ENTREGA FINAL DISEÑO DE SOFTWARE

Rodrigo Fernández
Antonio Salguero
Marco Adrián Vidaurre

1. Introducción

Nuestra entrega es el juego de combate. Hemos querido implementar todos los patrones. El juego es por consola, sin implementar una interfaz gráfica.

En esta memoria se explicará cómo se ha implementado cada patrón y que ha hecho cada integrante del grupo.

2. Código general

El proyecto ha sido dividido en tres partes. Cada integrante se ha encargado de hacer una parte aunque hemos tenido que hacer pequeños retoques de otras partes para juntarlo todo sin problema.

Es un juego de combate por turnos en el que te enfrentas a enemigos generados aleatoriamente de forma infinita. Cada vez que derrotas a uno, sumas un punto. Los enemigos tienen estadísticas diferentes y estilos de combate variados (agresivo, defensivo o aleatorio). Su comportamiento cambia en cada partida y pueden usar distintos ataques (aunque no les hemos asignado ninguno en el juego).

Como jugador, eliges entre varios tipos de ataques, cada uno con potencia distinta y con probabilidades de aplicar efectos negativos al enemigo, como paralizarlo o debilitarlo. También tienes una acción defensiva que te permite reducir el daño del próximo golpe. El combate sigue hasta que mueras, momento en el que se muestra tu puntuación final.

Rodrigo Fernández: Me he encargado del proyecto general. He creado el diagrama de clases para organizarlo todo bien. Al terminar el diagrama creé el proyecto vacío con las clases/interfaces vacías para establecer la organización de paquetes. Una vez terminado se lo he pasado a Antonio y Marco para que se pusieran con sus partes pero en la plantilla ya organizada. He hecho una versión inicial del paquete **MODEL**. Me he ido encargando de juntar los patrones y las distintas partes del proyecto,

también creando el singleton en el paquete **SERVICE,** para el manejo del daño.

Marco Adrián Vidaurre: Me he encargado del manejo del juego, es decir, cómo funciona el gameplay, las rondas y el spawneo de enemigos. Para esto me he encargado de hacer los paquetes de **CONTROLLER, STRATEGY** y **FACTORY.** También he tenido que modificar varias clases para adaptarlas al Controller.

Antonio Salguero: Me he encargado de las batallas del juego y del funcionamiento de los ataques del jugador y del enemigo. He hecho los paquetes **ACTION y STATE**. He modificado clases del paquete **MODEL** para mejorar su funcionamiento.

3. Patrón State

El patrón State ha sido implementado por Antonio. Lo usamos para que los personajes cambien su comportamiento según su estado durante el combate. Así, un personaje herido, paralizado o en recuperación actúa de forma distinta sin tener que meter condicionales por todo el código.

Para ello se han usado la interfaz *CharacterState* y las clases *InjuredState*, *NormalState*, *ParalyzedState* y *RecoveringState*.

4. Patrón Decorator

El patrón Decorator ha sido implementado por Antonio. Lo usamos para modificar los ataques de forma dinámica según las características del personaje o su estilo de combate. Así, a un ataque básico se le pueden añadir efectos como veneno, doble golpe o potencia extra.

Para ello se ha usado la interfaz *Action*, la clase abstracta *ActionDecorator* y las clases *StrongAttackDecorator*, *PoisonAttackDecorator*, *DoubleHitDecorator* y *BasicAttack*.

5. Patrón Abstract Factory

El patrón Abstract Factory lo ha hecho Marco Adrián. Lo usamos para generar enemigos distintos según el mundo en el que se encuentra el jugador, manteniendo el mismo tipo (guerrero, mago o mutante) pero con atributos y comportamiento adaptados al entorno.

Para ello se ha usado la interfaz *EnemyFactory* y las clases *JungleEnemyFactory*, *OceanEnemyFactory* y *DesertEnemyFactory*.

6. Patrón Singleton

El patrón Singleton lo ha hecho Rodrigo. Lo usamos para gestionar todos los cálculos de daño en una única clase que se usa en todo el juego, evitando duplicar lógicas. También se encuentra en el manejo del inicio del juego.
Para ello se ha creado la clase *DamageCalculator y GameHandler*.

7. Patrón Template Method

El patrón Template Method lo han hecho Rodrigo y Antonio. Lo usamos en la clase abstracta *Character* con el método performAction(). Las clases abstractas tienen métodos que juegos las hijas contienen

8. Patrón Facade

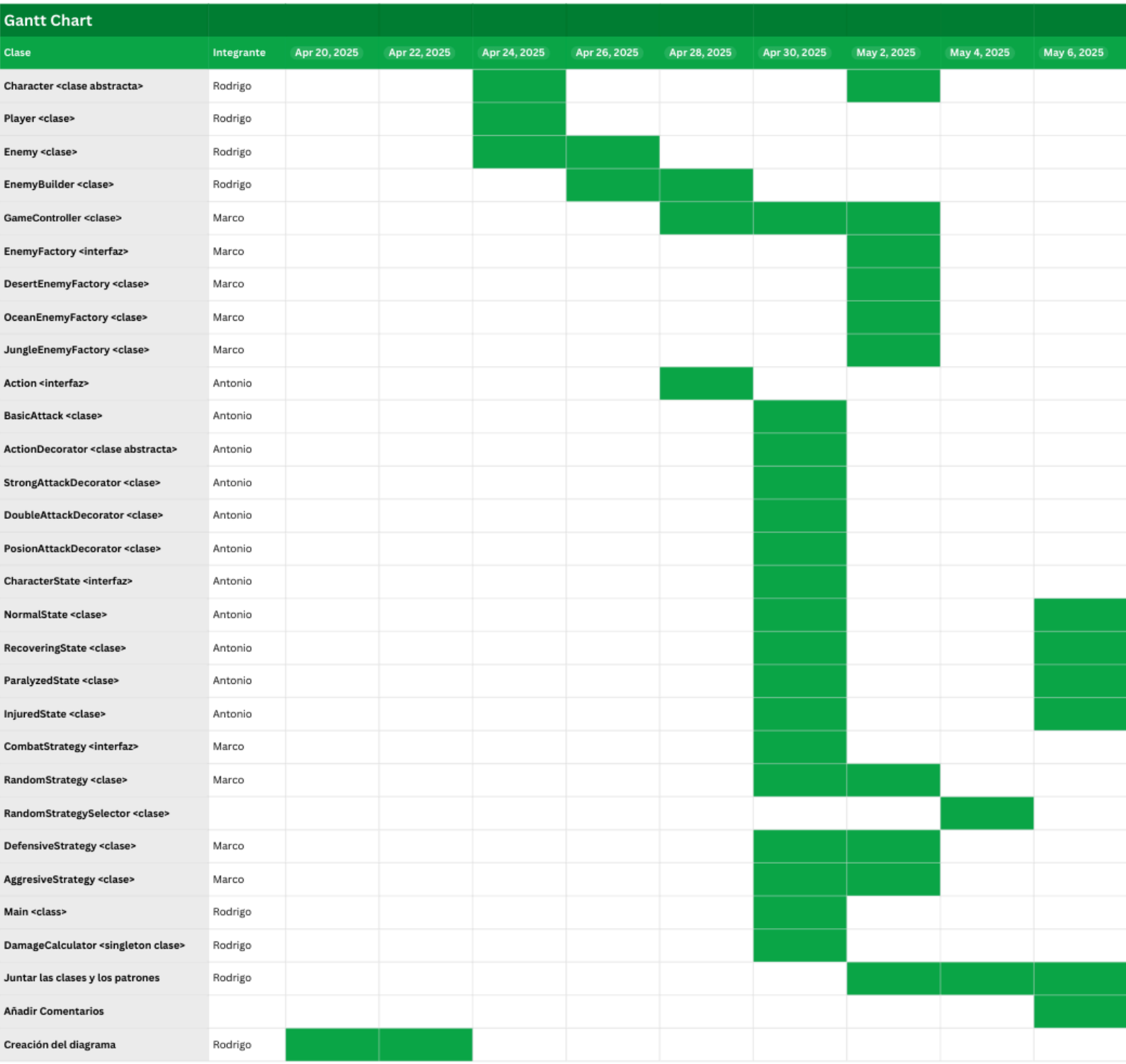
El patrón Facade lo ha hecho Marco. Ha sido implementado en el *GameController y Gamehandler* se ha usado para hacer la gestión del inicio del juego de manera ordenada gracias a los métodos de *createPlayer, createFactories, startGame*.

9. Patrón Strategy

El patrón Strategy lo ha hecho Marco. Lo usamos para gestionar la lógica de ataques del enemigo.

Para ello hemos creado las clases *CombatStrategy, DefensiveStrategy, AgressiveStrategy* y *RandomStrategy*.

DIAGRAMA DE GANTT



Faltan:

EnemyActionAssigner → hecho por Rodrigo día 08/05/2025

GameHandler → hecho por Marco y Rodrigo día 08/05/2025

Ajustes finales de varias clases en el último día.

AYUDAS

Hemos empleado IA para hacer los DEBUG de cada clase para verificar que el juego funciona como debe y para poner las estadísticas de los enemigos y no perder mucho tiempo en elegirlas

DIAGRAMA