



*ugr*

Universidad  
de Granada



Trabajo de fin de grado  
Grado en ingeniería informática(2016-2017)

# **Videojuego sencillo para PCs**

**Alumno**

Antonio David López Machado

**Tutor**

Carlos Ureña Almagro

---

# Índice

---

|   |           |
|---|-----------|
| <b>1. Resumen y palabras clave</b>                          | <b>2</b>  |
| <b>2. Resumen extendido y palabras clave en inglés</b>      | <b>2</b>  |
| <b>3. Motivación e introducción</b>                         | <b>3</b>  |
| 3.1 Contextualización del trabajo                           | 4         |
| 3.2 Problema en cuestión                                    | 4         |
| 3.3 Alcance de la memoria                                   | 5         |
| 3.4 Principales fuentes consultadas                         | 5         |
| <b>4. Objetivos del trabajo</b>                             | <b>6</b>  |
| <b>5. Resolución del trabajo</b>                            | <b>6</b>  |
| 5.1 Planificación y presupuesto                             | 7         |
| 5.1.1 Planificación   | 7         |
| 5.1.2 Tiempo estimado comparado con el tiempo real empleado | 11        |
| 5.1.3 Presupuesto   | 11        |
| 5.2 Análisis y diseño                                       | 12        |
| 5.2.1 Requisitos  | 12        |
| 5.2.2 Casos de uso  | 28        |
| 5.2.3 Diagrama conceptual                                   | 28        |
| 5.2.4 Diagrama de clases                                    | 29        |
| 5.2.5 Diagrama de estados                                   | 34        |
| 5.2.6 Diagrama de secuencia                                 | 41        |
| 5.2.7 Bocetos de las interfaces de usuario                  | 42        |
| 5.2.8 Algoritmos y estructuras no triviales                 | 51        |
| 5.3. Implementación y pruebas                               | 56        |
| 5.3.1 Implementación  | 56        |
| 5.3.2 Pruebas   | 56        |
| <b>6. Conclusiones y vías futuras</b>                       | <b>67</b> |
| <b>7. Recursos multimedia</b>                               | <b>68</b> |
| 7.1 Modelos 3D  | 68        |
| 7.2 Texturas  | 69        |
| 7.3 Sonidos   | 71        |
| <b>8. Bibliografía final</b>                                | <b>72</b> |
| <b>Anexo : Autoría y licencia del proyecto</b>              | <b>73</b> |
| <b>Anexo : Glosario</b>                                     | <b>74</b> |
| <b>Anexo : Manual de usuario</b>                            | <b>75</b> |

---

# 1. Resumen y palabras clave

El objetivo principal de este proyecto es el de profundizar más en el desarrollo de aplicaciones con un alto componente gráfico.

Para ello, se ha realizado un videojuego utilizando un motor gráfico propio. Dicho motor gráfico ha sido desarrollado con el fin de implementar el videojuego y, además, podría ser usado para desarrollar otros videojuegos.

El enfoque de nuestro motor gráfico es la realización de cualquier tipo de videojuego. Un punto a destacar dentro de las funcionalidades del motor gráfico es la detección de colisiones. De esta forma, se ha realizado un algoritmo de indexación espacial el cual nos permite detectarlas de una forma eficiente.

Otro punto a destacar dentro de las funcionalidades del motor de gráfico es la gestión de *eventos*<sup>[10]</sup> mediante diferentes dispositivos de entrada, como son el teclado y el *mando*<sup>[16]</sup> de consola.

El videojuego es de *género plataformas*<sup>[11]</sup> y representa cada objeto como uno o más *voxels*<sup>[33]</sup> para almacenar la información del escenario. Su objetivo será superar distintos niveles. Para ello, debemos esquivar o matar a los *enemigos*<sup>[7]</sup> y abrir las diferentes puertas resolviendo pequeños juegos lógicos, en los que debemos transportar una entidad llamada "*alma*<sup>[1]</sup>" a otra entidad llamada "*contenedor de alma*<sup>[5]</sup>".

Palabras clave : Motor gráfico , videojuego, colisión, voxel , género plataforma.

# 2. Resumen extendido y palabras clave en inglés

The main aim of this project is going into detail about the applications development which contain a high graphic component. Furthermore, it also improves the knowledge in illumination algorithms and 3D scene displaying.

In order to manage this idea, it has been developed a video game making use of our own made graphic engine. Thus, a video game has been made to demonstrate and test the building and improvement of such graphic engine. In addition, this graphic engine can be used in the development of other video games, including all kinds of games.

It is important to highlight the functionality of collision detection in our graphic engine. On this way, it has been done a spatial indexation algorithm which let us to detect such collisions in an efficient way.

Another important functionality of our graphic engine is the events management by different input devices, as are the keyboard and the console controller. Thus, the user can, for example, move the main character by using a controller or a keyboard, giving the game greater flexibility.

Moving to sound management, our graphic engine offers functionalities related to background music and sound effects, which enable users to customise some features. This can be also applied to window management, as the user can change its proportions, its title, its icon, and other features.

On the other hand, it is essential to explain some details about the video game. Firstly, the game genre is platforms genre. Secondly, the game shows every object as one or more voxels, which are basic units to store the scenario information. Thirdly, the game target is to complete different levels. For that, we must avoid enemies or kill them. In addition, we need to solve puzzles in order to open the different doors of the scenario. The puzzles are solved by carrying an entity called “soul” and putting it into another entity called “soul carrier”.

It is important to highlight that the user can interact with different kinds of avatars. They are relevant in the game, as they give the user some tips about how he or she can overcome the level. These tips are given through texts that appear when the main character talks with another character. On this way, the only thing that the user needs to do is guessing these tips.

Moving to map description, it will be filled with traps and blockages that the user must avoid in order to overcome the level. This gives the game a challenging tone which is an extra stimulus to the user.

Furthermore, the map is spilled with different kinds of items that the user can collect. Thus, we can collect an item called “crystal”, which increases the number of crystals that we have. In addition, we can also collect another item called “potion”, which heals the main character. This means that it increases our life percentage (our life bar refills a little).

Finally, the game plot is shown as a “cinematic” in which the story is written in a book and the user turns the pages in order to continue reading this story. Then, it is important to say that the book will be shown at the beginning of some game levels.

Key word : Graphic engine, video game, collision, voxel, platform genre.

### 3. Motivación e introducción

Esta sección incluirá varios apartados. En primer lugar se realiza una contextualización y una descripción del trabajo realizado. Además, incluirá el alcance de la memoria y las principales fuentes consultadas.

La motivación para realizar este proyecto ha sido derivada del alto interés hacia el sector de los videojuegos que he ido desarrollando a lo largo de los años. Además, estoy altamente interesado en desarrollar mi carrera como ingeniero informático dentro de este sector.

### 3.1 Contextualización del trabajo

Este proyecto se centra en dos ámbitos: por un lado en el desarrollo de un motor gráfico y, por otro, en el desarrollo de un videojuego utilizando el motor gráfico desarrollado.

El comienzo de los motores gráficos[VARONAS12] se remonta al año 1989. Este año se creó el primer motor gráfico, el cual era en 2D pero realizaba una simulación de 3D con una técnica de visualización de alturas.

Tras varios motores gráficos, los cuales producían un 3D simulado, en 1996 llegó el primer motor gráfico con 3D real, llamado Quake.

En la actualidad, los motores gráficos nos permiten generar escenarios con un complejo nivel de iluminación y de geometría de una forma sencilla e intuitiva. Además, nos permite generar multitud de sistemas de una forma casi automática, como puede ser la detección de colisiones y el procesamiento de animación.

Por otro lado, el primer videojuego[FIB08] se creó en el año 1952. Este videojuego era una versión computerizada del clásico tres en raya.

No fue hasta la década de los 90 cuando empezaron a desarrollarse videojuegos con un escenario 3D real.

En la actualidad los videojuegos tienen mecánicas y algoritmos cada vez más complejos. Por ejemplo, hay videojuegos los cuales tienen cambios climáticos que afectan tanto a la vegetación como a la fauna, migración de especies según la estación y generaciones procedurales complejas.

Por tanto, en ambos campos podemos ver que aún queda mucho por descubrir en ellos y es un tema de estudio del que se pueden sacar conclusiones relevantes.

### 3.2 Problema en cuestión

En este proyecto se pretende desarrollar un videojuego en C++ utilizando la librería gráfica OpenGL. El proyecto ha sido dividido en dos partes para que el software desarrollado sea reutilizable. Dichas partes son:

- **Motor gráfico:** conjunto de clases que implementan la funcionalidad central o de más bajo nivel del videojuego y que potencialmente puede ser utilizado para diversos videojuegos con distintas temáticas, avatares, etc.

- **Videojuego:** conjunto de clases específicamente relacionadas a los avatares, escenario y temáticas concretas específicamente relacionados con el videojuego que hemos desarrollado. Esta parte de nuestro proyecto utiliza la funcionalidad del motor gráfico descrito arriba.

### 3.3 Alcance de la memoria

En la memoria del proyecto se expone la planificación del proyecto, el análisis y diseño del proyecto, la descripción del software desarrollado y las pruebas realizadas sobre dicho software.

Las diferentes secciones del proyecto son :

- **Objetivos del trabajo:** sección donde se detallaran los principales objetivos a la hora de realizar este proyecto.
- **Resolución del trabajo:** sección amplia donde se describirán los diferentes aspectos del desarrollo del proyecto. Se divide en tres secciones:
  - **Planificación y presupuesto:** se detallarán tanto el proceso de planificación del proyecto como su presupuesto estimado.
  - **Análisis y diseño:** se detallaran todos los aspectos relativos al análisis completo y el diseño del proyecto.
  - **Implementación y pruebas:** se describirán los aspectos relacionados a la implementación del proyecto y a las pruebas realizadas .
- **Conclusiones y vías futuras:** se describen las posibles características futuras que se podrían implementar, así como las conclusiones a las que se ha llegado.
- **Recursos multimedia:** se mostrarán los diferentes recursos multimedia (imagen, modelo 3D y sonido) de terceros que han sido utilizados en la aplicación, junto con sus licencias.
- **Bibliografía final:** se listan los diferentes recursos bibliográficos que han sido utilizados tanto para la realización de esta memoria como para la implementación de la aplicación.
- **Anexo : Autoría y licencia del proyecto:** se detallan tanto los derechos de autor como la licencia que ha sido utilizada en la aplicación.
- **Anexo : Glosario:** se definen los diferentes términos relacionados con el proyecto.
- **Anexo : Manual de usuario:** explica al usuario final cómo utilizar el producto.

### 3.4 Principales fuentes consultadas

Se han utilizado varias fuentes, aunque las principales han sido:

- **OpenGL SuperBible, Sixth Edition 2014 Pearson Higher education** [[SELLERSETALL14](#)]. He consultado varios temas para diferentes aspectos del proyecto desarrollado. Los temas utilizados han sido:
  - Tema 2 para la creación y utilización de *shader* [[29](#)].
  - Tema 4 para la implementación de matrices, cámara y proyecciones.
  - Tema 5 para conocer los tipos de variables de los shader.

- Tema 7 para la visualización de *mallas*<sup>[15]</sup> con shader.
- Tema 12 para algunos detalles de la iluminación<sup>1</sup>.
- **Learn OpenGL** [\[VRIES15\]](#). Esta web vez ha sido utilizada para los siguientes carateristicas:
  - Generación de sombras arrojadas.
  - Uso de texturas *bump mapping*.
  - Calculo de luces direccionales y puntuales.
  - Transparencia en objetos.
- **SDL Wiki** [\[LANTINGA14\]](#). Aparte de varias consultas puntuales he utilizado esta web para las siguientes características:
  - Creación y gestión de la ventana de nuestra aplicación.
  - Detección y gestión del dispositivo de entrada de tipo mando.

## 4. Objetivos del trabajo

En esta sección podremos ver los objetivos del proyecto, las interdependencias entre ellos y los aspectos formativos previos que nos ayudarán a realizar dichos objetivos.

Los objetivos del proyecto son:

- Investigar en profundidad el concepto de motor gráfico y los elementos que lo componen.
- Desarrollar un motor gráfico propio utilizando la librería gráfica OpenGL.
- Usar el motor gráfico desarrollado para implementar un videojuego.
- Estudiar y desarrollar aspectos modernos en la informática gráfica para aplicarlos a nuestro juego, tales como el uso de texturas de normales y la creación de sombras arrojadas.

Los objetivos tendrán una interdependencia secuencial, es decir, para cumplir un objetivo debe haberse cumplido primero el anterior.

Como aspectos formativos previos se pueden destacar los conocimientos adquiridos en las asignaturas de informática gráfica y de sistema gráficos. En estas asignaturas se obtuvo el conocimiento sobre la librería gráfica OpenGL, el cual se ha utilizado en el proyecto.

## 5. Resolución del trabajo

En esta sección de la memoria mostraremos los métodos y procesos empleados en este proyecto.

Se ha realizado un glosario de términos para el mejor entendimiento de algunos conceptos relativos al proyecto. Dicho glosario se puede encontrar como un [anexo](#) al final de este documento.

## 5.1 Planificación y presupuesto

En esta sección podremos ver la planificación, presupuesto del proyecto así como la comparación entre el datos reales y los datos estimados.

### 5.1.1 Planificación

En este proyecto se ha utilizado como metodología de desarrollo un modelo iterativo e incremental. Cada iteración del proyecto ha incluido cada una de las siguientes fases:

- **Fase 1 - Análisis/Diseño:** todos los componentes de la iteración son analizados para determinar su implementación.
- **Fase 2 - Desarrollo/Pruebas:** se realizará tanto el desarrollo de cada componente como una serie de pruebas para comprobar el correcto funcionamiento de cada componente.
- **Fase 3 - Entrega :** se realizará un entregable funcional del software.
- **Fase 4 - Corrección de fallos :** se realiza la corrección de los errores detectados en dicho entregable.

Se han realizado un total de 6 iteraciones con una duración de un mes por iteración. La primera iteración comenzó el 1 de octubre, lo que nos permitió tener dos meses adicionales de desarrollo que se han utilizado tanto para el desarrollo de la documentación como para el desarrollo de nuestro escenario, creación/modificación de mallas y obtención de sonidos.

#### 5.1.2.1 Iteración 1

| Iteración 1          |            |              |
|----------------------|------------|--------------|
| Fase o tarea         | Inicio     | finalización |
| Análisis/Diseño      | 01/10/2016 | 07/10/2016   |
| Desarrollo/Pruebas   | 08/10/2016 | 28/10/2016   |
| Entrega              | 28/10/2016 | 28/10/2016   |
| Corrección de fallos | 29/10/2016 | 30/10/2016   |

Tabla 5.1: Planificación de la primera iteración.

#### Requisitos funcionales relacionados

- RF.W Ventana
- RF.SH Shader
- RF.LM Lector de modelos
- RF.S Sonido
- RF.MR Motor de rendering
- RF.TR Transformación



- **RF.AN Animación**
- **RF.C Cámara**

### 5.1.1.2 Iteración 2

| Iteración 2          |            |              |
|----------------------|------------|--------------|
| Fase o tarea         | Inicio     | finalización |
| Análisis/Diseño      | 01/11/2016 | 05/11/2016   |
| Desarrollo/Pruebas   | 06/11/2016 | 27/11/2016   |
| Entrega              | 27/11/2016 | 27/11/2016   |
| Corrección de fallos | 28/11/2016 | 30/11/2016   |

Tabla 5.2: Planificación de la segunda iteración.

#### **Requisitos funcionales relacionados**

- **RF.MT Material**
- **RF.I Iluminación**
- **RF.H Héroe**
- **RF.M Mapa.**

### 5.1.1.3 Iteración 3

| Iteración 3          |            |              |
|----------------------|------------|--------------|
| Fase o tarea         | Inicio     | finalización |
| Análisis/Diseño      | 01/12/2016 | 06/12/2016   |
| Desarrollo/Pruebas   | 07/12/2016 | 28/12/2016   |
| Entrega              | 28/12/2016 | 28/12/2016   |
| Corrección de fallos | 29/12/2016 | 30/12/2016   |

Tabla 5.3: Planificación de la tercera iteración.

#### **Requisitos funcionales relacionados**

- **RF.NP Npc**
- **RF.TX Texto**
- **RF.IN la Npc**
- **RF.E enemigo**
- **RF.IE la cuerpo a cuerpo**
- **RF.E.C Enemigo cuerpo a cuerpo**

- **RF.CL Colección**
- **RF.MN Menú**
  - **RF.MN.MA Menú de pausa**
  - **RF.MN.MP Menú Principal**
  - **RF.MN.MM Menú de muerte**
- **RF.IT Objeto**

#### 5.1.1.4 Iteración 4

| Iteración 4          |            |              |
|----------------------|------------|--------------|
| Fase o tarea         | Inicio     | finalización |
| Análisis/Diseño      | 01/01/2017 | 07/01/2017   |
| Desarrollo/Pruebas   | 08/01/2017 | 28/01/2017   |
| Entrega              | 28/01/2017 | 28/01/2017   |
| Corrección de fallos | 29/01/2017 | 30/01/2017   |

Tabla 5.4: Planificación de la cuarta iteración.

#### Requisitos funcionales relacionados

- **RF.CB Módulo de combate**
- **RF.SP Sistema de partículas**
- **RF.PR Projectiles**
- **RF.SPR Sistema de proyectiles**
- **RF.ID la a distancia**
- **RF.E.D Enemigo a distancia**
- **RF.AR Arma**
- **RF.CD Controlador de dispositivos.**

#### 5.1.1.5 Iteración 5

| Iteración 5          |            |              |
|----------------------|------------|--------------|
| Fase o tarea         | Inicio     | finalización |
| Análisis/Diseño      | 01/02/2017 | 04/02/2017   |
| Desarrollo/Pruebas   | 05/02/2017 | 26/02/2017   |
| Entrega              | 26/02/2017 | 26/02/2017   |
| Corrección de fallos | 26/02/2017 | 28/02/2017   |

Tabla 5.5: Planificación de la quinta iteración.

**Requisitos funcionales relacionados**

- RF.CO Compañero
- RF.RE Región
- RF.P Perfil
- RF.AM Agrupación de modelos
- RF.PC Pantalla de carga
- RF.CM Carga de mapa
- RF.N Notificación
- RF.T Trampa
- RF.SG Sistema de guardado
- RF.A Alma
- RF.CA Contenedor de alma
- RF.PU Puerta

## 5.1.1.6 Iteración 6

| Iteración 6          |            |              |
|----------------------|------------|--------------|
| Fase o tarea         | Inicio     | finalización |
| Análisis/Diseño      | 01/03/2017 | 04/03/2017   |
| Desarrollo/Pruebas   | 05/03/2017 | 27/03/2017   |
| Entrega              | 27/03/2017 | 27/03/2017   |
| Corrección de fallos | 28/03/2017 | 31/03/2017   |

Tabla 5.6: Planificación de la sexta iteración.

**Requisitos funcionales relacionados**

- RF.MC Menú de controles
- RF.CR Pantalla de créditos
- RF.CI Pantalla de cinemática
- RF.CFG Configuración
- RF.MO Módulo de menú de opciones
- RF.IH Interfaz de héroe
- RF.BM Bump mapping
- RF.S Sombras

### 5.1.2 Tiempo estimado comparado con el tiempo real empleado

Para este desarrollo se ha marcado un ritmo constante de trabajo. Dicho ritmo ha sido de 6 horas diarias con 5 días semanales de trabajo. Como podemos ver en la figura 5.1, algunas iteraciones han necesitado un aumento del ritmo de trabajo para realizar todo su contenido.

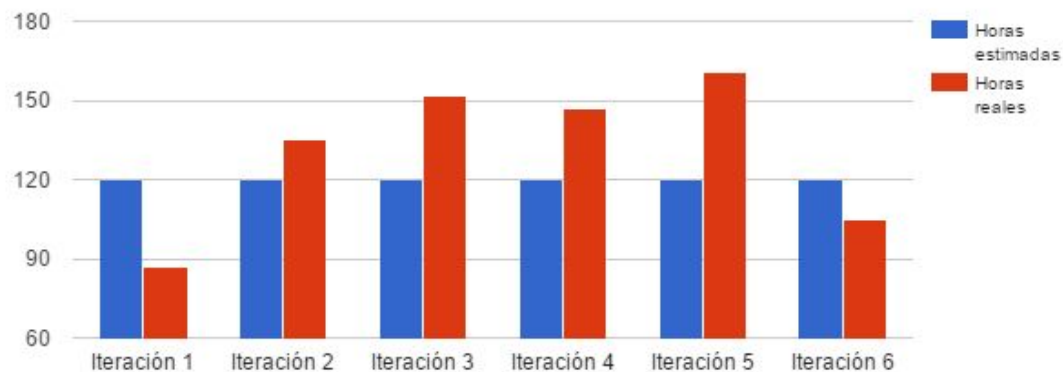


Figura 5.1: Comparación de tiempos

### 5.1.3 Presupuesto

Como se marcó un ritmo constante de trabajo, podremos determinar un presupuesto por número de horas dedicadas a dicho proyecto.

| Nº horas/día | Nº días/semana | Precio/hora | Presupuesto semanal |
|--------------|----------------|-------------|---------------------|
| 6            | 5              | 15€         | 450€                |

Tabla 5.7: Coste de personal.

Como podemos ver en la tabla 5.7, el coste de personal para este desarrollo será de un total de 450 euros semanales. Como podemos apreciar, el desarrollador tendrá 2 días de descanso por semana.

Basándonos en estas estimaciones podemos observar en la tabla 5.8 cuál sería el presupuesto total de nuestro proyecto. Se ha estimado que el tiempo de desarrollo de dicho proyecto tendrá una duración de 34 semanas (desde el 1 de octubre al 28 de mayo).

| Presupuesto/semana | Nº de semanas | Coste hardware | Coste software | Presupuesto total |
|--------------------|---------------|----------------|----------------|-------------------|
| 450€               | 34€           | 900€           | 0€             | 16200€            |

Tabla 5.8: Presupuesto total

Como podemos apreciar, el gasto realizado en términos de equipamiento (portátil, mando, ratón, etc) ha sido de un total de 900 euros. En cuanto a coste de software, este proyecto no tiene ningún gasto adicional, ya que se ha utilizado software con licencia open-source.

## 5.2 Análisis y diseño

En esta sección se incluirán todos los aspectos del análisis y del diseño de nuestra aplicación. Podremos ver tanto los requisitos del proyecto como los diferentes diagramas que definirán las distintas partes de nuestro sistema.

### 5.2.1 Requisitos

#### 5.2.1.1 Requisitos Funcionales

Los requisitos de nuestro sistema están divididos en dos secciones las cuales son:

- **Sección del videojuego:** se mostrarán los requisitos para el usuario final del videojuego desarrollado.
- **Sección del motor gráfico:** en esta sección se mostrarán los requisitos del motor gráfico desarrollado para el programador que lo utilice en la creación de un videojuego.

##### 5.2.1.1.1 Sección del videojuego

#### **RF.W Ventana**

El sistema debe proveer al usuario de una ventana.

**RF.W.1** El sistema debe permitir el cambio del tamaño de la ventana.

**RF.W.2** El sistema debe permitir maximizar, minimizar y cerrar la ventana.

**RF.W.3** El sistema debe permitir cambiar la ventana de modo (pantalla completa, modo ventana).

#### **RF.S Sonido**

El sistema debe gestionar el sonido de la aplicación.

**RF.S.1** El sistema debe permitir aumentar o reducir el volumen del sonido del videojuego.

**RF.S.2** El sistema debe ajustar el volumen del sonido según la distancia a la que se produjo con respecto al *héroe*<sup>[13]</sup>.

#### **RF.M Mapa**

El sistema debe proveer al usuario de un *mapa*<sup>[17]</sup>.

**RF.M.1** El mapa debe finalizar cuando el héroe llegue a una determinada área del escenario.

**RF.M.2** El mapa debe visualizar cada elemento que contiene que este en el rango visual del usuario.

**RF.M.3** Los objetos del mapa producirán colisiones con los *avatares*<sup>[2]</sup> que interactúan con el.

**RF.M.4** El mapa puede tener objetos decorativos los cuales no producen colisión con los avatares.

**RF.M.5** Cuando un mapa haya sido cargado y se hayan visualizado sus cinemáticas (si las hubiere) se mostrará el título de dicho mapa durante un periodo determinado de tiempo.

### **RF.MN. Menú**

El sistema debe proveer al usuario de un conjuntos de *menús*<sup>[18]</sup>.

**RF.MN.1** El sistema debe visualizar el menú cuando esté activado.

### **RF.MP Menú principal**

El sistema debe proveer al usuario de un menú inicial.

**RF.MP.1** El sistema debe visualizar el menú principal cuando se ejecute la aplicación.

**RF.MP.2** El menú debe cambiar de opción seleccionada cuando el usuario pulse los botones de movimiento.

**RF.MP.3** El menú debe activar la opción seleccionada cuando el usuario haya pulsado el botón de acción.

**RF.MP.4** El menú principal debe iniciar la carga del mapa inicial cuando el usuario haya activado la opción "New game".

**RF.MP.5** El menú principal debe iniciar la carga del mapa actual del usuario cuando haya activado la opción "Continue".

**RF.MP.6** El menú principal no debe permitir la selección de la opción "Continue" cuando el usuario no tenga un progreso registrado.

**RF.MP.7** El menú principal debe activar el menú de opciones cuando el usuario haya activado la opción de "Options".

**RF.MP.8** El menú principal debe activar el menú de controles cuando el usuario haya activado la opción de "Control".

**RF.MP.9** El menú principal debe cerrar la aplicación cuando el usuario haya activado la opción "Quit".

**RF.MP.10** El menú principal debe ser desactivado una vez el usuario haya pulsado alguna de las opciones posibles.

### **RF.PA Menú de pausa**

El sistema debe proveer al usuario de un menú pausa.

**RF.PA.1** El menú de pausa al ser activado debe mostrar por consola la información de rendimiento del videojuego.

**RF.PA.2** El menú de pausa debe cambiar de opción seleccionada cuando el usuario pulse los botones de movimiento.

**RF.PA.3** El menú debe activar la opción seleccionada cuando el usuario haya pulsado el botón de acción.

**RF.PA.4** El menú de pausa debe desactivarse cuando el usuario haya activado la opción "Resume".

**RF.PA.5** El menú de pausa debe activar al menú principal cuando el usuario haya activado la opción "Quit".

**RF.PA.6** El menú de pausa debe cambiar su estado de activado a desactivado (y viceversa) cuando el usuario haya pulsado el botón de pausa.

**RF.PA.7** Cuando el menú de pausa esté activado el juego no debe ser actualizado, es decir, el juego quedará pausado en el estado que tenía cuando el menú de pausa se activó.

**RF.PA.8** Cuando el menú de pausa sea desactivado el juego debe continuar desde el estado en el que se pauso.

#### **RF.MM Menú de muerte**

El sistema debe proveer al usuario de un menú de muerte.

**RF.MM.1** El menú de muerte debe ser activado cuando la vida del héroe haya llegado a 0.

**RF.MM.2** El menú de muerte debe activar el menú principal cuando el usuario haya pulsado el botón de acción. El menú de muerte pasará a estar desactivado.

#### **RF.PC Pantalla de carga**

El sistema debe proveer al usuario de un pantalla de carga.

**RF.PC.1** La pantalla de carga debe ser activada cuando el mapa actual active su bandera de finalización.

**RF.PC.2** La pantalla de carga debe estar activada mientras el mapa actual tenga activa la bandera de "cargando".

**RF.PC.3** La pantalla de carga debe visualizar una animación sencilla.

#### **RF.MO Menú de opciones**

El sistema debe proveer al usuario de un menú de opciones.

**RF.MO.1** El menú de opciones debe cambiar de opción seleccionada cuando el usuario pulse los botones de movimiento arriba o abajo.

**RF.MO.2** El menú de opciones debe cambiar de valor la opción actualmente seleccionada cuando el usuario pulse los botones de movimiento derecha o izquierda.

**RF.MO.2.1** Cuando tenga seleccionada la opción "Resolution" debe cambiar entre las posible resoluciones de nuestra aplicación.

**RF.MO.2.2** Cuando tenga seleccionada la opción "Window" debe cambiar entre el modo ventana y el modo pantalla completa.

**RF.MO.2.3** Cuando tenga seleccionada la opción "Volume" debe aumentar o disminuir el volumen general de nuestra aplicación.

**RF.MO.3** El menú debe activar la opción seleccionada cuando el usuario haya pulsado el botón de acción. El usuario solo podrá activar las opciones "Save" y "Quit"

**RF.MO.4** El menú debe registrar la nueva configuración del usuario cuando el usuario haya activado la opción "Save".

**RF.MO.5** El menú debe mantener la configuración anterior del usuario cuando el usuario haya activado la opción "Quit".

**RF.MO.6** Cuando el usuario haya activado la opción "Save" o "Quit" el menú pasará a estar desactivado y se activará el menú principal.

#### **RF.MC Menú de controles**

El sistema debe proveer al usuario de un menú de controles.

**RF.MC.1** El menú de controles debe mostrar los controles del videojuego para teclado como para mando.

**RF.MC.2** El menú de controles debe activar el menú principal cuando el usuario haya pulsado el botón de acción. El menú de controles pasará a estar desactivado.

**RF.CR Pantalla de créditos**

El sistema debe proveer al usuario de una pantalla de créditos.

**RF.CR.1** La pantalla de créditos debe ser activada cuando el último mapa haya finalizado.

**RF.CR.2** La pantalla de créditos debe mostrar los créditos del videojuego.

**RF.CR.3** La pantalla de créditos debe activar el menú principal cuando el usuario haya pulsado el botón de acción. La pantalla de créditos pasará a estar desactivado.

**RF.CI Pantalla de cinemáticas**

El sistema debe proveer al usuario de una pantalla de cinemáticas.

**RF.CI.1** La pantalla de cinemáticas será activada cuando el mapa recién cargado contenga un conjunto de cinemáticas a visualizar.

**RF.CI.2** Debe permitir la visualización de cada cinemática.

**RF.CI.3** Debe permitir visualizar la siguiente cinemática al pulsar el botón de acción.

**RF.CI.4** La pantalla de cinemática debe desactivarse al pulsar el botón de acción y ya se estuviera visualizando la última cinemática.

**RF.IH Interfaz de héroe**

El sistema debe proveer al usuario de un interfaz de héroe.

**RF.IH.1** Debe mostrar al usuario la vida y la cantidad de *cristales*<sup>[4]</sup> obtenidos por el héroe .

**RF.IH.2** Debe actualizar la vida del héroe cuando éste recupere o pierda vida.

**RF.IH.3** Debe actualizar la cantidad de cristales obtenidos cuando el héroe obtenga un nuevo cristal.

**RF.SP Sistema de partículas**

El sistema debe proveer al usuario de *sistemas de partículas*<sup>[30]</sup>.

**RF.SP.1** El sistema de partículas debe generar nuevas *partículas*<sup>[23]</sup> cuando el número de partículas vivas no supere el número máximo de partículas vivas.

**RF.SP.2** Las partículas generadas por el sistema deben tener unas características aleatorias pero siempre dentro de un rango de valores determinado.

**RF.SP.3** Cuando el tiempo de vida de una partícula sea consumido esta será eliminada del sistema.

**RF.SP.4** Las partículas de un sistema realizan un movimiento progresivo con una velocidad constante.

**RF.H Héroe**

El sistema debe proveer al usuario de un héroe.

**RF.H.1** El sistema debe visualizar el modelo del héroe.

**RF.H.2** El sistema debe permitir el movimiento con velocidad constante del héroe en los ejes X y Z a través del escenario al pulsar los botones de movimiento. Si el héroe colisiona contra un objeto del escenario su movimiento será detenido en esa dirección.

**RF.H.3** Cuando el héroe esté realizando un movimiento en el eje X o Z y el usuario realice un cambio de dirección la orientación del héroe será modificada por la nueva dirección indicada por el usuario.



**RF.H.4** El héroe podrá orientarse en un total de 8 direcciones según los botones que el usuario haya pulsado.

**RF.H.5** Si el héroe no está sobre un voxel, la gravedad del sistema hará que tenga un movimiento negativo en el eje Y, con un incremento de velocidad progresivo. Si el héroe colisiona con un voxel que esté bajo él, su movimiento en el eje Y se detendrá.

**RF.H.6** El sistema debe permitir que el héroe pueda realizar un salto (un movimiento positivo en el eje Y el cual irá decrementando su velocidad) cuando el usuario pulse el botón de salto.

**RF.H.7** El salto del héroe será detenido cuando su velocidad de movimiento en el eje Y sea 0 o cuando colisione contra un voxel que esté sobre él.

**RF.H.8** El sistema debe procesar y visualizar las animación del héroe para cada acción que puede realizar.

### **RF.NP Npc**

El sistema debe proveer al usuario de *npcs*<sup>[21]</sup>.

**RF.NP.1** El sistema debe visualizar el modelo del npc.

**RF.NP.2** Se visualizará un diálogo de acción sobre nuestro héroe cuando esté cerca del npc y pueda interactuar con él.

**RF.NP.3** Si un npc y el héroe están interactuando y el usuario pulsa el botón de acción, se dejará de mostrar el mensaje actual y se mostrará el siguiente mensaje sobre el avatar emisor de él.

**RF.NP.4** Cuando el npc y el héroe estén interactuando y se esté visualizando el último mensaje de la conversación, cuando el usuario pulse el botón de acción, la interacción entre el héroe y el npc habrá finalizado.

### **RF.E Enemigo**

El sistema debe proveer al usuario de enemigos.

**RF.E.1** El enemigo debe ser eliminado cuando su vida llegue a 0.

**RF.E.2** El enemigo debe pasar al estado activado cuando el héroe entre en su rango de acción

**RF.E.3** Si el enemigo se encuentra un obstáculo en el camino el enemigo podrá realizar un salto para evadir dicho obstáculo.

**RF.E.4** Si el enemigo no está sobre un voxel, la gravedad del sistema hará que tenga un movimiento negativo en el eje Y con un incremento de velocidad progresivo. Si el enemigo colisiona contra un voxel que esté bajo él, su movimiento en el eje Y será detenido.

### **RF.E.C Enemigo cuerpo a cuerpo**

El sistema debe proveer al usuario de enemigos de tipo cuerpo a cuerpo.

**RF.E.C.1** El sistema debe visualizar el modelo del *enemigo cuerpo a cuerpo*<sup>[9]</sup>.

**RF.E.C.2** El enemigo cuerpo a cuerpo activo debe perseguir al héroe.

### **RF.E.D Enemigo distancia**

El sistema debe proveer al usuario de enemigos a distancia

**RF.E.D.1** El sistema debe visualizar el modelo del *enemigo a distancia*<sup>[8]</sup>.

**RF.E.D.2** El enemigo a distancia activado debe perseguir al héroe cuando el héroe se encuentre a cierta distancia de él.

**RF.E.D.3** Si el héroe está cerca del enemigo este intentara alejarse de él.

### **RF.CB Módulo de combate**

El sistema debe proveer al usuario de un sistema de combate.

**RF.CB.1** El héroe debe poder realizar ataques con espada al pulsar el botón de ataque.

**RF.CB.2** El héroe podrá realizar una secuencia de 3 ataques con diferente animación cuando el arma equipada sea la espada.

**RF.PU.3** Cuando el héroe ataque y el arma equipada sea la espada, se debe reproducir el sonido correspondiente a dicha acción.

**RF.CB.4** El ataque del héroe producirá una reducción de vida en un enemigo si éste es alcanzado por el ataque.

**RF.CB.5** Cuando el arma equipada sea la ballesta, el héroe lanzara un *proyectil*<sup>[25]</sup> al pulsar el botón de ataque.

**RF.CB.6** El héroe o enemigo a distancia realizará una animación de disparo cuando lance una flecha.

**RF.PU.7** Cuando el héroe o enemigo a distancia lance una flecha se debe reproducir el sonido correspondiente a dicha acción.

**RF.CB.8** El sistema debe permitir que el héroe pueda cambiar de arma entre espada y ballesta.

**RF.CB.9** El héroe podrá alzar su escudo y protegerse del daño cuando el golpe impacte sobre el escudo. El usuario pulsará el botón de escudo para realizar dicha acción.

**RF.CB.10** Cuando el héroe tenga alzado su escudo y un golpe impacte sobre dicho escudo, se debe reproducir el sonido correspondiente a dicha acción.

**RF.CB.11** Cuando el héroe sea golpeado se producirá un movimiento por el impacto. Si en la trayectoria del movimiento se encuentra un objeto de la escena, el movimiento de impacto se detendrá.

**RF.CB.12** El enemigo cuerpo a cuerpo debe realizar un ataque cuando éste esté lo suficientemente cerca del héroe.

**RF.CB.13** El ataque del enemigo cuerpo a cuerpo producirá una reducción de vida de nuestro héroe si éste es alcanzado por el ataque y su escudo no para el golpe.

**RF.PU.14** Cuando el enemigo cuerpo a cuerpo realice un ataque, se debe reproducir el sonido correspondiente a dicha acción.

**RF.CB.15** El enemigo a distancia debe lanzar un proyectil hacia la posición del héroe.

**RF.CB.16** Cuando el enemigo sea golpeado se producirá un movimiento por el impacto. Si en la trayectoria del movimiento se encuentra a otro enemigo, el movimiento de impacto no se producirá.

**RF.CB.17** Cuando el héroe o un enemigo sea golpeado se debe reproducir el sonido correspondiente a dicha acción.

### **RF.CO Compañero**

El sistema debe proveer al usuario de un compañero<sup>[3]</sup>.

**RF.CO.1** El compañero debe dialogar con nuestro héroe cuando un área de texto haya sido activada.

**RF.CO.2** El compañero debe seguir a nuestro héroe por todo el escenario.

**RF.CO.3** El compañero debe cesar su movimiento cuando esté cerca del héroe

### **RF.RT Régión de texto**

El sistema debe proveer al usuario de *regiones de texto*<sup>[28]</sup>.

**RF.RT.1** La región de texto debe producir la interacción mediante mensajes entre el compañero y el héroe cuando este último haya entrado en su radio de actividad.

**RF.RT.2** Si una región de texto está activada mostrará el mensaje actual sobre el avatar emisor de él.

**RF.RT.3** Cuando una *región*<sup>[26]</sup> esté activada y el tiempo de vida del mensaje actual haya sido consumido, se dejará de mostrar dicho mensaje y se mostrará el siguiente mensaje sobre el avatar emisor de él.

**RF.RT.4** Cuando el último mensaje de la región de texto activada haya consumido su tiempo de vida, la región pasará al estado desactivado.

**RF.RT.5** Una región de texto solo podrá ser activada una vez.

### **RF.A Alma**

El sistema debe proveer al usuario de almas.

**RF.A.1** El héroe podrá portar un alma. El héroe portará/dejará un alma cuando se pulse el botón de acción cerca de una.

**RF.A.2** Cuando el alma sea portada irá cambiando la posición con respecto a la posición y orientación del héroe.

**RF.A.3** El héroe sólo podrá portar un alma simultaneamente.

**RF.A.4** Si el héroe suelta el alma ésta debe mantener la posición donde fue soltada.

**RF.A.5** El héroe no podrá efectuar ataques ni defenderse mientras esté portando un alma.

**RF.A.6** Cuando un alma es captada por un contenedor de alma, ésta ajusta su posición con respecto a la del contenedor y el héroe no podrá portar de nuevo dicha alma.

### **RF.CA Contenedor de alma**

El sistema debe proveer al usuario de contenedores de alma.

**RF.CA.1** El contenedor de alma debe captar un alma que haya sido soltada a una distancia cercana a él.

**RF.CA.2** Un contenedor de alma que haya captado un alma debe pasar al estado activado.

**RF.CA.3** Un contenedor de alma que esté activado no podrá captar otra alma.

### **RF.PU Puerta**

El sistema debe proveer al usuario de puertas.

**RF.PU.1** La puerta no debe permitir a los avatares pasar a través de ella cuando esté desactivada.

**RF.PU.2** La puerta debe permitir a los avatares pasar a través de su marco cuando esté activada.

**RF.PU.3** La puerta debe realizar una animación de apertura cuando sea activada.

**RF.PU.4** La puerta debe ser activada cuando el contenedor vinculado a él pase al estado activo.

**RF.PR Projectil**

El sistema debe proveer al usuario de proyectiles.

**RF.PR.1** El proyectil debe realizar un movimiento lineal constante.

**RF.PR.2** El proyectil debe tener una animación de rotación constante en el eje Z.

**RF.PR.3** El proyectil debe desaparecer cuando choque contra un objeto de la escena o avatar (sin incluir npc o compañero).

**RF.PR.4** El proyectil debe producir un sonido determinado cuando choque contra un objeto de la escena o avatar (sin incluir npc o compañero).

**RF.PR.5** El proyectil debe producir una reducción de vida sobre un avatar (héroe/enemigo) si ésta impacta sobre él. En el caso del héroe debe no estar escudándose en dicha dirección.

**RF.SPR Sistema de proyectiles**

El sistema debe proveer al usuario de *sistemas de proyectiles*<sup>[31]</sup>.

**RF.SPR.1** Debe detectar cuándo el héroe está en su rango de acción para pasar al estado activado.

**RF.SPR.2** Debe lanzar proyectiles de forma continuada y con las mismas características mientras esté activo.

**RF.SPR.3** Cuando el héroe no esté en su rango de acción pasará al estado desactivado.

**RF.T Trampa**

El sistema debe proveer al usuario de *trampas*<sup>[32]</sup>.

**RF.T.1** La trampa debe activarse cuando el héroe esté posicionado sobre ella.

**RF.T.2** Cuando una trampa sea activada se reproducirá el sonido correspondiente a dicha acción.

**RF.T.3** La trampa tendrá un tiempo de retardo entre su activación y su animación de activación. En dicho periodo el héroe no sufrirá daño aunque esté posicionado sobre la trampa.

**RF.T.4** Cuando una trampa comience su animación de activación o desactivación se reproducirá el sonido correspondiente a dicha acción.

**RF.T.5** Una trampa activada debe reducir la vida del héroe de forma continuada si éste está posicionado sobre ella y ha finalizado su animación de activación.

**RF.T.6** La trampa no se desactiva mientras el héroe esté sobre ella.

**RF.T.7** La trampa se mantendrá activada durante un periodo determinado de tiempo cuando el héroe ya no esté sobre ella. La trampa se desactiva pasado ese periodo.

**RF.T.8** Cuando la trampa pase a estar desactivada se producirá una animación de ocultación descendente y la trampa quedará oculta.

**RF.IT Objeto**

El sistema debe proveer al usuario de objetos.

**RF.IT.1** El objeto debe ser consumido por el héroe cuando esté a una distancia concreta de él.

**RF.IT.2** La *poción*<sup>[24]</sup> no debe ser consumida por el héroe cuando éste tenga su barra de vida llena, aunque esté a la distancia correcta para consumir dicha poción.

**RF.IT.3** Cuando el héroe obtenga un cristal, su valor será sumado al total de cristales obtenidas por el héroe.

**RF.IT.4** Cuando el héroe consuma una poción aumentará su vida actual.

**RF.IT.5** Cuando el héroe consuma un objeto se reproducirá el sonido correspondiente a dicha acción.

#### **RF.AR Arma**

El sistema debe proveer al usuario de armas.

**RF.AR.1** El arma podrá ser de dos tipos, de cuerpo a cuerpo o a distancia

**RF.AR.2** El arma debe ser visualizada en el escenario.

#### **RF.CD Controlador de dispositivos**

El sistema debe proveer al usuario de un controlador de dispositivos.

**RF.CD.1** El sistema debe captar tanto los eventos de teclado como eventos de mando simultáneamente.

**RF.CD.2** El controlador debe detectar si hay un mando conectado cuando el sistema es inicializado.

**RF.CD.3** El controlador debe detectar la conexión del mando en tiempo de ejecución.

**RF.CD.4** El controlador debe detectar la desconexión del mando en tiempo de ejecución.

**RF.CD.5** El sistema debe visualizar una *notificación*<sup>[20]</sup> que indique al usuario la conexión/desconexión del mando.

#### **RF.C Cámara**

El sistema debe proveer al usuario de un sistema de cámara.

**RF.C.1** Debe permitir la visualización de la escena.

**RF.C.2** Debe seguir los movimientos del héroe cambiando tanto su posición como su objetivo.

**RF.C.3** Debe permitir al usuario usar el modo vista cuando pulse botón de vista.

**RF.C.4** No debe permitir el uso del modo vista cuando haya algún menú activado.

**RF.C.5** Cuando el modo vista esté activado, la cámara debe realizar un cambio gradual de su posición para realizar un movimiento de alejamiento en el eje Z con respecto al héroe.

**RF.C.6** Debe tener una posición máxima de alejamiento cuando está activado el modo vista.

#### **RF.N Notificación**

El sistema debe proveer al usuario de notificaciones.

**RF.N.1** Una notificación pasará al estado activado cuando otro objeto (mapa o partida) del sistema le de la señal de activación.

**RF.N.2** Una notificación activada debe ser visualizada durante un periodo determinado de tiempo.

**RF.N.3** Cuando una notificación activada consuma su tiempo de vida pasará al estado desactivado y dejará de ser visualizada.

#### **RF.P Perfil**

El sistema debe proveer al usuario de un perfil de rendimiento.

**RF.P.1** Perfil debe mostrar el tiempo medio para la visualización y para la actualización.

**RF.P.2** Perfil debe mostrar la media de frame por segundos del videojuego.

**RF.P.3** Perfil debe mostrar el tiempo de ejecución de nuestra aplicación.

### **RF.S Sombras**

El sistema debe proveer al usuario de sombras en la escena.

**RF.S.1** El sistema debe permitir la generación dinámica de sombras de todos los elementos que contenga nuestra escena.

**RF.S.2** El sistema no debe generar sombras de los contenedores de alma.

### **RF.SG Sistema de guardado**

El sistema debe proveer al usuario de un sistema de guardado.

**RF.SG.1** El sistema debe registrar el progreso del usuario ( mapa actual y cristales obtenidos) al superar un mapa.

### **RF.I Iluminación**

El sistema debe proveer al usuario de un sistema de iluminación.

**RF.I.1** El sistema debe visualizar el conjunto de luces que contenga el escenario.

## **5.2.1.1.2 Sección motor gráfico**

### **RF.W Window**

El motor gráfico debe proveer de un gestor de ventana

**RF.W.1** El motor gráfico debe permitir la creación de una ventana la cual contendrá una barra de título.

**RF.W.2** El motor gráfico debe permitir el cambio del tamaño de la ventana.

**RF.W.3** El motor gráfico debe permitir cambiar la ventana de modo (pantalla completa o modo ventana).

**RF.W.4** El motor gráfico debe permitir el cambio de nombre de la ventana.

**RF.W.5** El motor gráfico debe permitir el cambio de icono de la ventana.

**RF.W.6** El motor gráfico debe permitir ocultar o mostrar la ventana.

**RF.W.7** El motor gráfico debe permitir cerrar la ventana.

**RF.W.8** El motor gráfico debe permitir el refresco de la información que esté en la ventana.

### **RF.S Sonido**

El motor gráfico debe proveer de un gestor de sonido

**RF.S.1** El motor gráfico debe permitir cargar ficheros de audio en memoria.

**RF.S.2** El motor gráfico debe permitir la creación tanto de *música*<sup>[19]</sup> como de *efectos*<sup>[6]</sup>.

**RF.S.3** El motor gráfico debe permitir reproducir un sonido.

**RF.S.4** El motor gráfico debe permitir pausar un sonido que está en reproducción.

**RF.S.5** El motor gráfico debe permitir parar un sonido que está en reproducción.

**RF.S.6** El motor gráfico debe permitir resumir un sonido pausado.

**RF.S.7** El motor gráfico debe permitir el uso de diferentes canales para la reproducción de sonido.

**RF.S.8** El motor gráfico debe permitir cambiar el volumen de un sonido.

**RF.S.9** El motor gráfico debe permitir cambiar el volumen de un sonido según un parámetro de distancia.

#### **RF.MN Menú**

El motor gráfico debe proveer de varios tipos de menús.

**RF.MN.1** El motor gráfico debe permitir visualizar y actualizar cualquier tipo de menú.

**RF.MN.2** El motor gráfico debe permitir activar un menú en cualquier momento.

**RF.MN.3** El motor gráfico debe permitir el cambio de la textura de fondo de nuestro menú.

#### **RF.MP. Menú principal**

El motor gráfico debe proveer de la creación y uso de menús principales.

**RF.MP.1** El motor gráfico debe permitir añadir las opciones por defecto que mostrará el menú, así como la textura de dicha opción.

#### **RF.PA Menú de pausa**

El motor gráfico debe proveer de la creación y uso de menús de pausa.

**RF.PA.1** El motor gráfico debe permitir añadir las opciones por defecto que mostrará el menú, así como la textura de dicha opción.

#### **RF.MM Menú de muerte**

El motor gráfico debe proveer de la creación y uso de menús de muerte.

**RF.MM.1** El motor gráfico debe permitir añadir la textura frontal que mostrará la información de este menú.

#### **RF.PC Pantalla de carga**

El motor gráfico debe proveer de la creación y uso de pantallas de carga.

**RF.PC.1** El motor gráfico debe permitir añadir la textura frontal de la animación que mostrará el menú.

**RF.PC.2** El motor gráfico debe permitir cambiar el tiempo entre frames de la animación de carga.

#### **RF.MO Menú de opciones**

El motor gráfico debe proveer de la creación y uso de menús de opciones.

**RF.MO.1** El motor gráfico debe permitir añadir la textura frontal que mostrará la información de este menú.

#### **RF.MC Menú de controles**

El motor gráfico debe proveer de la creación y uso de menús de controles.

**RF.MC.1** El motor gráfico debe permitir añadir la textura frontal que mostrará la información de este menú.

#### **RF.CR Pantalla de créditos**

El motor gráfico debe proveer de la creación y uso de pantallas de créditos.

**RF.CR.1** El motor gráfico debe permitir añadir la textura frontal que mostrará la información de este menú.

**RF.CI Pantalla de cinemáticas**

El motor gráfico debe proveer de la creación y uso de pantallas de cinemáticas.

**RF.CI.1** El motor gráfico debe permitir añadir las texturas de cada cinemática.

**RF.IH Interfaz de héroe**

El motor gráfico debe proveer de la creación y uso de interfaces de héroe

**RF.IH.1** El motor gráfico debe proveer al programador de una interfaz que muestre la información del protagonista del videojuego.

**RF.SH Shader**

El motor gráfico debe proveer de un sistema de shader.

**RF.SH.1** El motor gráfico debe permitir la compilación del fragment shader y vertex Shader.

**RF.SH.2** El motor gráfico debe permitir el enlace del fragment shader y vertex shader (ambos compilados) a un programa.

**RF.SH.3** El motor gráfico debe permitir el uso de un shader compilado.

**RF.SH.4** El motor gráfico debe permitir la creación de fragments shader y vertexs shader a través de un fichero de texto.

**RF.TR Transformación**

El motor gráfico debe proveer de un sistema de transformaciones.

**RF.TR.ME Matrices estáticas:**

El motor gráfico debe permitir la creación de matrices independientes del tiempo.

**RF.TR.ME.1** El motor gráfico debe permitir realizar traslaciones sobre los objetos.

**RF.TR.ME.2** El motor gráfico debe permitir realizar rotaciones en cualquier eje sobre los objetos.

**RF.TR.ME.3** El motor gráfico debe permitir realizar escalados sobre los objetos.

**RF.TR.ME.4** El motor gráfico debe permitir crear transformaciones compuestas de dos o más transformaciones.

**RF.TR.MD Matrices dinámicas:**

El sistema debe permitir la creación de matrices dependientes del tiempo.

**RF.TR.MD.1** El motor gráfico debe permitir realizar movimientos lineales.

**RF.TR.MD.2** El motor gráfico debe permitir realizar movimientos rotatorios.

**RF.TR.MD.3** El motor gráfico debe permitir realizar movimientos oscilantes.

**RF.TR.MD.4** El motor gráfico debe permitir realizar movimientos estáticos en el tiempo.

**RF.TR.MCO Matrices compuestas:**

El motor gráfico debe permitir la creación de conjuntos de matrices dinámicas.

**RF.TR.MCO.1** El motor gráfico debe permitir resetear las matrices dinámicas de una matriz compuesta a su estado inicial.

**RF.TR.MCO.2** El motor gráfico debe permitir añadir una matriz dinámica junto a su tiempo de vida en una matriz compuesta.

**RF.TR.MCO.3** El motor gráfico debe permitir actualizar la matriz activa de una matriz compuesta al instante de tiempo actual.



**RF.AN Módulo de animación**

El motor gráfico debe proveer de un sistema de animación.

**RF.AN.1** El motor gráfico debe permitir añadir matrices compuestas en una animación.

**RF.AN.2** El motor gráfico debe permitir actualizar las matrices compuestas de una animación al instante de tiempo actual.

**RF.AN.3** El motor gráfico debe permitir resetear una animación a su estado inicial.

**RF.MR Motor Rendering**

El motor gráfico debe proveer de un motor de rendering.

**RF.MR.GE Grafo de escena**

**RF.MR.GE.1** El motor gráfico debe permitir añadir nodos a un grafo.

**RF.MR.GE.2** El motor gráfico debe permitir visualizar y actualizar un grafo.

**RF.MR.GE.3** El grafo debe permitir activar un grafo.

**RF.MR.M3 Modelo 3D**

**RF.MR.M3.1**-El motor gráfico debe permitir cargar una modelo 3D en la gpu.

**RF.MR.M3.2** El motor gráfico debe permitir visualizar una modelo 3D.

**RF.MR.M3.3** El motor gráfico debe permitir generar la caja englobante de una modelo 3D.

**RF.AM Agrupación de modelos.**

El motor gráfico debe proveer de un sistema de agrupación de modelos por material.

**RF.AM.1** El motor gráfico debe permitir indicar el material de la agrupación de modelos.

**RF.AM.2** El motor gráfico debe permitir añadir un nuevo modelo a la agrupación.

**RF.AM.3** El motor gráfico debe permitir cargar la agrupación de modelos en la gpu.

**RF.MT Material**

El motor gráfico debe proveer de la creación y uso de materiales.

**RF.MT.1** El motor gráfico debe permitir indicar los componentes ambiental, difusa y especular de un material.

**RF.MT.2** El motor gráfico debe permitir indicar la textura de un material.

**RF.MT.3** El motor gráfico debe permitir activar un material.

**RF.MT.T Textura.** El material debe proveer del uso de texturas.

**RF.MT.T.1** El motor gráfico debe permitir crear una textura a partir de una imagen.

**RF.MT.T.2** El motor gráfico debe permitir activar una textura.

**RF.BM Bump mapping**

El material debe proveer del uso de texturas de normales.

**RF.BM.1** El material debe permitir el uso de una textura de normales.

**RF.CL Colección**

El motor gráfico debe proveer de un sistema de colecciones.

**RF.CL.1** Una colección debe permitir obtener un recurso multimedia.

**RF.CFG Configuración**

El motor gráfico debe proveer de un sistema de configuración.

**RF.CFG.1** El sistema debe permitir registrar la configuración realizada por el usuario en un fichero json.

**RF.CFG.2** El sistema debe permitir obtener la configuración realizada por el usuario a partir de un fichero json.

### **RF.SG Sistema de guardado**

El motor gráfico debe proveer de un sistema de gestión del progreso de un usuario.

**RF.SG.1** El sistema debe permitir registrar el progreso del usuario en un fichero json.

**RF.SG.2** El sistema debe permitir obtener el progreso realizado por el usuario a partir de un fichero json.

### **RF.LM Lector de modelos**

El motor gráfico debe proveer de un lector de modelos.

**RF.LM.1** El sistema debe permitir la carga de modelos 3D a partir de ficheros con extensión .obj

**RF.LM.2** El sistema debe permitir calcular las normales de una malla a partir de sus vértices y triángulos.

**RF.LM.3** El sistema debe permitir calcular el centro de masas de una malla a partir de sus vértices.

**RF.LM.4** El sistema debe permitir el cálculo de la tangente y bitangente de una malla.

### **RF.N Notificaciones**

El motor gráfico debe proveer de un sistema de notificaciones.

**RF.N.1** Debe permitir el cambio del tiempo de vida de una notificación.

**RF.N.2** Debe permitir la selección de la posición, tamaño y textura de una notificación.

### **RF.TX Texto**

El motor gráfico debe proveer de un sistema de texto.

**RF.TX.1** Debe permitir cambiar la posición de un texto.

**RF.TX.2** Debe permitir seleccionar la fuente, color y tamaño del texto.

**RF.TX.3** Debe permitir el procesamiento de una cadena de texto y la creación de una textura con ella.

### **RF.P Perfil**

El motor gráfico debe proveer de un sistema de perfil de rendimiento.

**RF.P.1** El perfil debe permitir calcular el tiempo medio de visualización y actualización.

**RF.P.2** El perfil debe permitir contabilizar el número de frames que se han producido desde que se inició el videojuego.

**RF.P.3** El perfil debe permitir mostrar el perfil de rendimiento en un instante de tiempo.

### **RF.S Sombras**

El motor gráfico debe proveer de un generador de sombras arrojadas.

**RF.S.1** -El generador de sombras debe permitir cambiar su cámara de posición, objetivo e inclinación.

**RF.S.2** El generador de sombras debe permitir cambiar su proyección ortográfica.

**RF.S.3** El generador de sombras debe permitir cambiar el shader que utilizara.

**RF.S.4** El generador de sombras debe permitir generar la textura de profundidad.

**RF.S.5** El generador de sombras debe permitir activar la textura de profundidad.

**RF.RE Región** El motor gráfico debe proveer de un sistema de regiones.

**RF.RE.1** Una región debe permitir cambiar su posición, radioactividad y estado.

**RF.RE.RT Región de texto** El motor gráfico debe proveer de regiones de texto.

**RF.RE.RT.1** La región de texto debe permitir la actualización de su estado actual.

**RF.RE.RT.2** El región de texto debe permitir indicar el tiempo entre cada mensaje de dicha región.

**RF.RE.FM Región de finalización de mapa**

El motor gráfico debe proveer de regiones de finalización de mapa.

**RF.RE.FM.1** La *región de finalización de mapa*<sup>[27]</sup> debe permitir la actualización de su estado actual.

**RF.M Mapa**

El motor gráfico debe proveer de la creación y uso de mapas.

**RF.M.1** El mapa debe permitir visualizar y actualizar cada elemento que contiene.

**RF.M.2** El mapa debe permitir activar y desactivar el sonido del mapa.

**RF.M.3** El mapa debe permitir la detección de colisiones entre objetos.

**RF.M.4** El mapa debe permitir eliminar un elemento del mapa indexado para que no tenga colisión.

**RF.M.5** El mapa debe permitir su creación a través de un fichero json.

**RF.CM Carga de mapa**

El motor gráfico debe proveer de un cargador de mapas.

**RF.CM.1** El motor gráfico debe permitir la creación del mapa a través de una hebra.

**RF.CD Controlador de dispositivos**

El motor gráfico debe proveer de un controlador de dispositivos.

**RF.CD.1** El controlador de dispositivos debe permitir el consumo de los eventos que estén activos en ese momento.

**RF.CD.2** El controlador de dispositivos debe permitir actualizar los eventos de teclado y mando.

**RF.CD.3** El controlador de dispositivos debe permitir cambiar el estado de un botón.

**RF.C Cámara**

El motor gráfico debe proveer de un sistema de cámara.

**RF.C.1** Debe permitir la creación y activación de una cámara.

**RF.C.2** Debe permitir el cambio de posición, objetivo e inclinación.

**RF.C.3** Debe permitir la creación y activación de una proyección perspectiva.

**RF.C.4** Debe permitir la creación y activación de una proyección ortográfica.

**RF.I Iluminación**

El motor gráfico debe proveer de un sistema de iluminación.

**RF.I.1** El sistema debe permitir la creación de luces puntuales.

**RF.I.2** El sistema debe permitir la creación de luces direccionales.

**RF.I.3** El sistema debe permitir activar múltiples luces simultáneamente.

#### **RF.IE IA enemigo cuerpo a cuerpo**

El motor gráfico debe proveer de una IA<sup>[14]</sup> de enemigo cuerpo a cuerpo.

**RF.IE.1** La IA debe permitir indicar el próximo movimiento de un enemigo cuerpo a cuerpo.

#### **RF.ID IA enemigo a distancia**

El motor gráfico debe proveer de una IA de enemigo a distancia.

**RF.ID.1** La IA debe permitir indicar el próximo movimiento de un enemigo a distancia.

#### **RF.IN IA Npc**

El motor gráfico debe proveer de una IA de npc.

**RF.IN.1** El motor gráfico debe permitir añadir nuevos estados a una IA.

**RF.IN.2** El motor gráfico debe permitir obtener el estado actual de una IA.

**RF.IN.3** El motor gráfico debe permitir cambiar el estado actual de una IA.

**RF.IN.4** El motor gráfico debe permitir eliminar todos los estados de una IA.

#### **RF.SP Sistema de partículas**

El motor gráfico debe proveer de sistemas de partículas.

**RF.SP.1** El motor gráfico debe permitir indicar el material, número de partículas máxima y el rango posible de valores de su posición, velocidad y tiempo de vida iniciales.

### **5.2.1.2 Requisitos no funcionales**

**RNF.1** El sistema debe tener un correcto funcionamiento en todos los aspectos que contiene tanto en Window como en Linux.

**RNF.2** El sistema debe tener una tasa de frame estable de aproximadamente 50-60 frames por segundo, con independencia del sistema operativo.

**RNF.3** El sistema no debe tardar más de 30 segundos en realizar la carga inicial.

**RNF.4** El sistema no debe tardar más de 5 segundos en cargar un escenario.

**RNF.5** El sistema no debe tener ninguna disminución de rendimiento (disminución de los frames por segundos) durante su uso.

**RNF.6** El sistema debe reaccionar de forma correcta cuando el usuario realice modificaciones sobre la ventana (minimizar, maximizar, arrastre y pantalla completa). Este tipo de acciones no deben producir un efecto negativo sobre el usuario durante su uso.

**RNF.7** El videojuego debe reaccionar de forma correcta a la suspensión del sistema operativo. Cuando éste sea activado, el videojuego debe seguir en funcionamiento de forma correcta.

**RNF.8** El sistema debe ser intuitivo y fácil de usar por cualquier tipo de usuario.

**RNF.9** El sistema debe tener unos controles cómodos e intuitivos para el usuario.

**RNF.10** El sistema debe permitir el uso de mando y de teclado para captar los eventos del usuario.

### 5.2.1.3 Requisitos del sistema

#### Requisitos mínimos

Para que el videojuego opere de forma correcta, es decir, tenga una fluidez superior a los 30 frames por segundo, los requisitos mínimos son:

- **Sistema operativo:** Window 7/8/10 64 bits , Linux 64 bits
- **Procesador:** Intel core 2 Quad 2.50GHz o AMD equivalente
- **Memoria:** 3 GB de RAM
- **Gráficos:** Nvidia GeForce GT 440, Intel HD Graphics 4600
- **OpenGL:** 3.0 o superior
- **Almacenamiento:** 100 mb de espacio disponible.

#### Requisitos recomendados

Para que el videojuego opere de una forma óptima, es decir, tenga una fluidez superior a los 60 frames por segundo, los requisitos mínimos son:

- **Sistema operativo:** Window 7/8/10 64 bits , Linux 64 bits
- **Procesador:** Intel core i7 2.00GHz o AMD equivalente
- **Memoria:** 3 GB de RAM
- **Gráficos:** Nvidia GeForce GT 840m, Intel HD Graphics 4600
- **OpenGL:** 3.0 o superior
- **Almacenamiento:** 100 mb de espacio disponible.

### 5.2.2 Casos de uso

Debido a que tanto el héroe como los otros tipos de avatares tienen un gran número de estados, las combinaciones de los estados de los distintos elementos nos dan una cantidad de interacciones demasiado elevada como para ser reflejados mediante casos de uso.

### 5.2.3 Diagrama conceptual

El diagrama conceptual nos permitirá tener una visión general del sistema. De esta forma podremos comprender mejor su estructura.

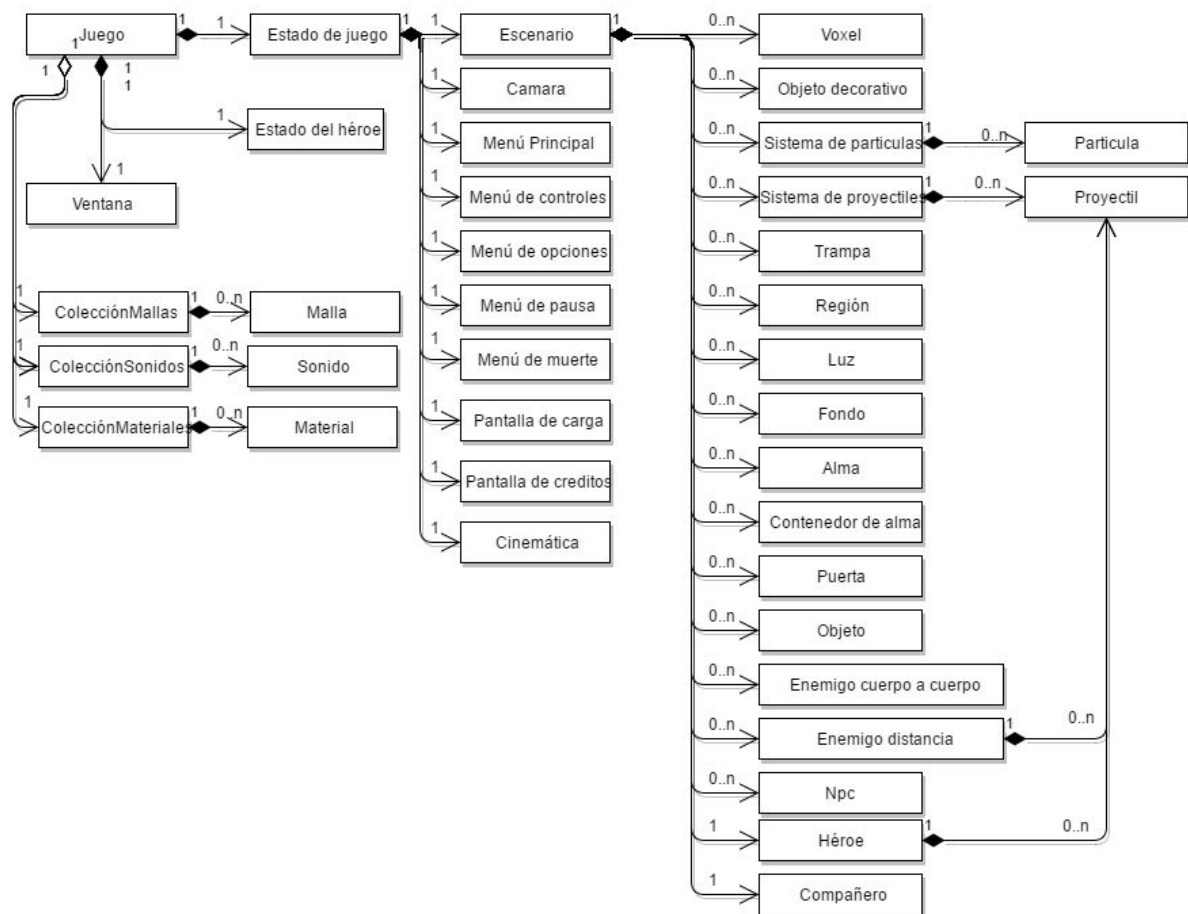


Figura 5.2: Diagrama conceptual del proyecto

### 5.2.4 Diagrama de clases

Los diagramas de clases nos permiten representar cada componente del sistema, así como las relaciones que hay entre ellos. Dado que nuestro sistema es amplio en cuanto a número de elementos, se ha realizado una división del diagrama de clases en diferentes componentes para su mejor entendimiento.

Los diagramas no han sido introducidos en esta sección debido a que su tamaño dificulta su visualización. Los diagramas de clases pueden encontrarse en el Anexo 1. No obstante, hay un diagrama de clases que muestra la jerarquía de herencia organizada por componentes.

### Lista de componentes

Un componente es una agrupación de clases según su funcionalidad. Por ejemplo, un componente podría ser el sistema de animación que contiene todas las clases relativas a dicho sistema.

#### Partida

Agrupar los elementos necesarios para gestionar el estado de los diferentes elementos del juego actual.

La clase principal del juego ("Game") será la que gestione cada elemento del sistema, indicando qué elementos se actualizan y cuáles no en cada momento.

Contiene las siguientes clases: *game*, *GameState*, *Herostate*, *Context*, *Window*, *Notification*, *ShadowManager*.

### **Objetos 3D**

Representa los diferentes objetos 3D del sistema. Cada uno de ellos debe poder visualizarse y actualizarse en el sistema.

La clase "Object3D" es abstracta y de ella heredan todos los objetos que deben ser visualizados en la escena.

Contiene las siguientes clases: *Object3D*, *ProjectileSystem*, *RootMap*, *EnemyList*, *Weapon*, *Menú*, *HeroState*, *VoxelGroup*, *Mesh*, *ObjectGroup*, *Notification*, *Item*, *SpikeTrap*, *NodeSceneGraph*, *Particle*, *ParticleSystem*, *ObjectScene*, *Soul*, *SoulCarrier*, *Avatar*, *Text*.

### **Avatar**

Agrupar los diferentes elementos relacionados con los avatares. La clase Avatar es abstracta y de ella heredan todos los avatares de nuestro sistema. Además, contiene la clase relativa al arma que llevarán algunos tipos de avatares (héroe y enemigo).

Contiene las siguientes clases: *Avatar*, *AvatarMove*, *Hero*, *Npc*, *Mate*, *Weapon*, *Enemy*, *Projectile*.

### **Enemigo**

Agrupar los diferentes elementos relacionados con los enemigos: tanto los distintos tipos de enemigos como las inteligencias artificiales que utilizan.

Un enemigo es un tipo de avatar que debe activarse cuando el héroe esté en su rango de actividad.

Contiene las siguientes clases: *EnemyList*, *Enemy*, *MeleeEnemy*, *RangedEnemy*, *IAEnemy*, *IARangedEnemy*, *IAEnemy*.

### **Npc**

Agrupar los diferentes elementos relacionados con los npcs.

Un npc es un tipo de avatar que interacciona con el héroe a través de mensajes.

Contiene las siguientes clases: *Npc*, *IANpc*, *Text*.

### **Menú**

Agrupar los diferentes menús del sistema, como el menú principal, pantalla de créditos y pantalla de carga, entre otros.

Un menú se visualizará en el sistema cuando presente el estado activado. Cada tipo de menú tiene una funcionalidad distinta.

Contiene las siguientes clases: *Menu*, *CreditScreen*, *ControlMenu*, *MovieScreen*, *LoadingScreen*, *OptionsMenu*, *PauseMenu*, *MainMenu*, *DeadMenu*.

### **Matrices**

Agrupar las diferentes tipos de matrices del sistema. Se pueden clasificar en matrices estáticas, dinámicas y compuestas. Cada tipo de matriz actualiza su información de diferente forma.

Contiene las siguientes clases: *Matrix4f*, *Matrix4fDynamic*, *LinearMovement*, *MatrixStatic*, *AccelerateMovement*, *OscillateRotation*, *MatrixScript*, *AxisRotation*.

### **Animación**

Agrupar los diferentes elementos relacionados con el sistema de animaciones. Dicho sistema contiene un conjunto de matrices compuestas que se actualizan dependiendo del instante de tiempo en el que se encuentren.

Contiene las siguientes clases: *ScriptAnimation*, *Matrix4fDynamic*, *ScriptLMD*, *MatrixScript*, *AnimatinList*.

### **Contexto**

Es un contenedor que almacena la información necesaria para visualizar cada objeto. Entre otros elementos almacena la pila de matrices y la de materiales así como el shader actual.

Contiene las siguientes clases: *Context*, *Shader*, *MatrixStack*, *MaterialStack*.

### **Estado de juego y cámara**

Representa el estado actual de la partida en curso y el estado de la cámara.

Contiene las siguientes clases: *GameState*, *Camera*.

### **Objetos en la escena**

Un objeto en la escena representa a un objeto 3D al cual se le han aplicado unas transformaciones y tiene una localización espacial en nuestra escena.

Contiene las siguientes clases: *ObjectScene*, *DecorationObject*, *Voxel*, *Door*.

### **Escena**

Agrupar los principales elementos para la creación y visualización de un mapa (escena).

Contiene las siguientes clases: *LoaderThread*, *RootMap*, *NodeSceneGraph*, *EntryNGE*.

### **Sistema de puertas**

Agrupar a los diferentes elementos que gestiona la lógica del sistema de puertas.

Contiene las siguientes clases: *Door*, *Soul*, *SoulCarrier*.



**Sistema de partículas/proyectiles y objetos**

Agrupar diferentes sistemas de la aplicación como pueden ser los sistemas de proyectiles y partículas, el sistema de objetos y las diferentes trampas de la escena.

Contiene las siguientes clases: *ItemList*, *Item*, *Projectile*, *ProjectileSystem*, *SpikeTrap*, *ParticleSystem*, *particle*.

**Malla**

Agrupar a los principales elementos relacionado con la carga y visualización de modelos 3D.

Contiene las siguientes clases: *Mesh*, *FileObj*, *ObjectGroup*.

**Material**

Agrupar los diferentes elementos del material, los cuales son los diferentes componentes del material en sí y la textura que tiene dicho material.

Contiene las siguientes clases: *Material*, *Texture*.

**Iluminación**

Agrupar los elementos del sistema de iluminación. Los diferentes tipos de luz son : luz puntual y luz direccional.

Contiene las siguientes clases: *Light*, *PointLight*, *DirectionalLight*.

**Sombras**

Agrupar los diferentes elementos del sistema de sombras, el cual generará y visualizará las sombras de los objetos de nuestra escena.

Contiene las siguientes clases: *ShadowManager*, *ShadowTexture*.

**Región**

Agrupar los diferentes tipos de regiones que nuestra escena puede contener. Estos tipos son: región de texto y región de fin de mapa.

Contiene las siguientes clases: *Region*, *TextRegion*, *EndMapRegion*.

**Sonido**

Agrupar los diferentes tipos de sonidos. Estos tipos son: música y efecto.

Contiene las siguientes clases: *Sound*, *Music*, *Effect*.

**Colección**

La finalidad de este componente es la no duplicidad de recursos multimedia en el sistema, mediante el uso compartido de estos recursos a través de la colección específica. Los diferentes tipos de colección son : colección de mallas, de materiales y de sonidos.

Contiene las siguientes clases: *Collection*, *MeshCollection*, *MaterialCollection*, *SoundCollection*.

### Gestor de ficheros

Agrupar los elementos que gestionan los diferentes ficheros del sistema. Estos ficheros pueden ser ficheros de configuración o ficheros de progreso del juego.

Contiene las siguientes clases: *OptionManager*, *SavedManager*.

### Controlador

Agrupar los diferentes elementos del controlador de dispositivos. Su funcionalidad es la de gestionar los eventos tanto de teclado como de mando de consola.

Contiene las siguientes clases: *Controller*, *GamepadController*, *KeyboardController*, *ControllerManager*.

## Jerarquía de herencia

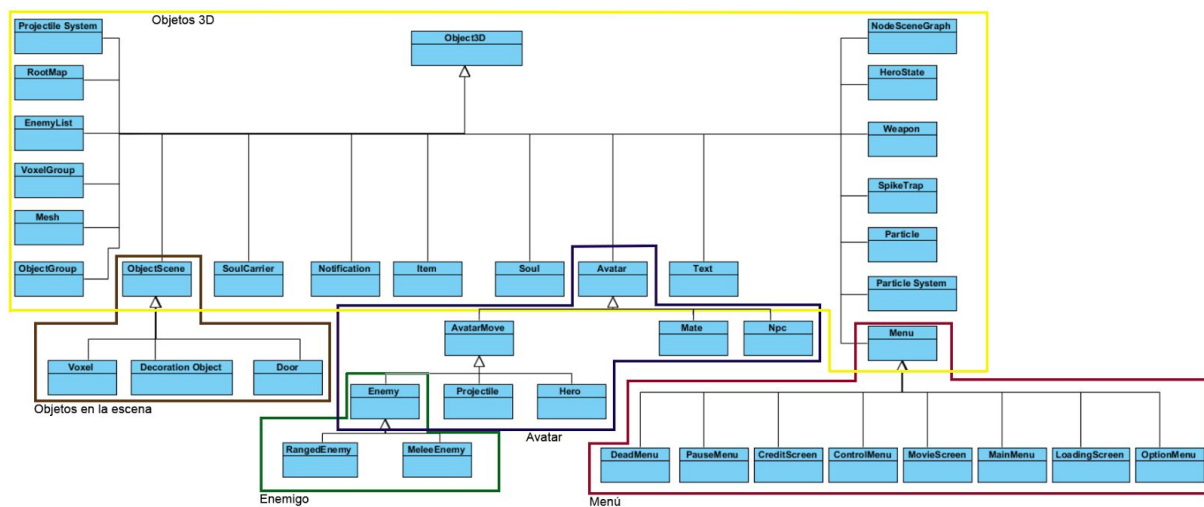


Figura 5.3: Jerarquía de herencia Objeto 3D

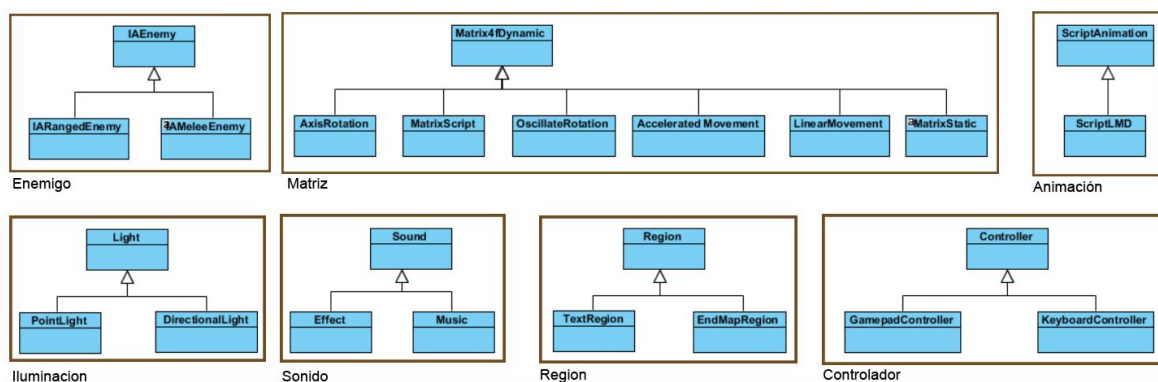


Figura 5.4: Jerarquía de herencia

### 5.2.5 Diagrama de estados

Los diagramas de estados nos permiten representar los estados de un objeto y las condiciones para pasar de un estado a otro. Solo se ha realizado el diagrama estados de los objetos los cuales tenían un conjunto de estados relevante para ser representado en un diagrama.

Las transiciones entre estados se han indicado en términos de eventos. Los principales eventos provocados por el usuario se muestran en la tabla 5.9. En esta misma tabla se indican las teclas que desencadenan cada evento.

| Evento         | Teclado | Mando              |
|----------------|---------|--------------------|
| Movimiento     | W/A/S/D | Joystick izquierdo |
| Acción         | E       | A                  |
| Salto          | K       | B                  |
| Ataque         | L       | R1                 |
| Escudo         | i       | L1                 |
| Cambio de arma | Q       | Y                  |
| Pausa          | Esc     | Start              |
| Vista          | V       | Back               |

Tabla 5.9: Eventos del sistema

## Partida

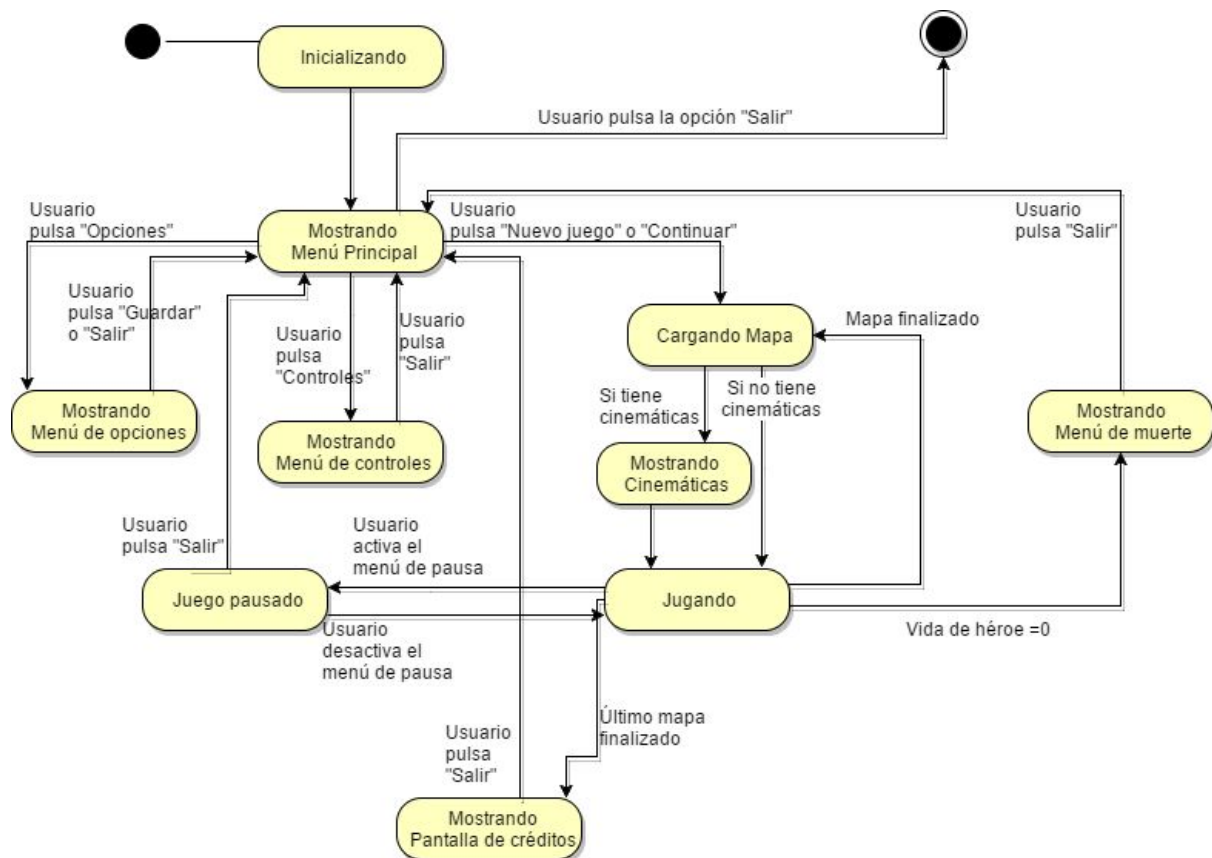


Figura 5.5: Diagrama de estados de una partida

## Héroe

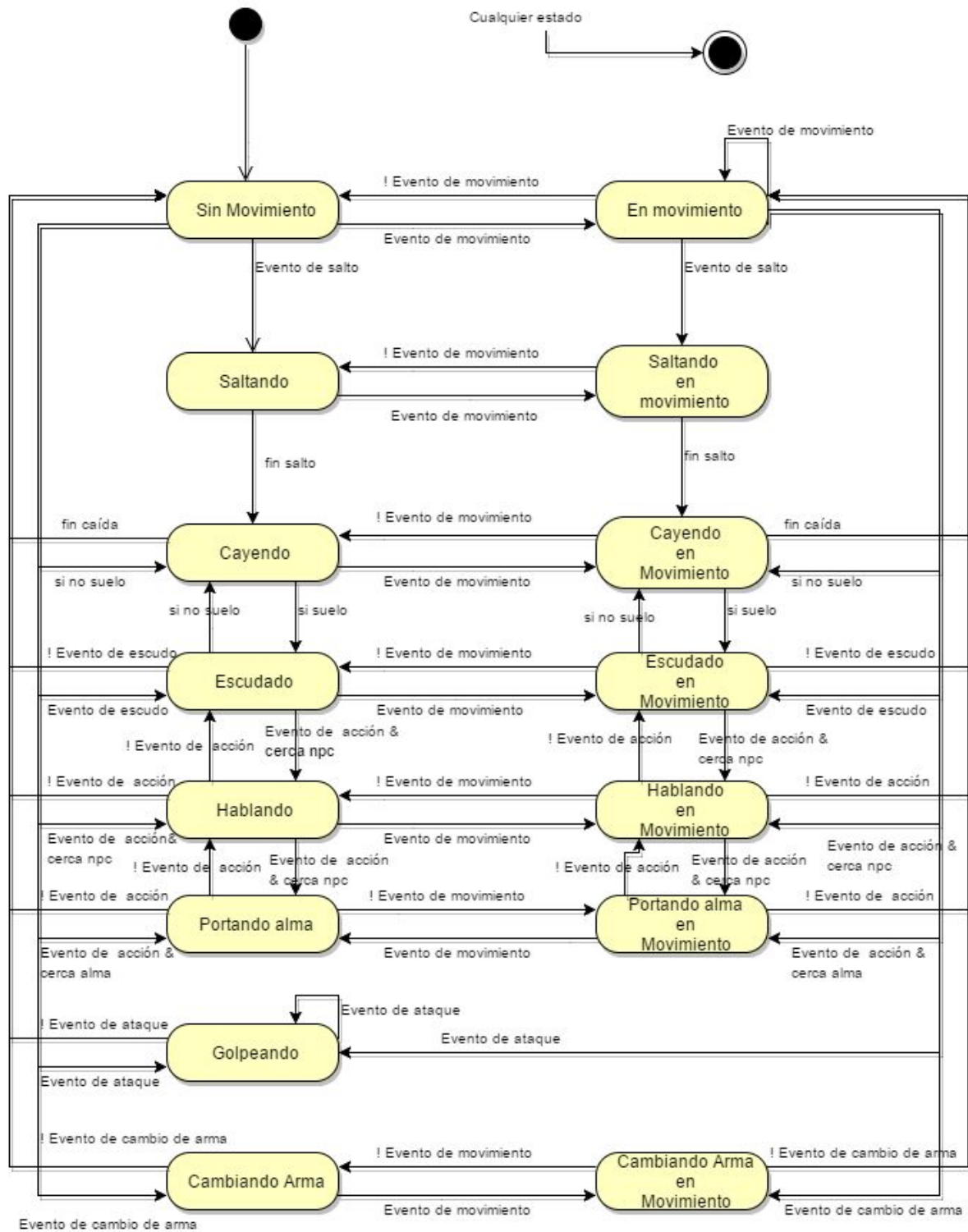


Figura 5.6: Diagrama de estados del héroe

### Interacción del héroe con el entorno

En este diagrama de estados podemos ver parte de la interacción con el entorno del héroe. El estado inicial de esta interacción es denominado “estado actual” y podríamos describirlo como un estado abstracto, el cual puede ser cualquier estado del héroe excepto el estado “golpeado” o el estado “movimiento de impacto”.

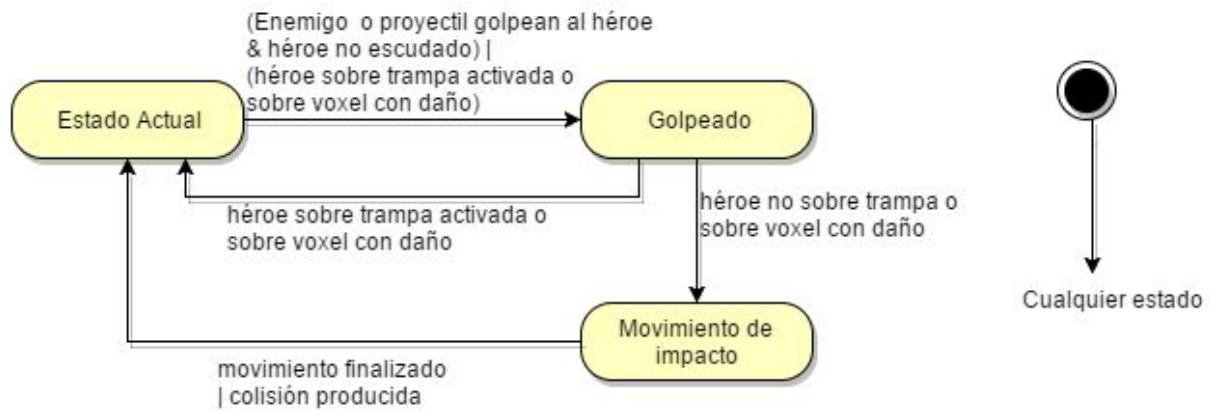


Figura 5.7: Diagrama de estados de la interacción del héroe con el entorno

### Enemigo cuerpo a cuerpo

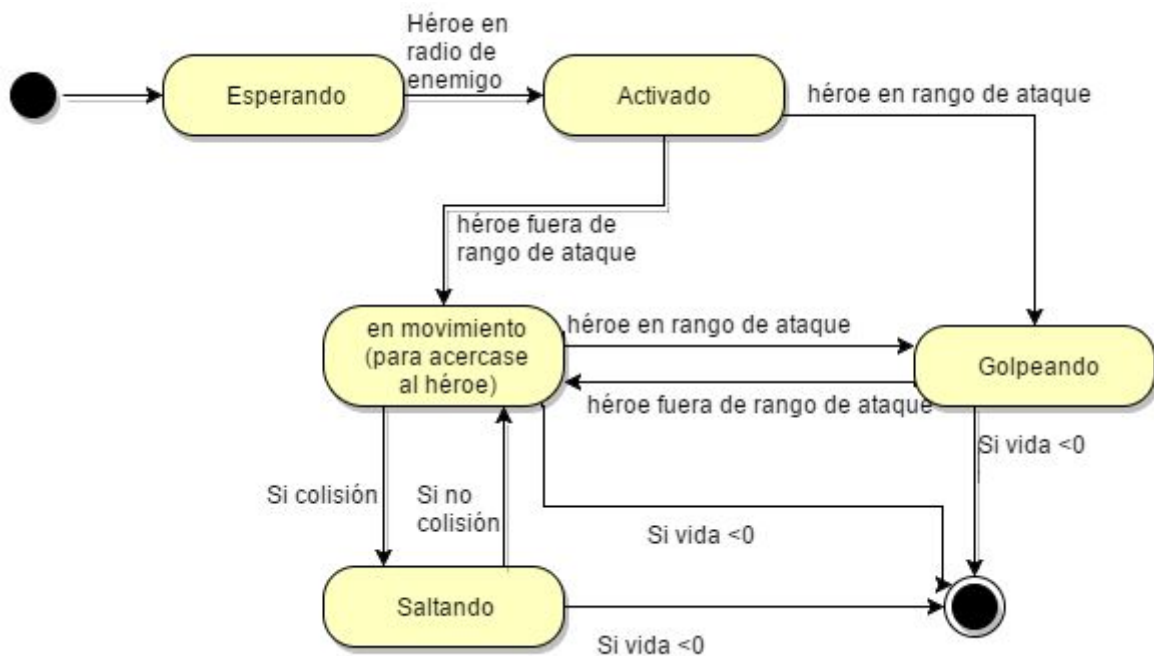


Figura 5.8: Diagrama de estados del enemigo cuerpo a cuerpo

## Enemigo a distancia

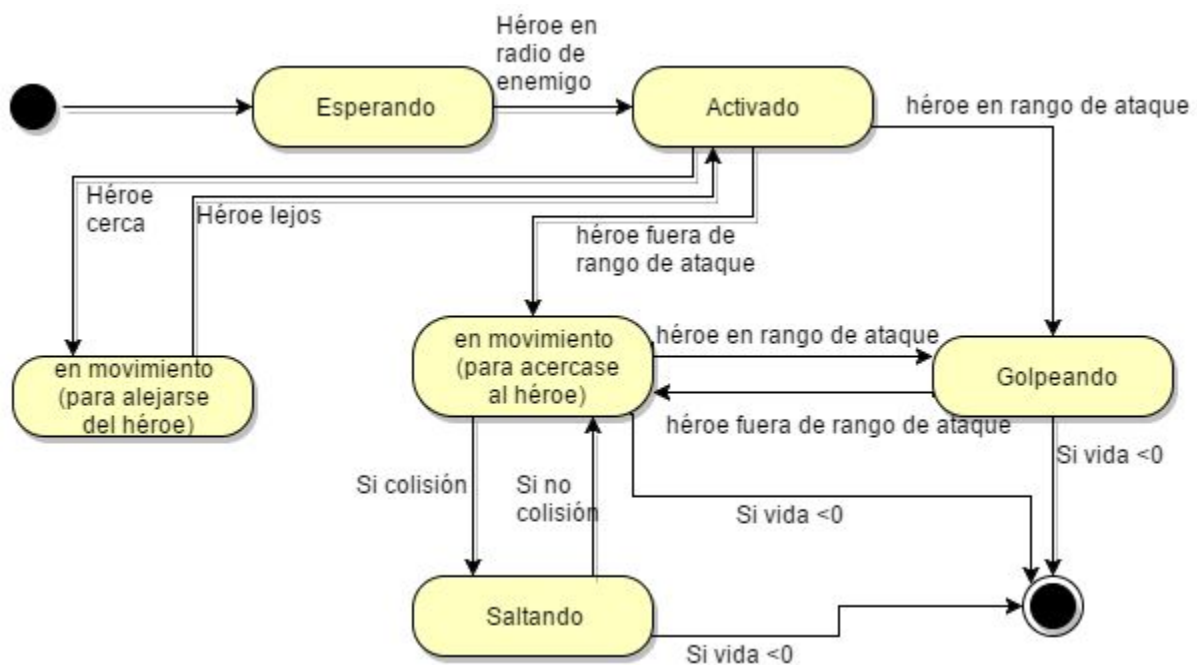


Figura 5.9: Diagrama de estados del enemigo a distancia

## Interacción del enemigo con el entorno

En este diagrama de estados podemos ver parte de la interacción con el entorno del enemigo. El estado inicial de esta interacción es denominado "estado actual" y podríamos describirlo como un estado abstracto el cual puede ser cualquier estado del enemigo excepto el estado "golpeado" o el estado "movimiento de impacto".

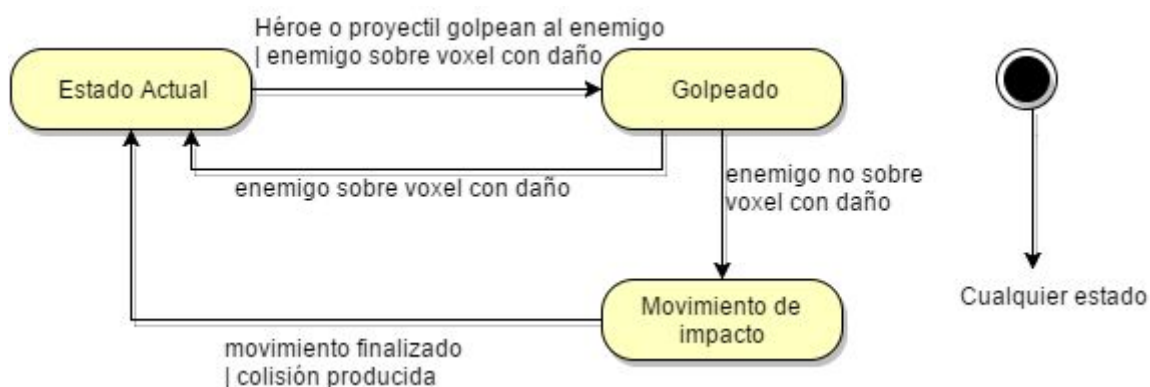


Figura 5.10: Diagrama de estados de la interacción del enemigo con el entorno

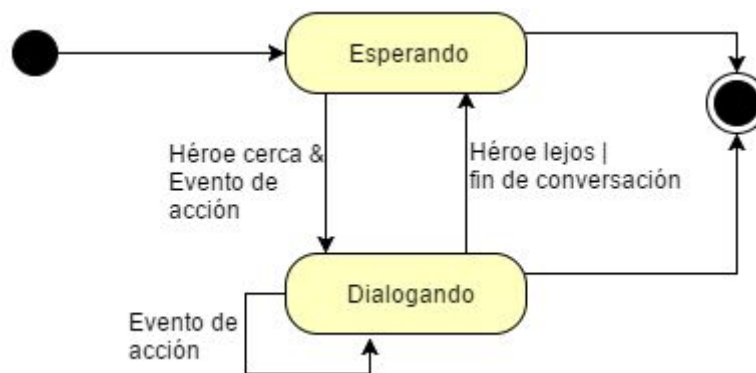
**Npc**

Figura 5.11: Diagrama de estados del npc

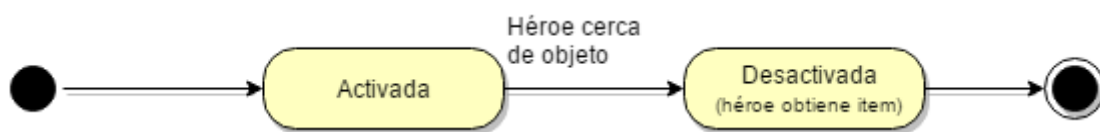
**Objeto**

Figura 5.12: Diagrama de estados del objeto

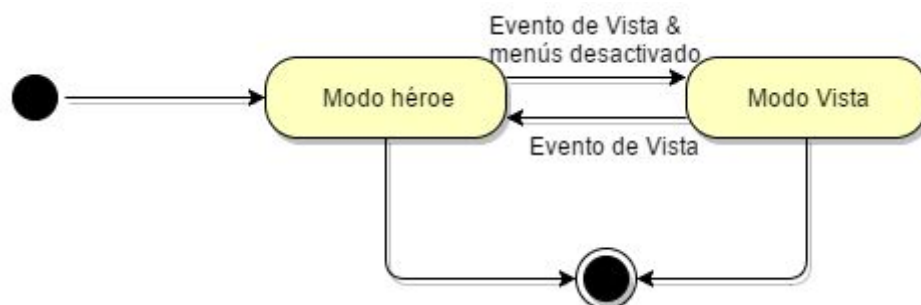
**Cámara**

Figura 5.13: Diagrama de estados de la cámara



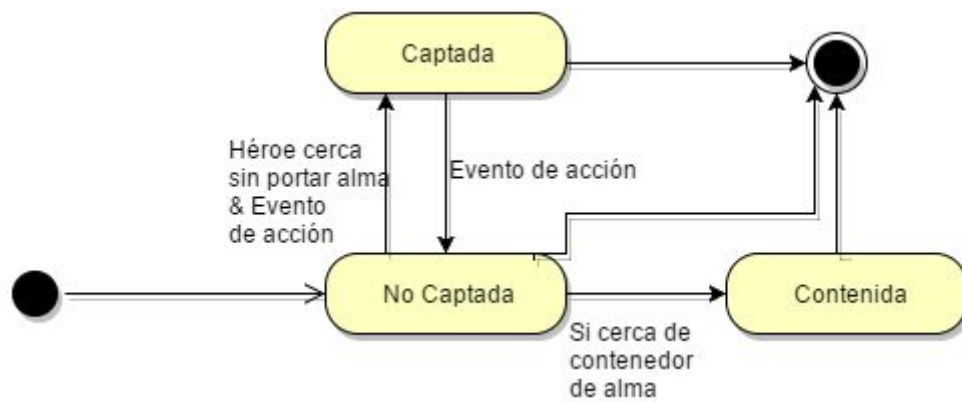
**Alma**

Figura 5.14: Diagrama de estados del alma

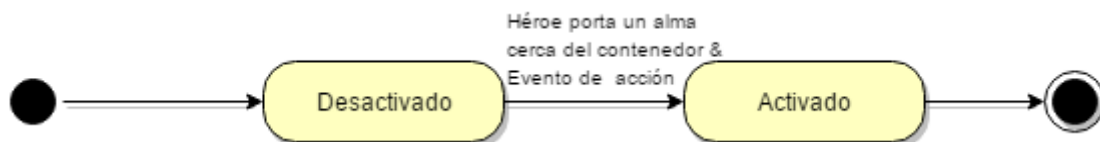
**Contenedor de alma**

Figura 5.15: Diagrama de estados del contenedor de alma

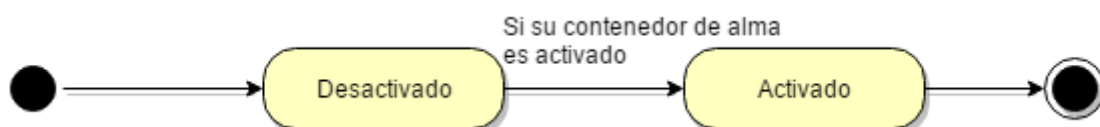
**Puerta**

Figura 5.16: Diagrama de estados de la puerta

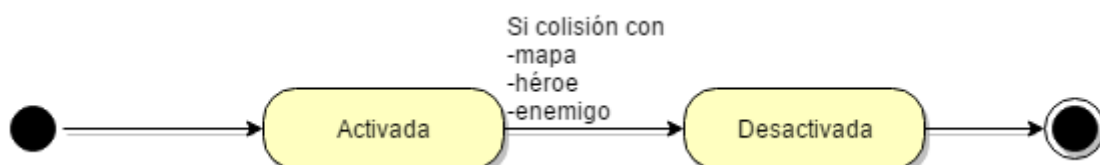
**Proyectil**

Figura 5.17: Diagrama de estados del proyectil

### Sistema de proyectiles

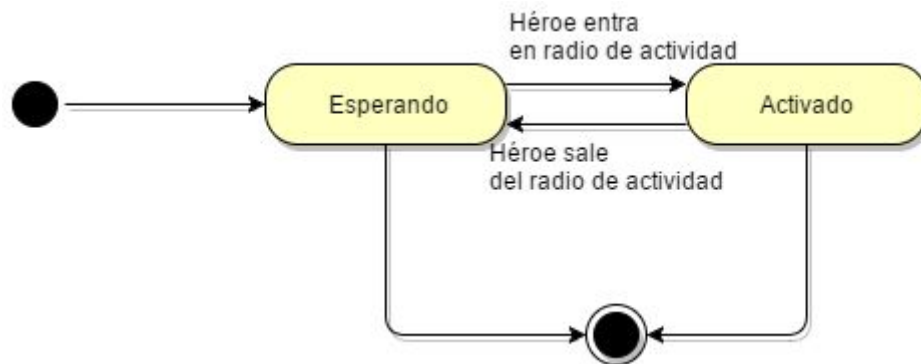


Figura 5.18: Diagrama de estados del sistema de proyectiles

### Trampa

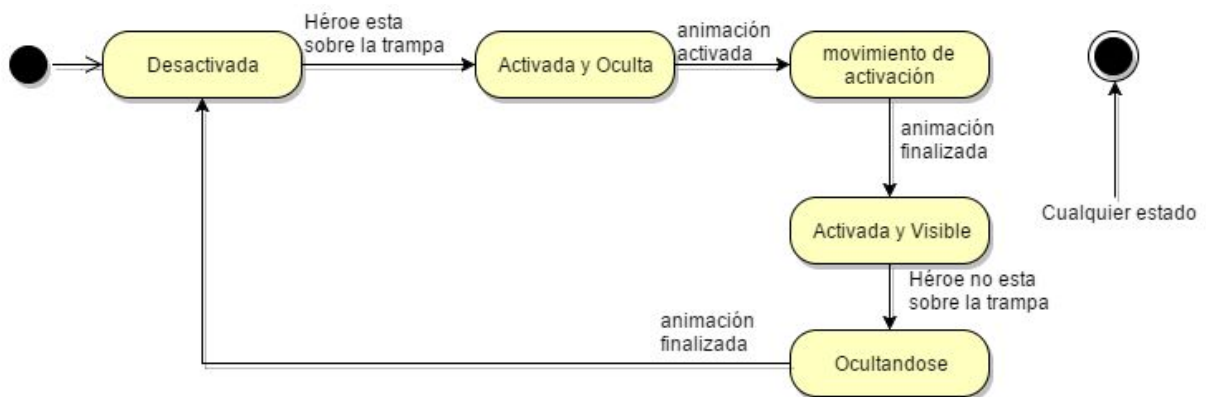


Figura 5.19: Diagrama de estados de la trampa

### 5.2.6 Diagrama de secuencia

Debido a que tanto el héroe como los otros tipos de avatares tienen un gran número de estados, las combinaciones de los estados de los distintos elementos nos dan una cantidad de interacciones demasiado elevada como para ser reflejados mediante diagramas de secuencia.

### 5.2.7 Bocetos de las interfaces de usuario

Esta sección incluye los bocetos previos que se han utilizado como referencia para el desarrollo de las interfaces de usuario.

#### Menú principal

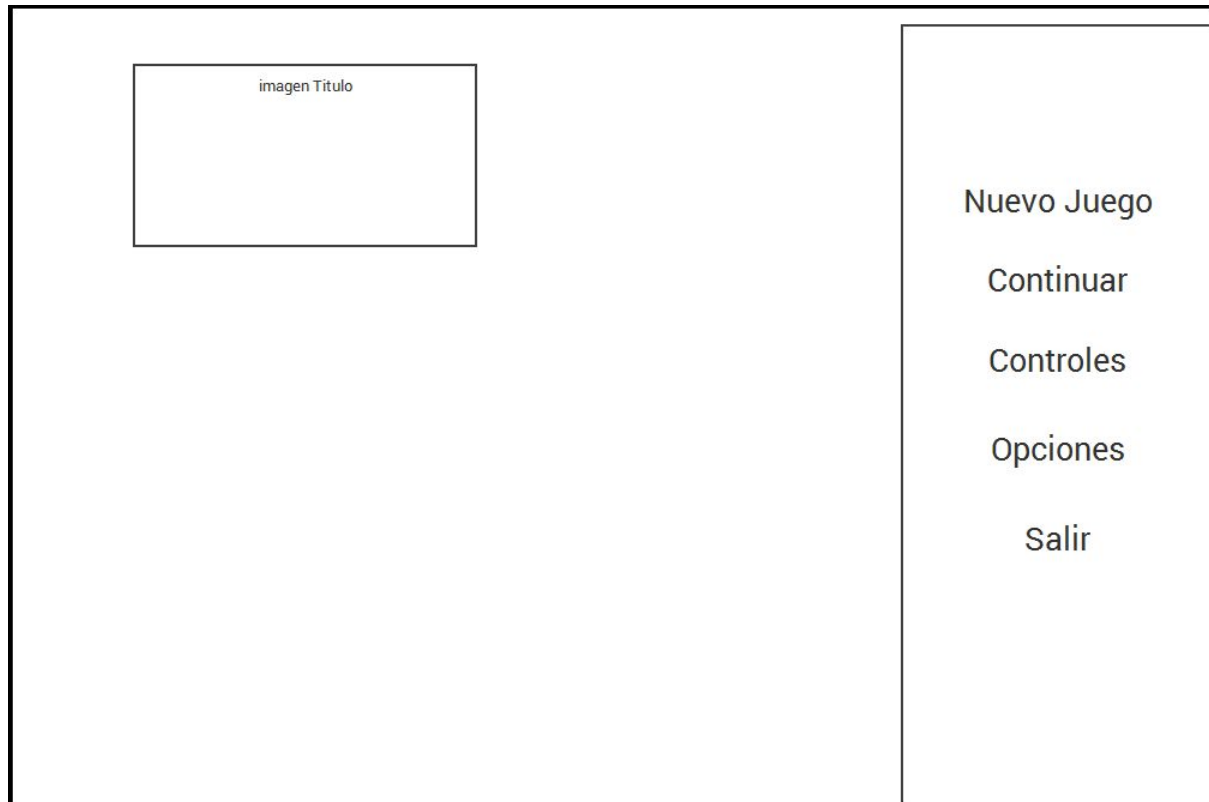


Figura 5.20: Boceto de menú principal

#### Lista de componentes

- **Botón nuevo juego:** al pulsar este botón el juego empezará a cargar el mapa inicial y pasará a la pantalla de carga.
- **Botón continuar:** nos empezará a cargar el mapa actual en el que está el jugador y pasará a la pantalla de carga. Si el jugador no tiene un progreso en el juego, este botón no tendrá ninguna acción
- **Botón controles:** al pulsarlo llevará al usuario al menú de controles.
- **Botón opciones:** al pulsarlo llevará al usuario al menú de opciones.
- **Botón salir:** al pulsar este botón la aplicación se cerrará.

## Menú de opciones

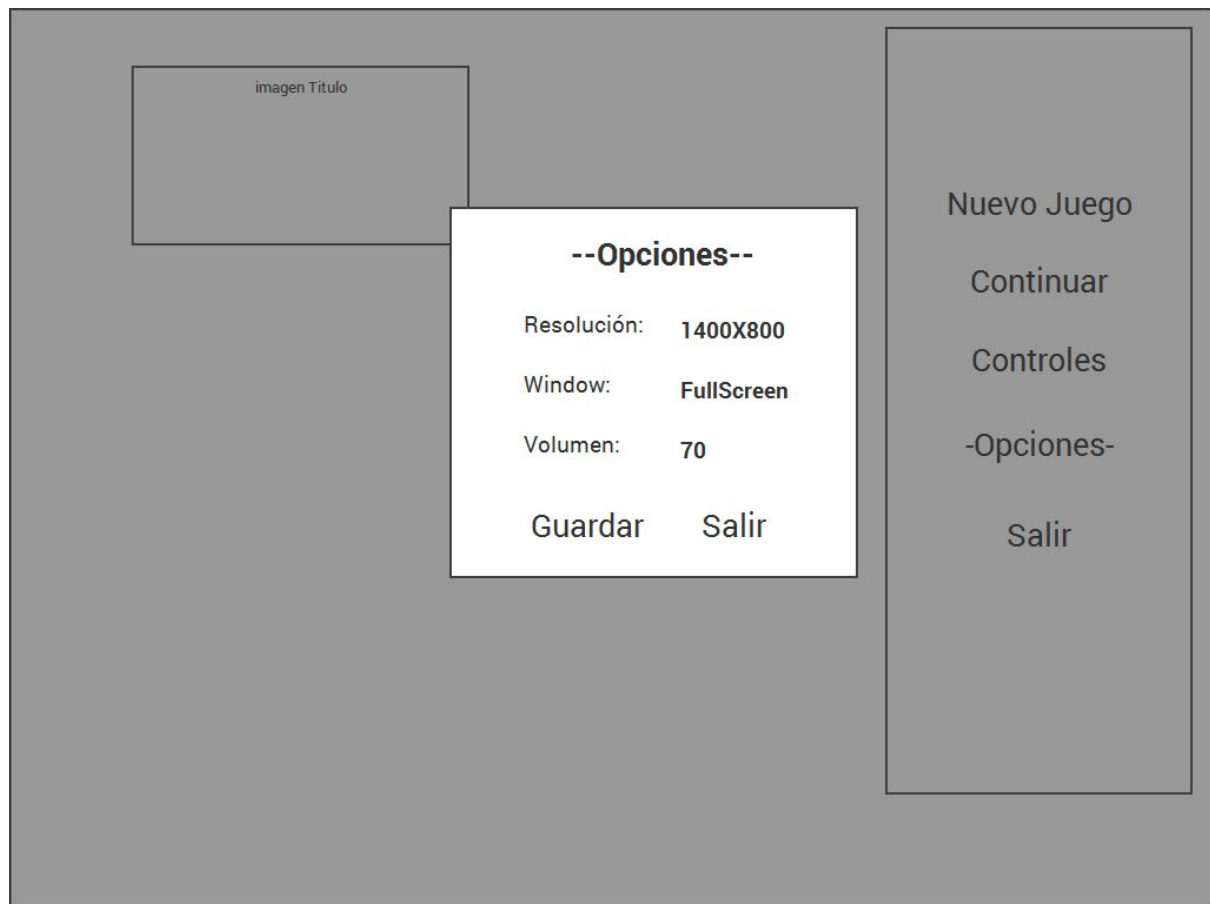


Figura 5.21: Boceto de menú de opciones

## Lista de componentes

- **Opción resolución:** con esta opción podremos cambiar la resolución actual de nuestro videojuego.
- **Opción ventana:** con esta opción indicaremos si la aplicación estará en modo pantalla completa o en modo ventana.
- **Opción volumen:** con esta opción el usuario podrá cambiar el volumen general de nuestra aplicación.
- **Botón guardar:** al pulsarlo el sistema guardará los cambios en la configuración y volverá al menú principal.
- **Botón salir:** al pulsarlo el sistema no guardará los cambios en la configuración y volverá al menú principal.

## Menú de controles

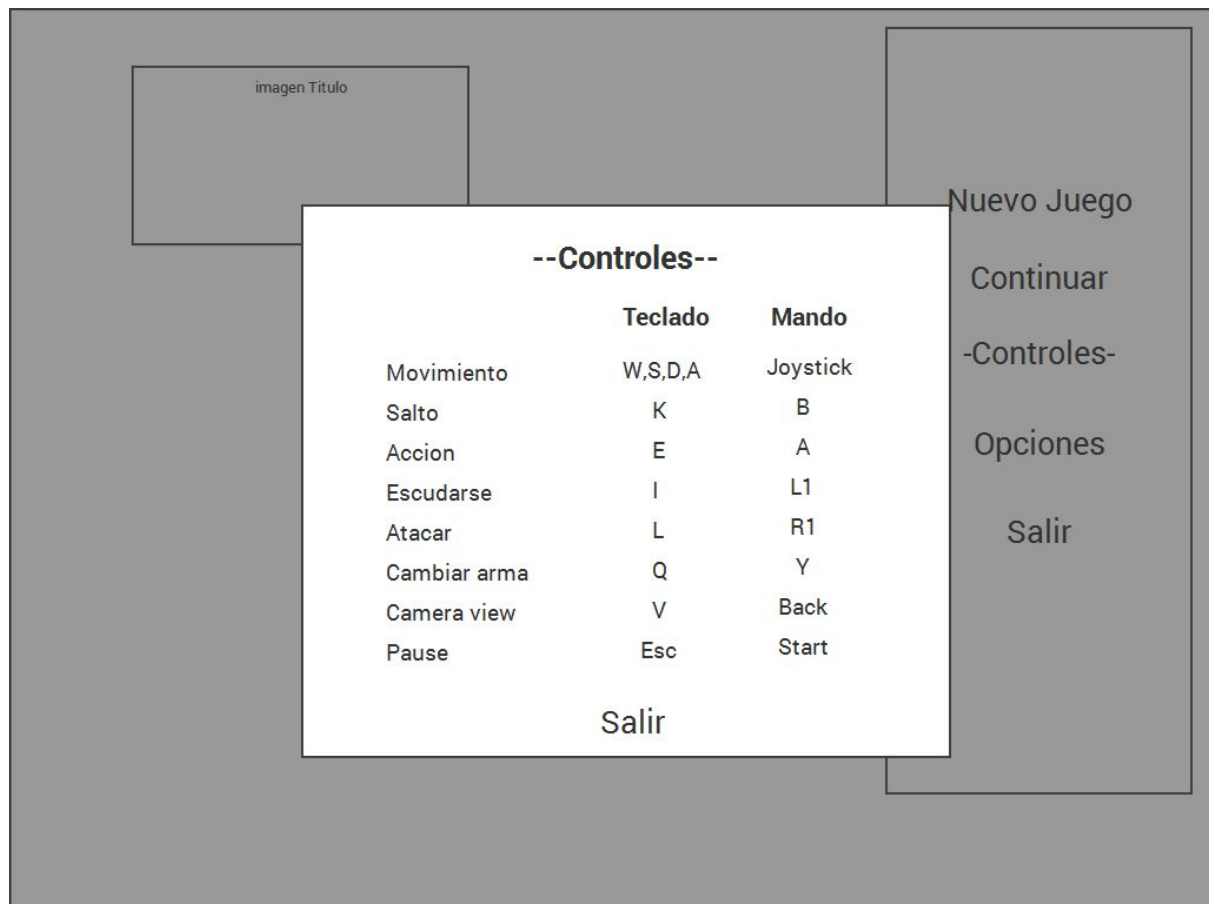


Figura 5.22: Boceto de menú de controles

## Lista de componentes

- **Controles mando:** mostrará los controles de nuestro mando para cada acción posible del héroe.
- **Controles teclado:** mostrará los controles de nuestro teclado para cada acción posible del héroe.
- **Botón salir:** al pulsarlo llevará al usuario al menú principal.

### Menú de pausa

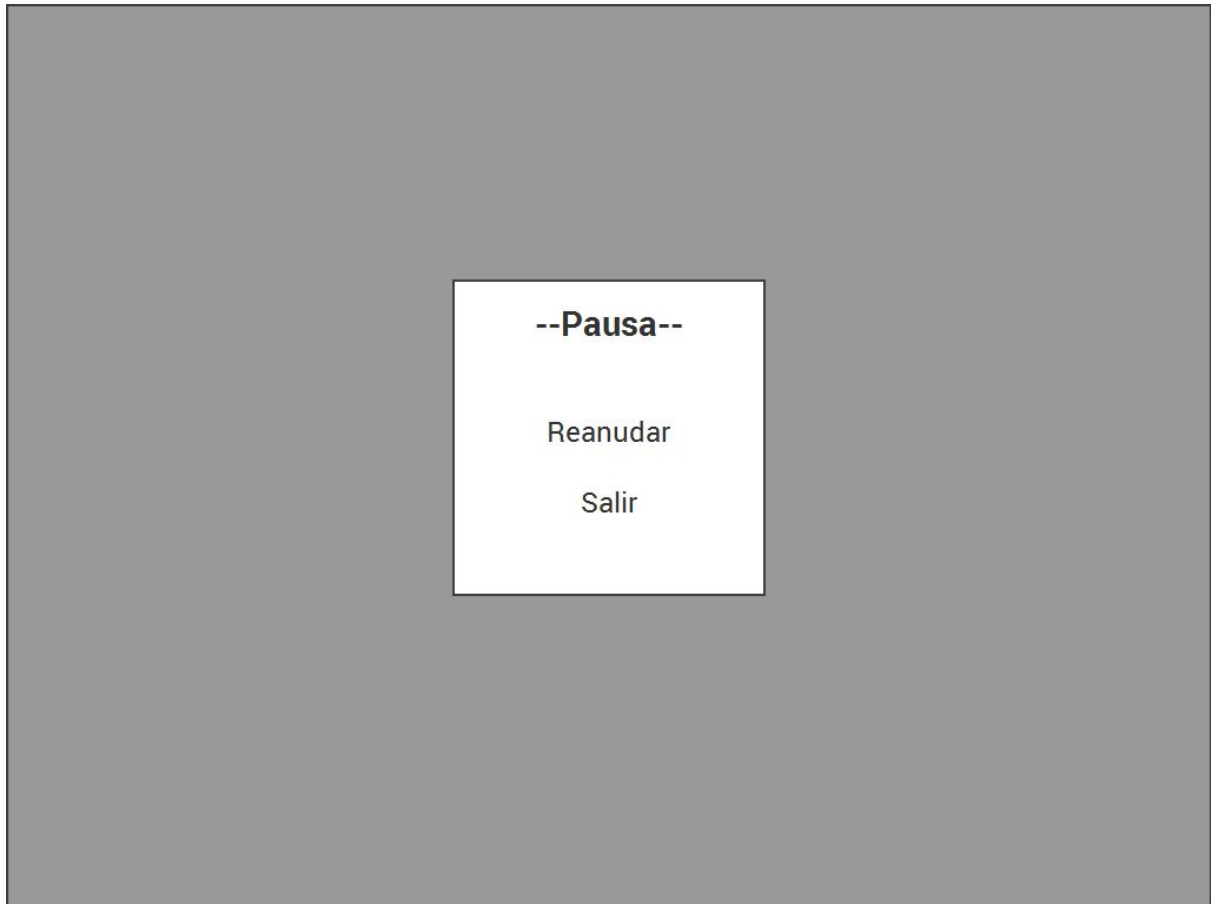


Figura 5.23: Boceto de menú de pausa

### Lista de componentes

- **Botón reanudar:** al pulsarlo se desactiva el menú de pausa y el usuario puede continuar jugando.
- **Botón salir:** al pulsarlo llevará al usuario al menú principal.

## Interfaz de juego

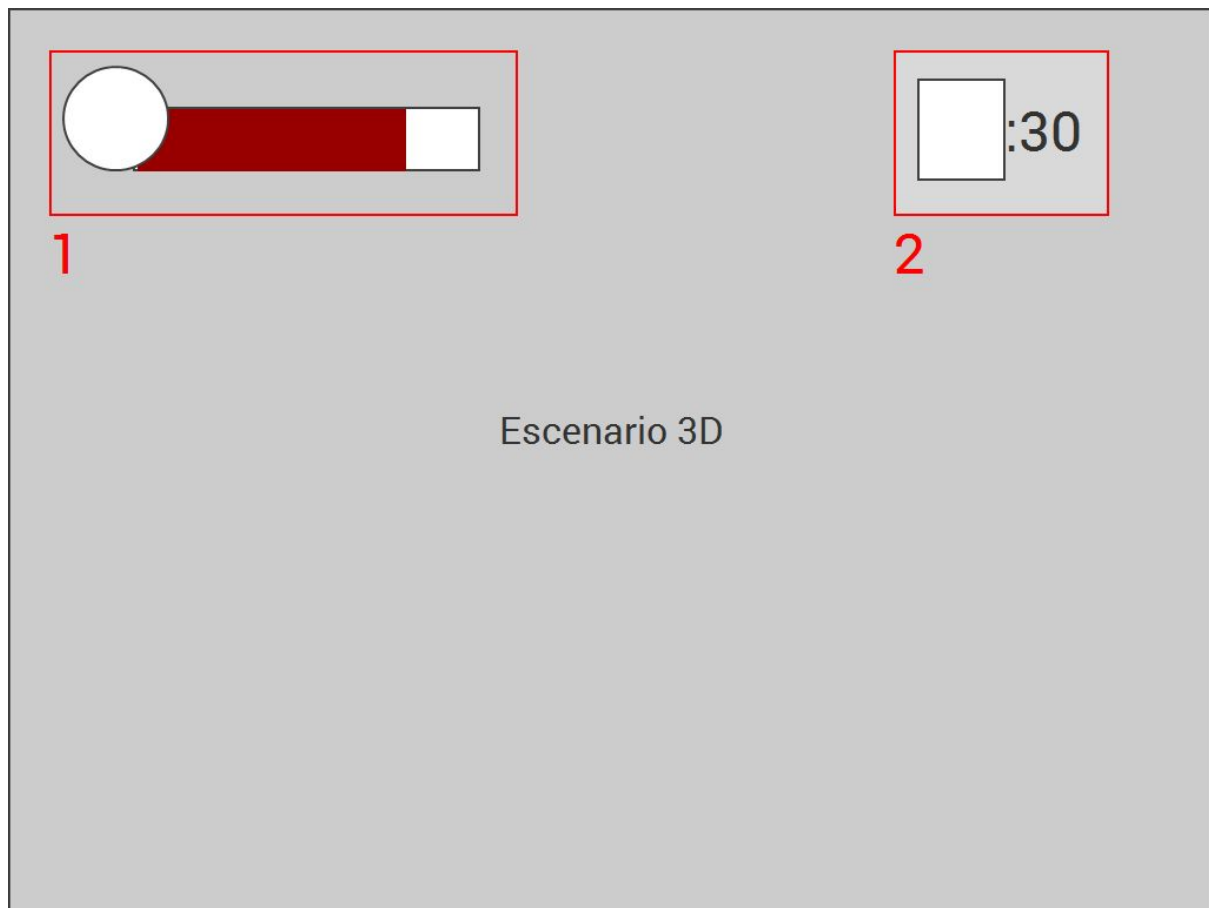


Figura 5.24: Boceto de interfaz de usuario

### Lista de componentes

- **Sección de vida:** mostrará la vida actual del héroe.
- **Sección de cristales:** mostrará la cantidad de cristales recolectadas por el usuario.

### **Pantalla de carga**



Figura 5.25: Boceto de pantalla de carga

### **Lista de componentes**

En esta pantalla no habrá interacción con el usuario. Solo mostrará una animación de carga y será utilizada cuando el videojuego esté cargando un mapa nuevo.



## Pantalla de cinemáticas

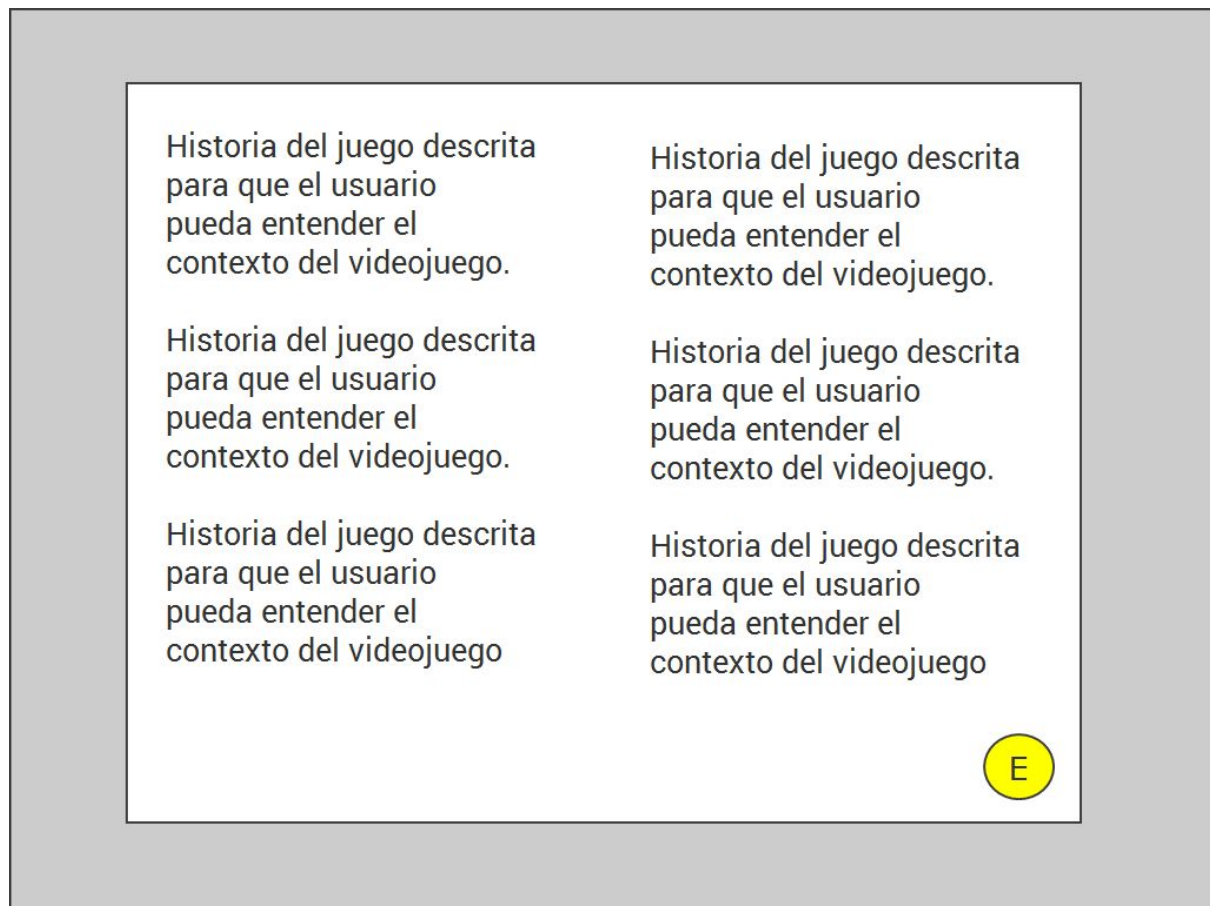


Figura 5.26: Boceto de pantalla de cinemáticas

### Lista de componentes

- **Panel principal:** mostrará la historia de nuestro videojuego.
- **Botón siguiente:** al pulsarlo llevará al usuario a la siguiente cinemática.

## Pantalla de créditos

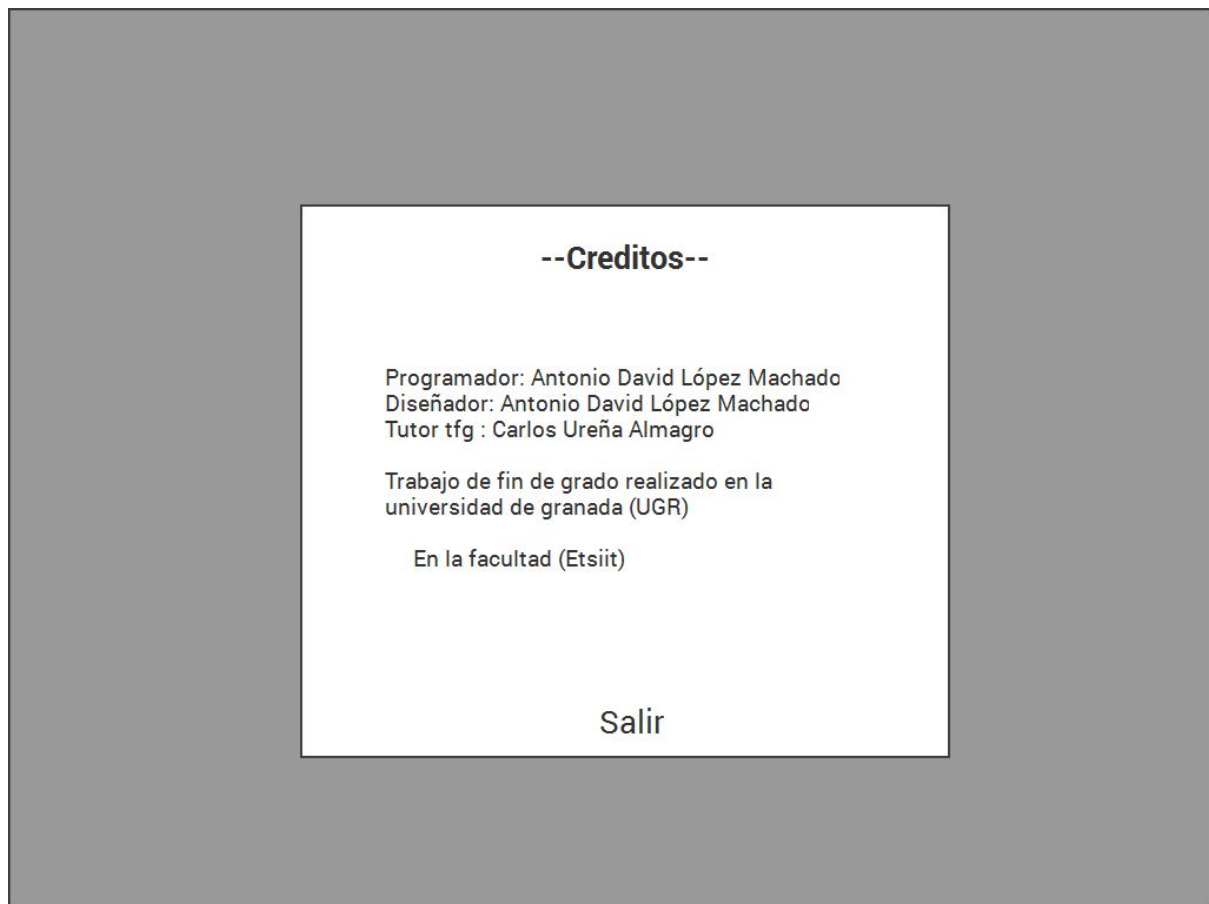


Figura 5.27: Boceto de pantalla de créditos

## Lista de componentes

- **Panel principal:** texto con los componentes del equipo de desarrollo.
- **Botón salir:** al pulsarlo llevará al usuario al menú principal.

### Pantalla de muerte



Figura 5.28: Boceto de pantalla de muerte

### Lista de componentes

- **Botón salir:** al pulsarlo llevará al usuario al menú principal.

## 5.2.8 Algoritmos y estructuras no triviales

En esta sección podremos ver la explicación detallada de cada algoritmo y estructura no trivial de nuestro sistema.

### 5.2.8.1 Grafo de escena y algoritmo de visualización

Se ha utilizado un *grafo de escena*<sup>[12]</sup>[\[UREÑA17\]](#)[\[MOREIRAETALL13\]](#) para el almacenamiento y visualización de nuestros objetos 3D. En este grafo se pueden diferenciar tres tipos de nodo según su contenido. Los tipos de nodo son:

- Un objeto 3D, el cual puede ser tanto una subrama del árbol como una malla o cualquier tipo de elemento que herede de la clase objeto3D.
- Un Material, el cual indicará tanto los componentes del material como la textura con la que se visualizará nuestro objeto.
- Una matriz de transformación, la cual indicará la transformación que se le aplicará a nuestro objeto.

Para realizar la visualización de nuestra escena debemos recoger cada elemento del grafo de escena y activarlo/renderizarlo. Según el elemento actual se pueden realizar diferentes acciones:

- Si el elemento es un objeto 3D lanzaremos su método de visualización. De esta forma, si el objeto 3D es una subrama, comenzaremos a recorrerla y si es una malla la renderizaremos en nuestra escena.
- Si el elemento es un material o una matriz, la introduciremos en la pila correspondiente para que cuando se vaya a visualizar un objeto 3D se active tanto el último material añadido a la pila como el producto de las matrices apiladas.

### 5.2.8.2 Matrices de transformación dinámicas y algoritmo de animación

En nuestro motor podemos diferenciar varios tipos de matrices de transformación según su complejidad y dinamismo, los principales tipos de matrices son:

- **Matriz estática:** esta matriz tiene un tamaño de 4x4 y no se ve afectada por el tiempo. Puede definir tres tipos de transformación, las cuales son:
  - Traslación
  - Rotación en un eje arbitrario
  - Escalado
- **Matriz dinámica:** es un objeto que cuando se fija un valor de tiempo permite obtener una matriz estática que potencialmente puede ser distinta para cada momento de tiempo. Este tipo de matriz nos permite generar movimiento en los objetos. Los tipos de movimientos que podremos realizar son:
  - Movimiento Lineal
  - Rotación sobre un eje y a una velocidad angular predeterminada
  - Rotación oscilante la cual tendrá un rango angular de movimiento y realizará dicha rotación solo dentro de su rango

- **Matriz compuesta:** es un tipo de matriz dinámica que tiene asociado una matriz dinámica para distintos intervalos de tiempo, lo cual a su vez es una composición de matrices dinámicas.

Nuestro algoritmo de animación se centra en nuestra estructura de matrices compuestas. Dicha matriz contendrá múltiples matriz dinámicas, pero sólo podrá tener activa una matriz del conjunto en cada instante de tiempo. La matriz que esté activa determinará la transformación que se producirá sobre el objeto animado en cada instante de tiempo.

La forma de determinar la matriz que está activa actualmente es mediante el tiempo de vida de la matriz dinámica. Cuando la matriz activa consume su tiempo, nuestra matriz compuesta desactiva esa matriz y activa la siguiente matriz del vector y comienza a consumir su tiempo de vida.

El objeto que utilice esta matriz compuesta actualizará su movimiento con la matriz que está actualmente activa en dicha matriz compuesta. De esta forma podemos realizar una secuencia de movimientos en un objeto generando así su animación.

#### 5.2.8.3 Índice espacial

Esta estructura nos permite la detección de colisiones de forma eficiente en tiempo entre un objeto (héroe, enemigo, flecha, ...) y un elemento del escenario ( voxel u *objeto decorativo*<sup>[22]</sup> ).

Si no se realizara esta indexación espacial para detectar una colisión, deberíamos comprobar si el objeto colisiona con cada elemento del mapa, lo cual tendría una complejidad cuadrática. Con este algoritmo para detectar una colisión deberíamos comprobar si el objeto colisiona con los elementos cercanos a él, sigue teniendo una complejidad cuadrática pero la constante oculta es infinitamente menor.

Par realizar dicha indexación espacial contaremos con la clase ObjetoEscena, la cual tendrá la posición de un objeto y su caja englobante. A partir de esta clase podremos construir una estructura la cual almacenará qué objetos hay en cada posición o voxel de la escena. Esta estructura será un array bidimensional de vectores de ObjetoEscena.

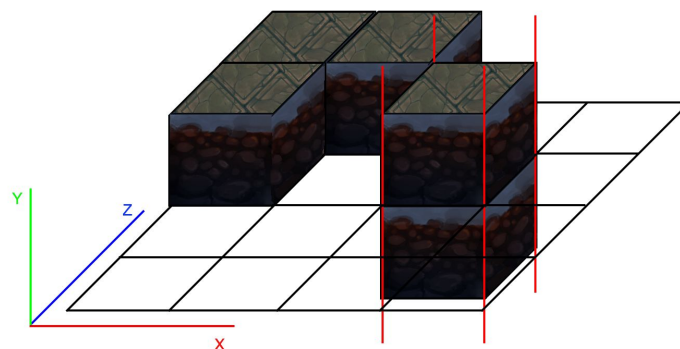


Figura 5.29: Esquema de la indexación espacial

Como podemos ver en la figura 5.29 para acceder a un voxel determinado primero debemos indicar su posición en el eje X y Z, para así obtener el vector de elementos que tiene dicho voxel. De esta forma podemos comprobar si un objeto tiene colisión con alguno de los elementos de ese vector.

En el caso de que un objeto ocupe varias casillas (esté en varios voxeles contiguos en X y Z) ese objeto será introducido en cada uno de los vectores de los diferentes voxeles que ocupe.

#### **5.2.8.4 Conjunto de colecciones**

Este tipo de estructura nos permitirán almacenar y compartir los diferentes tipos de recursos multimedia. De esta forma sólo serán cargados en memoria una vez.

Una colección estará formada por los siguientes elementos:

- Vector de elementos, el cual contendrá el conjunto de recursos multimedia de un mismo tipo.
- Una indexación, la cual nos permite indicar el recurso que queremos utilizar de una forma más sencilla y eficiente.

#### **5.2.8.5 Algoritmo de agrupación de objetos por material**

Dado a que al renderizar los objetos de la escena éstos no están ordenados por su material, se producen un gran número de activaciones y desactivaciones de materiales, lo cual produce una bajada del rendimiento de nuestra aplicación.

Para solucionar dicho problema se ha implementado este algoritmo, el cual agrupa cada voxel de la escena que comparta material en una única malla, la cual es renderizada. Esto nos produce una enorme mejora de la eficiencia, puesto que se reduce el número de elementos a renderizar y se producen menos cambios de material.

Este algoritmo es solo utilizado para los voxeles que crean el mapa, y no es utilizado para los objetos decorativos (con o sin colisión), ya que su número es mucho menor y no es necesario realizar este tipo de algoritmo en ellos.

#### **5.2.8.6 Algoritmo recolector de basura por cuenta de referencia**

Dado que nuestro grafo de escena es un grafo dirigido acíclico, un algoritmo elemental de liberación de memoria del grafo podría llevar a que un nodo que es referenciado más de una vez se intente borrar en múltiples ocasiones.

Cada nodo de nuestro grafo de escena tendrá un contador de referencias. Dicho contador se inicializa a 0 y aumentará su valor en uno cuando dicho nodo sea añadido a un grafo de escena.

Cuando se realice la destrucción del grafo de escena y se referencie a un nodo para su eliminación, se le disminuirá en 1 su contador de referencias. De esta forma si el contador de referencias no es 0, el nodo no es eliminado, puesto que es compartido con otro subgrafo. Cuando el contador de referencias llegue a 0 el nodo será eliminado.

De esta forma no se realizará una eliminación sobre un nodo que ya haya sido eliminado.

#### 5.2.8.7 Inteligencia artificial del enemigo

Nuestra inteligencia artificial para los enemigos del sistema utiliza un algoritmo de obtención de la mejor opción posible.

El algoritmo en sí comprobará en cada una de las 8 direcciones en las que puede orientarse un avatar cuál es la dirección que posiciona al enemigo más cerca de la posición actual del héroe. En el caso de la IA del enemigo a distancia, seleccionara la opción que aleje más al enemigo de la posición actual del héroe.

Además, si la mejor dirección de movimiento posible hace que nuestro enemigo colisione con otro enemigo, no será seleccionada dicha opción sino que se seleccionará la siguiente mejor opción.

#### 5.2.8.8 Algoritmo de mapeado topológico (*bump mapping*)

También denominado mapeo de normales[[VRIES15-3](#)] es un algoritmo que nos permite codificar en una imagen las variaciones a pequeña escala de la orientación de una superficie que, a una escala mayor, es plana, evitando el uso de una gran cantidad de polígonos muy pequeños para realizar estas variaciones de orientación.

Para ello necesitamos el tipo de textura denominada “mapa de normales”. Este tipo de textura contiene información sobre la normal que tendrá cada fragmento de la textura. Este algoritmo nos permite usar esas normales en vez de las normales de vértice o de cara, dándonos así un mayor nivel de realismo.

Por ejemplo, este tipo de algoritmo es muy usado en videojuegos, ya que con dichas normales podemos tener una malla más simple y a través de dicho mapa de normales obtenga la malla el nivel de detalle que requiere.

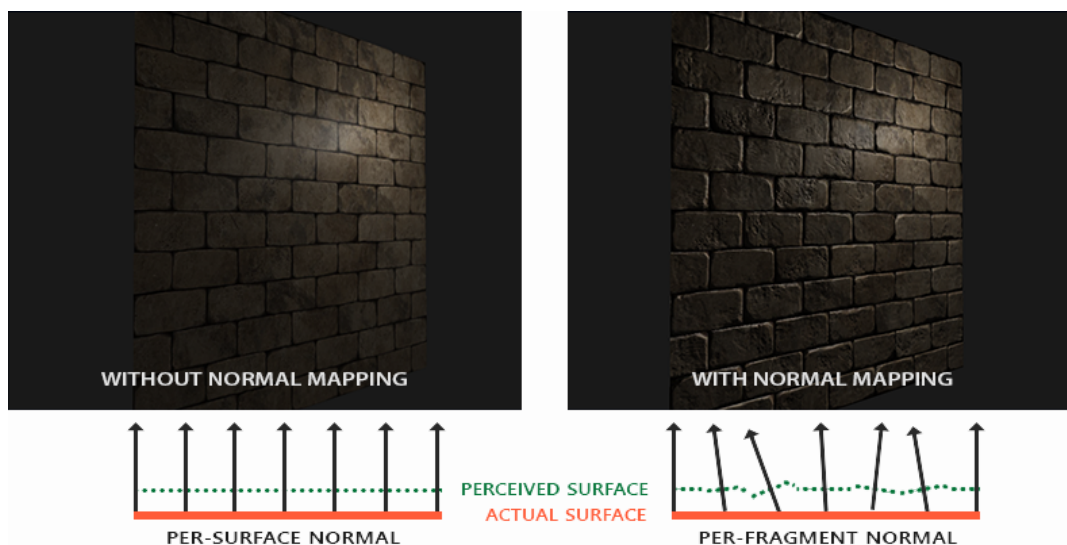


Figura 5.30. Mapa de normales aplicada a una textura de ladrillos[[VRIES15-3](#)].

Como podemos apreciar en la figura 5.30 el mapa de normales nos da una mejora a nivel de detalle considerable.

#### 5.2.8.9 Algoritmo de mapeado de sombras

Este algoritmo nos permite la generación dinámica de sombras arrojadas[VRIES15-1] en nuestra escena. Como bien sabemos, en OpenGL cada objeto se renderiza de forma individual, es decir, no conoce los objetos que ya están en la escena por lo que no podemos conocer si un objeto tiene una sombra arrojada de otro.

La idea principal del algoritmo es la renderización de la escena desde la posición de una luz. Los objetos que podamos ver desde dicha posición estarán iluminados y los que no podamos ver estarán en sombra. Por esto ya podemos intuir que este algoritmo tendrá dos fases bien diferenciadas.

En la primera fase del algoritmo el objetivo es conocer la profundidad que tiene nuestro escenario con respecto a un punto de luz. Para ello, realizamos este renderizado en el que la cámara está posicionada en dicho punto de luz. Dicha información de profundidad la obtenemos a partir del z-buffer y es guardada en una textura la cual será utilizada en la segunda fase.

En la segunda fase del algoritmo se realiza la renderización con la cámara principal de la escena. Para que en esta renderización el usuario pueda ver las sombras es necesario utilizar para este renderizado la textura de profundidad generada en la primera fase. Mediante dicha textura nuestro fragment shader tendrá la información suficiente para poder calcular si hay sombra o no en un determinado fragmento.

Como podemos ver en la figura 5.31 la idea principal es que cuando estamos calculando el color de un fragmento comprobamos también la profundidad del objeto con la profundidad de nuestra textura de profundidad. Si la profundidad de la textura es menor, significa que dicho fragmento está en sombra, en caso contrario estaría iluminado dicho fragmento.

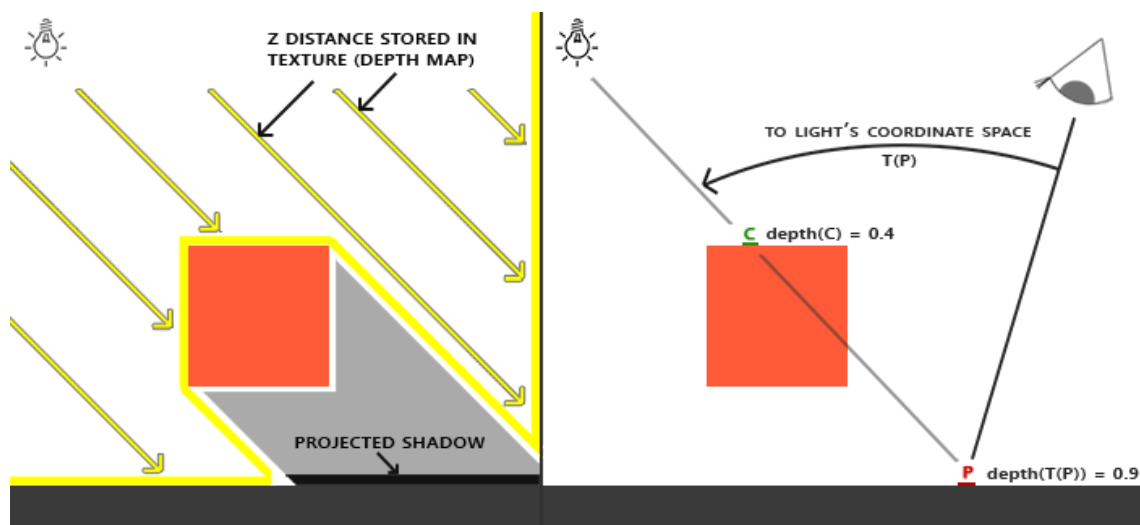


Figura 5.31. Esquema del algoritmo de mapeado de sombras[VRIES15-1].



Además, cuando se calcula la sombra también se realiza un sencillo algoritmo de suavizado de sombras. Dicho algoritmo calcula para cada fragmento la sombra de los 4 fragmentos vecinos y realiza una media de dichas sombras. De esta forma, el contorno de la sombra no es en sierra sino que tiene un mayor suavizado.

## 5.3. Implementación y pruebas

En esta sección podemos ver tanto las tecnologías utilizadas en el proyecto como las diferentes pruebas realizadas sobre los diferentes elementos que componen el proyecto.

### 5.3.1 Implementación

Las principales tecnologías utilizadas han sido:

- **SDL2:** esta librería nos ha permitido tanto la creación y manejo de la ventana principal de nuestro juego como el manejo de eventos de teclado y mando de una forma sencilla. Además, hemos utilizado dos librerías adicionales de la SDL2, una para el procesamiento de texto(SDL\_TTF) y la otra ha sido utilizada para el procesamiento del sonido(SDL\_mixer).
- **OpenGL moderno:** para el apartado de renderización de la escena hemos utilizado OpenGL en sus versiones más modernas(3.0+), ya que nos daba la flexibilidad que era requerida. Además, hemos utilizado shaders para el cálculo tanto del color de cada pixel como de la posición de cada vértice de un objeto.

### 5.3.2 Pruebas

Como ya hemos mencionado con anterioridad se han realizado un conjunto de test sobre el contenido de cada iteración durante su fase de desarrollo/pruebas.

No se han realizado pruebas unitarias sobre el código desarrollado debido al alto nivel interacción de cada componente con el sistema. El tipo de pruebas realizado han sido pruebas de funcionalidad sobre cada componente desarrollado.

A continuación podemos ver todas las pruebas realizadas en cada iteración y el resultado que se obtuvo.

| Iteración 1 |                                |   |           |
|-------------|--------------------------------|---|-----------|
| Fecha       | Elementos a inspeccionar       | Naturaleza de la prueba                             | Resultado |
| 10/10/2016  | Window : <i>createWindow()</i> | Comprobar que la ventana se crea correctamente      | éxito     |
| 10/10/2016  | Window : <i>resizeWindow()</i> | Comprobar que la ventana puede modificar su tamaño. | éxito     |

|            |  |   |         |
|------------|--|---|---------|
| 10/10/2016 | <i>Window : fullscreen()</i>                         | Comprobar que la ventana puede pasar de modo pantalla completa a modo ventana y viceversa.  | éxito   |
| 10/10/2016 | <i>Window : window()</i>                             | Comprobar que el nombre de la ventana es modificable.   | éxito   |
| 10/10/2016 | <i>Window</i>  | Comprobar que nuestra ventana tiene la funcionalidad estándar (minimizar, maximizar, cerrar) de la barra de título                                    | éxito   |
|            |  |   |         |
| 11/10/2016 | <i>Shader : compile()</i>                            | Comprobar que vertexShader y fragmentShader pueden ser compilados correctamente(no producir errores)  | éxito   |
| 11/10/2016 | <i>Shader : linkShaders()</i>                        | Comprobar que podemos enlazar a un programa un vertexshader y un fragmetnShader ya compilados.  | éxito   |
| 11/10/2016 | <i>Shader : getProgram()</i>                         | Comprobar que la activación de un programa ya enlazado no produce ningún error en el sistema.   | éxito   |
| 12/10/2016 | <i>Sound : loadSound()</i>                           | Comprobar la correcta captación de los ficheros de audio tanto de música como de efectos  | éxito   |
| 12/10/2016 | <i>Sound : play(),stop(),pause()</i>                 | Comprobar que el audio se puede reproducir/pausar   | éxito   |
| 12/10/2016 | <i>Sound: changeVolume()</i>                         | Comprobar que el audio puede cambiar su volumen.  | éxito   |
|            |  |   |         |
| 13/10/2016 | <i>Game : loop()</i>                                 | Comprobar que el funcionamiento del bucle principal de la aplicación no produce error.  | éxito   |
| 13/10/2016 | <i>FileObj</i>                                       | Comprobar que la información parseada coincide con la del modelo 3d.  | éxito   |
| 14/10/2016 | <i>FileObj: calculate_Normals</i>                    | Comprobar que se realiza de forma correcta el cálculo de las normales de vértice de los objetos.  | éxito   |
| 14/10/2016 | <i>FileObj : calculate_origin</i>                    | Comprobar que se realiza de forma correcta el cálculo del centro de masas de cada objeto.   | éxito   |
|            |  |   |         |
| 15/10/2016 | <i>Mesh : visualization()</i>                        | Comprobar que la malla visualizada en nuestro sistema coincide con el modelo 3D leído.  | fracaso |
| 16/10/2016 | <i>Mesh : visualization()</i>                        | Comprobar que la malla visualizada en nuestro sistema coincide con el modelo 3D leído.  | éxito   |
| 15/10/2016 | <i>NodeSceneGraph : visualization()</i>              | Comprobar el correcto recorrido del árbol de nodos  | éxito   |
|            |  |   |         |
| 17/10/2016 | <i>Matrix4f : rotate,scale,translation</i>           | Comprobar que el funcionamiento de las principales transformaciones(rotation,escalado, traslación) producen la transformación calculada teóricamente. | éxito   |
| 17/10/2016 | <i>OscillateRotation,LinearMovement,AxisRotation</i> | Comprobar que la actualización de nuestra matriz dependiente del tiempo coinciden con los datos calculados teóricamente.                              | éxito   |
| 17/10/2016 | <i>MatrixStack</i>                                   | Comprobar que nuestra estructura fifo apila/desapila correctamente los elementos  | fracaso |
| 19/10/2016 | <i>MatrixStack</i>                                   | Comprobar que nuestra estructura fifo apila/desapila correctamente los elementos  | éxito   |
| 20/10/2016 | <i>MatrixStack,NodeSceneGraph</i>                    | Comprobar que el árbol de nodos aplica correctamente las transformaciones a cada objeto   | éxito   |

|            |   |   |         |
|------------|---|---|---------|
|            |   |   |         |
| 21/10/2016 | Camera : projection<br>Orthographic/Perspective | Comprobar que la creación de una proyección coincide con lo calculado teóricamente independientemente del tipo de proyección que sea. | fracaso |
| 22/10/2016 | Camera : projection<br>Orthographic/Perspective | Comprobar que la creación de una proyección coincide con lo calculado teóricamente independientemente del tipo de proyección que sea. | éxito   |
| 22/10/2016 | Camera : createCamera()                         | Comprobar que la posición introducida y la posición real de la cámara coinciden   | éxito   |
| 22/10/2016 | Camera : ActivateCamera()                       | Comprobar que la cámara puede ser activada en un shader.  | éxito   |
| 22/10/2016 | Camera : activate Orthographic/<br>Perspective  | Comprobar que ambos tipos de proyección pueden ser activados en un shader.  | éxito   |
|            |   |   |         |
| 26/10/2016 | ScriptLMD : update()                            | Comprobar que cada elemento está activo mientras no agote su tiempo   | éxito   |
| 26/10/2016 | ScriptLMD : update()                            | Comprobar que cuando un elemento agota su tiempo el nuevo elemento activo es el siguiente en el vector.                               | éxito   |

Tabla 5.9: Pruebas sobre los componentes de la primera iteración.

| Iteración 2 |                              |  |           |
|-------------|------------------------------|--|-----------|
| Fecha       | Elementos a inspeccionar     | Naturaleza de la prueba  | Resultado |
| 07/11/16    | Texture,fileObj              | Comprobar que la textura visualizadas coincide con las del modelo importado tanto en aspecto como en asignación de la textura a la malla(coordenadas de textura) | éxito     |
| 07/11/16    | Material,Mesh                | Comprobar que podemos visualizar el material y textura en un objeto y que esta coincide con los datos del modelo 3d  | éxito     |
| 07/11/16    | MaterialStack,NodeSceneGraph | Comprobar que la asignación de material a un objeto coincide con lo especificado en nuestro árbol de nodos   | éxito     |
|             |                              |  |           |
| 07/11/16    | Light                        | Comprobar que la luz incide en los objetos de nuestra escena y que sus parámetros lumínicos coinciden con lo especificado en la creación de la luz.              | éxito     |
|             |                              |  |           |
| 08/11/16    | Hero : visualization         | Comprobar la correcta visualización de todos los modelos que componen al héroe   | éxito     |
| 10/11/16    | Hero: updateState            | Comprobar que el héroe reacciona a los eventos de teclado correctamente(realiza la acción correcta)  | fracaso   |
| 11/11/16    | Hero: updateState            | Comprobar que el héroe reacciona a los eventos de teclado correctamente(realiza la acción correcta)  | éxito     |
| 12/11/16    | AvatarMove : moveBody()      | Comprobar que el héroe cambia su dirección   | éxito     |

|          |                                     |   |         |
|----------|-------------------------------------|---|---------|
|          |                                     | correctamente ante un evento  |         |
|          |                                     |   |         |
| 14/11/16 | RootMap                             | Comprobar que cada objeto de nuestro mapa es visualizado y este está en la posición,dirección y tamaño especificados en su creación | éxito   |
| 14/11/16 | RootMap: detectColisión             | Comprobar que cuando la geometría de un objeto interseca la de otro se detecte como una colisión entre los objetos                  | éxito   |
| 15/11/16 | AvatarMove:moveBody,<br>Hero:update | Comprobar que el héroe colisiona correctamente en cada dirección  | fracaso |
| 16/11/16 | AvatarMove:moveBody,<br>Hero:update | Comprobar que el héroe colisiona correctamente en cada dirección  | éxito   |
| 17/11/16 | AvatarMove : gravity                | Comprobar que la caída realiza correctamente dicho movimiento ascendente(velocidad y posicionamiento)                               | éxito   |
| 17/11/16 | AvatarMove:gravity                  | Detectar colisiones verticales,héroe use movimientos descendentes correctamente   | fracaso |
| 18/11/16 | AvatarMove:gravity                  | Detectar colisiones verticales,héroe use movimientos descendentes correctamente   | éxito   |
|          |                                     |   |         |
| 24/11/16 | AvatarMove: activeJump              | Comprobar la correcta reacción del héroe cuando el usuario use la acción de salto   | éxito   |
| 25/11/16 | AvatarMove : Jump                   | Comprobar que al saltar realiza correctamente dicho movimiento ascendente(velocidad y posicionamiento)                              | éxito   |
| 25/11/16 | AvatarMove : Jump                   | Comprobar que el héroe desactiva la acción de salto en el momento adecuado(cuando la velocidad llegue a 0)                          | éxito   |
| 26/11/16 | AvatarMove : Jump                   | Comprobar la desactivación del salto al colisionar con un objeto que está encima de él  | éxito   |

Tabla 5.10: Pruebas sobre los componentes de la segunda iteración.

| Iteración 3 |                          |  |           |
|-------------|--------------------------|--|-----------|
| Fecha       | Elementos a inspeccionar | Naturaleza de la prueba  | Resultado |
| 08/12/16    | Npc : visualization      | Comprobar la correcta visualización al npc   | éxito     |
| 08/12/16    | Npc                      | Comprobar que la posición de un npc coincide con la posición especificada en sus parámetros de creación. | éxito     |
| 08/12/16    | Npc : updateState        | Comprobar que la animación visualizada de un npc coincide con lo especificado en su animación            | éxito     |
|             |                          |  |           |
| 10/12/16    | Text ,Window             | Comprobar la correcta inicialización del gestor de Textos de SDL2  | éxito     |
| 11/12/16    | Text                     | Comprobar que el texto visualizado coincide con el texto introducido(mensaje,fuente,tamaño)              | fracaso   |
| 12/12/16    | Text                     | Comprobar que el texto visualizado coincide con el texto introducido(mensaje,fuente,tamaño)              | éxito     |
| 11/12/16    | Text : updateState       | Comprobar que el texto se posiciona en la  | éxito     |

|          |   |  |         |
|----------|---|--|---------|
|          |   | posición introducida como parámetro  |         |
|          |   |  |         |
| 12/12/16 | IA_Npc  | Comprobar que cada mensaje introducido en la IA es introducido en la posición correcta   | éxito   |
| 12/12/16 | IA_Npc  | Comprobar que al solicitar el siguiente mensaje se devuelve el mensaje y emisor correcto   | éxito   |
| 13/12/16 | Npc   | Comprobar que el héroe solo interacciona con el npc cuando está en su área de acción   | fracaso |
| 14/12/16 | Npc   | Comprobar que el héroe solo interacciona con el npc cuando está en su área de acción   | éxito   |
| 13/12/16 | Npc   | Comprobar que los mensajes los visualiza el emisor correcto y están en la posición correcta  | fracaso |
| 14/12/16 | Npc   | Comprobar que los mensajes los visualiza el emisor correcto y están en la posición correcta  | éxito   |
| 13/12/16 | Npc   | Comprobar que el héroe visualiza el diálogo de acción cuando este en el área de interacción de un npc  | éxito   |
|          |   |  |         |
| 16/12/16 | Enemy : visualization                             | Comprobar la correcta visualización de todos los modelos que componen al enemigo   | éxito   |
| 17/12/16 | IA_Enemy : nextPosition                           | Comprobar que la IA selecciona la opción más óptima para acercarse al héroe y que no produzca una colisión entre enemigos.   | fracaso |
| 18/12/16 | IA_Enemy : nextPosition                           | Comprobar que la IA selecciona la opción más óptima para acercarse al héroe y que no produzca una colisión entre enemigos.   | éxito   |
| 17/12/16 | Enemy : updateState                               | Comprobar que el enemigo realiza el movimiento que su IA le ordena.  | éxito   |
| 17/12/16 | Enemy : updateState                               | Comprobar que el enemigo realiza un salto cuando este no puede acercarse al héroe.   | éxito   |
|          |   |  |         |
| 18/12/16 | MeshCollecion,Soundcollection, MaterialCollection | Comprobar que la indexación es correcta y la devolución de recursos multimedia se hacen en el orden adecuado.  | éxito   |
|          |   |  |         |
| 19/12/16 | PauseMenu,MainMenu :updateState                   | Comprobar que el usuario puede interaccionar con los menús correctamente(Navegación por las opciones y el funcionamiento de cada opción)                                 | éxito   |
| 19/12/16 | PauseMenu,DeadMenu                                | Comprobar que nuestro escenario no es actualizado cuando el menú está activado   | éxito   |
| 19/12/16 | MainMenu  | Comprobar que la activación de dicho menú se realiza cuando la aplicación ha sido iniciada o cuando otro menú realiza su activación(el usuario vuelve al menú principal) | éxito   |
| 19/12/16 | PauseMenu   | Comprobar que la activación de dicho menú se realiza cuando el usuario pausa el juego y no está activado otro menú.  | éxito   |
| 19/12/16 | DeadMenu  | Comprobar que la activación de dicho menú se realiza cuando la vida del héroe sea igual o inferior a 0.  | éxito   |
|          |   |  |         |
| 24/12/16 | Item  | Comprobar la correcta visualización y animación de cada tipo de objeto(moneda,poción)  | éxito   |

|          |               |  |         |
|----------|---------------|--|---------|
| 24/12/16 | Item          | Comprobar que si el héroe está cerca de un moneda esta será obtenida por el usuario y aumentará su total de monedas  | fracaso |
| 25/12/16 | Item          | Comprobar que si el héroe está cerca de un moneda esta será obtenida por el usuario y aumentará su total de monedas  | éxito   |
| 26/12/16 | Item          | Comprobar que si el héroe está cerca de una poción esta será obtenida por el usuario y aumentará su salud. Si el héroe tiene la vida máxima no obtendrá dicha poción | éxito   |
| 26/12/16 | ItemList,Item | Comprobar que al coger el héroe una moneda/poción esta es correctamente eliminada de la estructura   | éxito   |

Tabla 5.11: Pruebas sobre los componentes de la tercera iteración.

| Iteración 4 |  |  |           |
|-------------|--|--|-----------|
| Fecha       | Elementos a inspeccionar                               | Naturaleza de la prueba  | Resultado |
| 10/01/17    | Hero,Enemy animation                                   | Comprobar que el enemigo y el héroe activan la animación correcta cuando estén realizando la acción correspondiente.                           | éxito     |
| 10/01/17    | Hero,Enemy animation                                   | Comprobar que el enemigo y el héroe desactivan la animación correcta cuando dejen de hacer la acción correspondiente.                          | éxito     |
| 10/01/17    | Hero,Enemy :<br>initAnimation(),UpdateState()          | Comprobar que la velocidad de cada animación coincide con la velocidad estipulada en al crear cada animación                                   | éxito     |
| 12/01/17    | ScriptLmd resetState(),<br>hero, Enemy : updateState() | Comprobar que al desactivar una animación esta vuelve a su estado original para su siguiente activación.                                       | éxito     |
| 14/01/17    | Hero:updateState();<br>Enemy : takeDamage();           | Comprobar que el héroe puede golpear a un enemigo y dañarlo cuando este cerca de él y nuestro héroe está orientado hacia dicho enemigo.        | éxito     |
| 14/01/17    | AvatarMove: impactMove()                               | Comprobar que cuando un avatar sea golpeado este realice un movimiento de impacto en la orientación opuesta a la del golpe                     | fracaso   |
| 15/01/17    | AvatarMove: impactMove()                               | Comprobar que cuando un avatar sea golpeado este realice un movimiento de impacto en la orientación opuesta a la del golpe                     | éxito     |
| 14/01/17    | Heroe : takeDamage()                                   | Comprobar que el héroe puede ser dañado por un enemigo cuerpo a cuerpo si este está cerca de él y está orientado hacia dicho héroe.            | éxito     |
| 16/01/17    | Heroe : takeShield()                                   | Comprobar que el héroe puede bloquear un golpe de un enemigo cuerpo a cuerpo si este está cerca de él y está orientado hacia dicho héroe.      | éxito     |
| 18/01/17    | Particle system: updateState()                         | Comprobar que el número de partículas activas no superar al límite de partículas posibles.   | éxito     |
| 18/01/17    | Particle system: updateState()                         | Comprobar que todas las partículas respetan los límites de cada parámetro y que estos límites coinciden con lo especificado en sus parámetros. | éxito     |

|          |   |  |         |
|----------|---|--|---------|
|          |   |  |         |
| 19/01/17 | Projectile : visualization()                              | Comprobar que el modelo y textura de los proyectiles son visualizados correctamente  | éxito   |
| 19/01/17 | Projectile : initAnimation()<br>, updateState()           | Comprobar que la animación del proyectil funciona correctamente(se inicia y realiza su movimiento correctamente)                                   | fracaso |
| 20/01/17 | Projectile : initAnimation()<br>, updateState()           | Comprobar que la animación del proyectil funciona correctamente(se inicia y realiza su movimiento correctamente)                                   | éxito   |
| 19/01/17 | Projectile : updateState()                                | Comprobar que el movimiento y dirección del proyectil coincide con el especificado en sus parámetros.  | éxito   |
| 19/01/17 | Projectile : updateState()                                | Comprobar que el proyectil es desactivado cuando colisione con un avatar o con el escenario  | éxito   |
| 19/01/17 | Projectile : updateState()                                | Comprobar que el proyectil realiza daño a un avatar al colisionar con este y que dicho daño coincide con el especificado en sus parámetros.        | éxito   |
| 19/01/17 | Projectile : updateState()                                | Comprobar que el héroe puede bloquear un proyectil. Dicho proyectil no le ocasionará daños y sera desactivado debido a la colisión.                | fracaso |
| 20/01/17 | Projectile : updateState()                                | Comprobar que el héroe puede bloquear un proyectil. Dicho proyectil no le ocasionará daños y sera desactivado debido a la colisión.                | éxito   |
| 21/01/17 | ProjectileSystem: updatestate()                           | Comprobar que el sis.proyectiles es activado cuando el héroe esté en su área de activación.  | éxito   |
| 21/01/17 | ProjectileSystem: updatestate()                           | Comprobar que el sis.proyectiles es desactivado cuando el héroe no esté en su área de activación.  | éxito   |
| 22/01/17 | ProjectileSystem: updatestate()                           | Comprobar que el sistema de proyectiles puede disparar proyectiles y que estos son creados con los parámetros especificados                        | éxito   |
| 22/01/17 | ProjectileSystem:<br>ProjectileSystem(),<br>updatestate() | Comprobar que el tiempo entre proyectil y proyectil de nuestro sistema coincide con el especificado en sus parámetros.                             | fracaso |
| 23/01/17 | ProjectileSystem:<br>ProjectileSystem(),<br>updatestate() | Comprobar que el tiempo entre proyectil y proyectil de nuestro sistema coincide con el especificado en sus parámetros.                             | éxito   |
|          |   |  |         |
| 25/01/17 | RangedEnemy: visualization()                              | Comprobar la correcta visualización de todos los modelos que componen al enemigo a distancia   | éxito   |
| 25/01/17 | IARangedEnemy:<br>nextPosition()                          | Comprobar que la IA del enemigo a distancia selecciona la opción más óptima para alejarse del héroe y que no produzca una colisión entre enemigos. | fracaso |
| 26/01/17 | IARangedEnemy:<br>nextPosition()                          | Comprobar que la IA del enemigo a distancia selecciona la opción más óptima para alejarse del héroe y que no produzca una colisión entre enemigos. | éxito   |
| 25/01/17 | RangedEnemy: updateState()                                | Comprobar que el enemigo a distancia realiza el movimiento que su IA le ordena.  | éxito   |
| 25/01/17 | IAMeleeEnemy, laRangedEnemy<br>: nextPosition()           | Comprobar que ambas IAs detectan correctamente a los demás enemigos y no ordenan un movimiento que provoque una colisión entre ellos.              | éxito   |



|          |  |  |       |
|----------|--|--|-------|
|          |  |  |       |
| 25/01/17 | Weapon : visualization()   | Comprobar la correcta visualización del modelo que componen un arma  | éxito |
| 25/01/17 | Weapon : getDamage();<br>Hero,Enemy : takeDamage()<br>Projectile : updateState() | Comprobar que cuando un avatar realiza un ataque sobre otro y este último es golpeado la reducción de vida coincide con el arma del avatar que realizó dicho ataque. | éxito |
|          |  |  |       |
| 27/01/17 | ControllerManager  | Comprobar que los eventos de teclado y mando pueden ser efectuados simultáneamente.  | éxito |
| 27/01/17 | GamePadController constructor  | Comprobar que nuestra aplicación detecta correctamente nuestro mando y los joystick que contiene.  | éxito |
| 27/01/17 | GamePadController :<br>checkEvent()  | Comprobar que al pulsar botones de nuestro mando las acciones que se produzcan coincidan con las acciones producidas por nuestro teclado.                            | éxito |
| 27/01/17 | GamePadController :<br>addGamePad(),removeGamePad()<br>d()                       | Comprobar que la desconexión/conexión de nuestro mando no produce ningún problema ni en la aplicación ni en su funcionamiento si este está conectado.                | éxito |

Tabla 5.12: Pruebas sobre los componentes de la cuarta iteración.

| Iteración 5 |  |  |           |
|-------------|--|--|-----------|
| Fecha       | Elementos a inspeccionar                   | Naturaleza de la prueba  | Resultado |
| 07/02/17    | Mate : visualization()                     | Comprobar la correcta visualización y posicionamiento de los modelos que componen un compañero   | éxito     |
| 07/02/17    | Mate : updateState()                       | Comprobar que nuestro compañero sigue a nuestro héroe en todo momento sin que detenga su movimiento mientras esté lejano a él.                                     | fracaso   |
| 08/02/17    | Mate : updateState()                       | Comprobar que nuestro compañero sigue a nuestro héroe en todo momento sin que detenga su movimiento mientras esté lejano a él.                                     | éxito     |
| 07/02/17    | Mate : updateState()                       | Comprobar que nuestro compañero activa su animación cuando esté realizando un movimiento y la desactiva cuando no lo realiza.                                      | éxito     |
|             |  |  |           |
| 08/02/17    | TextRegion,EndMapRegión :<br>updateState() | Comprobar que nuestra región es activada cuando nuestro héroe esté en su interior.   | éxito     |
| 09/02/17    | TextRegion : updateState()                 | Comprobar que el emisor de cada mensaje coincide con el especificado en sus parámetros.  | éxito     |
| 09/02/17    | TextRegion : updateState()                 | Comprobar que el diálogo visualizado por el compañero sigue los movimientos de dicho compañero y que este es visualizado el tiempo especificado en sus parámetros. | éxito     |
| 09/02/17    | TextRegion : updateState()                 | Comprobar que el mensaje visualizado es el que corresponde según los parámetros iniciales de creación.   | éxito     |
|             |  |  |           |
| 10/02/17    | Profile                                    | Comprobar que realiza los cálculos que realiza   | éxito     |



|          |                                 |   |         |
|----------|---------------------------------|---|---------|
|          |                                 | dicha clase coinciden con los calculos teoricos.  |         |
| 10/02/17 | Profile : showResult            | Comprobar que los resultados mostrados son visualizados de forma correcta.  | éxito   |
|          |                                 |   |         |
| 11/02/17 | ObjectGroup : add               | Comprobar que la malla es añadida correctamente en la agrupación  | éxito   |
| 11/02/17 | ObjectGroup: init               | Comprobar que la agrupación es cargada correctamente en nuestra gpu   | éxito   |
|          |                                 |   |         |
| 12/02/17 | Notification : visualization()  | Comprobar que la notificación es activada y visualizada correctamente   | éxito   |
| 12/02/17 | Notification : updateState()    | Comprobar que el tiempo durante el que está activada/visible la notificación coincide con el especificado en sus parámetros.  | fracaso |
| 13/02/17 | Notification : updateState()    | Comprobar que el tiempo durante el que está activada/visible la notificación coincide con el especificado en sus parámetros.  | éxito   |
| 12/02/17 | Notification : updateState()    | Comprobar que la notificación cambia su posición con relación a los movimientos realizados por el héroe sin que dicha notificación cambie su posición relativa a la cámara. | éxito   |
|          |                                 |   |         |
| 15/02/17 | SpikeTrap : visualization()     | Comprobar la correcta visualización y posicionamiento( según sus parámetros) de una trampa.   | éxito   |
| 15/02/17 | SpikeTrap : updateState()       | Comprobar que a trampa es activada cuando el héroe está sobre ella. y es desactivada cuando se aleje.   | éxito   |
| 15/02/17 | SpikeTrap : updateState()       | Comprobar que a trampa realiza correctamente(velocidad y duración)tanto la animación de activación como la animación de desactivación.                                      | fracaso |
| 16/02/17 | SpikeTrap : updateState()       | Comprobar que a trampa realiza correctamente(velocidad y duración)tanto la animación de activación como la animación de desactivación.                                      | éxito   |
| 15/02/17 | SpikeTrap : updateState()       | Comprobar que a trampa realiza daño al héroe cuando la trampa está activada y su animación de activación ha finalizado.   | éxito   |
|          |                                 |   |         |
| 15/02/17 | LoaderThread : run()            | Comprobar que podemos crear correctamente nuestro mapa desde una hebra sin que se produzca ninguna error de compilación.  | fracaso |
| 17/02/17 | LoaderThread : run()            | Comprobar que podemos crear correctamente nuestro mapa desde una hebra sin que se produzca ninguna error de compilación.  | éxito   |
| 17/02/17 | LoadingScreen : visualization() | Comprobar que la visualización y posicionamiento de nuestro pantalla de carga coincide con lo estipulado en su constructor.   | éxito   |
| 17/02/17 | LoadingScreen : updateState()   | Comprobar que la textura de carga es actualizada en el tiempo estipulado en la construcción del objeto.   | éxito   |
| 17/02/17 | LoadingScreen : updateState()   | Comprobar que cuando nuestro mapa ha terminado de cargar la pantalla de carga es desactivada.   | éxito   |

|          |   |   |         |
|----------|---|---|---------|
|          |   |   |         |
| 19/02/17 | <i>SaveManager : save()</i>                     | Comprobar que el sistema puede plasmar el progreso del usuario en un fichero json.  | éxito   |
| 19/02/17 | <i>MainMenu: updateState()</i>                  | Comprobar que el sistema obtiene la información del usuario cuando éste selecciona la opción "Continue"   | éxito   |
| 19/02/17 | <i>SaveManager</i>                              | Comprobar que el sistema puede obtener el progreso de un usuario a través de un fichero json.   | éxito   |
|          |   |   |         |
| 22/02/17 | <i>Soul,SoulCarrier, Door : visualization()</i> | Comprobar la correcta visualización y posicionamiento de los objetos de las clases → alma, contenedor de alma, puerta   | éxito   |
| 23/02/17 | <i>Soul : updateState()</i>                     | Comprobar que al realizarse el evento de acción cerca de un alma nuestro héroe porta dicha alma en sus brazos   | éxito   |
| 23/02/17 | <i>Soul : updateState()</i>                     | Comprobar que cuando un alma es portada su posición es correcta en las diferentes direcciones que puede tomar el héroe  | éxito   |
| 23/02/17 | <i>Soul : updateState()</i>                     | Comprobar que al realizar un evento de acción si el héroe está portando un alma está deja se ser portada y se queda en la posición actual del héroe                   | éxito   |
| 23/02/17 | <i>Soul : updateState()</i>                     | Comprobar que si el héroe ya está portando un alma no puede portar otra.  | fracaso |
| 24/02/17 | <i>Soul : updateState()</i>                     | Comprobar que si el héroe ya está portando un alma no puede portar otra.  | éxito   |
| 25/02/17 | <i>SoulCarrier : updateState()</i>              | Comprobar que cuando el héroe deja de portar un alma cerca de un contenedor de alma dicha alma es posicionada en el contenedor de alma y este pasa a estar activado.  | éxito   |
| 25/02/17 | <i>Hero,SoulCarrier : updateState()</i>         | Comprobar que el héroe no puede portar un alma la cual ya está dentro de un contenedor de alma.   | éxito   |
| 26/02/17 | <i>Door : updateState()</i>                     | Comprobar que al activarse un contenedor de alma la puerta enlazada con dicho contenedor también es activada.   | éxito   |
| 26/02/17 | <i>Door : updateState()</i>                     | Comprobar que cuando una puerta es activada esta realiza su animación de apertura(a la velocidad adecuada) y la colisión de este objeto es eliminada de nuestro mapa. | éxito   |

Tabla 5.13: Pruebas sobre los componentes de la quinta iteración.

| Iteración 6 |  |  |           |
|-------------|--|--|-----------|
| Fecha       | Elementos a inspeccionar                   | Naturaleza de la prueba  | Resultado |
| 07/03/17    | <i>ControlMenu,MainMenu: updateState()</i> | Comprobar que al seleccionar la opción "Control" de nuestro menú principal se activa y es visualizado nuestro menú de controles        | éxito     |
| 07/03/17    | <i>ControlMenu.updateState()</i>           | Comprobar que si esta el menú activado al pulsar el botón de acción el menú pasa a estar desactivado y se visualiza el menú principal. | éxito     |
| 09/03/17    | <i>CreditScreen</i>                        | Comprobar que al finalizar el último mapa del videojuego la pantalla de créditos es activa y es  | éxito     |

|          |  |  |         |
|----------|--|--|---------|
|          |  | visualizada  |         |
| 09/03/17 | <i>CreditScreen.updateState()</i>                          | Comprobar que si está activado al pulsar el botón de acción el menú pasa a estar desactivado y se visualiza el menú principal.                                   | éxito   |
| 10/03/17 | <i>MovieScreen</i>   | Comprobar que al cargar un mapa que contenga cinemática la pantalla de cinemáticas es activa y es visualizada, en caso contrario no es activada.                 | éxito   |
| 10/03/17 | <i>MovieScreen.updateState()</i>                           | Comprobar que si está activado al pulsar el botón de acción el menú pasa a estar desactivado y se visualiza el menú principal.                                   | éxito   |
|          |  |  |         |
| 13/03/17 | <i>OptionsMenu</i>   | Comprobar que al seleccionar la opción "Option" de nuestro menú principal se activa y es visualizado nuestro menú de opciones                                    |         |
| 13/03/17 | <i>OptionsMenu : updateState()</i>                         | Comprobar que si esta el menú activado al pulsar el botón de acción en las opciones "Save" o "Quit" pasa a estar desactivado y se visualiza el menú principal.   | éxito   |
| 13/03/17 | <i>OptionsMenu : updateState()</i>                         | Comprobar que podemos cambiar de valor cualquier opción del menú entre los valores que se introdujeron en su creación.   | éxito   |
| 14/03/17 | <i>OptionsMenu : updateState()</i><br><i>OptionManager</i> | Comprobar que al pulsar la opción "Save" el sistema guarda la configuración actual del usuario en un fichero json y se aplicaran los cambios en nuestro sistema. | fracaso |
| 15/03/17 | <i>OptionsMenu : updateState()</i><br><i>OptionManager</i> | Comprobar que al pulsar la opción "Save" el sistema guarda la configuración actual del usuario en un fichero json y se aplicaran los cambios en nuestro sistema. | éxito   |
| 14/03/17 | <i>OptionsMenu : updateState()</i>                         | Comprobar que al pulsar la opción "Quit" el sistema no guarda la configuración seleccionada.   | éxito   |
| 14/03/17 | <i>Game,optionManager</i>                                  | Comprobar que al iniciar el videojuego es cargada la última configuración que realizó el usuario.  | éxito   |
|          |  |  |         |
| 16/03/17 | <i>HeroState</i>   | Comprobar que la posición de la interfaz permite visualizar de forma clara cada elemento.  | éxito   |
| 16/03/17 | <i>HeroState:UpdateState</i>                               | Comprobar que cuando el héroe tenga un aumento/reducción de vida nuestra interfaz es actualizada y muestra la vida actualizada del héroe                         | éxito   |
| 16/03/17 | <i>HeroState:UpdateState</i>                               | Comprobar que cuando el héroe obtenga una moneda nuestra interfaz es actualizada y muestra la cantidad de monedas obtenidas hasta el momento.                    | éxito   |
|          |  |  |         |
| 19/03/17 | <i>Material</i>  | Comprobar que la textura de bump mapping es cargada sin errores.   | éxito   |
| 19/03/17 | <i>Material,Mesh</i>                                       | Comprobar que la rotación en cualquier ángulo y eje de la textura no provocan un mal funcionamiento de las normales  | éxito   |
| 19/03/17 | <i>Material,Mesh</i>                                       | Comprobar que la rotación en cualquier ángulo y eje de la textura no provocan un mal funcionamiento de las normales  | fracaso |
| 20/03/17 | <i>Material,Mesh</i>                                       | Comprobar que la rotación en cualquier ángulo y  | éxito   |

|          |                                   |  |         |
|----------|-----------------------------------|--|---------|
|          |                                   | eje de la textura no provocan un mal funcionamiento de las normales  |         |
|          |                                   |  |         |
| 24/03/17 | <i>ShadowManager</i>              | Comprobar que la textura de profundidad almacena el z-buffer actual en la escena.                            | fracaso |
| 25/03/17 | <i>ShadowManager</i>              | Comprobar que la textura de profundidad almacena el z-buffer actual en la escena.                            | éxito   |
| 25/03/17 | <i>ShadowManager</i>              | Comprobar que la renderización del escenario utiliza la textura de profundidad                               | éxito   |
| 25/03/17 | <i>ShadowManager, game.loop()</i> | Comprobar que visualmente se muestra la sombra arrojada de todos los objetos del escenario                   | fracaso |
| 26/03/17 | <i>ShadowManager, game.loop()</i> | Comprobar que visualmente se muestra la sombra arrojada de todos los objetos del escenario                   | éxito   |
| 26/03/17 | <i>objscene, game.loop()</i>      | Comprobar que no se visualiza la sombra de los objetos a los que se le haya indicado que no produzcan sombra | éxito   |

Tabla 5.14: Pruebas sobre los componentes de la sexta iteración.

## 6. Conclusiones y vías futuras

Tras la finalización del proyecto, se puede concluir que el desarrollo de un motor gráfico permite al programador una gran libertad a la hora de implementar nuevos juegos.

En el ámbito profesional, este proyecto me ha ayudado a conocer en una mayor profundidad el sector de los videojuegos tanto en su desarrollo como en la creación de un buen diseño para el desarrollo de un videojuego.

En el ámbito académico el desarrollo de este proyecto me ha ayudado a afianzar en gran medida mis conocimientos sobre la librería gráfica OpenGL y sobre el uso e implementación de shader.

Por otro lado, los objetivos planificados para este proyecto han sido alcanzados en su totalidad.

Finalmente, cabe destacar que existen vías de desarrollo que sería interesantes de desarrollar. Estas vías son:

- Implementación de un algoritmo de oclusión ambiental el cual nos permitirá dar un aspecto visual más realista a los videojuegos que desarrollemos utilizando nuestro motor gráfico.
- Mejora en el apartado de animaciones. En vez de crear las animaciones desde cero, poder cargarlas y utilizarlas a partir de un modelo animado.
- El desarrollo de una aplicación para la creación y edición visual de escenarios, los cuales serán usados en nuestro videojuego.

- Añadir un sistema de inventario donde el usuario podrá guardar objeto que encuentre en los escenarios.
- Añadir un sistema de equipamiento. A partir de dicho sistema el usuario podrá equipar a nuestro héroe con diferentes armas y armaduras y éstas tendrán un beneficio el cual puede ser un aumento de vida o de daño.
- Continuar con el desarrollo tanto de escenarios como de enemigos y objetos en nuestro videojuego.
- Añadir un modo cooperativo donde un usuario pueda jugar con otro usuario de forma local.

## 7. Recursos multimedia

En esta sección podemos ver la información de cada recurso multimedia utilizado en el proyecto. A continuación podemos ver las diferentes licencias utilizadas.

- CC0 Public Domain : <https://creativecommons.org/publicdomain/zero/1.0/legalcode>
- CC Attribution 3.0 <https://creativecommons.org/licenses/by/3.0/>
- CC Attribution 4.0 <https://creativecommons.org/licenses/by/4.0/>
- CC Attribution- NonCommercial 4.0 : <https://creativecommons.org/licenses/by-nc/4.0/>
- Bitgem license : <https://shop.bitgem3d.com/pages/terms-of-use-license>

### 7.1 Modelos 3D

- Knight , rougher, CC Attribution-NonCommercial 4.0 , <https://sketchfab.com/models/aa451ea41ecd4e439f199aea4955bad0#> , modificado y utilizado en los objetos : knightHead, knightArm, knightBody, knightFoot, knightFootInv, knightHand, knightHandShield
- free zombie commoner, Bitgem, Bitgem license , <https://shop.bitgem3d.com/collections/free-3d-models/products/low-poly-zombie-commoner> , modificado y utilizado en los objetos : enemyBody, enemyFoot, enemyHand, enemyHead, Club
- Great Sword of Frozen Night , ZugZug , CC Attribution 4.0 <https://skfb.ly/GoHQ> , modificado y utilizado en los objetos : sword.obj
- Flaming Forge shield ,pipclank , CC Attribution 4.0 <https://skfb.ly/JUrX> , modificado y utilizado en los objetos: shield.obj
- Crystal , SomjadeChunthavorn ,CC Attribution 4.0 <https://skfb.ly/CssO> , modificado y utilizado en los objetos : crystal.obj
- Skeleton archer, bitgem , Bitgem license , <https://shop.bitgem3d.com/collections/free-3d-models/products/low-poly-skeleton-archer> ) , modificado y utilizado en los objetos : arrow.obj, rangedBody.obj, rangedHead.obj, rangedFoot.obj, rangedHand.obj, crossBow.obj
- Cute micro ghouls gobrot , bitgem , Bitgem license , <https://shop.bitgem3d.com/collections/free-3d-models/products/low-poly-micro-ghouls-gobrot> , utilizado en los objetos : mateHead.obj, mateHand.obj

- Imperial Window-frame big , supaluci ,CC Attribution 4.0 <https://skfb.ly/JSXt> , utilizado en los objetos : window.obj
- Hyrule Castle Courtyard (Ocarina Of Time 3D) , Dillon , CC Attribution 4.0 <https://skfb.ly/YFW7> , modificado y utilizado en los objetos :s: pillar.obj
- Winter Gnome , Brianna Fromont , CC Attribution 4.0 <https://skfb.ly/WOXp> , modificado y utilizado en los objetos : ghostButler.obj
- furniture #1 , ericdraw , CC Attribution 4.0 <https://skfb.ly/lo8o> , modificado y utilizado en los objetos : bed.obj,bedsideTable.obj
- Dungeon , Yuliia Tsukanova , CC Attribution 4.0 <https://skfb.ly/N6T6> , modificado y utilizado en los objetos : torch.obj,barrel.obj
- Chandelier , \_\_ramen\_\_ , CC Attribution 4.0 <https://skfb.ly/XJv8> , modificado y utilizado en los objetos : chandelier.obj,candleHolder.obj
- Low Poly Game Scene , Partho Borthakur , CC Attribution 4.0 <https://skfb.ly/HJEQ> , modificado y utilizado en los objetos : flag.obj
- Column , RvBVakama , CC Attribution 4.0 <https://skfb.ly/UFT6> , modificado y utilizado en los objetos : column.obj
- Potion Bottle , pixomnia , CC Attribution 4.0 <https://skfb.ly/FTCM> , utilizado en los objetos : potion.obj
- Dinner Table , Evgeniya , CC Attribution 4.0 <https://skfb.ly/KENA> , utilizado en los objetos : chair.obj, rub.obj
- Book Case , JmPrsh153 , CC Attribution 4.0 <https://skfb.ly/BqWx> , modificado y utilizado en los objetos : books.obj
- Chest , pepedrago , CC Attribution 4.0 <https://skfb.ly/KDJz> , modificado y utilizado en los objetos : trunk.obj
- Tiki Treasure! , glenatron , CC Attribution 4.0 <https://skfb.ly/VLRo> , utilizado en los objetos : pot.obj, pot2.obj
- Medieval Door , Josh997 , CC Attribution 4.0 <https://skfb.ly/ERMs> , utilizado en los objetos : door.obj
- Spell Glyph , amdragg , CC Attribution 4.0 <https://skfb.ly/NRVr> , modificado y utilizado en los objetos : glyph.obj
- Soul Stealer Bard fan art , lucak , CC Attribution 4.0 <https://skfb.ly/MXVB> , modificado y utilizado en los objetos : soul.obj

## 7.2 Texturas

- Swamp location , bocharova ,CC Attribution-NonCommercial 4.0 <https://skfb.ly/FAZH> modificado y utilizado en la textura : cubeGrass.png
- Low Poly Game Scene , Partho Borthakur ,CC Attribution 4.0 , <https://skfb.ly/HJEQ> utilizado en la textura : flag.png
- Knight , rougher , CC Attribution-NonCommercial 4.0 , <https://skfb.ly/OMpw> utilizado en la textura : heroTexture.jpg
- Great Sword of Frozen Night , ZugZug , CC Attribution 4.0 <https://skfb.ly/GoHQ> , utilizado en la textura : swordTexture.png
- Flaming Forge shield , pipclank , CC Attribution 4.0 <https://skfb.ly/JUrX> , utilizado en la textura : shieldTexture.png



- Dungeon , Yuliia Tsukanova ,CC Attribution 4.0 <https://skfb.ly/N6T6> , utilizado en la textura : torch.png
- Free hand painted texture pack 11 , bitgem is licensed , <https://shop.bitgem3d.com/collections/free-3d-models/products/texture-pack-11> , utilizado en la textura : cubeDungeon.png,cubeDungeonTrap.png
- Hyrule Castle Courtyard (Ocarina Of Time 3D) , Dillon , CC Attribution 4.0 <https://skfb.ly/YFW7> , utilizado en la textura: cubeWall.png,pillar.png, invisibleCubeWall.png,cubeTrap.png
- Crystal , SomjadeChunthavorn , CC Attribution 4.0 <https://skfb.ly/CssO> , used in the obj: TEX\_crystal.png
- Free zombie commoner , bitgem , <https://shop.bitgem3d.com/collections/free-3d-models/products/low-poly-zombie-commoner> , utilizado en la textura: enemyTexture.png
- SKELETON ARCHER , bitgem license , <https://shop.bitgem3d.com/collections/free-3d-models/products/low-poly-skeleton-archer> , utilizado en la textura: archerTexture.png
- Imperial Window-frame big , supaluci , CC Attribution 4.0 <https://skfb.ly/JSXt> , utilizado en la textura : windowTex.png
- Winter Gnome , Brianna Fromont , CC Attribution 4.0 <https://skfb.ly/WOXp> , utilizado en la textura : butlerTexture.png
- Wooden Box 2D , Alucard , CC Attribution 3.0 <http://opengameart.org/content/wooden-box-2d> , utilizado en la textura: cubeBox.png
- Table , louvey , CC Attribution 4.0 <https://skfb.ly/IOH8> -> utilizado en la textura: furnitureText.png
- furniture #1 , ericdraw , CC Attribution 4.0 <https://skfb.ly/lo8o> , utilizado en la textura : bedTexture.png
- Column , RvBVakama , CC Attribution 4.0 <https://skfb.ly/UFtE> , utilizado en la textura : column.png
- Potion Bottle , pixomnia , CC Attribution 4.0 <https://skfb.ly/FTCM> , utilizado en la textura : potionTexture.png
- Dinner Table , Evgeniya , CC Attribution 4.0 <https://skfb.ly/KENA> , utilizado en la textura: chairTexture.png, tableTexture.png,rugTexture.png, loading1.png, loading2.png, loading3.png, mainmMenuBack.png
- Cute micro ghoull gobrot , bitgem , <https://shop.bitgem3d.com/collections/free-3d-models/products/low-poly-micro-ghoull-gobrot> , modificado y utilizado en la textura : mateTexture.png
- Book Case , JmPrsh153 , CC Attribution 4.0 <https://skfb.ly/BqWx> , utilizado en la textura: bookTexture.png
- gamepad , IO-Images , CC0 Public Domain <https://pixabay.com/es/controlador-gamepad-juegos-de-video-1784571/> , utilizado en la textura: gamepadV.png gamepadX.png, controlScreen.png
- Chest , pepedrago , CC Attribution 4.0 <https://skfb.ly/KDJz> , utilizado en la textura: trunkTexture.png
- Tiki Treasure! , glenatron , CC Attribution 4.0 <https://skfb.ly/VLRo> , utilizado en la textura : potTexture.png, mainMenuBack.png

- Medieval Door , Josh997 , CC Attribution 4.0 <https://skfb.ly/ERMs> , utilizado en la textura: doorTexture.png
- Spell Glyph , amdragg , CC Attribution 4.0 <https://skfb.ly/NRVr> , utilizado en la textura: glyphTexture.png
- libro , 3dman\_eu , CC0 public domain <https://pixabay.com/es/examinar-p%C3%A1gina-despl%C3%A1cese-hasta-1019877/> , utilizado en la textura : movieScreen\*.png
- Pergament , Niedec , CC0 public domain <https://pixabay.com/es/documento-medieval-p%C3%A1gina-pergament-481243/> , utilizado en la textura : movieScreen\*.png
- castillo , Merio , CC0 public domain <https://pixabay.com/es/bandera-aislados-torre-castillo-988860/> , utilizado en la textura : mainMenuBack.png
- castillo , 95C , CC0 public domain <https://pixabay.com/es/castillo-silueta-dibujo-2084599/> , utilizado en la textura : mainMenuBack.png
- Soul Stealer Bard fan art , lucak , CC Attribution 4.0 <https://skfb.ly/MXVB> , utilizado en la textura : soulTexture.png

## 7.3 Sonidos

- Pop sound effect , Debsound. , the Attribution Noncommercial. <http://www.freesound.org/people/debsound/sounds/320549/> , utilizado en el sonido : openSound.wav
- Coins 1 , ProjectsU012 , CC Attribution <http://www.freesound.org/people/ProjectsU012/sounds/341695/> , utilizado en el sonido: coin.wav
- Zombie , Under7dude , CC Attribution <http://www.freesound.org/people/Under7dude/sounds/163447/> , utilizado en el sonido: enemyHit.wav
- Getting hit - Eeoh 01 (Mouth Fx) man voice , Flying\_Deer\_Fx , CC Attribution [http://www.freesound.org/people/Flying\\_Deer\\_Fx/sounds/369003/](http://www.freesound.org/people/Flying_Deer_Fx/sounds/369003/) , utilizado en el sonido heroHit.wav
- Shield Hit 1 , Liam Gilchrist , the Creative Commons <http://www.freesound.org/people/CTCollab/sounds/223630/> , utilizado en el sonido: shield.wav
- Bow01.wav , Erdie , Creative Commons 3 (https://creativecommons.org/licenses/by/3.0/) <http://www.freesound.org/people/Erdie/sounds/65733/> , utilizado en el sonido: shoot.wav
- Sword\_02.wav , dermotte , Creative Commons 3.0 <http://www.freesound.org/people/dermotte/sounds/263011/> , utilizado en el sonido : sword.wav
- Footsteps of soldier on ground.wav , Iberian\_Runa , Creative Commons 3.0 [http://www.freesound.org/people/Iberian\\_Runa/sounds/243794/](http://www.freesound.org/people/Iberian_Runa/sounds/243794/) , utilizado en el sonido : walking.wav



- Arrow\_damage.wav , braqoon , Creative Commons 0.0  
<http://www.freesound.org/people/braqoon/sounds/161098/> , utilizado en el sonido : arrowHit.wav
- Jump.wav , acebrian , Creative Commons 0.0  
<http://www.freesound.org/people/acebrian/sounds/380471/> , utilizado en el sonido : jump.wav
- Button 05.wav , JarredGibb , Creative Commons 0.0  
<http://www.freesound.org/people/JarredGibb/sounds/219476/> , utilizado en el sonido: trapActivated.wav
- Sharpening Knife , LaCezio , Creative Commons non commercial license  
<http://www.freesound.org/people/LaCezio/sounds/320344/> , utilizado en el sonido : animationTrap.wav
- Snowfall final.mp3 , ShadyDave , Creative Commons non commercial license  
<http://www.freesound.org/people/ShadyDave/sounds/262259/> , utilizado en el sonido: backSoundLoop.wav

## 8. Bibliografía final

[FIB08] Facultat d'Informàtica de Barcelona (2008). Retro informática: Historia de los videojuegos [En línea] <https://www.fib.upc.edu/retro-informatica/historia/videojocs.html> , [2017, 29 Mayo ]

[LANTINGA14-1] Sam Lantinga (2014). “SDL Wiki : Display and Window Management” [En línea], <https://wiki.libsdl.org/CategoryVideo> , [2016, 28 Octubre]

[LANTINGA14-2] Sam Lantinga (2014). “SDL Wiki : Game Controller and Joystick Mapping”[En línea] , <https://wiki.libsdl.org/CategoryGameController> , [2017, 29 Enero]

[MILO17] Milo Yip(2011), RapjidJSON: Tutorial [En línea] , [http://rapidjson.org/md\\_doc\\_tutorial.html](http://rapidjson.org/md_doc_tutorial.html) , [2017, 28 Febrero]

[MOREIRAETALL13] Artur Moreira, Henrik Vogelius Hansson, Jan Haller(2013), “SFML Game Development, Packt Publishing”, Tema 3: “Rendering the scene”, Páginas 55-60

[SELLERSETALL14-1] Graham Sellers, Richard S.Wright, Jr. , Nicholas Haemel(2014), “OpenGL SuperBible, Pearson Higher education”, Tema 2: “Our first OpenGL program” , Páginas 16-22

[SELLERSETALL14-2] Graham Sellers, Richard S.Wright, Jr. , Nicholas Haemel(2014), “OpenGL SuperBible, Pearson Higher education”, Tema 4: “Math for 3D Graphics” , Páginas 66-82

[SELLERSETALL14-3] Graham Sellers, Richard S.Wright, Jr. , Nicholas Haemel(2014), “OpenGL SuperBible, Pearson Higher education”, Tema 5: “Data” , Páginas 92-107

[SELLERSETALL14-4] Graham Sellers, Richard S.Wright, Jr. , Nicholas Haemel(2014), “OpenGL SuperBible, Pearson Higher education”, Tema 7: “Vertex Processing and Drawing Commands” , Páginas 231-234

[SELLERSETALL14-5] Graham Sellers, Richard S.Wright, Jr. , Nicholas Haemel(2014), “OpenGL SuperBible, Pearson Higher education”, Tema 12: “Rendering Techniques” , Páginas 504-521

[SHEN16] Shen Weiwei(2016), “Bbsmax: Modern OpenGL” [En línea], <https://www.bbsmax.com/A/rV57LgPdPD/> , [2016, 08 Octubre ]

[UREÑA17] Apuntes Informática Gráfica, Tema 2 . Modelado de objetos , Doble Grado Ingeniería informática y matemáticas, Carlos Ureña Almagro.

[VARONAS12] Nico Varonas(2012): Neoteo: Los motores gráficos más importantes [En línea], <http://www.neoteo.com/los-motores-graficos-mas-importantes-de-la-histori/> , [2017, 29 Mayo]

[VRIES15-1] Joey de Vries(2015), “Learn OpenGL: Shadow Mapping” [En línea] , <https://learnopengl.com/#!Advanced-Lighting/Shadows/Shadow-Mapping> , [2017, 28 Marzo]

[VRIES15-2] Joey de Vries(2015), “Learn OpenGL: Materials” [En línea], <https://learnopengl.com/#!Lighting/Materials> , [2016, 20 Noviembre]

[VRIES15-3] Joey de Vries(2015), “Learn OpenGL: Normal Mapping” [En línea], <https://learnopengl.com/#!Advanced-Lighting/Normal-Mapping> , [2017, 28 Marzo]

[VRIES15-4] Joey de Vries(2015), “Learn OpenGL: Blending” [En línea], <https://learnopengl.com/#!Advanced-OpenGL/Blending> , [2017, 3 Enero]

#### Otro material

- Diversas consultas puntuales a los sitios:
  - Stack OverFlow ( <https://es.stackoverflow.com/> )
  - SDL Wiki ( <https://wiki.libsdl.org/FrontPage> )

## Anexo : Autoría y licencia del proyecto

En este anexo se describen tanto la autoría del proyecto como el tipo de licencia.

### Autoría

Cualquier información o demostración del proyecto realizado en este TFG deberá incluir tanto el nombre de la universidad donde se realizó como el nombre del alumno y del profesor que realizó la labor de tutor.

## Licencia del proyecto

Este proyecto se ha realizado usando la licencia *GNU General Public License v3.0*, la cual se puede consultar en el siguiente enlace: <https://www.gnu.org/licenses/gpl-3.0.en.html>.

Esta licencia permitirá a otros realizar videojuegos o proyectos utilizando como base el desarrollo realizado en este trabajo de fin de grado. Sin embargo, deberá indicar la información tanto del autor como del tutor de este trabajo de fin de grado, así como de la universidad donde se realizó.

## Anexo : Glosario

En este anexo se definen los términos del sistema desarrollado para un mejor entendimiento del trabajo realizado.

### Lista de términos

1. **Alma** : objeto que el héroe debe transportar a un contenedor de alma.
2. **Avatar** : entidad no estática que maneja el usuario o que se relaciona con él.
3. **Compañero** : tipo de avatar que acompaña al héroe en todo momento, siguiendo sus movimientos. Además, ayuda al héroe en la aventura mediante pistas.
4. **Cristal** : elemento del mapa/escena el cual es consumido por el héroe al estar cerca de él. Al ser consumido aumentará el número de cristales obtenidos por el héroe.
5. **Contenedor de alma** : objeto que se activa cuando el héroe coloca un alma en él.
6. **Efecto** : tipo de sonido que se reproducirá cuando se efectúe alguna acción por parte de algún avatar.
7. **Enemigo** : también denominado “avatar no jugable agresivo” . Son avatares que deben eliminar al héroe.
8. **Enemigo a distancia** : tipo de enemigo que lanzará flechas hacia el héroe.
9. **Enemigo cuerpo a cuerpo** : tipo de enemigo que ataca a nuestro héroe con armas de corto alcance, como un garrote.
10. **Evento** : Acción del usuario mediante teclado o mando que inicia una operación en el sistema.
11. **Género plataforma** : categoría de videojuegos que se basa en la superación de obstáculos ya sea saltando o con cualquier otro tipo de acción.
12. **Grafo de escena** : estructura de datos que mediante su recorrido y procesamiento nos permite visualizar el escenario actual.
13. **Héroe** : avatar que es controlado por el usuario. Es el protagonista del juego. Con él podemos interaccionar con el escenario.
14. **IA** : acrónimo de “Inteligencia Artificial”. Es la lógica que sigue el sistema para toma de decisiones de los distintos elementos no estáticos o que no maneja el usuario.
15. **Malla** : conjunto de triángulos y vértices que forman un objeto 3D.
16. **Mando** : dispositivo mediante el cual podemos activar los distintos eventos del sistema.
17. **Mapa** : escena con un conjunto de elementos que son visualizados por el usuario.

18. **Menú** : colección de opciones que permiten al usuario activar/desactivar distintos aspectos del sistema.
19. **Música** : tipo de sonido que se reproducirá en bucle y de forma constante mientras el usuario esté en un nivel.
20. **Notificación** : objeto que será visualizado por un determinado periodo de tiempo y que mostrará información al usuario.
21. **Npc** : también denominado “avatar no jugable pasivo”. Son otros personajes que ayudan al héroe en la aventura mediante pistas.
22. **Objeto decorativo** : objeto estático que puede ocupar más de un voxel. Sirve de decoración del escenario.
23. **Partículas** : elemento básico que se utilizará para representar el fuego .
24. **Poción** : elemento del mapa/escena el cual es consumido por el héroe al estar cerca de él. Al ser consumido aumentará la vida actual del héroe.
25. **Proyectil** : flecha que es lanzada por un avatar o un sistema de proyectiles.
26. **Región** : área del escenario la cual producirá una acción cuando el héroe entre en ella.
27. **Región de final de mapa** : región que al activarse finaliza el mapa actual e inicia el próximo mapa.
28. **Región de texto** : región que al activarse inicia un diálogo entre el compañero y el héroe.
29. **Shader** : proceso que se ejecuta en la GPU el cual nos permite calcular el color de cada fragmento de un objeto y calcular la posición de cada vértice del objeto en la escena.
30. **Sistema de partículas** : sistema que gestiona y actualiza un conjunto de partículas en nuestro sistema.
31. **Sistema de proyectiles** : arma fijada a una pared. Esta arma dispara flechas.
32. **Trampa** : arma fijada al suelo que acciona pinchos hacia arriba cuando el héroe está sobre ella.
33. **Voxel** : también denominado “volumetric pixel”. Constituye la unidad básica de nuestro sistema.

## Anexo : Manual de usuario

### Argumento

Erase una vez un mago que vivía en un gran castillo. Su alma siempre había sido pura. Un día, el mago encontró un extraño cristal oscuro y lo cogió, sin percatarse de la energía maligna que emanaba de él. Entonces, el cristal de energía consumió su alma.

El mago, poseído por la maldición, quería extender esa energía maligna por todo el mundo. Para lograr su objetivo, comenzó a acumular la energía maligna en un enorme cristal oscuro.

El mayordomo del mago vio todo lo ocurrido, con horror. Entonces decidió ir a la ciudad para encontrar un valiente guerrero que pudiese vencer a su malvado maestro.

Solo un guerrero debilucho quiso ayudar al mayordomo, quien no tuvo mas opción que aceptar su oferta, puesto que era el único candidato. Por lo menos era un hombre débil con una buena espada.

Cuando nuestros dos hombres llegaron al castillo, el malvado mago los atacó. No obstante, el villano no tuvo mucha suerte, ya que al lanzar una bola de fuego se quemó la cuerda que sujetaba una lámpara que, al instante le cayó encima, matándolo.

Sin embargo, nuestros dos buenos hombres estaban de suerte, ya que el cristal se rompió cuando el mago cayó sobre él, evitando que el mal se extendiese por el mundo.

Y así es como nuestro débil guerrero salvó el mundo.

Nuestros héroes estaban felices pero cansados, por lo que el mayordomo invitó al guerrero a pasar la noche en el castillo.

No obstante, no se percataron de que el cristal se había roto en pequeños fragmentos y, para su mala suerte, el guerrero fue a dormir a una habitación en la que había uno de los fragmentos.

De repente, el cristal se movió y maldijo el alma del guerrero, sumergiendolo en 300 años de sueño, una muy larga siesta.

## Operaciones generales

| Acciones       | Teclado | Mando              |
|----------------|---------|--------------------|
| Movimiento     | W/A/S/D | Joystick izquierdo |
| Acción         | E       | A                  |
| Salto          | K       | B                  |
| Ataque         | L       | R1                 |
| Escudo         | i       | L1                 |
| Cambio de arma | Q       | Y                  |
| Pausa          | Esc     | Start              |
| Vista          | V       | Back               |

## Inicio de partida

Cuando el videojuego termine de cargar cada recurso multimedia, se mostrará el menú principal del juego donde podremos iniciar una partida, cargar nuestra partida y cambiar la configuración del videojuego.

## Configuración

Los aspectos que el usuario podrá cambiar serán 3: resolución del videojuego, tipo de ventana (pantalla completa o modo ventana) y el volumen general del videojuego.

## Guardar y cargar progreso

### Guardar

El progreso del jugador será guardado de forma automática cada vez que supere un nivel del videojuego.

### Cargar

Para continuar el progreso actual se deberá seleccionar la opción de “*Continue*” en el menú principal del videojuego.

## Abandonar una partida

Para finalizar una partida deberemos pulsar el botón de pausa y seleccionar la opción “Quit”. Ten en cuenta que si abandonas la partida de esta forma no se guardará el progreso realizado en el nivel actual.

## Partida terminada

Cuando el héroe haya perdido todos los puntos de vida la partida habrá terminado y el usuario volverá al menú principal.

## Avatares

### Cody

#### Humano, edad desconocida

A pesar de ser un guerrero endeble nunca dudará en luchar por la justicia y defender a los demás del mal que asola el mundo.



### Elgum

#### Espíritu, 303 años

El espíritu de quien una vez fue un gran mago el cual por desgracia fue consumido por el mal

**Zombi y esqueleto****No muerto, edad desconocida**

Cuerpos resucitados por el mal imbuido en los cristales, no dudarán en intentar absorber tu alma

**James****Espíritu, 328 años**

El apuesto mayordomo del que una vez fue un mago magnífico. Su bigote y cejas eran conocidas por todos los habitantes de la ciudad.

## Objetos

Cristal que tiene esencia maligna en su interior. Deberás recolectar para purificar dicho mal.



Poción con hierbas curativas diversas que curaran tu salud.




## Obstáculos

La trampa de pared detectará al usuario y comenzará a lanzar flechas



La trampa de suelo detectara al usuario cuando esté encima de ella y hará parecer pinchos.

## Mecánicas

Deberás portar el alma (  ) hasta un contenedor de almas(  ) el cual será activado al soltar dicha alma en ella. Al ser activada el contenedor de almas la puerta (  ) se abrirá.

## Sistema de batalla

El sistema de batalla será simple e intuitivo. El usuario podrá atacar a los enemigos tanto a distancia o cuerpo a cuerpo y podrá moverse por el escenario durante el combate.

## Consejos útiles

### Combate fuego con fuego

Será mas fácil para ti matar a un enemigo a distancia utilizando la ballesta ya que dicho enemigo es mortal en cuerpo a cuerpo.

### Observa el tiempo de recarga

Para intentar predecir las trampas del juego deberás saber sus tiempos de acción para así poder anticiparte a ellas