

TOWER3D: DISEÑO, DESARROLLO Y COMERCIALIZACIÓN DE UN VIDEOJUEGO MULTIPLATAFORMA 3D CON TECNOLOGÍAS LIBRES.



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

TOWER3D: Diseño, desarrollo y comercialización de un videojuego multiplataforma 3D con tecnologías libres.

Alfredo Arteaga Sánchez

Septiembre, 2016



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA
Departamento de Tecnologías y Sistemas de Información

PROYECTO FIN DE CARRERA

TOWER3D: Diseño, desarrollo y comercialización de un videojuego multiplataforma 3D con tecnologías libres.

Autor: Alfredo Arteaga Sánchez
Director: Dr. Carlos González Morcillo

Septiembre, 2016

Alfredo Arteaga Sánchez

Ciudad Real – Spain

E-mail: alfredo.arteaga@uclm.es

Teléfono: +34 659 93 49 01

Web site:

© 2016 Alfredo Arteaga Sánchez

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Se permite la copia, distribución y/o modificación de este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.3 o cualquier versión posterior publicada por la *Free Software Foundation*; sin secciones invariantes. Una copia de esta licencia esta incluida en el apéndice titulado «GNU Free Documentation License».

Muchos de los nombres usados por las compañías para diferenciar sus productos y servicios son reclamados como marcas registradas. Allí donde estos nombres aparezcan en este documento, y cuando el autor haya sido informado de esas marcas registradas, los nombres estarán escritos en mayúsculas o como nombres propios.

TRIBUNAL:

Presidente:

Secretario:

Vocal:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE

SECRETARIO

VOCAL

Fdo.:

Fdo.:

Fdo.:

*A mis padres, que son los mejores, a mi familia en general y a Isa (mi particular Clippo).
También a mi familia peluda (Cheetos, Gordopilo, Ñapas, Raspas, Zoidberg y Rati, va por
vosotros también.)*

Resumen

La industria del desarrollo de videojuegos está cambiando. Desde la última década, el perfil del desarrollador de videojuegos se ha diversificado. Anteriormente, la industria del desarrollo de videojuegos estaba formada por grandes compañías que realizaban videojuegos en los que se requería un equipo de desarrollo enorme y cuyo tiempo de desarrollo podía extenderse varios años.

Actualmente, han aparecido factores que han conseguido que el perfil de desarrollador varíe, permitiendo que pequeñas empresas o diseñadores autónomos publiquen sus propios juegos. Factores como el abaratamiento de la conectividad de datos en dispositivos móviles, la mayor heterogeneidad del perfil del consumidor de videojuegos y la aparición de múltiples plataformas digitales de distribución de videojuegos con precios económicos, son los que han permitido este cambio en el espectro de la industria del desarrollo de videojuegos.

Sin embargo, a pesar de la reducción de la inversión necesaria en el desarrollo de un videojuego en materia de edición, distribución y marketing, suele ser necesario un importante desembolso en herramientas profesionales que ayuden al desarrollo e implementación del videojuego.

El presente trabajo de PFC surge con el objetivo de crear un prototipo de un videojuego utilizando en exclusiva software libre, reduciendo así el coste en software de manera drástica y obteniendo todos los beneficios del software libre.

Para el prototipo se ha escogido un videojuego sencillo de tipo Tower Defense, que consiste en que el jugador debe situar estratégicamente unas torres que disparan proyectiles, con el fin de evitar que una horda de enemigos cruce un mapa de extremo a extremo.

Para el desarrollo del videojuego se analizarán las herramientas disponibles que sean software libre, eligiendo las más adecuadas según diferentes criterios, como nivel de madurez, funcionalidades ofrecidas, documentación disponible y conocimientos actuales del desarrollador sobre la herramienta.

Se analizarán después las opciones que existen en el mercado para obtener rentabilidad del desarrollo, hablando tanto de herramientas software como de modelos de negocio.

Por último, se realizará un pequeño cálculo de costes, enfrentado a los costes estimados del mismo proyecto con tecnología propietaria, para obtener conclusiones acerca de la idoneidad del uso de software libre para el desarrollo de videojuegos en plataformas móviles.

Índice general

Resumen	XI
Índice general	XIII
Índice de cuadros	XVII
Índice de figuras	XIX
Índice de listados	XXIII
Listado de acrónimos	XXV
1. Introducción	1
1.1. Justificación	2
1.2. Estructura del documento	4
1.2.1. Capítulo 1: Introducción	4
1.2.2. Capítulo 2: Objetivos	4
1.2.3. Capítulo 3: Antecedentes	4
1.2.4. Capítulo 4: Método de trabajo	4
1.2.5. Capítulo 5: Resultado Final	4
1.2.6. Conclusiones y propuestas	4
1.2.7. Bibliografía	4
1.2.8. Apéndices	5
2. Antecedentes	2
2.1. La industria del videojuego	2
2.1.1. Aumento de usuarios	2
2.2. Géneros de videojuegos	2
2.2.1. Lucha	3
2.2.2. Arcade	5
2.2.3. Disparos	6

2.2.4.	Carreras	8
2.2.5.	Estrategia	9
2.2.6.	Massively Multiplayer Online (MMO)	11
2.3.	Diseño de un videojuego	12
2.3.1.	Las emociones como parte del videojuego	12
2.3.2.	Equipo de desarrollo de un videojuego	13
2.3.3.	Motores de juego	16
3.	Objetivos	1
3.1.	Objetivo general	1
3.2.	Objetivos específicos	1
3.2.1.	Objetivos funcionales o de diseño	1
3.2.2.	Objetivos técnicos o de implementación	2
3.2.3.	Objetivos Comerciales	3
4.	Método de trabajo	2
4.1.	Metodología	2
4.2.	Hardware	3
4.3.	Motor de Juego	3
4.4.	Otras herramientas de Software	8
4.4.1.	Sistema Operativo	8
4.4.2.	Emacs	8
5.	Resultados	1
5.1.	Diseño del videojuego	1
5.1.1.	Desarrollo del concepto	1
5.2.	Preparando el entorno	4
5.2.1.	Instalación de Android NDK y SDK	4
5.2.2.	Compilando Ogre3D para Android	4
5.2.3.	Descarga y compilación de las dependencias de OGRE3D	5
5.3.	Arquitectura del proyecto	8
5.3.1.	Gestión de Enemigos	8
5.3.2.	Gestión de las torres	12
5.3.3.	Gestión del mapa	16
5.3.4.	Sistemas Transversales o de Apoyo	21
5.4.	Decisiones Técnicas	23

5.4.1.	Gestión de la entrada	26
5.4.2.	Interfaz de usuario	26
5.4.3.	Técnicas de Sombreado	27
5.4.4.	Ogre3D y SDL	29
5.4.5.	Otros ajustes	31
5.5.	Análisis de Costes	32
5.5.1.	Investigación y establecimiento de entorno de desarrollo.	33
5.5.2.	Desarrollo del prototipo	33
5.6.	Monetización y Marketing	33
5.6.1.	Definición del producto y mercado objetivo	34
5.6.2.	Modelos de negocio	34
5.6.3.	Calculo de amortización	37
5.6.4.	Marketing	38
6.	Conclusiones y propuestas	2
6.1.	Conclusiones	2
6.2.	Objetivos de diseño	2
6.3.	Objetivos técnicos o de implementación	4
6.4.	Conclusiones globales	8
6.5.	Propuestas de trabajo futuro	11
6.5.1.	Mejoras en la calidad gráfica	11
A.	Guia rápida de uso	2
A.0.1.	Pantalla de cargando	2
A.0.2.	Pantalla de Menú	2
A.0.3.	Pantalla Principal de juego	2
B.	Estructura Física del proyecto	8
	Bibliografía	10

Índice de cuadros

2.1. Categorías de videojuegos	10
5.1. Coste de Investigación	33
5.2. Coste de Desarrollo	33
5.3. Cuadro de Amortización	38

Índice de figuras

1.1. Distribución del mercado actual por dispositivos [11]	1
1.2. Aumento significativo de la distribución digital del videojuego [11]	2
2.1. Captura de pantalla del juego Street Fighter II (Capcom)	3
2.2. Captura de pantalla del juego King of Fighters (SDK)	4
2.3. Captura de pantalla del juego Super Smash Bros (Nintendo)	4
2.4. Captura de pantalla del juego Knights of the Round (Capcom)	5
2.5. Captura de pantalla del Space Invaders juego (Taito Corporation)	6
2.6. Captura de pantalla de Doom (Id Software)	7
2.7. Captura de pantalla de Max Payne (Remedy Entertainment)	8
2.8. Captura de pantalla de TrackMania (Nadeo)	9
2.9. Captura de pantalla de Starcraft (Blizzard)	10
2.10. Captura de pantalla de Freeciv (The Freeciv Project)	11
2.11. Captura de pantalla de Lineage II (NCSoft)	12
2.12. Gráfico Flow	13
2.13. BluePrint Editor	18
2.14. Editor Unity	19
4.1. Captura de Pantalla de Rush Hour Hero (implementado con Gameplay3D) .	5
4.2. Captura de Pantalla de Experience Curiosity, desarrollado por NASA Jet Propulsion Laboratory (implementado con Blend4Web)	6
4.3. Captura de Pantalla de Scruffy, desarrollado por Bran Barnes, autor de Dim3) .	7
4.4. Captura de Pantalla del editor de Godot	8
4.5. Captura de Pantalla de Steam Quest, desarrollado por Kívano (implementado con Godot)	9
4.6. Captura de Pantalla de TorchLight, desarrollado por Runic Games (implementado con Ogre)	10
4.7. Captura de Pantalla de Emacs. Se puede observar como están activadas las extensiones de FlyCheck, Yasnippet y Projectile.	11
4.8. Captura de la edición de uno de los botones en GNU Image Manipulation Program (GIMP).	11

4.9. Animando un enemigo por cinemática inversa en Blender.	13
5.1. Dos robots de ambientación Steam Punk	3
5.2. Iconos de construcción de torres	4
5.3. Relación entre la versión de Android y el nivel de Application Program Interface (API) [17].	5
5.4. Etiqueta 1.9 del código fuente de Ogre3D.	6
5.5. Diagrama de Clases del módulo de gestión de enemigos.	9
5.6. Diagrama de Clases del módulo de gestión de torres.	13
5.7. Diagrama de Clases del módulo de gestión de mapas.	17
5.8. Diagrama de estados del juego	22
5.9. Editor de mapas del juego, elaborado en Blender mediante el lenguaje Python	24
5.10. Editor de mapas del juego, con los vértices de tipo road resaltados.	25
5.11. Pantalla de juego realizada con tiles.	26
5.12. Pantalla de juego realizada con una sola malla de geometría estática.	27
5.13. Captura de pantalla de SampleBrowser de Ogre3D, que utiliza SDKTrays para su interfaz usuario.	28
5.14. Captura de pantalla de Open Dungeons en la que se puede apreciar la interfaz, implementada con CEGUI.	28
5.15. Captura de pantalla de Energy Tycoon, cuya interfaz está implementada con MYGUI.	29
5.16. Comparativa de modos de sombreado. Se pueden observar los bordes duros de la sombra mediante Stencil Buffer.	31
5.17. Compras en Pokemon GO (desarrollado por Niantic).	35
5.18. Kayak, un buscador de vuelo, se rentabiliza por comisiones de venta.	36
5.19. Minecraft: Pocket Edition (desarrollado por Mojang).	37
5.20. Imágen de un video del Canal LuzuGames, especializado en juegos.	39
6.1. Ejemplo de un botón del HUD.	2
6.2. Ejemplo de un botón del HUD.	4
6.3. Textura del enemigo de tipo Robot. El efecto de metalizado se ha conseguido mediante la aplicación de varios filtros.	7
6.4. Un ejemplo de Baking para mejorar un modelo de baja densidad poligonal.	12
6.5. Ogre3D Particle Lab que permite experimentar con diferentes parámetros de partículas. Funciona tanto en Windows como en Linux (bajo Wine).	13
6.6. Pantalla de carga de juego Clash of Clans (desarrollado por SuperCell). Se puede ver el dibujo de gran calidad, junto con una barra de progreso y un consejo ofrecido al jugador.	14
6.7. Niebla en Ogre3D.	15

6.8. Ejemplo de Lluvia en Ogre3D mediante partículas.	16
6.9. Captura de pantalla de Clash Royale (desarrollado por Supercell).	17
6.10. Logros de Pokemon GO (desarrollado por Niantic), representados por medallas	17
6.11. Tablero de clasificación de Clash Of Clans (SuperCell).	18
A.1. Pantalla de bienvenida al juego	3
A.2. Menú Principal de Tower3D	4
A.3. Pantalla de juego de TOWER3D. El jugador todavía no ha hecho nada. . . .	5
A.4. Ubicación Válida, se puede observar el círculo verde.	5
A.5. Ubicación Inválida, se puede observar el círculo rojo.	6
A.6. Mensaje advirtiéndole al jugador que falta dinero para construir o mejorar una torre.	6
A.7. Panel de Mejora. El botón con la flecha hacia arriba permite mejorar la torre.	7
A.8. El jugador se ha quedado sin vidas y ha perdido.	7

Índice de listados

5.1.	Comando para averiguar si el sistema es Big Endian o Little Endian en Linux.	6
5.2.	Líneas a sustituir en CMakeList.txt	6
5.3.	Código para establecer Little Endian en CMakeList.txt. Cambiar FALSE por TRUE si el sistema es Big Endian	7
5.4.	Corregir los paths de éstas líneas. Solamente será necesario cambiar el número de versión (en este ejemplo 4.9)	7
5.5.	Comando para compilar Ogre para Android. Los 4 últimos flags no figuran en [10]	7
5.6.	Ejemplo del contenido de enemys.cfg	10
5.7.	Ejemplo del contenido de waves.cfg	12
5.8.	Ejemplo del contenido de towers.cfg	15
5.9.	Un archivo graphML	18
5.10.	Código para lanzar una búsqueda por anchura.	19
5.11.	Script de overlays para dibujar los botones del menú.	30
5.12.	Carga de la pantalla de Loading. Se aprecia la diferencia de código según la plataforma.	32
6.1.	Ejemplo de bifurcación de código mediante el uso de OGRE_PLAFORM. .	5

Listado de acrónimos

BSP	Binary Space Partition
GPL	GNU Public License
FPS	First Person Shooter
IA	Inteligencia Artificial
RTS	Real Time Strategy
TBS	Turn Based Strategy
MMO	Massively Multiplayer Online
MMORPG	Massively Multiplayer Online Roleplaying Game
PFC	Proyecto de Fin de Carrera
SDK	Software Development Kit
NDK	Native Development Kit
APK	Android Application Package
HUD	Heads-up Display
FSF	Free Software Foundation
API	Application Program Interface
GCC	GNU Compiler Collection
GIMP	GNU Image Manipulation Program
SDL	Simple DirectMedia Layer
STL	Standard Template Library
CEGUI	Crazy Eddie's GUI
GPU	Graphic Processing Unit
CPU	Central Processing Unit
HID	Human Interface Devices
JNI	Java Native Interface

CPM	Costo por mil
FAQ	Frequently Asked Questions
ADB	Android Debug Bridge
SD	Secure Digital

Capítulo 1

Introducción

El negocio del desarrollo de videojuegos ha sufrido en la última década considerables cambios en su estructura de negocio. Dicho cambio ha sido impulsado por diferentes factores, muchos de ellos tecnológicos.

Uno de dichos factores es la introducción en los últimos 10 años de los dispositivos móviles, como tablets y smartphones. En la figura 1.1 se muestra la distribución del mercado actual de los videojuegos en función de los dispositivos objetivo.

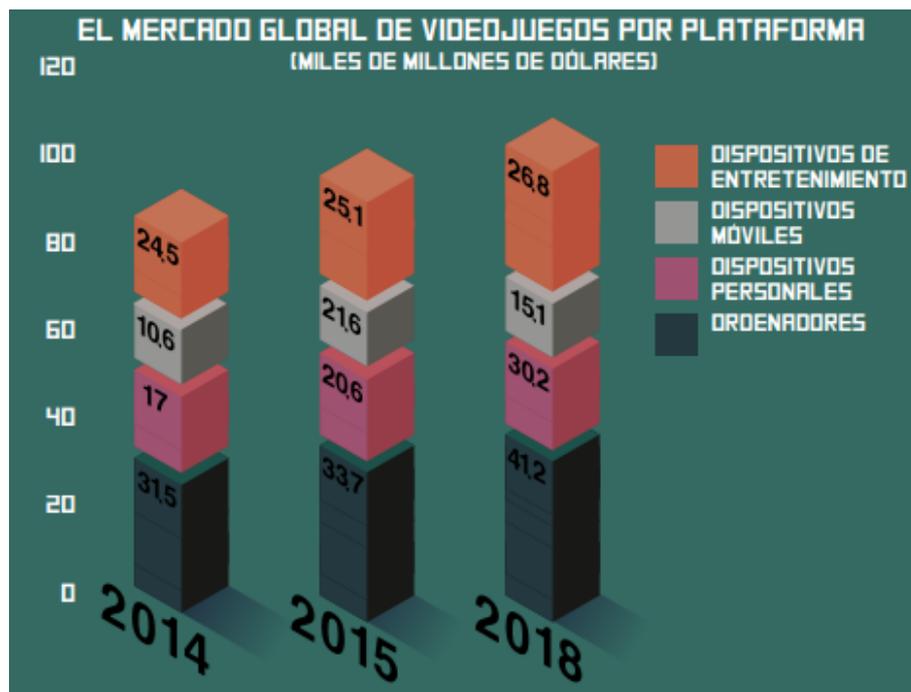


Figura 1.1: Distribución del mercado actual por dispositivos [11]

Otro de ellos es un cambio en la cadena de distribución de los videojuegos. La distribución digital de los videojuegos está creciendo a la par que se ve mermada la distribución tradicional en formato físico (ver figura 1.2).

El cambio en la cadena de distribución ha facilitado mucho la autoedición de videojuegos, lo que a su vez ha causado la proliferación de los juegos independientes, creados por un único

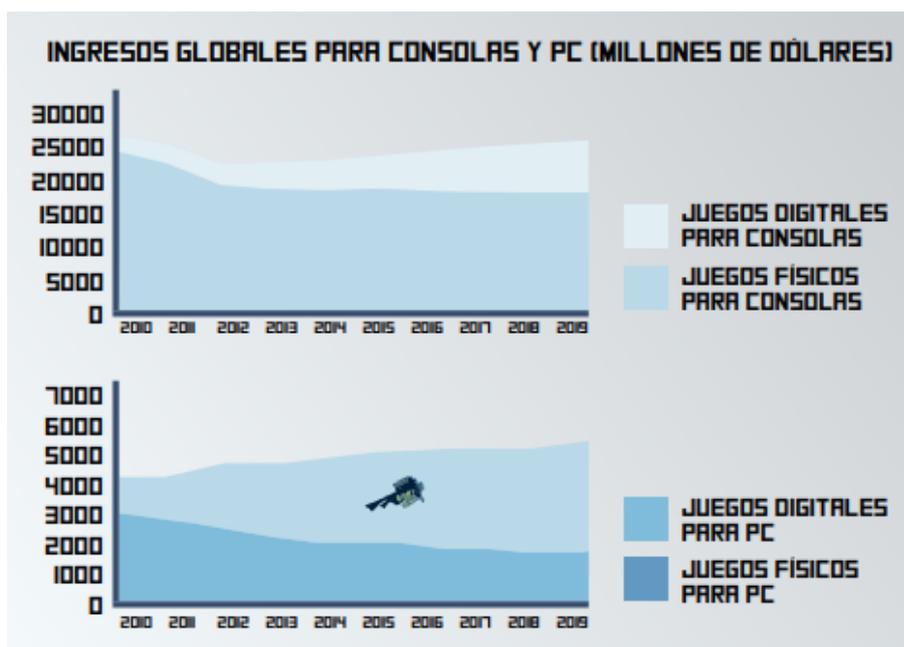


Figura 1.2: Aumento significativo de la distribución digital del videojuego [11]

desarrollador, o bien, por pequeñas compañías dedicadas al desarrollo de videojuegos.

Es en esta situación actual donde el presente Proyecto de Fin de Carrera (PFC) cobra sentido.

1.1 Justificación

Actualmente existen múltiples plataformas o frameworks de desarrollo adaptados exclusivamente para la creación de videojuegos. Sin embargo, las alternativas de software libre son bastante limitadas, y normalmente están más enfocadas al ámbito educativo o experimental que a la implementación de videojuegos con enfoque comercial.

El presente PFC se ha planteado como una oportunidad para incluir una amplia diversidad de metodologías, tecnologías y conocimientos en general sobre el desarrollo de videojuegos. Así, se pretende elaborar un prototipo de videojuego utilizando software libre para dispositivos móviles.

¿Por qué Software Libre?

Existiendo en una cantidad elevada de motores de juego y herramientas de software propietario, que tienen un nivel de madurez alto y que pueden ser relativamente asequibles, surge la pregunta de por qué limitarse al software libre.

Al margen del punto de vista filosófico que engloba el movimiento del software libre[28], el software libre tiene varias características en el ámbito técnico que lo hacen deseable:

- El software libre es (casi siempre) gratuito, con lo que el presupuesto para el desarro-

llo de un juego se reduce. Además, posibilita comenzar el desarrollo de un videojuego sin necesidad de un inyección económica inicial.

- El código fuente está disponible para su estudio. Esto permite solucionar lagunas en la documentación del programa o librería que se esté utilizando.
- Con el código fuente disponible, es posible realizar depuraciones de forma sencilla, lo que facilita mucho la solución de errores no fatales.
- Los proyectos de software libre suelen tener una comunidad de usuarios bastante amplia, que además suele poseer unos conocimientos técnicos bastante avanzados, ya que muchos de los que colaboran en dichas comunidades probablemente han realizado aportaciones al código fuente.

Plataforma objetivo principal

La plataforma objetivo principal que se ha escogido es Android, aunque las tecnologías que se deben usar en el juego deben ser portables a otras plataformas. Se ha seleccionado Android por diversos motivos:

- Es una plataforma accesible, son Software Development Kit (SDK) y Native Development Kit (NDK) gratuito y con un amplio soporte en foros.
- La calidad gráfica y el detalle de los modelos es menor en un entorno móvil que en una consola o un PC. Esto permite reducir el tiempo invertido en horas en el apartado gráfico.
- Es el mercado idóneo para un juego casual.
- No es necesario adquirir hardware adicional para programar para la plataforma Android, como si lo es para iOS.

Enfoque

Se ha afrontado el proyecto mezclando dos enfoques directamente, que en cierta medida se podrían considerar que están enfrentados:

Enfoque académico Este enfoque se ve reflejado por el uso casi exclusivo de software libre, que, aunque ha sido usado de manera profesional por algunos desarrolladores y compañías de videojuegos, no es de uso habitual en dispositivos móviles en videojuegos en 3D. Así, el presente proyecto pretende también realizar una labor de investigación que permite determinar si el uso de software libre es viable o no, y sobre todo, en caso de ser viable, si se rentabilizaría su uso en un entorno profesional.

Enfoque comercial Mediante una estimación de costes, el estudio de los costes de publicación del videojuego en una plataforma de difusión de aplicaciones móviles (Google Play Store), se realizará un estudio sobre como obtener la máxima rentabilidad posible

del desarrollo. Este enfoque comercial es uno de los motivos por los que el videojuego se ha realizado para el entorno Android es su amplia difusión, la amplitud de su mercado potencial y el bajo coste asociado a la distribución del juego.

1.2 Estructura del documento

Para la redacción de este PFC se ha seguido la estructura ofrecida en la normativa de la Escuela Superior de Informática (UCLM). La estructura es la siguiente:

1.2.1 Capítulo 1: Introducción

El presente capítulo, en el que se expone brevemente el concepto global y la justificación del PFC

1.2.2 Capítulo 2: Objetivos

En este capítulo, se realiza un desglose de los objetivos del proyecto, desglosándoles en objetivos de índole funcional y objetivos de índole técnica.

1.2.3 Capítulo 3: Antecedentes

Se detalla en este capítulo el estado actual de la industria del videojuego, así como los elementos a tener en cuenta a la hora de diseñar un videojuego. Se mencionan también la estructuración más común de un equipo de desarrollo, así como algunas de las herramientas actuales más utilizadas.

1.2.4 Capítulo 4: Método de trabajo

En este capítulo se presenta la metodología utilizada en el desarrollo de dicho proyecto, así como las herramientas que han sido utilizadas para llevarlo a cabo.

1.2.5 Capítulo 5: Resultado Final

En este capítulo se detallan las dificultades técnicas encontradas en la implementación del proyecto, así como diversas técnicas utilizadas. Se ofrece un diseño técnico de la implementación final, detallando patrones de diseño utilizados. Así mismo se ofrece también un breve estudio de costes para analizar la futura rentabilidad del proyecto

1.2.6 Conclusiones y propuestas

En este capítulo se enumeran unas cuantas posibilidades de trabajo futuro, así como se analiza el trabajo que se ha llevado a cabo.

1.2.7 Bibliografía

Este capítulo contiene las fuentes bibliográficas consultadas por el alumno para la realización del PFC

1.2.8 Apéndices

Los diferentes apéndices del este documento son:

- A. Manual del videojuego. El manual del videojuego contienen las instrucciones que debe seguir el jugador para jugar.
- B. Estructura Física. Estructura física de los archivos que componen el código fuente y los recursos, tanto gráficos como sonoros del videojuego.

Capítulo 2

Antecedentes

Este capítulo contiene información sobre el estado actual sobre el mundo de desarrollo de videojuegos, dando primero una visión global de la industria del videojuego, para concretar después sobre el desarrollo del mismo.

2.1 La industria del videojuego

La industria de los videojuegos mueve alrededor de 63.000 millones de dolares anuales por todo el mundo, rivalizando con el tamaño de la industria del cine, que genera alrededor de 87 millones de dólares. Muchos videojuegos generan beneficios equiparables a las recaudaciones de superproducciones de Hollywood, y su desarrollo puede requerir un presupuesto igualmente elevado, por lo que una juego que no alcance el objetivo de rentabilidad puede llevar a la quiebra al estudio que lo desarrolló.

2.1.1 Aumento de usuarios

Se ha registrado en los últimos años un aumento importante de usuarios, debido a la realización de videojuegos que son capaces de llegar a un público más amplio, que hasta entonces no había hecho uso de los videojuegos. Como se explica en [11], Nintendo inició esta tendencia con la consola de sobremesa Wii. La inclusión de un sistema de control revolucionario, a la par que extremadamente sencillo de manejar, consiguió que muchos hogares que hasta entonces no habían tenido consola, compraran la Wii.

La explosión de los Smartphones, también ha abierto un mercado muy importante para los videojuegos. La facilidad con la que el usuario es capaz de descargarse un juego y jugar, hace que el público objetivo sea aún más amplio.

En 2.1 puede verse una división de los videojuegos, dependiendo de su plataforma objetivo, así como las características más importantes que los identifican de forma general.

2.2 Géneros de videojuegos

En la actualidad existe una gran diversidad en cuanto al genero de videojuegos. Existen juegos diseñados para tener enganchado al jugador durante horas, así como juegos cuya única pretensión es ofrecer divertimento en pequeños espacios de tiempo libre que tenga el



Figura 2.1: Captura de pantalla del juego Street Fighter II (Capcom)

jugador. En esta sección se pretende realizar una clasificación de videojuegos por género. Se debe tener en cuenta que, aunque hay algunos géneros bien consolidados, en algunos casos las diferencias entre géneros es, cuando menos difusa, pudiendo un juego ser catalogado en más de una categoría.

2.2.1 Lucha

Los videojuegos de lucha son aquellos en los que el jugador debe, manejando un personaje, deshacerse de un enemigo. El jugador dispone de una barra de salud, que al principio está completa, pero que irá menguando con los ataques enemigos. Si la barra del jugador desaparece completamente, el jugador habrá perdido. Aunque existen en 3D, la mayor popularidad de estos juegos se alcanzó con videojuegos 2D en vista lateral, a la que posteriormente se le añadió la posibilidad de zoom, siendo Art of Fighting uno de los primeros videojuegos en incorporarla. Se pueden distinguir varios subtipos:

Lucha 1vs1 En este tipo de juegos, el jugador elegía un único personaje, con el que debía derrotar al resto de personajes del juego. Es usual que cada personaje tenga un escenario propio, en el que refleja detalles del personaje (como su nacionalidad, o su profesión). Probablemente el juego más conocido, y con el que este género de juegos se hizo popular es Street Fighter 2 (ver figura 2.1).

Lucha por equipos Es una evolución del género Lucha 1vs1, en la que el jugador en vez de seleccionar un único personaje, elige varios con lo que forma un equipo. Este equipo suele estar formado por tres personajes, de los que el jugador controla solamente una



Figura 2.2: Captura de pantalla del juego King of Fighters (SDK)

a la vez. Puede normalmente cambiar en mitad del combate un personaje por otro, y para ganar debe eliminar a todos los personajes del equipo contrario. Este modo de juego lo popularizó la saga King of Fighters de la compañía SDK (ver figura 2.2).

Lucha todos contra todos Este tipo de videojuego de lucha suele involucrar a múltiples jugadores a la vez, en escenarios más grandes usualmente que en los otros juegos de lucha. Prima especialmente la espectacularidad y en cierto modo la comicidad de los estilos de lucha y de los personajes, buscando un entretenimiento más distendido. Un buen ejemplo de este género es la saga Super Smash Bros de Nintendo (ver figura 2.3).



Figura 2.3: Captura de pantalla del juego Super Smash Bros (Nintendo)

Según especifica [18] los juegos de lucha tienen las siguientes particularidades técnicas:

- Un conjunto amplio y detallado de animaciones de pelea.
- Detección precisa de golpes.
- Un sistema de entrada capaz de detectar y gestionar múltiples y combinadas combinaciones de teclas, movimientos de joystick, etc..
- Escenarios detallados y amplios, aunque estáticos.

2.2.2 Arcade

Arcade es probablemente uno de los géneros más amplios y difusos de los videojuegos. El término arcade tiene una connotación temporal, ya que suele utilizarse para referirse a videojuego antiguos. También suele usarse para referirse a los videojuegos que se encuentran en los salones recreativos, que funcionan con monedas. Por último, con el término arcade también se clasifican los juegos que ponen gran énfasis en una jugabilidad sencilla, que va aumentando la dificultad conforme se va avanzando en el videojuego, y que normalmente presenta una acción bastante frenética y constante. Algunos tipos de juegos arcade son:

beat'em up Normalmente en 2D, con una vista lateral y scroll horizontal, el jugador debe escoger un personaje con el que avanzar por cada nivel derrotando a todos los enemigos que se encuentra. En este tipo de juegos es muy común que exista un enemigo de final de fase, que suele emplear ataques más poderosos y tener más vida que los enemigos normales. En videojuegos como *Knights of the Round*, se añade además un elemento de progreso del personaje, que va ganando habilidades o poder conforme va acabando con los enemigos. En la figura 2.4 se pueden ver los clásicos elementos de la interfaz de un beat'em up, como la barra de vida, o el marcador de puntos.



Figura 2.4: Captura de pantalla del juego *Knights of the Round* (Capcom)

shoot'em up Este tipo de juegos el jugador suele manejar un avión, nave espacial u otro

objeto similar esquivando los disparos de los enemigos e intentando derribarlos con los disparos propios. Si el scroll es vertical, la vista suele ser cenital, mientras que si es lateral, el scroll suele ser horizontal. Space Invaders es un claro ejemplo de juegos shoot'em up.



Figura 2.5: Captura de pantalla del Space Invaders juego (Taito Corporation)

2.2.3 Disparos

En los juegos de disparos, el jugador maneja a un único personaje, que debe cruzar un nivel abatiendo a los enemigos mediante armas de fuego. En esta categoría podemos distinguir varios subgéneros.

FPS Los First Person Shooter (FPS) son juegos en los que el jugador normalmente controla a un personaje con una vista en primera persona a lo largo de escenarios [30]. Dichos escenarios normalmente han sido interiores, por necesidades de rendimiento. Actualmente se encuentran indistintamente escenarios exteriores e interiores, ambos con gran nivel de detalle. Fue, junto con los videojuegos de estrategia, los primeros videojuegos que explotaron la posibilidad de jugar varios jugadores a través de una red. Así, Doom (ver figura 2.6, publicado por Id Software en 1993, ya permitía jugar en red a través de un módem, tanto en modo cooperativo como en competitivo. Según [18] un FPS puede caracterizarse por:

- Renderizado eficiente de enormes mundos virtuales en 3D.
- Mecanismo eficiente de movimiento de cámara y apuntado del personaje.
- Animaciones de alta calidad para los brazos del personaje y el arma.

- Una amplia gama de potentes armas de fuego.
- Debido a al sistema de colisiones y la forma en la que se desplaza el jugador, da la sensación de que el personaje flota ligeramente sobre el escenario.
- Animaciones de alta calidad para los enemigos y aliados del jugador.
- Inteligencia Artificial (IA) para los enemigos y los personajes aliados.
-



Figura 2.6: Captura de pantalla de Doom (Id Software)

Disparos en tercera persona En este tipo de videojuego, el jugador maneja a un personaje que aparece en pantalla. La vista del juego suele seguir al personaje, desde un posición un poco más retrasada y elevada, permitiendo contemplar al personaje y un alto porcentaje del escenario. Este tipo de juegos suele incorporar elementos de tipo beat'em up cuando los enemigos están demasiado próximos como para utilizar armas a distancia. Ciertos juegos de este género incorporan elementos de juegos de plataformas, debiendo el personaje realizar saltos o movimientos de cierta dificultad para poder pasar de una zona de un nivel a otra.

Es un género muy amplio, en el que se han producido diferentes mejoras en ciertos juegos pioneros, que después han sido reproducidos en otros juegos posteriores, marcando una tendencia clara. Es el caso de Tomb Raider, con una mezcla de complicados puzzles y acción, así como de Max Payne, que introdujo una ralentización del tiempo percibida como el usuario, denominada tiempo bala. En la figura 2.7 se puede observar un juego de disparos en tercera persona, con la clásica perspectiva retrasada y elevada respecto de la posición del personaje que controla el jugador.



Figura 2.7: Captura de pantalla de Max Payne (Remedy Entertainment)

2.2.4 Carreras

Los videojuegos de carreras son aquellos en los que el jugador controla alguna clase de vehículo, con el cual intenta cumplir un objetivo, que casi siempre es acabar el primero corriendo en un circuito. Este género abarca desde juegos que podrían englobarse en la categoría arcade, como TrackMania (ver figura 2.8) hasta juegos que pretenden simular una conducción lo más fiel posible a la realidad, como la saga Gran Turismo. En este tipo de juegos, es bastante usual que el detalle visual esté centrado en los vehículos y en las cercanías del circuito. Las características técnicas de los juegos de carreras suelen ser [18]:

- Se utilizan varios trucos para el renderizado de los elementos de fondo más lejanos, tales como utilizar objetos bidimensionales para el dibujo de árboles o colinas.
- El circuito normalmente se divide en secciones bidimensionales relativamente simples denominadas sectores. Estos sectores se utilizan tanto para optimizar el renderizado como para ayudar a la IA de los vehículos no manejados por jugadores así como otros usos técnicos diferentes.
- La cámara normalmente sigue al vehículo como si fuera un juego con perspectiva de tercera persona, aunque en muchos juegos puede ser cambiada por el jugador, moviéndola más lejos o más cerca del vehículo, hasta llegar incluso a situarse dentro de la cabina de pilotaje del mismo.
- Cuando la cámara atraviesa un túnel o cualquier otro espacio cerrado, es necesario realizar un esfuerzo en el movimiento de la misma para evitar que colisione con los

objetos del escenario.



Figura 2.8: Captura de pantalla de TrackMania (Nadeo)

2.2.5 Estrategia

Los juegos de estrategia son aquellos en los que el jugador maneja una organización (puede ser una civilización, una raza, un conglomerado empresarial) cuyo fin es derrotar a sus rivales, normalmente mediante el combate. Las sagas Warcraft, Command and Conquer, Civilization y Age of Empires son unos ejemplos de los videojuegos que hicieron tremendamente popular este género. Si tenemos en cuenta la manera de jugar, se pueden clasificar los juegos de estrategia en dos grandes géneros:

RTS Los Real Time Strategy (RTS) son videojuegos de estrategia en los que la acción sucede en tiempo real, sin interrupciones de ningún tipo. Todos los jugadores y la IA realizan las acciones de manera simultánea. Un ejemplo de este tipo de videojuegos es la saga StarCraft, de Blizzard. En la figura 2.9 se puede observar una captura de Starcraft (Blizzard), uno de los RTS más populares a pesar de haber sido publicado en 1998.

TBS En los videojuegos de tipo Turn Based Strategy (TBS) la acción está dividida en turnos. Así, cuando el jugador está en su turno, es el único que puede realizar acciones, aunque en algunos juegos se permite al jugador jugar en el turno del oponente cuando es atacado. La saga más conocida de TBS es Civilization de Sid Meier. También merece mención la saga Heroes of Might and Magic. En la figura 2.10 se puede observar una típica interfaz de un TBS.

Como se puede ver en [18], algunas de las prácticas técnicas más usadas son:

Consola	PC	Dispositivos Móviles
Funciona en Hardware dedicado	Funciona en Windows, Mac o Linux	Funciona en Tablets y Smartphones
Desarrollo caro	Coste y género variable	Menor coste de desarrollo
Amplia variedad de género	Amplia Variedad de género	Juegos sociales y casuales
Sistema controlado por el fabricante	No hay control unificado del sistema	
Venta física y digital	Mayoría de ventas en digital	Venta digital. Mayor número de jugadores potenciales.

Cuadro 2.1: Categorías de videojuegos



Figura 2.9: Captura de pantalla de Starcraft (Blizzard)



Figura 2.10: Captura de pantalla de Freeciv (The Freeciv Project)

- Las unidades suelen ser diseñadas en baja resolución, para permitir que el sistema pueda mostrar un gran número de ellas.
- El mapa de juegos suele estar diseñado mediante la técnica de mapa de alturas, que consiste en representar mediante diferentes colores la altura de cada punto del mapa.
- Los jugadores suelen tener la posibilidad de construir nuevas estructuras además de unidades.
- La interacción del usuario suele requerir solamente el ratón. Utilizando el cursor del ratón para seleccionar unidades o estructuras, aparecerá una botonera que permite realizar todas las acciones asociadas a dicha estructura o unidad. Es costumbre incluir varios accesos rápidos o Hotkeys, para permitir una mayor rapidez a la hora de realizar acciones. Esto es de gran importancia en el juego competitivo online.

2.2.6 MMO

Los MMO son videojuegos diseñados para permitir el juego simultáneo de una gran cantidad de jugadores online. Buenos ejemplos de esta categoría son World of Warcraft (Blizzard) y Guild Wars (NCSoft). Normalmente todos los jugadores comparten un único y vasto mundo 3D en el que se desarrolla el juego. La enorme cantidad de jugadores que el juego debe soportar condiciona de manera notable el diseño del juego, de tal manera que el detalle gráfico es menor con respecto a otro juego no MMO y se necesita el soporte de una enorme batería de servidores para soportar el número de jugadores conectados de forma simultánea. En la figura 2.11 se puede observar un combate en el Massively Multiplayer Online Role-

playing Game (MMORPG) Lineage II (NCSOFT), en el que se pueden ver varios jugadores colaborando en la pelea.



Figura 2.11: Captura de pantalla de Lineage II (NCSOFT)

2.3 Diseño de un videojuego

En esta sección se señalan diversos aspectos del desarrollo de un videojuego, empezando por la experiencia del jugador, hasta la metodología necesaria para desarrollar un videojuego, así como un análisis de diversas utilidades del mercado.

2.3.1 Las emociones como parte del videojuego

«Los juegos son una serie de elecciones interesantes.» Sid Meier

El primer paso para el diseño de un videojuego es entender como afectan las emociones del jugador al juego. Si un juego es una serie de decisiones interesantes, es necesario que los diseñadores de juegos entiendan que es lo que hace esas decisiones interesantes. Para mantener el interés y motivación del jugador, los juegos suelen recurrir a estrategias como reducir las consecuencias negativas de las acciones del jugador, mientras se maximiza el feedback enviado al jugador, con las que generar emociones positivas en el jugador.

Como se indica en [2] las emociones cumplen cinco roles diferentes que pueden aumentar el grado de afección del jugador con el videojuego:

Disfrute: El jugador disfruta las sensaciones creadas por las emociones.

Fijar la atención: Las emociones afectan a la atención y concentración del jugador. Así, la emoción de afrontar una situación difícil o tensa, normalmente fija la atención del jugador más en el videojuego que una situación carente de riesgo alguno.

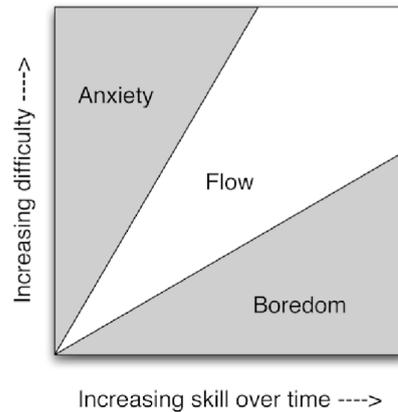


Figura 2.12: Gráfico Flow

Toma de decisiones: Las emociones ayudan en la toma de decisiones. Por ejemplo, es más probable que el jugador elija como ayudante en un juego a un personaje que sea de su agrado, que a otro personaje con el que no siente afinidad.

Rendimiento: Las emociones negativas (como la frustración) llevan al jugador a realizar las acciones que el juego recompensa de forma más positiva, mejorando su rendimiento en el juego.

Aprendizaje: Las emociones recompensan y motivan el aprendizaje.

El psicólogo Mihály Csíkszentmihályi, realizó un estudio sobre las emociones que generan diferentes actividades, en los seres humanos, encontrando que las actividades que generaban emociones más satisfactorias son aquellas con objetivos claros y un equilibrio perfecto entre la dificultad de la actividad y la habilidad que se gana con su realización. Este autor denominaba Flow al equilibrio entre dificultad y habilidad en el que una persona se siente recompensada y aumenta su filiación con la actividad realizada. En el siguiente diagrama se puede observar un gráfico, en el que se aprecian las tres diferentes zonas en las que se puede encontrar una actividad.

Así, es importante graduar correctamente la dificultad de un videojuego, así como que el jugador vea incrementada su habilidad con el mismo, para que no caiga en la frustración o en el aburrimiento.

2.3.2 Equipo de desarrollo de un videojuego

En el desarrollo de un videojuego profesional, el equipo involucrado varía sustancialmente dependiendo del alcance de mercado del videojuego. Así, puede variar desde una única persona en un desarrollo independiente a cientos de personas en una superproducción. En un equipo estándar de desarrollo, suele haber diferentes roles:

Diseñador

Muchos proyectos de desarrollo tiene un desarrollador líder, responsable del guión del juego. En dicho guión se especifica de forma global los aspectos funcionales del juego, sin entrar en detalles técnicos. Así, el diseñador del juego será el encargado de realizar el equivalente al guión de una película, detallando las acciones permitidas al jugador, qué efecto en general tendrán sobre el entorno, qué objetivos primarios y secundarios tiene el juego, etc...

Programador

Los programadores del juego son obviamente responsables de escribir el código fuente del juego. Además, es su función proporcionar pequeñas herramientas que ayuden al resto del equipo de desarrollo a integrar su trabajo con el juego, como pueden ser editores de niveles, modos de testeo en el juego para probar modelos 3D, etc... Los programadores también son los encargados de realizar la integración final de todos los elementos del videojuego, por lo que se requiere una comunicación fluida con el resto de roles, para asegurarse de que el resultado final es el deseado. No es poco frecuente que los programadores se especialicen en áreas concretas del desarrollo de videojuegos. Podemos destacar los siguientes perfiles generales, aunque pueden existir muchos más:

Programadores Gráficos Los programadores gráficos son los encargados de crear el código necesario para la visualización de los elementos gráficos que conforman el universo del juego. En el caso de que el juego no esté utilizando un motor de juego ya existente y se esté creando uno propio, también serían los encargados de realizar la parte gráfica de dicho motor de juego.

Programadores IA Se responsabilizan de crear un sistema de decisiones para la IA, que permite reaccionar de manera acertada a las acciones del jugador. Normalmente suelen trabajar con lenguajes de alto nivel que suelen ser interpretados, para poder realizar pruebas rápidas sin necesidad de recompilar el proyecto.

Programadores de sonido Los programadores de sonido trabajan de forma estrecha con el equipo artístico de sonido para conseguir un entorno sonoro agradable al usuario.

Programadores de Herramientas Se encargan de realizar pequeñas utilidades para facilitar el trabajo del resto de los equipos. Ejemplos de dichas utilidades son editores de mapas, herramientas de testeo rápido, etc..

Equipo artístico de sonido

Los efectos de sonido y la música son componentes fundamentales para que la experiencia de juego sea agradable al jugador. Un efecto de sonido que no esté bien integrado, o una música que no sea apropiada, pueden reducir la inmersión del jugador en el juego, o directamente, arruinar la experiencia de juego.

Músicos Los músicos son los encargados de crear la música del videojuego. Normalmente es un rol que no requiere dedicación completa, ya que la duración de su trabajo suele ser mucho menor que la duración del proyecto global de desarrollo.

Ingenieros de sonido Dependiendo de la composición del equipo de desarrollo, un músico puede encargarse también de los efectos de sonido, o bien tener personal especializado para esta tarea. Es muy común adquirir los efectos de sonido a otra empresa para incluirlos directamente en el juego, evitando el coste extra de tener que contratar ingenieros de sonido.

Artistas

Los artistas son los componentes del equipo que crean los recursos gráficos del videojuego. Normalmente se especializan en un área en concreto, como pueden ser modelado 3D, texturizado 2D, tipografías, etc... Es muy común que los artistas y los programadores tengan problemas entre ellos, debido a la diferencia de objetivos de cada colectivo. Así, un artista podría querer aumentar el número de polígonos de un modelo para añadirle más detalle, mientras que el programador estaría en contra, ya que más polígonos significaría que el rendimiento del juego empeoraría. Aunque el hardware avanza cada vez más deprisa, se espera del equipo artístico la máxima optimización de sus recursos, de tal manera que no sólo el resultado visual se correcto, si no que no se resienta el rendimiento. Dependiendo del tipo de tarea de la que se ocupan los artistas, pueden dividirse en tres tipos:

Animadores Los animadores son uno de los trabajos más solicitados en la industria del videojuego. Su trabajo es construir todos los modelos 3D que tengan movimiento en el videojuego. Partiendo de un bosquejo (o una breve descripción) del diseñador jefe, deben ser capaces de crear un modelo 3D que se ajuste a esas especificaciones. Trabajan con programas 3D como Maya, 3D StudioMax o Blender. Normalmente combinan formas primitivas, sobre las que realizan diversas operaciones hasta obtener la forma del objeto deseado. También son responsables de crear las animaciones necesarias de dichos objetos, según los requerimientos del juego. Es muy común que para realizar animaciones de modelos que existen en el mundo real (como personas, animales, vehículos, etc...) se documenten observando las contrapartidas reales de dichos modelos, para intentar reproducir con fidelidad los movimientos. Es posible, dependiendo del tipo de videojuego, que tengan que realizar animación facial, sea para adecuarla a la dicción de los personajes, o para representar emociones de los mismos. Suelen tener también a su cargo la realización de los vídeos introductorios del videojuego, o los vídeos de los trailers del mismo, si los hubiera. En estos vídeos, el animador no está restringido por los límites de capacidad de hardware del dispositivo objetivo del videojuego, ya que el renderizado del vídeo no se realizará en tiempo real.

Modeladores 3D Se encargan de modelar el mundo en el que el videojuego tiene lugar. Realizan escenarios para el juego, tanto de exteriores (un bosque), como de interiores (una habitación). Son los encargados de la iluminación ambiental.

Texturizadores A diferencia de los animadores y los modeladores, los texturizadores trabajan con herramientas 2D. Es el encargado de crear las imágenes que después envolverán los modelos 3D, dotándolos de detalle

Productor

El productor se encarga de la gestión global del proyecto, intentando que todo funcione. También es función suya mediar en los posibles conflictos que surjan entre los diferentes equipos.

Betatester

Los betatesters son las personas que realizan pruebas sobre la versión en desarrollo del juego para poder detectar posibles bugs. Es una de las labores más infravaloradas, a pesar de su importancia capital a la hora de conseguir un videojuego fluido, con sentido, y sin fallos. Es muy importante que no sean betatester de los juegos aquellas personas que han intervenido en su desarrollo, ya que sus pruebas estarían condicionadas de forma inconsciente por su trabajo. Unas pruebas bien realizadas ahorran mucho tiempo al equipo de desarrollo, ya que los errores son detectados mucho antes, y el coste de corregirlos es mucho menor, que si son detectados por los jugadores.

Playtester

Los playtesters son muchas veces confundidos con los betatesters. Sin embargo, un playtester solamente prueba una funcionalidad concreta del juego, en un escenario acotado. Por el contrario, un betatester prueba el juego de manera global. La función de playtester muchas veces es cubierta por miembros del equipo de desarrollo, debido normalmente a la falta de tiempo disponible para realizar las pruebas.

2.3.3 Motores de juego

Los motores de juego son entornos de programación, ya sea en forma de IDEs, librerías o rutinas de programación, que facilitan el diseño y desarrollo de videojuegos. En el mercado existen una amplísima variedad de motores de juego. El concepto de motor de juego surgió en los 90 de la mano del prolífico diseñador de juegos John Carmack durante el desarrollo del famoso videojuego Doom, cuando trabajaba para la empresa Id Software. El motor creado se llamó Doom Engine o id Tech1. Dicho motor, fue revolucionario para la época, aunque contenía también serías limitaciones.

- No presenta realmente un entorno 3D, ya que no es posible incluir habitaciones sobre

habitaciones. Aunque era una limitación importante, también tenía sus ventajas, ya que facilitaba mucho el dibujado de un minimapa, por no tener que representar diferentes alturas. También hizo que proliferaran los editores de niveles con vista cenital, siendo Doom uno de los primeros juegos en el que los usuarios podían diseñar mapas propios.

- Uso del sistema Binary Space Partition (BSP) para organizar el renderizado de los objetos que deben aparecer en pantalla, evitando el proceso de objetos que no aparecen en pantalla por estar demasiado lejos u ocultos tras otro objeto más cercano a la cámara.
- No es posible que el jugador mire hacia arriba o hacia abajo, debido a que las paredes siempre se visualizan verticales.

El código fuente de del Doom Engine fue liberado bajo licencia GNU Public License (GPL) en 1999. Actualmente, Id Software ha terminado el desarrollo de la versión 6 de dicho motor, siendo Doom (Versión 2016) el primero juego en incorporarlo.

Unreal Engine 4

Unreal Engine es un motor de juego para PC y consolas creado por la compañía Epic Games para el juego Unreal en 1998. Es uno de los motores de de juego más utilizado profesionalmente. Actualmente el motor de juego se encuentra en su versión número 4. Algunas de sus principales características son, según [14]:

- BluePrint Scripting. Permite la definición de entidades del juego sin necesidad de programarlas con código. En vez de eso, se utiliza un editor gráfico que permite enlazar y relacionar de diferentes maneras las propiedades de dicha entidad (Comportamientos, aspecto visual, etc..) . Además, diferentes BluePrints pueden comunicarse entre ellos de forma sencilla. Gracias a los BluePrints es posible realizar prototipos rápidos de juegos de forma sencilla. También facilita la integración el equipo artístico en la parte técnica del juego, al no requerir conocimientos técnicos avanzados.
- Soporte para todas las nuevas funcionalidades de DirectX 11 y 12.
- Efectos visuales en cascada, que permite realizar sistemas de partículas de forma sencilla, teniendo un bajo coste para la CPU.
- El editor postural Persona Animator permite previsualizar animaciones de mallas con esqueletos, así como otras muchas funcionalidades, como alterar la física de dichas mallas.
- Acceso al código fuente. Aunque no es un Open Source, todo el código fuente de Unreal Engine 4 está disponible desde su repositorio de GitHub (<https://github.com/EpicGames>). El tener acceso al código fuente permite poder consultar el código cuando la documentación encontrada no es suficiente, o realizar extensiones propias al motor de juego.
- Recarga de código en caliente. Unreal Engine 4 permite realizar en cambios «en caliente» en código C++. Es posible ver el efecto del cambio realizado sin necesidad de

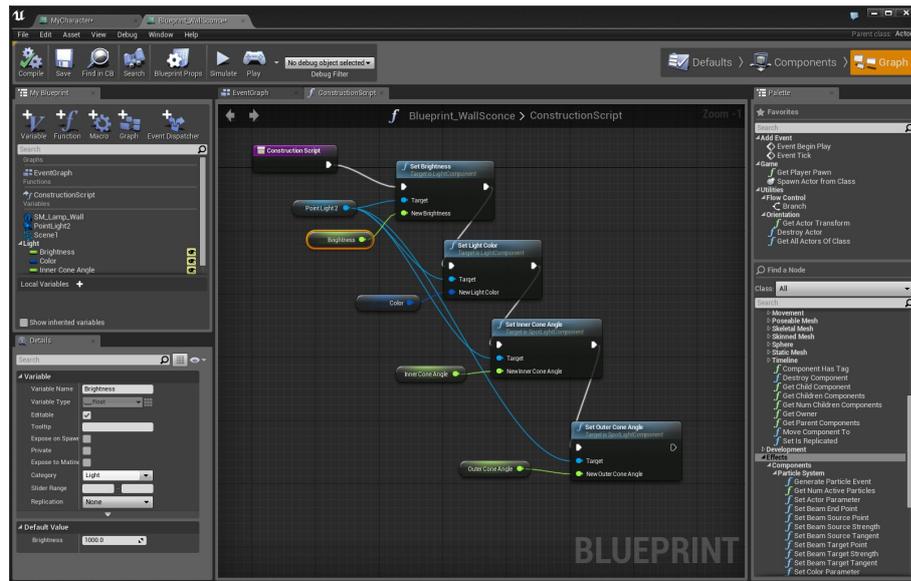


Figura 2.13: BluePrint Editor

relanzar el juego.

- «Gratis». Unreal Engine 4 es gratuito siempre y cuando el juego realizado con el no supere los 3000\$ de beneficio [15]. A partir de esa cifra, deberá pagarse una tasa del 5 % de los beneficios obtenidos.

Unity

Unity[29] es un motor de juego que goza de una gran popularidad. Permite realizar videojuegos multiplataforma, soportando una gran cantidad de dispositivos diferentes como Windows, OS X, Linux, Xbox 360, PlayStation 3, Playstation Vita, Wii, Wii U, iPad, iPhone, Android y Windows Phone.

A continuación se detallan algunas de sus características:

- Curva de aprendizaje sencilla. La manera en la que Unity hace un extensivo uso del Drag & Drop en la interfaz de su editor, hace que trabajar con él sea fácil y muy intuitivo.
- Gran cantidad de recursos. Debido a su popularidad Unity es probablemente el motor de juego con más recursos disponibles (en su mayoría de pago) para el desarrollador. Es posible comprar Assets a un precio asequible.
- Scripting en C# y Javascript. El poder trabajar con lenguajes de muy alto nivel, como C#, Javascript o, en menor medida Boo, permite reducir las líneas de código necesarias para la codificación del juego.
- Gratis. La versión Personal de Unity es gratuita, siempre y cuando el videojuego no supere los 100.000\$ de beneficio.



Figura 2.14: Editor Unity

Ogre3D

Ogre3D es un motor de renderizado orientado al desarrollo de videojuegos. Si bien Ogre3D no es un motor de juego completo, se ha utilizado como base para el desarrollo de la parte visual de muchos videojuegos. Es OpenSource bajo una licencia MIT. Ogre3d es multiplataforma, cuenta con una comunidad de usuarios, que si bien no es demasiado amplia, si es bastante activa. Actualmente la última versión estable es la versión 1.9, encontrándose la versión 2.1 en desarrollo. Las principales características de Ogre3D son:

- Sistema de plugins, que permite no incluir en el desarrollo aquellas funcionalidades que no son necesarias para el videojuego.
- Gestión de la escena mediante un grafo, que parte de un nodo principal denominado rootNode. Dicho grafo se utiliza para diversas funcionalidad, como obtener una jerarquía en la aplicación de transformaciones a los nodos de la escena, así como permitir realizar búsquedas en los nodos del árbol a la hora de realizar consultas de colisión o de trazado de rayos.
- Genera ejecutables relativamente pequeños. Un ejecutable creado con Ogre3D enlazado de forma estática, tiene un tamaño aproximado de unos 10 Mb.
- Ogre3D carece de un editor integrado como Unity o Unreal Engine. El programador debe utilizar herramientas de terceros tanto para el diseño 3D de los objetos del videojuego, como para la edición del código fuente del mismo.

Capítulo 3

Objetivos

En este capítulo se especifican los objetivos que se desean conseguir con la realización del PFC.

3.1 Objetivo general

El objetivo de este PFC es el diseño e implementación de un prototipo de videojuego utilizando software libre. El prototipo debe ser totalmente jugable, y además, debe poder ser portado a varias plataformas diferentes.

3.2 Objetivos específicos

Para una mayor claridad se van a separar los objetivos en dos apartados diferentes:

Objetivos funcionales o de diseño En este subapartado se especificarán los objetivos que son absolutamente independientes de la implementación, es decir, solamente aquellos que afectan a la experiencia de juego.

Objetivos técnicos o de implementación En este subapartado se detallan todos los objetivos que tienen que ver con la implementación del prototipo, teniendo escasa influencia en la experiencia de juego.

Objetivos comerciales En este subapartado se detallan los objetivos relativos a la monetización del juego.

3.2.1 Objetivos funcionales o de diseño

Para conseguir una experiencia de juego lo más satisfactoria posible para el prototipo, se incluyen los siguientes objetivos:

Drag and Drop El juego funcionará con el paradigma Drag and Drop, que es muy utilizado en los dispositivos móviles por su sencillez. Los botones serán de tamaño grande para poder ser utilizados cómodamente en una pantalla táctil.

Sombreado Los objetos que se encuentren en el juego deben arrojar sombras sobre el mapa, para mejorar la experiencia del juego.

Mejora de Torres Las torres que el jugador puede construir para defenderse de los enemi-

gos deben poder evolucionar, mejorando su capacidad defensiva.

Heads-up Display (HUD) sencillo y claro El HUD debe ser sencillo y claro, ya que no se dispondrá de una resolución de pantalla alta, por estar orientado a dispositivos móviles. Deberá mostrar al jugador al menos la siguiente información:

- Tiempo que falta para que la siguiente oleada de enemigos entre en el mapa.
- Dinero que tiene almacenado el jugador.
- Puntos que ha realizado el jugador en la partida actual.
- Número de oleada de enemigos restantes para ganar el juego.
- Número de vidas restantes.

Sonido El juego tendrá efectos sonoros, que incrementarán la experiencia de juego. Se buscarán recursos de audio de licencia Creative Commons para los efectos de sonido.

3.2.2 Objetivos técnicos o de implementación

A continuación se detallan los objetivos que no tienen gran influencia en la experiencia del juego:

Código en C++ C++ es el lenguaje elegido para implementar el videojuego. C++ es una opción excelente para el desarrollo de videojuegos. Es la evolución natural del lenguaje C, que era el lenguaje de referencia para la programación de videojuegos a finales de los años 90 [25].

Modelos Tridimensionales Se emplearán modelos 3D con animación mediante esqueletos asociados a grupos de vértices. Los modelos serán texturizados con UV-Mapping

Multiplataforma El juego debe funcionar en al menos dos plataformas, Linux y Android. Se debe crear un entorno de trabajo tal que permita construir de forma sencilla el juego para ambas plataformas.

Respuesta en tiempo real Este requisito es fundamental para que el videojuego puede tener buena acogida entre el público. Se realizarán las optimizaciones que se estimen necesarias para asegurar la mínima latencia en la respuesta del juego a las acciones del usuario.

Editor de niveles Se realizará un pequeño editor de niveles, que permita diseñar nuevos niveles sin tener conocimientos técnicos o de programación.

Diseño de texturas Se realizarán texturas para los modelos, utilizando algún editor gráfico 2D.

PathFinding Se implementará un sistema de pathfinding que utilizarán los enemigos para poder seguir el camino que atraviesa el nivel. Dicho sistema debe ser rápido para no perjudicar el rendimiento del juego.

Seguimiento de enemigos Se implementará un sistema para que las torres sean capaces de detectar posibles objetivos y orientarse hacia ellos para efectuar un disparo.

Almacenamiento de records Será posible almacenar las mejores puntuaciones de los jugadores, para ofrecer así una motivación extra para que el jugador vuelve a jugar más partidas.

Configurable El juego será lo más configurable posible, de tal manera que sea posible añadir nuevos tipos de torres o de enemigos sin necesidad de volver a construir el proyecto.

Tamaño pequeño Debido a que uno de los objetivos es el desarrollo para plataforma móvil, es necesario que el tamaño de la Android Application Package (APK) no sea excesivo.

Uso de Software Libre La mayoría de las librerías utilizadas deben ser software libre, en especial las más importantes, como el motor de juego o la librería de sonido.

3.2.3 Objetivos Comerciales

Los objetivos comerciales son aquellos que tiene que ver con el cálculo del rendimiento económico que podría obtenerse de una comercialización del prototipo.

Estudio de modelos de negocio Se analizarán las diferentes opciones disponibles actualmente para obtener rentabilidad económica de un videojuego.

Marketing Se analizarán las opciones disponibles para realizar la publicidad del prototipo del prototipo en el mercado en caso de su comercialización.

Calculo de costes Se realizará un cálculo aproximado de los costes asociados al desarrollo del proyecto, contabilizando horas de trabajo, costes de distribución y publicidad e inversión en hardware y software.

Capítulo 4

Método de trabajo

En este capítulo se expone la metodología de trabajo elegida, así como, las herramientas seleccionadas, tanto de software como de hardware para llevar a cabo el prototipo.

4.1 Metodología

La selección de una metodología de trabajo que funcione correctamente cuando el equipo de desarrollo es un única persona es un decisión complicada. Si bien Scrum es una de las metodologías más utilizadas en el desarrollo de videojuegos [20], esta fuertemente apoyada en el trabajo en equipo y en la coordinación de todos los miembros del Scrum.

Sin embargo, se ha decido utilizar una versión de Scrum modificada para equipos de desarrollo de un solo miembro. Tal como se explica en [22], los elementos de metodología Scrum son las siguientes:

Dueño del producto En este caso, la figura de dueño del producto no existe, por lo que es el propio desarrollador será el que priorice las historias, de acuerdo con el director del PFC.

Sprint Se establece una duración de Sprint de tres semanas. Los sprints empezarán un lunes y acabarán un viernes.

Pila de producto Se descomponen los objetivos funcionales y técnicos en varias historias pequeñas. Dichas historias forman la pila de producto.

Planificación de Sprint Se realiza siempre el domingo antes de un nuevo sprint por la tarde. Se elegirán las tareas que conformarán el nuevo sprint, en función de diferentes características.

Retrospectiva de Sprint Se realiza los domingos por la mañana. Se observa si se han cumplido los plazos y cuales han sido los motivos de las posibles desviaciones, si las hay.

Scrum No se realizan scrums diarios, por que no tiene sentido realizar una reunión de equipo con una única persona.

Puntos de Historia Siguiendo una de las alternativas que se muestran en [22] se ha elegido que un punto de historia se equivalente a un día (ideal)-persona. Como día ideal se

define una jornada de trabajo en la que se consigue una concentración máxima y en total ausencia de distracciones. Se estima una jornada de trabajo en 4 horas.

Velocidad del Equipo La velocidad del equipo es la cantidad de puntos de historia que el equipo puede sacar adelante en un Sprint. Como los sprints son de 19 días (tres semanas, menos el último sábado y domingo) y sólo existe una persona en el equipo de desarrollo, la velocidad ideal de equipo serían de 19 puntos de historia. Sin embargo, debe ajustarse con un factor de dedicación, que contemple distracciones, problemas técnicos no planificados u otros factores que puedan reducir la cantidad de trabajo sacado adelante por el equipo. Este factor se ajusta de manera experimental en cada retrospectiva del sprint. Para el primer sprint, se determina un factor de dedicación del 50 %, lo que la velocidad de equipo queda reducida a 9 puntos de historia.

Gráfico de Burndown Este tipo de gráficos se emplean para ir reflejando el avance del Sprint. Sin embargo, debido a que el equipo es de una sola persona, se ha optado por hacer un gráfico de Burndown global para todo el proyecto, en el que se reflejará el progreso global.

4.2 Hardware

El hardware que se ha utilizado para la implementación del prototipo es:

- Ordenador de sobremesa con procesador ADM Athlon 64 X2 Dual Core 6000, 4GB de memoria Ram DDR2 y tarjeta gráfica ATI Radeon 6400.
- Monitor 22' Asus.
- Smartphone Lenovo A806 con 8 núcleos a 1.7 GHz, 2GB de memoria RAM y Android 4.4(KitKat).
- Tablet SPC de 7' con 4 nucleos y 1 GB de RAM. Sistema operativo Android 4.4 (KitKat).

4.3 Motor de Juego

A la hora de elegir un motor de juego, se determinan los siguientes requisitos imprescindibles:

Software Libre Es necesario que el motor se haya publicado bajo alguna licencia que sea catalogada como Software Libre. Se han seguido las recomendaciones de la Free Software Foundation (FSF) [13] respecto a que considerar una licencia de Software Libre.

Codificación en C++ El lenguaje de programación con el que trabaje el motor debe ser C++.

Soporte para Android El motor de juego debe incluir la plataforma Android como una de sus plataformas soportadas.

Soporte Gráficos 3D El motor de juego debe soportar gráficos en 3D mediante OPENGL. Se valorarán adicionalmente los siguientes criterios bajo este apartado:

- Soporte de sistemas de partículas.
- Alto rendimiento
- Documentación disponible.
- Comunidad de usuarios activa.

Wikipedia mantiene una lista bastante completa de motores de juego, con las características soportadas y las licencias que aplican a cada motor de juego [33]. Si se eliminan todos aquellos que no cumplen los criterios mencionados anteriormente, los motores que quedan son: GamePlay3D, Blend4Web, Dim3, Ogre3D y Godot. A continuación se detalla ligeramente cada motor, explicando el motivo por el que ha sido descartado o elegido.

GamePlay3D

GamePlay3D [7] es un motor 3D cuyo desarrollo está liderado por varios miembros del equipo de desarrollo de BlackBerry. Actualmente su última versión estable es la 3.0. Algunas de sus características más importantes son:

- Sistema de Audio 3D integrado.
- Soporte para OpenGL y OPENGL ES
- Soporte para ratón, teclado, táctil y gamepad.
- Permite definir las interfaces de usuario mediante archivos de propiedades.

Aunque es un motor bastante completo, ha sido descartado porque la documentación que existe al respecto es escasa y la comunidad de usuarios es bastante pequeña. Otro factor importante es que no parece haber sido utilizado para la implementación de ningún videojuego importante.

Blend4Web

Con un enfoque diferente al resto de los motores de juego, Blend4Web hace uso de WebGL, una API javascript para renderizado interactivo de gráficos 2D y 3D[1]. Así, la forma que tiene Blend4Web de ser multiplataforma no es hacer cross-compiling para la plataforma objetivo, sino que depende de que la plataforma objetivo tenga un navegador Web con soporte WebGL. Las características más importantes son:

- No necesita ningún plugin de navegador para funcionar.
- Uso de Blender como herramienta primaria de implementación del videojuego..
- Soporte de Audio 3D.
- Soporte de físicas y detección de colisiones.



Figura 4.1: Captura de Pantalla de Rush Hour Hero (implementado con Gameplay3D)

Un enfoque totalmente original y la portabilidad sin tener que preocuparse de otras plataformas la hacen una elección interesante. Sin embargo, el rendimiento todavía parece estar lejos de los motores de juegos tradicionales, por lo que ha sido descartado.

Dim3

Dim3 utiliza un enfoque algo parecido a Blend4Web. También soporta WebGL para conseguir que sus juegos sean portables. Aunque tiene algunas características que los diferencian:

- Dim3 tiene su propio editor de proyectos, en el que se encuentra organizada toda la información de modelos 3D, animaciones, etc..
- Dim3 posee también un editor de animaciones integrado.

Aunque parece que Dim3 tienen un mejor rendimiento que Blend4Web, tiene una comunidad de usuarios realmente pequeña y no existe una documentación que acompañe al motor de juego. Tampoco ha sido utilizado en juegos de gran envergadura.

Godot

Godot es un completo motor de juego desarrollado por una comunidad de usuarios. Actualmente se encuentra en la versión estable 2.1. Ofrece un completo IDE (ver figura 4.4 que permite la gestión de todo los elementos que conforman un videojuego, desde modelos 3D, sonido, gráficos 2D, etc...

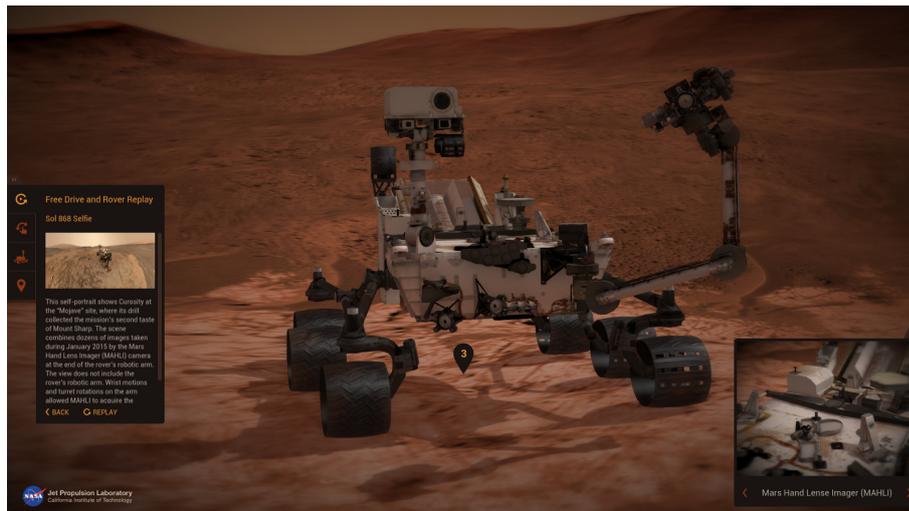


Figura 4.2: Captura de Pantalla de Experience Curiosity, desarrollado por NASA Jet Propulsion Laboratory (implementado con Blend4Web)

Ha sido ampliamente utilizado por la industria de videojuegos de América Latina, antes de ser liberado y que la comunidad de usuarios se encargaron de su mantenimiento y evolución. Soporta OSX, Windows y Linux, y es capaz de generar videojuegos para dichos sistemas y otros muchos más, entre los que se encuentra Android. Características remarcables de Godot son:

- Edición en vivo. Es posible pausar el juego y realizar ciertas modificaciones al mismo con el editor. Dichas modificaciones son permanentes, es decir, se traducen al código fuente. Es una funcionalidad muy interesante para realizar ajustes de grano fino.
- Editor visual de shaders para 2D y 3D.
- Editor visual de animaciones.

Godot es un motor de juego muy interesante, muy visual gracias a su editor integrado, que permite una gestión global del proyecto y de cada una de sus partes de forma sencilla.

Por lo que Godot ha sido descartado, es simplemente por la ausencia de documentación física seria (es decir, libros técnicos) sobre el motor. Sin embargo, Godot puede ser una excelente elección.

Ogre3D

Ogre (u Ogre3D) no es en realidad un motor de juego, sino uno de los motores de renderizado de código abierto más grandes que existen [21]. Esto quiere decir que no incluye funcionalidades que si incluiría un motor de juego estándar. Por ejemplo:

- Ogre3D no incluye soporte de sonido de ninguna clase.
- Tampoco incluye un motor de físicas.



Figura 4.3: Captura de Pantalla de Scruffy, desarrollado por Bran Barnes, autor de Dim3)

- No dispone de una interfaz especializada para la gestión de la IA.
- No incluye la gestión de la entrada del usuario. No es capaz de gestionar teclados, ratones, joystick ni cualquier otra entrada del usuario.

Sin embargo, Ogre tiene unas características que son únicas respecto al resto de motores analizados:

- Es muy maduro. Ogre3D empezó en el 2001 y se ha mantenido en constante desarrollo hasta la actualidad.
- Tiene una comunidad de usuarios muy amplia y activa.
- Ha sido utilizado en algunos juegos profesionales de alta categoría, como Torchlight [16] (ver figura 4.6).
- Tiene un excelente rendimiento, tanto en PC como en plataforma móvil.

Por estos motivos Ogre3D ha sido la opción elegida para la implementación del prototipo.

Puesto que Ogre3D carece de todas las funcionalidades necesarias para ser un motor de juego, es necesario complementarlas con otras librerías. Así, se han seleccionado las siguientes:

Boost Es una librería multipropósito programada en C++. Tiene la ventaja de que en su inmensa mayoría se compone de cabeceras, por lo que no es necesario compilarla casi nunca [5].

Simple DirectMedia Layer (SDL) SDL es una librería multimedia multiplataforma, que fa-

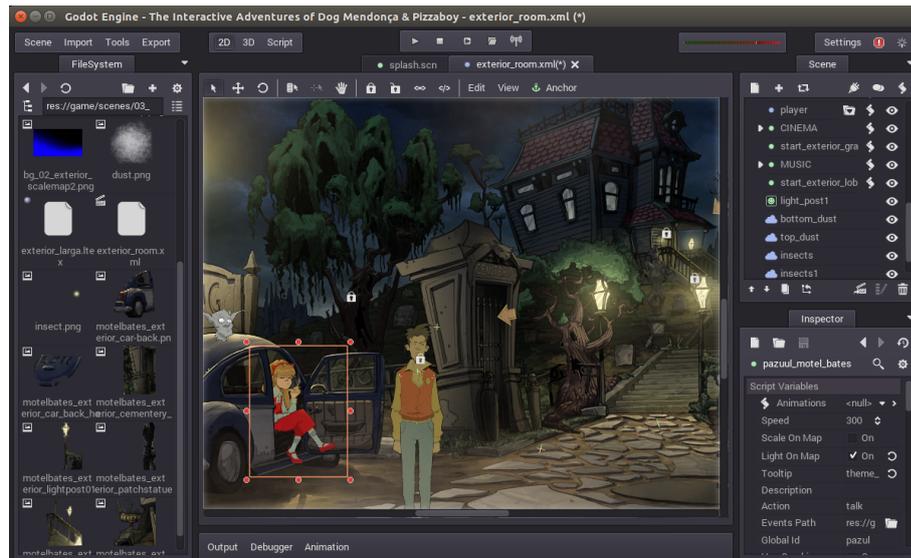


Figura 4.4: Captura de Pantalla del editor de Godot

cilita el uso al hardware de vídeo, sonido, teclado, ratón, etc... [23].

4.4 Otras herramientas de Software

A continuación se detallan otras herramientas de software que han sido necesarias para el desarrollo del PFC.

4.4.1 Sistema Operativo

El desarrollo se ha realizado íntegramente bajo una distribución ArchLinux del sistema GNU/Linux [32]. Si bien su carácter de Rolling Release puede hacerla algo inestable, su gran documentación, así como el amplio repertorio de software no oficial AUR, permite que cumpla como entorno de desarrollo de forma aceptable.

Buscando un entorno ligero en el que primara la economía de recursos y la sencillez, se ha optado por utilizar OpenBox como gestor de ventanas, apoyado en componentes del escritorio LXDE, que permite una interfaz fácil, sencilla y que consume pocos recursos. Esto permite que el entorno arranque de forma rápida y ocupe poca memoria.

Puesto que OpenBox no incluye ningún gestor de archivos propio, se ha seleccionado el gestor de archivos PCManFm, también de la distribución LXDE. PCManFm se caracteriza por su ligereza, rapidez y sencillez, lo que lo hace un candidato ideal, pues está perfectamente alineado con los objetivos que desean obtener con el presente entorno de trabajo.

4.4.2 Emacs

Emacs es uno de los editores más conocidos en sistemas operativos GNU/Linux (aunque está disponible en diversas plataformas más). Es un editor avanzado, autodocumentado,



Figura 4.5: Captura de Pantalla de Steam Quest, desarrollado por Kívano (implementado con Godot)

personalizable y extensible [27]. Se ha elegido este editor en vez de un IDE de desarrollo como puede ser NetBeans o CodeBlocks, para buscar una vez más la economía en recursos utilizados (ver figura 4.7).

Emacs tiene una configuración de teclado muy particular, que es una de los motivos por los que la curva de aprendizaje del mismo es bastante pronunciada. Al estar enfocado al uso sin ratón, es necesario familiarizarse con dicha configuración de teclado para poder hacer uso de todo el potencial del editor.

Al ser un editor extensible, Emacs cuenta con una amplísima variedad de extensiones, que permiten reflejar aquellas características de un IDE pesado que se deseen de forma independiente, teniendo de esta manera un completo control sobre el equilibrio recursos-funcionalidad del editor.

Por defecto, Emacs incluye varios modos mayores, que son los encargados de definir ciertos comportamientos del editor que aportan utilidad y usabilidad al trabajo para un tipo concreto de lenguaje. Así, existen modos mayores, para C++, Java, XML, etc... Estos modos mayores otorgan cierta funcionalidad avanzada como por ejemplo la autoindentación.

Sin embargo, para mejorar la usabilidad y las funcionalidades ofrecidas por el modo mayor de C++ en Emacs, se ha optado por instalar varias extensiones, de las cuales las siguientes son las principales:

Projectile Projectile permite el manejo de forma sencilla de proyectos complejos en Emacs.

Ofrece funcionalidades como el acceso rápido a archivos dentro del proyecto buscándolos por el nombre, o la posibilidad de ejecutar grep sobre todos los archivos del proyecto.



Figura 4.6: Captura de Pantalla de TorchLight, desarrollado por Runic Games (implementado con Ogre)

Yasnippet Yasnippet es un sistema de plantillas, que realiza una función rápida y sencilla de asistente de contenido. Así, ofrece funciones de autocompletado, analizando los diferentes archivos que se han abierto en Emacs.

FlyCheck Flycheck es un analizador sintáctico que es capaz de mostrar al usuario los errores de sintaxis al vuelo, es decir, mientras el usuario va escribiendo código. Soporta más 30 lenguajes de programación diferentes.

Clang-format Es una pequeña extensión que permite ejecutar la utilidad clang-format, que realizar un formateo inteligente del código, corrigiendo identaciones incorrectas y código mal formateado.

AUCTex Es un plugin que ofrece ayuda contextual y funciones de indentación, entre otras muchas funcionalidades, para la edición de documentos \LaTeX .

Aunque existe una extensión especialidad de dotar de la funcionalidad de un entorno de desarrollo pesado a Emacs, conocida como CEDET, se ha desechado su uso debido a que ralentiza mucho el editor, lo que va en contra del objetivo de un entorno ligero para el desarrollo.

Señalar que también se ha utilizado Emacs para la edición de este documento en \LaTeX .

GNU Compiler Collection (GCC)

GCC en su versión 5.3.0 es el compilador estándar para el sistema operativo GNU/LINUX. Solamente se ha utilizado para las compilaciones para GNU/Linux, ya que para Android es necesario utilizar la versión que viene incluida con el NDK de Android.

```

File Edit Options Buffers Tools C++ YASnippet Help

bool _keepLiving;
/**
 * Lives of the player
 */
int _lives;
/**
 * Money of the player
 */
int _money;
/**
 * Player score
 */
int _score;
/**
 * Time since Last Message is Showed (to check if its time to hide it)
 */
float _timeSinceLastMessage;

private:
/**
 * Create Scene Map
 */
void createScene();
/**
 * Create Light
 */
void createLight();
/**
 * Draw HUD
 */
void createHUD();
/**
 * Calls when a Hud element is clicked.
 * @param element: HudElement: One id clicked
 * @param SDL_Event Event that store the click data
 */
bool hudElementClicked(QGraphicsOverlayElement *element, const SDL_Event *e);
/**
 * Checks the hit of the projectiles thrown by the towers
 * @param deltaTime Time elapsed since previous call
 */
void checkTowerHit(float deltaTime);
}

PlayState.h 57% L121 | GEMaster | C++ | A YASnippet | TOWER3D | AC PYC110 | Abbrev

```

Figura 4.7: Captura de Pantalla de Emacs. Se puede observar como están activadas las extensiones de FlyCheck, Yasnippet y Projectile.

GIMP

GIMP es un programa de edición de imágenes digitales en formatos no vectoriales. GIMP empezó en 1995 como un proyecto de dos estudiantes de universidad. Actualmente, GIMP es una aplicación muy completa con muchas funcionalidades valiosas, que es distribuido de manera gratuita [24].

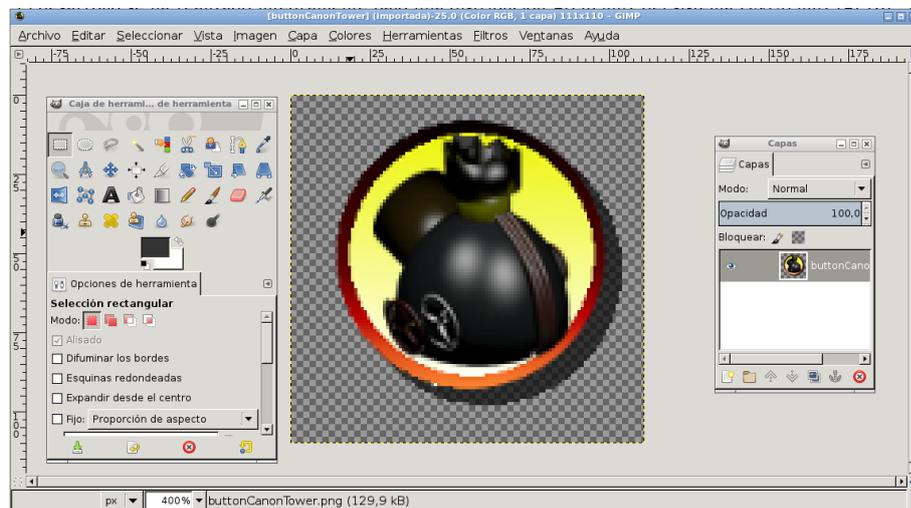


Figura 4.8: Captura de la edición de uno de los botones en GIMP.

GIMP ha sido utilizado en este proyecto para cualquier trabajo que requiriera edición de imágenes, desde creación de texturas (su función principal en este proyecto) a la creación de elementos de la interfaz de usuario, con botones o logotipos. Las principales características de GIMP que han sido relevantes para este proyecto son:

Multicapa GIMP gestiona de manera sencilla el uso de múltiples capas para facilitar el dibujo

jado. Esto permite restringir el uso de las diferentes herramientas de pintado, selección, filtros a un ámbito definido, sin que afecte al resto del dibujo. También da la posibilidad al usuario de elegir como quiere combinar dichas capas, ajustando el nivel de transparencia, y permitiendo elegir que capas han de posicionarse al frente de la imagen, y cuáles deben posicionarse detrás.

Filtros GIMP incluye una amplia variedad de filtros, que permiten realizar efectos sobre la imagen. Filtros como el desenfoco de movimiento, que se ha utilizado para generar las vetas de metal del material con el que están hechos algunos modelos 3D. Otro filtro muy utilizado ha sido el de biselado, para dotar a ciertos botones e iconos de una mayor sensación de profundidad.

Gradientes El sistema de generación de gradientes de GIMP, permite generar una variedad enorme de gradientes diferentes, de forma muy sencilla, consiguiendo un acabado más profesional de los elementos 2D de la interfaz de usuario.

Blender

Blender es un software de diseño 3D. Utilizado para realizar películas profesionales como Bick Buck Bunny o Sintel, Blender tiene características que lo hacen la mejor alternativa Open Source para realizar diseñar objetos 3D, y animarlos. Su carácter OpenSource y la activa comunidad de usuarios que posee, consigue que existan una amplia variedad de plugins, normalmente escritos en Python, que permiten extender el funcionamiento del programa para realizar una enorme variedad de tareas.

En es proyecto Blender ha sido la herramienta elegido para diseñar todos los elementos 3D incluidos en el juego, como por ejemplo las torres o los mapas. También se ha utilizado Blender para realizar todas las animaciones del juego.

En particular se ha hecho amplio uso de la opción de cinemática inversa que ofrece Blender para la generación de animaciones a partir de esqueletos (ver figura 4.9). Esta opción permite al usuario especificar una serie de pasos de tiempo (keyFrames) en las que especifica una pose objetivo del esqueleto, siendo Blender el encargado de calcular, teniendo en cuenta las articulaciones del esqueleto del modelo, la animación necesaria para pasar de una pose a la siguiente [12].

Android NDK y SDK

Para el desarrollo en la plataforma Android, es necesario tanto el SDK de Android (que permite el desarrollo en JAVA), como el NDK, que permite el desarrollo de software nativo, en C y C++.

Aunque en la documentación de Ogre3D, se menciona la versión 8b del NDK, el desarrollo de esta aplicación se ha realizado con la versión 10e del NDK, necesaria para dar soporte a versiones más actualizadas de Android.

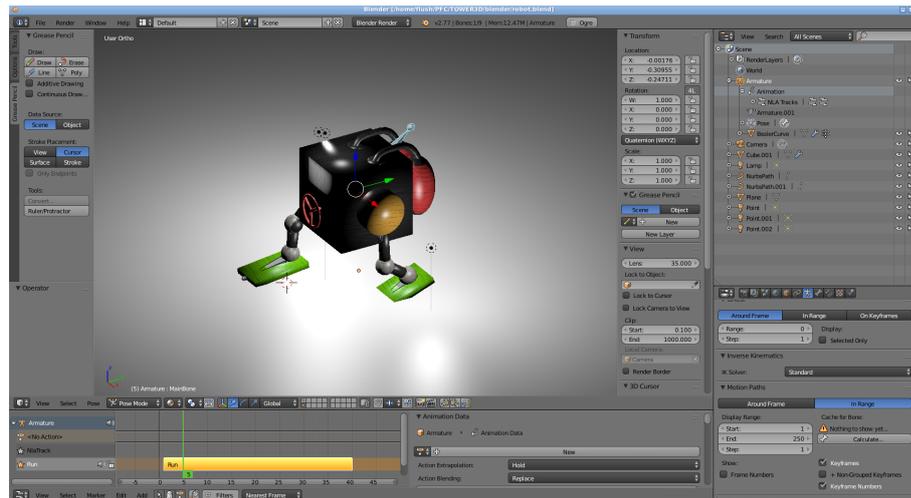


Figura 4.9: Animando un enemigo por cinemática inversa en Blender.

Para el SDK, se ha elegido la versión 1.5.

Para el cross-compiling necesario para compilar OGRE3D y sus dependencias se ha elegido Cmake en su versión 3.5.2.

Para la compilación para la plataforma Android, el NDK incluye su propia versión de GCC y su propia sistema para gestionar las dependencias y los diferentes flags de compilación y linkado necesarios.

Capítulo 5

Resultados

En este capítulo se muestran los resultados logrados con este PFC, así como las técnicas que ha sido necesario utilizar para llegar a los mismos. Para ello se describirán varias fases, desde la ejecución

5.1 Diseño del videojuego

En esta sección se trata de explicar, sin entrar en detalles técnicos, los detalles funcionales del videojuego. Siguiendo la estructura marcada por [3] sobre los documentos necesarios para el desarrollo de un videojuego, aunque se realizarán versiones reducidas de los mismos.

5.1.1 Desarrollo del concepto

En este apartado se incluyen breves explicaciones del concepto del videojuego, proporcionando pequeños ejemplos de juegos ya existentes, que ayudan a concretar de forma sencilla el concepto subyacente del videojuego. Los diferentes subapartados son:

El concepto básico Una explicación breve de apenas dos líneas, en las que debe esbozarse el concepto del juego.

La propuesta del juego Se expone de qué va el juego, por qué puede ser exitoso y como se obtendrá rentabilidad económica del mismo.

GamePlay Se describe que acciones puede realizar el jugador en el juego.

El concepto básico TOWER3D será un juego de tipo Tower Defense, en el que el jugador se defenderá de hordas de enemigos que atraviesan un mapa siguiendo un camino, colocando y mejorando torres defensivas a lo largo de un mapa.

La propuesta del juego En este apartado se exponen las principales características del juego, que permiten obtener una idea bastante aproximada del concepto:

- TOWER3D estará orientado a plataformas móviles. Esto permitirá reducir costes en el arte gráfico, así como poder facilitar la distribución del juego a un número mayor

de jugadores que en plataformas sobremesa tradicionales. De esta manera, el gasto de distribución del juego también se reduce.

- TOWER3D es un juego sencillo, que debe pretender ser usado como juego casual. Así, su público objetivo no serán necesariamente jugadores tradicionales, sino un ‘target’ mucho más amplio. Las partidas no durarán más de 15 minutos usualmente, así que no será necesario retener mucho tiempo la atención del jugador.
- El nivel de dificultad estará cuidadosamente equilibrado, para que al jugador le resulte un reto, permitiendo un efecto de enganche que haga al jugador volver a jugar una partida en cualquier tiempo de ocio disponible, aunque sea breve.
- Es importante que el juego no ocupe mucho espacio en la memoria del dispositivo que se instale, ya que se busca que el juego no sea elegible por el usuario para desinstalar a la hora de liberar memoria de su dispositivo.
- Tendrá una ambientación futurista de estilo Steam Punk (ver figura 5.1) simplificada, que es vistosa y a la vez relativamente sencilla de modelar.
- El jugador dispondrá de diferentes modelos de torre, para que pueda realizar diferentes estrategias.
- Las torres podrán ser mejoradas, para que el jugador tenga una sensación de progreso a lo largo de la partida.
- Los records serán almacenados en el dispositivo, de tal manera que el jugador tenga una motivación para volver a jugar, para superarse a si mismo.

GamePlay Una vez empezado una partida de TOWER3D, el jugador dispone de un tiempo fijo para realizar sus acciones antes de que empiecen a aproximarse enemigos al mapa.

Acabado dicho tiempo, empezarán a aparecer enemigos en el mapa, que irán siguiendo el camino que cruza dicho mapa, hasta llegar al otro lado.

Cada enemigo que consiga cruzar al otro lado del mapa, resta una vida al jugador (que empieza con una cantidad de vidas iniciales). Si el jugador se queda sin vidas, pierde. Para intentar evitar que los enemigos crucen el mapa, el jugador puede realizar determinadas acciones. Todas las acciones que puede realizar el jugador le cuestan dinero, del que dispone una cantidad inicial. El jugador obtendrá dinero adicional cada vez que una de sus torres acabe con un enemigo. El dinero otorgado por cada enemigo destruido es diferente, según el tipo de enemigo. Con ese dinero, el jugador puede construir torres defensivas. Dichas torres funcionan de manera autónoma, es decir, el jugador sólo debe situarlas en el mapa. Para construir las torres, el jugador dispone de unos iconos en la parte inferior derecha (figura 5.2). Si el jugador arrastra un icono a una posición válida del mapa (una que no esté en un camino y que no contenga otra torre), una nueva torre del tipo correspondiente al icono arrastrado será construido.



Figura 5.1: Dos robots de ambientación Steam Punk

Una vez construidas esas torres, serán capaces de girar sobre sí mismas, apuntando al primer enemigo que entre dentro de su radio de alcance, y disparándole hasta acabar con el o hasta que salga de su radio de alcance. Las torres pueden dañar más de un enemigo con un único proyectil, si estos enemigos están lo suficientemente juntos.

El jugador puede seleccionar cada torre pulsando sobre ella. Al hacerlo, aparecerá un panel informativo con las características de dicha torre, además de un botón que al pulsarlo, permite al jugador mejorar las características de dicha torre, pagando una cantidad de dinero.

Cada torre tiene una serie de niveles de mejora disponibles, siendo imposible mejorarlas cuando llegan al nivel más alto.

El juego se compone de varias oleadas de enemigos. Una vez finalizada una oleada de enemigos, el jugador dispondrá de un tiempo definido hasta que empiece la siguiente oleada. Cada oleada puede componerse de uno o varios tipos de enemigos. De cada tipo puede haber



Figura 5.2: Iconos de construcción de torres

un número diferente de enemigos.

5.2 Preparando el entorno

En este apartado se describe la preparación del entorno de desarrollo. Es necesario un apartado propio en el documento, ya que el entorno de desarrollo tiene muchas particularidades.

5.2.1 Instalación de Android NDK y SDK

Aunque en [10] se comenta que se ha probado con la versión 8b, es una versión muy antigua que no da soporte a las versiones actuales de Android. Este prototipo se ha realizado con la última versión disponible (en el momento en el que se empezó el desarrollo del mismo), la versión 10e. Es importante no instalar la versión de x64, porque el toolchain de Ogre3D no funciona correctamente con esta versión.

Es importante también decidir qué versión será el requisito mínimo del prototipo. Es necesario conocer también el nivel de API que corresponde a dicha versión, puesto que será uno de los parámetros necesarios a la hora de generar la aplicación para Android. En la figura 5.3 se puede observar la relación entre el nivel de API y la versión de Android.

En [17] está publicada una tabla que relaciona cada versión de Android su API correspondiente.

Para el prototipo se escogió la versión de Android 4.2, que corresponde al nivel de API 17.

5.2.2 Compilando Ogre3D para Android

Aunque existe una guía para la compilación de Ogre3D en Android [10], está algo desactualizada y omite varios pasos importantes. En este apartado se remarcan las correcciones que son necesarias para que la compilación de Ogre3D funcione correctamente.

Platform Version	API Level	VERSION_CODE	Notes
Android 7.0	24	N	Platform Highlights
Android 6.0	23	M	Platform Highlights
Android 5.1	22	LOLLIPOP_MR1	Platform Highlights
Android 5.0	21	LOLLIPOP	
Android 4.4W	20	KITKAT_WATCH	KitKat for Wearables Only
Android 4.4	19	KITKAT	Platform Highlights
Android 4.3	18	JELLY_BEAN_MR2	Platform Highlights
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1	Platform Highlights
Android 4.1, 4.1.1	16	JELLY_BEAN	Platform Highlights
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1	Platform Highlights
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH	
Android 3.2	13	HONEYCOMB_MR2	
Android 3.1.x	12	HONEYCOMB_MR1	Platform Highlights
Android 3.0.x	11	HONEYCOMB	Platform Highlights
Android 2.3.4	10	GINGERBREAD_MR1	Platform Highlights
Android 2.3.3			
Android 2.3.2	9	GINGERBREAD	
Android 2.3.1			
Android 2.3			
Android 2.2.x	8	FROYO	Platform Highlights
Android 2.1.x	7	ECLAIR_MR1	Platform Highlights
Android 2.0.1	6	ECLAIR_0_1	
Android 2.0	5	ECLAIR	
Android 1.6	4	DONUT	Platform Highlights
Android 1.5	3	CUPCAKE	Platform Highlights
Android 1.1	2	BASE_1_1	
Android 1.0	1	BASE	

Figura 5.3: Relación entre la versión de Android y el nivel de API [17].

Descarga del código fuente

Es necesario descargar la versión 1.9 del código fuente de Ogre3D. Sin embargo, no es conveniente descargar la etiqueta 1.9 del repositorio de Ogre3D (<https://bitbucket.org/sinbad/ogre/>) porque tras el etiquetado de la misma se subieron archivos que son necesarios para la posterior compilación del código fuente. En la figura 5.4 se puede observar que la fecha de etiquetado de la versión estable 1.9 es el 0-12-2013.

Concretamente, la versión etiquetada no compila porque le faltan dos archivos. Debe bajarse la versión de la rama, que tiene cambios mucho más actuales, y que si compila correctamente.

5.2.3 Descarga y compilación de las dependencias de OGRE3D

Las instrucciones que se indican en [10] son correctas. Las dependencias de Ogre3D compilan correctamente sin necesidad de ningún ajuste.

Compilando Ogre3D

Ogre3D para Android se compila con la herramienta CMake. CMake es una herramienta multiplataforma que permite generar o automatizar código.

Como se remarca en [10], es necesario cambiar los nombres de las toolchain por los apro-

Author	Commit	Message	Date	Builds
David Rogers	6d9849	Update RTSS compile definition in a couple scripts	2013-12-01	
David Rogers	f33621	Correct the asset mode feature strings.	2013-12-01	
David Rogers	c8745e	Reduce a lot of warnings generated with -Wundef	2013-12-01	
David Rogers	883361	Add asset mode as a config option.	2013-12-01	
David Rogers	3c9e8f	Local merge	2013-11-30	
David Rogers	faf66c	New script compilers are the default and the old ones were removed. We can get rid of these old remnants now.	2013-11-30	
David Rogers	185581	[GLES2] Bloom compositor shader fix.	2013-11-30	
David Rogers	8c0c24	Merged in screen/ogre-v1-9 (pull request #237) Fix showtopper crash in 1.9	2013-11-30	
Janik Heller	746956	Fix an #ifdef for ARM CPUs triggering under certain conditions on non-ARM CPUs due to OgrePlatformInformation.h not being included.	2013-11-30	
David Rogers	322653	Missed the crowd refs in the Cg shader	2013-11-29	
David Rogers	199351	Remove old and unused instancing sample resources	2013-11-29	
David Rogers	d8faf6c	Update the sample RTSS define in a couple places that I missed it.	2013-11-28	
David Rogers	72a83b	pull request #236, Fix Freetype detection	2013-11-28	
David Rogers	76c68a	Fix an EndlessWorld sample crash on OS X. The first entry in the world map becomes not quite null and therefore not equal to the end leading ...	2013-11-23	
David Rogers	41d8f5	[OS X] Fix a leak of GLSupport objects.	2013-11-23	
David Rogers	5c8974	Added tag v2-9-0 for changeset 4578d45d5f5b0	2013-11-21	
David Rogers	4578d45	Mark 1.9 as stable.	2013-11-21	
David Rogers	8d14ee	Merged in quatern/juan.ogre.fork/v1.9 (pull request #235) Remove #ifdef conditional from class definition.	2013-11-20	
Juan Borda	327472	Remove #ifdef conditional from class definition. This caused class to be one size when seen and instantiated from a cpp and another when mo...	2013-11-20	
David Rogers	2efaf79	Fix build error in the GL render system when threading is enabled.	2013-11-20	
David Rogers	ae55cb	Fix some warnings that spam when building with Visual Studio	2013-11-20	
David Rogers	738ac26	The uneven spacing in this file has been bugging me for ages.	2013-11-20	
David Rogers	4d5677e	[OS X] Fix the parentheses for OS X architectures. These are not CMake variables, they are meant to be interpreted by xcode, which is w...	2013-11-20	

Figura 5.4: Etiqueta 1.9 del código fuente de Ogre3D.

piados, ya que la instalar una versión más nueva del NDK de Android, el nombre es diferente. En el caso del prototipo, que utiliza la versión 10e del NDK de Android, el nombre de la tool-chain es arm-linux-androideabi-4.9.

Una vez corregido esto, es posible que sea necesario corregir varios problemas

Problema con la detección del Endianess Es posible que dé un error al intentar averiguar si estamos trabajando con un sistema operativo Little Endian o Big Endian. La solución para este problema es en primera instancia averiguar si el sistema es Big Endian o Little Endian (en Linux, ver listado 5.1). Una vez determinado, sustituir las líneas que aparecen en el listado 5.2 por 5.3.

```
lscpu | grep Endian
```

Listado 5.1: Comando para averiguar si el sistema es Big Endian o Little Endian en Linux.

```
# determine system endianess if (MSVC) # This doesn't work on VS
2010 # MSVC
only builds for intel anyway set(OGRE_TEST_BIG_ENDIAN FALSE) else()
include(TestBigEndian) test_big_endian(OGRE_TEST_BIG_ENDIAN) endif()
}
```

Listado 5.2: Líneas a sustituir en CMakeList.txt

Errores al buscar el directorio de las librerías de C++ Es necesario, si no se opera con la versión 8b del NDK de Android, cambiar las rutas que apuntan a los directorios de include y libs del NDK. Dicha información se encuentra en el archivo CMake/toolchain/

```
set(OGRE_TEST_BIG_ENDIAN FALSE)
```

Listado 5.3: Código para establecer Little Endian en CMakeList.txt. Cambiar FALSE por TRUE si el sistema es Big Endian

android.toolchain.cmake. En la versión de Ogre del prototipo, son las líneas 693 y 694 (ver 5.4).

```
set( __stlIncludePath
"\${ANDROID_NDK}/sources/cxx-stl/gnu-libstdc++/4.9/include" ) set(
__stlLibPath
"\${ANDROID_NDK}/sources/cxx-stl/gnu-libstdc++/4.9/libs/\${
  ANDROID_NDK_ABI_NAME}"
)
```

Listado 5.4: Corregir los paths de estas líneas. Solamente será necesario cambiar el número de versión (en este ejemplo 4.9)

Errores FreeType y PKGConfig Es necesario modificar el comando de ejecución de CMake que viene en [10], ya que faltan por añadir ciertos flags de compilación que son necesarios para que la ejecución funcione correctamente (ver listado 5.5).

```
cmake -DCMAKE_TOOLCHAIN_FILE="../../CMake/toolchain/android.toolchain.
  cmake"
-DOGRE_DEPENDENCIES_DIR="/home/flush/PFC/ogredeps/build"
-DANDROID_ABI=armeabi-v7a -DANDROID_NATIVE_API_LEVEL=17
-DANDROID_TOOLCHAIN_NAME=arm-linux-androideabi-4.9 ..
-DFREETYPE_FT2BUILD_INCLUDE_DIR=/home/flush/PFC/ogredeps/build
-DPKG_CONFIG_EXECUTABLE=/usr/bin/pkg-config -
  DCMAKE_C_COMPILER_ENV_VAR=CC
-DCMAKE_CXX_COMPILER_ENV_VAR=CXX
```

Listado 5.5: Comando para compilar Ogre para Android. Los 4 últimos flags no figuran en [10]

Errores en la compilación de SampleBrowser SampleBrowser es un explorador de ejemplos que viene incluido en el código fuente de Ogre3D. Presenta una sencilla interfaz donde se pueden probar diferentes técnicas y funcionales de Ogre3D mediante ejemplos simples. Como SampleBrowser viene preparado para funcionar en Android, es muy útil compilarlo para ver como trabaja Ogre3D con Android. Sin embargo, tampoco esto exento de errores:

- En el fichero Android.mk, en el cual, entre otras cosas se especifican las librerías que se enlazan con SampleBrowser, el nombre de la librería zziplib está mal escrito. Donde dice -lzzip debería poner -lzziplib.

- En el fichero `Application.mk`, es necesario cambiar la manera en la que se enlazan las librerías `Standar Template Library (STL)` (el NDK de Android incluye una versión propia y reducida de las mismas) de estática a dinámica. Para ello hay que cambiar la línea `APP_STL=static` por `APP_STL=shared`.
- las instrucciones `include` de la librería `OIS` no están correctamente especificadas. La estructura de directorios no concuerda correctamente con la estructura de `OIS` que se encuentra en las dependencias de `Ogre3D`.

5.3 Arquitectura del proyecto

El proyecto se ha estructurado en cuatro módulos bien diferenciados:

Gestión de enemigos Encargado de la generación de enemigos, de moverlos por el mapa y de gestionar sus puntos de vida.

Gestión del mapa Se encarga del pathfinding y del dibujado del mapa.

Gestión de las torres Es el encargado de construir las torres, así como del sistema de seguimiento de las torres.

Sistemas transversales o de apoyo Diversos sistemas que no tienen una agrupación funcional, como el sistema de sonido.

5.3.1 Gestión de Enemigos

La gestión de enemigos engloba varios aspectos:

- Definición de tipos de enemigos y sus atributos asociados.
- Movimiento de los enemigos.
- Gestión de la aparición y desaparición de enemigos en el mapa (sea porque han llegado al final del mismo, o porque han sido destruidos)
- Gestión de las oleadas de enemigos.

En la figura 5.5 se puede observar de manera global las clases que intervienen en la gestión de enemigos:

A continuación se detallan las cuatro clases que intervienen, una por una:

Enemy

Es la clase que representa el estado de un enemigo. Almacena toda la información del estado del enemigo, como su velocidad, sus puntos de vida... etc. También contiene información sobre la ruta que debe seguir para cruzar el mapa, en forma de un vector de waypoints. Además, almacena todos los componentes necesarios para la visualización del enemigo, como la malla, los sistemas de partículas o el `SceneNode` de `Ogre3D`. Así mismo, en su destructor se

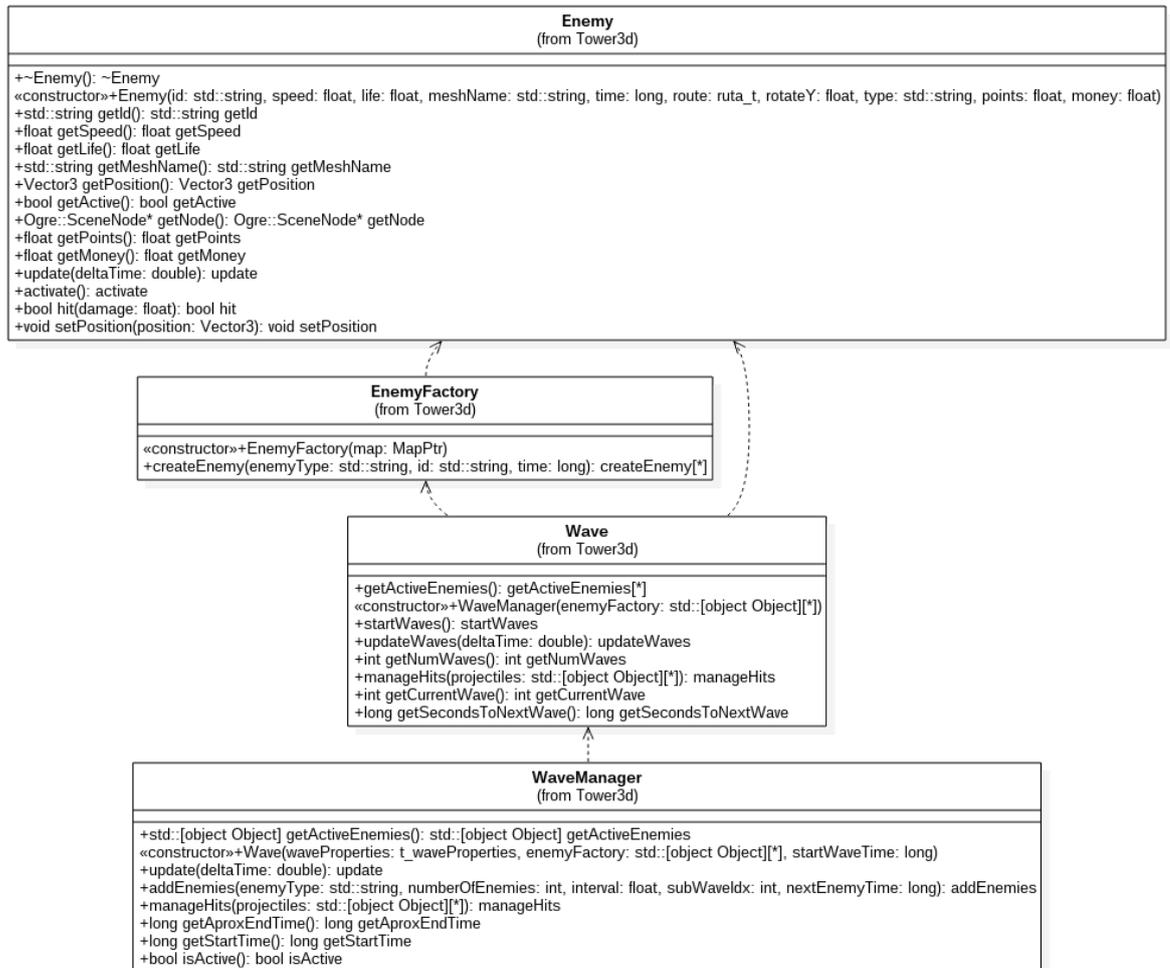


Figura 5.5: Diagrama de Clases del modulo de gestión de enemigos.

encarga de destruir todos los componentes visuales asociados. Los métodos mas importantes de la clase Enemy son:

update Este método es ejecutado en todas las iteraciones del bucle de juego para actualizar la posición del enemigo si está activado o, en caso contrario, comprobar si su tiempo de activación se ha cumplido. Si el modelo 3D correspondiente al enemigo tiene definida una animación denominada `Run`, esta animación es actualizada también. Este método devuelve `true`, si el enemigo ha llegado al final del mapa y `false` en caso contrario.

hit Este método se ejecuta cada vez que un enemigo es impactado por un proyectil. El único parámetro de este método es el número de puntos de daño que realiza el proyectil. Dichos puntos de daño son restados de la vida que le queda al enemigo. Si el resultado es igual o menor que 0, este método devuelve `true`, para indicar que el enemigo ha sido destruido. En caso contrario devuelve `false`.

EnemyFactory

Siguiendo el patrón Factory, esta clase se encarga de la creación de objetos de tipo Enemy. Para ello, hace eso un fichero de configuración denominado `enemys.cfg` en el cual se guarda la información relativa a cada tipo de enemigo. De esta manera se facilita enormemente la tarea de definir nuevos enemigos, ya que no es necesario recompilar el proyecto.

```
[Monocycle]
speed=2 life=25 rotateY=-90 meshName=Monocycle.mesh points=10; money
    =5;

[Robot] speed=1 life=200 rotateY=-90 meshName=robot.mesh points=30;
    money=15;
```

Listado 5.6: Ejemplo del contenido de `enemys.cfg`

En el listado 5.6 se muestra un ejemplo del archivo `enemys.cfg`, que se interpreta de la siguiente manera:

Identificador Entre corchetes aparece el identificador del enemigo. En el ejemplo 5.6 se definen dos enemigos: `Monocycle` y `Robot`. Todos los atributos que se encuentran entre el identificador de un enemigo y el siguiente identificador de enemigo (o el final del archivo) aplican al primero.

speed Con el atributo `speed` se definen cuantas unidades de OGRE3D es capaz de recorrer el enemigo en un segundo de tiempo. La velocidad del enemigo es uniforme durante toda la vida del mismo.

life Este atributo sirve para indicar cuantos puntos de vida debe sufrir el enemigo para ser destruido.

rotateY En ocasiones, sobre si se utilizan modelos de terceros, es posible que sea necesario realizar alguna transformación. Así, es posible aplicar al enemigo una rotación inicial sobre la vertical (Eje Y en OGRE3D) para asegurarse de que el enemigo está orientado correctamente.

meshName Es el nombre del archivo `.mesh` en el que se encuentra el modelo del enemigo. El formato `.mesh` es el formato que OGRE3D soporta para sus modelos.

points El número de puntos que recibe como recompensa el jugador cada vez que una de sus torres destruye un enemigo de este tipo.

money La cantidad de dinero que gana un jugador cuando un enemigo de este tipo es destruido por una de sus torres.

Wave

Representa una oleada de enemigos. Una oleada es una serie de enemigos, que pueden ser del mismo tipo o no, que aparecen en el mapa de forma más o menos continuada. Así, en

el objeto Wave se definen cuantos enemigos aparecen de cada tipo, cada cuanto aparecen, etc... Además, cada objeto Wave lleva asignado un tiempo de inicio de la oleada (que marca cuando deben aparecer los enemigos en el mapa) y un tiempo de fin de la misma, que el sistema utiliza para calcular el inicio de la siguiente oleada. Cada objeto Wave es responsable de actualizar la posición de sus enemigos, así como de comprobar que enemigos son impactados por los proyectiles lanzados por las torres.

Los métodos más importantes de la clase Wave son:

updateWaves Este método se ejecuta en todas las iteraciones del bucle de juego. Se encarga de ejecutar el método `update` de todos los enemigos que componen la oleada. Esta ejecución actualiza la posición del enemigo, si está activo, o lo activa si no lo está y ha alcanzado su tiempo de activación. Si alguna de las ejecuciones del método `update` devuelve `true`, significa que el enemigo ha llegado al final del mapa, por lo que el enemigo es eliminado. El método `updateWaves` devuelve un valor igual al número de enemigos que han llegado al final del mapa. Dicho valor será utilizado para restar dicha cantidad a las vidas restantes del jugador.

manageHits Este método recibe como parámetro una lista de los proyectiles disparados por las torres que han llegado a su punto de impacto. Después comprueba si algún enemigo está dentro del rango de splash del proyectil. Si es así, se ejecuta el método `hit` del objeto Enemy correspondiente, pasándole como parámetro el número de puntos de daño del proyectil. Si el método devuelve `true`, significa que el enemigo ha sido destruido con dicho impacto. En ese caso, se obtienen las recompensas en puntos y dinero que correspondan al tipo de enemigo y se acumulan. Una vez finalizadas todas las comprobaciones, este método devolverá las recompensas en puntos y dinero acumuladas por el jugador por los enemigos destruidos.

WaveManager

WaveManager actúa como factoría para la creación de objetos de tipo Wave, de forma similar a como funciona EnemyFactory. Además, en este caso WaveManager gestiona además todos los objetos Wave creados, y se encarga de actualizarlos y de activarlos cuando alcanzan su tiempo de inicio de oleada, siguiendo pues el patrón Mediator. También invoca a cada objeto Wave activo para que compruebe si alguno de los enemigos que lo forman sufre daño por alguno de los proyectiles activos lanzados por las torres. Como en el caso de EnemyFactory, se utiliza también un fichero de configuración que define las características de cada oleada (Como número de enemigos, tipo, intervalos de aparición, etc...).

Como se puede observar en 5.7 cada oleada se define con los siguientes atributos:

Identificador Entre corchetes se encuentra siempre el identificador `wave` seguido de un número (o de sin número, si es la primera oleada). No es necesario que las oleadas estén

```
[wave]
enemyType1=Monocycle enemyNumber1=1 enemyInterval1=1

enemyType2=Robot enemyNumber2=1 enemyInterval2=3 enemyDelay2=-10

[wave1] enemyType1=Monocycle enemyNumber1=20 enemyInterval1=1

enemyType2=Robot enemyNumber2=5 enemyInterval2=3 enemyDelay2=-10

[wave2] enemyType1=Monocycle enemyNumber1=20 enemyInterval1=1

enemyType2=Robot enemyNumber2=5 enemyInterval2=3 enemyDelay2=-5
```

Listado 5.7: Ejemplo del contenido de waves.cfg

definidas por orden. Una oleada se compone de varios grupos de enemigos, que están definidos por los atributos que se encuentran a continuación del identificador.

enemyType Este atributo va siempre seguido de un número, que indica a que grupo de enemigos de la oleada pertenece. El valor de este atributo indica el tipo de enemigo del que serán todos los enemigos de este grupo. El tipo debe estar definido en enemys.cfg.

enemyNumber Número de enemigos que conformar este grupo. Debe ir seguido de un número para indicar a que grupo de enemigos pertenece. Debe ser mayor que 0.

enemyInterval Número de segundos que pasan entre la aparición de un enemigo del grupo y el siguiente. Si se aumenta los enemigos aparecerán mas espaciados, y si se reduce, más juntos.

enemyDelay Es el único atributo opcional. Permite adelantar o retrasar la aparición de un grupo de enemigos. De esta manera es posible conseguir que varios grupos aparezcan de forma simultanea.

5.3.2 Gestión de las torres

La gestión de las torres engloba entre otros los siguientes aspectos:

- Creación de torres y definición de tipos de torre.
- Mejora de las torres
- Sistema de seguimiento de enemigos para las torres.
- Gestión de proyectiles lanzados
- Control de validez del lugar de construcción de las torres.

En la figura 5.6 se puede observar de manera generalizada las clases que intervienen en la gestión de las torres.

El módulo se compone de las siguientes tres clases:

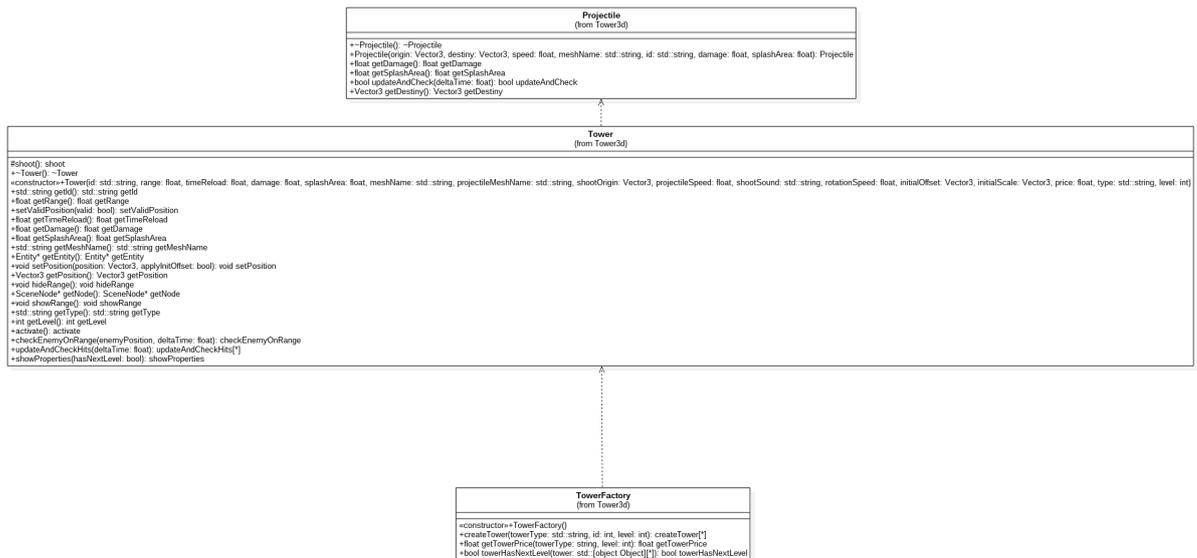


Figura 5.6: Diagrama de Clases del módulo de gestión de torres.

Projectile

Esta clase representa un proyectil disparado por una torre. Tiene atributos como el modelo3D que representa el proyectil, el daño que inflige, la velocidad a la que viaja, etc... El método más importante de la clase Projectile es:

updateAndCheck Este método actualiza la posición del proyectil y comprueba si ha llegado a su destino. Es importante resaltar que el destino de un proyectil es fijado en el momento en el que se dispara y no varía después. Esto quiere decir que un enemigo lo suficientemente rápido podría no ser dañado por un proyectil que se ha disparado contra él, porque podría estar lejos de la zona de impacto para cuando el proyectil llegara a su destino. Si el método devuelve true, es que el proyectil ha llegado a su punto de impacto.

Tower

Esta clase representa una torre que ha sido o está siendo construida por el jugador. Lo métodos más importantes de la clase Tower son:

activate Este método es el invocado cuando el jugador posiciona una torre en el mapa. Dicha torre pasa a estar anclada en el terreno (ya no se moverá siguiendo el cursos del ratón) y empezará a disparar a los enemigos que se acerquen.

CheckEnemyOnRange Este método recibe como parámetro un enemigo y se ejecuta una vez en cada iteración del bucle de juego por cada enemigo que este activo en el mapa. Se compone de tres pasos:

- Si la torre no tiene un enemigo asignado o bien el enemigo que tenía asignado se

ha salido del rango de acción de la torre, se comprueba si el enemigo pasado por parámetros se encuentra dentro del rango de acción de la torre. Si es el caso, pasa a ser el nuevo enemigo objetivo de la torre.

- Se comprueba si la torre esta alineada con el enemigo objetivo. Si no es el caso, se gira la torre hacia el enemigo, realizando una rotación sobre la vertical de tantos grados como le permite su velocidad de giro.
- Por último se comprueba si la cadencia de tiro permite disparar al enemigo objetivo de la torre (si es que lo hay). También se comprueba que la torre esté correctamente orientada hacia el enemigo, permitiéndose una tolerancia en la orientación de dos grados. Si se cumplen ambas condiciones, la torre lanza un proyectil al sitio donde se encuentra el enemigo este momento.

updateAndCheckHits Este método comprueba si los proyectiles lanzados por la torres han llegado a su punto de impacto, ejecutando el método `updateAndCheck` de la clase `Projectile` para cada proyectil lanzado por la torre. Este método devuelve una lista con los proyectiles que han llegado a su punto de impacto, que su vez son eliminados de la lista que mantiene la torre de proyectiles lanzados, por lo que dejan de estar bajo su control.

showProperties Este método muestra un panel con las características de la torre, en el cual, si la torres es mejorable, aparece un botón de mejorar torre.

TowerFactory

Otro clase que sigue el patrón `Factory`. De manera similar al `EnemyFactory` y al `WaveManager`, esta clase también lee toda la información referente a los tipos de torres disponibles para el jugador de un archivo de configuración, denominado `towers.cfg`. Dicho archivo permite especificar diferentes tipos de torres, incluso definir nuevos sin necesidad de recompilar el proyecto. En este archivo se definen cosas como el precio de la torre, el alcance, la malla 3D y las características de sus proyectiles. Además, si una torre puede ser mejorada, se definen las diferentes mejoras de la torre. Para ello se definen torres con el mismo nombre, seguidas de un número que indica que grado de mejora es. Si no lleva número, es la torre básica. Si lleva un 1, es la primera mejora, si lleva un 2, la segunda, etc... Un ejemplo de archivo de configuración.

Mas en detalle, un ejemplo del archivo `towers.cfg` puede verse en el listado 5.8.

La forma de definir torres es:

Identificador Como en los archivos anteriores, entre corchetes se situa el identificador de la torre. Si el identificador no tiene un número al final, se trata de una torre de nivel 0. En cambio si lo tiene, indica el nivel de la torre, siendo pues una mejora de la misma. Así, según el ejemplo de 5.8, se han definido dos tipos de torres: `CanonTower`

```

[CanonTower]
range=3 damage=10 timeReload=2 splashArea=2 shootPosition=0 1 0
projectileMeshName=cannonBall.mesh meshName=torreCanon.mesh
  projectileSpeed=20
shootSound=canonShoot.wav rotationSpeed=90 initialOffset=0 0.3 0
  initialScale=
0.5 0.5 0.5 price=50

[CanonTower1] range=3 damage=20 timeReload=2 splashArea=2
  shootPosition=0 1 0
projectileMeshName=cannonBall.mesh meshName=torreCanon.mesh
  projectileSpeed=20
shootSound=canonShoot.wav rotationSpeed=90 initialOffset=0 0.3 0
  initialScale=
0.5 0.5 0.5 price=50

[machineGunTower] range=4 damage=2 timeReload=1 splashArea=1
  shootPosition=0 1
0 projectileMeshName=bullet.mesh meshName=machineGunTower.mesh
projectileSpeed=50 shootSound=bullet.wav rotationSpeed=90
  initialOffset=0 0.1
0 initialScale= 0.3 0.3 0.3 price=25

[machineGunTower1] range=4 damage=4 timeReload=0.8 splashArea=1
shootPosition=0 1 0 projectileMeshName=bullet.mesh
meshName=machineGunTower.mesh projectileSpeed=50 shootSound=bullet.
wav
rotationSpeed=90 initialOffset=0 0.1 0 initialScale= 0.3 0.3 0.3
price=25

```

Listado 5.8: Ejemplo del contenido de towers.cfg

y machineGunTower; ambas tienen una mejora disponible. No pueden existir mejoras de torre que no sean correlativas, es decir, si existe CanonTower3, deben estar definidas CanonTower, CanonTower1 y CanonTower2

range Indica el alcance de la torre, en unidades de Ogre3D. El alcance se utiliza tanto para como alcance de detección de enemigos así como para la distancia máxima a la que se puede disparar.

damage El número de puntos de daño que causa a un enemigo los proyectiles disparados por esta torre.

timeReload El tiempo en segundos que debe pasar tras el disparo de una torre, para que esta pueda volver a efectuar el disparo. Es la inversa de la cadencia de disparo.

splashArea Cuando un proyectil impacta, no sólo hace daño en el punto exacto donde impacta, si no también en un área circundante. Este atributo indica el tamaño de dicho área.

shootPosition Este atributo es un vector tridimensional, cuyas coordenadas están separadas

por espacios. Indica el punto relativo a la posición de la torre desde el que se disparará el proyectil. Esto es necesario debido a que el cañón de la torre (o el elemento desde el que debería salir el proyectil) no suele coincidir con el centro de la torre.

projectileMeshName Indica el modelo 3D del proyectil que lanza esta torre.

projectilSpeed Este atributo indica la velocidad a la que viaja el proyectil. Aunque se puede indicar cualquier valor positivo, valores menos que 10 no funcionan bien porque el proyectil es demasiado lento.

shootSound Efecto de sonido que debe reproducirse al disparar la torre. Es bastante importante que dicho efecto de disparo sea corto y no tenga ningún silencio al principio, ya que el jugador pierde la sensación de sincronismo entre el disparo y el sonido.

rotationSpeed Indica con que velocidad puede rotar la torre mientras se orienta hacia su enemigo objetivo. La velocidad es indicada en grados por segundo. Valores por debajo de 45 son demasiado lentos, y normalmente la torre no consigue orientarse nunca para disparar contra el enemigo. Tampoco funcionan bien valores por encima de 180, porque no se aprecia la rotación, parece que la torre simplemente se orienta correctamente en solo paso, sin movimiento.

initailOffset Este atributo permite indicar un vector tridimensional de desplazamiento que se aplica a la torre antes de ser construida. Es útil si el modelo de la torre no tiene su centro en la base, ya que si no se aplica el desplazamiento en estos casos, la torre se ve cortada por el plano.

initialScale Es posible indicar en este atributo un escalado inicial en la torre. Así, es posible incorporar cualquier modelo de torre de terceros, sin necesidad de tener que escalarlo en un editor 3D.

5.3.3 Gestión del mapa

Las clases que se utilizan para la gestión del mapa son las siguientes:

A continuación se detallan las clases por separado:

MapTile

Representa una porción de Mapa, que está dividido en casillas. Antes de que se tomara la decisión de utilizar una única malla estática para visualizar el mapa (ver sección de decisiones técnicas) también poseía una representación visual. Representa la posición de la casilla en el mapa.

graphmlBoost

Es una clase de utilidades que permite leer un archivo en formato GraphML [19], así como realizar búsquedas para encontrar un camino óptimo en un grafo. Este grafo es el que

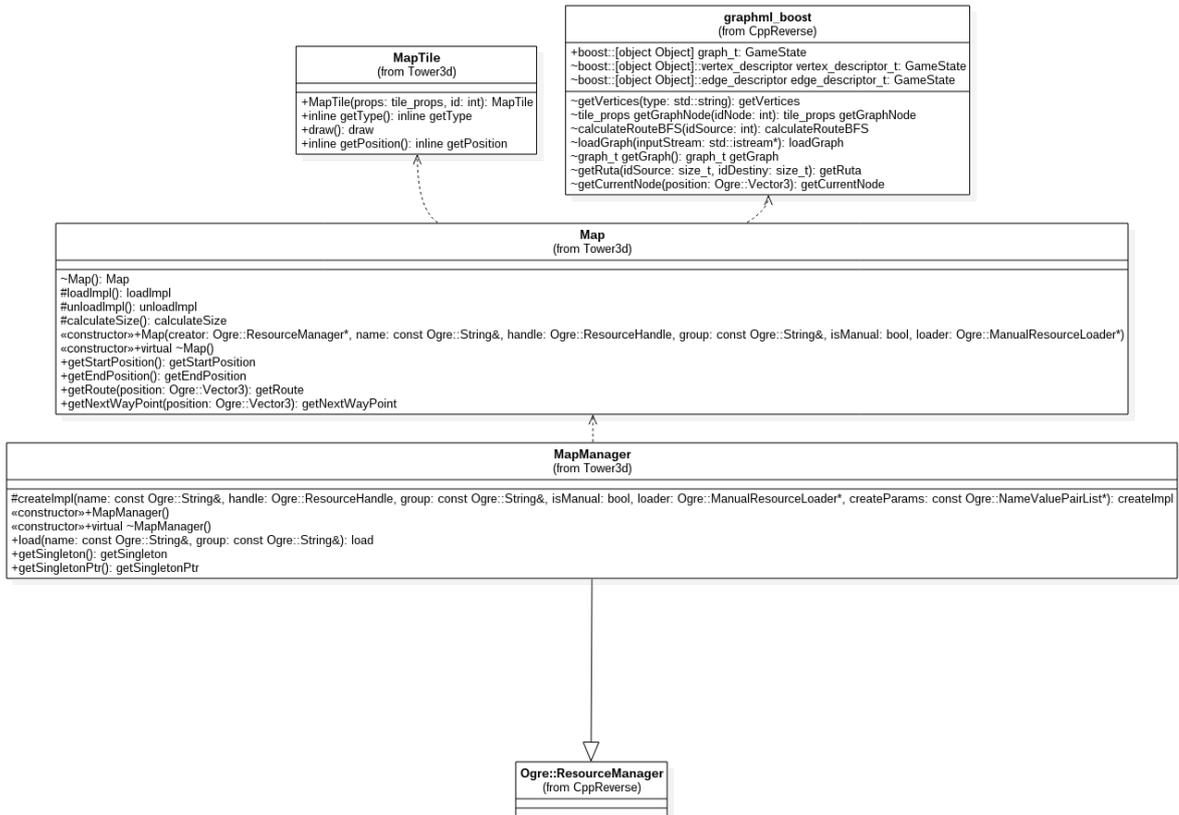


Figura 5.7: Diagrama de Clases del módulo de gestión de mapas.

se utilizará para calcular el camino que siguen los enemigos al cruzar el mapa. Hace uso de la librería Boost[5] tanto para las búsquedas como para el parseo del fichero GraphML.

Un archivo GraphML es un archivo XML con unas etiquetas concretas. En el listado 5.9 se puede ver un ejemplo.

Las partes que componen un archivo graphML son:

<graphml> Etiqueta raíz.

<key> Cada una de estas etiquetas define un atributo de los nodos y de las aristas. La etiqueta estos atributos:

id Identificador del atributo.

for Tiene dos posibles valores: **node** si el atributo que se está definiendo es aplicable a los nodos, o **edge** si es aplicable a las aristas.

attr.name Nombre del atributo

attr.type Tipo del atributo.

<graph> Esta etiqueta sirve para indicar la definición de un grafo. Un archivo graphML puede contener cualquier número de grafos. Posee dos atributos:

```

<graphml xmlns="http://wgraphml.graphdrawing.org/xmlns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
<key id="keyIdNodo" for="node" attr.name="id" attr.type="string"/>
<key id="keyX" for="node" attr.name="x" attr.type="string"/>
<key id="keyY" for="node" attr.name="y" attr.type="string"/>
<key id="keyZ" for="node" attr.name="z" attr.type="string"/>
<key id="keyType" for="node" attr.name="type" attr.type="string"/>
<key id="keySource" for="edge" attr.name="source" attr.type="int"/>
<key id="keyTarget" for="edge" attr.name="target" attr.type="int"/>
<key id="keyWeight" for="edge" attr.name="weight" attr.type="int"/>
<graph id="level.001" edgedefault="undirected" >
  <node id="0" >
    <data key="keyIdNodo">0</data>
    <data key="keyX">10.037900924682617</data>
    <data key="keyY">0.0</data>
    <data key="keyZ">-7.0</data>
    <data key="keyType">terrain</data>
  </node>
  <node id="1" >
    ...
  <edge id="e1386" source="0" target="1" >
    <data key="keySource">0</data>
    <data key="keyTarget">1</data>
  </edge>
  <edge id="e1390" source="694" target="722" >
    <data key="keySource">694</data>
    ...

```

Listado 5.9: Un archivo graphML

id Identificador del grafo.

edgedefault Indica si las aristas son por defecto dirigidas (*directed*) o no dirigidas (*undirected*)

<node> Define un nodo del grafo. Su único atributo es un identificador. Dicho identificador podrá ser referenciado posteriormente cuando se define las aristas.

<edge> Define una arista del grafo. Posee varios atributos, de los cuales en el prototipo sólo se utilizan tres:

id Identificador de la arista.

source Identificador del nodo origen de la arista.

target Identificador del nodo destino de la arista.

<data> Puede encontrarse dentro de *node* o de *edge*. Se utiliza para dar valor a los atributos definidos por las etiquetas *key*.

La librería Boost tiene una función capaz de parsear estos archivos graphML y convertirlos en un grafo en memoria, sobre el que se pueden aplicar búsquedas, tanto en profundidad como en anchura (amén de otras búsquedas con Heurística, Dijkstra, etc..).

Sin embargo el manejo que hace Boost de los grafos es muy poco intuitivo. Por defecto, si se utiliza cualquiera de los métodos de búsqueda que se ofrecen en la librería, no se obtendrá ninguna respuesta. Se recorrerá el grafo internamente, pero no se registrarán los caminos óptimos.

Para poder registrar los caminos óptimos es necesario establecer `visitors`. Los `visitors` son objetos que contiene varios métodos que son ejecutados en diferentes momentos mientras se recorre el grafo en busca de los caminos óptimos. En 5.10 se muestra un ejemplo del código necesario para ejecutar una búsqueda por anchura con Boost. Se puede observar que en ningún momento de la búsqueda se especifica el nodo al que se quiere llegar, solamente el nodo origen. Esto es debido que no existen algoritmos que obtengan el camino más corto entre un nodo origen y un nodo destino que sean más rápidos que los algoritmos que obtienen todos los caminos más cortos a todos los nodos destino de un grafo a partir de un único nodo origen. Así, siempre que se ejecuta una búsqueda en anchura, se calculan todos los caminos más cortos des un único nodo origen [6].

```
_p = std::vector<vertex_descriptor_t>(boost::num_vertices(_graph));
_p[idSource] = idSource;
boost::breadth_first_search(
    _graph, idSource, // s,
    boost::visitor(boost::make_bfs_visitor(std::make_pair(
        boost::record_distances(d, boost::on_tree_edge()),
        boost::record_predecessors(&_p[0], boost::on_tree_edge()))
    ));
```

Listado 5.10: Código para lanzar una búsqueda por anchura.

La clase `graphml_boost` sigue el patrón Facade, ya que es utilizada como la única puerta de acceso a la información del grafo, así como a las funcionalidades de `pathfinding`.

Los métodos más importantes de la clase son:

Este método obtiene la ruta más corta entre dos nodos del grafo. Puesto que la ruta para un mapa es siempre la misma (el nodo origen y el destino no varían), solamente se calcula una vez y después se almacena en un atributo de la clase. Para devolver la ruta, se utiliza una estructura denominada `ruta_t`, que es una lista de nodos, con sus respectivas coordenadas en el espacio tridimensional.

Este método permite obtener todos los vértices de un tipo determinado. Los vértices pueden ser de cuatro tipos:

- road** Es un vértice de tipo camino. En estos vértices no pueden construirse torres. Los enemigos siempre circulan por este tipo de vértices,
- terrain** Es un vértice de terreno. En estos vértices se permiten la construcción de torres. Por este tipo de vértices no pueden circular los enemigos.
- start** Este vértice (es único por grafo) indica el punto desde el que aparecen los enemigos en el mapa.

end Este vértice (es único por grafo) indica el punto al que los enemigos intentan llegar. Si un enemigo alcanza este vértice, desaparece del mapa y hace que el jugador pierda una vida.

Map

Si cuando se describía la clase `graphml_boost` se comentaba que seguía el patrón Facade, para aislar toda la lógica de Boost sobre grafos, y ofrecer un único punto de entrada, evitando acoplamiento entre diferentes módulos, se puede decir algo parecido de la clase `Map`. La clase `Map` también sigue el patrón Facade, en este caso, sirviendo como puerta de acceso al resto de módulos para acceder a la funcionalidad de los mapas.

En este caso además, `Map` hereda de la clase `Ogre::Resource`. Este herencia permite que se pueden referenciar a los mapas como recursos genéricos dentro de `Ogre`, así como utilizar la carga automática de recursos que realiza `Ogre3D`.

Los métodos mas importantes de la clase `Map` son:

loadImpl Este método es ejecutado por la clase `MapManager` cuando se solicita la carga del recurso. En este método se produce toda la inicialización de la clase.

unloadImpl Este método se ejecuta cuando ya no se necesita el Recurso. Hace las veces de destructor.

getStartPosition Devuelve el vértice del grafo en el que los enemigos empiezan a aparecer en el mapa. También incluye las coordenadas tridimensionales del vértice.

getEndPosition Devuelve el vértice del grafo al que los enemigos deben dirigirse. También incluye las coordenadas tridimensionales del vértice.

getRoute Devuelve la ruta que debe seguir un enemigo para atravesar el mapa siguiente el camino. La ruta es un array de vértices ordenados de forma inversa que indican el camino hasta el destino.

containsRoad Este método recibe como parámetro una `Axis Align Bounding Box`. Esto es, una caja hueca que está alineada con los ejes del mundo de `Ogre3D`. Este método comprueba si algún vértice de tipo `road` se encuentra dentro de dicha caja. Se utiliza para comprobar si una torre puede construirse en un lugar en concreto.

También es responsabilidad de la clase `Map` dibujar la geometría estática que conforma el modelo 3D del mapa.

MapManager

Esta clase hereda de la clase `ResourceManager` de `Ogre3D`. De esta manera, se beneficia del parseo automático que realiza `Ogre3D` cuando inicializa sus recursos. Al vincular dicha clase a la extensión «.mapdef», cada vez que `Ogre3D` encuentra un archivo con esta extensión, invoca a `MapManager` para que se haga cargo de el. `MapManager` es, por herencia

de `Ogre::ResourceManager`, una clase que sigue el patrón Singleton de Ogre3D. Aunque en otro apartado más adelante se hablará con más detalle sobre el uso de clases Singleton en Ogre3D, es importante remarcar que el patrón Singleton de Ogre3D, no es un patrón Singleton al uso, ya que requiere una inicialización previa. Así, es necesario crear la instancia de la clase `MapManager` antes de poder utilizarla. Y debe hacerse antes de que Ogre3D realice la carga de recursos. Los métodos más importantes de la clase `MapManager` son:

MapManager El constructor es uno de los métodos más importantes de un gestor de recursos de Ogre. tiene tres responsabilidades básicas:

- Definir el patrón (o patrones) por el que se identificarán los recursos que gestiona esta clase. Para ello, debe introducirse en la variable heredada `mScriptPatterns`.
- Registrarse a sí mismo como `scriptLoader`, lo que hará que sea llamado por el gestor de recursos de Ogre3D cuando se encuentre un archivo que cumpla el patrón.
- Registrarse a sí mismo como `resourceManager`, de tal manera que sea llamado cuando la aplicación quiere obtener recursos del tipo que gestiona esta clase.

5.3.4 Sistemas Transversales o de Apoyo

Sistema de Sonido

Para la implementación del sistema de sonido de la aplicación, se ha utilizado la librería SDL [23]. Para evitar en la medida de lo posible el acoplamiento del resto de la aplicación con SDL, se han creado dos clases: `SoundManager` y `Sound`.

SoundManager Es una clase que sigue el patrón Singleton (tal y como lo usa Ogre), que funciona como clase de utilidades para reproducir efectos de sonido y música. Tiene cuatro métodos importantes:

loadSound Carga un archivo de sonido, convirtiéndolo en un objeto de la clase `Mix_Chunk`, para poder ser reproducido por SDL. Este método solamente es invocado por la clase `Sound`.

playFX Reproduce un efecto de Sonido por el primer canal libre disponible.

playMusic Reproduce una música de forma continuada por el canal número 1.

stopMusic Detiene la reproducción de música por el canal número 1.

Sound Esta clase recubre la estructura de sonido que utiliza SDL para que el juego no quede acoplado a dicha estructura. Los métodos más importantes son:

Sound Esta clase posee dos constructores. Uno sin parámetros, en el cual se crea el objeto, pero no se le asigna ningún archivo de sonido. Este tipo de inicialización requerirá de

una llamada posterior al método `setSound` para asignarle un fichero de sonido. El otro constructor admite como parámetro un string que debe especificar la ruta completa a un archivo de audio `.wav`. Este constructor llama directamente al método `setSound`, que parsea el archivo y lo convierte en una estructura reproducible por SDL.

setSound Este método admite como parámetro un string que debe especificar la ruta completa a un archivo de audio `.wav`. Este método carga el archivo en memoria en un formato reproducible por SDL, dejándolo referenciado por la variable de clase privada `_chunk`.

Control de Flujo del juego

Para controlar el flujo de la aplicación se ha utilizado una máquina de estados. En la figura 5.8 se pueden ver los estados que forman el juego.

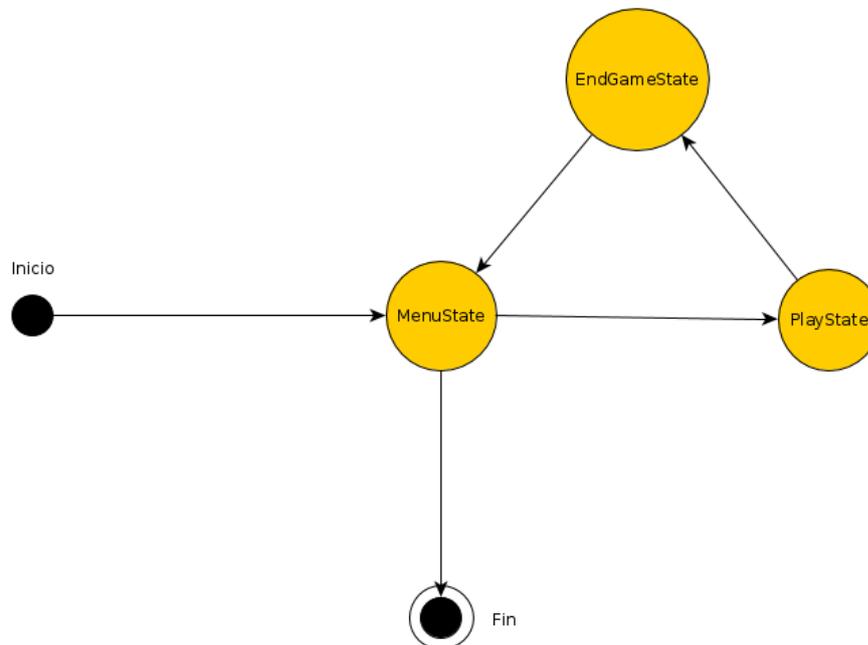


Figura 5.8: Diagrama de estados del juego

MenuState Representa el menú principal del juego, y da acceso a los diferentes estados. Es el único punto desde el que se puede salir de la aplicación. Aunque en el diagrama se muestra como primera estado, es necesario indicar que previo a este estado se muestra una pantalla de carga cuando se inicia el juego, aunque esta pantalla de carga está fuera de la máquina de estados. Desde este estado se da acceso a la pantalla de records.

PlayState Este es el estado principal del juego. Coordina todos los módulos indicados anteriormente para conseguir que el juego funcione correctamente. Tiene una lógica bastante importante y es responsable de varias funcionalidades:

- Almacena el estado global de la partida. Así, es el encargado de almacenar los puntos que ha ganado el jugador, así como las vidas restantes y el dinero del que dispone.
- Gestiona todas las acciones realizadas por el usuario. Esto incluye la detección de pulsaciones en los botones, selección de torres construidas y construcción de las mismas.
- Comprueba que las torres son construidas en localizaciones correctas.
- Muestra mensajes en pantalla cuando el jugador intenta realizar una acción para la que no dispone de dinero suficiente.
- Es el encargado de dar por finalizada la partida, sea con la victoria o la derrota del jugador.
- También actualiza la información del HUD mostrada al jugador, para que se ajuste a la realidad.

EndGameState este estado es llamado por PlayState cuando la partida ha terminado, sea con la victoria o la derrota del jugador. Es el encargado de mostrar un mensaje en el que se le informa al jugador si ha ganado o perdido, y se le permite introducir un nombre para asociarlo a su puntuación. Una vez completado esto, llama al MenuState.

5.4 Decisiones Técnicas

En este apartado se discuten decisiones técnicas que han tenido un impacto muy importante en el proyecto.

Edición de Mapas

Para el diseño de mapas, se ha decidido realizar un pequeño editor en Blender, que permita generar de forma fácil el grafo a un formato estándar como es GraphML. En la figura 5.9 puede verse el aspecto del editor.

Para crear un mapa, es necesario primero añadir en Blender un objeto Grid, de las dimensiones deseadas. Este objeto debe llamarse obligatoriamente grafo. Después, se ejecuta el script de Python que aparece en la pantalla. Al ejecutarlo, aparece a la izquierda un panel que permite marcar cada vértice como 4 tipos de terreno diferentes:

terrain Terreno limpio donde se pueden construir las torres.

road Camino por donde circulará el enemigo. Aquí no se pueden construir torres.

start Primera casilla del camino. Donde aparecen los enemigos.

end Última casilla del camino. Cada enemigo que llegue a esta casilla restará una vida al jugador.

Internamente, el editor funciona añadiendo y quitando vértices de cuatro grupos deno-

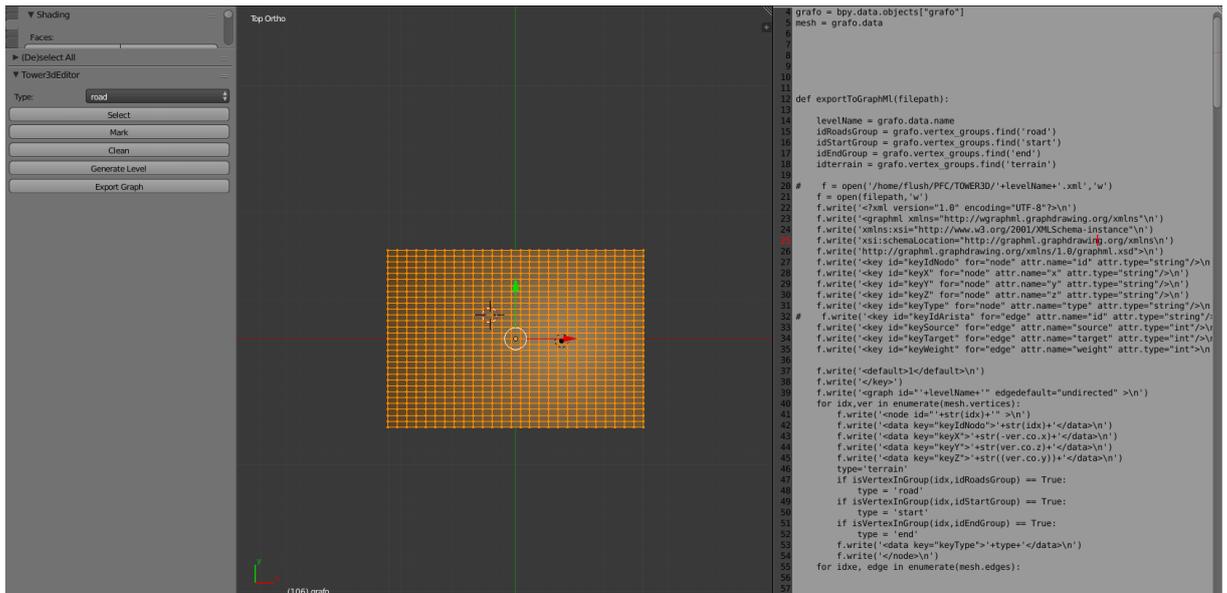


Figura 5.9: Editor de mapas del juego, elaborado en Blender mediante el lenguaje Python

minados igual que los tipos que se han mostrado anteriormente. El editor permite acciones como marcar los vértices seleccionados como de un tipo en concreto, eliminar de un grupo unos vértices seleccionados, o bien seleccionar en la pantalla los vértices de un único tipo (ver figura 5.10).

Una vez todos los vértices del grid han sido asignados a un tipo en concreto, los grupos de vértices deben cumplir las siguientes reglas:

- Solamente debe existir un vértice de tipo end y otro de tipo start.
- Todo vértice de tipo road debe estar adyacente a dos vértices de tipo road, a un vértice de tipo road y a un vértice de tipo start, o bien a un vértice de tipo road y un vértice de tipo start.
- Los vértices de tipo road no pueden formar un bucle.

Cumplidas todas las reglas anteriores, se puede pulsar sobre el botón Export Graph, que generará un archivo de extensión mapdef, que estará en formato graphML. Dicho archivo contendrá toda la información necesaria sobre el mapa para que pueda ser utilizado en el juego.

Se puede ver que existe un botón llamado Generate Level. Dicho botón sería para generar el mapa a partir de pequeños cubos texturizados, pero dicho enfoque se abandonó por tener un mal rendimiento. La forma adoptada actualmente para generar la malla del mapa, es crear una copia del objeto grafo, extruirla para crear un objeto 3D, texturizarla y exportarla en formato Ogre3D, nombrándola de la misma manera que el archivo GraphML generado anteriormente (salvo la extensión).

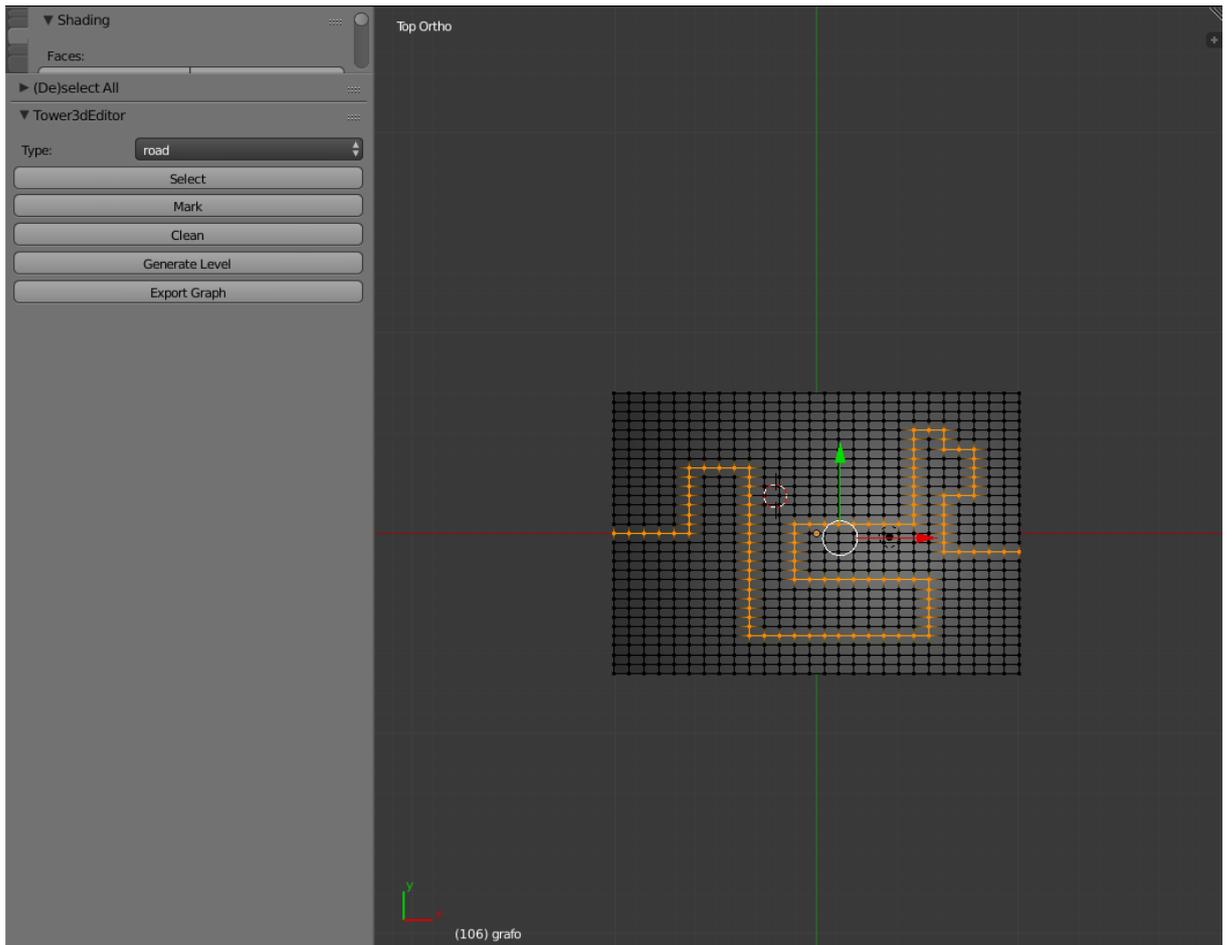


Figura 5.10: Editor de mapas del juego, con los vértices de tipo road resaltados.

Dibujado del mapa

En un principio se tomó la decisión fundamentada en una práctica anterior, de construir el mapa a partir de pequeños tiles que serían texturizados de forma individual. De esta manera se podría tener un catálogo de tiles para cada tipo, y combinarlos, generando mapas diferentes de forma sencilla.

Tras realizar la implementación de esta manera, se observa que el juego funciona de manera fluida en plataforma PC, pero no así en plataforma Android, donde el juego cae por debajo de 15 frames por segundo.

Tras una investigación se descubre las tarjetas gráficas no funcionan bien con múltiples objetos pequeños sencillos, sino que prefieren objetos grandes y complejos [9]. Una vez realizado el cambio, se experimenta una mejora importante de rendimiento, pasando en plataforma PC de 40 FPS a 210FPS y en plataforma Android, de 15 FPS a 35 FPS.

Sin embargo, debido a este cambio realizado a última hora, el texturizado del mapa ha quedado muy simple. En la figura 5.11 se puede observar el aspecto que tenía dibujado con tiles, mientras que la figura 5.12 muestra el aspecto actual.



Figura 5.11: Pantalla de juego realizada con tiles.

5.4.1 Gestión de la entrada

Aunque Ogre incluye OIS como dependencia, para la gestión de entrada del usuario, se ha descartado su uso, por la dificultad de su uso en Android. Aunque posible, OIS requiere mucho código adicional para funcionar en Android, y su soporte para multitouch no funciona correctamente, fallando de forma aleatoria. Así, se ha decidido utilizar SDL para la gestión de entrada de usuario, que no requiere ningún código adicional. Además SDL aporta grandes ventajas respecto al uso de OIS, como por ejemplo, es capaz de invocar el teclado virtual de Android solamente con indicar que se va a proceder a una entrada de texto.

5.4.2 Interfaz de usuario

Para la generación de la interfaz de usuario, se valoraron las siguientes opciones:

SDKTrays Es una simple utilidad para generar interfaces gráficas que se incluye con los ejemplos de Ogre3D. No se ha elegido esta opción por dos motivos: Obliga a utilizar OIS, cuyo uso ha sido descartado para Android (ver apartado anterior) y no admite scripts, todo se hacer por código fuente, lo que obliga a recompilar con cada cambio.

Crazy Eddie's GUI (CEGUI) Es una utilidad muy potente y compleja que permite realizar interfaces de usuario complejas. Entre sus virtudes se encuentra la posibilidad de realizar temas, que cambian fácilmente el aspecto de la interfaz sin tener que modificar el código. Se ha descartado por falta de documentación sobre su funcionamiento en Android y por su excesiva complejidad.

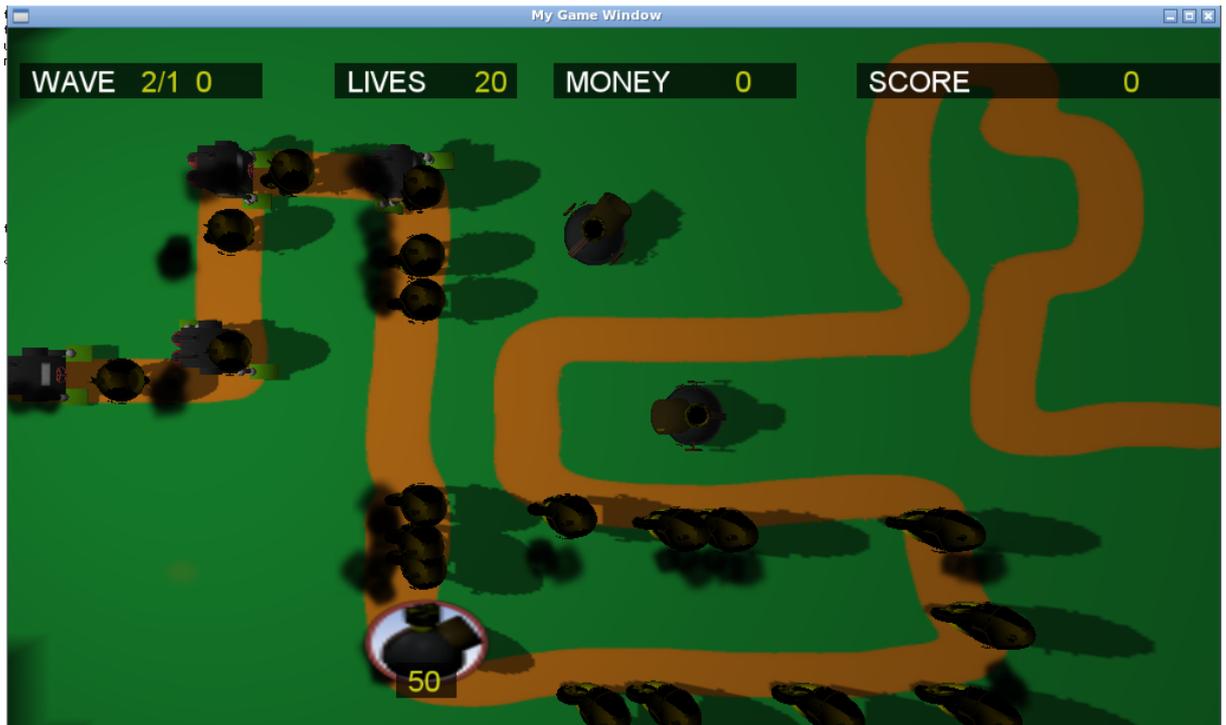


Figura 5.12: Pantalla de juego realizada con una sola malla de geometría estática.

MYGUI Es una librería más sencilla de utilizar que CEGUI, pero casi con la misma potencia. Sin embargo, requiere de la implementación de Shaders programados adicionales para sustituir algunos que no están disponibles en GLES 2. Por eso ha sido también descartada.

Overlays Es la utilidad básica para generar superficies 2D de Ogre. Permite definición por scripts y funciona de la misma manera tanto en plataforma PC como en Android. Esta opción ha sido la escogida para este desarrollo, ya que no era necesario incluir ninguna dependencia adicional, y proporciona la funcionalidad necesaria. En el listado 5.11 se puede ver un ejemplo.

5.4.3 Técnicas de Sombreado

Ogre3D soporta dos técnicas de sombreado, que producen resultados diferentes:

Stencil Buffer Las sombras producidas por esta técnica tienen las siguientes características [31]:

- Las sombras siempre tienen bordes duros, es decir no hay un degradado suave al final de la sombra.
- Un objeto puede recibir su propia sombra

Mapas de Texturas Las sombras producidas mediante mapas de texturas tienen las siguientes características:



Figura 5.13: Captura de pantalla de SampleBrowser de Ogre3D, que utiliza SDKTrays para su interfaz usuario.



Figura 5.14: Captura de pantalla de Open Dungeons en la que se puede apreciar la interfaz, implementada con CEGUI.



Figura 5.15: Captura de pantalla de Energy Tycoon, cuya interfaz está implementada con MYGUI.

- La sombras tienen bordes suaves, y pueden tener un color propio.
- Un objeto no puede ser a la vez emisor y receptor de sombras.
- Un objeto no puede recibir su propia sombra.
- Al estar basado en su mayoría en el uso de Graphic Processing Unit (GPU), descarga en gran medida la Central Processing Unit (CPU)

En la figura 5.16 se puede ver una comparativa de los dos tipos de sombreado.

Para el prototipo se ha escogido utilizar mapas de texturas, porque no se ha conseguido que la técnica Stencil Buffer funcione en Android. Aunque hubiera funcionado, probablemente los mapas de texturas son una opción mejor que Stencil Buffer, ya que son más ligeros para la CPU.

5.4.4 Ogre3D y SDL

A la hora de utilizar Ogre3D y SDL de manera conjunta en Android, existen dos alternativas diferentes: que Ogre3D crea la ventana o que SDL cree la ventana.

Ogre3D crea la ventana

El soporte para EGL de Ogre3D no está un gran estado de madurez. Aún así, funciona mucho mejor si Ogre3D es el que crea la ventana y el contexto EGL (EGL es una API que se encuentra entre OPENGL ES 2.0 y el sistema de ventanas de Android). Este enfoque tiene unas cuantas particularidades a tener en cuenta:

```

container Panel(menu/background)
{
    transparent false
    left 0.0
    top 0.0
    width 1
    height 1
    material MatBackgroundMenu

    element Panel (menu/Exit)
    {
        left 0.05
        top 0.05
        width 0.15
        height 0.20
        material MatCloseButton
    }
}

```

Listado 5.11: Script de overlays para dibujar los botones del menú.

- El control de las Human Interface Devices (HID) de SDL no funciona correctamente si no es SDL el que crea la ventana. Por lo tanto se pierde uno de los motivos por los que se utiliza SDL.
- Ogre3D en dispositivo móvil tarda entre 5-10 segundos en cargar. Durante esos segundos, la pantalla permanecerá en negro.
- Al ser Ogre3D el que crea la ventana, es el encargado también de implementar el código Java Native Interface (JNI) para la gestión de la aplicación en Android (debe implementar métodos para cuando la aplicación se minimiza, se para o se carga). Este código no existe como tal en la librería Ogre3D, si no que debe extraerse de los ejemplos que se incluyen en el empaquetado. Dicho código no está bien documentado, y se mezcla con funcionalidades que pueden no ser interesantes para el desarrollo.

Se puede observar que este enfoque tiene desventajas muy importantes, que lo hacen claramente descartable.

SDL crea la ventana

SDL tiene un soporte para Android que funciona muy bien, y es fácil y sencillo de utilizar. El empaquetado de SDL viene con una actividad de Android preparada para ejecutarse con SDL. La documentación de SDL para Android es muy completa y clara, y no presenta prácticamente ninguna laguna de importancia. Estas son las características de este enfoque:



Figura 5.16: Comparativa de modos de sombreado. Se pueden observar los bordes duros de la sombra mediante Stencil Buffer.

- Ogre3D tiene errores de implementación cuando acepta un contexto EGL creado de manera externa. Es necesario realizar ciertos ajustes al código para que funcione correctamente [8].
- Al ser SDL el que crea la ventana, es posible cargar una imagen mediante SDL antes de comenzar a cargar Ogre3D, con lo la pantalla ya no se queda en negra mientras se espera a que Ogre3D esté disponible.

Aunque este enfoque ha requerido bastante tiempo invertido en corregir errores de Ogre3D en el manejo de contextos EGL externos, tiene claras ventajas respecto al enfoque anterior, por lo que es el enfoque que se ha elegido para el prototipo.

5.4.5 Otros ajustes

En este apartado se van a indicar una serie de pequeños ajustes que han sido necesarios para que el prototipo pueda funcionar en Android.

Problemas con las rutas

Debido al empaquetado que se realiza en las aplicaciones de Android, las rutas relativas de los recursos no coinciden con las rutas que se especifican cuando el juego se ejecuta en Linux. Es necesario utilizar un objeto de tipo `Ogre3d::FileSystemLayer` para el acceso cuando se accede desde Android. En el listado 5.12 se puede observar la manera en la que se diferencia la forma de acceso según la plataforma.

```
SDL_Renderer *renderer = SDL_CreateRenderer(window, -1, 0);

#ifdef OGRE_PLATFORM == OGRE_PLATFORM_ANDROID
    Ogre::FileSystemLayer* mFSLayer = OGRE_NEW_T(
        Ogre::FileSystemLayer, Ogre::MEMCATEGORY_GENERAL)(
            OGRE_VERSION_NAME);
    SDL_Surface* surface = SDL_LoadBMP(mFSLayer->getConfigFilePath(
        "loading.bmp").c_str());
#endif
// Load image as SDL_Surface
#ifdef OGRE_PLATFORM != OGRE_PLATFORM_ANDROID
    SDL_Surface* surface = SDL_LoadBMP("loading.bmp");
#endif
```

Listado 5.12: Carga de la pantalla de Loading. Se aprecia la diferencia de código según la plataforma.

Conflictos entre SDL_Image y Ogre3D

Una de las características interesantes que posee SDL es que está modularizado. Así se puede incluir únicamente aquellos módulos que sean necesarios. En el desarrollo, se quiso incluir el Módulo SDL_Image en el proyecto, para poder cargar una pantalla de Loading que estuviera en formato png (todos los formatos de imagen distintos de BMP los maneja el módulo SDL_Image).

Sin embargo SDL_Image tiene como dependencia la librería FreeImage, en una versión que es incompatible con la versión que necesita Ogre3D, puesto que Ogre3D también la tiene como dependencia. Se solventó el problema almacenando la pantalla de carga como formato BMP, para que no hiciera falta el módulo SDL_Image.

Carga automática de Plugins

La carga automática de plugins de Ogre no está soportada para Android, por lo que es necesario cargarlos de forma manual.

5.5 Análisis de Costes

En el caso del presente trabajo, es necesario separar costes de investigación para la implementación en la plataforma Android y establecimiento de un entorno de desarrollo que sea operativo también en LINUX, respecto a costes de desarrollo del propio juego en sí.

Puesto que los costes que no existen (o son despreciables) los costes en Hardware y Software necesarios para el desarrollo, se tendrán en cuenta solamente los costes humanos.

5.5.1 Investigación y establecimiento de entorno de desarrollo.

Ha sido necesaria una ardua labor de investigación para poder establecer un entorno de desarrollo para la plataforma Android que funcionara correctamente. El desconocimiento de dicha plataforma para desarrollos nativos, así como la escasez de documentación de calidad al respecto han determinado que las horas de investigación sean con mucho la mayor parte del coste de este proyecto.

Se empezó a investigar la plataforma Android a partir del 1 de Noviembre de 2015. La dedicación ha ido subiendo y bajando, según el tiempo que se ha podido dedicar al trabajo. En la tabla 5.1 se puede observar una relación aproximada de las horas invertidas, en total unas 860 horas.

Periodo	Horas por Semana	Horas Totales
01/11/2015 - 31/12/2015	12	96
01/01/2016 - 01/03/2016	20	160
01/03/2016 - 20/06/2016	40	600
20/06/2016 - 04/07/2016	2	4

Cuadro 5.1: Coste de Investigación en horas.

Asumiendo un sueldo medio de un desarrollador de juegos, se puede aproximar una tasa de 25\$ por hora trabajada. Así, el coste de la investigación y establecimiento del entorno de desarrollo alcanza los 21.500\$.

5.5.2 Desarrollo del prototipo

Debido a la labor de investigación realizada, el coste del desarrollo ha sido bastante bajo (5.2).

Periodo	Horas por Semana	Horas Totales
15/05/2016 - 20/06/2016	10	40
20/06/2016 - 04/07/2016	80	160

Cuadro 5.2: Coste de Desarrollo en horas.

Asumiendo la misma tasa del apartado anterior (25\$ por hora), el coste del desarrollo se puede cifrar en 5000\$.

5.6 Monetización y Marketing

En este apartado se presenta un breve plan de comercialización y de publicidad del juego, explorando cual es la mejor forma de obtener beneficios con el juego.

Aunque el resultado de este PFC es un pequeño prototipo que demuestre la viabilidad o no de un desarrollo independiente para dispositivos móviles en el mercado de los videojuegos, se pasa a continuación a estudiar aspectos de una futura comercialización.

5.6.1 Definición del producto y mercado objetivo

Este juego ha sido diseñado teniendo en mente el mercado para dispositivos móviles, concretamente Android. Dicho mercado tiene ventajas muy interesantes respecto a otras plataformas de distribución:

- Solo requiere una cuota de 25\$ para poder publicar aplicaciones en la plataforma Google Play.
- Es una plataforma con millones de usuarios, con un amplio espectro de usuarios.
- Existe mucha documentación sobre la publicación de aplicaciones en Google Play.
- Si la aplicación es gratuita (Porque utilice un modelo de negocio basado en compras integradas o en anuncios), el usuario puede descargarla sin facilitar información económica, lo que la hace más accesible.

Este juego buscar ser un juego casual, que el usuario puede tener en la memoria de su SmartPhone sin molestar, y que pueda utilizarse para pequeños ratos de ocio.

5.6.2 Modelos de negocio

Hay muchos modelos de negocio posibles en negocio de las aplicaciones para móviles. Como se menciona en [26], se pueden distinguir 12 modelos de negocio:

De marca Es un modelo en el que no se busca la rentabilidad económica, sino mejorar la reputación de un empresa o aportar publicidad a la misma. No es un modelo adecuado para los videojuegos en general, aunque podría utilizarse en aplicaciones accesorias a los mismos, que publicaría la misma empresa autora para dar más notoriedad al videojuego.

Sin duda este modelo no es aplicable a TOWER3D.

Parte del Servicio Solapándose un poco con lo mencionado anteriormente para en el modelo de marca. En este caso se basa únicamente en aplicaciones accesorias o que forman parte de un servicio más global.

De la misma manera que el anterior, no es aplicable a TOWER3D.

Publicidad En este caso, la aplicación puede descargarse de manera gratuita. La rentabilidad se obtienen porque al usuario se le muestra publicidad, normalmente servida por una tercera empresa. Dicha empresa es la que paga al autor de la aplicación, normalmente en función de los accesos que se hayan realizado a las páginas publicitadas o bien del número de anuncios que haya visto el usuario.

Puede llegar a ser molesto para el usuario, si el número de anuncios es excesivo, si interrumpe el ritmo de juego o si los anuncios son a pantalla completa y no pueden saltarse.

Es un modelo de negocio muy utilizado en aplicaciones independientes, ya que gracias a que la aplicación es gratuita, la distribución de la aplicación es mayor.

Es uno de los modelos que podrían aplicarse a TOWER3D.

Compras Integradas En este modelo, la aplicación es gratuita. Sin embargo, dentro de la propia aplicación es posible comprar ciertos objetos que dan ventaja al usuario respecto al resto de jugadores. Es muy utilizado en juegos que tienen un fuerte componente social, ya que la competencia facilita que los usuarios se decidan a comprar dentro de la aplicación. En la figura 5.17 se pueden apreciar las compras integradas de un videojuego.

Este modelo podría aplicarse perfectamente a TOWER3D, aunque habría que determinar que objeto podría comprar los jugadores.



Figura 5.17: Compras en Pokemon GO (desarrollado por Niantic).

Información Son aplicaciones gratuitas en las que el usuario necesita introducir cierta información personal que luego la empresa autora de la aplicación puede vender a terceros para obtener rentabilidad.

No es modelo aplicable a TOWER3D.

mCommerce Son aplicaciones que utilizan la aplicación como un canal de ventas para sus productos. Están directa y únicamente enfocadas a la venta de productos, buscando que

la facilidad de compra proporcionada por la aplicación permita aumentar sus ventas.

Claramente este modelo no es aplicable a TOWER3D.

Audiencias La descarga de estas aplicaciones son gratuitas, y tienen como objetivo la obtención de una gran base de usuarios. No se puede aplicar a TOWER3D.

Afiliación De descarga gratuita, el beneficio se obtiene mediante la comisión por venta de aplicaciones de terceros. Un claro ejemplo son los buscadores de hoteles o vuelos. En la figura 5.18 Una vez más, no es aplicable a TOWER3D.



Figura 5.18: Kayak, un buscador de vuelo, se rentabiliza por comisiones de venta.

Suscripciones Son aplicaciones en las que el usuario paga por tiempo de uso. Suelen realizarse pagos de forma periódica, semanal, quincenal, mensual, etc...

Tampoco es un modelo aplicable a este tipo de videojuegos.

Precios Bajos Con estas aplicaciones se busca el beneficio en una cantidad muy alta de compras. Al obtener una rentabilidad mínima con cada descarga, es necesario una enorme cantidad de ellas para poder rentabilizar el juego. Este modelo se utiliza en muchos videojuegos independientes.

Es uno de los modelos que podrían tener buen resultado con TOWER3D.

Precios Altos En este modelo, el precio de la aplicación es sensiblemente más alto y debe ajustarse con precisión en función del público objetivo, puesto que se corre el riesgo de que el alto precio eche para atrás al posible comprador. En la figura 5.19 se puede ver una captura de pantalla de Minecraft Pocket, de precio actual 6,99 \$. Es un claro ejemplo de este modelo de negocio.



Figura 5.19: Minecraft: Pocket Edition (desarrollado por Mojang).

Aunque es un modelo aplicado en algunos videojuegos, es necesario casi siempre tener ya un mercado de otra plataforma, o bien un público muy fiel.

Este modelo, aunque utilizado para algunos videojuegos, no es apropiado para TOWER3D.

Sin modelo de Negocio claro Ciertas aplicaciones se publican si un modelo de negocio claro, buscando más la satisfacción personal y que guste al usuario.

En este caso no se utilizará este modelo de negocio, ya que se desea conseguir una rentabilidad.

De entre los modelos vistos, se optaría por una fusión de dos modelos:

Publicidad Se incorporarán anuncios al juego, de tal manera que se pueda obtener un beneficio sin que sea el usuario el que pague por ello.

Precios Bajos Existirá una segunda versión del juego, a un precio muy bajo, que permitirá al usuario jugar sin publicidad.

5.6.3 Cálculo de amortización

En este apartado se va a realizar un cálculo simplificado del nivel de descargas necesarias para que el juego empiece a producir beneficios.

Teniendo en cuenta que el coste total del videojuego es de 26.500\$, se va a proceder a un breve cálculo del número de descargas necesarias para amortizar el juego. Se tienen en cuenta las siguientes condiciones:

- El precio de la versión sin publicidad del juego es de 1\$.
- El Costo por mil (CPM) de visualizaciones se paga a 1\$ y el click a 0.2\$ (Acceso a la página publicitada).

- Se supone que un 80 % de los usuarios, descargará la aplicación con publicidad, mientras que un 20 % la descargará sin publicidad.
- Asumimos que cada usuario verá una media de 5 anuncios al día.
- Asumimos también que un 2 % de los usuarios de la aplicación con publicidad hará click en un anuncio al día.

En la tabla 5.3 se puede observar un ejemplo de cuadro de amortización basado en el número de descargas. Si bien el mayor ingreso inmediato lo da las descargas, hay que tener en cuenta que el ingreso por anuncios es mensual (se ha hecho el cálculo para 30 días).

Tipos de Beneficio	10.000 Descargas	20.000 Descargas	40.000 Descargas
Compra de Aplicación(20 %)	2000\$	4000\$	8000\$
Visualizacion Anuncios(80 %) (0.001\$ * 30 días * 5 anuncios)	1200\$	2400\$	4800\$
Clicks en Anuncios(0.16 %) (0.2\$ * 30 días)	960\$	1920\$	3840\$
Total	4160\$	8320\$	16640\$

Cuadro 5.3: Amortización según número de descargas.

5.6.4 Marketing

Para dar publicidad al juego, se proponen varios frentes.

Reseñas en páginas especializadas Existen en internet una cantidad enorme de blogs y páginas dedicadas a los videojuegos, muchas de ellas muy centradas en Android. Una buena manera de publicitar el juego es ofrecerlo de manera gratuita a todos los blogs especializados que sea posible para que lo reseñen. De esta manera, el juego recibe una publicidad muy importante, gracias a la popularidad de dichos blogs. En alguno de los blogs más importantes, no es extraño que haya que invertir algo de dinero para que realicen la reseña.

También es importante estar seguro de ofrecer un producto de calidad, ya que una mala reseña podría hundir la popularidad del producto.

Youtubers Todo lo explicado anteriormente para las páginas especializadas, aplica también a una serie de canales o personas que suben videos a Youtube y que tienen muchos seguidores.

Imagen de Marca Es necesario crear alrededor del juego un entorno en internet que el posible comprador pueda utilizar para informarse sobre el producto, hacer preguntas o consultar un Frequently Asked Questions (FAQ).

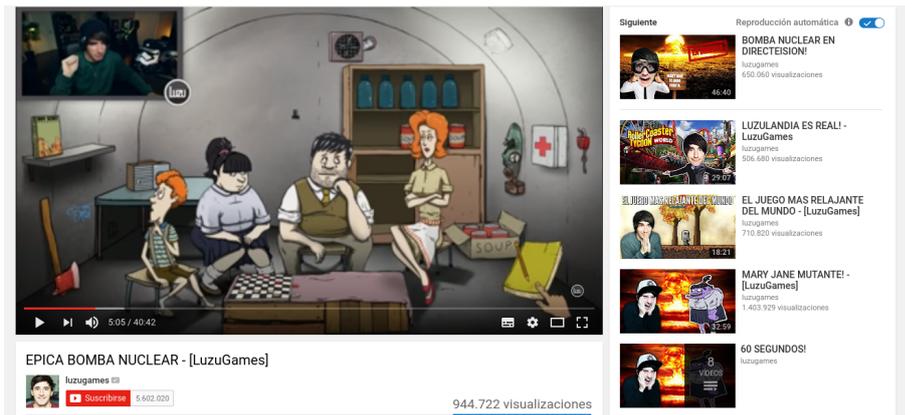


Figura 5.20: Imágen de un video del Canal LuzuGames, especializado en juegos.

Es muy importante ser muy cuidadoso con la página web, para dar sensación de profesionalidad con la misma, evitando por ejemplo que la página principal de la aplicación esté en un plataforma diferente como FaceBook.

Redes Sociales Es importante darle a la máxima visibilidad en las redes sociales, ya que son un canal de comunicación barato y con un enorme potencial de venta.

Mostrar un perfil muy activo en redes sociales suele permitir captar la atención de posibles compradores.

Publicitar Anuncios En la mayoría de las redes sociales (especialmente relevante en FaceBook) se permite al usuario promocionar su página para que aparezca como link patrocinado.

Es una manera de expandir el público objetivo como una inversión monetaria no muy alta.

Capítulo 6

Conclusiones y propuestas

6.1 Conclusiones

A continuación se indican los diferentes objetivos que se han planteado y el grado de éxito en la consecución de los mismos.

6.2 Objetivos de diseño

Los objetivos de diseño marcados son los siguientes.

Drag and Drop Se han implementado botones en un tamaño de 110 x 110 píxeles, que son lo suficientemente grandes para ser pulsados con el dedo. En la figura 6.1 se puede ver un ejemplo de botón.



Figura 6.1: Ejemplo de un botón del HUD.

La manera en la que el usuario construye torres sigue el paradigma del Drag and Drop, ya que debe arrastrar el botón de la torre que desea construir al lugar del mapa donde se desea construir.

Esto se ha conseguido lanzando rayos desde la posición del ratón hasta que colisio-

nan con la geometría estática del mapa. Una vez se conoce el punto de colisión, se posiciona la torre en dicho lugar de colisión.

Debido a que no se han implementado ningún sistema de casillas o grid que determinen los puntos de posicionamiento válido de las torres, el movimiento de la torres es fluido, aunque ha sido necesario implementar una lógica que impida que una torre se construya en un espacio que colisione con otra torre.

Tanto para esto, como para evitar que las torres se construyan en los caminos, se han utilizado Axis Align Bounding Box para detectar colisiones.

Sombreado Para conseguir este objetivos, se ha utilizado la técnica que propone Ogre3D de sombras mediante mapas de texturas, con un tamaño de textura de sombra de 1024. Se hicieron pruebas con un tamaño de textura de sombra de 512, pero la sombra no quedaba bien definida.

Para la iluminación, se ha incluido una única luz de tipo foco, situada muy por encima del mapa y ligeramente a la izquierda, apuntando hacia el origen del mundo de Ogre3D. Se realizó una ligera rectificación de la posición del foco, centrándolo un poco más para evitar unas sombras demasiado alargadas, que ralentizaban la ejecución del mismo.

Mejora de Torres Gracias al sistema de configuración que se ha incluido en varios aspectos del juego, incluir las mejoras de las torres ha sido realmente sencillo. Simplemente se trata de definir torres con el mismo nombre que la torre que se quiere mejorar, añadiendo al nombre el nivel de la torre (si la torre es de nivel 0, no es necesario añadir nada). Esto permite cambiar cualquier característica de la torre al mejorarla, incluida la malla 3D.

Es una opción muy interesante el poder cambiar el modelo 3D de una torre cuando se va mejorando, ya que ofrecemos al usuario información a primera vista del nivel en el que se encuentra una torre.

Además, el sistema explora automáticamente el archivo de configuración de torres para ver si la torre actual puede mejorarse, buscando una definición de torre con el mismo nombre, pero un nivel más. Si no existe, no se muestra el icono de mejorar la torre.

HUD sencillo y claro Para la implementación del HUD se han utilizado Overlays, que al ser manejadas por scripts, han permitido realizar pruebas rápidas sin necesidad de recompilar el proyecto. Aunque los Overlays no funcionan todo lo bien que deberían (en ocasiones es necesario añadir paneles vacíos para conseguir la correcta visualización del Overlay).

Se ha utilizado una fuente clásica que facilita la legibilidad y es reconocible por el usuario, la fuente Arial.

Se han distinguido con diferentes colores las etiquetas fijas de los valores que pueden ir cambiando para facilitar la lectura del HUD. Además, se ha dotado a cada texto de un fondo semitransparente que los hace resaltar respecto al mapa, con lo que se evita que se confunda con el fondo.

En la figura 6.2 se pueden apreciar los detalles del sombreado. También se puede apreciar que se ha añadido un fondo semitransparente a los mensajes que pueden aparecer en pantalla, para resaltarlos sobre el mapa. El color rojo del mensaje también se ha incluido para captar la atención del jugador.

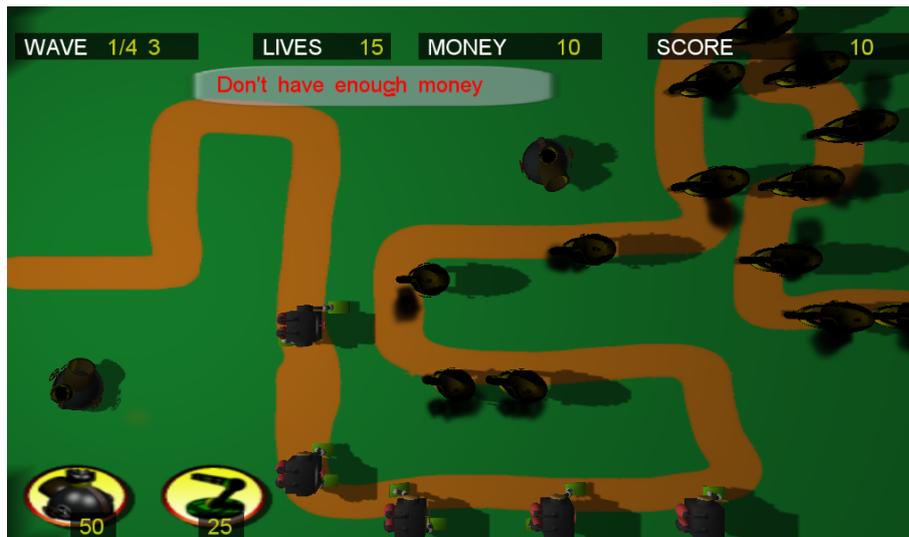


Figura 6.2: Ejemplo de un botón del HUD.

Sonido Se han incluido pequeños efectos sonoros a los disparos de las torres, que varían según su tipo. Los sonidos tienen licencia CC-0 y se ha obtenido de la página www.freesound.org.

En conjunto, se puede decir que los objetivos de diseño se han cumplido, aunque existe un amplio margen de mejora para los aspectos gráficos y sonoros del juego.

6.3 Objetivos técnicos o de implementación

Código en C++ C++ ha sido el lenguaje utilizado para todos los aspectos del videojuego (excluyendo por supuesto el lenguaje de scripting propio que utiliza Ogre3D para la definición de materiales, overlays, etc...).

Se ha hecho especial énfasis en el uso de las funcionalidades propias de C++ 11, como las funciones lambda.

Se ha hecho un uso masivo de las funcionalidades ofrecidas por las STL, sobre todo en cuanto a las operaciones sobre vectores.

Ha sido necesario, en algunos casos, realizar pequeños ajustes debido a que las librerías STL que incluye el NDK de Android no incluyen el soporte completo. Por ejemplo no ha sido posible utilizar la función `stof` por no estar soportada en Android.

Multiplataforma Ha sido necesario un esfuerzo importante para conseguir que el juego funcionara en la plataforma Android. No así para que el juego funcionara de forma correcta en Linux. La diferencia de esfuerzo para una plataforma u otra radica en la ausencia de documentación fiable para la plataforma Android.

Así mismo, ha sido necesario el uso masivo de directivas del procesador, para bifurcar el código en ciertos puntos en el que el código de Android y de Linux debe ser absolutamente diferente. Para ello se ha utilizado el archivo de cabecera `OgrePlatform.h`, que da valor a la variable de preprocesado `OGRE_PLATFORM`.

En el listado 6.1 se puede observar una bifurcación de código en la gestión de ventanas.

```
#if OGRE_PLATFORM == OGRE_PLATFORM_ANDROID
JNIEnv* env = (JNIEnv*)SDL_AndroidGetJNIEnv();

jclass class_sdl_activity = env->FindClass("org/libSDL/app/
    SDLActivity");
jmethodID method_get_native_surface = env->GetStaticMethodID(
    class_sdl_activity, "getNativeSurface", "()Landroid/view/
    Surface;");
jobject raw_surface = env->CallStaticObjectMethod(
    class_sdl_activity, method_get_native_surface);
ANativeWindow* native_window = ANativeWindow_fromSurface(env,
    raw_surface);

if ( !native_window )
    return;
opt["externalWindowHandle"] = Ogre::StringConverter::toString( (
    uint)native_window );
#endif
```

Listado 6.1: Ejemplo de bifurcación de código mediante el uso de `OGRE_PLATFORM`.

Respuesta en tiempo real Han sido necesarias varias optimizaciones para conseguir que el juego funcione de manera fluida en dispositivo móvil.

Sin duda la más importante (también la que más impactó en el desarrollo) fue el cambio de enfoque respecto a la construcción de la geometría del mapa. Se pasó de un mapa construido por tiles a un mapa formado por una única geometría estática. Este cambio mejoró mucho el rendimiento en dispositivo móvil.

Otro cambio necesario fue en los efectos de sonido. Fue necesario recortar pequeños intervalos al principio del sonido de disparo, para que el sonido fuera más sincronizado con la animación de disparo.

Otro cambio que mejoró ligeramente el rendimiento, fue el calcular la ruta que deben seguir los enemigos una única vez, ya que la ruta no varía en todo el juego.

Editor de niveles El editor de niveles se ha realizado en Python, para poder ser ejecutado dentro de Blender. Es un editor muy sencillo, del que ya se han dado detalles en otro apartado anterior.

El editor permite definir mediante un grid de Blender, el camino que utilizan los enemigos para cruzar el mapa, incluyendo su punto de inicio y su punto final.

Al realizar el editor dentro de Blender, permitirá realizar de forma sencilla mapas diferentes, y dotarles de detalle, incluyendo objetos de relieve como árboles, piedras, arbustos o cualquier otra cosa.

Diseño de texturas Para el diseño de texturas se ha utilizado GIMP. El procedimiento general de diseño de texturas ha sido la exportación del mapa UV a una imagen. Dicha imagen se incorpora después a GIMP como una capa con un nivel de transparencia alto y se va dibujando encima.

Se ha hecho uso de algunos filtros (como el desenfoque de movimiento) de GIMP para conseguir diversos efectos. En la figura 6.3 se puede observar un ligero efecto de metalizado.

PathFinding El sistema de PathFinding ha sido implementado gracias a la librería Boost, que ofrece diversos algoritmos de búsqueda.

Puesto que los caminos no tienen pesos, se ha optado por una sencilla búsqueda en anchura, que permite obtener el camino para cada enemigo.

En un principio, cada enemigo sólo conocía el punto siguiente de la ruta al que dirigirse, de tal manera que cuando lo alcanzaba, tenía que volver a ejecutarse el algoritmo de PathFinding para conocer el punto siguiente.

Esto se ha cambiado, actualmente sólo se realiza el cálculo una única vez para todos los enemigos. Sin embargo, no es descartable que en un futuro se vuelva a la lógica anterior, ya que permitiría al enemigo reaccionar ante cambios del entorno o poder utilizar mapas con múltiples caminos.

Seguimiento de enemigos Para el seguimiento de enemigos, se consideró y probó primero utilizar el sistema de AutoTracking que permite en Ogre3D que un nodo de escena se oriente hacia otro nodo de escena. Sin embargo, a pesar de que a dicha orientación se le pueden poner restricciones, no se consiguieron resultados satisfactorios (las torres giraban sobre ejes que en teoría les habían sido restringidos, y acababan volteándose hacia arriba).

Además, el AutoTracking era instantáneo, con lo que no era posible definir una velocidad de rotación.

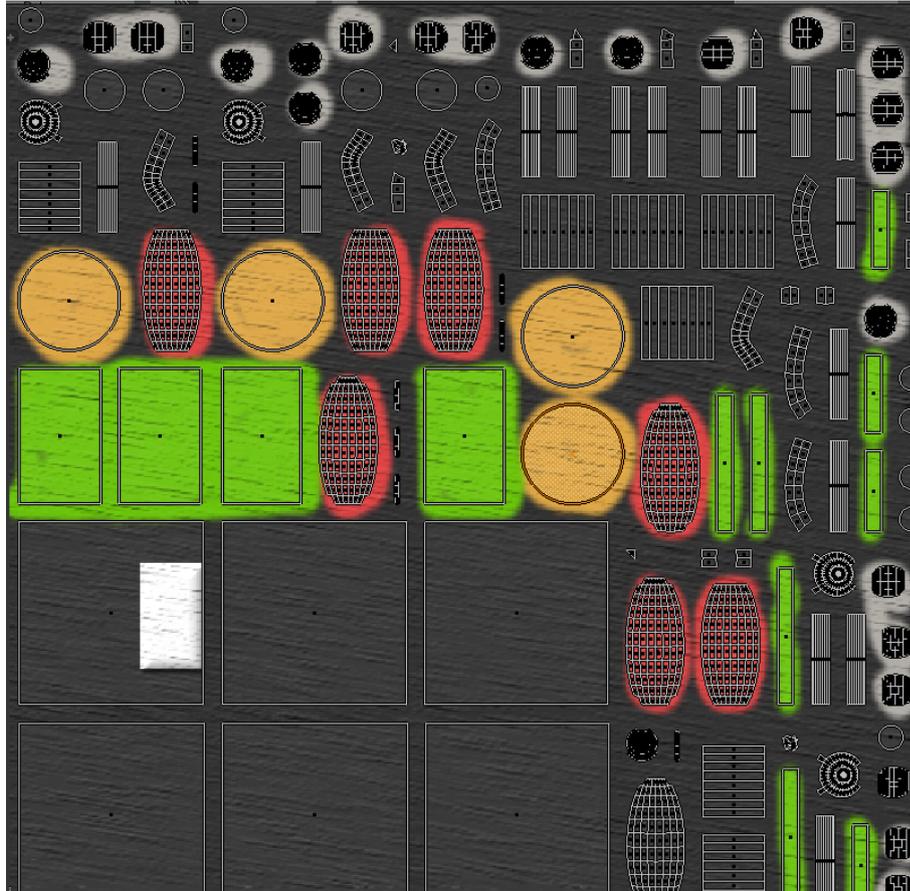


Figura 6.3: Textura del enemigo de tipo Robot. El efecto de metalizado se ha conseguido mediante la aplicación de varios filtros.

Descartado pues el AutoTracking, se realizó un pequeño algoritmo que orientaba la torre hacia el enemigo realizando rotaciones sobre el eje Y. A grandes rasgos, el algoritmo determinaba el ángulo entre el eje X local de la torre y el vector que apuntaba hacia el enemigo. Según el signo de dicho ángulo, se rotaba sobre el eje Y en un sentido o en otro, siempre una cantidad de grados que no fuera mayor que la determinada por la velocidad de rotación de la torre ni mayor que el ángulo que formaban los dos vectores mencionados anteriormente.

Almacenamiento de records Se ha implementado un sencillo sistema de almacenamiento de records en texto plano.

Configurable El juego es totalmente configurable.

Es posible añadir enemigos nuevos sin necesidad de añadir una sola línea de código. Solamente hay que añadirlo al fichero `enemys.cfg`.

Es posible añadir nuevos tipos de torres o mejoras de las torres ya existentes. En este caso hay que realizar dos operaciones:

- Es necesario añadir la torre al archivo `towers.cfg`, en que se especificarán todas

las características de la misma.

- Es necesario añadir al fichero de overlay el botón que permita añadir la nueva torre. Esto no requiere compilación y es bastante sencillo

También es posible cambiar las diferentes oleadas de enemigos, alterando el número de enemigos, su tipo o el intervalo con el que salen. Solamente es necesario modificar el archivo waves.cfg. También se puede, por supuesto, añadir nuevas oleadas para aumentar la duración del juego.

Tamaño pequeño El tamaño de la APK es de unos 11MB, de los cuales la mayoría pertenecen al propio Ogre3D. Es un tamaño que se encuentra por debajo de la media de lo que ocupan los juegos 3D actuales.

Uso de Software Libre Absolutamente todas las herramientas y librerías utilizadas para el desarrollo de este proyecto han sido Software Libre

6.4 Conclusiones globales

De la experiencia desarrollando este proyecto, se han extraído las siguientes conclusiones:

- Se podido constatar que utilizar una librería que no está siendo utilizada de forma activa para desarrollar en una plataforma concreta, aumenta enormemente la dificultad de cualquier desarrollo. Hay una gran falta de documentación de calidad para Ogre3D en Android, en incluso algunos bugs.

La mayor fuente de documentación que existe en para Ogre3D en Android es en los propios foros de la comunidad, si bien, muchas veces encontrar la respuesta a un tema concreto es complicado.

También es cierto que la comunidad es muy activa y suelen responder con prontitud a la dudas formuladas en el foro, aunque no siempre conocen la respuesta correcta, y a veces es complicado formular la pregunta correcta.

La mayoría de la documentación que existe en la wiki de la aplicación esta horriblemente desactualizada, en particular por ejemplo lo referente al sistema de overlays, que fue extraído como componente, pero no se reflejó en la documentación.

Tampoco ayuda que el último libro publicado sobre Ogre3D hable de la versión 1.7, no existiendo documentación más reciente para la versión actual.

- Implementar un juego en el que sea posible incluir nuevos elementos en el juego mediante lenguaje de scripts o archivos de configuración, facilita mucho posteriores ampliaciones.

El utilizar C++11 tiene grandes ventajas a la hora de codificar, aunque tiene una desventaja bastante importante, que es el tiempo de compilación. Con c++11 el tiempo de

compilación llega a multiplicarse por 5 fácilmente en algunos casos.

El tiempo para compilar TOWER3D con C++11 puede llegar a exceder los 8 minutos, lo que hace muy incómodo la introducción de pequeños cambios.

Haciendo el sistema lo más configurable posible, se evita parcialmente esta desventaja, además de ganar la posibilidad de añadir nuevas opciones al juego sin necesidad de escribir una línea de código.

También es necesario destacar que el sistema de scripting de los overlays tiene errores importantes, como por ejemplo, es necesario añadir un panel con texto vacío antes de añadir un panel con texto, porque el primer panel con texto no se dibuja en la pantalla (el texto aparece vacío).

- Es muy importante conocer bien la plataforma sobre la que se trabaja, para evitar realizar trabajo que luego no sea productivo, como ha sucedido en este proyecto con el diseño de mapas mediante tiles.

El error fue ocasionado por la falta de conocimiento del hardware que iba a ejecutar la plataforma, sumado a un mal diseño inicial. El coste en horas de este error excede el 10% de las horas empleadas en el desarrollo del videojuego, lo que es un lastre importante.

En este caso, este problema si que se encontraba documentado en la Wiki de Ogre 3D, bajo la sección de problemas comunes.

- La primera vez que se monta un entorno de desarrollo para una plataforma que se desconoce, el tiempo invertido en ello puede dispararse.

El establecimiento de un entorno de desarrollo que fuera capaz de mostrar una figura 3D en Android fue una pesadilla, y culpable de más del 80% de las horas de investigación invertidas.

Los problemas de compilación no documentados de Ogre3D, salvo en los foros, los bugs que presenta en el manejo de contextos externos EGL, y sobre todo y ante todo, la falta de un modo verboso de log en la ejecución de Ogre3D, hizo que las horas de investigación se dispararan hasta poner en peligro el desarrollo del proyecto.

La frustración generada por esta situación impacto también fuertemente en el rendimiento de las horas invertidas.

- Que Android esté tan diversificado a veces puede no ser bueno para el desarrollador, ya que cada dispositivo hay veces que es un mundo para trabajar en modo desarrollador.

Se han realizado pruebas en diferentes dispositivos con resultados varios como:

- Dispositivos que no son detectados por Android Debug Bridge (ADB) en el ordenador con el que se va a desarrollar, pero que si son detectados en otros, que tienen exactamente la misma configuración.

- Dispositivos que fallan al intentar habilitar el modo de depuración de Android.
- Dispositivos que no dejan instalar mediante adb las aplicaciones, por fallos en la validación de la firma (a pesar de estar deshabilitada). Sin embargo, sí que permiten instalar directamente desde el archivo acAPK en la tarjeta Secure Digital (SD).

Al final, es necesario disponer de una batería de dispositivos amplia para poder probar diferentes opciones.

- El tiempo invertido en investigación probablemente se hubiera reducido a la mitad si primero se hubiera implementado una base SDL sobre Android, y luego utilizar Ogre3D sobre SDL. Realmente SDL está mucho más consolidado que Ogre3D, sobre todo para la portabilidad en Android.

SDL funciona de maravilla en Android. No tiene problemas con la gestión de contextos EGL, y provee de funciones avanzadas como invocar al teclado de Android.

También la gestión con las actividades de Android es mucho mayor, ya que incluye el código necesario para poder minimizar las aplicaciones sin que esto afecte a la aplicación.

La integración de sonido con Android funciona muy bien, sin ningún tipo de problema.

- Es un problema que no exista un estándar para la construcción de código en C++. Las librerías oscilan entre CMake, make y utilidades propias que generan los flags de compilación adecuados para construir la librería para la plataforma deseada.

Realmente es un problema cuando compilar una librería en concreto lleva más tiempo que el invertido en el desarrollo de una aplicación.

- Los tiempos de compilación de Ogre3D son un problema que se reduce notablemente si en vez de incluir «Ogre.h», se incluye solamente las cabeceras que se necesiten. Además, el uso del Class Forwarding también ayuda a que los tiempos de compilación no se disparen mucho.

Realmente la inclusión de la cabecera genérica de Ogre, puede llegar a multiplicar por dos el tiempo de compilado. Incluyendo exactamente las cabeceras necesarias, la compilación se hace un proceso más llevadero.

El Class Forwarding, a pesar de su limitada aplicación, realmente aligera el parseo de cabeceras que al final es lo que lleva más tiempo en la compilación

- Ogre3D no tiene una gestión de errores excesivamente buena. La mayoría de los fallos que se han ido encontrando se han solucionado mirando el código fuente de Ogre. Es un problema muy serio que existan errores que no se vean reflejados en el log. (Como shaders necesarios para GLES2 o la falta de Edge List a la hora de generar sombras).

Este problema quizá es el más grave de todos. Ogre3D no avisa cuando le falta información para llevar a cabo una operación, simplemente no la lleva a cabo y punto. En el caso de que si un modelo no incluye la lista de aristas no genere sombra, al menos se merecería un warning en el Log, si no un error.

Como valoración final, y dando respuesta a la pregunta que se presentaba al inicio de PFC de si es posible crear un juego para Android con software libre, podemos decir que si es posible, pero con matices.

Lo primero a tener en cuentas es, que si solamente se va a realizar un único juego, el coste de montar el entorno hace que el juego no sea rentable. En el caso de que se vayan a desarrollar una batería de pequeños juegos es algo más sencillo amortizar el coste inicial.

En cualquier caso, a la vista de las cifras obtenidas, parece mucho más rentable utilizar un entorno comercial, como Unity, que da toda clase de facilidades y funciona Out of the Box.

6.5 Propuestas de trabajo futuro

En este apartado se pretende mostrar diferentes acciones que pueden realizarse para mejorar el prototipo de cara a su posible comercialización.

6.5.1 Mejoras en la calidad gráfica

Sin duda, uno de los apartados a mejorar claramente es la calidad gráfica. Es necesario realizar mejoras en diferentes apartados:

Mejora en el texturizado de modelos En general, las texturas utilizadas son demasiado simples, sin conseguir el efecto de material realista que deberían reflejar. Uno de los pasos más importantes a realizar podría ser realizar un proceso de Baking de la textura, para mejorar la calidad de la misma.

Baking en general se refiere al proceso en el cual se realiza un precálculo de algo para agilizar su uso posteriormente. En concreto, el proceso de Render Baking es la creación o modificación de una textura de un modelo en referencia a un sistema de iluminación, normalmente complejo. Al realizar este proceso de forma separada al juego, es posible invertir mucho más tiempo en el mismo, generando texturas de más calidad [4].

Gracias al Baking, podemos conseguir que un modelo genere sombras sobre sí mismo, lo cual no está permitido en Ogre3D en el caso de sombreado por mapa de texturas.

El proceso tienen las siguientes ventajas:

- Es posible utilizar una figura de mayor densidad poligonal para hacer el baking, y luego utilizar la textura generada sobre otro modelo más ligero, que pueda utilizarse en el juego. De este manera, dicho modelo gana mucho realismo, sin suponer una mayor carga para el juego.

- Permite utilizar una configuración de luces muy compleja para generar la textura, ya que al estar precalculado no afectará al rendimiento del juego (que si se vería resentido por el uso de muchas fuentes de luz).
- Es posible utilizar modelos de iluminación que no estén permitidos por el motor de juego, como la radiosidad.
- Se pueden crear texturas que imiten materiales reflectantes de forma sencilla.

En la figura 6.4 se puede observar un ejemplo de Baking.



Figura 6.4: Un ejemplo de Baking para mejorar un modelo de baja densidad poligonal.

Mejoras en el sistema de partículas Actualmente se utilizan sistemas de partículas únicamente para simular el humo del escape de un tipo de enemigos. Sería conveniente utilizarla para algunas de las siguientes acciones:

Cuando las torres disparan Las torres deberían despedir humo y partículas cuando realicen un disparo.

Esto aportaría mayor realismo y mejoraría el aspecto dinámico de las torres. Otros efectos posibles serían que las torres levantaran un breve polvo al girar.

Con el impacto de los proyectiles El impacto actual de proyectiles no realiza ningún efecto visual mas allá de la aséptica desaparición del proyectil y de los enemigos destruidos por él.

Sería interesante incorporar algunos efectos de explosión o fuego cuando el proyectil impacta. El efecto debería ser diferente para cada tipo de proyectil,

Al mejorar una torre Las partículas cuando se producen evoluciones o mejoras en algún aspecto del juego, son motivadoras para el usuario y mejoran el aspecto global del juego. Resaltan este aspecto del juego visualmente.

Destrucción de enemigos Se podrían utilizar partículas para simular la destrucción en pequeñas y diferentes piezas de un enemigo cuando es impactado por un proyectil. De nuevo, este tipo de efectos consiguen producir un efecto motivador en el jugador, que hace que el jugador desee volver a jugar.

Para implementar estos diferentes sistemas de partículas, existe una herramienta publicada por el usuario akem321 del foro de Ogre3D que permite realizar pruebas y experimentar con diferentes parámetros del sistema de partículas (disponible en <http://www.ogre3d.org/forums/viewtopic.php?f=11&t=61559>). En la figura 6.5 se puede ver el aspecto de la aplicación.

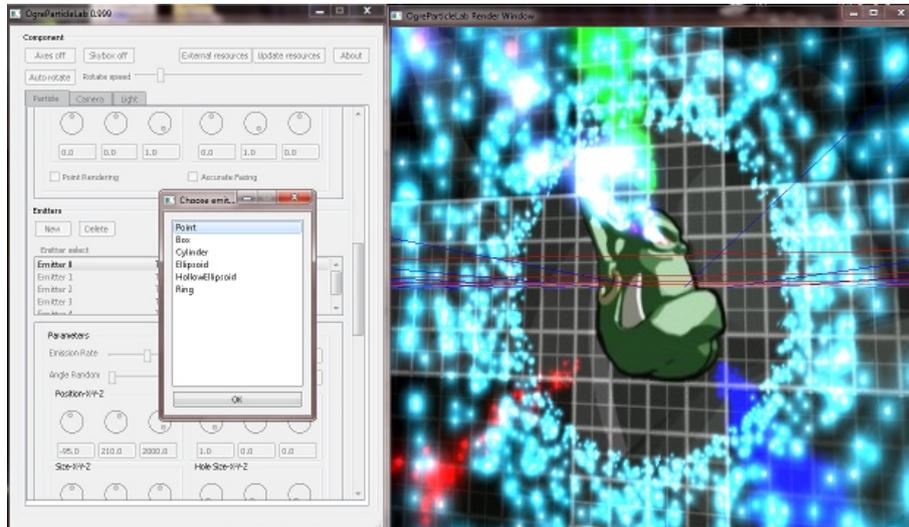


Figura 6.5: Ogre3D Particle Lab que permite experimentar con diferentes parámetros de partículas. Funciona tanto en Windows como en Linux (bajo Wine).

Mejoras en la interfaz de usuario La interfaz de usuario es mejorable en muchos puntos. Es una inversión que no sería muy costosa y que sería interesante, ya que mejoraría mucho la interfaz del juego

- Los botones deberían tener un efecto de pulsado. Actualmente, la imagen no varía cuando es pulsado por el jugador, dando la impresión de estar formando parte de una imagen conjunto en vez de ser un control separado.
- La visualización del panel de records es muy mejorable.
- La pantalla de carga es demasiado simple. Podría contener una escena renderizada del juego a gran calidad, mejorar con efectos y filtros de un programa de retoque de imágenes como GIMP.

Es típica en algunos juegos la inclusión de pequeños consejos al jugador cuando el juego está cargando. De esta manera se ameniza la espera y el jugador consigue información valiosa. En la figura 6.6 se puede observar un buen ejemplo de lo expuesto en este punto.

- Mejorar la combinación de colores. Es necesario escoger una paleta de colores para la interfaz de usuario y utilizar de forma constante durante todo el juego. Esto crea una apariencia de cohesión que refuerza el estilo del juego.



Figura 6.6: Pantalla de carga de juego Clash of Clans (desarrollado por SuperCell). Se puede ver el dibujo de gran calidad, junto con una barra de progreso y un consejo ofrecido al jugador.

Niebla y otros efectos Para disimular un poco la simpleza de las texturas, sería interesante incluir efectos de niebla, que podrían dar una mayor profundidad a los colores con los que están texturizados los modelos 3D.

Debería ser una niebla ligera, que enriquezca el aspecto visual del videojuego, pero no entorpezca la visibilidad del jugador. Sería interesante la posibilidad de poder recrear otros efectos ambientales como:

Lluvia Es posible conseguir un efecto de lluvia a través de sistemas de partículas. También es posible añadir un Pixel Shader que refleja el efecto de gotas en el cristal de la cámara, lo que dará un efecto aún mejor.

Noche Una iluminación mucho más atenuada, de un color diferente que simule la luz de la luna, crearía una agradable variedad en los escenarios, proporcionando un toque distinto.

Movimiento de cámara Actualmente, la cámara está fija, sin que el usuario pueda moverla de ninguna manera. Aunque en este tipo de juegos, es bastante común que la cámara esté fija, dar al usuario la posibilidad de cambiar la posición y la orientación de la misma, puede hacer que el usuario aprecie mejor los detalles de los modelos, mejorando su experiencia de juego.

Multijugador

Aunque los juegos de tipo TOWER DEFENSE han sido juegos generalmente en los que no se permite la opción de multijugador, están empezando a aparecer juegos en los que se permite la opción multijugador, como Clash Royale (Supercell). En la figura 6.9



Figura 6.7: Niebla en Ogre3D.

La opción multijugador que se implementaría para TOWER3D sería muy sencilla:

- Cada jugador vería un mapa completo en el que figuraría su zona y la zona del jugador rival.
- El mapa sería atravesado de extremo a extremo por un camino.
- Cada jugador sólo puede construir torres en su mitad del camino.
- Cada jugador sufrirá oleadas de enemigos, que irán siendo cada vez más difíciles de derrotar.
- Cada jugador, además de comprar torres, podrá comprar tropas para atacar al rival, a un coste mucho más barato que las torres.
- Cada tropa comprada iniciará inmediatamente el camino hacia el mapa del jugador rival.
- Los jugadores deberán intentar sincronizar sus compras de tropas junto con las enviadas por el propio juego contra el enemigo, para maximizar sus efectos.
- El primer jugador que se quede sin vidas pierde.

El juego no debe perder su concepto de ser un juego casual, de tal manera que las oleadas enviadas por el juego deben ir subiendo la dificultad, de tal manera que una partida no exceda de tiempo de juego en 15 minutos.

Se valorará la posibilidad de construir torres que aporten mejoras a las tropas, o directa-

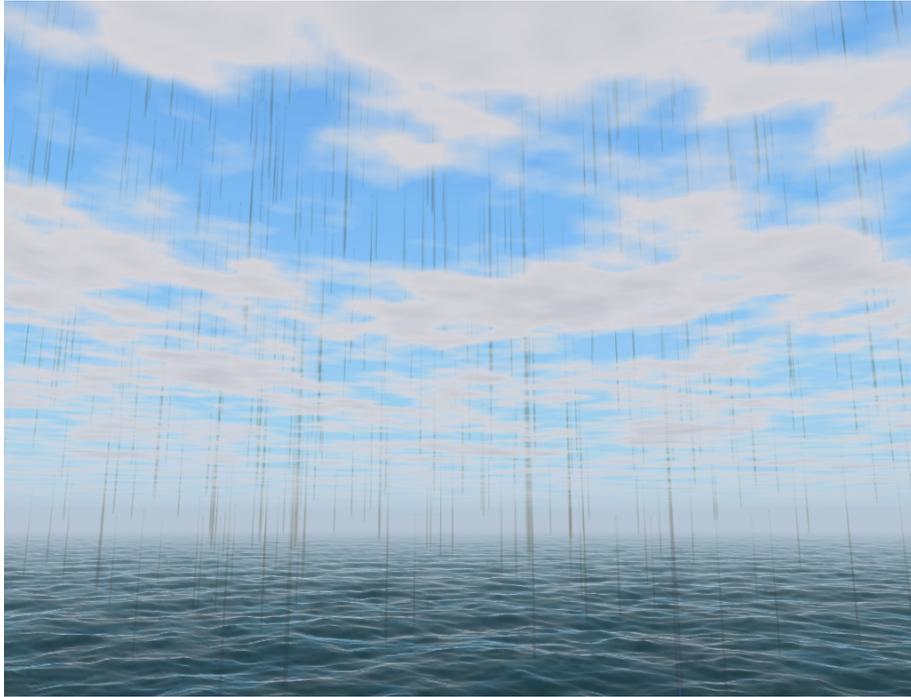


Figura 6.8: Ejemplo de Lluvia en Ogre3D mediante partículas.

mente nuevos tipos de tropas. Dichas torres deberían tener una capacidad defensiva muy leve, o directamente no tener ninguna, para evitar que un tipo de torres se muy bueno ofensiva o defensivamente.

La mejor manera para realizar la implementación del jugador sería utilizar Google Play Games Services.

Google Play Games Services es una API que engloba muchos servicios online ofrecidos por Google, amén de incluir funcionalidades típicas para juego en red que facilitan en gran manera su implementación.

Actualmente, la API para C++ de Google Play Games Services incluye las siguientes funcionalidades:

Autorización Permite establecer un control de acceso y membresía en el juego. Este requisito es imprescindible, lo que obligará al usuario a crearse una cuenta para poder utilizar el juego.

Logros Los logros son un sistema para reforzar el nivel de adherencia del usuario al juego. Se ofrecen pequeñas recompensas al jugador por realizar pequeños hitos. Estos hitos tienen la virtud de conseguir que el jugador siga jugando a un juego, probablemente descubriendo aspectos del mismo que había ignorado, para conseguir estas pequeñas recompensas. Muchos juegos actuales, como Pokemon Go (Niantic) implementan un sistema de logros (ver figura 6.10).

En TOWER3D se podrían establecer diferentes logros, por ejemplo:



Figura 6.9: Captura de pantalla de Clash Royale (desarrollado por Supercell).



Figura 6.10: Logros de Pokemon GO (desarrollado por Niantic), representados por medallas

- Logros por número de victorias.
- Logros por victorias rápidas.
- Logros por cantidad de torres construidas.
- Logros por número de tropas enviadas al combate.
- Logros por llegar a mejorar una torre a un nivel determinado.

Tableros de clasificación Estos tableros permiten establecer una clasificación, en base diferentes criterios, que permiten al jugador ubicarse dentro de la comunidad de jugadores del juego (ver figura 6.11). Este tipo de tableros fomentan mucho la competitividad del juego. En TOWER3D sería muy interesante establecer una tablero ordenado por victorias, que permitiera a los usuarios retar a los jugadores que estén cerca en el tablero. También, en el caso de una comunidad de jugadores muy amplia, se podrían establecer varias ligas, cada una con su propio tablero.



Figura 6.11: Tablero de clasificación de Clash Of Clans (SuperCell).

Multijugador basado en turnos Google Play Games Services ofrece métodos en su API que permiten de forma sencilla establecer un sistema multijugador basado en turnos.

En el caso de TOWER3D, no parece tener sentido utilizar un sistema basado en turnos.

Multijugador en tiempo real De manera similar al anterior, Google Play Games Services ofrece métodos en su API para facilitar la implementación de un sistema multijugador en tiempo real.

Este sería el API principal del que haría uso TOWER3D.

Eventos y Búsquedas Los eventos son un sistema para recoger información del juego, de tal manera que pueda ser utilizado para mejorarlo. Se pueden asignar diferentes disparadores a los eventos, como por ejemplo que se supere un determinado nivel, o que se adquiera un objeto concreto.

En el caso las búsquedas, son sucesos especiales en los que se activa determinado funcionalidad del juego. Dichos sucesos especiales están activos solamente por tiempo limitado. Normalmente la funcionalidad activada no es accesible fuera de las búsquedas.

En TOWER3D, se podrían utilizar eventos para saber que torres son las más usadas, o si hay alguna que prácticamente no se use, para poder equilibrar las torres y que todas sean apetecibles al jugador.

En el caso de las búsquedas, se podrían instaurar eventos temporales en los que estuviera disponible un nuevo tipo de torre, o de enemigo.

Guardado de partidas Este servicio permite el guardado de partidas en la nube, de tal manera que no ocupan espacios en el dispositivo del usuario.

En TOWER3D este servicio sería muy interesante, aunque no para modo multijugador.

Conexiones cercanas Permite al jugador establecer contacto y jugar con otros jugadores que se encuentren cerca físicamente. Esto añade un punto social a cualquier juego, en el que el feedback del usuario es mayor en partidas multijugador, por poder contemplar las facciones y reacciones del rival a sus acciones.

Est

TOWER3D podría hacer uso de esta funcionalidad, aunque requeriría que tuviera una comunidad de usuarios bastante amplia, para que no fuera muy difícil encontrar otro jugador.

Estadísticas del jugador Con esta funcionalidad se pueden almacenar estadística del jugador. Es complementaria totalmente a la funcionalidad de tableros de clasificación, aunque puede ofrecer información más pormenorizada, como por ejemplo:

- Cual es el tipo de torre más usada.
- Contra que jugador has ganado más veces y contra cual has perdido más.
- Victoria más rápida del jugador.

ANEXOS

Anexo A

Guia rápida de uso

En este apartado se explica cómo se debe jugar a TOWER3D.

A.0.1 Pantalla de cargando

Cuando se inicia el juego, dependiendo del dispositivo, es posible que se muestre la pantalla que puede verse en la figura A.0.1, mientras el juego termina de cargar.

Esta pantalla normalmente sólo se visualiza en dispositivos Android, ya en PC carga demasiado rápido y no se visualiza. Dependiendo del dispositivo, es normal que esta pantalla pueda permanecer visible entre 2 y 30 segundos.

A.0.2 Pantalla de Menú

En la figura A.0.2 se observa la pantalla del menú principal, con tres grandes botones, para facilitar su uso en pantalla táctil.

- El botón con forma de aspa cierra el juego, volviendo al punto de partida.
- El botón con forma de triángulo empieza una nueva partida.
- El botón con forma de listado, muestra las puntuaciones más altas.

A.0.3 Pantalla Principal de juego

La pantalla principal del juego tiene este aspecto (figura A.0.3):

En la parte superior de la pantalla aparece la información del estado de la partida. Se puede ver la siguiente información:

- WAVE. El primer número es la oleada actual. El siguiente es el total de oleadas que hay que superar para ganar el juego. El último son los segundos que faltan antes de que empiece la siguiente oleada.
- LIVES. El número de vidas que le quedan al jugador. Si llega a 0, el jugador pierde.
- MONEY. El dinero que le queda al jugador. Empieza con un valor de 100.
- SCORE. Los puntos que lleva el jugador. Cada enemigo abatido da puntos al jugador.

Como fondo de la pantalla se ve el mapa, de color verde las zonas en las que se puede construir torres, y de fondo marrón el camino, por el que avanzarán los enemigos. En la



Figura A.1: Pantalla de bienvenida al juego

captura de pantalla, podemos observar enemigos de dos tipos que avanzan hacia el final del camino, y tres torres, de dos tipos diferentes, que intenta frenarlos.

En la parte Inferior se pueden ver los dos tipos de torres que el jugador tiene disponible para construir, con su respectivo precio debajo.

Construcción de una torre. Para Construir una torre, el jugador solamente debe hacer clic (o marcar con el dedo en dispositivos táctiles) en alguno de los iconos de torre disponibles en la parte inferior izquierda y sin soltar (o levantar el dedo), arrastrarlo a una ubicación válida del mapa. El jugador sabrá que es una ubicación valida porque el círculo que muestra el rango de alcance de la torre estará en verde (ver figura A.0.3. En caso contrario estará en rojo. Si el jugador suelta el botón del ratón o levanta el dedo en una ubicación no válida, la torre no se construirá (ver figura A.0.3). Ejemplo de construcción d.e torre con ubicación correcta:

Si el usuario intenta construir una torre o mejorarla, pero no tiene dinero suficiente, le aparecerá un mensaje informando de ello (ver figura A.0.3)

Mejora de las torres. El jugador puede mejorar una torre que ya ha construido, seleccionándola con el cursor (o con el dedo). Al hacerlo, aparecerá un panel con las características de dicha torre. Dichas características son:

level Nivel de la torre. Todas empiezan en nivel 0 y pueden ser mejoradas.

range Alcance de la torre. Cuanto más alto sea el valor, más lejos podrá la torre detectar enemigos y atacarles.

damage La cantidad de puntos de daño que causa un proyectil lanzado por la torre.



Figura A.2: Menú Principal de Tower3D

splash Área de efecto del proyectil. Cuando un proyectil impacta, no solo realiza daño al enemigo al que apuntaba, si no a todos los enemigos que se encuentren lo suficientemente cerca. Cuanto más alto sea el valor del 'splash', el proyectil puede hacer daño a objetivos más lejanos del punto de impacto.

Además, si esa torre no ha sido mejorada a su nivel máximo, aparecerá un botón para mejorarla A.0.3.

Fin de Partida. Tanto si el jugador sobrevive a todas las oleadas como si pierde, le aparecerá un panel (ver figura A.0.3 indicando si ha ganado o perdido, los puntos que llevaba hasta el momento, y podrá introducir su nombre para registrarlo en los records.



Figura A.3: Pantalla de juego de TOWER3D. El jugador todavía no ha hecho nada.



Figura A.4: Ubicación Válida, se puede observar el círculo verde.



Figura A.5: Ubicación Inválida, se puede observar el círculo rojo.

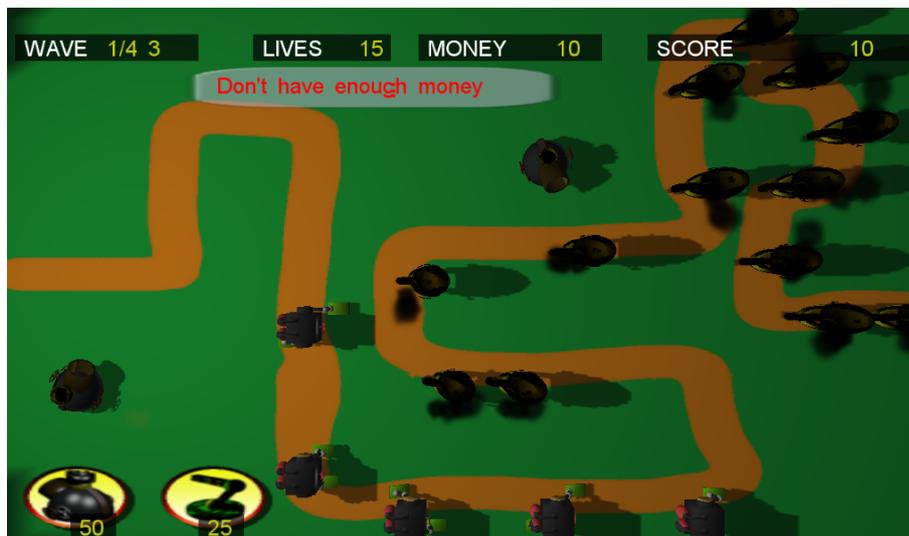


Figura A.6: Mensaje advirtiendo al jugador que falta dinero para construir o mejorar una torre.



Figura A.7: Panel de Mejora. El botón con la flecha hacia arriba permite mejorar la torre.

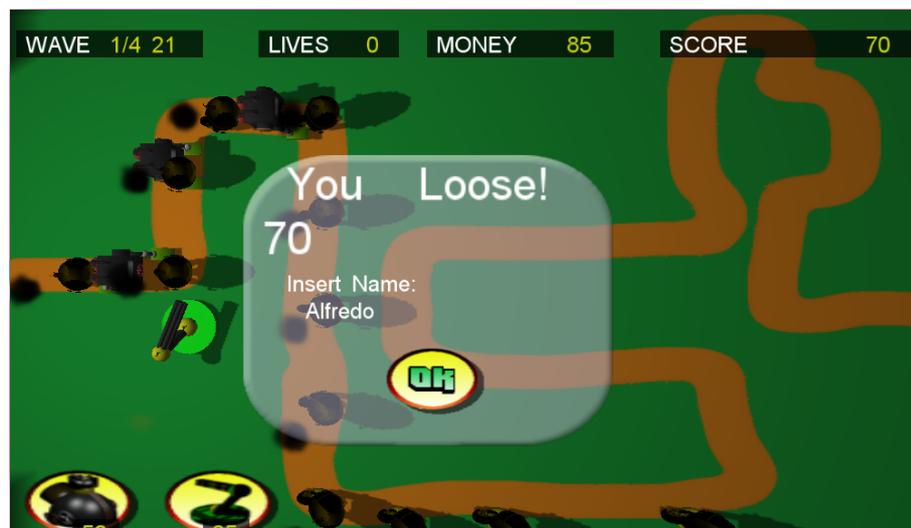


Figura A.8: El jugador se ha quedado sin vidas y ha perdido.

Anexo B

Estructura Física del proyecto

Debido a que Android requiere para sus aplicaciones una configuración determinada, es necesario adaptar la aplicación a la siguiente estructura de directorios.

/app Raíz del proyecto. Debe incluir el archivo `AndroidManifest.xml`, en el que se especifican varias opciones de la aplicación.

/src En esta carpeta se almacenan las clases Java de la aplicación.

/jni En esta carpeta se incluye el código nativo en C o C++. Es necesario incluir dos archivos llamados `Application.mk` y `Android.mk`, en los que se incluyen instrucciones sobre la compilación y el linkado de la aplicación para Android.

/include En esta carpeta se incluyen los archivos de cabecera de todas las clases C++ definidas.

/src En esta carpeta se incluye el código fuente de las clases de C++ que incluye el proyecto.

/assets En esta carpeta se incluyen todos los recursos necesarios para la aplicación. Archivos de configuración, gráficos, sonido etc...

/fonts Incluye el archivo `arial.ttf` y el archivo `Tower3DFonts.fontdef`, en el que se define el tamaño con el que se va a usar la fuente, en un dialecto específico para la definición de fuentes de Ogre3D.

/maps Incluye los archivos de definición de mapas, tanto el modelo 3D, como las texturas y los grafos asociados al mismo.

/material Incluye material genérico, como el usado para sombrear los paneles del HUD

/menu Incluye las imágenes de todos los botones del menú y su imagen de fondo.

/overlays Incluye la definición de todos los overlays utilizados en el juego. Incluye también algunos recursos utilizados por dichos overlays, como los botones de construcción de torres.

/soldiers Incluye modelos 3D y texturas de los enemigos que aparecen en el mapa.

sounds Aquí se encuentran todos los sonidos que se utilizan en el juego.

tiles Aquí se encuentran las texturas utilizadas con el planteamiento anterior de la construcción del mapa por tiles.

towers Modelos 3D y texturas de las torres. También la textura con la que dibuja el rango de alcance.

Bibliografía

- [1] Sumeet Arora. *WebGL Game Development*. Packt Publishing, 2014.
- [2] Richard Bartle, Chris Bateman, et al. *Beyond Game Design: Nine Steps Towards Creating Better Videogames*. Course Technology, 2009.
- [3] Bob Bates. *Game Engine Architecture*. Thomson Couse Technology, 2004.
- [4] Blender. Render baking. <https://www.blender.org/manual/render/workflows/bake.html>, 2014. Accedido: 2016-08-09.
- [5] Boost. Boost background information. <http://www.boost.org/users/index.html/>, 2016. Accedido: 2016-06-07.
- [6] Boost. Boost graph library: Graph theory review. http://www.boost.org/doc/libs/1_57_0/libs/graph/doc/graph_theory_review.html#sec:bfs-algorithm, 2016. Accedido: 2016-02-07.
- [7] GamePlay3D Community. Gameplay3d - free 2d/3d game framework. <http://www.gameplay3d.io>, 2015. Accedido: 2016-08-15.
- [8] Ogre3D Community. Android and sdl. <http://www.ogre3d.org/forums/viewtopic.php?f=5&t=80619&p=504953&hilit=android+current+context#p504953>, 2014. Accedido: 2016-05-15.
- [9] Ogre3D Community. Common mistakes. <http://www.ogre3d.org/tikiwiki/Common+Mistakes>, 2016. Accedido: 2016-06-25.
- [10] Ogre3D Users Community. Cmake quick start guide - android. <http://www.ogre3d.org/tikiwiki/tiki-index.php?page=CMake+Quick+Start+Guide&tikiversion=Android>, 2015. Accedido: 2015-11-05.
- [11] Asociacion Española de empresas productoras y desarrolladores de videojuegos y Software de Entretenimiento. Libro blanco del desarrollo español de videojuegos 2015. <http://www.dev.org.es/images/stories/docs/libro%20blanco%20videojuegos%202015%20final%20low.pdf>, 2015. Accedido: 2016-05-12.
- [12] Gordon Fisher. *Blender 3D Basics*. Packt Publishing, 2012.

- [13] Free Software Foundation. Lista de licencias con comentarios. <https://www.gnu.org/licenses/license-list.es.html#SoftwareLicenses>, 2015. Accedido: 2016-08-03.
- [14] Epic Games. About unreal engine 4. <https://www.unrealengine.com/unreal-engine-4>, 2016. Accedido: 2016-06-27.
- [15] Epic Games. Unreal engine 4 release and royalty tracking guidelines. <https://www.unrealengine.com/release>, 2016. Accedido: 2016-06-27.
- [16] Runic Games. Torchlight. <http://www.torchlight.com>, 2009. Accedido: 2016-08-08.
- [17] Google. <uses-sdk>| android development. <https://developer.android.com/guide/topics/manifest/uses-sdk-element.html>, 2016. Accedido: 2015-11-15.
- [18] Jason Gregory. *Game Engine Architecture*. CRC Press, 2015.
- [19] GraphML Working Group. the graphml file format. <http://graphml.graphdrawing.org>, 2016. Accedido: 2015-12-29.
- [20] Clinton Keith. *Agile game development with Scrum*. Addison-Wesley, 2010.
- [21] Felix Kerger. *Ogre 3D 1.7 Beginner's Guide : Create Real Time 3D Applications Using OGRE 3D from Scratch*. Packt Publishing, 2010.
- [22] Henrik Kniberg. *Scrum and XP from the Trenches*. Lulu, 2007.
- [23] Sam Lainga. Simple directmedia layer. <https://www.libsdl.org/index.php>, 2016. Accedido: 2016-02-05.
- [24] Olivier Lecarne and Karine Delvare. *The book of GIMP*. No Starch Press, 2013.
- [25] Noel Llopis. *C++ for game programmers*. C++ for Game Programers, 2003.
- [26] Luis Picurelli. Los 12 modelos de negocio de aplicaciones. <https://www.yeeply.com/blog/12-modelos-de-negocio-de-aplicaciones>, 2013. Accedido: 2016-08-17.
- [27] Richard Stallman. *GNU Emacs Manual*. GNU Press, 2002.
- [28] Richard Stallman. ¿que es el software libre? <https://www.gnu.org/philosophy/free-sw.html>, 2015. Accedido: 2016-07-12.
- [29] Unity Technologies. Acerca de unity. <http://unity3d.com/es>, 2016. Accedido: 2016-06-25.

-
- [30] David Vallejo and Cleto Martin. *Desarrollo de Videojuegos: Un Enfoque Práctico*. David Vallejo, Carlos Gonzalez y David Villa, 2015.
- [31] David Vallejo and Cleto Martin. *Desarrollo de Videojuegos: Un Enfoque Práctico*. Carlos Gonzalez, Javier Albusac, Cesar Mora and Santiago Sanchez, 2015.
- [32] Judd Vinet and Aaron Griffin. About openbox. <https://www.archlinux.org/about/>, 2016. Accedido: 2016-02-05.
- [33] Wikipedia. List of game engines. https://en.wikipedia.org/wiki/List_of_game_engines, 2016. Accedido: 2016-07-03.

Este documento fue editado y tipografiado con \LaTeX
empleando la clase **arco-pfc** que se puede encontrar en:
https://bitbucket.org/arco_group/arco-pfc

[Respetar esta atribución al autor]