

TRABAJO FINAL: PARADIGMAS DE PROGRAMACION

GRUPO PI



10 de noviembre de 2024

INGENIERIA EN INFORMATICA

RODRIGUEZ Emanuel, RIVERA Víctor, RODRIGUEZ Elías y VILLA Gabriel

Tabla de Contenidos

[Introducción 1](#_Toc182337267)

[Desarrollo 1](#_Toc182337268)

[Personajes 1](#_Toc182337269)

[Hechizos 2](#_Toc182337270)

[Integración con Prolog 3](#_Toc182337271)

[Testing 3](#_Toc182337272)

[Simulación 4](#_Toc182337273)

[Conclusión 4](#_Toc182337274)

# Introducción

Este proyecto tiene como objetivo desarrollar un sistema de combate estratégico basado en personajes, inspirado en un mundo de magia y enfrentamientos entre magos y mortífagos. A lo largo del desarrollo, nos enfocamos en aplicar principios de programación orientada a objetos y patrones de diseño que permitieran estructurar el sistema de manera modular y flexible. Utilizamos Java para la implementación del núcleo del juego, y Prolog como una herramienta para la toma de decisiones automatizada de los personajes, aportando inteligencia al comportamiento de los personajes en combate.

El sistema presenta una jerarquía de personajes con diferentes habilidades y hechizos, así como un diseño que integra hechizos específicos de ataque y defensa. Esta estructura, junto con la incorporación de patrones de diseño como Composite y Strategy, asegura que cada personaje y batallón pueda funcionar de forma independiente o en conjunto, promoviendo la reutilización de código y la flexibilidad para realizar futuras extensiones. La simulación del combate y las pruebas unitarias adicionales nos permitieron validar el funcionamiento del sistema en condiciones variadas, proporcionando una experiencia de juego equilibrada y dinámica.

# Desarrollo

## Personajes

Para modelar los personajes en el sistema, partimos de una clase abstracta llamada Personaje, que centraliza los atributos y métodos esenciales compartidos por todos los personajes jugables, tanto Magos como Mortífagos. Esta clase abstracta Personaje actúa como una base común de la cual heredan dos clases abstractas específicas: Mago y Mortifago. Estas clases intermedias abstractas nos permiten añadir características específicas y estructurales de cada tipo, manteniendo una jerarquía lógica y facilitando la extensibilidad del sistema. A su vez, de Mago y Mortifago heredan subclases concretas que representan roles específicos, como Auror, Profesor, y Estudiante para los Magos, y Comandante y Seguidor para los Mortífagos.

Consideramos importante que cada tipo de Mago y Mortífago tenga estadísticas y hechizos predefinidos, dado que esta configuración ayuda a reflejar mejor las diferencias de poder y habilidades entre los personajes. Por ejemplo, un Seguidor no debería poseer mayores estadísticas que un Comandante, y esta estructura jerárquica permite definir restricciones claras y precisas. Este enfoque también simplifica la implementación, ya que no es necesario personalizar o gestionar cada personaje individualmente.

Para el manejo de los hechizos y efectos aplicados a los personajes, optamos por utilizar una estructura de datos Map, lo cual permite un acceso directo a los elementos mediante las claves (keys). Esta elección hace que sea eficiente y rápido encontrar y modificar los hechizos o efectos específicos de cada personaje, lo que contribuye a un manejo dinámico y flexible durante el juego.

En cuanto a la clase Batallón, aunque podría haber sido modelada como una subclase de Personaje dado que comparte ciertos métodos y comportamiento, decidimos no hacerlo, ya que Batallón tiene atributos adicionales que no son comunes a la clase Personaje. En su lugar, implementamos un patrón de diseño Composite para Batallón, donde esta clase está compuesta por una colección de objetos de tipo Personaje. Esto nos permite formar agrupaciones de personajes de manera coherente y escalable. Sin embargo, para asegurar que los batallones estén compuestos exclusivamente de Magos o Mortífagos (pero no de ambos), empleamos la clase PersonajeFactory como intermediaria para la creación de los batallones. La PersonajeFactory se encarga de instanciar batallones homogéneos, restringiendo el tipo de personajes que puede incluir cada batallón.

Este enfoque jerárquico también facilita la organización y equilibrio del juego, ya que cada rol (Auror, Profesor, Estudiante para Magos; Comandante y Seguidor para Mortífagos) tiene un lugar claro y una función establecida dentro de la jerarquía de poder:

* **Magos**: Auror – Profesor – Estudiante.
* **Mortífagos**: Comandante – Seguidor.

El uso de esta jerarquía asegura que los personajes y sus habilidades están alineados con el sentido del juego, donde el orden refleja sus capacidades de menos a más poderosas.

## Hechizos

Para los hechizos, implementamos una clase abstracta HechizoStrategy que define un método principal ejecutar, el cual actúa como interfaz para ejecutar el comportamiento de cada hechizo. Esta estructura permite que cada subclase de HechizoStrategy implemente su propia lógica de acuerdo con el tipo de hechizo, ya sea de ataque o de defensa. Algunos hechizos pueden enfocarse en infligir daño y aplicar efectos específicos sobre el objetivo, mientras que otros pueden utilizarse para curar al lanzador o protegerlo de ataques futuros.

Cada hechizo viene con estadísticas predefinidas, tales como el costo de energía para lanzarlo, su duración (en caso de aplicar efectos continuos), el valor de daño o curación, y cualquier efecto adicional. Estos parámetros se establecen de antemano y varían según el tipo de hechizo, lo que permite una distinción clara entre hechizos de Magos y Mortífagos. De esta manera, cada personaje tiene acceso a un conjunto de hechizos adecuado a su tipo, manteniendo una coherencia temática y mecánica en el juego.

Este enfoque no solo facilita la flexibilidad en la implementación de nuevos hechizos, sino que también hace que el sistema sea extensible, permitiendo la adición de nuevos efectos o variaciones en los hechizos sin necesidad de modificar las clases principales. Además, la separación de los hechizos en subclases específicas permite personalizar de forma sencilla el comportamiento único de cada hechizo, como el tipo de efecto aplicado, la magnitud de daño o curación.

## Integración con Prolog

Para la integración con Prolog, desarrollamos un archivo .pl que contiene reglas y hechos predeterminados que sirven como base de conocimiento para el juego. Desde Java, nos encargamos de consultar y cargar dinámicamente hechos relacionados con personajes, sus hechizos, y el estado de personajes debilitados en el contexto del juego.

La elección de utilizar Prolog se basa en la necesidad de automatizar y optimizar las decisiones que cada personaje toma en función del contexto del juego. Prolog permite establecer una lógica de decisión de manera declarativa, facilitando que cada personaje pueda elegir entre atacar o defenderse según la situación. Los métodos principales de esta integración incluyen la toma de decisiones de cada personaje (decidir si ataca o se defiende), la elección de un hechizo adecuado para atacar o defenderse, y la selección del objetivo de ataque. Para la elección de objetivos, Prolog aplica una lógica de selección aleatoria entre los enemigos debilitados, considerando aquellos cuya vida está por debajo de un umbral específico (por ejemplo, la mitad de sus puntos de vida).

Desde Java, hemos implementado funcionalidades que permiten la carga inicial de hechos en Prolog, como los personajes y sus hechizos, así como la actualización continua del estado de los personajes. Esto incluye la posibilidad de eliminar hechos antiguos y actualizar el estado de “debilitado” de cada personaje según los eventos en el juego. Las consultas a Prolog incluyen la evaluación de las reglas de decisión descritas anteriormente, lo que permite un flujo de decisiones adaptativo y eficiente para cada personaje en cada turno.

## Testing

Para el testing, dividimos las pruebas en tres casos de prueba con JUnit: uno para los personajes, otro para la integración con Prolog, y un último para los hechizos. En cada conjunto de pruebas, realizamos pruebas unitarias de las funcionalidades de todas las clases involucradas, asegurando que cada componente del sistema funcione como se espera de manera independiente y conjunta.

1. **Pruebas de personajes:** En este conjunto de pruebas, nos enfocamos en evaluar la clase base Personaje, ya que esta engloba las funcionalidades principales compartidas por todas las subclases de magos y mortífagos. Verificamos que la clase maneje correctamente sus atributos, métodos, y efectos, dado que las diferencias entre subclases se limitan al tipo y a sus estadísticas iniciales. Además, realizamos pruebas específicas sobre la clase batallón, asegurándonos de que la lógica de composición funcione adecuadamente y permita gestionar personajes como un grupo.
2. **Pruebas de Prolog**: Estas pruebas se centran en la integración entre Java y Prolog, específicamente en la creación, actualización y consulta de hechos en la base de conocimiento. Verificamos que Java pueda cargar y consultar correctamente los datos en Prolog, y que las reglas implementadas tomen decisiones precisas en función de diferentes contextos del juego. Además, simulamos diversos escenarios de personajes en diferentes estados de vida para probar la toma de decisiones automática (como atacar o defenderse) y la selección de hechizos, asegurándonos de que las reglas respondan de acuerdo con lo esperado.
3. **Pruebas de hechizos**: En esta sección, evaluamos el comportamiento de cada tipo de hechizo, tanto de ataque como de defensa. Probamos que los hechizos apliquen correctamente los efectos y el daño especificado, así como los costos y limitaciones definidas en las clases de hechizo. Realizamos pruebas tanto en hechizos de curación como en los que aplican efectos especiales, asegurándonos de que las estadísticas y condiciones de los hechizos cumplan con los requisitos definidos para cada tipo de personaje (magos y mortífagos).

En conjunto, este enfoque de pruebas unitarias con JUnit permite evaluar cada módulo del sistema de forma exhaustiva y asegura que el sistema en su totalidad funcione correctamente en diferentes escenarios del juego, validando la coherencia y robustez de la lógica implementada.

## Simulación

Para la prueba del juego, creamos una simulación de una batalla a muerte en el método main, enfrentando un batallón de mortífagos contra otro de magos. Para introducir variabilidad en los resultados, incorporamos aleatoriedad en el orden de los turnos de ataque y establecimos que la batalla finalice cuando uno de los batallones se quede sin personajes, coronando como ganador al equipo que aún conserve unidades con vida.

El objetivo principal de esta prueba es evaluar la lógica de combate y la integración de la toma de decisiones automática mediante Prolog. Esto incluye acciones como la recuperación de energía y la aplicación de efectos al final de cada ronda, reflejando el flujo de una batalla auténtica. Nos aseguramos de equilibrar los batallones incluyendo distintos tipos de personajes en cada equipo, lo que contribuye a que los resultados finales de cada simulación varíen de manera natural; a veces ganan los magos, y otras veces los mortífagos.

# Conclusión

En conclusión, este trabajo práctico nos permitió implementar y profundizar en conceptos avanzados de programación orientada a objetos, como herencia, polimorfismo, y patrones de diseño, particularmente el patrón Composite y Strategy. Mediante la estructuración de clases abstractas y concretas para personajes, hechizos y batallones, logramos un diseño modular y escalable que facilita la incorporación de nuevos personajes o hechizos en el futuro.

La integración de Prolog aportó una capa adicional de automatización en la toma de decisiones de los personajes, permitiendo simular batallas estratégicas en las que cada unidad toma decisiones en función del contexto de la batalla. Esta conexión entre Java y Prolog no solo contribuyó a la complejidad del proyecto, sino que también demostró el potencial de combinar paradigmas de programación para resolver problemas complejos.

La simulación de las batallas y el uso de pruebas unitarias nos permitió validar la solidez del sistema, identificar y corregir errores, y asegurar que la lógica del juego funcione como se espera en diversos escenarios. En conjunto, este proyecto fue una experiencia enriquecedora que nos brindó la oportunidad de aplicar teoría y prácticas de programación en el desarrollo de un sistema de combate interactivo y adaptable.