**Informe Trabajo Practico 1**

**Paradigmas de Programación**

**Lautaro Emiliano Lo Presti**

**Universidad de San Andres**

**Introducción**

Este trabajo se enfocó en el desarrollo de un sistema orientado a objetos en C++, ambientado en un juego de rol. La consigna propuso dividir el trabajo en tres etapas: desde el diseño de clases hasta la simulación de combates. El objetivo fue aplicar principios como herencia, polimorfismo, abstracción y composición.

Se trabajó con distintos tipos de personajes y armas, se incorporó generación aleatoria de datos y se programó una dinámica de combate simple pero funcional. En este informe se detalla cómo se resolvió cada parte del desarrollo y los comandos utilizados para compilar correctamente el proyecto.

**Ejercicio 1**

**Jerarquía de Armas**

Para representar las armas, se creó una interfaz base Arma en el archivo Arma.hpp, de la cual derivan dos clases abstractas:

* ArmaCombate para armas físicas (espada, lanza, garrote, hacha simple y hacha doble).
* ArmaMagica para ítems mágicos (poción, amuleto, bastón y libro de hechizos).

Cada clase implementa al menos cinco atributos (como daño base, durabilidad, velocidad de ataque, etc.) y métodos necesarios para manipular esos datos. Las clases se organizaron en carpetas separadas según su tipo (Combate y Magicas).

**Jerarquía de Armas**

De forma similar, los personajes se definieron a partir de una interfaz Personaje (en Personaje.hpp), con dos clases abstractas derivadas:

* Guerreros: incluye clases como Barbaro, Paladin, Caballero, Mercenario y Gladiador.
* Magos: incluye Hechicero, Conjurador, Brujo y Nigromante.

Cada personaje tiene atributos como vida, energía, resistencia, armadura y nombre, y métodos para interactuar con el entorno (atacar, recibir daño, mostrar información, etc.). También se implementó la relación de composición con armas (has-a), permitiendo que cada personaje tenga un par de armas asignadas.

Diagrama

El contenido generado por IA puede ser incorrecto.**Organización**Todos los archivos del ejercicio se encuentran en la carpeta Ejercicio\_1/. El archivo main1.cpp se utilizó para probar la creación e interacción básica entre personajes y armas. Además, se incluyó un diagrama UML (UML\_Ejercicio1.png) que resume la estructura de clases implementada.

**Ejercicio 2**

**Generador Aleatorio**

Para generar personajes y armas automáticamente, se utilizó la función rand() junto con srand() inicializada con time(0). Esto permitió simular decisiones aleatorias dentro de los rangos definidos:

* Un número entre 3 y 7 para la cantidad de personajes tipo mago y guerrero.
* Un número entre 0 y 2 para decidir cuántas armas tendrá cada uno.

**Clase Factory**

Se implementó una clase llamada Factory (en Factory.hpp y Factory.cpp) que actúa como fábrica de objetos y permite instanciar en tiempo de ejecución tanto personajes como armas. Esta clase tiene métodos como:

* personaje\_desarmado(Personajes *tipo*)
* crear\_arma(Armas *tipo*)
* personaje\_armado(Personajes *tipo*, pair<unique\_ptr<Arma>, unique\_ptr<Arma>> *armas*)

Todos los objetos generados (personajes y armas) son devueltos como unique\_ptr, lo que garantiza un buen manejo de memoria y evita fugas.

**Organización**

Todo el código está en la carpeta Ejercicio\_2/. En main2.cpp se implementó una prueba que genera una cantidad aleatoria de personajes (magos y guerreros), les asigna armas también aleatorias, y muestra por pantalla la información generada para verificar que se hayan creado correctamente.

**Ejercicio 3**

**Mecánica del Juego**

Se implementó una batalla por turnos entre dos personajes, con una lógica similar a piedra-papel-tijera. Las acciones posibles son:

1. Golpe Fuerte
2. Golpe Rápido
3. Defensa y Golpe

La interacción entre ellas es la siguiente:

* Golpe Fuerte gana a Golpe Rápido → hace 10 de daño.
* Golpe Rápido gana a Defensa y Golpe → hace 10 de daño.
* Defensa y Golpe bloquea Golpe Fuerte y contraataca → hace 10 de daño.
* Si ambos personajes eligen la misma acción → no hay daño, se repite el turno.

El combate termina cuando uno de los personajes llega a 0 HP.

**Elección de acciones**

* El **Jugador 1** elige su acción manualmente desde teclado.
* El **Jugador 2** selecciona su acción de forma aleatoria usando rand().

Esto se repite en cada ronda, actualizando los puntos de vida (HP) y mostrando la situación actualizada tras cada ataque.

**Interacción con armas**

Cada personaje inicia con un arma previamente asignada (aunque en esta versión básica las armas no modifican el daño, se podrían extender fácilmente para hacerlo). En cada turno, se informa qué personaje atacó, qué acción eligió y con qué arma lo hizo.

**Interfaz de juego**

El main3.cpp implementa un pequeño menú que muestra:

* Los HP actuales de ambos jugadores.
* La acción seleccionada por el jugador.
* El resultado de la ronda (daño infligido o empate).

**Organización**

Todo el código de este ejercicio está en la carpeta Ejercicio\_3/, donde se encuentran main3.cpp y auxiliares.hpp, que incluye funciones de utilidad para la simulación del combate.

**Compilacion**

Para realizar la compilación es necesario tener instalados g++ y make. Hay un único archivo makefile para todo el trabajo, solamente tenes que ejecutar en terminal el comando make seguido del número de ejercicio:

* make ej1
* make ej2
* make ej3

Estos comandos van a compilar y ejecutar el programa. Si desea eliminar los archivos de compilación debe ejecutar el comando “make clean” en la terminal.

**Warnings del compilador**

Durante el desarrollo, no se detectaron warnings al compilar con las opciones utilizadas.