Proyecto Integrador

Esperanza Román Meléndez A01739617

Instituto Tecnológico y de Estudios Superiores de Monterrey

Programación orientada a objetos (Gpo 851)

León Felipe Guevara Chávez

Verano 2025

29 de julio de 2025

Índice

Introducción

Se nos planteó como reto crear un simulador de batallas, pensando en un universo que me gusta y que a la vez permitiera desarrollar distintos tipos de personajes con habilidades, atributos y comportamientos, decidí basar mi proyecto en el mundo mágico de Harry Potter, específicamente en uno de los momentos más icónicos de la saga que es la batalla de Hogwarts. Así que, a lo largo del desarrollo de este simulador, se implementaron conceptos como herencia, polimorfismo, clases abstractas, sobrecarga de operadores, etc. Además busqué que la simulación no solo fuera funcional sino también entretenida de hacer y coherente con la historia del universo.

El simulador permite enfrentar a dos ejércitos que son el ejército de Dumbledore y el ejército de Voldemort compuestos por personajes como aurores, elfos y mortífagos, quienes luchan hasta que un bando gana, además se me ocurrió que cada personaje tuviera características propias por lo que cada batalla es diferente.

Desarrollo

Todo parte de una clase base llamada personaje que contiene los atributos y comportamientos que comparten todos los personajes del mundo mágico como el nombre, la salud, el ataque y el nivel, además de métodos para acceder y modificar estos datos, con lo que se llega a establecer una base común desde la cual se pueden construir personajes más específicos. A partir de esta clase base, se derivan otras clases como Auror, Elfo y Mortífago, que representan distintos tipos de combatientes y cada una modifica algunos aspectos particulares del personaje para reflejar el rol en el universo en el que se basa el programa. Así que gracias a esta estructura no fue necesario repetir código para cada tipo de personaje, ya que todos heredan los elementos comunes desde Personaje y solo se definen los elementos propios cuando es necesario, como habilidades especiales o atributos únicos de cada clase derivada.

Además como cada tipo de personaje tiene comportamientos distintos cuando ataca o recibe daño se utilizó polimorfismo , con lo que se permitió que el programa decida automáticamente cuál versión de un método debe usarse dependiendo del tipo específico del personaje, así por ejemplo que cuando se llama al método atacar o imprimir se ejecuta la versión correcta para cada tipo de personaje sin necesitar de condicionales, esto se logra con el uso de métodos virtuales que permiten que el comportamiento se adapte al tipo del objeto lo hace que el código sea más limpio y más fácil. Por otro lado, para asegurarse que todas las clases derivadas implementaran su propia lógica de funcionamiento en algunos aspectos esenciales, como determinar si el personaje sigue con vida se declaró un método en la clase base, el cual obliga a que todas las clases hijas lo definan, así que con esto garantizamos que cada personaje tenga una lógica que se adapte a sus habilidades.

Así mismo, se implementó la sobrecarga, con la que fue posible simplificar la forma de mostrar la información de un personaje en pantalla y también se incorporó excepciones para el manejo de errores durante el proceso de carga de los archivos que contienen la información de los ejércitos. Para ello se incluyó una validación que detecta si un archivo no se pudo abrir correctamente y en ese caso en lugar de continuar con un error silencioso el programa lanza una excepción, lo que permite detener la ejecución y mostrar un mensaje de advertencia al usuario.

Desmenuzando los archivos…

Personaje

Con el archivo Personaje.hpp se definen qué datos y funciones tiene un personaje en general, ahí se declaran atributos como vida, salud, ataque, nivel y nombre, que son básicos para todos los personajes. También tiene métodos para obtener y modificar esos atributos, como getVida(), setVida(), getSalud(), setSalud() y así con los demás, para que otras partes del programa puedan saber o cambiar esos valores, asimismo se incluyen funciones para mostrar el porcentaje de salud, imprimir una barra visual que indica cuánta vida le queda al personaje y los métodos para atacar, recibir ataques e imprimir sus datos, sin embargo hay un método llamado estaVivo() que no tiene implementación porque cada tipo de personaje debe definirlo, lo que asegura que todos tengan su propia forma de decir si están vivos o no. También sobrecarga el operador para facilitar imprimir información básica del personaje en consola.

Ahora hablando del archivo cpp, ahí es donde se encuentran las implementaciones de todos esos métodos que estaban declarados en el hpp, por ejemplo el constructor por defecto asigna valores iniciales básicos como vida en 100, salud igual a vida, ataque en 10, nivel 1 y nombre vacío, mientras que el constructor con parámetros permite crear personajes con valores personalizados, las funciones get y set simplemente devuelven o actualizan el valor correspondiente. El método porcentajeSalud() calcula el porcentaje de vida restante dividiendo la salud actual entre la vida total y multiplicando por 100, y la función imprimeBarra() usa ese porcentaje para dibujar en la consola una barra hecha con caracteres, donde los símbolos # indican la salud restante y los - la parte vacía, mostrando también el porcentaje numérico después

Y cuando el personaje recibe un ataque, el método recibeAtaque() reduce la salud restando los puntos de ataque recibidos, y si la salud baja de cero se pone en cero para no tener valores negativos, así para atacar a otro personaje, está la función atacar() qie calcula el daño basándose en el nivel de ambos personajes, o sea que si el objetivo tiene un nivel mayor, el daño es menor y se calcula como un número aleatorio entre 1 y la mitad del ataque y si el nivel es igual o menor, el daño es más fuerte, con un rango mayor. Después se llama al método recibeAtaque() del objetivo para descontar ese daño, y después al método imprimir()

que muestra en pantalla todos los datos importantes del personaje, como nombre, vida total, salud actual, ataque y nivel, y también llama a imprimeBarra() para mostrar la barra de salud.

Clases hijas: Auror, Elfo y Mortífago

Para empezar, con Auror se define que hereda de la clase Personaje, pero aquí se establecen sus atributos especiales como el poder del patronus y la resistencia a la magia oscura. El código tiene un constructor que inicializa estos atributos y otro que los recibe como parámetros para crear un auror con características específicas, también tiene métodos para obtener y cambiar esos valores y la función imprimir muestra toda la información del auror, incluyendo sus atributos especiales. En cambio, en la función atacar el auror hace daño al objetivo, y si el objetivo es un Mortífago el daño aumenta según el poder del patronus (que siguiendo la historia del mundo mágico no tiene mucha lógica pero fue necesario para las implementaciones deseadas). Cuando recibe un ataque puede reducir el daño según su resistencia a la magia oscura y además tiene un método que verifica si está vivo, si su salud baja a cero puede usar su poder del patronus para curarse un poco y seguir luchando, siempre que tenga suficiente energía.

El archivo de Elfo describe el comportamiento de los personajes de tipo Elfo, que también heredan de Personaje. Aquí se definen atributos únicos como el poder élfico, la magia curativa y si el elfo es libre o no, hay dos constructores, uno por defecto y otro con parámetros que inicializan esos atributos. Los métodos para obtener y cambiar esos atributos están presentes y la función imprimir muestra todos los datos, incluyendo los propios del elfo. El ataque del elfo es similar al de auror pero siempre suma un bonus basado en su poder élfico y al recibir daño si el elfo es libre puede reducir el daño recibido y además puede usar su magia curativa para sanar un poco y también tiene un método para verificar si sigue vivo, que le permite curarse automáticamente si tiene magia suficiente y es libre.

En último lugar tenemos a los Mortifagos, que tienen atributos como el nivel de oscuridad y si tienen la marca tenebrosa, hay constructores para inicializar estos valores, y métodos para obtener y modificar esos atributos. Con la función imprimir se muestran sus datos, incluyendo esos atributos y en el ataque, el Mortífago hace daño parecido a los otros, pero si su nivel de oscuridad es alto tiene chance de hacer un ataque doble y al recibir daño solo se resta salud directamente. El método que verifica si sigue vivo permite que si tiene suficiente

nivel de oscuridad y la marca tenebrosa, pueda sobrevivir con salud mínima y pierde parte de su nivel de oscuridad para mantenerse en pie.

Exercise.cpp

Primero se declaran dos arreglos para los ejércitos, uno para los personajes de Dumbledore y otro para los de Voldemort. Ambos tienen un tamaño máximo de 20 y también se lleva un contador que indica cuántos personajes hay en cada ejército. Después de eso se intenta cargar la información desde los archivos de texto que contienen los datos de los personajes, pero si ocurre algún problema al abrir alguno de los archivos, se lanza un error y el programa se detiene. Si todo va bien, se imprimen los datos de los personajes de cada ejército en pantalla para que el usuario vea las estadísticas de cómo están antes de iniciar la batalla.

Cuando empieza la función batalla, se imprime un mensaje que anuncia el inicio del combate, se usan dos variables “i1” y “i2” que indican cuál personaje de cada ejército va a atacar, primero ataca un personaje del ejército de Dumbledore al del ejército de Voldemort y si el personaje atacado muere, se muestra un mensaje de que ha muerto y se guarda su nombre en un arreglo de bajas, se libera la memoria del personaje y se reemplaza en el arreglo por el último personaje del ejército, además de que también se ajusta el tamaño del ejército y se verifica que no se haya salido del rango.

Luego si todavía queda alguien vivo en el otro ejército, ese personaje contraataca, se repite el mismo proceso pero ahora con el personaje del ejército de Dumbledore como víctima. Esto continúa en rondas hasta que uno de los dos ejércitos se queda sin personajes vivos. O sea que supongamos que en la primera ronda, Harry ataca a Bellatrix, si Bellatrix no sobrevive entonces el programa imprime que Bellatrix ha muerto, guarda su nombre como una baja, libera su memoria y saca al personaje del arreglo, reemplazándola por el último mortífago que quede, ahora, si todavía queda algún personaje en el ejército de Voldemort, Lucius o el siguiente disponible toma el lugar y contraataca, supongamos que Lucius ahora le devuelve el golpe a Harry y si Harry sobrevive el turno pasa a otro personaje, por ejemplo ahora le tocaría atacar a Hermione contra el siguiente enemigo y así.

Esto se repite como si fuera un duelo por turnos entre un personaje de un equipo y otro del equipo contrario cambiando de atacante cada turno hasta que uno de los ejércitos se queda sin personajes vivos y en ese momento se declara al equipo ganador o se indica que todos murieron

Además también se revisa al final si algún personaje sobreviviente tiene menos de 30 puntos de salud y en ese caso se imprime un mensaje diciendo que necesita ayuda urgente. Y para finalizar, una vez que todo ha terminado se libera la memoria de todos los personajes que siguen vivos, para asegurarse de que no queden cosas ocupando espacio innecesariamente.

Conclusión

La verdad es que este proyecto me gustó muchísimo, desde que supe que podía elegir la temática mi cabeza fue directa al universo de Harry Potter porque es una saga que me encanta desde hace añisimos. Me pareció una forma de auto motivarme al combinar algo que disfruto muchísimo con algo que estoy aprendiendo y la verdad me parecía tan difícil y confuso, y eso hizo que todo el proceso fuera mucho más entretenido y llevadero. Me emocionaba ver cómo los personajes tomaban forma, cómo podía hacer que se atacaran entre sí, cómo podía mostrar los resultados en pantalla que fue muy difícil organizar mis ideas en esa parte pero de igual manera fue como estar creando una mini historia.

Pero aunque lo disfruté mucho también hubo momentos muy complicados, hubo días en los que no tenía idea de qué hacer, me sentía perdidísima con errores que no entendía por qué aparecían o con cosas que se rompían sin saber cómo arreglarlas. A veces modificaba una sola línea y de repente todo dejaba de funcionar y eso era muy frustrante. Me tomó bastante tiempo buscar soluciones, probar una y otra vez, cambiar partes del código, leer ejemplos, revisar los archivos, ajustar detalles y ver que seguía sin salir bien, una de las cosas que más me ayudó fue poder apoyarme en herramientas como la inteligencia artificial. Gracias a eso encontré algunas explicaciones más claras y hasta formas más sencillas de hacer ciertas cosas que al principio había hecho de manera más complicada.

Lo que más me costó fue hacer que toda la simulación funcionara bien en consola. Hacer que los personajes atacaran por turnos, que se imprimieran los mensajes correctamente, que las bajas se guardaran, que todo el flujo tuviera sentido, pero al final valió la pena porque me siento muy satisfecha con el resultado, de haberlo terminado, de haber aprendido tanto en el proceso, de haberlo hecho con un tema que me apasiona y de haber visto cómo algo que parecía imposible al principio poco a poco se fue armando hasta volverse un proyecto al que ahora le tengo cariño.

Referencias

GeeksforGeeks. (2025, 15 mayo). *File Handling through C++ Classes*. GeeksforGeeks. https://www.geeksforgeeks.org/cpp/file-handling-c-classes/

Alex. (2024, 25 noviembre). *25.1 — Pointers and references to the base class of derived objects*. https://www.learncpp.com/cpp-tutorial/pointers-and-references-to-the-base-class-of-derived-objects/

Cplusplus. (s.f.). type_info. https://cplusplus.com/reference/typeinfo/type_info/

GeeksforGeeks. (2025b, julio 23). *Exception Handling in C++*. GeeksforGeeks. https://www.geeksforgeeks.org/cpp/exception-handling-c/

OpenAI. (2025). *ChatGPT* (Versión GPT-4o) [Modelo de lenguaje AI]. https://chat.openai.com

Anthropic. (2025). *Claude AI* (Versión Claude 3) [Modelo de lenguaje AI]. https://claude.ai