MEMORIA DEL PROYECTO
JUEGO DE COMBATE POR TURNOS

Hecho por Jorge García e Ivan Aranda

Índice

Descripción del juego

Este proyecto consiste en un juego de combate por turnos por consola, donde el usuario controla al personaje principal y debe enfrentarse a enemigos en diferentes escenarios.

El objetivo es sobrevivir al mayor número de combates posibles consiguiendo puntos y como objetivo final derrotar a todos los enemigos existentes.

Características principales:

Combate por turnos: El sistema determina aleatoriamente quién ataca primero, y luego el jugador y el enemigo se turnan para realizar acciones. Las acciones de los enemigos se gestionan de forma aleatoria y en base a su comportamiento y atributos.
Múltiples tipos de personajes: El jugador puede enfrentarse a diferentes categorías de enemigos (Cyborgs, Demonios, Mutantes, etc.).
Estados de personaje: Los personajes pueden experimentar diferentes estados durante el combate (Sano, Envenenado, Paralizado, Quemado, o Muerto).
Estrategias de combate: Los enemigos utilizan diferentes estrategias (Agresiva, Pasiva, Neutral) que determinan su comportamiento.
Sistemas de daño: El cálculo de daño considera los atributos de los personajes y se aplica mediante el uso de una clase que calcula el daño junto con diferentes estados que aplican directamente el efecto.
Diferentes mundos: Los combates se desarrollan en diversos escenarios que afectan a las características de los enemigos.
Acciones posibles: El jugador cada turno podrá tomar diferentes decisiones y tipos de ataque o defensa.

Estructura del proyecto

El proyecto está organizado en los siguientes paquetes principales:

characters: Definición de las diferentes clases de personajes (Player, Enemy, Cyborg, Demon, Mutant y Character).
characterState: Implementación del patrón State para los diferentes estados posibles de un personaje.
actions: Acciones disponibles durante el combate

controller: Control general del juego
factories: Fábricas para la creación de enemigos.
managers: Gestores de diferentes aspectos del juego (CombatManager, WorldManager)
strategies: Estrategias de combate para los enemigos
templates: Plantilla de comportamiento para diferentes tipos de enemigos
utils: Utilidades como cálculo de daño, generadores de power-ups y decisiones aleatorias

Patrones de diseño implementados

Patrón State

Implementado en el paquete **characterState**, permite que un personaje cambie su comportamiento según su estado actual (sano, envenenado, paralizado, quemado o muerto). Cada estado modifica cómo el personaje puede interactuar durante el combate.

La clase CharacterState es una interfaz que encapsula el comportamiento asociado con un estado particular del contexto.

Los estados implementados son:

CharacterHealthyState: Estado normal, sin penalizaciones
CharacterPoisonedState: El personaje pierde salud gradualmente cada turno
CharacterParalyzedState: El personaje pierde un turno.
CharacterBurningState: Recibe daño por quemaduras cada turno
CharacterDeadState: El personaje ha sido derrotado

Estas clases concretas implementan un comportamiento asociado con un estado del contexto.

El contexto de este patrón de estado viene implementado por la clase Character, una clase muy importante que comienza con el estado base

CharacterHealthyState y se encarga de asignar el cambio de estado pertinente en base a diferentes eventos de la partida.

Patrón Strategy

Implementado en el paquete **strategies**, define diferentes estrategias de combate para los enemigos.

Estrategia: Define una interfaz común a los algoritmos que soporta. La clase que actua como estrategia es la interfaz CombatStrategy que implementaran las clases concretas de acorde al comportamiento que se le quiera dar.

Contexto: Está compuesto por un objeto de tipo Strategy e instanciado con una estrategia concreta. En este caso el contexto será EnemyBehaviorTemplate ya que se encarga de gestionar el comportamiento de los enemigos.

EstrategiaConcretas: Define una implementación a un algoritmo mediante la interfaz de Estrategia.

> **AgresiveCombatStrategy**: Prioriza ataques con alto daño.
> **PassiveCombatStrategy**: Prioriza defensa frente al ataque.
> **NeutralCombatStrategy**: Balance entre ataque y defensa.

Patrón Template Method

Implementado en **templates**, proporciona el esqueleto de comportamiento general que heredarán diferentes tipos de enemigos (Cyborg, Demon, Enemy, ...), permitiendo personalizar ciertos aspectos mientras mantiene una estructura común:

> EnemyBehaviorTemplate: Define el comportamiento base para enemigos genéricos. El método act es el identificativo de este patrón y el que se ha decidido aplicar de forma general para el resto de enemigos.

Patrón Abstract Factory

Implementado en **factories**, facilita la creación de diferentes tipos de enemigos adaptados a cada mundo. El sistema incluye:

El cliente por así llamarlo que se encarga de gestionar las fábricas y ordenar lo que se debe crear es la clase WorldManager que gestiona el sistema de mundos y enemigos.

EnemyAbstractFactory: Interfaz de factoría. Provee los métodos para que las fábricas concretas creen los objetos.

World1AbstractFactory, **World2AbstractFactory**, **World3AbstractFactory**: Factorías concretas para cada mundo específico.

El producto abstracto serán las clases de enemigos Demon, Cyborg y Mutant.

El producto concreto son las clases específicas para cada tipo de enemigo en cada mundo:

> **World1Cyborg**, **World2Cyborg**, **World3Cyborg**
> **World1Demon**, **World2Demon**, **World3Demon**
> **World1Mutant**, **World2Mutant**, **World3Mutant**

Patrón Decorator

Implementado en el paquete **actions**, se encarga de modificar los ataques segun diferentes condiciones mediante el patrón decorador.

De una forma más técnica este patrón añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad, es decir, permite agregar dinámicamente funcionalidades suplementarias a un objeto individual y no a toda la clase.

Las partes de este patrón son:

Component: Define la interfaz de los objetos a los que se les pueden asignar nuevas responsabilidades, es decir, objetos que pueden ser decorados. En este caso es ActionComponent e implementa el método principal y otros métodos auxiliares.

Decorator: Clase abstracta que mantiene una referencia al objeto Componente y define una interfaz conforme. En este caso es EffectAttackDecorator.

ConcreteDecorator: Son los decoradores concretos que asignan los estados del state Pattern, son las clases como ParalyzingAttackDecorator, PoisoningAttackDecorator o BurningAttackDecorator.

ConcreteComponent: Es el componente base que será decorado. En este caso hay dos. BasicAttackComponent que desarrolla todos los ataques y BasicDefenseComponent que desarrolla las defensas.

Patrón Singleton

Implementado en el paquete **utils**, se encarga de crear una sola instancia de una clase de forma static para que pueda ser usada directamente en otras clases. Este patrón es útil cuando queremos garantizar que solo una instancia de esa clase será creada y evita errores que pueden aparecer cuando múltiples instancias de una clase pueden estar ocurriendo.

En este caso la clase es DamageCalculator y se implementa siguiendo la técnica del patrón singleton con su respectivo getInstance().

Patrón Facade

Implementado en el paquete **controller**, este patrón permite gestionar varias clases y sirve como unificador para manejar el desarrollo general del juego. La clase se llama Game Controller e implementa toda la lógica de combates y la creación de mundos así como la creación del propio personaje. Este patrón es útil cuando quieres tener una clase que unifique el resto y gestione el comportamiento general de un proyecto. De esta forma para el usuario es mucho más sencillo utilizar el código ya que el main solo se encarga de poner en marcha esta clase y todo el comportamiento de las clases queda gestionada por la clase GameController.

Manual de Usuario

Inicio del Juego

Al iniciar el juego, se te pedirá que configures tu personaje:

1. Distribuye puntos entre los diferentes atributos (fuerza, defensa y vida). Si no se introducen valores razonables dentro de los parámetros establecidos o ocurre algún error se pedirá al usuario que vuelva a introducir nuevos valores.
2. Introducir un nombre para tu personaje

Controles Básicos

Durante los combates, interactúas con el juego mediante comandos de texto:

> Introduce el número correspondiente a la acción que deseas realizar

Acciones Disponibles

En tu turno, puedes elegir entre varias acciones:

> **Ataque básico**: Causa daño basado en tu fuerza
> **Defensa**: Reduce el daño recibido en el próximo turno
> **Ataque paralizador** (10 stamina): Paraliza al enemigo perdiendo un turno
> **Ataque quemador** (10 stamina): Quema el enemigo
> **Ataque venenoso** (10 stamina): Envenena al enemigo

Progresión

A medida que derrotas enemigos:

1. Te enfrentarás a enemigos cada vez más poderosos de diferentes mundos
2. Cada vez que derrotes enemigos ganarás puntos y mejorarás atributos.

La parte interesante principal del juego trata por un lado de conseguir la mayor puntuación posible o en su defecto eliminar a todos los enemigos. Para esto el

usuario deberá ir aprendiendo los comportamientos típicos de cada tipo de enemigo y usar su ingenio para tomar las mejores decisiones posibles.

Los enemigos realizan diferentes acciones de acorde por un lado a sus comportamientos característicos y por otro lado a factores aleatorios por lo que el azar es un importante factor en el desarrollo de la partida.

Fin del Juego

El juego termina cuando tu personaje es derrotado o cuando derrotes a todos los enemigos. Cuantos más puntos obtengas más satisfacción conseguirás.