[75.07 / 95.02]

Algoritmos y programación III

Trabajo práctico 2: TowerDefense

Estudiantes:

Nombre	Padrón	Mail
Julio Piñango	105867	jpinango@fi.uba.ar
Juan Cruz Diez	109882	jdiez@fi.uba.ar
Mariano Ruggeri	107042	marianoruggeri0403@gmail.com
Manuel Perez	108192	manuperez@fi.uba.ar
Julianna Sánchez	109621	jsanchezm@fi.uba.ar

Tutora: Maia Naftali

Nota final:

Índice

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de manera grupal aplicando todos los conceptos vistos en el curso, utilizando un lenguaje de tipado estático (Java) con un diseño del modelo orientado a objetos y trabajando con las técnicas de TDD e Integración Continua.

2. Supuestos

-Sólo puede haber una torre por parcela.
-Los enemigos cuando llegan al final de la pasarela causan daño y dan créditos al jugador.
-Al matar la hormiga 11, independientemente si ya estaba en la pasarela o es nueva, esta le otorga 2 de crédito al jugador.
-Se pueden crear varias defensas por turno.
-Se puede poner una trampa arenosa sobre otra.
-La lechuza no da créditos al jugador.
-El topo no da créditos al jugador.
-La trampa arenosa solo afecta a los enemigos que queden en esa misma posición después del turno, es decir, que si un enemigo en un turno pasa por la trampa pero no cae en esa posición específica, la trampa no lo afecta.
-Cuando la hormiga cae en una trampa arenosa, se queda atorada hasta que se elimine la trampa.

3. Diagramas de clases

Diagrama de clase principal: incluye las clases generales del trabajo.

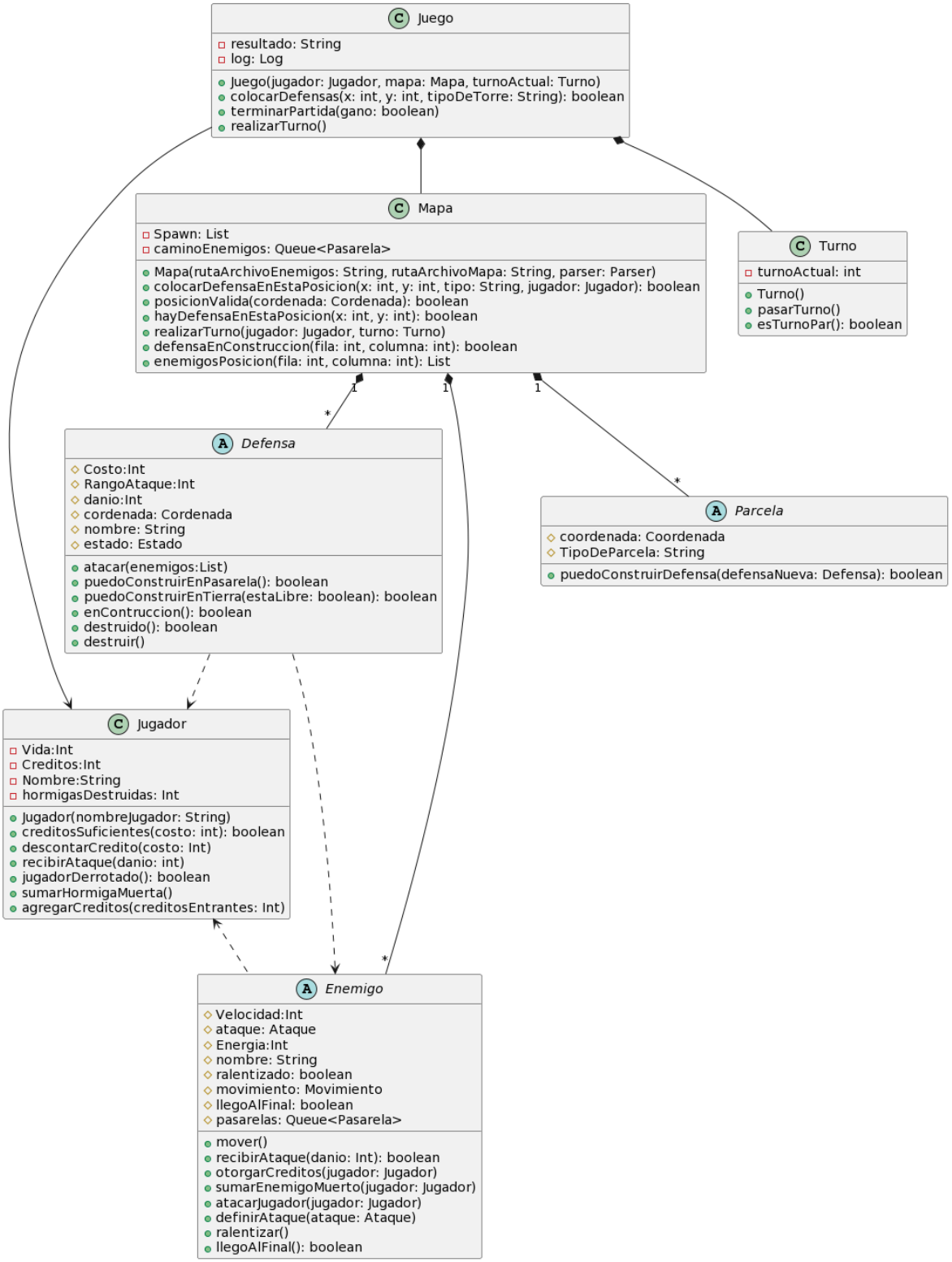
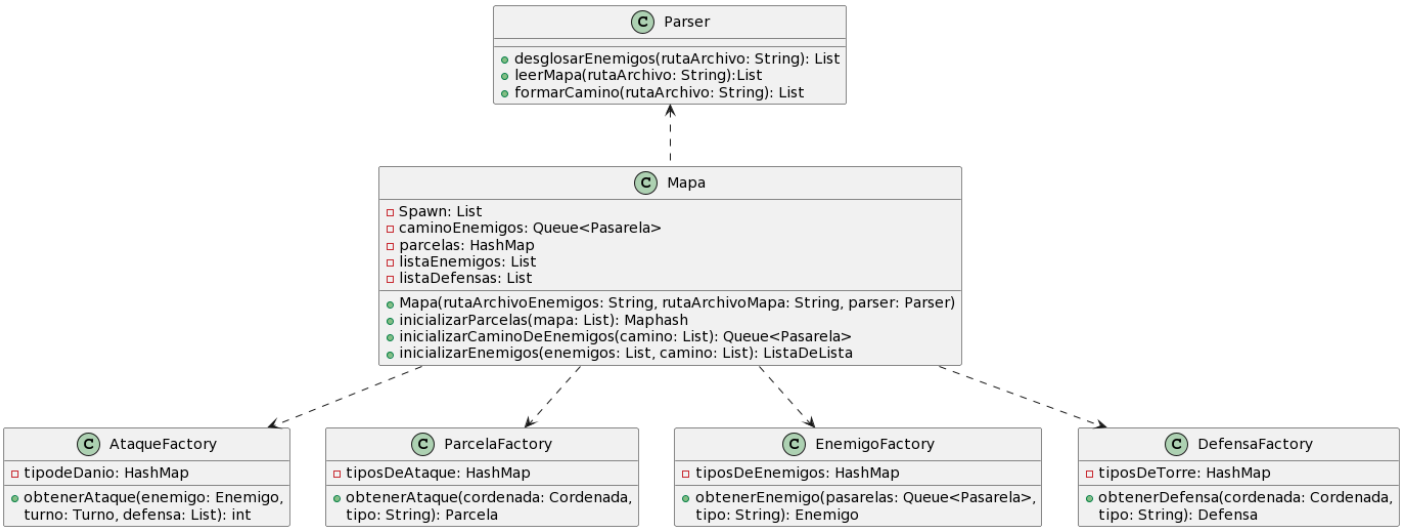

Juego
- resultado: String
- log: Log
+ Juego(jugador: Jugador, mapa: Mapa, turnoActual: Turno)
+ colocarDefensas(x: int, y: int, tipoDeTorre: String): boolean
+ terminarPartida(gano: boolean)
+ realizarTurno()
Mapa
- Spawn: List
- caminoEnemigos: Queue<Pasarela>
+ Mapa(rutaArchivoEnemigos: String, rutaArchivoMapa: String, parser: Parser)
+ colocarDefensaEnEstaPosicion(x: int, y: int, tipo: String, jugador: Jugador): boolean
+ posicionValida(cordenada: Cordenada): boolean
+ hayDefensaEnEstaPosicion(x: int, y: int): boolean
+ realizarTurno(jugador: Jugador, turno: Turno)
+ defensaEnConstruccion(fila: int, columna: int): boolean
+ enemigosPosicion(fila: int, columna: int): List
Turno
- turnoActual: int
+ Turno()
+ pasarTurno()
+ esTurnoPar(): boolean
1
1
1
*
*
Defensa
Costo:Int
RangoAtaque:Int
danio:Int
cordenada: Cordenada
nombre: String
estado: Estado
+ atacar(enemigos:List)
+ puedoConstruirEnPasarela(): boolean
+ puedoConstruirEnTierra(estaLibre: boolean): boolean
+ enContruccion(): boolean
+ destruido(): boolean
+ destruir()
Parcela
coordenada: Coordenada
TipoDeParcela: String
+ puedoConstruirDefensa(defensaNueva: Defensa): boolean
Jugador
- Vida:Int
- Creditos:Int
- Nombre:String
- hormigasDestruidas: Int
+ Jugador(nombreJugador: String)
+ creditosSuficientes(costo: int): boolean
+ descontarCredito(costo: Int)
+ recibirAtaque(danio: int)
+ jugadorDerrotado(): boolean
+ sumarHormigaMuerta()
+ agregarCreditos(creditosEntrantes: Int)
*
Enemigo
Velocidad:Int
ataque: Ataque
Energia:Int
nombre: String
ralentizado: boolean
movimiento: Movimiento
llegoAlFinal: boolean
pasarelas: Queue<Pasarela>
+ mover()
+ recibirAtaque(danio: Int): boolean
+ otorgarCreditos(jugador: Jugador)
+ sumarEnemigoMuerto(jugador: Jugador)
+ atacarJugador(jugador: Jugador)
+ definirAtaque(ataque: Ataque)
+ ralentizar()
+ llegoAlFinal(): boolean

Diagrama del Parser: incluye las relaciones que existen desde el Parser.



Diagramas de las clases abstractas.

Diagrama de Parcela

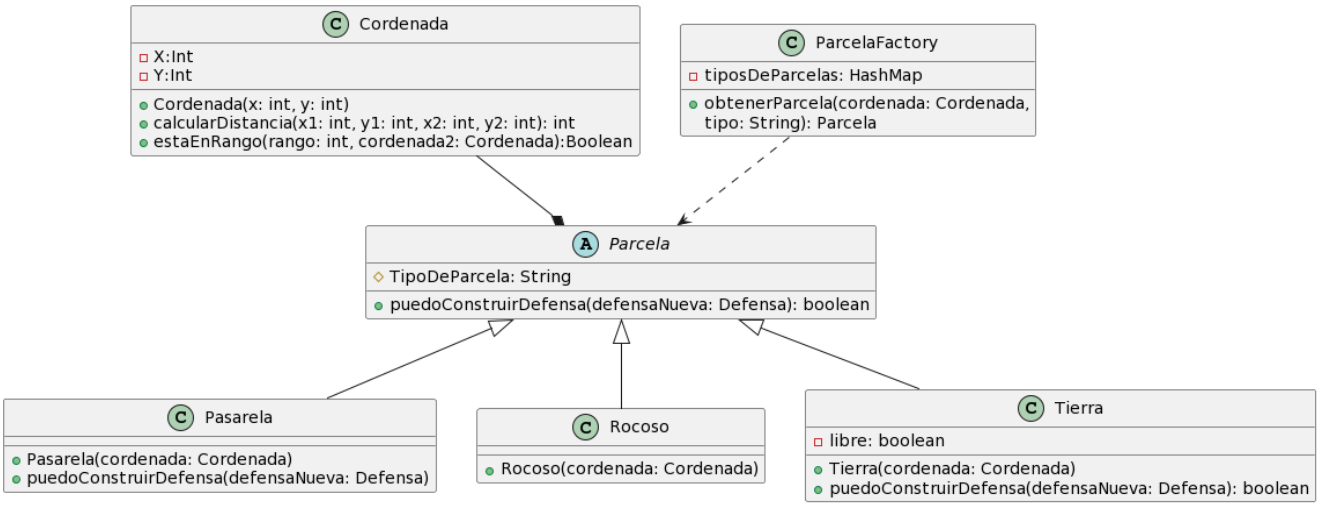
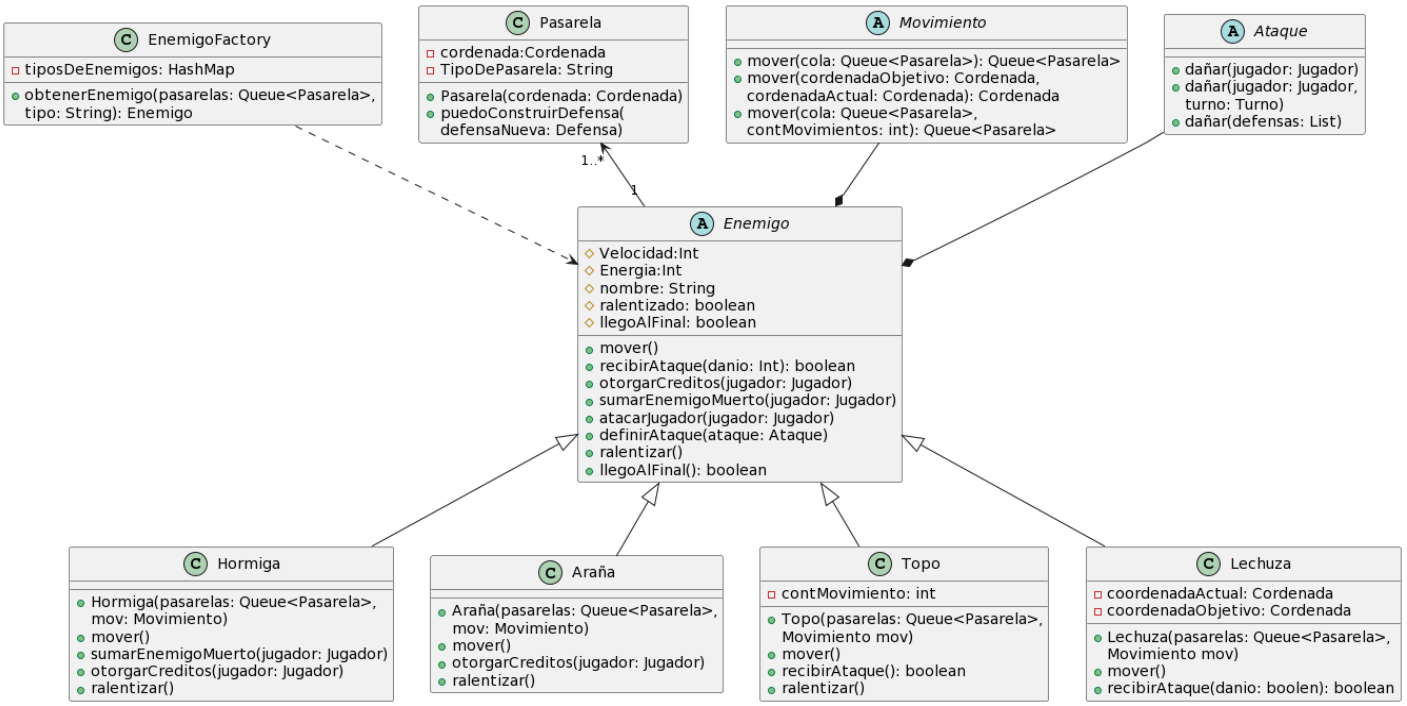
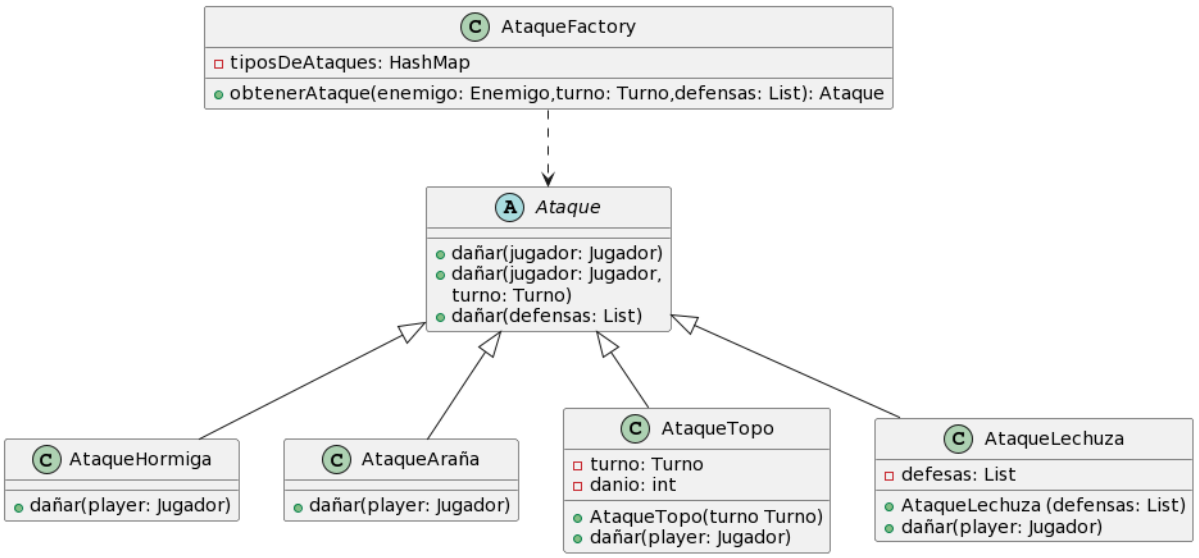


Diagrama Enemigo



Ataque del Enemigo



Movimiento del Enemigo

Está dividido en dos diagramas para que sea más legible.

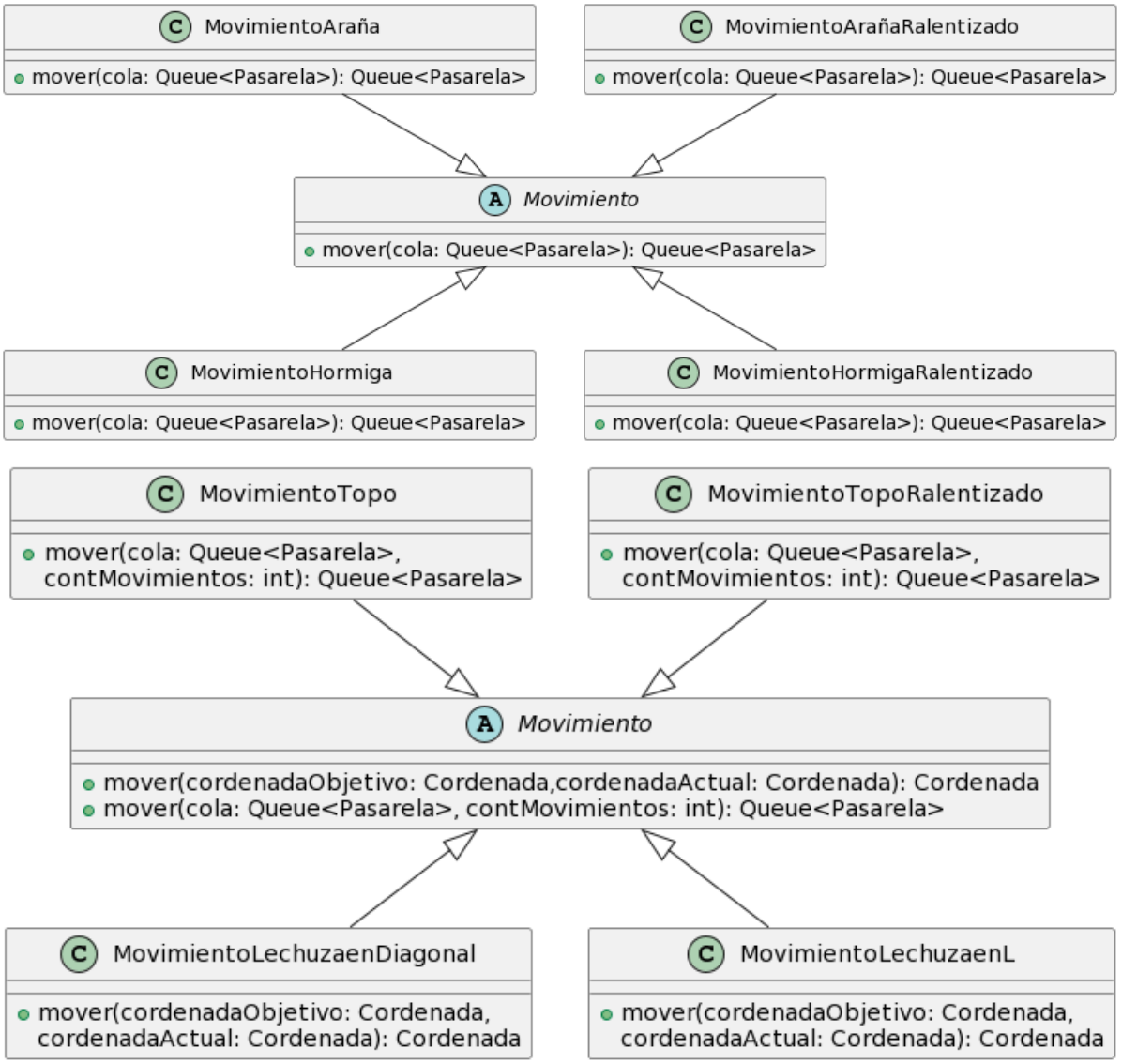
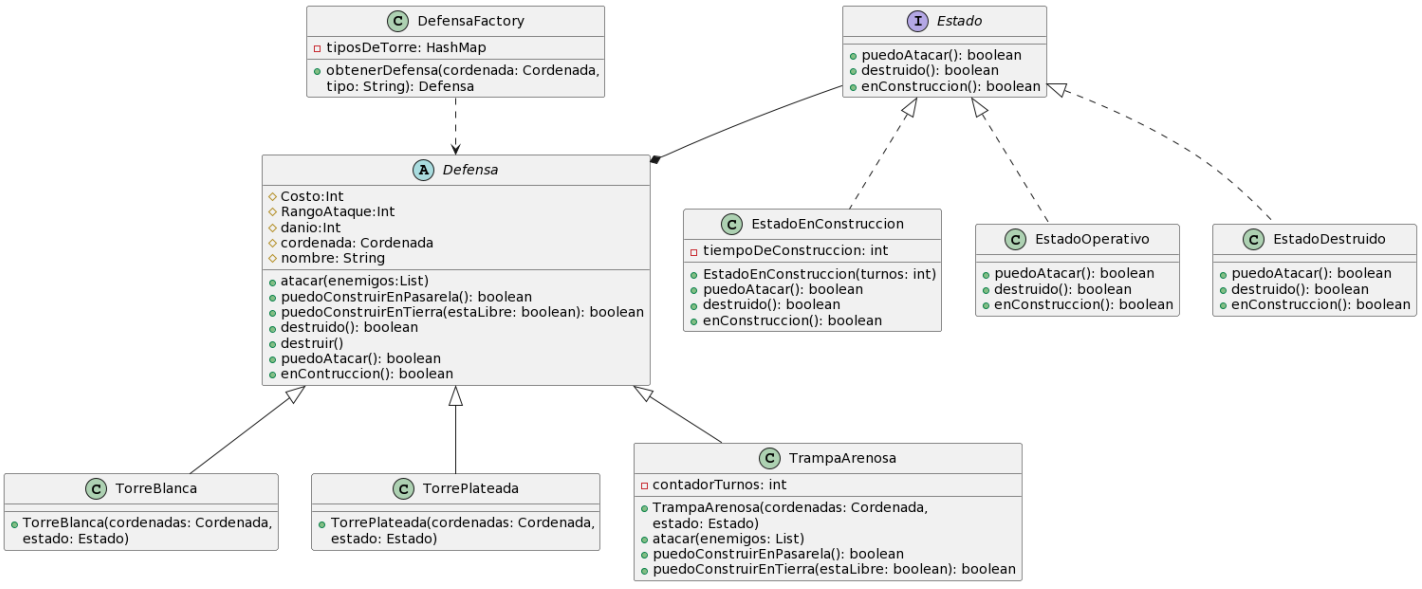
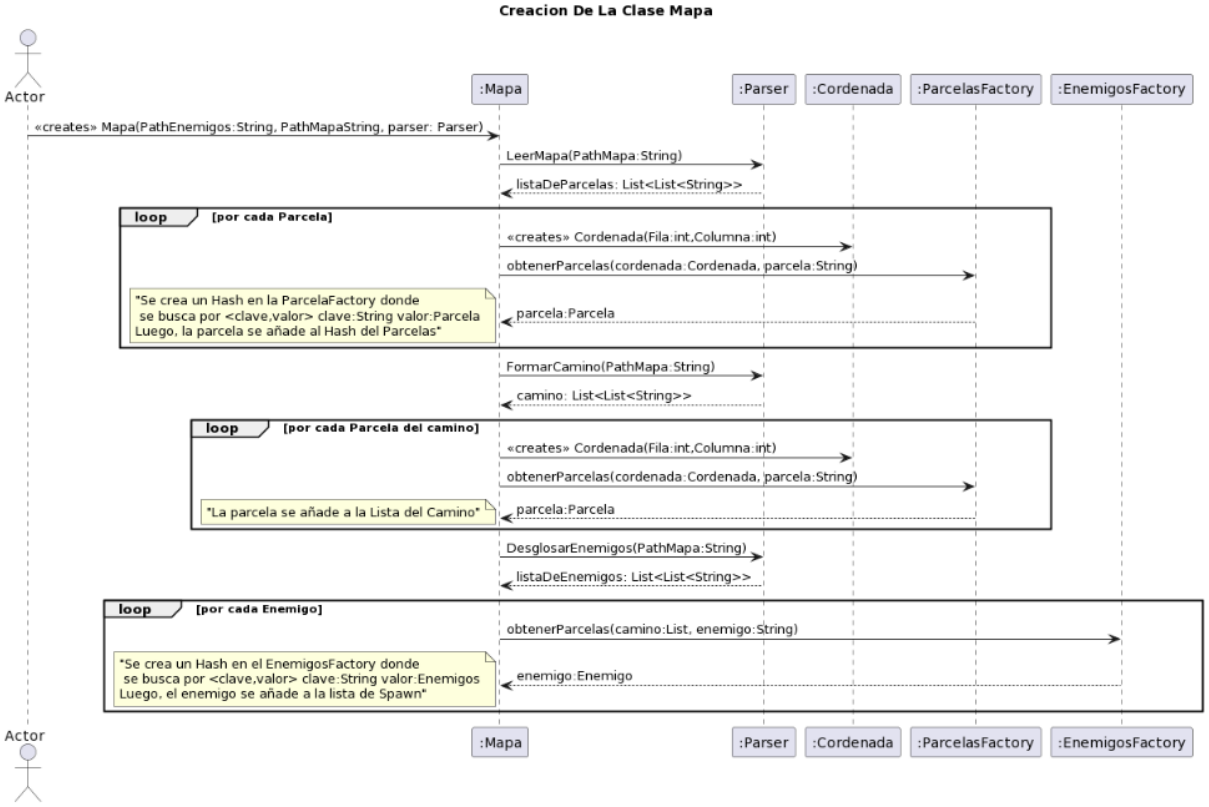
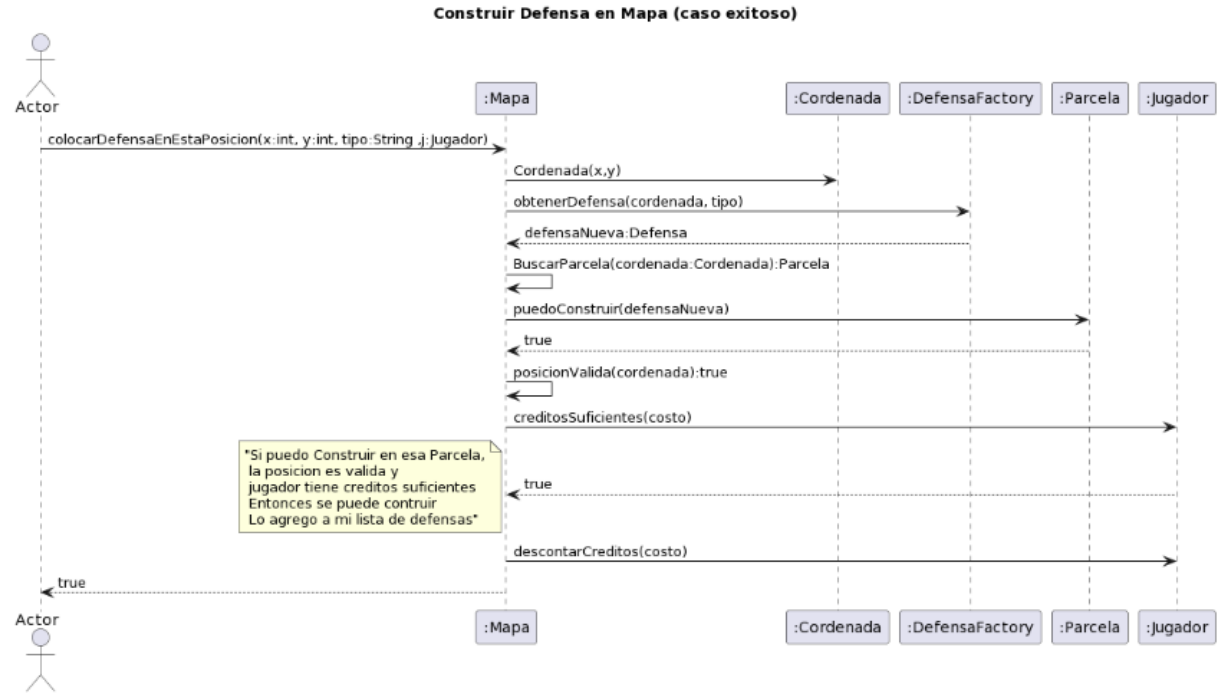


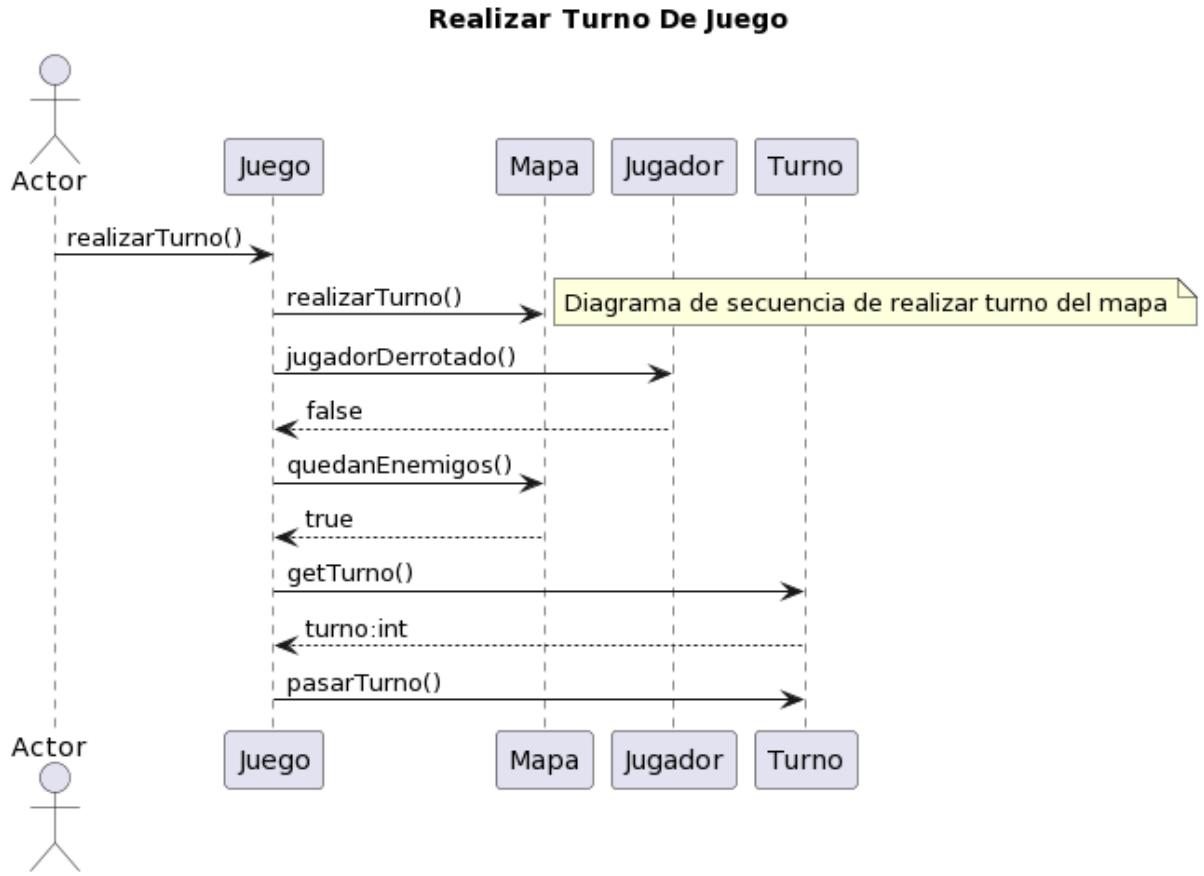
Diagrama de Defensa

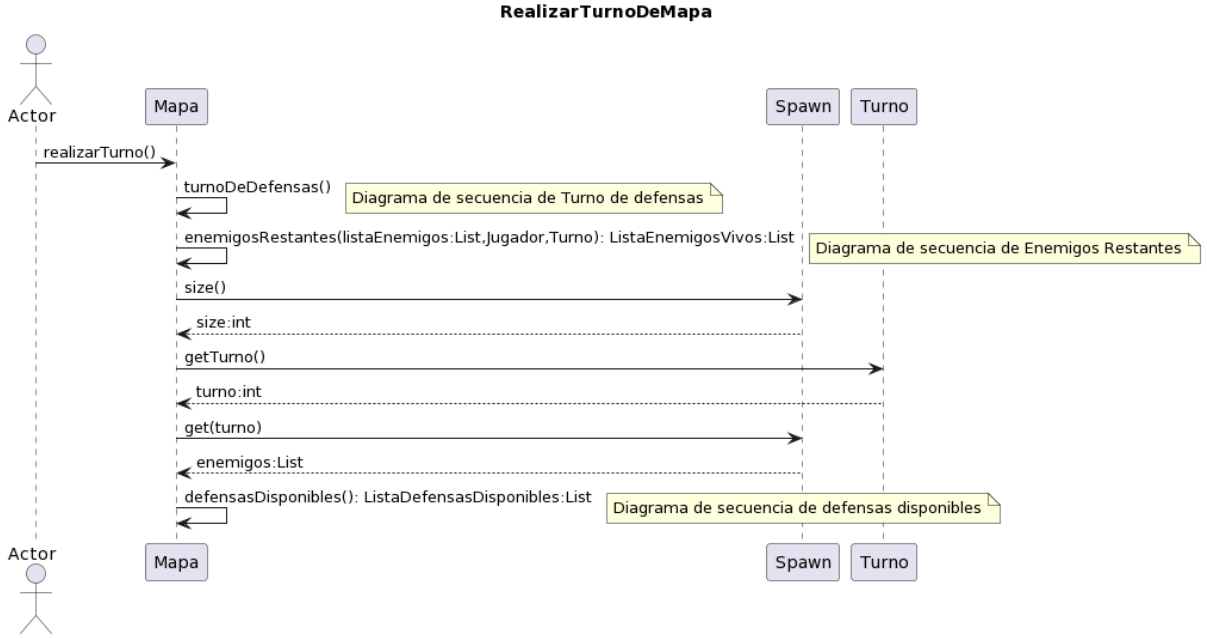


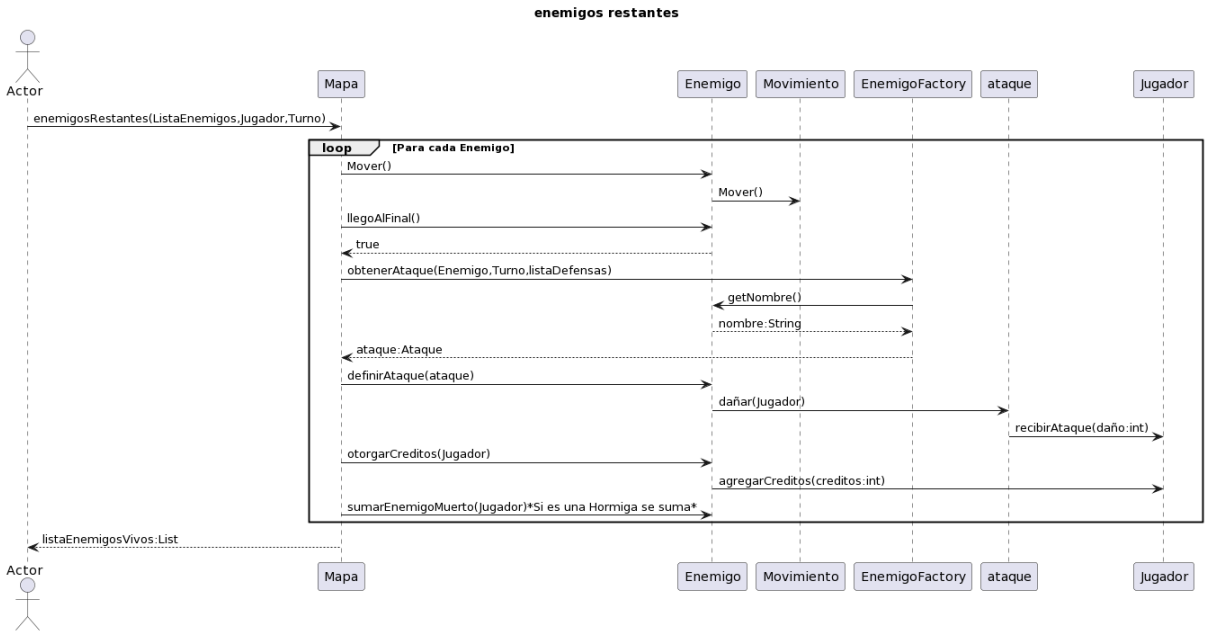
4. Diagramas de secuencia

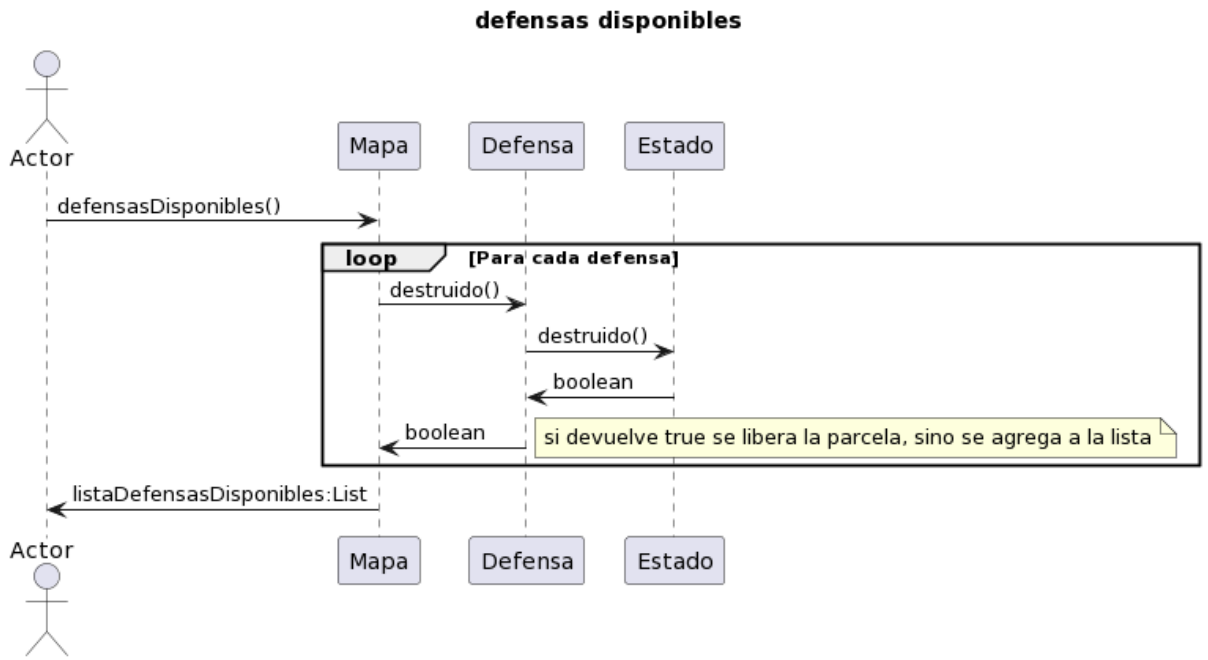


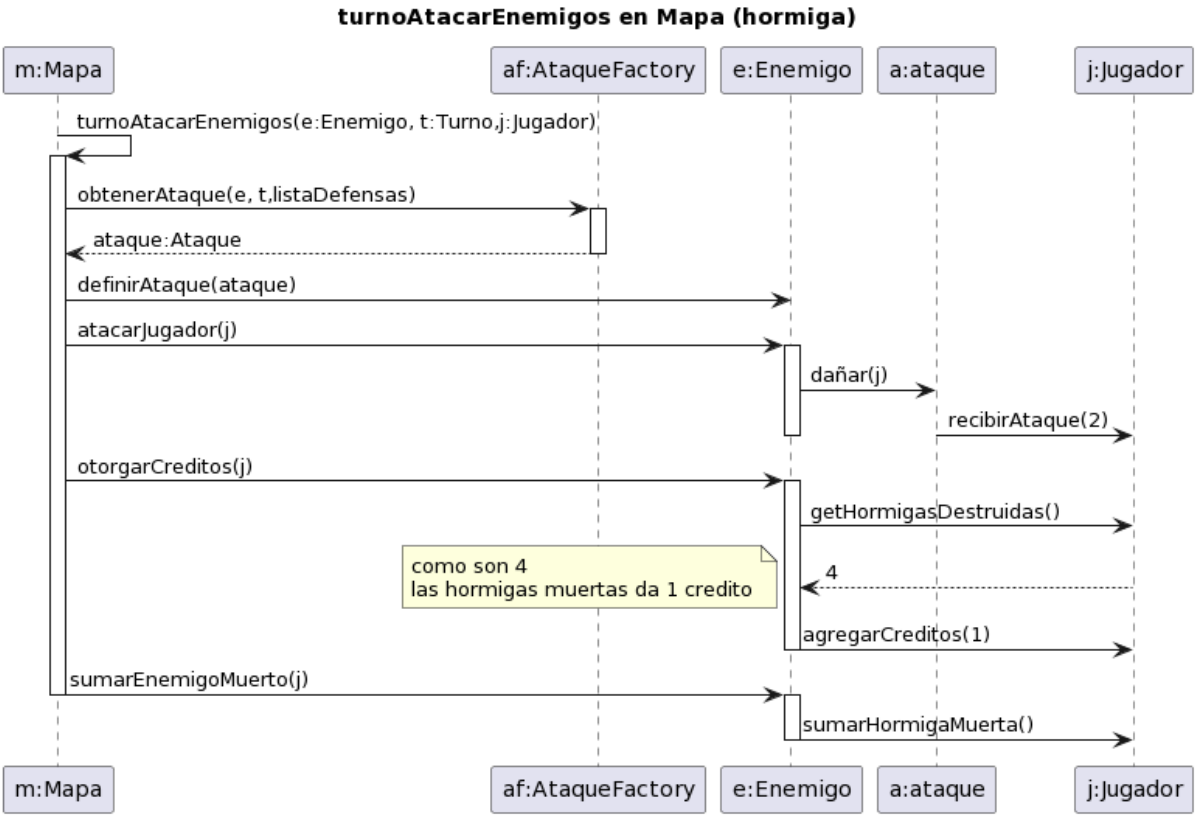


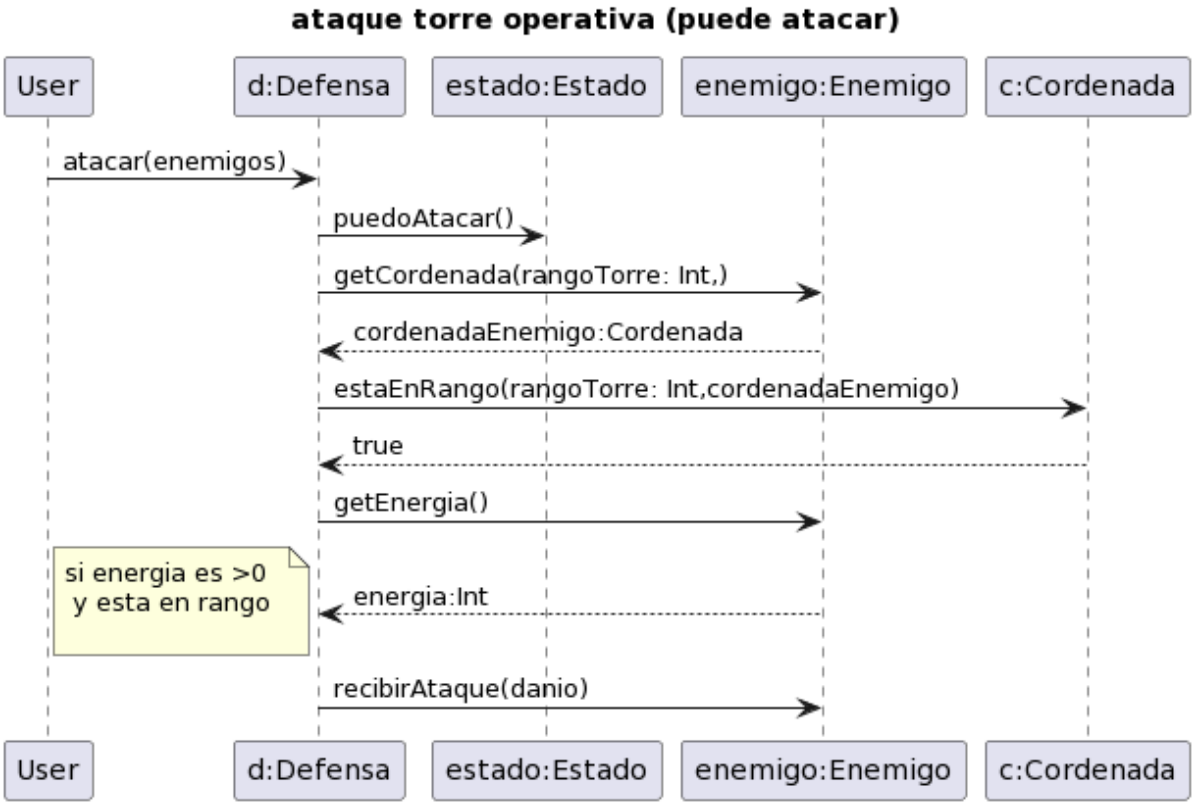

Realizar Turno De Juego
Actor
Juego
Mapa
Jugador
Turno
realizarTurno()
realizarTurno()
Diagrama de secuencia de realizar turno del mapa
jugadorDerrotado()
false
quedanEnemigos()
true
getTurno()
turno:int
pasarTurno()
Actor
Juego
Mapa
Jugador
Turno

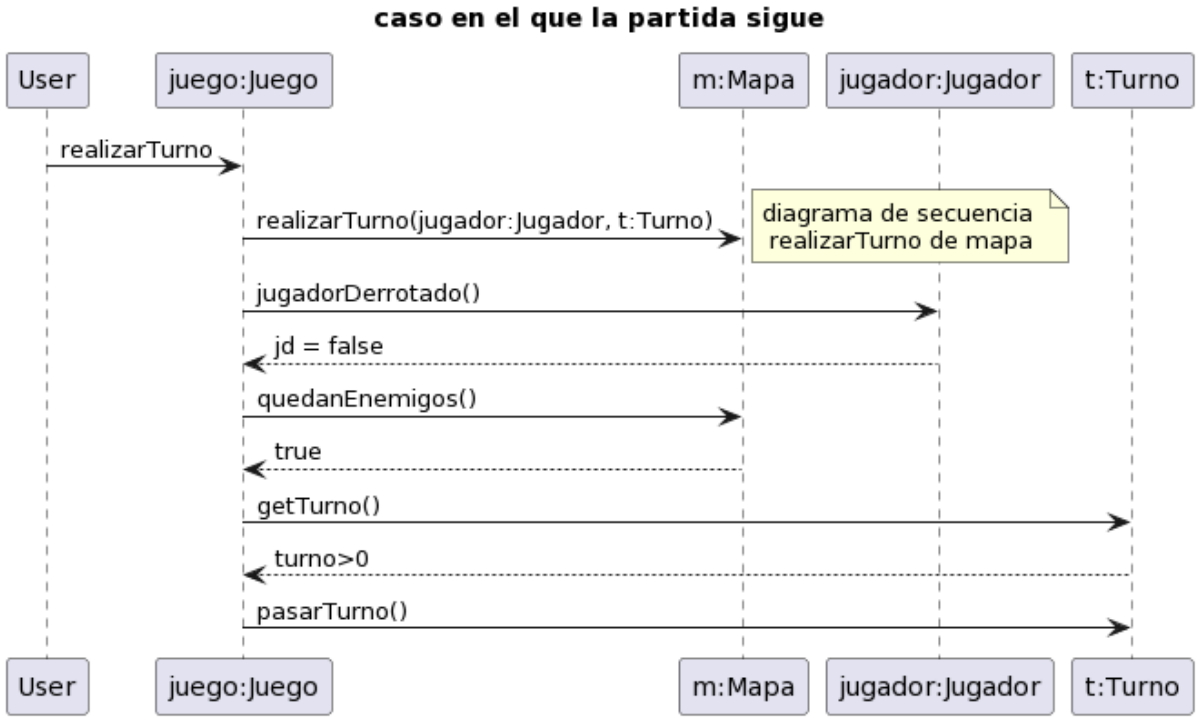

RealizarTurnoDeMapa
Actor
Mapa
Spawn
Turno
realizarTurno()
turnoDeDefensas()
Diagrama de secuencia de Turno de defensas
enemigosRestantes(listaEnemigos:List,Jugador,Turno): ListaEnemigosVivos:List
Diagrama de secuencia de Enemigos Restantes
size()
size:int
getTurno()
turno:int
get(turno)
enemigos:List
defensasDisponibles(): ListaDefensasDisponibles:List
Diagrama de secuencia de defensas disponibles
Actor
Mapa
Spawn
Turno

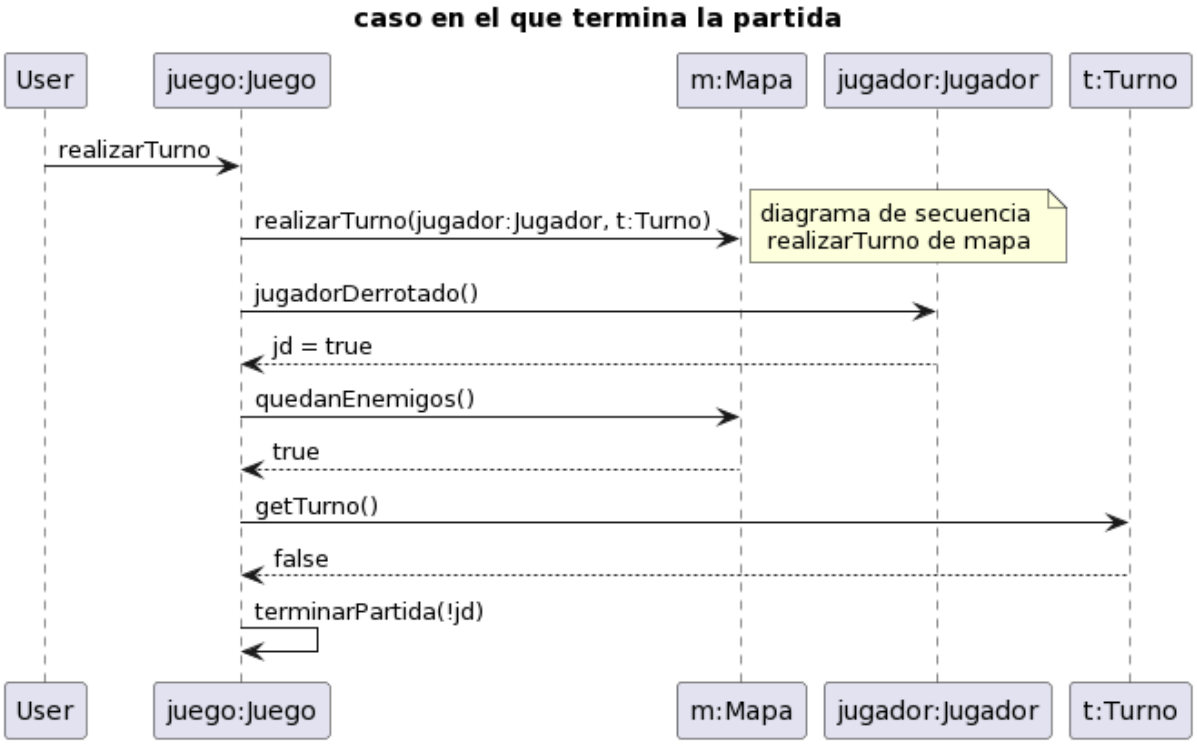

enemigos restantes
Actor
Mapa
Enemigo
Movimiento
EnemigoFactory
ataque
Jugador
enemigosRestantes(ListaEnemigos,Jugador,Turno)
loop
[Para cada Enemigo]
Mover()
Mover()
llegoAlFinal()
true
obtenerAtaque(Enemigo,Turno,listaDefensas)
getNombre()
nombre:String
ataque:Ataque
definirAtaque(ataque)
dañar(Jugador)
recibirAtaque(daño:int)
otorgarCreditos(Jugador)
agregarCreditos(creditos:int)
sumarEnemigoMuerto(Jugador)*Si es una Hormiga se suma*
listaEnemigosVivos:List
Actor
Mapa
Enemigo
Movimiento
EnemigoFactory
ataque
Jugador

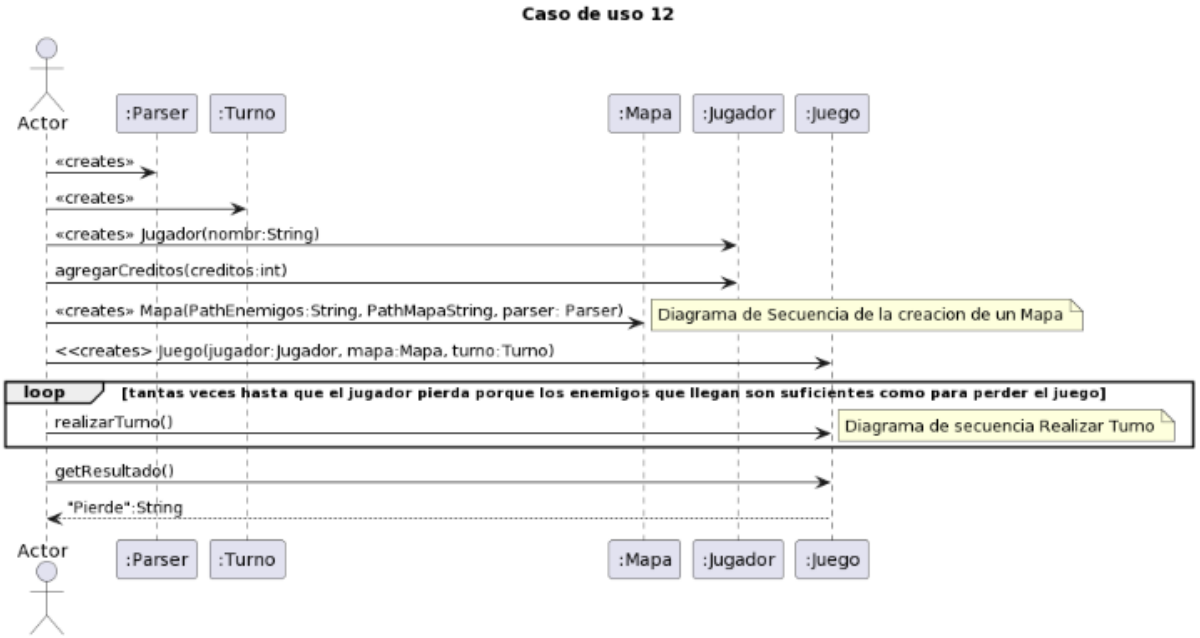

defensas disponibles
Actor
Mapa
Defensa
Estado
defensasDisponibles()
loop
[Para cada defensa]
destruido()
destruido()
boolean
boolean
si devuelve true se libera la parcela, sino se agrega a la lista
listaDefensasDisponibles:List
Actor
Mapa
Defensa
Estado

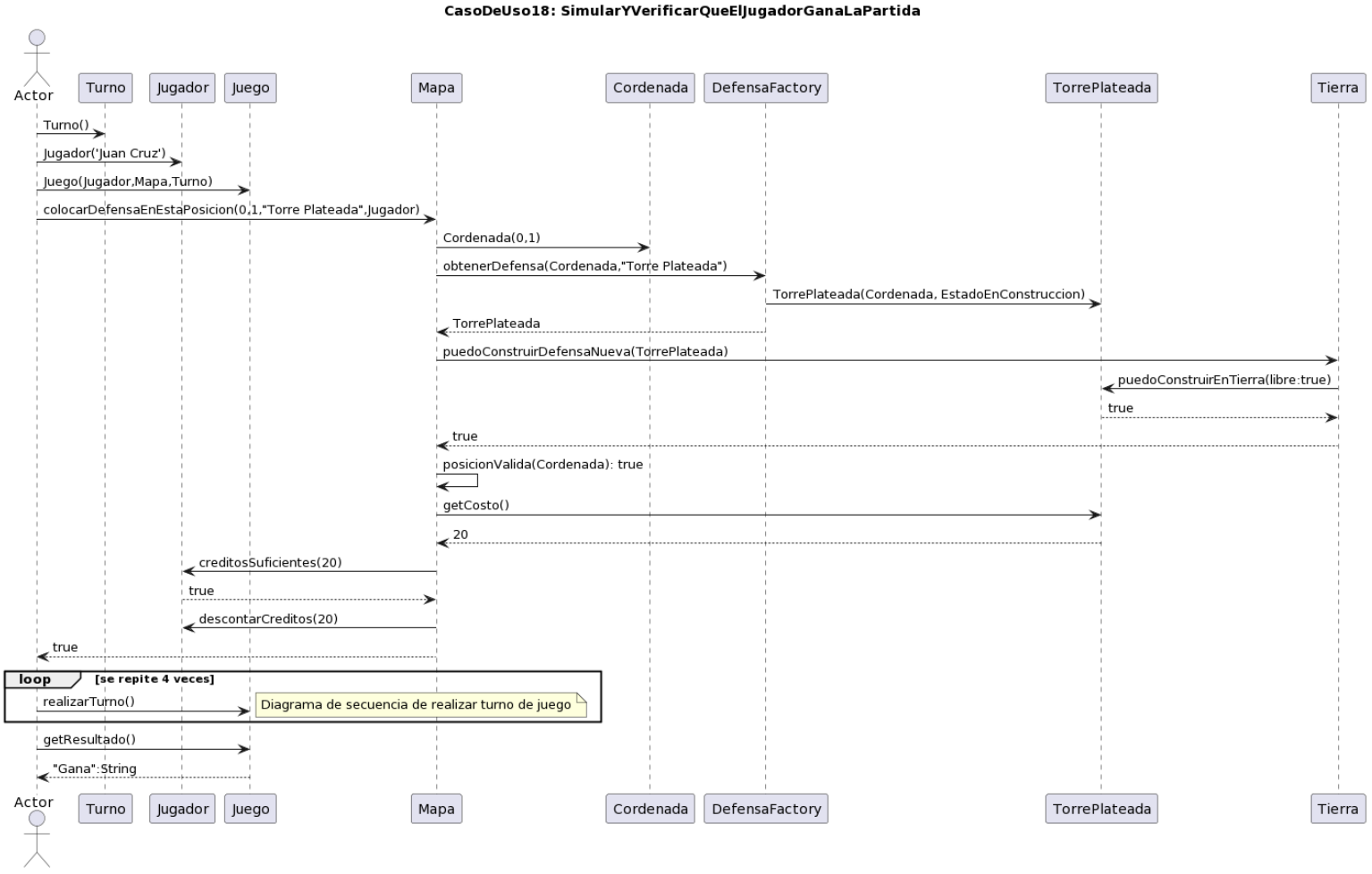

turnoAtacarEnemigos en Mapa (hormiga)
m:Mapa
af:AtaqueFactory
e:Enemigo
a:ataque
j:Jugador
turnoAtacarEnemigos(e:Enemigo, t:Turno,j:Jugador)
obtenerAtaque(e, t,listaDefensas)
ataque:Ataque
definirAtaque(ataque)
atacarJugador(j)
dañar(j)
recibirAtaque(2)
otorgarCreditos(j)
getHormigasDestruidas()
como son 4
las hormigas muertas da 1 credito
4
agregarCreditos(1)
sumarEnemigoMuerto(j)
sumarHormigaMuerta()
m:Mapa
af:AtaqueFactory
e:Enemigo
a:ataque
j:Jugador


ataque torre operativa (puede atacar)
User
d:Defensa
estado:Estado
enemigo:Enemigo
c:Cordenada
atacar(enemigos)
puedoAtacar()
getCordenada(rangoTorre: Int,)
cordenadaEnemigo:Cordenada
estaEnRango(rangoTorre: Int,cordenadaEnemigo)
true
getEnergia()
si energia es >0
y esta en rango
energia:Int
recibirAtaque(danio)
User
d:Defensa
estado:Estado
enemigo:Enemigo
c:Cordenada

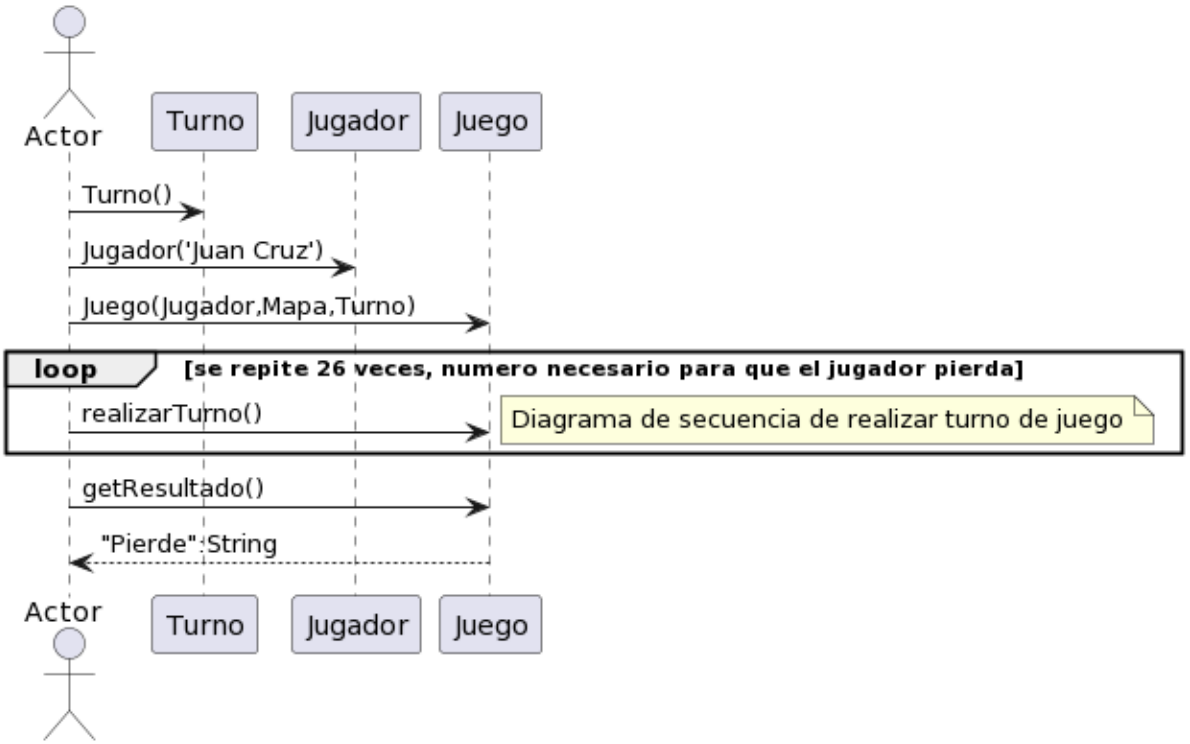

caso en el que la partida sigue
User
juego:Juego
m:Mapa
jugador:Jugador
t:Turno
realizarTurno
realizarTurno(jugador:Jugador, t:Turno)
diagrama de secuencia
realizarTurno de mapa
jugadorDerrotado()
jd = false
quedanEnemigos()
true
getTurno()
turno>0
pasarTurno()
User
juego:Juego
m:Mapa
jugador:Jugador
t:Turno


caso en el que termina la partida
User
juego:Juego
m:Mapa
jugador:Jugador
t:Turno
realizarTurno
realizarTurno(jugador:Jugador, t:Turno)
diagrama de secuencia
realizarTurno de mapa
jugadorDerrotado()
jd = true
quedanEnemigos()
true
getTurno()
false
terminarPartida(!jd)
User
juego:Juego
m:Mapa
jugador:Jugador
t:Turno

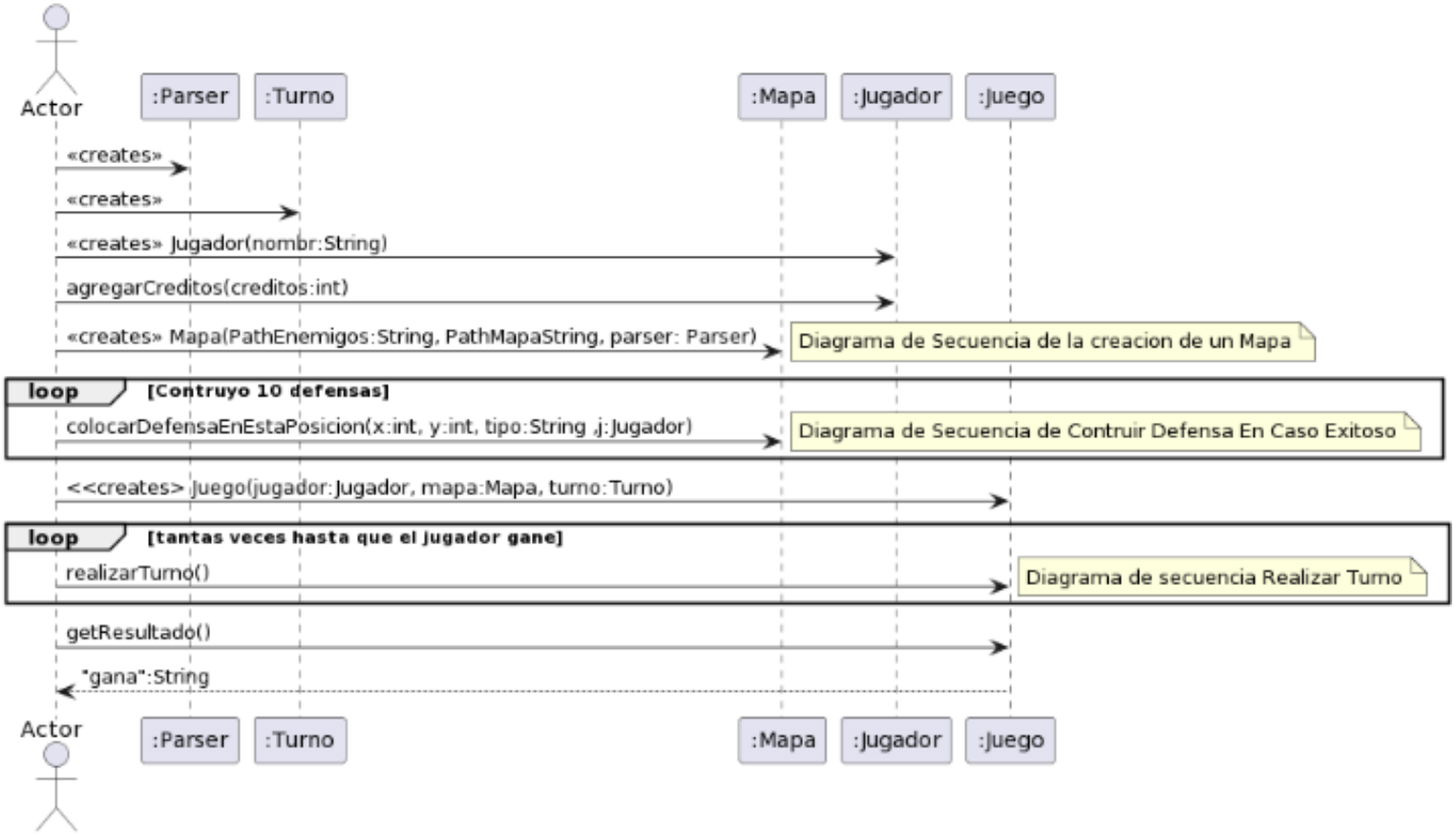

Caso de uso 12
Actor
:Parser
:Turno
:Mapa
:Jugador
:Juego
«creates»
«creates»
«creates» Jugador(nombr:String)
agregarCreditos(creditos:int)
«creates» Mapa(PathEnemigos:String, PathMapaString, parser: Parser)
Diagrama de Secuencia de la creacion de un Mapa
<<creates> Juego(jugador:Jugador, mapa:Mapa, turno:Turno)
loop
[tantas veces hasta que el jugador pierda porque los enemigos que llegan son suficientes como para perder el juego]
realizarTurno()
Diagrama de secuencia Realizar Turno
getResultado()
"Pierde":String
Actor
:Parser
:Turno
:Mapa
:Jugador
:Juego


CasoDeUso18: SimularYVerificarQueElJugadorGanaLaPartida
Actor
Turno
Jugador
Juego
Mapa
Cordenada
DefensaFactory
TorrePlateada
Tierra
Turno()
Jugador('Juan Cruz')
Juego(Jugador,Mapa,Turno)
colocarDefensaEnEstaPosicion(0,1,"Torre Plateada",Jugador)
Cordenada(0,1)
obtenerDefensa(Cordenada,"Torre Plateada")
TorrePlateada(Cordenada, EstadoEnConstruccion)
TorrePlateada
puedoConstruirDefensaNueva(TorrePlateada)
puedoConstruirEnTierra(libre:true)
true
true
posicionValida(Cordenada): true
getCosto()
20
creditosSuficientes(20)
true
descontarCreditos(20)
true
loop
[se repite 4 veces]
realizarTurno()
Diagrama de secuencia de realizar turno de juego
getResultado()
"Gana":String
Actor
Turno
Jugador
Juego
Mapa
Cordenada
DefensaFactory
TorrePlateada
Tierra

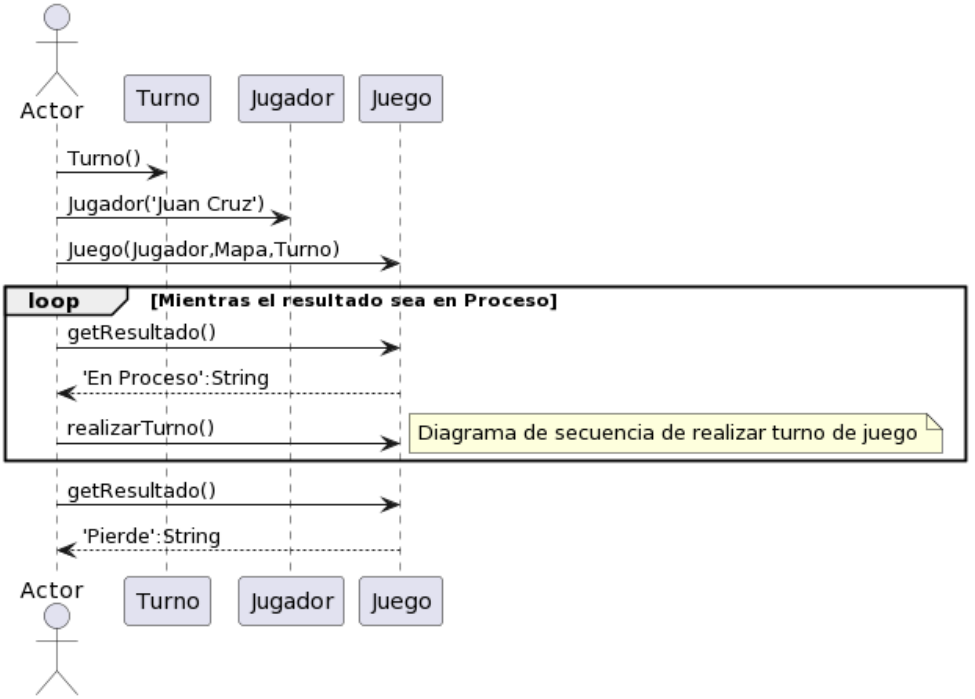
CasoDeUso19: SimularYVerificarQueElJugadorPierdeLaPartida



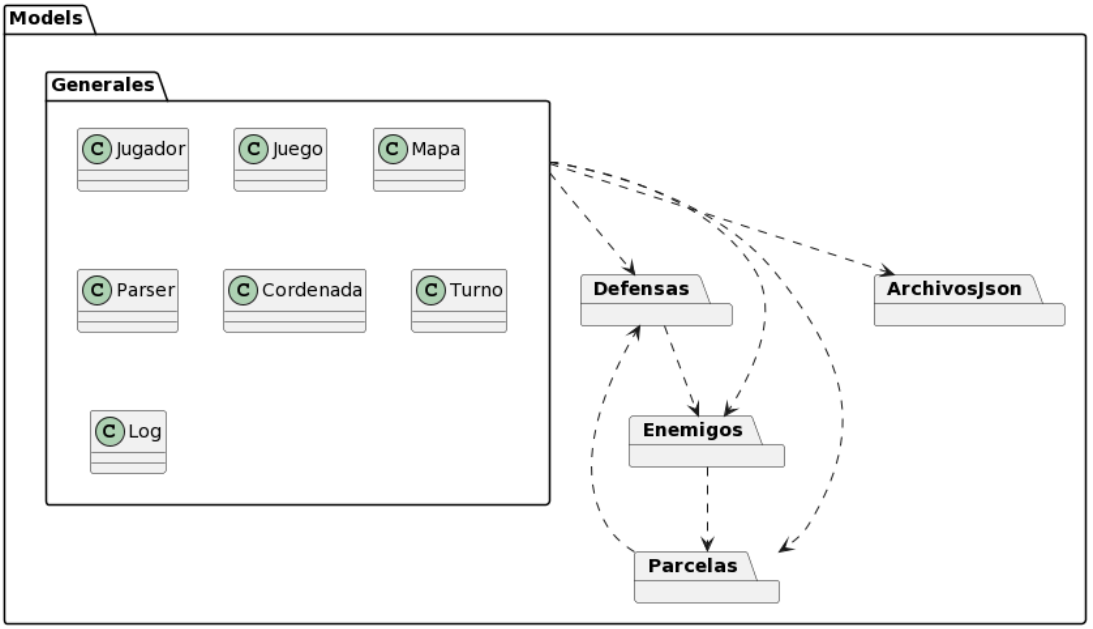
Caso de uso 10

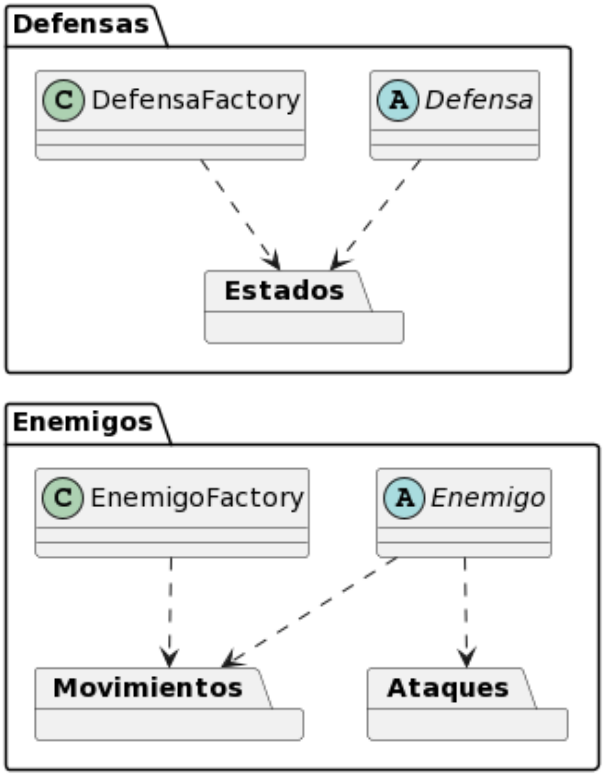


casoDeUso12: VerificarQueSiLasUnidadesEnemigasLlegadasALaMetaMatanAlJugadorEstePierdeElJuego

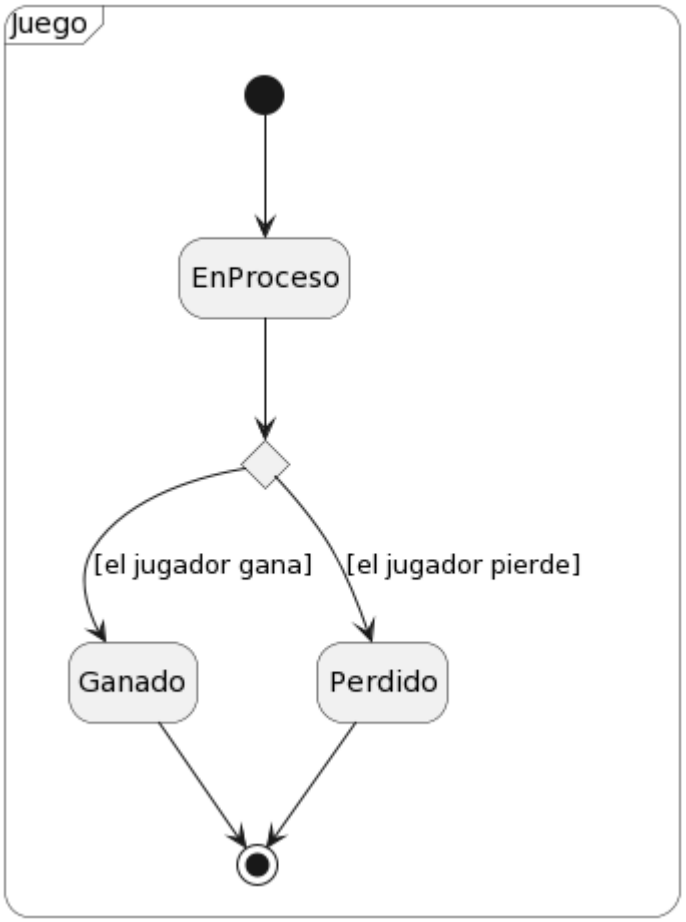


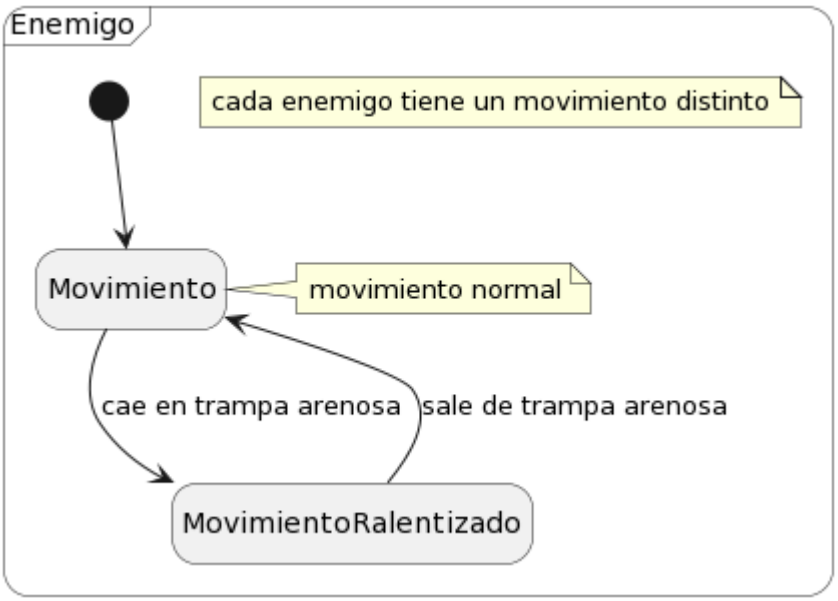
5. Diagramas de paquetes



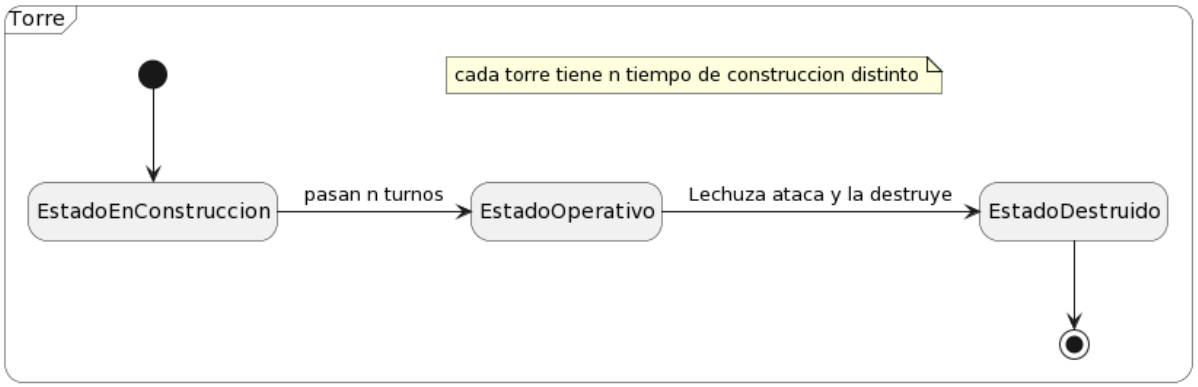


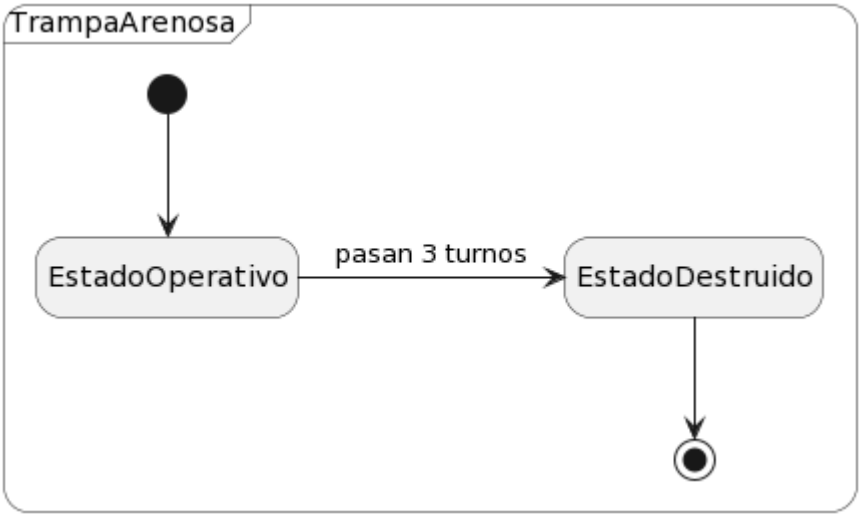
6. Diagramas de estado





Diagramas de estados de las defensas





7. Detalles de implementación

Para la realización del trabajo creamos las siguientes clases:

- **Juego**: lleva el control de todo el juego, y el que tiene el acceso al turno y al mapa.
- **Turno**: lleva el conteo de los turnos del juego.
- **Mapa**: es la clase que tiene acceso a la clase Parser y se encarga de armar la estructura del mapa. También es quién tiene toda la información del juego, como los enemigos y defensas que existen.
- **Parser**: se encarga de la lectura de los archivos json y los devuelve en listas de string para que puedan ser usados por Mapa.
- **Log**: se encarga de los mensajes de la consola.
- **Cordenada**: ayuda a poder identificar las posiciones en donde se ubican las parcelas en el mapa.
- **Defensa**: es una clase abstracta que se encarga de los comportamientos de las defensas del juegos.
  Las clases que heredan de esta son: **TorrePlateada**, **TorreBlanca** y **TrampaArenosa**.
- **DefensaFactory**: se encarga de la creación de las clases de Defensa.
- **Estado**: interfaz de los tipos de estado que puede tener una defensa.
  Las clases que implementan esta interfaz son: **EstadoDestruido**, **EstadoEnConstruccion** y **EstadoOperativo**.
- **Enemigo**: clase abstracta que se encarga del comportamiento que tienen los enemigos del juego.
  De esta clase heredan: **Lechuza**, **Topo**, **Araña** y **Hormiga**.
- **EnemigoFactory**: se encarga de la creación de las clases de Enemigo.
- **Movimiento**: clase abstracta que tiene los comportamientos generales de los movimientos del enemigo.
  De esta clase heredan: **MovimientoAraña**, **MovimientoArañaRalentizado**, **MovimientoHormiga**, **MovimientoHormigaRalentizado**, **MovimientoLechuzaenDiagonal**, **MovimientoLechuzaenL**, **MovimientoTopo** y **MovimientoTopoRalentizado**.
- **MovimientoFactory**: se encarga de la creación de las clases de Movimiento.
- **Ataque**: clase abstracta que se encarga de los ataques de los enemigos.
  De esta clase heredan: **AtaqueAraña**, **AtaqueHormiga**, **AtaqueTopo** y **AtaqueLechuza**.
- **AtaqueFactory**: se encarga de la creación de las clases de Ataque.
- **Parcela**: clase abstracta que contiene los comportamientos de las parcelas del mapa.
  De esta clase heredan: **Rocoso**, **Tierra** y **Pasarela**.
- **ParcelaFactory**: se encarga de la creación de las clases de Parcela.

Se aplicó el patrón de diseño State para el estado de las defensas, que consta de una interfaz Estado la cual es implementada por las clases "EstadoEnConstruccion", "EstadoOperativo" y "EstadoDestruido". También se utiliza

este patrón de diseño en el movimiento y el ataque del enemigo, salvo que en vez de una interfaz se usó una clase abstracta.

Se utilizó el patrón Singleton para Log, y así poder acceder a esa clase desde cualquier otra, ya que está clase solo se encarga de los mensajes de la consola.

8. Excepciones

-**ExcepcionConstruirDefensaEnPosiocionInvalida**: se captura cuando se intenta colocar una defensa en una posición inválida.

Las posiciones inválidas serían: cuando se intenta colocar la defensa en una parcela que no es la que le corresponde, o una torre encima de otra.

Esta excepción la creamos para mejorar la experiencia del usuario, ya que así se le puede avisar que lo que está intentando hacer (crear una defensa en una posición inválida) no está permitido en el juego.