# Proyecto final de Paradigmas y Técnicas de Programación

## Javier Escobar Serrano y Marcos Garrido Ferrer

## Descripción general

Nuestro juego va a ser un roguelike de exploración de mazmorras con gráficos 2,5D (similares a los de Paper Mario). En el juego, el planeta ha sido invadido completamente por la vegetación. Se dice que el rápido crecimiento de las plantas se debe a una fuente de agua contaminada que se encuentra en las profundidades de un templo antiguo. El jugador toma el rol de un explorador desesperado, y se adentra en la mazmorra con el objetivo de encontrar dicha fuente, si es que existe.

La mazmorra consta de varios pisos, cada uno con distintas salas: trampas, enemigos, tesoros... Cada planta tiene una temática distinta, incrementándose la dificultad progresivamente a lo largo del descenso.

El jugador deberá explorar cada planta para encontrar equipamiento con el que enfrentarse a los enemigos. Cuanto mayor sea el riesgo que tome, mayor será su recompensa si consigue salir con vida de las salas. Cada piso cuenta con una sala de jefe, que al ser derrotado se abrirá el acceso al piso inferior. En el último piso, habrá un jefe final que tendrá que ser derrotado para completar el juego.

Al empezar la partida, el jugador contará con recursos limitados: equipamiento básico para sobrevivir temporalmente. La partida concluye cuando el jugador es derrotado o consigue vencer al jefe final. No se conservará el progreso entre partidas, pero será posible realizar un guardado para poder continuar la partida en otro momento.

El sistema de combate será el siguiente:

* Al pulsar el botón de ataque, aparecerá un marcador encima del enemigo más cercano.
* El personaje atacará automáticamente al enemigo marcado.
* Los enemigos pueden soltar equipamiento al morir, que el jugador podrá recoger y utilizar.
* Tenemos pensado que existan 4 tipos de armas: espada corta, espada larga, arco y vara mágica. Habrá subclases de cada una de ellas con efectos distintos.

El jugador contará con un inventario limitado de alrededor de 8 objetos. Si teniendo el inventario lleno recoge otro ítem, soltará el que tenga equipado en ese momento para reemplazarlo.

## Historias de usuario

Como jugador, me gustaría poder guardar la partida para continuarla más adelante si no tengo tiempo.

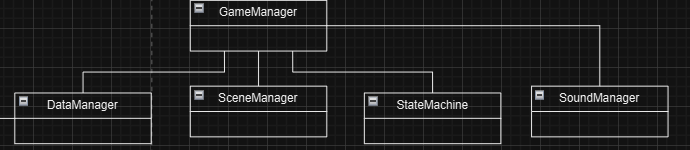
Como jugador, querría que el juego guardara mi puntuación de alguna manera (ya sea por el número de enemigos derrotados, piso alcanzado...) para poder competir con mis amigos.

Como usuario, me gustaría poder ajustar los controles para jugar de forma más cómoda.

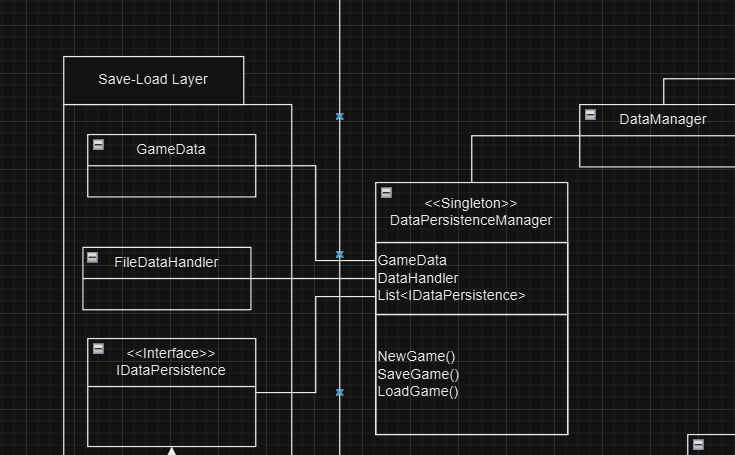
Como jugador, me gustaría que hubiera una selección de dificultad. De esta manera, cuando estoy aprendiendo el juego sería más benevolente; y tendría un reto mayor si decido dedicarme a fondo en el juego.

## Justificación del UML

1- Singletons

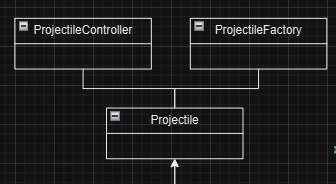


Muchas de las clases “manager” de la arquitectura pueden implementarse como clases Singleton. Aunque no accedamos a ellas desde muchas otras clases (god object / spaghetti code), conviene utilizar este patrón para asegurarnos de que se instancia uno y solo uno de estos managers.

2- Guardado de partida

Los elementos que serán guardados en memoria implementarán la interfaz IDataPersistence. De esta forma, el Singleton DataPersistenceManager realizará la lectura y escritura de los datos.

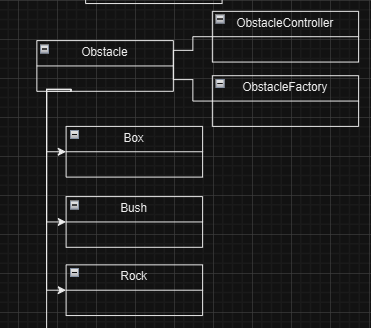
3- Factorías, puentes y decoradores



Para abstraer el código, tenemos pensado utilizar factorías de algunas clases. En el caso de los proyectiles, como queremos tenerlos de varios tipos (flechas, magia…), conviene utilizar este patrón. Además, estamos utilizando de forma indirecta el patrón Bridge, puesto que dividimos los proyectiles en abstracción e implementación. Estos mismos patrones los utilizamos para enemigos, objetos…

También utilizaremos el patrón Decorator a la hora de crear a los enemigos. Esta abstracción nos facilita crear enemigos similares pero con alguna característica distinta, sin tener que escribir una clase para cada combinación posible de cualidades.

4- Flyweight



Los obstáculos que vamos a crear tienen diferentes características (si son destructibles o no, si son sólidos o no…). No obstante, los obstáculos de un mismo tipo compartirán todos sus atributos menos su posición. Cada obstáculo guardará una referencia a sus características principales y su posición. De esta manera, evitamos guardar información redundante en cada una de las instancias.