Pestaña 1

PERSONAJES (herencia):

- Identidad:
    - nombre: identificador mostrado en combate.
    - tipoClase: enum o texto (GUERRERO, MAGO, SACERDOTE).
- Vida:
    - vidaMax
    - vidaActual
- Recurso (según diseño):
    - recursoMax (mana/energía)
    - recursoActual
    - Si una clase no usa recurso, debe existir igualmente (por uniformidad) con valores 0 o una implementación que lo ignore.
- Estadísticas base:
    - ataqueBase: contribuye a daño físico o general.
    - poderMagico (opcional, pero recomendable si hay daño mágico).
    - defensaBase: reduce daño recibido.
    - precision / evasión (opcional).
- Equipamiento:
    - arma: referencia a Arma (obligatoria).
- Colecciones asociadas:
    - estadosActivos: lista de Estado (ArrayList).
    - hechizos: lista de Hechizo (ArrayList) o catálogo por clase.
    - cooldowns: diccionario/mapa si se implementa reutilización.
- Estado de combate:
    - vivo: no hace falta si se deriva de vida, pero se permite.
    - defendiendo (opcional).

Guerrero:

- **Geralt** (Personaje BUENO)
    - Armas: **Aerondight:** la mejor espada de plata, este arma mejora su daño con cada muerte.(es para monstruos)

      Espada de acero: para combatir contra los humanos
- **Imlerith** (Personaje MALO)
    -

Rol: daño físico sostenido + resistencia.
Rasgos esperados:

- vidaMax alta, defensaBase alta.
- recurso opcional (puede ser 0 y no usar hechizos, o energía simple).
- Acciones:
    - ataque con arma como acción principal.
    - opcional: habilidad de “Golpe poderoso” (daño más alto con coste o cooldown).
- Polimorfismo:
    - puede sobrescribir recibirDanio para aplicar reducción adicional si está

defendiendo.
o puede sobrescribir métodos de cálculo si se diferencian daño

físico/mágico.

Mago:

- **Yennefer** (Personaje BUENO)
- **Caranthir** (Personaje Malo)

Rol: daño mágico y aplicación de estados (quemadura).
Rasgos esperados:

• vidaMax baja/media, defensaBase baja.

• recursoMax alto (mana).

• Acciones:

- ataque básico (si se permite) más débil.
- hechizos ofensivos como núcleo: daño directo y daño en el tiempo. •

Polimorfismo:

- puede sobrescribir un método tipo calcularPoderHechizo() o similar. o sus hechizos aplican Quemadura con duración y potencia.

Sacerdote:

Ciri (Personaje BUENO)
Eredin (Personaje MALO)

Rol: soporte, curación directa y curación en el tiempo (Renovar).
Rasgos esperados:

• vidaMax media, defensaBase media/baja.

• recursoMax medio/alto (mana).

• Acciones:

- curación directa a un aliado.
- curación por turnos (Renovar).
- opcional: hechizo ofensivo ligero o disipación (extra).

• Polimorfismo:

- puede sobrescribir reglas de curación (por ejemplo, bonificación al curar). o puede tener prioridad de IA distinta (curar al aliado más débil).

Clase base Arma

Atributos mínimos

- nombre: texto descriptivo.
- tipo: enum o texto (CUERPO_A_CUERPO / A_DISTANCIA).
- danioBase: entero.
- modificador: opcional (por ejemplo, multiplicador o bonificación fija). •

probCritico y multiplicadorCritico: opcional, para introducir variabilidad. •
precision: opcional si se implementa fallo/esquiva.

Métodos mínimos

- int calcularDanio(Personaje atacante, Personaje defensor): método central. Debe usar stats del atacante (por ejemplo, ataqueBase) y considerar, si se desea, stats del defensor (defensa, evasión). Este método debe ser **polimórfico**: subclases lo pueden sobrescribir.
- String descripcion(): texto corto para mostrar en selección o resumen. •

Opcional: boolean golpea(...) si se separa precisión de daño.

Reglas de diseño

- El arma no debe modificar directamente vida. Solo calcula daño.
- La reducción por defensa puede implementarse en:
  - recibirDanio del defensor (recomendado, centraliza defensa), o
  - calcularDanio del arma (válido si se quiere que algunas armas ignoren defensa). Debe elegirse una estrategia y mantenerla consistente.

4.2 Subclases obligatorias

Se requieren, como mínimo, dos subclases:

a) ArmaCuerpoACuerpo

Concepto: armas de contacto (espada, hacha, maza).
Diferencias esperadas (ejemplos implementables)

- Daño más estable o más alto en base.
- Puede beneficiarse más del ataqueBase del personaje.
- Opcional: puede tener crítico mayor o multiplicador mayor.

• Opcional: penalización contra ciertos estados (no obligatorio).

Polimorfismo

• Sobrescribe calcularDanio(...) para aplicar su fórmula particular.

b) ArmaADistancia

Concepto: arco, ballesta, varita, rifle (según ambientación).

Lo que te faltaba en el Modelo de Personajes

Métodos mínimos (interfaz funcional) de la clase base Personaje

- `boolean estaVivo()`: devuelve si el personaje puede actuar.
- `void recibirDanio(int cantidad)`: aplica reducción por defensa si procede, actualiza vida y gestiona muerte.
- `void curar(int cantidad)`: suma vida sin superar vidaMax.
- `boolean gastarRecurso(int coste)`: comprueba y descuenta recurso si es posible.
- `void equiparArma(Arma arma)`: asigna el arma actual.
- `void aplicarEstado(Estado estado)`: añade un estado a la colección aplicando reglas de acumulación.
- `void procesarEstados()`: recorre estados activos, aplica su efecto por turno, reduce duración y elimina expirados.
- `int calcularDanioBasicoContra(Personaje objetivo)`: útil para ataques simples; si se delega todo en Arma, puede omitirse.
- `Accion elegirAccion(...)`: (opcional) en modo jugador pide entrada; en modo IA decide.

Reglas para estados (acumulación/renovación)

- Debe definirse una política clara, implementable con colecciones.
- Opción simple (recomendada): Si se aplica un estado con el mismo nombre, se renueva duración (se reemplaza o se actualiza). Si es distinto, se añade.
  +1
- La política elegida debe ser consistente y aplicarse dentro de aplicarEstado.

Lo que te faltaba en Armas

4.3 Catálogo de armas y selección

- Debe existir un conjunto de armas disponibles para el juego.
- Se recomienda gestionarlo con una colección y un diccionario: Lista/ArrayList de armas para

mostrar opciones al usuario. Diccionario/Map para acceder por nombre o por id (Map<String, Arma> o Map<Integer, Arma>).
+1

- Durante la creación de equipo, cada personaje debe terminar con un arma equipada.
- Puede ser por auto-asignación según clase o elegida de un catálogo por el usuario.

5) Hechizos

Clase base Hechizo

- **Atributos mínimos:**
  - `nombre`: identificador del hechizo.
  - `costeRecurso`: entero (mana/energía).
  - `tipoObjetivo`: enum (ENEMIGO_UNICO, ALIADO_UNICO, PROPIO, TODOS_ENEMIGOS, TODOS_ALIADOS).
    +2
  - `descripcion`: texto corto para mostrar en consola.
  - `cooldownMax`: entero (turnos de reutilización) opcional, pero recomendado.
  - `potenciaBase`: entero usado en el cálculo del efecto.
- **Métodos mínimos:**
  - `boolean puedeLanzarse(Personaje caster)`: verifica recurso y cooldown.
  - `void lanzar(Personaje caster, Personaje objetivo, Combate contexto)`: ejecuta el efecto (descuenta recurso).
    +1
  - `int calcularEfecto(Personaje caster, Personaje objetivo)`: opcional para separar el cálculo.

Gestión de cooldowns

- Si se implementa, cada personaje debe tener una estructura tipo Map<String, Integer> o Map<Hechizo, Integer>.
- Al lanzar un hechizo, se registra el cooldownMax y cada ronda se reduce hasta 0.

Tipos de hechizo requeridos (Mínimo funcional de 4)

- **Daño directo:** (ej. Bola de Fuego). Hace daño inmediato a un enemigo.
  +1
- **Curación directa:** (ej. Sanación). Restaura vida inmediata a un aliado.
- **Aplicación de daño en el tiempo (DoT):** (ej. Quemadura/Veneno). Aplica un Estado que hace daño por turno durante X rondas.
  +1
- **Aplicación de curación en el tiempo (HoT):** (ej. Renovar). Aplica un Estado de curación por turno durante X rondas.

---

6) Estados (Efectos por turnos)

Clase base Estado

- **Atributos mínimos:**
  - `nombre`: identificador del estado.
  - `turnosRestantes`: entero > 0.
  - `potenciaPorTurno`: entero (daño/curación por ciclo).
  - `tipo`: enum (DOT, HOT, MODIFICADOR).
    +1
- **Métodos mínimos:**
  - `void alAplicar(Personaje objetivo)`: se ejecuta al añadirse.
  - `void alProcesarTurno(Personaje objetivo)`: aplica daño/curación una vez por ronda.
  - `void alExpirar(Personaje objetivo)`: se ejecuta cuando turnos llega a 0.
  - `void reducirDuracion()`: decrementa turnosRestantes.

Estados requeridos (Subclases o instancias polimórficas)

- **Quemadura (DoT):** Resta vida cada turno.
- **Veneno (DoT):** Resta vida cada turno (suele durar más con menor potencia).
  +1
- **Renovar (HoT):** Cura vida cada turno sin superar vidaMax.

Interacción con la muerte

- Si un personaje muere (vida 0), deja de actuar y sus estados dejan de procesarse (mínimo) o se limpian.

---

7) Construcción de equipos

- El usuario elige 3 personajes, permitiendo repeticiones.
- El equipo enemigo puede ser por selección manual (Modo A) o por una IA simple (Modo B).

---

8) Flujo del juego (Consola)

- **Menú inicial:** iniciar combate / salir.
- **Creación de equipo:** elegir 3 clases y asignar armas.
- **Creación equipo enemigo:** manual o IA.
- **Bucle de combate por rondas:**
  1. Mostrar estado del combate.
  2. Jugador: elegir acción y objetivo por personaje vivo.
  3. Enemigo: acción por IA por personaje vivo.
  4. Procesar estados por turno.
  5. Comprobar victoria/derrota (cuando los 3 personajes de un equipo están a 0 de vida).
     +1

---

9) Estructuras de datos obligatorias

- **Array o ArrayList:** para equipos y listas de estados/hechizos.
- **Diccionario/Map (para al menos uno de estos):** Cooldowns de hechizos por personaje, catálogo de hechizos por clase, o inventario de armas disponibles.

1. Invariantes de la clase Personaje (Reglas internas)

El documento es muy específico con ciertas reglas matemáticas que no deben romperse nunca:

- La vida actual (`vidaActual`) debe estar siempre estrictamente entre `0` y `vidaMax`.
- El recurso actual (`recursoActual`) debe estar siempre entre `0` y `recursoMax`.
- Un personaje con `vidaActual = 0` se considera fuera de combate.

2. Reglas estrictas de Selección de Objetivos

Para el bucle de combate, el diseño exige que valides a quién se puede apuntar:

- **Regla de oro:** No se permite seleccionar personajes muertos en ningún caso.
- **Ataques con arma:** Siempre deben tener como objetivo a un enemigo vivo.
- **Hechizos:** Su objetivo depende de su tipo (ofensivo = enemigo vivo; apoyo/curación = aliado vivo o uno mismo).
  +1

3. Representación visual por Consola (toString)

El profesorado evaluará cómo se ve el juego en la terminal. Se exige que cada personaje ofrezca una representación textual.

- Debes implementar un método `toString()` o uno específico llamado `resumenCombate()` en la clase `Personaje`.
- Debe mostrar: nombre, clase, vida actual/máxima, recurso actual/máximo, estados activos (y sus turnos restantes) y el arma equipada.

4. Extensiones Opcionales (Punto 11 del PDF)

No son obligatorias para aprobar, pero el documento las lista para conseguir **nota extra**. Si quieres apuntar al 10, considera añadir alguna de estas a tu documento de diseño:

- Orden de turnos basado en una estadística de "velocidad" en lugar de un orden fijo.
- Mecánicas de críticos, esquiva, o resistencias elementales (fuego, naturaleza, sagrado).
- Hechizos en área (que afecten a todo el equipo rival o aliado de golpe).
- Un sistema de experiencia y niveles al terminar el combate.

En resumen: Tu estructura principal (Personajes, Armas, Hechizos, Estados, Bucle y Estructuras de Datos) está perfecta y cumple con los requisitos mínimos evaluables. Solo añade esos detalles de validación de objetivos y el método `toString()` para ir sobre seguro.

1. Acciones extra en el combate (Opcionales)

Aparte de "Atacar con arma" y "Lanzar hechizo", el documento menciona dos acciones más que puedes implementar para el turno de un personaje:

+1

- **Defender:** reduce el daño recibido hasta el siguiente turno o aumenta la defensa temporalmente.
- **Pasar turno:** el personaje decide no hacer nada.

2. Distribución exacta de hechizos por clase (Punto 5.4 del PDF)

El documento especifica cómo deben repartirse los hechizos obligatorios entre las clases para que el diseño sea coherente:

- **Mago:** Debe tener el hechizo de daño directo (ej. Bola de Fuego) y el hechizo que aplica Quemadura.
  +2
- **Sacerdote:** Debe tener la curación directa y la curación en el tiempo (Renovar).
- **Guerrero:** Puede no tener hechizos (lista vacía) o tener habilidades físicas (ej. Golpe Poderoso) que se modelen bajo el mismo sistema de hechizos usando coste o cooldown. El motor del juego debe permitir que un personaje no tenga hechizos; en ese caso, la opción de "Lanzar hechizo" no debe aparecer o debe gestionarse como inválida.
  +2

3. El Resumen Final de la partida

Cuando se cumple la condición de victoria y un equipo pierde a todos sus personajes , el combate finaliza inmediatamente y el sistema **debe mostrar un resumen final**.

+1

- Este resumen debe incluir: el equipo ganador, el número de rondas jugadas y el estado final de cada personaje.
- *(Opcional para nota)*: También puedes mostrar una tabla con las estadísticas de daño y curación total realizada por cada personaje y equipo.

4. Requisitos exactos de salida por consola por Ronda

En la sección del flujo del juego, el PDF especifica la información que *obligatoriamente* debe imprimirse en la terminal durante cada ronda de simulación:

- El estado actual de vida (y maná/energía si aplica).
- Los estados activos de cada personaje y sus turnos restantes.
- La acción realizada paso a paso (quién actúa, qué hace y a quién afecta).
- Un reporte de los efectos aplicados durante la fase de estados (los daños, las curaciones y las expiraciones de estados).

PERSONAJES

PERSONAJES

BANDO BUENO (Equipo de Geralt)

1. Geralt de Rivia

- **Clase Base:** Guerrero**.**
- **Rol:** Daño físico sostenido y resistencia.
- **Estadísticas Base:** `vidaMax` alta (ej. 800), `defensaBase` alta (ej. 50), `ataqueBase` muy alto (ej. 85).
- **Recurso:** Vigor (`recursoMax` = 100). Aunque es opcional para guerreros, usar una energía simple unifica el diseño.
- **Armas Equipables (Herencia):**
    - ***Espada Aerondight:*** Hereda de `ArmaCuerpoACuerpo`. Beneficia enormemente del `ataqueBase` de Geralt.
    - ***Ballesta Ursina:*** Hereda de `ArmaADistancia`. Menor daño base, pero ignora parte de la defensa enemiga.
- **Catálogo de Hechizos / Habilidades:** Las habilidades de guerrero se pueden modelar usando la clase `Hechizo` con coste de Vigor o cooldown**.**
    - ***Golpe Poderoso:*** Daño directo masivo a un enemigo.
    - ***Señal de Quen (Escudo):*** Extensión opcional que aplica un Estado de tipo `MODIFICADOR` para aumentar la defensa temporalmente.
- **Polimorfismo:** Sobrescribe el método `recibirDanio` para aplicar una reducción enorme si en su turno eligió la acción "Defender".

2. Yennefer de Vengerberg

- **Clase Base:** Mago**.**
- **Rol:** Daño mágico y aplicación de estados (DoT).
- **Estadísticas Base:** `vidaMax` baja (ej. 400), `defensaBase` baja (ej. 20), `poderMagico` muy alto (ej. 100).
- **Recurso: Maná/Caos (`recursoMax` alto, ej. 300).**
- **Armas Equipables (Herencia):**
    - ***Daga de Plata:*** Hereda de `ArmaCuerpoACuerpo`. Su ataque básico es muy débil.
    - ***Cuervo de Cristal:*** Hereda de `ArmaADistancia`. Lanza proyectiles mágicos simples.
- **Catálogo de Hechizos:** Su núcleo ofensivo.
    - ***Rayo de Yennefer:*** Hechizo de Daño Directo inmediato a un enemigo único.
    - ***Fuego de Vengerberg:*** Aplica el estado `Quemadura` (DoT) que resta vida cada turno durante X rondas.

- **Polimorfismo:** Sobrescribe un método tipo `calcularPoderHechizo()` para que sus hechizos escalen mejor con su estadística de `poderMagico`**.**

3. Ciri

- **Clase Base:** Sacerdote (Adaptada como Soporte/Sanadora).
- **Rol:** Soporte, curación directa y curación en el tiempo (HoT).
- **Estadísticas Base:** `vidaMax` media (ej. 600), `defensaBase` media, `ataqueBase` alto.
- **Recurso:** Poder de la Vieja Sangre (`recursoMax` medio/alto**).**
- **Armas Equipables (Herencia):**
  - ***Espada Zireael:*** Hereda de `ArmaCuerpoACuerpo`. Tiene un `multiplicadorCritico` opcional muy alto.
  - ***Amuleto Mágico:*** Hereda de `ArmaADistancia`. Permite atacar desde lejos con magia pura.
- **Catálogo de Hechizos:**
  - ***Poción de Golondrina:*** Curación directa que restaura vida inmediata a un aliado.
  - ***Aura de la Vieja Sangre:*** Aplica el estado `Renovar` (HoT), curando vida cada turno a un aliado sin superar su `vidaMax`.
- **Polimorfismo:** Sobrescribe las reglas de curación; por ejemplo, otorga una bonificación al curar a un aliado que tenga muy poca vida.

Triss Merigold

Clase Base: Mago (variante híbrida ofensiva/soporte)

Diferente a Yennefer porque se enfoca más en control y en combinar daño inmediato con estados, mientras Yennefer es pura potencia de DoT.

Estadísticas Base: vidaMax media (450), defensaBase baja (25), poderMagico alto (90), recursoMax alto (280).

Armas Equipables:

- *Daga Encantada* → hereda de ArmaCuerpoACuerpo. Daño base bajo pero aplica un pequeño tick de fuego al golpear (opcional).
- *Amuleto de Mariposas* → hereda de ArmaADistancia. Proyectiles mágicos ligeros.

Catálogo de Hechizos:

- *Llamarada de Triss*: daño directo inmediato a un enemigo único.
- *Trampa de Fuego*: aplica el estado Quemadura igual que Yennefer, pero con menor potencia y más duración (aquí puedes demostrar el polimorfismo de `calcularPoderHechizo()` con resultados distintos entre las dos magas).

- *Escudo Mágico* (opcional): estado de tipo MODIFICADOR que sube la defensaBase de un aliado temporalmente.

Polimorfismo: Sobrescribe `calcularPoderHechizo()` para que sus estados duren más turnos pero con menos daño por tick que los de Yennefer, diferenciando a las dos magas de forma real.

BANDO MALO (La Cacería Salvaje)

4. Imlerith

- **Clase Base:** Guerrero**.**
- **Rol:** Tanque pesado y daño físico sostenido**.**
- **Estadísticas Base:** `vidaMax` altísima (ej. 1000), `defensaBase` altísima (ej. 70), poca `precision` o velocidad.

- **Recurso:** Ira (Valor 0 o energía simple).
- **Armas Equipables (Herencia):**
  - ***Maza Gigante:*** Hereda de `ArmaCuerpoACuerpo`. Daño base extremadamente alto, pero penaliza la precisión.

  - ***Hacha Arrojadiza:*** Hereda de `ArmaADistancia`. Útil si el jugador no le permite acercarse.
- **Catálogo de Hechizos / Habilidades:**
  - ***Embestida Brutal:*** Habilidad física equivalente a un hechizo de daño directo con un `cooldownMax` alto (se registra en el diccionario/mapa de cooldowns).

- **Polimorfismo:** Puede sobrescribir los métodos de cálculo para resistir mejor el daño físico que el mágico debido a su pesada armadura.

5. Caranthir

- **Clase Base:** Mago**.**
- **Rol:** Daño mágico y aplicación de daño en el tiempo.
- **Estadísticas Base:** `vidaMax` media (ej. 500), `defensaBase` baja, `poderMagico` alto.
- **Recurso:** Magia de Escarcha (`recursoMax` alto).
- **Armas Equipables (Herencia):**
  - ***Báculo de Navegador:*** Hereda de `ArmaCuerpoACuerpo`.
  - ***Orbe de Escarcha Blanca:*** Hereda de `ArmaADistancia`. Dispara esquirlas de hielo.
- **Catálogo de Hechizos:**
  - ***Lanza de Hielo:*** Daño directo inmediato a un enemigo.

    - ***Tormenta de Nieve:*** Hechizo que aplica el estado `Veneno` (adaptado narrativamente como "Congelación"). Resta vida cada turno y puede durar más turnos con menor potencia.

- **Polimorfismo:** Sus hechizos de daño en el tiempo sobrescriben la potencia base para durar más tiempo que los de Yennefer.

6. Eredin (Rey de la Cacería)

- **Clase Base:** Sacerdote (Adaptado como Líder/Soporte Oscuro).
- **Rol:** Mantener vivos a sus generales.
- **Estadísticas Base:** `vidaMax` alta, `defensaBase` media, `poderMagico` medio.
- **Recurso:** Magia Oscura (`recursoMax` medio/alto).
- **Armas Equipables (Herencia):**
    - ***Espada de la Cacería:*** Hereda de `ArmaCuerpoACuerpo`.
    - ***Proyección Espectral:*** Hereda de `ArmaADistancia`.
- **Catálogo de Hechizos:**
    - ***Sacrificio Oscuro:*** Curación directa a un aliado (Imlerith o Caranthir).
    - ***Presencia del Rey:*** Aplica el estado `Renovar` (HoT) a un miembro de la Cacería.
    - ***Disipación:*** Hechizo opcional para limpiar estados alterados (como la Quemadura de Yennefer**).**
- **Polimorfismo:** Sobrescribe la prioridad de su Inteligencia Artificial. Eredin siempre buscará curar al aliado que esté más débil.

Nithral

Clase Base: Guerrero (tanque de élite, general de la Cacería)

Diferente a Imlerith porque Nithral es más agresivo y rápido; Imlerith es el tanque puro. Nithral tiene menos vida pero más capacidad ofensiva.

Estadísticas Base: vidaMax alta (850), defensaBase alta (55), ataqueBase alto (90), recurso: Ira (opcional, valor simple).

Armas Equipables:

- *Espada Espectral de la Cacería* → hereda de ArmaCuerpoACuerpo. Alto daño base, buen multiplicadorCritico.
- *Lanza Espectral* → hereda de ArmaADistancia. Lanzamiento de proyectiles de energía espectral.

Catálogo de Hechizos / Habilidades:

- *Corte Espectral*: habilidad física (modelada como Hechizo con cooldown) de daño directo alto a un enemigo.

- *Armadura Espectral* (opcional): estado MODIFICADOR que sube su propia defensaBase durante X turnos, simulando que invoca su armadura fantasmal.

Polimorfismo: Sobrescribe `recibirDanio()` para ser más resistente al daño mágico que al físico (distinto a Imlerith, que resiste ambos por igual), reforzando que dos guerreros pueden comportarse de manera diferente.

1. Las Clases Base y sus Métodos Obligatorios

- **Clase Base `Personaje`:**
  - *Atributos:* nombre, tipoClase, vidaMax/vidaActual, recursoMax/recursoActual, ataqueBase, defensaBase, arma (objeto Arma), estadosActivos (ArrayList), hechizos (ArrayList),
  - *Métodos:* `estaVivo()`, `recibirDanio(int)`, `curar(int)`, `gastarRecurso(int)`, `equiparArma(Arma)`, `aplicarEstado(Estado)`, `procesarEstados()`, y un `toString()` o `resumenCombate()` para pintar su estado en consola.

- **Clase Base `Arma`:**
  - *Atributos:* nombre, tipo (Cuerpo a cuerpo/Distancia), danioBase, modificador/critico.
  - *Métodos:* `calcularDanio(Personaje atacante, Personaje defensor)` (Este método no debe modificar la vida directamente, solo devolver un número entero).

- **Clase Base `Hechizo`:**
  - *Atributos:* nombre, costeRecurso, tipoObjetivo (ENEMIGO, ALIADO, PROPIO), cooldownMax, potenciaBase.
  - *Métodos:* `puedeLanzarse(Personaje caster)`, `lanzar(Personaje caster, Personaje objetivo)`.
- **Clase Base `Estado`:**
  - *Atributos:* nombre, turnosRestantes, potenciaPorTurno, tipo (DOT, HOT, MODIFICADOR).
  - *Métodos:* `alAplicar()`, `alProcesarTurno()`, `alExpirar()`, `reducirDuracion()`

Los 3 Estados obligatorios (Subclases de `Estado`, gestionados en el bucle del final de ronda):

- 🔥 **Quemadura (DOT):**
  - *Lore:* El objetivo está en llamas por la magia de Yennefer.
  - *Efecto (`alProcesarTurno`):* Pierde 50 HP durante 3 turnos.
- 🧪 **Veneno (DOT):**
  - *Lore:* Intoxicación por sangre negra o frío de la Escarcha Blanca.
  - *Efecto (`alProcesarTurno`):* Pierde 30 HP durante 5 turnos (dura más que la quemadura pero hace menos daño por turno).
- 💚 **Renovar (HOT):**
  - *Lore:* Regeneración celular por pociones o magia.
  - *Efecto (`alProcesarTurno`):* Recupera 40 HP durante 4 turnos (nunca superando la `vidaMax`).

3. El Grimorio: Hechizos y Estados obligatorios

Aquí es donde entra el sistema de **diccionarios (Map)** para los *cooldowns* y el **ArrayList** para los hechizos de cada clase.

Los 4 Hechizos obligatorios (Implementan la clase base `Hechizo`):

1. **Daño Directo:** `Señal de Igni` (Yennefer/Geralt) o `Lanza de Hielo` (Caranthir).
   - *Objetivo:* ENEMIGO_UNICO. Aplica daño inmediato usando `potenciaBase` + `poderMagico`.
2. **Curación Directa:** `Poción de Golondrina` (Ciri) o `Sacrificio Oscuro` (Eredin).
   - *Objetivo:* ALIADO_UNICO. Restaura vida inmediata (ej. +200 HP).
3. **Aplicación de DoT (Daño en el tiempo):** `Fuego de Vengerberg` (Yennefer) o `Escarcha Corrosiva` (Caranthir).
   - *Objetivo:* ENEMIGO_UNICO. No hace daño al golpear, pero le inyecta el Estado **Quemadura** al enemigo.
4. **Aplicación de HoT (Curación en el tiempo):** `Aura de la Vieja Sangre` (Ciri) o `Presencia del Rey` (Eredin).
   - *Objetivo:* ALIADO_UNICO. Le aplica el Estado **Renovar** a un compañero.

Cada turno tiene diferentes fases, lo primero que se revisa es el estado.

Combate

COMBATE

1. El Motor de Combate (Clase `Combate` o `Juego`)

El PDF exige un ciclo de vida muy específico divido en 3 Fases por Ronda. No puedes mezclar todo; debe seguir este orden estricto para evitar errores lógicos:

Estructura del Bucle Principal (`while`): Mientras (EquipoGeralt tiene vivos Y EquipoCaceria tiene vivos):

1. **Fase 1: Acciones (Turnos)**
   - **Recorres todos los personajes vivos (ordénalos por velocidad si quieres nota extra, o fijo Lobo -> Cacería).**
   - **Si es turno del Jugador: Muestras menú (Atacar/Hechizo/Defender). Validas entrada. Ejecutas acción.**
   - **Si es turno de la IA (Enemigo): Elige un objetivo aleatorio o al más débil (según la inteligencia de Eredin) y ataca.**
   - ***Importante:*** **Si un personaje muere en medio de la ronda (ej. Geralt mata a Imlerith), Imlerith no debe actuar si le tocaba después.**
2. **Fase 2: Procesamiento de Estados (El paso del tiempo)**
   - **Al terminar todos los turnos, iteras sobre todos los personajes.**
   - **Llamas a `personaje.procesarEstados()`.**
   - **Aquí es donde Yennefer sufre daño por veneno o Ciri se cura por Renovar.**
   - **Si un estado llega a 0 turnos, se borra de la lista.**
3. **Fase 3: Comprobación de Victoria**
   - **Verificas: ¿Queda alguien vivo en el equipo A? ¿Y en el B?**
   - **Si un equipo muere entero -> `break` del bucle y fin del juego.**

---

2. La Lógica de "Apilar Estados" (Crucial para evaluación)

El PDF es muy estricto con esto. No basta con añadir estados al ArrayList. Debes implementar esta lógica dentro del método `aplicarEstado(Estado nuevoEstado)` de la clase `Personaje`:

Algoritmo obligatorio:

1. **Recorrer la lista `estadosActivos` del personaje.**
2. **¿Ya existe un estado con el mismo `nombre` (ej. "Quemadura")?**

- **SÍ: No añadas uno nuevo. Simplemente reinicia sus `turnosRestantes` al máximo (o suma duración, según tu diseño, pero reiniciar es lo estándar y más fácil).**
  +1
- **NO: Añade el `nuevoEstado` al ArrayList.**

Esto evita que Imlerith tenga 5 iconos de "Quemadura" a la vez; solo tendrá una quemadura que se refresca.

---

3. Gestión de Cooldowns con Mapas (Diccionarios)

Mencionaste que tenías el mapa, pero el PDF pide usarlo activamente. En la clase `Personaje`, necesitas: `Map<String, Integer> cooldowns`.

- **Al lanzar hechizo: Si Triss lanza "Llamarada" (Cooldown: 3 turnos), guardas en el mapa: `cooldowns.put("Llamarada", 3)`.**
- **Al validar lanzamiento: Antes de lanzar, verificas: `if (cooldowns.containsKey("Llamarada") && cooldowns.get("Llamarada") > 0) -> return false;`.**

- **En cada ronda: Debes recorrer el mapa y restar 1 a todos los contadores.**

---

4. Salida por Consola (El "Front-end")

El profesor evaluará que la consola muestre exactamente lo que pasa. No puede ser un "Geralt ataca". Debe ser detallado:

Formato obligatorio por ronda:

Plaintext

=== RONDA 4 ===

[ESTADO]

Geralt: 500/800 HP | Estados: [Renovar (1 turno)]

Imlerith: 200/1000 HP | Estados: [Quemadura (2 turnos)]

[ACCIONES]

> Geralt usa "Golpe Poderoso" contra Imlerith. ¡300 de daño!

> Imlerith muere.

[EFECTOS DE ESTADO]

> Geralt recupera 40 HP por Renovar.

> El estado Renovar de Geralt ha expirado.

---

5. Menú de Selección de Equipos (Colecciones)

Antes del combate, necesitas un "Catálogo".

- **Crea una clase `Catalogo` o `GameData` con listas estáticas: `todosLosPersonajes`, `todasLasArmas`.**
- **Usa un `Map<String, Personaje>` para buscar rápido.**
- **El usuario debe escribir "Geralt" y el sistema debe clonar/instanciar a Geralt y meterlo en su `ArrayList<Personaje> equipoJugador`.**

---

6. Resumen Final (Post-Combate)

Cuando el `while` termina, el PDF pide un informe final obligatorio:

- **Equipo Ganador.**
- **Número de rondas totales.**
- **Estado final de los supervivientes (cuánta vida les quedó).**
- ***(Opcional para el 10)*: Tabla de "Daño Total Realizado" por cada personaje (necesitarías una variable contador `dañoTotal` en la clase Personaje que sume cada vez que ataca).**

Resumen de Tareas para Programar (Checklist del 100%)

1. **[ ] Clase `Combate`: Implementar el bucle `while` con las 3 fases (Acción -> Estados -> Check).**
2. **[ ] Método `aplicarEstado`: Programar la lógica de "si existe, renueva; si no, añade".**
3. **[ ] Cooldowns: Implementar el `Map` y la resta de -1 turno en cada ronda.**

4. **[ ] Validaciones: Asegurar en el código que `vida` nunca baja de 0 y que no se puede atacar a muertos.**
   +1
5. **[ ] Consola: Asegurar que imprimes los logs de la "Fase de Estados" separado de la "Fase de Acciones".**
6. **[ ] Main: Menú para elegir los 3 héroes del catálogo.**