



# 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 general	v
Índice de figuras	vii
Índice de tablas	ix
Índice de algoritmos	xi
<b>1 Introducción</b>	<b>1</b>
1.1 Presentación del Documento . . . . .	1
1.2 Estado del arte y motivación . . . . .	1
<b>2 Objetivos del proyecto</b>	<b>3</b>
2.1 Visión general . . . . .	3
2.2 Definición del proyecto . . . . .	3
2.2.1 Objetivos . . . . .	3
2.2.2 Alcance . . . . .	4
2.3 Producto final . . . . .	4
2.4 Descripción de la realización . . . . .	4
2.4.1 Método de desarrollo . . . . .	4
2.4.2 Tareas principales: . . . . .	5
2.4.3 Productos intermedios: . . . . .	6
2.5 Organización . . . . .	6
2.5.1 Esquema organizativo . . . . .	6
2.5.2 Plan de recursos humanos . . . . .	7
2.6 Condiciones de ejecución . . . . .	7
2.6.1 Entorno de trabajo . . . . .	7
2.6.2 Diálogo durante el proyecto . . . . .	8
2.6.3 Recepción de productos . . . . .	8
2.6.4 Control de cambios . . . . .	8
2.7 Planificación . . . . .	8
	v

2.7.1	Plan de trabajo . . . . .	8
2.7.2	Diagrama de Gantt . . . . .	17
2.7.3	Diagrama de Red . . . . .	19
2.8	Presupuesto . . . . .	24
2.8.1	Software . . . . .	24
2.8.2	Hardware . . . . .	24
2.8.3	Recursos humanos . . . . .	24
2.8.4	Total . . . . .	24
<b>3</b>	<b>Especificación de Requisitos</b>	<b>25</b>
3.1	Visión general . . . . .	25
3.2	Requisitos funcionales . . . . .	25
3.3	Requisitos no-funcionales . . . . .	26
3.3.1	Usabilidad . . . . .	26
3.3.2	Entorno . . . . .	26
3.3.3	Rendimiento . . . . .	26
3.4	Criterios de validación . . . . .	26
<b>4</b>	<b>Especificación del Diseño</b>	<b>29</b>
4.1	Visión general . . . . .	29
4.2	Arquitectura . . . . .	29
4.3	Diagramas de actividad . . . . .	29
4.4	Diagrama de clases . . . . .	29
4.5	Tecnologías utilizadas . . . . .	29
4.5.1	SFML . . . . .	29
4.5.2	C++ . . . . .	30
4.5.3	JSON . . . . .	31
4.5.4	TGUI . . . . .	32
4.5.5	JsonCpp . . . . .	32
<b>5</b>	<b>Consideraciones sobre la implementación</b>	<b>33</b>
5.1	Visión general . . . . .	33
5.2	Reglas de estilo . . . . .	33
5.3	Entorno de desarrollo . . . . .	33
5.3.1	Atom . . . . .	33
5.3.2	GNU Compiler Collection . . . . .	34
5.3.3	Git . . . . .	35

5.3.4	Make . . . . .	35
5.3.5	darkFunction Editor . . . . .	36
5.3.6	XML to JSON . . . . .	36
5.3.7	GIMP . . . . .	37
5.4	Desarrollo del juego . . . . .	37
5.4.1	Generación de mapas . . . . .	37
5.4.2	Personaje . . . . .	37
5.4.3	Enemigos . . . . .	37
<b>6</b>	<b>Plan de Pruebas</b>	<b>39</b>
6.1	Pruebas unitarias . . . . .	39
6.2	Pruebas de integración . . . . .	40
6.3	Pruebas de hardware . . . . .	40
6.4	Pruebas de usabilidad . . . . .	40
<b>7</b>	<b>Manual de Usuario</b>	<b>41</b>
<b>8</b>	<b>Incidencias</b>	<b>43</b>
8.1	Visión general . . . . .	43
8.2	Conversión de XML a JSON . . . . .	43
8.3	Versión de SFML del repositorio desactualizada . . . . .	43
8.4	Error de inicialización de palanca de juego . . . . .	44
<b>9</b>	<b>Conclusiones y Líneas Futuras</b>	<b>45</b>
9.1	Visión general . . . . .	45
9.2	Objetivos cumplidos . . . . .	45
9.3	Consideraciones del trabajo realizado . . . . .	45
9.4	Líneas Futuras . . . . .	45
	<b>Acrónimos</b>	<b>47</b>
	<b>Agradecimientos</b>	<b>49</b>





## Índice de figuras

### Capítulo 2

2.1	Modelo incremental . . . . .	5
2.2	Esquema Organizativo . . . . .	7
2.3	Diagrama del plan de trabajo 1 . . . . .	9
2.4	Diagrama del plan de trabajo 2 . . . . .	10
2.5	Diagrama del plan de trabajo 3 . . . . .	11
2.6	Diagrama del plan de trabajo 4 . . . . .	12
2.7	Diagrama del plan de trabajo 5 . . . . .	13
2.8	Diagrama del plan de trabajo 6 . . . . .	14
2.9	Diagrama del plan de trabajo 7 . . . . .	15
2.10	Diagrama del plan de trabajo 8 . . . . .	16
2.11	Diagrama de Gantt . . . . .	18
2.12	Diagrama de precedencias 1 . . . . .	19
2.13	Diagrama de precedencias 2 . . . . .	20
2.14	Diagrama de precedencias 3 . . . . .	21
2.15	Diagrama de precedencias 4 . . . . .	22
2.16	Diagrama de precedencias 5 . . . . .	23

### Capítulo 4

4.1	Logotipo de SFML . . . . .	30
4.2	Logotipo de C++ . . . . .	31
4.3	Logotipo de JSON . . . . .	31
4.4	Logotipo de TGUI . . . . .	32

### Capítulo 5

5.1	Logotipo de Atom . . . . .	34
5.2	Logotipo de GCC . . . . .	34
5.3	Logotipo de Git . . . . .	35

5.4	Logotipo de Make . . . . .	36
5.5	Logotipo de darkFunction . . . . .	36
5.6	Logotipo de GIMP . . . . .	37

## Índice de tablas

2.1	Presupuesto: Hardware . . . . .	24
2.2	Presupuesto: Recursos Humanos . . . . .	24
2.3	Presupuesto: Total . . . . .	24



## Índice de algoritmos



## 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 procedualmente 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,

## 1. INTRODUCCIÓN

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 desarrolladores 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, SFML o libGDX. Para este proyecto se ha decidido usar un motor de este tipo, en concreto SFML.

### 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

---

//TODO Cambiar visión general, no corresponde con la realidad

### 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

## 2. OBJETIVOS DEL PROYECTO

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 rigurosa 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.



Figura 2.1: 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.

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.

## 2. OBJETIVOS DEL PROYECTO

### 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:

- **Betateesting** 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.



Figura 2.2: Esquema Organizativo

### 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:

- 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 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. OBJETIVOS DEL PROYECTO

### 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 las 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












Id	Nombre del recurso	Trabajo	Detalles	105 ene 15							12 ene 15							19 ene 15							
				S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S
1	Desarrollador	354 horas	TrabajaC																						
	Seguimiento 1	3 horas	TrabajaC																						
	Seguimiento 2	3 horas	TrabajaC																						
	Seguimiento 3	3 horas	TrabajaC																						
	Seguimiento 4	3 horas	TrabajaC																						
	Seguimiento 5	3 horas	TrabajaC																						
	Seguimiento 6	3 horas	TrabajaC																						
	Seguimiento 7	3 horas	TrabajaC																						
	Seguimiento 8	3 horas	TrabajaC																						
	Seguimiento 9	3 horas	TrabajaC																						
	Seguimiento 10	3 horas	TrabajaC																						
	Organización	3 horas	TrabajaC								3h														
	Análisis de herramientas de c	9 horas	TrabajaC																						
	Análisis de técnicas de gener	9 horas	TrabajaC																						
	Diseño del juego	15 horas	TrabajaC																						
	Formación	30 horas	TrabajaC																						
	Diseño	45 horas	TrabajaC																						
	Implementación	95 horas	TrabajaC																						
	Betatesting	45 horas	TrabajaC																						
	Prueba de experiencia de usu	55 horas	TrabajaC																						
	Despliegue de la versión fina	15 horas	TrabajaC																						
	Cierre del proyecto	3 horas	TrabajaC																						

Figura 2.3: Diagrama del plan de trabajo 1

2. OBJETIVOS DEL PROYECTO












Id	Nombre del recurso	Trabajo	Detalles	26 ene 15							02 feb 15							09 feb 15								
				V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S
1	Desarrollador	354 horas	Trabajo																							
	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																							
	Seguimiento 9	3 horas	Trabajo																							
	Seguimiento 10	3 horas	Trabajo																							
	Organización	3 horas	Trabajo																							
	Análisis de herramientas de a	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																							
	Prueba de experiencia de usu	55 horas	Trabajo																							
	Despliegue de la versión fina	15 horas	Trabajo																							
	Cierre del proyecto	3 horas	Trabajo																							

Figura 2.4: Diagrama del plan de trabajo 2



[illegible]

Figura 2.5: Diagrama del plan de trabajo 3

2. OBJETIVOS DEL PROYECTO

Id	Nombre del recurso	Trabajo	Detalles	09 mar '15							16 mar '15							23 mar '15																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
				X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L

Figura 2.6: Diagrama del plan de trabajo 4

Id	Nombre del recurso	Trabajo	Detalles	30 mar '15							06 abr '15							13 abr '15						
				X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M
1	Desarrollador	354 horas	Trabajo																					
	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																					
	Seguimiento 9	3 horas	Trabajo																					
	Seguimiento 10	3 horas	Trabajo																					
	Organización	3 horas	Trabajo																					
	Análisis de herramientas de a	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																					
	Prueba de experiencia de us	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

2. OBJETIVOS DEL PROYECTO

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

Figura 2.8: Diagrama del plan de trabajo 6

Id	Nombre del recurso	Trabajo	04 may '15							11 may '15							18 may '15							25 may '15						
			L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D
1	Desarrollador	354 horas Trabajo		3h	3h	3h	3h	8h			6h	3h	3h	3h	3h		3h	3h	3h	3h	8h	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																												
	Seguimiento 9	3 horas Trabajo																												
	Seguimiento 10	3 horas Trabajo									3h																			
	Organización	3 horas Trabajo																												
	Análisis de herramientas de a	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																												
	Prueba de experiencia de usu	55 horas Trabajo																												
	Despliegue de la versión fina	15 horas Trabajo																												
	Cierre del proyecto	3 horas Trabajo																												

Figura 2.9: Diagrama del plan de trabajo 7

2. OBJETIVOS DEL PROYECTO

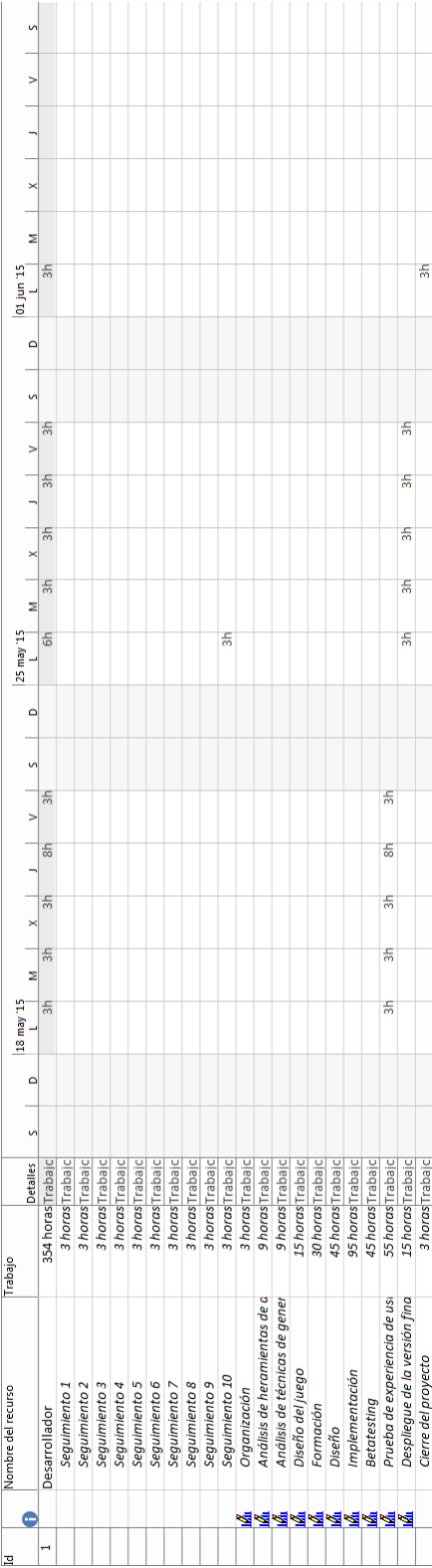


Figura 2.10: Diagrama del plan de trabajo 8

### **2.7.2. Diagrama de Gantt**

## 2. OBJETIVOS DEL PROYECTO

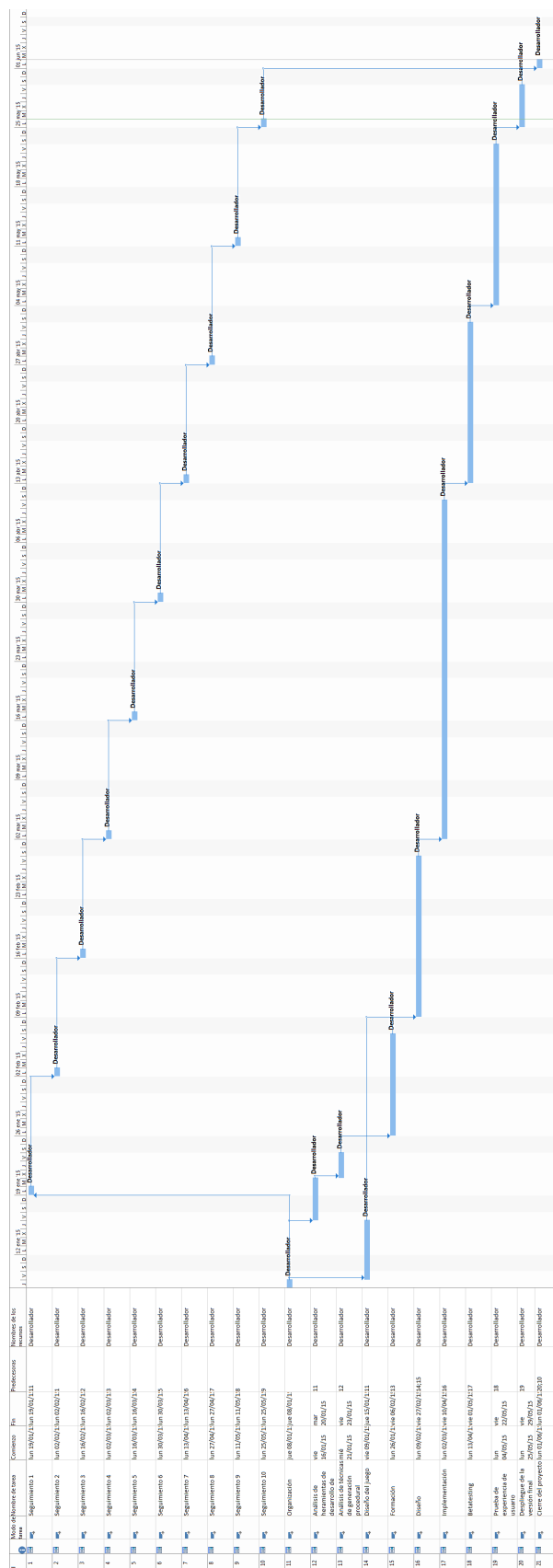


Figura 2.11: Diagrama de Gantt



### 2.7.3. Diagrama de Red

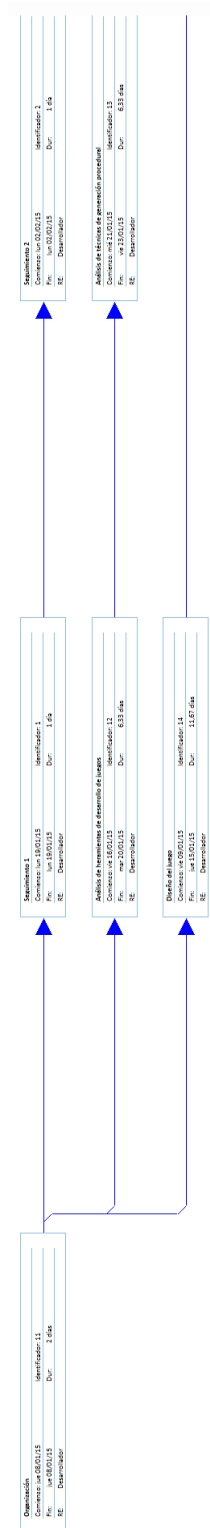


Figura 2.12: Diagrama de precedencias 1

## 2. OBJETIVOS DEL PROYECTO

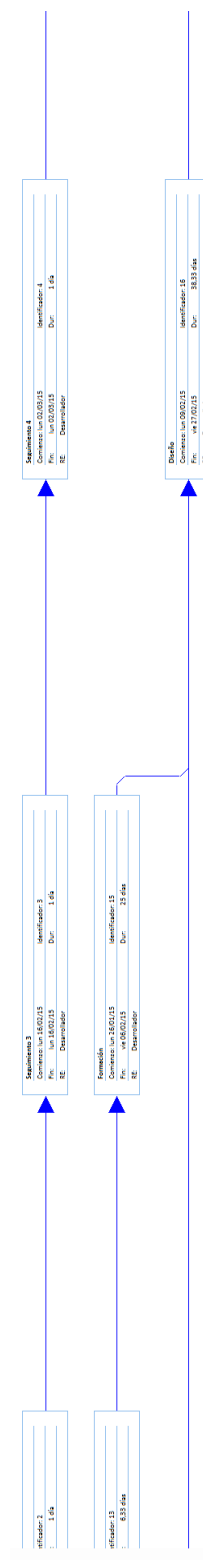


Figura 2.13: Diagrama de precedencias 2

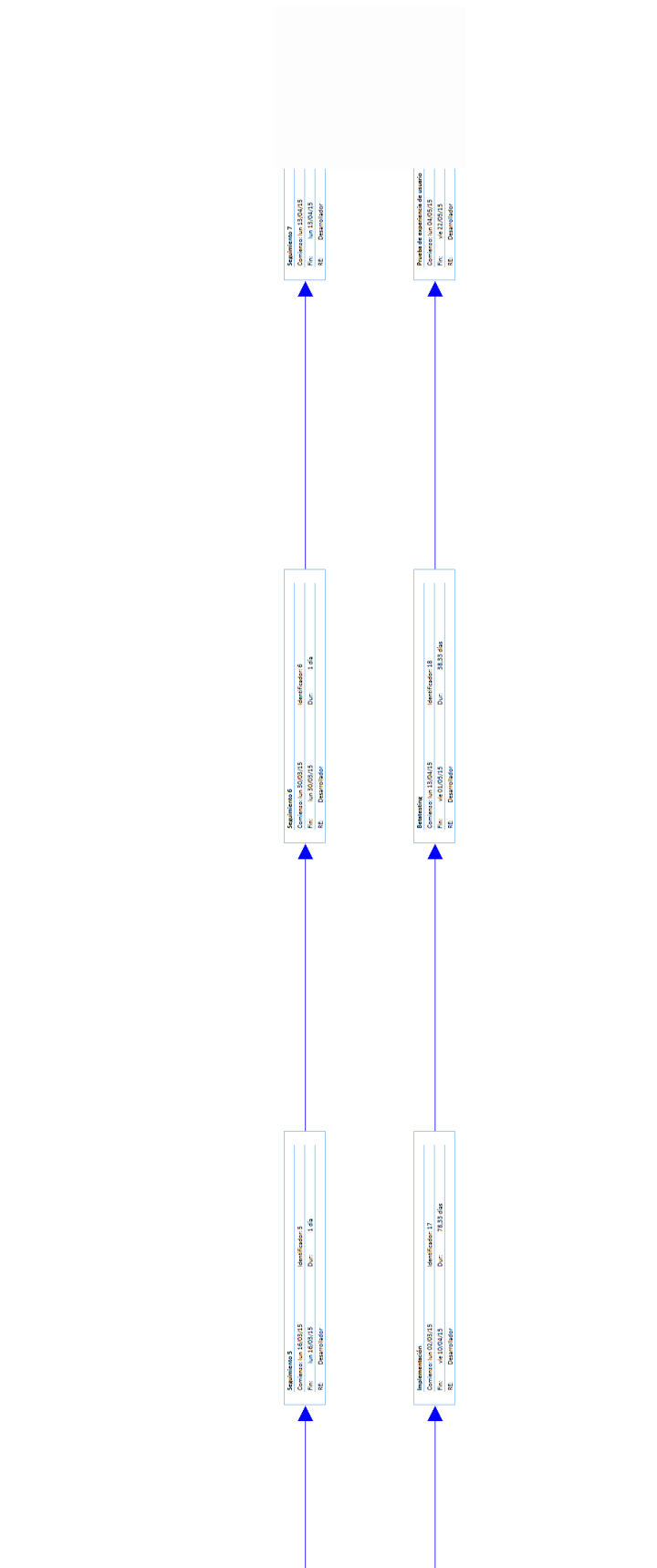


Figura 2.14: Diagrama de precedencias 3

2. OBJETIVOS DEL PROYECTO

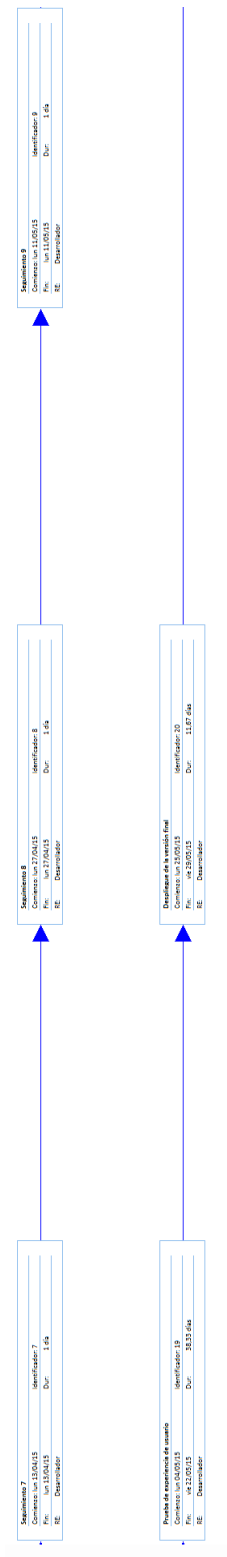


Figura 2.15: Diagrama de precedencias 4

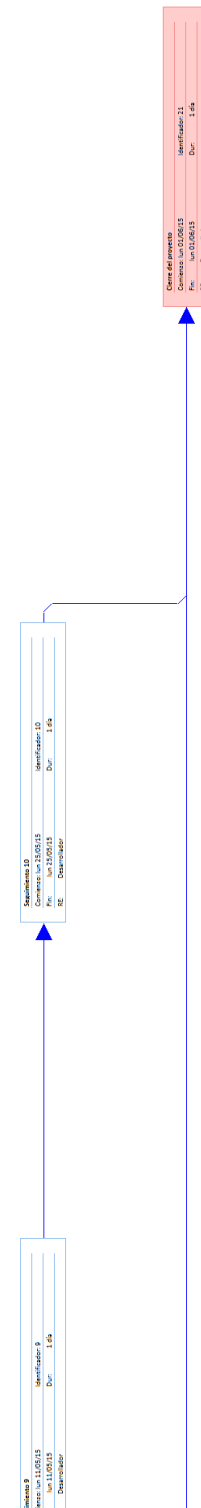


Figura 2.16: Diagrama de precedencias 5

## 2.8. PRESUPUESTO

### 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

Tabla 2.1: Presupuesto: Hardware

Nombre	Precio(€)	Unidades	Importe total(€)
Ordenador de sobremesa	1500	1	1500
Ordenador portátil	700	1	700

### 2.8.3. Recursos humanos

Tabla 2.2: Presupuesto: Recursos Humanos

Rol	Precio/hora(€/h)	Carga de trabajo(h)	Importe total(€)
Jefe de proyecto	30	54	1620,00
Programador	25	321	8025,00
Tester	15	90	1350,00

### 2.8.4. Total

Tabla 2.3: Presupuesto: Total

Tipo	Total
Recursos Humanos	10995
Recursos Hardware	2200
<b>Total</b>	<b>13195</b>

## 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 dividido 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 expectativas 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

### 4.2. ARQUITECTURA

### 4.3. DIAGRAMAS DE ACTIVIDAD

### 4.4. DIAGRAMA DE CLASES

### 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) es una librería multiplataforma de desarrollo de software diseñada para proveer una interfaz simple a varios componentes multimedia en ordenadores. 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 gráfico 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.

#### 4. ESPECIFICACIÓN DEL DISEÑO



Figura 4.1: 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++ 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.

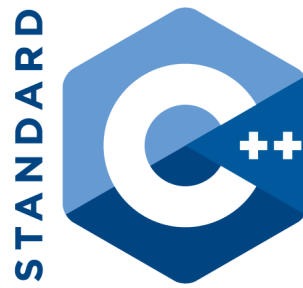


Figura 4.2: Logotipo de C++

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) 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 4.3: 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. ESPECIFICACIÓN DEL DISEÑO

### 4.5.4. TGUI

Texus' Graphical User Interface (TGUI) 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 4.4: 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 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.

## 5. CONSIDERACIONES SOBRE LA IMPLEMENTACIÓN

---

### 5.1. VISIÓN GENERAL

//TODO

### 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 caracteres 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 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,

## 5. CONSIDERACIONES SOBRE LA IMPLEMENTACIÓN

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) 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 también 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 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 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 básica 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 fichero 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 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.



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 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) 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. Generación de mapas

//WIP

### 5.4.2. Personaje

//WIP

### 5.4.3. Enemigos

//WIP



## 6. 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.

### 6.1. 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.

### 6.2. 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.

### 6.3. 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.

### 6.4. 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 menús 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.

## **7. MANUAL DE USUARIO**

---





## 8. INCIDENCIAS

---

### 8.1. 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.

### 8.2. 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.

//TODO Poner imagen que demuestre que se convierte mal

### 8.3. VERSIÓN DE 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

## 8. INCIDENCIAS

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.

### 8.4. ERROR DE INICIALIZACIÓN DE PALANCA DE JUEGO

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

```
//TODO Imagen del error "Failed to initialize inotify, joystick connections and disconnections won't be notified"
```

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 desinstalaron 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.

## **9. CONCLUSIONES Y LÍNEAS FUTURAS**

---

### **9.1. VISIÓN GENERAL**

### **9.2. OBJETIVOS CUMPLIDOS**

### **9.3. CONSIDERACIONES DEL TRABAJO REALIZADO**

### **9.4. LÍNEAS FUTURAS**



## Acrónimos

- API** Application Programming Interface. 2
- FSF** Free Software Foundation. 30, 34
- GCC** GNU Compiler Collection. 34
- GIF** Graphics Interchange Format. 36, 37
- GIMP** GNU Image Manipulation Program. 37
- GNU** GNU's Not Unix. 34, 37
- IEC** International Electrotechnical Commission. 30
- ISO** International Organization for Standardization. 30
- JSON** JavaScript Object Notation. 31, 32, 34, 36, 43
- LLVM** Low Level Virtual Machine. 30
- MPEG** Moving Picture Experts Group. 37
- OpenAL** Open Audio Library. 29
- OpenGL** Open Graphics Library. 29
- SDL** Simple DirectMedia Layer. 30
- SFML** Simple and Fast Multimedia Library. 2, 29, 30, 32, 36, 43, 44
- TCP** Transmission Control Protocol. 29
- TGUI** Texus' Graphical User Interface. 32, 43, 44
- UDP** User Datagram Protocol. 29
- UTF** Unicode Transformation Format. 33
- XML** Extensible Markup Language. 31, 36, 43



## **Agradecimientos**

- Mi estructura ósea, por el apoyo que me ha proporcionado todos estos años.

