UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

INGENIERO TÉCNICO DE TELECOMUNICACIONES: SISTEMAS ELECTRÓNICOS

"DESARROLLO DE UNA APP INFANTIL EDUCATIVA SOBRE EL SISTEMA OPERATIVO ANDROID"

PROYECTO FIN DE CARRERA

Junio – 2014

AUTOR: David Marhuenda Segura

DIRECTOR/ES: César Fernández Peris

VISTO BUENO Y CALIFICACIÓN DEL PROYECTO

Título proyecto:

Proyectante: _____

Director/es: _____

_____

VºBº director/es del proyecto: Fdo.: Fdo.:

Lugar y fecha: _____

CALIFICACIÓN NUMÉRICA ☐ MATRÍCULA DE HONOR ☐

Méritos justificativos en el caso de conceder Matrícula de Honor:

_____

_____

_____

_____

Conforme presidente: Fdo.:	Conforme secretario: Fdo.:	Conforme vocal: Fdo.:

Lugar y fecha: _____

RESUMEN

Mediante este documento se pretende mostrar una visión general de cómo abordar el desarrollo de un proyecto para el sistema operativo Android de Google. Para ello se analizarán las características técnicas y de desarrollo de este sistema operativo y analizaremos las ventajas e inconvenientes respecto a otros sistemas operativos disponibles. Además se incluirá una aplicación a modo de caso práctico donde se abarcarán tanto las fases de análisis, desarrollo e implementación.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1
INTRODUCCIÓN Y OBJETIVOS

1.1 INTRODUCCIÓN

El proyecto desarrollado nace a raíz de una inquietud y motivación personal, abarcar el diseño y desarrollo de aplicaciones para el sistema operativo Android.

Bajo este pretexto se ha desarrollado una aplicación educativa basada en mini-juegos interactivos (abarcando letras, números, formas geométricas y colores) dirigida a un público infantil.

Por lo tanto la aplicación tiene un claro objetivo pedagógico a la vez que ameno y entretenido.

1.2 OBJETIVOS

Este proyecto consta de dos objetivos principales. El primero de ellos consiste en una inquietud personal en la inmersión del desarrollo de aplicaciones para Android tanto a nivel de análisis, diseño, desarrollo como de utilización de recursos y herramientas de trabajo para este fin.

El segundo objetivo trata de transformar lo antes mencionado a una aplicación real. Habiendo hecho un estudio de mercado previo y viendo que lo que ya existía no cumplía mis expectativas personales, “Proyecto Educa” es una ampliación y mejora de lo ya existente.

Como consecuencia de lo expuesto, podemos resumir los objetivos en los siguientes puntos:

- Ofrecer una visión general de los distintos dispositivos móviles actuales disponibles, analizando las posibilidades que nos ofrecen cada uno de ellos.
- Conocer Android, su arquitectura y sus características.
- Adentrarse en el desarrollo para Android pasando por sus 3 fases: Análisis, diseño y desarrollo.

1.3 FASES DE DESARROLLO

En la primera fase de desarrollo se llevó a cabo un estudio acerca de la plataforma Android, estudiando la plataforma de desarrollo y los diferentes lenguajes de programación para poder desarrollar. Una vez finalizado esto, se pusieron en práctica los conocimientos adquiridos en forma de diferentes ejemplos y pequeños ejercicios para ir familiarizándome con el entorno de trabajo.

En la segunda fase se analizó que aplicación se podría realizar. Después de investigar y descubrir carencias muy comunes en todas las aplicaciones estudiadas, se dispuso a desarrollar una aplicación que supliera y mejorara todas esas carencias.

En la tercera fase se diseñó cómo iba a ser y cómo iba a funcionar la aplicación. Al ser el público final un público infantil, se tuvo extremo cuidado en realizar una interfaz de usuario fácil e intuitiva para que mediante tacto y audio el usuario se pudiese desenvolverse sin problemas.

La cuarta y última fase consistió en el desarrollo y testeo del producto final, realizando numerosas pruebas y correcciones.

1.4 EQUIPOS Y MEDIOS DE DESARROLLO

Para la elaboración de este proyecto se ha empleado un equipo de desarrollo con las siguientes características:

- Sistema Operativo: Windows 7 Home Premium.
- Librerías de desarrollo: Android SDK Manager R22.3, Java SDK7.
- Herramientas de desarrollo: IDE Eclipse 4.2.3 con el componente ADT instalado.

Para las pruebas durante la fase de desarrollo se ha utilizado un dispositivo Samsung Galaxy Note II.

Para la edición de imágenes y audio se ha utilizado el software libre *Paint.NET* y *Audacity* respectivamente.

Y finalmente para la creación de la documentación se ha utilizado la suite ofimática *Microsoft Word 2010*.

1.5 ESTRUCTURA DE LA MEMORIA

En este apartado se comentará rápidamente cada uno de los capítulos de este documento con el objetivo de facilitar su lectura:

- Capítulo 1: Introducción y descripción de los objetivos y motivaciones para la realización del proyecto, así como las fases de desarrollo y el equipo utilizado para llevarlo a cabo.
- Capítulo 2: Estudio y comparación de las plataformas móviles actuales.
- Capítulo 3: Breve estudio del sistema operativo Android tanto de sus características principales, origen, arquitectura, las aplicaciones y qué las componen.
- Capítulo 4: Demostración del proceso de desarrollo para Android y descripción de las herramientas que tenemos disponibles para realizar esto.
- Capítulo 5: Explicación de las diferentes fases en el desarrollo de una aplicación para Android a través de un caso práctico.
- Capítulo 6: Posibilidades de futuro de la aplicación desarrollada y conclusiones finales obtenidas al finalizar el proyecto.

CAPÍTULO 2
PLATAFORMAS MÓVILES

2.1 INTRODUCCIÓN

Hoy en día estamos rodeados de dispositivos móviles. Estos conviven con nosotros siendo ya parte de nuestro día a día y de nuestra sociedad, pero comencemos por el principio. Una definición más o menos exacta de lo que es un dispositivo móvil sería la siguiente: "A*paratos de pequeño tamaño, con algunas capacidades de procesamiento, móviles o no, con conexión permanente o intermitente a una red, con memoria limitada, diseñados específicamente para una función, pero que pueden llevar a cabo otras funciones más generales*". Esta definición englobaría desde un teléfono móvil y una tableta a una PDA o un lector de libros electrónicos.

De los elementos antes mencionados, cabe destacar sin lugar a dudas, el teléfono móvil y más concretamente como este ha evolucionado hasta el día de hoy, el llamado teléfono inteligente o *Smartphone*, aunque es importante añadir que poco a poco las tabletas están ganando terreno respecto a estos.

Según los datos de la Comisión del Mercado de Telecomunicaciones (CMT) a fecha de Marzo de 2009, en España la media de teléfonos móvil por persona es superior al de uno por habitante y según la Unión Internacional de Telecomunicaciones (UIT) prevé que a lo largo de 2014 se pasarán de los 7.000 millones en todo el mundo.

Figura 1: Prototipo estándar de Smartphone

Teniendo en cuenta todos estos datos, la posibilidad de negocio en la telefonía móvil resulta muy interesante desde un punto de vista económico gracias a las facilidades que tenemos a la hora de desarrollar Software (ya sea de pago o gratuito pero económicamente rentable gracias a la publicidad) y poder distribuirlo al mundo entero mediante las diferentes plataformas de distribución digital de aplicaciones móviles como “App Store” o “Google Play”.

2.2 PLATAFORMAS MÓVILES ACTUALES

A continuación se hará un breve repaso y descripción de los sistemas operativos móviles más utilizados en la actualidad.

2.2.1 iOS

iOS es un sistema operativo móvil de la empresa Apple Inc. Originalmente desarrollado para el iPhone (iPhone OS), siendo después usado en dispositivos como el iPod Touch, iPad y el Apple TV. Apple, Inc.

La interfaz de usuario de *iOS* está basada en el concepto de manipulación directa usando gestos multitáctiles. Los elementos de control consisten en deslizadores, interruptores y botones. La respuesta a las órdenes del usuario es inmediata y provee de una interfaz fluida. La interacción con el sistema operativo incluye gestos como deslices, toques, pellizcos, los cuales tienen definiciones diferentes dependiendo del contexto de la interfaz. Se utilizan acelerómetros internos para hacer que algunas aplicaciones respondan a sacudir el dispositivo (por ejemplo, para el comando deshacer) o rotarlo en tres dimensiones (un resultado común es cambiar de modo vertical al apaisado u horizontal).

iOS deriva de *Mac OS X*, que a su vez está basado en *Darwin BSD*, y por lo tanto es un sistema operativo Tipo *Unix*.

iOS cuenta con cuatro capas de abstracción: la capa del núcleo del sistema operativo, la capa de "Servicios Principales", la capa de "Medios" y la capa de "Cocoa Touch". La versión actual del sistema operativo (iOS 7.1) ocupa aproximadamente 1.1 GB.

2.2.2 ANDROID

Android es un sistema operativo basado en el *kernel* de *Linux* diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas. Inicialmente desarrollado por Android Inc. que google respaldó económicamente y más tarde compró en 2005. Android fue presentado en 2007 junto la fundación del *Open Handset Alliance*: un consorcio de compañías de hardware, software y

telecomunicaciones para avanzar en los estándares abiertos de los dispositivos móviles.

Algunas de las características más importantes son:

- Plataforma adaptable a pantallas de diferentes resoluciones.
- Soporte de diferentes tecnologías de conectividad (GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, HSDPA, HSPA+, NFC Y WiMAX).
- Mensajería en forma de SMS y MMS.
- Navegador Web.
- Soporte para Java.
- Soporte de los siguientes formatos multimedia: WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, BMP.
- Multitáctil.

2.2.3 WINDOWS PHONE

Windows Phone es un sistema operativo móvil desarrollado por *Microsoft* como sucesor de *Windows Mobile*. A diferencia de su predecesor, está enfocado en el mercado de consumo en lugar de en el mercado empresarial. Con *Windows Phone*, Microsoft ofrece una interfaz de usuario que integra varios de sus servicios propios como *SkyDrive*, *Skype* y *Xbox Live* en el sistema operativo. Compite directamente contra Android de Google y iOS de Apple.

2.2.4 BLACKBERRY

BlackBerry OS es un sistema operativo móvil desarrollado por RIM para los dispositivos BlackBerry. El sistema permite multitarea y tiene soporte para diferentes métodos de entrada adoptados por RIM para su uso en computadoras de mano, particularmente la *trackwheel*, *trackball*, *touchpad* y pantallas táctiles.

Su desarrollo se remonta la aparición de los primeros *handheld* en 1999.

Esta claramente orientado a su uso profesional como gestor de correo electrónico y agenda. Desde la cuarta versión se puede sincronizar el dispositivo con el correo electrónico, el calendario, tareas, notas y contactos de *Microsoft Exchange Server*, además es compatible también con *Lotus Notes* y *Novell GroupWise*.

BlackBerry Enterprise Server (BES) proporciona el acceso y organización del email a grandes compañías identificando a cada usuario con un único *BlackBerry PIN*. Los usuarios más pequeños cuentan con el software *BlackBerry Internet Service*, programa

más sencillo que proporciona acceso a Internet y a correo POP3 / IMAP / Outlook Web Access sin tener que usar BES.

2.2.5 SYMBIAN OS

Symbian es un sistema operativo propiedad de Nokia y que en el pasado fue producto de la alianza de varias empresas de telefonía móvil entre las que se encontraban Nokia, Sony Mobile Comminications, Psion, Samsung, Siemens, Arima; Benq, Fujitsu, Lenovo, LG, Motorola, Mitsibishi Electric, Panasonic, Sharp, etc. Sus orígenes provenían de su antepasado *EPOC32*, utilizado en *PDA's* y *Handhelds* de PSION.

Figura 2: Interfaz de usuario y logo de los diferentes S.O móviles

2.3 COMPARACIÓN ENTRE PLATAFORMAS MÓVILES

En este apartado se comentarán los diferentes aspectos de las plataformas de desarrollo para Smartphones, analizando, sobre el punto de vista del desarrollador y del usuario, las ventajas e inconvenientes de cada uno de ellos.

En la siguiente tabla se compararán las diferentes características de las plataformas mencionadas:

	Apple iOS 7	Android 4.3	Windows Phone 8	BlackBerry OS 7	Symbian 9.5
Compañía	Apple	Open Handset Alliance	Microsoft	RIM	Symbian Foundation
Núcleo del SO	Mac OS X	Linux	Windows NT	Mobile OS	Mobile OS
Licencia de *software*	Propietaria	Software libre y abierto	Propietaria	Propietaria	Software libre
Año de lanzamiento	2007	2008	2010	2003	1997
Fabricante único	Sí	No	No	Sí	No
Variedad de dispositivos	modelo único	muy alta	media	baja	muy alta
Soporte memoria externa	No	Sí	Sí	Sí	Sí
Motor del navegador web	WebKit	WebKit	Pocket Internet Explorer	WebKit	WebKit
Soporte Flash	No	Sí	No	Si	Sí
HTML5	Sí	Sí	Sí	Sí	No
Tienda de aplicaciones	App Store	Google Play	Windows Marketplace	BlackBerry App World	Ovi Store
Número de aplicaciones	825.000	850.000	160.000	100.000	70.000
Coste publicar	$99 / año	$25 una vez	$99 / año	sin coste	$1 una vez
Actualizaciones automáticas del S.O.	Sí	depende del fabricante	depende del fabricante	Sí	Sí
Familia CPU soportada	ARM	ARM, MIPS, Power, x86	ARM	ARM	ARM
Máquina virtual	No	Dalvik	.net	Java	No
Aplicaciones nativas	Siempre	Sí	Sí	No	Siempre
Lenguaje de programación	Objective-C, C++	Java, C++	C#, muchos	Java	C++
Plataforma de desarrollo	Mac	Windows, Mac, Linux	Windows	Windows, Mac	Windows, Mac, Linux

Tabla 1: Comparativa de características de los sistemas operativos móviles

Primeramente nos fijaremos en iOS. Al ser *Apple* el fabricante único, nos encontramos con grandes ventajas en sus terminales como un muy buen diseño, funcionalidad y facilidad de uso. Además su *Apple Store* destaca junto a *Google Play* de Android por su ingente cantidad de aplicaciones de todo tipo para sus usuarios.

Ser un sistema cerrado nos da ventajas, pero esto supone menos posibilidades de cambiar la forma de funcionar del teléfono y un control más rígido sobre las aplicaciones publicadas.

Nos centraremos ahora en Android. En esta plataforma nos encontramos con que tenemos licencia de software libre y código abierto, esto es una ventaja tanto para desarrolladores como para usuarios ya que podemos personalizar los terminales al máximo.

Otro punto a destacar es sin duda que no tenemos un único fabricante, con lo cual, tenemos una amplísima oferta de terminales con diferentes precios y características además del ya mencionado *Google Play* donde disponemos de un catálogo amplísimo con todo tipo de aplicaciones para descargar.

Windows Phone llegó pisando fuerte y con una gran inversión. Nos encontramos con un sistema operativo con un diseño moderno, práctico y atractivo. Su hándicap fue llegar demasiado tarde. Apple y Google ya habían conquistado gran parte del mercado y eso se puede ver reflejado en la cantidad de aplicaciones que dispone en su *Windows Marketplace*.

Respecto a BlackBerry OS y Symbian nos encontramos ante dos sistemas operativos poco atractivos para los usuarios de hoy en día y eso se ve reflejado en sus respectivas tiendas online de aplicaciones. Hoy por hoy su público ha migrado a sistemas operativos como iOS o Android y parece ser que tienen sus días contados.

En definitiva y fijándonos en la tabla anterior, desde el punto de vista del usuario, nos encontramos como iOS y Android destacan muy por encima de las demás con sus respectivos *App Store* y *Google Play* por la gran cantidad de aplicaciones disponibles, siento esto hoy en día lo que más atrae al usuario medio de Smartphone. Windows Phone pelea por subir puestos y BlackBerry OS y Symbian quedaron hace tiempo muy atrás.

2.4 ¿POR QUÉ ELEGIR ANDROID?

Una vez hemos descrito y analizado los diferentes sistemas operativos que nos podemos encontrar para Smarthones, se explicará por que la plataforma escogida para la realización de este proyecto ha sido el sistema operativo Android de Google.

Si miramos el siguiente gráfico:

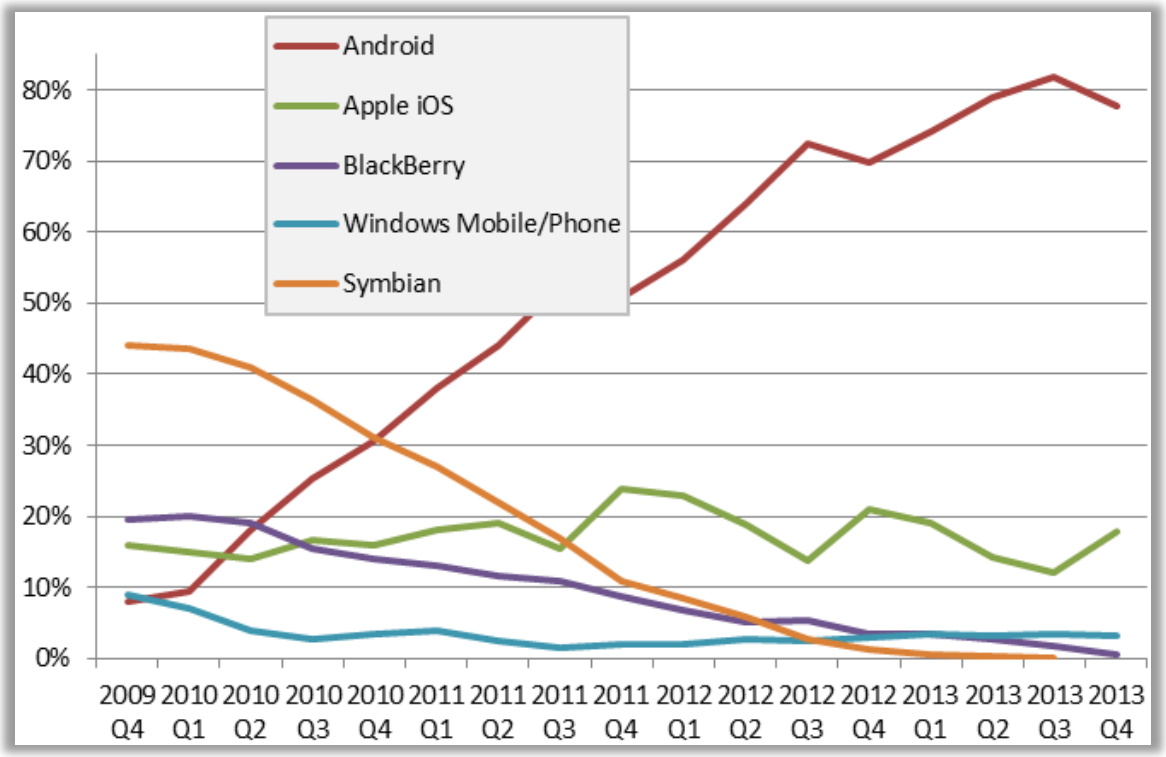


Figura 3: Cuota de mercado de las plataformas móviles

Podemos observar como sobre el año 2009/2010 Symbian era la mejor plataforma de desarrollo gracias a su amplio abanico de usuarios respecto a sus competidores. A medida que los Smartphones fueron adaptándose a nuestra sociedad, estos incrementaron su cuota de mercado, destacando Android por encima del resto con mucha diferencia. A día de hoy, Android con un 78% de cuota de mercado y Apple con un 18%, lideran el sector de los Smarphones.

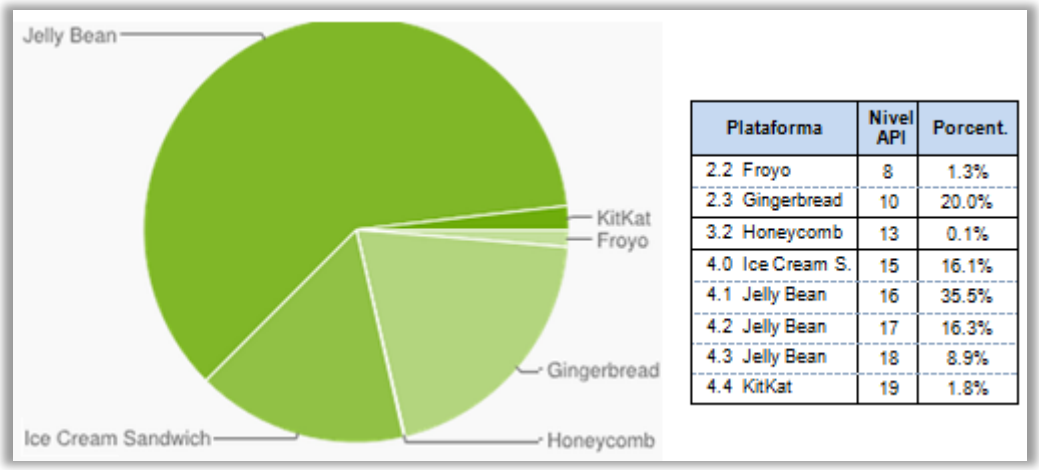
Por lo tanto y con la mente puesta en que el proyecto llegue al máximo número de usuarios posibles y habiendo analizado los datos anteriores, en principio Android sería la mejor plataforma posible para el lanzamiento del proyecto, pero antes de tomar una decisión, necesitamos también analizar las posibilidades desde el punto de vista del desarrollador.

Si nos fijamos en la *Tabla 1: Comparativa de características de los diferentes sistemas operativos móviles*, tenemos dos motivos muy importantes por los que inclinar la balanza a favor de Android. El primero de ellos se trata del coste de publicación. Mientras que en Android sólo necesitamos pagar 25$ para poder publicar nuestros desarrollos, para iPhone o iPad necesitaremos desembolsar 99$/año, suponiendo, en ciertas ocasiones, un coste inasumible para el desarrollador independiente. El segundo motivo consiste en qué plataformas es posible llevar a cabo el desarrollo. Si queremos trabajar para Android, podremos usar un MAC o cualquier PC con Windows o Linux mientras que si queremos trabajar para Apple necesitaremos obligatoriamente un MAC, incrementando aún más el coste de desarrollo.

En definitiva y habiendo hecho un análisis de mercado, tanto para llegar a una gran cantidad de usuarios como para ofrecer las máximas comodidades posibles a los desarrolladores, Android sería la mejor plataforma de desarrollo para este este proyecto.

○ *¿PARA QUE VERSIÓN DE ANDROID DESARROLLAR?*

Una vez hemos decidido que la mejor opción es desarrollar para Android, deberemos decidir también qué versión de este elegir. Desde su lanzamiento en noviembre de 2007, Android se ha ido actualizando regularmente (Mirar *CAPÍTULO 3.3: EVOLUCIÓN*), por lo tanto no deja de ser una opción importante saber para qué versión del sistema operativo trabajar. Si miramos la siguiente gráfica:



Plataforma	Nivel API	Porcent.
2.2 Froyo	8	1.3%
2.3 Gingerbread	10	20.0%
3.2 Honeycomb	13	0.1%
4.0 Ice Cream S.	15	16.1%
4.1 Jelly Bean	16	35.5%
4.2 Jelly Bean	17	16.3%
4.3 Jelly Bean	18	8.9%
4.4 KitKat	19	1.8%

Figura 4: Cuota de mercado de las versiones de Android

Podremos ver el porcentaje de usuarios que accedieron a *Google Play Store* durante dos semanas terminado el 4 de febrero de 2014, es decir un reflejo del porcentaje de usuarios que usan las distintas versiones de Android.

En principio la versión más usada seria la *4.1 Jelly Bean* pero si trabajásemos directamente para esta versión, dejaríamos fuera a una gran parte de posibles usuarios de nuestra aplicación. Por lo tanto usaremos la versión *2.3 Gingerbread*, esta nos proporciona todas las herramientas que necesitamos para el desarrollo de este proyecto y será compatible prácticamente con casi toda la totalidad de usuarios de Android.

CAPÍTULO 3
SISTEMA OPERATIVO ANDROID

3.1 INTRODUCCIÓN

Android es un sistema operativo para dispositivos móviles (principalmente para Smartphone y Tablet). Está basado en *GNU/Linux* e inicialmente fue desarrollado por Google.

Esta plataforma permite el desarrollo de aplicaciones por terceros (personas ajenas a Google), para lo cual, los desarrolladores deben de escribir código gestionado en el lenguaje de programación Java y controlar los dispositivos por medio de bibliotecas desarrolladas o adaptadas por Google, es decir, escribir programas en C u otros lenguajes, utilizando o no las bibliotecas de Google (compilándolas a código nativo de ARM). Sin embargo, este esquema de desarrollo no es oficialmente soportado por Google.

La mayoría del código fuente de Android ha sido publicado bajo la licencia del software Apache, una licencia de software libre y código abierto.

Las principales características de Android son las siguientes:

- Cuenta con un **framework de aplicaciones** que permite **reutilizar** o **sustituir** las aplicaciones existentes.
- Cuenta con una **máquina virtual** especialmente **optimizada para dispositivos móviles** conocida como *Dalvik VM*.
- **Navegador** integrado basado en el **motor** de código abierto **WebKit**.
- Capaz de procesar **gráficos en 2D** (gracias a la librería SGL) y **gráficos 3D** basados en la especificación *OpenGL ES 1.X/2.0* (dependiendo de la versión de Android).
- Soporte para el almacenamiento de datos estructurados mediante las librerías **SQLite**.
- Soporte de **múltiples formatos multimedia** tanto de audio como video o imagen (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- **Telefonía** GSM.
- **Comunicaciones** siguiendo los protocolos estándar **Bluetooth**, **EDGE**, **3G**, y **Wifi**.

- Soporte de múltiples dispositivos hardware como **cámaras**, **GPS**, **brújula** o **acelerómetros**.

- Potente entorno de desarrollo que incluye un emulador, herramientas de depuración o un complemento para el *IDE Eclipse*.

3.2 ORIGEN

Google adquiere Android Inc. en el año 2005. Se trataba de una pequeña compañía que acababa de ser creada orientada a la producción de aplicaciones para terminales móviles. Ese mismo año empiezan a trabajar en la creación de una máquina virtual Java optimizada para móviles (*Dalvik VM*).

En el año 2007 se crea el consorcio *Handset Alliance* con el objetivo de desarrollar estándares abiertos para móviles. Está formado por Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel y otros. Una pieza clave de los objetivos de esta alianza es promover el diseño y difusión de la plataforma Android. Sus miembros se han comprometido a publicar una parte importante de su propiedad intelectual como código abierto bajo licencia *Apache v2.0*.
En noviembre del 2007 se lanza una primera versión del Android SDK. Al año siguiente aparece el primer móvil con Android (*T-Mobile G1*). En octubre Google libera el código fuente de Android principalmente bajo licencia de código abierto Apache (licencia GPL v2 para el núcleo). Ese mismo mes se abre *Android Market*, para la descarga de aplicaciones.

En abril del 2009 Google lanza la versión 1.5 del SDK que incorpora nuevas características como el teclado en pantalla. A finales del 2009 se lanza la versión 2.0 y durante el 2010 las versiones 2.1, 2.2 y 2.3.

Durante el año 2010 Android se consolida como uno de los sistemas operativos para móviles más utilizados, con resultados cercanos a iOS e incluso superando al sistema de Apple en EE.UU.

En el 2011 se lanzan la versión 3.0, 3.1 y 3.2 específica para tabletas y la 4.0 tanto para móviles como para tabletas. Durante este año Android se consolida como la plataforma para móviles más importante alcanzando una cuota de mercado superior al 50%.

En 2012 Google cambia su estrategia en su tienda de descargas online, reemplazando *Android Market* por *Google Play Store*. Donde en un solo portal unifica tanto la descarga de aplicaciones como de contenidos. En este año aparecen las versiones 4.1 y

4.2 del SDK. Android mantiene su espectacular crecimiento, alcanzando a finales de año la cuota de mercado del 70%.

En 2013 se lanzan las versiones 4.3 y 4.4 (*KitKat*). A finales de este año la cuota de mercado llega al 80%.

3.3 EVOLUCIÓN

Android ha ido evolucionando con el paso del tiempo. Desde que la primera versión comercial, Android 1.0, fue lanzada en septiembre de 2008, Android ha visto un número considerable de actualizaciones de su sistema operativo. Estas actualizaciones típicamente corrigen fallos del sistema y agregan nuevas funcionalidades.

Desde abril de 2009, las versiones de Android han sido desarrolladas bajo un nombre en clave y lanzamiento en orden alfabético: *Apple Pie*, *Banana Bread*, *Cupcake*, *Donut*, *Éclair*, *Froyo*, *Gingerbread*, *Honeycomb*, *Ice Cream Sandwich*, *Jelly Bean* y *KitKat*. La actualización más reciente es *KitKat 4.4* la cual fue anunciada en septiembre de 2013, y lanzada en el proyecto *Android Open Source* un mes después.

Versión	Características
1.0 Apple Pie	Android 1.0 *Apple Pie*, es la primera versión comercial del software. Fue lanzado el 23 septiembre de 2008. El primer dispositivo Android en incorporarlo fue el HTC Dream (comercializado también como *T-Mobile G1*). Incluía las siguientes características: • *Android Market*: Programa para la descarga y actualización de aplicaciones. • Navegador Web para visualizar páginas webs en HTML y XHTML con la posibilidad de abrir múltiples páginas mostradas como ventanas. • Soporte para cámara. Sin embargo esta versión carece de la opción de cambiar la resolución de la misma, balance de blancos, calidad, etc. • Acceso a servidores de correo electrónico por web, soporte POP3, IMAP4 y SMTP. • Sincronización con *Gmail*, *Google Contacts* y *Google Calendar*. • *Google Maps* con *Latitude* y *Google Street View* para ver

	mapas e imágenes por satélite, así como para encontrar negocios locales y obtener direcciones usando GPS. • *Google Sync*, el cual permite la sincronización de *Gmail*, los contactos y el calendario • *Google Search*, para que los usuarios puedan buscar a través de internet, en aplicaciones del teléfono móvil, en contactos, en calendario, etc. • Mensajería instantánea *Google Talk*. • Mensajería instantánea por mensajes de texto y MMS. • Reproductor de medios para la importación, y reproducción de archivos multimedia, sin embargo, esta versión carece de soporte de vídeo y estéreo por Bluetooth. • Las notificaciones aparecen en la barra de estado, con opciones para configurar alertas por sonidos, LED o vibración. • La marcación por voz permite marcar y llamar sin escribir nombre o número. • El fondo de escritorio permite al usuario configurar una imagen de fondo o una foto detrás de los iconos y widgets de la pantalla de inicio. • Reproductor de vídeo de *YouTube*. • Otras aplicaciones incluyen: Alarma, Calculadora, Marcación (teléfono), Pantalla de inicio (*launcher*), Imágenes (Galería) y ajustes. • Soporte para *Wi-Fi* y *Bluetooth*.
Android 1.1 Banana Bread	El 9 de febrero de 2009 fue lanzada la actualización Android 1.1 *Banana Bread*. Inicialmente solo para el *HTC Dream* así que en principio solo sirve para este teléfono. Android 1.1 fue conocido como "Petit Four" internamente, aunque este nombre no se utilizó oficialmente. La actualización resolvió fallos, cambio la API y agregó una serie de características: • Detalles y reseñas disponibles cuando un usuario busca negocios en los mapas a través de *Google Maps*. • Posibilidad de guardar archivos adjuntos en los mensajes.
Android 1.5 Cupcake	El 30 de abril de 2009, La actualización de Android 1.5 *Cupcake* fue lanzada. Está basada en el núcleo de Linux 2.6.27.

	La actualización incluye nuevas características y correcciones del interfaz de usuario: • Soporte para teclados virtuales de terceros con predicción de texto y diccionario de usuarios para palabras personalizadas. • Soporte para *Widgets* - vistas de miniaturas de las aplicaciones que pueden ser incrustadas en otras aplicaciones (tal como la pantalla inicio) y recibir actualizaciones periódicas. • Grabación y reproducción en formatos *MPEG-4* y *3GP*. • Auto-sincronización y soporte para Bluetooth estéreo añadido (perfiles *A2DP* y *AVRCP*) • Características de Copiar y pegar agregadas al navegador web. • Las fotos de los usuarios pueden ser mostradas y agregadas para los contactos. • Marcas de fecha/hora mostradas para eventos en los registros de llamadas y acceso con un "click" a la tarjeta de un contacto desde un evento del registro de llamadas. • Pantallas de transiciones animadas. • Agregada opción de auto-rotación. • Agregada la animación de inicio por defecto. • Posibilidad de subir vídeos a *YouTube*. • Posibilidad de subir fotos a *Picasa*.
Android 1.6 Donut	El 15 de septiembre de 2009, fue lanzado el SDK de Android 1.6 *Donut*, basado en el núcleo Linux 2.6.29. En la actualización se incluyeron numerosas características nuevas: • Mejora en la búsqueda por entrada de texto y voz para incluir historial de favoritos en los contactos y en la web. • Posibilidad para que los desarrolladores puedan incluir su contenido en los resultados de las búsquedas. • Motor multi-lenguaje de síntesis de voz para permitir a cualquier aplicación de Android "hablar" una cadena de texto. • Búsqueda facilitada y posibilidad de ver capturas de las aplicaciones en el *Android Market* (actualmente *Google Play*). • Galería, cámara y videocámara con mejor integración y con

	rápido acceso a la cámara. • La galería ahora permite a los usuarios seleccionar varias fotos para eliminarlas. • Actualización del soporte de la tecnología para CDMA/EVDO, 802.1x y VPNs. • Soporte para resoluciones de pantalla WVGA. • Mejoras de velocidad en búsqueda y aplicaciones de cámara. • Nueva herramienta de desarrollo *GestureBuilder*.
Android 2.0/2.1 Eclair	El 26 de octubre de 2009, el SDK de Android 2.0 (con nombre en clave *Eclair*) fue lanzado, basado en el núcleo de Linux 2.6.29. Los cambios incluyen: • Mejora en las cuentas de usuario permitiendo a estos agregar múltiples cuentas al dispositivo para sincronización de correo y contactos. • Soporte e intercambio de correos, con la posibilidad de buscar correos desde múltiples cuentas. • Soporte para *Bluetooth* 2.1. • Mejora de la lista de contactos dando la posibilidad de "tocar" un foto de un contacto y seleccionar si se desea llamar, enviar un SMS o un correo electrónico. • Posibilidad, para que en todos los mensajes SMS y MMS guardados, de poder eliminar los mensajes más antiguos de forma automática cuando se ha sobrepasado un límite definido previamente. • Nuevas características para la cámara, incluyendo soporte de flash, zoom digital, modo escena, balance de blancos, efecto de colores y enfoque macro. • Mejora de la velocidad de tecleo en el teclado virtual, con diccionario inteligente que aprende el uso de palabras e incluye nombres de contactos como sugerencias. • Renovada interfaz de usuario del navegador web con imágenes en miniatura de marcadores, zoom con doble "click" y soporte para HTML5. • Agenda del calendario mejorada. Ahora muestra el estado de los eventos organizados y si los invitados acudirán a los

	mismos. También se ha mejorado la capacidad de invitar a nuevas personas a los eventos. • Optimización en velocidad del *hardware* y *GUI* renovada. • Soporte para más tamaños de pantalla y resoluciones, con mejor ratio de contraste. • Mejorado *Google Maps* a la versión3.1.2. • Clase *MotionEvent* mejorada para rastrear eventos *multi-touch.* • Adición de fondos de pantalla animados, permitiendo la animación de imágenes de fondo de la pantalla inicio para mostrar movimiento.
Android 2.2 Froyo	El 20 de mayo de 2010 fue lanzado el SDK de Android 2.2 *Froyo*. Está basado en el núcleo Linux 2.6.32. • Optimizaciones en velocidad, memoria y rendimiento. • Mejoras adicionales de rendimiento de las aplicaciones implementadas mediante compilación *Just-in-time* (JIT) • Integración del motor de JavaScript V8 de Chrome en el navegador web. • Soporte para el servicio *Android Cloud* • Soporte para *Microsoft Exchange* mejorado, incluyendo políticas de seguridad, consulta a la *Global Access List* (GAL), sincronización de calendario, y borrado remoto. • Mejoras en la aplicación del lanzador con accesos directos de las aplicaciones del teléfono y navegador web. • Funcionalidad de establecimiento de red por *USB* y *Wi-Fi hotspot*. • Agregada opción para deshabilitar el acceso de datos sobre red móvil. • Actualizada la aplicación *Android Market* con características de grupo y actualizaciones automáticas • Cambio rápido entre múltiples lenguajes de teclado y diccionario. • Discado por voz e intercambio de contactos por Bluetooth • Soporte para contraseñas numéricas y alfanuméricas. • Soporte para subida de archivos en la aplicación del

	navegador. • Soporte para la instalación de aplicaciones en la memoria externa. • Soporte para *Adobe Flash*. • Soporte para pantallas de alto número de *PPI* (320 ppi), como 4" 720p. • Galería que permite a los usuarios ver pilas de imágenes mediante un gesto de zoom.
Android 2.3 Gingerbread	El 6 de diciembre de 2010 fue lanzado, el SDK de Android 2.3 *Gingerbread*. Está basado en el núcleo Linux 2.6.35. Los cambios incluyen: • Actualizado el diseño de la interfaz de usuario con incrementos en velocidad y simpleza. • Soporte para tamaños y resoluciones de pantalla extra-grandes (WXGA y mayores). • Soporte nativo para *SIP* y telefonía por internet *VoIP*. • Entrada de texto del teclado virtual más rápido e intuitivo, con mejoras en precisión, texto sugerido y entrada por voz. • Mejoras en la funcionalidad de copiar/pegar, permitiendo a los usuarios seleccionar una palabra al presionar-mantener, copiar y pegar. • Soporte para *Near Field Communication* (NFC), permitiendo al usuario leer la etiqueta NFC incrustada en un póster, sticker o anuncio publicitario. • Nuevos efectos de audio tales como reverberación, ecualizador, virtualización de audífonos y aumento de bajos. • Nuevo gestor de descargas que da a los usuarios un fácil acceso a cualquier archivo descargado a través del navegador, correo electrónico u otra aplicación. • Soporte para múltiples cámaras en el dispositivo, incluyendo cámara frontal-facial, si está disponible. • Soporte para reproducción de video por *WebM*/*VP8*, en codec de audio por *AAC*. • Mejoras en la administración de la energía, con un mayor rol activo en aplicaciones de administración que se

	mantienen activas en el dispositivo por mucho tiempo. • Mejorado el soporte para el desarrollo de código nativo. • Cambio desde *YAFFS* a *ext4* en dispositivos nuevos. • Mejoras en audio, gráficos y entrada para desarrolladores de juegos. • Recolector de basura concurrente para incrementar el rendimiento. • Soporte nativo para más sensores (tales como giroscopio y barómetro). • Varias mejoras y arreglos de la API. • Rebaja de la seguridad de SSL (*Secure Sockets Layer*) al usar protocolos de cifrado inseguros. • Soporte de chat de video o voz, usando Google Talk. • Soporte a la biblioteca *Open Accessory*. • Mejoras en el sistema. • Mejoras en el rendimiento por red del Nexus S 4G. • Arreglado un fallo de Bluetooth en el Samsung Galaxy S. • Mejoras a la aplicación de correo electrónico. • Animación de sombras al deslizar por listas. • Mejoras al software de la cámara. • Mejorada la eficiencia de la batería. • Arreglado fallo en la búsqueda por voz. • Soporte de *Google Wallet* para el Nexus S 4G, esta versión es exclusiva para usuarios en Canadá.
Android 3.0 Honeycomb	El 22 de febrero de 2011, sale el SDK de Android 3.0 (*Honeycomb*). Fue la primera actualización exclusiva para tablet, lo que quiere decir que sólo es apta para tablets y no para teléfonos Android. Está basada en el núcleo de Linux 2.6.36. El primer dispositivo con esta versión fue la tableta *Motorola Xoom*, lanzada el 24 de febrero de 2011. Las características de la actualización incluyen: • Soporte optimizado para tablets, con una nueva y "virtual" interfaz de usuario holográfica. • Agregada barra de sistema, con características de acceso rápido a notificaciones, estados y botones de navegación

	suavizados, disponible en la parte inferior de la pantalla. • Añadida barra de acción (*Action Bar* en inglés), dando acceso a opciones contextuales, navegación, widgets u otros tipos de contenido en la parte superior de la pantalla. • Multitarea simplificada. Tocando Aplicaciones recientes en la barra del sistema permite a los usuarios ver de forma instantanea las tareas en curso y saltar rápidamente de una aplicación a otra. • Teclado rediseñado, permitiendo una escritura rápida, eficiente y acertada en pantallas de gran tamaño. • Interfaz simplificada y más intuitiva para copiar/pegar. • Las pestañas múltiples reemplazan las ventanas abiertas en el navegador web, además de la característica de auto-completar texto y un nuevo modo de "incógnito" permitiendo la navegación de forma anónima. • Acceso rápido a las características de la cámara como la exposición, foco, flash, zoom, cámara facial-frontal, temporizador u otras. • Permitir ver álbumes y otras colecciones de fotos en modo pantalla completa en galería, con un fácil acceso a vistas previas de las fotografías. • Nueva interfaz de contactos de dos paneles y desplazamiento rápido para permitir a los usuarios organizar y reconocer contactos fácilmente. • Nueva interfaz de correo de dos paneles para hacer la visualización y organización de mensajes más eficiente, permitiendo a los usuarios seleccionar uno o más mensajes. • Soporte para video chat usando *Google Talk*. • Aceleración de hardware. • Soporte para microprocesadores multi-núcleo. • Habilidad para encriptar todos los datos del usuario. • Mejoras en el uso de HTTPS con *Server Name Indication* (SNI). • Refinamiento a la interfaz de usuario. • Conectividad para accesorios USB. • Lista expandida de aplicaciones recientes.

	• Widgets redimensionables en la pantalla de inicio. • Soporte para teclados externos y dispositivos punteros. • Soporte para joysticks y gamepads. • Soporte para reproducción de audio *FLAC*. • Bloqueo de *Wi-Fi* de alto rendimiento, manteniendo conexiones *Wi-Fi* de alto rendimiento cuando la pantalla del dispositivo está apagada. • Soporte para *proxy HTTP* para cada punto de acceso *Wi-Fi* conectado • Mejoras de soporte de hardware, incluyendo optimizaciones para un amplio rango de tabletas. • Incrementada la capacidad de las aplicaciones para acceder a archivos de las tarjetas SD, por ejemplo para sincronización. • Modo de vista de compatibilidad para aplicaciones que no han sido optimizadas para resoluciones de pantalla de tabletas. • Nuevas funciones de soporte de pantalla, dando a los desarrolladores un mayor control sobre la apariencia de la pantalla en diferentes dispositivos Android. • Corrección de errores menores y mejoras de seguridad, mejoras en estabilidad y *Wi-Fi*. • Actualización del *Android Market* con actualizaciones texto de términos y condiciones de fácil lectura. • Actualización de *Google Books*. • Mejoras en el soporte de *Adobe Flash* del navegador. • Mejoras en la predicción de escritura a mano en chino. • Soporte para "Pay as You Go" para tabletas 3G y 4G.
Android 4.0 Ice Cream Sandwich	El SDK para Android 4.0.0 (*Ice Cream Sandwich*), basado en el núcleo de Linux 3.0.1, fue lanzado públicamente el 19 de octubre de 2011. Gabe Cohen de Google declaró que Android 4.0 era "teóricamente compatible" con cualquier dispositivo Android 2.3 en producción en ese momento, pero sólo si su procesador y memoria RAM lo soportaban. El código fuente para Android 4.0 se puso a disposición el 14 de noviembre de 2011. La actualización incluye numerosas novedades, entre ellas:

- Botones suaves, en Android 3.x están ahora disponibles para usar en los teléfonos móviles.
- Separación de widgets en una nueva pestaña, listados de forma similar a las aplicaciones.
- Facilidad para crear carpetas, con estilo de arrastrar y soltar.
- Lanzador personalizable.
- Buzón de voz mejorado con la opción de acelerar o retrasar los mensajes del buzón de voz.
- Funcionalidad de *pinch-to-zoom* en el calendario.
- Captura de pantalla integrada (manteniendo presionado los botones de bloqueo y de bajar volumen).
- Corrector ortográfico del teclado mejorado.
- Habilidad de acceder a aplicaciones directamente desde la pantalla de bloqueo.
- Funcionalidad copiar-pegar mejorada.
- Mejor integración de voz y dictado de texto en tiempo real.
- Desbloqueo facial, característica que permite a los usuarios desbloquear los equipos usando software de reconocimiento facial.
- Nuevo navegador web con pestañas bajo la marca de *Google Chrome*, permitiendo hasta 15 pestañas de forma simultánea.
- Sincronización automática del navegador con los marcadores de *Google Chrome* del usuario.
- Nueva tipografía para la interfaz de usuario, *Roboto*.
- Sección para el uso de datos dentro de la configuración que permite al usuario poner avisos cuando se acerca a cierto límite de uso y desactivar los datos cuando se ha excedido dicho límite.
- Capacidad para cerrar aplicaciones que están usando datos en segundo plano.
- Aplicación de la cámara mejorada sin retardo en el obturador, ajustes para el time lapse, modo panorámico y la posibilidad de hacer zoom durante la grabación.
- Editor de fotos integrado.

	• Nuevo diseño de la galería, organizada por persona y localización. • Aplicación *People* actualizada con integración en redes sociales, actualización de estados e imágenes en alta resolución. • *Android Beam*, una característica de Near Field Communication que permite el rápido intercambio de corto alcance de enlaces web favoritos de un navegador de internet, información de contactos, direcciones, vídeos de YouTube y otros datos. • Soporte para el formato de imagen *WebP*. • Aceleración por hardware de la interfaz de usuario. • *Wi-Fi* Direct. • Grabación de vídeo a 1080P para dispositivos con Android de serie. • Numerosas optimizaciones y corrección de errores. • Mejoras en gráficos, bases de datos, corrección ortográfica y funcionalidades *Bluetooth*. • Nueva API para los desarrolladores, incluyendo una API de actividad social en el proveedor de contactos. • Mejoras en el calendario. • Nuevas aplicaciones de la cámara en la mejora de la estabilidad en los videos y resolución QVGA. • Mejoras de accesibilidad tales como la mejora de acceso al contenido para lectores de pantalla. • Mejoras de estabilidad. • Mejor rendimiento de la cámara. • Rotación de la pantalla más fluida. • Mejoras en el reconocimiento de los números en el teléfono.
Android 4.1 Jelly Bean	Google anunció Android 4.1 (*Jelly Bean*) en conferencia el 27 de julio de 2012. Basado en el núcleo de Linux 3.0.31, *Jelly Bean* fue una actualización incremental con el enfoque primario de mejorar la funcionalidad y el rendimiento de la interfaz de usuario. La mejora de rendimiento involucró el "Proyecto Butter", el cual usa anticipación táctil, triple buffer, latencia vsync extendida y un

	arreglo en la velocidad de cuadros de 60 fps para crear una fluida interfaz de usuario. Android 4.1 *Jelly Bean* fue lanzado bajo *AOSP* el 9 de julio de 2012, y el Nexus 7, el primer dispositivo en correr *Jelly Bean*, fue lanzado el 13 de julio de 2012.
Android 4.4 KitKat	• Cambios en la interfaz. • Agenda como listín de contacto. • *Hangouts* se integra con los *SMS*. • Optimizado para sistemas como poca memoria RAM. • *NFC* y tarjetas virtuales para usarlo como una cartera. • Optimización del uso de los sensores. • Gestión de archivos e impresión. • *Google Now* cobra mayor protagonismo.

Tabla 2: Características de las versiones de Android

3.4 ARQUITECTURA

En la siguiente figura podemos ver un esquema donde se muestran las diferentes capas que componen el sistema operativo Android. Cada una de estas capas utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos propios a las capas de niveles superiores:

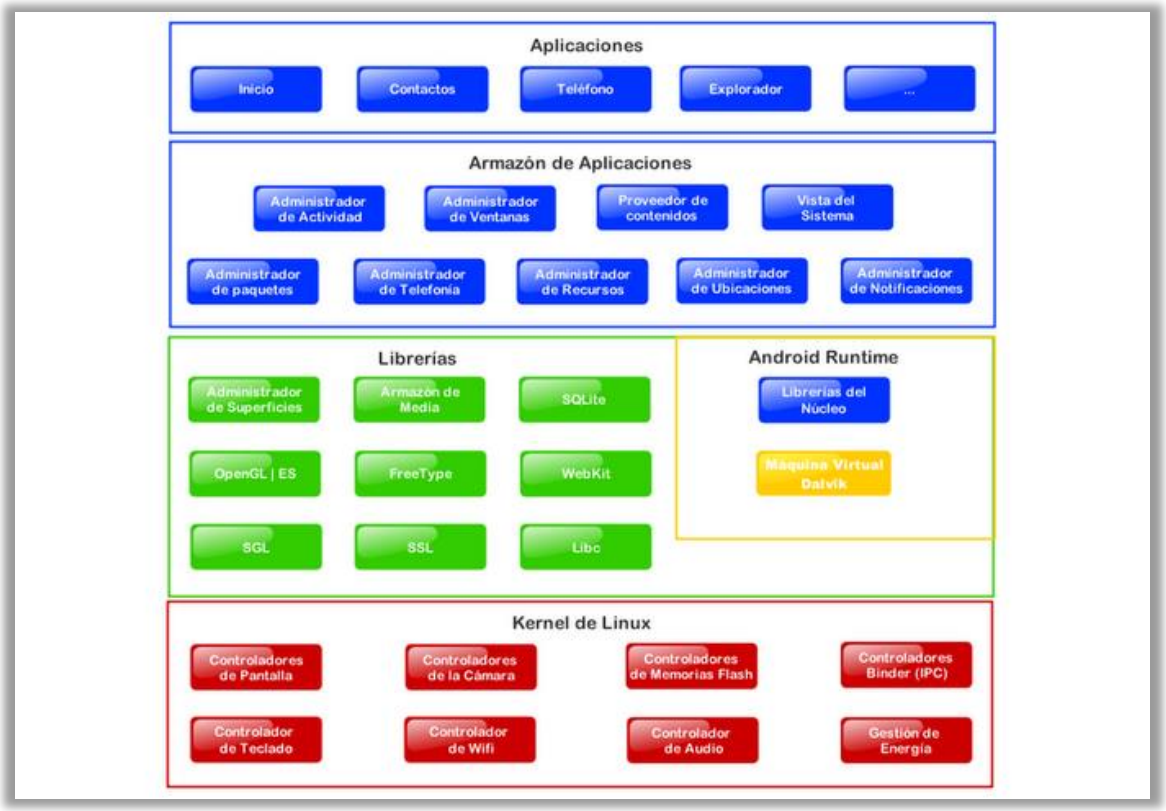


Figura 5: Arquitectura de Android

3.4.1 NÚCLEO LINUX

Android utiliza el **núcleo de Linux 2.6** como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluye un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las librerías de control o drivers necesarios dentro de este kernel de Linux embebido en el propio Android.

3.4.2 RUNTIME DE ANDROID

Está basado en el concepto de máquina virtual utilizado en Java. Dado las limitaciones de los dispositivos donde ha de correr Android (poca memoria y procesador limitado) no fue posible utilizar una máquina virtual Java estándar. Google tomó la decisión de crear una nueva, la máquina virtual *Dalvik*, que respondiera mejor a estas limitaciones.

Algunas características de la máquina virtual Dalvik que facilitan esta optimización de recursos son que ejecuta ficheros *Dalvik* ejecutables (.dex), formato optimizado para ahorrar memoria. Además, está basada en registros. Cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina virtual *Dalvik*. Delega al kernel de Linux algunas funciones como *threading* y el manejo de la memoria a bajo nivel.

También se incluye en el *Runtine de Android* el "core libraries" con la mayoría de las librerías disponibles en el lenguaje Java.

3.4.3 LIBRERÍAS NATIVAS

La siguiente capa se corresponde con las librerías utilizadas por Android. Éstas han sido escritas utilizando C/C++ y proporcionan a Android la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas librerías constituyen el corazón de Android.

Entre las librerías más importantes ubicadas aquí, se pueden encontrar las siguientes:

- *Librería libc:* Incluyen todas las cabeceras y funciones según el estándar del lenguaje C. Todas las demás librerías se definen en este lenguaje.

- *Librería Surface Manager:* Es la encargada de componer los diferentes elementos de navegación de pantalla. Gestiona también las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.

- *OpenGL/SL y SGL:* Representan las librerías gráficas y, por tanto, sustentan la capacidad gráfica de Android. *OpenGL/SL* maneja gráficos en 3D y permite utilizar, en caso de que esté disponible en el propio dispositivo móvil, el hardware encargado de proporcionar gráficos 3D. Por otro lado, *SGL* proporciona gráficos en 2D, por lo que será la librería más habitualmente utilizada por la mayoría de las aplicaciones. Una característica importante de la capacidad gráfica de Android es que es posible desarrollar aplicaciones que combinen gráficos en 3D y 2D.

· *Librería Media Libraries:* Proporciona todos los códecs necesarios para el contenido multimedia soportado en Android (vídeo, audio, imágenes estáticas y animadas, etc.)

· *FreeType:* Permite trabajar de forma rápida y sencilla con distintos tipos de fuentes.

· *Librería SSL:* Posibilita la utilización de dicho protocolo para establecer comunicaciones seguras.

· *Librería SQLite:* Creación y gestión de bases de datos relacionales.

· *Librería WebKit:* Proporciona un motor para las aplicaciones de tipo navegador y forma el núcleo del actual navegador incluido por defecto en la plataforma Android.

3.4.4 NIVELES DE FRAMEWORK DE APLICACIONES

Representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo "framework", representado por este nivel.

Entre las API más importantes ubicadas aquí, se pueden encontrar las siguientes:

· *Activity Manager:* Conjunto de API que gestiona el ciclo de vida de las aplicaciones en Android.

· *Window Manager:* Gestiona las ventanas de las aplicaciones y utiliza la librería Surface Manager.

· *Telephone Manager:* Incluye todas las API vinculadas a las funcionalidades propias del teléfono (llamadas, mensajes, etc.).

· *Content Provider:* Permite a cualquier aplicación compartir sus datos con las demás aplicaciones de Android. Por ejemplo, gracias a esta API la información de contactos, agenda, mensajes, etc. será accesible para otras aplicaciones.

· *View System:* Proporciona un gran número de elementos para poder construir interfaces de usuario (GUI), como listas, mosaicos, botones,

"check-boxes", tamaño de ventanas, control de las interfaces mediante teclado, etc. Incluye también algunas vistas estándar para las funcionalidades más frecuentes.

· *Location Manager:* Posibilita a las aplicaciones la obtención de información de localización y posicionamiento

· *Notification Manager:* Mediante el cual las aplicaciones, usando un mismo formato, comunican al usuario eventos que ocurran durante su ejecución: una llamada entrante, un mensaje recibido, conexión Wi-Fi disponible, ubicación en un punto determinado, etc. Si llevan asociada alguna acción, en Android denominada **Intent**, (por ejemplo, atender una llamada recibida) ésta se activa mediante un simple clic.

· *XMPP Service:* Colección de API para utilizar este protocolo de intercambio de mensajes basado en XML.

3.4.5 NIVEL DE APLICACIONES

Este nivel contiene, tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o de su propio desarrollo. Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores.

3.5 APLICACIONES

A continuación se explicarán una serie de elementos que pueden componer una aplicación para Android.

3.5.1 ACTIVIDADES

El concepto de actividad en Android representa una unidad de interacción con el usuario, es lo que coloquialmente llamamos una pantalla de la aplicación. Una aplicación puede estar formada por una o varias actividades, de forma que el usuario puede ir navegando entre actividades. En concreto, Android suele disponer de un botón (físico o en pantalla) que nos permite volver a la actividad anterior, es decir la actividad que ha creado a la actividad actual.

Toda Actividad ha de tener una vista asociada, que será utilizada como interfaz de usuario. Esta vista suele ser de tipo *Layout* aunque no es imprescindible.

Una aplicación estará formada por un conjunto de actividades independientes, es decir se trata de clases independientes que no comparten variables, aunque todas trabajan para un objetivo común. Otro aspecto importante es que toda actividad ha de ser una subclase de Activity.

Cuando las aplicaciones cuentan con varias actividades, las cuales son totalmente independientes, se suele indicar a una de ellas como la principal, de forma que cuando se inicia la aplicación esta es la actividad que aparecerá en primer lugar al usuario, y en el momento que se quiere cambiar a otra actividad sólo es necesario iniciarla desde la actual.

○ *CICLO DE VIDA DE UNA ACTIVIDAD*

Una característica importante que diferencia a Android de otros sistemas operativos es el proceso de liberación de memoria, es decir, la destrucción de un proceso, el cual, no es controlada directamente por la aplicación sino que será el sistema quien determine cuando eliminar un proceso dependiendo de cuán importante es el proceso que se está ejecutando y de la memoria disponible en ese determinado momento.

Si una vez eliminado el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso pero perdiendo el estado que tenía la aplicación. En estos casos será la responsabilidad del programador guardar el estado de las actividades si queremos recuperar esa información.

Por lo tanto, Android es sensible al ciclo de vida de una actividad la cual se compone de los siguientes 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. El programador debe 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.

Cuando una actividad cambia de estado, se generarán eventos que podrán ser capturados. A continuación se muestra un esquema del ciclo de vida de una actividad y que eventos pueden ser capturados:

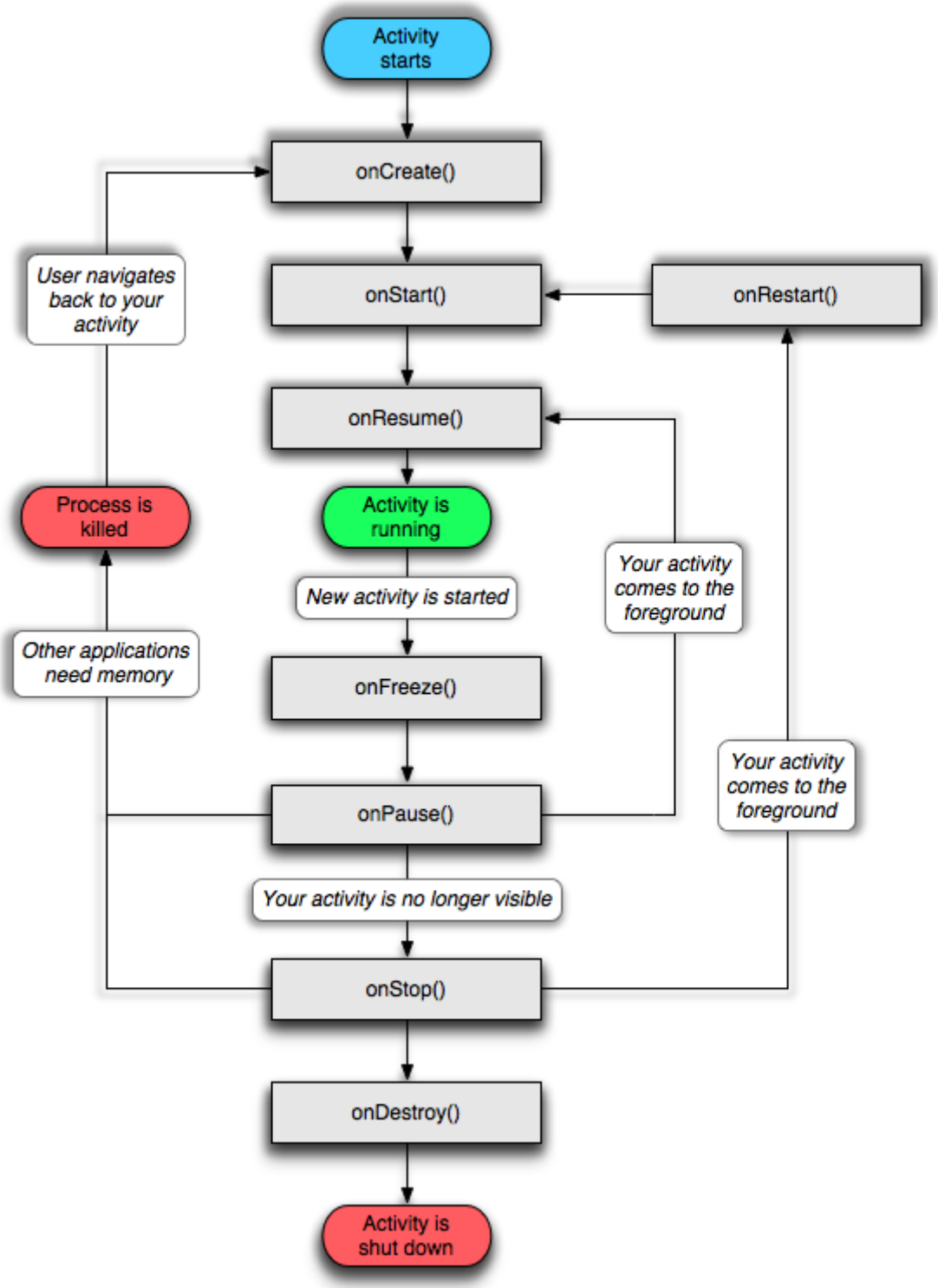


Figura 6: Ciclo de vida de una actividad

- **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 del estado de la actividad (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 actividad 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. 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. Si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

3.5.2 INTENCIONES

Una *intención* representa la voluntad de realizar alguna acción o tarea; como realizar una llamada de teléfono o visualizar una página web. Una intención nos permite lanzar una actividad o servicio de nuestra aplicación o de una aplicación diferente.

Existen dos tipos de intenciones:

- **Intenciones explícitas:** Se indica exactamente el componente a lanzar. Su utilización típica es la de ir ejecutando los diferentes componentes internos de una aplicación.

- **Intenciones implícitas:** Pueden solicitar tareas abstractas, como "quiero tomar una foto" o "quiero enviar un mensaje". Además las intenciones se resuelven en tiempo de ejecución, de forma que el sistema mirará cuantos componentes han registrado la posibilidad de ejecutar ese tipo de intención. Si encuentra varias el sistema puede preguntar al usuario el componente que prefiere utilizar.

Las intenciones las utilizaremos cuando queramos:

- lanzar una *actividad* (startActivity() y startActivityForResult())
- lanzar un *servicio* (startService())
- lanzar un *anuncio de tipo broadcast* (sendBroadcast())
- conectarnos con un *servicio* (bindService())

Cuando se crea una Intención, esta contiene información de interés para que el sistema la trate adecuadamente o para el componente que recibe la intención. Puede incluir la siguiente información:

- **Nombre del componente:** Identificamos el componente que queremos lanzar con la intención. En caso de no indicarse se utilizará otra información de la intención para obtener el componente a lanzar. A este tipo de intenciones se les conocía como intenciones explícitas.

- **Acción:** Una cadena de caracteres donde indicamos la acción a realizar (o en caso de un *Receptor de anuncios -Broadcast receiver-* la acción que tuvo lugar y queremos reportar). La clase *Intent* define una serie de constantes para acciones genéricas que son listadas a continuación.

Constante	Componente a lanzar	Acción
ACTION_CALL	Actividad	Inicia una llamada de teléfono
ACTION_MAIN	Actividad	Arranca como actividad principal de tarea
ACTION_SYNC	Actividad	Sincroniza datos en un servidor con los datos de un dispositivo móvil.
ACTION_BATTERY_LOW	Receptor de anuncios	Advertencia de batería baja.
ACTION_HEADSET_PLUG	Receptor de anuncios	Los auriculares han sido conectados o desconectados.
ACTION_SCREEN_ON	Receptor de anuncios	La pantalla es activada.

Tabla 3: Algunas acciones de las intenciones

Categoría: Complementa a la acción. Indica información adicional sobre el tipo de componente que ha de ser lanzado. El número de categorías puede ser arbitrariamente ampliado. No obstante, en la clase *Intent* se definen una serie de categorías genéricas que podemos utilizar.

Constante	Significado
CATEGORY_BROWSABLE	La actividad lanzada puede ser con seguridad invocada por el navegador para mostrar los datos referenciados por un enlace - por ejemplo, una imagen o un mensaje de correo electrónico.
CATEGORY_HOME	La actividad muestra la pantalla de inicio, la primera pantalla que ve el usuario cuando el dispositivo está encendido o cuando la tecla HOME es presionada.
CATEGORY_LAUNCHER	La actividad puede ser la actividad inicial de una tarea y se muestra en el lanzador de aplicaciones de nivel superior.
CATEGORY_PREFERENCE	La actividad a lanzar es un panel de preferencias.

Tabla 4: Algunas categorías de las intenciones

Una categoría suele utilizarse junto con una acción para aportar información adicional. Por ejemplo, indicaremos ACTION_MAIN a las actividades que pueden utilizarse como puntos de entrada de una aplicación. Indicaremos además CATEGORY_LAUNCHER para que la actividad sea mostrada en la pantalla de inicio.

3.5.3 SERVICIOS

Habrá veces que necesitemos ejecutar parte de una aplicación en segundo plano, debajo de otras actividades y además que no precise de ninguna interacción con el usuario, para ello, la opción más adecuada serán los servicios.

Un servicio se trata de una ejecución de parte de una aplicación en segundo plano la cual puede estar en ejecución indefinidamente o controlada desde una actividad.

○ *CICLO DE VIDA DE UN SERVCIO*

Un servicio tiene un ciclo de vida diferente al de una actividad. A continuación podemos ver un diagrama que ilustra el ciclo de vida de un servicio:

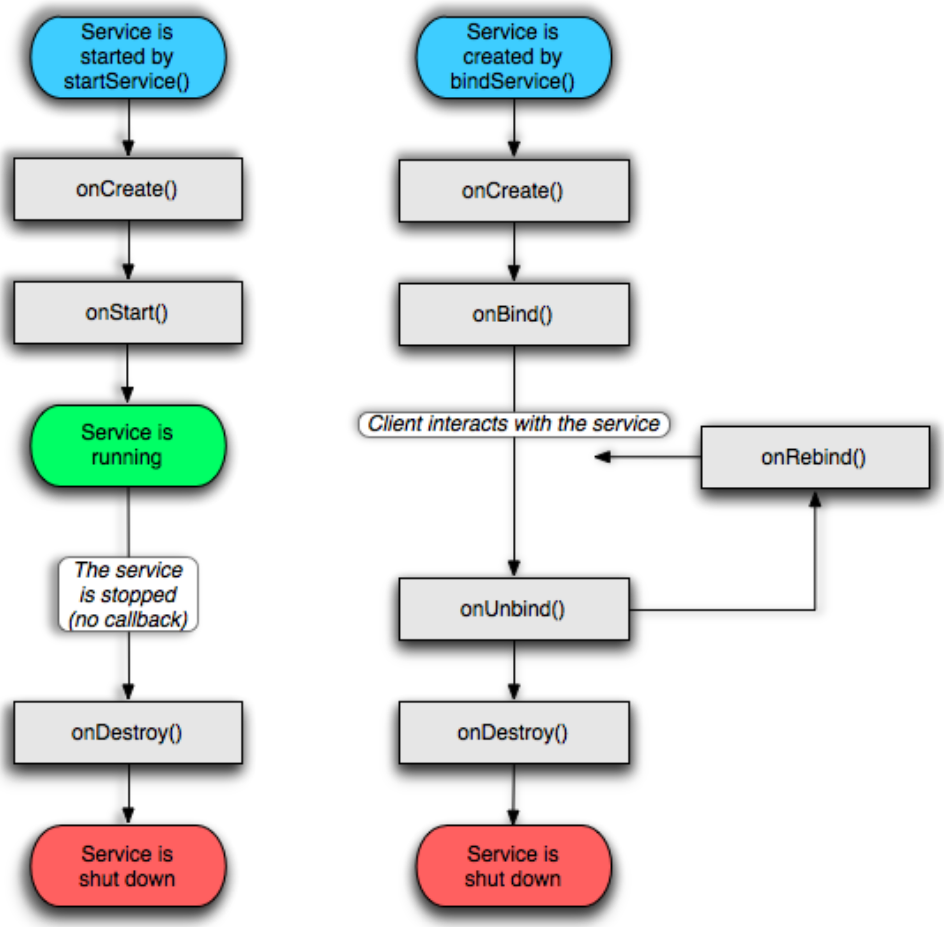


Figura 7: Ciclo de vida de un servicio

Existen 2 tipos de servicios según se hayan creado. Las funciones de estos servicios son diferentes, y por lo tanto, también su ciclo de vida.

Si el servicio es iniciado mediante el método *startService*, el sistema comenzará creándolo y llamando a su método *onCreate*. A continuación llamará a su método *onStartCommand* con los argumentos proporcionados por el cliente. El servicio continuará en ejecución hasta que sea invocado el método *stopService* o *stopSelf*.

Si se producen varias llamadas a *startService* no supondrá la creación de varios servicios, aunque sí que se realizarán múltiples llamadas a *onStartCommand*. No importa cuántas veces el servicio haya sido creado, parará con la primera invocación de *stopService* o *stopSelf*. Sin embargo, podemos utilizar el método *stopSelf(int*

startId) para asegurarnos que el servicio no parará hasta que todas las llamadas hayan sido procesadas.

Cuando se inicia un servicio para realizar alguna tarea en segundo plano, el proceso donde se ejecuta podría ser eliminado ante una situación de baja memoria. Podemos configurar la forma en que el sistema reaccionará ante esta circunstancia según el valor que devolvamos en *onStartCommand*. Existen dos modos principales: devolveremos START_STICKY si queremos que el sistema trate de crear de nuevo el servicio cuando disponga de memoria suficiente y devolveremos START_NOT_STICKY si queremos que el servicio sea creado de nuevo solo cuando llegue una nueva solicitud de creación.
Teniendo en cuenta que los servicios pueden estar largo tiempo en ejecución, el ciclo de vida del proceso que contiene nuestro servicio es un asunto de gran importancia. Conviene aclarar que en situaciones donde el sistema necesite memoria conservar un servicio siempre será menos prioritario que la actividad visible en pantalla, aunque más prioritario que otras actividades en segundo plano. Dado que el número de actividades visibles es siempre reducido, un servicio solo será eliminado en situaciones de extrema necesidad de memoria.

Podemos también utilizar *bindService(Intent servicio, ServiceConnection conexion, int flags)* para obtener una conexión persistente con un servicio. Si dicho servicio no está en ejecución, será creado (siempre que el *flag* BIND_AUTO_CREATE esté activo), llamándose al método *onCreate*, pero no se llamará a *onStartCommand*. En su lugar se llamará al método *onBind(Intent intencion)* que ha de devolver al cliente un objeto *IBinder* a través del cual se podrá establecer una comunicación entre cliente y servicio. Esta comunicación se establece por medio de una interfaz escrita en *AIDL*, que permite el intercambio de objetos entre aplicaciones que corren en procesos separados. El servicio permanecerá en ejecución tanto tiempo como la conexión esté establecida, independientemente de que se mantenga o no la referencia al objeto *IBinder*.

3.5.4 RECEPTORES DE ANUNCIOS

Un receptor de anuncios (*BroadcastReceiver*) recibe y reacciona ante anuncios globales de tipo *broadcast*, es decir, un aviso en el dispositivo del usuario (generalmente en la barra superior y en forma de icono) para informar al usuario de eventos externos o internos que han ocurrido o están ocurriendo**.** Existen muchos originados por el sistema como por ejemplo "batería baja", "llamada entrante", etc. Aunque, las aplicaciones también puede lanzar un anuncio *broadcast* (el usuario puede tener una aplicación de seguimiento GPS, por ejemplo, para correr y esta se notificará que está activa mediante un anuncio *broadcast*). Los receptores de anuncios no tienen interfaz

de usuario, aunque pueden iniciar una actividad o creara una notificación para informar al usuario.

El ciclo de vida de un *BroadcastReceiver* es muy sencillo solo dispone del método *onReceive*. De hecho un objeto *BroadcastReceiver* sólo existe durante la llamada a *onReceive*. El sistema instancia el objeto *BroadcastReceiver*, llama a este método y cuando termina destruye el objeto. Un detalle importante es que no hace falta tener la aplicación en marcha donde se define el *BroadcastReceiver* para que este se active.

El método *onReceive* es ejecutado por el hilo principal de la aplicación. Por lo tanto no puede bloquear al sistema (Ver ciclo de vida de una actividad). Si tienes que realizar una acción que puede bloquear al sistema se tendrá que lanzar un hilo secundario. Si por el contrario queremos una acción persistente en el tiempo resulta muy frecuente lanzar un servicio. Desde un *BroadcastReceiver* no se puede mostrar un cuadro de diálogo o unirse a un servicio. Para lo primero, en su lugar puedes lanzar una notificación. Para lo segundo, puede utilizar *startService* para arrancar un servicio.

Una aplicación puede registrar un receptor de anuncios de dos maneras: En *AndroidManifest.xml* y en tiempo de ejecución mediante el método *registerReceiver*.

3.5.5 PROVEEDORES DE CONTENIDO

Los proveedores de contenido (*content providers*) son una de las herramientas más constructivas que nos ofrece Android para desarrollar. Nos van a permitir tener acceso a información proporcionada por otras aplicaciones o compartirla con ellas.

Android incluye una serie de proveedores de contenidos para los tipos de datos comunes como audio, vídeo, imágenes, información de contacto personal, etc. siendo necesario para algunos de ellos tener el permiso adecuado para leer los datos que almacena, añadiendo dicho permiso en el archivo *AndroidManifest* de la aplicación.

Cuando hagamos una consulta al proveedor de contenidos, este devolverá un objeto de tipo cursor, por lo tanto la forma más práctica para almacenar la información será en una base de datos.

Utilizando términos del modelo de bases de datos, un proveedor de contenidos proporciona sus datos a través de una tabla simple, donde cada fila es un registro y cada columna es un tipo de datos con un significado particular. En la siguiente tabla veremos un ejemplo de *CallLog* y permite acceder al registro de llamadas telefónicas, la información que nos proporciona tiene la siguiente estructura:

Id	Date	Number	Duration	Type
1	13/10/14 16:10	555376548	30	INCOMING_TYPE
2	25/05/14 01:15	879542875	14	OUTGOING_TYPE
3	01/01/14 23:56	776543289	65	MISSED_TYPE
4	03/10/14 14:12	997654810	28	OUTGOING_TYPE

Tabla 5: Ejemplo de estructura de datos de un proveedor de contenidos

3.5.6 RECURSOS

La utilización de recursos en Android es un aspecto muy a tener en cuenta en el diseño de una aplicación y su uso es extremadamente sencillo, basta con añadir un fichero dentro de una carpeta determinada de nuestro proyecto. Para cada uno de los recursos que añadamos el sistema crea, de forma automática, un id de recurso dentro de la clase R.

- Tipos de recursos

Carpeta	Descripción
res/drawable R.drawable	Ficheros en bitmap (.png, .jpg o .gif). Ficheros PNG en formato Nine-patch (.9.png). Ficheros XML con descriptores gráficos (ver clase Drawable)
res/layout R.layout	Ficheros XML con los Layouts usados en la aplicación.
res/menú R.menu	Ficheros XML con la definición de menús. Podemos asignar una actividad o una vista.
res/anim R.anim	Fichero XML que permiten definir una animaciones Tween también conocidas como animaciones de vista.
res/animator R.animator	Ficheros XML que permiten modificar las propiedades de un objeto a lo largo del tiempo. Solo desde la versión 3.0.
res/xml R.xml	Otros ficheros XML, como los ficheros de preferencias
res/raw R.raw	Ficheros que se encuentran en formato binario. Por ejemplo ficheros de audio o vídeo.
res/values	Ficheros XML que definen un determinado valor para definir un color, estilo, cadena de caracteres, textos en varios idiomas…

Tabla 6: Diferentes tipos de recursos

Además de los recursos que podamos añadir, también será posible utilizar los recursos del sistema. Esto mejorará el rendimiento ya que no consumen memoria de nuestra aplicación.

Para acceder a los recursos del sistema desde código, usaremos la clase *Android.R*. Por ejemplo:

Android.R.drawable.ic_menu_edit

Para accede desde XML utilizaremos la sintaxis habitual pero comenzando con @android, por ejemplo:

@android:drawable/ic_menu_edit

3.5.7 SEGURIDAD Y PERMISOS

La seguridad es algo de suma importancia en cualquier sistema. Si se descargasen archivos maliciosos, estos podrían ser perjudiciales para los usuarios violando su intimidad si por ejemplo estos obtuviesen datos de sus contactos o correos electrónicos.

En ciertas plataformas como Windows Mobile los usuarios están desprotegidos respecto a esto, en otras como en iOS del iPhone toda aplicación ha de ser validada por Apple y pasar ciertos filtros de calidad antes de poder ser instalado en uno de sus dispositivos. Esto asegura la protección de sus usuarios pero limita a los pequeños programadores y da un poder total a Apple.

Android propone un esquema de seguridad que protege a sus usuarios sin imponer un sistema controlado por una empresa. La seguridad en Android se sostiene en los siguientes 3 pilares:

- Al estar basado en Linux se aprovechará la seguridad que este incorpora. De esta forma Android puede impedir que las aplicaciones tengan acceso directo al *hardware* o interfieran con recursos de otras aplicaciones.

- Toda aplicación ha de ser firmada con un certificado digital que identifique a su autor. La firma digital también nos garantiza que el fichero de la aplicación no ha sido modificado. Si se desea modificar la aplicación está tendrá que ser firmada de nuevo, y esto solo podrá hacerlo el propietario de la clave privada. No es preciso (ni frecuente) que el certificado digital sea firmado por una autoridad de certificación.

- Para que una aplicación pueda acceder a partes del sistema que puedan comprometer la seguridad, Android ha incorporado un sistema de permisos, de esta forma el usuario conoce los riesgos antes de instalar la aplicación.

En el siguiente esquema se pueden ver algunos de los permisos más utilizados en Android:

Nombre	Peligrosidad	Descripción
CALL_PHONE	Muy alta	Permite realizar llamadas sin la intervención del usuario.
SEND_SMS	Muy alta	Permite mandar SMS.
INTERNET	Muy alta	Permite el acceso a internet
READ_PHONE_STATE	Alta	Permite leer el estado del teléfono y el acceso al EMEI (identificador del teléfono GSM), IMSI (identificador tarjeta SIM) y al identificador de 64 bits que Google asigna a cada terminal.
WRITE_EXTERNAL_STORAGE	Alta	Permite el borrado y modificación de archivos de la memoria exterior.
READ_OWNER_DATA	Alta	Permite leer información sobre el propietario del teléfono.
READ_EXTERNAL_STORAGE	Moderada	Permite leer archivos de la memoria exterior.
ACCES_FINE_LOCATION	Moderada	Permite indicar tu ubicación mediante satélites GPS.
ACCES_COARSE_LOCATION	Moderada	Permite indicar tu ubicación basada en telefonía móvil (Cel-ID) y Wifi.
WRITE_APN_SETTINGS	Moderada	Permite la configuración del punto de acceso a la red.
MANAGE_APP_TOKENS	Moderada	Permite averiguar que aplicaciones están corriendo en segundo plano y cambiar el orden.
RECORD_AUDIO	Moderada	Permite grabar sonido desde el micrófono.
VIBRATE	Baja	Permite hacer vibrar el teléfono.
CAMERA	Baja	Permite el acceso a la cámara del teléfono.
ACCES_WIFI_STATE	Baja	Permite el acceso a información sobre las redes WiFi disponibles.

Tabla 7: Algunos permisos de Android

Para solicitar un determinado permiso en tu aplicación, no tienes más que incluir una etiqueta *<uses-permission>* en el fichero *AndroidManifest.xml* de tu aplicación. En el siguiente ejemplo se solicitan dos permisos:

```
<manifest package="org.example.mi_aplicacion" >
  …
  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
  <uses-permission android:name="android.permission.SEND_SMS"/>
</manifest>
```

3.5.8 EL ARCHIVO ANDROID MANIFEST

El archivo *Android Manifest* es el archivo más importante del proyecto y deberá estar presente en la raíz del mismo.

Dicho archivo se generará automáticamente y se podrá modificar de forma gráfica o editando el texto y contendrá información esencial sobre la aplicación.

Las principales tareas del archivo *Android Manifest* son las siguientes:

- Utiliza el nombre de paquete Java como identificador único de la aplicación.
- Describe los componentes de la aplicación: Actividades, servicios, proveedores de contenido... Para ello utiliza el nombre de las clases que implementan cada uno de estos componentes y publica sus capacidades. Esto permite al sistema operativo conocer que componentes tiene y bajo qué condiciones pueden ser lanzados.
- Especifica que permisos tiene la aplicación para acceder a partes protegidas del API que proporciona el sistema Android.
- Declara la mínima versión del sistema operativo en el que funcionará la aplicación.
- Indica las librerías que utiliza el proyecto y por lo tanto tienen que ser empaquetadas al crear la aplicación.
- Permite declarar una clase 'Instrumentation' que sirve para monitorizar la interacción de la aplicación con el sistema. Esta declaración solo estará presente mientras se desarrolla y prueba la aplicación. Ya que será borrada antes de que la aplicación se vaya a publicar.

A continuación se mostrará un ejemplo sencillo de cómo está estructurado este archivo:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.llamarportelefeno"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.llamarportelefeno.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
</manifest>
```

Figura 8: Ejemplo del archivo Android Manifest

CAPÍTULO 4
DESARROLLO PARA ANDROID

4.1 INTRODUCCIÓN

Una vez se ha expuesto que es, de dónde surgió el sistema operativo Android y habiendo visto que es una aplicación desarrollada para Android y los diferentes elementos que la componen, en el presente capítulo se expondrá cuáles son las diferentes herramientas de desarrollo y de trabajo necesarias para el desarrollo de aplicaciones. Además también veremos cómo probar y testear nuestras aplicaciones para finalmente publicarlas en *Google Play*.

4.2 ENTORNO DE TRABAJO

Para poder comenzar el desarrollo de una aplicación para Android primeramente necesitaremos saber dónde hacerlo, para ello tenemos a nuestra disposición una serie de herramientas y librerías para conseguir dicho fin.

Primeramente es recomendable instalar el *IDE* de *Eclipse*, podríamos trabajar con otros pero el *plug-in ADT* está desarrollado principalmente para este *IDE*, por lo tanto la mejor opción para evitar posibles fallos o problemas de compatibilidad del *plug-in* es usar este entorno de desarrollo.

A continuación instalaremos el SDK de Android. En la actualidad existe un paquete de descarga denominado *SDK ADT Bundle* que combina el SDK de Android con la plataforma de desarrollo *Eclipse* y el *plug-in ADT*, el cual permite el uso del SDK de Android desde *Eclipse*. Este *plug-in* permite crear y depurar aplicaciones de forma sencilla añadiendo una serie de capacidades a Eclipse que mejoran la experiencia de desarrollo de la siguiente forma:

- Proporciona un asistente para nuevos proyectos, creando todos los archivos de base necesarios para una nueva aplicación Android.
- Automatiza y simplifica el proceso de desarrollo de las aplicaciones
- Proporciona un editor de código que sirve de ayuda para escribir código XML válido para los archivos de recursos, interfaces de usuario o el archivo *Android Manifest.*
- Permite exportar el proyecto como un paquete APK y firmarlo digitalmente para ser distribuido a los usuarios.

Una vez realizado esto, debemos lanzar el programa "SDK Manager" que gestiona la instalación de las herramientas Android en nuestro equipo.

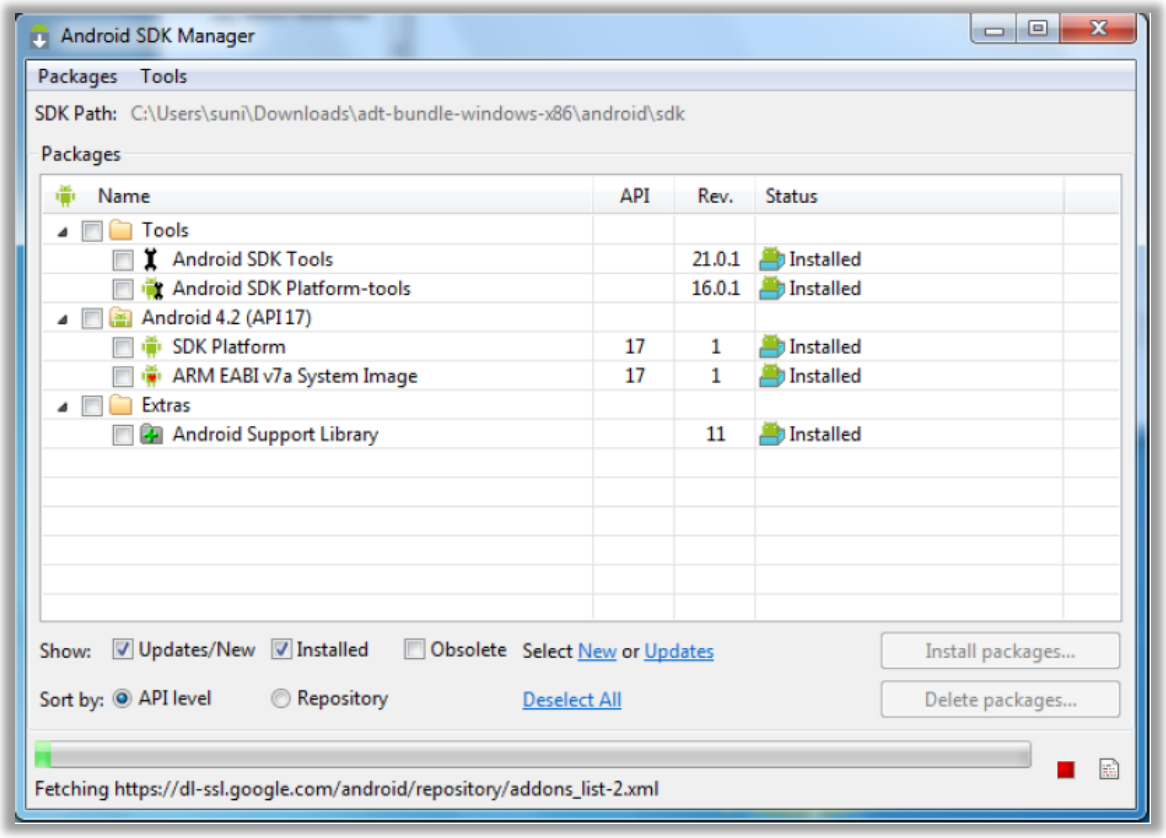


Figura 9: Android SDK Manager

Para empezar a desarrollar una aplicación es imprescindible tener instalada una plataforma de desarrollo deseada.

4.3 CREACIÓN DE UN PROYECTO

Una vez tenemos configurado correctamente el entorno de trabajo, nos dispondremos a crear un nuevo proyecto para empezar a desarrollar. El *plug-in ADT* que hemos instalado nos hará mucho más sencilla esta tarea ya que incluye herramientas para la creación de dichos proyectos con lo que rápidamente podremos empezar a trabajar.

Para la creación de un nuevo proyecto, tendremos que, desde el menú ***File*** elegir la opción ***New*** y después ***Android Project Application.***

A continuación tendremos que indicar una serie de parámetros necesarios para la creación y configuración del proyecto:

ELEMENTO	DESCRIPCIÓN
Application Name	Nombre de la aplicación mostrada en el dispositivo.
Project Name	Nombre del proyecto que estamos creando.
Package Name	Indica el espacio de nombres para los paquetes del código fuente de la aplicación (siguiendo las mismas convenciones que en el lenguaje de programación Java).
Minimum Required SDK	Indica la versión mínima de Android que debe tener instalado el terminal para poder ejecutar la aplicación.
Target SDK	Indica para qué versión está pensada la aplicación. No es importante y sólo influye en determinadas opciones de los dispositivos donde la ejecutemos.
Compile With	Indica a Eclipse con qué versión del SDK debe compilarse.
Theme	Indica el tema por defecto de la aplicación.

Tabla 8: Parámetros de configuración de un proyecto

A continuación indicaremos una imagen para nuestra aplicación e también un nombre para nuestra actividad principal.

Una vez hemos finalizado la creación de nuestro proyecto, se habrán creado una serie de carpetas y archivos que se explicarán a continuación:

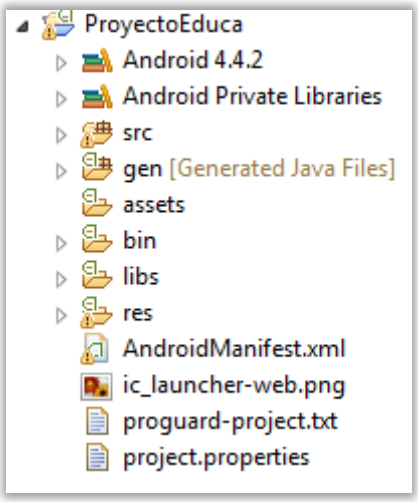


Figura 10: Ficheros que componen un proyecto

ELEMENTO	DESCRIPCIÓN
<Versión de Android> (por ejemplo Android 4.4.2)	Archivo JAR que contiene las librerías de Android y cuya versión se especifica en el momento de creación del proyecto.
SRC	Código fuente de la aplicación.
gen	Contiene los archivos Java creados automáticamente por el plug-in ADT como por ejemplo el archivo R.java.
assets	Carpeta para almacenar posibles recursos.
bin	Archivos generados automáticamente.
libs	Contiene las librerías que se enlazarán al proyecto.
res	Directorio en el que se almacenan los recursos de una aplicación como imágenes, texto, etc.
AndroidManifest.xml	El archivo de manifiesto del proyecto.
ic_launcher-web.png	Imagen de la aplicación.
project.properties	Contiene los parámetros de configuración del proyecto.

Tabla 9: Descripción de los ficheros que componen un proyecto

4.4 EJEMPLO DE UNA APLICACIÓN SENCILLA

Habiendo ya creado un proyecto y sabiendo que partes lo componen, nos disponemos ahora al desarrollo de una aplicación sencilla a modo de ejemplo.

En principio se podría desarrollar el típico programa “Hola Mundo”, pero necesitamos exponer ciertos conceptos antes de hacer frente a la aplicación que origina este proyecto. Por eso diseñaremos un programa el cual hace llamadas telefónicas.

Primeramente comenzaremos definiendo la interfaz de usuario de la actividad principal (este programa sólo dispondrá de una actividad, por lo tanto, sólo esta interfaz de usuario tendrá relación con esta actividad. Si el programa tuviese más de una actividad deberíamos hacer una interfaz de usuario por cada actividad), esto se podrá realizar de dos formas distintas. La primera de ellas es utilizando las

herramientas gráficas que nos proporciona el *ADT*. Esto puede ser muy útil y rápido ya que simplemente arrastraremos los elementos que deseamos a nuestra actividad.

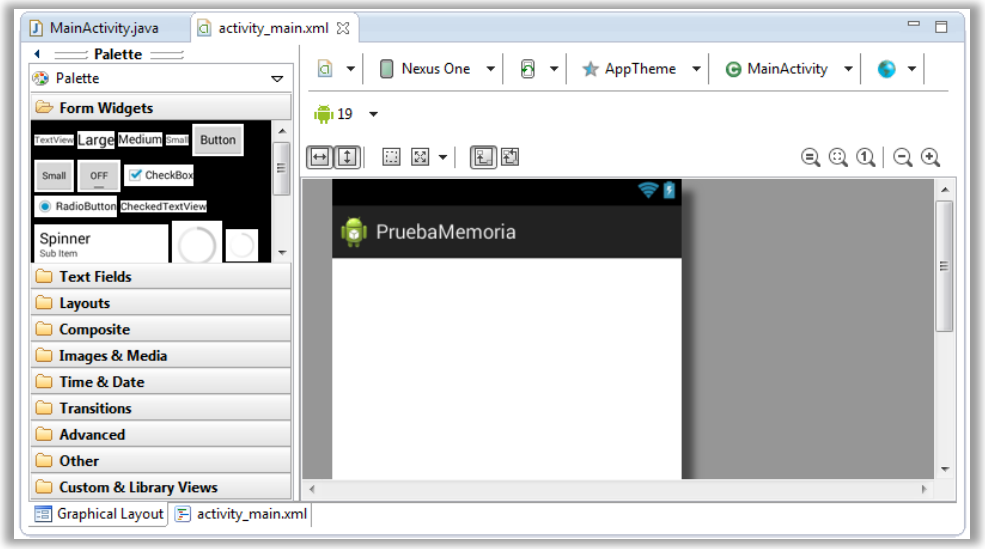


Figura 11: Entorno grafico de Eclipse

Mediante esta opción también será posible previsualizar el diseño de la aplicación mediante:

- Dispositivos reales (en este caso, trabajamos con el *Nexus One*) o con multitud de tamaños de pantalla diferentes.
- Diferente orientación (apaisada o *landscape*).
- Diferentes temas para la actividad.

Otras veces necesitaremos mayor precisión o realizar diseños más complejos y resultará mucho más cómodo trabajar directamente editando el archivo XML

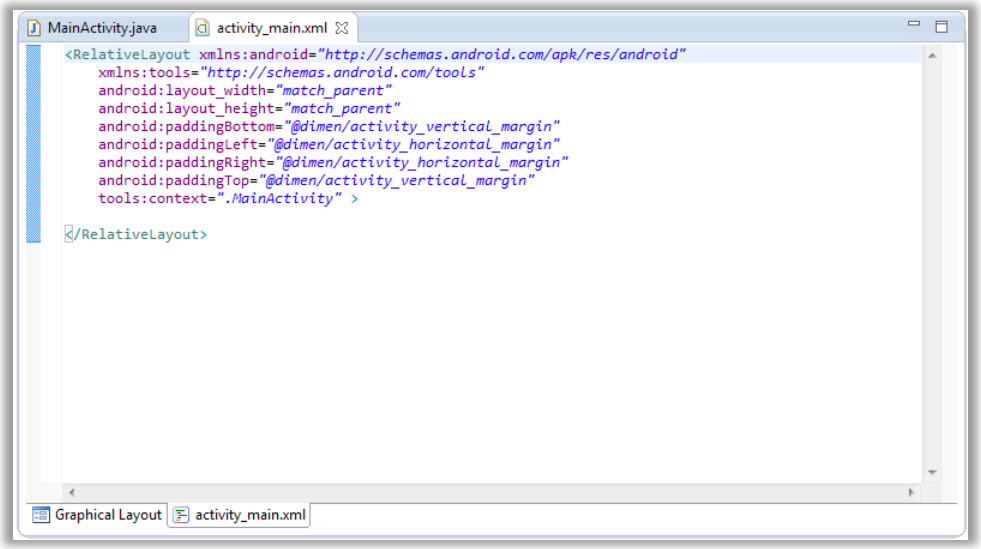


Figura 12: Editor de texto XML de Eclipse

Para este sencillo ejemplo se ha diseñado la siguiente interfaz de usuario:

Figura 13: Interfaz de usuario de ejemplo

El código XML relacionada con la interfaz de usuario que acabamos de mostrar será el siguiente:

Figura 14: Archivo XML de la interfaz de usuario de ejemplo

A continuación se explicará cómo se ha diseñado esta interfaz de usuario.

1. El *Layout* utilizado será del tipo *LinearLayout*. Esto quiere decir que las vistas o elementos de la interfaz serás colocados de forma ordenada uno sucediendo al siguiente, en nuestro caso de forma vertical. Podríamos haber utilizado otra diferente, pero por la sencillez de nuestra aplicación no ha sido necesario. Mediante los atributos *layout_height* y *layout_width* se indica que el espacio que se quiere ocupar. En nuestro caso al haberle dado el valor *match_parent* su tamaño será de toda la pantalla, tanto a lo alto como a lo ancho. Por último se ha accedido a los recursos para establecer un fondo de pantalla mediante una imagen depositada en la carpeta *drawable* de los recursos.

2. Ahora ya se podrán añadir las vistas que necesitamos. En este caso, un texto explicando qué hacer. Se podrán editar sus atributos como su color o su tamaño además se establecerá la cadena de texto utilizando los recursos. Esto supondrá tener todas las cadenas de texto en una carpeta determinada por lo que nos facilitará la edición y traducción (si fuera necesario) de los mismos.

3. A continuación se añadirá un recuadro para texto editable, donde introduciremos, por defecto, un número de teléfono. El tamaño de la vista será ajustada a lo alto y expandida a lo ancho.

4. Por último añadimos un botón, ajustando su tamaño igual a la vista anterior, usando los recursos para establecer un texto y asignándole un método que veremos más adelante.

En principio el diseño de una interfaz de usuario, como podemos ver, puede ser muy sencillo. Sólo necesitamos un *Layout*, es decir, una especie de contenedor y en dicho elemento depositar las vistas que necesitemos editando sus atributos para que quede a nuestro gusto.

Ahora vamos a ver el archivo JAVA asociado a este interfaz de usuario:

MainActivity.java

```
package com.example.llamarportelefeno;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity {
    EditText editTexto;
    String telefono;
    String mensajeAlerta="El teléfono introducirdo es INCORRECTO, vuelve a intentarlo";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editTexto = (EditText)findViewById(R.id.editText1);
    }

    public void llamadaTelefono (View view){
        telefono = editTexto.getText().toString();
        if (telefono.length()<3){
            Toast.makeText(this, mensajeAlerta, Toast.LENGTH_SHORT).show();
        }
        else{
            Intent intent = new Intent(Intent.ACTION_CALL,Uri.parse("tel:"+telefono));
            startActivity(intent);
        }
    }
}
```

Figura 15: Archivo JAVA de la aplicación de ejemplo

Dentro de la clase principal, hemos definido dos métodos:

- El primero de ellos se trata del método *onCreate* y sería el punto de partida de la actividad y la puerta de entrada a la aplicación. En él hemos definido qué *Layout* le corresponde esta actividad mediante el método *setContentView* e indicamos que el *EditText* que hemos definido en esta clase se trata del *EditText* del archivo XML "encontrándolo" con el método *findViewById* para poder trabajar con él desde el archivo JAVA.
  El método *onCreate* corresponde a uno de los métodos que engloban el ciclo de vida de la actividad, en principio en esta aplicación no necesitamos trabajar con el resto de métodos (*onStart*, *onResume*, *onRestart*, *onPause*, *onStop* y *onDestroy*) debido a la sencillez de la misma, dichos métodos los veremos más adelante cuando se explique el proyecto principal por el que se ha redactado este documento.

- El segundo método realiza la llamada. Para ello, realizamos una intención de tipo llamada telefónica pasándole el número que hemos introducido en el *EditText* anterior.

Antes de testear esta aplicación de ejemplo necesitamos indicar en el archivo *Android Manifest* que queremos permitir hacer llamadas telefónicas, para ello lo haremos desde la pestaña "Permissions" añadiéndola mediante el botón "Add":

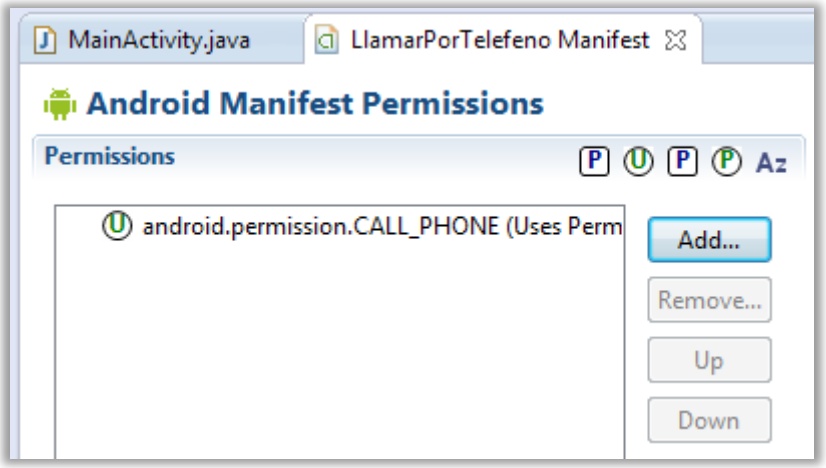


Figura 16: Pestaña de permisos de Android Manifest

O directamente en la pestaña *AndroidManifest.xml* escribiendo:

<uses-permission Android:name="android.permission.CALL_PHONE"/>

4.5 DISPOSITIVO VIRTUAL

El siguiente paso será el de ejecutar una aplicación. Para ello necesitaremos un terminal Android configurado y preparado para esta tarea, si no disponemos de uno, recurriremos a la creación de un dispositivo virtual AVD (*Android Virtual Device*). Para ello, desde el programa *SDK Manager*, debemos abrir el programa *ADV Manager (Android Virtual Device Manager)* y crear una nueva máquina virtual.

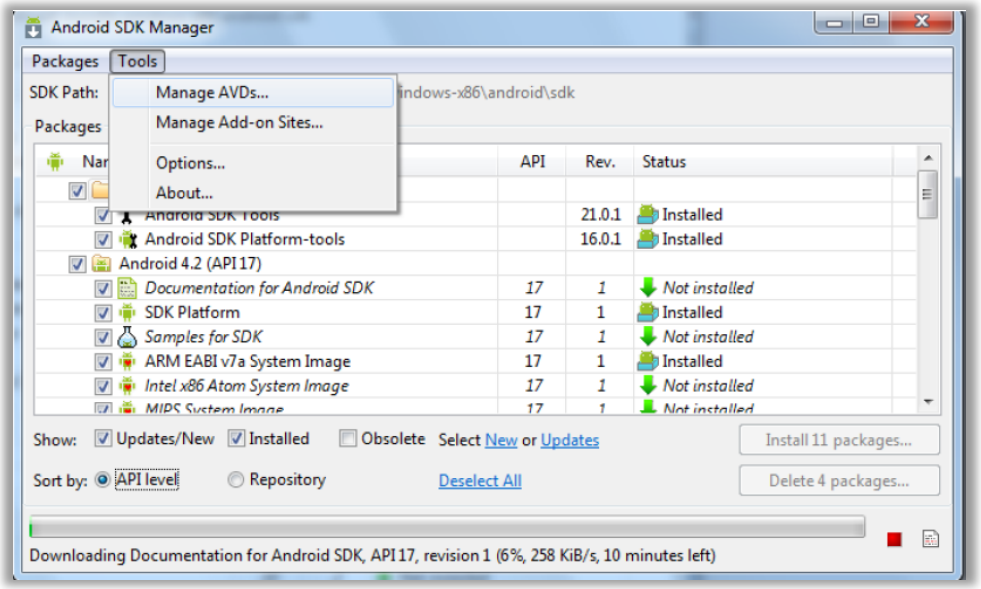


Figura 17: Creación del dispositivo virtual (I)

A continuación pulsamos sobre el botón “new” para crear la máquina.

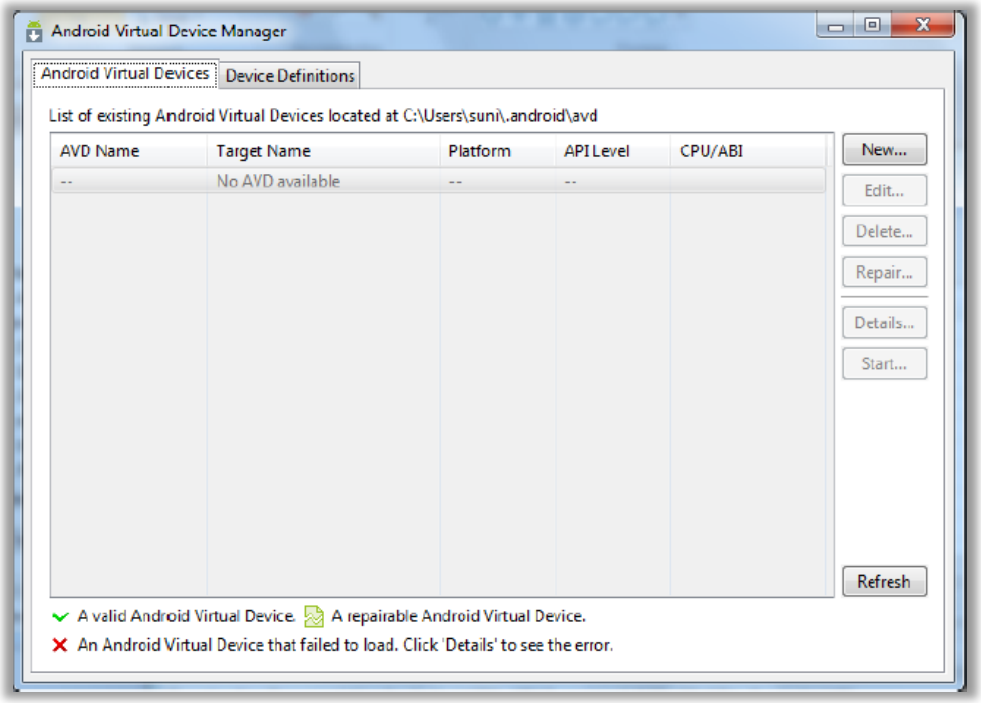


Figura 18: Creación del dispositivo virtual (II)

Una vez hecho esto, nos saldrán las siguientes opciones de configuración:

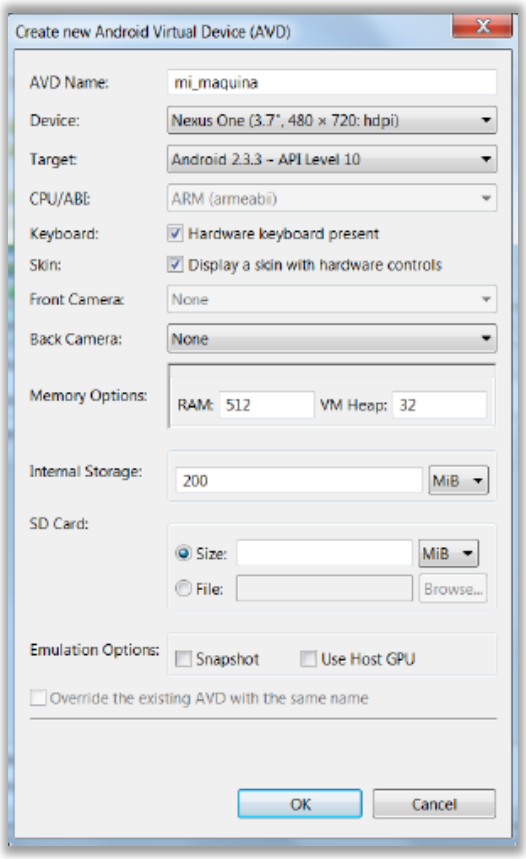


Figura 19: Opciones de configuración del dispositivo virtual

Nombre	Descripción
AVD Name	Nombre de nuestra máquina virtual.
Device	Terminales reales para simular.
Target	Versión de Android sobre el que correrá nuestro dispositivo.
CPU/ABI	Tipo de CPU del dispositivo virtual.
Keyboard	Habilitar un teclado físico.
Skin	Configuración de la pantalla. Podemos seleccionar el tipo de pantalla automáticamente o ingresar la resolución.
Front Camera	Posibilidad de usar la webcam del PC como cámara frontal.
Back Camera	Posibilidad de usar la webcam del PC como cámara trasera.
Memory Options	Establecer memoria del dispositivo.
Internal Storage	Establecer la memoria interna del dispositivo.
SD Card	Configurar una SD Card virtual.
Emulation Options	Snapshot: Acelera el arranque del simulador. Use host GPU: Usa la GPU real del PC para ayudar en las simulaciones son gráficos.

Tabla 10: Descripción de las opciones de configuración del AVD

4.6 EJECUTAR UNA APLICACIÓN

Como ya hemos comentado, una de las opciones para ejecutar y testear nuestras aplicaciones de Android es a través de un emulador o dispositivo virtual.

Una vez el dispositivo virtual ha sido creado, para ejecutar una aplicación simplemente se pulsaría sobre el botón ***Run*** de la barra de herramientas de Eclipse y se lanzaría la aplicación seleccionada sobre el emulador.

Antes de ser lanzada, *Eclipse* realizará las siguientes opciones:

1. Compilar el proyecto.
2. Crear una configuración de ejecución por defecto para el proyecto.
3. Instalará y arrancará la aplicación en el dispositivo virtual.

Una vez finalizados estos pasos, el dispositivo virtual aparecerá de la siguiente forma:

Figura 20: Aplicación de ejemplo ejecutándose en dispositivo virtual

Si miramos al emulador nos encontramos con dos zonas claramente diferenciadas. En la parte izquierda se sitúa nuestra aplicación y en la parte derecha ciertos botones que simulan parte del funcionamiento de un teléfono real, como subir o bajar el volumen, botón de menú etc. No aparece un teclado físico, ya que este dispositivo virtual no ha sido configurado para que ello.

Ahora ya podremos testear y comprobar el funcionamiento de la aplicación.

1. Introducir un número en el *EditText*.

Figura 21: Testeo de la aplicación de ejemplo (I)

2. Pulsar el botón para lanzar la intención de llamada telefónica.

Figura 22: Testeo de la aplicación de ejemplo (II)

3. Al no haber ningún problema, lanza la intención y realiza la llamada.

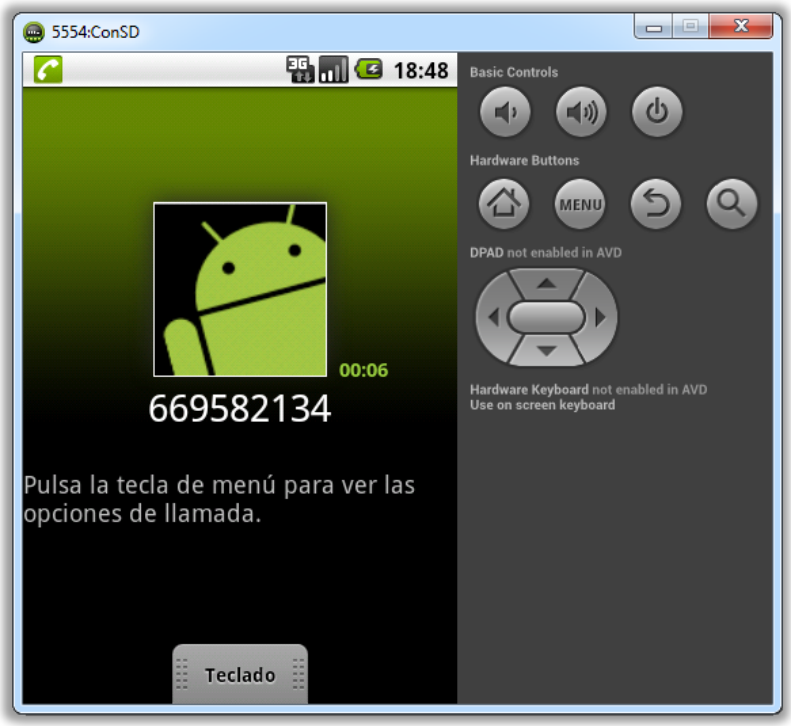


Figura 23: Testeo de la aplicación de ejemplo (III)

Otra forma de ejecutar la aplicación sería lanzarla en modo depuración, esto hará que el emulador se inicie en modo "Waiting For Debugger", que le indica al depurador que está siendo enlazado con el dispositivo virtual.

Este modo de trabajo es muy útil para:

- Ver los diferentes procesos e hilos de ejecución.
- Ver las variables y puntos de ruptura.
- Poder hacer un seguimiento del código paso a paso.

Además a través de consola se podrán ver los mensajes referentes a la ejecución de la aplicación y sobre el estado del dispositivo virtual y a través del *LogCat* podremos hacer un seguimiento de los eventos producidos en el dispositivo virtual.

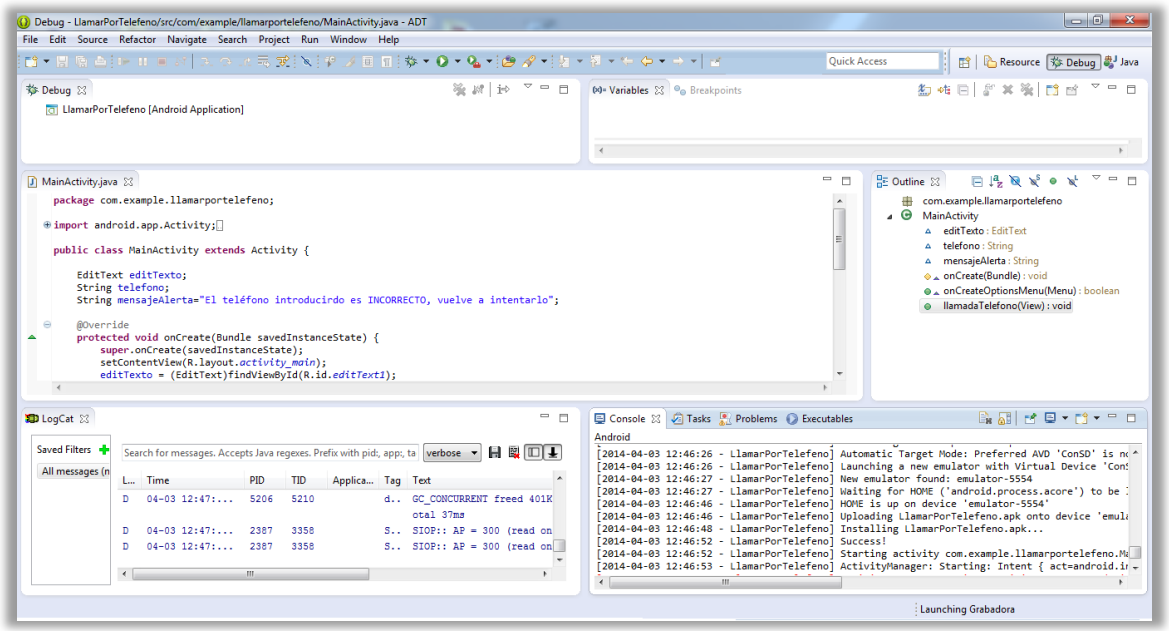


Figura 24: Modo depuración de Eclipse

Por último, si disponemos de un dispositivo compatible, podremos probar nuestra aplicación en él. Para ello será necesario haber configurado dicho dispositivo para tal fin. Esta tarea es muy sencilla, solamente necesitamos activar la opción de "Depuración USB" en el menú de desarrollo del terminal.

4.7 DISTRIBUCIÓN DE LA APLICACIÓN

Una vez finalizada la aplicación, sería muy interesante distribuirla, en este punto se explicará cómo.

Una vez la aplicación ha sido testeada y libre de fallos necesitamos firmarla con un certificado digital si queremos distribuirla, para ello, tenemos dos formas: A través de internet o a través de la plataforma de distribución digital de Google, *Google Play*.

Como hemos dicho, las aplicaciones han de ser firmadas con un certificado digital para ser instaladas en un dispositivo Android. Hasta ahora, mientras desarrollábamos y testeábamos, podíamos ejecutar las aplicaciones en nuestro terminal usando un certificado digital generado por el *SDK*, pero esto no es válido, a la hora de la distribución, necesitamos una firma digital y esta permite identificar al autor e impide que la aplicación pueda ser manipulada, además cabe decir que Google exige un periodo de 25 años de validez del certificado para poder publicar la aplicación en *Google Play*.

A continuación se explicarán los pasos necesarios crear el certificado digital de forma sencilla desde *Eclipse*:

1) Acceder a la opción *File > Export > Android > Export Android Aplication*

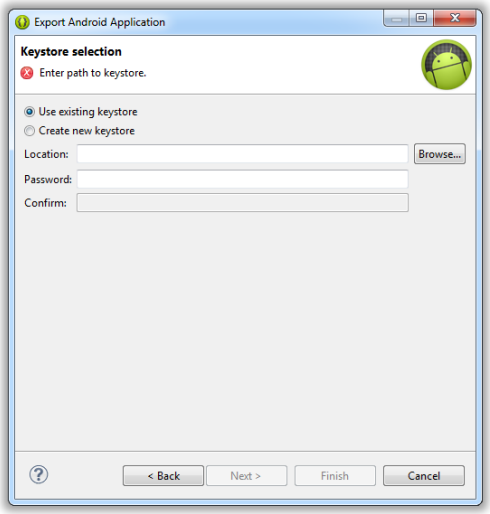


Figura 25: Creación del certificado digital (I)

2) A continuación te pedirá información para crear una nueva clave que será incluida en el fichero anterior. Recordar que necesita un periodo mínimo de validez de 25 años. El resto de campos no es necesario rellenarlos todos.

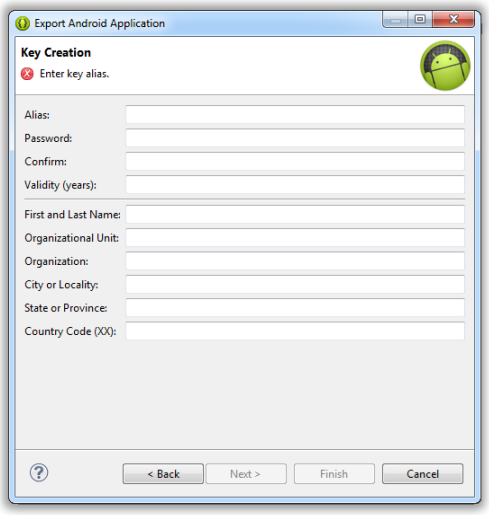


Figura 26: Creación del certificado digital (II)

Una vez hecho esto, tendremos nuestro fichero *.apk* que será el que distribuyamos a nuestro público potencial. Para ello tenemos dos opciones: A través de internet

(opción poco recomendable para usuarios nóveles por las dificultades que conlleva instalar la aplicación de esta forma) o a través de *Google Play* (muy recomendable por su sencillez), para ello seguiremos los siguientes pasos:

1) Acceder a *http://play.google.com/apps/publish/* y abonaremos 25$ una sola vez para poder empezar a publicar.
2) Pulsaremos sobre *Add new application*, introduciremos el nombre de nuestra aplicación en los diferentes idiomas que serán publicados y subiremos el archivo *apk*.
3) Añadiremos información e imágenes para que, de cara a posibles usuarios, nuestra aplicación resulte mucho más atractiva.
4) Por último seleccionaremos si queremos que nuestra aplicación sea gratuita o de pago.

Tras unos minutos nuestra aplicación estará lista para cualquier usuario.

CAPÍTULO 5
LA APLICACIÓN "PROYECTO EDUCA"

5.1 INTRODUCCIÓN

Una vez se ha expuesto qué es y cómo se desarrollan aplicaciones para el sistema operativo Android y se han asimilado estos conocimientos, es el momento de plasmar lo aprendido en la aplicación que da origen a este proyecto.

Primeramente se comentará por qué se ha elegido esta aplicación como proyecto fin de carrera. A continuación se expondrá cómo ha sido diseñada y finalmente se detallarán los aspectos más importantes de cómo se ha implementado.

5.2 ¿QUÉ APLICACIÓN DESARROLLAR?

Después de darle vueltas a esta idea, yo quería plasmar todos, o gran parte, de los conocimientos que había aprendido en una aplicación, pero que además esa aplicación llegara a mucha gente y fuese útil.

Tenía una idea en la cabeza, una aplicación interactiva: ver, tocar, escuchar y rápidamente pensé en algún tipo de juego para niños.

Habiendo hecho un pequeño estudio sobre este tipo de aplicaciones que actualmente hay disponibles de este tipo, me di cuenta que eran muy limitadas y además no suponían un reto al carecer de un "premio" por hacerlo mejor en cada partida jugada. De todo esto surgió la idea de esta aplicación, una aplicación que incluyese diferentes juegos donde se aprendiese un poco de todo y además te puntuasen por ello, con el consecuente reto de querer hacerlo cada vez mejor para superarte a ti mismo o a los demás.

5.3 ESTRUCTURA DE LAS ACTIVIDADES

El público potencial al que va dirigido esta aplicación es un público mayoritariamente infantil (por supuesto, es muy aconsejable que los padres estén al lado de ellos para orientarles si en algún momento se pierden), por lo tanto se ha puesto mucho énfasis en diseñar una interfaz de usuario muy vistosa y colorida, algo que atraiga la atención del usuario sin olvidar lo más importante, que sea rápida e intuitiva. El usuario debe ser capaz, simplemente con mirar o escuchar de orientarse y saber dónde está en cada momento.

En la siguiente imagen podemos ver la estructura jerárquica de las actividades que componen la aplicación:

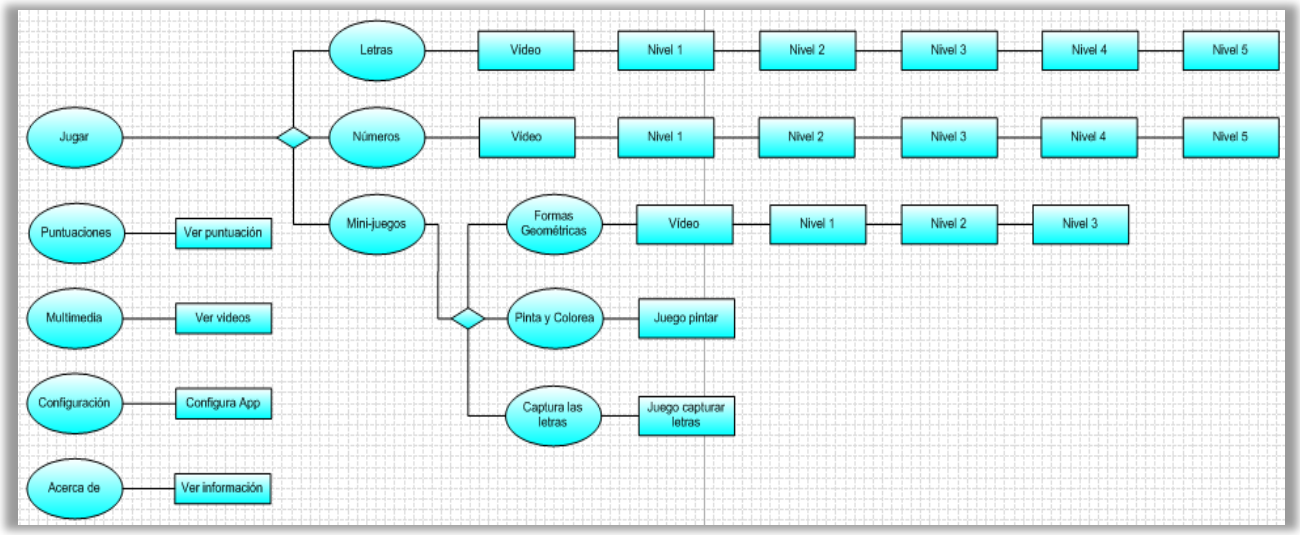


Figura 27: Jerarquía de actividades de la App

A continuación se realizará una breve descripción de cada una de las actividades y se mostrará un pequeño mapa a base de capturas de pantalla del dispositivo de testeo de cómo han sido diseñados las actividades de los diferentes juegos. Explicar más en detalle el diseño de las actividades una por una y comentar su archivo XML es una tarea que se verá en el siguiente punto.

Una vez se ha pulsado el icono de la aplicación en nuestro dispositivo, esta arrancará mediante una pequeña animación en forma de presentación que consiste, primeramente, en la aparición y giro de 720˚ del título principal, y a continuación la aparición uno por uno de los botones que dan acceso a los diferentes juegos, o menús.

Figura 28: Animación inicial

Acabada la animación inicial, se mostrará definitivamente la primera actividad, el menú principal.

Figura 29: Actividad del menú principal

Jugar: Mediante este botón se accede a los diferentes juegos que componen la aplicación.

Puntuaciones: Se consultan las puntuaciones de los juegos obtenidas por los usuarios.

Multimedia: Recopilación del material multimedia que aparece a los largo de los juegos para una visualización rápida.

Configuración: Acceso al menú de configuración para cambiar diferentes parámetros a gusto del usuario.

Acerca de: Información sobre la aplicación y el autor de la misma.

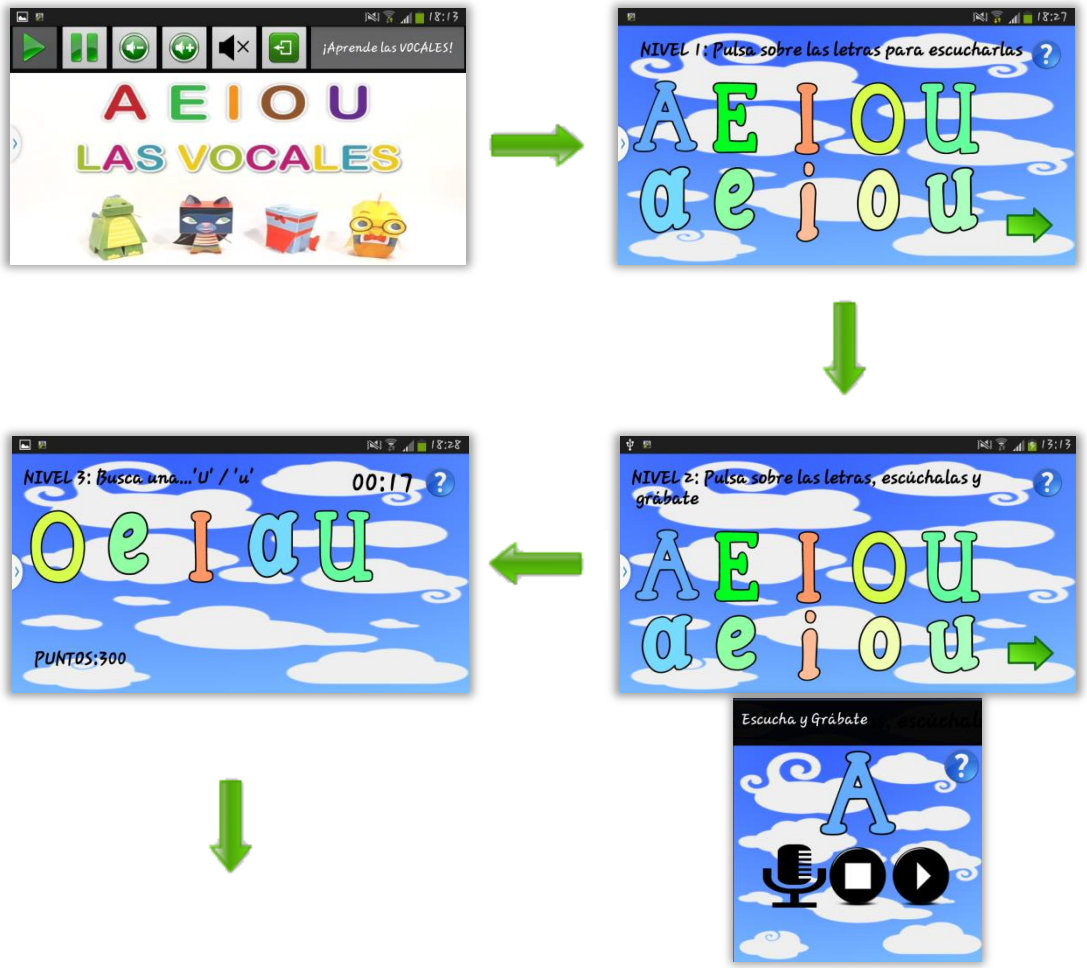
Salir: Cierra la aplicación.

Pulsando el botón *Jugar* se accederá a un menú donde se podrá seleccionar 3 tipos de juegos: letras, números y mini-juegos.

Figura 30: Actividad del menú de juegos

Letras: Comenzará con un vídeo educativo y a continuación habrán 5 niveles donde se deberán realizar diferentes ejercicios que, una vez finalizados, se dará la posibilidad al jugador de guardar los puntos conseguidos a través de un nombre para, en cualquier momento, poder consultarlo o comparar las diferentes partidas registradas.

A continuación se muestra la estructura de actividades que sigue este juego:



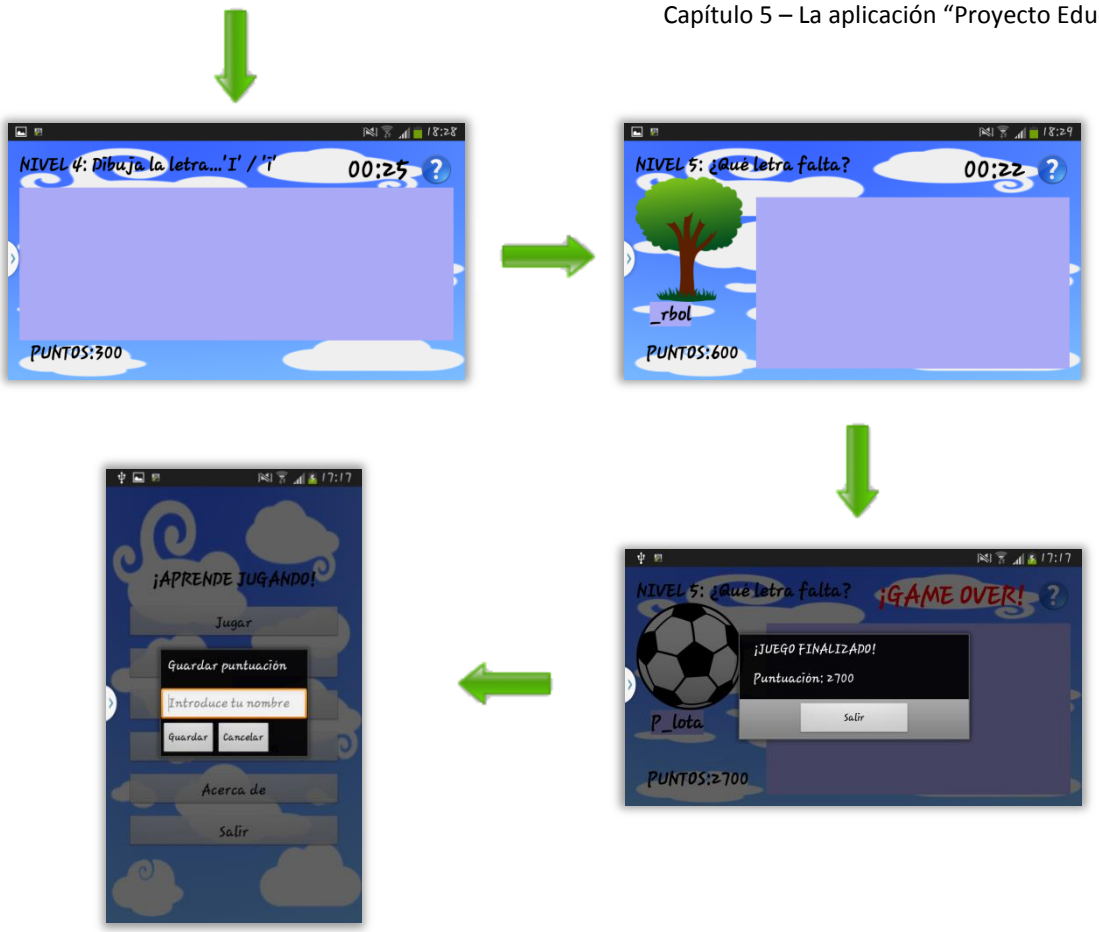


Figura 31: Estructura de actividades del juego de letras

Números: La estructura de actividades de este juego es muy similar a la anterior. Empezará con un video educativo para continuar con 5 niveles donde se deberán realizar diferentes ejercicios para finalizar registrando nuestros puntos.

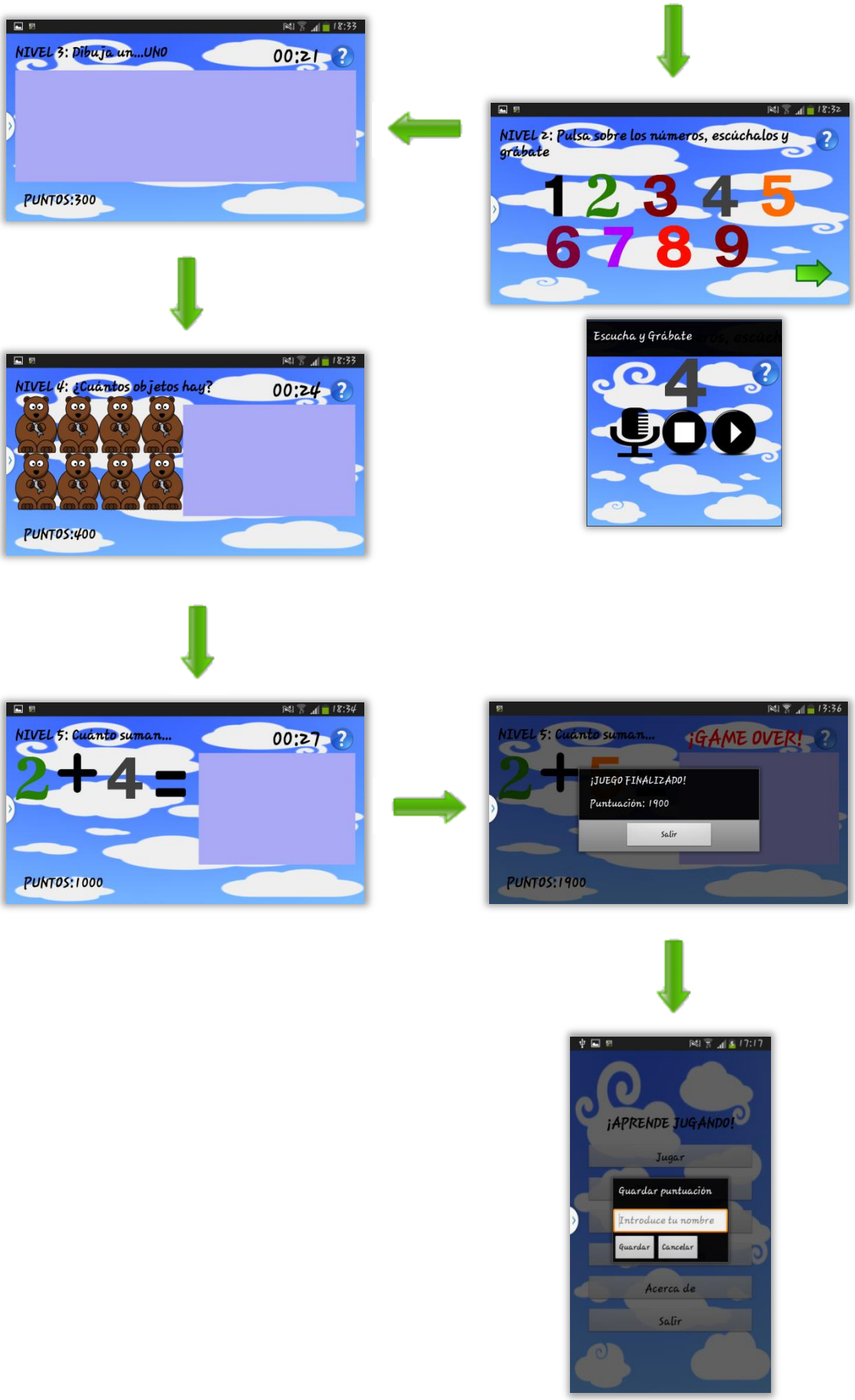


Figura 32: Estructura de actividades del juego de números

Mini-Juegos: Nos encontraremos con 3 tipos de mini-juegos: El primero es una adaptación del clásico juego de recreativa "Asteroides", en el segundo se podrá pintar con los dedos y el último consiste en diferentes niveles (al igual que en los juegos anteriores) con el que se trabajará con formas geométricas.

Figura 33: Actividad del menú de mini-juegos

La estructura que se ha diseñado para este último juego sigue los patrones de los juegos ya explicados, comenzando por un vídeo educativo para continuar superando diferentes niveles e ir consiguiendo puntos.

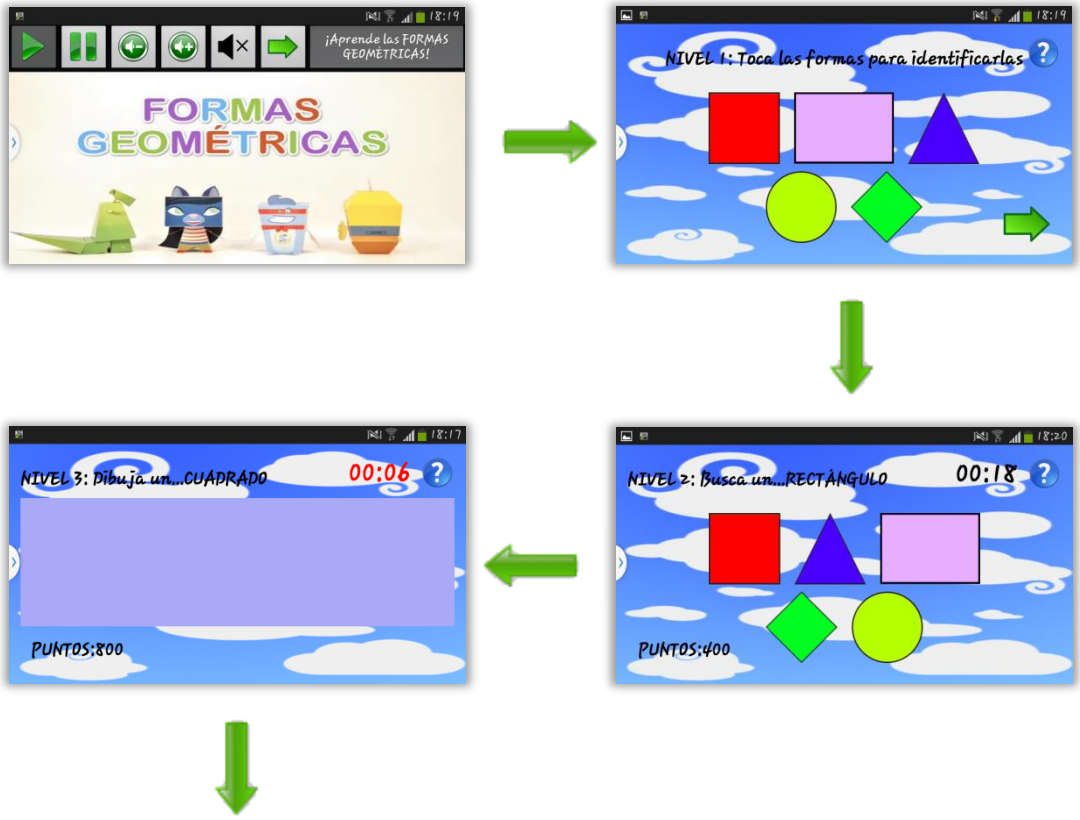


Figura 34: Estructura de actividades del juego de formas geométricas

5.4 DISEÑO DE LAS ACTIVIDADES Y ARCHIVOS XML

A continuación se explicará el diseño de cada una de las actividades por separado y se comentará brevemente lo más destacable de su archivo XML, pero antes es muy interesante comentar que para aplicar atributos a las vistas se han utilizado hojas de estilo.

○ *¿QUÉ SON LAS HOJAS DE ESTILO?*

Hay muchas similitudes entre el diseño HTML y el diseño de actividades en Android, en ambos casos se utiliza un lenguaje de programación para realizar diseños independientes del tamaño de pantalla donde vamos a visualizar los resultados. En el diseño web las hojas de estilo en cascada o CSS tienen una gran utilidad ya que permiten crear un patrón de diseño y aplicarlo a varias páginas, con Android pasa algo similar. Esta herramienta es conocida como *estilos* y te permitirá crear patrones de estilo que podrán ser utilizados en cualquier parte de la aplicación con lo que se ahorrará mucho trabajo consiguiendo además un diseño homogéneo en toda la aplicación.

A continuación se muestra un ejemplo del uso de esta herramienta para implementarla en el *Layout*:

Figura 35: Ejemplo de aplicación de estilo en una vista

La hoja de estilo se creará en el directorio *res/values/styles.xml* con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="MiEstilo"
     parent="@android:style/TextAppearance.Medium">
       <item name="android:layout_width">match_parent</item>
       <item name="android:layout_height">wrap_content</item>
       <item name="android:textColor">#00FF00</item>
       <item name="android:typeface">monospace</item>
   </style>
</resources>
```

Figura 36: Ejemplo del archivo *styles.xml*

En la siguiente figura se podrá ver parte de la hoja de estilo de este proyecto:

```
<style name="estiloTextoTituloMenuPrincipal"
    parent="@android:style/TextAppearance.Medium">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textSize">25sp</item>
    <item name="android:gravity">center</item>
    <item name="android:layout_marginBottom">10dip</item>
    <item name="android:layout_marginTop">10dip</item>
    <item name="android:textColor">@color/colorTextoMenuPricipal</item>
</style>

<style name="estiloTextoTituloMenuPrincipal.Botones">
    <item name="android:textSize">20sp</item>
    <item name="android:layout_marginBottom">1dip</item>
    <item name="android:paddingTop">10dip</item>
    <item name="android:paddingBottom">10dip</item>
    <item name="android:textColor">@color/colorTextoBotonesMenuPrincipal</item>
</style>

<style name="estiloTextoTituloMenuPrincipalLarge"
    parent="@android:style/TextAppearance.Medium">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textSize">50sp</item>
    <item name="android:gravity">center</item>
    <item name="android:layout_marginBottom">10dip</item>
    <item name="android:layout_marginTop">10dip</item>
    <item name="android:textColor">@color/colorTextoMenuPricipal</item>
</style>

<style name="estiloTextoTituloMenuPrincipalLarge.Botones">
    <item name="android:textSize">35sp</item>
    <item name="android:layout_marginBottom">1dip</item>
    <item name="android:paddingTop">10dip</item>
    <item name="android:paddingBottom">10dip</item>
    <item name="android:textColor">@color/colorTextoBotonesMenuPrincipal</item>
```

Figura 37: Parte de hoja de estilo del proyecto

5.4.1 MENÚ PRINCIPAL

Figura 38: Actividad del menú principal

Es la primera y la actividad principal de la aplicación. Mediante este menú se podrá acceder a cada una de las partes que compone este software, como:

- Ver información sobre la aplicación y el autor de esta (botón *Acerca de*).
- Poder configurar diferentes parámetros (botón *Configuración*).
- Acceder al diferente material multimedia de forma rápida (botón *Multimedia*) .
- Consultar las puntuaciones registradas (botón *Puntuaciones*).
- Acceder a los diversos juegos (botón *Jugar*).

Antes de que se explique cómo se ha diseñado el menú principal, se comentará como se han diseñado los archivos XML que componen la animación inicial.

Primeramente se deberán crear los ficheros XML de cada una de las animaciones en el directorio *res/anim/nombre_del_archivo_de_animación.xml* y configurar los atributos de estas.

Para el título principal (*Figura 39*), se configuran sus atributos de forma que la animación, tanto la rotación del texto (*rotate*) como la transparencia (*alpha*) dure 3 segundos. La rotación tendrá su origen en el centro del texto (*pivotX* y *pivotY*) y rotará 720˚ (*fromDegrees*, *toDegrees*), además se configurará para que el texto aparezca en pantalla (*fromAlpha*, *toAlpha*).

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate
        android:duration="3000"
        android:fromDegrees="0"
        android:toDegrees="720"
        android:pivotX="50%"
        android:pivotY="50%" />
    <alpha
        android:duration="3000"
        android:fromAlpha="0"
        android:toAlpha="1" />
</set>
```

Figura 39: Archivo XML de la animación del texto principal

Los botones simplemente aparecerán uno tras otro de forma coordinada. Se ha tenido en cuenta cuando aparece cada botón en pantalla (*startOffSet)* para que no haya solapamiento y cada animación siga a la siguiente.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
    <alpha
        android:startOffset="3000"
        android:duration="500"
        android:fromAlpha="0"
        android:toAlpha="2" />
</set>
```

Figura 40: Archivo XML de la animación del botón jugar

Una vez se ha explicado cómo se han creado los archivos XML de las animaciones se comentará cómo se ha diseñado el menú principal. En principio su archivo XML es muy sencillo. Consta de un texto principal (Vista de tipo *TextView*) y seis botones (Vistas de tipo *Button*).

```
<TextView
    android:id="@+id/Titulo_menu_principal"
    style="@style/estiloTextoTituloMenuPrincipal"
    android:textStyle="bold"
    android:text="@string/tituloAplicacion" />
```

Figura 42: Archivo XML del menú principal (II)

```
<Button
    android:id="@+id/Boton_jugar"
    style="@style/estiloTextoTituloMenuPrincipal.Botones"
    android:onClick="lanzarJuego"
    android:text="@string/botonJugar" />
```

Figura 41: Archivo XML del menú principal (I)

Cada vista tiene su propio identificador (id) para su posterior identificación por código Java, su hoja de estilo y utiliza los recursos de la carpeta *String* para facilitar la escritura de textos. Además, los botones mediante el atributo *onClick,* harán referencia a un método situado en el archivo Java.

Cabe destacar que se ha elegido un *LinearLayout* para contener las vistas ya que no se necesita un orden demasiado estricto y como añadido, mediante la vista *GestureOvelayView* se permite al usuario poder pintar sobre la pantalla sin que esta opción tenga un fin más allá que un entretenimiento. Lo que se pinta se borrará pasado un segundo y medio (*fadeOffSet*).

```
<LinearLayout
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="30dip"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:background="@drawable/fondo_nubes"
    tools:context=".MenuPrincipal" >
```

Figura 44: Archivo XML del menú principal (IV)

```
<android.gesture.GestureOverlayView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/gestures"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gestureStrokeType="multiple"
    android:fadeOffset="1500">
```

Figura 43: Archivo XML del menú principal (III)

Por último queda comentar como se ha diseñado el menú que aparece una vez se ha pulsado el botón físico "menú" del dispositivo.

Figura 45: Menú

Para el diseño de este menú se deberá crear un archivo XML en el directorio *res/menu/nombre_del_archivo.xml* en mi caso lo he llamado *menu.xml*

Cada icono que se incluya deberá tener un identificador, un icono (es posible usar los iconos proporcionados por el sistema) y un título o texto.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/juega"
        android:icon="@android:drawable/btn_star_big_on"
        android:title="@string/Jugar">
    </item>
    <item
        android:id="@+id/puntua"
        android:icon="@android:drawable/ic_dialog_dialer"
        android:title="@string/Puntuaciones">
    </item>
    <item
        android:id="@+id/multim"
        android:icon="@android:drawable/ic_menu_slideshow"
        android:title="@string/MultimediaMenu">
    </item>
    <item
        android:id="@+id/config"
        android:icon="@android:drawable/ic_menu_preferences"
        android:title="@string/ConfiguracionMenu">
    </item>
    <item
        android:id="@+id/acercaDe"
        android:icon="@android:drawable/ic_menu_info_details"
        android:title="@string/AcercaDeMenu">
    </item>
    <item
        android:id="@+id/salir"
        android:icon="@android:drawable/ic_menu_revert"
        android:title="@string/botonSalir">
    </item>
</menu>
```

Figura 46: Archivo XML del menú

5.4.2 ACERCA DE

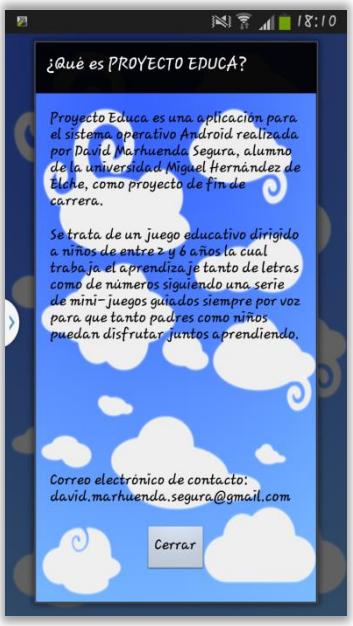


Figura 47: Actividad "Acerca de"

Está actividad explica en que consiste la aplicación, quién la ha hecho y para quien va dirigida. Básicamente su objetivo es informar.

Para el diseño se ha usado un *Layout* de tipo *Relative* para tener más control sobre la posición de las vistas.

Sólo han sido necesarias vistas de tipo *TextView* y *Button.*

```
<TextView
    android:id="@+id/textoAcercaDe"
    style="@style/estiloTextoAcercaDe"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:text="@string/explicacionAcercaDe" />

<TextView
    android:id="@+id/textoAcercaDe2"
    style="@style/estiloTextoAcercaDe"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textoAcercaDe"
    android:text="@string/explicacionAcercaDe2" />

<TextView
    android:id="@+id/textoContacto"
    style="@style/estiloTextoContacto"
    android:layout_above="@+id/Boton_cerrar"
    android:layout_alignParentLeft="true"
    android:layout_marginBottom="20dip"
    android:text="@string/contacto" />

<Button
    android:id="@+id/Boton_cerrar"
    style="@style/estiloBotonAcercaDe"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="30dp"
    android:onClick="lanzarCerrar"
    android:text="@string/cerrar" />
```

Figura 48: Archivo XML de la actividad “Acerca de” (I)

Como en gran parte de la aplicación, incluido el menú principal, mediante la carpeta de recursos *drawable* se ha utilizado una imagen como fondo de pantalla.

Usamos los atributos *layaut_above*, *layaut_below*, *layout_alignParentLeft* y *layout_alignParentTop* para ordenar las vistas.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/RelativeLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fondo_nubes"
    android:orientation="vertical" >
```

Figura 49: Archivo XML de la actividad “Acerca de” (II)

5.4.3 CONFIGURACIÓN

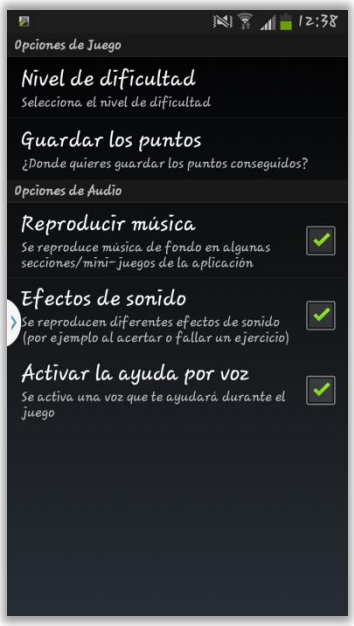


Figura 50: Actividad "Configuración" (I)

El objetivo de esta actividad consiste en poder configurar ciertas opciones de la aplicación a gusto del usuario.

Se han creado dos tipos de opciones:

- *Opciones de juego*: se podrá cambiar la dificultad del juego, la cual influye en el tiempo que disponemos para hacer las pruebas o la cantidad de “letras a capturar” en uno de los mini-juegos, o decidir si guardamos los puntos conseguidos en la memoria interna o externa del dispositivo.
- *Opciones de audio*: se podrán activar o desactivar el audio de fondo, los efectos de sonido de alguno de los mini-juegos o la ayuda por voz que algunos juegos usan.

Figura 52: Actividad "Configuracion" (III)

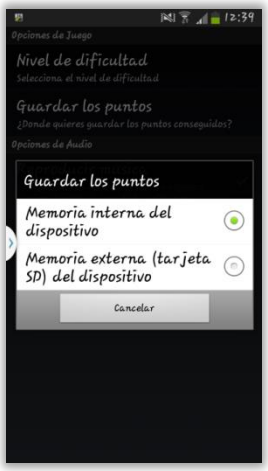


Figura 51: Actividad "Configuracion" (II)

El diseño del archivo XML de esta actividad difiere de otro tipo de actividades. Se deberá crear el archivo en el directorio *res/xml/nombre_del_archivo.xml*, en mi caso lo he llamado *preferencias.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:key="preferencias_principal" >
    <PreferenceCategory android:title="@string/opcionesJuego">
        <ListPreference
            android:key="dificultad"
            android:title="@string/dificultad"
            android:summary="@string/explicacionDificultad"
            android:entries="@array/tiposDificultad"
            android:entryValues="@array/tiposDificultadValores"
            android:defaultValue="1"/>
        <ListPreference
            android:key="memoria"
            android:title="@string/tipoMemoria"
            android:summary="@string/explicacionMemoria"
            android:entries="@array/tiposMemoria"
            android:entryValues="@array/tiposMemoriaValores"
            android:defaultValue="0"/>
    </PreferenceCategory>
    <PreferenceCategory android:title="@string/opcinesAudio">
        <CheckBoxPreference
            android:key="musica"
            android:title="@string/reproducirMusica"
            android:summary="@string/explicacionMusica"/>
        <CheckBoxPreference
            android:key="efectos_sonido"
            android:title="@string/efectorSonido"
            android:summary="@string/explicacionEfectos"/>
        <CheckBoxPreference
            android:key="ayudaPorVoz"
            android:title="@string/ayudaPorVoz"
            android:summary="@string/explicacionayudaPorVoz"/>
    </PreferenceCategory>
</PreferenceScreen>
```

Figura 53: Archivo XML de la actividad "Configuración"

Se incluirán cada una de las vistas de la actividad "Configuración" en la etiqueta *PreferenceScreen* y podrán ser organizadas por categorías mediante *PreferenceCategory*.

El atributo *key* es de gran importancia ya que mediante él se accederá por código Java a las propiedades de las diferentes vistas que componen el archivo "Configuración".

Las vistas utilizadas han sido *CheckBoxPreference* y *ListPreference*. Para esta última, se crearán *Arrays* con las posibles opciones a seleccionar en el directorio *res/values/arrays.xml* como se puede ver en la siguiente figura:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="tiposMemoria">
        <item>Memoria interna del dispositivo</item>
        <item>Memoria externa (tarjeta SD) del dispositivo</item>
    </string-array>

    <string-array name="tiposMemoriaValores">
        <item>0</item>
        <item>1</item>
    </string-array>

    <string-array name="tiposDificultad">
        <item>Fácil</item>
        <item>Normal</item>
        <item>Difícil</item>
    </string-array>

    <string-array name="tiposDificultadValores">
        <item>1</item>
        <item>2</item>
        <item>3</item>
    </string-array>
</resources>
```

Figura 54: Archivo XML *Arrays*

Se accederá a través de código Java a las diferentes opciones mediante el número entero que se le ha asignado a los *arrays*.

5.4.4 MULTIMEDIA

Figura 55: Actividad "Multimedia"

En esta actividad se encontrará el material multimedia (vídeos) que son accesibles a través de los juegos, pero de esta manera se podrán consultar rápidamente cuando el usuario lo desee.

Cabe añadir que, al igual que en la mayoría de las actividades, se tendrá acceso a un botón de ayuda para evitar que el usuario se pierda en algún momento.

Figura 56: Ayuda de la actividad "Multimedia"

Para esta actividad se ha usado un *Layout* de tipo *Relative* ya que necesitamos tener el control de las posiciones de las vistas. Para el título se ha utilizado una vista *TextView* y para acceder a los vídeos se han utilizado *Buttons*, estando estos ordenados mediante un *TableLayout* y así poder posicionar fácilmente los botones cuando tengamos la pantalla tanto en posición horizontal como en vertical.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/RelativeLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fondo_nubes"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/tituloMultimedia"
        style="@style/estiloTituloMultimedia"
        android:text="@string/tituloMultimedia"/>

    <TableLayout
        android:id="@+id/tablaMultimedia"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/tituloMultimedia">
        <TableRow android:gravity="center" >
            <Button
                android:id="@+id/boton_Vocales"
                style="@style/estiloBotonMultimedia"
                android:drawableBottom="@drawable/video_vocales"
                android:text="@string/tituloVocalesMultimedia"
                android:onClick="lanzarVideoVocalesMultimedia"
                android:layout_span="1" />
            <Button
                android:id="@+id/boton_Colores"
                style="@style/estiloBotonMultimedia"
                android:drawableBottom="@drawable/video_colores"
                android:text="@string/tituloColoresMultimedia"
                android:onClick="lanzarVideoColoresMultimedia"
                android:layout_span="1" />
        </TableRow>
```

Figura 57: Parte del archivo XML de la actividad "Multimedia"

Cada botón hará uso de la carpeta *drawable* de los recursos para establecer una imagen de fondo y también de un identificador para poder trabajar con las vistas desde el archivo Java. Mediante el atributo *onClick* se lanzará un método para visualizar los vídeos.

Una vez descrita esta parte de la actividad "Multimedia" se continuará describiendo el diseño de la actividad que reproduce los vídeos. Dicha actividad se puede ver en la siguiente figura:

Figura 58: Actividad "VideoMultimedia"

Para esta actividad se ha creado una interfaz propia para la reproducción de los vídeos, es decir, no dependemos de la interfaz multimedia que pueda tener el dispositivo donde estemos usando la aplicación.

Para el control del vídeo disponemos de diferentes botones (play, pause, subir o bajar volumen, etc.). Mediante un *EditText* (el cual tendremos deshabilitada su escritura) sabremos que vídeo se está reproduciendo introduciendo un texto. Por último, el video se reproducirá en una vista de tipo *VideoView*.

En las siguientes figuras se podrá ver parte del archivo XML de esta actividad:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">
```

Figura 59: Archivo XML de la actividad "VideoMultimedia" (I)

El *Layout* que se utilizará será el *Relative* para así posicionar con facilidad las vistas en pantalla.

```
<LinearLayout
    android:id="@+id/ButtonsLayout"
    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/bajar_volumen"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:src="@drawable/bajar_volumen"/>

    <ImageButton
        android:id="@+id/subir_volumen"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:src="@drawable/subir_volumen"/>

    <ImageButton
        android:id="@+id/silenciar"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:src="@drawable/mute"/>
```

Figura 60: Archivo XML de la actividad "VideoMultimedia" (II)

Las vistas de tipo botón con las cuales se controlará la reproducción del vídeo y el título del mismo se ordenarán mediante un *LinearLayout* horizontal.

```
<android.gesture.GestureOverlayView
       android:id="@+id/gestures_videos"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"
       android:gestureColor="#000000"
       android:gestureStrokeType="multiple"
       android:layout_below="@+id/ButonsLayout"
       android:fadeOffset="1000">

       <VideoView
            android:id="@+id/surfaceView"
            android:layout_height="fill_parent"
            android:layout_width="fill_parent" />

</android.gesture.GestureOverlayView>
```

Figura 61: Archivo XML de la actividad "VideoMultimedia" (III)

Por último con el atributo *layout_below* se situará la vista del vídeo por debajo de los botones, además como contenido extra, se habilitará el poder dibujar sobre el vídeo añadiendo una vista *GestureOverlayView*.

5.4.5 PUNTUACIONES

Según el jugador haya configurada la aplicación para que guarde los datos en la memoria externa o interna del dispositivo, visualizaremos los puntos de las partidas de una manera u otra.

El diseño de esta actividad es sencillo. Se trata de un texto en forma de título (*TextView*) y tres botones (*Buttons*) que nos direccionarán a las listas de puntuaciones de los diferentes juegos.

Figura 62: Actividad "Puntuaciones"

Se ha utilizado un *TableLayout* para posicionar de forma ordenada los 3 botones con sus respectivos títulos y un *RelativeLayout* para situar este contenedor en el centro, el botón de ayuda (*Button*) en la parte superior derecha, el botón de retroceso (*Button*) en la parte inferior izquierda y el título de la actividad (*TextView*) en la parte superior central.

```
<Button
    android:id="@+id/boton_Atras_Menu_Puntuaciones"
    style="@style/estiloBotonMenuPuntuaciones"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_marginBottom="15dip"
    android:layout_marginLeft="15dip"
    android:background="@drawable/flecha_atras"
    android:onClick="lanzarAtras" />

<Button
    android:id="@+id/botonAyudaJugar"
    style="@style/estiloBotonAyuda"
    android:background="@drawable/ayuda"
    android:onClick="lanzarAyuda" />
```

Figura 63: Archivo XML de la actividad "Puntuaciones" (I)

```
<TextView
    android:id="@+id/tituloMenuPuntuaciones"
    style="@style/estiloTituloMenuPuntuaciones"
    android:text="@string/tituloMenuPuntuaciones"/>
```

Figura 64: Archivo XML de la activida "Puntuaciones" (II)

```
<TableLayout
    android:id="@+id/tablaMenuPuntuaciones"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_below="@+id/tituloMenuPuntuaciones">

    <TableRow android:gravity="center" >
        <TextView
            android:id="@+id/texto_Menu_Puntuaciones_Letras"
            style="@style/estiloTextoMenuPuntuaciones"
            android:text="@string/tituloVocalesMenuPuntuaciones"
            android:layout_span="1" />

        <TextView
            android:id="@+id/texto_Menu_Puntuaciones_Numeros"
            style="@style/estiloTextoMenuPuntuaciones"
            android:text="@string/tituloNumerosMenuPuntuaciones"
            android:layout_span="1" />

        <TextView
            android:id="@+id/texto_Menu_Puntuaciones_Formas"
            style="@style/estiloTextoMenuPuntuaciones"
            android:text="@string/tituloFormasMenuPuntuaciones"
            android:layout_span="1" />
    </TableRow>
```

Figura 65: Archivo XML de la actividad "Puntuaciones" (III)

```
    <TableRow android:gravity="center" >
        <Button
            android:id="@+id/boton__Puntuaciones_Letras"
            style="@style/estiloBotonMenuPuntuaciones"
            android:background="@drawable/letras_logo"
            android:onClick="lanzarPuntuacionesLetras"
            android:layout_span="1" />

        <Button
            android:id="@+id/boton_Puntuaciones_Numeros"
            style="@style/estiloBotonMenuPuntuaciones"
            android:background="@drawable/numeros_logo"
            android:onClick="lanzarPuntuacionesNumeros"
            android:layout_span="1" />

        <Button
            android:id="@+id/boton_Puntuaciones_Formas"
            style="@style/estiloBotonMenuPuntuaciones"
            android:background="@drawable/formas_icon"
            android:onClick="lanzarPuntuacionesFormas"
            android:layout_span="1" />
    </TableRow>
</TableLayout>
```

Figura 66: Archivo XML de la actividad "Puntuaciones" (IV)

Una vez se pulse uno de los botones, se visualizarán las puntuaciones del juego seleccionado. Como los diseños son prácticamente iguales (sólo difieren en las imágenes decorativas que incorporan) se dispondrá a explicar solamente uno de ellos.

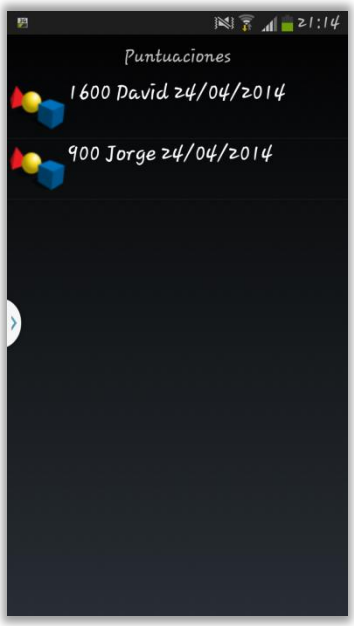


Figura 67: Lista de puntuaciones

En esta actividad se muestran los puntos conseguidos, el nombre que se ha utilizado para identificar al jugador y la fecha de cuando tuvo lugar la partida.

Esta actividad está compuesta por un título (vista de tipo *TextView*) y bajo esta vista se situará un contenedor de tipo *FrameLayout,* el cual si no hubiera ninguna puntuación guardada mostraría un mensaje informando sobre ello y sino, mediante una vista de tipo *ListView* se visualizarían las puntuaciones contenidas en la memoria del dispositivo.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/Puntuaciones"
            android:gravity="center"
            android:layout_margin="10dp"
            android:textSize="10pt"/>
    <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="0dip"
            android:layout_weight="1">
            <ListView
                    android:id="@android:id/list"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:drawSelectorOnTop="false" />
            <TextView
                    android:id="@android:id/empty"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:text="@string/NoPuntuaciones" />
    </FrameLayout>
</LinearLayout>
```

Figura 68: Archivo XML de la actividad "ListaPuntuaciones"

Las filas donde aparecen las puntuaciones de cada jugador han sido personalizadas de manera que aparezca una imagen a la izquierda para así diferenciar de forma más visual de que juego son las puntuaciones que se están consultado. Esto se hace creando otro archivo XML, en este caso, el contenedor será de tipo *RelativeLayout* ya que se requiere posicionar con precisión la imagen y el texto. Mediante los atributos *?android:attr/listPreferredItemHeight* y *?android:attr/textAppearanceLarge* se establece el alto o ancho del contenedor o de la vista a partir de un parámetro de configuración del sistema.

```
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="?android:attr/listPreferredItemHeight">
        <ImageView
              android:id="@+id/icono"
              android:layout_width="?android:attr/listPreferredItemHeight"
              android:layout_height="match_parent"
              android:layout_alignParentLeft="true"
              android:src="@drawable/formas_icon_lista"/>
        <TextView
              android:id="@+id/titulo_formas"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_toRightOf="@id/icono"
              android:layout_alignParentTop="true"
              android:textAppearance="?android:attr/textAppearanceLarge"
              android:singleLine="true" />
</RelativeLayout>
```

Figura 69: Archivo XML "ElementoLista"

5.4.6 JUGAR

Esta actividad tiene un diseño y una estructura muy similar a la actividad “Puntuaciones” difiriendo solamente en las imágenes de los botones, por lo tanto su archivo XML ya ha sido previamente comentado.

Figura 70: Actividad del menú de los juegos

5.4.6.1 MINI-JUEGOS

Figura 71: Actividad del menú de los mini-Juegos

Una vez más, para agilizar el proceso de desarrollo de esta aplicación, se ha reutilizado una estructura y diseño previamente implementada en otras actividades, por lo tanto ya ha sido comentada y explicada con anterioridad.

- ***FORMAS GEOMÉTRICAS***

Este mini-juego está compuesto por varias actividades. La primera de ellas será un video educativo cuyo diseño ya ha sido explicado (*CAPÍTULO 5.4.4 MULTIMEDIA*). La segunda actividad se trata del primer nivel del juego y consiste en que el usuario toque los botones que componen las formas geométricas, donde estas tendrán un audio asociado para así aprender e identificar de qué forma se trata. Una vez se quiera seguir jugando, basta con pulsar la flecha.

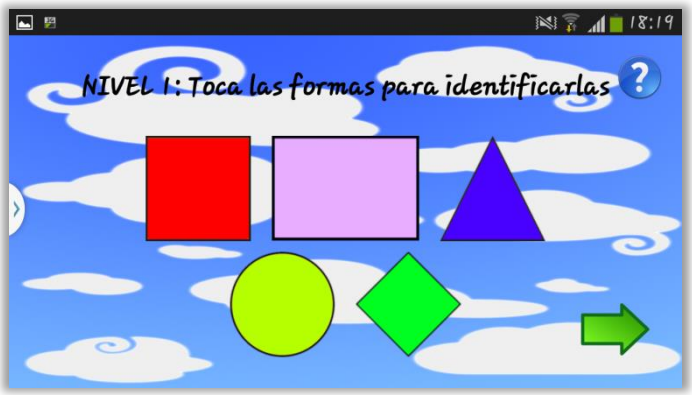


Figura 72: Actividad del primer nivel del juego de las formas

Durante todo el juego se podrá acceder al botón de ayuda donde se informará de que hacer en caso de perderse.

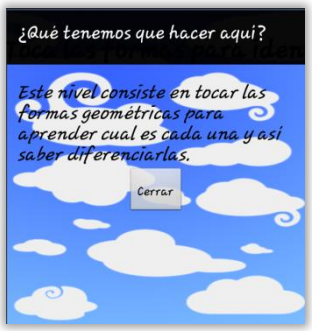


Figura 73: Ayuda de la actividad "FormasGeometricas"

El archivo XML consiste en un *TextView* que representa el título del nivel donde se encuentra el usuario, los botones de ayuda y continuar y los cinco botones que componen cada una de las formas geométricas a trabajar, estos estructurados mediante un contenedor de tipo *LinearLayout* vertical que contiene dos *LinearLayout* horizontales, uno para cada fila de botones, todo ello contenido en un *RelativeLayout* para no perder la posibilidad de un diseño más estructurado.

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_gravity="center" >

    <Button
        android:id="@+id/boton_Circulo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="10dip"
        android:background="@drawable/circulo_juego_formas"
        android:onClick="Sonido_Circulo" />

    <Button
        android:id="@+id/boton_Rombo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dip"
        android:background="@drawable/rombo_juego_formas"
        android:onClick="Sonido_Rombo" />
</LinearLayout>
```

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginBottom="10dip" >

    <Button
        android:id="@+id/boton_Cuadrado"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dip"
        android:layout_marginRight="10dip"
        android:background="@drawable/cuadrado_juego_formas"
        android:onClick="Sonido_Cuadrado" />

    <Button
        android:id="@+id/boton_Rectangulo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dip"
        android:layout_marginRight="10dip"
        android:background="@drawable/rectangulo_juego_formas"
        android:onClick="Sonido_Rectangulo" />

    <Button
        android:id="@+id/boton_Triangulo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dip"
        android:layout_marginRight="10dip"
        android:background="@drawable/triangulo_juego_formas"
        android:onClick="Sonido_Triangulo" />
</LinearLayout>
```

```
<Button
    android:id="@+id/boton_siguiente_nivel_juego_formas"
    style="@style/estiloBotonSiguienteNivel"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:background="@drawable/flecha_siguiente_nivel"
    android:onClick="lanzarSiguienteNivel" />

<Button
    android:id="@+id/botonAyudaJuegoFormasPrimerNivel"
    style="@style/estiloBotonAyuda"
    android:background="@drawable/ayuda"
    android:onClick="lanzarAyuda" />
```

Figura 74: Parte del archivo XML del primer nivel del juego de las formas

En la tercera actividad se encuentra el nivel 2 de este mini-juego. Se nos indicará que figura debemos buscar mediante un audio (si se ha configurado la aplicación para ello) y mediante texto por un *TextView* que irá cambiado, al igual que la posición de las figuras que también irán cambiando. Tendremos un tiempo límite para ello (el tiempo dependerá de la dificultad seleccionada además de que cuando resten 10 segundos para finalizar, el temporizador cambiará a un color rojo a modo de advertencia). Si acertamos incrementaremos nuestra puntuación y esto se verá reflejado mediante una animación y un audio al igual que si fallamos.

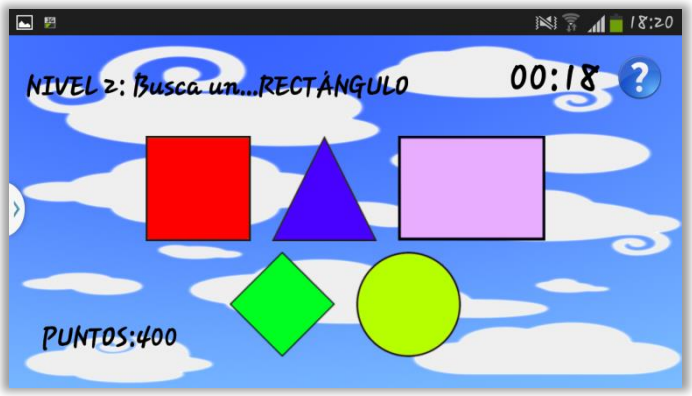


Figura 75: Actividad del segundo nivel del juego de formas (I)

El archivo XML es similar al de la actividad anterior, pero es necesario puntualizar varias cosas:

- Los botones que componen cada una de las formas geométricas no tendrán una imagen asociada, esa imagen será modificada dinámicamente en el archivo Java.
- Será necesario incluir varios *TextView* más, tales como los puntos (donde tendremos dos *TextView*, uno será un texto estático y el otro un texto que indicará los puntos según estos aumenten) el temporizador o la información que nos indica que figura será la siguiente a buscar (al igual que con los puntos, habrán dos TextView, uno será estático y el otro irá cambiado según qué forma hay que buscar).

```
<TextView
    android:id="@+id/puntos_juego_formas_segundo_nivel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@style/estiloTituloJuegosFormas"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_marginBottom="15dip"
    android:layout_marginLeft="15dip"
    android:text="@string/puntos" />

<TextView
    android:id="@+id/puntos_conseguidos_juego_formas_segundo_nivel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@style/estiloTituloJuegosFormas"
    android:layout_toRightOf="@id/puntos_juego_formas_segundo_nivel"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="15dip" />
```

```
<TextView
    android:id="@+id/contador_juego_formas_segundo_nivel"
    style="@style/estiloCuentaAtras"
    android:layout_marginRight="15dip"
    android:layout_toLeftOf="@id/botonAyudaJuegoFormasSegundoNivel" />
```

Figura 76: Parte del archivo XML del segundo nivel del juego de las formas

Una vez superado cada uno de los niveles se nos informará de nuestra puntuación hasta el momento.

Figura 77: Actividad del segundo nivel del juego de las formas (II)

El último nivel consiste en dibujar la figura que se indique (mediante texto y si está habilitada la opción, audio) la cual irá cambiando de forma automática.

Figura 78: Actividad del tercer nivel del juego de formas (I)

En principio el diseño de esta actividad solo difiere de la anterior en la superficie donde dibujamos. Se trata de una vista de tipo *GestureOverlayView*. Se ha utilizado un fondo de color diferente sobre esta superficie para que el usuario intuya donde tiene que dibujar, además de configurar el pincel a un color negro y limitar el tiempo de reconocimiento de formas en 300 milisegundos.

Una vez finalizado el juego se mostrará en una pantalla de diálogo los puntos totales conseguidos, se volverá al menú principal y se dará al usuario la opción de escribir un nombre y guardar la partida.

```
<android.gesture.GestureOverlayView
        android:id="@+id/gestures_formas"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#A9A9F5"
        android:gestureColor="#000000"
        android:gestureStrokeType="multiple"
        android:layout_below="@id/explicacionFormasTercerNivel"
        android:layout_above="@id/puntos_juego_formas_tercer_nivel"
        android:fadeOffset="300">
</android.gesture.GestureOverlayView>
```

Figura 79: Parte del archivo XML del tercer nivel del juego de las formas

Figura 80: Actividad del tercer nivel del juego de formas (II)

○ *PINTA Y COLOREA*

El segundo de los mini-juegos consiste sencillamente en pintar utilizando la pantalla táctil del dispositivo.

Se dispondrá de diferentes botones para seleccionar el color del lienzo, del pincel, el grosor de este y para borrar el dibujo. Además de los botones que hasta hora ya se han usado como el de ayuda o el de volver a la actividad anterior.

En la parte central estará la superficie donde el jugador podrá pintar, esta es una vista que se ha creado especialmente para este fin.

Figura 81: Actividad del juego pintar

Se ha usado un contenedor *RelativeLayout* para situar de una forma ordenada y precisa todas estas vistas.

```
<Button
    android:id="@+id/botonCambiarFondoJuegoPintar"
    style="@style/estiloBotonJuegoPintar"
    android:drawableBottom="@drawable/lienzo"
    android:text="@string/textoBotonLienzo"
    android:padding="0dip"
    android:onClick="lanzarCambiarColorLienzo" />
```

```
<Button
    android:id="@+id/botonCambiarColorPincelJuegoPintar"
    style="@style/estiloBotonJuegoPintar"
    android:layout_below="@+id/botonCambiarFondoJuegoPintar"
    android:drawableBottom="@drawable/pincel_pintar"
    android:text="@string/textoBotonPincel"
    android:padding="0dip"
    android:onClick="lanzarCambiarColorPincel" />
```

```
<TextView
    android:id="@+id/pintar_aqui"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="30sp"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:hint="@string/textoPintarAqui"/>
```

```
<Button
    android:id="@+id/botonNivelGrosorJuegoPintar"
    style="@style/estiloBotonJuegoPintar"
    android:drawableBottom="@drawable/nivel_grosor_pincel"
    android:layout_below="@+id/botonCambiarColorPincelJuegoPintar"
    android:text="@string/textoBotonNivelGrosor"
    android:padding="0dip"
    android:onClick="lanzarNivelGrosor" />
```

Figura 82: Archivo XML de la actividad del juego pintar (I)

```
<Button
    android:id="@+id/botonNuevaPaginaJuegoPintar"
    style="@style/estiloBotonJuegoPintar"
    android:layout_alignParentRight="true"
    android:drawableBottom="@drawable/nueva_pagina"
    android:layout_below="@id/botonAyudaJuegoPintar"
    android:text="@string/textoBotonNuevaPagina"
    android:padding="0dip"
    android:onClick="lanzarNuevaPagina" />
```

```
<com.example.proyectoeduca.VistaLienzo
    android:id="@+id/vistaLienzo"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:layout_toRightOf="@id/botonCambiarFondoJuegoPintar"
    android:layout_toLeftOf="@id/botonNuevaPaginaJuegoPintar" />
```

Figura 83: Archivo XML de la actividad del juego pintar (II)

Como se puede ver, se ha configurado el atributo *onClick* de los botones para que se ejecute un método concreto en el archivo Java. Dicho método, para los botones donde se selecciona el color del lienzo, del pincel y su grosor, abrirá una nueva actividad en forma de ventana de diálogo donde se podrá elegir (pulsando sobre una imagen) el parámetro deseado ya sea un color o un grosor.

Figura 84: Ventanas de diálogo de la actividad "Pintar"

En principio los archivos XML de estas ventanas de diálogo tienen la misma estructura. Todas las vistas estarán en un contenedor tipo *ScrollView*, así se conseguirá crear una actividad con una pantalla deslizable para incorporar muchos elementos (en este caso botones).

Cada botón accederá a los recursos donde se le asignará una imagen y tendrá asociado un método en su atributo *onClick* el cual devolverá una cadena de caracteres para identificar los diferentes colores o un dato entero para identificar el grosor, esto se verá más detenidamente en el punto *5.5 IMPLEMENTACIÓN*.

En la siguiente figura se podrá ver parte del archivo XML donde el jugador podrá seleccionar el color del lienzo, en principio los otros dos archivos restantes serán muy parecidos, simplemente cambiarán en el método al que llaman en el atributo *onClick* y la imagen usada para el botón.

```
<ScrollView
    android:id="@+id/ScrollView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" >

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:layout_gravity="center" >
            <Button
                android:id="@+id/lienzo_amarillo"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:background="@drawable/lienzo_amarillo"
                android:onClick="lanzarLienzoAmarillo" />
            <Button
                android:id="@+id/lienzo_amarillo_verde"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:background="@drawable/lienzo_amarillo_verde"
                android:onClick="lanzarLienzoAmarilloVerde" />
        </LinearLayout>
```

Figura 85: Archivo XML para seleccionar el color del lienzo

○ *CAPTURA LAS LETRAS*

Previamente al desarrollo de este proyecto, desconocía como desarrollar una aplicación completa para Android así que entre la documentación consultada se explicaba cómo realizar esto desarrollando el clásico juego "Asteroides" de máquina recreativa. Una vez completados los ejercicios planteados y por lo tanto el juego, no quise que todo ese trabajo se desechara, por lo tanto, modifiqué el código para que dicho juego se adaptara de forma lúdica y entretenida a este proyecto.

Así que por lo tanto, tenemos una adaptación del juego "Asteroides" donde una nave, lanza lápices (en lugar de misiles) a las letras del abecedario (en lugar de a asteroides), una vez el lápiz impacta en la letra, esta es "capturada" y desaparece no sin antes escuchar un clip de audio donde se nos informa de que letra ha sido capturado.

Comentar también que dependiendo del nivel de dificultad aparecerán más letras o menos en pantalla.

Figura 86: Actividad "JuegoNave"

Para el desarrollo del archivo XML se ha creado una vista propia llamada *VistaJuego* donde se situarán todos los elementos que participan en el juego.

Una vez el juego ha finalizado se sobrescribirá en pantalla un mensaje de que el juego ha terminado.

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <com.example.proyectoeduca.VistaJuego
          android:id="@+id/VistaJuego"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent"
          android:background="@drawable/juego_nave_fondo" />
    <TextView
        android:id="@+id/Victoria"
        style="@style/estiloTextoVictoriaJuegoNave"
        android:text="@string/textoVictoriaJuegoNave" />
</FrameLayout>
```

Figura 87: Archivo XML de la actividad"JuegoNave"

5.4.6.2 LETRAS

Este juego está compuesto por varias actividades siendo cada una de ellas un nivel diferente sin tener en cuenta el vídeo explicativo cuyo diseño ya ha sido comentado (*CAPÍTULO 5.4.4 MULTIMEDIA*).

En el primer nivel del juego se tendrá que interactuar con las letras que aparece en pantalla (vistas de tipo *Button*), cada letra tendrá un audio asociado, de esta manera el jugador las identificará con su sonido.

Figura 88: Actividad del primer nivel del juego de las letras

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >
    <Button
        android:id="@+id/boton_A"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/letra_a_juego_vocales"
        android:onClick="Sonido_Letra_A" />
    <Button
        android:id="@+id/boton_E"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/letra_e_juego_vocales"
        android:onClick="Sonido_Letra_E" />
    <Button
        android:id="@+id/boton_I"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/letra_i_juego_vocales"
        android:onClick="Sonido_Letra_I" />
    <Button
        android:id="@+id/boton_O"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/letra_o_juego_vocales"
        android:onClick="Sonido_Letra_O" />
    <Button
        android:id="@+id/boton_U"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/letra_u_juego_vocales"
        android:onClick="Sonido_Letra_U" />
</LinearLayout>
```

Figura 89: Archivo XML del primer nivel del juego de las letras (I)

Para el título se ha usado una vista *TextView*. Para las letras se ha creado un *LinearLayout* vertical que contiene dos *LinearLayout* horizontales, situando las letras mayúsculas en el primero y las minúsculas en el segundo. Para la imagen de fondo y de los botones se ha recurrido a la carpeta *drawable* de los recursos del sistema, todo esto contenido en un *RelativeLayout* para tener un mayor control de las posiciones de las vistas.

Durante los diferentes niveles que componen este juego se podrá acceder al botón de ayuda para asesorar al jugador que hacer en caso de perderse.

```
<TextView
    android:id="@+id/explicacionVocales"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@style/estiloTituloJuegosVocales"
    android:layout_marginLeft="15dip"
    android:layout_alignParentTop="true"
    android:text="@string/textoTituloJuegoVocalesPrimerNivel" />
```

```
<Button
    android:id="@+id/boton_siguiente_nivel_juego_vocales"
    style="@style/estiloBotonSiguienteNivel"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:background="@drawable/flecha_siguiente_nivel"
    android:onClick="lanzarSiguienteNivel" />

<Button
    android:id="@+id/botonAyudaJuegoLetrasPrimerNivel"
    style="@style/estiloBotonAyuda"
    android:background="@drawable/ayuda"
    android:onClick="lanzarAyuda" />
```

Figura 90: Archivo XML del primer nivel del juego de las letras (II)

Una vez se han asimilado los conceptos, pulsando sobre la flecha se avanzará hacia el siguiente nivel.

Figura 91: Actividad del segundo nivel del juego de las letras (I)

Ahora el jugador aparte de poder identificar cada letra por su sonido, podrá grabarse para así afianzar los conocimientos adquiridos. Si se pulsa sobre una letra se abrirá una ventana de diálogo en la cual aparecerá una vista de

Figura 92: Actividad del segundo nivel del juego de las letras (II)

tipo *Button* con la letra en cuestión y tres botones en la parte inferior. Con el primero de ellos en forma de micrófono, pulsando se podrá grabar sonido, a continuación pulsado sobre el botón *Stop* dejará de grabar y finalmente para escuchar la grabación se pulsará sobre el botón *Play*.

El archivo XML de este nivel es prácticamente igual al del nivel anterior difiriendo en el método asociado a cada letra que en lugar de solamente escuchar el sonido asociado, se escuchará dicho sonido y se abrirá una ventana de diálogo.

Para el archivo XML de la ventana de diálogo se ha usado un *RelativeLayout* para posicionar el botón que representa la letra (cuyo fondo de pantalla será modificado dinámicamente desde código Java dependiendo de qué letra abramos) y un *LinearLayout* horizontal que representan los botones de la interfaz de usuario para la grabación de sonido.

```
<LinearLayout
    android:id="@+id/linearLayout1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_below="@id/letra_abierta">

    <Button
        android:id="@+id/boton_grabar_vocales_nivel_2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/microfono"
        android:onClick="grabar" />

    <Button
        android:id="@+id/boton_parar_vocales_nivel_2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/stop_icon"
        android:onClick="detenerGrabacion" />

    <Button
        android:id="@+id/boton_play_vocales_nivel_2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/play_icon"
        android:onClick="reproducir" />
</LinearLayout>
```

Figura 93: Archivo XML del segundo nivel del juego de las letras

En este nivel tres, se tendrá que buscar la letra que se indique por pantalla (si mediante la configuración de las opciones de la aplicación se ha habilitado la ayuda por voz, también el jugador podrá guiarse mediante sonido).

Figura 94: Actividad del tercer nivel del juego de las letras

En la parte superior tendremos diferentes vistas de tipo *TextView* y un botón (ayuda para el jugador). Sólo uno de los texto será estático (el que nos informa en qué nivel nos encontramos), los restantes (letra a buscar y cuenta atrás) irán variado según sea necesario por código Java.

En la parte central tendremos cinco botones que irán cambiando dinámicamente cuando acertemos o fallemos. En principio estos botones no tendrán ningún fondo de pantalla asociado ya que irán cambiando continuamente. El contenedor donde se depositan es un sencillo *LinearLayout* horizontal.

Por último en la parte inferior tendremos dos textos, uno estático y otro que se incrementará mediante una animación según el jugador vaya acertando y ganando puntos.

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dip"
    android:gravity="center"
    android:orientation="horizontal"
    android:layout_below="@id/explicacionVocalesTercerNivel">
    <Button
        android:id="@+id/boton_Letra_1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="Letra_1_pulsada" />
    <Button
        android:id="@+id/boton_Letra_2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="Letra_2_pulsada" />
    <Button
        android:id="@+id/boton_Letra_3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="Letra_3_pulsada" />
    <Button
        android:id="@+id/boton_Letra_4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="Letra_4_pulsada" />
    <Button
        android:id="@+id/boton_Letra_5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="Letra_5_pulsada" />
</LinearLayout>
```

Figura 95: Archivo XML del tercer nivel del juego de las letras (I)

```
<TextView
    android:id="@+id/puntos_juego_vocales_tercer_nivel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@style/estiloTituloJuegosVocales"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_marginBottom="15dip"
    android:layout_marginLeft="15dip"
    android:text="@string/puntos" />

<TextView
    android:id="@+id/puntos_conseguidos_juego_vocales_tercer_nivel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@style/estiloTituloJuegosVocales"
    android:layout_toRightOf="@id/puntos_juego_vocales_tercer_nivel"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="15dip" />

<Button
    android:id="@+id/botonAyudaJuegoLetrasTercerNivel"
    style="@style/estiloBotonAyuda"
    android:background="@drawable/ayuda"
    android:onClick="lanzarAyuda" />

<TextView
    android:id="@+id/contador_juego_vocales_tercer_nivel"
    style="@style/estiloCuentaAtras"
    android:layout_marginRight="15dip"
    android:layout_toLeftOf="@id/botonAyudaJuegoLetrasTercerNivel" />
```

Figura 96: Archivo XML del tercer nivel del juego de las letras (II)

En el penúltimo nivel el jugador tendrá que escribir la letra que se le indique la cual irá cambiando continuamente.

Figura 97: Actividad del cuarto nivel del juego de las letras

```
<android.gesture.GestureOverlayView
        android:id="@+id/gestures_vocales"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#A9A9F5"
        android:gestureColor="#000000"
        android:gestureStrokeType="multiple"
        android:layout_below="@id/explicacionVocalesCuartoNivel"
        android:layout_above="@id/puntos_juego_vocales_cuarto_nivel"
        android:layout_marginTop="15dip"
        android:fadeOffset="500">
</android.gesture.GestureOverlayView>
```

Figura 98: Archivo XML del cuarto nivel del juego de las letras

El archivo XML de este cuarto nivel será casi idéntico al anterior, a excepción de la superficie donde dibujamos las letras. Esto se hará sobre una vista de tipo *GestureOverlayView*, donde además podremos configurar algunos atributos del pincel con el que dibujamos, como el color. En principio se dejará al usuario un tiempo prudencial

para escribir la letra y que la aplicación reconozca el patrón de escritura una vez se separa el dedo de la pantalla táctil (500 milisegundos).

Por último, en el quinto nivel el jugador tendrá que completar la palabra con la letra que falta, para ello se dispondrá de una imagen asociada a la palabra en concreto.

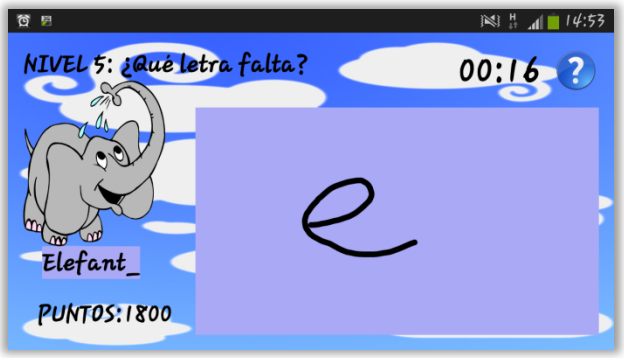


Figura 99: Actividad del quinto nivel del juego de las letras

Los textos del nivel donde nos encontramos el contador y los puntos no difieren de los niveles anteriores, son vistas de tipo *TextView*. También la superficie donde escribimos será la ya usada, simplemente la situamos a la derecha de una vista *ImageView* y el texto asociado en la parte inferior de esta donde mediante código Java tanto las imágenes como los textos irán cambiando según vayamos acertando en el juego.

```
<ImageView
    android:id="@+id/imagenJuegoVocalesQuintoNivel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/explicacionVocalesQuintoNivel" />

<TextView
    android:id="@+id/textoImagenJuegoVocalesQuintoNivel"
    style="@style/estiloTituloJuegosVocales"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/imagenJuegoVocalesQuintoNivel"
    android:background="#A9A9F5"
    android:layout_marginLeft="20dip" />
```

Figura 100: Archivo XML del quinto nivel del juego de las letras

5.4.6.3 NÚMEROS

En este juego nos encontraremos que algunas actividades siguen el mismo patrón que el juego anteriormente explicado. Tanto el vídeo inicial como el primer y segundo nivel siguen la misma estructura de actividades en lo que a diseño del archivo XML se refiere. Únicamente cambiará el recurso (ya sea vídeo o imagen) a utilizar. Por lo tanto se comenzará explicando el nivel 3 de este juego, el cual si difiere del juego anterior.

Figura 101: Actividad del segundo nivel del juego de los números (I)

Figura 102: Actividad del segundo nivel del juego de los números (II)

El nivel 3 de este juego consistirá, como ya hemos hecho en otros, en dibujar el número que se indique. En principio el diseño de la actividad y su archivo XML ya ha sido explicado y únicamente difiere en los patrones de reconocimiento. Archivo que se comentará en el siguiente punto (*CAPÍTULO 5.5 IMPLEMENTACIÓN*).

Figura 103: Actividad del tercer nivel del juego de los números

En el cuarto nivel el jugador deberá contar. Tendremos un texto en la parte superior izquierda (vista *TextView*) donde se informará del nivel donde nos encontramos e información de que debemos hacer. Tendremos otros textos en la parte superior derecha (contador de tiempo) y en la parte inferior izquierda (recuento de puntos). En

el centro situaremos una zona para escribir el resultado y otra donde aleatoriamente irán apareciendo diferente cantidad de imágenes.

Figura 104: Actividad del cuarto nivel del juego de los números

En el diseño del archivo XML de esta actividad ya se ha explicado cómo crear los textos del título, temporizador y contador de puntos, además también se ha explicado que tipo de vista es donde escribimos y como modificar sus atributos así que directamente se comentará la zona de las imágenes.

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:orientation="horizontal">

    <Button
        android:id="@+id/objeto1_juego_numeros"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/oso_juego_numeros" />
    <Button
        android:id="@+id/objeto2_juego_numeros"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/oso_juego_numeros" />
    <Button
        android:id="@+id/objeto3_juego_numeros"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/oso_juego_numeros" />
    <Button
        android:id="@+id/objeto4_juego_numeros"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/oso_juego_numeros" />
    <Button
        android:id="@+id/objeto5_juego_numeros"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/oso_juego_numeros" />
</LinearLayout>
```

Figura 105: Archivo XML del cuarto nivel del juego de los números

Se han creado dos *LinearLayout* horizontales dentro de un *LinearLayout* vertical, donde en cada uno de los *LinearLayout* horizontales tendremos una serie de botones que representan cada uno de los objetos. Dichos botones tendrán un identificador claramente diferenciable para poder trabajar con ellos desde código Java y así saber cuántos de los botones hacer visibles o invisibles para que el jugador realice sin problemas el recuento aleatorio de objetos.

Por último en el quinto nivel se deberá de realizar una suma sencilla (el resultado nunca pasará de 9).

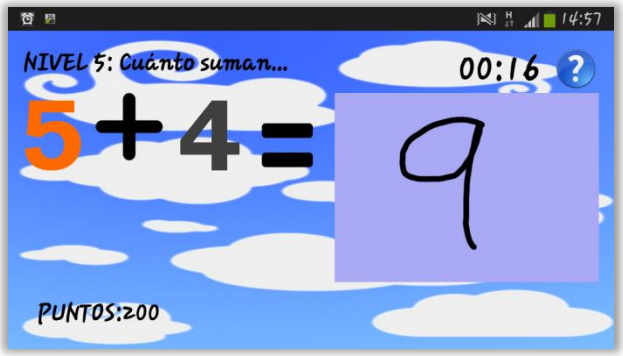


Figura 106: Actividad del quinto nivel del juego de los números

La única diferencia que presenta esta actividad respecto a la anterior es la zona donde situamos las imágenes de la suma. Se colocarán cuatro imágenes (vistas de tipo *ImageView*): dos estáticas (símbolo suma y símbolo igual) y dos imágenes que representarán los números a sumar que irán cambiando automáticamente. Para ello se establecerá un identificador en cada uno de ellos y desde el archivo Java se accederá a los recursos *drawable* de la aplicación y se cambiará el atributo del fondo de la vista según sea necesario.

```
<ImageView
    android:id="@+id/primerNumeroASumar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/explicacionNumerosQuintoNivel"
    android:layout_marginTop="20dip" />

<ImageView
    android:id="@+id/segundoNumeroASumar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/primerNumeroASumar"
    android:layout_toRightOf="@+id/simbolo_suma" />

<ImageView
    android:id="@+id/simbolo_suma"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/explicacionNumerosQuintoNivel"
    android:layout_toRightOf="@+id/primerNumeroASumar"
    android:background="@drawable/suma" />

<ImageView
    android:id="@+id/simbolo_igual"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/segundoNumeroASumar"
    android:layout_toRightOf="@+id/segundoNumeroASumar"
    android:background="@drawable/igual" />
```

Figura 107: Archivo XML del quinto nivel del juego de los números

5.5 IMPLEMENTACIÓN

Una vez se ha explicado la estructura de actividades de la aplicación, el diseño de cada una de ellas y cómo se han desarrollado mediante el lenguaje XML a continuación se comentarán los aspectos más importantes de su implementación, por lo tanto esta sección no pretende ser un tutorial de desarrollo con Java para Android, sino una puesta en escena de los conocimientos adquiridos.

5.5.1 CÓDIGO COMPARTIDO POR LAS ACTIVIDADES

A lo largo del desarrollo de este software, ha habido código que se ha repetido, por lo tanto he creído conveniente que en lugar de hacer referencia una y otra vez lo mismo, era mejor comentarlo a parte.

○ *BOTÓN DE AYUDA*

Durante la aplicación es posible, si el jugador se siente perdido, mostrarle una pequeña ventana de diálogo en forma de ayuda. Para ello, una vez tengamos el archivo XML con la información a mostrar (ya sea qué hacer o cómo hacerlo) y habiendo configurado el atributo *onClick* del botón para que lance el método "lanzarAyuda" una vez se pulse el icono correspondiente en la pantalla, se pondrá crear esta ventana de ayuda.

```
//Método que abre un cuadro explicativo de diálogo
public void lanzarAyuda(View view){

    //Creamos el cuadro de diálogo lo enlazamos con el archivo xml y le ponemos un título
    dialogoAyuda = new Dialog(JuegoFormasPrimerNivel.this);
    dialogoAyuda.setContentView(R.layout.dialog_ayuda_formas_primer_nivel);
    dialogoAyuda.setTitle("¿Qué tenemos que hacer aquí?");

    //Enlazamos el botón de cerrar con el xml y le establecemos transparencia
    botonCerrarAyuda = (Button)dialogoAyuda.findViewById(R.id.botonCerrarAyudaFormasPrimerNivel);
    botonCerrarAyuda.getBackground().setAlpha(225);

    //Activamos el escuchador de eventos, cuando pulsemos se cierra el cuadro de diálogo
    botonCerrarAyuda.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            // TODO Auto-generated method stub
            dialogoAyuda.dismiss();
        }
    });

    //Mostramos el cuadro de diálogo
    dialogoAyuda.show();
}
```

Figura 108: Código Java para la ventana de ayuda

Después de haber importado las librerías *Dialog* y *DialogInterface* se instanciará el *dialog* con la instrucción *new Dialog*, a continuación mediante la instrucción *setContentView* definiremos el recurso XML que contiene el *layout* de la interfaz de usuario. Después, de forma opcional se le podrá poner un título y finalmente enlazamos el botón cerrar del archivo XML con una vista *Button* definida en el archivo Java y nombrada *botonCerrarAyuda* para así poder capturar el evento cuando el jugado haga click sobre este botón y cerrar el cuadro de diálogo.

○ *CRÓNÓMETRO CONTRARELOJ*

Primeramente en el método *onCreate* se accederá a la configuración de la aplicación y según estos datos se establecerá el tiempo total (en milisegundos).

```
//Accedemos a las preferencias para averiguar la dificultad elegida por el usuario
String dificultad = pref.getString("dificultad", "1");
int nivelDificultad = Integer.parseInt(dificultad);
//FÁCIL = 1
//NORMAL = 2
//DIFÍCIL = 3

if (nivelDificultad == 1){
    startTime = 21000;
}
else if (nivelDificultad == 2){
    startTime = 31000;
}
else if (nivelDificultad == 3){
    startTime = 60000;
}
```

Figura 109: Código Java para el cronómetro contrarreloj (I)

Se elegirá el intervalo de tiempo en el cual es decrementado el contador (1 segundo). Instanciamos la clase *ContadorJuegoFormas* (este extracto de código es del archivo Java del juego de las formas geométricas) y mediante la variable *tiempoAtrás*, la cual está enlazada con el archivo XML, se escribirá por pantalla el tiempo establecido inicialmente por el contador para iniciar la cuenta atrás.

```
//Creamos y arrancamos el contador
interval = 1000; //Decrementamos a intervalos de 1000ms, 1 seg
contadorCuentaAtras = new ContadorJuegoFormas(startTime, interval);
tiempoAtras.setText(tiempoAtras.getText() + String.valueOf(startTime / 1000));
contadorCuentaAtras.start();
```

Figura 110: Código Java para el cronómetro contrarreloj (II)

La clase que ejecuta la cuenta atrás tiene dos métodos, *onFinish* y *onTick*. El método *onFinish* se ejecutará una vez el tiempo llegue a cero, en ese momento se podrá enviar

un mensaje por pantalla al jugador y, o bien pasar al siguiente nivel si no nos encontramos en el último nivel, o si nos encontramos, finalizar el juego. El método *onTick* se ejecuta una vez por intervalo de tiempo. Se ha asignado la condición para cuando queden menos de 10 segundos para finalizar el tiempo, cambié el color del texto donde se muestra el tiempo al usuario a rojo a modo de advertencia.

```
//Aquí se ejecutarán los métodos del contador
public class ContadorJuegoFormas extends CountDownTimer {
    public ContadorJuegoFormas(long startTime, long interval) {
       super(startTime, interval);
    }
    @Override
    public void onFinish() {
        tiempoAtras.setText("¡GAME OVER!");
        DialogoFinalizarJuego();
    }

    @Override
    public void onTick(long millisUntilFinished) {
        if (millisUntilFinished>=10000){
            tiempoAtras.setText("00:" + millisUntilFinished / 1000);
        }
        else{
            tiempoAtras.setText("00:0" + millisUntilFinished / 1000);
            tiempoAtras.setTextColor(Color.RED);
        }
    }
}
```

Figura 111: Código Java para el cronómetro contrarreloj (III)

○ *CUADRO DE DIÁLOGO PARA GUARDAR LAS PARTIDAS*

Una vez se ha lanzado una partida, mediante el método *onActivityResult* se espera que retornen desde el último nivel del juego hasta el menú principal diferentes datos, como a que juego se ha jugado (-*mensajeControl*- para saber dónde guardar los puntos) y la puntuación.

```
@Override
protected void onActivityResult (int requestCode, int resultCode, final Intent data){
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode==1234 && resultCode==RESULT_OK && data!=null) {
        mensajeControl = data.getExtras().getString("mensajeControl");
```

Figura 112: Código Java para guardar la puntuación (I)

Antes de crear la ventana de dialogo para guardar la partida se obtendrá la fecha actual instanciando un objeto de la clase *SimpleDataFormat* y dándole un formato con la fecha actual del sistema en milisegundos.

```
//Establecer el formato de la fecha
SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy");
fecha = formatoFecha.format(new Date(System.currentTimeMillis()));
```

Figura 113: Código Java para guardar la puntuación (II)

Una vez hecho esto, se crea una ventana de diálogo con un *layout* previamente diseñado donde tendremos un *EditText* para introducir un nombre y dos botones, uno para guardar y otro para cancelar.

```
//Guardar la puntuación introduciendo un nombre, una puntuación y la fecha
custom = new Dialog (MenuPrincipal.this);
custom.setContentView(R.layout.dialog_guardar_puntuacion);
nombreJugador = (EditText)custom.findViewById(R.id.nombre);
Boton_Guardar = (Button)custom.findViewById(R.id.botonGuardar);
Boton_Cancelar = (Button)custom.findViewById(R.id.botonCancelar);
custom.setTitle("Guardar puntuación");
```

Figura 114: Código Java para guardar la puntuación (III)

Añadimos un escuchador de eventos a los botones y dependiendo de a que partida se ha jugado, se guardarán los puntos en un lugar u otro.

```
Boton_Guardar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        puntuacion = data.getExtras().getInt("puntuacion");
        nombre_jugador = nombreJugador.getText().toString();
        if (mensajeControl.equalsIgnoreCase("juegoFormasFinalizado")){
            almacenFormas.guardarPuntuacionFormas(puntuacion, nombre_jugador, fecha);
            lanzarPuntuacionesFormas();
        }
        else if (mensajeControl.equalsIgnoreCase("juegoVocalesFinalizado")){
            almacenLetras.guardarPuntuacionLetras(puntuacion, nombre_jugador, fecha);
            lanzarPuntuacionesLetras();
        }
        else if (mensajeControl.equalsIgnoreCase("juegoNumerosFinalizado")){
            almacenNumeros.guardarPuntuacionNumeros(puntuacion, nombre_jugador, fecha);
            lanzarPuntuacionesNumeros();
        }

        custom.dismiss();
    }
});

Boton_Cancelar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        custom.dismiss();
    }
});

custom.show();
```

Figura 115: Código Java para guardar la puntuación (IV)

○ *ANIMACIONES*

Las animaciones deben activarse en el método *onCreate* o en el método *onResume*. Habrá que importar las librerías *Animation* y *AnimationUtils* y cargar el archivo XML que tenemos en la carpeta anim de los recursos. Para que comience la animación se usa la instrucción *startAnimation*.

```
//Establecemos las animaciones del menú principal
Animation animacionTituloMenuPrincipal = AnimationUtils.loadAnimation(this, R.anim.animacion_titulo_menu_principal);
tituloMenuPrincipal.startAnimation(animacionTituloMenuPrincipal);

Animation animacionBotonJugar = AnimationUtils.loadAnimation(this, R.anim.animacion_boton_jugar);
botonJugar.startAnimation(animacionBotonJugar);

Animation animacionBotonPuntuaciones = AnimationUtils.loadAnimation(this, R.anim.animacion_boton_puntuaciones);
botonPuntuaciones.startAnimation(animacionBotonPuntuaciones);

Animation animacionBotonMutimedia = AnimationUtils.loadAnimation(this, R.anim.animacion_boton_multimedia);
botonMultimedia.startAnimation(animacionBotonMutimedia);

Animation animacionBotonConfiguracion = AnimationUtils.loadAnimation(this, R.anim.animacion_boton_configuracion);
botonConfiguracion.startAnimation(animacionBotonConfiguracion);

Animation animacionBotonAcercaDe = AnimationUtils.loadAnimation(this, R.anim.animacion_boton_acerca_de);
botonAcercaDe.startAnimation(animacionBotonAcercaDe);

Animation animacionBotonSalir = AnimationUtils.loadAnimation(this, R.anim.animacion_boton_salir);
botonSalir.startAnimation(animacionBotonSalir);
```

Figura 116: Código Java para las animaciones del menú principal

○ *REPRODUCCIÓN DE VÍDEOS*

Esta clase se implementará mediante bastantes métodos, por lo tanto se irán explicando por separado.

Una vez se han asignado las vistas *Button* que controlan la reproducción del vídeo (reproducir, pausar, subir o bajar volumen etc.) definidas en el archivo Java, con el archivo XML de esta actividad en el método *onCreate* a través de la instrucción *findViewById*, se asignará un escuchador de eventos a cada uno de esos botones y dependiendo en el estado en que se encuentren harán una tarea u otra, ya sea por ejemplo reproducir el vídeo, pausarlo, etc.

```
bPlay.setOnClickListener(new OnClickListener() {
    public void onClick(View view) {
       if (mediaPlayer != null) {
           if (pause) {
               //Habilitamos el botón pause cuando el vídeo esté ejecutándose
               bPause.setEnabled(true);
               //Deshabilitamos el botón play cuando el vídeo esté ejecutándose
               bPlay.setEnabled(false);
               mediaPlayer.start();
           }
           else {
               playVideo();
           }
       }
    }
});
```

Figura 117: Código Java para la reproducción de vídeos (I)

El método *playVideo* se encarga de obtener una ruta de reproducción, en este caso desde los recursos de la aplicación, pero también se podría obtener vía *streaming*. Además crea el objeto *MediaPlayer* y se le asigna una ruta y una superficie de visualización. También se le asignarán varios escuchadores de eventos que serán necesarios.

Una vez el vídeo ha sido preparado se situará la posición de reproducción a los milisegundos establecidos por la variable *savePos* y se configurará el volumen.

```
private void playVideo() {
    try {
        pause = false;
        mediaPlayer = new MediaPlayer();
        camino = Uri.parse("android.resource://com.example.proyectoeduca/"+R.raw.aprendelasformasgeometricas);
        mediaPlayer.setDataSource(this, camino);
        mediaPlayer.setDisplay(surfaceHolder);
        mediaPlayer.prepare();
        mediaPlayer.setOnBufferingUpdateListener(this);
        mediaPlayer.setOnCompletionListener(this);
        mediaPlayer.setOnPreparedListener(this);
        mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
        mediaPlayer.seekTo(savePos);
        mediaPlayer.setVolume(volumen, volumen);
    }
    catch (Exception e) {
        Toast.makeText(this, "ERROR: Este vídeo no existe o no está disponible", Toast.LENGTH_SHORT).show();
    }
}
```

Figura 118: Código Java para la reproducción de vídeos (II)

El método *onPrepared* implementa la interfaz *onPreparedListener*. Es invocado cuando el vídeo está listo para la reproducción. En este momento se conoce el alto y ancho del vídeo y se pone en reproducción.

```
public void onPrepared(MediaPlayer mediaplayer) {
    int mVideoWidth = mediaPlayer.getVideoWidth();
    int mVideoHeight = mediaPlayer.getVideoHeight();
    if (mVideoWidth != 0 && mVideoHeight != 0) {
        surfaceHolder.setFixedSize(mVideoWidth, mVideoHeight);
        //Deshabilitamos el botón play cuando el video esté activo
        bPlay.setEnabled(false);
        mediaPlayer.start();
    }
}
```

Figura 119: Código Java para la reproducción de vídeos (III)

Los dos primeros métodos de la siguiente imagen implementarán las interfaces *OnBufferingUpdateListener* y *OnCompletionListener*. El primero se utilizará para el porcentaje de obtención del búfer de reproducción y el segundo será invocado cuando el vídeo en reproducción llegue al final.

Los siguientes tres métodos implementan la interfaz *SurfaceHolder.Callback* para el control de la superficie de visualización.

```
public void onBufferingUpdate(MediaPlayer arg0, int percent) {}

public void onCompletion(MediaPlayer arg0) {}

public void surfaceCreated(SurfaceHolder holder) {
    playVideo();
}

public void surfaceChanged(SurfaceHolder surfaceholder, int i, int j, int k) {}

public void surfaceDestroyed(SurfaceHolder surfaceholder) {}
```

Figura 120: Código Java para la reproducción de vídeos (IV)

Es importante tener un control sobre el ciclo de vida de la actividad, ya que tendremos que eliminarla, pausarla o reproducirla.

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

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

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

Figura 121: Código Java para la reproducción de vídeos (V)

Cabe la posibilidad de que la actividad por cualquier motivo pueda detenerse, ya sea por la recepción de una llamada o por que se active el fondo de pantalla del dispositivo. En ese momento se guardará el tiempo en el cual se encontraba reproduciéndose el vídeo para que cuando volvamos, se pueda continuar dese ese preciso instante.

```
@Override
protected void onSaveInstanceState(Bundle guardarEstado) {
    super.onSaveInstanceState(guardarEstado);
        if (mediaPlayer != null) {
            int pos = mediaPlayer.getCurrentPosition();
            //Si lo hacemos desde los recursos, habría que cambiar este path por camino
            path = camino.toString();
            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");
        }
}
```

Figura 122: Código Java para la reproducción de vídeos (VI)

Y finalmente tenemos un método que mostrará mediante un mensaje *Toast* el nivel de volumen al que se encuentra el vídeo.

```
//Método que nos mostrará por pantalla el nivel de volumen del vídeo
private void comprobarNivelVolumen(float volumen) {

    if (volumen == 0.0){
        Toast.makeText(this, "Volumen SILENCIADO", Toast.LENGTH_SHORT).show();
    }
    else if (volumen >= 0.08 && volumen<=0.15){
        Toast.makeText(this, "Volumen al MÍNIMO", Toast.LENGTH_SHORT).show();
    }
    else if (volumen >= 0.45 && volumen<=0.55){
        Toast.makeText(this, "Volumen NORMAL", Toast.LENGTH_SHORT).show();
    }
    else if (volumen == 1.0){
        Toast.makeText(this, "Volumen al MÁXIMO", Toast.LENGTH_SHORT).show();
    }
}
```

Figura 123: Código Java para la reproducción de vídeos (VII)

- ***SERVICIO***

A lo largo de la aplicación se podrá activar o desactivar, ya sea de forma manual por el usuario a través de la configuración de la aplicación o de forma automática por la misma aplicación cuando sea necesario, el servicio que nos ofrece un audio de fondo.

Para ello la clase *AudioDeFondo* deberá extender de *Service* en lugar de *Activity*. Se definirá una variable de la clase *MediaPlayer* la cual será un audio de los recursos del sistema (carpeta *raw*).

Una vez arrancado el servicio -*startService(new Intent(this, AudioDeFondo.class))*- el primer método en ejecutarse será *onStartCommand* donde se establecerá que el audio se repita en bucle (*setLooping*) y que empiece (*start*). Si el dispositivo se quedase sin suficiente memoria para mantener el servicio activo y tuviese que destruirlo, mediante el comando *START_STICKY*, le diríamos al sistema que una vez volviera a tener memoria suficiente, creara otra vez el servicio.

Cuando queremos detener el servicio -*stopService(new Intent(this, AudioDeFondo.class))*- se accede al método *onDestroy* y este, parará la reproducción del objeto MediaPlayer.

```
public class AudioDeFondo extends Service {

    MediaPlayer audioDeFondo;

    @Override
    public void onCreate() {
        audioDeFondo = MediaPlayer.create(this,R.raw.audio_de_fondo);
    }

    @Override
    public int onStartCommand(Intent intenc, int flags, int idArranque) {
        audioDeFondo.setLooping(true); //Hacemos que el audio se repita en bucle
        audioDeFondo.start();
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        audioDeFondo.stop();
    }
    @Override
    public IBinder onBind(Intent intencion) {
        return null;
    }

}
```

Figura 124: Código Java para el servicio de música

○ *ALMACEN DE PUNTUACIONES*

La aplicación permitirá guardar las puntuaciones de dos formas distintas, en la memoria interna del dispositivo o en su memoria externa.

Para realizar esto y para que la lista de puntuaciones sea lo más independiente posible, se definirán tres interfaces para cada uno de los juegos y a su vez 3 clases que implementarán sus respectivas interfaces para acceder a la memoria interna y otras 3 clases para acceder a la memoria externa.

Como en principio el código es muy similar para los 3 juegos, a continuación se explicará cómo se almacenan los datos de las partidas del juego de las letras a modo de ejemplo.

Para la interfaz se indicarán los métodos que posteriormente se implementarán en las clases donde guardaremos la puntuación. El primero de ellos servirá para guardar la puntuación con los parámetros de puntos, nombre y fecha. El segundo devolverá una lista de un cierto tamaño de las partidas guardadas en memoria.

```
public interface AlmacenPuntuacionesLetras {

    public void guardarPuntuacionLetras(int puntos, String nombre, String fecha);
    public Vector<String> listaPuntuacionesLetras(int cantidad);
}
```

Figura 125: Interfaz juego de las letras

Por defecto los ficheros almacenados sólo son accesibles por la aplicación que los creó, es decir, no pueden ser leídos por otras aplicaciones y estos, por lo general, serán guardados en la carpeta *data/data/nombre_del_paquete_files*. Una vez se desinstale la aplicación, los ficheros serán borrados automáticamente.

Como se ha dicho, la clase que almacena y muestra la lista de las partidas jugadas implementará la interfaz.

```
public class AlmacenPuntuacionesFicheroInternoLetras implements AlmacenPuntuacionesLetras {
```

Figura 126: Almacenar y visualizar datos de la memoria interna (I)

Se definirá el nombre del fichero donde se almacenan los datos y el contexto en el cual actúa.

```
private static String FICHERO_LETRAS = "puntuacionesLetras.txt";
private Context context;

public AlmacenPuntuacionesFicheroInternoLetras(Context context) {
      this.context = context;
}
```

Figura 127: Almacenar y visualizar datos de la memoria interna (II)

En los métodos donde se guardan y visualizan datos es muy importante el control de errores, así pues, es obligatorio el uso de *try*/*catch* para el acceso a ficheros.

Para guardar datos se deberán pasar por argumentos un dato de tipo entero (puntos) y dos cadenas de caracteres (nombre y fecha). Se creará un objeto de tipo *FileOutputStream* (útiles para la escritura de ficheros de texto) y mediante la instrucción *openFileOutput* se abrirá un fichero para habilitar la escritura.

Una vez se tenga la cadena de texto con los puntos, el nombre y la fecha, dicha cadena se guardará en el fichero y este se cerrará (es muy importante cerrar el fichero una vez se ha leído o escrito en él).

```
@Override
public void guardarPuntuacionLetras(int puntos, String nombre, String fecha) {
    try {
        FileOutputStream f = context.openFileOutput(FICHERO_LETRAS, Context.MODE_APPEND);
        String texto = puntos + " " + nombre + " " + fecha + "\n";
        f.write(texto.getBytes());
        f.close();
    } catch (Exception e) {
        Log.e("Proyecto Educa", e.getMessage(), e);
    }

}
```

Figura 128: Almacenar y visualizar datos de la memoria interna (III)

Para para visualizar los puntos se procederá a leer un fichero instanciando un objeto *FileInputStream* (útiles para la lectura de ficheros de texto) que será un fichero donde tenemos los puntos almacenado pasándole por argumento la cantidad de datos a mostrar.

Se leerá línea a línea guardando los datos en un vector para luego visualizarlo.

```
@Override
public Vector<String> listaPuntuacionesLetras(int cantidad) {
    Vector<String> result = new Vector<String>();
    try {
            FileInputStream f = context.openFileInput(FICHERO_LETRAS);
            BufferedReader entrada = new BufferedReader(new InputStreamReader(f));
            int n = 0;
            String linea;
            do {
              linea = entrada.readLine();
                if (linea != null) {
                  result.add(linea);
                    n++;
                }
            } while (n < cantidad && linea != null);
            f.close();
    } catch (Exception e) {
            Log.e("Proyecto Educa", e.getMessage(), e);
    }
    return result;
}
```

Figura 129: Almacenar y visualizar datos de la memoria interna (IV)

Escribir o leer en un fichero en la memoria externa difiere en algunos matices respecto a lo ya explicado en la memoria interna.

El directorio donde se escribirá o leerá el fichero será diferente, además se deberá comprobar el estado de la memoria (si la instrucción *Environment.MEDIA_MOUNTED* devuelve un valor verdadero, es posible escribir y leer de una memoria externa).

```
private static String FICHERO_LETRAS = Environment.getExternalStorageDirectory() + "/puntuacionesLetras.txt";
String stadoSD = Environment.getExternalStorageState();
```

Figura 130: Almacenar y visualizar datos de la memoria externa (I)

```
@Override
public void guardarPuntuacionLetras(int puntos, String nombre, String fecha) {

    try {
          if (!stadoSD.equals(Environment.MEDIA_MOUNTED)) {
              Toast.makeText(context, "IMPOSIBLE GUARDAR EN LA MEMORIA EXTERNA DEL DISPOSITIVO", Toast.LENGTH_LONG).show();
              return;
          }
          FileOutputStream f = new FileOutputStream(FICHERO_LETRAS, true);
          String texto = puntos + " " + nombre + " " + fecha + "\n";
            f.write(texto.getBytes());
            f.close();
        } catch (Exception e) {
            Log.e("ProyectoEduca", e.getMessage(), e);
        }
}
```

Figura 131: Almacenar y visualizar datos de la memoria externa (II)

```
@Override
public Vector<String> listaPuntuacionesLetras(int cantidad) {

    Vector<String> result = new Vector<String>();
    try {
        if (!stadoSD.equals(Environment.MEDIA_MOUNTED) && !stadoSD.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
            Toast.makeText(context, "IMPOSIBLE LEER LA MEMORIA EXTERNA DEL DISPOSITIVO", Toast.LENGTH_LONG).show();
            return result;
        }
        FileInputStream f = new FileInputStream(FICHERO_LETRAS);
        BufferedReader entrada = new BufferedReader(new InputStreamReader(f));
        int n = 0;
        String linea;
        do {
            linea = entrada.readLine();
            if (linea != null) {
                result.add(linea);
                n++;
            }
        } while (n < cantidad && linea != null);
        f.close();
    } catch (Exception e) {
            Log.e("ProyectoEduca", e.getMessage(), e);
    }
    return result;
}
```

Figura 132: Almacenar y visualizar datos de la memoria externa (III)

○ *RECONOCIMIENTO DE PATRONES DE DIBUJO*

En diferentes juegos será necesario que el jugador dibuje (con su dedo o con un lápiz táctil) una forma geométrica, letra o número que se le indique. El proceso para realizar esto es similar para los tres casos expuestos, por lo tanto se explicará cómo reconocer formas geométricas a modo de ejemplo.

Primero de todo a través del emulador se debe crear una biblioteca *Gestures* mediante la aplicación *Gestures Builder*. Esta biblioteca se hará dibujando nosotros mismos los patrones que queramos que sean reconocidos y dándoles un nombre para poder identificarlos.

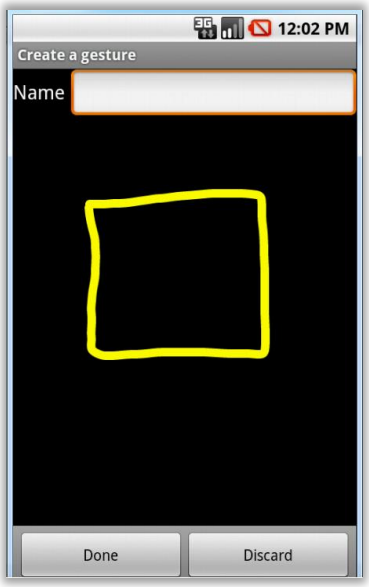


Figura 133: Creación de una biblioteca *Gesture*

Una vez acabada, el archivo que se ha generado, se descargará y se copiará en el directorio *res/raw/(nombre_biblioteca_gesture)* de los recursos de la aplicación para así tener acceso a él.

Para tener ese acceso se definiría una variable de tipo *GestureLibrary* (en mi caso la he llamado *librería*) y se accedería a ella mediante la instrucción *fromRawResource* en el método *onCreate*, para que el archivo *Gesture* se cargue una vez la actividad se inicia.

```
//Cargamos la librería de gestures
libreria = GestureLibraries.fromRawResource(this, R.raw.gestures_formas);
if (!libreria.load()) {
    finish();
}
```

Figura 134: Usar una biblioteca *Gesture* (I)

Ahora tendríamos en el *layout* de la actividad una vista de tipo *GestureOverlayView* que es donde el jugador dibujará, por lo tanto esa vista tendrá que ser enlazada con el archivo Java en una variable del mismo tipo y aplicarle un escuchador de eventos para reconocer cuando se está dibujando, además de implementar la interfaz *OnGesturePerformedListener* en esta clase.

```
//Evento para las gestures de las formas
gesturesView =(GestureOverlayView) findViewById(R.id.gestures_formas);
gesturesView.addOnGesturePerformedListener(this);
```

Figura 135: Usar una biblioteca *Gesture* (II)

El último paso sería saber cuándo el jugador ha acertado dibujando (en este caso una forma geométrica) y cuándo no.

Sabiendo que figura tenemos que buscar (en este ejemplo es una figura, valdría igual para una letra o número creando la biblioteca correspondiente) cuando el jugador dibuje, el escuchador de eventos lanzará el método *onGesturePerformed*. Este método creará una lista con las formas que tenemos en la librería de mayor parecido al dibujo hecho por el jugador al menor. Se cogerá el primer elemento de esta lista y se comparará con la figura buscada, si coinciden, la figura buscada será la dibujada y el jugador habrá acertado, sino el jugador habrá fallado.

```
//Método que detecta que figura hemos dibujado y si es la buscada
public void onGesturePerformed(GestureOverlayView ov, Gesture gesture) {
    ArrayList<Prediction> predictions=libreria.recognize(gesture);
    if (predictions.size()>0){
        String comando = predictions.get(0).name;
        if (comando.equalsIgnoreCase("cuadrado")){
            if (formaADibujar.equalsIgnoreCase("cuadrado")){
                DibujoAcertdo();
                EstablecerFormaParaDibujar();
            }
            else{
                DibujoFallado();
                EstablecerFormaParaDibujar();
            }
        }
        else if (comando.equalsIgnoreCase("rectangulo")){
            if (formaADibujar.equalsIgnoreCase("rectangulo")){
                DibujoAcertdo();
                EstablecerFormaParaDibujar();
            }
            else{
                DibujoFallado();
                EstablecerFormaParaDibujar();
            }
        }
        else if (comando.equalsIgnoreCase("triangulo")){
            if (formaADibujar.equalsIgnoreCase("triangulo")){
                DibujoAcertdo();
                EstablecerFormaParaDibujar();
            }
            else{
                DibujoFallado();
                EstablecerFormaParaDibujar();
            }
        }
```

Figura 136: Reconocimiento de una forma *Gesture*

○ *TRANSLADAR LAS PUNTUACIONES DESDE EL ÚLTIMO NIVEL HASTA EL MENÚ PRINCIPAL*

Una de las dificultades que me encontré durante el desarrollo de esta aplicación fue averiguar cómo guardar los puntos una vez el jugador llegara al último nivel del juego que en ese momento estuviera jugando.

La idea inicial fue lanzar *intents*, añadiendo como dato la puntuación obtenida en ese nivel (sumada al anterior si lo hubiere) para ir abriendo niveles (actividades) y una vez superado el juego, abrir de nuevo una nueva actividad para situar al jugador en el menú principal. Eso suponía un problema que consistía en que no se cerraban las actividades que íbamos abriendo nivel tras nivel y se perdía el control de la aplicación, por lo tanto se pensó en otra solución más eficiente.

Se avanzará actividad a actividad sumando la puntuación obtenida y enviándola al siguiente nivel, pero una vez llegados al último de ellos, en vez de volver a lanzar una actividad mediante un *intent* para llegar al menú principal, se volverá hacia atrás, devolviendo la puntuación final, donde la actividad anterior la recibirá y la enviará otra vez hacia atrás cerrando la actividad actual. De esta forma jamás se perderá el control de actividades de la aplicación. Esto se conseguirá añadiendo información a un *intent* como se ha hecho hasta ahora, pero lanzando la actividad esperando un resultado.

```
//Lanza el siguiente nivel del juego
private void lanzarSiguienteNivel(){
    Intent i = new Intent(this,JuegoVocalesQuintoNivel.class);
    i.putExtra("puntuacion", puntuacion);
    startActivityForResult(i, 1234);
}
```

Figura 137: Código Java para lanzar un *intent* que espera un resultado

A su vez, cuando los puntos se reciban, lo harán mediante el método *onActivityResult*.

```
//Devolvemos la información de la puntuación actividad a actividad hasta llegar al menú
@Override
protected void onActivityResult (int requestCode, int resultCode, Intent data){
    if (requestCode==1234 && resultCode==RESULT_OK) {
        String mensajeControl = data.getExtras().getString("mensajeControl");
        int puntuacion = data.getExtras().getInt("puntuacion");

        if (mensajeControl.equalsIgnoreCase("juegoVocalesFinalizado")){
            Intent intent = new Intent();
            intent.putExtra("mensajeControl","juegoVocalesFinalizado");
            intent.putExtra("puntuacion", puntuacion);
            setResult(RESULT_OK, intent);
            finish();
        }
    }
}
```

Figura 138: Método que recibe el resultado y devuelve información

○ *ASIGNAR SONIDOS A BOTONES*

El primer nivel de los juegos principales (formas geométricas, letras y números) consiste en un concepto más "teórico" que "práctico". El jugador vera una actividad con varios botones que representan formas, letras o número a través de sus respectivas imágenes y tocándolos aprenderá a identificar de cual se trata mediante un audio que tendrá asociado cada uno de ellos.

Cómo la forma de implementar esto es similar en los tres tipos de juegos, se procederá a explicarlo en uno de ellos.

Antes de nada será necesario disponer de los archivos de audio (en mi caso he utilizado el software libre *Audacity* como medio de grabación) en los recursos de la aplicación, más concretamente en el directorio *res/raw/(nombre_archivo_audio).mp3*

Ya en la clase donde implementamos el primer nivel del juego, se definirán las variables que identifican los diferentes sonidos que se van a utilizar, así mismo también se definirá la clase *SoundPool* que será la que se utilice para la reproducción de audio (principalmente usada para reproducción de audios cortos y de tamaño máximo de 1Mb)

Cabe añadir que no solamente en el primer nivel de los juegos mencionados se usa la clase *SoundPool* para la reproducción de audio de forma rápida, en varias clases más será necesario su uso.

```
//Declaramos los archivos de sonido
SoundPool soundPool;
int idForma_Cuadrado;
int idForma_Rectangulo;
int idForma_Triangulo;
int idForma_Circulo;
int idForma_Rombo;
```

Figura 139: Uso de la clase *SoundPool* en botones (I)

En el método *onCreate* de la clase será donde se carguen los archivos de audio.

```
//Carga del audio de cada una de las formas geométricas
soundPool = new SoundPool( 5, AudioManager.STREAM_MUSIC , 0);
idForma_Cuadrado = soundPool.load(this, R.raw.cuadrado, 0);
idForma_Rectangulo = soundPool.load(this, R.raw.rectangulo, 0);
idForma_Triangulo = soundPool.load(this, R.raw.triangulo, 0);
idForma_Circulo = soundPool.load(this, R.raw.circulo, 0);
idForma_Rombo = soundPool.load(this, R.raw.rombo, 0);
```

Figura 140: Uso de la clase *SoundPool* en botones (II)

Cuando se instancia un objeto *SoundPool*, en el constructor hay que establecer tres parámetros. El primero de ellos indica cuantas pistas son posibles reproducir de manera simultánea. El segundo es el tipo de *stream* del audio. El tercero es la calidad de la reproducción, aunque actualmente no se implementa.

Cada botón diseñado en el *layout* de esta actividad tiene asociado en su atributo *onClick* un método diferente que será ejecutado al ser pulsado.

```
//Activamos el sonido de las diferentes letras al pulsar los botones
public void Sonido_Cuadrado (View view){
    soundPool.play(idForma_Cuadrado, 1, 1, 1, 0, 1);
}

public void Sonido_Rectangulo (View view){
    soundPool.play(idForma_Rectangulo, 1, 1, 1, 0, 1);
}

public void Sonido_Triangulo (View view){
    soundPool.play(idForma_Triangulo, 1, 1, 1, 0, 1);
}

public void Sonido_Circulo (View view){
    soundPool.play(idForma_Circulo, 1, 1, 1, 0, 1);
}

public void Sonido_Rombo (View view){
    soundPool.play(idForma_Rombo, 1, 1, 1, 0, 1);
}
```

Figura 141: Uso de la clase *SoundPool* en botones (III)

En dichos métodos será donde se reproducirá el sonido, para ello se debe usar el método *play* e indicarle el identificador de pista, el volumen para el canal izquierdo y derecho, la prioridad (por si en algún momento la cantidad de pistas reproducidas en un instante es superior al permitido, detener la pista con la prioridad más baja), el número de repeticiones y el ratio de reproducción.

Una vez hecho todo esto, cada vez que se pulse un botón se escuchará su audio asociado.

○ *GRABACIÓN DE AUDIO A TRAVÉS DEL MICRÓFONO*

En su segundo nivel, en los juegos de las letras y los números se dará la posibilidad de grabar la voz del jugador.

Se comienza declarando las variables necesarias. Las variables *MediaRecorder* y *MediaPlayer* se utilizarán para la grabación y reproducción respectivamente, la variable *fichero* identifica el nombre del fichero donde se guardará la grabación y por

último la constante *LOG_TAG* será utilizada como etiqueta para identificar de donde proviene un posible fallo.

```
//Declaramos las variables usadas para grabar
private static final String LOG_TAG = "Grabadora";
private MediaRecorder mediaRecorder;
private MediaPlayer mediaPlayer;
private static String fichero = Environment.getExternalStorageDirectory().getAbsolutePath()+"/audio.3gp";
```

Figura 142: Grabación de audio (I)

En el método *onCreate* se asociarán las vistas de tipo botón con sus respectivas vistas en el layout de esta actividad para poder utilizarlas desde código Java, además se deshabilitará el uso de botón detener y reproducir hasta que no haya una grabación.

```
bGrabar = (Button) findViewById(R.id.boton_grabar_vocales_nivel_2);
bDetener = (Button) findViewById(R.id.boton_parar_vocales_nivel_2);
bReproducir = (Button) findViewById(R.id.boton_play_vocales_nivel_2);
bDetener.setEnabled(false);
bReproducir.setEnabled(false);
```

Figura 143: Grabación de audio (II)

Los tres botones del layout (grabar, detener y reproducir) tendrán configurado su atributo *onClick* para que una vez pulsados accedan a métodos que permitan grabar, detener y reproducir audio respectivamente.

```
public void grabar(View view) {

    Toast.makeText(this, "GRABANDO", Toast.LENGTH_SHORT).show();

    //Cuando grabamos deshabilitamos la posibilidad de grabar de nuevo o reproducir
    bGrabar.setEnabled(false);
    bDetener.setEnabled(true);
    bReproducir.setEnabled(false);

    mediaRecorder = new MediaRecorder();
    mediaRecorder.setOutputFile(fichero);
    mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.
    try {
        mediaRecorder.prepare();
    } catch (IOException e) {
        Log.e(LOG_TAG, "Fallo en grabación");
    }
    mediaRecorder.start();
}
```

Figura 144: Grabación de audio (III)

Una vez se esté grabando un audio, solo estará habilitado el botón de detener, los otros dos (grabar y reproducir) mientras estemos grabando no se podrán pulsar.

Una vez instanciado un objeto de la clase *MediaRecorder* se establece donde será guardada la grabación (*setOutputFile*), el dispositivo que se utiliza como fuente del sonido, en este caso, el micrófono del dispositivo (*setAudioSource*) , el formato de salida del audio, el cual se ha establecido como formato 3GPP (*setOutputFormat*) y su codificación (*setAudioEncoder*). Con la instrucción *start*, comienza la grabación.

Finalizada la grabación se pulsará el botón para detenerla, en ese momento, los botones de grabar y reproducir serán los que estén habilitados, para sobrescribir la grabación o escucharla.

Ahora sólo será necesario parar la grabación (*stop*) y liberar los recursos que se están utilizando (*release*).

```
public void detenerGrabacion(View view) {

    Toast.makeText(this, "GRABACIÓN DETENIDA", Toast.LENGTH_SHORT).show();

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

     mediaRecorder.stop();
     mediaRecorder.release();
 }
```

Figura 145: Grabación de audio (IV)

Por último, se podrá reproducir la grabación que acabamos de realizar instanciando un objeto de la clase *MediaPlayer*, indicando de que fichero se trata (*setDataSource*) y por último reproduciendo el sonido (*prepare* y *start*).

```
public void reproducir(View view) {

    Toast.makeText(this, "REPRODUCIENDO", Toast.LENGTH_SHORT).show();

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

Figura 146: Grabación de audio (V)

5.5.2 MENÚ PRINCIPAL

Primeramente la clase *MenuPrincipal* deberá ser una actividad y se definirán todas las vistas que se van a utilizar, desde el título y los botones del menú, los objetos donde se guardarán las diferentes puntuaciones y las variables utilizadas para esto.

```
public class MenuPrincipal extends Activity {

    //Declaracion de vistas
    TextView tituloMenuPrincipal;
    Button botonJugar;
    Button botonPuntuaciones;
    Button botonMultimedia;
    Button botonConfiguracion;
    Button botonAcercaDe;
    Button botonSalir;

    //Declaramos el objeto para guardar las puntuaciones
    public static AlmacenPuntuacionesFormas almacenFormas;
    public static AlmacenPuntuacionesLetras almacenLetras;
    public static AlmacenPuntuacionesNumeros almacenNumeros;

    //Declaración de la variable para acceder a las preferencias de la aplicación
    SharedPreferences pref;

    //Variables utilizadas en el cuatro de dialogo para introducir un nombre en la partida
    Dialog custom;
    EditText nombreJugador;
    Button Boton_Guardar;
    Button Boton_Cancelar;
    String nombre_jugador;
    int puntuacion;
    String mensajeControl;
    String fecha;
```

Figura 147: Código Java del menú principal (I)

Una vez definidas las variables, en el método *onCreate* establecemos el *layout* de la actividad llamando al método *setContentView* accediendo a los recursos donde se ha diseñado el archivo XML. Ahora llamando al método *findViewById*, se podrá establecer una relación entre las vistas previamente definidas y las vistas del archivo XML para poder trabajar con ellas desde código Java.

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

    //Enlazamos el archivo java con el xml
    tituloMenuPrincipal = (TextView)findViewById(R.id.Titulo_menu_principal);
    botonJugar = (Button)findViewById(R.id.Boton_jugar);
    botonPuntuaciones = (Button)findViewById(R.id.Boton_puntuaciones);
    botonMultimedia = (Button)findViewById(R.id.Boton_multimedia);
    botonConfiguracion = (Button)findViewById(R.id.Boton_configuracion);
    botonAcercaDe = (Button)findViewById(R.id.Boton_acercade);
    botonSalir = (Button)findViewById(R.id.Boton_salir);

    //Establecemos transparencia a los botones
    botonJugar.getBackground().setAlpha(225);
    botonPuntuaciones.getBackground().setAlpha(225);
    botonMultimedia.getBackground().setAlpha(225);
    botonConfiguracion.getBackground().setAlpha(225);
    botonAcercaDe.getBackground().setAlpha(225);
    botonSalir.getBackground().setAlpha(225);
```

Figura 148: Código Java del menú principal (II)

El menú, el cual aparece una vez se ha pulsado el botón físico "menú" del dispositivo, se ha añadido de forma opcional, pero ha sido interesante aprender cómo implementarlo. Para ello se necesitan dos métodos, el primero de ellos es *onCreateOpcionsMenu*, donde mediante el archivo XML que se ha diseñado para el menú, se mostrará de forma visual por pantalla. El segundo método *onOptionsItemSelected* se usará para cuando se pulse a un botón del menú, este realice una acción, en este caso, llama a un método para abrir las actividades de las diferentes secciones de la aplicación.

```java
//Método con el que creamos el menú cuando pulsamos la tecla de menú en el dispositivo
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
    return true; /** true -> el menú ya está visible */
}

//método que cuando pulsamos un botón del menú creado, nos lleva a la venta seleccionada
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.juega:
            lanzarJuego(null);
            break;
        case R.id.puntua:
            lanzarPuntuaciones(null);
            break;
        case R.id.multim:
            lanzarMultimedia(null);
            break;
        case R.id.config:
            lanzarConfiguracion(null);
            break;
        case R.id.acercaDe:
            lanzarAcercaDe(null);
            break;
        case R.id.salir:
            lanzarSalir(null);
            break;
    }
    return true;
}
```

Figura 149: Código Java del menú principal (III)

```java
//Método que lanza la actividad de Multimedia
public void lanzarMultimedia (View view){
    Intent i = new Intent(this,Multimedia.class);
    startActivity(i);
}

//Método que lanza la actividad de Configuración
public void lanzarConfiguracion (View view){
    Intent i = new Intent(this,Preferencias.class);
    startActivity(i);
}

//Método que lanza la actividad de acerca de
public void lanzarAcercaDe (View view){
    Intent i = new Intent(this,AcercaDe.class);
    startActivity(i);
}
```

Cada uno de los botones del menú principal en su archivo XML tendrá configurado el atributo *onClick* para lanzar un método mediante *Intents* para que este direccione al jugador hacia las diferentes secciones de la aplicación.

Figura 150: Código Java del menú principal (IV)

En ciertos momentos será necesario tener en cuenta el ciclo de vida de la actividad. Justo antes de que la actividad sea cargada se revisan las opciones de configuración del usuario para activar o no la música de fondo, o averiguar dónde guardar la puntuación.

```
@Override
protected void onResume() {
    super.onResume();

    /*Cuando la aplicación está a punto de estar activa, comprobamos si está habilitada o
    deshabilitada la música de fondo*/
    pref = PreferenceManager.getDefaultSharedPreferences(this);
    if (pref.getBoolean("musica", true)==true){
        startService(new Intent(MenuPrincipal.this,AudioDeFondo.class));
    }
    else{
        stopService(new Intent(MenuPrincipal.this,AudioDeFondo.class));
    }

    //Dependiendo de las preferencias los puntos seguardaran en la memoria interna o externa
    if (pref.getString("memoria", "1").equals("0")) {
        almacenFormas = new AlmacenPuntuacionesFicheroInternoFormas(this);
        almacenLetras = new AlmacenPuntuacionesFicheroInternoLetras (this);
        almacenNumeros = new AlmacenPuntuacionesFicheroInternoNumeros (this);
    }
    else if (pref.getString("memoria", "1").equals("1")) {
        almacenFormas = new AlmacenPuntuacionesFicheroExternoFormas(this);
        almacenLetras = new AlmacenPuntuacionesFicheroExternoLetras(this);
        almacenNumeros = new AlmacenPuntuacionesFicheroExternoNumeros(this);
    }

}
```

Figura 151: Código Java del menú principal (V)

Además cuando la actividad sea cerrada o finalizada, se deberá detener el audio de fondo.

```
@Override
protected void onDestroy() {
    super.onDestroy();

    //Cuando la aplicación es destruida se para el audio de fondo
    stopService(new Intent(MenuPrincipal.this,AudioDeFondo.class));
}
```

Figura 152: Código Java del menú principal (VI)

Puede ser que el usuario quiera salir o cerrar la aplicación de dos formas, pulsando el botón "salir" o pulsando el botón físico de su dispositivo "atrás". Si lo hace mediante esta última, se capturará el evento y se advertirá al usuario antes de realizar ninguna acción. Si es de la primera forma, directamente se advertirá al usuario. Para realiza dicha advertencia se creará una ventana de diálogo de tipo *AlertDialog* con dos botones, uno para confirmar y otro para denegar la acción a realizar.

```java
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    super.onKeyDown(keyCode, event);
    if ((keyCode == KeyEvent.KEYCODE_BACK)) {
        mensajeSalir();

        // Si el listener devuelve true, significa que el evento esta procesado, y nadie debe hacer nada mas
        return true;
    }
    //para las demas cosas, se reenvia el evento al listener habitual
    return super.onKeyDown(keyCode, event);
}

//Diálogo de advertencia al usuario para ver si desea salir de la aplicación
public void mensajeSalir(){
    AlertDialog.Builder dialogoSalir = new AlertDialog.Builder(this);
    dialogoSalir.setMessage("¿Seguro que quieres salir?");
    dialogoSalir.setPositiveButton("Si", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            // Do nothing but close the dialog
            finish();
            dialog.dismiss();
        }
    });

    dialogoSalir.setNegativeButton("No", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            // Do nothing
            dialog.dismiss();
        }
    });

    AlertDialog alert = dialogoSalir.create();
    alert.show();
}
```

Figura 153: Código Java del menú principal (VII)

5.5.3 ACERCA DE

Esta actividad es bastante sencilla. Se establece el *layout* de un archivo XML que se ha diseñado previamente accediendo a los recursos con el método *setContentView* y únicamente hay un botón que realiza una función (volver atrás).

```java
public class AcercaDe extends Activity{

    //Declaración de las vistas
    Button botonCerrar;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.acercade);

        //Enlazamos el archivo java con el xml
        botonCerrar = (Button)findViewById(R.id.Boton_cerrar);

        //Establecemos transparencia al botón
        botonCerrar.getBackground().setAlpha(225);
    }

    //Método para cerrar la ventana actual
    public void lanzarCerrar(View view){
        finish(); //Finish() vuelve a la actividad anterior matando la actual
    }
}
```

Figura 154: Código Java de la actividad "Acerca de"

5.5.4 CONFIGURACIÓN

Esta clase deberá extender de *PreferenceActivity* en lugar de *Activity* además se accederá a los recursos de la carpeta *xml* donde habíamos diseñado el archivo de preferencias para establecer el *layout* de la Actividad.

```
public class Preferencias extends PreferenceActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferencias);
    }
}
```

Figura 155: Código Java de la actividad "Preferencias"

Cabe añadir que aunque la instrucción *addPreferenceFromResource* debería ser evitada ya que no está en uso en las versiones más actuales de Android, para el uso que se le va a dar en esta aplicación y versión de la misma con la que se está trabajando, puede ser utilizada.

5.5.5 MULTIMEDIA

En el método *onCreate* se establece el *layout* (accediendo a la carpeta *layout* de los recursos y utilizando su archivo XML) y se enlazan las vistas que se han definido al principio de la clase con las vistas del archivo XML, para así poder trabajar con ellas desde código Java.

En esta actividad básicamente el jugador se encuentra con diferentes botones que lanzarán los archivos multimedia (vídeos).

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

    //Enlazamos el archivo java con el xml
    botonCerrar = (Button)findViewById(R.id.boton_Cerrar_Multimedia);
    botonVocales = (Button)findViewById(R.id.boton_Vocales);
    botonColores = (Button)findViewById(R.id.boton_Colores);
    botonAyuda = (Button)findViewById(R.id.botonAyudaMultimedia);


    //Establecemos transparencia al botón
    botonCerrar.getBackground().setAlpha(225);
    botonVocales.getBackground().setAlpha(225);
    botonColores.getBackground().setAlpha(225);
}
```

Figura 156: Código Java de la actividad "Multimedia" (I)

Cada uno de los botones tendrá asociado un método en su atributo *onClick,* el cual primeramente detendrá el audio de fondo si este está activado para que no se solape con el audio del vídeo y a continuación a través de un *Intent*, transmitiremos un dato para identificar de que vídeo se trata y lanzaremos la actividad de *VideoMultimedia* para visualizarlo (cómo visualizar un vídeo ha sido explicado en el punto *5.1.1 Código compartido por las actividades*).

```
//Método que lanza el vídeo de las vocales
public void lanzarVideoVocalesMultimedia (View view){
    stopService(new Intent(this,AudioDeFondo.class));
    Intent i = new Intent(this,VideoMultimedia.class);
    i.putExtra("videoTransmitido", "aprendelasvocales");
    startActivity(i);
}
```

Figura 157: Código Java de la actividad "Multimedia" (II)

Cuando la actividad esté a punto de arrancar se comprobará accediendo a las preferencias si es necesario mantener la música activa (si venimos del menú principal) o volver a activar la música (si venimos de visualizar un vídeo).

```
@Override
protected void onResume() {
    super.onResume();

    /*Cuando la aplicación está a punto de estar activa, comprobamos si está habilitada o
    deshabilitada la música de fondo*/
    pref = PreferenceManager.getDefaultSharedPreferences(this);
    if (pref.getBoolean("musica", true)==true){
        startService(new Intent(this,AudioDeFondo.class));
    }
    else{
        stopService(new Intent(this,AudioDeFondo.class));
    }

}
```

Figura 158: Código Java de la actividad "Multimedia" (III)

5.5.6 PUNTUACIONES

Para acceder a visualizar las puntuaciones, previamente deberemos seleccionar cual deseamos ver a través de un menú.

Esta actividad es muy sencilla, se tratará de 3 botones desde donde se lanzarán las actividades para ver las puntuaciones de los diferentes juegos y un botón de ayuda. Se ha configurado el atributo *onClick* para que mediante un método, los botones que lanzan las puntuaciones lo hagan a través de un *Intent*.

```
//Lanzamos la actividad para visualizar los puntos del juego de las letras
public void lanzarPuntuacionesLetras (View view){
    Intent i = new Intent(this,PuntuacionesLetras.class);
    startActivity(i);
}
```

Figura 159: Extracto del código Java del menú de puntuaciones

La actividad que muestra las puntuaciones heredará de *ListActivity* ya que ha de visualizar una *ListView*. Además se llamará a *setListAdapter* para indicar el adaptador con la lista de elementos a visualizar. Mediante *ArrayAdapter* se crean las vistas del ListView a partir de los datos almacenados en un *array*, su constructor tiene tres parámetros. El primero es el contexto, el cual será el de esta actividad. El segundo el *Layout* utilizado para representar cada elemento de la lista, dicho *Layout* se ha creado en la carpeta *res/layout*. El último es un *array* con los *strings* a mostrar.

```
public class PuntuacionesLetras extends ListActivity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.puntuaciones);
        setListAdapter(
                new ArrayAdapter<String>(this,
                        R.layout.elemento_lista_letras,
                        R.id.titulo_letras,
                        MenuPrincipal.almacenLetras.listaPuntuacionesLetras(10)));
    }
}
```

Figura 160: Código Java para visualizar puntuaciones del juego de letras

5.5.7 JUGAR

Aquí nos encontramos otro menú desde el cual a través de diferentes métodos lanzamos las actividades de los juegos.

Hay que destacar que una vez lanzada la actividad de un juego, se esperan ciertos datos (puntuaciones y juego jugado) como ya se ha explicado en el punto *5.5.1 Código compartido por las actividades*.

```
//Método para lanzar la actividad del juego de las letras
public void lanzarJuegoLetras(View view){
    stopService(new Intent(this,AudioDeFondo.class));
    Intent i = new Intent(this,VideoMultimediaJuegoVocales.class);
    startActivityForResult(i, 1234);
}

//Método para lanzar la actividad del juego de los numeros
public void lanzarJuegoNumeros(View view){
    stopService(new Intent(this,AudioDeFondo.class));
    Intent i = new Intent(this,VideoMultimediaJuegoNumeros.class);
    startActivityForResult(i, 1234);
}

//Método para lanzar la actividad de los mini-juegos
public void lanzarMiniJuegos(View view){
    Intent i = new Intent(this,MiniJuegos.class);
    startActivityForResult(i, 1234);
}
```

Figura 161: Código Java del menú de juegos

5.5.7.1 MINI-JUEGOS

Una vez más esta actividad se trata de un menú con tres botones que nos dirigen a tres actividades diferentes (juego de las formas geométricas, juego de pintar y juego de la nave). Una de ellas requerirá que se retornen ciertos datos (puntuación e identificación del juego), por lo tanto se lanzará la actividad esperando resultados.

```
//Método que lanza el juego de las formas
public void lanzarMiniJuegoFormas (View view){
    stopService(new Intent(this,AudioDeFondo.class));
    Intent i = new Intent(this,VideoMultimediaJuegoFormas.class);
    startActivityForResult(i, 1234);
}
```

Figura 162: Código Java del menú de mini-juegos (I)

Las dos restantes serán lanzadas de forma normal.

```
//Método que lanza el juego de pintar
public void lanzarMiniJuegoPintar (View view){
    Intent i = new Intent(this,JuegoPintar.class);
    startActivity(i);
}

//Método que lanza el juego de la nave
public void lanzarMiniJuegoNave (View view){
    stopService(new Intent(this,AudioDeFondo.class));
    Intent i = new Intent(this,JuegoNave.class);
    startActivity(i);
}
```

Figura 163: Código Java del menú de mini-juegos (II)

○ *FORMAS GEOMÉTRICAS*

Una vez lanzado el vídeo (el cual ya se ha explicado su código) en el primer nivel nos encontramos ante una actividad en la cual acceder a los recursos donde tenemos audios guardados es lo más importante. Al jugador se le presentan 5 botones cada uno representado por una forma geométrica y al pulsar sobre alguno de ellos, un audio informará sobre cual es (para ver como se ha implementado mirar el punto *5.1.1 Código compartido por las actividades*)

Cuando se quiera pasar al siguiente nivel, al pulsar sobre la flecha se abrirá un cuadro de diálogo donde se advertirá si el usuario quiere continuar.

```
//Método para avanzar al nivel 2
public void lanzarSiguienteNivel(View view){

    AlertDialog.Builder dialogoSalir = new AlertDialog.Builder(this);
    dialogoSalir.setMessage("¿Seguro que quieres pasar al nivel 2\n(a partir de ahora ganarás puntos en las pruebas)");
    dialogoSalir.setPositiveButton("Si", new DialogInterface.OnClickListener() {

        public void onClick(DialogInterface dialog, int which) {
            // Do nothing but close the dialog
            confirmacionSiguienteNivel();
            dialog.dismiss();
        }
    });

    dialogoSalir.setNegativeButton("No", new DialogInterface.OnClickListener() {

        @Override
        public void onClick(DialogInterface dialog, int which) {
            // Do nothing
            dialog.dismiss();
        }
    });

    AlertDialog alert = dialogoSalir.create();
    alert.show();
}
```

Figura 164: Código Java del primer nivel del juego de las formas

Una vez en el segundo nivel de este juego primeramente se definen las variables que identificarán los sonidos en forma de ayuda (informando que figura es la próxima a buscar o informando también si hemos acertado o fallado en forma de onomatopeya).

```
//Declaración del archivo de audio
SoundPool soundPool;
int idSonido_Acierto;
int idSonido_Error;
int idSonido_BuscarCuadrado;
int idSonido_BuscarRectangulo;
int idSonido_BuscarTriangulo;
int idSonido_BuscarCirculo;
int idSonido_BuscarRombo;
```

Figura 165: Código Java del segundo nivel del juego de las formas (I)

Además de más variables que serán necesarias como por ejemplo las vistas que se usarán en esta actividad.

```
//Declaración de las vistas
Button botonAyuda;
Button botonCerrarAyuda;
Button Forma_1;
Button Forma_2;
Button Forma_3;
Button Forma_4;
Button Forma_5;
TextView siguienteForma;
TextView textoPuntuacion;
TextView tiempoAtras;
```

Figura 166: Código Java del segundo nivel del juego de las formas (II)

Los dos primeros botones consisten en el botón superior derecha que aparece en todas las actividades para mostrar la ventana de diálogo de ayuda y el botón que aparece en dicho diálogo para cerrarla. Los botones *Forma_1* a *Forma_5* serán los botones que corresponden a las formas geométricas. Por último tendremos tres vistas de tipo *TextView*, todas mostrarán información basada en texto por pantalla. La primera mostrará que forma es la siguiente a buscar, la segunda marcará la puntuación actual y la última consiste en el contador de tiempo atrás.

Las demás variables declaradas se irán explicando a medida que vayan apareciendo.

En el método *onCreate* se establece la relación entre las vistas definidas en el archivo Java y las vistas del *layout* definidas en el archivo XML. Donde hay que destacar que las vistas *Forma_1* a *Forma_5* serán los botones donde se establecen aleatoriamente imágenes de formas geométricas.

```
//Enlazamos el archivo java con el xml
botonAyuda = (Button)findViewById(R.id.botonAyudaJuegoFormasSegundoNivel);
siguienteForma = (TextView)findViewById(R.id.formaAbuscar);
textoPuntuacion = (TextView)findViewById(R.id.puntos_conseguidos_juego_formas_segundo_nivel);
Forma_1 = (Button)findViewById(R.id.boton_Forma_1);
Forma_2 = (Button)findViewById(R.id.boton_Forma_2);
Forma_3 = (Button)findViewById(R.id.boton_Forma_3);
Forma_4 = (Button)findViewById(R.id.boton_Forma_4);
Forma_5 = (Button)findViewById(R.id.boton_Forma_5);
tiempoAtras = (TextView)findViewById(R.id.contador_juego_formas_segundo_nivel);
```

Figura 167: Código Java del segundo nivel del juego de las formas geométricas (III)

Y también se cargan los sonidos.

```
//Carga del audio
soundPool = new SoundPool( 5, AudioManager.STREAM_MUSIC , 0);
idSonido_Acierto = soundPool.load(this, R.raw.acierto, 0);
idSonido_Error = soundPool.load(this, R.raw.error, 0);
idSonido_BuscarCuadrado = soundPool.load(this, R.raw.buscarcuadrado, 0);
idSonido_BuscarRectangulo = soundPool.load(this, R.raw.buscarrectangulo, 0);
idSonido_BuscarTriangulo = soundPool.load(this, R.raw.buscartriangulo, 0);
idSonido_BuscarCirculo = soundPool.load(this, R.raw.buscarcirculo, 0);
idSonido_BuscarRombo = soundPool.load(this, R.raw.buscarrombo, 0);
```

Figura 168: Código Java del segundo nivel del juego de las formas (IV)

Habrá un método para saber qué forma es la siguiente a buscar, para ello, se establecerá un número aleatorio que nunca se repetirá entre turno y turno. Dependiendo de cuál sea ese número se buscará una figura u otra. Además accediendo a la configuración de usuario se indicará por voz que figura es la buscada, si esta opción está activada.

Jamás entre turno y turno tocará buscar la misma figura, ya que mediante la variable *numeroAleatorioBuscarForma* guardamos la última forma buscada y esta no podrá repetirse nunca.

```
//Método que nos indica la siguiente forma a buscar
public void EstablecerFormaParaBuscar(){

    //Evitamos que se repita la figura a buscar para buscar siempre una diferente a la anterior
    do{
        numeroAleatorioBuscarForma = (int) Math.floor(Math.random()*5+1); //Valor aleatorio entre 1 y 5
    } while(numeroAleatorioBuscarForma == numeroAnteriorBuscarForma);

    numeroAnteriorBuscarForma = numeroAleatorioBuscarForma;
    numeroFormaABuscar = numeroAleatorioBuscarForma;

    if (numeroAleatorioBuscarForma == 1){

        siguienteForma.setText("CUADRADO");
        //Comprobamos si la ayuda por voz está activada
        if (pref.getBoolean("ayudaPorVoz", true)==true){
            soundPool.play(idSonido_BuscarCuadrado, 1, 1, 1, 0, 1);
        }
        formaABuscar = "cuadrado";
    }
    else if (numeroAleatorioBuscarForma == 2){

        siguienteForma.setText("RECTÁNGULO");
        //Comprobamos si la ayuda por voz está activada
        if (pref.getBoolean("ayudaPorVoz", true)==true){
            soundPool.play(idSonido_BuscarRectangulo, 1, 1, 1, 0, 1);
        }
        formaABuscar = "rectangulo";
    }
```

Figura 169: Código Java del segundo nivel del juego de las formas (V)

Otro método será el que establezca la combinación de las formas en pantalla (habrá hasta 10 diferentes). La forma de implementar este método es similar al anterior que se ha descrito. Habrán diez combinaciones distintas (se usa el método *Math.random*

para ello) y dependiendo de qué número se obtenga las figuras serán ordenadas de una forma u otra, además el orden jamás se repetirá dos veces seguidas.

```
//Método para establecer las figuras en pantalla de forma aleatoria
public void EstablecerDibujoDeLasFiguras(){

    //Evitamos que se repita la colocación de las figuras del turno anterior
    do{
        numeroAleatorioEstablecerDibujos = (int) Math.floor(Math.random()*10+1); //Valor aleatorio entre 1 y 5
    } while(numeroAleatorioEstablecerDibujos == numeroAnteriorEstablecerDibujos);

    numeroAnteriorEstablecerDibujos = numeroAleatorioEstablecerDibujos;

    if (numeroAleatorioEstablecerDibujos == 1){

        Forma_1.setBackgroundResource(R.drawable.cuadrado_juego_formas);
        Forma_2.setBackgroundResource(R.drawable.rectangulo_juego_formas);
        Forma_3.setBackgroundResource(R.drawable.triangulo_juego_formas);
        Forma_4.setBackgroundResource(R.drawable.circulo_juego_formas);
        Forma_5.setBackgroundResource(R.drawable.rombo_juego_formas);

        ordenDeFormas = 1;
    }
```

Figura 170: Código Java del segundo nivel del juego de las formas (VI)

Sabiendo que forma tenemos que buscar y que posición tienen las figuras en pantalla, será fácil saber si el usuario ha acertado cuando pulse un botón. Por ejemplo si el jugador pulsa el primer botón y estamos buscando un "cuadrado" solamente las combinaciones de formas 1, 6 y 10 serán las correctas, ya que estas en esas combinaciones tendrán un cuadrado en el primer botón.

```
public void Forma_1_pulsada (View view){

    if (formaABuscar.equalsIgnoreCase("cuadrado")){

        if (ordenDeFormas == 1){
            FiguraAcertada();
            EstablecerFormaParaBuscar();
            EstablecerDibujoDeLasFiguras();
        }
        else if (ordenDeFormas == 6){
            FiguraAcertada();
            EstablecerFormaParaBuscar();
            EstablecerDibujoDeLasFiguras();
        }
        else if (ordenDeFormas == 10){
            FiguraAcertada();
            EstablecerFormaParaBuscar();
            EstablecerDibujoDeLasFiguras();
        }
        else{
            FiguraFallada();
            EstablecerFormaParaBuscar();
            EstablecerDibujoDeLasFiguras();
        }
    }
```

Figura 171: Código Java del segundo nivel del juego de las formas (VII)

Cuando el jugador acierte o falle se comprobará si se necesitan reproducir los efectos de sonido referidos a estas acciones además de activar una animación.

```
//Método que se ejecuta cuando acertamos la figura
public void FiguraAcertada(){

    puntuacion = puntuacion + incrementoPuntuacion;

    //Comprobamos si debemos activar lo efectos de sonido
    if (pref.getBoolean("efectos_sonido", true)==true){
        soundPool.play(idSonido_Acierto, 1, 1, 1, 0, 1);
    }

    textoPuntuacion.setText(Integer.toString(puntuacion));
    textoPuntuacion.startAnimation(animacionPuntos);
}

//Método que se ejecuta cuando fallamos la figura
public void FiguraFallada(){

    //Comprobamos si debemos activar lo efectos de sonido
    if (pref.getBoolean("efectos_sonido", true)==true){
        soundPool.play(idSonido_Error, 1, 1, 1, 0, 1);
    }

    textoPuntuacion.setText(Integer.toString(puntuacion));
}
```

Figura 172: Código Java del segundo nivel del juego de las formas geométricas (VIII)

Es posible que la actividad pase a un estado de pausa o esta se pare. Para no perder ciertos datos como la puntuación, la figura a buscar y el orden de las figuras en pantalla, hay dos métodos para ello, uno para guardar los datos y otro para recuperarlos. Una vez recuperados habría que volver a establecer por pantalla las imágenes de las figuras en el orden donde estaban.

```
//Guardamos las variables cuando pasemos a un estado de pausa en la aplicación
@Override
protected void onSaveInstanceState(Bundle guardarEstado) {
      super.onSaveInstanceState(guardarEstado);
      guardarEstado.putInt("puntuacion", puntuacion);
      guardarEstado.putInt("numeroFormaABuscar", numeroFormaABuscar);
      guardarEstado.putInt("ordenDeFormas", ordenDeFormas);
}

//Cargamos los datos que teníamos al volver a estar la aplicación activa
@Override
protected void onRestoreInstanceState(Bundle recEstado) {
    super.onRestoreInstanceState(recEstado);
    puntuacion = recEstado.getInt("puntuacion");
    textoPuntuacion.setText(Integer.toString(puntuacion));
```

Figura 173: Código Java del segundo nivel del juego de las formas geométricas (IX)

Ya en el tercer y último nivel tendremos métodos muy parecidos al nivel anterior, por ejemplo, esta vez no habrá que buscar una forma geométrica, sino dibujarla, por lo tanto, de forma aleatoria se indicará al jugador que forma geométrica debe dibujar exactamente. Mediante la instrucción *Math.random* aleatoriamente se asignará un número a una variable y dependiendo de qué número sea se deberá dibujar una forma

u otra, estableciendo de nuevo por pantalla el nombre de la figura a mostrar -*setText ("Nombre_de_la_figura")*- y comprobando si es necesario usar la ayuda por voz para dar facilidades al jugador.

Lo más significativo en este nivel sería reconocer los patrones de dibujo que realiza el jugador (concepto ya explicado en el capítulo *5.5.1 Código compartido por las actividades*).

Los métodos que se lanzan cuando el jugador acierta o falla son idénticos a los del nivel anterior.

Una vez el tiempo del nivel haya finalizado se abrirá un cuadro de diálogo informando de los puntos conseguidos y estos, junto con información para identificar el juego que se acaba de jugar, se enviarán de nivel inferior a nivel inferior para una vez en el menú principal, guardarlos y visualizarlos en su correspondiente apartado.

```
//Crea el diálogo de aviso para finalizar el juego y recontar los puntos totales
private void DialogoFinalizarJuego() {

    AlertDialog.Builder dialogoSalir = new AlertDialog.Builder(this);
    dialogoSalir.setMessage("¡JUEGO FINALIZADO!\n\nPuntuación: "+puntuacion);
    dialogoSalir.setPositiveButton("Salir", new DialogInterface.OnClickListener() {

        public void onClick(DialogInterface dialog, int which) {
            VolverAlMenu();
            dialog.dismiss();
        }
    });
    AlertDialog alert = dialogoSalir.create();
    alert.show();
}
```

Figura 174: Código Java del tercer nivel del juego de las formas (I)

```
//Devolvemos la información de la puntuación actividad a actividad hasta llegar al menú
private void VolverAlMenu(){
    Intent intent = new Intent();
    intent.putExtra("mensajeControl","juegoFormasFinalizado");
    intent.putExtra("puntuacion", puntuacion);
    setResult(RESULT_OK, intent);
    finish();
}
```

Figura 175: Código Java del tercer nivel del juego de las formas (II)

○ *PINTA Y COLOREA*

Como ya se ha explicado en la parte de diseño, este mini-juego se compone de diferentes actividades (una, la principal, mediante la cual vemos los botones para cambiar las propiedades del lienzo o del pincel y otras donde modificamos estas propiedades) y una vista que es sobre la que se dibujará utilizando la pantalla táctil del dispositivo.

Primeramente se explicará cómo se han desarrollado las actividades y a continuación se pasará a explicar el desarrollo de la vista personalizada.

A la actividad principal del mini-juego se le ha llamado *JuegoPintar* y se comienza definiendo las variables que se van a utilizar, ya sean los diferentes botones para seleccionar los colores del lienzo o el pincel, las variables donde guardamos estos datos o la vista personalizada (*vistaLienzo*), así mismo también se define una variable llamada *acción* la cual es usada para saber si el usuario ha pulsado o está deslizando el dedo por la pantalla, dicha información será utilizada para saber cuándo y cómo pintar.

```
//Declaración de las vistas
Button botonAyuda;
Button botonCerrarAyuda;
Button botonCambiarColorLienzo;
Button botonCambiarColorPincel;
Button botonNuevaPagina;
Button botonNivelGrosorPincel;
RelativeLayout layout;
TextView pintaAqui;
TextView coordenadaPosicionX;
TextView coordenadaPosicionY;
VistaLienzo vistaLienzo;
String accion;
```

```
//Declaración de las variables de las selecciones del usuario
String seleccionFondoLienzo;
String seleccionColorPincel;
int nivelGrosorPincel;

//Declaración del cuadro de dialogo de ayuda
Dialog dialogoAyuda;

float posicionX;
float posicionY;
```

Figura 176: Código Java de la actividad "JuegoPintar" (I)

En el método *onCreate* se enlazarán las vistas previamente definidas con sus respectivas vistas del archivo XML, incluidos el contenedor *Layout* (para así poder cambiar el color de fondo de la actividad de forma dinámica) y la vista personalizada. A esta se le asignará un escuchador de eventos que detectará cuando ha sido pulsada la pantalla táctil. Para ello, previamente se deberá haber implementado su interfaz correspondiente a esta clase.

```
public class JuegoPintar extends Activity implements OnTouchListener{
```

Figura 177: Código Java de la actividad "JuegoPintar" (II)

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

    //Enlazamos el archivo java con el xml
    botonAyuda = (Button)findViewById(R.id.botonAyudaJuegoPintar);
    botonCambiarColorLienzo = (Button)findViewById(R.id.botonCambiarFondoJuegoPintar);
    botonCambiarColorPincel = (Button)findViewById(R.id.botonCambiarColorPincelJuegoPintar);
    botonNuevaPagina = (Button)findViewById(R.id.botonNuevaPaginaJuegoPintar);
    botonNivelGrosorPincel = (Button)findViewById(R.id.botonNivelGrosorJuegoPintar);
    layout = (RelativeLayout)findViewById(R.id.Layout_Juego_Pintar);
    pintaAqui = (TextView)findViewById(R.id.pintar_aqui);
    coordenadaPosicionX = (TextView)findViewById(R.id.posicionX);
    coordenadaPosicionY = (TextView)findViewById(R.id.posicionY);
    vistaLienzo = (VistaLienzo)findViewById(R.id.vistaLienzo);

    //Establecemos transparencia al botón
    botonCambiarColorLienzo.getBackground().setAlpha(225);
    botonCambiarColorPincel.getBackground().setAlpha(225);
    botonNuevaPagina.getBackground().setAlpha(225);
    botonNivelGrosorPincel.getBackground().setAlpha(225);

    //Valores por defecto o iniciales del pincel
    nivelGrosorPincel = 8;
    seleccionColorPincel = "#000000";

    vistaLienzo.setOnTouchListener(this);
}
```

Figura 178: Código Java de la actividad "JuegoPintar" (III)

Como ya se ha comentado, el usuario podrá cambiar el color del lienzo y el color y grosor del pincel. Una vez se pulse uno de los botones que realizan estas acciones, se abrirá una ventana una nueva actividad en forma de ventana de diálogo, y esta esperará un dato, ya sea una cadena de caracteres para identificar un color o un número entero para un grosor. La actividad se lanzará esperando un resultado, además para poder identificar qué resultado es devuelto, cada botón tendrá su código identificativo diferente.

```
//Método para lanzar la actividad de selección del color del lienzo
public void lanzarCambiarColorLienzo(View view){
    Intent i = new Intent(this,SeleccionarLienzo.class);
    startActivityForResult(i, 1234);
}

//Método para lanzar la actividad de selección del color del pincel
public void lanzarCambiarColorPincel(View view){
    Intent i = new Intent(this,SeleccionarPincel.class);
    startActivityForResult(i, 2468);
}

//Método para lanzar la actividad de selección del grosor del pincel
public void lanzarNivelGrosor(View view){
    Intent i = new Intent(this,SeleccionarGrosor.class);
    startActivityForResult(i, 1357);
}
```

Figura 179: Código Java de la actividad "JuegoPintar" (IV)

El botón que borra lo que haya en pantalla cambiará el estado de la variable acción, hecho esto, se enviarán los datos a la vista, una vez allí, la vista detectará que se debe borrar el dibujo. Esto se verá con más detenimiento cuando se explique la implementación de la vista.

```
//Método que borra lo que hay en pantalla
public void lanzarNuevaPagina(View view){
    accion = "borrar";
    vistaLienzo.establecerDatos(posicionX, posicionY, nivelGrosorPincel, seleccionColorPincel, accion);
}
```

Figura 180: Código Java de la actividad "JuegoPintar" (V)

Una vez el jugador ha seleccionado el color del lienzo, el color del pincel o el grosor, se enviarán unos datos que el método *onActivityResult* recogerá. Habrá un código que identificará que botón, y en consecuencia, que parámetros se quieren modificar. Se guardarán los datos recogidos en su variable correspondiente para, o establecer un color de fondo de pantalla (en el caso que el usuario haya pulsado ese botón para modificarlo) o guardar el color/grosor del lienzo y posteriormente enviarlo a la vista para modificar sus propiedades a la hora de dibujar.

```
@Override
//Aquí llega la información de los colores del lienzo y el color y grosor del pincel seleccionada
protected void onActivityResult (int requestCode, int resultCode, Intent data){

    //Si nos llega información del color del lienzo (cod: 1234)
    if (requestCode==1234 && resultCode==RESULT_OK) {
        seleccionFondoLienzo = data.getExtras().getString("color");

        //Se establece el color de fondo o del lienzo
        layout.setBackgroundColor(Color.parseColor(seleccionFondoLienzo));

        //Si el color de fondo es negro, cambiamos el color de las coordenadas para que se vean bien
        if (seleccionFondoLienzo.equalsIgnoreCase("#000000")){
            coordenadaPosicionX.setTextColor(Color.WHITE);
            coordenadaPosicionY.setTextColor(Color.WHITE);
        }

        else{
            coordenadaPosicionX.setTextColor(Color.BLACK);
            coordenadaPosicionY.setTextColor(Color.BLACK);
        }
    }

    //Si nos llega información del color del pincel (cod: 2468)
    if (requestCode==2468 && resultCode==RESULT_OK) {
        seleccionColorPincel = data.getExtras().getString("color_pincel");
    }

    //Si nos llega información del grosor del pincel (cod: 1357)
    if (requestCode==1357 && resultCode==RESULT_OK) {
        nivelGrosorPincel = data.getExtras().getInt("nivel_grosor");
    }
}
```

Figura 181: Código Java de la actividad "JuegoPintar" (VI)

El método *onTouch* será lanzado de forma automática cuando se toque la pantalla. Se guardarán las posiciones X e Y de la posición del dedo y se diferenciarán 3 estados: El primer estado, "down", será cuando el usuario toque (sin deslizar) la pantalla, aquí se borrará el mensaje en pantalla que informa al usuario de donde dibujar. El segundo estado, "move", será cuando el usuario desplace el dedo por la pantalla, aquí se imprimirá por pantalla (por la parte inferior derecha) las coordenadas X e Y a modo informativo. El tercer estado, "up", será cuando se levante el dedo de la pantalla y deshabilitará las visualización de las coordenadas por pantalla.

Mientras el usuario esté tocando la pantalla, se irán enviando datos a la vista, ya sea de las coordenadas, de las propiedades del pincel o del estado de la acción. El uso de estos datos se verá más adelante.

```
@Override
public boolean onTouch(View vista, MotionEvent evento) {
    // TODO Auto-generated method stub

    posicionX = evento.getX();
    posicionY = evento.getY();

    //Si pulsamos la pantalla
    if (evento.getAction() == MotionEvent.ACTION_DOWN){
        accion = "down";
        pintaAqui.setHint(null); //Cuando pulsemos eliminamos el mensaje de ayuda al usuario
    }

    //Cuando movamos el dedo por la pantalla, pintamos
    if (evento.getAction() == MotionEvent.ACTION_MOVE){
        accion = "move";
        coordenadaPosicionX.setText("X: "+Float.toString(posicionX));
        coordenadaPosicionY.setText("Y: "+Float.toString(posicionY));
        coordenadaPosicionX.setVisibility(View.VISIBLE);
        coordenadaPosicionY.setVisibility(View.VISIBLE);
    }

    if (evento.getAction() == MotionEvent.ACTION_UP){
        accion = "up";
        coordenadaPosicionX.setVisibility(View.INVISIBLE);
        coordenadaPosicionY.setVisibility(View.INVISIBLE);
    }
    vistaLienzo.establecerDatos(posicionX, posicionY, nivelGrosorPincel, seleccionColorPincel, accion);
    return true;
}
```

Figura 182: Código Java de la actividad "JuegoPintar" (VII)

Como ya se ha dicho varias veces, cuando el usuario desee cambiar un atributo ya sea del lienzo o del pincel, pulsará el botón correspondiente y lo seleccionará. Para los 3 botones la forma de realizar esto es similar, por lo tanto se explicará a modo de ejemplo el código de la actividad que devuelve la cadena de caracteres usada para cambiar el color del lienzo.

Una vez aparezca la actividad donde se puede cambiar el color del lienzo, se mostrarán en pantalla múltiples botones, cada uno de ellos tiene el atributo *onClick* configurado para que cuando sea pulsado, lance un método. Por ejemplo si se quiere cambiar el

color del lienzo a amarillo, será lanzado el siguiente método, el cual devolverá el código hexadecimal en forma de cadena de caracteres y este será recogido en la actividad principal de este mini-juego tal y como ha sido explicado, esto funciona para los colores, si quisiéramos modificar el grosor del pincel en lugar de devolver una cadena de caracteres, se devolvería un valor entero.

```
//Método para devolver el color amarillo a la actividad anterior
public void lanzarLienzoAmarillo(View view){
    Intent intent = new Intent();
    intent.putExtra("color","#FFFF00");
    setResult(RESULT_OK, intent);
    finish();
}
```

Figura 183: Código Java de la actividad "SeleccionarLienzo"

Una vez explicados la actividad principal y las ventanas de diálogo que cambian las posibles propiedades del dibujo, se explicará cómo se dibuja en pantalla, es decir, la vista que hace que se pueda dibujar.

Primeramente se necesitará definir e inicializar dos vectores en los cuales se guardarán los trazos que se han dibujado y con qué pinceles han sido dibujados, además de demás variables como la acción que se está realizando, la posición del dedo en pantalla, el color y grosor del pincel etc. Y se crearán dos objetos, uno de tipo *Path*, y otro de tipo *Paint*, siendo el primero de ellos la línea dibujada y el segundo el pincel con el que se dibuja esa línea.

```
public Vector <Path> Trazos;
public Vector <Paint> Pinceles;
Path trazo = new Path();
Paint pincel = new Paint();
public float posicionX;
public float posicionY;
public int grosorPincel;
public String colorPincel;
String accion;

public VistaLienzo(Context context, AttributeSet attrs) {
    super(context, attrs);

    Trazos = new Vector <Path>();
    Pinceles = new Vector <Paint>();
}
```

Figura 184: Código Java de la vista "VistaLienzo" (I)

Las vistas necesitan dos métodos, una controla el tamaño.

```
@Override
protected void onSizeChanged(int ancho, int alto, int ancho_anterior, int alto_anterior){}
```

Figura 185: Código Java de la vista "VistaLienzo" (II)

La otra lo que se muestra, en ese caso, el dibujo realizado al tocar la pantalla táctil.

Como ya se ha comentado, en todo momento que el usuario esté tocando la pantalla, se enviarán datos a la vista y esta se irá actualizando dependiendo del estado o la acción en la que se encuentre.

Cuando se ha pulsado, acción "down" se creará un nuevo trazo y pincel, se accederán a las propiedades del pincel seleccionadas previamente por el usuario y se establecerá el comienzo del dibujo en la posición X e Y donde se ha pulsado (*moveTo*), por último se añade el trazo y pincel creados a sus respectivos vectores.

```
@Override
protected void onDraw(Canvas canvas) {

    //Si la accion que hemos realizado es presionar la pantalla
    if (accion == "down"){
        trazo = new Path();
        pincel = new Paint();
        pincel.setStrokeWidth(grosorPincel);
        pincel.setStyle(Style.STROKE);
        pincel.setColor(Color.parseColor(colorPincel));
        trazo.moveTo(posicionX, posicionY);
        trazo.lineTo(posicionX, posicionY);
        Trazos.add(trazo);
        Pinceles.add(pincel);
    }

    //Si nos movemos por la pantalla trazamos lineas
    else if (accion == "move"){
        trazo.lineTo(posicionX, posicionY);
    }

    //Se borrarán los vectores donde están guardados los trazos y pinceles
    else if (accion == "borrar"){

        Trazos.clear();
        Pinceles.clear();
    }

    //Para dibujar se recorre el vector de los trazos y sus respectivos pinceles
    for (int i=0; i<Trazos.size(); i++)
    {
        canvas.drawPath(Trazos.get(i), Pinceles.get(i));
    }
    invalidate(); //Redibujamos constantemente
}
```

Figura 186: Código Java de la vista "VistaLienzo" (III)

Si se desplaza el dedo por pantalla, acción “move” se dibujará una línea por donde se está desplazando dicho dedo.

Si por el contrario el estado de la acción es “borrar”, se borrarán los vectores donde está guardado el dibujo realizado.

El dibujo se refrescará de forma constante (*invalidate*) al mismo tiempo que recorre los vectores de las trazas y pinceles realizados para formar el dibujo.

Por último, el siguiente método simplemente recoge los datos que se han enviado desde la actividad principal de este mini-juego a la vista, tanto de la posición como del pincel, para poder utilizarlos.

```
//Método que recibe parámetros del pincel para usarlos en esta clase y dibujar
public void establecerDatos (float x, float y, int grosor, String color, String tipoMovimiento){
    posicionX = x;
    posicionY = y;
    grosorPincel = grosor;
    colorPincel = color;
    accion = tipoMovimiento;
}
```

Figura 187: Código Java de la vista "VistaLienzo" (IV)

- ***CAPTURA LAS LETRAS***

Durante los meses que estuve aprendiendo como desarrollar aplicaciones para Android, seguí un tutorial donde paso a paso se explicaba, aparte de diferentes aspectos sobre Android, el desarrollo de un juego, el clásico juego "Asteroids" de máquinas recreativas. En principio no iba a incluirlo en este proyecto, pero pensé que podía ser buena idea modificar el código para adaptarlo a un posible mini-juego que estuviera en sintonía con la aplicación que aquí explico. Por lo tanto se ha implementado un mini-juego en el cual mediante una nave, en lugar de disparar a asteroides y que estos se rompan en diferentes partes, se disparará a letras que estarán flotando en pantalla y estás, una vez sean impactadas, emitirán un sonido (para que también por medios auditivos se trabajen las letras) y desaparecerán.

Para ello se han creado dos clases, la clase *Grafico* y la clase *VistaJuego*. La primera de ellas se crea a raíz de la cantidad de elementos gráficos que se mostrarán por pantalla: una nave, misiles (que en realidad son lapices), imágenes de letras…Dado que el comportamiento de estos elementos es similar y con el fin de reutilizar código se ha creado esta clase.

La clase *VistaJuego* (que en realidad es una vista personalizada) es la responsable de la ejecución del juego en sí, aunque primeramente se explicará cómo se ha implementado la actividad donde se encuentra el *layout* de esta vista.

Por lo tanto en esta actividad se comienza definiendo las variables que se van a utilizar, como el audio, los sensores y el acceso a las preferencias de la aplicación. Además de la vista que controla el juego en si la cual se ha llamado *vistaJuego*.

En el método *onCreate* se establece el *layout* de la actividad, se enlaza la vista personalizada (*vistaJuego*) que acabamos de definir con su respectiva vista en el *layout* y se carga el audio desde los recursos. Finalmente como será más sencillo tener una respuesta de la actividad desde la vista vistaJuego, habrá que establecer esta como actividad padre, ya que en un principio no es una actividad, sino una vista. Esta parte se verá más clara más adelante cuando se explique la vista en sí.

```
public class JuegoNave extends Activity{

    VistaJuego vistaJuego;
    MediaPlayer audio_fondo_juego_nave;
    List<Sensor> listSensors;
    SensorManager mSensorManager;
    SharedPreferences pref;

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

        //Audio de fondo
        audio_fondo_juego_nave = MediaPlayer.create(this, R.raw.audio_juego_nave);
        audio_fondo_juego_nave.setLooping(true);
        audio_fondo_juego_nave.setVolume(0.5f, 0.5f);

        vistaJuego = (VistaJuego)findViewById(R.id.VistaJuego);
        vistaJuego.setVistaVictoria(findViewById(R.id.Victoria));

        //Para poder manejar ciertas instrucciones dentro de la vista que hemos creado (VistaJuego)
        //necesitamos esta actividad como padre
        vistaJuego.setPadre(this);
    }
```

Figura 188: Código Java de la actividad "JuegoNave" (I)

También será importante tener un control del ciclo de vida de la actividad, en el momento en el que pase a un estado de pausa se deberá detener el manejo de sensores, sino estos consumirán memoria del dispositivo bajando su rendimiento.

Una vez la actividad vuelve a estar activa se deberá reproducir el sonido de fondo (si la aplicación está configurada para ello) y volver a activar los sensores para ser usados.

Cuando la actividad sea destruida, los sensores también lo serán al igual que la música de fondo, que será detenida.

```
@Override
protected void onPause() {
    super.onPause();
    audio_fondo_juego_nave.pause();
    mSensorManager.unregisterListener(vistaJuego); //Paramos los sensores
    vistaJuego.getThread().pausar();
}
@Override
protected void onResume() {
    super.onResume();

    vistaJuego.getThread().reanudar();

    //Utilización de sensor para mover la nave
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    listSensors = mSensorManager.getSensorList(Sensor.TYPE_ORIENTATION);
    if (!listSensors.isEmpty()) {
        Sensor orientationSensor = listSensors.get(0);
        mSensorManager.registerListener(vistaJuego, orientationSensor,SensorManager.SENSOR_DELAY_GAME);
    }

    //Cuando la aplicación está a punto de estar activa, comprobamos si está habilitada o
    //deshabilitada la música de fondo
    pref = PreferenceManager.getDefaultSharedPreferences(this);
    if (pref.getBoolean("musica",true)==true){
        audio_fondo_juego_nave.start();
    }
    else{
        audio_fondo_juego_nave.pause();
    }
}
@Override
protected void onDestroy() {
    vistaJuego.getThread().detener();
    audio_fondo_juego_nave.stop();
    super.onDestroy();
}
```

Figura 189: Código Java de la actividad "JuegoNave" (II)

Para la clase *Grafico* se definirán las variables que se van a utilizar, como el ancho y alto del objeto o su superficie o radio de colisión.

```
public class Grafico {

    private Drawable drawable;    //Imagen que dibujaremos
    private double posX, posY;    //Posición
    private double incX, incY;    //Velocidad de translación
    private int angulo, rotacion;//Ángulo y velocidad rotación
    private int ancho, alto;      //Dimensiones de la imagen
    private int radioColision;    //Para determinar colisión

     //Donde dibujamos el gráfico (usada en view.ivalidate)
    private View view;

     // Para determinar el espacio a borrar (view.ivalidate)
    public static final int MAX_VELOCIDAD = 20;

    //Constructor
    public Grafico(View view, Drawable drawable){
        this.view = view;
        this.drawable = drawable;
        ancho = drawable.getIntrinsicWidth();
        alto = drawable.getIntrinsicHeight();
        radioColision = (alto+ancho)/4;
    }
```

Figura 190: Código Java de la clase "Grafico" (I)

El método *dibujaGrafico* es el encargado de dibujar el objeto *drawable* en pantalla (en un *canvas*). El método *setBounds* indica los límites donde se dibujará el objeto y se dibujará con el método *draw*. Con el método *invalidate* se redibujan los objetos de forma constante para así apreciar el movimiento de los mismos.

```
//Método que dibuja un objeto de tipo drawable
public void dibujaGrafico(Canvas canvas){
    canvas.save();
    int x=(int) (posX+ancho/2);
    int y=(int) (posY+alto/2);
    canvas.rotate((float) angulo,(float) x,(float) y);
    drawable.setBounds((int)posX, (int)posY, (int)posX+ancho, (int)posY+alto);
    drawable.draw(canvas);
    canvas.restore();
    int rInval = (int) Math.hypot(ancho,alto)/2 + MAX_VELOCIDAD;
    view.invalidate(x-rInval, y-rInval, x+rInval, y+rInval);
}
```

Figura 191: Código Java de la clase "Grafico" (II)

El método *incrementaPos* es utilizado para modificar la posición y el ángulo del gráfico según la velocidad de translación y de rotación. Se le pasará el parámetro *factor* que cuanto más alto sea, a mayor velocidad se moverá el objeto. Además si un objeto llega al final de la pantalla aparecerá por el lado contrario.

```
//Método para mover el objeto por pantalla
public void incrementaPos(double factor){
    posX+=incX * factor;

    // Si salimos de la pantalla, corregimos posición
    if(posX<-ancho/2) {posX=view.getWidth()-ancho/2;}
    if(posX>view.getWidth()-ancho/2) {posX=-ancho/2;}
    posY+=incY * factor;
    if(posY<-alto/2) {posY=view.getHeight()-alto/2;}
    if(posY>view.getHeight()-alto/2) {posY=-alto/2;}
    angulo += rotacion * factor; //Actualizamos ángulo
}
```

Figura 192: Código Java de la clase "Grafico" (III)

A continuación tenemos dos métodos más. El primero mide la distancia a otro gráfico y el segundo comprueba si esa distancia es lo suficientemente pequeña para considerarlo una colisión entre dos gráficos.

```
public double distancia(Grafico g){
    return Math.hypot(posX-g.posX, posY-g.posY);
}

public boolean verificaColision(Grafico g){
    return(distancia(g) < (radioColision+g.radioColision));
}
```

Figura 193: Código Java de la clase "Grafico" (IV)

El resto de métodos serán los *setters* y *getters* necesarios.

Para finalizar la explicación de este mini-juego resta por comentar la vista que hace que todo funcione.

En el constructor se crean los diferentes objetos *drawable* (nave, misil y letras) y se inicializan con una posición, un ángulo y una velocidad determinadas.

Aunque antes será necesario declarar las variables que se van a utilizar. Se necesitarán varios objetos de tipo *Grafico* como la nave o el misil y un vector del mismo tipo donde se almacenarán las letras que estarán "flotando" por el escenario. Las variables en si, se irán comentando a medida que sea necesario, tendremos variables para el control de la nave o el misil, para el hilo secundario de ejecución o para acceder a las preferencias y los efectos de sonido.

```
public class VistaJuego extends View implements SensorEventListener {

    // //// LETRAS //////
    private Vector<Grafico> Letras; // Vector con las letras
    private Vector<String> sonidosDeLetras; // Vector con el nombre de cada sonido de las letras

    // //// NAVE //////
    private Grafico nave; // Gráfico de la nave
    private int giroNave; // Incremento de dirección
    private float aceleracionNave; // aumento de velocidad
    private int sensibilidadGiroNave = 4; // a menor valor, mayor sensibilidad de giro

    // //// MISIL //////
    private Grafico misil;
    private static int PASO_VELOCIDAD_MISIL = 12;
    private boolean misilActivo = false;
    private int tiempoMisil;

    // //// THREAD Y TIEMPO //////
    // Thread encargado de procesar el juego
    private ThreadJuego thread = new ThreadJuego();
    // Cada cuanto queremos procesar cambios (ms)
    private static int PERIODO_PROCESO = 50;
    // Cuando se realizó el último proceso
    private long ultimoProceso = 0;
```

Figura 194: Código Java de la vista "VistaJuego" (I)

```
// MOVIMIENTO DE LA NAVE Y DISPARO //
private float mX=0, mY=0;
private boolean disparo=false;

// SENSORES //
private boolean hayValorInicial = false;
private float valorInicial;

// //// MULTIMEDIA //////
SoundPool soundPool;
int idDisparo, idExplosion;
int idLetraA, idLetraE, idLetraI, idLetraO, idLetraU;

// PREFERENCIAS //
SharedPreferences pref;

// PUNTUACIONES //
private int puntuacion=0;

///// ESTADOS DE JUEGO /////
public static final int ESTADO_JUGANDO = 0;
public static final int ESTADO_VICTORIA = 1;

private int estado;
private View vistaVictoria;

//Para realizar la respuesta de la actividad va a ser más sencillo hacerlo desde
//VistaJuego en lugar de desde Juego. El problema es que esta clase es un vista no una
//actividad. Para solucionar el problema puedes usar el siguiente truco.
private Activity padre;
public void setPadre(Activity padre) {
    this.padre = padre;
}
```

Figura 195: Código Java de la vista "VistaJuego" (II)

Al constructor se le pasarán 2 variables. La primera de tipo *Context* permitirá acceder al contexto de la aplicación por ejemplo para la utilización de recursos. La segunda de tipo *AttributeSet* permitirá acceder a los atributos de la vista cuando sea creada desde XML.

Así que en el constructor se definirán objetos de tipo *Drawable* que representarán las letras a las que el jugador debe disparar, la nave y el misil. A dichas letras se le asignarán sus respectivas imágenes ubicadas en los recursos de la aplicación al igual que a la nave y al misil.

```
public VistaJuego(Context context, AttributeSet attrs) {
    super(context, attrs);

    //Definimos las imágenes que se mostrarán en pantalla
    Drawable drawableLetraA, drawableLetraB, drawableLetraC, drawableLetraD;
    Drawable drawableLetraE, drawableLetraF, drawableLetraG, drawableLetraH;
    Drawable drawableLetraI, drawableLetraJ, drawableLetraK, drawableLetraL;
    Drawable drawableLetraM, drawableLetraN, drawableLetraO, drawableLetraP;
    Drawable drawableLetraQ, drawableLetraR, drawableLetraS, drawableLetraT;
    Drawable drawableLetraU, drawableLetraV, drawableLetraW, drawableLetraX;
    Drawable drawableLetraY, drawableLetraZ;
    Drawable drawableNave;
    Drawable drawableMisil;
```

Figura 196: Código Java de la vista "VistaJuego" (III)

```
//Asignamos las imágenes de las letras que tenemos en los recursos
drawableLetraA = context.getResources().getDrawable(R.drawable.letra_a_juego_nave);
drawableLetraB = context.getResources().getDrawable(R.drawable.letra_b_juego_nave);
drawableLetraC = context.getResources().getDrawable(R.drawable.letra_c_juego_nave);
drawableLetraD = context.getResources().getDrawable(R.drawable.letra_d_juego_nave);
drawableLetraE = context.getResources().getDrawable(R.drawable.letra_e_juego_nave);
drawableLetraF = context.getResources().getDrawable(R.drawable.letra_f_juego_nave);
drawableLetraG = context.getResources().getDrawable(R.drawable.letra_g_juego_nave);
drawableLetraH = context.getResources().getDrawable(R.drawable.letra_h_juego_nave);
drawableLetraI = context.getResources().getDrawable(R.drawable.letra_i_juego_nave);
drawableLetraJ = context.getResources().getDrawable(R.drawable.letra_j_juego_nave);
drawableLetraK = context.getResources().getDrawable(R.drawable.letra_k_juego_nave);
drawableLetraL = context.getResources().getDrawable(R.drawable.letra_l_juego_nave);
drawableLetraM = context.getResources().getDrawable(R.drawable.letra_m_juego_nave);
drawableLetraN = context.getResources().getDrawable(R.drawable.letra_n_juego_nave);
drawableLetraO = context.getResources().getDrawable(R.drawable.letra_o_juego_nave);
drawableLetraP = context.getResources().getDrawable(R.drawable.letra_p_juego_nave);
drawableLetraQ = context.getResources().getDrawable(R.drawable.letra_q_juego_nave);
drawableLetraR = context.getResources().getDrawable(R.drawable.letra_r_juego_nave);
drawableLetraS = context.getResources().getDrawable(R.drawable.letra_s_juego_nave);
drawableLetraT = context.getResources().getDrawable(R.drawable.letra_t_juego_nave);
drawableLetraU = context.getResources().getDrawable(R.drawable.letra_u_juego_nave);
drawableLetraV = context.getResources().getDrawable(R.drawable.letra_v_juego_nave);
drawableLetraW = context.getResources().getDrawable(R.drawable.letra_w_juego_nave);
drawableLetraX = context.getResources().getDrawable(R.drawable.letra_x_juego_nave);
drawableLetraY = context.getResources().getDrawable(R.drawable.letra_y_juego_nave);
drawableLetraZ = context.getResources().getDrawable(R.drawable.letra_z_juego_nave);

//Asignamos la imágen de la nave que tenemos en los recursos
drawableNave = context.getResources().getDrawable(R.drawable.nave);

//Asignamos la imagen del misil que tenemos en los recursos
drawableMisil = context.getResources().getDrawable(R.drawable.lapiz_misil);
```

Figura 197: Código Java de la vista "VistaJuego" (IV)

En el constructor también se crearán los dos vectores, uno de tipo *Grafico* para almacenar letras, el otro de cadena de caracteres para poder identificar y posteriormente asignar sonidos a dichas letras.

```
//Hacemos un vector con las letras
Letras = new Vector<Grafico>();

//Hacemos un vector con el nombre de cada sonido de las letras
sonidosDeLetras = new Vector<String>();
```

Figura 198: Código Java de la vista "VistaJuego" (V)

Dependiendo del nivel de dificultad escogido por el usuario en el apartado de configuración, aparecerán más o menos letras por pantalla. Si por ejemplo el usuario quiere jugar en nivel fácil, solamente le serán mostradas las letras vocales, si jugara en nivel de dificultar medio, aparcería la primera mitad del abecedario y si por el contrario jugar en difícil, "flotarían" por el escenario todas las letras del abecedario. Por lo tanto, se accede a la configuración para realizar esto y rellenar el vector con cantidad de letras establecida.

Para rellenar el vector se instanciará un objeto *Grafico* por cada letra, asignándole una imagen, velocidad, ángulo y rotación aleatorios.

```
//Obtenemos de las preferencias la dificultad elegida por el usuario
SharedPreferences pref = context.getSharedPreferences("com.example.proyectoeduca_preferences", Context.MODE_PRIVATE);
String dificultad = pref.getString("dificultad", "1");
int nivelDificultad = Integer.parseInt(dificultad);
//FÁCIL = 1
//NORMAL = 2
//DIFÍCIL = 3
//Si jugamos en fácil sólamente apareceran las vocales
if (nivelDificultad == 1){
    //Rellenamos el vector con letras
    Grafico letraA = new Grafico(this, drawableLetraA);
    letraA.setIncY(Math.random() * 4 - 2);
    letraA.setIncX(Math.random() * 4 - 2);
    letraA.setAngulo((int) (Math.random() * 360));
    letraA.setRotacion((int) (Math.random() * 8 - 4));
    Letras.add(letraA);
```

Figura 199: Código Java de la vista "VistaJuego" (VI)

A su vez también se rellena el vector de caracteres con un nombre identificativo para cada letra

```
//Rellenamos el vector del nombre de las letras
String audioLetraA = new String ("audioLetraA");
sonidosDeLetras.add(audioLetraA);

String audioLetraE = new String ("audioLetraE");
sonidosDeLetras.add(audioLetraE);

String audioLetraI = new String ("audioLetraI");
sonidosDeLetras.add(audioLetraI);

String audioLetraO = new String ("audioLetraO");
sonidosDeLetras.add(audioLetraO);

String audioLetraU = new String ("audioLetraU");
sonidosDeLetras.add(audioLetraU);
```

Figura 200: Código Java de la vista "VistaJuego" (VII)

La nave y el misil no hará falta iniciar su velocidad, ángulo o rotación, estos parámetros se establecerán cuando el jugador esté interactuando con la pantalla.

Para los efectos de sonido se usará la clase *SoundPool*, la cual ya ha aparecido varias veces en este proyecto.

```
//Creamos la nave
nave = new Grafico(this, drawableNave);

//Creamos el misil
misil = new Grafico(this, drawableMisil);

//Sonido VFX
soundPool = new SoundPool( 5, AudioManager.STREAM_MUSIC , 0);
idDisparo = soundPool.load(context, R.raw.disparo, 0);
idExplosion = soundPool.load(context, R.raw.explosion, 0);
idLetraA = soundPool.load(context, R.raw.letra_a, 0);
idLetraE = soundPool.load(context, R.raw.letra_e, 0);
idLetraI = soundPool.load(context, R.raw.letra_i, 0);
idLetraO = soundPool.load(context, R.raw.letra_o, 0);
idLetraU = soundPool.load(context, R.raw.letra_u, 0);
```

Figura 201: Código Java de la vista "VistaJuego" (VIII)

Android realiza un proceso de varias pasadas para determinar el ancho y el alto de cada vista dentro de un *layout* . Cuando finalmente ha establecido las dimensiones de una vista, llamará a su método *onSizeChanged* que nos indica como parámetros el ancho y alto asignados. Una vez se sepan estos datos, se podrán situar las letras en pantalla en posiciones aleatorias y la nave en el centro de la misma.

No se debe olvidar iniciar el hilo de ejecución secundario. Este es usado para que no se colapse el hilo principal y así evitar cualquier tipo de problema que pueda surgir.

```
@Override
protected void onSizeChanged(int ancho, int alto, int ancho_anter, int alto_anter) {
    super.onSizeChanged(ancho, alto, ancho_anter, alto_anter);

    // Una vez que conocemos nuestro ancho y alto.
    for (Grafico letra: Letras) {
        // Evitamos que al principio aparzca una letra cerca de la nave
        do {
            letra.setPosX(Math.random()*(ancho-letra.getAncho()));
            letra.setPosY(Math.random()*(alto-letra.getAlto()));
        } while(letra.distancia(nave) < (ancho+alto)/5);
    }

    // Situamos la nave para que aparzca en medio
    nave.setPosX((ancho-nave.getAncho())/2);
    nave.setPosY((alto-nave.getAlto())/2);

    //Llamamos a la clase ThreadJuego (el objeto creado a través de ella es es thread
    //la cual es un bucle infinito a actualizarFisica()
    ultimoProceso = System.currentTimeMillis();
    thread.start();
}
```

Figura 202: Código Java de la vista "VistaJuego" (IX)

En el método onDraw se dibujarán los objetos de tipo *Grafico*. Si se ha accionado el misil, también este será dibujado.

Una vez la partida pase a un estado de victoria se verá reflejado en pantalla haciendo visible un mensaje.

```
//Método que dibuja las vistas
@Override
synchronized protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    for (Grafico letra: Letras) {
        letra.dibujaGrafico(canvas);
    }

    nave.dibujaGrafico(canvas);

    if (misilActivo){
        misil.dibujaGrafico(canvas);
    }

    if (estado == ESTADO_VICTORIA){
        vistaVictoria.setVisibility(VISIBLE);
    }
}
```

Figura 203: Código Java de la vista "VistaJuego" (X)

El método *actualizaFisica* se llama de forma constante y actualiza los movimientos de los objetos en pantalla cada *PERIODO_PROCESO* milisegundos.

```
synchronized protected void actualizaFisica() {
    long ahora = System.currentTimeMillis();
    // No hagas nada si el período de proceso no se ha cumplido.
    if (ultimoProceso + PERIODO_PROCESO > ahora) {
          return;
    }
    // Para una ejecución en tiempo real calculamos retardo
    double retardo = (ahora - ultimoProceso) / PERIODO_PROCESO;
    ultimoProceso = ahora; // Para la próxima vez
    // Actualizamos velocidad y dirección de la nave a partir de
    // giroNave y aceleracionNave (según la entrada del jugador)
    nave.setAngulo((int) (nave.getAngulo() + giroNave * retardo));
    double nIncX = nave.getIncX() + aceleracionNave * Math.cos(Math.toRadians(nave.getAngulo())) * retardo;
    double nIncY = nave.getIncY() + aceleracionNave * Math.sin(Math.toRadians(nave.getAngulo())) * retardo;
    // Actualizamos si el módulo de la velocidad no excede el máximo
    if (Math.hypot(nIncX,nIncY) <= Grafico.getMaxVelocidad()){
          nave.setIncX(nIncX);
          nave.setIncY(nIncY);
    }
    // Actualizamos posiciones X e Y
    nave.incrementaPos(retardo);
    for (Grafico letra : Letras) {
        letra.incrementaPos(retardo);
    }
    // Actualizamos posición de misil
    if (misilActivo) {
        misil.incrementaPos(retardo);
        tiempoMisil-=retardo;
        if (tiempoMisil < 0) {
            misilActivo = false;
        }
        else {
            for (int i = 0; i < Letras.size(); i++)
                if (misil.verificaColision(Letras.elementAt(i))) {
                    destruyeLetra(i);
                    break;
                }
        }
```

Figura 204: Código Java de la vista "VistaJuego" (XI)

```
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) {}
                   }
               }
           }
       }
}
```

Esta clase será la que llame continuamente al método para actualizar los movimientos de los objetos, se trata de un hilo secundario de la aplicación.

Cuando el juego este parado, no se actualizará el movimiento de los objetos y se detendrá el uso de sensores.

Figura 205: Código Java de la vista "VistaJuego" (XII)

El método *onTouchEvent* captura cuando el usuario toca la pantalla táctil. Cuando la acción sea golpear la pantalla se accionará el disparo de misil, si en cambio el jugador arrastra el dedo, la nave se moverá hasta que este deje de tocar la pantalla.

```
//Método para el control de la nave mediante la pantalla táctil
@Override
public boolean onTouchEvent (MotionEvent event) {
   super.onTouchEvent(event);
   float x = event.getX();
   float y = event.getY();
   switch (event.getAction()) {
       case MotionEvent.ACTION_DOWN:
              disparo=true;
              break;
       case MotionEvent.ACTION_MOVE:
              float dx = Math.abs(x - mX);
              float dy = (mY - y);
              if (dy<10 && dx>10){
                     giroNave = Math.round((x - mX) / 4);
                     disparo = false;
              } else if (dx<10 && dy>10){
                     aceleracionNave = Math.round((mY - y) / 35);
                     disparo = false;
              }
              break;
       case MotionEvent.ACTION_UP:
              giroNave = 0;
              aceleracionNave = 0;
              if (disparo){
                  ActivaMisil();
              }
              break;
   }
   mX=x; mY=y;
   return true;
}
```

Figura 206: Código Java de la vista "VistaJuego" (XIII)

Los siguientes dos métodos implementan la interfaz *SensorEventListener*

```
//Métodos para el manejo de sensores que giran la nave
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy){}

@Override
public void onSensorChanged(SensorEvent event) {
    float valor = event.values[1];
    if (!hayValorInicial){
        valorInicial = valor;
        hayValorInicial = true;
    }
    giroNave=(int) (valor-valorInicial)/sensibilidadGiroNave ;
}
```

Figura 207: Código Java de la vista "VistaJuego" (XIV)

Una vez que el misil toque una letra de la pantalla esta será destruida, es decir desaparecerá y a su vez emitirá un sonido que indicará que letra ha sido impactada. Cuando esto pase a parte de eliminar el objeto del vector *Grafico*, también se eliminará su identificador del vector de caracteres para que no vuelva a sonar la letra ya destruida.

```
//Método que destruye la letra que es impactada por el misil
private void destruyeLetra(int i) {

    //Vemos que letra hemos destruido
    String sonidoAreproducir = sonidosDeLetras.elementAt(i);

    //Comparamos la letra con todas las que tenemos y activamos su sonido
    if (sonidoAreproducir.equalsIgnoreCase("audioLetraA")){
        soundPool.play(idLetraA, 1, 1, 0, 0, 1);
    }
```

```
//Eliminamos la letra y el sonido que acaba de ser destruido por el misol
Letras.remove(i);
sonidosDeLetras.remove(i);
misilActivo = false; //Eliminamos el misil
puntuacion += 1000; //Sumamos puntuación al usuario por impactar correctamente

//Antes de lanzar efectos de sonido, comprobamos que han sido habilitados en las preferencias
//Usamos getContext por que necesitamos trabajar en este contexto y al ser una vista View
//necesitamos el contexto de la actividad que es Juego.class
pref = PreferenceManager.getDefaultSharedPreferences(getContext());
if (pref.getBoolean("efectos_sonido",true)==true){
    soundPool.play(idExplosion, 1, 1, 0, 0, 1);
}

//Cuando no queden letras pasaremos el juego a un estado de victoria
if (Letras.isEmpty()) {
    estado = ESTADO_VICTORIA;
    salir();
}
```

Figura 208: Código Java de la vista "VistaJuego" (XV)

El misil saldrá desde la misma posición de la nave y sólo estará activo (visible) un tiempo determinado si no impacta contra ningún objeto. Además cuando el misil es disparado y el juego está configurado para ello, se escuchará un pequeño efecto de sonido.

```
//Método que es llamado cuando lanzamos el misil
private void ActivaMisil() {

    misil.setPosX(nave.getPosX());
    misil.setPosY(nave.getPosY());
    misil.setAngulo(nave.getAngulo());
    misil.setIncX(Math.cos(Math.toRadians(misil.getAngulo())) * PASO_VELOCIDAD_MISIL);
    misil.setIncY(Math.sin(Math.toRadians(misil.getAngulo())) * PASO_VELOCIDAD_MISIL);
    tiempoMisil = (int) Math.min(this.getWidth() / Math.abs(misil.getIncX()), this.getHeight() / Math.abs(misil.getIncY())) - 2;
    misilActivo = true;

    //Antes de lanzar efectos de sonido, comprobamos que han sido habilitados en las preferencias
    //Usamos getContext por que necesitamos trabajar en este contexto y al ser una vista View
    //necesitamos el contexto de la actividad que es Juego.class
    pref = PreferenceManager.getDefaultSharedPreferences(getContext());

    if (pref.getBoolean("efectos_sonido",true)==true){
        soundPool.play(idDisparo, 1, 1, 1, 0, 1);
    }
}
```

Figura 209: Código Java de la vista "VistaJuego" (XVI)

5.5.7.2 LETRAS

Tanto el vídeo introductorio, el primer nivel (dar funcionalidad a los botones para que estos reproduzcan sonido) y el segundo nivel de este juego (habilitar la grabación de audio a través del micrófono del dispositivo) ya han sido explicados en el punto *5.1.1 Código compartido por las actividades*. Aun así quedarían algunos puntos a comentar del segundo nivel que sí se explicarán a continuación.

El segundo nivel de este juego se compone de dos actividades, la primera de ellas visualmente es idéntica al primer nivel. Tenemos el botón de ayuda, el botón para avanzar al siguiente nivel cuando el jugador lo crea conveniente y los botones centrales que representarán las vocales mayúsculas y minúsculas, pero a diferencia del nivel anterior que simplemente cuando se tocaba una letra, se escuchaba un audio, esta vez cuando se toca una letra se abre una nueva actividad en forma de diálogo donde le pasaremos información sobre que letra se ha tocado.

```
public void Abrir_Letra_A (View view){

    Intent i = new Intent(this,JuegoVocalesSegundoNivelAbrirLetra.class);
    i.putExtra("letraEnviada", "letraA");
    startActivity(i);
}

public void Abrir_Letra_E (View view){

    Intent i = new Intent(this,JuegoVocalesSegundoNivelAbrirLetra.class);
    i.putExtra("letraEnviada", "letraE");
    startActivity(i);
}

public void Abrir_Letra_I (View view){

    Intent i = new Intent(this,JuegoVocalesSegundoNivelAbrirLetra.class);
    i.putExtra("letraEnviada", "letraI");
    startActivity(i);
}

public void Abrir_Letra_O (View view){

    Intent i = new Intent(this,JuegoVocalesSegundoNivelAbrirLetra.class);
    i.putExtra("letraEnviada", "letraO");
    startActivity(i);
}

public void Abrir_Letra_U (View view){

    Intent i = new Intent(this,JuegoVocalesSegundoNivelAbrirLetra.class);
    i.putExtra("letraEnviada", "letraU");
    startActivity(i);
}
```

Figura 210: Código Java del segundo nivel del juego de las letras (I)

En esta segunda actividad del nivel se recibirán los datos para saber qué imagen de letra establecer y qué audio escuchar para trabajar con la letra que se ha pulsado.

```
//Datos de los vídeos recibidos
Bundle extras = getIntent().getExtras();
String letraEnviada = extras.getString("letraEnviada");

if (letraEnviada.equalsIgnoreCase("letraA")){

    letraIdentificativa ="A";
    letra.setBackgroundResource(R.drawable.letra_a_juego_vocales);
    soundPool.play(idLetra_A, 1, 1, 1, 0, 1);
}

else if (letraEnviada.equalsIgnoreCase("letraE")){

    letraIdentificativa ="E";
    letra.setBackgroundResource(R.drawable.letra_e_juego_vocales);
    soundPool.play(idLetra_E, 1, 1, 1, 0, 1);
}
```

Figura 211: Código Java del segundo nivel del juego de las letras (II)

A continuación se podrá grabar sonidos tal y como se ha explicado en el punto *5.1.1 Código compartido por las actividades*.

El tercer nivel consiste en que el jugador busque una letra, para ello habrá un método que le indique al jugador que letra debe buscar en ese momento. Con la función *Math.random* se obtendrá un número al azar y dependiendo de cuál sea ese número se deberá buscar una letra u otra. Escribiéndola por pantalla (*setText*), la letra será mostrada por pantalla al usuario.

Este código ha sido implementado para que entre turno y turno no se tenga que buscar la misma letra. Así mismo se comprobará si es necesario activar la ayuda por voz.

```
//Método que nos indica la siguiente forma a buscar
public void EstablecerLetraParaBuscar(){

    //Evitamos que se repita la figura a buscar para buscar siempre una diferente a la anterior
    do{
        numeroAleatorioBuscarLetra = (int) Math.floor(Math.random()*5+1); //Valor aleatorio entre 1 y 5
    } while(numeroAleatorioBuscarLetra == numeroAnteriorBuscarLetra);

    numeroAnteriorBuscarLetra = numeroAleatorioBuscarLetra;
    numeroLetraABuscar = numeroAleatorioBuscarLetra;

    if (numeroAleatorioBuscarLetra == 1){

        siguienteLetra.setText("'A' / 'a'");
        //Comprobamos si la ayuda por voz está activada
        if (pref.getBoolean("ayudaPorVoz", true)==true){
            soundPool.play(idSonido_BuscarLetraA, 1, 1, 1, 0, 1);
        }
        letraABuscar = "letraA";
    }
```

Figura 212: Código Java del tercer nivel del juego de las letras (I)

El orden en que las letras están distribuidas en pantalla también irá cambiando entre turno y turno. El proceso es el mismo, mediante un número aleatorio estableceremos combinaciones diferentes de letras en pantalla (hasta 10 combinaciones) y ninguna de ellas podrá repetirse entre turnos.

```
//Método para establecer las figuras en pantalla de forma aleatoria
public void EstablecerDibujoDeLasLetras(){

    //Evitamos que se repita la colocación de las figuras del turno anterior
    do{
        numeroAleatorioEstablecerDibujosLetras = (int) Math.floor(Math.random()*10+1); //Valor aleatorio entre 1 y 5
    } while(numeroAleatorioEstablecerDibujosLetras == numeroAnteriorEstablecerDibujosLetras);

    numeroAnteriorEstablecerDibujosLetras = numeroAleatorioEstablecerDibujosLetras;

    if (numeroAleatorioEstablecerDibujosLetras == 1){

        Letra_1.setBackgroundResource(R.drawable.letra_a_juego_vocales);
        Letra_2.setBackgroundResource(R.drawable.letra_e_juego_vocales);
        Letra_3.setBackgroundResource(R.drawable.letra_i_juego_vocales);
        Letra_4.setBackgroundResource(R.drawable.letra_o_juego_vocales);
        Letra_5.setBackgroundResource(R.drawable.letra_u_juego_vocales);

        ordenDeLetras = 1;
    }

    else if (numeroAleatorioEstablecerDibujosLetras == 2){

        Letra_1.setBackgroundResource(R.drawable.letra_a_minuscula);
        Letra_2.setBackgroundResource(R.drawable.letra_e_minuscula);
        Letra_3.setBackgroundResource(R.drawable.letra_i_minuscula);
        Letra_4.setBackgroundResource(R.drawable.letra_o_minuscula);
        Letra_5.setBackgroundResource(R.drawable.letra_u_minuscula);

        ordenDeLetras = 2;
    }
```

Figura 213: Código Java del tercer nivel del juego de las letras (II)

Por último queda explicar cómo constatar que el jugador ha acertado la letra que se le indica. Los 5 botones centrales tendrán su atributo *onClick* configurado para cuando el jugador pulse en un botón este lance su método establecido en el archivo Java. Ahora que ya sabemos que letra se ha de buscar y sabemos el orden de estas, sabremos si hemos acertado o no, por ejemplo si el jugador tiene que buscar la letra "A" esta solo estará presente en el primer botón en las combinaciones de letras 1 y 2, por lo tanto si se busca la letra A y el orden de las letras es el 1 o el 2 y se pulsa el primer botón, se habrá acertado sino el jugador habrá fallado.

```
public void Letra_1_pulsada (View view){

    if (letraABuscar.equalsIgnoreCase("letraA")){

        if (ordenDeLetras == 1){
            LetraAcertada();
            EstablecerLetraParaBuscar();
            EstablecerDibujoDeLasLetras();
        }
        else if (ordenDeLetras == 2){
            LetraAcertada();
            EstablecerLetraParaBuscar();
            EstablecerDibujoDeLasLetras();
        }
        else{
            LetraFallada();
            EstablecerLetraParaBuscar();
            EstablecerDibujoDeLasLetras();
        }
    }
```

Figura 214: Código Java del tercer nivel del juego de las letras (III)

Cuando el jugador acierte ganará puntos y esto se verá reflejado en forma de una pequeña animación, y si está habilitado, en forma de efecto de sonido. Si por el contrario falla, no ganará puntos ni habrá animación, pero si está habilitado si se escuchará un sonido.

```
//Método que se ejecuta cuando acertamos la figura
public void LetraAcertada(){

    puntuacion = puntuacion + incrementoPuntuacion;

    //Comprobamos si debemos activar lo efectos de sonido
    if (pref.getBoolean("efectos_sonido", true)==true){
        soundPool.play(idSonido_Acierto, 1, 1, 1, 0, 1);
    }

    textoPuntuacion.setText(Integer.toString(puntuacion));
    textoPuntuacion.startAnimation(animacionPuntos);
}

//Método que se ejecuta cuando fallamos la figura
public void LetraFallada(){

    //Comprobamos si debemos activar lo efectos de sonido
    if (pref.getBoolean("efectos_sonido", true)==true){
        soundPool.play(idSonido_Error, 1, 1, 1, 0, 1);
    }

    textoPuntuacion.setText(Integer.toString(puntuacion));
}
```

Figura 215: Código Java del tercer nivel del juego de las letras (IV)

Puede pasar que el ciclo de vida de la actividad pase a un estado de pausarse o detenerse y se podrían perder los datos, para que esto no ocurra habrán dos métodos que serán lanzados de forma automática cuando la actividad pase a esos estados. En el método *onSaveInstanceState* se guardarán los datos que necesitamos y en el método *onRestoreInstanceState* se recuperarán. Cabe añadir que una vez recuperados será necesario volver a establecer los puntos en pantalla y la distribución de las letras.

```
//Guardamos las variables cuando pasemos a un estado de pausa en la aplicación
@Override
protected void onSaveInstanceState(Bundle guardarEstado) {
      super.onSaveInstanceState(guardarEstado);
      guardarEstado.putInt("puntuacion", puntuacion);
      guardarEstado.putInt("numeroLetraABuscar", numeroLetraABuscar);
      guardarEstado.putInt("ordenDeLetras", ordenDeLetras);
}

//Cargamos los datos que teníamos al volver a estar la aplicación activa
@Override
protected void onRestoreInstanceState(Bundle recEstado) {
    super.onRestoreInstanceState(recEstado);
    puntuacion = recEstado.getInt("puntuacion");
    textoPuntuacion.setText(Integer.toString(puntuacion));
```

Figura 216: Código Java del tercer nivel del juego de las letras (V)

Una vez finalice el tiempo mediante una ventana de diálogo se informarán de los puntos conseguidos y se podrá avanzar al siguiente nivel. Esto ocurrirá en cada nivel del juego.

```
//Crea el diálogo de aviso para acceder al siguiente nivel
private void DialogoSiguienteNivel() {

    AlertDialog.Builder dialogoSalir = new AlertDialog.Builder(this);
    dialogoSalir.setMessage("¡NIVEL SUPERADO!\n\nPuntuación: "+puntuacion);
    dialogoSalir.setPositiveButton("Continuar", new DialogInterface.OnClickListener() {

        public void onClick(DialogInterface dialog, int which) {
            // Do nothing but close the dialog
            lanzarSiguienteNivel();
            dialog.dismiss();
        }
    });
    AlertDialog alert = dialogoSalir.create();
    alert.show();
}
```

Figura 217: Código Java del tercer nivel del juego de las letras (VI)

En el cuarto nivel el jugador se encontrará con cosas similares, se le indicará un letra, pero no para buscarla sino para dibujarla. El código para indicar de qué letra se trata será el mismo que el del nivel anterior cuando se buscaba una, por lo tanto no se repetirá su explicación. A su vez en el *punto 5.1.1 Código compartido por las actividades* también se ha explicado como reconocer patrones de escritura y los demás métodos que aparecen en este nivel como la ventana de ayuda o el contador de tiempo se han explicado en dicho punto, por lo tanto una vez se acabe el tiempo aparecerá una ventana de diálogo informando de cómo ha ido el nivel jugado y se avanzará hasta el último nivel.

En este quinto y último nivel aparecerá un dibujo y una palabra relacionada con ese texto a la cual le falta una letra así que se utilizará el reconocimiento de letras mediante una biblioteca *Gesture* para dibujar la letra que falta.

En principio todos los métodos de este nivel (reconocimiento de escritura, cuenta atrás, cuando el jugador acierta o falla y la ventana de ayuda) ya han sido explicados, sólo restaría por comentar el método que irá cambiado las imágenes con su texto por cada turno.

Una vez se ha establecido el *layout* en el método *onCreate* se deberá enlazar las vistas *ImageView* y *TextView* (imagen que aparece cada turno y su texto respectivamente) que previamente se han definido en el archivo Java con sus respectivas vistas en el archivo XML para poder trabajar con ellas desde código.

```
siguienteTexto = (TextView)findViewById(R.id.textoImagenJuegoVocalesQuintoNivel);
siguienteDibujo = (ImageView)findViewById(R.id.imagenJuegoVocalesQuintoNivel);
```

Figura 218: Código Java del quinto nivel del juego de las letras (I)

Habrá 25 imágenes diferentes que pueden aparecer de forma aleatoria. Esto se hará como hasta ahora se ha hecho varias veces, se obtendrá un número al azar y dependiendo de qué número salga, se mostrará cierta imagen. A su vez para la palabra asociada a esta imagen el jugador, de forma aleatoria, le desaparecerá una de las letras y la forma de realizar esto es similar de nuevo, una vez se sabe qué imagen se ha de establecer, se obtendrá un número al azar y dependiendo de qué número sea se eliminará una letra u otra de ese ese texto.

No hay que olvidar que las imágenes usadas como fondo de los botones de los diferentes niveles de los diferentes juegos están en la carpeta *drawable* de los recursos de la aplicación.

```
//Aleatoriamente se establece un dibujo
private void EstablecerDibujoYTexto() {

    //Evitamos que se repita la letra a buscar para buscar siempre una diferente a la anterior
    do{
        numeroAleatorioImagenMostrar = (int) Math.floor(Math.random()*25+1); //Valor aleatorio entre 1 y 25
    } while(numeroAleatorioImagenMostrar == numeroAnteriorImagenMostrar);

    numeroAnteriorImagenMostrar = numeroAleatorioImagenMostrar;
    numeroLetraADibujar = numeroAleatorioImagenMostrar;

    if (numeroAleatorioImagenMostrar == 1){
        siguienteDibujo.setBackgroundResource(R.drawable.abeja);
        numeroAleatorioLetraSecundaria = (int) Math.floor(Math.random()*3+1);
        if (numeroAleatorioLetraSecundaria == 1){
            siguienteTexto.setText("_beja");
            letraADibujar = "letraA";
        }
        else if (numeroAleatorioLetraSecundaria == 2){
            siguienteTexto.setText("Ab_ja");
            letraADibujar = "letraE";
        }
        else if (numeroAleatorioLetraSecundaria == 3){
            siguienteTexto.setText("Abej_");
            letraADibujar = "letraA";
        }

    }
```

Figura 219: Código Java del quinto nivel del juego de las letras (II)

Como siempre, una vez acabe el tiempo mediante una ventana de diálogo se informará de los puntos finales conseguidos y se volverá al menú principal tal y como se ha explicado en el punto *5.1.1 Código compartido por las actividades*, volviendo actividad a actividad transfiriendo la puntuación final y cerrando actividades.

5.5.7.3 NÚMEROS

Al igual que pasa con el juego anterior, tanto el vídeo introductorio, el primer nivel (dar funcionalidad a los botones para que estos reproduzcan sonido) y el segundo nivel de este juego (habilitar la grabación de audio a través del micrófono del dispositivo) ya han sido explicados en el punto *5.1.1 Código compartido por las actividades*, tan solo diferiría en que en lugar de usar las imágenes de letras almacenadas en la carpeta *drawable* de los recursos, apareciesen imágenes de números.

A su vez en el tercer nivel el jugador tendría que dibujar el número que se le indicase, por lo tanto el código, a excepción de usar una librería *Gesture* de números, sería igual que el cuarto nivel del juego explicado anteriormente. Por lo tanto se pasará directamente a explicar el cuarto nivel de este juego, que si difiere de los demás.

En él se tendrá que usar una biblioteca *Gesture* de números para contar la cantidad de objetos que aparecen por pantalla, aquí lo realmente interesante (aparte de reconocer que número dibuja el jugador por pantalla, que ya se ha hecho en el punto *5.1.1 Código compartido por las actividades*) es explicar cómo en cada turno cambia la cantidad de objetos a mostrar para ser contados.

En principio, recordando el archivo XML se tenían nueve botones que iban a ser usados como los objetos o imágenes para contar, pero no se había establecido todavía ninguna imagen, eran botones "vacíos". Bien, ahora para trabajar a partir de esto, en esta actividad se definen nueve vistas de tipo botón que se serán los nueve botones antes mencionados y estos serán enlazados con sus respectivas vistas en el archivo XML para poder trabajar con ellos desde código (esta parte, como siempre, se hará en el método *onCreate*).

```
objeto1 = (Button)findViewById(R.id.objeto1_juego_numeros);
objeto2 = (Button)findViewById(R.id.objeto2_juego_numeros);
objeto3 = (Button)findViewById(R.id.objeto3_juego_numeros);
objeto4 = (Button)findViewById(R.id.objeto4_juego_numeros);
objeto5 = (Button)findViewById(R.id.objeto5_juego_numeros);
objeto6 = (Button)findViewById(R.id.objeto6_juego_numeros);
objeto7 = (Button)findViewById(R.id.objeto7_juego_numeros);
objeto8 = (Button)findViewById(R.id.objeto8_juego_numeros);
objeto9 = (Button)findViewById(R.id.objeto9_juego_numeros);
```

Figura 220: Código Java del cuarto nivel del juego de los números (I)

Una vez se puedan usar estas vistas desde código, se mostrará el método que hace aparecer cierta cantidad de imágenes de forma aleatoria por pantalla turno a turno.

Mediante el método *Math.random* se obtendría un número al azar entre 1 y 9, y este número sería el número de imágenes que aparecerían por pantalla, pero para que no aparezcan siempre en el mismo orden y estructura se usará de nuevo el método *Math.random* para que si por ejemplo se tiene que mostrar cuatro imágenes por pantalla que no se muestren siempre en las posiciones uno, dos tres y cuatro, todas en fila, sino que esas posiciones vayan cambiando.

```
private void EstablecerCantidadObjetosAContar() {

    //Evitamos que se repita el número
    do{
        //Valor aleatorio entre 1 y 9
        numeroAleatorioDibujarNumero = (int) Math.floor(Math.random()*9+1);
    } while(numeroAleatorioDibujarNumero == numeroAnteriorDibujarNumero);

    numeroAnteriorDibujarNumero = numeroAleatorioDibujarNumero;
    numeroNumeroADibujar = numeroAleatorioDibujarNumero;

    if (numeroAleatorioDibujarNumero == 1){
        objeto1.setBackgroundResource(0);
        objeto2.setBackgroundResource(0);
        objeto3.setBackgroundResource(R.drawable.oso_juego_numeros);
        objeto4.setBackgroundResource(0);
        objeto5.setBackgroundResource(0);
        objeto6.setBackgroundResource(0);
        objeto7.setBackgroundResource(0);
        objeto8.setBackgroundResource(0);
        objeto9.setBackgroundResource(0);

        numeroADibujar = "uno";
    }
```

Figura 221: Código Java del cuarto nivel del juego de los números (II)

```
else if (numeroAleatorioDibujarNumero == 4){
    //Es posible que los dibujos aparezcan de formas distintas
    int variasOpcionesParaCuatroObjetos = (int) Math.floor(Math.random()*1+1);
    if (variasOpcionesParaCuatroObjetos == 1){
        objeto1.setBackgroundResource(R.drawable.oso_juego_numeros);
        objeto2.setBackgroundResource(R.drawable.oso_juego_numeros);
        objeto3.setBackgroundResource(R.drawable.oso_juego_numeros);
        objeto4.setBackgroundResource(R.drawable.oso_juego_numeros);
        objeto5.setBackgroundResource(0);
        objeto6.setBackgroundResource(0);
        objeto7.setBackgroundResource(0);
        objeto8.setBackgroundResource(0);
        objeto9.setBackgroundResource(0);
    }
    else if (variasOpcionesParaCuatroObjetos == 2){
        objeto1.setBackgroundResource(0);
        objeto2.setBackgroundResource(R.drawable.oso_juego_numeros);
        objeto3.setBackgroundResource(R.drawable.oso_juego_numeros);
        objeto4.setBackgroundResource(0);
        objeto5.setBackgroundResource(0);
        objeto6.setBackgroundResource(0);
        objeto7.setBackgroundResource(R.drawable.oso_juego_numeros);
        objeto8.setBackgroundResource(R.drawable.oso_juego_numeros);
        objeto9.setBackgroundResource(0);
    }
    numeroADibujar = "cuatro";
}
```

Figura 222: Código Java del cuarto nivel del juego de los números (III)

Por último, en el quinto nivel de este juego, también se usará el reconocimiento de números a través de una biblioteca *Gesture* pero para solucionar sencillas sumas. Por lo tanto, como los métodos que se usan en esta actividad ya se han repasado varias veces se explicará el método en el cual se establecen las sumas y como estas se verifican si son correctas o no.

Para establecer las sumas se necesitan dos métodos, uno de ellos para hacer la suma en sí (*EstablecerSuma*) y otro para establecer la imagen de los dos números que serán sumados (*EstablecerNumerosASumar*).

Para el primer método, se obtendrán dos números de forma aleatoria (el resultado de la suma no podrá dar un valor mayor que nueve) y se sumarán de manera que se obtendrá una cadena de caracteres, la cual será comparada con la biblioteca *Gesture* de números para verificar si el número dibujado por el usuario es el correcto, tal y como se ha explicado en el punto *5.1.1 Código compartidos por las actividades*.

```
//Método para establecer que suma calcular
public void EstablecerSuma() {

    do{
        aleatorioPrimerNumero = (int) Math.floor(Math.random()*9+1);
        aleatorioSegundoNumero = (int) Math.floor(Math.random()*9+1);

        sumaAleatoria = aleatorioPrimerNumero + aleatorioSegundoNumero;
    } while (sumaAleatoria > 9);

    EstablecerNumerosASumar();

    if (sumaAleatoria == 1){
        numeroADibujar = "uno";
    }
    else if (sumaAleatoria == 2){
        numeroADibujar = "dos";
    }
    else if (sumaAleatoria == 3){
        numeroADibujar = "tres";
    }
    else if (sumaAleatoria == 4){
        numeroADibujar = "cuatro";
    }
    else if (sumaAleatoria == 5){
        numeroADibujar = "cinco";
    }
    else if (sumaAleatoria == 6){
        numeroADibujar = "seis";
    }
    else if (sumaAleatoria == 7){
        numeroADibujar = "siete";
    }
```

Figura 223: Código Java del quinto nivel del juego de los números (I)

Para establecer las imágenes de los dos números por pantalla se comprobará que número ha salido y se establecerá de imagen de vista su respectivo número guardado en la carpeta *drawable* de los recursos.

```
//Establecemos la imagen para los número a sumar
private void EstablecerNumerosASumar(){

    if (aleatorioPrimerNumero == 1){
        primerNumero.setBackgroundResource(R.drawable.numero_1_juego_numeros);
    }
    else if (aleatorioPrimerNumero == 2){
        primerNumero.setBackgroundResource(R.drawable.numero_2_juego_numeros);
    }
    else if (aleatorioPrimerNumero == 3){
        primerNumero.setBackgroundResource(R.drawable.numero_3_juego_numeros);
    }
    else if (aleatorioPrimerNumero == 4){
        primerNumero.setBackgroundResource(R.drawable.numero_4_juego_numeros);
    }
    else if (aleatorioPrimerNumero == 5){
        primerNumero.setBackgroundResource(R.drawable.numero_5_juego_numeros);
    }
```

Figura 224: Código Java del quinto nivel del juego de los números (II)

Como ya se ha comentado, los métodos que activan la animación y efectos de sonido cuando el jugador acierta o falla, muestra la pantalla de ayuda o el tiempo ya se han explicado en otros niveles.

Acabado el juego, se mostrarán los puntos finales conseguidos por el jugador y estos serán retornados actividad a actividad hasta llegar al menú principal para ser guardados.

CAPÍTULO 6
VISIÓN DE FUTURO Y CONCLUSIONES

6.1 VISIÓN DE FUTURO

Todo proyecto comienza con muchas ideas, las cuales se quieren plasmar todas sin excepción, pero disponer de un tiempo limitado para finalizar un proyecto hace que se tengan que descartar dichas ideas para implementarlas en futuras versiones de la aplicación. En este sub apartado se comentarán esas ideas:

- ***Mini-juego "Captura las letras":*** Aparición de una lista de letras a capturar en cierto orden que habría que seguir. Es decir, en principio el juego sería el mismo, se le presentaría al jugador una nave que dispara "lápices" para impactar y "capturar" las letras que flotan por la pantalla, pero en lugar de disparar sin criterio alguno, habría una serie de letras predefinidas aleatoriamente en cada partida que serían mostradas al usuario y este debería "capturar" las letras según ese orden establecido, dotando al juego de un mayor reto.

- ***Mini-juego "Pinta y colorea":*** Ampliación de la cantidad de colores y grosores de pincel a seleccionar por el jugador para darle a este mayores opciones y posibilidades.

- ***Puntuaciones:*** Posibilidad de subir las puntuaciones obtenidas por los diferentes juegos a un servidor de internet y consultar las listas de puntuaciones a través de una página web, lo que aumentaría la competitividad ya que se daría la posibilidad de acceder a las puntuaciones de cualquier jugador a través de internet.

- ***Juegos:*** Aumentar la cantidad de niveles disponibles tanto para los juegos de letras como de números.

6.2 CONCLUSIONES

Cuando comencé este proyecto, me planteé ciertos objetivos. Quería enfrentarme a una tecnología que prácticamente fuese desconocida para mí y aprender a trabajar con ella superando los problemas que pudiesen surgir por el camino.

Elegir esa tecnología fue sencillo, habiendo realizado un curso de iniciación a Android, atrayéndome la idea de cómo se desarrollaba para esta plataforma y viendo que en dicho curso se daban pinceladas de todo lo que se podría hacer y aprender, quise profundizar más en el tema. Además tenía el conocimiento de que desarrollar para Android suponía poder llegar a un gran público de manera rápida y económica.

Una vez acabé de formarme a través de libros y tutoriales surgió la siguiente pregunta: *¿Qué aplicación desarrollar?*

Dándole vueltas a esa pregunta me surge la idea de plasmar un poco de cada una de las cosas que he ido aprendiendo en una aplicación y dicha aplicación tenía que ser útil. Finalmente este proyecto es el resultado de esa idea.

Al principio de este documento se plantearon 3 objetivos que eran los siguientes:

- Ofrecer una visión general de los distintos dispositivos móviles actuales disponibles, analizando las posibilidades que nos ofrecen cada uno de ellos.
- Conocer Android, su arquitectura y sus características.
- Adentrarse en el desarrollo para Android pasando por sus 3 fases: Análisis, diseño y desarrollo.

Y una vez se ha finalizado este documento y este proyecto he de decir que se han cumplido los objetivos planteados. Se ha explicado las muchas alternativas que tenemos a la hora de desarrollar para plataformas móviles y por qué hemos elegido Android para ello. Además se ha explicado qué es Android y las muchas características que lo componen.

Por último me gustaría añadir que dicho todo esto, se ha cumplido mi objetivo personal que consistía en adquirir las capacidades necesarias para desarrollar una aplicación para Android desde cero aplicando los conocimientos que he ido adquiriendo a través de la consulta de libros o superando las dificultades y errores que he podido cometer cuando desarrollaba la aplicación que se explica en este documento.

En conclusión este proyecto me ha servido para satisfacer todos los objetivos planteados y me abre la posibilidad de adentrarme en un mundo que desconocía como es el desarrollo de software para plataformas móviles.