Trabajo Práctico 1

I102 – Paradigmas de Programación – G:1

Alumno: Baltasar Oostdijk Laumann

Numero de Legajo: 36617

Universidad: Universidad de San Andrés (UdeSA)

Fecha: 13/04/2025

Introducción

En el siguiente informe se detalla la resolución del Trabajo Practico 1 de la materia de Paradigmas de la Programación (I102) junto a Mariano Scaramal, profesor de las clases teóricas y Marcos Attwell, profesor de las clases tutoriales. Todos los programas referidos a la resolución de cada uno de los ejercicios fueron compilados mediante un MakeFile. Además, se utilizaron los flags adecuados para asegurar que el código no genere ningún warning. También se hace uso de un archivo “utilities.h” en el que se incluyen todas las librerías necesarias para todos los ejercicios junto a la definición de varios enum class.

En caso de querer compilar los ejercicios, basta con asegurarse de estar posicionado sobre la carpeta I102_TP1_G1_Oostdijk e ingresar el comando “make” junto al ejercicio que se desea correr (ej1, ej2, ej3) en la terminal. Lo que generara tres ejecutables, un “ejercicio1”, un “ejercicio2” y un “ejercicio3”, correspondientes a cada ejercicio.

1. Ejercicio 1 – Definición de Clases

En este ejercicio se definieron, por un lado, la interfaz Armas y, por otro, la interfaz Personajes. Ambas interfaces permiten hacer uso de la herencia y polimorfismo para .... . Heredando de la primera, se definen dos clases abstractas, ítems_magicos y armas_combate. Heredando de la segunda, se definen otras dos clases abstractas, magos y guerreros.

En lo que respecta a las clases derivadas, cada interfaz cuenta con 9 armas y 9 personajes. Siguen una distribución de 4 armas en ítems mágicos, 5 en armas de combate, 4 personajes en magos y 5 personajes en guerreros.

Atributos y Métodos de las Armas

Centrado en las Armas, todas cuentas con atributos protected como Tipo, Ataque, Durabilidad, Nivel y Subtipo, aunque estas se definen en las clases abstractas para mantener la propiedad de interfaz de la clase Armas. Sumado a eso, dependiente del tipo de arma, se pueden encontrar mas atributos como Duración y Energía, en el caso de los ítems mágicos, y Peso, Alcance y Velocidad, para el caso de las armas de combate. Todos los mencionados son atributos tipo int a excepción de Tipo y Subtipo, que son tipo const string.

A pesar de no estar implementada en el código, la idea es utilizar dichos atributos para volver más dinámico a los enfrentamientos, en los que las armas podrían romperse en caso de agotar su durabilidad, costar más energía dependiendo de su peso, alcance y velocidad, afectar al enemigo por más turnos dependiendo de su duración o aumentar su ataque y velocidad en caso de ser de mayor nivel.

En cuanto a los métodos, se definen en la interfaz de Armas como métodos virtuales puros tanto a los getters para el Tipo, Subtipo, Ataque, Durabilidad y Nivel, como dos funciones tipo void llamadas ataque_primario() y ataque_secundario() para ofrecer a cada arma dos ataques distintos que, a pesar de no estar implementado, podría ofrecer mas aleatoriedad a los enfrentamientos pudiendo definir a un ataque más fuerte o veloz que otro. Como las clases abstractas poseen atributos particulares a ellas (energía, duración, peso, alcance y velocidad), sus getters se definen en las clases Item_mag y Arma_comb como virtuales puros junto a sus respectivos constructores y destructores.

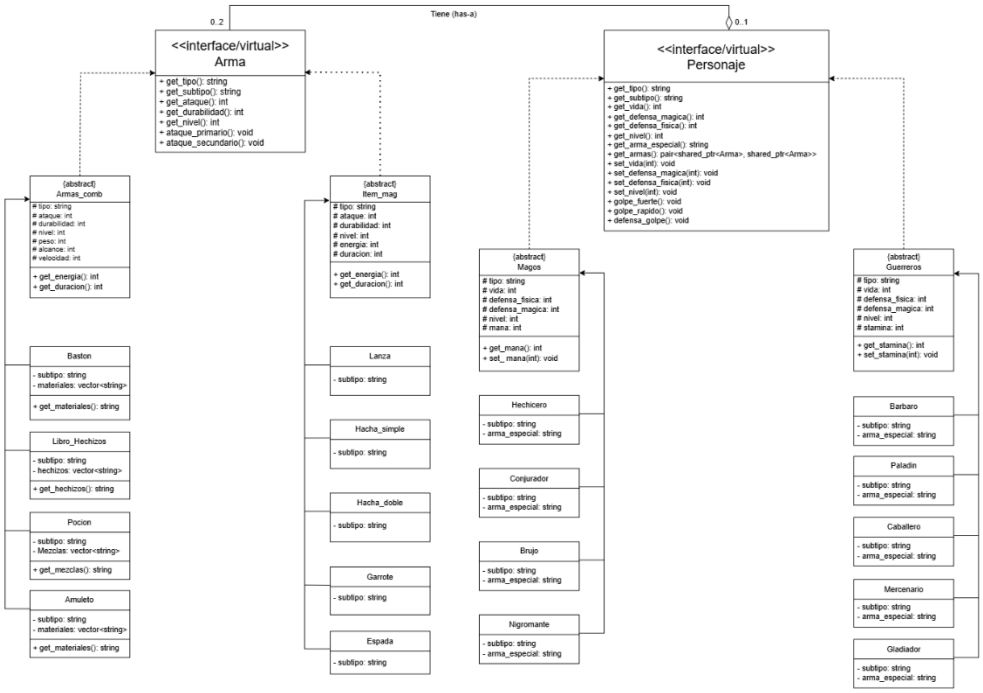
Atributos y Métodos de los Personajes

Continuando con los Personajes, todos ellos cuentan con atributos protected como Tipo, Vida, Defensa Magica, Defensa Física, Nivel, Armas y Subtipo. Sin embargo, todos se definen en las clases abstractas para mantener la propiedad de interfaz de la clase Personajes. Adicionalmente, se definen mas atributos privados como Mana, en el caso de los magos, y Stamina, en el caso de los guerreros.

Ambos atributos cumplen la misma funcion de definir el personaje es capaz de atacar o esta agotado. Todos los atributos mencionados son de tipo int a excepción de Tipo y Subtipo, que son const string, y Armas que es un pair de shared pointers a objetos de tipo Arma.

Una adición particular al sistema de armas, personajes y enfrentamientos es lo que se define como Tipo Especializado y Arma Especializada. Cada tipo de personaje (mago o guerrero) se especializa en un tipo de arma (item mágico o arma de combate). Es así que, si un guerrero utiliza un arma de combate, este realizara un 10% de daño adicional contrario a que si utilizase un item mágico. Además, cada subtipo de personaje (Hechicero, Paladin, Conjurador y demas) se especializa en un único subtipo de arma (Bastón, Lanza, Amuleto y demas) si un personaje utiliza el arma en el que esta especializado realizara un 5% de daño adicional al anterior 10% sumando un total de 15% de daño adicional.

En cuanto a los métodos, se definen en la interfaz de Personajes, como métodos virtuales puros, a los getters y setters para el Tipo, Subtipo, Vida, Defensa Magica, Defensa Física, Armas, Arma Especializada. También se definen los tipos de ataque para el enfrentamiento, ofreciendo así la opción de especificar los ataques de cada uno de los personajes para aumentar el nivel de realidad y hacer más dinámico al enfrentamiento. Como las clases abstractas poseen atributos particulares a ellas (Mana y Stamina), sus getters y setters se definen en las clases Magos y Guerreros como virtuales puros junto a sus respectivos constructores y destructores.



(Figura 1) Diagrama UML de clases

2. Ejercicio 2: Fabrica de Personajes

El objetivo del Ejercicio 2 fue desarrollar un sistema que permita generar personajes armados de manera aleatoria utilizando el patrón de diseño Factory. El sistema diferencia entre dos tipos principales de personajes: magos y guerreros, y para cada uno se asignan dos armas elegidas al azar entre todas las disponibles. En todos los casos se utilizan punteros inteligentes de tipo shared pointer para manejar de forma segura las asignaciones de memoria. También se hace uso de varios enum class para poder generar un numero aleatorio y asignarle un tipo de personaje o arma según se necesite.

Para implementar este comportamiento se utilizó una clase PersonajeFactory, la cual se encarga de instanciar personajes y armas a partir de los enum class Personas y Armas_lista. Esta clase concentra toda la lógica de creación, permitiendo que el resto del sistema trabaje con interfaces más abstractas y desacopladas del detalle de implementación.

Las funciones de utilidad que permiten generar valores aleatorios, seleccionar armas y construir personajes con sus respectivas armas se encuentran en source_ej2.cpp y están declaradas en source_ej2.h. Estas funciones permiten crear una cierta cantidad aleatoria de magos y guerreros con armas randomizadas a partir de las librerías cstdlib y ctime. Finalmente la imprimir_personajes() junto a mostrar_personaje() facilita la impresión de cada uno de los personajes por la terminal

El archivo main_ej2.cpp integra todos estos componentes, generando listas de personajes magos y guerreros, armados de forma aleatoria, y los muestra por pantalla. Todo esto se organiza en un entorno modular y reutilizable, facilitando su integración con otros ejercicios del proyecto.

3. Ejercicio 3: Enfrentamientos Simulados

En este caso, se debía simular un enfrentamiento entre dos personajes, utilizando el código implementado en el ejercicio 2 para crear dichos personajes. El usuario debe elegir tanto el personaje como las dos armas y, luego, batallar contra otro generado aleatoriamente.

El combate guarda semejanzas con el famoso piedra-papel-tijera, ofreciendo a los personajes 3 ataques distintos (“Golpe Fuerte”, “Golpe Rápido” y “Defensa y Golpe”). Las reglas son idénticas al mencionado juego, Golpe Fuerte vence a Golpe Rápido, Golpe Rápido vence a Defensa y Golpe y Defensa y Golpe vence a Golpe Fuerte. El enfrentamiento acaba cuando alguno de los personajes se queda sin vida y todas las rondas del enfrentamiento se imprimen en la terminal de la siguiente manera.