Multimedia y Ciclo de Vida

Introducción

Una de las funciones más habituales de los modernos teléfonos móviles es su utilización como reproductores multimedia. Su uso más frecuente es como reproductores MP3, para la reproducción de vídeos y televisión a través de Internet. El API de Android viene preparado con excelentes características de reproducción multimedia, permite la reproducción de gran variedad de formatos, tanto de audio como de vídeo. El origen de los datos puede ser tanto un fichero local como un *stream* obtenido desde Internet. Todo este trabajo lo realiza principalmente una clase `MediaPlayer.`

Antes de comenzar la descripción de estos contenidos, comenzaremos el capítulo con un aspecto de vital importancia en el desarrollo de aplicaciones en Android, el ciclo de vida de una aplicación. Es decir, cómo las aplicaciones son creadas, puestas en espera y finalmente destruidas.

Objetivos:

- Comprender el ciclo de vida de una actividad Android.
- Utilizar de forma correcta los diferentes eventos relacionados con el ciclo de vida.
- Aprender cuándo y cómo guardar el estado de una actividad.
- Repasar las facilidades multimedia disponibles en Android, que formatos soporta y las clases que hemos de utilizar.
- Describir la clase MediaPlayer, utilizada para la reproducción de audio y video.
- Dotar a la aplicación Asteroides de un control adecuado de su ciclo de vida y de varios efectos de audio.

Ciclo de vida de una actividad

El ciclo de vida de una aplicación Android es bastante diferente al ciclo de vida de una aplicación en otros S.O., como Windows. La mayor diferencia es que, en Android el ciclo de vida es controlado principalmente por el sistema, en lugar de ser controlado directamente por el usuario.

<Video Ciclo de Vida de una Aplicación Android 1>

http://polimedia.upv.es/visor/?id=6e7d2dc9-c8d7-1144-8db7-33c58545d059

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, conocidos como actividades. Además de varias actividades una aplicación también puede contener servicios. El ciclo de vida de los servicios será estudiado en el capítulo 8. Son las actividades las que realmente controlan el ciclo de vida de las aplicaciones, dado que el usuario no cambia de aplicación, sino de actividad. El sistema va a mantener una pila con las actividades previamente visualizadas, de forma que el usuario va a poder regresar a la actividad anterior pulsando la tecla "atrás".

Una aplicación Android corre dentro de su propio proceso Linux. Este proceso es creado para la aplicación y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación.

Una característica importante, y poco usual, de Android es que la destrucción de un proceso no es controlado directamente por la aplicación. En lugar de esto, es el sistema quien determina cuando destruir el proceso, basándose en el conocimiento que tiene el sistema de las partes de la aplicación que están corriendo (actividades y servicios), qué tan importante son para el usuario y cuánta memoria disponible hay en un determinado momento.

Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso, pero se habrá perdido el estado que tenia esta aplicación. En estos casos, va a ser responsabilidad del programador almacenar el estado de las actividades, si queremos que cuando sea reiniciada conserve su estado.

Como vemos, Android es sensible al ciclo de vida de una actividad, por lo tanto necesitas comprender y manejar los eventos relacionados con el ciclo de vida si quieres crear aplicaciones estables.

Una actividad en Android puede estar en uno de estos cuatro estados:

Activa (*Running*): La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.

Visible (*Paused*): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.

Parada (*Stopped*): Cuando la actividad no es visible, se recomienda guardar el estado de la interfaz de usuario, preferencias, etc.

Destruida (*Destroyed*): Cuando la actividad termina al invocarse el método *finish()*, o es matada por el sistema Android, sale de la pila de actividades.

Cada vez que una actividad cambia de estado se van a producir eventos que podrán ser capturados por ciertos métodos de la actividad. A continuación se muestra un esquema que ilustra los métodos que capturan estos eventos.

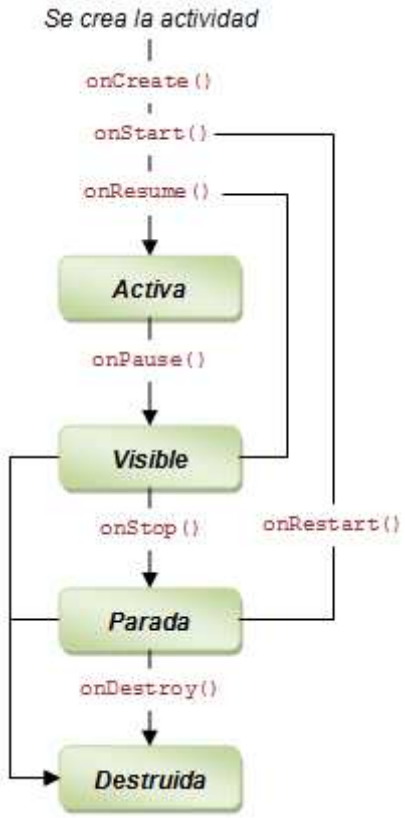


Figura 5: *Ciclo de vida de una actividad.*

onCreate(Bundle): Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de instancia (en una instancia de la clase Bundle), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.

onStart(): Nos indica que la actividad está a punto de ser mostrada al usuario.

onResume(): Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.

onPause(): Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra aplicación es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.

onStop(): La actividad ya no va a ser visible para el usuario. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

onRestart(): Indica que la actividad va a volver a ser representada después de haber pasado por*onStop()*.

onDestroy(): Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón <volver> o cuando se llama al método *finish()*. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

Ejercicio paso a paso: *¿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.

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

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

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

3. Añade los siguientes métodos:

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

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

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

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

@Override protected void onRestart() {
   super.onRestart();
   Toast.makeText(this, "onRestart", Toast.LENGTH_SHORT).show();
```

```
}

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

4. Ejecuta la aplicación y observa la secuencia de `Toast.`

5. Pulsa el botón *Acerca de* y luego regresa a la actividad. Observa la secuencia de `Toast.`

6. Pulsa el botón *Jugar* y luego regresa a la actividad. Observa la secuencia de `Toast.`

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

Ejercicio paso a paso: *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:

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

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.

3. Incluye la siguiente variable en la actividad `Juego`.

```
private VistaJuego vistaJuego;
```

4. Al final de `onCreate` añade:

```
vistaJuego = (VistaJuego) findViewById(R.id.VistaJuego);
```

5. Incorpora los siguientes métodos a la actividad:

```
@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 valido hacerlo en `VistaJuego`.

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

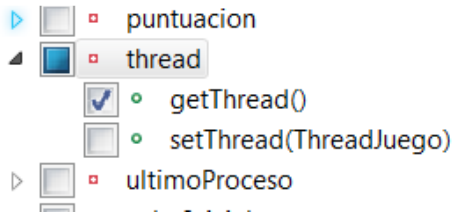
```
class ThreadJuego extends Thread {
   private boolean pausa,corriendo;

   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 `actualiza Fisica()`, 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 {…}`.

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;

}
```

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:


10:51
nes activas
Descarga
RAM
Almacenam
Aplicaciones activas: 3
Salir de todo
WhatsApp
RAM: 21,61MB, CPU: 0,15%
Salir
Asteroides
RAM: 3,31MB, CPU: 0,90%
Salir
Música
RAM: 17,45MB, CPU: 0,00%
Salir

¿Qué proceso se elimina?

Como hemos comentado Android mantiene en memoria todos los procesos que quepan aunque estos no se estén ejecutando. Una vez que la memoria está llena y el usuario decide ejecutar una nueva aplicación, el sistema ha de determinar qué proceso de los que están en ejecución ha de ser eliminado. Android ordena los procesos en una lista jerárquica, asignándole a cada uno una determinada "importancia". Esta lista se confecciona basándose en los componentes de la aplicación que están corriendo (actividades y servicios) y el estado de estos componentes.

Para establecer esta jerarquía de importancia se distinguen los siguientes tipos de procesos:

Proceso de primer plano: (*Foreground process*) Hospeda una actividad en la superficie de la pantalla y con la cual el usuario está interactuando (su método `onResume()` ha sido llamado). Debería haber solo uno o unos pocos procesos de este tipo. Sólo serán eliminados cómo último recurso, si es que la memoria está tan baja que ni siquiera estos procesos pueden continuar corriendo.

Proceso visible: (*Visible process*) Hospeda una actividad que está visible en la pantalla, pero no en el primer plano (su método `onPause()` ha sido llamado). Considerado importante, no será eliminado a menos que sea necesario para mantener los procesos de primer plano.

Proceso de servicio: (*Service process*) Hospeda un servicio que ha sido inicializado con el método `startService()`. Aunque estos procesos no son directamente visibles al usuario, generalmente están haciendo tareas que para el usuario son importantes (tales como reproducir un archivo mp3 o mantener una conexión con un servidor de contenidos). El sistema siempre tratará de mantener esos procesos corriendo, a menos que los niveles de memoria comiencen a comprometer el funcionamiento de los procesos de primer plano o visibles.

Proceso de fondo: (*Background process*) Hospeda una actividad que no es actualmente visible al usuario (su método `onStop()` ha sido llamado). Si estos procesos son eliminados no tendrán un directo impacto en la experiencia del usuario. Generalmente, hay muchos de estos procesos, por lo tanto el sistema para asegurar que el último proceso visto por el usuario sea el último en ser eliminado.

Proceso vacío: (*Empty process*) No hospeda a ningún componente de aplicación activo. La única razón para mantener ese proceso es tener un "caché" que permita mejorar el tiempo de activación en la próxima vez que un componente de su aplicación sea ejecutado.

<Video Ciclo de Vida de una Aplicación Android 2>

http://polimedia.upv.es/visor/?id=baae6ab7-e653-6642-877e-710837e94d60

Práctica: *Aplicando eventos del ciclo de vida en la actividad inicial de Asteroides*

Los conceptos referentes al ciclo de vida de una aplicación vamos 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.

1. Abre el proyecto Asteroides.

2. Busca un fichero de audio (los formatos soportados por Android se listan en este capítulo). 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 uno de la Web del curso.*

3. Abre la actividad `Asteroides` y añade las siguientes líneas en el método `onCreate():`

```
MediaPlayer mp = MediaPlayer.create(this, R.raw.audio);
mp.start();
```

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

5. 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();

mp.start();
```

6. Verifica que funciona correctamente.

Práctica: *Aplicando eventos del ciclo de vida en la actividad inicial de Asteroides (II)*

1. Tras realizar el ejercicio anterior, ejecuta la aplicación y pulsa en el botón Acerca de… La música ha de detenerse.

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.

3. Verifica que cuando abres la actividad *Acerca de…* la música conticua reproduciendose.

4. Pulsa ahora en el botón *Jugar* para abrir la actividad *Juego*. En teoría la música tendría que detenerse dado que la actividad *Asteroides* ya no es visible, por lo que 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.

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.

Práctica: *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.

1. Utilizando el método del ciclo de vida adecuado llama a la siguiente función para detener el uso de sensores:

```
mSensorManager.unregisterListener(this);
```

El parámetro de este método corresponde al objeto `SensorEventListener` del que queremos dejar de recibir eventos. En nuestro caso nosotros mismos.

2. Cuando nuestra actividad vuelva a ejecución queremos que se active la lectura de sensores. Mueve la línea de código que realiza esta función desde `onCreate()` al método adecuado.

Guardando el estado de una actividad

Cuando el usuario ha estado utilizando una actividad, y tras cambiar a otras, regresa a la primera, lo habitual es que esta permanezca en memoria y continúe su ejecución sin alteraciones. Como hemos explicado, en situaciones de escasez de memoria, es posible que el sistema haya eliminado el proceso que ejecutaba la actividad. En este caso, el proceso será ejecutado de nuevo, pero se habrá perdido su estado, es decir, se habrá perdido el valor de sus variables y el puntero de programa. Como consecuencia, si el usuario estaba rellenando un cuadro de texto o estaba reproduciendo un audio en un punto determinado perderá esta información. En este apartado estudiaremos un mecanismo sencillo que nos proporciona Android para resolver este problema.

<Video Guardar estado Actividades>

http://polimedia.upv.es/visor/?id=9793556b-6331-8049-a316-ba7a3f5dd5cb

NOTA: *Cuando se ejecuta una actividad sensible a la inclinación del teléfono, es decir puede verse en horizontal o en vertical, se presenta un problema similar al anterior. La actividad es destruida y vuelta a construir con las nuevas dimensiones de pantalla y por lo tanto se llama de nuevo al método* `onCreate()`. *Antes de que la actividad sea destruida también resulta fundamental guardar su estado.*

Dependiendo de lo sensible que sea la perdida de la información de estado se podrá actuar de diferentes maneras. Una posibilidad es por ejemplo, no hacer nada. Siguiendo con el ejemplo anterior el usuario tendría que volver a rellenar el cuadro de texto o el audio volvería a reproducirse desde el principio.

En caso de tratarse de información extremadamente sensible, se recomienda el uso del método `onPause()` para guardar el estado y onResume() para recuperarlos. Por supuesto, la información sensible ha de almacenarse en un medio permanente como un fichero o una base de datos. Se trata del método más fiable, como vimos en el ciclo de vida de una actividad, los métodos onStop() y onDestroy() no tienen la garantía de ser llamados.

También tenemos una tercera posibilidad que, aunque no tenga una fiabilidad tan grande, resulta mucho más sencilla. Consiste en utilizar los siguientes dos métodos:

> **onSaveInstanceState(Bundle)**: Se invoca para permitir a la actividad guardar su estado. Ojo, si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.
>
> **onRestoreInstanceState(Bundle)**: Se invoca para recuperar el estado guardado poronSaveInstanceState().

Veamos un ejemplo de lo sencillo que resulta guardar la información de una variable tipo cadena de caracteres y entero.

```
String var;
       int pos;
```

```
@Override
protected void onSaveInstanceState(Bundle guardarEstado) {
      super.onSaveInstanceState(guardarEstado);
      guardarEstado.putString("variable", var);
      guardarEstado.putInt("posicion", pos);
}

@Override
protected void onRestoreInstanceState(Bundle recEstado) {
      super.onRestoreInstanceState(recEstado);
      var = recEstado.getString("variable");
      pos = recEstado.getInt("posicion");
}
```

La ventaja de usar estos métodos es que el programador no ha de buscar un método de almacenamiento permanente, es el sistema quien hará este trabajo. El inconveniente es que el sistema no nos garantiza que sean llamados. En casos de extrema necesidad de memoria se podría destruir nuestro proceso sin llamarlos.

Práctica: *Guardando el estado en Asteroides*

1. Ejecuta el proyecto Asteroides.

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

3. Utilizando los métodos para guardar el estado de una actividad, trata de quecuando 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);
```

4. Verifica el resultado.

Solución: Una posible solución al ejercicio anterior se muestra a continuación:

```
@Override
   protected void onSaveInstanceState(Bundle estadoGuardado){
         super.onSaveInstanceState(estadoGuardado);
```

```
        if (mp != null) {
                int pos = mp.getCurrentPosition();
                estadoGuardado.putInt("posicion", pos);
        }
}

@Override
protected void onRestoreInstanceState(Bundle estadoGuardado){
        super.onRestoreInstanceState(estadoGuardado);
        if (estadoGuardado != null && mp != null) {
                int pos = estadoGuardado.getInt("posicion");
                mp.seekTo(pos);
        }
}
```

Para reforzar el uso de estos métodos te recomendamos el ejercicio planteado en el apartado del libro 6.4.1 donde se pide recordar el punto de reproducción de un vídeo.

Utilizando multimedia en Android

La integración de contenido multimedia en nuestras aplicaciones resulta muy sencilla gracias a la gran variedad de facilidades que nos proporciona el API.

Concretamente podemos reproducir audio y vídeo desde orígenes distintos:

- Desde un fichero almacenado en el dispositivo.
- Desde un recurso que está incrustado en el paquete de la aplicación (fichero .apk).
- Desde un *stream* que es leído desde una conexión de red. En este punto admite dos posibles protocolos (http:// y rstp://)

También resulta sencilla la grabación de audio y vídeo, siempre que el *hardware* del dispositivo lo permita.

En la siguiente lista se muestran las clases de Android que nos permitirán acceder a los servicios Multimedia:

MediaPayer: Reproducción de audio/vídeo desde ficheros o *streams*.

MediaControler: Visualiza controles estándar para mediaPlayer (pausa, stop).

VideoView: Vista que permite la reproducción de vídeo.

MediaRecorder: Permite grabar audio y vídeo.

AsyncPlayer: Reproduce lista de audios desde un *thread* secundario.

AudioManager: Gestiona varias propiedades del sistema (volumen, tonos…).

AudioTrack: Reproduce un búfer de audio PCM directamente por *hardware*.

SoundPool: Maneja y reproduce una colección de recursos de audio.

JetPlayer: Reprocuce audio y video interactivo creado con JetCreator.

Camera: Cómo utilizar la cámara para tomar fotos y video.

FaceDetector: Identifica la cara de la gente en un bitmap.

La plataforma Android soporta una gran variedad de formatos, muchos de los cuales pueden ser tanto decodificados como codificados. A continuación, mostramos una tabla con los formatos multimedia soportados. No obstante algunos modelos de móviles pueden soportar formatos adicionales que no se incluyen en la tabla, como por ejemplo DivX.

Cada desarrollador es libre de usar los formatos incluidos en el núcleo del sistema o aquellos que solo se incluyen en algunos dispositivos.

Tipo	Formato	Codifica	Decodifica	Detalles	fichero soportado
Audio	AAC LC/LTP	X	X	Mono/estéreo con cualquier combinación estándar de frecuencia > 160 Kbps y ratios de muestreo de 8 a 48kHz	3GPP (.3gp) MPEG-4(.mp4) No soporta raw AAC (.aac)MPEG-TS (.ts)
	HE-AACv1	a partir 4.1	X		
	HE-AACv2		X		
	AAC ELD	a partir 4.1	a partir 4.1	Mono/estéreo, 16-8kHz	
	AMR-NB	X	X	4.75 a 12.2 Kbps muestreada a @ 8kHz	3GPP (.3gp)
	AMR-WB	X	X	9 ratios de 6.60 Kbps a 23.85 Kbps a @ 16kHz	3GPP (.3gp)
	MP3		X	Mono/estéreo de 8 a 320 Kbps, frecuencia de muestreo constante (CBR) o variable (VBR)	MP3 (.mp3)
	MIDI		X	MIDI tipo 0 y 1. DLS v1 y v2. XMF y XMF móvil. Soporte para tonos de llamada RTTTL / RTX, OTA y iMelody.	Tipo 0 y 1 (.mid, .xmf, .mxmf). RTTTL / RTX (.rtttl, .rtx), OTA (.ota) iMelody (.imy)
	Ogg Vorbis		X		Ogg (.ogg) Matroska (.mkv a partir 4.0)
	FLAC		a partir 3.1	mono/estereo (no multicanal)	FLAC (.flac)
	PCM/WAVE	a partir 4.1	X	8 y 16 bits PCM lineal (frecuencias limitadas por el	WAVE (.wav)

				hardware)	
Imagen	JPEG	X	X	Base + progresivo	JPEG (.jpg)
	GIF		X		GIF (.gif)
	PNG	X	X		PNG (.png)
	BMP		X		BMP (.bmp)
	WEBP	a partir 4.0	a partir 4.0		WebP (.webp)
Video	H.263	X	X		3GPP (.3gp) MPEG-4 (.mp4)
	H.264 AVC	a partir 3.0	X	Baseline Profile (BP)	3GPP (.3gp) MPEG-4 (.mp4)
	MPEG-4 SP		X		3GPP (.3gp)
	WP8		a partir 2.3.3	Streaming a partir 4.0	WebM (.webm) Matroska (.mkv)

Tabla 5: *Formatos multimedia soportados en Android.*

<Video Multimedia en Android>

http://polimedia.upv.es/visor/?id=a84975eb-23a7-1440-9bce-ca61093a2c78

La vista VideoView

La forma más fácil de incluir un vídeo en tu aplicación es incluir una vista de tipo `VideoView.` Veamos cómo hacerlo en el siguiente ejercicio

Ejercicio paso a paso: *Reproducir un vídeo con VideoView*

1. Crea una nueva aplicación.

2. Reemplaza el fichero `ref/layout/activity_main.xml` por

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <VideoView
        android:id="@+id/video"
        android:layout_width="320px"
        android:layout_height="240px"/>
</LinearLayout>
```

3. Reemplaza el siguiente código en la clase `MainActivity:`

```
public class videoView extends Activity {
private VideoView videoView;

@Override public void onCreate(Bundle savedInstanceState){
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       videoView =(VideoView)findViewById(R.id.video);
       //de forma alternative si queremos un streaming usar
       //videoView.setVideoURI(Uri.parse(URLstring));
       videoView.setVideoPath("/mnt/sdcard/video.mp4");
       videoView.start();
       videoView.requestFocus();
}
}
```

4. En el método `setVideoPath()` estamos indicando un fichero local.

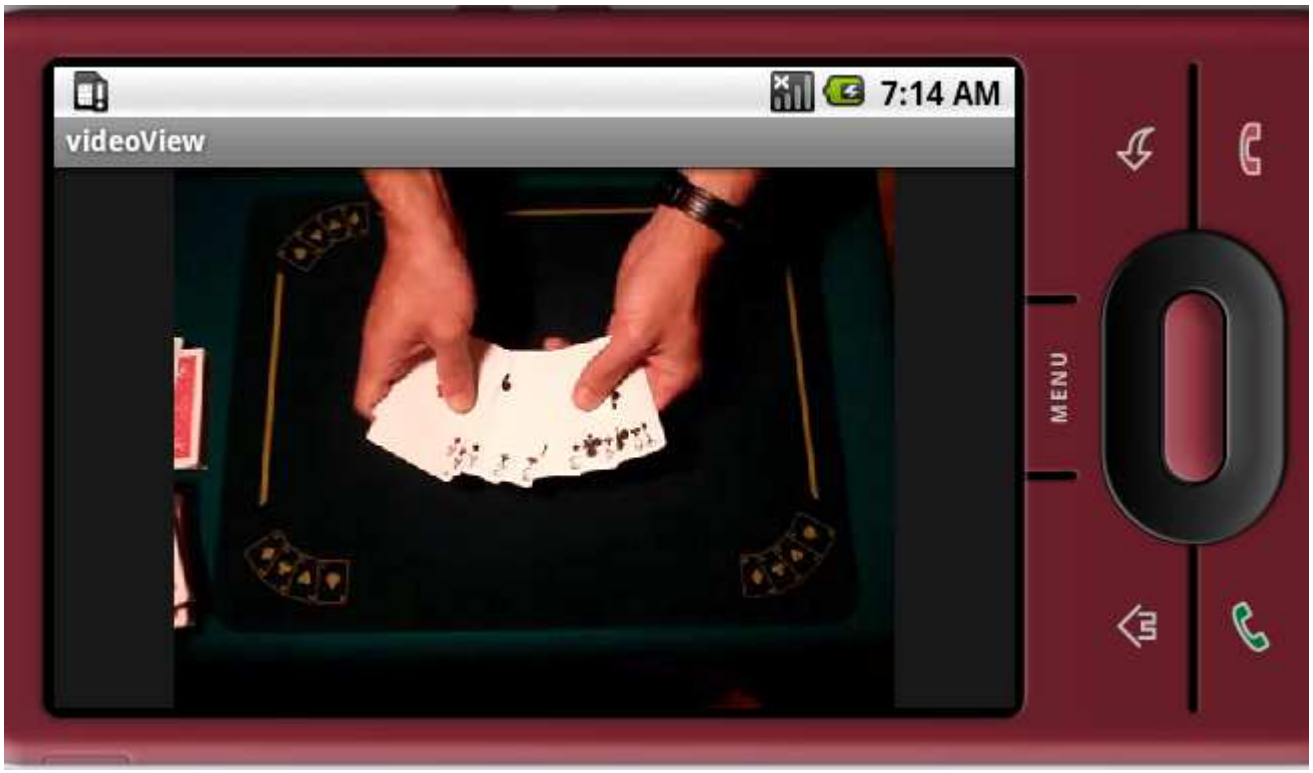
5. Busca un fichero de video en codificación mp4. Renombra este fichero a *video.mp4*.

6. Es necesario almacenar un fichero de video en el simulador. Para ello, utiliza la opción del menú *Window* > *Show View* >*Others…* > *Android* > *File Explorer*. Te mostrará el sistema de ficheros del emulador.

7. Selecciona la carpeta *mnt/sdcard* y utiliza el botón donde aparece un teléfono con una flecha para almacenar un nuevo fichero. Indica el fichero *video.mp4*.

NOTA: el dispositivo ha de disponer de almacenamieno externo.

8. Ejecuta la aplicación y observa el resultado. Recuerda que con *Ctrl-F11* puedes



cambiar la orientación. Si no funciona trata de usar otro tipo de emulador.

9. Modifica el fichero xml para que el vídeo aparezca centrado y ocupe toda la pantalla del teléfono, tal y como se muestra en la imagen de arriba

10. Añade la siguiente línea antes de la llamada al método `start();`

```
videoView.setMediaController(new MediaController(this));
```

11 . De esta forma permitimos que el usuario pueda controlar la reproducción del vídeo mediante el objeto `MediaController.`

12. Observa el resultado.

La clase MediaPlayer

La reproducción multimedia en Android se lleva a cabo principalmente por medio de la clase`MediaPlayer`. Veremos a continuación las características más importantes de esta clase y cómo podemos sacarle provecho.

Un objeto `MediaPlayer` va a poder pasar por gran variedad de estados: inicializados sus recursos (*initialized*), preparando la reproducción (*preparing*), preparado para reproducir (*prepared*), reproduciendo (*started*), en pausa (*paused*), parado (*stoped*), reproducción completada (*playback completed*), finalizado (*end*) y con error (*error*). Resulta importante conocer en qué estado se encuentra dado que muchos de los métodos solo pueden ser llamados desde ciertos estados. Por ejemplo, no podemos ponerlo en reproducción (método `start()`) sino está en estado preparado. O no podremos ponerlo en pausa (`pause()`), si está parado. Si llamamos a un método no admitido para un determinado estado se producirá un error de ejecución.

La siguiente tabla permite conocer los métodos que podemos invocar desde cada uno de los estados y cuál es el nuevo estado al que iremos tras invocarlo. Existen dos tipos de métodos, los que no están subrayados representan métodos llamados de forma síncrona desde nuestro código, mientras que los que están subrayados representan métodos llamados de forma asíncrona por el sistema.

Estado salida	Estado entrada							
	Idle	**Initialized**	**Preparing**	**Prepared**	**Started**	**Paused**	**Stoped**	**Playback Completed**
Initialized	setDataSource							
Preparing		prepareAsinc					prepareAsinc	
Prepared		prepare	onPrepared	seekTo			prepare	
Started				start	seekTo start	start		start
Paused					pause	seekTo pause		
Stoped				stop	stop	stop	stop	stop
Playback Completed					onCompletion			seekTo
End	Release	release	release	release	release	release	release	release
Error	onError	onError	onError	onError	onError	onError	onError	onError

Tabla 6: *Transiciones entre estados de la clase MediaPlayer*

Reproducción de audio con MediaPlayer

Si el audio o vídeo se va a reproducir siempre en nuestra aplicación resulta interesante incluirlo en el paquete `.apk` y tratarlo como un recurso. Este uso ya ha sido ilustrado al comienzo del capítulo. Recordemos como se hacía:

1. Crea una nueva carpeta con nombre raw dentro de la carpeta res.
2. Arrastra a su interior el fichero de audio. Por ejemplo audio.mp3.
3. Añade las siguientes líneas de código:

```
MediaPlayer mp = MediaPlayer.create(this, R.raw.audio);
    mp.start();
```

Si deseas parar la reproducción tendrás que utilizar el método `stop()` . Si a continuación quieres volver a reproducirlo utiliza el método `prepare()` y a continuación `start()`. También puedes usar `pause()` y`start()` para ponerlo en pausa y reanudarlo.

Si en lugar de un recurso prefieres reproducirlo desde un fichero utiliza el siguiente código. Observa como en este caso es necesario llamar al método `prepare()`. En el caso anterior no ha sido necesario dado que esta llamada se hace desde `create()`.

```
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(RUTA_AL_FICHERO);
mp.prepare();
mp.start();
```

Un reproductor multimedia paso a paso

En el siguiente ejercicio vamos a profundizar en el objeto *MediaPlayer* por medio de un ejemplo, donde trataremos de realizar un reproductor de vídeos personalizado.

Ejercicio paso a paso: *Un reproductor multimedia pasó a paso*

1. Crea una nueva aplicación con los siguientes datos:

```
Project name: VideoPlayer
Application name: VideoPlayer
Package name: org.example.videoplayer
Create Activity: VideoPlayer
Min SDK Version: 3
```

2. En la carpeta `res/drawable` arrastra los cuatro ficheros de iconos: play.png, pause.png, stop.png*y* log.png que se muestran a continuación. (Botón derecho / guardar como para descargarlos)

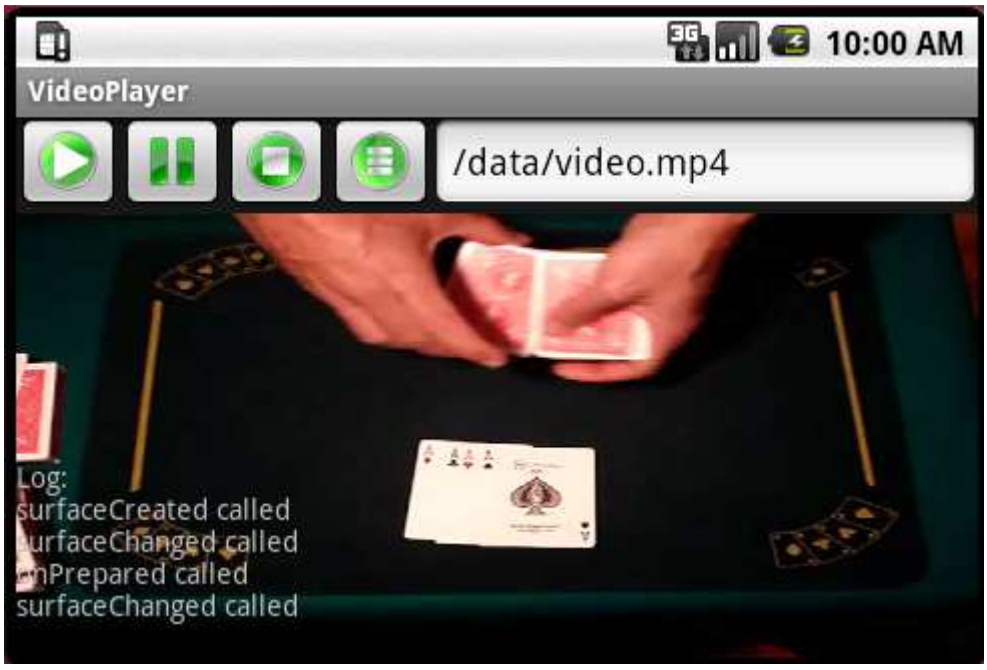
 play.png	 pause.png	 stop.png	 log.png

3. Reemplaza el fichero el fichero *res/layout/main.xml* por:

```
<RelativeLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_height="fill_parent"
       android:layout_width="fill_parent">
    <LinearLayout
           android:id="@+id/ButonsLayout"
           android:layout_height="wrap_content"
           android:layout_width="fill_parent"
           android:orientation="horizontal"
           android:layout_alignParentTop="true">
        <ImageButton android:id="@+id/play"
                     android:layout_height="wrap_content"
                     android:layout_width="wrap_content"
                     android:src="@drawable/play"/>
        <ImageButton android:id="@+id/pause"
                     android:layout_height="wrap_content"
                     android:layout_width="wrap_content"
                     android:src="@drawable/pause"/>
        <ImageButton android:id="@+id/stop"
                     android:layout_height="wrap_content"
                     android:layout_width="wrap_content"
                     android:src="@drawable/stop"/>
        <ImageButton android:id="@+id/logButton"
                     android:layout_height="wrap_content"
                     android:layout_width="wrap_content"
                     android:src="@drawable/log"/>
        <EditText    android:id="@+id/path"
```

```
                    android:layout_height="fill_parent"
                    android:layout_width="fill_parent"
                    android:text="/data/video.3gp"/>
    </LinearLayout>
    <VideoView android:id="@+id/surfaceView"
               android:layout_height="fill_parent"
               android:layout_width="fill_parent"
               android:layout_below="@+id/ButonsLayout"/>
    <ScrollView android:id="@+id/ScrollView"
               android:layout_height="100px"
               android:layout_width="fill_parent"
               android:layout_alignParentBottom="true">
        <TextView    android:id="@+id/Log"
                    android:layout_height="wrap_content"
                    android:layout_width="fill_parent"
                    android:text="Log:"/>
    </ScrollView>
</RelativeLayout>
```

La apariencia del *Layout* anterior se muestra a continuación:



4. Reemplaza el código de la clase `VideoPlayer` por:

```
public class VideoPlayer extends Activity implements
                    OnBufferingUpdateListener, OnCompletionListener,
                    MediaPlayer.OnPreparedListener, SurfaceHolder.Callback {
        private MediaPlayer mediaPlayer;
        private SurfaceView surfaceView;
        private SurfaceHolder surfaceHolder;
        private EditText editText;
        private ImageButton bPlay, bPause, bStop, bLog;
        private TextView logTextView;
        private boolean pause;
```

```
	private String path;
	private int savePos = 0;

	public void onCreate(Bundle bundle) {
		super.onCreate(bundle);
		setContentView(R.layout.main);
		surfaceView = (SurfaceView) findViewById(R.id.surfaceView);
		surfaceHolder = surfaceView.getHolder();
		surfaceHolder.addCallback(this);
		surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
		editText = (EditText) findViewById(R.id.path);
		editText.setText(
				"http://personales.gan.upv.es/~jtomas/video.3gp");
		logTextView = (TextView) findViewById(R.id.Log);
		bPlay = (ImageButton) findViewById(R.id.play);
		bPlay.setOnClickListener(new OnClickListener() {
			public void onClick(View view) {
				if (mediaPlayer != null) {
					if (pause) {
						mediaPlayer.start();
					} else {
						playVideo();
					}
				}
			}
		});
		bPause = (ImageButton) findViewById(R.id.pause);
		bPause.setOnClickListener(new OnClickListener() {
			public void onClick(View view) {
				if (mediaPlayer != null) {
					pause = true;
					mediaPlayer.pause();
				}
			}
		});
		bStop = (ImageButton) findViewById(R.id.stop);
		bStop.setOnClickListener(new OnClickListener() {
			public void onClick(View view) {
				if (mediaPlayer != null) {
					pause = false;
					mediaPlayer.stop();
				}
			}
		});
		bLog = (ImageButton) findViewById(R.id.logButton);
		bLog.setOnClickListener(new OnClickListener() {
			public void onClick(View view) {
				if (logTextView.getVisibility()==TextView.VISIBLE)
{
					logTextView.setVisibility(TextView.INVISIBLE)
;
				} else {
					logTextView.setVisibility(TextView.VISIBLE);
				}
			}
		});
		log("");
```

```
}
```

5. Como puedes ver la aplicación extiende la clase `Activity.` Además implementamos cuatro interfaces que corresponden a varios escuchadores de eventos. Luego continúa la declaración de variables. Las primeras corresponden a diferentes elementos de la aplicación y su significado resulta obvio. La variable `pause` nos indica si el usuario ha pulsado el botón correspondiente, la variable `path` nos indica dónde está el vídeo en reproducción y la variable `savePos` almacena la posición de reproducción.

6. Añade:

```
private void playVideo() {
            try {
                  pause = false;
                  path = editText.getText().toString();
                  mediaPlayer = new MediaPlayer();
                  mediaPlayer.setDataSource(path);
                  mediaPlayer.setDisplay(surfaceHolder);
                  mediaPlayer.prepare();
                  // mMediaPlayer.prepareAsync(); Para streaming
                  mediaPlayer.setOnBufferingUpdateListener(this);
                  mediaPlayer.setOnCompletionListener(this);
                  mediaPlayer.setOnPreparedListener(this);
                  mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
                  mediaPlayer.seekTo(savePos);
            } catch (Exception e) {
                  log("ERROR: " + e.getMessage());
            }
      }
```

7. El código continúa con la definición del método `playVideo().` Este método se encarga de obtener la ruta de reproducción, crear un nuevo objeto `MediaPlayer,` luego se le asigna la ruta y la superficie de visualización, a continuación se prepara la reproducción del vídeo. En caso de querer reproducir un *stream* desde la red, esta función puede tardar bastante tiempo, en tal caso es recomendable utilizar en su lugar el método `prepareAsync()` que permite continuar con la ejecución del programa, aunque sin esperar a que el vídeo esté preparado. Las siguientes tres líneas asignan a nuestro objeto varios escuchadores de eventos que serán descritos más adelante. Tras preparar el tipo de audio, se sitúa la posición de reproducción a los milisegundos indicados en la variable `savePos`. Si se trata de una nueva reproducción, esta variable será cero.

8. Añade el código:

```
public void onBufferingUpdate(MediaPlayer arg0, int percent) {
            log("onBufferingUpdate percent:" + percent);
      }

      public void onCompletion(MediaPlayer arg0) {
            log("onCompletion called");
      }
```

9. Los métodos anteriores implementan los interfaces `OnBufferingUpdateListener` y `OnCompletionListener.` El primero irá mostrando el porcentaje de obtención de

búfer de reproducción, mientras que el segundo será invocado cuando el vídeo en reproducción llegue al final.

10. Añade el código:

```
 public void onPrepared(MediaPlayer mediaplayer) {
             log("onPrepared called");
             int mVideoWidth = mediaPlayer.getVideoWidth();
             int mVideoHeight = mediaPlayer.getVideoHeight();
             if (mVideoWidth != 0 && mVideoHeight != 0) {
                    surfaceHolder.setFixedSize(mVideoWidth, mVideoHeight)
;
                    mediaPlayer.start();
             }
       }
```

11. El método anterior implementa la interfaz `OnPreperedListener.` Es invocado una vez que el vídeo ya está preparado para su reproducción. En este momento podemos conocer el alto y el ancho del vídeo y ponerlo en reproducción.

12. Añade el código:

```
 public void surfaceCreated(SurfaceHolder holder) {
             log("surfaceCreated called");
             playVideo();
       }

       public void surfaceChanged(SurfaceHolder surfaceholder,

                    int i, int j, int k) {
             log("surfaceChanged called");
       }

       public void surfaceDestroyed(SurfaceHolder surfaceholder) {
             log("surfaceDestroyed called");
       }
```

13. Los métodos anteriores implementan la interfaz `SurfaceHolder.Callback.` Se invocarán cuando nuestra superficie de visualización se cree, cambie o se destruya. Los métodos que siguen corresponden a acciones del ciclo de vida de una actividad:

14. Añade el código:

```
       @Override protected void onDestroy() {
              super.onDestroy();
              if (mediaPlayer != null) {
                     mediaPlayer.release();
                     mediaPlayer = null;
              }
```

```
}
```

15. Este método se invoca cuando la actividad va a ser destruida. Dado que un objeto de la clase `MediaPlayer` consume muchos recursos, resulta interesante liberarlos lo antes posible.

16. Añade el código:

```
@Override public void onPause() {
        super.onPause();
        if (mediaPlayer != null & !pause) {
                mediaPlayer.pause();
        }
}

@Override public void onResume() {
        super.onResume();
        if (mediaPlayer != null & !pause) {
                mediaPlayer.start();
        }
}
```

17. Los dos métodos anteriores se invocan cuando la actividad pasa a un segundo plano y cuando vuelve a primer plano. Dado que queremos que el vídeo deje de reproducirse y continúe reproduciéndose en cada uno de estos casos, se invocan a los métodos `pause()` y `start(),` respectivamente. No hay que confundir esta acción con la variable `pause` que lo que indica es que el usuario ha pulsado el botón correspondiente.

18. Añade el código:

```
@Override
protected void onSaveInstanceState(Bundle guardarEstado) {
        super.onSaveInstanceState(guardarEstado);
        if (mediaPlayer != null) {
                int pos = mediaPlayer.getCurrentPosition();
                guardarEstado.putString("ruta", path);
                guardarEstado.putInt("posicion", pos);
        }
}

@Override
protected void onRestoreInstanceState(Bundle recEstado) {
        super.onRestoreInstanceState(recEstado);
        if (recEstado != null) {
                path = recEstado.getString("ruta");
                savePos = recEstado.getInt("posicion");
        }
}
```

19. Cuando este sistema necesita memoria, puede decidir eliminar nuestra actividad. Antes de hacerlo llamará al método `onSaveInstanceState` para darnos la

oportunidad de guardar información sensible. Si más adelante el usuario vuelve a la aplicación, esta se volverá a cargar, invocándose el método onRestoreInstanceState, donde podremos restaurar el estado perdido. En nuestro caso la información a guardar son las variables `path` y savePos, que representan el vídeo y la posición que estamos reproduciendo.

20. Ocurre el mismo proceso cuando el usuario cambia la posición del teléfono. Es decir, cuando el teléfono se voltea las actividades son destruidas y vueltas a crear, por lo que también se invocan estos métodos.

21. Añade el código:

```
    private void log(String s) {
          logTextView.append(s + "\n");
    }
}
```

22. El último método es utilizado por varios escuchadores de eventos para mostrar información sobre lo que está pasando. Esta información puede visualizarse o no, utilizando el botón correspondiente.

Introduciendo efectos de audio con SoundPool

Hemos aprendido a utilizar la clase `MediaPlayer` para reproducir audio y video. En un primer ejercicio vamos a aprender a utilizarla para introducir efectos de audio en Asteroides. Como veremos esta clase no resulta adecuada para este uso. A continuación se introducirá la clase `SoundPool` y se mostrará mediante un ejercicio como esta sí que resulta eficiente para nuestro juego.

Ejercicio paso a paso: *Introduciendo efectos de audio con MediaPlayer*

1. Abre el proyecto Asteroides.

2. Copia los siguientes ficheros a la carpeta *res/raw*.

disparo.mp3

explosion.mp3

3. Abre la clase `VistaJuego` y añade las siguientes variables:

```
MediaPlayer mpDisparo, mpExplosion;
```

4. En el constructor de la clase inicialízalas de la siguiente forma:.

```
    mpDisparo = MediaPlayer.create(context, R.raw.disparo);
mpExplosion = MediaPlayer.create(context, R.raw.explosion);
```

5. Añade en el método `ActivaMisil()` de `VistaJuego:`

```
mpDisparo.start();
```

6. Añade en el método `destruyeAsteroide()` de `VistaJuego:`

```
mpExplosion.start();
```

7. Ejecuta el programa y verifica el resultado. ¿Qué pasa cuando disparamos o destruimos un asteroide? ¿El sonido se oye de forma inmediata?

La clase `SoundPool` maneja y reproduce de forma rápida recursos de audio en las aplicaciones.

Un `SoundPool` es una colección de pistas de audio que se pueden cargar en la memoria desde un recurso (dentro de la APK) o desde el sistema de archivos. `SoundPool` utiliza el servicio de la clase `MediaPlayer` para descodificar el audio en un formato crudo (PCM de16 bits), lo que después permite reproducirlo rápidamente por el hardware sin tener que decodificarlas cada vez.

Los sonidos pueden repetirse en un bucle una cantidad establecida de veces, definiendo el valor al reproducirlo, o dejarse reproduciendo en un bucle infinito con -1. En este caso, será necesario detenerlo con el método `stop()`.

La velocidad de reproducción también se puede cambiar. El rango de velocidad de reproducción va de 0.5 a 2.0. Una tasa de reproducción de 1.0 hace que el sonido se reproduzca en su frecuencia original. Una tasa de reproducción de 2.0 hace que el sonido se reproduzca al doble de su frecuencia original, y una tasa de reproducción de 0.5 hace que se reproduzca a la mitad de su frecuencia original.

Cuando se crea un `SoundPool` hay que establecer en un parámetro el máximo de pistas que se pueden reproducir simultáneamente. Este parámetro no tiene por qué coincidir con el número de pistas cargadas. Cuando se pone una pista en reproducción (método `play()`) hay que indicar una prioridad. Esta prioridad se utiliza para decidir que se hará cuando el número de reproducciones activas exceda el valor máximo establecido en el constructor. En este caso, se detendrá el flujo con la prioridad mas baja, y en caso de que haya varios, se detendrá el más antiguo. En caso de que el nuevo flujo sea el de menor prioridad, este no se reproducirá.

Una lisita de todos los métodos de esta clase la puedes encontrar en el siguiente link:

http://developer.android.com/reference/android/media/SoundPool.html

Ejercicio paso a paso: *Introduciendo efectos de audio con SoundPool*

1. El proyecto Asteroides elimina todo el código introducido a partir del punto 3, del ejercicio anterior.

2. Abre la clase `VistaJuego` y añade las siguientes variables:

```
// //// MULTIMEDIA //////
SoundPool soundPool;
int idDisparo, idExplosion;
```

3. En el constructor de la clase inicialízalas de la siguiente forma:

```
soundPool = new SoundPool( 5, AudioManager.STREAM_MUSIC , 0);
idDisparo = soundPool.load(context, R.raw.disparo, 0);
idExplosion = soundPool.load(context, R.raw.explosion, 0);
```

En el constructor de `SoundPool` hay que indicar tres parámetros. El primero corresponde al máximo de reproducciones simultáneas. El segundo es el tipo de stream de audio (normalmente `STREAM_MUSIC`). El tercero es la calidad de reproducción, aunque actualmente no se implementa.

Cada una de las pistas ha de ser cargada mediante el método `load()`. Existen muchas sobrecargas para este método. La versión utilizada en el ejemplo permite cargar las pistas desde los recursos. El último parámetro corresponde a la prioridad, aunque actualmente no tiene ninguna utilidad.

4. Añade en el método `ActivaMisil()` de `VistaJuego:`

```
soundPool.play(idDisparo, 1, 1, 1, 0, 1);
```

El método `play()` permite reproducir una pista. Hay que indicarle el identificador de pista; el volumen para el canal izquierdo y derecho (0.0 a 1.0); La prioridad; El número de repeticiones (-1= siempre, 0=solo una vez, 1=repetir una vez, …) y el ratio de reproducción, con el que podremos modificar la velocidad o pitch (1.0 reproducción normal, rango: 0.5 a 2.0)

5. Añade en el método `destruyeAsteroide()` de `VistaJuego:`

```
soundPool.play(idExplosion, 1, 1, 0, 0, 1);
```

6. Los parámetros utilizados para la explosión son similares, solo hemos introducido una prioridad menor. La consecuencia será que si ya hay un total de 5 (ver constructor) pistas reproduciéndose y se pide la reproducción de un nuevo disparo. El sistema detendrá la reproducción de la explosión más antigua, por tener esta menos prioridad.

Ejecuta el programa y verifica el resultado. ¿Qué pasa cuando disparamos o destruimos un asteroide? ¿El sonido se oye de forma inmediata?

7. Modifica algunos de los parámetros del método `play()` y verifica el resultado.

Grabación de audio

Las APIs de Android disponen de facilidades para capturar audio y video, permitiendo su codificación en diferentes formatos. La clase MediaRecorder te permitirá de forma sencilla integrar esta funcionalidad en tu aplicación.

La mayoría de dispositivos disponen de micrófono para capturar audio, sin embargo esta facilidad no ha sido integrada en el emulador. Por lo tanto, has de probar los ejercicios de este apartado en un dispositivo real.

La clase `MediaRecorder` dispone de una serie de métodos que podrás utilizar para configurar la grabación:

`setAudioSource(int audio_source)` – Dispositivo que se utilizará como fuente del sonido. Normalmente `MediaRecorder.AudioSource.MIC`. También se pueden utilizar otras constantes como `DEFULT`, `CAMCORDER`, `VOICE_CALL`, `VOICE_COMUNICATION`, …

`setOutputFile (String fichero)` – Nombre del fichero de salida.

`setOutputFormat(int output_forma)` – Formato del fichero de salida. Se pueden utilizar constantes de la clase `MediaRecorder.OutputFormat`: `DEFAULT`, `AMR_NB`, `AMR_WB`, `RAW_AMR` (ARM), `MPEG_4`(MP4) y `THREE_GPP` (3GPP).

`setAudioEncoder(int audio_encoder)` – Codificación del audio. Cuatro posibles constantes de la clase `MediaRecorder.AudioEncoder`: `AAC`, `AMR_NB`, `AMR_WB` y `DEFAULT`.

`setAudioChannels(int numeroCanales)` – Especificamos el número de canales 1: mono y 2: estéreo.

`setAudioEncodingBitRate (int bitRate)` (desde nivel de API 8) – Especificamos los bits por segundo (bps) a utilizar en la codificación.

`setAudioSamplingRate (int samplingRate)` (desde nivel de API 8) – Especificamos el número de muetras por segundo a utilizar en la codificación.

`setMaxDuration (int max_duration_ms)` (desde nivel de API 3) – Indicamos una duración máxima para la grabación. Tras este tiempo se detendrá.

`setMaxFileSize (long max_filesize_bytes)` (desde nivel de API 3) – Indicamos un tamaño máximo para el fichero. Al alcanzar el tamaño se detendrá.

`prepare()` – Prepara la grabación para la captura de datos. Antes de llamarlo hay que configurar la grabación y después ya podemos invocar al método `start()`.

`start()` – Comienza la captura

`stop()` – Finaliza la captura

`reset()` – Reinicia el objeto como si lo acabáramos de crear. Hay que volver a configurarlo.

`release()` – Libera todos los recursos utilizados de forma inmediata. Si no llamas al método se liberarán cuando el objeto sea destruido.

La clase `MediaRecorder` también dispone de métodos que podrás utilizar para configurar la grabación de video. Más información en: http://developer.android.com/reference/android/media/MediaRecorder.html

La siguiente tabla muestra los diferentes métodos que pueden ser ejecutados en cada estado y el estado que alcanzaremos tras la llamada:

Estado salida	**Estado entrada**						
	Initial	**Initialized**	**DataSource Configured**	**Prepared**	**Recording**	**Error**	
Initial		reset	reset	reset	reset stop	reset	
Initialized	setAudioSource setVideoSource	setAudioSource setVideoSource					
DataSource Configured		setOutputFormat	setAudioEncoder setVideoEncoder setOutputFile setVideoSize setVideoFrameRate setPreviewDisplay				
Prepared			prepare				
Recording				start			
Released	releas						
Error	llamada incorrecta	llamada incorrecta	llamada incorrecta	llamada incorrecta	llamada incorrecta		

Tabla 7: *Transiciones entre estados de la clase MediaRecorder*

Ejercicio paso a paso: *Grabación de audio utilizando MediaRecoder*

1. Crea un nuevo proyecto con nombre *Grabadora*. El nombre del paquete ha de ser `org.example.grabadora.`

2. Reemplaza el Layout main.xml por el siguiente código:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <Button
        android:id="@+id/bGrabar"
        android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="Grabar"
            android:onClick="grabar"/>
        <Button
            android:id="@+id/bDetener"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Detener Grabacion"
            android:onClick="detenerGrabacion"/>
        <Button
            android:id="@+id/bReproducir"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Reproducir"
            android:onClick="reproducir"/>
    </LinearLayout>
```

3. Reemplaza el código de la actividad por:

```
public class GrabadoraActivity extends Activity {
    private static final String LOG_TAG = "Grabadora";
    private MediaRecorder mediaRecorder;
        private MediaPlayer mediaPlayer;
        private static String fichero =
Environment.                getExternalStorageDirectory().getAbsolutePat
h()+"/audio.3gp";

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

        public void grabar(View view) {
            mediaRecorder = new MediaRecorder();
            mediaRecorder.setOutputFile(fichero);
             mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC)
;
             mediaRecorder.setOutputFormat(MediaRecorder.

             OutputFormat.THREE_GPP);
             mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.

                                           AMR_NB);
            try {
                mediaRecorder.prepare();
            } catch (IOException e) {
                Log.e(LOG_TAG, "Fallo en grabación");
            }
            mediaRecorder.start();
        }

        public void detenerGrabacion(View view) {
            mediaRecorder.stop();
            mediaRecorder.release();
        }
```

```
    public void reproducir(View view) {
          mediaPlayer = new MediaPlayer();
          try {
                mediaPlayer.setDataSource(fichero);
                mediaPlayer.prepare();
                mediaPlayer.start();
          } catch (IOException e) {
                Log.e(LOG_TAG, "Fallo en reproducción");
          }
    }
}
```

Comenzamos declarando dos objetos de las clase `MediaRecorder` y `MediaPlayer` que serán usados para la grabación y reproducción respectivamente. También se declaran dos `String:` La contante `LOG_TAG` será utilizada como etiqueta para identificar nuestra aplicación en el `fichero` de Log. La variable ficheroidentifica en nombre del fichero donde se guardará la grabación. Este fichero se almacenará en una carpeta especialmente creada para nuestra aplicación. NOTA: En la unidad 8 se introducirá la gestión de ficheros. En el método `onCreate()` no se realiza ninguna acción especial.

A continuación se han introducido tres métodos que serán ejecutados al pulsar los botones de nuestro Layout. El significado de cada uno de los métodos que se invocan acaba de ser explicado.

4. Abre AndroidManifest.xml y en la pestaña *Permissions* pulsa el botón *Add…* selecciona *Uses Permissions*. En el desplegable *Name* indica `RECORD_AUDIO`. Repite este proceso para usar el permiso `WRITE_EXTERNAL_STORAGE.`

5. Ejecuta de nuevo la aplicación y verifica el resultado.

Ejercicio paso a paso: *Grabación de audio utilizando MediaRecoder (II)*

La aplicación anterior resulta algo confusa de utilizar. Sería más sencillo si los botones que no pudiéramos utilizar en un determinado momento estuvieran deshabilitados. Para conseguirlo vamos a utilizar el método `Button.setEnabled(boolean)`. Veamos como conseguirlo:

1. Declara las siguientes tres variables al principio de la Actividad:

```
private Button bGrabar, bDetener, bReproducir;
```

2. En el método onCreate() inicializa estos tres botones. Además comenzaremos deshabilitando dos de los botones que al principio de la aplicación no deben ser pulsados:

```
bGrabar = (Button) findViewById(R.id.bGrabar);
bDetener = (Button) findViewById(R.id.bDetener);
bReproducir = (Button) findViewById(R.id.bReproducir);
bDetener.setEnabled(false);
bReproducir.setEnabled(false);
```

3. En el método `grabar()` añade el siguiente código:

```
bGrabar.setEnabled(false);
bDetener.setEnabled(true);
bReproducir.setEnabled(false);
```

4. En el método `detenerGrabacion()` añade el siguiente código:

```
bGrabar.setEnabled(true);
bDetener.setEnabled(false);
bReproducir.setEnabled(true);
```

5. Ejecuta de nuevo la aplicación y verifica el resultado.