JUEGO 2D PARA ANDROID

Memoria del proyecto

de Ingeniería Técnica en Informática de Sistemas

Realizado por

Sergio Pérez Carretero

y dirigido por

Jordi Pons Aróztegui

Escuela de Ingeniería

Sabadell, Septiembre de 2013

El sotasignat, ***Jordi Pons Aróztegui,***
Professor de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el treball al que correspon la present memòria
ha estat realitzat sota la seva direcció per

Sergio Pérez Carretero

I per a que consti firma la present.
Sabadell, ***Setembre*** de ***2013***

Signat: ***Jordi Pons Aróztegui***

Hoja de resumen

Título:

Juego 2D para Android

Autor:

Sergio Pérez Carretero

Tutor: **Fecha:**

Jordi Pons Aróztequi Septiembre 2013

Titulación:

Ingeniería Técnica en Informática de Sistemas

Palabras clave:

Castellano:, *Smartphone*, *Sistema operativo*, *Java*, *AndEngine*, *Librería, OpenGL*, *App*, *Android*, *SDK*

Català: *Smartphone*, *Sistema operatiu*, *Java*, *AndEngine*, *Llibreria, App*, *Android*, *SDK*

English: *Smartphone*, *Operating System*, *Java*, *AndEngine*, *Library, OpenGL*, *App*, *Android*, *SDK*.

Resumen del Proyecto:

Castellano:

El proyecto consistirá en diseñar desde cero un juego para smartphones que utilicen el sistema operativo Android. El objetivo del juego será sobrevivir a las oleadas de enemigos, que aparecerán en el extremo derecho de la pantalla y se dirigirán al lado opuesto, durante el tiempo suficiente para superar el nivel actual. Al mismo tiempo tendremos que tratar de esquivar los obstáculos y recoger los objetos que nos proporcionarán un bonus de puntos.
En el juego habrá un número finito de niveles, cada uno más difícil que el anterior.

Català:

El projecte consistirà en dissenyar des de zero un joc per smartphones que utilitzin el sistema operatiu Android. L'objectiu del joc serà sobreviure a las onades d'enemics, que apareixeran a l'extrem dret de la pantalla i es dirigiran al costat oposat, durant el temps suficient per superar el nivell actual. Al mateix temps haurem d'esquivar els obstacles i recollir els objectes que ens proporcionaran un bonus de punts.

En el joc hi haurà un número finit de nivells, cadascun més difícil que l'anterior.

English:

The project will on designing from scratch a game for smartphones which use the Android operating system. The game's objective will to survive the enemy waves, that will appear from the right border of the screen and will head to the opposite side, enough time to pass the current level. Meanwhile will have to evade the obstacles and gather objects that will provide us a bonus in points.

The game will have a finite number of levels, each one harder than the previous.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

1. INTRODUCCIÓN

1.1. Presentación

El objetivo del proyecto es desarrollar un juego 2D para Android, concretamente uno catalogado dentro del género arcade, al que llamaremos SpaceFighter.

Actualmente a nivel mundial sólo el 10% de la población posee un smartphone. Pero por país la situación cambia. Así en Singapur prácticamente todos usan uno ya que el 90% de ellos es dueño de uno. En la lista le sigue en segundo lugar Hong Kong con el 61%, y el tercero pasa a Europa con Suecia con el 52%, y luego le sigue en la lista más países europeos que andan con porcentaje promedio al 40%. De estos casi el 51% usaba Android a finales de 2011, como vemos en la gráfica:

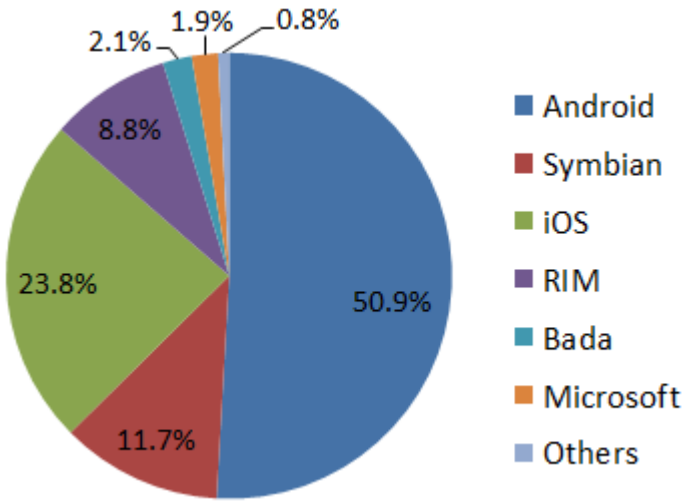

Figura 1: Estadística S.O.

Y según los datos del tercer trimestre del 2012 en cuanto a uso de sistemas operativos móviles en smartphones en todo el mundo, Android sigue predominando sobre el resto con un 72.4% de uso.

Como datos curiosos tenemos que: el 61% de los smartphones son utilizados principalmente para jugar. Y el 90% de las aplicaciones descargadas no se usan más de 10 veces.

Como esto es así, decidí hacer un juego 2D offline para la plataforma Android. 2D porque aunque los juegos más comúnmente producidos hoy día son los 3D, los juegos 2D son más casuales que los 3D, y teniendo en cuenta la estadística anterior, encajaría mejor en el contexto que tenemos. Y offline por que mucha de la gente que usa smartphones para jugar lo hace mientras usa transporte público donde la conectividad es reducida o nula.

1.2. Motivación

Tengo dos motivaciones principales para realizar este proyecto: La primera es mejorar mi comprensión del lenguaje java y aprender a usar el SDK (Software Development Kit) de Android. Hasta el momento, mi contacto con el lenguaje ha sido casi nulo y nulo por completo en con las herramientas proporcionadas de Google y en mi opinión es muy conveniente familiarizarse con ambas debido a su gran extensión.

Mi segunda motivación, como en el caso de lo anteriormente mencionado, es también la primera vez que voy a desarrollar un juego y la primera vez que uso AndEngine o cualquier otro motor gráfico. Creo que me ayudará a comprender mejor el funcionamiento interno de los juegos en general y ampliar conocimientos sobre operaciones de transformación de imágenes y otros temas vistos en asignaturas de la propia titulación como Técnicas Gráficas, por ejemplo.

1.3. Objetivos

Un resumen de mis objetivos personales, a parte de crear un juego que sea, no innovador, pero mínimamente entretenido, serían los siguientes:

- Conseguir algo de soltura desarrollando aplicaciones en Java.
- Aprender a trabajar con el SDK de Android, conocer sus herramientas, límites y funcionamiento.
- Ver y entender el funcionamiento interno de los juegos en 2D (movimiento de sprites, eventos, detección de colisiones y animación simultánea de imágenes).

Objetivos funcionales:

- Abarcar el máximo número de versiones de Android sobre las que funcione el juego: Debido a la gran variedad de dispositivos que tienen instalado el sistema operativo Android, hoy en día tenemos una numerosa cantidad de versiones del sistema a las que tenemos que proporcionar compatibilidad, por suerte el motor que usaremos da soporte hasta la versión 2.2, que equivaldría a dispositivos de alrededor de 3 años de antigüedad.

Y por último, los no funcionales:

- Conseguir que móviles con procesadores de 800 MHz puedan instalar y ejecutar el juego sin problemas. Esto se traduce en, que la aplicación ocupe un espacio mínimo (entre 5 y 10MB como máximo) y que funcione de manera fluida (que no abuse del uso de RAM).

- Que sea intuitivo: Es decir, que siga el funcionamiento de la mayoría de aplicaciones del género y no sea necesario en ningún momento explicación alguna sobre la lógica del juego.

- Que la base de datos sólo contenga datos vitales: Que la cantidad de datos almacenados no supere 1MB de espacio.

1.4. Estado del Arte

Para la realización de este proyecto he tomado como referencia otros juegos del mismo género, como lo son "Invaders", o el clásico "Defender", o incluso juegos del tipo conocido como "Side Scrolling Game" como sería el mítico Super Mario. Este juego guardará mucha similitud con la mecánica básica de estos títulos.

Me gustaría aportar e implementar cosas nuevas al género, pero tengo presente que en todos ellos se necesitaron equipos de programadores, diseñadores gráficos con experiencia y más tiempo del que yo dispongo, pero al menos me gustaría imitar el diseño general.

1.5. Contenido

A continuación mostraré el estudio de viabilidad donde veremos los requisitos tanto funcionales como no funcionales del proyecto, alternativas en su elección de recursos, un calendario con las fases del proyecto y una evaluación de riesgos. Aquí veremos si llevar este proyecto a cabo es viable o si tenemos que elegir una alternativa.

Luego continuaré con un listado de los recursos que se necesitarán para la realización del proyecto y su uso.

La siguiente parte contendrá lo referente al diseño de la aplicación y comentaremos la estructura y contenido de la base de datos.

Finalmente encontraremos las conclusiones, aquí hablaremos de la desviación temporal respecto a lo programado inicialmente, los fallos u objetivos no conseguidos y su posible solución o mejora o ampliación, mi valoración personal sobre el proyecto y, por último, la bibliografía donde indicaré todas las fuentes consultadas.

2. ESTUDIO DE VIABILIDAD

2.1 Introducción

Existen una gran variedad de juegos para Android y van en aumento tanto su calidad, como su variedad en cuanto a géneros y categorías. Lo que intentaré es coger uno de los géneros más extendidos y desarrollar un juego que contenga todas las características básicas de dicha categoría.

2.1.1 Tipología y palabras clave

El juego se llamará SpaceFighter y será una aplicación implementada para la plataforma Android. Las palabras claves son: Android, App, Java y AndEngine.

2.1.2 Descripción

Como se ha dicho anteriormente hoy en día el uso de los smartphones en la mayoría de países desarrollados es muy elevado, y no tan solo para llamar, sino también para consultar el correo electrónico, navegar por internet, escuchar música o jugar. Esto último ha derivado en la formación de numerosos estudios indie* debido a la gran fuente de ingresos para los desarrolladores de dichos estudios.

Cabe mencionar que muchos de esos juegos requieren conexión de datos para jugar a pesar de no ser un juego multijugador propiamente dicho. Por tanto, en mi opinión un juego que solo requiera de dicha conexión para descargarse e instalarse y posteriormente no la necesitase más sería una mejora respecto a los anteriores.

2.1.3 Objetivos del proyecto

- Aprender a utilizar el SDK de Android y mejorar los conocimientos de java.(O1)
- Entender el funcionamiento de los juegos en 2D en términos generales. (O2)
- Diseñar un juego que proporcione cierto entretenimiento al usuario(O3)
- Desarrollar el juego haciendo que sea fácilmente ampliable, en cuanto a niveles, incorporación de ítems, objetivos, etc. (O4)
- Implementar un sistema de base de datos para conservar el progreso del usuario. (O5)
- Añadir un sistema de puntuaciones. (O6)

Indie: Se dice que un juego es indie cuando ha sido desarrollado por un programador no profesional o por estudios con presupuestos muy bajos.

Priorización de los objetivos:

OBJETIVO	CRÍTICO	PRIORITARIO	SECUNDARIO
O1	X		
O2	X		
O3	X		
O4	X		
O5		X	
O6			X

Tabla 1: Tabla de priorización objetivos.

2.1.4 Partes Interesadas

- Stakeholders:

Nombre	Descripción	Responsabilidad
A	Responsable de la entidad	Aprobación del proyecto. Participa en su definición y hace el seguimiento del proyecto.
B	Responsable contable	Descripción de requisitos y funcionalidades. Hace el seguimiento del proyecto.
C	Director del Proyecto	Supervisa el trabajo del alumno haciendo un seguimiento constante del proyecto.

Tabla 2: Tabla de Stakeholders.

- Perfiles de usuario:

Nombre	Perfil	Responsabilidad
U1	Usuario	Crear partida, seleccionar opciones, jugar, etc.

Tabla 3: Tabla de perfiles de usuario.

- Project Team:

Nombre	Descripción	Responsabilidad
A	Jefe del proyecto (CP)	Define, gestiona, planifica y controla el proyecto.
B	Analista (A)	Colabora con el jefe de proyectos en el estudio de viabilidad y en la planificación. Participa en el diseño y la validación.
C	Programador (P)	Diseña y desarrolla la aplicación de acuerdo con el análisis y la planificación prevista. Participa en el proceso de validación e implementación.
D	Técnico de pruebas (TP)	Realiza los test y participa en el proceso de control de errores.

Tabla 4: Tabla de Project Team.

2.1.5 Referencias

1. Normativa de Projectes d'Enginyeria Tècnica.

http://uab.cat/Document/541/595/Normativa_PFCNovembre2010.pdf

2. LOPD(Ley orgánica de Protección de Datos)

http://noticias.juridicas.com/base_datos/Admin/lo15-1999.html

3. Android Developer Agreement

http://www.android.com/es/developer-distribution-agreement.html

4. AndEngine main site.

http://www.andengine.org/

Producto y documentación del proyecto.

El juego consistirá en sobrevivir a los enemigos que intentarán acabar con el jugador lanzando naves contra él, al tiempo que deberá esquivar los obstáculos que se presenten. Cada vez que se destruya un enemigo el jugador ganará puntos que se verán reflejados en la puntuación final.

El sistema consta de:

- Se entregará una aplicación gratuita. En el futuro se podría subir a Google Play, donde los usuarios de Android podrían descargarla.
- Se elaborará una memoria del proyecto.

2.2 Requisitos del Proyecto

Si el juego pide demasiados recursos del sistema, jugar en algunos dispositivos podría ser imposible. Veamos a continuación los requisitos.

2.2.1 Requisitos funcionales

a) Continuación de partidas guardadas.
b) Sistema de pausa de juego.
c) Selección del nivel de dificultad.
d) Aumento de dificultad al subir de nivel.
e) Funcionamiento del juego sin errores ni ralentizaciones.

2.2.2 Requisitos no funcionales:

a) Conseguir que móviles con procesadores de 800 MHz puedan instalar y ejecutar el juego sin problemas. Esto se traduce en, que la aplicación ocupe un espacio mínimo (entre 5 y 10MB como máximo) y que funcione de manera fluida (que no abuse del uso de RAM).
b) Que la interfaz sea intuitiva.
c) Conexión de internet solo necesaria para descarga.
d) Que el sistema de base de datos sólo contenga datos vitales: Que la cantidad de datos almacenados no supere 1MB de espacio.

2.2.3 Restricciones del sistema:

a) La aplicación se ha de desarrollar para Android.
b) El sistema en el que se ejecute debe tener un mínimo de 10MB de memoria libre.

2.2.4 Catalogación y priorización de los requisitos

Prioridad de los Requisitos Funcionales

	A	B	C	D	E
Esencial				X	X
Condicional	X	X			
Opcional			X		

Tabla 5: Tabla de P. de R.F.

Prioridad de los Requisitos Funcionales

	A	B	C	D
Esencial		X	X	
Condicional	X			
Opcional				X

Tabla 6: Tabla de P. de R.N.F.

Relación de los requisitos con los objetivos del sistema:

	REQUISITOS FUNCIONALES					REQUISITOS NO FUNCIONALES			
	A	B	C	D	E	A	B	C	D
O1	X	X	X	X	X	X	X	X	X
O2		X	X	X	X		X		
O3			X	X	X		X		
O4	X	X	X	X	X	X	X	X	X
O5	X	X	X	X		X		X	X
O6		X					X		X

Tabla 7: Tabla de Relación de los Req. con los Obj. del sistema

2.3 Alternativas y selección de la solución

2.3.1 LibGDX

Usar LibGDX en vez de AndEngine como motor gráfico del juego.

Características: LibGDX es un framework multiplataforma de desarrollo de juegos para Windows, Linux y Android. Está escrito en Java con una mezcla de C/C++ para dar soporte y rendimiento a tareas relacionadas con el uso de la física y procesamiento de audio. De esta forma, sólo hay que preocuparse por la parte que codificas en lenguaje Java mientras el framework se encarga de empaquetar todo el código nativo de las aplicaciones.

Este es el ciclo de vida de una aplicación utilizando LibGDX:

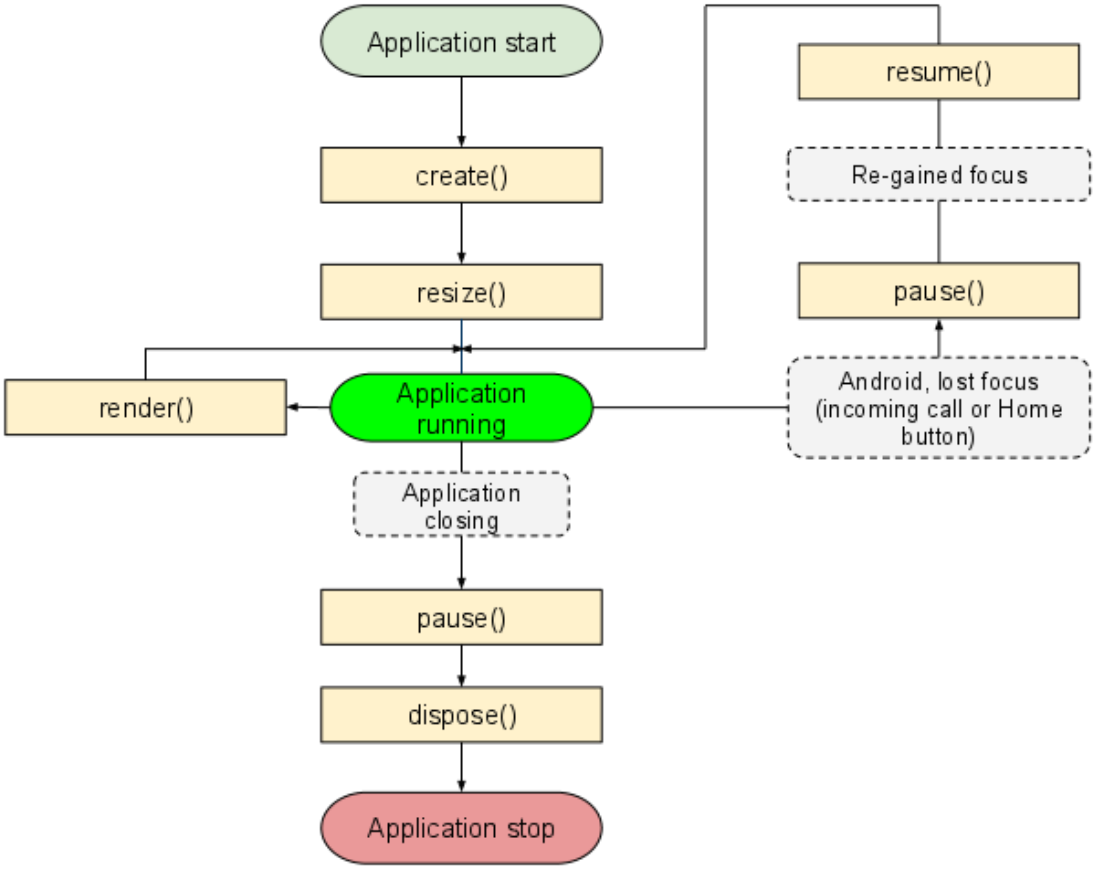

Figura 2: Ciclo de vida LinGDX.

Costes: Ninguno, es software gratuito.

2.3.2 Cocos2d

Cocos2d es un framework, basado en Pyglet, escrito en Python para crear juegos en 2D, y presentaciones gráficas que también cuenta con una versión para iPhone.

Con pocas líneas de código se obtienen variados efectos visuales, rápido y fácil manejo de Sprites y transiciones entre escenas.

Características: Con pocas líneas de código se obtienen variados efectos visuales, rápido y fácil manejo de Sprites. Control de flujo entre escenas, aplicación de acciones a sprites de forma fácil, menús, renderizado de textos, etc. Basado en OpenGL.

Costes: Ninguno, también es software gratuito.

2.3.3 Crear un motor gráfico propio

Esta opción, aunque existe, yo no la considero como tal. Consistiría en escribir el código necesario para proveer al juego de un motor de renderizado de gráficos 2D, un motor u otro sistema que gestionase las físicas(colisiones, gravedad…), los cambios de escena, etc. Y que hiciese de forma más o menos óptima todas las operaciones matemáticas que dichos elementos implican, como escalado, rotación, traslación…

Costes: Económicos ninguno, pero programar todo lo mencionado precisaría de unos conocimientos de los que carezco y de un esfuerzo y tiempo enormes, siendo mucho más difícil la implementación de los recursos para crear el juego que el propio juego.

2.3.4 Solución propuesta

Por lo mencionado anteriormente la opción de crear mi propio motor gráfico queda descartada ya que daría incluso más trabajo que aprender a usar cualquiera de los tres motores y hacer un juego con él.

Aunque cualquiera de las tres opciones hubiera sido aceptable para el desarrollo del juego, la opción de cocos2d parecía más multiplataforma y de uso general que AndEngine que sólo está soportada por Android, además AndEngine tiene una extensión para incorporar cocos2d.

Finalmente en cuanto a LibGDX (y también cocos2d en menor medida) los ejemplos me parecieron más simples y de fácil aprendizaje que los ejemplos de LibGDX, dichos ejemplos se pueden descargar de Google Play y se puede comprobar como algunos ejemplos son simplemente sobre cómo mover un sprite por la pantalla, lo cual realmente facilita mucho su aprendizaje.

2.4 Planificación del proyecto

El proyecto sigue una planificación lineal. Dicha metodología se suele dar en proyectos donde la fecha de entrega está marcada por un usuario final o comprador.

Para estimar la duración se tendrá en cuenta, a parte del procedimiento técnico usado, recursos, costes, etc. Durante desarrollo de la aplicación, a veces, los objetivos cambian de prioridad, aumenta la complejidad prevista o el consiguiente aumento de costes. Estos problemas a veces se pueden arrastrar hasta el final del proyecto.

El proyecto está planificado de la siguiente manera:

- Calendario del proyecto: El proyecto será desarrollado desde Enero de 2012 hasta el Junio de 2012 con una dedicación media de 20 horas semanales. En total se le dedicarán 297 horas.
- Fecha de inicio: 10/01/2012
- Fecha de finalización: 30/06/2012
- Herramientas de planificación y control utilizadas: Microsoft Project.

2.4.1 WBS(Work Breakdown Structure)

Fases y actividades del proyecto

Fases	Descripción
Iniciación	Incluye la definición del proyecto, asignación y matriculación.
Planificación	Incluye el estudio de viabilidad y la planificación.
Análisis	Incluye el análisis de requisitos funcionales y no funcionales.
Diseño	Incluye el diseño de la capa de datos, de control y de interfaz. Diseño de los test.
Desarrollo	Incluye el desarrollo de la aplicación.
Test i pruebas	Incluye test unitarios y de integración.
Generación de documentos	Incluye la documentación del proyecto.
Cierre del proyecto	Incluye el cierre del proyecto. El director del proyecto firma la aceptación y cierre del proyecto.
Defensa del proyecto	Incluye la defensa del proyecto ante el jurado.

Tabla 8: Tabla de Fases y actividades del proyecto

Diagrama WBS:

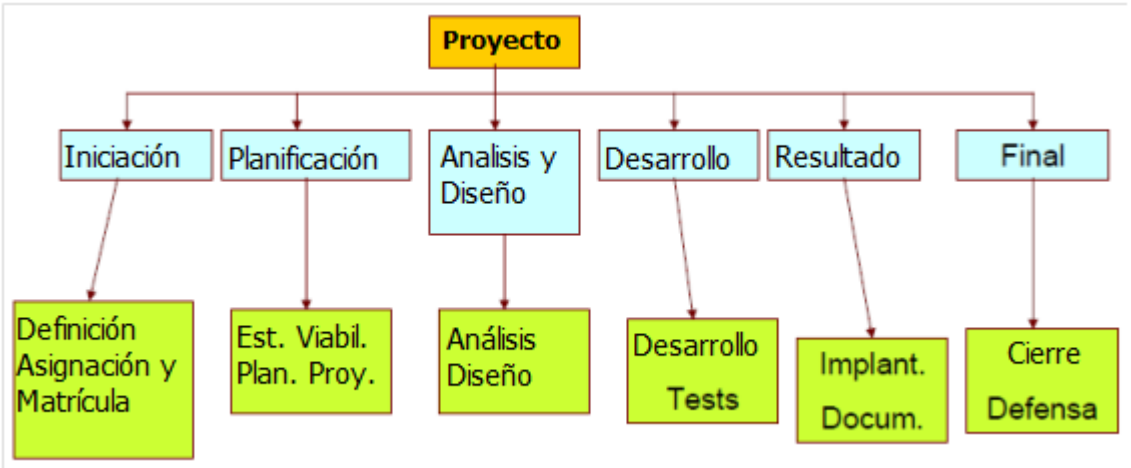

Figura 3: Diagrama WBS.

2.4.2 Recursos del Proyecto

Recursos humanos

Recursos Humanos	Valoración
Jefe del Proyecto (CP)	50€/h
Analista(A)	38€/h
Programador(P)	25€/h
Técnico de pruebas(TP)	20€/h
Diseñador Gráfico(DG)	22€/h

Tabla 9: Tabla de Recursos Humanos.

Recursos Materiales

Se usarán recursos materiales propios. El desarrollo se hará utilizando software libre, tanto Eclipse, un entorno de desarrollo gratuito, como las extensiones de AndEngine para este.

A parte se utilizará un PC con Windows 8 Pro para el desarrollo de la aplicación y un Samsung Galaxy Ace, y en ocasiones un Samsung Galaxy SII, para hacer los test.

2.4.3 Calendario de los Recursos

Los recursos humanos se usarán durante todo el proyecto:

- Jefe del proyecto: Asignación del proyecto, aprobación de las diferentes fases del proyecto y cierre del proyecto.
- Analista: Análisis y diseño, implantación y puntos de control de análisis, diseño y desarrollo.
- Programador: Diseño, desarrollo y test. Parcialmente en la implantación.
- Técnico de pruebas: Fase de test.
- Diseñador Gráfico: Diseño. Diseñar las imágenes y texturas usadas para crear, objetos, menús, etc.

Los recursos materiales se utilizarán principalmente en las fases de desarrollo, test e implantación.

2.4.4 Calendario del Proyecto

Todas las fases se desarrollan usando un modelo lineal. Por lo tanto, cada fase no empezará hasta que la fase anterior no se haya completado.

En la fase de desarrollo se prevé un modelo ágil de tal manera que el diseño, el desarrollo y el test sigan un modelo iterativo

La fase de generación de documentos será al final porque incluirá los documentos elaborados durante el desarrollo del proyecto, inicio, estudio de viabilidad y planificación del proyecto.

2.4.5 Planificación Detallada

	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	**Proyecto**	**153,75 days**	**Wed 05/10/11**	**Fri 23/03/12**		
2	Inicio del proyecto: asignación y matriculación del pro	2 hrs	Wed 05/10/11	Wed 05/10/11		Jefe de Proyecto;Director de Proyecto[10%]
3	**Planificación**	**20 days**	**Mon 14/11/11**	**Mon 28/11/11**		
4	Estudio de viabilidad	20 hrs	Mon 14/11/11	Wed 16/11/11	2	Jefe de Proyecto
5	Aprobación Estudio Viabilidad(milestone)	1 hr	Thu 17/11/11	Thu 17/11/11	4	Jefe de Proyecto;Director de Proyecto[10%]
6	Plan del proyecto	39 hrs	Mon 21/11/11	Fri 25/11/11	5	Jefe de Proyecto
7	Aprobación Plan del proyecto	1 hr	Mon 28/11/11	Mon 28/11/11	6	Jefe de Proyecto;Director de Proyecto[10%]
8	**Análisis de la Aplicación**	**8,5 days**	**Mon 28/11/11**	**Thu 15/12/11**		
9	Análisis de requisitos(casos de uso)	10 hrs	Mon 28/11/11	Wed 30/11/11	7	Analista
10	Análisis de datos(base de datos)	5 hrs	Wed 30/11/11	Thu 01/12/11	9	Analista
11	Análisis de la seguridad y la legalidad	5 hrs	Fri 02/12/11	Mon 12/12/11	10	Analista
12	Análisis de elements gráficos	10 hrs	Mon 12/12/11	Wed 14/12/11	11	Analista
13	Documentación del Análisis	3 hrs	Wed 14/12/11	Thu 15/12/11	12	Analista[50%];Diseñador Gráfico[50%]
14	Aprobación del Análisis (milestone)	1 hr	Thu 15/12/11	Thu 15/12/11	13	Analista[50%];Jefe de Proyecto;Director de Proyecto[10%]
15	**Diseño de la Aplicación**	**9,93 days**	**Thu 15/12/11**	**Fri 30/12/11**		
16	Diseño de la base de datos	10 hrs	Thu 15/12/11	Tue 20/12/11	14	Analista[80%];Programador[20%]
17	Diseño modular de la aplicación	3 hrs	Thu 15/12/11	Fri 16/12/11	14	Analista[80%];Programador[20%]
18	Diseño de la interficie, ayuda en línea	20 hrs	Thu 15/12/11	Thu 22/12/11	14	Analista[80%];Programador[20%]
19	Diseño gráfico(objetos del juego, escenarios)	13,33 hrs	Thu 22/12/11	Wed 28/12/11	14	Analista[40%];Programador[30%];Diseñador Gráfico
20	Diseño de las pruebas	5 hrs	Wed 28/12/11	Thu 29/12/11	19;17;18;16	Analista[60%];Programador[20%];Técnico de pruebas[20%]
21	Documentación del diseño	3 hrs	Thu 29/12/11	Fri 30/12/11	20	Analista
22	Aprobación del diseño	0,95 hrs	Fri 30/12/11	Fri 30/12/11	21	Analista[50%];Jefe de Proyecto;Director de Proyecto[10%]
23	**Desarrollo de la aplicación**	**31,25 days**	**Fri 30/12/11**	**Tue 14/02/12**		
24	Preparación del entorno de desarrollo	25 hrs	Fri 30/12/11	Tue 10/01/12	22	Programador
25	Configuración de la base de datos	20 hrs	Tue 10/01/12	Tue 17/01/12	24	Programador
26	Módulo de adquisición de datos y funcionalidades	100 hrs	Tue 10/01/12	Tue 14/02/12	24	Programador
27	Desarrollo de interficie del usuario	100 hrs	Tue 10/01/12	Tue 14/02/12	24	Programador
28	**Test y pruebas**	**8,5 days**	**Tue 21/02/12**	**Mon 05/03/12**		
29	Prueba untiárias	15 hrs	Tue 21/02/12	Mon 27/02/12	25;26;27	Programador[50%];Técnico de pruebas[50%]
30	Pruebas de integración	15 hrs	Mon 27/02/12	Fri 02/03/12	29	Programador[10%];Técnico de pruebas[90%]
31	Documentación de desarrollo y test	3 hrs	Fri 02/03/12	Mon 05/03/12	30	Programador
32	Aprobación del desarrollo y pruebas(milestone)	1 hr	Mon 05/03/12	Mon 05/03/12	31	Jefe de Proyecto[50%];Analista;Programador[25%];Director de Proyecto[10%]
33	**Implantación**	**3,75 days**	**Mon 05/03/12**	**Fri 09/03/12**		
34	Instal·lación	5 hrs	Mon 05/03/12	Tue 06/03/12	32	Analista[70%];Programador[30%]
35	Pruebas reales	10 hrs	Tue 06/03/12	Fri 09/03/12	34	Analista[40%];Programador[40%];Técnico de pruebas[20%]
36	Generación de documentos(memoria del proyecto)	30 hrs	Fri 09/03/12	Tue 20/03/12	35	Jefe de Proyecto
37	Cierre del proyecto	1 hr	Tue 20/03/12	Tue 20/03/12	36	Jefe de Proyecto;Director de Proyecto[10%]
38	Defensa del proyecto	24 hrs	Fri 16/03/12	Fri 23/03/12		Jefe de Proyecto

Figura 4: Planificación detallada.

2.4.6 Planificación Temporal

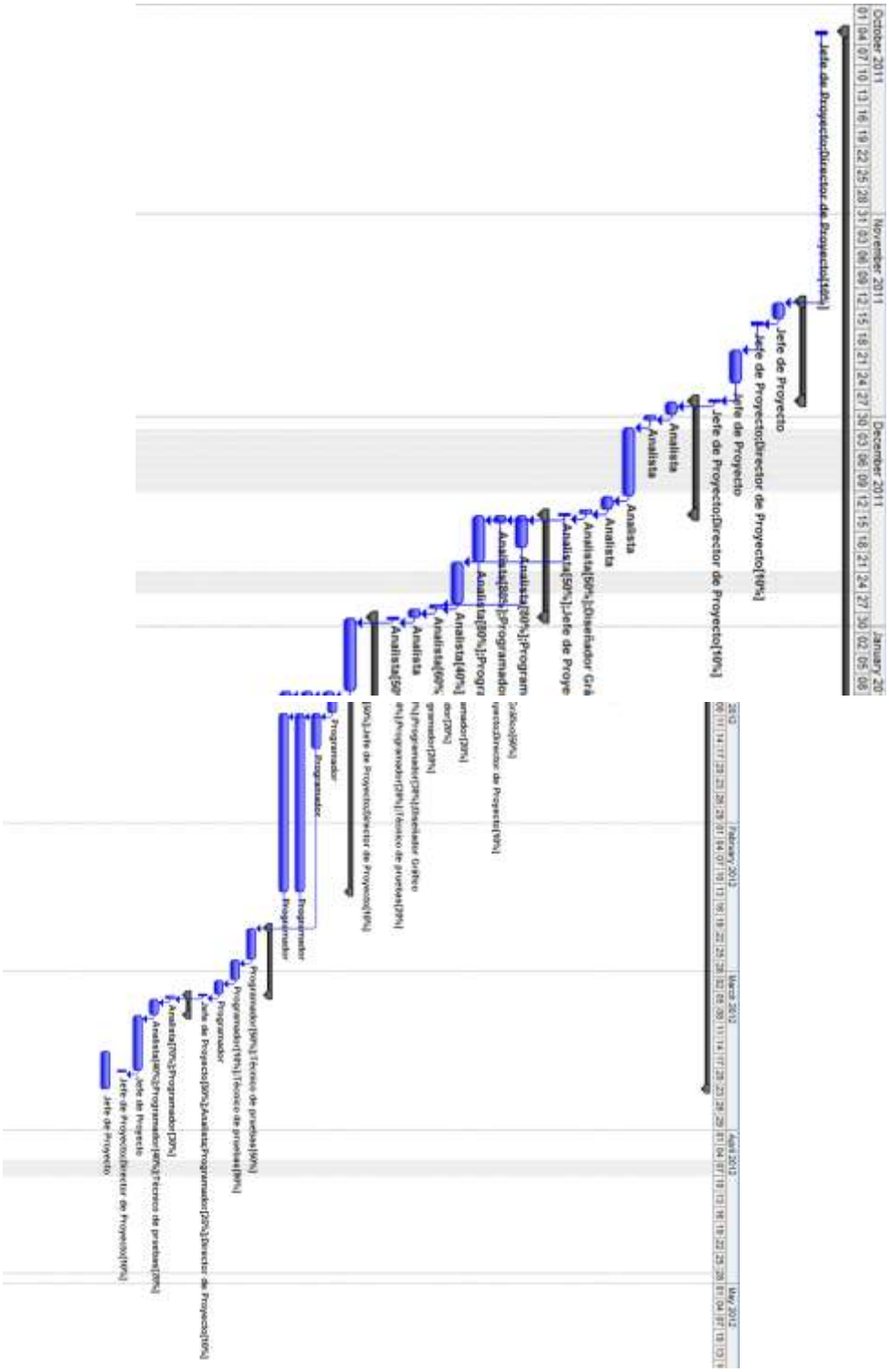

Figura 5: Diagrama de Gantt

2.5 Evaluación de Riesgos

La evaluación de riesgos siempre es una de las partes más importantes antes del inicio de un proyecto. Se han de definir muy bien los riesgos que nos podemos encontrar a lo largo del proyecto.

2.5.1 Lista de Riesgos

R1. Falta de alguna tarea necesaria: Plan de proyecto. No se cumplen los objetivos del proyecto.

R2. Presupuesto mínimo: Plan de proyecto. Pérdidas económicas y de calidad.

R3. Planificación temporal optimista: Plan de proyecto. No se acaba en la fecha prevista, aumentan los recursos.

R4. Cambio de requisitos: Estudio de viabilidad, análisis. Retraso en el desarrollo y el resultado.

R5: Fase de test incorrecta: Desarrollo, implantación. Falta de calidad, deficiencias en la operativa, insatisfacción de usuarios, pérdida económica.

R6. Falta de adopción de medidas de seguridad: Estudio de viabilidad, análisis, desarrollo. Pérdida de información, incumplimiento legal, pérdidas económicas.

R7. Incumplimiento de alguna norma, reglamento o legislación: En cualquier fase. No se cumplen los objetivos, repercusiones legales.

R8: Herramientas de desarrollo inadecuadas: Desarrollo. Retraso en la finalización del proyecto, empeoramiento de calidad.

R9: Equipo del proyecto demasiado reducido: Plan de proyecto. Retraso en la finalización del proyecto, no se cumplen los objetivos.

R10: Abandono del proyecto antes de su finalización: En cualquier fase. Pérdidas económicas, frustración.

2.5.2 Catalogación de Riesgos

	Probabilidad	**Impacto**
R1	Alta	Crítico
R2	Alta	Crítico
R3	Alta	Crítico
R4	Alta	Marginal
R5	Media	Crítico
R6	Alta	Crítico
R7	Media	Crítico
R8	Baja	Crítico
R9	Media	Crítico
R10	Media	Catastrófico

Tabla 10: Tabla de Catalogación de riesgos.

2.5.3 Plan de Contingencia

	Solución que adoptar
R1	Revisar el plan del proyecto, modificar la planificación.
R2	Renegociar con el cliente, afrontar posibles pérdidas, hacer un seguro, reciclaje de elementos de software libre.
R3	Aplazar alguna funcionalidad, afrontar posibles pérdidas, hacer un seguro, proponer condiciones de finalización.
R4	Renegociar con el cliente, aplazar una funcionalidad, modificar la planificación y presupuesto
R5	Diseñar los test con antelación, realizar test automáticos, negociar contrato de mantenimiento, dar garantías.
R6	Revisar la seguridad en cada fase, aplicar políticas de seguridad activas.
R7	Revisar las normas y legislación, consultar un experto, afrontar posibles repercusiones penales.
R8	Mejorar la formación del equipo, prever herramientas alternativas, mejorar la calidad.
R9	Pedir un aplazo, negociar con el cliente, afrontar pérdidas, contratación de más personal.
R10	No tiene solución.

Tabla 11: Tabla de Plan de contingencia.

2.6 Presupuesto

2.6.1 Estimación Coste Personal

Jefe de Proyecto	120h	6000€
Analista	85h	3230€
Programador	273.43h	6835.75€
Diseñador Gráfico	24h	528€
Técnico de pruebas	14.83h	296.6€
TOTAL	517.26h	16890.35€

Tabla 12: Tabla de Estimación de coste personal.

2.6.2 Estimación coste de recursos

	Coste Amortización
PC Programador	600€
Microsoft Office	119€
Microsoft Project	360€
TOTAL	1079€

Tabla 13: Tabla de Estimación de coste de recursos.

2.6.3 Resumen y análisis coste – beneficio

Coste de desarrollo del Proyecto 16942.34€

Coste de amortización del material 1079.00€

Total: 17969.35€

2.6.4 Conclusiones

Tras realizar todos los análisis y estudios necesarios, considero que la realización del proyecto es viable ya que, si bien implicará hacer la labor de todo el equipo de desarrollo antes expuesto con el coste tanto en tiempo como económico que eso teóricamente supondría, al no ser este un proyecto real y disponer ya de todas las herramientas y recursos necesarios o ser estos gratuitos y ser el objetivo de este proyecto sobretodo el aprendizaje, y no la creación de un producto de prestaciones comerciales, concluyo que el proyecto es, efectivamente, viable.

3. Recursos teóricos utilizados

3.1 Palabras Clave

- **Motor Gráfico**: El motor gráfico es parte del programa que controla, gestiona y actualiza los gráficos (en 2D o 3D) en tiempo real.
- **Eclipse**: Entorno de programación que nos permite programar, compilar , depurar y ejecutar en Java.
- **Compilar**: En programación, llamamos compilar a traducir un lenguaje de alto nivel(en este caso java) a código absoluto o lenguaje binario o máquina.
- **Sprite**: Consiste en un mapa de bits 2D que se dibujan directamente en un destino de representación(pantalla) sin usar canalización de transformaciones, iluminación o efecto. Por lo que requieren poco uso de CPU.
- **CPU**: Unidad de Procesamiento Central. Interpreta las instrucciones y procesa los datos contenidos en los programas.
- **Multitouch**: La tecnología multitouch(o multitáctil) consiste en una pantalla táctil que es capaz de reconocer simultáneamente múltiples puntos de contacto.
- **Frame**: Se trata de una imagen particular en una sucesión de imágenes.

3.2 Lenguajes de programación utilizados

Para el desarrollo de esta aplicación usaremos Java. En otras plataformas normalmente podemos elegir lenguaje de programación ya que herramientas como, por ejemplo, Unity 3D, otro motor gráfico, permite la utilización de Java, C/C++, Python, etc. y hasta un lenguaje de scripting propio del motor. Pero para este juego nos vemos obligados a trabajar en Java, ya que Android sólo soporta dicho lenguaje.

Cabe mencionar, que si bien antes había visto y ejecutado algunos pequeños aplicativos en Java, nunca antes me había embarcado en un proyecto de estas dimensiones utilizando este lenguaje. Java tiene detrás una compañía como es Oracle, y es uno de los lenguajes más extendidos hoy día, lo cual implica se puede encontrar mucha documentación y ayuda mediante diferentes vías. Además su sintaxis deriva mucho de C y C++, aunque no tiene tantas facilidades de bajo nivel, lo cual ayuda mucho ya que sí que conozco algo más esos lenguajes.

Junto con Java, para programar en Android se requiere el SDK de dicho sistema operativo proporcionado por Google de forma gratuita. El SDK (Software Development Kit) contiene una serie de herramientas y complementos y extensiones que nos permiten, usar componentes propios de Android, como por ejemplo ventanas de aviso, almacenar datos en memoria, etc.

3.3 Herramientas y entorno de trabajo

Inicialmente las herramientas utilizadas para llevar una planificación correcta del proyecto han sido una versión estudiante del Microsoft Project para realizar la planificación del proyecto y el diagrama de Gantt correspondiente.

El entorno de trabajo para este proyecto será un PC de sobremesa. También se usará un Galaxy Ace y Galaxy SII para compilar, probar y depurar la aplicación, debido a que aunque si bien sí

que podemos utilizar el emulador de Eclipse, éste es bastante lento y no ofrece toda la funcionalidad y operatividad que nos ofrece el dispositivo físico en sí, como por ejemplo, la función multitouch.

La herramienta utilizada para programar en Java será Eclipse Juno proporcionado por Google que viene con las herramientas del SDK de Android listas para instalar. Todo ello sobre un sistema operativo Windows 8 Pro. Tanto el SDK como Eclipse son software gratuito.

A parte de este software también utilizaremos AndEngine. Esta herramienta, creada por Nicolas Gramlich, es un conjunto de funciones y librerías gratuitas, que nos facilitarán la programación para este tipo de aplicaciones. Se encargará de gestionar el movimiento de sprites, animaciones, físicas y otra serie de aspectos que Android no incorpora.

3.4 Tecnología

AndEngine es un motor gráfico 2D que usa OpenGL ES 2 para juegos para la plataforma Android. En AndEngine se usa una terminología propia. A continuación explicaré los conceptos básicos:

- **BaseGameActivity**: El BaseGameActivity es la raíz del juego, que contiene el motor y crea la vista donde se va a dibujar todo. Hay siempre un solo Engine por cada BaseGameActivity. Si bien es cierto que se podrían crear varios, y referenciarlos apropiadamente pero en ningún caso esto será necesario.
- **Engine**: El Engine es el motor interno del juego, se encarga de ir dibujando en pantalla y actualizando objetos en la escena, frame a frame, que contiene todo el contenido que lleva el juego. Normalmente hay una escena por Engine en un mismo momento del tiempo, pero en algunos casos puede haber más de una, como por ejemplo para hacer una pantalla partida (como en el famoso Tetris). En mi caso como no necesito tener varias escenas al mismo tiempo en pantalla, simplemente voy cambiando entre escenas y mostrándolas.
- **IResolutionPolicy**: Una implementación de IResolutionPolicy interface es parte del EngineOptions. Te hace abstraerte de la resolución del terminal, tú trabajas para una resolución y el AndEngine se encarga del resto, si el juego fuese a ejecutarse en un dispositivo con una pantalla de diferente resolución AndEngine, mediante las librerías de OpenGL, escalaría la resolución más próxima a la de ese dispositivo, pero siempre manteniendo el IResolutionPolicy que indiquemos.
- **Camera**: Un objeto Camera define el rectángulo de la parte de la escena actualmente visible, no tiene porqué ser la escena completa. Normalmente hay una cámara por escena. Hay subclases específicas que permiten hacer zoom y mover la cámara suavemente. Por ejemplo, podríamos diseñar una escena de 3000px por 2000px y hacer una cámara de 200*200 que siguiera a un sprite o a tu dedo mediante un evento. De esta forma conseguiríamos un efecto como el del videojuego Pokemon en el que la cámara sigue al personaje principal cuando se desplaza.

- **Scene**: La clase Scene es el contenedor para todos los objetos que se van a dibujar en la escena, visibles o no en ese momento. Una escena puede tener Layers(capas), que son capas para ordenar objetos. Hay subclases de la Scene como CameraScene/HUD/MenuScene que tienen comportamientos específicos.
- **Entity**: Una entidad es un objeto que puede ser dibujado, como imágenes, texto, líneas, rectángulos y otras figuras. Una entidad tiene posición/rotación/zoom/color... Casi todo lo que vemos en una Scene hereda de Entity y se pueden aplicar EntityModifiers que nos permitirán aplicar ciertas acciones a dichos objetos, como por ejemplo, movimiento de un punto A a uno B, rotación...
- **ITextureRegion**: Una ITextureRegion define un rectángulo en el que guardaremos en memoria la imagen en sí, probablemente de un archivo .png por ejemplo. Antes, por temas de cómo estaba implementado AndEngine las imágenes debían ser una potencia de 2, por suerte ya no.
- **BitmapTextureAtlas**: Un BitmapTextureAtlas no es más que un contenedor para ITextureRegion's, si bien no es necesario usarlo ya que además añade cierta complejidad a la hora de programar, sí es cierto que mejora el rendimiento. A grandes rasgos lo que hace este objeto es declarar un rectángulo en el que podremos encajar diferentes imágenes permitiendo así al motor cargar todas las imágenes de una sola vez.
- **IUpdateHandler**: Este elemento es un objeto que se puede asignar o bien al motor del juego o bien a la escena. En cualquiera de los casos su función es ejecutar cada vez que se cambie de frame, un fragmento de código de tu elección.

Todas las actividades que hereden de **BaseGameActivity** tendrán 4 funciones y seguirán esta estructura básica:

```
public class GameActivity extends BaseGameActivity
{
    public EngineOptions onCreateEngineOptions()
    {
        return null;
    }
```

Esta es la primera función que se ejecuta cuando cargamos el activity. En ella definiremos el objeto camera, sus opciones, las opciones del propio engine, por ejemplo si queremos habilitar multitouch, el sonido o música para el juego, y devolveremos el engine.

```
public void onCreateResources(OnCreateResourcesCallback
pOnCreateResourcesCallback) throws IOException
    {

    }
```

En la función onCreateResources es donde se lleva a cabo toda la reserva de memoria de los sprites, sonidos y los tipos de fuentes para los textos que vayamos a utilizar. En mi caso, yo he diseñado una clase manager que sería llamada una sola vez desde aquí a la que le pasamos el engine, la camera, el contexto, es decir la actividad de Android y un objeto VertexBufferObjectManager, que como su nombre indica no es más que un almacén en el que guardaremos momentáneamente(de forma totalmente automática) los datos necesarios para las diferentes operaciones que realicemos.

```
public void onCreateScene(OnCreateSceneCallback
pOnCreateSceneCallback) throws IOException
    {

    }
```

En la función onCreateScene es donde se dibuja la escena que se mostrará en pantalla. Aquí se crean entidades tipo Background, Sprite, Text, AnimatedSprite, etc.

```
public void onPopulateScene(Scene pScene, OnPopulateSceneCallback
pOnPopulateSceneCallback) throws IOException
    {

    }
```

Por último, tenemos la función onPopulateScene. Bien, inicialmente esta función no era ni es necesaria, fue incorporada más tarde por Nicolas Gramlich ya que la mayoría de los juegos tienen una pantalla de presentación(SplashScreen) y esta función simplemente hacía más cómoda su implementación. Su uso típico se trata de crear la escena tras haber creado la escena de SplashScreen, podríamos verlo como un segundo onCreateScene.

En el caso de los sprites y textos les indico en qué posición de la escena deberán estar. A la hora de crear objetos del tipo sprite también he de indicar cuáles serán sus texturas y en el caso de los textos sus fuentes (que previamente tendremos que haber definido en la función onCreateResources).

Después de haber creado estos objetos, AndEngine nos proporciona un seguido de funciones para trabajar con ellos. Por ejemplo la función registerTouchArea(sprite1) registra el área ocupada por el sprite y cuando el usuario toca dicha área esta lanza un evento o interrupción. A mí se me permite escribir el código que desee que se ejecutará cuando esta interrupción sea lanzada, esto puede servir por ejemplo para arrastrar un sprite por la pantalla siguiendo el movimiento del dedo.

Además de esto, tenemos funciones como setColor() o setText() para los objetos de tipo texto, scale(tamaño) para los sprites o animate(numeroDeRepeticiones, tiempoDeAnimación) para los sprites animados.

4. Diseño

4.1. Clases

A continuación explicaremos con más detalle la funcionalidad y el objetivo de cada clase. En Android cada pantalla que vislumbremos es una clase de tipo *Activity**. Cuando se lanza una actividad equivale a lanzar una nueva ventana en Windows. Además de esto, el motor gráfico que usamos (AndEngine) nos permite tener en una misma Activity diferentes *Scenes*. Esto equivale a una misma ventana en Windows pero que disponga de diferentes visualizaciones.

Todas las clases de tipo Activity tendrán las funciones onCreateEngineOptions, onCreateResources, onCreateScene y onPopulateScene proporcionadas por AndEngine, que se encargarán de inicializar los datos de la cámara, cargar todos los Sprites e imágenes necesarias y crear la Scene principal de la Activity. Una actividad puede tener más de una escena. Es más, no es recomendable crear más de una actividad para todo el juego, ya que el cambio entre actividades es más lento que el cambio entre escenas.

4.1.1. Clase GameActivity

Esta es la clase principal del juego, aunque realmente no hace mucho porque he intentado utilizar siempre un modelo singleton, es decir que desde aquí sólo llamamos a instancias de otras clases que son las que realmente hacen el trabajo. A pesar de lo mencionado, hay cosas que sí se deben hacer aquí, como la ejecución de las funciones descritas más arriba. También, al ser esta la actividad real que ejecutará Android, tendremos que gestionar aquí todo lo relacionado con el sistema operativo. En mi caso aquí configuro a mi gusto el sistema de pausa de la aplicación ya que el ciclo de vida de una aplicación de Android no me conviene a la hora de pausar mi juego. Este es el ciclo de vida de una actividad en Android:

Activity: Cada uno de estos elementos supone una interacción con el usuario (generalmente una ventana), y se corresponde con una clase que hereda de la clase Activity.

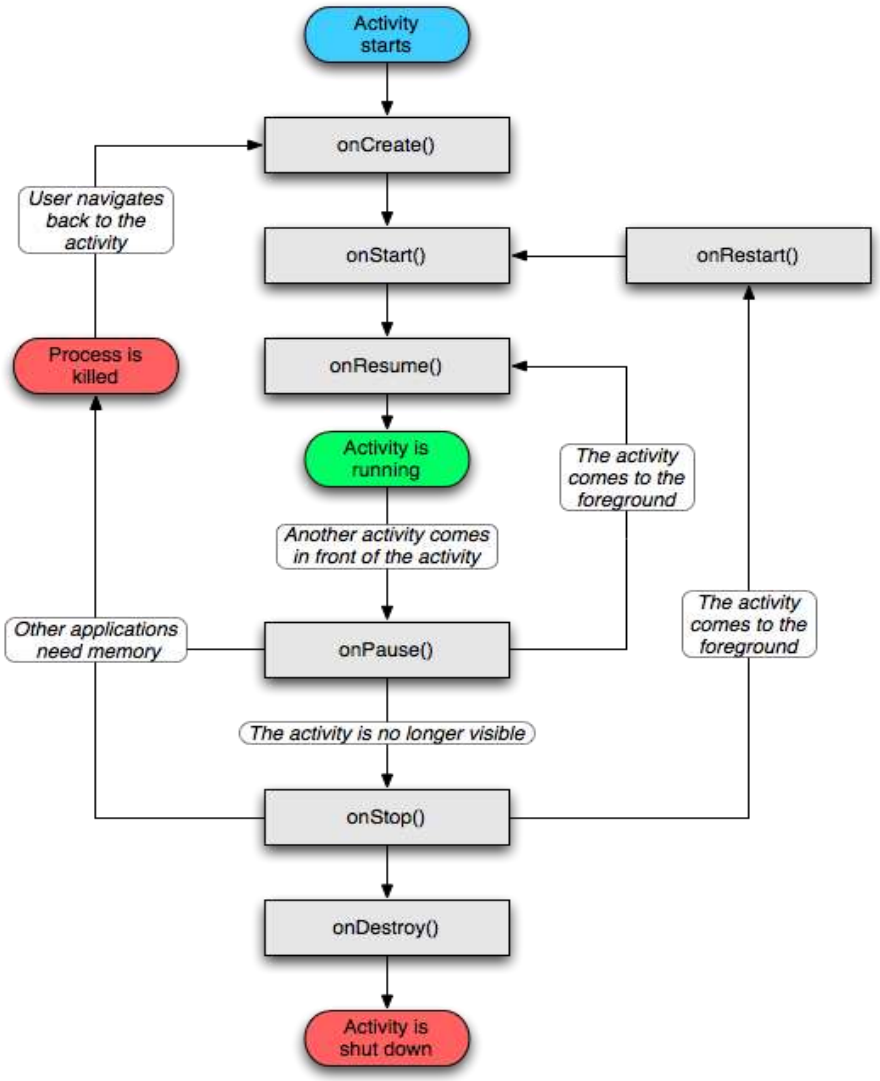

Figura 6: Ciclo de una actividad en Android.

Por tanto, modifico el onPause() genérico de Android por uno que realmente pause el juego ya que el engine no viene configurado así por defecto.

4.1.2. Clase BaseScene

Esta clase es muy simple, lo único que hace es heredar de la clase Scene que viene ya predefinida por AndEngine y añade funciones abstractas creadas por mí que heredarán las escenas que utilizaremos en el juego. Su única función es proporcionar algo más de comodidad a la hora de programar.

4.1.3. Clase Cooldown

Esta clase lo único que hará será, como su nombre indica, proporcionarnos un sistema de reutilización de un objeto, en mi caso la uso para determinar con qué cadencia podemos disparar proyectiles.

4.1.4. Clase ResourcesManager

Esta clase es la que se encargará realmente de cargar los recursos necesarios según se vayan necesitando y de liberarlos una vez no sean necesarios. Se ha intentado implementar un patrón singleton de manera que cada vez que necesitemos crear algún sprite o cualquier operación que requiera de recursos externos como imágenes, sonidos o fuentes, esta clase será instanciada para su uso específico.

4.1.5. Clase SceneManager

Aquí como en el caso anterior, he aplicado el mismo patrón, cada vez que necesitemos gestionar un cambio de un tipo de escena a otro instanciaremos a SceneManager y esta se encargará de crear la nueva escena, borrar la anterior y liberar el espacio que esta ocupase en memoria. A parte también tendrá algunos métodos que nos permitirán obtener cierta información de la escena actualmente mostrada en pantalla.

4.1.6. Clase ObtaclePool/ProjectilePool/RewardPool/TargetsPool

Este conjunto de clases tendrán la misma función las unas con las otras con la única excepción de los objetos que emplearán. La mecánica de la clase es crear un sistema mediante el cual podamos llamar a la clase y esta nos proporcione un objeto de cierto tipo, por ejemplo TargetsPool nos proveerá de un sprite animado que será una nave enemiga. Dicho objeto tras ser usado(ya sea porque ha salido de la escena o porque ha sido destruido) será reciclado y puesto de nuevo a disposición de cualquiera que llame a la clase. Con este método lo que conseguimos es que en lugar de estar creando y destruyendo objetos simplemente los reubicamos, lo cual supone una mejora de rendimiento.

4.1.7. Clase MainMenuScene

Esta clase y todas las que sigan a partir de ahora serán "clases escena", esto es, que su función será sencillamente la de dibujar algo en la pantalla utilizando los recursos que les proporcionará

el ResourcesManager. Todas estas clases heredarán de la clase previamente descrita, BaseScene. Cambiaremos entre ellas usando el SceneManager.

Esta clase en concreto será la encargada de mostrar el menú principal, por el que podremos navegar con libertad para poder iniciar la partida, entrar al menú de opciones o salir del juego.

4.1.8. Clase FirstOptionsSubMenu/DifficultyMenuScene

Estas dos clases tienen la misma funcionalidad. Por desgracia el sistema de menú que nos da AndEngine en mi opinión está lejos de ser óptimo, ya que debido a que el motor trabaja con escenas, el hecho de cambiar de una parte del menú a otra(submenús) si bien de cara al usuario no supondrá ningún inconveniente, para el desarrollador sí que lo hará ya que tendrá que programarlo como si fuesen escenas diferentes que heredan del menú padre.

La primera, FirstOptionsMenú, será la escena en la que tengamos la primera lista de opciones, en mi caso será dificultad y un botón de volver al menú principal.

La segunda será la escena hija de la anterior, en la que se podrá seleccionar el nivel de dificultad en el que deseemos jugar.

4.1.9. Clase GameScene

Esta clase será la más importante, se ocupará de dibujar y gestionar todo lo que pase en la escena de juego. Por tanto, será la que contenga la mayor parte de la lógica del juego y la que cree todos los objetos del juego y los utilice. Aquí crearemos un IUpdateHandler que será el que gestione las colisiones, el que se modifique el número de vidas que nos quedan y actualice los puntos que vamos consiguiendo al eliminar enemigos.

4.1.10. Clase HighScoreScene

Esta escena sencillamente nos mostrará los 3 mejores resultados, previamente almacenados, que hayamos obtenido a lo largo del tiempo al completar un nivel y nos permitirá pasar al siguiente.

4.1.11. Clase LoadingScene

Nos creará una escena intermedia entre las escenas de juego y el menú principal tras pulsar el botón de Play o al volver del juego al menú pulsando el botón de Back del dispositivo. Todo mientras se cargan los recursos necesarios para mostrar la escena siguiente, como alternativa a esperar en una pantalla en negro.

4.1.12. Clase SplashScene

Esta clase nos mostrará la primera escena que veremos siempre al entrar al juego, sólo nos mostrará un logo con el nombre del juego durante unos segundos y desaparecerá para dar paso al menú principal.

Por supuesto, como he hecho que cada escena sea también una clase independiente eso implica que pueden ser instanciadas varias veces con resultados diferentes. En el caso de GameScene, esta clase se usaría para todos los niveles y nos permite cambiar detalles del nivel en concreto en el que estemos(dificultad, enemigos, escenario…) con muy poco esfuerzo. O nos permitiría crear diversas pantallas de carga distintas sin necesidad de crear una escena totalmente nueva.

El diagrama de flujo que seguirá el juego será de la siguiente manera:

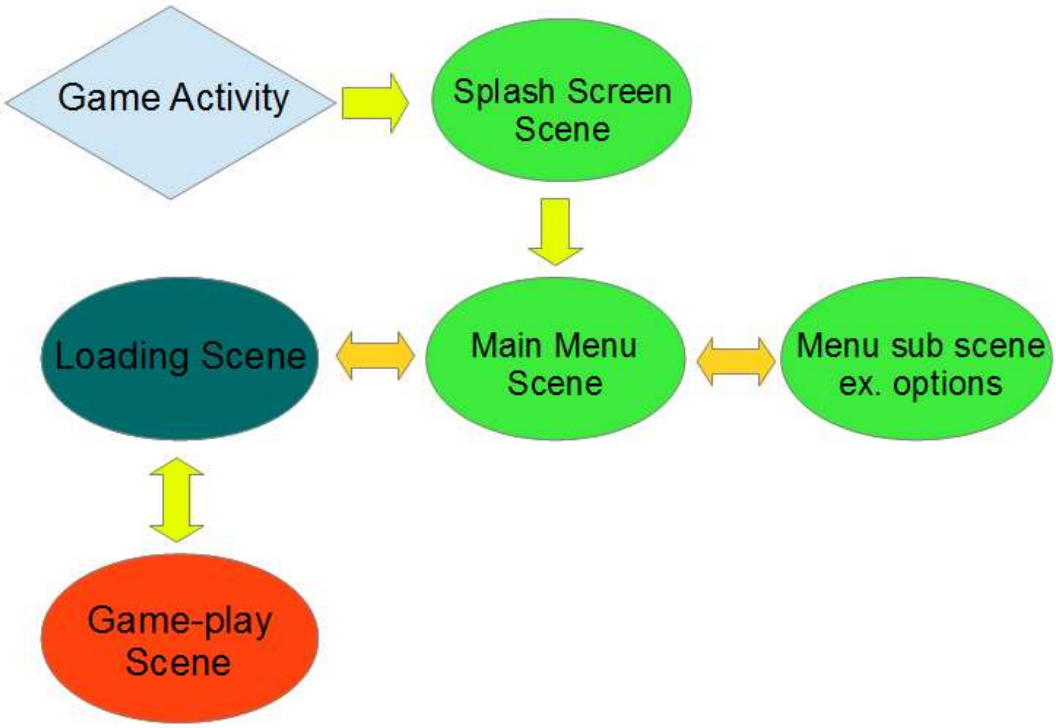

Figura 7: Diagrama de flujo del juego.

4.2. Diseño de la Base de Datos

El diseño de la base de datos es el último de los puntos del diseño. Aunque lo estoy llamando bases de datos no se trata de bases de datos convencionales, utilizando SQL por ejemplo. Si no que Google ha diseñado para Android un sistema llamado SharedPreferences. La clase SharedPreferences nos proporciona un framework general que nos permite guardar y recuperar pares de tipos de datos primitivos. Se puede usar SharedPreferences para guardar booleans, floats, ints, longs y strings. Estos datos guardados perdurarán a través de todas las sesiones que el usuario inicie de esa aplicación(incluso si la aplicación se destruye).

Para obtener un objeto de la clase SharedPreferences. Utilizaremos unos de estos dos métodos:

- getSharedPreferences() - Usado si se necesitan varios archivos identificados por nombre en los que almacenar datos.
- getPrecferences() – Usado si tan sólo necesitas un archivo de preferencias en el que almacenar datos de tu aplicación.

Para escribir valores:

1- Llamamos a edit() para obtener un SharedPrefences.Editor.
2- Añadimos los datos con métodos como putBoolean() y putString.
3- Y confirmamos que queremos introducir los datos con un commit().

Para leer valores utilizamos métodos como getBoolean() y getString().

Como se puede observar la ventaja que tiene sobre el uso bases de datos tradicionales es que es mucho más simple que estas. Pero la desventaja es que sólo nos permite relacionar dos datos entre ellos y no nos permite hacer búsquedas compuestas ni crear tablas de gran tamaño, etc. Es decir, las bases de datos son mucho más potentes que este sistema. Pero en mi caso que sólo tenemos que almacenar un poco de información por cada nivel nos sirve perfectamente.

Los datos que almacenaremos al crear el objeto SharedPreferences inicialmente serán:

```
editor.putBoolean("INITIALIZED", true);
editor.putInt("Level1HighScore1", 0);
editor.putInt("Level1HighScore2", 0);
editor.putInt("Level1HighScore3", 0);
editor.putInt("Level2HighScore1", 0);
editor.putInt("Level2HighScore2", 0);
editor.putInt("Level2HighScore3", 0);
editor.putInt("Difficulty", 2);
editor.putInt("CurrentLevel", 1);
editor.putInt("CurrentLifes", 3);
```

Al principio creamos una entrada que comprobaremos cada vez que entremos al juego para saber si hemos inicializado la base de datos. Es decir lo anteriormente descrito sólo se ejecutará al instalar la aplicación. Luego todos los datos se actualizarán a tiempo real.

5. Funcionamiento de la aplicación

5.1. Descripción

El funcionamiento básico de la aplicación se divide en dos partes:

- Una parte estática dedicada a la presentación del logo del juego y la selección de dificultad en el menú de opciones.
- Una segunda parte donde el jugador destruirá a las naves enemigas que se presenten intentando no impactar contra ellas ni contra los meteoritos que se interpongan en su trayectoria. Tambíen intentando reunir las esferas de energía que le proporcionarán un pequeño bonus de puntos. El nivel se completará si el jugador consigue no ser destruido dentro del límite de tiempo, ya que cada vez que lo haga perderá una vida y sólo dispondrá de tres para superar el juego.

5.2. Parte Estática

5.2.1. Pantalla de presentación.

En esta primera pantalla veremos como aparece progresivamente el logotipo del juego y tras unos segundos se atenúa hasta desaparecer.

Figura 8: Pantalla presentación.

5.2.2. Menú principal y de opciones

Desde la pantalla de presentación iremos al menú de opciones, en esta pantalla podremos, a parte de iniciar una partida, navegar por el menú de opciones y marcar la opción de dificultad en la que queramos jugar. Tendremos para elegir entre fácil, normal o difícil. Si seleccionásemos la opción de Play iniciaríamos una partida. En caso de ser la primera vez que ejecutamos el juego o en caso de haber dejado una partida a medias automáticamente continuaremos desde el último nivel en el que salimos de la aplicación y con el número de vidas que nos quedasen.

Figura 9: Pantalla de menú principal.

Figura 10: Pantalla de menú de opciones.

Figura 11: Pantalla de opciones de dificultad.

5.3. Parte Dinámica

Aquí estará la parte del juego en la que podremos jugar realmente.

5.3.1. Empezar juego

En esta pantalla podremos hacer dos movimientos básicos desplazar nuestra nave de forma vertical para esquivar tanto las naves enemigas como los meteoritos o recoger esferas de energía y disparar a las naves enemigas ya sea para sumar puntos como para no impactar contra ellas y morir. Esta parte tendrá a su vez dos partes también:

5.3.1.1. Pantalla de juego

En esta parte se desarrolla toda la mecánica anteriormente descrita, aquí podremos disparar a las naves enemigas que surgirán del borde derecho de la escena y se dirigirán al izquierdo. Para disparar sencillamente tocamos la pantalla en el punto a donde queramos dirigir el proyectil láser y este se moverá en línea recta hacía el punto marcado. Para mover nuestra nave tocaremos el sprite y lo arrastraremos a la posición que deseemos dentro del mismo eje vertical.

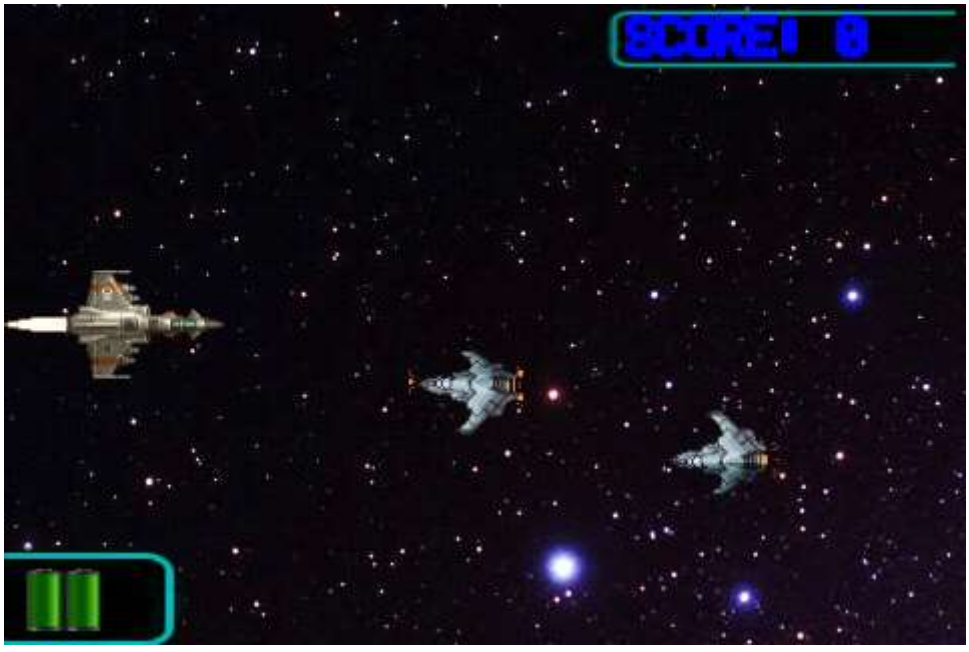

Figura 12: Pantalla de juego.

5.3.1.2. Visualización de resultados

Después de transcurrido un tiempo que irá incrementando según el nivel y la dificultad nos aparecerá en pantalla un mensaje que nos indicará que hemos completado el nivel y tras unos segundos iremos a una pantalla de clasificación en la que podremos observar nuestras tres mejores puntuaciones en ese nivel.

Figura 13: Pantalla de visualización de resultados.

5.4. Puesta en marcha

Se ha creado una interfaz muy intuitiva con la finalidad de poder usar la aplicación sin necesidad de manuales.

Las aplicaciones en AndEngine tienen el problema de que el emulador de Eclipse actualmente no funciona demasiado bien y no permite hacer ciertas cosas. Esto implica que todas las prueba se deben hacer en algún dispositivo real con Android. El móvil ha sido un Galaxy Ace y el juego no da errores pero sí un problema. El problema en cuestión es que tras cambiar varias veces entre escenas en el móvil transforma todos los sprites de la pantalla en rectángulos de colores y pasados unos instantes la aplicación se cierra.

A continuación detallaré las características básicas del dispositivo en el que se han realizado la mayoría de pruebas y se ha depurado el código:

- Pantalla de 3.5 pulgadas y 480 pixeles de alto por 320 pixeles de ancho.
- Dimensiones 112.4 x 59.9 x 11.5 mm.
- 2GB de almacenamiento externo, 158MB de interno y 278MB de memoria RAM.

- Procesador Qualcomm MSM7227 800MHz, GPU Adreno 200.
- TFT touchscreen capacitativo, 16M colores(permite multitouch).
- Bateria Standard, Li-Ion 1350 mAh.
- Android OS, v2.3.6 Gingerbread.

La instalación de la aplicación no requiere de la atención del usuario y la gestión de los datos se hace automáticamente en segundo plano.

6. Test y pruebas

Inicialmente el proyecto ha estado desarrollado con un pc con Microsoft Windows 8 Pro, con el entorno de programación Eclipse instalado. La aplicación está pensada para dispositivos que usen Android y por ese motivo se han hecho todas las pruebas sobre un Galaxy Ace, que tiene el sistema Android nativo distribuido por Google, sin ningún tipo de modificación hecha por los fabricantes (como es el caso de la gran mayoría de móviles de LG, Samsung, HTC...)

Mi opinión es que el dispositivo dispuesto para las pruebas es lo que ahora se considera un teléfono de gama baja y no ha dado problemas de rendimiento o de sobrecalentamiento. Esto es bastante bueno teniendo en cuenta que los smartphones que se venden a día de hoy son igual o por lo general mucho más potentes que el Galaxy Ace y por tanto deducimos que si el juego ha funcionado bien en este dispositivo con tan pocas prestaciones así debería de ser con otros dispositivos mejores. Visto esto podríamos decir que los requisitos mínimos para la aplicación serían los siguientes:

- 10MB de almacenamiento interno como mínimo y 158MB de memoria RAM.
- Procesador Qualcomm MSM7227 800MHz o mejor.

Las pruebas realizadas se han basado en usar la aplicación en dos dispositivos diferentes(Samsung Galaxy Ace 2.3.6 GingerBread modificada y Samsung Galaxy SII con Android 4.1.2 JellyBean), ya que es el número de dispositivos a mi alcance. También mediante el uso del emulador de Eclipse podemos realizar pruebas emulando diferentes versiones de Android y configurando el Android Virtual Device Manager para que tenga las mismas características que ciertos dispositivos físicos de los que no disponemos.

Los smartphones configurados y emulados son: Nexus S con Android 4.0.3, Samsung Galaxy SII con Android 2.3.7 y 4.0.1 y HTC Desire HD 2.2 y en ninguno de ellos ha habido errores de ejecución. El único problema ha sido en mi propio terminal, quizás por ser una versión modificada de Android. Aunque haciendo una búsqueda por la red sobre el problema al parecer es un asunto que tiene que ver con el modelo del hardware usado, a la que no se ha dado solución por parte del desarrollador de AndEngine.

El juego está pensado para una pantalla con una resolución de 480x320. Para resoluciones mayores como en el otro dispositivo físico en el que lo probé(Galaxy SII) observé que el propio AndEngine usando OpenGL redimensionaba todo(tanto la escena como los objetos en ella dibujados) dejando unos bordes negros y para resoluciones menores no se han hecho pruebas, pero en este caso también se debería redimensionar la pantalla para adaptarse, aunque en mi opinión no es recomendable jugar en esas resoluciones debido a que ya con la resolución de 480x320 resulta bastante más complicado jugar que en resoluciones mayores en las que los propios dedos del jugador no entorpecen tanto al ser la pantalla algo más grande.

Samsung Galaxy Ace

Prueba	Resultados
Carga de la Splashscreen.	La pantalla de presentación aparece lentamente y se va apagando poco a poco hasta dar paso al menú de forma correcta. Funcionan los efectos.
Carga del juego.	Cuando se pulsa el botón Play se crea una nueva partida sin problemas y todo se pone en movimiento sin ralentizaciones.
Navegar por los menús.	Ningún problema o fallo.
Eliminar enemigos y mover la nave.	No hay errores, los sprites de los enemigos desaparecen para ser sustituidos por una animación de explosión y se puede mover la nave al mismo tiempo que se dispara.
Superar nivel.	Tras un mensaje y una breve espera se pasa a la pantalla de clasificación y se puede continuar al siguiente nivel. Sin fallos.
Almacenaje de datos.	Toda la información guardada se mantiene tras completar el nivel y tras destruir la aplicación y volver a crearla. Todo correcto.

Tabla 14: Tabla de test – Samsung Galaxy Ace.

Nexus S

Prueba	Resultados
Carga de la Splashscreen.	Sin errores.
Carga del juego.	Cuando se pulsa el botón Play se crea una nueva partida sin problemas y todo se pone en movimiento sin problemas.
Navegar por los menús.	Ningún problema o fallo.
Eliminar enemigos y mover la nave.	Los sprites desaparecen y se mueven cuando deben hacerlo y las animaciones funcionan correctamente.
Superar nivel.	Todo correcto.
Almacenaje de datos.	Se conserva toda la información.

Tabla 15: Tabla de test – Nexus S.

Samsung Galaxy SII 2.3.7

Prueba	Resultados
Carga de la Splashscreen.	Sin errores.
Carga del juego.	Cuando se pulsa el botón Play se crea una nueva partida sin problemas y todo se pone en movimiento sin problemas.
Navegar por los menús.	Ningún problema o fallo.
Eliminar enemigos y mover la nave.	Los sprites desaparecen y se mueven cuando deben hacerlo y las animaciones funcionan correctamente.
Superar nivel.	Todo correcto.
Almacenaje de datos.	Se conserva toda la información.

Tabla 16: Tabla de test – Samsung Galaxy SII 2.3.7.

Samsung Galaxy SII 4.0.1

Prueba	Resultados
Carga de la Splashscreen.	Sin errores.
Carga del juego.	Cuando se pulsa el botón Play se crea una nueva partida sin problemas y todo se pone en movimiento sin problemas.
Navegar por los menús.	Ningún problema o fallo.
Eliminar enemigos y mover la nave.	Los sprites desaparecen y se mueven cuando deben hacerlo y las animaciones funcionan correctamente.
Superar nivel.	Todo correcto.
Almacenaje de datos.	Se conserva toda la información.

Tabla 17: Tabla de test – Samsung Galaxy SII 4.0.1.

HTC Desire HD

Prueba	Resultados
Carga de la Splashscreen.	Sin errores.
Carga del juego.	Cuando se pulsa el botón Play se crea una nueva partida sin problemas y todo se pone en movimiento sin problemas.
Navegar por los menús.	Ningún problema o fallo.
Eliminar enemigos y mover la nave.	Los sprites desaparecen y se mueven cuando deben hacerlo y las animaciones funcionan correctamente.
Superar nivel.	Todo correcto.
Almacenaje de datos.	Se conserva toda la información.

Tabla 18: Tabla de test – HTC Desire HD.

Samsung Galaxy SII 4.1.2

Prueba	Resultados
Carga de la Splashscreen.	Al ser un dispositivo físico real y no un emulador se puede observar una mejora muy notable de la velocidad de instalación y el arranque de la app.
Carga del juego.	Cuando se pulsa el botón Play se puede observar una clara mejoría en lo que a velocidad se refiere al cambiar de escenas.
Navegar por los menús.	Ningún problema o fallo.
Eliminar enemigos y mover la nave.	Los sprites desaparecen y se mueven cuando deben hacerlo y las animaciones funcionan correctamente.
Superar nivel.	Todo correcto.
Almacenaje de datos.	Se conserva toda la información.

Tabla 19: Tabla de test – Samsung Galaxy SII 4.1.2.

En lo que a la base de datos se refiere se comprobó varías veces que las puntuaciones se almacenaban correctamente para cada nivel y se mantenían.

En el apartado de desarrollo del juego hubieron muchos fallos a nivel de programación, pero todos fueron resueltos a medida que se depuraba el código. Fallos a nivel de lógica del juego también hubo bastantes. El más significativo fue que para poder mover el sprite de la nave y disparar el láser al mismo tiempo la función de multitouch tenía que estar siempre activada en el engine, pero eso suponía un problema. Al poder procesar dos o más dedos al mismo tiempo podías tocar la pantalla sin parar y eso ocasionaba que el juego tuviese una dificultad casi nula, ya que lo único que teníamos que hacer era esquivar los meteoritos. Es por esta razón que se creó la clase Cooldown para hacer que sólo se pudiese disparar cada cierto tiempo.

7. Conclusiones

7.1. Desviación temporal

Al contrario de lo previsto, la parte de diseño que inicialmente consideré que llevaría poco tiempo, no lo ha hecho. Este quizás ha sido el error más grave en lo que a este apartado se refiere. Esto ha sido a causa de que aunque sí se previó el tiempo que llevaría el desarrollo y aprendizaje de los diferentes lenguajes y solución de errores de forma dinámica, no se calculó bien la gran cantidad de tiempo invertida en la edición de imágenes y sonidos, debido a que se desconocía la complejidad de dicho proceso y se consideró que sería más simple. Nunca antes había utilizado programas de edición de imágenes y sonidos como los usados para el proyecto, en mi caso he usado Gimp 2.0 para las imágenes y Audacity para los sonidos, ambos son gratuitos.

Concepto	**Tiempo Estimado**	**Tiempo Real**
Iniciación	2h	2h
Planificación	61h	44h
Análisis	34h	31h
Diseño	55h	95h
Desarrollo	245h	205h
Test i pruebas	34h	20h
Implementación	15h	5h
Generación de documentos	30h	20h
Cierre del proyecto	1h	1h
Defensa del proyecto	24h	10h
Total	**501h**	**433h**

Tabla 20: Tabla de Desviación temporal.

7.2. Falta de recursos de trabajo

Para facilitar el desarrollo de este proyecto, más que ninguna otra cosa, hubiese necesitado la ayuda de algún diseñador y alguien con conocimientos de edición de sonidos. Ninguno de los dos procesos me ha aportado demasiado en lo que a aprendizaje del sistema o el motor gráfico se refiere y ambos han sido muy costosos en lo que a tiempo respecta. En parte causado todo por ideas que tenía sobre cómo mostrar una imagen y no saber cómo hacerlo o bien tras invertir cierta cantidad de horas intentándolo descubría que no se podían llevar a cabo.

En lo que a sonido respecta, si bien el impacto no ha sido tan agravado, cabe mencionar que no tenía ni idea de cómo utilizar la enorme cantidad de posibilidades que ofrecía Audacity ni qué debía modificar para conseguir los resultados que deseaba, por lo que modificaba parámetros a ciegas y se convirtió en un proceso de prueba y error un poco engorroso.

Dicho retraso ha ocasionado que ideas que aportaban algo más de riqueza a la lógica del juego que tenía muy claras para su implementación no se hayan podido llevar a cabo en el tiempo que poseía.

7.3. Ampliaciones

Si hubiera tenido un equipo con más personas se podrían haber implementado cosas como diferentes tipos de láser, más animaciones, más objetos para recoger con distintos efectos beneficiosos para el usuario, un sistema de compra de mejoras para la nave, sistema de puntuaciones online o incluso un modo de combate Jugador vs. Jugador.

Como ya he dicho, he tenido que editar yo mismo las imágenes con las cuales creamos los sprites del juego. La mayoría de las imágenes que se han usado han sido sacadas de bases de datos de imágenes libres, es decir que aunque han sido creadas por otra persona y posteriormente editadas por mí, en teoría poseo el permiso para utilizarlas como a mí me convenga, pero no puedo estar seguro. Esto significa que la aplicación no puede subirse al Google Market ahora mismo ya que si lo hiciera y percibiera ingresos podrían denunciarme por copyright.

Como ya he dicho arriba la cantidad de ampliaciones que puede recibir un juego de este género son innumerables, no hay más que observar otros juegos de esta misma categoría para darse cuenta de las posibilidades. La primera que yo haría sería implementar un sistema de compra de mejoras para la nave utilizando los puntos ganados durante los diferentes niveles. Esto tal y como está dispuesto actualmente la codificación de la app no sería para nada tan complicado. Tal vez podríamos añadir al menú principal el botón "mejoras", al pulsarlo se crearía un nuevo tipo de escena en la que tendrías las mejoras disponibles, quizás desbloqueables según el nivel máximo que se haya alcanzado. Hasta este punto todo sería muy sencillo de implementar debido a como están organizadas las clases del proyecto. En cuanto las mejoras posibles, pues en mi opinión podrían ser muy variadas, por ejemplo tenía la intención inicial de implementar un tipo de objeto recogible que te proporcionase un escudo que te protegiese del daño durante unos instantes, esto se podría implementar así o como mejora, activable cada x tiempo pulsando un botón en el HUD.

Otra posible ampliación seria permitir el juego de jugador contra jugador online. Esto significaría que a través de internet un jugador podría jugar contra otro. Además también se podría hacer un sistema de puntuaciones o logros en función de estas batallas.

A nivel de configuración podría permitir al usuario elegir si desea que el teléfono vibre o no, o si desea eliminar las animaciones. A mi parecer esto no era muy necesario ya que en la gran mayoría de teléfonos el juego funcionará fluidamente y no debería de dar problemas.

Más adelante el juego podría también contar con un mayor número de niveles y unidades enemigas, obstáculos o incluso jefes finales para cada nivel.

De momento los enemigos no se diferencian de los meteoritos más que en la velocidad que pueden alcanzar pero no sería muy complicado hacer que te disparasen también en los niveles muy avanzados.

7.4. Valoración del proyecto

De los objetivos propuestos se han conseguido tanto los funcionales como los no funcionales. Cabe destacar que el juego podría haber tenido mucha más jugabilidad, durabilidad, complejidad, animaciones y efectos, etc. pero se ha de tener cuenta que por desgracia debido al desconocimiento del entorno y plataforma para que se programaba la mayor parte del tiempo se ha invertido en aprender y no en parte creativa del proyecto como me hubiese gustado.

7.5. Valoración personal

Personalmente creo que este proyecto me ha ayudado mucho a conocer el lenguaje Java, algo del lenguaje específico de Android, aunque no mucho ya que en su mayor parte el motor gráfico hace de capa intermedia y con él es con el que interactuamos la mayoría de veces, y como acabo de destacar, sobretodo he aprendido sobre AndEngine, gráficos 2D y los problemas que pueden surgir a medida que se desarrolla un proyecto utilizando esta herramienta.

Todo el trabajo realizado me ha parecido interesante, el trabajar con imágenes en movimiento y arreglar errores visuales ocasionados por un error en la codificación no me ha parecido para nada pesado, al contrario, resolver dichos errores me parecía satisfactorio, en cambio los fallos provocados por la mala edición de alguna imagen y la correspondiente reedición de esta sí que lo ha hecho.

Finalmente destacar que toda la implementación y ver cómo se va dando forma a las ideas que van surgiendo me ha parecido muy entretenido y si bien al principio no me lo parecía, al final el hecho de buscar ejemplos o tutoriales o descomponer los propios ejemplos proporcionados por el creador de AndEngine y readaptarlos para aplicarlos como solución a tu problema me ha parecido muy didáctico en lo que a programar se refiere.

Bibliografía

- Foro oficial de AndEngine en el que se exponen soluciones a muchos de los problemas más comunes y donde hacer consultas a otros programadores: http://www.andengine.org/forums/gles2/
- StackOverflow: web en la que se proponen preguntas a otros programadores, no sólo de AndEngine si no de índole general: http://stackoverflow.com/
- Introducción a AndEngine(tutorial desactualizado): http://droideando.blogspot.com.es/2011/03/introduccion-andengine-parte-i.html
- Blog personal en el que se explican muchas de las funcionalidades de AndEngine: http://www.matim-dev.com/tutorials.html
- AndEngine Guides: Blog en el que se explican con gran detalle algunas de los tipos de datos y otros elementos que emplea AndEngine: http://andengineguides.wordpress.com/
- Video explicativo sobre la configuración de Eclipse para usar AndEngine: http://www.youtube.com/watch?v=lQW1WQOCri0
- Ejemplos de AndEngine: https://github.com/nicolasgramlich

Signat: ***Sergio Pérez Carretero***