FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Java

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

Alumnos:	Padrón
Avalo, Alejandro Valentin	87679
Di Grazia, Diego	108970
Miranda Iglesias, Joaquin	97500
Mogilewski, Aldana	98770
Nazarian, Juan Martin	90206

Corrector: Villores, Alejo

Índice

1. Introducción

A continuación se presenta el informe correspondiente al segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar un juego basado en el ya existente *TowerDefense*. El mismo fue desarrollado en conjunto por 5 integrantes, utilizando el paradigma de Programación Orientada a Objetos, la técnica de desarrollo de software Test Driven Development, utilizando el lenguaje de programación *Java* junto con *JavaFX*.

2. Supuestos

- Al llegar a la meta, los enemigos atacan y son eliminados en el siguiente turno.
- Cuando una lechuza elimina a una defensa, la defensa es eliminada del mapa.
- Los enemigos son borrados del juego al morir.
- El jugador gana la partida si no quedan enemigos vivos.

3. Detalles de implementación

Antes de iniciar con la implementación del programa, se diseño un modelo general del cual surgieron distintas ideas e implementaciones sobre la manera de implementarlo. En un primer lugar, se agregaron todas las clases que se consideraron necesarias, sin ningun grado de abstracción, con la idea de que el mismo se vaya haciendo visible a medida que se avance con la ejecución del proyecto en sí.

A medida que se fue avanzando en el trabajo, se hizo notable la necesidad de aplicar clases abstractas, interfaces, y distintos patrones de diseño como los vistos en clase.

Los patrones de diseño implementados fueron:

- *Strategy*
- *Abstract Factory*
- *Observer*

El patrón *Strategy* fue utilizado para la transición entre estados. Por ejemplo, al construir una torre, se utiliza este patrón para que la misma pase del estado `En Construcción` al estado `Terminado` en tiempo de ejecución, sin tener que preguntar el mismo y tomar una decisión antes de que suceda. Lo mismo se implementó con los enemigos, al pasar de `Vivo` a `Muerto`.

El patrón *Abstract Factory* fue implementado para la creación de Parcelas y Enemigos. Este patrón permite encapsular la creación de distintos objetos y garantizar que dichos objetos pertenezcan a una misma familia. En el caso de este trabajo, se crean objetos de *PasarelaNormal*, *PasarelaMeta*, *PasarelaLargada*, *Rocoso* y *Tierra*, y lo mismo sucede con *TorreBlanca* y *TorrePlateada*.

El patrón *Observer* se utilizó para la comunicación entre el backend y el frontend. De esta manera, se cuenta con un controlador, que le pide la información al backend para entregarsela al frontend, con el objetivo de que este último pueda mostrar satisfactoriamente lo que sucede a medida que se desarrolla el juego. Este patrón permite la comunicación entre objetos evitando la necesidad de que los mismos esten acoplados, y no sea necesario que tengan conocimiento directo entre ellos.

4. Diagramas de clase

A continuación se muestran los diagramas de clases tentativos del modelo, los mismos fueron divididos en diagramas más pequeños para su mejor legibilidad.

La idea fue ir mostrando de a partes todas las clases y sus relaciones. De esta manera es posible identificar mejor los métodos y atributos que posee cada una, y como esta conectada con el resto de las clases.

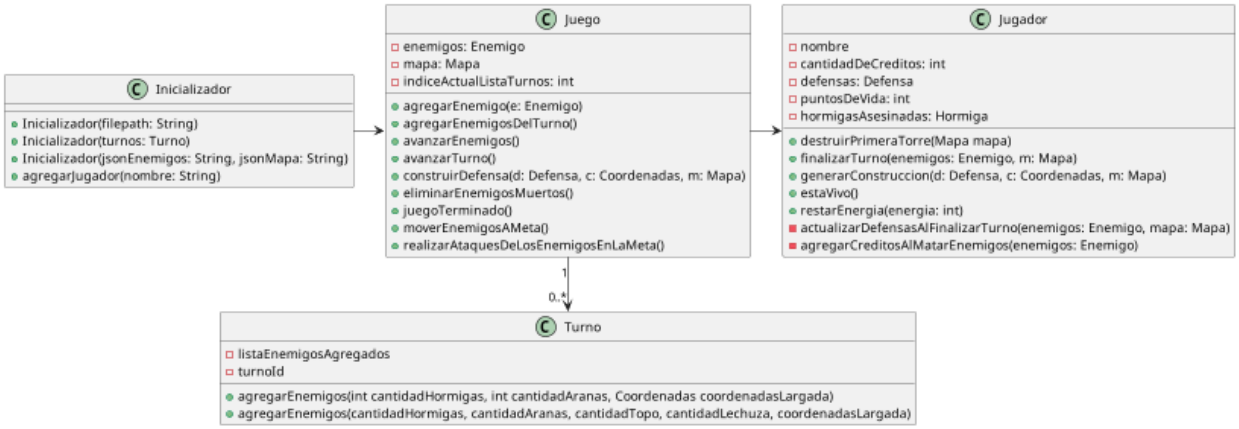

Figura 1: Diagrama de clases del *Juego*.

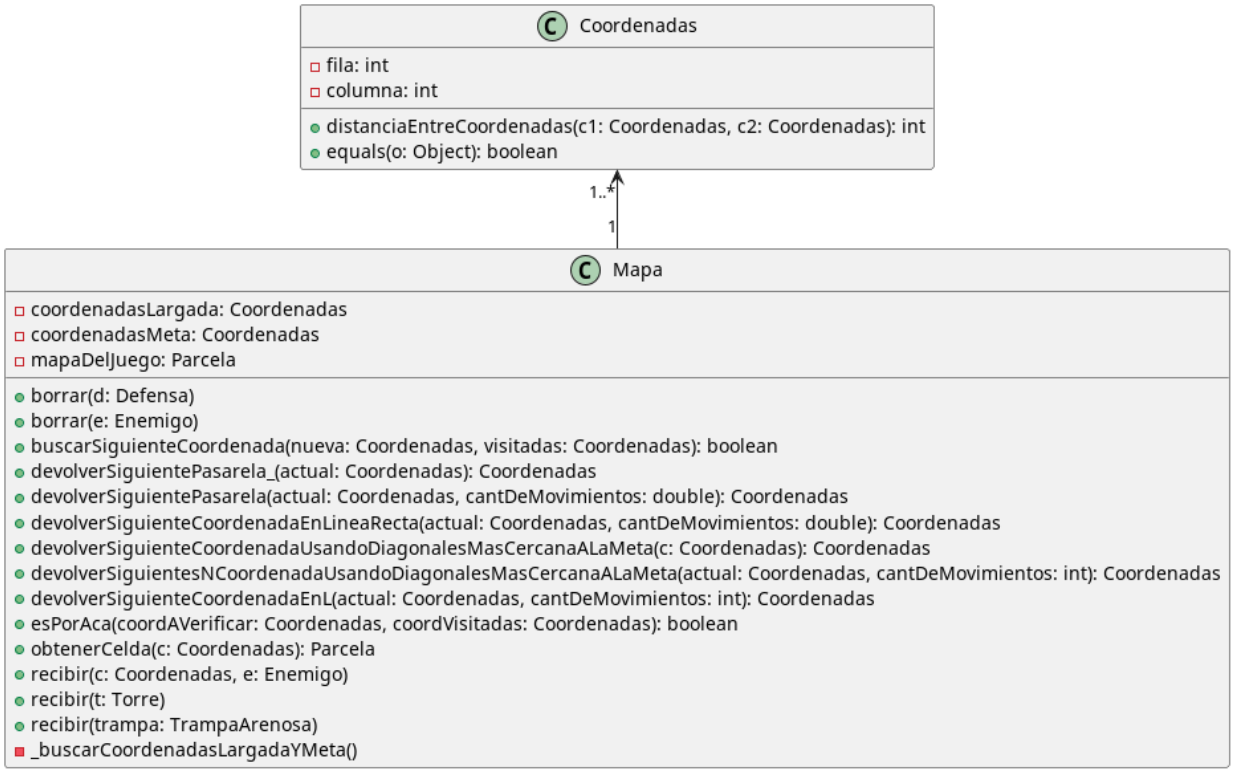

Figura 2: Diagrama de clases del *Mapa*.

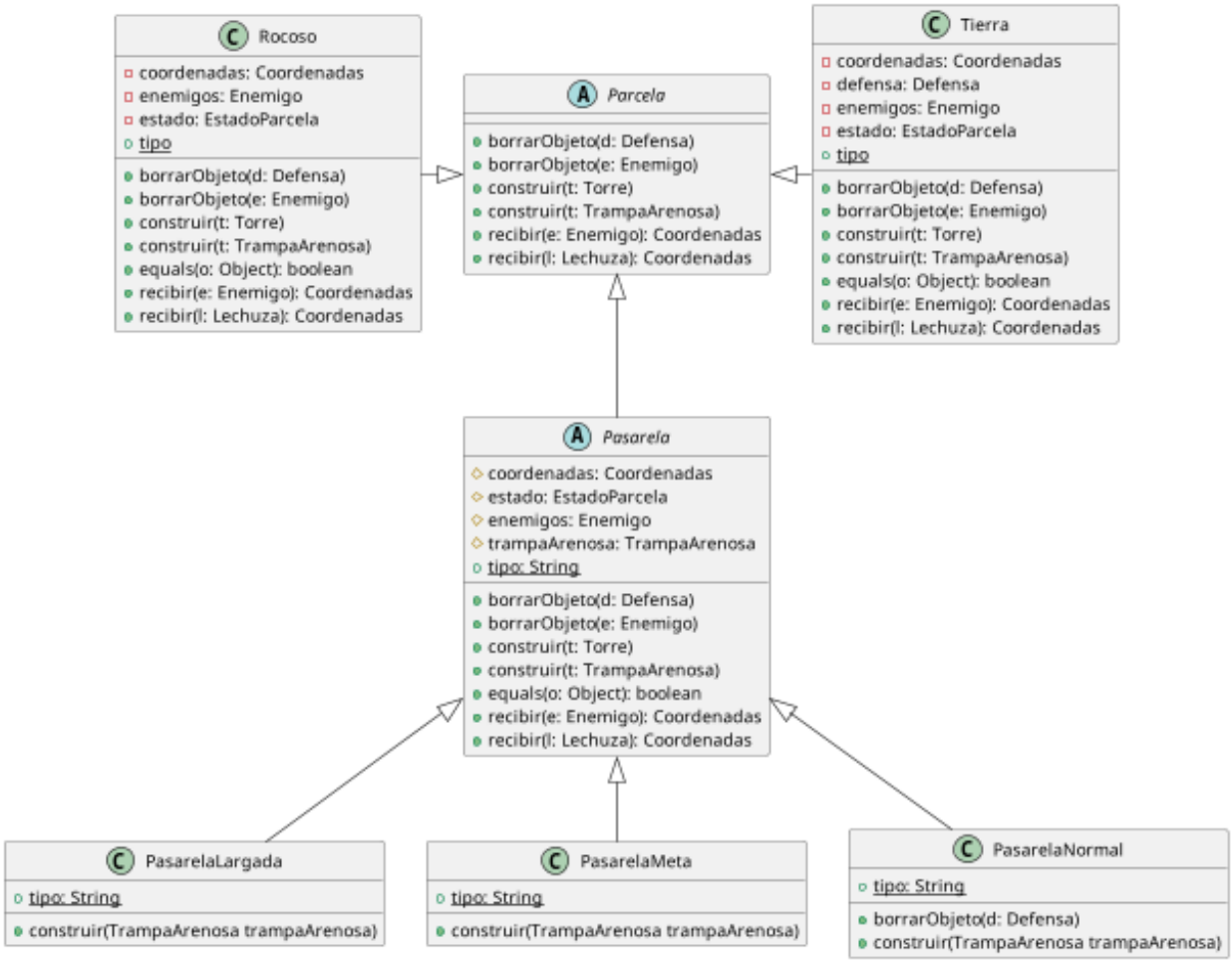

Figura 3: Diagrama de clases de *Parcela*.

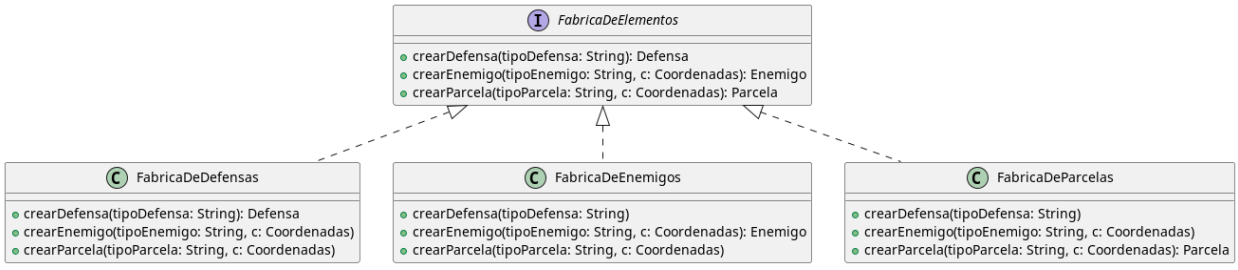

Figura 4: Diagrama de clases de *Fabrica*

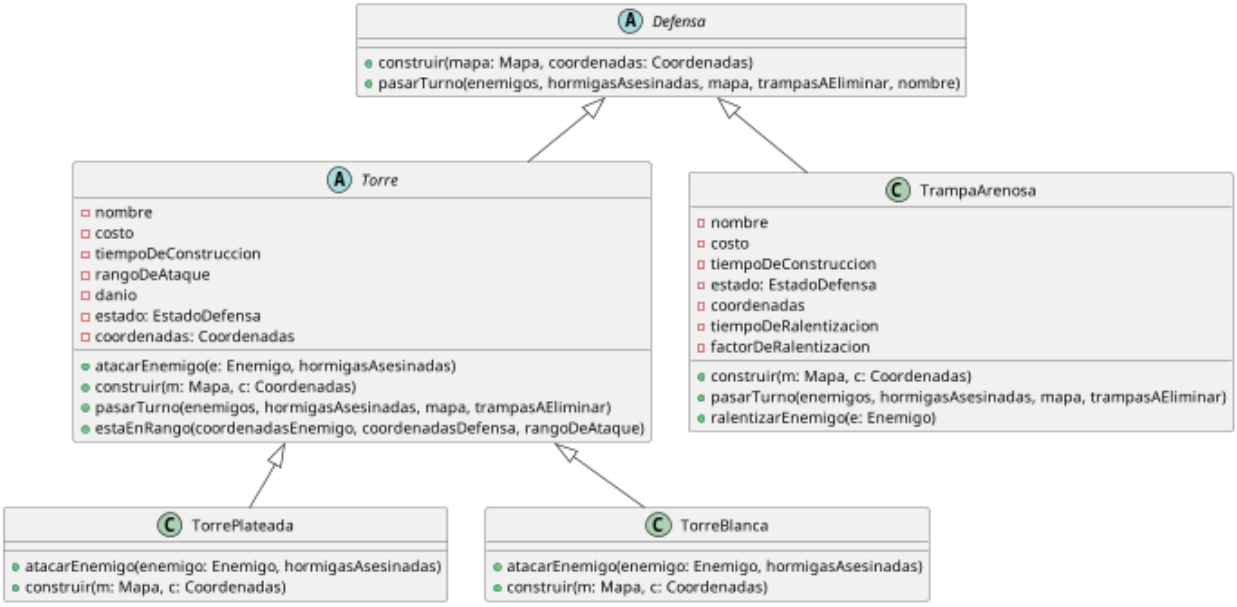

Figura 5: Diagrama de clases de *Defensa*.

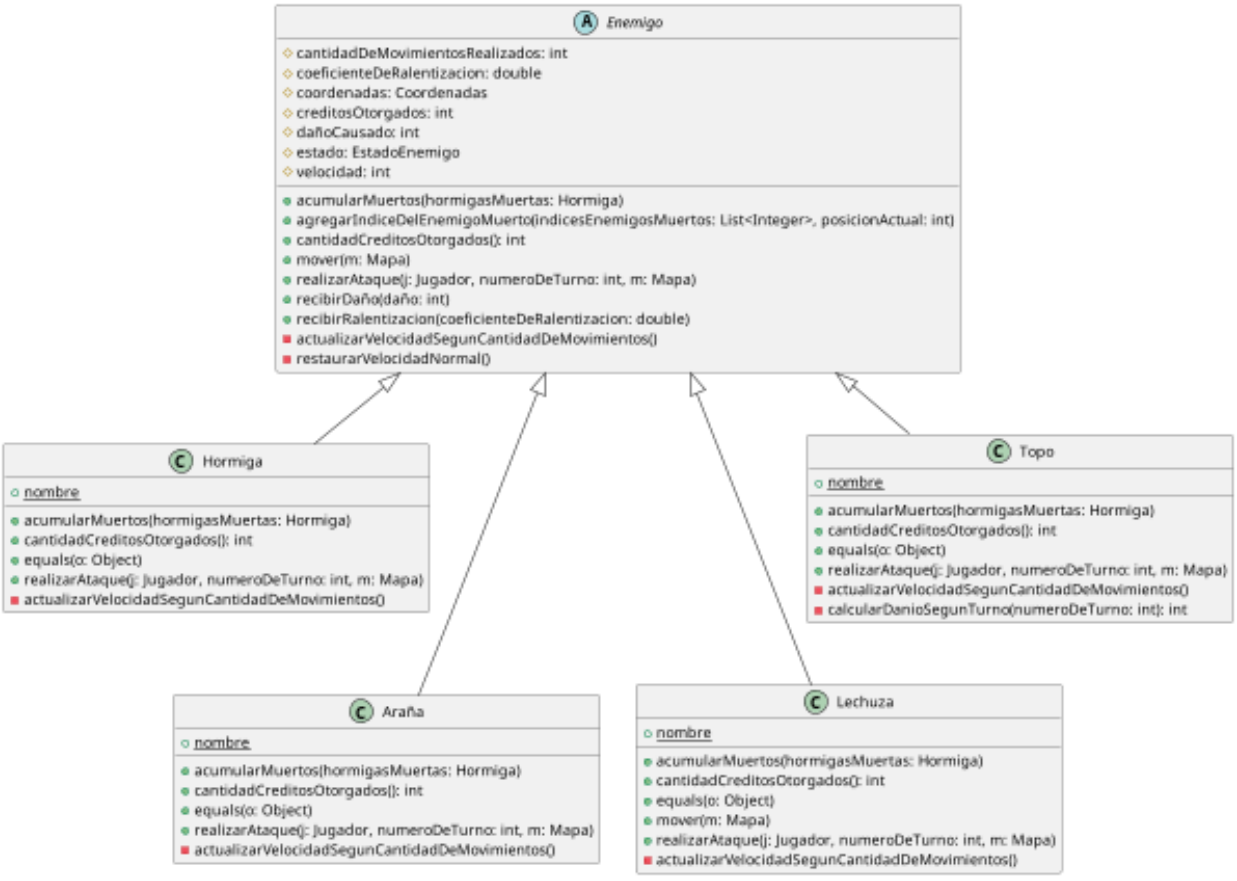

Figura 6: Diagrama de clases de *Enemigo*.

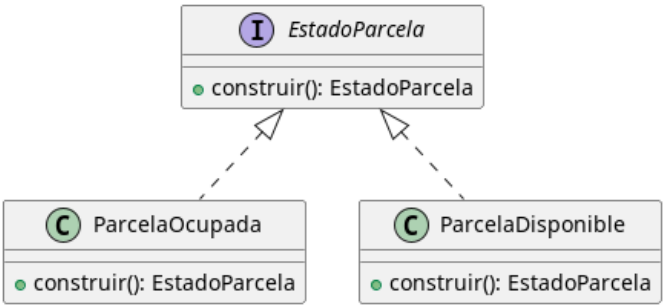

Figura 7: Diagrama de clases del Estado de la Parcela.

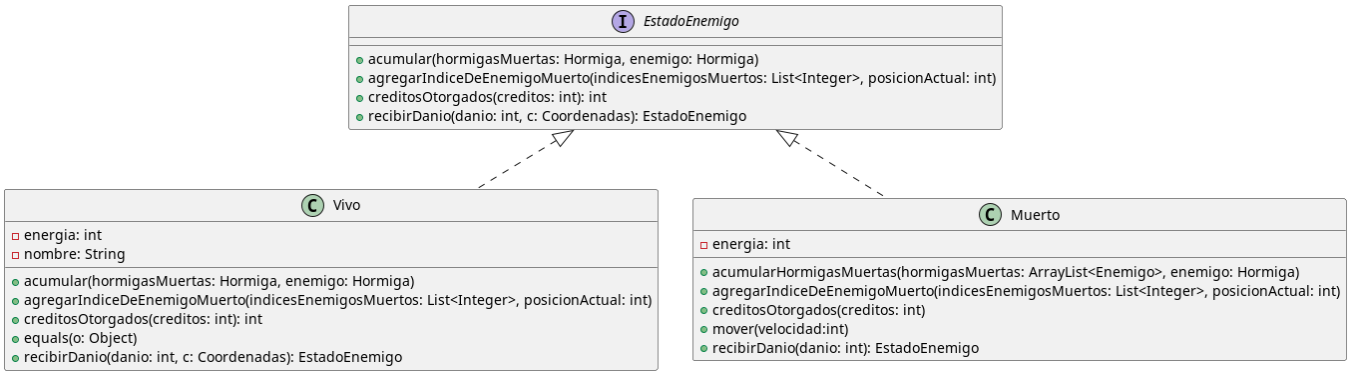

Figura 8: Diagrama de clases del Estado del Enemigo.

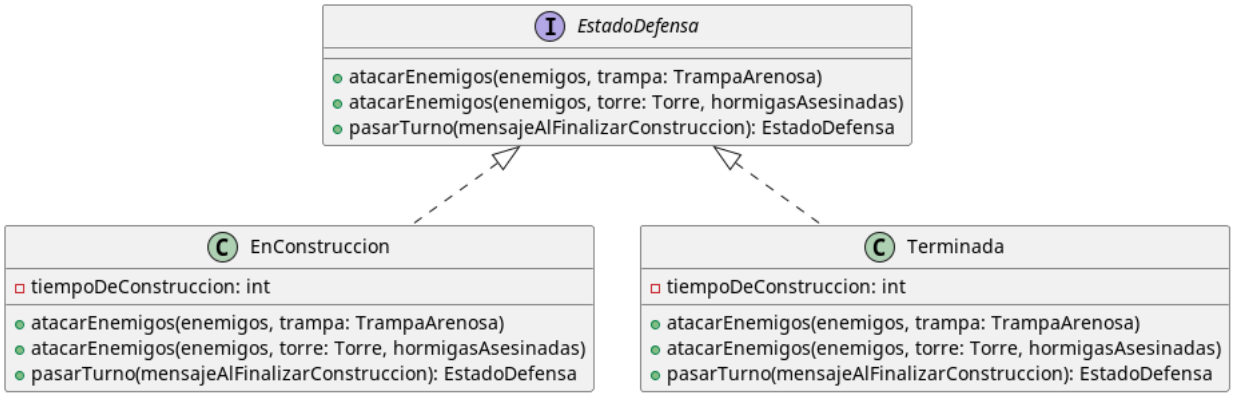

Figura 9: Diagrama de clases del Estado de la Defensa.

5. Diagramas de secuencia

5.1. Caso de uso 8

A continuación, se presenta un diagrama del ataque a una hormiga, los cambios que esto implica y como la misma le otorga créditos al jugador al morir. El mismo fue dividido en dos diagramas para lograr una mayor legibilidad.

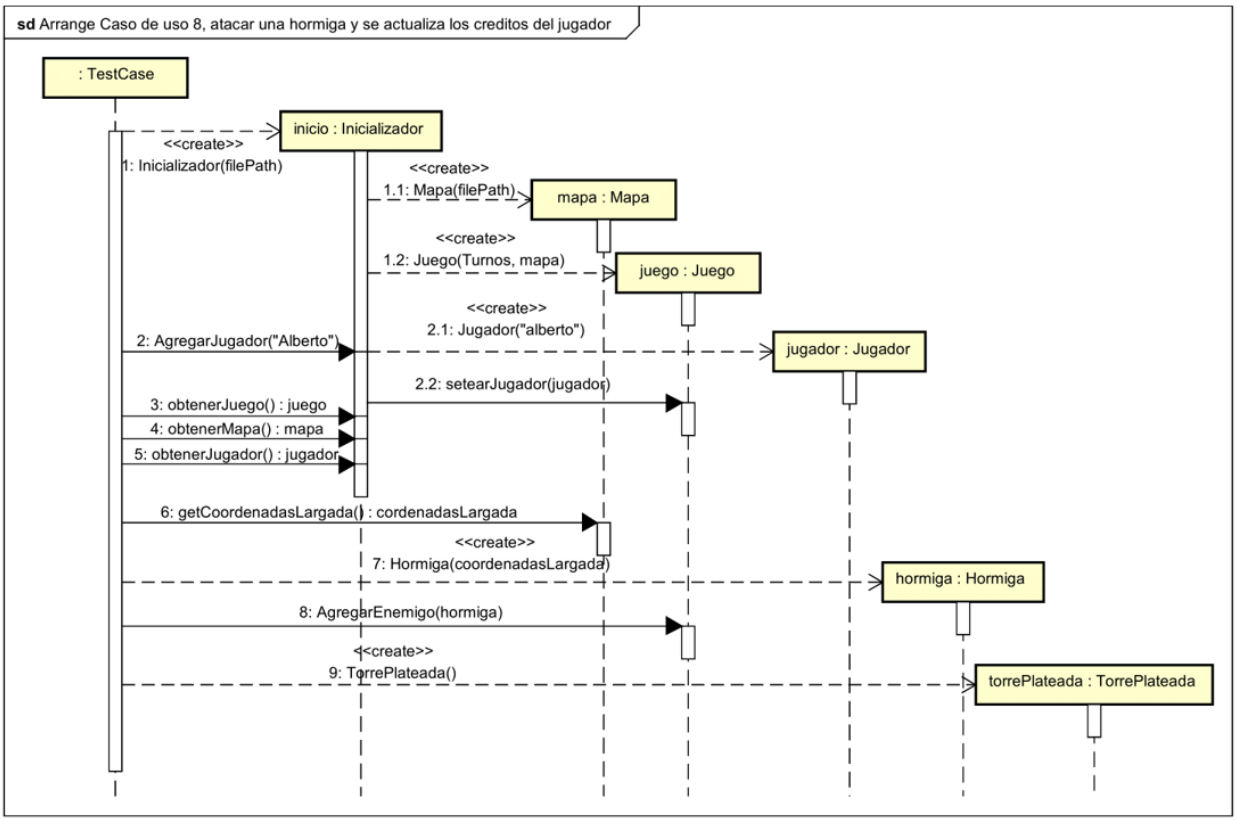

Figura 10: Diagrama de secuencia 2 - act.

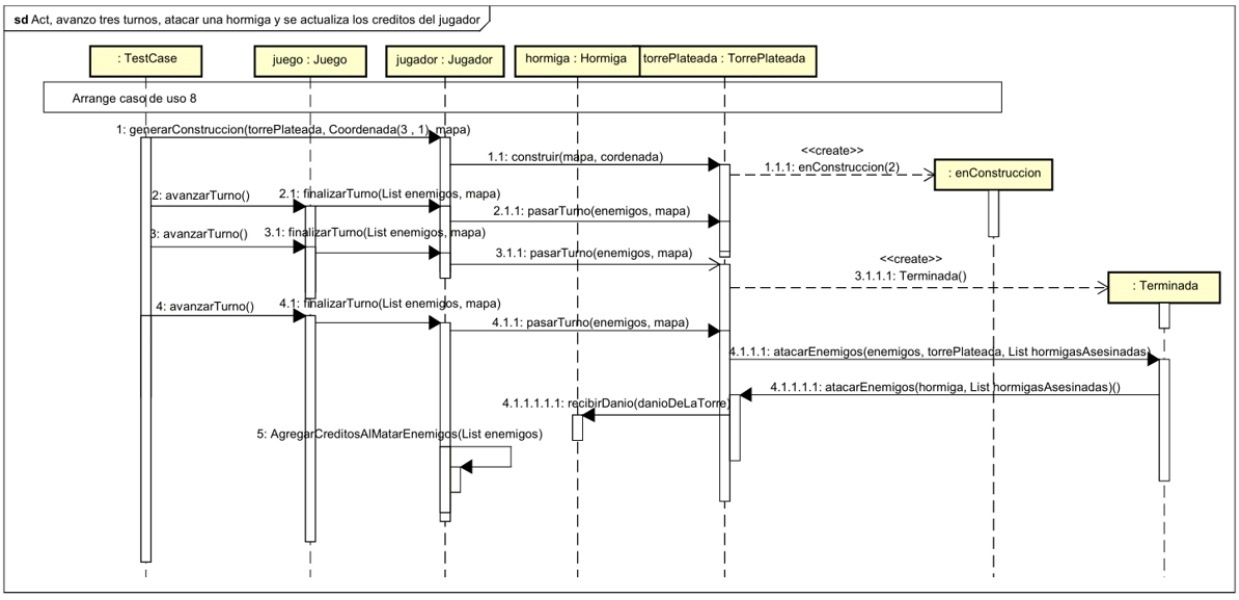

Figura 11: Diagrama de secuencia 2 - arrange.

5.2. Caso de uso 9

En el siguiente diagrama se muestra el caso de un enemigo moviéndose en el mapa al pasar de turno. A su vez, se pueden observar todas las inicializaciones necesarias de los objetos para el juego.

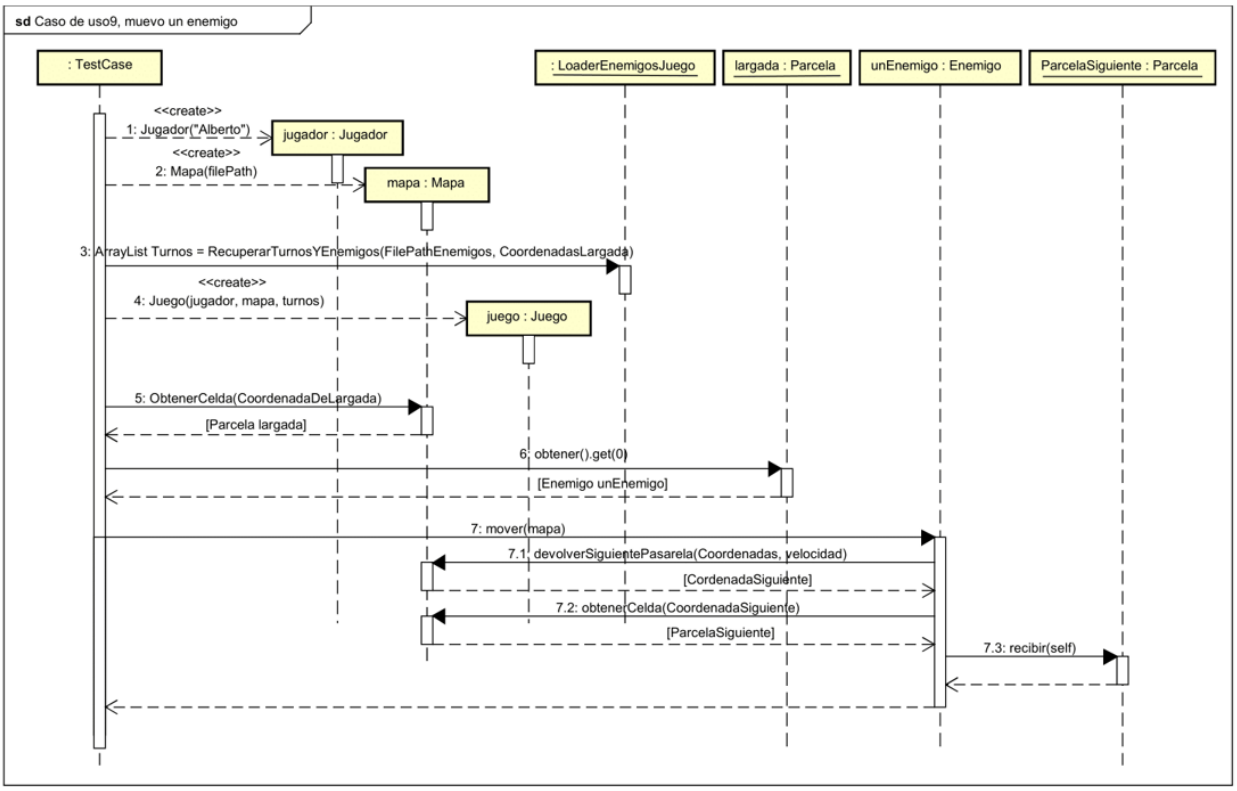

Figura 12: Diagrama de secuencia 1.

6. Diagramas de estado

A continuación se presentan diagramas de estado para algunos objetos del juego.

6.1. Torres

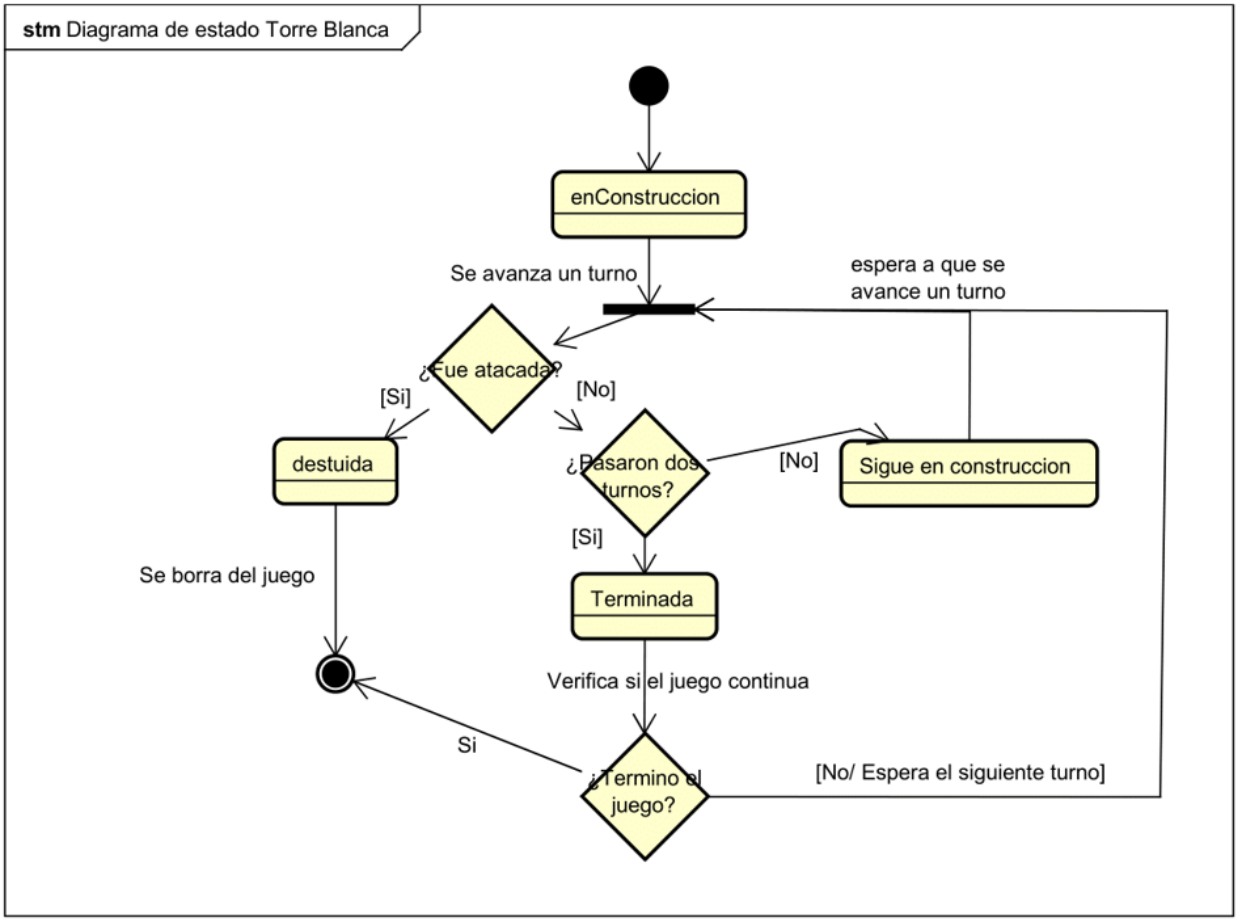

Figura 13: Diagrama de Estado - Torre Blanca.

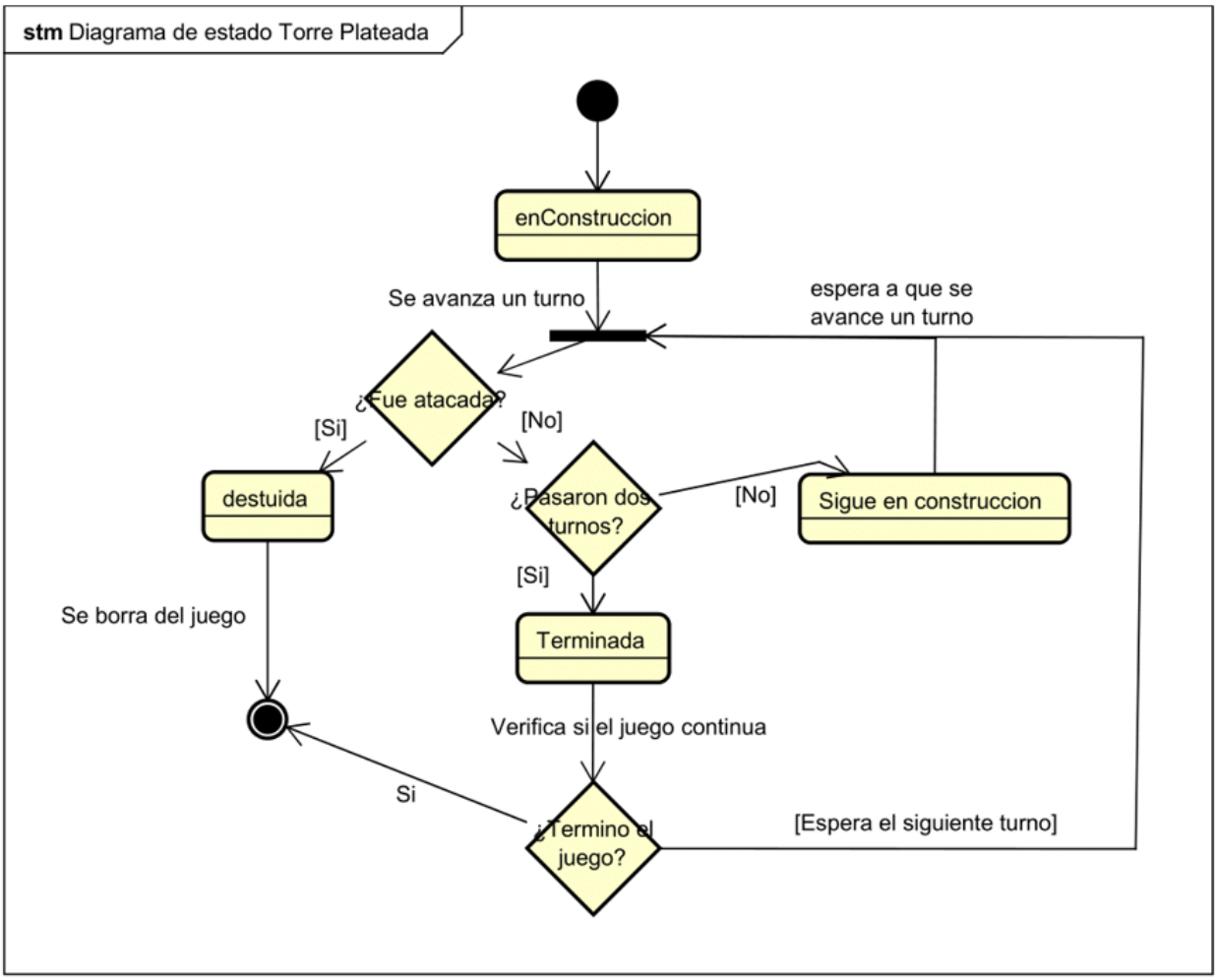

Figura 14: Diagrama de Estado - Torre Plateada.

6.2. Enemigos

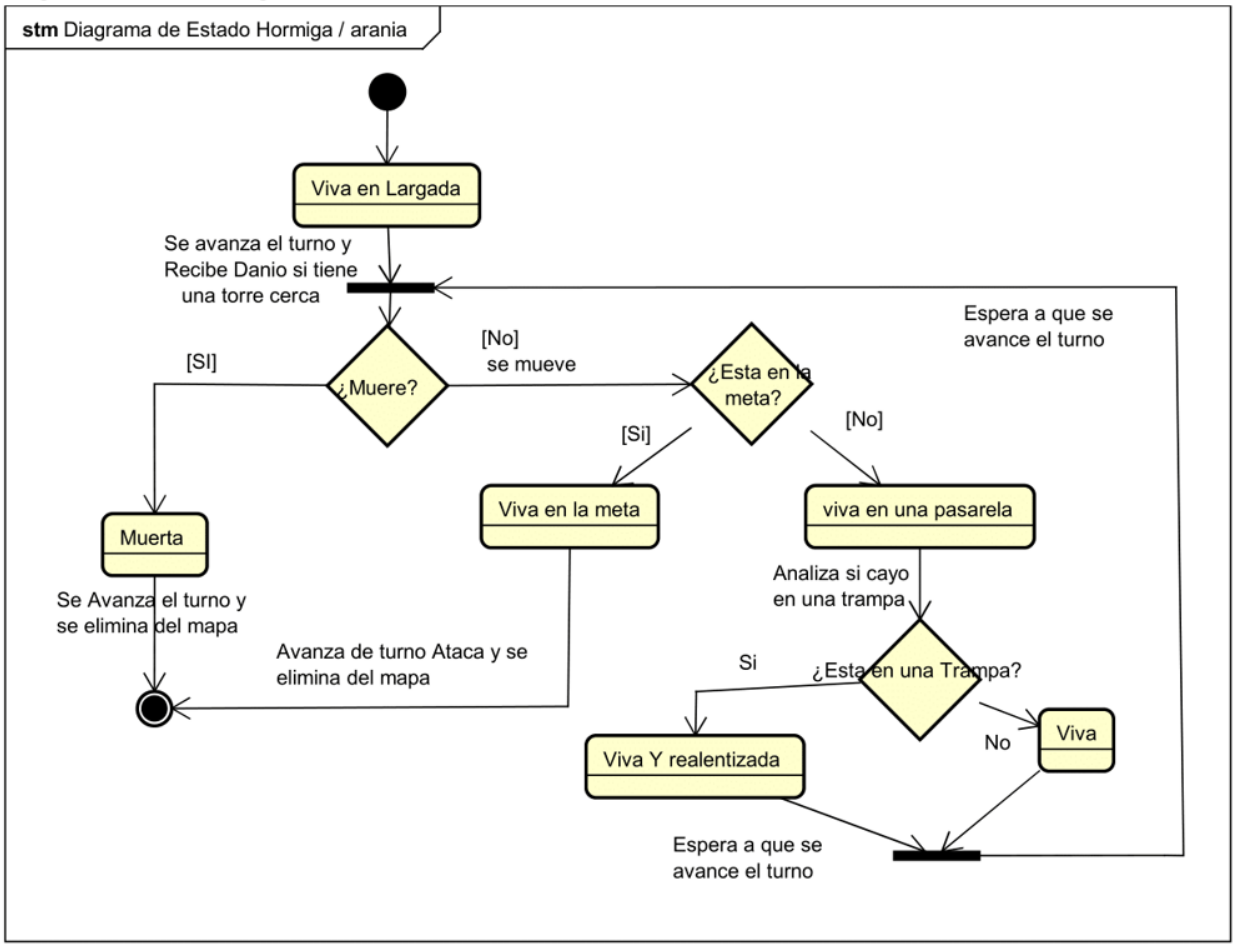

Figura 15: Diagrama de Estado - Hormiga.

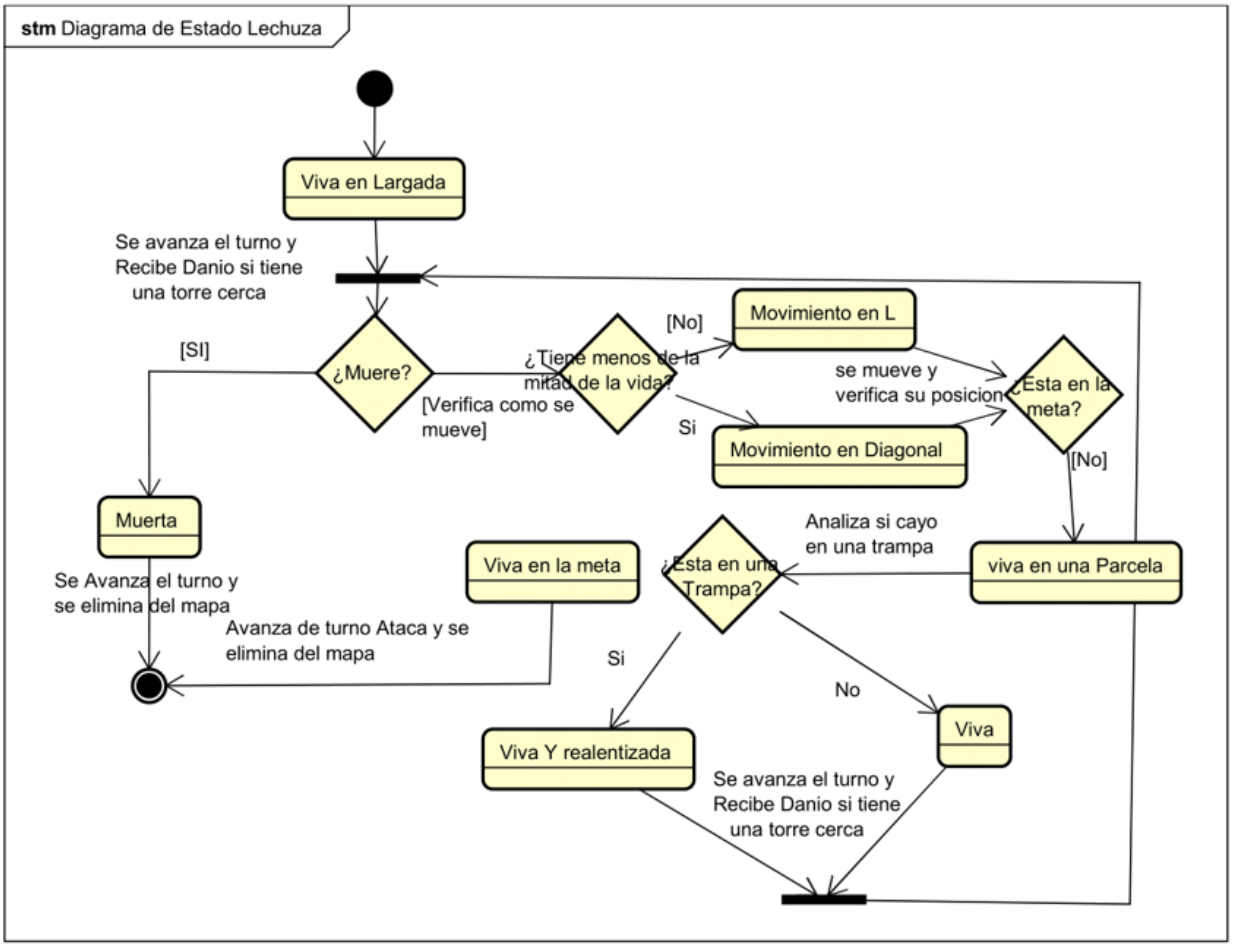

Figura 16: Diagrama de Estado - Lechuza.

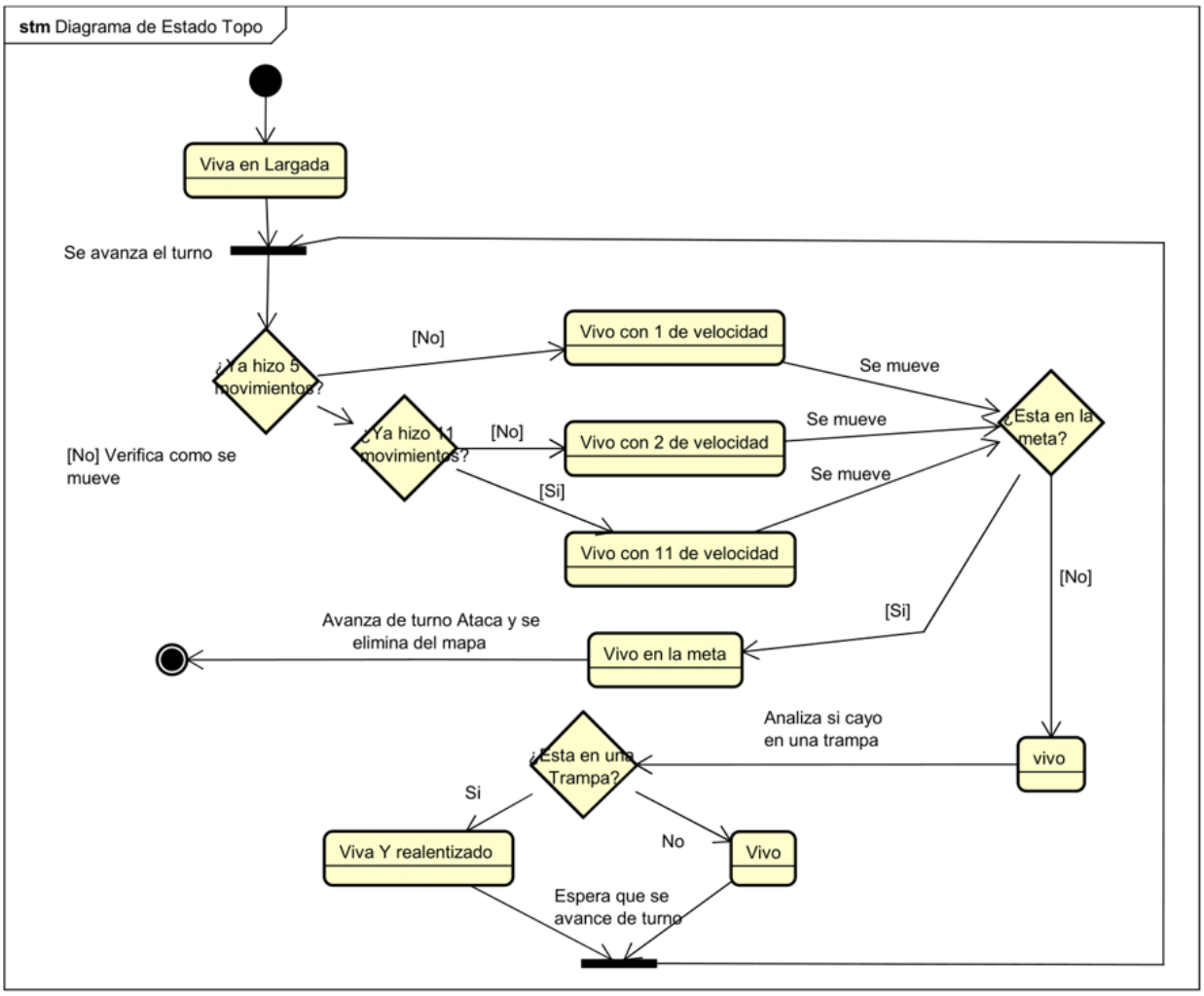

Figura 17: Diagrama de Estado - Topo.

7. Diagramas de paquetes

A continuación se muestran los diagramas de paquetes. Se decidió separar los mismos para mostrar las dependencias entre cada uno correctamente.

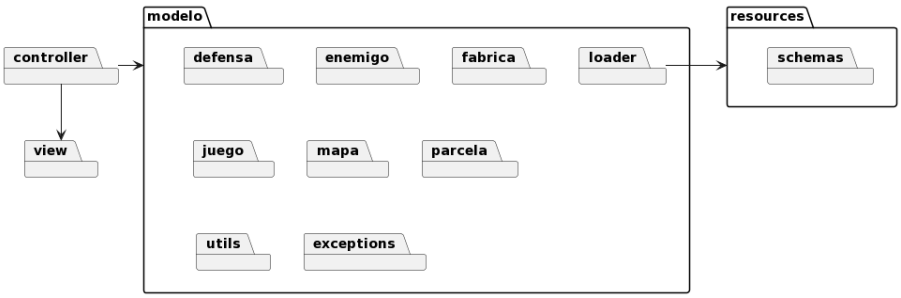

Figura 18: Diagrama de Paquetes - General.

7.1. Juego y Fábrica

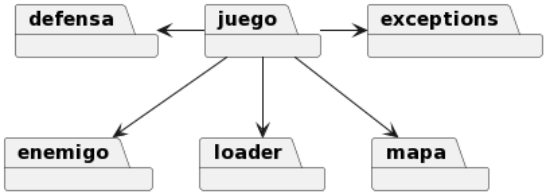

Figura 19: Diagrama de Paquetes - juego.

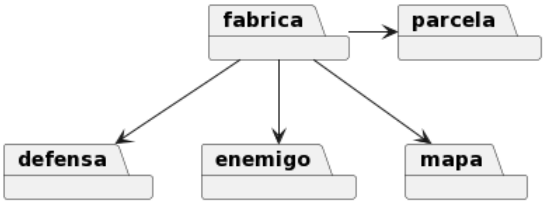

Figura 20: Diagrama de Paquetes - fábrica.

7.2. Mapa y Parcela

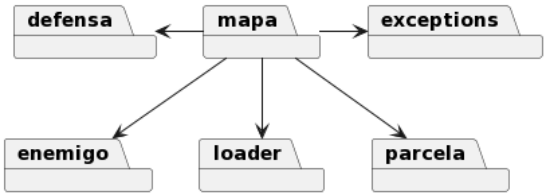

Figura 21: Diagrama de Paquetes - mapa.

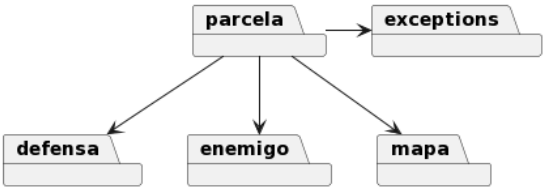

Figura 22: Diagrama de Paquetes - parcela

7.3. Defensa y Enemigo

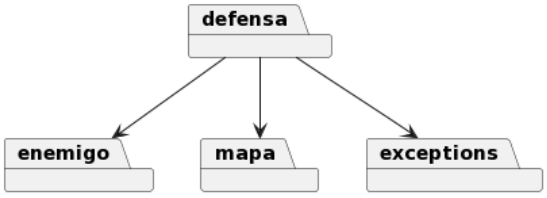

Figura 23: Diagrama de Paquetes - defensa

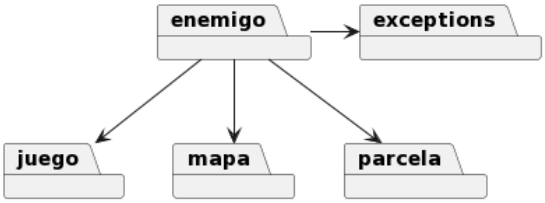

Figura 24: Diagrama de Paquetes - enemigo.

7.4. Loaders y utilidades

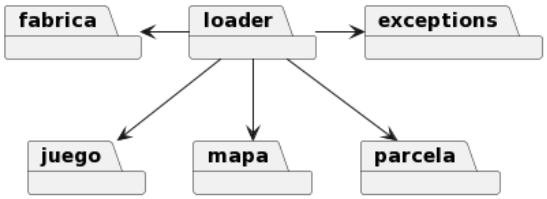

Figura 25: Diagrama de Paquetes - loader.

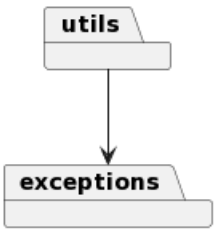

Figura 26: Diagrama de Paquetes - utils.

8. Excepciones

8.1. Defensa

DefensaEnConstruccion Se utiliza cuando una defensa que se encuentra en construcción intenta atacar a un enemigo. Se la atrapa en el método pasarTurno en Torre y en TrampaArenosa. No se toma ninguna acción al respecto.

FueraDeRango Se utiliza cuando una Torre intenta atacar a un enemigo que está fuera de rango. Se la atrapa en el método atacarEnemigos de la clase Terminada. No se toma ninguna acción al respecto.

8.2. Enemigo

ElEnemigoEstaMuerto Se la utiliza cuando una defensa intenta atacar a un enemigo que está muerto. Se la atrapa en el método atacarEnemigos de la clase Terminada. No se toma ninguna acción al respecto.

ElEnemigoMurioDuranteElAtaque Esta excepción se utilizó durante los tests para verificar que un enemigo atacado haya muerto.

8.3. Loaders

FormatoEnemigoInvalido Es utilizada para indicar que el formato del archivo Json de enemigos es inválido. No se la atrapa.

FormatoJsonInvalido Es utilizada para indicar que el formato del archivo Json es inválido, y se utilizó durante los tests. No se la atrapa.

FormatoMapaInvalido Es utilizada para indicar que el formato del archivo Json de mapa es inválido. No se la atrapa.

8.4. Parcela

NoEsPosibleRecibirEnemigosEnParcela Se la utiliza en las clases Rocoso y Tierra, ya que esas parcelas no deberían poder recibir un enemigo, a excpeción de la Lechuza.

NoSePudoBorrarElEnemigo Se utiliza al intentar mover un enemigo a una parcela ocupada o una en la cual el enemigo no pueda ser recibido. Se la atrapa en el método avanzarEnemigo, y se imprime el mensaje "No se pusdo borrar enemigo".

NoSePudoConstruir Indica que la defensa no se pudo construir en la parcela, ya sea debido a que la parcela se encontraba ocupada o que se intente construir sobre el tipo de parcela incorrecto. Se la atrapa en el controlador ÇontroladorMouseDraggedz no se toman acciones al respecto.