Trabajo Práctico 1: Manejo de Clases, juego de combate

Materia: Paradigmas de la Programación.
Hecho por: Rodrigo Rosin Caruso

Introducción

En el presente trabajo práctico, se desarrolla un sistema de combate, implementando conceptos de la programación orientada a objetos (POO). Lo principal es diseñar un modelo variado y extensible, que permita la interacción entre personajes (magos y guerreros) y distintos tipos de armas (físicas y mágicas). Se utiliza herencia, polimorfismo, clases abstractas y el patrón Factory para la generación dinámica (en runtime) de entidades.

El trabajo se estructura en tres puntos principales:

- Diseño de clases: Se definen los personajes (Magos y Guerreros), y las armas (Ítems Mágicos y Armas de Combate), cada una con atributos y comportamientos únicos que influyen en el cálculo de daño y las estrategias de combate.

- Generación aleatoria: Mediante el patrón Factory, se crean personajes y armas de forma **aleatoria**, con diferentes propiedades.

- Simulación de combate: Se implementa un sistema de batalla clásico de piedra-papel-tijera, donde las acciones (ataque rápido, fuerte, o defensa y golpe) determinan el resultado de cada enfrentamiento.

La realización de todo destaca por su escalabilidad, que permite añadir nuevos tipos de personajes o armas sin modificar la lógica existente. El documento detalla las decisiones de diseño, la implementación y las interacciones entre los componentes del sistema.

Compilación

En el archivo .zip entregado, se adjuntan 3 archivos tipo *Makefile* que se utilizan para la compilación. Hay uno por ejercicio, uno en cada carpeta. Con el comando <*make*> por la terminal, una vez ubicado en la carpeta de cada ejercicio, se compilan todos los archivos necesarios para poder crear el ejecutable de cada ejercicio. Cuando efectuado, se generan los archivos .o para crear el ejecutable.

Posteriormente, se tiene que usar la línea <*./nombre_del_ejecutable*> para poder correr el ejecutable. Su nombre es Ej1.exe, Ej2.exe y Ej3.exe, respectivamente.

Ejercicio 1:

Armas

Las primeras 2 clases que implementé fueron las de Ítems Mágicos y Armas de combate. Ambas son clases abstractas, que heredan de una misma interfaz Armas.

Los Ítems Mágicos son objetos con propiedades que afectan el combate y las habilidades de los personajes. Su clase base define los atributos enteros *damage, crítico, poderMagico*, y los booleanos *consumible* y *maldito*. Cada ítem mágico hereda de esta clase y añade características propias.

- Bastón: Tiene un atributo de elemento, (el material con el que fue "construido": Abeto, Roble, Metal) que modifica su daño, peso y largo. Su daño crítico se calcula considerando la lentitud, que depende del largo/peso.
- Libro de Hechizos: Puede "legendarizar", hacer legendario, a un personaje si no está maldito. Su daño y efectividad varía según la cantidad de hojas. Cuantas más, más daño, crítico y poder mágico.
- Poción: Puede ser buena (aumenta vida) o mala (reduce vida), con una probabilidad de caerse al usarla. Su creador determina quién puede purificarla (sacarle la maldición).
- Amuleto: Dá escudo y doble magia, y puede legendarizar a un personaje si no está maldito.

Las armas de combate, en cambio, tienen otras formas de daño, con atributos como sharpness y letalidad. Las armas específicas son:

- Hacha Simple y Hacha Doble: Su material (Madera, Hierro, etc.) afecta su daño y propiedades. Pueden quebrarse, reduciendo su daño, y algunas son legendarias (como las de oro).
- Espada: Su material (Hierro, Diamante, etc.) define su aura y letalidad. Puede doblarse, reduciendo su daño, pero puede relucirse para mejorar sus atributos.
- Lanza y Garrote: Pueden estar envenenadas, aumentando su daño. La lanza tiene chance de hacer "headshot", el garrote puede añadir pinchos para mayor letalidad.

Estas armas tienen métodos para afilarse, repararse o lanzarse, y algunas pueden estar malditas o ser legendarias. El daño total considera atributos como peso, longitud y efectos puntuales (como veneno).

La lógica común que implementan ambas clases abstractas, aportan métodos como calcularDamTotal() que obtiene el daño total del ataque mediante diferentes cálculos dependiendo del estado del ítem y personaje, y los valores.

ItemMagico calcula el daño total sumando el daño crítico (hacerDamCritico()) y el ataque místico (ataqueMistico()), este último se añade solo si el ítem es místico.

ArmaCombate también usa calcularDamTotal(), y lo aborda parecido. El daño base (calcularDamage()) y daño adicional (calcularAddedDamage()) son influenciados por los atributos puntuales como filo, letalidad y otras (peso y longitud, por ejemplo), y con esto se hacen las cuentas.

Ambas clases incluyen los métodos de ataque (damAtaqueRapido(), damAtaqueFuerte() y damDefensaYGolpe()) que añaden +10 al daño total. Esta estructura permite que tanto los efectos mágicos como las propiedades físicas de las armas se integren en el cálculo de daño.

Personajes

Luego, implementé el sistema de personajes. Este está diseñado utilizando herencia y polimorfismo para crear las clases bases abstractas Guerreros y Magos. Su interfaz, Personaje, establece los métodos virtuales comunes que todos los personajes deben implementar, incluyendo los tres tipos de ataque (rápido, fuerte y defensivo), la gestión de armas (obtener, equipar y remover), y el acceso a atributos como la salud y nombre.

Los personajes mágicos, derivados de la clase Magos, tienen diversos atributos orientados al mundo mágico, como el uso de “habilidades sobrenaturales”, maná e invocaciones. Todos los magos comparten ciertas mecánicas: sus ataques se potencian cuando usan ítems mágicos (recibiendo un aumento del 33% en daño) y obtienen un bonus adicional del 50% al daño cuando alcanzan el estado legendario.

Las cuatro subclases de magos implementan enfoques distintos: los Hechiceros manejan pociones cuyos efectos escalan según el maná disponible; los Conjuradores pueden lanzar hechizos y maldecir o purificar armas; los Brujos pueden transformar y maldecir; y los Nigromantes invocan criaturas cuyo daño es afectado por su cantidad de maná.

Por otro lado, los personajes guerreros, de la clase Guerreros, tienen ataques y métodos enfocados a la fuerza física y brutalidad de los mismos. Al igual que los magos, los guerreros aumentan 33% el daño cuando usan armas físicas, y su estado legendario no sólo mejora su daño en un 50% sino que también bonifica su atributo "Fuerza del Rey".

Las cinco subclases de guerreros presentan características diferentes: los Bárbaros basan sus habilidades en atributos físicos como peso y fuerza; los Paladines combinan capacidades de combate con habilidades vinculadas a su fe; los Caballeros “utilizan” equipo pesado y ataques elementales; los Mercenarios tienen “técnicas audaces” como robar armas; y los Gladiadores recurren a rituales espartanos que consumen recursos especiales como sangreEspartana.

Todas estas, tanto para Magos como para Guerreros, en el fondo son formas distintas y únicas de hacer las cuentas del daño que devuelve el arma. Esto es lo que considero más importante de la lógica de este ejercicio: las armas no son las que hacen el daño a los personajes, sino que solo lo calculan. Los personajes son los que luego interactúan entre sí, mediante las armas que retornan el daño total a producir. Si es cierto que hay ciertos ítems que

sí tienen funciones que bajan o suben vida, pero en las funciones estrictamente necesarias para el “piedra-papel-tijera”, esto no ocurre.

Métodos y diversidad

En cuanto al sistema de combate e interacción que se podría implementar, se presentan múltiples capas estratégicas mediante la gestión de recursos (maná para magos, atributos físicos para guerreros) y el vínculo entre diferentes tipos de armas y habilidades.

Las sinergias diseñadas son las siguientes: los magos obtienen mayores beneficios de los ítems mágicos mientras que los guerreros lo hacen de las armas físicas. El estado legendario, que tiene un 10% de probabilidad de activarse al crear el personaje también influencia la estrategia, ya que añade potenciadores, tanto a los ítems como a los personajes.

Con todo esto entonces, se obtiene una base sólida de un sistema de combate RPG, con la suficiente variedad como para implementar distintas opciones de combate e interacciones entre personajes e items.

Ejercicio 2:

La clase PersonajeFactory implementa un patrón de diseño Factory que centraliza y simplifica la creación de todos los elementos del juego de forma dinámica y aleatoria. Esta “fábrica abstracta” se encarga tanto de generar personajes, Magos y Guerreros, como objetos (armas e ítems mágicos), de forma aleatoria y en runtime. Implementé los métodos crearMago(), crearGuerrero(), crearItemMagico() y crearArmaDeCombate() que proporcionan toda la lógica de creación.

También, se emplean los *enum* NombreMagos y NombreGuerreros que contienen nombres temáticos (Gandalf, Conan, etc.), que luego son asignados aleatoriamente a los personajes creados. Para los magos, que incluyen Hechiceros, Conjuradores, Brujos y Nigromantes, se establece un rango de salud entre 90-150 puntos, y a los guerreros (Bárbaros, Paladines, Caballeros, etc.) uno con 95-170 puntos de salud. Cada personaje puede llevar entre 0 y 2 armas, con un 50% de probabilidad de que sean mágicas o de combate, distribuidas equitativamente entre los distintos tipos disponibles (Pociones, Hachas, Espadas, etc.).

El sistema de generación aleatoria se inicializa una única vez mediante inicializarSemilla(), que llama a srand(time(nullptr)). Y para los rangos de números, mediante generarNumerosRandom() se permite obtener valores dentro del rango limInf y limSup específicos para controlar la aleatoriedad.

La creación de personajes en el método principal crearPersonajes() genera entre 3 y 7 magos y guerreros. En el main, justamente, se imprime por pantalla un ejemplo de la creación de los personajes, que crea entre 3 y 7 magos, y entre 3 y 7 guerreros.

Ejercicio 3:

El archivo batalla.cpp es en donde se utiliza todo lo previo, y está diseñado para simular un sistema de combate entre personajes, que pueden ser tanto guerreros como magos, cada uno con sus propias armas y/o ítems.

Se empieza por inicializar la semilla para la generación de números aleatorios, asegurando variabilidad en la ejecución. Luego, a través de la clase PersonajeFactory, se crean los personajes disponibles (como fue mencionado, de forma aleatoria). El usuario debe seleccionar uno de ellos, y se valida que su elección esté dentro de los límites aceptables. Si el personaje elegido tiene armas, se le permite al usuario seleccionar una como la actual. En caso de que el personaje no tenga armas, el sistema lo informa.

El oponente, posteriormente, es seleccionado de forma aleatoria entre los personajes restantes, asegurándose de que no sea el mismo que eligió el jugador. También se muestran los detalles del enemigo.

Una vez terminado esto, se inicia el combate. Este se desarrolla mientras ambos personajes conserven puntos de vida. En cada iteración del bucle, el jugador debe elegir una acción ofensiva, mientras que el enemigo selecciona la suya aleatoriamente. Las combinaciones posibles de acciones se analizan una por una, y siguen la lógica del piedra-papel-tijera de la consigna. Dependiendo de la acción y del tipo de arma, se calcula el daño infligido utilizando métodos como ataqueRapido(), ataqueFuerte() o defensaYGolpe(). Si un personaje no tiene armas, el daño se reduce a un valor fijo –10. Después de cada intercambio de ataques, se actualizan los puntos de vida de ambos personajes. El bucle se interrumpe si alguno de ellos llega a cero.

Al finalizar el combate, el programa informa el resultado: puede ser una victoria, una derrota o un empate si ambos personajes quedan sin vida simultáneamente.

Conclusiones

El TP me ayudó mucho a entender polimorfismo de forma adecuada, y cómo manejar cuando uno tiene muchos warnings e ir corrigiéndolos. Otra cosa que me interpeló fue el cómo implementar bien los #include, y como asignar atributos y crear métodos necesarios, sin crear de más.