FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Algo-Defense

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

Alumno	Número de padrón	Email
Lucas Williams	103018	lwilliams@fi.uba.ar
Ariel Folgueira	109473	afolgueira@fi.uba.ar
Agustín de la Rosa	108275	adelarosa@fi.uba.ar
Carlos Orqueda	101806	corqueda@fi.uba.ar
Manuel Mendoza	107422	mamendoza@fi.uba.ar

Índice

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo practico de la materia de Algoritmos y Programación III que consiste en desarrollar una aplicación relacionada al famoso juego Tower Defense, sin embargo esta versión no es en tiempo real, si no por turnos. Nuestro objetivo es aplicar los conceptos explicados a lo largo de la materia, utilizando un lenguaje de tipado estático llamado Java, con un diseño del modelo orientado a objetos y trabajando con las técnicas de TDD e Integración Continua. El objetivo es desarrollar la aplicación completa, incluyendo el modelo de clases, sonidos e interfaz gráfica. La aplicación sera acompañada por pruebas unitarias e integrales y documentación de diseño.

2. Supuestos

2.1. Supuesto 1

El usuario final de la aplicación la ejecutará con los archivos json que posean el formato correcto.

2.2. Supuesto 2

El topo nunca recibe daño porque no sale de la tierra

2.3. Supuesto 3

Las torres atacan a todos los enemigos en su rango

2.4. Supuesto 4

El rango de las torres es la distancia entre una parcela objetivo y la parcela donde esta la torre

2.5. Supuesto 5

El jugador tiene una pantalla de 1920x1080 pixeles

3. Diagramas de clase

Un diagrama de clase es un diagrama estático en el cual se representa la estructura de un sistema compuesto por clases, reflejando así sus atributos, métodos y las relaciones con otros objetos. A continuación se presentan algunos diagramas de clase correspondientes al trabajo practico.

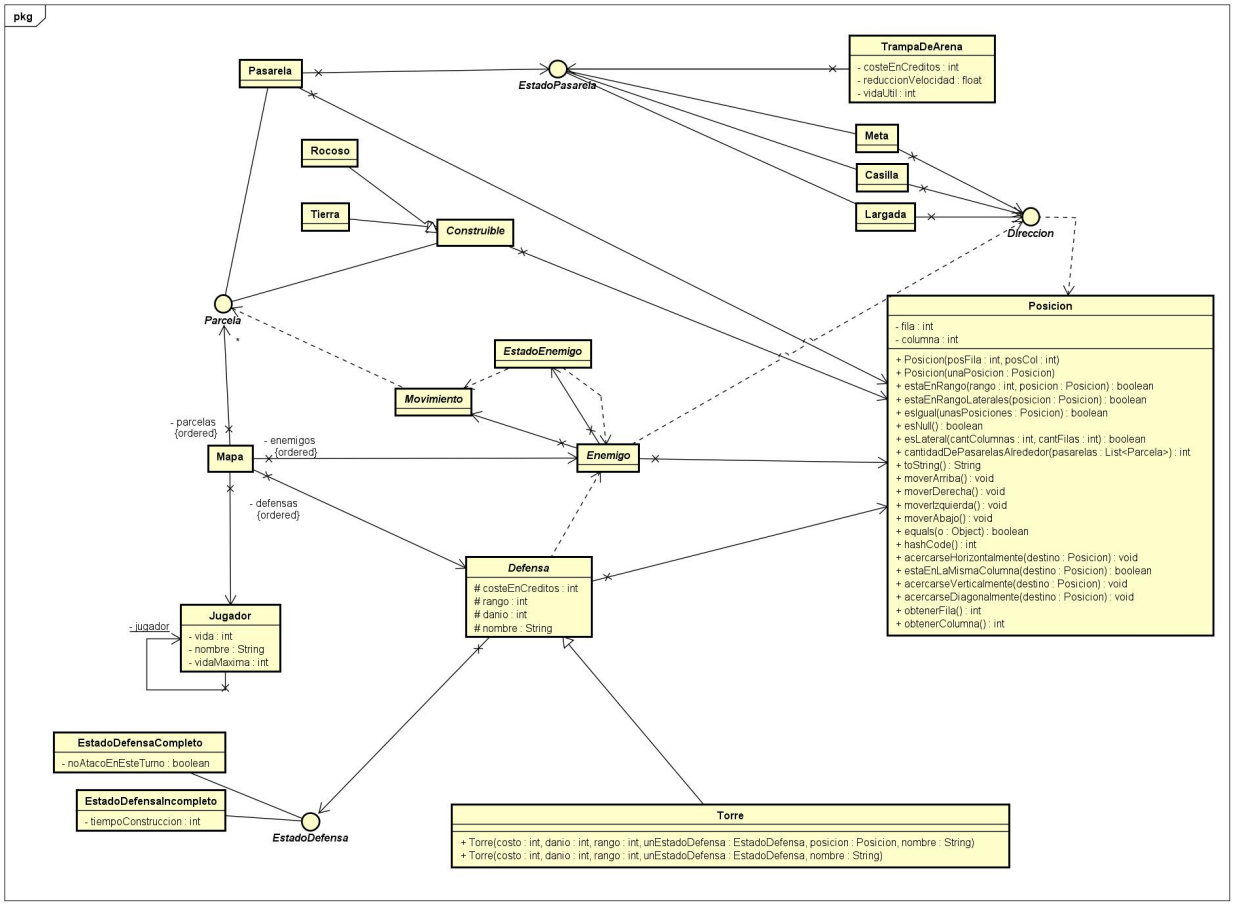

Figura 1: Diagrama clases principal (capaz no se ve muy bien pero la idea es mostrar la interacción con posición).

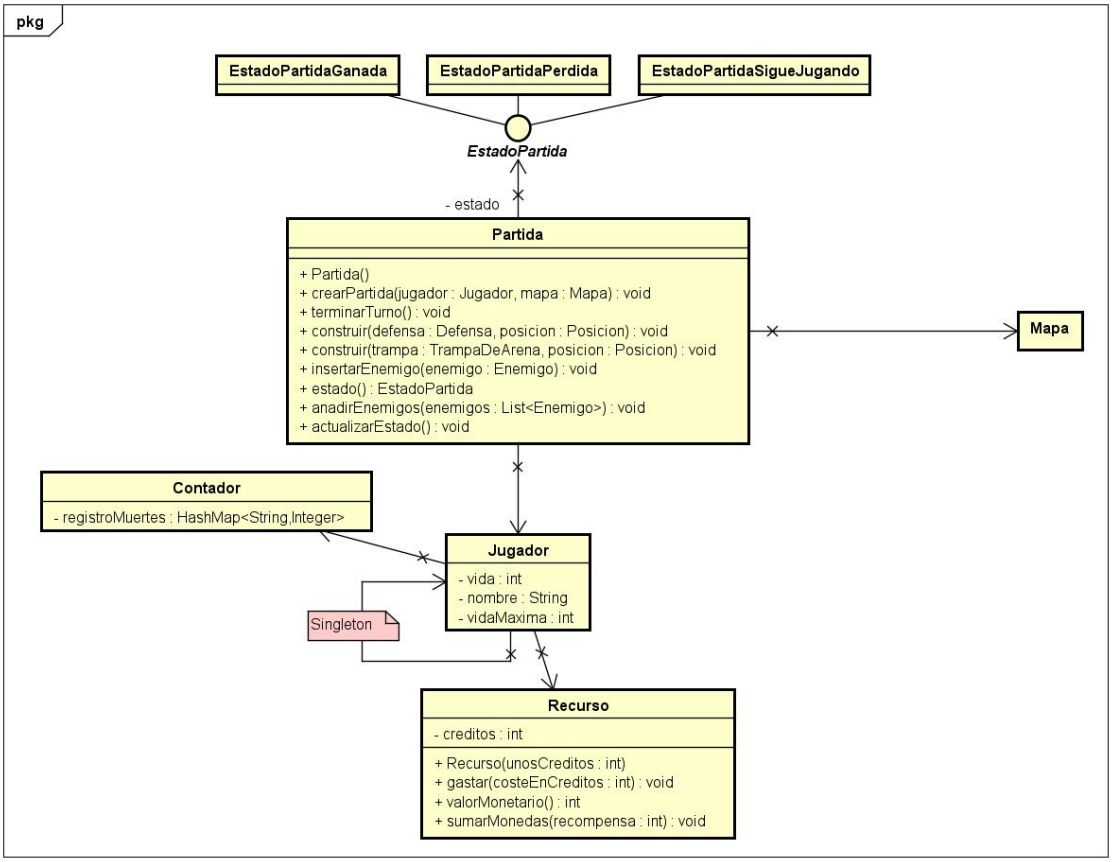

Figura 2: Diagrama de funcionalidad de partida.

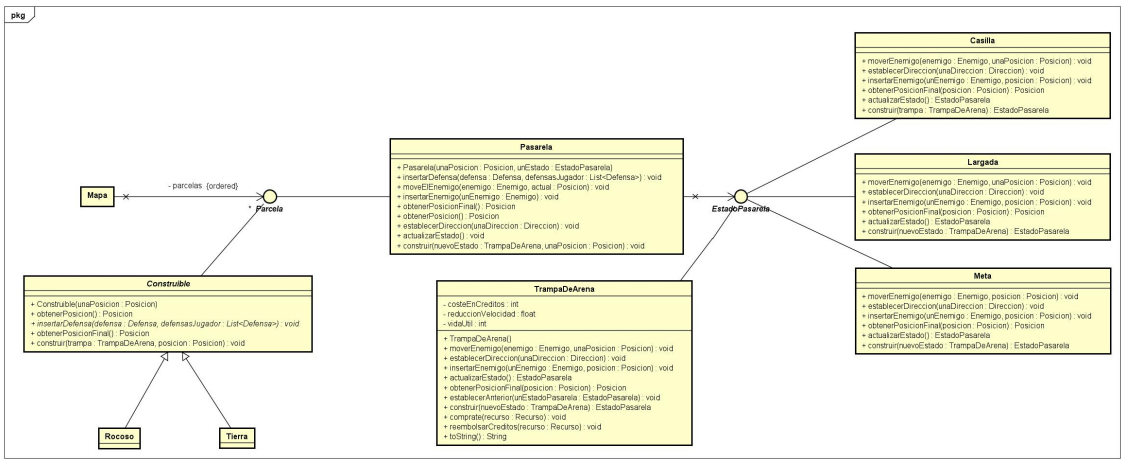

Figura 3: Diagrama 3 relaciones de parcela.

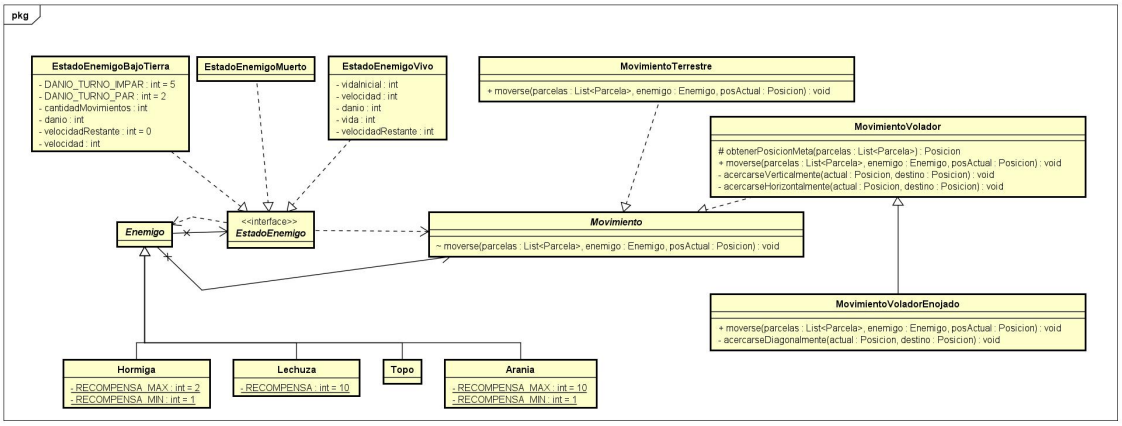

Figura 4: Diagrama 4 relaciones de enemigo.

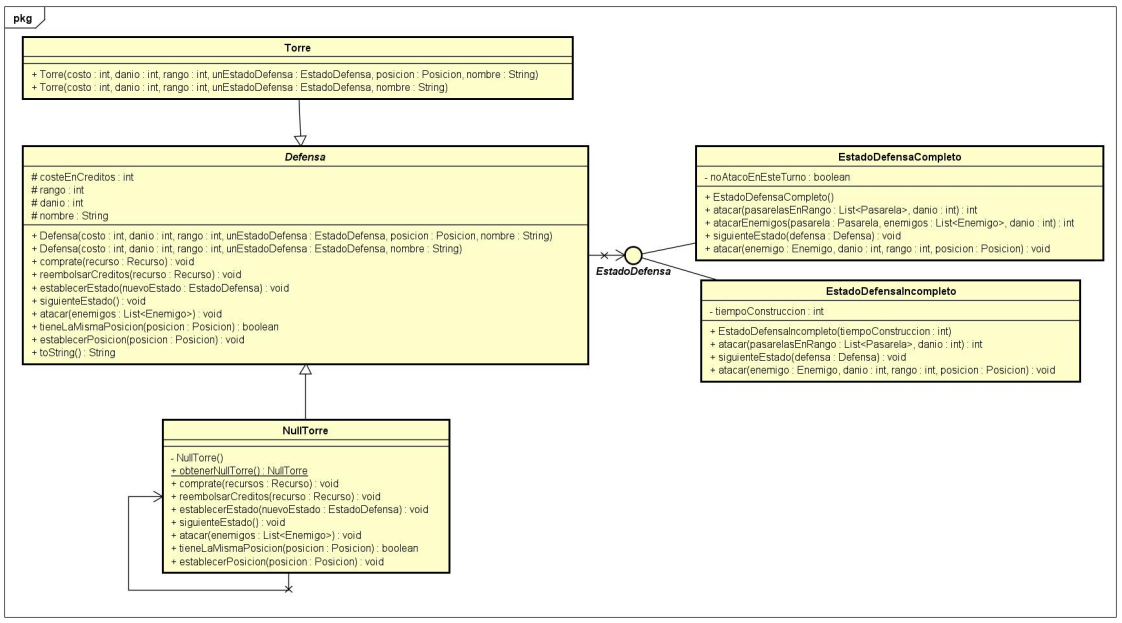

Figura 5: Diagrama 5 relaciones de Defensas.

4. Diagramas de secuencia

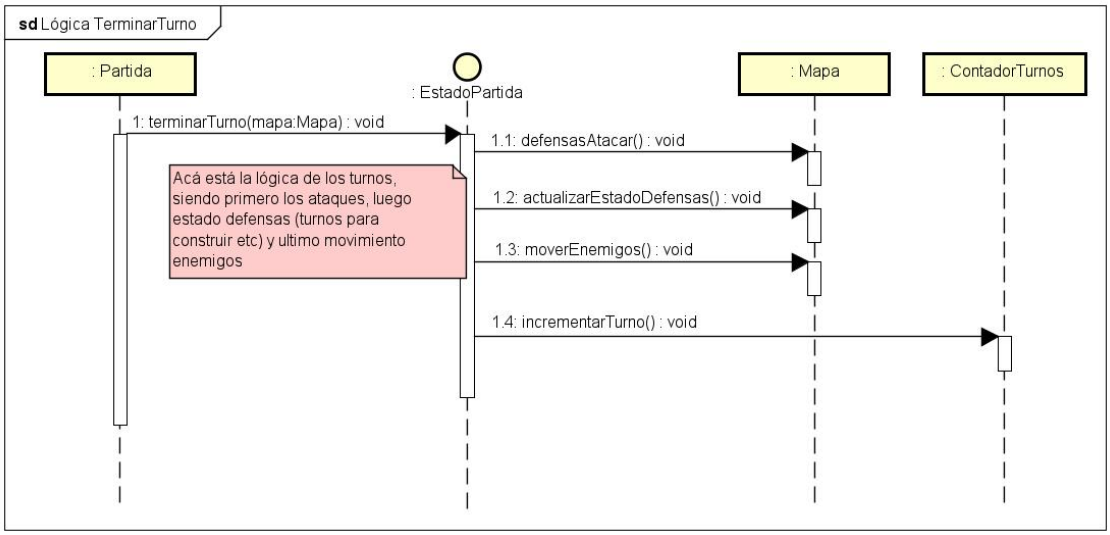

Figura 6: Secuencia de Terminar Turno.

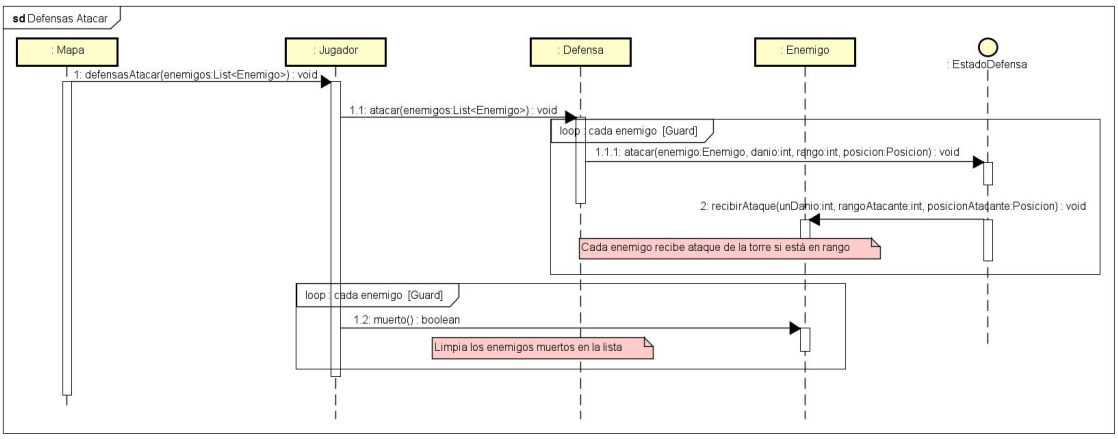

Figura 7: Secuencia de ataque de defensas.

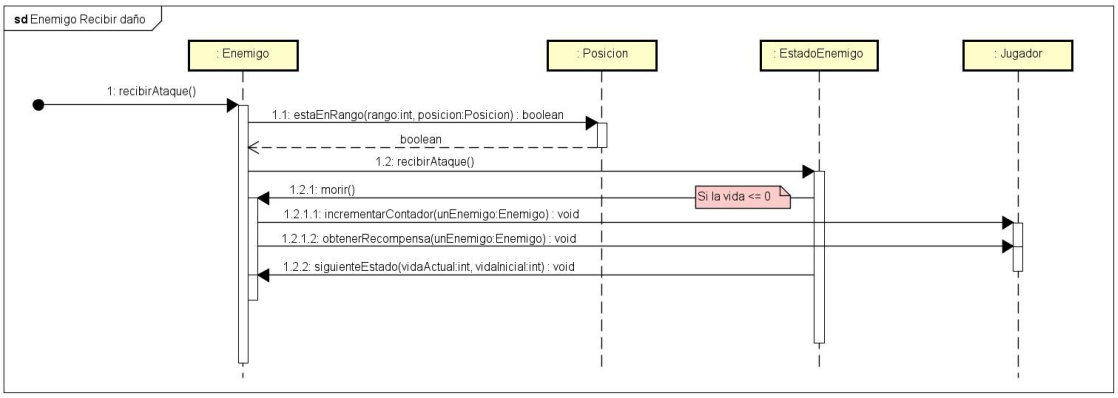

Figura 8: Secuencia del enemigo recibiendo daño.

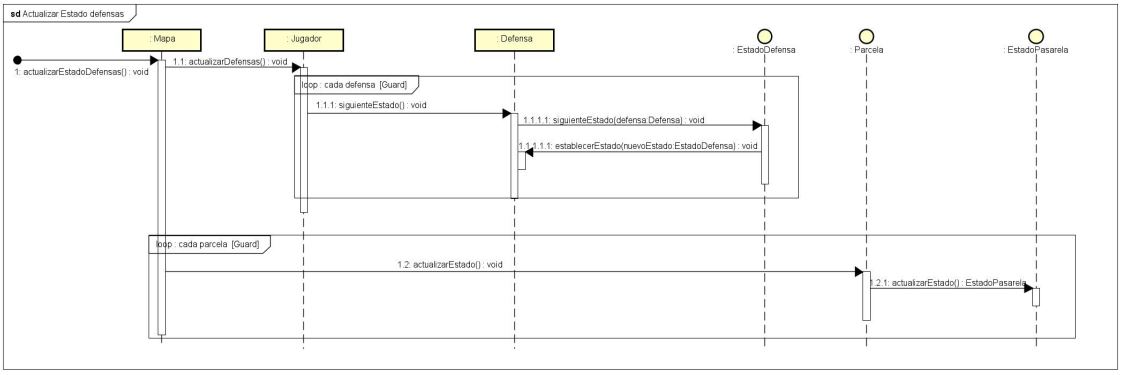

Figura 9: Secuencia de la actualizacion del estado de las defensas.

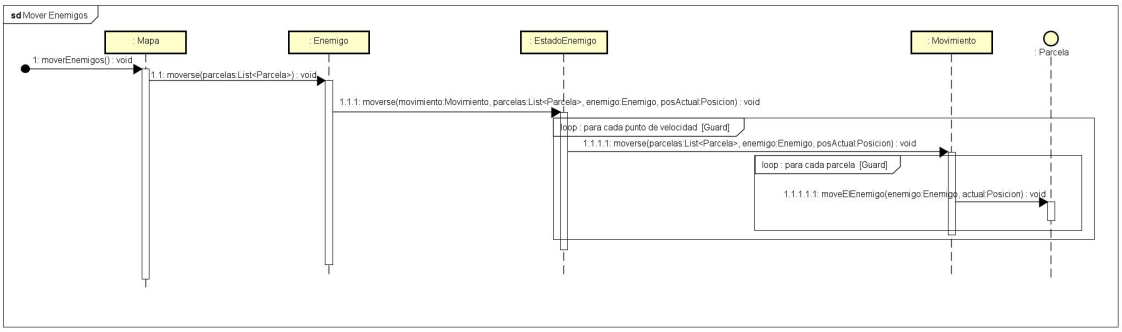

Figura 10: Secuencia del movimiento de los enemigos.

5. Diagramas de paquetes

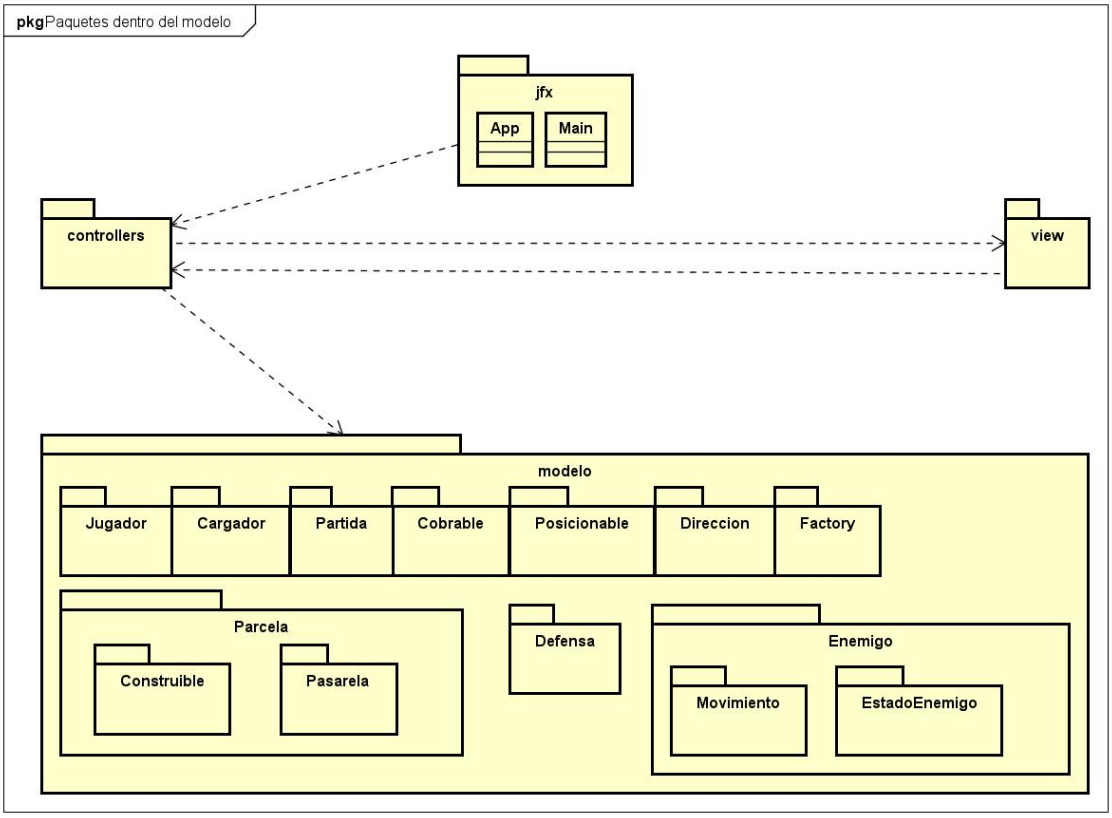

Figura 11: Diagrama de paquetes con su respectivos participantes al modelo, muestra el MVC.

6. Diagramas de estados

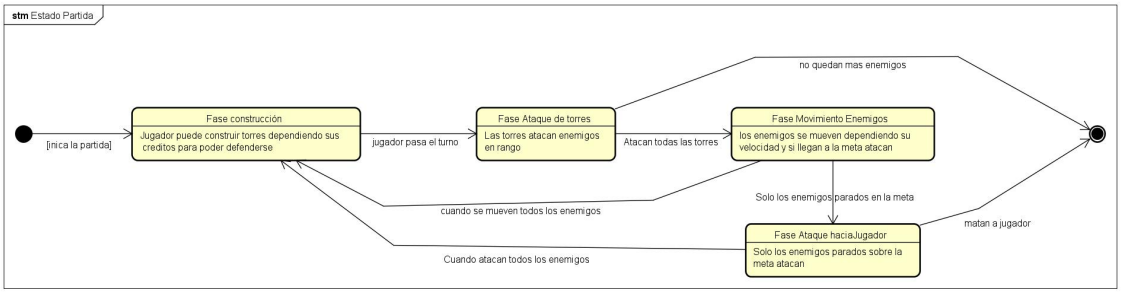

Figura 12: Diagrama de estado 1.

7. Detalles de implementación

7.1. Uso de herencia

Utilizamos herencia en las clases de Enemigos y de Construibles ya que las entidades que descienden de estas comparten todo el comportamiento y responden a .es un". Además crearlas sin herencia daría un resultado igual con mucha repetición innecesaria de código, el uso de esta herramienta se hizo respetando el principio de Liskov.

7.2. Uso de Double Dispatch

El patrón doble dispatch se utiliza una secuencia tal que

```
public void construir(Defensa torre) {
     partida.construir(torre);
 }

 public void construir(TrampaDeArena unaTrampa, Posicion unaPosicion){
     partida.construir(unaTrampa, unaPosicion);
 }
```

se instruye que mapa sepa construir dependiendo una defensa o una trampa de arena

7.3. Uso del patrón Fachada

El patrón fachada fue aplicado a la entidad Cargador para que el usuario se abstraiga del tipo de archivo utilizado para cargar los datos del juego, permitiendo así, de ser necesario, poder realizar cambios en el tipo de archivo sin tener que cambiar la implementación del controlador del juego.

7.4. Uso del patrón Singleton

Usamos el patrón singleton en las clases App, Juego, Jugador y Logger ya que son entidades que deben ser instanciadas una sola vez durante la ejecución del programa y sus funcionalidades son utilizadas a lo largo de toda la aplicación, lo que hace que sea mas práctico proveer un acceso global a estas antes que estar pasando a las mismas como parámetros.

7.5. Uso de polimorfismo implementando una Interfaz

```
EstadoPartida estado = new EstadoPartidaSigueJugando();
        Mapa mapa = obtenerMapaGenerico();
        Enemigo enemigo = new Hormiga(1,1,1);

        assertDoesNotThrow(() -> estado.insertarEnemigo(enemigo, mapa));
```

En los casos donde entidades de distintos tipo debían compartir un mismo mensaje polimórfico optamos por implementar interfaces lo cual permite a diversos objetos responder al mismo mensaje de manera adecuada según su estado actual, añadiendo así una capa de abstracción que encapsula la información de la entidad que recibe el mensaje y no afecta a la implementación de la entidad que envía el mensaje en caso de que se produzca un cambio a futuro.

7.6. Uso del patrón Observer

Utilizamos patrón observer en una parte del MVC para poder actualizar los datos del jugador setenado solamente una StringProperty sin tener que modificar la vista de manera directa.

7.7. Uso del patrón MVC

El uso del patrón MVC nos permite organizar estructuralmente el proyecto (ya que es un patrón de arquitectura). Consta de la lógica de negocio (M). Las view (V), las cuales consisten en todos los elementos tangibles por el usuario por ejemplo imágenes y sonido. Por último los controladores (C), son quienes tienen la lógica de las view.

8. Excepciones

Para las Excepciones se trató que su nombre error sea explicativo.

DefensaNoIdentificadaException Cuando no se encuentra la defensa en el sistema Factory (el cual recibe un string y posición), salta la excepción.

DefensaNoSePudoConstruir Ya sea por posición invalida, o coste no correcto con recursos del jugador.

EnemigoNoIdentificadoException Ocurre lo mismo que con defensa, ya que se utiliza patrón Factory.

EnemigosJsonParseException Error en la lectura del JSON usando parse de jsonSimple.

ParcelaNoIdentificadaException Ocurre lo mismo que con defensa, aya que se utiliza patrón Factory.

ParcelaNoPuedeContenerEnemigo Parcela no puede contener enemigo ya que no es una pasarela.

ParcelaNoPuedeContenerTrampa Parcela no puede contener trampa ya que no es una pasarela.

RecursosInsuficientesException Recursos insuficientes para comprar una defensa.