UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de videojuegos sobre la plataforma Android

Proyecto Final de Carrera

Ingeniería Técnica Superior en Informática de Gestión

Autor: Sergio Musoles Tornador

Director: Juan Vicente Capella Hernández

28/09/2014

Resumen

Este PFC está enfocado a la elaboración de tres videojuegos para dispositivos móviles con sistema operativo Android. Para ello se ha estudiado la arquitectura del sistema operativo Android, así como las herramientas necesarias para el desarrollo de aplicaciones para Android. Tras esto, se ha pasado a investigar y aprender el funcionamiento de una librería *Open Source* para el desarrollo de videojuegos (*AndEngine*). Y como conclusión y aplicación de lo aprendido, se han desarrollado tres videojuegos utilizando la librería elegida y el resto de herramientas necesarias.

Palabras clave: Android, videojuego, AndEngine, java, eclipse

Tabla de contenidos

1. Introducción 9
1.1 Motivación 9
1.2 Descripción del proyecto 11
1.3 Objetivos del proyecto 12
1.4 Estructura de la memoria 14
2. Tecnología utilizada 15
2.1 ¿Qúe es Android? 16
2.2 Características principales 17
2.3 Arquitectura del sistema 19
2.4 Máquina virtual Dalvik 22
2.5 Partes de una aplicación 23
2.6 Ciclo de vida de una aplicación 30
2.7 Entorno de desarrollo de Android 32
2.8 Análisis motores de juego 2D 34
2.9 API AndEngine 38
3. Videojuego I: Vestir princesas 42
3.1 Descripción del videojuego 42
3.2 Análisis y especificación de requisitos 45
3.3 Diseño 48
3.4 Implementación 53
4. Videojuego II: Explotar granos 59
4.1 Descripción del videojuego 59
4.2 Análisis y especificación de requisitos 62
4.3 Diseño 65
4.4 Implementación 69

5. Videojuego III: Minicars 72
5.1 Descripción del videojuego…….. 72
5.2 Análisis y especificación de requisitos…….……74
5.3 Diseño ……...77
5.4 Implementación ……80
6. Planificación y costes 83
6.1 Planificación del proyecto ……83
6.2 Costes del proyecto……... 85
6.3 Pruebas ……..87
7. Conclusiones 88
7.1 Conclusiones generales ……88
7.2 Conclusiones personales……... 90
7.3 Trabajo futuro……... 91
8. Bibliografía 92
Anexo I: Monetización de apps 93
Anexo II: Manual "Vestir princesas" 94
Anexo III: Manual "Explotar granos"……..96
Anexo IV: Manual "Minicars"…….97

1. Introducción

En este primer capítulo de la memoria se explica la motivación que me hizo llevar a cabo este proyecto, así como la descripción detallada de dicho proyecto. Sin olvidar los objetivos que el proyecto persigue. Y, por último, una guía acerca de los diferentes capítulos de la memoria, por tal de facilitar la lectura de esta.

1.1 Motivación

La tecnología móvil evoluciona a pasos agigantados y nos presenta en la actualidad el llamado teléfono inteligente (o *smartphone* en inglés). Los teléfonos inteligentes se caracterizan por el hecho de ser ordenadores de bolsillo con unas posibilidades cada vez más cercanas a los ordenadores de sobremesa.

Son varios los sistemas operativos que existen para teléfonos inteligentes, sin embargo, Android está pegando fuerte en el mercado actual, al tratarse de software libre que además da multitud de facilidades tanto a desarrolladores que quieran desarrollar aplicaciones, como a usuarios que quieren gestionar su información sin restricciones.

La curiosidad por estos nuevos dispositivos (y todos los *gadgets* que estos incorporan), que cada vez están más presentes en la sociedad, junto con la curiosidad por el S.O. de Google en concreto, es lo que me ha llevado a tratar este tema en mi proyecto de final de carrera.

Por otro lado, tenemos los videojuegos, cada vez más presentes en nuestra sociedad. Una forma relativamente nueva de expresarnos y enviar nuestros conocimientos o emociones a los jugadores, que los recibirán y percibirán de una forma diferente a como lo hacen cuando ven una película o leen un libro.

Los videojuegos han dejado de ser un producto enfocado a un cierto sector para transformarse en algo que cualquier persona en cualquier lugar puede disfrutar. Gracias a los smartphones podemos ver a todo tipo de personas jugando en el metro o en el autobús sin necesidad de adquirir un dispositivo únicamente para ello, se puede hacer con el mismo móvil con el que se realizan llamadas o se consulta el tiempo.

Los videojuegos son tan importantes para un dispositivo móvil que gran parte del éxito de un S.O. depende de las facilidades que este dé para desarrollar videojuegos y, por tanto, del catálogo que este ofrezca a sus usuarios. Además, los bajos costes de desarrollo y el bajo precio de salida de este tipo de productos hace que sea un sector en continua expansión que mueve miles de millones de euros al año, con enormes empresas que empiezan a desarrollar para Iphone o Android, lo que lo hace un tema muy interesante para la investigación y con unas interesantes perspectivas de futuro.

Desde siempre me ha fascinado la idea de poder crear un mecanismo de juego, unos escenarios, unos personajes, etc. y que las personas puedan llegar a divertirse, disfrutar o aprender con ello, así que desde el principio pensé en combinar ambas cosas y de ahí salió la idea del proyecto.

Además, se ha dado la posibilidad de desarrollar este proyecto en el marco de un convenio con una empresa, teniendo que proponer la idea tanto al jefe de la empresa como al tutor del proyecto en la universidad. De esta manera, se vivirá la experiencia del desarrollo de videojuegos en un entorno laboral real, en una empresa dedicada al desarrollo de aplicaciones y videojuegos para móvil. Y así, vivir en primera persona como está creciendo el mundo de las aplicaciones y videojuegos para móvil, como se puede conseguir dinero en este negocio, y ver la evolución y crecimiento de una pequeña empresa en este mundo.

1.2 Descripción del proyecto

El proyecto consiste en el estudio, análisis y puesta en práctica del desarrollo de videojuegos en dos dimensiones sobre la plataforma Android, así como la monetización de dichos videojuegos (ganar dinero gracias a nuestros videojuegos).

En concreto se ha estudiado el entorno de programación disponible y las herramientas y conceptos que se deben utilizar para desarrollar aplicaciones para el sistema operativo de Android, así como el funcionamiento interno de dicho sistema operativo.

A continuación se ha profundizado en el desarrollo de videojuegos para Android en dos dimensiones. Así que hemos analizado diferentes herramientas y librerías y hemos escogido la que mejor se acoplaba a nuestras necesidades para el desarrollo de videojuegos. El siguiente paso ha sido estudiar y coger soltura con la opción elegida (*AndEngine*) para así pasar al desarrollo de nuestros videojuegos.

En la elección hemos tenido en cuenta la comunidad de gente (así como los ejemplos disponibles) que hay detrás de la opción elegida, y la dificultad de aprendizaje que tiene dicha librería.

Como parte importante del proyecto y demostración de lo aprendido, se han desarrollado diferentes videojuegos con diferentes características:

-En el primer videojuego se han puesto a prueba los conocimientos adquiridos durante la fase de estudio y análisis del proyecto, así como los conocimientos adquiridos durante la carrera. Se trata de un juego sencillo enfocado a los/as niños/as pequeños/as en el que el manejo del juego consiste en arrastrar y soltar (*drag and drop*). Es un juego donde podrás vestir a una muñeca con las diferentes prendas de ropa disponibles (camisas, pantalones, zapatos, complementos, etc...), y cambiarle las partes de la cara (pelo, ojos y boca) para acabar con una muñeca a tu gusto.

-El segundo juego tiene un funcionamiento diferente, en este tendrás que realizar una acción en la que cada pantalla irá aumentando la dificultad, y podrás ir superando nivel a nivel hasta llegar al final. Este juego está basado en la típica máquina de feria en la que tienes que golpear a los topos que salen de su madriguera, pero este tiene un toque de humor juvenil, ya que consiste en ir explotando los granos que le van apareciendo al protagonista o a la protagonista por toda la cara. Existen diferentes niveles de dificultad que tendrás que ir superando poco a poco, así como diferentes elementos que podrán aparecer en la cara, lo que hará que aumente o disminuya tu puntuación hasta alcanzar el siguiente nivel o fracasar en el intento.

-El tercer juego es del tipo de carreras, donde se tendrán que ir superando los diferentes niveles (circuitos) para ir desbloqueando nuevos circuitos. La forma de superar cada nivel será por el tiempo en que se tarda en completar el circuito. En este juego entran en juego los motores de físicas para detectar las colisiones de los vehículos con los diferentes elementos del circuito (obstáculos en la carretera, las paredes del circuito, etc.).

1.3 Objetivos del proyecto

El objetivo primordial del proyecto que se plantea es conocer la arquitectura del sistema operativo Android, así como aprender el funcionamiento de algún motor gráfico externo, para así desarrollar diferentes videojuegos y sacar beneficio económico de ellos.

La primera fase del proyecto consiste en estudiar y conocer el funcionamiento interno de Android, algo necesario para crear aplicaciones o videojuegos para este sistema operativo.

La segunda fase consistirá en analizar, estudiar, y preparar el entorno de desarrollo necesario para poder iniciar la construcción de nuestros videojuegos. En esta fase también deberemos elegir el motor gráfico a utilizar, y aprender su funcionamiento.

En la tercera fase, en la que ya tenemos todos los conocimientos necesarios para iniciar el desarrollo de nuestros videojuegos, pasamos a implementar las versiones finales de nuestros videojuegos. El objetivo de esta fase es obtener tres videojuegos completamente funcionales, con los que el usuario pueda interactuar, utilizando el motor gráfico elegido.

Asumimos una cuarta fase en la cual estudiaremos las posibilidades de monetización de los videojuegos para sacar beneficios a estos, y aplicaremos el modelo de monetización elegido a al menos uno de nuestros productos. Y tras esto, se sacará al mercado al menos uno de los tres videojuegos.

En resumen, estos son los objetivos en la realización de este proyecto:

Objetivos principales

- Estudiar el funcionamiento interno de Android.
- Analizar y estudiar el entorno de desarrollo y las herramientas internas o externas que el sistema operativo Android tiene para el desarrollo de videojuegos.
- Desarrollar tres videojuegos en dos dimensiones utilizando las herramientas seleccionadas en el análisis y sacarlos al mercado.
- Estudio de las posibilidades de monetización de los videojuegos y aplicar lo aprendido en los videojuegos desarrollados.

Objetivos concretos

Los objetivos concretos a alcanzar en lo que respecta a los tres productos que se desarrollarán son:

1. Realizar tres productos funcionales, de diferente dificultad y diferente tipo de juego, en el que se integren los conocimientos adquiridos.
2. Sacar al mercado al menos uno de dichos productos y monetizarlos para sacar beneficio económico a nuestros productos.
3. Obtener la capacidad de acotar un coste en tiempo y dinero para la realización de un videojuego sobre la plataforma Android a través de la experiencia adquirida con los tres productos construidos.

1.4 Estructura de la memoria

En este apartado se describe la estructura que presenta la memoria, por tal de facilitar la lectura de esta. Este documento está dividido en un total de 8 apartados, los cuales pasamos a detallar a continuación.

En primer lugar, el capítulo 1 como vemos, presenta la introducción, la motivación del proyecto, la descripción de este y los objetivos a cumplir. El capítulo 2, en cambio, trata acerca del sistema operativo Android y, entre otras cosas, se describe su estructura y el funcionamiento de sus aplicaciones. Además, también encontraremos el análisis de las diferentes herramientas y motores de juego para el desarrollo de videojuegos en Android. Como conclusión de la parte de análisis y estudio de diferentes motores gráficos, haremos un estudio de las características del motor gráfico AndEngine, que es el elegido para desarrollar los videojuegos de este proyecto.

Una vez que ya conocemos los conceptos básicos acerca de Android y del motor gráfico *AndEngine*, en el capítulo 3 se habla sobre el primer videojuego desarrollado, con nombre “Vestir princesas”. En este apartado podremos ver una descripción general de este, así como las partes más importantes de su análisis y especificación, y de su diseño. También explicaremos las cosas más importantes de su implementación.

En el capítulo 4 se hablará sobre el siguiente videojuego desarrollado, con nombre “Explotar granos”. Y de la misma manera, en el capítulo 5 se hablará del último videojuego desarrollado, con nombre “Minicars”, en el que además se hará hincapié en las funciones utilizadas del motor de física Box2D utilizado para el desarrollo de este videojuego.

A continuación, tenemos el capítulo 6, en él se presenta la planificación del proyecto, el orden en que se han realizado las diferentes tareas, y como no, el apartado donde se detalla el coste económico del proyecto.

Para finalizar, en el capítulo 7 se presentan las conclusiones del proyecto, y en el capítulo 8 detallaremos la bibliografía.

Sin embargo, no se acaba aquí la memoria, pues también hay cuatro anexos. En el primero explicaremos las diferentes maneras de monetizar una aplicación o videojuego móvil. En el segundo se encuentra el manual de usuario de “Vestir princesas” donde se explica, a nivel de usuario, todo lo necesario para poder jugar al videojuego. En el tercero se encuentra el manual de usuario de “Explotar granos”, así como en el cuarto anexo se encuentra el manual de usuario de “Minicars”.

2. Tecnología utilizada

Cuando nos disponemos a desarrollar una aplicación para un nuevo sistema operativo con el que no hemos trabajado con anterioridad, lo primero que hay que hacer es estudiar el funcionamiento de este, así como el funcionamiento de sus aplicaciones. Especialmente si se trata de un sistema operativo móvil, pues estos son bastante cerrados y las aplicaciones que desarrollemos para ellos se estructurarán de forma diferente a como se estructuran en otros sistemas operativos.

Igual o más importante es analizar las diferentes herramientas con las que podemos trabajar para desarrollar una aplicación que funcione en el sistema operativo objetivo, por tal de poder escoger las herramientas que más se ajustan a nuestros requisitos. Y, por último, también es útil llevar a cabo un análisis de las aplicaciones del mismo ámbito desarrolladas para dicho sistema operativo, pues estas nos pueden ayudar a decidir los objetivos concretos de nuestra aplicación y a evitar errores durante el desarrollo.

La elección del sistema operativo utilizado ha sido Android, ya que la programación de aplicaciones y juegos para este sistema operativo se puede hacer en el lenguaje Java, que ha sido el que más se ha practicado a lo largo de los estudios universitarios. Otros punto a favor por el que también ha sido elegido este sistema operativo es porque es libre, y también porque las herramientas de desarrollo básicas necesarias son gratuitas y se pueden instalar en cualquier sistema operativo.

En resumen, el estudio y análisis del sistema operativo Android ha sido un primer paso muy importante antes de embarcarnos en el desarrollo de los videojuegos. En este capítulo se explican los conceptos más importantes estudiados y se exponen los análisis llevados a cabo para poder desarrollar videojuegos para Android cometiendo el mínimo número de errores posibles en todos los ámbitos.

Empezaremos por una explicación de qué es el sistema operativo Android. Seguiremos con la estructura de las aplicaciones para este sistema operativo. Y para concluir propondremos el entorno de desarrollo que Google ha preparado para el desarrollador de Android.

Por último, se describirá el análisis de las herramientas disponibles para llevar a cabo el desarrollo de videojuegos, con las ventajas e inconvenientes de las diferentes alternativas.

2.1 ¿Qué es Android?

Android es un conjunto de software que constituye un ecosistema para las aplicaciones móviles. Dentro de este conjunto se incluye un sistema operativo móvil, lo que significa que Android está dirigido principalmente a teléfonos inteligentes (o *smartphones*) y a *tablets*. Y detrás de este software se encuentra la empresa Google Inc., archiconocida multinacional dedicada a ofrecer servicios y productos basados generalmente en Internet.

Android está basado en Linux, de hecho, el núcleo del sistema operativo está basado en el kernel [1] 2.6 de este. El desarrollo de este sistema operativo no fue iniciado por Google, sino que lo inició una empresa llamada Android Inc., que poco tiempo después despertó el interés de Google y se vio absorbida por esta, manteniendo eso si, el nombre de la idea originaria.

Actualmente el desarrollo de la plataforma Android, se lleva a cabo mediante la Open Handset Alliance, que es un consorcio creado en el 2007 con el objetivo común de desarrollar estándares para dispositivos móviles. Este consorcio está liderado por Google. El primer código que vio la luz a raíz de la creación de este consorcio, fue Android.

En octubre de 2008 se hace público el proyecto OpenSource conocido como Android, el sistema operativo para dispositivos móviles desarrollado en su mayor parte por Google, y que pasa a distribuirse bajo una licencia apache 2.0.

Junto con esta publicación se inicia la andadura del Android Market, plataforma que usa Android para la distribución de sus aplicaciones, y se lanza al mercado el primer terminal android comercial, conocido como HTC Dream o G1.

El éxito de Android se consolida en noviembre de 2009, cuando el Motorola Droid/Milestone, supera el récord de ventas del dispositivo iPhone, desarrollado por Apple, considerado su mayor competidor en el sector de los dispositivos móviles.

[1] Principal responsible de facilitar a los distintos programas acceso seguro al hardware del ordenador y gestionar los recursos del sistema.

2.2 Características principales

La principal característica del sistema operativo de Google es su optimización, todo su desarrollo se basa en la optimización de recursos por parte de este, con el objetivo de conseguir un sistema fluido en su funcionamiento sobre los dispositivos móviles y una buena optimización de batería.

La ejecución de aplicaciones se basa en el uso de la Máquina Virtual Dalvik[2], a la que se aplicó una optimización para su correcto funcionamiento en dispositivos móviles. Las aplicaciones que ejecuta el sistema están escritas en Java y empaquetadas en un formato conocido como Android Package (.apk).

Cada una de las aplicaciones del sistema es completamente independiente de todas las demás, lanzan cada una su propio proceso sobre el kernel linux sobre el que se ejecuta el sistema, cada una usa su propia máquina virtual Java con el fin de que el uso de diferentes aplicaciones no se interfiera entre sí.

Cada una de las aplicaciones que se ejecutan recibe su propio ID de usuario, al que se le asignan una serie de permisos, que se usan para poder acceder de manera única a los ficheros generados por la aplicación, garantizando así la seguridad de las diferentes aplicaciones y del sistema en general.

Formó parte del desarrollo del sistema la integración de un navegador web basado en WebKit, con el objetivo de hacer la experiencia de navegación del usuario más rápida, incorporando también un motor de optimización de páginas webs para el soporte de dispositivos móviles.

Otra de las características que cabe mencionar, es el desarrollo de una librería de gráficos 2D nativa, que ayudará a asumir el objetivo general del sistema, la rapidez y fluidez de este, o en términos más técnicos, la optimización de los recursos del dispositivo en su grado máximo. Para el uso de gráficos 3D, se confió inicialmente en la especificación OpenGL 1.0, la cual se ha ido actualizando de forma conjunta al sistema operativo.

Android incluyó también un motor de bases de datos adaptado a dispositivos móviles, en ese caso el motor elegido es SQLite[3] , que permite el almacenamiento de datos de forma local, lo que puede suponer una limitación, pero aporta gran sencillez de cara al desarrollador para usarla desde sus aplicaciones.

Dado el crecimiento de los dispositivos móviles y el gran número de capacidades que nos ofrecían, el desarrollo del sistema incluyó soporte para multitud de archivos multimedia, como podrían ser MP3, JPEG, MPEG4, etc., el cual como la mayoría de las características, se ha ido incrementando con cada una de las actualizaciones del sistema.

---

[2] Máquina virtual utilizada por la plataforma Android para ejecutar sus aplicaciones.

[3] Sistema reducido de gestión de base de datos relacional.

Otra característica importante era ofrecer control para todo el hardware adicional que estaban incluyendo los fabricantes en los dispositivos móviles, como serían acelerómetros, GPS, cámaras, 3G, Bluetooth, etc., además por supuesto de la telefonía GSM[4].

Por último Android creó un SDK[5] completo, con *debugger* y un *plugin* IDE[6] para el entorno de desarrollo Eclipse, facilitando así a los desarrolladores introducirse en este nuevo sistema, consiguiendo así una gran comunidad de investigadores que contribuyeran a la evolución del sistema.

4 Sistema global para las comunicaciones móviles establecido como estándar, libre de cargos, para la telefonía móvil digital.
5 *Software Development Kit*, conjunto de herramientas necesario para el desarrollo de aplicaciones para una plataforma o sistema operativo concreto.
6 *Integrated Development Environment*, programa informático compuesto por un conjunto de herramientas de programación.

2.3 Arquitectura del sistema

De igual forma que la gran mayoría de los sistemas operativos, y dado que está basado en Linux, adopta la arquitectura de este, es decir, una serie de capas, en las que se organizan cada uno de los componentes del sistema.

En la capa más profunda del sistema encontramos el *kernel*, la base del sistema operativo, en la capa inmediatamente superior encontramos las librerías del sistema, entre las que está contenido el *Android Runtime*, en la siguiente capa encontramos el *Application Framework,* y finalmente, en la capa superior, que es con la que interactúa el usuario, encontramos las aplicaciones.

El siguiente esquema nos da una idea detallada de la arquitectura del sistema operativo Android.

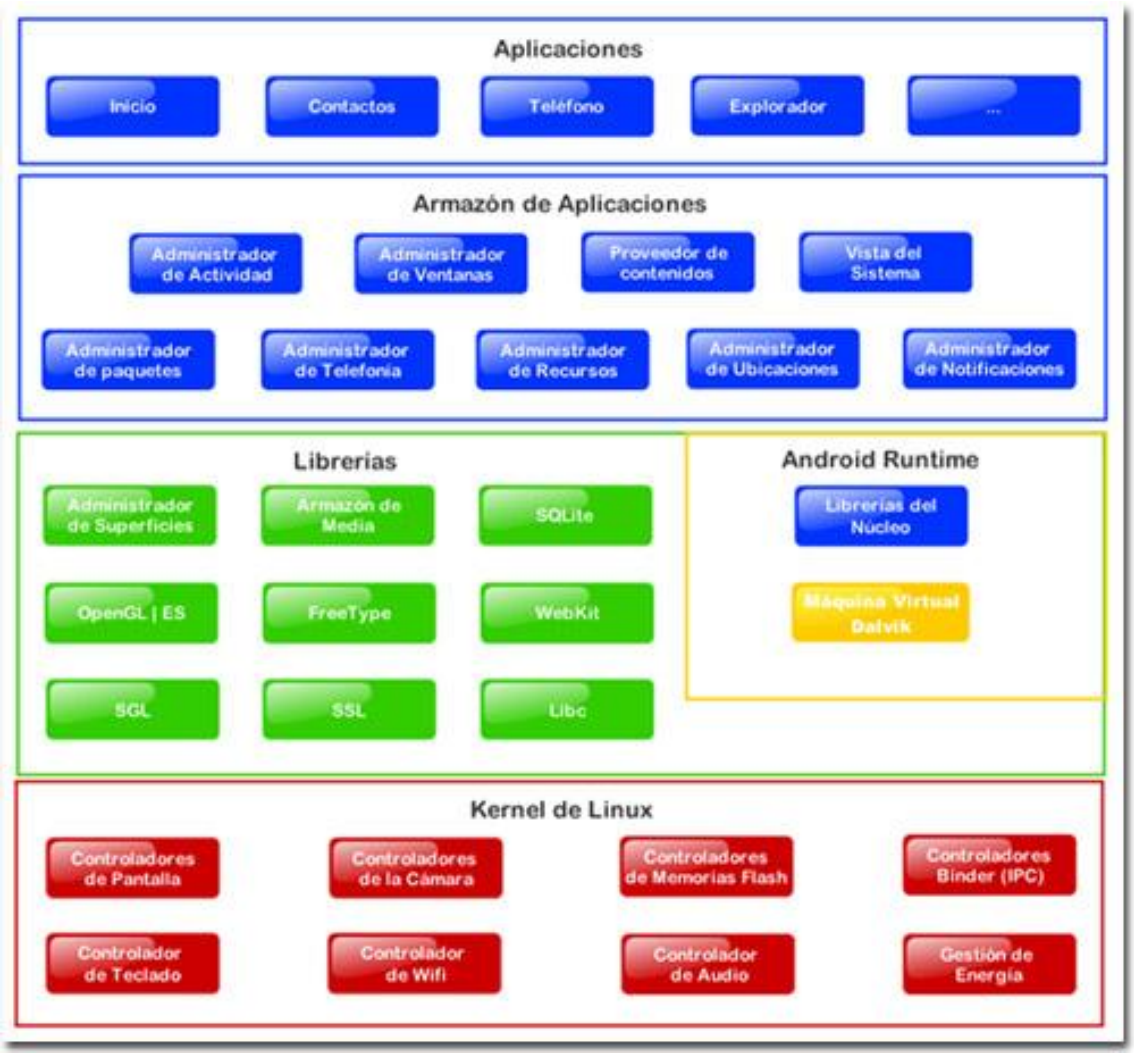

Imagen 1: Estructura interna del sistema operativo Android

2.3.1 Aplicaciones

Cada una de las aplicaciones desarrolladas sobre la plataforma Android, incluirán como base un cliente de email (correo electrónico), calendario, gestor de SMS's y una serie de API's. Todas estas aplicaciones comparten un *framework* común y están escritas en lenguaje JAVA.

2.3.2 Esqueleto de una aplicación

Dado que Android es un sistema de código libre, todos los desarrolladores tienen acceso al código fuente de las aplicaciones básicas del sistema, Google ha dispuesto esto de esta forma para evitar el desarrollo de multitud de componentes que respondan a los mismos eventos o acciones.

Esto proporciona a los programadores la posibilidad de que estos componentes sean modificados o usados de forma completamente libre por estos, permitiendo a estos llevar a cabo sus desarrollos de forma mucho más rápida y sencilla, ya que no tienen por qué iniciar sus aplicaciones desde cero en todos los casos.

2.3.3 Librerías

Android nos proporciona de forma nativa una serie de librerías que podemos usar de forma sencilla desde cada una de nuestras aplicaciones para integrar gran cantidad de funcionalidades a nuestras aplicaciones.

Entre estas librerías, podemos encontrar librerías *SSL*[7] para el soporte de certificados y seguridad, *SQLite* que nos permite gestionar el acceso a la base de datos local, *OpenGL 3D* para la integración de gráficos en tres dimensiones en la aplicación, *WebKit* para la optimización de la navegación de páginas web, el *Media Framework* para gestionar la gran cantidad de archivos multimedia que permite reproducir el sistema de forma nativa, y el *Surface Manager*, que nos permite gestionar las diferentes pantallas en interfaces que podemos integrar en las aplicaciones, entre otras muchas.

[7] Secure Socket Layer, protocolos criptográficos estándar que se encargan de proporcionar comunicaciones seguras a través de una red.

2.3.4 Runtime de Android

Android incluye en su sistema un conjunto de librerías básicas que proporcionan al sistema la mayoría de funcionalidades básicas disponible en el núcleo del lenguaje de programación Java.

Como se ha mencionado con anterioridad en este documento, cada aplicación Android ejecuta su propio proceso, con su propia instancia de la máquina virtual Dalvik, la cual ha sido desarrollada de forma que un dispositivo sea capaz de ejecutar varias máquinas virtuales simultáneamente de forma eficiente. Para conseguir esta eficiencia, la máquina virtual inicia archivos ejecutables, con la extensión .Dex, cuyo formato esta optimizado para un consumo de memoria mínimo.

La máquina virtual está basada en registros, y ejecuta una serie de clases compiladas por un compilador Java que se han convertido al formato .Dex mediante el uso de la herramienta incluida "dx". Además, esta máquina virtual se basa en el kernel de Linux para ofrecer las funcionalidades subyacentes, como podrían ser la gestión de memoria o el *threading* [8]a bajo nivel.

[8] Subproceso creado por el sistema operativo destinado a dividir procesos mayores en pequeñas partes que faciliten su ejecución.

2.4 Máquina virtual Dalvik

La máquina virtual Dalvik, es una máquina virtual intérprete que ejecuta archivos en el formato .dex, *Dalvik Executable*, el cual como ya se ha hecho referencia anteriormente, esta optimizado para el almacenamiento eficiente y ejecución mapeable en memoria.

El objetivo primordial de esta máquina virtual, es el mismo que el de la gran mayoría de estas, permitir que el código fuente sea compilado a un *bytecode* de forma independiente al dispositivo en el que se vaya a ejecutar finalmente, encargándose ella misma de interpretarlo y ejecutar el programa. El motivo principal por el cual se optó por esta máquina virtual antes que la de Java, era la extrema necesidad de optimizar recursos y enfocar el funcionamiento de los programas hacia dispositivos en los que la memoria, el procesador y el almacenamiento son escasos.

Otra característica importante de la máquina Dalvik, es el hecho de haber sido optimizada para que múltiples instancias de ella puedan funcionar de forma simultánea con un bajo impacto en el rendimiento de la memoria del dispositivo. El objetivo de este diseño, es proteger las aplicaciones, de forma que el cierre forzado de alguna de ellas no afecte al funcionamiento del resto de aplicaciones.

La diferencia básica entre la máquina virtual Dalvik y la máquina Java tradicional, es que esta última se basa en el uso de pilas, y la Dalvik usa registros, ya que los teléfonos móviles están optimizados para la ejecución basada en los mismos.

A pesar de que el lenguaje usado para el desarrollo de aplicaciones para Android es Java, el *bytecode* de Java no es ejecutable bajo Android, de igual forma que las librerías de java usadas por Android, que son ligeramente diferentes a las usadas por el *Java Standard* (JAVA SE), guardando eso sí ciertas características en común.

El uso de la máquina virtual Dalvik, permite reducir considerablemente el tamaño de las aplicaciones, lo que es importante dado que el almacenamiento en este tipo de dispositivos es escaso, para conseguir esto lo que hace es buscar información repetida en diversas clases y reutilizarla.

El conocido como "*Garbage Collector*" de Java, usado para limpiar objetos que ya no se utilizan en nuestros programas del espacio de memoria, ha sido perfeccionado en Android, para mantener siempre el máximo espacio posible en la memoria, ayudando así a la optimización del sistema.

Para llevar más aún esta optimización Android usa de forma intensiva el lenguaje XML[9] para la generación de los ficheros que definen las interfaces gráficas, además de algunos otros datos, esto implica que estos archivos deben ser referenciados a lo hora de ejecutar la compilación, permitiendo así que su conversión a *bytecode* pueda mejorar el rendimiento de nuestras aplicaciones.

[9] XML, siglas en ingles de *eXtensible Markup Language* ("lenguaje de marca extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

2.5 Partes de una aplicación

En este apartado se definen y explican los componentes básicos que forman cada una de las aplicaciones desarrolladas para la plataforma Android. No todas las aplicaciones requieren todos estos componentes para ser funcionales, pero si la combinación de algunos de ellos.

En caso de querer usar alguno de estos componentes, deberemos especificarlo en el archivo AndroidManifest.xml, el cual especificaremos más adelante en el documento.

2.5.1 Activity

Este componente representa una única pantalla de la aplicación y se encarga de relacionarla directamente con una interfaz con la cual el usuario podrá interactuar. De entre las diversas actividades que pueden formar una aplicación, una de ellas se marcará como actividad principal o "*Main*", esta será la actividad encargada de iniciar la aplicación cuando el usuario la lance desde el menú del dispositivo.

De igual manera una actividad puede iniciar otras actividades con el objetivo de componer la aplicación de diversas pantallas o interfaces con las que poder interactuar con el usuario, ya sea mostrándole datos que la aplicación ha generado, obteniendo datos que el usuario nos facilite mediante su interacción con esta, etc.

Seguidamente se detallará el funcionamiento de un componente de tipo *activity*, ya que es el tipo de componente más importante que encontramos en una aplicación y el único utilizado para implementar los videojuegos del proyecto, junto con los *intents*, que nos sirven para "lanzar" nuevas actividades.

Toda aplicación debe tener, al menos, una actividad. Se trata de la actividad *Main*, la cual se lanzará cuando el usuario indique que quiere ejecutar la aplicación en cuestión.

Para crear un componente del tipo actividad, deberemos crear una clase en Java que herede de la clase *Activity*. Y esta clase deberá definir una interface que la actividad presentará al usuario. Android da una serie de herramientas para definir estas interfaces utilizando el lenguaje XML, lo que nos permite separar, en la medida de lo posible, la parte visual de la actividad de la parte lógica. Entendiendo como parte lógica la parte que, ante una entrada del usuario, hace una serie de operaciones para darle una respuesta. Esta lógica se implementa en lenguaje Java.

La definición de una interface para una actividad en Android se lleva a cabo mediante la construcción de un árbol de elementos gráficos llamados *views* (ver Imagen 2). Las *views* pueden estar prefabricadas o desarrolladas por el propio desarrollador de la aplicación, según las necesidades de este.

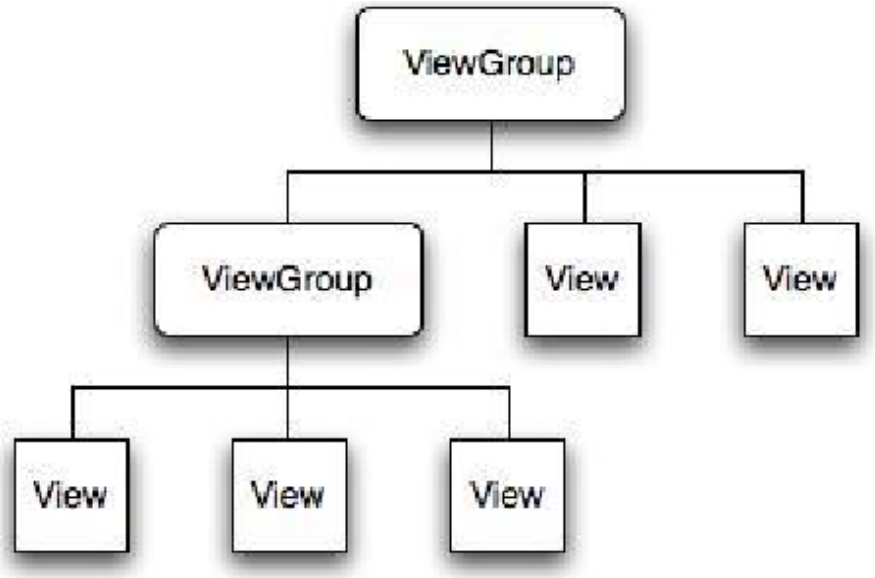

Imagen 2: Esquema de un árbol de views. Son este tipo de estructuras las que definen la interfaz de una actividad.

Una *view* es una estructura de datos que mantiene parámetros y contenidos para gestionar la interacción del usuario con un rectángulo específico. Este rectángulo es un fragmento del área de la pantalla ocupado por la actividad cuya interfaz está compuesta por la *view* en cuestión.

Es decir, a la hora de visualizar una actividad, esta se mostrará en un área específica de la pantalla, más grande o más pequeña según si la estamos visualizando a pantalla completa o no. La construcción de una interfaz consiste en dividir dicha área en rectángulos, cada uno gestionado a través de una *view* diferente.

Existen dos tipos de *views:* los *ViewGroup,* que son *views* que dentro pueden tener otras *views* de cualquier tipo y dotan a estas *views* interiores de una serie de propiedades concretas; y los *widgets,* que son *views* raíz, las cuales llevan a cabo algún tipo de interacción con el usuario, independiente del resto de *views,* dentro del rectángulo que se les asigna.

Ejemplos de *ViewGroup* son las *layouts,* unas *views* que condicionan la forma en que el área ocupada por el *ViewGroup* se reparte entre sus *views* internas. La repartición puede ser dividiendo el área horizontalmente, verticalmente, etc. dependiendo del tipo de *layout* escogido.

En cambio, ejemplos de *widgets* son: el botón, la caja para rellenar con texto, la lista de texto, entre otras. La mayoría de *widgets* necesarios ya vienen prefabricados en la API de Android y ya traen implementada la interacción con el usuario. Por tanto, en caso de utilizarse un *widget* prefabricado, el desarrollador únicamente deberá leer los datos que el *widget* recoge del usuario (pulsado de un botón, texto insertado, etc.) y llevar a cabo con ellos el propósito que desee.

Eso sí, para leer estos datos, en lugar de llevar a cabo una comprobación constante, normalmente se deben definir una serie de operaciones que, correctamente configuradas, son llamadas por el S.O. cuando una de las *views* tiene nueva información que presentar. Nuevamente vemos como el funcionamiento de los componentes de una aplicación en Android es asíncrono y es el S.O. el que nos avisa de los sucesos cuando ocurren. En concreto, en una actividad, con el mecanismo asíncrono evitando bloquear el *thread* donde se está ejecutando dicha actividad con bucles largos, lo cual provocaría que no se pudiera llevar a cabo una interacción con el usuario.

Ciclo de vida de una actividad

Como ya hemos dicho, una actividad en Android funciona de forma asíncrona y el *thread* donde se ejecuta esta actividad debe quedar libre por tal de que los eventos que activa el usuario se reciban con rapidez.

Por este motivo, en la clase *Activity* se declaran una serie de operaciones, las cuales deberemos sobre-escribir en la actividad que creemos, pues a través de ellas llevaremos a cabo las acciones que queremos que dicha actividad haga. Estos métodos son llamados por el S.O. cuando se dan una serie de sucesos concretos, son las operaciones a través de las cuales gestionamos el ciclo de vida de la actividad (ver Imagen 3).

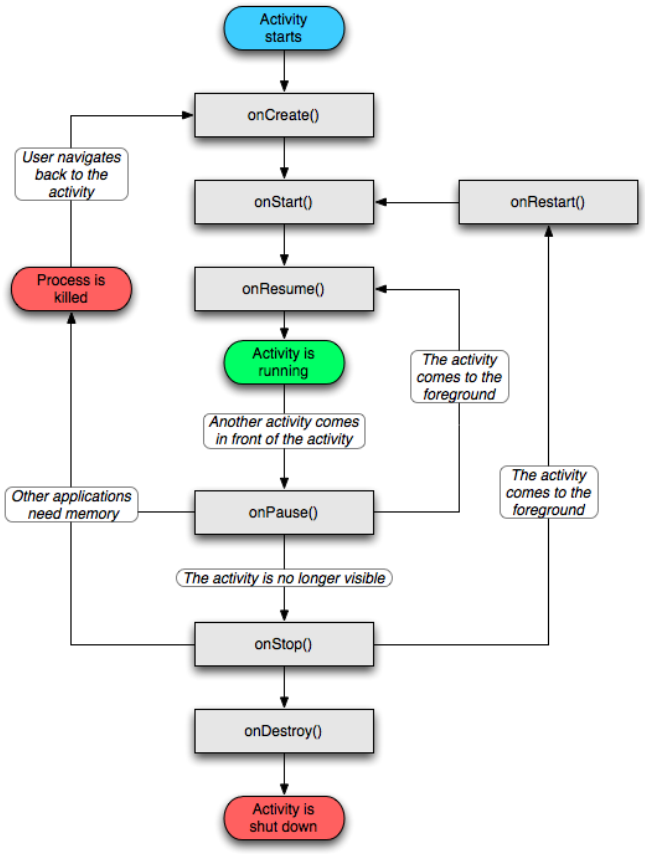

Imagen 3: Esquema del ciclo de vida de un componente del tipo *Activity*, de una aplicación del sistema operativo Android.

A continuación se describen las operaciones a través de las cuales se gestiona el ciclo de vida de la actividad:

- *onCreate():* Operación que llama el sistema operativo a la hora de crear la instancia de la actividad.

  Esta operación es muy importante, pues es donde tenemos que introducir la jerarquía de *views* que define la interfaz de la actividad. Esto se hace mediante la llamada a *setContentView(),* independientemente de si cargamos dicha jerarquía de un fichero XML (lo más aconsejable) o de si la definimos en el propio código Java. Además, también deberemos inicializar los datos y llevar a cabo las operaciones cuyos resultados se mantengan estáticos durante toda la ejecución de la actividad.

- *onStart():* Esta operación será llamada por el S.O. cuando la actividad vaya a pasar a ser visible para el usuario, antes de hacerlo.

  En ella deberemos llevar a cabo los preparativos para una visualización concreta de la actividad. Preparativos que se deberán hacer cada vez que se visualice la actividad de nuevo, pues en caso contrario se habrían llevado a cabo en *onCreate()*.

- *onResume():* Esta operación la llama el S.O. justo después de que la actividad pase a ser visible para el usuario y hasta que no se acabe su ejecución la interacción con el usuario no podrá llevarse a cabo. Se suele utilizar para activar animaciones o efectos que deben empezar justo cuando el usuario ya está viendo la actividad.

- *onPause():* Otra de las operaciones más importantes en el ciclo de vida de una actividad. Esta operación es llamada por el S.O. cuando la actividad va a pausarse, pues otra actividad va a pasar a ejecutarse en primer plano.

  Se trata de la última operación que el S.O. Android nos asegura que será llamada. Y es que, una vez ejecutada esta operación, por falta de memoria principal el S.O. puede matar la actividad antes o después de ejecutar *onStop()*. Por este motivo, en *onPause()* deberemos hacer persistentes los datos que aún no hayamos guardado.

  Además, en esta operación también se suelen pausar animaciones u operaciones que se estén ejecutando en otro *thread* y consuman mucha CPU, pues cuando la actividad se pausa probablemente estas operaciones ya no es necesario que se estén ejecutando.

  En cualquier caso, las tareas de *onPause()* se deben llevar a cabo rápidamente, pues hasta que no acabe su ejecución la actividad no podrá dejar de ser visible y, por tanto, la nueva actividad que vaya a ser presentada no podrá visualizarse.

- *onStop():* El S.O. llama a esta operación cuando la actividad acaba de ser retirada de la vista del usuario. Aquí se suelen llevar a cabo acciones que liberan los recursos del dispositivo y que mientras la actividad estaba visible no se podían llevar a cabo, ya sea porque habrían tardado mucho en realizarse o porque el usuario las habría visto.

  El S.O. puede matar la actividad antes, durante o después de esta operación si necesita liberar memoria principal.

- *onRestart():* Esta operación se llama cuando la actividad, que permanece invisible, va a volver a visualizarse. En ella se pueden llevar a cabo preparativos que únicamente sea necesario realizar cuando la actividad pasa a ser visible después de una pausa anterior. Ver Imagen 3 para más detalles acerca de las operaciones llamadas posteriormente a esta.

  Una vez llamada la operación *onRestart(),* la actividad nuevamente no podrá ser eliminada por el S.O. hasta después de que se vuelva a llamar a *onPause(),* pues el S.O. da la prioridad más alta a la actividad visible en cada momento.

- *onDestroy():* Esta operación será llamada por el S.O. cuando la instancia de la actividad en cuestión vaya a ser eliminada, antes de llevar a cabo la eliminación. En ella se deben realizar las acciones de liberación de recursos necesarias.

  Esta operación solo será llamada si la actividad va a ser eliminada por indicación explicita del desarrollador de la aplicación, a causa de que el usuario haya presionado el botón *Back* o debido a una necesidad de recursos leve por parte del S.O. Pero si, por el contrario, el S.O. necesita memoria rápidamente, este eliminará el proceso entero donde se esté ejecutando la actividad sin llamar a *onDestroy(),* aunque tampoco hará falta, pues al eliminar el proceso se liberarán todos los recursos automáticamente.

Y hasta aquí las operaciones a implementar para gestionar el ciclo de vida de una actividad. Nótese que no todas tienen por qué redefinirse en la subclase de *Activity* que creemos, solo las operaciones en las que nos sea necesario llevar a cabo algunas acciones. Para el resto de operaciones dejaremos el comportamiento por defecto implementado en la superclase *Activity*.

2.5.2 Services

Un servicio es una tarea que el sistema ejecuta en *background*[10], y que no tiene ninguna interfaz de usuario asociada, dado que el usuario en ningún momento llegará a interactuar de forma directa con este. Este componente se encarga de realizar operaciones de larga duración de forma que el usuario pueda seguir usando la

[10] Proceso o rutina de ejecución que se ejecuta en segundo plano, normalmente con una prioridad baja con el fin de no saturar el sistema operativo.

aplicación mientras se generan los resultados esperados, o de realizar las tareas relacionadas con procesos remotos.

Algunos ejemplos de servicios podrían ser el reproductor multimedia o la escritura de ficheros en la memoria del dispositivo. Otro componente de la aplicación, como por ejemplo una actividad, se encargará de iniciar el servicio y dejarlo en *background* para poder interactuar con él y obtener los resultados en cuanto estos estén disponibles.

2.5.3 Broadcast Recievers

Este componente recibe y reacciona con otros eventos de tipo *broadcast* que puede producir la propia aplicación o el sistema en si, como podría ser el aviso de batería baja, SMS, etc.

2.5.4 Intents

Este componente es el encargado de activar los componentes mencionados hasta ahora, mediante este tipo de mensajes asíncronos. Este componente nos permite enviar mensajes a todo el sistema, a un *activity* o a un servicio concreto indicando en este la acción que se quiere llevar a cabo. El propio sistema se encargará de decidir quien realizará la acción que se ha solicitado en el *intent*.

2.5.5 Content Providers

Este componente nos permitirá acceder a los datos que queremos usar en nuestra aplicación desde las diferentes fuentes de datos que nos proporciona Android. Puede almacenar los datos en el sistema de ficheros del sistema, en la base de datos SQLite de la aplicación, una web o cualquiera de los otros lugares de almacenamiento definidos anteriormente.

A través de este tipo de componente podemos consultar y modificar los datos, en caso de que el *provider* usado lo permita.

2.5.6 Layout

Uno de los puntos más importantes en el diseño de aplicaciones para dispositivos móviles es la interfaz de usuario, ya que será aquello que llame la atención de los potenciales compradores y lo que en definitiva hará que estos se decidan a usar nuestra aplicación.

Actualmente el diseño de interfaces gráficas es el punto más débil de Android, ya que no dispone de una entorno de diseño completo, sino que es una pequeña parte del plugin que integramos en Eclipse, debido a esta incidencia es necesario conocer el lenguaje XML para poder diseñar interfaces atractivas.

Las interfaces en Android se definen sobre *layout's*, que son ficheros XML en los que definimos la posición, la forma, y una serie de propiedades de cada uno de los elementos gráficos que formarán parte de nuestra interfaz, mediante una serie de *tags xml*, que un compilador se encarga de traducir en posiciones sobre la pantalla.

Como hemos comentado con anterioridad una misma aplicación puede tener más de un *layout*, de hecho, lo más habitual es que tengan más de uno, ya que cada *activity* tiene su propia interfaz, y por lo tanto su propio *layout*.

2.5.7 AndroidManifest.xml

Cada aplicación Android que desarrollemos debe tener un archivo AndroidManifest.xml, exactamente con este nombre, en su directorio raíz. Este fichero presenta información esencial sobre la aplicación al sistema, información que el sistema es imprescindible que tenga para poder ejecutar la aplicación. Algunas de las cosas que el *manifest* se encarga de aportar son las siguientes:

- Indica el *package* de la aplicación, el cual se usa como identificador único de esta dentro del sistema.
- Definir cada uno de los componentes de la aplicación, las activities, servicios, broadcast recievers y content providers de los cuales se compone la aplicación. Además nombra las clases que implementan cada uno de estos componentes y publica sus propiedades. Estas declaraciones son usadas por el sistema para saber las capacidades de los componentes y en qué momento deben ser iniciados.
- Determina que procesos se encargarán de acoger a los componentes de la aplicación.
- Indica que permisos debe tener la aplicación de cara a acceder a partes protegidas de la API e interactuar con otras aplicaciones.
- A su vez indica los permisos que deben tener otras aplicaciones para poder interactuar con los componentes que forman nuestra aplicación.
- Lista las clases que se encargan de proporcionar los perfiles y demás información de la ejecución de la aplicación. Estas declaraciones únicamente están presentes en el AndroidManifest.xml mientras la aplicación está siendo desarrollada y probada, y son retirados cuando la aplicación se publica.
- Se declara el nivel mínimo de la API que el dispositivo debe ser capaz de ejecutar para poder usar la aplicación.
- Lista las librerías con las que se debe vincular la aplicación para poder funcionar correctamente.

2.6 Ciclo de vida de una aplicación

Las aplicaciones Android, no tienen control de su ciclo de vida, ya que es el propio sistema el que se encarga de gestionar este para poder tomar la decisión que más convenga al rendimiento del sistema, y por tanto del dispositivo en que se ejecute, en todo momento.

Esto quiere decir que el ciclo de vida de una aplicación Android lo maneja el sistema basándose en las necesidades del usuario, los recursos disponibles, las acciones solicitadas, etc. En caso de tener una aplicación en funcionamiento que este consumiendo una gran cantidad de recursos del sistema, y decidimos arrancar una segunda aplicación, el sistema se encarga de comunicar a la primera aplicación, que ahora queda en segundo plano, que liberé todos los recursos que le sea posible, y en caso de ser necesario puede llegar a forzar su cierre.

En Android los recursos del sistema son normalmente limitados y es por este motivo por el que el sistema tiene más control sobre todas sus aplicaciones que cualquier otro sistema de escritorio habitual.

Como hemos comentado con anterioridad, en la mayoría de los casos, cada aplicación lanza su propio proceso de Linux. Este proceso se crea para la aplicación cuando la iniciamos y seguirá ejecutándose hasta que no sea necesario y el sistema solicite los recursos que está usando para otras aplicaciones.

Para tomar todas están decisiones relacionadas con el ciclo de vida de las aplicaciones, Android ordena los procesos por importancia, tal y como se detalla a continuación:

2.6.1 Foreground process

Es el proceso que contiene la actividad que se está mostrando en ese momento por pantalla, para lo que se ha invocado el método onResume(). Habitualmente el número de *Foreground process* que se ejecutan de forma simultánea en el sistema es muy limitado, y estos procesos serán finalizados únicamente si aun liberando todos los procesos posibles del sistema, la memoria de este sigue siendo escasa.

2.6.2 Visible process

Es un proceso que contiene una actividad que es visible, pero que no está en primer plano, se ha creado debido a la ejecución del método onPause(). En ejemplo podría ser el estar leyendo un email y pulsar sobre una *url* de forma que el navegador web se inicia, pasando en ese momento la aplicación del mail de *Foreground Process* a *Visible Process* y siendo el navegador el nuevo *Foreground Process*. Este tipo de procesos tienen un nivel de prioridad alto para el sistema, por lo que este trata de evitar su cierre en la medida de lo posible.

2.6.3 Service process

Este servicio es similar al que podría ejecutar cualquier distribución de Linux. Este tipo de proceso se encarga de hacer tareas en segundo plano que normalmente tienen una cierta importancia, por lo que el sistema nunca lo finalizará a no ser que sea imprescindible para poder mantener en ejecución todos los procesos *Foreground* y *Visible*. Este estado si inicia mediante la ejecución del método startService().

2.6.4 Background process

Es un proceso que contiene una actividad que actualmente no es visible para el usuario, y al que se llega después de la ejecución del método onStop(). Este tipo de proceso no tiene una importancia demasiado elevada, ya que puede ser alguna aplicación que se inició tiempo atrás y no se ha vuelto a usar, y que por lo tanto quedo en *background*. Es por este motivo por el cual es importante liberar todos los recursos que nos sea posible cuando nuestra aplicación pase a *background*, evitando así que el sistema decida cerrarla porque hace un uso indebido de los recursos de este.

2.6.5 Empty process

Es un proceso que no contiene nada, y que es usado por el sistema como caché en el momento en el que se crea un nuevo proceso. Es común la eliminación de este tipo de procesos por parte del sistema con el objetivo de liberar memoria para la ejecución de procesos con mayor prioridad.

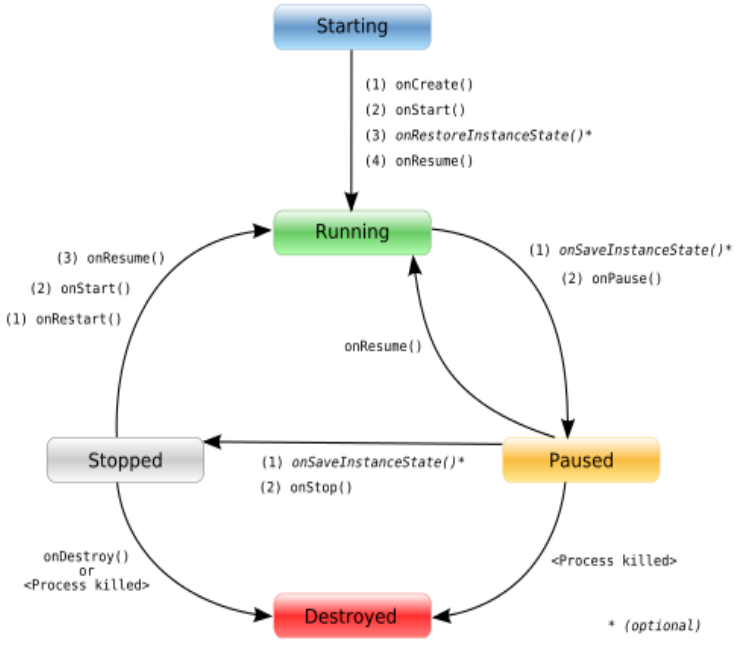

Imagen 4: Esquema del ciclo de vida de una aplicación Android

2.7 Entorno de desarrollo de Android

A continuación se va a describir el entorno de desarrollo que Android pone a disposición de los desarrolladores. Hay que destacar que una de las ventajas de este entorno es que se puede instalar en Windows, en Linux y en Mac O.S. X.

Lo único que necesitamos para desarrollar una aplicación es el SDK (*Software Development Kit*) de Android. Un kit que incluye una serie de herramientas como son: el *Traceview*, herramienta que nos permite extraer estadísticas de rendimiento de la aplicación; el *LogCat*, que nos presenta los mensajes que se imprimen desde el código durante la ejecución de una aplicación; y herramientas para generar los instaladores de las aplicaciones que desarrollemos; entre otras herramientas de utilidad.

Además, el SDK incluye la API (*Application Programming Interface*) de Android, unas librerías que contienen todas las clases y operaciones que debemos utilizar para poder comunicarnos con el S.O. y de esta forma poder, por ejemplo, definir una actividad.

En cuanto al IDE (*Integrated Development Environment*) o entorno que integra las herramientas necesarias para desarrollar aplicaciones, podemos utilizar cualquiera. Incluso podríamos utilizar un editor de textos normal para programar la aplicación y no utilizar un IDE. Aun así, se recomienda encarecidamente el uso de Eclipse. Y es que existe un *plugin* oficial llamado ADT (*Android Development Tools*) que una vez instalado en Eclipse nos permite acceder a la API y a las diferentes herramientas que el SDK nos blinda directamente desde la interfaz de Eclipse, facilitando enormemente el desarrollo de las aplicaciones.

Si utilizamos Eclipse, por tanto, podemos visualizar el *LogCat*, controlar el *Traceview* o generar el fichero .apk (el instalador) resultante de la aplicación, desde la propia interfaz de Eclipse.

Por último, en cuanto al lenguaje de programación, la API se encuentra en Java y el lenguaje recomendado para programar es este. Java es un lenguaje menos eficiente que C y sus variantes, por lo tanto no se suele utilizar para desarrollar videojuegos. Pero también tiene una gran ventaja y es que, como se ha comentado con anterioridad, es un lenguaje interpretado, lo que significa que se interpreta a través de una máquina virtual en tiempo real.

De esta forma podemos evitar tener que compilar nuestra aplicación para diferentes arquitecturas, pues con una sola compilación, la aplicación funcionará en todos los dispositivos sin problemas.

Sin embargo, el uso de C sigue siendo imprescindible para desarrollar con éxito algunas partes de un videojuego complejo, como la gestión de físicas. Así que a través de un paquete llamado NDK (*Native Development Kit*) podemos utilizar funciones programadas en C, que se ejecutarán de forma nativa durante la ejecución de la

aplicación. Claro está que, al ser C un lenguaje no interpretado, deberemos compilar este código para cada una de las arquitecturas donde la aplicación de Android vaya a ejecutarse. Para facilitar las cosas, Android nos permite introducir en el mismo instalador de la aplicación varias compilaciones diferentes de las operaciones en C, por tal de que el mismo instalador funcione en varias arquitecturas al mismo tiempo.

Nótese que cada nueva versión de Android que sale a la luz amplía el número de acciones que podemos llevar a cabo utilizando únicamente el lenguaje C. Se espera que en un futuro se puedan llegar a desarrollar aplicaciones únicamente haciendo uso de este lenguaje.

Pero en este proyecto se han desarrollado las aplicaciones haciendo uso únicamente del lenguaje Java. Aunque, para obtener la eficiencia que aporta una gestión de las físicas del videojuego en lenguaje C, se ha utilizado una librería de físicas desarrollada en este lenguaje. Eso sí, dicha librería esta oculta por un mapeo entre sus operaciones y el lenguaje Java, con lo cual, aunque realmente haya una parte del videojuego funcionando en C, desde el punto de vista del desarrollador solo se ha trabajado en Java.

2.8 Análisis motores de juego 2D

Existen muchas alternativas de motores 2D. Un motor no es más que un *framework* con funciones que facilitan acciones de todo tipo, desde la representación de *sprites*, hasta la colisión entre entidades físicas, pasando por la carga de mapeados o la gestión de efectos sonoros.

2.8.1 AndEngine

Imagen 5: Logo AndEngine

El primero que barajé fue el propio **Andengine,** que finalmente terminé utilizando, pero también presentaba algunos problemas. Como ventajas, tenía que era un motor con muchos ejemplos de los que extraer información, con soporte constante por su creador, actualizaciones frecuentes, y en general, un motor serio, diseñado específicamente para android, que además ha sido utilizado para la realización de juegos comerciales en la Play Store. Sin embargo, a causa de la potencia de sus funciones, el motor está bastante encapsulado, utiliza a su vez elementos de Box2D, que es a su vez otro motor, y esa abstracción presenta algunos problemas de rendimiento si se compara con otros motores más "cercanos"al dispositivo. Además, carece de documentación, por lo que no existe una fuente en la que detallen las funciones existentes y sus comportamientos.

Más información en: http://www.andengine.org/

2.8.2 Box2D

Imagen 6: Logo Box2D

Box2D es un motor de uso bastante extendido en PC, tiene todas las funciones necesarias para colisiones entre distintas formas geométricas, uso de la gravedad, y un entorno amigable que facilita el desarrollo. Sin embargo, su versión Android está bastante más limitada. Tampoco existe documentación, no tiene una base de ejemplos tan rica como *AndEngine* ni un soporte tan constante. Existen distintos ports de la versión de PC a Android, todos con poca utilidad práctica.

Más información en: http://code.google.com/p/androidbox2d/
http://www.box2d.org/

2.8.3 Cocos2D

Imagen 7: Logo Cocos2D

Cocos2D es una muy interesante opción para iOS, pero en Android existen exclusivamente algunos ports que simplemente sustituyen el código de Objetive-C por Java para hacerlo compatible. Además, todo el soporte (bastante abundante) está orientado a la versión iOS del motor, por lo que fue descartado rápidamente como opción.

Más información en: http://code.google.com/p/cocos2d-android/

2.8.4 Angle

Imagen 8: Logo Angle

Angle es un proyecto dedicado a ofrecer un motor en 2D, basado en Open GL ES, que ofrezca un buen rendimiento para plataformas Android. Es uno de los más interesantes tras *AndEngine* porque ofrece una serie de ejemplos y tutoriales bastante interesantes, que ofrecen un acercamiento al uso del motor, ya que al igual que el resto, carece de documentación.

Más información en: http://code.google.com/p/angle/

2.8.5 Jpct-ae

Imagen 9: Logo Jpct-ae

Jpct-ae es un port de JPCT para Android, soporta 3D, pero de nuevo carece de documentación o de actualizaciones frecuentes. Existen vídeos en internet que enseñan distintas demostraciones sobre las capacidades del motor.

Más información en: http://www.jpct.net/jpct-ae/

2.8.6 Libgdx

Imagen 10: Logo Libgdx

Libgdx es un motor 2D/3D con mucha actividad, que está siendo constantemente actualizado y revisado por sus creadores, y que dice ofrecer un rendimiento muy superior a la mayoría de motores existentes para Android debido a una gran optimización. Ha sido usado para juegos comerciales en la *Play Store*, y a diferencia del resto de motores, incluye una rica API en la que se documenta el *framework* al completo, ofreciendo explicaciones sobre las clases y funciones incluidas. Tiene una WIKI con material y tutoriales para empezar a familiarizarse, y dispone de una rica base de ejemplos y proyectos con el código completamente disponible para aprender en base a ellos. Supone una opción perfectamente recomendable para iniciarse en el desarrollo de juegos para Android, pero finalmente me decanté por *Andengine*.

Más información en: http://code.google.com/p/libgdx/
http://libgdx.badlogicgames.com

2.9 API AndEngine

Una vez decidida la temática del proyecto, el primer paso a llevar a cabo era recabar toda la información posible sobre el desarrollo de videojuegos para la plataforma Android, después de un exhaustivo proceso de búsqueda, llegamos a la conclusión de que había dos forma de hacerlo, o bien usando las API's nativas de google o bien usando algunas API's externas.

Llevar a cabo el desarrollo mediante el uso de las API's nativas de Android, implicaba un gran número de horas de trabajo e investigación, sin tener la certeza de obtener un resultado final aceptable, dado que su complejidad es elevada y la documentación al respeto escasa.

El uso de API's externas facilitaba la tarea, ya que la documentación era mucho más extensa y el grado de complejidad de estas se reducía de forma significativa. De forma que se decide hacer uso de alguna API orientada al diseño de videojuegos.

Era una requisito indispensable que la API elegida fuera *OpenSource*, por lo que después de una ardua tarea de investigación la escogida fue la librería AndEngine, ya que cumplía con todos los requisitos previamente establecidos, nos proporcionaba gran cantidad de funcionalidades relacionadas directamente con el desarrollo de videojuegos, y sobre todo es de código libre.

En el momento de llevar a cabo el desarrollo de videojuegos, hay una serie de procesos que pueden llegar a ser realmente complejos de desarrollar, como podrían ser la carga de imágenes o el motor de físicas, y es en estos puntos en donde *AndEngine*, resulta extremadamente funcional.

En los siguientes apartados detallamos las principales funcionalidades que nos ofrece esta librería.

2.9.1 Actividad principal

Es parte básica del desarrollo de cualquier juego, una actividad que se encarga de gestionar todo el ciclo de vida del juego, sistema de puntuación, temporización en caso de ser necesaria, avance de los mapas, etc., en resumen todo aquello que hace que la experiencia de juego del usuario goce de cierta fluidez.

Es en este punto en el que usamos la llamada *BaseGameActivity* del *AndEngine*, que contiene las clases principales del flujo de juego. Para poder controlar este flujo, debemos crear una actividad que derive de *BaseGameActivity* y extender la implementación de sus métodos principales para adaptarlos a nuestras necesidades.

Siendo algunos de los eventos más típicos que deberemos controlar, el arranque del juego, la pantalla de inicio, la selección de nivel, personajes y demás configuraciones, en caso de que estás puedan llegar a existir, pausar el juego, etc.

2.9.2 Motor

El motor (*Engine*) es la pieza encargada de hacer que el juego pueda llegar a funcionar. Dispone de un hilo de ejecución que refresca la pantalla, es decir aquello que el usuario está viendo, cada intervalo de ciertos milisegundos, previamente definidos por el programador del juego. En cada *tick* se encarga de sincronizar los refrescos de la pantalla, con la nueva situación de todos los elementos que forma parte de ella, y en definitiva de actualizar la escena.

EngineOptions se encargan de definir en qué orientación poner la pantalla, cargar la cámara y elegir si la visualización ocupará toda la pantalla o no del dispositivo.

IresolutionPolicy es una implementación que es parte de *EngineOptions*. Le dice a *AndEngine* cómo operar con los diferentes tamaños de pantalla de los diferentes dispositivos. Una de las funciones es *RatioResolucionPolicy*, que maximiza la vista de la superficie a las limitaciones de la pantalla, mientras mantiene un ratio específico. Eso quiere decir que los objetos no se distorsionarán mientras se tenga el tamaño máximo posible.

2.9.3 Escenas

La escena es el contenedor de todos y cada uno de los *sprites* que se deben visualizar por pantalla, y para esto *AndEngine* nos facilita una clase, con una serie de métodos base que debemos completar para adaptarlos a las necesidades de nuestro juego.

Las escenas pueden estar formadas por diferentes capas y entidades, en las cuales podemos diferenciar entre mapas, sistemas de puntuación e información al jugador, objetos con los que interactuar, personajes del juegos, etc. Hay subclases, como la CameraScene/HUD/MenuScene que se dibujan ellas solas en la misma posición de la *Scene* sin importar dónde esté posicionada la cámara.

2.9.4 Cámara

Una cámara define el rectángulo de la escena que es mostrado en la pantalla, ya que toda la escena nunca es visible todo el tiempo. Normalmente hay una cámara por escena, excepto por las *SplitScreenEngines*. Hay subclases que permiten hacer zoom y suavizar cambios de posición de la cámara.

2.9.5 Entidad

Una entidad es un objeto que puede ser dibujado, como los *Sprites* (gráficos), *Rectangles* (rectángulos), *Text* (texto) o *Lines* (líneas). Una entidad tiene posición rotación, escala, color, etc.

2.9.6 Texturas

Las texturas son imágenes en memoria. Estas imágenes serán utilizadas para visualizar *sprites* o fondos con los que componer la escena de nuestro juego. Es por esto que en el momento en el que se inicia el desarrollo de un juego, por una parte se gestiona la información lógica de los objetos y por otra parte el aspecto visual que finalmente tendrá, o lo que es lo mismo, las texturas que mostraremos.

El *BitmapTextureAtlas* se puede intentar ver como un lienzo, donde se irán poniendo los *TextureRegion* que se vayan creando. Al ser la base de todas las imágenes debe tener un tamaño considerable que sea potencia de dos.

Una *TextureRegion* (región de textura) define un rectángulo en la textura. Una *TextureRegion* es usada por los *Sprites* para que el sistema sepa qué parte de la textura grande está mostrando el *Sprite*.

2.9.7 Sprites

Los *sprites* son objetos que se visualizarán en el juego y en la mayoría de los casos serán interactivos o incluso animados. Existen diversos tipos de *sprites*, tiles en el que las imágenes tienen forma de matriz, animados en los que el *sprite* se compone de varias imágenes o fotogramas, que se irán intercambiando a lo largo de la acción del juego para simular un movimiento o animación, o *background* los que serán usados para dibujar fondos.

2.9.8 Físicas y colisiones

Adicionalmente disponemos de las funcionalidades necesarias para detectar colisiones e incluso aplicarles algunos efectos de física, mediante los cuales simulamos movimientos provocados por la interacción de diferentes elementos del juego, como serían el no poder atravesar una pared, o detectar que una bala ha impactado en un enemigo, provocando así nuestro victoria, consiguiendo así comportamientos mucho más reales de los objetos durante la acción del juego.

Para poder llevar a cabo toda esta interacción entre objetos, que como hemos comentado anteriormente tiene una complejidad de desarrollo bastante elevada, se usa una extensión basada en el motor de físicas de código libre Box2D.

2.9.9 Música y efectos de sonido

Otra característica de este *framework* es la posibilidad de reproducir algunos tipos de ficheros de sonido. Permitiéndonos controlar ciertas características de la reproducción como serían subir y bajar el volumen, repetir el sonido, saber si está siendo reproducido en un momento concreto, escoger un punto concreto de la pista de audio, entre otras funciones adicionales.

3. Videojuego I: Vestir princesas

Estamos en la parte de la memoria donde se describen en profundidad los diferentes productos construidos, así como las decisiones que se han tomado para su desarrollo, la explicación de cómo se han adoptado los conocimientos adquiridos hasta ahora y los aspectos conceptuales directamente relacionados con funcionalidades de estos.

En primer lugar se ha desarrollado un videojuego con una mecánica simple, he aquí toda la información sobre este.

3.1 Descripción del videojuego

"Vestir princesas" está basado en el típico juego de niños (mayoritariamente niñas) que está tan de moda en el mundo de los videojuegos de móviles, ya que se trata de vestir a una muñeca a nuestro gusto, juego al que todos hemos jugado en nuestra infancia con muñecos de verdad. Este producto responde al tipo de videojuego que tiene éxito entre los niños, ya que se trata de un juego con una mecánica muy simple y que no requiere de mucha atención para el aprendizaje por parte del jugador.

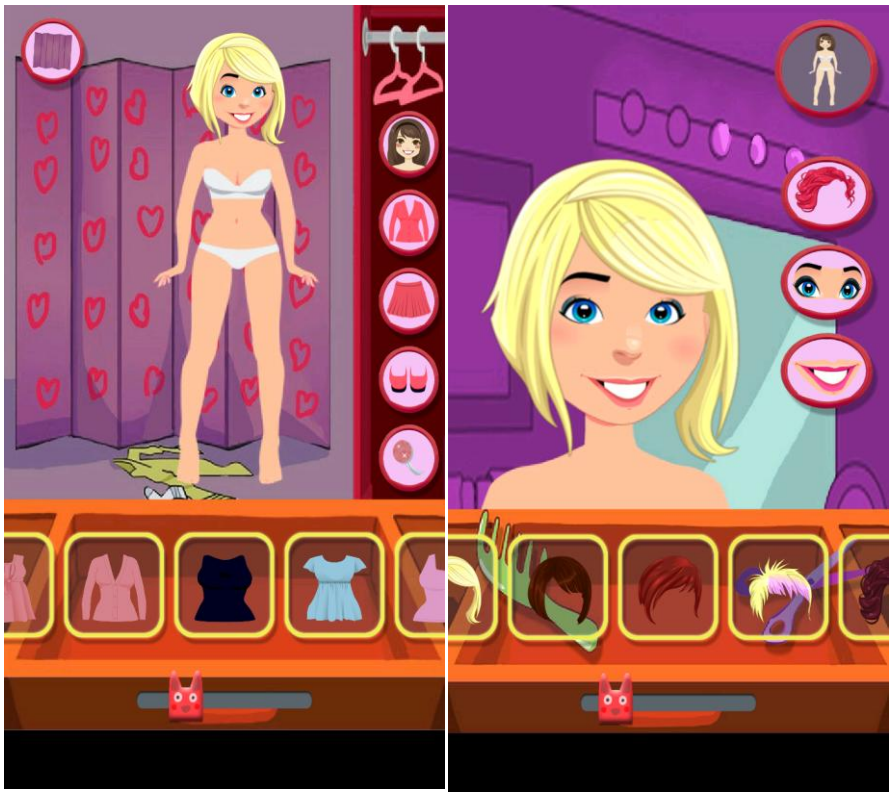

Imagen 11: Capturas de pantalla de la interfaz del juego "Vestir princesas" donde aparecen las dos pantallas principales del juego.

Elementos del juego

El juego consta de varios elementos principales (cada pantalla) que interaccionan creando la experiencia de juego:

La muñeca: se trata del personaje que nosotros tenemos que vestir a nuestro gusto, arrastrando las prendas de ropa a la zona en la que van puestas (los pantalones a las piernas, los zapatos a los pies, etc.).

La cabeza: se trata de la cabeza del personaje que nosotros tenemos que modificar a nuestro gusto, arrastrando los ojos que queramos a la zona de los ojos, y de igual forma con la boca y el pelo.

Las prendas de vestir: en la parte inferior de la pantalla se mostrará la lista de objetos que existen para vestir a nuestro personaje. La lista de objetos a mostrar variará según el botón de tipo de prenda (situados en la parte derecha de la pantalla) que pulsemos.

Las partes de la cara: al igual que en las prendas de vestir, en la parte inferior de la pantalla se mostrará la lista de partes de la cara que existen para modificar la cara de nuestro personaje. La lista de objetos a mostrar variará según el botón de parte de la cara (situados en la parte derecha de la pantalla) que hayamos pulsado para modificar.

Botones de prendas: en la parte derecha de la pantalla se encuentran los botones de los diferentes tipos de prendas que tenemos para vestir a nuestro personaje. Según el botón que pulsemos se mostrarán unos objetos u otros en la parte inferior de la pantalla. También tenemos un botón para cambiar de pantalla e ir a la pantalla de modificar las partes de la cara.

Botones de partes de la cara: al igual que en la pantalla para modificar la ropa de nuestro personaje, los botones se encuentran en la parte derecha de la pantalla, y cada uno representa una parte de la cara a modificar (pelo, ojos y boca). Según el botón que pulsemos se mostrará la lista de partes de la cara correspondiente en la parte inferior de la pantalla.

Barra de scroll: debajo de donde se muestran las prendas o partes de la cara tenemos una barra de *scroll*, esta funciona desplazando el muñeco a lo largo de la barra, para así poder ver todas las prendas o partes de la cara listadas.

Mecánica del videojuego

La meta del juego es vestir a la muñeca y cambiarle la cara para hacerla a nuestro gusto, disponiendo de diferentes prendas de ropa al igual que diferentes colores y formas para las partes de la cabeza.

Es preciso aclarar que el juego no tiene final, de forma que el jugador puede vestir y cambiar las partes de la muñeca las veces que quiera.

Manejo por parte del usuario

El control de la interfaz se realiza a través de la pantalla táctil. El usuario debe arrastras la prenda que quiere ponerle a la muñeca a su zona correspondiente en el cuerpo. Una vez la prenda entra en contacto con dicha zona, se colocará automáticamente en la posición correcta. Solo se puede arrastrar una prenda a la vez.

Para ver todas las prendas de las que se dispone para vestir a la muñeca podemos desplazar el *scroll* situado en la parte de debajo de donde están las prendas.

En cambio, si lo que se quiere es ver el resto de prendas de las diferentes partes del cuerpo (zapatos, pantalones, camisetas o complementos), lo que el usuario deberá hacer es pulsar el botón correspondiente situado en la parte derecha de la pantalla. En esta zona disponemos de un botón para cambiar a la pantalla de cambiar las partes de la cara de la muñeca. Esta pantalla funciona exactamente igual que la de vestir a la muñeca.

El videojuego se ha desarrollado con el objetivo de poderse reproducir en el máximo número de versiones de Android disponible, ya que es un juego que no requiere demasiada potencia y que, por tanto, debería funcionar correctamente en dispositivos poco potentes. Por tanto, se trata de un videojuego preparado para reproducirse en la versión 2.1 y superiores de Android.

Herramientas utilizadas

En cuanto a las herramientas utilizadas, la aplicación se ha desarrollado utilizando el lenguaje Java, el entorno de desarrollo que proporciona el *plugin* ADT para Eclipse y el motor de juego *AndEngine*.

La aplicación funciona en la versión 2.1 y superior de Android, y por este motivo se trata de un juego con prácticamente total compatibilidad de dispositivos Android del mercado, incluido las *tablets*.

3.2 Análisis y especificación de requisitos

A continuación se presenta la especificación del videojuego. La fase de especificación ha sido ajustada al tamaño del proyecto; como se trata de una aplicación sencilla y que además se debe hacer en un período de tiempo reducido no es viable hacer una especificación propia de un proyecto más complejo y largo.

3.2.1 Diagrama de casos de uso

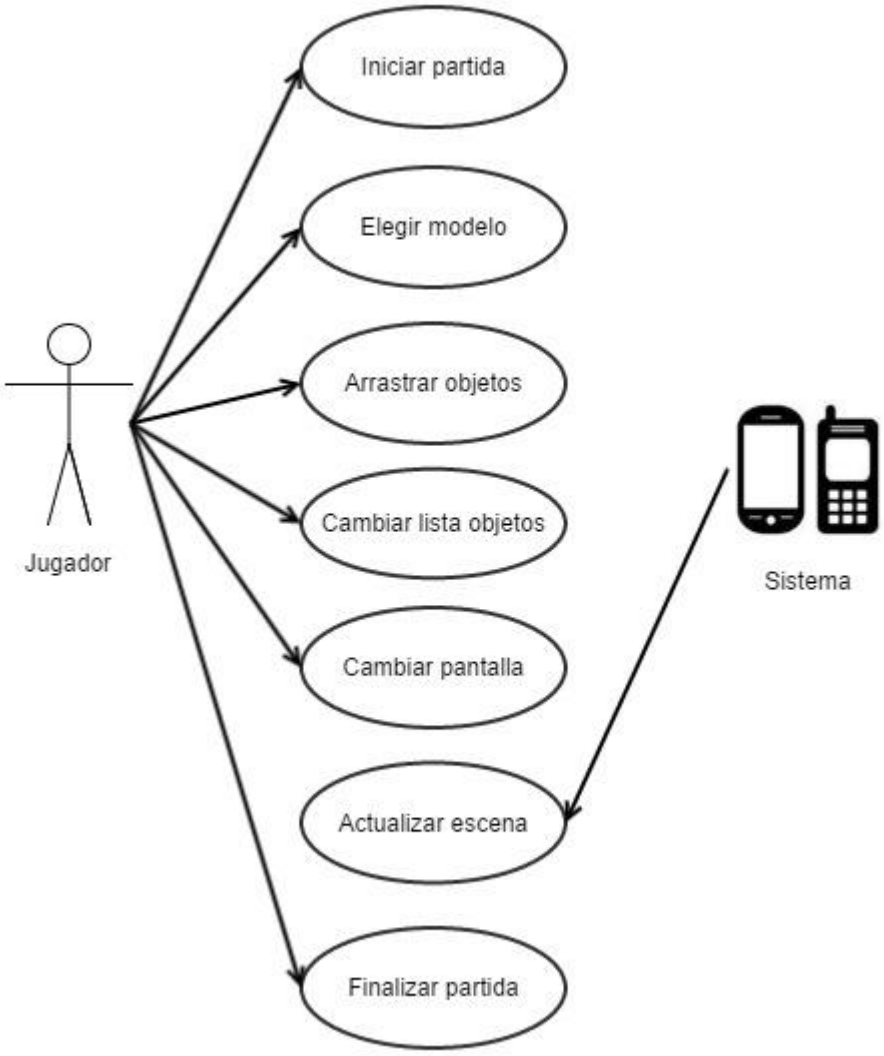

Imagen 12: Diagrama de casos de uso del videojuego "Vestir princesas"

3.2.2 Definición de los casos de uso

Caso de uso: Iniciar partida

Actor: Jugador
Descripción: El jugador podrá iniciar una partida en cualquier momento.
Escenario: El usuario inicia la aplicación y tras pulsar en cualquier lugar de la pantalla indica al sistema que quiere empezar una nueva partida.

Caso de uso: Elegir modelo

Actor: Jugador
Descripción: El jugador podrá elegir la modelo de princesa con la que jugar la partida.
Escenario: El usuario deberá pulsar sobre la modelo que quiera para empezar a jugar la partida, es decir, para poder vestirla o cambiarle las facciones de la cara.

Caso de uso: Arrastrar objetos

Actor: Jugador
Descripción: El jugador dispondrá de la posibilidad de arrastrar los diferentes objetos (ya sean prendas de ropa o partes de la cara, según la pantalla en la que se encuentre) para vestir o cambiar las facciones de la cara de la princesa.
Escenario: El jugador pulsa sobre un objeto, y sin soltar, deberá arrastrarlo por la pantalla hasta situarlo en la zona correspondiente para que este se coloque automáticamente. Una vez el objeto ha sido colocado, ya no se encontrará en la lista inferior hasta que no sea reemplazado en el cuerpo de la princesa por otro objeto de su misma categoría.

Caso de uso: Cambiar lista objetos

Actor: Jugador
Descripción: El jugador podrá pulsar en los diferentes botones situados en la parte derecha para cambiar la lista de objetos a mostrar y que se encuentran disponibles para ponerle a la princesa.
Escenario: El jugador podrá pulsar sobre un botón y la lista de objetos situados en la parte inferior de la pantalla cambiará según el botón pulsado. Si hubiera algún objeto suelto por la pantalla sin haber sido colocado correctamente en la princesa, este volverá a su posición en la lista sin ser colocado en la princesa.

Caso de uso: Cambiar pantalla

Actor: Jugador
Descripción: El jugador podrá ir cambiando de pantalla, entre la pantalla del vestidor para vestir a la princesa, y la pantalla del maquillador, para cambiar de peinado, ojos, y boca, a la princesa.
Escenario: El jugador podrá pulsar sobre un botón situado en la parte derecha de la pantalla para cambiar de pantalla. Podrá pasar del vestidor al maquillador guardándose el estado en el que se encuentra la princesa en cada pantalla. Los cambios realizados en la pantalla de maquillaje también se verán en la pantalla del vestidor.

Caso de uso: Actualizar escena

Actor: Sistema
Descripción: El jugador se encuentra jugando una partida y quiere que el sistema simule en tiempo real dicha partida.
Escenario: El sistema irá actualizando la escena cada cierto tiempo, es lo que se denomina actualización *frame* a *frame*. El "reloj" avisa al sistema de que ya ha pasado el tiempo que dura un *frame* desde la última vez que se ejecutó este caso de uso, así que el sistema actualiza los elementos del mundo teniendo en cuenta el tiempo que ha pasado desde la última actualización.

Caso de uso: Finalizar partida

Actor: Jugador
Descripción: Habiendo una partida en marcha, el jugador desea finalizar la partida.
Escenario: El jugador indica al sistema que quiere finalizar la partida en curso. El sistema cierra dicha partida sin guardar el estado de la princesa, pasando a permitir que el jugador elija otra modelo (Caso de uso "Elegir modelo") distinta para volver a jugar.

3.3 Diseño

En los siguientes apartados se pasa a describir los diferentes componentes que forman el videojuego, así como las clases que lo componen (diagrama de clases).

3.3.1 Componentes de la aplicación

Las aplicaciones para el S.O. Android se dividen en componentes y estos componentes se clasifican en varios tipos. En este apartado se van a mostrar los diferentes componentes que conforman el videojuego “Vestir princesas”. La siguiente imagen, aparte de mostrar los componentes, nos va a permitir entender fácilmente cómo interactúan entre ellos y como el usuario puede navegar por ellos.

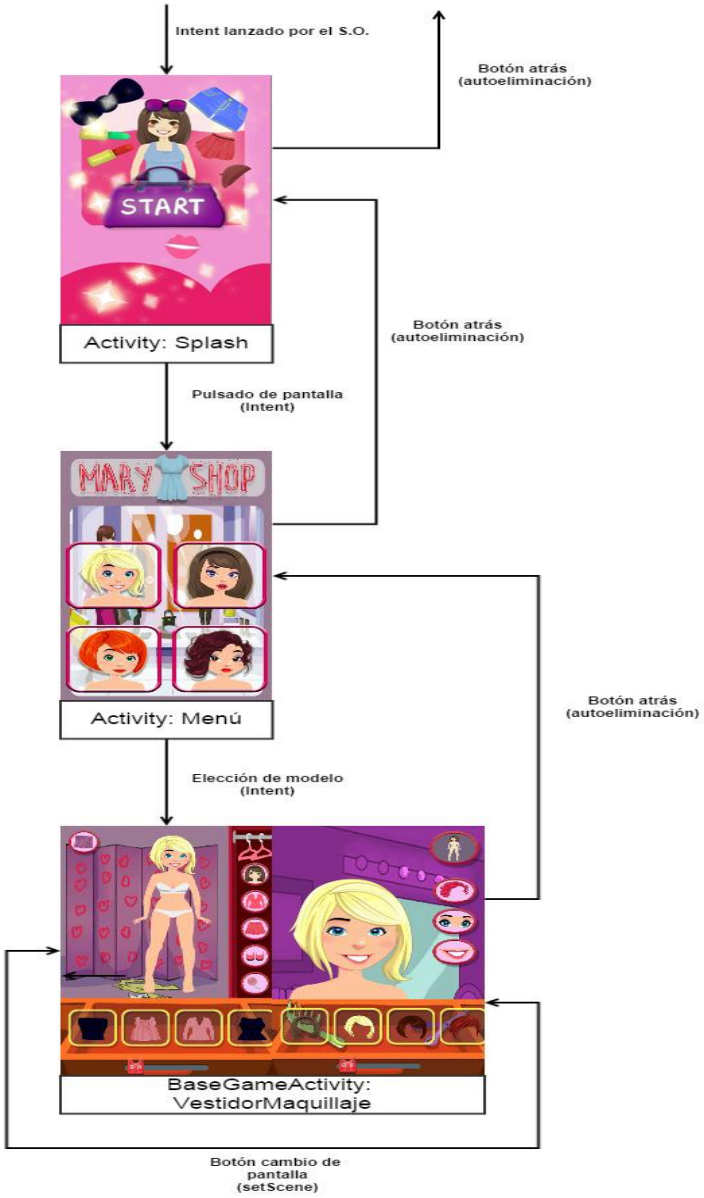

Imagen 13: Diagrama de componentes (o Mapa de Navegación) de la aplicación "Vestir Princesas".

En primer lugar, podemos apreciar como los tres componentes que conforman la aplicación son del tipo Actividad (*Activity)*, ya que el componente *BaseGameActivity* que es de la librería AndEngine, extiende de *BaseActivity*, que a su vez extiende de *Activity*. Como ya vimos en apartados anteriores, tenemos un componente de este tipo por cada pantalla de nuestra aplicación. Podemos decir, por tanto, que las actividades son el equivalente a las vistas de las aplicaciones tradicionales. Es por esto que este diagrama de componentes es equivalente al Diagrama de Navegabilidades de diseño, el cual nos muestra los diferentes caminos de navegación que un usuario puede tomar para moverse por la aplicación.

En el diagrama, cada una de las flechas que va de una actividad a otra, representa una navegación que puede ser efectuada durante la ejecución de la aplicación y contiene dos datos muy importantes:

- En primer lugar, el tipo de suceso que se deberá dar o el tipo de acción que el usuario deberá realizar, según el caso, sobre la actividad actual para ir de esta hasta la actividad a la que se dirige la flecha. Un ejemplo puede ser *Pulsado de pantalla,* que, como el propio nombre indica, viene a significar que para ir de la actividad actual a la siguiente, debemos tocar cualquier punto de la pantalla.

- En segundo lugar, entre paréntesis se encuentran las acciones que realizará la actividad en la que nos encontramos para que al usuario se le muestre la actividad destino, una vez que se ha dado el suceso o acción que se necesita para llevar a cabo el cambio de pantalla. Estas acciones pueden ser tres, aunque en este caso solo hemos gastado dos:

    - *Intent:* lo que significa que la actividad actual crea una nueva actividad, mediante el envió de un *intent,* y se almacena en la primera posición de la pila de actividades. Pasando a ser visible para el usuario la nueva actividad creada. El *intent* no es más que el mecanismo implementado en la API de Android para crear instancias de una actividad desde el código.

    - Auto Eliminación: en este caso, la instancia de la actividad que se está mostrando se elimina a sí misma. Esto da lugar a que la instancia de actividad que está arriba de todo de la pila de actividades vuelva, de nuevo, a ser visible para el usuario. La actividad que se encontraba a la cabeza de la pila era la actividad destino.

    - *Intent* + Auto Eliminación: en este caso se combinan las dos acciones explicadas con anterioridad. La actividad actual se auto elimina, pero no pasa a ser visible la actividad que se encuentra a la cabeza de la pila de actividades. En su lugar, una nueva instancia de actividad, creada por la actividad actual antes de auto eliminarse, pasa a ser visible para el usuario.

Una vez dada la explicación general acerca de cómo entender el Diagrama de Componentes, a continuación se va a describir el funcionamiento concreto de cada una de las actividades que podemos visualizar en el diagrama anterior:

- En primer lugar tenemos la actividad *Splash,* cuya única función es mostrar por pantalla una imagen, la cual al ser pulsada lanzará la actividad *Menu,* iniciando una nueva partida de nuestro juego. Cabe decir que es el Sistema Operativo Android el que envía el *intent* que crea la instancia de la Actividad *Splash* y lo hace cuando el usuario le indica que desea ejecutar la aplicación "Vestir Princesas" apretando sobre el icono de dicha aplicación. Si en este momento el usuario pulsa el botón *Back* la aplicación se cerrará.

- En segundo lugar tenemos la actividad *Menu*, cuya función es mostrar en pantalla las diferentes modelos iniciales que el usuario puede elegir para iniciar el modo de "edición" de la princesa. Cuando el usuario pulsa en una de las cuatro opciones, esta actividad lanza un nuevo *Intent* para iniciar la actividad de *VestidorMaquillaje*. Si estando en esta actividad el usuario pulsa el botón *Back*, esta actividad se auto eliminará y pasará a ser visible la actividad de *Splash*.

- Por último tenemos la actividad *VestidorMaquillaje* que se encarga de la ejecución de toda la partida: recibe la entrada del usuario (la elección de la modelo inicial de la princesa), se encarga del pintado de los gráficos que se actualizan en tiempo real, del manejo y control de los objetos existentes en las diferentes escenas de esta actividad, así como del manejo de cambio de escena (del vestidor al maquillador). En el momento en que el jugador pulsa el botón *Back* esta actividad se auto elimina, dejando en primer lugar de la pila a la actividad anterior, que en este caso sería el *Menu*.

Dicho de otra manera, decir que en cualquier actividad podemos apretar el botón *Back* del terminal para auto eliminarla y volver a la actividad que se encuentre en la parte superior de la pila de actividades de la aplicación.

3.3.2 Diagrama de clases

Continuamos con la fase de diseño de “Vestir Princesas”. Se ha decidido hacer directamente el diagrama de clases, en lugar de realizar primero el modelo conceptual de los datos, debido a la baja magnitud de estos videojuegos y a la necesidad de realizarlos de forma rápida para entrar en los plazos de entrega de la empresa.

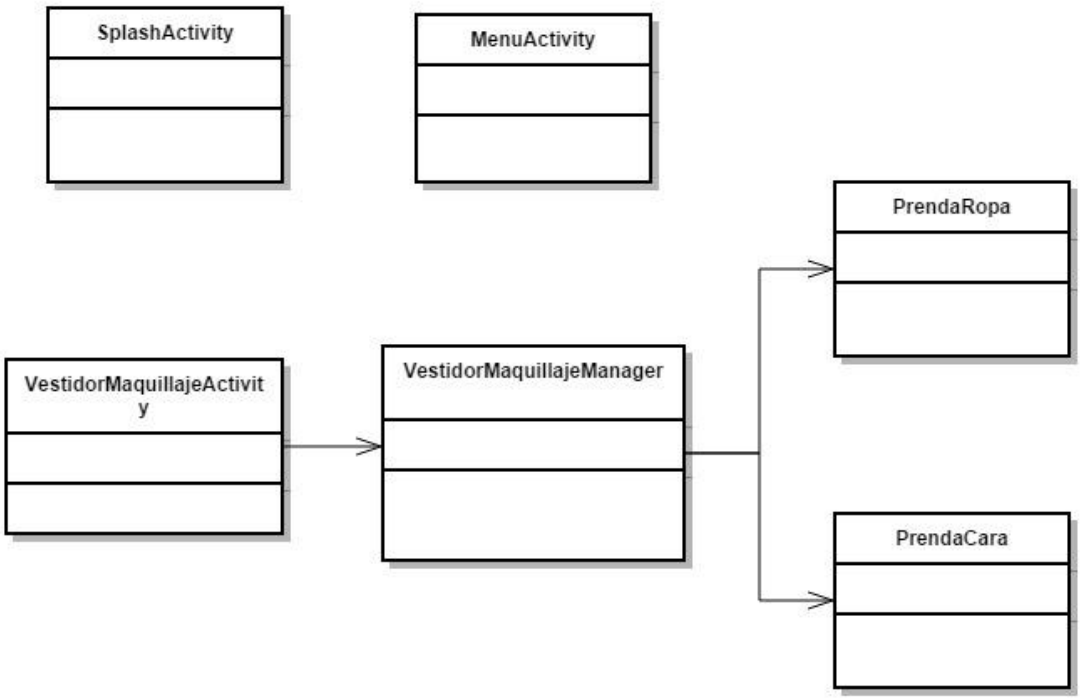

Imagen 14: Diagrama de clases del videojuego "Vestir Princesas"

A continuación se procederá a explicar las clases del diagrama:

- *SplashActivity*: la primera actividad que se lanza de la aplicación. En esta actividad (que hereda de *Activity*) únicamente se carga una imagen de fondo con un receptor de entrada, que en caso de activarse realiza una acción fija, como es lanzar la actividad de *MenuActivity*.

- *MenuActivity*: en esta actividad cargamos una imagen de fondo, y además cargamos cuatro imágenes que realizarán la función de botones, que al ser pulsados lanzan la siguiente actividad (*VestidorMaquillajeActivity*) pasándole la opción elegido a través del intent.

- *VestidorMaquillajeActivity*: esta actividad extiende de la clase *BaseGameActivity* del motor gráfico utilizado *AndEngine*. Está

actividad contiene un objeto de la clase *VestidorMaquillajeManager* que es la que gestiona todo lo relacionado con la pantalla de juego, tanto la carga de recursos, como el control de la interacción con el usuario, etc. Esta clase se ha creado para que mientras se cargan los recursos se pueda mostrar una pantalla de "cargando...", además de establecer el orden de carga de recursos y creación de escenarios (aunque la clase que se encarga de cargar los recursos y de crear los escenarios es la clase *VestidorMaquillajeManager*).

- *VestidorMaquillajeManager*: esta clase es la que contiene toda la lógica de la pantalla del juego, donde se puede vestir o cambiar las facciones de la cara de la princesa. Aquí es donde se realiza la carga de todos los recursos que se utilizarán en esta parte del juego, así como la creación de los escenarios y la implementación de la lógica para manejar los eventos lanzados por el usuario, tales como el pulsado de los botones, el desplazamiento de las prendas de ropa o partes de la cara, etc.

- *PrendaRopa*: esta clase extiende de la clase *Sprite* del motor gráfico *AndEngine*. Como su nombre indica, se utiliza para los objetos de las prendas de ropa de la muñeca, y en esta clase se guardan las posiciones del sprite en diferentes momentos, al igual que su estado, de si se encuentra en uso (esta puesto o siendo puesto en la princesa) o no. También se guarda el tipo de ropa que es, refiriéndonos a si es una prenda de la parte superior del cuerpo (camiseta, chaquea, etc.), inferior (falda, pantalón, etc.), zapatos o complementos.

- *PrendaCara*: esta clase es prácticamente idéntica a la clase *PrendaRopa* pero se utiliza para las partes de la cara, y también se guarda el tipo de parte de cara que es, ya sea pelo, ojos o boca.

3.4 Implementación

El propósito de este capítulo es describir al lector cómo se ha implementado la arquitectura y el diseño detallado en los apartados anteriores. Se realizará un estudio en profundidad de las funcionalidades implementadas dando al lector una visión más detallada sobre el funcionamiento interno.

Gestión de recursos

A continuación se va a detallar cual ha sido la solución tomada para la gestión de los recursos externos que utiliza la aplicación. Se explicará cual es el formato de fichero para cada tipo de archivo y la estructura del mismo.

El proyecto que se ha desarrollado utiliza varios tipos de ficheros externos, estos están almacenados en la carpeta "gfx" que está a su vez dentro de la carpeta "assets" del proyecto y está compuesta por la siguiente estructura:

- Carpeta "*imágenes":* Se almacenan las imágenes que utiliza el motor del juego. Únicamente las que utiliza el motor del juego, ya que el resto de imágenes se guardan en la estructura interna del proyecto y no se consideran archivos externos a la aplicación.

- Carpeta "*audio":* Lugar donde se albergan los ficheros de audio y los efectos sonoros del videojuego.

Al igual que ocurre con la carpeta externa al proyecto, el videojuego también utiliza la memoria interna del dispositivo para almacenar aquellos ficheros que se actualizan o modifican a lo largo de la ejecución del videojuego.

A continuación se van a describir cuales han sido los tipos de ficheros que se han utilizado para el desarrollo del proyecto:

- Imágenes y animaciones: Para las imágenes y animaciones se ha utilizado mayoritariamente el formato *PNG (Portable Network Graphics).* Se ha escogido este formato principalmente debido a que acepta transparencias y evita la pérdida de calidad de imagen. También se ha utilizado el formato JPG (*Joint Photographic Experts Group*) para las imágenes que no tienen transparencias ya que su tamaño es menor que en el formato PNG.

- Sonidos: El formato utilizado para los archivos de sonido ha sido *ogg*.

SplashActivity y MenuActivity

Android ofrece la posibilidad de crear las propias interfaces en lenguaje *Java* pero esto resulta más tedioso que crearlas en formato *xml*. De esta manera además conseguimos separar la representación gráfica del código de la aplicación desarrollada.

Los elementos son dibujados en la interfaz en el orden en el que aparecen en el fichero *xml*. Cada fichero se compila a partir del árbol de elementos que lo forman, y como consecuencia de esto, solo puede tener un elemento raíz.

En la clase *SplashActivity,* con su respectivo *layout* (*splash_layout),* solo tenemos un elemento del tipo *RelativeLayout,* con una imagen de fondo, para detectar cuando el usuario pulsa en la pantalla y así pasar a la siguiente actividad. Esta clase está creada como modo de presentación del juego.

En la siguiente imagen se muestra el menú de seleccionar modelo (*MenuActivity*, con el *layout* correspondiente *menu_layout*). En este menú aparecen elementos del tipo *ImageView*, que hacemos que tengan la función de un botón, así como elementos del tipo *LinearLayout* para distribuir las imágenes (botones) en las posiciones que queremos.

Imagen 15: Pantalla para seleccionar princesa (MenuActivity)

VestidorMaquillajeActivity y VestidorMaquillajeManager

En estas dos clases se encuentra implementada prácticamente la totalidad del juego, es decir, la pantalla de vestir a la princesa y la pantalla de cambiarle las partes de la cara, y en estas es donde entra en juego el motor gráfico *AndEngine*.

La actividad que se lanza es la de *VestidorMaquillajeActivity*, y esta clase posee un objeto de la clase *VestidorMaquillajeManager*, que es un gestor donde tenemos los métodos para cargar los recursos, para crear los escenarios, para manejar la lógica del juego, etc. Esto se ha hecho de esta manera para que en la actividad *VestidorMaquillajeActivity* se cargue y se muestre al usuario un fondo animado (para que el usuario sepa que el juego se está cargando) mientras el gestor se encarga de cargar todos los demás recursos.

La clase *VestidorMaquillajeActivity* extiende de la clase *BaseGameActivity* del motor gráfico *AndEngine*. Al extender de esta clase, hay que sobrescribir cuatro métodos obligatoriamente, que son los siguientes:

- public EngineOptions onCreateEngineOptions(): en este método es donde creamos y definimos la cámara y las opciones del motor del juego, como la orientación de la pantalla, la resolución de la pantalla, la interfaz para gestionar la adaptación a la resolución de la pantalla automáticamente, etc.

- public void onCreateResources (OnCreateResourcesCallback pOnCreateResourcesCallback): aquí es donde debemos cargar los recursos del juego. Este método se ejecuta automáticamente después del método *onCreateEngineOptions*. En este videojuego, aquí es donde se creará el objeto *VestidorMaquillajeManager*, y donde le diremos al gestor que cargue los recursos para la imagen de cargando el juego.

- public void onCreateScene (OnCreateSceneCallback pOnCreateSceneCallback): este es el siguiente método que se llama automáticamente una vez finalizada la carga de recursos. Aquí es donde se debe crear la escena del juego, añadir todos los elementos que componen la pantalla, gestionar la lógica del juego, etc. En este caso, llamaremos al método de *createCargandoScene* de la clase *VestidorMaquillajeManager* para que cree la escena de cargando recursos.

- public void onPopulateScene(Scene pScene, OnPopulateSceneCallback pOnPopulateSceneCallback): este método se ejecuta una vez ha finalizado el método onCreateScene. En este caso, en este método iremos llamando a los métodos del gestor VestidorMaquillajeManager encargados de cargar todos los recursos y de crear las escenas del

videojuego. Como se ha dicho antes, esto se ha hecho así para que mientras se cargan todos los recursos y se crean todas las escenas, el usuario vea una pantalla de cargando y no se crea que la aplicación se ha quedado bloqueada.

Como hemos dicho, la clase *VestidorMaquillajeManager* es la encargada de gestionar la mayor parte del juego. A continuación expongo los métodos que contiene esta clase:

- VestidorMaquillajeManager(BaseGameActivity activity, Engine engine, Camera camera): este es el constructor de la clase, en el que le pasamos el objeto *VestidorMaquillajeActivity (activity)*, el motor de juego creado (*engine*), y la cámara (*camera)*.

- loadCargandoSceneResources(): método donde cargamos los recursos para mostrar la imagen animada que se muestra mientras se cargan el resto de recursos.

- createCargandoScene(): método donde creamos la escena con la imagen animada de cargando los recursos y la mostramos al usuario

- loadVestidorSceneResources(): método donde se cargan todos los recursos necesarios para la escena del vestidor, donde podremos vestir a la princesa.

- createVestidorScene(int opcion): método donde creamos la escena del vestidor. Aquí es donde se crean todos los elementos de la escena y se le añaden. Al crear dichos elementos, ya sean los botones, las prendas de ropa, etc., sobrescribimos cuando haga falta algunos métodos de dichos elementos, como lo es el método onAreaTouched(TouchEvent pSceneTouchEvent, **float** pTouchAreaLocalX, **float** pTouchAreaLocalY), de esta forma gestionamos la interacción del usuario con dichos elementos. A este método le pasamos como parámetro la opción elegida por el usuario en el menú.

- loadMaquillajeSceneResources(): método donde se cargan todos los archivos necesarios para la escena de maquillaje de la princesa, donde podemos cambiar las partes de la cara de esta.

- createMaquillajeScene(int opcion): método donde creamos la escena del maquillador. La creación de dichos elementos, así como la gestión de la interacción del usuario, está creada de igual forma a la que hay en el método createVestidorScene(int opcion). El parámetro *opcion* también es la opción elegida por el usuario en el menú.

- ocultarPrendas(): método para ocultar todas las prendas de la escena del vestidor para después pasar a mostrar solamente las prendas que el usuario a elegido.

- ocultarPartesCara(): método para ocultar todas las partes de la cara de la escena del maquillador para pasar a mostrar las partes que el usuario a elegido.

- cambiarCaraVestidor(): método que se llama cuando se cambia de la escena del maquillador a la escena del vestidor. Este método sirve para actualizar la cara, es decir, para que los cambios producidos en la pantalla del maquillador se vean reflejados en la pantalla del vestidor.

- cargarCara(int opcion): método para cargar la cara inicial de la princesa, el parámetro *opcion* es la opción elegida por el usuario en la actividad *MenuActivity*.

- cargarCaraMaquillaje(int opcion): igual que el método anterior, pero para cargar la cara en la escena del maquillador.

- comprobarColisionConZona(final PrendaCara actual): método que se llama cada vez que se mueve una parte de la cara, en la escena del maquillador, para detectar cuando la parte de la cara que queremos colocar entra en colisión con la zona de la cara a la que pertenece, para así cambiar dicha parte de la cara.

- comprobarColisionConCuerpo(final PrendaRopa actual): igual que el método anterior, pero este en la escena del vestidor. Es decir, se ejecuta mientras se está moviendo alguna prenda de ropa, en la escena del vestidor, para así detectar cuando la prenda elegida entra en contacto con la zona del cuerpo a la que corresponde.

- setCurrentScene(SceneType scene): método para cambiar de escena. A este método se le pasa como parámetro el tipo de escena al que queremos cambiar, para así cambiar de la escena de “cargando”, a la escena del “vestidor” y a la de “maquillador”.

PrendaRopa y PrendaCara

Estas dos clases extienden de la clase *Sprite* del motor gráfico *AndEngine*. Estas clases se han creado para añadirle atributos nuevos a la clase, como lo son el tipo de prenda que es, o el tipo de parte de la cara que es. También tienen otro atributo para saber si la prenda o parte de la cara esta en uso, es decir, están puestas en la princesa o no.

La lógica implementada del juego hace que no podamos tener más de dos prendas sueltas por el escenario, así como que cuando ponemos un tipo de prenda que ya tenía puesta la princesa, la anterior se quita automáticamente. También existe la posibilidad de cambiar de fondo, de utilizar una barra de scroll para poder ver todas las prendas, etc. En resumen, con este videojuego queda demostrado que solo con sprites (los elementos de la escena), y manejando los eventos que lanza el usuario al interaccionar con el dispositivo, se puede realizar un juego entretenido para los niños.

4. Videojuego II: Explotar granos

En segundo lugar se ha desarrollado un videojuego con una mecánica muy simple pero de diferente tipo que el videojuego anterior. En este videojuego tendremos que ir pasándonos pantallas aumentando el nivel de dificultad del juego. A continuación toda la información sobre "Explotar granos".

4.1 Descripción del videojuego

"Explotar granos" está basado en los juegos de mecánica muy simple en los que hay que repetir una acción hasta conseguir la cantidad de puntos necesarios para superar el nivel de dificultad. Este tipo de juegos son adictivos, ya que se trata de partidas rápidas y además vas superando niveles y la dificultad del juego va en aumento. Debido a nuestra competitividad y nuestro afán por superarnos, todos quieren llegar hasta el final del juego, donde se encontrarán con un nivel diferente, el cual va por tiempo y tendremos que intentar conseguir la mayor cantidad de puntos posibles antes de que se acabe el tiempo. En este nivel se mostrará la mejor puntuación y la puntuación que llevas durante el juego, para que intentes mejorar dicha puntuación.

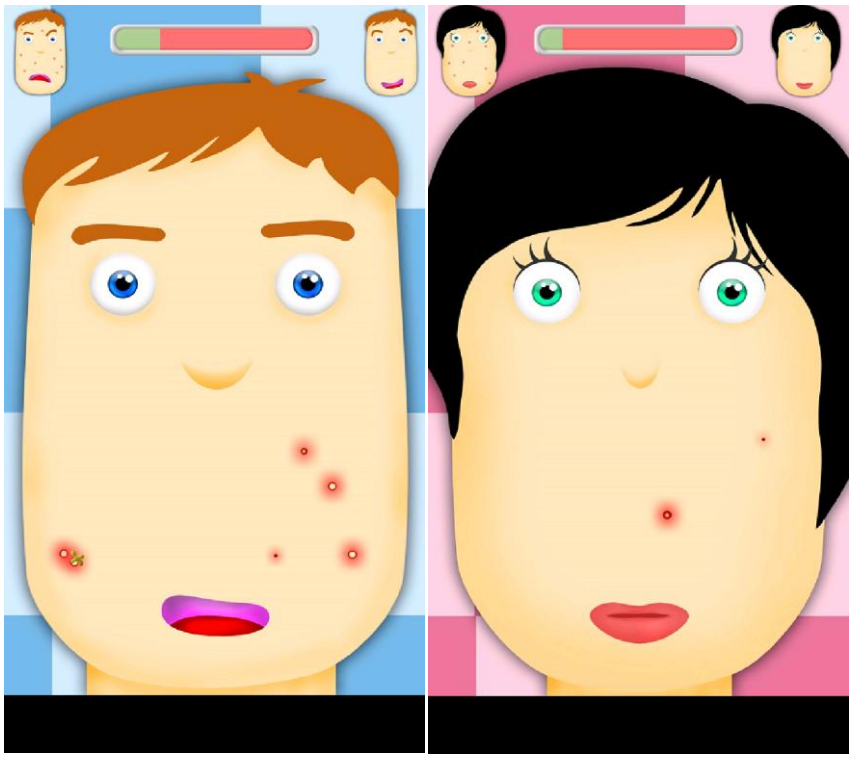

Imagen 16: Capturas de la pantalla principal del juego "Explotar granos", con la opción de chico o chica

Elementos del juego

El juego consta de varios elementos principales con los que hay que interaccionar para vivir la experiencia del juego:

La cabeza: se trata del personaje al cual le tenemos que ir explotando los granos, según la elección que hagamos en la pantalla inicial del juego, el personaje será un hombre o una mujer.

Los granos rojos: estos granos son los que salen en mayor cantidad en la cara de nuestro personaje. Deberemos ir explotándolos para conseguir puntos, si no los explotamos enseguida nos irán restando puntos.

Los granos verdes: estos granos son de diferente color, y cuando los explotamos ganamos una cantidad mayor de puntos que explotando los granos rojos. Pero también nos irán restando más puntos que los granos rojos si no los explotamos. Este tipo de granos aparecerán a partir del nivel 5.

Los besos: de vez en cuando, y a partir del nivel 7, nos aparecerán unos besos que nos da la pareja del personaje. Estos besos no hay que tocarlos, si los pulsamos, la pareja del personaje se enfadará y nos restará muchos puntos.

La barra de puntuación: En esta barra podemos ver el progreso de la partida. Cuando explotamos granos vamos ganando puntos y la barra verde va aumentando. Si la barra verde ocupa todo el espacio, significa que hemos ganado la partida y hemos superado el nivel. Si por el contrario la barra roja es la que ocupa todo el espacio, quiere decir que hemos perdido la partida.

Mecánica del videojuego

La meta del juego es explotar los granos que le aparecen a nuestro personaje hasta conseguir los puntos necesarios para superar el nivel. Si por el contrario perdemos muchos puntos, perderemos la partida.

Este juego dispone de niveles, empezaremos por el nivel 1 y tendremos que ir superándolos para pasar de nivel. En cada nivel irá aumentando la zona por la que salen granos, la velocidad a la que salen, irán apareciendo más elementos como besos y granos verdes, etc. Y como nivel final, hay un nivel extremo, en el cual disponemos de 45 segundos para lograr la mayor puntuación posible.

Manejo por parte del usuario

El control de la interfaz se realiza a través de la pantalla táctil. El usuario debe pulsar en los elementos (granos rojos, granos verdes, besos) que van apareciendo en la cara del personaje.

Herramientas utilizadas

Al igual que en el primer videojuego de “Vestir princesas”, la aplicación se ha desarrollado utilizando el lenguaje Java, el entorno de desarrollo que proporciona el *plugin* ADT para Eclipse y el motor de juego *AndEngine*.

La aplicación funciona en la versión 2.1 y superior de Android, y por este motivo se trata de un juego con prácticamente total compatibilidad de dispositivos Android del mercado, incluido las *tablets*.

4.2 Análisis y especificación de requisitos

A continuación se presenta la especificación del videojuego. Al igual que para el anterior videojuego, la fase de especificación ha sido ajustada al tamaño del proyecto; como se trata de una aplicación sencilla y que además se debe hacer en un período de tiempo reducido no es viable hacer una especificación propia de un proyecto más complejo y largo.

4.2.1 Diagrama de casos de uso

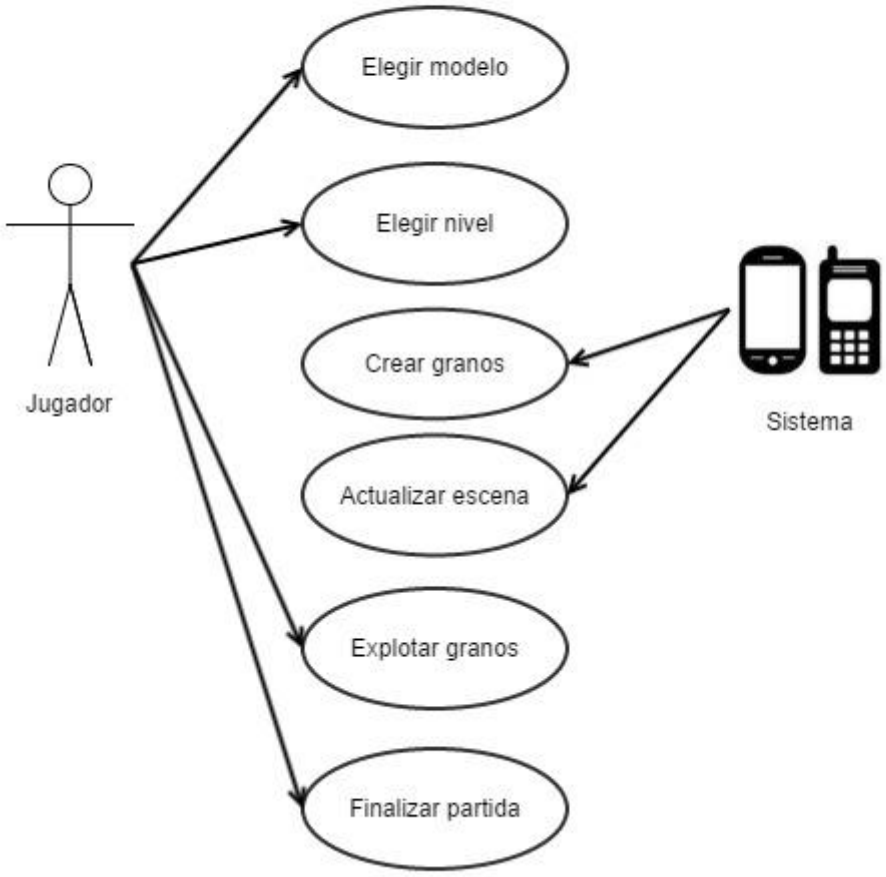

Imagen 17: Diagrama de casos de uso del videojuego "Explotar granos"

3.2.2 Definición de los casos de uso

Caso de uso: Elegir modelo

Actor: Jugador
Descripción: El jugador podrá iniciar una partida en cualquier momento.
Escenario: El usuario inicia la aplicación y tras pulsar en uno de los dos personajes le indica al sistema que quiere empezar una nueva partida con dicho personaje.

Caso de uso: Elegir nivel

Actor: Jugador
Descripción: El jugador podrá elegir el nivel al que quiere jugar.
Escenario: El usuario deberá elegir el nivel al que quiere jugar para iniciar la partida. Solo podrá jugar a los niveles que tenga abiertos. Para ir abriendo nuevos niveles tiene que superar los niveles anteriores.

Caso de uso: Crear granos

Actor: Sistema
Descripción: El sistema crea y añade nuevos elementos a la partida.
Escenario: El sistema irá creando cada cierto tiempo aleatorio nuevos granos y añadiéndolos al escenario para que el jugador pueda pulsarlos y explotarlos. Dependiendo del nivel, también podrá ir creando granos de diferentes tipos y besos.

Caso de uso: Actualizar escena

Actor: Sistema
Descripción: El jugador se encuentra jugando una partida y quiere que el sistema simule en tiempo real dicha partida.
Escenario: El sistema irá actualizando la escena cada cierto tiempo, es lo que se denomina actualización *frame* a *frame*. El "reloj" avisa al sistema de que ya ha pasado el tiempo que dura un *frame* desde la última vez que se ejecutó este caso de uso, así que el sistema actualiza los elementos del mundo teniendo en cuenta el tiempo que ha pasado desde la última actualización.

Caso de uso: Explotar granos

Actor: Jugador
Descripción: El jugador podrá pulsar en los diferentes elementos (granos y besos) que se van creando y añadiendo en la escena.
Escenario: El jugador podrá pulsar sobre un elemento de los que van apareciendo en pantalla, ya sea un grano rojo, verde, o un beso, y según el tipo de elemento que fuera, se añadirá o restará puntos al jugador, para así superar el nivel en el que está jugando.

Caso de uso: Finalizar partida

Actor: Jugador
Descripción: Habiendo una partida en marcha, el jugador desea finalizar la partida.
Escenario: El jugador indica al sistema que quiere finalizar la partida en curso. El sistema cierra dicha partida y permite al jugador volver a la pantalla de elegir modelo.

4.3 Diseño

En los siguientes apartados se pasa a describir los diferentes componentes que forman el videojuego, así como las clases que lo componen (diagrama de clases).

4.3.1 Componentes de la aplicación

Las aplicaciones para el S.O. Android se dividen en componentes y estos componentes se clasifican en varios tipos. En este apartado se van a mostrar los diferentes componentes que conforman el videojuego “Explotar granos”. La siguiente imagen, aparte de mostrar los componentes, nos va a permitir entender fácilmente cómo interactúan entre ellos y como el usuario puede navegar por ellos.

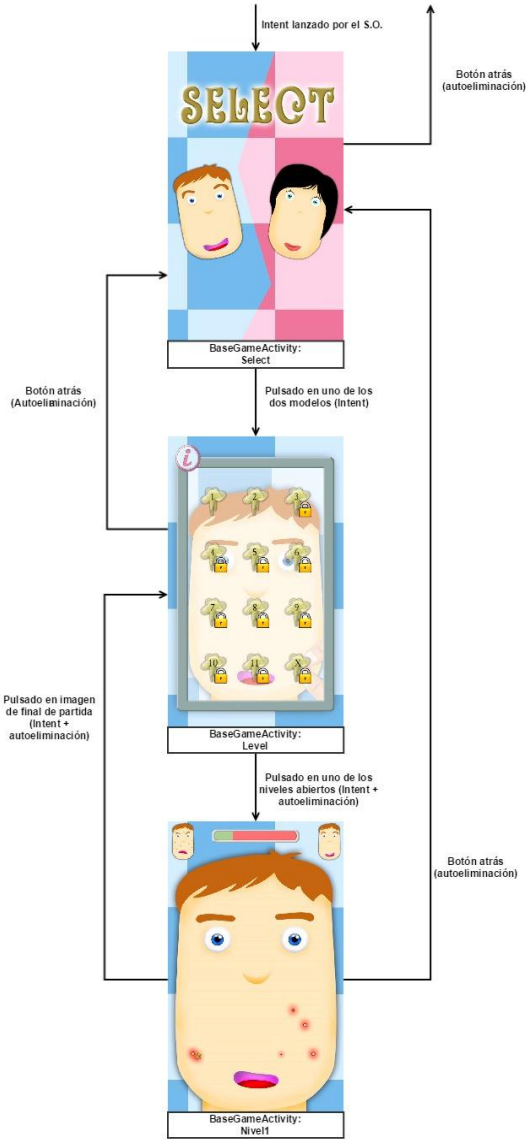

Imagen 18: Diagrama de componentes (o Mapa de Navegación) de la aplicación "Explotar granos".

Como podemos observar, los tres componentes principales de la aplicación son Actividades del tipo *BaseGameActivity*. Se ha optado por hacerlo de esta manera debido a la soltura que ya se tiene en el manejo de la librería AndEngine, y por una de sus grandes ventajas, como lo es el escalado automático que se hace de las pantallas según el tamaño de la pantalla del dispositivo móvil, por lo que en cualquier dispositivo que ejecutemos el videojuego la distribución de los elementos en cada pantalla será exactamente igual en todos.

Recordar para entender el diagrama anterior, que cada una de las flechas que va de una actividad a otra representa una navegación que puede ser efectuada durante la ejecución de la aplicación. En la información de estas flechas indicamos, en primer lugar, el tipo de suceso que se deberá dar o el tipo de acción que el usuario deberá realizar sobre la actividad actual para ir hasta la actividad a la que se dirige la flecha. Y en segundo lugar, entre paréntesis, las acciones que realizará la actividad en la que nos encontramos para que al usuario se le muestre la actividad destino. Estas acciones pueden ser tres: *intent,* auto eliminación, *intent + auto eliminación* (explicadas anteriormente).

Pasamos a describir el funcionamiento concreto de cada una de las actividades que podemos visualizar en el diagrama anterior:

- En primer lugar tenemos la actividad *Select,* cuya función es mostrar por pantalla las dos opciones de los personajes a elegir que tenemos para el videojuego. El usuario debe elegir una de las dos opciones, lo que dictará el personaje con el que jugaremos a continuación. Para ello, tiene que pulsar en el rostro de una de las dos opciones, y al pulsar sobre esta, se lanzará la actividad *Level*. Cabe decir que es el sistema operativo Android el que envía el *intent* que crea la instancia de la actividad *Select,* y lo hace cuando el usuario le indica que desea ejecutar la aplicación "Explotar granos". Si estando en la actividad *Select* el usuario pulsa el botón *Back,* la aplicación se cerrará.

- En segundo lugar tenemos la actividad *Level*, en la cual tenemos los diferentes niveles de dificultad para jugar. En la primera partida solo podremos pulsar sobre el botón de nivel1, ya que los demás niveles estarán bloqueados hasta que el usuario vaya superando los niveles anteriores. Cuando el usuario pulsa en uno de los niveles esta actividad lanza un nuevo *Intent* para iniciar la actividad *Nivel* (cada nivel tiene su propia clase, ya que cada nivel tiene sus propias características) y además esta actividad se autoeliminará. Si estando en la actividad *Level* el usuario pulsa el botón *back*, está actividad se auto eliminará dejando paso a la actividad anterior (*Select*).

- Por último tenemos la actividad *Nivelx* (donde x es el número del nivel de la actividad) que es la actividad principal del videojuego. En está es donde se centra la parte importante del juego, ya que es la pantalla de juego donde aparece la cara del personaje que hemos elegido en la pantalla de *Select* y donde tenemos que explotar los granos de nuestro

personaje. Si estando en está pantalla el usuario pulsa el botón *back*, se pasará a mostrar la siguiente actividad que hay en la pila de actividades, que no es otra que la de *Select*, ya que la actividad *Level* se auto elimina al lanzar el *intent* para pasar a la pantalla de *Nivel*. Otra opción es, cuando el usuario acaba la partida, ya sea con victoria o derrota, se muestra una imagen con el resultado (*Win* o *Lose*) y el usuario puede pulsar sobre esta imagen y se lanza un nuevo *intent* para lanzar la actividad de *Level*, y la actividad de *Nivel* se auto elimina. Esto se ha implementado así debido a los problemas para actualizar la pantalla de *Level* para mostrar desbloqueados los niveles que se ha pasado el usuario. De esta forma se consigue este objetivo sin ningún problema.

4.3.2 Diagrama de clases

Como siguiente paso se representa el diagrama de clases del videojuego “Explotar granos”. Debido a la baja magnitud de este videojuego, se trata de un diagrama de clases sencillo.

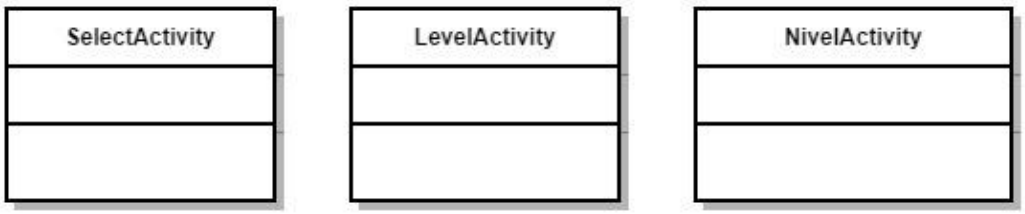

Imagen 19: Diagrama de clases del videojuego "Explotar granos"

A continuación se procederá a explicar las clases del diagrama:

- *SelectActivity*: la primera actividad que se lanza de la aplicación. En esta actividad (que hereda de *BaseGameActivity*, al igual que todas las clases de este videojuego) se carga una imagen de fondo y se añaden dos *Sprites* haciendo la función de botones, que al pulsar sobre uno de los dos se lanza la actividad *LevelActivity* pasándole como parámetro a través del *intent* la opción elegida por el usuario.

- *LevelActivity*: en esta actividad cargamos una imagen de fondo, y además cargamos los botones para que el usuario pueda seleccionar el nivel al que quiere jugar. Cuando el usuario pulsa sobre uno de estos botones para seleccionar el nivel, se lanza un *intent* con la actividad *Nivel* que corresponda al nivel seleccionado. Además también añadimos un botón de información para mostrar una imagen a modo de tutorial del juego.

- *NivelActivity*: esta clase es la que contiene la lógica de la pantalla principal del juego, donde tenemos que explotar los granos de nuestro personaje. Aquí es donde se muestra la cara del personaje y se van creando los granos y demás elementos y añadiéndolos al escenario, donde se gestiona la interacción del usuario pulsando sobre los granos, donde se controla la puntuación del usuario para ver si gana la partida y pasa de nivel, o por el contrario, pierde la partida.

4.4 Implementación

El propósito de este capítulo es describir al lector cómo se ha implementado la arquitectura y el diseño detallado en los apartados anteriores. Se realizará un estudio en profundidad de las funcionalidades implementadas dando al lector una visión más detallada sobre el funcionamiento interno.

Gestión de recursos

A continuación se va a detallar cual ha sido la solución tomada para la gestión de los recursos externos que utiliza la aplicación. El proyecto que se ha desarrollado utiliza varios tipos de ficheros externos, estos están almacenados en la carpeta "gfx" y en la carpeta "Font" que está situada a su vez dentro de la carpeta "assets" del proyecto. En esta carpeta se almacenan todas las imágenes, ya que todas las clases del juego están implementadas extendiendo la clase BaseGameActivity del motor gráfico *AndEngine*. Esta carpeta está compuesta por la siguiente estructura:

- Carpeta "*pantallaSelect*": aquí se almacenan todas las imágenes necesarias para la clase *SelectActivity*.
- Carpeta "*pantallaLevel*": en esta carpeta se almacenan todas las imágenes necesarias para la clase *LevelActivity*.
- Carpeta "pantallaJuego": aquí se almacenan todas las imágenes necesarias para la clase *NivelActivity*.
- Carpeta "*sonidos*": lugar donde se albergan los ficheros de audio y los efectos sonoros del videojuego.
- Carpeta "font": carpeta que contiene el archivo para la fuente utilizada en el texto del videojuego.

SelectActivity y LevelActivity

Estas, al igual que todas las clases de este videojuego, extienden de la clase *BaseGameActivity*. En ellas simplemente se ha implementado un menú con botones. En *SelectActivity* el menú consta de las dos caras de los personajes a elegir, el hombre o la mujer. A continuación, cuando pulsamos en una de las dos caras, pasamos a la pantalla de *LevelActivity* con el fondo ya del personaje elegido.

En *LevelActivity* crearemos un menú con los diferentes niveles a elegir para jugar. El juego empezará con solo el nivel 1 desbloqueado, y cuando se pase un nivel, se

desbloqueará el nivel siguiente. Este control lo llevamos guardando una variable en las preferencias de usuario (*SharedPreferences*) para saber qué nivel es el último que el usuario ha superado.

NivelActivity

Al final, debido a los tiempos de entrega de la aplicación, se ha optado por crear una clase para cada nivel del juego, dejando como futuro trabajo la mejora del código y la reutilización de este.

Aquí se encuentra la parte principal del juego, es la pantalla donde mostramos al personaje y donde van apareciendo los granos que tendremos que ir explotando. Para la creación de los granos, se ha utilizado un controlador de tiempo (*TimerHandler*) el cual ejecuta el código que queremos cuando pasa el tiempo que le pasamos como parámetro. A continuación se explica brevemente la función que se ha implementado en cada método:

- onCreateEngineOptions(): método donde especificamos las opciones del motor del juego.

- onCreateResources(): método donde cargamos todos los recursos necesarios para esta pantalla (imágenes, audio, etc.).

- onCreateScene(): método donde creamos la escena y añadimos el fondo y la barra de la puntuación. Desde aquí se llama al método para crear el gestionador del tiempo (*TimerHandler)*.

- crearTimeHandler(): aquí es donde creamos el gestionador de tiempo, en el cual cada vez que pase el tiempo marcado se creará un grano llamando a un método crearGrano(), y cada vez que esto ocurra modificaremos el tiempo (un valor aleatorio) que tiene que pasar para que se vuelva a ejecutar el código de su interior.

- crearGrano(): en este método es donde se crean los granos rojos, granos verdes, o besos, según el nivel de dificultad. Los granos también pueden aparecer en distintas zonas de la cara según la dificultad del nivel. En este método se gestiona todo esto, se le asigna una zona de la cara (posición x e y), y aleatoriamente se crea uno de los elementos anteriormente mencionados, sobre escribiendo para dicho elemento el método onAreaTouched() para gestionar la respuesta del videojuego al evento de tocado del elemento por parte del usuario. Cuando el usuario pulsa, mostraremos la animación correspondiente y añadiremos los puntos y modificaremos la barra de puntos.

- mostrarPantallaWin(): este método se lanzará cuando el usuario alcance los puntos necesarios para superar el nivel. Simplemente mostraremos una imagen a pantalla completa para que el usuario sepa que ha

conseguido superar dicho nivel. Cuando el usuario pulse en la imagen, se lanzará la actividad *LevelActivity*.

- mostrarPantallaLose(): igual que el método anterior, pero para cuando el usuario ha perdido.

- onSetContentView(): en este método creamos un FrameLayout con código Java para así añadir un banner de publicidad, con esto conseguimos monetizar nuestra aplicación.

5. Videojuego III: Minicars

Para el tercer videojuego desarrollado, se ha optado por buscar algo nuevo a lo que no se había hecho en los dos videojuegos anteriores. Este videojuego trata de un juego de carreras, en el cual tendrás que pasarte un circuito (acabando antes del tiempo marcado) para poder desbloquear el siguiente circuito. Para el desarrollo de este videojuego se ha utilizado una extensión del motor gráfico *AndEngine*, como ha sido la extensión de *Physics Box2D*.

5.1 Descripción del videojuego

"Minicars" es un juego típico de carreras de coches, pero en este tus contrincantes no serán otros coches, sino el cronómetro y los obstáculos que haya en la carretera. El juego es muy fácil de jugar, ya que el manejo del coche será con un joystick táctil situado en la esquina inferior izquierda de la pantalla. En un principio solo habrá disponible un circuito, y para jugar a los siguientes circuitos tendrás que superar el circuito anterior.

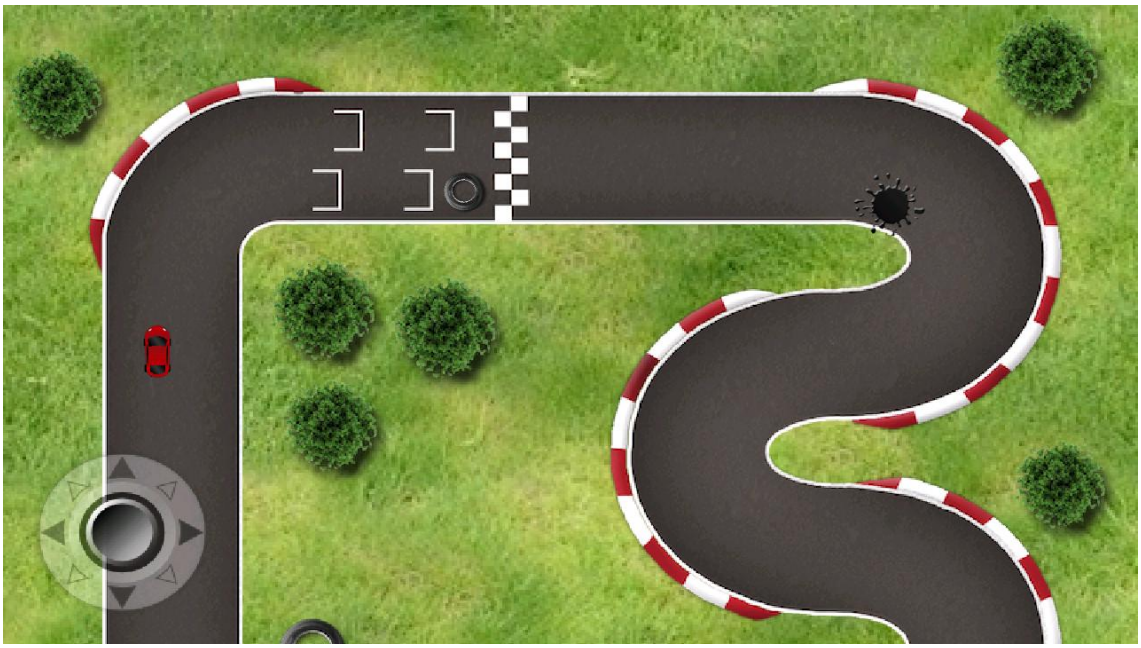

Imagen 20: Captura de la pantalla principal del juego "Minicars"

Elementos del juego

El juego consta de varios elementos principales con los que hay que interaccionar para vivir la experiencia del juego:

El coche: se trata del vehículo que manejaremos, con el que tendremos que recorrer el circuito.

El joystick: con este control de fácil manejo desplazaremos el coche a lo largo del circuito.

Los obstáculos: en los circuitos habrán diferentes obstáculos para entorpecer la conducción del coche y así añadir dificultad y entretenimiento a la partida.

El circuito: el circuito en sí, el trazado de la carretera. Este trazado tendrá "muros" a los lados de la carretera para que el coche no pueda salirse en exceso del circuito trazado.

El cronómetro: este cronómetro nos mostrará el tiempo que llevamos en el circuito, para así ver si vamos bien de tiempo o no.

Mecánica del videojuego

La meta del juego es acabar la carrera antes del tiempo marcado. Con esto conseguiremos ganar la carrera y desbloquear un circuito nuevo.

En un principio solo tenemos abierto un circuito, y al ir superando las carreras, se irán abriendo nuevos circuitos.

Manejo por parte del usuario

El control del coche se realiza a través de la pantalla táctil, gracias a un joystick situado en la parte inferior izquierda de la pantalla. Tendremos que pulsar y mantener pulsado la bola del joystick y moverla en la dirección que queremos que se mueva el coche.

Herramientas utilizadas

La aplicación se ha desarrollado utilizando el lenguaje Java, el entorno de desarrollo que proporciona el *plugin* ADT para Eclipse y el motor de juego *AndEngine*, además, se ha añadido una extensión de física del motor gráfico, la extensión *Physics Box2D*.

5.2 Análisis y especificación de requisitos

A continuación se presenta la especificación del videojuego. Al igual que para el anterior videojuego, la fase de especificación ha sido ajustada al tamaño del proyecto; como se trata de una aplicación sencilla y que además se debe hacer en un período de tiempo reducido no es viable hacer una especificación propia de un proyecto más complejo y largo.

5.2.1 Diagrama de casos de uso

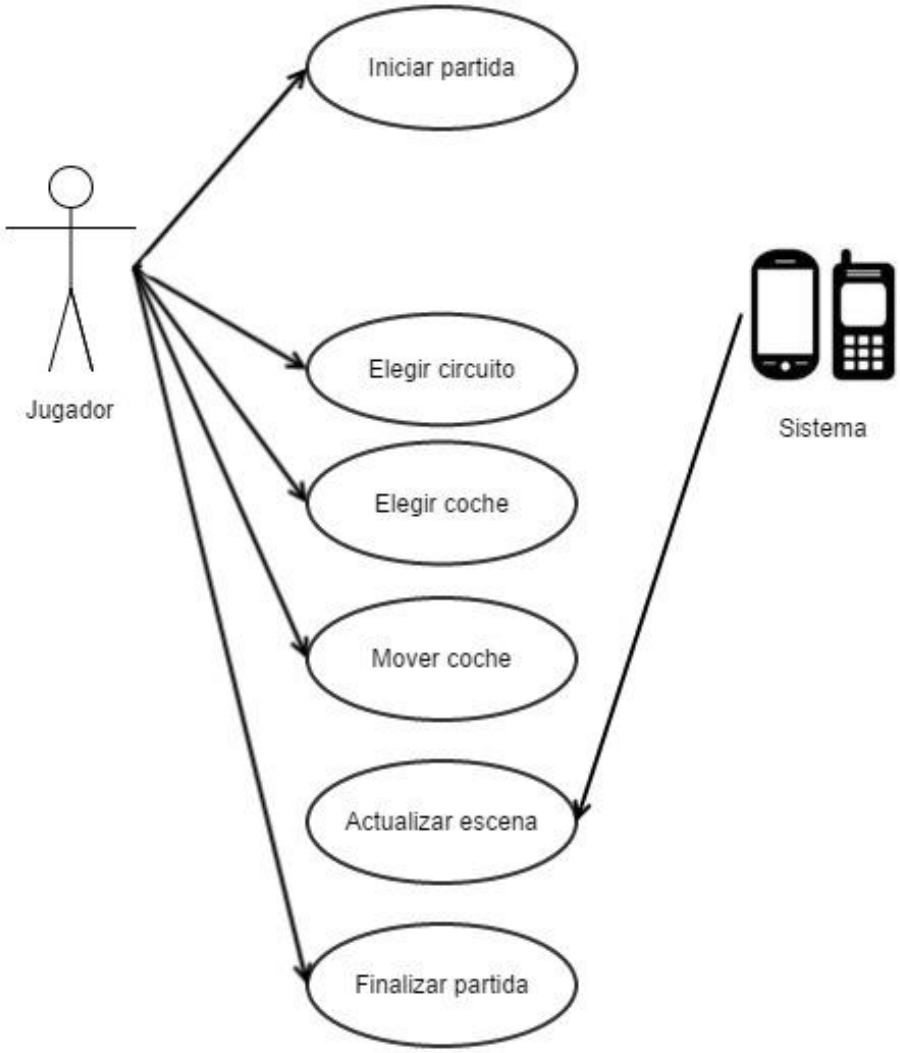

Imagen 21: Diagrama de casos de uso del videojuego "Minicars"

5.2.2 Definición de los casos de uso

Caso de uso: Iniciar partida

Actor: Jugador
Descripción: El jugador podrá iniciar una partida en cualquier momento.
Escenario: El usuario inicia la aplicación y tras pulsar en el botón *Start* indica al sistema que quiere empezar una nueva partida.

Caso de uso: Elegir circuito

Actor: Jugador
Descripción: El jugador podrá elegir el circuito al que desea jugar.
Escenario: El usuario deberá pulsar sobre el circuito al que quiera jugar entre los que tenga disponibles, ya que para poder jugar a todos los circuitos tienes que haber ganado la carrera en los circuitos anteriores.

Caso de uso: Elegir coche

Actor: Jugador
Descripción: El jugador podrá elegir el coche con el que quiere correr la carrera.
Escenario: El usuario deberá pulsar sobre el coche que quiere para correr la carrera y así pasar a la siguiente pantalla con dicho coche.

Caso de uso: Mover coche

Actor: Jugador
Descripción: El jugador podrá mover el coche en la dirección que quiera.
Escenario: El usuario deberá mover la bola del joystick táctil para desplazar el coche a lo largo del circuito en la dirección que quiera.

Caso de uso: Actualizar escena

Actor: Sistema
Descripción: El jugador se encuentra jugando una partida y quiere que el sistema simule en tiempo real dicha partida.
Escenario: El sistema irá actualizando la escena cada cierto tiempo, es lo que se denomina actualización *frame* a *frame*. El "reloj" avisa al sistema de que ya ha pasado el tiempo que dura un *frame* desde la última vez que se ejecutó este caso de uso, así que el sistema actualiza los elementos del mundo teniendo en cuenta el tiempo que ha pasado desde la última actualización.

Caso de uso: Finalizar partida

Actor: Jugador
Descripción: Habiendo una partida en marcha, el jugador desea finalizar la partida.
Escenario: El jugador indica al sistema que quiere finalizar la carrera en curso. El sistema cierra dicha partida sin guardar el estado de la carrera, pasando a permitir que el jugador elija otro coche (Caso de uso “Elegir coche”) distinto para volver a jugar.

5.3 Diseño

En los siguientes apartados se pasa a describir los diferentes componentes que forman el videojuego, así como las clases que lo componen (diagrama de clases).

5.3.1 Componentes de la aplicación

En este apartado se van a mostrar los diferentes componentes que conforman el videojuego “Minicars”. La siguiente imagen, aparte de mostrar los componentes, nos va a permitir entender fácilmente cómo interactúan entre ellos y como el usuario puede navegar por ellos.

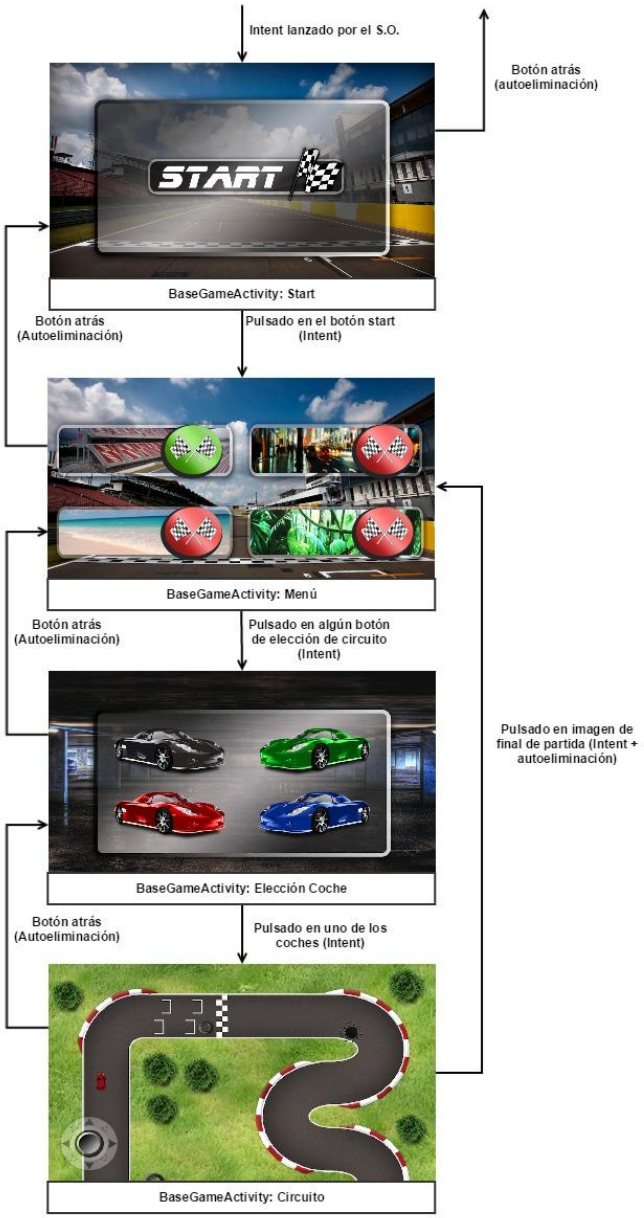

Imagen 22: Mapa de navegación del videojuego "Minicars"

Como podemos observar, los cuatro componentes de la aplicación son Actividades del tipo *BaseGameActivity*.

Recordar para entender el diagrama anterior, que cada una de las flechas que va de una actividad a otra representa una navegación que puede ser efectuada durante la ejecución de la aplicación. En la información de estas flechas indicamos, en primer lugar, el tipo de suceso que se deberá dar o el tipo de acción que el usuario deberá realizar sobre la actividad actual para ir hasta la actividad a la que se dirige la flecha. Y en segundo lugar, entre paréntesis, las acciones que realizará la actividad en la que nos encontramos para que al usuario se le muestre la actividad destino. Estas acciones pueden ser tres: *intent,* auto eliminación, *intent + auto eliminación* (explicadas anteriormente).

Pasamos a describir el funcionamiento concreto de cada una de las actividades que podemos visualizar en el diagrama anterior:

- En primer lugar tenemos la actividad *Start,* cuya función es introducir el juego y mostrar un botón de *Start* para pasar el menú de elección del circuito. Esta pantalla de introducción del juego se ha creado con vistas a en un futuro crear diferentes modos de juego, y así poder poner más botones en este menú. Cabe decir que es el sistema operativo Android el que envía el *intent* que crea la instancia de la actividad *Start,* y lo hace cuando el usuario le indica que desea ejecutar la aplicación "Minicars". Si estando en la actividad *Start* el usuario pulsa el botón *Back,* la aplicación se cerrará.

- En segundo lugar tenemos la actividad *Menu*, en la cual tenemos los diferentes circuitos para jugar. En la primera partida solo podremos pulsar sobre el botón del primer circuito, ya que los demás circuitos estarán bloqueados hasta que el usuario vaya superando los circuitos anteriores. Cuando el usuario pulsa en uno de los circuitos de esta actividad lanza un nuevo *Intent* para iniciar la actividad *Elección coche*. Si estando en la actividad *Menu* el usuario pulsa el botón *back*, está actividad se auto eliminará dejando paso a la actividad anterior (*Start*).

- En tercer lugar tenemos la actividad *EleccionCoche*, en la cual tendremos cuatro coches diferentes a elegir para correr en el circuito elegido previamente. Al pulsar en uno de los coches, se lanza un *intent* para pasar a la actividad de *Circuito*.

- En último lugar tenemos la actividad *CircuitoX* (donde x es el número del circuito, ya que hay varios circuitos). Aquí es donde se encuentra la parte importante del juego, es decir, la carrera. El coche con el que se corre es el elegido en la actividad anterior (*EleccionCoche*). Si pulsamos el botón *back* volveremos a la actividad de elegir el coche, pero si acabamos la carrera y pulsamos en la imagen, se lanzara un *intent* con la actividad de *Menu* y se auto eliminará esta actividad.

5.3.2 Diagrama de clases

Como siguiente paso se representa el diagrama de clases del videojuego "Minicars". Debido a la baja magnitud de este videojuego, se trata de un diagrama de clases sencillo.

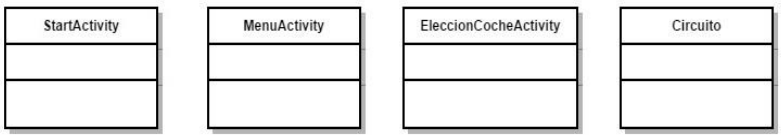

Imagen 23: Diagrama de clases del videojuego "Minicars"

A continuación se procederá a explicar las clases del diagrama:

- *StartActivity*: la primera actividad que se lanza de la aplicación. En esta actividad (que hereda de *BaseGameActivity*, al igual que todas las clases de este videojuego) se carga una imagen de fondo y se añade un *Sprite* haciendo la función de botón, que al pulsar sobre este se lanza la actividad *MenuActivity*.

- *MenuActivity*: en esta actividad se carga una imagen de fondo y se añaden varios botones con una imagen que identifica al circuito que es, y al pulsar sobre uno de los botones se lanza la actividad *EleccionCocheActivity* pasándole como parámetro a través del *intent* la opción elegida por el usuario.

- *EleccionCocheActivity*: en esta actividad se carga una imagen de fondo y se añaden varios botones con los diferentes coches que el usuario puede escoger, y al pulsar sobre uno de los coches se lanza la actividad *Circuito* (la actividad del circuito correspondiente al elegido por el usuario) pasándole como parámetro a través del *intent* la opción del coche elegido por el usuario.

- *Circuito*: esta clase es la que contiene la lógica de todo el manejo de la pantalla principal del juego. En esta es donde se cargan los recursos necesarios para la construcción del circuito, se crea el escenario, se gestiona el control del coche a través del joystick táctil, etc.

5.4 Implementación

El propósito de este capítulo es describir al lector cómo se ha implementado la arquitectura y el diseño detallado en los apartados anteriores. Se realizará un estudio en profundidad de las funcionalidades implementadas dando al lector una visión más detallada sobre el funcionamiento interno.

Gestión de recursos

A continuación se va a detallar cual ha sido la solución tomada para la gestión de los recursos externos que utiliza la aplicación. El proyecto que se ha desarrollado utiliza varios tipos de ficheros externos, estos están almacenados en la carpeta "gfx" y "font" que está situada a su vez dentro de la carpeta "assets" del proyecto. En esta carpeta se almacenan todas las imágenes, ya que todas las clases del juego están implementadas extendiendo la clase BaseGameActivity del motor gráfico *AndEngine*. Esta carpeta está compuesta por la siguiente estructura:

- Carpeta "*start*": aquí se almacenan todas las imágenes necesarias para la clase *StartActivity*.
- Carpeta "*menu*": en esta carpeta se almacenan todas las imágenes necesarias para la clase *MenuActivity*.
- Carpeta "elegircoche": aquí se almacenan todas las imágenes necesarias para la clase *EleccionCocheActivity*.
- Carpeta "*circuito*": aquí se almacenan todas las imágenes necesarias para la clase *Circuito*.
- Carpeta "font": carpeta que contiene el archivo para la fuente utilizada en el texto del videojuego.

StartActivity, MenuActivity y EleccionCocheActivity

Estas, al igual que todas las clases de este videojuego, extienden de la clase *BaseGameActivity*. En ellas simplemente se ha implementado un menú con botones. En *StartActivity* el menú consta de un botón para pasar a la elección del circuito. A continuación, cuando pulsamos en este botón, pasamos a la pantalla de *MenuActivity* con los circuitos disponibles para elegir.

En *MenuActivity* crearemos un menú con los diferentes circuitos a elegir para jugar. El juego empezará con solo el circuito 1 desbloqueado, y cuando se pase un circuito, se desbloqueará el siguiente. Este control lo llevamos guardando una variable en las preferencias de usuario (*SharedPreferences*) para saber qué circuito es el último que el usuario ha superado.

Una vez elegido el circuito se pasa a la clase *EleccionCocheActivity*, donde mostramos un menú con los cuatro coches diferentes a elegir. Cuando el usuario pulsa en uno de los coches se lanza la actividad con el circuito que previamente había elegido el usuario, y se le pasa como parámetro el coche elegido para cargarlo en la siguiente actividad.

Circuito

Se ha decidido crear una clase para cada circuito ya que cada circuito es de diferente forma.

Aquí se encuentra la parte principal del juego, es la pantalla donde se corre en el circuito. Para el control del tiempo se ha utilizado un controlador de tiempo (*TimerHandler*) el cual ejecuta el código que queremos cuando pasa el tiempo que le pasamos como parámetro. Y para crear unos muros invisibles para que el coche no se pueda salir en exceso de la carretera, se ha utilizado una extensión (*Box2D*) para el control de físicas. A continuación se explica brevemente la función que se ha implementado en cada método:

- onCreateEngineOptions(): método donde especificamos las opciones del motor del juego.

- onCreateResources(): método donde cargamos todos los recursos necesarios para esta pantalla (imágenes, fuente, etc.).

- onCreateScene(): método donde creamos la escena y llamamos a los otros métodos para completar la creación de la escena y manejar la interacción con el usuario. Desde aquí se llama al método para crear el gestionador del tiempo (*TimerHandler)*.

- crearTimeHandler(): aquí es donde creamos el gestionador de tiempo, en el cual cada vez que pase el tiempo marcado (1 segundo) se cambiará el tiempo mostrado en el cronómetro .

- iniciarParedes(): aquí definimos los muros del circuito para que el coche no se salga en exceso de la carretera. Estos son cuerpos físicos que se crean utilizando la extensión *Box2D*.

- iniciarCoche(): aquí creamos el coche como una caja con propiedades físicas. Esto también se hace con la extensión *Box2D*.

- anyadirRuedas(): aquí creamos obstáculos para añadir al circuito como lo son unas ruedas de coche, estas también tendrán propiedades físicas para que el coche pueda chocar con ellas.

- iniciarControles(): aquí creamos el joystick y gestionamos la interacción del usuario con este y el movimiento del coche que produce dicha interacción con el joystick.

- iniciarPuntosControl(): en este método creamos y añadimos varios *sprites* a lo largo del circuito como puntos de control para saber que el coche ha pasado por esos puntos y que ha pegado una vuelta completa al circuito.

6. Planificación y costes

En este capítulo se llevará a cabo la demostración de las horas empleadas para cada tarea. Además, a partir de las horas utilizadas calcularemos los costes que habría tenido el proyecto si hubiese sido desarrollado por un equipo de personas en una empresa ficticia.

6.1 Planificación del proyecto

La planificación no es más que un instrumento de trabajo a través del cual podemos acotar el tiempo de trabajo que llevará un proyecto y avistar a tiempo posibles desviaciones en la planificación, por tal de evitar que la duración del proyecto sea mayor de la que teníamos prevista o que, en caso de alargarse, se alargue lo mínimo posible.

Por tanto, la planificación va cambiando a medida que avanza el proyecto, ajustándose a la realidad que se va dando y detallándose cada vez más. A continuación se presentan las tareas y subtareas en las que se dividió la totalidad del proyecto, con las horas utilizadas para estas tareas.

	Nombre	Duración
1	**Estudio y análisis de Android**	123h
2	**Estudio S.O. Android**	77h
3	Estudio de Android y su funcionamiento	22h
4	Estudio de la estructura y funcionamiento interno de la aplicación	39h
5	Estudio del entorno y herramientas de desarrollo	16h
6	**Análisis**	6h
7	Análisis y selección de las herramientas para desarrollar un videojuego	6h
8	**Estudio de las librerias**	40h
9	Estudio del motor gráfico AndEngine	40h
10		
11	**Primer videojuego "Vestir princesas"**	85h
12	Definición y especificación	5h
13	Implementación, prueba y corrección de errores	80h
14		
15	**Segundo videojuego "Explota granos"**	43h
16	Definición y especificación	3h
17	Implementación, prueba y corrección de errores	40h
18		
19	**Tercer videojuego "Minicoches"**	59h
20	Definición y especificación	3h
21	Implementación, prueba y corrección de errores	56h
22		
23	Documentación (memoria del proyecto)	60h
24		
25	Horas totales invertidas	370h

Imagen 24: Planificación del proyecto. Se muestran las tareas y subtareas en las que se ha dividido el proyecto y las horas dedicadas a cada una de ellas.

Como vemos, el proyecto se puede dividir en cinco fases. La primera de ellas consiste en un estudio y análisis de la plataforma Android. Destaca el análisis de herramientas para el desarrollo de videojuegos, así como del estudio de las librerías empleadas para el desarrollo del videojuego.

En la segunda fase, se llevó a cabo el desarrollo del primer videojuego, "Vestir Princesas". La tercera fase está compuesta por el desarrollo de "Explotar granos" un videojuego con un mecanismo diferente al primero, aunque más simple y, por tanto, con menos tiempo de desarrollo. Y en la cuarta fase se llevó a cabo el desarrollo del tercer videojuego, "Minicoches". En este videojuego entra en juego el motor de física Box2D que posee AndEngine, que se trata de una extensión de la librería principal AndEngine.

En concreto entre las subtareas entendemos como "Definición" a la tarea de definir el concepto del videojuego, pero también se incluye en ella la definición de los menús que tiene el videojuego, así como la definición y diseño de los niveles (incluyendo escenarios, etc.).

La "Especificación" en cambio se refiere, como su nombre indica, a la tarea de definir de forma más técnica y precisa las funciones que va a tener el videojuego, una vez hecha la definición inicial de este. Seguiremos con el diseño, donde se incluye el diseño de la arquitectura del videojuego, es decir, cómo se van a implementar las funciones definidas durante la "Especificación".

"Implementación" incluye la tarea de diseñar los algoritmos concretos con los cuales se va a responder a las peticiones que se hagan a una clase determinada y, cómo no, también incluye la implementación de estas. Nótese que normalmente el diseño de los algoritmos se hace en la fase de "Diseño" pero, en este caso, debido al poco tiempo de desarrollo disponible se ha juntado este diseño de operaciones con la implementación, en una misma tarea.

También tenemos la tarea de "Prueba y corrección de errores", que como su nombre indica ha incluido la prueba de la aplicación y, en caso de un funcionamiento no deseado, la corrección del error. Esta fase de pruebas se ha ido realizando conjuntamente con la implementación, probando la aplicación constantemente en un dispositivo móvil.

Después de la fase de construcción del tercer videojuego, se encuentra la quinta fase, donde se ha elaborado la documentación. Esta incluye la redacción de la memoria.

El orden preciso de ejecución de las diferentes tareas no se muestra. El motivo es que se ha seguido el mismo orden planificado sin ningún cambio al respecto, realizando cada tarea una vez ha sido finalizada la tarea anterior.

6.2 Costes del proyecto

En este apartado se detalla el coste económico final del proyecto, desglosando los diferentes gastos según las horas trabajadas por cada persona. Durante el proyecto ha sido una sola persona la que ha llevado a cabo todas las tareas, pero estas tareas se enmarcan en diferentes roles con diferente reconocimiento cada uno.

En primer lugar, se destaca el rol de director del proyecto, el cual se encarga de gestionar la evolución del proyecto mediante la planificación de este. Al mismo tiempo se encarga de unir y revisar la documentación final. Seguidamente tenemos al analista, encargado de definir los diferentes videojuegos, construyendo, en última instancia, la especificación de los tres productos.

Continuamos con el diseñador, que es el que estudia los entresijos del S.O. Android para poder definir la estructura completa de los tres productos a construir, teniendo en cuenta las propiedades de extensibilidad, portabilidad y reusabilidad, unidas a la propiedad de eficiencia.

También es necesario un diseñador artístico que se encargue de los recursos gráficos y sonoros del videojuego, pero no lo hemos tenido en cuenta en nuestra planificación ya que los recursos me han sido dados, por lo que no me ha supuesto ningún coste de tiempo.

Por último, se encuentra el programador, el cual a través del diseño construye las diferentes clases de que se componen los tres videojuegos y lleva a cabo las pruebas para asegurarse de que funcionen correctamente. Este también necesita conocer el S.O. Android, para poder programar utilizando su API.

Cabe decir que aunque algunas tareas las deben llevar a cabo dos roles al mismo tiempo, como este proyecto está hecho por una única persona, no se contarán esas horas por duplicado. Pues al fin y al cabo, no es tan raro que una aplicación para Android sea desarrollada enteramente por una sola persona.

En la siguiente tabla podemos ver el coste total de cada rol de trabajo necesarios en el proyecto.

Rol de trabajo	**Precio por hora**	**Horas invertidas**	**Coste**
Diseñador	20€	72’5	1.450€
Analista	20€	5’5	110€
Programador	15€	232	3.480€
Director	25€	60	1.500€
		COSTE TOTAL	6.540€

Seguidamente se adjunta una tabla con el coste de los materiales empleados durante la realización del proyecto. Cabe decir que el software que conforma el entorno de desarrollo de Android es totalmente gratuito. Sin embargo, el emulador de Android para PC funciona realmente mal, siendo imprescindible adquirir, al menos, un terminal para desarrollar la aplicación.

En cuanto al resto del software, la licencia de Windows 7 ha sido la que llevaba el ordenador y no se ha contado como coste adicional, pues en caso de no disponer de ella se habría optado por utilizar una variante gratuita de Linux. Y lo mismo ocurre con el resto de aplicaciones utilizadas, como son el procesador de textos, el editor de imágenes y la herramienta para crear la especificación de las aplicaciones. Pues se ha optado por utilizar aplicaciones de pago si se tenía la licencia o aplicaciones gratuitas si no se tenía esta, ya que casi siempre hay una alternativa gratuita.

Otro coste muy importante es la conexión a Internet, pues toda la documentación oficial acerca de Android y acerca del desarrollo de aplicaciones para la plataforma se encuentra en internet. Además de innumerables ejemplos y información de gran valor. Por tanto, aunque nos ayudemos también de libros, es imprescindible contar con conexión a la red.

Smartphone		200€
Ordenador		600€
Conexión a internet	3 meses * 40€	120€
	COSTE TOTAL	920€

Concluimos, que el coste total de desarrollar el proyecto, sumando el coste de los recursos humanos y el coste de los materiales, sería de 7460€.

6.3 Pruebas

Al ser tres videojuegos, las pruebas forman una parte fundamental del desarrollo del mismo.

A cada pequeño paso que se ha dado en el desarrollo se ha ido probando exhaustivamente la aplicación. Esto ha sido posible gracias a que se puede cargar la aplicación en el teléfono desde Eclipse.

Además se ha podido ir cambiando el entorno de la aplicación e idear nuevas formas de lidiar con los problemas. Esto se debe a que un videojuego para Smartphone debe ser dinámico y lo más divertido posible. Parte de esa diversión se debe a que el movimiento de los elementos sea un poco más rápido, que el aprendizaje de uso sea el más sencillo posible o que la interfaz de usuario sea simple.

7. Conclusiones

7.1 Conclusiones generales

Llegados a este punto de la memoria, a continuación se redactan las conclusiones más importantes extraídas de la realización del proyecto. Los objetivos del proyecto se han cumplido prácticamente al completo y a través de ellos podemos extraer conclusiones muy valiosas.

En primer lugar, fruto del estudio y análisis del S.O. Android, podemos concluir que:

- El desarrollo y comercialización de aplicaciones en Android (sean videojuegos o no) es relativamente fácil y barato. Pues el carácter abierto del S.O. junto con las facilidades que Google da al desarrollador facilita mucho la tarea.

- Aun así, las herramientas de desarrollo están lejos de ser perfectas. Pues durante el desarrollo de los videojuegos se han detectado fallos. En concreto, el emulador de Android para PC por el momento ofrece un rendimiento muy malo y es imposible desarrollar un videojuego complejo haciendo las pruebas a través de este emulador. Si no tenemos un dispositivo con ecosistema Android donde ir haciendo las pruebas, no podremos llevar a cabo el desarrollo con éxito.

- Por último, la fragmentación entre diferentes dispositivos con diferentes versiones de Android es un problema. Pues si queremos que nuestra aplicación llegue a más de la mitad de usuario de Android no es viable lanzar una aplicación que utilice una versión concreta del S.O. hasta, al menos, un año después de su salida. Y aun así estaremos llegando a poco más de la mitad de usuario. Esto provoca que mejoras en la API de Android no puedan ser utilizadas por el desarrollador hasta pasado un buen tiempo.

Por tanto, Android es un ecosistema que supone una buena plataforma para el desarrollador a día de hoy, pues ya tiene cierta madurez. Pero aún quedan algunos puntos que mejorar, los cuales enriquecerán aún más la experiencia del desarrollador con la plataforma.

En cuanto al desarrollo de videojuegos en concreto, las conclusiones extraídas son las siguientes:

- Se pueden desarrollar juegos relativamente exigentes en cuanto a número de elementos en pantalla y magnitud de los escenarios haciendo uso únicamente del lenguaje Java y de algún motor gráfico adicional, como ha sido en este caso.

- Cuando desarrollamos videojuegos para dispositivos móviles normalmente hay que priorizar soluciones eficientes en cuanto a tiempo, ante soluciones eficientes en espacio. Y es que, mientras en este tipo de dispositivos la memoria RAM suele ser extensa y es difícil de desbordar, no se puede decir lo mismo de las CPUs, pues estas son limitadas y algoritmos que hagan un uso excesivo de estas acaban afectando muy negativamente al rendimiento del videojuego.

- Otra conclusión importante, es que los videojuegos deben implementar controles adaptados a los dispositivos donde se van a jugar y se debe evitar, por ejemplo, el mapeo de botones en pantalla táctil sin una justificación en cuanto a usabilidad.

7.2 Conclusiones personales

Las conclusiones personales extraídas de la realización del proyecto se pueden resumir en que se han adquirido los conocimientos y experiencia necesarios para poder, en un futuro, desarrollar videojuegos para Android con facilidad. Así como para cualquier otra plataforma móvil, con un período de adaptación a esta relativamente corto. Abriendo de esta forma el camino para poder dedicarme al desarrollo de videojuegos en un futuro.

Pero si nos paramos a detallar este punto, en concreto, se ha conseguido:

- Adquirir los conocimientos necesarios para poder desarrollar tanto aplicaciones simples, como videojuegos que muevan gráficos en tiempo real.

- Se ha aprendido a utilizar el motor gráfico *AndEngine*.

- Se han integrado los conocimientos adquiridos durante la carrera con los nuevos conocimientos, adaptándolos a un sistema operativo diferente. Así, conceptos de Ingeniería del Software, de programación, de planificación de proyectos, se han mezclado para dar lugar a los tres productos desarrollados durante la realización del proyecto.

- Se han desarrollado tres videojuegos funcionales en un periodo de tiempo corto, debido a las necesidades y los tiempos marcados por la empresa.

- Se ha aprendido a monetizar nuestros videojuegos y se ha puesto en práctica en uno de ellos, el cual se ha publicado en el mercado.

- Se ha adquirido la capacidad de acotar un coste en tiempo y dinero para la realización de un videojuego sobre la plataforma Android a través de la experiencia adquirida con los tres productos construidos durante la realización del proyecto de final de carrera.

7.3 Trabajo futuro

Este proyecto con total seguridad no finalizará aquí, pues existen diferentes ramas por donde ampliarlo y mejorarlo.

Una de las mejoras a hacer, y más importante, es la mejora del código escrito en la implementación de los tres videojuegos. Debido a que este proyecto ha sido realizado en una empresa se ha tenido un tiempo limitado para la realización de los productos finales, optando así a escribir código buscando alcanzar el objetivo final, sin tener en cuenta la reutilización de código, la extensibilidad, la portabilidad, y demás aspectos para hacer más entendible el código final. Esta mejora es muy importante para que cualquiera que quiera mejorar el proyecto entienda más fácilmente como está implementado dicho producto, y también para la reutilización y comprensión de código para construir videojuegos con similares características.

Otra de las mejoras que se podrían hacer es en el juego de "Minicars", se podría cambiar el modo de control del coche, cambiando el modo actual de joystick por un modo en el que el coche siempre vaya hacia delante (acelerando), y el usuario solo tenga que pulsar la pantalla para frenar, y girar a la izquierda o derecha el dispositivo para que el coche gire (utilización del acelerómetro).

También se podrían llevar a cabo algunas ampliaciones como estas:

- En el juego de "Minicoches" se podría implementar un modo online, en el cual el jugador se enfrentara a otro jugador de cualquier parte del mundo, elegido aleatoriamente.

- También se podría añadir la opción de compartir con Facebook, Twitter, etc., y así poder ver la puntuación de tus amigos, tanto en el juego de "Explotar granos" como en el juego de "Minicars". Esto sería un aliciente más para que los usuarios jugarán a este juego, ya que el poder rivalizar con tus amigos convierte a los juegos en más adictivos.

Este proyecto ha sido realizado en el marco de colaboración con una empresa. Esto significa que aprovechando los conocimientos adquiridos se van a realizar nuevos juegos, aumentando cada vez más el nivel de calidad y dificultad de los juegos.

Otro de los planes futuros es el desarrollo de estos mismos videojuegos para otras plataformas móviles, como lo son iOS y Windows Phone.

8. Bibliografía

[1] Zechner, Mario. Beginning Android Games. Editorial Apress – 2011

[2] Lequerica, Joan Ribas. Manual imprescindible de desarrollo de aplicaciones para Android. Editorial Anaya Multimedia – 2012

[3] Silva, Vladimir. Advanced Android 4 Games. Editorial Apress – 2012

[4] J. F. DiMarzio. Practical Android 4 Games Development. Editorial Apress – 2011

[5] Android Official Developer Program: [Consulta 20/06/2014]

http://android.developer.com

[6] Foro especializado en Android: [Consulta 30/06/2014]

http://www.android-spa.com

[7] Foro especializado en Android: [Consulta 15/07/2014]

http://www.stackoverflow.com

[8] Empresa de monetización de aplicaciones Startapp: [Consulta 20/08/2014]

http://startapp.com/

[9] AndEngine: [Consulta 10/07/2014]

http://www.andengine.org/

[10] Wikipedia [Consulta 10/09/2014]

http://es.wikipedia.org/

Anexo I: Monetización de apps

La monetización de una aplicación consiste en ganar dinero gracias a nuestra aplicación. Hay diferentes maneras de conseguir este propósito:

Apps Gratuitas con In-App Advertising

Aplicaciones, cuya descarga es gratuita, que incluyen publicidad de terceros como vía para monetizar sus espacios publicitarios. Los formatos son muy variados, desde pequeños banners en la parte superior de la pantalla hasta vídeos a pantalla completa. En este caso el usuario descarga la app gratuitamente y además con todas las funcionalidades. ¿Qué precio tiene que pagar? Ver publicidad. Este es el caso de muchas apps, algunas tan conocidas como Angry Birds, y este será el caso utilizado para monetizar los productos de este proyecto. Hay muchas empresas de publicidad, como lo son: AdMob (de Google), Startapp, Leadbolt, Mobfox, etc.

Apps Freemium con compras In-App

Las apps Freemium son aplicaciones gratuitas que ofrecen servicios o prestaciones adicionales a cambio de un pago extra. Según diferentes estudios, esta categoría es la que genera más ingresos en las distintas tiendas de aplicaciones. Esta es la opción recomendada para apps con gran volumen de descargas como puede ser el caso de Candy Crush Saga, o Clash of Clans, entre otras.

Aplicaciones de Pago

El usuario paga por la descarga. El usuario visita el market en cuestión y paga un precio por descargar la app. Tanto Apple como Google se quedan con un porcentaje, el 70% para el desarrollador y el 30% para ellos. En términos generales, a los usuarios no les gusta pagar y cada vez les gusta menos. Dentro de las apps de pago existe una variante, el de las versiones *Lite* (o de prueba). Son apps gratis que el usuario puede descargar para probar el "*producto*" antes de comprar la versión completa que sí hay que pagar. Son apps que, por ejemplo, traen funcionalidades "*bloqueadas*" (casi siempre las más interesantes), en el caso de los videojuegos suelen venir con unos pocos niveles de prueba…

Apps con modelo de Suscripción

Se trata de Apps en la que los usuarios pueden suscribirse, previo pago, claro. Periódicos y revistas suelen utilizar este modelo de negocio.

Anexo II: Manual “Vestir princesas”

En este anexo se detalla cómo navegar por las diferentes pantallas de “Vestir Princesas”, así como el manejo de todos los elementos que componen el videojuego. Después de leer esta guía, el usuario sabrá todo lo necesario para poder jugar al videojuego con éxito.

Mecanismos y controles

El objetivo de “Vestir Princesas” es muy simple. En la pantalla del vestidor se situará la princesa en el centro de la pantalla y nosotros deberemos vestirla a nuestro gusto. Si pulsamos el botón para cambiar a la pantalla del maquillador, en el centro de la pantalla se encontrará la cabeza de la princesa, y podremos cambiarle el peinado, el color de ojos, y el color de labios, a nuestro gusto. Este videojuego no tiene final, el usuario puede estar el tiempo que quiera cambiando el aspecto de nuestra princesa.

Tanto para cambiar las prendas de ropa, como para cambiar el aspecto de la cara, se utiliza el deslizamiento con el dedo. Para ello, debemos pulsar en la prenda que queramos, de las que se ven en la parte inferior de la pantalla, y sin soltar, arrastrar hasta que la prenda (o parte de la cara) entre en contacto con la zona correspondiente de la princesa. Una vez entre en contacto la prenda o parte de la cara con la zona correspondiente, esta se colocará automáticamente de la forma correcta en la princesa.

Interfaz

Nada más iniciar la aplicación nos encontraremos con la pantalla de splash, en la que hay una imagen con la protagonista del videojuego. Para poder pasar el menú de elección de princesa, lo único que hay que hacer es tocar sobre cualquier punto de la superficie de la pantalla táctil.

Tras esto, llegamos a la pantalla del menú de elección de la princesa. Aquí disponemos de cuatro botones con las caras de las princesas para elegir, y según el que pulsemos, iniciaremos la partida con la princesa teniendo el aspecto elegido.

La siguiente pantalla que aparecerá será la del vestido. Aquí es donde empieza el juego de verdad. Para empezar, en la parte inferior de la pantalla se mostrarán las prendas de la parte superior del cuerpo. Debajo de las prendas hay situada una barra que sirve de scroll para poder ver todas las prendas disponibles.

Para poner cualquiera de estas prendas solo tenemos que hacer lo explicado anteriormente, pulsar cualquier prenda, y manteniendo pulsado, arrastrar hasta la zona del cuerpo en la que va situada dicha prenda.

En la esquina superior izquierda tenemos un botón que sirve para cambiar el fondo de la pantalla, situando así a nuestra princesa en diferentes escenarios. En el lateral derecho de la pantalla se encuentran los siguientes botones, por orden de arriba a abajo: cambiar a maquillador, prendas superiores, prendas inferiores, zapatos y abalorios. Podemos cambiar el tipo de prendas que se muestran en la parte inferior pulsando los botones correspondientes. Si pulsamos el botón de cambiar a maquillador, el juego cambiará totalmente de escena.

La pantalla de maquillador funciona exactamente igual que la pantalla de vestidor. En la parte inferior se muestran las partes de la cara disponibles para cambiar, y en la parte derecha de la pantalla se encuentran los botones para cambiar el tipo de partes de la cara a mostrar, y un botón para volver a la pantalla del vestidor. Si hemos hecho alguna modificación en la cara de la princesa en la pantalla del maquillador, este cambio se verá reflejado en la pantalla del vestidor.

Anexo III: Manual “Explotar granos”

En este anexo se detalla cómo navegar por las diferentes pantallas de “Explotar granos”, así como el manejo de todos los elementos que componen el videojuego. Después de leer esta guía, el usuario sabrá todo lo necesario para poder jugar al videojuego con éxito.

Mecanismos y controles

El objetivo de “Explotar granos” es acabar con los granos lo más rápido posible para así acumular más puntos y superar el nivel. En la pantalla de juego donde aparece la cara de nuestro personaje irán apareciendo granos y el usuario deberá pulsar sobre ellos para explotarlos, y cuando aparezcan besos el usuario deberá intentar evitarlos para no perder puntos.

Interfaz

Al iniciar la aplicación nos encontraremos con un menú en el que habrá dos botones con la cara de nuestros personajes. Deberemos elegir uno de los dos para jugar con el y pasar a la siguiente pantalla.

Una vez elegido el personaje se mostrará la pantalla de selección de nivel. Los niveles que hay sin candado son los niveles disponibles para jugar, y para tener disponible un nivel se debe superar el nivel anterior. En la esquina superior izquierda tenemos un botón de información para ver una imagen a modo de tutorial de cómo funciona el videojuego.

Una vez seleccionado el nivel que queremos para jugar, se pasa a la pantalla principal del videojuego. En esta pantalla podemos ver en la parte superior una barra rellenada en parte de color rojo y de color verde. Esta barra hace la función de medidor de la puntuación, cuantos más puntos consigamos, más parte de barra verde habrá. El objetivo es conseguir los puntos suficientes para rellenar la barra completamente de verde, y así superar el nivel.

Para conseguir esto puntos lo único que tenemos que hacer es ir explotando los granos que van apareciendo en la cara de nuestro personaje. Los granos rojos tienen una puntuación diferente según el nivel de dificultad de la pantalla, los granos verdes tienen una puntuación superior a los rojos, y los besos hay que evitar pulsarlos, ya que si los pulsas te restarán puntos. Si los granos permanecen mucho tiempo en la cara sin que el usuario los explote empezarán a restar puntos. Tanto si conseguimos rellenar la barra de color verde, o por el contrario, se han restado tantos puntos que se ha rellenado de rojo, se mostrará una imagen para avisar al usuario de que ha superado el nivel, o de que ha perdido. El usuario deberá pulsar en la imagen y así volver al menú de elección del nivel de dificultad para volver a jugar.

Anexo IV: Manual “Minicars”

En este anexo se detalla cómo navegar por las diferentes pantallas de “Minicars”, así como el manejo de todos los elementos que componen el videojuego. Después de leer esta guía, el usuario sabrá todo lo necesario para poder jugar al videojuego con éxito.

Mecanismos y controles

El objetivo de “Minicars” es superar los circuitos acabando las carreras antes del tiempo marcado. Para ello, el usuario tiene que manejar el coche con el joystick táctil que hay en la parte inferior izquierda de la pantalla.

El manejo del joystick es muy simple, hay que pulsar en la bola del centro de este y, sin soltar, desplazar en la dirección en la que queremos que se desplace el coche.

Interfaz

Al iniciar la aplicación nos encontraremos con un menú en el que habrá un botón de *Start* para que al pulsar pasemos a la pantalla de elección del circuito. Los circuitos que tienen un círculo verde son los circuitos disponibles para jugar, y para tener disponible un circuito se debe superar el circuito anterior.

Una vez seleccionado el circuito que queremos para jugar, se pasa a la pantalla de selección del coche, donde tenemos que pulsar en uno de los cuatro coches para correr con este y pasar a la pantalla principal del videojuego.

Cuando ya hemos elegido el coche, pasamos a la pantalla del circuito, en esta veremos el coche situado en la meta y el joystick en la esquina inferior izquierda. En la parte superior tenemos el tiempo máximo que podemos tardar para ganar la carrera, y también tenemos el tiempo actual que llevamos de carrera. Si conseguimos acabar la carrera antes del tiempo máximo aparecerá una imagen indicándonos que hemos ganado, y si hemos superado el tiempo máximo, aparecerá una imagen indicando que el usuario ha perdido.