TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

# "El Secreto de la Pirámide: Experiencia audiovisual interactiva centrada en mecánicas inteligentes"

**Curso 2018/2019**


**Alumno/a:**

Jaime Rodríguez Martínez

**Director/es:**

José del Sagrado Martínez

# ÍNDICE

# 1. INTRODUCCIÓN

En este apartado se realizará una introducción del proyecto, donde se presentará la situación actual de los videojuegos y la proposición del proyecto, se detallarán los objetivos de este y se explicará la estructura del resto de la memoria.

## 1.1. Presentación del problema

La industria del videojuego ya está tan presente en nuestras vidas que hace años que no necesita presentación alguna. Llegando incluso a generar más beneficios que las industrias del cine y la de la música juntas (Marketing Directo, 2018), los videojuegos se han convertido en un producto cotidiano, acompañando a muchas personas a lo largo de su vida. El objetivo de estos videojuegos dista mucho. Desde juegos diseñados para "matar el tiempo" como podrían ser el renombrado "Candy Crush", hasta simuladores realistas de vuelo usados por las compañías aéreas o el ejército para entrenar a futuros pilotos sin riesgos. Según un estudio de la AEVI (Asociación Española de Videojuegos), "*en 2017 la producción del sector de los videojuegos fue de 3.577 millones de euros, generando 22.828 empleos"* (LLORENTE & CUENCA, 2018).

Los videojuegos son aplicaciones informáticas, por lo que el éxito o fracaso que tengan como producto residirá en un principio en la correcta aplicación de técnicas propias de la ingeniería del software para su definición, gestión, control y desarrollo. Una característica de los videojuegos es la disparidad de sus elementos, como el diseño de escenarios, la implementación de mecánicas, la creación del arte, la inteligencia artificial, el guión narrativo, los efectos de sonido y la música. Este tipo de proyectos exigen un intensivo trabajo de ingeniería con equipos de desarrollo enormes. Por ejemplo, el título "Grand Theft Auto V (GTA V)" requirió la participación de más de 1000 desarrolladores, según explicó Leslie Benzies, director de Rockstar North en una entrevista concedida a Develop.

La mayor peculiaridad de los videojuegos, la que les diferencia de otras formas de arte, es su alta interactividad entre el usuario y la máquina, creando una historia única para cada persona pues consistirá en su propia interacción con los elementos del videojuego. Además, normalmente en un videojuego, arte e ingeniería se relacionan estrechamente para conseguir una experiencia de usuario más completa y una mayor inmersión en el mundo virtual presentado.

Con el fin de abarcar este proyecto, será necesario aplicar los conocimientos adquiridos a lo largo de la carrera, como desarrollo ágil, gestión de proyectos software, desarrollo de interfaces de usuario y programación. La clave del éxito del proyecto residirá en realizar una planificación realista y de calidad y, por supuesto, cumplir con la misma lo máximo posible.

***The Secret Of The Pyramid*** (el título del videojuego) nos sumergirá en una aventura a través de las ruinas de una pirámide para descubrir todos sus secretos. El jugador tomará el control de **Jaina**, una intrépida arqueóloga, con el fin de superar todas las trampas y enemigos que presentarán las ruinas.

### 1.2. Objetivos del proyecto

El objetivo de este trabajo de fin de grado es crear un videojuego desde cero hasta conseguir un producto completamente jugable y cuya dificultad se adapte a la habilidad de cualquier jugador. El juego se desarrollará en un ambiente 2D de aspecto retro con el fin de abarcar todas las facetas creativas posibles.

El videojuego se basa en mecánicas de acción y plataformas, que son dos de los géneros más representativos del sector. De hecho, "The Legend of Zelda" y "Super Mario Bros" son respectivamente los adalides de cada uno de estos géneros. Sin embargo, es la combinación de ambos lo que consigue un efecto trepidante y adictivo ya que el jugador tendrá que estar atento al entorno en todo momento para evitar caer de alguna plataforma al mismo tiempo que se enfrenta a los enemigos. El jugador tendrá que mejorar su habilidad en el juego para conseguir superarlo, siendo la clave de esto último la **rejugabilidad**.

Al tratarse de un juego de acción y plataformas, el objetivo será añadir mecánicas de salto y disparo. Además, para que el juego sea interesante, será necesario implementar una Inteligencia Artificial que complete y modifique la dificultad. Por la misma razón, el videojuego deberá contar con una ambientación adecuada y un diseño consistente en su conjunto, es decir, que la música, la paleta de colores, el personaje, los enemigos y las trampas deberán ser acordes conforme a un tema, que en este caso se tratará del antiguo Egipto.

Se pretende también adquirir el conocimiento necesario para manejar un motor gráfico que permita la creación de un videojuego completo, como Unity, y ganar soltura con los aspectos creativos más básicos de un videojuego, como puede ser la creación y edición de imágenes en dos dimensiones o la creación de audios sencillos y melodías.

### 1.3. Estructura del documento

En el apartado 1 se presenta una introducción en la que se contextualizan los videojuegos en la sociedad actual y se detallan los objetivos del proyecto.

En el apartado 2, se da un análisis sobre las distintas alternativas de motores de videojuegos y el porqué de su elección.

Después, en el apartado 3, se especifican el género, tipo de gráficos, audiencia y plataforma elegida donde poder jugar.

Tras esto, en el apartado 4, se describen las mecánicas y los distintos elementos del videojuego.

En el apartado 5 se concreta la metodología de desarrollo software y se justifica la elección de esta.

En el apartado 6 se describe la planificación seguida para el desarrollo del proyecto con su diagrama de Gantt asociado.

Seguidamente, en el apartado 7, se relata la ejecución del proyecto a través de cada una de las fases previamente comentadas en la planificación.

Finalmente, en el apartado 8 se presentan los resultados mediante varios diagramas de componentes y se detallan las conclusiones del proyecto.

## 2. PLATAFORMA DE DESARROLLO: MOTORES DE VIDEOJUEGOS

El primer paso para desarrollar un videojuego será decidir con qué motor gráfico hacerlo. Este no tiene porqué ser el mejor motor existente, sino el que mejor se adapte a los requisitos del proyecto.

En este apartado se explicará lo que es un motor de videojuego, se presentarán algunos ejemplos y se hará una comparación entre los tres motores más convenientes para el proyecto tras lo cual se elegirá uno para el desarrollo del videojuego.

### 2.1. ¿Qué es un motor gráfico?

Un motor gráfico o de videojuegos es un término que hace referencia a una serie de librerías de programación que permiten el diseño, la creación y la representación de un videojuego. (Carrasco, 2018). El motor provee al desarrollador de un renderizado de gráficos 2D o 3D según el caso, un sistema de sonido, scripting, cálculo de físicas y detección de colisiones y animación.

Una de las causas principales del crecimiento de la escena **indie** es la existencia de una gran variedad de motores de alta calidad al alcance de todo el mundo. Cada vez más personas son capaces de crear un videojuego utilizando estos motores, en muchos casos incluso de manera gratuita.

Ejemplos de estos motores podrían ser: **Unreal Engine** de gran calidad gráfica, **Source2**, motor de las sagas clásicas "Half-Life" y "Portal", **Unity**, que equilibra potencia y versatilidad, **GameMaker**, motor del indie "Hotline Miami" o **Godot**, motor completamente Open Source (cualquier persona puede acceder al código fuente del programa y editarlo personalmente).

De entre todos los motores disponibles, se ha decidido analizar tres de ellos por ser gratuitos y ofrecer características que podrían interesar para este proyecto. Para su evaluación se considerarán las siguientes características:

- **Dificultad de aprendizaje:** el motor no debe ser muy complejo de utilizar dado que al no tener ningún tipo de experiencia previa usando un motor gráfico esto podría complicar el avance del proyecto.
- **Calidad gráfica:** el motor debe ofrecer unas buenas características gráficas para que el videojuego se vea bonito y bien.
- Calidad de las físicas: el sistema de colisiones, fuerzas y transformaciones debe ser de buena calidad e independientes de la plataforma para evitar crear un juego inestable que dé lugar a diferentes experiencias entre los usuarios.
- **Documentación, tutoriales y comunidad:** al no poder invertir mucho tiempo en el aprendizaje del motor, será necesario que existan una buena cantidad de tutoriales, documentación y foros donde poder consultar las dudas que vayan surgiendo a lo largo del desarrollo, ya que se aprenderá a utilizar la tecnología al mismo tiempo que se utiliza.
- **Precio:** a pesar de que los tres motores a analizar sean gratuitos, a la hora de distribuir el videojuego esto puede cambiar. No será la característica más determinante, pero si será importante su discusión de cara al futuro del proyecto.

### 2.2. Unity

El motor de videojuegos de Unity es la solución líder a nivel mundial de creación de videojuegos. Es la elección del 45% de los desarrolladores del mundo, siendo por tanto el motor más usado mundialmente.

Esto no es de extrañar ya que es un motor gratuito y fácil de instalar en distintas plataformas (PC o Mac) y permite el desarrollo multiplataforma, es decir, que con el mismo código se puede tener un videojuego en Android, iOS, Windows, PlaySation, Xbox 360 y navegadores Web.

Como consecuencia de su fama, existe una gran cantidad de documentación en internet sobre cómo trabajar con la plataforma. Además, Unity integra la **Asset Store**, que consiste en una tienda donde se pueden descargar recursos creados por otros desarrolladores de manera gratuita o por un módico precio. Todo esto añadido a la gran facilidad de su interfaz acaba resultando en un motor muy conveniente y efectivo para la creación de un primer videojuego, sobre todo para equipos pequeños.

La versión Personal de Unity es completamente gratuita e incluye todas las características disponibles del motor excepto servicios en la nube y el desarrollo para consolas. Se pueden comercializar los videojuegos creados en esta versión sin tener que pagarle nada a Unity siempre y cuando los ingresos no superen los 100.000 $ brutos anuales.

**Ventajas**

- Gran cantidad de documentación y tutoriales gracias a su accesibilidad.

- Gran versatilidad: se puede crear sin demasiada dificultad cualquier tipo de videojuego (2D, 3D, plataforma, FPS, RPG) para cualquier tipo de plataforma.

- La Asset Store completa la accesibilidad ofreciendo una infinidad de soluciones, addons, recursos gráficos, etc. de manera gratuita en muchos casos.

- Permite el uso del IDE Visual Studio Code y se usa el lenguaje **C#**, que es un lenguaje orientado a objetos muy sencillo de aprender y utilizar.

**Desventajas**

- Los gráficos y las animaciones que pueden crearse no son tan bonitas como las de otros motores especializados en esto.

- El hecho de que tanta gente use este motor, hace que exista una mayor cantidad de videojuegos de mala calidad, lo cual aporta una mala fama al motor y que un juego esté creado con Unity puede influenciar negativamente a un consumidor.

### 2.3. Godot

Godot es sin duda el motor más recientemente publicado de la lista, siendo su fecha de lanzamiento febrero de 2014. La gran característica de este motor es que es de código abierto, por lo que su uso es completamente gratuito y se pueden desarrollar características para el mismo motor que ayuden con la creación del videojuego. Funciona en Windows, Mac y Linux y permite la exportación de sus videojuegos a estas mismas plataformas, además de dispositivos móviles (Android o iOS) y web.

A pesar de no contar con tanta cantidad de documentación como Unity, ni tener una Asset Store donde descargar módulos ya desarrollados, Godot ofrece un motor 2D completo, avanzado e independiente que no necesita falsificar el 2D en un espacio 3D (como Unity o Unreal) y un sofisticado sistema de animación de gran calidad con soporte para escenas animadas y animación por esqueletos.

Es un motor orientado a equipos pequeños y proyectos sencillos. Ofrece lo necesario de manera relativamente sencilla, aunque completamente libre. Gracias a ser Open Source, la cantidad de características que posee se incrementan habitualmente, lo que da un gran potencial a este motor.

**Ventajas**

- Código abierto y de libre uso, por lo que se puede monetizar cualquier producto sin necesidad de licencias y se puede modificar el motor para conseguir determinados comportamientos en un videojuego.

- Sistema de animación y motor 2D de muy alta calidad.

- Soporte multiplataforma.

- Permite el uso del **IDE** (Entorno de Desarrollo) Visual Studio Code y se usa el lenguaje C#.

**Desventajas**

- Los gráficos y las animaciones que pueden crearse no son tan bonitas como las de otros motores especializados en esto.

- No existe tanta documentación online y la interfaz es poco intuitiva y difícil de manejar.

- No es una plataforma reconocida y además no es sencillo aprender a utilizarla.

### 2.4. Unreal Engine

Unreal Engine, desarrollado por Epic Games, es sin duda el motor con mayor capacidad gráfica de los comparados. Ofrece una gran cantidad de herramientas para manejo de partículas e iluminación dinámica. Al contrario que los dos ejemplos anteriores, este motor está orientado a equipos grandes y que se dediquen a desarrollar videojuegos de alta calidad, conocidos como Triple A.

El motor fue desarrollado para la realización de juegos **FPS** (First Person Shooter), pero se ha acabado adaptando a una gran variedad de géneros, como sigilo, lucha o RPG. Utiliza C++ como lenguaje, el cual es bastante conocido entre desarrolladores, y al igual que Unity permite desarrollar para cualquier plataforma.

Unreal tiene un sistema de Blueprint, que funciona de manera parecida a los scripts, pero sin necesidad de saber programar. Además, están diseñados de manera que se complementen con el lenguaje C++, por lo que es una muy buena herramienta a la hora del desarrollo.

Al igual que Unity, dispone de una tienda, la Market Place, donde se pueden adquirir recursos y componentes creados por otros desarrolladores. Además, cada mes Unreal regalará algunos de estos componentes para su uso en Unreal, lo cual permite el acceso a herramientas de muy alta calidad por el simple hecho de utilizar este motor.

Finalmente, Unreal cuenta con un módulo muy potente para gestionar juegos en 2D de cualquier tipo (2D puro o 2.5D): Papers 2D, que permite la gestión de profundidad, colisiones, físicas, orden de elementos, menús…

Actualmente este motor es completamente gratuito, pero si algún producto desarrollado con Unreal consigue unas ganancias superiores a los 3000$ en 3 meses, se deberá abonar un 5% de dichos beneficios como royalties.

**Ventajas**

- Gran cantidad de documentación y tutoriales.

- Opciones gráficas de muy alta calidad al acceso de cualquier persona.

- Market Store donde descargar componentes creados por otros desarrolladores

- Permite el uso del IDE Visual Studio Code y se usa el lenguaje **C#**, que es un lenguaje orientado a objetos muy sencillo de aprender y utilizar

**Desventajas**

- Orientado a equipos grandes

- Dificultad de aprender en poco tiempo

### 2.5. Elección de motor

La elección del motor debe realizarse siempre en base a las condiciones del proyecto. Se quiere crear un juego en 2D, desarrollado por un equipo de una sola persona en una cantidad de tiempo de aproximadamente 3 meses.

Los tres motores ofrecen características muy buenas para el desarrollo de videojuegos en 2D, sin embargo, el mayor fuerte de Unreal reside en el renderizado de juegos en 3D o 2.5D, pero como no se poseen conocimientos sobre modelado 3D ni tiempo para adquirirlos, esta opción será menos apetecible.

Para ser capaces de completar el proyecto en la cantidad de tiempo estimada, lo más conveniente será utilizar un motor cuyo aprendizaje no requiera demasiado tiempo ni sea muy tedioso. Con este motivo, Unreal quedará completamente descartado, dado que, al estar orientado a equipos grandes y proyectos complejos, el software es más sofisticado y requiere de mucho más tiempo para ser aprendido.

En cuanto a los dos restantes, Unity será la opción más lógica, ya que es el motor con la interfaz más sencilla y al ser utilizado por tanta gente dispone de mucha más cantidad de información para ayudar a la hora de aprender a utilizar el software. Godot requiere mucho más tiempo para su aprendizaje, ya que a pesar de existir foros no existen una gran cantidad de tutoriales que muestren cómo desarrollar las partes más básicas de un videojuego.

A continuación, una tabla que resume la elección del motor en función de sus ventajas.

| Comparativa de Motores de Videojuegos | Unreal Engine 4 | Unity | Godot |
|---|---|---|---|
| **Dificultad de aprendizaje** | Alta | Baja | Media |
| **Calidad gráfica** | Muy alta | Media | Media |
| **Calidad del motor de físicas** | Muy alta | Alta | Media |
| **Cantidad de documentación** | Muy alta | Muy alta | Media |
| **Cantidad de tutoriales** | Alta | Muy alta | Baja |
| **Actividad de la comunidad** | Media | Muy alta | Media |
| **Gratuito** | Por debajo de los 3.000$ trimestrales | Por debajo de los 100.000$ anuales | Siempre |

*Tabla 1: Comparación entre motores de videojuegos*

Se ha elegido Unity 2018.3 para realizar el proyecto debido, principalmente, a que el tiempo de ejecución es limitado y el equipo consiste en una sola persona, por lo que será necesario que el aprendizaje del motor no requiera mucho tiempo y poder disponer de la máxima cantidad de recursos posible para facilitar la creación. Además, existe una gran cantidad de documentación y tutoriales que harán aún más rápido este proceso. Finalmente, al estar el sistema basado en componentes, la organización y reutilización de partes comunes es mucho más sencilla, lo que facilita mucho la creación de niveles.

## 3. ESPECIFICACIONES DEL VIDEOJUEGO

Una vez decidida la plataforma, el siguiente paso será definir los aspectos del videojuego que lo definirán.

En este apartado se describirán las características que definen el estilo del videojuego y por tanto la forma en que el usuario percibirá el mismo. Estas serán su género, su estilo gráfico, la plataforma en que podrá ser jugado y la audiencia a la que irá dirigido.

### 3.1. Género

Al igual que en la música, el cine y el arte en general, los videojuegos también pueden ser clasificados conforme a un género. Sin embargo, la diferencia que hace único a este producto audiovisual es que su clasificación dependerá fundamentalmente de la **mecánica de juego.** También dependerá de la estética y ambientación, pero en menor medida.

En el género de **acción** el jugador tendrá que hacer uso de su habilidad, reflejos y puntería para superar los problemas planteados por el videojuego, los cuales suelen darse en un contexto de combate o de superación de obstáculos. Es considerado el género más amplio, llegando a abarcar varios subgéneros, como el de disparos (**shooters**), el de peleas (**beat'em up**) o el de **plataformas**.

En el género de plataformas, el jugador controlará a un personaje con el que tendrá que evitar obstáculos físicos corriendo, saltando, agachándose o escalando.

La combinación de ambos géneros sumerge al jugador en un mundo donde la llave para la victoria serán su propia habilidad y reflejos para sortear enemigos y obstáculos. Esta mezcla suele ofrecer una experiencia trepidante y adictiva, ya que la única forma de mejorar será conociendo mejor el videojuego para poder anteponerse a sus peligros, lo cual implica la repetición numerosa de un mismo nivel.

### 3.2. Gráficos

Los videojuegos de plataformas están estrechamente relacionados con las dos dimensiones por su naturaleza, ya que son las limitaciones de espacio de las 2 dimensiones las que hacen tan interesante la exploración del movimiento en estos entornos. De hecho, aún en la época de las tres dimensiones, la mayoría de los títulos de este género son en 2 o **2.5 dimensiones**.

La intención de este videojuego es recuperar la esencia de los clásicos y combinarla con la tecnología de hoy en día. Es por esto por lo que toda la estética del videojuego está basada en los diseños 16-bits.

### 3.3. Plataforma

La plataforma elegida para el videojuego será ordenador Windows, Mac o Linux por las razones siguientes:

- Disponer de la plataforma donde probar el videojuego.
- En ordenador el rendimiento gráfico es mucho mejor.
- Una gran cantidad de la población dispone de ordenador personal, siendo concretamente Windows el sistema operativo más extendido.
- Se pueden utilizar sistema de distribución de aplicaciones como **Steam**, que permitirá capitalizar el producto.

Cualquier opción de desarrollo para videoconsola, como Nintendo Switch, Play Station 4 o Xbox One ha sido descartada dado que al no se dispone de la infraestructura necesaria para probar el videojuego ni de las licencias necesarias de Unity.

Por falta de tiempo no se ha considerado podido implementar la opción móvil, que en un principio era una de las intenciones, pero sin duda sería una plataforma idónea para el videojuego

### 3.4. Propósito y audiencia

La razón de ser de *El Secreto de la Pirámide* es ofrecer una experiencia desafiante y entretenida. Es un videojuego intenso lleno de secretos y retos.

La audiencia potencial de este videojuego vendrá definida, en un principio, por el sistema de clasificación por edades usado en Europa: **PEGI**. Este sistema nació en 2003 con motivo de englobar los sistemas nacionales ya existentes en un uno único que valiera para toda Europa (PEGI). Su función es ayudar a los padres a decidir de manera informada sobre los videojuegos que compran. Está respaldado por los desarrolladores de videojuegos, editores y por las tres grandes empresas dedicadas a la creación de videoconsolas: Nintendo, Sony y Microsoft.

De todos los posibles descriptores de contenido que hay en PEGI: drogas, palabrotas, sexo, apuestas, etc. sólo interesa el de violencia dado que es lo único presente en el videojuego. Por tanto, el videojuego podría ser tanto PEGI 7 como PEGI 12. Al ser la violencia presente en un entorno de fantasía y no es ni detallada ni realista, El Secreto de la Pirámide obtendría una clasificación de PEGI 7, es decir, que estará recomendado para personas con 8 años o más.

## 4. ELEMENTOS Y MECÁNICAS DEL JUEGO

Una vez definido el estilo del videojuego, se decidirán las mecánicas de juego de este y los elementos basados en dicho estilo que acabarán formando el videojuego.

En este apartado se describen las mecánicas del juego a través de sus objetivos, criterios de éxito y fracaso, controles y flujo del juego. También se describen todos los elementos gráficos del juego y su funcionalidad dentro del mismo. Finalmente, se explicarán la música y efectos de sonido creados.

### 4.1. Objetivo

El usuario tendrá que llegar al final de cada nivel sin morir para completar el videojuego. Por el camino, podrá recoger tesoros y acabar con enemigos para conseguir más puntuación. Podrá volver a empezar las veces que quiera para superar su propia marca tanto de tiempo como de puntuación.

### 4.2. Criterios de éxito

Los criterios de éxito están directamente relacionados con los objetivos, dado que se considerará como exitoso completar un nivel en el menor tiempo posible y con la máxima puntuación.

### 4.3. Criterios de fracaso

El criterio de fracaso será que la vida del personaje llegue a 0, cuya penalización será tener que volver a empezar desde el principio.

### 4.4. Controles

Los controles son bastante similares a los usados en otros juegos de ordenador. Para controlar el movimiento del personaje principal, se podrán utilizar tanto las teclas de dirección como las teclas A (izquierda) y D (derecha). Para activar el disparo, se tendrá que utilizar la tecla R y para saltar la barra espaciadora. Finalmente, la tecla ESCAPE servirá para pausar el juego dentro de un nivel.

### 4.5. Nivel

Un nivel es el espacio donde el jugador interaccionará con los elementos del videojuego para cumplir los objetivos definidos en el subapartado **4.1 Objetivos**. En el caso de este videojuego la progresión es lineal por lo que los niveles consistirán en secciones de un mundo más grande y el jugador completará el videojuego cuando supere cada nivel. Cada nivel tendrá una instancia del personaje principal y el resto de los elementos especificados a continuación (enemigos, decoraciones, trampas y tesoros) repartidos a lo largo de todo su recorrido

#### 4.5.1. Personaje principal

| Nombre | Animación | Descripción |
|---|---|---|
| **IdleJaina (Jaina parada)** |  | Animación que se reproduce en bucle cuando el personaje está parado |
| **RunningJaina (Jaina corriendo)** |  | Animación que se reproduce en bucle cuando el personaje está en movimiento |
| **JumpingJaina (Jaina saltando)** |  | De izquierda a derecha, cada **sprite** se reproducirá cuando el personaje esté ascendiendo, cuando llegue al punto más alto y cuando empiece a caer. |
| **LandingJaina (Jaina aterrizando)** |  | Animación de transición entre el salto y volver a estar en el suelo. |

*Tabla 2: Personaje principal*

### 4.5.2. Enemigos

| Nombre | Animación | Descripción |
|---|---|---|
| **Spider (Araña)** |  | Tienen 3 puntos de vida y se mueven en bucle vertical mente entre suelo y techo. Enfadados perseguirán al jugador cuando estén bajando del techo. Están en grutas estrechas. |
| **Bat (Murciélago)** |  | Tienen 2 puntos de vida y disparan un proyectil al jugador. Enfadados dispararán un abanico de 3 proyectiles. Están en sitios altos y abiertos. |

*Tabla 3: Enemigos*

### 4.5.3. Tesoros

| Nombre | Sprite | Descripción |
|---|---|---|
| **Small Treasure (Tesoro pequeño)** |  | Otorgan 1 punto al jugador y están por todo el mapa indicando el camino a seguir. |
| **Medium Treasure (Tesoro mediano)** |  | Otorgan 3 puntos al jugador y están en lugares relativamente difíciles de acceder. |
| **Big Treasure (Tesoro grande)** |  | Otorgan 25 puntos al jugador, sólo hay 1 en cada nivel y son muy difíciles de encontrar y conseguir |
| **Health (Vida)** |  | Otorgan un punto de salud al jugador y están en zonas difíciles de acceder o como recompensa al final de cada nivel. |

*Tabla 4: Tesoros*

### 4.5.4. Elementos del nivel

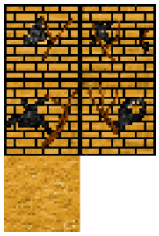
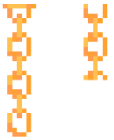
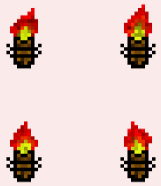
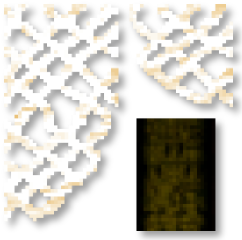
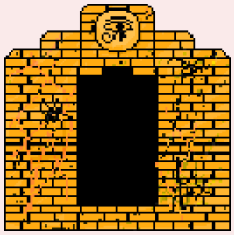
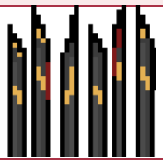
| Nombre | Sprites | Descripción |
|---|---|---|
| **Frontground (Primer plano)** |  | Utilizados para los suelos, paredes y techos. |
| **Background (Fondo)** |  | Se utiliza como fondo en todos los niveles. |
| **Chain (Cadena)** |  | Se utiliza como decoración en todos los niveles. |
| **Torch (Antorcha)** |  | Se utiliza como animación (para simular el movimiento del fuego) y se sitúa en columnas y puertas como elemento decorativo. |
| **Decorations (Decoraciones): Telarañas y columnas** |  | Se utilizan como elementos estáticos de decoración repartidos por los niveles. Se ha añadido sombra a la imagen para poder distinguir mejor las telarañas. |
| **Door (Puerta)** |  | Si están al principio del nivel, sólo son decoración, pero al final del nivel será la meta que el jugador tiene que alcanzar para avanzar de nivel. |
| **Spikes (Pinchos)** |  | Se situarán en fosas y supondrán una amenaza letal para el jugador. |

*Tabla 5: Elementos del nivel*

### 4.5.5. Proyectiles y partículas

| Nombre | Sprite | Descripción |
|---|---|---|
| **Bullet (Bala)** |  | Será el proyectil disparado por el jugador. |
| **Blood (Sangre)** |  | Será el proyectil de los murciélagos y también se reutilizará para el efecto de sangrado al recibir daño. |
| **GunSmoke (Humo de pistola)** |  | Se utilizará como partícula de humo cuando se dispare. |

*Tabla 6: Proyectiles y partículas*

### 4.5.6. Interfaz Gráfica

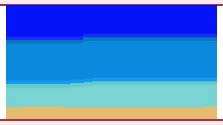
| Nombre | Sprite | Descripción |
|---|---|---|
| **BloodScreen (Pantalla de sangre)** |  | Cubrirá toda la pantalla del juego cuando dañen al personaje principal de manera que alerte al jugador del peligro. |
| **Button (Botón)** |  | Se utilizará para los botones del menú principal y de pausa |
| **FinishBackground (Fondo del final)** |  | Se utilizará como fondo en la escena de victoria al final del videojuego. |
| **Heart (Corazón)** |  | Representará la vida restante del jugador. Se encontrará arriba a la izquierda en los niveles de juego. |

*Tabla 7: Interfaz gráfica*

### 4.6. Música y sonido

La música será creada con **OpenMPT**, que es un programa de seguimiento de módulos de audio gratuito, para darle un estilo **chiptune**. La idea es que suene old-school, pero al mismo tiempo mantenga una atmósfera de exótica y misterio, como el antiguo Egipto.

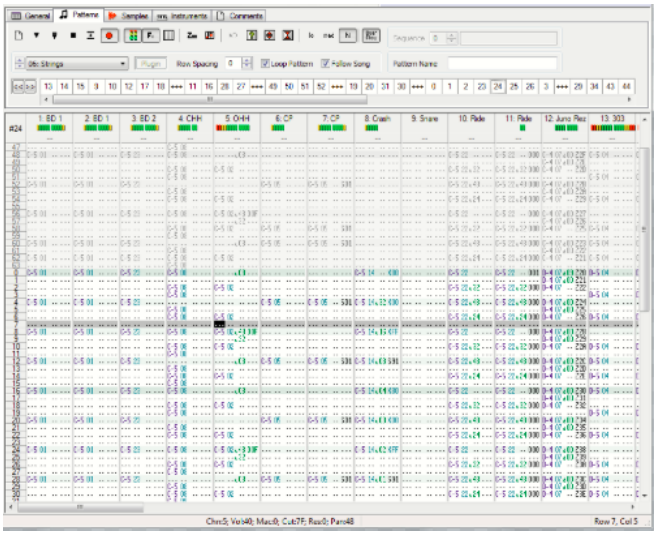


*Ilustración 1: Interfaz de OpenMPT*

Para los efectos de sonido se utilizará **Bfxr**, que es un programa especializado en esto. Estos efectos están detallados en el apartado 7. Ejecución del proyecto, 4.4 Efectos de sonido. En resumen, existirá un efecto de sonido para el salto del jugador y su disparo, el disparo de los murciélagos, obtener un tesoro o dañar o ser dañado.

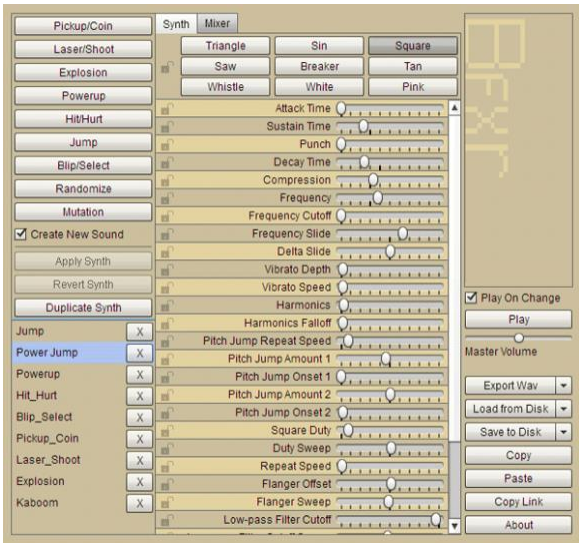



*Ilustración 2: Interfaz de Bfxr*

### 4.7. Flujo del videojuego

El diseño de los niveles está hecho de manera que se incremente la dificultad conforme el jugador avance ofreciendo en cada momento un reto más difícil pero siempre justo. De esta manera es la habilidad del jugador la que mejora con el videojuego y no su personaje. Además, permitirá a los jugadores nuevos a acostumbrarse a los controles y el juego antes de enfrentarse a los retos más difíciles, al mismo tiempo que el jugador veterano tendrá la experiencia del reto que busca.

# 5. METODOLOGÍA DEL DESARROLLO SOFTWARE

El desarrollo de un videojuego es un proceso complicado y de gran magnitud. Por ello habrá que tomar el rol de gestor de proyectos además del de desarrollador. Esto significa que se realizarán de manera sincronizada tanto tareas de gestión y planificación como de desarrollo y testeo.

En este apartado se definirá el proyecto y su alcance, de manera que se pueda realizar una planificación del trabajo que defina el ciclo de vida de la gestión del proyecto.

## 5.1. Definición del proyecto

El videojuego será en 2 dimensiones y su género será acción y plataformas. El objetivo será ofrecer a los jugadores un desafío trepidante e intenso en el que poder probar sus habilidades al límite y pasar un buen rato intentando completar un videojuego que les hará exigir lo mejor de sí mismos.

El juego estará compuesto por varios niveles que el jugador completará llegando al final de estos. La longitud de los niveles no será demasiado grande (aproximadamente 2 minutos por nivel) para alentar al jugador a volver a empezar si pierde, lo que le ayudará a mejorar.

A lo largo del escenario, habrá diferentes trampas y enemigos. Las trampas estarán relacionadas con el factor plataformas del videojuego, ya que serán indestructibles y eliminarán al jugador al contacto. Los enemigos tendrán diferentes características (vida, comportamiento, apariencia…). Eliminar enemigos dará puntos al jugador, al igual que recoger las vasijas repartidas por todo el nivel.

El usuario usará las teclas de dirección para controlar el movimiento del personaje principal. Con el espacio realizará un salto y con la tecla R disparará un proyectil. Tendrá que usar estas habilidades para esquivar trampas y eliminar enemigos. El disparo tendrá un enfriamiento o **cooldown**, que significa que el jugador tendrá que esperar un tiempo antes de volver a usarlo (como si fuera la cadencia de un arma). Esto añade dificultad al juego y evita que se abuse de la mejor defensa que tendrá el jugador contra los obstáculos que se le presenten: su ataque.

Habrá una cámara ortogonal que siga al jugador por el nivel de manera fluida y mantenga la resolución del videojuego de manera que en cualquier dispositivo se vea de la misma forma.

Todos los elementos del juego contarán con una animación (enemigos, jugador y elementos decorativos). Las acciones del personaje y los enemigos tendrán sonidos. La música del proyecto se ambientará en Egipto con un estilo arcade y será compuesta personalmente.

La interfaz de usuario reflejará la información más importante del jugador (tiempo, puntos y vida) ocupando la menor cantidad de pantalla posible.

Habrá un menú de pausa desde el que se podrá volver al menú principal o salir del juego. Habrá un menú principal desde el que se pueda empezar una partida, consultar los controles o salir del juego. Habrá un menú de victoria y de derrota jugables, desde los que se podrá empezar una nueva partida o quitar el juego.

La ambientación será oscura y misteriosa. El juego deberá tener una **curva de dificultad** exigente pero equilibrada, de manera que el jugador pueda tener un aprendizaje consistente antes de empezar a

afrontar los retos más difíciles. La curva de dificultad de un videojuego consiste en su dificultad o exigencia a lo largo de la duración de una partida. Los jugadores con una habilidad muy por encima de la curva no tendrán ninguna dificultad y se aburrirán, mientras que los que estén muy por debajo no serán capaces de conseguir avanzar y se frustrarán.

Con una curva de dificultad equilibrada, el jugador más casual podrá dominar rápido las mecánicas y progresar en el juego sin llegar a frustrarse, al mismo tiempo que el jugador más experto seguirá teniendo un desafío que afrontar sin aburrirse.

Todos los diseños gráficos, sonidos, animaciones y música se realizarán personalmente para asegurar una concordancia en el estilo y evitar problemas con el copyright.

### 5.2. Desglose de los requisitos principales y tiempo estimado

A continuación, se presenta una lista con los requisitos más importantes del videojuego y una estimación del tiempo de trabajo que costará su implementación.

| Número de requisito (Anexo) | Requisitos principales | Estimación aproximada del tiempo (Horas) |
|---|---|---|
| **1** | Personaje principal que se mueva con las teclas de dirección. Animaciones fluidas. | **15** |
| **2** | Disparo y salto del personaje principal. Interacción del proyectil con el entorno. | **15** |
| **3** | Enemigos animados y dotados de una IA. Interacción de estos con el entorno y el jugador. | **40** |
| **4** | Diseño de trampas y coleccionables. Interacción de estos con el jugador. | **10** |
| **5** | Implementar cámara que siga al jugador fluidamente y gestión de la resolución. | **5** |
| **6** | GUI no intrusiva que muestre la puntuación, salud restante y tiempo de juego. | **10** |
| **7** | Sistema de menús (inicial, pausa, opciones, victoria y derrota) | **30** |
| **8** | Efectos de sonido y música en consonancia con el estilo retro y la ambientación en Egipto. | **10** |
| **9** | Sistema de niveles completo (tutorial, nivel 1 y nivel 2) | **65** |

*Tabla 8: Requisitos principales y duración*

En total, se estima una duración de 200 horas. Como se trabajarán alrededor de 3 horas por día, el proyecto tomará 3 meses hasta ser completado. El desarrollo del proyecto se dividirá en 3 iteraciones que durarán un mes cada una y que darán lugar a una versión con la que poder realizar pruebas y sobre la cual seguir desarrollando las implementaciones de la siguiente iteración.

### 5.3. Metodología ágil

Tras haber obtenido una visión más general del alcance del proyecto, el siguiente paso será elegir el método de desarrollo a seguir que más beneficios aporte. Dadas las características del proyecto, parece lo más lógico optar por la metodología ágil o scrum. Al tratarse de un videojuego, los requisitos podrán cambiar sobre la marcha, pudiendo añadirse o eliminarse funcionalidades si se comprueba que estos cambios añadirán valor al producto.

Además, el equipo de desarrollo, formado por una persona, no posee experiencia en ningún proyecto parecido, lo que significa que se realizará un aprendizaje por descubrimiento, una característica propia de las metodologías agiles.

Finalmente, scrum nos permitirá contar con versiones funcionales del producto, que podrán ser mejoradas conforme a la retroalimentación obtenida en la fase de pruebas. Esto quiere decir que el desarrollo será incremental e iterativo, lo cual tiene mucho sentido en un videojuego ya que este debe ser constantemente probado y mejorado.

El problema de la metodología ágil en este proyecto lo encontramos a la hora de la retroalimentación mencionada previamente. Esto se debe a que el equipo está formado por una sola persona, por lo que no se podrán realizar reuniones donde se obtenga esta retroalimentación ni donde se pueda adaptar la planificación, las cuales son una parte vital de la metodología.

Para remediar esto, se contó desde el primer momento con la ayuda de personas con diferente habilidad, gusto y conocimiento en los videojuegos. Estas personas se encargaron de probar el producto en cada fase de pruebas, gracias a lo cual se pudieron descubrir ‘bugs’ que fueron remediados, además de opiniones y sugerencias que moldearon el producto hasta su resultado final. El hecho de que tuvieran diferentes estilos de juego aportó bastante a la hora de otorgar al juego un enfoque más abierto que permitiera la entrada de un público más general.

A continuación, se presenta una figura que describe la metodología ágil. En ella se puede ver cómo después de un ciclo de diseño, desarrollo y pruebas, se obtiene una versión del producto mejor que la anterior:

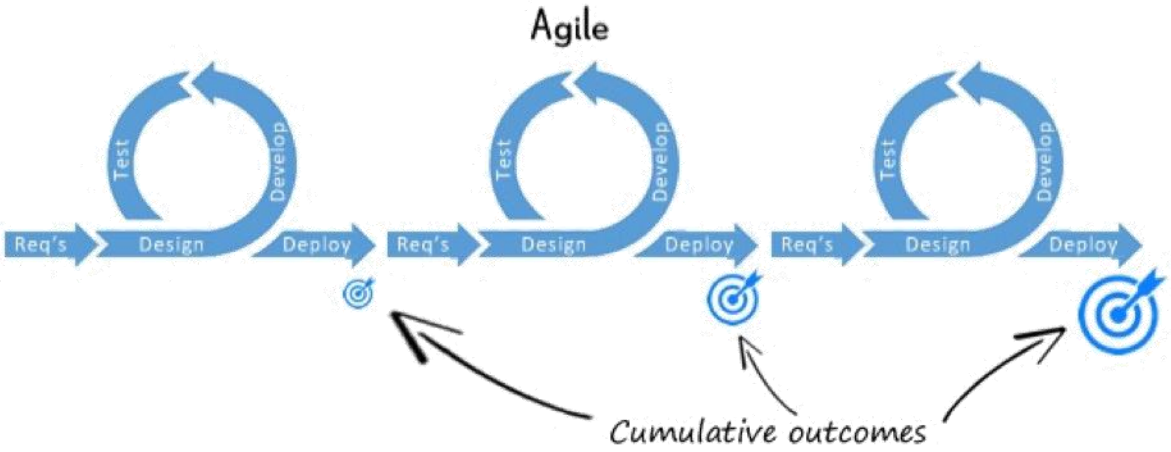



*Ilustración 3: Representación gráfica de la metodología ágil*

# 6. PLANIFICACIÓN

La planificación será una parte vital del proyecto dado que su cumplimiento definirá el éxito o fracaso de este.

En este apartado se detallarán los sprints en los que se ha repartido el proyecto y el diagrama de Gantt creado para el mismo. Lo que se persigue a partir de la planificación es abordar el desarrollo de los distintos sprints como hilo conductor del desarrollo del proyecto.

## 6.1. Definición de sprints

Scrum al ser una metodología de desarrollo ágil tiene como base la idea de creación de ciclos breves para el desarrollo, que comúnmente se llaman iteraciones y que en Scrum se llamarán **sprints** (Gallego, 2017).

El primer sprint consistirá en la creación de un entorno de pruebas y del personaje principal. Se empezará un proyecto nuevo, se utilizarán los componentes necesarios para hacer un nivel simple donde probar el movimiento, ataque con sus respectivas animaciones para el personaje principal con el que jugará el jugador.

Durante el segundo sprint se crearán todos los elementos del juego con los que el jugador puede interactuar, además del comportamiento de estas interactuaciones. Estos elementos serán:

- Los enemigos, los cuales consisten en una araña y un murciélago para los que se implementará una inteligencia artificial y animaciones.
- Las trampas y tesoros, que acabarán con el jugador o le recompensarán con puntos y salud extra.
- El sistema de salud del jugador y los enemigos, que dará lugar a las interactuaciones entre estos.

Para el tercer sprint, se implementará todo el sistema de menús y la interfaz gráfica en la que el jugador podrá obtener información del videojuego como la vida, puntos y tiempo. También se añadirán la música del juego y todos los efectos de sonido necesarios.

El cuarto y último sprint servirá para crear los niveles de juego, lo cual llevará mucho tiempo ya que obligará a iterar sobre tareas realizadas previamente para adaptar componentes a las necesidades de diseño que vayan surgiendo a lo largo de la implementación de los niveles. Finalmente, se realizarán las pruebas necesarias con otras personas y se cerrará el proyecto y creará la memoria.

## 6.2. Diagrama de Gantt

El gráfico de Gantt permite identificar la actividad en que se estará utilizando cada uno de los recursos y la duración de esa utilización, de tal modo que puedan evitarse periodos ociosos innecesarios y se dé también al administrador una visión completa de la utilización de los recursos que se encuentran bajo su supervisión (Hinojosa, 2003)

A continuación, se muestra el diagrama de Gantt donde se reflejan las precedencias de tareas. Este ha sido creado con el programa Proyect Libre:

| Name | Duration | Start | Predecessors |
|---|---|---|---|
| **Sprint 1: Creación del personaje principal y entorno de pruebas** | 27 days | **5/1/19 7:00 AM** | |
| Gestión de la arquitectura de la aplicación | 2 days | 5/1/19 7:00 AM | |
| **Creación de zona de pruebas** | 3 days | **5/3/19 8:00 AM** | **2** |
| Diseño | 1 day | 5/3/19 8:00 AM | |
| Implementación | 2 days | 5/6/19 8:00 AM | 4 |
| **Creación del personaje principal** | 22 days | **5/8/19 8:00 AM** | **3** |
| **Sistema de movimiento del jugador** | 12 days | **5/8/19 8:00 AM** | |
| **Animaciones de movimiento** | 3 days | **5/8/19 8:00 AM** | |
| **Movimiento del jugador** | 6 days | **5/13/19 8:00 AM** | **8** |
| **Creación de la cámara** | 3 days | **5/21/19 8:00 AM** | **11** |
| **Sistema de ataque del jugador** | 8 days | **5/24/19 8:00 AM** | **7** |
| **Animaciones de ataque** | 2 days | **5/24/19 8:00 AM** | |
| **Ataque del jugador** | 6 days | **5/28/19 8:00 AM** | **19** |
| Diseño | 3 days | 5/28/19 8:00 AM | |
| Implementación | 2 days | 5/31/19 8:00 AM | 23 |
| Pruebas | 1 day | 6/4/19 8:00 AM | 24 |
| Pruebas de integridad | 2 days | 6/5/19 8:00 AM | 18 |
| **Sprint 2: Creación de todos los elementos interactuables** | 26 days | **6/7/19 8:00 AM** | **6** |

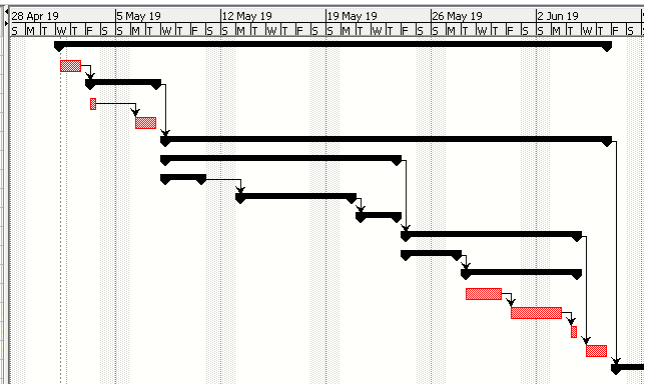



*Ilustración 4: Diagrama de Gantt del primer sprint*

| Name | Duration | Start | Predecessors |
|---|---|---|---|
| **Sprint 1: Creación del personaje principal y entorno de pruebas** | 27 days | **5/1/19 7:00 AM** | |
| **Sprint 2: Creación de todos los elementos interactuables** | 26 days | **6/7/19 8:00 AM** | **6** |
| **Enemigos** | 11 days | **6/7/19 8:00 AM** | |
| **Araña** | 5 days | **6/7/19 8:00 AM** | |
| Diseño | 1 day | 6/7/19 8:00 AM | |
| Implementación | 2 days | 6/10/19 8:00 AM | 30 |
| Pruebas | 2 days | 6/12/19 8:00 AM | 31 |
| **Murciélago** | 5 days | **6/14/19 8:00 AM** | **29** |
| Diseño | 1 day | 6/14/19 8:00 AM | |
| Implementación | 2 days | 6/17/19 8:00 AM | 34 |
| Pruebas | 2 days | 6/19/19 8:00 AM | 35 |
| Pruebas de integridad | 1 day | 6/21/19 8:00 AM | 33 |
| **Sistema de vida** | 7 days | **6/24/19 8:00 AM** | **28** |
| Diseño | 2 days | 6/24/19 8:00 AM | |
| Implementación | 3 days | 6/26/19 8:00 AM | 39 |
| Pruebas | 2 days | 7/1/19 8:00 AM | 40 |
| **Trampas** | 3 days | **7/3/19 8:00 AM** | **38** |
| Diseño | 1 day | 7/3/19 8:00 AM | |
| Implementación | 1 day | 7/4/19 8:00 AM | 43 |
| Pruebas | 1 day | 7/5/19 8:00 AM | 44 |
| **Recompensas** | 4 days | **7/8/19 8:00 AM** | **42** |
| Diseño | 1 day | 7/8/19 8:00 AM | |
| Implementación | 2 days | 7/9/19 8:00 AM | 47 |
| Pruebas | 1 day | 7/11/19 8:00 AM | 48 |
| Pruebas de integridad | 1 day | 7/12/19 8:00 AM | 46 |
| **Sprint 3: Creación de la interfaz gráfica, los menús y el sonido** | 24 days | **7/15/19 8:00 AM** | **27** |

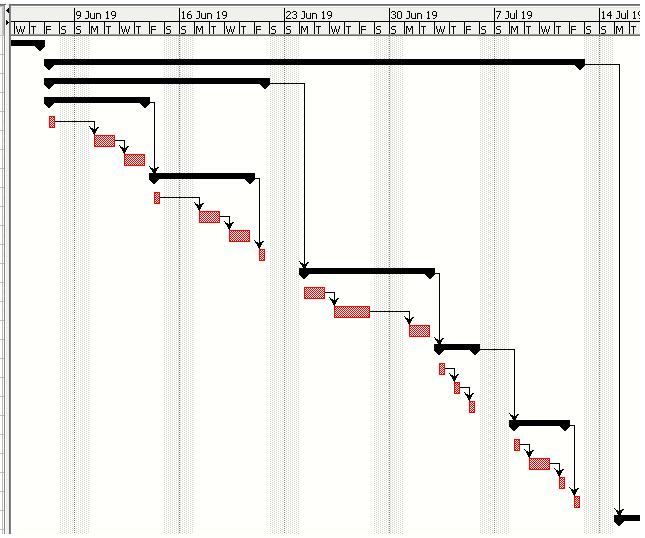



*Ilustración 5: Diagrama de Gantt del segundo sprint*

| Name | Duration | Start | Predecessors |
|---|---|---|---|
| **Sprint 2: Creación de todos los elementos interactuables** | 26 days | **6/7/19 8:00 AM** | **6** |
| **Sprint 3: Creación de la interfaz gráfica, los menús y el sonido** | 24 days | **7/15/19 8:00 AM** | **27** |
| **Salud, tiempo y puntuación** | 6 days | **7/15/19 8:00 AM** | |
| Diseño | 1 day | 7/15/19 8:00 AM | |
| Implementación | 3 days | 7/16/19 8:00 AM | 54 |
| Pruebas | 2 days | 7/19/19 8:00 AM | 55 |
| **Sistema de menús** | 12 days | **7/23/19 8:00 AM** | **53** |
| Diseño | 2 days | 7/23/19 8:00 AM | |
| **Menú principal** | 2 days | **7/25/19 8:00 AM** | **58** |
| Implementación | 1 day | 7/25/19 8:00 AM | |
| Pruebas | 1 day | 7/26/19 8:00 AM | 60 |
| **Menú de opciones** | 2 days | **7/29/19 8:00 AM** | **59** |
| Implementación | 1 day | 7/29/19 8:00 AM | |
| Pruebas | 1 day | 7/30/19 8:00 AM | 63 |
| **Menú de pausa** | 2 days | **7/31/19 8:00 AM** | **62** |
| Implementación | 1 day | 7/31/19 8:00 AM | |
| Pruebas | 1 day | 8/1/19 8:00 AM | 66 |
| **Menú de derrota** | 2 days | **8/2/19 8:00 AM** | **65** |
| Implementación | 1 day | 8/2/19 8:00 AM | |
| Pruebas | 1 day | 8/5/19 8:00 AM | 69 |
| **Menú de victoria** | 2 days | **8/6/19 8:00 AM** | **68** |
| Implementación | 1 day | 8/6/19 8:00 AM | |
| Pruebas | 1 day | 8/7/19 8:00 AM | 72 |
| **Creación de sonido** | 5 days | **8/8/19 8:00 AM** | **57** |
| Diseño | 4 days | 8/8/19 8:00 AM | |
| Implementación | 1 day | 8/14/19 8:00 AM | 75 |
| Pruebas de integridad | 1 day | 8/15/19 8:00 AM | 74 |
| **Sprint 4: Creación de los niveles, pruebas y cierre del proyecto** | 27 days | **8/16/19 8:00 AM** | **52** |

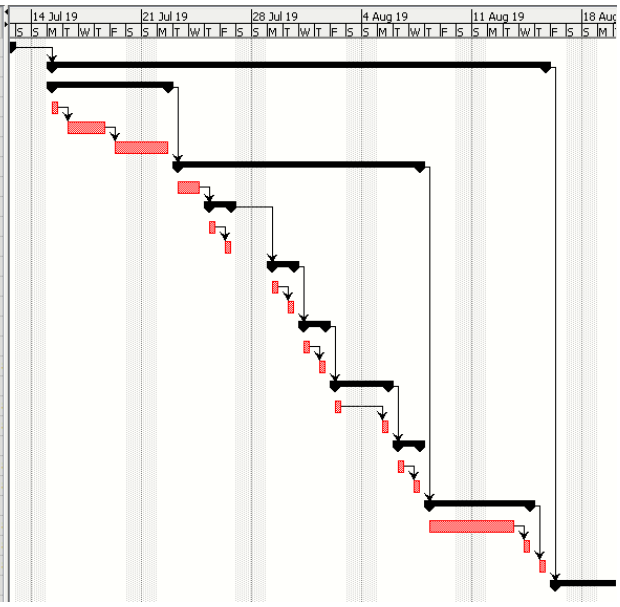



*Ilustración 6: Diagrama de Gantt del tercer sprint*

| Name | Duration | Start | Predecessors |
|---|---|---|---|
| **Sprint 3: Creación de la interfaz gráfica, los menús y el sonido** | 24 days | **7/31/19 8:00 AM** | **27** |
| **Sprint 4: Creación de los niveles, pruebas y cierre del proyecto** | 27 days | **9/3/19 8:00 AM** | **51** |
| **Creación de los niveles** | 24 days | **9/3/19 8:00 AM** | |
| **Tutorial** | 8 days | **9/3/19 8:00 AM** | |
| Diseño | 1 day | 9/3/19 8:00 AM | |
| Implementación | 5 days | 9/4/19 8:00 AM | 80 |
| Pruebas | 2 days | 9/11/19 8:00 AM | 81 |
| **Nivel 1** | 8 days | **9/13/19 8:00 AM** | **79** |
| Diseño | 1 day | 9/13/19 8:00 AM | |
| Implementación | 5 days | 9/16/19 8:00 AM | 84 |
| Pruebas | 2 days | 9/23/19 8:00 AM | 85 |
| **Nivel 2** | 8 days | **9/25/19 8:00 AM** | **83** |
| Diseño | 1 day | 9/25/19 8:00 AM | |
| Implementación | 5 days | 9/26/19 8:00 AM | 88 |
| Pruebas | 2 days | 10/3/19 8:00 AM | 89 |
| **Pruebas** | 2 days | **10/7/19 8:00 AM** | **78** |
| Pruebas de integridad | 2 days | 10/7/19 8:00 AM | |
| Pruebas de usuario | 2 days | 10/7/19 8:00 AM | |
| **Cierre del proyecto** | 1 day | **10/8/19 5:00 PM** | **91** |
| Generación del ejecutable | 1 day | 10/9/19 8:00 AM | |

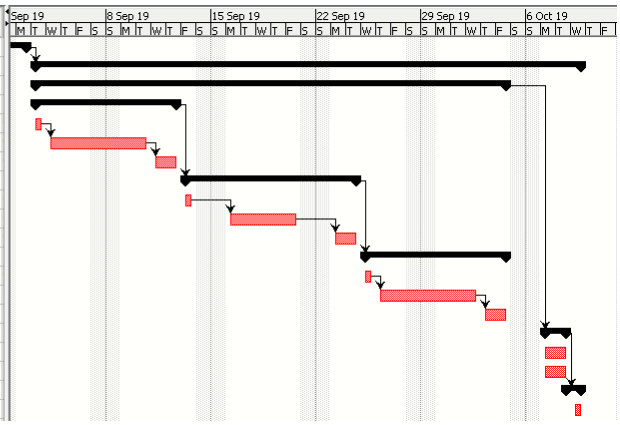



*Ilustración 7: Diagrama de Gantt del cuarto sprint*

Para concluir el proyecto, se creará el ejecutable y se elaborará la memoria, cuyo tiempo no ha sido considerado ya que no forma parte del proyecto como tal.

# 7. EJECUCIÓN DEL PROYECTO

En este apartado se recoge todo el proceso de desarrollo del videojuego a lo largo de los cuatro sprints explicados en el apartado anterior. Se explicarán las diferentes características de Unity utilizadas y la creación de las mecánicas a través de la implementación de componentes.

## 7.1. Sprint 1: Preparación y personaje principal

### 7.1.1. Preparación de Unity

Lo primero de todo consistirá en instalar Unity y crear un proyecto en 2 dimensiones. El proyecto empezará con una **scene** (escena) vacía y diferentes ventanas con funcionalidades que se describirán más adelante.

Las escenas serán las contenedoras del entorno del videojuego y sus menús. Esto quiere decir que tanto el nivel de juego, como el menú inicial y los créditos finales podrán estar separados en escenas diferentes. La ventaja de esto es que alivia la ejecución del juego, ya que solo habrá una escena cargada a la vez. Además, facilita el trabajo ya que al separar los niveles cada uno tendrá menos elementos y por tanto estarán menos sobrecargados.

Para organizar los diferentes elementos que se usarán en el proyecto, se utilizará la siguiente estructura de directorios.

| Addons | Funcionalidades añadidas |
|---|---|
| **Animations** | Clips con las animaciones usadas |
| **Fonts** | Fuentes usadas para el texto |
| **Materials** | Materiales usados para las particulas |
| **PhysicsMaterials** | Materiales usados para las fisicas |
| **Prefabs** | Objetos prefabricados (enemigos, coleccionables) |
| **Scenes** | Escenas del videojuego |
| **Scripts** | Todos los scripts |
| **Sound** | Música y sonido |
| **Sprites** | Elementos gráficos estáticos |
| **Tilemap** | Cuadrículas, paletas y pinceles para crear niveles |

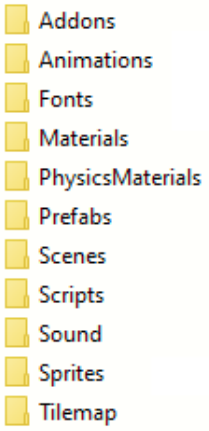



*Tabla 9: Árbol de directorios*

### 7.1.2. Apartado gráfico

Al tratarse de un videojuego en 2 dimensiones de estética arcade, todos los diseños están basados en el estilo 16-bits que podríamos encontrar en juegos como Super Mario Bros. Para ello, se ha utilizado **Piskel**, una herramienta de edición de **sprites**. Los sprites son mapas de bits en 2 dimensiones dibujados directamente en un destino de representación, careciendo por tanto de iluminación o efectos.

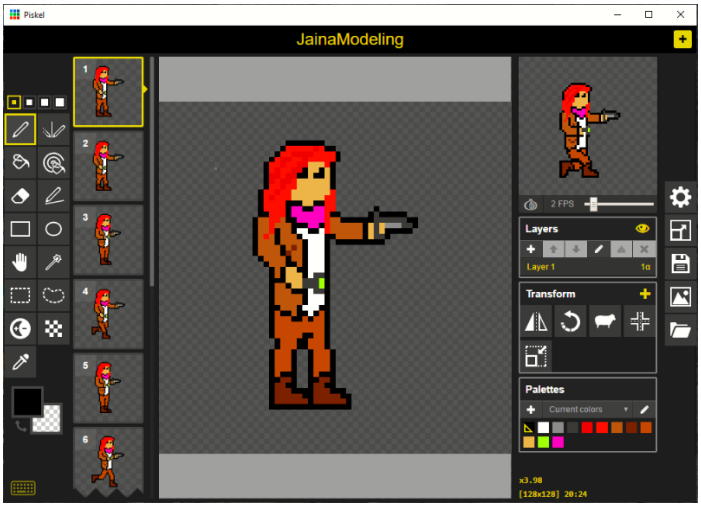



*Ilustración 8: Creación de Jaina en Piskel*

Gracias al editor, será posible no sólo la creación, sino también la exportación de estos archivos en un formato deseado, el cual será una hoja de sprites con todos los sprites de un elemento del videojuego, como puede ser el personaje principal, un enemigo o algún elemento decorativo.

Tras esto, se importarán los archivos al proyecto de Unity, donde se usará el editor de imágenes de Unity para cambiar el tipo de textura de png a Sprite 2D y, si es necesario, dividir la hoja de sprites en cada uno de sus elementos para obtener un Sprite múltiple que se usará para crear las animaciones.

Una vez tratados los diseños, se podrán usar en la escena. Para empezar, se creará una escena llamada Test que se usará para hacer pruebas en un entorno prescindible, para así evitar ocasionar cualquier problema a los niveles de juego reales. Esta se eliminará cuando se cree la aplicación para ahorrar memoria.

### 7.1.3. Creación del nivel: escenario de pruebas

Las escenas de Unity son los contenedores del entorno y los menús del juego. Cada escena representa un nivel único, y se usan para mantener limpio el proyecto y aliviar el uso de memoria al sólo cargar una escena cada vez, lo cual es una de las grandes ventajas ofrecidas por Unity.

Nada más crear una escena, por defecto contiene una cámara principal (**Main Camera**) y una luz direccional (D**irectional Light**). Por ser un videojuego en 2 dimensiones, la luz direccional no se usará y podrá eliminarse. Si el videojuego fuera en 2.5 dimensiones, sin embargo, esta luz sí que sería útil. La cámara, por otro lado, será el elemento más importante de la escena ya que dará al jugador la perspectiva del juego y será el elemento a través del cual reciba toda la información de su interacción con el programa.

El primer paso será crear un espacio donde poder situar al personaje principal. Para ello se usará una de las características más recientes de Unity: los **tilemaps**. Un tilemap es un **grid** compuesto de **tiles**. Un grid es una cuadricula de elementos relacionados. Los tiles son imágenes, en este caso de 16 bits, con las cuales

se crean imágenes más grandes y visualmente atractivas, como un nivel o área dentro del mismo. Pueden ser tanto rectangulares como isométricos, siendo en este caso cuadrados.

Gracias al **asset** del TileMap, será posible crear un escenario sin mayor complicación que situar las imágenes en espacios de la cuadrícula. Un asset es una librería creada por otro usuario que se puede adquirir en la tienda por el precio propuesto. En este caso, el asset es de Unity y está disponible gratuitamente. Estas imágenes estarán situadas en la Tile Palette, que consistirá en una herramienta que nos permitirá elegir el tipo de elemento con el que se quiere 'pintar' y el pincel que se quiere utilizar, habiendo también una opción para borrar elementos y otras utilidades características de una herramienta gráfica. En esta también se elegirá el tilemap donde se quiere pintar el elemento.

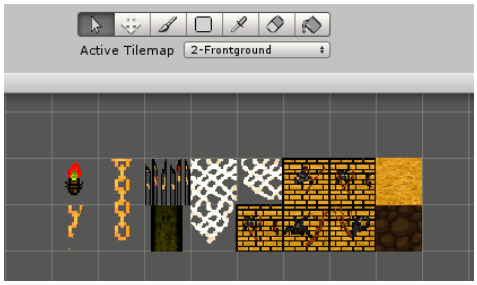



*Ilustración 9:Paleta de Tiles*

Se podrá también dotar a todo el tilemap de un collider que servirá para que los elementos no puedan atravesar el nivel y caerse a través del suelo si tienen gravedad.

Este nivel de pruebas será el escenario perfecto para probar las mecánicas de movimiento del personaje que se implementarán a continuación, como la velocidad, la altura del salto, etc.

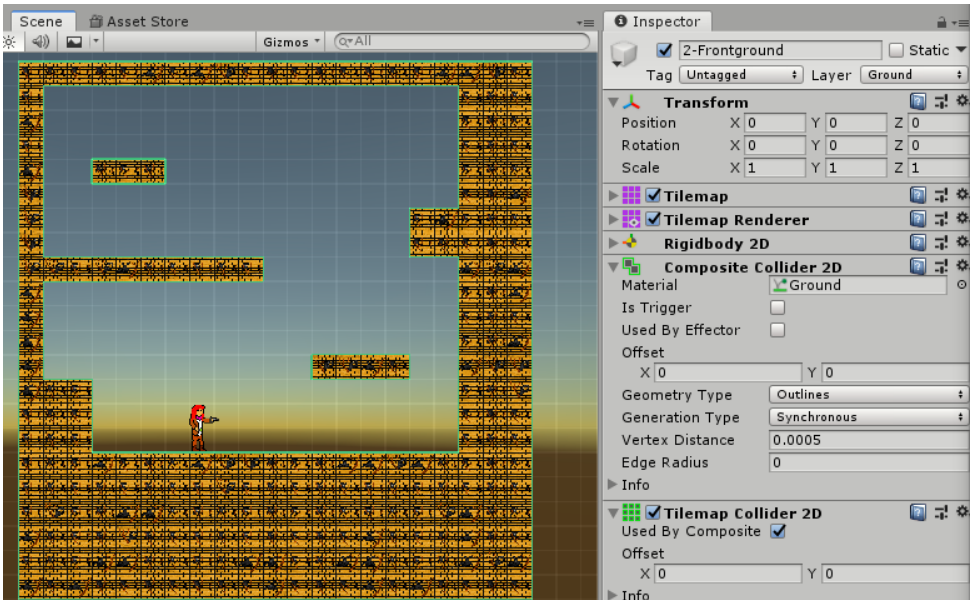



*Ilustración 10: Nivel de pruebas*

### 7.1.4. Creación del personaje principal

**Movimiento**

Tras crear una superficie experimental, se sitúa en la escena al personaje principal y se le añade un componente RigidBody2D y un BoxCollider2D. Estos servirán para dar físicas al personaje y detección de colisiones respectivamente y serán componentes referenciados en el script para el movimiento del personaje.

Este será el primer script creado y el más importante, ya que será el controlador del jugador. Se añadirá a la carpeta de scripts y se abrirá en Visual Studio Code. La clase Monobehaviour es una de las clases que forman el núcleo principal de las API de Unity. Todos los scripts vinculados a objetos (en el caso de Unity los llamaremos **GameObjects**) deben derivar de esta clase ya que nos expondrán diversos eventos y funciones que nos serán muy útiles a la hora de desarrollar nuestro proyecto.

Las funciones de MonoBehaviour más importantes y que más se usarán en distintos scripts serán **Start**, **Update** y **FixedUpdate**.

**Start** se ejecutará tan pronto se cree en la escena el objeto contenedor del script.

**Update** se ejecutará a cada frame, lo que quiere decir que, si el juego funciona a 30 FPS, la función Update se ejecutará 30 veces cada segundo. Pero como la frecuencia de llamada de la función no será siempre la misma ya que ejecución de nuestro juego no será siempre homogénea, dependerá de la carga de objetos, efectos, etc. Por todo esto la función Update se suele usar para el movimiento de objetos que no dependan de la física, actualizar datos en la **UI**, entrada de datos y temporizadores simples.

**FixedUpdate** en cambio se ejecuta a un intervalo fijo, tal y como su nombre indica, independientemente del **framerate** (cantidad de fotogramas por segundo) al que se esté ejecutando el juego, siendo su uso recomendable para todo aquello que esté relacionado con físicas.

Con estos conocimientos, se crea el script que controlará el movimiento. Básicamente, en la función Start (la que se ejecuta al crearse el objeto en la escena) se obtendrán las referencias a los componentes del personaje, que son el Animator (animaciones), el RigidBody (físicas) y la clase PlayerInput que servirá para traducir las entradas de los controles del jugador en información dentro del sistema.

Con estas referencias, se usan 3 métodos en FixedUpdate, ya que al tratarse de las físicas del personaje que el jugador manejará, lo conveniente es que estas se ejecuten de manera independiente al framerate. Esos métodos son:

**PhyscsCheck**: se usará para determinar si el jugador está en el suelo o en el aire.

**GroundMovement**: servirá para aplicar la velocidad horizontal al personaje y girarle cuando se cambie la dirección del movimiento.

**MidAirMovement**: controla el movimiento vertical del personaje, saltando si este está en el suelo y se pulsa el botón o limitando la velocidad de caída para evitar problemas con la cámara.

Cabe destacar el uso de **RaycastHit2D** para los comprobadores del suelo, los cuales serán dos (uno por cada pie en el eje x) y estarán en los bordes de la caja de colisiones. Previamente este comprobador era

sólo uno en el centro del eje x del personaje, pero esto hacía que el personaje cayera de las plataformas cuando estuviese con más de la mitad del cuerpo fuera.

Esto producía un efecto indeseable en el aspecto plataformas del juego, ya que la intención debe ser proporcionar al jugador saltos agradables y arriesgados, en los que sea posible saltar muy al borde para llegar más lejos. Con esta forma de comprobar si el personaje está en el suelo y con el "tiempo de Coyote" se consigue este deseado efecto.

El tiempo de Coyote es una ventana muy pequeña de tiempo durante la cual el jugador podrá saltar, aunque ya no esté en el suelo porque haya sobrepasado el borde de la plataforma por muy poco. Es una estrategia usada regularmente en juegos de plataformas para reducir un poco la dificultad y aumentar la satisfacción del jugador al conseguir realizar saltos muy al límite.

Para aplicar fuerza en el eje x o y al personaje cuando se toquen los controles indicados, hará falta usar la referencia al RigidBody del personaje. Los RigidBody permiten al GameObject contenedor actuar bajo el control de las físicas. Se usará en el personaje principal para que este pueda recibir fuerza al saltar o moverse y para que se vea afectado por la gravedad. Como característica especial, el Rigidbody del personaje principal, y de hecho de todos los enemigos y objetos, no permitirá que roten en el eje z. Esto es así para mantener la orientación de los sprites siempre de pie sobre el eje x, de manera que los objetos no puedan "caerse".

**Animaciones**

Al empezar la creación del script y las diferentes pruebas en el mismo para implementar el movimiento del personaje, este sólo consistía en una imagen. Una vez se acaba de implementar todo el movimiento, se añadirán las animaciones acordes con el mismo al Animator del personaje para gestionar su proceso. Por este motivo hace falta referenciar dicho componente en el script de movimiento, para dar valores a las variables que se usarán en el Animator y así determinar qué animación ejecutar. Estos valores serán estar en el suelo, la velocidad en el eje x y la velocidad en el eje y.

Existirán una animación para cuando Jaina esté parada, que se llamará IdleJaina y se dará cuando la velocidad en el eje x sea 0 y esté en el suelo. Otra animación RunnningJaina para cuando esté en movimiento (velocidad en x mayor o menor de 0, dependiendo de si el movimiento es hacia la derecha o hacia la izquierda) por el suelo. Un árbol de animaciones para la animación de salto, que será un **JumpBlendTree** y comprenderá tres fases: el salto de Jaina (velocidad en y mayor que 0), el momento en el que empieza a caer (velocidad en el eje y igual a 0) y la caída (velocidad en el eje y menor que 0). Finalmente, una animación de aterrizaje LandingJaina que servirá de transición entre el momento en que toca el suelo después de estar cayendo y volver a las animaciones de movimiento en el suelo.

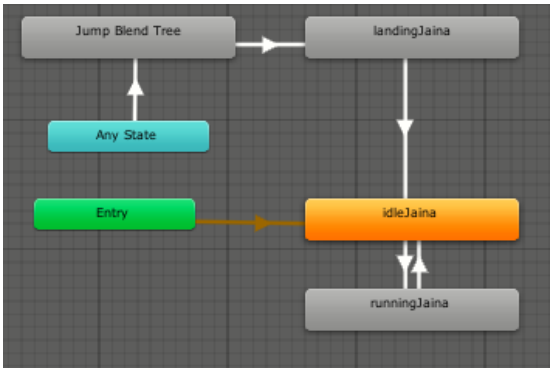



*Ilustración 12: Animator de Jaina*

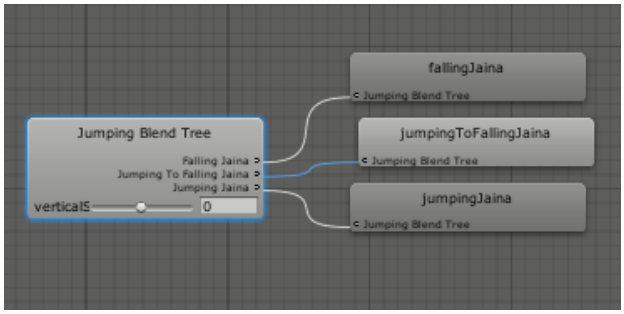



*Ilustración 11:Árbol de animaciones*

Finalmente, para poder interactuar con el personaje recién creado, hará falta usar unos controles y algo que traduzca la información dentro del juego. Unity incluye en sus librerías métodos para conseguir directamente la información de si un botón, ya sea de mando, teclado o táctil, se está presionando o no, por lo que podría incluirse directamente en el código. Sin embargo, al ser el videojuego multiplataforma, habría que repetir mucho código para cada tipo de dispositivo. Por tanto, la mejor práctica será crear un script llamado PlayerInput donde se recoja la información de cualquier controlador y se traduzca de la misma forma para ahorrar código en PlayerMovement y mantener abstracción.

Durante la implementación del movimiento del personaje principal y la creación de un entorno se realizarán pruebas de manera continua. Estas no sólo servirán para determinar si algo no funciona, sino que además permitirán ajustar los valores de salto y velocidad, el tamaño de la **hitbox** (caja de colisión), la posición de los comprobadores del entorno, etc.

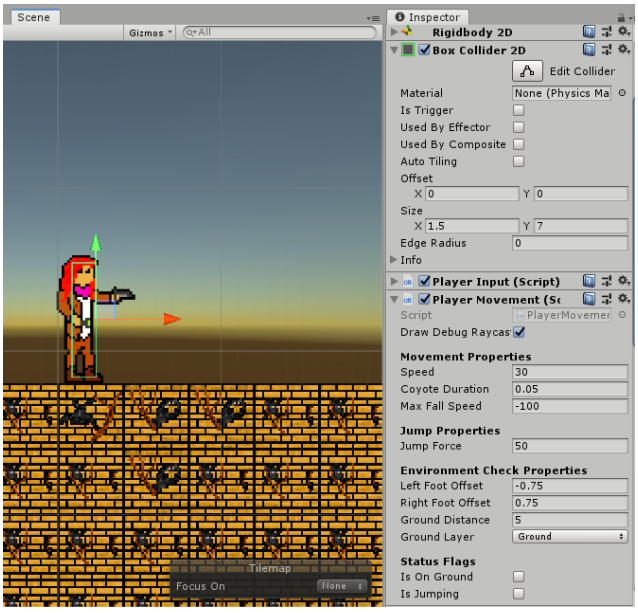



*Ilustración 13: Jaina y sus componentes*

### 7.1.5. Cámara

**Diseño**

La cámara seguirá al jugador, se moverá de manera fluida y tendrá una proyección ortográfica hacia el nivel.

**Implementación**

El script que controlará el movimiento de la cámara será CameraFollow. Este script calculará la posición del jugador cada **frame** y se moverá desde donde estaba hasta ese punto. Para suavizar el movimiento se aumenta un poco el tiempo que la cámara tarda en llegar a la posición del jugador.

Con esto podría darse la cámara por finalizada, sin embargo, existe un problema. Este tiene que ver con que el tamaño de la cámara depende de la resolución de la pantalla, lo cual afecta a la experiencia del jugador dado que percibirá el mundo de manera diferente dependiendo de la resolución del dispositivo, y esto no es para nada consistente. Una vista consistente en todos los sistemas significa que todos los jugadores tienen el mismo contexto de manera que todos los jugadores verán el mismo entorno de la manera en que el diseñador del juego lo ha pretendido.

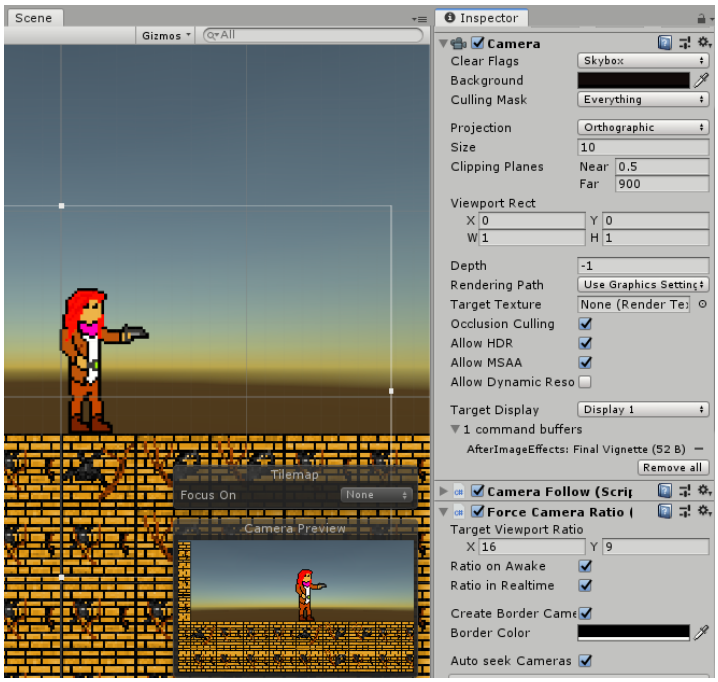


*Ilustración 14: Cámara y sus components*

Para lograr este propósito, hará falta descargar un componente de la **Asset Store**. Esto es la tienda de Unity, donde se pueden encontrar paquetes de características diferentes y con precios variados. El paquete requerido se llama **AutoLetterBox**: es completamente gratuito y consiste en tan sólo un script. Este script se añade a los componentes de la cámara principal, y su función será mantener la resolución de la cámara siempre igual. Para ello, crea otra cámara que escalará con la resolución, manteniendo la cámara principal con la misma resolución y creando consecuentemente bordes negros que asegurarán la consistencia del tamaño de la cámara, de ahí el nombre "Letter Box", que significa buzón porque parece que la cámara está mirando a través de este, por tanto, teniendo siempre la misma resolución de imagen.

Otra alternativa, y la más utilizada en videojuegos del mercado, consiste en permitir al jugador elegir una resolución para el videojuego, y adaptar el mismo a las distintas opciones. Sin embargo, por falta de tiempo se consideró esta opción como la mejor alternativa.

### 7.1.6. Sistema de ataque

**Diseño**

El jugador podrá disparar una pistola que el personaje lleva en la mano. La bala dañará a los enemigos y se destruirá al impactar. También dejará una estela de humo. Sólo se podrá disparar hacia delante horizontalmente, ya que tomaría mucho tiempo crear todas las animaciones para cada ángulo de disparo.

**Implementación**

Lo primero que será necesario para implementar el disparo del arma será crear un objeto en la escena al que llamaremos GunMuzzle, que en español significa cañón de la pistola. Este objeto se usará simplemente como una referencia de posición para que las balas se disparen siempre desde la pistola, para que así la bala no parezca que salga del cuerpo del jugador.

Este objeto además será hijo del personaje principal. Esto es así para que cuando el personaje se gire, el cañón se mantenga en la pistola.


*Ilustración 15: Jerarquía de la instancia de Jaina*

Para controlar el disparo, se usará un script que detecte cuando el jugador presiona el botón de disparar y cree una bala que salga del arma en la dirección que mira el jugador. Para que el jugador no pueda disparar continuamente, lo cual haría el videojuego mucho más fácil y menos interesante, aparte de parecer estar roto, se usará un **cooldown** o tiempo de enfriamiento. Esto consiste en un contador que bloquee el uso de una habilidad durante un tiempo después de haber usado esta. Esto dará una cadencia

de disparo que además ayudará a que el jugador crea que está disparando un arma de verdad, ya que hace más realista el disparo.

Se usará una referencia a PlayerInput para detectar que el jugador presiona el botón de disparo y otra a PlayerMovement para calcular la dirección de la bala en función de la dirección del personaje. Finalmente, se añadirá la referencia a la posición del cañón y al objeto bala que se disparará.

### 7.1.7. Balas

**Prefabs**

Se usará un objeto prefabricado para las balas. Un objeto prefabricado o **prefab** es un objeto cuyo uso se repite mucho. Por ejemplo, un enemigo, una trampa o una plataforma. Todos los objetos que hacen el nivel en el tilemap son de hecho prefabs que Unity gestiona con el sistema de tilemaps. Pero para crear un prefab propio, hará falta crear un objeto en la escena y tras configurarlo por completo, mover este objeto al árbol de directorios, en la carpeta de prefabs.

De esta manera, Unity automáticamente creará un archivo. prefab con el objeto en el estado que lo dejamos. Así, se podrá arrastrar el objeto desde la carpeta hasta la escena para usarlo, sin necesidad de estar copiando y pegando otros. Además, la configuración puede cambiarse una vez creado, y eso afectará a todos los prefabs de la escena. Si fueran copias, habría que reconfigurar todos los objetos uno a uno. Esta característica es también una de las que hacen a Unity popular, muy parecido a los blueprints de Unreal Engine.

**Diseño**

Se creará un objeto bala con su sprite correspondiente, una caja de colisión, para detectar cuando impacte y dos scripts: uno que detecte esta colisión, aplique el daño si golpea a un enemigo y se destruya y otro que mueva la bala de manera constante durante su existencia. También se usará otro script que destruya la bala después de un tiempo, para limpiar la escena de los objetos bala que haya creados y así no abusar de la memoria del ordenador. La razón para crear un script aparte y no incluir este simple código en el script de la bala es porque este comportamiento será necesario en otros objetos, así que se creará como componente único para ahorrar código.

*Ilustración 16: Jerarquía del proyectil*

**Implementación**

El objeto Projectile constará de un RigidBody 2D para moverse, el script de destrucción tras 2 segundos y el script que mantendrá en movimiento a la bala durante este tiempo. Este objeto además tendrá un hijo que se llamará Bullet, el cual tendrá la caja de colisión, el renderizado de la bala y un script que controle la colisión de la bala. De esta manera, podemos hacer que la bala se destruya sin destruir otras cosas, como la estela de humo que deja y el sonido del disparo. En este aspecto Unity resulta muy útil gracias a su sistema de herencia en la escena.

Finalmente, se añadirá un objeto más a la herencia de Projectile, llamado bulletSmoke y que será, como su nombre indica, humo dejado por la bala en su trayectoria. Este objeto será un **Particle System** (sistema de partículas). El editor de partículas de Unity permitirá diseñar el comportamiento de las partículas desde su creación hasta su destrucción. La idea es conseguir un efecto parecido al del humo que dejaría una pistola al disparar una bala.

Para ello, se diseña un Sprite y se crea un **material** con este. Un material es una definición acerca de cómo la superficie debería ser renderizada, incluyendo referencias a texturas utilizadas, tintes de color y más. Hará falta crear un material para poder utilizarlo con el Particle System. Una vez se tiene el material, se añade al renderer del Particle System para que las operaciones realizadas se rendericen con este material. Tras esto, se prueban valores hasta conseguir el efecto deseado.

En resumen, se crearán en zonas aleatorias por detrás de la bala emisiones de este material de humo durante una duración corta. Estas se moverán en direcciones aleatorias y disminuirán en tamaño hasta desaparecer.

Con esto se finalizará la bala, quedando únicamente por añadir el sonido del disparo para que se active siempre que se cree una bala y así ahorrar código.

El script de la bala hará que esta se destruya al chocar con un enemigo, el nivel o las decoraciones. Para detectar esto, se usará una característica de Unity que permite agrupar los objetos bajo etiquetas y así poder filtrar los objetos sin tener que referenciarlos.

Una vez finalizado el sistema de disparo, se harán las pruebas correspondientes para comprobar que todo funciona y ajustar la velocidad de la bala, el tiempo de enfriamiento, el efecto de las partículas, etc.

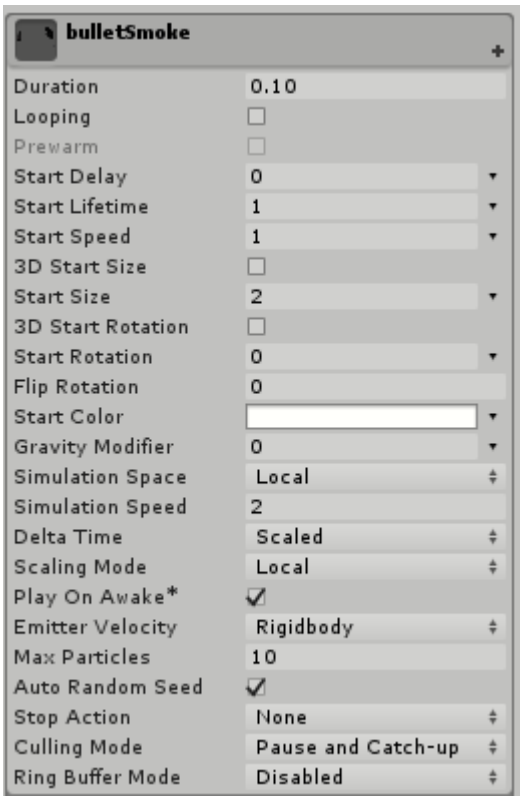



*Ilustración 17: Efecto de partículas de humo*

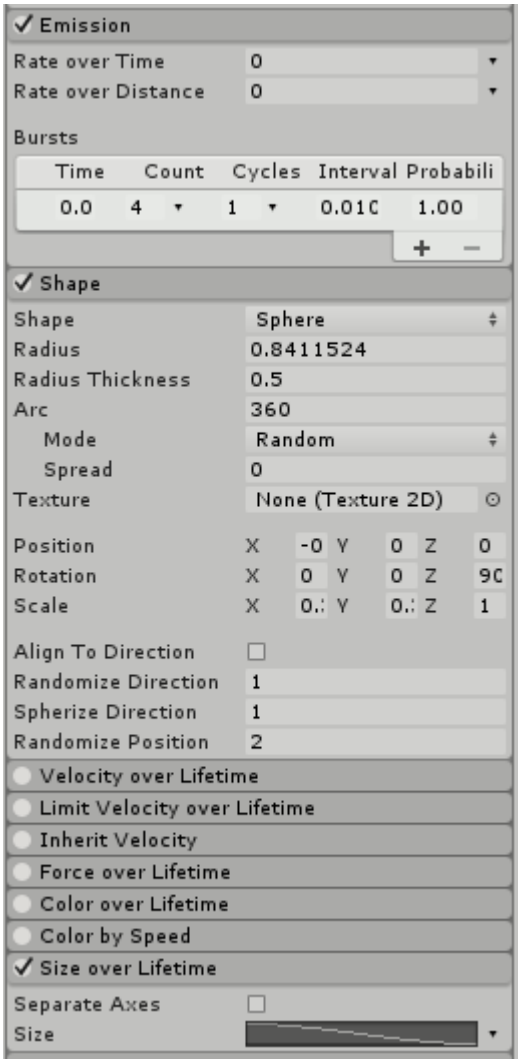



*Ilustración 18: Efecto de partículas de humo*

## 7.2. Sprint 2: Enemigos, sistema de salud y elementos interactivos

### 7.2.1. Enemigos

**Diseño**

Ahora que el personaje puede disparar, se crearán enemigos que justifiquen el uso de esta habilidad. Estos enemigos consistirán en un murciélago que dispara gotas de sangre al jugador y una araña que se mueve en vertical y ataca al jugador.

**Murciélago**

**Diseño**

Habrá un enemigo murciélago que se mantenga suspendido en el aire moviendo las alas y ataque al jugador disparándole una gota de sangre. El patrón de ataque debe ser reconocible, pero al mismo tiempo añadir un factor de aleatoriedad para hacer cada partida diferente. Cuando tenga poca vida, el ataque debe será más fuerte.

**Implementación**

Para la creación del murciélago será necesario un script que controle la IA, un Animator, una caja de colisiones para detectar cuando recibe un disparo o cuando choca con el jugador y otro script que administre la vida del murciélago en función de estas colisiones, el cual se explicará tras acabar los enemigos.

Para el Animator, se creará un estado BatFlying (Murciélago Volando) en el cual el murciélago moverá las alas estático y otro BatFlyingFast en el que moverá las alas más rápido, el cual se activará cuando el murciélago esté herido y por tanto en su estado de ira.

Este estado de ira está definido en el script de la IA del murciélago. Se activa cuando el murciélago sólo tiene una vida restante y hace que el murciélago empiece a disparar 3 proyectiles en un patrón de abanico, como se puede apreciar en la imagen.

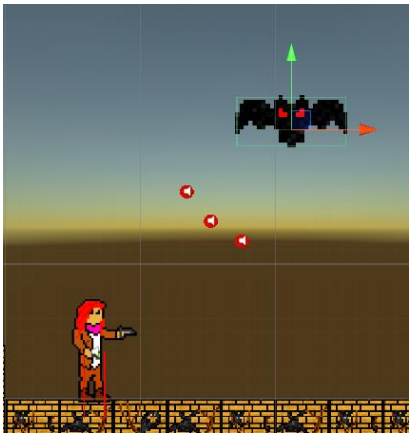


*Ilustración 19: Murciélago atacando*

El ataque normal disparará aleatoriamente un proyectil en una de las 3 direcciones del ataque en estado de ira. Sólo disparará si el jugador se encuentra en un rango estimado y ha pasado un tiempo dado desde el último disparo.

Al igual que en PlayerController, el ataque del jugador, cuando el murciélago dispare instanciará un proyectil que será un prefab. Este proyectil se llama BloodDrop (Gota de Sangre) y como el proyectil del jugador tendrá un RigidBody para el movimiento, un collider para gestionar su colisión y un script que administre la colisión y otro para su destrucción tras un tiempo. Estos scripts serán los mismos que se usaron en el proyectil del jugador, ya que se reutilizarán. Para adaptar la detección de colisiones,

simplemente se añadirá una variable que detecte si el proyectil es enemigo o del jugador y una condición que realice daño al jugador si un proyectil enemigo le toca.

Además, se le añadirá un AudioSource con el sonido del disparo para que se active en cuanto se instancie el proyectil.

Finalmente se prueba el murciélago para ajustar sus valores hasta los deseados.

**Araña**

**Diseño**

El enemigo araña se moverá en vertical desde el techo hasta el suelo y viceversa. Cuando esté en estado de ira, atacará al jugador en el descenso moviéndose hacia él a más velocidad de la normal, tras lo cual volverá a ascender recto. Esta mecánica no debe resultar muy hostigadora para el jugador.

**Implementación**

Para la implementación de la araña hará falta reutilizar el RaycastHit2D usado en PlayerMovement. Esta vez, serán usados para colocar controladores del entorno en cada esquina y cada lado de la araña. De esta manera, se podrá cambiar la dirección de la araña cuando toque el suelo o techo, y se evitará que atraviese paredes.

Para el estado de ira, que se activará cuando la araña tenga 1 vida restante, se añadirá un movimiento horizontal en la dirección del jugador mientras la araña esté descendiendo. Para evitar que sea muy hostigadora, se suaviza el movimiento hacia los lados usando un umbral.

Para la animación, se repetirá el mismo patrón que con el murciélago, creando una transición a una animación de movimiento más rápida cuando la araña esté en estado de ira.

Finalmente, un RigidBody para el movimiento de la araña, una caja de colisiones y un script que administre la salud de la araña en función de estas colisiones.

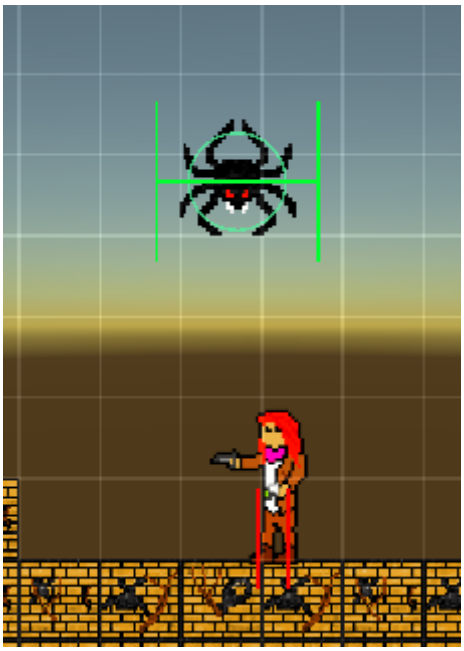


*Ilustración 20: Araña atacando al jugador*

### 7.2.2. Sistema de salud

**Diseño**

Tanto el personaje principal como los enemigos deberán tener un sistema de salud que administre la cantidad de golpes que pueden recibir antes de desaparecer. La salud funcionará diferente entre el personaje principal y el resto de los enemigos, por lo que se realizarán dos scripts.

**Salud Enemiga**

**Diseño**

Para los enemigos, el script destruirá la instancia del enemigo en la escena cuando este se quede sin vida, bien por chocar con el personaje principal o bien por haber recibido suficientes disparos. Además, al morir otorgarán al jugador una determinada cantidad de puntos.

**Implementación**

Para la implementación del sistema de salud enemigo, hará falta crear un script llamado EnemyHealth. En este script se realizarán dos funciones: dañar al enemigo y eliminar al enemigo. Dañar al enemigo consistirá en reducir la vida actual del mismo en determinados eventos. Estos serán cuando una bala golpee al enemigo o cuando el enemigo colisione con el jugador. En el caso de la bala, será el script del proyectil el que referencie a este script y utilice la función de daño para reducir la salud del enemigo. En el caso de

colisionar, será este script el que, mediante una referencia a la caja de colisiones del enemigo, infrinja una cantidad de daño igual a la salud máxima del enemigo (para asegurar que se destruye al enemigo) y también dañe en un punto de vida al jugador. Para ambos casos, se reproducirá un clip de audio y un efecto de partículas de sangrado.

Para crear este efecto, se utilizará de nuevo un ParticleSystem. Como material se usará un Sprite que consiste en una bola roja que representará una gota de sangre. Se modificarán los valores hasta obtener un pulso aleatorio de muchas gotas de sangre que se muevan en diferentes direcciones y caigan al suelo.

**Salud del Jugador**

**Diseño**

Si el jugador perdiera al ser tocado por un enemigo, el juego sería demasiado difícil. Por eso podrá recibir una cantidad de golpes antes de perder, además de que existirán objetos que recuperen su salud. En caso de que el jugador perdiera, bien por ataques de enemigos o por golpear alguna trampa, se acabará la partida y se llevará al jugador al menú de 'Game Over', donde podrá volver al menú o empezar otra partida. Para añadir dificultad al juego, la vida del jugador se mantiene a lo largo de los niveles del juego y sólo se reinicia al volver a empezar una nueva partida.

**Implementación**

Lo primero de todo será crear un script que lleve la cuenta de la vida de Jaina en cada momento y gestione los eventos que le provoquen daño o sanen su salud. Se crearán una variable para la vida máxima y para la vida actual. La vida actual se almacenará globalmente, de manera que al cambiar de escena (superando el nivel o perdiendo, por ejemplo) no se pierda este valor, ya que todos los objetos de la escena se destruirán y se iniciarán otros. Por tanto, en la función Start habrá una referencia a esta variable para inicializar la vida de Jaina actual. Para almacenar la variable se utilizará la clase PlayerPrefs del motor de Unity, la cual sirve exactamente para este tipo de acciones.

En cuanto a la función para gestionar el daño de Jaina, esta funcionará igual que la implementada en EnemyHealth, restando la cantidad de daño a la vida total, instanciando las mismas partículas de sangre que previamente se crearon para los enemigos y comprobando si esta es igual o inferior a cero para ejecutar la función de muerte. Esta función, por el contrario, será completamente diferente a la de los enemigos, ya que en vez de destruir a Jaina lo que hará será desactivarla en la escena (para que desaparezca) y cargar el menú de Game Over (derrota). Esto es así porque destruir al personaje principal en la escena rompería todas las referencias al mismo y por tanto el juego.

Esta función se ejecutará cuando Jaina impacte con un enemigo o un proyectil enemigo, lo que la hará perder la salud correspondiente, o cuando caiga en los pinchos, lo cual la dañará por un valor igual a su salud matándola directamente.

Finalmente, la función para recuperar la salud de Jaina será igual que la de dañarla, pero en vez de restar vida, se sumará y en lugar de comprobar si la vida está por debajo de cero, se comprobará que no sea igual

o superior a la vida máxima para que no se pueda tener más salud de la establecida. Esta función se ejecutará cuando Jaina pase por encima de una vasija de salud, de las cuales se hablará más adelante.

### 7.2.3. Trampas

**Diseño**

Para añadir interés al aspecto del **plataformeo**, se añadirán a lo largo de cada nivel fosas de pinchos y sierras giratorias que provocarán la muerte del personaje principal en el acto. Estas se usarán solo cuando sea necesario, para evitar añadir demasiada dificultad al juego.

**Implementación**

Para la implementación de las trampas simplemente se necesitará un Sprite y un script en el caso de la sierra giratoria (para hacer que gire). Los pinchos se añadirán como un tile para hacer mucho más sencillo el trabajo de colocación. De esta manera, simplemente se podrán 'pintar' todos los pinchos que hagan falta en la cuadricula del nivel. Además, existirán en un tilemap aparte de los del nivel, de manera que se pueda referenciar a todos los pinchos simplemente con referenciar este tilemap, simplificando así mucho las operaciones que se realicen con ellos y ahorrará la creación de varias cajas de colisiones, lo que significará un mejor rendimiento. La sierra giratoria será un objeto prefabricado (prefab) que utilizará un script para rotar en el eje z simulando el giro de una sierra. Para ello se utilizará la función Rotate del motor de Unity. Cada sierra tendrá una caja de colisiones

Para que estas trampas dañen a Jaina, se añadirá la etiqueta 'Trap' al proyecto y se seleccionará para el Tilemap de pinchos y para el prefab de la sierra. De esta manera las trampas estarán siempre identificadas, y en la gestión de colisiones del jugador en PlayerController simplemente habrá que ejecutar la función de dañar a Jaina por el total de su vida cuando el objeto con el que colisione tenga una etiqueta de trampa.

### 7.2.4. Tesoros

**Diseño**

Las recompensas serán de dos tipos: vida en vasijas y puntos en tesoros. La recompensa de vida consistirá en una vasija con un Ankh (cruz con la parte superior en forma de óvalo) y sólo se podrán encontrar en zonas peligrosas con enemigos. Estas sanarán 1 punto de vida a Jaina, de manera que el jugador podrá decidir dependiendo de su situación si merece la pena arriesgarse para intentar remontar una partida en la que haya perdido mucha salud.

En cuanto a los tesoros, estas consistirán en diferentes objetos que otorgarán al jugador distintas cantidades de puntos en función de su rareza. Estarán repartidos por todo el mapa y serán más difíciles de conseguir en tanto que más puntos otorguen.

**Implementación**

La implementación de estos objetos es bastante ya que consistirán en un Sprite con una caja de colisiones y un script.

Al igual que con la salud actual del personaje, los puntos se almacenarán en una variable global para que no desaparezcan al cargar otra escena y se reiniciarán al principio de cada partida. Se implementará una función en el script GameMaster que añada una cantidad determinada de puntos. Su ejecución se dará dentro de la gestión de colisiones de Jaina, donde además se destruirá el objeto y se instanciarán unas partículas muy similares al efecto de sangrado, pero esta vez doradas y con un comportamiento levemente diferente: se esparcen menos y caen al suelo con más velocidad. Para identificar estos objetos en la colisión, se etiquetarán con la palabra 'Treasure'. La vasija que dará vida tendrá la etiqueta 'Health' y al colisionar con una se realizarán las mismas acciones que con otras recompensas, pero con la diferencia de que la función ejecutada esta vez será la implementada anteriormente en PlayerHealth que servía para sumar vida al jugador.

Se utilizará el mismo script, TreasureController, para todas las recompensas. Este tendrá dos funciones: indicar la cantidad de vida o de puntos que se darán al jugador cuando recoja estos objetos y otorgar un movimiento que haga que los objetos parezcan estar flotando. Para lo primero bastará con una variable pública cuyo valor modificaremos en el Inspector, mientras que para lo segundo usaremos una función senoidal que varíe la posición del objeto dentro de una amplitud vertical a lo largo del tiempo.

## 7.3. Sprint 3: Interfaz de usuario, sistema de menús y efectos de sonido

### 7.3.1. Interfaz de usuario

**Diseño**

Se creará una interfaz de usuario limpia y clara en la que se proporcione al jugador la información específica que necesita. Esto será: su salud actual, el tiempo de juego y su puntuación total. La salud estará situada en la esquina superior izquierda de la pantalla y consistirá en una fila de corazones en la que cada corazón representa una vida. Se ha optado por este diseño antes que por, por ejemplo, una barra de vida porque al ser un patrón claro y reconocible en los videojuegos arcade, el usuario relacionará este diseño con los anteriores y entenderá que cada corazón es una vida. Por el contrario, con una barra de vida el jugador no podría estar seguro en cada momento de cuanta vida tiene exactamente, pudiendo esto empeorar su experiencia ya que dificulta la toma de decisiones al no saber la cantidad de peligro exacta, si no aproximada, que corre.

Para el tiempo, se ha elegido el centro y para la puntuación la esquina superior derecha. La distribución es muy reconocible ya que es parecida a la de juegos clásicos como "Metroid", "The Legend of Zelda" o "Mario Bros". El contador del tiempo mostrará la cantidad de tiempo que el jugador lleva en un determinado nivel y cuando este muera o llegue al final, se sumarán los tiempos para que el jugador sepa en total lo que ha tardado en terminar su partida. El contador de puntos se mantendrá a lo largo de los niveles y se mostrará también al acabar la partida.

Finalmente, se requerirá de algún elemento que notifique al jugador cuando este reciba daño, el cual añadirá detallismo al producto y servirá para ayudar al jugador a reconocer los eventos del juego y sus consecuencias.

**Implementación**

Para crear la interfaz de usuario hará falta crear un nuevo objeto en la escena: un Canvas. Este objeto es el área donde estarán todos los elementos UI. Se configurará este objeto para que en vez de ocupar la totalidad de la pantalla ocupe la misma área que la cámara. De esta manera, cuando la cámara haga uso del AutoLetterBox, la interfaz no se saldrá a los bordes negros. También se configurará para que los elementos de la UI escalen con el tamaño de la pantalla y se establecerá una resolución de referencia de 1600 x 900 (16:9, que es el tamaño de la cámara).

Dentro del Canvas, se crearán 3 objetos diferentes: un objeto para gestionar el sistema de vida y dos objetos donde insertar texto, uno para la puntuación y otro para el tiempo. Se colocará cada elemento en su lugar correspondiente y se procederá a la creación del script que inyecte los valores en estos elementos.

Empezando por la salud, se crearán en el objeto playerHeartUI, situado dentro del Canvas, cinco imágenes de corazones colocados en fila. Después, en el script PlayerHealth se crearán una variable array de Image (imágenes) donde se añadirán los corazones de la interfaz y una función que será ejecutada al empezar un nivel y cada vez que haya una interacción con la vida de Jaina. Esta función recorrerá el array de corazones y activará o desactivará las imágenes de derecha a izquierda, representando así la pérdida o recuperación de salud.

En cuanto al tiempo y la puntuación, ambas variables serán creadas en el script GameMaster y se añadirán como PlayerPrefs de la misma forma que la vida del jugador anteriormente y al empezar una nueva partida, se reiniciarán sus valores. Para modificar el texto de los componentes PlayerPointsUI (puntuación) y GameTimeUI (tiempo) creados previamente dentro del Canvas, se añadirán estos al script GameMaster para crear la referencia. Para el tiempo, se actualizará el valor a cada segundo. Se obtendrá el tiempo total del juego y se formateará para mostrarlo como se ve en la imagen. En cuanto a la puntuación, se iniciará a 0 y se actualizará cada vez que el jugador consiga ejecutar la función **addPoints**, situada en este script pensando precisamente en esto.

Finalmente, se añadirá un borde de sangre que cubra la totalidad de la interfaz. Este elemento permanecerá desactivado hasta que se dañe al jugador. Entonces, al activarse aparecerá la imagen de la sangre y se la hará desparecer lentamente, todo esto realizado dentro del script PlayerHealth. Este efecto servirá para causar tensión y peligro en el jugador.

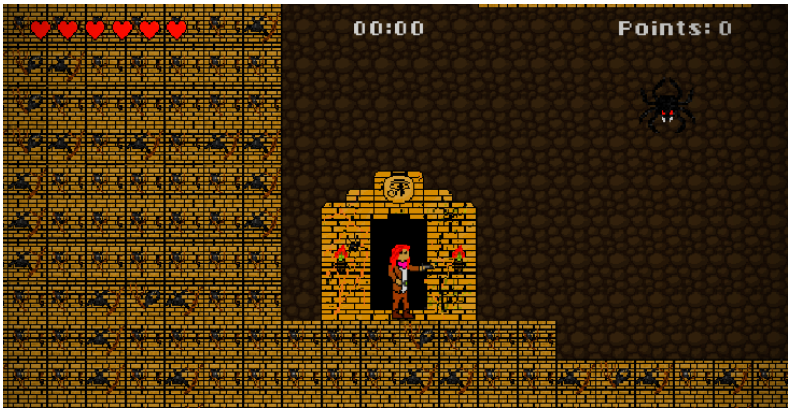


*Ilustración 21: Interfaz de Usuario*

### 7.3.2. Sistema de menús

**Diseño**

Para completar la experiencia y otorgar al jugador control sobre el juego, se creará un sistema de menús que incluirá: un menú inicial que servirá como entrada del juego, un menú donde aparezcan los controles para que sirva de guía al jugador, un menú de pausa para que el jugador pueda dejar la partida en cualquier momento si lo necesita y volver a ella más tarde, un menú de Game Over donde el jugador irá cuando pierda en el juego y finalmente un menú de victoria al que llegará cuando consiga superar el videojuego. El mapa de navegación entre escenas será el siguiente:

**Menú principal**

El menú principal permitirá al jugador empezar la partida, ver los controles o salir de la aplicación. Consistirá en el título del videojuego, un botón para cada una de las acciones mencionadas anteriormente y la imagen de la puerta utilizada en los niveles como punto de inicio y meta, que en esta escena invita al jugador a adentrarse en la pirámide.

Para crear todos los elementos del menú, hará falta un objeto Canvas al igual que para la IU. En este, se pondrán las imágenes y textos y se creará un objeto donde se colocarán los botones. Para establecer las acciones que realizará cada botón, hará falta crear un objeto **GameMaster** (no confundir con el script) al que se le añadirá el script **SceneController**. Este script contendrá las funciones que servirán para cambiar de escena. El siguiente requisito será añadir este objeto al botón, lo que dará acceso al uso de estas funciones, por lo que el último paso será seleccionar la función para cada botón.

La fuente elegida para los botones y más genéricamente para todos los textos del videojuego es *8-bit madness*, de Tyler Dunn (www.dafont.com)


*Ilustración 22: Menú principal*

**Menú de Opciones**

Este menú se creará de la misma manera que el menú de inicio. En esta escena, el jugador podrá ver los controles del videojuego y directamente empezar una nueva partida o volver al menú. Las acciones de los botones se implementarán de la misma manera que en el menú de inicio.


*Ilustración 23: Menú de opciones*

**Menú de Pausa**

Este menú sólo será accesible desde los niveles jugables. Cuando el jugador pulse el botón designado para la pausa, que por defecto será escape, se parará el tiempo en la escena y aparecerá un menú con 3 botones: un botón con el que volver a la escena en el mismo estado en el que se había pausado, otro para volver al menú y finalmente otro para salir directamente del juego. También se podrá restaurar el flujo normal del juego pulsando el botón de pausa de nuevo.

Para la gestión del sistema de pausa se creará un script específico que active y desactive la interfaz de usuario con los botones, al mismo tiempo que cambie la variable **timeScale**, perteneciente a la clase "Time" de Unity, de manera que su valor sea 0 cuando el menú esté activo (el tiempo no avanza) o 1 cuando se vuelva al juego (el tiempo transcurre de forma normal).

Esta IU pertenecerá a la misma IU de Jaina donde se encuentran la vida, tiempo y puntuación, pero estará separada para que no interfieran entre ellas.


*Ilustración 24: Menú de pausa*

**Menú de victoria y derrota**

Tanto el menú de victoria como el de derrota no serán menús regulares, si no que en vez de eso serán niveles de cámara fija jugables. En el caso del menú de derrota, consistirá en una habitación con pinchos y una puerta. Aparecerán tanto la puntuación como el tiempo total jugado en la IU y la pantalla estará recubierta por el mismo efecto de sangre que aparece cuando dañan al jugador. Este tendrá que decidir entre saltar a los pinchos para volver al menú o entrar por la puerta para empezar de nuevo.


*Ilustración 25: Menú de derrota*

Se ha tomado esta decisión porque se pretende que el ritmo del videojuego sea rápido, y un menú con un botón donde pulsar volver a empezar puede sacar al jugador de la experiencia y darle una dificultad por la que abandonar el producto. De esta forma, volver a empezar una partida consistirá únicamente en mantener pulsado un botón para entrar automáticamente por la puerta. Esto hace que el jugador se sumerja más en las mecánicas y el flujo del videojuego.

El menú de victoria será igual, pero simulando la salida de la pirámide hacia un desierto. Habrá una puerta que servirá para volver a empezar de nuevo y un desierto abierto el cual llevará al jugador al menú principal al adentrarse en este.


*Ilustración 26: Menú de victoria*

### 7.3.3. Efectos de Sonido

Toda la música y efectos de sonido han sido creados en **chiptune** con la finalidad de dar una atmósfera de juego old-school. Se ha utilizado **OpenMPT** para la música y **Bfxr** para los efectos de sonido.

**Música**

En el menú principal, el jugador empezará escuchando un ritmo chiptune muy marcado y animado que incita a jugar. Este ritmo será también usado en el menú de Victoria, lo que, tras la música de los niveles que se explicará a continuación, dará una sensación de liberación y de final del viaje.

Cuando empieza a jugar, este ritmo sigue sonando, dando una sensación de continuación, pero se mezcla con una melodía decreciente en escala arábica que empieza a sonar al poco tiempo. Esta crea una sensación de incomodidad, exotismo y peligro. Además, es bastante pegadiza, lo cual es un punto muy a favor en los videojuegos, ya que hace que el producto sea más icónico al extender su melodía. Un ejemplo de esto se puede ver en la música de Super Mario, la cual pocas personas no serían capaces de reconocer. Esta música sonará a lo largo de todos los niveles jugables.

Finalmente, unos acordes agresivos y una melodía en descenso para el menú de Game Over, que invitan a abandonar ese sitio lo antes posible. Tener una música incómoda para el menú de derrota incita al jugador a querer perder menos para evitar volver ahí. Cabe anotar que el hecho de que la música sea incómoda no afecta ni a su calidad ni a que suene bien o mal. La música usada en películas de miedo está

hecha para resultar incómoda al espectador y para lograrlo se necesita crear una pieza de gran calidad para que sea capaz de transmitir este sentimiento.

**Efectos**

Los efectos creados pretenden simular los sonidos de los eventos que ocurren en el videojuego. Por ejemplo, cuando el personaje salta se reproduce un sonido parecido al que todo el mundo ha escuchado en juegos de plataformas clásicos como “Mario”, “Megaman” o “Metroid”. También un sonido para el disparo, otro para cuando cada enemigo reciba daño, intentando simular lo que en otros videojuegos se ha usado como sonidos para una araña o en el caso del murciélago un sonido agudo parecido a su gruñido real. También un sonido para cuando el murciélago dispare

Finalmente, un sonido para cuando el personaje principal reciba daño, otro para cuando rompa algún tesoro, que suena como romper algo de porcelana, y por último un sonido misterioso para cuando llega al final de un nivel.

A continuación, una imagen con todos los efectos de sonido y canciones dentro del proyecto:

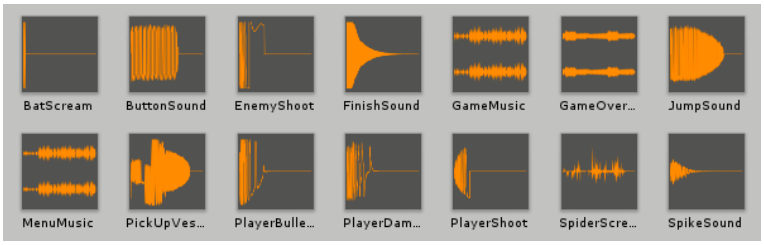



*Ilustración 27: Efectos de sonido y música*

## 7.4. Sprint 4: Creación de niveles, pruebas finales y cierre del proyecto

**Diseño**

La creación de niveles consistirá en primera instancia en la creación de varios Tilemaps situados en diferentes capas de profundidad. El cuerpo del nivel, es decir el suelo, techos y paredes, estarán en el nivel más externo, llamado Frontground. Seguidamente, estarán todos los objetos por defecto, como enemigos, jugador, etc. en la capa Default. Después, una capa donde colocar las trampas llamada Interactables y luego una capa para las decoraciones: telarañas, columnas, antorchas y cadenas, que será Decorations. Finalmente, una capa para el fondo de piedras llamada Background.

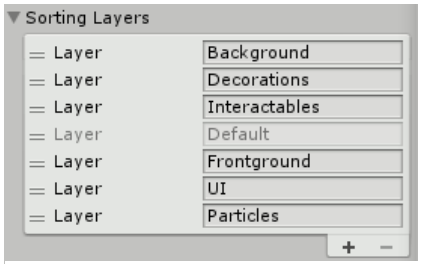



*Ilustración 28: Capas de orden*

Para elaborar el Frontground se utilizará un pincel que elige un elemento aleatorio de un array que en este caso estará compuesto por los 4 modelos diferentes de bloques de ladrillos.

Al mismo tiempo y de manera retroactiva se añadirán elementos al nivel como enemigos, tesoros y vidas extra.

### 7.4.1. Nivel 1 - Tutorial

Este nivel será el primero que los jugadores tendrán que afrontar, lo cual lo hace un nivel esencial porque será donde estos prueben por primera vez los controles, y el objetivo será hacer el proceso de aprendizaje lo más ligero y sencillo posible de manera que puedan sumergirse pronto en el juego.

Por este motivo, la cantidad de enemigos en este nivel se mantendrá reducida, habiendo tan solo 4 arañas y dos murciélagos, además de ser la mayoría de los encuentros con estos enemigos opcional. Esto se debe a que, al principio del nivel, se darán dos caminos opcionales al jugador, uno por abajo más fácil y otro al que tendrán que llegar saltando y que es más difícil, pero tiene más recompensas. Ambos pueden cogerse en una misma partida de manera que le jugador pueda explorar ambos caminos sin tener que empezar una nueva partida. Esta es una de las formas de hacer un videojuego entretenido tanto para el jugador más veterano como para el más casual.

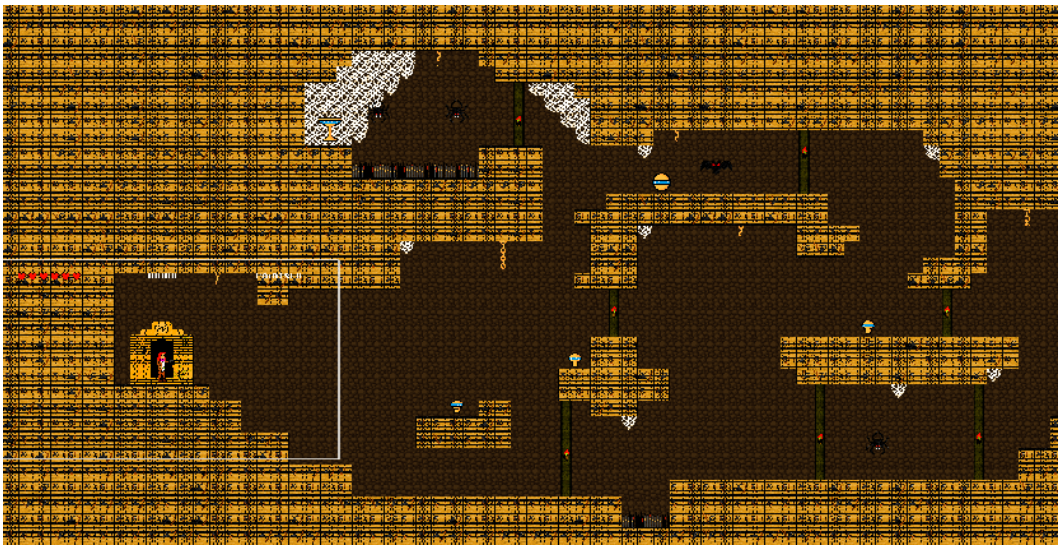


*Ilustración 29: Nivel 1, primera parte*

La segunda mitad del nivel consistirá en varias plataformas sobre fosas de pinchos donde los jugadores tendrán que saltar continuamente para avanzar y afrontar un murciélago y una araña que habrá por el camino. Esta parte se centra en obligar al jugador saltar y atacar saltando, dado que son las mecánicas clave para conseguir superar el videojuego. Al final del nivel, al jugador le esperan una puerta que sugiere un avance de nivel, y lo es, y una vasija que le dará un punto de salud si hubiera perdido alguno a lo largo del nivel.

Se colocarán tesoros de pocos puntos por todo el recorrido para orientar al jugador hacia la meta. Se pondrán tesoros de mucha más puntuación escondidos en lugares más peligrosos, para que el jugador tenga la sensación de esforzarse por conseguir algo mejor y se sienta recompensado por explorar.

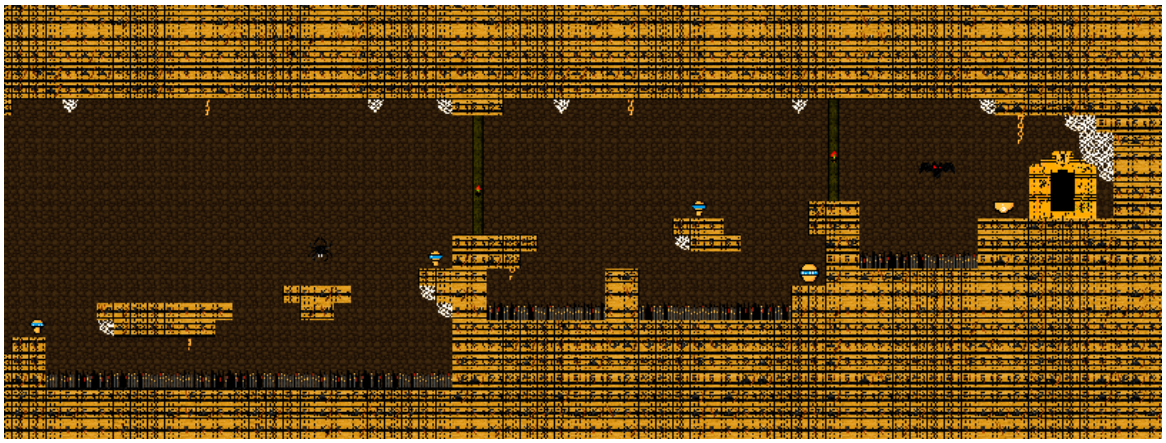


*Ilustración 30: Nivel 1, segunda parte*

### 7.4.2. Nivel 2 – RetroChallenge

Este nivel será la piedra angular del videojuego, ya que será el más largo y el que ofrezca un mayor reto. En este nivel el jugador tendrá que poner a prueba lo aprendido en el tutorial. Habrá varias fosas con pinchos, una cantidad de enemigos muchísimo mayor que la del tutorial (17 murciélagos y 34 arañas) y además se aprovecharán mucho mejor las habilidades de estos enemigos. Por ejemplo, habrá arañas en las subidas de nivel para hacer más efectivo su ataque hacia abajo y en zonas en las que el jugador necesitará enfrentarse a ellas para poder avanzar. Se pondrán también cerca de murciélagos y otras arañas para que, si el jugador falla sus ataques al objetivo y les da a estas, entren en modo de ataque y se conviertan en una nueva amenaza para el jugador que penalizará la falta de habilidad disparando. Esto hará que el jugador prefiera asegurar mejor sus disparos y que mejore su habilidad para evitar este problema.

Por otro lado, los murciélagos estarán cerca de caídas a pinchos, ya que sus disparos normalmente obligarán al jugador a saltar por lo que estar cerca de pinchos es la manera de convertir un movimiento de evasión en otro peligro para el jugador, que tendrá que manejar bien al personaje para mantener el control durante el salto y no caer.

Cada enemigo ha sido posicionado con un sentido dentro del diseño del nivel. No hay enemigo que rellene espacio, si no que cada enemigo utiliza a otros enemigos o al mismo entorno para ponérselo difícil al jugador, quien tendrá que hacer lo mismo, aprovechar el entorno, para superarles. Por ello habrá plataformas en las que el jugador tenga un buen disparo a los enemigos y zonas donde poder refugiarse.

Las partes más importantes del nivel consisten en una zona en la que el jugador puede elegir entre dos caminos, siendo el de arriba de nuevo el más difícil (cosa que el jugador probablemente entienda dado que se le ha enseñado eso en el tutorial). La principal razón de hacer que el camino de arriba sea el difícil es que el jugador puede caer abajo, lo que le haría no solo tener que superar ese camino si no que tendrían

que volver a subir. Al final del camino difícil hay una vasija que le recompensará con un punto de vida por su valentía.

*Ilustración 31: Nivel 2, bifurcación*

Después habrá una zona secreta llena de arañas donde habrá otra vasija y en la que el jugador deberá tener cuidado para no enfadar a demasiadas arañas al mismo tiempo. A continuación, la parte más difícil del nivel junto con el final: una subida en la que un murciélago al que no se puede alcanzar dispara al jugador mientras este debe sortear o acabar con las arañas que hay en la subida para poder avanzar. Tras conseguir subir, el jugador estará en vísperas de la parte final o podrá ir a una zona a la izquierda donde conseguirá puntos y podrá vengarse del murciélago que no le dejaba subir.

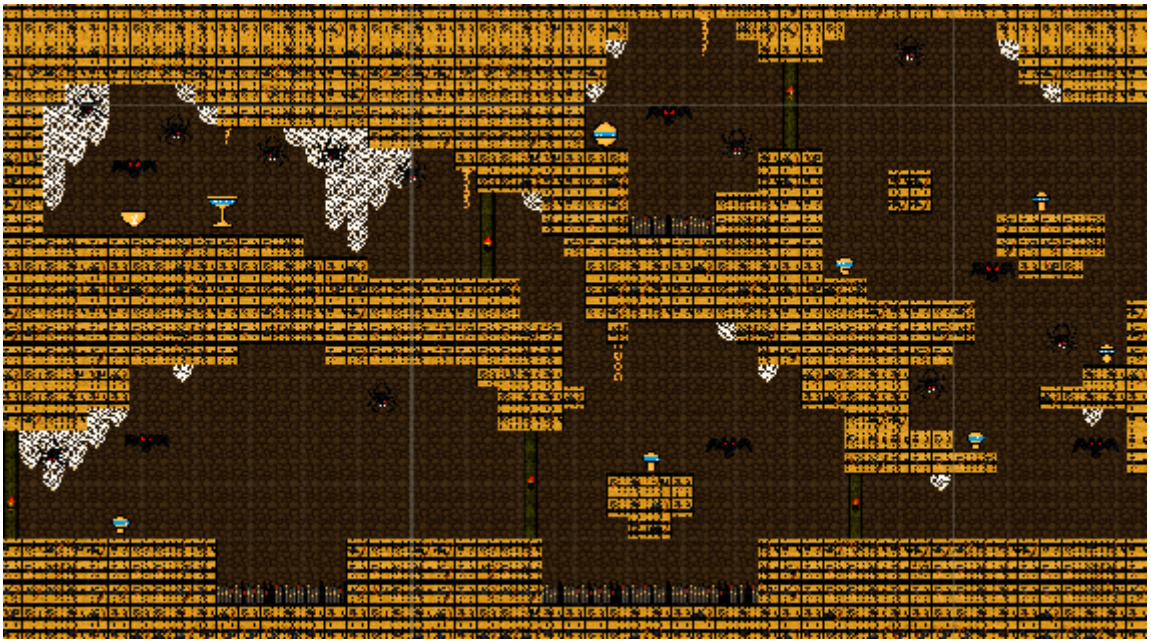


*Ilustración 32: Nivel 2, zona secreta y ascenso*

La parte del final es una prueba de habilidad en la que el jugador demostrará haber dominado el salto y el disparo por completo, teniendo que usarlos de manera combinada para no caer en la fosa al mismo tiempo que elimina a una gran cantidad de enemigos. En una parte de esta fosa se ha dejado abierto un agujero hacia una zona con una vida extra, para recompensar por la exploración al jugador atento que lo descubra.

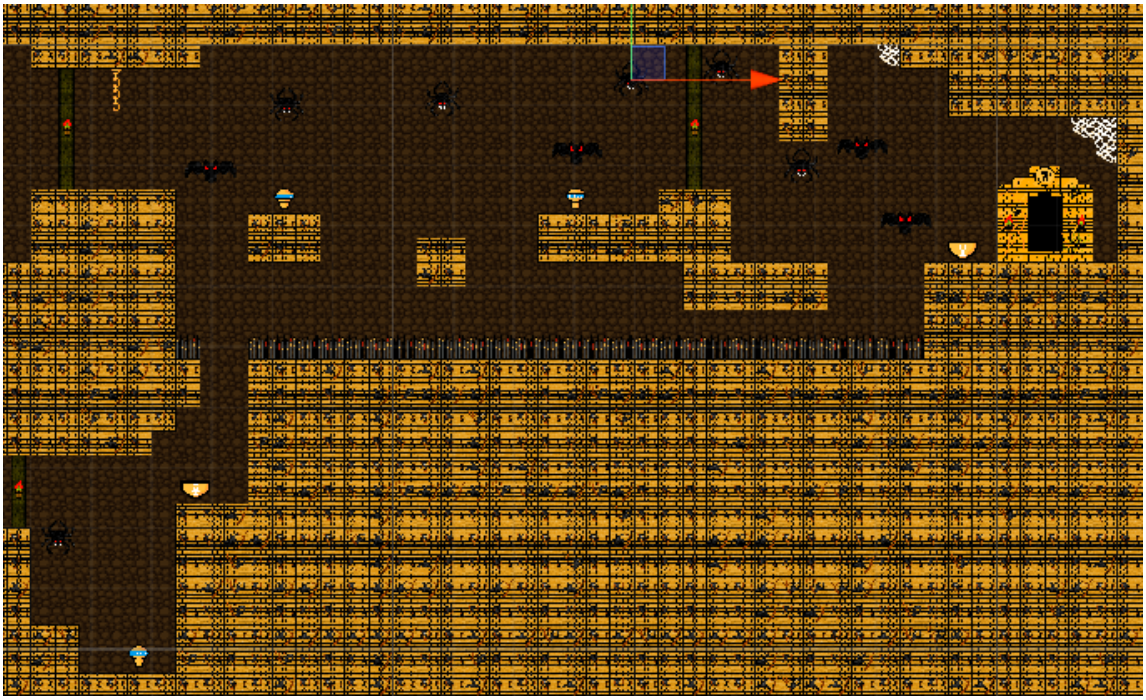


*Ilustración 33: Nivel 2, final*

### 7.4.3. Nivel 2 – GetttinOver

El último nivel es más sencillo y corto que el anterior. La idea para este nivel consiste en que suponga un alivio para el jugador que acaba de superar la parte más difícil de la partida. Al principio, habrá dos zonas con una vida extra relativamente fácil de conseguir para que el jugador recupere la salud que ha perdido antes.

Este nivel es en vertical, a diferencia de los anteriores, para no resultar repetitivo. La esencia de este nivel es que el jugador tendrá que subir hasta lo más alto para llegar a la meta, pero podrá caer en cualquier momento, volviendo a zonas anteriores. Esta mecánica se ha inspirado en el juego *Getting Over It*, el cual trata la frustración, dificultad e injusticia como componentes alrededor de los cuales gira el argumento del videojuego.

Este nivel tendrá tan solo 12 murciélagos y 20 arañas, muy repartidos y cuyo objetivo es entorpecer el avance del jugador, ya que la clave de este nivel será el plataformeo en vez de la acción, al contrario que en el nivel anterior. Por esta razón los enemigos estarán en zonas a las que el jugador solo pueda llegar con su ataque tras haber avanzado saltando. No se intentará aprovechar sus habilidades tanto, si no que más bien se hará lo opuesto de manera que la verdadera dificultad sea saltar y llegar a la plataforma siguiente en vez de acabar con un enemigo.

La parte más interesante de este nivel se encuentra justo al final, donde las plataformas se encontrarán en posiciones mucho más difíciles de alcanzar y habrá una horda de enemigos situados en posiciones clave para que el jugador se vea obligado a tener que avanzar antes para poder acabar con ellos. Esto será la prueba definitiva en un nivel que trataba de explotar todo lo posible la mecánica de plataformeo.

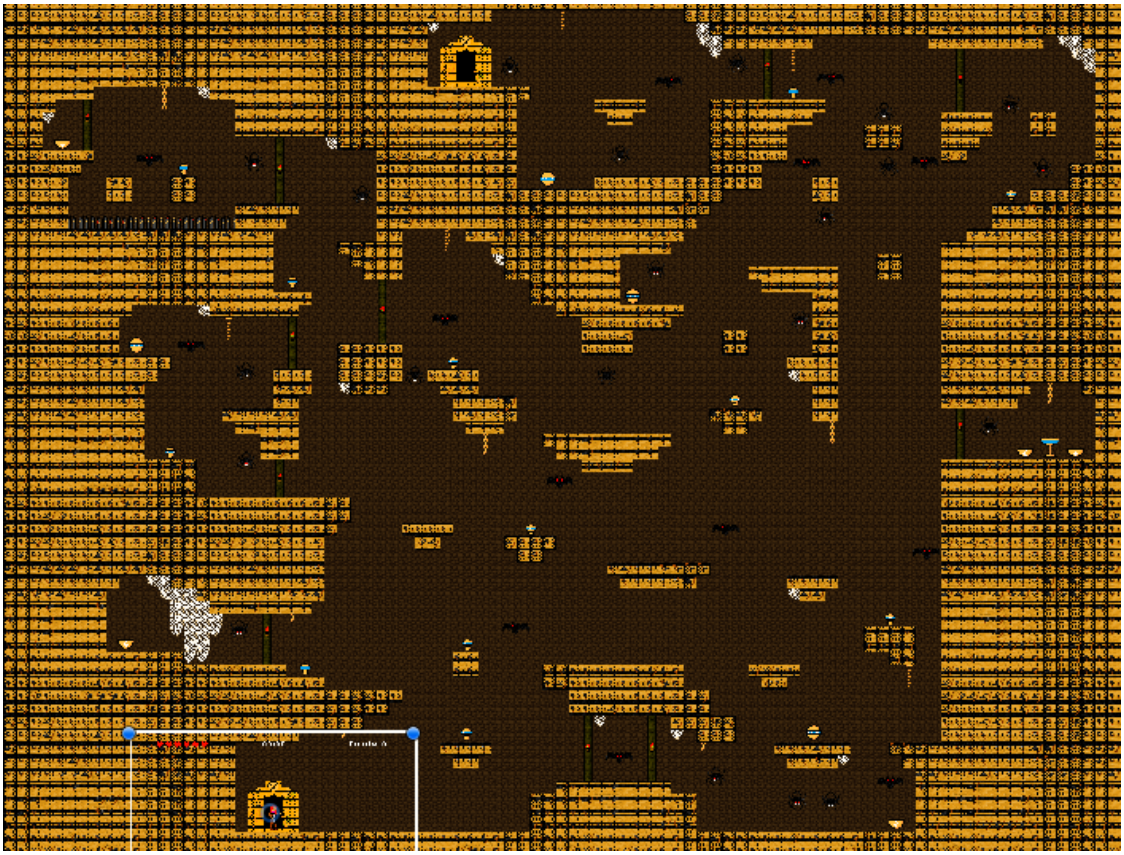


*Ilustración 34: Nivel 3 completo*

### 7.4.4. Pruebas de integridad

Al final de la implementación de cualquier componente se han realizado distintas pruebas para comprobar que este funciona correctamente y en algunos casos para ajustar los valores de ciertas variables. En este apartado se resumirán todas ellas y también se hablará de las pruebas realizadas por otras personas.

Las pruebas de integridad más relevantes se realizaron al principio del proyecto con la creación del script de movimiento del jugador. Se probaron muchos valores diferentes hasta conseguir la velocidad y el salto ideal, de 4 bloques de alto y 9 de ancho. Es de vital importancia conocer las capacidades exactas de movimiento del personaje en un juego de plataformas, y gracias a las pruebas se pudo conseguir y se crearon los niveles alrededor de esta medida.

Después, con el ataque, se realizaron muchas pruebas para comprobar que las partículas se instanciaran correctamente, que las balas desaparecieran al chocar con enemigos o muros, que hicieran daño y para

saber cuánto debería ser la duración de estas, que acabó siendo un segundo de manera que no pudieran llegar muy lejos y acabar con enemigos que ni siquiera se pudieran ver.

Las últimas pruebas más relevantes fueron con los enemigos, con los que se estuvo mucho tiempo hasta configurar de manera satisfactoria tanto las mecánicas de su inteligencia artificial como su vida, velocidad de movimiento, velocidad de los proyectiles, etc. Estas pruebas se realizaron sobre todo al final del proyecto, cuando los niveles estaban creados. De esta manera, se podía saber que la configuración mejoraría el comportamiento de estos en el entorno existente, mejorando de esta manera la calidad del videojuego.

El resto de las pruebas simplemente consisten en comprobar que un elemento funcione como estaba previsto, por ejemplo, que los tesoros desaparezcan y añadan puntuación, que las trampas eliminen al jugador, que cada botón lleve a la escena correcta, que el sistema de corazones funcione correctamente…

Al finalizar el proyecto, se realizaron varias pruebas completando el videojuego para comprobar que todo funcionaba correctamente. Además, se pidió a varias personas que probaran el videojuego para obtener retroalimentación de este. Esto sirvió para corregir el diseño de algunos niveles, como el tutorial, cuyo primer salto resultaba muy molesto al principio y acabó creándose un paso por debajo del salto para que quien quisiera avanzar sin realizar dicho salto pudiera hacerlo. También para modificar la posición de muchos enemigos, la velocidad de disparo de los murciélagos y los controles.

### 7.4.5. Cierre del proyecto

Para finalizar el proyecto, habrá que generar el ejecutable en Unity. Para ello, hay que entrar en la pestaña *Build Settings*, donde se pueden añadir las escenas y asignarles un número (este número es el que se usa cuando el jugador cambia a otra escena). Una vez añadidas y ordenadas, Unity ofrece diferentes plataformas para las cuales construir el proyecto. Se seleccionará PC, Mac y Linux porque la intención es crear el videojuego para ordenador. Aquí también podría seleccionarse Android como plataforma, pero al no haber tenido tiempo suficiente no se ha podido implementar una versión móvil.

Tras seleccionar la opción deseada, se clica en “Build” y Unity crea un proyecto en el directorio deseado con las carpetas y ejecutables siguientes:

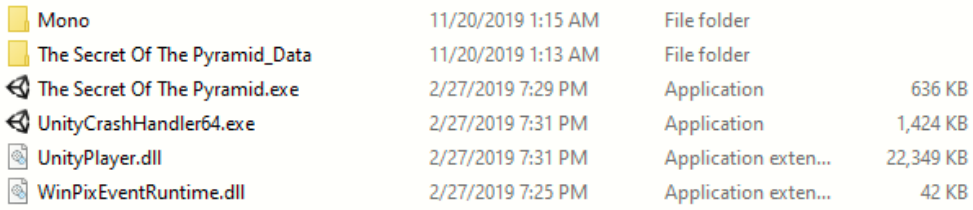



*Ilustración 35: Archivos creados*

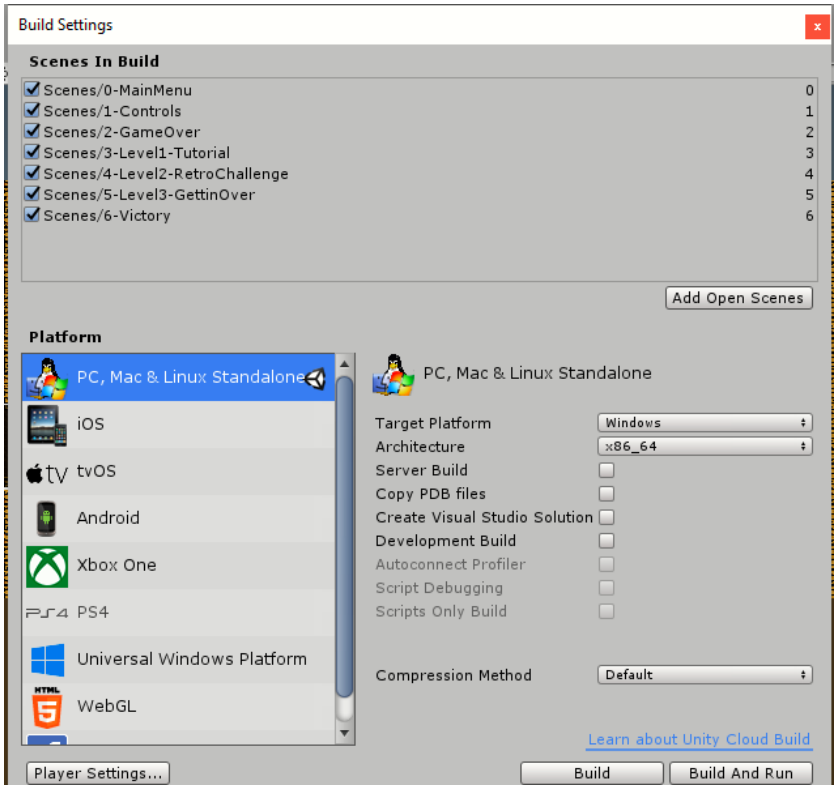



*Ilustración 36: Opciones de creación de Unity*

# 8. RESULTADOS Y CONCLUSIÓN

En este apartado se evalúa el proyecto y se detallan los diagramas de componentes del producto resultante. Para terminar, se da una conclusión del proyecto.

## 8.1. Evaluación del proyecto

El Proyecto final tiene una evaluación generalmente positiva a pesar de no haber conseguido implementar una versión para móvil como se pensaba al principio por razones de tiempo. Aun así, se han creado tres niveles perfectamente funcionales y fluidos, que en conjunto hacen un videojuego con buen ritmo y desafío. Además, estos niveles dejan claro el concepto del videojuego, el cual se seguirá desarrollando.

Durante las pruebas no solo ha mostrado una muy buena funcionalidad, sino que también ha resultado muy interesante y entretenido a las personas que lo probaron, siendo de hecho la crítica más repetida a los controles, que como se dijo anteriormente acabaron por cambiarse, en vez de a aspectos del videojuego.

## 8.2. Diagramas de Componentes

### 8.2.1. Diagrama Global

Un diagrama de las relaciones entre los componentes globales más importantes.

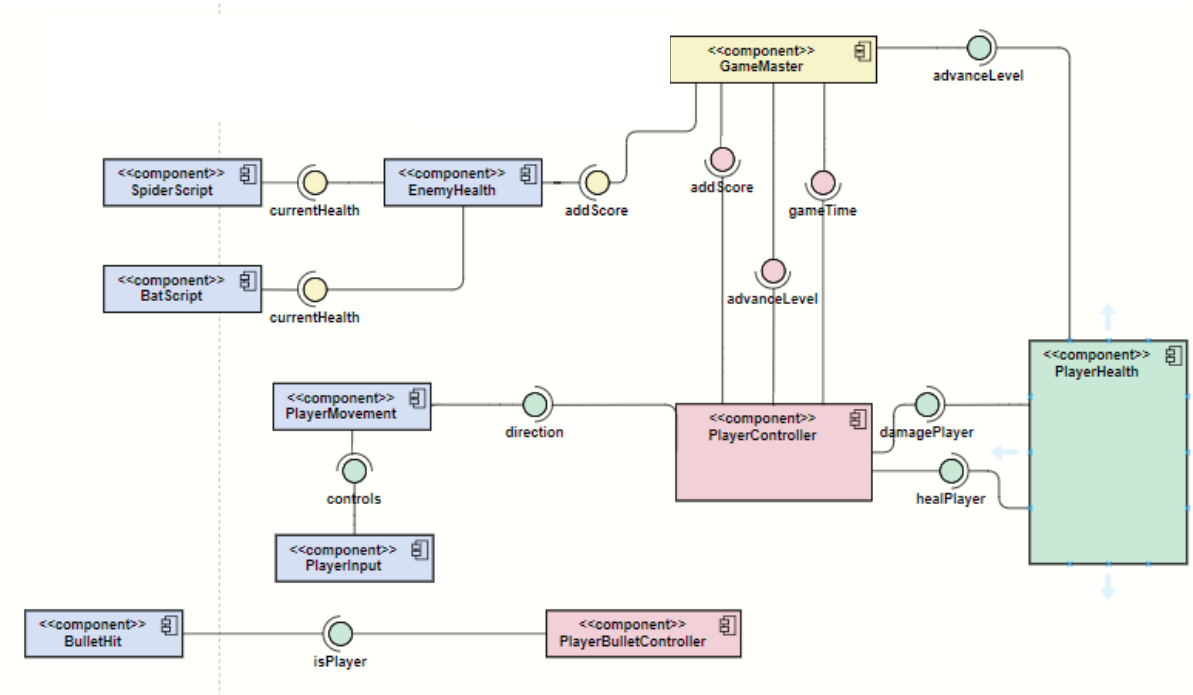



*Ilustración 37: Diagrama de componentes global*

### 8.2.2. Jaina

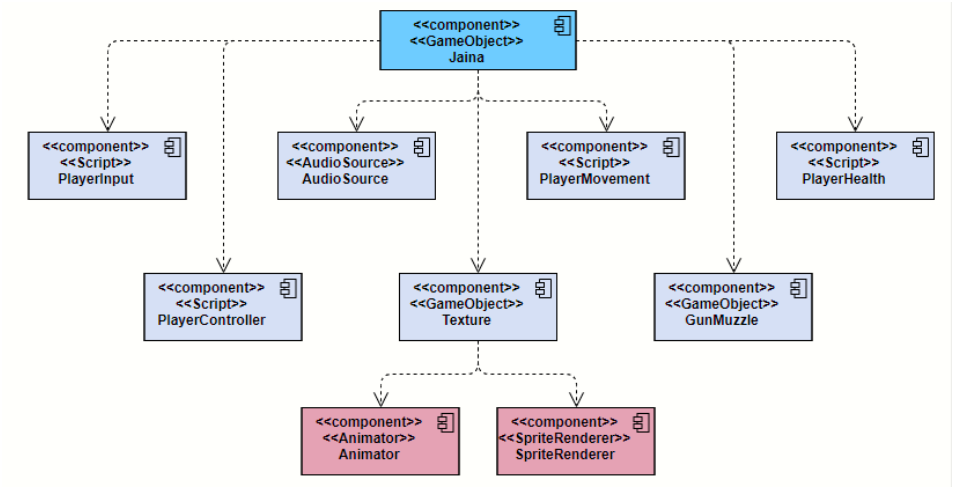



*Ilustración 38: Diagrama de componentes de Jaina*

**Jaina:** Es la entidad que el jugador controlará en el videojuego.

**PlayerInput:** Script que funciona de interfaz entre el hardware con el que el jugador juega y los comandos dentro del juego.

**PlayerController:** Script encargado de las colisiones (recoger puntos, recibir daño, recoger vida) y el ataque del personaje (presionar R para disparar).

**AudioSource:** Objeto encargado de reproducir la canción de fondo, de manera que haya música mientras el personaje esté en pantalla.

**Texture:** Entidad en la que se han encapsulado los gráficos del objeto.

> **Animator:** Objeto encargado de las transiciones entre animaciones del personaje: parado, correr y saltar.
>
> **SpriteRenderer:** Objeto encargado de renderizar los gráficos del jugador.

**PlayerMovement:** Script que controla el movimiento del jugador y sus transiciones: parado, correr y saltar. Recibe de PlayerInput la traducción del hardware con la que ejecuta la acción correspondiente.

**GunMuzzle:** Objeto que representa la posición en la que se instanciarán los proyectiles del jugador.

**PlayerHealth:** Script encargado de gestionar la salud del jugador y de instanciar partículas y efectos según si el jugador recibe daño o se sana.

### 8.2.3. Murciélago

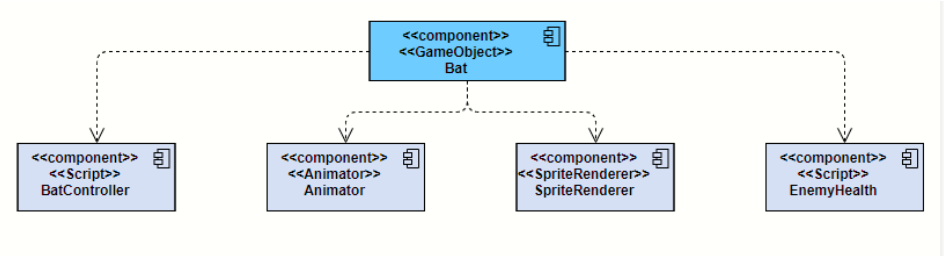



*Ilustración 39: Diagrama de componentes del enemigo murciélago*

**BatController:** Script que gestiona el ataque del murciélago al jugador en función del rango y su propia salud restante (con una vida el ataque será más poderoso).

**Animator**: Objeto encargado de reproducir la animación del murciélago batiendo las alas.

**SpriteRenderer**: Objeto encargado de renderizar los gráficos del murciélago.

**EnemyHealth**: Script encargado de gestionar la salud del murciélago y de instanciar partículas y efectos si se recibe daño.

### 8.2.4. Araña

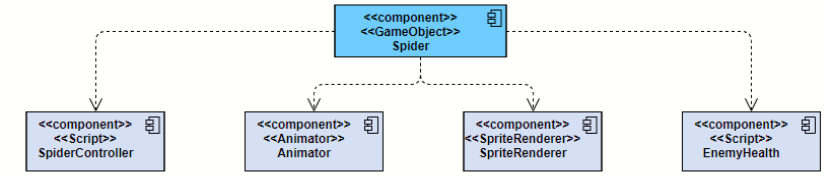



*Ilustración 40: Diagrama de componentes del enemigo araña*

**SpiderController:** Script que gestiona el movimiento en vertical de la araña rebotando en pared y suelo y que activa la persecución al jugador cuando se tiene una vida

**Animator:** Objeto encargado de reproducir la animación de la araña moviendo las patas.

**SpriteRenderer:** Objeto encargado de renderizar los gráficos del murciélago.

**EnemyHealth:** Script encargado de gestionar la salud de la araña y de instanciar partículas y efectos si se recibe daño.

### 8.2.5. Cámara

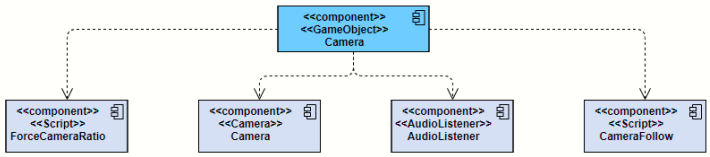



*Ilustración 41: Diagrama de componentes de la cámara*

**ForceCameraRatio:** Script que crea los bordes negros alrededor de la cámara para adaptar el videojuego a cualquier resolución.

**Camera:** Objeto encargado de instanciar una cámara que transmita la renderización de los objetos.

**AudioListener:** Objeto encargado de recibir el audio que suena en pantalla.

**CameraFollow:** Script encargado de hacer que la cámara siga al jugador en el que se podrá también configurar la suavidad de esta persecución.

### 8.2.6. Interfaz de Usuario

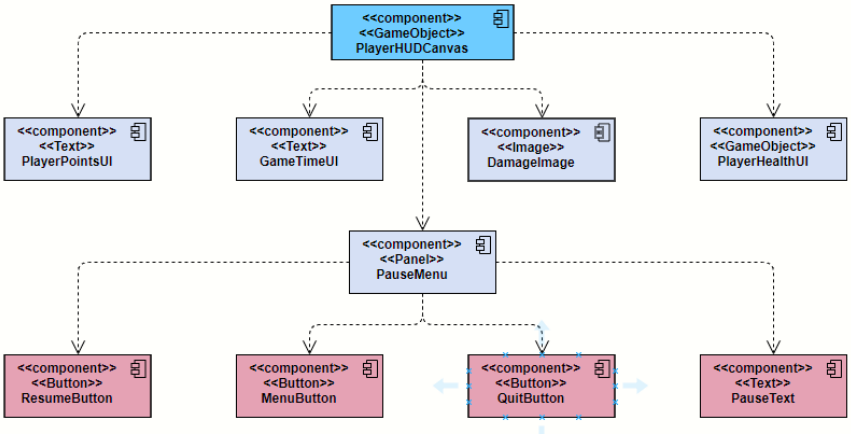



*Ilustración 42: Diagrama de componentes de la interfaz de usuario*

**PlayerPointsUI:** Texto que muestra los puntos del jugador.

**GameTimeUI:** Texto que muestra el tiempo jugado.

**PauseMenu:** Panel con todos los elementos que se muestran cuando el jugador pausa la partida.

**ResumeButton:** Botón para desactivar la pausa.

**MenuButton:** Botón para volver al menú.

**QuitButton:** Botón para quitar el juego.

**PauseText:** Texto en el que pone “Pause”

**DamageImage:** Imagen con sangre que cubre la pantalla y que aparecerá durante un breve momento cuando dañen al jugador.

**PlayerHealthUI:** Objeto contenedor de los corazones que representan la salud restante del jugador.

## 8.3. Diagrama Máquina de Estados

Un diagrama máquina de estados sirve para ofrecer una descripción abstracta del comportamiento de un sistema. Este comportamiento se analiza y representa como una serie de eventos que pueden ocurrir y que variarán o no el estado del objeto. Aquí se presentarán los dos diagramas más representativos del videojuego: la navegación entre escenas, que determinará la escena en la que esté el jugador en cada momento y la inteligencia artificial enemiga, que representa los estados del enemigo y las acciones que realiza en cada uno de ellos.

### 8.3.1. Navegación entre escenas

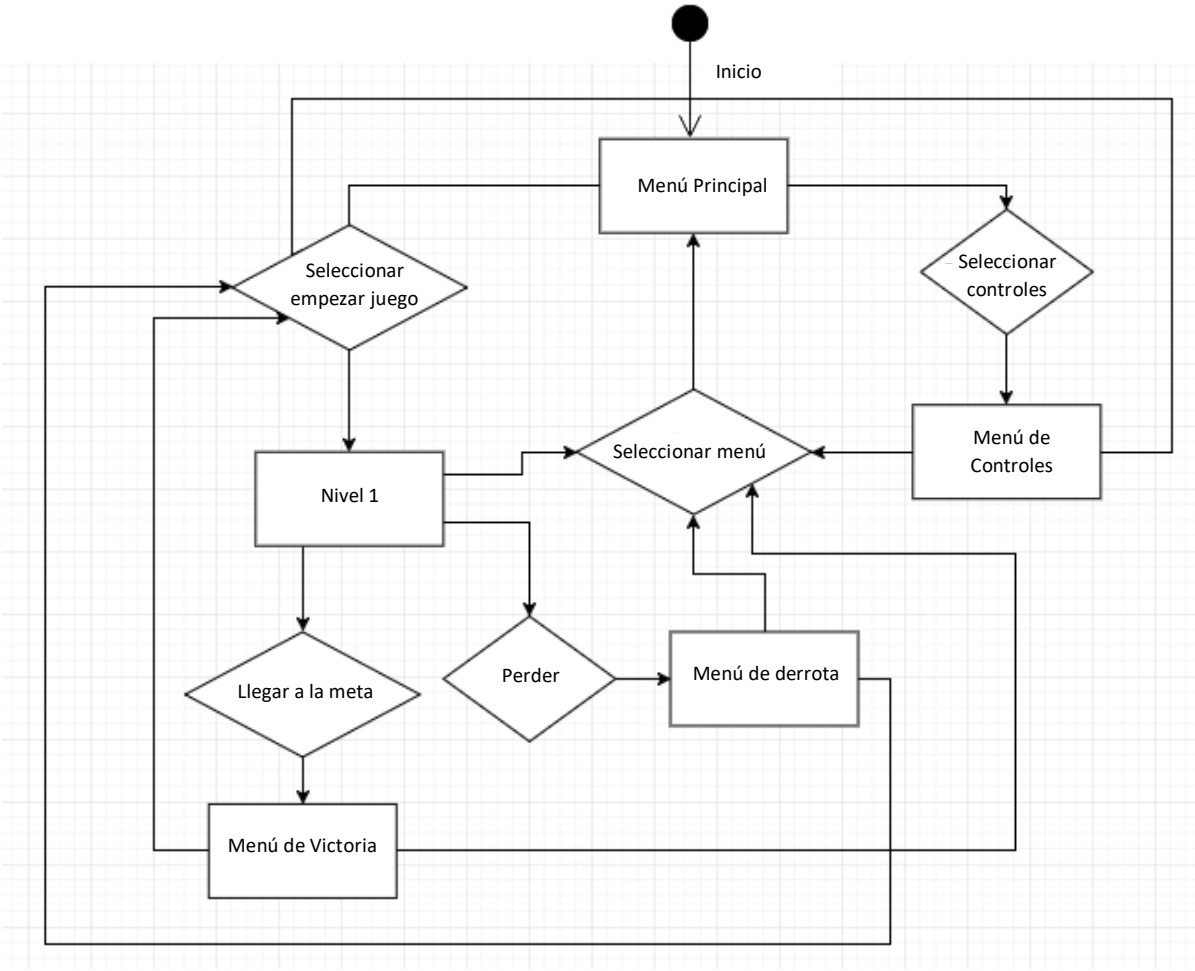



*Ilustración 43: Diagrama Máquina de Estados de la navegación entre menús*

### 8.3.2. Inteligencia Artificial Enemiga

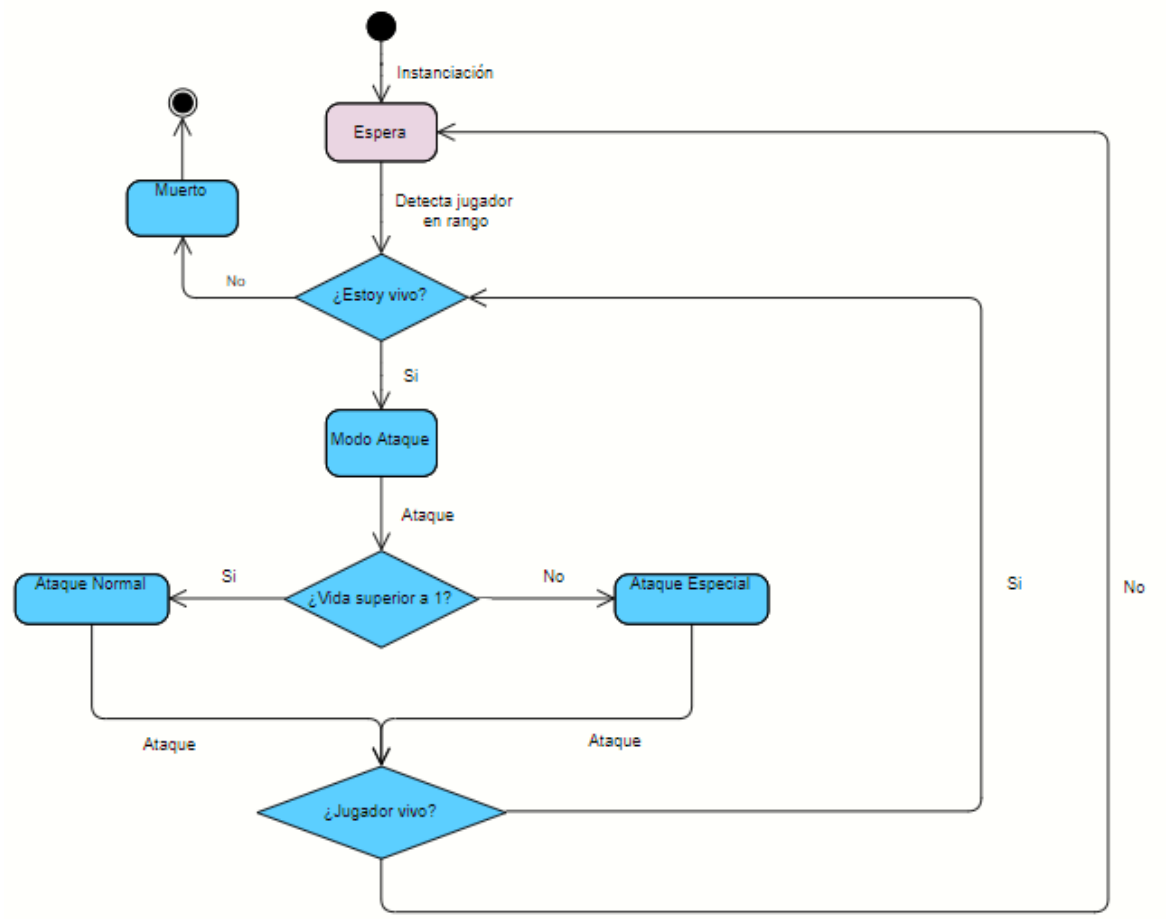



*Ilustración 44: Diagrama Máquina de Estados de la IA de los enemigos*

### 8.4. Conclusiones

En cómputo general, estoy muy contento con el resultado. Creo que el ritmo del juego y las sensaciones que produce son muy acordes a lo que se pretendía conseguir. La IA añade complejidad y las mecánicas se adaptan tanto a jugadores novatos como expertos, ofreciendo una gran rejugabilidad. Se penaliza hacer las cosas rápido y sin pensar al principio, pero conforme se vuelve a jugar se aprende y mejora, lo cual te hace acabar superando las partes más difíciles y sentir una gran satisfacción.

Creo que la planificación ha sido clave para conseguir finalizar el videojuego a tiempo. Fue difícil intentar seguirla por completo, tanto por la naturaleza de este tipo de proyectos como por ser una sola persona, lo que conlleva que ciertas partes del diseño se vuelvan mucho más costosas. Sin embargo, tener una imagen global del proyecto fue muy importante para saber en todo momento lo que faltaba por implementarse.

No tener ningún tipo de experiencia con las tecnologías utilizadas ha sido también un factor negativo a la hora de intentar seguir con la planificación. Sin embargo, gracias a la gran cantidad de documentación de Unity se pudo mantener cierto ritmo, lo cual demuestra que una buena planificación del software a utilizar puede resultar clave para el éxito de un proyecto.

Como aspectos a mejorar, definitivamente el más importante seria mejorar la calidad del diseño de los gráficos. Al no tener conocimientos sobre píxel art o sobre dibujar de ningún tipo, los diseños no llegan a ser todo lo bonito que podrían ser. Además, como el objetivo del proyecto era conseguir un videojuego completo, no se dedicó tanto tiempo a la creación de más sprites que añadieran más variedad y belleza gráfica.

Cabe destacar que tras la experiencia de crear un videojuego por cuenta propia es algo demasiado tedioso y complicado al ser el único en abarcar cada uno de sus campos (inventiva, diseño, programación, música y sonido). Por tanto, considero que en el futuro la manera sensata de afrontar un videojuego sería con un grupo de personas reducido donde cada uno cumpliera con un rol especializado. Sin embargo, considero que la creación del Secreto de la Pirámide me ha servido mucho para crecer en áreas que jamás había probado de no ser por esta oportunidad y creo que eso me ha afectado de manera positiva como desarrollador.

El videojuego seguirá desarrollándose en el futuro próximo, en conjunto con personas que compensen el aspecto gráfico y artístico, con el objetivo de conseguir hacer una versión para dispositivos móvil y añadir un jefe al final.

Finalmente, la creación de un videojuego puede ser un proceso duro y complicado en solitario, pero con una buena planificación y con la colaboración de otras personas este proceso se volvería mucho más sencillo y entretenido. Un videojuego es un proyecto único que junta tecnología y arte para conseguir crear las obras de entretenimiento con las que todo el mundo disfruta hoy en día.

# 9. Anexo

## 9.1. Bibliografía


Carrasco, A. C., 2018. *Blogs Universidad Politécnica de Madrid.* [En línea]
Available at: https://blogs.upm.es/observatoriogate/2018/07/04/que-es-un-motor-de-videojuegos/

Gallego, M. T., 2017. *Metodología Scrum,* Barcelona: Universitat Oberta de Catalunya.

Hallam, J. & Yannakakis, G. N., 2007. TOWARDS OPTIMIZING ENTERTAINMENT IN COMPUTER GAMES. *Applied Artificial Intelligence,* pp. 933-971.

Hinojosa, M. A., 2003. [En línea]
Available at: http://www.colegio-isma.com.ar/Secundaria/Apuntes/Mercantil/4%20Mer/Administracion/Diagrama%20de%20Gantt.pdf

Jackson, S., 2014. *Mastering Unity 2D Game Development.* 2 ed. : Packt Publishing.

Jones, B. y otros, 2014. *Dynamic sprites: artistic authoring of interactive animations.* Utah: University of Utah.

Keith, C., 2010. *Agile Game Development with Scrum.* 1 ed. Boston: Addison-Wesley.

Koster, R., 2013. *Theory of Fun for Game Design,* New York: Paraglyph Press.

LLORENTE & CUENCA, 2018. *El sector de los videojuegos en España: impacto económico y escenarios fiscales,* España: AEVI (Asociación Española de Videojuegos).

Marketing Directo, 2018. *Marketing Directo.* [En línea]
Available at: https://www.marketingdirecto.com/digital-general/digital/el-sector-de-los-videojuegos-supera-al-cine-y-la-musica-en-el-ocio-audiovisual

Mirna Paula Silva, V. d. N. S. & Chaimowicz, L., 2015. *Dynamic Difficulty Adjustment Through an Adaptive AI,* s.l.: s.n.

Okita, A., 2014. *Learning C# Programming with Unity 3D.* s.l.:CRC Press.

PEGI, 2019. *Pegi.* [En línea]
Available at: https://pegi.info/
[Último acceso: 20 11 2019].

Voll, K., 2016. *Less is more: Designing awesome AI.* San Francisco, GDC.

Yannakakis, G. N. & Togelius, J., 2018. *Artificial Intelligence and Games,* Londres: Springer.

### 9.2. Glosario

**Beat'em up:** género de videojuegos que ofrece combate cuerpo a cuerpo entre el protagonista y un número improbable de oponentes.

**Chiptune:** estilo musical que utiliza el sonido distintivo de instrumentos sintetizados característico de los videojuegos arcade.

**Cooldown:** tiempo de espera que necesitamos para volver a utilizar una habilidad en concreto.

**E-sports:** competiciones de videojuegos estructuradas a través de jugadores, equipos, ligas, publicistas, organizadores, comentaristas, patrocinadores y espectadores. Se puede jugar de forma amateur o profesionalizada.

**FPS (First Person Shooter):** género de videojuegos centrado en las armas y desde una perspectiva en primera persona.

**Idle:** animaciones que hace el personaje cuando el jugador no está jugando.

**Indie:** videojuegos independientes creados por individuos o pequeños grupos, sin apoyo financiero de distribuidores.

**Plataformas:** género de videojuegos que se caracteriza por tener que caminar, correr, saltar o escalar sobre una serie de plataformas y acantilados, con enemigos, mientras se recogen objetos para poder completar el juego

**Plataformeo:** secciones de un juego que, no siendo esencialmente del género de plataformas, contienen fases o zonas que deben superarse siguiendo el esquema de este género.

**Prefab:** es la copia de un objeto del juego, convertida en un componente reusable. Se encuentra en la carpeta del proyecto como un objeto normal y está serializado como un archivo en el disco. Un prefab puede contener una jerarquía de objetos del juego.

**Rejugabilidad:** término usado en videojuegos y juegos de mesa para describir el potencial que tiene un título de seguir siendo jugado después de que se termine la primera vez.

**Scene:** en español se podría conocer como escenario. Contiene los entornos y menús del videojuego, además de todos los elementos situados en ella y su jerarquía.

**Script:** cada archivo C# del juego. Contiene la lógica de programación de un aspecto especifico o de un elemento del videojuego.

**Shooter:** género de videojuegos centrado en las armas

**Sprite:** mapa de bits en dos dimensiones que se dibuja directamente sin usar canalización de transformaciones o iluminación.

**Testing:** fase de desarrollo de un producto consistente en las pruebas sobre el software de este.

**2.5 dimensiones:** juegos en 3D que usan gráficos poligonales para representar el mundo y/o personajes, pero cuyo juego se limita a un plano 2D.

### 9.3. Índice de ilustraciones

### 9.4. Índice de tablas

### 9.5. Código fuente e instrucciones de uso

Todo el código elaborado en este proyecto se puede encontrar en el repositorio de Github siguiente:

https://github.com/JaiRoma97/the-secret-of-the-pyramid

El proyecto completo se encuentra en el enlace de drive siguiente:

https://drive.google.com/drive/u/0/folders/1LaX3GFky_o_bbhKRHTn-dxIfbyqWs3mY

Tras la descarga del directorio "Jugable" en formato zip, descomprimir el archivo con 7zip o Winrar y simplemente iniciar el videojuego dando doble click al archivo ejecutable con el nombre del título del videojuego.

Al iniciar el juego se recomendará ver los controles en el menú principal. Estos consisten en usar la R para disparar, el espacio para saltar y las flechas de dirección o AD para moverse. Una vez leídos, se puede empezar el primer nivel al presionar el botón de "*Jugar*".

### 9.6. Requisitos funcionales

<table>
<tr><td>RF-01</td><td>Empezar partida</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema permitirá elegir entre empezar partida, ver los controles del juego o quitar el juego.</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en el menú principal.</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El usuario inicia la aplicación</td></tr><tr><td>P2</td><td>El usuario escoge empezar partida</td></tr><tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>Se comienza una nueva partida y se reinician la puntuación, tiempo y vida.</td></tr>
<tr><td>Excepciones</td><td></td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-02</td><td>Ver los controles</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema permitirá elegir entre empezar partida, ver los controles del juego o quitar el juego.</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en el menú principal.</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El usuario inicia la aplicación</td></tr><tr><td>P2</td><td>El usuario escoge ver controles</td></tr><tr><td>P3</td><td>El sistema inicia la escena de los controles</td></tr><tr><td>P4</td><td>El motor gráfico muestra la interfaz con la información sobre los controles</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>El jugador está en la escena de los controles</td></tr>
<tr><td>Excepciones</td><td></td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-03</td><td>Mover al personaje</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema reaccionará a los controladores de dirección y moverá al personaje en esta.</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida.</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>P1</td><td>El usuario inicia la aplicación</td></tr>
<tr><td>P2</td><td>El usuario escoge empezar partida</td></tr>
<tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr>
<tr><td>P4</td><td>El usuario acciona el controlador de dirección</td></tr>
<tr><td>P5</td><td>El sistema traduce la entrada como una velocidad en el eje x</td></tr>
<tr><td>P6</td><td>El motor gráfico aplica esta velocidad sobre el personaje durante cada frame</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td>El personaje se mueve en la dirección pulsada por el jugador</td></tr>
<tr><td>Excepciones</td><td>El personaje está muerto</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-04</td><td>Hacer saltar al personaje</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema reaccionará a los controladores de salto y hará saltar al personaje.</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida.</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>P1</td><td>El usuario inicia la aplicación</td></tr>
<tr><td>P2</td><td>El usuario escoge empezar partida</td></tr>
<tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr>
<tr><td>P4</td><td>El usuario acciona el controlador de salto</td></tr>
<tr><td>P5</td><td>El sistema traduce la entrada como una fuerza en el eje y</td></tr>
<tr><td>P6</td><td>El motor gráfico ejecuta esta aceleración moviendo al personaje en el eje y</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td>El personaje salta y está en el aire</td></tr>
<tr><td>Excepciones</td><td>El personaje está muerto</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-05</td><td>Disparar</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema detectará los controladores de disparo e instanciará un proyectil.</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida.</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>P1</td><td>El usuario inicia la aplicación</td></tr>
<tr><td>P2</td><td>El usuario escoge empezar partida</td></tr>
<tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr>
<tr><td>P4</td><td>El usuario acciona el controlador de disparo</td></tr>
<tr><td>P5</td><td>El sistema traduce la entrada como una instanciación de un proyectil con la dirección del personaje y una velocidad</td></tr>
<tr><td>P6</td><td>El motor gráfico ejecuta esta velocidad sobre el proyectil</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>El proyectil se mueve en la dirección disparada hasta impactar</td></tr>
<tr><td>Excepciones</td><td>El personaje está muerto o aún no ha pasado el tiempo de recarga</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-06</td><td>Eliminar enemigo</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema reducirá la salud de un enemigo cada vez que este colisione contra el jugador o un proyectil del jugador. Cuando esta sea menor que 0 lo destruirá.</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida.</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>P1</td><td>El usuario inicia la aplicación</td></tr>
<tr><td>P2</td><td>El usuario escoge empezar partida</td></tr>
<tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr>
<tr><td>P4</td><td>El usuario dispara o choca con un enemigo</td></tr>
<tr><td>P5</td><td>El sistema de colisiones detecta el impacto contra el enemigo</td></tr>
<tr><td>P6</td><td>El enemigo pierde una cantidad de salud que depende del tipo de impacto</td></tr>
<tr><td>P7</td><td>El sistema detecta que la salud de enemigo es igual o inferior que 0</td></tr>
<tr><td>P8</td><td>El sistema da un punto al jugador y elimina al enemigo</td></tr>
<tr><td>P9</td><td>El motor gráfico muestra en la interfaz la nueva puntuación y la animación de muerte del enemigo</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>El jugador tiene un punto más y el enemigo desaparece de la pantalla</td></tr>
<tr><td>Excepciones</td><td>El personaje está muerto</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-07</td><td>Perder salud</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema reducirá la salud del personaje cada vez que este colisione contra un enemigo, un proyectil enemigo o una trampa. Cuando esta sea menor que 0 lo destruirá.</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida.</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>P1</td><td>El usuario inicia la aplicación</td></tr>
<tr><td>P2</td><td>El usuario escoge empezar partida</td></tr>
<tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr>
<tr><td>P4</td><td>El usuario choca con un enemigo, proyectil enemigo o trampa</td></tr>
<tr><td>P5</td><td>El sistema de colisiones detecta el impacto</td></tr>
<tr><td>P6</td><td>El jugador pierde una cantidad de salud que depende del tipo de impacto</td></tr>
<tr><td>P7</td><td>El motor gráfico muestra en la interfaz una pantalla llena de sangre y elimina una cantidad de corazones igual a la cantidad de vida perdida</td></tr>
<tr><td>P8</td><td>El sistema detecta que la salud del personaje es igual o inferior que 0 e inicia el menú de derrota</td></tr>
<tr><td>P9</td><td>El motor gráfico ejecuta la animación de muerte y elimina al personaje de la pantalla</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>El jugador está en el menú de derrota</td></tr>
<tr><td>Excepciones</td><td>El personaje tiene una salud superior a 1</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-08</td><td>Recoger tesoro</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El jugador recogerá tesoros chocando con ellos para conseguir puntuación</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida.</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>P1</td><td>El usuario inicia la aplicación</td></tr>
<tr><td>P2</td><td>El usuario escoge empezar partida</td></tr>
<tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr>
<tr><td>P4</td><td>El usuario atraviesa un tesoro</td></tr>
<tr><td>P5</td><td>El sistema de colisiones detecta el impacto</td></tr>
<tr><td>P6</td><td>El jugador gana una cantidad de puntos que depende del tipo de tesoro</td></tr>
<tr><td>P7</td><td>El motor gráfico muestra en la interfaz la nueva puntuación y la animación de destrucción del tesoro</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td>El jugador tiene una cantidad de puntos mayor y el tesoro ha desaparecido</td></tr>
<tr><td>Excepciones</td><td>El personaje está muerto</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-09</td><td>Recoger vida extra</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema aumentará en uno la salud del personaje cada vez que este colisione con un contenedor de salud</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida.</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>P1</td><td>El usuario inicia la aplicación</td></tr>
<tr><td>P2</td><td>El usuario escoge empezar partida</td></tr>
<tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr>
<tr><td>P4</td><td>El usuario choca con un contenedor de salud</td></tr>
<tr><td>P5</td><td>El sistema de colisiones detecta el impacto</td></tr>
<tr><td>P6</td><td>El sistema añade un punto de salud al personaje</td></tr>
<tr><td>P7</td><td>El motor gráfico muestra en la interfaz un nuevo corazón</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td>El jugador tiene un punto de salud más</td></tr>
<tr><td>Excepciones</td><td>Tiene la salud máxima</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-10</td><td>Completar nivel</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema iniciará el escenario siguiente</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida.</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El sistema inicia una escena</td></tr><tr><td>P2</td><td>El usuario hace que el jugador atraviese la puerta al final del nivel</td></tr><tr><td>P3</td><td>El sistema de colisiones detecta el impacto</td></tr><tr><td>P4</td><td>El sistema inicia el siguiente nivel</td></tr><tr><td>P5</td><td>El motor gráfico ejecuta una animación con partículas</td></tr><tr><td>P6</td><td>El sistema reinicia el tiempo</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>El jugador está en el nivel siguiente</td></tr>
<tr><td>Excepciones</td><td>El personaje está muerto</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-11</td><td>Pausar el juego</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema parará el tiempo de juego y mostrará una interfaz con las opciones de reanudar el juego volver al menú o salir</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida no pausada</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El usuario inicia la aplicación</td></tr><tr><td>P2</td><td>El usuario escoge empezar partida</td></tr><tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr><tr><td>P4</td><td>El usuario acciona el controlador de pausa</td></tr><tr><td>P5</td><td>El sistema detecta la entrada y para el tiempo de juego</td></tr><tr><td>P6</td><td>El motor gráfico muestra la interfaz de pausa</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>El videojuego está pausado</td></tr>
<tr><td>Excepciones</td><td></td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-12</td><td>Reanudar el juego</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema devolverá el juego a su flujo normal</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida pausada</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>P1</td><td>El usuario inicia la aplicación</td></tr>
<tr><td>P2</td><td>El usuario escoge empezar partida</td></tr>
<tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr>
<tr><td>P4</td><td>El usuario acciona el controlador de pausa o presiona el botón de reanudar</td></tr>
<tr><td>P5</td><td>El sistema detecta la entrada y restaura el tiempo de juego</td></tr>
<tr><td>P6</td><td>El motor gráfico oculta la interfaz de pausa</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td>El videojuego no está pausado</td></tr>
<tr><td>Excepciones</td><td></td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-13</td><td>Volver al menú desde la interfaz de pausa</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema iniciará la escena del menú principal</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en una partida pausada</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>P1</td><td>El usuario inicia la aplicación</td></tr>
<tr><td>P2</td><td>El usuario escoge empezar partida</td></tr>
<tr><td>P3</td><td>El sistema inicia la escena del primer nivel</td></tr>
<tr><td>P4</td><td>El usuario presiona el botón de menú</td></tr>
<tr><td>P5</td><td>El sistema detecta la entrada e inicia la escena del menú principal</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td>El jugador está en el menú principal</td></tr>
<tr><td>Excepciones</td><td></td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-14</td><td>Volver a empezar partida</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema iniciará la escena del primer nivel y restaurará todas las variables globales a su estado inicial</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en el menú de victoria o de derrota.</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El sistema inicia la escena</td></tr><tr><td>P2</td><td>El usuario acciona el controlador de movimiento</td></tr><tr><td>P3</td><td>El personaje se mueve en la dirección accionada</td></tr><tr><td>P4</td><td>El personaje atraviesa la única puerta que hay en pantalla</td></tr><tr><td>P5</td><td>El sistema de colisiones detecta el impacto</td></tr><tr><td>P6</td><td>El sistema restaura las variables globales a su estado inicial</td></tr><tr><td>P7</td><td>El sistema inicia la escena del primer nivel</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>El jugador está en el nivel 1 sin puntos, con el tiempo a 0 y la salud máxima</td></tr>
<tr><td>Excepciones</td><td></td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-15</td><td>Volver al menú desde la derrota</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema iniciará la escena del menú principal</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en el menú de victoria o de derrota.</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El sistema inicia la escena</td></tr><tr><td>P2</td><td>El usuario acciona el controlador de movimiento</td></tr><tr><td>P3</td><td>El personaje se mueve en la dirección accionada</td></tr><tr><td>P4</td><td>El personaje choca con las trampas</td></tr><tr><td>P5</td><td>El sistema de colisiones detecta el impacto</td></tr><tr><td>P6</td><td>El sistema inicia la escena del menú principal</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>El jugador está en el menú principal</td></tr>
<tr><td>Excepciones</td><td></td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

<table>
<tr><td>RF-16</td><td>Ganar la partida</td></tr>
<tr><td>Versión</td><td>1.00</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01: Jugar una partida</td></tr>
<tr><td>Descripción</td><td>El sistema iniciará la escena de victoria donde el jugador podrá empezar otra partida o volver al menú</td></tr>
<tr><td>Precondición</td><td>El usuario debe estar en el tercer nivel de una partida.</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>P1</td><td>El sistema inicia la escena del tercer nivel</td></tr><tr><td>P2</td><td>El usuario atraviesa la puerta final del tercer nivel</td></tr><tr><td>P3</td><td>El sistema de colisiones detecta el impacto</td></tr><tr><td>P4</td><td>El sistema inicia la escena de victoria</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>El jugador está en el menú de victoria</td></tr>
<tr><td>Excepciones</td><td></td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

UNIVERSIDAD DE ALMERÍA


## Resumen/Abstract

En este trabajo de fin de grado se desarrollará un videojuego en 2D de estilo retro y género plataformas y acción con unas mecánicas entretenidas que añadan profundidad y diversión al producto.

Este videojuego pretende ofrecer una complejidad e interés a través del diseño tal que consiga enganchar al jugador desde el primer momento. Este proyecto multidisciplinar mezcla tanto el desarrollo del código del videojuego completo como la gestión de todo el proyecto, su diseño y la creación de todos sus elementos.

GRADO EN INGENIERÍA INFORMÁTICA, 2018/2019