Universitat Oberta de Catalunya

uoc.edu

Deeper into the Dungeon

Joaquim Gifre Rosales
Máster Universitario en Diseño y Programación de Videojuegos
Trabajo Final de Máster

Jordi Duch Gavaldà
Joan Arnedo Moreno

Fecha Entrega
6 de junio de 2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Deeper into the Dungeon*
Nombre del autor:	*Joaquim Gifre Rosales*
Nombre del consultor/a:	*Jordi Duch Gavaldà*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	06/2021
Titulación:	Máster Universitario en Diseño y Programación de Videojuegos
Área del Trabajo Final:	*Trabajo Final de Máster*
Idioma del trabajo:	*Castellano*
Palabras clave	*videojuegos, mazmorra, procedural*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El objetivo principal del proyecto es crear un juego perteneciente a género 'roguelike'. Eso significa que se quiere crear un juego de acción donde el jugador deba superar los diferentes retos de las salas de una mazmorra que se generará de forma aleatoria cada vez que inicie un nivel.
Para el desarrollo del juego, se ha trabajado siguiendo las pautas de la metodología ágil Scrum. Primeramente, se han definido las tareas a realizar en un 'backlog' y se ha definido los 'sprints' para que tuviesen una duración de una semana. Acto seguido, se han priorizado las diferentes tareas y se han asignado a los 'sprints'. En caso de no terminar una tarea, ésta pasaba al siguiente 'sprint'. Los nuevos requerimientos y bugs encontrados, pasan al 'backlog' y se asignan a los 'sprints'.
El resultado conseguido es un juego en el que el jugador puede desplazarse por los niveles generados de forma aleatoria, sorteando trampas y derrotando a enemigos con la finalidad de encontrar el objeto que le permita avanzar al siguiente nivel. Cada cierto número de niveles superados el objetivo cambia, apareciendo una sala con un jefe de nivel que debe ser derrotado para progresar.

La conclusión obtenida es que, a pesar de generar siempre niveles nuevos basados en el número de componentes disponibles, debido a que los retos ofrecidos por cada sala son asignados de forma aleatoria, la distribución de dichos retos es irregular y en ocasiones desbalanceada.

Abstract (in English, 250 words or less):

The main objective of this project is to create a roguelike game. This means that an action game is going to be made in which the player must overcome the different challenges of the dungeon rooms that will be generated randomly every time a level is started.
For the game development, the main guidelines from the agile methodology Scrum have been followed. First of all, different tasks have been defined and added to a backlog. Sprint's length has been defined to last for a week. Right after it, the different tasks defined for the project have been prioritized and assigned to the sprints. If a task is not finished by the end of the sprint, it is assigned to the next one. All the new requirements and bugs will be added to the backlog to be prioritized and assigned to the appropriate sprints.
The final result is a game in which the player can move through randomly generated levels, avoiding traps and defeating enemies in order to find the object that will allow him to advance to the next level. After beating a certain number of levels, the level goal changes, spawning in a room with a boss to be defeated in order to progress.

The conclusion obtained is that, despite generating new levels based on a certain number of available components, due to assigning randomly the challenges to each room, the distribution of these challenges is irregular and sometimes unbalanced.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Se quiere implementar un generador de niveles en 3D el cual cree niveles a partir de una serie de componentes predefinidos. Cada uno de estos componentes predefinidos, tendrá asociado un comportamiento diferente, el cual premiará al jugador o le ofrecerá un reto.
Debido a la necesidad de experimentar que los niveles generados sean disfrutables, se añadirán las diferentes mecánicas de movimiento y combate típicas de un juego del género ARPG.
Se ha decidido realizar este trabajo ya que, a pesar de que en el mercado se pueden encontrar muchos juegos de características y mecánicas similares, la gran mayoría de títulos son realizados en 2D.
Se quiere conseguir un generador de niveles aleatorios que sea capaz de colocar en la escena los diferentes componentes predefinidos, rotándolos para que encajen y descartándolos en caso de no poder ser colocado en el lugar especificado. La generación de niveles empezará por un nodo central e irá expandiendo los diferentes puntos de aparición del componente en cuestión.

1.2 Objetivos del Trabajo

Los objetivos que se proponen para este trabajo son los siguientes:

- **Creación de la mazmorra** de forma aleatoria basándose en componentes predefinidos.
- **Asignación de comportamientos** a los diferentes componentes que conformen el nivel: aparición del jugador, aparición de enemigos, aparición de trampas, aparición de cofres del tesoro, etc.
- **Creación de un personaje jugable** que pueda moverse e interactuar con el escenario generado.
- **Creación de los menús e interfaces** necesarios para diferentes situaciones del juego: menú principal, menú de pausa, HUD del juego, menú de ‘Game Over’, etc.
- **Creación de un sistema de juego en bucle** que permita ir avanzando niveles cada vez más extensos.
- **Inclusión de variedad** mediante la configuración de nuevos enemigos, componentes y trampas.

1.3 Enfoque y método seguido

El proyecto se ha desarrollado en tres etapas. Estas etapas se han correspondido a tres entregas que responden a las siguientes finalidades: entrega de una **versión parcial**, entrega de una **versión jugable** y entrega de la **versión final**.
Para el desarrollo de este proyecto, se ha partido de un proyecto nuevo y se han dividido los objetivos marcados para el trabajo en tres categorías:

- **Objetivos de implementación de mecánicas**: relacionados con la generación de niveles aleatorios, las mecánicas de movimiento y combate del personaje principal, la IA de los enemigos, las trampas y los objetos interactivos.
- **Objetivos de integración de mecánicas**: relacionados con la inclusión y coexistencia de todas las mecánicas y lógica desarrollada en los objetivos del punto anterior.
- **Objetivos de ampliación**: relacionados con la ampliación de variedad en contenido (componentes, enemigos o trampas).

Se quiere completar cada uno de estos grupos para la correspondiente entrega, de este modo, se pretende pasar de un proyecto de nuevo a uno con los cimientos de todas las mecánicas que se vayan a incorporar, luego se quiere trabajar la integración de las mecánicas para tener un juego funcional con todos los objetivos marcados y, por último, centrarse en la inclusión de nuevos elementos que se aprovechen de todas las mecánicas ya pulidas e integradas.

1.4 Planificación del Trabajo

Se quiere centrar el trabajo en la lógica del proyecto, es decir, en la programación de las diferentes mecánicas del juego. Por lo tanto, se van a utilizar recursos de terceros para los siguientes puntos:

- Modelos 3D: procedentes de la ‘Asset Store’ de Unity
- Elementos 2D para la interfaz (UI): procedentes de la ‘Asset Store’ de Unity.
- Animaciones: la mayoría de las utilizadas provienen de ‘Mixamo’, otras más sencillas se han realizado desde el propio editor de Unity.
- Pistas de audio: procedentes del portal opengameart.org
- Efectos visuales: procedentes de la ‘Asset Store’ de Unity.

En el punto de partida, las tareas que se considera que van a realizarse y se añaden al '*backlog*' del proyecto son las siguientes:

- **Creación de la mazmorra de forma aleatoria utilizando componentes predefinidos**. Se van a separar los componentes en tres tipos (habitaciones, pasillos e intersecciones). Deberá ser personalizable para que se pueda especificar, para cada componente, qué posibles componentes pueden usarse en el momento de expansión.
- **Creación de un personaje jugable que pueda moverse por el escenario generado**. Se añadirán animaciones y efectos de sonido al personaje. El movimiento se realizará mediante input continuo y con cámara encima del hombro.
- **Creación de los diferentes menús y elementos de interfaz**. Se añadirá la lógica suficiente para realizar la navegación entre escenas. Se diseñará también la plantilla del HUD.
- **Creación del punto de fin de nivel**. Si el jugador accede a dicho punto deberá aparecer el menú de fin de nivel y poder volver al menú principal.
- **Creación de los enemigos**. Se implementará un NPC que funcionará mediante una FSM ('Finite State Machine') y contará con animaciones y sonidos. Deberá poder moverse por dentro de su sala, perseguir al jugador y detenerse a una distancia en la que pueda atacar.
- **Creación de un sistema de salud y las mecánicas de combate**. Se añadirá la lógica necesaria para que, tanto el jugador como los enemigos, puedan recibir daño. En caso de que el jugador pierda toda la salud, deberá aparecer el menú de 'Game Over'.
- **Creación de trampas**. Se añadirá al juego un elemento que podrá infligir daño tanto a jugador como enemigos.
- **Creación de un sistema de asignación de finalidad aleatoria a los diferentes componentes de la mazmorra**. Esta finalidad, dotará a los componentes de retos o recompensas para el jugador.
- **Creación del bucle de juego**. El jugador deberá poder avanzar por diferentes niveles, cada vez más extensos. Al llegar a cierto nivel, deberá aparecer un jefe que deberá ser derrotado con tal de seguir avanzando.
- **Inclusión de variedad** para los componentes predefinidos que conforman la mazmorra como enemigos y jefes.
- **Corrección de errores y afinación.** Tareas de mantenimiento, corrección de errores y modificaciones de parámetros para tener el juego correctamente ajustado.
- **Tareas de documentación.** Mantenimiento de la página con el '*showcase*', redacción de los diferentes documentos para cada entrega y grabación de los diferentes vídeos.

Así pues, la planificación inicial de dichas tareas en el marco de tiempo establecido, se corresponde a la siguiente estimación:

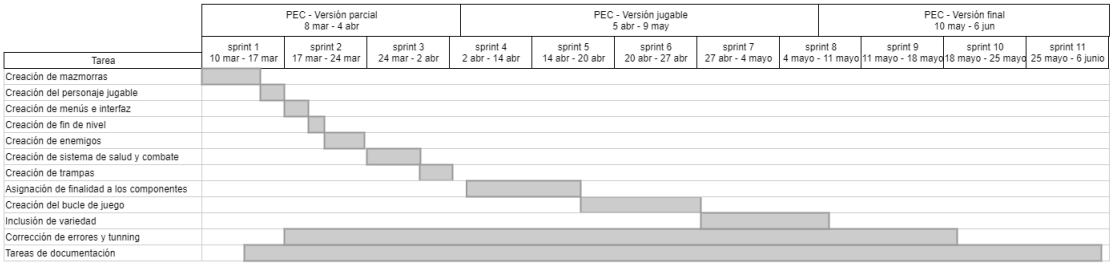

Ilustración 1 - Diagrama de Gantt, estimación inicial

Los hitos que se quiere conseguir para las diferentes entregas del trabajo son los siguientes:

- **Entrega de versión parcial:** se entregarán diferentes escenas de Unity en las que se podrá ver los diferentes objetivos de implementación de mecánicas. Se creará una escena para realizar las diferentes pruebas con la creación de niveles y otra para probar las mecánicas relacionadas con el personaje inicial, los enemigos, las trampas y los objetos consumibles.
- **Entrega de versión jugable:** se integrarán los diferentes componentes de las escenas de la versión parcial de modo que se pueda jugar de principio a fin.
- **Entrega de versión final**: se añadirán diferentes configuraciones para enemigos, trampas y componentes para la creación de niveles. Se trabajará para acabar de ajustar las diferentes mecánicas buscar y corregir los posibles bugs restantes.

Para la realización del proyecto, se ha utilizado un repositorio privado de GitHub, de donde se ha aprovechado la creación de un proyecto para ir asignando las tareas como '*issues*' en diferentes '*milestones*' (representación de los '*sprints*') para poder visualizar el trabajo mediante un Kanban.
Durante el desarrollo del proyecto, se han ido observando cambios en las necesidades y requisitos esperados para el propio juego. Estos cambios, se han añadido como tareas en el '*backlog*' y se han ido repartiendo en los '*sprints*' posteriores.

1.5 Breve sumario de productos obtenidos

Los diferentes elementos obtenidos una vez finalizado el proyecto han sido:

- Un **generador de niveles** aleatorios el cual es capaz de combinar diferentes componentes para expandir hasta cierta profundidad un nodo inicial. A cada uno de los componentes que forman el nivel, se le asigna una finalidad. Es la finalidad lo que decide si en un componente aparecerán, por ejemplo, enemigos o un tesoro.
- Un **personaje jugable** capaz de: moverse por el nivel, manejar la cámara para observar a su alrededor, interactuar con elementos del escenario, atacar, defenderse y esquivar.
- Unos **enemigos** que, mediante una *FSM*, pueden merodear por el entorno, perseguir al jugador y atacarlo.
- Unos **objetos consumibles** que recuperan puntos de salud u otorgan puntuación al jugador.
- Un tipo de **trampa**, disparada cuando un jugador o enemigo pasan a cierta distancia. Afecta a jugador y enemigos por igual.
- Un conjunto de **elementos de interfaz**. Dichos elementos incluyen: un menú principal, un menú de pausa, un menú de fin de nivel, un menú de fin de partida y ventanas de opciones, de créditos y un manual del jugador.
- Un **minimapa** que muestra la ubicación de los diferentes elementos del nivel y da un mecanismo de orientación al jugador.

1.6 Breve descripción de los otros capítulos de la memoria

- **Capítulo 2: Estado del arte**. Un breve repaso al género del videojuego y a algunas de las tecnologías disponibles para desarrollarlo.
- **Capítulo 3: Definición del juego**. Conceptualización del juego que se quiere desarrollar y las interacciones de los diferentes elementos.
- **Capítulo 4: Diseño técnico**. Explicación de las herramientas utilizadas y acerca de cómo se han desarrollado finalmente los diferentes elementos del juego.
- **Capítulo 5: Muestra de niveles**. Simulación de una iteración del juego, con explicaciones sobre los niveles generados.
- **Capítulo 6: Manual de usuario**. Explicación acerca de los objetivos del juego y los controles del jugador.
- **Capítulo 7: Conclusiones**. Conclusiones extraídas a partir del trabajo realizado.
- **Capítulo 8: Glosario**. Definiciones de vocabulario utilizado en el trabajo.
- **Capítulo 9: Bibliografía**.

2. Estado del arte

2.1 Revisión sobre el género del juego

El género '*roguelike*' es un subgénero de los juegos de tipo RPG que se caracteriza por la generación procedural de las mazmorras (niveles) en el que siempre se empieza por el nivel más bajo del calabozo con el equipo y habilidades más básicos.
En este tipo de juegos, hay la posibilidad de mejorar el personaje a medida que se avanza en el juego, aunque la muerte aparece como un hecho permanente. Es decir, la muerte del personaje implica que el jugador reaparezca en el nivel inicial del juego habiendo perdido todo el equipo y habilidades conseguidas en la partida anterior (aunque puede haber excepciones).
Con el tiempo, han ido apareciendo juegos que incluyen únicamente algunas de las diferentes características de este género y, este nuevo género, se llama '*roguelite*'.
A continuación, se introduce una serie de juegos que incluyen algunas de las características que se quieren introducir en el juego.

The Binding of Isaac (juego multiplataforma)

Este es un juego de acción de disparos en 2D en el que encarnamos a Isaac y debemos avanzar a través de los diferentes pisos del sótano, derrotando a enemigos y jefes de nivel mientras encontramos diferentes objetos que dan características y habilidades nuevas al personaje.
El juego consta de un mapeado basado en un sistema de cuadriculas que se van conectando y ofrecen al jugador diferentes retos o recompensas.
En caso de morir, el jugador pierde todas las mejoras y debe volver a empezar desde el principio.

Ilustración 2 - Gameplay del juego 'The Binding of Isaac'

Moonlighter (juego multiplataforma)

Este es un juego que pertenece a los juegos '*roguelite*'. Es un título de acción en 2D en el que encarnamos a un personaje propietario de la principal tienda del pueblo.
El juego consta de dos partes diferenciadas por el ciclo noche-día. Durante la noche, el jugador debe explorar mazmorras generadas de forma procedural y avanzar ciertos niveles para encontrar al jefe y derrotarlo.
Durante el día, el jugador debe hacer el papel de tendero y poner a la venta los diferentes objetos que haya encontrado en la mazmorra durante su expedición, fijando el precio justo para los clientes y llegando a vigilar los posibles ladrones.
El jugador debe reunir los materiales y objetos que encuentra en la mazmorra, teniendo que vigilar el inventario y priorizando unos elementos por encima de otros ya que unos le ayudarán a mejorar su armadura y armas y otros le darán más dinero para mejorar la tienda y el pueblo.
Podemos ver que cada mazmorra genera niveles basados en una temática propia. La disposición de dichos niveles sigue también forma de cuadricula. Cada una de las habitaciones ofrece diferentes retos al jugador y algunas veces incluyen hasta secretos.
En caso de morir, el jugador solamente pierde los objetos que haya conseguido en ese viaje a la mazmorra.

Ilustración 3 - Gameplay del juego 'Moonlighter'

Soul-Knight (juego para dispositivos móviles y Nintendo Switch)

Este es un juego de acción de disparos en 2D, se puede clasificar como un '*roguelike*' con toques de '*bullet hell*'. En él, se pueden seleccionar diferentes personajes antes de adentrarse a la mazmorra. Cada uno de estos personajes tiene una habilidad concreta que va a ayudar para superar el juego.

En este caso, el jugador encuentra diferentes armas esparcidas por los niveles y, tras superar ciertos niveles, puede mejorar alguna característica del personaje.

El diseño de sus niveles es aleatorio y está basado en diferentes temáticas. Las temáticas cambian una vez se ha avanzado cierto número de niveles y se ha derrotado al jefe final.

Las salas del juego están basadas en un sistema de cuadriculas y ofrecen diferentes retos o recompensas al jugador.

En caso de morir, el jugador pierde las mejoras aplicadas al personaje durante el transcurso de la partida además de sus armas.

Ilustración 4 - Gameplay del juego Soul-Knight

2.2 Revisión sobre la tecnología

Herramientas de control de versiones

Categoría de herramientas de software que ayudan a gestionar los cambios en el código fuente a lo largo del tiempo. Utiliza un tipo especial de base de datos para realizar un seguimiento de todas las modificaciones realizadas en el código.
En caso de cometer un error, siempre se pueden recuperar versiones anteriores de dicho código.
Algunas de las herramientas de control de versiones del mercado son:

- **Git**: es el sistema de control de versiones más moderno y común actualmente. Es un proyecto de código abierto con un mantenimiento activo desarrollado en 2005 por el creador del '*kernel'* del sistema operativo Linux, Linus Torvalds.

Ilustración 5 - Logotipo Git

- **Subversion (SVN):** es un sistema de control de versiones libre y de código abierto. Apareció a principios de los 2000 por CollabNet, Inc como sustituto del sistema CVS.

Ilustración 6 - Logotipo Subversion (SVN)

Pese a haber múltiples opciones en cuanto al control de versiones, para este proyecto se utilizará un repositorio Git privado hospedado en GitHub. Esta decisión se ha tomado debido a la experiencia acumulada con el sistema y a que el uso más extenso hace más fácil encontrar solución a los problemas que puedan aparecer.

‘Engines’ y kits de desarrollo

Actualmente hay multitud de entornos de desarrollo de videojuegos. Entre ellos, podemos encontrar: Unity, Unreal Engine, CryEngine, Amazon Lumberyard, Godot, etc.
De entre todo el abanico de posibilidades, se han considerado como probables tres:

- **Unity 3D**: es un motor de videojuegos creado por Unity Technologies que ofrece la posibilidad de desarrollar videojuegos tanto en 2D como en 3D. Con este motor se puede desarrollar para diferentes plataformas, incluyendo desarrollo para ordenadores, dispositivos móviles, consolas de sobremesa y entorno web.
  Los proyectos desarrollados con este entorno son programados con el lenguaje C#.
  Tiene una tienda dedicada con la que se pueden comprar y descargar diferentes componentes para el editor o recursos para facilitar la creación de videojuegos (modelos 3D, ‘*sprites*’, efectos visuales y sonoros, animaciones, etc.)

Ilustración 7 - Logotipo Unity

- **Unreal Engine**: es un motor de videojuegos creado por Epic Games que ofrece la posibilidad de desarrollar videojuegos en 2D y 3D. Este motor permite el desarrollo para diferentes plataformas, incluyendo desarrollo para ordenadores, dispositivos móviles, consolas de sobremesa y entorno web
  Los juegos desarrollados con este entorno, pueden ser programados mediante un sistema de ‘*visual scripting*’ (llamado ‘*blueprints*’) o directamente en C++. Los ‘*blueprints*’ son entendidos por el compilador como código C++.
  Dispone también de una tienda con la que se pueden comprar y descargar diferentes componentes tanto para el editor como recursos para facilitar la creación de videojuegos.

Ilustración 8 - Logotipo Unreal Engine

- **Godot**: es un motor de videojuegos de código abierto con licencia del MIT. Puede ser utilizado para el desarrollo de videojuegos en 2D y 3D.
  Los juegos desarrollados con este entorno, pueden ser exportados a la plataforma de PC, dispositivos móviles y entornos web.
  También tiene una librería desde la que se pueden descargar diferentes elementos para facilitar el desarrollo de videojuegos, añadiendo contenido de otros desarrolladores al motor.

Ilustración 9 - Logotipo Godot

Pese a las diferentes opciones, para el desarrollo de este proyecto se utilizará Unity 3D. Esta decisión viene respaldada por la experiencia y familiaridad que se tiene con el entorno y el lenguaje de programación asociado.
Unreal Engine a pesar de ser un motor muy potente, posee una curva de aprendizaje más pronunciada que otros entornos de desarrollo y no se dispone del tiempo suficiente para aprender acerca del entorno y desarrollar el proyecto entero.
Godot es un sistema gratuito cuyos módulos de terceros son también gratuitos. El problema principal del entorno, a parte del desconocimiento, es que es un entorno nuevo y poco conocido con mucha menos cantidad de información disponible en la red.

3. Definición del juego

3.1 Aspectos generales del juego

Para este proyecto, se va a desarrollar un juego en 3D con aspectos del género '*roguelike*'. Concretamente, los aspectos que se quieren emular de dicho género son:

- **Generación procedural de niveles**: a partir de diferentes componentes se quiere diseñar un algoritmo que, aleatoriamente, vaya interconectando diferentes componentes a partir de un nodo central.
- **Muerte permanente**: este aspecto se quiere suavizar en el juego de modo que, en caso de muerte, el jugador pierda únicamente el progreso conseguido en el nivel actual. Una vez se reinicie el nivel, este se generará de nuevo, de modo que el jugador deberá volver a explorar todo el nivel.

En cuanto al diseño de las mazmorras (niveles), se van a definir a partir de un nodo inicial y se irá expandiendo de forma similar al modo de búsqueda de un grafo mediante el algoritmo BFS ('*Breadth First Search*'). Indicándole al algoritmo una cierta profundidad, se aumentará el número de componentes instanciados entre el nodo inicial y los extremos.
El juego se diseñará pensando en la plataforma PC. Se decide el uso de dicha plataforma puesto a que es un elemento que suele estar en la mayoría de hogares.
Debido a la decisión de la plataforma a utilizar, se va a dirigir el diseño de los controles al uso de teclado y ratón.

3.2 Ambientación del juego

Se quiere diseñar un juego ambientado en un mundo de fantasía épica medieval. Este tipo de ambientación puede encontrarse en juegos del género RPG y, en ellos, se puede apreciar criaturas de diferentes especies y razas coexistiendo o enfrentándose en el mismo universo.
Esta ambientación, define como van a ser ciertos elementos del juego:

- **Los componentes de la mazmorra** tendrán las paredes de piedra, la iluminación se basará en antorchas y se dispondrá de elementos decorativos en las paredes (estandartes, por ejemplo).
- **Los enemigos** que aparezcan en el juego serán criaturas de fantasía, en este caso concreto, '*goblins*'. Las armas con las que ataquen al protagonista serán objetos romos, como huesos o porras.
- **El protagonista** del juego será un caballero, dotado de una maza y escudo, para hacer frente a los diferentes retos que se encuentre en el camino.
- **Las trampas** que aparezcan en el juego serán plataformas en el suelo que harán aparecer un conjunto de lanzas en cuanto sean activadas.
- **Los puntos de aparición del protagonista y de fin de nivel** vendrán indicados por runas y tendrán efectos visuales para llamar la atención del jugador.

3.3 Elementos del juego y sus interacciones

En este proyecto, van a aparecer los siguientes elementos:

- **Protagonista (héroe)**: avatar del jugador dentro del mundo del juego. Será quien deberá avanzar por la mazmorra y superar los diferentes retos generados.
- **Enemigos**: avatares NPC que aparecerán dentro de la mazmorra y se van a enfrentar al jugador (héroe).
- **Punto de aparición del jugador**: punto del nivel generado en el que aparecerá el jugador al empezar la partida. Se corresponde con el nodo central de la mazmorra.
- **Punto de fin de nivel**: punto del nivel al que deberá llegar el jugador para avanzar al siguiente nivel. Se encontrará en uno de los componentes más alejados del nodo inicial.
- **Trampas**: elementos del nivel los cuales deberán ser sorteados por parte del jugador para evitar sufrir daño.
- **Consumibles**: objetos que el jugador podrá recoger para beneficiarse de sus efectos.

Las diferentes interacciones que van a tener dichos elementos, pueden verse en la siguiente ilustración:

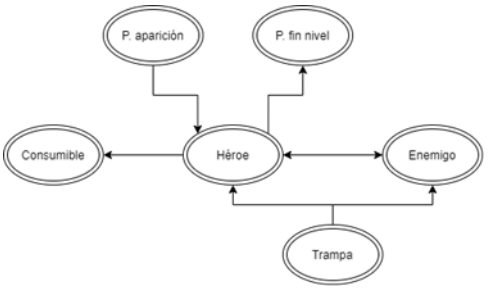

Ilustración 10 - Diagrama de interacción de elementos del juego

Se puede ver que dichas interacciones se resumen en:

- **Interacción del Punto de aparición**: hace aparecer al jugador al iniciar el nivel.
- **Interacciones del Héroe**: puede utilizar los objetos consumibles que encuentre en el nivel para recuperar salud, ganar puntos o activar el fin de nivel, puede interactuar con el Punto de fin de nivel para avanzar hasta la siguiente zona y derrotar a enemigos para conseguir puntos.
- **Interacciones del Enemigo**: pueden combatir al Héroe para derrotarlo y provocar el fin de la partida.
- **Interacciones de la Trampa**: pueden ser activadas tanto por enemigos o como el héroe. En ambos casos, infligirán sus efectos a quien se encuentre en la zona de afectación al activarse.

3.4 Objetivos planteados al jugador

El objetivo principal del juego es la exploración de un nivel generado aleatoriamente para conseguir puntos mientras va superando los diferentes retos ofrecidos, activar el portal para avanzar al siguiente nivel y conseguir llegar a dicho punto sin ser derrotado.
Cada vez que el jugador supere un nivel, la siguiente mazmorra generada será de extensión mayor, aumentando el número de retos generados y el tiempo que deba pasar el jugador explorando.
Los puntos se pueden conseguir de diferentes maneras:

- Abriendo cofres del tesoro.
- Recogiendo monedas soltadas por enemigos.
- Derrotando a los enemigos o al jefe de la mazmorra.
- Avanzando al siguiente nivel.

Una vez que el jugador haya avanzado cierto número de niveles, el objetivo del juego pasa de ser “encuentra el objeto ‘x’ para activar la salida del nivel”, a ser “derrota el jefe del nivel para seguir avanzando”.
En cuanto haya superado un nivel de jefe, el siguiente calabozo generado volverá a tener la extensión inicial, reiniciando de este modo el ciclo de generación de niveles.

4. Diseño técnico

4.1 Herramientas empleadas

Para el desarrollo de este juego se han utilizado herramientas para las siguientes finalidades:

- **Motor de videojuegos**: se ha utilizado Unity 3D para la creación del videojuego. Esto incluye el desarrollo de los scripts que contienen la lógica del juego, el diseño de los diferentes elementos y ventanas que aparecerán en la UI del juego y la creación de objetos y el manejo de sus animaciones.
- **Control de versiones**: se ha utilizado Git, mediante el servicio de GitHub. De este modo se ha ido trabajando en un repositorio, creando ramas para cada nueva '*issue*' o '*bug*' encontrado y llevándolo a una rama principal una vez estaba la tarea completada.
- **Monitorización del trabajo**: se ha utilizado el propio GitHub para establecer la lista de '*backlog*', mediante las '*issues*'. Con los '*milestones*' se han simulado los diferentes '*sprints*'. De este modo, en lugar de recurrir a otras herramientas, como por ejemplo Trello, se ha utilizado un mayor repertorio de las características ofrecidas por GitHub a la hora de mantener un Kanban, el '*backlog*' e ir aplicando la metodología de trabajo Scrum.
- **Edición de audio**: se ha utilizado el programa gratuito Audacity. Este es un programa que permite la edición de audio y se ha utilizado para exportar al mismo formato todos los archivos de audio utilizados en el juego. El formato en cuestión ha sido *.ogg.

4.2 Inventario y descripción de '*assets*' y recursos del juego

Animaciones

La mayoría de las animaciones que se encuentran en este proyecto provienen de **Mixamo**. Entre ellas, se hallan las animaciones relacionadas con el movimiento de los enemigos y el protagonista.
Además de estas animaciones, podemos encontrar:

- **Animaciones modificadas**: en un principio, se observaron anomalías en las animaciones relacionadas con andar y correr de los personajes. Para modificarlas, se utilizó el '*asset*' **UMotion**. De este modo, se arreglaron o modificaron a conveniencia '*frame*' a '*frame*' desde la aplicación de Unity.
- **Animaciones creadas desde el editor de Unity**: para el desarrollo de animaciones sencillas para elementos concretos del juego, como las trampas, el activador de fin de nivel o los cofres, se ha utilizado el editor de animaciones por defecto de Unity.

Dentro del proyecto, todos estos 'assets' se encuentran en la ruta '*./Assets/Animations/x*' donde 'x' se corresponde al elemento a quien pertenecen las animaciones.

Audio

Todos los archivos de audio que hay en el proyecto son procedentes del sitio web opengameart.org. Se encuentran todos los ficheros dentro de la ruta '*./Assets/Audio*'.
Además de encontrar en esta ruta los diferentes ficheros de audio utilizados en el juego, se puede encontrar un '*AudioMixer*' que contiene los diferentes canales utilizados ('*master*', '*music*', '*player*', '*enemy*' y '*effects*') expuestos para su manipulación posterior en la pantalla de ajustes del juego.

Interfaz (UI)

Para los diferentes elementos de la interfaz se ha utilizado elementos procedentes de dos fuentes diferentes:

- **Classic RPG GUI**: pack de '*assets*' procedente de la 'Asset Store' de Unity de *PONETI*. En él se encuentran los diferentes elementos que, combinados, son utilizados para el diseño de los diferentes menús, ventanas y elementos del HUD del juego. Se pueden encontrar en la ruta *'./Assets/Classic_RPG_GUI*'.
- **Procedentes de internet**: para algunos de los símbolos que aparecen en el minimapa y para las fuentes utilizadas, se ha recurrido a internet, para encontrar elementos gratuitos y con la adecuada licencia de uso para ser añadidos al proyecto. Estos elementos pueden ser encontrados en las rutas *'./Assets/Fonts/*' y './Assets/Images/'.

La textura que dibuja el contenido del minimapa se encuentra en la ruta '*./Assets/Minimap/RenderedTexture/*' y su contenido lo escribe una cámara en tiempo de ejecución.

Modelos 3D y efectos visuales (sistemas de partículas)

Los diferentes modelos que se han utilizado para la creación de todos los elementos que aparecen en el juego (componentes de la mazmorra, protagonista, enemigos, cofres, trampas, portales y consumibles) al igual que los diferentes efectos visuales (mediante sistemas de partículas de Unity) proceden todos del pack de 'assets' **PolygonDungeon** de *SYNTY STUDIOS.*
Este pack de assets se encuentra en el proyecto dentro de la ruta '*./Assets/PolygonDungeon/*'.

Scripts

Todo el conjunto de scripts desarrollados para cumplir con las diferentes mecánicas del juego se encuentran organizados dentro de la ruta '*./Assets/Scripts/'*.

Prefabs

En la ruta '*./Assets/Prefabs/*' se encuentran los diferentes elementos del juego que se han ido reutilizando dentro del proyecto. Estos elementos están formados por conjuntos de modelos, elementos de UI, scripts y otros componentes cuya procedencia es mencionada en otros apartados del capítulo.

Unity Packages

Hay en el proyecto diferentes '*packages*' de Unity. Entre ellos podemos encontrar los siguientes:

- **Cinemachine**: sirve para crear diferentes tipos de cámara virtual. Mediante estas cámaras virtuales, la cámara a la que sean asociadas, se comportará de la forma que haya sido configurada. Así se simplifica la creación de cámaras que sigan a un personaje, que se muevan con una entrada de input o que esquiven ciertos elementos de la escena.
- **Input System**: nuevo sistema de input creado por Unity. De este modo se abandona el anterior sistema que utilizaba 'Axis' configurados des de las preferencias del editor y se accede a un sistema que funciona mediante la declaración de acciones.
  A estas acciones se les puede asociar diferentes tipos de entrada y, cuando una de estas condiciones de entrada se cumple, se generan eventos que deben ser recogidos en un script.
  Es más sencillo de escalar y asociar a diferentes dispositivos.
- **TextMeshPro**: añade nuevas opciones para la implementación de elementos de la interfaz. Permite el diseño e inclusión de efectos a las fuentes utilizadas. Los elementos de este '*package*' son más nítidos que los elementos por defecto que ofrece Unity para la interfaz.
- **Universal Render Pipeline (URP)**: ofrece un entorno a partir del que añadir efectos de post procesado. Además del post procesado, añade la posibilidad de programar '*shaders*' de forma visual mediante el '*ShaderGraph*'. Para el proyecto se ha utilizado únicamente para añadir efectos de post procesado y embellecer el resultado final.
- **Unity – NavMeshComponents**: añade diferentes componentes con los que configurar los '*NavMesh*'. Permite la construcción y configuración de '*NavMesh*' en tiempo de ejecución.

De los diferentes '*packages*' mencionados, todos menos el de '*NavMeshComponents*' pueden ser encontrados dentro del '*Package Manager*' de Unity. Para encontrar otros '*packages*', como la expansión para los '*NavMesh*' se debe buscar en el repositorio oficial de GitHub de Unity y descargar la versión adecuada para la versión de Unity utilizada.

4.3 Explicación detallada de los componentes del juego

Generador de mazmorras

Este es el componente principal del juego. Se encuentra en la escena del juego y, en cuanto ésta carga, empieza a generar el nivel en base a ciertos parámetros de construcción.
Los parámetros públicos que afectan a la construcción del nivel y que por lo tanto se pueden modificar desde el editor son:

- **Nivel de profundidad de generación**: es el encargado de indicar la profundidad con la que se va a generar la mazmorra. Se entiende como profundidad a la distancia entre el nodo inicial (profundidad 0) a un nodo de los extremos (profundidad n).
- **Porcentaje de componente sin finalidad**: valor que puede ir de 0 a 50. Indica la posibilidad de que tiene un componente que acabe de ser instanciado en el nivel de no tener asignada una funcionalidad. El cálculo aplicado en el momento de utilizar este porcentaje hace que, a mayor profundidad, menor la posibilidad de instanciar un componente vacío.
- **Porcentaje bajo de activador de fin de nivel**: valor que puede ir del 0 al 30. Indica la posibilidad de que aparezca el objeto que activa el fin de nivel entre profundidad 1 y profundidad n-1.
- **Porcentaje alto de activador de fin de nivel**: valor que puede ir del 0 al 10. En esta ocasión, indica la posibilidad de que aparezca el objeto que activa el fin de nivel a profundidad n, siempre y cuando no haya aparecido anteriormente.
- **Nodo inicial**: '*gameObject*' que se corresponde a un componente (habitación) de la mazmorra. Este será el primero en ser instanciado, es decir, será el componente de profundidad 0.
- **Nodo jefe**: '*gameObject*' que se corresponde a un componente (habitación) de la mazmorra. Este componente aparece únicamente si la mazmorra generada es de profundidad máxima (en este caso profundidad 5). De cumplirse la condición, este componente aparece conectado a uno de los componentes de máxima profundidad.

En los siguientes subapartados, se va a hablar de: que son los componentes y en qué consisten, cómo funciona el método de generación y de las diferentes finalidades para los componentes y sus funciones.

¿Qué son los componentes?

Los componentes de la mazmorra son los '*gameObject*' que van a ser instanciados por el generador con la finalidad de crear el nivel del juego.
Los componentes se han generado pensando en tres tipos posibles: pasillos, encrucijadas y habitaciones.

- **Pasillo (CORRIDOR)**: componente de conexión que dispone única y solamente de dos salidas ('*spawnPoints*' o puntos de unión).

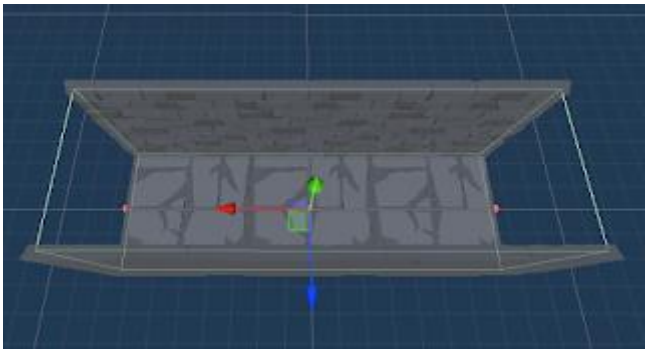

Ilustración 11 - Componente de tipo pasillo (CORRIDOR)

- **Encrucijada (JUNCTION)**: componente de conexión que dispone de como mínimo tres salidas ('*spawnPoints*' o puntos de unión).

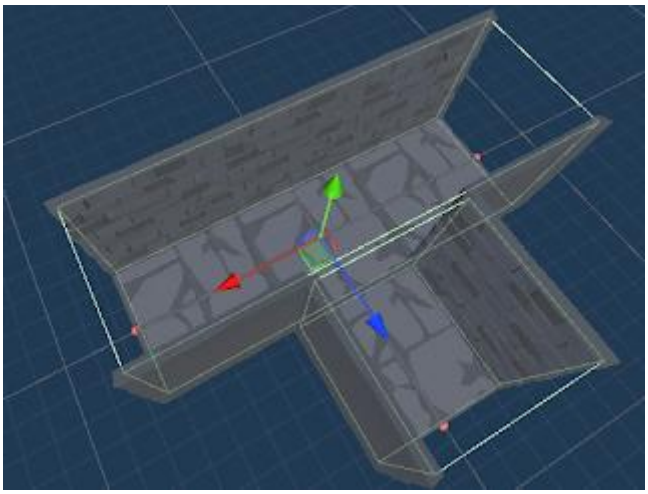

Ilustración 12 - Componente de tipo encrucijada (JUNCTION)

- **Habitación (ROOM)**: componente que va a tener como mínimo un punto de salida (‘*spawnPoint*’ o punto de unión). Van a ser de mayor extensión y dispondrán de finalidades que no se van a asignar a los otros tipos de componentes.

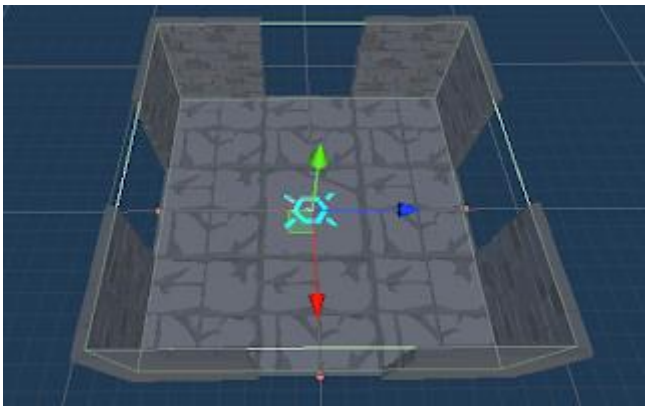

Ilustración 13 - Componente de tipo habitación (ROOM)

Este tipo de clasificación sirve únicamente para poder organizar los diferentes componentes creados. Cada componente es individualmente configurable y dispone de una lista de posibles componentes con los que expandirse de cada uno de los tipos mencionados.
Los componentes han sido diseñados para que no se expandan utilizando componentes del mismo tipo. A pesar de ello, se pueden encontrar a veces componentes del mismo tipo conectados porque se cumplen ciertas condiciones.
El método de instanciación y unión de los componentes será explicado en un apartado posterior.

¿Qué elementos forman los componentes?

Todos los componentes que se han generado tienen la misma composición. Esta composición se basa en un conjunto de ‘*GameObjects*’ hijos que sirven a modo de contenedor para los diferentes elementos (modelos, detectores, funcionalidades, etc.) y cada uno dispone de sus scripts para controlar aspectos concretos del componente.
Los elementos que forman un componente son:

- **Modelos (‘*Meshes*’)**: en este elemento se guardan de forma organizada en paredes, suelo, puertas y ‘*deadEnds*’, los diferentes elementos que forman la estructura general del componente.
  Contiene además el componente de tipo ‘*NavMeshSurface*’. Este componente nos permite generar el ‘*NavMesh*’ para este componente concreto, teniendo en cuenta la distribución de los elementos hijo.
- **Puntos de expansión (‘*SpawnPoints*’)**: en este elemento se guardan los diferentes objetos que serán utilizados para instanciar los nuevos componentes de la mazmorra.
  En él encontramos el componente ‘*DungeonComponentSpawner*’, encargado de expandir el componente en cuestión. Dispone de

información de los ‘*spawnPoints*’ disponibles y de los posibles componentes a instanciar.
Cada uno de los ‘*spawnPoints*’ dispone de información al respecto de que puerta o ‘*deadEnd*’ debe ser utilizado en caso de expandirse con éxito o fracaso y de uno de los posibles centros relativos.

- **Centro relativo del componente**: este elemento puede ser único o tener múltiples, esto depende de la forma que se dé al componente de la mazmorra.
  Este elemento nos sirve para poder orientar de forma adecuada los siguientes nodos del nivel, es decir, al instanciar un nuevo componente en un ‘*spawnPoint*’ se comprueba que ambos puntos de unión utilizados (del componente en expansión y del componente nuevo) formen un ángulo concreto.
- **Detectores de colisión**: este elemento alberga diferentes objetos que contienen un detector. El número de detectores dependerá de la forma que tenga el componente de la mazmorra, ya que deben cubrir toda la superficie posible.
  Estos detectores son un ‘*Collider*’ que funciona a modo de ‘*Trigger*’. Se encuentran en una ‘*Layer*’ específica que únicamente puede interactuar con elementos de ella misma.
  La función que tienen estos detectores es saber si un componente nuevo invade el espacio ocupado por otro.
- **Asignador de finalidad**: este elemento contiene diferentes objetos. Cada uno de ellos se encarga de realizar una función concreta (instanciar al jugador, poner trampas, cofres del tesoro, enemigos, el libro para activar el fin de nivel o el fin de nivel) y dispone de los recursos necesarios para cumplirla.
  Todos estos objetos se encuentran desactivados. El elemento tiene la lógica para activar uno u otro según unas condiciones concretas y las posibles finalidades asignadas.
  Las finalidades disponibles se encuentran en una lista de finalidades disponibles y se escogen de forma aleatoria para cada componente de la mazmorra. A pesar de ello, hay finalidades que se asignan a pesar de no encontrarse en la lista de disponibles por el hecho de cumplirse unas condiciones concretas, como por ejemplo encontrarse a máxima profundidad y no haber instanciado el activador del fin de nivel.
- **Ambientación**: este elemento contiene diferentes objetos que organizan los elementos decorativos del componente de la mazmorra. Podemos encontrar antorchas para la iluminación y piedras o estandartes para decoración.

¿Cómo se genera la mazmorra?

La generación de los niveles ocurre gracias a un conjunto de scripts distribuido en diferentes objetos. Dichos scripts son los siguientes:

- **DungeonGenerator**: es el script principal y se encuentra en un objeto existente de la escena. Se encarga de conocer la profundidad que se está generando y la profundidad máxima. Mediante dos listas, una de profundidad actual y otra de profundidad siguiente, se encarga de ir iterando e instanciando los diferentes componentes. Cuando llega a la máxima profundidad, llama a los componentes restantes para poner muros en las salidas disponibles. Funciona mediante corrutinas, esperando a que el componente actual acabe de instanciar todos los posibles componentes antes de avanzar a otro componente. Cuando acaba con la lista de profundidad actual, intercambia la de siguiente nivel con la actual y vuelve a llenar la de siguiente profundidad.
- **DungeonComponentSpawner**: es el script principal de los componentes de la mazmorra. Tiene conciencia de los puntos desde donde puede instanciar nuevos componentes, al igual que de los posibles componentes que pueden aparecer en cada posición. En todo momento sabe cuántos '*spawnPoints*' le quedan disponibles para instanciar los nuevos componentes. En caso de intentar expandir un punto que ya está en contacto con otro punto de otro componente, se marca como ya expandido y quedan conectados.
- **ComponentSpawnPoint**: se encarga de expandir un 'spawnPoint' concreto de un componente. Eso quiere decir que instancia uno de los posibles componentes, lo intenta colocar en todas las posibles posiciones y lo deja en una posición que quepa en el espacio disponible. Si no cabe en ninguna de las posiciones, lo intenta con otro componente. En caso de quedarse sin componentes disponibles, coloca una pared. Se encarga de comunicar a ***DungeonComponentSpawner*** si ha colocado componente o pared para que actualice los 'spawnPoints' disponibles. El método de colocación funciona con corrutinas para esperar a que los detectores de los componentes devuelvan la información de si están colisionando con otro componente o no.
- **DungeonComponentCollisionDetector**: este script controla si el componente está colisionando con otro componente. Funciona con corrutinas para poder controlar el tiempo de evaluación. Si pasado 'x' tiempo no ha colisionado, se marca como correcto y se notifica al ***DungeonComponentSpawner***. Se ha tenido que calcular con corrutinas porque los métodos 'OnTriggerEnter/Stay/Exit' no se disparan automáticamente, mediante corrutinas se puede poner un temporizador y dar tiempo de reacción.

- **DungeonComponentCenter**: este script tiene un método para devolver la dirección entre una posición y el centro del componente. Dicho centro es relativo al 'spawnPoint' usado. Se utiliza para colocar en la rotación correcta el nuevo componente que va a ir en un punto concreto.

La generación de la mazmorra sigue los pasos ilustrados en el siguiente diagrama:

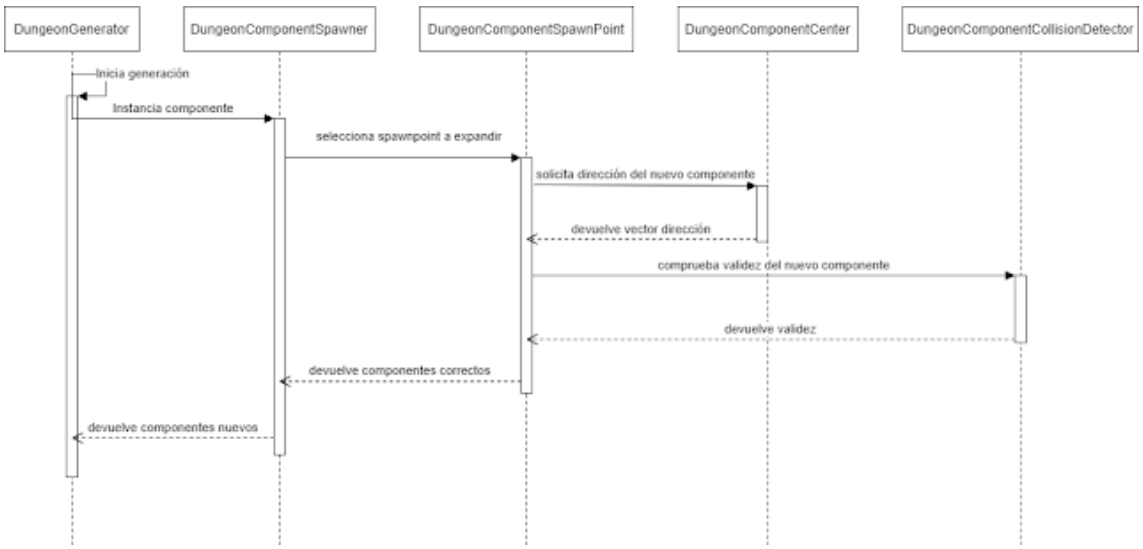

Ilustración 14 - Diagrama de secuencia de generación de mazmorra

Para explicar más detalladamente cómo funciona la colocación de los componentes, primeramente, se va a mostrar en la siguiente imagen la representación de los puntos de aparición y el centro relativo de un componente:

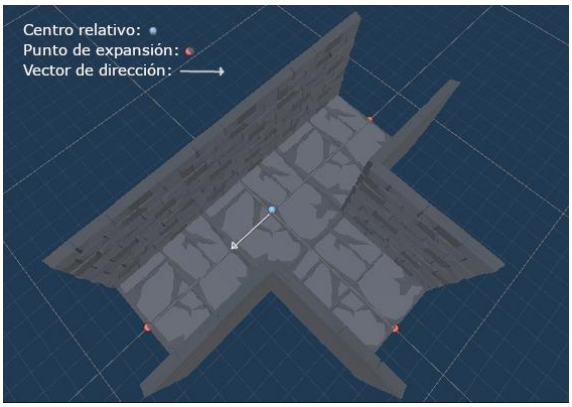

Ilustración 15 - Diagrama de componente de mazmorra

Antes de expandir un componente, se juntan las tres listas de posibles componentes, una para cada posible tipo (pasillos, encrucijadas y habitaciones), y además se mezclan. Esta mezcla se realiza porque se quiere un orden aleatorio a la hora de ir intentando la expansión de cada punto.
Cuando un componente va a ser expandido, los pasos que sigue son los siguientes:

1. Para cada punto de expansión disponible (aún sin expandir), se informan los posibles componentes para utilizar y se mezclan.
2. Se empieza la expansión con el primer elemento de la lista mezclada.
3. Para colocar correctamente el siguiente componente, se selecciona uno de los puntos de expansión de éste y se rota el nuevo componente para que el ángulo formado por el vector que va del centro al punto en expansión y el vector que va del centro del nuevo componente al punto seleccionado forman 180 grados. Esto implica que ambas salidas están alineadas y, por lo tanto, quedan correctamente.
4. Una vez que se ha rotado el componente nuevo, este es desplazado para que la posición de ambos puntos de expansión coincida.
5. En caso de que el nuevo componente ocupe el espacio de un componente existente, se selecciona otro punto de expansión y se repite desde el punto 3.
6. En caso de que el componente encaje correctamente, el punto que ha sido expandido activa la puerta que se corresponde a dicho punto y ambos puntos de expansión (el expandido y el seleccionado del nuevo componente) quedan marcados como puntos ya utilizados.
7. En caso de que se seleccionen todos los puntos de expansión del nuevo componente y no encaje en ninguna de las posiciones, se destruye el nuevo componente y se intenta con el siguiente de la

lista. Si no hubiera ningún componente más en la lista, se activaría la pared asociada a dicho punto de expansión.

Para que la colocación de los componentes funcione correctamente, se han utilizado corutinas. Esto ha añadido asincronía al proceso, pero era necesaria ya que la detección de colisiones de los nuevos componentes no se activa en el mismo '*frame*' que se ejecuta el código de instanciación.
Esto provoca que se deba añadir un retraso en la comprobación y que un evento lanzado por el detector de colisiones deba ser escuchado por el punto que se está expandiendo.
Cuando recibe el evento, el nuevo componente ha quedado validado y marcado como correcto o incorrecto.
Una vez todos los puntos han sido expandidos, el componente en expansión lanza un evento para indicar que ha finalizado de expandirse. El generador de mazmorra es el encargado de escuchar este evento y al recibirlo solicita al componente que se ha expandido todos los nuevos elementos instanciados. De este modo, estos nuevos componentes son añadidos a la lista de siguiente profundidad y se le asigna finalidad al componente que se acaba de expandir.
Hay un caso especial de unión de componentes que se produce si, en el momento de la expansión, el punto a expandirse se encuentra colisionando con el punto de expansión de otro componente validado que no esté siendo utilizado.
En este caso, en cuanto se va a expandir el nodo, se marcan ambos puntos en contacto como puntos utilizados y el punto que se está expandido coloca la puerta.
Una vez se han expandido todos los componentes posibles, para los elementos de máxima profundidad se lanza un script que coloca paredes en todos los puntos de expansión disponibles. De este modo, el nivel generado queda cerrado.

Asignaciones de finalidad

Los componentes de la mazmorra tienen un script llamado '**DungeonComponentPurposeAssigner**' mediante el que se asigna una de las posibles finalidades al componente.
El momento en el que se llama a dicho script para asignar la finalidad de un componente se corresponde con la finalización de su expansión. Cuando se avisa al script '**DungeonGenerator**' para indicarle que el componente en expansión ha terminado, además de recuperar los diferentes componentes añadidos, se le asigna una finalidad.
La asignación de finalidades depende de diferentes parámetros: profundidad actual, profundidad máxima, porcentajes de probabilidades, y si se ha instanciado el fin de nivel o el activador de fin de nivel. Principalmente, la asignación depende de la profundidad a la que se encuentra el componente.
Así pues, si el componente se corresponde al nodo inicial (profundidad 0) se le asignará automáticamente la finalidad de instanciar al jugador.

En caso de encontrarse en una profundidad entre 1 y n – 1 se mira lo siguiente:

- En caso de no haber instanciado todavía el activador de fin de nivel, se comprueba mediante la selección de un entero aleatorio si dicho valor entra en el porcentaje mínimo de aparición del activador de fin de nivel. De ser así, se le asignaría la finalidad de activar el fin de nivel.
  En caso contrario, se seleccionaría de forma aleatoria una de las finalidades que el componente tenga asignadas como posibles.
- En caso de haber instanciado el activador de fin de nivel, la finalidad asignada a un componente se escoge de forma aleatoria de entre las posibilidades que tenga asignadas dicho componente.

En caso de encontrarse en profundidad máxima (profundidad n), si no se ha asignado todavía un componente como fin de nivel, se asigna. El fin de nivel solamente puede ser asignado en profundidad máxima, por lo tanto, dicha finalidad se asignará automáticamente al primer componente de esta.
Una vez instanciado el final de nivel, las siguientes finalidades se asignarán de la siguiente forma:

- En caso de no haber instanciado el activador de fin de nivel, se comprueba la posibilidad de asignación mediante la selección de un entero aleatorio. Esta vez, se utiliza el porcentaje máximo de aparición del activador.
  Si entra dentro del rango de valores, se asigna la finalidad de activar el fin de nivel. De lo contrario, se selecciona de forma aleatoria una finalidad de la lista de posibles.
- En caso de haber instanciado el activador de fin de nivel, se asigna de forma aleatoria la finalidad de entre una de las asignadas como posibles.
- En caso de llegar al último componente de profundidad máxima y no haber instanciado aún el activador, se fuerza a que el último se corresponda al activador.

En caso de encontrarse en un nivel en que el objetivo sea eliminar el jefe de fin de nivel, el último componente instanciado tiene como finalidad posible la aparición de un jefe. Igual que en el caso de la aparición de jugador, hay unos componentes de mazmorra específicos que tienen únicamente dichas finalidades.

Cuando un componente se asigna la finalidad para activar el fin de nivel o para hacer aparecer el fin de nivel, lanza uno de dos posibles eventos. Estos eventos son recogidos por el '**DungeonGenerator**' para modificar unos parámetros booleanos de control. Estos mismos parámetros son los que se pasan al asignador de finalidad en el momento de decidir qué finalidad asignar a un componente expandido.
Las diferentes finalidades que pueden ser asignadas y su funcionamiento son las siguientes:

- **Aparición de jugador**: esta finalidad hace aparecer al jugador en una posición específica del componente. Dicha posición es configurable a través del editor. En este caso concreto, se ha ubicado dicha posición encima de la runa del centro del componente.
  El jugador aparecerá una vez el manager del estado del juego indique que el nivel ya ha sido cargado. Esto ocurre cuando el generador de mazmorra ha expandido el nodo inicial hasta la profundidad asignada y se han añadido paredes a profundidad máxima.

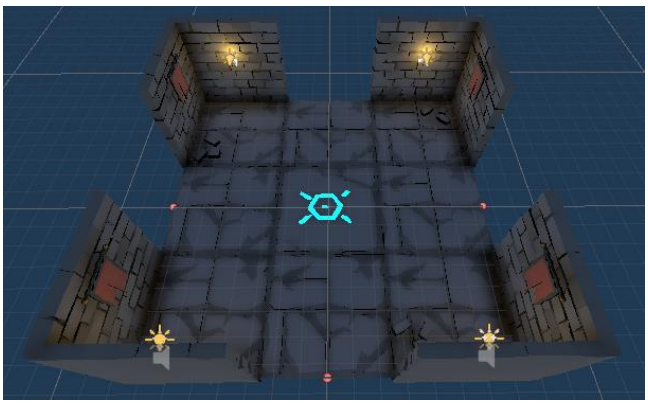

Ilustración 16 - Habitación que instancia al jugador

- **Aparición de enemigos**: esta finalidad hace aparecer cierto número de enemigos en el componente. La aparición de dichos enemigos se limita mediante dos valores.
  Un primer valor indica cuantos enemigos puede haber al mismo tiempo en el mismo componente. El segundo, indica el número máximo de enemigos que puede hacer aparecer.
  Una vez ha sido eliminado el total de enemigos instanciados concurrentemente, el '*spawner*' entra en '*cooldown*'. Pasado dicho '*cooldown*' se vuelven a instanciar enemigos.
  Esto ocurre siempre y cuando no se hayan hecho aparecer ya el total de enemigos del componente.

Los enemigos disponibles para instanciar se encuentran en una lista. El enemigo a instanciar, se selecciona de forma aleatoria de entre los elementos de dicha lista.
Lo mismo ocurre con la posición en la que aparecen los enemigos en la sala. Se selecciona de forma aleatoria de entre una lista de posibles posiciones.

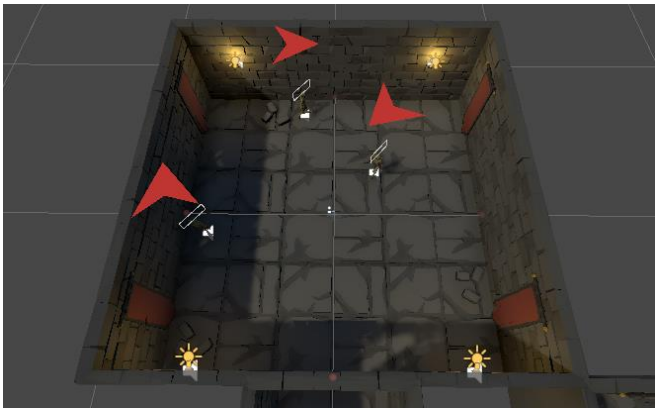

Ilustración 17 - Habitación instanciando enemigos

- **Aparición de trampas**: este comportamiento se puede asignar a cualquiera de los tres tipos de componentes. Su funcionamiento se corresponde a la asignación de una trampa aleatoria de entre las posibilidades de una lista en una de las posibles posiciones de aparición.
  El número de trampas posibles depende según el tipo de componente instanciado. Todo es configurable desde el editor.
  El número total de trampas que se van a instanciar en un componente es aleatorio y va de 1 al número máximo de trampas que pueda instanciar un componente concreto.

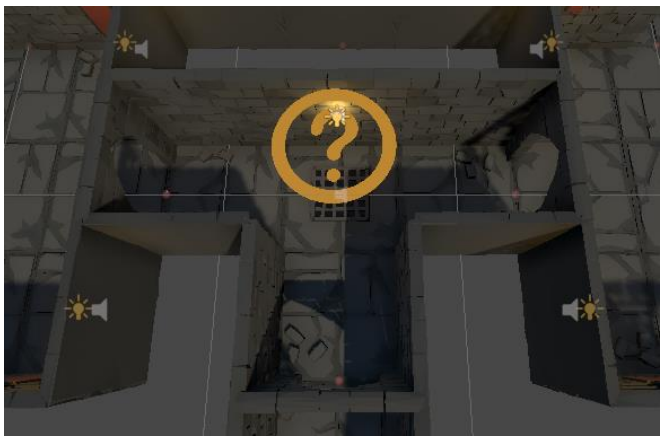

Ilustración 18 - Corredor instanciando trampa

- **Aparición de tesoro**: esta finalidad hace aparecer uno de los diferentes tipos de cofre disponibles en el componente. Dicho cofre ofrece una cantidad concreta de puntos al jugador dependiendo de la categoría del mismo.

Ilustración 19 - Habitación instanciando tesoro

- **Aparición de activador de fin de nivel**: esta finalidad hace aparecer un objeto consumible, es decir, el jugador lo utiliza si colisiona con él. Este objeto activa el portal de fin de nivel, permitiendo al jugador avanzar en la mazmorra.

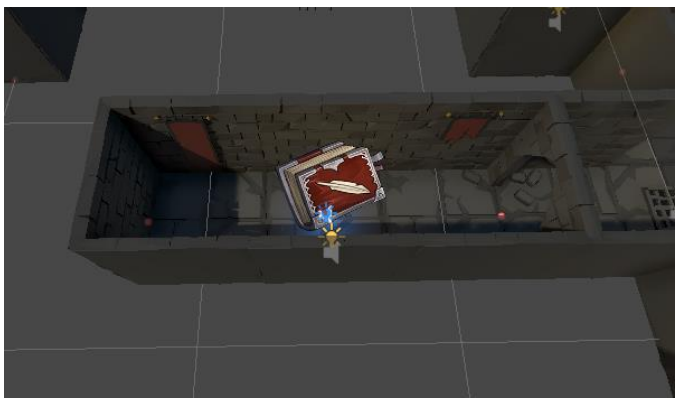

Ilustración 20 - Pasillo instanciando activador de fin de nivel

- **Aparición de fin de nivel**: esta finalidad aparece siempre en el primer componente de máxima profundidad. Una vez activado, permite al jugador avanzar al siguiente nivel del juego desde un menú que aparece cuando el jugador sube encima.

Ilustración 21 - Habitación con portal de fin de nivel

- **Aparición de sala del jefe**: esta finalidad aparece únicamente en el nivel de profundidad correspondiente a la máxima y lo hace únicamente en un componente.
  Esta finalidad hace aparecer un monstruo de fin de nivel el cual es más duro y grande que los enemigos comunes.
  Además, esta finalidad adopta aspectos del sistema de aparición de enemigos, ya que cuando la salud del jefe baja de ciertos límites, aparece una oleada de enemigos.
  Una vez derrotado el jefe, aparece una plataforma de fin de nivel ya activa. De este modo, el jugador puede seguir avanzando en la mazmorra.

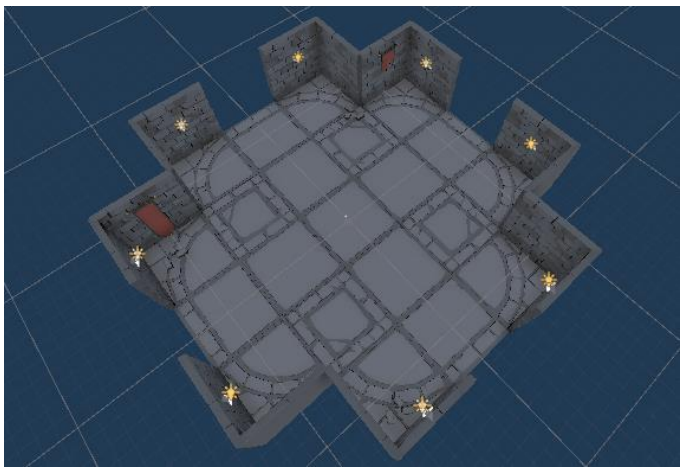

Ilustración 22 - Habitación de jefe de mazmorra

Personaje principal

El desarrollo del protagonista del juego se ha realizado mediante el uso de un componente de tipo '*CharacterController*'. Este tipo de componente ofrece métodos para mover el objeto y para saber si se encuentra en el suelo.
Para la interacción del jugador con el avatar, se ha utilizado el nuevo sistema de input de Unity. Mediante este nuevo sistema y un mánager que mantiene el control de los diferentes eventos lanzados, los diferentes scripts del personaje obtienen los inputs del jugador.
La cámara en el hombro implementada, ha sido implementada mediante 'Cinemachine'. Gracias a la cámara virtual de tipo 'FreeLook' y el sistema de entrada de Unity, se ha conseguido de forma sencilla una cámara que mire siempre por encima del hombro del personaje, que evite colisiones y que describa diferentes radios de movimiento dependiendo de la altura a la que se encuentra.

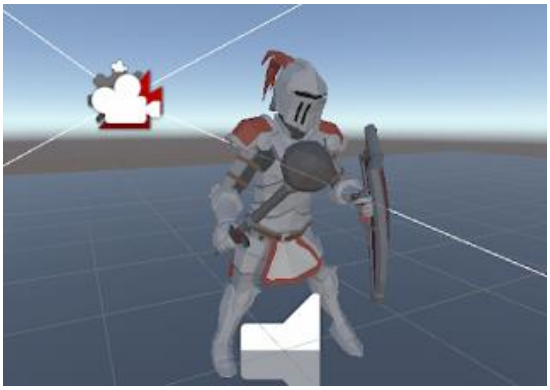

Ilustración 23 - Protagonista en animación 'Idle'

Las mecánicas implementadas para el jugador son las siguientes:

- **Movimiento**: el personaje se puede desplazar por el escenario. Este movimiento es calculado en función a la dirección en la que apunta la cámara, de modo que al avanzar siempre se moverá alejándose de ella. Además, el jugador puede incrementar la velocidad de movimiento y realizar un sprint.
- **Esquive ('*dash*'):** el jugador puede realizar una maniobra de esquive. De este modo, puede salir del alcance de los ataques enemigos o de las trampas. Consiste en el movimiento en una dirección concreta durante unos instantes de tiempo después de haber utilizado la acción.
  La dirección del esquive depende del input de movimiento entrado por el usuario en el momento de activarse. En caso de no estar introduciendo ninguna entrada de movimiento, la dirección por defecto del esquive es hacia atrás.
  Esta mecánica depende también de la posición en la que se encuentre mirando el jugador.

- **Bloqueo:** el jugador puede alzar el escudo del protagonista y protegerse de los ataques de los enemigos. Esta mecánica funciona mediante un '*Trigger'* colocado en frente del jugador.
  Cuando los enemigos entran en dicho '*Trigger'* quedan almacenados en una lista de 'objetivos cuyos ataques son bloqueables'.
  En caso de recibir el ataque de un enemigo, al ir a calcular el cálculo de daño, se comprobará primero si el jugador se estaba defendiendo y si el ataque proviene de un amigo de la lista de bloqueables.

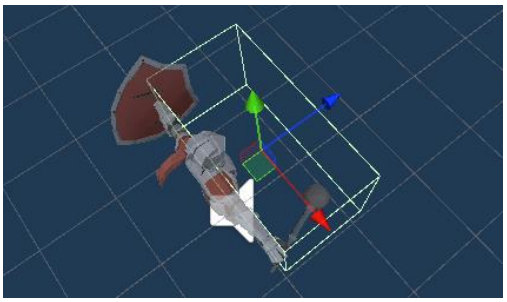

Ilustración 24 - Zona de bloqueo

Aun estando bloqueando los ataques, el jugador va a reproducir una animación de impacto. Esta animación es menos exagerada y más rápida que la de impacto normal.
Los ataques que provengan de enemigos fuera de la zona de bloqueo, impactarán sobre el jugador y le causaran daño aun encontrándose con el escudo levantado.

- **Ataque**: el jugador puede atacar para infligir daño a los enemigos. Se ha utilizado un par de animaciones conectadas con lo que se simula una combinación de golpes en caso de introducir múltiples entradas de ataque seguidas.
  Igual que en el caso del bloqueo, se ha añadido un '*Trigger*' que almacena los diferentes enemigos que entran dentro de este radio de acción.

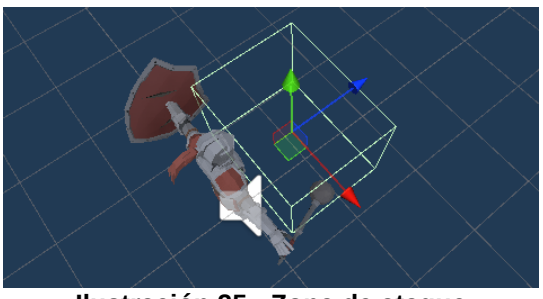

Ilustración 25 - Zona de ataque

Para el cálculo de daño, se ha añadido a las animaciones de ataque un evento. Este evento provoca que se llame a un método que recorre la lista de enemigos dentro del detector y les inflija daño a todos.

Para el correcto funcionamiento de las mecánicas de bloqueo y ataque, se han añadido comprobaciones para limpiar la lista de enemigos almacenados en las zonas.
Esto ha sido necesario ya que, durante el combate, los enemigos son eliminados y destruidos. Esto produce que haya un '*null'* en la lista de elementos y la mejor opción es eliminarlo de dicha lista.

Enemigos

Se ha desarrollado un tipo de entidad NPC que se enfrenta al jugador con la finalidad de reducir su salud a 0. Este tipo de NPC es el enemigo y su inteligencia artificial se basa en una máquina de estados finitos, de ahora en adelante FSM.
Que una IA se base en una FSM significa que:

- Su modelo de comportamiento tiene un conjunto de estados (finitos).
- Solamente puede encontrarse en un estado en un instante de tiempo.
- El estado en el que se encuentra determina la acción que va a realizar.
- Tiene un estado inicial.
- En cada estado, el conjunto de entradas (estímulos) que reciba influenciará en la transición al siguiente estado.

El modelo de FSM desarrollado para los enemigos del juego es el siguiente:

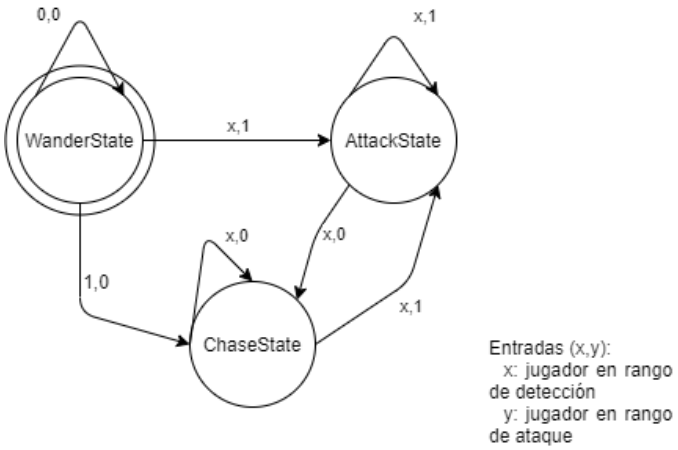

Ilustración 26 - FSM del enemigo: diagrama de estados

Como puede observarse en el diagrama, el conjunto de entradas que tiene en cuenta la FSM son: jugador en rango de detección y jugador en rango de ataque. A partir de estas dos entradas, la IA decidirá cómo reaccionar en todo momento.
Los valores '0' del diagrama se corresponden al valor booleano '*false*', los '1' se corresponden a '*true*' y las 'x' se corresponden a 'cualquier valor'.

Podemos observar que la IA de los enemigos consta de tres estados:

- **WanderState**: en este estado, el enemigo deambula por el componente. Para hacerlo, se escoge un punto aleatorio dentro de un radio y se le dice al componente '*NavMeshAgent*' que se dirija hacia allí.
  Antes de dirigir al agente hasta el punto designado, se comprueba que la destinación se encuentre dentro del '*NavMesh*' del componente y que sea alcanzable. En caso de no ser posible el desplazamiento hacia ese punto, se selecciona otro.
  El agente se desplaza hacia el punto escogido y, una vez alcanzado dicho punto, escoge otro.
  Este es el estado inicial y si el jugador no lo interrumpe, el enemigo se mantiene de forma indefinida en dicho estado.
  Si se diera el caso de que el jugador entra en el '*Trigger*' que funciona como detector de proximidad, el enemigo se da cuenta y pasa al estado de persecución.
  Si se diera el caso en el que el jugador entra en el '*Trigger*' que funciona como detector de rango de ataque, el enemigo pasaría directamente a atacar al jugador. Este caso no debería darse nunca ya que el detector de rango de ataque se encuentra contenido dentro del de proximidad, no se ha desarrollado enemigos que ataquen a distancia.
- **ChaseState**: en este estado, el enemigo persigue al jugador. Para hacerlo, establece la posición del jugador como posición objetivo del '*NavMeshAgent*'.
  El objetivo del enemigo en este estado es reducir la distancia que le separa del jugador para poder atacarle en cuanto entre dentro del '*Trigger*' de detección del rango de ataque.
  Mientras el enemigo se encuentra en este estado, su velocidad de movimiento es mayor a la de merodear.
- **AttackState**: en este estado, dado que el jugador se encuentra dentro del rango de ataque, el enemigo ataca. El funcionamiento del ataque del enemigo es similar al del jugador: este dispone de un '*Trigger*' que delimita el rango del ataque y un evento dentro de la animación del ataque aplica el cálculo de daño.
  En caso de que el jugador salga del rango de ataque, el enemigo pasa a perseguirlo hasta que consigue tenerlo de nuevo a alcance.

Hay una transición de estados forzada que puede devolver al enemigo al estado '*WanderState*'. La causa de esta transición es la muerte del jugador. Se ha añadido esta casuística para que, en caso de que muera el jugador, no haya un séquito de enemigos alrededor atacando sin descanso.

La FSM va controlada por un script general llamado '**EnemyFSM**'. Este script mantiene una variable con el estado actual y un conjunto de variables que se corresponden a los diferentes estados posibles.
Los estados heredan de una interfaz que les define:

- Un método que contiene la acción a realizar en el estado actual.
- Un método que contiene la comprobación para producir la transición a otro estado.
- Diferentes métodos para realizar la transición a los posibles estados.

En el script general se asigna el estado inicial y se inician los diferentes estados de la máquina y se realizan las suscripciones a los eventos necesarias.
A cada '*frame*', se llama a la realización de la acción del estado actual y a la comprobación para mantenerse en el estado o saltar a otro.

Ilustración 27 - Enemigo básico

Para ayudar con la realización del movimiento y el ataque, se han añadido scripts que realizan esas funciones específicas.
Así pues, en el script relacionado con el movimiento del enemigo, vemos que se dispone del '*NavMeshAgent*' y de métodos que reciben una posición destino y una velocidad.
Para el script relacionado con el ataque del enemigo, vemos que se dispone de los elementos necesarios para obtener la salud del objetivo que se encuentre dentro del rango de ataque y, en caso de poder atacar, se realiza la animación de ataque. El cálculo de daño se realizará a partir de un evento lanzado dentro de la propia animación. El ataque del enemigo tiene un tiempo de '*cooldown*' para no dejar al enemigo atacando ininterrumpidamente.
En cuanto el enemigo muere, se dispara una serie de eventos para:

- Añadir los puntos necesarios al jugador.
- Calcular si soltar un objeto y, en caso afirmativo, instanciarlo en el nivel.

Los enemigos disponen de una lista de posibles objetos a soltar una vez muerto: pociones de salud y monedas de oro.
En cuanto mueren, se hace un cálculo de probabilidad para ver si se debe instanciar un objeto consumible. En caso afirmativo, se selecciona de forma aleatoria si dicho objeto será una poción de salud u oro y se instancia en el nivel.

Hay otro tipo de enemigo diseñado para el juego: los jefes de fin de nivel. Solamente aparecen en el componente diseñado para dicha finalidad y aprovechan la gran mayoría de componentes utilizados para los enemigos básicos.

Ilustración 28 - Jefe de nivel

Las diferencias que hay entre el jefe y los enemigos normales se encuentran en:

- **Tamaño del modelo**: el modelo utilizado para el jefe tiene la escala modificada para que sea mayor que el del enemigo básico.
- **Tamaño de los detectores**: al ser un modelo mayor, el detector de rango de ataque es mayor. Además, el rango de detección también es mayor y ha sido ampliado por la parte posterior para que no pueda ser ‘pillado desprevenido’.
- **Valor de ataque y salud**: los valores de daño que puede infligir el jefe son mayores para añadir peligrosidad al enemigo. El valor de salud también es mayor.
- **Scripts de salud y muerte**: parten de la misma interfaz que los scripts del enemigo, pero la diferencia se encuentra en que pueden lanzar los eventos necesarios para activar las oleadas durante el enfrentamiento y para instanciar el portal de fin de nivel una vez derrotado.

Trampas

Se ha creado un tipo de trampa para el juego. Este, consiste de una plataforma colocada en el suelo del que sale un conjunto de lanzas al ser pisada.
En caso de alcanzar tanto al jugador o a enemigos en el momento de aplicar el cálculo del daño, se les quita una cantidad de puntos de salud.

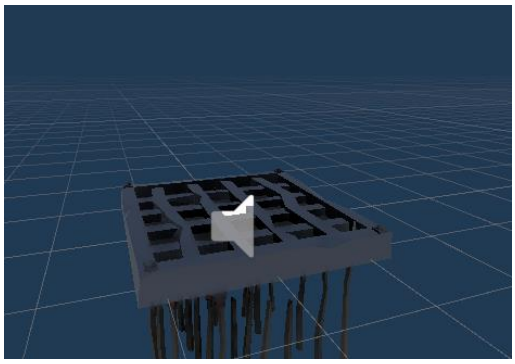

Ilustración 29 - Trampa desactivada

El detector de la trampa almacena una lista con los diferentes objetos situados encima suyo. De este modo, cuando la trampa es activada se recorre dicha lista y se aplican los efectos a los diferentes elementos.

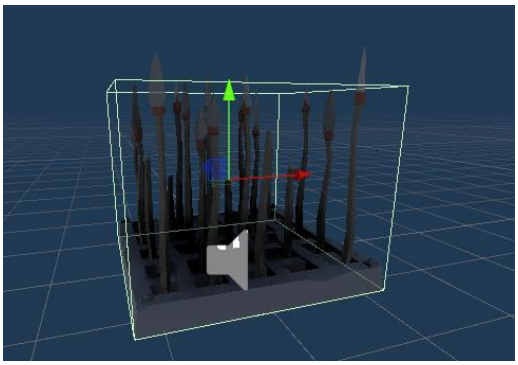

Ilustración 30 - Trampa disparada

La trampa funciona con un '*AnimationController*' propio al que se han añadido dos animaciones: una animación de activación y otra de recarga. Además, se incluye un estado de reposo (un estado vacío) para aguardar al momento de la animación.
Cuando se activa la trampa, lo primero que ocurre es que se reproduce un sonido. Este sonido quiere servir de '*feedback*' al jugador, indicando que alguna cosa ha ocurrido y que es mejor moverse. Acto seguido, se inicia la animación de activación, la cual tiene una ventana de '*wind-up*' antes de acabar por dispararse.
En el momento en que se dispara la trampa, se reproduce otro sonido, para dotar de efecto la acción de la propia trampa.
Cuando se ha reproducido la animación de activación, se reproduce la de recarga. En ella, las lanzas vuelven a entrar en la plataforma y quedan ocultas a la vista.

Para evitar que se activen muy frecuentemente, las trampas disponen de un tiempo de '*cooldown*' antes de poder ser disparada de nuevo. Al pasar dicho '*cooldown*' también se reproduce una pista de audio para indicar al jugador lo que ha ocurrido.
El cálculo de daño y la inicialización del tiempo de '*cooldown*' son llamadas mediante un conjunto de eventos colocados en las diferentes animaciones.
Dichos eventos se recogen en el script que hace de mánager de animaciones de la trampa y estos métodos llaman a las funciones pertinentes.

Cofres del tesoro

Este elemento aparece dentro de los componentes a los que se le ha asignado la finalidad de instanciar un tesoro. Tienen un detector de proximidad para saber si el jugador se encuentra dentro de un radio de acción.
En caso de estar dentro de este radio de interacción, el cofre muestra un elemento de interfaz que se encuentra siempre mirando a la cámara. Dicho elemento indica al jugador el botón a utilizar para interactuar con el cofre.
Una vez se interactúa con un cofre, este ofrece una recompensa en forma de puntos. Después de la primera vez, el cofre queda marcado como utilizado y no volverá a dar más puntos al jugador.

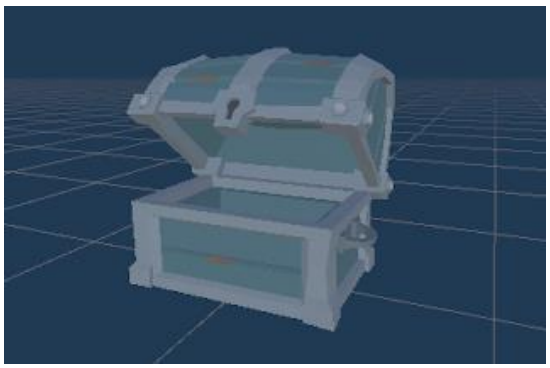

Ilustración 31 - Ejemplo de cofre de plata

Hay tres categorías de cofre: de madera, de plata y de oro. Cada uno de estos cofres da al jugador una cantidad diferente de puntos, siendo el de madera la cantidad más baja y el de oro la más alta.
Se ha añadido un '*AnimationController*' al cofre mediante el que se han diseñado las diferentes animaciones (de apertura y cierre) para el cofre. Dispone además de efectos sonoros para acompañar a la animación que se esté reproduciendo.

Portal de fin de nivel

Este elemento aparece en un componente de la mazmorra que se encuentre en la profundidad máxima. Siempre aparece en el primer componente analizado de dicha profundidad, sin importar el tipo de componente.

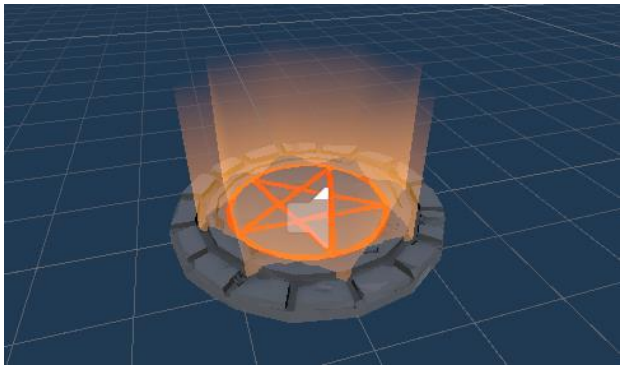

Ilustración 32 - Punto de fin de nivel

Si el jugador ha conseguido el objeto para activar el portal, el portal se muestra con un efecto de luz anaranjada saliendo del centro. En caso contrario, dicho efecto está desactivado.
Si el portal se encuentra activo y el jugador pasa por encima, activa un '*Trigger*' en el objeto que desata una transición de '*zoom-out*' de la cámara mientras se activan sistemas de partículas y un sonido. El punto final de los efectos lo pone la aparición de un menú de fin de nivel.
Desde este menú, el jugador puede decidir seguir explorando el nivel actual, avanzar al siguiente o volver al menú principal.
En cualquier caso, la activación del '*Trigger*' ofrece al jugador una pequeña cantidad de puntos por haber completado el nivel. Se ha controlado para que únicamente se ofrezcan dichos puntos una única vez.

Objetos consumibles

Se ha creado un conjunto de objetos que pueden ser recogidos por el jugador pasándoles por encima. Esta acción activa un '*Trigger*' que lanza un método para aplicar el efecto del objeto.
Los objetos consumibles aparecen en el nivel por dos motivos posibles:

- Se ha derrotado a un enemigo y se ha activado, en el momento de la muerte, la posibilidad de que dicho enemigo suelte un objeto.
- Se corresponde al objeto que activa el fin de nivel y aparece al asignarse una finalidad a un componente de la mazmorra.

Los enemigos pueden soltar dos objetos diferentes:

- **Monedas de oro**: este tipo de objeto otorga al jugador cierta cantidad de puntos.

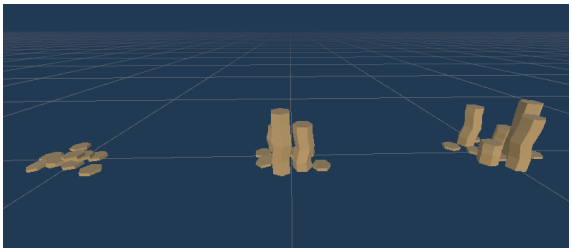

Ilustración 33 - Monedas de oro

- **Pociones de salud**: este tipo de objeto devuelve cierta cantidad de puntos de salud al jugador.

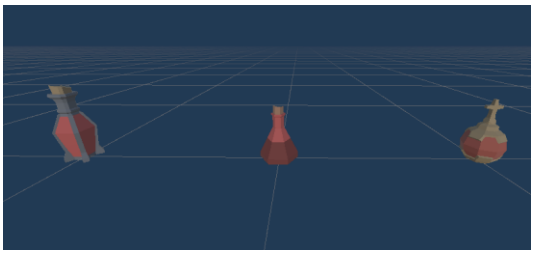

Ilustración 34 - Pociones de salud

En ambos casos, existen tres categorías de objeto: pequeño, mediano y grande. Cada una de estas categorías tiene un valor de efecto diferente. La categoría del objeto se selecciona en el momento de aparición del consumible y se le asignan de este modo el valor del efecto y el modelo que va a aparecer en el juego.
El objeto que activa el fin de nivel se diferencia de estos porque aparece al crear el nivel, en el componente cuya finalidad asignada se corresponde a hacer aparecer el activador.

Ilustración 35 - Activador de fin de nivel

El libro (activador) dispone de un '*Trigger*' que se utiliza para detectar que el jugador entra en contacto con él. Al hacerlo, se avisa al mánager que dispone de la información del estado del juego para indicar que se ha recogido el activador de fin de nivel.
El activador de nivel, dispone de un script que hace que el libro rote en el eje 'Y' a la vez que un '*AnimationController*' reproduce una animación sencilla creada con el editor de Unity para simular que el objeto está

flotando. Con estos dos elementos, además de una luz azulada y un sistema de partículas, se intenta hacer que el activador resulte lo más atractivo posible al jugador.
A todos los objetos consumibles se les ha añadido un componente de tipo '*AudioSource*' para poder reproducir un sonido cuando el jugador interactúe con ellos. De este modo, se quiere dar '*feedback*' e indicar que alguna cosa ha ocurrido.

Minimapa

A cierta etapa del desarrollo, se apreció la necesidad de un sistema de orientación mínimo que ayudase al jugador a ubicarse dentro del nivel. Por ello, se implementó un minimapa que aparece en la esquina superior derecha del HUD.

Ilustración 36 – Minimapa

Este elemento se ha implementado mediante el uso de una '*RenderedTexture*' y una segunda cámara añadida a la escena que se encarga de almacenar su salida en la textura. La cámara sigue al jugador en todo momento, aunque nunca altera su rotación. De este modo, la orientación del mapa siempre es la misma y permite ser utilizado como referencia en la navegación por el nivel.
Para poder mostrar dicho elemento en la interfaz, se ha creado un vacío al que se le ha añadido lo siguiente:

- Una **'RawImage'**: este tipo de elemento de UI puede recibir como textura de entrada una textura del tipo '*RenderedTexture*'. Así pues, este es el elemento que muestra el minimapa.
- Una **máscara**: este elemento será el padre de la '*RawImage*' y se le asignarán dos componentes de Unity: '*Image*' y *'Mask*'. Como '*Image*' se le asigna una imagen circular y gracias al '*Mask*' conseguimos darle al minimapa el aspecto circular.
- Una **imagen decorativa**: este elemento se corresponde a una '*Image*'. En este caso tiene forma de anillo y se ha posicionado para que decore el borde del minimapa.

Se ha añadido a los diferentes elementos del juego un '*Canvas*' con una imagen que sirve de icono para mostrar en el minimapa:

- **Flecha verde**: se ha asignado al jugador, este elemento rota con el jugador, con lo que en el mapa se puede observar la dirección en la que se está observando.
- **Flecha roja**: se ha asignado a los enemigos, este elemento rota con los enemigos, de este modo, se puede observar la dirección en que se desplazan los enemigos.
- **Interrogantes**: se ha asignado a los elementos de finalidad, de este modo, al activar uno, se muestra en el minimapa. No están incluidos en este icono el activador de fin de nivel ni el portal ni los enemigos.
- **Exclamación**: se ha asignado al portal de fin de nivel.
- **Libro**: se ha asignado al activador de fin de nivel.
- **Calavera**: se ha asignado al jefe de fin de nivel. Este elemento también rota dependiendo de la dirección en la que se mueve el enemigo.

Todos los iconos estáticos del minimapa (entendiendo por estáticos los interrogantes, exclamación y libro) contienen un script que rota el icono al activarse cambiando su '*transform.right*' por el '*Vector3.right*', de este modo, todos los iconos se muestran apuntando en la misma dirección.
Ciertas '*layers*' se han desactivado de la cámara principal para que no mostrara los iconos del minimapa, mientras que en la cámara del minimapa se ha desactivado las '*layers*' pertenecientes a los modelos de los enemigos y jugador.

Elementos de interfaz

En este apartado, se van a ver los diferentes menús y ventanas creadas para el juego, así como las funcionalidades de sus diferentes componentes.

Menú principal

Este elemento de la UI aparece en pantalla en cuanto se inicia el juego. Es un menú principal que consta de diferentes opciones: iniciar partida, opciones, manual de juego, créditos y salir del juego.

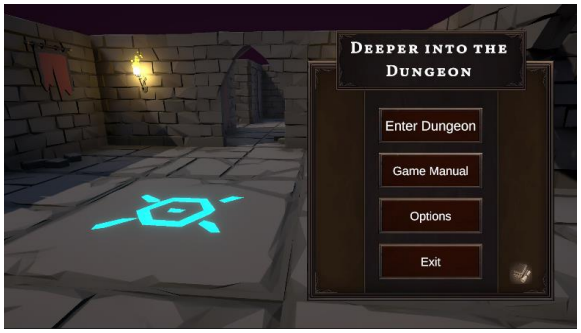

Ilustración 37 - Menú principal

Tal y como puede verse en la imagen, en el menú principal hay cinco botones:

- **Enter Dungeon**: se inicia la partida. Se le lleva a la escena de juego y empieza un nivel de profundidad 2.
- **Game Manual**: aparece en pantalla una ventana con la explicación de los objetivos del juego y los controles del protagonista.
- **Options**: aparece en pantalla una ventana con diferentes opciones de configuración, tanto gráficas como sonoras.
- **Exit**: se cierra el juego.
- **Credits (botón-imagen de libros)**: aparece en pantalla una ventana con '*scroll*' animado que muestra una serie de créditos y agradecimientos.

Ventana de opciones

Este elemento de la interfaz aparece en pantalla si se selecciona el botón de opciones. Esto puede ocurrir en el menú principal o durante la partida, desde el menú de pausa.

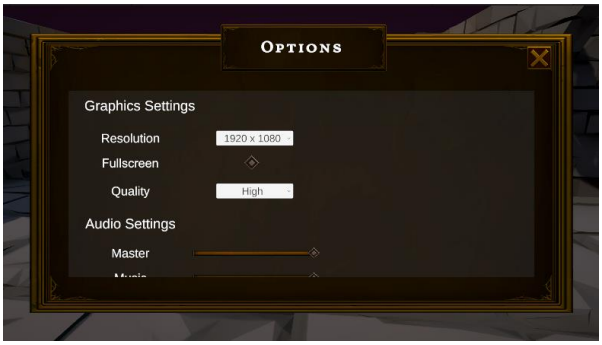

Ilustración 38 - Ventana de opciones

En las opciones gráficas, podemos encontrar:

- **Resolución**: elemento de tipo '*dropdown'* que permite seleccionar una resolución diferente para el juego. La lista de opciones de resolución se selecciona de entre las disponibles para la pantalla.
- **Pantalla completa**: elemento de tipo '*checkbox*'. Permite cambiar entre el juego en ventana y el juego en pantalla completa.
- **Calidad**: elemento de tipo '*dropdown*' que permite seleccionar una de las diferentes calidades que se encuentren en las preferencias del editor. Estas definiciones influyen en efectos como la iluminación, las sombras, las partículas, etc.

En el apartado de ajustes sonoros, encontramos diferentes '*sliders*' que sirven para controlar diferentes aspectos del '*AudioMixer'.* Las diferentes categorías de audio son:

- **Master**: afecta al volumen del audio en general.
- **Music**: afecta al volumen de la música y efectos sonoros ambientales.
- **Player**: afecta al volumen de los diferentes efectos relacionados con el jugador: pasos, bloqueo, daño y ataque.
- **Enemies**: afecta al volumen de los efectos relacionados con los enemigos: pasos, gritos de avistamiento, ataque y daño.
- **Effects**: afecta al volumen de los efectos relacionados con elementos del entorno: trampas, cofres y consumibles.

Los ajustes se guardan en variables dentro de ''*PlayerPrefs'* para que haya persistencia de los ajustes entre las diferentes sesiones de juego del jugador.
La ventana consta de 3 botones: una cruz en la parte superior derecha y un botón de descarte y otro de guardado en la parte inferior. Tanto el descarte como la cruz deshacen las modificaciones realizadas y cierran la ventana, mientras que al guardar los cambios prevalecen y se guardan en '*PlayerPrefs'*.

Ventana de manual de jugador

Se ha añadido una ventana para hacer la función de manual de juego. En dicha ventana se encuentra tanto la explicación de los posibles objetivos del juego como la explicación de los diferentes controles y las acciones que desencadenan.

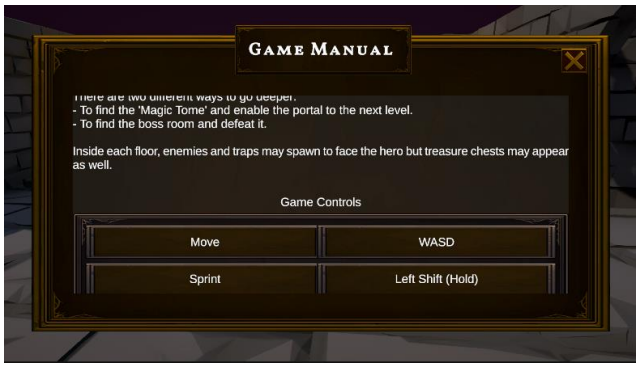

Ilustración 39 - Ventana de manual de juego

Dicha pantalla consta de un 'scroll' manual que contiene los diferentes elementos mencionados y además tiene un botón en cruz en la parte superior derecha que permite al jugador cerrar la ventana en todo momento.

Ventana de créditos

Se ha añadido una ventana para listar los créditos y agradecimientos del juego. Para conseguirlo, se ha añadido un '*scroll*' con el texto en cuestión y se ha añadido un '*AnimationController*' para crear una animación desde el editor de Unity que desplace el texto por la pantalla a una cierta velocidad.

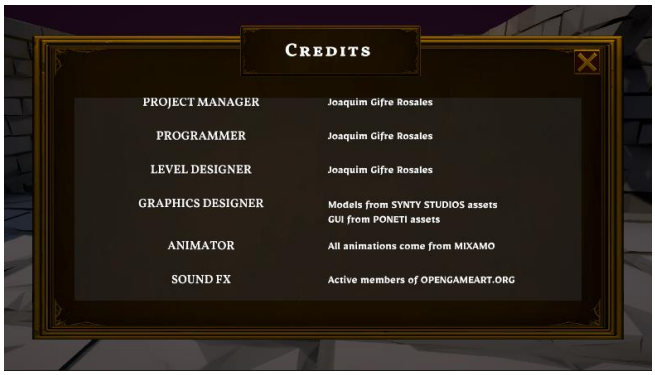

Ilustración 40 - Ventana de créditos

La ventana dispone de un botón en forma de cruz en la esquina superior derecha desde el que se puede cerrar la ventana. Al haber programado el controlador de la animación para que no tenga tiempo de salida, cada vez que se abre la ventana se lanza el '*Trigger*' que activa la animación, obligando a desplazar el texto desde el principio.

Pantalla de carga

Se ha añadido un elemento de interfaz que ocupa toda la pantalla y oculta al jugador todo lo que ocurre detrás. Se muestra al iniciar un nivel, mientras el generador construye la mazmorra.

Ilustración 41 - Pantalla de carga

Consta de un simple texto para decir al jugador lo que está ocurriendo, además de un 'slider' que se va actualizando según la profundidad actual que se esté expandiendo por parte del editor.
El último detalle añadido a la pantalla son las imágenes de tres engranajes. Estos han sido animados mediante un script que hace que vayan girando en un eje concreto.
Con estos elementos móviles, se quiere indicar al jugador que el juego está trabajando y no se ha quedado congelado.

Interfaz del juego (HUD)

Se ha añadido una interfaz sencilla durante los niveles para que el jugador disponga de toda la información que pueda necesitar.

Ilustración 42 - Interfaz del juego (HUD)

Se han añadido a la pantalla los siguientes elementos:

- **Indicador de profundidad**: este indicador se encuentra en la parte superior izquierda de la pantalla e indica el nivel en el que se encuentra el jugador. Este número va incrementando a medida que el jugador supera los niveles.
- **Barra de salud**: este elemento es un '*slider*' y muestra en todo momento la cantidad de puntos de salud que tiene el jugador. Se encuentra al lado del indicador de profundidad.

- **Minimapa**: este elemento está situado en la esquina superior derecha y muestra información en tiempo real de los alrededores del jugador.
- **Objetivo**: este elemento en forma de texto está situado debajo del minimapa. Está formado por diferentes textos, dos para los niveles cuyo objetivo es encontrar el libro y dos para los niveles con objetivo de derrotar al jefe.
  Uno de los dos textos es de color blanco, para indicar que todavía debe ser cumplido. En cuanto el objetivo del nivel es completado, se activa el segundo texto, de color verde y tachado, para indicar que ha sido cumplido.
  Al iniciar un nivel se comprueba qué tipo de objetivo va a haber en el nivel y se activa un conjunto de textos u otro.
- **Indicador de puntuación**: este elemento se encuentra en la esquina inferior derecha y muestra la cantidad de puntos de que dispone el jugador.
  Estos puntos pueden ser conseguidos de diferentes modos: derrotando enemigos, obteniendo monedas, abriendo cofres, entrando al portal de fin de nivel o derrotando al jefe.
  Estos puntos son acumulativos durante toda la partida, aunque en caso de reiniciar el nivel (desde el menú de pausa o de GameOver) se pierde la cantidad de puntos que se hubiese conseguido durante el transcurso de dicho nivel.
  En caso de reiniciar la mazmorra, el contador de puntos se reinicia completamente.

Además de estos elementos fijos en la pantalla, durante el transcurso del juego se pueden observar dos elementos más en la interfaz. Estos elementos aparecen puntualmente y se corresponden al letrero que aparece encima de los cofres para recordar al jugador el botón de interacción y las barras de salud de los enemigos.

Menú de pausa

Se ha añadido la opción de pausar el juego. Esto se hace mediante la modificación del parámetro '*Time.timeScale*', en caso de valer 1 el juego se mueve a velocidad normal, pero si se reduce a 0 se detiene el movimiento de todos los elementos del juego.
Cuando esto ocurre, aparece un menú de pausa desde el cual el jugador puede: reanudar el juego, reiniciar el nivel, reiniciar la partida, acceder al menú de opciones y volver al menú principal.

Ilustración 43 - Menú de Pausa

Los diferentes botones añadidos cumplen las siguientes funciones:

- **Resume**: permite reanudar la partida desde el propio menú. Eso oculta el menú de pausa y devuelve el juego a su estado normal.
- **Restart Depth**: permite reiniciar la profundidad actual. Esto quiere decir que el jugador va a perder la cantidad de puntos ganada durante ese intento de superar el nivel.
  Además, a consecuencia de reiniciar la profundidad, se genera también un nuevo nivel, con lo que el jugador deberá enfrentarse a retos distintos.
- **Restart Dungeon**: permite reiniciar el juego, es decir, el jugador vuelve a la profundidad inicial y se reinicia su puntuación completamente.
- **Options**: permite abrir la ventana de ajustes desde dentro del juego, de este modo se permite cambiar la configuración de los diferentes parámetros mientras se juega.
  Al igual que si se guardaran las modificaciones de los ajustes en el menú principal, los nuevos valores se almacenan en '*PlayerPrefs*'.
- **Back to Menu**: permite al jugador volver al menú principal. Todo el progreso de la partida se pierde.

Menú de fin de nivel

Este elemento de la interfaz aparece en pantalla en caso de que el jugador haya activado un portal de fin de nivel y haya interactuado con él.
Las opciones que aparecen en este menú dan opción al jugador de: seguir explorando el nivel actual, avanzar al siguiente nivel y volver al menú principal.

Ilustración 44 - Menú de fin de nivel

Las acciones realizadas por los diferentes botones son las siguientes:

- **Keep Exploring**: oculta el menú y devuelve el control del protagonista al jugador. De este modo, se ofrece al jugador la posibilidad de seguir explorando el nivel y conseguir más puntos.
- **Go Deeper**: avanza al siguiente nivel. Este hecho produce que se incremente la profundidad de la mazmorra, haciendo que el siguiente nivel sea de mayor extensión.
  En caso de avanzar un nivel después de haber derrotado al jefe, el siguiente nivel tendrá la profundidad inicial. Este botón permite básicamente mantener vivo el ciclo del juego.
- **Back to Menu**: acaba con la partida actual y transporta al jugador al menú inicial.

Menú de fin de partida (Game Over)

Este elemento de la interfaz aparece en pantalla en caso de que el jugador pierda todos sus puntos de salud. En ese caso, el jugador puede reiniciar el nivel, reiniciar la partida o volver al menú.

Ilustración 45 - Menú de fin de partida

Las acciones que están asociadas a los diferentes botones del menú son:

- **Restart Depth**: esta opción reinicia el nivel actual. La única pérdida para el jugador se corresponde a los puntos conseguidos durante el anterior intento.
- **Restart Dungeon**: esta opción reinicia la partida del todo. Eso implica que el juego se inicie desde 0, con un nivel de profundidad mínima y la puntuación del jugador reseteada.
- **Back to Menu**: esta opción permite al jugador salir al menú principal. Se pierden todos los progresos de la partida.

Managers

Durante el desarrollo del juego se ha visto que había cierta información que era requerida o necesaria en diferentes elementos del propio juego.
Para manejar de forma eficiente y controlada dicha información, se han generado diferentes scripts a modo de mánager para cumplir con este propósito.
La característica principal de los siguientes scripts es que aplican el patrón '***Singleton'*** para mantener una única instancia dentro del juego, permitiendo a los diferentes elementos del juego acceder a una instancia estática, con la que usar los diferentes métodos públicos disponibles.

AtmosphereAudioManager

Este script pertenece al objeto que controla el audio ambiental del juego. Se utiliza para poder modificar el sonido ambiental, de la música del menú principal a los ruidos de la mazmorra, desde diferentes sitios.
De este modo al cargar las escenas se puede modificar, en caso de necesidad, la pista de audio que se escucha.
Al ser utilizado el patrón '*Singleton*' el objeto no se destruye entre escenas y esto ayuda a que el sonido ambiental no cese entre la carga de distintos niveles.

GameStateManager

Este script sirve para controlar los estados en los que se encuentra el juego. Gracias a él, se puede obtener desde diferentes componentes del juego información que resuelve las siguientes dudas posibles:

- ¿Está pausado el juego?
- ¿Se ha activado el fin de nivel?
- ¿Se ha muerto el jugador?
- ¿El jugador se encuentra en la plataforma de fin de nivel?
- ¿Se ha construido el nivel?

Además, dispone de la información necesaria para contabilizar correctamente la puntuación del jugador y para crear de forma correcta los diferentes niveles.
Tiene la información acerca de la profundidad actual, la profundidad mínima o inicial y la profundidad máxima a generar. Por lo que es el encargado de decir al generador de mazmorras que empiece a crear el nivel.
Este script sirve también de puente para los elementos de la interfaz, encargándose de llamar al encargado de la interfaz para pedir que actualice los valores del nivel actual, puntuación u objetivo del jugador.

InputManager

Este script dispone de los métodos necesarios para obtener información acerca del input introducido por el jugador. De este modo se encuentra centralizado.
Mantiene una referencia de los scripts del jugador que utilizan el sistema de input de Unity y asigna los eventos producidos por dicho input a las funciones correspondientes del script adecuado.
Al tener que crear un objeto 'jugador' nuevo en cada nivel, las asignaciones de los diferentes eventos deben ser eliminadas para el anterior jugador y realizadas de nuevo para el actual.

InGameUxManager

Este elemento mantiene una referencia de los diferentes elementos de la interfaz y permite acciones como:

- **Activación/desactivación de elementos**: permite activar o desactivar los diferentes menús y ventanas de UI para que sean mostrados o sigan ocultos.
- **Actualización de valores**: actualiza valores mostrados en elementos de la interfaz.
- **Centraliza la vuelta al menú**: llama al input manager para limpiar las suscripciones, fuerza que el juego no esté en pausa y carga el menú principal.

Mayormente, distribuye las peticiones que le llegan de otros elementos a los elementos conjuntos de UI correspondientes.

5. Muestra de niveles

En este capítulo se comparten capturas de pantalla de la vista general de los niveles generados en un ciclo de juego. Se considera un ciclo de juego a los niveles generados desde la profundidad inicial a la profundidad máxima (con jefe de nivel).
Para ilustrar dichos niveles, se utilizarán imágenes tomadas desde el editor de Unity, con lo que se podrá apreciar tanto el nivel generado como los diferentes iconos del minimapa. Estos iconos ayudarán a entender el nivel generado.

El nivel generado a profundidad 2, es decir, con dos expansiones entre los componentes más alejados y el nodo central es el siguiente:

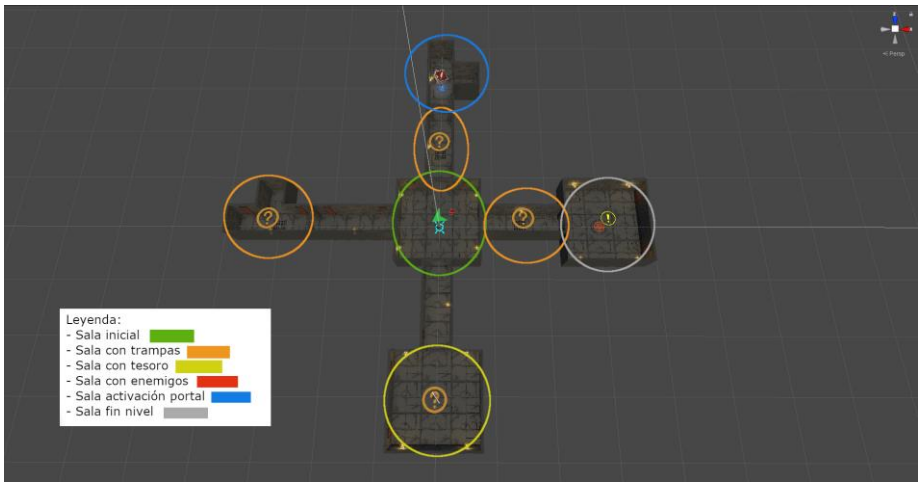

Ilustración 46 - Ejemplo de nivel a profundidad 2

Como puede verse en la imagen anterior, el jugador aparece en un nodo central. A esta profundidad, el objetivo principal del jugador es conseguir el libro que activará el portal de fin de nivel y luego alcanzar el fin de nivel.
Para conseguir dicho objetivo el jugador deberá seguir los siguientes pasos:

- Dirigirse ‘hacia el norte’ y obtener el libro que active el fin de nivel.
- Desandar el camino hasta llegar al nodo central.
- Dirigirse ‘hacia el oste’ y entrar en el portal de fin de nivel.

En caso de querer conseguir más puntos, el jugador puede dirigirse primero ‘hacia el sud’ e interactuar con el cofre que se encuentra en ese componente.

Al superar el nivel, el jugador aparecerá en un nuevo nivel, ahora de profundidad 3. En este caso, el nivel generado ha sido el siguiente:

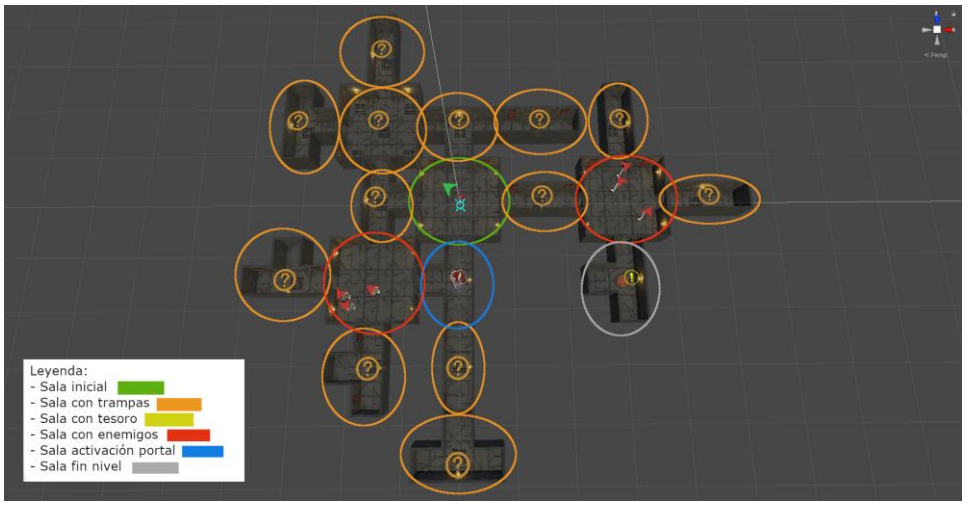

Ilustración 47 - Ejemplo de nivel a profundidad 3

El objetivo del jugador sigue siendo el mismo, conseguir el libro para activar el fin de nivel y dirigirse a la salida.
En este caso, si el jugador quiere conseguir más puntos, deberá dirigirse a las habitaciones con enemigos y derrotarlos.

Superando en este nivel, se consigue llegar la profundidad 4, y el '*layout*' generado en esta ocasión ha sido el siguiente:

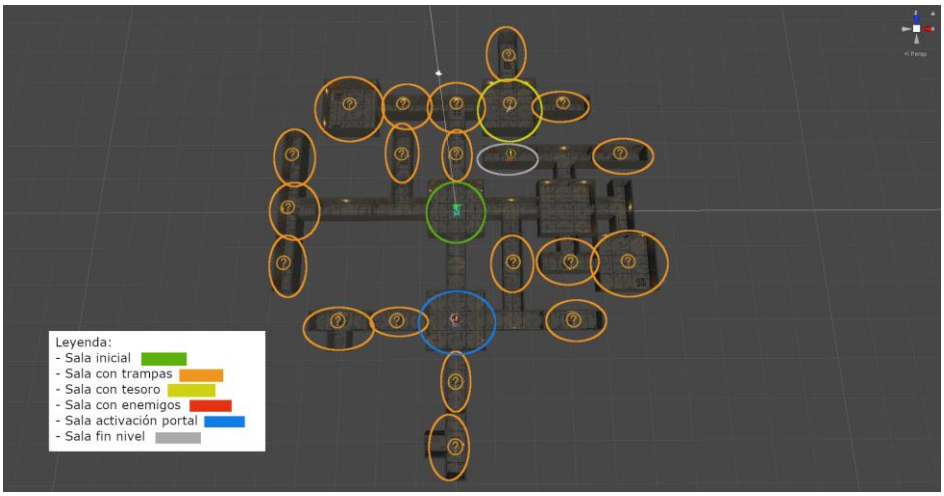

Ilustración 48 - Ejemplo de nivel a profundidad 4

Siguiendo con el mismo objetivo, el jugador debe conseguir el libro y salir del nivel por el portal. En esta ocasión, se dispone de un cofre del tesoro para agrandar la puntuación del jugador.

En esta ocasión, en cuanto el jugador cruza el portal, el nivel generado se corresponde a un nivel de profundidad máxima. Así pues, el objetivo cambia y para poder superar el nivel, debe encontrar la sala del jefe y derrotarlo para hacer aparecer el portal.

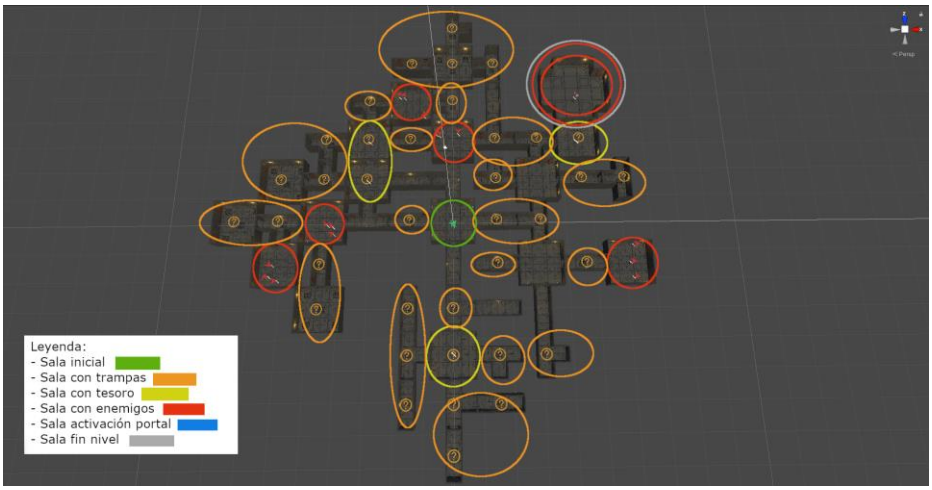

Ilustración 49 - Ejemplo de nivel a profundidad 5 (jefe final)

En esta ocasión, se puede ver que hay múltiples elementos de cada tipo, con lo que el jugador puede enfrentarse a diferentes desafíos y conseguir diferentes tesoros antes de enfrentarse contra el jefe del nivel.
En el momento en que el jugador derrote al jefe y consiga entrar al portal, el siguiente nivel generado se corresponderá a uno de profundidad mínima para iniciar otro ciclo de juego.
La puntuación conseguida hasta el momento y el contador de profundidad, seguirán incrementando.

Como puede observarse en las diferentes imágenes, cada vez que se genera un nivel, la distancia entre el centro y el extremo es mayor.
Si se observan con detenimiento las imágenes, se puede apreciar como una misma 'rama' nunca instancia dos veces consecutivas un componente del mismo tipo.
Aun así, se pueden observar excepciones, aunque estas vienen dadas por la coincidencia de haber colocado dos componentes del mismo tipo cerca, desde dos ramas diferentes.
En estos puntos es donde se generan ciclos dentro de los componentes.
En caso de no haberse desarrollado esta condición, para llegar a un extremo diferente el jugador debería dirigirse primero hacia el nodo inicial y dirigirse desde allí a la otra rama.
Como la asignación de finalidades en los componentes se asigna de forma completamente aleatoria y, además, los componentes tienen distintas cantidades de finalidades posibles produce que, en ocasiones, la variedad de retos que ofrece un nivel sea pobre.

6. Manual de usuario

6.1 Instrucciones de arranque

El fichero ejecutable del juego ha sido creado utilizando '*Winrar*'. Para hacerlo, se ha generado un fichero autoextraíble el cual se descomprime en una carpeta temporal en el momento de la ejecución para poder ejecutar el juego.
Para poder ejecutarlo, simplemente debe darse doble '*click*' encima del ejecutable.

6.2 Manual del juego

El objetivo del juego es avanzar todo lo posible en la mazmorra. Para hacerlo, se deben ir superando los diferentes retos y conseguir atravesar el portal de fin de nivel.
Primeramente, debe cumplirse con el objetivo de activación del portal. Este viene indicado en la esquina superior derecha, justo debajo del minimapa y puede ser uno de los dos siguientes:

- **Conseguir el libro que activa el portal**. Este se encontrará en algún componente de la mazmorra. Viene indicado en el minimapa por el icono de un libro.
- **Conseguir derrotar al jefe del nivel**. En ciertos niveles, aparece un enemigo más poderoso en una sala especial. Dicho enemigo viene indicado en el minimapa con una calavera. En caso de derrotarlo, aparecerá el portal de fin de nivel.

Los controles para realizar las diferentes acciones con el personaje son los siguientes:

Entrada (input del jugador)	**Acción realizada**
W/A/S/D	**Movimiento**: Adelante / Izquierda / Atrás / Derecha
Mayus (*shift*) izquierdo (MANTENER durante **movimiento**)	**Correr (*Sprint)***
Barra espaciadora (*Space*)	**Esquive ('*Dash*')**
Click Izquierdo	**Atacar**
Click Derecho (MANTENER)	**Bloquear**
Esc	**Pausa**

Este manual de acciones y objetivos puede ser visto desde el menú principal del juego.

7. Conclusiones

A lo largo de este trabajo, se ha utilizado de forma recurrente elementos asíncronos (‘*Corrutine*’) en diferentes métodos y llamadas de eventos (‘*Action*’ en su gran mayoría).
Si algo puede darse por trabajado y peleado se encuentra en estos temas. Principalmente debido al número de problemas encontrados con la generación de los niveles, cuyos scripts deben esperar cierto tiempo para recibir las validaciones de los componentes y poder generar un nivel sin errores.
Un aspecto que se ha podido observar, es la dificultad a la hora de ‘debugar’ estos comportamientos asíncronos y la importancia de utilizar de forma correcta los tipos de ‘*yield return*’ disponibles.
El otro aspecto observado es la cantidad de errores que se pueden generar si se van asignando eventos sin eliminar las suscripciones antiguas o de objetos destruidos, pudiendo ejecutar múltiples veces un fragmento de código o dando error por intentar acceder a un objeto que ya no existe.

Inicialmente se plantearon los siguientes objetivos:

- **Objetivos relacionados con la implementación de mecánicas del juego**: relacionados con la generación de niveles aleatorios, las mecánicas de movimiento y combate del personaje principal, la IA de los enemigos, las trampas y los objetos interactivos.
- **Objetivos relacionados con la integración de las mecánicas implementadas**: relacionados con la inclusión y coexistencia de todas las mecánicas y lógica desarrollada.
- **Objetivos relacionados con la ampliación de contenido**: relacionados con la ampliación de variedad en componentes, enemigos o trampas.

De estos objetivos, únicamente se ha dejado incompleto el objetivo relacionado con la ampliación de contenido. A medida que se ha ido progresando en el proyecto, se ha visto como aparecían necesidades y características que debían ser implementadas o corregidas para conseguir una experiencia de juego mejor.
En este aspecto, se puede ver que en un principio no estaba contemplado, por ejemplo, el minimapa o el indicador de objetivo. Aun así, durante el desarrollo del juego apareció la necesidad de un mecanismo que facilitara la orientación para el jugador y de indicación sencilla del objetivo del nivel.
Así pues, pese a no poder incluir la cantidad de variedad que se esperaba en un principio, se ha conseguido desarrollar un juego que cubre las necesidades básicas del jugador y sin errores en su ejecución.

La metodología de trabajo seguida, ha sido la metodología ágil Scrum. Aun siendo usada sin todas sus características (no se han realizado los '*daily meetings*' ni se han puntuado las tareas utilizando '*story points*', por ejemplo) el trabajar con una metodología ágil, ha facilitado la absorción de cambios en los requisitos y la corrección de los diferentes errores producidos.
Hay que decir, que la generalización de ciertas tareas, ha producido en ocasiones una estimación optimista. De este modo, durante los diferentes '*sprints*' que se han llevado a cabo, ha ocurrido en ocasiones que ciertas tareas quedaban sin acabar y debían ser desplazadas al siguiente '*sprint*'.

Las posibilidades de ampliación y mejora del proyecto a futuro que se contemplan habiendo finalizado este trabajo son:

- **Mejora del sistema de generación de la mazmorra**: se observa que, al depender completamente de la aleatoriedad a la hora de generar los niveles, la variedad de finalidades que aparecen puede llegar a ser simple. La posibilidad de mejora que se observa, es la inclusión de una estructura de software que almacene información de los componentes colocados y las finalidades asignadas para, de este modo, generar niveles con mayor diversidad.
- **Mejora de la IA de los enemigos**: se ha dejado de lado durante el desarrollo de este proyecto la utilización de árboles de comportamiento en lugar de la FSM utilizada. Mayormente debido a la familiaridad que se tenía con las FSM y que se quería centrar el trabajo en la generación de los niveles. Además de poder ampliar los comportamientos que tienen los agentes y pulir los aspectos que ya tienen disponibles.

8. Glosario

- **Backlog**: lista de tareas incompletas que suelen estar ordenadas de forma descendente en cuanto a prioridad.
- **Breadth First Search (BFS)**: algoritmo de búsqueda en un grafo, frecuentemente usado en árboles. Parte de la raíz y explora primero los nodos vecinos antes de seguir con el siguiente nivel.
- **Bullet hell**: tipo de juego, usualmente en 2D, en el que hay en pantalla un gran número de proyectiles enemigo. El jugador debe tener destreza para esquivar los proyectiles y derrotar a los enemigos.
- **Issues**: tarea a realizar dentro del '*backlog*'
- **Kanban**: sistema de información visual en el que se disponen las diferentes tareas a realizar. Dichas tareas se encuentran agrupadas en categorías dependiendo de su estado de desarrollo. Habitualmente las categorías son: para hacer (*To Do*), en desarrollo (*WIP, Work In Progress*) y completada (*Done*).
- **Milestones**: punto de referencia con el que contemplar el desarrollo de un elemento. En este caso, los '*milestones*' de GitHub se han utilizado como fechas de finalización de los '*sprints*'.
- **NPC**: siglas para '*Non-Player Character*', es decir, cualquier personaje que aparezca en un videojuego que no sea controlado por el jugador.
- **Scrum**: metodología de trabajo que consiste en el desarrollo continuo e incremental. Adaptando el trabajo a los cambios en las necesidades y requisitos.
- **Sprints**: unidad de tiempo fija definida en el proceso Scrum.
- **Visual scripting**: forma grafica de manipular elementos y comportamientos sin necesidad de escribir código.

9. Bibliografía

Lista de enlaces a las diferentes imágenes de terceros utilizadas en la memoria:

1. **The Binding of Isaac**: [fecha de consulta: 3 de mayo de 2021] Disponible en: https://i2.wp.com/gamerfocus.co/wp-content/uploads/2015/02/the-binding-of-isaac-rebirth-afterbirth-expansi%C3%B3n-detalles-edmund-mcmillen-nicalis-1.jpg?resize=740%2C415&ssl=1
2. **Moonlighter**: [fecha de consulta: 3 de mayo de 2021] Disponible en: https://i.blogs.es/157e4e/moonlighter-02/450_1000.jpg
3. **Soul-Knight**: [fecha de consulta: 3 de mayo de 2021] Disponible en: https://i1.wp.com/www.esferaiphone.com/uploads/Soul-Knight-1.jpeg?resize=1100%2C619
4. **Git**: [fecha de consulta: 6 de mayo de 2021] Disponible en: https://upload.wikimedia.org/wikipedia/commons/thumb/e/e0/Git-logo.svg/1280px-Git-logo.svg.png
5. **Subversion**: [fecha de consulta: 6 de mayo de 2021] Disponible en: https://upload.wikimedia.org/wikipedia/commons/2/22/Apache_Subversion_logo.svg
6. **Unity**: [fecha de consulta: 6 de mayo de 2021] Disponible en: https://upload.wikimedia.org/wikipedia/commons/thumb/1/19/Unity_Technologies_logo.svg/1280px-Unity_Technologies_logo.svg.png
7. **UnrealEngine**: [fecha de consulta: : 6 de mayo de 2021] Disponible en: https://cdn2.unrealengine.com/Unreal+Engine%2FblogAssets%2F2017%2FNOVEMBER+2017%2FDebugging+UFunction+Invoke%2FFB_UE_Generic-1200x630-099efdf05b3d515f62e20fe45124f26f8574d7fc.jpg
8. **Godot**: [fecha de consulta: 6 de mayo de 2021] Disponible en: https://godotengine.org/themes/godotengine/assets/press/logo_small_color_light.png

Lista de enlaces a las diferentes herramientas consultadas (motores de juego y control de versiones):

1. **Unity 3D**: [fecha de consulta: 6 de mayo de 2021] Disponible en: https://unity.com/es
2. **Unreal Engine**: [fecha de consulta: 6 de mayo de 2021] Disponible en: https://www.unrealengine.com/en-US/
3. **Godot Engine**: [fecha de consulta: 6 de mayo de 2021] Disponible en: https://godotengine.org/
4. **Git:** [fecha de consulta: 6 de mayo de 2021] Disponible en: https://git-scm.com/
5. **GitHub (cliente de Git):** [fecha de consulta: 6 de mayo de 2021] Disponible en: https://github.com/
6. **Subversion:** [fecha de consulta: 6 de mayo de 2021] Disponible en: https://tortoisesvn.net/

Lista de enlaces a las herramientas y '*packages*' externos utilizados para el desarrollo del proyecto:

1. **Mixamo**: Disponible en: https://www.mixamo.com/
2. **Audacity**: Disponible en: https://audacity.es/
3. **Unity – NavMeshComponents**: Disponible en: https://github.com/Unity-Technologies/NavMeshComponents
4. **OpenGameArt**: Disponible en: https://opengameart.org/
5. **Fuentes de texto**: Disponible en: https://www.1001freefonts.com/
6. **UMotion Pro**: Disponible en: https://assetstore.unity.com/packages/tools/animation/umotion-pro-animation-editor-95991
7. **Polygon Dungeon**: Disponible en: https://assetstore.unity.com/packages/3d/environments/dungeons/polygon-dungeons-low-poly-3d-art-by-synty-102677
8. **Classic RPG GUI – PONETI:** Disponible en: https://assetstore.unity.com/packages/2d/gui/classic-rpg-gui-160253

Lista de enlaces con los vídeos realizados para las diferentes entregas del trabajo de fin de máster y su defensa, de este modo se ofrece visión de la evolución del proyecto, el resultado final y explicaciones más detalladas de todas las características desarrolladas:

1. **Video de la versión parcial** ('*showcase*' PEC2): https://youtu.be/9f-mXkgg3rM
2. **Video de desarrollo de la versión parcial** (explicación detallada del funcionamiento): https://youtu.be/TiHrwrX1IdQ
3. **Video de la versión jugable** ('*showcase*' PEC3): https://youtu.be/U22S8_0DZrg
4. **Video de desarrollo de la versión jugable** (explicación detallada del funcionamiento): https://youtu.be/JryQnTj4RIE
5. **Video de la defensa del proyecto**: https://youtu.be/XsembrYLR-k