[75.07 / 95.02]
Algoritmos y programacion III

Trabajo practico 2: AlgoDefense

Estudiantes:

Nombre	Padron	Mail
Otero Silvera Maximiliano	108634	motero@fi.uba.ar
Pla Tomas Alberto	106705	tpla@fi.uba.ar
Oviedo Ignacio Sebastian	109821	ioviedo@fi.uba.ar
Fernandez Lucas	109250	lufernandez@fi.uba.ar
Velurtas Joaquin Mateo	109655	jvelurtas@fi.uba.ar

Tutor: Diego Sanchez

Supuestos

Largada y Meta

Se tomó a la primera pasarela registrada como la Largada y a la última en registrarse como la Meta.

TrampaArenosa y Topo

Un topo al pasar por una TrampaArenosa cambia su estado EsSubterraneo a NoEsSubterraneo y resulta vulnerable ante las torres.

Parcela y defensa destruida

Al destruirse una defensa, la parcela queda ocupada por la misma en su estado destruido por lo cual esa parcela queda ‘inutilizada’.

Estadísticas no mencionadas

Algunas estadísticas no mencionadas en el enunciado (ej. energía de Topo) se tomaron arbitrariamente.

Ataques de TrampaArenosa

La TrampaArenosa resultaba muy complicada de utilizar ya que para poder ralentizar un enemigo este justo debia caer en la parcela en la cual estaba colocada la misma. Por lo cual decidimos en que si el enemigo al moverse pasa por la trampa este se vea ralentizado.

Detalles de implementacion

Modelo

El modelado para la solución de este problema consta de una clase Partida, con la cual interactúa el usuario a través de la GUI. Además de delegar a la clase Turno para que el juego actúe, la Partida está compuesta por un objeto Mapa y Turno, dependiendo de CreadorMapa para crear el mapa.

Mapa compuesto de parcelas, delega a la Parcela que le indica Partida la construcción de una Defensa, dependiendo de qué clase de Defensa y que clase de Parcela se trate se aceptará la defensa o se arroja una excepción. En caso de que esta defensa se construya correctamente, se agregara a la lista defensas de turno.

Parcela tiene una Posición para saber en qué parte del mapa está ubicada, además es una abstracción la cual heredan Tierra, Pasarela, Meta, Largada y Rocoso cumpliendo la relación ‘es un’. Las parcelas se encargan de contener los enemigos que se encuentran en tal posición.

En el caso de Tierra y Pasarela pueden aceptar una defensa (TorreBlanca (y sus subclases) y TrampaArenosa respectivamente, en caso contrario se arroja una excepción) la construcción de la misma es delegada a Construible (interfaz la cual implementan EsConstruible y NoEsConstruible) [Concluyendo con esta delegación en un patrón de diseño State, se utilizó este patrón ya que este tipo de Parcela tienen dos estados construible y no construible, y nos permite resolver de forma polimórfica este caso de la construcción].

Por otro lado, Turno tiene un Jugador el cual está compuesto por Créditos al cual le delega todo lo relacionado a el dinero del juego, el Jugador en sí se encarga de recibir daño de Enemigo(s).

Turno tiene un Camino el cual lo utiliza para aparecer los enemigos y pasarselo a Enemigo(s). Para saber qué enemigos aparecer Turno depende de CreadorEnemigos el cual requiere el número del turno para saber que tipo de enemigos y que cantidad crear.

Camino está compuesto por Parcela(s), generalmente Pasarela(s) aunque hay excepciones las cuales utiliza un tipo de enemigo (Lechuza). Camino se encarga de delegar a las parcelas la administración de los enemigos, por su parte sabe mover a los enemigos a la parcela indicada.

Turno además, tiene Enemigo(s) y Defensa(s) las cuales les indica en cada turno que deben actuar.

Defensa es una abstracción que delega a la interfaz SaludDefensa la administración por turno de la misma (a SaludDefensa la implementan Operativa y Destruida) [Este caso también es una implementación del patrón State ya que tenemos ambos estados de la Defensa y necesitamos resolverlos de manera polimórfica]. A Defensa la heredan TorreBlanca y TrampaArenosa.

TrampaArenosa delega a ObjetivoTrampa el buscar a qué objetivos atacar (lista de enemigos) y luego de tres veces turnos esta pasará a su estado Destruida. A TrampaArenosa la hereda NoTrampa [Patron NullObject, ya que cuando se agrega un enemigo a una pasarela la pasarela le envia el mensaje a la trampa de que debe atacar y tal vez no hay una trampa construida en tal pasarela]

TorreBlanca delega a ObjetivoTorre el buscar a qué objetivo atacar (el más cercano y en rango a la misma), antes de poder atacar la torre se encuentra en un EstadoDesactivado y luego de X turnos se cambia a EstadoActivado, ambos estados saben administrar por turno que hacer [Otro caso de utilización del patrón State (interfaz de nombre Estado)]. A TorreBlanca la heredan NoTorre [Patrón NullObject, ya que hay casos en los que se requiere una torre como objetivo pero puede que el jugador no haya construido una torre aún] y TorrePlateada (una mejora en estadísticas de TorreBlanca).

Por otro lado, Enemigo es una abstracción que utiliza el patrón State, al igual que las Defensa(s) la interfaz SaludEnemigo (que la implementan Vivo y Muerto), además implementa dos patrones Strategy, la interfaz Volador (que la implementan NoEsVolador y EsVolador) y la interfaz Subterráneo (que la implementan NoEsSubterraneo y EsSubterraneo). La razón de esta elección de patrones es resolver de forma polimórfica los casos.

A Enemigo lo heredan Arania, NoEnemigo [patrón NullObject, misma razón que para NoTorre], Hormiga, Topo y Lechuza.

Los enemigos para moverse delegan a su camino que los mueva tantas veces como velocidad les quede o lleguen a la meta. Por su parte los enemigos saben como les afectan las defensas del usuario (recibir danio o ser ralentizados).

Lechuza tiene asociado ObjetivoLechuza ya que le delega la decisión de a que defensa atacar. Lechuza para saber por dónde moverse depende de Ruta, una interfaz la cual implementan RutaL y RutaH. Clases las cuales dependen de CreadorCaminoL y CreadorCaminoH respectivamente para otorgarle a la lechuza el camino que requiere.

Por último, SingleLogger es el Logger del juego el cual implementa el patrón de diseño singleton para estar disponible globalmente a cambio de romper el principio de única responsabilidad.

Vista

Al momento de comunicar a la vista y los controladores con el modelo, para mantener mvc, se propuso aplicar el patrón de diseño de Observers sin embargo al explorar las opciones decidimos que sería mejor trabajar mediante getters en donde la vista o el controlador le preguntaran el estado al modelo cada vez que deba actualizarse. Si bien esto compromete el principio de “tell dont ask”, la alternativa de utilizar Observers nos pareció más confusa en el código y generaba conflicto con el principio de responsabilidad única al agregar la responsabilidad de implementar un observer a cada clase que podía actualizar su estado. Aun sabiendo que violar Tell dont ask es mayor a violar Single responsability nos guiamos por cual opcion era mas simple de desarrollar.

Excepciones

CreditosInsuficientesError

Esta excepción analiza el caso en el que los créditos a gastar superen a los créditos actuales, en caso afirmativo se lanza la excepción.

DefensaEnTerrenoErroneoError

Esta excepción analiza el caso en el que se quiera colocar una instancia de defensa de un tipo no compatible con la parcela, en caso afirmativo se lanza la excepción

EnemigoNoRalentizableError

Esta excepción la lanza un enemigo al querer ralentizarlo y no se pueda.

GanarPartidaError

Esta excepción analiza el caso en el que la lista de enemigos no tiene ningún enemigo restante, en caso afirmativo se lanza la excepción.

PerderPartidaError

Esta excepción analiza el caso en el que el jugador tiene vida menor o igual a cero, en caso afirmativo se lanza la excepción.

SpawnNoEnLargadaError

Esta excepción la lanzan todas las parcelas excluyendo la Largada al querer utilizar el método aparecerEnemigos().

TerrenoDeConstruccionInvalidoError

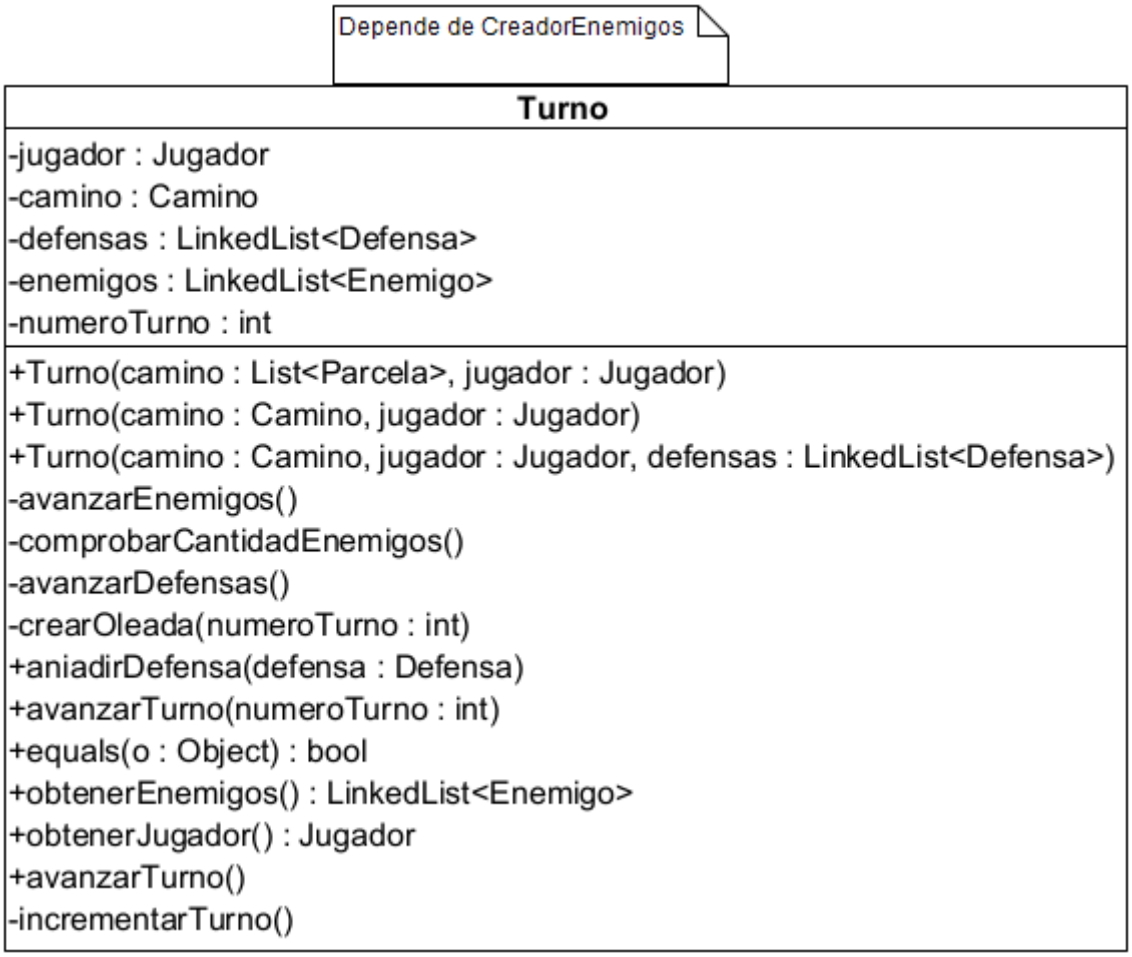
Esta excepción la lanzan todas las parcelas excluyendo Tierra y Pasarela al querer utilizar el método construir().

Diagramas de Clase

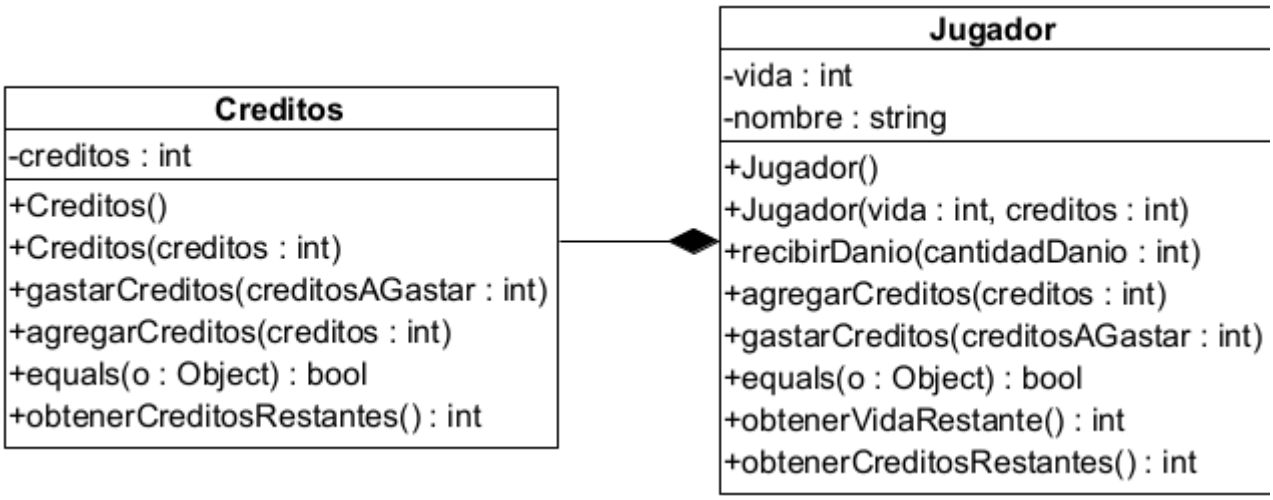
Partida

Partida
-turno : Turno -mapa : Mapa
+Partida(parcelas : List<List<Parcela>>, camino : List<Pasarela>) +Partida(parcelas : List<List<Parcela>>, camino : List<Pasarela>, jugador : Jugador) +iniciarJuego() +iniciar() +construirDefensa(defensa : Defensa, coordenadaX : int, coordenadaY : int) +equals(o : Object) : bool +avanzarTurno() +obtenerMapa() : List<List<Parcela>> +obtenerEnemigos() : LinkedList<Enemigo> +obtenerJugador() : Jugador +obtenerTurno() : Turno

Turno

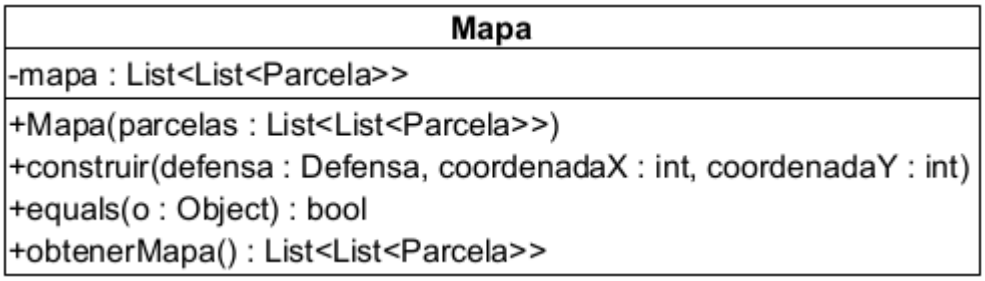


Jugador

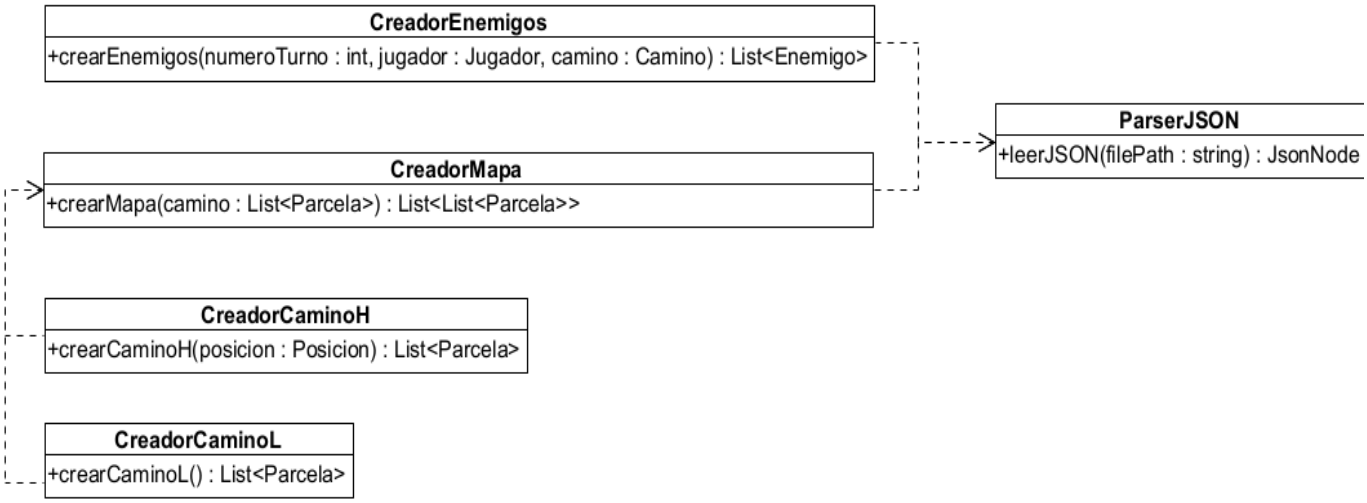


Mapa

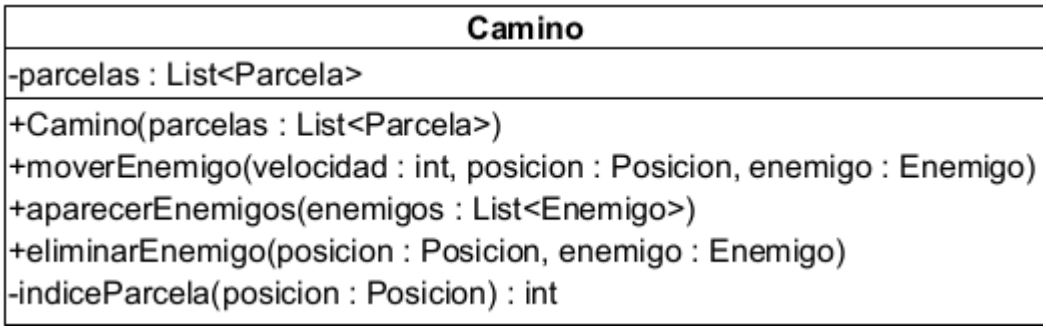
Mapa



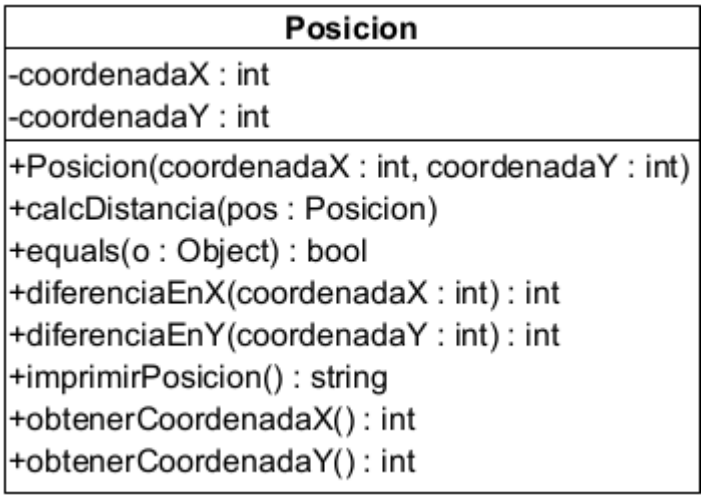
Creadores de parcelas y Parser



Camino

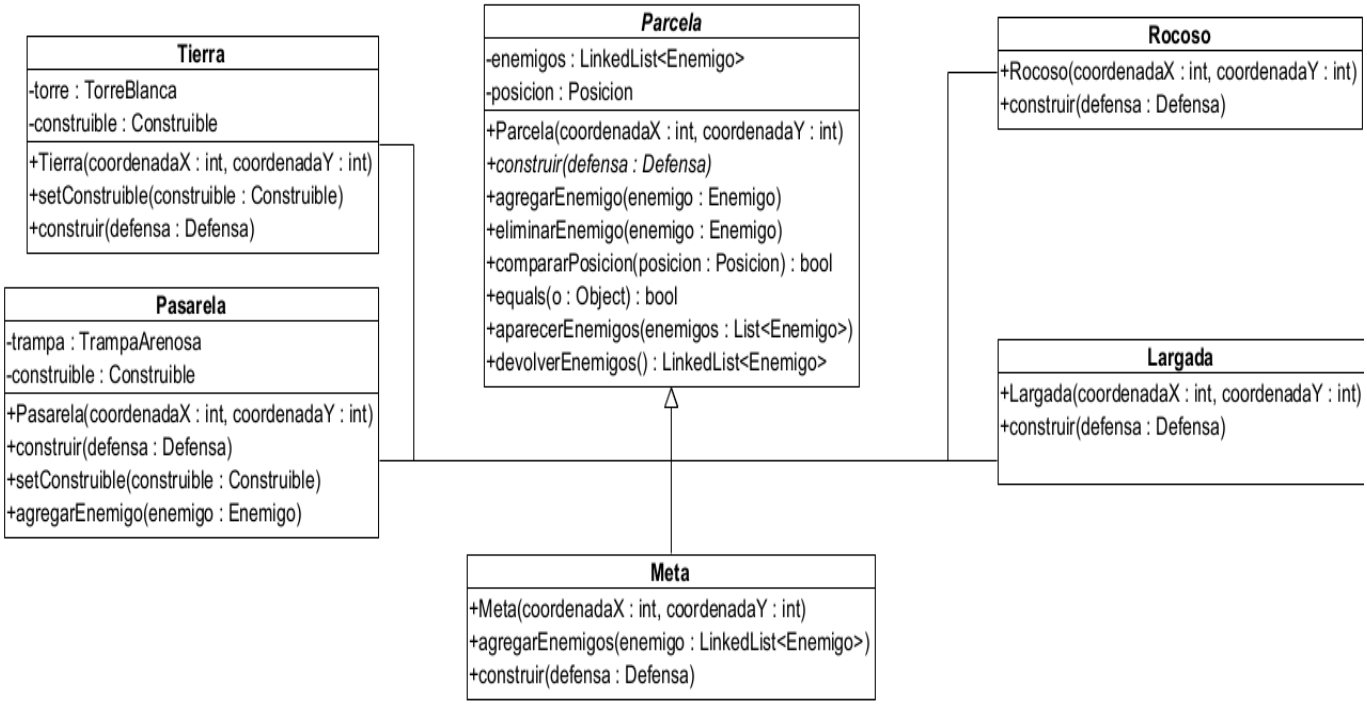


Posicion

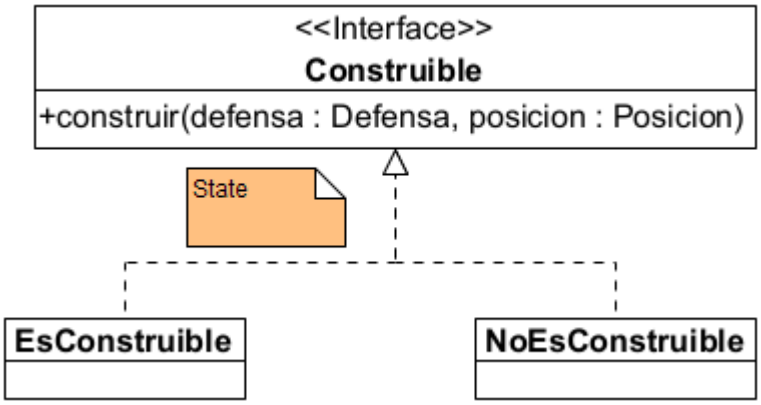


Parcelas

Parcela

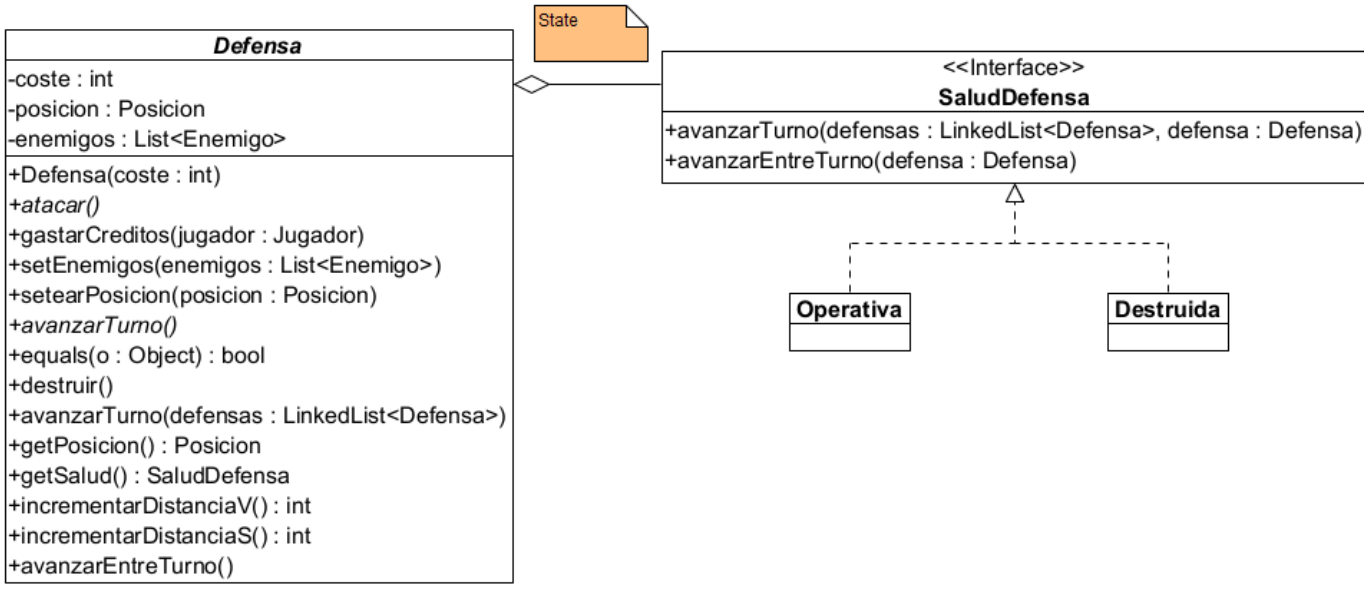


Construible

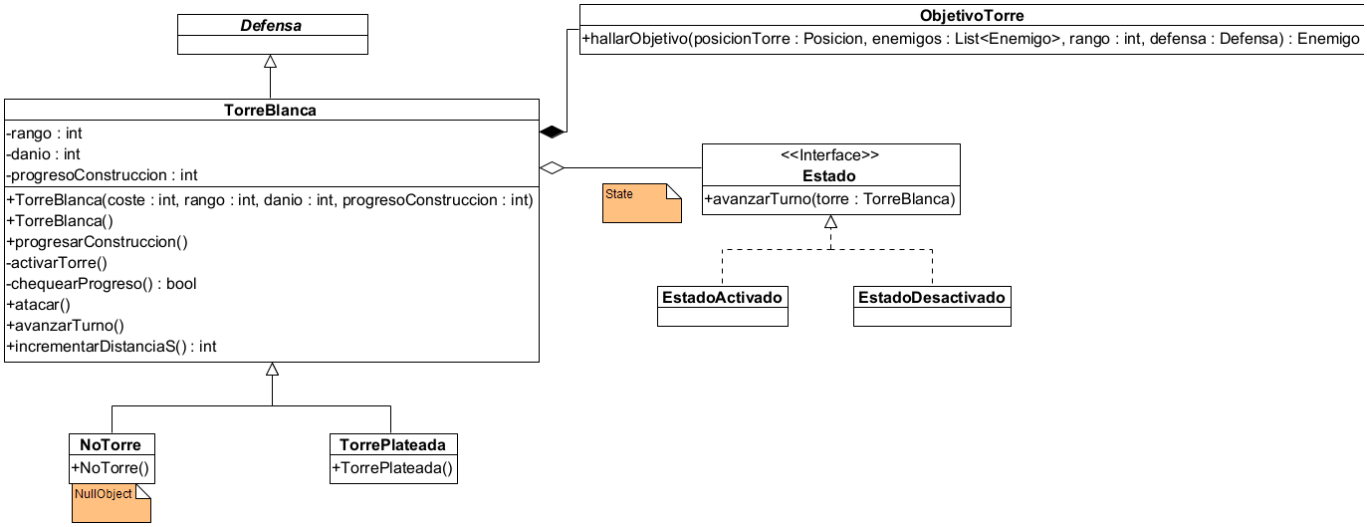


Defensas

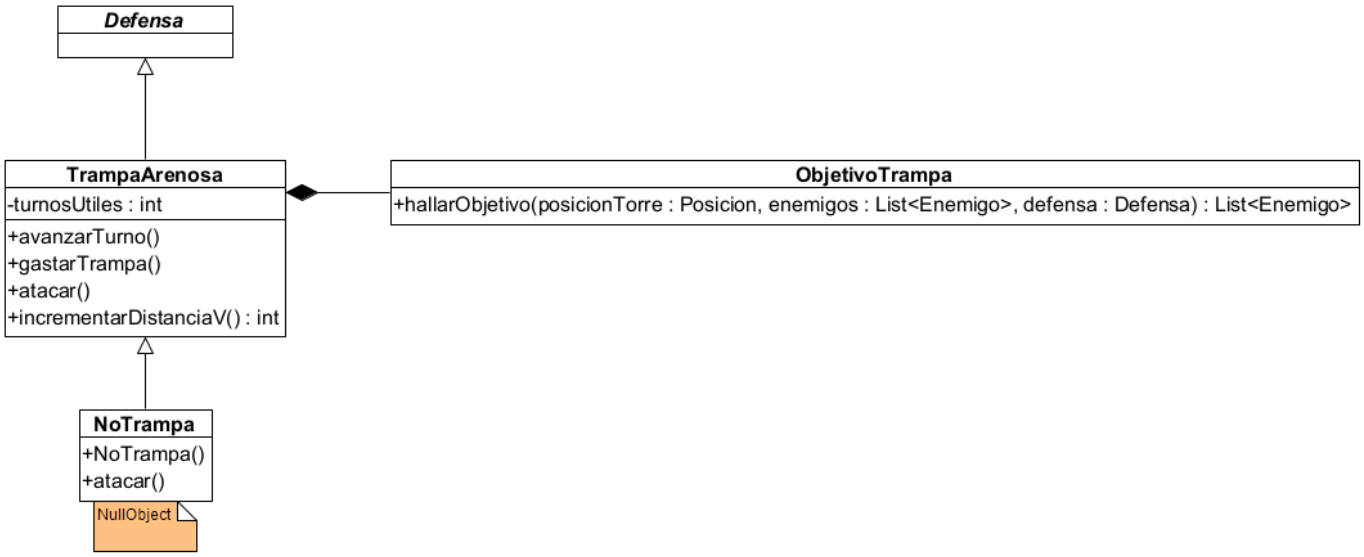
Defensa



Torres

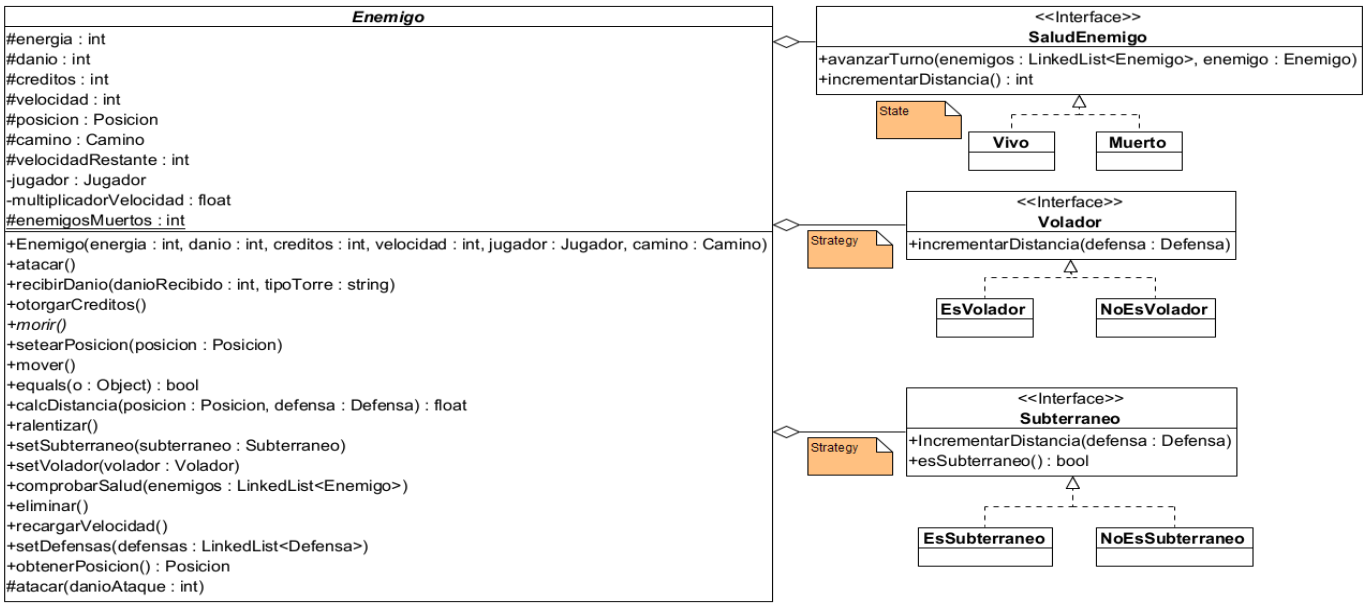


TrampaArenosa

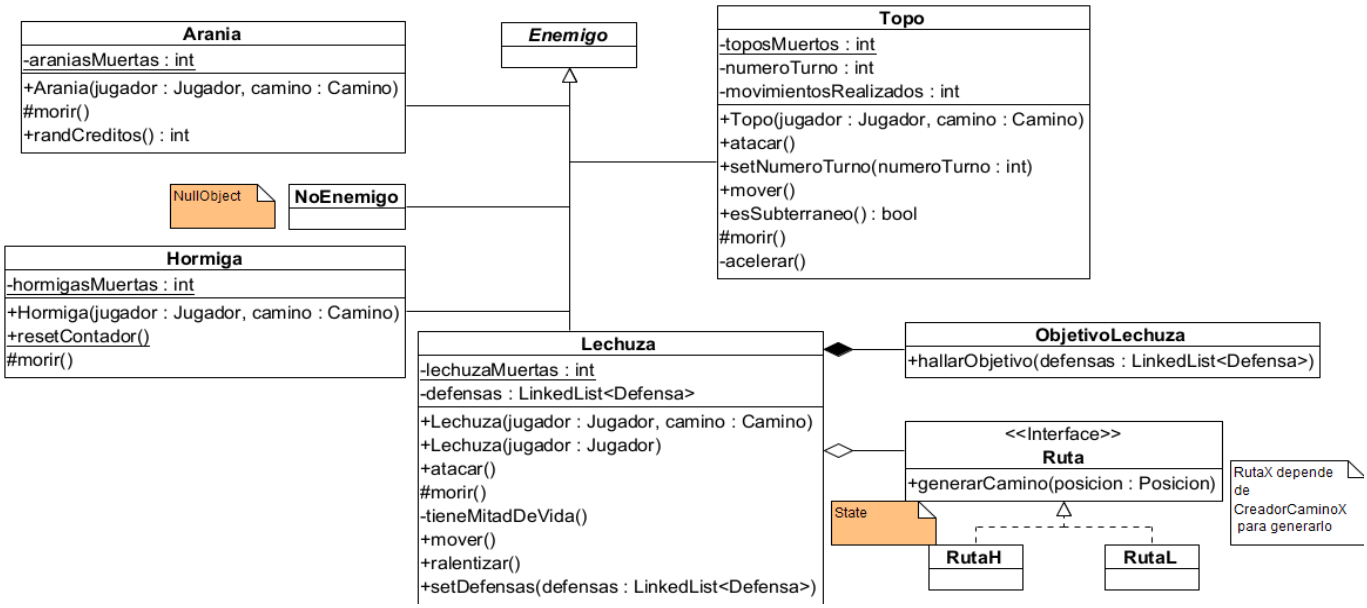


Enemigos

Enemigo

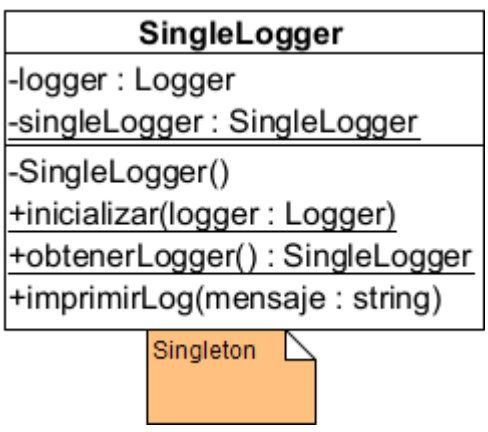


Enemigos



Logger

SingleLogger

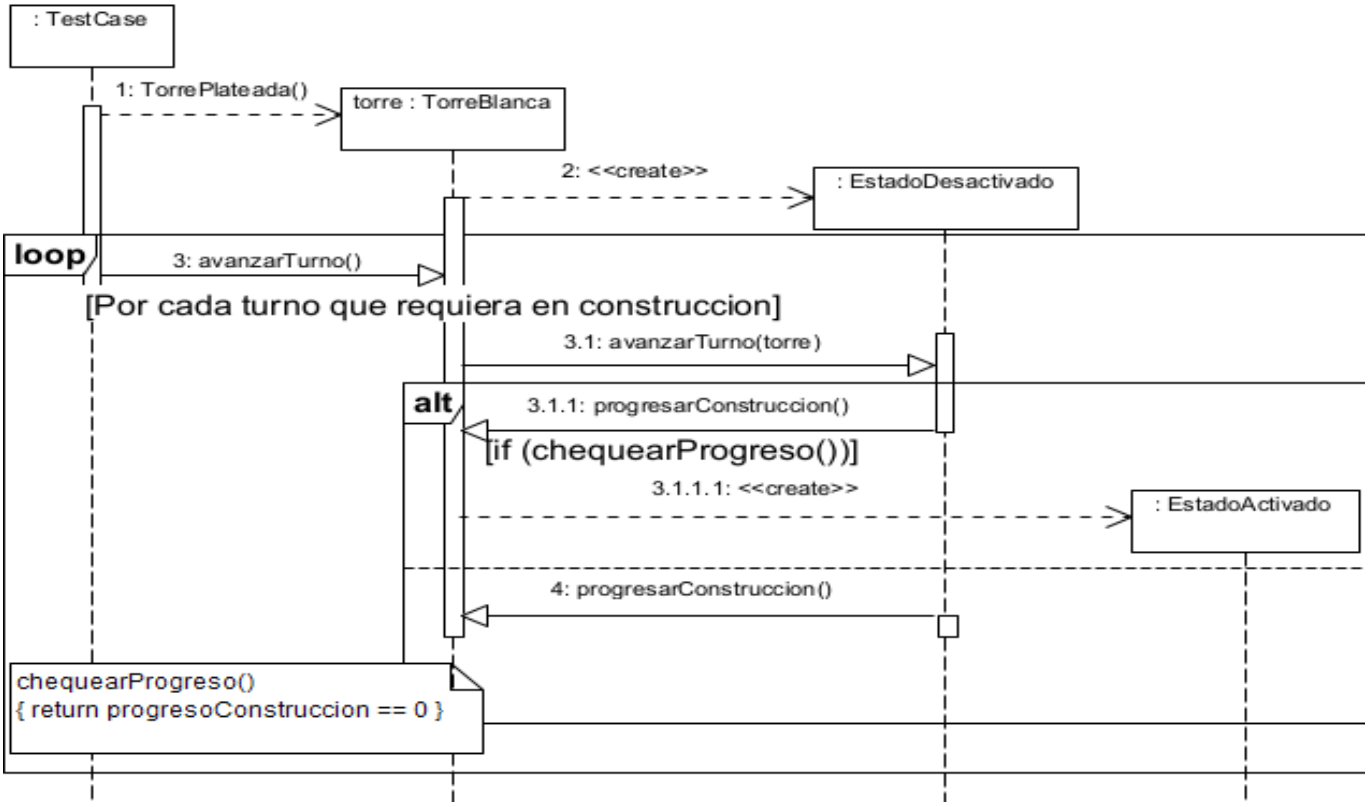


Diagramas de Secuencia

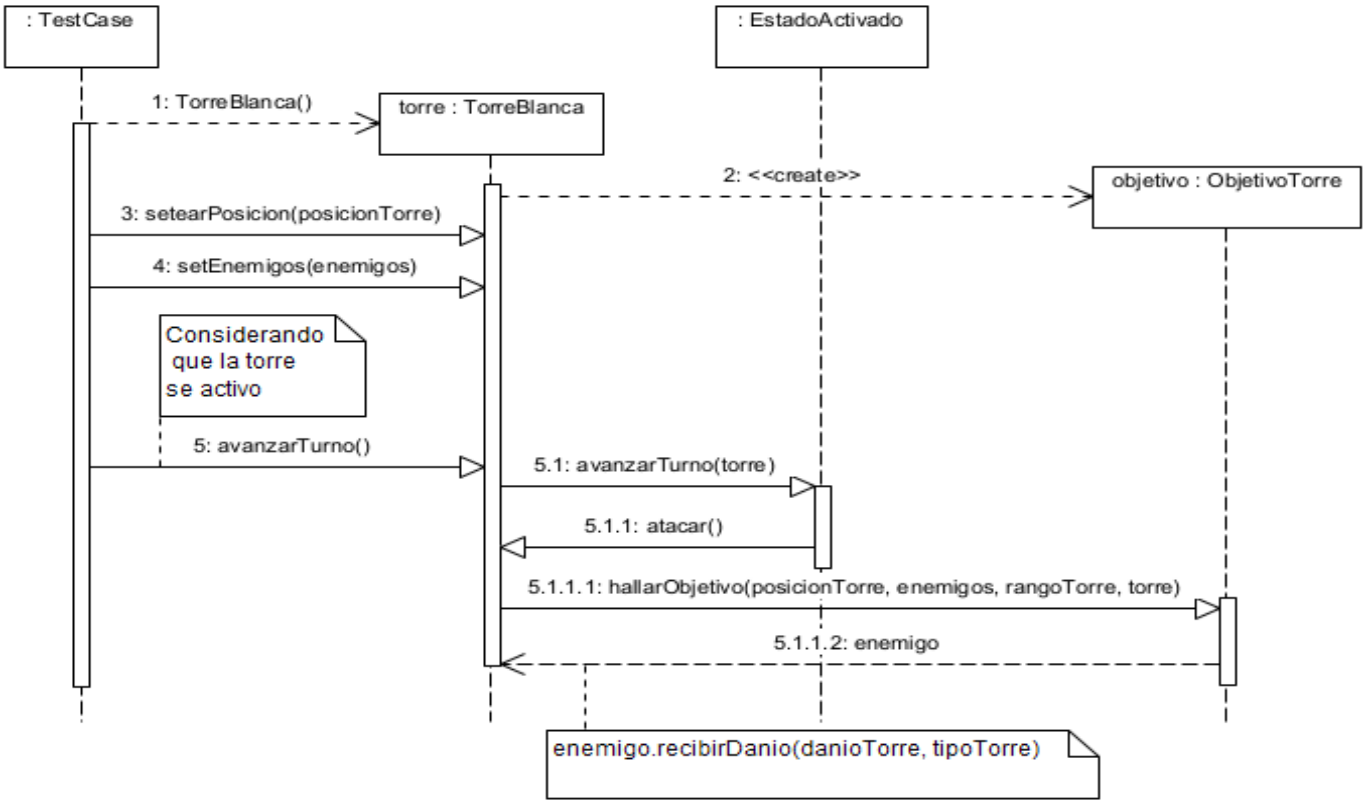
De los casos de uso planteados en el enunciado, rescatamos las siguientes secuencias interesantes:

Torres y trampas

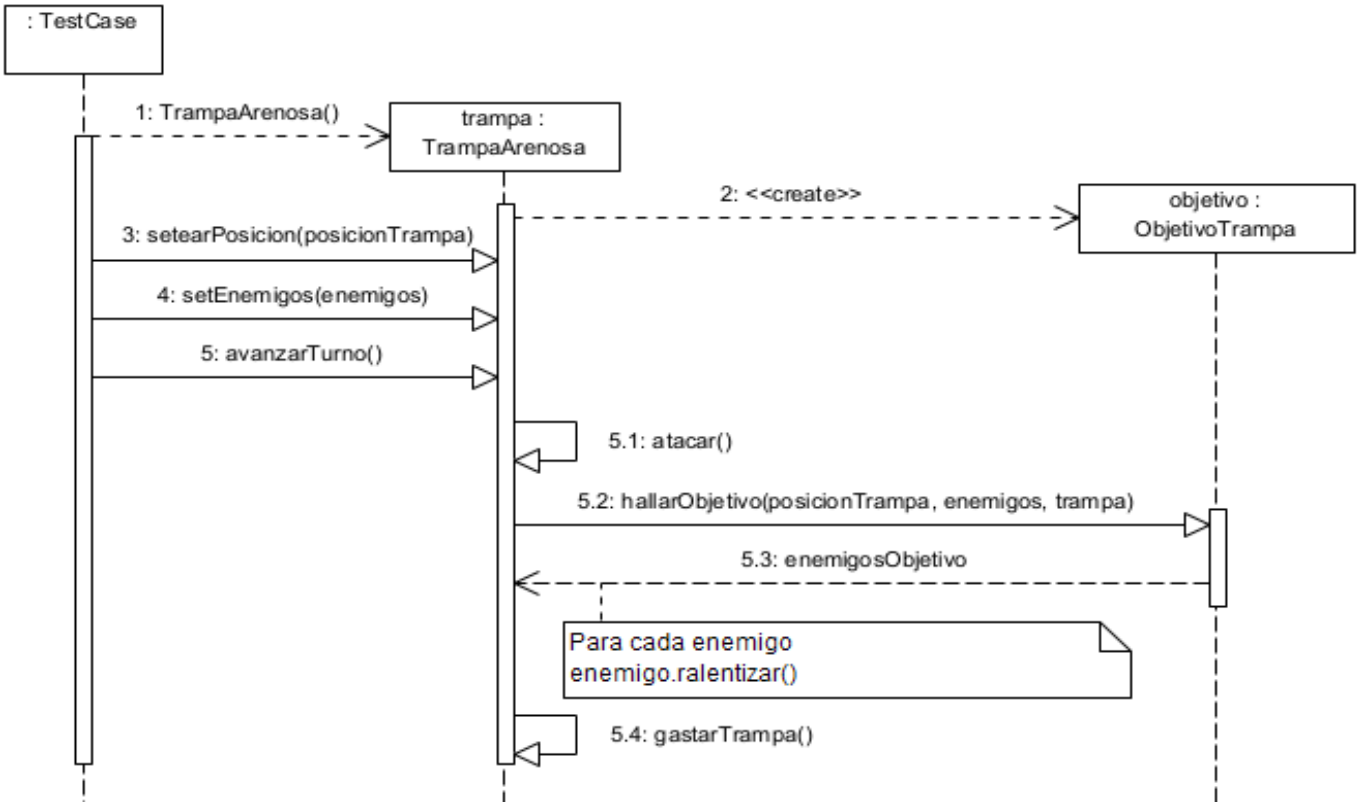
Activacion de una torre



Hallar enemigo torres



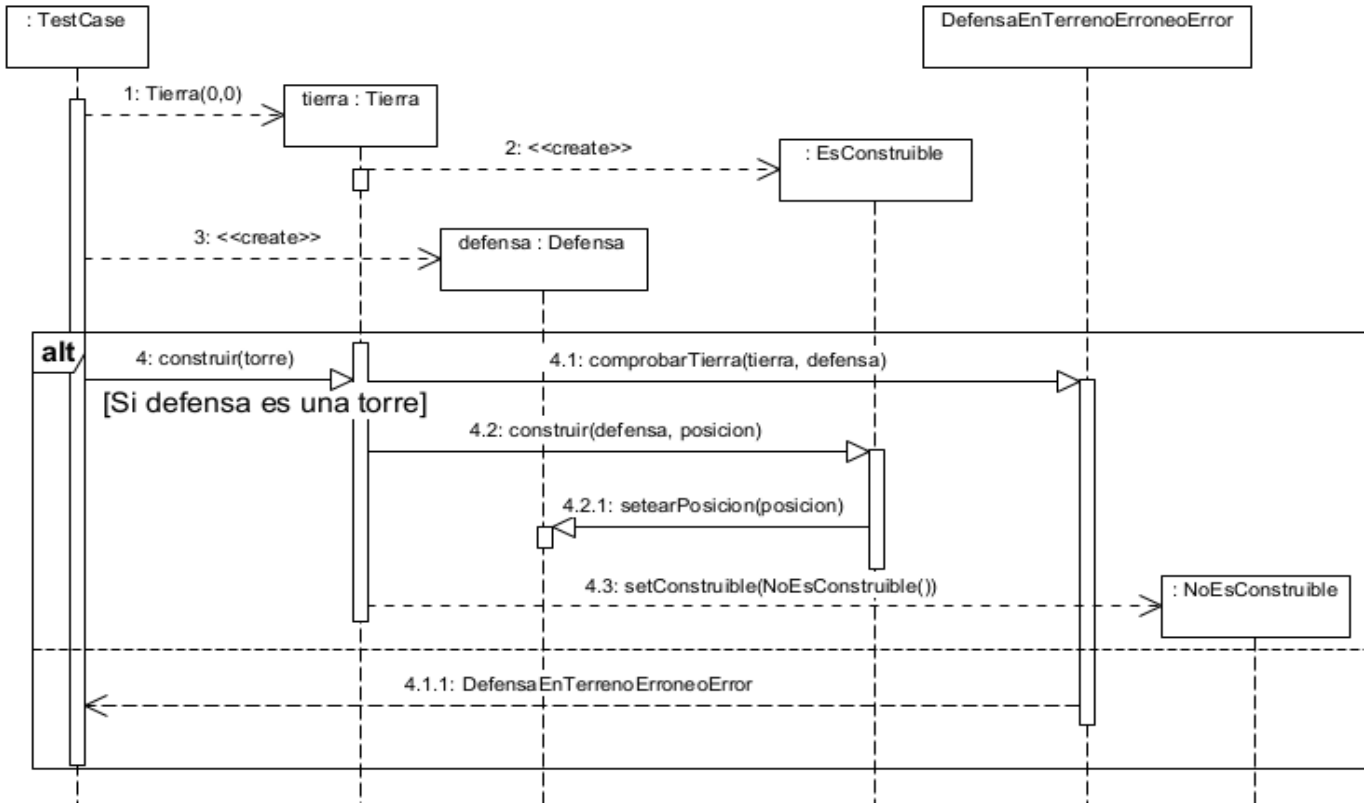
Hallar enemigos trampas



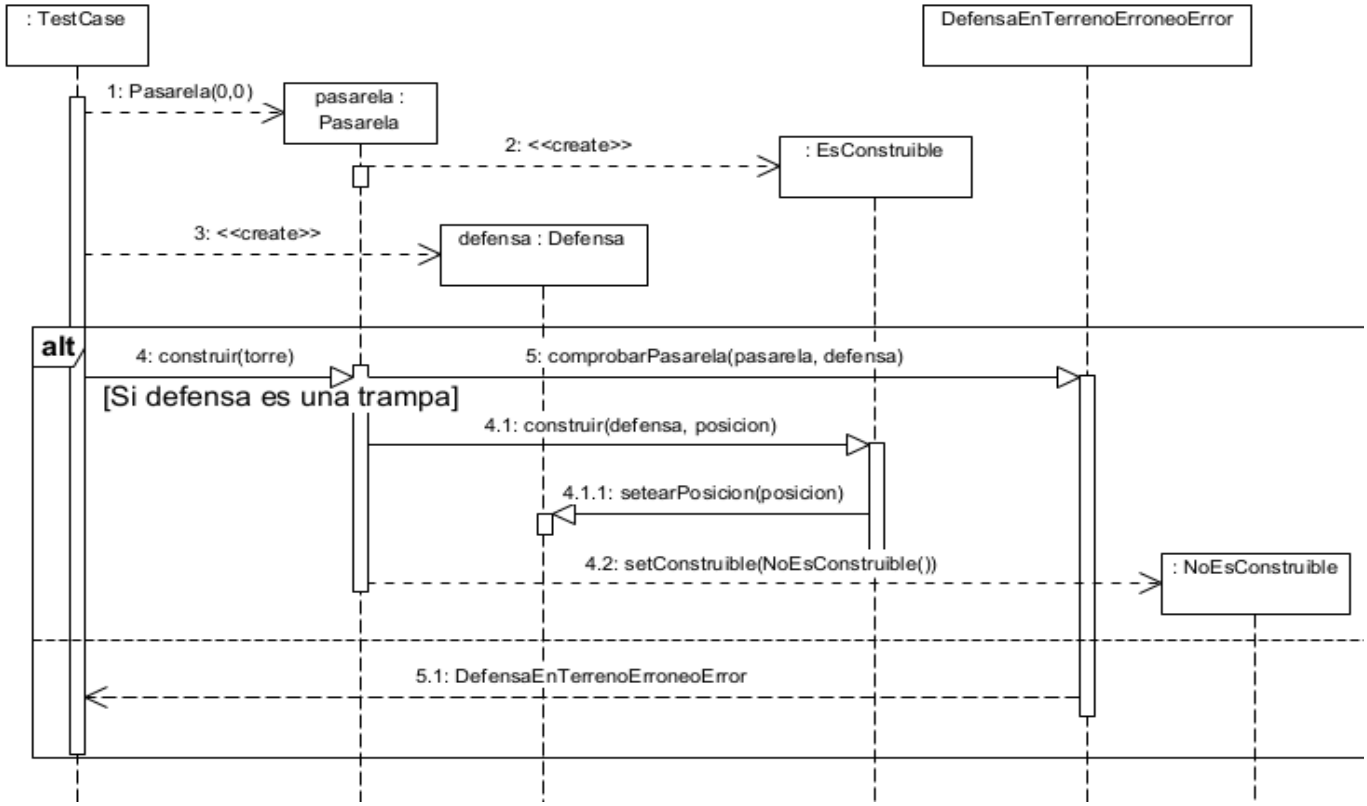
Construccion en las diferentes parcelas

Meta, Largada y Rocoso unicamente arrojan una excepcion al recibir el mensaje construir(). Por lo cual decidimos prescindir de diagramar estos casos.

Construccion en Tierra

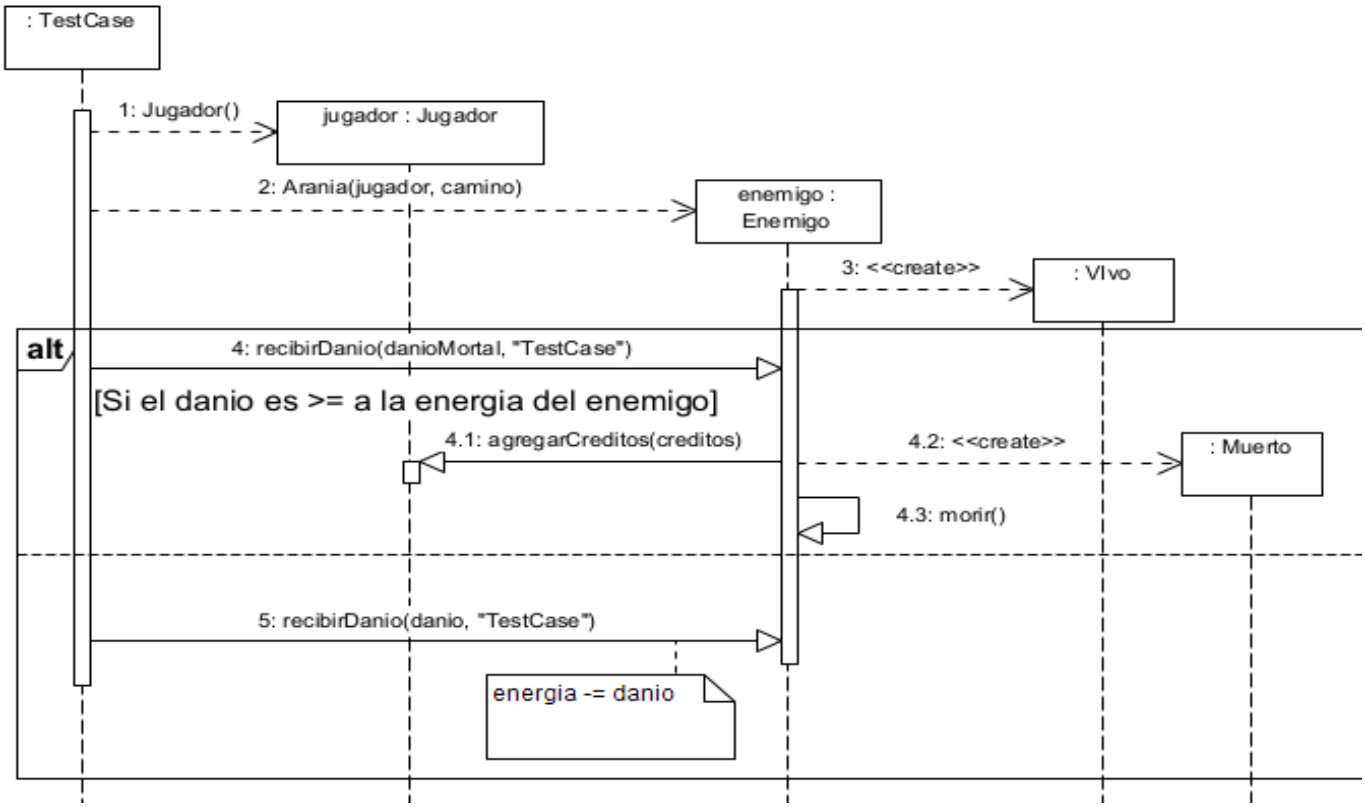


Construccion en Pasarela



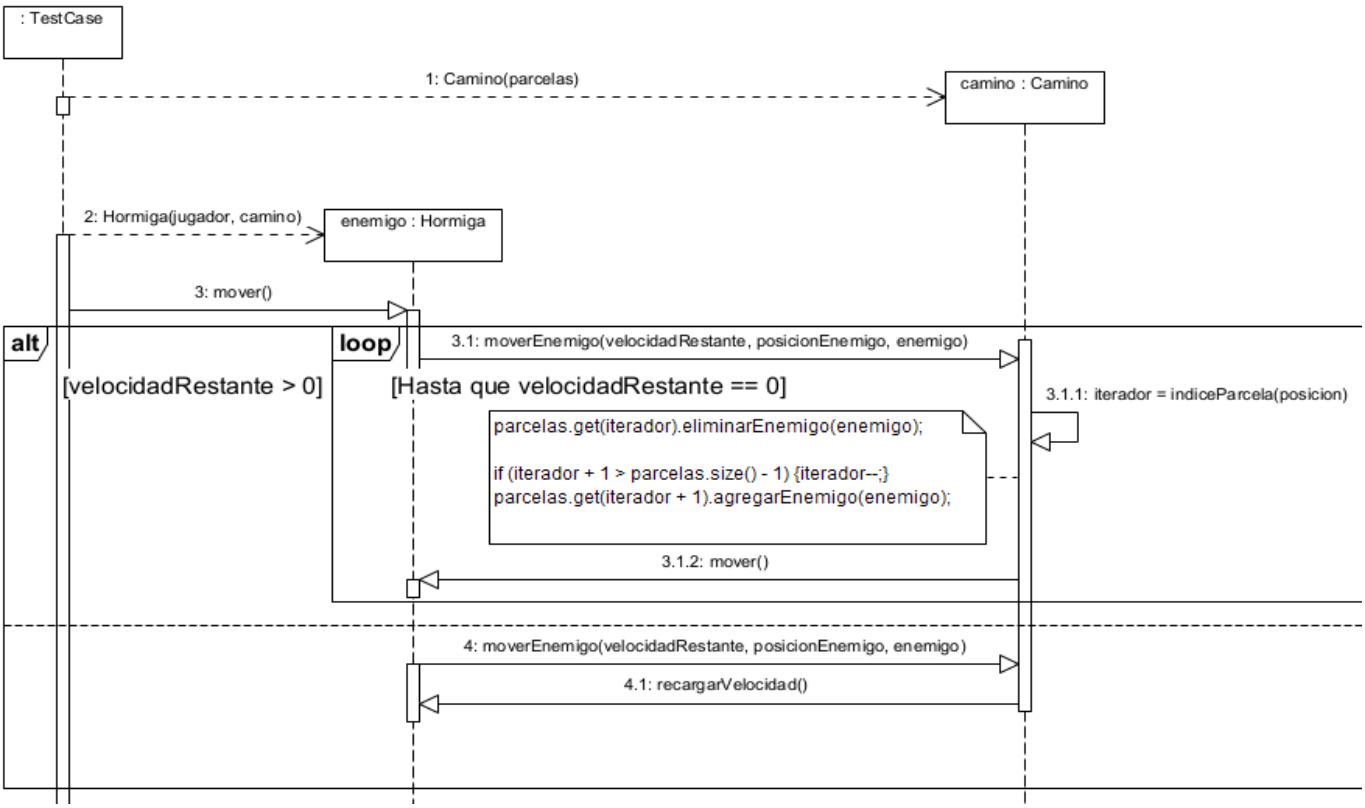
Enemigos

Muerte de un enemigo



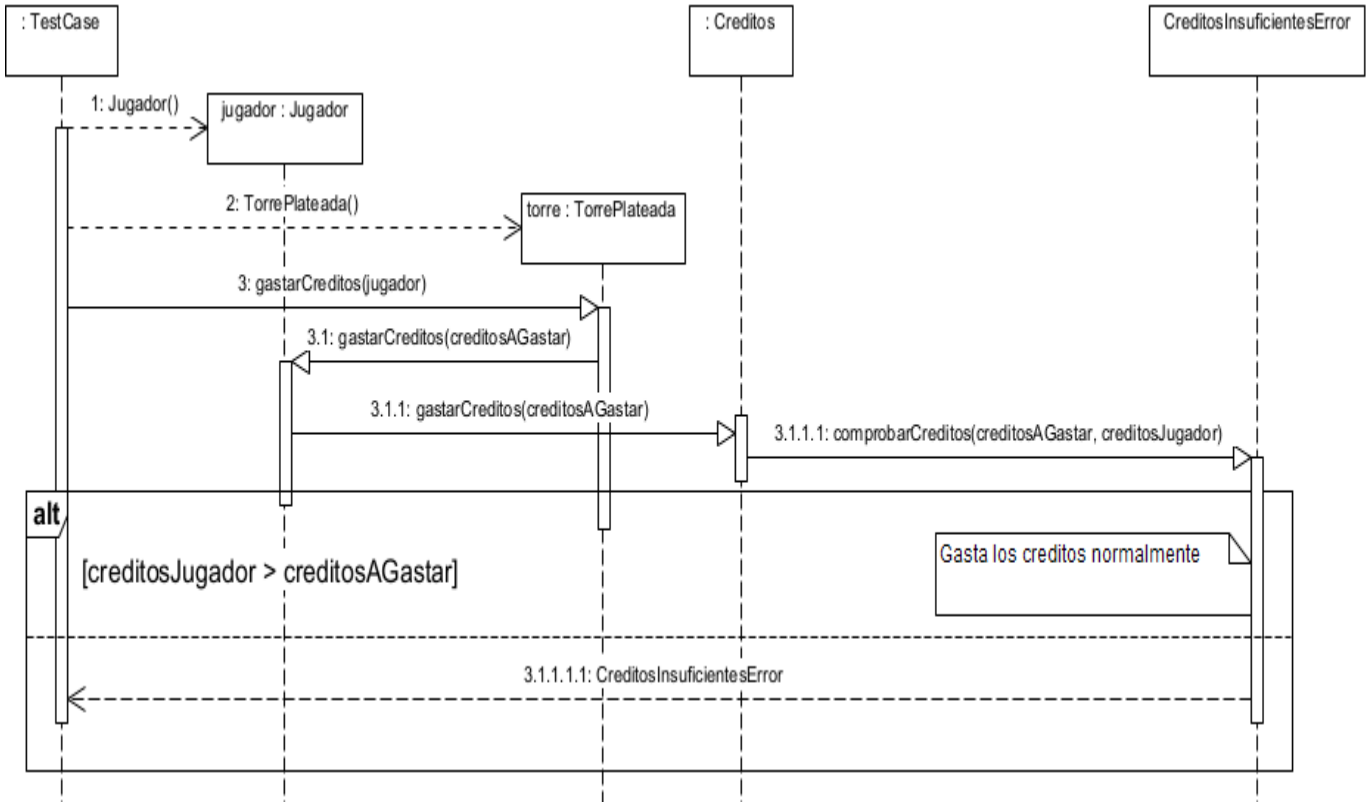
Movimiento de un enemigo

Misma secuencia para todos los tipos de enemigos. En casos como la lechuza que sobreescribe mover() lo hace para utilizar un camino alterno al normal. O en casos como el Topo es para acelerar si debe hacerlo.

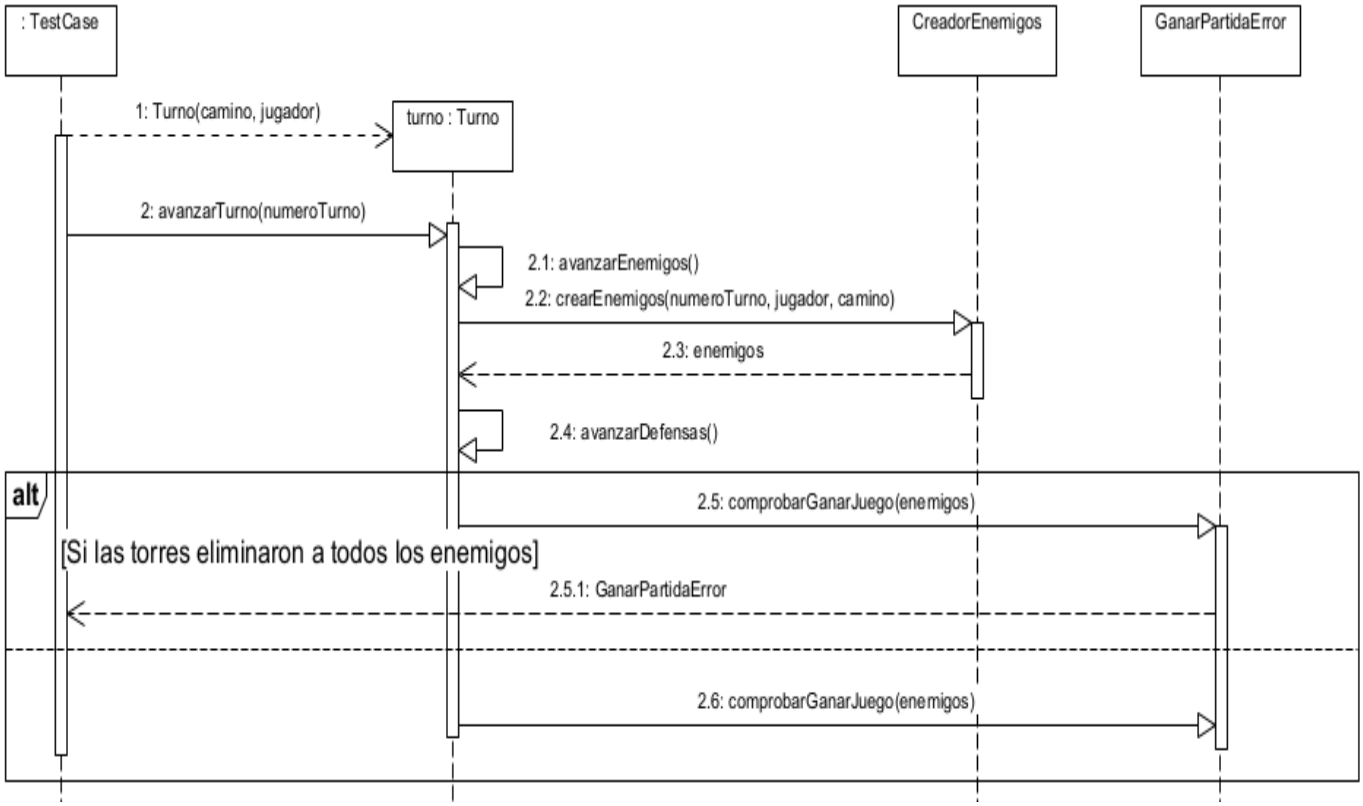


Analisis de excepciones

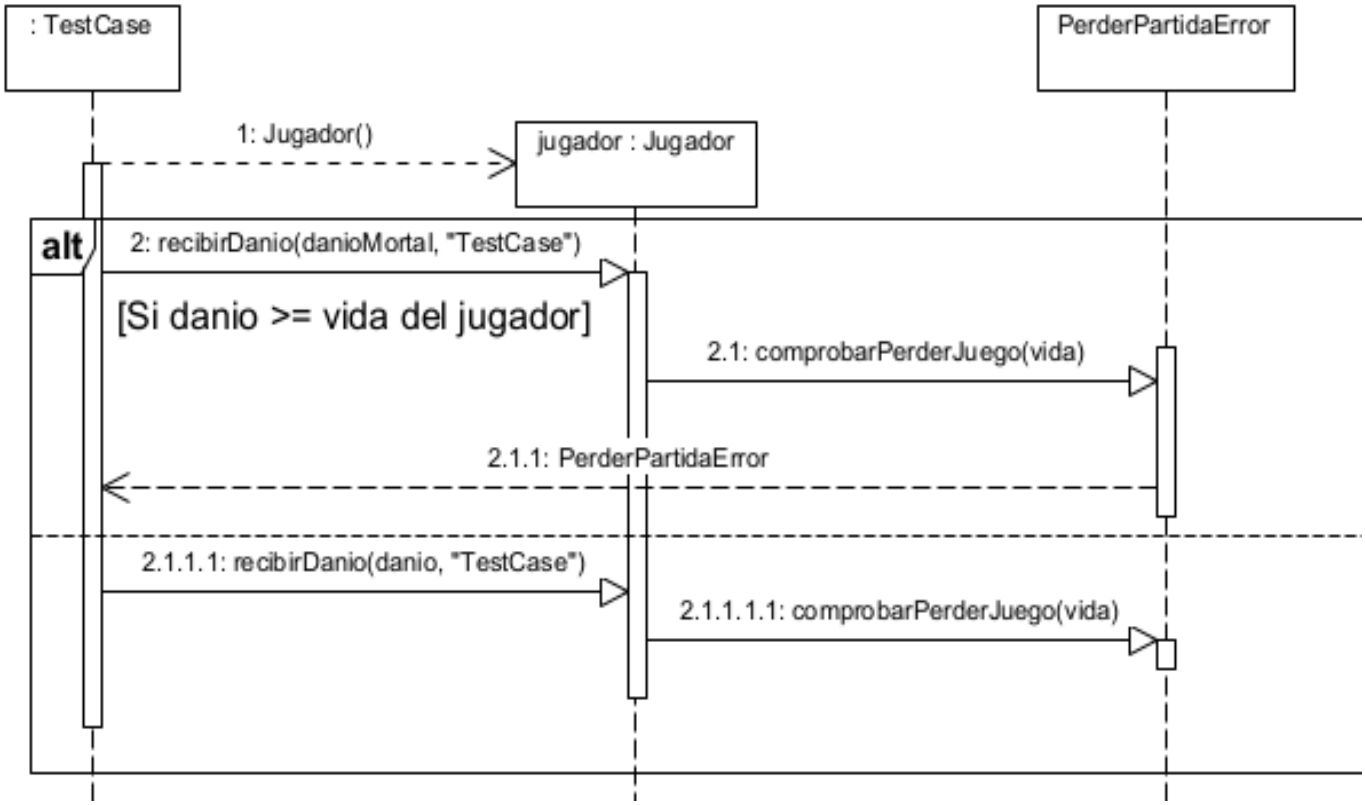
Analisis CreditosInsuficientesError



Analisis de ganar una partida



Analisis de perder una partida



Diagramas de Estados

Diagramas de Paquetes