Deusto
Facultad de Ingeniería
Universidad de Deusto
Ingeniaritza Fakultatea
Deustuko Unibertsitatea

Grado en Ingeniería Informática
Informatikako Ingeniaritzako Gradua

Proyecto fin de grado
Gradu amaierako proiektua

Diseño e implantación de un videojuego 2D con niveles creados proceduralmente

Unai Alonso Álvarez

Director: Andoni Eguíluz Morán

Bilbao, julio de 2015

Resumen

Hoy en día, la industria del videojuego o entretenimiento digital está en pleno auge, y es una opción laboral más que válida. Sin embargo, al ser un área con tanta gente aficionada, existen muchos proyectos y empresas dedicadas a ello, y su número está aumentando. Por este motivo, la competitividad es muy grande para los nuevos proyectos que surjan, y es difícil hacerse un hueco en la industria. A pesar de que existen varias ofertas educativas para formarse en el tema y así introducirse en el mercado laboral, los expertos coinciden en que la manera más eficaz de dar el salto es creando juegos. Este proyecto apunta a crear un producto que, además de servir como currículum, pudiera ser comercializable.

Descriptores

Videojuego, SFML, generación procedural, Linux, Action RPG

Índice general

Índice de figuras

Capítulo 1

Capítulo 2

Capítulo 4

Capítulo 5

Capítulo 6

Índice de tablas

Capítulo 2

Índice de algoritmos

Capítulo 5

1. INTRODUCCIÓN

1.1. PRESENTACIÓN DEL DOCUMENTO

El presente informe describe el proyecto de desarrollo de Kingdom of Hatred, un videojuego en dos dimensiones con niveles generados proceduralmente detallando tanto los objetivos que se pretenden alcanzar con el proyecto, como las fases, actividades y recursos necesarios para llevarlo a cabo.

El contenido de este documento se estructura en torno a los siguientes apartados:

- **Introducción:**
  Definición del contenido del documento, resumen del estado del arte y motivación.
- **Objetivos del proyecto:**
  Establecimiento de los objetivos del proyecto, su alcance y las tareas a realizar.
- **Especificación de requisitos:**
  Descripción de los requisitos del proyecto, analizados desde distintos puntos de vista.
- **Especificación del diseño:**
  Descripción de la solución elegida y cómo ha sido aplicada.
- **Consideraciones de la implementación:**
  Descripción de los aspectos más destacables de la implementación.
- **Plan de pruebas:**
  Definición del plan de pruebas utilizado para garantizar la calidad del producto.
- **Manual de usuario:**
  Descripción del uso del producto desarrollado al usuario.
- **Incidencias:**
  Descripción de los problemas encontrados en el desarrollo y cómo de han solucionado.
- **Conclusiones:**
  Análisis de los objetivos alcanzados y consideraciones adicionales.

1.2. ESTADO DEL ARTE Y MOTIVACIÓN

La industria de los videojuegos es el sector económico involucrado en el desarrollo, la distribución, la mercadotecnia, la venta de videojuegos y del hardware asociado. Engloba a docenas de disciplinas de trabajo y emplea a miles de personas alrededor del mundo.

Esta ha mantenido en los últimos años su posición como principal industria de ocio audiovisual e interactivo, con una cuota de mercado muy superior a la del cine y la música. Por esta razón, enfocar una carrera profesional orientada al desarrollo de videojuegos es cada vez una opción más viable y habitual. En el pasado el desarrollo de videojuego era algo casi exclusivo de grandes empresas, ya que las herramientas necesarias para ello eran escasas y no estaban al alcance de todos. Sin embargo,

en los últimos años estas se han hecho mucho más accesibles y eso ha hecho que aparezcan muchos desarrolladores nuevos, con lo cual la competitividad en el mercado es bastante grande.

Motores de videojuegos

Un motor de videojuego es un término que hace referencia a una serie de librerías de programación que permiten el diseño, la creación y la representación de un videojuego. Del mismo modo existen motores de juegos que operan tanto en consolas de videojuegos como en sistemas operativos. La funcionalidad básica de un motor es proveer al videojuego de un motor de renderizado para los gráficos 2D y 3D, motor físico o detector de colisiones, sonidos, scripting, animación, inteligencia artificial, redes, streaming, administración de memoria y un escenario gráfico.

Entre los motores disponibles, se pueden diferenciar dos tipos: los que ofrecen editor visual y los que no.

Los primeros son los más conocidos y utilizados por su sencillez y curva de aprendizaje. Estos motores habitualmente dan una base funcional para un juego, en el que la arquitectura del software ya está definida, y el usuario se limita a añadir elementos al juego, delegando en el motor el cometido de gestionarlos. Consecuentemente, estos motores son muy flexibles y pueden ser utilizados para crear prácticamente cualquier tipo de juego, pero tiene la debilidad de que la eficiencia no es siempre la máxima. Ejemplos de este tipo de motores podrían ser: Unity3D, Unreal Engine, CryEngine... Dado que uno de los objetivos fundamentales del proyecto es aprender a desarrollar videojuegos, se ha decidido no utilizar un motor de este tipo, ya que se pierde en gran medida el control sobre el software.

Por otro lado existen otros motores que no cuentan con editor visual, de forma que en términos sencillos proveen al usuario con una API para lo necesario en la realización de un videojuego. Estas alternativas conllevan un aumento muy grande de la carga de trabajo, pero tienen la ventaja de que los desarrolldores tienen el control total de lo que ocurre en cada parte del software, y al no tener una arquitectura preestablecida, pueden adaptarla al juego que se está creando, de forma que se consiga la mayor eficiencia. Entre los motores disponibles de este tipo se pueden nombrar SFML, SDL o libGDX. Para este proyecto se ha decidido usar un motor de este tipo, en concreto SFML.

Desarrollos AAA y desarrollos independientes

En la industria del desarrollo de videojuegos, se conocen como AAA o triple A los desarrollos que cuentan con los más altos niveles de presupuesto y promoción para su realización. Por tanto, se espera que un título considerado AAA sea de una gran calidad y que sus ventas sean inmensas. Estos desarrollos son llevados a cabo por empresas de la industria ya establecidas como EA [5], Ubisoft [14], Valve [15] y otras, aunque también existen casos de desarrollos de este tipo en empresas jóvenes gracias a la presencia de un inversor.

Por otro lado, existen los desarrollos independientes, los cuales están en pleno auge desde hace unos pocos años. Estos cuenta con un presupuesto mucho más limitado, y la fuerza de producción suele ser la de un equipo de pocas personas. Actualmente, estos equipos se basan en créditos para llevar a cabo sus proyectos, aunque últimamente la financiación colectiva está cogiendo mucha fuerza. Los equipos están formados tanto por profesionales de la industria que han querido dar rienda

suelta a su creatividad con un proyecto propio como por aficionados a videojuegos que aspiran a poder trabajar en el sector. Uno de los grandes problemas de los desarrollos independientes es que, al haber muchísismos equipos de desarrollo, la competitividad es grandísima, y solo unos pocos consiguen hacerse un hueco a un nivel más profesional.

RPGs de acción

El juego que se va a desarrollar pertenece al género de los juegos de rol de acción. Estos juegos comparten muchas características con los juegos de rol tradicionales, pero su pricipal diferencia está en que ofrecen combates en tiempo real. Se ha elegido este género porque, además de que el alumno es un gran aficionado, supone un reto extra a la hora de desarrollar, ya que es necesario que la aplicación tenga en todo momento un respuesta en tiempo real y se han de gestionar adecuadamente todas las entidades para conseguirlo.

La principal inspiración del juego ha sido la saga *The Legend of Zelda* [13], en particular la última entrega que se lanzó para GBA, subtitulada *The Minish Cap*. Como otras influencias se pueden mencionar *Hyper Light Drifter*, *Sword of Mana* o *Hero Siege*. En los últimos años este género ha ganado mucha fuerza, en gran parte gracias al más que notable salto que dió a las tres dimensiones, el cual atrajo a muchos nuevos jugadores. Sin embargo, los juegos en dos dimensiones han quedado ligeramente relegados, y lo más habitual es que sean desarrollados por empresas independientes o para consolas portátiles.

Figura 1.1: Pantalla de inicio de *The Legend of Zelda The Minish Cap*

Motivación

Este proyecto nace de la afición a los productos de entretenimiento digital y a la creación de los mismos, en concreto, al género de los juegos en dos dimensiones. El proyecto va a consistir en el desarrollo de una demo técnica de un juego de este tipo, desde el análisis de requisitos, diseño del juego y del software y su implementación. Además, los niveles del juegos tendrán que ser generados proceduralmente, así que se deberán implementar algoritmos adecuados para estos propósitos, junto con los demás requisitos típicos de un software tradicional (usabilidad, estabilidad...). El resultado principal consistirá en una demo de calidad, especialmente en el apartado de software, que pudiese

competir con productos similares del mercado, además de servir como experiencia de aprendizaje para el desarrollo de futuros proyectos de esta índole.

2. OBJETIVOS DEL PROYECTO

2.1. VISIÓN GENERAL

En esta sección se pueden ver las distintas partes que componen el capítulo.

- **Visión general:**
  Definición del contenido del capítulo.
- **Definición del proyecto:**
  Establecimiento de los objetivos del proyecto y su alcance.
- **Producto final:**
  Descripción funcional del producto final.
- **Descripción de la realización:**
  Descripción del método de desarrollo que se ha utilizado, especificación de productos intermedios, todas las tareas del proyecto y su asociación con los perfiles del equipo incluyendo EDT.
- **Planificación:**
  Estimación de cargas por perfil y grafos para la representación de la planificación de tareas.
- **Presupuesto:**
  Definición del presupuesto necesario para la realización del proyecto.

2.2. DEFINICIÓN DEL PROYECTO

2.2.1. Objetivos

El objetivo principal de este proyecto es conseguir un producto jugable y estable. Por tanto, se deben llevar a cabo un análisis de requisitos, diseño e implementación. El juego no debe estar terminado, pero es necesario que las características principales estén implementadas y pulidas en una demo técnica. Por otro lado, la arquitectura del software debe estar preparada para ser fácilmente escalable y ampliable. En términos de la industria, el hecho de preparar un segmento del juego completamente se conoce como "corte vertical".

En cuanto a los objetivos secundarios del proyecto, el primero es puramente didáctico. Debido a la naturaleza de este tipo de software, el cual debe tener una respuesta en tiempo real, estable y funcionar en una gama grande de hardware, el software debe estar construido de manera específica, y el objetivo es aprender a crear arquitecturas aptas para este tipo de aplicaciones. En segundo lugar, se quieren aprender y aplicar técnicas y patrones de diseño software conveniente en este ámbito. Por último, se quiere realizar un producto que pudiera ser desarrollado hasta el punto de ser comercial.

2.2.2. Alcance

Atendiendo a las premisas señaladas anteriormente, las funcionalidades que deberá soportar Kingdom of Hatred serán:

- Una arquitectura de software que permita añadir, eliminar y modificar elementos fácilmente.
- Niveles generados proceduralmente.
- Una interfaz gráfica de usuario que permita acceder a las partidas y mostrar controles.
- Una experiencia de juego pulida.

2.3. PRODUCTO FINAL

El producto final será una demo técnica, en la cual deberán estar implementadas las funcionalidades principales del juego. Al iniciarse el juego se mostrará el menú principal, el cual contará con las siguientes opciones:

- **Play:**
  Ejecutará el juego, el cual tendrá las siguientes características:
  - El mapa del juego será generado aleatoriamente.
  - El mapa estará poblado de enemigos, cuya localización también será aleatoria. Habrá distintos tipos de enemigos, y todos intentarán dañar al jugador.
  - El jugador podrá moverse, realizar un ataque básico cuerpo a cuerpo y dispondrá de dos habilidades. Todos estos mecanismos serán los que utilizará para acabar con los enemigos.
  - El juego finaliza cuando el jugador es abatido o cuando todos los monstruos son derrotados.
- **Credits:**
  Mostrará una pantalla estática con la información referente a los desarrolladores y los orígenes de los gráficos y sonidos utilizados.
- **Exit:**
  Cerrará la aplicación.

2.4. DESCRIPCIÓN DE LA REALIZACIÓN

2.4.1. Método de desarrollo

Kingdom of Hatred se desarrollará mediante un sistema iterativo e incremental. Este proceso de desarrollo suple las carencias del modelo de cascada, el modelo tradicional que establece una rigorosa jerarquía en las fases del desarrollo y requiere completar una fase para comenzar la siguiente. En la Figura 2.1 se puede observar el diagrama del modelo incremental.

El desarrollo incremental permite desarrollar una parte funcional del proyecto en cada etapa, reservando la mejora o extensión de funcionalidades para el futuro y por tanto controlando la complejidad y los riesgos. Además, este sistema permite a los desarrolladores aprovechar conocimiento adquirido en etapas previas e incorporar nuevo conocimiento y nuevas técnicas en fases venideras.

Figura 2.1: Modelo incremental

Adicionalmente, esta metodología confía en el desarrollo guiado por tests. Esta práctica consiste en el desarrollo de tests antes que código, y después se genera el mínimo código posible para completar esos tests. El objetivo de esta metodología es lograr código limpio y funcional, la idea es que los requisitos se traducen a evidencia, de forma que si los tests se completan satisfactoriamente, se garantiza que el software cubre dicho requisitos.

2.4.2. Tareas principales

El desarrollo de Kingdom of Hatred comprende las siguientes tareas o actividades:

Lanzamiento del proyecto

- **Organización**
  Actividad mediante la que se define y prepara la planificación, asignación de misiones y el lanzamiento del proyecto y sus sucesivas fases.
- **Seguimiento**
  Realización del seguimiento y control del desarrollo del proyecto, que permita la rápida detención y solución de problemas que puedan dificultar la buena marcha del mismo.

Análisis de herramientas y técnicas

- **Análisis de herramientas para desarrollo de juegos**
  Investigar distintas alternativas que existen para el desarrollo de juegos.
- **Análisis de técnicas de generación procedural** Investigar distintos algoritmos de generación procedural que sean adecuados para la generación de niveles.

Diseño del juego

- Diseño del juego Crear el documento de desarrollo de juego que definirá cómo será este.

Desarrollo del software

- **Formación** Aprendizaje en la creación de juegos.

- **Diseño** Diseño del software del juego.
- **Implementación** Implementación del juego.

Validación técnica y de usabilidad:

- **Betatesting** Uso intensivo del juego en busca de bugs.
- **Prueba de experiencia de usuario** Recoger opiniones para mejorar la experiencia de usuario.

Distribución y cierre del proyecto

- **Despliegue de la versión final**
  Preparar el juego para el usuario final.
- **Cierre del proyecto**
  Cierre del proyecto.

2.4.3. Productos intermedios

Los productos intermedios que se generarán en cada una de las fases son:

- **Diseño del juego:**
  - Documento de diseño de juego.
- **Desarrollo del software:**
  - Documento con la especificación del software.
- **Validación técnica y usabilidad:**
  - Informe de evaluación del juego.

2.5. ORGANIZACIÓN

2.5.1. Esquema organizativo

La organización del proyecto se articula en torno al comité de dirección y al equipo de trabajo que se va a encargar de desarrollar el producto, en función de la estructura de la Figura 2.2.

- **Comité de dirección**
  Su función principal es asesorar el proyecto y la toma de decisiones. Formado por el jefe de proyecto.
- **Equipo de trabajo**
  El órgano encargado de diseñar, desarrollar y testear el contenido del proyecto en función de las diferentes fases estipuladas. Formado por el programador, el diseñador y el tester.

2.5.2. Plan de recursos humanos

El equipo de trabajo estará formado por los siguientes perfiles directamente relacionados con las diferentes áreas de competencias que abordan el proyecto:

Figura 2.2: Esquema organizativo

- **Jefe de proyecto:** Sus funciones son realizar las actividades de organización, coordinación y seguimiento del proyecto. Por otro lado, deberá encargarse del diseño del juego. Con un jefe de proyecto es suficiente. Para el perfil será necesario alguien con experiencia en comunicación interpersonal, liderazgo y solución de problemas.
- **Programador:** Encargado de diseñar e implementar el software en base al diseño del juego. Será necesario conocimiento de ingeniería del software para llevar a cabo las tareas asignadas a este perfil.
- **Tester:** Encargado de probar el producto y reportar errores y sugerir mejoras. Para hacer las pruebas de fase beta será suficiente con un tester. Sin embargo, para las prueba de experiencia de usuario sería conveniente contar con el mayor número de personas posibles. El tester debe ser tener imaginación y ser creativo para llevar al juego a situaciones límite y encontrar el mayor número de errores posible.

2.6. CONDICIONES DE EJECUCIÓN

2.6.1. Entorno de trabajo

El lugar de trabajo habitual será el domicilio del trabajador. El calendario y horario serán 3 horas al día, de lunes a viernes. Los medios informáticos para la ejecución corren a cargo del trabajador. Los medios de los que se disponen actualmente y se usarán son los siguientes:

- **Hardware**
  - Ordenador de sobremesa completo con monitor secundario
  - Ordenador portátil
- **Software**
  - Todo el software que se use será gratuito, así que este apartado no supondrá un problema.

2.6.2. Diálogo durante el proyecto

Durante el proyecto y para la correcta marcha del mismo, una comunicación constante entre el comité de dirección del proyecto y el equipo de desarrollo va a ser necesaria, y por tanto realizada.

El equipo de desarrollo deberá atender a todas la reuniones para discutir el progreso, las mejoras y los requisitos del proyecto.

2.6.3. Recepción de productos

Los productos generados durante el proyecto son el punto de partida para trabajo venidero, y deben ser enviados al jefe de proyecto para evaluación y aprobado.

Tras el envío, habrá 3 días laborables para comunicar cualquier error o comentario al equipo de trabajo, el cual procederá a crear una revisión del documento y comenzará un nuevo periodo de aprobación. Si pasados los 3 días laborables no se ha recibido respuesta, la entrega se considerará válida.

Durante el desarrollo de cualquiera de los productos del proyecto, el jefe de proyecto asume la responsabilidad si cualquiera de ellos no se hace o no alcanza la calidad esperada. Consecuentemente, el jefe de proyecto debe cercionarse que el trabajo de todos marcha adecuadamente y tomar medidas correctoras en caso contrario.

El jefe de proyecto será el portavoz del equipo de desarrollo durante la marcha del proyecto, y tendrá que comunicarse con el tutor e informarle adecuadamente del estado de la realización en cada momento.

2.6.4. Control de cambios

Tanto jefe de proyecto como el tutor tienen potestad para pedir modificaciones en las especificaciones, diseños o desarrollos ya realizados. En caso de querer modificar algo, se seguirá el siguiente procedimiento:

1. Comunicación formal del solicitante de las modificaciones solicitadas.
2. Valoración, por parte del jefe del proyecto y el equipo de desarrollo, de la repercusión técnica, económica y el plazo de ejecución.
3. Presentación de una propuesta valorada al solicitante.
4. Notificación, por parte del solicitante, de la aprobación o no de la propuesta.
5. En caso afirmativo, modificación del plan de trabajo y presupuesto.

2.7. PLANIFICACIÓN

2.7.1. Plan de trabajo

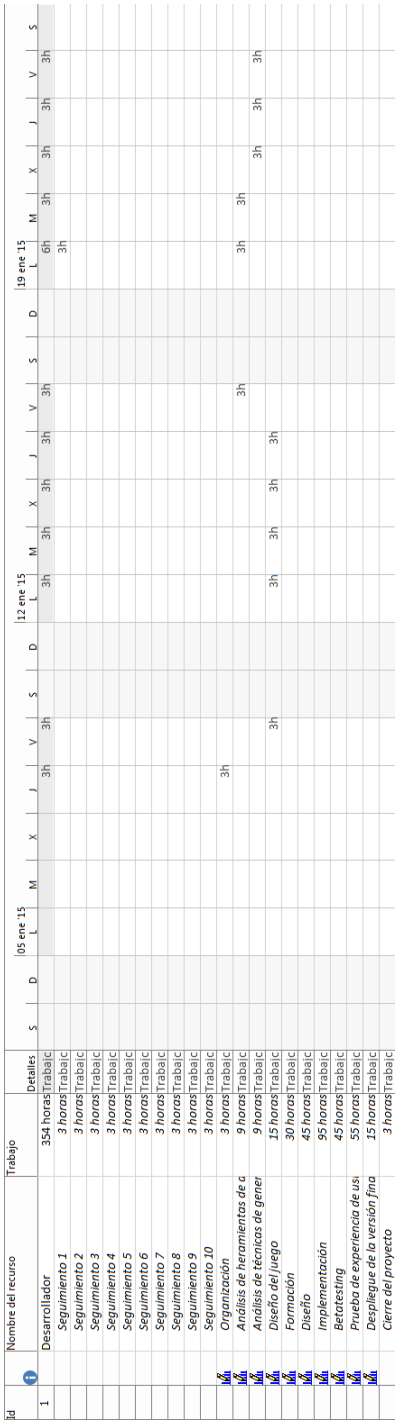
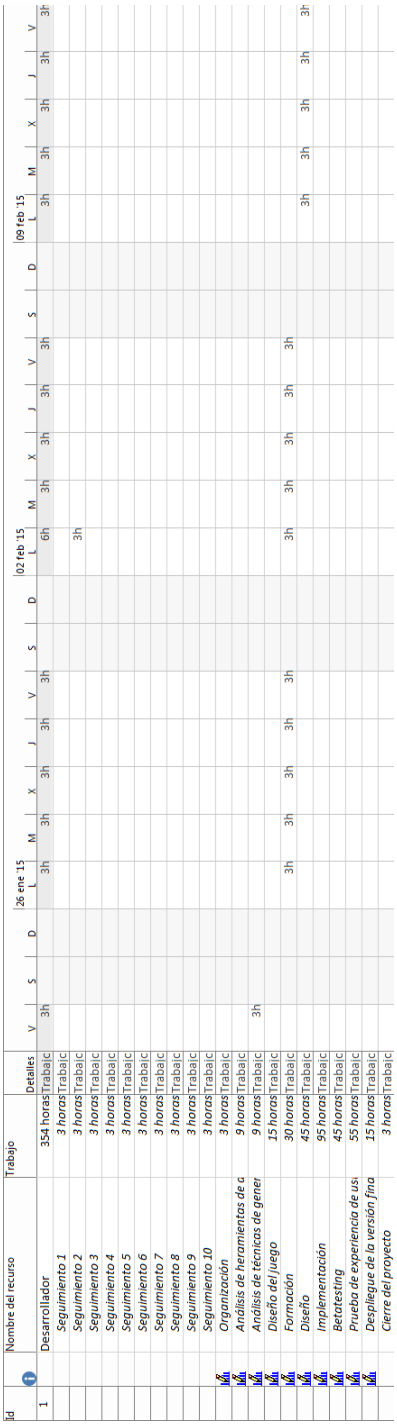


Figura 2.3: Diagrama del plan de trabajo 1



Id	Nombre del recurso	Trabajo	Detalles	V	S	D	26 ene '15 L	M	X	J	V	S	D	02 feb '15 L	M	X	J	V	S	D	09 feb '15 L	M	X	J	V
1	Desarrollador	354 horas	Trabaj	3h			3h	3h	3h	3h	3h			6h	3h	3h	3h	3h	3h		3h	3h	3h	3h	3h
	Seguimiento 1	3 horas	Trabaj																						
	Seguimiento 2	3 horas	Trabaj											3h											
	Seguimiento 3	3 horas	Trabaj																						
	Seguimiento 4	3 horas	Trabaj																						
	Seguimiento 5	3 horas	Trabaj																						
	Seguimiento 6	3 horas	Trabaj																						
	Seguimiento 7	3 horas	Trabaj																						
	Seguimiento 8	3 horas	Trabaj																						
	Seguimiento 9	3 horas	Trabaj																						
	Seguimiento 10	3 horas	Trabaj																						
	Organización	3 horas	Trabaj																						
	Análisis de herramientas de d	9 horas	Trabaj																						
	Análisis de técnicas de gener	9 horas	Trabaj		3h																				
	Diseño del juego	15 horas	Trabaj																						
	Formación	30 horas	Trabaj				3h	3h	3h	3h	3h			3h	3h	3h	3h	3h							
	Diseño	45 horas	Trabaj																		3h	3h	3h	3h	3h
	Implementación	95 horas	Trabaj																						
	Betatesting	45 horas	Trabaj																						
	Prueba de experiencia de usu	55 horas	Trabaj																						
	Despliegue de la versión fina	15 horas	Trabaj																						
	Cierre del proyecto	3 horas	Trabaj																						

Figura 2.4: Diagrama del plan de trabajo 2

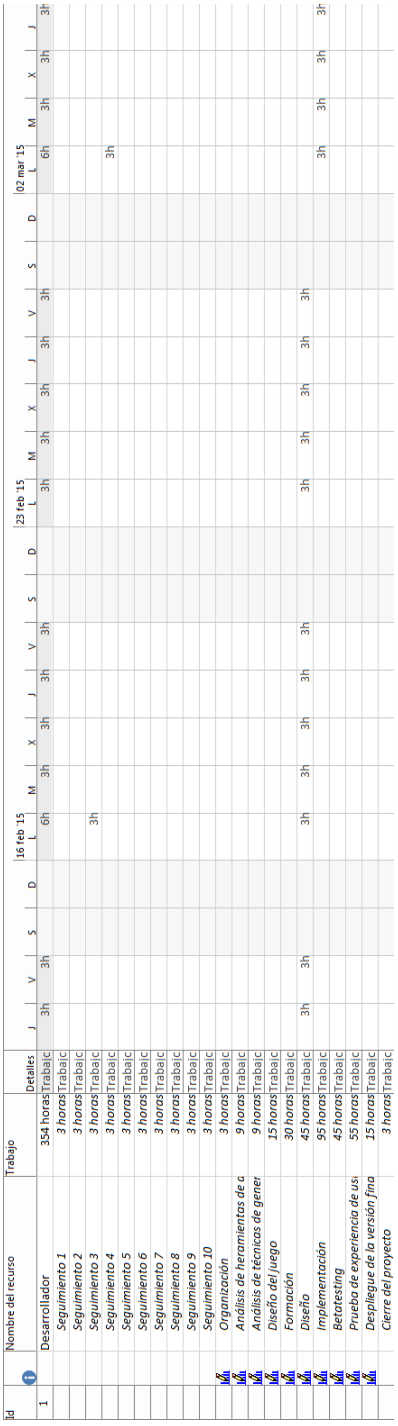
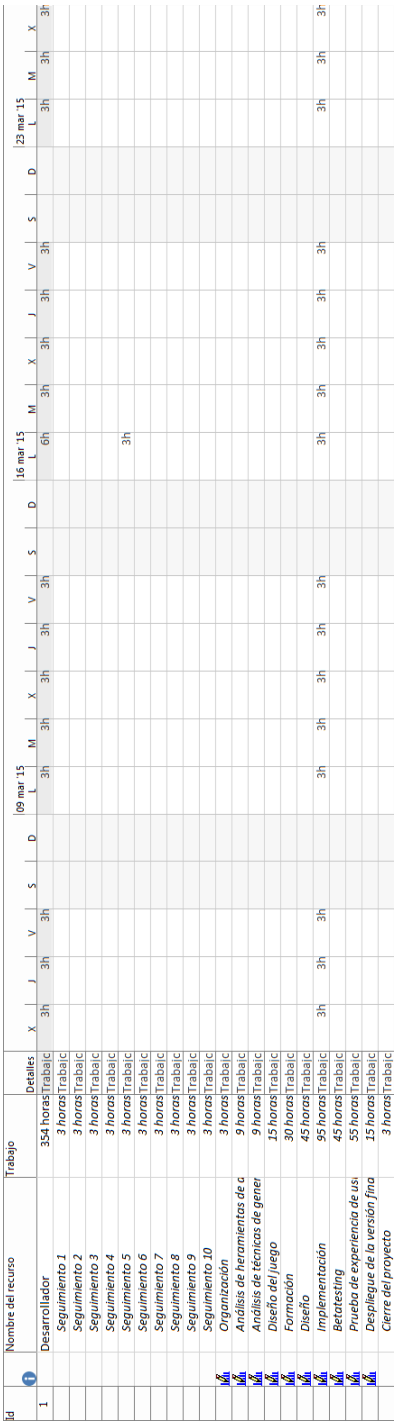
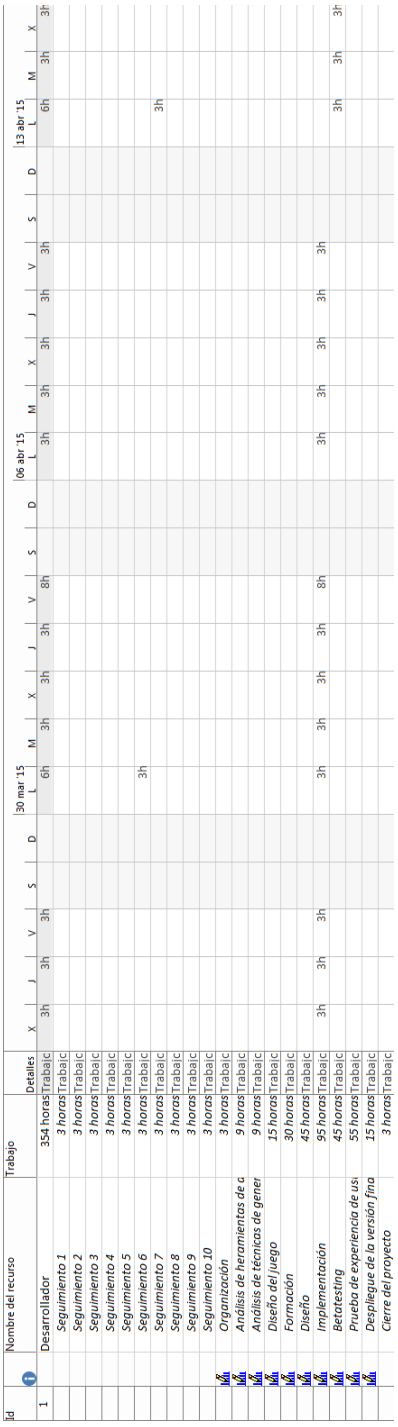


Figura 2.5: Diagrama del plan de trabajo 3



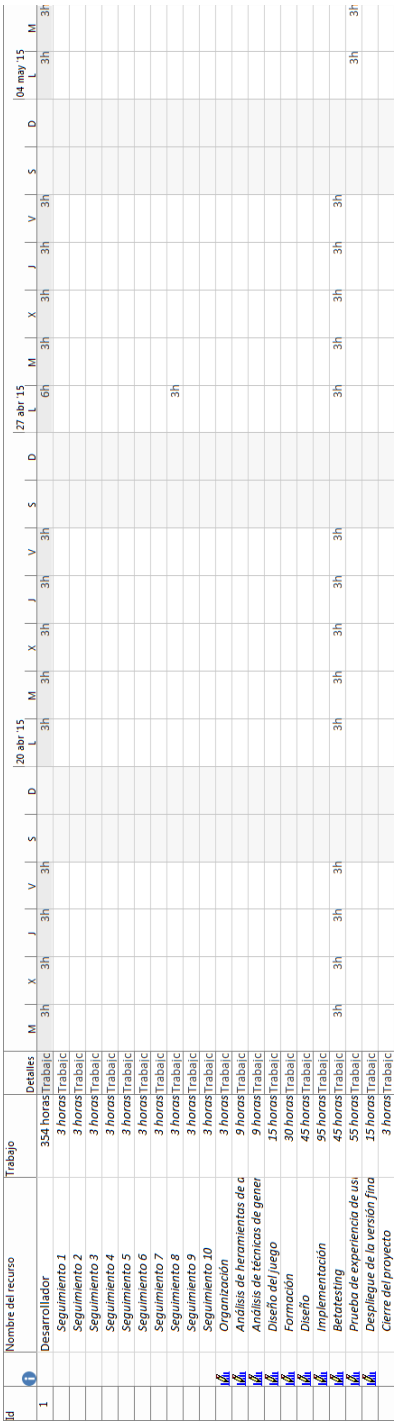
Id	Nombre del recurso	Trabajo	Detalles	X	J	V	S	D	09 mar '15 L	M	X	J	V	S	D	16 mar '15 L	M	X	J	V	S	D	23 mar '15 L	M	X
1	Desarrollador	354 horas	Trabajo	3h	3h	3h			3h	3h	3h	3h	3h			6h	3h	3h	3h	3h			3h	3h	3h
	Seguimiento 1	3 horas	Trabajo																						
	Seguimiento 2	3 horas	Trabajo																						
	Seguimiento 3	3 horas	Trabajo																						
	Seguimiento 4	3 horas	Trabajo													3h									
	Seguimiento 5	3 horas	Trabajo																						
	Seguimiento 6	3 horas	Trabajo																						
	Seguimiento 7	3 horas	Trabajo																						
	Seguimiento 8	3 horas	Trabajo																						
	Seguimiento 9	3 horas	Trabajo																						
	Seguimiento 10	3 horas	Trabajo																						
	Organización	3 horas	Trabajo																						
	Análisis de herramientas de d	9 horas	Trabajo																						
	Análisis de técnicas de gener	9 horas	Trabajo																						
	Diseño del juego	15 horas	Trabajo																						
	Formación	30 horas	Trabajo																						
	Diseño	45 horas	Trabajo																						
	Implementación	95 horas	Trabajo	3h	3h	3h			3h	3h	3h	3h	3h			3h	3h	3h	3h	3h			3h	3h	3h
	Betatesting	45 horas	Trabajo																						
	Prueba de experiencia de usu	55 horas	Trabajo																						
	Despliegue de la versión fina	15 horas	Trabajo																						
	Cierre del proyecto	3 horas	Trabajo																						

Figura 2.6: Diagrama del plan de trabajo 4



Id	Nombre del recurso	Trabajo	Detalles	X	J	V	S	D	30 mar '15 L	M	X	J	V	S	D	06 abr '15 L	M	X	J	V	S	D	13 abr '15 L	M	X
1	Desarrollador	354 horas	Trabajo	3h	3h	3h			6h	3h	3h	3h	8h			3h	3h	3h	3h	3h			6h	3h	3h
	Seguimiento 1	*3 horas*	Trabajo																						
	Seguimiento 2	*3 horas*	Trabajo																						
	Seguimiento 3	*3 horas*	Trabajo																						
	Seguimiento 4	*3 horas*	Trabajo																						
	Seguimiento 5	*3 horas*	Trabajo																						
	Seguimiento 6	*3 horas*	Trabajo						3h																
	Seguimiento 7	*3 horas*	Trabajo																				3h		
	Seguimiento 8	*3 horas*	Trabajo																						
	Seguimiento 9	*3 horas*	Trabajo																						
	Seguimiento 10	*3 horas*	Trabajo																						
	Organización	*3 horas*	Trabajo																						
	Análisis de herramientas de d	*9 horas*	Trabajo																						
	Análisis de técnicas de gener	*9 horas*	Trabajo																						
	Diseño del juego	*15 horas*	Trabajo																						
	Formación	*30 horas*	Trabajo																						
	Diseño	*45 horas*	Trabajo																						
	Implementación	*95 horas*	Trabajo	3h	3h	3h			3h	3h	3h	3h	8h			3h	3h	3h	3h	3h			3h	3h	3h
	Betatesting	*45 horas*	Trabajo																						
	Prueba de experiencia de usu	*55 horas*	Trabajo																						
	Despliegue de la versión fina	*15 horas*	Trabajo																						
	Cierre del proyecto	*3 horas*	Trabajo																						

Figura 2.7: Diagrama del plan de trabajo 5



Id	Nombre del recurso	Trabajo	Detalles	M	X	J	V	S	D	20 abr '15 L	M	X	J	V	S	D	27 abr '15 L	M	X	J	V	S	D	04 may '15 L	M
1	Desarrollador	354 horas	Trabajo	3h	3h	3h	3h			3h	3h	3h	3h	3h			6h	3h	3h	3h	3h			3h	3h
	Seguimiento 1	3 horas	Trabajo																						
	Seguimiento 2	3 horas	Trabajo																						
	Seguimiento 3	3 horas	Trabajo																						
	Seguimiento 4	3 horas	Trabajo																						
	Seguimiento 5	3 horas	Trabajo																						
	Seguimiento 6	3 horas	Trabajo																						
	Seguimiento 7	3 horas	Trabajo																						
	Seguimiento 8	3 horas	Trabajo														3h								
	Seguimiento 9	3 horas	Trabajo																						
	Seguimiento 10	3 horas	Trabajo																						
	Organización	3 horas	Trabajo																						
	Análisis de herramientas de d	9 horas	Trabajo																						
	Análisis de técnicas de gener	9 horas	Trabajo																						
	Diseño del juego	15 horas	Trabajo																						
	Formación	30 horas	Trabajo																						
	Diseño	45 horas	Trabajo																						
	Implementación	95 horas	Trabajo																						
	Betatesting	45 horas	Trabajo	3h	3h	3h	3h			3h	3h	3h	3h	3h			3h	3h	3h	3h	3h				
	Prueba de experiencia de usu	55 horas	Trabajo																					3h	3h
	Despliegue de la versión fina	15 horas	Trabajo																						
	Cierre del proyecto	3 horas	Trabajo																						

Figura 2.8: Diagrama del plan de trabajo 6

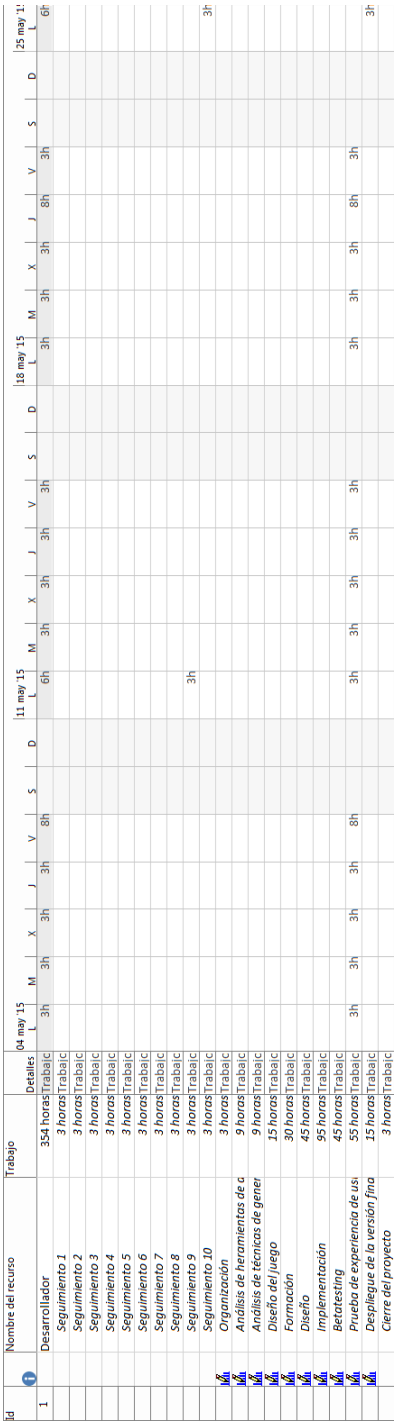


Figura 2.9: Diagrama del plan de trabajo 7

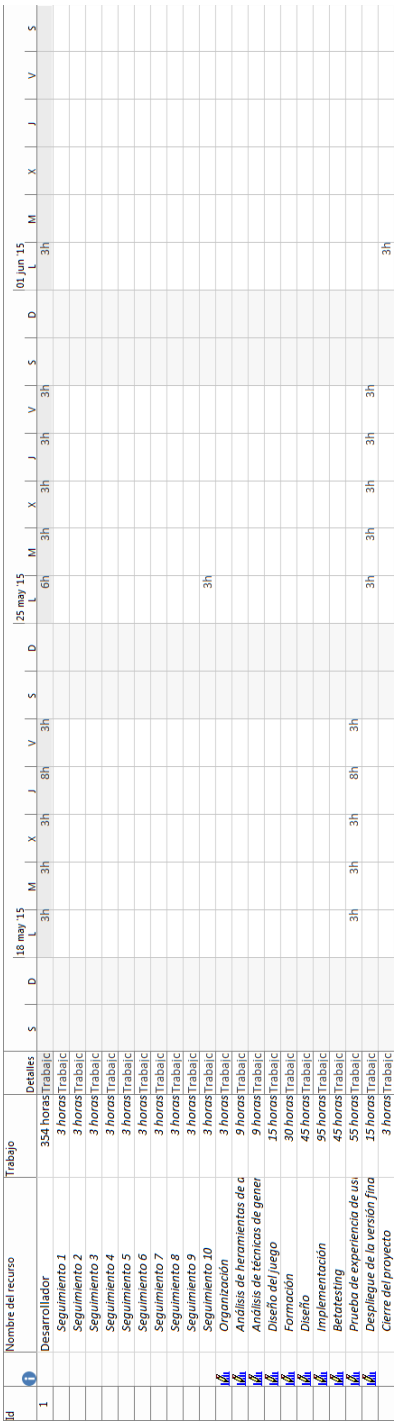


Figura 2.10: Diagrama del plan de trabajo 8

2.7.2. Diagrama de Gantt

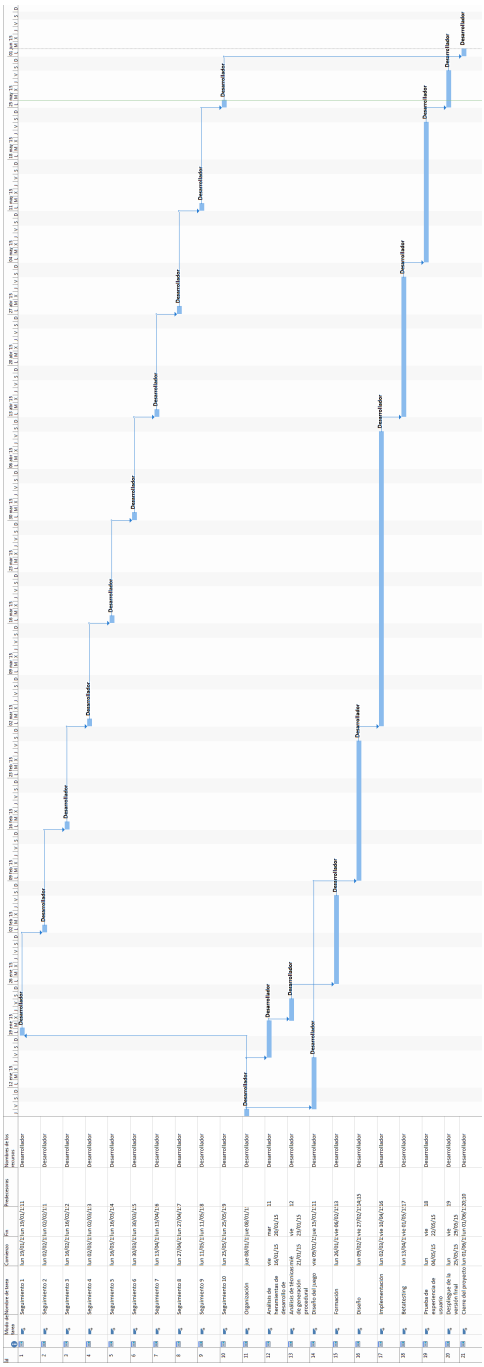


Figura 2.11: Diagrama de Gantt

2.7.3. Diagrama de precedencias

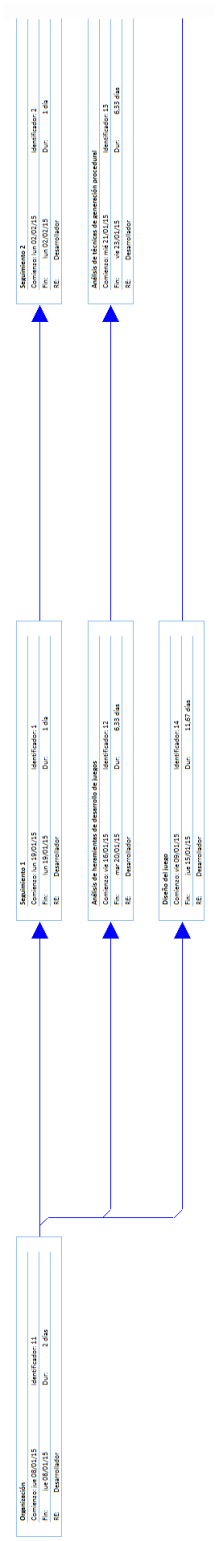


Figura 2.12: Diagrama de precedencias 1

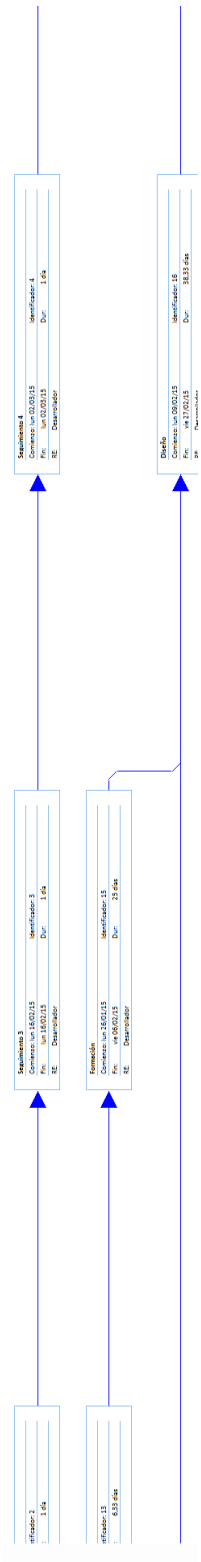


Figura 2.13: Diagrama de precedencias 2

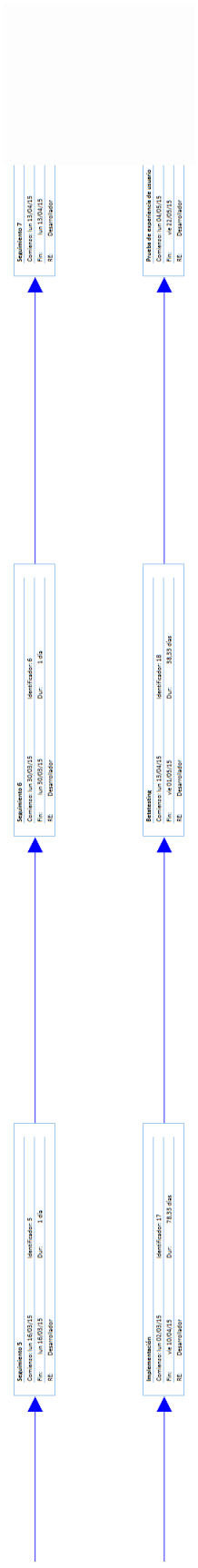


Figura 2.14: Diagrama de precedencias 3

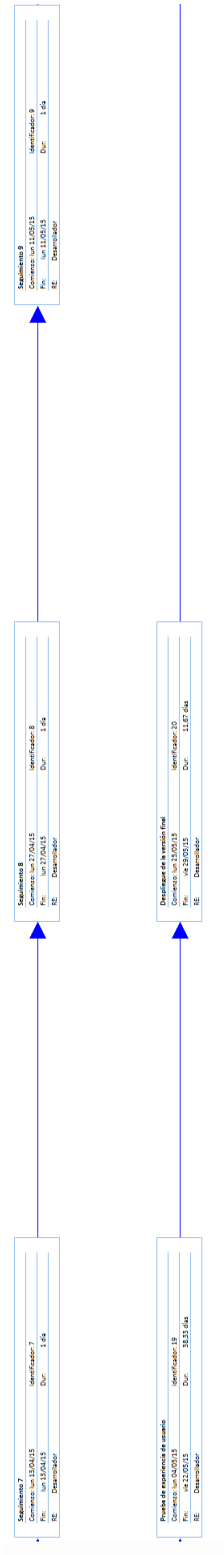


Figura 2.15: Diagrama de precedencias 4

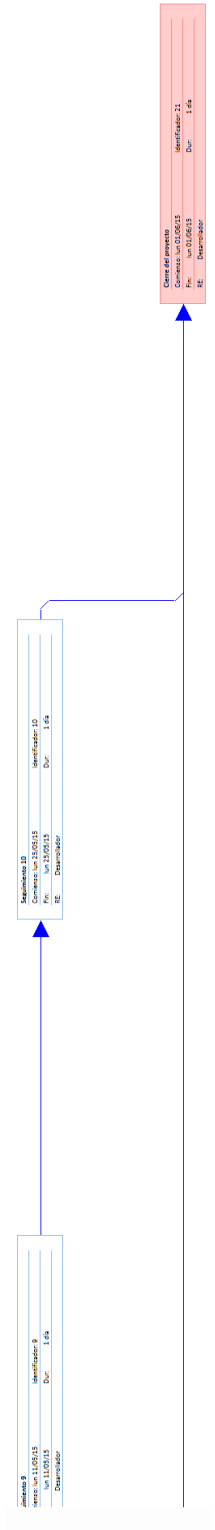


Figura 2.16: Diagrama de precedencias 5

2.8. PRESUPUESTO

En esta sección se detalla el coste planificado del proyecto, calculando primero los costes individuales de recursos hardware, software y humanos. Para terminar, se calcula el coste total del proyecto sumando los tres costes mencionados.

2.8.1. Software

Todas las herramientas utilizadas para la realización del proyecto han sido de código abierto o gratuitas, de forma que el coste derivado de la adquisición de productos o servicios software es nulo.

2.8.2. Hardware

En la tabla 2.1 se detalla el gasto derivado de los dispositivos hardware necesarios para el proyecto.

Tabla 2.1: Presupuesto de hardware

Nombre	*Precio(€)*	*Unidades*	*Importe total(€)*
Ordenador de sobremesa	1.500,00	1	1.500,00
Ordenador portátil	700,00	1	700,00

2.8.3. Recursos humanos

En la tabla 2.2 se detallan los honorarios por cada uno de los roles, en función de las horas trabajadas.

Tabla 2.2: Presupuesto de recursos humanos

Rol	*Precio/hora(€/h)*	*Carga de trabajo(h)*	*Importe total(€)*
Jefe de proyecto	30,00	54	1.620,00
Programador	25,00	321	8.025,00
Tester	15,00	90	1.350,00

2.8.4. Total

Finalmente, en la tabla 2.3 se detalla el presupuesto total del proyecto.

Tabla 2.3: Presupuesto total

Tipo	*Total*
Recursos Humanos	10.995,00
Recursos Hardware	2.200,00
Total	**13.195,0**

3. ESPECIFICACIÓN DE REQUISITOS

3.1. VISIÓN GENERAL

En este capítulo se especifican los requisitos que deben ser cumplidos por el proyecto a desarrollar. Para un mejor entendimiento de los mismos, se han divididos en dos bloques:

- **Requisitos funcionales:**
  Funcionamiento que el juego debe proveer.
- **Requisitos no funcionales:**
  Requisitos relacionados con la usabilidad, el entorno y el rendimiento.

3.2. REQUISITOS FUNCIONALES

Requisitos funcionales que describen el funcionamiento del producto. A continuación se muestran los requisitos del juego:

- **RF0**
  El nivel que constará la demo debe ser generado proceduralmente en cada sesión y debe poblarse de enemigos.
- **RF1**
  El juego debe ser capaz de detectar cuando distintas entidades colisionan entre ellas o con el escenario y actuar en consecuencia.
- **RF2**
  La cámara del juego debe seguir al jugador en los ejes X e Y.
- **RF3**
  El juego debe detectar las entradas del jugador, comprobar el estado actual del mismo y actuar en consecuencia.
- **RF4**
  El juego debe detectar cuando se han cumplido las condiciones para fin de partida y terminarla.
- **RF5**
  Cada enemigo debe tener una inteligencia artificial distinta.
- **RF6**
  El jugador debe tener a su disposición tres tipos de ataque.
- **RF7**
  El juego debe contar con una pantalla para mostrar los controles.

3.3. REQUISITOS NO-FUNCIONALES

Los requisitos no funcionales describen características requeridas del sistema, el proceso de desarrollo o cualquier otro aspecto que tenga alguna restricción.

3.3.1. Usabilidad

Estos requisitos permiten que el juego cumpla las espectativas del usuario final.

- **RU0**
  Durante las partidas el juego debe mostrar en todo momento los datos relevantes al usuario mediante una interfaz gráfica.
- **RU1**
  La interfaz gráfica debe estar diseñada de forma que no sea molesta para el jugador.
- **RU2**
  El juego debe controlarse únicamente con el teclado.

3.3.2. Entorno

Estos requisitos definen el entorno de uso del juego.

- **RE0**
  El código del juego debe ser multiplataforma, de forma que pueda compilarse en Windows, Linux y OSX.
- **RE1**
  La versión final debe incluir las librerías necesarias para el correcto funcionamiento del juego.

3.3.3. Rendimiento

Requisitos relacionados con el tiempo de realización de las tareas de la aplicación, márgenes de error...

- **RR0**
  El juego debe funcionar a un mínimo de 30 FPS en cualquier plataforma.
- **RR1**
  Durante las partidas, no debe haber errores que provoquen un final inesperado de la aplicación.

3.4. CRITERIOS DE VALIDACIÓN

Los requisitos previamente mencionados están sujetos a procesos de validación antes de la entrega final del proyecto. Para comprobar el cumplimiento de los requisitos, el producto final es contrastado con los requisitos iniciales, estudiando los cambios que pudieran surgir. El método de desarrollo está guiado por pruebas, de forma que el cumplimiento exitoso de dichas pruebas validará los distintos requisitos del sistema objetivamente, mientras que si las pruebas no se ejecutan exitosamente indicarán la existencia de requisitos incompletos.

El grado de incumplimiento del proyecto estará directamente relacionado con el porcentaje de requisitos cumplidos, evaluando el grado de completitud objetivamente.

De la misma forma, cualquier implementación que mejore la estabilidad o funcionalidad del sistema que no esté reflejado en los requisitos iniciales se considerará una parte extra de la evaluación del proyecto por el director del mismo.

4. ESPECIFICACIÓN DEL DISEÑO

4.1. VISIÓN GENERAL

En este capítulo se describe el trabajo de diseño realizado para el proyecto, así como el entorno y las herramientas que han sido utilizadas para llevarlo a cabo.

4.2. ARQUITECTURA

La arquitectura básica del juego es la que se muestra en la siguiente imagen. El usuario genera entradas al juego mediante el teclado, después estas entradas son procesadas por la lógica del juego y se realizan las acciones pertinentes en función del estado del juego. Una vez actualizado el estado del juego ha sido actualizado, los módulos de audio y vídeo generan las salidas correspondientes y las muestran mediante la pantalla y el dispositivo reproductor de audio actual.

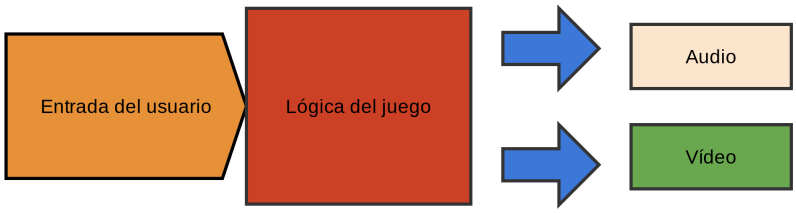


Figura 4.1: Diagrama de arquitectura

4.3. DIAGRAMA DE CLASES

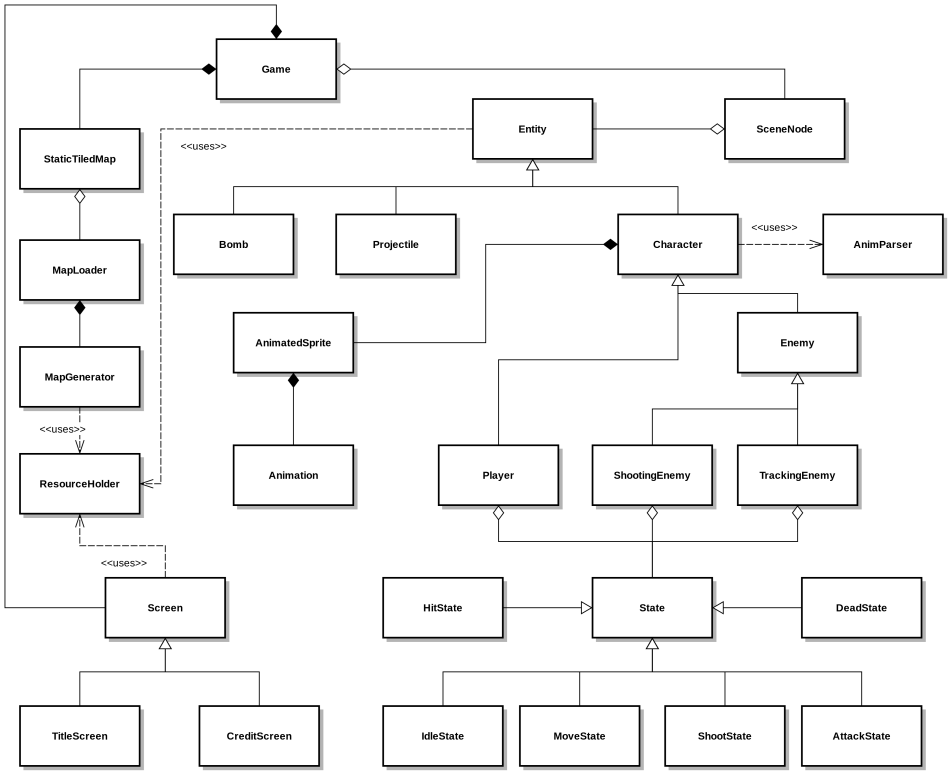


Figura 4.2: Diagrama de clases

4.4. DIAGRAMAS DE ACTIVIDAD

A continuación se muestra el diagrama de actividad del programa desde que se ejecuta hasta que finaliza.

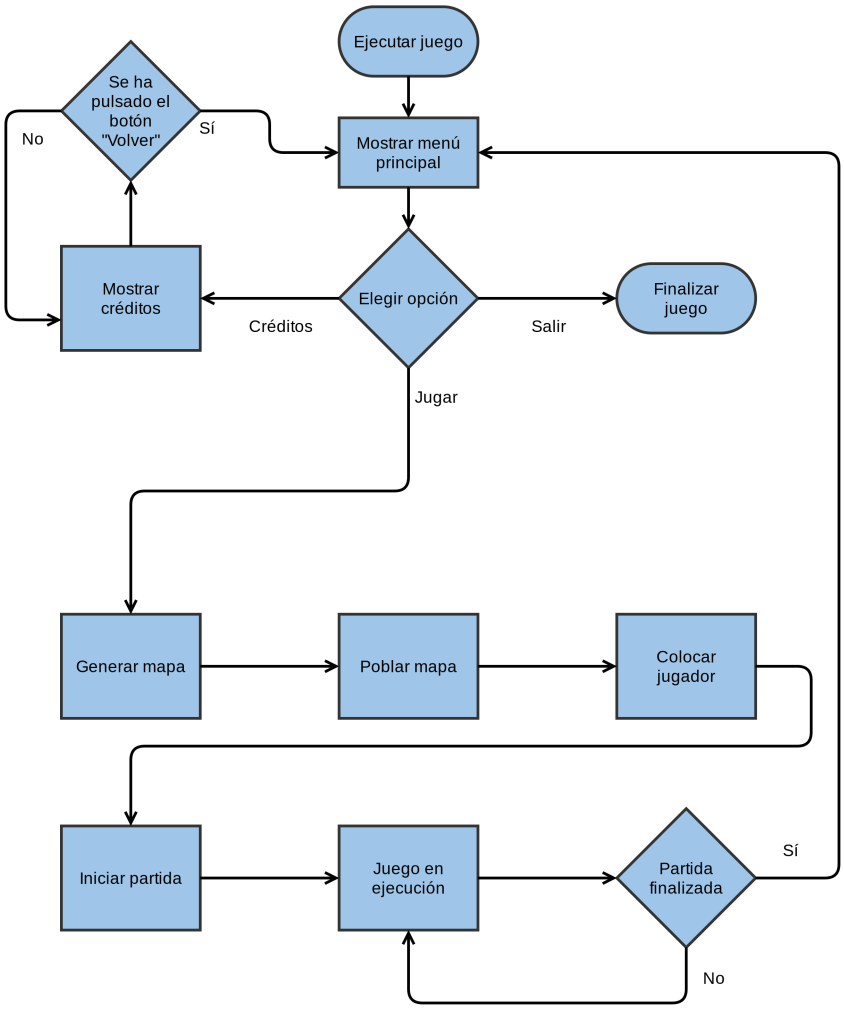


Figura 4.3: Diagrama de actividad

4.5. TECNOLOGÍAS UTILIZADAS

En este capítulo se describirán las tecnologías utilizadas para el desempeño del proyecto. Además, se dará una breve explicación de por qué han sido utilizadas y qué beneficios aportan al proyecto.

4.5.1. SFML

Simple and Fast Multimedia Library (SFML) [11] es una librería multiplataforma de desarrollo de software diseñada para proveer una interfaz simple a varios componentes multimedia en ordenado-

res. Está escrita en C++ con enlaces disponibles para C, D, Java, Python, Ruby, .NET, Go, Rust, OCaml, Euphoria y Nim. Existen también compilaciones experimentales para dispositivos móviles.

SFML gestiona tanto la creación e interacción de ventanas como de contextos OpenGL. También provee de un módulo grñafico para gráficos acelerados por hardware en 2D el cual incluye representación de texto utilizando FreeType, un módulo de audio que se sirve de OpenAL y un módulo de conexión para comunicación básica por TCP y UDP.

SFML es un software gratuito y de código libre provisto bajo los términos de la licencia zlib/png. Está disponible para Windows, Linux, OS X y FreeBSD.

Figura 4.4: Logotipo de SFML

Se ha decidido utilizar SFML en el proyecto ya que uno de los objetivos del mismo es aprender a crear una arquitectura software apropiada para videojuegos, de forma que los motores con editor visual no eran una opción, al abstraer al usuario de ella. A pesar de que la librería de referencia para este tipo de aplicaciones es SDL, se ha decidido usar SFML ya que a diferencia de la primera, la cual está escrita en C, SFML está escrita en C++ y concebida con orientación a objetos. Esto supone una ventaja ya que el proyecto ha sido desarrollado en C++ y el paradigma de programación utilizado ha sido el de la orientación a objetos.

4.5.2. C++

C++ [3] es un lenguaje de programación de propósito general. Tiene características de programación imperativa, orientada a objetos y genérica, mientras provee facilidades para la manipulación de memoria a bajo nivel.

Está diseñado pensando en la programación de sistemas, sistemas embebidos, sistemas con recursos limitados y grandes sistemas, con el rendimiento, la eficiencia y la flexibilidad de uso como sus requisitos de diseño. C++ también ha sido útil en otros muchos contextos, siendo fortalezas clave la infraestructura de software y aplicaciones con recursos limitados, incluyendo aplicaciones de escritorio, servidores, aplicaciones de rendimiento crítico y software de entretenimiento. C++ es un lenguaje compilado, con implementaciones del mismo disponibles en muchas plataformas y provistas por varias organizaciones, incluyendo FSF, LLVM, Microsoft e Intel.

C++ está estandarizado por ISO, con la última versión estándar ratificada y publicada por SFML en diciembre de 2014 como ISO/IEC 14882:2014 (informalmente conocida como C++14). Muchos otros lenguajes de programación han sido influenciados por C++, entre los que se encuentran C#, Java, y versiones posteriores a 1998 de C.

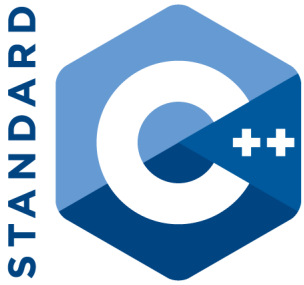


Figura 4.5: Logotipo de C++

Se ha decidido utilizar C++ para el desarrollo del proyecto ya que es el lenguaje estándar de la industria, y dado que el producto busca competir, es importante que esté hecho con las mejores herramientas. C++ es en este caso dicha herramienta, por su eficiencia y flexibilidad.

4.5.3. JSON

JavaScript Object Notation (JSON) [9] es un formato estándar abierto que usa texto legible por humanos para transmitir objetos de información consistentes en pares atributo-valor. Es principalmente utilizado para transmitir información entre un servidor y una aplicación web como alternativa a XML.

Aunque originalmente fue derivado del lenguaje de programación Javascript, JSON es un formato de datos independiente del lenguaje. El código necesario para generar y analizar información en JSON está disponible en muchos lenguajes de programación.

Figura 4.6: Logotipo de JSON

Se ha decidido utilizar JSON frente a XML porque el equipo disponía de experiencia previa con JSON y no con XML. Ambas tecnologías ofrecen características similares, pero JSON es más simple y por tanto más fácil de mantener.

4.5.4. TGUI

Texus' Graphical User Interface (TGUI) [12] es una librería de GUI multiplataforma en C++ para SFML. Entre sus características más destacables encontramos la facilidad de uso, la portabilidad de código, la posibilidad de modificar interfaces sin necesidad de recompilar, un gestor de texturas externo que evita la recarga de imágenes y está provisto bajo la misma licencia que SFML, zlib/png.

Figura 4.7: Logotipo de TGUI

Se ha decidido utilizar TGUI en el proyecto para agilizar el desarrollo de las interfaces de usuario. Hubiese sido posible crear una implementación propia con los elementos necesarios, pero hubiese consumido recursos que eran de gran valor para otros apartados del proyecto.

4.5.5. JsonCpp

JsonCpp [18] es una librería C++ que permite manipular documentos JSON, incluyendo serialización y deserialización a y desde cadenas de texto. También preserva comentarios existentes en los pasos de serialización y deserialización, haciendo el formato conveniente para archivos generados por usuarios.

De entre las distintas opciones para trabajar con JSON en C++, se ha decantado por esta ya que su integración en el proyecto era muy sencilla, al igual que su uso.

4.6. DISEÑO DEL JUEGO

En esta sección se presentará el diseño del juego, es decir, la definición del comportamiento del mismo dentro de una partida.

4.6.1. Jugador

El jugador podrá moverse libremente por el mapa y utilizar cualquiera de sus habilidades en todo momento, siempre que no esté ya usando una o esté siendo desplazado por haber sido alcanzado por un ataque enemigo. No existirá límite en el número de veces que puede usar cualquiera de sus habilidades.

Si el jugador es dañado, su barra de vida, la cual siempre será visible durante una partida, se reducirá en un cuarto. De esta manera, si el jugador es dañado cuatro veces en la misma partida, morirá y la partida se considerará un fracaso. No existirá manera de regenerar la barra de vida.

El enemigos dispone de tres habilidades: ataque cuerpo a cuerpo, disparo y poantar bomba. Si un jugador colisiona con un enemigo mientras ejecuta el ataque cuerpo a cuerpo y el lado de la colisión coincide con el lado hacia el que está ejecutando el ataque, el enemigo será dañado y el jugador quedará impune. En cualquier otro caso si el jugador es alcanzado por un enemigo o un proyectil enemigo, será dañado.

4.6.2. Enemigos

Habrá dos tipos de enemigo en el juego, ambos con el objetivo de acabar con el jugador. Ambos estarán quietos hasta que el jugador entre en su radio de acción, y se quedarán quietos si el jugador sale del mismo.

El primer enemigo buscará chocarse contra el jugador, de forma que buscará la ruta más corta hasta el mismo y procederá a recorrerla. En caso de que el jugador se mueva volverá a buscar una ruta hasta él. Si el enemigo colisiona con el jugador, lo dañará y lo empujará hacia atrás. Este enemigo morirá si es alcanzado por cualquiera de las habilidades del jugador.

El segundo enemigo, en vez de buscar una ruta hasta el jugador, buscará la ruta más corta para alienarse con él y la recorrerá. Una vez alineado, disparará al jugador infinitas veces, con un breve espacio de tiempo entre disparo y disparo. Los disparos de este enemigo no podrán ser bloqueados o eliminados de ninguna manera, solo podrán ser esquivados. Este enemigo morirá si es alcanzado por cualquiera de las habilidades del jugador, y tanto sus disparos como su contacto físico directo dañarán al jugador y lo empujarán.

4.6.3. Colisiones con el mundo

Tanto el personaje como los enemigos deben colisionar adecuadamente con el mundo, es decir, solo deben ser capaces de atravesar las celdas que, tanto a nivel lógico como visual, son navegables. En caso de que intenten atravesar una celda que no cumpla alguno de los requisitos, el juego debe parar su movimiento y colocarlo justo al lado de la casilla que pretendía atravesar.

4.6.4. Colisiones con objetos

Habrá dos objetos con los que se pueda colisionar: proyectiles y bombas.

Los proyectiles no colisionarán con el mundo, es decir, serán capaces de atravesar cualquier obstáculo. Los proyectiles lanzados por el jugador solo dañarán a enemigos y viceversa.

Las bombas no podrán ser colocadas en partes del mapa que no se puedan atravesar. Una vez colocadas, permanecerán inactivas un breve periodo de tiempo, y tras esto explotarán. Mientras la bomba está inactiva, no tendrá ninguna consecuencia colisionar con ella. No obstaculizará el movimiento y no dañará. Sin embargo, a la hora de explotar, la caja de colisión de la bomba aumentará y dañará a cualquier cosa que colisione con ella, ya sea un enemigo o el propio jugador.

4.6.5. Objetivo

El objetivo del juego es eliminar a todos los enemigos del mapa dentro del tiempo límite fijado. Este tiempo será visible en todo momento en pantalla, y su valor se actualizará cada segundo. Se considerará una partida existosa si el jugador consigue acabar con todos ellos, y se considerará un fracaso si los enemigos consiguen matarlo o se le agota el tiempo límite.

4.7. PLAN DE PRUEBAS

Durante la realización del proyecto han habido muchas pruebas para garantizar la calidad del código e intentar minimizar al máximo los errores. En este capítulo se explican las pruebas hechas.

En la siguiente lista se muestran los tipos de prueba que se han hecho:

- **Pruebas unitarias:**
  Descripción de las pruebas unitarias llevadas a cabo en el proyecto.
- **Pruebas de integración:**
  Descripción de las pruebas de integración llevadas a cabo.
- **Pruebas de hardware:**
  Descripción de las pruebas de hardware realizadas.
- **Pruebas de usabilidad:**
  Descripción de las pruebas de usabilidad realizadas.

4.7.1. Pruebas unitarias

A lo largo del proyecto se han realizado muchas pruebas unitarias. Estas consisten en dividir el código a partes mínimas para garantizar que el funcionamiento del mismo es el esperado. La realización de estas pruebas han sido enfocadas a la búsqueda de errores en los siguientes elementos:

- **Condiciones booleanas:** verificar el comportamiento del sistema al finalizar y asegurar que las variables evaluadas tienen asignados los valores esperados.
- **Indices de matrices:** verificar que en ningún momento se accede a posiciones fuera del rango de las matrices.
- **Comprobar valores nulos:** comprobar que donde se espera que haya una instancia de una clase verdaderamente la haya, de forma que el valor no sea nulo.
- **Operaciones de conversión:** asegurar que la conversión de un tipo a otro de datos es la apropiada, reforzando el código para que una conversión inadecuada cause un error en la aplicación.
- **Condiciones alternativas:** garantizar que al menos una rama es siempre satisfecha.

- **Iteraciones:** asegurar que las condiciones sean correctas de forma que nunca se generen bucle infinitos.
- **Impresión de secuencia:** utilizada para comprobar si la aplicación ejecuta bucles o condicionales imprimiendo la secuencia en consola.

4.7.2. Pruebas de integración

La realización de estas pruebas consiste en garantizar que el funcionamiento de cada módulo sea correcto. Los módulos han sido probados de distintas maneras para asegurar que siempre se ejecutaban satisfactoriamente. Además, se han hecho pruebas que verifican que las interfaces de comunicación entre distintos módulos son correctas. Se ha prestado especial atención a este apartado para evitar que el error de un módulo pudiera propagarse a toda la aplicación, evitando así fallos en cadena.

4.7.3. Pruebas de hardware

Cuando el producto se encontraba en sus etapas finales, se ha procedido a compilar y ejecutar el juego en distintas configuraciones de hardware para comprobar que funcionaba correctamente. Gracias a la colaboración de compañeros y amigos ha habido posibilidad de probar el producto en una gama variada de hardware. En total se ha compilado y ejecutado el juego satisfactoriamente en catorce configuraciones distintas. Además, estas configuraciones disponían de distintos sistemas operativos, entre los que se encuentran Ubuntu, Linux Mint, Windows 7, Windows 8.1 y OS X. Esto demuestra que el código generado por el proyecto es multiplataforma.

4.7.4. Pruebas de usabilidad

Una parte fundamental de los juegos es la experiencia de usuario. Para conseguir la mejor posible, se ha contado con la colaboración de distintas personas que han ayudado a calibrar distintos menúes e interfaces gráficas. Las pruebas consistían en sesiones cortas de juego, en las que los usuarios compartía sus observaciones respecto a la interfaz y usabilidad general. Gracias a las opiniones recogidas, se han ajustado las posiciones y los tamaños de los elementos de la interfaz gráfica para acomodarlos a la mayoría de los usuarios. Por otro lado, se han ajustado los menús para ser más intuitivos.

5. CONSIDERACIONES SOBRE LA IMPLEMENTACIÓN

5.1. VISIÓN GENERAL

En este capítulo se recogen las reglas de estilo seguidas a la hora de codificar el proyecto, así como una descripción y justificación de las herramientas utilizadas para la realización del mismo. Por último, se explica cómo han sido llevados a cabo los apartados más destacables del desarrollo del juego.

5.2. REGLAS DE ESTILO

A pesar de que no existe un estándar oficial para escribir código, hay numerosas guías de estilo y mejores prácticas que son usadas por la mayoría de la comunidad de programadores. Estas guías suelen estar agrupadas por lenguaje de programación y facilitan la lectura, modificación y comprensión de código ajeno, y al mismo tiempo sirven de buenas prácticas.

- El código estará escrito y comentado en inglés, con la intención de hacerlo lo más universal posible.
- Todos los archivos fuente estarán codificados en UTF-8, para permitir el uso de carácteres especiales.
- Los nombres de los archivos comenzarán en mayúsculas, para separar palabras se escribirá la primera letra de cada una en mayúsculas.
- Los nombres de las variables y funciones estarán escritos en minúscula, para separar palabras se escribirá la primera letra de cada una en mayúsculas, empezando por la segunda.
- Se darán nombres descriptivos a las variables, funciones y archivos.
- Inmediatamente antes de la cabecera de cada función se comentará su funcionalidad, así como la especificación de sus parámetros de entrada y el resultado de su salida.
- Cualquier otra clarificación de fragmentos de código será debidamente comentada.

5.3. ENTORNO DE DESARROLLO

Para el desarrollo del proyecto se han utilizado varias herramientas para facilitar la codificación, compilación y búsqueda de errores.

5.3.1. Atom

Atom [2] es un editor de texto desarrollado por Github. Es una herramienta que permite personalizar cualquier cosa, pero también permite ser usada productivamente sin tocar ningún archivo

de configuración. Atom ofrece integración con Node.js y está diseñado de forma completamente modular, de forma que es posible acoplar un número indefinido de módulos a su núcleo mínimo. Además, su versión más mínima ya dispone de todas las características necesarias para empezar a trabajar. Por último, Atom es de código libre y, por tanto, gratuito.

Figura 5.1: Logotipo de Atom

Este software ha sido la herramienta principal para el proyecto. Ha sido utilizada para la codificación de todo el proyecto, para el retoque de los archivos JSON y para la generación de los archivos de configuración necesarios. Cualquier editor de texto sencillo podría haber sido utilizado, pero se ha optado por Atom ya que provee características avanzadas para codificar, tales como soporte para distintos lenguajes, que además pueden ser ampliadas mediante módulos gratuitos.

5.3.2. GNU Compiler Collection

El GNU Compiler Collection (GCC) [6] es un conjunto de compiladores creados por el proyecto GNU. GCC es software libre y lo distribuye la FSF bajo la licencia general pública GPL.

Estos compiladores se consideran estándar para los sistemas operativos derivados de UNIX, de código abierto y tambiñen de propietarios, como Mac OS X. GCC requiere el conjunto de aplicaciones conocido como *binutils* para realizar tareas como identificar archivos objeto u obtener su tamaño para copiarlos, traducirlos o crear listas, enlazarlos, o quitarles símbolos innecesarios.

Originalmente GCC solo compilaba C, pero posteriormente se extendió para compilar C++, Fortran, Ada y otros.

Figura 5.2: Logotipo de GCC

Se ha decidido utilizar este compilador ya que, además de ser gratuito, es el estándar para muchas plataformas, de forma que su fiabilidad es muy alta.

5.3.3. Git

Git [8] es un software de control de versiones diseñado por Linus Torvalds pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir interfaces de usuario. Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

Figura 5.3: Logotipo de Git

Se ha decidido utilizar Git como sistema de control de versiones de entre las muchas opciones disponibles ya que el equipo tenía experiencia previa trabajando con la herramienta. Además, Git ofrece muchas ventajas, es flexible, potente y rápido.

5.3.4. Make

Make [10] es una herramienta de gestión de dependencias, típicamente, las que existen entre los archivos que componen código fuente de un programa, para dirigir su recompilación o generación automáticamente. Si bien es cierto que su función bñasica consiste en determinar automáticamente qué partes de un programa requieren ser recompiladas y ejecutar los comandos necesarios para hacerlo, también lo es que Make puede usarse en cualquier escenario en el que se requiera, de

alguna forma, actualizar automáticamente un conjunto de archivos a partir de otro, cada vez que éste cambie.

Make es muy usada en los sistemas operativos tipo Unix/Linux. Por defecto lee las instrucciones para generar el programa u otra acción del ficher Makefile. Las instrucciones escritas en este fichero se llaman dependencias.

Figura 5.4: Logotipo de Make

Make ha sido de gran ayuda para el proyecto ya que, al utilizar la librería SFML, la generación de ejecutables no era directa, eran necesarios dos pasos: compilar y enlazar. En ambos pasos era imprescindible incluir todas las librerías necesarias y especificar unos parámetros concretos. Make ha servido para automatizar estos dos pasos, evitando así múltiples errores.

5.3.5. darkFunction Editor

darkFunction Editor [4] es un estudio de gráficos gratuito y de código libre que permite definir matrices gráficas rápidamente y construir animaciones complejas, que pueden ser exportadas como GIF o XML [16].

Figura 5.5: Logotipo de darkFunction

Este programa ha sido utilizado para generar archivos XML que contenían las coordenadas de cada fotograma de animación. Además, también ha creado archivos XML que definen las animaciones por fotogramas. Esta herramienta ha sido de un valor incalculable, ya que permitía hacer mediante una interfaz gráfica sencilla lo que a mano suponía una carga de trabajo grandísima.

5.3.6. XML to JSON

XML to JSON [17] es una sencilla aplicación web que permite convertir al instante y de forma gratuita archivos XML a JSON y viceversa. Está alojada en una web mantenida por Osys.

Esta herramienta ha sido utilizada en el proyecto para convertir los archivos XML generados por la aplicación previamente mencionada en archivos JSON, que serían los después utilizados por el producto.

5.3.7. GIMP

GNU Image Manipulation Program (GIMP) [7] es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito. Forma parte del proyecto GNU y está disponible bajo la licencia pública general de GNU. Es el programa de manipulación de gráficos disponible en más sistemas operativos.

GIMP tiene herramientas que se utilizan para el retoque y edición de imágenes, dibujo de formas libres, cambiar el tamaño, recortar, hacer fotomontajes, convertir a diferentes formatos de imagen, y otras tareas más especializadas. Se pueden también crear imágenes animadas en formato GIF e imágenes animadas en formato MPEG usando un plugin de animación.

Figura 5.6: Logotipo de GIMP

Este programa se ha utilizado para editar las imágenes que se han utilizado en el juego. Se ha decidido utilizar esta herramienta por ser la mejor de su condición entre todas las opciones gratuitas.

5.4. DESARROLLO DEL JUEGO

5.4.1. Animaciones

Todos los personajes del juego cuentan con una animación visual distinta para cada acción que realizan, ya sea moverse, atacar o usar alguna habilidad. Para animarlos, es necesario contar con una imagen estática para cada paso de la animación y después pintarlas en la secuencia correcta mediante algoritmia.

Estas imágenes pueden estar en dos formatos distintos: cada una en un fichero o todas juntas en el mismo, lo cual se denomina matriz de imágenes. Las segundas son mucho más usadas ya que las

operaciones de carga de ficheros son lentas y costosas, de forma que es mucho más eficiente cargar una sola imágen que varias, aún cuando el tamaño de esta es grande.

El inconveniente que tienen las matrices de imágenes es que la complejidad de los algoritmos de pintado aumenta, ya que no es suficiente simplemente con dibujar la imagen entera en el momento correcto, sino que es necesario dibujar un pedazo concreto de la imagen en un momento concreto. Cada uno de estos pedazo corresponde a cada paso de las animaciones, y pueden ser de tamaño variable entre sí.

Hay que hacerle saber al programa cuáles son las coordenadas y el tamaño de cada pedazo, y existen dos formas de hacerlo: introduciendo los datos a mano, lo cual es muy costoso e inconveniente en caso de necesidad de modificación, o leerlas de un archivo. En este caso, se ha optado por utilizar archivos JSON para almacenar las coordenadas de las imágenes y la definición de animaciones.

Estos archivos han sido generados a partir de los archivos XML proporcionados por el programa darkFunction Editor. Para la conversión de XML a JSON, se ha utilizado la aplicación web XML to JSON.

A continuación se muestran las estructura de los archivos JSON que contiene animaciones y coordenadas.

Algoritmo 5.1: Estructura de las animaciones en JSON

```
{
  "animations": {
    "-spriteSheet": "PlayerSprites.json",
    "-ver": "1.2",
    "anim": [
      {
        "-name": "IdleDown",
        "-loops": "0",
        "cell": [
          "-index": "0",
          "-delay": "1",
          "spr": {
            "-name": "Player/Idle/Down/0",
            "-x": "0",
            "-y": "0",
            "-z": "0"
          ]
        }
      }
    ]
  }
}
```

Algoritmo 5.2: Estructura de las coordenadas en JSON

```
{
  "img": {
    "-name": "Player.png",
    "-w": "1208",
    "-h": "3736",
    "definitions": {
      "dir": {
        "-name": "Player",
        "dir": [
          {
            "-name": "Idle",
            "dir": [
              {
                "-name": "Down",
                "spr": [
                  {
                    "-name": "0",
                    "-x": "15",
                    "-y": "9",
                    "-w": "18",
                    "-h": "23"
                  }
                ]
              }
            ]
          }
        ]
      }
    }
  }
}
```

Como se puede ver, el archivo de animaciones contiene el nombre del archivo de coordenadas y una lista de animaciones. Cada animación cuenta con un nombre y una lista de celdas. Las celdas contienen la ruta de las coordenadas que corresponden al paso de animación que se desea. El resto de datos que se pueden observar en la figura no se utilizan, simplemente están ahí porque los generaba darkFunction Editor.

Por otro lado, el archivo de coordenadas contiene el nombre de la imagen de la que se deben extraer las celdas, así como el valor de su altura y anchura. Tras esto, contiene el nombre de la entidad y una lista de estados. Los archivos contienen un estado por cada acción que puede realizar el personaje. Como la visualización de cada estado varía en función de la dirección en la que se ejecute, cada estado contiene una lista para cada dirección: arriba, abajo e izquierda. La derecha se consigue volteando las imágenes en tiempo de ejecución. Por último, cada dirección contiene una

lista de coordenadas. Cada coordenada tiene un número de identificación y cuatro parámetros: la coordenada en el eje horizontal, la coordenada en el eje vertical, la altura y la anchura. El resto de datos no se utilizan, pero existen porque fueron generados por darkFunction Editor.

El funcionamiento del algortimo es en realidad bastante sencillo. Recibe el nombre del archivo de animaciones, lo carga. De él extrae el nombre del archivo de coordenadas y lo carga también. A continuación, genera un objeto de animación por cada entrada en la lista de animaciones del primer JSON. Para ello, va leyendo la lista de celdas de cada animación la ruta de las coordenadas. Por último, se dirige al archivo de coordenadas y busca en el árbol JSON el objeto existente el dicha ruta, y extrae los parámetros necesarios.

5.4.2. Generación de mapas

Como ya se ha definido, uno de los requisitos del proyecto era que el mapa del juego fuese generado proceduralmente. Para ello, se investigaron diferentes técnicas y algoritmos con este propósito, y, tras barajar distintas posibilidades, se optó por el siguiente algoritmo, denominado *Drunkard's walk*.

Este es un sencillo algoritmo que se puede utilizar para generar terrenos cuadriculados. Generar un escenario con una sola ejecución del algoritmo garantiza que el nivel estará completamente conectado, con una mezcla de espacios abiertos y estrechos muy variable.

A continuación se definen los pasos básicos del algoritmo:

- Inicializar todas las celdas del mapa como infranqueables.
- Seleccionar una celda del mapa aleatoriamente.
- Convertir esa celda en navegable.
- Mientras no se haya llegado al número de navegables estipulado:
  - Moverse una celda en una dirección aleatoria.
  - Si la celda es infranqueable, volverla navegable y actualizar número de navegables.

Para resultados óptimos, el algoritmo requiere que el mapa pueda ser redimensionado dinámicamente, de forma que se evite que los niveles toquen los límites con resultados insatisfactorios. Sin embargo, se le han hecho unos ajustes al algoritmo para adaptarlo a las necesidades y deseos del proyecto.

En primer lugar, el mapa no se redimensiona, el generador recibe una altura y una anchura que no varían. Para evitar el problema citado anteriormente, se ha impuesto la restricción de que el algoritmo debe dejar al menos dos celdas como infranqueables en cada límite.

Con esto ya se conseguía un mapa para la partida, pero en los videojuegos, la escenografía juega un papel muy importante en la experiencia de usuario, de forma que era necesario hacer el ambiente visualmente agradable. Para ello, se decidió que el entorno sería una isla, y se creó un método recursivo que, dadas las coordenadas de una celda infranqueable, marca todas las demás del mismo tipo que sean colindantes a ella, y después hace lo mismo con las marcadas. De esta manera, se puede marcar cada conjunto aislado de celdas no navegables.

Este método se aplica sobre la celda en el eje de las coordenadas, de forma que quedan marcadas todas las celdas que rodean a las celdas navegables. Tras esto, se aplican una serie de sencillos algoritmos que sustituyen las gráficas de las celdas marcadas por distintas gráficas de agua, dando la sensación de isla.

5.4.3. Sistemas de estados para entidades

Uno de los requisitos más importantes para conseguir una experiencia de usuario satisfactoria es que el control responda adecuadamente. Esto, a pesar de parecer fácil de conseguir, es complicado, ya que controlar lo que debe hacer un personaje en base a su estado actual y las entradas que recibe se vuelve exponencialmente complicado a medida que se añaden nuevas posibilidades de acción.

Por ello, se decidió que se crearía un sistema de estados para controlas las acciones de los personajes. De esta forma, las entidades serían en esencia máquinas de estados finitas, y cada estado se encargaría de gestionar las entradas relevantes para el mismo y efectuaría el cambio de estado adecuado de ser necesario.

Esta característica se ha implementado utilizando funcionalidades de la programación orientada a objetos, las cuales son proporcionadas por C++. Se ha creado una clase base *State*, la cual contiene un método polimórfico de actualización, el cual es reimplementado en cada clase hija para aportar las funcionalidades requeridas en cada uno. De esta manera, se han creado estados como *ShootState*, *MoveState* o *AttackState* tanto para el jugador como para los enemigos.

5.4.4. IA enemiga

Existen dos tipos de enemigos en el juego, cada uno con un comportamiento distinto. El primero de ellos persigue al jugador y lo daña al contacto. El segundo se alinea con el jugador y lo dispara. El funcionamiento de la IA de ambos es muy sencilla: cuentan con un radio de acción y, si el jugador se encuentra en él, se sirven del algoritmo A* [1] para buscar el camino hacia la posición que desean.

El destino del primer enemigo siempre es la posición del jugador, de forma que la ruta solo se recalcula si la posición actual del jugador es distinta a la última posición en la que se calculó la trayectoria.

El segundo enemigo funciona de manera ligeramente distinta. En vez de buscar una ruta hasta el jugador, busca la ruta más corta que lo lleve hasta un punto en el cual se alinee con él. Desde ahí, el enemigo procede a disparar al jugador cada un breve espacio de tiempo.

La razón por la cual se ha decidido utilizar el algoritmo A* frente a otras alternativas es que, en el desarrollo de videojuegos, este algoritmo es sin duda el más ampliamente utilizado para movimientos, ya que por definición es óptimo y completo, lo que quiere decir que siempre encuentra una solución y siempre es la óptima o una de las óptimas. Además, el coste computacional que requiere es más que asequible en comparación con los resultados que proporciona.

5.4.5. Gestión de escenas

Para la gestión de escenas de manera sencilla y eficiente, se ha desarrollado un *scene graph*, que consiste en una estructura de árbol binario formada por múltiples nodos, llamados nodos de escena. Cada nodo contiene un objeto a ser dibujado en pantalla, habitualmente una entidad.

Cada nodo puede tener un número arbitrario de nodos hijo, pero solo puede tener un nodo padre. Los nodos solo guardan su posición, rotación y escala relativos a su padre. Esto implica que un cambio a cualquiera de los atributos mencionados de una entidad es aplicado automáticamente

a las entidades hijas, lo cual es extremadamente útil cuando se quieren hacer operaciones sobre grupos. En caso de querer acceder a las posiciones, rotaciones o escalas absolutas de un nodo hijo, es suficiente con escalar en la estructura acumulando las transformaciones relativas hasta llegar a la raíz.

En videojuegos, esta técnica es muy utilizada para la gestión de mundos grandes. En estas aplicaciones, los requisitos de memoria son muy a tener en cuenta al diseñar una *scene graph*. Por esta razón, muchos de estos sistemas se basan en la instanciación para reducir costes de memoria y aumentar la velocidad. Esto quiere decir que tan solo se guarda una copia de los datos de una entidad, y se referencia en los nodos que sea necesario.

En ejecución, cada vez que se quiere actualizar o dibujar la escena, es suficiente con llamar al método correspondiente del nodo raíz, ya que este ejecuta el método sobre sí mismo e invoca el mismo método de todos sus nodos hijos, los cuales se comportan de la misma manera. Así se consigue que todos los nodos se actualicen o dibujen siempre que sea necesario.

6. INCIDENCIAS

6.1. VISIÓN GENERAL

Como es normal, a lo largo del proyecto han surgido diferentes incidencias que han retrasado el proyecto o han hecho que el alcance o los requisitos del mismo hayan tenido que ser modificados. A continuación se muestran los problemas más relevantes y cómo se han solucionado.

6.2. CONVERSIÓN DE XML A JSON

Como ya se ha explicado, se decidió utilizar JSON en el proyecto en vez de XML por ser más conveniente. Sin embargo, no existe o no se ha encontrado ninguna herramienta que generase archivos con coordenadas gráficas y animaciones en JSON, pero existen muchas que crean archivos XML, así que se decidió que se traducirían. Hacerlo a mano no era viable, ya que llevarían demasiado tiempo y existían muchas posibilidades de cometer errores. La segunda opción consistía en crear un pequeño que recibiese un XML y lo tradujese a JSON, pero de nuevo esta opción no era conveniente por los mismos problemas, además de que la complejidad del algoritmo sería grande. Por tanto se decidió buscar una herramienta que los convirtiese, y tras barajar unas cuantas posibilidades se decidió usar el servicio web XML to JSON.

XML to JSON cumplió su cometido, pero también introdujo nuevos problemas. En JSON, los valores de cada par clave-valor pueden ser de distintos tipos: cadenas, números, objetos, matrices, valores booleanos o valor nulo. La aplicación web citada convertía adecuadamente los valores del XML a JSON, todos excepto uno. Por alguna razón, el conversor en vez de escribir valores numéricos, escribía cadenas que contenían el número. Esto conllevó que el algoritmo hecho para analizar los JSON y crear las animaciones fallaba, ya que esperaba encontrar números y encontraba cadenas. Por suerte, la solución fue bastante sencilla: simplemente se leyeron los valores como cadenas y, utilizando funciones de la librería estándar de C++, se convirtieron a números antes de ser usadas. A pesar de que esta solución hace que el algoritmo sea algo más lento, se consideró la solución más apropiada, ya que la aplicación de la misma era muy sencilla y el coste temporal mínimo.

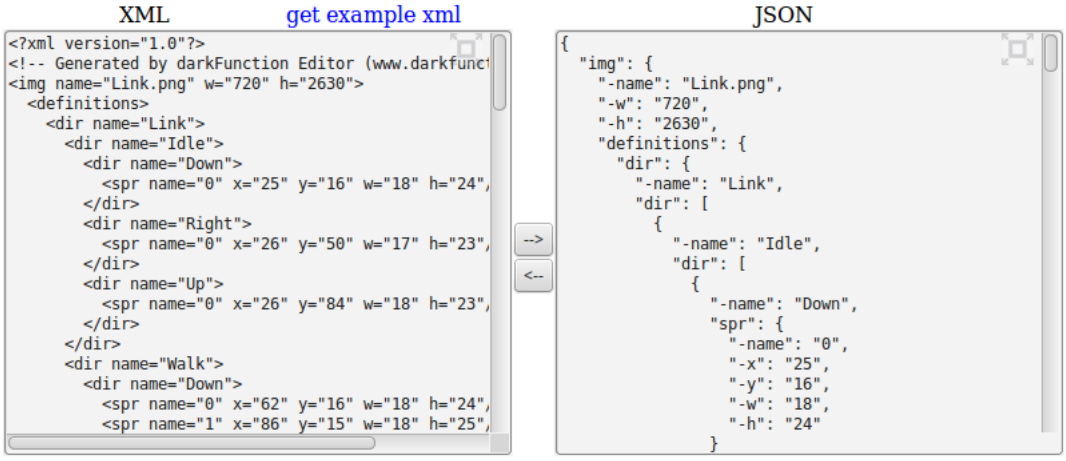


Figura 6.1: Ejemplo del error de conversión de números

6.3. SFML DEL REPOSITORIO DESACTUALIZADA

TGUI, la librería utilizada para crear las interfaces gráficas de usuario, es muy estricta con las versiones de SFML con las que trabaja. Cada versión exige una versión de SFML y solo ofrece compatibilidad con versiones posteriores, nunca retrocompatibilidad. En el momento en el que instaló por primera vez la entonces última versión de TGUI, esta requería utilizar la versión 2.2 de SFML, que también era en ese momento la última disponible. Dado que SFML se instaló desde el repositorio oficial, se asumió que se habría instalado dicha versión, pero no fue así. Por algún motivo, el repositorio aún servía la versión anterior, la 2.1, de forma que al intentar utilizar la librería TGUI aparecían un montón de errores y la aplicación no funcionaba. En este punto había dos opciones: utilizar una versión más antigua de TGUI que fuese compatible con SFML 2.1 o actualizar SFML a la versión 2.2. Se decidió actualizar SFML, ya que la nueva versión corregía algunos errores y traía mejoras en la funcionalidad, lo que ayudaría a conseguir un producto final más robusto.

6.4. ERROR DE INICIALIZACIÓN DE PALANCA DE JUEGO

En un punto del proyecto, sin razón aparente, apareció repentinamente el siguiente error al ejecutar el juego:

```
alonai@alonaiBook ~/Github/Kingdom-of-Hatred $ ./KingdomOfHatred
Failed to initialize inotify, joystick connections and disconnections won't be notified
```

Figura 6.2: Captura del error de inicialización

Aparentemente, el error se da debido a que SFML no detecta adecuadamente las palancas de control. Cabe mencionar que, aunque SFML provea soporte para palancas de control y mandos, no se han utilizado en el proyecto en ningún momento, de forma que no debería haber errores a causa de ello. Sin embargo, SFML siempre inicializa el módulo encargado de ello, y en ocasiones falla. Se investigó sobre el problema y se vió que ya había sido reportado por varios usuarios desde la versión 2.1 de SFML, y al parecer ocurría en sistemas que utilizaban un sistema operativo basado

en Ubuntu cuando había muchos archivos abiertos. Asumiendo esto, el error debería desaparecer si se cerraban archivos, pero no era así. El error persistía y no desaparecía a pesar de que se reiniciara el ordenador o se recompilase el programa. Finalmente, la forma de solucionarlo fue la siguiente: se desintalaron tanto SFML como TGUI y se reinstalaron en sus versiones más recientes. Durante el desarrollo del proyecto la versión 2.3 SFML fue lanzada, y TGUI se actualizó también con una versión compatible con SFML 2.3. Se procedió a reinstalar ambas librerías y, al recompilar el código, el error desapareció.

7. CONCLUSIONES Y LÍNEAS FUTURAS

7.1. VISIÓN GENERAL

Este capítulo recoge, a modo de conclusión, la visión del equipo de trabajo tras la realización del proyecto, expresando sus opiniones acerca del trabajo realizado teniendo en cuenta contribuciones personales y desarrollo profesional.

7.2. OBJETIVOS CUMPLIDOS

A pesar de no contar con tanto tiempo como se hubiese deseado para la realización del proyecto debido a diversas responsabilidades, se podría considerar que los objetivos del proyecto han sido cumplidos, tal como se definen en el capítulo correspondiente.

- Se ha desarrollado un producto jugable y estable.
- Las características principales están implementadas.
- La arquitectura del software está preparada para añadir y modificar con facilidad nuevas funciones.
- La investigación hecha para desarrollar del proyecto ha permitido aprender mucho sobre el desarrollo de soluciones software de este tipo.
- Se han aprendido técnicas y patrones de diseño utilizados comúnmente en la industria, y se han implementado algunos de ellos.
- El producto está en una fase que, si se añadiesen más funcionalidades y otras características de juegos, podría comercializarse.

Por otro lado, los requisitos definidos del software han sido cumplidos también:

- El nivel es generado proceduralmente en cada sesión y es poblado de enemigos.
- El juego es capaz de detectar cuando distintas entidades colisionan entre ellas o con el escenario y actúa en consecuencia.
- La cámara del juego sigue al jugador en los ejes X e Y.
- El juego detecta las entradas del jugador, comprueba el estado actual del mismo y actúa en consecuencia.
- El juego detecta cuando se han cumplido las condiciones para fin de partida y la termina.
- Cada enemigo tiene una inteligencia artificial distinta.
- El jugador tiene a su disposición tres tipos de ataque.
- El juego cuenta con una pantalla para mostrar los controles.
- Durante las partidas el juego muestra en todo momento los datos relevantes al usuario mediante una interfaz gráfica.
- La interfaz gráfica no molesta al jugador.

- El juego se controla únicamente con el teclado.
- El código del juego es multiplataforma, de forma que puede compilarse en Windows, Linux y OSX.
- La versión final incluye las librerías necesarias para el correcto funcionamiento del juego.
- El juego funciona a un mínimo de 30 FPS en cualquier plataforma.
- Durante las partidas, no hay errores que provoquen un final inesperado de la aplicación.

7.3. CONSIDERACIONES DEL TRABAJO REALIZADO

Este proyecto nació de la afición a los videojuegos y del deseo a crearlos. Desde siempre el alumno ha sido un gran aficionado a jugar a estos producto, y a medida que jugaba se iba interesando más y más por ellos. Ya desde pequeño, tras pasar las épocas de astronauta y bombero, el alumno decía que se quería dedicar a ello, y con el paso de los años esa decisión no hizo más que reafirmarse. Existen muchas maneras de entrar a formar parte del mundo del desarrollo de videojuegos, pero todos los profesionales coinciden en que, a pesar de que existen ofertas formativas muy buenas, sin duda la mejor manera es simplemente haciendo juegos. Con esto en mente, el alumno se lanzó a desarrollar el proyecto con la esperanza de que fuese la mejor experiencia educativa posible.

Para desarrollar el proyecto, el alumno tuvo que considerar una serie de aspectos. El primero fue el lenguaje de programación, ya que sería la herramienta principal del desarrollo, y la elección del mismo tendría gran repercusión. Dado que C++ es el lenguaje utilizado profesionalmente para el desarrollo de juegos, el alumno se decantó por el mismo, ya que aspira a ser un profesional del área. Además, en el grado obtuvo unas nociones básicas del mismo, de forma que la curva de aprendizaje se vería aligerada. Tras esto, comparó distintos motores y frameworks, y finalmente se decantó por SFML ya que está escrito en C++ y permitía al alumno desarrollar el software a su antojo. El alumno no tenía ninguna experiencia previa con SFML, de forma que tuvo que pasar por una fase de aprendizaje.

Para asimilar el aprendizaje teórico, el alumno desarrolló unos cuantos programas pequeños para hacer con SFML, de esta forma se familiarizó con la librería y fue capaz de detectar errores e inconveniencias. Para solucionarlos y mejorar su uso de la librería, se sirvió de la comunidad de SFML, accediendo a su foro, su wiki y a otros recursos encontrados en internet.

Por otro lado, el alumno no tenía conocimiento alguno sobre la generación procedural. Sin embargo, era un área en la que estaba muy interesado, porque el hecho de ofrecer una experiencia única en cada partida lo seducía. Por ello, se lanzó a investigar sobre distintos algoritmos de generación, enfocándose principalmente en los de generación de mapas. En este punto huelga decir que el alumno se sorprendió con la cantidad de recursos que había al respecto, no se esperaba encontrar tantísima información. Tras barajar distintas posibilidades, se decantó por utilizar un algoritmo que, pese a su sencillez, otorga unos resultados muy satisfactorios. Esta decisión se tomó en base a que por desgracia no se disponía tanto tiempo como se hubiese deseado para el desarrollo.

En general, hay que decir que pese a que se poseía conocimiento previo respecto a muchas áreas utilizadas como la programación, manipulación gráfica o el desarrollo de videojuegos, el alumno considera que ha sido una experiencia muy enriquecedora, y que gracias a ella ha aprendido un montón de técnicas y ha descubierto muchos recursos de aprendizaje nuevos que, aunque no ha podido utilizarlos todos en el desarrollo de este proyecto, sin duda serán de gran ayuda y de un

valor incalculable para el futuro.

7.4. LÍNEAS FUTURAS

Aunque se han completado todos los objetivos definidos para el proyecto, hay ciertas extensiones que se podrían llevar a cabo para mejorar el producto. El juego se puede extender con múltiples características, algunas de ellas podrían ser:

- Habilidades: Sería interesante que el jugador tuviese a su disposición más herramientas con las que interactuar con el juego. Además, podría crearse un sistema que le permitiese mejorar las habilidades que ya tuviese y asignarlas a los controles que prefiera. Por suerte este tipo de cambios estaban previstos para el futuro, y la arquitectura del software está pensada para que sea sencillo y seguro añadir más elementos de este tipo.
- Generación de mapas: En la versión actual siempre se utiliza el mismo algoritmo para generar mapas. A pesar de que los resultados son bastante variados, añadir distintos algoritmos de generación supondrían un valor añadido al proyecto. Nuevamente, el software está preparado para soportar cambios de estas características.
- Enemigos: Al solo existir dos tipos de enemigos, el juego es bastante monótono. Por ello, añadir más enemigos con comportamientos únicos añadiría dinamicidad y diversión al juego. Por otro lado, se podría pensar en crear enemigos más poderosos, con una IA única, que podrían ser enemigos finales para los niveles. Una vez más, el software está pensado para añadir nuevos enemigos con facilidad, aunque habría que diseñar la parte relativa a los nuevos enemigos únicos.
- Historia: A pesar de que en el pasado prácticamente todos los juegos fuesen arcade y no tuviesen apenas historia, hoy en día la narrativa de un juego tiene la misma o más importancia que la jugabilidad. Por ello, desarrollar una historia para el juego e implementar elementos para poder contarla adecuadamente sería muy enriquecedor. Los elementos a añadir serían cuadros de diálogo para los personajes, un sistema de secuencias etc.

Si todas o algunas de las extensiones fuesen llevadas a cabo, sin duda el producto final sería de una calidad notable, lo cual se traduce en que podría ser puesta en venta al público y haría competencia con otros juegos del mismo corte.

Bibliografía

[1] *«Introduction to A*»*. URL: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html (visitado 24-06-2015).

[2] *«Página oficial de Atom»*. URL: https://atom.io/ (visitado 24-06-2015).

[3] *«Página oficial de C++»*. URL: https://isocpp.org/ (visitado 24-06-2015).

[4] *«Página oficial de darkFunction Editor»*. URL: http://darkfunction.com/editor/ (visitado 24-06-2015).

[5] *«Página oficial de EA»*. URL: http://www.ea.com/ (visitado 24-06-2015).

[6] *«Página oficial de GCC»*. URL: https://gcc.gnu.org/ (visitado 24-06-2015).

[7] *«Página oficial de GIMP»*. URL: http://www.gimp.org/ (visitado 24-06-2015).

[8] *«Página oficial de Git»*. URL: https://git-scm.com/ (visitado 24-06-2015).

[9] *«Página oficial de JSON»*. URL: http://json.org/ (visitado 24-06-2015).

[10] *«Página oficial de Make»*. URL: https://www.gnu.org/software/make/ (visitado 24-06-2015).

[11] *«Página oficial de SFML»*. URL: http://www.sfml-dev.org/ (visitado 24-06-2015).

[12] *«Página oficial de TGUI»*. URL: https://tgui.eu/ (visitado 24-06-2015).

[13] *«Página oficial de The Legend of Zelda»*. URL: http://www.zelda.com/ (visitado 24-06-2015).

[14] *«Página oficial de Ubisoft»*. URL: https://www.ubisoft.com/ (visitado 24-06-2015).

[15] *«Página oficial de Valve»*. URL: https://www.valvesoftware.com/ (visitado 24-06-2015).

[16] *«Página oficial de XML»*. URL: http://www.w3.org/XML/ (visitado 24-06-2015).

[17] *«Página oficial de XML to JSON»*. URL: http://www.utilities-online.info/xmltojson/ (visitado 24-06-2015).

[18] *«Repositorio oficial de JsonCpp»*. URL: https://github.com/open-source-parsers/jsoncpp (visitado 24-06-2015).

Acrónimos

API Application Programming Interface. 2

EA Electronic Arts. 2, 59

FPS Frames per second. 56

FSF Free Software Foundation. 34, 42

GBA Game Boy Advance. 3

GCC GNU Compiler Collection. 42, 43, 59

GIF Graphics Interchange Format. 44, 45

GIMP GNU Image Manipulation Program. 45, 59

GNU GNU's Not Unix. 42, 45

IA Inteligencia Artificial. 49, 57

IEC International Electrotechnical Commission. 34

ISO International Organization for Standardization. 34

JSON JavaScript Object Notation. 35, 36, 42, 45–48, 51, 59

LLVM Low Level Virtual Machine. 34

MPEG Moving Picture Experts Group. 45

OpenAL Open Audio Library. 34

OpenGL Open Graphics Library. 34

RPG Role Playing Game. 3

SDL Simple DirectMedia Layer. 2, 34

SFML Simple and Fast Multimedia Library. 2, 33, 34, 36, 44, 52, 53, 56, 59

TCP Transmission Control Protocol. 34

TGUI Texus' Graphical User Interface. 36, 52, 53, 59

UDP User Datagram Protocol. 34

UTF Unicode Transformation Format. 41

XML Extensible Markup Language. 35, 44–46, 51, 59

A. MANUAL DE USUARIO

A.1. MENÚ PRINCIPAL

Al ejecutar la aplicación, se le muestra al usuario un menú principal. Dicho menú cuenta con tres opciones:

- Jugar
  Esta opción dará comienzo a una partida. El jugador será traslado a la pantalla de juego, donde se desarrollará la partida.
- Créditos
  Esta opción mostrará la pantalla de créditos, la cual contará con un botón *Atrás*, el cual devolverá al usuario al menú principal.
- Salir
  Esta opción cerrará la aplicación.

A.2. OBJETIVO

Una vez dentro de la partida, existen dos formas de finalizarla:

- **Eliminando a todos los enemigos**: El jugador, valiéndose de las tres habilidades de las que dispone, debe acabar con todos los enemigos que hay en el nivel. Si lo consigue, la partida terminará y el jugador será enviado de vuelta al menú principal automáticamente.
- **Excediendo el tiempo de partida**: Alternativamente, si el jugador pasa demasiado tiempo sin conseguir eliminar a todos los enemigos del nivel y la cuenta atrás mostrada en pantalla llega a cero, la partida finalizará automáticamente, devolviendo al jugador al menú principal.

A.3. CONTROLES

El jugador, para interactuar con el juego, debe utilizar el siguiente esquema de controles:

- En el menú principal, el usuario debe interactuar con los botones mediante el ratón, es decir, para seleccionar cualquier de las opciones, debe hacer click en el botón correspondiente.
- Una vez en partida, el resto de las entradas se realizan mediante el teclado. Para moverse por el escenario, el jugador debe utilizar las teclas direccionales. Para utilizar sus habilidades, deberá utilizar las teclas A, S o D en función de la que quiera ejecutar.
  - **A**: Disparar flecha.
  - **S**: Colocar bomba.
  - **D**: Ataque cuerpo a cuerpo.

Agradecimientos

No quisiera finalizar esta memoria sin dar las gracias a todas aquellas personas que me han apoyado durante estos meses de trabajo. En especial quisiera dar las gracias a:

- **Mis padres**, por darme la vida, mantenerme (a pesar de haber salido tonto) y haberme dado la oportunidad de hacer lo que quería y apoyarme en todo momento (menos cuando me quise dejar el pelo largo, ahí tuve que pelear).
- **Andoni Eguíluz Morán**, por aceptar dirigir el proyecto y por la ayuda y buen trato que siempre he recibido de su parte.
- **Mis compañeros y amigos**, tanto de la facultad como de fuera de ella. Por animarme en los momentos de bajón, aguantar mis chapas e idas de olla y ayudarme cuando lo he necesitado. Y todo ello sin pagarles nada.
- **Desarrolladores de videojuegos**, especialmente indies, por la gran inspiración que han supuesto y sin la cual seguramente este proyecto ni siquiera hubiese nacido.
- **Mi estructura ósea**, por el apoyo que me ha proporcionado todos estos años.

Y en general a toda persona, ya sea amigos, compañeros, desarrolladores, artistas, músicos, directores de cine, actores, actrices, médicos, carpinteros, fontaneros, camareros, barrenderos, dependientes de tiendas de animales, profesores, estudiantes, ninis, técnicos de televisión, ingenieros de lavavajillas, pregoneros, abogados, arquitectos, carniceros, cirujanos, dentistas, fotógrafos, jardineros, modelos, psiquiatras, soldados, taxistas, caballeros andantes, recepcionistas, veterinarios, panaderos, enfermeros, electricistas o bomberos que, ya sea directa o indirectamente, hayan conseguido sacarme una sonrisa durante este último año. De verdad, a todos, gracias de corazón.