

**5693: WOXFARE, Un videojuego de supervivencia.
Diseño e implementación del Gameplay**

Memoria del Proyecto Final de Carrera
de Ingeniería en Informática realizado
por Fernando Perera Megia y dirigido por
Enric Martí Gòdia Bellaterra,
Junio de 2014



El abajo firmante, Enric Martí Gòdia
profesor de la Escuela de Ingeniería de la UAB,

CERTIFICA:

Que el trabajo al que corresponde esta memoria ha sido realizado bajo su dirección
por Fernando Perera Megia

Y para que conste firma la presente.

Firmado: Enric Martí Gòdia

Bellaterra, Junio de 2014

AGRADECIMIENTOS

Me gustaría dedicar este proyecto a mi padre, Francisco Perera, por el apoyo y la curiosidad que me inculcó durante los años de carrera. Si esa curiosidad nunca me hubiera embarcado en aventuras como esta.

A mi compañero de proyecto, y amigo, Alejandro Vázquez, por la paciencia y ayuda, además de agradecerle haber sabido tratar conmigo y motivarme día a día.

RESUMEN

Woxfare es un videojuego de tipo supervivencia desarrollado por 2 personas como Proyecto Final de Carrera de Ingeniería Informática. Este videojuego se compone principalmente de un editor de escenarios y un gameplay.

En esta memoria se explican las partes pertenecientes al diseño e implementación del juego, de la interfaz de usuario, de los enemigos, de las armas, de los objetos y la carga de todos estos. Además incluye el desarrollo de las partes visuales y la inclusión de sonidos.

El resultado obtenido es un juego dónde se pueden cargar escenarios diseñados por el propio usuario a través del Editor del videojuego y jugarlos. Se ha buscado jugabilidad y sencillez, al mismo tiempo que resulte divertido. Este videojuego ha sido validado por una serie de personas que nos han dado su opinión.

RESUM

Woxfare és un videojoc de tipus supervivencia desenvolupat per 2 persones com a Projecte Final de Carrera de Enginyeria Informàtica. Aquest videojoc es compon principalment de un editor d'escenaris i un gameplay.

En aquesta memoria s'expliquen les parts pertanyent al disseny i implementació del videojoc, de l'interfàcia d'usuari, dels enemics, de les armes, dels objectes i de carregar tot això. A més, inclou el desenvolupament de les parts visuals i la inclusió de sons.

El resultat obtingut és un videojoc on es poden carregar escenaris dissenyats pel propi usuari a través del Editor del videojoc i jugar-los. S'ha buscat jugabilitat i sencillesa, al mateix temps que resulti divertit. Aquest videojoc ha sigut validat per una sèrie de persones que ens han donat la seva opinió.

ABSTRACT

Woxfare, is a survival videogame developed by 2 people as a Degree Project in Computer Engineering. This videogame is composed with an scene editor and the gameplay.

In this report is exposed the task of the design and development of the game, the user interface, de enemies, the weapons, the objects and the load of all of them. Also it includes the development of the art and the sounds.

As a result of the project it has been developed a videogame where the user can make their own scenes and play them into the game. This videogame has been checked by a sort of people giving their opinions.

ÍNDICE

Agradecimientos	V
Resumen	VII
Resum	VII
Abstract.....	VII
Índice	IX
1 Introducción	1
1.1 Objetivos.....	2
1.2 Woxfare, The Game Concept.....	2
1.2.1 ¿Por qué Woxfare?	3
1.2.2 Motivos y finalidades del juego	3
1.2.3 Mecánica de juego	3
1.2.4 Objetos de Escena.....	4
1.2.5 Armas	4
1.2.6 Controles.....	5
1.2.7 Partidas	6
1.3 Elección del motor gráfico	6
1.3.1 Unreal Engine (UDK).....	7
1.3.2 Ogre 3D Engine (Object - Oriented Rendering Engine)	8
1.3.3 Unity Engine	9
1.3.4 Comparativa final.....	10
1.4 Elección del motor de físicas	11
1.4.1 Bullet Physics	11
1.4.2 NVIDIA PhysX	11
1.4.3 Comparativa Final	12
1.5 Tecnologías usadas	13
1.6 Objetivos individuales de este proyecto.....	14
1.7 Planificación del trabajo	14
2 Desarrollo	16
2.1 Woxfare: Gameplay	16
2.1.1 Gestor de Archivos.....	18
2.1.2 Jugador.....	20
2.1.3 Gestor Enemigos	21
2.1.4 Gestor de Objetos	23
2.1.5 Gestor de Ítems.....	23

2.1.6	Gestor de Armas	24
2.1.7	Lógica del Juego	24
2.1.8	Interfaz Gráfica de Usuario (GUI).....	25
2.1.9	Cámara	29
2.1.10	Módulo de Físicas.....	30
2.1.11	Módulo de Sonido	31
2.2	Contribución en Woxfare: editor	31
2.2.1	Ogre	31
2.2.2	Optimización de escenario.....	32
3	Resultados	35
3.1	Resultados de Woxfare.....	35
3.2	Resultados del Modelado	36
3.3	Resultados de GUI.....	37
3.4	Resultados de la integración.....	41
3.5	Validación de Usuarios	41
3.6	Futuro del proyecto	42
4	Conclusiones	43
5	Bibliografía y Referencias	45

1 INTRODUCCIÓN

En esta memoria se expone el diseño y el desarrollo de un proyecto de videojuegos formado por dos personas: Alejandro Vázquez y Fernando Perera. El proyecto está formado por una parte común, donde se diseña el concepto de juego y se toman las decisiones iniciales del proyecto, y una parte individual, donde cada uno ha realizado un conjunto de tareas. Este capítulo introductorio muestra un enfoque general de los propósitos, motivaciones y objetivos del proyecto, así como la distribución de las tareas, los medios utilizados y los contenidos del proyecto.

La primera pregunta que se nos plantea al escoger un proyecto de este tipo es tener en cuenta los roles que cada uno deberá asumir en su realización. Ambos queríamos realizar un proyecto relacionado con videojuegos, pero no disponer de habilidades artísticas ni el tiempo necesario para completar todos los aspectos de un videojuego nos quisimos centrar en las mecánicas y la forma de crear el videojuego y su editor antes que en el aspecto artístico.

Según esta visión, intentamos minimizar aspectos de la historia y el arte en el juego, dándole un carácter editable y personal en las partidas que el jugador pueda crear y jugar.

Nos hemos inspirado en varios juegos 2D de supervivencia, uno de ellos es BoxHead (Figura 1.1) y otro es Age of Zombies (Figura 1.2). Nos gusta mucho la mecánica de juego de ambos y hemos querido llevar el mundo de 2D a uno de 3D.

Nuestra propuesta no se queda ahí si no que pretende ser un juego comunitario que motive al usuario tanto a crear escenarios, como a compartirlos, a jugarlos y a puntuarlos. El concepto de comunitario no pretende sólo permitir compartir los diseños y puntuaciones, sino que la propia herramienta esté desarrollada por la comunidad en las futuras modificaciones. No obstante, el proyecto expuesto en esta memoria pretende ser una base de trabajo para implementar las mecánicas principales del videojuego de tipo supervivencia y de la edición de escenarios para este videojuego.

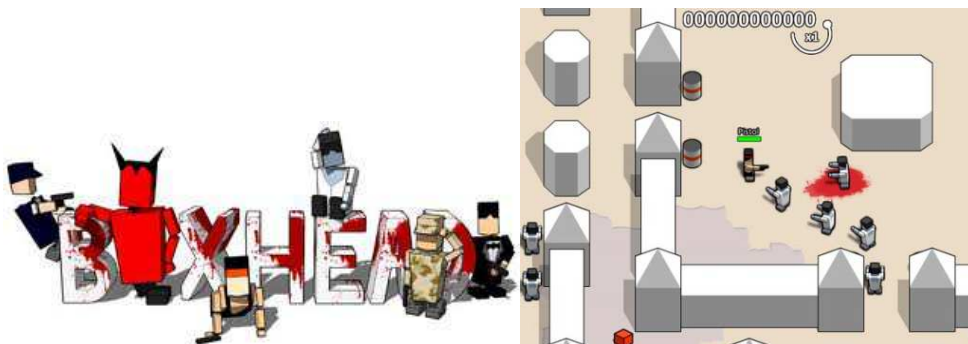


Figura 1.1: Pantalla de Inicio y Capturas Ingame de BoxHead



Figura 1.2: Pantalla de Inicio y Capturas Ingame de Age of Zombies

1.1 OBJETIVOS

El objetivo principal que pretendemos es el diseño e implementación de un videojuego de tipo supervivencia (Woxfare) y un editor de escenarios y partidas (Omega). Este objetivo se puede separar en dos proyectos, pero intentando la reutilización de partes comunes. Durante todo el desarrollo del videojuego se prevé sincronizar varias veces los trabajos y mantener contacto día a día. Los objetivos a cumplir por el proyecto los siguientes:

- a) **Diseño del concepto de juego.** Existen otros juegos en el mercado muy parecidos a éste, por lo que no pretendemos innovar en este sentido, sino que queremos diseñar, implementar y hacer un proyecto que sea nuestro de principio a fin.
- b) **Diseño del editor.** Muchos usuarios de videojuegos, con un cierto nivel de experiencia, modifican los juegos para añadir personajes, texturas de alta calidad, e incluso en algunos juegos cómo el Counter Strike [1]. Hay una gran comunidad que realizan mapas, crean servidores, etc. Por ello se ha pensado que el juego que queremos realizar vaya acompañado de un editor, que haga posible de forma fácil todas estas modificaciones y no haga falta ser un experto.
- c) **Identificación de partes reutilizables.** El proyecto es divisible en tareas únicas e independientes. Para ello se deberán de realizar implementaciones que sean utilizables tanto en el juego como en el editor. Un claro ejemplo es la visualización 3D.
- d) **Dividir las tareas de forma equitativa en 2 proyectos.** Se pretende que el trabajo sea el mismo para ambos proyectos, e igualmente interesantes para los dos miembros.
- e) **Solucionar los problemas.** Resolver los imprevistos y errores que se puedan producir en el desarrollo del proyecto.
- f) **Testear en un entorno real el juego y el editor.** Implementaremos un juego y un editor que deberán ser validados en un entorno real, por usuarios.
 - a. Queremos crear un editor fácil de usar, divertido e intuitivo, proporcionando comandos intuitivos, facilidad para localizar los menús y diálogos.
 - b. Queremos crear un juego divertido y atractivo. Que permita al usuario interactuar de forma sencilla con el entorno.
- g) **Trabajo en equipo.** El trabajo en equipo es algo muy importante en el mundo laboral y que hemos querido que forme parte de nuestro proyecto. Por ello hemos queremos dividir las tareas de forma que se necesite la colaboración de ambos compañeros.

1.2 WOXFARE, THE GAME CONCEPT

En cualquier desarrollo de un videojuego debe de haber un documento llamado 'Gameconcept' donde se recopila toda la información y mecánica del juego que se pretende realizar. Este documento marca la base y los objetivos para llevar a cabo el desarrollo del juego así como los personajes, enemigos, arte del juego, mecánica, objetivos principales, etc.

En este apartado se exponen esas ideas:

- ¿Por qué Woxfare?

- Motivos y finalidades del juego
- Mecánica del juego
- Personajes y Enemigos
- Objetos de escena
- Las armas
- Los controles
- ¿Cómo funcionan las partidas?

1.2.1 ¿POR QUÉ WOXFARE?

El nombre proviene de un juego de palabras. El arte del juego son básicamente cajas (box en inglés) y el juego va a tratar de una guerrilla de supervivencia (warfare). Se unieron ambas palabras y nació el nombre de Woxfare.

1.2.2 MOTIVOS Y FINALIDADES DEL JUEGO

- **Divertido:** Una clara definición de juego es que sea divertido y entretenido. Este es nuestra meta más importante.
- **Fácil:** Un juego que no sea complicado y que para alguien que no tenga mucho rodaje en los videojuegos.
- **Competitivo:** Que quieras superarte partida tras partida para conseguir más puntuación que en partidas anteriores.

1.2.3 MECÁNICA DE JUEGO

Woxfare es un juego en 3D que consiste en sobrevivir a oleadas de enemigos en escenarios cerrados. La estética del juego será cúbica donde los escenarios y los propios personajes estarán hechos con cubos (ver figura 1.3). El personaje tendrá que ir derrotando a los enemigos recogiendo objetos para abastecerse de munición y armas y también poder recuperar vida. Por cada enemigo que derrote irá ganando puntos que se irán guardando en su puntuación.

El punto de vista del personaje y del escenario será en tercera persona, parecido a los juegos anteriormente mencionados, para de esta forma tener una visión más amplia de donde nos vienen los enemigos.

Personajes y Enemigos

La inspiración de los personajes y los enemigos ha sido obtenida de la página CubeCraft [2]. Partiendo de estos diseños se han realizado los primeros bocetos que se muestran en la figura 1.3.

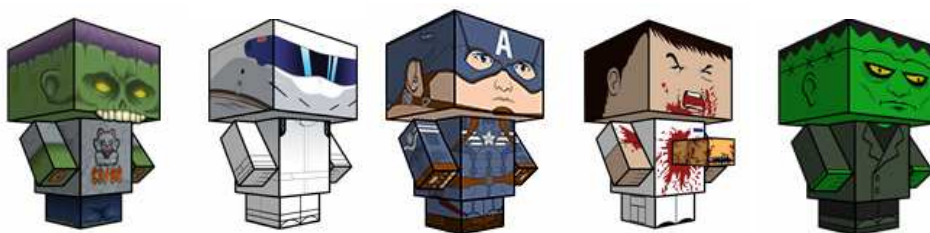


Figura 1.3: Modelos de personajes de CubeCraft

El personaje principal podrá desplazarse por el escenario con total libertad, saltando y cogiendo objetos. Podrá disparar a los enemigos o zafarse de ellos.

Los enemigos podrán ser de diferentes tipos, es decir tendrán asociada una Inteligencia distinta.

- **Enemigos Simples:** Estos enemigos pueden parecer sencillos de matar si se encuentran solos, pero hay que tener mucho cuidado si te encuentras con un grupo de ellos ya que siguen al personaje principal sin descanso.
- **Enemigos Armados:** Al contrario que sus compañeros éstos van armados, dispararán al personaje principal desde una distancia segura.
- **Enemigos Jefes Simples:** Persiguen al jugador igual que los enemigos simples, pero tienen mayor tamaño.
- **Enemigos Jefes Armados:** Al mismo tiempo que siguen al personaje principal van disparándole y tienen mayor tamaño.

1.2.4 OBJETOS DE ESCENA

La composición del escenario será cúbica por lo que el cubo será el elemento básico constructivo para cualquier edificación (ver figura 1.4). Se podrán agrupar diferentes cubos para crear paredes, edificios y todas las construcciones que se quieran.



Figura 1.4: Posibles objetos para generar objetos de escena e ítems

1.2.5 ARMAS

Las armas que el personaje principal serán de diferentes tipos (figura 1.5):

- **Blaster:** Una pistola, con una cadencia de tiro alta pero con gran versatilidad.
- **Escopeta:** Útil contra enemigos cercanos pero mejor no usarla contra aquellos alejados.
- **Ametralladora:** Ráfagas de balas contra los enemigos.

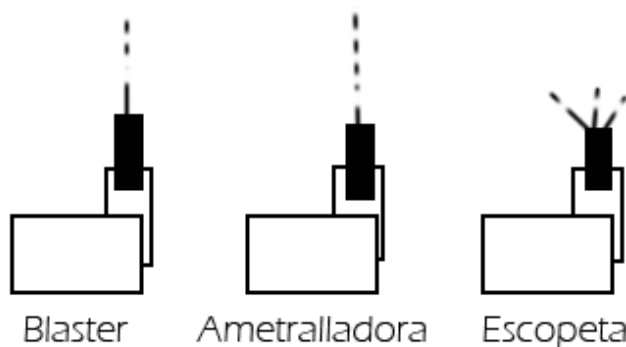


Figura 1.5: Vista superior del personaje y el arma

1.2.6 CONTROLES

El modo de moverse del personaje sigue un poco la línea de los juegos mencionados anteriormente. El personaje podrá moverse hacia delante, atrás y rotar 360 grados. Además, podrá cambiar de arma, saltar y disparar. Los disparos únicamente irán en línea recta, según la orientación de nuestro personaje. No se podrá disparar hacia arriba ni hacia abajo. También añadimos el movimiento lateral y el cambio de sentido rápido. En la figura 1.7 vemos indicado los movimientos posibles del personaje, donde la línea roja representa el movimiento de rotación 360 grados en ambos sentidos y las flechas azules representan hacia donde se puede desplazar.

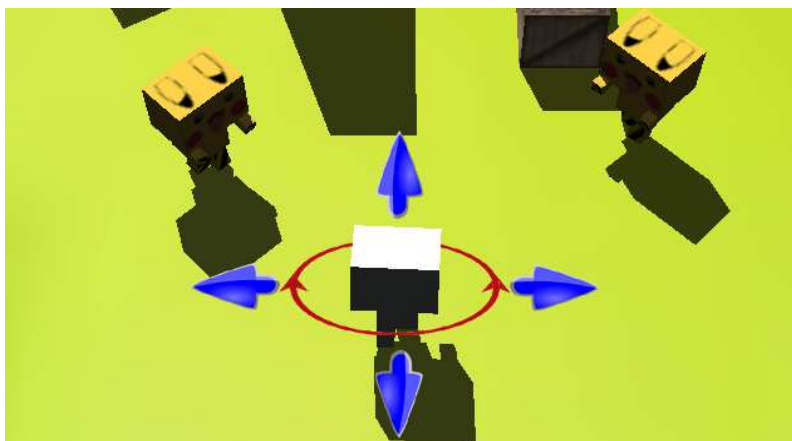
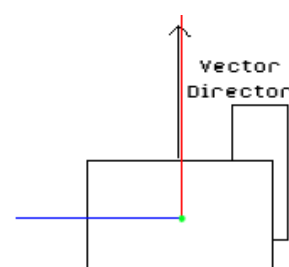


Figura 1.7: Escena del Gameplay con esquema de controles.

En la figura 1.8 se muestra una tabla de los controles por defecto y su acción, con una pequeña descripción, además de un esquema que indica el vector director y los ejes de coordenadas, marcando de azul el eje X, de rojo el Z y de verde el Y.

Acción	Descripción
Mover hacia delante	El personaje mueve en la dirección que marca el vector de orientación.
Mover hacia atrás	El personaje se mueve en dirección opuesta al vector de orientación.
Mover hacia la derecha	El personaje se mueve en dirección perpendicular al vector de orientación en el plano XZ, hacia la izquierda
Mover hacia la izquierda	El personaje se mueve en dirección perpendicular al vector de orientación en el plano XZ, hacia la derecha.
Giro a la izquierda	El personaje rota sobre si mismo mientras se mantiene pulsado el botón, de forma gradual, hacia la izquierda.
Giro a la derecha	El personaje rota sobre si mismo mientras se mantiene pulsado el botón, de forma gradual, hacia la



	derecha.
Giro 180 grados	El personaje cambia el sentido, girando 180 grados el vector de orientación.
Disparar	El personaje dispara en la dirección que marca el vector de orientación.
Saltar	El personaje saltará. Permite que el personaje se mueva hacia donde quiera durante el salto.
Cambiar de arma arriba	Cambia de arma a la siguiente disponible de la lista.
Cambiar de arma abajo	Cambia de arma a la anterior disponible de la lista.
Pausa	Detiene el juego y activa el menú de pausa.

Figura 1.8: Tabla de acciones.

1.2.7 PARTIDAS

Cada partida estará dividida en diferentes rondas. Cada una de estas rondas está compuesta por un tiempo de espera y una etapa de ataque. Durante la etapa de ataque, el jugador deberá acabar con todos los enemigos, acabando así la ronda, y dando pie a la siguiente. El personaje deberá acumular el máximo número de puntos, acumulando rachas de enemigos eliminados. Como ayuda, durante la etapa de ataque y el tiempo de espera caerán algunos objetos del cielo que darán vida, puntos, munición o armas al personaje cuando los coja, y que desaparecerán pasado un tiempo.

El número de rondas, de enemigos y de objetos de cada partida, además del escenario, variará según las especificaciones que el jugador haya hecho durante la fase de edición. Puede haber partidas con rondas finitas, y con rondas infinitas.

Para el caso de partidas con rondas finitas, se marcará un número mínimo y máximo de enemigos y un número de rondas. El juego recalculará cuántos enemigos deben salir por ronda para llegar a la última ronda con el número máximo de enemigos y haber hecho un incremento equitativo.

En el caso de las partidas con rondas infinitas, se marcará un número de enemigos mínimo (para la primera ronda) y un incremento de enemigos, de forma que en cada ronda saldrán más enemigos que en la anterior y el juego finalizará cuando el personaje muera.

1.3 ELECCIÓN DEL MOTOR GRÁFICO

El motor gráfico nos aporta toda la funcionalidad necesaria para poder visualizar objetos 3D por pantalla, desde el renderizado de estos objetos, hasta el posicionamiento de luces y cámaras. Proporciona una interfície entre el uso de tecnologías de más bajo nivel como podría ser OpenGL [3] o DirectX [4] y el programador o diseñador. Además, algunos de los motores estudiados nos aportan toda una interfície para poder desarrollar el videojuego, ayudando, por ejemplo, en la gestión de movimientos del personaje o de eventos que puedan suceder.

La realización de un motor propio nos permitiría realizar una tecnología que podríamos utilizar en los dos proyectos pero que al mismo tiempo nos exigiría una dedicación excesiva de horas, horas suficientes como para abarcar otro proyecto por lo que esta opción fue inicialmente descartada.

La elección de un motor gráfico apropiado para la realización del editor y el juego ha sido difícil por la gran disponibilidad de motores en el mercado y la diversificación de filosofías entre ellos. Por ello se ha realizado un estudio para ver las ventajas y desventajas de cada uno.

En la figura 1.9 se exponen brevemente algunos motores gráficos que hay disponibles en el mercado conjuntamente con algunas de sus características más relevantes.

			
Nombre	Unreal Engine	Ogre 3D	Unity 3D
Desarrollador	Epic Games	The OGRE Team	Unity Technologies
Página oficial	www.unrealtechnology.com	www.ogre3d.org	www.unity3d.com
Lanzamiento inicial	1998	1999	2005
Última versión estable	Enero de 2012	Noviembre de 2013	Marzo de 2014
Género	Motor de juego	Motor de renderizado	Motor de juego
Programado en	C++	C++	C++
Sistema operativo	Multiplataforma	Multiplataforma	Multiplataforma
Licencia Pro	2500\$	-	1500\$
Precio Publicación	99\$ + 25% Beneficios (superiores a 5000\$)	-	Únicamente licencia pro
Idiomas	Inglés	Inglés	Inglés y español

Figura 1.9: Resumen con información sobre los motores gráficos.

1.3.1 UNREAL ENGINE (UDK)

Motor gráfico de juegos triple A, para PC y consolas creado por la compañía Epic Games. Este motor tiene su origen en un shooter de primera persona llamado Unreal 1988 el cual ha sido la base para juegos como Unreal Tournament, Turok, Bioshock, etc. También se ha utilizado en otros géneros como el rol y juegos de perspectiva en tercera persona. Está escrito en C++, siendo compatible con varias plataformas Windows, Linux y Mac y la mayoría de consolas. Unreal Engine también ofrece varias herramientas adicionales de gran ayuda para diseñadores y artistas.

La última versión de este motor es el Unreal Engine 4 [5], está diseñado para la tecnología:

- DirectX 9 (para las plataformas Windows XP/Vista/7 de 32/64-bit y Xbox 360)
- DirectX 10 (para plataformas Windows Vista/7 32/64-bit)
- OpenGL (para plataformas Linux, Mac OS X de 32/64-bit y PlayStation 3).

Para el desarrollo de videojuegos con Unreal Engine, Epic Games pone a disposición de los usuarios un entorno de desarrollo que es el Unreal Development Kit (UDK) [6] que permite utilizar UnrealScript para programar el Gameplay y que también incorpora intérpretes para GLSL, HLSL y Cg para poder programar Shaders. En la figura 1.10 se muestra un ejemplo de la potencia capacidad de este motor donde se puede apreciar la calidad de las texturas y la iluminación que permite el motor de Unreal.



Figura 1.10: Ejemplo de la capacidad de Unreal

1.3.2 OGRE 3D ENGINE (OBJECT - ORIENTED RENDERING ENGINE)

Un motor de renderizado de escenas 3D que permite al desarrollador incorporar otras librerías de físicas, audio, entrada/salida, etc. El código es abierto por lo que es muy fácil de modificar y utilizar en otros proyectos utilizando lo mínimo indispensable de la API que proporciona. OGRE tiene una comunidad muy activa de usuarios y se han realizado juegos comerciales como Ankh, Torchlight, etc.

La última versión de este motor es el Ogre 1.9 y soporta Windows, Linux y Mac. Los editores de 3D como 3DSMax, Maya, Blender contienen exportadores de objetos para poder ser leídos desde Ogre. A pesar que la curva de aprendizaje de Ogre es elevada, una vez se conoce el funcionamiento interno es relativamente fácil de usar.

Aunque OGRE no es en sí una herramienta de desarrollo de videojuegos, existe a su alrededor una gran comunidad de desarrolladores enfocados al mundo del videojuego y que ayudándose unos a otros y aportando distintas ideas y mecanismos, se consigue que sea una plataforma casi perfecta para el desarrollo de los mismos. Por otra parte en la comunidad muchos desarrolladores han creado librerías con elementos específicos para cubrir las carencias de OGRE respecto al desarrollo de juegos, como son librerías físicas, librerías de diseño de escenarios, librerías de diseño de inteligencia artificial, frameworks estructurados para el desarrollo de juegos y otras muchas más que hacen un poco más fácil el desarrollo de software 3D. En la figura 1.11 podemos ver una captura de una escena renderizada con Ogre donde se aprecian las transparencias y las sombras.



Figura 1.11: Ejemplo de la capacidad de Ogre

1.3.3 UNITY ENGINE

Unity es una herramienta para el desarrollo de videojuegos, que se está convirtiendo en un estándar. Consta de un motor gráfico potente con físicas y shaders para las iluminaciones. El gameplay es programable gracias a su herramienta MonoDevelop que permite realizar scripts en 3 lenguajes diferentes: JavaScript, C# y Boo (derivado de Python). A su vez posee un editor gráfico que permite crear escenarios, luces, skyboxes, etc. ayudando al proceso de desarrollo enormemente.

La potencia gráfica es equiparable a la de Ogre, pero no es tan elevada como la de Unreal Engine, y así mismo el editor tiene menos funcionalidades. Por otra parte el punto fuerte de Unity, es la portabilidad, está hecho de tal forma que el mismo código y recursos son fácilmente exportables de una plataforma (sea Pc o móvil) a otra en sencillos pasos.



Figura 1.12: Ejemplo de la capacidad de Unity

1.3.4 COMPARATIVA FINAL

Todos los motores explicados tienen ventajas y desventajas para la realización del proyecto, por lo que se ha debido de pensar detenidamente cuál es el apropiado para la realización de ambos proyectos tanto el editor Omega como el juego Woxfare. En la figura 1.13 se muestran los puntos fuertes de cada motor. Con esos datos podemos elegir que motor utilizaremos.

Característica	Unreal	Unity	Ogre	Motor propio
Gratuito (si se quiere publicar)	X	✓	✓	✓
Tiempo para empezar	Medio	Medio	Medio	Muy Elevado
Motor de físicas	✓	✓	X	X
Lenguaje de script propio	✓	✓	X	X
Incorporación en editor y juego	X	X	✓	✓
Curva de aprendizaje	Alta	Media	Media	Baja
Documentación y tutoriales	~	Si	Si(comunidad)	No haría falta
Multiplataforma	✓	✓	✓	✓ (Mas tiempo)
Código fuente	X	X	✓	✓

Figura 1.13: Comparativa de motores gráficos

Unreal Engine tiene a favor que es un motor de juegos triple A, con una potencia gráfica abrumadora, con físicas incorporadas y multiplataforma. Por otra parte a pesar de estas magnificas características incorpora muy poca documentación, así que la curva de aprendizaje es muy elevada. En referencia al editor, éste ya incorpora el Kit de Desarrollo Unreal [7], por lo que no lo podríamos utilizar en ambos proyectos y se ha querido que la diversificación sea la mínima posible para reutilizar el máximo posible de código en ambos proyectos.

Unity Engine el motor por excelencia de las compañías independientes, por la licencia gratuita que proporciona la compañía. Muchas compañías lo escogen por su usabilidad, por la gran variedad de documentación y tutoriales disponibles en su comunidad y por la fácil incorporación en cualquier plataforma ya sea móvil, consola u ordenador. Unity sería perfecto para la realización de Woxfare, pero no funcionaría para el editor por lo que se ha descartado.

Finalmente nos quedamos con Ogre, ya que:

- Programar en C++ orientado a objetos es sencillo y a la vez eficiente.
- Es un motor enfocado al renderizado y no a juegos, pero pese a ello existen gran cantidad de librerías enfocadas a este aspecto.
- Facilidad de incorporación de un motor de físicas.
- No dispone de un entorno de desarrollo (IDE), lo que hace que todo deba ser programado, pero las librerías encapsulan las llamadas más básicas evitando así la programación a bajo nivel, por lo que trabaja a un nivel medio y sencillo.
- Al no disponer de IDE, podemos realizar perfectamente el editor para el juego.
- Ogre funciona con librerías en su mayoría gratuitas y abiertas, al igual que todo su código fuente.
- En el caso de distribución y venta del juego en el futuro, así como en la propia adquisición de la herramienta, no habría un coste adicional.

- Es multiplataforma, se puede desarrollar para Windows, MAC o Linux.
- En un futuro puede ser portado a dispositivos móviles ya que tiene soporte para plataformas Android y iOS
- Tiene una gran comunidad de usuarios a su alrededor, por lo que hay gran cantidad de tutoriales, ayudas y los foros están siempre muy activos.

1.4 ELECCIÓN DEL MOTOR DE FÍSICAS

El motor de físicas es el encargado de las simulaciones físicas de la escena. Nos provee de un conjunto de instrucciones para este tipo de simulaciones, permitiendo unir cuerpos físicos con la visualización del motor gráfico. También permite detectar y comprobar colisiones.

De los motores existentes en el mercado, decidimos estudiar Bullet Physics y NVidia PhysX, ya que creemos que son los más relevantes del mercado. En la figura 1.14 se muestran algunas características de cada uno. Seguidamente se comentarán con más detalle cada uno de ellos y nuestra elección.



		
Nombre	Bullet Physics Library	PhysX
Desarrollador	Game Physics Simulation	Nvidia
Página oficial	http://bulletphysics.org/	http://www.nvidia.es/object/nvidia-physx-es.html
Última versión estable	2012	2012
Género	Motor de Físicas	Motor de Físicas
Programado en	C++	C++
Sistema operativo	Multiplataforma	Multiplataforma (GPU Nvidia)
Idiomas	Ingles	Ingles

Figura 1.14: Resumen con información sobre los motores físicos.

1.4.1 BULLET PHYSICS

Bullet Physics [7] es un motor de físicas Open Source con detección de colisiones de objetos. Normalmente se utiliza en juegos y en efectos visuales de películas. Algunos de los videojuegos en los que se han utilizado son Grand Theft Auto IV, Red Dead Redemption, Toy Story 3,... También se ha utilizado en películas como Hancock o Bold entre otras. Bullet nos permite realizar simulaciones de cuerpos rígidos (esfera, cubo, cilindros,...) y blandos (telas, cuerdas, objetos deformables, etc.) con detección de colisiones.

1.4.2 NVIDIA PHYSIX

PhysiX [8] es un motor de físicas desarrollado por Ageia y posteriormente comprado por NVidia. Es uno de los motores de físicas más extendidos en el mercado, de hecho los motores previamente mencionados Unreal Engine y Unity utilizan PhysiX. También Hay muchos videojuegos famosos como Borderlands 2, Castlevania, Mafia II,... Esta librería de físicas utiliza aceleración GPU, pero únicamente con chips de NVidia. Como vemos en la figura 1.15, al aplicar la tecnología PhysX los resultados en humo, fuego y escombros son claramente más vistos.



Figura 1.15: Resumen con información sobre los motores físicos.

1.4.3 COMPARATIVA FINAL

Después de buscar información sobre los 2 motores hacemos un resumen de los detalles de estos a la hora de usarlos en nuestro proyecto en la figura 1.16.

PhysiX, no es una buena opción para desarrollar el proyecto ya que únicamente funciona con GPU's NVidia, esto nos limita al hacer el juego ya que si el usuario no dispone de una tarjeta NVidia no podrá disfrutar al 100% de nuestro juego.

Característica	Bullet Physics	NVidia PhysX	Motor propio
Gratuito (si se quiere publicar)	✓	✓	✓
Tiempo para empezar	Medio	Alto	Muy Elevado
Curva de aprendizaje	Media	Media	Baja
Documentación y tutoriales	Si(comunidad)	Si	No haría falta
Multiplataforma	✓	✓ (solo con GPU's NVidia)	✓ (Mas tiempo)
Código fuente	✓	✗	✓

Figura 1.16: Comparativa de motores de física

Aunque después de ver algunas características de cada uno de los motores, vemos que el motor PhysX ganaríamos en potencia, pero optamos por el motor Bullet Physics por las siguientes razones:

- Un motor de físicas fácil de incorporar a nuestro proyecto.
- Hemos trabajado previamente con Bullet Physics, a nivel básico.
- En el proyecto se busca la compatibilidad. Bullet tiene otro punto a favor ya que no depende del fabricante de la GPU.
- Open Source al igual que Ogre.

La elección más adecuada, teniendo en cuenta que vamos a utilizar Ogre, es Bullet, ya que esta es Open Source al igual que Ogre, hay frameworks y muchos ejemplos de cómo unir ambas librerías y tuvimos una primera toma de contacto en la asignatura de gráficos por computador.

1.5 TECNOLOGÍAS USADAS

Tomadas decisiones decidimos usar el conjunto de tecnologías que se muestran en la figura 1.17. Inicialmente empezamos usando Qt Creator como entorno de desarrollo, pero problemas con el debugger nos hicieron cambiar a Visual Studio.

Editor Omega:

- Qt GUI, usada para la implementación de toda la GUI del editor.
- Ogre3D, para la visualización de los objetos de escena.
- XML, para la comunicación con el juego y para el salvado.
- CSS para mantener la estética con todas las plataformas
- Visual Studio, como entorno de desarrollo.
- Qt Designer, para generar los diálogos.

Juego Woxfare:

- Ogre3D, para la visualización de objetos y GUI.
- FMOD, para la reproducción de sonidos.
- Bullet Physics Library, como librería de físicas.
- Blender, para el diseño de personajes y animaciones.
- Visual Studio, como entorno de desarrollo.
- XML, para la comunicación con el editor.

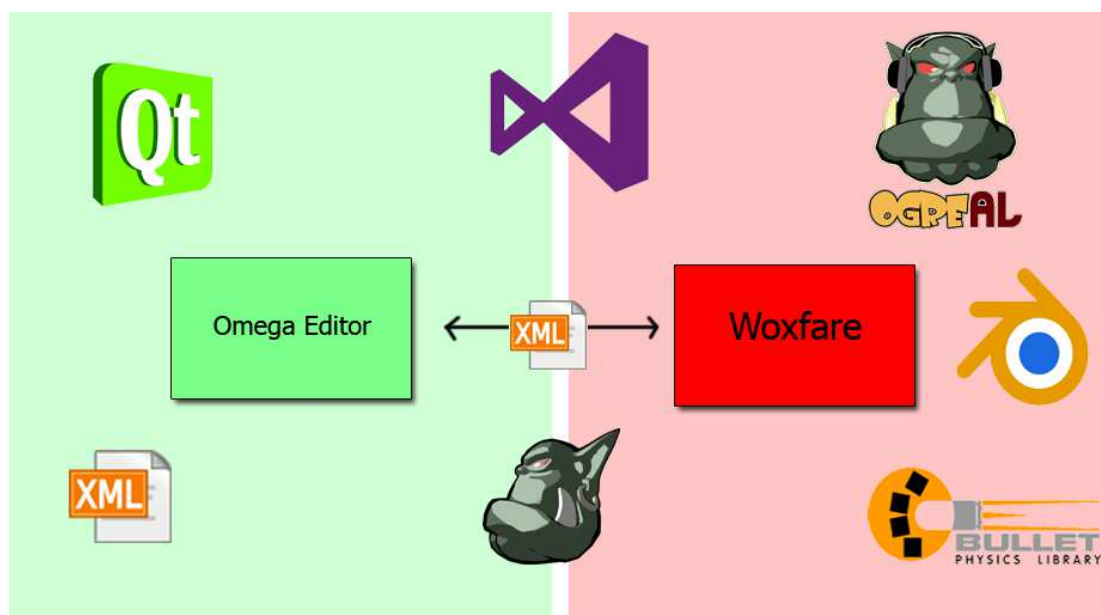


Figura 1.17: Tecnologías usadas en el proyecto

1.6 OBJETIVOS INDIVIDUALES DE ESTE PROYECTO

A continuación mostramos los objetivos individuales propuestos y desarrollados en esta memoria, y que hacen referencia a la parte del diseño y desarrollo del gameplay Woxfare. El principal objetivo es el diseño e implementación del gameplay Woxfare con la siguiente funcionalidad:

- a) Cargar escenarios creados por el editor Omega.
- b) El jugador podrá moverse en todas direcciones en un mapa tridimensional, pudiendo moverse en el eje vertical a través de saltos. También se permitirá al jugador disparar.
- c) Se verá al jugador en tercera persona, con vista desde arriba.
- d) Los enemigos serán controlados por Inteligencia Artificial.
- e) El jugador podrá recoger ítems que dejarán caer los enemigos y le proporcionarán munición, vida u oro.
- f) Diseñar un sistema fácilmente ampliable, a través de estructuras que lo permitan.
- g) Emplear algunas de las implementaciones de la visualización en el editor.

1.7 PLANIFICACIÓN DEL TRABAJO

Este proyecto lo desarrollan dos personas diferentes, en dos grandes módulos diferenciados (Editor y Gameplay), pero están íntimamente relacionados el uno con el otro. Parte del trabajo que se desarrolla en un módulo se puede reutilizar, con pequeños cambios, al otro módulo. Teniendo en cuenta esto, las tareas a realizar se planifican en el siguiente orden:

- a) Documentación y primeros pasos con Ogre3D [1].
- b) Diseño general de la aplicación.
- c) Creación del esqueleto de la aplicación.
- d) Diseño del modelo del Jugador y los Enemigos
- e) Implementación del Jugador y control de teclado.
- f) Implementación del módulo Gestor de Enemigos.
- g) Inclusión del módulo de físicas.
- h) Implementación de los módulos de Objetos e Ítems.
- i) Implementación de la GUI.
- j) Implementación del módulo Gestor de Archivos.
- k) Inclusión del módulo de sonido.

En esta primera parte de la implementación se crea una aplicación básica de Ogre3D y se hacen pruebas usando la documentación existente en las wikis. Estas pruebas nos permiten ver como trabajar con Ogre de forma básica, aunque la documentación ha sido constante durante todo el proyecto. A continuación, planificamos como separar las diferentes partes de la aplicación y como se desarrollarán cada una de ellas en el tiempo. A partir del diseño creamos un esqueleto de la aplicación, donde podemos ver los diferentes elementos de los que constará.

A partir de este punto, se van desarrollando cada uno de los módulos que componen la aplicación y se van incorporando a los demás.

Las tareas a) y j) son compartidas en ambos proyectos. Se puede reaprovechar la mayoría del trabajo de uno a otro. Por eso, el a) se hace en primer lugar, dando pie a que el editor pueda usar parte de este módulo.

Igual, el punto j) también comparte parte de la implementación en el editor, y es necesario situarlo en las últimas partes del desarrollo esperar la estructura del XML y adaptar la implementación.

Una parte de las tareas k) y g) ha sido implementada por Alejandro Vázquez, que ha incorporado las librerías FMOD y Bullet Physics al proyecto y ha creado los manager para su control.

En el capítulo 2 se detallan los módulos concretos de los que se compone el proyecto Woxfare gameplay y el desarrollo de cada uno de ellos, además del desarrollo de algunas partes de Woxfare editor.

En el capítulo 3 se exponen los resultados de la validación, tanto propia como con usuarios y se dan líneas posibles del futuro del proyecto.

En el capítulo 4 se muestran las conclusiones después de la realización del proyecto y algunas posibles mejoras.

Finalmente, encontramos las referencias y la bibliografía utilizada.

2 DESARROLLO

2.1 WOXFARE: GAMEPLAY

En esta memoria se expone el desarrollo del videojuego Woxfare, desde el diseño de los personajes hasta la interacción de éstos con el entorno que les rodea. El proyecto se compone de diferentes módulos (figura 2.1). En cada uno de ellos se desarrollan los elementos que componen el juego y la forma de iniciar y cargar estos elementos.

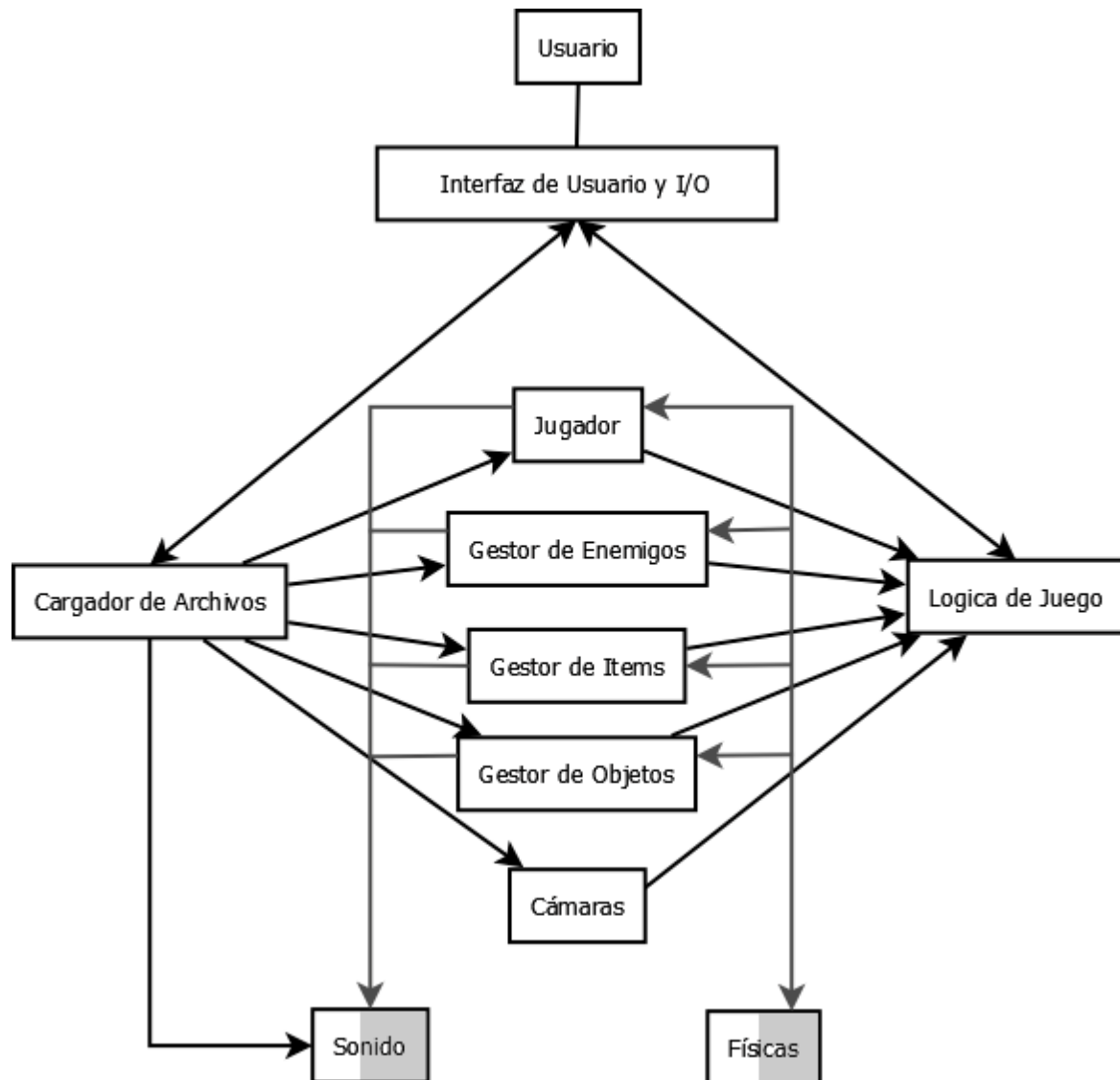


Figura 2.1: Diagrama de módulos

Esta estructura en la aplicación nos permite cargar la información necesaria en el jugador, gestor de enemigos, gestor de ítems y gestor de objetos a través del cargador de archivos. Más tarde, los módulos donde se ha cargado la información interactuarán con los módulos de sonido, físicas y de lógica de juego.

La aplicación desarrollada tiene el ciclo de vida mostrado en la figura 2.2. Cuando la aplicación se inicia, se abre el menú principal, que nos ofrece la opción de cargar el juego o de salir. Una vez

cargado el juego entra en la fase donde se desarrolla la partida (Bucle de juego). En cualquier momento de esta fase de juego, podemos pulsar Pausa para ir al menú de Pausa y volver al menú principal. Cuando salimos del bucle de juego y vamos al menú principal, el juego se descarga y para volver a jugar deberemos cargar algún juego de nuevo. Solo se puede salir completamente del juego desde el menú principal.

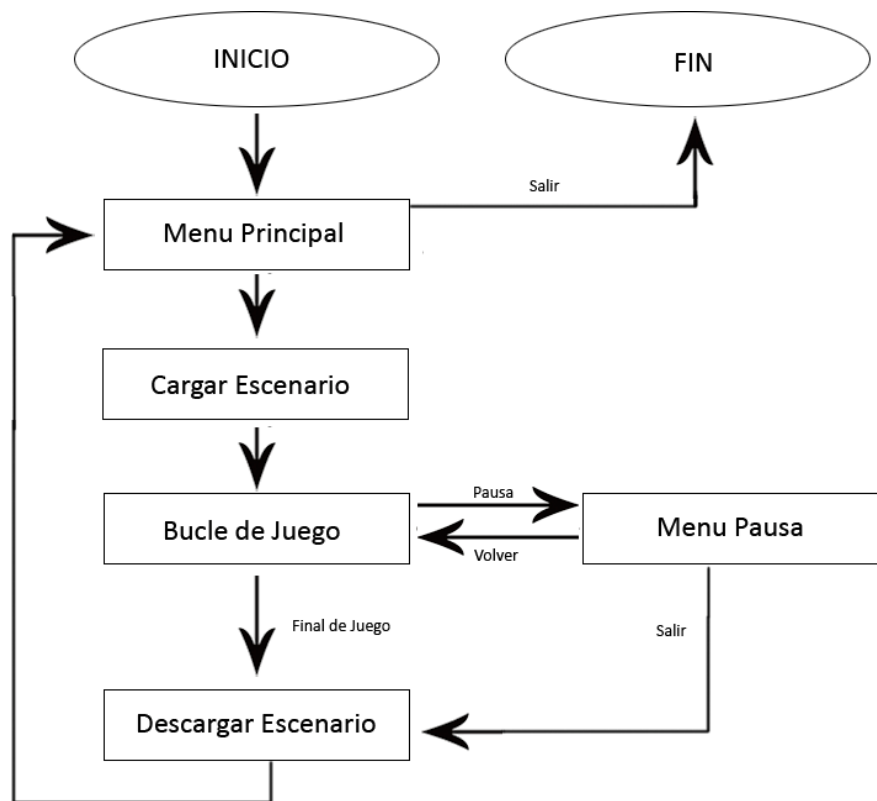


Figura 2.2: Ciclo de vida de la aplicación

En los siguientes apartados se mostrará como se han desarrollado todos los elementos que componen el juego:

- a) En el módulo de **Interfaz Gráfica de Usuario (GUI) y entrada/salida** se muestra el diseño y desarrollo de la interfaz gráfica de usuario y de la capturar los eventos de teclado durante el juego.
- b) El módulo **Cargador de Archivos** nos permitirá cargar la partida personalizada por el jugador y los recursos necesarios para la visualización de esta.
- c) En el módulo de **Lógica de Juego** encontramos en detalle el ciclo de juego: cómo se va desarrollando el juego. Es el encargado de determinar si se ha finalizado la partida o no, si se necesitan más enemigos, etc.
- d) El módulo **Gestor de Objetos** nos permitirá la carga y visualización del escenario y de los objetos de escena.
- e) En el módulo **Gestor de Enemigos** se encarga de cargar, crear, y destruir los enemigos bajo demanda de la lógica de juego y del cargador de archivos.
- f) En el módulo de **Cámaras** se desarrolla la cámara en tercera persona usada en el juego.

- g) El módulo de **Jugador** nos permite visualizar el modelo del jugador en el juego, así como editarlo, moverlo y descargarlo.
- h) El módulo **Gestor de Ítems** desarrolla la lógica de los ítems, los objetos de escena que el jugador puede coger.
- i) En el módulo **Físicas** se incluye la librería de físicas Bullet Physics y se desarrolla toda la interacción de choques y movimiento. No obstante, esta memoria solo contiene como se ha utilizado los elementos de Bullet y el manager desarrollado por Alejandro Vázquez.
- j) El módulo **Sonido** incorpora la librería FMOD [2] y aplica sonidos al videojuego a través del manager desarrollado por Alejandro Vázquez.

Como punto de partida, se usa un esqueleto básico de Ogre3D [3], que hemos personalizado para nuestra aplicación. Éste esqueleto carga los archivos principales que necesita Ogre3D para inicializarse, crea la pantalla de renderizado. Hemos incluido también en esta carga la inicialización del audio y los menús. Es necesario hacer esta primera carga de recursos de Ogre3D para poder mostrar los menús.

2.1.1 GESTOR DE ARCHIVOS

Para representar los elementos que componen nuestros escenarios y nuestras partidas nos hará falta una estructura de datos que permita almacenar la información de los diferentes elementos y nos permita diferenciarlos entre ellos.

Llamaremos entidad a cada uno de los elementos que conforman un escenario de Woxfare Gameplay. Tenemos 5 posibles tipos de entidades:

- **Jugador:** Representa el personaje que conducimos.
- **Bots:** Representan los enemigos, a quienes tenemos que matar y que intentarán matarnos a nosotros.
- **Objetos Dinámicos:** representan los objetos de escena que están atados a golpes o la gravedad. No son recolectables ni destruibles.
- **Ítems:** Son los objetos de ayuda que sueltan algunos enemigos al morir. Estos ítems son recolectables y desaparecen al poco tiempo.
- **Escenario:** El escenario representa todos los objetos estáticos de la escena. No se pueden mover. Un ejemplo son las paredes o pequeños muros.

El módulo gestor de archivos se encargará de cargar la información organizada en 3 tipos de archivo xml, recursos de ogre e imágenes.

- **XML:** La estructura xml que nos proporciona el editor Omega que incluye información interna del juego, como los atributos de jugador y enemigos, la posición de materiales o texturas.
- **Recursos de Ogre:** los componen el archivo del modelo del escenario y los diferentes scripts de materiales. Estos archivos se copian a las carpetas internas del programa para poder usarlos.
- **Imágenes:** Las imágenes son las texturas que el juego creado por el usuario necesita. Todos estos archivos también se deben copiar a las carpetas internas del programa.

Con el objetivo de poder cargar toda la información, se ha creado una estructura de clases (figura 2.3) que nos permite añadir nuevos objetos de distintos tipos y que se usa en objetos dinámicos, ítems, enemigos y armas. Tanto jugador como escenario son únicos y por tanto ellos mismos pueden hacer de generador.

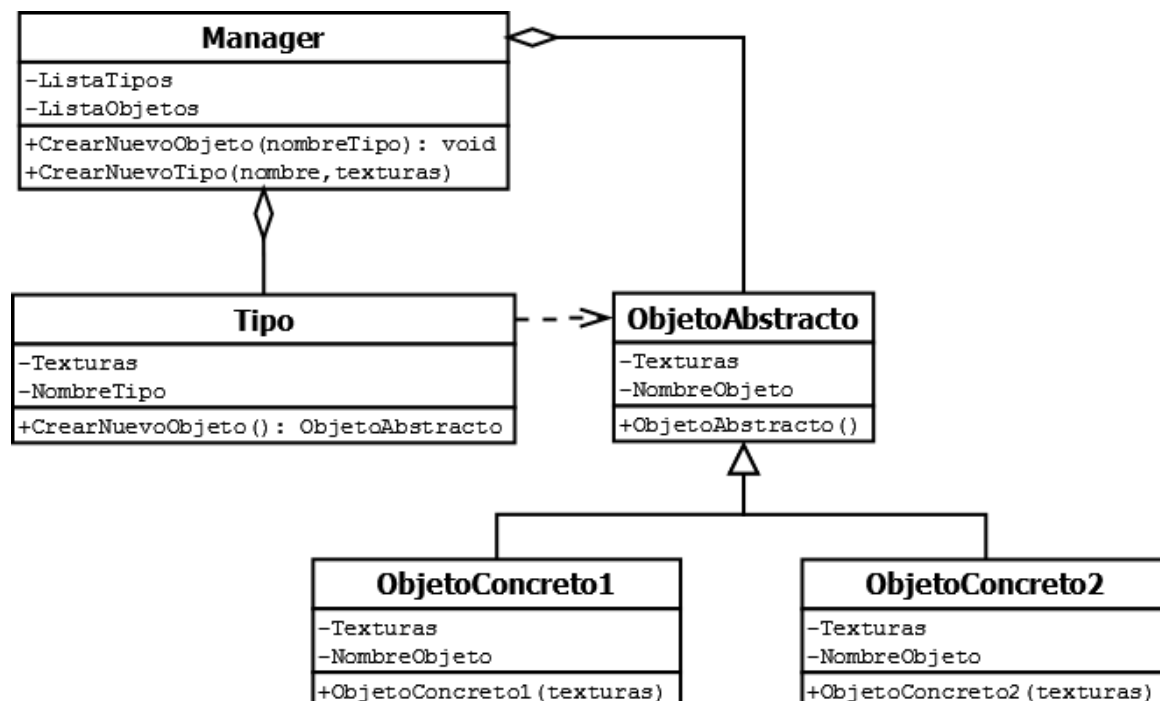


Figura 2.3: Diagrama de clases simplificado.

El **manager** ofrece una interfaz de creación y destrucción de Tipos y Objetos de los diferentes tipos, y podrá ser usado desde cualquier punto de la aplicación. Para ello, implementa el patrón de diseño Singleton [4].

El **tipo** guardará la información cargada desde el cargador de archivos. Esta información son los nombres de los materiales cargados en Ogre, donde se incluye la textura y un nombre básico para distinguirlos entre sí. Además actúa como generador de **objetos abstractos**, creando **objetos concretos**.

Una vez tenemos creado un tipo, podemos crear diferentes objetos concretos usando el manager y el nombre del tipo deseado.

Esto se hace con el objetivo de crear diferentes tipos de *objeto concreto1* y *objeto concreto2*. Un ejemplo sería: Queremos construir dos tipos de ítems que dan vida, con diferentes imágenes, y dos tipos de ítems de oro, con diferentes imágenes. En este ejemplo, el ítem sería el objeto abstracto, el ítem de vida sería el objeto concreto 1 y el ítem de oro sería el objeto concreto 2. Para cada uno de los cuatro objetos deseados se creará un tipo que generará los objetos concretos deseados con sus texturas y su funcionalidad.

La estructura varía ligeramente en los diferentes módulos donde se ha usado. En cada uno de ellos se explica qué particularidades tiene ese módulo y cómo se adapta este diagrama.

En el caso del escenario y el jugador, al ser únicos, la información interna se carga en las propias entidades. Posteriormente, igual que todas las entidades de Woxfare, se inicializará antes de empezar el juego y se descargará al finalizarlo.

2.1.2 JUGADOR

El modelo 3D del jugador y todas sus animaciones están creadas a través de la herramienta Blender [5] y exportado al formato de Ogre3D [6] usando blender2ogre [7]. El personaje es un muñeco (figura 2.4) básico con un torso, una cabeza, dos brazos y dos piernas que no puede doblar.

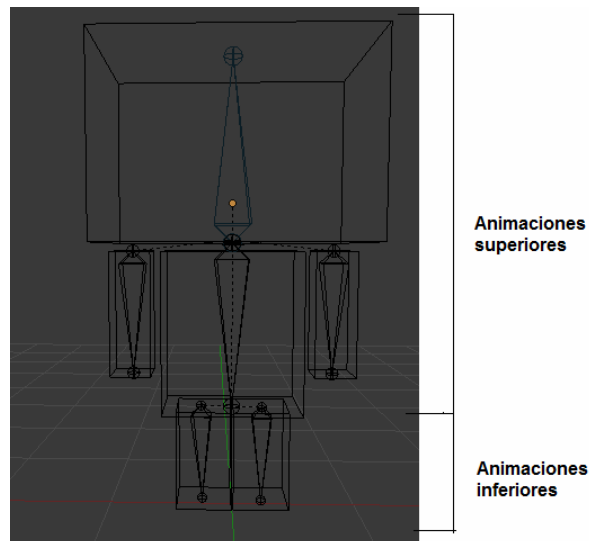


Figura 2.4: Modelo y esqueleto en Blender.

Además, a este modelo básico se le añaden texturas y color desde el editor Omega [8], que nos proporciona esta información en forma de XML. Los atributos para crear el jugador personalizado:

- **Mapa de Texturas y color.** Incluye las texturas del cuerpo la cabeza y el color del modelo.
- **Puntos de Vida Máximos.** Son los puntos de vida máximos con los que empezará el jugador.
- **Armas que puede usar.** Una lista con los nombres de las diferentes armas que puede usar.

Para almacenar y usar las texturas de forma más optima, se hace un procesado que encuentra las texturas con el mismo nombre y usa un solo material para éstas. Estas texturas personalizadas por el creador de la partida darán un aspecto propio la partida, pudiendo orientar el juego a diferentes mundos.

Las diferentes armas que el jugador puede usar estarán almacenadas en un vector, incluyendo en este vector las que ya dispone y las que no.

A demás, la entidad jugador debe almacenar la información necesaria durante la partida:

- **Puntos de Vida actuales.** La variable numérica que marca los puntos de vida actuales del jugador.
- **Oro (o puntos) acumulados.** El oro que lleva acumulado en la partida.
- **Arma actual.** Un apuntador al arma que está usando en ese momento.
- **Lista de Armas.** Una lista de todas las armas que puede usar el jugador.
- **Posición.** Posición en la que se encuentra el jugador en cada momento.

- **Animación Superior.** La animación superior que se está reproduciendo en ese momento.
- **Animación Inferior.** La animación inferior que se está reproduciendo en ese momento.

Estos elementos adicionales nos permitirán saber el estado en que se encuentra el jugador y la información necesaria como el arma que está usando.

Hemos realizado el modelo en Blender y lo hemos animado usando un esqueleto. Es un esqueleto simple que permite mover brazos, piernas, cabeza y torso por separado. Hemos separado estas animaciones en superiores e inferiores: Las **animaciones superiores** son las que modifican los huesos de la cabeza, torso y brazos. Hemos creado animación de ataque cuerpo a cuerpo, de disparo, de salto y caída, de estado de reposo (Idle), de persecución, de nacimiento y de muerte. Por otra parte, las **animaciones inferiores** solo afectan a los huesos de las piernas. En esta parte hay animación de Idle, de caminar, de nacimiento, de muerte y de salto y caída.

2.1.3 GESTOR ENEMIGOS

En esta primera versión de Woxfare tenemos 4 enemigos (bots) distintos, aunque realmente son 2 comportamientos y 2 modelos distintos, combinados entre ellos, que detallaremos en este apartado. Para realizar esta combinación sin necesidad de repetir código se ha seguido la estructura mostrada en la figura 2.5, donde aparece combinada con la explicada en el apartado 2.3.

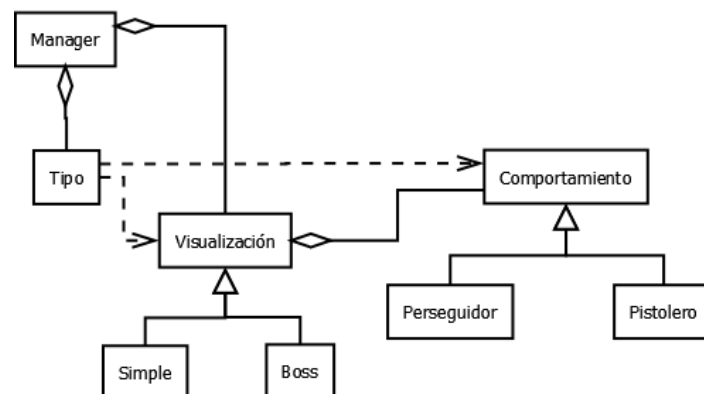


Figura 2.5: Diagrama de clases simplificado.

Por un lado, agrupamos los distintos comportamientos de forma que éstos se puedan usar independientemente del modelo que los esté interpretando. Todos los comportamientos son máquinas de estados. Diferentes eventos, como la distancia al jugador o la vida actual, pueden hacer cambiar de estado. Diferenciamos 2 comportamientos: Follower y Shotter. Ambos tipos de comportamiento implementan una máquina de estados como la de la figura 2.6, pero con alguna variación de comportamiento en cada estado.

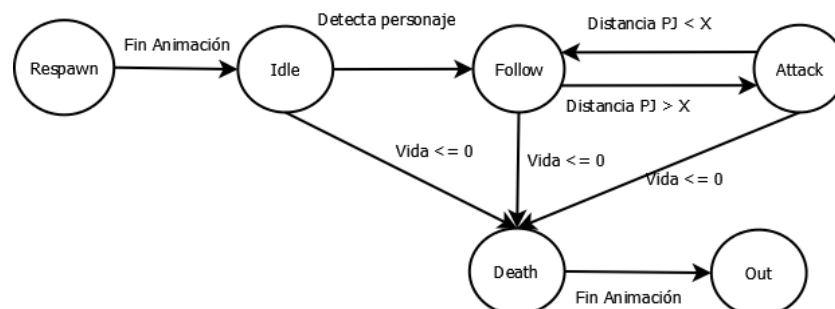


Figura 2.6: Diagrama de estados de la IA.

Todos los enemigos empiezan en un estado inicial de Respawn, o nacimiento, donde ejecutan una animación de salida. Posteriormente, al acabar la animación, pasan al estado de Idle, o espera. Si detectan al personaje lo perseguirán (follow) hasta situarse a una distancia concreta de él y pasará al estado de ataque, o attack. En cualquiera de estos últimos 3 estados el enemigo puede quedar a 0 de vida y esto provocará que pase al estado de muerte, que reproducirá la animación de muerte. Una vez acabe la animación de muerte, se pondrá al enemigo en estado out, o fuera, y se indicará al manager que debe destruirse.

Los dos comportamientos que implementan esta máquina de estados son:

- Follower (Perseguidor): Este tipo de enemigo perseguirá al personaje hasta la muerte. Cuando este cerca de él, le atacará cuerpo a cuerpo.
- Shotter (Pistolero): Este tipo de enemigo te persigue igual que el anterior, pero manteniendo la distancia, desde donde disparará y acertará con cierta probabilidad.

La diferencia principal entre ambos tipos de comportamiento es, principalmente, la distancia que permite la transición del estado de seguimiento (follow) al estado de ataque (attack) y la propia implementación de este estado de ataque.

Por otro lado, disponemos de 2 tipos de visualización de enemigo, aunque visualmente solo se diferencian en el tamaño y las texturas que el jugador haya dado a uno u a otro, como vemos en la figura 2.7. Cada uno de estos tipos contiene un comportamiento, que se debe añadir antes de inicializarlo.



Figura 2.7: Secuencia de diferentes animaciones.

Al cargar la partida, se inicializan los tipos de enemigo, para después poder pedir la creación de uno u otro durante el juego. Estos tipos guardan la siguiente información sobre el enemigo:

- **Mapa de Texturas y color.** Incluye las texturas del cuerpo la cabeza y el color del modelo.
- **Puntos de Vida Máximos.** Son los puntos de vida máximos con los que empezará el enemigo.

- **Oro (o puntos) que proporciona al morir.** Puntos que el jugador recibirá al eliminar a este tipo de enemigo.

- **Arma equipada.** En caso de que el enemigo vaya armado (comportamiento shotter), éste atributo nos marcará que arma está usando.

- **Tipo de visualización.** Que tipo de visualización debe usarse para este tipo de enemigo.

- **Tipo de comportamiento.** Que tipo de comportamiento sigue este tipo de enemigo.

Además, cuando se añade un nuevo enemigo a la escena, se inicializa y se necesitan, igual que con el personaje, algunos atributos más como son los puntos de vida actuales, indicador del estado, etc.

2.1.4 GESTOR DE OBJETOS

El módulo gestor de objetos se encarga de almacenar y gestionar los objetos del escenario. Estos objetos los dividimos en partes estáticas (o inmóviles) y partes dinámicas (partes móviles).

Las partes estáticas representan el terreno, paredes y edificios, formados por cubos, que no se moverán en toda la partida. Estos componentes son exportados desde el editor en forma de una única malla con el fin de eliminar las caras no visibles y la cantidad de materiales usados en el escenario. Además, el escenario se dividirá en secciones para introducirlo como objeto físico y, si hiciera falta, visualizarlo por partes, haciendo más óptima la visualización.

Las partes dinámicas son aquellas partes del escenario que están atadas a la gravedad así como a fuerzas provocadas por el jugador o los enemigos. A diferencia de las partes estáticas, éstas son representadas con cubos de forma que se pueden mover independientemente unos de los otros. De estos objetos cargaremos las texturas que utiliza cada tipo, agrupando los materiales iguales en una fase de optimización de materiales.

Usamos un manager como interficie para crear tipos y objetos. Durante la carga, usaremos el manager para crear los tipos de objetos. En este caso, los tipos de objeto tendrán un vector de posiciones, que son las posiciones iniciales de todos los objetos de ese tipo. Al empezar el juego se crearán todos los objetos de cada tipo, situándose en estas posiciones. Este diseño es similar al mostrado en la figura 2.3,

2.1.5 GESTOR DE ÍTEMS

El gestor de ítems se encarga de los objetos del juego que el personaje puede recoger. En este caso, los ítems van apareciendo, por tanto no podremos crearlos al principio, sino que deberemos disponer de un método que nos los permita generar durante la ejecución del juego. Creamos los tipos durante la carga, igual que con los objetos dinámicos, pero, en este caso, en la carga no se almacenan posiciones.

En el momento que queramos generar un ítem de algún tipo concreto, usaremos el Manager para crear un ítem dando su nombre de tipo y la posición. En nuestro caso, los ítems se generarán según una probabilidad de aparición cada vez que un enemigo muere. Cada enemigo tendrá asignados unos tipos de ítems. Elegirá uno y llamará al manager para crearlo.

Además, en el momento de la carga, se pueden añadir argumentos extra que el ítem puede tratar o no. Por ejemplo: en el caso de los ítems de munición se añade el nombre del arma como argumento "weaponName". De esta forma, solo el ítem de munición tendrá que tratar ese argumento extra, mientras que los ítems de vida y oro no.

Para saber en qué momento el jugador ha recogido un objeto usamos la técnica polling, preguntando en cada iteración a cada objeto si ha sido recogido o no. En caso afirmativo, el manager ejecuta el método de recolección, destruye el objeto y lo elimina de su lista. Hemos elegido este método por su sencillez de implementación y, dado que los ítems solo permanecen un tiempo en la escena, no produce grandes aumentos del tiempo entre *frames*.

2.1.6 GESTOR DE ARMAS

El módulo gestor de armas permite crear las armas que llevarán el jugador y los enemigos. Igual que con los objetos y los ítems, en una primera fase se cargarán las armas disponibles en el juego, generando diferentes tipos de armas. Estos tipos podrán ser usados tanto por el jugador como por los enemigos.

Cuando se inicializa el jugador, usa el gestor de armas para generar un vector de armas, que representarán las armas disponibles por el jugador durante todo el juego. Sin embargo, el jugador inicialmente solo podrá usar la primera arma, y las demás empezaran inhabilitadas. La primera vez que el jugador coja munición de un arma de la que no dispone, este arma se activará.

Los bots sólo dispondrán de un arma como máximo y también se creará a través del gestor y se almacenará en el propio enemigo.

Cada una de las armas lleva asociadas diferentes propiedades que las diferenciará:

- Tipo de disparo:
 - **Disparo único:** Dispara un único disparo a una velocidad media.
 - **Disparo rápido:** Dispara un único disparo a velocidad alta.
 - **Disparo múltiple:** Dispara 3 disparos en triangulo, pudiendo dar hasta a 3 enemigos.
- Número de Balas
- Daño por disparo
- Color

Se ha simulado el disparo utilizando un test de colisión que lanza un rayo desde una posición en una dirección concreta. Además, para el efecto visual se ha usado Billboards [17] y una pequeña animación de texturas. Un Billboard es un plano invisible que siempre está encarado a cámara. Esto nos permite reproducir una pequeña animación o una imagen para dar algún tipo de efecto como, por ejemplo, que un barril esté en llamas. Si la cámara se mueve, el Billboard se encara automáticamente, siempre mostrándose frontalmente hacia ésta.

2.1.7 LÓGICA DEL JUEGO

Se trata del módulo principal del juego: Desde el momento en que hacemos clic el botón de jugar hasta que se acaba el juego.

Cada juego forma una partida. Una partida está formada por un conjunto de rondas. Cada ronda a su vez tendrá 2 fases: un tiempo de espera, donde no hay enemigos que matar, y uno de acción, donde el jugador tendrá que eliminar a todos los enemigos. La partida, también, es la unidad indivisible que podemos cargar en el juego. Toda la información importada del editor Omega [16] conformará los detalles de estas rondas.

Para finalizar una ronda, el jugador debe acabar con todos los enemigos presentes en el escenario. Cuando lo consiga, se comprobará si el número de rondas ha llegado a su máximo, de forma que el juego acaba. En caso contrario, se recalcularán los atributos de la ronda, dando lugar al inicio de la siguiente. Los atributos recalculados en cada ronda son el número de enemigos y el número de muertos actual de la ronda.

Una vez introducido el funcionamiento de las partidas, a continuación explicaremos más al detalle este módulo.

En primer lugar, se inicializa la cámara, el jugador, el escenario, los enemigos y los ítems, creando los objetos visuales necesarios. Justo después de esto, se calculan los atributos iniciales (número de enemigos para esta ronda, número de ronda e incremento de enemigos entre rondas) para la ronda 1. A continuación entramos en una fase de espera, donde el jugador puede moverse libremente por el escenario, pero aún no hay presente ningún enemigo. Esta fase de espera dura unos 5 – 6 segundos, y nos sirve para poder recoger los ítems que hayan podido quedar desperdigados. Una vez finalizada esta espera, empezamos a poner enemigos en intervalos de tiempo definidos, desde las salidas que el usuario marcó desde el editor. Si ya no quedan más enemigos por añadir, solo hay que esperar a que el jugador los elimine a todos. En ese momento, comprobamos si se ha llegado al final o no, teniendo en cuenta que un -1 en el número máximo de rondas indicaría que no se debe acabar nunca.

En la figura 2.8 mostramos un diagrama de flujo que muestra lo que se ha explicado anteriormente. Como vemos, dentro del bucle se capturan las pulsaciones del teclado y se actualizan los diferentes elementos que conforman la escena y, posteriormente, se trata la lógica que sigue nuestro juego, al crear enemigos y al gestionar las rondas.

2.1.8 INTERFAZ GRÁFICA DE USUARIO (GUI)

La GUI se compone principalmente de 3 menús: Menú Principal, Menú Pausa y la GUI visible durante el juego. Cada una de ellas pretende ser sencilla y clara, para que el usuario final pueda entenderlo con facilidad.

Todos los menús de la interfaz de usuario siguen 3 etapas: entrada, visualización y salida.

- La **entrada** crea los elementos visibles del menú, como botones o imagen de fondo.
- La etapa de **visualización** es el tiempo en que el menú está visible. Esto se indica a través de una variable de estado. Durante esta etapa se escuchan los eventos de clic sobre los botones de la GUI y se actualizan los elementos cambiantes.
- La etapa de **salida** destruye los objetos visibles del menú.

En la implementación de la GUI se ha usado *OgreBites* [18] y *SDKTrays* [19], que nos proporcionan un conjunto de elementos como botones o paneles, y permite esperar eventos de estos a través de un *Listener*.

Para insertar las diferentes imágenes que vemos en los diferentes menús se han usado *Overlays* de *Ogre3D* [20], que nos permiten mostrar imágenes sin que podamos recoger ningún evento de ellos.

a) Menú Principal

El menú principal constará de una imagen con el título Woxfare, además de los botones de Jugar, Cargar y salir (figura 2.9).

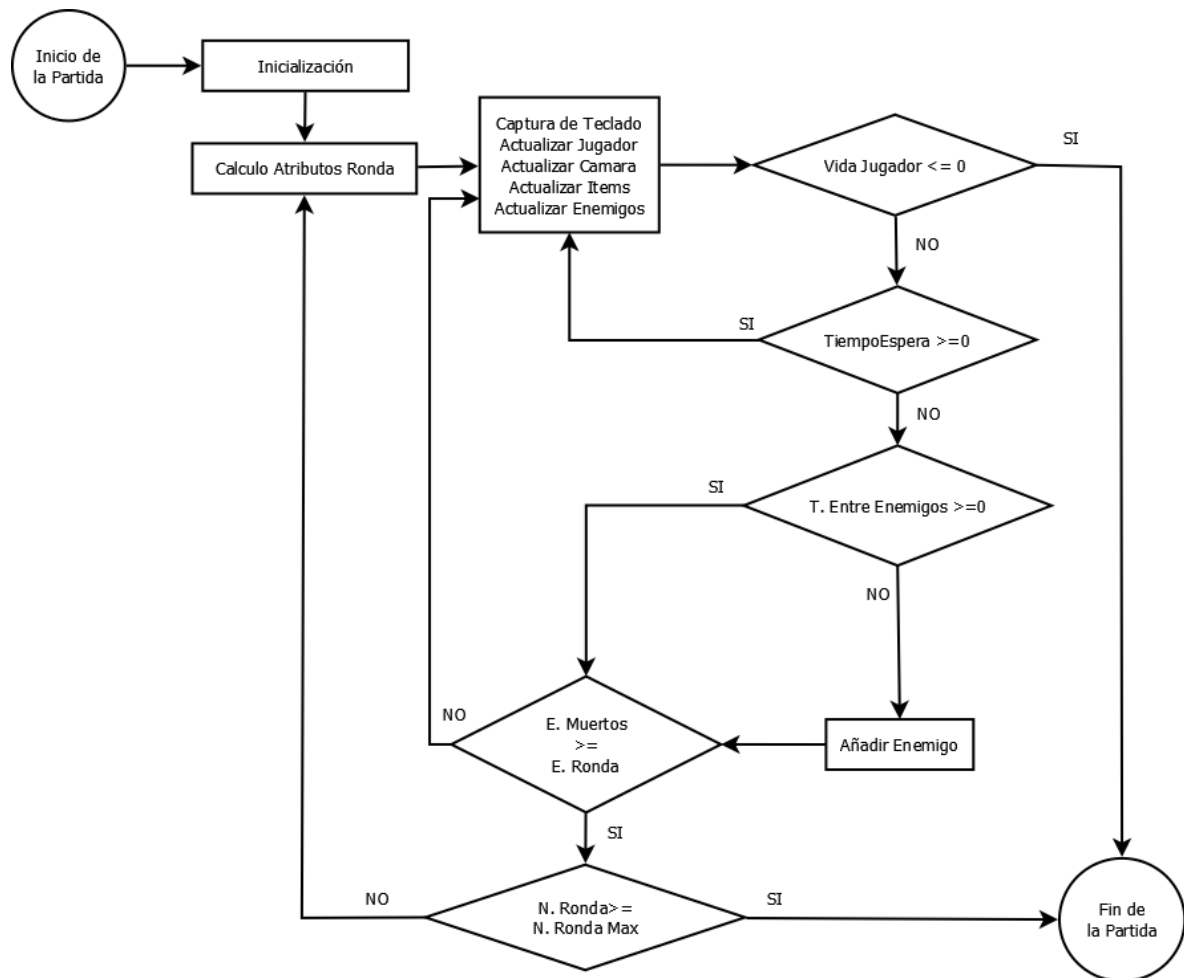


Figura 2.8: Diagrama de flujo del bucle principal.

- **Jugar:** Empieza deshabilitado. Permitirá jugar al juego que previamente hayamos cargado.
- **Cargar:** Abre una ventana que permite cargar un archivo xml, con la composición del escenario. Si se ha cargado con éxito algún escenario se habilitará el botón de Jugar.
- **Salir:** Permite cerrar el juego por completo.

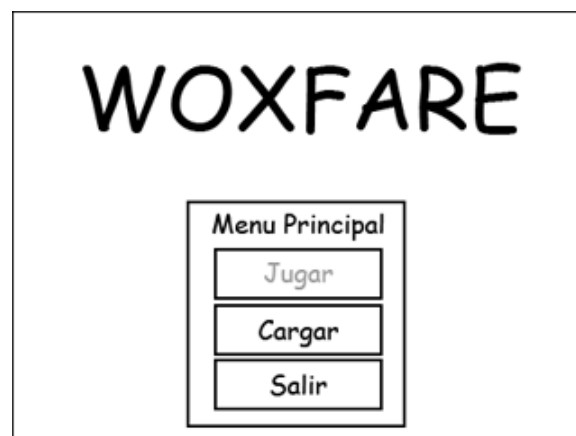


Figura 2.9 Esbozo del menú principal.

b) Menú de Pausa

El menú de pausa se abre cuando el jugador pulsa ESC durante el juego. Este menú permite dos acciones (figura 2.10):

- **Continuar:** Permite continuar jugando la partida actual.
- **Salir:** Cierra la partida y vuelve al menú principal.

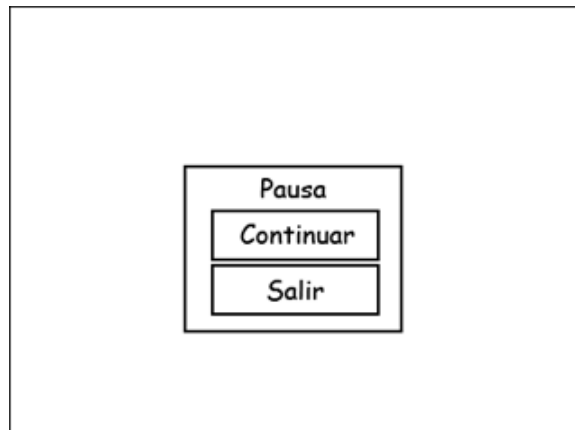


Figura 2.10: Esbozo del menú de pausa.

c) GUI de Juego

El menú del juego deberá mostrar toda la información necesaria durante las partidas. Combina tanto información escrita como imágenes. También incluye información numérica. Se ha tenido que implementar la función que inserta una textura con un número u otro dado un entero. Esta función carga una textura que se divide en 10 partes iguales y tiene escritos números del 0 al 9 (figura 2.11). A través del mapa UV del Overlay modificamos el número mostrado. Esto nos permite mostrar números más vistosos que solo usando texto con alguna fuente especial.

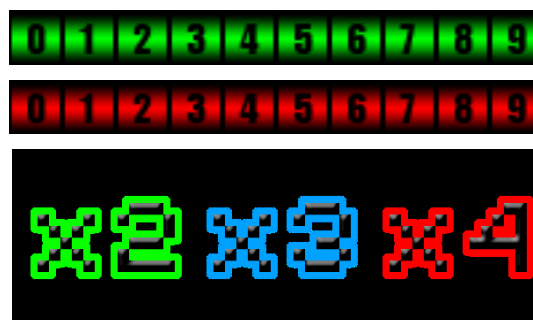


Figura 2.11: Texturas para números.

La interfaz de juego contendrá la siguiente información (figura 2.12):

- **Arma actual.** Indicador del tipo de arma: blaster, escopeta o ametralladora
- **Munición Actual y máxima.** Munición que tiene el arma que se esta usando.
- **Vida.** Puntos de vida actuales del jugador.
- **Puntos.** Puntos, o oro, acumulados por el jugador.
- **Enemigos eliminados.** Número de enemigos eliminados en esa ronda.

- **Racha activa.** Indicador de la racha de bajas. Esta indicado con un multiplicador.

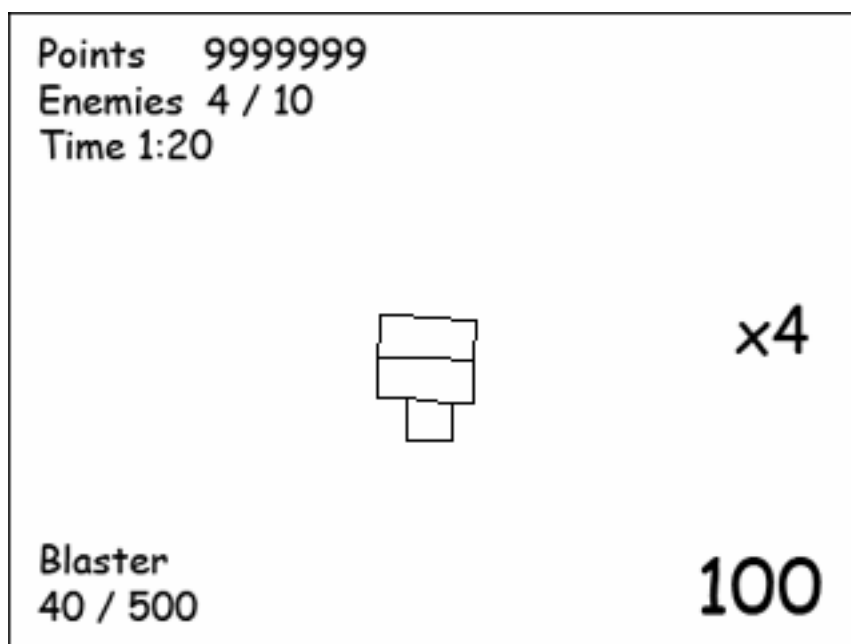


Figura 2.12: Esbozo de la pantalla de juego

d) Inicio y Fin de la partida

Además de las GUI con alguna función integrada, diseñamos y añadimos 3 pantallas más que no tienen ningún tipo de funcionalidad:

- **Pantalla de carga:** Esta pantalla aparecerá mientras se carga el juego, antes del inicio, y mostrará los controles por defecto sobre una imagen de un teclado.
- **Pantalla de Partida ganada:** Cuando ganemos la partida, antes de volver al menú principal, se mostrará un texto indicando que hemos ganado.
- **Pantalla de Partida perdida.** Si perdemos la partida, se mostrará un texto indicando que hemos perdido.

Usaremos Overlays de Ogre para mostrar por pantalla imágenes que cubren gran parte de ésta y que nos proporcionan la información antes descrita.

En este módulo también se ha implementado el control de teclado, que permitirá al usuario interactuar con la aplicación en la fase de juego. Se ha usado la librería OIS [21] para la captura de los eventos del teclado.

Nos interesa saber constantemente qué teclas se están pulsando durante la partida. Nuestra aplicación esperaba eventos al pulsar o soltar una tecla. Con el objetivo de poder captar la pulsación simultánea de varias teclas y contemplar, también, la pulsación prolongada de éstas se almacena en un vector las teclas cuando son pulsadas y se eliminan del vector al soltarlas.

Además, la estructura del manager de teclado nos permite definir las teclas a usar para las diferentes acciones que nos permite el juego. Esto nos puede permitir cambiar las teclas de las acciones incluso mientras el juego está en funcionamiento. Las acciones disponibles en Woxfare y

sus controles por defecto son mostrados en la figura 2.13. Todas estas acciones permiten al jugador moverse por el escenario y interactuar con los objetos presentes en la escena.

Tecla	Acción
W	Mover hacia delante
S	Mover hacia atrás
A	Mover hacia la derecha
D	Mover hacia la izquierda
	Giro a la izquierda
	Giro a la derecha
	Giro 180 grados
	Disparar
Espacio	Saltar
Ctrl.	Cambiar de arma arriba
Shift	Cambiar de arma abajo
Escape	Pausa

Figura 2.13: Acciones y teclas por defecto.

2.1.9 CÁMARA

El objetivo principal de la cámara es permitir al jugador tener siempre una amplia y clara visión del escenario, y de los enemigos que le rodean por los lados. Dado que el personaje se mueve de forma limitada al un plano 2D excepto cuando salta, y que solo puede disparar a la misma altura, vemos conveniente usar una cámara en tercera persona, que siga al personaje en todo momento, situada prácticamente encima de él.

Como vemos en la figura 2.14, la cámara permite ver la zona que rodea al jugador. Para mantener esta zona visible, la cámara seguirá al jugador en todo momento, situándolo en el centro de visión.

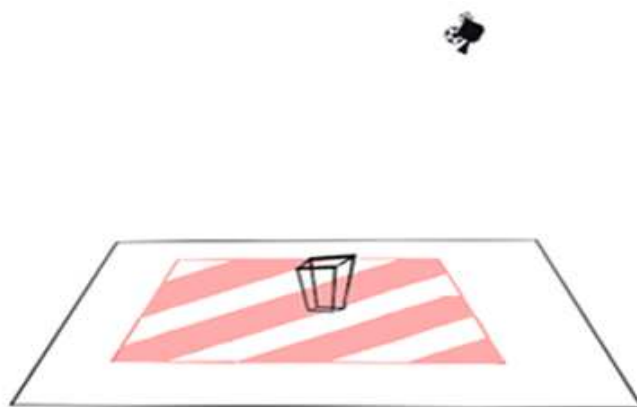


Figura 2.14: Esquema para representar la zona visible del escenario.

2.1.10 MÓDULO DE FÍSICAS

El modulo de físicas nos permite realizar el cálculo del movimiento, disparos y choques de nuestro videojuego. Para el uso de la librería Bullet Physics hemos creado un manager que nos permite añadir y destruir elementos físicos de la escena y hemos adaptado la estructura de la aplicación.

Para poder distinguir los diferentes elementos (enemigos, escenario, objetos dinámicos o ítems) que obtenemos de los test de colisión creamos una clase intermedia entre las entidades de Woxfare y los objetos físicos de bullet (`btRigidBody` [22]). De esta clase heredan los enemigos, las partes del escenario, los ítems y los objetos dinámicos, como se muestra en la 2.15. Contiene una única función y una enumeración de tipos que permite distinguir qué clase de elemento se trata.

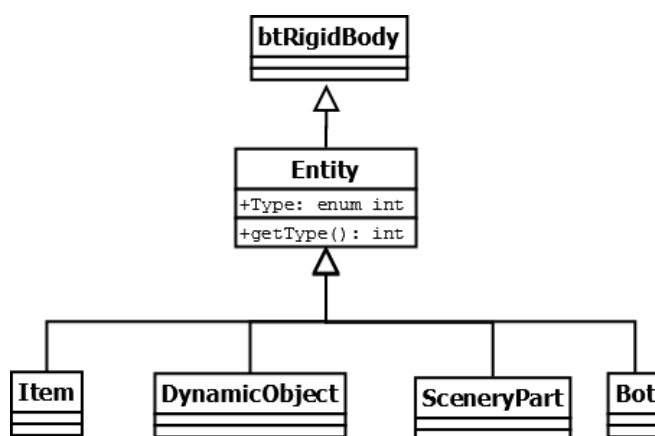


Figura 2.15: Diagrama de clases simplificado.

Para todos los objetos menos el escenario hemos usado cajas para delimitar el espacio físico que ocupa (figura 2.16). Creemos que esta simplificación no afecta al efecto visual que produce en el choque. Además, La rotación de los objetos está limitada al eje y para evitar movimientos extraños provocados por las fuerzas aplicadas en diferentes puntos.

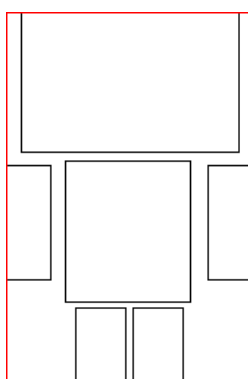


Figura 2.16: Esquema del modelo y del objeto físico en rojo.

El movimiento de todos los elementos móviles de la escena se realiza a través de *Bullet*, aplicando fuerzas o impulsos a los objetos para provocar que se muevan.

Para adaptar la forma del objeto físico del escenario a su visualización hemos realizado un proceso en que se calcula el objeto físico a partir de los triángulos que componen la forma del escenario [16].

2.1.11 MÓDULO DE SONIDO

El módulo de sonido nos permite reproducir sonidos en toda la aplicación. Desde la música de fondo hasta sonidos de botones o disparos. Para poder reproducir sonido desde cualquier punto de la aplicación, se centraliza toda la implementación en un manager, que podrá ser usado desde cualquier punto de la aplicación. Este manager usa la librería de audio FMOD. El desarrollo de este manager y la inclusión de la librería FMOD han sido desarrollados por Alejandro Vázquez [16].

Aquellas entidades que necesiten usar sonidos, podrán cargarlos durante la fase de inicialización y reproducirlos en cualquier momento de la ejecución, haciendo uso del manager. Los sonidos que se han incluido en la aplicación son:

- Música del Menú Principal
- Sonido de Botones de los menús
- Música de fondo de la Partida
- Sonido de disparos
 - o Sonido de Escopeta
 - o Sonido de Pistola
 - o Sonido de Ametralladora
- Sonidos de recogida de ítems
- Sonido al sufrir daño

Los sonidos han sido escogidos por nosotros mismos de otros videojuegos ya existentes. La mayoría de ellos de Counter Strike, como los sonidos de armas, la música del menú principal y la música de fondo durante el juego. Los demás sonidos se han escogido intentando que estos sean estándar para la mayoría de escenarios que se pueden construir con el editor.

2.2 CONTRIBUCIÓN EN WOXFARE: EDITOR

Con el fin de reducir el tiempo de desarrollo, en este proyecto también se ha desarrollado el módulo de Ogre y el módulo de optimización de escenario en el editor. En el diagrama de módulos de la figura 2.17 mostramos los módulos del proyecto *“Woxfare: Un videojuego de supervivencia. Diseño e implementación del editor Omega”* [16]. Las partes son el módulo de Ogre, encargado de la visualización, y el Optimizador, que realizará la optimización de escenario para el Gameplay.

2.2.1 OGRE

La utilización de la librería Ogre es similar tanto en gameplay como en editor. El trabajo se dividió de forma que formaría parte de este proyecto la preparación del módulo de Ogre del editor de forma que Alejandro Vázquez pudiera usar esa funcionalidad sin tener que entrar al detalle en como funciona Ogre.

Este módulo de Ogre integra el motor gráfico de Ogre3D y se implementan los diferentes elementos que generarán los objetos visuales en la escena, además de los objetos visuales que permitirán editar el cubo y el jugador.

Se han modelado a través de código Ogre3D los diferentes modelos usados en el editor: Objetos del escenario (cubos sólidos), el cubo de edición (separando el modelo por caras) y el personaje principal (separando el modelo por caras).

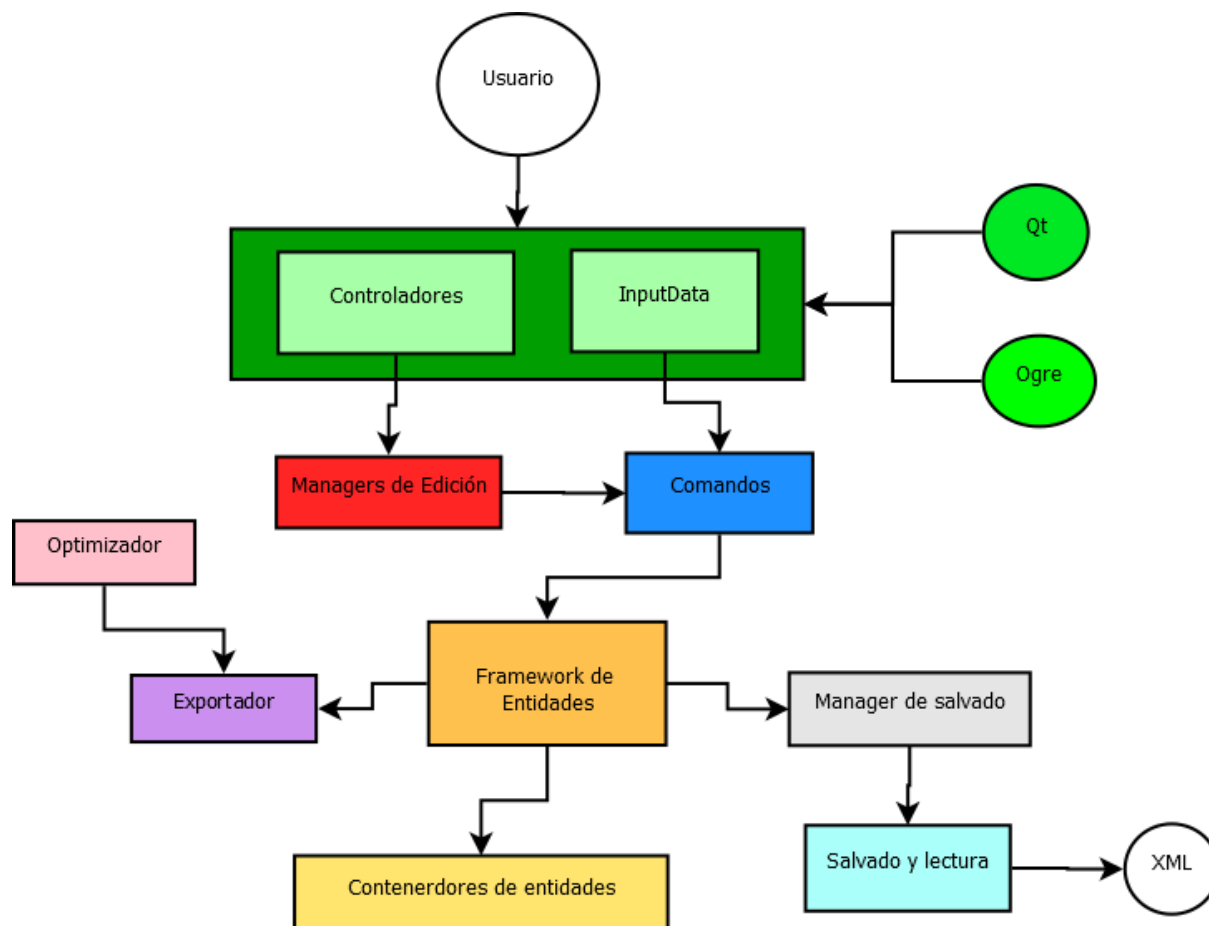


Figura 2.17: Diagrama de módulos del proyecto Woxfare: Editor [16].

Se ha implementado, también, la interacción ratón con los objetos visualizados, es decir, la interacción que provoca mover el cursor o hacer clic en la edición de escenarios, en lo referente a Ogre3D. Para esta interacción se ha hecho uso de rayos que permite lanzar Ogre3D [23] para ver que objetos colisionan con él. No obstante, el rayo sólo comprueba la colisión con la Caja mínima encarada a ejes (AABB) y por tanto no se pueden aplicar rotaciones si queremos usar este método. En la figura 2.18 mostramos un ejemplo del funcionamiento del rayo. Como vemos, pese a que el objeto esta torcido, el rayo colisiona únicamente con la caja AABB, dando resultados incorrectos en la colisión. En nuestro caso, si mantenemos los objetos alineados no se producirá este efecto.

2.2.2 OPTIMIZACIÓN DE ESCENARIO

Aunque la optimización del escenario sólo se usa en el gameplay, la implementación se ha hecho en el editor con la idea de simplificar la carga en el gameplay. Esta optimización consta de 2 partes:

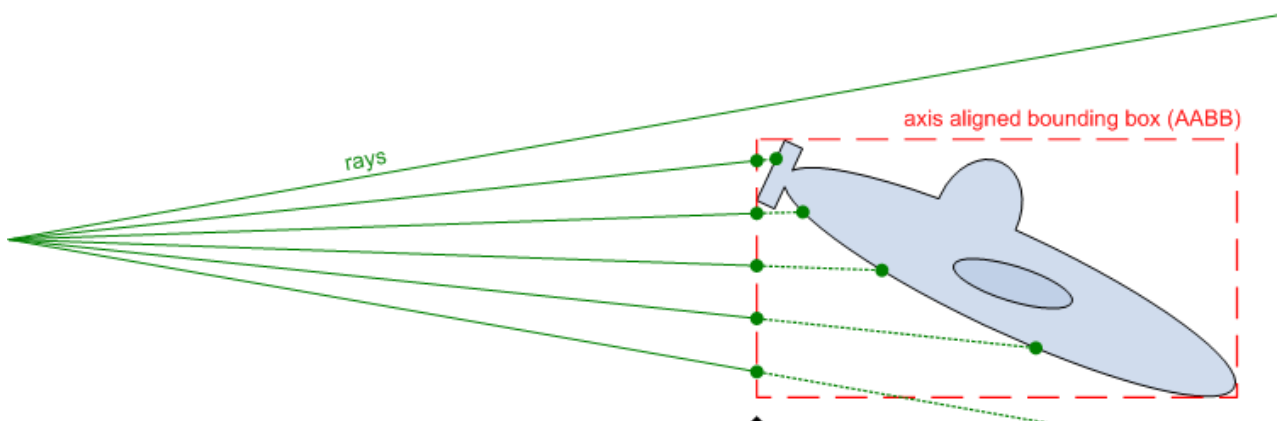


Figura 2.18: Ejemplo de funcionamiento del rayo

- Por un lado, encuentra las caras que están entre cubos consecutivos y las elimina, como muestra la figura 2.19. Al ser posiciones enteras, podemos saber que cubo esta tocando con otro y podemos eliminar la caras no visibles de una construcción, ahorrando el procesado de estas caras durante la ejecución del juego. La cara roja de la figura será eliminada ya que nunca se podría llegar a ver.

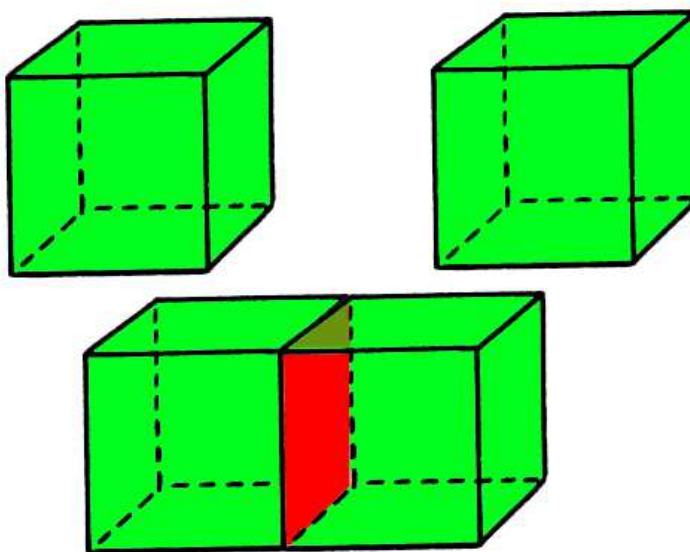


Figura 2.19: Esquema para representar la optimización de caras..

- Por otro lado, igual que con las demás entidades del gameplay, se optimizan los materiales, buscando los que tengan la misma textura y el mismo color y los unifica en uno solo. En la figura 2.20 vemos a la izquierda el cubo del editor formado por 6 materiales distintos: top, bottom, right, left, front y back; donde top, right, back y fron son iguales, de color azul, y left y bottom son iguales entre ellos, pero diferentes con los anteriores. El mismo objeto representado en el juego solo constará de 2 materiales: blue y yellow.

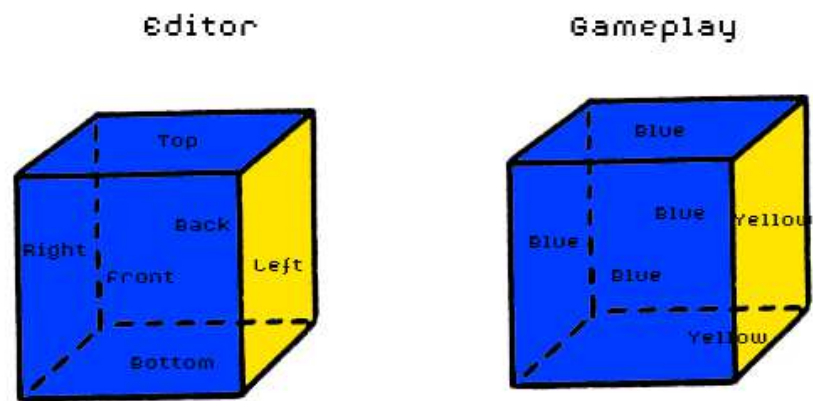


Figura 2.20: Esquema para representar la zona visible del escenario.

3 RESULTADOS

En el apartado de resultados exponemos los resultados obtenidos durante el diseño e implementación de Woxfare (Gameplay). Se comprueba también que la aplicación funciona correctamente y que se ha integrado bien con el editor Omega.

Con el objetivo de mostrar estos resultados, dividiremos los apartados en 5 partes:

1. Resultados de la implementación del Gameplay, cargando un escenario de editor Omega, donde mostramos diferentes escenas del juego.
2. En segundo lugar, los resultados del modelado del personaje y las animaciones.
3. Resultados del diseño e implementación de la GUI y comparativa con el diseño inicial.
4. Los resultados de la integración con el editor de escenarios [16].
5. Como último punto se mostrarán algunas ideas sobre el futuro del proyecto.

3.1 RESULTADOS DE WOXFARE

A lo largo de este proyecto hemos diseñado e implementado un videojuego 3D de tipo supervivencia, que permite personalizar la apariencia del personaje, de los enemigos, de los objetos de escena y del escenario. Además permite personalizar la secuencia de acción pudiendo elegir el número de enemigos, la posición de salida, el número de enemigos más poderosos (jefes) o los ítems que dejará caer cada uno. También permite modificar las armas que portaran cada uno de ellos, incluido el jugador, al que se le podrán asignar varias.

En las figuras 3.1 y 3.2 veremos 2 capturas del juego, donde podemos ver al jugador saltando y siendo disparado por los enemigos. También vemos que se han utilizado el uso de sombras, que proporciona Ogre 3D.

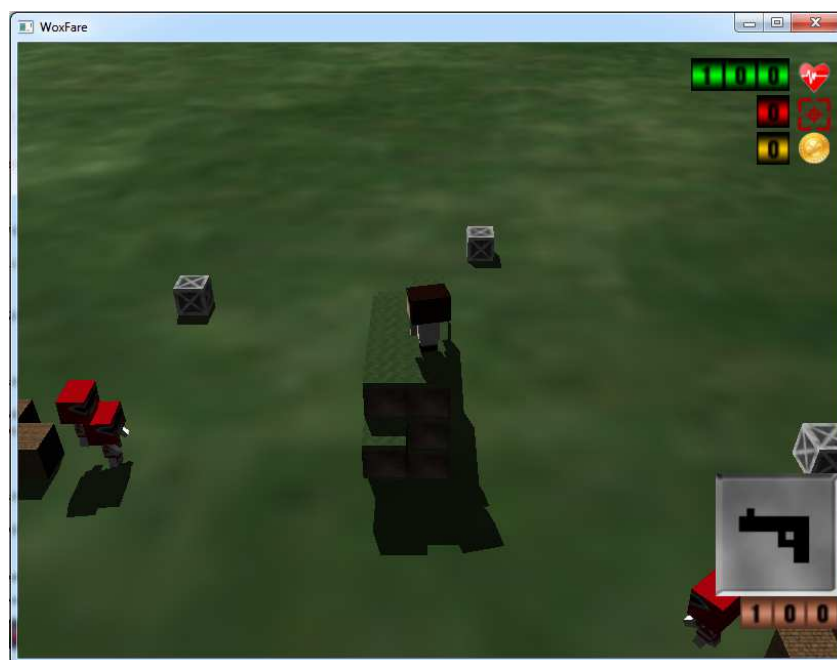


Figura 3.1: Captura del juego donde con el personaje quieto.

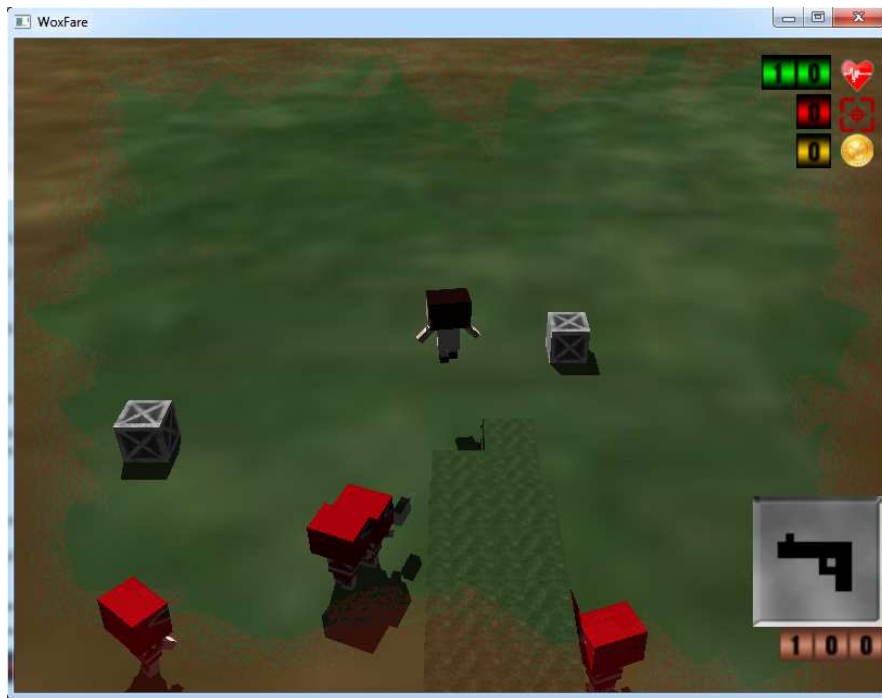


Figura 3.2: Captura del juego con el personaje saltando recibiendo disparos.

3.2 RESULTADOS DEL MODELADO

Hemos diseñado y modelado el personaje utilizando la herramienta blender y lo hemos exportado al formato de Ogre3D (.mesh, .skeleton y .material) usando Blender2Ogre y las herramientas de conversión de Ogre, Ogre Command Tools [24]. Como resultado, obtenemos el modelo y el conjunto de recursos que necesitaremos para su visualización en el juego. En la figura 3.3 podemos ver el modelo base.

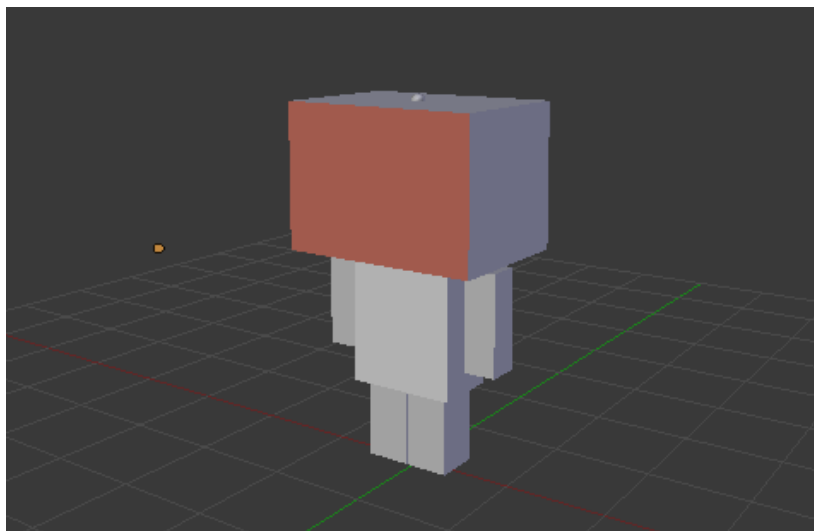


Figura 3.3: Modelo Base

También quisimos visualizar a través de Blender el modelo texturizado, aunque los modelos cargados en el juego serán texturizados desde el editor (Omega). En la figura 3.4 vemos una prueba

de texturizado de un modelo con un lanzamisiles. Las imágenes del modelo se obtuvieron en la Web de cubecraft [2]; la del arma desde google, una imagen de Team Fortres 2 [25].

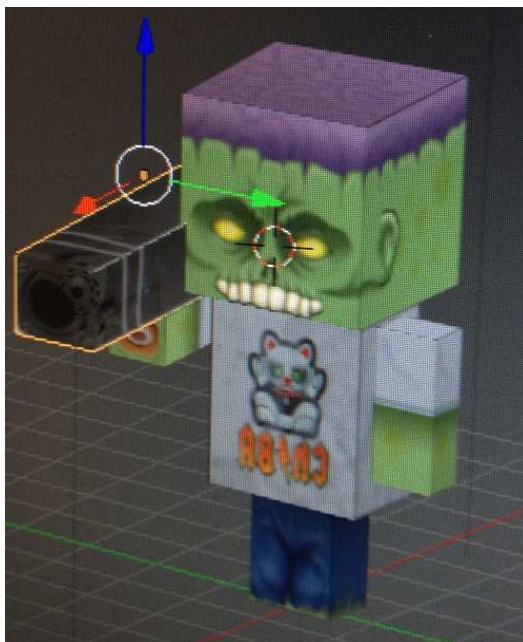


Figura 3.4: Imagen de cubecraft a la izquierda, y modelo de prueba a la derecha.

Además hemos diseñado las diferentes animaciones. En la figura 3.5 vemos diferentes secuencias de las animaciones.

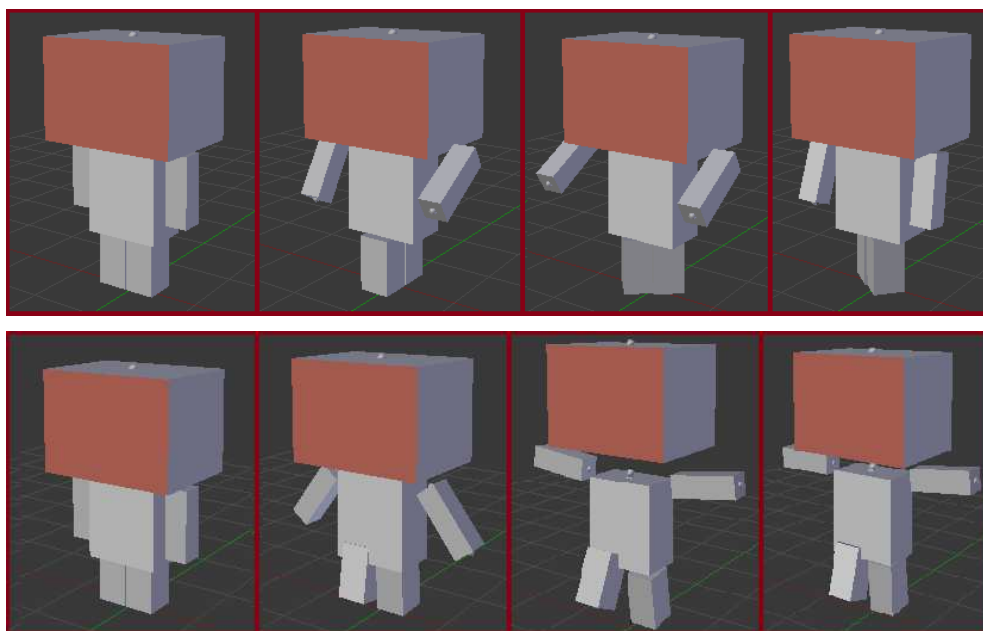


Figura 3.5: Secuencias de las animaciones de estado de reposo y salto.

3.3 RESULTADOS DE GUI

Como resultados de la GUI podemos mostrar los diferentes menús que conforman el juego:

- a) Menú Principal (figura 3.6)

- b) Menú de Pausa (figura 3.7)
- c) GUI de juego (figura 3.8)
- d) Pantalla de Controles (figura 3.9)
- e) Pantalla de Juego Ganado (figura 3.10)
- f) Pantalla de Juego Perdido (figura 3.11)

En las figuras se muestra la GUI diseñada durante el proceso de desarrollo y como finalmente ha quedado el resultado.

En la GUI de juego (figura 3.6) podemos ver números del 1 al 5 que nos indican los diferentes elementos que debemos visualizar mientras jugamos:

1. Puntos de Vida
2. Enemigos muertos en esa ronda
3. Oro total
4. Imagen del tipo de arma y balas
5. Racha de enemigos.

En la figura 3.8 vemos como después de clicar el botón de Load se abre el dialogo para seleccionar el archivo que contiene la información XML.



Figura 3.6: Secuencia de diferentes animaciones.

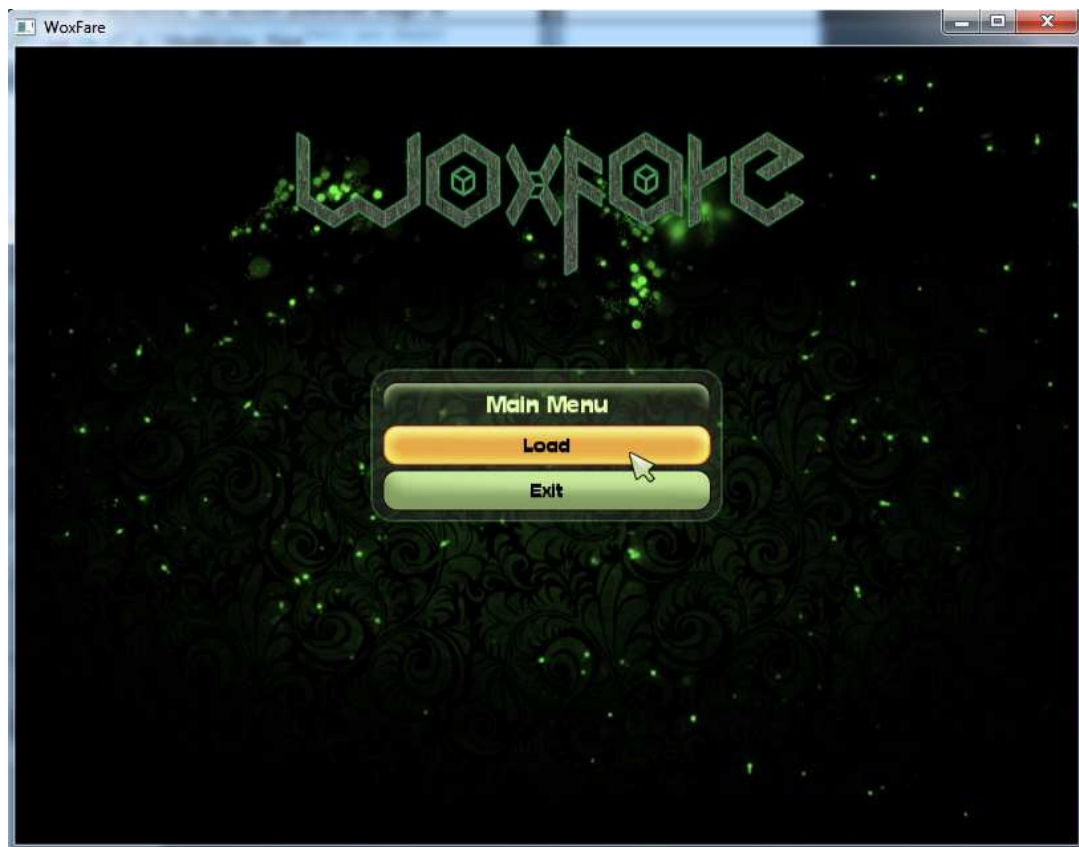


Figura 3.7: Captura del menú principal.

Como podemos ver en la figura 3.7 los botones tienen un efecto al pasar por encima que los cambia a color amarillo. Además, al hacer clic sobre ellos reproducen un sonido.

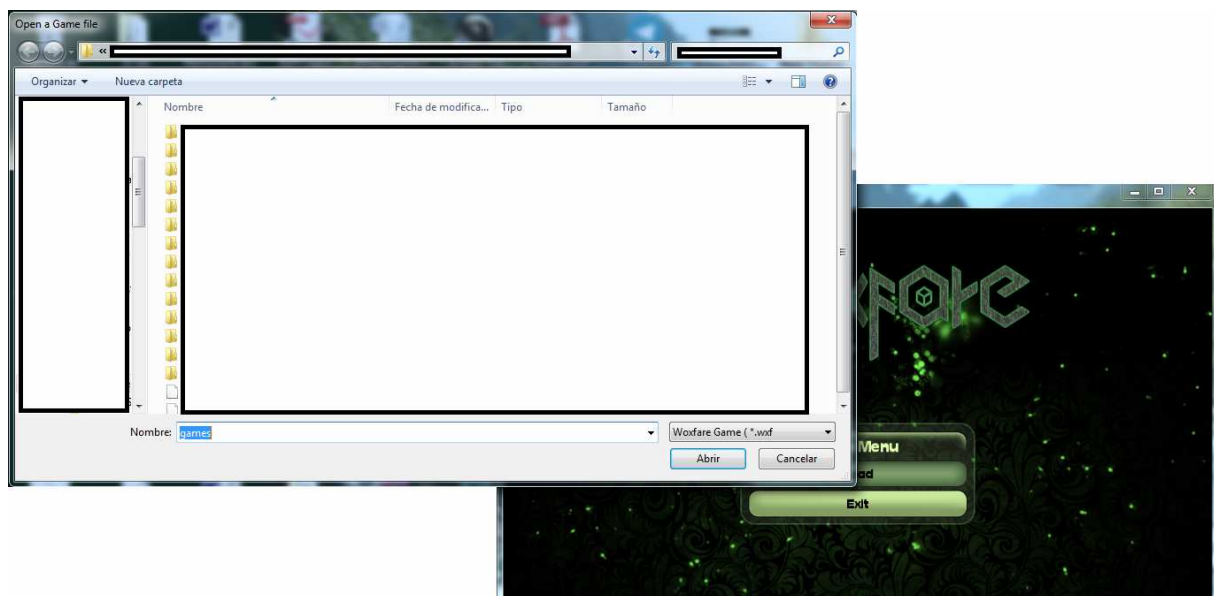


Figura 3.8: Captura del dialogo para buscar juego.

En la figura 3.9 mostramos las teclas por defecto mientras se cargan los recursos del juego.



Figura 3.9: Captura de la pantalla de carga.

Cuando ganamos (figura 3.10) o perdemos (figura 3.11) el juego, nos podremos seguir moviendo durante unos segundos hasta ser devueltos al menú principal. Mientras estamos en esa espera, se visualizará la imagen de Juego ganado o Juego perdido.



Figura 3.10: Captura de juego ganado.



Figura 3.11: Captura de juego perdido..

3.4 RESULTADOS DE LA INTEGRACIÓN

Las comunicaciones entre los dos proyectos se realizan a través de un documento XML y un conjunto de archivos asociados. En este sentido, la integración ha sido sencilla, dado que no hay módulos de proyectos diferentes que interaccionen entre ellos. A pesar de ello, se ha tenido que adaptar el formato varias veces para llegar al mejor consenso.

Además, dado que algunas tareas han sido compartidas, hemos sincronizado constantemente los proyectos y los avances que cada uno hacíamos. Para esta sincronización se han intentado seguir un conjunto de reglas de estilo de programación y se ha compartido el trabajo a través del repositorio Assembla, usando SVN. Casi finalizando el proyecto nos hemos visto obligados a cambiar de repositorio por haber excedido la capacidad.

3.5 VALIDACIÓN DE USUARIOS

Se ha realizado una pequeña prueba ante usuarios de la cual se pueden sacar las siguientes conclusiones:

- a) Los menús son fáciles de entender, además de una GUI de juego clara, aunque demasiado sencilla.
- b) Al principio la mayoría de jugadores intentan disparar y orientarse con el ratón. La forma de movimiento se les hace extraña en un principio.

- c) Les hubiera más gustado diversidad en las armas y las formas de disparo: granadas o lanzamisiles.
- d) Muchos han destacado que podría llegar a ser un juego bastante divertido dado que permite modificar los escenarios y los enemigos desde el editor, a diferencia de otros juegos del estilo.
- e) Un modo multijugador le daría mucho entretenimiento.

Como resultado podemos destacar que les ha gustado la idea de poderse crear sus propios mundos y poder jugarlos de forma supervivencia, pero aun quedaría un largo camino si se quisiera comercializar.

3.6 FUTURO DEL PROYECTO

Como se destaca en los capítulos introductorios, este proyecto pretendía ser una base de trabajo a través de la cual formarse y realizar un pequeño prototipo de videojuego de tipo supervivencia. Vistos los resultados, se podría valorar seguir trabajando tanto en el proyecto para dar más jugabilidad a través de nuevas opciones, o para adaptar el juego a plataformas móviles.

Si se continuara, se desarrollaría, también, una plataforma Web a través de la cual compartir los escenarios, puntuaciones y diseños de cada jugador, permitiendo crear, poco a poco, una comunidad.

El último punto a destacar es que, en caso de continuar con el proyecto, se optaría por una filosofía OpenSource con la misma idea social que persigue todo el videojuego.

4 CONCLUSIONES

A nivel global:

- Se ha desarrollado un juego 3D llamado Woxfare y un editor que permite modificar los elementos del juego.
- Se ha diseñado el concepto inicial del juego, que se ha madurado durante el desarrollo. A partir del concepto inicial se han diseñado la implementación de juego y editor, repartiéndolo de forma equitativa en módulos.
- Este proyecto se ha realizado entre 2 personas, con tareas diferenciadas, pero relacionadas y repartidas equitativamente.
- Se ha diseñado el proyecto fácilmente ampliable, de forma que se pueda continuar trabajando en él, y crear diversidad en los elementos existentes.

A nivel individual:

- Nos hemos documentado sobre las tecnologías Ogre3D, para la visualización, y Bullet Physics, para las simulaciones físicas.
- Hemos diseñado el concepto de juego para Woxfare, con todas las mecánicas, tipos de enemigos, de armas, de ítems y de puntuaciones.
- Se han diseñado e implementado los diferentes menús para acceder y salir del juego y la GUI mostrada durante este.
- Se ha diseñado e implementado el ciclo principal de la partida, dividida en rondas y donde los enemigos aparecen a lo largo de estas, siguiendo el concepto de juego.
- Se ha implementado una cámara en tercera persona con vista desde arriba, que sigue al personaje en todo momento.
- Se ha incorporado la librería Bullet Physics y se ha usado en el movimiento y disparos del juego.
- Se ha incorporado la librería FMOD para el audio y se han usado diferentes sonidos de ambiente y de disparos.
- Hemos desarrollado el trabajo en equipo entre dos personas, y creemos que esto ha sido beneficioso para ambos.
- Se ha valorado el uso de Billboards para simular los disparos y se ha utilizado este método para la implementación.

Como mejoras podemos destacar:

- Añadir diferentes tipos de arma, ítems y enemigos, tanto comportamiento como forma. La forma de los ítems, armas y enemigos es simple. Se podría tratar de implementar alguna forma de añadir complementos a los enemigos, o poder crear la forma del arma a través de cubos. Esta diversificación le daría más versatilidad al juego.
- Añadir y mejorar los efectos visuales y auditivos, permitiendo incluso que sean editables por parte del jugador, para dar más credibilidad a las escenas y capacidad de inmersión.

- Uso de la red, tanto para compartir escenarios como para jugar multijugador. El proyecto pretende ser encarado a compartir escenarios en comunidad para que cualquiera pueda jugar con ellos. El modo multijugador podría dar un punto extra de diversión, no solo en partidas 2 jugadores sino el poder llegar ampliarlo a más.

5 BIBLIOGRAFÍA Y REFERENCIAS

- [1] <http://store.steampowered.com/app/10/?l=spanish>
Store de Steam con información e imágenes sobre Counter Strike 1.6. Junio de 2004
- [2] <http://www.cubecraft.com/> - Junio de 2014
Sitio Web con diseños para construir en papel de personajes famosos en formas cúbicas. Junio de 2014
- [3] <http://www.opengl.org/> - Junio de 2014
Sitio Web oficial de la librería OpenGL.
- [4] <http://www.directx.es/> - Junio de 2014
Sitio Web oficial de la librería OpenGL.
- [5] <https://www.unrealengine.com/blog/unreal-engine-42-release>
Sitio Web oficial que contiene las notas del último release. Junio de 2004
- [6] <https://www.unrealengine.com/products/udk/>
Sitio Web oficial con información sobre el Unreal Development Kit. Junio de 2004
- [7] <http://www.bulletphysics.org/>
Sitio Web oficial de Bullet Physics. Junio de 2004
- [8] <http://www.nvidia.es/object/nvidia-physics-es.html>
Sitio Web oficial de NVidia PhysX. Junio de 2004
- [9] <http://www.ogre3d.org/>
Sitio Web oficial de Ogre3D. Junio de 2004
- [10] <http://www.fmod.org/>
Sitio Web oficial de FMOD. Junio de 2004
- [11] <http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Ogre+Wiki+Tutorial+Framework>
Esqueleto básico de Ogre3D para descargar. Junio de 2004
- [12] <http://es.wikipedia.org/wiki/Singleton>
Explicación del patrón de diseño Singleton en wikipedia. Junio de 2004
- [13] <http://www.blender.org/>
Sitio Web oficial de la herramienta Blender. Junio de 2004
- [14] http://www.ogre3d.org/docs/manual/manual_9.html#The-Mesh-Object
Manual oficial de Ogre3D con información sobre el objeto Malla (Mesh Object). Junio de 2014
- [15] <https://code.google.com/p/blender2ogre/>
Repositorio de código de la herramienta blender2ogre. Junio de 2004

[16] Alejandro Vázquez. (2014). *Woxfare: un videojuego de supervivencia. Diseño e implementación del editor Omega*.

Proyecto de final de carrera de Alejandro Vázquez que complementa el trabajo realizado en este proyecto.

[17] <http://www.ogre3d.org/tikiwiki/tiki-index.php?page=-billboard>

Sitio Web con información sobre los Billboards de Ogre3D. Junio de 2014

[18] <http://www.ogre3d.org/tikiwiki/tiki-index.php?page=OgreBites>

Sitio Web con información y primeros pasos usando OgreBites.

[19] <http://www.ogre3d.org/tikiwiki/tiki-index.php?page=SdkTrays>

Sitio Web con información y primeros pasos usando SDKTrays con Ogre3D.

[20] http://www.ogre3d.org/docs/manual/manual_12.html#SEC15

Manual oficial de Ogre3D con información sobre Overlays. Junio de 2014

[21] <http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Using+OIS>

Sitio Web con información y primeros pasos usando OIS con Ogre3D. Junio de 2014

[22] <http://bulletphysics.org/Bullet/BulletFull/classbtRigidBody.html>

Referencia de la clase btRigidBody de Bullet Physics. Junio de 2014

[23] <http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Intermediate+Tutorial+3>

Manual de referencia para el uso de RayCastQuery de Ogre3D. Junio 2014

[24] <http://www.ogre3d.org/download/tools>

Sitio Web oficial de descarga de Ogre Command-Line Tools. Junio de 2014

[25] <http://www.teamfortress.com/>

Sitio Web oficial del videojuego Team Fortress 2 (TF2). Junio de 2014

[26] <http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Tutorials>

Tutoriales básicos y avanzados de Ogre3D. Junio de 2014

[27] <http://www.ogre3d.org/tikiwiki/tiki-index.php>

Wiki colaborativa de Ogre3D, con manuales detallados y ejemplos prácticos. Junio de 2014

[28] <http://www.ogre3d.org/tikiwiki/QtOgre>

Tutorial para unir Ogre3D con Qt. Junio de 2014

[29] http://bulletphysics.org/mediawiki-1.5.8/index.php/Tutorial_Articles

Tutoriales sobre Bullet Physics. Junio de 2014

[30] <http://c.conclase.net/curso/>

Manual de referencia de C++. Junio de 2014

[31] <http://www.gamasutra.com/>

Noticias y actualidad sobre videojuegos. Junio de 2014

[32] <http://www.gamedev.net/page/index.html>

Trucos y guías de programación sobre videojuegos.