Universitat Oberta de Catalunya

uoc.edu

# Guardians of the Forest

**Alexis Daniel Fuentes Pérez**
Máster Universitario en Diseño y Programación de Videojuegos
Trabajo Final de Máster

**Helio Tejedor Navarro**
**Joan Arnedo Moreno**

Junio 2020

**FICHA DEL TRABAJO FINAL**

| | |
|---|---|
| **Título del trabajo:** | *Guardians of the Forest* |
| **Nombre del autor:** | *Alexis Daniel Fuentes Pérez* |
| **Nombre del consultor/a:** | *Helio Tejedor Navarro* |
| **Nombre del PRA:** | *Joan Arnedo Moreno* |
| **Fecha de entrega (mm/aaaa):** | 06/2020 |
| **Titulación:** | *Máster Universitario en Diseño y Programación de Videojuegos* |
| **Área del Trabajo Final:** | *Trabajo Final de Máster* |
| **Idioma del trabajo:** | *Español* |
| **Palabras clave** | *RPG, Guardians, Forest* |


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

Se ha realizado un videojuego RPG en 2D llamado Guardians of the Forest, dónde los jugadores deberán gestionar recursos para ajustar sus estadísticas y obtener el equipamiento y armamento deseado, para poder proteger, luchar, y sobrevivir en el mayor número de estancias. Existe una multitud de objetos, armas, y consumibles, que permiten tener muchos modos de juego diferente, como por ejemplo cuerpo a cuerpo, a distancia, mágico, tanque, supervivencia... Además, los jugadores se adentrarán en un bosque infinito formado por estancias autogeneradas y pseudoaleatorias de dificultad creciente, dónde encontrarán diferentes tipos de enemigos y jefes a los que deberá superar.

Para el desarrollo del proyecto se ha seguido una planificación y una metodología iterativa e incrementativa. Cómo resultado de este se ha obtenido el videojuego para PC en Windows, Mac, y Linux, una experiencia realizada con usuarios dónde se testeó y valoró el producto, y un backlog de tareas para mejorar y difundir el videojuego.

Como conclusiones, se ha obtenido un mayor conocimiento de los RPG, así como una mejora personal en el desarrollo de videojuegos, todos los hitos fueron alcanzados, y el juego ha obtenido numerosas críticas buenas de los jugadores que lo probaron.

**Abstract (in English, 250 words or less):**

I have made a 2D RPG video game called Guardians of the Forest, where players must manage resources to adjust their statistics and obtain the desired equipment and weapons, in order to protect, fight, and survive in the largest number of rooms. There are a multitude of objects, weapons, and

consumables, which allow you to have many different game modes, such as melee, ranged, magical, tank, survival... In addition, players will join an infinite forest formed by self-generated and pseudo-random rooms of increasing difficulty, where they will find different types of enemies and bosses that they must overcome.

Iterative and incremental methodology and a planning have been followed for the development of the project. As a result of this, I have obtained the video game for PC on Windows, Mac, and Linux, an experience carried out with users where the product was tested and evaluated, and a backlog of tasks to improve and disseminate the video game.

As conclusions, a greater knowledge of RPGs has been obtained, as well as a personal improvement in the development of video games, all objectives were reached, and the game has obtained numerous good reviews from the players who tried it.

# Agradecimientos

Agradecerle a mi familia, por haber estado siempre a mi lado apoyándome, por creer en mí, por ayúdame en todo lo que he necesitado, por animarme en la consecución de mis logros, y por todo el afecto que he recibido de ellos.

Agradecerle a todo el profesorado que me ha guiado y ayudado, tanto durante la realización de este Trabajo Final de Máster, como en las diferentes asignaturas que conformaban el Máster Universitario en Diseño y Programación de videojuegos.

Agradecerle a mi pareja y amigos, por haber estado junto a mi apoyándome.

# Índice

## Lista de figuras

# 1. Introducción

## 1.1 Contexto y justificación del Trabajo

Los juegos han estado presentes en la vida de las personas desde la antigüedad y han sido un ejercicio de entretenimiento y desarrollo intelectual, desde el uso de muñecos, balones, e imaginación, a juegos colectivos como el escondite. Los videojuegos por su parte son una adaptación informática de los juegos, que encajan mejor en una sociedad caracterizada por el avance tecnológico e innovador, dónde se presentan unas reglas a seguir por parte del usuario para conseguir o resolver el objetivo que este presenta.

Actualmente los videojuegos abarcan un público bastante mayor que en sus inicios, pues al principio se concebían más para jugadores casuales que buscaban algo de diversión y entretenimiento, y hoy en día este concepto ha sido ampliado a sectores como la educación, la competición deportiva, e incluso la sanidad como tratamientos psicológicos o de superación personal.

Se quiere realizar un videojuego RPG con gestión de recursos, que permita al usuario, por un lado, descubrir que *setup* óptima debe conseguir con su personaje, y por otro lado pensar en una estrategia a seguir con dicha *setup* para poder avanzar lo máximo posible dentro del mundo, sin olvidar ni dejar de lado el objetivo final de entretenimiento que se quiere proporcionar el jugador.

Para ello, se ha desarrollado Guardians of the Forest, un videojuego dónde el jugador deberá administrar un conjunto de recursos iniciales para configurar las estadísticas de su personaje y diseñar un inventario de armas y objetos acorde a su estilo de juego antes de adentrarse al bosque maldito. Una vez dentro, deberá superar el mayor número posible de estancias, luchando contras hordas enemigas y jefes que no se lo pondrán nada fácil, pues tras vencer a cada jefe, las estancias subirán de nivel y con ello la dificultad de juego.

## 1.2 Objetivos del Trabajo

Este trabajo presenta cinco objetivos principales y una serie de objetivos secundarios que permiten lograr una experiencia completa y lograda de cara a los usuarios finales, los jugadores.

El primer objetivo principal es la administración de recursos dentro del juego, el polen dorado y hojas del valle. El guardián comenzará con una

cantidad fija de ambos, y deberá canjearlos en el Valle por puntos en sus estadísticas (salud, resistencia, agilidad, fuerza, e intelecto) y en comprar y obtener diferentes objetos, consumibles, y armas. Se debe ofrecer una gran variedad equilibrada de opciones para que así el jugador deba analizar lo que más le conviene y crearse una *setup* de condiciones que funcione y tenga sentido. Los mejores guardianes serán aquellos que sepan encontrar combinaciones óptimas para el estilo de juego que quieran seguir (mágicos, tanques, luchadores, …) y que les permita sobrevivir y acabar con el mayor número de demonios y jefes posibles.

El segundo objetivo principal consiste en ofrecer mecánicas de uso diferentes en cada una de las armas y objetos, para que la forma de jugarlos y combinarlos sea numerosa. En relación a las armas, tendríamos armas a cuerpo a cuerpo y a distancia, pero luego dentro de las armas cuerpo a cuerpo podemos encontrar espadas o dagas, y en las armas a distancia desde bastones mágicos, a varitas o arcos, por lo que cada una tiene unas mecánicas exclusivas de uso, y, además, los guardianes pueden sostener dos armas al mismo tiempo, una en cada mano. A esto además le sumamos las pasivas de cada arma, desde ralentización al enemigo, probabilidad de envenenar o quemar, robo de vida, y si sumamos otras características como puede ser el cooldown o enfriamiento de cada una, pues tenemos un gran abanico de opciones a analizar… En cuanto a los objetos, podemos obtener desde objetos que reducen enfriamientos, otorgan resistencias, aumentan la distancia de nuestros proyectiles, a consumibles como pociones de invisibilidad, agilidad, escudos... Dominar el uso de cada uno de ellos, y la forma de combinarlos será parte de la experiencia del jugador y la destreza que tenga.

El tercer objetivo principal consiste en disponer de estancias autogeneradas y pseudoaleatorias que conformen el bosque maldito, que den al jugador una sensación de vagar por bosques diferentes cada vez, y que cada partida sea única y diferente a cualquier anterior.

El cuarto objetivo principal se basa en ofrecer diferentes tipos de enemigos y jefes, cada uno de ellos con un comportamiento e inteligencia artificial propia y característica. De esta forma, cada enemigo dispondrá de un set de ataques y habilidades, desde ataques cuerpo a cuerpo, hechizos dañinos, habilidades de curación, áreas de daño... así como niveles de enfriamientos, vida, velocidad, o resistencias diferentes.

El quinto objetivo principal reside en la dificultad creciente del juego. Cada nivel estará formado por varias estancias, y tras superar al enemigo jefe, se podrá continuar al siguiente nivel y aumentará la dificultad actual. La dificultad residirá en la vida, daño, agilidad, poder y alcance de habilidades de los enemigos a los que nos enfrentaremos. Los primeros niveles serán más sencillos para adaptar al usuario al juego, y conforme iremos avanzando irán aumentando porcentualmente.

A continuación, se listarán los objetivos secundarios.

1. Videojuego 2D con apariencia de profundidad.
2. Sistema de movimiento libre en cuatro direcciones.
3. Sistema de físicas y colisiones.
4. Sistema de estadísticas general aplicable tanto a jugadores como enemigos.
5. Inventario completo del guardián e inventario rápido (mano izquierda y derecha).
6. Sistema de pausa.
7. Pantallas de inicio, fin de partida, opciones y controles.
8. Diseño del Valle de la Luz
9. Diseño y programación de comerciantes, artesanos, y resto de habitantes del Valle.
10. Sistema de conversación e interacción con NPCs.
11. Sistema de compra y devolución de items.
12. Programación de armas, objetos, y consumibles.
13. Sistema de loot.
14. Diseño del bosque.
15. Autogeneración de estancias pseudoaleatorias del bosque.
16. Diseño y programación de la IA de los enemigos y jefes.
17. Sistema de puntuación basada en niveles.
18. Dificultad creciente del juego basada en niveles.
19. Animaciones acordes y sincronizadas con las acciones.
20. Sonidos realistas.
21. Interfaz usable y útil.
22. Integración de joysticks.

## 1.3 Enfoque y método seguido

En este trabajo se ha decidido desarrollar un producto nuevo desde cero, que viene a ser el videojuego en sí. Para ello, se ha escogido la estrategia de desarrollo iterativo y creciente, porque permite ir desarrollando diferentes hitos del videojuego comenzando por los requerimientos más simples o necesarios, y luego ir incrementando el con más funcionalidades hasta lograr un producto completo.

De esta manera se puede comenzar por ejemplo con el hito del sistema de movimiento en cuatro direcciones, se analiza, se implementa, se prueba, y luego continuamos con otro requerimiento e iteramos de nuevo. Esta estrategia permite también ir adaptando el producto y añadiéndole nuevas funcionalidades de forma incrementativa, obteniendo un output válido al final de cada iteración que podríamos enseñar o sacar a release si consideramos que es momento oportuno.

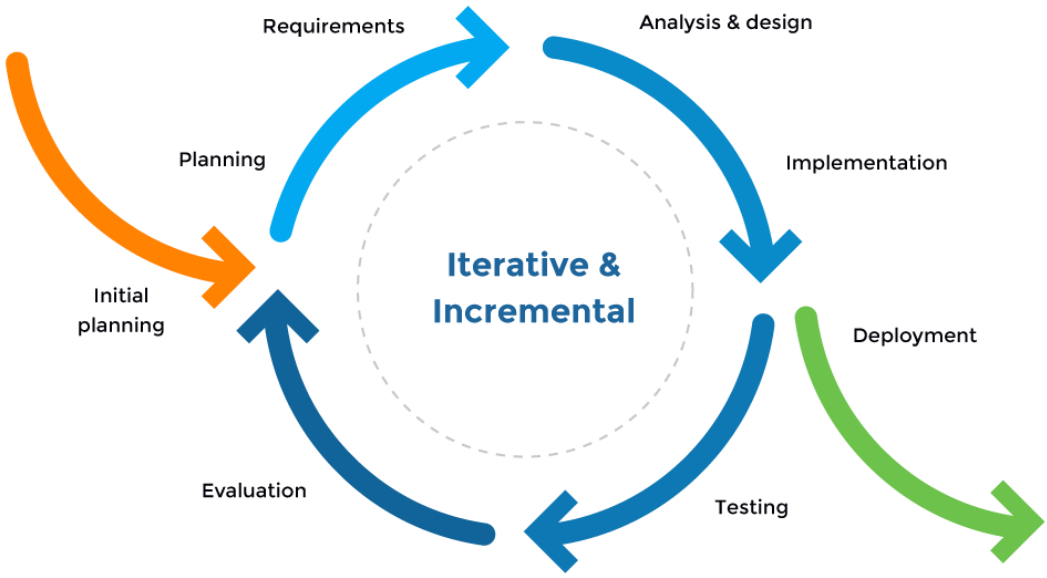


**Ilustración 1. Desarrollo Iterativo [1]**

## 1.4 Planificación del Trabajo

En relación con la planificación del trabajo, como recurso para la realización del videojuego cuento conmigo mismo como desarrollador único. Para cuantificar el tiempo disponible y lo que se le va a dedicar a cada objetivo para llegar en plazos con la entrega del producto final, se ha realizado un diagrama de Gantt con todos los objetivos.

Se han organizado con respecto a los plazos de las respectivas entregas de PEC2 y PEC3. Además, existen dos tareas (animaciones y sonidos) que se empezarán a realizar en paralelo una vez empiecen a introducirse elementos relacionados. Por último, se ha querido dejar un margen de tiempo libre al final de cada plazo de entrega para cualquier imprevisto que pueda surgir.

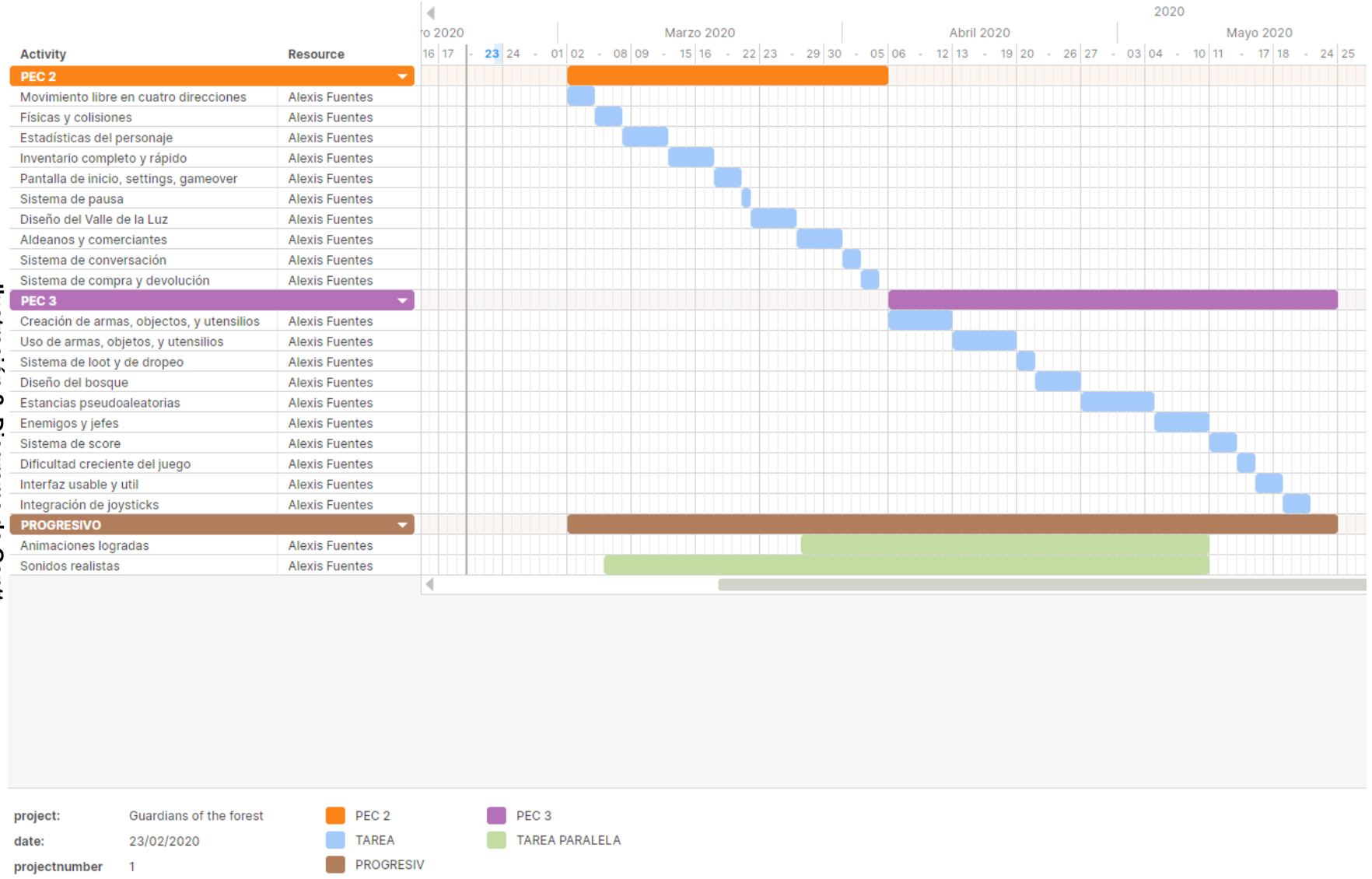


**Ilustración 2. Diagrama de Gantt**

## 1.5 Breve sumario de productos obtenidos

Como resultado de este trabajo se han obtenido los siguientes productos:

1. Ejecutables del videojuego 'Guardians of the Forest' para las plataformas de PC Windows, Linux y Mac.
2. Múltiples videos explicativos del videojuego que han sido utilizados para las entregas de las diversas PECs.
3. Video tráiler promocional del videojuego.
4. Documento de diseño del videojuego.
5. Diagrama de Gantt con la planificación seguida.
6. Repositorio en GitHub con todo el contenido del proyecto y tags con cada una de las releases / entregas realizadas.
7. Resultados de una encuesta sobre opiniones y valoraciones del videojuego en diferentes ámbitos de jugadores entre 18 y 29 años.
8. Backlog de tareas futuras para introducir nuevas funcionalidades y mejorar el producto actual.

## 1.6 Breve descripción de los otros capítulos de la memoria

A continuación, se hará una explicación de los contenidos de cada capítulo que conforman este trabajo final.

- Estado del arte. Realizaré una revisión sobre el género de los RPG, y las tecnologías que han usado ciertos títulos conocidos y que pueden guardar similitud con el mío.
- Definición del juego. Profundizaré en la conceptualización del juego, sus personajes, historia, y ambientación.
- Diseño técnico. Explicaré el entorno de desarrollo elegido, los assets y componentes desarrollados, las lógicas y algoritmos más importantes que he implementado, así como las IA de los enemigos.
- Diseño de niveles. Mostraré el nivel de Valle de la Luz, así como la base de los niveles dentro del Bosque Maldito, y las partes más importantes de cada uno.
- Manual de usuario. Explicaré todo lo necesario para poder jugar, así como una explicación del apartado de controles que hay dentro del juego.
- Conclusiones. En este apartado dejaré las conclusiones que he obtenido durante la realización de este Trabajo Final de Máster, así como los conocimientos más importantes que he adquirido.

# 2. Estado del arte

## 2.1 Revisión sobre el género

El género RPG se caracteriza por la adoptación de un rol dentro del mundo jugable, dónde nuestros personajes presentan parámetros como la vida, la defensa, o el daño, los cuáles pueden maximizarse o ajustarse, y el resultado de nuestras acciones dependerán de la cuantía de cada uno de ellos.

En sus inicios, los videojuegos RPG surgieron debido al gran auge que estaban teniendo los juegos de rol de mesa como *Dungeons & Dragons*, lo que llevó a su digilitazación. De esta manera, comenzaron a aparecer los primeros videojuegos RPG como *The Game of Dungeons / DND*, que se realizaron para el sistema PLATO (Programmed Logic Automated Teaching Operations), el primer sistema informático de gestión de aprendizaje.

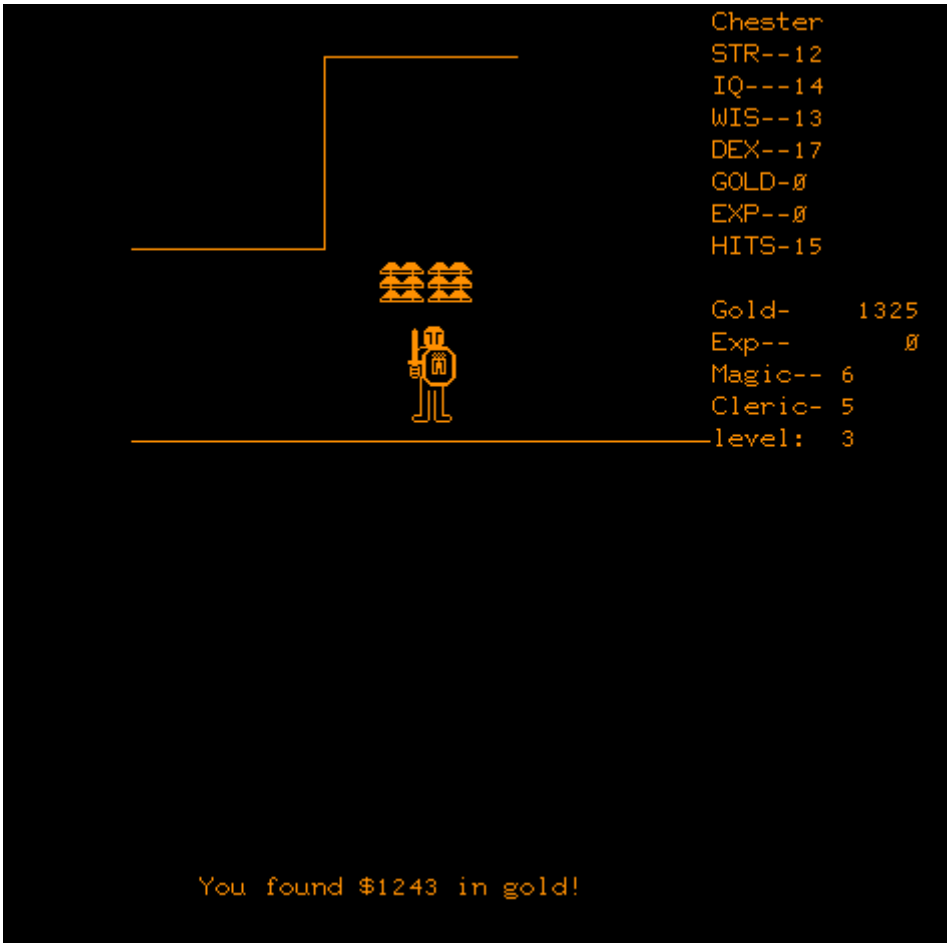


**Ilustración 3. DND.**

En la actualidad, los videojuegos RPG han evolucionado y podemos encontrar diferentes tipos de subgéneros y grandes títulos que los representan y lideran los mercados:

- **Rol clásico**. El jugador se representa con un personaje que debe progresar y avanzar a la vez que evoluciona en un mundo sujeto a reglas. Un ejemplo claro de este tipo de RPG es la saga principal de los videojuegos *Pokémon*, donde el protagonista debe formar un equipo con diferentes criaturas (las cuáles presentan diferentes estadísticas, tipos, y movimientos) y entrenarlas para mejorarlas, poder derrotar a los rivales, y obtener el título de campeón.

- **Mazmorras multiusuario**. Este tipo de juegos se caracterizan por transcurrir en un servidor, y a diferencia de los de rol clásico, estos persisten en el tiempo, es decir, la partida no comienza y termina cuando el jugador lo desea, y detrás de ellos existe toda una comunidad. Además, la interacción en este tipo de juegos suele ser en modo texto, usando lenguaje natural. En este subgénero podemos encontrar juegos como *Simauria* o *Reinos de Leyenda*.

- **Rol multijugador masivo**. Es la evolución de las mazmorras multiusuario. En estos videojuegos, las personas se conectan con sus personajes a servidores para jugar, interaccionar entre ellos, y cooperar o competir en línea. Estas partidas persisten en el tiempo. El mejor ejemplo de este género es World of Warcraft, título líder en los MMORPG, dónde los jugadores disponen de un montón de razas y clases disponibles, así como ramas de especialización, y un montón de atributos y características que mejorar con equipo y armamento que consiguen en mazmorras y estancias.

**Ilustración 4. Pokémon Sword.**

## 2.2 Revisión sobre la tecnología

En relación con las tecnologías usadas hoy en día para el desarrollo de títulos RPG, encontramos que las grandes compañías detrás de juegos como *Pokémon* o *World of Warcraft* hacen uso de sus propios motores privados para el desarrollo de estos. Por otro lado, si buscamos compañías de menor tamaño, o estudios indies, si podemos encontrar más información acerca de las tecnologías y motores que usan. A continuación, se realizará una comparativa entre diferentes videojuegos RPG y sus entornos de desarrollo:

| **Videojuego** | **Compañía** | **Tecnología** | **Lenguaje** |
|---|---|---|---|
| Moonlighter | Digital Sun | Unity | C# |
| Stardew Valley | Eric Barone | Microsoft XNA | C# |
| Borderlands 3 | 2K Games | Unreal Engine | C++ |
| Skyrim | Bethesda Game Studios | Creation Engine | Papyrus |
| Slay the Spire | MegaCrit | LibGDX | Java |
| Albion Online | Sandbox Interactive | Unity | C# |
| Last Oasis | Donkey Crew | Unreal Engine | C++ |

Cómo se puede observar hay muchas opciones válidas a la hora de escoger una tecnología para desarrollar un videojuego RPG, ya que este género no se focaliza ni necesita de una tecnología o motor específico para obtener unos grandes resultados. Si bien si encontramos que suelen ser más característico el uso de los lenguajes C# o C++, pero esta predominancia es más típica del sector de desarrollo de videojuegos en general, que de serlo específicamente de los RPG.

# 3. Definición del juego

## 3.1 Historia, ambientación y/o tema

Desde hace siglos, el Valle de la Luz ha sido un lugar de paz y armonía situado en el centro del Bosque de Eldryn, el corazón de nuestro mundo. Sus habitantes se han dedicado a proteger y cuidar los bosques, así como la fauna y seres mitológicos que lo habitan, y esta misión ha ido heredándose durante generaciones.

Sin embargo, hace diez años este equilibrio se rompió. Una noche un estrepitoso ruido, como si de una explosión se tratase, inundó todo el valle, y se pudo observar como un motón de rayos de luz violeta salían del interior del bosque hasta el cielo. Fue entonces cuando se formó un equipo de exploradores para acudir de emergencia a los bosques y ver de qué se trataba. Una vez dentro, el bosque estaba muy cambiado… La vegetación había tomado unos tonos violetas y rosáceos, como si hubieran enfermado o de una infección se tratase, y no había ni rastro de los animales y criaturas que solían habitar esos lugares. De pronto, una sombra se movió entre las hierbas y una criatura fantasmagórica con túnica negra se abalanzó sobre ellos. Se trataba de un demonio, hacía muchísimo tiempo que ninguno había aparecido por Eldryn. Tras acabar con el enemigo, observaron como cerca de ellos un rayo de energía violeta proveniente del bosque chocó con unas rocas, y cómo emergían varios golems arcano. Eran demasiado poderosos, y lucharon durante varios minutos hasta derrotarlos. Desgraciadamente, esta batalla se cobró la vida de la mitad del escuadrón. Asustados, los supervivientes comenzaron a correr en dirección al valle para avisar a todos los habitantes del peligro que había aparecido en los bosques, cuando otro aro de luz brillante los atravesó a todos y se empezó a materializar tomando elementos del bosque, formando un ser que recordaba a los árboles del lugar, pero lleno de una energía diabólica. En este momento se dieron cuenta de lo que estaba pasando… La explosión que habían oído había sido una fractura en el espacio uniendo el antiguo mundo de los demonios con el de Eldryn, una fractura que había convertido el bosque en un umbral entre ambos mundos, y estas luces violetas que veían no eran más que las almas demoniacas que atravesaban el portal, listo para atacar y conquistar las tierras. La batalla era devastadora, y ya solo quedaban en pie dos exploradores entre los que se encontraba una habilidosa maga. Ante la situación, esta decidió sacrificarse y lanzar un conjuro de protección contra el Valle de la Luz, para que las de las flores del valle nacieran guardianes que pudieran contener a los demonios en los bosques, y teletransportó a su compañero de vuelta al valle para que alertara a los habitantes.

Desde ese día, los habitantes del pueblo cultivan y cuidan las flores del valle, ahora denominadas Flores de la Luz, las cuales al madurar se transforman en guardianes cuya misión es adentrarse al bosque y mantener el mal dentro, de manera que ningún inocente más tenga que

perder su vida. Los habitantes del pueblo proporcionan a estos guardianes equipamiento, armas, y savias de conocimiento a cambio del polen y restos de su Flor de la Luz para poder seguir cultivando nuevas. Además, se cuenta que estos guardianes pueden oír la voz de la maga que los creó, una voz que los guía y los ayuda, a la que ellos llaman la Primera Guardiana.

## 3.2 Definición de los personajes

Los personajes de este juego están formados por los guardianes, los aldeanos del Valle de la Luz, y los demonios.

- **Guardián**. Es el personaje controlado por el jugador. Se materializan a partir de las flores del Valle de la Luz. Su objetivo es configurar sus estadísticas y hacerse con el armamento y equipamiento necesario para controlar a los demonios dentro de los bosques. Son capaces de equiparse y usar un arma con cada mano. Al morir, el hechizo se mitiga y vuelven a transformarse en flores, pero sin posibilidad de volver a crecer.

- **Primera Guardiana**. Es la maga que con su sacrificio lanzó con conjuro de protección sobre el valle y sus flores. Se les aparece a los guardianes en forma de susurros para guiarles en su camino.

- **Comerciantes**. Venden savias de conocimiento, capaces de subir las estadísticas de los guardianes al ser consumidas, a cambio de polen dorado de la Flor de la Luz.
  - Goren, el soldado. Vende savia que aumenta la resistencia.
  - Meredith, la enfermera. Vende savia que aumenta la salud.
  - Kevin, el mago. Vende savia que aumenta el intelecto.
  - Caitlyn, la luchadora. Vende savia que aumenta la fuerza.
  - Abra, el ex corredor. Vende savia que aumenta la agilidad.

- **Artesanos**. Venden armamento, equipamiento, y consumibles a cambio de hojas de la Flor de la Luz.
  - Damián, el guerrero. Especialista en armamento cuerpo a cuerpo y daño físico.
  - Elena, la costurera. Especialista en equipamiento para batallas de todo tipo.
  - Waldo, el alquimista. Especialista en consumibles con efectos únicos.
  - Circe, la bruja. Especialista en armamento mágico y poderes oscuros.
  - Orión, el cazador. Especialista en armas a distancia y en puntería certera.
  - Rose, la cuidadora. Especialista en objetos y artilugios que confunden a enemigos.

- **Segadores.** Demonios cuerpo a cuerpo. Se caracterizan por llevar túnicas negras y guadañas con las que cortan a sus enemigos.

- **Golems.** Demonios mágicos. Se caracterizan por lanzar hechizos rúnicos con los que desintegra a sus enemigos.

- **Ancianos.** Demonios mayores. Introducen su alma dentro de los árboles más toscos y resistentes. A la hora de atacar embisten con todo su cuerpo y hace uso de poderes ancestrales de curación.

- **Jinetes**. Demonios mayores. Posee a los pinos con las copas más altas. Expertos en batallas, atacan con sus feroces brazos a los incautos que se le acercan, y además disparan hojas encantadas que criban al enemigo y con las que les drena la vida.

### 3.3 Interacción entre los actores del juego

A continuación, se dispondrán las diferentes relaciones que pueden darse entre los personajes del videojuego.

- **Guardián - Primera Guardiana**. La Primera Guardiana se comunicará a través de susurros con el guardián para guiarle en su aventura.

- **Guardián - Comerciantes**. El guardián podrá hablar con los comerciantes para comprar y consumir diferentes tipos de savias.

- **Guardián - Artesanos**. El guardián podrá hablar con los artesanos para poder comprar y/o vender equipamiento, armas, y consumibles para su aventura.

- **Guardián - Demonios**. El guardián tendrá que luchar a muerte contra todos los tipos de demonios, pues estos una vez le vean intentarán acabar con él.

- **Primera Guardiana - Comerciantes y artesanos**. No pueden interaccionar, únicamente saben de qué de alguna manera sigue ligada a los guardianes por los que oyen de boca de estos.

- **Primera Guardiana - Demonios**. En el pasado luchó contra ellos y falleció. Ahora se encuentra en otro plano y no pueden tocarla.

- **Comerciantes y artesanos - Demonios**. Los habitantes no serían rival para los demonios, y se encuentran protegidos dentro del valle gracias al trabajo de los guardianes.

## 3.4 Concepts art

El arte en el que se ha inspirado este trabajo coincide con el de títulos cómo *Moonlighter* o *The Binding of Isaac*, en los que la temática roguelike está bastante presente.


**Ilustración 5. Moonlighter: Poblado**


**Ilustración 6. Moonlighter: Mazmorras**

**Ilustración 7. The Binding of Isaac: Mazmorras.**

## 3.5 Arte y diseño final

En este apartado se mostrará el arte utilizado para diferentes elementos clave de la historia, como los personajes principales y componentes más destacables.

**Ilustración 8. Flor de la Luz.**

**Ilustración 9. Guardián.**

**Ilustración 10. Segador.**

**Ilustración 11. Segador atacando.**

**Ilustración 12. Golem.**

**Ilustración 13. Golem atacando.**

**Ilustración 14. Anciano.**

**Ilustración 15. Anciano atacando.**

**Ilustración 16. Anciano curándose.**

**Ilustración 17. Jinete.**

**Ilustración 18. Jinete atacando.**

**Ilustración 19. Jinete disparando.**

**Ilustración 20. Goren.**

**Ilustración 21. Meredith.**

**Ilustración 22. Kevin.**

**Ilustración 23. Caitlyn.**

**Ilustración 24. Abra.**

**Ilustración 25. Damián.**

**Ilustración 26. Elena.**

**Ilustración 27. Waldo.**

**Ilustración 28. Circe.**

**Ilustración 29. Orión.**

**Ilustración 30. Rose.**

# 4. Diseño técnico

## 4.1 Entorno elegido

Estuve analizando diferentes motores de videojuegos, entre los que destacaban Unity, Unreal Engine, y Godot. Estuve pensando en utilizar Unreal Engine por el hecho de poder utilizar C++ como lenguaje de programación, que desde mi punto de vista es más potente y óptimo que otros lenguajes. Por otro lado, Godot me llamaba bastante la atención por ser más reciente que los otros dos y el gran auge y buen acogimiento que está recibiendo por parte de la comunidad de desarrolladores. Finalmente, decidí escoger Unity por ser el motor con el que hemos trabajado durante todo el máster, ya que gracias a ese hecho tengo grandes conocimientos sobre él y, además, también lo he utilizado personalmente en otros proyectos como en diferentes Game Jams en las que he participado.

En específico se va a ser uso de la versión LTS de Unity 2018.4.18f1.

El motor de Unity presenta ciertas características que, en mi opinión, lo hacen una muy buena opción para desarrollar este videojuego:

- **GameObject y modularización con componentes.** En Unity todo lo que introduzcas en la escena es un GameObject el cual va ganando funcionalidades según los componentes que les vayas añadiendo. El hecho de poder desarrollar tú dichos componentes, o modificarlos mediante extensión o herencia, es un factor bastante potente pues puedes crear componentes genéricos para diferentes tipos de objetos que guarden cierta similitud base, o modificar componentes más complejos de Unity para adaptarlo a tu caso específico.

- **Motor de físicas**. El uso de rigidbodies para incorporar físicas a nuestros objetos es bastante sencillo, y una vez dominas el uso de los mismo se convierte en una herramienta bastante potente, desde el control de eventos de colisiones, el control de tics de *sleep*, o la simulación.

- **Cámaras virtuales**. El componente de cinemachine es bastante útil para lograr cámaras virtuales que sigan al jugador según la parametrización configurada.

- **Networking**. Unity dispone de las librerías UNet y HLAPI para gestionar funcionalidades multijugador, desde el lado del servidor o host como de desde el lado de los clientes, y como trabajo futuro de este TFM se quiere incorporar un modo cooperativo en línea.

## 4.2 Requisitos técnicos del entorno de desarrollo.

A continuación, se detallan los requisitos técnicos del editor de Unity:

- **Windows**
  - **SO**: Windows 7 (SP1+) y Windows 10, 64-bit
  - **CPU**: Arquitectura X64 con set de instrucciones SSE2.
  - **GPU**: DX10, DX11, y DX12

- **Mac**
  - **OS:** Sierra 10.12.6+
  - **CPU:** Arquitectura X64 con set de instrucciones SSE2.
  - **GPU**: Intel y AMD con Metal.

- **Linux**
  - **OS**: Ubuntu 16.04, Ubuntu 18.04, y CentOS 7
  - **CPU**: Arquitectura X64 con set de instrucciones SSE2.
  - **GPU**: Nvidia y AMD con Vulkan o OpelGL 3.2+.

## 4.3 Herramientas empleadas

Se lista el total de herramientas empleadas para el desarrollo completo del videojuego.

- **Unity 2018.4.18f1**, como *game engine* del videojuego.
- **Visual Studio 2019**, como editor de programación.
- **Pixlr**, como editor online de imágenes para retocar sprites y otros elementos de la UI.
- **GitHub**, como repositorio git, gestor de versiones y releases.
- **Itch.io**, como repositorio de assets en línea.
- **Unity Asset Store**, como repositorio de assets en línea.
- **Adobe Illustrator CC 2019**, para la creación del logo personal.
- **Adobe Premier Pro CC 2019**, para la edición de videos.
- **Google Form**, para las pruebas con usuarios.

## 4.4 Descripción de assets y recursos del juego

En este apartado explicaré todos los assets creados y utilizados. Quiero destacar que mi trabajo se ha centrado en la programación y en la creación del videojuego en sí, y no en el diseño sprites, pues he cursado el itinerario de programación avanzada en durante el máster.

### *Animations*

Aquí se encuentran todas las animaciones y animators creados para los guardianes, los segadores, los golems, los jinetes, los ancianos, y animaciones para la interfaz de usuario.

### *Audio*

Creé un AudioMixer para poder controlar el volumen del juego desde dentro de la partida, con el canal Master.

### *Fonts*

Aquí se localizan las fuentes utilizadas en el juego. En concreto, la fuente Londrina y sus variantes.

### *Items*

En esta carpeta residen todos los objetos que he creado para el juego. Para ello creé un scriptable 'Item' (el cuál explicaré con más detalle en la sección de items) con todos los atributos que definen a un objeto dentro de la lógica de Guardians of the Forest, y todos estos elementos son instancias de dicho scriptable. De esta manera, si se desea mejorar o disminuir las estadísticas de un ítem para equilibrarlo, basta con ir a la instancia del objeto y cambiar los valores deseados desde el inspector

- **Consumibles**. Contiene todos los consumibles, a saber, las pociones de vida, la de invisibilidad, la de escudo, y la de velocidad.

- **Major Weapons**. Contiene las armas "mayores" o más poderosas. Estas armas suelen causar mayores estragos a los enemigos o proporcionan grandes utilidades, pero a cambio tienen un mayor enfriamiento de uso. Aquí podemos encontrar los diferentes arcos (daño, fuego, veneno, robo de vida, y ralentización), los bastones mágicos (veneno y curación), y las espadas (daño y ralentización).

- **Minor Weapons**. Contiene las armas "menores" o secundarias. Estas armas tienen un daño moderado o bajo, pero a cambio tienen poco enfriamiento de uso. Ideales para usar como arma secundaria de un arma mayor, y jugar con los tiempos de enfriamiento. Aquí podemos encontrar las dagas (veneno, fuego, y robo de vida), las varitas (daño, fuego, y ralentización).

- **Passives**. Contiene los objetos de equipamiento. Otorgan pasivas que aumentan diferentes estadísticas. Encontramos aquí el cinturón de daño, las botas de velocidad, el yelmo de resistencia, la pechera de resistencia, los guantes de enfriamiento, el gorro de distancia, y el collar enfriamiento.

- **Stats**. Contiene las savias de conocimiento. Aumentan una estadística al consumirla. Tenemos las savias de agilidad (velocidad), vida (salud), intelecto (daño mágico), resistencia (defensa), y fuerza (daño físico).

- **Utilities**. Contiene los artilugios de utilidad. Este tipo de objetos es parte del trabajo futuro de este TFM.

### *Navigation2D*

Se quería hacer uso del componente Navigation de Unity para convertir los enemigos en agentes y que pudieran desplazarse por el mapa utilizando el algoritmo A*, esquivando los obstáculos físicos. Sin embargo, este componente solo funciona en mundos 3D, por lo que investigué y encontré este asset de pago que adapta con un script al componente y permite su uso.

Se ha utilizado en los prefabs de los enemigos (segadores, golems, jinetes, y ancianos) de forma complementaria a la IA que les he desarrollado yo, la cual explicaré en el apartado correspondiente.

### *Palettes*

Aquí se encuentran todos los tiles creados para los diferentes tilemaps que forman parte de las escenas del juego.

### *Particles*

En esta carpeta se encuentra un asset de partículas, sprites, y materiales que he utilizado como base para crear los diferentes sistemas de partículas de mi juego.

### *Prefabs*

Contiene todos los prefabs que he almacenado para mantener una sincronía entre elementos de diferentes escenas, como los guardianes, elementos de la interfaz de usuario, objetos clave para el gameplay, así como elementos que se repiten en abundancia como la decoración (árboles, rocas, césped…) que tienen scripts y características especiales para crear el efecto de profundidad 2D, y elementos *spawnmeables* como los sistemas de partículas creados para los hechizos de los bastones y varitas, los hechizos de los enemigos, los proyectiles de los arcos, y los elementos que forman los bosques autogenerados.

### *Resources*

Aquí se localizan los recursos que se cargan desde código durante el juego.

- **Dialogues**. Contiene los diálogos del juego en formato JSON, de manera que si se quiere internacionalizar y traducir a otros idiomas sería bastante sencillo.

- **Prefabs > States**. Contiene los prefabs de las partículas de estado (quemado, envenenado, ralentizado) que son cargados por código al sufrir el estado determinado.

### *Scenes*

Contiene las escenas que componen el videojuego (AD, Start, Valley, Forest y End) y el Navigation generado para la escena Forest.

### *Scripts*

Contiene todos los scripts que desarrollé para el videojuego.

- **EnemyState**. Carpeta con todos los estados para la máquina de estados de la IA de los enemigos.

- **AudioController**. Script para controlar y mantener entre escenas el AudioMixer.

- **AutoSortOrder**. Script para ordenar de forma automática el layer de un elemento trigger del escenario según la posición del jugador cuando pasa por alrededor suyo. Proporciona el efecto de profundidad deseado, por ejemplo, atravesando matojos o pisando hierbas. Se utiliza para la escena estática del valle.

- **AutoSortOrder2**. Script para ordenar de forma automática el layer del jugador o del enemigo al moverse cerca de elementos que han sido colocado con el algoritmo de generación pseudoaleatoria de la sala, por lo que usan un order específico según un algoritmo basado en la posición Y del elemento. Hace uso de un GameObject hijo que se posiciona en la planta de los pies del jugador. Proporciona el efecto de profundidad deseado, por ejemplo, atravesando matojos o pisando hierbas. Se utiliza en la escena autogenerada del bosque.

- **Bar**. Script que llevan las barras de vida/escudo para ajustar el valor del relleno mediante una función pública.

- **BarController**. Script que controla las barras de vida y escudo del jugador actualizándolas con la cantidad de vida y escudo actual.

- **CommonUI**. Script con funciones comunes para elementos de la UI, como reproducir el sonido *hover* al pasar por encima de un elemento.

- **CrossForest**. Script para atravesar a una nueva estancia del bosque maldito.

- **Data**. Clase estática que sirve para guardar datos, como las estadísticas, el inventario, los cooldowns, … entre escenas.

- **DestroyParticles**. Script para destruir las partículas que lo contienen una vez estas terminen su ejecución.

- **DialogueEvent**. Script que mediante el uso del DialogueController abre diálogos de evento (que no permiten al usuario moverse hasta completarlos) al entrar en un área específica.

- **DialoguesController**. Script donde se centralizan los diálogos del juego. Carga los diálogos desde ficheros JSON situados en Resources. Se ha programado para que en un fichero de dialogo venga tanto el nombre del individuo, como todas sus ventanas de dialogo.

```
{
	"speaker": "Goren, el Soldado",
	"texts": [
		"Hola chico!\nEste es mi primer diálogo\n¿Cómo estas?",
		"Este es mi segundo dialogo!!",
		"Y aquí va mi último dialogo!\nQue tengas un buen día."
	]
}
```

- **EndController**. Script de la pantalla de fin del juego, dónde se cargan el nivel máximo obtenido y se le muestra al usuario.

- **EnemyAI**. Inteligencia artificial de los enemigos. Hace uso de una máquina de estados. Será explicada en detalle en el apartado de IA de los enemigos.

- **EnemyBarController**. Script que controla las barras de vida de los enemigos actualizándolas con la cantidad de vida y escudo actual. Estas barras son instanciadas en un canvas especial sólo para las barras de los enemigos (que se encuentra por detrás del canvas del jugador, asi estas vidas no se pintaran por encima de elementos importantes para el player), y este script también actualiza la posición de las mismas según la posición del enemigo y un offset.

- **EnemySpell**. Script contenido en los hechizos lanzados por los enemigos. Tienen toda la información necesaria para poder desplazar el hechizo por el mundo y dañar al jugador.

- **ForceResolution**. Script que fuerza una resolución 16:9 en la escena del bosque, debido a un problema con la resolución en las

*builds standalone* para Mac en algunos MacBookPro Retina. Es un problema conocido [7] [8] [9], y la fuente de esta solución es la siguiente: http://gamedesigntheory.blogspot.com/2010/09/controlling-aspect-ratio-in-unity.html

- **ForestController**. Script que contiene el algoritmo de autogeneración de bosques pseudoaleatorios, así como el cálculo de niveles y dificultad. Será explicado en detalle en el apartado de autogeneración de bosques pseudoaleatorios.

- **GameplayController**. Script que contiene elementos generales del Gameplay como el sistema de pausa, guardado de datos, panel de controles…

- **IdleAnimationRnd**. Script que lleva cada uno de los habitantes del valle, y simula una animación de idle usando los sprites de cada uno. Es un algoritmo que usando un valor aleatorio de tiempo mínimo y máximo de frame de la animación, y un valor aleatorio para el siguiente sprite de idle a utilizar simula perfectamente las animaciones para todos los aldeanos sin necesidad de crear los animators ni animations para cada uno de ellos.

- **InitEvent**. Script que controla el evento de inicio de partida dónde podemos observar cómo nuestro personaje se forma a partir de una Flor de la Luz como producto de algún tipo de hechizo.

- **InventoryController**. Script que gestiona todo el inventario del jugador, así como el inventario rápido de las manos izquierda y derecha, la adición o eliminación de elementos del inventario, la obtención de pasivas, la gestión del dinero (compras y ventas)… Se explicará con más detalle en el apartado del inventario.

- **Item**. ScriptableObject creado para la plantilla de todos los items de nuestro juego. Se explicará más en detalle en el apartado de items.

- **Merchant**. Script que poseen los mercaderes y los artesanos. Permite especificar que objetos vende dicho habitante, y abre (y cierra) el panel tienda con los mismos.

- **MyDialogue**. Clase utilizada para parsear los JSON de los diálogos.

- **PlayerController**. Script que controla las funciones principales del jugador como el movimiento, su orientación, el evento de su muerte, y los parámetros de su animator. Es aquí donde se hace la traducción del stat de agilidad a velocidad, y dónde afectan las ralentizaciones. Quiero destacar que en este script se establece la

orientación del personaje según unos valores float que coinciden con los valores threshold que toma en el Blend Tree del animator. Por último, he de comentar también que así es dónde se controla el empuje del personaje al recibir daño.

- **PlayerDeath**. Script del evento de muerte del personaje. Lo lleva el prefab de la Flor de la Luz muerta.

- **Projectile**. Script contenido en los proyectiles y hechizos lanzados por el jugador. Tienen toda la información necesaria para poder desplazar el hechizo por el mundo y dañar a los enemigos.

- **SceneController**. Script que contiene las funciones para navegar entre escenas y reiniciar los datos guardados.

- **SceneDuration**. Script para controlar la duración de una escena y pasar de forma automática a la siguiente. Usada en la escena de mi logo personal.

- **ShopController**. Script que controla las tiendas del juego. Será comentado con claridad en el apartado de la tienda.

- **Speaker**. Script que llevan todas las entidades que entablan un dialogo. Hace uso del DialogueController.

- **StatsController**. Script que centraliza las estadísticas de una entidad del juego, así como el sufrimiento de daño y daños en el tiempo. Es usado tanto por los guardianes como por los enemigos. Será explicado mejor en el apartado de Stats.

- **StatsPanel**. Script que lee los datos del StatsController del jugador y los dibuja en un panel de la UI.

- **WeaponController**. Script donde se centraliza el uso de armas por parte de los guardianes. Se explica en detalle en el apartado de armas.

### *Sounds*

Contiene todos los clips de sonidos utilizados. Son parte de un bundle de sonidos que compré en la Unity Asset Store.

### *Sprites*

Aquí están almacenados todos los sprites del videojuego.

- **Hand Painted – Town Tileset**. Asset comprado en Unity Asset Store.
- **AD**. Logo personal creado con Adobe Illustrator CC 2019.
- **Art Package**. Asset comprado en Unity Asset Store.

- **Art Package > Interface**. Assets modificados en Pixlr para adaptarlos al estilo buscado.
- **Buttons**. Asset gratuito de Unity Asset Store.
- **Characters**. Asset gratuito de Itch.io
- **Icons**. Asset gratuito de Itch.io.
- **Items**. Asset gratuito de Itch.io.
- **Stats**. Asset gratuito de Itch.io.
- **Windows**. Asset gratuito de Unity Asset Store.

## 4.5 Esquema de arquitectura del juego y sus componentes

En este apartado mostraré la arquitectura de cada escena que compone mi videojuego.

### *Escena: Logo principal*

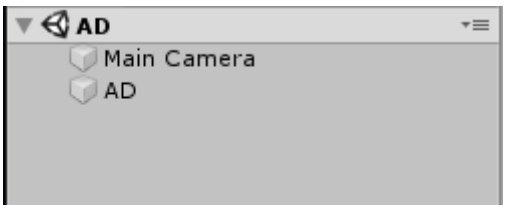


**Ilustración 31. Escena AD. Arquitectura.**

### *Escena: Inicio*

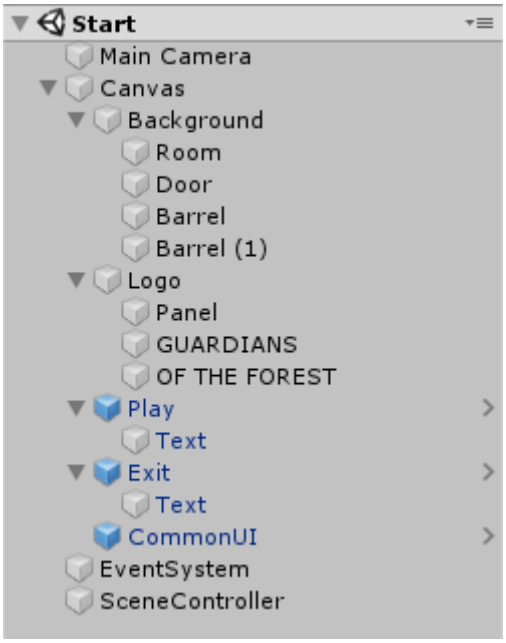


**Ilustración 32. Escena Start. Arquitectura.**

### *Escena: Valle de la Luz*

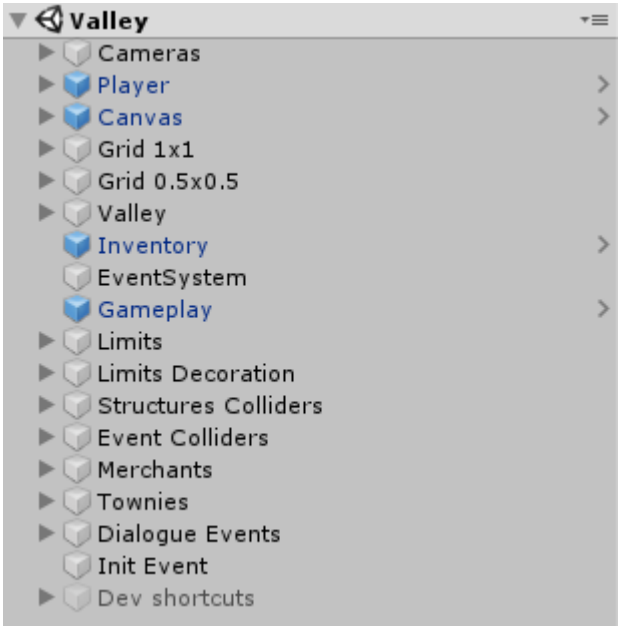


**Ilustración 33. Escena: Valley. Arquitectura**

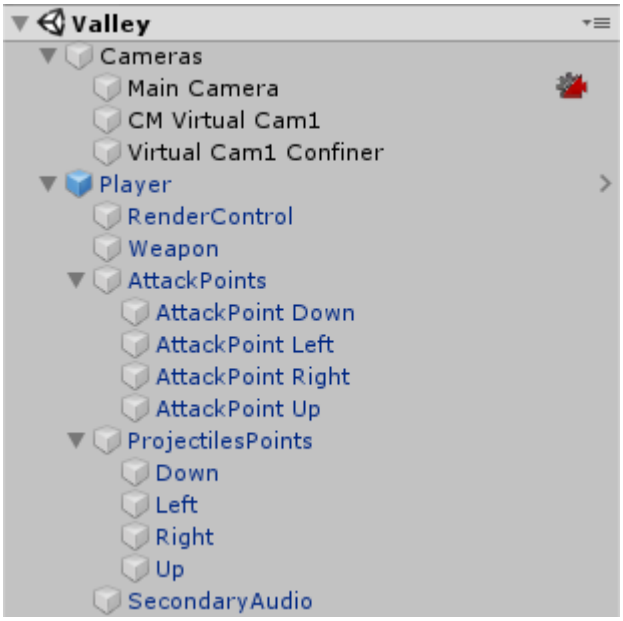


**Ilustración 34. Escena: Valley. Cámaras y Player.**

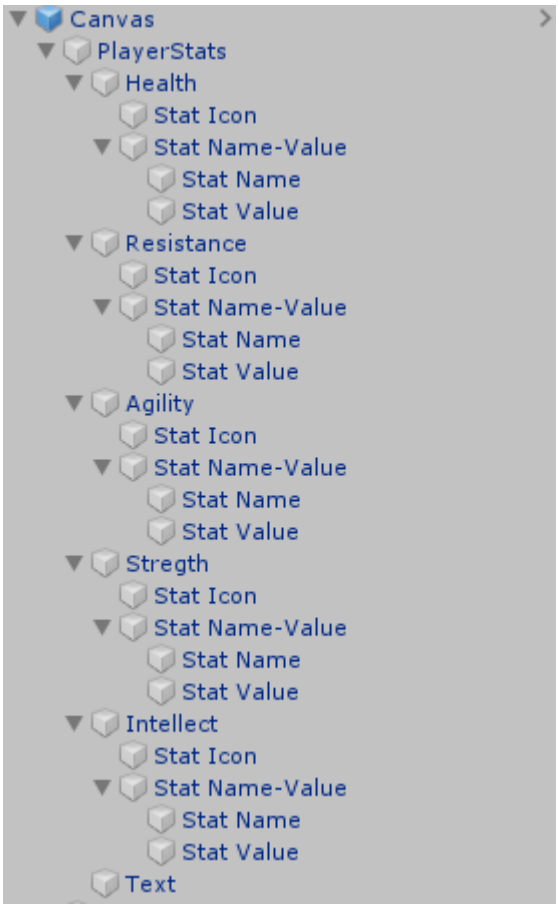


**Ilustración 35. Canvas: Player stats.**

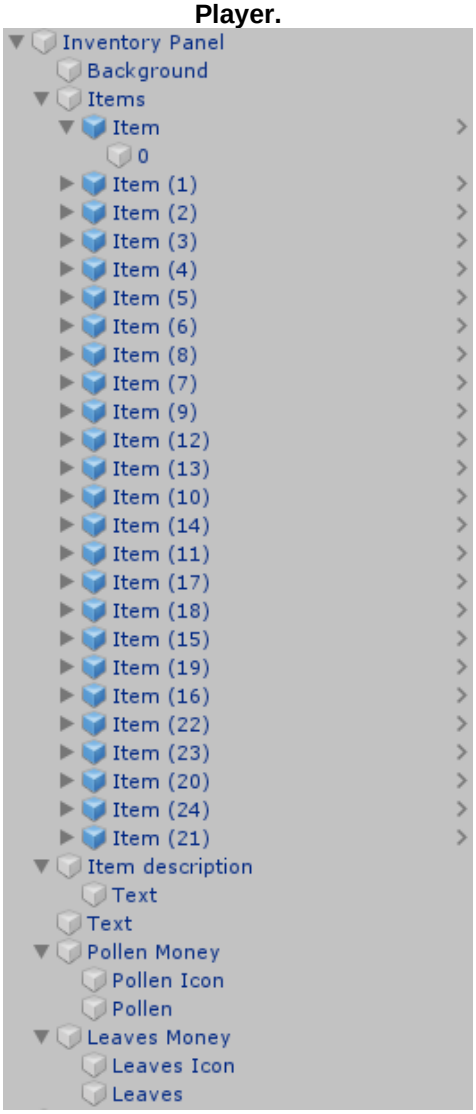


**Ilustración 36. Canvas: Inventario**

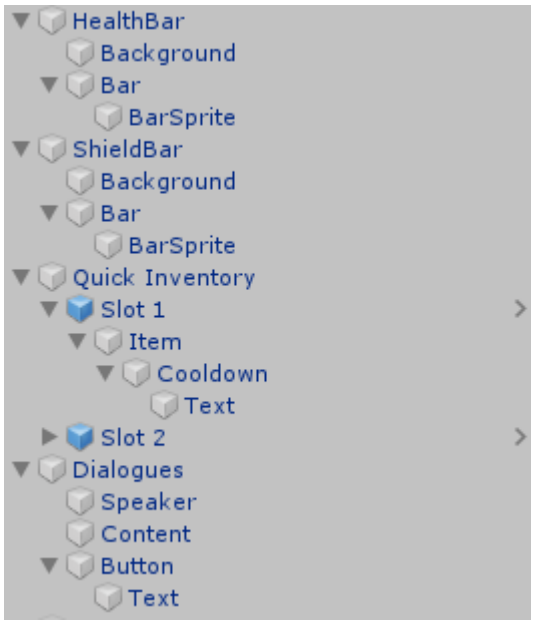


**Ilustración 37. Canvas: Barras de vida y escudo, Inventario rápido, Diálogos**

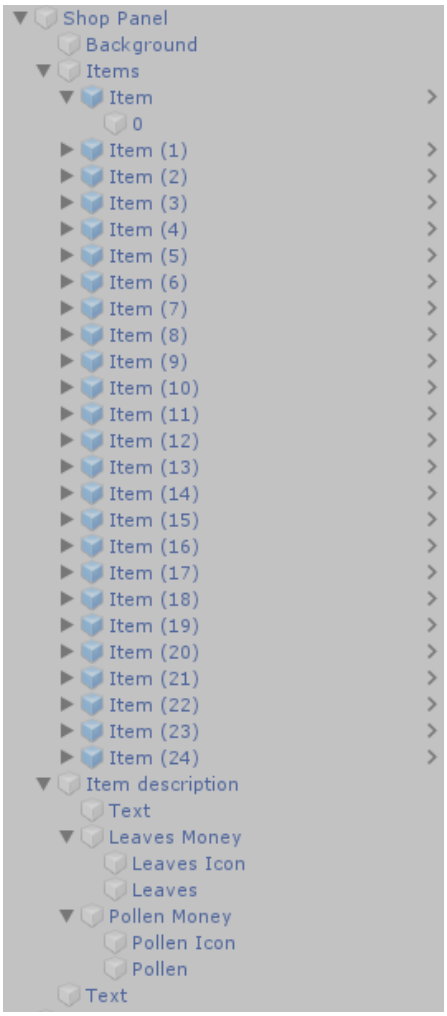


**Ilustración 38. Canvas: Tienda**

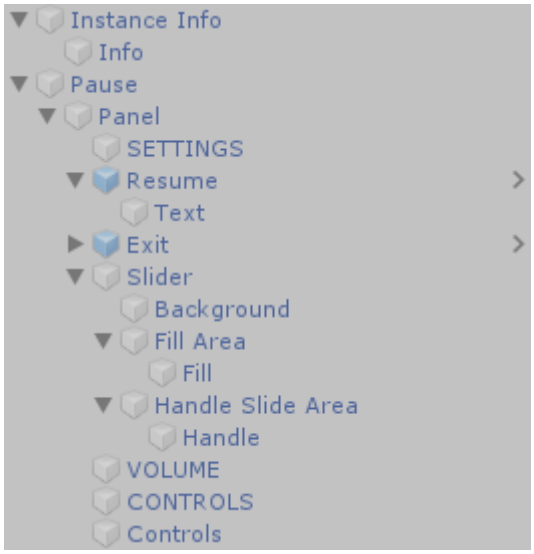


**Ilustración 39. Canvas: Información de estancia y pausa**

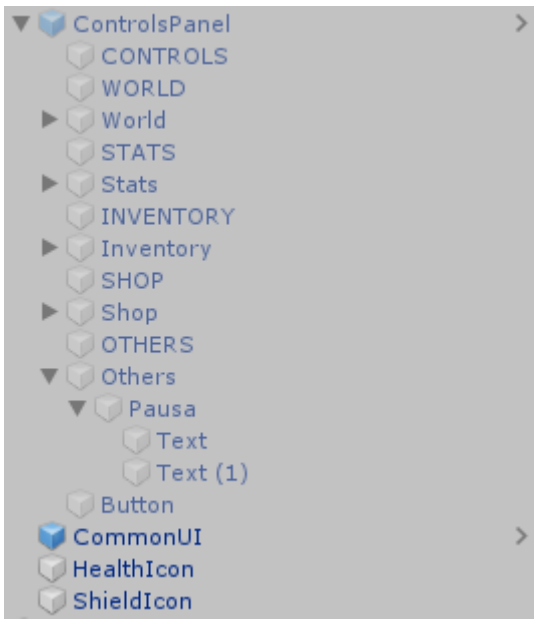


**Ilustración 40. Canvas: Controles, CommonUI e Iconos**


**Ilustración 41. Escena: Valley. Grids**


**Ilustración 42. Escena: Valley. Decoración**

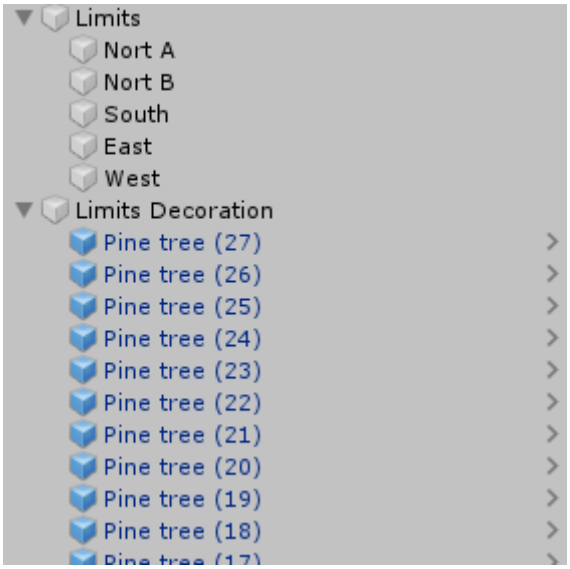


**Ilustración 43. Escena: Valley. Límites del mapa.**

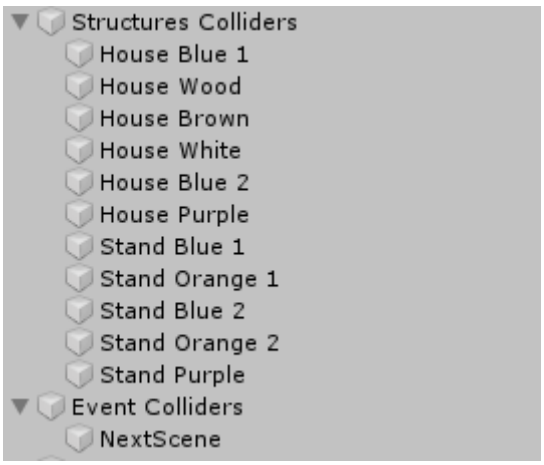


**Ilustración 44. Escena: Valley. Colliders de profundidad y eventos de colisión**


**Ilustración 45. Escena: Valley. Habitantes.**


**Ilustración 46. Escena: Valley. Eventos de diálogos, evento de inicio, y atajos de desarrollo**

### *Escena: Bosque maldito*

Los detalles de los elementos “Player” y “Canvas” serán omitidos porque son prefabs y ya han sido mostrado en la escena del Valle de la Luz.

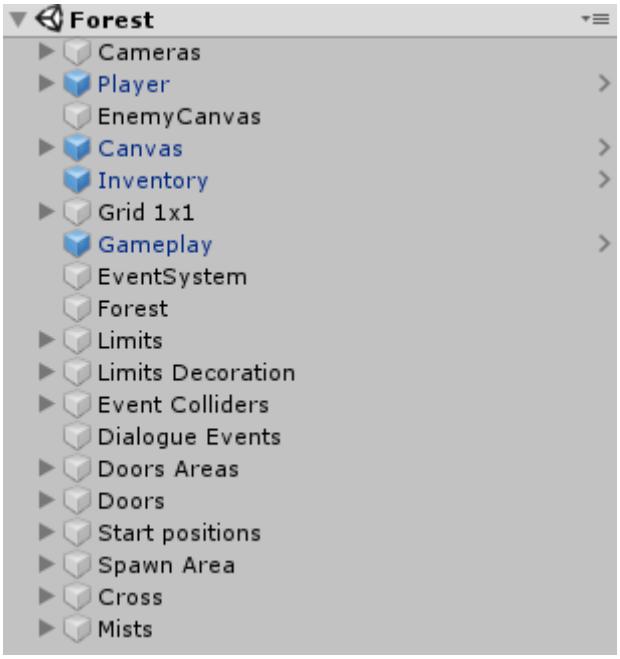


**Ilustración 47. Escena: Forest. Arquitectura.**


**Ilustración 48. Escena: Forest. Cámaras**

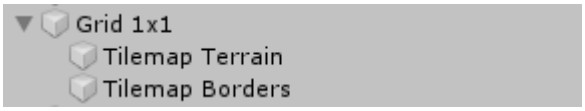


**Ilustración 49. Escena: Forest. Grids.**

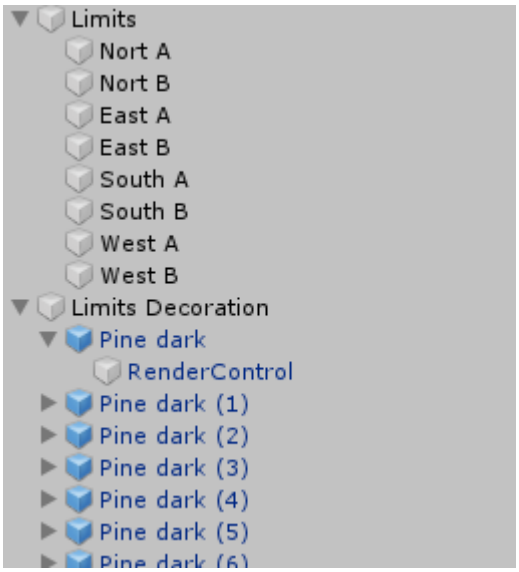


**Ilustración 50. Escena: Forest. Límites del mapa.**

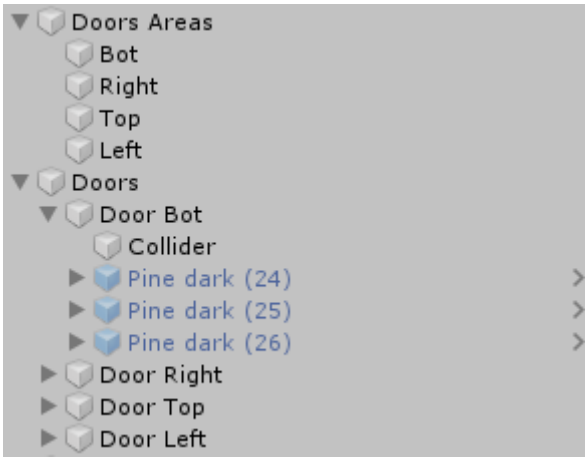


**Ilustración 51. Escena: Forest. Puertas**

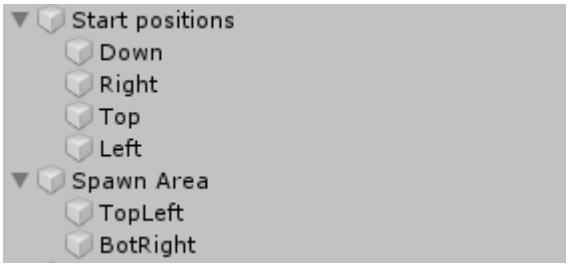


**Ilustración 52. Escena: Forest. Posiciones de inicio y área de spawn**

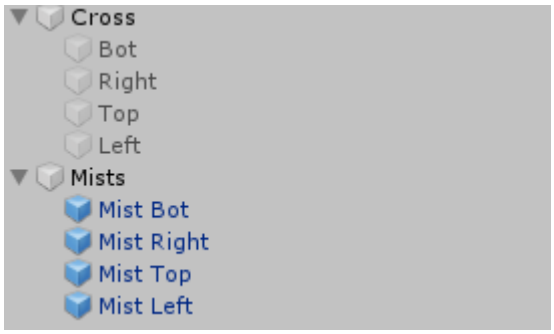


**Ilustración 53. Escena: Forest. Cruces y nieblas.**

*Escena: Fin*


**Ilustración 54. Escena: Fin. Arquitectura**

## 4.6 Ítems scriptables

Para facilitar la programación, diseño, y balanceo de objetos se realizó un scriptable object llamado Item. En esta clase se recogen todos los datos necesarios para definir los diferentes tipos de objetos, armas,

equipamiento, y consumibles que existen en Guardians of the Forest, así como sus características. Tenemos desde los valores de daño, área de efecto, enfriamiento, probabilidad de daño en el tiempo… para ítems equipables, como valores pasivos, cantidad y duración de consumibles, *powerup* de stat en savias, cómo los costes monetarios del objeto, su icono para la UI, y clips sonoros.

Se definido un conjunto de placeholders (por ejemplo, `PLACEHOLDER_DAMAGE = "{damage}"`) los cuales permiten establecer desde el inspector la descripción que queramos utilizándolos, y que estos serán reemplazados cuando se vayan a mostrar en la UI por el valor de su característica correspondiente.

Además, también se ha definido un color definido por el tipo de ítem que sea, que actualmente es utilizado en el nombre del ítem así para el usuario es más fácil distinguirlos. De esta forma tenemos:

- Gris. Color por defecto.
- Rojo. Color para las armas.
- Cian. Color para el equipamiento.
- Violeta. Color para los consumibles.
- Amarillo. Color para los artilugios.
- Negro. Color para las savias.

A continuación, se muestra un ejemplo de ítem desde el inspector, y cómo se ve in-game.


**Ilustración 55. Scriptable in-game: UI.**

**Ilustración 56. Scriptable in-game: World**

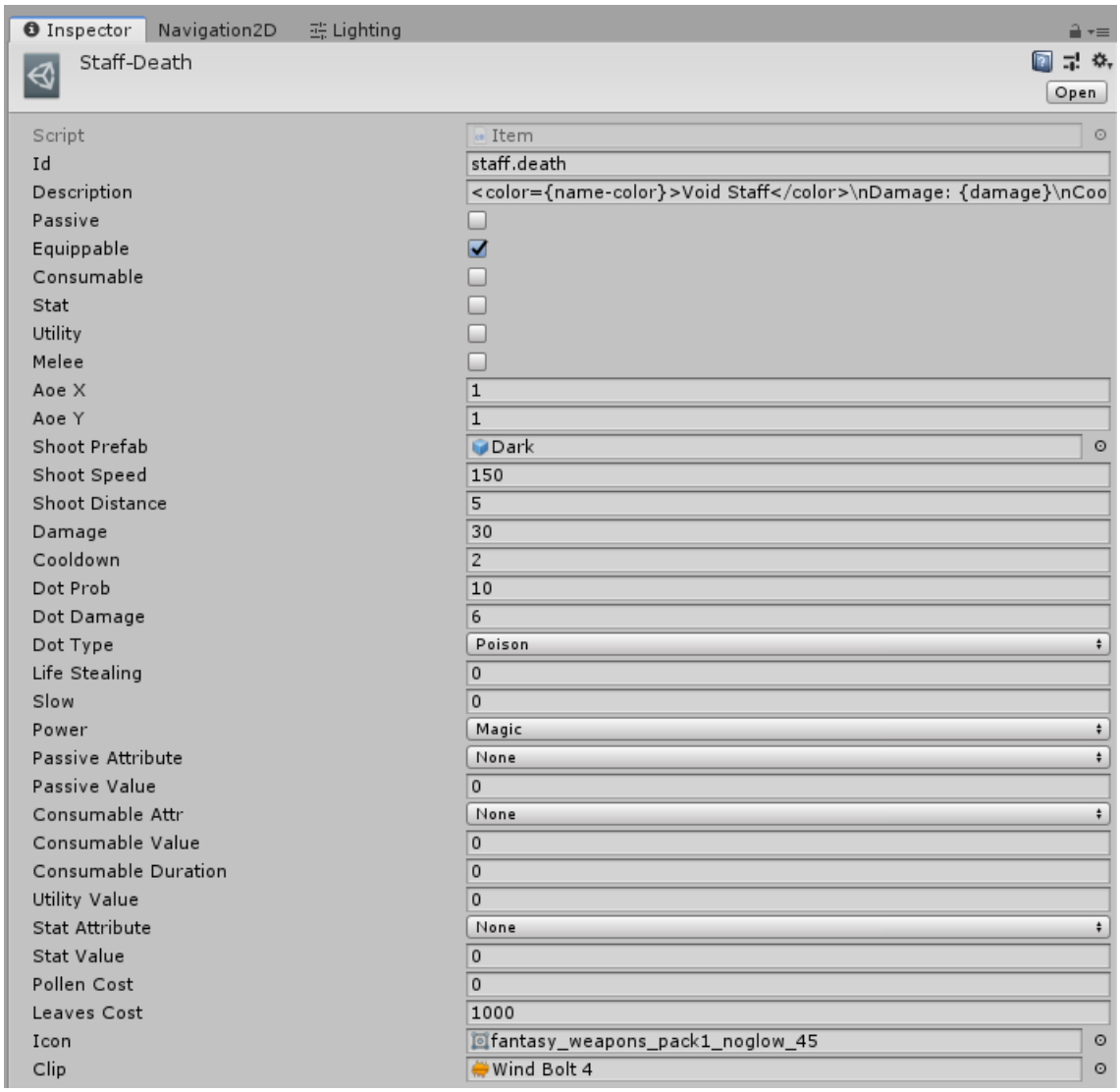


**Ilustración 57. Scriptable en Inspector.**

## 4.7 Inventarios y tiendas

Tanto para el inventario como para las tiendas se ha seguido una estrategia bastante similar. Se explicará inicialmente el inventario y luego, se detallarán las diferencias que presenta la estrategia para las tiendas.

### *Inventario*

La lógica del inventario reside en el script InventoryController. En memoria los datos del inventario consisten en un array del tipo scriptable Item (de tamaño configurable, actualmente veinticinco), mientras que para la UI consiste en un array de imágenes con background a las cuales se les cambia el *source*.

En la inicialización, si no existen datos guardados, el array de items estará vacío, y se carga los slots de imágenes de la UI mediante código,

iterando por los hijos del panel. En caso de que existan datos, en memoria estos se copian desde la clase estática Data.

Por otro lado, existe también el inventario rápido, es decir, las manos izquierda y derecha de nuestro guardián, donde podemos equiparnos las armas. En memoria, esto también se trata de un array de tamaño configurable (dos elementos actualmente) los cuales reaccionan a los clics del ratón y que se explicará con más detalle en el apartado de armas. Para la UI tenemos otra lista con los slots que corresponden a las imágenes de las casillas con background. La inicialización es similar a como se hace para el inventario.

En este script se presentan métodos para añadir y quitar items del inventario, y las lógicas que esto conlleva, por ejemplo, añadir o eliminar estadísticas pasivas si el ítem las presenta, controlar que no esté en enfriamiento y queramos venderla o desequiparla, controlar que esté en el inventario rápido (manos izquierda y derecha) para eliminarlas de aquí también, etc. Tenemos también los métodos para efectuar la compra, pues el “dinero” (polen y hojas) reside en el inventario del guardián.

En referencia a los slots de la UI del inventario, todos tienen *events triggers* con el ratón con *Pointer Enter* y *Pointer Exit* (para abrir y cerrar el cajetín de descripción de dicho ítem), y con *Pointer Click* para interactuar con el ítem (equipar armas en mano izquierda o derecha, usar consumibles, vender items…).

Por otro lado, esté script es el encargado de reflejar los cooldowns de los items del inventario rápido. En memoria es un array de float del mismo tamaño que el inventario rápido, donde el índice indica el ítem al que hacer referencia, y el contenido es el tiempo restante de enfriamiento, y en la UI se representa con una lista de layers de imágenes semitransparente y texto que están colocados encima de los slots.

Al usar un ítem desde WeaponController (hablaremos más en profundidad en el apartado de armas), se hacen llamadas a los métodos públicos *IsInCooldown()*, *GetQuickItem()*, y *AddCooldown()* con los índices de cada mano (izquierda y derecha, 0 y 1) para comprobar que el ítem se pueda usar, obtenerlo, y añadirle cooldown tras su uso. Además, en el *Start()* de InventoryController se hace una llamada de *InvokeRepeating()* al método privado *CooldownTicks*() para que se ejecute cada 0.1 segundos. Este método es el encargado de reducir el cooldowns de las armas equipadas que lo tengan, permitiendo que en el *Update()* se dibuje la cuenta atrás para aquellos con un cooldown superior a 0.

Por último, quiero destacar que para ahorrar cálculos y no tener que recorrer continuamente los datos, el dibujado del inventario completo (*DrawInventory()*) y del rápido (*DrawQuickInventory()*) se realizan solo si ha habido alguna actualización en dichos datos, o si acabamos de abrir el inventario después de tenerlo cerrado.


**Ilustración 58. Inventario.**


**Ilustración 59. Inventario rápido.**

### *Tiendas*

En relación con las tiendas, toda la lógica reside en el script ShopController. La estrategia de almacenamiento, uso, y dibujado de datos es igual que para el inventario completo, a excepción que aquí en el panel de descripción de un ítem se muestra además su coste.

Entre las diferencias notables, para inicializar la tienda con los objetos, se realiza desde el script Merchant (el cual poseen los vendedores) dónde se llama al método público *OpenShop()* y se le manda por parámetros la lista de items scriptables que vende.

Por otro lado, el *event trigger Pointer Click* llama al método privado *InteractItemEvent()* el cual hace a su vez una llamada al método publico *BuyItem()* del InventoryController para comprobar si se puede efectuar la compra.


**Ilustración 60. Tienda**

## 4.8 Uso de armas

El uso del armamento en Guardians of the Forest está centralizado en la clase WeaponController, componente presente en los guardianes.

El videojuego presenta por ahora cinco clases de armas distintas, espadas, dagas, bastones, varitas, y arcos. Cada arma presenta a su vez presenta en su scriptable un ID y en su prefijo la clase a la que pertenece, por ejemplo “sword.”.

Al GameObject del guardián se le añadió cómo hijo un GameObject “Weapon” en el centro de este con un Sprite Renderer, así como “AttackPoints” con GameObjects para marcar la posición de ataque melee en las cuatro direcciones, y “ProjectilesPoints” con GameObjects para marcar la posición de ataque a distancia en las cuatro direcciones.

Se realizaron animaciones para los cinco tipos de armas en las cuatro direcciones, modificando la posición y rotación del GameObject hijo “Weapon”. Estas animaciones se pueden encontrar en Animations/Male, por ejemplo, Attack-BowDown.

Para usar las armas debes usar el clic izquierdo o derecho en el mundo, y tener un arma equipada en dicha casilla. Para comprobar que los clics se realizan sobre el mundo y no sobre elementos de la UI, se programó la función *IsMouseOverUI()* dónde se recoge el PointerEventData y se realiza un RaycastAll comprando el número de resultados que devuelve. Además, también se comprueba con InventoryController que tenga un arma en dicha mano, y que esta no se encuentra en cooldown.

Justo en el momento de usar el arma, se añade un cooldown global a ambas manos, cuya función es que exista un breve lapso entre dos ataques seguidos para evitar dos ataques simultáneos.

Durante la ejecución del ataque, primero se reemplaza el Sprite del GameObject “Weapon” por el correspondiente en la propiedad *icon* del scriptable del item, y luego ejecuta el trigger para la animación correspondiente, por ejemplo, “AttackSword”, y el Blend Tree correrá el clip correspondiente según la orientación del jugador. A continuación, dependiendo del tipo de arma ocurre lo siguiente:

- **Espadas y dagas**. Mientras se ejecuta la animación, se llama al método *AttackArea()*, al cual se le pasa el scriptable del ítem y la orientación del jugador. En este método se calcula un área a partir del punto de “AttackPoints” correspondiente, utilizando los atributos aoeX y aoeY del scriptable, y se obtiene todos los overlaps en dicha área con el layer de los enemigos. A continuación, para cada uno de los enemigos alcanzados, se obtiene su correspondiente StatsController y se les hace daño con el método publico *Hurt().*

- **Varitas y bastones**. Mientras se ejecuta la animación, se comprueba si el poder del arma es curativo.

    - Si es un arma curativa, se llama a *Heal()*, y se le pasa el scriptable del ítem. A continuación, se llama al método *Heal()* del StatsController del player, y se instancia las partículas del hechizo contenidas en el scriptable.

    - Si es un arma destructiva, se llama a método AttackProjectile(), al cual se le pase el scriptable del ítem y la orientación del jugador. En este método se calcula desde cuál de los puntos de ProjectilesPoints se va a lanzar el hechizo, y se instancia el prefab del hechizo contenido en el scriptable. Estos tienen dentro un componente Projectile, en el cual se les establece mediante el método *Launch(),* las características de este, como el daño que hace, la dirección, el alcance, la velocidad … Estos proyectiles tienen rigidbody y colliders para detectar la colisión con enemigos y dañarles mediante la obtención de su StatsController y la llamada a *Hurt().*

- **Arcos**. Mientras se ejecuta la animación, se llama al método *AttackProjectile()* y se realizan los mismos pasos que para los bastones destructivos.

Para calcular el daño realizado con las armas, se hace una llamada a *GetTotalDamage(),* y se le pasa el scriptable del ítem. Aquí se comprueba si el poder es físico o mágico, y se le suma al poder el arma la estadística que tengamos en fuerza o intelecto, respectivamente.

Además, para los proyectiles también se hace una llamada a *GetTotalDistance()* para obtener la distancia máxima de alcance de dicho proyectil (hechizo o flecha). Para ello se suma el alcance del arma

con la estadística oculta de alcance del jugador (se verá mejor en el apartado de Estadísticas de personajes).

Existe además para cada animación un evento en frame de animación en su finalización, dónde se llama al método *StopWeapon()* el cúal limpia el sprite del gameObject “Weapon”.

Por último, para finalizar el ataque se hace una llamada al método *AddCooldown()* del InventoryController para añadir el enfriamiento correspondiente al arma. Antes de hacerlo, se hace un cálculo del enfriamiento que debe establecerse por si el jugador tiene porcentaje de reducción de enfriamiento en sus estadísticas.

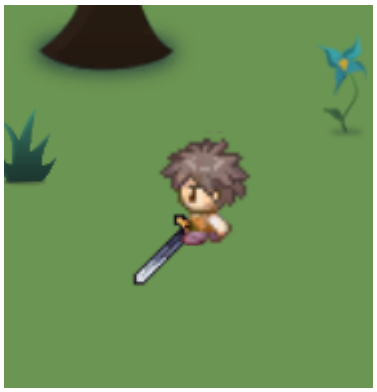

**Ilustración 61. Ataque con espada.**

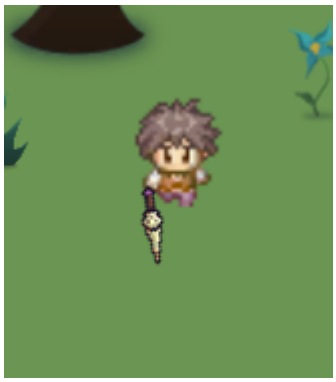

**Ilustración 62. Ataque con daga.**

**Ilustración 63. Ataque con bastón.**

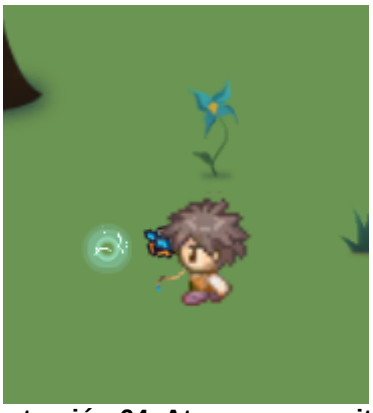

**Ilustración 64. Ataque con varita.**

**Ilustración 65. Curación con bastón: 1**

**Ilustración 66. Curación con bastón: 2**

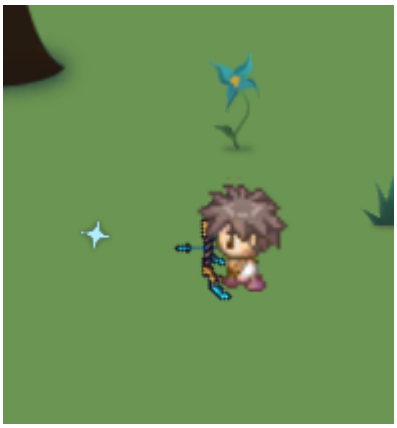

**Ilustración 67. Ataque con arco.**

## 4.9 Estadísticas de personajes

Las estadísticas de todos los personajes, guardines y enemigos están centralizadas en el componente StatsController. Existen dos tipos de stats o estadísticas, las generales y las ocultas.

Estadísticas generales:

- **Vida**. Determina la salud máxima del personaje.
- **Resistencia**. Determina la amortiguación del daño recibido.
- **Agilidad**. Corresponde con la velocidad del personaje.
- **Fuerza**. Su puntuación se le suma al daño de las armas física.
- **Intelecto**. Su puntuación se le suma al daño de las armas mágicas.

Estadísticas ocultas:

- **Barrera**. Reduce parte del daño recibido. Los guardianes lo consiguen mediante la ingestión de pociones.
- **Alcance**. Aumenta la distancia de hechizos y flechas. Los guardianes lo consiguen con la adquisición de cierto equipo.
- **Reducción de cooldown**. Reduce el cooldown de las habilidades. Los guardianes lo consiguen con la adquisición de cierto equipo.
- **Invisibilidad.** Te hace inalcanzable a la vista de los demás, por lo que no pueden atacarte. Los guardianes lo consiguen mediante la ingestión de pociones.

Aquí podemos encontrar todos los métodos de utilidad para obtener los stats actuales, sumarles o quitarle valores, y comprobar que nuestro personaje sigue con vida.

En esta clase además reside el método público *Hurt(),* con el que podemos dañar a los personajes. Si el daño recibido no es verdadero, podremos mitigar hasta un máximo igual a la resistencia que tengamos,

llamemos a este resultado daño efectivo. Luego si tenemos barrera, podremos mitigar el daño efectivo hasta un máximo igual a la barrera actual, la cual se agota y es consumida. El daño restante con el que nos quedamos es el que será restado a la salud actual, y si tenemos un *source* del daño, será empujado en dirección contraria. Tras esta operación, se comprueba las posibles pasivas que tuviera al arma con la que recibido el daño.

- **Robo de vida**. El atacante se cura un porcentaje del daño final realizado a la salud del objetivo igual al del robo de vida del arma utilizada.

- **Ralentización**. Si el personaje no está ya ralentizado, se le aplica en una corutina un estado de ralentización durante un tiempo predeterminado en un porcentaje igual al aplicado por el arma. La ralentización se traduce a una disminución de la velocidad del personaje en su correspondiente script de movimiento (PlayerController o EnemyAI), y al spawn de partículas de ralentización alrededor del individuo.

- **Daño en el tiempo (DoT)**. Algunas armas tienen una probabilidad de dañar en el tiempo al enemigo (actualmente por quemado o envenenamiento). El daño de las DoT es verdadero, por lo que ignoran las resistencias del enemigo. Estas DoT se resuelven en corutinas en las que se instancia las partículas correspondientes al DoT alrededor del objetivo, y donde se llama a *Hurt()* cada tic de DoT durante un tiempo determinado.


**Ilustración 68. Estadísticas del guardián.**

**Ilustración 69. DoT quemado.**

**Ilustración 70. Ralentización.**

**Ilustración 71. DoT envenenado.**

## 4.10 Inteligencia Artificial de los enemigos

En Guardians of the Forest actualmente existen un total de cuatro enemigos, los segadores, los golems, los ancianos, y los jinetes. A todos se las ha dotado de IA y está se encuentra centralizada en el script EnemyAI, el cual está presente en cada uno de ellos.

El corazón de esta IA es una máquina de estados, donde a partir de un estado inicial y mediante los tics en *Update* y los triggers de eventos para el estado actual ocurren cada una de las transiciones, dotándoles de una inteligencia y comportamiento bastante bueno.

Se creó una interfaz IEnemyState dónde se definen las funciones que estarán presente en todos los estados:

- ***FixedUpdateState()*** y ***UpdateState().*** Son invocados para el estado actual en los *FixedUpdate()* y *Update()* de EnemyAI.

- ***GoToXXXState()***. Representan las transiciones a otros estados, por ejemplo, *GoToPatrolState().*

- ***OnTriggerYYY().*** Son invocados en los eventos de colisión OnTrigger de EnemyAI.

- ***ListenTriggerEvents().*** Son invocados en diferentes funciones públicas de EnemyAI que son llamadas por eventos de frame en animaciones.

Se definieron los estados PatrolState, SeekState, AttackState, HealState, y Seek State. Cada uno de ellos serán explicados en subapartados siguientes, así como las transiciones que presentan.

Por último, quiero destacar también que los enemigos disponen de un rango de visión realizado con un circle collider trigger.

### *EnemyAI*

Es la clase principal de IA y movimiento de los enemigos. En el constructor se inicializan todos los estados que lleva la máquina, y se establece como esta actual el estado inicial de la misma, PatrolState.

Como se ha explicado con anterior, el funcionamiento reside en la ejecución del nodo actual. Por ejemplo, en el *FixedUpdate()* se hace una llamada a *currentState.FixedUpdateState()*, y en *Update()* una llamada a *currentState.Update(),* y lo mismo para los eventos *OnTriggerEnter2D()*, *OnTriggerStay2D()*, y *OnTriggerExit2D()*. De esta manera será la lógica que reside en cada nodo quien se encargue de realizar las transiciones correspondientes.

Además de la ejecución de la máquina de estado, esta clase también presenta el resto de las utilidades que complementan a la IA.

Por un lado, para el desplazamiento de los enemigos se está haciendo uso de Navigation2D, un asset que extiende el componente Navigation de Unity para poder ser utilizado en 2D. De esta forma, los enemigos han sido establecidos como agentes, y cada uno de los posibles obstáculos como obstáculos de la malla de navegación.

Por otro lado, se programaron una serie de funcionalidades, así como lógicas y transiciones globales, es decir, ciertas lógicas o transiciones que pueden ocurrir desde cualquier estado.

- **Empuje por daño**. Comprueba si el personaje debe ser empujado por el último daño recibido.

- **Condición de muerte**. Comprueba si el enemigo sigue vivo, y en caso negativo realiza la transición del estado actual hacia el estado de muerte, *currentState.GoToDeathState()*.

- **Muerte.** Función pública que es llamada desde el estado de muerte. Destruye al enemigo e instancia las partículas de desvanescencia. Este método se encuentra aquí y no dentro del estado de muerte porque también implica la destrucción de otros elementos asociados al enemigo los cuales no son accesibles desde el estado, como las barras de vida en el canvas de enemigos.

- **Orientación del sprite.** Existe una función para actualizar la orientación del enemigo según un parámetro que actualizan los nodos de estado, acordes a la lógica de cada uno, con la llamada a alguna de las siguientes funciones:

    - *LookTarget().* Mantiene el sprite orientado hacia el target.

    - *LookRandom().* Orienta el sprite de forma aleatoria cada cierto tiempo.

    - *LookToDestination().* Mantiene el sprite orientado hacia el punto de destino al que se desplaza.

- **Estado de curación**. En caso de ser un enemigo con poderes de curación, comprueba cada cierto tiempo parametrizable si debe transitar al estado de curación para sanarse.

- **Provocación**. Es llamado cuando el enemigo recibe daño, y se comprueba si el enemigo puede visualizar a la fuente de daño para marcarlo como objetivo, y transitar desde el estado actual al estado de persecución, *currentState.GoToSeekState()*.

- **Control de la velocidad del agente**. Es invocado desde los estados en los que el enemigo se desplaza, y actualiza la velocidad del agente según sus stats y posibles ralentizaciones.

- **Rango de ataque**. Determina si el enemigo está en rango para atacar al objetivo.

- **Obtención de puntos aleatorios del mapa**. Obtiene puntos aleatorios en el mapa dentro de la zona de spawn que forma la instancia.

- **Funciones públicas llamadas por eventos en frame de animación**. Trasladan la información de evento ocurrida en la animación al estado actual, utilizando el enumerable EnemyTriggerEvent.

- **Parametrización de ataques y habilidades**. Permiten configurar las características del enemigo.

### *PatrolState*

En este estado el enemigo se desplaza de forma aleatoria por la estancia.

Para buscar un punto al que ir, se hace uso de la función *GetRandomPoint()* de EnemyAI. Una vez el enemigo establece su próximo punto, también se guarda en cuanto tiempo decidirá buscar un nuevo punto al que ir.

Mientras el enemigo está en este estado mantiene su mirada en el punto al que se dirige.

Presenta las siguientes transiciones propias (sin contar las globales):

- **SeekState**. Se produce cuando el jugador se encuentra dentro del campo de visión del enemigo y es visible (es decir, no es invisible). Se realiza con el evento de *OnTriggerStay*.

### *SeekState*

En este estado el enemigo persigue al target que tiene marcado mientras comprueba si ha alcanzado el rango de ataque necesario.

Para ello, se comprueba primero que el target siga siendo visible (no invisible), y en segundo lugar que el enemigo no esté en periodo de recuperación (por retroceder al haber recibido daño). A continuación, se actualiza el destination del Navigation del agente con la posición del target. Para comprobar el rango de ataque, hace uso de la función *IsAtAttackRange()* de EnemyAI.

Durante este estado, el enemigo mantiene su visión en el target.

Presenta las siguientes transiciones propias (sin contar las globales):

- **PatrolState**. Se produce en caso de que el target deje de ser visible, es decir, que se vuelva invisible.

- **AttackState**. Se produce en caso de que se encuentre en rango de ataque.

### *AttackState*

En este estado el enemigo realiza sus ataques contra el target mientras controla que este siga siendo visible y se encuentre en rango de ataque.

Los enemigos presentan un cooldown entre ataques, el cual también se controla dentro de este estado, y cuando el enemigo puede atacar su habilidad dependerá del tipo de enemigo que sea:

- **Enemigo físico**. Se lanzará la función *RunPhysicalAttack()*. Esta función lanza el trigger de animación “PhysicalAttack”. En estas animaciones hay un evento de frame, para marcar cuando efectuar el daño, que lo recibiremos desde EnemyAI a través de *ListenTriggerEvents()*, que invocará a su vez al método *PhysicalAttackEvent()*. En esta función se calcula un área de daño (según parametrización) en la dirección en la que está orientado el enemigo y se obtienen todos los overlaps con la capa del jugador. Con cada uno de los resultados, se obtiene su StatsController y se le hiere con *Hurt().*

- **Enemigo mágico**. Se lanzará la función *RunMagicalAttack().* Esta función lanza el trigger de animación “MagicalAttack”. En estas animaciones hay un evento de frame, para marcar cuando lanzar el hechizo, que lo recibiremos desde EnemyAI a través de

*ListenTriggerEvents()*, que invocará a su vez al método *MagicalAttackEvent()*. Los enemigos mágicos disponen de dos gameObjects hijos para determinar la posición de lanzamiento de hechizo. En este método se obtiene la posición del GameObject hijo correspondiente a la orientación del enemigo desde el que se lanzara el proyectil, y se instancia el prefab del hechizo del enemigo. Además, se calcula el ángulo que hay entre el enemigo y el target, y se gira el proyectil para que esté orientado hacia él. Por último, se obtiene de la instancia del hechizo el componente EnemySpell, y se le establece todas las características de este, como el daño que hace, la dirección, el alcance, la velocidad … Estos proyectiles tienen rigidbody y colliders para detectar la colisión con los guardianes y dañarles mediante la obtención de su StatsController y la llamada a *Hurt().*

- **Enemigo híbrido**. Se lanzará la función HybridAttacker(). Aquí se comprueba si el target está dentro del alcance cuerpo a cuerpo o no, mediante el parámetro hybridRange. En caso afirmativo, se lanza la función *RunPhysicalAttack()*, y en caso negativo la función *MagicalAttackEvent()*. Este comportamiento es posible debido a que el rango normal de ataque de estos enemigos corresponde con caso en el que puede atacar con hechizos a distancia, mientras que, si nos acercamos más, pues pasan al modo cuerpo a cuerpo.

Quiero destacar aquí que el ancestral es un enemigo físico con la habilidad de curación, cuya transición está controlada desde EnemyAI.

Durante este estado, el enemigo mantiene su visión en el target.

Por último, he de comentar que está presente la función *GetTotalDamage()* con la cual se obtiene el daño total causado, que es igual al daño del enemigo sumado a su fuerza o intelecto, dependiendo de si está haciendo daño físico o mágico.

Presenta las siguientes transiciones propias (sin contar las globales):

- **Patrol**. Se produce si el target deja de ser visible, es decir, se vuelve invisible.

- **Seek**. Ocurre cuando deja de tener en rango de ataque al target.

### *HealState*

Este estado es alcanzado por enemigos que tienen habilidad para curarse, como el ancestral.

Si aún no se ha curado, lanza el método *RunHealing()*, el cual invoca el trigger de animación “Healing”. Esta animación tiene un evento en frame

para que la curación tenga efecto en el momento preciso del casteo de habilidad, evento le llega al estado desde EnemyAI a través de *ListenTriggerEvents()*, que a su vez invoca al método *HealingEvent().* En este método se realiza la curación, y se actualiza la última vez que se curó en EnemyAI, pues recordemos que este estado se puede llegar desde cualquier otro, es una transición “global”, y es EnemyAI quien lo lanza cada cierto tiempo y según la probabilidad de curación del enemigo.

Una vez ha finalizado la curación, se retoma el estado anterior, al que podemos ir porque tenemos guardado siempre una referencia al estado anterior del estado actual.

### *DeathState*

Es el estado de muerte del enemigo.

Si el enemigo no ha ejecutado aun su muerte, la ejecuta llamando a EnemyAI *KillEnemy().* Este método instancia las partículas de muerte del enemigo, y lo destruye a él y a su barra de vida del canvas de enemigos.

No existen transiciones para salir del estado de muerte.

## 4.11 Generación de estancias pseudoaleatorias

El objetivo actual de Guardians of the Forest es conseguir llegar al nivel máximo de estancias de bosque superados, en las que la dificultad aumenta conforme al nivel. Para darle al usuario una sensación de navegación dentro de un bosque y de estar superando estancias diferentes, se decidió generar las estancias o salas del bosque de forma pseudoaleatoria

Está lógica reside en la combinación del script ForestController con la escena Forest, la cual sirve de nivel base.

A continuación, se muestra la base que es utilizada para la generación de estancias, y luego se explicará más en detalle las lógicas utilizadas dentro de ForestController.

**Ilustración 72. Base de las estancias.**

### *Generación de estancias*

En el script ForestController encontramos en la parametrización los arrays de los diferentes tipos de objetos que se instanciarán en el mundo para generar el bosque, así como el número mínimo y máximo de elementos para cada grupo. De esta manera tenemos objetos sólidos (obstáculos), objetos *triggers* (traspasables), y objetos de hierbas (para rellenar el suelo), y el número máximo y mínimo para cada grupo. Luego, a parte, tenemos un array de enemigos y un array de jefes, y estos no presentan mínimos ni máximo porque la cantidad de cada uno de ellos depende de la estancia y el nivel.

En Guardians of the Forest el bosque está organizado en niveles, y a su vez, cada nivel está formado por cuatro estancias. Una estancia es la sala que el guardián debe superar en cada iteración, y el nivel es factor que aumenta la dificultad de las estancias. La estancia número 4 de cada nivel corresponde al enfrentamiento con un jefe, dónde tras superarlo y avanzar, se aumentará el nivel y, por consiguiente, la dificultad. De forma interna, solo existe el número de estancia, el cual se divida entre 4 para obtener el nivel en el que estamos, y usamos el módulo para obtener la estancia dentro del nivel. Para saber en qué nivel y estancia nos encontramos, se crearon los métodos estáticos y públicos *GetLevelNumber()* y *GetInstanceNumber()*.

Para la instanciación de elementos en el mapa se definió un algoritmo en la función ***PlaceRandomObjectInInstance()***, la cual recibe por parámetros el array de la colección a instanciar, y el número mínimo y máximo de elementos. Inicialmente se calcula un número aleatorio de elementos a instanciar entre el mínimo y el máximo posible. En caso de que nos encontremos en un enfrentamiento contra un jefe y la colección

que se está instanciando sea la de las hierbas, este número se aumentará considerablemente. A continuación, para cada elemento a instanciar se escoger uno al azar de la colección. En caso de que la colección coincida con la de los jefes, se comprueba que no sea igual al último jefe enfrentado si dicha colección es mayor a uno. Luego este objeto se instancia en el mundo dentro de una posición que se encuentre dentro del área *spawneable* de la escena, la cual la obtenemos con la función *getRandomPositionInSpawnArea()*, y comprobamos si no colisiona con ningún otro objeto de la escena haciendo uso de la funcionalidad *OverlapCollider()* del componente Collider2D de nuestro objeto instanciado, o si está obstruyendo alguna de las puertas de la estancia, usando el método propio *CheckIfObjectIntersectDoors().* En caso de que exista dicha colisión, el objeto es destruido e instanciado de nuevo en otra posición y se vuelve a realizar las comprobaciones. Este proceso se repite hasta un número máximo de intentos parametrizable, eliminando y saltando la iteración de este elemento en caso de que no seamos capaces de situarlo correctamente, para evitar así que problemas de rendimiento durante la generación de estancias e incluso la posibilidad de bloqueo por caer en un bucle infinito. En el caso de que hayamos colocado satisfactoriamente nuestro elemento y el elemento no pertenezca a la colección de enemigos o jefes, nos dirigimos a configurar su renderizado en el mundo. Establecemos su sortOrding acorde a su posición Y en el mundo, para lograr el efecto de profundidad buscado, y en el caso de que no pertenezca a la colección de obstáculos, invertimos su sprite en el eje X de forma aleatoria (50% probabilidad). Para finalizar, añadimos el objeto instanciado a una lista que es devuelta como output del método.

Una vez explicado el algoritmo que vamos a utilizar para la instanciación de elementos, explicaremos el orden y el funcionamiento de la generación de la estancia al completo.

1. **Localización del jugador**. Cada vez que el jugador atraviesa un pasillo/puerta, se almacena desde que dirección venía y luego se le coloca en posición natural para que su orientación al caminar coincide con la que estaba realizado. Por ejemplo, si el jugador camina hacia arriba para atravesar la puerta superior, en la siguiente estancia aparecerá por la puerta inferior orientado hacia arriba.

2. **Generación de la estancia.** Se aumenta el número de estancia actual, y se realizan los siguientes pasos en orden:

    a. **Instanciación de enemigos**. Se obtiene el número de estancia, y si no es estancia de jefe, se hace uso de la función *PlaceRandomObjectInInstance()* para instanciar tantos elementos de la colección enemigos igual al número de estancia, es decir, uno, dos o tres. Si nos encontramos en la estancia de jefe, es decir el número cuatro, hacemos el mismo procedimiento, pero con la colección de jefes e

indicando que solo se instanciará uno. A continuación, utilizamos el array de objetos instanciados para mandarlos a la función creada ***SetEnemiesDificulty().*** En esta función, dependiendo del nivel actual se obtiene un factor de mejora (20% por nivel superado) y se configura la dificultad de los enemigos. A todos los enemigos se le mejora la salud, y de forma controlada (limitando que no supere un umbral configurable), su agilidad. A continuación, se procede a realizar las mejoras específicas por tipo de enemigo:

i. **Físico**. Se mejora su fuerza.

ii. **Mágico**. Se mejora su intelecto, y de forma controlada la velocidad de sus hechizos.

iii. **Sanador**. Se mejora de forma controla su poder de curación.

iv. **Succionador de vida.** Se mejora de forma controla su porcentaje de robo de vida.

b. **Instanciación de obstáculos**. En caso de que no nos encontremos en la estancia jefe, se llama a la función *PlaceRandomObjectInInstance()* con la colección de obstáculos y su número mínimo y máximo de elementos.

c. **Instanciación de triggers**. Se llama a la función *PlaceRandomObjectInInstance()* con la colección de elementos traspasables y su número mínimo y máximo de elementos.

d. **Instanciación de hierbas.** Se llama a la función *PlaceRandomObjectInInstance()* con la colección de hierbas y su número mínimo y máximo de elementos.

3. **Generación de puertas**. Se calcula de forma aleatoria cual será la puerta/pasillo que se abrirá una vez el guardián supere la sala.

4. **Información de estancia.** Se lanza una corutina que realiza una animación de introducción a la estancia dónde se le muestra al guardián en qué nivel y estancia se encuentra.

5. **Condición de finalización**. Se lanza en un *InvokeRepeating()* la función *CheckFinishedInstance()* para que cada 2 segundos (configurable) compruebe si el guardián ya ha superado la sala, es decir, si ya ha acabado con todos los enemigos que había. En caso afirmativo, se desactiva la niebla y colliders del pasillo que contiene la puerta al siguiente nivel, así como activar el collider con evento el evento CrossForest correspondiente para guardar

los datos y recargar la escena, provocando la generación de una siguiente estancia.

Para finalizar me gustaría comentar que prefabs existen en cada una de las colecciones actualmente:

- **Obstáculos.** Pino maldito, abeto maldito, árbol maldito, montículo de rocas, y roca grande.
- **Traspasables**. Flor verde, y flor maldita.
- **Hierbas**. Hierbajo pequeño, mediano, y grande.
- **Enemigos**. Segador y golem.
- **Jefes**. Ancestral y jinete.

A continuación, mostraré algunas capturas de pantalla de diferentes estancias generadas durante las partidas.

**Ilustración 73. Estancia pseudoaleatoria 1.**

**Ilustración 74. Estancia pseudoaleatoria 2.**

**Ilustración 75. Estancia pseudoaleatoria 3.**

**Ilustración 76. Estancia de jefe 1.**

**Ilustración 77. Estancia de jefe 2.**

# 5. Diseño de niveles

## 5.1 Valle de la Luz

Originalmente, se realizó un boceto del Valle de la Luz que lucía un poco diferente a su aspecto final.

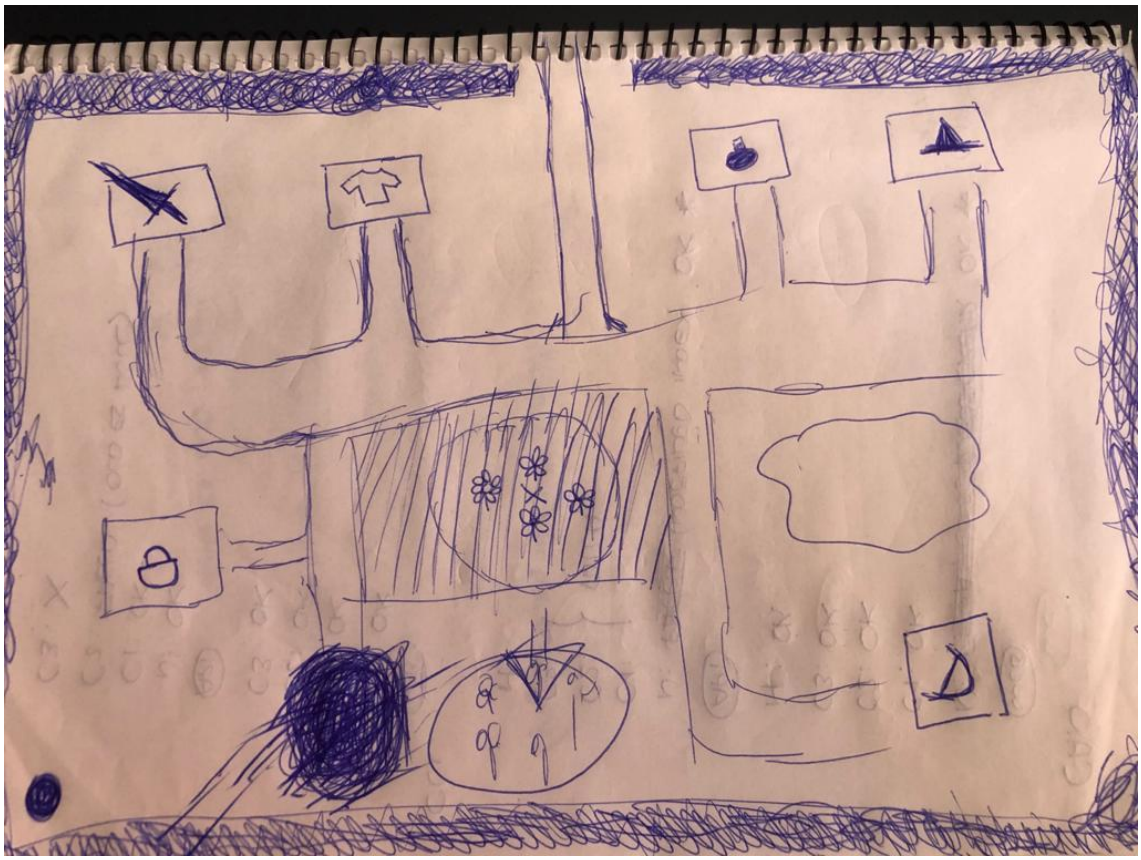

**Ilustración 78. Boceto del Valle de la Luz**

En un inicio las flores de la luz estarían en el centro del mapa y desde aquí surtirían los caminos para cada una de las tiendas. Sin embargo, esto no me convenció mucho, pues el centro del poblado debería ser un sitio concurrido y dónde se realice algún tipo de actividad, entonces decidí trasladar la plantación para la parte sur del mapa, en medio de una prolongación del bosque, y decidí colocar en el centro un mercado, dónde poder comprar las savias de conocimiento.

De esta manera, se obtuvo el siguiente diseño para el Valle de la Luz:

**Ilustración 79. Mapa del Valle de la Luz.**

Nuestro jugador comienza en la parte sur del mapa, en esa pequeña abertura de bosque dónde se encuentran las flores de la luz plantadas. Justo encima, situado en el centro del poblado, tenemos el mercado, el lugar donde encontraremos a los comerciantes que venden savias de conocimiento, y de dónde salen caminos para las casas de cada uno de los artesanos, así como el camino que se dirige al norte para entrar al bosque maldito. El camino que se dirige al oeste lleva a la casa de la cuidadora, quien venderá artilugios en próximas versiones, mientras que si seguimos el camino que va al sureste, nos encontraremos con el cazador, que nos suministrará diferentes tipos de arcos y utilidades de distancia. Si nos dirigimos al noroeste, nos encontramos (de izquierda a derecha), la casa del guerrero, que vende espadas, dagas, y elementos de fuerzas, y la casa de la tejedora, quien vende equipamiento con utilidades diversas. Si nos dirigimos al noreste, nos encontramos (de izquierda a derecha) la casa del alquimista, quien nos suministrará pociones de diferente tipo, y la casa de la bruja, dónde se venden artilugios mágicos y útiles de cooldown.

El valle, al encontrarse en el centro de un bosque, se encuentra bordeado por árboles, los cuales hacen de barrera física al jugador y la cámara para que no puedan salirse del mismo, a no ser que cojan el camino del norte.

## 5.2 Bosque maldito

El bosque maldito fue diseñado como una base sobre la que generar de forma automática diferentes estancias. Está se mantuvo fiel a su diseño original, y no sufrió demasiadas alteraciones.

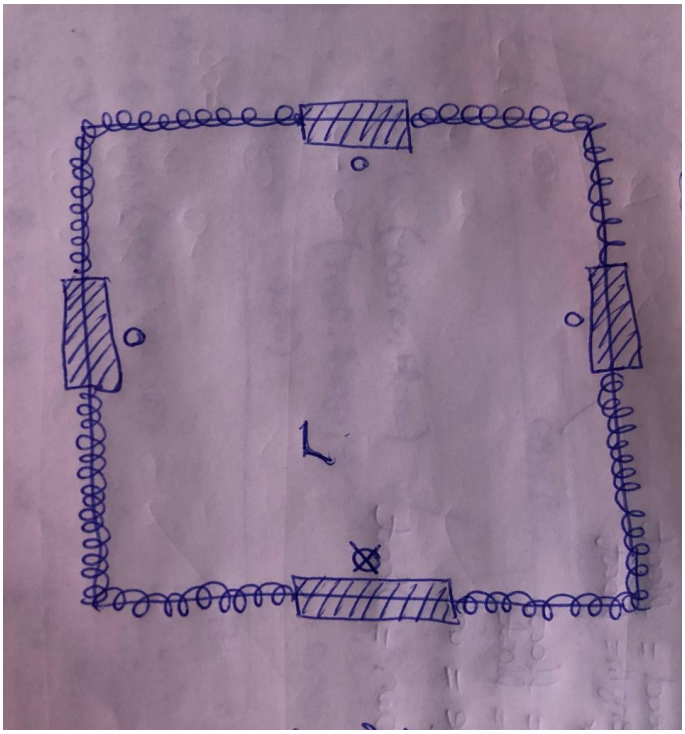

**Ilustración 80. Boceto del bosque maldito.**

Se decidió que la extensión de este debía coincidir con el tamaño de la cámara del jugador, para tener unas estancias con un tamaño suficiente para jugar y que el jugador pudiera observarlo en su plenitud en todo momento, a diferencia del Valle de la Luz.

**Ilustración 81. Mapa del bosque maldito.**

En este mapa tenemos una estancia rectangular correspondiente a áreas dentro del bosque maldito, las cuales están rodeadas por árboles que hacen de muralla, y dónde hay cuatro clareas por donde poder

continuar. Sin embargo, estas presentan una densa niebla demoníaca la cual no podemos atravesar, y tras purificar de demonios la zona, una de ellas se desvanecerá permitiéndonos continuar por dicho camino.

# 6. Experiencias con usuarios

## 6.1 Participantes

Se realizaron pruebas con jóvenes entre 18 y 29 años que aceptaron de forma voluntaria probar el videojuego y dejar un feedback en una encuesta realizada con Google Form. Se obtuvo una participación de un total de 17 personas.

## 6.2 Pruebas realizadas

Las pruebas consistieron en enviar una versión compilada del videojuego que contenía los hitos esperados terminados, con el fin de ver si los usuarios se sentían cómodos con el gameplay, entendían el juego, y podían desarrollar la aventura sin problemas. Además, también servía ver si algún *tester* descubría algún bug, o algún ítem, habilidad, o enemigo desequilibrado en comparación con el resto.

De estas pruebas se obtuvieron los siguientes feedback relevantes:

- Los ítems yelmo y pechera daban demasiada resistencia y provocaba que en los primeros niveles no pudieras sufrir daño. Fue corregido, los valores de resistencia que otorgaban fueron reducidos.

- Las barras de vida de los enemigos se veían por encima del panel de pausa. Fue corregido, se creó un canvas solo para elementos de los enemigos el cuál se renderiza por debajo del canvas con los elementos del jugador.

- Los efectos de sonido de algunos botones de la UI no se ajustaban al volumen general de los settings. Fue corregido, se aplicó el canal de master a dichos AudioSource.

- La barra correspondiente a la barrera, que se encuentra junto a la de salud, era confusa y a no ser que te tomaras la poción no se entendía que era. Fue corregido, se añadió iconos a las barras de vida y barrera.

- El camino que tomar tras finalizar una estancia era difícil de ver, pues en un inicio las nieblas solo estaban en el camino del que venías y en el camino que se abriría, y las otras dos salidas estaban cerradas con árboles. Fue corregido, se añadió la niebla a los cuatro caminos, y una vez finalizas la estancia, se desvanece la niebla con el camino a continuar.

Además, se les pidió a estos usuarios que especificaran que versión habían probado, así como una valoración en diferentes aspectos, como jugabilidad, dificultad, o expectativas frente a futuras versiones. Se obtuvieron los siguientes resultados:

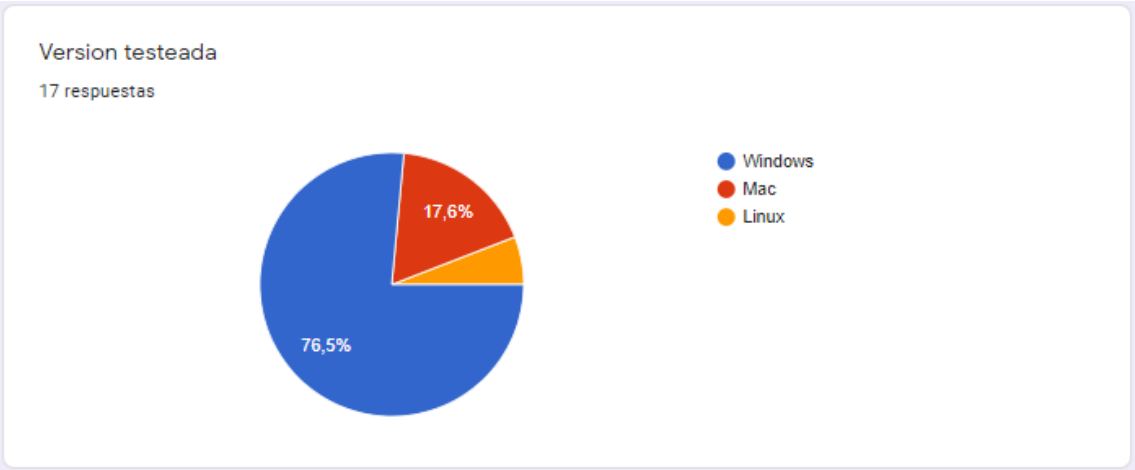


**Ilustración 82. Testers: Versión**

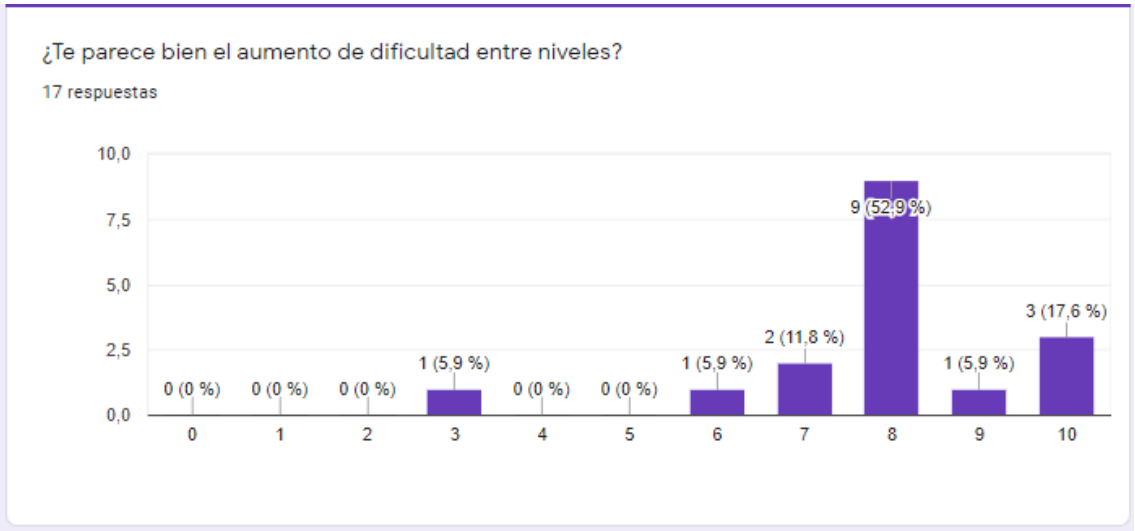


**Ilustración 83. Testers: Dificultad**

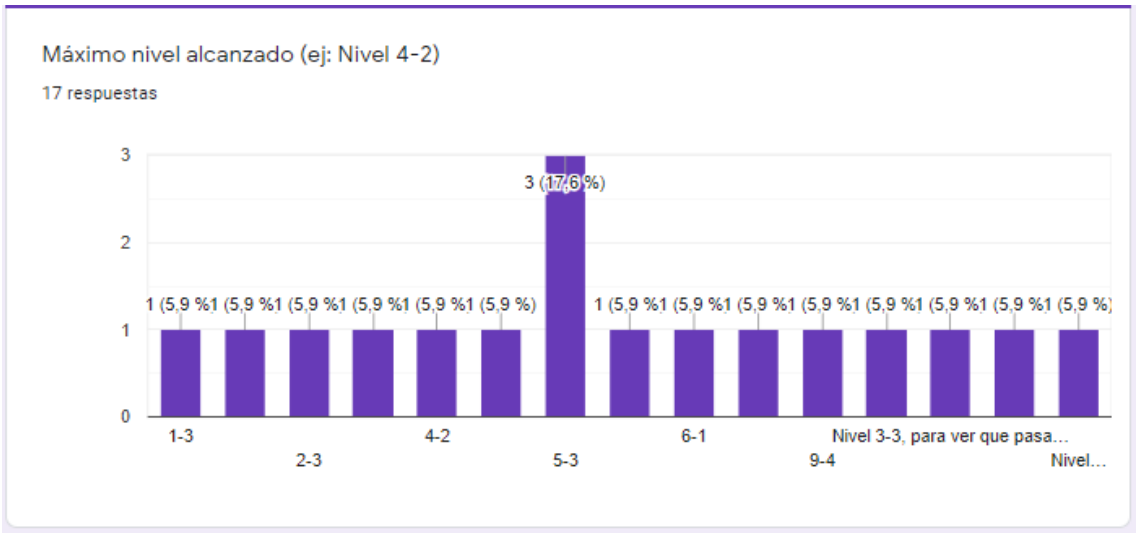


**Ilustración 84. Testers: Nivel alcanzado**

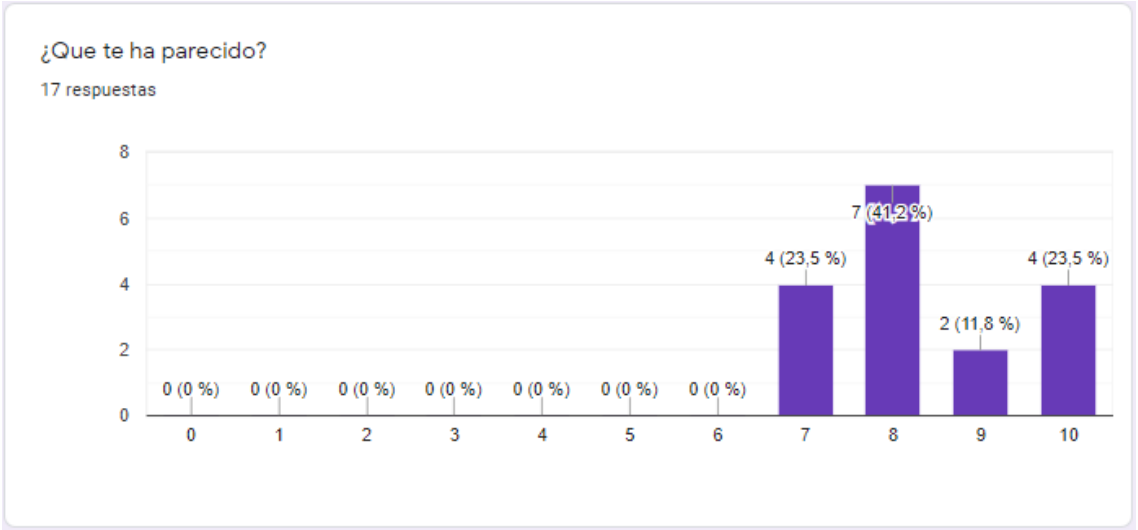


**Ilustración 85. Testers: Valoración**

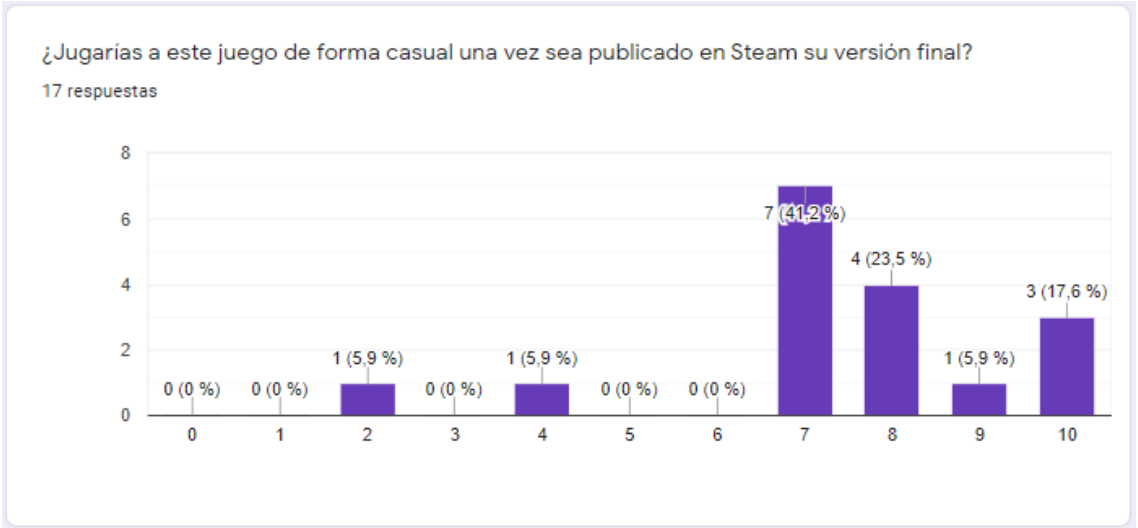


**Ilustración 86. Testers: Steam**

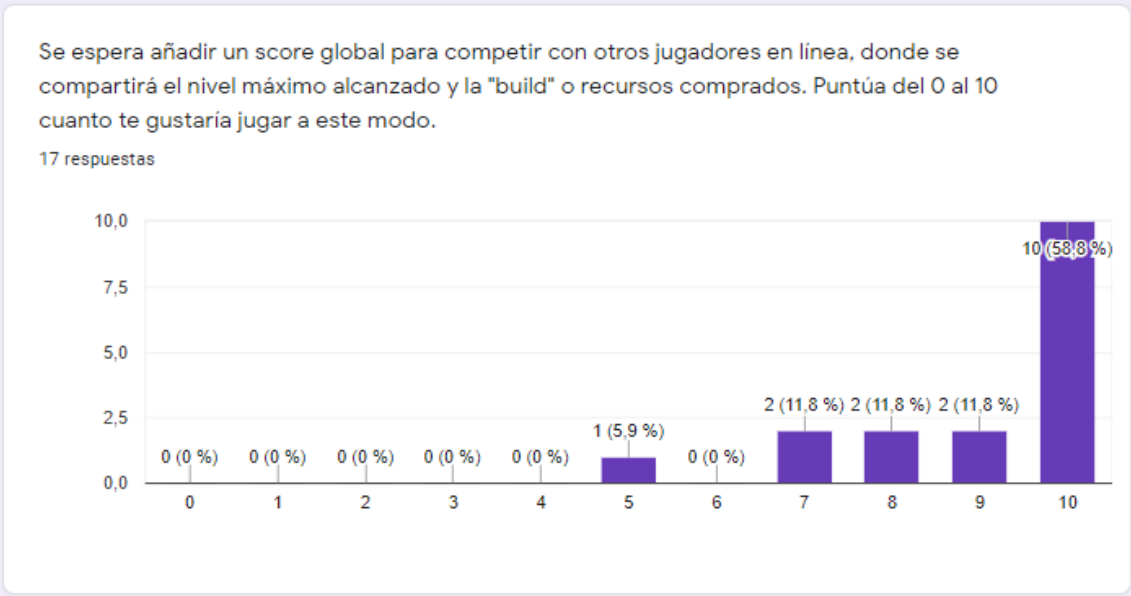


**Ilustración 87. Testers: Competir en línea**

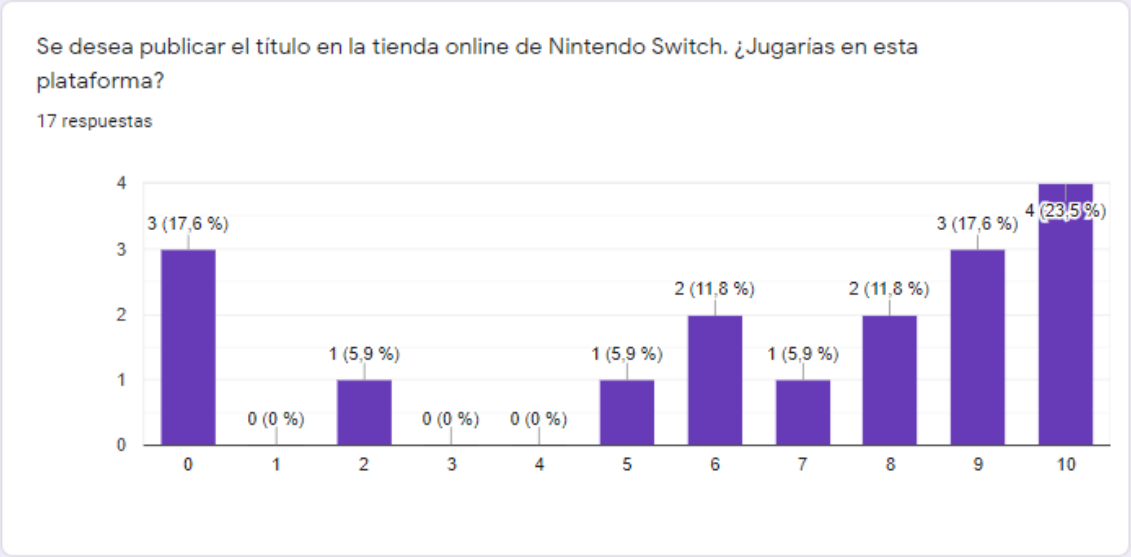


**Ilustración 88. Testers: Switch**


**Ilustración 89. Testers: Testing futuro**

## 6.3 Conclusiones

Realizar esta experiencia con usuarios me ayudo no solo a descubrir algunos desbalances, como es el caso del yelmo o la pechera, si no que me ayudó a tener una perspectiva de la visión del jugador frente al juego, pues a ninguno se les dijo cómo tener que jugar, si no que fueran ellos mismo quienes a través de los diálogos y eventos del juego supieran avanzar. Gracias a esto último, pude ver como varios de estos usuarios tuvieron ciertas dificultades para detectar cuál era el siguiente camino que tomar tras superar una estancia, o que algunos usuarios no entendieran para que era la barra vacía que salía junto a la de salud, lo que me permitió mejorar estos aspectos y hacer el juego más *user-friendly* para los jugadores.

Por otro lado, a los usuarios se les preguntó acerca de la dificultad percibida, así como el escalado de la misma conforme aumentaban los niveles, y obtuve una valoración bastante positiva sobre la misma, y viendo el resultado de nivel máximo alcanzado puedo observar que varios usuarios alcanzaron niveles bastantes altos, teniendo en cuenta que cada nivel está formado por cuatro estancias.

Además, a los testers se les preguntó si estarían dispuestos a jugar al videojuego si fuera publicado en tiendas cómo Steam o Nintendo Shop. La respuesta fue bastante positiva en cuanto a la plataforma Steam, mientras que para Nintendo Shop fue algo más dispar. Diversos usuarios me comentaron en diferentes apartados sobre opiniones, que no lo jugarían en Nintendo Switch debido a que no disponen de la consola o no es su plataforma de juego habitual.

Por último, se les pidió a los usuarios una valoración sobre el juego en general, y para mi agrado el juego gustó bastante entre los jugadores, y la mayoría estaría dispuesto a jugar en futuras versiones dónde haya competición en línea, por ejemplo.

# 7. Manual de usuario

## 7.1 Requerimientos técnicos del hardware

**Sistema operativo**:

- Windows 7 (SP1+) o Windows 10, ambos de 64-bit
  - CPU: Soporte para el set de instrucciones SSE2.
  - GPU: DX10, DX11, o DX12

- Mac Sierra 10.12.6+
  - CPU: Soporte para el set de instrucciones SSE2.
  - GPU: Intel o AMD con Metal.

- Linux Ubuntu 16.04, Ubuntu 18.04, o CentOS 7
  - CPU: Soporte para el set de instrucciones SSE2.
  - GPU: Nvidia y AMD con Vulkan o OpelGL 3.2+.

**Almacenamiento en disco duro:** 2 GB

## 7.2 Instrucciones de juego

Para lanzar el juego hay que abrir el ejecutable “Guardians of the Forest”, y seleccionar la calidad de juego deseada, así como la resolución.


**Ilustración 90. Ejecutable del juego.**

Una vez iniciemos el juego, veremos la pantalla de presentación con mi logo personal.

**Ilustración 91. Escena del logo personal.**

A continuación, veremos el menú de inicio del juego y deberemos de darle al botón PLAY.


**Ilustración 92. Escena de inicio.**

Una vez accedes al mundo de juego, veremos el Valle de la Luz, y cómo una de las flores está sufriendo una transformación y aparece un guardián, nuestro jugador. En la parte inferior saldrá un bocadillo de texto que nos guiará en el inicio de la historia. Debemos darle clic al icono > que veremos abajo a la derecha, o pulsar la barra espaciadora para continuar estos diálogos.


**Ilustración 93. Introducción al Valle de la Luz.**

Al final de estos diálogos, se explica al jugador que si desea ver todos los controles de juego acceda al menú de pausa pulsando *escape*.

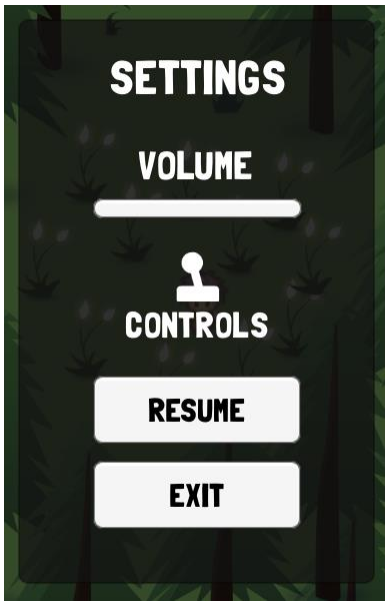


**Ilustración 94. Panel de pausa.**

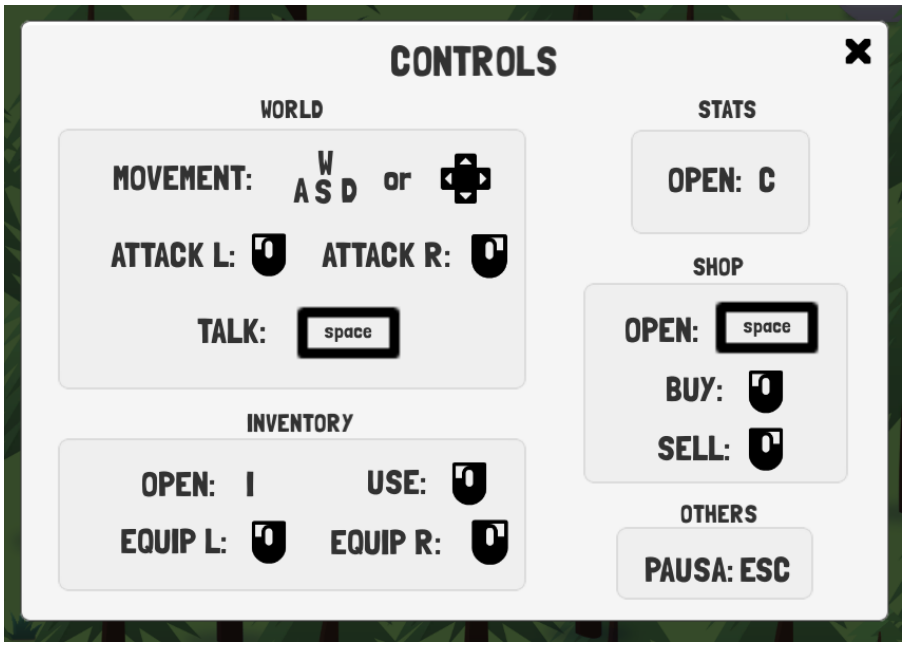


**Ilustración 95. Panel de controles.**

A continuación, se explicarán los controles de juego de Guardians of the Forest.

- **Mundo**.

    - **Movimiento**: Utilizar las teclas WASD o las flechas de movimiento.
    - **Atacar con mano izquierda**: Clic izquierdo del ratón en el mundo.
    - **Atacar con mano derecha**: Clic derecho del ratón en el mundo.
    - **Hablar**. Barra espaciadora.
- **Estadísticas**.
    - **Mostrar / ocultar**: Tecla C.
- **Inventario**.
    - **Mostrar / ocultar**: Tecla I.
    - **Usar consumible**: Clic izquierdo del ratón sobre el ítem.
    - **Equipar arma en mano izquierda**: Clic izquierdo del ratón sobre el ítem.
    - **Equipar arma en mano derecha**: Clic derecho del ratón sobre el ítem.
- **Tienda**.
    - **Abrir**. Barra espaciadora.
    - **Comprar**. Clic izquierdo del ratón sobre el ítem en la tienda.
    - **Vender**. Clic derecho del ratón sobre el ítem en el inventario (la tienda debe estar abierta).
- **Otros controles.**
    - **Pausa**. Escape.

# 8. Conclusiones

## 8.1 Conocimientos adquiridos

Con la realización de este trabajo he adquirido un montón de conocimiento nuevo, así como profundizar aun más en aquellos que ya tenía:

- He aprendido más acerca de la historia de los RPG y cómo se realizaron estos primeros títulos, así como conocimientos sobre la vertiente de juegos de rol de mesa en la que se inspiraron.

- He aumentado mis conocimientos sobre el motor Unity, cómo la utilización de múltiples grids, con diferentes tamaños y subcapas, para tener diferentes layers de detalle; la poderosa utilidad que presentan los scriptables, así como su uso y definición; la adaptación de componentes pensados para 3D y su aplicación en el mundo 2D; la carga de recursos al vuelo desde código, facilitando la externalización de elementos como diálogos, lo que permitiría una fácil internalización del juego; el perfeccionamiento de uso de corutinas para crear eventos de juego, como pueden ser las DoT o la transformación de los guardianes; etc.

- He mejorado mi algoritmia y programación, creando interfaces, y refactorizando y modularizando comportamientos en scripts, como por ejemplo en StatsController, permitiendo la centralización de lógicas comunes para diferentes tipos de instancias que tienen una base común.

- He mejorado mi capacidad para desarrollar inteligencias artificiales, perfeccionando máquinas de estados que puedan recibir información no solo de eventos del exterior, si no también de eventos propios de la instancia, como puede ser un frame de animación, o cambios de estados que dependen de una probabilidad en el tiempo y de ciertos parámetros, como ocurre por ejemplo con el estado HealState; así como la combinación de diferentes estrategias de IA para obtener resultado mejores, cómo la combinación del A* del Navigation con mi máquina de estados.

- He aumentado mis conocimientos sobre rendimientos, controlando los cálculos de renderizado del inventario, o el cálculo de *autosorting* realizado en los scripts AutoSortOrder y AutoSortOrder2.

- He mejorado la sincronización de animaciones y acciones en el juego, sincronizándolas con el uso de eventos en frame.

## 8.2 Objetivos planteados

Se han conseguido todos los objetivos principales planteados para este trabajo de forma satisfactoria.

- Administración de recursos dentro del juego. Se ha conseguido ofrecer al usuario la necesidad de organizar sus recursos para obtener el equipamiento y armamento deseado, pero no sólo en cuanto a qué ítems querer usar y llevar, sino también a administrar sus valores, estadísticas, y pasivas, así como la forma de combinarlos, para poder obtener el máximo rendimiento en su guardián.

- Mecánicas de uso diferentes en cada una de las armas y objetos. Se realizó un total de 31 ítems diferentes, entre los que encontramos objetos equipables, consumibles, pasivos, y estadísticos, cada uno de ellos con características únicas y formas de interacción diferentes.

- Estancias autogeneradas y pseudoaleatorias. Se consiguió realizar un bosque que se autogenera con diferentes tipos de elementos, desde enemigos a elementos sólidos a decorativos, permitiendo la experiencia de vagar por un bosque infinito dónde cada estancia es diferente a la anterior y donde nunca habrá dos partidas iguales.

- Diferentes tipos de enemigos y jefes. Se creó un total de cuatro enemigos distintos y con habilidades diferentes, entre los que tenemos a los segadores, enemigos físicos cuerpo a cuerpo; los golems, enemigos mágicos a distancia; los ancestrales, jefes robustos con poderes de curación y daño cuerpo a cuerpo; y los jinetes, jefes poderosos híbridos que hacen daño mágico a distancia, y daño físico cuerpo a cuerpo.

- Dificultad creciente del juego. Se ha definido un algoritmo que aumenta la dificultad según el nivel en el que te encuentres, mejorando las estadísticas de los enemigos de forma controlada, así como el poder de sus habilidades.

## 8.3 Planificación y metodología

Tanto la planificación cómo la metodología de desarrollo iterativo e incremental han sido seguidas a la perfección, lo cual se puede comprobar con el sistema de control de versiones utilizado, GitHub, dónde se iban comiteando las tareas en el orden y tiempo planificado.

Si bien, algunas tareas tomaron más tiempo de lo pensado, cómo pudo ser la definición y uso de cada uno de los objetos y armas, luego se recuperó ese tiempo completando más rápido otros hitos.

El único hito que no se pudo terminar fue el de la incorporación de joysticks al juego, debido a que no se tuvo en cuenta la realización de la memoria como parte del proyecto planificado. Sin embargo, este hito se había decido dejar para el final, pues era un punto opcional que no afectaba directamente al trabajo que se esperaba entregar, es decir, se trataba de uno de los puntos extra para aumentar la experiencia de los usuarios.

## 8.4 Líneas de trabajo futuro

Se presentan las siguientes líneas de trabajo futuro para este videojuego:

- Incorporación de joysticks. Era uno de los puntos que se esperaba hacer para este trabajo, el cuál ha sido desplazado a trabajo futuro por causas ya explicadas anteriormente.

- Competición en línea. Se desea establecer un sistema de competición en línea, dónde al final de cada partida el jugador pueda subir su puntuación a un servidor, y exista una clasificación dónde muestre el top de usuarios que ha alcanzado niveles más altos, y con que *build* o composición de equipamiento consiguió llegar.

- Modo cooperativo. Se desea introducir un modo cooperativo en línea de hasta dos jugadores, dónde los guardianes puedan superar el bosque juntos, permitiendo una mayor gestión de recursos, pues podrían comprarse equipamientos complementarios y coordinarse a la hora de pelear.

- Nuevos objetos. Se quiere realizar nuevos tipos de objetos como los artilugios, objetos de utilidad que podrían ser desde trampas para enemigos, cómo recetas con elementos que podrían soltar los enemigos al morir.

# 9. Glosario

A continuación, se definen diversos términos que han sido usado durante el desarrollo de la memoria:

- **GameObject**. Es el objeto base en Unity. Por sí solos no hacen nada, necesitan de componentes para darles funcionalidad.
- **Sleep**. Término en inglés que significa dormir.
- **Game Jams**. Evento de desarrolladores de videojuegos en los que se debe realizar un videojuego acorde a una temática en un tiempo límite.
- **C++.** Lenguaje de programación creado en 1979.Diseñado por Bjarne Stroustrup.
- **C#**. Lenguaje de programación creado en 2000. Diseñado por Microsoft.
- **Rigidbody**. Componente de físicas. Permite al GameObject ser afectado por las físicas.
- **Cinemachine.** Componente de Unity para facilitar el control y movimiento de cámaras.
- **UNet**. Unity Networking. Librería para soporte de funcionalidades en línea y multijugador.
- **HLAPI**. High Level API de multijugador en Unity.
- **Game Engine**. Término en inglés que significa motor de videojuego.
- **Sprites**. Mapa de bits que se dibuja en pantalla, o seria de imágenes en un mismo archivo.
- **UI.** Interfaz de usuario.
- **Git**. Sistema de control de versiones open source.
- **Asset**. Ítem que puede ser utilizado en un videojuego.
- **Prefabs**. Objeto reutilizable en Unity. Se crea a partir de un GameObject cualquiera.
- **Tilemaps**. Mapa de mosaicos que se utiliza en el diseño de niveles de videojuegos 2D.
- **Spawn .** Término en inglés que significa aparecer.
- **Bundle.** Paquete de assets.
- **Layer**. Término en inglés que significa capa.
- **Powerup.** Término en inglés que significa mejorar o potenciar.
- **In-game**. Término en inglés para hacer referencia a estar dentro de la partida.
- **Background**. Término en inglés para hacer referencia al fondo.
- **Source.** Término en inglés para hacer referencia a la fuente origen de algo.
- **Arrays.** Vector de elementos.
- **Collider.** Componente de Unity que define un área con la que se pueden detectar colisiones.
- **Feedback.** Termino en inglés que significa retroalimentación.
- **Tester.** Persona que prueba el funcionamiento de un videojuego.
- **User-friendly**. Expresión inglesa que hace referencia a algo que es fácil de usar por el usuario.

# 10. Bibliografía


1. Agile methodology — Zomato Case study - Bhavz Kakarla https://medium.com/@srisayi.bhavani/agile-methodology-zomato-case-study-311da3388518 (10/04/2020)

2. Historia de los videojuegos: Los orígenes del género RPG http://www.destinorpg.es/2015/08/historia-de-los-videojuegos-los.html (12/04/2020)

3. El origen y evolución de los juegos RPG https://steemit.com/spanish/@gabox/la-historia-y-evolucion-de-los-juegos-rpg-historia-de-los-videojuegos-97-retrospectiva (12/04/2020)

4. The Game of Dungeons/dnd (1975) http://crpgaddict.blogspot.com/2012/02/game-69-game-of-dungeonsdnd-1975.html (12/04/2020)

5. PLATO: El padre de las comunidades online nacido antes de Internet https://parceladigital.com/2018/03/05/ordenador-plato (12/04/2020)

6. Programación en Unity 2D. Introducción a los videojuegos. Pierre Bourdin Kreitz, Jordi Duch Gavaldà, Heliodo Tejedor Navarro. PID_00236757.

7. What's wrong with the aspect ratio in standalone build? https://answers.unity.com/questions/1362360/whats-wrong-with-the-aspect-ratio-in-standalone-bu.html (26/05/2020)

8. Enforce Aspect Ratio of Game Window standalone for Mac OSx https://forum.unity.com/threads/solved-enforce-aspect-ratio-of-game-window-standalone-for-mac-osx.458347/ (26/05/2020)

9. Standalone Mac build ignores aspect ratio in MacBookPro 15 retina https://answers.unity.com/questions/1451234/standalone-mac-build-ignores-aspect-ratio-in-macbo.html (26/05/2020)