Polimorfismo y Abstracción

Kenneth Rodriguez

Resumen

Este proyecto busca ejemplificar el uso de los conceptos de Polimorfismo y Abstracción en la Programación Orientada a Objetos. Para lograr este objetivo, desarrollé un RPG con interfaz de línea de comandos que aplica estos conceptos.

Alcance

Debido a que el objetivo del proyecto es entender los conceptos de POO, utilizar gráficos para el programa es algo más allá del alcance del proyecto. Toda comunicación con el usuario se realiza a través de la consola.

Guía del Usuario

Al iniciar el programa, el usuario (o también “jugador”) controlará las acciones de un grupo de aventureros clásicos de RPG: Guerreros, Magos, Pícaros y Clérigos. Con ayuda de este grupo, se enfrentará a monstruos enemigos.

Al iniciar el combate, la aplicación determinará el orden de iniciativa. Es decir, el orden en que los héroes y monstruos tomarán sus turnos.

Al iniciar su turno, el jugador decidirá qué acción tomará cada miembro de su grupo. Estás acciones pueden ser: ATACAR, DEFENDER o SANAR (en caso del Clérigo).

Cuando el jugador determine las acciones que tomará cada miembro de su grupo, se ejecutarán estas acciones, y la del enemigo, en el orden de iniciativa.

Este proceso se repetirá hasta que el enemigo sea derrotado, o todo el grupo del jugador sea derrotado.

Arquitectura del Sistema

La clase principal de este programa es la clase abstracta **Personaje**, de la que heredarán el resto de las clases. Esta clase cuenta con atributos clásicos de un personaje de RPG como:

- Poder (Ataque)
- Defensa
- Agilidad
- Magia

Estos atributos servirán para describir las capacidades y habilidades de cada personaje, ya sea controlado por el jugador o por el programa.

Esta clase cuenta con métodos como atacar, sanar, y ejecutar, pero el método actuar() es un método abstracto cuya implementación depende de si el personaje será controlado por el jugador o no.

De la clase Personaje heredan otras 2 clases abstractas, la clase **Aventurero** y la clase **Monstruo**. La clase Aventurero representa a los personajes controlados por el jugador, y su método actuar() refleja esto, ya que su implementación requiere de una entrada del usuario para seleccionar la acción a realizar. Por otro lado, el mismo método actuar() en la implementación de la clase Monstruo no requiere de una decisión del jugador, sino que determina la acción a realizar utilizando valores aleatorios que actúan como un dado.

De la clase Aventurero heredan 4 clases jugables: **Guerrero**, **Pícaro**, **Mago** y **Clérigo**. Cada una de estas 4 clases tiene sus propias ventajas y desventajas distribuidas en sus estadísticas principales. Por ejemplo, el Guerrero tiene alto ataque, el Pícaro tiene alta agilidad, el Mago tiene alta magia, y el Clérigo es el único que puede sanar aliados.

De la clase Monstruo, heredan 2 clases: **Goblin** y **Orco**. Similar a los hijos de la clase Aventurero, estas 2 clases tienen sus propias fortalezas y debilidades. En la implementación actual del programa solo se utiliza la clase Goblin.

En método main(), el juego inicia con los 4 héroes disponibles para el jugador y se enfrentan a un sólo goblin. Al ser un RPG por turnos, el programa toma la forma de un simple ciclo while(), donde el combate continúa hasta cumplir 1 de 2 condiciones: que el enemigo sea derrotado o que todos los héroes del jugador sean derrotados.

Conclusión

Considero que este proyecto fue un buen ejercicio para aplicar los conceptos de la programación orientada a objetos aprendidos hasta el momento. Este paradigma es uno muy utilizado en el mundo de los videojuegos ya que la representación de datos como clases e instancias es una analogía bastante directa y que tiene sentido.

Apliqué la Abstracción al crear clases que no pueden ser implementadas de manera directa, como las clases Personaje, Aventurero y Monstruo. Estas clases sirven como molde para generar el resto de las instancias que se utiliza el juego, mas no deben ser instanciadas directamente.

Por otro lado, el Polimorfismo se presenta principalmente en la diferencia de comportamiento entre las clases Aventurero y Monstruo. Cuando objetos de estas clases realizan una acción, las instancias de Monstruo realizan el comportamiento de forma automática, mientras que las instancias de Aventurero requieren de interacción con el usuario para elegir una acción.