Dead beans

Table of Contents

1. INTRODUCCIÓN

1.1 Descripción del Proyecto

Este proyecto consiste en hacer un videojuego estilo roguelike en el cual el jugador pueda desafiar a la mazmorra varia veces, consiguiendo hacerse más fuerte cada vez, con sistema de combate, generación de mazmorras, de inventario y otros.

1.2 Objetivos

Los objetivos eran crear un sistema jugable que incluyera guardado y carga de partida, combate, enemigos que suelten objetos, inventario, tienda para vender materiales y comprar equipamiento y una jugabilidad en loop.

1.3 Tecnologías Utilizadas

1.3.1 Herramientas de desarrollo

- **Unity**: Motor de desarrollo utilizado para la creación del videojuego, responsable de la lógica de juego, diseño de niveles y renderizado.
- **JetBrains Rider**: Entorno de desarrollo (IDE) usado para programar en C#, con integración avanzada para proyectos Unity.
- **Git**: Sistema de control de versiones distribuido utilizado para gestionar el código fuente del proyecto de forma eficiente y segura.

1.3.2 Backend y servicios

- **LAMP (Linux, Apache, MySQL, PHP)**: Stack de tecnologías web utilizado para implementar la API del proyecto:
  - **Linux**: Sistema base del contenedor Docker.
  - **Apache**: Servidor web responsable de exponer la API.
  - **MySQL**: Base de datos relacional donde se almacena la información de los enemigos, ítems y jugadores (para los rankings).
  - **PHP**: Lenguaje del lado del servidor con el que se ha desarrollado la API REST.
- **Docker**: Herramienta empleada para contenerizar tanto el entorno LAMP como el túnel de acceso externo mediante Cloudflare, facilitando el aislamiento del entorno y la portabilidad del backend.
- **Cloudflare Tunnel**: Servicio utilizado para exponer la API local de forma segura a través de internet, asignándole el dominio personalizado `apis.dfran49.com` sin necesidad de abrir puertos en el router. El túnel también se ejecuta en Docker y se comunica con el contenedor LAMP mediante una red interna.

1.3.3 Infraestructura y despliegue

- **Servidor local**: Todo el entorno backend (contenedores Docker) se ejecuta en el equipo personal del desarrollador, lo que permite un control total sobre el entorno y los datos sin depender de servicios de terceros.

2. ANÁLISIS Y DISEÑO

2.1 Análisis de Requisitos

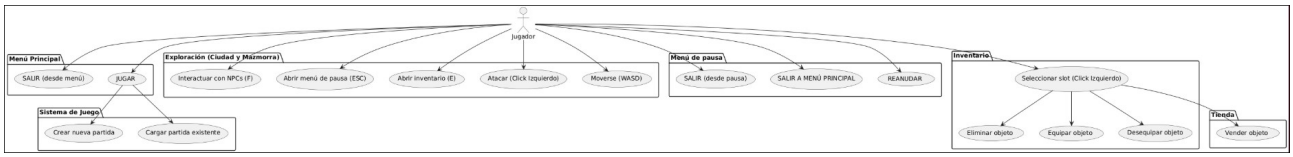
Requisitos funcionales:

- Que el jugador se pueda mover con las teclas wasd.
- Que el jugador pueda pausar y reanudar el juego con la tecla esc.
- Que el jugador pueda abrir y cerrar el inventario con la tecla e.
- Que el jugador pueda interactuar con los npc con la tecla f.

Requisitos no funcionales:

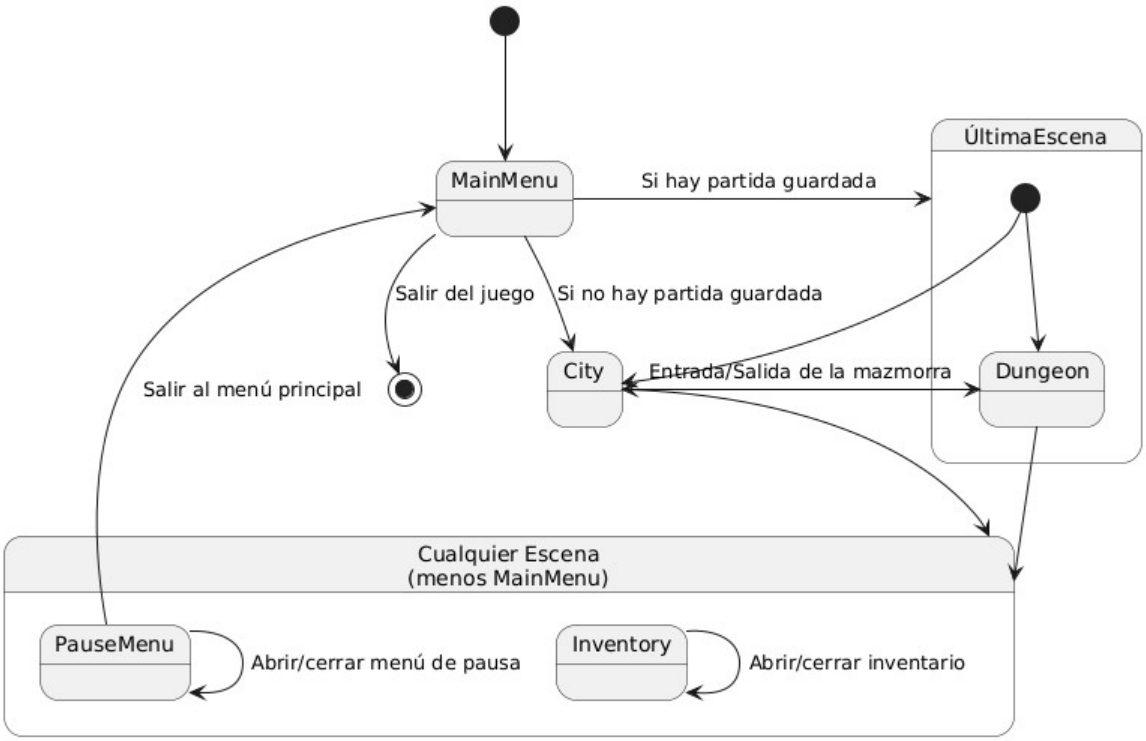
- El juego debe ejecutarse sin problemas de rendimiento.
- La interfaz debe ser intuitiva y aportar información relevante.
- Los objetos deben seguir un patrón modular que permita escalabilidad.

2.2 Casos de Uso



2.3 Diagrama de Flujo del Juego

Las escenas existentes y la navegación son:



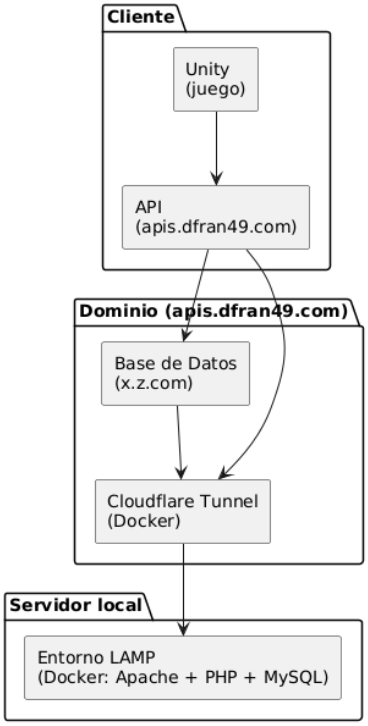
2.4 Arquitectura del Sistema

El sistema implementa una arquitectura cliente-servidor, en la que el videojuego desarrollado en Unity actúa como cliente y se comunica con una API externa para acceder a los datos persistentes. La arquitectura se compone de los siguientes elementos:

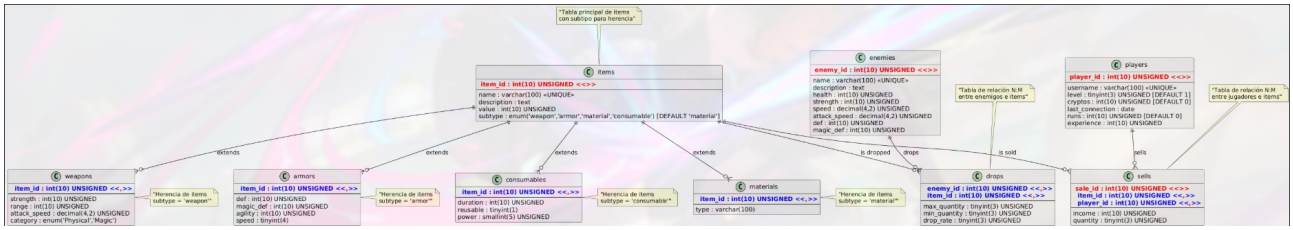
- **Cliente (Unity)**: Ejecuta el juego en el dispositivo del usuario. Desde Unity se realizan peticiones HTTP a la API para obtener y enviar información relevante, como datos de enemigos, ítems o puntuaciones de jugadores.
- **API REST (PHP)**: Interfaz web que expone varios endpoints para el acceso controlado a la base de datos. La API recibe solicitudes del cliente Unity, las procesa y devuelve respuestas en formato JSON.
- **Base de datos (MySQL)**: Contiene la información estructurada del juego, incluyendo enemigos, ítems y datos de jugadores para los leaderboards. Está alojada en el mismo entorno que la API.
- **Servidor local con Docker**: Todo el backend (Apache + PHP + MySQL) está desplegado mediante contenedores Docker en un ordenador personal del desarrollador. Se configura una red interna entre los contenedores para facilitar la comunicación entre servicios.
- **Cloudflare Tunnel**: Permite exponer de forma segura la API al exterior a través de internet, sin necesidad de abrir puertos directamente en el router. Se ha asignado un dominio personalizado (`apis.dfran49.com`) para acceder al servicio desde Unity.

Este diseño permite separar las responsabilidades entre cliente y servidor, garantizando un sistema más modular, mantenible y seguro.

A continuación, un diagrama de arquitectura:



2.5 Diseño de la Base de Datos



3. ACCESO A DATOS

3.1 Conexión con la API PHP

El juego hace peticiones a la API mediante UnityWebRequest, y guarda esos json en la carpeta de datos del juego.

Las peticiones son por http.

3.2 Gestión de Datos del Jugador

Los datos del jugador solo se gestionan de forma local. Al iniciar el juego por primera vez, se genera un json predeterminado. En ese momento, se crea un ScriptableObject del jugador, al cual se le aplican los datos del json. Cuando se modifican datos como la cantidad de cryptos, la vida actual, el inventario y la escena actual, se guardan al ScriptableObject. Y finalmente, al salir de la aplicación o cambiar de escena, se guardan los cambios del ScriptableObject al json de player, y a la siguiente vez que se lana el juego, se cargan esos datos en vez de los predeterminados

3.3 Sistema de Guardado/Carga

El sistema de guardado y carga es mayormente la gestión de datos del jugador y la conexión a la API.

3.4 Manejo de Estadísticas

El prefab del jugador tiene dos componentes, uno para las estadísticas base, y otro para la gestión de la vida. En esos componentes se modifican las estadísticas leídas del ScriptableObject del jugador.

3.5 Serialización de Datos

Como ya se ha mencionado, al cambiar de escena o salir del juego, se guardan los datos del ScriptableObject de player en json. Los ScriptableObject tienen integración con clases serializables que permiten convertirlas a json directamente. A parte del ScriptableObject del jugador, están los de los objetos y de los enemigos (Que se crean al leer los datos del json generado por las peticiones de la API).

4. BASE DE DATOS

4.1 Diseño del Modelo de Datos

players representa a cada usuario.

sells representa la lista de las ventas de los objetos por cada jugador.

items representa a cada objeto (Con su subtipo).

enemies representa a cada enemigo.

drops representa los items que puede soltar un enemigo al morir.

4.2 Relaciones entre Entidades

La tabla drops es muchos a muchos porque un enemigo puede soltar más de un item, y un item puede ser soltado por más de un enemigo.

La tabla sells es muchos a muchos porque un item puede ser vendido por muchos jugadores, y muchos jugadores pueden vender un item.

Las tablas hijas de items existen para diferenciar los distintos tipos de items existentes.

4.3 Scripts SQL de Creación

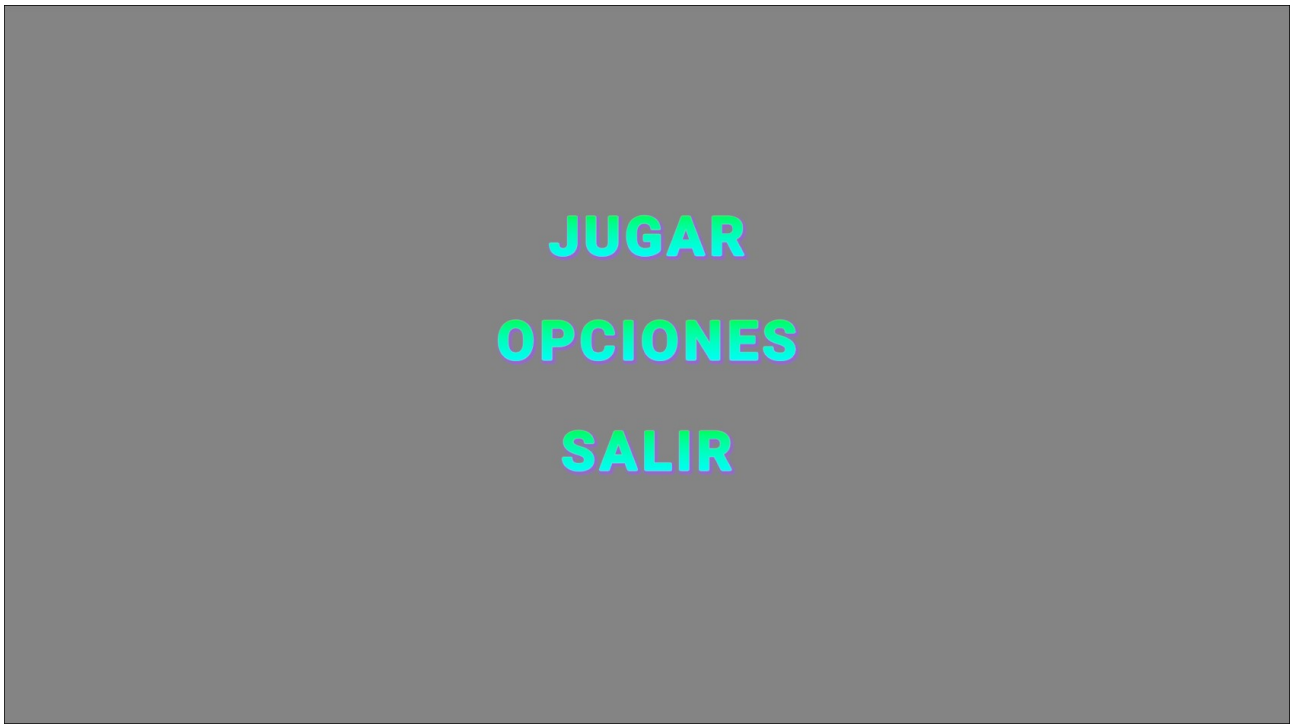
Se encuentra en la misma carpeta, se llama “deadbeans.sql”.

5. INTERFAZ GRÁFICA (UI/UX)

5.1 Diseño de Interfaces

Las interfaces son las siguientes, adjunto capturas:

5.2 Menú Principal



5.3 HUD del Juego

5.4 Sistema de Inventario

5.5 Menú de Pausa

5.6 Interfaz de Tienda

6. DESARROLLO DEL JUEGO EN UNITY

6.1 Estructura del Proyecto

La estructura es un poco un caos, hay scripts dentro de Assets/Resources/Characters, y en carpetas internas, como la del jugador (Dentro de esta hay algunos componentes globales como las estadísticas y la vida). Tambien hay scripts en Assets/Resources/Scripts, estos si están mejor ordenados por carpetas.

6.2 Sistema de Escenas

6.2.1 Menú de Inicio

La escena de inicio donde se puede salir de la app o comenzar a jugar.

6.2.2 Base/Ciudad

La escena donde el jugador puede curarse, vender objetos o adentrarse en la mazmorra.

6.2.3 Mazmorra

La escena donde el jugador derrota enemigos para conseguir objetos, y tras derrotar al jefe vuelve a la ciudad.

6.3 Generación Procedural

6.3.1 Algoritmo de Generación de Salas

El algoritmo de salas se basa en un array de 6x5 (5x5 para las salas normales). Se parte de la sala de spawn en (0,0) y se empieza a generar desde la sala de arriba de esta, (0,1). En cada sala se ve si en las salidas disponibles (No bloqueadas por dar hacia fuera del array) hay salas, si no, se genera una sala, si hay más de unas salida como esta, se pueden generar más salas nuevas, pero la probabilidad disminuye. Si al otro lado hay una sala, hay un 50% de probabilidades de abrir la puerta para conectarlas. Finalmente, se genera la sala del boss encima de una sala que tenga hueco, y se abre la puerta de arriba de esta.

Tras eso, se asignan las puertas, se les quita el sprite de bloqueo y se les asigna un tp a la siguiente sala en base a la dirección, cambiando la posición del jugador.

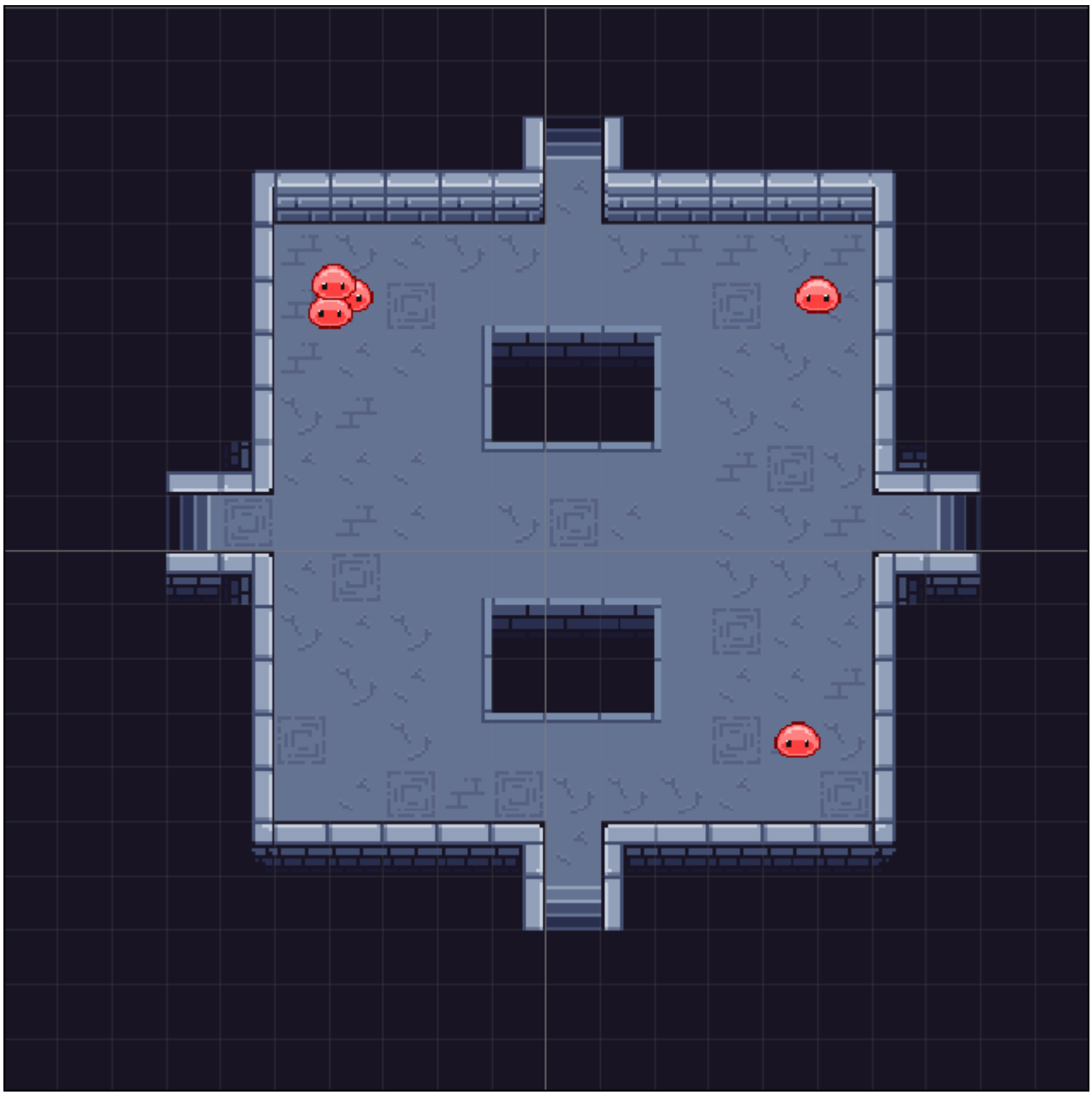
Las salas que se generan son en base a ciertos prefabs ya generados que están pasados como parámetro mediante el editor de unity.

Aquí dejo un ejemplo de una mazmorra generada:

6.3.2 Sistema de Spawning de Enemigos

En cada sala, hay 4 puntos definidos para generar los enemigos. Hay un mínimo y un máximo de enemigos por sala. Se calcula cuantos se van a generar (entre mínimo y máximo) y tras eso, se recorren los puntos de forma aleatoria (Para evitar que siempre aparezcan en los mismos puntos) y se genera uno por cada punto elegido automáticamente.

Aquí dejo un ejemplo de una sala con enemigos:



6.4 Sistema de Combate

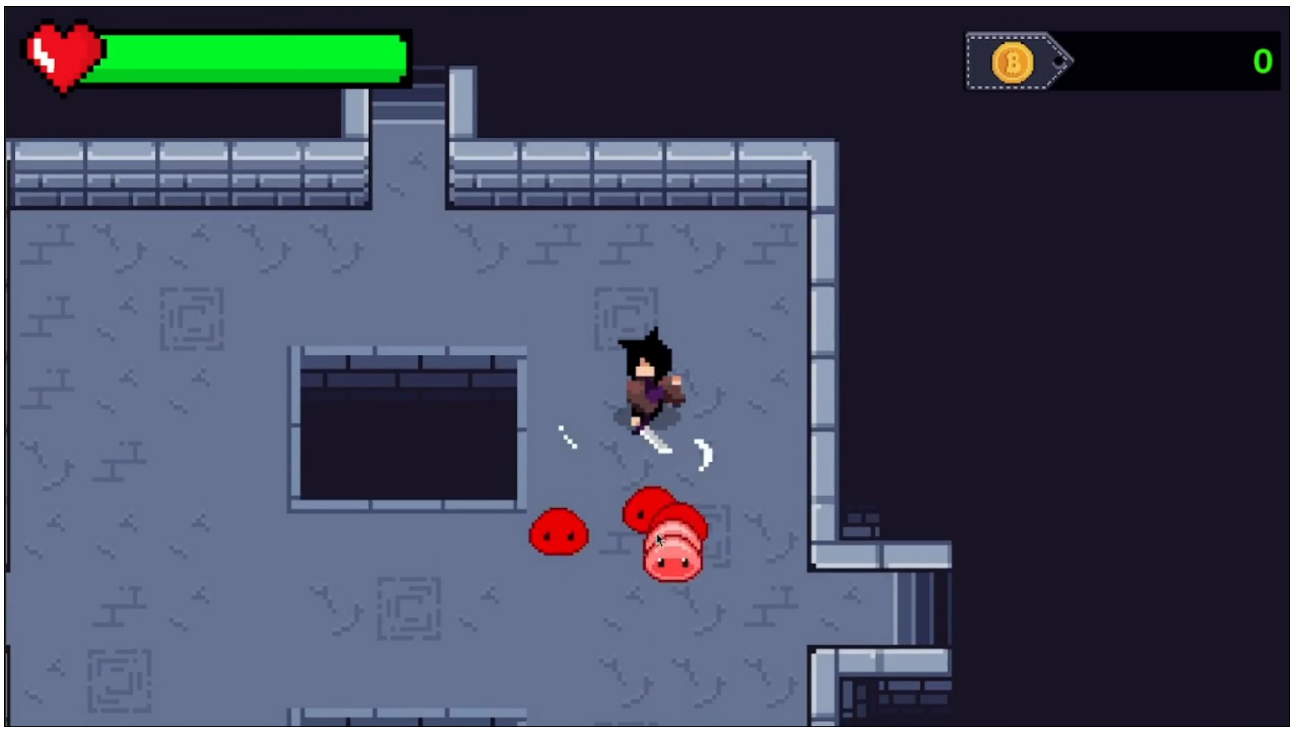
El sistema de combate es simple. El jugador ataca, se comprueba si el ataque golpea a los enemigos y les aplica daño.

En el caso de los enemigos, solo se mueven hacia el jugador, y si lo tocan, le hacen daño.

Cuando cualquier entidad recibe daño, se reproduce una animación y son empujados hacia atrás, además de parar el tiempo del juego unos pocos milisegundos.

Cuando el jugador recibe daño, no puede volver a recibir daño hasta pasado 1 segundo.

A continuación dejo capturas de los enemigos y del jugador recibiendo daño:



6.5 Sistema de Items y Drop Tables

Cuando un enemigo muere, se lee la lista de dropTables sacada de su ScriptableObject (Que ha cargado el json recibido de la API), se recorren todos los elementos, y por cada uno se hacen 2 random, uno para ver si se genera el item, y otro para ver cuantos se generan (Entre el mínimo y el máximo).

6.6 Mecánicas del Jugador

Las mecánicas del jugador se hacen mediante un sistema de inputs.

6.6.1 Movimiento

El movimiento es con las teclas wasd, y se aplica dándole una linearVelocity al rigidbody2d del jugador.

6.6.2 Ataque

El ataque se lanza con click izquierdo y según la dirección mirada, asigna un punto como el centro del ataque, y tras eso, lanza una corrutina que comprueba si hay entidades dentro de un círculo con origen el centro del ataque, y una vez se les ha aplicado daño en un ataque, las entidades no pueden volver a recibir daño del mismo ataque. La corrutina termina cuando termina la animación de ataque (Asignado en el animador de unity).

6.6.3 Interacción

Se lanza con la tecla f, se le asigna una función cuando se entra al rango de algún npc, y cuando se sale, se elimina.

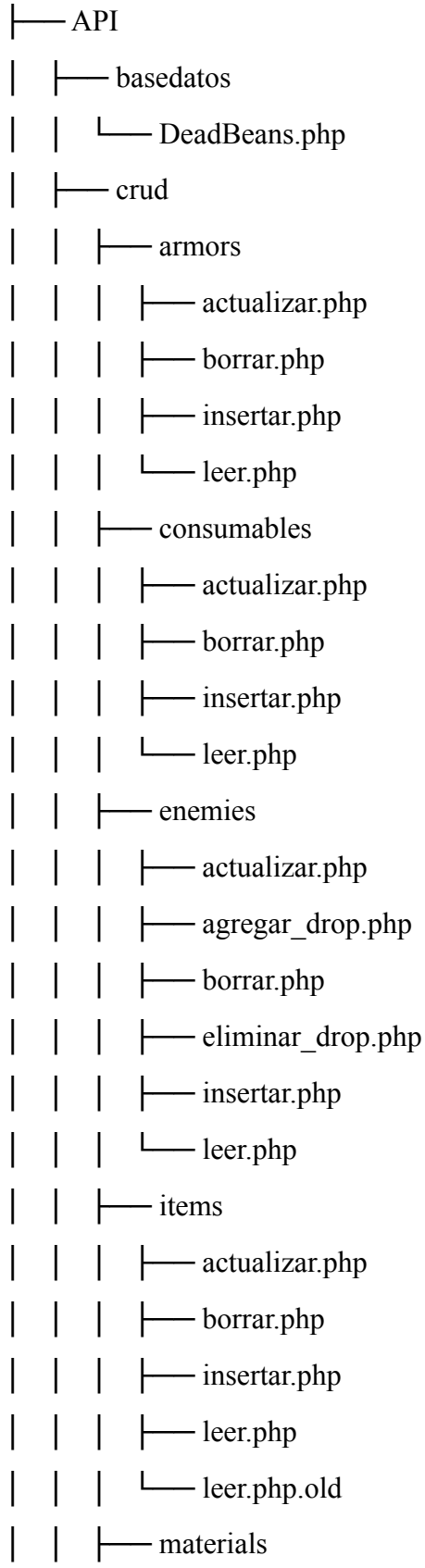
6.6.4 Recolección de Items

No hay interacción del jugador aquí, es simplemente un área alrededor del jugador, en la cual si entran los items, se recogen.

7. API PHP Y COMUNICACIÓN

7.1 Estructura de la API

La estructura de la api es la siguiente:



7.2 Endpoints Implementados

Base URL

https://apis.dfran49.com/API/crud/

Endpoints por Tabla

Armors (Armaduras)

Leer: https://apis.dfran49.com/API/crud/armors/leer.php

Insertar: https://apis.dfran49.com/API/crud/armors/insertar.php

Actualizar: https://apis.dfran49.com/API/crud/armors/actualizar.php

Borrar: https://apis.dfran49.com/API/crud/armors/borrar.php

Consumables (Consumibles)

Leer: https://apis.dfran49.com/API/crud/consumables/leer.php

Insertar: https://apis.dfran49.com/API/crud/consumables/insertar.php

Actualizar: https://apis.dfran49.com/API/crud/consumables/actualizar.php

Borrar: https://apis.dfran49.com/API/crud/consumables/borrar.php

Enemies (Enemigos)

Leer: https://apis.dfran49.com/API/crud/enemies/leer.php

Insertar: https://apis.dfran49.com/API/crud/enemies/insertar.php

Actualizar: https://apis.dfran49.com/API/crud/enemies/actualizar.php

Borrar: https://apis.dfran49.com/API/crud/enemies/borrar.php

Agregar Drop: https://apis.dfran49.com/API/crud/enemies/agregar_drop.php

Eliminar Drop: https://apis.dfran49.com/API/crud/enemies/eliminar_drop.php

Items (Objetos)

Leer: https://apis.dfran49.com/API/crud/items/leer.php

Insertar: https://apis.dfran49.com/API/crud/items/insertar.php

Actualizar: https://apis.dfran49.com/API/crud/items/actualizar.php

Borrar: https://apis.dfran49.com/API/crud/items/borrar.php

Materials (Materiales)

Leer: https://apis.dfran49.com/API/crud/materials/leer.php

Insertar: https://apis.dfran49.com/API/crud/materials/insertar.php

Actualizar: https://apis.dfran49.com/API/crud/materials/actualizar.php

Borrar: https://apis.dfran49.com/API/crud/materials/borrar.php

Players (Jugadores)

Leer: https://apis.dfran49.com/API/crud/players/leer.php

Insertar: https://apis.dfran49.com/API/crud/players/insertar.php

Actualizar: https://apis.dfran49.com/API/crud/players/actualizar.php

Borrar: https://apis.dfran49.com/API/crud/players/borrar.php

Sells (Ventas)

Leer: https://apis.dfran49.com/API/crud/sells/leer.php

Insertar: https://apis.dfran49.com/API/crud/sells/insertar.php

Actualizar: https://apis.dfran49.com/API/crud/sells/actualizar.php

Borrar: https://apis.dfran49.com/API/crud/sells/borrar.php

Weapons (Armas)

Leer: https://apis.dfran49.com/API/crud/weapons/leer.php

Insertar: https://apis.dfran49.com/API/crud/weapons/insertar.php

Actualizar: https://apis.dfran49.com/API/crud/weapons/actualizar.php

Borrar: https://apis.dfran49.com/API/crud/weapons/borrar.php

Resumen de Operaciones CRUD

Cada tabla soporta las siguientes operaciones básicas:

CREATE (Insertar): Crear nuevos registros

READ (Leer): Obtener registros existentes

UPDATE (Actualizar): Modificar registros existentes

DELETE (Borrar): Eliminar registros

Operaciones Especiales

La tabla Enemies incluye operaciones adicionales para gestionar drops:

Agregar Drop: Añadir objetos que pueden soltar los enemigos

Eliminar Drop: Remover objetos de la lista de drops de un enemigo

Total de Endpoints

30 endpoints CRUD básicos (4 operaciones × 7 tablas + 2 operaciones especiales)

8 tablas con funcionalidad completa CRUD

8. DESPLIEGUE

8.1 Configuración del Servidor

Mi equipo usa Arch Linux, y gestiono docker desde consola + portainer. El servidor como ya se ha dicho antes, es un LAMP alojado en un contenedor de docker.

8.2 Despliegue de la API PHP

La API está subida a la carpeta APP del LAMP mediante una carpeta bindeada a mi sistema.

8.3 Configuración de Base de Datos

La configuración usada es la predeterminada, con la contraseña de acceso que da la imagen de docker que he usado.

8.4 Configuración de Cloudflare

He configurado un subdominio para mi dominio, lanzando el contenedor de docker para el tunel, y asignando una red interna entre el contenedor del tunel y el de LAMP. Tras eso, en cloudflare le he indicado la ip y el puerto del LAMP en la red interna del docker.

9. CÓDIGO Y REPOSITORIO GITHUB

9.1 Estructura del Repositorio

El contenido del repositorio es el proyecto de unity, y la carpeta de documentación, ambos en la raíz.

9.2 Versionado y Commits

He estado trabajando en ramas fuera de main, intentando hacer una rama para cada funcionalidad, pero al final he acabado haciendolo casi todo en “CombatSystem”.

10. CONCLUSIONES Y TRABAJO FUTURO

10.1 Objetivos Cumplidos

Un juego con loop, con generación de mazmorra procedural, sistema de inventario, sistema de guardado, venta de objetos y lectura de datos de una api.

10.2 Dificultades Encontradas

A la hora de buildear el proyecto, me he dado cuenta de que el juego no funcionaba porque los ScriptableObjects no se importaban y tampoco se generaban fuera del editor de unity.

10.3 Posibles Mejoras

Cambiar el sistema de gestión de datos de ScroptableObjects a leer y modificar directamente los json.

10.4 Ampliaciones Futuras

Me habría encantado hacerle mejoras, convertir el juego en algo disfrutable, pero con los problemas que me ha dado la verdad es que no me ha dejado muchas ganas. Posiblemente en un tiempo lo retome, pero haciéndolo de 0, con la experiencia adquirida.