FACULTAD DE INGENIERIA
Universidad de Buenos Aires

# Trabajo Práctico 2 — Java

[7507/9502] Algoritmos y Programación III
Curso 1
Segundo cuatrimestre de 2023


| **Alumno** | **Padrón** | **Email** |
|---|---|---|
| Angie Isabella Valdivia Wong | 103727 | avaldivia@fi.uba.ar |
| Gabriela Alejandra Yanes Salas | 108325 | gyanes@fi.uba.ar |
| Francisco Lopez | 107614 | frlopez@fi.uba.ar |
| Ignacio Urbieta | 108311 | iurbieta@fi.uba.ar |
| Roy Fiorilo | 108419 | rfiorilo@fi.uba.ar |

# Índice

# 1. Introducción

Este informe detalla la solución implementada para el segundo trabajo práctico de la materia "Algoritmos y Programación III". El objetivo principal fue el desarrollo colaborativo de una aplicación, integrando de manera efectiva los conocimientos adquiridos a lo largo del curso. Para ello, se empleó Java, un lenguaje de programación de tipado estático, enfocándonos en una arquitectura de diseño orientada a objetos. Además, se adoptaron prácticas de Test Driven Development (TDD) y de Integración Continua, asegurando así la calidad y eficiencia del software desarrollado.

# 2. Supuestos

Dentro de la implementación de nuestro proyecto para el trabajo práctico, hemos establecido varios supuestos clave que guían el funcionamiento y las limitaciones del código. Estos son:

- **Movilidad del Jugador:** Hemos definido que el jugador solo puede desplazarse a través de las casillas de tipo "Camino". Esta decisión implica que el resto del mapa, compuesto por diversas otras casillas, está fuera del alcance del movimiento del jugador.
- **Comportamiento del Seniority:** Diseñamos el Seniority como una clase que se independiza del turno y ella misma sabe cuando cambiar.

# 3. Diagramas de clases

Para los diagramas de clase empezamos presentando dichos diagramas sin metodos (para poder observar bien las relaciones) y luego extenderemos la explicacion mostrandolos con algunos metodos.

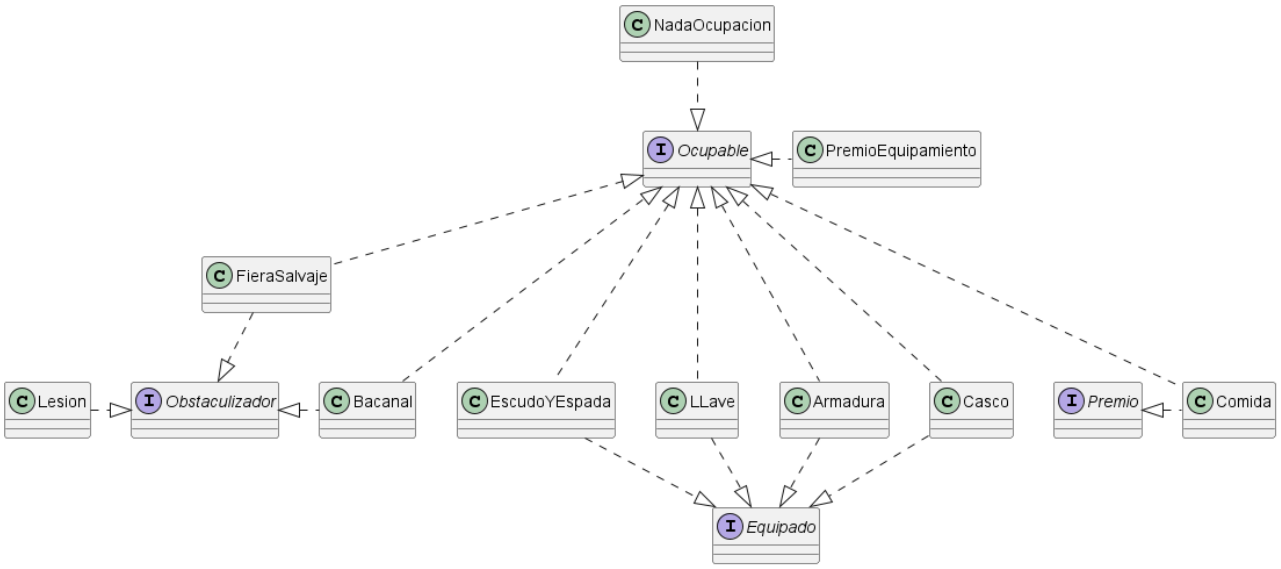


Figura 1: Diagrama de clases parte 1 (sin métodos)

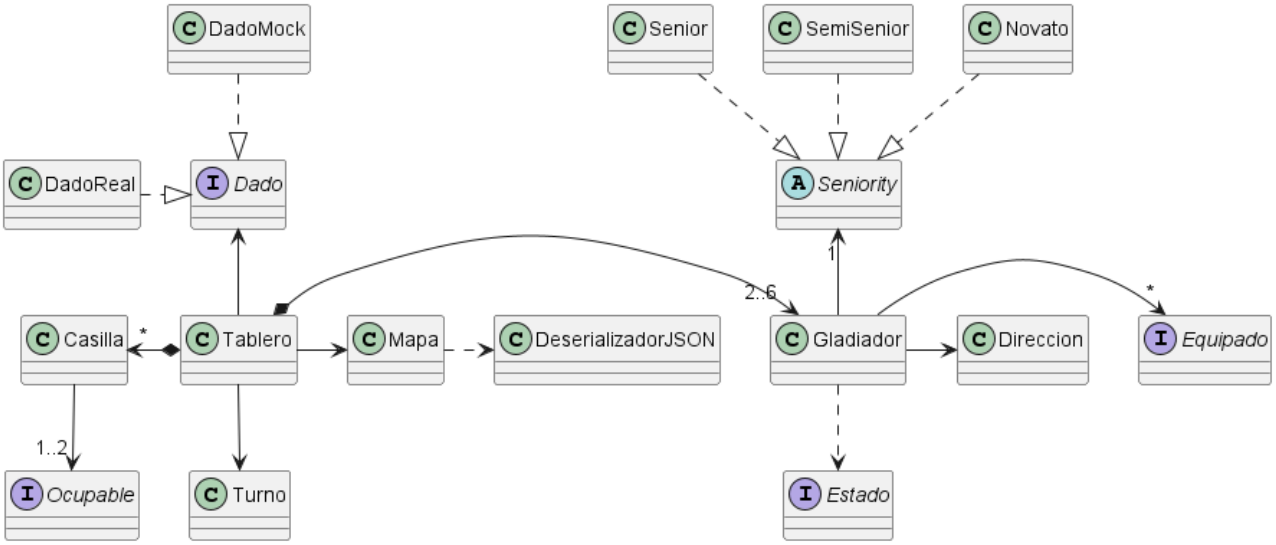


Figura 2: Diagrama de clases parte 2 (sin métodos)

## 3.1. Desarrollaremos sobre el Diagrama

- **Clase Mapa:** Representa un mapa compuesto por casillas. Incluye métodos para inicializar mapas de prueba o reales y obtener el mapa.
- **Clase DeserializadorJSON:** Se encarga de deserializar datos JSON para construir objetos Ocupable. Contiene métodos para extraer dimensiones del mapa, contenido de cada celda y transformar a un objeto Ocupable.
- **Clase Tablero:** Representa el tablero del juego con métodos para validar el turno, avanzar, finalizar el juego, etc.
- **Clase Turno:** Representa el turno actual en el juego con métodos para jugar, obtener el turno, avanzar al siguiente turno y validar si el juego debe finalizar.

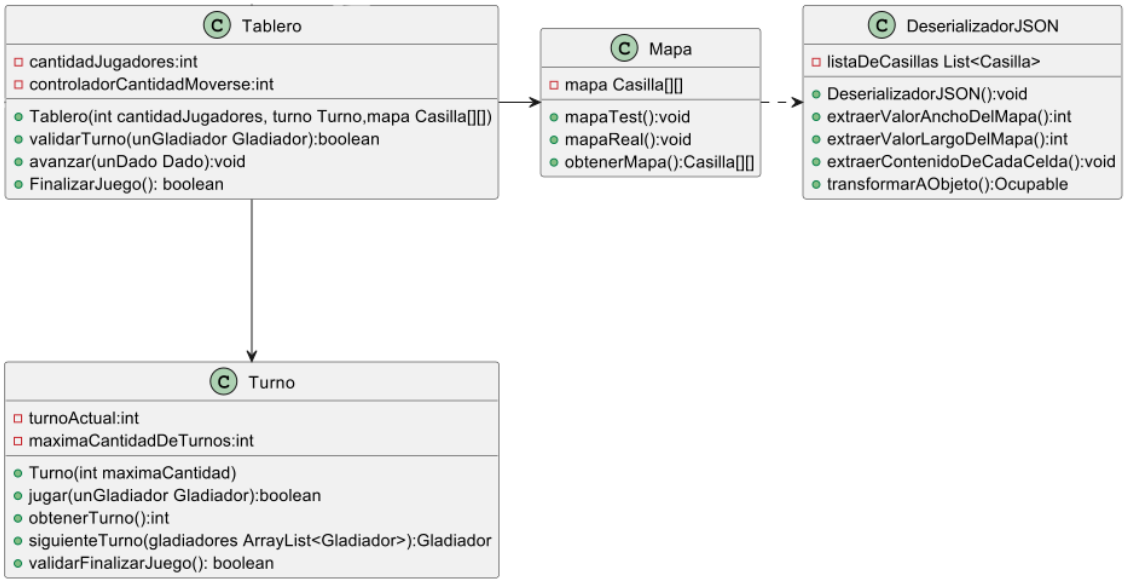


Figura 3: Relacion De Tablero con Turno, Mapa y Desrealizador JSON

Vemos entonces su relacion y los metodos que se aplican en cada una de las clases (las demasrelacione las mostramos en los diagramas de mas arriba).

- **Clase Gladiador:** Representa a un gladiador en el juego con atributos como energía, posición, estado de lesión y dirección. Tiene métodos para realizar acciones como avanzar, retroceder, combatir, verificar si tiene una llave, lesionarse, rehabilitarse y verificar si está lesionado.
- **Clase Direccion:** Representa una dirección con posiciones en X e Y. Contiene métodos para obtener la fila y columna, y para obtener la próxima casilla con camino en un mapa.
- **Interfaz Estado:** Define un método abstracto `ejecutarAccion` que toma una lista de equipamiento y no devuelve nada.
- **Clase Seniority y subclases (Novato, SemiSenior, Senior):** Representan niveles de experiencia con métodos para modificar energía y sumar turnos.

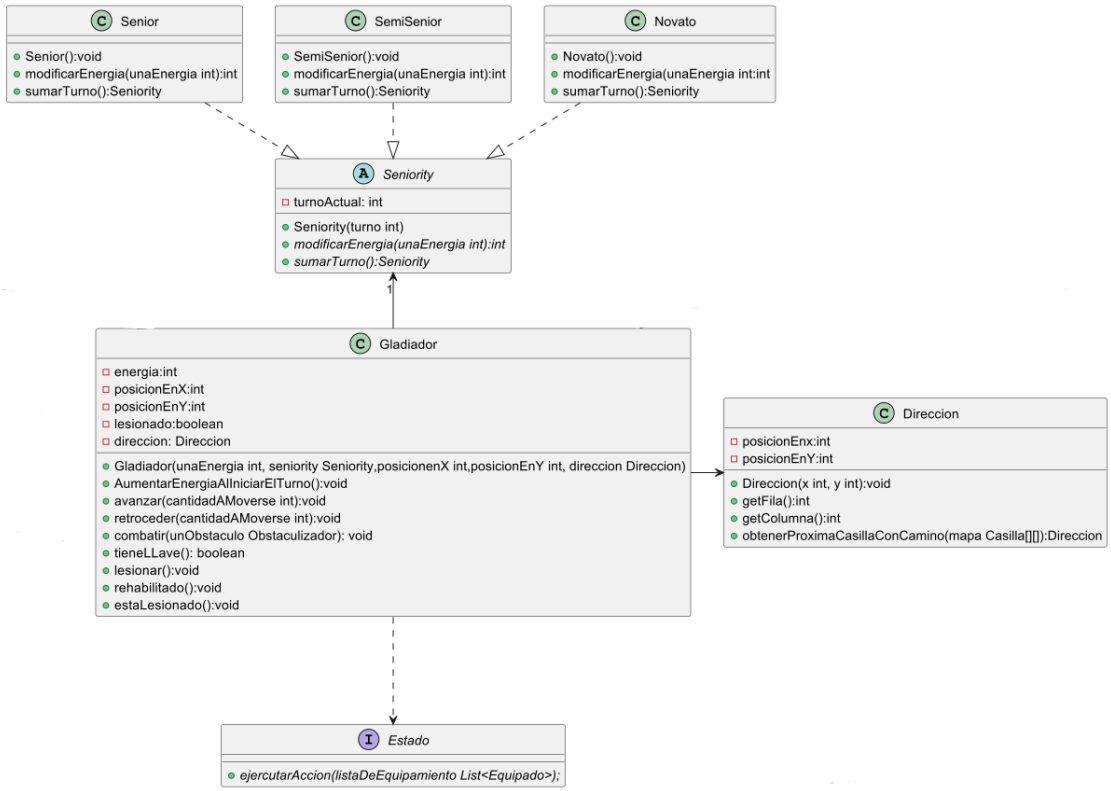


Figura 4: Relacion De Gladiador con Seniority, Direccion y Estado

- **Interfaz Equipado:** Define un método abstracto `modificarEnergia` que toma un entero como parámetro y devuelve un entero.
- **Clase NadaOcupacion:** Implementa la interfaz Ocupable con métodos para interactuar con la ocupación.
- **Interfaz Equipado:** Define un método abstracto `modificarEnergia` que toma un entero como parámetro y devuelve un entero.
- **Clases de Equipamiento (Casco, EscudoYEspada, Armadura, LLave, Comida):** Implementan la interfaz Equipado con métodos para modificar energía.

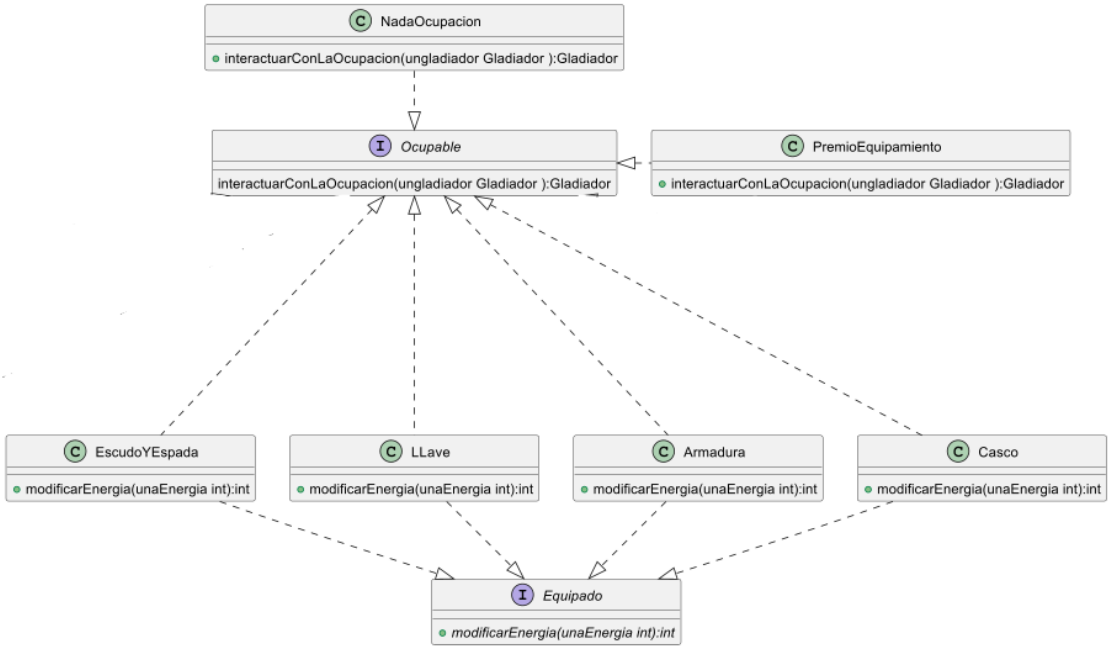


Figura 5: Relacion De Ocupable con los equipamientos, NadaOcupacion y Equipado

- **Clase Casilla:** Representa una casilla en el mapa con métodos para interactuar con la ocupación y obtener el equipo.
- **Interfaz Dado:** Define un método abstracto `lanzarDado` que devuelve un entero.
- **Clases DadoMock y DadoReal:** Implementan la interfaz Dado con métodos para lanzar el dado.
- **Interfaz Ocupable:** Define un método `interactuarConLaOcupacion` que toma un gladiador como parámetro y no devuelve nada.
- **Interfaces Premio y Obstaculizador:** Definen métodos abstractos para modificar energía y combatir, respectivamente.
- **Clases FieraSalvaje, Bacanal, Lesion y PremioEquipamiento:** Implementan las interfaces Premio y Obstaculizador.

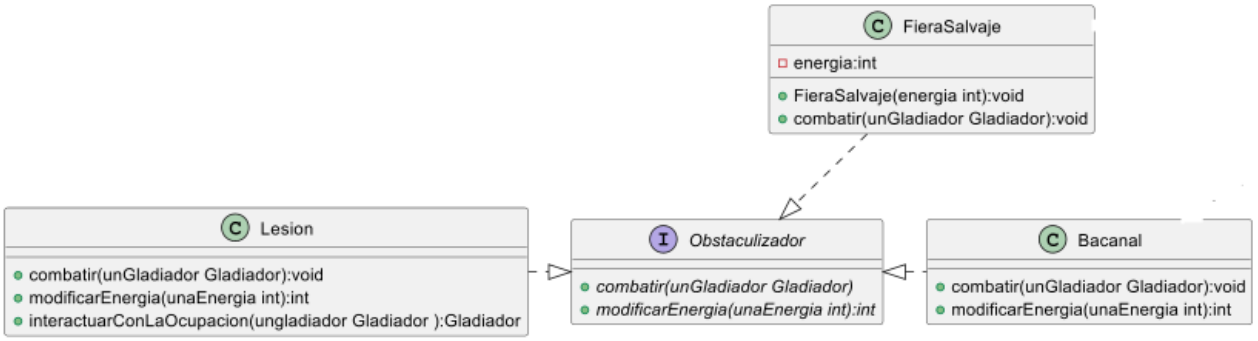


Figura 6: Relacion De Obstaculizador con Lesion, FieraSalvaje y Bacanal

# 4. Diagramas de secuencia

Para ambos diagramas de secuencia, tomamos por inicializado el mapa, turno, tablero, gladiador, direccion, dado y posicion.

Como primer diagrama de secuencia, veremos la secuencia de un gladiador que cae en una casilla con una lesion y pierde su siguiente turno. Dividimos el diagrama en 2 para que sea mas legible:

**UnJugadorLesionadoNoPuedeJugarElSiguienteTurno**

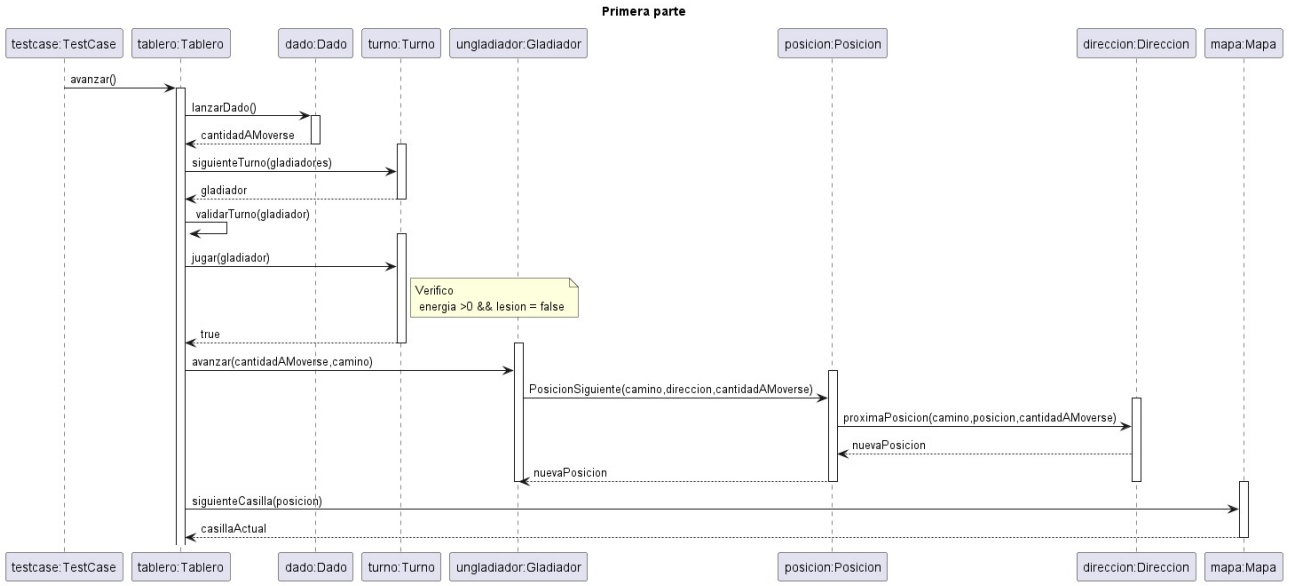


Figura 7: Parte 1

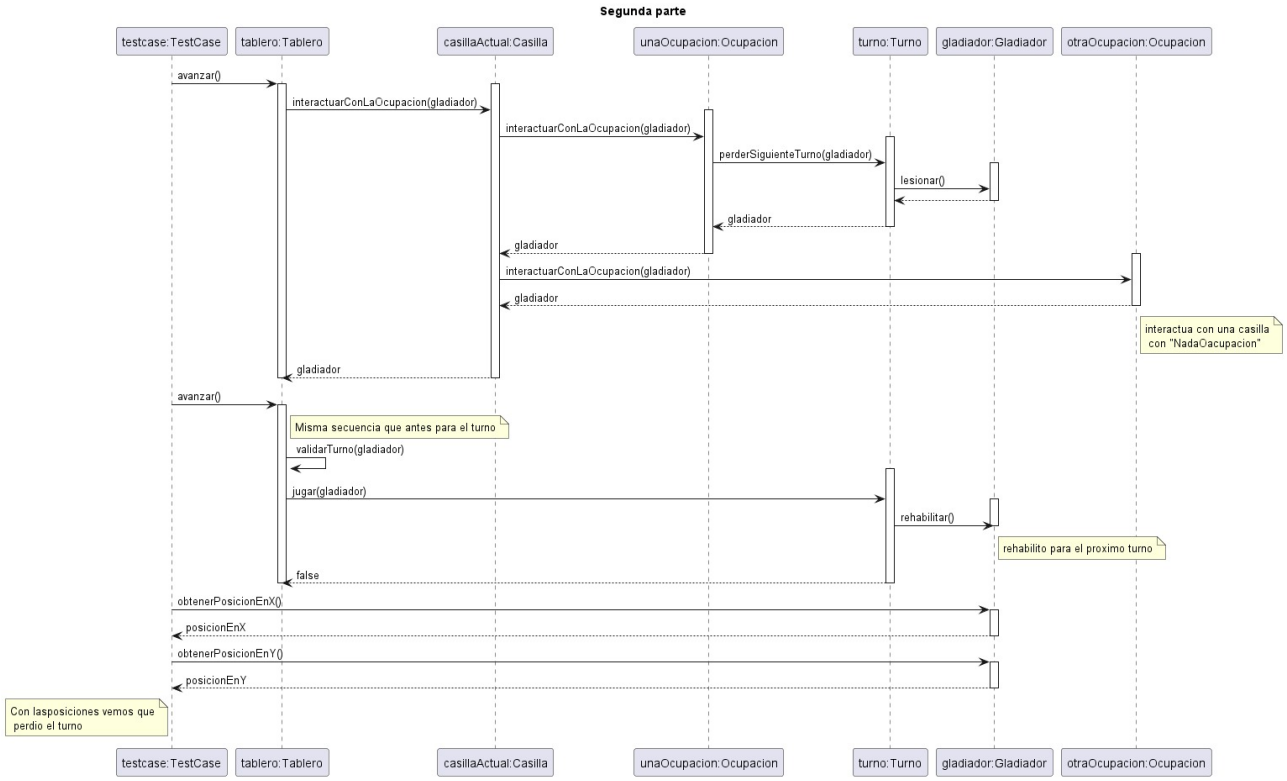


Figura 8: Parte 2

Para el segundo diagrama de secuencia, veremos un gladiador que cae en una casilla con un equipamiento y una fiera salvaje (test7 CasosDeUso1). Para este diagrama de secuencia lo dividiremos en dos partes que pueda ser mas legible.

### VerificarQueSiHayUnCombateConUnaFieraSalvajeYTieneCascoPierde15Puntos

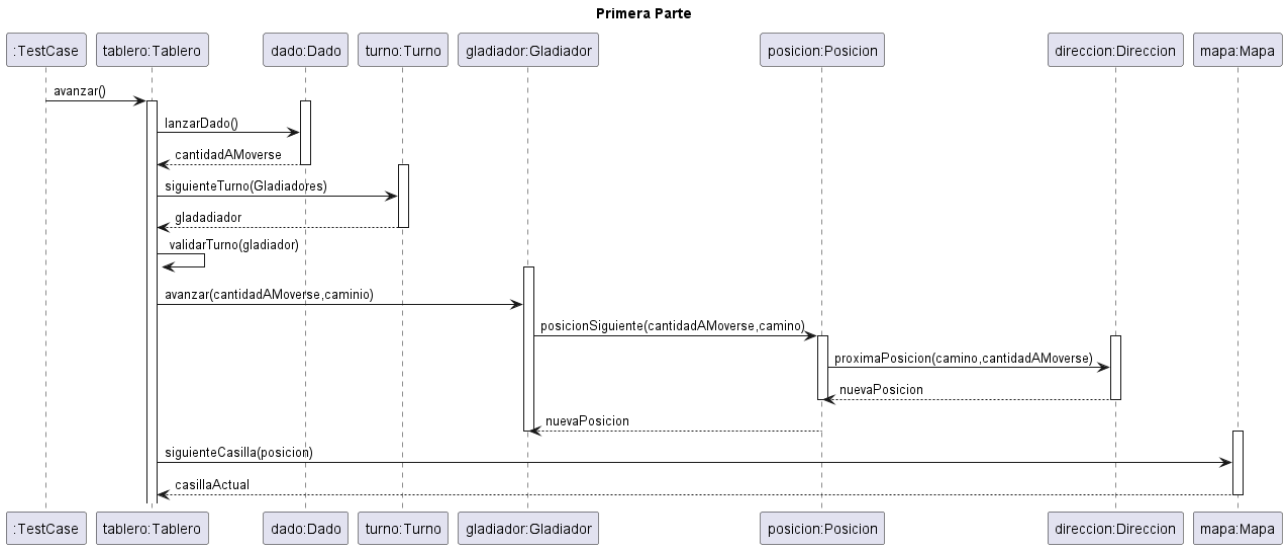


Figura 9: Parte 1

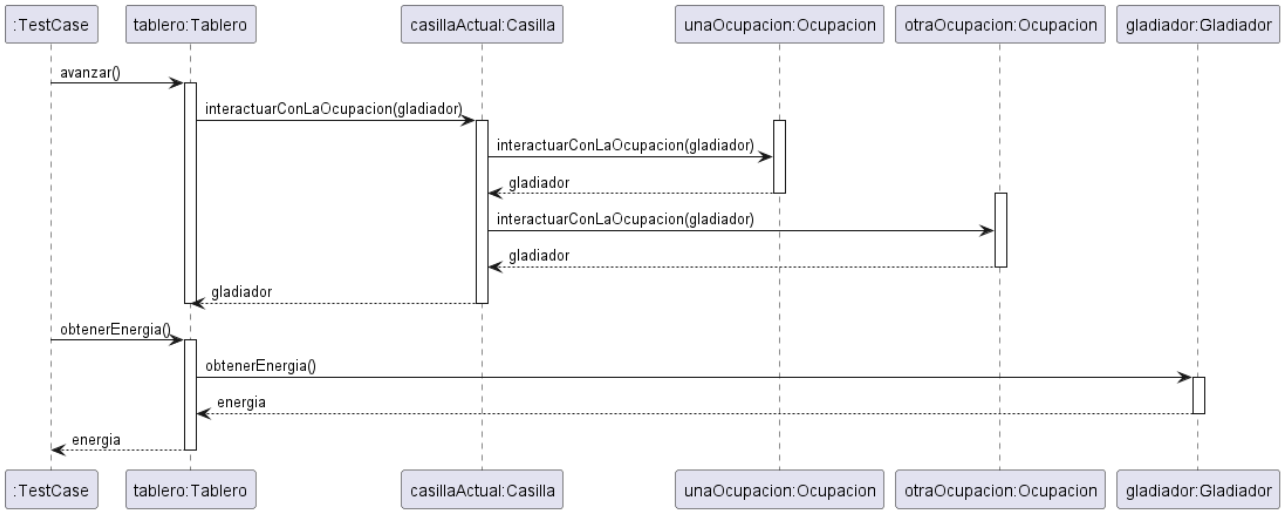


Figura 10: Parte 2

# 5. Diagramas de paquetes

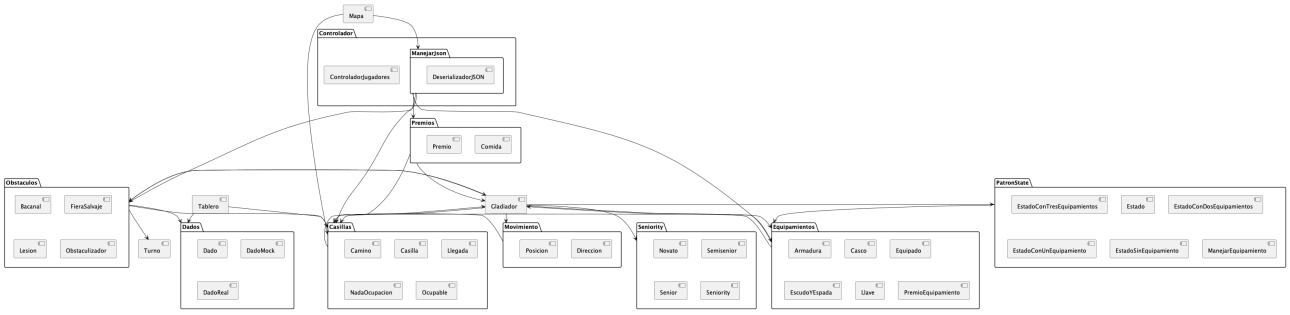


Figura 11: Diagrama de paquetes del modelo

# 6. Diagramas de estado

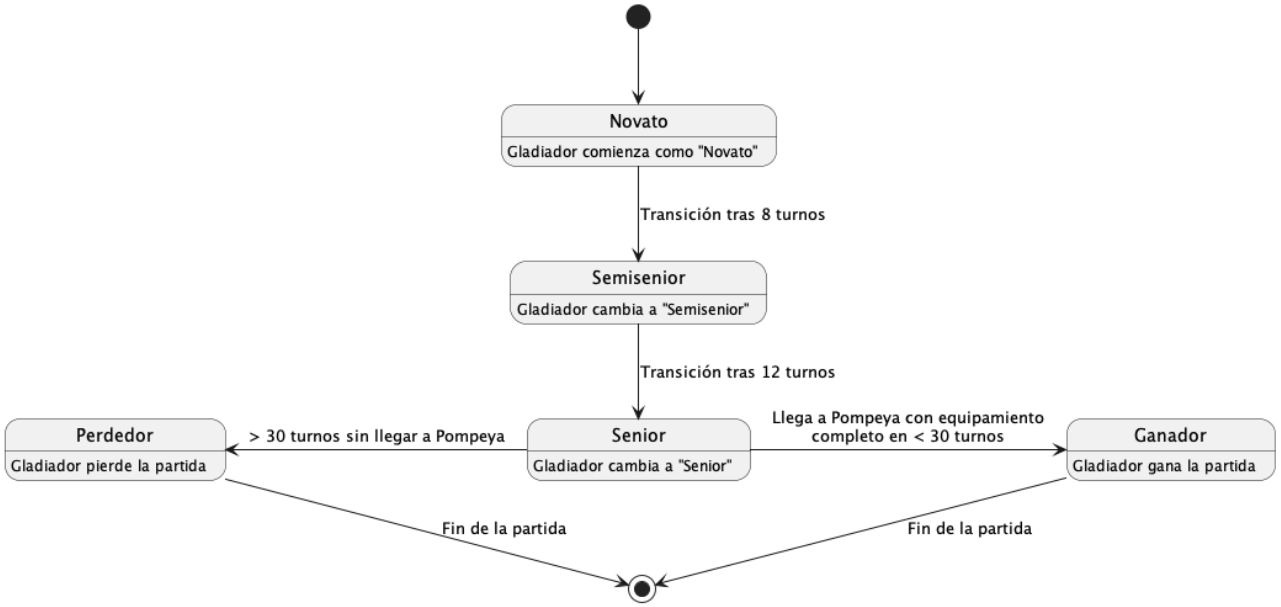


Figura 12: Diagrama de estados que muestra de forma simple los cambios de Seniority de un gladiador a lo largo de la partida.

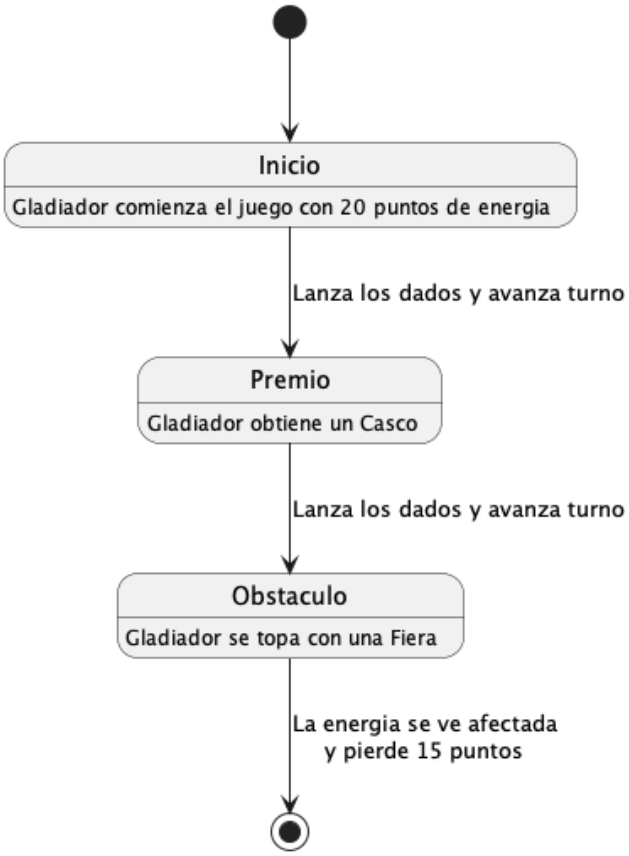


Figura 13: Diagrama de estados que muestra muestra el estado del gladiador al enfrentarse a una fiera, estando equipado con un casco como armadura, y detalla el impacto resultante en su energía.

# 7. Detalles de implementación

Para este trabajo practico pensamos en como diseñar varios objetos que puedan comunicarse entre si, para nuesto modelo utilizamos principalmente, un objeto Gladiador (que sera la representacion de jugador) y un tablero (el cual sera el ejecutante principal de la mayoria de las acciones de nuestro modelo).

Algunos detalles de la implementación:

## 7.1. Mapa

Diseñamos el mapa como una matriz. Este mapa contiene el camino por el cual el gladiador se mueve, teniendo en cuenta que solo se va a poder mover por el camino y no por el resto del mapa. Para esto, construimos nuestra matriz y luego, deserealizando el .json, vamos añadiendo el camino a nuestra matriz.

## 7.2. Tablero

El Tablero es la clase principal de nuestro modelo, ya que encapsula el mapa, la lista de jugadores, el dado y el registro del turno actual. Esta entidad permite no solo la adición de nuevos jugadores, sino también su progresión en el juego. Mediante la función "avanzar", se lanzan el dado y se mueven por el mapa conforme a las casillas indicadas. Esta función también se encarga de validar la casilla correspondiente a visitar y determinar cuál gladiador será el siguiente en jugar.

## 7.3. Seniority

Además, implementamos un sistema de "seniority". Este sistema está diseñado para reconocer y ejecutar el cambio de estado de los jugadores, permitiéndoles avanzar al siguiente nivel de seniority. Para construir el seniority realizamos una herencia entre clases, ya que al compartir métodos y atributos, pudimos reutilizar el código. Además nos independizamos de Turno, ya que asumimos que el seniority tiene que ir avanzando a medida que pasan los turnos (sin importar si el jugador tiene energía o no).

## 7.4. Equipamiento

Para el equipamiento aplicamos el patrón de diseño State, en el que se vinculan los estados y se va actualizando el estado a medida que se van añadiendo equipamientos al gladiador. Decidimos utilizar este patrón ya que nos permite cambiar el estado del equipamiento del gladiador en tiempo de compilación, con esto podemos independizarnos del uso de condicionales y acoplamiento en el código.

## 7.5. Obstáculos

En nuestro juego, gestionamos tres tipos de obstáculos, cada uno con un impacto distinto en la mecánica del juego:

1. **Obstáculo Fiera Salvaje:** Cuando el gladiador se enfrenta a este obstáculo, su energía se reduce. La cantidad de energía disminuida depende del equipamiento del gladiador.
2. **Obstáculo Lesión:** Este obstáculo impone un estado de *lesión* al gladiador, impidiéndole jugar el próximo turno. Mediante la función `rehabilitar`, restablecemos el estado de *lesión* a falso, para que el gladiador pueda seguir jugando.
3. **Obstáculo Bacanal:** Aquí, el gladiador debe tomar una cantidad de tragos determinada por el lanzamiento del dado. La energía del gladiador disminuye en 4 puntos por cada trago consumido.