UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo e implementación de mecánicas y enemigos en un videojuego de sigilo

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: de Castro Isasi, Daniel

Tutor/a: Lluch Crespo, Javier

CURSO ACADÉMICO: 2021/2022

Resum

La finalitat del projecte presentat consisteix en el plantejament i desenvolupament d'un videojoc 2D, classificat en el gènere de plataformes amb algunes mecàniques de sigil en conjunt amb dos companys més. Aquest treball se centrarà en la implementació del personatge del jugador i les seves funcionalitats, així com els enemics i la seva intel·ligència artificial, encara que també s'ha participat en altres seccions del joc tal com s'explicarà al llarg de la memòria. Per al desenvolupament del videojoc s'ha utilitzat Unity com a motor gràfic.

Paraules clau: Unity, Videojuego, 2D, Plataformas, Sigilo, IA, Prototipado

Resumen

La finalidad del proyecto presentado consiste en el planteamiento y desarrollo de un videojuego 2D clasificado en el género de plataformas con algunos toques de sigilo en conjunto con otros dos compañeros. Este trabajo se centrará en la implementación de la protagonista y sus funcionalidades, así como en los enemigos y la inteligencia artificial de los mismos, aunque también se ha participado en otros apartados del videojuego como se explicará a lo largo de la memoria. Para el desarrollo del juego se ha utilizado Unity como motor gráfico.

Palabras clave: Unity, Videojoc, 2D, Plataformes, Sigil, IA, Prototipat

Abstract

This project consists in planning and developing a 2D video game, which would be classified in the platforming and stealth genres, helped by two other partners. This assignment will focus on the implementation of the player character as well as the enemies and their artificial intelligence, although, as we will see through this paper, some help has been provided in another sections of the video game. We have used Unity's game engine to develop the game.

Key words: Unity, Videogame, 2D, Platforming, Stealth, AI, Prototyping

Índice general

Índice general v
Índice de figuras vii
Índice de tablas vii

1 Introducción 1
1.1 Motivación . . . 1
1.2 Objetivos . . . 1
1.3 Metodología . . . 2
1.4 Estructura de la memoria . . . 3
1.5 Colaboraciones . . . 4
2 Estado del arte 5
2.1 El videojuego y sus inicios como medio de entretenimiento . . . 5
2.2 Importancia de los videojuegos en el mundo contemporáneo . . . 6
2.3 El sigilo en los videojuegos . . . 6
2.4 Videojuegos en línea . . . 7
2.5 Motores gráficos más utilizados . . . 8
3 Análisis del problema 11
3.1 Casos de uso . . . 11
3.2 Especificación de requisitos . . . 14
3.3 Explicación del videojuego . . . 17
3.4 Plan de trabajo . . . 18
4 Diseño de la solución 23
4.1 Tecnología utilizada: Unity . . . 23
4.2 Interfaz . . . 24
4.3 Niveles . . . 25
4.4 Protagonista . . . 26
4.5 Enemigos y jefe . . . 26
4.6 Objetos . . . 30
4.7 Interacción con el escenario . . . 30
4.8 Misceláneo . . . 30
5 Desarrollo de la solución 33
5.1 Niveles . . . 33
5.2 Protagonista . . . 34
5.2.1 Movimiento . . . 35
5.2.2 Combate . . . 36
5.2.3 Inventario . . . 37
5.3 Objetos . . . 37
5.4 Enemigos . . . 38
5.4.1 Máquinas de estados . . . 39
5.4.2 Estado: Quieto . . . 40
5.4.3 Estado: Patrullar . . . 40
5.4.4 Estado: Perseguir . . . 40

5.4.5 Estado: Atacar . . . 41
5.4.6 Estado: Señuelo . . . 42
5.4.7 Estado: Fase 1 Jefe . . . 42
5.4.8 Estado: Fase 2 Jefe . . . 43
5.4.9 Estado: Cargar Jefe . . . 43
5.4.10 Estado: Disparar Jefe . . . 43
5.4.11 Estado: Morir . . . 44
5.4.12 Enemigos comunes . . . 44
5.4.13 Jefe final . . . 45
5.5 Escenario e interacciones . . . 45
5.6 Interfaces . . . 46
5.6.1 Cuadros de texto . . . 46
5.7 Almacenamiento persistente . . . 46
5.8 Sonido . . . 47

6 Pruebas **49**
6.1 Beta . . . 49
6.2 Feedback . . . 50
6.3 Enlaces de interés . . . 50

7 Conclusiones **51**

8 Relación con los estudios **53**

9 Trabajos futuros **55**

10 Glosario **57**

Bibliografía **61**

A Controles **63**

B ODS **65**

Índice de figuras

1.1 Modelo Ágil vs Cascada . . . 3

2.1 Pong . . . 6
2.2 Shoplifting Boy (1979) y Dishonored (2012) . . . 7
2.3 SpaceWars! (1962) y World of Warcraft (2004) . . . 8
2.4 Unreal Engine: Bioshock y REDEngine: The Witcher 3 . . . 9

3.1 Caso de uso Menú principal . . . 11
3.2 Caso de uso Menú pausa . . . 12
3.3 Caso de uso Protagonista . . . 13
3.4 Caso de uso Enemigo . . . 14
3.5 Inicio del juego . . . 17
3.6 Sprints en el desarrollo del proyecto . . . 19
3.7 Ejemplo de tarjeta de Trello . . . 20
3.8 Planificación real . . . 21

4.1 Herramienta Tilemaps . . . 24
4.2 Sprite del protagonista . . . 26
4.3 Sprites de los enemigos . . . 27
4.4 Máquina de estados de enemigos comunes . . . 27
4.5 Máquina de estados del jefe final . . . 27

5.1 Nivel de tutorial . . . 34
5.2 Controlador de animaciones . . . 35

Índice de tablas

3.1 Reparto de tareas . . . 22

4.1 Transiciones del menú principal . . . 24
4.2 Transiciones del menú de pausa . . . 25
4.3 Transiciones del menú de victoria . . . 25
4.4 Transiciones del menú de muerte . . . 25
4.5 Agentes: Estado patrullar . . . 28
4.6 Agentes: Estado quieto . . . 28
4.7 Agentes: Estado perseguir . . . 28
4.8 Agentes: Estado atacar . . . 28
4.9 Agentes: Estado señuelo . . . 28
4.10 Agentes: Jefe, estado fase 1 . . . 29

4.11 Agentes: Jefe, estado fase 2 29
4.12 Agentes: Jefe, estado cargar 29
4.13 Agentes: Jefe, estado disparar 29
4.14 Agentes: Estado morir 29

A.1 Controles 63

CAPÍTULO 1
Introducción

En este primer capítulo se presentarán las razones que han motivado a la realización de este proyecto de final de grado, así como los objetivos que se han querido llevar a cabo y la metodología seguida durante el transcurso del desarrollo del juego. Más adelante se explicará la estructura que va a seguir esta memoria y los distintos estudiantes que han colaborado en el desarrollo del proyecto.

1.1 Motivación

Los videojuegos han estado presentes en mi vida desde pequeño, empezando jugando a las consolas portátiles de Nintendo, pasando por las sobremesas de Sony y acabando dedicando la mayoría de ese tiempo con juegos de PC. Desde siempre he sentido curiosidad por la programación y las dos cosas se acabaron por juntar, preguntándome continuamente como escribiendo líneas de código, se podían crear esos mundos en los que pasaba varias horas todas las semanas.

A lo largo de la carrera esta curiosidad fue aumentando junto con los conocimientos de programación hasta llegar a la asignatura de programación de videojuegos, donde descubrí que me gustaba tanto programar esos juegos como jugarlos después, por lo que me pareció una decisión sencilla dedicar este último trabajo a desarrollar un videojuego como tantos otros que he jugado desde mi infancia hasta ahora.

1.2 Objetivos

El objetivo de este TFG será el de desarrollar un videojuego suficientemente complejo e interesante como para que sea capaz de entretener al usuario que decida jugarlo en un futuro. Para ello se seguirán una serie de objetivos más pequeños que darán forma a todo el proyecto, desde el juego hasta todo el trabajo que este conlleva:

- **Género de plataformas y sigilo:** El videojuego debe distinguirse como una experiencia principalmente de plataformas, con algunos toques de sigilo para mejorar la experiencia del jugador, ofreciendo diferentes opciones para superar los retos que se presenten.
- **Alta interactividad del entorno:** A la hora de idear el juego se le ha dado mucha importancia a que el jugador sienta que forme parte del mundo en el que se encuentra. Para ello se permitirá que realice varias acciones con el entorno que le rodea y no solo con los enemigos.

- **Enemigos reactivos a estímulos:** Los enemigos estarán programados de forma que serán capaces de reaccionar tanto al entorno que les rodea como a las acciones del jugador, de forma que un enemigo con la misma programación sea capaz de funcionar en cualquier escenario y situación.
- **Posibilidad de jugar con un amigo:** El juego ofrecerá la posibilidad de jugar en un modo cooperativo *online* con otro jugador, de forma que se pueda disfrutar solo o con un amigo.

Además de estos, uno de nuestros objetivos principales ha sido desarrollar el videojuego con constancia y siguiendo una metodología ágil que nos permita seguir una organización clara desde el principio, evitando en la medida de lo posible cargas excesivas de trabajo de cara al final.

1.3 Metodología

A la hora de desarrollar el videojuego se ha seguido una metodología ágil caracterizada por un desarrollo incremental, realizando diferentes iteraciones y añadiendo y puliendo diferentes elementos en cada una de ellas. En concreto, nos hemos acercado al modelo *scrum*, el cual se caracteriza por una mayor cantidad de *sprints* con menos carga de trabajo pero más cortos y frecuentes. Esta metodología se puede dividir por las siguientes fases:

- **Planificación:** En esta fase se especifica el fin con el que se realiza cada sprint, es decir, los objetivos y el desarrollo de las UTs, riesgos, plazos de entrega y las pruebas de aceptación que se realizarán.
- **Desarrollo:** En esta fase se implementan las UTs hechas en la anterior etapa y se llevan a cabo las pruebas de especificación, todo en el periodo definido para el sprint.
- **Revisión:** Una vez realizado el sprint, se busca la opinión del cliente, mostrando los avances que se han realizado y pidiendo una evaluación que pueda ser utilizada en la siguiente fase.
- **Retroalimentación:** Los datos obtenidos en la fase anterior se aplican al proyecto y se estudian los posibles cambios que se podrán realizar en la planificación de los siguientes *sprints*.

Esta metodología aporta una gran flexibilidad y rapidez a la hora de desarrollar el proyecto, en especial tratándose de un trabajo en el que han colaborado varias personas. Su gran porcentaje de éxito, como es secundado por diversos estudios, es la razón por la que cada vez más empresas deciden aplicar este modelo de trabajo. Uno de estos estudios, proveniente del *Standish Group Chaos* [1], lo podemos ver en la figura 1.1, donde se presenta la diferencia de porcentajes de éxito entre esta metodología y la de cascada, otra de las más utilizadas por las empresas.

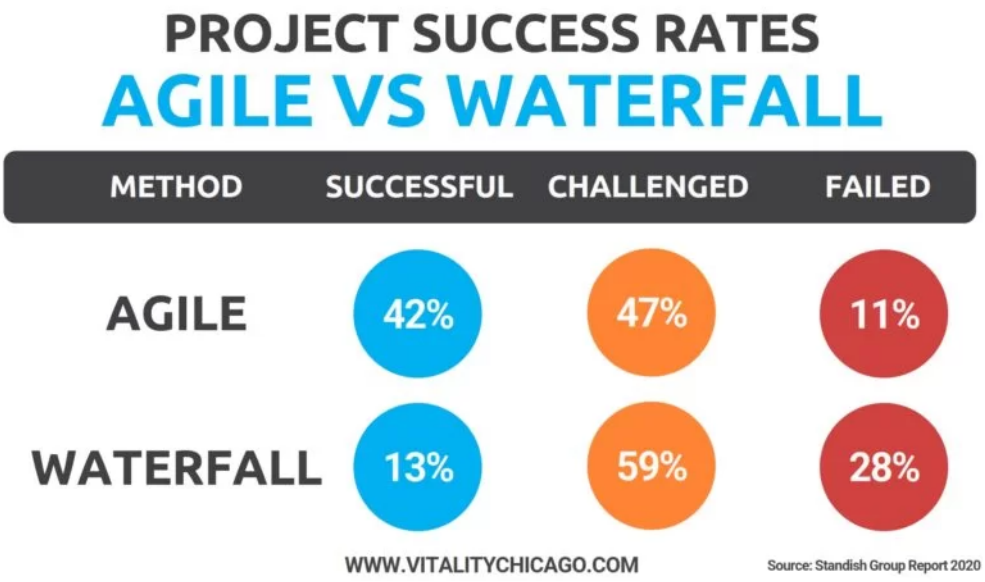

Figura 1.1: Modelo Ágil vs Cascada

Al aplicar el modelo ágil para cualquier trabajo en una empresa, son necesarios tres roles fundamentales:

- **Scrum Master:** Es el encargado de garantizar que el equipo cumple los objetivos aportados por la metodología ágil mediante la facilitación de la comunicación entre el cliente y el equipo de desarrollo. Esta persona debe disponer de un conjunto de habilidades tanto comunicativas como estratégicas, actuando como un entrenador de equipo.
- **Product owner:** Es quien hace lo posible por mantener la visión del producto inicial, asegurando que se cumplen las necesidades del cliente así como sus clientes. Tiene como objetivo maximizar el valor de la entrega y se encarga de interactuar con los clientes y manejar el *Backlog*.
- **Equipo de desarrollo:** Son aquellos trabajadores con un objetivo común, los cuales se encargan de cumplir con las tareas en el plazo establecido, así como interactuar en menor medida con los clientes cuando sea necesario para entender sus necesidades. El equipo suele estar formado por un grupo de entre cinco y nueve trabajadores.

1.4 Estructura de la memoria

En este apartado se hablará de las diferentes partes en las que va a consistir esta memoria, así como una pequeña explicación de cada una de ellas.

- El primer capítulo consiste en una **introducción**, donde se habla de las motivaciones detrás de la realización de este trabajo, así como la metodología que se ha seguido durante todo el proceso.
- En el siguiente capítulo se presentará el **estado del arte**, donde se hablará del estado de los videojuegos, desde que surgieron hasta la importancia de los mismos en la actualidad, centrándonos en aquellos con más relevancia de cara a nuestro proyecto por su similitud en género o temática.

- En el **análisis del problema** se hablará en términos generales del juego, haciendo hincapié en los primeros pasos seguidos en el desarrollo, la metodología utilizada y los requisitos que se han tenido en cuenta a lo largo de todo el proyecto.
- La parte central de la memoria se dividirá en los apartados de **diseño** y **desarrollo** de la solución, donde se explicará detalladamente cada uno de los elementos pertenecientes al juego. En el primero de estos apartados se hablará de forma general de la fase de planificación y diseño del juego, centrándose en que cosas se han considerado importantes en cada uno de los elementos y las soluciones utilizadas. Por otra parte, la fase de desarrollo se enfoca en la implementación del videojuego en Unity, explicando con exactitud como se ha puesto en práctica todo lo mencionado en el apartado de diseño.
- A continuación tendremos el apartado de **pruebas**. Como última fase del desarrollo del juego se presentará a compañeros que lo probarán y darán su opinión, además de aportar *feedback* sobre aquellas cosas que menos convenzan o que sea necesario mejorar. Todos estos comentarios, así como las mejoras realizadas a partir de ellos, se presentarán en esta sección.
- En el siguiente apartado se presentarán las conclusiones a las que se han llegado durante el desarrollo del videojuego y la escritura de la práctica.
- Por último, se hablará la **relación con los estudios** y los **trabajos futuros**, donde se especificarán mejoras que añadiríamos en caso de contar con más tiempo.

1.5 Colaboraciones

Este trabajo se ha hecho en colaboración con dos compañeros, habiendo hecho cada uno diferentes componentes del juego hasta llegar al conjunto presentado. A continuación veremos una lista con cada uno de los componentes del grupo y el trabajo realizado por cada uno de ellos:

- **Daniel De Castro Isasi:** Inteligencia artificial de los enemigos y jefe final, movimiento y comportamiento del protagonista, combate, iluminación, funcionamiento de los objetos y gestión de las animaciones.
- **Marcos Calero Domínguez: Planificación, desarrollo y mantenimiento de un videojuego de sigilo** Encargado principal de las interfaces e inventario, gestión de datos persistente, interacciones con los escenarios, cuadros de texto y sonido.
- **Eva Vidal García: Implementación y desarrollo de un videojuego de sigilo con infraestructura multijugador.** Sistema *online*.

Además, hay ciertos aspectos del videojuego que han hecho mediante la colaboración de los diferentes integrantes, como podría ser el diseño de los escenarios.

CAPÍTULO 2
Estado del arte

En este capítulo se hablará sobre la historia de los videojuegos, desde su comienzo hace varias décadas hasta la actualidad y los avances que han surgido en los últimos años. Más adelante nos centraremos en aquellos géneros en los que se podría clasificar el desarrollado para este trabajo.

2.1 El videojuego y sus inicios como medio de entretenimiento

Actualmente, la industria del videojuego es el sector del entretenimiento que más dinero mueve, hasta el punto de generar más beneficios que el cine y los deportes juntos en EEUU en el año 2020. Este dato, aunque aumentado debido a la pandemia de los últimos años, saca a relucir la importancia que tienen actualmente, pero no siempre fue así.

El origen de los videojuegos se remonta a la década de los 50, presentándose el considerado primer videojuego **Nim** [2] en la feria mundial de Nueva York del año 1940. Fue jugado por cerca de 50.000 personas en los seis meses que estuvo disponible, pero no fue hasta el año 1972 cuando se comercializó la primera consola. Esta consola, cuyo prototipo llamaron *The brown box* y que acabó comercializándose como *Magnavox Odyssey* en EEUU, donde fue lanzada al mercado por Philips y fue un éxito en ventas pese a ser descontinuada en tan solo tres años, llegando a vender alrededor de 300.000 consolas. En España se le llamó *Overkai* y no llegó hasta dos años más tarde, en 1974. *Overkai* tuvo un total de 28 juegos, entre los cuales se encontraba el famoso **Pong** [3].

Una de las empresas que más ayudaron al surgimiento de los videojuegos fue **Atari**, fundada en 1972. Esta compañía fue pionera en los juegos *arcade* y se mantuvo como la empresa más importante relacionada con los videojuegos hasta mediados de la siguiente década, cuando más de 15 compañías empezaron desarrollando juegos de forma independiente.

Fue también Atari quien popularizó los videojuegos como *hobby* doméstico al lanzar al mercado la *Atari VCS*, también conocida como *Atari 2600* en 1977. Esta consola surgió al mercado con 10 juegos simples, pero con la posibilidad de insertar *cartuchos ROM*, lo cual fue aprovechado por desarrolladores de todo el mundo. La consola vendió por debajo de las expectativas de la compañía hasta tres años después, en 1980, con el lanzamiento del juego **Space Invaders**, que revitalizó e incrementó hasta nuevos límites las ventas de la consola.

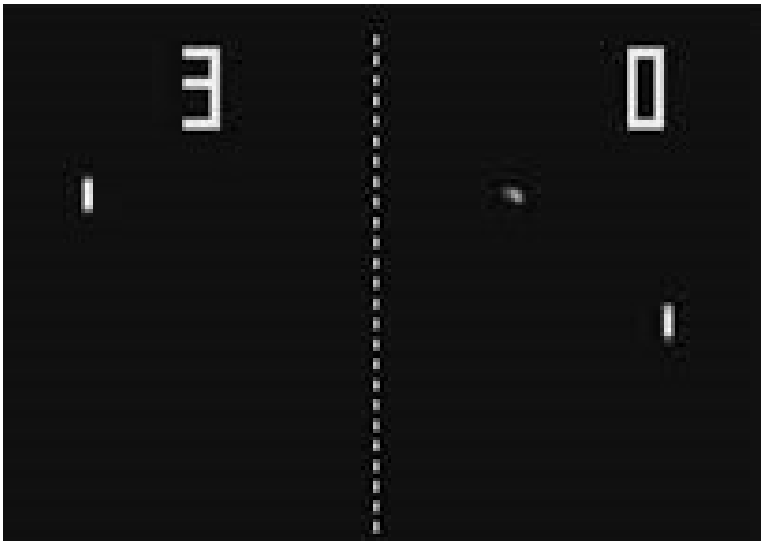

Figura 2.1: Pong

2.2 Importancia de los videojuegos en el mundo contemporáneo

Una vez presentados los orígenes de los videojuegos como industria y las empresas que ayudaron a que sean tan notables como lo son actualmente, nos centraremos en la importancia de los mismos en los últimos años y, principalmente, durante la pandemia y el confinamiento de 2020.

Durante estos primeros meses de pandemia en los que no se permitía a la gente salir de sus casas, fue necesario buscar otras vías de comunicación y conectar con otras personas. Como podemos ver en algunos artículos como *Video Games Help People to Connect and Engage During COVID-19* [4], es en este momento en el que los videojuegos toman una importancia mucho mayor que la que tienen hasta este momento, pues mucha gente recurre a ellos como alternativa para estos impedimentos. Estos ayudan a muchas de estas personas, ya sea mediante juegos puramente multijugador, que permiten jugar con otra gente, o por el sentimiento de cooperación al compartir las experiencias personales mediante redes sociales.

Además. en un estudio realizado por la **Universidad de Oxford** [5] durante el segundo confinamiento nacional de Reino Unido, se establecía que las experiencias de competición y conexión social aportadas por los videojuegos observados por el estudio, contribuían al bienestar de aquellas personas que los jugaban, actuando positivamente en la salud mental de las mismas.

2.3 El sigilo en los videojuegos

Un juego de sigilo es aquel en el que se insta al jugador a atacar a los enemigos sin ser visto o incluso evitarlos por completo. En estos juegos suele existir la opción de esconderse, agacharse, usar disfraces o jugar con la iluminación de los niveles como medidas para evitar ser descubierto por los enemigos y, a diferencia de los juegos de acción más tradicionales, dan muy pocas oportunidades al jugador en un enfrentamiento directo contra ellos, instándole a buscar otras formas de resolver los conflictos utilizando aquellas herramientas de las que disponga en cada momento. Este tipo de juegos hacen énfasis en

un buen diseño de niveles, una inteligencia artificial con mayor capacidad de reacción y la variedad de herramientas dadas al jugador.

El juego considerado como **primer juego de sigilo** fue **Shoplifting Boy**, publicado en el año 1979, que consistía en robar todo lo posible en una tienda, evitando las zonas de visión de los enemigos para evitar que te atrapase la policía. 2 años después sale al mercado **Castle Wolfenstein**, donde el jugador debía atravesar un castillo lleno de guardias con el objetivo de conseguir los planes secretos de los enemigos y escapando sin ser visto. Otro de los primeros y más importantes juegos de sigilo fue el primer **Metal Gear**, desarrollado por Konami y dirigido por Hideo Kojima, una de las personalidades más importantes de la industria incluso actualmente. Este juego fue el primero en llevar el género del sigilo a una audiencia más amplia dominada por los videojuegos de acción.

Este género ha ido evolucionando y cogiendo importancia con el paso de los años, convirtiéndose en uno de los más importantes actualmente con referentes como la saga **Splinter Cell**, que ya cuenta con siete entregas publicadas por Ubisoft. Otra de las sagas modernas que más éxito ha tenido, tanto comercialmente como en lo referente a la crítica, ha sido **Dishonored**, con dos entregas principales. Esta saga se caracteriza por un diseño de niveles excelente y el gran abanico de posibilidades que da al jugador a la hora de enfrentar cualquier situación.

Figura 2.2: Shoplifting Boy (1979) y Dishonored (2012)

2.4 Videojuegos en línea

Durante los primeros años de los videojuegos, se vio el surgimiento de algunos juegos que tenían la posibilidad de jugarse en multijugador con al menos una persona más. Este es el caso de juegos de los que se ha hablado anteriormente como el **Pong**. Uno de los avances más importantes de cara al comienzo de los juegos con multijugador en línea fue el origen de la red PLATO en 1960. Utilizando esta red empezaron a surgir los primeros juegos online, siendo el primero **SpaceWars!** [6] en 1962, un juego que permitía jugar con otra persona más a través de dicha red. A partir de este momento fueron surgiendo varios juegos que aprovechaban la tecnología aportada por PLATO, como fueron **Empire**, un juego de estrategia para hasta ocho jugadores, o **Spasim**, un juego multijugador con hasta 32 jugadores, considerado como el primer juego 3D multijugador de disparos en primera persona. Estos dos juegos surgieron en la primera mitad de la década de los 70.

En 1979 es creado el primer videojuego basado en los juegos de rol de mesa **Dungeons and Dragons** por un grupo de estudiantes universitarios, surgiendo a partir del mismo los primeros juegos MUD o *Multi-User Dungeons // Multi-User Domains*. Estos fueron los primeros juegos multijugador de rol, principalmente en formato texto, en los cuales los jugadores podían interactuar con ellos o el entorno mediante lenguaje natural. Esto dio paso a las primeras comunidades virtuales, aunque no fue hasta los años 90 cuando se

popularizaron este tipo de juegos debido a una mayor disponibilidad del Internet. Algunos de los juegos más importantes que surgieron en esta época fueron el **Starcraft** en 1998 o el **Counter-Strike** en 1999. Además, gracias a esta popularización surgió un nuevo género dentro de los videojuegos multijugador denominados MMO, cuyas siglas significan *Massive Multiplayer Online*, juegos pensados para juntar a una gran cantidad de jugadores dentro de un mismo mundo virtual. Algunos de los más influyentes en estos primeros años fueron el **Ultima Online** de 1997 y el **Everquest** en 1999.

Incluso con el surgimiento de estos nuevos géneros y algunos de los juegos más influyentes de los mismos, no fue hasta la siguiente década, a partir del año 2000, cuando se expandió la popularidad de los juegos multijugador gracias a la reducción considerable del coste de servidores. Es además en este momento cuando surge el juego que llevó este género a todos los públicos, **World of Warcraft**, juego que a día de hoy sigue considerándose uno de los *MMO* más influyentes y con una mayor cantidad de jugadores.

Figura 2.3: SpaceWars! (1962) y World of Warcraft (2004)

2.5 Motores gráficos más utilizados

Uno de los apartados más importantes a la hora de desarrollar un videojuego es la elección del motor a utilizar. Un motor de videojuegos es una aplicación diseñada específicamente con el propósito de facilitar el proceso de producción de dichos juegos, proporcionando una interfaz gráfica y acceso a librerías importantes para dicho desarrollo. Además, son muy útiles en caso de querer reutilizar o adaptar parte de juegos hechos previamente para producir unos nuevos, economizando el proceso.

Además, muchos de los motores de videojuegos utilizados presentan un apartado gráfico en el que ver una *renderización* previa del juego en la fase de desarrollo, sistemas de sonido, físicas y funciones para disponer de una inteligencia artificial.

A continuación se presentarán varios de los motores con más renombre dentro de la industria del videojuego, utilizados para juegos de todo tipo y presupuesto, desde aquellos *AAA* hechos por cientos de personas hasta el más pequeño juego *indie* hecho por dos o tres personas.

- **Unreal Engine:** Es uno de los motores más utilizados en la actualidad. La primera versión del motor surge en 1998 y desde entonces ha ido mejorando y ampliando sus funcionalidades hasta llegar a la quinta versión del mismo, lanzada al mercado el día 5 de abril de 2022. En un principio el motor estaba destinado a *shooters* en primera persona para PC e hizo su debut junto con el juego **Unreal**, aunque desde entonces se ha utilizado para una amplia variedad de géneros y consolas, así como móviles y dispositivos de realidad virtual. Para la programación utiliza el lenguaje

C++. Algunos juegos reconocidos desarrollados con este motor van desde pequeños *indies* como **Hello Neighbour** o **A hat in time** hasta juegos con millones de dólares en presupuesto como la saga **Bioshock** o **Borderlands**.

Debido a la nueva versión, es un motor muy a tener en cuenta de cara al futuro de la industria. Esta nueva versión incluye una fidelidad gráfica superior a la de sus competidores hasta la fecha de realización de este trabajo. Esto es principalmente debido a la adición de dos nuevos sistemas: **Nanite**, que permite cargar millones de polígonos en escena sin disminuir el rendimiento, y **Lumen**, un nuevo sistema de iluminación dinámica que mezcla los sistemas tradicionales y nuevas técnicas de *raytracing* que no requieren de un cálculo tan costoso.

- **Unity:** Junto con Unreal Engine, es el motor más utilizado a la hora de hacer videojuegos de todo tipo y estilos. Unity surge en 2005 dedicado plenamente al desarrollo de videojuegos en 3D y no es hasta 2013 cuando añade soporte oficial para el 2D, aunque previamente los desarrolladores podían conseguir un "falso"2D mediante técnicas poco convencionales como el añadir texturas a paneles planos. Al igual que con Unreal Engine, se puede ver la gran versatilidad del motor con algunos ejemplos de videojuegos desarrollados en el mismo, como **Hollow Knight**, **Cuphead** o **Rust**. Más adelante se profundizará en este motor y todas sus ventajas y posibilidades.

- **Motores propios:** Por otro lado, una gran parte de empresas, en especial aquellas más grandes de la industria, desarrollan sus juegos basándose en motores propios que van actualizando a lo largo del tiempo, teniendo de esta manera un control total del mismo. Una de las principales razones para decantarse por el software desarrollado en la empresa podría ser abaratar el coste de producción, ya que un mismo motor o variaciones del mismo se puede utilizar para diversos juegos y se elimina la necesidad de pagar un porcentaje del dinero generado por los juegos a la empresa propietaria de los motores (sistema de monetización utilizado por Unreal Engine y Unity entre otros). En esta categoría se encuentran algunos motores como el **Creation Engine** de *Bethesda*, **REDEngine** de *CD Project Red* o **Source Engine** de *Valve*

Además de estos, también existen otros motores utilizados principalmente en la escena independiente debido a factores como la facilidad y accesibilidad que presentan o por ajustarse más a las exigencias de los desarrolladores. Estos podrían ser algunos motores menos reconocidos, como **Godot**, un motor gratuito y de código abierto desarrollado en su totalidad por la comunidad, siendo actualizado acorde a las exigencias generales de los desarrolladores que lo utilizan, o **RPG Maker**, un motor cuya mayor utilidad es permitir el desarrollo de videojuegos relativamente complejos sin necesidad de saber programar.

Figura 2.4: Unreal Engine: Bioshock y REDEngine: The Witcher 3

CAPÍTULO 3
Análisis del problema

En el capítulo siguiente se dará una visión general del proyecto realizado, estableciendo los diferentes casos de uso que nos ayudarán con el posterior desarrollo del juego, así como los requisitos que tendrá que cumplir, tanto funcionales como no funcionales. A continuación se dará una breve explicación de la idea central del videojuego, así como los métodos seguidos durante dicho desarrollo, hablando de la planificación y los tiempos reales que se han llevado a cambio.

3.1 Casos de uso

Previamente, al comienzo del desarrollo del videojuego, se han realizado un grupo de casos de uso utilizados para exponer las funcionalidades que deberán estar presentes en el mismo. Estos casos de uso harán referencia a las acciones que el usuario podrá realizar desde el menú principal (Figura 3.1), desde el menú de pausa (Figura 3.2), las acciones que podrá hacer el protagonista del juego (Figura 3.3) y aquellas que podrán realizar los enemigos (Figura 3.4). A continuación veremos estos casos de uso, así como la explicación de cada una de las opciones presentes en ellos.

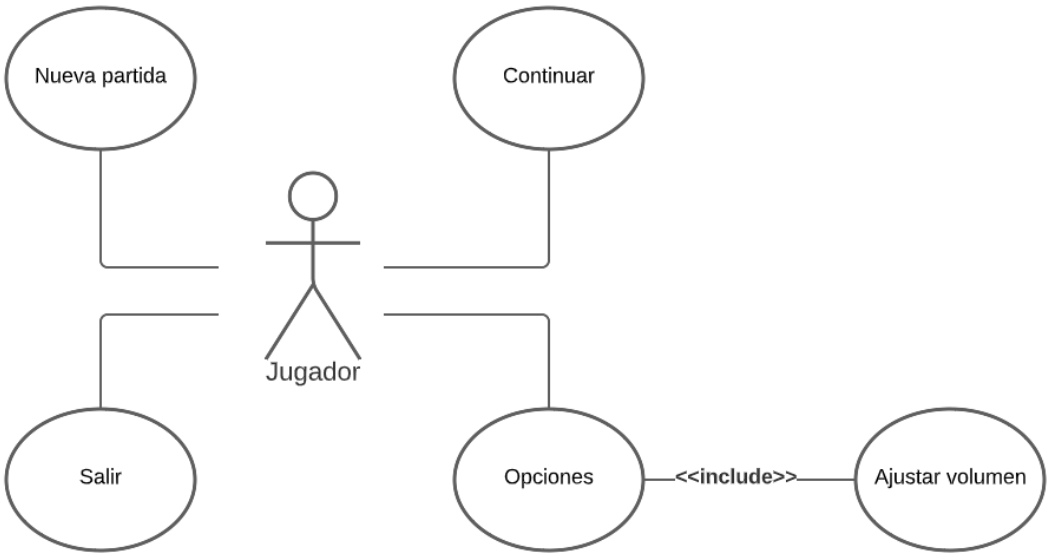

Figura 3.1: Caso de uso Menú principal

- **Nueva partida** : El jugador genera una nueva partida, empezando desde el principio y eliminando los datos guardados anteriormente.
- **Continuar** : El usuario comienza desde el último nivel guardado. Si no contiene datos anteriores, se iniciará una nueva partida desde el principio.
- **Salir** : El usuario se desconecta del juego.
- **Opciones** : Permite al usuario entrar en las opciones. De esta forma podrá cambiar en nivel de volumen general de la aplicación mediante el botón de **Ajustar volumen**.

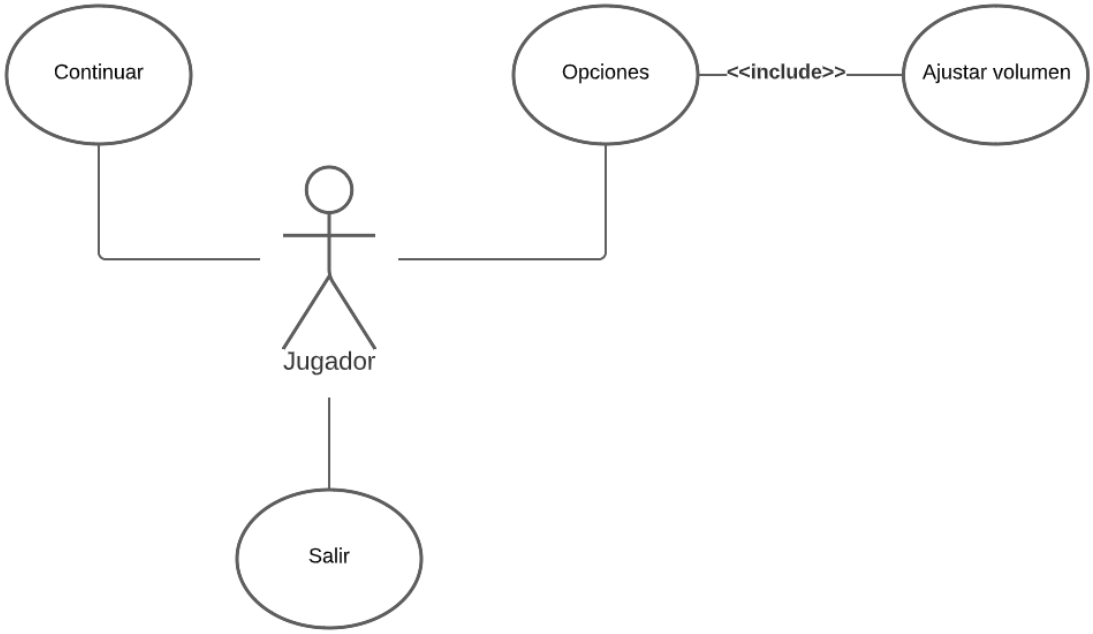

Figura 3.2: Caso de uso Menú pausa

- **Continuar** : El jugador cierra el menú de pausa y continua con la partida desde donde se había quedado.
- **Opciones** : El usuario entra en el apartado de opciones, siendo esta la única forma de acceder al botón **Ajustar volumen** y nivelar el volumen de la aplicación.
- **Salir** : El usuario vuelve al menú principal.

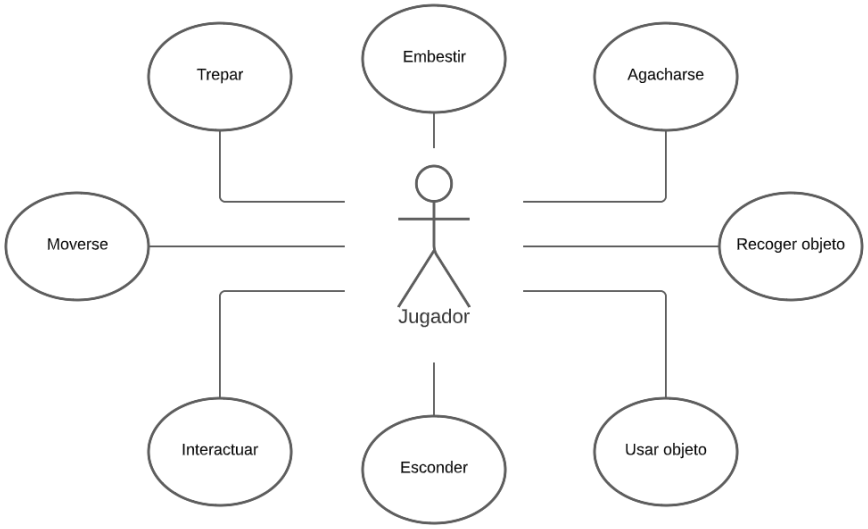

Figura 3.3: Caso de uso Protagonista

- **Moverse** : El protagonista se desplaza por el nivel
- **Trepar** : El jugador es capaz de trepar por las escaleras de mano repartidas por el nivel
- **Embestir** : Realiza un desplazamiento veloz, embistiendo a los enemigos a su paso.
- **Agacharse** : El personaje principal se agacha, haciendo que su movimiento sea más lento, pero también más difícil de detectar por los enemigos
- **Recoger objeto** : Los objetos cercanos al jugador se meterán automáticamente en su inventario.
- **Usar objeto** : Será posible utilizar los objetos del inventario si se han seleccionado previamente. La función realizada al usar un objeto dependerá de cuál esté seleccionado en ese momento.
- **Esconder** : El protagonista es capaz de esconderse detrás de las cortinas presentes en el nivel con el objetivo de no ser detectado.
- **Interactuar** : El protagonista será capaz de interactuar con diversos elementos del escenario marcados con una pista visual al estar cerca.

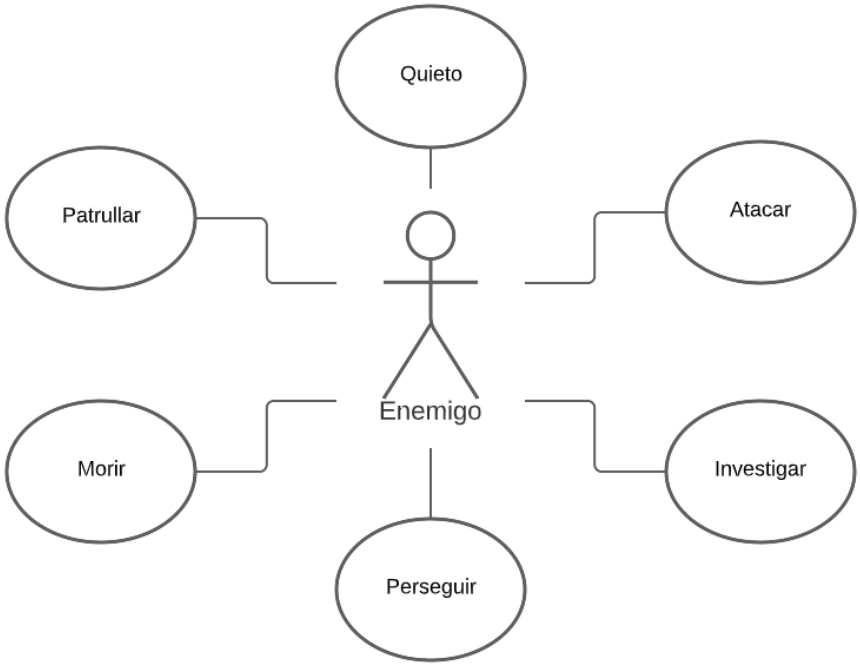

Figura 3.4: Caso de uso Enemigo

- **Perseguir** : El enemigo camina en dirección al jugador.
- **Morir** : Se produce cuando la vida del enemigo se reduce a cero.
- **Patrullar** : El enemigo se desplaza por el mapa en línea recta, variando el sentido cuando sea necesario.
- **Quieto** : El enemigo se mantiene inmóvil.
- **Atacar** : Realiza un ataque que daña al jugador en caso de estar en el rango adecuado.
- **Investigar** : El enemigo se dirige hacia la ubicación donde se encuentra un objeto a investigar.

3.2 Especificación de requisitos

En la siguiente sección se especifican los requisitos presentes en el proyecto, diferenciando entre funcionales y no funcionales.

Requisito funcional 1	
Nombre	Nueva Partida
Descripción	El jugador debe ser capaz de empezar una partida nueva en el momento deseado, generando nuevos datos y eliminando aquellos guardados anteriormente

Requisito funcional 2	
Nombre	Continuar partida
Descripción	Si el jugador ha completado un nivel, podrá continuar desde el siguiente, aunque haya salido del juego, manteniendo su progreso

Requisito funcional 3	
Nombre	Volver al menú
Descripción	El jugador podrá en cualquier momento volver al menú principal desde el menú de pausa, al morir o al acabar un nivel

Requisito funcional 4	
Nombre	Salir del juego
Descripción	El jugador podrá salir mediante el menú principal

Requisito funcional 5	
Nombre	Menú de pausa
Descripción	Durante un nivel, el jugador podrá acceder al menú de pausa en cualquier instante, congelando el juego temporalmente

Requisito funcional 6	
Nombre	Morir
Descripción	En el caso en el que la vida del protagonista descienda a 0, el juego se pausará y mostrará el menú de muerte

Requisito funcional 7	
Nombre	Recoger objeto
Descripción	El jugador deberá ser capaz de añadir objetos a su inventario al acercarse a ellos cuando están en el suelo del nivel

Requisito funcional 8	
Nombre	Usar objeto
Descripción	El jugador podrá utilizar un objeto mientras este presente en su inventario y lo tenga seleccionado

Requisito funcional 9	
Nombre	Atacar
Descripción	El jugador podrá realizar un ataque mientras tenga el objeto 'Espada' equipado desde su inventario

Requisito funcional 10	
Nombre	Embestida
Descripción	El jugador podrá realizar una embestida siempre que la acción esté disponible, no pudiéndose utilizar de forma continuada sin esperar unos segundos

Requisito funcional 11	
Nombre	Abrir cofre
Descripción	Si el jugador se encuentra en una posición cercana a un cofre deberá poder abrirlo

Requisito funcional 12	
Nombre	Interactuar con palancas
Descripción	Si el jugador se encuentra en una posición cercana a una palanca, podrá accionarla

Requisito funcional 13	
Nombre	Siguiente nivel
Descripción	El juego debe permitir acceder al nivel siguiente una vez se haya llegado al final de aquel en el que el jugador se encuentre

Requisito funcional 14	
Nombre	Guardar progreso
Descripción	Al completar el nivel se deberá guardar la información del personaje, de forma que se mantengan los objetos que este tenga en el inventario de cara a la carga del siguiente nivel. Estos datos guardados deben mantenerse en caso de cerrar el juego para poder continuar la partida

Requisito funcional 15	
Nombre	Sonidos
Descripción	Deberán existir sonidos representativos para las acciones principales del juego, los cuales se reproducirán en los momentos oportunos

Requisito funcional 16	
Nombre	Movimiento
Descripción	Tanto el jugador como los enemigos deben ser capaces de desplazarse por el escenario

Requisito funcional 17	
Nombre	Mostrar rango de lanzamiento
Descripción	En el caso de objetos concretos, se deberá representar gráficamente la trayectoria del lanzamiento previamente a usar el objeto

Requisito no funcional 1	
Nombre	Corrección funcional
Descripción	El videojuego deberá ser capaz de funcionar en cualquier ordenador

Requisito no funcional 2	
Nombre	Accesibilidad
Descripción	Aquellas personas menos experimentadas en los videojuegos deberían ser capaces de disfrutar con él, por lo que las mecánicas del juego deberán ser sencillas y fácilmente entendibles

Requisito no funcional 3	
Nombre	Disponibilidad
Descripción	La aplicación se encontrará en funcionamiento mientras el jugador no cierre su ejecución

3.3 Explicación del videojuego

La idea principal del videojuego presentado consiste en superar un grupo de niveles, llegando hasta el final de cada uno de ellos, controlando a la guerrera que hace la función de protagonista. Por el camino, el jugador se encontrará con enemigos que harán lo posible para acabar con el usuario, intentando matarle antes de que cumpla su objetivo.

El juego se clasifica dentro del género de plataformas 2D, teniendo una perspectiva en dos dimensiones y *Scroll lateral* donde la cámara se moverá junto al protagonista, enseñando una zona limitada del nivel acorde a lo que la guerrera podría ver en cada momento. Como en el resto de juegos de este género, el jugador deberá saltar, trepar y combatir para poder llegar al fin de cada uno de los niveles. En total habrá tres niveles, siendo el primer nivel una experiencia cercana a un tutorial y a partir de ahí aumentando de tamaño y dificultad hasta terminar con un jefe final en la última parte del tercer nivel.

Además, como se ha explicado anteriormente, el juego tendrá un gran componente de sigilo. Es por ello que el jugador contará con varios objetos durante el transcurso de los niveles que serán utilizados para evitar a los enemigos en la medida de lo posible. Algunos de estos objetos son un señuelo para atraerles o una bola de agua utilizada para apagar antorchas y evitar de esta forma ser detectado. Además, el escenario también estará diseñado acorde a esta idea, permitiendo al jugador esconderse detrás de cortinas para esconderse.

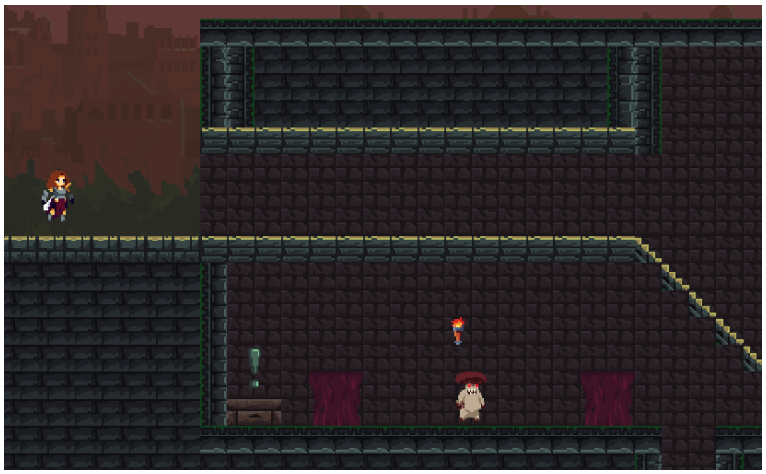

Figura 3.5: Inicio del juego

3.4 Plan de trabajo

En esta última sección, antes de pasar al diseño del videojuego se discutirá la aplicación realizada de la metodología ágil y en concreto del modelo *scrum* para la producción de nuestro proyecto. Este proceso se ha dividido en tres *sprints* principales, dando un mes para la realización de cada uno de ellos. También se ha añadido un primer *Sprint 0* además de los mismos, el cual está enfocado a la preproducción del juego. Esto engloba varias sesiones de *brainstorming*, el desarrollo de las diferentes *UT* para el *backlog* y la realización de un primer *diagrama de Gantt* con los tiempos estimados para los objetivos especificados en este mismo *sprint*. Por último, antes de explicar lo hecho en cada una de las iteraciones, es importante destacar que a la hora del desarrollo se ha utilizado un sistema de prototipos. Con esto se quiere decir que en vez de desarrollar el primer nivel e ir realizando cada una de las mecánicas conforme han ido siendo necesarias, se ha partido de un diseño muy básico a partir del cual se han ido implementando cada una de estas mecánicas por separado, juntándolas todas más adelante cuando se llega al funcionamiento deseado.

A continuación se presenta la separación realizada por *sprints* en el desarrollo del juego:

- **Sprint 0:** Previamente al desarrollo del proyecto, se añadirá este primer *sprint* auxiliar en el cual se realizarán varias tareas importantes. La primera de ellas consistirá en una lluvia de ideas o *Brainstorming* entre los diferentes integrantes del grupo sobre la temática y mecánicas que se quieren añadir al videojuego. De esta forma es posible planificar el proyecto con antelación, evitando posibles problemas que pudiesen surgir en un futuro. El siguiente paso será separar estas ideas y mecánicas en diferentes tareas pequeñas en las que dividir el proyecto. Estas tareas ayudarán a la separación del trabajo, así como la planificación del mismo. Por último, se desarrollará un *Diagrama de Gantt* por medio del cual se establecerán los tiempos estimados para cada una de dichas tareas, organizando el proyecto en su totalidad.

- **Sprint 1:** En este *sprint* se comenzará el desarrollo del videojuego propiamente dicho. Este desarrollo se organiza mediante el uso de prototipos, de forma que se pudiesen probar cada una de las mecánicas por separado previamente a su implementación dentro del juego y los niveles como tal. Para ello, en este sprint se desarrollará un escenario básico en el que poder desarrollar y comprobar el funcionamiento de estas nuevas funcionalidades. Durante el transcurso de este sprint nos centraremos en el movimiento del jugador, ya que será la parte central para poder implementar el resto del juego. Una vez hecho esto, se programará el uso de escondites, así como el uso de algunos de los objetos que estarán disponibles para el jugador a lo largo del juego. Una vez realizadas esto, se pasará a diseñar algunos de los elementos más importantes del juego, como pueden ser las máquinas de estados en las que se basará el comportamiento de los enemigos comunes. Por último, y en conjunto con los otros dos integrantes con los que se ha desarrollado el videojuego, se diseñará e implementará el primer nivel, el cual hará la función de tutorial y presentará las diferentes mecánicas que el jugador podrá realizar. En este sprint tan solo se implementará el propio escenario, dejando la lógica y enemigos del nivel para más adelante conforme se vayan desarrollando los mismos.

- **Sprint 2:** En este *sprint* se llevará a cabo la mayoría de la programación e implementación del videojuego. Lo primero en lo que nos centraremos será en la iluminación que se utilizará a lo largo de los diferentes niveles que se implementen. Teniendo este objetivo en mente, al igual que con lo visto en el primer sprint, se utilizará

un escenario básico en el que realizar las diferentes pruebas que sean necesarias e implementar de forma exacta lo que se pretende tener más adelante en los escenarios del propio juego. Una vez acabada la iluminación, se continuará por la parte central de este trabajo, lo cual será la implementación de los enemigos. Para ello se utilizarán las máquinas de estados diseñadas anteriormente y se pondrán en práctica, programando cada uno de los diferentes comportamientos que deberán tener los mismos, así como las transiciones entre ellos. Una vez se tengan unos agentes básicos y funcionales, se implementará el sistema de combate, así como las funcionalidades de los objetos ligados al mismo, como puede ser la espada. También será en este momento cuando se añadirá la funcionalidad al señuelo, ya que no era posible hacerlo hasta tener a los agentes funcionales. Por último, acabaremos este *sprint* diseñando e implementando el primer nivel real, así como añadiendo el apartado visual tanto al jugador como a los enemigos, añadiendo los *sprites* y animaciones correspondientes para las funcionalidades programadas hasta el momento.

- **Sprint 3:** Empezaremos esta última etapa del desarrollo creando un último nivel para el juego, al final del cual se añadirá un jefe final. Para ello será necesario también replicar los pasos realizados con el resto de enemigos y aplicárselos. Estos serán diseñar e implementar una nueva máquina de estados, programar dichos estados y sus transiciones, y añadir el apartado visual al enemigo. Además, será a mitad de esta etapa cuando gente externa al proyecto probarán el juego, dándonos información de diferentes errores, así como mejoras que aplicar en el juego final.

Para organizar los *sprints* de forma sencilla se ha utilizado **Trello**, una plataforma *online* utilizada para la administración de proyectos mediante la colocación de tarjetas en diferentes listas que conforman un tablero. Cada una de estas tarjetas representa una tarea diferente a realizar, pudiendo moverlas de lista con facilidad. Para la organización de este proyecto se ha organizado de forma que todas las tareas comenzaban en la lista de *backlog*. A partir de ahí, aquellas que corresponden con el *sprint* actual se moverán a una nueva lista, de la que saldrán tan solo cuando se hayan completado o, en alguno de los casos, descartado por múltiples motivos. En la figura 3.6 podemos ver la separación de los diferentes *sprints* mediante listas en la plataforma de *trello*.

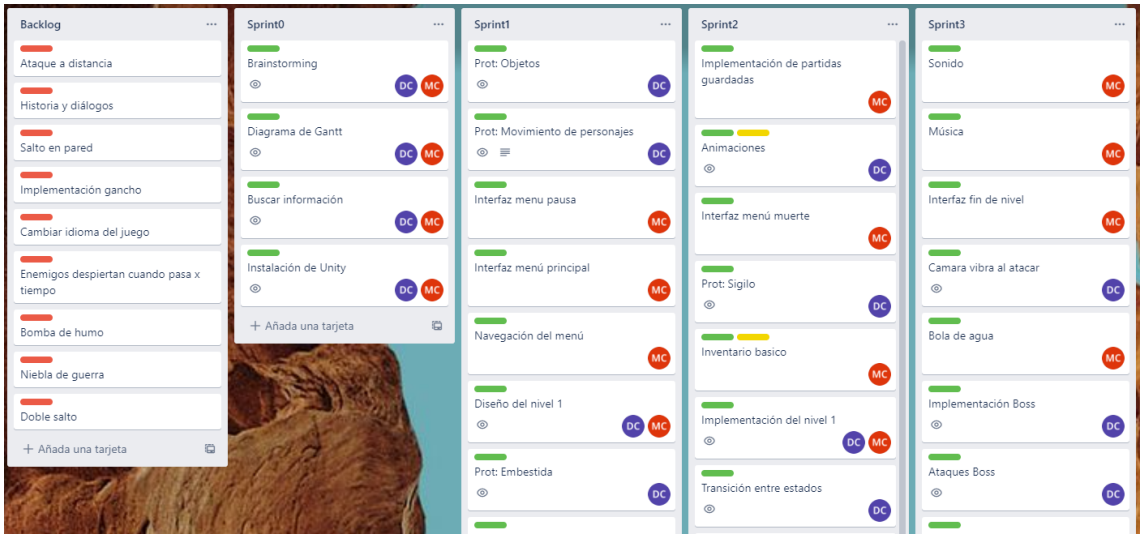

Figura 3.6: Sprints en el desarrollo del proyecto

Además, para cada una de estas tarjetas (figura 3.7) será posible, aparte de cambiar la lista en la que se sitúan, añadir una descripción, comentarios, fecha de realización y etiquetas que las definen, entre otras opciones. En el caso de este proyecto, las etiquetas se han utilizado de la siguiente forma:

- **Verde:** La tarea ha sido completada y ha pasado todas las pruebas establecidas.
- **Amarillo:** La tarea ha sido completada, pero ha sido necesario realizar diversos cambios debidos a las pruebas realizadas.
- **Naranja:** La tarea continua en desarrollo.
- **Rojo:** Se ha producido un error en las pruebas que es necesario solucionar en caso de querer continuar con la tarea.

Figura 3.7: Ejemplo de tarjeta de Trello

En la figura 3.8 podemos observar la planificación exacta en la que se han organizado estos *sprints* previamente mencionados. Esta planificación corresponde a los tiempos reales que se han necesitado para cada una de las actividades, los cuales varían ligeramente respecto a aquellos previstos al inicio, en los cuales además se incluían algunas de las tareas que han sido descartadas debido a varios problemas imprevistos o por decisión de los participantes en el proyecto. Este es el caso por ejemplo del gancho, funcionalidad que se quiso implementar, pero se acabó eliminando debido al diseño final de los niveles.

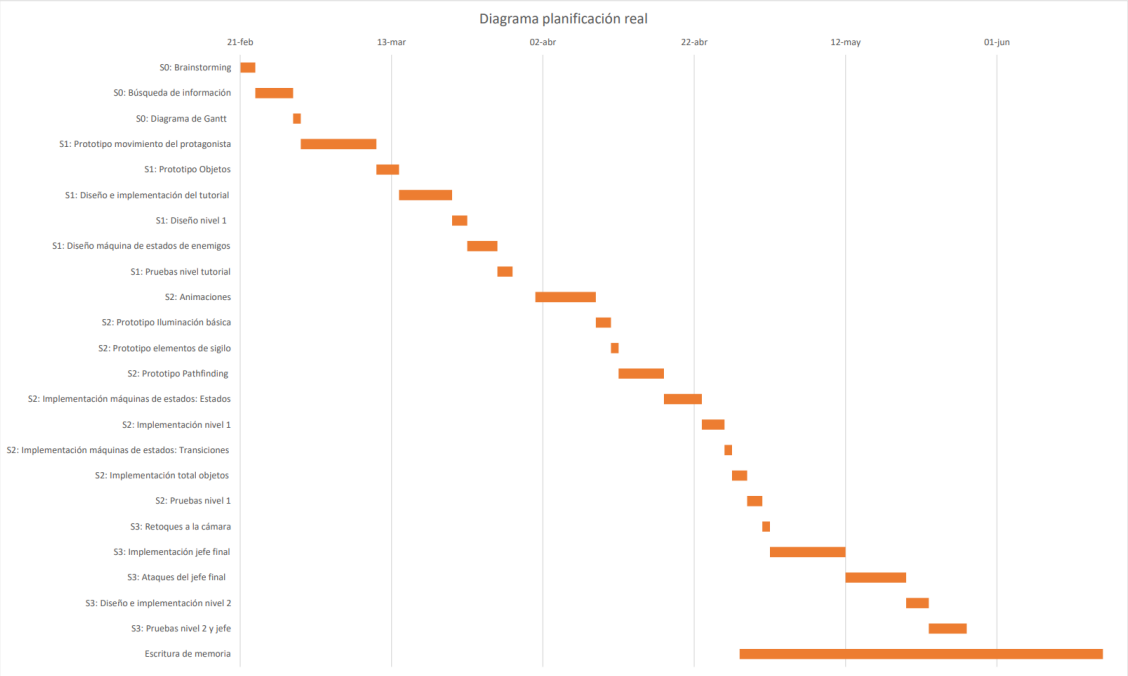

Figura 3.8: Planificación real

	Daniel de Castro	Marcos Calero
Sprint 0		
Brainstorming	X	X
Búsqueda de información	X	X
Diagrama de Gantt	X	X
Sprint 1		
Diseño de objetos	X	
Prototipo movimiento del protagonista	X	
Prototipo de objetos	X	
Interfaz menú principal		X
Interfaz menú de pausa		X
Diseño e implementación del nivel de tutorial	X	X
Diseño de nivel 1	X	X
Diseño de máquinas de estados de enemigos	X	
Pruebas de nivel de tutorial	X	X
Sprint 2		
Implementación de partidas guardadas		X
Animaciones	X	
Prototipo de iluminación básica	X	
Prototipo elementos de sigilo	X	
Inventario		X
Interfaz menú de muerte		X
Prototipo de *Pathfinding*	X	
Implementación de máquinas de estados: Estados	X	
Implementación nivel 1	X	X
Pruebas nivel 1	X	X
Sprint 3		
Retoques a cámara	X	
Interfaz de fin de nivel		X
Implementación del jefe final	X	
Ataques jefe final	X	
Interacción bola de agua con el entorno		X
Zonas secretas		X
Palancas		X
Interfaz del menú victoria		X
Música y sonido		X
Diseño e implementación del nivel 2	X	X
Pruebas del nivel 2 y jefe final	X	X

Tabla 3.1: Reparto de tareas

CAPÍTULO 4
Diseño de la solución

En esta sección se hablará de cada uno de los componentes presentes en el videojuego, así como la interacción presente entre los mismos. Al ser un trabajo conjunto, aunque se explicarán brevemente todos los elementos del juego, nos centraremos en aquellos pertenecientes a este trabajo.

4.1 Tecnología utilizada: Unity

Anteriormente, se ha hablado de lo importante que es la elección de un motor de videojuegos a la hora de desarrollar un juego y se han expuesto algunos de los más conocidos y ampliamente utilizados por toda la industria. A la hora de decidir que motor utilizar para este proyecto existen varios motivos a tener en cuenta, entre los que destaca el tipo de juego que se quiere desarrollar y la experiencia previa de las personas implicadas en el proyecto. Es por esto que se eligió Unity, un motor que se había utilizado con anterioridad en otra asignatura y con la que ya se había obtenido algo de experiencia externa.

Adicionalmente, actualmente Unity cuenta con varias funcionalidades que facilitan el desarrollo de un videojuego en 2D de este estilo, entre los que destacan la posibilidad de utilizar *Tilemaps* para el diseño de niveles como se explicará más adelante, además de disponer de un sistema de iluminado 2D y los algoritmos de Inteligencia Artificial buscados para el comportamiento de los enemigos.

Por otra parte, uno de los objetivos más importantes que se trataron al dar comienzo al proyecto fue el apartado de la inteligencia artificial. Para ello se quería recurrir tanto a las máquinas de estados para determinar los diferentes comportamientos y las transiciones entre los mismos y la posibilidad de aplicar algoritmos de búsqueda para que los *agentes* busquen al jugador. Para ello, en la fase de investigación se encontró un proyecto que ayudaría bastante en la tarea, aportando las herramientas necesarias para los objetivos pretendidos.

Además, debido al amplio uso que se le da a Unity en el ámbito independiente y como motor con el que empezar a desarrollar videojuegos, se puede encontrar mucha información muy útil sobre cada uno de los temas sobre los que se pudiese necesitar ayuda, ya sea con videos en los que desarrolladores realizan tutoriales basándose en su experiencia o mediante foros o diferentes páginas de documentación [7]. También podemos encontrar muchas respuestas a dudas que otros desarrolladores han podido tener en páginas como *Stack Overflow* [8].

Por último, un gran punto a favor de Unity es el precio de la licencia, siendo esta gratuita hasta alcanzar los 100.000 dólares anuales en beneficios. También cuenta con una

gran comunidad que aporta diferentes herramientas y recursos desde la tienda oficial de Unity (*Unity Asset Store* [9]), siendo muchos de ellos gratuitos.

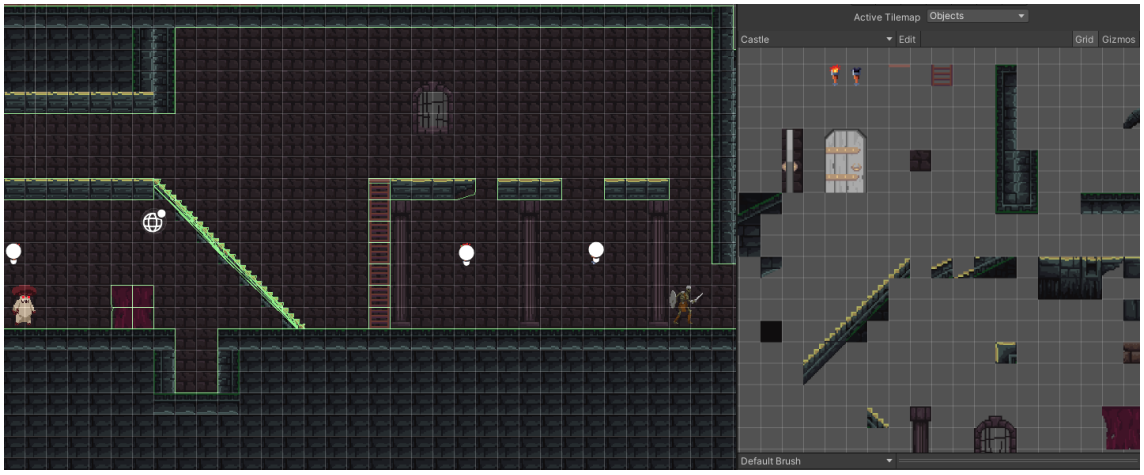

Figura 4.1: Herramienta Tilemaps

4.2 Interfaz

Una de las partes fundamentales de los videojuegos son las interfaces. En ellas se presentan muchas de las opciones al usuario, ya sea desde un menú con el que empezar a jugar o elementos visuales necesarios para entender correctamente el juego. A continuación se mostrará una breve explicación de cada una de estas interfaces y sus posibles transiciones. Primeramente, se hablará del menú principal, el cual se presentará al jugador al abrir el juego. En este menú deberá existir las siguientes transiciones:

Menú principal	
Descripción	Interfaz mostrada al comenzar el videojuego
Transiciones	
Primer nivel	Pulsar el botón Nueva partida
Nivel actual	Pulsar el botón Continuar partida
Menú opciones	Pulsar el botón Opciones
Menú principal	Pulsar el botón Volver
Salir del juego	Pulsar el botón Salir

Tabla 4.1

En cualquier momento el jugador será capaz de pausar el juego pulsando una tecla, abriendo de esa forma el menú de pausa. Esta interfaz tendrá las opciones y transiciones presentes en la tabla siguiente:

Menú pausa	
Descripción	Interfaz mostrada al pausar el videojuego
Transiciones	
Nivel actual	Pulsar el botón Continuar
Menú pausa	Accionar la tecla Escape
Menú opciones	Pulsar el botón Opciones
Menú pausa	Pulsar el botón Volver
Menú principal	Pulsar el botón Salir

Tabla 4.2

De la misma forma, existirá una interfaz de victoria que se lanzará al acabar cada uno de los niveles. Esta interfaz estará compuesta tal y como se indica a continuación:

Menú victoria	
Descripción	Interfaz mostrada al completar un nivel
Transiciones	
Nivel siguiente	Pulsar el botón Continuar
Menú principal	Pulsar el botón Salir

Tabla 4.3

Por último, también existirá una interfaz que surgirá cuando la vida del jugador baje a cero, la cual tendrá sus propias opciones, indicadas en la siguiente tabla:

Menú muerte	
Descripción	Interfaz mostrada al reducir la vida del jugador a cero
Transiciones	
Nivel actual	Pulsar el botón Continuar
Menú principal	Pulsar el botón Volver al menú

Tabla 4.4

4.3 Niveles

Para el desarrollo de niveles se ha utilizado un sistema de cuadrículas y *tilemaps* que han facilitado la edición y actualización de los niveles a lo largo del desarrollo del juego.

Este sistema funciona mediante una interfaz gráfica que posibilita el posicionamiento de los diferentes *tiles* a lo largo de la escena, conformando la totalidad del mapa. Estos elementos se pueden disponer en diferentes capas, permitiendo la presencia de varios en el mismo hueco de la cuadrícula, de forma que se puedan realizar niveles más complejos y recargados.

Para la estética de este videojuego se quería conseguir un ambiente lúgubre y oscuro en el que se pudiese aprovechar tanto jugable como visualmente la iluminación. Para ello se ha hecho uso de un paquete de *assets* encontrado en la tienda de Unity, añadiendo algunos *tiles* propios como los usados para puertas o palancas. El juego está ambientado en el interior de un castillo abandonado con el paso de los años y habitado actualmente por enemigos que querrán deshacerse del jugador en cuanto le vean.

Al pensar en como diseñar los niveles, se concluyó que sería interesante dar la máxima cantidad de opciones al jugador a la hora de afrontar los diferentes retos a lo largo

del juego. Para ello se ofrece al usuario varias formas de enfrentarse a los enemigos o de avanzar en algunos casos, animando a hacer una elección entre la opción más directa pero peligrosa, o evitando enfrentarse a aquellos enemigos a los que no sea necesario.

4.4 Protagonista

Para el diseño del protagonista se buscaba un aspecto visual de aventurero que pudiera defenderse de los peligros del castillo, por lo que se buscó un modelo acorde con estas características que contase con los *sprites* necesarios para poder animar las mecánicas de las que queríamos que dispusiese, liberando una gran cantidad de tiempo que se pudiese dedicar a la parte programática del mismo. De esta forma, el único aspecto visual en el que era necesario centrarse es en el flujo de las animaciones.

Centrándonos en el diseño jugable del personaje, se quiere aportar un abanico de opciones que permitan un movimiento fluido, así como la posibilidad de combatir o evitar a la gran mayoría de enemigos presentes en el juego. Para ello, el jugador podrá realizar las acciones detalladas en el caso de uso presente en el anterior apartado, siempre que se cumplan las condiciones necesarias para las mismas. Esto es, por ejemplo, solo podrá realizar la acción de esconderse cuando se sitúe en un lugar establecido para ello.

Figura 4.2: Sprite del protagonista

4.5 Enemigos y jefe

Los enemigos serán los obstáculos principales para el protagonista, dificultando su avance por los niveles. Para mantener una estética uniforme a lo largo de la experiencia, se ha optado por utilizar enemigos acordes a un juego de fantasía, como pueden ser los esqueletos. Además, se han incluido setas de aspecto humanoide y un jefe final en forma de un monstruo parecido a un murciélago.

A la hora de diseñar la lógica de los enemigos se ha utilizado una máquina de estados, la cual será diferente para los enemigos comunes y el jefe final. Estas máquinas de estados cubrirán todas las posibilidades de comportamiento programadas para los mismos, variando entre ellas reaccionando de forma dinámica al entorno y a las acciones del jugador. A continuación se presentarán las dos máquinas de estados y se explicará con detalle cada uno de los estados presentes en las mismas.

Figura 4.3: Sprites de los enemigos

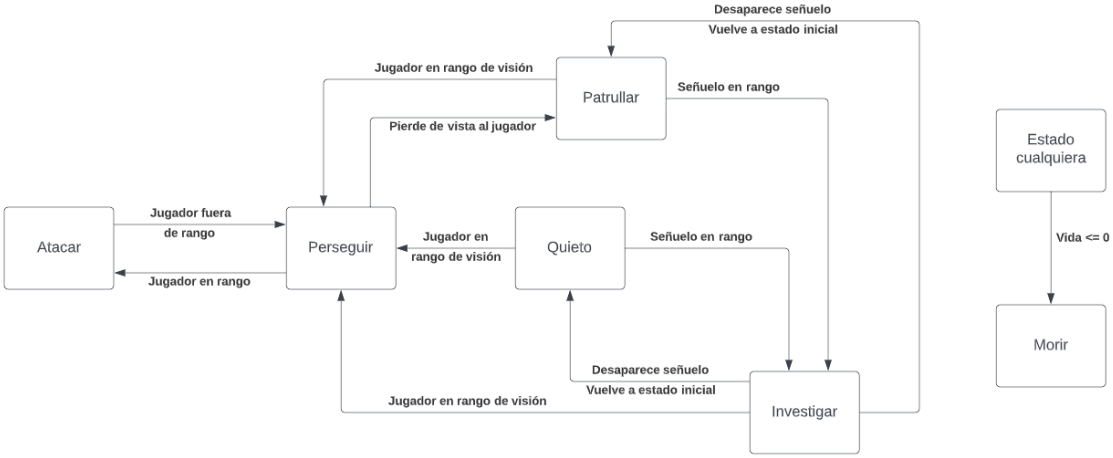

Figura 4.4: Máquina de estados de enemigos comunes

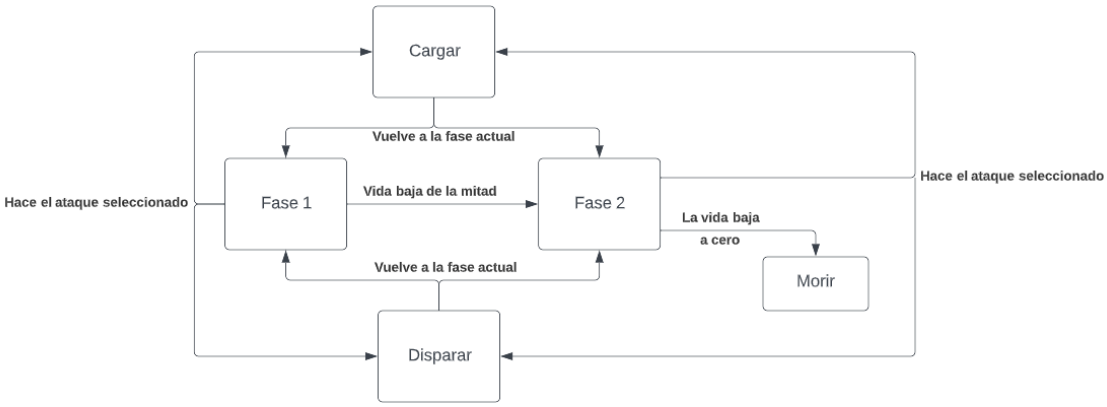

Figura 4.5: Máquina de estados del jefe final

Estado: Patrullar	
Descripción	El agente recorre el espacio que tenga disponible a una velocidad constante. La longitud de la patrulla puede variar, ya que depende del terreno que tenga disponible, variando de forma dinámica según la posición del agente en el mapa
Transiciones	
Señuelo	El agente detecta un señuelo lanzado por el jugador
Perseguir	El jugador entra en el campo de visión del agente
Morir	La vida del agente baja a cero

Tabla 4.5

Estado: Quieto	
Descripción	El agente permanece en su posición esperando estímulos externos
Transiciones	
Señuelo	El agente detecta un señuelo lanzado por el jugador
Perseguir	El jugador entra en el campo de visión del agente
Morir	La vida del agente baja a cero

Tabla 4.6

Estado: Perseguir	
Descripción	El agente sigue al jugador mediante un algoritmo de *pathfinding*
Transiciones	
Patrullar	Pierde de vista al jugador
Atacar	Alcanza al jugador, quedándose a rango de ataque
Morir	La vida del agente baja a cero

Tabla 4.7

Estado: Atacar	
Descripción	El agente se mantiene a rango de ataque del jugador y realiza la acción
Transiciones	
Perseguir	El jugador se aleja del agente y, por tanto, sale del rango de ataque del mismo
Morir	La vida del agente baja a cero

Tabla 4.8

Estado: Señuelo	
Descripción	Mientras que el jugador no está en rango de visión, el agente detecta un señuelo y se mueve hacia él
Transiciones	
Perseguir	El jugador entra en el rango de visión
Patrullar	El señuelo desaparece y el jugador no está en rango de visión
Morir	La vida del agente baja a cero

Tabla 4.9

Estado: Fase 1	
Descripción	El jefe tiene más de la mitad de la vida. Sus ataques son más flojos y fáciles de esquivar y te persigue de forma continua
Transiciones	
Fase 2	El jefe pierde la mitad de la vida
Cargar	Ataque elegido de forma aleatoria
Disparar	Ataque elegido de forma aleatoria

Tabla 4.10

Estado: Fase 2	
Descripción	El jefe ya ha perdido la mitad de la vida. Sus ataques se vuelven más fuertes y difíciles de esquivar. Sigue persiguiendo al jugador de forma continuada
Transiciones	
Cargar	Ataque elegido de forma aleatoria
Disparar	Ataque elegido de forma aleatoria
Morir	La vida del jefe baja a cero

Tabla 4.11

Estado: Cargar	
Descripción	Realiza una embestida a gran velocidad hacia la posición actual del jugador y continua hasta chocar con una pared
Transiciones	
Fase 1	El ataque acaba y el jefe todavía tiene más de la mitad de la vida
Fase 2	el ataque acaba y el jefe tiene menos de la mitad de la vida

Tabla 4.12

Estado: Disparar	
Descripción	Dispara una cantidad variable bolas de energía dependiendo de la fase en la que estuviese el jefe. En caso de ser la fase 2 además podrá realizar un nuevo ataque consistente en dejar caer piedras desde el techo
Transiciones	
Fase 1	El ataque acaba y el jefe todavía tiene más de la mitad de la vida
Fase 2	el ataque acaba y el jefe tiene menos de la mitad de la vida

Tabla 4.13

Estado: Morir	
Descripción	El agente muere

Tabla 4.14

4.6 Objetos

Durante el transcurso del juego el jugador necesitará hacer uso de varios objetos puestos a su disposición para poder avanzar, objetos a los que podrá acceder mediante la interfaz de inventario, de la que se ha hablado en su apartado correspondiente. Estos objetos se conseguirán a lo largo del juego en diferentes cofres repartidos por los niveles y ocasionalmente serán tirados al suelo por los enemigos al morir. Durante el avance de los niveles, el jugador encontrará los siguientes objetos:

- **Pociones de vida:** Al utilizarlas, el jugador regenerará una cantidad variable de vida, dependiendo del color de la poción. Son los únicos objetos que se eliminan del inventario una vez son utilizados, siendo importante gastarlas en el momento correcto.
- **Espada:** Es necesaria para poder atacar a los enemigos con los que te encuentras
- **Bola de agua:** Se utiliza para apagar las antorchas encontradas en el entorno durante todo el juego, consiguiendo una mayor interacción con el mapa y eliminando fuentes de luz para que los enemigos no encuentren al jugador.
- **Bola de fuego:** Al igual que el anterior objeto, su utilidad recae en interactuar con las antorchas, esta vez encendiéndolas.
- **Señuelo:** Al equipárselo, el jugador verá el trayecto que el señuelo trazará al ser lanzado. Al lanzarlo, el señuelo se mantendrá en el suelo durante unos segundos, atrayendo a los enemigos que estén a un rango máximo con el objetivo de distraerlos y poder avanzar evitando enfrentamientos.

4.7 Interacción con el escenario

Como ya hemos visto en los objetos, el jugador no solo podrá interactuar con los enemigos, sino que también con el propio escenario. Esto lo podrá hacer primeramente mediante la habilidad de encender o apagar antorchas según le convenga como ya hemos visto anteriormente, pero también habrá otras opciones.

Durante el transcurso de los niveles, el usuario encontrará varios cofres que podrá abrir para conseguir objetos, algunos de los cuales serán cruciales para poder continuar con el nivel. Además, existirán ciertos puntos en los que el jugador no podrá avanzar de forma directa y deberá encontrar la forma de continuar, ya sea buscando un nuevo camino desde donde abrir la puerta o interactuando con los objetos más próximos.

Por último, para completar las opciones del protagonista a la hora de moverse, el jugador podrá subir o bajar por escaleras de mano repartidas por el entorno, así como esconderse detrás de diversas cortinas para evitar a los enemigos y mantenerte fuera de su rango de visión.

4.8 Misceláneo

Además de todos los apartados de los que ya se ha hablado de forma general, es necesario hacer hincapié en algunos elementos que, aunque no son tan extensos, son igual de importantes a la hora del desarrollo y el correcto funcionamiento del videojuego. Algunos de estos elementos son:

- **Cámara:** Durante el juego la cámara seguirá al personaje principal con un poco de retraso, dando sensación de que se mueve de forma independiente. Además, cada vez que el jugador ataque o reciba daño hará una vibración, dando un *feedback* claro de que ha sucedido una de las dos cosas.
- **Sonido:** Durante todo el juego habrá música de fondo. Además, tanto el protagonista como los enemigos tienen sonidos para varias de las acciones que realizan, ya sea andar, saltar o recibir golpes, entre otras.

CAPÍTULO 5
Desarrollo de la solución

En el capítulo siguiente se profundizará en el desarrollo del proyecto. Esta parte se dividirá en los diferentes elementos del videojuego realizados para este proyecto, centrándonos en las herramientas utilizadas para dar las funcionalidades necesarias en cada caso y el uso que se ha dado a las mismas. Además, se explicará el proceso que realizado y la programación que hay detrás de cada uno de estos elementos. Durante el desarrollo del juego se han presentado varias dudas que se han ido resolviendo gracias a los diferentes foros y documentación existentes sobre Unity, además de algunos proyectos realizados por personalidades conocidas por sus tutoriales como Brackeys [10], BlackthornProd [11] o TheKiwiCoder [12].

En cada uno de los apartados se empezará discutiendo cada uno de los componentes que ha de tener dicho elemento y se continuará explicando como todo se junta y el código que establece el funcionamiento deseado. Además, tal y como se ha especificado anteriormente, el juego se ha ido desarrollando mediante prototipos, avanzando por funcionalidades y no por elementos, de forma que frecuentemente se realizaban avances en varios elementos de forma simultánea. Para el desarrollo de la memoria esto no se tendrá en cuenta, sino que se centrará en cada uno de los elementos por separado, partiendo de la versión final de los mismos y explicando el proceso que se ha realizado para llegar al mismo.

5.1 Niveles

Para el desarrollo de cada uno de los niveles se ha utilizado un sistema de *Tilemaps*, los cuales establecen una cuadrícula en la que poner de forma sencilla *Sprites* mediante una interfaz gráfica intuitiva. Gracias a estas facilidades aportadas por Unity se ha reducido considerablemente la complejidad de la implementación de los niveles.

Esta interfaz nos permite seleccionar la posición en la que se quiere colocar cada uno de estos *Sprites*, permitiendo además la localización en diferentes capas, seleccionando cuáles se quieren tener por encima del resto. De esta forma se puede dar sensación de profundidad, así como colocar diferentes *sprites* en el mismo cuadrado de la cuadrícula, aprovechando las transparencias que puedan tener y permitiendo una complejidad mayor en el diseño de los escenarios. Estas diferentes capas permiten dar una funcionalidad a cada una de ellas, separando, por ejemplo, las paredes de las escaleras o los escondites. De esta forma se permite un control mayor desde código y la posibilidad de añadir determinados componentes a una capa concreta, como podrían ser diferentes *Colliders* o *Triggers*.

La mayoría de los *Sprites* utilizados para los niveles han sido obtenidos mediante varios paquetes descargados de la *Unity Asset Store*, permitiéndonos centrarnos en el propio nivel. Una vez obtenidas las imágenes que se van a utilizar será necesario añadirlas a una paleta que Unity denomina *Tile Palette*. Desde aquí se podrán colocar en el mapa de forma sencilla. Esta herramienta además nos permitirá copiar otros *Sprites* previamente colocados, borrarlos, rellenar toda una zona con uno en concreto o seleccionarlos para mover o rotar los mismos.

Al desarrollar los niveles se ha priorizado la sensación de satisfacción y libertad por parte del jugador, permitiendo en ocasiones superar los desafíos propuestos de diferentes formas.

Por último, uno de los apartados más importantes de los niveles ha sido la iluminación, la cual es indispensable para dar el aspecto que se quería al juego, pero además ofrece distintas posibilidades a la hora de jugar el juego. Para la implementación de esta iluminación se ha utilizado las herramientas dadas por **URP: Universal Render Pipeline**, las cuales permiten la iluminación pretendida en un espacio 2D. En concreto, se ha utilizado una luz global para todo el nivel a una intensidad tenue que permite la sensación de oscuridad constante y luces de tipo *Spot* que ubican un foco de luz en un punto concreto, permitiendo cambiar el radio y rango de dicha luz. Estas últimas se han utilizado para las antorchas que hay repartidas por los niveles.

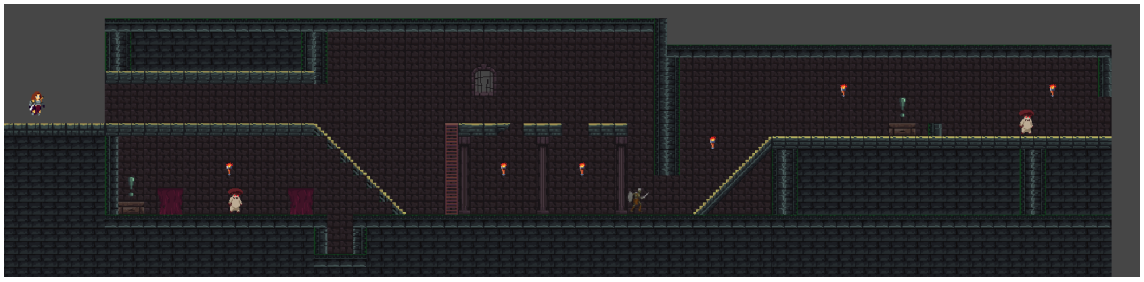

Figura 5.1: Nivel de tutorial

5.2 Protagonista

Comenzando por uno de los elementos más importantes del juego, el protagonista, lo dividiremos en dos partes diferentes, centrándonos primero en el apartado visual del mismo y luego en sus funcionalidades.

El jugador estará compuesto por un objeto vacío dentro del cual se ubican cada una de las partes del mismo, cada una con sus propios componentes, teniendo de esta forma las funcionalidades organizadas y permitiendo realizar cambios de forma mucho más sencilla.

Uno de estos objetos internos al protagonista, llamado **cuerpo**, será el utilizado para aportar el apartado visual. Para ello, como se ha explicado anteriormente en el apartado de diseño, se ha descargado un conjunto de *sprites* que cumplían con los objetivos que teníamos para el mismo. Gracias a estos *sprites* se adelantó este apartado de una forma mucho más rápida, pudiéndonos centrar en las animaciones y en la parte funcional. Para aplicar estos *sprites* tan solo es necesario añadir un componente **Sprite renderer** y seleccionar aquel que se quiera utilizar, en nuestro caso la imagen que representa al protagonista quieto. Por último, será necesario establecer correctamente el material y la *sorting layer* del mismo, de forma que se sitúe en la capa correcta y le afecte la iluminación del nivel.

Para acabar la parte visual habrá que realizar la máquina de estados de las animaciones. Para ello, el primer paso será añadir un nuevo componente denominado **Animator** y crearemos un **Animator controller**, donde nos permitirá establecer las animaciones y las transiciones entre las mismas tal y como se ve en la figura 5.2. Cada uno de los estados de dicho controlador hará referencia a una animación diferente, siendo las flechas entre las mismas las transiciones con sus respectivas condiciones de las que se hablará en el apartado de las funcionalidades.

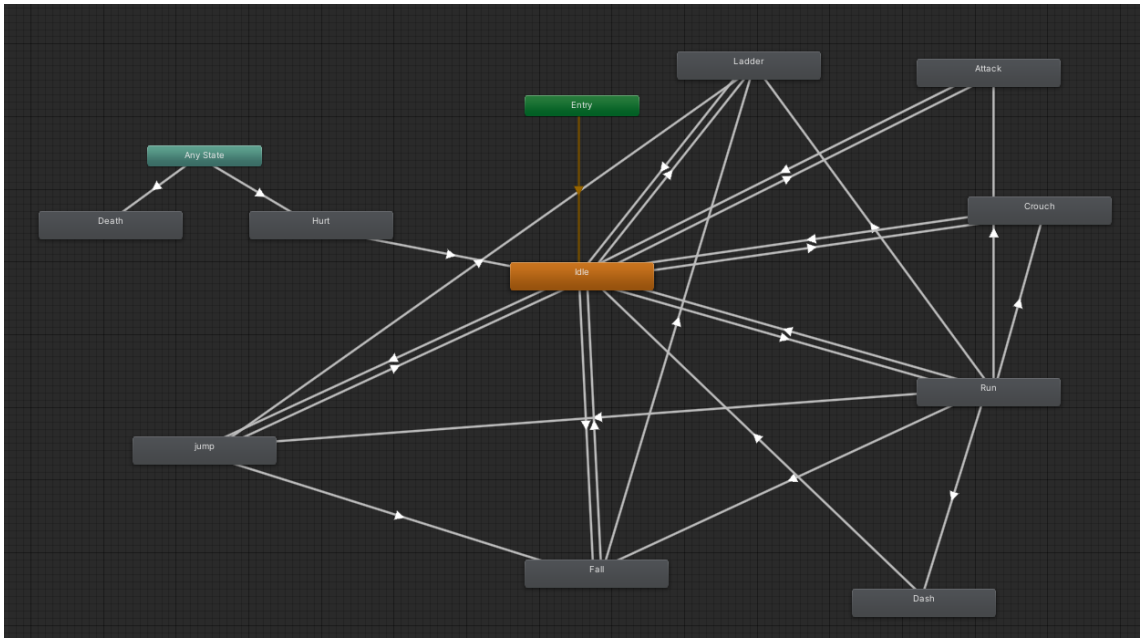

Figura 5.2: Controlador de animaciones

El resto de objetos que forman parte del protagonista dividen las funcionalidades de las que dispone el mismo, estando la mayoría en el propio objeto vacío **Player**. Estas funcionalidades se verán en las subsecciones presentadas a continuación.

5.2.1. Movimiento

En él encontramos, entre otros componentes, el *script* de movimiento, en el cual se encuentran todas las funcionalidades relacionadas con la posición del jugador, tanto el propio movimiento como la posibilidad de esconderse, subir o bajar escaleras de mano, saltar... Todas estas funciones estarán separadas en su propio método de forma que el código sea legible más fácilmente. Estas, a su vez, serán llamadas desde los métodos *FixedUpdate()* y *Update()* heredados de la clase *Monobehaviour*, dependiendo de si utilizan el motor de físicas aportado por Unity o no respectivamente. A continuación se presentarán cada una de las acciones realizadas en este *script* abstrayendo el código para su correcto entendimiento:

- **Movimiento:** El desplazamiento del personaje se realizará mediante el componente *RigidBody2D* presente en el objeto *player*. Este componente deberá ser de tipo dinámico para verse afectado por las físicas del videojuego, además de tener una detección de colisiones continua para evitar posibles errores. En cuanto al desarrollo del movimiento, se ha optado por realizar el desplazamiento mediante la variación de la velocidad del *Rigidbody2D*, siendo tan solo necesario utilizar una variación diferente según el lado hacia el cual se quiere mover el jugador, si se encuentra en el suelo o en el aire o si el personaje está agachado. Además, habrá que ser capaz de

saber cuando el jugador cambia de dirección de forma que el componente visual el personaje también gire.

- **Salto:** La programación del salto seguirá un funcionamiento similar al movimiento, añadiendo velocidad en el eje vertical del personaje cuando se ejecute la acción. Para que el jugador pueda saltar debe cumplir que no esté agachado ni escondido, además de estar tocando el suelo de forma que no pueda encadenar saltos, permitiendo volar al jugador. Además, para dar una mejor experiencia al usuario, se decidió eliminar la posibilidad del salto en aquellos casos en los que el personaje tenga un techo justo encima, haciendo que el movimiento y sus animaciones sean más fluidas que en caso de permitirlo en esos casos. Para ello se realizará un *BoxCast* desde la cabeza del protagonista que detectará la colisión con un techo en caso de haberlo.

- **Agacharse:** Como se ha dicho anteriormente, el objeto *player* está formado a su vez por varios objetos. Para agacharse se ha utilizado el objeto **cabeza**, sobre el cual se ha incorporado un nuevo *Collider* además del que utiliza el propio personaje. El diferenciarlo de esta forma nos permite utilizar los dos para detectar las colisiones cuando el personaje está de pie, pero desactivar el perteneciente a la cabeza cuando está agachado, lo que junto a un cambio en la animación establece la funcionalidad deseada. Además, al levantarse se deberá comprobar primero si hay algo encima del personaje que pueda bloquear la acción, evitando de esta forma estar de pie en sitios donde el personaje no cabe visualmente.

- **Embestida:** Esta funcionalidad se compone de dos partes principales. La primera de ellas, y más importante, es la propia embestida. El proceso para realizar la arremetida consiste en comprobar que la habilidad está disponible actualmente y, en caso de estarlo, aumenta en una gran cantidad la velocidad de movimiento del personaje durante un tiempo muy corto, dando esa sensación de embestida. Para evitar la posibilidad de utilizar este movimiento de forma continua, existe un tiempo de carga durante el cual la habilidad estará bloqueada.

- **Escaleras de mano:** Cuando el jugador interactúe con las escaleras de mano situadas por los niveles, se bloqueará el movimiento horizontal del mismo, permitiendo tan solo el movimiento vertical hasta que el jugador vuelva a tocar el suelo, suba hasta la parte superior de las escaleras o vuelva a apretar la tecla designada para la interacción con las mismas. Además, mientras el jugador esté interactuando con las escaleras, se eliminará la gravedad sobre el personaje, evitando que caiga al dejar de subir activamente, dando de esta forma la sensación de estar cogido a ellas.

- **Otras funcionalidades**: Dentro de este *script* existen varias funcionales añadidas que corresponden a la sección de interacción con el escenario y que, por tanto, se explicarán más adelante.

Además, en cada una de estas funciones se ha de modificar los valores de las condiciones que conforman las transiciones de la máquina de control de animaciones, de forma que funcionen acorde con el movimiento realizado por el protagonista.

5.2.2. Combate

El siguiente bloque importante de opciones ofrecidas por el protagonista son aquellas relacionadas con el combate. Estas mecánicas se sitúan en el *script* de **PlayerState**, también situado en el objeto vacío **player**.

El combate se divide en varias funcionalidades, la primera de las cuales consiste en ser capaz de dañar a los enemigos. Con este objetivo se crea un objeto interno al **Player** llamado **AttackPosition**. Este objeto marcará la posición alrededor de la cual deberán situarse los enemigos para ser dañados por el ataque. Para realizar el ataque se superpone un *collider* encima de dicho objeto del tamaño deseado en el cual se detectarán cada uno de los enemigos que se encuentren en esa posición, de forma que es posible acceder a los métodos de dichos enemigos y bajarles la vida como se explicará más adelante. Al igual que en el movimiento, estos ataques estarán animados. Para que la animación y el momento en el que el enemigo recibe el golpe ocurran de forma simultánea se ha hecho uso de una corrutina en la cual se retrasa ligeramente el comienzo de la animación, mientras que también bloquea la posibilidad de realizar el ataque directamente a continuación, de forma que sea necesario un tiempo corto entre ataque y ataque.

Para que el combate esté completo es necesario que el jugador también pueda recibir daño por parte de los enemigos. Para esto existe un contador de la vida que tiene el protagonista en cada momento, descendiendo en uno cada vez que es atacado por un enemigo y reiniciando al anterior punto de control, localizado al principio de cada uno de los niveles. Esta vida podrá ser vista por el jugador en cualquier momento como parte de la interfaz. **Los ataques de los enemigos se cubrirán en la sección dedicada a los mismos**

5.2.3. Inventario

Aunque fueron barajadas varias opciones, el inventario estará desarrollado mediante una clase denominada **Inventory**, la cual dispondrá de varios *Array* para manejar los huecos existentes, así como la cantidad de objetos que se encuentran en cada uno de ellos. Además, existirá un tercero utilizado para almacenar los propios objetos que se visualizarán en este inventario.

El almacenamiento de objetos funcionará de forma que al acercarse a los mismos, el jugador los recogerá de forma automática y se meterán siempre en la misma posición del inventario, dependiendo del objeto que sea. De esta forma cada objeto estará siempre en el mismo hueco, permitiendo, en el caso de las pociones, tener más de una del mismo tipo que se irán gastando con su uso. El resto de objetos tendrán una cantidad de usos ilimitados.

Además, será importante la presencia de un nuevo objeto en cada una de las escenas. Este objeto será **ItemAssets**, el cual contiene un *Script* del mismo nombre cuya función permite generar todos los objetos existentes y ser usado por el resto de funcionalidades en el sistema de inventario y objetos.

5.3 Objetos

Otro de los componentes principales de nuestro videojuego son los objetos. Estos objetos se encontrarán durante el transcurso de los niveles en distintos cofres y dejados caer por algunos de los enemigos presentes en los mismos. A continuación se diferenciará entre cada uno de ellos y su implementación:

- **Pociones de vida:** Es el objeto más sencillo de implementar. Al usarlo tan solo es necesario aumentar el contador de vida en una cantidad diferente dependiendo del color de la poción, lo que a su vez actualizará la interfaz de forma automática.

- **Espada:** La implementación de este objeto ha sido explicada en la sección de combate. Dicho combate solo será posible en caso de tener la espada equipada, en cualquier otro caso el jugador no será capaz de realizar ataques a los enemigos.

- **Bola de agua y fuego:** A efectos de la implementación, los dos objetos se realizarán de la misma forma, cambiando tan solo el *sprite* utilizado y su interacción con el escenario, apartado que no se discutirá en esta sección. Estos dos objetos deberán ser lanzados por el jugador en línea recta, desapareciendo cuando un tiempo concreto después. Para ello, los objetos deberán tener un *Rigidbody2D* del tipo *kinematic*, de forma que no se vea afectado por las fuerzas externas, pero sea posible aplicarle fuerzas desde código. Una vez hecho esto se utilizará otro de los objetos internos del protagonista, **RotatePoint** para situar el punto desde donde se crearán dichos objetos al ser utilizados. Para ello, el punto interno al *RotatePoint*, llamado **BulletTransform** se moverá trazando una órbita al rededor del protagonista según la posición en la que se sitúe el ratón en cada momento. Una vez utilizado el objeto, aparecerá en la posición de este punto en ese momento concreto, a partir de lo cual se ejecutará el código disponible dentro de la propia bola, el cual se encargará de aplicar una fuerza sobre la misma que la impulsará en dirección a la posición del ratón en el momento de la creación del objeto. Además, el objeto deberá tener un *Collider* para que sea posible su interacción con el escenario.

- **Señuelo:** El señuelo también será lanzado al ser utilizado, solo que a diferencia de las bolas de agua y fuego, este si se verá afectado por la gravedad, por lo que su trayectoria será parabólica en vez de lineal. Es por ello que se decidió añadir una pista visual al proceso de lanzamiento del señuelo, permitiendo al usuario tener mucho más claro donde se está lanzando. Para lograr esta trayectoria. que solo será visible al tener el señuelo seleccionado, se instanciarán varios puntos siguiendo el mismo método que para el lanzamiento, con la diferencia de que los cálculos realizados servirán para variar la posición a la que aparecen estos puntos en vez de para darle la velocidad correspondiente al señuelo. De esta forma estos puntos se pondrán siguiendo exactamente la trayectoria que seguirá el señuelo en caso de ser lanzado. Una vez lanzado el señuelo se mantendrá en el suelo un tiempo determinado. Tiempo en el cual realizará su función de atraer a los enemigos.

5.4 Enemigos

Otro de los pilares de los juegos comprendidos dentro de los mismos géneros que el realizado son los enemigos. Durante su desarrollo se barajaron varias formas de realizarlos, concluyendo que la que más se adecuaba a las funcionalidades deseadas consiste en una máquina de estados. Por medio de ella, los enemigos cambiarán su comportamiento de forma dinámica entre distintos previamente predefinidos, dependiendo de su entorno y las acciones del jugador.

Siguiendo este procedimiento, los enemigos siempre se situarán en uno de los estados definidos en la máquina de estados, siendo estos los mismos para todos los enemigos a excepción del jefe final. Dependiendo de las condiciones que se cumplan en cada momento, los enemigos, de forma independiente, podrán variar su estado y, por tanto, su comportamiento.

5.4.1. Máquinas de estados

Para desarrollar la máquina de estados se utilizará una clase a la que llamaremos **AiStateMachine**. En esta clase estarán definidos todos los métodos que serán necesarios para el correcto funcionamiento de la misma, entre los que se verán los siguientes:

- **Método constructor:** Se utilizará para crear una nueva máquina de estados, lo cual tendrá que hacer cada agente para poder realizar cualquiera de sus acciones. En este método se obtendrán los estados que conformen dicha máquina, así como el agente al que está asociada.
- **RegisterState():** Será necesario para registrar cada uno de los estados que se hayan definido para un agente de forma que posteriormente sea capaz de alternar a dicho estado.
- **GetState():** Permite al agente saber en qué estado se encuentra actualmente.
- **Update():** Realiza el método *Update* del estado en el que se encuentre el enemigo.
- **ChangeState():** Será el método que se utilizará para realizar la transición entre dos estados por parte del agente.

A la hora de crear un nuevo agente habrá que seguir el siguiente proceso por medio de los métodos ya explicados de la máquina de estados: El primer paso será generar la nueva máquina de estados para posteriormente registrar cada uno de los estados que se quiere que tenga. Además, este estado deberá estar incluido dentro del *enumerador* situado en el *script* **AiState**.

En este *script*, además del *enumerador* encargado de listar todos los estados existentes en el proyecto, también se ha desarrollado una interfaz en la que se encuentran cada uno de los métodos que deberán tener los estados creados. Estos métodos se utilizarán de la siguiente forma:

- ***GetId()*:** Tal y como indica el nombre del método, se utilizará para obtener el identificador del estado en caso de ser necesario. Este método se ha utilizado principalmente en la implementación de la propia máquina de estados.
- ***Enter()*:** Este método será llamado cada vez que el agente entre en un nuevo estado, realizándose tan solo una vez al principio del mismo.
- ***Update()*:** Será el cuerpo principal de la mayoría de los estados. En él se encontrará el fragmento de código que se quiere que se ejecute de forma constante mientras que el agente se sitúe en ese estado.
- ***Exit()*:** Será lo último que se llame en un estado una vez comienza la transición hacia otro estado. Al igual que el método *Enter()* tan solo se ejecutará una vez por cada vez que el agente salga de ese estado.

A continuación se profundizará en cada uno de los estados en los que los enemigos se pueden situar. En cada uno de los apartados se explicarán las condiciones que se han de dar para que el agente transite al estado correspondiente, así como su funcionamiento.

5.4.2. Estado: Quieto

Este estado es el más simple y uno de los dos posibles estados iniciales de los agentes. Como tal, solo será alcanzable cuando se requiera volver al estado inicial, lo que puede producirse cuando el enemigo pierde de vista a su objetivo, ya sea el jugador o un señuelo lanzado por el mismo.

Aunque visiblemente el enemigo no esté realizando ninguna acción mientras se encuentra en este estado, será necesario estar realizando comprobaciones de forma continua. Entre estas comprobaciones se encontrarán el control de la posición del jugador respecto al agente, al igual que, de forma secundaria, la posición de posibles señuelos lanzados por el jugador. En cualquiera de estos dos casos el agente deberá cambiar su estado. Por último, también se tendrá en cuenta si el agente es golpeado mientras está en estado **Quieto**, en cuyo caso se le deberá aplicar una fuerza en dirección contraria al origen del golpe y comenzará a perseguir al jugador.

5.4.3. Estado: Patrullar

El estado de patrulla es el otro posible estado inicial de los agentes. En este estado el enemigo recorrerá una zona delimitada de forma dinámica según la posición en la que se encuentre dentro del nivel. Esto quiere decir que no estará programado una delimitación por la que caminar, sino que patrullará por toda la zona que tiene disponible dependiendo de donde se sitúe. El agente podrá llegar a este estado en caso de perder de vista al jugador después de perseguirle o teniéndolo como estado inicial.

En este estado el agente caminará de forma continua hacia adelante hasta encontrarse con una pared o un hueco, en cuyo caso el enemigo girará y continuará patrullando hacia el lado contrario. Estos obstáculos se detectarán por medio de un objeto vacío situado en la parte inferior delantera de los agentes, a partir del cual se lanzarán dos *Linecast*, los cuales serán capaces de detectar las colisiones previamente a que el agente llegue a esas posiciones. De esta forma, en caso de que el *Linecast* que se lanza de forma horizontal detecte una pared, o el *Linecast* vertical lanzado hacia abajo no detecte suelo, el agente girará.

Además, durante transcurso de este estado es necesario comprobar constantemente si el personaje o un señuelo lanzado por el jugador están a rango, de forma que en caso de detectarlos el enemigo cambiará de estado a **perseguir** o **Señuelo** respectivamente. También deberá detectar si el jugador le golpea, pasando a también a perseguirle en ese caso.

5.4.4. Estado: Perseguir

En este estado el agente deberá seguir al jugador siempre que este esté al alcance, es decir, que el agente sea capaz de llegar a él y que esté situado a una distancia máxima previamente definida.

Para que el agente sea capaz de perseguir al jugador es necesario utilizar una técnica de *Pathfinding* que le diga que camino debe trazar para llegar a dicha posición de la forma más óptima siempre que sea posible. Para ello se utilizará un paquete externo a Unity denominado **A* Pathfinding Project** [13], el cual facilitará considerablemente esta tarea, llegando a ser capaz de eliminar todo requerimiento de código para realizar esta acción en caso de que sea necesario. Para la realización de este proyecto nos hemos apoyado en las funcionalidades añadidas, pero haciendo uso de nuestro propio código a la hora de realizar su implementación.

Una vez añadido el paquete externo, se creará un nuevo objeto vacío donde se situará la lógica referente al algoritmo de *pathfinding*. En este objeto se añadirá el *script* Pathfinder, añadido como parte del proyecto descargado. Una vez puesto este *script* se nos habilitarán varias opciones desde el propio *inspector* del objeto, en las cuales podremos configurar el tipo de mapa en el que se van a producir las persecuciones, en nuestro caso un mapa con forma de cuadrícula, el tamaño de cada uno de los nodos o cuadrados que la forman y las capas que queremos que actúen como obstáculo para los agentes, las cuales no serán atravesables por los mismos. Una vez configurado esto y especificado que se deberán utilizar físicas de 2D, al dar al botón de escanear se mostrarán aquellas zonas por las que los agentes serán capaces de pasar y que elementos les bloquearán el paso. Para acabar con las funcionalidades ya hechas gracias al paquete descargado, será necesario añadir a los agentes el *script* **Seeker**, el cual les permitirá interactuar con el grafo realizado mediante el código anterior.

A partir de ahí crearemos un nuevo archivo de código al que llamaremos **PathfindingA** en el cual comenzaremos la funcionalidad de **perseguir**. En este se predefinirán las velocidades que queremos que sigan los diferentes enemigos y estableceremos al jugador como objetivo. Este código funcionará de forma que se crean diferentes *Waypoints* o puntos de control entre el enemigo y el jugador, los cuales se irán actualizando a cada movimiento de cualquiera de los dos participantes. El agente intentará alcanzar siempre el siguiente punto de control, acercándose de esta forma al jugador. Una vez llegado al mismo pasará al siguiente en caso de existir, si no existiese entonces el agente ya se sitúa en la misma posición que el jugador.

Además, en este código se tendrá en cuenta la dirección a la que se dirige el agente para solucionar posibles problemas con las animaciones o la detección del jugador. Por último, especificar que todo este proceso de *pathfinding* solo se realizará en caso de que el agente se encuentre en el estado de **Perseguir**, para lo cual se situará el código troncal dentro de una cláusula condicional que lo verifique.

Una vez establecido todo lo necesario para la acción de perseguir al jugador, nos centraremos en el código del propio estado. En este estado será necesario activar la animación de movimiento y el *boolean* que controla el algoritmo de *pathfinding*. Una vez hecho esto, dentro de la función de *Update()* se realizarán las comprobaciones necesarias, siendo esto la distancia del agente al jugador o la presencia del mismo dentro del rango de ataque, en cuyos casos se pasará a los estados de **Patrullar** y **Ataque** respectivamente en caso de cumplir las condiciones necesarias. Además, se deberá tener en cuenta la posibilidad de ser atacado por el jugador o de la presencia de una pared que bloquee el camino, haciendo que sea imposible perseguir al jugador. Por último, será necesario desactivar el *boolean* del *pathfinding* en la función de *Exit()* para asegurarse de que el agente no persigue al jugador cuando no deba.

5.4.5. Estado: Atacar

En este estado se realiza el ataque de los agentes hacia el jugador. Para llegar a este estado, el enemigo debe haberse acercado lo suficiente al protagonista, transitando al estado cuando se alcanza una distancia mínima estipulada. Además, previamente al cambio de estado, se volverá a comprobar si el jugador se encuentra en una posición en la que visualmente el enemigo sea capaz de golpearle, de forma que no se produzcan ataques fuera del rango que percibe el jugador. Para realizar esta comprobación se utiliza un objeto vacío situado en la parte delantera del enemigo, posición en la cual se creará un *Collider* desde código con el que interactuará el protagonista en caso de situarse en dicha posición.

Una vez en este estado, el agente deberá eliminar toda velocidad que pudiese tener en ese momento, manteniéndose en su posición durante el tiempo que dure dicho ataque. A partir de ahí se lanzará la animación correspondiente y se comprobará por última vez que el jugador se encuentre a rango de ataque del enemigo. Esta comprobación no servirá para cancelar el ataque, puesto que la animación ya estará en proceso, sino que determinará si el jugador recibe daño o no, dando la sensación de que ha sido capaz de esquivar el ataque a tiempo en caso de no ser detectado. En cualquiera de los casos se ejecutará el sonido correspondiente al ataque y se cambiará al estado de **Perseguir**.

En este caso, a diferencia de en la mayoría de los estados, todo el código cuyo funcionamiento realiza las acciones descritas se situará en el método *Start()* del estado. Esto es debido a que se quiere que solo se realice el ataque una vez al entrar al estado y posteriormente pasar a perseguir al jugador de nuevo, de forma que el jugador verá como el enemigo se posiciona y ataca de forma constante, evitando ataques en momentos que no deberían realizarse. Por otra parte, al igual que en el resto de estados, el agente si deberá comprobar continuadamente si recibe un ataque del jugador.

5.4.6. Estado: Señuelo

Los agentes cambiarán a este estado cuando detecten un señuelo a una distancia determinada de los mismos, independientemente del estado en el que se encontrasen previamente, siempre que continúen con vida. Esta detección funciona de forma similar a aquella utilizada para que el agente comience a perseguir al jugador, con la salvedad de que en este caso será indiferente si el señuelo se sitúa en la parte delantera o trasera del enemigo.

Una vez se detecta dicho señuelo y se transita a este estado, el agente andará hacia él hasta conseguir alcanzarlo, momento en el que se quedará en la posición que se encuentre. Para ello se comprobará de forma continua la posición del señuelo, ya que podría no estar quieto todavía y su propia presencia, ya que desaparecerá con el tiempo. En el caso de que el señuelo salga del rango de detección o desaparezca, el agente volverá al estado que tenía asociado como inicial.

Al igual que en el resto de estados que implican movimiento, se deberá tener en cuenta la posición del objetivo con respecto al agente para girarlo en caso de que fuese necesario, asegurando el correcto funcionamiento de la programación y las animaciones.

5.4.7. Estado: Fase 1 Jefe

Es al estado en el que se encontrará el jefe final en cuanto el jugador entre en la zona delimitada para el combate. Además, es al que volverá después de cada ataque siempre que su vida sea mayor de la mitad.

En esta fase el agente perseguirá al jugador, habiendo una pequeña probabilidad de realizar un ataque por cada iteración que se realiza al método *Update()* del estado. Para ello se obtiene un número aleatorio en un rango muy grande y en caso de ser menor que la cantidad seleccionada, el agente atacará. Estos ataques se encontrarán en estados diferentes y solo se podrá alternar a ellos en caso de que dichos ataques estén recargados. Con estas dos comprobaciones, el rango y la recarga, se evita que el agente esté atacando de forma continua, permitiendo al jugador reaccionar y tener momentos en los que atacar al jefe. En esta fase los posibles ataques serán una carga hacia el jugador y disparos en forma de bolas de energía. Estos ataques se explicarán más adelante en sus propios estados.

Por último, el jefe también debe dejar claro que ha recibido un golpe, por lo que se seguirá aplicando la fuerza hacia el lado contrario del que es golpeado y se lanzará una animación, al igual que con los enemigos normales.

5.4.8. Estado: Fase 2 Jefe

La segunda fase del jefe será alcanzada cuando la vida del mismo baje hasta la mitad, siendo más peligroso en esta segunda parte del combate. Una vez en esta fase, después de cada uno de los ataques volverá a ella sin volver a entrar en la primera.

La programación realizada para esta parte del combate es comparable a la de la primera fase, siguiendo las mismas bases y haciendo uso de los mismos métodos. La diferencia entre las dos reside en los ataques que se realizan, siendo más fuerte el ataque del disparo de energía y añadiendo un nuevo ataque, el cual también se explicará en el estado de **disparar**.

5.4.9. Estado: Cargar Jefe

El estado de carga es el primer ataque que el jefe final podrá realizar, tanto desde la primera como desde la segunda fase.

Este ataque se realizará obteniendo la posición en la que se sitúa el jugador al entrar en el estado y aumentando en una gran cantidad la velocidad del enemigo en dicha dirección. Mientras que la dirección solo se cambiará al entrar en el estado, la velocidad se le aplicará de forma continua mientras el agente siga cargando, parando solo a los dos segundos de empezar la carga o al chocar con una de las paredes que conforman la habitación en la que se realiza la lucha. Para poder parar la carga a los dos segundos de comenzarla se hará uso de una *corrutina*, la cual se llamará al comenzar la carga y ejecutará el código interno simultáneamente al código que añade la velocidad al enemigo.

5.4.10. Estado: Disparar Jefe

En este estado se comprenden el resto de ataques que el jefe es capaz de realizar. Dos de estos ataques serán variaciones del mismo dependiendo de la fase en la que se encuentre el jefe, mientras que el tercero será un nuevo ataque que solo podrá realizar al estar en la segunda fase, teniendo una probabilidad menor que el resto. Separaremos, por tanto, la explicación en dos:

- **Disparos de energía:** Consiste en un disparo de bolas de energía desde la posición del agente hacia varias direcciones, las cuales variarán según la fase en la que se encuentre. En la primera fase el agente lanzará cuatro bolas, una en cada dirección, mientras que cuando su vida sea inferior a la mitad lanzará 8 bolas, yendo una en cada dirección, incluyendo las diagonales. Estas bolas avanzarán en línea recta hasta desaparecer pasados unos segundos y harán daño al jugador en caso de tocarle.
- **Lanzar Rocas:** Este ataque solo será lanzado en la segunda parte del combate y consiste en la caída de rocas desde el techo de la habitación. Estas rocas caerán en tres oleadas diferentes, apareciendo cada vez de una posición diferente, de forma que el jugador deba moverse para esquivarlas. Para ello se han creado varios objetos vacíos situados en la parte superior de la habitación, los cuales se han guardado en un *Array* en el código. De esta forma se irá recorriendo el *Array* e instanciando las rocas en algunos de esos objetos. A partir de ahí las rocas caerán hasta el suelo, afectadas por la gravedad, dañando al jugador en caso de chocar con él.

5.4.11. Estado: Morir

Los enemigos llegarán a este estado cuando su vida quede por debajo de cero, lo cual ocurrirá al ser golpeado un número determinado de veces por el jugador. Una vez ocurrido esto se seguirán una serie de pasos hasta que el agente es eliminado de la escena. Estos pasos se realizarán en el método *Enter()* del estado, ya que queremos que se realicen una única vez.

Una vez la vida llegue a cero, el primer paso será realizar la animación de muerte, la cual será diferente según el tipo de enemigo que lo ejecute, al igual que el sonido seleccionado. Seguidamente, se desactivará uno de los objetos que forma parte de todo enemigo y que es utilizado para que el enemigo haga daño en caso de que el jugador lo toque. De esta forma, una vez el enemigo esté muerto, no dañará al jugador en caso de acercarse. Por último, se ha implementado una función mediante la cual los enemigos dejarán caer una poción en algunos casos cuando mueran.

Una vez se hayan realizado todos los pasos anteriores dará comienzo una corrutina mediante la cual se desactivarán las físicas del enemigo, así como su *collider*, manteniendo el cadáver en una posición estática durante el tiempo predefinido hasta que desaparece.

5.4.12. Enemigos comunes

Una vez explicados todos los posibles estados pertenecientes a las máquinas de estados presentes en el juego, hablaremos de los tipos de enemigos.

Durante el transcurso de los niveles nos encontraremos con dos tipos de enemigos comunes, los esqueletos y las setas humanoides. Estos dos tipos de enemigos se diferenciarán principalmente en el aspecto visual, del cual hablaremos a continuación.

Para realizar el aspecto visual de los enemigos se ha seguido el mismo proceso que con el protagonista, creando un objeto vacío llamado **Enemy GFX** dentro del propio enemigo en el que se sitúan los componentes *Sprite Renderer* y *Animator*. Estos componentes tendrán un *Sprite* y unas animaciones diferentes según el tipo de enemigo. Los enemigos dispondrán de las animaciones siguientes:

- **Quieto:** Utilizado en el estado con su mismo nombre o cuando intenta andar hacia una pared.
- **Corriendo:** Utilizado en los estados de perseguir, patrullar y señuelo.
- **Ataque:** Animación realizada cada vez que el agente entra en el estado de atacar.
- **Herido:** Se ejecuta cuando el enemigo recibe un golpe por parte del jugador.
- **Muerto:** Animación utilizada cuando pasa al estado **morir**.

Además de los cambios visuales, también hay ligeras diferencias entre ellos, como pueden ser el rango de ataque o la velocidad de movimiento de cada uno de los enemigos.

Aparte del componente que da el aspecto visual a los enemigos, los agentes dispondrán de otros tres objetos vacíos utilizados realizar correctamente las patrullas, atacar al jugador en caso de situarse delante del enemigo y un área a su alrededor en la que daña al protagonista en caso de acercarse excesivamente.

5.4.13. Jefe final

Por último tenemos el jefe final, encontrado en la última área del nivel 2. A diferencia de los enemigos comunes, este agente tendrá la habilidad de volar, siendo necesario jugar de una forma diferente para poder acabar el juego.

Al igual que el resto de agentes, el jefe final estará compuesto por varios objetos vacíos, encargados del aspecto visual y los daños de diferente tipo al jugador. Centrándonos en el aspecto visual, este agente dispondrá de las siguientes animaciones:

- **Volar:** Es la animación base del enemigo, la cual se realizará de forma constante mientras que no se estén ejecutando el resto.
- **Ataque:** Animación realizada cada vez que el agente realiza el ataque de **carga**.
- **Herido:** Se ejecuta cuando el enemigo recibe un golpe por parte del jugador.
- **Muerto:** Animación utilizada cuando pasa al estado **morir**.

Para los ataques realizados a distancia, el jefe se mantendrá en su posición mientras continúa con la animación de volar.

5.5 Escenario e interacciones

Por último, hablaremos de forma más concreta de las diferentes partes de los escenarios, así como algunas de las interacciones que el jugador tendrá con el mismo. De estas interacciones, este trabajo se ha centrado en las siguientes:

- **Escaleras:** A la hora de diseñar el juego, se optó por utilizar un *collider* con forma de cápsula para el protagonista. Esto se hizo con la idea de evitar que se bloquease o hiciese movimientos poco fluidos al moverse por el mapa. De la misma forma se utiliza un *collider* en forma de rampa para las escaleras, lo que junto con el *sprite* seleccionado da la sensación de escalera. Este conjunto de *colliders* establece un funcionamiento ya correcto de las escaleras con una única excepción. Dejándolo de esta forma se presenta el problema de que el personaje resbalará hacia abajo al mantenerse quieto en ellas. Para solucionar esto se ha creado un nuevo *collider* en el protagonista, esta vez con forma cuadrada. De la misma forma, en las escaleras se crea otro nuevo con la forma de las mismas. Por medio de código se comprobará cuando el personaje se mantiene estático en ellas y se intercambiarán los *colliders* hasta que vuelva a ponerse en movimiento. Además, aquellas escaleras que no tengan paredes detrás de las mismas también dispondrán de la funcionalidad explicada a continuación para las plataformas de una dirección.
- **Plataformas de una dirección:** Durante el transcurso de los niveles el jugador se encontrará con varias plataformas que tan solo se podrán atravesar desde la parte inferior sin hacer nada. Desde la parte superior, estas plataformas tendrán un *collider* que no permitirá el paso al jugador u otros agentes que puedan situarse encima de ellas. El jugador será capaz de atravesarlas usando la tecla escogida, lo que permitirá un giro de 180º al *collider*, dejando de esta forma que sea atravesada desde la parte superior pero no desde la inferior. Para esto, los *Colliders* serán utilizados mediante un nuevo componente denominado *Platform Efector*. Este componente permite establecer un *collider* solo desde un determinado ángulo, permitiendo variar la apertura del mismo así como su dirección. Gracias a ello se nos permite variar

la dirección desde código, invirtiéndola durante un tiempo definido al pulsar la tecla correspondiente y colocándolo de la forma inicial pasado unos segundos. Este mismo funcionamiento se aplicará también a las escaleras, permitiendo el comportamiento deseado para las mismas.

5.6 Interfaces

Uno de los elementos más importantes a la hora de dar cohesión a un videojuego son las interfaces. Estas servirán para conectar las diferentes escenas en las que se dividirá, además de permitir acceder a varias opciones que no podrían ser accedidas sin el uso de las mismas.

Para el desarrollo de las interfaces se ha utilizado el tipo de objeto *Canvas*. Este objeto se utiliza para englobar a todos los elementos de los que va a estar formada una interfaz gráfica, permitiendo un mejor orden, escalado y funcionalidad de los mismos, además de facilitar el posicionamiento de estos elementos en el videojuego.

Dentro de las diferentes interfaces del juego podremos encontrar los siguientes elementos:

- **TextMeshPro:** Es utilizado para los textos de las interfaces. Permite el cambio del texto o sus propiedades desde código.
- **Button:** Cada uno de los botones que forman parte de la interfaz. Estos permiten la navegación entre interfaces o incluso entre diferentes escenas del juego. Estos botones tendrán diferentes métodos que se ejecutarán al pulsarlos, realizando el comportamiento deseado
- **Image:** Componente utilizado para presentar *Sprites* en la propia interfaz. Este es utilizado, por ejemplo, para los objetos presentes en el inventario o los corazones que indican la salud restante del jugador.
- **Slider:** Utilizado para variar el volumen del sonido desde la interfaz de opciones.

5.6.1. Cuadros de texto

Además de las interfaces utilizadas para conectar las diferentes escenas o indicar la salud o los objetos del inventario del jugador, también se han utilizado diferentes cuadros de texto que aparecen a lo largo del juego al abrir los cofres situados a lo largo de los diferentes niveles. Estos cuadros de texto funcionan de forma diferente a los textos de los que se han hablado anteriormente.

Para estos cuadros de diálogo se quería un funcionamiento especial por el cual el mensaje apareciese de forma gradual, permitiendo además al jugador pasar cada una de las frases como si se tratase de un diálogo con otro personaje. Para ello se ha utilizado un editor externo denominado Inky y el *Plugin* **Ink**. Además, el juego quedará pausado cuando estos cuadros de texto estén activos, evitando que tanto el jugador como los enemigos se muevan hasta cerrarlos.

5.7 Almacenamiento persistente

Uno de los puntos importantes durante el desarrollo del videojuego es la posibilidad de salir en mitad de una partida y poder empezar de nuevo desde ese nivel con los

datos que se tenían. Es por ello que al acabar un nivel se deberá guardar la salud de la que dispone el jugador, los objetos que tiene en el inventario y el nivel en el que se ha quedado. De esta forma, cuando el jugador vuelva a entrar en la partida, comenzará desde el último nivel en el que se encontraba con los datos que tenía en ese momento.

Para realizar el guardado de estos datos se utilizará un documento externo con formato *JSON* en el cual se estructurarán los datos indicados anteriormente mediante pares nombre - valor. Estos pares serán sobreescritos cada vez que se deba guardar de nuevo la partida y leídos al intentar continuar la partida desde el menú principal. Estos datos se guardarán utilizando un algoritmo de cifrado para evitar posibles manipulaciones de la partida guardada.

5.8 Sonido

Con el objetivo de aumentar la inmersión del jugador en el juego, así como para convertirlo en una experiencia mucho más disfrutable se han añadido sonidos a algunas de las acciones más importantes o repetidas del videojuego. Estos sonidos, así como la música añadida tanto para el menú principal como para los niveles, han sido obtenidos desde una página web de recursos gratuitos [14] utilizados dentro del desarrollo de videojuegos. Estos sonidos añadidos al juego estarán relacionados con los movimientos y acciones del protagonista, así como los enemigos.

Para añadir estos sonidos se utiliza una nueva clase que almacenará cada uno de los diferentes clips de audio. Además, dispondrá de varios métodos que serán utilizados tanto para ejecutarlos en los momentos necesarios como para alterar el volumen de los mismos desde el menú de opciones del juego.

CAPÍTULO 6
Pruebas

Las secciones presentadas a continuación se centrarán en las pruebas que se realizaron una vez acabada la parte de desarrollo principal del proyecto. Una vez llegado a este punto se buscaron varias personas que pudiesen probar el proyecto realizado y darnos su opinión, así como hablarnos de todos los errores que pudiesen haber encontrado para poder mejorar el producto lo máximo posible. Por tanto, se organizarán los dos siguientes apartados de la siguiente forma: Un primer apartado centrado en los errores encontrados y las soluciones aportadas y un segundo apartado en el que nos centramos en aquellas sugerencias realizadas por aquellos que han probado el juego, así como las mejoras realizadas en caso de haberse tenido en cuenta. Además, se añadirá un último apartado en el que podremos encontrar los enlaces al propio juego y a un video mostrando una partida completa.

6.1 Beta

Como se ha especificado anteriormente, en este apartado se presentarán algunos de los fallos encontrados por las personas que probaron nuestro juego en esta fase temprana del desarrollo. A la hora de realizar un proyecto real, se contrata a un grupo de gente encargado de realizar las pruebas durante varias fases del desarrollo, permitiendo realizar cambios y arreglar problemas en todo momento. En nuestro caso se ha realizado este proceso tan solo al final del proyecto, lo que hizo que solucionar algunos de los errores fuese más complicado.

En aquellos elementos que conciernen a este trabajo podemos destacar varios errores relacionados con los enemigos y su comportamiento. En varios segmentos del segundo nivel, los enemigos eran capaces de atravesar ciertas paredes al perseguir al jugador, lo que ocurría debido a una mala configuración del grafo realizado por el algoritmo de *Pathfinding* A*.

Otro error relativamente común relacionado con los enemigos ocurría en el estado de **patrulla**. Cuando un enemigo caminaba hacia una escalera, había ciertas posibilidades de quedarse atascado o no supiese como reaccionar a la misma. Esto se pudo solucionar de una forma sencilla, añadiendo las escaleras a la capa en la que se encontraba el resto del terreno, de forma que al detectar una escalera el enemigo cambiase de dirección.

Por último, con respecto a los enemigos, por como estaba diseñada la máquina de estados en un principio, existía la posibilidad de que un enemigo muriese más de una vez, haciendo la animación y sonido de muerte varias veces. Esto se solucionó alterando ligeramente las transiciones que comunicaban el resto de estados con el estado de muerte.

También se encontraron varios errores con algunas de las acciones que el jugador podía realizar. Entre ellas destacar la posibilidad de utilizar la espada mientras se está agachado. Al permitir esta acción, el jugador podía dañar a los enemigos estando agachado, pero la animación no se realizaba. Esto se solucionó bloqueando la posibilidad de ataque o el uso de otros objetos al estar agachado, permitiendo solo realizar dichas acciones al estar en la posición predeterminada del personaje.

6.2 Feedback

Además de los múltiples errores encontrados por aquellas personas que probaron el videojuego, también se nos dieron diferentes ideas de mejora que podrían ser implementadas para una mejor experiencia del jugador. A continuación se mostrarán algunas de las mismas y como se han implementado dentro del juego.

Uno de los comentarios que se nos hicieron más veces fue que el combate se sentía demasiado difícil e injusto. Esto era debido a que el jugador presentaba un rango de ataque muy pequeño, provocando que los jugadores se sintiesen frustrados durante el combate, ya que era muy difícil no recibir un golpe cada vez que atacabas. Es por ello que se decidió incrementar el rango de ataque del jugador y disminuir ligeramente el de los enemigos.

De la misma forma, el combate contra el jefe final se sentía demasiado complicado, sobre todo teniendo en cuenta que si el jugador muere en combate contra él, debe empezar de nuevo el nivel. Para dar una experiencia más satisfactoria se decidió bajar las estadísticas de este enemigo, rebajando ligeramente su velocidad, rango de ataque y los porcentajes de probabilidad en los que ataca, haciendo que estos sean menos frecuentes.

Por último, muchos de estos usuarios comentaron la posibilidad de obtener objetos en otros lugares que no fueran cofres, aumentando así la capacidad de curación de cada partida y permitiendo correr más riesgos. Para ello se diseñó el sistema de caída de objetos al derrotar enemigos, incrementando la cantidad de pociones que el protagonista puede obtener.

6.3 Enlaces de interés

A continuación se encuentra el enlace desde el cual será posible descargarse el juego realizado para poder probarlo, así como el enlace a un video en el cual se podrá ver como acabar el juego de principio a fin, así como ver en funcionamiento las diferentes mecánicas implementadas y los niveles y ambientación en la que se sitúan:

- **Juego:** https://drive.google.com/file/d/18BU2D2G6KN65v95JufoolS9uCPc2hspT/view?usp=sharing
- **Video:** https://media.upv.es/#/portal/video/b40b5250-fa19-11ec-a73b-db9432f33999

CAPÍTULO 7
Conclusiones

Durante el desarrollo de este proyecto se han afianzado los conocimientos obtenidos con anterioridad en el desarrollo de videojuegos. Previamente, se habían realizado varios prototipos dentro del motor de Unity, además de un primer nivel de un juego completo para una de las asignaturas de la universidad. Es por ello que aunque ya se tenían conocimientos previos, ha sido necesario investigar y aprender diferentes funcionalidades aportadas por Unity de las que no se tenía constancia hasta el momento.

En cuanto a los ejercicios propuestos y presentados en el comienzo de esta memoria, se han conseguido en su gran mayoría, dejando tan solo de lado el último objetivo, el cual consistía en realizar una modalidad multijugador del juego, sin realizar por falta de tiempo, aunque su desarrollo está en proceso como parte de otro trabajo de final de grado. En cuanto al resto de objetivos, todos se ven reflejados en los otros dos TFG que componen el grupo en el que se ha realizado el videojuego. En lo referente a lo realizado para este trabajo en concreto:

- **Género de plataformas y sigilo:** Desde el principio del desarrollo del juego se tuvo este objetivo en mente, por lo que en todo momento los niveles se diseñaron siguiendo las características propias de estos géneros, permitiendo además diferentes formas de resolver algunos de los enfrentamientos, pudiendo elegir entre un enfoque más sigiloso o atacando de forma directa a los enemigos.
- **Enemigos reactivos a los estímulos:** Gracias al uso de las máquinas de estados para controlar el comportamiento de los diferentes enemigos presentes en el juego se ha conseguido cumplir este objetivo, permitiendo a cada uno de los *agentes* reaccionar de forma dinámica a los cambios en su entorno o de los movimientos y ataques realizados por el jugador.

Por último, aun y no formar parte de este TFG en concreto, también se ha cumplido el objetivo relacionado con la interactividad del entorno, permitiendo al jugador sentirse parte del mundo en el que se sitúa la acción. Esto se ha conseguido con elementos como los escondites o la posibilidad de apagar o encender las antorchas que se sitúan a lo largo de los escenarios.

CAPÍTULO 8
Relación con los estudios

Durante el transcurso de la carrera, ha habido varias asignaturas que se podrían relacionar con el proyecto realizado. En el transcurso de los cuatro cursos se han visto las bases de diferentes lenguajes de programación, y aunque en ninguna de las asignaturas obligatorias se ha utilizado C# (lenguaje utilizado por Unity) sí que han sido útiles los conocimientos generales aportados en **IIP: Introducción a la programación** o **PRG: Programación** utilizando Java, así como otras asignaturas más centradas en C.

Además, centrándonos en las asignaturas de mención cursadas, aun y que no están estrechamente relacionados, la asignatura de **TIA: Técnicas, Entornos y Aplicaciones de Inteligencia Artificial** ha sido útil a la hora de entender y realizar las máquinas de estados que se han utilizado para la inteligencia artificial de los enemigos. También ha sido bastante útil de cara al desarrollo del proyecto la asignatura de **SGI: Introducción a los sistemas gráficos interactivos** de cara a la parte más teórica del desarrollo de videojuegos, así como para conocer mejor el origen de los gráficos por ordenador y, por tanto, los videojuegos.

En el último curso se eligió la optativa de **IPV: Introducción a la programación de videojuegos**, en la cual se explicó los fundamentos a la hora de desarrollar un videojuego con Unity. Además, en esta asignatura se realizó el primero juego en Unity, ya que previamente tan solo se habían realizado varios prototipos con conocimientos obtenidos por cursos de internet.

Por último, resaltar la importancia que ha tenido la asignatura de **GPR: Gestión de proyectos** a la hora de aclarar como realizar un proyecto de este tamaño, así como la forma de orientarlo, estructurarlo y planificarlo.

CAPÍTULO 9
Trabajos futuros

A continuación nos centraremos en varios de los apartados que nos habría gustado incluir o mejorar dentro del proyecto en caso de haber dispuesto de más tiempo. Algunas de estas cosas harán referencia a cosas que se pretendían incluir dentro de la versión final del juego, mientas que otras serán cosas que nos gustaría añadir para proyectos futuros.

- Durante la primera etapa del juego se pretendían añadir varias funcionalidades que por falta de tiempo no se han acabado metiendo en la fase final. Estas serían dos nuevas habilidades que el jugador podría realizar en cualquier momento. La primera consistiría en un ataque a distancia similar al utilizado para encender o apagar antorchas, pero que permitiría dañar a los enemigos, mientras que la otra permitiría al jugador utilizar un gancho para mejorar la movilidad a lo largo de los niveles.
- Actualmente, el juego consta de un primer escenario a modo de tutorial y otros dos niveles en los que poner a prueba las cosas presentadas y superar los diferentes obstáculos. En un futuro se podrían añadir varios niveles más, aumentando la duración del juego y pudiendo añadir nuevas mecánicas a partir de los cuales estuviesen diseñados estos nuevos escenarios.
- Sería interesante también añadir nuevos tipos de enemigos con máquinas de estados diferentes, de forma que hubiese más variedad, haciendo el juego más entretenido y desafiante.
- Como se ha dicho anteriormente, uno de los añadidos pendientes consiste en un modo multijugador *online*. Este modo está siendo desarrollado actualmente como parte de otro TFG.

CAPÍTULO 10
Glosario

- **AAA:** Clasificación que se le da a los videojuegos producidos por una gran empresa, típicamente con marketing y desarrollo con un gran presupuesto y, por tanto, tomando grandes riesgos económicos, ya que requieren de una gran cantidad de ventas para ser rentables.

- **Agente:** Nombre que reciben cada una de las entidades que componen un sistema inteligente. Estos agentes deben ser capaces de reaccionar a los estímulos de su entorno de la forma esperada. En el caso de este videojuego, los enemigos se definirían como agentes.

- **Animator controller:** Permite ordenar los diferentes clips que conformarán las animaciones de la identidad a la que se asocie. Por medio de una máquina de estados se podrán controlar las transiciones entre dichos clips, pudiendo editar varios parámetros para cada transición, así como las condiciones que se han de cumplir para que se realice.

- **Animator:** Componente asociado al objeto que se pretende animar. A este componente se le deberá pasar un *Animator Controller* que definirá la máquina de estados usada.

- **Arcade:** Tipo de juego creado originalmente para salas de juegos recreativos, siendo necesario el pago de una moneda para poder jugar a los mismos.

- **Array**: Tipo de estructura de datos que almacena varios elementos del mismo tipo.

- **Assets:** Consiste en una representación de un objeto dentro del proyecto de Unity. Esto puede ser por ejemplo un modelo, ya sea 2D o 3D, o un archivo de audio, entre otras cosas.

- **Backlog:** Lista con todas las actividades que van a formar parte de un proyecto.

- **Boolean:** Tipo de datos utilizado en programación cuyo valor puede ser 'Verdadero' o 'Falso'.

- **Boxcast:** Función de Unity que genera un cuadrado y devuelve el primer *Collider* contra el que choca.

- **Brainstorming:** Método utilizado previamente al comienzo del desarrollo con el objetivo de generar la mayor cantidad de ideas posibles, algunas de las cuales acabaran desechándose.

- **Cartuchos ROM:** Método de almacenamiento utilizado, entre otros usos, en los juegos de las primeras consolas que salieron al mercado. Las siglas hacen referencia a *Read-only Memory*.
- **Collider:** Componente utilizado en los objetos en los que se quiere detectar colisiones con otros objetos. En el desarrollo de este proyecto se han utilizado de varios tipos dependiendo de la forma requerida. Entre ellos se encuentran el *BoxCollider* o *Capsule Collider* en forma de cuadrado o cápsula respectivamente.
- **Corrutina:** Fragmento de código que se ejecutará de forma paralela al código principal al que se le pueden añadir retardos en el caso de ser necesario.
- **Diagrama de Gantt:** Gráfica en la que se presenta la organización y tiempos estimados en el desarrollo de un proyecto.
- **Enumerador:** Tipo de datos en programación compuesto por varias cadenas de texto concretas en forma de constante, utilizado para mejorar la lectura y entendimiento del código, evitando el uso de números o cadenas escritas a mano que puedan causar confusión o errores
- **Feedback:** Opinión aportada por aquellas personas que han probado el videojuego, la cual ha sido utilizada para la corrección y mejora del mismo.
- **Indie:** Clasificación utilizada para referirse a aquellos videojuegos realizados por un estudio pequeño con un presupuesto modesto.
- **Inspector:** Ventana de Unity que permite la visualización y edición de las propiedades y componentes de un objeto.
- **JSON:** Formato de documento cuyas siglas significan *JavaScript Object Notation* utilizado típicamente para el intercambio de datos.
- **Kinematic:** Propiedad de los *RigidBody* que permite establecer si un objeto ha de ser afectado por el motor de físicas integrado en Unity. En caso de estar activado, las físicas no le afectarán.
- **Linecast:** Línea generada en la dirección establecida que devolverá el primer *collider* con el que colisione.
- **Máquina de estados:** Modelo de comportamiento formado por varios estados, así como las transiciones entre los mismos y las condiciones que se han de cumplir para activarlas.
- **Monobehaviour:** Clase básica de Unity que puede ser utilizada por cualquier objeto presente en una escena. Proporciona funciones y eventos estándar para dichos objetos, como puede ser la función *Start()* o *Update()*.
- **Online:** Utilizado para referirse a un juego multijugador a través de Internet.
- **Pathfinding:** Algoritmo cuyo objetivo es trazar el camino posible más corto entre dos puntos.
- **Plugin:** Complemento de una aplicación más grande que permite añadir diferentes funcionalidades extra con las que la aplicación no contaba de por sí.
- **Rigidbody:** Componente que permite que un objeto interactúe con el sistema de físicas integrado en Unity.

- **Raytracing:** Técnica utilizada para dar un mayor realismo a la iluminación mediante la simulación de los rayos de luz dentro de una imagen, así como su interacción con las superficies presentes en la misma.
- **Script:** Secuencia de código o comandos.
- **Scroll lateral** Videojuego cuya acción se realiza dentro de un mismo plano en el que el jugador podrá desplazarse principalmente hacia la izquierda o derecha.
- **Shooter:** Género que engloba a los videojuegos en los que se utilizan armas de fuego o se realizan acciones comparables en sus mecánicas a la de disparar.
- **Sorting Layer:** Método de ordenación aportado por Unity para los videojuegos 2D. Permite elegir el orden en el que se renderizan los objetos en pantalla, estableciendo que objetos se encuentra por delante del resto.
- **Spot:** Tipo de luz que ilumina en un ángulo establecido desde el inspector.
- **Sprint:** Intervalos pequeños de tiempo en los que se dividen las diferentes tareas de un proyecto realizado mediante metodología ágil.
- **Sprite:** Gráficos 2D utilizados para representar un objeto dentro de la escena de Unity. Se utilizan para todo el apartado visual del juego, desde los personajes hasta el escenario.
- **Sprite Renderer:** Componente utilizado en los objetos para asociar un *Sprite* a los mismos.
- **Tile:** Cada una de las celdas que conforman el escenario. A cada uno de ellos se le puede asociar un *Sprite*.
- **Tilemap:** Conjunto de *Tiles* que forma el escenario de cada uno de los niveles. Cada *Tilemap* puede contener varias capas para una mayor complejidad en el nivel.
- **Tile Palette:** Herramienta aportada por Unity y utilizada para la implementación de los *Tilemaps*.
- **Trigger:** Propiedad activable en un *Collider* que permite darle un nuevo uso. En caso de estar activado, el *Collider* no reaccionará a las colisiones, pero si las detectará, siendo esto utilizable mediante código.
- **Unity Asset Store:** Portal de Unity en el que se pueden descargar o comprar diferentes paquetes de *Assets*.
- **UT:** Cada una de las tareas en las que se divide el proyecto en la metodología ágil.
- **Waypoints:** Cada uno de los objetivos en los que el agente divide el camino a seguir durante el algoritmo de *Pathfinding*

Bibliografía

[1] **Standish Group**. Why Agile is Better than Waterfall [En línea] https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects/.

[2] **The History Of Gaming: An Evolving Community**. Riad Chikhani [TechCruch+] https://techcrunch.com/2015/10/31/the-history-of-gaming-an-evolving-community/.

[3] **The ultimate history of videogames**. Steven L. Kent [Crown; Illustrated edición].

[4] **Video Games Help People to Connect and Engage During COVID-19**. Raisa Santos [Health Policy Watch] https://healthpolicy-watch.news/video-games-helps-people-to-connect/.

[5] **Groundbreaking new study says time spent playing video games can be good your well**. Entrevista a: Professor Przybylski [University of Oxford] https://www.ox.ac.uk/news/2020-11-16-groundbreaking-new-study-says-time-spent-playing-video-games-can-be-good-your-well.

[6] **Before the Crash: Early Video Game History**. Mark J. P. Wolf [Wayne State University Press, 2012].

[7] **Unity Documentation**. Unity. [En línea] https://docs.unity3d.com/2021.2/Documentation/Manual/index.html.

[8] **Stack Overflow**. [En línea] https://es.stackoverflow.com.

[9] **Unity Asset Store**. Unity. [En línea] https://assetstore.unity.com/?gclid=CjwKCAjwquWVBhBrEiwAt1KmwuublLSALWo-KSP88ml3h_BhiB3XjW0Vo7ZI26n-3qZ4MtjBspqRwxoCh40QAvD_BwE&gclsrc=aw.ds.

[10] **Brackeys**. Github. [En línea] https://github.com/Brackeys.

[11] **BlackthornProd**. Github. [En línea] https://github.com/BlackthornProd.

[12] **Máquinas de estados para IA**. TheKiwiCoder Youtube. [En línea] https://www.youtube.com/watch?v=1H9jrKyWKs0&list=PLyBYG1JGBcd0091c1ZfX9ZN5oVUW7AFVy&index=3.

[13] **A* Pathfinding Project**. [En línea] https://arongranberg.com/astar/.

[14] **OpenGameArt**. [En línea] https://opengameart.org.

[15] **Blood, Sweat, and Pixels**. Jason Schreier [Harper Paperbacks, 2017].

APÉNDICE A
Controles

A continuación se mostrarán los controles básicos a tener en cuenta a la hora de jugar al videojuego:

	Teclas
Movimiento	A / D
Salto	Espacio
Agacharse	C
Embestida	Mayús
Escaleras de mano	W / S
Bajar de escaleras o plataformas de madera	Mantener S
Interactuar // Esconderse	E
Usar objeto	Click izquierdo
Inventario	1-6
Menú de pausa	ESC
Mirar hacia abajo	S estando quieto

Tabla A.1: Controles

APÉNDICE B
ODS

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS)

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**		X		
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**			X	
ODS 9. **Industria, innovación e infraestructuras.**			X	
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Durante las primeras fases del diseño y desarrollo de este proyecto se pretendían conseguir unos objetivos distintos a aquellos propuestos en estos objetivos de desarrollo sostenible. Es por ello que el juego no estará estrechamente relacionado con ninguno de los mismos, aunque si puede ayudar al cumplimiento de varios de ellos, tal y como podemos ver en varios de los apartados de la propia memoria y que se explicará a continuación.

El **ODS** con el que más está relacionado sería el de **Salud y bienestar**. En todo momento, uno de los objetivos principales a la hora de diseñar cualquier juego debe ser conseguir que aquella persona que lo vaya a jugar se divierta, lo cual afecta de forma directa a su bienestar. Además, como hemos visto en el apartado de la **Importancia de los videojuegos en el mundo contemporáneo**, existen numerosos estudios que relacionan de forma directa los videojuegos con el bienestar de los jugadores al cumplir ciertas condiciones, sobre todo en épocas en las que existen complicaciones para las conexiones sociales como ha sido el confinamiento. Es por ello que la existencia de juegos con los que poder desconectar durante unas horas o relacionarte con otras personas puede afectar de forma muy positiva en la salud mental de los jugadores. Además, aunque no sería en el caso de nuestro juego, con el paso del tiempo los videojuegos se están empleando cada vez más en procesos de rehabilitación, así como educación en clases de distintas edades, siendo capaces de ayudar con diferentes competencias tan importantes como lo puede ser la creatividad, así como enseñar de forma lúdica algunas asignaturas como puede ser historia, dando a los alumnos una motivación mayor a la que presentarían en muchos otros casos.

Otro de los objetivos de desarrollo sostenible con el que se podría enlazar el desarrollo del videojuego, aunque en menor medida, sería el de **Trabajo decente y crecimiento económico**. Esto es debido a que, como se ha explicado en la memoria del trabajo, la industria de los videojuegos es actualmente el sector de entretenimiento que más dinero genera, por encima de otros sectores como podría ser el cine. Además, este sector está todavía consolidándose, creciendo más cada año. Esto además afectaría a nuestro proyecto en el caso de continuar con su desarrollo en un futuro, participando en este sector en crecimiento.

Por último, otro de los objetivos que se podría relacionar con nuestro proyecto sería el ODS **Industria, innovación e infraestructuras**. Este objetivo estaría estrechamente relacionado con el anterior en cuanto al crecimiento económico que ha caracterizado a esta industria desde los comienzos a la actualidad. Además, podemos ver que es posiblemente una de las industrias en las que más importancia se le ha dado a la innovación debido, entre otras cosas, a los grandes saltos que se han producido en la tecnología en las últimas décadas. Esto es fácilmente distinguible al ver los grandes cambios que se han llevado a cabo en este sector desde sus orígenes.