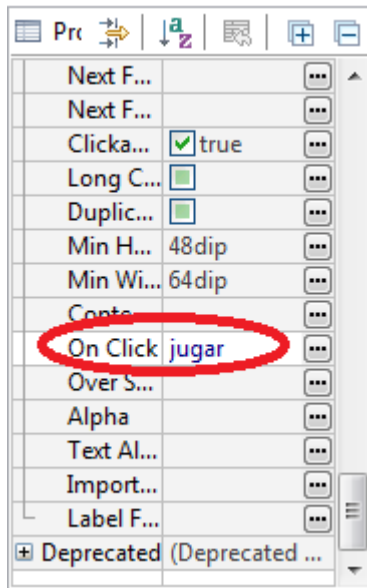
    public void salir(View v) {
        finish();
    }
}

```

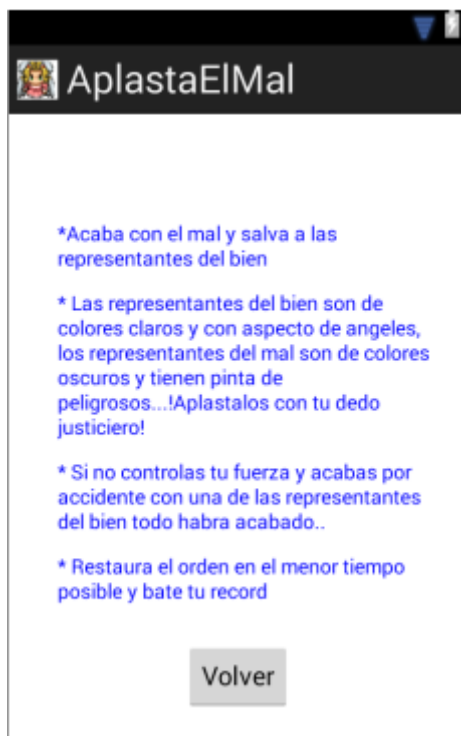
Asociamos los métodos a los botones, mostraré como hacerlo con el botón jugar ya que el resto se hace igual, primero seleccionamos el botón y nos fijamos en la zona de propiedades:



Bajamos hasta abajo del todo y buscamos el apartado On Click, allí escribimos el nombre del método que queremos asociar al botón:



La Activity Instrucciones solo mostrará texto con las instrucciones del juego y un botón para volver, el aspecto del layout es algo así:



El código de la clase Instrucciones es:

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.Window;

public class Instrucciones extends Activity {
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.activity_instrucciones);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.instrucciones, menu);
    return true;
}

public void volver(View v){
    finish();
}
}

```

Asociamos el método volver al botón de igual forma que vimos antes.

2.7 – IMPLANTACIÓN DE SONIDO EN EL JUEGO

El sonido en un juego es el broche de oro, va a hacer que las cosas sean mas animadas, además si nuestro juego tiene éxito sus melodías y sonidos van a ser recordados siempre.

Para ayudarnos a implementar el sonido al juego vamos a crear 3 interfaces que mas tarde vamos a implementar, empezamos con la interface de música, tiene todos los mecanismos para ayudarnos a manejar la música del juego, es decir, los controles típicos de un reproductor de música: play, stop y pause. Después el método setLooping lo usaremos para poner la música en modo bucle/loop. Con el método setVolume indicaremos el volumen de salida de nuestra música. Continuamos creando tres métodos (isPlaying, isStopped, isLooping) que usaremos para comprobar si la música se esta reproduciendo, esta parada o esta en modo bucle/loop. Y para terminar usaremos el método dispose para liberar los recursos de nuestro reproductor una vez que ya no lo necesitemos:

```

public interface Music {

    public void play();

    public void stop();
}

```

```

public void pause();

public void setLooping(boolean looping);

public void setVolume(float volume);

public boolean isPlaying();

public boolean isStopped();

public boolean isLooping();

public void dispose();
}

```

La segunda interface nos ayudará a gestionar el sonido del juego, simplemente necesitaremos dos métodos para los efectos de sonido, el método play para reproducirlo y el método dispose que liberara el efecto de sonido de la memoria RAM:

```

public interface Sound {

    public void play(float volume);

    public void dispose();

}

```

Con la ultima interface que vamos a usar conseguiremos reducir en numero de instancias de nuestro código y simplemente la usaremos para crear nuevos objetos de Sonido o Audio, pasándole como parámetro el nombre del archivo:

```

public interface Audio {

    public Music newMusic(String filename);

    public Sound newSound(String filename);

}

```

Ahora hablaremos de las clases que vamos a usar, empezamos con la mas compleja, la clase AndroidMusic, en esta clase implementamos la interface Music, pero también implementamos la interface OnCompletionListener que nos pide sobrescribir el método onCompletion. Esta interface se encarga de hacer una llamada a su método una vez que la reproducción a terminado, es decir, cuando termina de sonar la música porque a llegado a su fin.

El bloque synchronized se encargara automáticamente de gestionar los estados del reproductor (reproduciendo, parado, pausado, preparado). Con ello conseguiremos no

tener dos estados diferentes a la vez y por lo tanto no tener excepciones `IllegalStateException`.

Empezamos la interface creando un objeto `MediaPlayer` que sera el encargado de preparar, reproducir, pausar y parar nuestra música. También creamos un booleano para almacenar el estado preparado del reproductor de música:

```
public AndroidMusic(AssetFileDescriptor assetDescriptor) {  
  
    mediaPlayer = new MediaPlayer();  
  
    try {  
  
        mediaPlayer.setDataSource(assetDescriptor.getFileDescriptor(),  
  
            assetDescriptor.getStartOffset(),  
  
            assetDescriptor.getLength());  
  
        mediaPlayer.prepare();  
  
        isPrepared = true;  
  
        mediaPlayer.setOnCompletionListener(this);  
  
    } catch (Exception e) {  
  
        throw new RuntimeException("Error al cargar la musica");  
  
    }  
  
}
```

Ya en el constructor primero iniciamos nuestro `mediaplayer` y encapsulamos todo en un bloque `try-catch` por si acaso hay problemas al cargar el archivo de música.

Establecemos nuestra fuente de música con el metodo `setDataSource`, que nos pide como parámetro un `FileDescriptor` (usamos el parámetro `AssetFileDescriptor` con su método `getFileDescriptor` que nos devuelve un archivo de la carpeta `assets` para poder leer sus datos, así como el desplazamiento y su longitud), el punto inicial de nuestro archivo de música (con el método `getStartOffset` establecemos el inicio) y el final del archivo de música (con el método `getLength` establecemos el final).

Preparamos el archivo para su reproducción con el método `prepare` y guardamos el estado `true` en nuestra variable booleana.

Para terminar establecemos la llamada al método `onCompletion` y creamos una nueva excepción con un mensaje personalizado.

```
public void dispose() {  
  
    if (mediaPlayer.isPlaying())  
  
        mediaPlayer.stop();  
  
        mediaPlayer.release();  
  
    }
```

Comprobamos si se esta reproduciendo y en este caso pararemos la reproducción con el método stop y liberaremos el recurso de nuestro mediaplayer con el método release.

```
public boolean isLooping() {  
  
    return mediaPlayer.isLooping();  
  
    }
```

Devuelve true en caso de que el reproductor este en modo bucle/loop

```
public boolean isPlaying() {  
  
    return mediaPlayer.isPlaying();  
  
    }
```

Devuelve true en caso de que el reproductor este reproduciendo la música.

```
public boolean isStopped() {  
  
    return !isPrepared;  
  
    }
```

Comprobaremos nuestra variable booleana y nos devolverá false en caso de que se este reproduciendo.

```
public void pause() {  
  
    if (mediaPlayer.isPlaying())  
  
        mediaPlayer.pause();  
  
    }
```

Comprobaremos si se esta reproduciendo y en este caso pausamos la reproducción.

```

public void play() {

    if (mediaPlayer.isPlaying())

        return;

    try {

        synchronized (this) {

            if (!isPrepared)

                mediaPlayer.prepare();

            mediaPlayer.start();

        }

    } catch (IllegalStateException e) {

        e.printStackTrace();

    } catch (IOException e) {

        e.printStackTrace();

    }

}

```

Lo primero que hacemos es comprobar si se esta reproduciendo y en este caso devolvemos la función y ya no hace nada mas. En el caso contrario encapsulamos todo en un bloque try-catch manejando dos excepciones. Una vez dentro del bloque, sincronizamos el estado del reproductor y comprobamos si esta preparado a través de nuestra variable booleana, en caso de que no lo este lo preparamos con el método prepare y para terminar empezamos la reproducción con el método start.

```

public void setLooping(boolean isLooping) {

    mediaPlayer.setLooping(isLooping);

}

```

Ponemos el reproductor en modo bucle/loop a través del método mediaPlayer setLooping, indicándolo con el parámetro booleano de nuestro método isLooping.

```

public void setVolume(float volume) {

```



```
        mediaPlayer.setVolume(volume, volume);  
    }  
}
```

Indicamos el volumen derecho e izquierdo a través del método mediaPlayer setVolume y usando nuestro parámetro float volume.

```
public void stop() {  
    mediaPlayer.stop();  
    synchronized (this) {  
        isPrepared = false;  
    }  
}
```

Paramos la reproducción con el método mediaPlayer stop y seguidamente sincronizamos el estado del reproductor y ponemos nuestra variable a false indicando que no esta preparada la reproducción.

```
public void onCompletion(MediaPlayer player) {  
    synchronized (this) {  
        isPrepared = false;  
    }  
}
```

Para terminar sobreescribimos el método onCompletion y simplemente sincronizamos el estado y ponemos nuestra variable a false.

Ahora vamos a hablar de la clase encargada del audio, AndroidAudio:

```
import java.io.IOException;  
  
import android.app.Activity;  
  
import android.content.res.AssetFileDescriptor;
```

```

import android.content.res.AssetManager;

import android.media.AudioManager;

import android.media.SoundPool;


import com.example.slang.interfaces.Audio;

import com.example.slang.interfaces.Music;

import com.example.slang.interfaces.Sound;


public class AndroidAudio implements Audio {

```

```

    AssetManager assets;

```

```

    SoundPool soundPool;

```

Comentar que a parte de implementar nuestra interface Audio, estamos importando las interfaces de Sonido y Música, empezamos creando un objeto AssetManager y SoundPool.

```

public AndroidAudio(Activity activity) {

    activity.setVolumeControlStream(AudioManager.STREAM_MUSIC);

    this.assets = activity.getAssets();

    this.soundPool = new SoundPool(20, AudioManager.STREAM_MUSIC, 0);

}

```

Ya en el constructor le añadimos como parámetro una Activity y usamos el método setVolumeControlStream que nos permitirá cambiar el volumen con los controles de hardware del dispositivo y nos pide como parámetro una fuente de música que le indicamos que va a ser un stream de música a través de la clase AudioManager.

En nuestra variable assets almacenamos una instancia de la carpeta assets.

Y en la variable soundPool almacenamos un nuevo sonido SoundPool indicando como parámetro el numero máximo de sonidos simultáneos (20), el tipo de sonido

(AudioManager.STREAM_MUSIC) y la calidad de nuestro sonido (actualmente no tiene ningún efecto, por lo que su valor por defecto es 0).

```
public Music newMusic(String filename) {  
  
    try {  
  
        AssetFileDescriptor assetDescriptor = assets.openFd(filename);  
  
        return new AndroidMusic(assetDescriptor);  
  
    } catch (IOException e) {  
  
        throw new RuntimeException("Error al cargar: " + filename + "");  
  
    }  
  
}
```

Nos devolverá un nuevo objeto AndroidMusic que cargaremos desde la carpeta assets.

```
public Sound newSound(String filename) {  
  
    try {  
  
        AssetFileDescriptor assetDescriptor = assets.openFd(filename);  
  
        int soundId = soundPool.load(assetDescriptor, 0);  
  
        return new AndroidSound(soundPool, soundId);  
  
    } catch (IOException e) {  
  
        throw new RuntimeException("Error al cargar: " + filename + "");  
  
    }  
  
}
```

Y para terminar el método newSound nos devolverá un nuevo objeto AndroidSound que cargaremos desde la carpeta assets, almacenando en memoria su id con el método load.

En ambos casos manejamos la excepción IOException en caso de que algo vaya mal a la hora de cargar los archivos.

La clase AndroidSound se encarga del sonido y consta del siguiente código:

```
import android.media.SoundPool;

import com.example.slang.interfaces.Sound;

public class AndroidSound implements Sound {

    int soundId;

    SoundPool soundPool;

    public AndroidSound(SoundPool soundPool, int soundId) {

        this.soundId = soundId;

        this.soundPool = soundPool;

    }

    public void play(float volume) {

        soundPool.play(soundId, volume, volume, 0, 0, 1);

    }

    public void dispose() {

        soundPool.unload(soundId);

    }

}
```

Empezamos creando una variable int, que sera donde almacenaremos la id de nuestro efecto de sonido y creamos un objeto soundPool que nos ayudara a gestionar y reproducir los efectos de sonido.

En el constructor de la clase simplemente almacenamos los parámetros del constructor en las dos variables que hemos creado para esta clase.

Para reproducir los efectos de sonido usaremos el método play de la clase SoundPool:

```
play(soundID, leftVolume, rightVolume, priority, loop, rate)
```

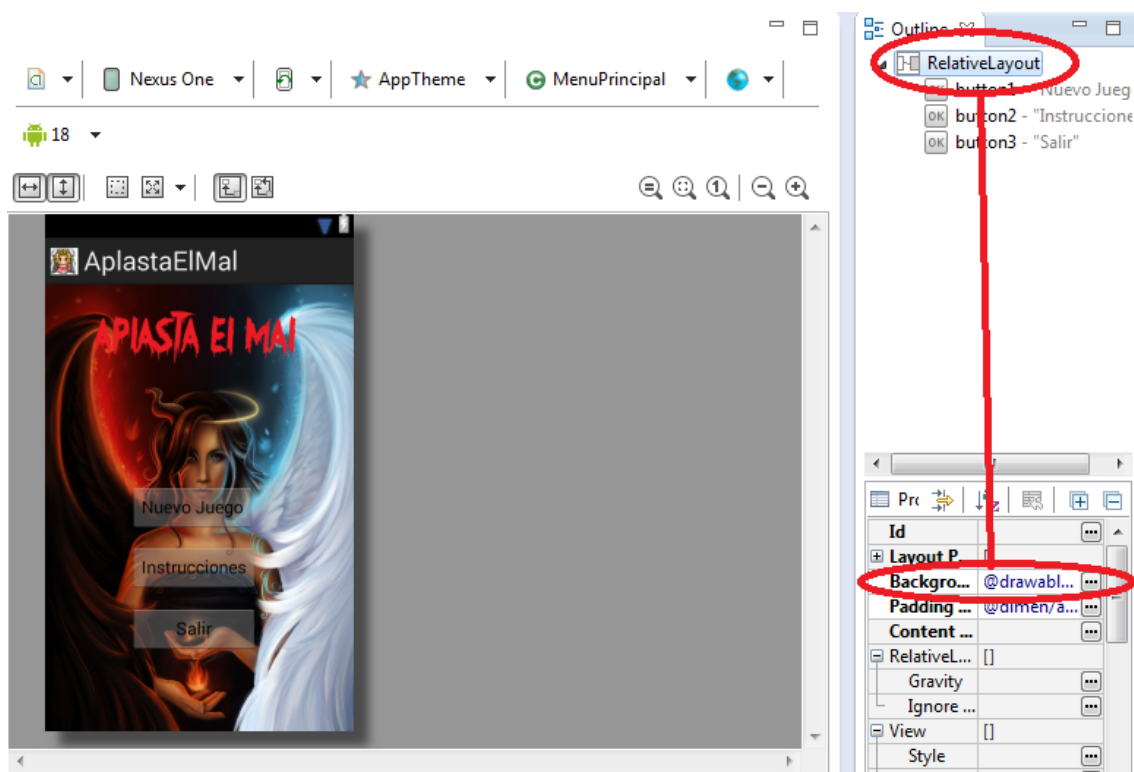
Nos pide como parámetro la id del sonido que la conseguiremos mas adelante con el método load de esta misma clase, el volumen del canal izquierdo y derecho (usaremos los parámetros de nuestro método en este caso, el rango va desde 0.0 a 1.0), la prioridad de reproducción (0 es la mas baja), modo loop (0 desactivado y -1 activado) y para finalizar la tasa de reproducción (el valor normal es 1, pero su rango va desde 0.5 a 2.0).

Para liberar el recurso de la memoria en nuestro SoundPool usaremos el método unload que nos pide como parámetro la id del sonido a liberar.

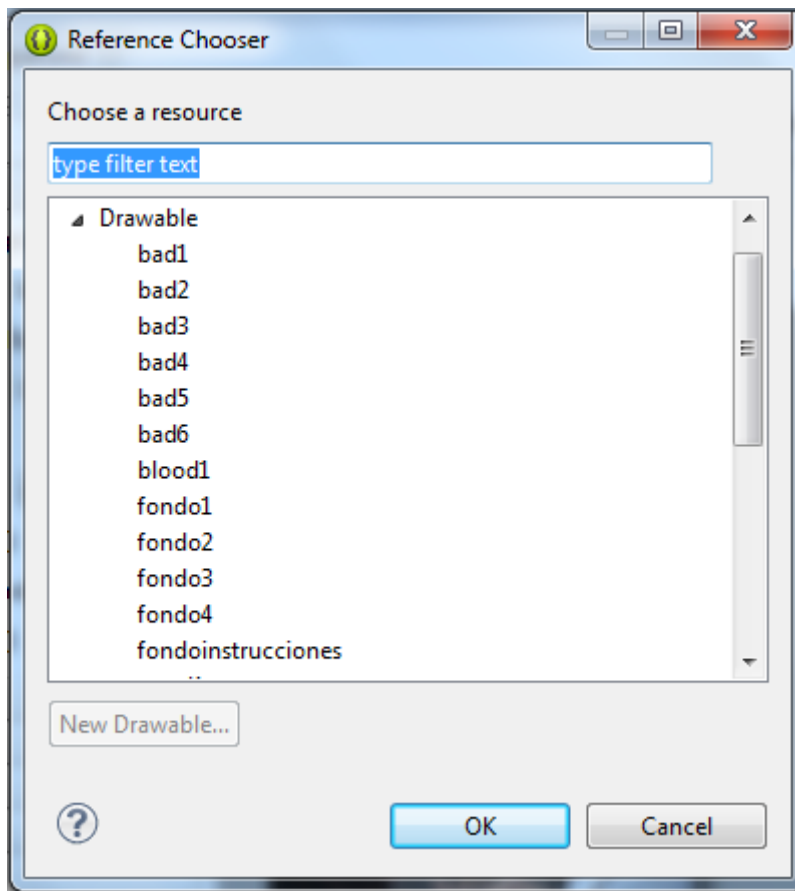
2.8 – FONDOS EN EL JUEGO

Los fondos en nuestro juego ayudan a mejorar su presencia, usamos fondos en el menú principal, en la clase Instrucciones y dentro del terreno de juego, antes de continuar hay que tener una cosa en cuenta, las imágenes de fondo tienen una resolución de 480x800 por lo que se visualización en dispositivos con pantalla mas grande no será la correcta.

Los fondos que hemos usado en las Activities MenúPrincipal e Instrucciones se colocan de la misma manera así que mostraremos como se hace solo en uno, primero localizamos el layout propio de nuestra Activity, en sus componentes picamos sobre el layout, en la zona de propiedades veremos el apartado Background, hacemos clic sobre ...:



Se nos abrirá una ventana en la cual veremos un listado de recursos, buscamos la parte de Drawable y la extendemos, elegimos la imagen y pulsamos sobre OK:



El fondo del juego es otro tema, se ha hecho un array de bitmaps y se ha declarado como atributo de la clase GameView, también hemos declarado un entero para guardar la posición, su uso lo veremos mas tarde:

```
private Bitmap[] fondos = {BitmapFactory.decodeResource(getResources(),  
R.drawable.fondo1),  
                        BitmapFactory.decodeResource(getResources(),  
R.drawable.fondo2),  
                        BitmapFactory.decodeResource(getResources(),  
R.drawable.fondo3),  
                        BitmapFactory.decodeResource(getResources(),  
R.drawable.fondo4)};
```

```
private int posicion;
```

Luego con la ayuda de Random hemos conseguido que la elección del fondo sea aleatorio cada vez que se construye la SurfaceView, por lo que ponemos en el constructor de GameView lo siguiente:

```
public GameView(Context context) {  
    super(context);
```

```
Random aleatorio = new Random();  
posicion = aleatorio.nextInt(fondos.length);
```

Guardamos en posición un numero aleatorio que depende de la longitud del array de fondos, luego en el método onDraw sustituimos el fondo negro del Canvas con:

```
canvas.drawBitmap(fondos[posicion], 0, 0, null);
```

Como apunte final indicar que las imágenes de fondo debido a su tamaño se han guardado en /res/drawable-hdpi

2.9 – CUESTIÓN DE RENDIMIENTO

El tema del rendimiento en los videojuegos es muy importante, un juego mal optimizado puede arruinarnos la experiencia jugable.

Que un juego esté bien optimizado supone que esta bien programado y que va a ser capaz de aprovechar los recursos del sistema en el que se ejecute (ordenador, consola, móvil...).

Lo primero que debemos tener en cuenta es que los objetos creados no son gratis, ocupan memoria, por lo que no debemos crear mas objetos de los necesarios, también debemos tener cuidado con el uso de la CPU, esto lo hemos visto en la parte que empezamos a trabajar con hilos.

No debemos trabajar con archivos muy grandes ya que su gestión puede ralentizar todo, también hay quien dice que si evitamos los getter y los setters internos puede aumentar el rendimiento el triple.

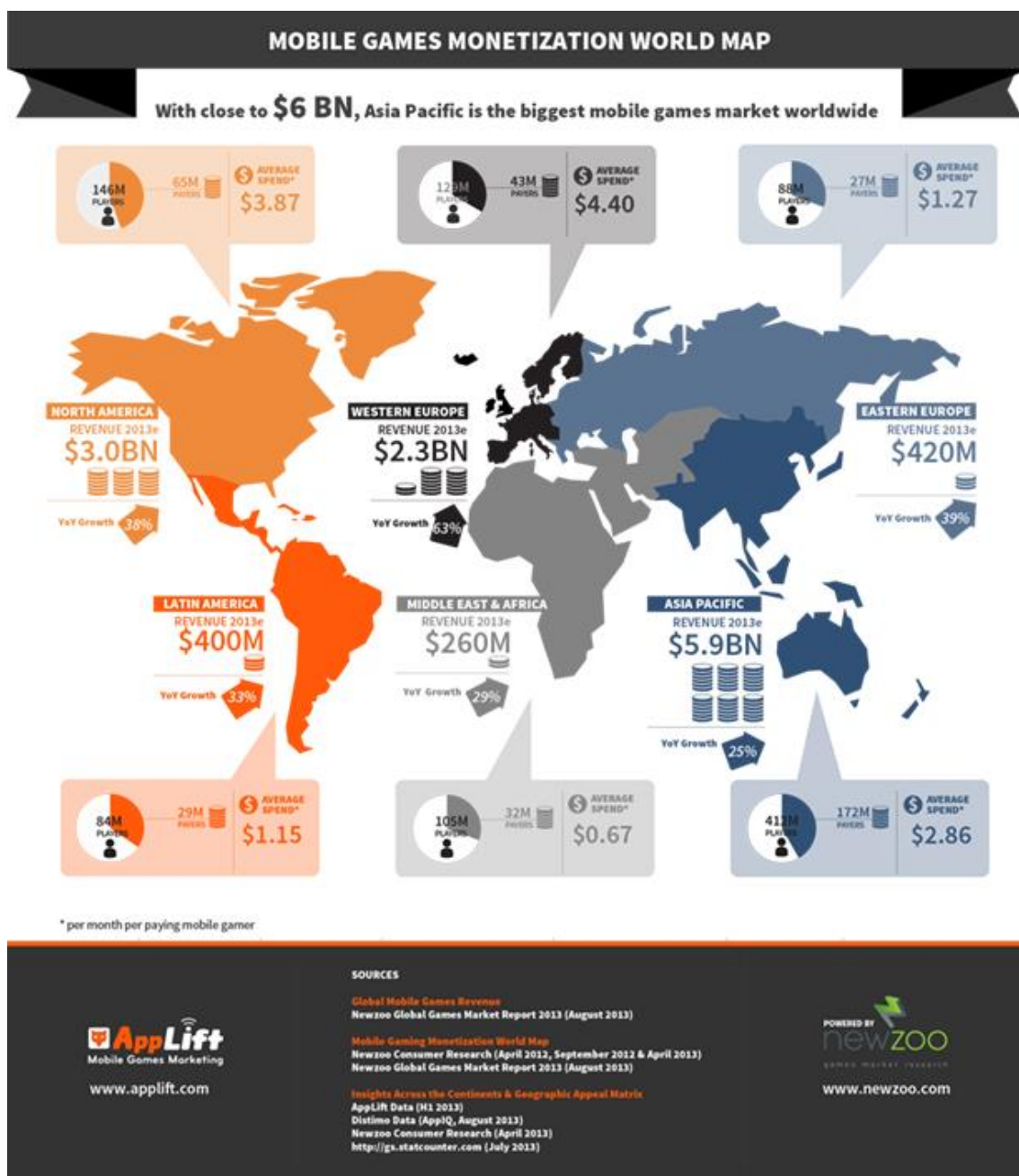
3 – EL JUEGO ESTÁ TERMINADO: ¿AHORA QUE?

Nuestro juego ya ha terminado, hemos impresionado a nuestros familiares y amigos, vale....¿y ahora que?

Podemos quedarnos el juego para nosotros y distribuirlo entre conocidos o bien quizás queramos dar el salto profesional y publicarlo en la tienda de Google.

3.1 – SITUACIÓN DEL MERCADO DE JUEGOS MÓVILES

El mercado de los juegos para dispositivos móviles está en crecimiento, a continuación podemos ver un gráfico de las previsiones de crecimiento del mercado hasta el 2016:



Esta imagen hace referencia a un estudio (adjunto el enlace en la bibliografía) del que se pueden sacar las siguientes conclusiones:

- El mercado global de juegos móviles crecerá anualmente un 27.3% y doblar en 2016 para alcanzar un valor de 23.9 mil millones de dólares.
- Este crecimiento deriva del crecimiento del número de jugadores, así como de un mayor consumo medio de contenidos y por un gasto medio más elevado por jugador.
- Los juegos para el formato Tabletas crecerá a un ritmo endiablado de 400% hasta 2016 para alcanzar un valor de mercado de 10 mil millones de dólares.
- El mercado en la actualidad se compone de 966 millones de jugadores, un 78% de los 1 200 millones de jugadores mundiales.
- 368 millones de consumidores en todo el mundo (38% de todo los jugadores) gasta mensualmente de media unos 2.78\$ en contenidos de juegos. En 2015 este número será un 50% y subirá hasta el 3.07 \$.
- Con un 48% de los ingresos globales, la región Asia-Pacífico es de lejos el mercado de juegos móviles más desarrollado.
- Las regiones occidentales y de este crecerán rápidamente a una velocidad por encima de los 33%. Dentro de las regiones asiáticas, China y otros países de la región serán crecimientos semejantes.
- El gasto medio por jugador es mayor en Europa (4,4\$) cuando comparado con USA (3,87\$), pero estos últimos con el mayor porcentaje de jugadores móviles e, el 45% de la población mundial de jugadores.
- El coste de adquisición de jugadores varía entre 1,11\$ y el 0,74\$ en iOS y Android respectivamente en los países de América Latina (mercado que necesita todavía madurar) y los costes más altos de entre 3,70\$ y 1,71\$ en el mercado asiático mucho más consolidado (incluye Australia)
- De forma general, los mercados de Europa Occidental, América del Norte y Asia Pacífico seguirán siendo los mercados más atractivos para los editores de videojuegos móviles.

Las conclusiones que sacamos de todo esto es que estamos en un buen momento para desarrollar juegos para dispositivos móviles, pero eso sí, vamos a tener mucha competencia.

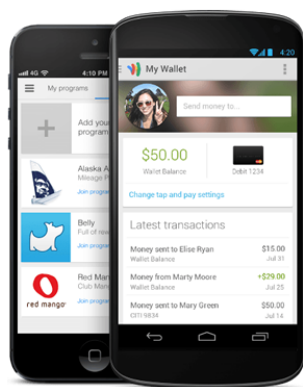
3.2 – Google Play Store

Google Play es la tienda de aplicaciones de Android (antes conocida como Android Market), en Octubre del 2012 se superaron las 700000 aplicaciones disponibles, no solo se pueden encontrar aplicaciones sino que también podemos encontrar libros, revistas, música e incluso películas y series.

Para vender aplicaciones debes crearte una cuenta de comerciante de Google Checkout, y subir el archivo de la aplicación a los servidores. Para crearte una cuenta

de comerciante deberás proporcionar información privada, de contacto y financiera. El precio de la aplicación se puede cambiar en cualquier momento siempre y cuando no la hayas publicado anteriormente como gratuita. Los intervalos de precios permitidos son entre 0,99 y 200 dólares estadounidenses. Los desarrolladores de las aplicaciones de pago reciben un 70% del precio total de la aplicación, mientras que el 30% restante es destinado a las empresas. El beneficio obtenido de 'Google Play' es pagado a los desarrolladores a través sus cuentas en el sistema Google Checkout.

La gran novedad que aporta Google Play hace referencia a los desarrolladores: estos serán capaces de hacer su contenido disponible en un servicio abierto el servicio de Google que ofrece una retroalimentación y sistema de calificación similar a YouTube. Los desarrolladores tendrán un entorno abierto y sin obstáculos para hacer su contenido disponible. El contenido puede subirse al mercado después de tres pasos: registrarse como comerciante, subir y describir su contenido y publicarlo. Para registrarse como desarrollador y poder subir aplicaciones hay que pagar una cuota de registro (US\$ 25,00) con tarjeta de crédito (mediante Google Checkout)



Envía dinero, lleva menos encima y ahorra mucho más

Envía dinero estés donde estés
Envía dinero a amigos y familiares a través de la aplicación o del sitio de Google Wallet, o adjuntando dinero en Gmail. [Más información](#)

Todas tus ofertas y tarjetas de fidelización en un único sitio
Canjea fantásticas ofertas de tus empresas favoritas con Google Wallet para ahorrar al realizar compras. [Más información](#)

Guarda tu tarjeta en Google Wallet





NOMBRE Y UBICACIÓN DEL DOMICILIO


España (ES)

Nombre Código postal

MÉTODO DE PAGO

Tarjeta de débito o de crédito

Número de tarjeta    

Fecha de caducidad / Código de seguridad 

Dirección de facturación

☒ La dirección de facturación es la misma que el nombre y la dirección particular

Acepto las [Condiciones de servicio](#) y el [Aviso de privacidad](#) de Google Wallet.

☒ Deseo recibir ofertas especiales de Google Wallet, boletines informativos e invitaciones para dar mi opinión sobre productos.

[Aceptar y crear](#)

En cuanto a restricciones hay que destacar el cambio de política que tuvo Google en el 2012 a la hora de publicar en Google Play, se establecieron condiciones como por ejemplo los desarrolladores tenían que abstenerse de usar nombres y/o iconos muy similares a aplicaciones populares para evitar engañar a los usuarios. La descripción de la aplicación tiene que ajustarse a lo que realmente realiza y no estar llena de palabras clave para manipular la clasificación o la relevancia en los resultados de búsqueda.

Las aplicaciones ya no pueden molestar a los usuarios con spam, queda prohibido usar las notificaciones, modificar marcadores e instalar accesos directos para mostrar publicidad no deseada. No pueden recopilar ni revelar información personal, ni enviar

SMS, correos u otro tipo de mensajes sin confirmación por parte del usuario y sin saber el destinatario.

Tampoco los desarrolladores pueden publicar aplicaciones Android que simplemente son un WebView de una página web que no es de su propiedad. No pueden ofrecer incentivos a los usuarios para que voten positivamente su aplicación y todos los pagos de dentro de las aplicaciones tienen que ser realizados a través de Google Play y Google Wallet, ya no está permitido usar servicios de terceros.

Este cambio de política fue importante por ser un cambio claramente orientado a proteger al usuario, los desarrolladores contaron con un plazo de 30 días para adaptarse, de no hacerlo se borrarían sus aplicaciones.

3.3 – LAS CLAVES DEL ÉXITO

Hoy por hoy los móviles y tabletas se han sumado a los ordenadores y consolas como plataformas de juegos, pero su filosofía no es la misma.

Los usuarios de móviles usan los juegos para sus dispositivos para rellenar momentos muertos, por ejemplo, un viaje en autobús o el esperar por alguien, por lo tanto se requiere rapidez, se requiere una jugabilidad directa y sin rodeos.

Los juegos complejos y con una historia absorbente rara vez se suelen ver en dispositivos móviles (aunque los hay) y esto es debido a la filosofía que trae consigo el ser una plataforma móvil.

Una clave que la experiencia nos ha demostrado que suele ser sinónimo de éxito es la simplicidad, el hacer un juego simple no tiene por que ser necesariamente malo, si no podemos ver fenómenos como AngryBirds o el CandyCrush, ambos juegos han triunfado por su apuesta directa y sin complicaciones.

Como conclusión podemos decir que las claves del éxito en un juego móvil son la jugabilidad directa y la sencillez.

3.3.1 – UN ÉXITO INESPERADO, FLAPPY BIRD

Hablando de sencillez y de éxito es obligado pararnos en uno de los mayores fenómenos del mundo de los juegos móviles, y no solo por su jugabilidad si no por el culebrón que ha traído consigo, hablamos, como no, del Flappy Bird.



Flappy Bird es un juego desarrollado por el Vietnamita Dong Nguyen que está en el punto de mira de la prensa especializada por los últimos acontecimientos, a continuación voy a poner por orden cronológico extractos de noticias relacionadas:

- 03/02/2014: El juego imposible Flappy Birds gobierna Apple Store y Google Play
- 07/02/2014: Flappy Bird, la sensación móvil que gana 36.000 euros al día
- 08/02/2014: El creador de Flappy Birds retira el exitoso juego
- 10/02/2014: iTunes se llena de copias de Flappy Bird
- 10/02/2014: El creador de Flappy Bird, amenazado de muerte
- 10/02/2014: Subastan un móvil con Flappy Bird por 90.000 dólares
- 10/02/2014: Nintendo no ha demandado a Flappy Bird
- 11/02/2014: Flappy Bird, retirado por "miedo a provocar adicción"
- 12/02/2014: El creador de VVVVVV crea un juego inspirado en Flappy Bird
- 15/02/2014: FlappyBaby, un nuevo clon basado en Flappy Bird
- 17/02/2014: Los clones de Flappy Bird, prohibidos por Google y Apple

Como podemos ver el éxito puede llegar de la manera mas inesperada, ahora solo queda una duda ¿sera todo verdad o solo una elaborada maniobra de marketing? Me temo que solo el tiempo lo dirá.

4 – BIBLIOGRAFIA

- Uso de la clase SurfaceView:
<http://developer.android.com/reference/android/view/SurfaceView.html>
- Uso de la clase Canvas:
<http://developer.android.com/reference/android/graphics/Canvas.html>
- Dibujo en SurfaceView, animaciones, uso de sprites, detección de colisiones:
<http://www.edu4java.com/androidgame.html>
- Implantación de música en el juego:
<http://elbauldeandroid.blogspot.com.es/2013/02/creacion-de-un-juego-3-modulo-audio.html>
- Uso de SharedPreferences (solo usuarios registrados):
<http://www.iessanandres.com/aulavirtual/mod/resource/view.php?id=4740>
- Consejos para la optimización de aplicaciones:
<http://web.ing.puc.cl/~iic3686/android2010-2/Ayudantia%20Android.pdf>
- Información sobre Google Play y sus políticas:
<http://www.xatakandroid.com/play-store/google-play-endurece-las-condiciones-para-publicar-aplicaciones-en-beneficio-de-los-usuarios>
<http://www.startcapps.com/blog/como-registrarse-como-desarrollador-en-google-play-y-app-store/>
http://es.wikipedia.org/wiki/Google_Play#Aplicaciones
- Situación del mercado de aplicaciones móviles:
<http://www.danielparente.net/es/2013/10/30/crecimiento-del-mercado-de-juegos-moviles/>
- Recopilación de noticias sobre Flappy Bird:
http://www.meristation.com/search/apachesolr_search/noticias/flappy%20bird

- Sprites usados:
<http://translate.google.co.uk/translate?hl=en&sl=ja&tl=es&u=http%3A%2F%2Fwww.famitsu.com%2Ffreegame%2Ftool%2Fchibi%2Findex2.html&sandbox=1>