Universitat Oberta de Catalunya

uoc.edu

The Crypt of the Other Gods

Roberto González Suárez
Máster en Diseño y Desarrollo de Videojuegos
Trabajo Final de Máster

Joan Arnedo Moreno
Jordi Duch Gavaldà
03/01/21

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Copyright © AÑO TU-NOMBRE.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free

Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

C) Copyright

© (el autor/a)
Reservados todos los derechos. Está prohibido la reproducción total o parcial de esta obra por cualquier medio o procedimiento, comprendidos la impresión, la reprografía, el microfilme, el tratamiento informático o cualquier otro sistema, así como la distribución de ejemplares mediante alquiler y préstamo, sin la autorización escrita del autor o de los límites que autorice la Ley de Propiedad Intelectual.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*TFM: Desarrollo del juego The Crypt of the Other Gods*
Nombre del autor:	*Roberto González Suárez*
Nombre del consultor/a:	*Jordi Duch Gavaldà*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	01/2021
Titulación::	*Master de Diseño y Programación de Videojuegos*
Área del Trabajo Final:	*Trabajo Final de Máster*
Idioma del trabajo:	Español
Palabras clave	*Videojuego, roguelike, 3D*

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

El trabajo cubre la explicación, metodología y arquitectura del desarrollo de un videojuego en vista isométrica con mecánicas de shooter rápido. El juego desarrollado, **The Crypt of the Other Gods**, basa su jugabilidad en un movimiento constante y el uso de distintos tipos de armas que el jugador debe desbloquear a base de eliminar los enemigos que encuentra a su paso. Con la capacidad de generar escenarios de manera aleatoria, permite una rejugabilidad elevada ya que los mapas siempre serán distintos para el jugador.

Durante la planificación, investigación y desarrollo del proyecto, se han implementado algunas mecánicas, de manera que el desarrollo del proyecto ha sido orgánico, y se ha logrado realizar un prototipo funcional del juego, por lo que el producto puede considerarse un producto mínimo viable.

El resultado final es un juego con las mecánicas base definidas al que se puede agregar de manera sencilla material como niveles, enemigos, de manera que casi es un “framework” para el desarrollo del propio juego.

Abstract (in English, 250 words or less):

The work covers the explanation, methodology and architecture of the development of a video game in isometric view with fast shooter mechanics. The game developed, **The Crypt of the Other Gods**, bases its gameplay on constant movement and the use of different types of weapons that the player must unlock by eliminating the enemies that he encounters in his path. With the ability to generate scenarios randomly, it allows high replayability since the maps will always be different for the player.

During the planning, research and development of the project, some mechanics have been implemented, so that the development of the project has been organic, and a functional prototype of the game has been made, so the product can be considered a minimum viable product.

The end result is a game with defined basic mechanics to which material such as levels, enemies can be added in a simple way, so that it is almost a “framework” for the development of the game itself.

Agradecimientos

Agradecer a mi familia por la ayuda y soporte durante el desarrollo de mi carrera hasta este punto. A mi pareja, Eliana, por ser el mayor pilar y apoyo durante los años de estudio compaginado con una vida laboral, y ser siempre tan comprensiva, sin ella no lo podría haber logrado.

A mis amigos Manuel, Albert y Julio por prestar sus voces como añadidos e ideas durante los tests finales.

Índice

1. Introducción **1**

1.1 Contexto y justificación del Trabajo 1

1.2 Objetivos del Trabajo 1

1.3 Enfoque y método seguido 2

1.4 Planificación del Trabajo 3

1.5 Breve sumario de productos obtenidos 5

2. Estado del arte **6**

2.1 El género 6

2.2 Tecnología 10

3. Definición del juego **12**

3.1 Historia y ambientación 12

3.2 Definición de personajes y enemigos 14

3.3 Definición de armas 19

Balas 19

Plasma 22

Láser 25

Cohetes 26

3.4 Arte del juego 27

4. Diseño técnico **28**

4.1 Entorno técnico 28

4.2 Arquitectura 32

4.2.1 Arquitectura general 32

4.2.2 Armas 33

4.2.2.1 AbstractWeapon 34

4.2.2.2 LineWeapon 36

4.2.2.3 LaserWeapon 36

4.2.2.4 ElectricWeapon 37

4.2.2.5 BulletWeapon 38

4.2.3 Inventario 40

4.2.4 Sistema de compra 41

4.2.5 Integración 2D/3D 42

4.2.6 Generador de mapas 45

4.2.6.1 Postprocesado del nivel 48

4.2.6.1.1 Conexiones entre habitaciones 48

4.2.6.1.2 Colocación de llaves 49

4.2.6.1.3 Limpieza de paredes 50
4.2.7 Control de arenas 50
4.2.7 Inteligencia artificial 52
4.2.7.1 Armamento enemigo 54
4.2.7.2 NavMeshAgent 54
4.2 Utilización de Assets 56
4.2.1 Assets 3D 56
4.2.1 Assets 2D 57
4.2.2 Assets Audios 60

5. Diseño de niveles 61

6. Manual de usuario 65
6.1 Requerimientos técnicos del hardware 65
6.1 Instrucciones del juego 65

7. Conclusiones 66
7.1 Lecciones aprendidas 66
7.2 Objetivos y reflexión 67
7.3 Análisis de la planificación 67
7.4 Futuras líneas de trabajo 68

8. Glosario 70

9. Bibliografía 71

Lista de figuras

Figura 1: Base de la metodología Scrum 2
Figura 2: How Roguelike is your game 7
Figura 3: Imagen de Rogue 8
Figura 4: The Binding of Isaac 9
Figura 5: Enter the Gungeon 9
Figura 6: Ejemplo de Rogue-lite 2D clásico 10
Figura 7: Bullets per minute 11
Figura 8: Void Bastards 11
Figura 9: Nostromo, de Alien 12
Figura 10: Dead Space/Doom 2016 13
Figura 11: El protagonista 14
Figura 12: Zombie 15
Figura 13: Huella 15
Figura 14: Verdugo eléctrico 16
Figura 15: Soldado sin nombre 17
Figura 16: Jefe de nivel 18
Figura 17: Rifle 20
Figura 18: Ametralladora 21
Figura 19: Escopeta 22
Figura 20: Rifle de plasma 23
Figura 21: Ametralladora de plasma 23
Figura 22: Escopeta de plasma 23
Figura 23: Granada de plasma 24
Figura 24: Rifle Láser 25
Figura 25: Cañón eléctrico 26
Figura 26: Granadas 26
Figura 27: Lanzacohetes 27
Figura 28: Comparativa LWRP / HDRP 29
Figura 29: Comparativa de los RenderPipelines en un escenario 30
Figura 30: Ejemplo de postprocesado en el escenario del juego 31
Figura 31: arquitectura general 32
Figura 32: Barra de recarga 35
Figura 33: escopeta láser desechada 37
Figura 34: Sistema de compra 41
Figura 35: Menú de compra 42
Figura 36: Primeras aproximaciones a 2D/3D 43
Figura 38: Material 2D 43
Figura 37: Script de cambio de material en animación 44
Figura 39: Flow de LevelBuilder 45
Figura 40: Nivel generado aleatoriamente 46
Figura 41: Salas 47
Figura 42: Ejemplos de mapas generados 48
Figura 43: Ejemplo de distribución de mapa 49
Figura 44: Colocación de llave 49
Figura 45: Flow de control de habitación 51
Figura 46: Configuración de enemigos 52
Figura 47: LocalNavMeshBuilder 55
Figura 48: áreas navegables por los agentes 56
Figura 49: Diseño de personaje 58
Figura 50: Spritesheet plantilla para movimiento 58

Figura 51: Proceso de dibujado sobre la animación 59
Figura 52: spritesheet resultante 59
Figura 53: Orientaciones del personaje 59
Figura 54: Spritesheet de enemigo corriendo 60
Figura 55: Pasillos 61
Figura 56: Sala 62
Figura 57: Zona tóxica 62
Figura 58: Oficina 63
Figura 59: Enfermería 63
Figura 60: Sala final 64
Figura 61: Icono del juego 65

1. Introducción

1.1 Contexto y justificación del Trabajo

El mercado de los videojuegos ha sufrido un aumento considerable durante la última década llegando a vender 8,4 millones de videojuegos y 1,1 millones de consolas tan solo en 2019 [1].

Además de que el número de usuarios ha aumentado su cifra de 15 millones de videojugadores. Dentro de este aumento del mercado, un sector que ha encontrado su hueco y a un considerable nicho de mercado son los llamados videojuegos “indie”. Los juegos “indie” son juegos “pequeños” gestionados y producidos por autores con propio capital o donaciones de otros usuarios (como sistemas de crowdfunding), pero con un fuerte factor de innovación o una personalidad única (con un arte gráfico muy diferenciado del resto por ejemplo).

Con la principal idea de desarrollar un videojuego para demostrar los conocimientos adquiridos a lo largo del Máster de Diseño y Programación de Videojuegos. Tras valorar distintos géneros e ideas de videojuegos se decidió retomar la idea original del trabajo final de la asignatura 2D. Este trabajo final estaba basado en la generación de un juego del ámbito de “rogue-lite”. Con la idea de actualizar las mecánicas y adaptar el nuevo juego a un entorno 3D a modo de “remake”, la idea de como evolucionar un juego con todo lo aprendido y llevarlo a una nueva grado de calidad, es atractiva.

1.2 Objetivos del Trabajo

Los objetivos generales del trabajo fueron los siguientes:

1. Adecuada planificación y gestión de un proyecto de videojuegos.
2. Investigación y estudio de la plataforma Unity
3. Diseño de un videojuego
4. Juego base que engloba todo lo comprometido.

1.3 Enfoque y método seguido

Para el desarrollo del juego se ha generado una planificación inicial en tiempos donde se ha dividido las funcionalidades en tiempos divididos por “sprints”. Ya que se ha usado un sistema de metodología ágil, concretamente la metodología SCRUM.

Las metodologías ágiles se basan en un continuo desarrollo donde al final de cada “sprint” (un tiempo predefinido de desarrollo) debe de haber una evolución del producto, de manera que en cada sprint vemos un avance del producto.

La principal ventaja de este sistema es la capacidad de obtener en cada iteración una visión del producto y visualizar el incremento de valor realizado. Tras cada sprint se realiza una review del incremento realizado, y así se puede rectificar en la planificación o funcionalidad. Tras esta review del producto se generan nuevas tareas en caso de ser necesario y se planifica el siguiente sprint en base de las tareas planificadas al inicio del proyecto.

La metodología ágil SCRUM es de especial utilidad para el desarrollo de software con equipos. En este caso al ser un proyecto desarrollado por una persona se han eliminado del proceso las denominadas “ceremonias” (reuniones de seguimiento, retrospectivas…) y se ha relajado mucho el formato. Aún así la metodología ha servido para poder dividir el trabajo en paquetes entregables donde se puede visualizar el avance del desarrollo y poder adaptarse en tiempos de desarrollo.

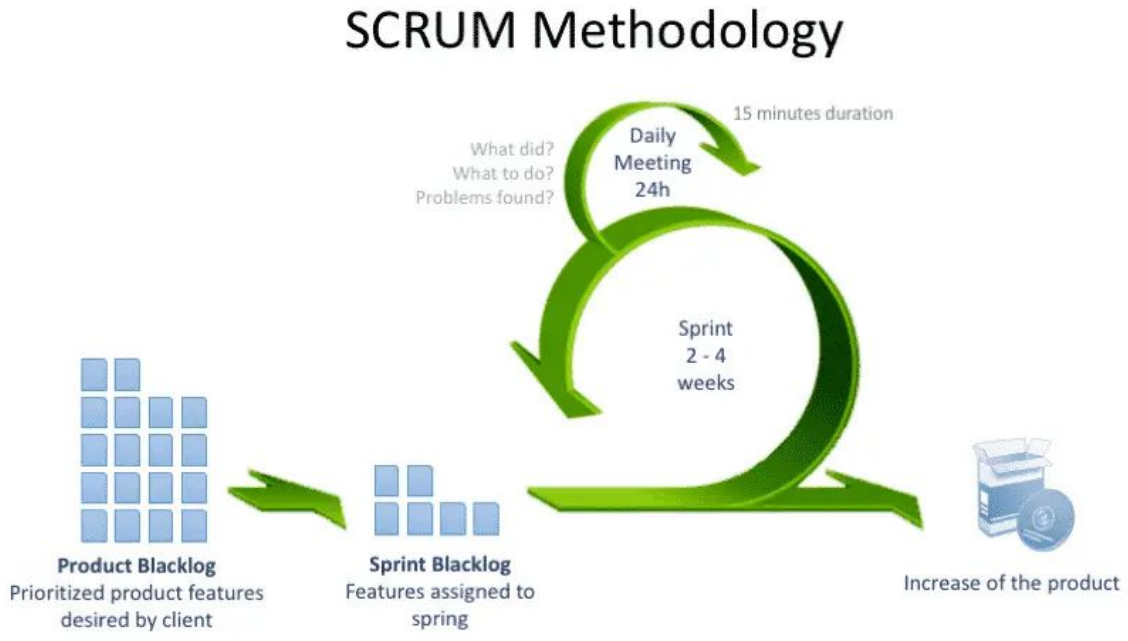

Figura 1: Base de la metodología Scrum [2]

1.4 Planificación del Trabajo

Para la primera entrega del Trabajo se dispuso de una planificación inicial que, como en todo desarrollo ágil, se vió modificada durante el propio desarrollo del juego. Debido al formato de entregas, se adaptó una estimación para poder generar los entregables con versiones jugables del producto en dichos momentos.

Los entregables fueron:

- PEC1: Diseño del videojuego. (inicio Octubre) Para esta entrega no se necesita el juego en fase de desarrollo, si no que se debe generar la documentación inicial y la planificación comentada anteriormente.
- PEC2: Versión parcial (Inicio de Noviembre) Esta entrega define como requisito una versión alpha del producto donde se puedan visualizar las mecánicas del juego y obtener una idea del prototipo para poder redefinir las mecánicas o realizar correcciones en caso necesario.
- PEC3: Versión jugable (Inicio Diciembre) La última entrega antes de la versión final. El requisito de esta versión entregable es ser una versión beta muy cercana a la versión final del juego.
- PEC Final. (Inicio Enero). Versión final del juego con la memoria y video de demostración.

Por ello, y en vista de que los entregables se dividen en tramos de tiempo de 1 mes, se genera un diagrama de planificación basado en tramos de 1 semana, por lo que los “sprints” en los que se divide el trabajo se divide en estos mismos tiempos.

Tarea	Octubre				Noviembre				Diciembre			
	Versión alpha				Versión jugable				Versión Gold			
Menús												
Control y movimiento jugador												
Comportamiento básico enemigos												
Comportamiento armas												
Objetos/Puertas												
Mapa 1												
Sistema de compra/mejora												
Desarrollo variante enemigos												
Mapa 2												
Mapa 3												
VFX / retoques gráficos												
Audio/ Música												
UI final												
Revisión final												

Como se puede apreciar en la planificación inicial los 3 entregables se dividieron en los 3 entregables. Cada evolutivo es un incremental de la versión anterior donde se en cada entrega se centra en un objetivo primordial.

- Entrega 1 (Alpha): Las funcionalidades programadas para la primera versión se centran en: menú, control de jugador y enemigos básicos. Con esta entrega se genera un mapa “testroom” donde se muestran las mecánicas de movimiento y disparo base. Además de poder poner a disposición del usuario las armas sin el sistema de recompensas definido.
- Entrega 2 (Beta): Versión jugable casi completa, integra el sistema de mejoras y desbloqueos. El usuario podrá jugar 2 mapas completos, pero sin efectos visuales o UI.
- Entrega 3 (Gold): Versión completa del juego donde se ha integrado la UI.

Durante el desarrollo del juego se produjeron algunos cambios de alcance sobre el desarrollo, que derivó a cambios en la planificación inicial.
Los cambios realizados fueron los siguientes:

- La realización de los mapas se cambia por un sistema de generación de mapas de manera aleatoria.
- Ciertas armas se modifican debido a pruebas de jugabilidad realizadas durante la fase beta.
- Se adelanta la inclusión de VFX y post procesados para poder integrar la generación de mapas. Este cambio era necesario para poder visualizar una versión cercana al entregable Gold, debido a la importancia de la calidad visual del producto y al uso del HDRP (el Render Pipeline de Unity) en una versión actualizada. Esta última razón exigía verificar el uso de cada Asset de Unity o cada efecto y su compatibilidad, ya que Unity permite utilizar distintos Render y no todos los Assets son siempre compatibles.

1.5 Breve sumario de productos obtenidos

El producto final es un juego en un estado “Early Access”, donde el jugador podrá jugar con el sistema de generación de niveles de manera que pueda ser reutilizable, y debido al sistema de desbloqueo y autoguardado (entre partidas), permite guardar el avance del jugador, de manera que los desbloqueos se almacenan en una memoria permanente y permite jugar varias sesiones por separado.

A continuación se describen los capítulos que componen el resto del documento.

Estado del arte
Se realiza un estudio del género y la situación de éste en la actualidad. Además de un análisis de la tecnología utilizada y el porqué.

Definición del juego
Desarrollo de las mecánicas del juego, diseño artístico, influencias y conceptos.

Diseño técnico
Definición del aspecto técnico del juego, explicación de todos los procesos, arquitecturas y componentes del juego.

Diseño de niveles
Definición de las habitaciones y las decisiones tomadas sobre el generador de niveles.

Manual de Usuario
Requerimientos mínimos y guía para el usuario.

Conclusiones
Conclusiones obtenidas del desarrollo y sobre los problemas encontrados (y cómo se han solucionado) durante el proceso.

2. Estado del arte

2.1 El género

El género del juego se puede situar dentro del “roguelite”, que se considera un subgénero del “roguelike”. Para poder concretar un poco tenemos que detallar las definiciones de cada género:

Roguelike es un subgénero del juego de rol caracterizado por el avance a través de mazmorras generadas de manera procedural. Entre otras características de este género se encuentra la muerte permanente del personaje.

El Rogue-lite por su parte es un género que coge tan sólo partes del Roguelike, permitiendo un sistema menos “hardcore”.

Sin embargo, la cantidad de definiciones de este género y los numerosos intentos de delimitar qué defino a un juego como roguelike o roguelite, tenemos otras variantes del género como el roguelikelike, según una serie de propiedades que tenga el juego[3].

Según la herramienta “How Roguelike is your game” la división de géneros se dividen por la agrupación de las siguientes propiedades: uso de ASCII, eventos narrativos, habilidades disponibles desde el inicio, uso de sólo teclado, eventos aleatorios, historia mínima, uso de una mazmorra, mundo basado en cuadrícula, muerte permanente, uso de turnos, juego táctico, hack & slash, los atributos del jugador se muestran y las referencias a “Rogue”.

Siguiendo estas referencias podemos determinar que nuestro juego cuenta con la categoría “roquelikelike”.

Determine your Roguelike rating.

- [ ] ASCII
- [ ] Most events are narrated
- [x] All user actions are available from the start
- [x] Keyboard only
- [x] Random environments
- [x] Minimum story
- [x] Has a dungeon
- [x] Grid-based world
- [ ] Permadeath
- [ ] Turn-based
- [ ] Tactical
- [x] Hacking and slashing
- [ ] Player attributes are shown
- [ ] Named "Rogue"

Your game is a Roguelikelike

Huh?

Figura 2: How Roguelike is your game

El juego que nos ocupa, “The Crypt of the Other Gods”, contiene parte de los elementos, por lo que como se comenta se puede englobar dentro de este subgénero, pero siempre estableciendo como base el género “shooter”. Evitando las mecánicas de turnos para el uso de movimientos y quitando el elemento táctico (al menos de manera concienzuda).

Las características del juego para basarse en poder catalogarlo como parte “rogue-lite” es el uso de experiencia para poder realizar desbloqueos de armas, una de las principales mecánicas del juego. Sin embargo, como se indica, la subcategorización es complicada.

La generación de un escenario de manera procedural también apoya la categorización del juego dentro de dicho género. De hecho esta es una de las características base del género, ya que se considera crucial para que un juego sea considerado “rogue-lite”.

El género del roguelike/lite ha revivido en los últimos años, con una gran fuerza, encontrando su nicho en el mercado “indie”.
El primer juego considerado de tal género es el “Rogue”[4],creado para ordenador en 1980, dando nombre a dicho género. Su premisa es introducir al jugador en la piel de un aventurero que debe superar una mazmorra para recuperar un amuleto. Su principal característica es la generación aleatoria de la mazmorra, algo inaudito hasta el momento.

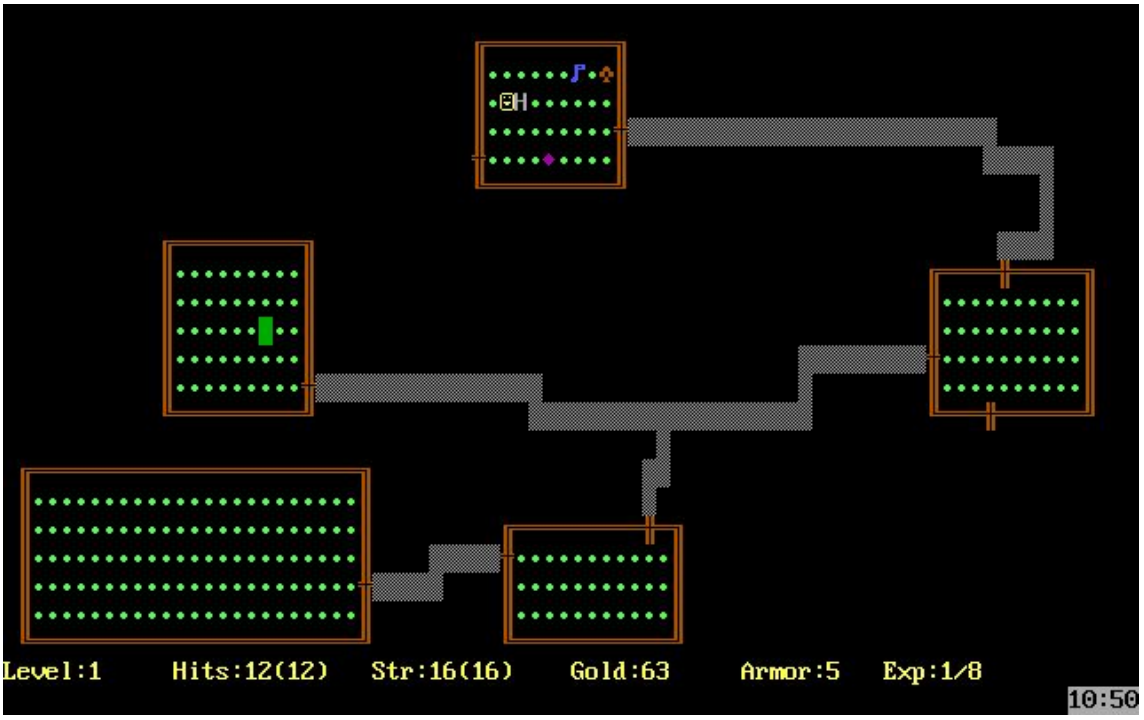

Figura 3: Imagen de Rogue [4]

pese a tener algunos juegos del género en consolas de 16 bits, y algunos juegos destacables (como la saga Diablo) el verdadero renacer del género se produce posteriormente sobre 2010.

Los principales motivos de la popularización del género son:

- El uso de generación de niveles procedurales que permiten rejugabilidad infinita.
- La dificultad asociada a este tipo de juegos. Los juegos “roguelike” suelen estar asociados a una dificultad elevada y suponer un reto al jugador.
- El uso de aleatoriedad en el equipo y distribución, permiten además aumentar la rejugabilidad, permitiendo jugar de distintas maneras cada vez (generando las llamadas “runs”).

Uno de los primeros roguelikes que revitalizaron el género fue **The Binding of Isaac**[5], un hyegi desarrollado por Edmun McMillen y Himsl Florian que se estrena en Septiembre de 2011. Con un sistema de muerte permanente, mazmorras aleatorias y compra de objetos que mejoran al personaje, el jugador debe atravesar una serie de capítulos para salir vivo.

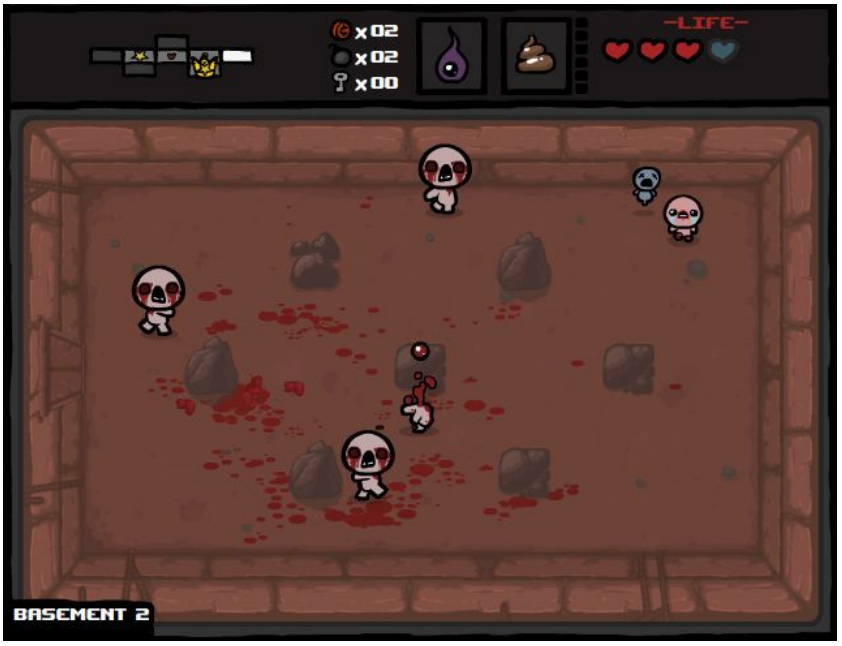

Figura 4: The binding of Isaac

Nuclear Throne[6] salió al mercado en 2015 y basa su jugabilidad en la habilidad del personaje en poder adquirir mutaciones para evolucionar. Estas evoluciones proporcionan habilidades únicas que se suman a un gran arsenal dentro del juego.

Otro juego famoso del género es **Enter the Gungeon**[7], de 2016. Aunque más categorizado como “juego de disparos” o “shooter”. Sus mecánicas son muy similares a The Binding of Isaac o Nuclear Throne, basándose en un extenso uso de armamento para poder avanzar en el juego.

Figura 5: Enter the Gungeon

Además de estos juegos, muchos otros han sido reconocidos dentro del género como el **Spelunky**[8] (en los inicios del resurgir) o **Hades**[9] como uno de los últimos juegos del género que ha tenido una gran acogida por el público.

Además de estos juegos podemos nombrar importantes aportaciones que tienden a mezclar géneros, lo que conlleva a afianzar que el género del roguelike en sí puede combinarse con otros y ser fluido. Entre varios ejemplos, tenemos juegos como **Dead Cells**[10], un juego que mezcla el género con el llamado "Castlevania" (llevándolo a llamar RogueVania), **Crypt of NecroDancer**[11], que mezcla el uso de los turnos con el género musical o Darkest Dungeon, que se basa en centrarse en un sistema RPG puro con una dificultad elevada.

2.2 Tecnología

Para el desarrollo del juego se ha utilizado como base el motor de Unity, en su versión 2019.3.5f1. Aunque las versiones de desarrollo utilizadas durante todo el master siempre han sido en LTS de 2018, cuando se planteó el inicio del proyecto se valoraron el uso de los distintos Render Pipelines de Unity para la generación del juego.

El principal motivo de la elección del motor es el conocimiento adquirido durante el master, frente a otros motores como GameMaker o Unreal.

Aunque habitualmente los rogue-lites se basan en uso de motores 2D (grandes ejemplos son el Binding of Isaac o Enter the Gungeon), la búsqueda de generar un ambiente visual llamativo era complicado usando el 2D (aunque Unity ha mejorado en sus últimas versiones respecto a la iluminación 2D), ya que dependería en su mayor parte de un arte muy cuidado y hecho expresamente para el juego. Por lo que el uso de un motor 3D permitiría hacer uso de Assets que aprovecharán de mejor manera el sistema de iluminación de alta definición, además de potenciar la calidad gráfica del juego.

Figura 6: Ejemplo de Rogue-lite 2D clásico

Aunque en la búsqueda de juegos Roguelite bajo entornos 3D en el mercado general ha sido infructuosa, si se pueden encontrar ejemplos bajo desarrollos de indies o pequeños proyectos.

La mayoría de Rogue-lites (o rogue-like) 3D que podemos encontrar en el mercado se combinan con el género de los FPS como puede ser “Bullets per Minute[12]” o “Void Bastards[13]” e incluyen un aspecto visual característico para ayudarse en la ambientación.

Figura 7: Bullets per minute juega con la distorsión del color

Figura 8: Void bastards usa un efecto de comic, con enemigos 2D en un entorno 3D

Este género nació en la plataforma de PC, donde ha tenido su mismo rejuvenecimiento en los juegos Indie. Aunque gracias a la extensión de los juegos indies en las consolas, podemos ver muchos juegos en plataformas como Playstation o especialmente en Switch, donde se ha generado un buen mercado. Sin embargo la facilidad de desarrollo y distribución en la plataforma de PC, sigue haciendo que ésta sea la plataforma fuerte para este tipo de género.

3. Definición del juego

3.1 Historia y ambientación

El juego está ambientado en un futuro oscuro donde el jugador está encerrado en una nave espacial a oscuras. Con la intención de generar un ambiente oscuro y opresor, se ha escogido una serie de elementos que conforman el mapa para generar entornos fríos y hostiles.

Una de las principales fuentes de inspiración ha sido Alien, donde el uso de pasillos estrechos, juegos de luces y sombras generan una atmósfera inquietante.

Figura 9: Imágenes de la nave Nostromo, donde sucede Alien

Si pudiéramos sacar mas referencias o ambientaciones en las que se ha basado el juego para intentar reproducir la ambientación buscada, nos podemos mover al medio de los videojuegos, donde la principal referencia en este caso es Dead Space.

En Dead Space el jugador se enfrenta a una mutación que ha transformado a la tripulación de una nave en monstruos llamados necromorfos. Con unas mecánicas innovadoras en su momento (y un uso de la interfaz nunca visto), consigue introducir al jugador en una ambientación donde el miedo y los sustos son la base principal.

Una vez definida la ambientación y tono buscado para los entornos, el ritmo de juego buscado era algo más rápido e inmediato. En este caso para la acción y jugabilidad la principal referencia es DOOM, el shooter por antonomasia, que siempre se ha caracterizado por la rapidez, también desprende en sus últimas entregas ambientes demoníacos y oscuros, por lo que no desentona en el tono buscado.

Figura 10: Dead Space y Doom 2016 son claros ejemplos de entornos oscuros scifi

Para la realización de la historia se ha elaborado un texto introductorio, como comúnmente se hacía en los manuales de los juegos, como en el caso de DOOM.

El texto se expone a continuación:

"Eres R. Carter, parte del equipo de rescate Ark. Hace tan solo unas horas tu equipo y tú han captado una señal de auxilio muy débil mientras volvían de una misión con éxito. Sin dudarlo, deciden ir directamente al origen de la señal.

Mientras se realizan los preparativos para acoplarse a la nave, descubren que es la Arkham, una nave que sirve para la exploración y como laboratorio. Sus comunicaciones se han cortado y los registros indican que volvían de recoger una serie de artefactos alienígenas en un planeta remoto.

Las puertas se abren y el equipo entra en las oscuras salas con sus linternas acopladas a sus rifles, pero rápidamente son asaltados por los integrantes del Arkham y mueren en el acto.
Tan solo tú consigues escapar de vuelta a la nave de rescate, pero descubres que la bahía de acople ha quedado destrozada, y tu vía de escape está al otro lado de la estación.

Tras meditarlo mucho y ver los horrores que deambulan por la nave, llegas a la conclusión de que tu única oportunidad es atravesar la nave para poder llegar a las cápsulas de escape, no sin antes intentar detonar el reactor principal y evitar que los demonios que albergan la Arkham puedan llegar a algún planeta habitado.

Cargas tu rifle y entras en los oscuros pasillos..."

Por lo tanto, el jugador deberá sortear una serie de mapas eliminando enemigos con el objetivo de poder escapar de la nave.

Como se puede observar en el nombre de la nave principal, otra ambientación usada de manera muy sutil son los relatos de Lovecraft. De manera que hay continuas referencias a la denominada saga de los mitos de Cthulhu.

3.2 Definición de personajes y enemigos

El personaje principal es R. Carter, sin nombre definido, debido al intento de no asociar un personaje o personalidad directamente (como el efecto generado en muchos juegos donde se evita la asociación de una personalidad directa, como el mencionado DOOM o Half-life). Sin embargo el nombre es una referencia a Randolph Carter, un personaje recurrente en las obras de Lovecraft.

Figura 11: Randolph Carter, el protagonista

Armado con un rifle inicialmente y la armadura de combate debe superar los niveles que le separan de la vía de escape de la Miskatonic.

Para el diseño del protagonista se buscaba un diseño de armadura con elementos fácilmente reconocibles, por ello se le agregó partes rojas, y así facilitar la localización dentro de la pantalla (aunque siempre estará en la parte central de ésta).

Debido a la necesidad de buscar un pixel art que fuera acorde con la temática, y no caer en un arte demasiado cómico o “bonito”, se usó de referencia como base para la plantilla del protagonista el juego Never Go Home[14], un indie en desarrollo que quedó sin finalizar.

Para la definición de los enemigos se generó una IA básica, que se basa en la búsqueda y ataque del jugador. En base a ello se generaron enemigos con

distintas capacidades, que se complementan entre tipos de enemigos para generar un conjunto que sea un reto para el jugador.

Zombie

"Antiguamente el equipo de mantenimiento y tripulación del Miskatonic eran humanos, ahora tras haber sido transformados y devorados entre ellos, no son más que un amasijo de huesos, carne expuesta y jirones de ropa… aunque aún puedes reconocerlos por su mono azul"

Figura 12: Zombie

El zombie es un enemigo con ataque cuerpo a cuerpo, y una velocidad de movimiento alta. Su función es pegarse al jugador y encerrarlo entre varios.

Salud	Ataque	Tipo
100	5	Melee

Huella

"Las huellas son antiguos humanos que han quedado expuestos a la otra realidad, y andan entre varias dimensiones ahora… imposibles de detectar sin un haz de luz directa, son una pesadilla que te descuartizarán sin que veas venir el ataque"

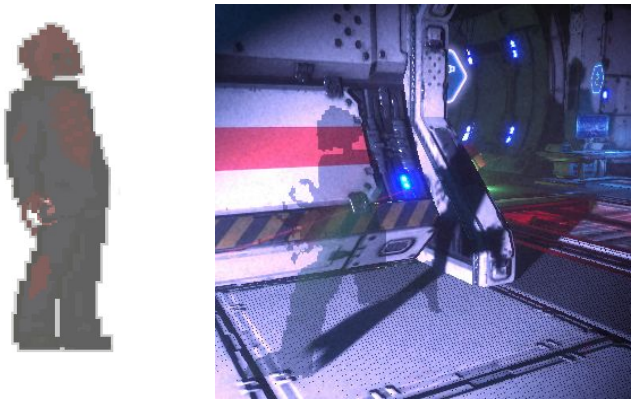

Figura 13: Huella

El enemigo denominado “huellas” o zombies-sombra (nombre interno al proyecto), son una versión idéntica al zombie, sin embargo con un uso de transparencias, son muy difíciles de ver en ambientes oscuros, sin embargo siguen proyectando sombra como si fueran un objeto opaco, por lo que la linterna o luces directas pueden exponerlos fácilmente.

Salud	Ataque	Tipo
100	5	Melee

Verdugo eléctrico

“Los verdugos eléctricos eran gran parte del personal de mantenimiento y soporte de la Miskatonic, tras la activación del gran artefacto alienígena, quedaron cargados de la energía de la nave en su asimilación. Son un peligro realmente alto si se llegan a acercar a tí, es recomendable eliminarlos desde la distancia”

Figura 14: Verdugo eléctrico

Una versión más peligrosa aún que los zombies normales, se muestran con un foco de luz azul incorporados, por lo que son siempre visibles y generan una iluminación extra en la habitación. Esto les hace especialmente visibles, junto al haz electrónico que desprenden, pero se compensa esta exposición al jugador con un daño en área. Es decir, el zombie realiza su conjunto de ataques cuerpo a cuerpo (como el zombie convencional), pero como añadido extra dispone de un rayo directo que hace daño por segundo al jugador (y otros enemigos) en un área cercana.

Salud	Ataque	Tipo
100	5 Melee / 5 por segundo en área	Melee/ área

Soldado sin nombre

"Mantenían el orden y la justicia en la Miskatonic, además de proteger al equipo científico durante las expediciones… sin embargo se vieron superados rápidamente por las criaturas salidas del artefacto. Ahora pertenecen a sus filas y harán lo posible por defender a sus nuevos amos".

Figura 15: Soldado sin nombre

Los "soldados sin nombre" (o Green en nombre interno al proyecto), son una versión de ataque de largo alcance de los enemigos que nos encontraremos. Con una variante de color verde claro del mismo uniforme que el personaje, estos enemigos son una IA que dispara el arma equipada, que es un componente separado, por ello se les puede equipar de manera sencilla armas y generar variantes. Los soldados básicos suelen ir equipados con 2 modelos de armas de plasma: disparos lentos (con unas bolas verdes de movimiento lento), o metralletas plasma (que disparan a meyor velocidad y cadencia).

Variante rifle

Salud	Ataque	Tipo
200	10	a distancia /lento

Variante ametralladora

Salud	Ataque	Tipo
200	5	a distancia /rápido

Atlach-Nacha

"Estas arañas enormes parecen ser la primera avanzada que ha salido del portal generado por el artefacto. Son criaturas que proveen completamente de otro plano existencial, a diferencia de los zombies o soldados que han sido asimilados.
Son extremadamente peligrosos y un enemigo contra el que es necesario usar todo el potencial disponible".

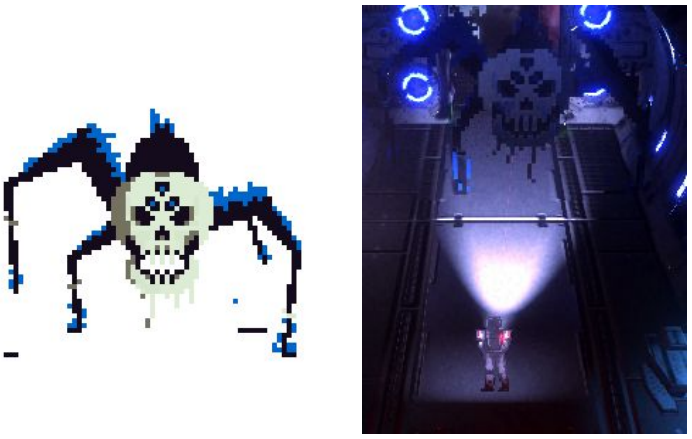

Figura 16: Jefe de nivel

Diseñado como enemigos pesados, las arañas son una versión pesada de enemigos que cuentan con una ametralladora de plasma (como la de los soldados), y la capacidad de generar descargas (como el verdugo eléctrico). Con una cantidad de vida considerable se pueden considerar mini-bosses.

Salud	Ataque	Tipo
600	10	a distancia / rápido

Debido que el sistema de armas se define por compras, los enemigos al ser eliminados agregan "experiencia" al jugador que puede ser canjeada, el listado de experiencia está definido a continuación, según la dificultad de los enemigos:

Enemigo	Experiencia obtenida
Zombie	10
Huella	20
Verdugo Eléctrico	30
Soldado sin nombre (versión rifle)	30
Soldado sin nombre (versión ametralladora)	30
Atlach-Nacha	50

3.3 Definición de armas

Un factor importante del juego es la cantidad de armamento disponible para el jugador. El sistema de desbloqueos, está pensado para que el jugador pueda comprar desbloqueos en varias sesiones de juego y pueda utilizarlas en las terminales de acceso, según situación. Por ello es importante que las armas tengan una utilidad diferenciada al menos definidas en grandes rasgos que el jugador pueda identificar. Por ello el juego divide las armas según el tipo de munición, siendo cada munición un tipo de rasgo que diferencia el arma, y a su vez cada arma tiene pequeñas variaciones dentro del tipo de munición.

Tipo de munición	Característica	Armas disponibles
Balas	Disparo directo, sin proyectil.	rifle, ametralladora, escopeta
Plasma	Disparos que rebotan, útil para uso en pasillos.	rifle, ametralladora, escopeta, lanzagranadas
Misil	Explosivos con daño en area elevado	lanzagranadas, lanzacohetes
Láser/Eléctrico	daño por tiempo.	rifle, escopeta

Para la compra de armas en los terminales del juego es necesario desbloquearlas canjeando “experiencia”. Por ello en cada arma se indica la experiencia necesaria para su desbloqueo.

Balas

Las categoría de balas se componen de armas con daño instantáneo, el disparo genera una línea directa en dirección al puntero mediante un LineRenderer. Al no instanciar una bala que debe atravesar el objetivo, el daño se realiza de manera directa mediante un Raycast.

Rifle

Arma básica con que el jugador inicia. Cadencia lenta pero gran daño. El rifle es el arma inicial del jugador, que le capacita a eliminar enemigos de todo tipo con una cierta dificultad.

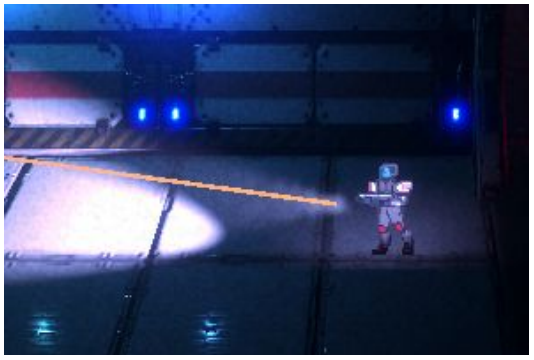

Figura 17: rifle

Daño	Capacidad del cargador	Cadencia de disparo	Coste
20	20	1 disparo por segundo	10

Metralleta

Una versión con más cadencia y menor daño que el rifle, sin embargo, es una opción bastante viable debido a que su cadencia de disparo supera con creces al daño acumulado que se realiza. Sin embargo, al quedarse más rápidamente sin balas, el tiempo de recarga es vital para aprovechar el arma.

Figura 18: Ametralladora

Daño	Capacidad del cargador	Cadencia de disparo	Coste
10	30	10 balas por segundo	50

Escopeta

Dispara un cono de daño, pero dispone de 2 balas en el cargador. Al hacer daño en área es una buena opción contra zombies que atacan en grupo, debido al elevado daño de los cartuchos.

Figura 19: Escopeta

Daño	Capacidad del cargador	Cadencia de disparo	Coste
20 * por línea * 4 lineas	2	1 disparo cada 2 segundos	50

Plasma

Las armas de plasma disparan balas con rebotes y explosión al último impacto. Su utilidad se centra en el uso en pasillos o sitios cerrados, sin embargo los rebotes pueden afectar al jugador, por lo que puede ser un arma de doble filo.

Rifle Plasma

El rifle de plasma permite disparar una bola de plasma que estalla tras tener un nº de rebotes parametrizados. Cada rebote (sea la superficie que sea) genera un daño sobre dicha superficie, por lo que si una bala de plasma rebota entre enemigos, todos recibirán daño.

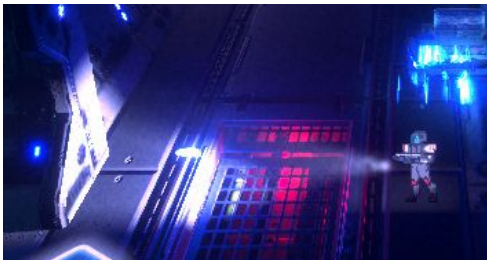

Figura 20: Rifle de plasma

Daño	Capacidad del cargador	Cadencia de disparo	Coste
30 por rebote	30	1 disparo por seg.	60

Metralleta plasma

Al igual que en su comparativa de balas, las versión ametralladora de plasma, hace menos daño pero las balas tienen más velocidad y cadencia.

Figura 21: Ametralladora de plasma

Daño	Capacidad del cargador	Cadencia de disparo	Coste
20 por rebote	30	3 disparos por segundo	100

Escopeta Plasma

Dispara 3 bolas de plasma con un “recoil” aleatorio, es decir el ángulo de las bolas varía en cada disparo.

Figura 22: Escopeta de plasma

Daño	Capacidad del cargador	Cadencia de disparo	Coste
20 por rebote * 3 disparos	4	2 disparos por segundo	100

Granada plasma

La granada de plasma genera un área eléctrica que hace “tics” de daño por segundo, por lo que mientras dure el área todo lo que esté dentro recibirá daño a lo largo del tiempo.

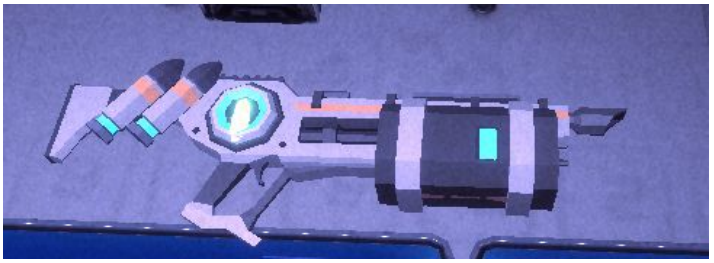

Figura 23: Granada de plasma

Daño	Capacidad del cargador	Cadencia de disparo	Coste
70 por segundo	5	1 disparo cada 3 segundos	120

Láser

Las armas láser realizan disparos continuados en el tiempo, con lo que una bala de un arma láser puede generar un disparo que está activo durante varios segundos realizando daño.

Rifle láser

El rifle láser genera un láser de manera continua durante unos segundos, que aunque no hace daño en gran medida por sí solo, sí que hace daño continuado que puede ser mortal. Genera rebotes en las normales del láser, por lo que en pasillos es muy útil.

Figura 24: Rifle Láser

Daño	Capacidad del cargador	Cadencia de disparo	Coste
50 por segundo	4	1 disparo cada 4 segundos	150

Cañón eléctrico

El cañón eléctrico ha sustituido a la escopeta láser. Este arma era demasiado problemática para el jugador y siempre terminaba asesinandose por el rebote de varios lasers y resultaba frustrante. Inspirada en la Thunderbolt de Quake 1, este arma ahora dispara descargas eléctricas hasta 3 enemigos cercanos siempre que estén en rango. Si el jugador dispara el arma y no hay enemigos cerca, la bala (la batería) se usará durante el tiempo de disparo (unos segundos), y mientras el disparo esté activo el jugador podrá acercarse a enemigos. Es decir, mientras esté el disparo encendido, el arma continuamente rastrea hasta 3 enemigos cercanos para hacer daño por tiempo.

Figura 25: Cañón eléctrico

Daño	Capacidad del cargador	Cadencia de disparo	Coste
50 por segundo * 3 rayos	5	Disparo cada 5 segundos	180

Cohetes

Las armas de cohete/granada, generan una explosión de daño elevada y son peligrosas en espacios cerrados, pero efectivas contra grupos.

Granada

La granada básica rebota varias veces antes de explotar y hacer un daño elevado. Si el número de rebotes no se alcanza, estallará por tiempo.

Figura 26: Granadas

Daño	Capacidad del cargador	Cadencia de disparo	Coste
80 explosión en área	5	1 disparo cada 3 segundos	150

Lanzacohetes

Dispara un cohete en línea recta que estalla al entrar en contacto, generando un área de daño. Ideal para espacios abiertos y oleadas conjuntas.

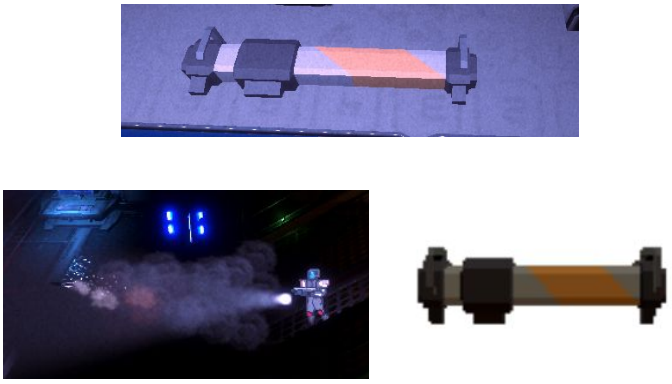

Figura 27: Lanzacohetes

Daño	Capacidad del cargador	Cadencia de disparo	Coste
80 explosión	5	1 disparo cada 3 segundos	160

3.4 Arte del juego

Como se ha definido anteriormente, la búsqueda de una ambientación oscura era el punto primordial. Tras establecer este punto y definir un uso de aplicación de personajes 2D con mapas 3D. Aunque diversas ideas de conceptos se llevaron a cabo sobre como “mezclar” los estilos, los estilos que se debatían para cada formato se dividieron sobre los siguientes puntos:

- Personajes:
    - 2D obligatoriamente:
        - Estilo de dibujo Darkest Dungeon, o Deep Sky Delericts: un estilo de comic noir.
        - Pixel Art
    - Escenarios:
        - Entornos realistas
        - Estilo Voxel
    - Objetos/armas
        - Voxel

- Low Poly

Finalmente por motivos de tiempo el uso de assets de Scifi para mapeados era más fácilmente de encontrar con estilos realistas. Para el uso de personajes se redujo a un estilo de pixel Art por tiempos, y el uso de armas y objetos en Low Poly.

4. Diseño técnico

4.1 Entorno técnico

Los Render Pipelines[15] de Unity permiten configurar y definir cómo es el renderizado y pueden otorgar una mayor calidad (con un mayor consumo de recursos). Los distintos RP disponibles en Unity son URP (Universal Render Pipeline, antes llamado LightWeight RP), el HDRP (High Definition) y el Built-In Render Pipeline (integrado en el motor).

Aunque existe una comparativa ofrecida por Unity, el resumen de las principales diferencias entre los distintos RP (comparativa) tenemos los siguientes:

	Built In	URP
Flare Layer	SI	NO
Depth + Normals Texture	SI	NO
Limite de luces	sin limite	8 por objeto, 256 por cámara
Efecto visual de Halo	SI	NO
Efecto Lends Flare	SI	NO

Sin embargo, aunque el URP parezca tener bastantes límites, (la tabla resume los más destacables), este está pensado para poder ofrecer un mayor rango de configuración a los dispositivos, permitiendo que el juego generado pueda ejecutarse en varias especificaciones.

En el caso de “The crypt of the other gods” la estética buscada se halla entre un pixel art para el diseño de los personajes, buscado con un entorno oscuro y realista. Por ello la búsqueda de efectos visuales entorno a la luz y sombra es

muy importante, por lo que se terminó decantando por el uso de HDRP, para poder aprovechar todas las mejoras gráficas que disponemos.

A continuación se puede observar un ejemplo de cómo afecta el uso de un Render PipeLine respecto a otro, utilizando el mismo escenario. Estos ejemplos han sido colgado en el Reddit[16] de Unity, en un hilo donde se realizan las comparativas visuales.

Figura 28: Comparativa LWRP / HDRP

Como se puede apreciar el sistema de sombras puede ser más definido, además de poder jugar con luces y sombras en mayores distancias y de manera más definida, además de agregar efectos visuales.

Otro ejemplo colgado en el mismo hilo sobre un escenario desértico

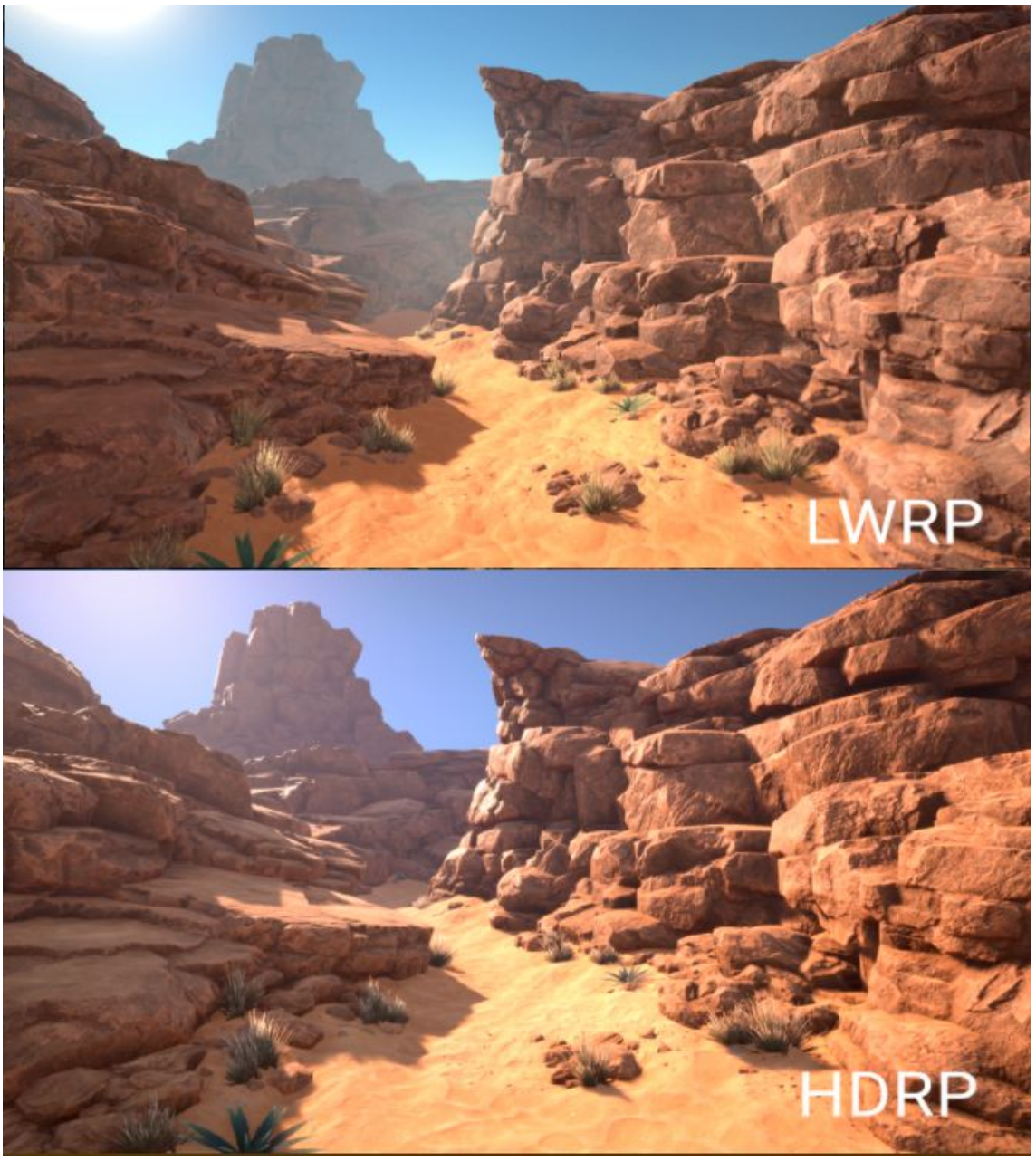

Figura 29: Comparativa de los RenderPipelines en un escenario

Por lo que el uso de luces y sombras observado en el HDRP era un objetivo a alcanzar para el desarrollo del juego.

Junto al uso del HDRP se agregan efectos de procesado para potenciar el ambiente oscuro buscado añadiendo los siguientes efectos:

- Niebla (efecto de niebla con alta exposición en la lejanía)
- Sombras de contacto: sistema de sombras de HDRP que contienen mayor detalle
- Oclusión de ambiente (ambient occlusion): simula sombras en superficies que están cercanas a otras (como esquinas, agujeros o desniveles)

- Mapa tonal (tonemapping): ajusta el nivel de colores, en este caso con una configuración a neutral, para impacto mínimo.
- Bloom: efecto óptico que genera brillo sobre una superficie que proyecta luz y afecta a otras superficies cercanas.
- Exposición (exposure): ajusta la exposición de las luces
- Desenfoque de movimiento (Motion Blur): desenfoque de los objetos al moverse.
- Profundidad de campo: desenfoque del mapeado lejano.
- Pelicula granular (film grain): efecto de granulado de cine.
- División de tonos (split toning): ajusta los tonos de la imagen, en este caso con unos ajustes azulados y grisáceos
- Balance de blancos (white balance): ajuste de colores para evitar dominaciones de color y corregir la temperatura de la imagen (en tonos).

A continuación podemos ver una imagen con todos las diferencias entre el pre y post procesado en la misma escena:

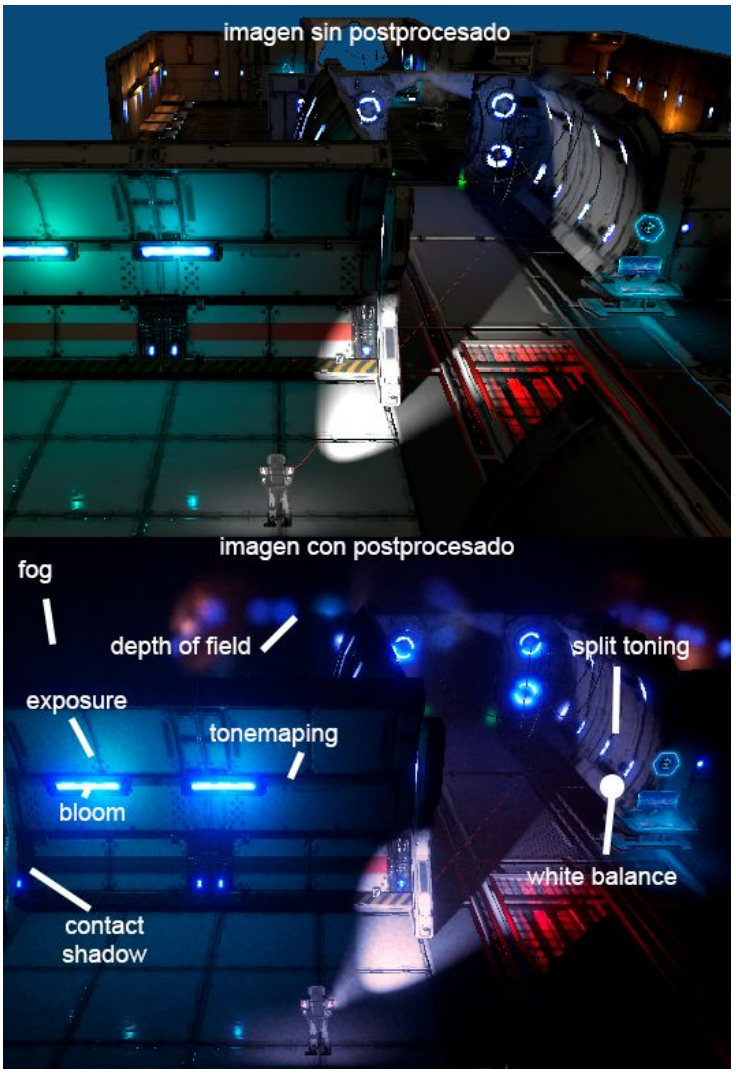

Figura 30: Ejemplo de postprocesado en el escenario del juego

4.2 Arquitectura

A continuación se define los componentes que conforman la arquitectura del sistema. Para ello se va a explicar como se genera el flujo de ejecuciones, el “Game Loop”. Una vez detallado como funciona el juego de manera global, se incidirá sobre la arquitectura de cada parte, como el generador de mapas, o el sistema de inventario-armamento, y sus interacciones.

4.2.1 Arquitectura general

El flujo del juego se basa en una rejugabilidad, por ello se optó por el uso de un generador de niveles que crea el nivel a la hora de comenzar. El generador de niveles es un GameObject dentro de la escena de Unity, que genera el mapa según una base de parámetros. Por ello se pueden generar escenas donde cada Generador de Mapas puede ser personalizado en cada escena de una manera distinta.

En el bucle de juego tienen interacción varios componentes encargados del control del jugador, el manejo de enemigos, el sistema de compra, UI y el inventario. Estos componentes de la arquitectura interactúan entre ellos para poder pasar información necesaria.

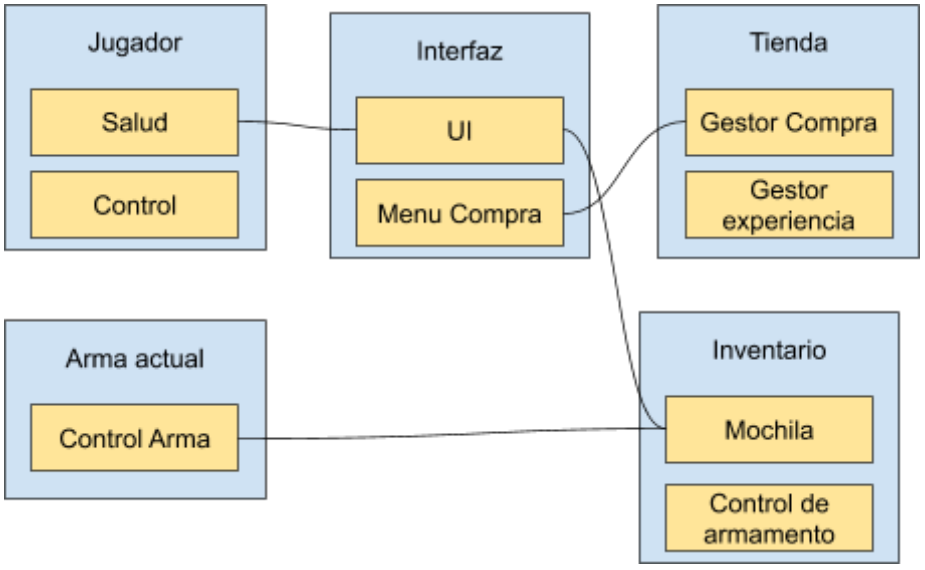

Figura 31: arquitectura general

Los componentes que conforman la experiencia de juego básica son:

- Salud: Componente que lleva la cuenta de la salud del personaje, se comunica con el componente de UI para actualizar la salud en la interfaz cada vez que se produce una modificación (por daño o curación). Además lleva la lógica para apagar los controles en caso de muerte del jugador.

- Control: El componente que recibe y controla el input del usuario. Gestiona el movimiento del jugador y las animaciones asociadas a éste.
- Control Arma: El arma equipada por el jugador en el momento actual. Lleva la lógica de disparo del propio arma, el sistema de recarga, y la comunicación con el inventario/mochila para extraer la munición.
- Mochila: El jugador puede llevar un nº de armas parametrizado, además de una cantidad de munición de cada tipo, independiente del arma equipada. Estos valores se llevan en este componente, que además interactúa con la interfaz para reflejar la munición y arma equipados.
- Menú Compra: El menú de compra se activa al usar terminales, el menú de compra interactúa con el sistema de compra para indicar las armas desbloqueadas en el menú y poder ofrecer al jugador si desbloquear o generar un arma.
- Control de armamento: Este componente realiza las funciones de soporte a otros componentes. Contiene información sobre las armas y munición, y ofrece todo tipo de funcionalidad, como obtener el tipo de munición según arma, generar armamento en el mapeado y transformación de la información entre distintos componentes.

4.2.2 Armas

El sistema de diseño de armas ha sido realizado de manera que se intentara aprovechar la funcionalidad al máximo minimizando el código, por ello existe el script AbstractWeapon, una clase abstracta que lleva la funcionalidad base y sirve de interfaz para el resto de componentes. De AbstractWeapon heredan las armas del juego según el tipo de disparo, y pueden ser personalizadas por un sistema de parámetros.

Las clases que heredan de Abstract Weapon son:

- Line Weapon: Este script se utiliza para armas cuyo disparo se generan mediante un Raycast al objetivo (sin generar balas), y dibujan un LineRenderer, este componente lo usan las armas de “balas” en el juego, como la escopeta, rifle o ametralladora.
- LaserWeapon: Arma de láser (en este caso el rifle láser), genera objetos del tipo láser (que contienen un LaserScript), y según el número de rebotes ajusta el inicio y fin de cada láser.
- Bullet Weapon: todas las armas que disparan munición contienen este script, que genera una bala desde un prefab al generar un disparo. Debido a que la bala instanciada contiene la lógica del tipo de munición, este script es lo bastante genérico como para poder ser usado en las armas de tipo “plasma” o en el lanzacohetes.
- ElectricWeapon: Este script es específico del cañón eléctrico, ya que escanea enemigos en un área cercana para realizar daño.

4.2.2.1 AbstractWeapon

El AbstractWeapon fue diseñado para que todos los componentes que tuvieran que interactuar con un componente de arma, no tuvieran que estar haciendo distinciones según la clase, por ello se creó como clase abstracta, ya que serviría de interfaz. Debido a que las armas tienen comportamiento en común, se ha movido funcionalidad a este script, por ello en vez de ser una interfaz pura, se ha generado como abstracta.

AbstractWeapon contiene la definición pública de los siguientes atributos:

- Tipo de arma y munición: variables de tipo enumerado que identifica el tipo de arma que representa, y el tipo de munición utilizada. Estos enumerados se utilizan para comunicarse entre los componentes, de manera que si se necesita recargar un arma, se le pasa la cantidad de munición requerida y el tipo, al componente de Mochila.
- Fire CoolDown: tiempo parametrizable para definir cuanto tiempo debe pasar entre disparo y disparo, con esto podemos convertir un rifle en una ametralladora, variando el tiempo y ajustando la cadencia de disparo.
- max Clip: la cantidad de munición que puede llevar como máximo el cargador del arma, y por lo tanto ajusta la necesidad de recarga. Variando este parámetro podemos convertir un cargador de 30 balas en un sistema de 2 cartuchos para una escopeta.
- RecoilX y recoilY: las variables de recoil (horizontal y vertical) representan la dispersión del arma. Un valor se añade de 0 al valor marcado, generando un aleatorio en cada disparo, de manera que se puede generar un cono de dispersión.
- Damage: daño realizado, en este caso está definido por el arma para cada disparo (o bala).
- Perfect Critic: cuando el usuario realiza una recarga perfecta, la munición que se agrega es crítica, con lo que el daño que se realiza en el disparo actual (si la bala es crítica) se multiplica por el valor de este parámetro. En otras palabras: indica el porcentaje por el que se realizará el daño. Ejemplo: 150 será un 150% del daño actual.
- Bullets Per Shoot: El número de balas que se generan por disparo, con este parámetro podemos definir la cantidad de disparos que realiza el arma (con toda su lógica asociada). Con esta variable se puede transformar un rifle en una escopeta, ajustando el número de balas disparadas en un solo tiro, ayudado con el ajuste del RecoilX/Y, y creando un cono de dispersión.
- Audioclips de recarga, equipación y disparo: los clips de audio para el sistema de recarga y el disparo y equipar el arma.
- Información del sistema de recarga: las variables que se definen a continuación indican los tramos de la barra de recarga:

- Standard Reload: tiempo máximo de la recarga (en caso de que el usuario no vuelva a pulsar Recarga). Es decir es el tiempo de recarga base.
- Perfect Reload: El tiempo donde comienza la barra de recarga perfecta, si el usuario pulsa de nuevo Recarga después del segundo indicado en esta variable, la recarga agregará munición crítica.
- Active Reload: Segundo en el que acaba la recarga perfecta y comienza la "recarga activa". Finaliza el tramo de agregar munición crítica, y comienza la de Activa. Durante este tramo el usuario si pulsa Recargar, la recarga finalizará antes de lo normal sin problema.
- Active Reload End: finalización del tiempo de recarga activa (y continuación del tramo de recarga estándar).

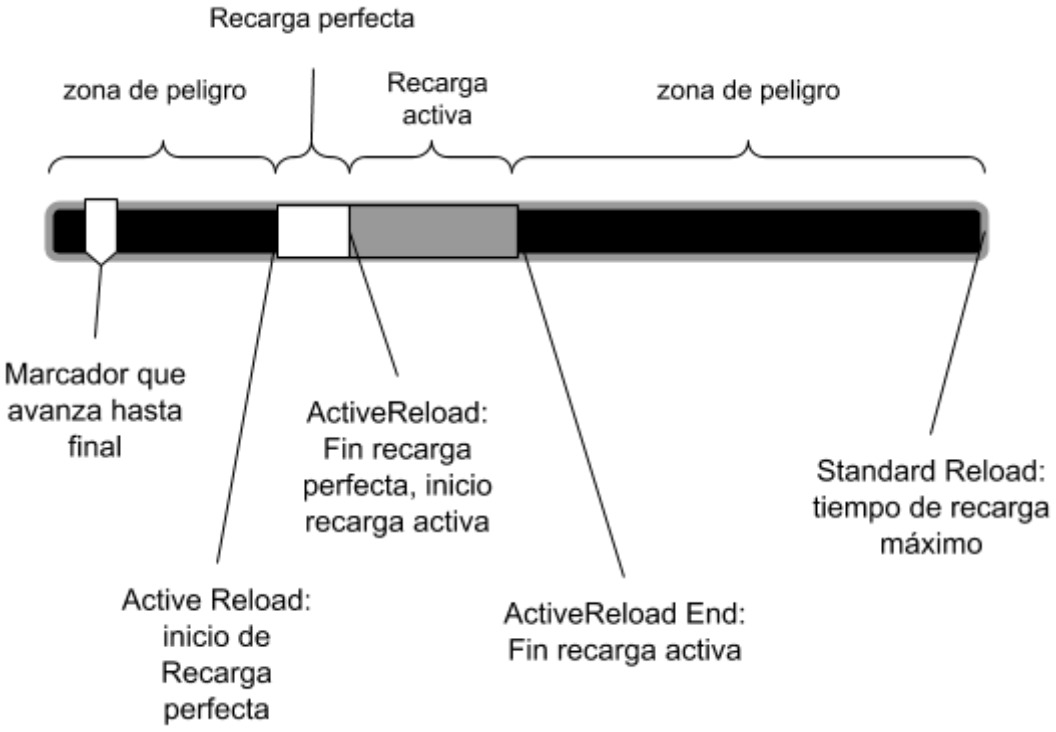

Figura 32: Barra de recarga

La clase de AbstractWeapon contiene la funcionalidad base de las armas, y dejando una llamada a una función abstracta en cada caso que puedan implementar las clases heredadas:

- Start: Como clase MonoBehaviour, el arma tiene que inicializar sus parámetros al activarse, genera un "muzzle" en caso de estar definido, y define la barra de recarga del arma. Acto seguido llama a la función abstracta que es definida por las clases heredadas:

    - OnStarting: función que se ejecuta en Start en caso de que cada arma deba configurar algo al inicializarse.
- Update: Controla el input del jugador para activar las variables que indican el disparo o recarga. Acto seguido se llama a la función abstracta para las clases heredadas:
    - OnUpdating: función que pueden definir las clases heredadas en la fase de Update.
- FixedUpdate: Si el usuario ha pulsado Recarga, en FixedUpdate se lanza la corutina de recarga (que genera la animación de la barra). En caso de haber pulsado Disparo, se llama a la función abstracta que deben implementar cada subclase:
    - Shoot: la función que genera el disparo, específica para cada clase heredada
    - Reload: función que se llama al inicio de una recarga, aunque toda la lógica de recarga está definida en la clase padre, se ha generado esta clase por si las clases heredadas deben generar algún tipo de lógica en la recarga.

4.2.2.2 LineWeapon

La lógica interna de esta clase, es la de ofrecer un tipo de arma que genera una línea entre la boca del arma y el objetivo. La funcionalidad de las armas en las funciones abstractas es la siguiente:

- OnStarting: instancia los lineRenderers (que provienen de prefabs, así se pueden configurar independientemente)
- Shoot: genera un raycast entre la boca del arma y la posición del cursor (en el mismo plano que el personaje), si existe Recoil configurado, se añade una variación aleatoria de dispersión al raycast. Se pinta un lineRenderer (de los pre-instanciados) desde el inicio del raycast hasta el objetivo (que puede ser el final del raycast o un objeto que haya chocado antes). En caso de colisionar con un Collider se realiza un SendMessage("Damage", damage), de manera que si el objeto impactado puede recibir daño lo procesa. Esta funcionalidad se genera por cada disparo configurado en "Bullets Per Shoot".

4.2.2.3 LaserWeapon

La clase de LaserWeapon funciona de manera muy parecida al LineWeapon, sin embargo, debido a la particularidad del rebote del laser y que está en vivo unos segundos (en vez de ser un disparo directo) se ha derivado en una nueva lógica.

Para definir la lógica del uso del láser, existen unos nuevos parámetros:

- Number of bounces: el nº de veces que puede rebotar el láser disparado.
- ActiveTime: el tiempo que dura activo el láser tras ser disparado.

La lógica del arma es la siguiente:

- OnStarting: Instancia y desactiva los prefabs del láser. El prefab del láser contiene la lógica de pintado del láser. El número de instancias que se generan es: “bullets per Shoot” multiplicado por el número de “balas por disparo”, de manera que si se configura el arma para disparar 2 láseres con 2 rebotes, se generan 4 instancias de láser.
- Shoot: los prefabs de láser instanciados se generan, y se dispara la corutina DisableLasers, que esperará el tiempo definido por el “ActiveTime” y desactiva los láseres.

La lógica de cada láser, con su propio script es el encargado de definir su inicio (la boca del arma), su dirección, y si alcanza una superficie, definir un vector de reflexión según la normal del impacto, de manera que tenemos la dirección del nuevo rebote. De esta manera se generan los láseres iniciales, se calculan los rebotes, y se pintan luego, y así hasta finalizar el número de rebotes.

Figura 33: escopeta láser desechada

Ejemplo de una variante fallida de arma “escopeta láser”, que generaba 3 láseres con 1 rebote cada uno.

4.2.2.4 ElectricWeapon

El cañón eléctrico se generó como variante de un arma láser, debido a su poca funcionalidad en el juego. Por ello se generó la variante que busca enemigos cercanos en un área por un tiempo activo para generar daño por tiempo. Para esta nueva arma los nuevos parámetros son:

- MaxEnemies: el nº de enemigos a los que se puede atacar dentro del área de alcance.
- Distance: el área de ataque del arma viene definido por un circulo cuyo radio es definido aqui.
- ActiveTime: igual que el script anterior, define el tiempo que los rayos están activos.

La lógica del arma se define así:

- OnStarting: los rayos se instancian y se desactivan, para tenerlos cargados previamente, se utiliza un Asset cuyo script LightningBoltScript

puede definir un inicio y un final como GameObjects y en tiempo real generará el rayo.

- Shoot: Con una lógica muy parecida a las armas láser, activa los rayos y genera una corutina que los desactiva una vez pasado el tiempo definido. Sin embargo aquí existe la mayor diferencia, y es que entre la activación y apagado de los rayos, una variable define el estado "Disparando" del arma, lo cual se verifica en cada update.
- OnUpdating: en cada update del MonoBehaviour, genera un Physics.OverlapSphere y localiza los Colliders dentro de la esfera. Se revisa cada Collider y si pertenece a un GameObject cuyo tag sea de "Enemy", se fija como inicio del rayo, el arma, y como fin el enemigo. Así hasta alcanzar a todos los enemigos o llegar al nº de enemigos definidos por "MaxEnemies". Esta ejecución en cada Update hace que si un enemigo entra o sale de la esfera de colisión, el estado de los rayos se actualice en tiempo real, redireccionando los rayos si un enemigo sale del área o entra alguno nuevo. Los enemigos afectados por rayos reciben el daño definido 1 vez por segundo.

4.2.2.5 BulletWeapon

El script cuya funcionalidad puede considerarse la más genérica, ya que contiene una serie de parámetros que sirven para configurar el comportamiento del arma, aparte de delegar la lógica del funcionamiento de la bala en el Prefab del proyectil disparado. Para la configuración de este tipo de arma, además de los parámetros heredados, se generan las siguientes:

- BulletPrefab: Aunque todas armas usan su munición desde un prefab, en este caso tiene mención especial, debido a que el proyectil instanciado, contiene un script Bullet.cs propio con la lógica de la bala, lo bastante configurable como para que sea un misil, una granada o una bola de plasma.
- UsePhysics: Si el arma usa la física en el disparo o no. En caso de usar física, se le aplica una fuerza al proyectil al instanciarlo, parametrizado. En caso negativo, el proyectil es simplemente instanciado. Este parámetro sirve de especial utilidad para generar armas que utilizan granadas.
- BulletForce: La fuerza aplicada al rigidbody del proyectil en caso de activarse el parámetro "UsePhysics".

La funcionalidad de este script en base a las funciones a implementar:

- OnStarting: Debido a que las balas se instancian en el propio disparo y no al inicio de la creación del arma, esta función se queda vacía.
- Shoot: la lógica del disparo se inicia igual que en el resto de armas: buscando la dirección del puntero, sin embargo en estecaso solo sirve para orientar la bala recién instanciada. En caso de usar física se agrega la fuerza al proyectil, y por último en caso de haber sido un proyectil

cuyo daño debía ser crítico, se asigna el daño al script de Bullet.cs en el proyectil, de manera que siempre la bala recibe el daño que debe de hacer por parte del arma.

Los proyectiles disparados usan un script Bullet, que contiene una serie de parametros que modifican el comportamiento de la bala. El componente es tan parametrizable que sirve para configurar cualquier tipo de bala/misil/granada.
Los parámetros que tiene un proyectil son:

- Speed: la velocidad a la que debe ir el proyectil, independientemente de la fuerza aplicada. En el caso de una granada se puede usar la fuerza inicial y colocar una velocidad de 0, en el caso de un proyectil de plasma, se aplica una velocidad alta sin física.
- bounces: el número de rebotes que puede tener el proyectil. Sirve para definir el número de rebotes antes de estallar en caso de una granada (usándose con física), o de un proyectil de plasma que rebota en las paredes a velocidad constante.
- MaxTimeAlive: un proyectil puede quedarse de manera indefinida vivo. Debido a errores con la física una granada puede quedarse en el suelo sin detectar rebotes, o un misil/bala puede haberse colado entre alguna apertura del mapa por error y quedar “en el vacío” de manera indefinida, por ello el proyectil pasa a activar la lógica de impacto tras el tiempo definido si no ha impactado antes.
- ExplosionPrefab: si se tiene asociado un prefab de explosión, el proyectil lo instanciará al impactar, si no se define prefab, el proyectil aplicará el daño directamente.
- KeepAfterExplosion: En caso de la granada de plasma, la explosión genera un área de daño alargado en el tiempo, por ello se precisó que el objeto de proyectil tenía que seguir vivo tras la explosión. Si no se activa, en caso de explosión, el proyectil se autodestruye.

La lógica del proyectil se realiza de la siguiente manera:

- Start: Se configura el uso de componentes internos y se activa el contador con “MaxTimeAlive” para la destrucción del proyectil.
- Update: se verifica si el proyectil debe destruirse por tiempo. En caso positivo, se aplica el nº de rebotes al máximo (indicando que el proyectil ha alcanzado el máximo de rebotes posible) y se invoca a una colisión. En caso negativo (el proyectil está vivo) se aplica la velocidad aplicada al proyectil.
- OnCollisionEnter: Si el número de rebotes es inferior al máximo, se incrementa el contador de rebotes, se genera el vector de rebote según la normal del impacto, y se rota el proyectil a su nueva dirección. En caso de haber alcanzado su máximo de rebotes (o de no tener), se verifica si se ha definido un “ExplosionPrefab”, en caso afirmativo se instancia la explosión, y si el “KeepAfterExplosion” está como False, se

destruirá el proyectil (en caso negativo, se pasa al script de la explosión como parámetro el propio proyectil, para que sea la explosión el encargado de destruirlo).

4.2.3 Inventario

El inventario actúa como una clase Singleton con la función DontDestroyOnLoad, de manera que se mantiene vivo durante la transición de mapas (al ser Singleton se verifica antes de instanciarse si ya existe alguna clase previa).

La funcionalidad del inventario sirve para:

- Instanciar y configurar las armas que se recogen por el mapa.
- Almacenar la cantidad de munición guardada en el inventario.
- Recibir el input de cambio de arma y cambiar el arma activa.
- Mostrar en la interfaz la cantidad de munición.
- Realizar recargas en las armas.

Para ello mantiene la información de la munición en variables privadas y se pueden acceder mediante las funciones: **GetAvailableAmmo(AMMO_TYPE ammo)** y **AddAmmoToInventory(int ammo, AMMO_TYPE ammoType)** y un setter. Como se puede apreciar siempre se utiliza el enumerado que indica el tipo de munición al que se quiere acceder, de manera que si un arma quiere preguntar si puede cargar o extraer munición, le pasa a **GetAvailableAmmo**, qué tipo de munición y usa. Para la recarga se utiliza **int ExtractAmmoFromInventory(int Ammo, AMMO_TYPE ammoType)**. Si un arma invoca al inventario para extraer 30 balas de tipo láser, se verificará la cantidad de munición que realmente se puede extraer (puede que el inventario tenga menos), y la función retorna la cantidad extraída con éxito (que puede ser menos si no se tenía suficiente munición, 0 si no se disponía, o la cantidad requerida) y actualiza la cantidad de munición del inventario y actualiza interfaz.

Para la configuración del armamento, cuando el jugador recoge un arma, el script se encarga de actualizar la interfaz para representar la munición y el arma usadas, reiniciar la barra de recarga con la nueva configuración y desactivar el arma anteriormente equipada. En caso de que el jugador haya recogido un arma y esté en su tope de armas equipadas, el script de inventario invocará al componente WeaponSpawnManager para que instancie en el mapa un arma igual a la que el jugador va a desechar (al intercambiar el arma actual pro la nueva recogida), con la información necesaria. Este componente (el WeaponSpawnManager), además sirve para proporcionar información a través de unos scriptables que contienen la información asociada a las arnas, como icono de munición, icono del arma u otra información asociada que no está asociada con su funcionamiento.

4.2.4 Sistema de compra

El sistema de compra se divide en 3 componentes: ShopTable, ShopMenu y ExperienceManager. Mientras que ShopTable lleva la gestión del terminal en el mapa ShopMenu tiene todo el sistema de lógica de compra y desbloqueos en la interfaz, que a su vez se soporta en el ExperienceManager, para verificar cuales son los desbloqueos del usuario y su experiencia acumulada.

- ShopTable: script sobre los terminales encontrados en el mapa, gestiona la activación del menú de compra al acercarse el usuario y pulsar Usar.
- ShopMenu: desactiva el arma del jugador (para no disparar al usar el menú), y renderiza el menú. Los botones de desbloquear armas o comprarlas se basan en la información disponible del Experience Manager.
- ExperienceManager: El sistema almacena en PlayerPrefs los desbloqueos del usuario, así como la experiencia que obtiene eliminando enemigos. De esta manera entre partidas se guarda lo acumulado.

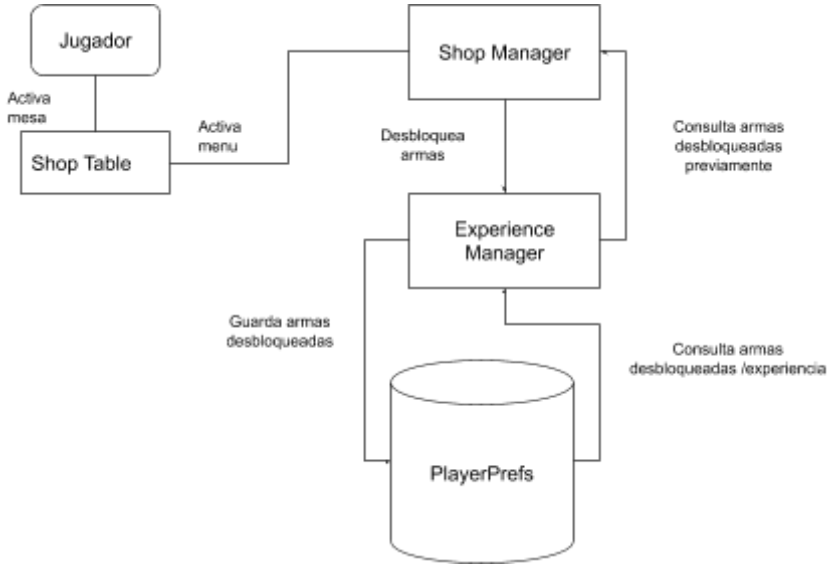

Figura 34: Sistema de compra

En definitiva el sistema de compra/experiencia permite al jugador realizar desbloqueos de armas a cambio de gastar experiencia obtenida eliminando enemigos. Tanto la cantidad de experiencia obtenida, como los desbloqueos se producen en memoria permanente, por lo tanto el jugador guarda tanto las armas como la experiencia entre partidas, ya que se almacena en disco, mediante el uso del sistema de Unity de PlayerPrefs.

Cuando el usuario se acerca a una terminal y activa dicha terminal, se activa el sistema de compra, que activa un menú en el canvas, desactivando el arma del

jugador y movimiento (para evitar que pueda disparar/moverse mientras compra. El menú a la hora de abrirse consulta al sistema que desbloqueos se tienen ya realizados y renderiza el menú.

Figura 35: Menú de compra

4.2.5 Integración 2D/3D

Para el uso de personajes 2D sobre entornos 3D era necesario y primordial el uso de luces y sombras, de manera que los sprites 2D reflejaran las luces y fueran capaces de proyectar sombras.

Para ello antes del desarrollo del proyecto se investigó con el uso de SpriteRenderer en Unity con el uso de materiales en HDRP. Finalmente se consiguió en unas primeras fases lograr el efecto siguiente, usando unos sprites del juego Metal Slug.

Figura 36: Primeras aproximaciones a 2D/3D

Para conseguir el efecto deseado se ha utilizado un SpriteRenderer con el diseño del personaje y un material con la siguiente configuración:

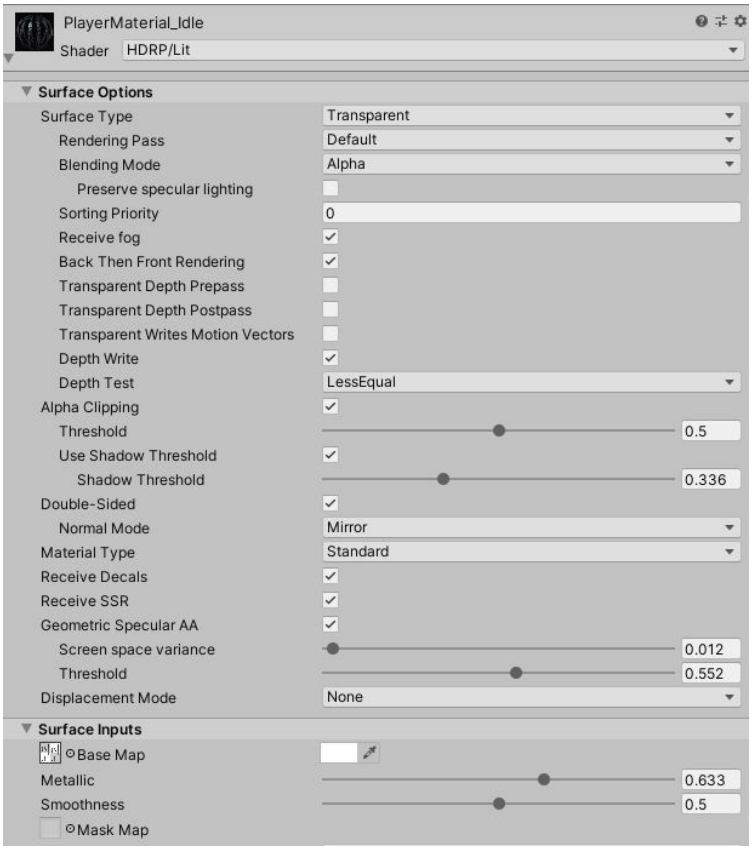

Figura 37: Material 2D

De manera que el material es capaz de ser transparente usando el alpha del mismo dibujo que el SpriteRenderer (usando la misma imagen que en

BaseMap). Para que el sistema funcione se deben dar las siguientes condiciones:

- La animación por frames debe estar activa, es decir el AnimationController debe tener la aminación puesta en dicho momento, y la animación está basada en frames.
- La textura debe corresponder con la imagen de la animación.

Es decir, con cada cambio en el AnimationController, además de cargar cada animación mediante Mecanim, es necesario además cargar una textura.

Para poder usar este sistema es necesario adaptar la máquina de estados de Mecanim para que en cada transición, además de cargar la animación mediante un script llamado SetNormalMap, se aplica la textura asociada a la animación, por ello por cada animación, existe un spritesheet con la animación, y una textura.

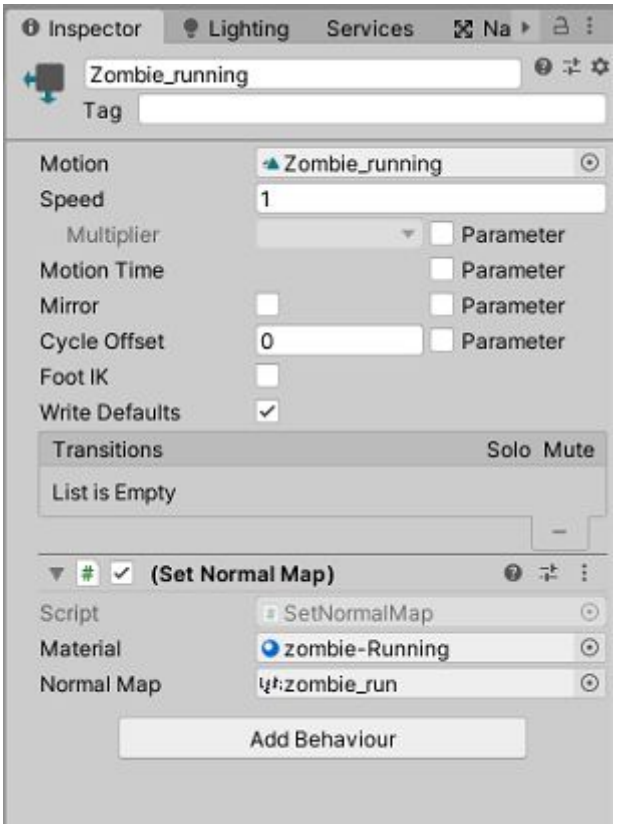

Figura 38: Script de cambio de material en animación

En la imagen superior podemos apreciar un estado de Mecanim donde se aplica la animacion “zombie_running”, además se aplica la textura y el normal Map, de manera que todo queda dentro del sistema de animaciones.

4.2.6 Generador de mapas

El proyecto comenzó con la idea de aplicar 2 mapas (con posibilidad de hasta 3) de manera estática, donde la aparición de enemigos estuviera indicada por puntos de “spawn”. Sin embargo en fase inicial de desarrollo rondaba la idea de aproximar el juego al roguelike lo más posible, generando un sistema aleatorio de niveles para ello. Aunque se revisaron varios métodos de generación de niveles, las limitaciones encontradas siempre eran sobre sistemas 2D simples, por lo que se complicaba la tarea de encontrar documentación al respecto de uso sobre componentes 3D.

Finalmente se optó por un sistema de generación aleatoria que verifica el solapamiento en 3D de habitaciones, este sistema permite generar niveles 3D en varias alturas, de manera que se puede dar un sistema de múltiples pisos en un nivel.

La versión inicial del sistema genera niveles de la siguiente manera:

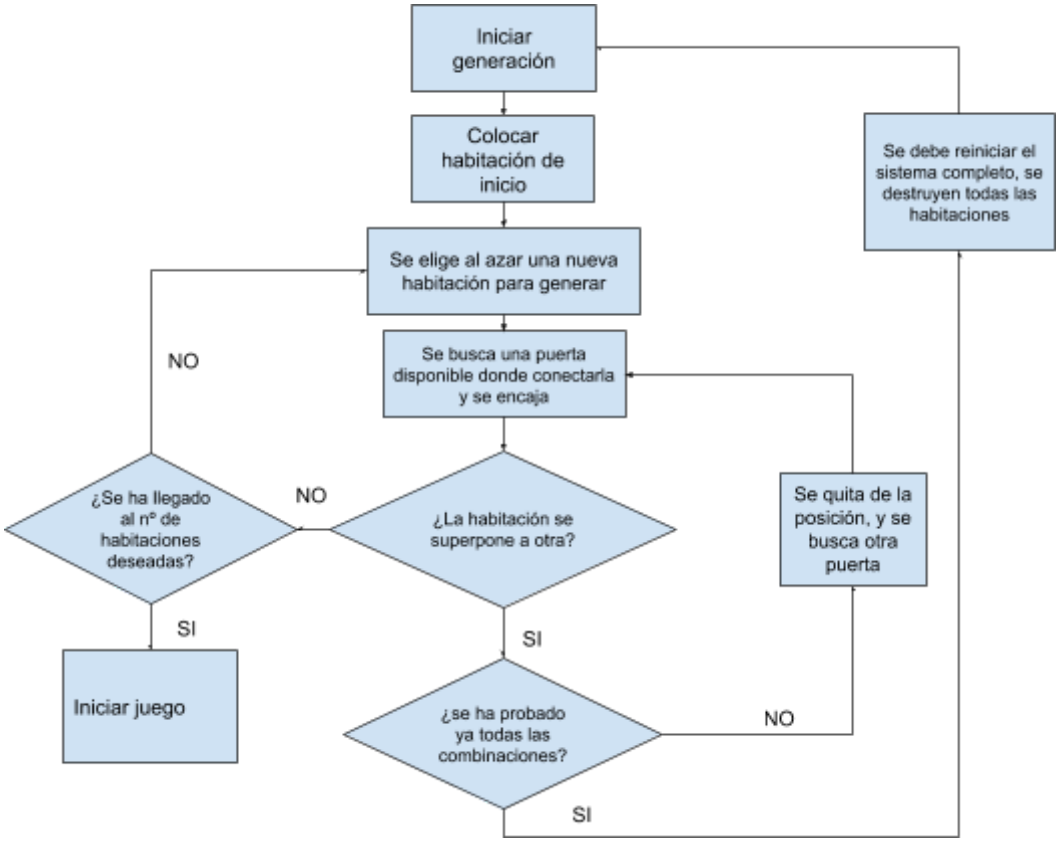

Figura 39: Flow de LevelBuilder

El sistema funciona de la siguiente manera:

1. El script **LevelBuilder** coloca la habitación de inicio, prediseñada de manera específica (no se coloca una habitación cualquiera).
2. Se añade la puerta de esta habitación al listado de puertas en el mapa posibles (para conectar nuevas habitaciones)

3. Se añade de un listado de habitaciones posibles una seleccionada aleatoriamente al mapa.
4. Se busca la manera de conectar una puerta de esta nueva habitación, con una de las puertas del mapa (seleccionada aleatoriamente).
5. Se rota la habitación para que la puerta conectada encaje.
6. Se emplaza la habitación.
7. Se comprueba si la nueva habitación ocupa físicamente el espacio de otra, es decir, se verifica si se solapa en el mapa.
8. En caso de solaparse, se procede a buscar otra puerta para conectar y reposicionar y rotar la habitación de nuevo, así hasta agotar todas las posibilidades. En caso de que la habitación no haya podido ser encajada en ningún otro sitio se destruye y se reinicia el sistema.
9. En caso de haber conectado la habitación de manera correcta, las puertas de la nueva habitación se colocan en el listado de posibles puertas para conectar, y se vuelve al punto 3.
10. Cuando se han ubicado todas las habitaciones, se coloca la habitación final, que igual que la inicial, es una habitación especial definida manualmente.

El generador de niveles tiene parametrizado un nº mínimo y máximo de habitaciones, de manera que si no se cumple el mínimo, el sistema se reinicia completo.

El sistema inicial se diseñó con habitaciones y pasillos de distintos tamaños, y aunque funcionaba de manera correcta, cuando el número de habitaciones aumenta el sistema tiende a quedarse demasiado tiempo generando niveles.

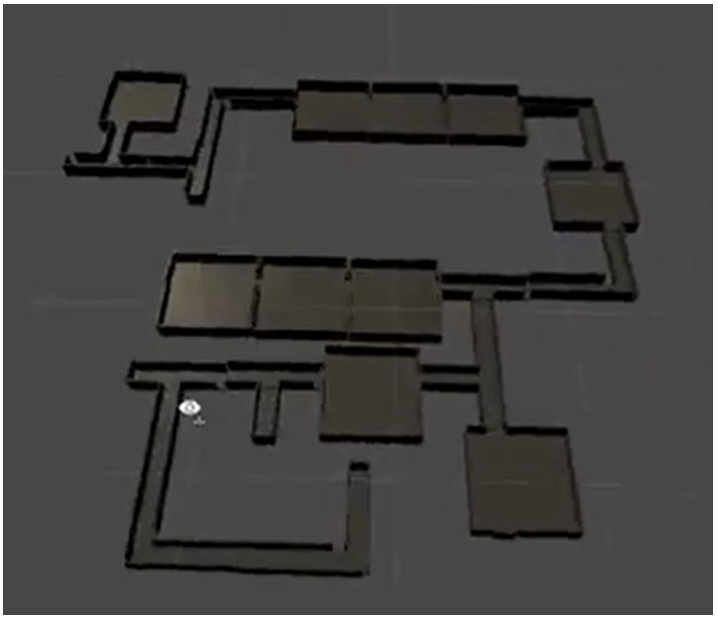

Figura 40: Nivel generado aleatoriamente

Con el sistema base generado se detectaron varias mejoras que evolucionaron el generador de niveles. Con el objetivo de generar mapas más grandes en tiempo efectivo, se realizaron las siguientes modificaciones:

- En vez de desechar el nivel al no poder ubicar una habitación, se desecharía la habitación del “pool” de habitaciones a colocar (si no hubo ninguna ubicación con éxito), y se pasaría a colocar otra, de manera que el sistema sólo se reiniciará si todas las habitaciones a colocar no han podido ser ubicadas. En otras palabras, el mapa ha quedado construido de una manera que no pueden ubicarse habitaciones nuevas de ninguna manera.
- El diseño de habitaciones se realiza de manera idéntica en tamaño y posición de puertas. De esta manera la generación del mapa se realizará mediante un sistema de cuadrículas, si todas las habitaciones miden lo mismo y los puntos donde están ubicadas las puertas están el mismo sitio, el sistema quedará conectado sin error. En caso de quererse habitaciones mas grandes, siempre debe ser un múltiplo de la unidad de una habitación, de manera que siga correspondiendo con una cuadrícula.

Figura 41: Salas

De manera que, como en la imagen de arriba, una habitación es cuadrada ocupando el ancho total posible y con 4 conexiones, otra solo dispone de 2 y tiene una forma modificada.

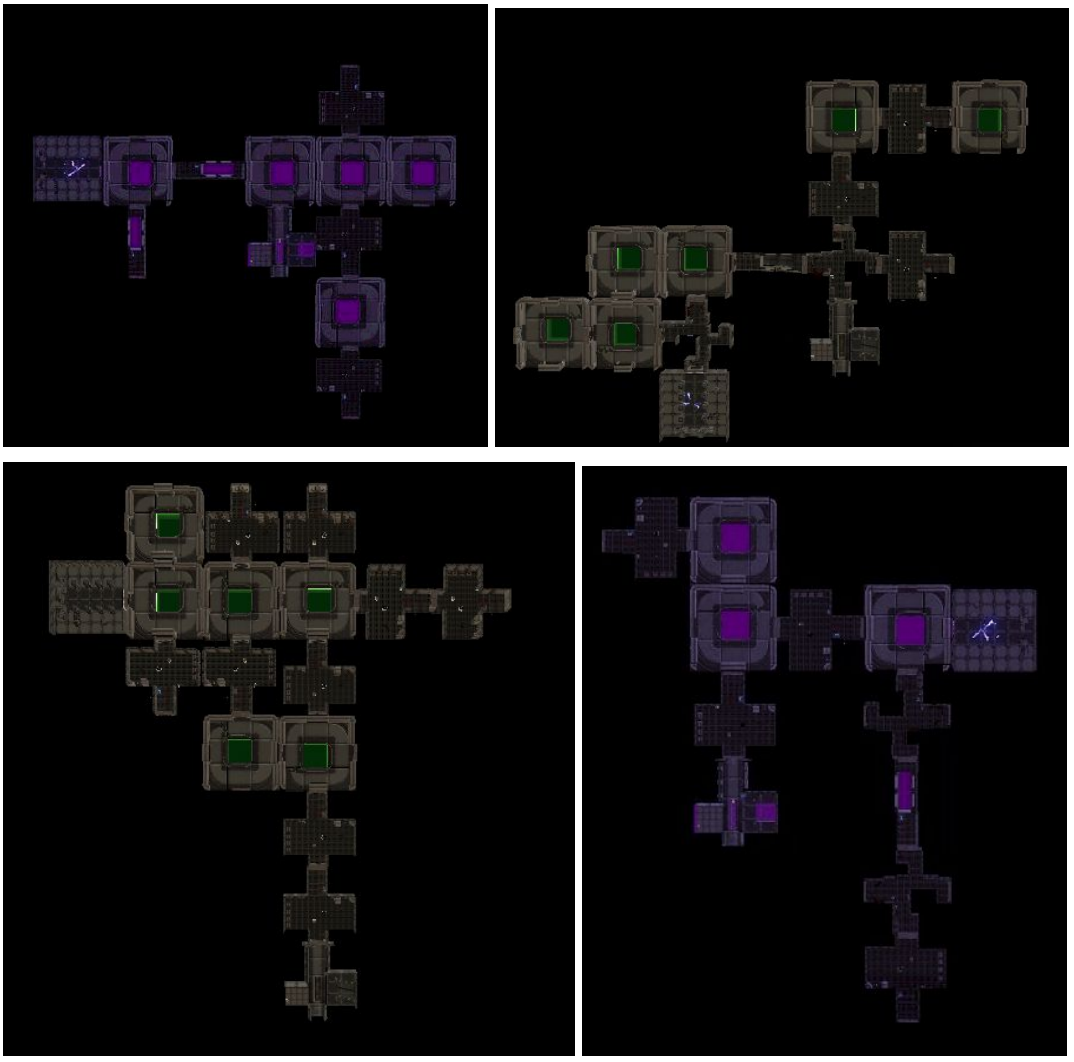

Figura 42: Ejemplos de mapas generados

4.2.6.1 Postprocesado del nivel

Una vez generado el mapa completo se procede a un postprocesado del generador de niveles, para realizar las conexiones, ubicar las llaves y limpiar paredes que dificultan la jugabilidad.

4.2.6.1.1 Conexiones entre habitaciones

Cuando las habitaciones se conectan entre ellas, se hacen mediante el contacto de 2 “Quads” (planos) en Unity, que tienen un Raycast asociado hacia el exterior de la habitación, de manera que el Quad de una puerta se coloca en la misma ubicación que la de la otra, quedando conectadas exactamente.

Como GameObject hijo de cada Quad, se ubica una puerta de manera estática (sin funcionalidad). Cuando las habitaciones se quedan colocadas, los Quad se eliminan y con ellos, las puertas estáticas. Estas puertas existen por sí una habitación se queda con puertas desconectadas, es decir “mirando al vacío”.

En dicho caso el Quad se limpia, pero la puerta se queda, pareciendo una puerta cerrada al usuario.

4.2.6.1.2 Colocación de llaves

Debido a que el sistema puede generar niveles donde la habitación final queda perfectamente colocada al lado de la habitación inicial, de manera que el jugador bastaría con cruzar 2 habitaciones para completar el nivel, sin necesidad de tener que recorrer todo el mapa.

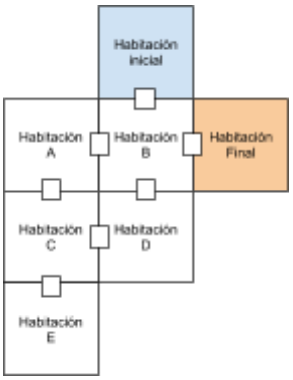

Figura 43: Ejemplo de distribución de mapa

En la imagen superior se puede apreciar un caso de mapa donde el jugador puede completar el nivel, simplemente saliendo de la habitación inicial y girando a la izquierda. Para ello se ha generado un sistema de colocación de llaves parametrizable. La habitación final tiene su puerta cerrada hasta que el jugador recolecta el número de llaves necesario (indicado al **LevelBuilder** por parámetro). Para ello el sistema de colocación de llaves, recorre todas las habitaciones, localizando su posición (su centro), luego calcula la distancia entre la habitación inicial y cada habitación, para detectar la más lejana. Una vez localizada, se ubica la llave en dicha habitación. Cada habitación dispone de 1 punto de spawn de llave por si resulta seleccionada. En el caso anteriormente puesto, la habitación seleccionada para generar la llave sería la E.

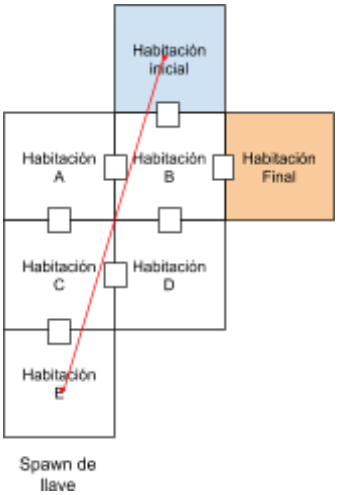

Figura 44: Colocación de llave

4.2.6.1.3 Limpieza de paredes

Una vez se han colocado todas las habitaciones en el mapa, todo está dispuesto para comenzar la partida. Sin embargo en fases avanzadas del generador de niveles, se apreció que debido a que las habitaciones tienen paredes por las 4 caras, y que las habitaciones pueden aparecer en cualquier orientación, las paredes dificultan la vista del jugador. Por ello, todas las habitaciones, una vez colocadas en el mapa, detectan la orientación y eliminan la pared (dejando un bloque invisible) de las paredes que dan al jugador.

4.2.7 Control de arenas

Debido al diseño de niveles, las habitaciones funcionan de manera similar a muchos roguelikes, como The Binding of Isaac, donde cada habitación es una arena que el jugador debe superar. El nombre del sistema viene dado a su similitud con el sistema creado por DOOM (2016) y DOOM ETERNAL, donde el jugador está encerrado con oleadas de enemigos que debe derrotar para seguir desbloqueando el nivel.

Cada habitación contiene un script con una serie de GameObjects que definen la estructura de una arena/habitación. A vista del usuario el sistema es el siguiente:

1. El jugador entra en la sala
2. Las habitaciones de la sala se cierran
3. La habitación selecciona una configuración de oleadas aleatoriamente, de las disponibles.
4. Comienzan a aparecer enemigos por distintos puntos de la sala (lo que se conoce como una oleada)
5. El jugador debe eliminar los enemigos que aparecen
6. Una vez los enemigos hayan sido eliminados, se verifica si existen más oleadas configuradas, en caso afirmativo, se vuelve al punto 3.

7. En caso de haber eliminado a la última oleada, se abren las puertas y la sala queda desbloqueada.
8. Si la sala tiene un cofre asignado, aparece.

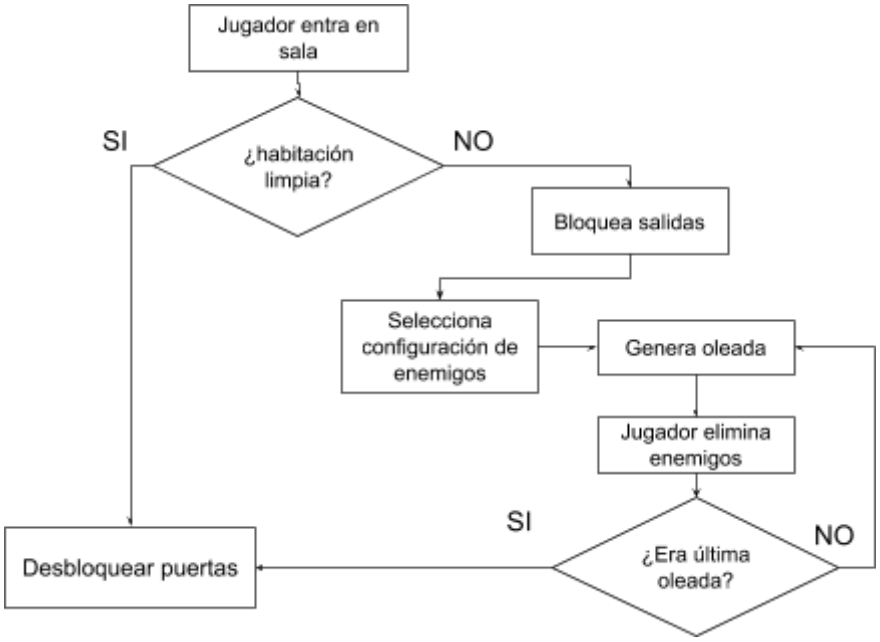

Figura 45: Flow de control de habitación

De esta manera las salas tienen un script **RoomManager**, con un trigger asociado, que ocupa un poco menos que la sala. Cuando el jugador entra en contacto con el Trigger, se procede a bloquear las puertas de la sala asociada y comienza el sistema de respawns.
Cada sala tiene una serie de configuraciones asociadas (explicado más adelante), que selecciona aleatoriamente, de manera que 2 salas idénticas, en el mismo mapa, pueden generan enemigos distintos.

Dicho sistema controla el nº de enemigos y el nº de respawns disponible en la sala: procesa la cola de enemigos de la oleada actual y asigna un enemigo a cada punto hasta ocupar todos los puntos de respawn. Existe la posibilidad de que con los puntos de respawn todos los enemigos de la oleada actual hayan aparecido, por lo que se satisface la aparición de la oleada. Sin embargo, si la oleada actual no se ha completado (existen más enemigos que puntos de spawns), antes de volver a aparecer en el primer spawn (reiniciando la lista), el sistema espera unos segundos, de manera que pueda quitarse el enemigo aparecido en primer lugar del punto de spawn.

Cuando el jugador elimina la oleada actual, el sistema verifica si hay más oleadas de enemigos disponibles, en caso afirmativo, espera un tiempo indicado y comienza a instanciar enemigos de nuevo. El sistema además permite la configuración de oleadas mediante Scriptables, de manera que generando scriptables de configuración de oleadas, podemos asignarlas a habitaciones y tener un sistema listo rápidamente.

Estos scriptables, llamados **RoomConfig**, contienen una configuración que se divide en la siguiente estructura:

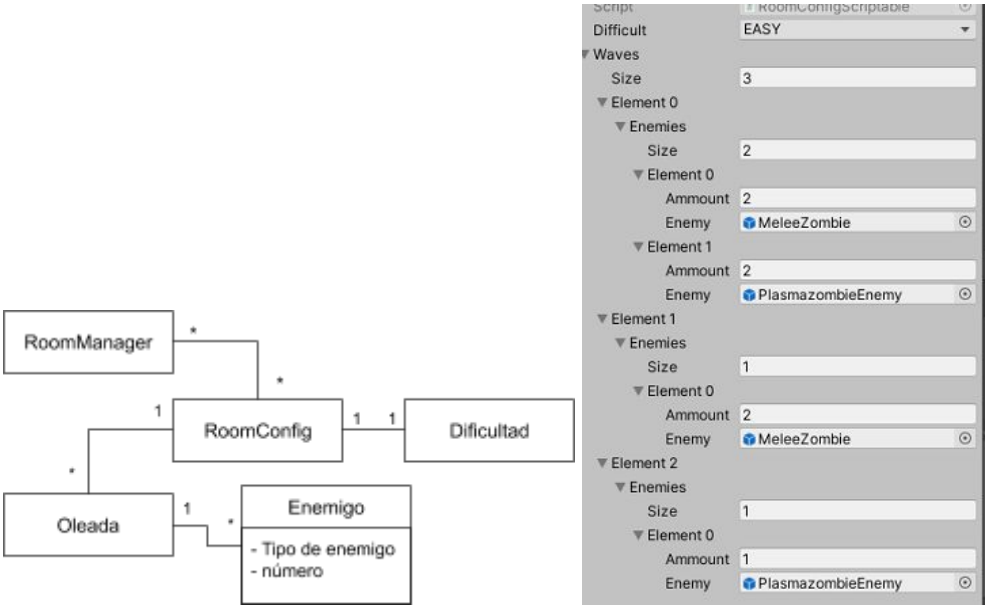

Figura 46: Configuración de enemigos

De manera que un Room Manager tiene asociado varios RoomConfig, y este a su vez tiene una dificultad indicada (para un posible sistema de dificultades en el juego). Una configuración de habitación tiene un nº de oleadas, y cada oleada tiene a su vez un número N de Enemigos, que se basan en una combinación de tipo de enemigo/nº de veces que aparece. La imagen superior indica 3 oleadas:

- Oleada 1:
    - 2 Zombies
    - 1 Zombie Plasma
- Oleada 2:
    - 1 Zombie
- Oleada 3:
    - 1 Zombie

4.2.7 Inteligencia artificial

Los enemigos del juego se rigen por una IA con un propósito general. De manera que la IA ha sido programada para todos los enemigos y se ajusta mediante una serie de parámetros. La IA enemiga es una máquina de estados que alterna entre los estados: disparar, perseguir, interceptar.

Esta IA se generó de manera genérica en paralelo al desarrollo del mapeado, de manera que la situación de las habitaciones no estaba clara aún. Debido al espacio de las salas, finalmente el estado de interceptar (que inicialmente buscaba la posición del jugador y su dirección para ir a su futura trayectoria) fue eliminado.

Los estados contienen una lógica interna (la acción del estado) y unos parámetros que varían según el enemigo:

Estado Perseguir
Parámetros - Radio de visión: el alcance en el que el enemigo actúa en el modo persecución
Lógica En este estado el enemigo localiza al jugador en su posición actual y se mueve a dicha posición
Entrada - Si el jugador entra en el rango de rango del “Radio de visión” del enemigo, entra en este estado.
Salida - Si el enemigo se encuentra a una distancia mayor que el “Radio de visión” pasaría al estado “Interceptar” - Si el enemigo se encuentra en el “Radio de ataque”, pasa el estado “Atacar”

Estado Interceptar
Lógica El enemigo localiza la posición futura del jugador (en base a su dirección, posición y velocidad), y se dirige a interceptar.
Entrada - Si el jugador sale del radio de “Rango de visión” del modo “Perseguir”
Salida - Si el enemigo se encuentra a una distancia menor que el “Radio de visión” pasaría al estado “Perseguir”.

Estado Atacar
Parámetros - Radio de Ataque: el alcance en el que el enemigo actúa en el modo ataque.

Lógica El enemigo apunta al jugador y utiliza el arma equipada (que es de manera genérica, igual que el sistema del jugador), si ésta está disponible. En caso de que no pueda, espera a que el arma pueda ser usada.
Entrada - Si el jugador entra en el rango de rango del “Radio de ataque” del enemigo, entra en este estado.
Salida - Si el enemigo se encuentra a una distancia mayor que el “Radio de ataque” pasaría al estado “Perseguir”

De manera que configurando los radios de movilidad y el arma equipada, obtenemos una variedad de enemigos. Por ejemplo, los zombies de ataque cuerpo a cuerpo, tienen un elevado rango de persecución.

4.2.7.1 Armamento enemigo

Al igual que el armamento del jugador, el armamento enemigo se basa en una clase abstracta de la que heredan las armas específicas. De esta manera la IA enemiga solo tiene que encargarse de ejecutar la función **Shoot** de la clase **EnemyWeapon**. Todos los enemigos tienen un GameObject invisible que siempre está apuntando al jugador, y rota alrededor del enemigo, de manera que el “arma” siempre está encarando correctamente.

De esta clase heredan 2 clases:

- EnemyMeleeWeapon: El arma activa el daño en un trigger asignado, es decir, el arma tiene un “área” donde realiza daño, y se activa/desactiva en cada ataque. Modificando este área, modificando el alcance del daño cuerpo a cuerpo
- EnemyBulletWeapon: Funciona de manera idéntica a las armas del jugador, sin embargo los enemigos no tienen limitaciones de cargador o munición, por lo que no deben recargar.

4.2.7.2 NavMeshAgent

Para el sistema de movimiento de los enemigos, se utiliza el NavMeshAgent, el sistema por defecto para la búsqueda de caminos y gestión de movimientos de Unity. Este sistema está basado en los agentes: entidades que se desplazan y son capaces de buscar rutas desde su posición hasta su destino de manera inteligente, y las superficies: las áreas sobre la que circulan dichos agentes.

Normalmente estos sistemas se utilizan agregando el componente NavMeshAgent a los personajes o elementos móviles y el componente NavMesh a los suelos o superficies. Tras eso se realiza un “bake”, un cálculo que genera las rutas y caminos disponibles en el escenario. La problemática de

este sistema es que requiere que el nivel esté pre-generado, ya que cualquier cambio en el mapeado en tiempo de ejecución, invalida los objetos compilados.

Debido a que el juego genera el mapeado en tiempo de ejecución (generando el nivel al cargar la escena) este método no resulta útil. Por ello es que se ha utilizado los componentes de NavMesh que Unity provee mediante su github[17]. Entre estos componentes se encuentra el NavMeshSourceTag y LocalNavMeshBuilder.

NavMeshSourceTag es un componente que se asigna a todos los suelos de las habitaciones pre-generadas, que define que son superficies por las que se pueden mover los agentes. Mientras que LocalNavMeshBuilder define un área, en cuyo interior, si existen superficies NavMesh, son calculadas durante el tiempo de ejecución.
De esta manera tras la generación del nivel, mientras los suelos de las habitaciones tengan el componente, y la habitación esté dentro del área del LocalNavMeshBuilder, el cálculo de la superficie se realizará con éxito.

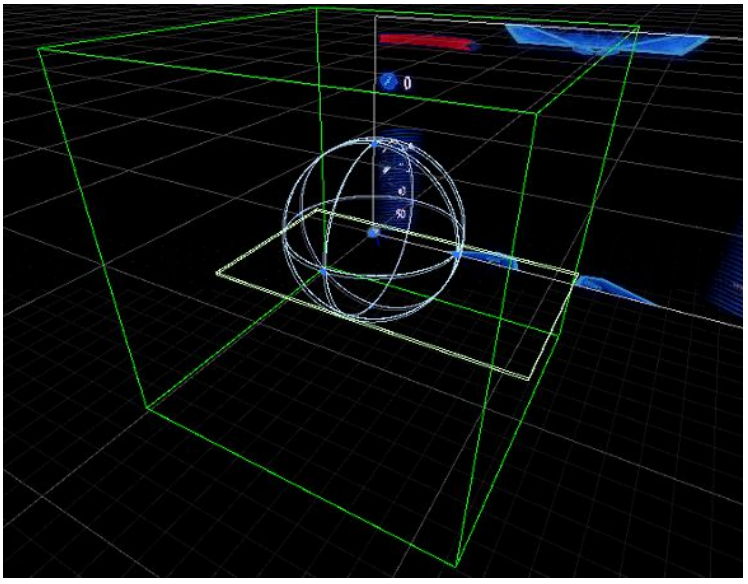

Figura 47: En verde, el área definida del LocalNavMeshBuilder

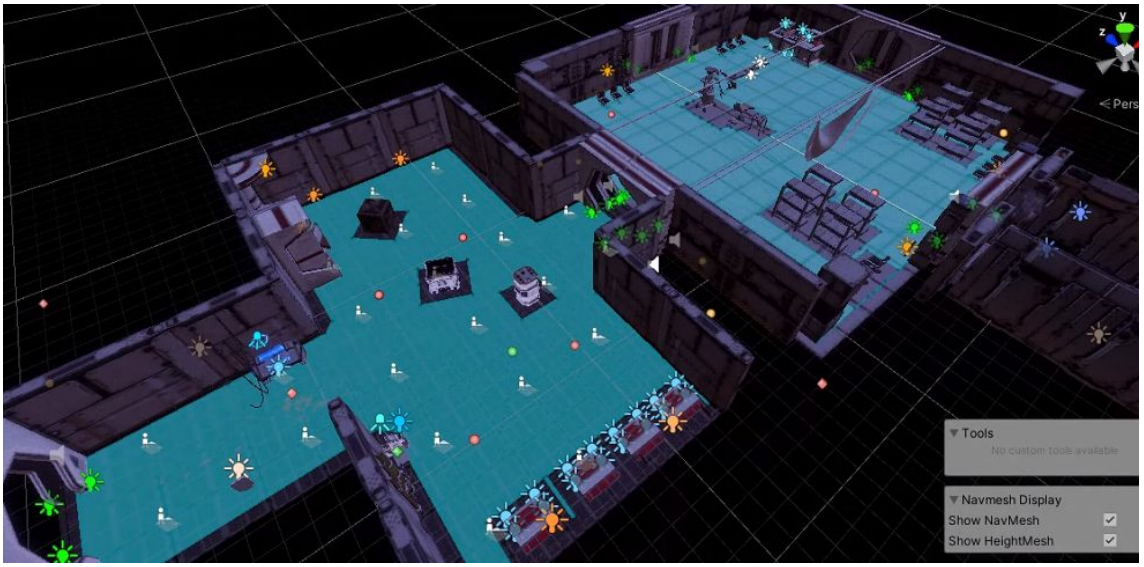

Figura 48: áreas navegables por los agentes

4.2 Utilización de Assets

Para el desarrollo del juego se han hecho uso de assets para la parte de modelados y entornos. La generación de los personajes 2D por el contrario, ha sido un proceso manual, que detalla en el apartado correspondiente el proceso de animación manual.

4.2.1 Assets 3D

El uso de Assets 3D para la elaboración de los entornos tenía como objetivo acelerar el proceso de generación de habitaciones mediante sistemas modulares que permiten acoplar paredes, puertas y suelos de manera fácil.

Los assets han sido comprados mediante la Asset Store o adquiridos mediante bundles a través de Humble Bundle.

Efectos
Efectos de sangre - Blood Splater Decal Package
https://assetstore.unity.com/packages/2d/textures-materials/blood-splatter-decal-package-7518
Efectos de partículas de Unity - Unity Particle Pack
https://assetstore.unity.com/packages/essentials/asset-packs/unity-particle-pack-5-x-73777
Efecto de láser - 3D Laser Pack
https://assetstore.unity.com/packages/vfx/particles/3d-lasers-pack-131685
Efectos de explosión y armas - War Fx
https://assetstore.unity.com/packages/vfx/particles/war-fx-5669
Efectos de portal y spawn enemigos - Realistic Effects Pack 4

https://assetstore.unity.com/packages/vfx/particles/spells/realistic-effects-pack-4-85675
Modelos Modelo de botiquín - First Aid Kit Army https://assetstore.unity.com/packages/3d/props/first-aid-kit-army-148353 Modelo de cajas - Industry Props Pack 4 https://assetstore.unity.com/packages/3d/props/industrial/industry-props-pack-4-66319 Granada y misil - Free Missile https://assetstore.unity.com/packages/3d/props/weapons/free-missile-72692 llave - KeyCard https://free3d.com/3d-model/sci-fi-door-and-key-card-286675.html Modelos de armas y algunos items de escenario - Polygon Scifi City https://assetstore.unity.com/packages/3d/environments/sci-fi/polygon-sci-fi-city-pack-115950
GUI Imágenes para menús - GUI kit Scifi Blue https://assetstore.unity.com/packages/2d/gui/gui-kit-scifi-blue-72673
Entornos Scifi Level Kit https://assetstore.unity.com/packages/3d/environments/sci-fi/sci-fi-level-kit-hd-29384 Top Down Scifi https://assetstore.unity.com/packages/3d/environments/sci-fi/top-down-sci-fi-18938

4.2.1 Assets 2D

Para la realización de los assets 2D se usaron plantillas bases realizando búsquedas de juegos pixel Art. Buscando un estilo serio se encontró como referencia base el estilo de personajes del juego Never Go Home.
Utilizando el estilo del personaje se comenzó a desarrollar los spritesheets de animación de cada movimiento, para ello se definió todas las poses y animaciones posibles. En el caso del protagonista, tendría más animaciones disponibles que los enemigos debido a los ángulos de disparo y movimiento. El protagonista tiene las siguientes animaciones disponibles:

- Mirando hacia arriba
    - Movimiento izquierda, derecha, arriba y abajo.
    - Pose quieto (idle)
- Mirando izquierda
    - Movimiento izquierda, derecha, arriba y abajo.
    - Pose quieto (idle)

- Mirando derecha
    - Movimiento izquierda, derecha, arriba y abajo.
    - Pose quieto (idle)
- Mirando abajo
    - Movimiento izquierda, derecha, arriba y abajo.
    - Pose quieto (idle)

Por ello se separó el diseño del personaje en 2 partes: la superior (que define el apuntado o hacia donde mira el personaje) y la inferior (la dirección de movimiento).

El siguiente paso fue el diseño del personaje basándose en el uso de un tamaño definido de 60 pixeles de alto aprox. y usando un pincel de 4x4 píxeles, usando para algún detalle menor 1 pixel.

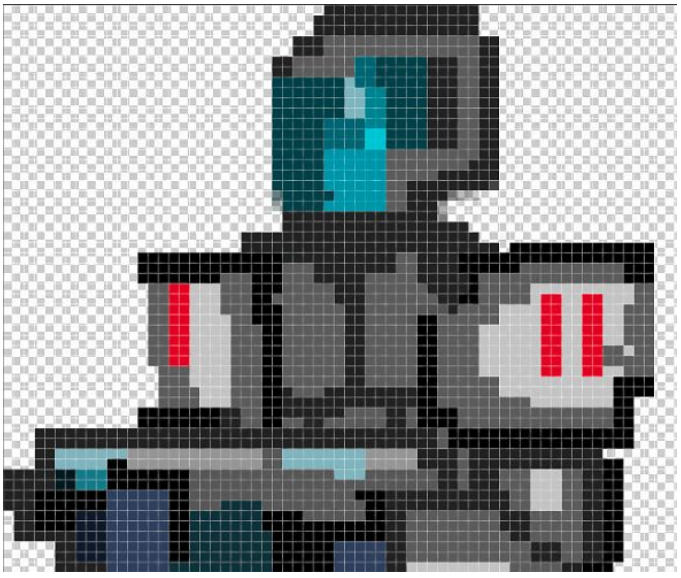

Figura 49: Diseño de personaje

A continuación, basándose en spritesheets de movimiento de personajes pixel art, se generaron las poses. El proceso manual fue de pintar la pose del personaje superponiendose a la plantilla de referencia, generando los frames 1 a 1. El proceso se realizó en Photoshop con el sistema de generación de gifs que permite trabajar frames como capas independientes.

Figura 50: Spritesheet plantilla para movimiento

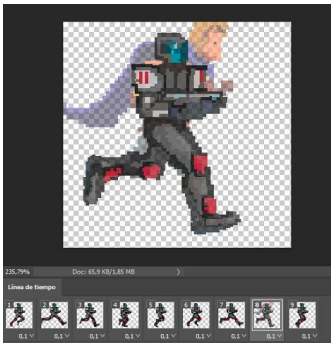

Figura 51: Proceso de dibujado sobre la animación

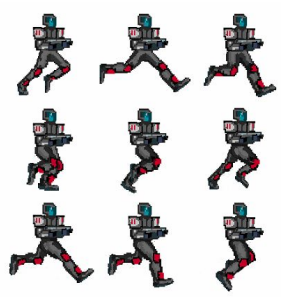

Figura 52: spritesheet resultante

El proceso se realiza con todas las animaciones, reutilizando las posibles posiciones del movimiento de las piernas (en las 4 direcciones posibles) y las posibles orientaciones del torso. El resultado son 8 posibles animaciones, con variaciones (por ejemplo: el personaje apuntando hacia la izquierda y caminando hacia la derecha, es la animación de avanzar hacia la izquierda al revés).

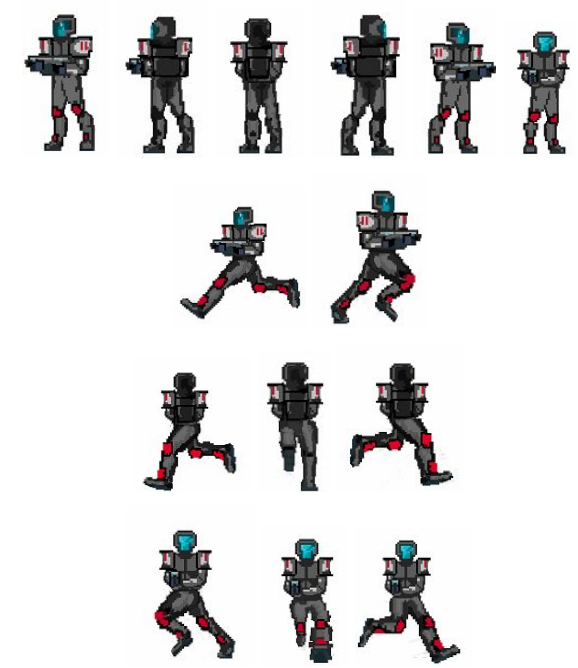

Figura 53: Orientaciones del personaje

El proceso se repite con los sprites de enemigos. Sin embargo se ha optado por reducir el número de animaciones, debido a que no requieren de tanto detalle y a un ahorro de tiempo y recursos.
Por ello los enemigos solo tienen animaciones laterales, (hacia izquierda o derecha), siempre mirando al jugador, rotando mediante código la orientación del spritesheet.

Figura 54: Spritesheet de enemigo corriendo

4.2.2 Assets Audios

Para el uso de audios se ha recurrido a audios adquiridos mediante librerías de HumbleBundle, o la web https://freesound.org/ que provee sonidos bajo licencia de uso libre.
El listado de audios utilizados de dicha web es el siguiente:

https://freesound.org/people/Jackjan/sounds/159406/
https://freesound.org/people/IchBinJager/sounds/272068/
https://freesound.org/people/UnderlinedDesigns/sounds/188499/
https://freesound.org/people/GreenFireSound/sounds/481099/
https://freesound.org/people/LeMudCrab/sounds/163458/
https://freesound.org/people/EFlexMusic/sounds/416280/
https://freesound.org/people/Paul368/sounds/200966/
https://freesound.org/people/niamhd00145229/sounds/422688/
https://freesound.org/people/ejfortin/sounds/49695/
https://freesound.org/people/Jarusca/sounds/521377/
https://freesound.org/people/Hybrid_V/sounds/321216/
https://freesound.org/people/Invisible_inks/sounds/517058/
https://freesound.org/people/xFromarge1/sounds/520036/
https://freesound.org/people/HughGuiney/sounds/352832/
https://freesound.org/people/superfreq/sounds/268101/
https://freesound.org/people/SuperPhat/sounds/531861/
https://freesound.org/people/GeronimoGeronimo/sounds/338061/
https://freesound.org/people/FlashTrauma/sounds/398274/
https://freesound.org/people/TannerSound/sounds/495054/
https://freesound.org/people/tonycarlisle/sounds/441653/
https://freesound.org/people/magnuswaker/sounds/523057/
https://freesound.org/people/pohwelly/sounds/507972/
https://freesound.org/people/egomassive/sounds/536741/
https://freesound.org/people/SilverIllusionist/sounds/518800/
https://freesound.org/people/Dpoggioli/sounds/213607/

Para el uso de músicas se ha hecho uso de packs de canciones adquiridos por Humble Bundle.

5. Diseño de niveles

En el caso actual el diseño de niveles requiere una aproximación distinta al de un juego normal, ya que nuestro sistema genera mazmorras de manera aleatoria. Por ello para el diseño se ha optado por generar cada habitación como una pequeña arena donde se ha revisado el diseño del combate.

Pasillos

Pasillo en forma de cruz con esquinas, en esta sala aparecen enemigos de cuerpo a cuerpo. El objetivo de la sala es proporcionar combate cuerpo a cuerpo en espacio cerrado en condiciones de poca luz.

Figura 55: Pasillos

Sala

Esta sala proporciona un espacio abierto de combate, donde pueden aparecer oleadas de enemigos de cualquier tipo. Sin embargo en esta sala es donde hay posibilidades de aparecer zombies semi-transparentes, por que las condiciones de luz de la sala son más favorables al jugador (ya que en la sala anterior a oscuras, eran demasiado complicados de ver).

Figura 56: Sala

Zona tóxica

La sala tóxica se provee con buenas condiciones de luz con una zona inaccesible en el centro. Por ello los enemigos que aparecen en esta sala son soldados con ataque de largo alcance, proveyendo combates a distancia de enemigos complicados de acceder.

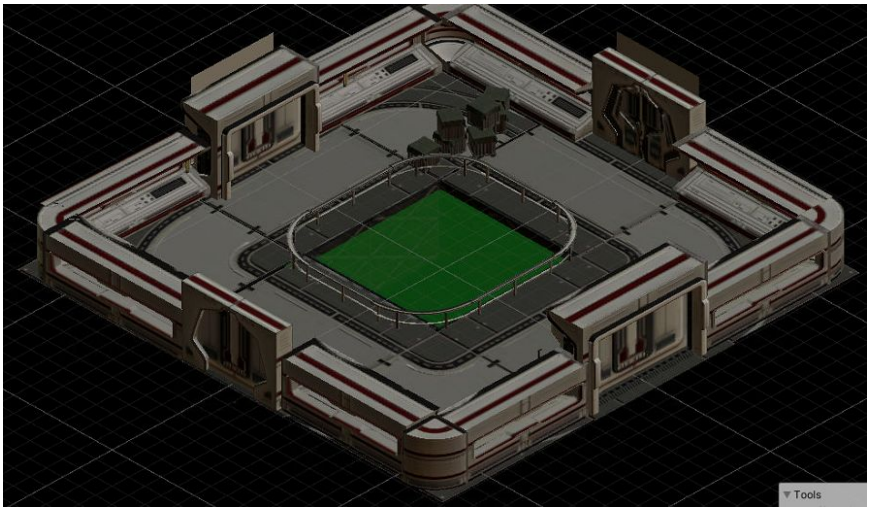

Figura 57: Zona tóxica

Oficina

Esta sala provee espacio grande pero con pequeños objetos como sillas o mesas distribuidos y una enorme columna de ordenadores en el centro. Esta sala provee ambos tipos de oleadas: zombies o soldados, por lo que ofrece variedad de combate.

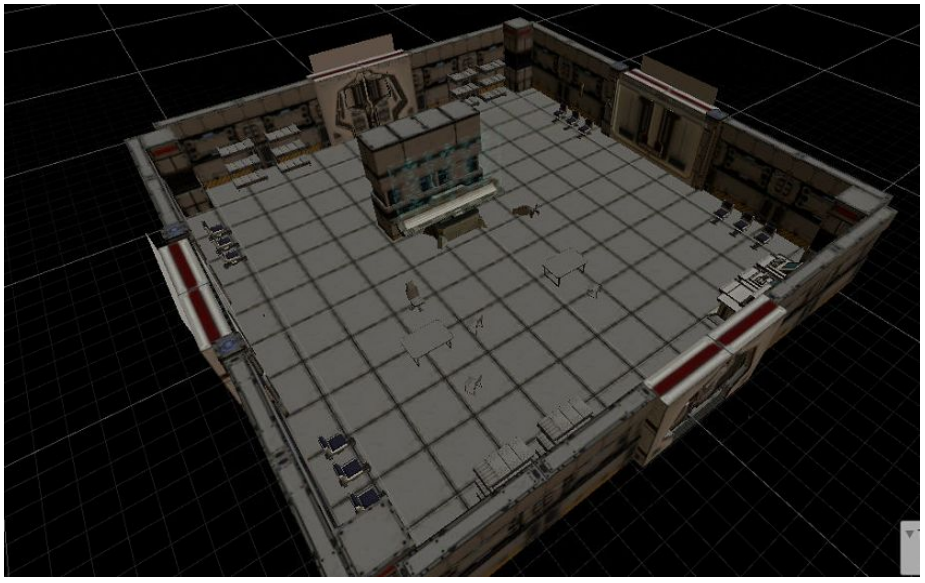

Figura 58: Oficina

Enfermería

Esta sala es una variante de la anterior donde el uso de una tela (a modo de separado de la zona de camas) dificulta la visión del jugador en una sala donde pueden aparecer también ambos tipos de enemigos.

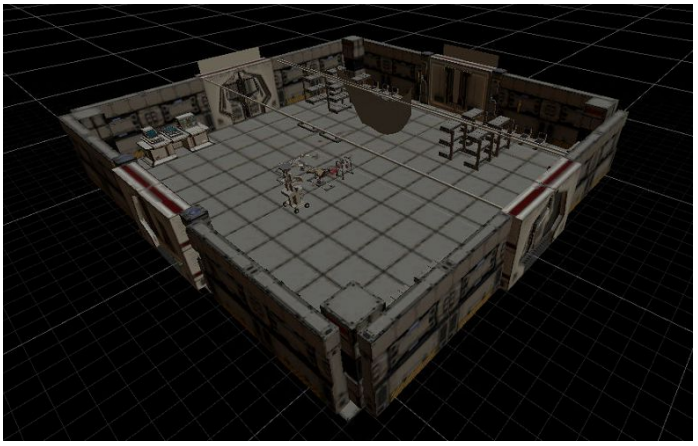

Figura 59: Enfermería

Sala final

Al final de cada nivel, la salida aguarda al jugador con una oleada especial. En esta sala el jugador se encuentra en un pasillo con pilares a enemigos con armas de largo alcance y a la araña enemiga.

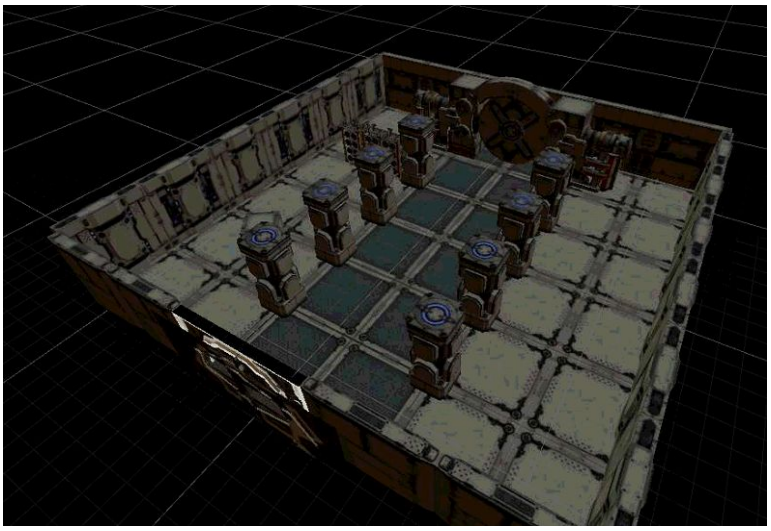

Figura 60: Sala final

6. Manual de usuario

6.1 Requerimientos técnicos del hardware

Los requisitos donde se ha probado el juego son:

- Windows 10
- Nvidia GForce 980 GTX
- 16 Gb RAM
- 1.2 GB de espacio en el disco duro
- DirectX versión 10
- Procesador i7-6700K 4GHz

6.1 Instrucciones del juego

Para lanzar el juego basta con hacer click en el ejecutable del juego, The Crypt of the Other Gods, con sus siglas **TCOTOG**.

Figura 61: Icono del juego

Una vez arrancado el juego accederemos al menú, donde podremos seleccionar entre jugar el tutorial o iniciar el juego.

- Si seleccionamos el tutorial, el usuario podrá conocer como jugar y las mecánicas del juego.
- Si seleccionamos iniciar el juego, comenzaremos la intro del juego, que podemos avanzar mediante clicks de ratón o pulsando Enter.

Una vez arrancado el juego, el usuario puede usar los siguientes controles para jugar:

- Movimiento: teclas WASD o flechas de dirección.
- apuntado: uso del ratón
- Recarga: R (pulsar de nuevo para realizar la recarga manual, en la barra de recarga)

- Cambio de arma: Rueda de ratón
- Usar: F (para recoger arma, abrir baúles o usar terminales de compra)

Para recargar se puede pulsar R

Para poder avanzar por los mapas el usuario debe eliminar a todos los enemigos de la sala actual. Una vez el jugador ha localizado la sala final, debe abrir la puerta a dicha sala con la llave, que está en el mapa. Por lo que una vez encontrada la llave (basta con tocarla para recogerla) se puede acceder a dicha sala y eliminando al jefe del mapa, se activará el portal al siguiente nivel.

7. Conclusiones

Durante el desarrollo del juego se han llegado a distintas conclusiones derivadas del proceso de desarrollo. A continuación se exponen divididas por categorías.

7.1 Lecciones aprendidas

- Una de las principales complicaciones del desarrollo del género roguelike/lite es el balance entre todos los actores del juego, de manera que no quede un juego demasiado sencillo o complicado. El balance es complicado y es fácil caer en que algún item/arma quede inservible o sea demasiado poderoso.
- La propuesta del desarrollo inicial se ha ido modificando con el tiempo según los avances y cambios. De manera que inicialmente el mapeado iba a ser estático y se realizó la mejora a un sistema de mapeado generado en tiempo de juego durante las primeras etapas. Cambios de armamentos tras verse que eran poco efectivos o carecían de sentido, y la inclusión de tipos de enemigos o mecánicas para hacer más atractivo el juego. Todos estos cambios se hacían tras realizar pequeñas demos jugables durante el desarrollo, lo que nos lleva a la siguiente conclusión.
- El uso de una metodología ágil que permite realizar pruebas en cada evolutivo del juego fue lo que produjo que se facilitaran los cambios para mejorar y adaptar el juego, frente a una metodología más estática.
- Durante el desarrollo he adquirido conocimientos de Unity de manera más avanzada al encontrarme problemas o planteamientos nuevos, como el uso de ciertas mecánicas nuevas.
- He mejorado la capacidad de generar una arquitectura reutilizable (como el sistema de armas) y fácilmente ampliable, de manera que en cierto modo el sistema generado sirve como base para poder desarrollar roguelikes modificando el sistema base implementado.

- La planificación inicial, aunque es viable, la modificación del sistema tras cada prueba alteró dicha planificación, y hubo que adaptarse a los cambios.

7.2 Objetivos y reflexión

El objetivo de la práctica era un juego con un alcance definido para ser un producto mínimo viable y con la suficiente variedad para que pudiera ser jugable durante 1 o 2 horas.

Aunque la capacidad de realizar niveles aleatorios genera mucha rejugabilidad en el juego y el uso de armas desbloqueables contribuye, el juego necesita de variedad de salas y enemigos en el diseño, sin embargo el sistema está preparado para ello.

Las pruebas de nivelación de las armas/enemigos se han realizado en entornos controlados e intentando mantener una dificultad elevada para mantener al jugador frente a un constante reto. Además de que el uso del guardado de la experiencia obtenida es permanente entre partidas, contribuye a que la muerte del jugador sea parte del proceso donde puede intentarlo tantas veces como quiera.

Aunque el sistema soporta el uso de joystick para la mayoría de sus controles, no se ha trasladado el funcionamiento del uso del ratón, siendo la parte más complicada la de alternar entre 2 tipos de controles muy distintos una misma mecánica.

Por ello la conclusión sobre los objetivos ha sido positiva, llegando a tener un juego base con un acabado positivo y considerándose un juego pequeño pero completo. Con la posibilidad de extender la variedad indicada previamente (de enemigos y tipos de sala) y algunos balanceos para alargar la vida del juego, se puede considerar su publicación como juego indie.

7.3 Análisis de la planificación

Como se ha comentado previamente, la planificación inicial fue modificada durante el desarrollo del proyecto. Las principales causas de las modificaciones fueron las siguientes:

- Aumento en la funcionalidad/tarea implementada: Durante el desarrollo algunas funcionalidades se expandieron. Un ejemplo fue el menú inicial, que pasó de ser un menú 2D a un entorno 3D interactivo mediante el ratón o la modificación de la interfaz de compra.
- Modificación de la funcionalidad o tarea: Algunas ideas de mecánicas o planteamientos no eran viables una vez puestas en práctica, como

armas que se desecharon o mecánicas en el mapa que no resultaron ser viables programáticamente.
- El mayor cambio fue la modificación de la generación de mapas estáticos a un generador de niveles. Este cambio aunque se estimó en la misma cantidad de tiempo que la generación de los mapas, a la hora de realizar el cambio, finalmente llevó más tiempo del estimado para que el sistema funcionara de manera correcta.
- Como se indicaba en las conclusiones iniciales, el uso de pequeñas versiones jugables donde se realizaban pruebas permitía reajustar las estimaciones

7.4 Futuras líneas de trabajo

Debido a tiempos y modificaciones en las estimaciones, algunas ideas que iban para el producto quedaron desechadas, por cuestión de plazos. A continuación se detallan:

- Sistema de opciones: El menú inicial tiene algunas animaciones ocultas para moverse a otra sala, la idea era tener un menú de opciones de audio donde poder ajustar los niveles de música, sonido de manera independiente, además de ajustar opciones de postprocesado para activarlas o desactivarlas una a una.
- Multiidioma: Gracias al uso de plugins como Lean Localization, el juego puede ser fácilmente traducido a múltiples idiomas, trasladando los textos a ficheros csv.
- Niveles predefinidos: Para complementar el sistema de niveles generados, existía la idea de generar mapas manualmente unificando las salas y teniendo mapas generados de manera estática, que supusieran retos para el jugador, como niveles adicionales.
- Variedad de enemigos: existen varios enemigos que no pudieron ser incluidos en el juego por plazo, como un tipo de enemigo inmortal que pudiera moverse entre salas, acosando siempre al jugador, o enemigos que tuvieran variedad de armamento del jugador, como sistemas de defensa con torretas armadas con rifle láser.

Algunas ideas más a largo plazo fueron anotadas para un producto final estable con posibilidad de ser comercializado:

- Tabla de puntuación: la posibilidad de agregar un modo arcade donde el jugador fuera contrareloj y pudiera visualizar su puntuación para compararla con otros jugadores.
- Modos alternativos: modo con “permadeath”, donde el jugador perdiera todos los avances en desbloqueables o experiencia al morir, de manera que no permitiera reintentos.

- Equipamiento: la posibilidad de además de utilizar armas, llevar equipamiento que añadiera variedad, como el uso de señuelos, armas automáticas que pudiera emplazar, mapas que enseñaran una visión global..
- Capacidad de mejora en las armas: invertir la experiencia en mejorar las estadísticas de las armas o subirlas de categoría, alargando la vida del juego.
- Sistema de aletoriedad en el armamento: al igual que juegos como Diablo o Borderlands, las armas que se recogen por el escenario pueden tener categorías según sus estadísticas

8. Glosario

- GameObject: objeto base de Unity. Se llenan de componentes que agregan funcionalidad.
- C#: Lenguaje de programación diseñado por Microsoft.
- Rigidbody: componente de Unity que permite la aplicación de físicas al GameObject al que se aplica.
- Git: sistema de control de versiones Open Source.
- Asset: ítem utilizado por el motor del videojuego.
- Bundle: conjunto de Assets.
- Humble Bundle: página que ofrece contenidos en paquetes de contenido.

9. Bibliografía

1. El videojuego facturó 1.479 millones de euros, con una base superior a los 15 millones de usuarios, en 2019 - Madrid, 19 Abril de 2019 http://www.aevi.org.es/anuario-aevi-2019/ (05/11/20)
2. Scrum Methodology definition - Diego Calvo, 7 Abril de 2018 https://www.diegocalvo.es/en/methodology-scrum-methodology-agil/scrum-methodology/ (05/11/20)
3. Roguelike, Roguelikelike, Roguelikelikelike, Or Roque - Alice O'Connor, 14 de Septiembre de 2016 https://www.rockpapershotgun.com/2016/09/14/what-is-a-roguelike/ (07/11/20)
4. I Hate Roguelikes, And So Should You! - R. Hunter Gough, 17 de Enero de 2020 https://www.gamasutra.com/blogs/RHunterGough/20200117/355794/I_Hate_Roguelikes_And_So_Should_You.php (07/11/20)
5. The Binding of Isaac. How One Roguelike Has Outlived the Competition - Ed Moreno, 30 Abril de 2018, https://goombastomp.com/binding-of-isaac-history/ (10/11/20)
6. Análisis de Nuclear Throne- R. Márquez, 7 de Diciembre de 2015 https://www.vidaextra.com/analisis/nuclear-throne-analisis (10/11/20)
7. Wot I Think: Enter the Gungeon - John Walker, 5 Abril de 2016 https://www.rockpapershotgun.com/2016/04/05/enter-the-gungeon-review/ (10/11/20)
8. Why Spelunky is the most important game of the decade - Chris Plante, 8 Noviembre de 2019 https://www.polygon.com/2019/11/8/20953893/spelunky-most-important-game-of-the-decade-2010 (10/11/20)
9. HADES is the perfect game of the Percy Jackson Generation - W.C. Hoag, 15 de Octubre de 2020 https://nerdist.com/article/hades-video-game-gen-z-greek-mythology/ (20/11/20)
10. Dead Cells succeeds where its competitors fail - Julie Nuncy, 16 de Agosto de 2016 https://www.wired.com/story/dead-cells-roguelike/ (20/11/20)
11. Crypt of the Necrodancer review - a truly engaging genre love-in, Alex Hern, 14 Mayo de 2015 https://www.theguardian.com/technology/2015/may/14/crypt-of-the-necrodancer-review-a-truly-engaging-genre-love-in (20/11/20)
12. Análisis de BPM: Bullets Per Minute - Disparos y rock en las mazmorras procedurales de Asgard - Pablo Casado, 28 Septiembre de 2020

https://www.eurogamer.es/articles/bpm-bullets-per-minute-analisis (21/11/20)
13. Void Bastards is a gorgeous sandbo of death and chaos - Danielle Riendau, 31 Mayo de 2019, https://www.vice.com/en/article/pajq58/void-bastards-bioshock-shooter (21/11/20)
14. Never Go Home - https://www.indiedb.com/games/never-go-home o https://www.goodnightgames.com/never-go-home (04/10/20)
15. Universal Render Pipeline- Unity, https://unity.com/es/srp/universal-render-pipeline (01/11/20)
16. HDRP vs LWRP, Using PBR Rocks, Reddit - Usuario ANGRY_MESH, 22 de Mayo de 2019 https://www.reddit.com/r/Unity3D/comments/brojim/heres_a_comparison_image_between_hdrp_and_lwrp/ (01/11/20)
17. NavMeshComponents, https://github.com/Unity-Technologies/NavMeshComponents (01/12/20)