

PRACTICA 1: ASTEROIDES X

Parte I: ¿Cuándo se llama a los eventos del ciclo de vida en una actividad?

En este ejercicio vamos a implementa todos los métodos del ciclo de vida de la actividad principal de Asteroides y añadiremos un **toast** para mostrar cuando se ejecuta. De esta forma comprenderemos mejor cuando se llama a cada método.

Paso 1. Abre la actividad *Asteroides* del proyecto Asteroides.

Paso 2. Añade en el método `onCreate()` el siguiente código:

`Toast.makeText(this,"onCreate", Toast.LENGTH_SHORT).show();`

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

    //bAcercaDe.setBackgroundResource(R.drawable.degradado);
    bAcercaDe = findViewById(R.id.button03);
    bAcercaDe.setOnClickListener((view) -> {
        bAcercaDe.startAnimation(animacion);
        lanzarAcercaDe( view: null);
    });

    TextView texto = (TextView) findViewById(R.id.textView);
    animacion = AnimationUtils.loadAnimation( context: this, R.anim.giro_con_zoom);
    texto.startAnimation(animacion);

    bPlay = (Button) findViewById(R.id.button01);
    Animation animacion2 = AnimationUtils.loadAnimation( context: this, R.anim.aparecer);
    bPlay.startAnimation(animacion2);

    bConfigurar = (Button) findViewById(R.id.button02);
    Animation animacion3 = AnimationUtils.loadAnimation( context: this, R.anim.desplazamiento_derecha);
    bConfigurar.startAnimation(animacion3);

    /*bSalir = findViewById(R.id.button04);
    bSalir.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            finish();
        }
    });*/

    libreria= GestureLibraries.fromRawResource( context: this, R.raw.gestures);
    if(!libreria.load()) {
        finish();
    }
    //GestureOverlayView gesturesView = (GestureOverlayView) findViewById(R.id.gestures);
    //gesturesView.addOnGesturePerformedListener(this);

    Toast.makeText( context: this, text: "onCreate", Toast.LENGTH_SHORT).show();
}
```

Paso 3. Añade los siguientes métodos:

```
@Override protected void onStart() {
    super.onStart();
    Toast.makeText( context: this, text: "onStart", Toast.LENGTH_SHORT).show();
}

@Override protected void onResume() {
    super.onResume();
    Toast.makeText( context: this, text: "onResume", Toast.LENGTH_SHORT).show();
}

@Override protected void onPause() {
    Toast.makeText( context: this, text: "onPause", Toast.LENGTH_SHORT).show();
    super.onPause();
}

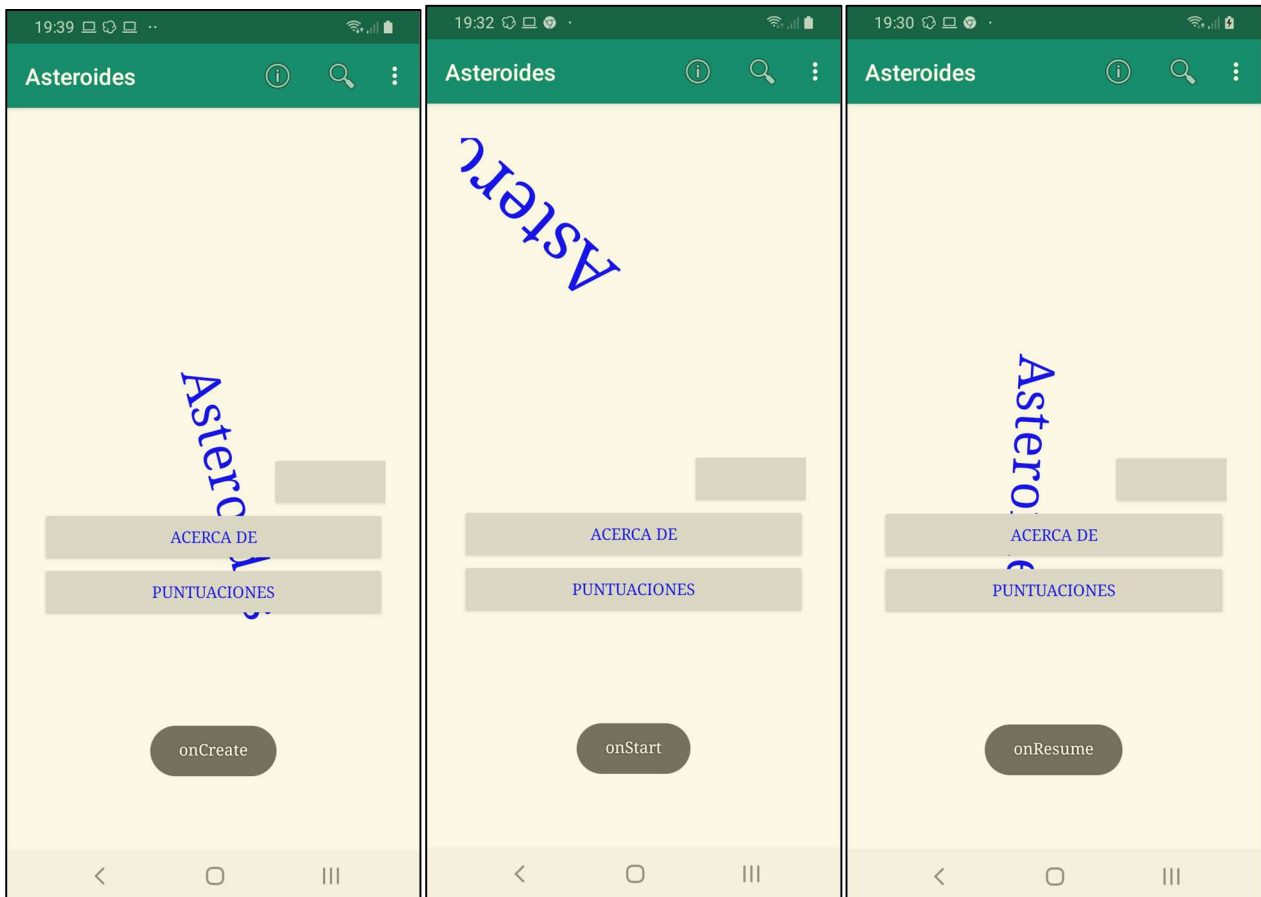
@Override protected void onStop() {
    Toast.makeText( context: this, text: "onStop", Toast.LENGTH_SHORT).show();
    super.onStop();
}

@Override protected void onRestart() {
    super.onRestart();
    Toast.makeText( context: this, text: "onRestart", Toast.LENGTH_SHORT).show();
}

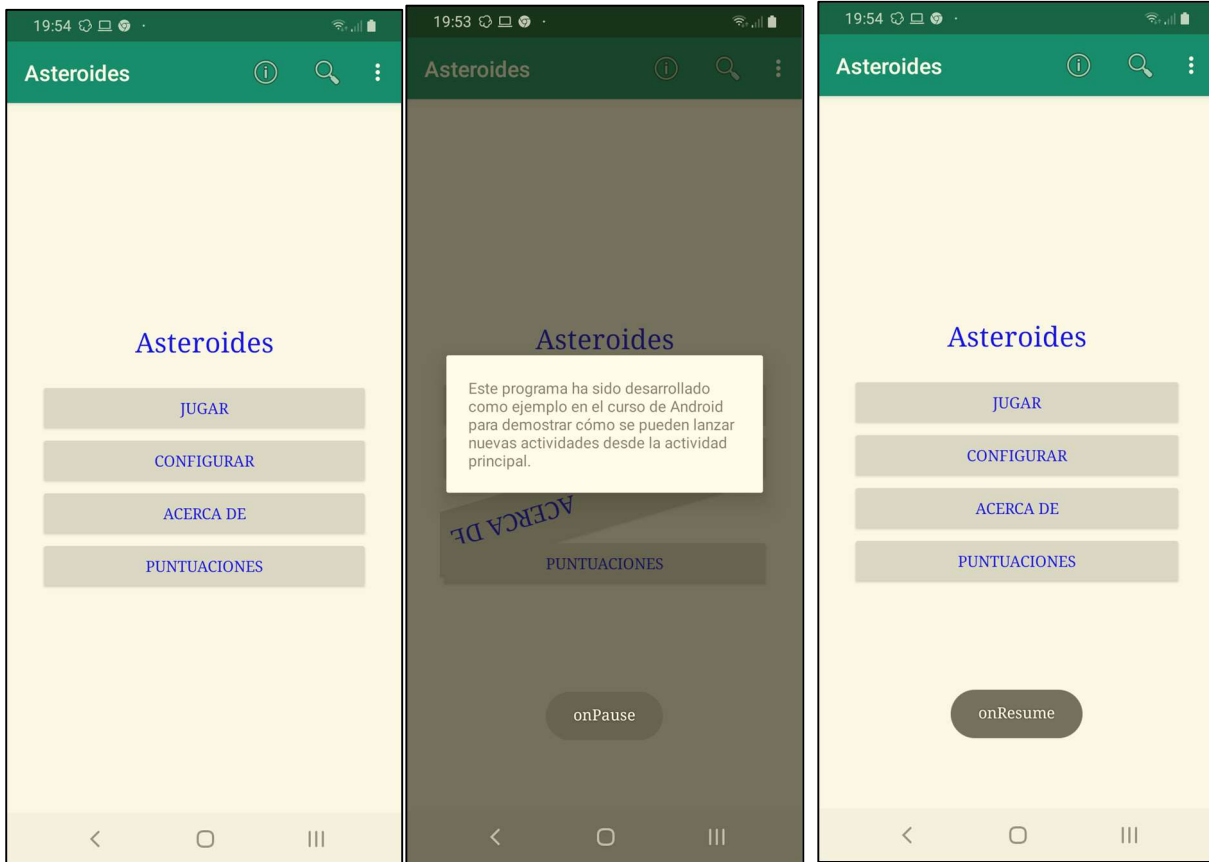
@Override protected void onDestroy() {
    Toast.makeText( context: this, text: "onDestroy", Toast.LENGTH_SHORT).show();
    super.onDestroy();
}
```

Paso 4. Ejecuta la aplicación y observa la secuencia de **Toast**.

Al arrancar pasa por los siguientes eventos. onCreate, onStart y onResume.

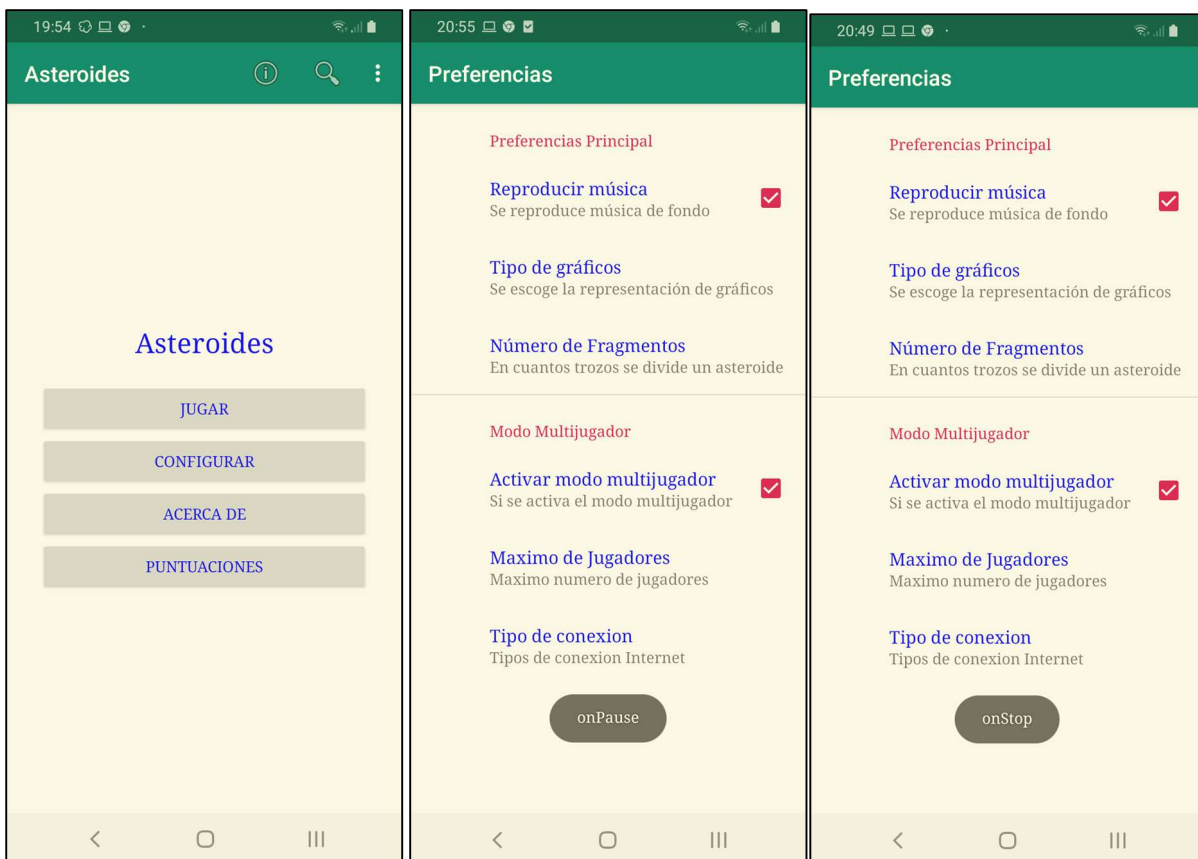


Paso 5. Pulsa el botón *Acerca de* y luego regresa a la actividad. Observa la secuencia de **Toast**.
Al hacer click en el botón *Acerca de*, pasa por el evento `onPause` y cuando vuelve por `onResume`.

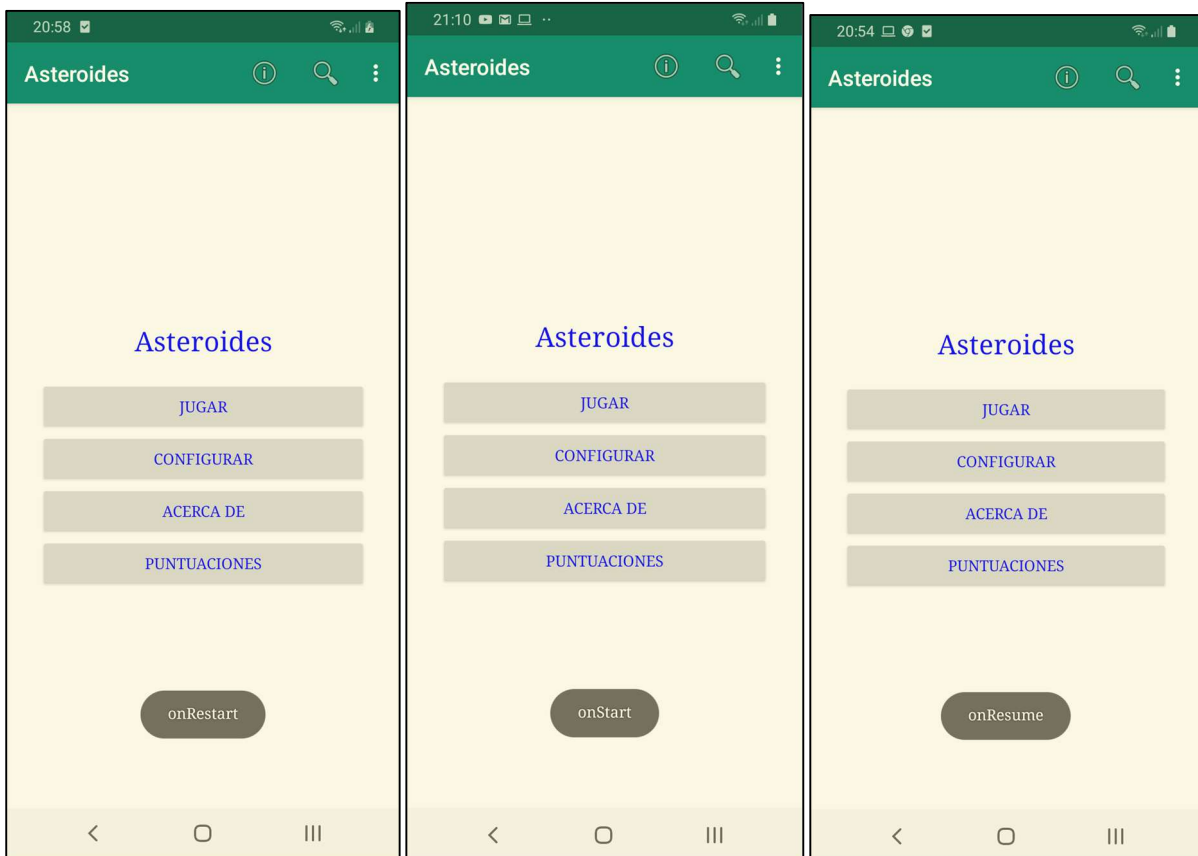


Paso 6. Selecciona la opción *Preferencias* y luego regresa a la actividad. Observa la secuencia de **Toast**.

Pasa por `onPause` (no se consigue apreciar visualmente) y por `onStop`.

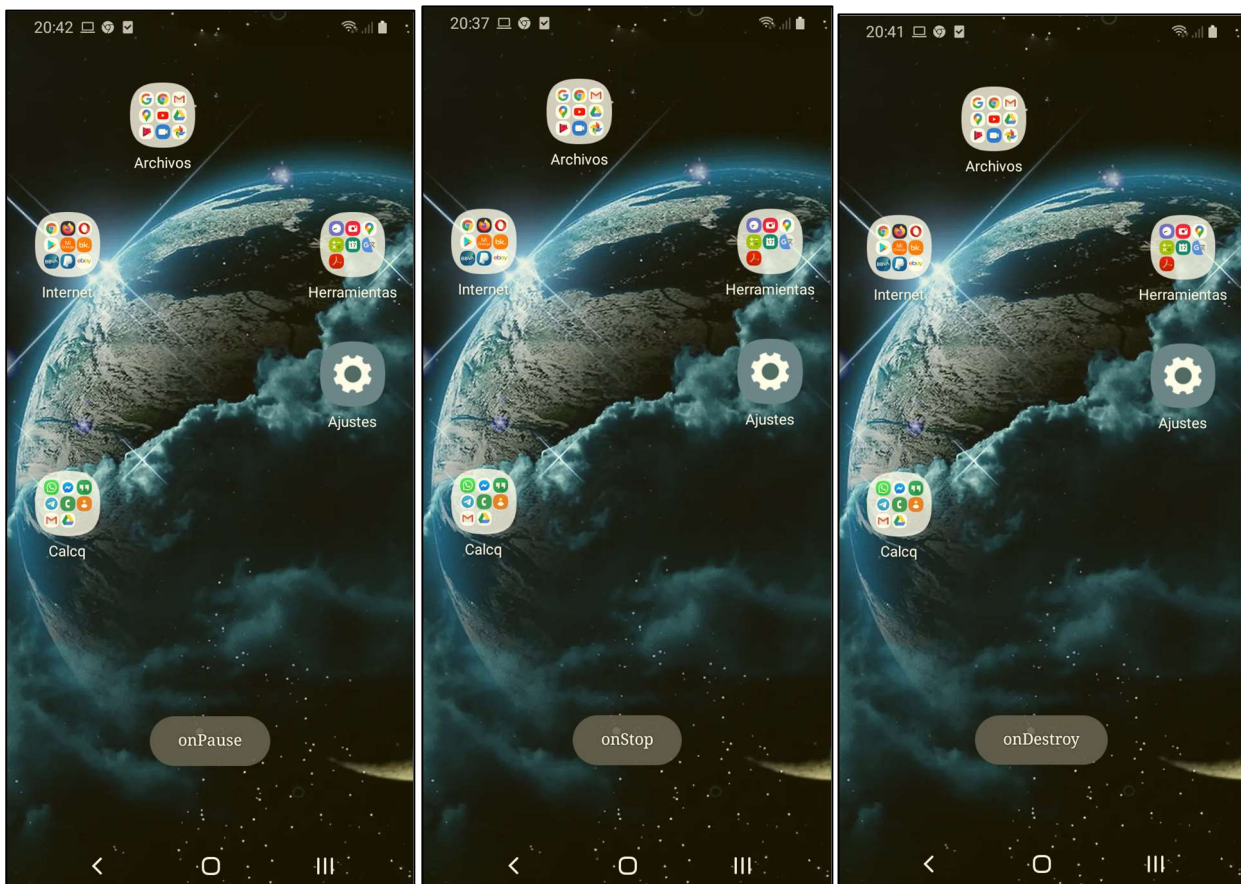


Al pulsar Retroceso y regresar a la actividad se pasa por la fase de onRestart, onStart y onResume.



Paso 7. Sal de la actividad y observa la secuencia de Toast.

Si salimos de la actividad se pasa por onPause, después por onStop y por último por onDestroy



Parte II: Aplicando eventos del ciclo de vida en la actividad inicial de Asteroides

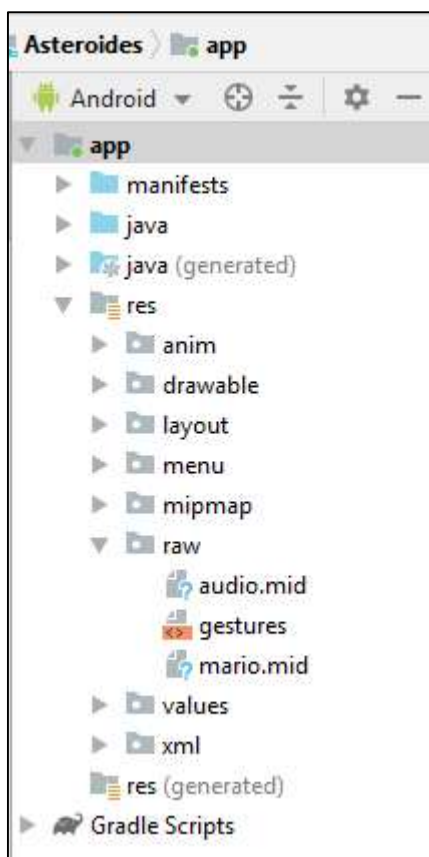
Los conceptos referentes al ciclo de vida de una aplicación son imprescindible para el desarrollo de aplicaciones estables en Android. Para reforzar estos conceptos te proponemos el siguiente ejercicio en el que vamos a reproducir una música de fondo en la actividad principal.

Paso 1. Abre el proyecto Asteroides.

Paso 2. Busca un fichero de audio (en este capítulo se listan los formatos soportados por Android). Renombra este fichero a *audio.xxx* y cópialo a la carpeta *res/raw*.

NOTA: Cada vez que ejecutes el proyecto este fichero será añadido al paquete .apk. Si este fichero es muy grande la aplicación también lo será, lo que ralentizará su instalación. Para agilizar la ejecución te recomendamos un fichero muy pequeño, por ejemplo un .mp3 de corta duración o un fichero MIDI (.mid). Si no encuentras ninguno puedes descargar este:

<http://www.dcomg.upv.es/~jtomas/android/ficheros/audio.mid>.



Paso 3. Abre la actividad MainActivity y declara el siguiente objeto:

```
public class MainActivity extends AppCompatActivity
    implements GestureOverlayView.OnGesturePerformedListener {

    private MediaPlayer mp;

    private GestureLibrary libreria;
```

Paso 4. Añade las siguientes líneas en el método onCreate():

```

    libreria= GestureLibraries.fromRawResource( context: this, R.raw.gestures);
    if(!libreria.load()) {
        finish();
    }
    //GestureOverlayView gesturesView = (GestureOverlayView) findViewById(R.id.gestures);
    //gesturesView.addOnGesturePerformedListener(this);

    Toast.makeText( context: this, text: "onCreate", Toast.LENGTH_SHORT).show();

    mp = MediaPlayer.create( context: this, R.raw.audio);
    mp.start();
}

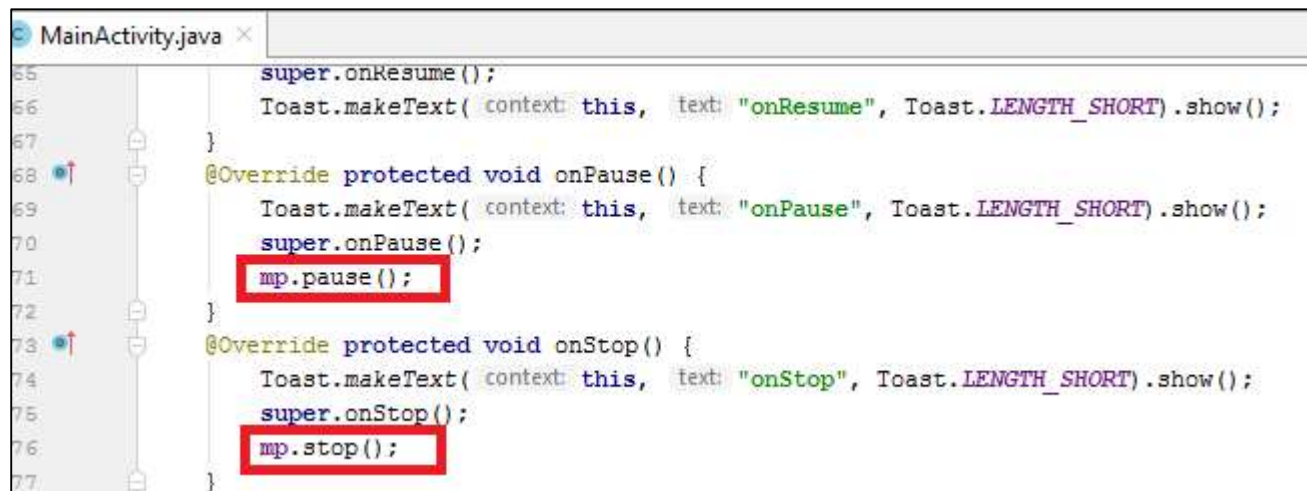
```

Paso 5. Ejecuta el proyecto y verifica que cuando sales de la actividad la música sigue sonando un cierto tiempo.

Si, sigue funcionando un cierto tiempo después de salir de la aplicación.

Paso 6. Utilizando los eventos del ciclo de vida queremos que cuando la actividad deje de estar **activa** el audio deje de escucharse. Puedes utilizar los métodos mp.pause() y/o mp.start().

Hemos visto que si salimos de la actividad se pasa por onPause, onStop y onDestroy. Podemos hacer el siguiente reparto:



```

MainActivity.java
65      super.onResume();
66      Toast.makeText( context: this, text: "onResume", Toast.LENGTH_SHORT).show();
67  }
68  @Override protected void onPause() {
69      Toast.makeText( context: this, text: "onPause", Toast.LENGTH_SHORT).show();
70      super.onPause();
71      mp.pause();
72  }
73  @Override protected void onStop() {
74      Toast.makeText( context: this, text: "onStop", Toast.LENGTH_SHORT).show();
75      super.onStop();
76      mp.stop();
77  }

```

Paso 7. Verifica que funciona correctamente.

Funciona correctamente al finalizar la aplicación finaliza el sonido.

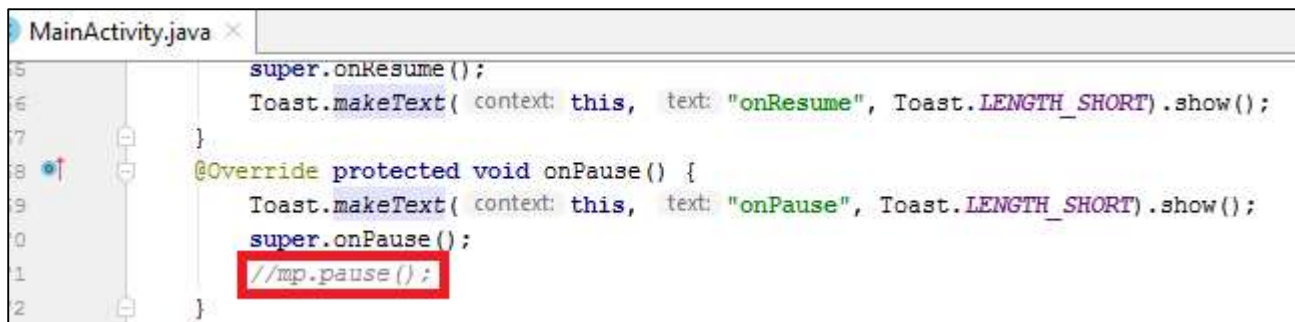
Parte III: Aplicando eventos del ciclo de vida en la actividad inicial de Asteroides(II)

Paso 1. Tras realizar el ejercicio anterior, ejecuta la aplicación y abre la actividad Acerca de... La música ha de detenerse.

Esta parte funciona correctamente. Al hacer click en el botón Acerca de, pasa por el evento onPause, donde precisamente hemos puesto un mp.pause(). Pero al volver no se reactiva la música.

Paso 2. Nos interesa que mientras parte de esta actividad esté visible (como ha ocurrido en el punto anterior) la música se escuche. Es decir, utilizando los eventos del ciclo de vida queremos que cuando la actividad deje de estar **visible** el audio deje de escucharse.

En el evento onPause se debe de comentar la función mp.pause().



```

MainActivity.java x
5      super.onResume();
6      Toast.makeText( context: this, text: "onResume", Toast.LENGTH_SHORT).show();
7  }
8  @Override protected void onPause() {
9      Toast.makeText( context: this, text: "onPause", Toast.LENGTH_SHORT).show();
10     super.onPause();
11     //mp.pause();
12 }

```

De esta forma al pasar de MainActivity a Acerca de ...la música continua ejecutándose.

Paso 3. Verifica que cuando abres la actividad Acerca de...la música continua reproduciéndose.

Si, continúa reproduciéndose.

Paso 4. Pasa ahora a una actividad que ocupe la totalidad de la pantalla (por ejemplo la actividad Juego o VistaLugarActivity). En teoría la música tendría que detenerse, dado que la actividad MainActivity ya no es visible y tendría que haber pasado a estado parada. Observa como la música acaba deteniéndose, pero es posible que tarde unos segundos. Esto se debe a que la llamada al método onStop() no es prioritaria, por lo que el sistema puede retardar su ejecución. En caso de tratarse de información visual, en lugar de acústica, este retardo no tendría una repercusión directa para el usuario, dado que la actividad no es visible.

Si queremos volver a la Actividad principal y que vuelva a sonar la música, debemos llamar en onStop() a mp.pause(), puesto que si dejamos mp.stop() es más complicado hacer que continúe después (se debe llamar al método prepare()). En cambio utilizando el método pause(), basta con llamar a mp.start() para activar el player.

- Cuando salimos hacia Acerca de, se pasa por el evento onPause y cuando vuelve por onResume.
- Cuando salimos de MainActivity hacia otra actividad, se pasa por onPause y por onStop. Al regresar a la actividad se pasa por la fase de onRestart, onStart y onResume.

Entonces finalmente quedaría la siguiente solución:

- onStop→ mp.pause();
- onRestart→mp.start();
- onDestroy→mp.stop();

Elegimos onRestart porque este evento no se da cuando arranca la aplicación y así de esta forma evitamos hacer 2 mp.start() entre onCreate y onStart al iniciar la aplicación.

```

MainActivity.java x
6      Intent i = new Intent( packageContext: this, Puntuaciones.class);
7      startActivity(i);
8  }
9
10  @Override protected void onStart() {
11      super.onStart();
12      Toast.makeText( context: this, text: "onStart", Toast.LENGTH_SHORT).show();
13  }
14  @Override protected void onResume() {
15      super.onResume();
16      Toast.makeText( context: this, text: "onResume", Toast.LENGTH_SHORT).show();
17  }
18  @Override protected void onPause() {
19      Toast.makeText( context: this, text: "onPause", Toast.LENGTH_SHORT).show();
20      super.onPause();
21  }
22  @Override protected void onStop() {
23      Toast.makeText( context: this, text: "onStop", Toast.LENGTH_SHORT).show();
24      super.onStop();
25      mp.pause();
26  }
27  @Override protected void onRestart() {
28      super.onRestart();
29      Toast.makeText( context: this, text: "onRestart", Toast.LENGTH_SHORT).show();
30      mp.start();
31  }
32  @Override protected void onDestroy() {
33      Toast.makeText( context: this, text: "onDestroy", Toast.LENGTH_SHORT).show();
34      super.onDestroy();
35      mp.stop();
36  }
37  }

```

Paso 5. Tras el problema detectado en el punto anterior, deshaz los cambios introducidos en esta práctica y deja la aplicación como se pedía en la práctica anterior.

Sólo hay que comentar las referencias al objeto MediaPlayer mp en onStop, onStart y en onDestroy.

```

@Override protected void onStop() {
    Toast.makeText( context: this, text: "onStop", Toast.LENGTH_SHORT).show();
    super.onStop();
    //mp.pause();
}

@Override protected void onRestart() {
    super.onRestart();
    Toast.makeText( context: this, text: "onRestart", Toast.LENGTH_SHORT).show();
    //mp.start();
}

@Override protected void onDestroy() {
    Toast.makeText( context: this, text: "onDestroy", Toast.LENGTH_SHORT).show();
    super.onDestroy();
    //mp.stop();
}

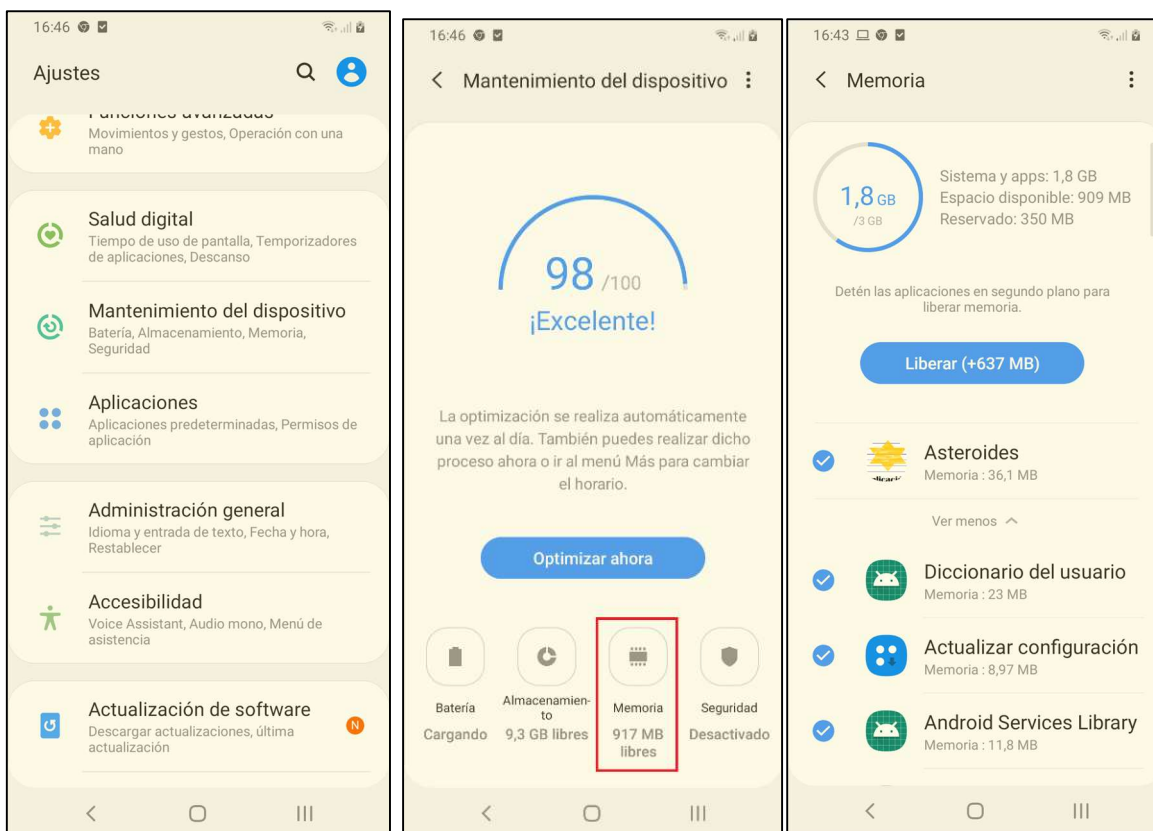
```


Parte IV: Aplicando eventos del ciclo de vida en la actividad Juego de Asteroides

Asteroides gestiona el movimiento de los objetos gráficos por medio de un *thead* que se ejecuta continuamente. Cuando la aplicación pasa a segundo plano, este *thead* continúa ejecutándose por lo que puede hacer que nuestro teléfono funcione más lentamente. Este problema aparece por una gestión incorrecta del ciclo de vida. El siguiente ejercicio veremos cómo solucionarlo:

Paso 1. Abre el proyecto Asteroides y ejecútalo en un terminal.

Paso 2. Pulsa el botón “Jugar” y cuando esté en mitad de la partida pulsa con el botón de *Inicio* (o *Casa*) para dejar a la actividad en estado *parada*. Las actividades en este estado no tendrían que consumir demasiados recursos. Sin embargo Asteroides sí que lo hace. Para verificarlos utiliza la aplicación “Administrador de tareas” (En las últimas versiones de Android, puedes abrirlo pulsando un segundo sobre el botón *Casa* y seleccionando el icono con forma de gráfico de tarta que aparece abajo a la izquierda). El resultado puede ser similar al que se muestra a continuación.



NOTA: Si el administrador de tareas de tu terminal no te permite mostrar el porcentaje de uso de la CPU te recomendamos que instales un programa que te lo permita. Por ejemplo, OS Monitor.

Como puedes ver la aplicación Asteroides está consumiendo casi el 50% del uso de CPU. Evidentemente algo hemos hecho mal. No es lógico que cuando la actividad **Juego** esté en segundo plano se siga llamando a `actualizaFisica()`. En este ejercicio aprenderemos a solucionarlo.

Paso 3. Incluye la siguiente variable en la actividad **Juego**.

```
public class Juego extends AppCompatActivity {  
    private VistaJuego vistaJuego;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.juego);  
    }  
}
```

Paso 4. Al final de `onCreate` añade:

```
public class Juego extends AppCompatActivity {

    private VistaJuego vistaJuego;

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

Paso 5. Incorpora los siguientes métodos a la actividad:

```
public class Juego extends AppCompatActivity {

    private VistaJuego vistaJuego;

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

    @Override protected void onPause() {
        super.onPause();
        vistaJuego.getThread().pausar();
    }

    @Override protected void onResume() {
        super.onResume();
        vistaJuego.getThread().reanudar();
    }

    @Override protected void onDestroy() {
        vistaJuego.getThread().detener();
        super.onDestroy();
    }
}
```

Lo que intentamos hacer con este código es poner en pausa el thread secundario cuando la actividad deje de estar activa y reanudarlo cuando la actividad recupere el foco. Además detener el thread cuando la actividad vaya a ser destruida.

NOTA: Realmente el thread será destruido al destruirse la actividad que lo ha lanzado. No obstante puede resultar interesante hacerlo lo antes posible. Observa como los eventos del ciclo de vida solo pueden ser escritos un `Activity`, por lo que no sería válido hacerlo en `VistaJuego`.

Paso 6. Abre la clase `VistaJuego` y busca la definición de la clase `ThreadJuego` y reemplázala por la siguiente:

```

public class ThreadJuego extends Thread {
    @Override
    public void run() {
        while (true) {
            actualizaFisica();
        }
    }
}

```

```

public class ThreadJuego extends Thread {

    private boolean pausa, corriendo;

    public synchronized void pausar() {
        pausa = true;
    }

    public synchronized void reanudar() {
        pausa = false;
        notify();
    }

    public void detener() {
        corriendo = false;
        if (pausa) reanudar();
    }

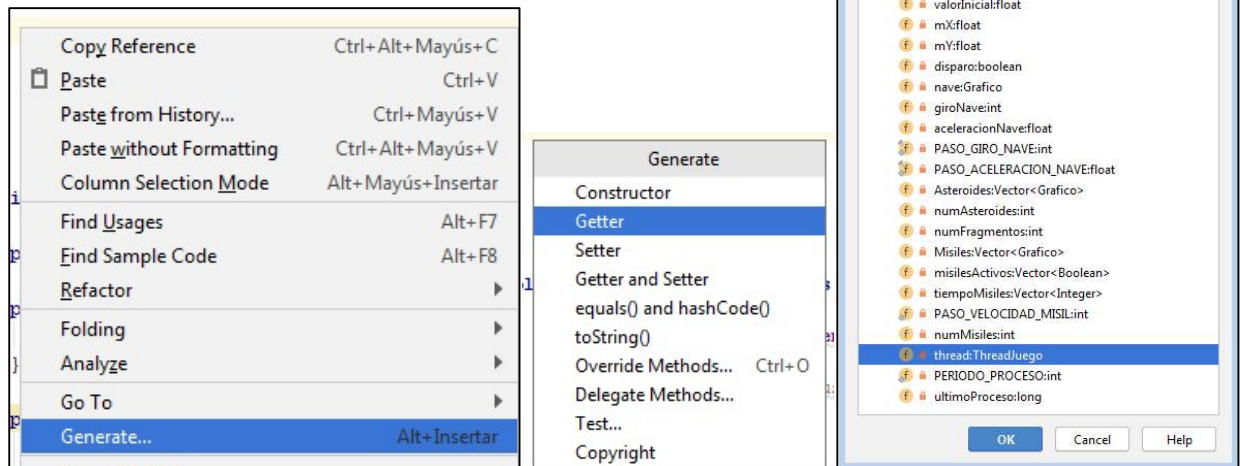
    @Override public void run() {
        corriendo = true;
        while (corriendo) {
            actualizaFisica();
            synchronized (this) {
                while (pausa)
                    try {
                        wait();
                    } catch (Exception e) {
                        //
                    }
            }
        }
    }
}

```

Comenzamos declarando las variables **pausa**, **corriendo**. Estas pueden ser modificadas mediante los métodos **pausar()**, **reanudar()** y **detener()**.

La palabra reservada **synchronized** impide el acceso concurrente a una sección del código. En la siguiente unidad explicaremos su utilización. El método **run()** se ha modificado de manera que en lugar de ser un bucle infinito, permitimos que termine poniendo la variable **corriendo** a **false**. Luego, tras llamar a **actualizaFisica()**, se comprueba si se ha activado **pausa**. En tal caso, se entra en un bucle donde ponemos en espera el *thread* llamando al método **wait()**. Este quedará bloqueado hasta que se llame a **notify()**. Esta acción se realizará desde el método **reanudar()**. La llamada a **wait()** puede lanzar excepciones por lo que es obligatorio escribirla dentro de un bloque **try {...} catch {...}**.

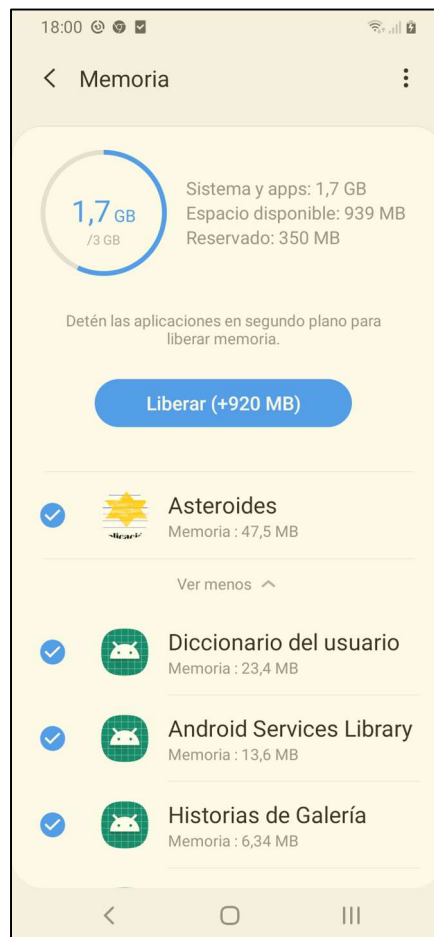
Paso 7. Dado que la variable **thread** es de tipo **private**, no puede ser manipulada desde fuera de **VistaJuego**. Para poder llamar a los métodos (**pausar**, **reanudar**,...) de este objeto vamos a incluir un método **getter**. Para ello, sitúa el cursor justo antes de la última llave de **VistaJuego**. Pulsa con el botón derecho en el código y selecciona la opción **Source / Generate Getters and Setters...** Marca solo el método **getThread()** tal y como se muestra a continuación:



Se insertará el siguiente código:

```
public ThreadJuego getThread() {
    return thread;
}
```

Paso 8. Ejecuta de nuevo la aplicación y repite el segundo punto de este ejercicio. En este caso el resultado ha de ser similar al siguiente:



Parece ser que después de todo, en las nuevas versiones no sólo no se gana nada sino que se pierde

Parte V: Aplicando eventos del ciclo de vida en la actividad Juego para desactivar los sensores

En la unidad anterior aprendimos a utilizar los sensores para manejar la nave en Asteroides. El uso de sensores ha de realizarse con mucho cuidado dado su elevado consumo de batería. Resulta importante que cuando nuestra actividad quedara en un segundo plano se detuviera la lectura de los sensores, para que así no siga consumiendo batería.

Paso 1. Crea los métodos `activarSensores()` y `desactivarSensores()` en la clase `VistaJuego`.



```
VistaJuego.java x
5
6
7
8 public void activarSensores() {
9
10 }
11 public void desactivaSensores() {
12
13 }
```

Paso 2. Mueve la línea de código que activa los sensores desde `onCreate()` al método `activarSensores()`.

Para que haya visibilidad del dato `mSensorManager` en toda la clase `VistaJuego` se debe definir como dato miembro. También una variable `context` que nos guarde el contexto de `VistaJuego`, ya que sino después va a resultar muy complicado obtener una referencia al contexto desde dentro de la función `activarSensores`:

```
//////////SENSORES//////////
private SensorManager mSensorManager;
private Context mcontext;
```

```
//////////CONSTRUCTOR VISTAJUEGO////////////////////////////////////////
mcontext=context;
}

public void activarSensores(){
    mSensorManager = (SensorManager) mcontext.getSystemService(Context.SENSOR_SERVICE);
    List<Sensor> listSensors = mSensorManager.getSensorList(Sensor.TYPE_ORIENTATION);
    if (!listSensors.isEmpty()) {
        Sensor orientationSensor = listSensors.get(0);
        mSensorManager.registerListener( listener: this, orientationSensor, SensorManager.SENSOR_DELAY_GAME);
    }
}
```

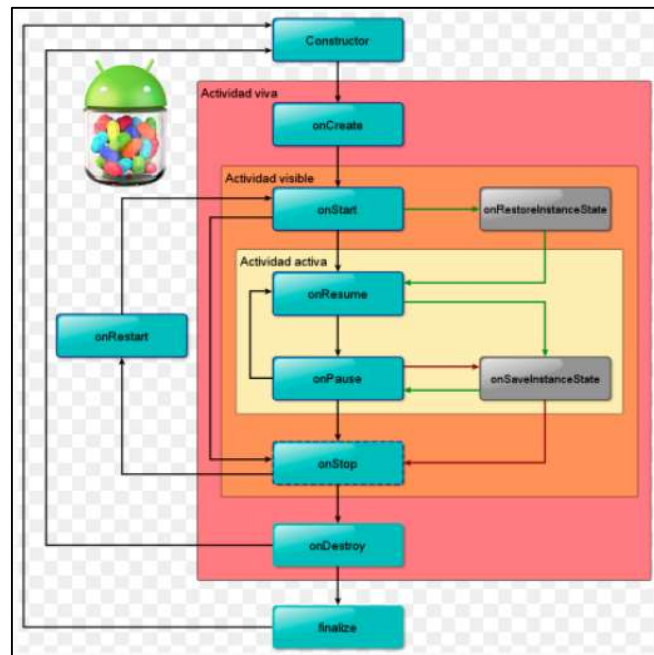
Paso 3. Para desactivar los sensores hay que llamar al siguiente método:

```
public void desactivaSensores(){
    mSensorManager.unregisterListener(this);
}
```

El parámetro de este método corresponde al objeto `SensorEventListener` del que queremos dejar de recibir eventos. En nuestro caso, nosotros mismos (`this`). Utiliza este método dentro de `desactivarSensores()`.

Paso 4. Utiliza los métodos del ciclo de vida adecuados para activar o desactivar los sensores. Recuerda que estos métodos los recoge la actividad, por lo que tendrás que incluirlos en la clase `Juego`.

Los eventos que utilizaremos de Juego serán `onResume` y `onPause`, que son los eventos que controlan que la actividad está completamente activa.



`onResume` → `activarSensores()`

`onPause` → `desactivarSensores()`

```

public class Juego extends AppCompatActivity {

    private VistaJuego vistaJuego;

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

    @Override protected void onPause() {
        super.onPause();
        vistaJuego.desactivaSensores();
        vistaJuego.getThread().pausar();
    }

    @Override protected void onResume() {
        super.onResume();
        vistaJuego.activarSensores();
        vistaJuego.getThread().reanudar();
    }

    @Override protected void onDestroy() {
        vistaJuego.getThread().detener();
        super.onDestroy();
    }
}

```

Parte VI: Guardando el estado en la actividad inicial de Asteroides

Paso 1. Ejecuta el proyecto Asteroides.

Paso 2. Cambia de orientación el teléfono. Observarás como la música se reinicia cada vez que lo haces.

No exactamente, se solapan dos ejecuciones de la canción.

Cuando arranca la aplicación pasa por onCreate, onStart, onResume.

Cuando se cambia de pantalla pasa por onPause, onStop, onSaveInstanceState y onDestroy (se sale de la pantalla inicial) y a continuación se empieza a cargar el layout landscape: onCreate, onStart, onRestoreInstanceState y onResume. A pasar dos veces por onCreate se solapan las dos melodías de las canciones.

Paso 3. Utilizando los métodos para guardar el estado de una actividad, trata de que cuando se voltea el teléfono, el audio continúe en el mismo punto de reproducción. Puedes utilizar los siguientes métodos:

```
int pos = mp.getCurrentPosition();  
mp.seekTo(pos);
```

Lo primero que hay que hacer es pausar la música en el evento onStop y guardar por donde va en onSaveInstanceState.

```
@Override protected void onStop() {  
    Toast.makeText( context: this, text: "onStop", Toast.LENGTH_SHORT).show();  
    super.onStop();  
    mp.pause();  
}  
  
@Override protected void onSaveInstanceState(Bundle estadoGuardado) {  
    super.onSaveInstanceState(estadoGuardado);  
    Toast.makeText( context: this, text: "onSaveInstanceState", Toast.LENGTH_SHORT).show();  
    if (mp != null) {  
        int pos = mp.getCurrentPosition();  
        estadoGuardado.putInt("posicion", pos);  
    }  
}
```

Al restaurar la música, primero apuntamos a la posición donde se quedó el Player en onRestoreInstanceState y en el siguiente evento, onResume, se activa con mp.Start().

```
@Override protected void onResume() {  
    super.onResume();  
    Toast.makeText( context: this, text: "onResume", Toast.LENGTH_SHORT).show();  
    mp.start();  
}  
  
@Override protected void onRestoreInstanceState(Bundle estadoGuardado) {  
    super.onRestoreInstanceState(estadoGuardado);  
    Toast.makeText( context: this, text: "onRestoreInstanceState", Toast.LENGTH_SHORT).show();  
    if (estadoGuardado != null && mp != null) {  
        int pos = estadoGuardado.getInt( key: "posicion");  
        mp.seekTo(pos);  
    }  
}
```

Paso 4. Verifica el resultado.

Funciona correctamente al pasar de un estado de pantalla al otro.