FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — AlgoRoma

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

Brando, Hernan	ebrando@fi.uba.ar
De Leon, Franco	fdeleon@fi.uba.ar
Fanti, Gerónimo	gfanti@fi.uba.ar
Romero, Fabiola	fromerop@fi.uba.ar
Valle, Valentino	vvalle@fi.uba.ar

Índice

1. Introducción

El presente informe reúne la documentación del segundo trabajo práctico de la materia Algoritmos y Programación III. Consiste en desarrollar una aplicación completa de un juego de mesa, con tablero y por turnos, utilizando un lenguaje de tipado estático como lo es Java, siguiendo un diseño de modelo orientado a objetos, trabajando con las técnicas de TDD e Integración Continua.

2. Objetivo

El objetivo del trabajo fue desarrollar una aplicación completa de a cuerdo a las especificaciones funcionales del juego y los supuestos realizados, incluyendo modelo de clases, sonidos, interfaz gráfica, pruebas unitarias e integradas y la presente documentación del diseño y desarrollo del mismo.

3. Especificaciones Funcionales del juego y Supuestos

AlgoRoma es un juego de tablero por turnos entre 2-6 jugadores ambientado en el imperio romano. Cada jugador tomará un gladiador con el objetivo de que escapen del coliseo romano hacia la ciudad de Pompeya. El primero en llegar gana el juego, y su gladiador tendrá como premio una casa al pie del monte Vesubio.

Todos los gladiadores parten del mismo punto y van recorriendo el mismo camino lineal y sinuoso en un mapa con obstáculos y recompensas hasta llegar al destino final. En el camino, irán adquiriendo el equipamiento necesario para poder llegar vivos a Pompeya y entrar a tomar posesión de esa casa.

El estado de cada jugador está representado por su nivel de equipamiento, su nivel de seniority y de salud, el cual consiste en un número entero. El listado de seniorities está detallado en la tabla 1.

Seniority	Turnos	Plus de energía por turno
Novato	1-8	0
Semi-Senior	8-12	5
Senior	12 en adelante	10

Tabla 1: Tabla de Plus de Energía por Seniority y Turno

En el inicio del juego se arrojará un dado y esto determinará quién empieza el juego, en función del orden en que los jugadores fueron cargados al comienzo. En dicho comienzo cada jugador estará desprovisto de todo equipamiento, su salud será de 20 puntos, su seniority será de Novato y estará en la casilla de partida que representa al Coliseo Romano del cual tiene que huir.

En cada ronda cada jugador que está en condiciones de jugar tira el dado, y avanza tantos casilleros como número obtuvo en el dado. En cada casillero pueden haber distintos premios y obstáculos representados en las tablas 2 y 3 respectivamente.

Hay un máximo establecido de 30 rondas que se pueden jugar, y en caso de que ningún jugador llegue pasadas esas rondas, el juego se finalizará automáticamente y se considerará empatado.

También, si un jugador que ya avanzó casi todo el tablero y al tirar el dado obtiene un número mayor que la distancia que lo separa del casillero de llegada (la ciudad de Pompeya), se considera que efectivamente ha llegado al casillero final, y se procede a validar cuál es el equipamiento del mismo. Si no posee una llave que le permita entrar a la casa del monte vesubio, entonces dicho

Premio	Qué sucede	Impacto
Comida	El jugador encuentra una tabla de sushi	La energía se ve incrementada en 15 puntos
Equipamiento	Mejora su equipamiento	En funcion de lo que tenga de equipamiento, es lo que recibe:
		(0) nada: Recibe Casco
		(1) Casco: Recibe Armadura
		(2) Armadura: Recibe Espada y Escudo
		(3) todo lo anterior: Recibe La Llave
		(4) La Llave: sigue con La Llave

Tabla 2: Tabla de premios

Obstáculo	Qué sucede	Impacto
Asiste a un Bacanal	El jugador tira un dado para determinar la cantidad de copas de vino a tomar	Saca 4 puntos de energía por cada trago tomado
Fiera salvaje hambrienta	Se desata una pelea. Se resta energía según el equipamiento que tenga el gladiador	Energía perdida según equipamiento:
		Nada: -20
		Casco: -15
		Casco y Armadura: -10
		Casco, Armadura, Escudo y espada: -2
		Llave de la casa: 0
Lesión	El gladiador se enoja con la vida, patea una piedra y debe esperar un turno sin avanzar	El turno siguiente no avanza ni tira el dado

Tabla 3: Tabla de Obstáculos

jugador retrocede hasta el casillero de la mitad del tablero. Si, en cambio, posee una llave que le permita entrar a dicha casa, entonces ese jugador ha ganado la partida.

4. Modelo de dominio

El modelo de dominio de la aplicación se basa en la gestión de un juego, con la interacción de elementos como jugadores, tablero, dados, reglas (ver especificaciones funcionales), obstáculos, premios, y la interfaz de usuario. A continuación, se presenta una descripción general del modelo de dominio basado en los paquetes y clases:
Model (Modelo): Contiene las clases que representan los elementos fundamentales del juego, como el estado del juego, el estado del jugador, la jerarquía de jugadores, el tablero, los obstáculos, los premios, y las reglas del juego.
UI (Interfaz de Usuario): Contiene las clases relacionadas con la interfaz de usuario, incluyendo vistas para la presentación del juego, pantalla de inicio, pantalla de finalización, vista de instrucciones, y otras vistas relacionadas con la interfaz de usuario.

5. Diagrama de Paquetes

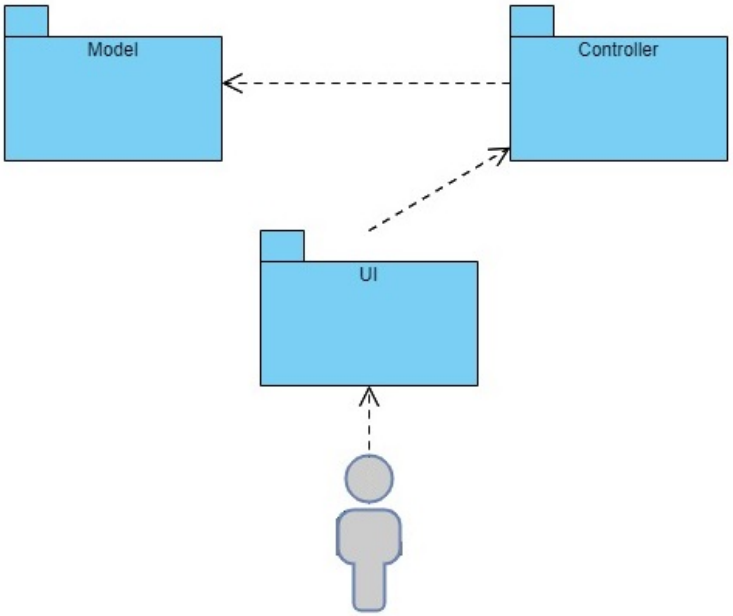

Figura 1: Diagrama de paquetes general.

6. Diagramas de clase

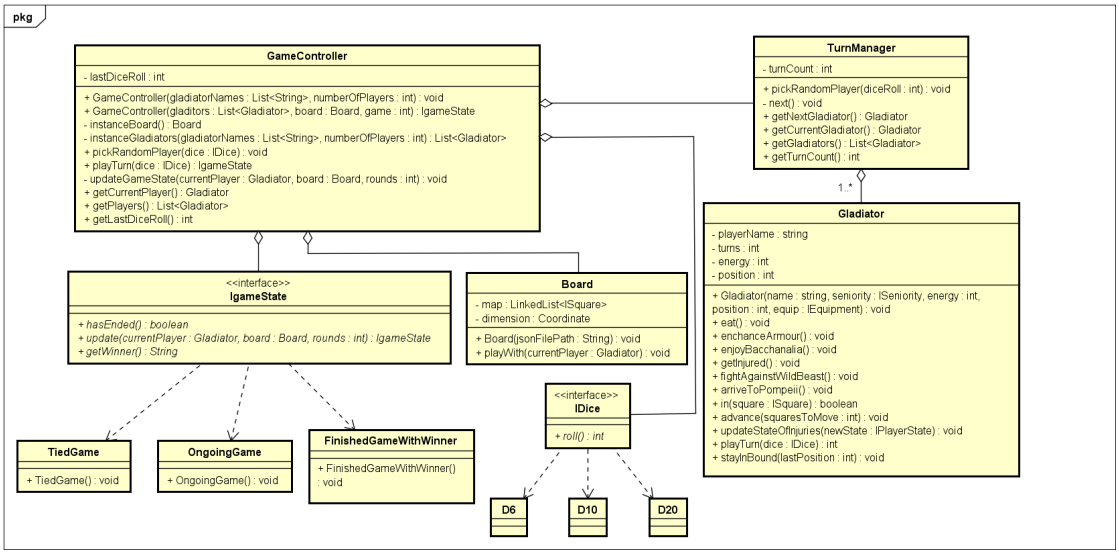

Figura 2: Diagrama de clase general.

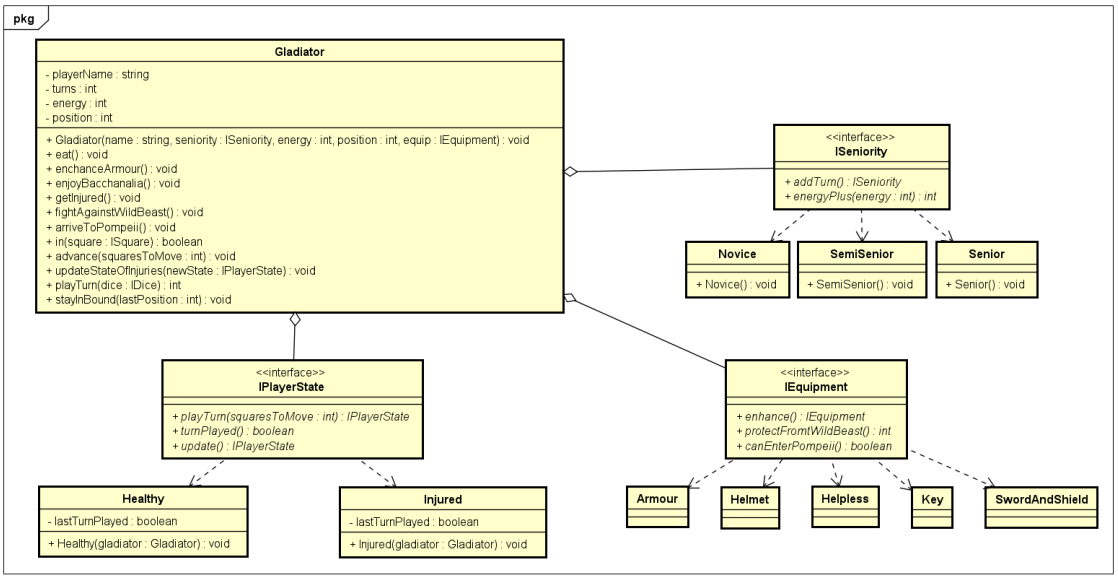

Figura 3: Diagrama de la clase Gladiator.

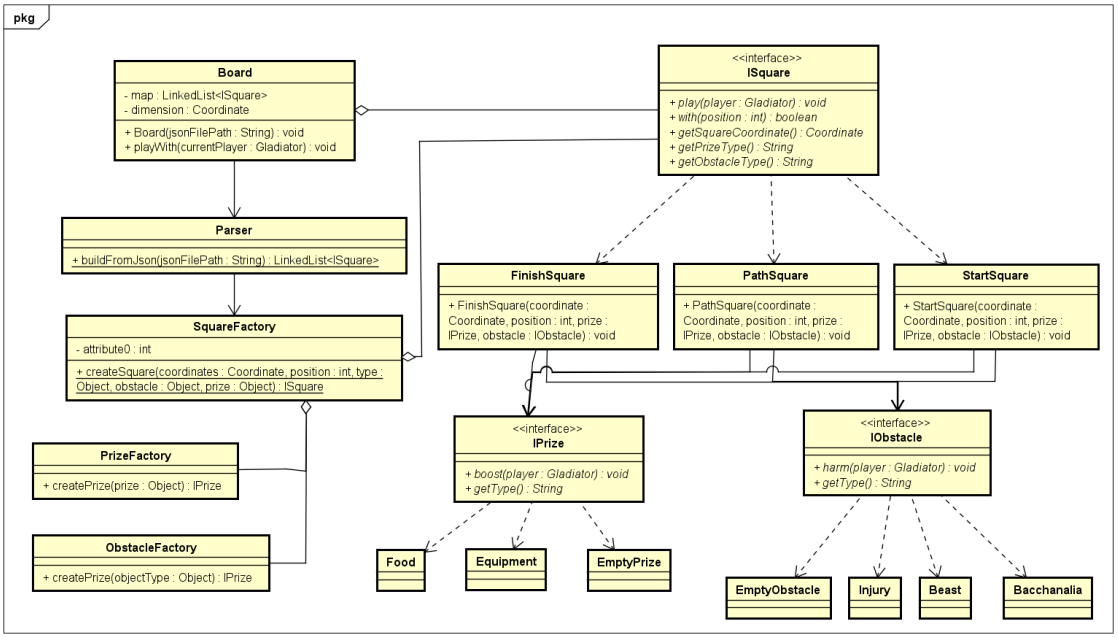

Figura 4: Diagrama de la clase Board.

7. Detalles de implementación

Para desarrollar la aplicación acorde al paradigma de programación orientada a objetos el programa a primera vista la aplicación está organizada en un patrón Modelo-Vista-Controlador, cada uno de estos componentes tiene responsabilidades específicas, lo que permite una mejor organización del código.
El Modelo representa los datos y la lógica de negocio de la aplicación. En el contexto del juego, el Modelo estaría compuesto por las clases relacionadas con la gestión del mismo, como el estado

del juego, el estado del jugador, la jerarquía de jugadores, el tablero, los obstáculos, los premios, y las reglas del juego. Estas clases encapsulan la lógica del juego y los datos subyacentes.
La Vista se encarga de presentar la interfaz de usuario y de mostrar los datos al usuario. En el contexto de la aplicación del juego, las vistas incluirían las diferentes pantallas y elementos visuales con los que el usuario interactúa, como la pantalla de inicio, la pantalla de juego, la pantalla de finalización, y otras vistas relacionadas con la interfaz de usuario.
El Controlador actúa como intermediario entre el Modelo y la Vista. Se encarga de manejar las interacciones del usuario, procesar las entradas, actualizar el modelo según las acciones del usuario, y por tanto actualizar la vista.

7.1. Modelo

El Modelo de dominio, en particular, está organizado en 5 packages principales: -

1. **attributes**, que tiene la finalidad de contener a todas las entidades de los atributos del juego, como ser las de los sub-packages: gameState, playerState, seniority y coordinate.
   - **GameState**, representa el estado interno del juego, este fue modelado mediante el patrón de diseño state. La interfaz IGameState establece cómo tiene que ser el comportamiento de cada uno de los estados del juego, y las clases FinishedGameWithWinner, OngoingGame y TiedGame implementan dicha interfaz y representan cada uno de los estados posibles. Estos estados dan al juego la posibilidad de continuar desarrollando partidas y tirando los dados cada jugador, o bien indican cuál es el estado del juego brindando los detalles acordes.
   - **playerState**, consiste en la abstracción correspondiente al estado interno del jugador. La interfaz IPlayerState modela el comportamiento y el alcance funcional de cada estado posible para cada jugador, y las clases Healthy e Injured implementan dicha interfaz, las cuales determinan principalmente si el jugador puede jugar o no en cada partida.
   - **seniority**, modela los distintos rangos de experiencia de los gladiadores, lo cual afecta en la cantidad de puntos obtenidos en cada partida, y sigue el mismo estilo de diseño mediante el patrón state. La interfaz ISeniority es la que brinda el comportamiento que debe cumplir cada implementación. En este caso, dichos estados posibles son los de las siguientes clases: Novice, SemiSenior, y Senior.
   - **Coordinate**, por su parte, es la clase que modela cada una de las ubicaciones bidimensionales para cada uno de los casilleros que puedan haber en el tablero.

2. **board**, es el package que modeliza todo lo relativo al tablero de juego, dividido también en 5 sub-packages (factory, obstacles, prizes, squares y board) con distintos roles cada uno:
   - **prizes**, es un sub-package cuya misión es, mediante el patrón Strategy, define una familia de comportamientos, se encapsula cada uno de ellos y se los hace intercambiables, de modo que hay premios diferentes en distintas casillas. La interfaz central para definir esta familia de premios es IPrize, y una serie de comportamientos completa esta familia para tres tipos de situaciones: el jugador tiene la dicha de disfrutar un plato de sushi; el jugador mejora su equipamiento en función de su equipamiento actual; y el jugador no tiene ningún premio en la casilla actual.
   - **obstacles**, es un sub-package que en analogía a prizes, tiene la misión de proveer mediante el patrón Strategy una familia de comportamientos, y una interfaz que los define. La interfaz en este caso es IObstacle, y los obstáculos existenes ya definidos son: El jugador pasa una noche de alcohol que le quita energía proporcional a los tragos tomados; el jugador enfrenta una temible bestia que pone en juego su nivel de energía en función de su equipamiento; el jugador pierde su próximo turno; y el jugador no tiene ningún obstáculo en la casilla actual.
   - **squares**, es uno de los sub-packages centrales del tablero, ya que modeliza la entidad en la que cada uno de los gladiadores juega su turno. Para esto, se dispone de una interfaz ISquare, y las siguientes implementaciones: StartSquare, PathSquare y FinishSquare. Se sigue un patrón Strategy para cada uno de los ISquare posibles, y a su vez se implementa un patrón

de double dispatch en el método play, el cual tiene dos niveles de despacho para determinar cuál método debe ser ejecutado. El primer nivel de despacho, selecciona un método basado en el tipo estático del receptor del mensaje. El receptor del mensaje es delegado hacia las clases de las interfaces IObstacle e IPrize, con lo cual en función del tipo estático se abren las posibilidades para el jugador. A su vez, cada premio y cada obstáculo tiene la función realizar el segundo nivel de despacho dinámicamente a través de distintos métodos propios al gladiador.
- **factory**, es un conjunto de clases implementadas en el patrón de diseño Factory Method. Cada una de las fábricas (SquareFactory, ObstacleFactory, PrizeFactory) tienen un método acorde que permite crear la entidad deseada en tiempo de ejecución, adjudicándole una posición en el tablero, una posición relativa respecto de las demas casillas. Pero en especial, la clase SquareFactory implementa también un patrón de diseño de double dispatch, esto se debe a que la resolución del método a implementar depende de los tipos de dos objetos en lugar de uno. En este caso tenemos el primer nivel de despacho a la creación del SquareFactory por el patrón Strategy, y en un segundo nivel a las definiciones por parte de ObstacleFactory y PrizeFactory.
- **Board** consisten en una clase suelta en el package, la cual tiene por función la de modelar la entidad del tablero de juego, gestionar el juego de los gladiadores en la correspondiente casilla. En el constructor se utiliza inyección de dependencias para favorecer la modularidad, los testeos unitarios y el mantenimiento del código. - **Parser**, es un intérprete de archivos json adaptado al modelo de dominio, de modo de leer archivos json con una sintaxis específica que detalla todas las características del tablero, como ser los obstáculos, premios y posición de cada una de las casilla que lo conforman. Esta clase trabaja en servicio de la clase Board, y en conjunto con el package Factory.

3. **equipment**, es un sub-package que también implementa el patrón de diseño State, de manera de poder brindar la versatilidad de que el gladiador vaya mejorando su equipamiento a lo largo de las distintas partidas. La interfaz que delimita el comportamiento que debe tener todo equipamiento es *IEquipment*, y los distntos estados posibles son los indicados en la Tabla 2.

4. **GameController**, es una clase central del modelo de dominio del juego. Se encarga de manejar la lógica del juego, incluyendo la gestión del tablero, el estado del juego, el turno de los jugadores y el lanzamiento de dados. Cada una de estas funcionalidades recién mencionadas son paquetes, subpaquetes o clases analizadas en otros items de esta sección, muchas de las cuales están inyectadas para bajar el acoplamiento al mínimo posible. Todas estas características hacen que GameController sea el controlador del modelo de dominio y el corazón del mismo.

5. **Gladiator**, representa en el modelo de dominio del juego a la entidad Jugador, y por lo tanto, su importancia es clara y su comportamiento acorde para permitir el movimiento del jugador y poder interactuar con cada beneficio y obstáculo del juego, actualizando su estado y energía a lo largo del juego.

6. **dice**, dice es un sub-package del modelo que concentra una interfaz de dado IDice, y diversas implementaciones. Su utilidad radica en que los avances son determinados aleatoriamente en función del número arrojado por el dado en cada partida de cada jugador.

7. **TurnManager**, por último, esta clase es la responsable de gestionar el turno de los jugadores en cada juego. Para ello utiliza un iterador sobre el listado de jugadores, y se encarga de indicar quién es el jugador siguiente de cada partida.

7.2. Vista

El Package UI representa el componente visual de la aplicación, y como tal está subdividido en diversas clases, una por cada vista que hay en el programa.

7.3. Controlador

La clase Controller actúa como el controlador principal para el juego, gestionando la lógica del juego, la interfaz de usuario y la reproducción de música, la cual está delegada en un controlador específico de música, dentro del mismo package.

8. Excepciones

InvalidJSonException Esta excepción en lanzada por el intérprete de archivos Json, en caso de que no se consiga generar una instancia de la clase Board.

InvalidInputException Esta excepción es lanzada por el controlador en caso de que se realice un ingreso inválido en las pantallas iniciales del programa, sea que se ingrese un número fuera del rango 2-6 o que los nombres de los gladiadores no cumplen con la pauta establecida por las especificaciones.

9. Diagramas de secuencia

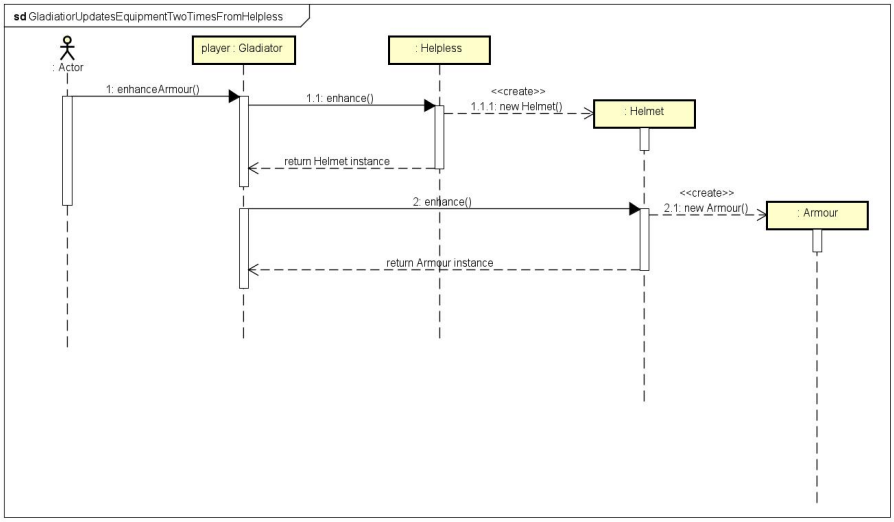

Figura 5: Un gladiador actualiza su equipamiento dos veces.

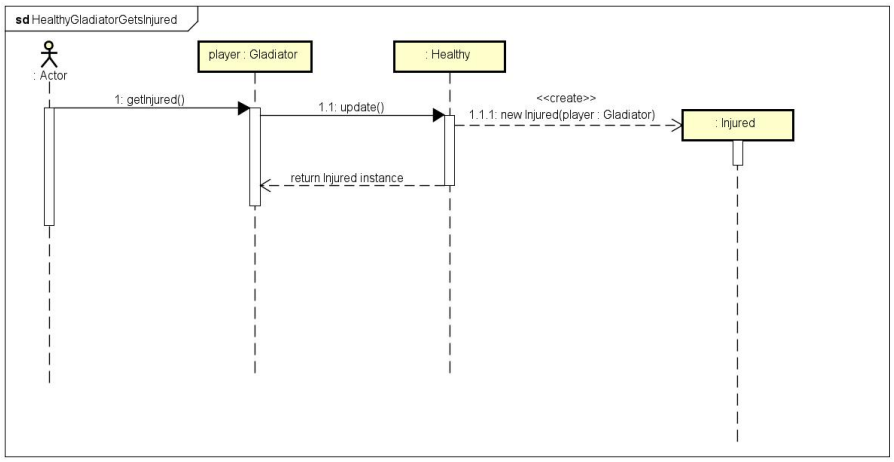

Figura 6: Un gladiador sano se lastima.

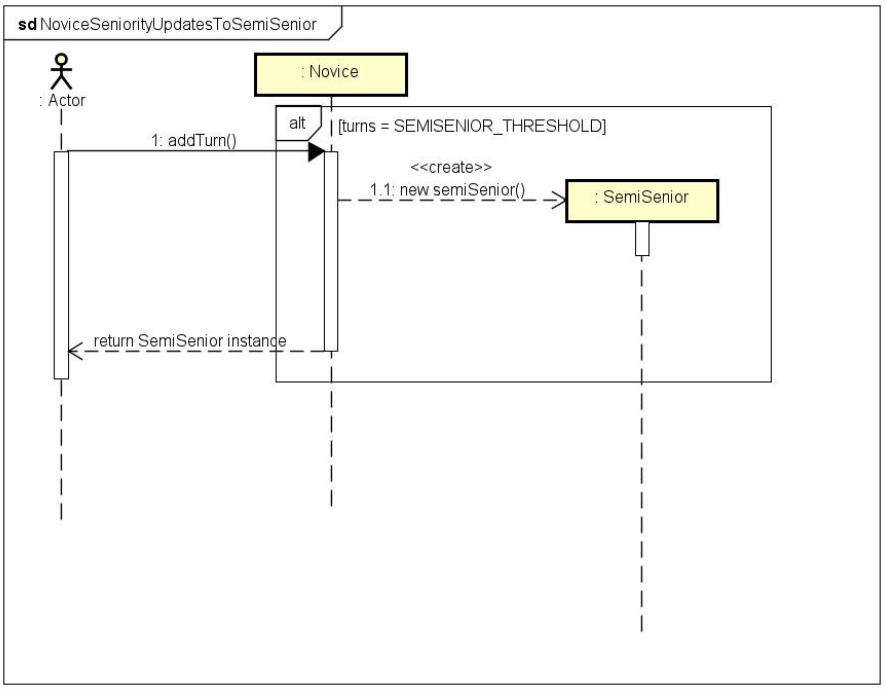

Figura 7: Seniority novice se actualiza a semi-senior.

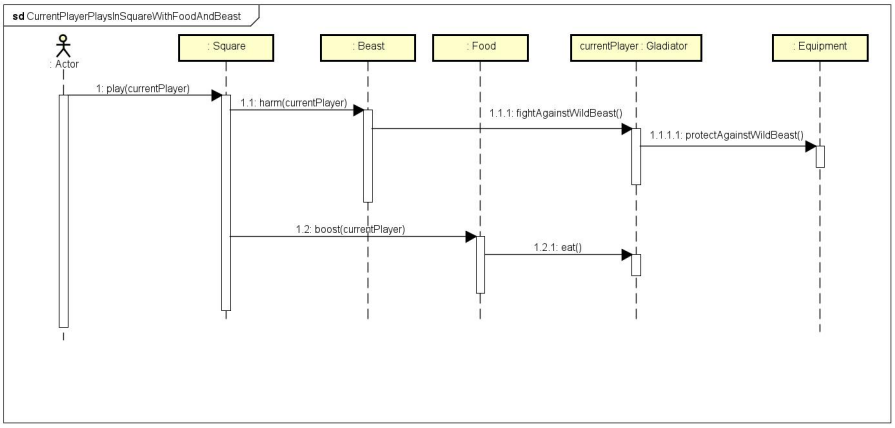

Figura 8: Un jugador cae en un casillero con comida y fiera salvaje.

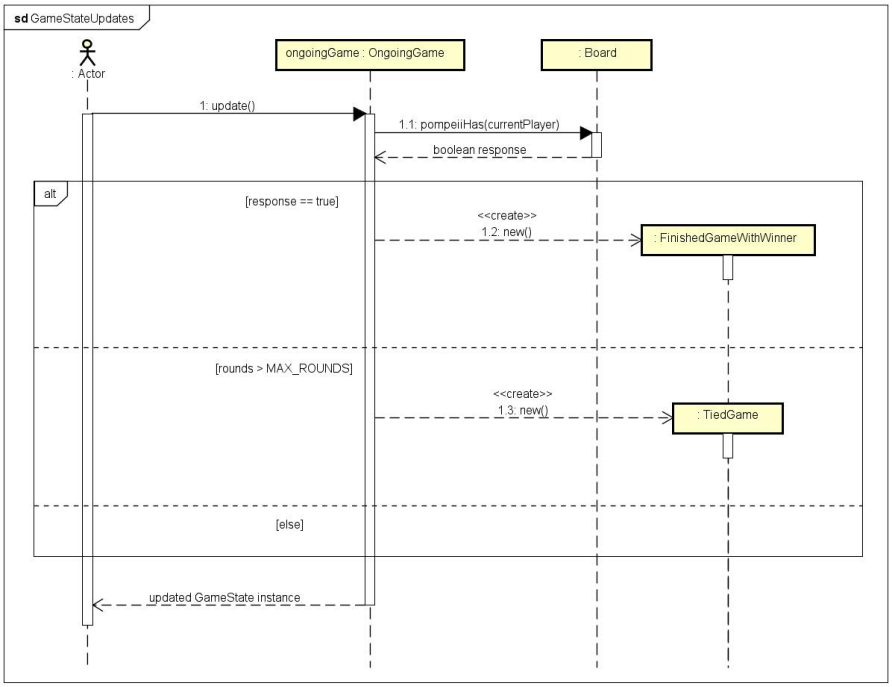

Figura 9: Se actualiza el estado del juego.

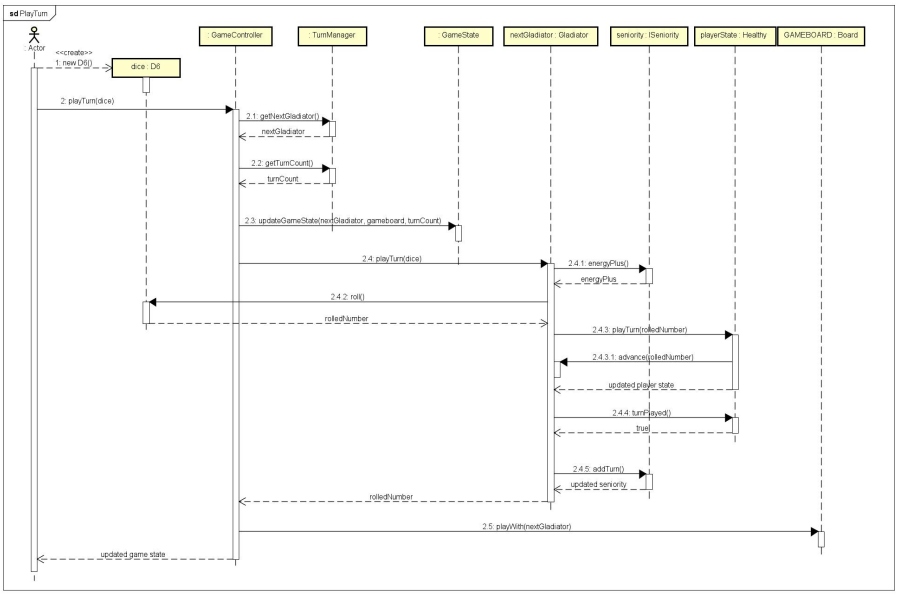

Figura 10: Metodo playTurn() de la clase GameController.

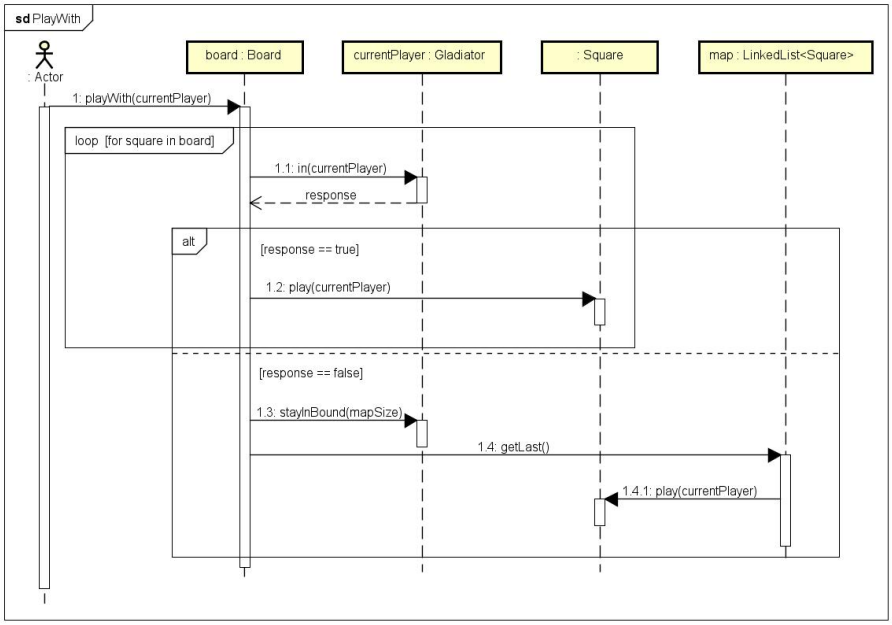

Figura 11: Metodo playWith() de la clase Board.

10. Diagramas de estado

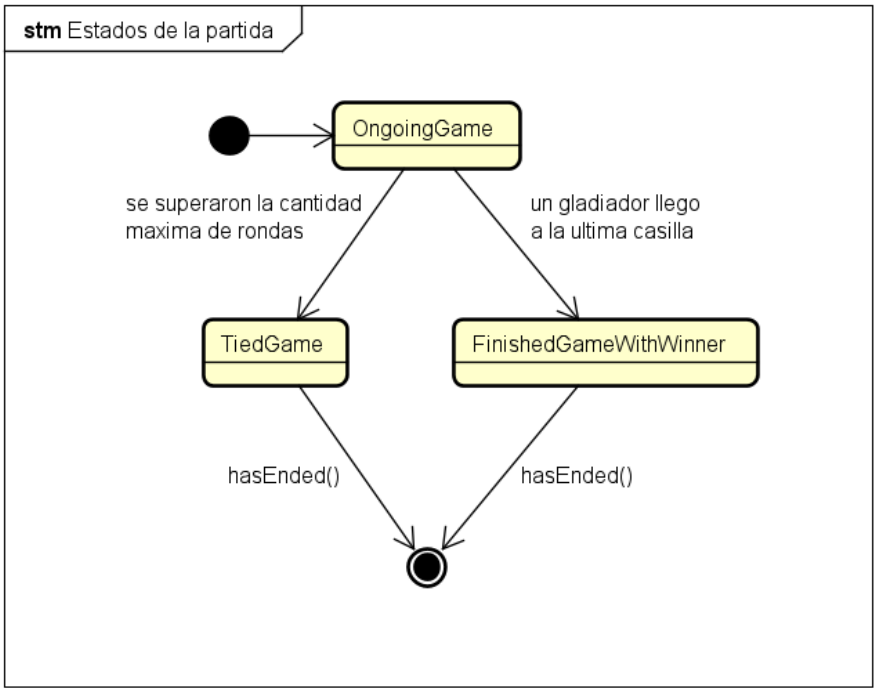

Figura 12: Estados de la partida.

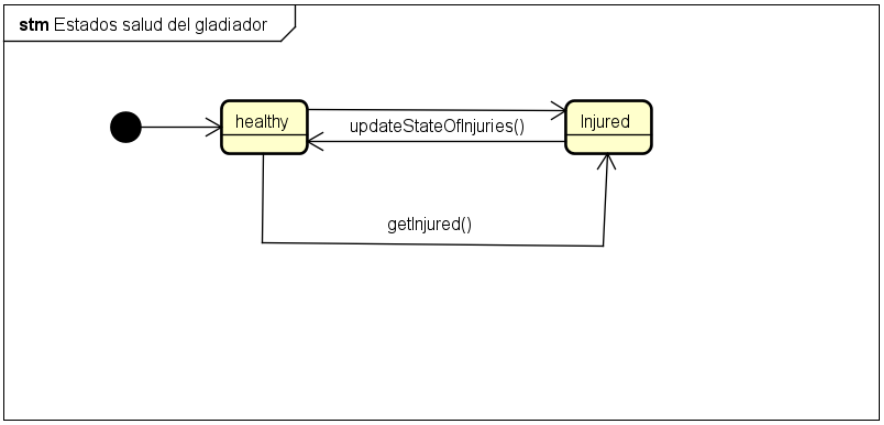

Figura 13: Estados de la salud de un gladiador.