UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Realización y desarrollo de un videojuego 2D utilizando la plataforma de Unity

Trabajo Fin de Grado

Grado en Tecnología Digital y Multimedia

AUTOR/A: Roselló Simó, Guillem

Tutor/a: Cerdá Boluda, Joaquín

CURSO ACADÉMICO: 2023/2024

Resumen

Los estudios del Grado de Tecnología Digital y Multimedia permiten, entre otras competencias, conocer a fondo el mundo del diseño, desarrollo y creación de mundos digitales desde cero. Para abordar el proyecto de **realización de un videojuego 2D utilizando la plataforma de Unity**, el enfoque ha sido construir una idea original desarrollando habilidades creativas y técnicas que, al mismo tiempo, pueda brindar una experiencia de juego interesante para los usuarios. La primera versión del juego consta de un personaje con su propia historia y contexto, que explorará un mundo decadente con varios desafíos.

El **proceso de desarrollo** ha sido complejo ya que ha habido que **asumir múltiples roles**: la creación del guion, diseño de personajes y fondos, programación de la lógica de juego y composición de la banda sonora. Cada etapa obliga a desplegar diferentes habilidades, tanto técnicas como artísticas.

El resultado final ha sido la base para el posterior desarrollo de un videojuego que reflejará un mundo cohesivo a ser ampliado, así como unas mecánicas y diseños que debería tener la calidad suficiente como para poder ser **publicado y distribuido en varias plataformas**.

Resum

Els estudis del Grau de Tecnologia Digital i Multimèdia permeten, entre altres competències, conèixer a fons el món del disseny, desenvolupament i creació de mons digitals des de zero. Per abordar el projecte de **realització d'un videojoc 2D utilitzant la plataforma Unity**, l'enfocament ha sigut construir una idea original desenvolupant habilitats creatives i tècniques que, al mateix temps, puga brindar una experiència de joc interessant per als usuaris. La primera versió del joc consta d'un personatge amb la seua pròpia història i context, que explorarà un món decadent amb diversos desafiaments.

El **procés de desenvolupament** ha sigut complex ja que ha calgut **assumir múltiples rols**: la creació del guió, disseny de personatges i fons, programació de la lògica de joc i composició de la banda sonora. Cada etapa obliga a desplegar diferents habilitats, tant tècniques com artístiques.

El resultat final ha sigut la base per al posterior desenvolupament d'un videojoc que reflectirà un món cohesiu a ser ampliat, així com unes mecàniques i dissenys que haurien de tindre la qualitat suficient com per a poder ser **publicat i distribuït en diverses plataformes**.

Abstract

The studies of the Digital Technology and Multimedia Degree allow, among other competencies, to thoroughly understand the world of design, development, and creation of digital worlds from scratch. **To approach the project of creating a 2D video game using the Unity platform**, the focus has been on building an original idea by developing creative and technical skills that, at the same time, can provide an interesting gaming experience for users. The first version of the game consists of a character with their own story and context, who will explore a decadent world with various challenges.

The development process has been complex since **multiple roles have had to be assumed**: creating the script, designing characters and backgrounds, programming game logic, and composing the soundtrack. Each stage requires deploying different skills, both technical and artistic.

The final result has been the basis for the subsequent development of a video game that will reflect a cohesive world to be expanded, as well as mechanics and designs that should have sufficient quality to be **published and distributed on various platforms.**

RESUMEN EJECUTIVO

La memoria del TFG del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de la tecnologías digitales y multimedia

CONCEPT (ABET)	CONCEPTO (traducción)	¿Cumple? (S/N)	¿Dónde? (páginas)
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	5
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	S	5-12
1.3. Setting of goals	1.3. Establecimiento de objetivos	S	6, 12
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	S	16-22
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	13-15, 22-39
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	S	50-52
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	40-49, 53-54

Índice

Figuras

Esquemas

Tablas

Código y scripts

Capítulo 1. Introducción

1.1 Contexto del proyecto

El **mundo de los videojuegos** es una de las ramas que permiten ser exploradas por los estudiantes de Grado en Tecnología Digital y Multimedia brindando la **oportunidad de adentrarse en esa materia con los suficientes conocimientos como para lograr un producto aceptable y de calidad**.

Además, dentro del mundo de los videojuegos, en los últimos tiempos, **el desarrollo independiente se encuentra en una etapa un resurgimiento** [1]. Según un informe de VG *Insights*, los juegos *indie* representaron el 31% de los ingresos totales en Steam en 2023, un aumento significativo desde el 25% en 2018. y por tanto, invertir esfuerzos en el desarrollo, puede tener una buena acogida entre los jugadores habituales y es por ello que ofrece una **buena oportunidad** para embarcarse en este tipo de trabajo.

De ahí surge la idea de realizar un proyecto con el objetivo principal de demostrar las habilidades creativas y técnicas adquiridas durante la formación académica en diseño y desarrollo de videojuegos.

De todas las opciones posibles, se opta por el diseño de un juego centrado en la **habilidad y la gestión de recompensas**. La idea es implementar una mecánica de controlar a un personaje cuyo objetivo sea derrotar a diferentes oponentes, cada vez más complejos, para obtener respuestas a un planteamiento inicial y existencial del personaje. Es un formato tradicional, pero se trabaja a su vez con cierto contenido narrativo (*lore*) para darle un toque conceptual y diferencial respecto al resto de juegos.

Características del proyecto:

- Desarrollo con **gráficos 2D**, poniendo énfasis en crear un estilo visual especial.
- Interfaz de usuario **intuitiva y fácil de usar**, siguiendo los principios de diseño de interacción.
- Motor de juego para gestionar las reglas, la lógica y sistema de control de movimientos y comportamiento de los personajes.
- **Aspecto creativo** reflejado en el diseño de los personajes, los escenarios, música y las mecánicas.
- **Planificación** del desarrollo usando las lecciones aprendidas en la rama de gestión de proyectos.
- Producto final de estilo **independiente** (*indie*).

1.2 Contenido del trabajo

Los condicionantes para desarrollar este trabajo para circunscribirse a las exigencias de la industria son principalmente:

- **Proceso de diseño artístico** junto con el **análisis funcional, diseño y desarrollo** (programación).
- Enumeración de las **principales fases del GDD** (*Game Design Document*) detallando y describiendo todos los aspectos del videojuego.

Capítulo 2. Objetivo

El **objetivo principal** de este trabajo es **documentar el proceso de desarrollo de un videojuego 2D** original, aplicando los principios del *Game Design Document* (GDD), y **presentar el producto final resultante**. Este enfoque implica elaborar y utilizar un GDD como herramienta central para guiar todo el proceso de desarrollo del videojuego, describiendo cómo evoluciona a lo largo del proyecto.

En el transcurso del trabajo, se detallarán las decisiones de diseño tomadas durante el desarrollo, abarcando los elementos clave del GDD. Esto recogerá los siguientes aspectos:

1. Título, presentación, visión general y concepto del juego.
2. Planificación.
3. Clasificación general.
4. Mecánicas del juego (*gameplay*).
5. Historia (*lore*).
6. Niveles del juego.
7. Personajes.
8. Elementos del juego.
9. Interacción.
10. Otros aspectos.

Previamente al desarrollo del GDD, se detallarán algunos aspectos previos como son:

1. Análisis previo y estudio de mercado.
2. Planificación: Metodología y herramientas usadas.
3. Creación de recursos.
4. Implementación (detalles de programación).

Finalmente, se presentará el videojuego terminado, contrastando el resultado final con las especificaciones originales del GDD y verificando el cumplimiento de los objetivos iniciales descritos en el GDD. Esto permitirá un análisis crítico del proceso de desarrollo y del producto final, identificando cómo el uso del GDD influyó en la toma de decisiones y en la cohesión del juego.

Como **objetivos complementarios**, se busca demostrar la **aplicación práctica de los conocimientos adquiridos** en el Grado de Tecnología Digital y Multimedia en la creación de un GDD completo y funcional. También se evaluará la **viabilidad del videojuego** para su distribución en plataformas comerciales, basándose en los elementos definidos en el GDD.

Al apoyar el proceso en el uso del GDD, de alguna manera se alinea el desarrollo realizado con las prácticas profesionales de la industria y permite comprender cómo se crea un videojuego desde su concepción hasta su finalización.

Capítulo 3. Análisis previo

3.1 Plataformas de publicación

El proyecto se plantea, en principio, para **plataforma de ordenadores PC** y como **juego 2D** con enfoque de batallas contra jefes y de forma que sea rejugable.

Las **principales plataformas de distribución digital** para este tipo de juegos imponen ciertos requisitos para publicar como desarrollador independiente. Las más importantes son *Steam* o *Epic Games* y *Origin* que tienen ventajas como una gran audiencia de usuarios, pero también desventajas como la alta competencia y la necesidad de pagar comisiones por ventas. Hay otras plataformas especializadas en juegos independientes como *GOG*, *Itch.io* y *Humble Store que* ofrecen un público más escaso pero más interesado en este tipo de juegos.

En concreto, plataformas como *Steam* y *Epic Games* se quedan con una parte de los ingresos por ventas, generalmente entre el 12-30% [2] y *Steam* también cobra una cuota de inscripción de $100 que se recupera una vez que el juego ha vendido $1,000. Las plataformas como *GOG*, *Itch.io*, *Humble Store* y *Game Jolt* suelen tener mejores condiciones.

Otro aspecto es la **promoción y marketing** que podría implicar crear una página web, mucho movimiento en redes sociales, etc. que puede requerir una inversión muy alta de tiempo y dinero. Una posible estrategia para promocionar un juego es acudir a cierto tipo de **generador de contenidos** que puede ser una buena forma de lanzamiento [3].

Para poder publicar, primero se ha de tener un producto desarrollado y terminado, que cumpla con los requisitos técnicos de la plataforma en cuanto a formatos, resoluciones y compatibilidad de sistemas operativos, **crear una cuenta de desarrollador** en la plataforma, preparar imágenes, videos y descripciones para presentar el juego y por último se ha de definir el precio y modelo de pago (suscripción, compra única, etc.).

Después, la plataforma realizará una revisión y verificará que se cumplan con todos los requisitos antes de aprobar la publicación.

3.2 Estudio de mercado

Analizando el mercado de este tipo, hay varios ejemplos populares como "*Enter the Gungeon*", "*Dead Cells*", "*Binding of Isaac*", "*Hades*", "Hollow Knight" y "Blasphemous".

Los jugadores a los que se dirige este tipo de videojuegos suelen ser jóvenes que disponen de acceso a las diferentes plataformas mencionadas. Los estilos gráficos retro y *pixelados* pueden resultar muy atractivos para este público y sus edades giran en torno a 16 y 35 años.

Los precios de este tipo de productos son a menudo más asequibles y de pago único que otros productos más elaborados y con empresas más potentes detrás.

3.2.1 Enter the Gungeon

Es un juego con características de disparos, *bullet hell*[1] y *roguelike*[2] que trata de un grupo de personajes que buscan llegar al fondo del *Gungeon*, un laberinto lleno de trampas y enemigos, con el objetivo de encontrar "la pistola que puede matar el pasado" y así redimir sus pecados.

1 Grandes cantidades de proyectiles y bombas a esquivar

2 Niveles aleatorios y con posibilidades de pérdida de progreso al morir

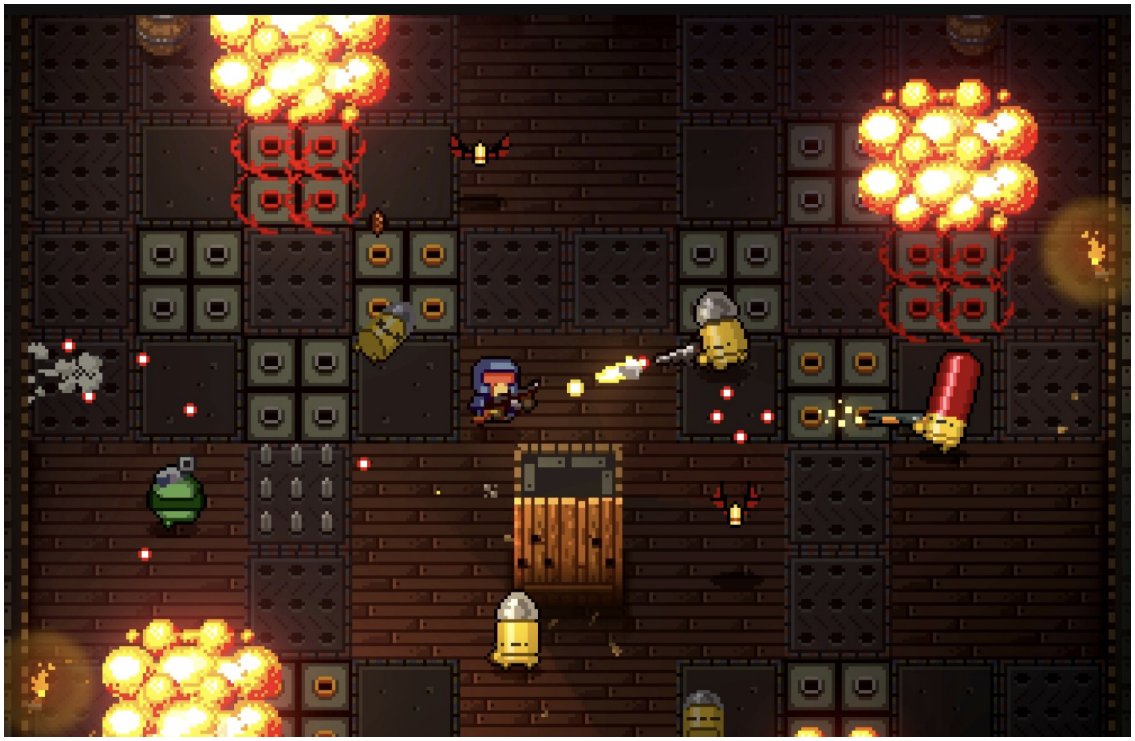


Figura 1: Enter the Gungeon

Cada piso del *Gungeon* es generado aleatoriamente, lo que hace que cada partida sea única.

El juego, publicado entre 2016 y 2017, vendió más de 200.000 copias en la primera semana en todas las plataformas (las ventas en *Steam* representaron el 75%) [4] y en 2020 alcanzó la cifra de **más de 3 millones de copias** a través de plataformas como *Steam, PlayStation*, *Xbox* y *Nintendo Switch* [5].

3.2.2 Dead Cells

Dead Cells es un juego no lineal de plataformas en el que un prisionero reencarnado en un cúmulo de células ha de escapar de una isla maldita llena de monstruos. Cada partida comienza desde el principio cuando mueres, pero puedes obtener mejoras permanentes entre los distintos intentos. Hay gran variedad de armas, habilidades y trampas.

A medida que se progresa en el juego se recopilan "células" de los enemigos derrotados, que se usan para comprar mejoras.

En junio de 2023, el juego había alcanzado los **10 millones de unidades vendidas a nivel global** [6] ofreciendo en ese tiempo 34 actualizaciones gratuitas y siendo el 60% de las ventas para plataforma de PC. El juego se lanzó anticipado en 2017 con lanzamiento final en 2018, estando disponible en múltiples plataformas como ordenador, *Nintendo Switch*, *PlayStation*, *Xbox*, *iOS* y *Android*. *Motion Twin* (desarrolladores) van a seguir ofreciendo actualizaciones y contenidos hasta 2025, lo que afianza la fidelidad de sus usuarios.

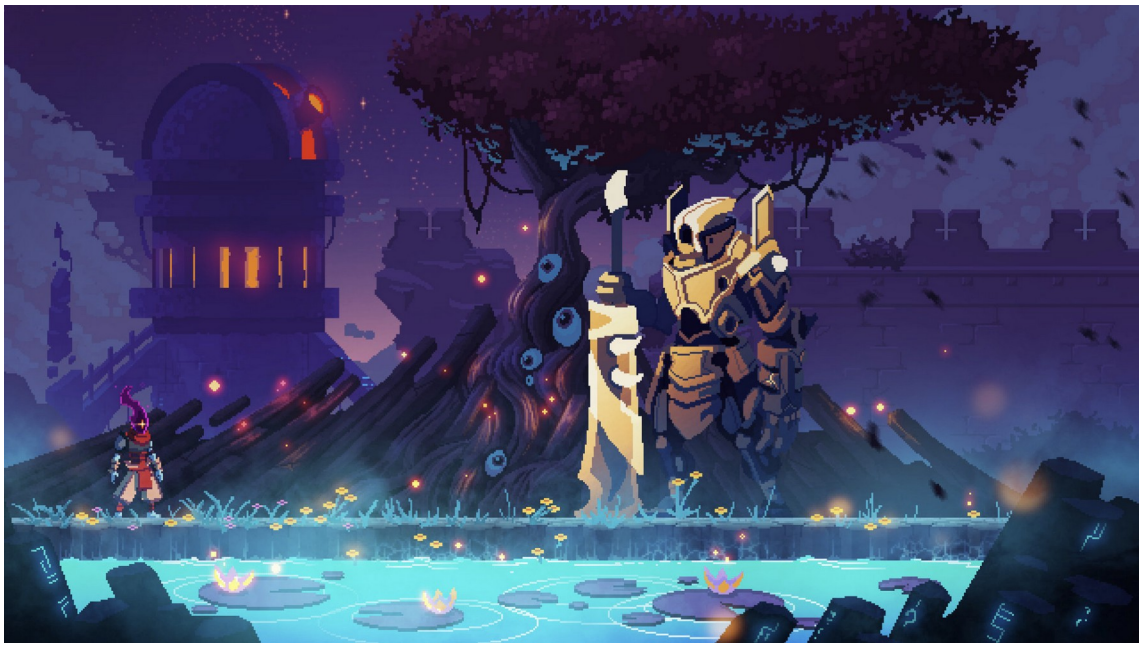


Figura 2: Dead Cells

3.2.3 The binding of Isaac

The Binding of Isaac es un juego de acción y aventura con elementos *roguelike*, desarrollado por *Edmund McMillen* y *Florian Himsl*, lanzado originalmente en 2011 para ordenador y que posteriormente se plataformó para Mac, Linux, consolas y dispositivos móviles.

El juego ha tenido una buena aceptación por parte de los usuarios y por la crítica ya que su trasfondo e historia tiene ciertos elementos que lo hacen muy especial.

Figura 3: Binding of Isaac

Respecto a su comercialización, según informó el propio *McMillen* en julio de 2014, se habían vendido **más de 3 millones de copias del juego** [7], estableciendo un buen indicador de que tuvo un gran éxito entre los usuarios en los tres primeros años.

Además de la versión original, se lanzaron varias expansiones y remakes, como *The Binding of Isaac: Rebirth* en ese año 2014, que mejoró la jugabilidad, los gráficos y añadió nuevos contenidos, con lanzamientos para múltiples plataformas como *PlayStation*, *Xbox* y *Nintendo Switch* entre otras [8].

3.2.4 Hades

Otro videojuego de acción y *roguelike* bastante exitoso y del estilo del desarrollado es *Hades*, publicado por *Supergiant Games*. Ambientado en la mitología griega, el juego usa el personaje de *Zagreo*, hijo de *Hades*, que ha de escapar del Inframundo. Tiene un estilo artístico inspirado en el cómic y banda sonora muy interesante, aspecto que *Supergiant Games* cuida especialmente. Su narrativa es profunda y muy bien escrita, con buenos diálogos que permiten conocer a fondo a los personajes mitológicos del Inframundo. Las mecánicas son las típicas de juego *roguelike*: cada vez es una experiencia diferente, con diferentes enemigos y desafíos.

La estrategia de comercialización se basó en el acceso previo al juego para pulirlo y mejorarlo hasta conseguir un producto robusto. Según datos del propio estudio, ha superado el millón de copias vendidas en total, contando con las ventas en período de acceso anticipado. En sólo unos pocos días, los usuarios adquirieron 300.000 copias que, unidas a las 700.000 de acceso anticipado, hacen **un total de 1 millón de copias** [9].

3.2.5 Hollow Knight

Hollow Knight es un videojuego de estilo *metroidvania*[3] desarrollado por el estudio independiente *Team Cherry* (de tres personas) que sumerge al jugador en el abandonado y peligroso reino de *Hallownest*, hogar de criaturas con aspecto de insectos.

Figura 4: Hollow Knight

3 Los juegos *metroidvania* se caracterizan por un mundo no lineal e interconectado, donde el jugador explora y adquiere nuevas habilidades para progresar, con una estética atractiva, combates desafiantes y jefes poderosos que ponen a prueba las habilidades y estrategia de los jugadores.

El personaje es un caballero que debe adentrarse en las profundidades de ese mundo para descubrir sus secretos. Obtiene nuevas habilidades y poderes que le permiten acceder a áreas previamente inaccesibles.

El diseño es muy variado y estéticamente potente y la banda sonora compuesta por *Christopher Larkin* confiere al juego una singular atmósfera. Hay batallas contra jefes, algunas de ellas muy difíciles de superar.

En términos de comercialización, *Hollow Knight* fue lanzado en 2017 para ordenador y posteriormente llegó a otras plataformas como *Nintendo Switch*, *PlayStation 4* y *Xbox One*. Según datos oficiales, ha vendido **más de 2,8, millones de copias a nivel mundial** [10], convirtiéndose en uno de los juegos independientes más exitosos de los últimos años.

Un punto interesante es la **financiación** de este proyecto. Se hizo a través de una **campaña de *Kickstarter*** que recaudó 57.000 dólares [11].

3.3 Entrega del producto para plataformas

Cada plataforma tiene su propia forma de admitir y gestionar sus suscripciones de desarrolladores. Para este punto, se pone **por ejemplo cómo hay que actuar respecto a *Steam***.

1. Registro en *Steamworks*: Se necesita crear una cuenta de *Steamworks* y pagar la tarifa de registro única.
2. Configuración de la aplicación: Se completa toda la información requerida sobre el juego como es el título, descripción, capturas de pantalla, imágenes y vídeos promocionales.
3. Subida del contenido, siguiendo requerimientos técnicos. Comprobar compatibilidad (*Windows*, *MacOS*, *Linux*).
4. Distribución a través de *SteamPipe*: La herramienta que *Steam* proporciona para la distribución de contenido y permite subir y gestionar los archivos del juego.
5. Archivos del juego: Deben estar bien organizados y empaquetados, incluyendo todos los recursos necesarios como imágenes, audio, etc.
6. Pruebas: Antes de lanzar el juego, se requiere comprobar que todo funcione correctamente.
7. Revisión y aprobación: Se remite el juego para su revisión y *Steam* comprobará el contenido para asegurarse de que cumple con sus normas.

Figura 5: Clasificación PEGI

Como acción adicional, es necesario registrar el juego para su distribución comercial según la clasificación PEGI[4] para videojuegos y podremos utilizar todo el material confeccionado en el punto 2 anterior y enviarlo (rellenando el formulario correspondiente) a través del sitio web https://pegi.info/es permitiendo que el Comité de PEGI pueda evaluar adecuadamente el contenido determinando la clasificación apropiada, como es la edad (3, 7, 12, 16 hasta 18 años) y advertencias sobre contenido específico como violencia, drogas, miedo, etc.

4 PEGI: Pan European Game Information

PEGI emite esta clasificación que debe ser incluida dentro de los materiales enviados a la plataforma elegida.

3.4 Público objetivo

El rango de edad principal serían jóvenes de entre 16 y 30 años, con el siguiente perfil:

- Aficionados a los juegos independientes (*indie*).
- Entusiastas de juegos de plataformas 2D, con estilos visuales retro o neo-retro.
- Usuarios que puedan demandar sesiones cortas de ocio.
- Personas que usan ordenadores personales con Windows para sus juegos.
- Interés en la cultura pop.
- Sin restricciones geográficas específicas.
- Usuarios con economías ajustadas.
- Gente interesada en la Ciencia Ficción e historias de crítica política.
- Suscriptores de plataformas digitales de juegos.

3.5 Conclusiones del análisis

Una vez analizados los principales juegos, la estrategia a seguir para poder ofrecer en el mercado el videojuego sería la siguiente:

1. Plantear el proyecto con algunos pequeños avances y vídeos y financiar el desarrollo **iniciando una campaña**, por ejemplo con *Kickstarter* o *Verkami*.
2. **Darse de alta en las plataformas** menos mayoritarias y que apoyan más y mejor a los desarrolladores independientes, aunque el número de usuarios potenciales sea menor.
3. Realizar una **primera versión** y ofrecer el producto de forma gratuita a usuarios para obtener una retroalimentación, con el fin de ir ampliando algunas funcionalidades y depurar los posibles fallos (informados por los propios usuarios).
4. Una vez pasada esta fase, **lanzar el producto** y usar, si es posible, a algunos **creadores de contenidos** para hacer una difusión más extendida. Es la forma más interesante de promocionar un juego.
5. Otro aspecto interesante sería formarse en el mundo de la creación de videojuegos y explorar la industria y los eventos existentes, como por ejemplo, suscribirse a *Xsolla* que ofrece soluciones para que desarrolladores y editores de juegos para monetizar sus títulos, distribución y herramientas de desarrollo y soporte. [15] [16]

Capítulo 4. Planificación

4.1 Metodología utilizada

La metodología usada para el desarrollo de este proyecto se basa en una adaptación de prácticas estudiadas en la asignatura de gestión de proyectos sobre técnicas ágiles de desarrollo con *GitHub* como sistema de control de versiones. Esta aproximación comienza con la planificación y gestión de tareas utilizando un **tablero *Kanban* y diagrama de *Gantt*** (explicado más adelante).

El desarrollo se basa en trabajar usando *Git* como repositorio del proyecto. El control de versiones con Git, se mantiene una sola rama (main) y se realizan *commits* pequeños y frecuentes.

4.2 Cronograma y fases del proyectos

El tablero *Kanban* que se ha usado, ha sido un tablero físico con tarjetas de papel que se han ido moviendo y quitando conforme avanzaba el proyecto. Divide el trabajo en tareas pequeñas y manejables, categorizadas como "Pendientes", "Por hacer", "En progreso" y "Completado" y así de un vistazo se puede ver en que punto se encuentra, sirviendo de guía.

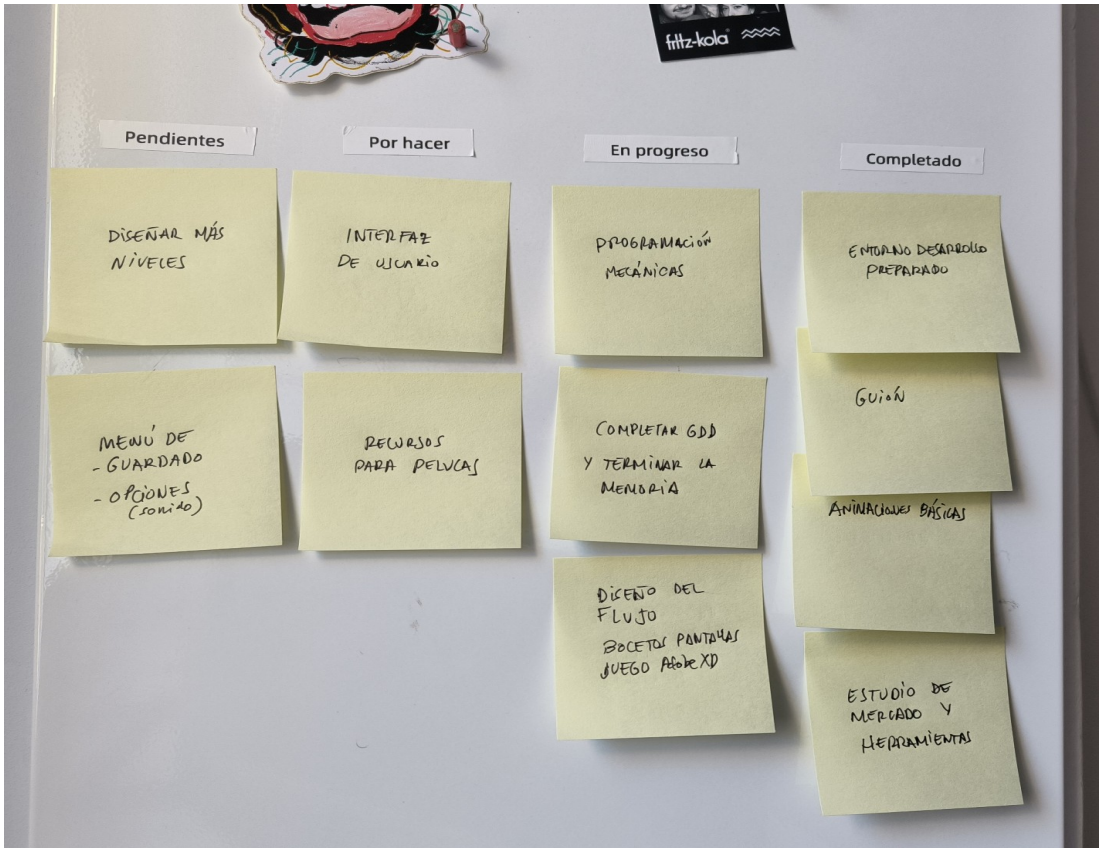


Figura 6: Diagrama de Kanban

En la imagen se aprecia un momento del avance del proyecto.

Al realizar el desarrollo en solitario, la metodología usada simplemente sirve como guía y planificación (no existe colaboración). Se han ido seleccionando las tareas a completar, y tras su completado, se realiza una breve revisión y retrospectiva.

Al inicio del proyecto se estimó la posible duración y para ajustar los tiempos, se ha realizado un diagrama de *Gantt* para poder establecer las fechas de cumplimiento de cada una de las entregas y realizar un seguimiento diario de los avances y retrasos.

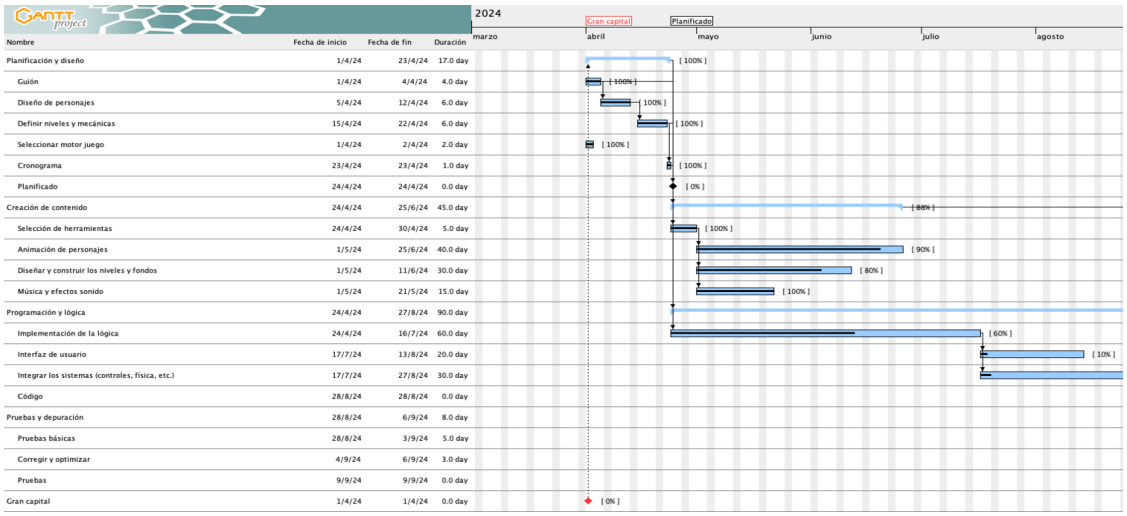


Figura 7: Diagrama de Gantt

4.3 Herramientas de gestión

Para la **gestión de proyectos** de este estilo podría usarse varias utilidades y programas, aunque por la simplicidad sólo se han usado básicamente las dos mencionadas (Kanban y Gantt)

4.3.1 Software de gestión de proyectos

Al tratarse de un trabajo de fin de grado en el que sólo hay un desarrollador **no es necesario el uso de este tipo de software** que permite organizar bien el trabajo y hacer un seguimiento de equipos y colaboradores. No obstante se ha hecho uso del *GanttProject* para realizar diagramas de seguimiento.

- *Jira*
- *Trello*
- *Asana*
- *Microsoft Project*
- *GanttProject*
- *Basecamp*
- *Monday.com*
- *ClickUp*

4.3.2 Herramientas de colaboración

Igualmente, las herramientas colaborativas no son necesarias.

- *Slack*
- *Microsoft Teams*
- *Google Workspace*

4.3.3 Herramientas de diagramación

En este caso sí que se ha hecho uso del *FreeMind* para mapas conceptuales a la hora de desarrollar la narrativa y el guión y *Draw* de *LibreOffice* para diagramas de flujo. Otras opciones podrían haber sido *Microsoft Visio*, *Lucidchart* o *Draw.io*.

4.3.4 Herramientas de control de versiones

Para el proyecto se ha hecho uso muy básico de *GitHub* que es la plataforma de desarrollo que utiliza el sistema de control de versiones Git, para alojar el proyecto.

4.3.5 Herramientas de gestión ágil

Se ha usado un tablero físico *Kanban* para revisar de vez en cuando (cuando las tareas a realizar se amontonaban y había que planificar mejor). Podría haberse usado uno digital (Scrum boards, Kanban boards) pero por la dimensión del proyecto y al no ser colaborativo, no es necesario.

4.3.6 Herramientas de gestión de documentos

Para esta parte se ha usado básicamente *Google Drive* y *LibreOffice*.

Capítulo 5. Diseño artístico

5.1 Arte y Diseño como herramienta narrativa

Una de las partes más esenciales para que un videojuego resulte atractivo y destaque sobre muchos otros es la **dirección artística**. Es esta misma la que suele atraer a potenciales clientes

Para la creación del mundo y su entorno, se ha puesto un gran énfasis a esta dirección artística, adoptando un cierto estilo ligeramente vulgar a la vez que industrial, que pretende sumergir al jugador en el entorno de una "megaciudad", rodeada por grúas, contenedores de cargamento y contaminación. Este entorno urbano solo se rompe una sola vez a la hora de llegar a una zona que se pretende sea un breve respiro de la ciudad.

También, más adelante en el desarrollo, se aspira a llevar una **dirección de narrativa visual**, es decir, transmitir la historia a los usuarios mediante ilustraciones o pistas visuales escondidas en el entorno, premiando a jugadores curiosos que deseen saber más acerca de la narración.

5.2 Creación de recursos gráficos

5.2.1 Diferentes recursos empleados en la primera versión

Se han creado una gran variedad de recursos, como pueden ser *sprites*[5] de jugador y enemigos:

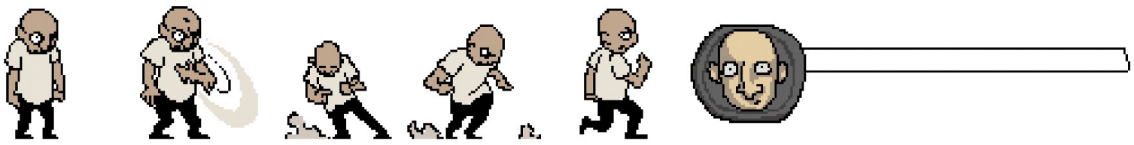


Figura 8: Sprites de Norman y vida

También se han creado fondos para cada uno de los niveles, en total 3 combates, y un cuarto fondo para lo que sería en una versión futura el combate final, además del fondo del *Hub* principal.

Figura 9: Fondo jefe Árbol

5 *Sprite*: Imagen o animación bidimensional que se utiliza como un elemento gráfico independiente en videojuegos, interfaces de usuario y otros entornos digitales. Los *sprites* pueden representar personajes, objetos, efectos visuales o cualquier otro elemento que necesite moverse o animarse dentro de una escena.

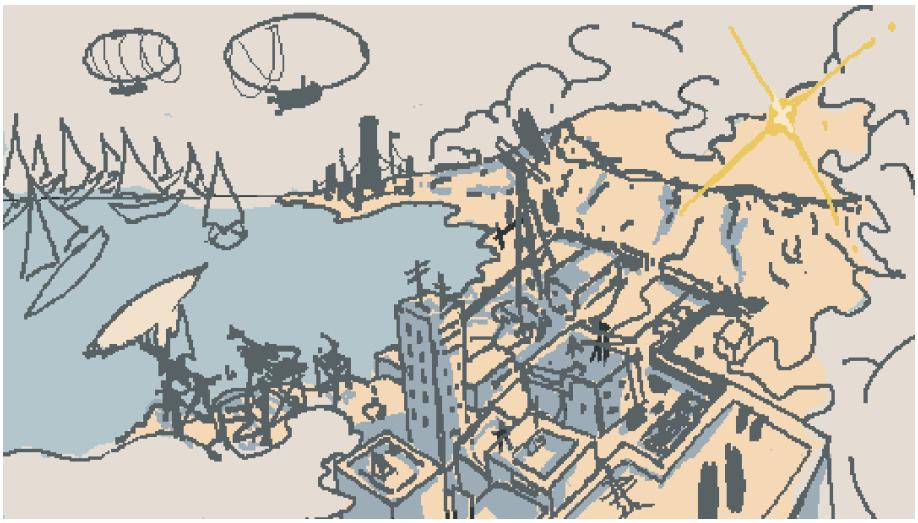


Figura 10: Fondo jefe UGT

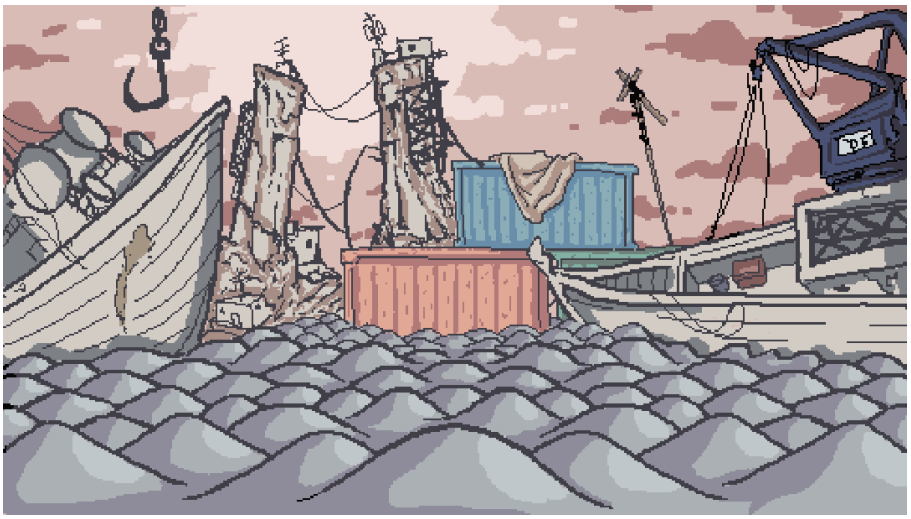


Figura 11: Fondo jefe Rose

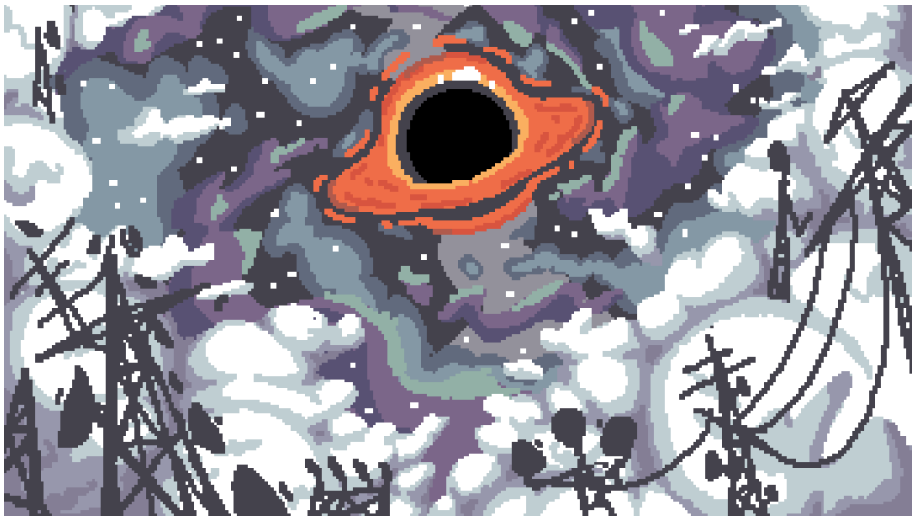


Figura 12: Fondo jefe final

5.2.2 Pantallas del juego

En las siguientes figuras se ven algunas de las pantallas usadas.

En el capítulo que hace referencia al resultado final, pueden verse algunos vídeos del juego en acción.

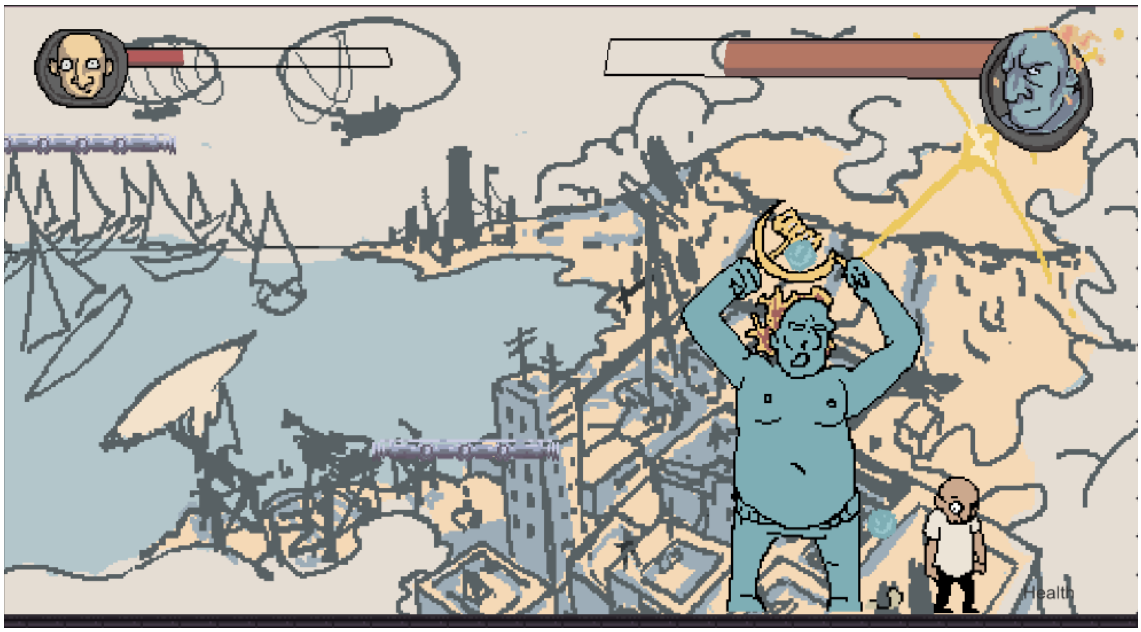


Figura 13: Pantalla del juego (UGT)

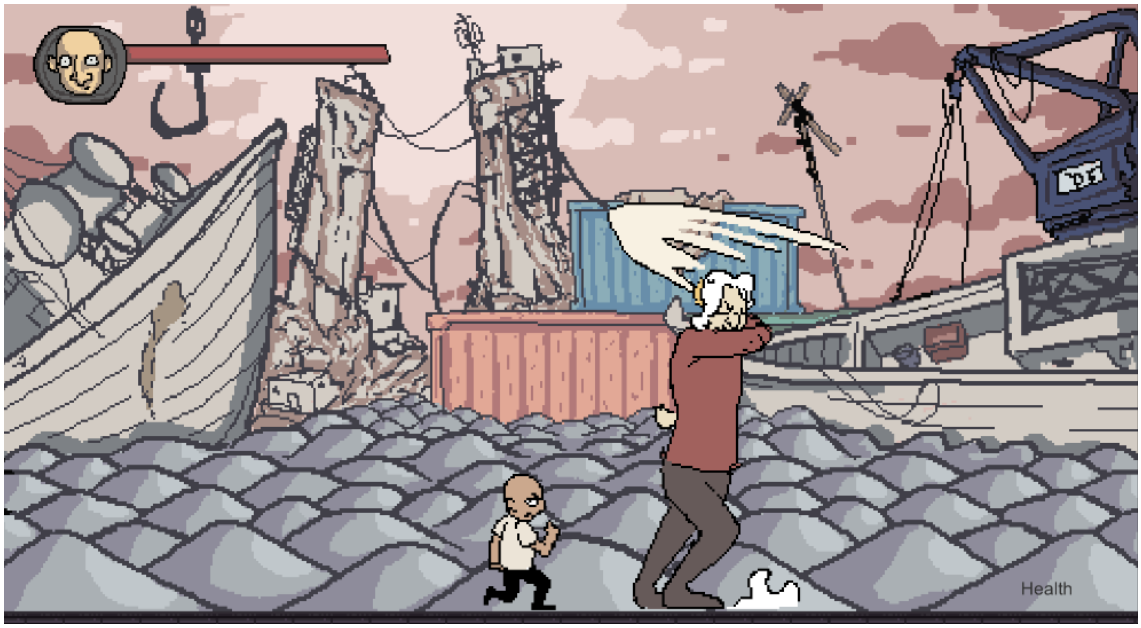


Figura 14: Pantalla del juego (ROSE)

5.2.3 Otros recursos

Para los recursos de la sección de plataformas entre niveles, también se han creado otros recursos como *tilemaps* o simples *assets* de decoración, utilidad o para efectos de diferentes ataques.

Figura 15: Recursos gráficos (1)

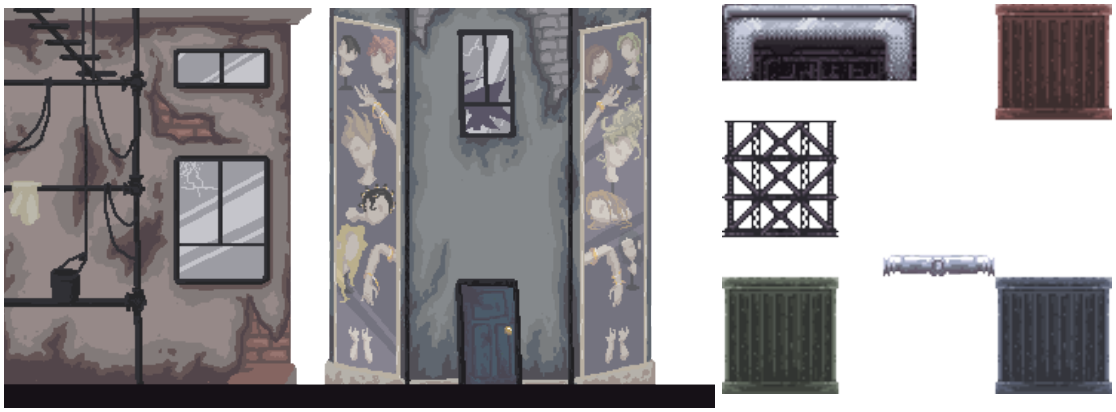


Figura 16: Recursos gráficos (2)

5.2.4 Uso de Adobe Photoshop

Para la realización de la gran mayoría de **recursos artísticos** se ha empleado el *Adobe Photoshop,* sobre el cual se realizaban los principales bocetos y estilos de cada uno de los ambientes creados para la primera versión del juego.

La gran mayoría de imágenes han sido diseñadas y trabajadas bajo un solo lienzo de *Photoshop*, al que se le iba integrando elementos para mantener una cohesión artística, y paletas concretas de colores. Muchos de estos gráficos posteriormente se exportarían y moverían a otros programas en caso de requerir la animación del mismo.

Todos los recursos artísticos han sido creados desde cero, para adecuarse con la visión y estilo del mundo que se presenta y crear un ambiente cohesivo y creíble.

5.2.5 Uso de Aseprite

Los **recursos** que necesitaban ser **animados** se han trabajado en *Aseprite*[6], un editor de gráficos rasterizados, diseñado concretamente para la fácil elaboración de *sprites,* que ofrece una gran variedad de funciones que facilitan con creces el trabajo de animación y adaptabilidad a la hora de trabajar con movimiento.

Los *sprites* se animaban usando esta aplicación, se exportaban y gracias a las opciones de exportación era muy fácil importarlas en *Unity* para posteriormente trabajar con ellas en el Animador de ese programa e integrarlas en el juego.

6 Página web de *Aseprite*: https://www.aseprite.org/

Figura 17: Transformación de Rose usando Aseprite

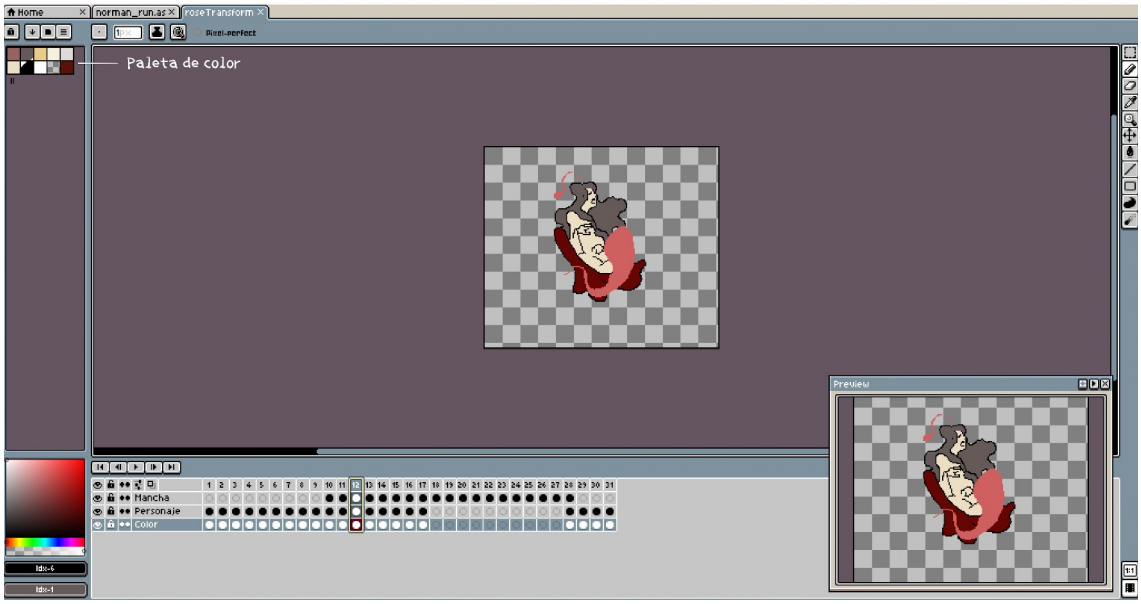


Figura 18: Entorno de trabajo con Aseprite

En este enlace se puede ver un ejemplo de los *sprites* generados.

https://youtu.be/t7pQMQ-il1o

5.3 Creación de recursos sonoros

Para el videojuego es necesario crear una **banda sonora original** y **efectos de sonido** para las diferentes acciones que se verán. Los formatos usados serán *.ogg*[7] para la música y *.wav*[8] para los efectos.

En el GDD se detalla el catálogo de todos los audios utilizados (música y efectos)

5.3.1 Uso de Garageband (MacOS)

Para poder hacer una música con estilo "retro" ha sido necesario descargarse el complemento de *Garageband* llamado *8-bit Legends* y un *plugin* gratuito conocido como *Magical 8-bit plug 2* que permite reproducir sonidos de onda cuadrada, triangular y ruido (*noise*).

Se han compuesto 6 canciones originales para los diferentes niveles, pantalla de salida, inventarios, créditos y una melodía para las transiciones.

7 OGG: Archivo de audio digital abierto y libre de patentes, desarrollado por la Fundación Xiph.Org

8 WAV: *Waveform Audio File Format*

El formato *.ogg* ha sido el usado para las canciones. En el GDD se encuentran los enlaces a todas las canciones para que puedan ser escuchadas.

Figura 19: Magical 8bit Plug 2

5.3.2 Uso de banco de sonidos

Para efectos especiales se ha usado el banco de sonidos de https://freesound.org/ que tiene casi 650.000 sonidos de libre uso.

En este caso, el formato elegido ha sido el *.wav*.

Como resulta a veces costoso encontrar en *freesound* los efectos sonoros deseados, también se explorará eleven labs como plataforma de IA de generación de sonidos.

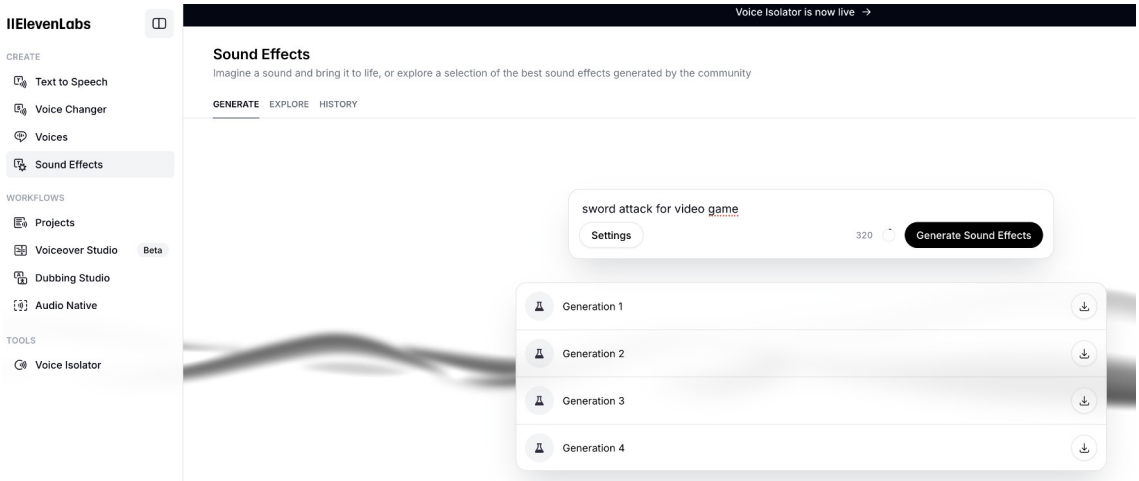


Figura 20: Uso de *elevenlabs* para efectos sonoros

Capítulo 6. Implementación

La implementación se ha realizado considerando las áreas más importantes a desarrollar, dejando de lado aquellas que pueden ser evolucionadas en versiones posteriores del juego y que no pueden ser abordadas en esta primera.

Se explica, en primer lugar, el tipo de juego, las herramientas y el leguaje de programación usado para desarrollar todas las funcionalidades, las clases utilizadas y el comportamiento de cada entidad en el juego, respecto a su entorno y al jugador.

6.1 Introducción a las mecánicas básicas del juego

El juego, en términos generales, se trata de un *roguelite*, es decir, un juego centrado en la rejugabilidad de partidas, en la que aparentemente se empieza desde cero, pero se conserva cierto progreso, como pueden ser mejoras en el personaje o algún tipo de dinero. Dentro de este género, las partidas constarán, primeramente, con una parte de plataformeo, reminiscente a juegos del estilo *metroidvania*, comentados anteriormente, disponiendo de ciertos lugares que solo pueden ser alcanzados tras haber hecho progreso en la historia y haber alcanzado ciertos objetivos, y otra segunda parte centrada en el combate contra jefes, que serán los principales objetivos a derrotar y que permitirán acceder a mejoras permanentes y acompañarán la historia principal del juego. A medida que el jugador avance y vaya superando los retos, más y más fragmentos de la historia se irán recuperando para finalmente mostrar el arco narrativo entero del jugador tras haber repetido la partida varias veces.

6.2 Unity, Arquitectura y diseño del código

Este juego se sostiene bajo el **motor de Unity**. Por lo tanto, los *scripts* se han escrito empleando **C#** y se han usado varias funciones de integración de animaciones usando el mismo motor de Unity, el **Unity Animator**, que, basándose en máquinas de estado y variables de control, permite crear un flujo de animaciones y transiciones que puede también ser accedido mediante interacciones con el código a tiempo real de ejecución.

Para ciertos elementos del juego, se ha dependido más de los mecanismos de integración del editor de Unity y para otros se ha manejado el flujo a través de C#. Es por esto que para la gran mayoría de funcionalidades se ha llevado una forma de integración mixta, intentando aprovechar al máximo cada uno de los sistemas disponibles en el motor de Unity. [21]

Los personajes (enemigos y jugador), constarán principalmente y de forma general de dos clases diferenciadas, heredadas de *MonoBehaviour* de Unity, que determinarán su comportamiento interactuando con los otros elementos del juego.

6.2.1 Salud y vida (Damagable y PlayerHealth)

Estas clases determinarán si un objeto del juego puede ser dañado, es decir, si es registrado como enemigo o a su vez de jugador. Estos *scripts* almacenan la vida máxima que puede llegar a tener una entidad del juego así como otras estadísticas aún no completamente implementadas, como pueden ser la mecánica de puntos vitales y postura, que aún necesitan más desarrollo para su correcta implementación.

Estos comportamientos afectan en todo aquello relacionado con la lógica de vida y muerte así como la progresión en los diferentes jefes. Se emplean mayoritariamente propiedades para almacenar valores y funciones de interacción que registran si una entidad ha sido dañada y cuanto daño ha recibido.

Por ahora, los *scripts* de salud se comportan de forma clásica, es decir, se registra en un único valor y se interacciona con una barra de vida para mostrarla por pantalla, no obstante, en un futuro,

los enemigos tendrán un sistema adicional, con varias barras de vida y el sistema de puntos vitales, que cada vez que se le acabe una barra de vida u otra de postura, se le descontará un punto vital, hasta que finalmente se le acaben los puntos vitales (los enemigos tendrían alrededor de 3 puntos). Este sistema está ya medio implementado, pero se ha decido no usarlo para posteriormente mejorarlo, ya que son necesarios recursos animados y gráficos, que están pendientes de su diseño y creación.

El siguiente código muestra la implementación de barras de vida y postura.

```
[SerializeField]
    private int _vitalPoints;
    public float VitalPoints
    {
        get
        {
            return _vitalPoints;
        } set {
            _vitalPoints = (int)value;
            // Vital Points <= 0 = Dead
            if (_vitalPoints <= 0)
            {
                IsAlive = false;
                // Call death function
            }
// Barra de vida clásica, bajo de 0, se llama a Stun, se le disminuye un punto vital.
private float _health = 100;
    public float Health
    {
        get
        {
            return _health;
        } set {
            _health = value;
            // Health < 0 = Stun
            if (_health <= 0)
            {
                Debug.Log("under 0 health");
                handleStun();
            }
        }
    }
// Barra de postura, cuando se baja a cero, la entidad se aturde, se le disminuye un punto vital dentro de handleStun() y
regenerará una barra de vida
    private float _poise = 100;
    public float Poise
    {
        get
        {
            return _poise;
        } set {
            _poise = value;
            // Poise < 0 = Stun
            if (_poise <= 0)
            {
```

```
            Debug.Log("under 0 poise");
            handleStun();
        }
    }
}
```

Código 1: Implementación barras vida y postura

En las siguientes lineas se mostrará la función de manejar los *Stuns* y otra de registrar golpes por parte del jugador:

```
public IEnumerator stunned(float duration)
    {
        Debug.Log("Stunned STUN");
        float elapsedTime = 0f;
        while (elapsedTime < duration)
        {
            // Check the cancel condition
            if (hitWhileStun)
            {

                Debug.Log("Exiting Stun!");
                // -1 Vital Point, no longer stunned
                VitalPoints -= 1;
                isInvincible = true;
                IsStunned = false;
                // Reset health
                _health = MaxHealth;
                hitWhileStun = false;
                Poise = MaxPoise;
                yield break; // Exit the coroutine
            }
            yield return null;
            elapsedTime += Time.deltaTime;
        }
        // Reset health out of time %
        // if not acted while stunned recover percentage of current health bar
        _health = MaxHealth*percentRecovered;
        Debug.Log("Health reset");
        IsStunned = false;
        yield return null;
}
```

Código 2: Manejo de *Stuns* y golpes

Esta función registrará los golpes por parte del jugador, filtrando los golpes cuando el enemigo está aturdido y cuando no. Dependiendo de cada uno, se ejecutará un código diferente.

```
public void OnHit(float damage,float poiseDamage)
    {
        Debug.Log("Hit " + IsStunned);
        if(IsStunned && IsAlive && !isInvincible)
        {
            hitWhileStun = true;
            Debug.Log("Hit While Stun");
            VitalPoints -= 1;
```

```
    } else if (IsAlive && !isInvincible) {
        Debug.Log("Hit for: "+damage);
        Debug.Log("Hit for poise: " + poiseDamage);
        // Normal attack, deals health damage and poise damage
        Poise -= poiseDamage;
        Health -= damage;
        isInvincible = true;
        healthBar.SetBossHealth(Health);
    }
}
```

Código 3: Función de ataque del jugador

Aquí se le pasa al enemigo el daño infligido por el jugador, que constará de dos tipos de daño, el de postura y el de vida. En caso de estar aturdido, se le descontará un punto de vida.

6.2.2 Comportamiento general, script de Behaviour

Este hace referencia a como una entidad reacciona al entorno y sobre todo al jugador. Cada jefe en el juego dispone de uno. Estos *scripts* se encargan de coordinar el comportamiento del jefe y constan con funciones que, basándose en objetos, como por ejemplo *colliders* que se tratan como *triggers* registrando así cuando un jugador entra en el rango de ataque de un enemigo, o también cuando sale, así como medidas de elección de patrones de ataque.

Se han empleado distintos métodos del flujo de comportamiento de los jefes, para así explorar qué ofrecen las diferentes funcionalidades ofertadas en Unity (corrutinas, máquinas de estado, etc). Éstas, que serán detalladas más adelante, constituyen una parte integral en el trabajo desarrollado ya que en su gran parte han sido el objeto de principal atención, dado que el objetivo primero era explorar patrones de ataque y el correcto flujo de los combates.

6.3 Características y funcionalidades programadas

Seguidamente, se analizarán los siguientes sistemas, que han sido los más relevantes a la hora de programar y que representan el foco central en esta primera parte de desarrollo.

6.3.1 Controles del jugador manejables, cómodos y responsivos

El desarrollo de un **control responsivo y cómodo** fue la parte inicial para poder trabajar con el jugador. Por lo tanto, en las primeras versiones de control, se trabajaba principalmente en perfeccionar y **pulir los controles y las físicas del jugador** para que el juego se sintiera fluido y cómodo.

Para la implementación de los controles, se ha usado el **nuevo sistema de *inputs* de Unity** (introducido en 2019), que permite mapear de forma más visual acciones a diferentes controles, que facilita la personalización y adaptabilidad, así como usar un sistema de acciones y eventos, que reduce con creces la complejidad del código y permite un diseño más claro del código. Estas características lo hacen mucho más flexible que el antiguo sistema de *inputs*, que pese a ser más sencillo, en proyectos más grandes puede resultar un poco confuso y difícil de adaptar.

En general, se le ha dado gran importancia a que los controles intenten dar esta sensación de suavidad y responsividad al jugador, ya que gran parte del juego también se realizarán elementos de plataformas clásicos, y para estos movimientos de saltos y equilibrios, es necesario disponer de buenos controles para no desesperar y frustrar al jugador.

Se empezó con funciones más básicas que tan solo movían el elemento *transform* del jugador a ciertas posiciones, ignorando las físicas programadas ya en Unity. No obstante, a la hora de probar el jugador, se sentía un movimiento muy artificial y falso, por lo que seguidamente se optó por otra forma de **mover el jugador: mediante físicas**.

Este nuevo método era más difícil de controlar como se deseaba, porque muchas veces se encontraban problemas a la hora de empezar y terminar movimientos (sensación de ir sobre hielo, resbalarse, saltos muy ligeros, sensación de muy poco peso, movimiento inicial lento y poco responsivo, etc.) y se requerían de muchas fuerzas de control para evitar que el jugador realizara movimientos poco cómodos y manejables. No obstante, con el tiempo y a base de pruebas, se consiguió reducir y concretar el número de parámetros necesarios para conseguir un movimiento más óptimo. Estos fueron los principales parámetros elegidos y programados en PlayerControl.cs:

- **Propiedades de control**, mayoritariamente booleanos como por ejemplo IsAlive, CanMove, IsDashing, IsJumping, OnWall, OnGround, que se encargan de limitar ciertos movimientos dadas ciertas características. Por ejemplo, impedir que el jugador pueda ejecutar un salto si no está tocando el suelo, o restringir el movimiento del mismo cuando está atacando.
- Funciones de movimientos y acciones:
  - Caída y gravedad: Se busca mejorar el salto para afectar a, por ejemplo, si el jugador presiona ligeramente el botón y si lo mantiene apretado, que es muy relevante a la hora de hacerle ver al usuario que tiene control sobre su personaje, así como implementar una cierta velocidad de caída para evitar que el jugador se sienta excesivamente ligero al saltar, teniendo en cuenta el limitar dicha aceleración hacia bajo para que no lo haga excesivamente.
  - Al salto se le añade un *Coyote Time*, que es un concepto en el diseño de juegos de plataformas que permite a los jugadores seguir saltando brevemente después de haber salido de un borde. Este pequeño margen de tiempo da la sensación de mayor control y fluidez permitiendo que los jugadores ejecuten saltos más precisos, incluso si no lo hacen en el momento exacto.

```
private void handleGravity()
    {
        // Aplicar gravedad adicional cuando se cae hacia bajo
        if (rb.velocity.y < 0)
        {
            rb.velocity += Vector2.up * Physics2D.gravity.y * (fallMultiplier - 1) * Time.fixedDeltaTime;
        }
        // Aplicar modificador de gravedad cuando se suelta antes el botón de salto
        else if (rb.velocity.y > 0 && jumpInputReleased)
        {
            rb.velocity += Vector2.up * Physics2D.gravity.y * (JumpEndEarlyGravityModifier - 1) * Time.fixedDeltaTime;
        }
        // Limitar velocidad vertical para prevenir una caída libre excesivamente rápida
        if (rb.velocity.y < -MaxFallSpeed)
        {
            rb.velocity = new Vector2(rb.velocity.x, -MaxFallSpeed);
        }
        // Resetear booleano para indicar que ya está en el suelo y por lo tanto, puede volver a saltar
        if (onGround)
        {
            IsJumping = false;
        }
    }
```

```
private void Update()
    {
        // Contador del Coyote Time
        if (onGround)
        {
            coyoteTimeCounter = CoyoteTime;
        }
        else
        {
            coyoteTimeCounter -= Time.deltaTime;
        }
        // Decrement jump buffer counter
        jumpBufferCounter -= Time.deltaTime;
        // Llamar a handleJump para manejar la lógica del salto
        handleJump();
    }
```

Código 4: Función Update para control de gravedad y salto

- Movimiento:
  - La función SetDirection se encarga de indicar hacia que dirección mirará el jugador, y es muy útil para coordinar los sprites del personaje y sus animaciones con los inputs que da el jugador.
  - La función de handleMovement() gestiona el movimiento horizontal del personaje. Si el jugador puede moverse y está vivo, se verifica la entrada del usuario en el eje X. Si hay movimiento (input mayor a 0.1), se calcula la velocidad objetivo y se aplica una fuerza en función de la diferencia entre la velocidad actual y la velocidad objetivo, limitando la fuerza para evitar aceleraciones excesivas. Si no hay entrada, se aplica una fuerza de desaceleración para reducir gradualmente la velocidad, o se detiene completamente al personaje si su velocidad es muy baja. Esto permite un control suave y responsivo del movimiento del personaje.

```
private void SetDirection(Vector2 moveInput)
    {
        if (moveInput.x > 0 && !isFacingRight)
        {
            //Mirar a la derecha
            isFacingRight = true;
        }
        else if (moveInput.x < 0 && isFacingRight)
        {
            //Mirar a la izquierda
            isFacingRight = false;
        }
    }
private void handleMovement()
    {
        float moveInputX = moveInput.x;
        if (CanMove && IsAlive){
            if (Mathf.Abs(moveInputX) > 0.1f) // Apply force when there is input
            {
                float targetSpeed = moveInputX * moveSpeed;
                float speedDiff = targetSpeed - rb.velocity.x;
                float movement = speedDiff * accelRate;
```

```
            // Clamp the force to prevent excessive acceleration
            movement = Mathf.Clamp(movement, -moveSpeed * accelRate, moveSpeed * accelRate);
            rb.AddForce(new Vector2(movement, 0f));
        }
        else // Apply deceleration force when there is no input
        {
            if (Mathf.Abs(rb.velocity.x) > 0.1f) // Check if the player is moving
            {
                float decelerationForce = rb.velocity.x * decelRate;
                rb.AddForce(new Vector2(-decelerationForce, 0f));
            }
            else // Stop the player completely if the velocity is very low
            {
                rb.velocity = new Vector2(0f, rb.velocity.y);
            }
        }
    }
```

Código 5: Función de movimiento

- Otras mecánicas como el *dash*, el ataque son más sencillas, no obstante, se han implementado también el uso de fuerzas, en el caso del *dash*, y restringiendo cualquier fuerza y movimiento, en el caso del ataque básico.

Para estos movimientos con físicas más complejas, se han observado los diferentes controles de otros juegos como podría ser Celeste, como ejemplo principal, ya que es famoso por sus controles muy responsivos y cómodos, y mediante la observación de como reaccionan los controles en otros juegos, es posible incorporar y mejorar los propios. Aún así, dado que el juego está todavía en fases iniciales del desarrollo, será necesario hacer lo que se llama *Fine Tuning*, es decir, ajustar estos parámetros para llegar a un manejo del juego personal y adaptado a lo que se requiere en cada caso.

6.3.2 Creación de comportamiento de enemigos e interacción con el jugador

Inicialmente, se quería adquirir un entendimiento mayor y explorar con el comportamiento de enemigos e interacciones en combate, creando formas simples en las que el jefe en cuestión puede leer movimientos del jugador y responder acorde con ellos. Se ha experimentado con varios mecanismos de recursión de patrones de los jefes dependiendo de movimientos del jugador, como previamente se ha comentado.

Se han desarrollado en su totalidad tres jefes, cada uno con sus particularidades. Adicionalmente, se expondrán por orden cronológico, ya que es relevante para la evolución de la ejecución del trabajo.

6.3.2.1 Jefe con corrutinas (La Ermitaña)

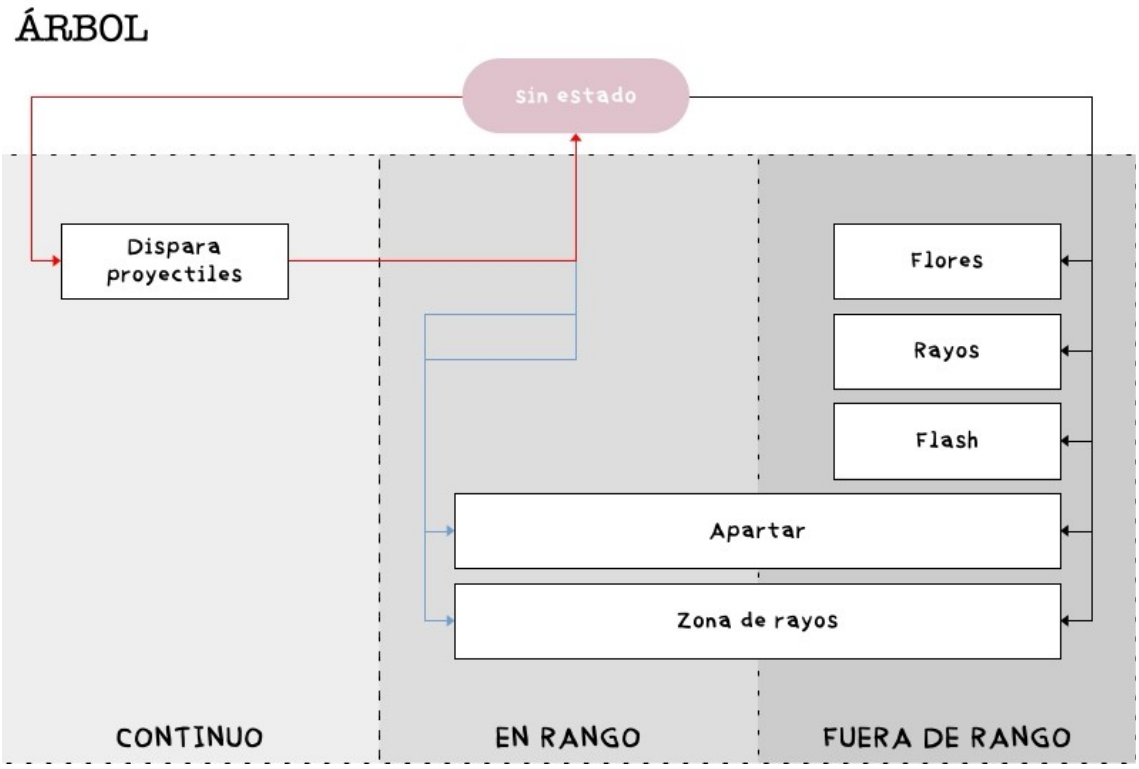
El árbol de la ermitaña fue el primer jefe desarrollado. Debido a que este jefe era estático, se optó por empezar con él, ya que sus patrones de ataques constaban principalmente en la generación de proyectiles y ataques en área que a afectarían al jugador desde un rango, obligando al jugador a acercarse al jefe inmóvil para dañarle. Es por esta simplicidad por la que se optó por emplear corrutinas para sus patrones de ataque y por lo que fue el primero en la fase de desarrollo.

Esta forma de abordar el problema resultó con rutinas que se ejecutaban de forma secuencial y para lo que pretendía abarcar este jefe en concreto fue suficiente. Sus ataques simples no requerían de mucha limpieza posterior ya que la mayoría eran autoconclusivos, es decir, empezaban en la corrutina y se cerraban o desactivaban en la misma. Por ejemplo, proyectiles se generaban y se destruían pasado un tiempo desde el mismo script de comportamiento de los mismos, lo que

permitía quizás prescindir de ciertos mecanismos de control, depuración de recursos y prevención de errores como en el caso de las máquinas de estado.

Pese a la simplicidad del jefe, no obstante, pronto el código comenzó a resultar difícil y confuso a la hora de trabajar con él, lo que motivó a la exploración de otros métodos más avanzados de cohesión de comportamiento de entidades.

Como se observa en el esquema, la entidad dispara proyectiles de forma continuada y sólo cuando el jugador entra en rango, se ejecutan dos ataques (que también se ejecutan fuera de rango, pero con porcentajes de probabilidad diferentes). Cuando está fuera de rango, el número de ataques es mayor.



Esquema 1: Flujo ataques "La Ermitaña"

```
private void ChooseAttack()
    {
        if (Damagable == null)
        {
            Debug.LogError("Damagable is not initialized!");
            return;
        }
        if (PlayerControl== null){
            Debug.LogError("PlayerControl is not initialized!");
            return;
        }
        //bool isPlayerAlive = PlayerControl.IsAlive;
        //Debug.Log("Elegir ataque"+PlayerControl.IsAlive); // Error popping up HERE!!!
        if (PlayerControl.IsAlive && !Damagable.IsStunned)
        {
            Debug.Log("Eligiendo");
            System.Random rng = new System.Random();
            int randomValue = rng.Next(100); // Random number between 0 and 99
            if (DetectionPushRange.InRangeForPush)
```

```
{
    Debug.Log("En rango");

    if (randomValue < 50) // 50% chance
    {
        // High chance attacks
        int highChanceRandom = rng.Next(3);
        if (highChanceRandom == 0)
        {
            Debug.Log("Back Off");
            StartCoroutine(BackOff());
        }
        else if (highChanceRandom == 1)
        {
            StartCoroutine(FlowerAttack());
        }
        else if (highChanceRandom == 2)
        {
            //StartCoroutine(LightBeam());
        }
    }
    else // 50% chance
    {
        // Medium chance attacks
        int mediumChanceRandom = rng.Next(2);
        if (mediumChanceRandom == 0)
        {
            StartCoroutine(lightBeamZone());
        }
        else if (mediumChanceRandom == 1)
        {
            StartCoroutine(LightFlash());
        }
    }
}
else
{
    // While not in range:
    // High chance: flowerAttack, lightBeam
    // Medium chance: immuneBackOff, lightFlash, backOff

    if (randomValue < 50) // 50% chance
    {
        // High chance attacks
        int highChanceRandom = rng.Next(2);
        if (highChanceRandom == 0)
        {
            StartCoroutine(FlowerAttack());
        }
        else if (highChanceRandom == 1)
        {
```

```
                    //StartCoroutine(LightBeam());
                }
            }
            else // 50% chance
            {
                // Medium chance attacks
                int mediumChanceRandom = rng.Next(3);
                if (mediumChanceRandom == 0)
                {
                    StartCoroutine(lightBeamZone());
                }
                else if (mediumChanceRandom == 1)
                {
                    StartCoroutine(LightFlash());
                }
                else if (mediumChanceRandom == 2)
                {
                    StartCoroutine(BackOff());
                }
            }
        }
    }
}
```

Código 6: Función aleatoria patrón de ataques

Esta función se hizo a posteriori, para añadirle un elemento de aleatoriedad al patrón de ataques, no obstante, se tiene preferencia por ciertos ataques dependiendo de la posición del jugador. Esta preferencia sigue siendo muy rudimentaria y simplemente distingue de dentro de rango y fuera de rango, basándose en la detección de colisiones. El cambio en estas detecciones se almacena en un booleano (`DetectionPushRange.InRangeForPush`) que a su vez se controla a través de un *script* adherido a la cápsula de colisión.

```
public GameObject beamPrefab;
public float spawnInterval = 2f;
public int iterations = 3;
public float beamDelay = 0.2f;
IEnumerator lightBeamZone() {
    finishedMainAttack = false;
    for (int i = 0; i < iterations; i++)
    {
        // Spawn the beam at the player's position with a delay
        Vector2 beamLocation = playerTransform.position;
        yield return new WaitForSeconds(beamDelay);
        GameObject beam = Instantiate(beamPrefab, beamLocation, Quaternion.identity);
        // Wait for the next spawn
        yield return new WaitForSeconds(spawnInterval);
    }
    finishedMainAttack = true;
}
```

Código 7: Ataque en particular

Aquí tenemos un ejemplo de lo que sería el código de un ataque en particular. Éste en concreto hace aparecer 3 rayos que se mostrarán después de un retraso introducido para dar tiempo al jugador. Como se puede ver, el ataque no es más que una corrutina ejecutada desde la función de elección de ataque. Los rayos se hacen desaparecer desde el propio script de los rayos, que tienen

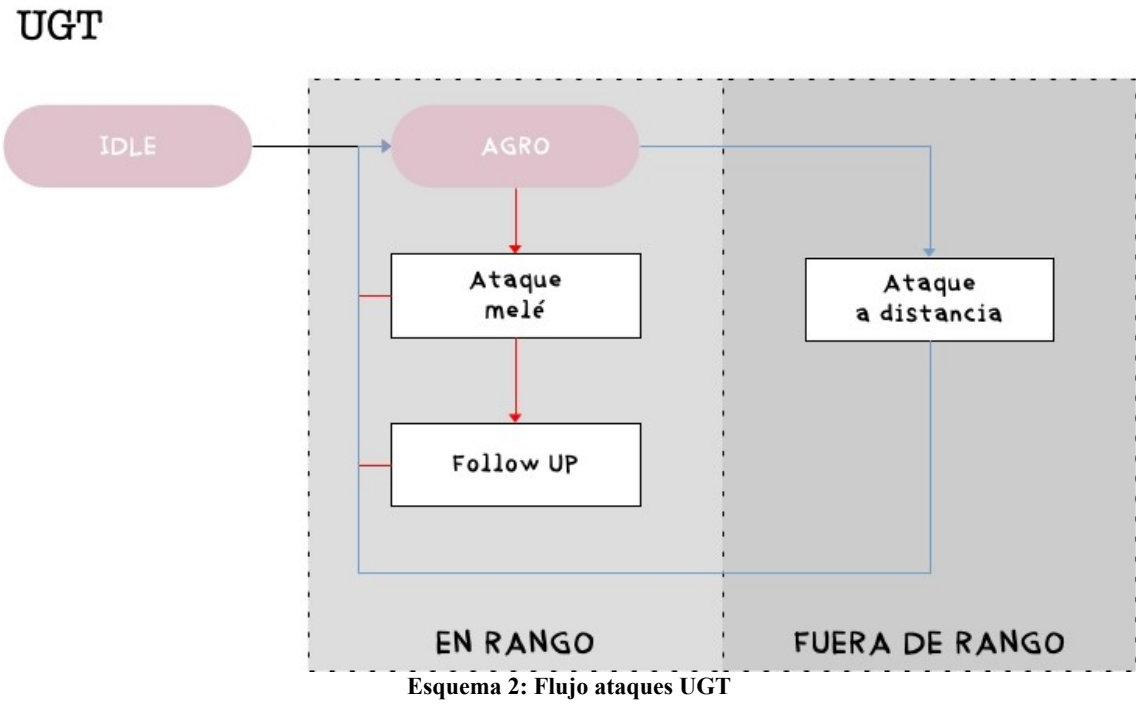
unas pocas funcionalidades, como bien desaparecer después de un tiempo, un controlador de animaciones y la capacidad de infligir daño al jugador.

6.3.2.2 Jefe con máquinas de estado y eventos de Animaciones (Coalición de Trabajadores)

La coalición de trabajadores (*UGT*) fue el segundo jefe desarrollado y, ya que constaba de un enemigo que disponía de movimiento autónomo y animaciones más complejas, debían ser acompañadas por sus correspondientes patrones de ataque. Para abordar este nuevo reto, se constituyó una máquina de estados finitos, *BossUGTBehaviour*, en la cual se manejaban las diferentes transiciones de estado y se almacenaban ciertas funciones para manejar ataques, que luego serían llamadas en las animaciones a través de Animation Events, que permitía activar y desactivar ciertos elementos (*hitboxes*, generación de proyectiles o inicio de movimientos) en *frames* concretos dentro de las animaciones.

Este enfoque fue mucho más fácil de explorar ya que corregir errores y controlar el flujo de ataques y respuestas por parte del jefe resultaba mucho más manejable y claro. Era una clara mejora al método de corrutinas. Aún así, pese a esta nueva mejora, al principio costó acostumbrarse a este nuevo método y debido a la mezcla con *Animation Events* de Unity, se usó inicialmente solamente para la ejecución de animaciones, y se manejaba el resto de la lógica de ataque a través de funciones declaradas en el script de la máquina de estado, accediéndose a ellas a través de eventos de animación.

En el esquema se explica el funcionamiento. Una vez el jugador ha entrado en rango, el jefe ya nunca vuelve a su estado "*Idle*". Queda siempre en estado "*Agro*"



Esquema 2: Flujo ataques UGT

```
private void Start()
    {
        ChangeState(new ugtIdle(this, player, animator)); // Initial state
    }

    private void Update()
    {
        currentState?.Execute();
    }
```

```
public void ChangeState(ugtState newState)
{
    currentState?.Exit();
    currentState = newState;
    currentState.Enter();
}
```

Código 8: Máquina de estados

En estas funciones se definen las funciones necesarias para crear una máquina de estado, basándose en una interfaz de un estado y en la que cada estado tendrá tres funciones (Enter(), Execute() y Exit()) que se ejecutarán a la entrada, como método Update, y a la salida respectivamente.

```
public ugtAgro(bossUGTbehaviour boss, GameObject player, Animator animator) : base(boss, player, animator) { }
    private float ugtMoveSpeed = 5.0f;
    private float distanceToAttack = 5.0f;
    public override void Enter()
    {
        //Debug.Log("Entering Agro State");
        // Face the player
            boss.FacePlayer();
    }
    public override void Execute()
    {
        // Follow the player
        if (player != null)
        {
            // Calculate horizontal direction
            Vector3 direction = (player.transform.position - boss.transform.position).normalized;
            direction.y = 0; // Ignore vertical movement
            // Move towards the player in horizontal direction only
            if (boss.CanMove)
            {
                boss.transform.position += direction * ugtMoveSpeed * Time.deltaTime;
            }
        }
       // Example transition to AttackState
       if (Vector2.Distance(boss.transform.position, player.transform.position) < distanceToAttack)
       {
           //boss.InRangeForMelee = true;
           boss.ChangeState(new ugtAttack(boss, player, animator));
       } else
       {
           boss.ChangeState(new ugtAgro(boss, player, animator));
       }
   }
```

Código 9: Ataques fuera y dentro de rango

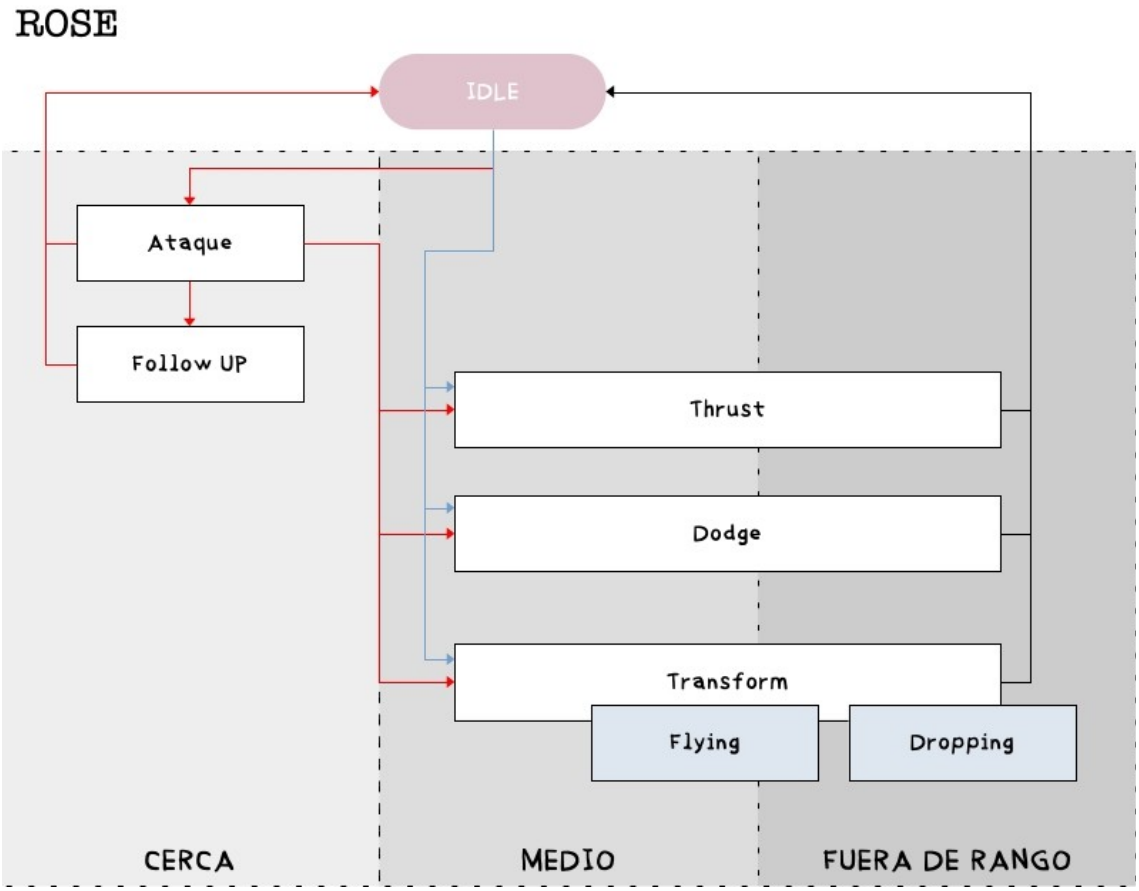
Aquí tenemos el ejemplo de un estado, el estado de *Agro*, que es uno de los más importantes de este jefe, ya que se encarga de perseguir al jugador. Primeramente, usamos la entrada para llamar a la función de observar al jugador y dependiendo de su distancia, se llamará otra vez al *Agro*, o si está en rango se ejecutará el ataque. Debido a los patrones más sencillos de este jefe, no es necesario una función de elección de estado, ya que hará ataques básicos a rango de melé y por el

contrario tirará proyectiles si sales de un rango que tiene definido. Por lo tanto tendrá un estado de persecución, otro a rango y uno que hará al jefe acercarse poco a poco al jugador.

6.3.2.3 Jefe con máquinas de estado finito con un enfoque mixto (Caballero Rosado)

El último jefe de esta versión inicial, el Caballero Rosado, consta de un jefe con un sistema de manejo de patrones de ataque muy similar al anterior. También se emplean eventos de animación para el manejo de ciertos ataques. Éste posee un sistema ligeramente más complejo que el anterior, ya que se integrarían ataques más diferentes, que no eran solo ataques básicos y aparición de proyectiles, ya que su movimiento estaría limitado a un ataque de esquive que ejecutaría ocasionalmente y constaba con otros movimientos un poco más complejos que simplemente avanzar hacia el jugador como en el anterior caso.

Debido a esta añadida complejidad, se decidió hacer una función ChooseAttack(int playerDistance), que dependiendo de la distancia del jugador (en rango de melee, en medio rango o a larga distancia) actuaría de una manera u otra, y que dentro de estas tres clasificaciones, elegiría una dado un valor pseudoaleatorio generado en el momento de elección.



Esquema 3: Flujo ataque Rose

```
public void ChooseAttack(int playerDistance)
    {
        int randomNumber = Random.Range(0, 100);
        Debug.Log("Choosing attack");
        Debug.Log(randomNumber);
        if (playerDistance == 0) {
            // Melee range = Attack, Follow Up, --Thrust
            if (randomNumber <= 65) {
                // Attack
```

```
                ChangeState(new roseAttack(this, player, animator));
            }
            else if (randomNumber > 65 && randomNumber <= 90) {
                // Dodge
                ChangeState(new roseDodge(this, player, animator));
            }
            else if (randomNumber > 65) {
                // Thrust
                ChangeState(new roseThrust(this, player, animator));
            }
        } else if (playerDistance == 1) {
            // Medium range = Thrust, Dodge, --Transform
            if (randomNumber <= 35) {
                // Thrust
                ChangeState(new roseThrust(this, player, animator));
            }
            else if (randomNumber > 35 && randomNumber <= 70) {
                // Dodge
                ChangeState(new roseDodge(this, player, animator));
            }
            else if (randomNumber > 70) {
                // Transform
                if (!hasTransformed)
                {
                    ChangeState(new roseTransform(this, player, animator));
                } else {
                    ChangeState(new roseThrust(this, player, animator));
                }
            }
        } else if (playerDistance == 2) {
            // Far range = Thrust, Dodge, --Transform
            if (randomNumber <= 35) {
                // Thrust
                ChangeState(new roseThrust(this, player, animator));
            } else if (randomNumber > 35 && randomNumber <= 70) {
                // Dodge
                ChangeState(new roseDodge(this, player, animator));
            } else if (randomNumber > 70) {
                if (!hasTransformed) {
                    // Transform
                    ChangeState(new roseTransform(this, player, animator));
                }
                else {
                    ChangeState(new roseThrust(this, player, animator));
                }
            }
        }
    }
```

Código 10: Manejo mejorado flujo de ataques

Dado este nexo de elección de patrones, resultaba mucho más accesible el manejo del flujo de ataques. Este sistema se pretende mejorar e implementar en el resto de jefes del juego para facilitar su ampliación y su depuración.

Junto a este nuevo método, también se usaron los eventos de animaciones y los estados de manera simultánea, aprovechando los puntos fuertes de cada sistema para intentar mejorar la interacción.

Aquí se mostrará el código de una secuencia de ataques:

```
public override void Enter()
    {
        Debug.Log("Entering Transform State");
        // Attack setup code
        boss.FacePlayer();
        animator.SetTrigger(animatorStrings.transformStart);
        // Calculate the target position directly above the initial position
        targetPosition = new Vector2(boss.transform.position.x, boss.transform.position.y + verticalDistance);
    }
    public override void Execute()
    {
        MoveUpwards();
        Debug.Log("Animation Finished: " + boss.animationFinished);
        Debug.Log("Movement Finished: " + movementFinished);
        if (movementFinished && boss.animationFinished)
        {
            boss.ChangeState(new roseFlying(boss, player, animator));
        }
    }
    public override void Exit()
    {
        Debug.Log("Exiting Transform State");
        // Cleanup code for attack state
    }
    private void MoveUpwards()
    {
        if (!movementFinished)
        {
            if (Vector2.Distance(boss.transform.position, targetPosition) > 0.1f)
            {
                    boss.transform.position = Vector2.MoveTowards(boss.transform.position, targetPosition, moveSpeed *
Time.deltaTime);
            }
            else
            {
                boss.transform.position = targetPosition;
                movementFinished = true;
            }
        }
    }
```

Código 11: Código de secuencia de ataques

Este sería un ejemplo de un estado en el que el jefe se transforma y se alza en el aire. Después de este estado, se ejecutarán otros dos más como parte de la secuencia de éste ataque, a los que se accederán únicamente a través de éste.

6.4 Sistemas de juego y mecánicas

6.4.1 Jugabilidad y combate

6.4.1.1 Movimientos básicos

Se pretende que el combate sea similar a otros juegos del mismo género, como podría ser el *Blasphemous* o *Hollow Knight*. Es decir, un combate relativamente sencillo en el que tu principal herramienta de infligir daño, consistirá en un ataque básico a corta distancia, por lo que el jugador deberá ser capaz de acercarse a sus objetivos e intentar no perder vida en sus enfrentamientos.

Este ataque básico será acompañado de otro movimiento esencial en la jugabilidad, este es, un *dash* mediante el cual podrás recorrer distancias de forma más rápida y mantenerte invulnerable durante un cierto tiempo.

Se pretende que los movimientos del jugador se mantengan muy sencillos, sin añadir muchos movimientos adicionales, aunque si se pretende añadir en próximas versiones una integración de ataques básicos en el aire, ciertas mejoras de movimiento y restricción del mismo, por ejemplo, al ser dañado y, como objetivo importante, un sistema de *parry*, es decir, una forma de bloquear ataaques y realizar un contraataque más poderoso. Por lo tanto se pretende acabar con, principalmente, tres grandes movimientos básicos del jugador, el ataque, el esquive y el bloqueo.

6.4.1.2 Dinámicas de peleas (En el presente vs. En el futuro)

Se busca que las peleas sean ligeramente más rápidas que de costumbre y se anima a que el jugador sea capaz de leer la pantalla y reaccionar rápidamente a los distintos patrones de ataque.

En el actual estado del juego, esta rapidez que se busca quizás no es del todo perceptible, ya que aún faltan varias mecánicas importantes por implementar, como bien puede ser el *parry*. Además, queda mucho *fine tuning* por hacer en cuanto al movimiento del personaje que, a medida que avance el proceso de desarrollo, irá mejorando el ritmo de las batallas para alcanzar lo que se tiene en mente.

6.4.1.3 Mecánicas de progresión

El juego inicialmente debe parecer ligeramente costoso, y a medida que el jugador vaya rejugando, irá obteniendo beneficios permanentes que afectarán a su partida general, y pese a tener que rejugar siempre los mismos combates, cada vez hacerlo de una forma ligeramente diferente, ya sea con más ayudas, más desafíos o variedad de habilidades, dependiendo del objetivo del jugador en esa partida.

6.4.1.4 Curación

En la primera versión del juego, no existe manera de curar al personaje mientras se está dentro de una partida, y esto es intencional, ya que se pretende que el jugador inicialmente se encuentre ligeramente abrumado con superar los tres combates iniciales con tan solo una barra de vida. Esto, no obstante, irá cambiando a medida que se vaya progresando, ya que el jugador encontrará métodos de curación, hasta finalmente conseguir un objeto comparable a la clásica poción de vida, que se agregará de forma permanente a disposición del usuario.

6.4.1.5 Penalización por muerte

La penalización por muerte es una forma de entorpecer la progresión y forzar al jugador a aprenderse los diferentes patrones de los enemigos que se irá encontrando. Consiste en disminuir la barra de vida máxima del personaje (máximo a mitad de vida) y obligar al jugador pagar una cantidad de dinero para quitarse esta aflicción (penalización), cosa que podrá hacer en el HUB principal del juego, en el edificio del Club Nocturno.

6.4.1.6 Tienda

Será en la tienda de pelucas donde el jugador podrá equiparse pelucas que le proporcionará con habilidades únicas. El jugador deberá alquilar las pelucas cada vez que quiera iniciar una partida y le proveerán con efectos únicos.

Se pretende que estas pelucas acompañen a la progresión del jugador, por ejemplo, al iniciar el juego el jugador se encuentra sin forma de curarse, no obstante, tras derrotar al primer jefe, se le proporciona una peluca capaz de curar ligeramente por cada golpe que ejecuta. Es decir, el jugador no conseguirá inmediatamente las pociones de curación, pero se le irá acompañando con mejoras apropiadas para su estado en el juego, creando así un sistema de recompensas progresivo y apropiado. Además, pelucas que en un principio podían ser ligeras ayudas para el jugador, en un juego tardío, podrían ser reutilizadas como retos en vez de ayudas, ya que se dispondrá de mejoras mucho más avanzadas.

6.4.1.7 Cordura

La cordura sería una mecánica que sería implementada en un más tardío estado de desarrollo del juego, y consistiría en un sistema de incremento de la dificultad a elección del jugador, destinado para gente más avanzada que quiere más retos y una ampliación de la experiencia base.

La idea sería añadir diferentes fases dentro de las peleas ya existentes de los jefes, que sean más complejas y que sean acompañadas de contenido narrativo, para agregar a los diferentes enemigos más profundidad de personaje y un sentimiento de recompensa al jugador, ya que está descubriendo más el mundo que explora, además de abrir nuevos caminos dentro del mapa.

6.5 Gestión de datos y persistencia

En la fase de diseño e implementación no se ha tenido en cuenta cómo gestionar los datos (guardar partidas y cargarlas desde el menú de inicio).

Esta funcionalidad se desarrollaría más adelante, ya que el principal objetivo de este trabajo es el de dejar bien terminados todos los niveles y el Hub inicial, así como las pantallas de transición entre niveles.

Unity proporciona varias formas de gestionar los datos de un juego [12] [13]:

- ***PlayerPrefs***: Sistema sencillo para almacenar y cargar datos de configuración y progreso del jugador. Se pueden guardar y cargar valores de tipo int, float y string.
- ***ScriptableObjects***: Los *ScriptableObjects* permiten crear y gestionar datos de juego reutilizables como configuraciones, tablas de datos, etc. Estos datos se pueden cargar y acceder fácilmente desde los scripts desarrollados.
- **Serialización de objetos**: Esta es la forma más adecuada ya que Unity puede serializar y deserializar automáticamente objetos de juego y sus componentes, lo que facilita el guardado y carga de partidas.

Por tanto, según la bibliografía [14], para poder abordar la gestión de los datos, serían necesarias las siguientes acciones:

- Crear un *script* para la gestión de guardado y carga de partidas: Programado en C# y llamado por ejemplo "SaveLoad" que se encargaría de todas las funcionalidades de guardar y cargar el progreso del juego.
- Construir una clase "SaveLoad" estática: Así se podrá acceder a sus métodos y variables desde cualquier parte del proyecto sin necesidad de usar GetComponent().
- Serializar los datos del juego, es decir, convertirlo a un formato que puede ser salvado y cargado más adelante: Dentro del script "SaveLoad" se importan las librerías necesarias para la serialización de los datos (System.Collections.Generic,

System.Runtime.Serialization.Formatters.Binary, System.IO). Esto permitirá convertir los datos del juego a un formato que se pueda guardar y cargar.

- Crear clases serializables: Crear una clase "Game" que contenga los datos del juego a guardar, como el progreso de los personajes, marcando esta clase como [System.Serializable] para que pueda ser serializada.
- Implementar el guardado de partidas: En el script "SaveLoad", crear una lista estática llamada "savedGames" para almacenar las partidas guardadas. Se agregaría un método estático "Save()" que tome los datos actuales del juego (Game.current) y los serializará en un archivo usando BinaryFormatter y FileStream.
- Implementación de la carga de partidas: Crear un método estático "Load(int index)" en el script "SaveLoad" que tome el índice de la partida a cargar desde la lista "savedGames", deserialice los datos y los asigne a Game.current para que el juego pueda continuar desde ese punto.
- Integración del guardado y carga en el juego: Desde otros scripts del juego, se llamará a los métodos "Save()" y "Load(int index)" de la clase "SaveLoad" cuando sea necesario, por ejemplo, al presionar un botón de "Guardar" o "Cargar" en el menú del juego.

Con todo, así se podrán gestionar los datos del juego desarrollado con Unity y permitirá a los jugadores el guardado y carga de sus partidas. Este enfoque funcionará en la mayoría de las plataformas, excepto en el reproductor web de Unity .

6.6 Pruebas y depuración

A la hora de la depuración y prueba del código se ha usado principalmente el uso de *Debug.Log()* imprimiendo por consola variables relevantes y información de qué partes del código se han estado ejecutando. Debido a la simplicidad de esta primera versión, no se han realizado pruebas excesivamente profundas, pese a que la fase de pruebas es de las más importantes antes de liberar una versión estable del juego.

6.6.1 Principales problemas

Los problemas más relevantes a la hora del desarrollo han sido dos focos principales,

6.6.1.1 Problemas con físicas

En un principio, el correcto funcionamiento de los controles nuevos dio muchos problemas, cuando se decidió optar por hacer el movimiento del jugador basado en fuerzas, en vez de mover el jugador con transformaciones, ya que fue difícil ajustar todos los parámetros y añadir las fuerzas necesarias para dar la sensación de unos controles responsivos.

6.6.1.2 Problemas con el animador

A veces el animador de Unity generaba algunos problemas, por ejemplo, no cargando animaciones correctamente, saltándose algún *frame* o a la hora de ejecutar algunos eventos de animación. Algunos de estos errores aún persisten de vez en cuando, ya que resulta siempre más difícil depurar elementos integrados con la interfaz de Unity directamente, a diferencia de si fueran problemas de código.

6.6.1.3 Problemas con interacciones

Algunos *scripts* hacían uso de corrutinas de forma secuencial, a la vez que usaban estados, lo que hacía que a veces un estado se ejecutara repetidas veces, lo que ocasionaba problemas con las animaciones y con la progresión de los ataques. Debido a esto se introdujeron más variables de control para comprobar que las funciones se ejecutaban tan solo una vez, lo que solucionó varios problemas.

Capítulo 7. Elaboración del GDD

7.1 Descripción general del juego

Título	El Gran Capital
Género	Acción/Aventura
Perspectiva de juego	Tercera persona
Modo de juego	Un jugador
Público objetivo	De 16 a 30 años
Idea central	El jugador irá desvelando poco a poco una historia tras múltiples partidas. Para ello, tendrá que enfrentarse a varios jefes que le ayudarán a desbloquear mejoras permanentes.
Objetivo	Derrotar a los jefes en los diferentes niveles para poder resolver la historia que se oculta tras los personajes del juego y el mundo mostrado.
Tema de juego	Combate y progresión
Clasificación PEGI	

Tabla 1: GDD - Descripción del juego

[

7.2 Contexto del juego

7.2.1 Historia (lore)

El mundo está siendo devorado por un núcleo galáctico y la humanidad, debido a la gruesa atmósfera que protege el mundo de los rayos cósmicos, permanece en la ignorancia, sin saber que sus últimos días se están acercando.

Norman pertenece a una familia adinerada, y pese a su rechazo inicial al estilo de vida de un hombre de negocios, ahora, bañado en la bebida y la mala vida, decide aprovecharse del fallecimiento de sus dos padres tras un accidente para reclamar parte de su herencia y se dispone a llegar al lugar de su vieja empresa familiar, y hablar con su primo, que ahora ha heredado el título de CEO.

Además, Norman comienza a tener sueños recurrentes acerca de la llegada de monstruos cósmicos y el principio del apocalipsis. Comienza a desconfiar de todos los que le rodean, desenmascarando así una conspiración de oriente para desestabilizar el sistema económico occidental, o al menos, lo que queda de él.

A través de este viaje conocerá a personajes como La Ermitaña, una vieja desencantada con el sistema, que ha conseguido transmitir su conciencia a lo que ella cree que es la forma de vida superior, los árboles, o una amalgama de obreros que formaron una Unión de Trabajadores, solo para ser corrompidos desde dentro y dejados a merced de las condiciones climáticas extremas del

planeta, muriendo y reencarnándose como uno solo a través de la unión de sus conciencias en una colectiva.

7.2.2 Personajes

Las tablas de cada personaje muestran vida y valor de cada ataque. Además, existe la postura (que es un valor que se va regenerando) y los puntos vitales que son como fases de regeneración total de los jefes (por ejemplo, si un jefe recibe daño y se le agota la vida, descuenta un punto vital y recupera toda la vida).

7.2.2.1 Norman

Es el personaje manejado por el jugador, hijo de familia rica, pero perdido en la vida. Atormentado por visiones del inframundo y abismos cósmicos. Quiere reclamar una parte del pastel de la herencia, del que su primo hermano se ha apropiado para él.

Vida	100
Ataque básico	25
Dash	-
Habilidades adquiridas	Peluca de fuego: Regenera +5 por cada básico Peluca de señora: Protección de -5% del daño recibido Peluca rastas: Mejora el dash

Tabla 2: GDD – Personaje NORMAN

7.2.2.2 La Ermitaña

Una mujer que se retiró a los pocos campos que quedaban, desencantada por la enormidad de su ciudad, y su gente. El día de su muerte pareció que su recuerdo quedó grabado en un árbol.

Vida	900
Postura	1500
Puntos vitales	3
Proyectiles	10
Flores	10
Rayos	20
Flash	5 por tic
Apartar	15
Zona de Rayos	20

Tabla 3: GDD – Personaje Ermitaña (ÁRBOL)

7.2.2.3 Coalición de trabajadores

Tras un incidente laboral en la meseta de Gaar, al noroeste de la ciudad, donde varios trabajadores fallecieron debido a las bajas condiciones laborales de le empresa Nom, un "espíritu" parece haber heredado los recuerdos de todos ellos. Ahora luchan como una coalición de trabajadores unidos en un solo ser, recordando su lucha.

Vida	1500
Postura	700
Puntos vitales	2
Melé	15
Follow UP	20
Distancia	10

Tabla 4: GDD – Personaje Coalición de trabajadores (UGT)

7.2.2.4 Rose

Un conocido de la familia de Norman, que viene reclamando un puesto en la empresa. Extranjero. Su apellido proviene de un lugar al este del país, en el que se puede encontrar uno con gente de auténtica influencia. Un verdadero hijo de la "realeza".

Vida	2000
Postura	500
Puntos vitales	3
Ataque básico	25
Follow UP	30
Thrust	25
Dodge	-
Transform	25 + 35

Tabla 5: GDD – Personaje Caballero Rosado (ROSE)

7.3 Juego

7.3.1 Objetivo

El objetivo del juego es navegar a través de los niveles y en cada iteración del juego (incursión), se pretende que el jugador llegue lo más lejos que pueda y mientras tanto vaya recogiendo los recursos necesarios para posteriormente gastarlos en el *Hub*, solamente para hacerse más fuerte y volver a intentar el desafío. No se esperará que el jugador se pase el juego en la primera incursión, ya que el objetivo es que se vaya reintentando hasta descubrir todos los secretos que encierra el juego en forma de historia.

El jugador deberá:

- Administrar sus monedas e invertirlas en dos diferentes unidades de gasto de recursos (tienda de pelucas y club nocturno).
- Intentar ir memorizando los patrones de ataque de los diferentes jefes para facilitar sus próximas incursiones.
- Ir superando los jefes, así ganando más mejoras para el personaje y facilitando la progresión en el futuro.
- Exploración e investigación, ya que se pretende que el jugador encuentre caminos ocultos en los momentos entre jefes, incluso llegando a poder encontrar peleas contra jefes ocultos.

7.3.2 Lógica del juego

- El jugador vuelve al inicio cuando su vida se agota. Sin embargo, podrá conservar algunos de los avances realizados antes de morir.
- La zona de combate tiene unos límites reconocibles por unos pilares metálicos
- Hay una serie de plataformas que permiten saltar sobre ellas para subir y bajar
- La caída desde una zona muy alta no provoca pérdida de vida. Sólo los ataques recibidos por criaturas o jefes
- Si la barra de vida o postura de un jefe se agota, consumirá un punto vital. Cuando los puntos vitales se terminen, el jefe será derrotado.
- Si un arma se utiliza demasiadas veces, se romperá. Por lo tanto, un arma tiene una vida que es la cantidad de veces que se puede utilizar antes de que se rompa

7.3.3 Mecánicas

7.3.3.1 Reglas

Una vez entrado en el juego, el jugador dispondrá de dos lugares que visitar. Uno es la tienda, en la que el jugador podrá entrar y hablar con un personaje que le pondrá en contexto. Es ahí cuando le indicará hablar con el otro personaje ubicado en el Club. Aquí podrá hablar con otro NPC[9], y acabado esto, ya se podrá disponer a empezar su juego.

Ahora el jugador deberá empezar una incursión, con sus escasos recursos, y se pretende que intente llegar lo más lejos posible, pero se espera que muera relativamente pronto en la expedición. Tras haber muerto, volverá a despertar en el Hub, donde podrá gastar los recursos adquiridos durante la expedición. Cuanto más vaya avanzando, más recursos tendrá y podrá mejorar su equipamiento inicial, para mejorar sus posibilidades en las siguientes partidas.

Las incursiones consistirán de dos principales partes diferenciadas. Estará en una primera instancia **la exploración** en la que el jugador podrá familiarizarse con el mundo e investigarlo, mientras intenta evitar caerse de las plataformas, pudiendo toparse con monedas, caminos ocultos o salas dónde podrá combatir con jefes.

Es aquí donde entrará el combate, la segunda parte distinguida del juego. En los combates el jugador debe derrotar a su oponente y obtener su código de acceso necesario para desbloquear la última puerta del juego. Es aquí donde el jugador podrá obtener las mayores recompensas para el progreso del jugador.

Para combatir, el jugador puede hacer uso de poderes especiales dados por las pelucas que puede encontrar en la tienda del Hub, que podrá ir mejorando a medida que vaya obteniendo dinero.

9 NPC: *Non-player Character*. Jugador no jugable (automático)

Cada muerte del jugador se verá penalizada por un pequeño decremento de la vida máxima del mismo. Esta penalización se podrá eliminar en el club nocturno, a cambio de algo de dinero, y volver a empezar con la vida máxima.

Una vez acabado el juego, el jugador podrá rejugarlo con una dificultad mayor qué aumentara los valores de vida del enemigo, así como los de daño, y además introducirá ataques adicionales.

7.3.3.2 Ayuda e información

El jugador recibirá una escasa información a través de cajas de diálogo y de indicaciones sutiles. Cualquier pista o ayuda se dará dentro del contexto narrativo, sin nunca romper la cuarta pared.

Además, en la pantalla de inicio, antes de comenzar, podrá consultar el manual básico de uso (básicamente las teclas a utilizar para el desarrollo del juego)

7.3.3.3 Puntuación y recompensas

El juego recompensará por posibles coleccionables encontrados en la parte de exploración, cada uno será asignado un valor monetario y al final, cuando el jugador muera o bien gane la partida, estos valores serán canjeados. Por otra parte, cada jefe derrotado otorgará un valor adicional, además de un objeto que si es llevado a la tienda de pelucas del Hub, dará la posibilidad de desbloquear una peluca adicional.

A parte de eso, en un futuro se piensa implementar una zona personal del jugador en la que se podrá ver visualmente el progreso. Empezará como una habitación vacía e irá evolucionando a medida que se avance en el juego.

7.3.4 Progresión del juego

7.3.4.1 Hub

Esta es la zona principal e inicial del juego, donde el jugador podrá acceder a la tienda de pelucas y obtener mejoras para su próxima incursión. Además, también podrá acceder al club nocturno para quitarse las penalizaciones por muerte. De aquí también podrá acceder a la parte principal del juego y al primer nivel.

7.3.4.2 Pantallas de transición

Son las que permiten ir progresando en el nivel hasta llegar al jefe final (con el que entrarán en combate). Son pantallas típicas de juego de plataformas en las que se podrán encontrar recompensas y en las que habrá que luchas con algunas criaturas.

7.3.4.3 Combate 1

El primer jefe (La Ermitaña) es un árbol que como se ha comentado, tiene una gran cantidad de ataques. Este nivel es opcional. Puede no jugarse y pasarse al nivel siguiente si se encuentra el camino adecuado.

7.3.4.4 Combate 2

El segundo jefe (UGT) es un trabajador que perseguirá al jugador una vez haya entrado en su rango. Si se quiere progresar al siguiente nivel, es obligatorio pasar por este.

7.3.4.5 Combate 3

El último de los jefes implementados (Rose) tiene mucha vida y muchos puntos vitales, por lo que será difícil de derrotar. Uno de sus ataques (*Transform*) es especialmente letal.

7.4 Elementos del juego

7.4.1 Ambientaciones

Como se ha comentado, el juego está ambientado en una ciudad contaminada y decadente. En el apartado artístico se han expuesto todos los fondos y algunas pantallas que muestran este mundo.

7.4.2 Armas y elementos coleccionables

Existen muy pocas armas. De hecho es que el único que dispone de una es Rose (una espada). El resto de personajes no usa armas. Sólo una serie de ataques mágicos y básicos.

Como elementos coleccionables, sólo se va a recolectar monedas que sirvan como recurso de intercambio para obtener otros (pelucas, más vida, etc.)

7.5 Interfaz

7.5.1 Diagrama de flujo

En el diagrama se muestra el **flujo del juego**. En todas las pantallas mientras se juega, transiciones y niveles, en caso de muerte, se conservan los objetos recolectados, pero se vuelve a la pantalla *HUB*.

En caso de pausar el juego, queda suspendido en pantalla de pausa y se reanuda cuando lo decida el jugador, volviendo exactamente al punto en el que se dejó el juego.

En la primera versión no se ha podido desarrollar el sistema de guardado, ni el de opciones. Otros temas pendientes son la implementación de la pausa y el diseño del jefe final (que se ha eliminado del diagrama)

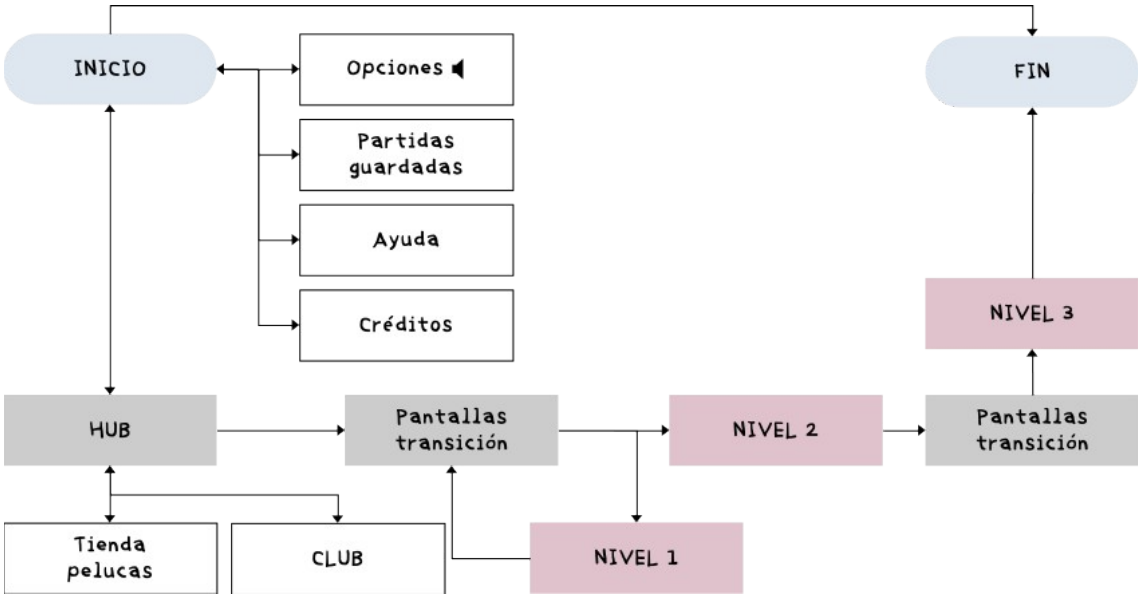


Figura 21: Diagrama de flujo del juego

7.5.2 Controles de teclado y ratón

El juego se ha diseñado sin opción de usar palanca de mando (*joystick*)

Respecto al teclado se usan las siguientes:

- Movimiento: Las habituales teclas (*A S W D*). También las flechas sirven.
- Ataque: Clic izquierdo del ratón
- Esquiva: Shift
- Salto: Barra espaciadora

- Interactuar: Tecla E
- Curación: R [Pese a no estar implementado]
- Pausa del juego: Tecla *ESC*

7.5.3 Interfaces GUI

7.5.3.1 Menú principal

- Inicio: Inicio del juego
- Partidas: Muestra las partidas guardadas
- Ayuda: Muestra pantalla de ayuda del juego
- Puntuaciones: Estadística de puntuaciones
- Créditos: Muestra la autoría del juego
- Salir: Salir del juego

7.5.3.2 Menú pausa

No está implementado, pero será una de las mejoras a desarrollar.

- Reanudar: Reanuda la partida
- Reinicia: Reinicia la partida sin guardar
- Guardar y salir: Guarda la partida y sale del juego
- Salir: Sale del juego sin guardar

7.6 Características visuales y sonoras

7.6.1 HUD[10]

En este punto se muestran dos de las principales pantallas. Un boceto de la pantalla de inicio y una captura de la pantalla de juego indicando en cada zona la información que aparecerá.

Figura 22: Pantalla de inicio (borrador)

10 HUD: Head-Up Display. Es la información que se muestra en pantalla durante la partida.

Figura 23: Pantalla de juego

7.6.2 Audio

7.6.2.1 Banda sonora

Hay un total de siete piezas de música que se pueden escuchar en estos enlaces:

- Pantalla de inicio HUB https://youtu.be/8HaxRXLDE5s
- Pantalla de inventario, opciones, etc. https://youtu.be/bp1_D-0A898
- Pantallas de transición https://youtu.be/IgLJs9QsLL4
- Combate "La Ermitaña" https://youtu.be/NeHqmItW1NU
- Combate UGT https://youtu.be/F4QRXMME7L8
- Combate Rose https://youtu.be/qYFlPT2OGP8
- Pantalla final de juego https://youtu.be/ttUnQHEDmMs

7.6.2.2 Efectos y sonidos

Catálogo de sonidos:

- Selección de opción de menú (al hacer clic)
- Ataque básico de Norman
- Movimiento rápido de Norman (*dash*)
- Salto de Norman
- Varios sonidos para el uso avanzado con pelucas de Norman (hasta tres sonidos)
- Recogida de recompensa: Sonido de monedas
- Sonido de muerte de Norman
- 6 sonidos para ataques de La Ermitaña: Proyectiles, flores, rayos, flash, apartar y zona de rayos.
- Sonido de muerte de La Ermitaña

- Ataque básico de UGT (a melé)
- Ataque Follow UP de UGT
- Ataque a distancia de UGT
- Sonido de muerte de UGT
- 3 sonidos para ataques de Rose: Básico, Follow UP, Thrust
- Dodge de Rose
- Transformación de Rose en mujer
- Sonido de muerte de Rose
- Victoria al final de cada nivel
- Victoria final de juego

7.7 Requisitos del sistema

- Teclado
- Ratón
- Monitor
- 1 GB RAM
- Procesador de doble núcleo
- Altavoces o auriculares (opcional)
- Disco duro
- Memoria gráfica
- Windows última versión

Capítulo 8. Evaluación económica

Más adelante (en el apartado de conclusiones) se realizará una estimación de los tiempos ya que se ha comprobado que, en este caso, ha faltado más margen para terminar todo el desarrollo completo.

Un tema importante es tratar de usar **herramientas gratuitas** o de sistema operativo para reducir costes del producto: *Ganttproject, Freemind, Libreoffice, GarageBand*, etc.

Basándonos en esos números aproximados, este sería el presupuesto:

Descripción	Tiempo	Coste	Total
Recursos humanos			
Diseño	4 semanas	300€	1.200€
Programación	12 semanas	600€	7.200€
Gráficos y animaciones	9 semanas	400€	3.600€
Música y sonido	2 semanas	400€	800€
Pruebas	7 semanas	400€	2.800€
Herramientas y software			
Phtoshop (suscripción anual)	1 año	240€	240€
Aseprite (licencia)	-	19,99€	19,99€
Elevenlabs (suscripción anual)	1 año	360€	360€
Otros gastos			
Clasificación PEGI		150€	150€
Registrar en plataforma (ejemplo Steam)		100€	91,99€
Imprevistos		1.500€	1.500€
Total aproximado			**18.000€**

Tabla 6: Presupuesto del proyecto

No se ha tenido en cuenta en este presupuesto la parte de **promoción ni marketing** que sin duda podría representar entre un 10% y un 20% del presupuesto total, llegando el presupuesto a unos 21.000€ en total. [17] [18] [19] [20]

Capítulo 9. Resultado final

9.1 Capturas de vídeos demostrativos

9.1.1 Ejemplo de inicio de partida HUB

Figura 24: Ejemplo de juego en youtube https://youtu.be/OUJR9SCyMkQ

9.1.2 Ejemplo de gameplay de pantallas de transición

Figura 25: Ejemplo de juego en youtube https://youtu.be/SU3qsv2g46w

9.1.3 Ejemplo de gameplay contra ÁRBOL "La Ermitaña"

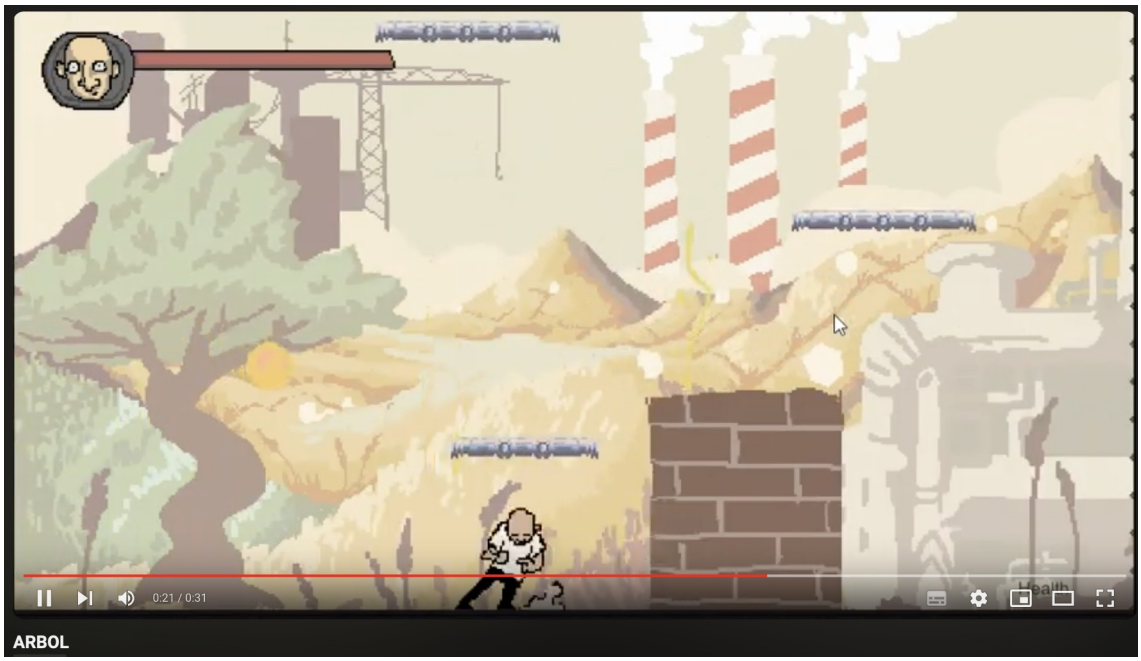


Figura 26: Ejemplo de juego en youtube https://youtu.be/-KHh5oIHw8E

9.1.4 Ejemplo de gameplay contra UGT

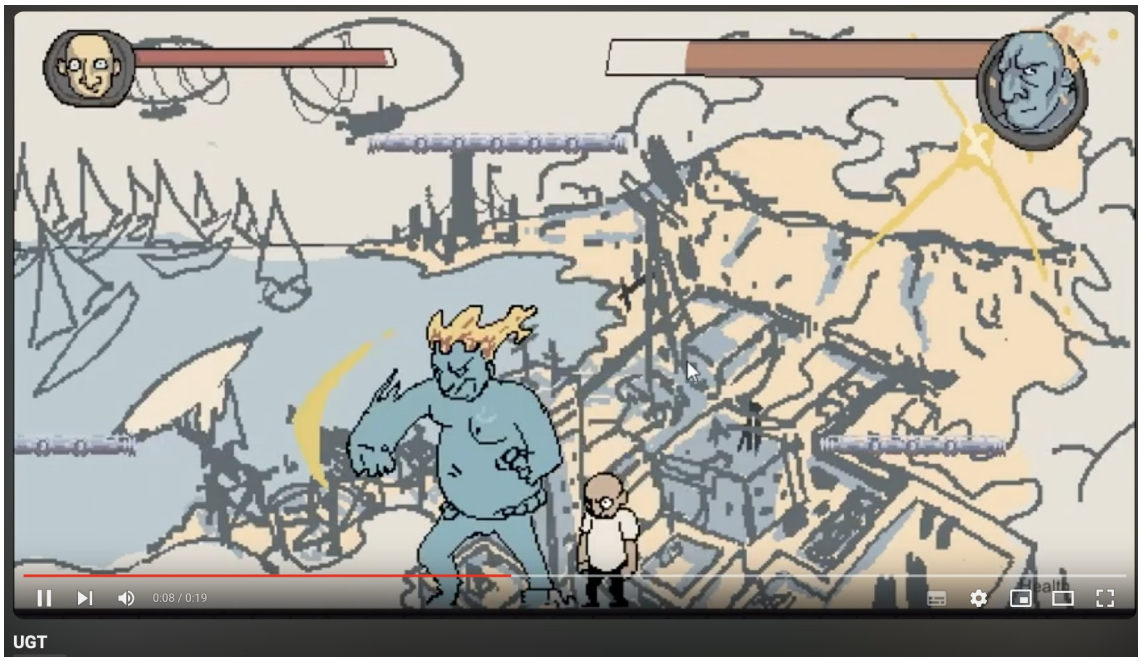


Figura 27: Ejemplo de juego en youtube https://youtu.be/BoWY9yv7dIo

9.2 Descripción del producto terminado

El producto final tiene concluidas las principales pantallas que conforman el juego:

- Hub inicial
- Pantallas de transición
- Niveles de jefes (tres niveles)

Todas las pantallas permiten ser jugadas, aunque no tienen toda la lógica de recompensas implementada.

9.3 Cumplimiento de objetivos iniciales

No ha sido posible desarrollar toda la historia inicial ni tampoco todas las funcionalidades, pero el juego se ha realizado de manera que sea sencillo poder implementar las mejoras que se han ido comentando a lo largo del documento.

Para poder terminar el proyecto, se ha hecho especial hincapié en desarrollar las mecánicas del juego en las **transiciones** y en las **peleas con los jefes de cada nivel**. Todo ello se ha resuelto, pero han quedado pendientes bastantes aspectos y sin ellos, el juego no se puede dar por terminado.

Ni siquiera se puede ofrecer una primera versión, ya que faltan algunos de los elementos importantes que conforman la narrativa de principio a fin del juego (no existe una continuidad; sólo es posible jugar aisladamente en cada una de las pantallas o niveles).

No obstante, se han establecido los pilares para, de una forma muy sencilla y organizada, poder continuar el desarrollo y llegar a publicar el juego en alguna de las plataformas mencionadas.

Capítulo 10. Conclusiones

10.1 Aprendizajes del proceso

Este ambicioso proyecto ha puesto de manifiesto que **implementar un juego de estas características como una sola persona puede ser un desafío muy considerable** ya que, como se ha expuesto, implica:

- Diseñar el juego creando una mecánica atractiva y equilibrada lo cual conlleva tiempo y reflexión.
- Programar la jugabilidad, los controles y la inteligencia de los enemigos.
- Realizar los gráficos y animaciones diseñando y animando los *sprites*, fondos y elementos del juego.
- Realizar el sonido y la música.
- Probar y ajustar todas las mecánicas una y otra vez.
- Apartado de marketing y publicidad, que en este caso, queda fuera del alcance del proyecto.

Como se ha comentado en el capítulo de evaluación económica, visto el tiempo que se ha dedicado a este trabajo, para desarrollar el juego tal y como se había planteado, es más que probable que la inversión en tiempo a realizar rondase los **9 meses de trabajo**, dedicando los siguientes porcentajes:

Fase	Porcentaje de tiempo a dedicar
Diseño	11%
Programación	32%
Gráficos y animaciones	22%
Música y sonido	9%
Pruebas	18%
Otros aspectos estéticos	8%

Tabla 7: Lecciones aprendidas

10.2 Posibles mejoras futuras

Como se ha comentado, el juego no se puede dar por terminado. A continuación se detalla un listado de **tareas a realizar para cerrar una primera versión** y una lista de posibles **mejoras y evoluciones**:

- Desarrollo de la página de inicio.
- Añadir la opción y lógica de guardado de partidas.
- Añadir la opción de ayuda, opciones y créditos a la pantalla de inicio.
- Añadir las criaturas que aparecen en las pantallas de transición, así como los elementos ocultos.

- Desarrollo de las pantallas de inventario (tienda de pelucas y club nocturno)
- Desarrollo de las mecánicas de:
  - Lógica de adquisición y uso de las pelucas.
  - Lógica de recompensas monetarias y contabilización.
- Añadir un nivel más (el jefe final) y posteriormente podrían añadirse más niveles intermedios, cambiando la progresión y hacer que la historia sea aún menos lineal y con más elementos narrativos que hagan que la historia sea más interesante.
- Implementar la posibilidad de pausar el juego en cualquier momento.
- Estadísticas de progresión en el juego.

10.3 Reflexión personal

En general, el resultado final del proyecto ha sido bastante **satisfactorio**.

La idea inicial, como se ha comentado, quizá era excesivamente ambiciosa y por limitaciones de tiempo ha habido que hacer ajustes, **simplificando y renunciando a algunos planteamientos iniciales**. En cualquier caso, este proyecto será ampliado buscando cumplir, en la medida de lo posible, con esa primera idea.

Además, como se ha explicado, la **programación de las mecánicas de los jefes**, en principio, estaban realizadas con **corrutinas** y durante el proceso se descubrió que era mucho más adecuado el uso de **programación por estados**, lo cual tuvo sus repercusiones en la implementación y en la parte de programación. En el trabajo se puede apreciar que el primer jefe está implementado de forma distinta a los otros dos.

La experiencia ha sido muy positiva y ha motivado un interés creciente en las opciones laborales existentes en el **mundo de la animación y los videojuegos**.

Uno de los aspectos más estimulantes de este proyecto ha sido el aprendizaje, ya que el punto de partida contemplaba algunos conceptos muy básicos respecto al desarrollo de videojuegos y se ha visto la necesidad de explorar **diferentes técnicas de programación** para asegurar el correcto funcionamiento del juego, descubriendo más a fondo las opciones que ofrece Unity para el desarrollo de videojuegos, lo que ha permitido adquirir nuevos conocimientos y habilidades.

Asimismo, al abordar los **aspectos creativos**, se han buscado herramientas adecuadas a las necesidades, como el uso de *Aseprite* para las animaciones y el uso de otro sistema operativo con el programa *GarageBand* para generar la banda sonora. Esta experiencia une dos mundos atractivos: el **desarrollo de videojuegos** y la faceta creativa de **elaborar guiones**, crear mundos imaginarios y la **animación de personajes digitales** en 2D.

Capítulo 11. Bibliografía

[1] Konvoy, "The Era of the Indie Game" https://www.konvoy.vc/content/the-era-of-the-indie-game [8/3/2024 Online]

[2] Ioana Grigorescu, "Vender su juego independiente en Internet: La guía completa" https://blog.payproglobal.com/es/selling-indie-game-online [3/3/2023 Online]

[3] Ioana Cozma - inBeat Agency. "Influencer Marketing for Video Games: The Complete Guide" https://inbeat.agency/blog/video-game-influencer-marketing [2/2/2024 Online]

Conduit Virtual, Inc. "Leveraging Mobile Game Influencer Marketing to Boost Your Game's Reach and Engagement" https://www.conduit.gg/es/blog/posts/leveraging-mobile-game-influencer-marketing-to-boost-your-game-s-reach-and-engagement [14/06/2024 Online]

[4] Daniel Perez - Shacknews. "Enter the Gungeon sells over 200k copies during launch week" https://www.shacknews.com/article/94078/enter-the-gungeon-sells-over-200k-copies-during-launch-week [12/4/2016 Online]

[5] William D'Angelo. "Enter the Gungeon Sales Top 3 Million Units Sold – Sales" https://www.vgchartz.com/article/442020/enter-the-gungeon-sales-top-3-million-units-sold/ [12/4/2020 Online]

[6] Paula García. "Dead Cells supera los 10 millones de ventas y planea seguir sacando contenido hasta 2025" https://www.eurogamer.es/dead-cells-supera-los-10-millones-de-ventas-y-planea-seguir-sacando-contenido-hasta-2025 [5/4/2024 Online]

[7] Wikipedia. "The binding of Isaac" https://es.wikipedia.org/wiki/The_Binding_of_Isaac [Online]

[8] Nicalis, Inc. "The Binding of Isaac: Rebirth" https://www.xbox.com/en-US/games/store/the-binding-of-isaac-rebirth/BQD45PQR4F4J [24/7/2015 Online]

[9] Sergio González. "Hades supera el millón de unidades vendidas; Supergiant Games celebra su éxito" https://as.com/meristation/2020/09/21/noticias/1600666237_052587.html [21/9/2020 Online]

[10] Alex Walker. "Hollow Knight Has Sold Over 2.8 Million Copies" https://www.kotaku.com.au/2019/02/hollow-knight-has-sold-over-2-8-million-copies/ [15/2/2019 Online]

[11] Mr. Platino. "Análisis «Hollow Knight», un metroidvania en miniatura" https://bigot.es/reviews/2019/07/20/analisis-hollow-knight-un-metroidvania-en-miniatura/ [20/7/2019 Online]

[12] Unity Documentation. https://docs.unity3d.com/es/2019.4/Manual/Console.html [2020 Online]

[13] Script Serialization Unity Technologies. https://docs.unity3d.com/es/2019.4/Manual/script-Serialization.html [15/05/2017 Online]

[14] envatotuts+. Como Guardar y Cargar el progreso del jugador en Unity. https://code.tutsplus.com/es/como-guardar-y-cargar-el-progreso-del-jugador-en-unity--cms-20934t [Online]

[15] Xsolla Funding. “An indie developer's guide to industry events, trade shows, and exhibitions” https://www.gamedeveloper.com/marketing/an-indie-developer-s-guide-to-industry-events-trade-shows-and-exhibitions [05/07/2023 Online]

[16] “Preguntas frecuentes” https://developers.xsolla.com/es/doc/faq/ [Online]

[17] dgraal. “Indie Game Budgeting: Smart Finance Tips” https://www.game-developers.org/indie-game-budgeting-smart-finance-tips [Online]

[18] Games jobs direct. https://www.gamesjobsdirect.com [Online]

[19] Enrique García. “Conseguir clasificación de PEGI cuesta más de 1.000 euros” https://as.com/meristation/2014/04/29/noticias/1398762540_130965.html [03/07/2017 Online]

[20] Steam. “Comprar Steam Direct Product Submission Fee” https://store.steampowered.com/sub/163632?l=latam [Online]

[21] Chris’ Tutorials. “2D Platformer Crash Course in Unity 2022” https://www.youtube.com/playlist?list=PLyH-qXFkNSxmDU8ddeslEAtnXIDRLPd_V [2023 Online]

[22] iVisual. “Cómo redactar un GDD para Videojuegos” https://ivisualformacion.com/blog/tutoriales/como-redactar-gdd-videojuegos [22/06/2022 Online]

[23] El Documentalista Audiovisual. “Documentación en Videojuegos: Documento de diseño (GDD)” https://eldocumentalistaudiovisual.com/2015/02/06/documentacion-en-videojuegos-documento-de-diseno-gdd/ [06/02/2015 Online]