Krahov`s Dragon

Proyecto Integrado DAM

Rubén Sierra Carcía

Índice

1. ANÁLISIS DEL PROBLEMA

1.1. Introducción

"Krakov's Dragon" es un videojuego de plataformas y aventura desarrollado en Godot 4, que narra la historia de un caballero cuyo pueblo ha sido destruido por un dragón. El jugador deberá enfrentarse a criaturas, superar obstáculos y obtener información de los aldeanos a cambio de oro, en un intento de descubrir quién o qué está detrás del caos desatado.

1.2. OBJETIVOS DEL PROYECTO

El proyecto tiene como principal objetivo el desarrollo de un videojuego de plataformas con elementos narrativos y de aventura, que ofrezca una experiencia de juego atractiva y desafiante para el usuario. A nivel técnico, se pretende implementar una aplicación completa que:

- Conecte con una base de datos externa mediante una API propia para almacenar el progreso del jugador.
- Permita guardar y recuperar partidas desde cualquier punto del juego.
- Integre un sistema de dificultad progresiva que aumente el reto conforme el jugador avanza en la historia.
- Añada componentes narrativos a través de personajes no jugables (NPCs) que aporten información y contenido argumental.
- Superar los contenidos estándar vistos en clase, aplicando nuevas tecnologías y enfoques, como el uso de Godot 4, GDScript, con el objetivo de explorar entornos de desarrollo profesional no abordados en profundidad durante el ciclo.

Además, se pretende que el jugador disfrute de una experiencia que, aunque aún en fase beta, plantee pequeños desafíos lógicos y de exploración, con una historia que va más allá de una simple mecánica de plataformas.

1.3. FUNCIONES Y RENDIMIENTOS DESEADOS

Aquí vamos a desglosar **qué funcionalidades debe tener el juego** y **qué se espera de su comportamiento**. Te presento una estructura base basada en lo que ya me contaste. Revísala y dime si quieres añadir o quitar algo:

El sistema debe ser capaz de ofrecer las siguientes funcionalidades clave:

- **Progreso del jugador:**
  Permitir al jugador guardar y recuperar el estado de su partida mediante una conexión a una base de datos remota.
- **Sistema de niveles:**
  Incluir varios niveles de dificultad creciente, diseñados para poner a prueba la habilidad del jugador conforme avanza.
- **Interacción con NPCs:**
  Integrar personajes no jugables con los que se puede interactuar para recibir información útil a cambio de oro u otros recursos.
- **Combate y enemigos:**
  Incorporar mecánicas de combate básicas contra criaturas, con IA simple, patrones de ataque y comportamiento variable.
- **Narrativa lineal:**
  Permitir que la historia se desarrolle en función del descubrimiento de pistas o diálogos, introduciendo un giro narrativo que aporta profundidad al argumento.
- **Sistema de economía básica:**
  El jugador podrá recolectar oro y gastarlo para obtener ventajas, como información o desbloqueo de ciertas rutas.
- **Controles y usabilidad:**
  El sistema debe contar con una interfaz accesible, fluida, y controles intuitivos, compatibles con teclado y mando.

1.4. PLANTEAMIENTO Y EVALUACIÓN DE DIVERSAS SOLUCIONES

Antes de implementar la solución definitiva, se valoraron varias alternativas en cuanto a motores, lenguajes, herramientas de backend y estrategias de guardado de datos. A continuación, se exponen las principales opciones consideradas:

Motor de desarrollo del juego

- **Unity:** Motor ampliamente conocido, potente y multiplataforma, pero con una curva de aprendizaje más elevada y reciente polémica por cambios en su modelo de licencias.
- **Unreal Engine:** Enfocado a gráficos de alta calidad, pero con requisitos de hardware más exigentes y sobrepotente para un proyecto 2D.
- ✅ **Godot 4:** Finalmente seleccionado por ser gratuito, de código abierto, ligero, multiplataforma, y con una comunidad creciente. Ofrece una excelente integración con GDScript, lo cual permite prototipar rápidamente.

Lenguaje de programación

- **C# o C++:** Lenguajes potentes pero con mayor complejidad para una producción individual en un marco limitado de tiempo.
- ✅ **GDScript:** Optado por su sencillez, integración nativa en Godot, sintaxis limpia, y buena documentación oficial.

Gestión de base de datos

- **SQLite:** Base de datos embebida útil para pruebas locales, pero no ideal para persistencia remota(Aunque mi objetivo inicial fue hacerla en local, el reto de poder conectar la Api con godot me superó, y optñe por esa opción).
- ✅ **MySQL + API en PHP:** Se optó por desarrollar una API RESTful en PHP alojada en un contenedor Docker, conectada a una base de datos MySQL. Esto permite guardar partidas de forma remota y replicar un entorno real de backend.

Estrategia de despliegue

- **Aplicación local con archivos planos:** Rechazada por no ser escalable ni realista para pruebas profesionales.
- ✅ **Uso de Docker:** Permite levantar un entorno LAMP (Linux, Apache, MySQL, PHP) fácilmente y mantener la consistencia entre entornos de desarrollo y producción.

1.5. JUSTIFICACIÓN DE LA SOLUCIÓN ELEGIDA

Tras valorar diversas opciones, se optó por una solución que combina el uso de **Godot 4 como motor principal**, **GDScript como lenguaje de programación**, y una **API externa desarrollada en PHP** conectada a una **base de datos MySQL** mediante **Docker**.

Esta elección se justifica por los siguientes motivos:

- **Godot 4** proporciona una plataforma ligera, eficiente y especialmente optimizada para juegos 2D, con una integración total con GDScript, lo que agiliza la curva de aprendizaje y la velocidad de desarrollo. Al tratarse de software libre y de código abierto, se alinea perfectamente con los principios educativos del ciclo formativo.
- **GDScript**, al estar diseñado específicamente para Godot, permite un desarrollo ágil y enfocado en lógica de juego, sin necesidad de configuración externa compleja.
- **El uso de una API externa** desarrollada en **PHP**, permite conectar el juego con una **base de datos remota**, simulando un escenario real de desarrollo profesional donde los datos del usuario (como el progreso o estadísticas) deben guardarse y consultarse de forma segura y estructurada.
- **Docker** ha sido empleado para levantar un entorno de servidor reproducible y portátil, facilitando el despliegue de la API sin necesidad de instalaciones manuales, lo cual incrementa la calidad y profesionalidad del proyecto.

1.6. MODELADO DE LA SOLUCIÓN

1.6.1. Recursos humanos

El desarrollo del proyecto ha sido llevado a cabo de forma individual, siendo el alumno responsable de todas las fases del mismo: análisis, diseño, implementación, pruebas, documentación y presentación. El tutor del módulo ha supervisado y orientado el proceso en las sesiones establecidas, realizando seguimiento del avance del proyecto.

1.6.2. Recursos hardware

Para el desarrollo del proyecto se ha utilizado el siguiente equipamiento:

- **Ordenador personal** con las siguientes características:
  - Procesador: Intel Core i7 13620 H
  - Memoria RAM: 16 GB
  - Disco duro SSD 500gb
  - Sistema operativo: Windows 11
- **Conexión a internet** para pruebas de API y acceso a documentación

- **Pantalla secundaria** para facilitar el diseño y depuración

1.6.3. Recursos software

Recurso	Descripción
Godot 4	Motor de desarrollo de videojuegos open source utilizado para construir todo el juego.
GDScript	Lenguaje de scripting propio de Godot, similar a Python, empleado para la lógica del juego.
PHP	Lenguaje backend usado para implementar la API REST que conecta el juego con la base de datos.
MySQL	Motor de base de datos relacional utilizado para almacenar el progreso y la información del jugador.
Docker + LAMP	Contenedor que integra Apache, MySQL y PHP para facilitar el despliegue de la API de forma local.
Git + GitHub	Herramientas empleadas para el control de versiones y alojamiento del código fuente del proyecto.

1.7. PLANIFICACIÓN TEMPORAL (No es del todo exacto)

Semana	Fecha aproximada	Tareas realizadas
Semana 1	17–23 marzo	Primer contacto con Godot 4 y adaptación al motor y su entorno
Semana 2	24–30 marzo	Familiarización con GDScript y pruebas básicas en Godot
Semana 3	31 marzo–6 abril	Diseño de niveles y mecánicas generales (fase de ideas y estructura)
Semana 4	7–13 abril	Desarrollo conceptual de niveles, definición de enemigos, obstáculos y narrativa

Semana	Fecha aproximada	Tareas realizadas
Semana 5	14–20 abril	Comienzo del desarrollo del tutorial: control del jugador, movimiento, físicas
Semana 6	21–27 abril	Continuación del tutorial: interacción con entorno, detección de colisiones
Semana 7	28 abril–4 mayo	Finalización del tutorial e integración de elementos jugables básicos
Semana 8	5–11 mayo	Creación del sistema de NPCs, lógica de interacción y sistema de vida del jugador
Semana 9	12–18 mayo	Ampliación de NPCs, diálogos y efectos; refinamiento del sistema de vida
Semana 10	19–25 mayo	Desarrollo de los tres niveles principales y diseño progresivo de dificultad
Semana 11	26 mayo–1 junio	Conexión del juego con la API en PHP; integración con base de datos vía Docker
Semana 12	2–8 junio	Pruebas finales, documentación del proyecto y entrega definitiva

2. DISEÑO E IMPLEMENTACIÓN DEL PROYECTO

El diseño del proyecto **Krakov's Dragon** se ha basado en una arquitectura cliente-servidor, en la cual el videojuego desarrollado en **Godot 4** actúa como cliente, y una **API REST desarrollada en PHP** funciona como servidor backend. Esta API se encarga de gestionar el guardado y recuperación del progreso del jugador, almacenando los datos en una **base de datos MySQL**.

El cliente, es decir, el propio juego, ha sido desarrollado íntegramente en **GDScript**, el lenguaje nativo de Godot, lo cual ha permitido una integración rápida de todas las funcionalidades de movimiento, combate, diálogos, carga de niveles y comunicación con la API externa. El juego está diseñado como una aventura de plataformas con componentes narrativos, donde el jugador avanza enfrentando enemigos, interactuando con NPCs y tomando decisiones clave.

La API ha sido desplegada en un entorno **Docker LAMP (Linux, Apache, MySQL, PHP)**, facilitando la portabilidad, el control del entorno y la consistencia durante el desarrollo. La estructura de datos de la base de datos incluye tablas como usuarios, progreso, niveles_superados.

El juego realiza llamadas HTTP desde GDScript mediante HTTPRequest para guardar o cargar los datos del jugador, intercambiando información en formato JSON con la API.

En cuanto a la interfaz, se ha priorizado la simplicidad y la jugabilidad, utilizando escenas de Godot organizadas jerárquicamente: menús, niveles, pantalla de tutorial, y escenas de diálogo. Cada nivel se ha diseñado con una dificultad creciente, añadiendo nuevos enemigos, trampas y decisiones que el jugador debe tomar, mientras que los NPCs aportan contenido narrativo esencial para avanzar en la historia.

El resultado es un videojuego funcional, con base de datos conectada, un flujo de juego progresivo.

3. FASE DE PRUEBAS

Durante el desarrollo de **Krakov's Dragon**, se llevaron a cabo diferentes tipos de pruebas enfocadas en garantizar el correcto funcionamiento del juego, la conexión con la base de datos y una experiencia de usuario fluida. Estas pruebas se realizaron principalmente de forma manual, aunque se utilizaron herramientas internas del motor Godot para el rastreo de errores.

Pruebas realizadas:

● **Pruebas de funcionalidad:**

- Movimiento del jugador, colisiones y detección de plataformas
- Sistema de daño y vidas
- Guardado y carga del progreso mediante la API (partidas persistentes)
- Interacción con NPCs (diálogos, respuestas, intercambio de oro)
- Transiciones entre niveles y control de la progresión
- Reinicio del juego desde el último punto guardado

● **Pruebas de integración:**

- Comunicación entre Godot y la API externa en PHP usando peticiones HTTP
- Envío y almacenamiento de datos en MySQL (oro, nivel, etc.)
- Recuperación de datos desde la base de datos al iniciar el juego
- Validación de respuestas JSON y manejo de errores

● **Pruebas de rendimiento:**

- Tiempos de carga entre escenas y llamadas a la API
- Respuesta del sistema ante múltiples llamadas consecutivas a la API

● **Pruebas de jugabilidad y usabilidad:**

- Evaluación del tutorial para asegurar que el jugador comprende la mecánica básica
- Testeo progresivo de la dificultad de los niveles
- Revisión del posicionamiento de los NPCs y puntos de interés
- Feedback de jugadores de prueba para mejorar la experiencia

4. Documentación de la aplicación

4.1. Introducción a la Aplicación

Krakov's Dragon es un videojuego de plataformas para un solo jugador, desarrollado en Godot 4. El jugador controla a un caballero que, tras ver su pueblo devastado por un dragón, se embarca en una aventura para descubrir la verdad y salvar a los suyos. A lo largo de su camino deberá enfrentarse a enemigos, interactuar con aldeanos y superar obstáculos mientras avanza por niveles de dificultad creciente.

El juego combina mecánicas clásicas de plataformas con una narrativa dinámica, donde los personajes no jugables (NPCs) ofrecen pistas, giros argumentales y progresión a través del intercambio de información. Aunque la partida se desarrolla en modo offline, el progreso se guarda mediante una conexión con una base de datos remota, lo que añade una capa de persistencia al sistema de juego.

4.2. Manual de Instalación

Cliente (Juego en Godot)

1. Clonar el repositorio del proyecto desde GitHub: https://github.com/Rebellant/TFG_DAM
2. Instalar **Godot 4.3** desde su sitio oficial: https://godotengine.org/download
3. Abrir el proyecto desde el editor de Godot.
4. Ejecutar directamente desde el editor (F5) o exportar a .exe si se desea una versión independiente.
5. También se puede usar el ejecutable (.exe) incluido en el repositorio para jugar directamente, sin necesidad del editor.
6. No es necesario realizar configuraciones adicionales. El juego está listo para jugar tras su apertura o instalación.

Backend (API PHP + Docker LAMP)

1. Descargar e instalar **Docker Desktop** desde:
   https://www.docker.com/products/docker-desktop
2. Crear una carpeta directamente en C:/ llamada Lamp y, dentro de ella, otra carpeta llamada app. La ruta completa quedará como:
   C:\Lamp\app
3. Descargar el archivo .zip que contiene la **API en PHP** y extraer su contenido dentro de la carpeta C:\Lamp\app.
4. Abrir la terminal de Windows (CMD o PowerShell) y ejecutar el siguiente comando para lanzar el contenedor Docker con el entorno LAMP:

docker run --name LAMP -p "80:80" -p "3306:3306" -v C:\Lamp\app:/app mattrayner/lamp:latest-1804

Este comando crea y ejecuta un contenedor con Apache, MySQL y PHP, montando la carpeta app como volumen para que la API esté accesible desde el navegador y desde el juego.

5. Asegurarse de que Docker esté en ejecución y que los puertos **80** (HTTP) y **3306** (MySQL) no estén siendo usados por otros servicios.
6. Una vez el contenedor esté levantado, la API estará disponible en:
   http://localhost/
   (o desde otra máquina, usando la IP del host en lugar de localhost).

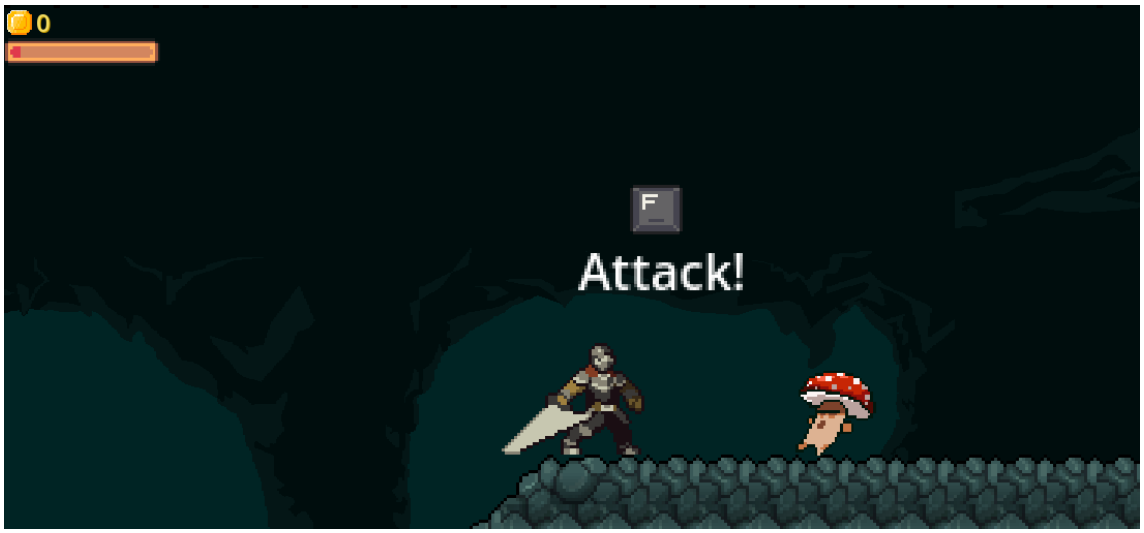
4.3. Manual de Usuario

Primera pantalla, podras jugar y crear una nueva partida o cintinuarla si ya tenias una.

Tutoriales que guian al jugador

Movimiento:

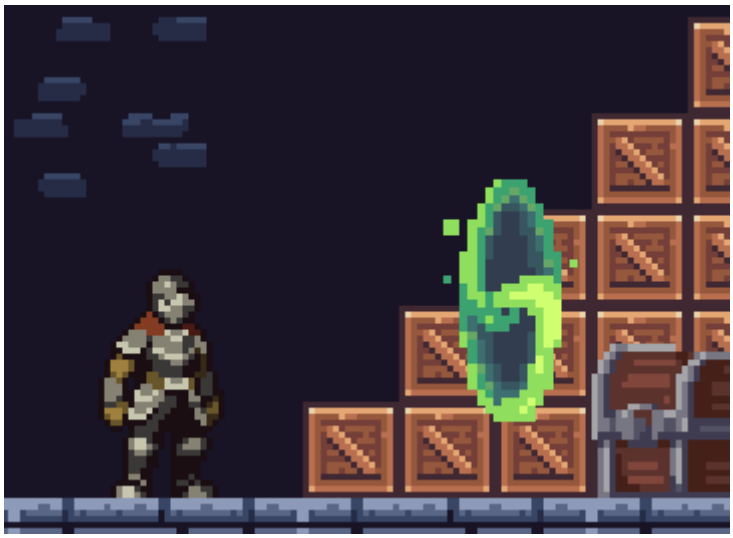
Sistema de combate guiado:



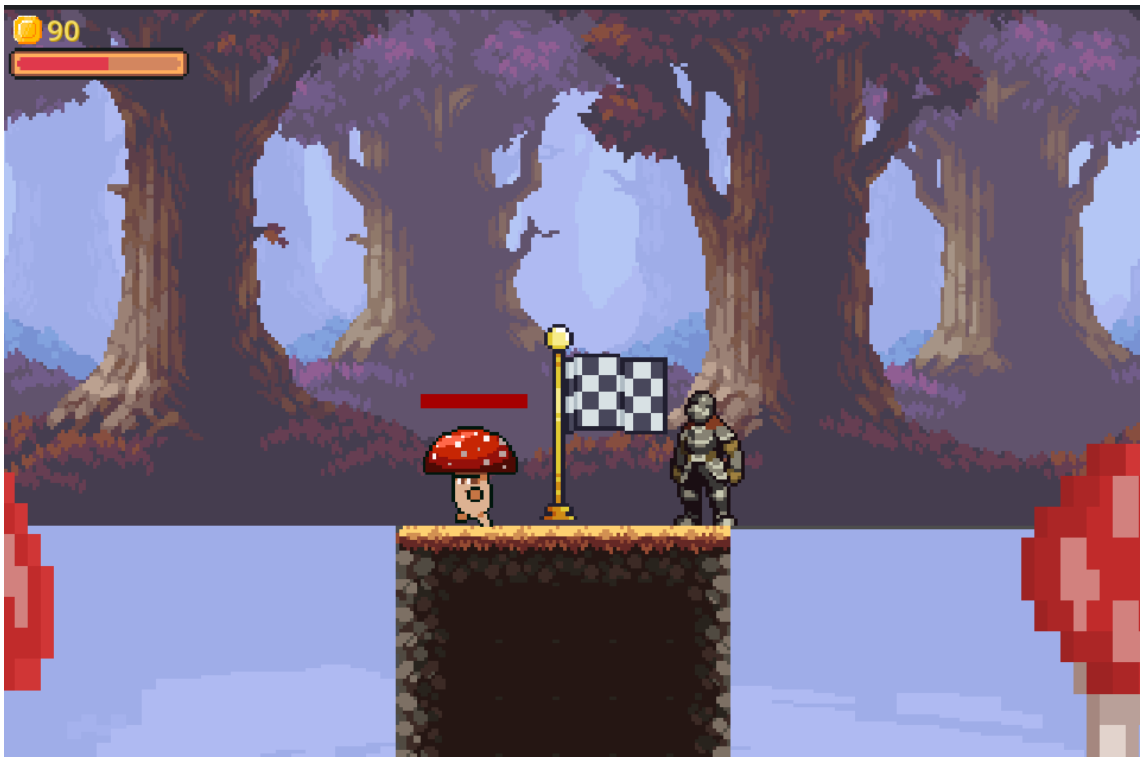
Interacción de diálogos intuitivos

Sistema de recolección de ítems:

Paso de niveles eficiente a través de portales



Guardado de progreso por checkpints:



Tradeo simple con los NPC`S

Menu de guardar y salir funcional con conexión a la API

Trampas para el jugador (pinchos)

Boss final

4.4. Manual de Administración

Actualmente no hay panel administrativo tradicional. La administración del sistema se realiza a través de herramientas de base de datos como PHPmyAdmin o Postman para gestionar endpoints y probar API.

5. Conclusiones Finales

5.1. Grado de Cumplimiento de los Objetivos

El proyecto ha cumplido la mayoría de los objetivos establecidos:

Sistema de combate

NPC con diálogos funcionales

Sistema de vida y recolección de esta

Sistema de comercio simple

Desenlace de la historia

Sistema de guardado y cargado de partida funcional

5.2. Propuesta de Modificaciones o Ampliaciones

Mejorar la calidad de guardado, pueden haber errores

Implementar cinematica final al matar al Boss final

Mejorar sistema de combate

6 Bibliografía

- **Godot Engine Documentation. Documentación oficial de Godot 4.3, utilizada como fuente principal para el desarrollo del proyecto con GDScript, incluyendo lógica de juego, diseño de escenas e interacción con APIs externas. Disponible en: https://docs.godotengine.org/en/stable/**
- **PHP Manual. Documentación oficial del lenguaje PHP, consultada para el desarrollo de la API del backend, manejo de peticiones HTTP y comunicación con bases de datos. Disponible en: https://www.php.net/manual/en/**
- **SQL Documentation (W3Schools). Referencia práctica para la construcción de consultas, definición de tablas y manipulación de datos en bases de datos relacionales. Disponible en: https://www.w3schools.com/sql/**
- **Docker Documentation. Guía oficial de Docker, utilizada para contenerizar los servicios del proyecto (como la API y la base de datos) y facilitar su despliegue en diferentes entornos. Disponible en: https://docs.docker.com/**
- **ChatGPT (OpenAI). Herramienta de asistencia basada en inteligencia artificial, utilizada como apoyo para la resolución rápida de problemas y dudas técnicas relacionadas con GDScript, SQL, PHP y configuración de contenedores Docker.**