Trabajo Práctico 2 — AlgoDefense

[7507/9502] Algoritmos y Programación III
Curso 2
Primer cuatrimestre de 2023

Grupo 2

Cusihuaman Altagracia, Luis Eduardo - lcusihuaman@fi.uba.ar - 101805
Giacobbe, Juan Ignacio - jgiacobbe@fi.uba.ar - 109866
Fernandez Boch, Valeria - vfernandezb@fi.uba.ar - 103512
Olaran, Sebastian - solaran@fi.uba.ar - 109410

Docente corrector:
Villores, Alejo

Índice

1. Introducción

El presente informe recopila la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III. Nuestro objetivo principal fue desarrollar una aplicación completa, que incluyera un modelo de clases, efectos de sonido apropiados y una interfaz gráfica intuitiva. Este trabajo se realizó de manera grupal, aplicando de manera integral todos los conceptos aprendidos durante 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

Es importante destacar que la realización de este trabajo práctico se basó en una serie de supuestos previamente establecidos. Estos supuestos se tomaron en consideración durante todas las etapas del desarrollo de la aplicación, desde la planificación inicial hasta la implementación final. A continuación, se detallan los supuestos fundamentales que orientaron nuestro trabajo:

- Se asumió el supuesto de que después del turno 15 no aparecerán más enemigos en el juego. Esta decisión se tomó con el objetivo de brindar una experiencia de juego equilibrada y garantizar que la partida tenga un final determinado dentro de un límite de tiempo establecido.
- Se asumió el supuesto de que cada terreno apto para defensas solo puede albergar una única defensa. Este supuesto se implementó con el objetivo de fomentar la planificación táctica y la toma de decisiones estratégicas por parte del jugador, asegurando un nivel adecuado de desafío en el juego.
- Se asumió el supuesto de que se nos proporcionará un archivo "mapa.jsonz un archivo .enemigos.json"válidos para iniciar la aplicación. La existencia y validez de estos archivos son necesarios para que la aplicación funcione adecuadamente y pueda generar el mapa y los enemigos correspondientes.
- Para el movimiento del topo asumimos que sólo puede moverse por pasarelas.

3. Detalles de implementación

Dentro del desarrollo del software, nos encontramos con problemas al implementar métodos relacionados con las acciones de mover y atacar por parte de los enemigos. Identificamos que los diferentes enemigos tenían comportamientos distintos al moverse y atacar, dependiendo del tipo de enemigo.

Para abordar este problema, aplicamos el Patrón Strategy, siguiendo el principio de diseño conocido como Çomposición sobre Herencia". Este principio promueve el uso de la composición de objetos en lugar de heredar de una clase existente para reutilizar funcionalidades y construir relaciones entre objetos.

Al utilizar el Patrón Strategy, encapsulamos las diferentes formas de moverse y atacar que tenían los enemigos. Esto nos brindó una mayor flexibilidad y modularidad en la implementación. En casos específicos, como el enemigo "topoçuyo comportamiento de movimiento cambia con la cantidad de turnos, y el enemigo "lechuzaçuyo comportamiento de vuelo depende de su cantidad de vida, el Patrón Strategy nos permitió adaptar fácilmente sus comportamientos sin tener que modificar la estructura general del código.

En cuanto al ataque, cada tipo de enemigo manifestaba su ataque de diferentes maneras. Mediante la encapsulación de comportamientos de ataque, implementamos el método correspondiente para cada enemigo, respetando el principio de Composición sobre Herencia.

Además, aplicamos el Patrón Strategy en las defensas. Utilizamos la encapsulación de comportamientos para atacar a los enemigos dependiendo de si las defensas estaban desplegadas o no.

Esto simplificó el cambio de estado de las defensas y favorece la flexibilidad y modularidad del código.

El principio de Composición sobre Herencia fue preferido en este caso debido a sus ventajas. Al optar por la composición, logramos un código más flexible, mantenible y extensible. Redujimos el acoplamiento entre las clases, permitiendo modificar y extender comportamientos específicos sin afectar otras partes del sistema. Además, la composición nos permitió modularizar adecuadamente los comportamientos y mantener un orden claro en el proyecto.

En resumen, al aplicar el Patrón Strategy y seguir el principio de Composición sobre Herencia, mejoramos la estructura y flexibilidad de nuestro software. Esto nos permitió resolver los problemas encontrados en el dominio, facilitando su mantenimiento y permitiendo futuras extensiones de manera más eficiente.

Por otro lado, hemos implementado el patrón de diseño Observer en las vistas para establecer una relación de dependencia uno a muchos entre los objetos observables (Algodefense) y los observadores (vistas). Al aplicar el patrón Observer, hemos logrado separar la lógica de actualización de las vistas de la lógica del modelo de datos. Este enfoque nos ha permitido mantener nuestras vistas actualizadas ,reflejando los cambios realizados en el modelo de datos sin la necesidad de intervención directa. Al utilizar el patrón de diseño Observer, hemos logrado una arquitectura de software más modular, desacoplada y mantenible.

Por último, hemos aplicado el patrón de diseño State para abordar el problema de verificar si las defensas están desplegadas o no. Al utilizar el patrón State, hemos logrado separar el comportamiento del sistema en función de su estado actual de las defensas.

Al utilizar el patrón de diseño State, hemos logrado un diseño modular y desacoplado en nuestro software para verificar el estado de las defensas. Esto nos brinda flexibilidad en el manejo de los comportamientos relacionados con las defensas, lo que nos facilitó el mantenimiento del código.

4. Excepciones

TorreNoDesplegada: Esta excepción fue creada con el objetivo de cuando una defensa no esté desplegada e intente atacar,no pueda y notifique por terminal

DefensasVacias: Esta excepción fue creada con el objetivo de evitar que se intente eliminar una defensa de una lista que está vacía,y que cuando lo haga que el juego continúe

FormatoJSONInvalido: Esta excepción fue creada con el objetivo de notificar que el json entregado tiene un formato invalido .

EnemigoNoDaniable: Esta excepción fue creada con el objetivo de notificar que el enemigo que se intento atacar,no puede recibir daño.

EnemigoFueraDeRango: Esta excepción fue creada con el objetivo de notificar que el enemigo que se intento atacar,esta fuera del rango de ataque de un defensa.

5. Diagramas de clase

Se ocultaron los métodos para dar claridad al diagrama al momento de leerlo.

5.1. Diagramas generales

Figura 1: Diagrama general de la implementación.

Este diagrama de clase ofrece una representación general de las relaciones entre las clases del trabajo. Muestra cómo las diferentes clases se conectan entre sí y las interfaces que implementan.

A continuación,detallaremos los atributos, métodos y relaciones específicas que no se muestran aquí.

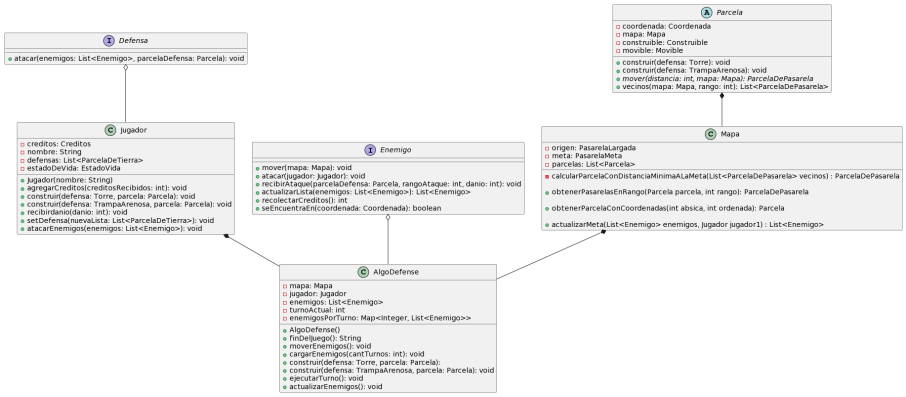

Figura 2: Diagrama general de la implementación detallado.

En resumen, el modelo de dominio de AlgoDefense se encarga de gestionar el mapa, los enemigos, las defensas y el progreso del juego, así como de notificar cambios a los observadores y generar registros.

5.2. Parcelas

Se adjunta una versión detallada con los métodos principales del diagrama de clases de las Parcelas en Algo Defense:

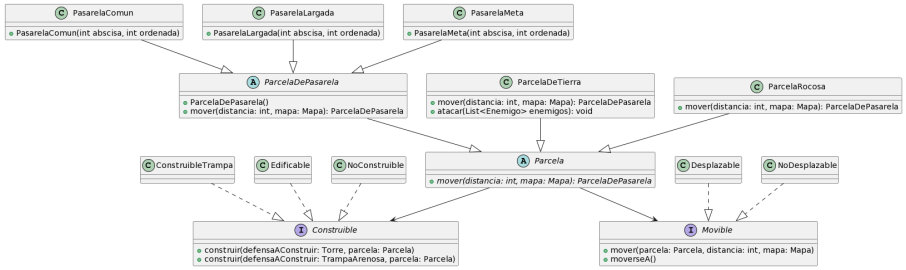

Figura 3: Diagrama Parcelas.

5.3. Enemigos

Se adjunta una versión detallada con los métodos principales del diagrama de clases de las Enemigos en Algo Defense:

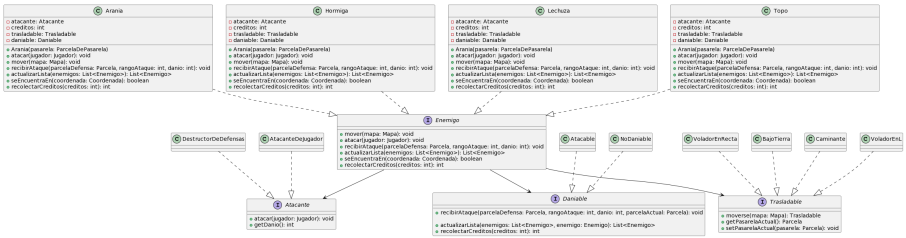

Figura 4: Diagrama de Enemigos.

5.4. Defensas

Se adjunta una versión detallada con los métodos principales del diagrama de clases de las Defensas en Algo Defense:

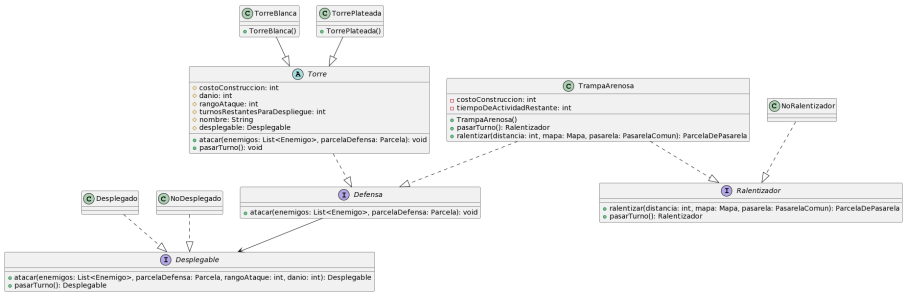

Figura 5: Diagrama de Defensas.

6. Diagramas de secuencia

6.1. Ataque del jugador

En esta serie de diagramas se desarrolla el ataque del jugador, detallando en diagramas más específicos ambas implementaciones de la abstracción Desplegable.

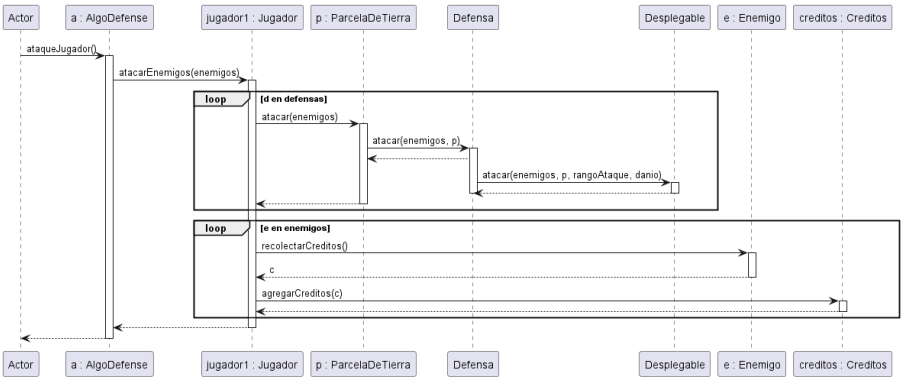

Figura 6: Se detalla el ataque del jugador hacia los enemigos.

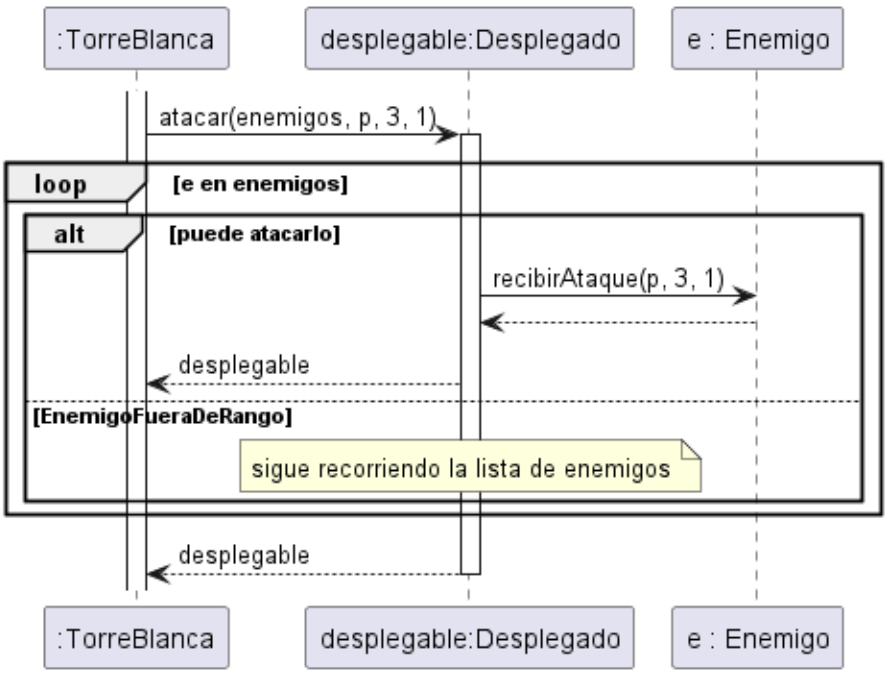

Figura 7: Respuesta de las defensas desplegadas al mensaje de atacar.

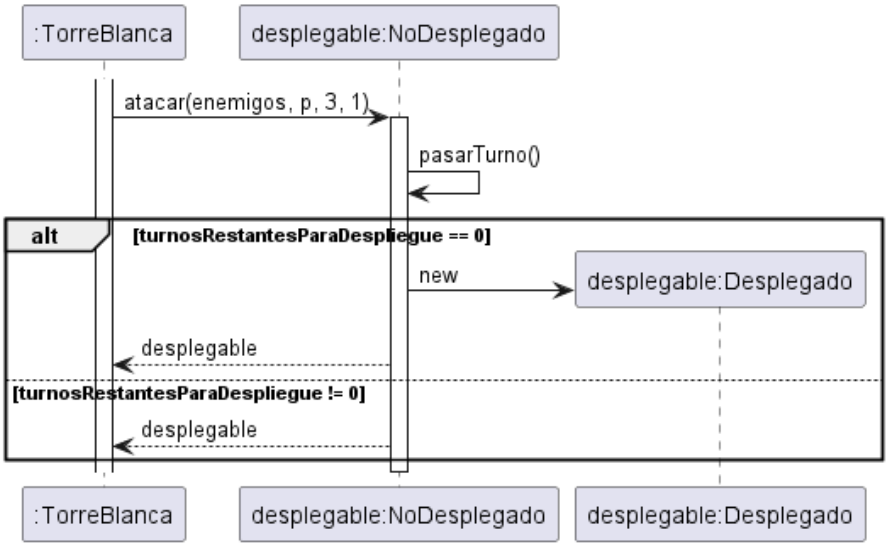

Figura 8: Respuesta de las defensas no deplegadas al mensaje de atacar.

6.2. Construir torres

En estos diagramas se detalla la construcción de las torres, mostrando las distintas alternativas de la secuencia si hay excepciones (detallando a qué excepción corresponde cada secuencia). También se adjunta la secuencia de construir para una ParcelaDeTierra y una ParcelaRocosa a fin de ver el comportamiento diferente que poseen.

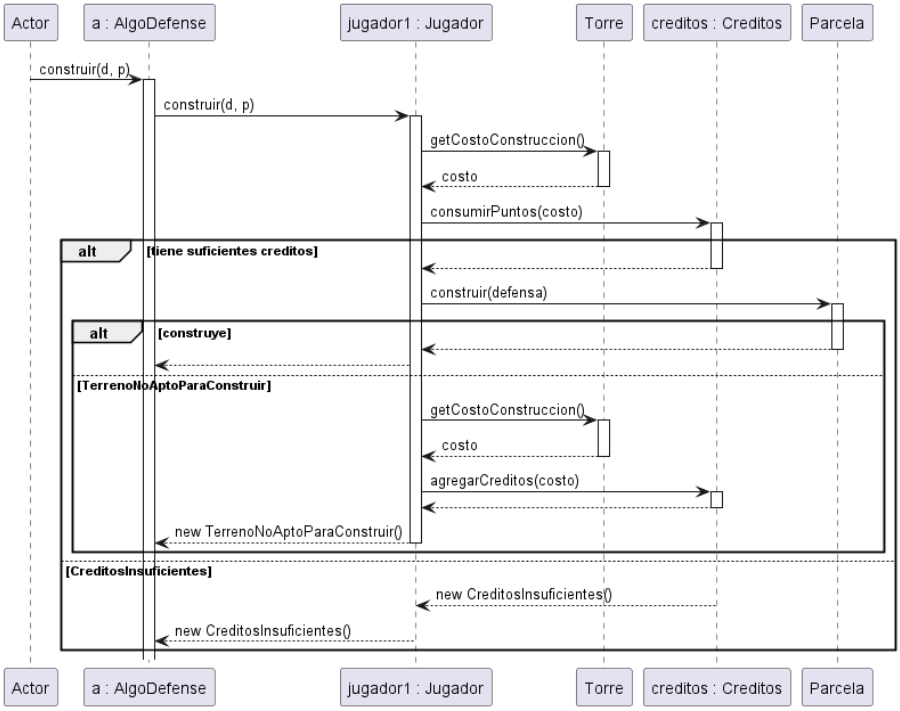

Figura 9: Interacción de AlgoDefense con las torres al enviarles el mensaje de construir.

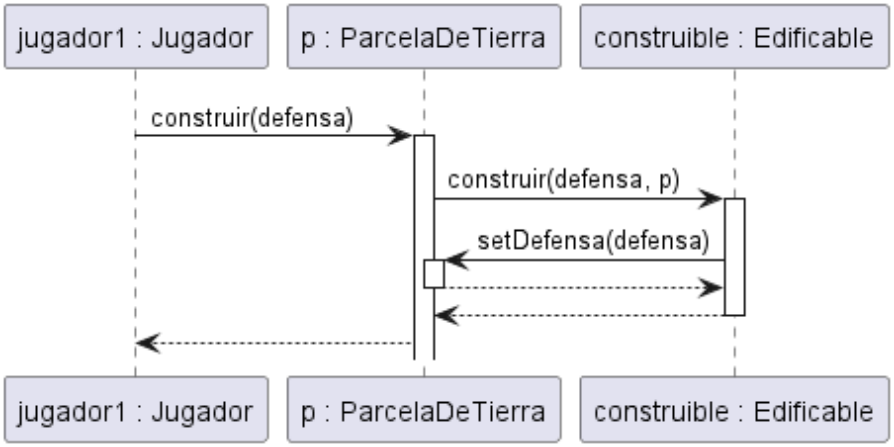

Figura 10: Se intenta construir una torre en una parcela edificable. No se termina en excepción.

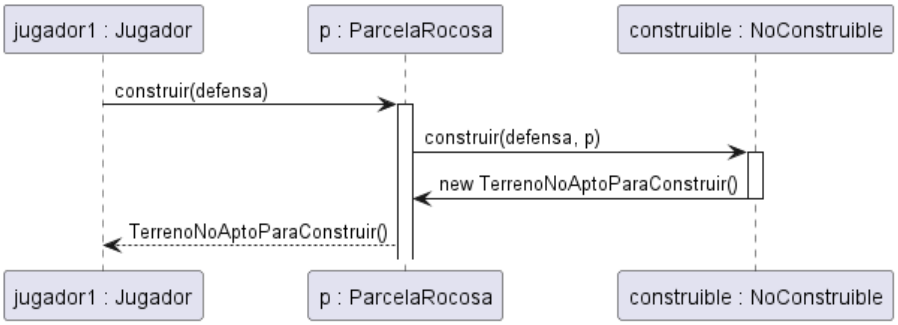

Figura 11: Se intenta construir una torre en una parcela no edificable. Se termina en excepción.

6.3. Movimiento de la Lechuza

Se detalla la respuesta de la Lechuza ante el mensaje mover, junto a ambas implementaciones de Trasladable que utiliza.

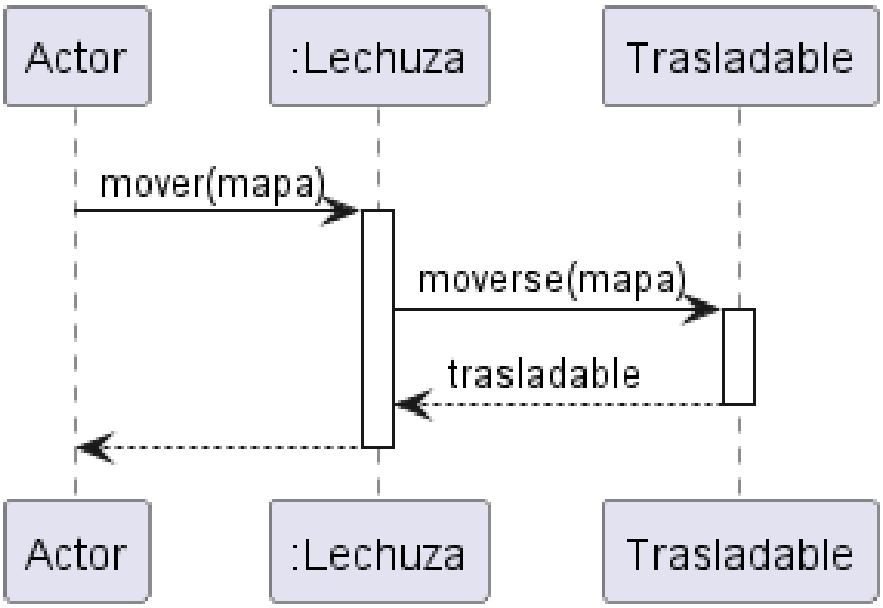

Figura 12: Se le enviía el mensaje mover a la Lechuza.

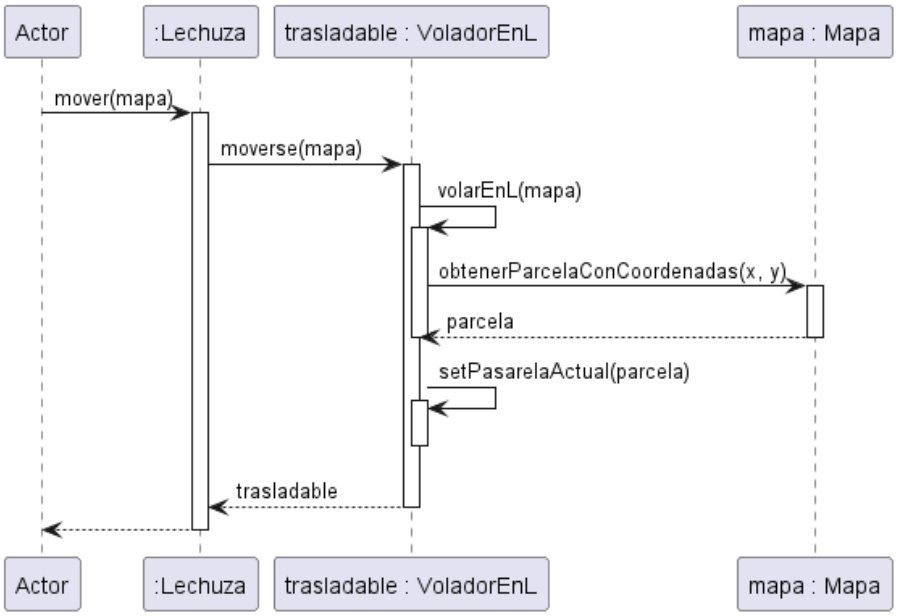

Figura 13: Se detalla la respuesta de la Lechuza ante el mensaje mover cuando esta se encuentra moviéndose en L.

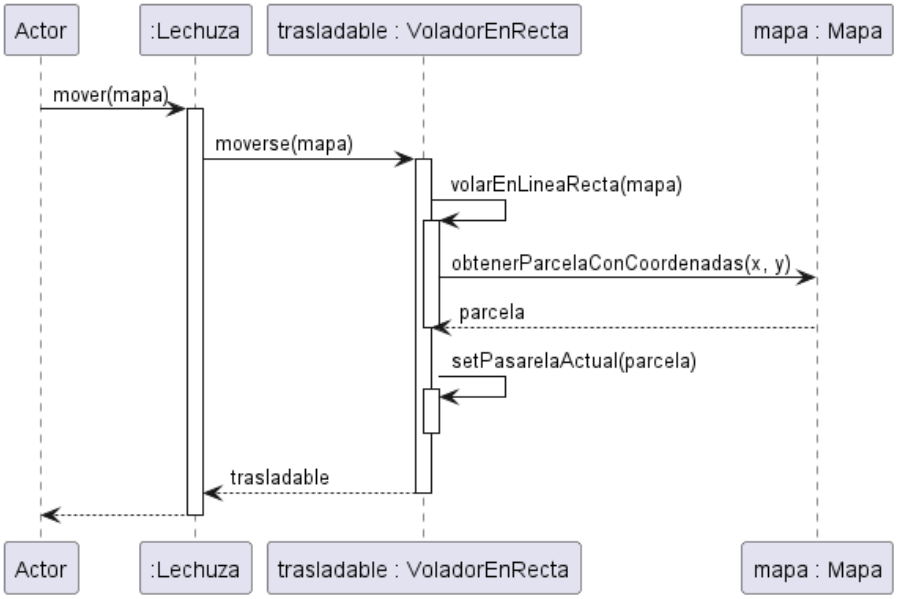

Figura 14: Se detalla la respuesta de la Lechuza ante el mensaje mover cuando esta se encuentra moviéndose en recta.

6.4. Mover de la Hormiga

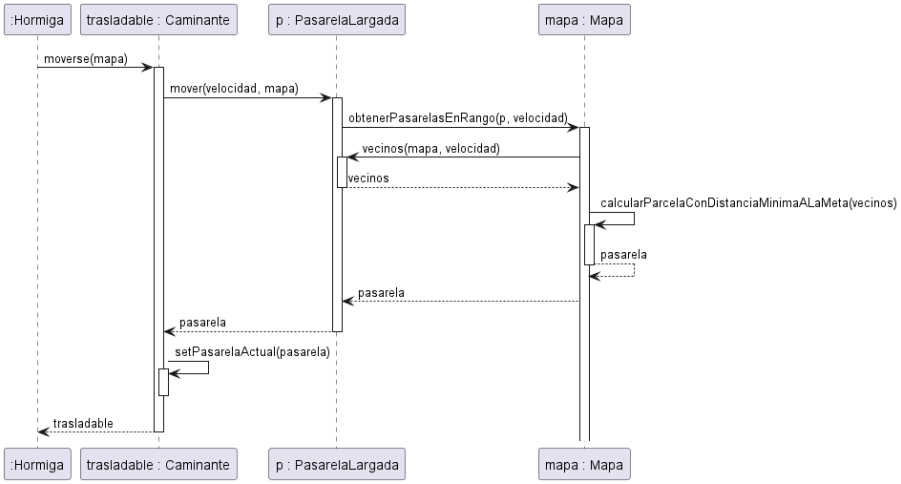

Figura 15: Se detalla la respuesta de la Hormiga ante el mensaje mover.

6.5. Movimiento de la Pasarela

Se adjuntan los diagramas de secuencia de PasarelaLargada y PasarelaComun a fin de observar cómo difieren en comportamiento.

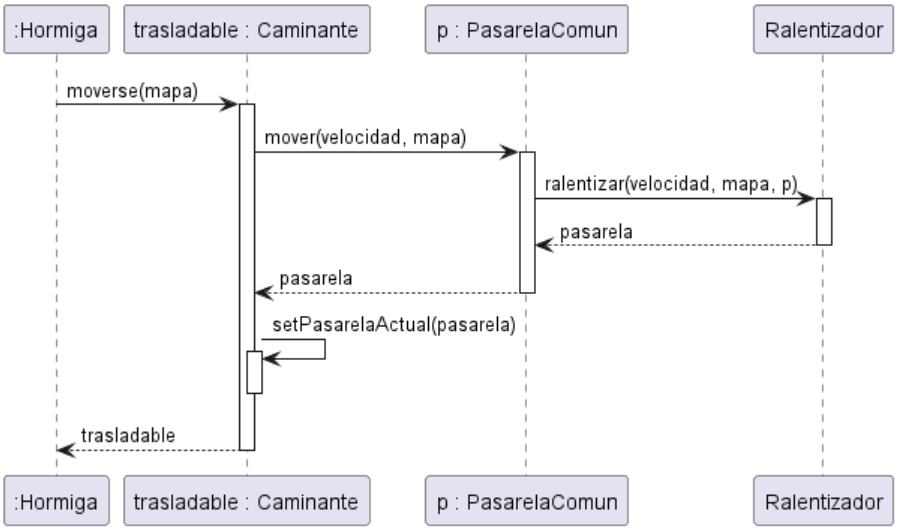

Figura 16: Se le envía a una PasarelaComun el mensaje de mover.

6.6. Mover de AlgoDefense

Se adjunta el diagrama de secuencia de moverEnemigos de AlgoDefense, donde se detalla la delegación del mover a los enemigos.

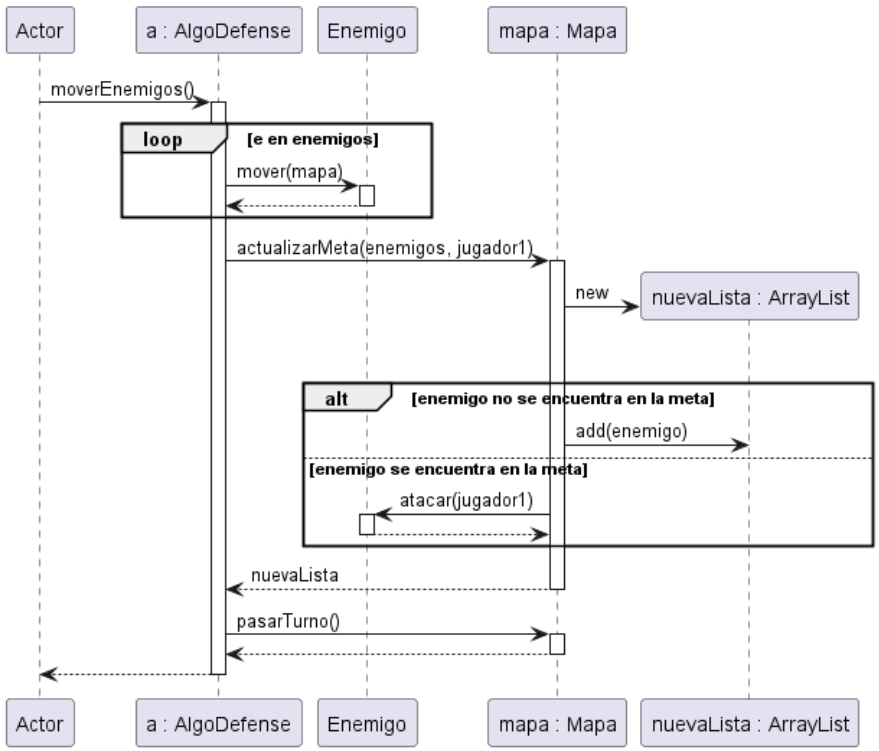

Figura 17: AlgoDefense le envía el mensaje mover a cada uno de los Enemigos que se encuentren en el Mapa.

6.7. Ataques de Enemigos al Jugador

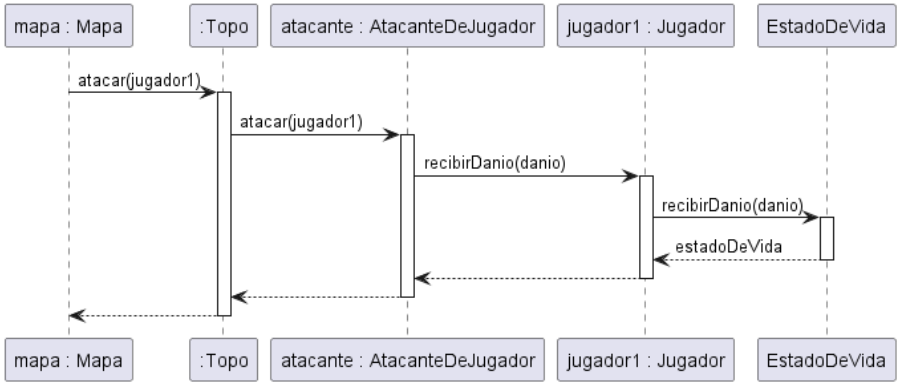

Figura 18: Un Enemigo(en este caso un Topo) ataca al Jugador.

7. Diagramas de paquetes

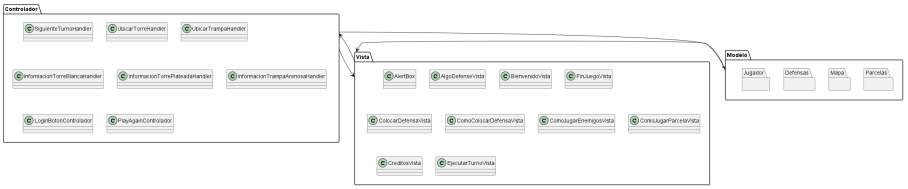

Figura 19: Un diagrama general del modelo de paquetes del Proyecto.

Se adjunta un diagrama de paquetes donde especificamos en nuestro modelo,para poder comprender la relacion del modelo con sus otras partes

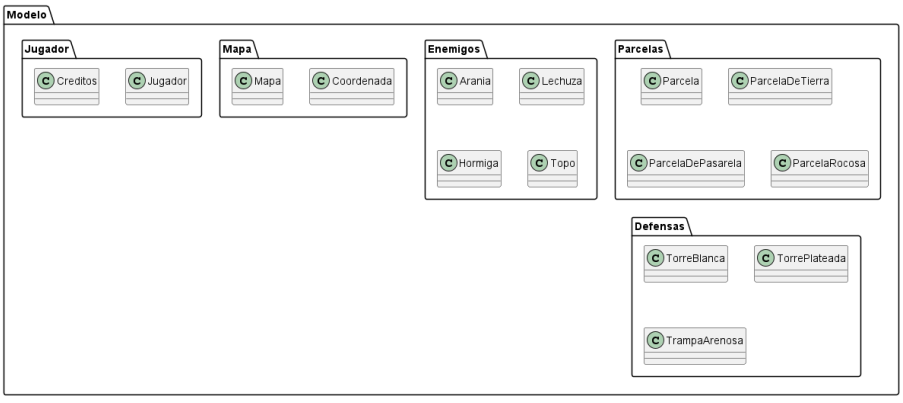

Figura 20: Diagrama de modelo detallado.

8. Diagramas de estado

8.1. Estado de Vida

Se adjunta un diagrama de estado de EstadoDeVida, para observar cuando se pasa de un EstadoVivo que es como se inicializa a un Estado Muerto, cuando muere un Enemigo o Jugador.

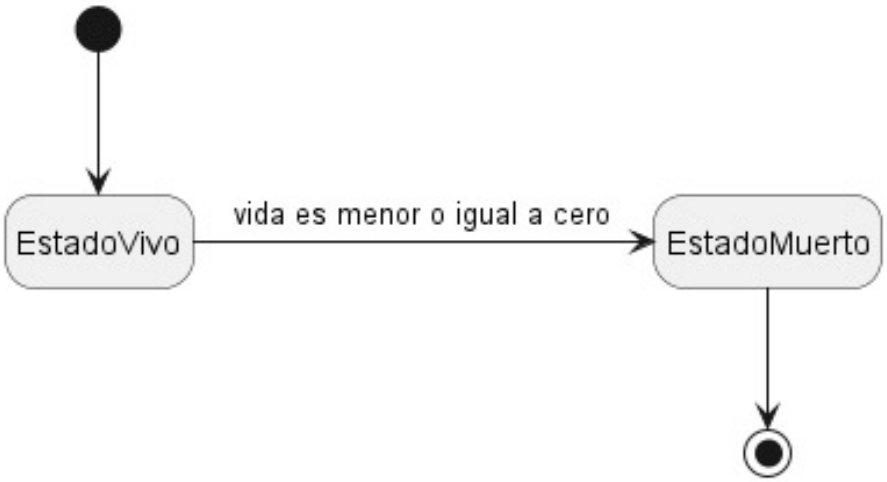

Figura 21: Diagrama de estado de EstadoDeVida.

8.2. Estado de Ralentizador

Se adjunta un diagrama de estado de Ralentizador, observando en qué ocasiones se pasa de un estado a otro. Siempre se inicializa en NoRalentizador.

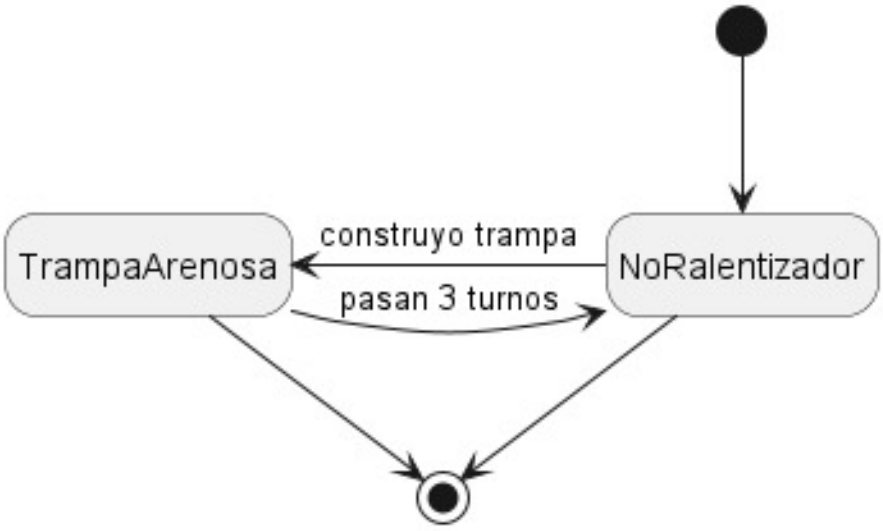

Figura 22: Diagrama de estado de Ralentizador.