Institut Puig Castellar
Santa Coloma de Gramenet

# ROUGE

**(Projecto de desarrollo)**
CFGS Desenvolupament d'Aplicacions Web

**Autores:** Jalel Jordan y Mateo Valdi Salva
**Grupo** A y B, **Curso académico** 2 DAW

## COPYRIGHT



**Resumen del proyecto**

El proyecto consiste en la creación de un videojuego de estilo roguelike ambientado en la época medieval, situado en un misterioso castillo del que no será posible escapar salvo que se cumplan determinadas condiciones. La propuesta surge a partir de la idea de combinar dos géneros de videojuegos: el roguelike y el metroidvania, tomando como inspiración las obras de Edmund McMillen y Team Cherry. Dichos títulos sirvieron como referencia para desarrollar un videojuego con una estética y enfoque similares, centrado en el desafío de dominar la jugabilidad y descubrir los secretos de la historia.

Asimismo, se desarrollará una página web desde la cual será posible descargar el juego, visualizar un tráiler, conocer información sobre la historia y consultar distintos detalles relacionados con el proyecto.

Con este proyecto se espera crear una página web funcional y dinámica para los usuarios, garantizar que la descarga del videojuego se realice sin fallos y conseguir que la versión final del juego represente de forma fluida la idea planteada inicialmente.

**Palabras clave:**

Roguelike

Metroidvania

Edmund McMillen

Team Cherry

Hollow Knight

The Binding of Isaac

**Abstract :**

## English

The project is about creating a roguelike-style video game set in the medieval era, inside a mysterious castle from which one cannot escape unless certain conditions are met. Players will explore a vast world surrounding the castle, encountering other people who dared to enter it. The project arose from the idea of combining two video game genres: roguelike and metroidvania, drawing inspiration from the works of Edmund McMillen (The Binding of Isaac) and Team Cherry (Hollow Knight). These games inspired the creation of a video game set in a similar style, focused on the challenge of mastering gameplay and unraveling the secrets of the story. Additionally, a website will be developed where the game can be downloaded, a trailer viewed, the story read, and game details consulted.

The results we expect from this project are to create a functional and dynamic website for the user. We also aim to ensure that downloading the game is free of errors, and that the final game is a smooth version of what we wanted to represent.

## Catalán

El projecte consisteix en la creació d'un videojoc d'estil roguelike ambientat en l'època medieval, situat en un misteriós castell del qual no serà possible sortir excepte si es compleixen determinades condicions. També es descobrirà un ampli món al voltant del castell, habitat per altres persones que s'han atrevit a entrar-hi. La proposta sorgeix amb la idea de combinar dos gèneres de videojocs: el roguelike i el metroidvania, prenent com a inspiració les obres d'Edmund McMillen i Team Cherry. Aquests títols van servir com a referència per desenvolupar un videojoc amb una estètica i un enfocament similars, centrat en el repte de dominar la jugabilitat i descobrir els secrets de la història.

A més, es desenvoluparà una pàgina web des de la qual serà possible descarregar el joc, visualitzar un tràiler, llegir informació sobre la història i consultar diferents detalls relacionats amb el projecte.

Amb aquest projecte s'espera crear una pàgina web funcional i dinàmica per als usuaris, garantir que la descàrrega del videojoc es realitzi sense errors i aconseguir que la versió final del joc representi de manera fluida la idea plantejada inicialment.

## Índex

# 1. Introducción

Este proyecto nace de la idea de dos estudiantes apasionados por los videojuegos, especialmente del género indie como los roguelike, metroidvania y plataformas. Inspirados por juegos como The Binding of Isaac y Hollow Knight, decidimos crear nuestro propio videojuego combinando estos estilos.

El objetivo principal es desarrollar un juego propio desde cero, a pesar de no tener experiencia previa en el diseño de videojuegos o personajes. Además, el proyecto incluye la creación de una página web donde se mostrará información sobre el desarrollo, características y avances del juego.

## 1.1 Contexto

La idea surge al analizar los videojuegos mencionados anteriormente y observar sus mecánicas principales. Nos planteamos la posibilidad de combinar estos elementos para crear un juego original con identidad propia.

Actualmente, la industria del videojuego indie está en crecimiento, gracias a plataformas como Kickstarter, que permiten a pequeños desarrolladores dar a conocer sus proyectos. Esto nos motivó aún más a iniciar nuestro propio desarrollo.

## 1.2 Justificación

Este proyecto tiene como objetivo recuperar la esencia de los videojuegos clásicos, donde la jugabilidad y la historia son los elementos más importantes.

Además, se trata de un proyecto formativo que nos permitirá:

- Aprender a desarrollar un videojuego desde cero
- Mejorar nuestras habilidades de programación
- Trabajar en equipo
- Conocer herramientas reales del sector

## 1.3 Objetivos

### *1.3.1 Objetivo general*

Desarrollar un videojuego funcional, optimizado y entretenido, junto con una página web informativa y atractiva. Que el usuario se sienta cómodo usando la página para saber más información sobre el juego y también de quiénes somos los creadores.

### *1.3.2 Objetivos específicos*

- Diseñar las mecánicas principales del juego.
- Implementar un sistema tipo roguelike (una vida por partida).
- Crear niveles con exploración tipo metroidvania.
- Desarrollar controles y físicas de plataformas.
- Diseñar una interfaz sencilla e intuitiva.
- Crear una página web del proyecto.
- Documentar todo el proceso de desarrollo.
- Aprender nuevas herramientas y tecnologías.
    - 

## 1.4 Metodología de trabajo

Para el desarrollo del proyecto se ha utilizado una metodología ágil basada en Scrum, que permite trabajar de forma flexible y organizada.

El trabajo se ha dividido en pequeñas fases llamadas *sprints*, donde se desarrollan diferentes partes del juego y se revisan de forma continua.

### *1.4.1 Herramientas utilizadas:*

- **Trello / Jira** → organización de tareas
- **Git** → control de versiones del código

Esta metodología nos ha permitido adaptarnos a cambios y mejorar el proyecto progresivamente.

## 1.5 Presupuesto del proyecto

### *Recursos materiales*

| Concepto | Detalle | Coste unitario (€) | Cantidad | Coste total (€) |
|---|---|---|---|---|
| Ordenadores | Equipos de desarrollo | 1.750 € | 2 | 3.500 € |
| Monitores | Pantallas | 80 € | 2 | 160 € |
| Teclados | Periféricos | 10 € | 2 | 20 € |
| Ratones | Periféricos | 10 € | 2 | 20 € |
| Sillas | Mobiliario | 20 € | 2 | 40 € |
| Escritorios | Mobiliario | 50 € | 2 | 100 € |

**Coste total = 3.840 €**

# 2. Descripción del proyecto

## 2.1 Requisitos del sistema

### *2.1.1 Requisitos funcionales*

Los requisitos funcionales que requieren el juego y la página web serían los siguientes:

- El jugador puede controlar un personaje con movimiento lateral, salto y ataque.
- El juego dispone de un mapa explorable con diferentes zonas conectadas entre sí.
- Generación de escenarios (elementos tipo roguelike).
- Existencia de enemigos con diferentes comportamientos y dificultad.
- Sistema de combate basado en vida y daño.
- Muerte del jugador con consecuencias significa fin de la partida completamente.
- Sistema de progresión del personaje (mejoras u objetos).
- Posibilidad de descargar el juego desde la página web.
- Poder acceder a la página web y obtener información relevante sobre el juego.
- Poder crear un usuario y que reciba información mediante mails de actualización y avisos.

### *2.1.2 Requisitos no funcionales*

Los requisitos no funcionales definen la calidad y características técnicas del sistema:

- Jugabilidad fluida y controles responsivos.
- Diseño visual coherente con una estética medieval y oscura.
- Música y efectos de sonido inmersivos.
- Tiempos de carga reducidos.
- Interfaz clara e intuitiva.
- Compatibilidad con sistemas operativos habituales (Windows).
- Código bien estructurado, organizado y mantenible.
- Cumplimiento de la licencia Creative Commons utilizada en el proyecto.

## 2.2 Tecnologías utilizadas

Después de establecer la organización en el apartado de funcionalidades generales, se inició la búsqueda de herramientas y tecnologías adecuadas para el desarrollo del proyecto.

Para el apartado del videojuego, se llevó a cabo un proceso de análisis y debate con el fin de determinar cuál de los siguientes motores de desarrollo resultaba más adecuado para el proyecto:

### *2.2.1 Motores de videojuegos evaluados*

## Unity

- **Gran comunidad:** Dispone de una comunidad muy amplia, lo que facilita encontrar tutoriales, documentación y soluciones a problemas.
- **Multiplataforma:** Permite exportar juegos a múltiples sistemas operativos como Windows, Android, iOS o consolas.
- **Flexibilidad:** Se adapta a diferentes tipos de proyectos, desde juegos sencillos hasta desarrollos más complejos.
- **Lenguaje de programación:** Utiliza C#, lo que facilita una buena organización del código.

**Contras:**

- Mayor consumo de recursos en comparación con otros motores más ligeros.

## Godot

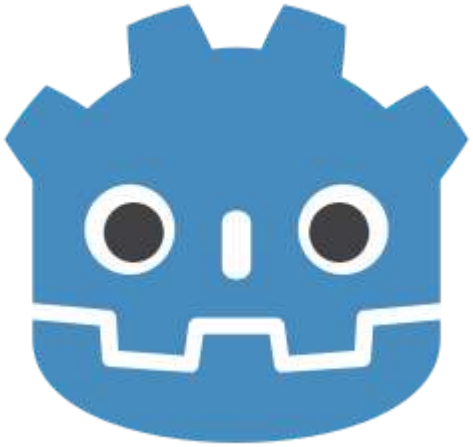
Godot es un motor de videojuegos de código abierto que ha ganado popularidad en los últimos años, especialmente en el desarrollo de juegos 2D.

- **Código abierto:** Completamente gratuito y sin licencias, lo que permite modificar el motor si es necesario.
- **Ligero:** Consume pocos recursos, lo que facilita su uso en equipos menos potentes.
- **Orientado a 2D:** Ofrece herramientas muy eficientes para el desarrollo de juegos en dos dimensiones.

- **Facilidad de uso:** su estructura es sencilla y rápida de aprender.

**Contras:**

- Menor comunidad y recursos en comparación con Unity.



## Game maker

GameMaker es un motor muy enfocado a principiantes y al desarrollo de juegos 2D, con un entorno visual sencillo.

- **Fácil de utilizar:** ideal para iniciarse en el desarrollo de videojuegos.
- **Especializado en 2D:** permite crear juegos de este tipo de forma rápida.
- **Entorno intuitivo:** incluye herramientas visuales que simplifican el desarrollo.

**Contras:**

- Requiere licencia de pago para acceder a todas sus funcionalidades.
- Menor flexibilidad en proyectos más complejos.


### *2.2.2 Tecnologías frontend evaluadas*

## HTML/CSS con JavaScript

HTML, CSS y JavaScript forman la base del desarrollo web estándar. Es la combinación más extendida para crear interfaces de usuario sin depender de frameworks externos.

*Pros:*

- Universalmente compatible: Funciona en cualquier navegador moderno sin necesidad de instalaciones adicionales.
- Sin dependencias externas: No requiere librerías ni frameworks, lo que reduce la complejidad del proyecto.
- Fácil de mantener: Al ser tecnologías ampliamente conocidas, cualquier desarrollador puede retomar el código sin curva de aprendizaje.
- Rendimiento ligero: Al no cargar frameworks pesados, las páginas responden más rápido.

*Contras:*

- Puede volverse difícil de escalar si el proyecto crece mucho en complejidad.
- Requiere más código manual para tareas que frameworks como React o Vue resuelven de forma automática.

## React

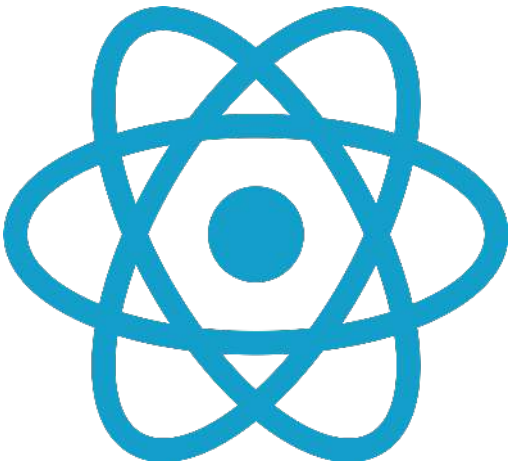
React es una librería de JavaScript desarrollada por Meta, orientada a la construcción de interfaces de usuario mediante componentes reutilizables.

*Pros:*

- Componentización: Permite dividir la interfaz en piezas independientes y reutilizables, facilitando el mantenimiento.
- Gran comunidad: amplia documentación, tutoriales y soporte activo.
- Virtual DOM: Mejora el rendimiento al actualizar solo los elementos que cambian en la página.
- Ecosistema amplio: Integración sencilla con librerías como React Router, Redux o Next.js.

*Contras:*

- Curva de aprendizaje: Requiere familiarizarse con JSX y conceptos como hooks o estados.
- Mayor complejidad inicial: para proyectos pequeños puede ser excesivo.



## Vue.js

Vue.js es un framework progresivo de JavaScript diseñado para ser adoptado de forma incremental, desde proyectos sencillos hasta aplicaciones completas.

*Pros:*

- Fácil de aprender: Su sintaxis es más cercana al HTML estándar que la de otros frameworks.
- Progresivo: Se puede integrar en partes de un proyecto sin necesidad de reescribir todo.
- Documentación muy clara: considerada una de las mejores entre los frameworks modernos.
- Ligero: Tiene un tamaño de bundle reducido en comparación con Angular.

*Contras:*

- Comunidad más pequeña que React, especialmente en proyectos a gran escala.
- Menos demanda en el mercado laboral respecto a React.

## Angular

Angular es un framework completo desarrollado por Google, pensado para aplicaciones web de gran envergadura con arquitectura robusta.

*Pros:*

- Framework completo: incluye todo lo necesario (enrutamiento, formularios, HTTP, etc.) sin necesidad de librerías adicionales.
- TypeScript por defecto: Favorece un código más seguro y estructurado.

- Ideal para proyectos grandes: su arquitectura basada en módulos facilita el trabajo en equipos grandes.

*Contras:*

- Curva de aprendizaje elevada: es el más complejo de los tres frameworks comparados.
- Excesivo para proyectos pequeños o medianos.
- Mayor tiempo de configuración inicial.

### *2.2.3 Tecnologías seleccionadas*

Para el desarrollo web vamos a usar la tecnología de **React** y para el desarrollo de la API para el manejo de usuarios y actividades, lo que vamos a usar es la tecnología de **Spring Boot**. Para el almacenamiento de datos usaremos la tecnología de **MySQL.** Para darle diseño y que quede más bonito, usaremos la tecnología de **CSS**.

#### 2.2.3.1 Frontend

**React:**

Nos hemos decantado por React como tecnología para el desarrollo del frontend por varias razones. En primer lugar, su arquitectura basada en componentes reutilizables nos permite estructurar la interfaz de forma modular, lo que facilita tanto el desarrollo como el mantenimiento del código a medida que el proyecto crece. En segundo lugar, la integración natural con JavaScript agiliza la lógica de la interfaz sin necesidad de herramientas adicionales.

Otro factor decisivo fue la facilidad para consumir APIs REST. React, combinado con herramientas como `fetch` o `axios`, simplifica enormemente las llamadas a los endpoints del backend, permitiendo gestionar las respuestas.

de forma eficiente mediante el sistema de estados y hooks como `useEffect` y `useState`.

Además, su amplia comunidad y documentación garantizan que cualquier duda o problema técnico tenga solución accesible, lo cual es especialmente valioso en un equipo con diferentes niveles de experiencia.

### CSS:

Para el diseño visual de la aplicación web utilizaremos CSS, el lenguaje estándar para aplicar estilos en entornos web. Aunque React gestiona la estructura y la lógica de la interfaz, CSS es el encargado de definir la apariencia visual: colores, tipografías, espaciados, animaciones y adaptación a diferentes tamaños de pantalla.

En concreto, haremos uso de CSS junto con la librería **Tailwind CSS**, que ofrece clases utilitarias predefinidas que agilizan considerablemente el proceso de maquetación sin necesidad de escribir hojas de estilo desde cero. Esto nos permite mantener un diseño consistente a lo largo de toda la aplicación y reducir el tiempo dedicado al apartado visual.

Entre sus ventajas destacamos su alta compatibilidad con React, la facilidad para implementar diseño responsive adaptado a distintos dispositivos, y la posibilidad de personalizar completamente la apariencia sin depender de componentes visuales cerrados como los de Bootstrap.

## 2.2.3.2 Backend

### Spring Boot:

Para el desarrollo del backend y la API REST hemos elegido Spring Boot, un framework de Java orientado a la creación de aplicaciones backend de forma rápida y estructurada.

La principal razón de esta elección es su configuración mínima: Spring Boot elimina gran parte del código repetitivo que requeriría un proyecto Java tradicional, permitiendo centrarse en la lógica de negocio. Ofrece una arquitectura clara basada en capas (controlador, servicio y repositorio) que facilita la organización del código y el trabajo en equipo.

También resulta especialmente adecuado para construir APIs REST, ya que incluye de forma nativa herramientas para gestionar rutas, autenticación, validación de datos y conexión con bases de datos a través de Spring Data JPA. Su integración con sistemas de seguridad como Spring Security permite implementar autenticación de usuarios de manera robusta y escalable.

Por último, al estar basado en Java, un lenguaje ampliamente conocido por el equipo, la curva de aprendizaje se reduce considerablemente frente a otras alternativas como Node.js o Django.


### 2.2.3.3 Base de datos

**MySQL:**

Para el almacenamiento de datos se ha optado por MySQL como sistema gestor de bases de datos relacional. Esta decisión se basa en varios factores.

En primer lugar, MySQL es una tecnología madura, ampliamente utilizada en proyectos web y con una comunidad muy activa, lo que garantiza abundante documentación y soporte ante cualquier problema. En segundo lugar, su naturaleza relacional se adapta perfectamente a la estructura de nuestros datos, ya que entidades como usuarios, actividades o puntuaciones presentan relaciones claras entre sí que se modelan de forma natural mediante tablas y claves foráneas.

Además, su integración con Spring Boot a través de Spring Data JPA es directa y bien documentada, lo que simplifica enormemente las operaciones de consulta, inserción y actualización de datos sin necesidad de escribir SQL manualmente en la mayoría de los casos.

Por último, MySQL es gratuito en su versión Community, lo que se ajusta a las necesidades y recursos de nuestro proyecto.


### 2.2.3.4 Videojuego

**Unity:**

Para el desarrollo del videojuego se ha elegido Unity como motor principal, y esta decisión se apoya en varios argumentos sólidos.

El factor más determinante ha sido su gran comunidad y ecosistema. Unity cuenta con una de las comunidades más grandes del sector, lo que se traduce en una enorme cantidad de tutoriales, foros, assets gratuitos en la Asset Store y soluciones documentadas ante cualquier problema técnico. Esto es especialmente valioso en un equipo que compagina distintos niveles de experiencia en desarrollo de videojuegos.

En segundo lugar, Unity ofrece un soporte muy completo para juegos 2D, que es la modalidad que hemos elegido para nuestro proyecto. Dispone de herramientas específicas como el sistema de Tilemaps para construir escenarios, el motor de físicas 2D integrado, el sistema de animaciones mediante Animator y Sprite sheets, y una cámara ortográfica optimizada para este tipo de juegos.

Otro aspecto relevante es su capacidad multiplataforma. Unity permite exportar el juego a múltiples plataformas como Windows, Android o iOS con cambios mínimos en el código, lo que abre la posibilidad de ampliar el alcance del proyecto en el futuro sin tener que reescribirlo desde cero.

Por último, su editor visual intuitivo facilita la organización de escenas, la gestión de objetos y la configuración de componentes sin necesidad de programar absolutamente todo, lo que agiliza el flujo de trabajo del equipo.

**C#:**

C# es el lenguaje de programación oficial de Unity y, por tanto, la opción natural para el desarrollo del juego. Sin embargo, más allá de esta integración directa, existen razones adicionales que refuerzan esta elección.

C# es un lenguaje orientado a objetos, fuertemente tipado y con una sintaxis clara y estructurada. Esto favorece la organización del código en proyectos con varios desarrolladores, ya que obliga a definir tipos de datos de forma explícita y reduce la aparición de errores difíciles de detectar en tiempo de ejecución.

Además, su similitud con otros lenguajes ampliamente conocidos como Java facilita la adaptación de los miembros del equipo que ya tienen experiencia en programación orientada a objetos, reduciendo así la curva de aprendizaje inicial.

C# también ofrece características modernas del lenguaje como LINQ, programación asíncrona con `async/await`, y un sistema de gestión de memoria automático mediante el recolector de basura, lo que permite centrarse en la lógica del juego sin preocuparse por la gestión manual de memoria, a diferencia de lenguajes como C++.

Por último, la documentación oficial de Unity está íntegramente orientada a C#, lo que garantiza que todos los ejemplos, tutoriales y referencias del motor sean directamente aplicables a nuestro proyecto.

## 2.3 Definición de funcionalidades

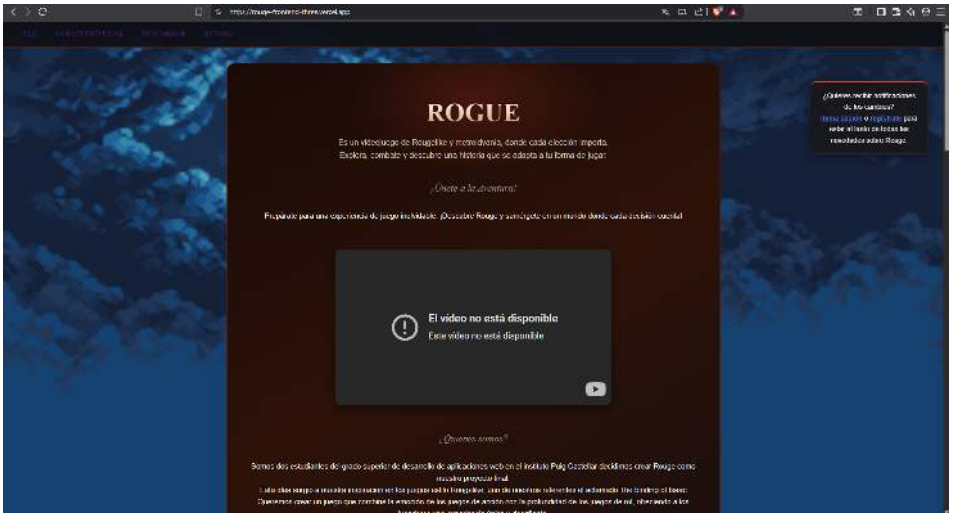
Las funcionalidades describen cómo se implementan los requisitos del proyecto, es decir, qué puede hacer el sistema y cómo lo hace.

- **Movimiento del personaje:** El jugador podrá moverse lateralmente, saltar e interactuar con el entorno mediante controles simples.
- **Sistema de combate:** El personaje podrá atacar enemigos y recibir daño, con una barra de vida visible.
- **Exploración:** el mapa estará dividido en zonas conectadas, algunas accesibles solo tras desbloquear habilidades.
- **Generación roguelike:** Ciertos elementos del mapa cambiarán en cada partida, aumentando la rejugabilidad.
- **Progresión del personaje:** El jugador podrá desbloquear habilidades y mejoras que afectarán a la jugabilidad.
- **Narrativa:** La historia se descubrirá progresivamente mediante NPCs, escenarios y textos.
- **Página web del proyecto:** Incluirá información del juego, historia, tráiler y enlace de descarga.

## 2.4 Descripción de los componentes web con sus funcionalidades

### *2.4.1 Parte cliente:*

**HOME:**

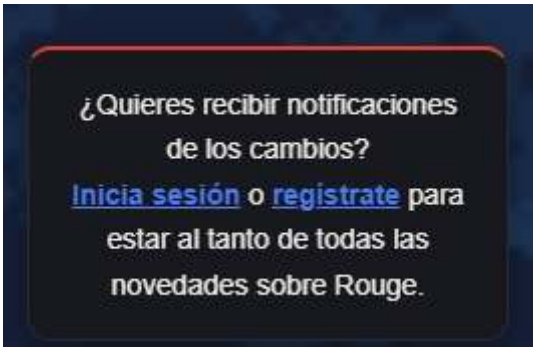



La pantalla de inicio, donde el cliente/usuario podrá navegar libremente y revisar información sobre el proyecto, teniendo la posibilidad de descargar.

**Header:**

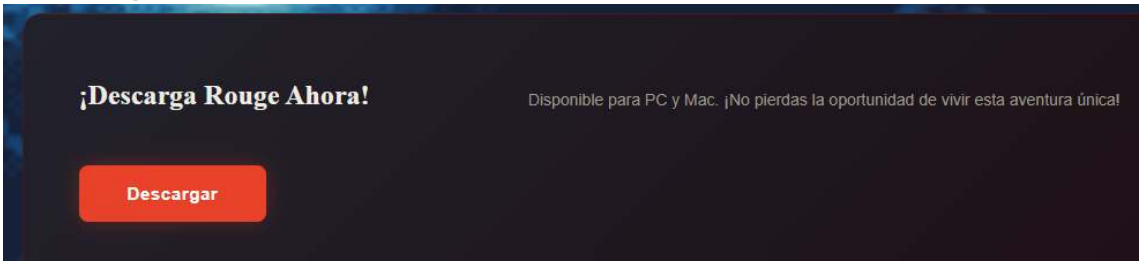

Este componente permitirá hacer una navegación más rápida por la única página particionada por secciones.

## Pop-up:



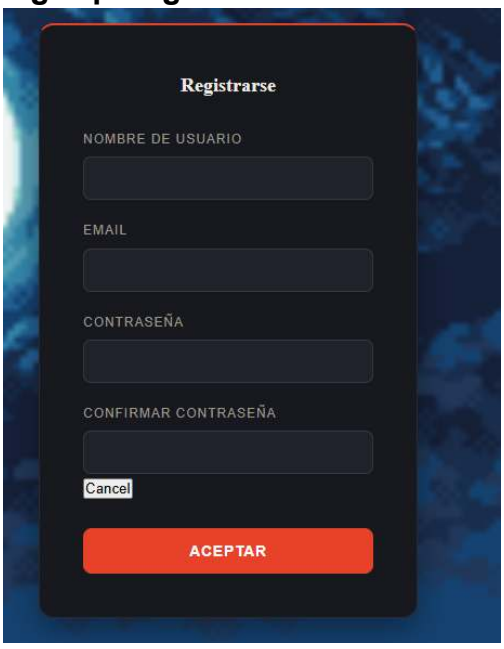

Este pop-up funciona para darle noción al usuario de que puede hacer un login o un sign-up para recibir notificaciones vía mails por eso mismo, el requisito de tener creado como mínimo un usuario creado para ya tener un registro.

## Descarga:



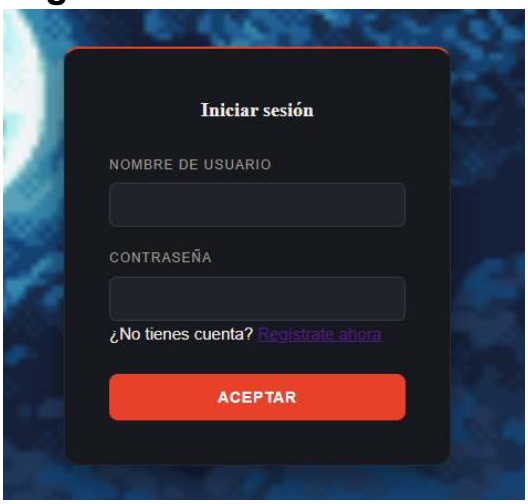

Este botón te permitirá tener acceso a nuestro proyecto descargando el setup del juego; esto no te redirige a ninguna página, sino que te mantiene dentro de la misma y te salta una ventana de Windows para determinar dónde poder guardar o dónde almacenar el setup en tu propio equipo. Todo esto pasa gracias a la pasarela que está hecha desde el backend.

## SignUp/Registrarse:



Este componente ya involucra el uso del una gran parte del backend permitiendo crearnos un usuario a partir de unos datos proporcionados por el frontend (los datos que se ingresan desde la imagen) que a la vez se guardan dentro de una base de datos.

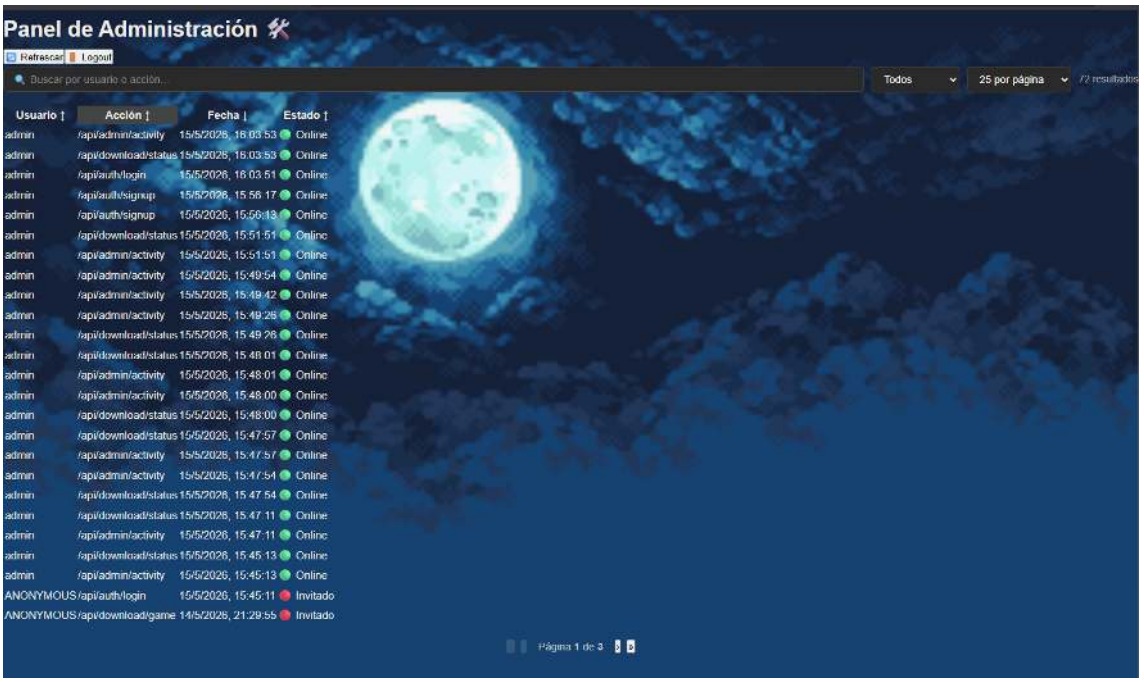
## Login/Iniciar sesión:




Aquí podremos llamar a la API del proyecto para que nos permite iniciar sesión con los datos correspondientes al usuario registrado dentro de la base de datos Dependiendo qué rol tenga el usuario puede llegar a entrar a la parte superior del sistema para revisar errores o actividades sospechosas

### *2.4.2 Parte Administración*

**Panel Admin:**




Este panel solamente es accesible teniendo una cuenta admin o un usuario con el rol de administrador, este panel lo que tiene de objetivo es revisar las actividades por sesión de usuarios registrados o los que están viendo sin registrarse.

## 2.5 Planificación del proyecto

La planificación se basa en una metodología ágil (Scrum/Kanban), que permite organizar el trabajo de forma flexible.

### *2.5.1 Organización del trabajo:*

- Uso de un tablero Kanban para visualizar tareas
- División del proyecto en sprints cortos
- Revisión continua del progreso
- Reasignación de tareas según necesidades

Esta metodología permite adaptarse a cambios durante el desarrollo y mejorar el proyecto de forma progresiva.

## 2.6 Desarrollo del videojuego

### *2.6.1 Planificación del desarrollo*

El desarrollo del videojuego comenzó con la creación del proyecto en Unity, entorno en el que se llevó a cabo la totalidad del desarrollo. Desde el inicio, el objetivo principal consistía en crear un videojuego rejugable y entretenido. Aunque las mecánicas fueron evolucionando a lo largo de las distintas etapas del desarrollo, siempre se mantuvo la misma base y estilo planteados inicialmente para el proyecto.

### *2.6.2 Desarrollo del menú principal*

La primera fase del desarrollo estuvo centrada en el diseño del menú principal, el cual incluía el título del videojuego, un botón que permitía acceder al menú de opciones y otro destinado a cerrar la aplicación.


### *2.6.3 Desarrollo del menú de opciones*

Posteriormente, se desarrolló la escena de opciones, que incorporaba la posibilidad de activar o desactivar el audio del juego, así como modificar el nivel de brillo. Para ello, se implementaron distintos botones: dos destinados al control del audio y cuatro encargados de ajustar el brillo en porcentajes comprendidos entre el 25 % y el 100 %. El funcionamiento de estas opciones fue programado mediante el archivo Opciones.cs, desarrollado en lenguaje C#.


Con el objetivo de permitir la transición entre escenas, como el paso del menú principal al menú de opciones, se creó el archivo **Escenas.cs**. Este script hacía uso de la librería **UnityEngine.SceneManagement**, necesaria para la gestión de escenas dentro del proyecto. Su funcionamiento consistía en la creación de funciones específicas destinadas a cargar cada una de las escenas disponibles en el videojuego. Asimismo, para que una escena pudiera utilizarse correctamente, era necesario añadirla previamente al apartado Build Settings de Unity.

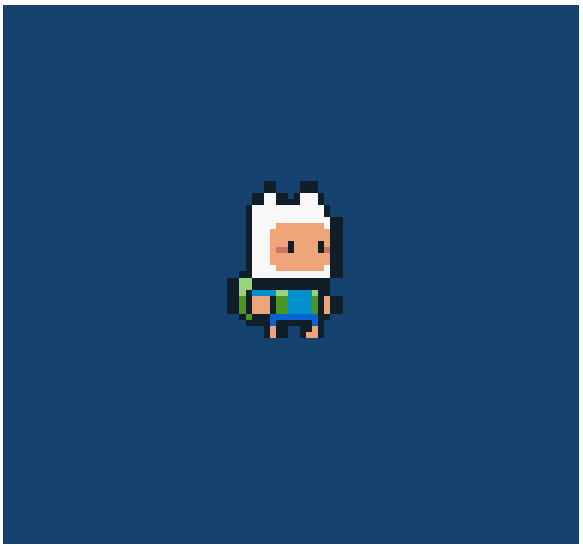
### *2.6.4 Desarrollo del nivel*

A continuación, se inició el desarrollo de la parte principal del videojuego: el nivel, el personaje, el mapa, los enemigos y las diferentes funcionalidades asociadas. En primer lugar, se configuró la escena para incorporar gravedad, un personaje controlable por el jugador y el terreno interactivo correspondiente al castillo. Una vez establecidas las bases del proyecto, el desarrollo se centró en el movimiento del personaje, incluyendo desplazamiento lateral, salto y ataque. Para ello, se utilizaron los archivos **MovimientoPersonaje.cs, HUD.cs, GameManager.cs, Vida.cs y AtaquePersonaje.cs.**

### *2.6.5 Desarrollo del personaje*

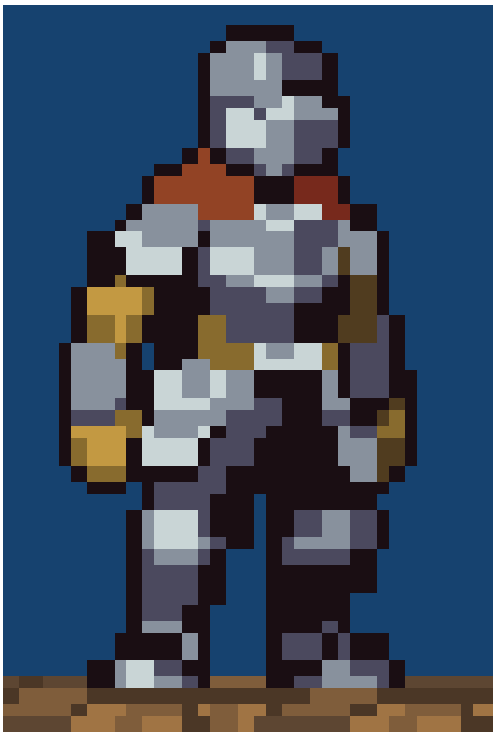
Estos archivos se encargaban de gestionar el movimiento del personaje y sus animaciones, el sistema de vida, las mecánicas de ataque y la interfaz HUD, donde el usuario podía visualizar tanto las vidas disponibles como la cantidad de oro acumulada.

Inicialmente, el personaje utilizado correspondía a un asset provisional implementado únicamente con fines de prueba y verificación del funcionamiento básico del sistema.



### *2.6.6 Desarrollo de las animaciones del personaje*

Una vez comprobado el correcto funcionamiento del movimiento, se incorporó el personaje definitivo y se comenzó el desarrollo de sus animaciones.



Para ello, se emplearon los archivos **MovimientoPersonaje.cs** y **GameManager.cs**. El proceso de animación consistía en combinar diferentes imágenes del personaje dentro del sistema de animaciones de Unity, definiendo posteriormente la duración y condiciones de ejecución de cada animación.

La primera animación desarrollada fue la animación idle, correspondiente al estado en el que el personaje permanece inmóvil. Esta animación estaba compuesta por seis imágenes que, reproducidas de forma secuencial, aportaban sensación de movimiento y vida al

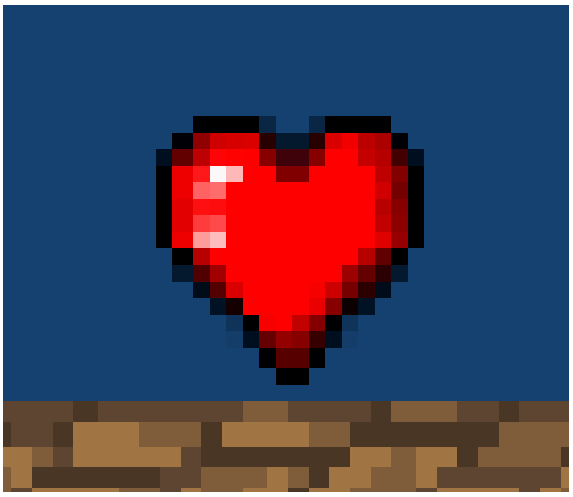
personaje. Posteriormente, mediante el sistema Animator de Unity, se configuraron las transiciones entre animaciones a través de un diagrama de flujo que determinaba en qué situaciones debía ejecutarse cada una. Este mismo procedimiento se aplicó también a las animaciones de los enemigos presentes en el videojuego.

### *2.6.7 Desarrollo de la vida del personaje*

En relación con el sistema de vida del personaje, participaron los archivos **HUD.cs**, **Vida.cs** y **GameManager.cs**. El sistema consistía en una cantidad fija de vidas definida desde **GameManager.cs** y representada visualmente en el HUD mediante imágenes seleccionadas para indicar al jugador el número de vidas restantes. Cada vez que el personaje recibía daño, una de estas imágenes se desactivaba visualmente para reflejar la pérdida de vida. Cuando todas las vidas se agotaban, el jugador moría.

En las primeras versiones del proyecto, la muerte del personaje provocaba simplemente la recarga automática de la escena. Sin embargo, posteriormente se implementó un overlay de derrota que aparecía sobre la pantalla y ofrecía al usuario la posibilidad de reiniciar el nivel o regresar al menú principal.


Inicialmente, las vidas estaban representadas mediante corazones. Más adelante, se decidió sustituirlas por calaveras, ya que estas encajaban mejor con la estética y ambientación general del videojuego.



En las primeras fases del desarrollo, el personaje disponía únicamente de cinco vidas. Más adelante, se añadió un sistema que permitía aumentar la vida máxima hasta un total de diez mediante la compra de mejoras en la tienda integrada dentro del menú de pausa.


### *2.6.8 Desarrollo del sistema de combate*

Posteriormente, se desarrolló el sistema de combate del personaje. Este permitía realizar ataques mediante la pulsación de una tecla, activando una animación en la que el personaje utilizaba su espada. Si durante el ataque se detectaba un enemigo dentro del área de impacto, este recibía daño hasta que su vida llegaba a cero. Para implementar esta funcionalidad se utilizaron los archivos **Enemigo.cs**, **EnemigoVida.cs** y **AtaquePersonaje.cs**.

A continuación, se desarrollaron los enemigos del videojuego, incluyendo su comportamiento, funcionamiento y parámetros. En esta fase participaron los archivos **Enemigo.cs**, **EnemigoVida.cs** y **EnemigoEstatico.cs**. Los enemigos disponían de una caja de colisión (HitBox) que permitía detectar el contacto con el personaje. Cuando el jugador colisionaba con un enemigo, perdía vida. Asimismo, los enemigos contaban con movimiento horizontal, animaciones y un sistema de vida que determinaba su muerte al llegar a cero.

### *2.6.9 Desarrollo de los enemigos*

En las primeras versiones, los enemigos eran completamente estáticos y actuaban únicamente como obstáculos. Más adelante, se incorporaron sistemas de movimiento, seguimiento del personaje, vida y muerte, además de las correspondientes animaciones.

Debido a la necesidad de añadir peligros adicionales dentro del escenario, se creó el archivo **EnemigoEstatico.cs**, una versión simplificada de **Enemigo.cs** sin funciones de movimiento. Este sistema permitió implementar trampas como pinchos capaces de dañar al jugador.

El primer enemigo desarrollado fue un slime, caracterizado por una velocidad lenta, un punto de vida y un amplio radio de detección del jugador.

Posteriormente, se creó el golem, con velocidad media, tres puntos de vida y un radio de detección intermedio.

Más adelante, se añadió el golem reforzado, caracterizado por una velocidad elevada, cinco puntos de vida y un radio de detección reducido.

Finalmente, se desarrollaron dos jefes principales: el Rey Slime y el Rey Golem.

El Rey Slime disponía de velocidad lenta, veinte puntos de vida y un radio de detección reducido.

Por su parte, el Rey Golem contaba con velocidad elevada, treinta puntos de vida y un radio de detección reducido.

Además de estos enemigos principales, también se implementaron diversos jefes slime situados en zonas ocultas del mapa cuya función consistía en bloquear el avance del jugador y delimitar determinadas áreas del escenario.

Los jefes slime contaban con velocidad lenta, diez puntos de vida y un radio de detección medio.

### *2.6.10 Desarrollo del sistema de monedas*

Posteriormente, se desarrolló el sistema de monedas y la tienda del videojuego. Este sistema consistía en diferentes tipos de monedas repartidas por el nivel que el jugador podía recoger y almacenar para posteriormente utilizarlas en la compra de mejoras, como el aumento de la vida máxima. Para ello, se utilizaron los archivos **GameManager.cs**, **Coin.cs** y **ShopManager.cs**.

Las monedas fueron implementadas como objetos trigger, permitiendo que el jugador las atravesara sin colisionar físicamente. Cuando el personaje entraba en contacto con una moneda, esta desaparecía y su valor se añadía automáticamente al contador mostrado en la interfaz.

Se implementaron cuatro tipos diferentes de monedas:

**Restos de slime**, con un valor de una moneda.

**Trozos de cobre**, con un valor de tres monedas.

**Trozos de oro**, con un valor de cinco monedas.

**Ojos de Ahriman**, con un valor de diez monedas.

### *2.6.11 Desarrollo del sistema de tienda*

Con el objetivo de dar utilidad al sistema de monedas, se desarrolló una tienda integrada dentro del menú de pausa. Esta tienda permitía adquirir mejoras para incrementar la vida máxima del personaje mediante el uso de monedas acumuladas. Su funcionamiento consistía en comprobar la cantidad de monedas disponibles y restar automáticamente el coste correspondiente al objeto adquirido, en este caso una poción de vida.

### *2.6.12 Implementación del sistema de seguimiento de cámara*

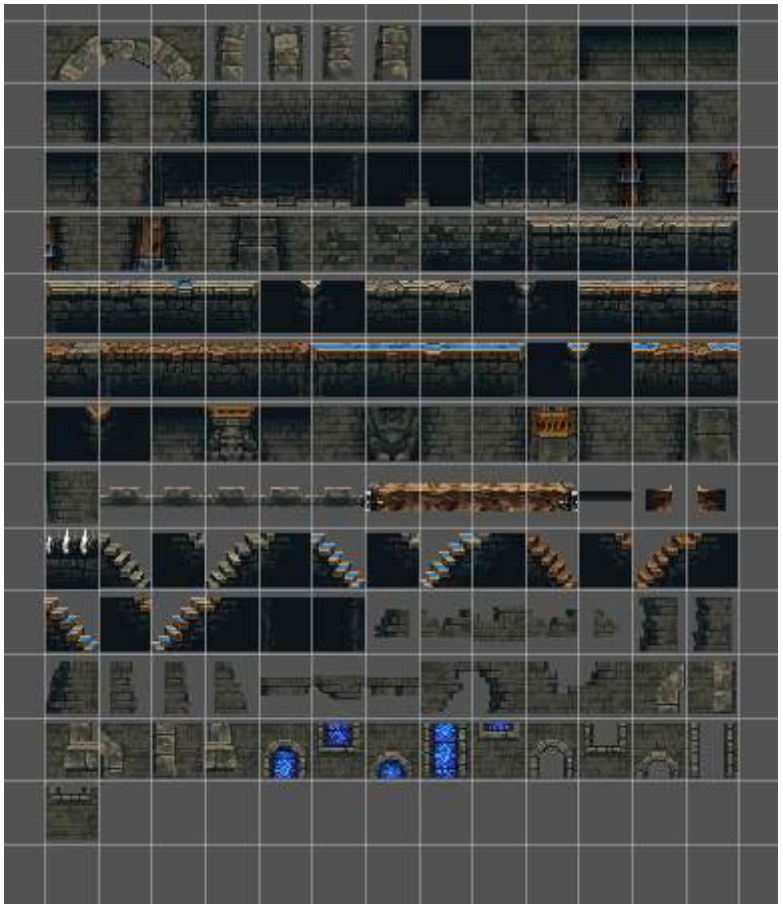
En las fases finales del desarrollo, se implementó un sistema de seguimiento de cámara mediante el archivo **CameraController.cs**. Este sistema garantizaba que la cámara permaneciera centrada en el personaje independientemente de su posición dentro del escenario.

### *2.6.13 Implementación del sistema de sonido al juego*

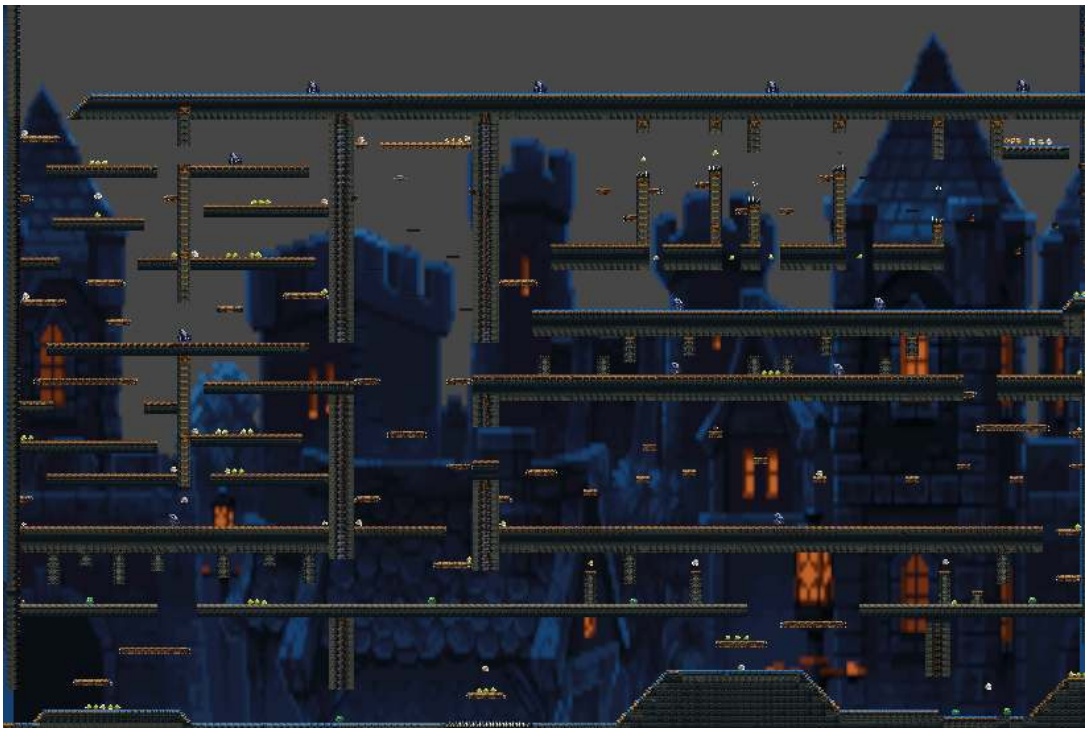
Una vez finalizadas las funcionalidades principales, se decidió añadir sonido al videojuego. Para ello, se incorporaron tres canciones diferentes, una destinada a cada escena del juego. El sistema de audio fue desarrollado mediante los archivos **AudioManager.cs** y **AudioGlobal.cs**, encargados de reproducir el audio correspondiente, controlar el volumen y gestionar la activación o desactivación global del sonido.

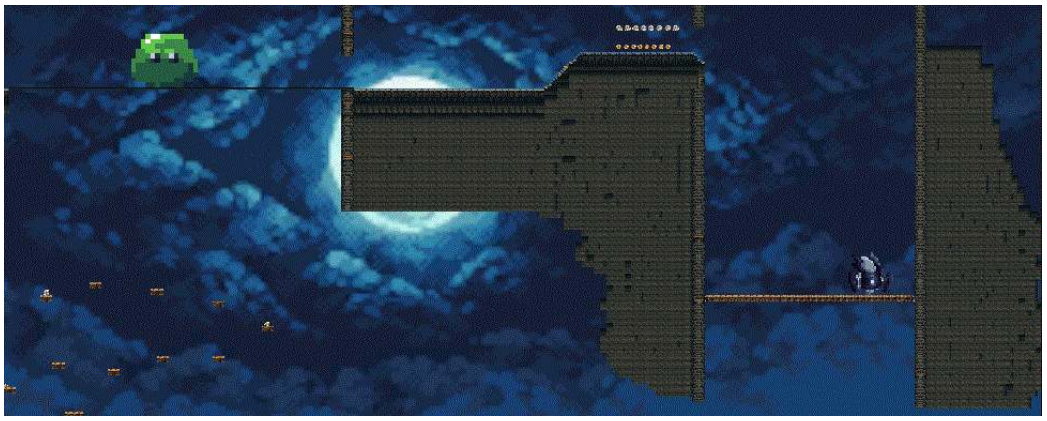
### *2.6.14 Creación del nivel*

Finalmente, se desarrolló el escenario completo del videojuego utilizando una Tile Palette, compuesta por diferentes assets con colisiones integradas que permitían construir el entorno jugable y facilitar el desplazamiento del personaje.



El resultado final fue un nivel completo en el que el jugador podía desplazarse, explorar, recoger objetos y enfrentarse a distintos enemigos.





### *2.6.15 Scripts utilizados*

A continuación, se presenta una lista con todos los scripts utilizados durante el desarrollo del proyecto, así como la funcionalidad y el apartado del videojuego en el que se aplican:

- `MovimientoPersonaje.cs` → Gestión del movimiento y las animaciones del personaje.
- `AtaquePersonaje.cs` → Control del sistema de ataque del jugador.
- `Vida.cs` → Administración de la vida del personaje.
- `HUD.cs` → Gestión de la interfaz de usuario y visualización de datos.
- `GameManager.cs` → Control general de la lógica principal del videojuego.
- `Opciones.cs` → Configuración de opciones de audio y brillo.
- `Escenas.cs` → Gestión de la transición entre escenas.
- `Enemigo.cs` → Control del comportamiento y movimiento de los enemigos.
- `EnemigoVida.cs` → Administración de la vida de los enemigos.
- `EnemigoEstatico.cs` → Gestión de enemigos estáticos y obstáculos dañinos.
- `Coin.cs` → Funcionamiento y recolección de monedas.
- `ShopManager.cs` → Gestión de la tienda y compra de mejoras.
- `CameraController.cs` → Seguimiento de la cámara respecto al personaje.
- `AudioManager.cs` → Reproducción y control de los efectos de sonido y música.
- `AudioGlobal.cs` → Gestión global del sistema de audio del videojuego.

# 3. Casos de uso

## 3.1 Caso de uso – Página web

**Nombre: Acceder y utilizar la página web del proyecto**

**Actor principal:**

Usuario (visitante de la web)

### Descripción:

El usuario accede a la página web del proyecto para obtener información sobre el videojuego, ver contenido multimedia y descargarlo.

### Flujo principal:

1. El usuario entra en la página web.
2. Visualiza el menú de navegación (FAQ, características, descargar, diseño).
3. Accede a las diferentes secciones informativas.
4. Consulta la información del juego (historia, características, estilo visual).
5. Puede ver contenido multimedia (imágenes o tráiler).
6. El usuario pulsa el botón **“Descargar”**.
7. Se inicia la descarga del videojuego.

### Flujos alternativos:

- 3a. El usuario accede directamente a una sección específica mediante el menú.
- 6a. El usuario no descarga el juego y solo consulta información.
- 6b. El usuario inicia sesión o se registra para recibir notificaciones.

### Resultado:

El usuario obtiene información del juego o lo descarga correctamente.

## 3.2 Caso de uso – Registro / Notificaciones (web)

### Nombre: Registro e inicio de sesión

### Actor principal:

Usuario

### Descripción:

El usuario puede registrarse o iniciar sesión para recibir notificaciones sobre novedades del juego.

### Flujo principal:

1. El usuario accede a la opción de registro/inicio de sesión.
2. Introduce sus datos (usuario, correo, contraseña).
3. El sistema valida los datos.
4. El usuario accede a su cuenta.
5. Recibe notificaciones sobre actualizaciones del juego.

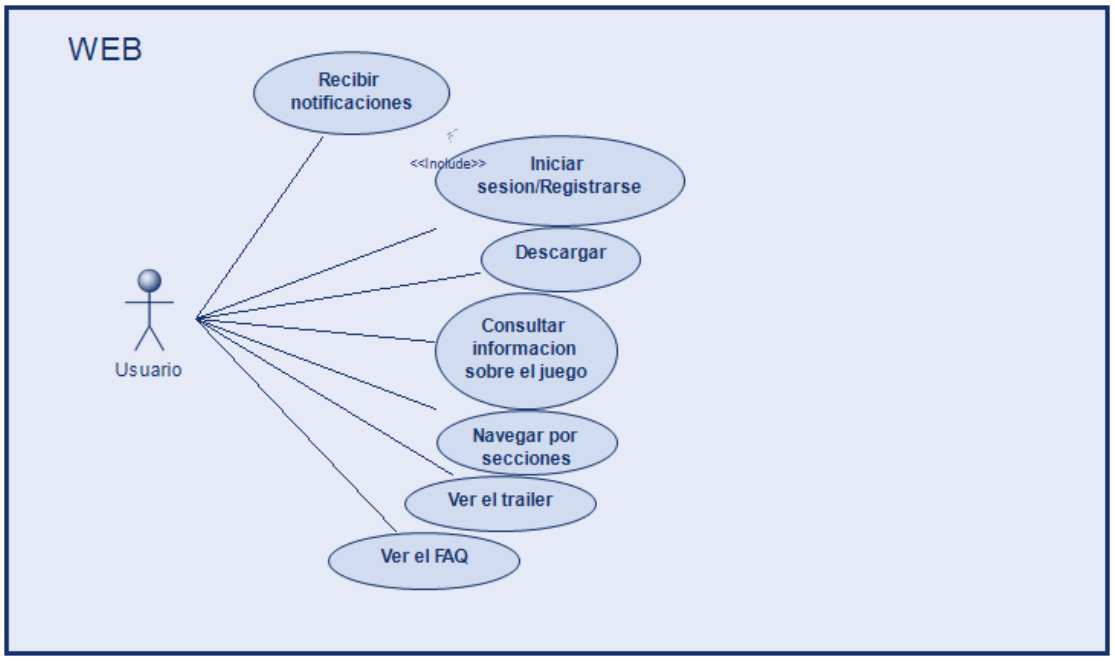
### Flujos alternativos:

- 3.ª Datos incorrectos → se muestra error.
- 2.ª Usuario ya registrado → inicia sesión directamente.

**Resultado:**

El usuario queda registrado o autenticado correctamente.

Muestra gráfica del caso de uso dentro de la parte de la página web gracias a modelio.




## 3.3 Caso de uso – Videojuego

### Nombre: Jugar una partida

### Actor principal:

Jugador

### Descripción:

El jugador inicia el juego y controla un personaje para explorar, combatir y progresar en el mundo del juego.

### Flujo principal:

1. El jugador inicia el juego.
2. Aparece el menú principal.
3. Selecciona “Nueva partida”.
4. Controla al personaje (movimiento y salto).
5. Explora el mapa.
6. Encuentra enemigos.
7. Combate con los enemigos.
8. Avanza por diferentes zonas.
9. Mejora habilidades o consigue objetos.

**Flujos alternativos:**

- 6.ª El jugador evita enemigos.
- 7.ª El jugador pierde el combate.
- 9.ª El jugador desbloquea nuevas habilidades.

**Flujo de error:**

- El jugador muere → pierde progreso parcial y reinicia desde un punto anterior.

**Resultado:**

El jugador progresa en el juego o reinicia tras morir.

## 3.4 Caso de uso – Combate

**Nombre: Sistema de combate**

**Actor principal:**

Jugador

**Descripción:**

El jugador interactúa con enemigos mediante un sistema de combate basado en ataque y vida.

**Flujo principal:**

1. El jugador encuentra un enemigo.
2. El enemigo detecta al jugador.
3. El jugador ataca.
4. El enemigo recibe daño.
5. El enemigo contraataca.
6. Se actualiza la vida de ambos.
7. El combate termina cuando uno es derrotado.

**Flujos alternativos:**

- 3.ª El jugador esquiva el ataque.
- 5.ª El enemigo tiene patrones especiales.

**Resultado:**

El enemigo es derrotado o el jugador pierde.

## 3.5 Caso de uso – Progresión del jugador

**Nombre: Mejorar personaje**

**Actor principal:**

Jugador

**Descripción:**

El jugador mejora sus habilidades o consigue objetos que afectan a la jugabilidad.

**Flujo principal:**

1. El jugador consigue un objeto o mejora.
2. Se añade al inventario o sistema de habilidades.
3. Se aplican mejoras (más vida, daño, habilidades nuevas).
4. El jugador continúa la partida con ventajas.

**Resultado:**

El personaje mejora y facilita el progreso en el juego.

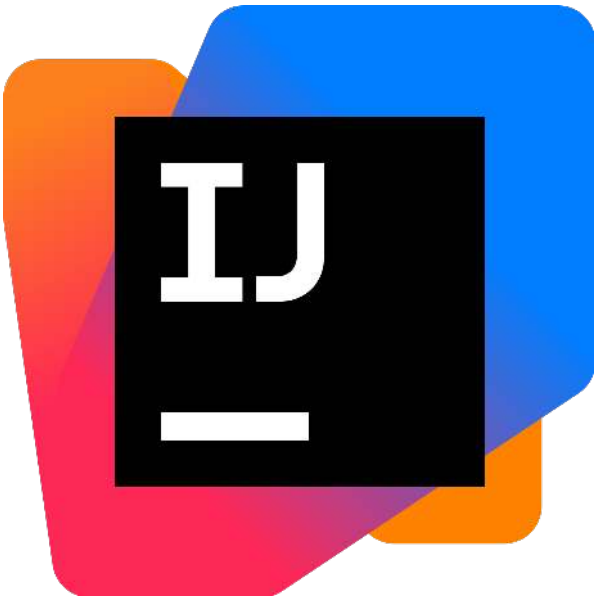
# 4. Tecnologías

## Visual Studio Code

Visual Studio Code es el editor de código fuente gratuito y de código abierto de Microsoft, y se ha consolidado como el entorno de referencia para el desarrollo de aplicaciones con React. Su integración nativa con Node.js, npm y JSX, junto con un ecosistema de extensiones orientadas a React, lo convierte en la herramienta ideal para construir aplicaciones web modernas basadas en componentes.

## IntelliJ

IntelliJ IDEA es el entorno de desarrollo integrado (IDE) de JetBrains, considerado el IDE de referencia para el desarrollo en Java y el ecosistema Spring. A diferencia de un simple editor de código, IntelliJ IDEA ofrece análisis de código profundo, refactorización avanzada, integración completa con Maven/Gradle y soporte nativo de Spring Boot, lo que lo convierte en la herramienta más productiva para desarrollar el backend del proyecto.



# Bibliografía

## Assets usados en el juego:

### Pixel Art Icon Pack - RPG

https://assetstore.unity.com/packages/2d/gui/icons/pixel-art-icon-pack-rpg-158343

Fantasy Wooden GUI : Free

https://assetstore.unity.com/packages/2d/gui/fantasy-wooden-gui-free-103811

Enemy Galore 1

https://assetstore.unity.com/packages/2d/characters/enemy-galore-1-pixel-art-208921

## Referencias que han ayudado

Canales de youtube

https://www.youtube.com/@RocketJam

https://www.youtube.com/watch?v=3SdMFPdSi7M

## Frameworks para la página web

Backend:
https://spring.io/projects/spring-boot

Frontend:
https://es.react.dev/learn

## Frameworks para el juego

Unity:
https://unity.com