Trabajo Práctico 1

Gonzalo Malaspina

Universidad de San Andrés

Compilación: para compilar se debe tener instalado g++ y make, se utilizó el makefile para facilitar la compilación. Se utilizaron las warnings Wall y Wextra, para evitar errores y facilitar la identificación de los mismos.

Ejecución: en ej2 y ej3 se encuentran los únicos 2 makefile de todo el trabajo práctico, para compilar se debe parar dentro de la carpeta ej2 (para probar el ej1 y ej2) y compilar con make, luego ejecutar make run y se puede borrar los compilados con make clean, repetir en ej3 para probar el simulador de batalla.

1. Introducción

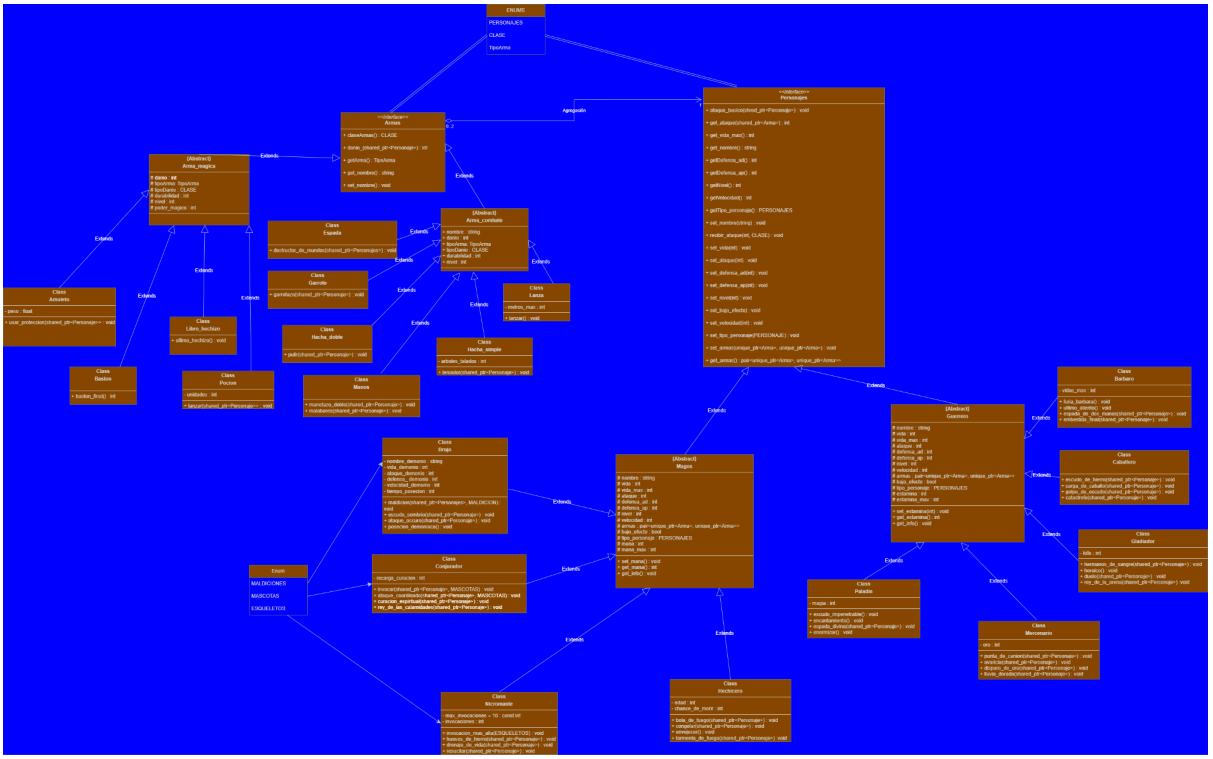
Este informe propone explicar el funcionamiento del repositorio Tp1-Paradigmas. La idea principal de este proyecto es crear un conjunto de clases que representan diferentes tipos de armas y personajes, organizadas en jerarquías que son tanto lógicas como extensibles. Usando conceptos clave como herencia, polimorfismo, clases abstractas e interfaces.

2. Estructura del Proyecto

El proyecto está formado por varios archivos .hpp y .cpp, donde cada uno contiene definiciones e implementaciones específicas para armas, personajes, utilidades adicionales como la fábrica de personajes y la simulación de peleas. Esta organización modular del código facilita la identificación de las funcionalidades y mantiene una separación clara de responsabilidades.

Cada subcarpeta sigue una jerarquía coherente de clases que facilita tanto la extensión como la comprensión del sistema. Las clases están organizadas en archivos individuales para seguir el principio de responsabilidad única.

Adjunto UML:



3. Clases Abstractas e Interfaces

Siguiendo con lo pedido en la primera consigna, se utilizaron interfaces comunes para los personajes y otra para las armas, que luego son adaptadas por clases abstractas para diferenciar los tipos (mágicas/magos o combate/guerreros), para por último obtener clases concretas (brujo, garrote, etc..).

3.1. Armas

Esta es la interfaz común para todas las armas del juego. Incluye únicamente métodos virtuales puros como, y todos completamente públicos, ya que una interfaz no puede contar con métodos privados o protegidos.

Además, el hecho que todos los métodos sean virtuales puro, obligan a que todas las clases derivadas implementen estos métodos, asegurando una estructura

consistente. Gracias a esto, los personajes pueden interactuar con cualquier arma sin preocuparse por su tipo específico, lo que introduce el polimorfismo.

3.2. Personaje

De manera similar, esta clase actúa como una interfaz que define las funcionalidades que cualquier personaje en el sistema debería tener. Conteniendo únicamente métodos virtuales.

Esto permite que un puntero a Personaje pueda apuntar a un Guerrero, un Brujo, un Paladín, etc., sin perder ninguna funcionalidad. Así, se logra una independencia entre la lógica del programa y las clases concretas.

3.3. Clases Abstractas: Arma_combate y Arma_magica

Estas clases funcionan como interfaces especializadas que heredan de armas(interfaz). Aunque no implementan directamente funcionalidades, sí definen categorías:

- Arma_combate: espadas, garrotes, lanzas, hacha simple, hacha doble y manos.
- Arma_magica: pociones, bastones, libros de hechizos y amuletos.

Este paso intermedio dentro de la jerarquía permite aplicar lógica específica a cada grupo de armas, además que nos permite definir métodos no virtuales, y definir atributos generales para las posteriores clases concretas (para que se puedan heredar deben ser protegidos).

3.4 Clases Abstractas: Magos y Guerreros

Al igual que en el caso de las armas, estas clases funcionan como un segundo interfaz, pero con métodos y atributos heredables, convirtiéndolos en clases abstractas.

4. Jerarquía de Armas

El sistema de armas en el repositorio se divide en dos grandes familias: armas físicas y armas mágicas, ambas heredando del interfaz Armas. Esta jerarquía está diseñada para ofrecer flexibilidad, reutilización de código y facilidad de ampliación. Cada arma concreta tiene sus propios valores de daño físico y mágico, así como un nombre distintivo.

4.1. Armas físicas (Arma_combate)

Estas clases representan objetos tradicionales usados en combate cuerpo a cuerpo. Heredan de Arma_combate, que a su vez hereda de armas(interfaz).

.

Espada:

- destructor_de_mundos() -> aumenta el daño del arma, pero se destruye después de utilizarse.

Garrote:

- garrotazo() -> el portador lanza el garrote, realizando daño al enemigo

Hacha doble:

- pulir() -> aumenta la defensa ad y ap del usuario

Hacha simple:

- leñador() -> tala un árbol y lleva la cuenta con un atributo en la clase, de cuantos árboles taló.

Lanza:

- lanzar() -> lanza la lanza y con un random calcula cuántos metros se lanzó, se guarda la distancia máxima en un atributo.

Manos:

"Clase creada por mi"

- manotazo_doble() -> lanza un doble manotazo al enemigo, hace x2 de daño
- malabares() -> ejecuta un cout.

4.2. Armas mágicas (Arma_magica)

Estas armas cuentan con poder mágico. También heredan de armas, pero a través de la subclase Arma_magica.

Poción:

- lanzar() -> si le quedan pociones (max 10), puede lanzar una poción e infligir daño

Bastón:

- baston_final() -> aumenta el poder mágico, se destruye al finalizar

Libro de Hechizos:

- ultimo_hechizo() -> aumenta el daño y poder mágico del libro, luego se destruye

Amuleto:

- usar_proteccion() -> aumenta la defensa ap del portador

Este diseño permite que los personajes mágicos interactúen exclusivamente con armas mágicas, respetando la lógica del mundo que se quiere representar.

4.3. Métodos comunes

Gracias a la herencia, todas las armas implementan métodos definidos en la interfaz e implementados en sus respectivas clases abstractas.

Este diseño facilita el trabajo del desarrollador: desde el punto de vista del personaje, todas las armas son intercambiables siempre que se respeten las reglas de clase.

5. Jerarquía de Personajes

Los personajes se organizan en una jerarquía que proviene del interfaz Personaje. Cada clase concreta tiene su predecesor abstracto dentro del universo del juego y está diseñada para manejar un tipo específico de daño, ya sea físico(guerreros) o mágico(magos).

5.1 Guerreros (Guerrero)

Esta clase representa a los personajes que cuentan con daño físico y heredan todos los métodos del interfaz Personajes. Esta clase permite crear punteros a sí misma

Bárbaro:

furia_barbara() -> aumenta el ataque

ultimo_aliento() -> si muere el barbaro, revive (hasta 2 veces)

espada_de_dos_manos() -> un ataque q duplica el daño del arma

embestida_final() -> un ataque que cuadruplica el daño del arma

Caballero:

escudo_de_hierro() -> escuda a un aliado, le aumenta la defensa ad

carga_de_caballo() -> ataque que duplica el daño del arma

golpe_de_escudo() -> ataque con arma

catastrofe() -> ataque que triplica el daño del arma

Gladiador:

hermanos_de_sangre() -> le baja las defensas del enemigo a 0 y le aumenta el daño

heroico() -> aumenta el daño y la defensa

duelo() -> ataca al enemigo con un arma

rey_de_la_arena() -> si le faltan 50hp se cura y aumenta su daño por cada vez que haya matado a un enemigo, luego ataca.

Mercenario:

punta_de_cañon() -> ataca con el arma

avaricia() -> roba oro

disparo_de_oro() -> dispara con el arma

lluvia_dorada() -> consume todo el oro y ejecuta un ataque multiplicado por el oro gastado

Paladín:

escudo_impenetrable() -> aumento de las defensas

encantamiento() -> aumenta su magia

espada_divina() -> consume toda su magia, ataca al enemigo realizando un daño extra por magia acumulada

enormizar() -> aumento de varias estadisticas y disminucion de velocidad

5.2 Magos (Magos)

Esta clase representa a los personajes con habilidades mágicas y también es un derivado del interfaz Personajes. Esta clase permite crear punteros a sí misma

Brujo:

maldicion() -> lanza una maldicion

escudo_sombrio() -> le da un escudo magico a un aliado

ataque_sombrio() -> ataca con un arma

posecion_demoniaca() -> se transforma en un demonio

Conjurador:

invocar() -> invoca una de sus mascotas

ataque_coordinado() -> hace un ataque coordinado con su mascota

curacion_espiritual() -> cura a un aliado +20

rey_de_las_calamidades() -> ataca al enemigo

Hechicero:

bola_de_fuego() -> lanza una bola de fuego

congelar() -> congela al enemigo

envejecer() -> gana años y chances de morir, despues se usa un random para ver si murio

tormenta_de_fuego() -> lanza una tormenta de fuego que se potencia con la edad del brujo

Nigromante:

invocacion_mas_alla() -> invoca un esqueleto (max 10)

huesos_de_hierro() -> aumenta la defensa ad de algún aliado

drenaje_de_vida() -> drena parte de la vida del rival

resucitar() -> resucitar a un aliado con max hp

Para concluir, se creó la clase fabrica de personajes, con la cual se crea en runtime los personajes, sin tener que instanciar y además se implementó el sistema de pelea al estilo piedra-papel-tijeras. Se puede comprobar su correcto funcionamiento corriendo el código, con la compilación y ejecución correcta.