Lautaro García Conejero

Informe TP1

¿Como compilar el código?

El código tiene un archivo makefile. Es decir, escribiendo en la terminal “make”, correrá un archivo main donde dejara elegir cual ejercicio ejecutar.

Ejercicio 1: Creación de personajes y armas

Personajes

Los personajes se dividen en dos grupos principales: **Guerreros** y **Magos**. Cada grupo tiene características y habilidades únicas que los hacen destacar en el combate. La **estructura de clases** está diseñada de manera modular y organizada, lo que facilita la expansión y personalización del sistema. A continuación, se describe de manera general la estructura del sistema de personajes.

Clases Base y Abstractas

Para mantener una estructura ordenada y fácil de mantener, se separaron las clases de personajes en diferentes archivos y carpetas. La **clase personaje** actúa como una **interfaz**, que contiene los métodos virtuales básicos que todos los personajes deben implementar, como **getters** para acceder a atributos comunes (vida, daño, armadura, etc.), el manejo de **armas** y las formas básicas de ataque (por ejemplo, ataque_rapido y atacar_con_arma), que son comunes a todos los personajes.

Métodos Comunes

El objetivo de esta estructura es permitir que todas las clases derivadas (tanto Guerreros como Magos) hereden estos métodos comunes sin tener que implementarlos de nuevo en cada clase. Además de los **getters básicos**, la clase personaje incluye funciones genéricas que manejan las armas del personaje y las acciones más simples de combate, como los métodos mencionados anteriormente.

Clases Derivadas y Especialización

Cada grupo de personajes (Guerreros y Magos) tiene su propia clase base abstracta. Estas clases abstractas sirven para definir los **atributos básicos** que serán comunes a todos los personajes de ese tipo, como, por ejemplo:

- **Vida**
- **Daño**
- **Armadura**

- **Resistencia mágica**
- **Estamina**

Cada clase derivada, como **Bárbaro**, **Paladín**, **Hechicero**, o **Brujo**, tiene sus propios métodos y atributos únicos, basados en estas características comunes. Esta estructura

Mecánicas Extras

Además de las habilidades individuales de cada personaje, existe un conjunto de **mecánicas globales** que afectan a todos los personajes en el juego. Estas mecánicas están principalmente basadas en los **efectos** que un personaje puede aplicar a sus enemigos. Estos efectos alteran las condiciones de la batalla y pueden ser tanto **negativos** como **positivos**.

Efectos

En el código, estos efectos están gestionados por un **enum llamado EFFECTO**, que contiene todos los posibles efectos que un jugador puede recibir durante el combate. Existen 9 efectos únicos, de los cuales **7 son negativos** y **2 son positivos**. Algunos ejemplos de estos efectos incluyen:

- **PARALIZAR**: El objetivo no puede atacar durante esa ronda.
- **QUEMADURA**: El objetivo recibe daño adicional durante las rondas y su capacidad de hacer daño se ve reducida.
- **PROTECCIÓN**: El daño que el personaje recibe es reducido entre un 20% y un 100%, dependiendo de un valor aleatorio calculado al inicio del ataque.

Los **efectos negativos** afectan negativamente al personaje, debilitando sus capacidades de ataque o defensa, mientras que los **efectos positivos** otorgan ventajas adicionales, como **proteger** al personaje o aumentar su poder de ataque.

Aplicación de los Efectos

Los efectos pueden ser aplicados de dos formas principales:

1. **Por ataques de armas**: Algunas armas tienen la posibilidad de aplicar efectos al momento de impactar a un enemigo. Por ejemplo, un **bastón mágico** puede aplicar un **efecto de veneno** en cada golpe.
2. **Por habilidades específicas**: Algunos personajes tienen habilidades únicas que les permiten aplicar efectos. Por ejemplo, el **Hechicero** tiene la capacidad de aplicar el efecto de **confusión**, mientras que el **Mercenario** puede **marcar** a un enemigo para que, tras varias rondas, se active un ataque especial.

Estructura y Gestión de Efectos

Los efectos son guardados en un vector dentro de la clase personaje, y cada uno de estos efectos se gestiona mediante un **struct** llamado **EfectoActivo**. Este struct contiene el **tipo de efecto** y la **duración restante**.

En cada **ronda de combate**, los efectos de todos los personajes son procesados mediante la función **procesar_efectos()**, que se asegura de aplicar y actualizar cada uno de los efectos activos, además de reducir su duración en cada ronda.

Tipos de Efectos

- **Negativos**:
    - **Parálisis**: El personaje no puede atacar en esa ronda.
    - **Confusión**: Existe una probabilidad de que el personaje se ataque a sí mismo.
    - **Miedo**: Aumenta el daño recibido por el personaje.
    - **Quemadura**: Causa daño continuo y reduce la capacidad de hacer daño.
    - **Veneno**: Causa daño mágico continuo durante varias rondas.
    - **Hemorragia**: Causa daño verdadero, un porcentaje de la vida máxima del objetivo.
- **Positivos**:
    - **Protección**: Reduce el daño recibido en un porcentaje aleatorio entre 20% y 100%.
    - **Potenciación**: Aumenta el daño de un aliado o del propio personaje durante un turno.

Marcas

El **Mercenario** cuenta con una mecánica especial de **Marcas** que actúa de manera similar a los efectos, pero con un enfoque en los enemigos que el mercenario **marca** para que, al pasar varias rondas, se desencadenen efectos especiales.

- Por ejemplo, **el Sicario** es una marca que, si pasan 2 rondas y el enemigo marcado tiene menos de 15% de vida, lo elimina automáticamente.
- Las marcas se procesan de la misma manera que los efectos, utilizando un sistema de **probabilidad** que depende del **oro** robado por el mercenario.

Invocaciones

El **Conjurador** tiene la habilidad de **invocar animales** que lo asisten en la batalla. Cada invocación tiene sus propias estadísticas, incluyendo vida, daño y efectos que pueden ser aplicados.

- Las **invocaciones** tienen una probabilidad de aplicar efectos dependiendo del tipo de animal invocado.
- **Los animales invocados** siempre **reciben el daño antes que el invocador**. Esto significa que las invocaciones actúan como **escudos** temporales.
- El conjurador también tiene la posibilidad de invocar un **dragón**, que realiza un **ataque de área** aplicando un efecto de **quemadura** al objetivo.

Enums y Structs Importantes

Varios **enums** y **structs** se utilizan para organizar y gestionar las mecánicas de juego de manera eficiente:

- **EFFECTO**: Enum que define los efectos posibles que un personaje puede recibir.
- **TIPO_DAÑO**: Enum que se utiliza para distinguir entre daño físico y daño mágico. Esto es importante porque las **armaduras** y **resistencias** funcionan de manera diferente según el tipo de daño recibido.
- **Marca**: Es una mecánica especial del **Mercenario**, que permite marcar a los enemigos para aplicar efectos o eliminar enemigos cuando cumplen ciertos requisitos, como la vida baja.
- **InvocacionAnimal**: Un struct que guarda las estadísticas de las invocaciones de animales por parte del **Conjurador**. Este struct contiene la vida, el daño, el tipo de invocación y la probabilidad de aplicar efectos.
- **INVOCACIONES_ANIMALES**: Enum que define los tipos de animales que pueden ser invocados y las características específicas de cada uno.
- **MATERIAL** y **ENCANTAMIENTO**: Estas son estructuras utilizadas en la clase **Arma**. Los **materiales** afectan al daño de las armas de combate, mientras que los **encantamientos** afectan a las armas mágicas. Ambos atributos son asignados aleatoriamente durante la creación del arma.

Armas

aLa clase **Arma** actúa como la base de todas las armas en el juego, y se presenta como una especie de "mini versión" de la clase **personaje**, ya que comparte algunas características similares. Mientras que los personajes tienen habilidades, estadísticas y atributos únicos, las armas también tienen sus propios atributos que definen su efectividad en combate. A continuación, se detalla la estructura general de las armas y sus diferencias.

Lautaro García Conejero

Interfaz y Clases Abstractas

La clase **arma** es una **interfaz** que contiene los métodos esenciales para obtener los **atributos básicos** de las armas, como el **nombre**, **daño físico**, **daño mágico**, y **durabilidad**. Además, tiene un método principal: **atacar()**, que es utilizado cuando un personaje desea usar un arma para atacar.

Las **clases abstractas** derivadas de Arma permiten definir las variables básicas comunes que todas las armas comparten, tales como el **daño físico** y **magia**.

Clases Derivadas

Cada tipo de arma tiene **dos ataques únicos**, con diferentes comportamientos en cuanto a **gasto de durabilidad** y **daño**. Además, algunas armas tienen atributos especiales, que las hacen destacar en el combate:

- **Lanza**: La lanza tiene la capacidad de **partirse a la mitad** para realizar un ataque más potente, pero con una durabilidad reducida.
- **Espada**: Si la espada llega a un umbral de **20 de durabilidad**, puede aplicar un **efecto de hemorragia**, que causa daño adicional durante varias rondas.
- **Poción**: Las pociones tienen la posibilidad de **explotar**, causando un gran daño, pero también rompiendo el arma en el proceso. Este tipo de arma tiene un alto riesgo, pero un alto potencial de recompensa.

Tipos de Armas en el Juego

1. **Armas de Combate**:
    - Hacha
    - Hacha doble
    - Espadas
    - Garrotes
    - Lanzas
2. **Armas Mágicas**:
    - Bastones
    - Libros de Hechizos
    - pociones
    - Amuletos

Cada una de estas armas tiene características únicas que las hacen adecuadas para diferentes tipos de combate. Algunas se centran en un daño rápido y directo, mientras que otras infligen **efectos mágicos** que pueden cambiar el curso de la batalla.

Equipos

La clase **Equipo** es una adición crucial en este sistema, ya que permite agrupar a varios personajes en un conjunto. Esta clase facilita tanto la funcionalidad de ciertos personajes, como la gestión de equipos durante las batallas en los **ejercicios 2 y 3**. Un equipo puede estar compuesto por varios personajes, y se encarga de gestionar su vida y estatus durante la partida.

Atributos del Equipo

La clase **Equipo** contiene varios atributos fundamentales para la gestión de los personajes:

1. **Nombre del equipo**: Un identificador único para cada conjunto de personajes.
2. **Tamaño máximo**: El número máximo de personajes que puede contener un equipo.
3. **Vector de personajes**: Un **vector** que almacena todos los personajes del equipo. Cada personaje es guardado como un puntero a personaje para facilitar la creación dinámica y la manipulación del equipo.
4. **Vector de jugadores vivos**: Una copia del vector de personajes, pero solo con los personajes que están vivos en el combate. Esto se utiliza para verificar cuántos personajes están disponibles para combatir.

Métodos del Equipo

Los métodos de la clase **Equipo** permiten realizar varias operaciones básicas sobre los personajes que contiene:

- **agregar_personaje()**: Añade un nuevo personaje al equipo. Si el equipo no ha alcanzado su tamaño máximo, el personaje se agrega al vector de personajes.
- **eliminar_muertos()**: Elimina a los personajes que están muertos del vector de personajes vivos, lo que facilita el seguimiento de la salud del equipo durante las batallas.
- **verificar_vivos()**: Devuelve la cantidad de personajes vivos en el equipo, lo que permite controlar la dinámica de combate.
- **devolver_vivos()**: Retorna el vector de personajes vivos para su uso en otras funciones, como la asignación de turnos o el cálculo de daño.

Lautaro García Conejero

Descripción General de los Métodos de los Personajes

A continuación, se presentan las habilidades y mecánicas principales de cada uno de los personajes. Estas habilidades son las más complejas del código y se definen de manera única para cada clase. Las mecánicas varían según el tipo de personaje, ya sea guerrero o mago, y otorgan características especiales que pueden influir decisivamente en el combate.

Bárbaro

El **Bárbaro** tiene una mecánica centrada en la **rabia**, la cual le permite aumentar su daño conforme aumenta su furia. La rabia no solo aumenta el daño, sino que también habilita dos habilidades adicionales:

- **Romper Huesos**: Realiza un ataque que **ignora la armadura** del enemigo, causando un daño más directo.
- **Miedo**: Aplica el efecto de **miedo** a los enemigos, lo que aumenta el daño que reciben.

Además, el **Bárbaro** tiene una **habilidad definitiva** que le otorga **inmunidad a la muerte durante 3 rondas** si se encuentra en estado de rabia, dándole una ventaja crucial en situaciones de vida o muerte.

Caballero

El **Caballero** tiene una **habilidad única** centrada en la **protección**:

- **Proteger a los Compañeros**: Utiliza un **escudo** que reduce o anula completamente el daño recibido por los compañeros cercanos. Esto lo convierte en un pilar defensivo para cualquier equipo.

El **Caballero** no tiene habilidades adicionales como los demás personajes, pero su capacidad para **proteger** es clave en las batallas en equipo.

Gladiador

El **Gladiador** tiene una mecánica basada en la **adrenalina**, que se obtiene infligiendo daño al enemigo. Esta adrenalina le otorga las siguientes ventajas:

- **Adrenalina**: Aumenta tanto su **armadura** como su **capacidad para esquivar** ataques.

- **Parálisis**: Al gastar adrenalina, el **Gladiador** puede aplicar el efecto de **Parálisis** al enemigo, inmovilizándolo durante un turno.

El **Gladiador** también tiene la habilidad de **esquivar** ataques cuando decide **"entretener al público"**, lo que le da **honor**. Cada golpe esquivado aumenta su honor, que a su vez **incrementa su daño**. Sin embargo, si el **Gladiador** es golpeado, pierde todo su honor y adrenalina. Esta mecánica de **esquivar** está directamente relacionada con el **honor**, lo que le otorga **más daño** y la habilidad de **ignorar efectos negativos**.

Mercenario

El **Mercenario** tiene una habilidad centrada en **robar oro** durante las batallas, lo que le otorga varias ventajas:

- **Robo de Oro**: El **oro** ganado le otorga la capacidad de realizar **ataques críticos**.
- **Contratar Sicarios**: Con el oro acumulado, el **Mercenario** puede contratar a un **sicario** que, después de dos rondas, **ejecutará** a un enemigo marcado que tenga menos del 15% de vida.

Además, el **Mercenario** tiene una **habilidad definitiva** que le permite **atacar con un cañón** a un enemigo marcado, causando un **gran daño**.

El **Mercenario** también tiene la capacidad de **robar armas** a los enemigos y conservarlas, o incluso **eliminar a un compañero** para **ganar estadísticas** adicionales, como más oro, vida y daño. Sin embargo, si **no hay compañeros vivos**, el **Mercenario** muere.

Paladín

El **Paladín** se centra en la **protección** y la **fe**:

- **Escudo**: Utiliza un **escudo** para bloquear ataques, protegiendo a sus aliados. Mientras el escudo está activo, el daño que recibe se reduce.
- **Fe**: El **Paladín** gana **fe** con cada ataque realizado. La **fe** le otorga más **estamina** y **vida**.

El **Paladín** puede **gastar fe** para **bendecir sus armas**, aumentando el daño de las mismas, o para realizar un ataque especial que **ignora la armadura** del enemigo.

Brujo

El **Brujo** tiene varias habilidades basadas en la manipulación de la magia oscura:

- **Maldecir Enemigos**: El **Brujo** puede **maldecir** a los enemigos, aplicando un **efecto negativo aleatorio** de los efectos definidos en el sistema de efectos.
- **Control Mental**: El **Brujo** puede **tomar el control de un enemigo**, obligándolo a atacar a su propio equipo.
- **Frenesí**: Si el **Brujo** es derrotado, entra en un **estado de frenesí**, donde puede seguir luchando incluso después de perder toda su vida. Si mata a un enemigo en su siguiente turno, **sobrevive**.
- **Pacto Demoníaco**: El **Brujo** puede **sacrificar vida y mana** para aumentar significativamente su **daño** durante 3 rondas.

Conjurador

El **Conjurador** se especializa en invocar criaturas para que luchen junto a él:

- **Invocar Animales**: El **Conjurador** puede invocar animales como **aliados**. Estos animales **reciben el daño en lugar del invocador**, lo que le permite al **Conjurador** mantenerse protegido.
- **Parálisis**: También puede **paralizar** a un enemigo, impidiéndole actuar durante una ronda.

El **Conjurador** puede invocar un **dragón**, que realiza un **ataque de área** a un enemigo y aplica un **efecto de quemadura**.

Hechicero

El **Hechicero** es un mago versátil que utiliza magia de control y evasión:

- **Confusión**: El **Hechicero** puede aplicar el efecto de **confusión**, que hace que el enemigo tenga la posibilidad de atacarse a sí mismo.
- **Escudo**: Puede **protegerse con un escudo** que bloquea ciertos ataques, ya sean físicos o mágicos.
- **Clonación**: El **Hechicero** tiene la habilidad de **clonarse**, lo que le permite evadir ataques y hacer que el enemigo se confunda, golpeando al clon en lugar de al Hechicero.
- **Rayo**: Lanza un **rayo** mágico que tiene la posibilidad de **hacer un daño crítico** al enemigo.

Lautaro García Conejero

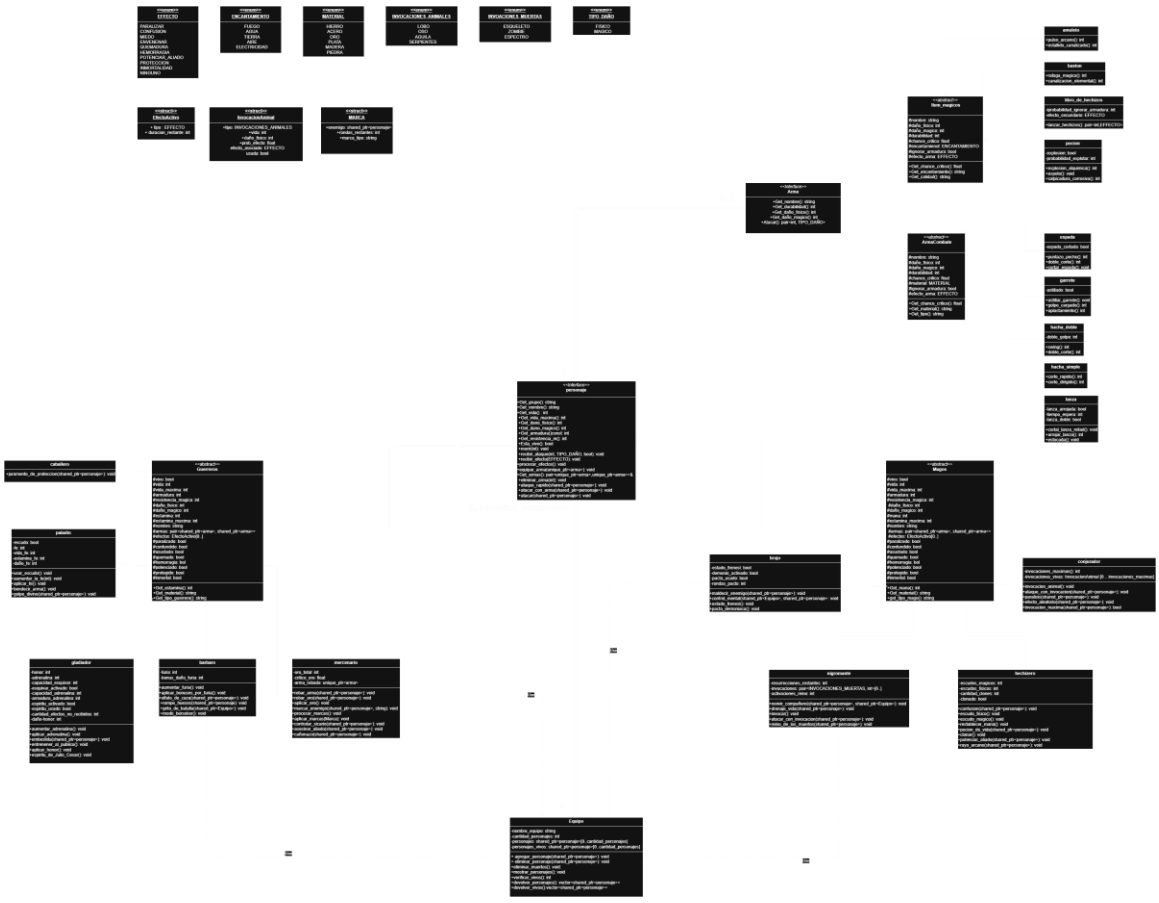
Nigromante

El **Nigromante** es un maestro de la muerte y la necromancia:

- **Revivir Compañeros**: El **Nigromante** puede **revivir a los compañeros muertos** con menos vida, dándoles una nueva oportunidad de luchar.
- **Drenaje de Vida**: El **Nigromante drena vida** de un enemigo, restaurando su propia salud.
- **Invocar Muertos**: El **Nigromante** puede invocar **muertos** para que luchen a su lado, actuando como **tanques** y **protegidos**.
- **Reino de los Muertos**: En esta habilidad definitiva, el **Nigromante** envía a un enemigo al **Reino de los Muertos**, donde le **roba un 20% de sus estadísticas**. Si el **Nigromante** sobrevive, se queda con esas estadísticas; si no, las pierde.

UML - Diagrama de clases

A continuación, se presenta el diagrama de clases que muestra la jerarquía entre personajes, armas y relaciones. Este diagrama es clave para entender la estructura del sistema y cómo las clases interactúan entre sí. El UML muestra la relación entre clases como `personaje` y sus subclases, así como las armas y sus diferentes tipos de efectos. El UML también se encuentra en la carpeta de su ejercicio correspondiente



Ejercicio 2: Fábrica de Personajes y Armas

En este ejercicio, se implementa una **fábrica de personajes** que facilita la creación de personajes y armas en el sistema. Esta fábrica está estructurada para ser flexible y permitir la creación tanto **aleatoria** como **manual** de los personajes y sus equipos. Las funciones dentro de la fábrica son **estáticas**, lo que significa que no es necesario crear una instancia de la clase para utilizarlas, simplificando el proceso de creación y mejorando la modularidad.

Estructura y Funcionalidad de la Fábrica

La fábrica de personajes y armas está compuesta por tres funciones principales:

1. **Crear Personajes**: Esta función permite crear personajes de forma aleatoria o manual, dependiendo de los parámetros proporcionados. Los personajes pueden ser de tipo **Guerrero** o **Mago**, y dentro de estos tipos, se pueden crear subtipos específicos como **Bárbaro**, **Hechicero**, **Brujo**, entre otros.

2. **Crear Armas**: Al igual que los personajes, las armas pueden ser creadas de manera aleatoria o manual. Las armas pueden ser de **combate** (como espadas, hachas,

garrotes) o **mágicas** (como bastones, libros de hechizos, amuletos). Cada arma tiene sus propios atributos, como **daño físico**, **daño mágico**, **chance de crítico** y **durabilidad**.

3. **Equipar Personajes**: Esta función se encarga de asignar las armas al personaje recién creado. El número de armas que se asignan se elige a través de un parámetro que se recibe desde la terminal, donde el jugador puede decidir cuántas armas quiere que tenga el personaje (de 0 a 2 armas).

Uso de Enums para Facilitar la Creación

El sistema de creación de personajes y armas se basa en **4 enums** que representan todas las clases y subclases derivadas disponibles. Esto permite que el código sea más claro y modular, ya que facilita la asignación de tipos de personajes y armas sin necesidad de utilizar cadenas de texto o valores numéricos arbitrarios. Los enums incluyen:

- **CLASES_PERSONAJES**: Enum que representa los tipos de personajes disponibles (Guerrero, Mago, etc.).
- **TIPO_ARMA**: Enum que incluye los tipos de armas de combate y mágicas (espada, hacha, bastón, etc.).
- **EFECTOS**: Enum para representar los efectos que las armas y habilidades pueden aplicar a los personajes (quemadura, veneno, etc.).
- **MATERIALES y ENCANTAMIENTOS**: Enums que definen los materiales y encantamientos posibles para las armas, afectando sus estadísticas y propiedades especiales.

Creación Aleatoria y Manual de Personajes y Armas

Dependiendo de los parámetros de entrada, la creación de personajes y armas puede ser **aleatoria** o **manual**:

- **Creación Aleatoria**: El tamaño del equipo, la cantidad de armas por jugador, y las características de los personajes y armas son generados aleatoriamente usando **rand()**. Para garantizar que cada ejecución sea diferente, se utiliza una **semilla aleatoria** creada con la función srand(time(0)), lo que asegura que los valores generados sean distintos en cada ejecución.
- **Creación Manual**: A través de la terminal, el jugador puede seleccionar el tipo de personaje que quiere crear (por ejemplo, **Paladín**, **Brujo**, etc.), así como el número de armas que el personaje tendrá. Esto permite que el usuario tenga más control sobre los personajes y sus armas.

Creación de Equipos

La **función de creación de equipos** usa la clase **Equipo** previamente definida. Esta clase permite gestionar un grupo de personajes y facilita el manejo de los equipos dentro del juego. Con esta función, es posible:

- **Asignar personajes a un equipo**: La creación del equipo es completamente aleatoria, pero también permite la personalización, ya que el jugador puede elegir las características de los personajes antes de asignarlos al equipo.
- **Visualizar el equipo**: Los personajes que conforman el equipo son mostrados a través de la función mostrar_personajes() de la clase **Equipo**, que proporciona una representación clara de todos los miembros del equipo.

Conclusión

Este sistema de fábrica es flexible y robusto, permitiendo tanto la creación aleatoria como manual de personajes y armas. Utilizando **enums** y **rand()**, la creación se hace dinámica y variada, ofreciendo una experiencia diferente en cada ejecución. Además, la **clase Equipo** gestiona de manera eficiente la composición de equipos de personajes, lo que facilita tanto el proceso de creación como el de manejo durante el combate.

Ejercicio 3: Piedra, Papel o Tijera

En este ejercicio, se implementa un sistema de combate simple basado en el clásico juego de **Piedra, Papel o Tijera**, donde el jugador se enfrenta a la "máquina". Este sistema permite a los personajes realizar un ataque con tres posibles opciones: **Piedra**, **Papel** o **Tijera**, cada uno con sus propias reglas y contraposiciones.

Funcionalidad General del Juego

El juego se basa en decisiones tomadas tanto por el **jugador** como por la **máquina** en cada ronda. Para garantizar que cada ejecución sea distinta, se usa la función srand(time(0)), que genera una **semilla aleatoria** para las decisiones de ambos jugadores. Esto asegura que cada partida sea única.

El Proceso de Decisión

1. **Función Decision_juego()**: Esta es la función principal que permite al **jugador** decidir qué tipo de ataque utilizar en cada ronda. El jugador elige entre tres opciones:
    - **Piedra**
    - **Papel**
    - **Tijera**

La opción seleccionada es convertida en un valor del **enum** DECISION_JUEGO, que define las tres posibles jugadas del juego.

2. **Enum DECISION_JUEGO**: Este enum contiene los tres posibles movimientos del juego: **Piedra**, **Papel** y **Tijera**. Cada movimiento tiene una relación de **fortaleza** y **debilidad** con los otros, lo que establece las reglas del juego:
    - **Piedra** vence a **Tijera**.
    - **Tijera** vence a **Papel**.
    - **Papel** vence a **Piedra**.

Estos valores se utilizan para determinar los resultados del enfrentamiento.

Determinación del Ganador

3. **Función Ganador()**: Una vez que el jugador y la máquina han tomado su decisión, se llama a la función Ganador(). Esta función compara las jugadas de ambos y decide quién ha ganado según las reglas del juego. El ganador es determinado por las siguientes reglas:
    - **Piedra** gana a **Tijera**.
    - **Tijera** gana a **Papel**.
    - **Papel** gana a **Piedra**.

Si ambos jugadores eligen el mismo movimiento, se considera un **empate**.

Manejo de la Vida de los Jugadores

4. **Función de Resultado de la Ronda**: Una vez que se determina el ganador, se llama a una última función que **concluye** la ronda. Dependiendo de los resultados (quién ganó o si fue un empate), la función muestra por consola lo siguiente:
    - **Quién atacó y con qué**: Especifica qué jugada realizó cada jugador.
    - **Vida de los jugadores**: Se actualiza y muestra la vida restante de los jugadores después de cada ronda, lo que permite seguir el progreso del combate.

Si el jugador gana, **la máquina pierde vida**. Si la máquina gana, **el jugador pierde vida**. En caso de empate, no hay cambios en las vidas de los jugadores.

Detalles de Implementación

1. **Generación de Movimiento Aleatorio para la Máquina**: La máquina selecciona su movimiento de manera **aleatoria** utilizando el mismo sistema de generación

aleatoria con rand() y srand(time(0)), garantizando que cada ronda sea impredecible.

2. **Lógica de Comparación de Movimientos**: Una vez que ambos jugadores han decidido sus movimientos, la función Ganador() compara las jugadas y calcula quién es el vencedor, tomando en cuenta las reglas de interacción entre los tres movimientos posibles.