

# TP2-GPSCHALLENGE

[7507/9502] Algoritmos y Programación III Curso 2  
Primer cuatrimestre de 2022



Maria Dolores Pavón - 108221 - Dolo-pavon01



Julieta Perez Goldstein - 107997 - Perezgjulietta



Camila Ayala - 107047 - camilaayala01



Melina Aylen Loscalzo Acosta - 106571 - Melulatana



Kevin Vasquez - 97548 - kaibakev1984



# WUHOZH

## TP2- GpsChallenge



GRUPO:10

- 1 **Introducción**

---
- 2 **Supuestos**

---
- 3 **Diagrama De Clases**

---
- 4 **Diagrama De Secuencia**

---
- 5 **Diagrama de Paquetes**

---
- 6 **Diagrama De Estado**

---
- 7 **Detalle de Implementación**

---
- 8 **Excepciones**

---
- 9 **Demostración Del  
Funcionamiento Del Juego**

---

# 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. La aplicación consiste de un juego de estrategia por turnos. El escenario es una ciudad y el objetivo, guiar un vehículo a la meta en la menor cantidad de movimientos posibles.

# SUPUESTOS

**Debido a que ciertas especificaciones no fueron provistas por la cátedra y en ciertas ocasiones el criterio de cómo abordar alguna situación quedaba a cargo del alumno, se tuvieron en cuenta los siguientes supuestos:**



**Puede haber un tipo de obstáculo y un tipo de sorpresa en la misma posición.**



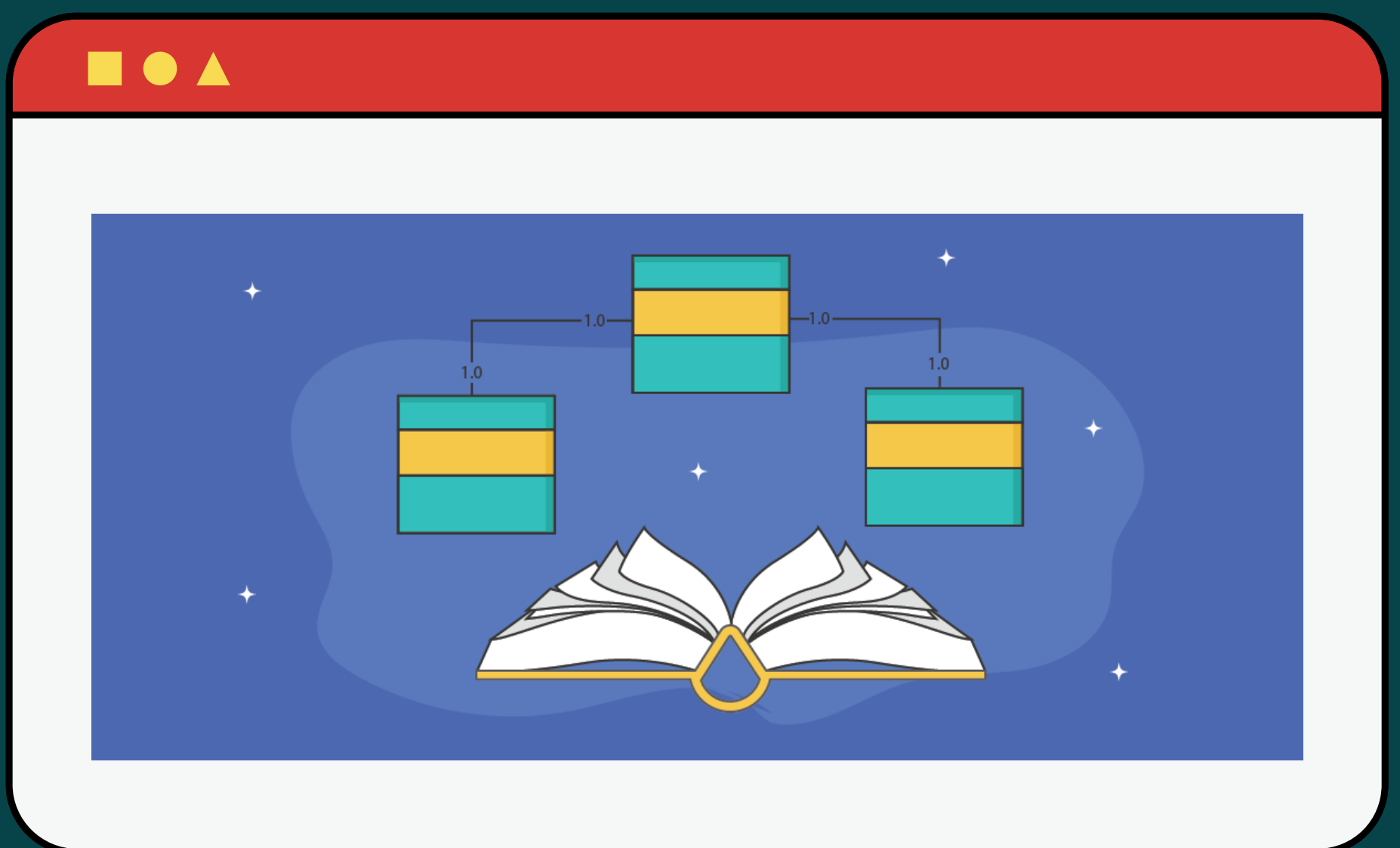
**El vehiculo puede pasar más de una vez por por cada sorpresa.**



**Cuando ocurre el cambio de vehiculo Auto a Auto4x4 el contador de pozos se encuentra en cero.**

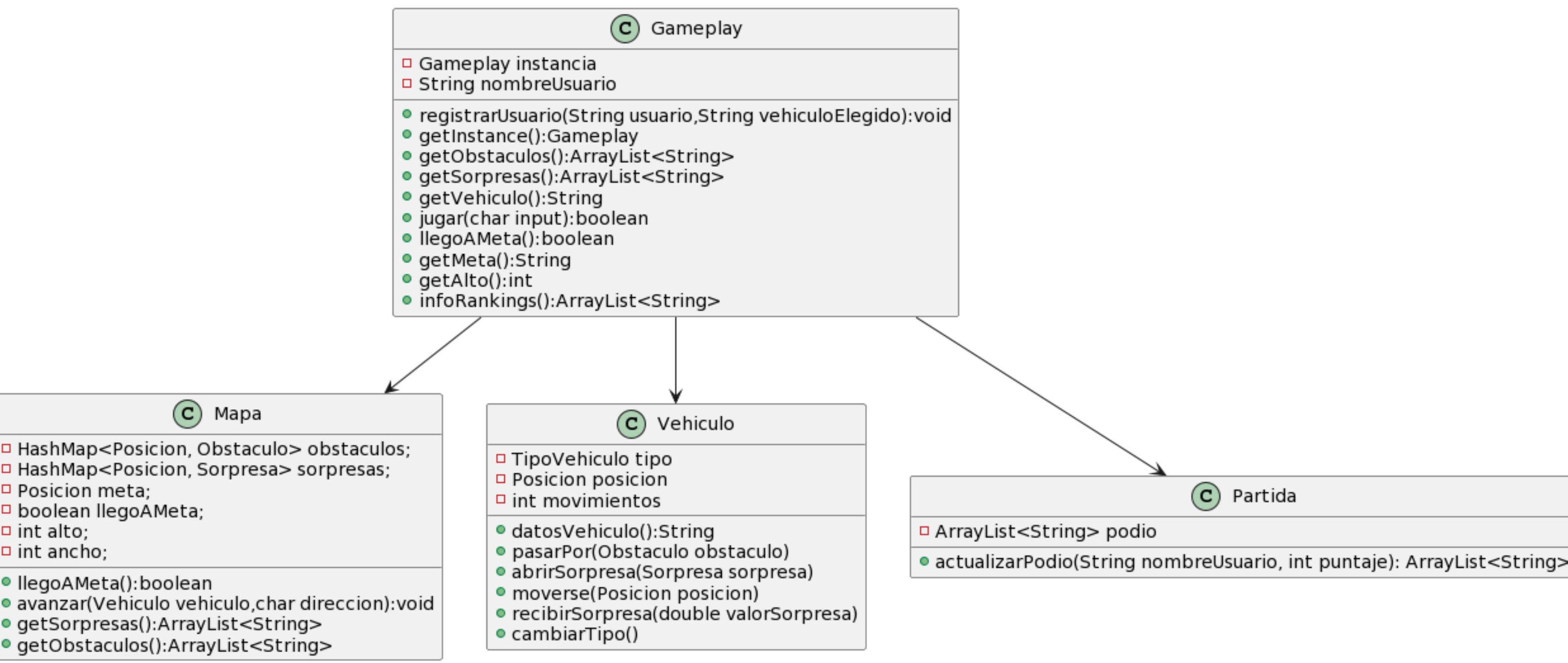
# DIAGRAMAS DE CLASES

**La clase es una construcción de casi todos los lenguajes orientados a objetos. Esto hace que el diagrama de clases sea el diagrama estructural más importante a la hora de modelar diseño detallado y programación.**

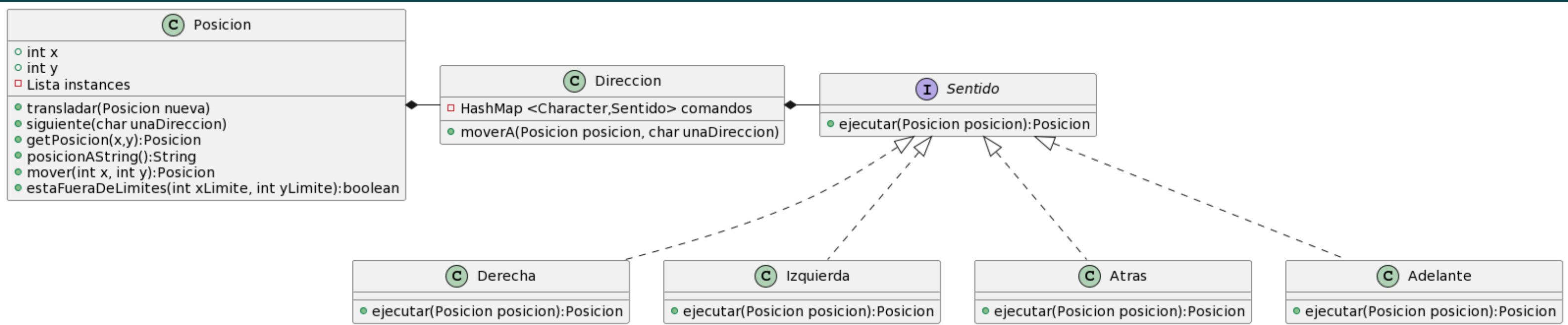


**En esta sección se encuentran todas las entidades que fueron utilizadas y las relaciones estáticas entre ellas en los siguientes diagramas:**

# Diagrama de Clases de Gameplay

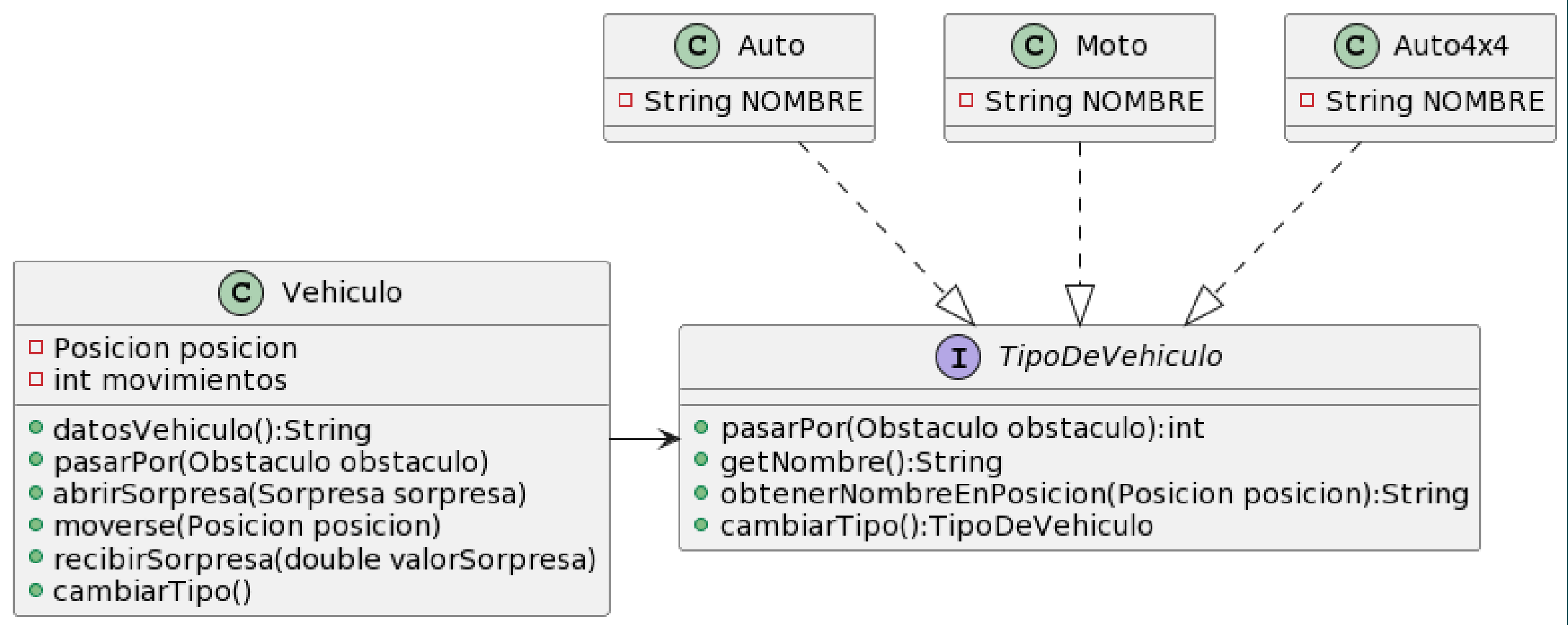


# Diagrama de Clases de Movimiento

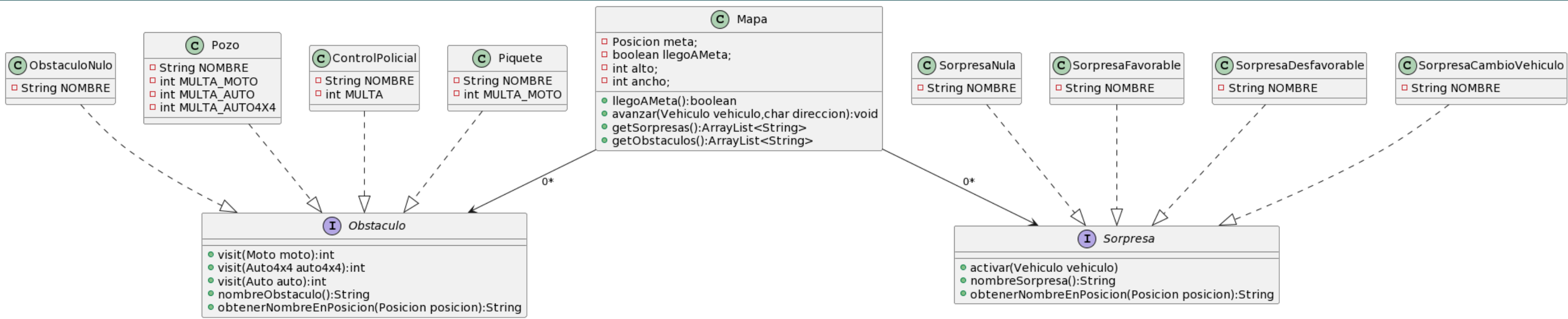




# Diagrama de Clases de Vehiculo

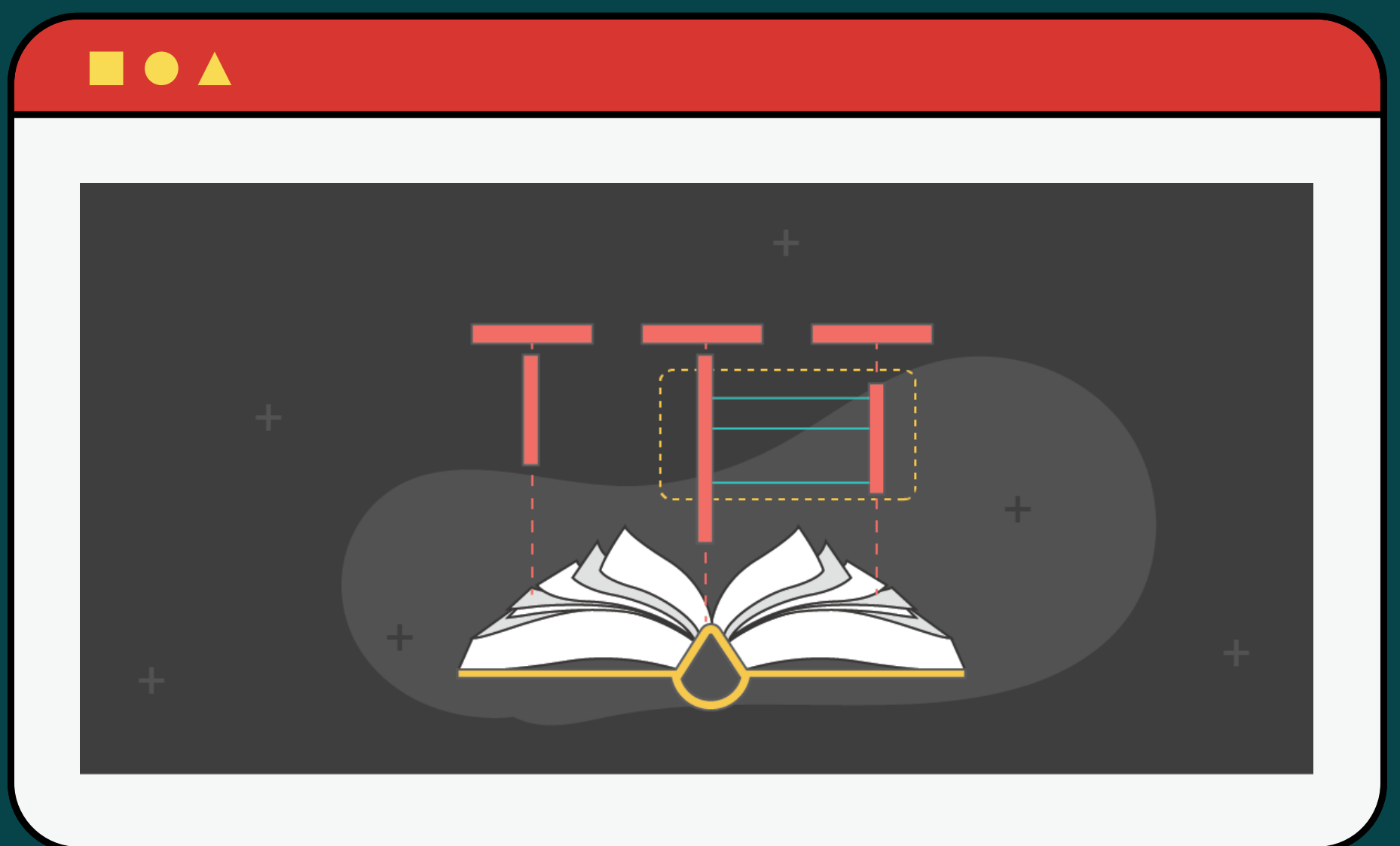


# Diagrama de Clases de Mapa



# DIAGRAMA DE SECUENCIA

**Dado que los diagramas de secuencia son equivalentes en semántica a los de comunicación. Lo cierto es que, por la forma en que se dibujan, los diagramas de secuencia son más aptos para representar el paso del tiempo y analizar temporalmente un escenario.**

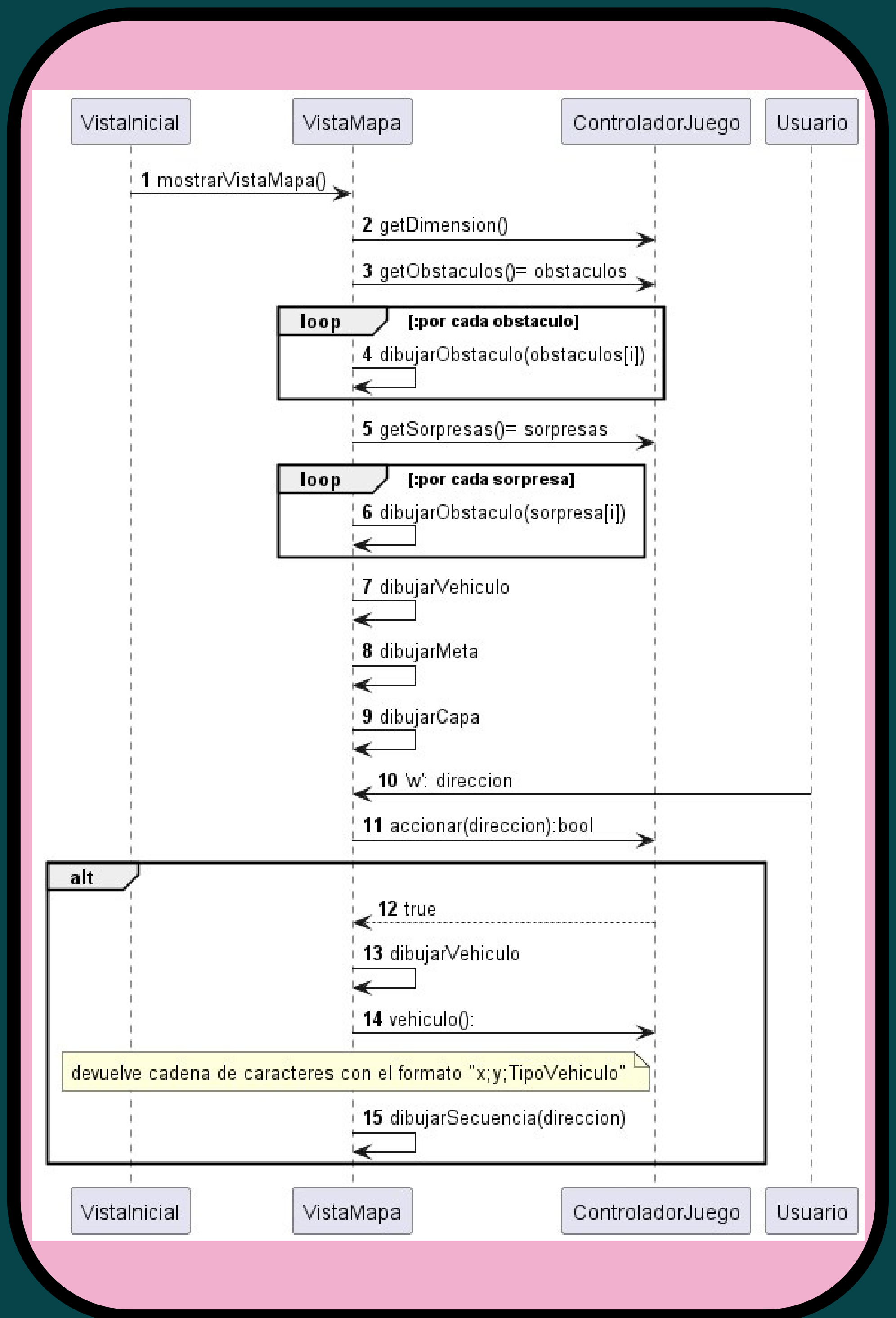


**En los siguientes diagramas, las entidades subrayadas representan instancias de esas mismas clases y las no subrayadas representan abstracciones (interfaces).**

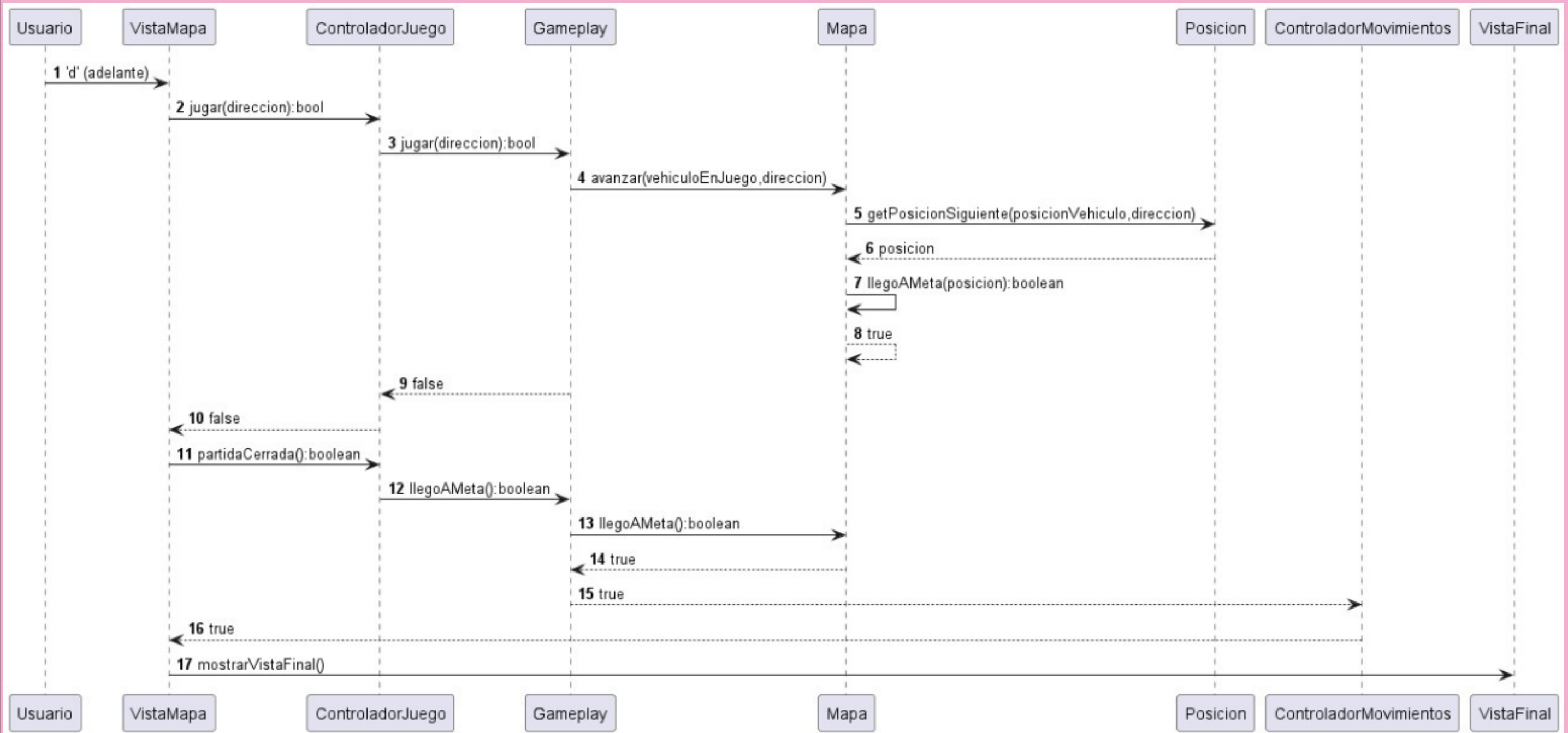


# Diagrama de Secuencia

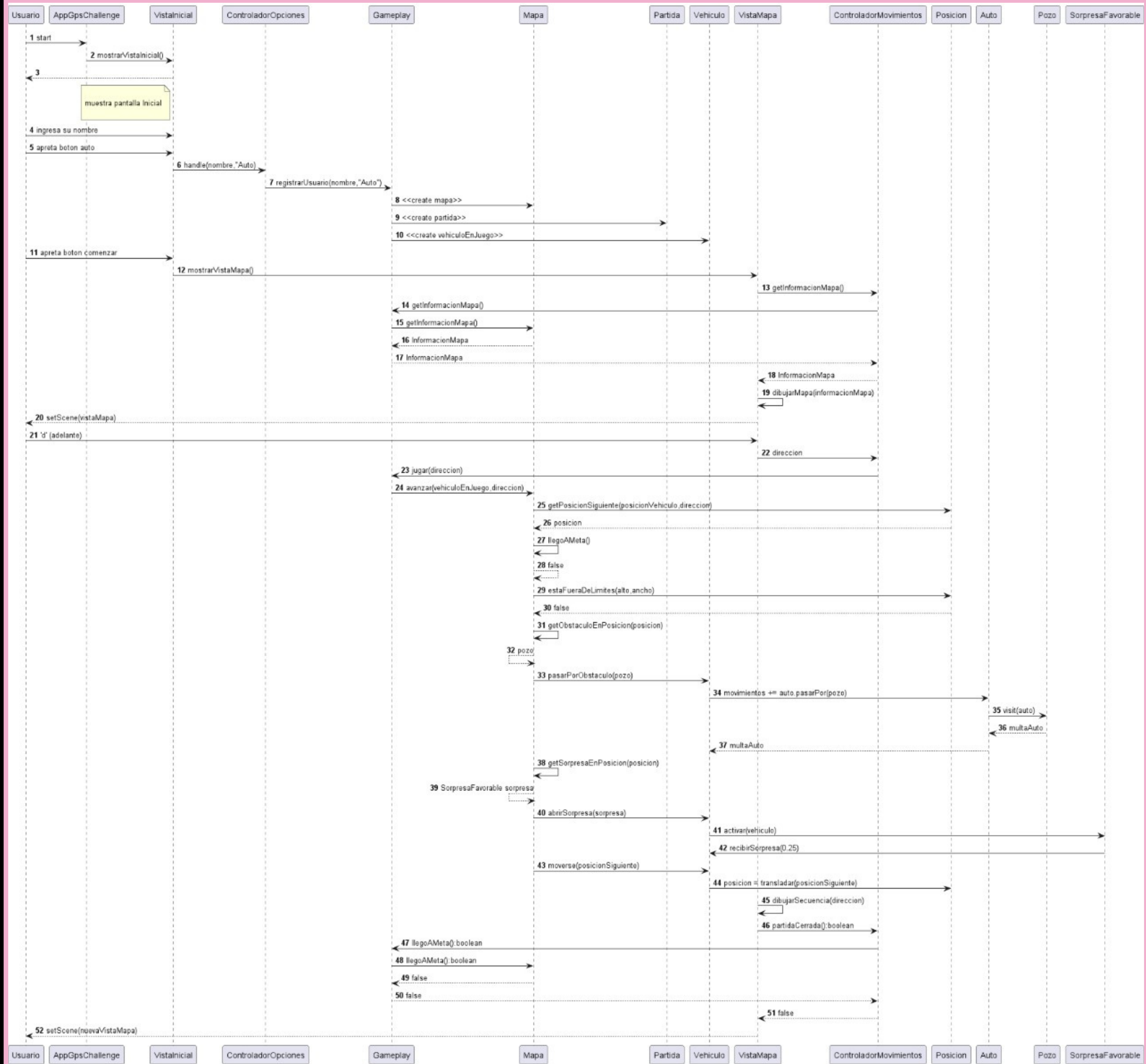
## Secuencia Dibujar Mapa



# Secuencia Finaliza Juego

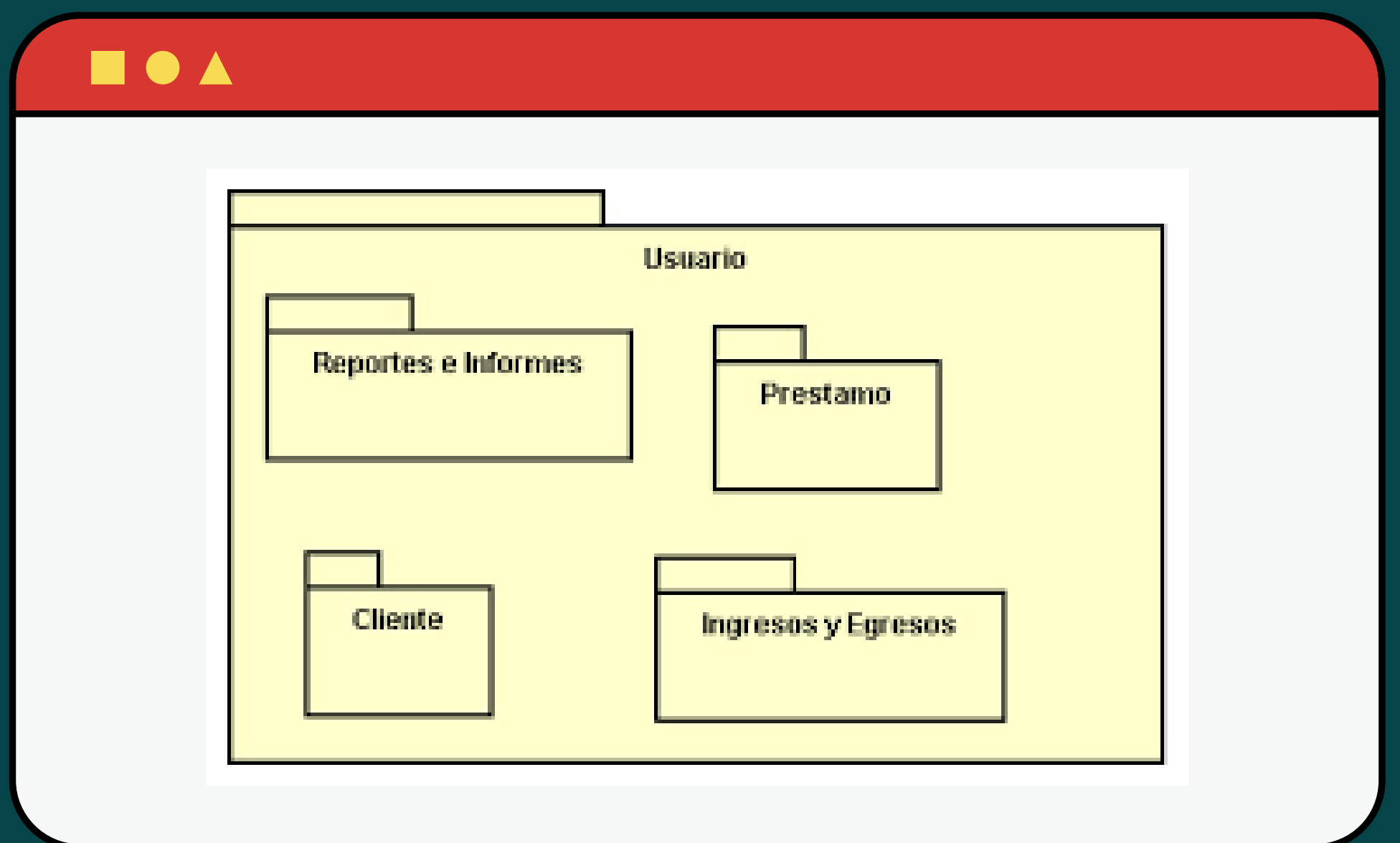


# Secuencia Partida Juego



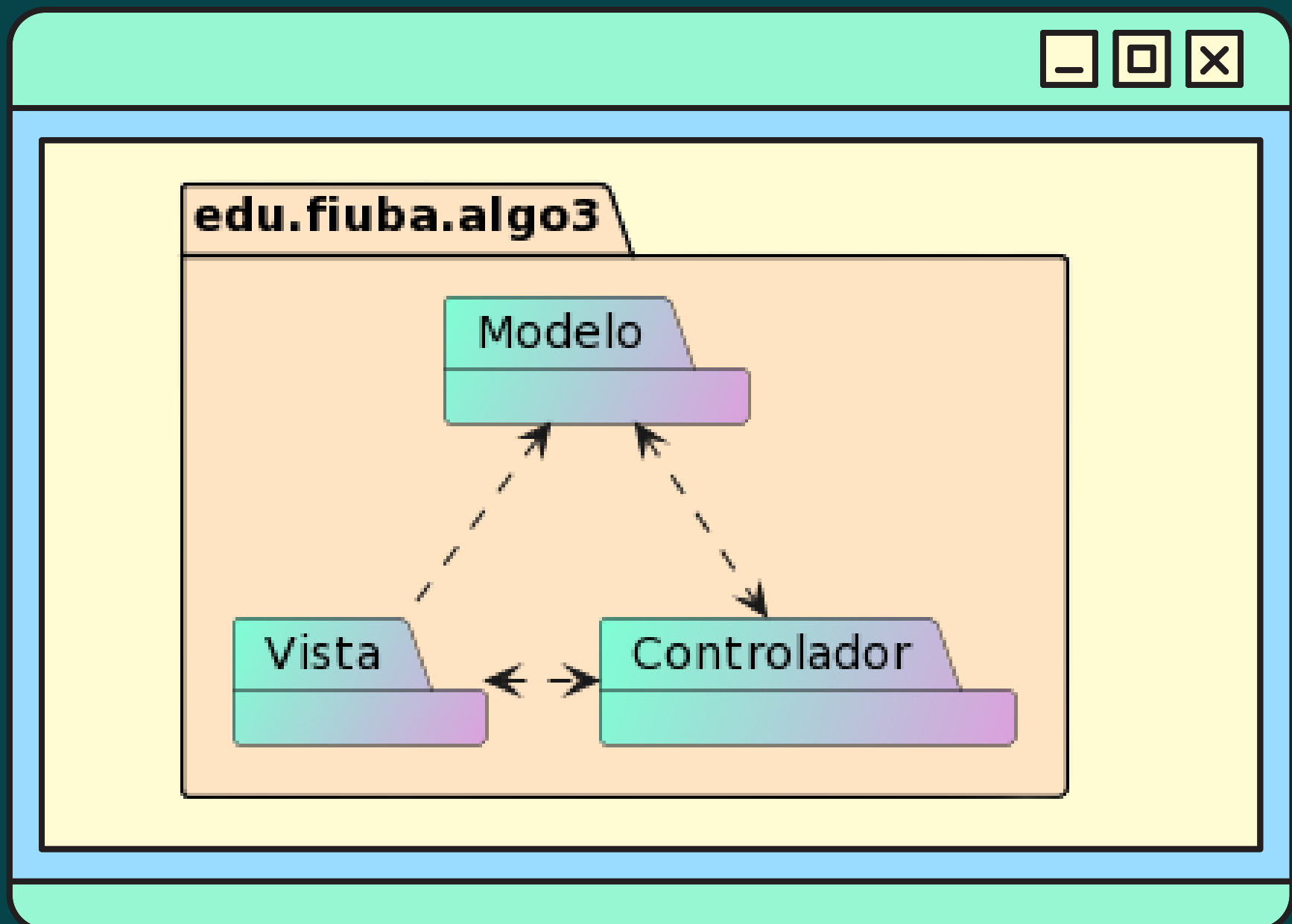
# TIAGRAFAS DE PAQUETES

**El diagrama de paquetes de UML es una herramienta que sirve para agrupar elementos estáticos y es, por definición, un elemento estructural. Cuando decimos que sirve para agrupar elementos estáticos estamos englobando varios tipos de diagramas.**

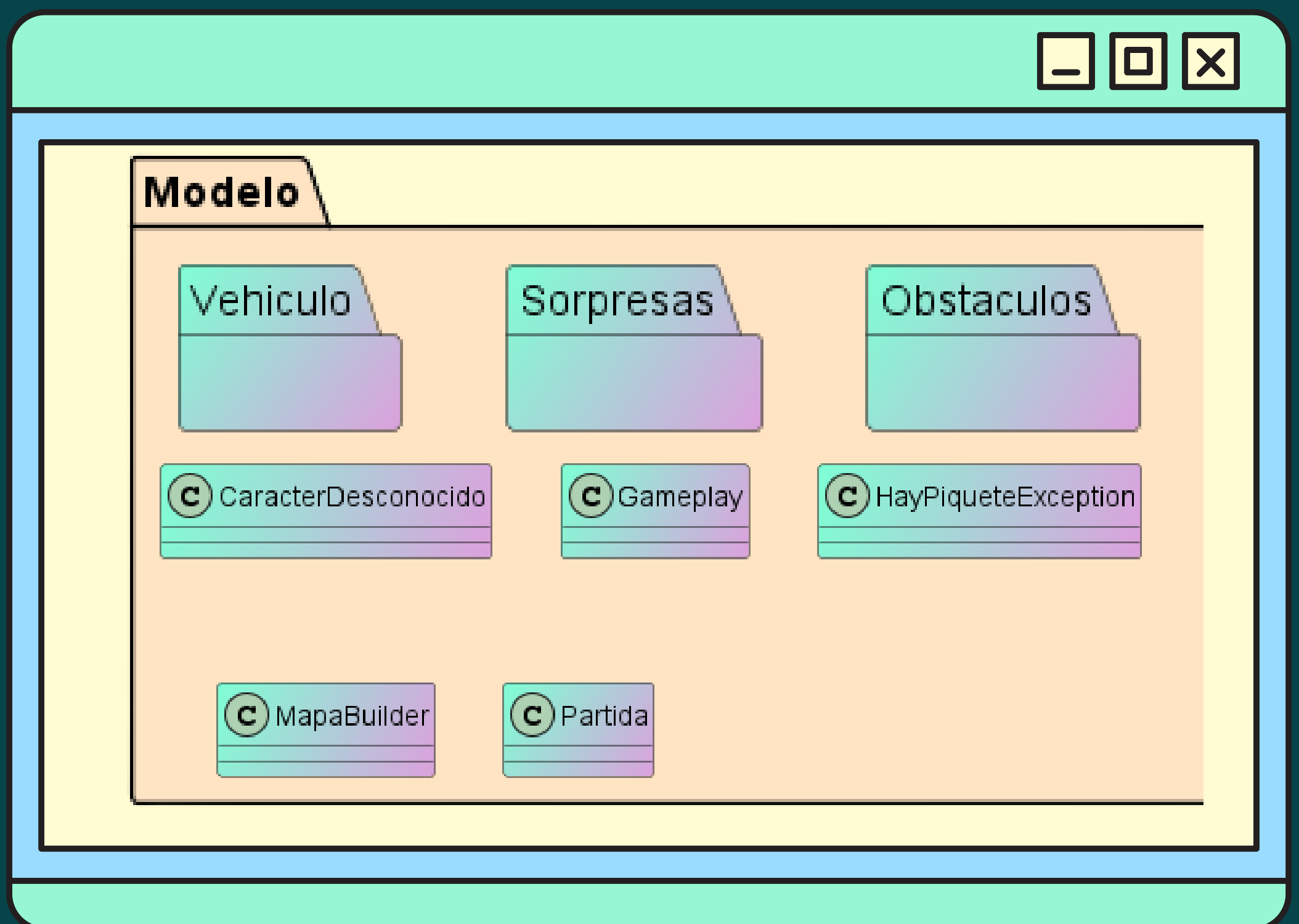


**En este apartado se muestra la organización del proyecto.**

# ITAGRAFAMES PAQUETES

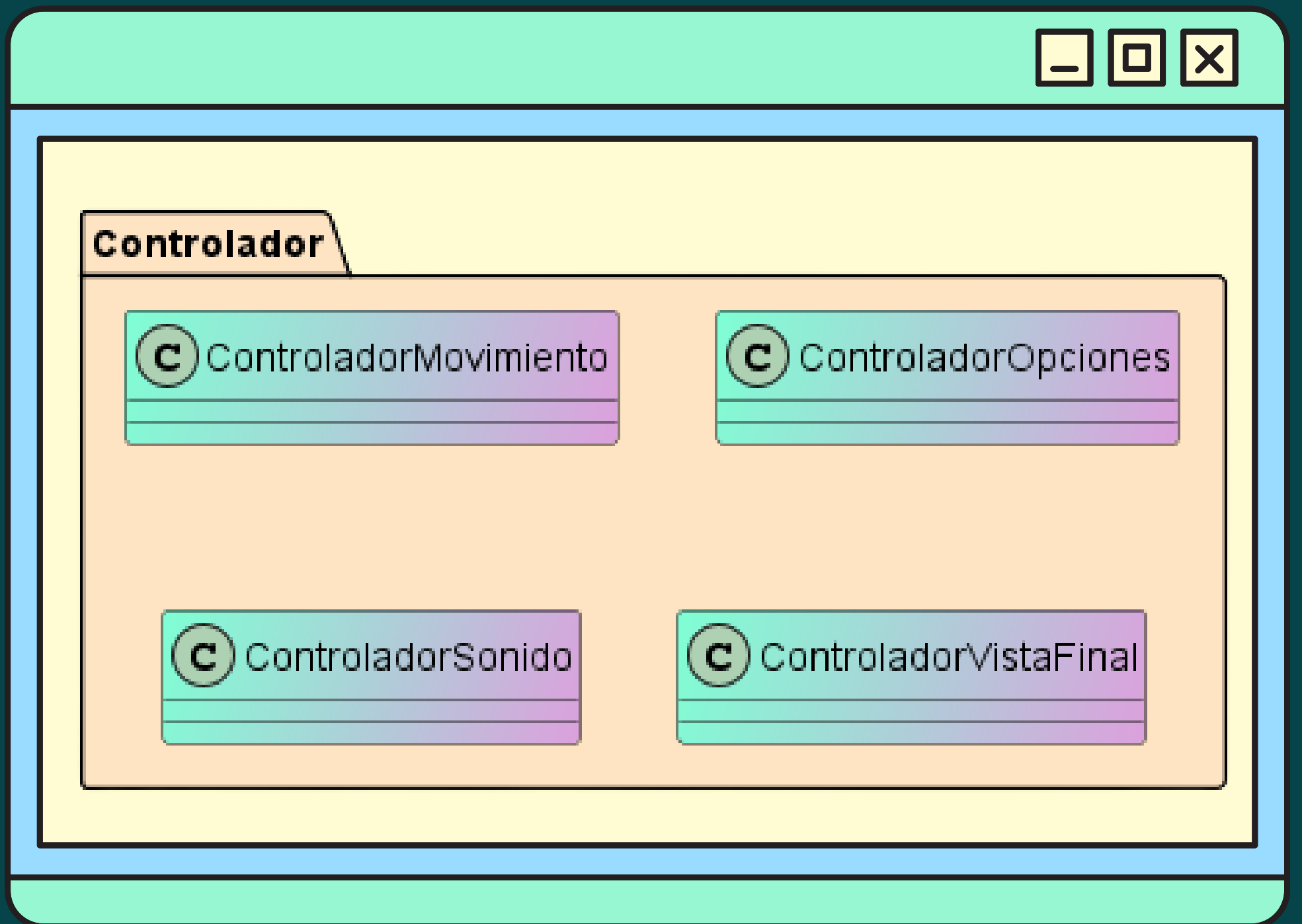


**Patrón Modelo-Vista-Controlador (MVC)**

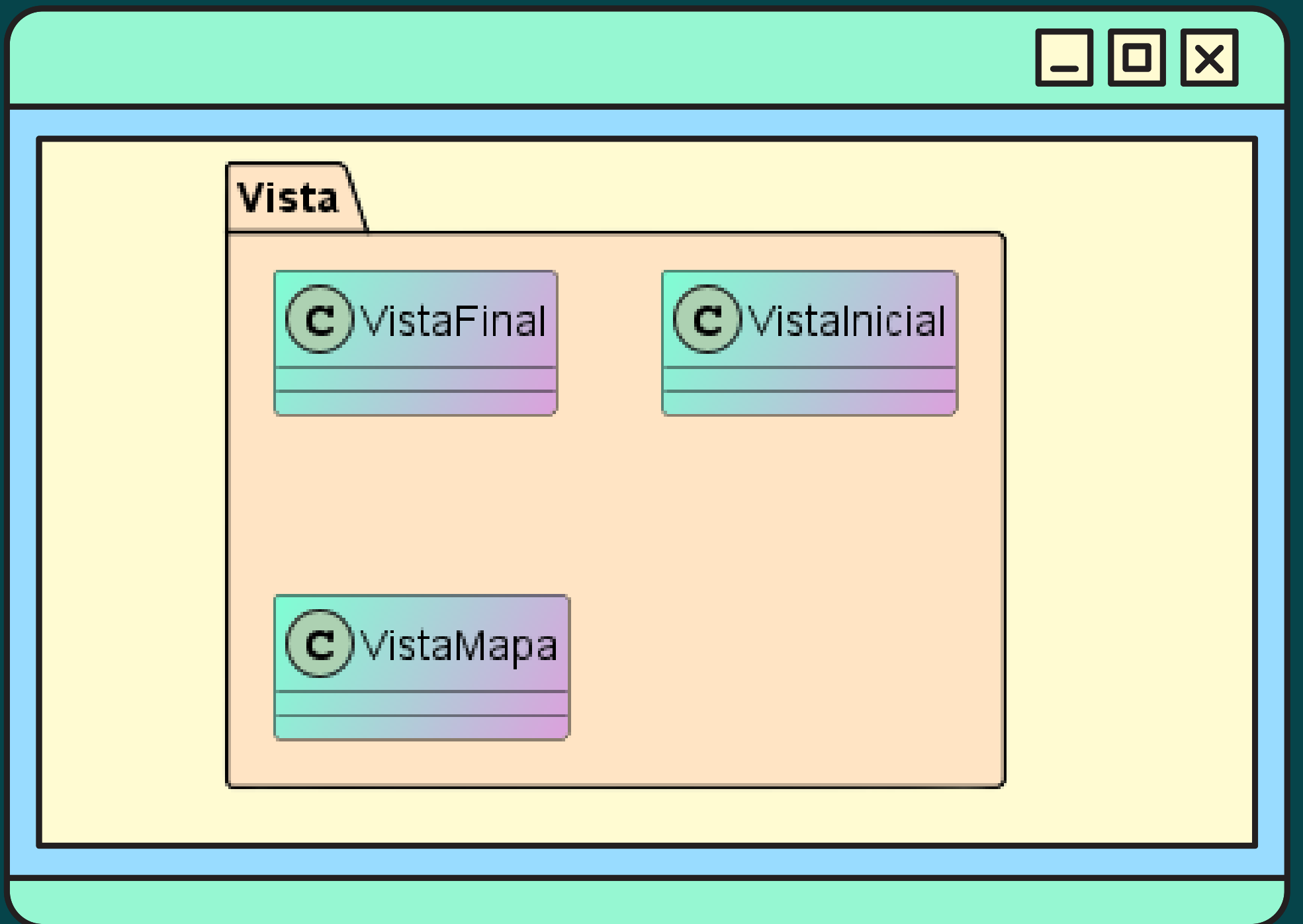


**Subpaquetes del Modelo**

# DIAGRAMA DE PAQUETES



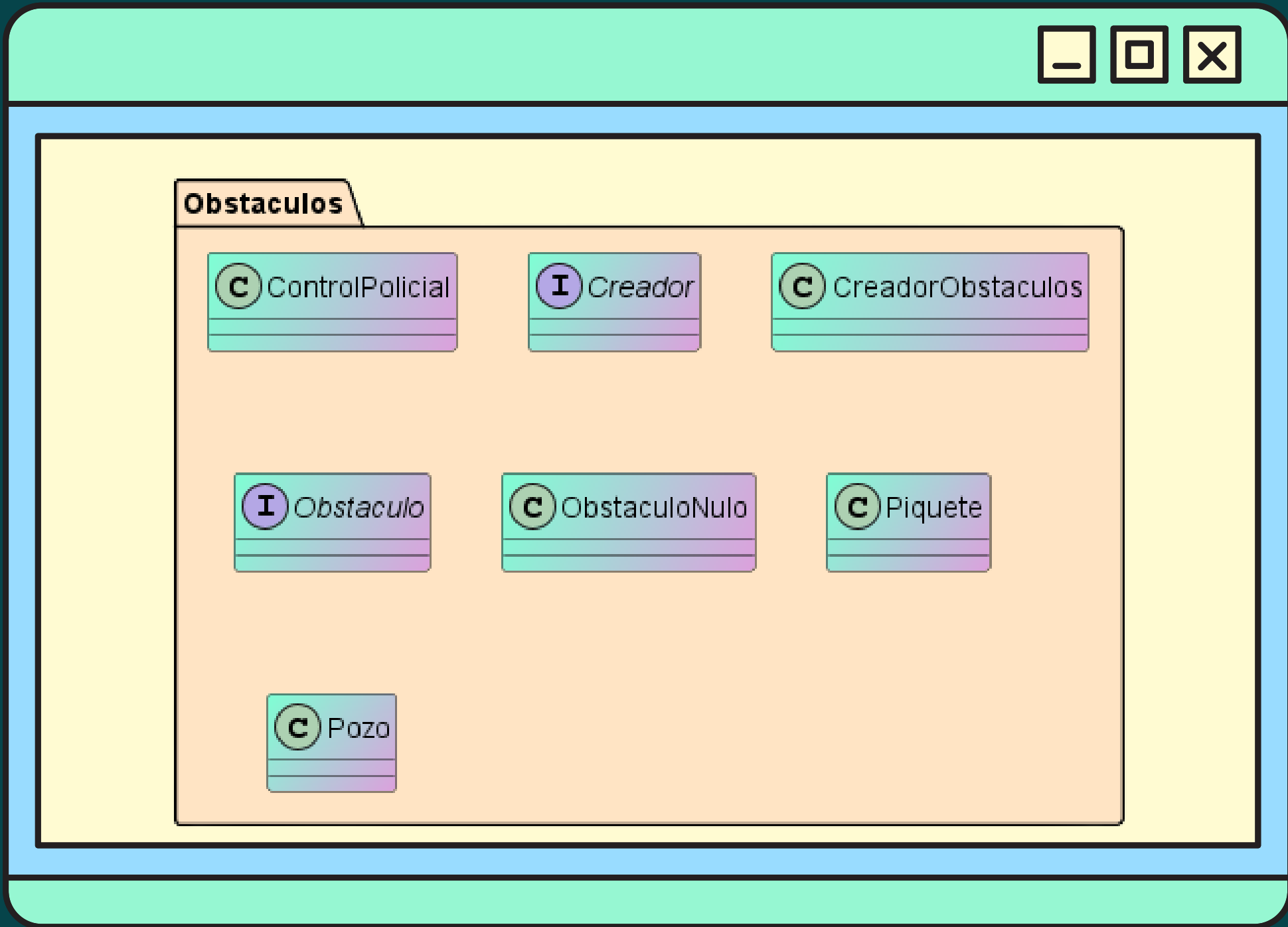
Subpaquetes del Controlador



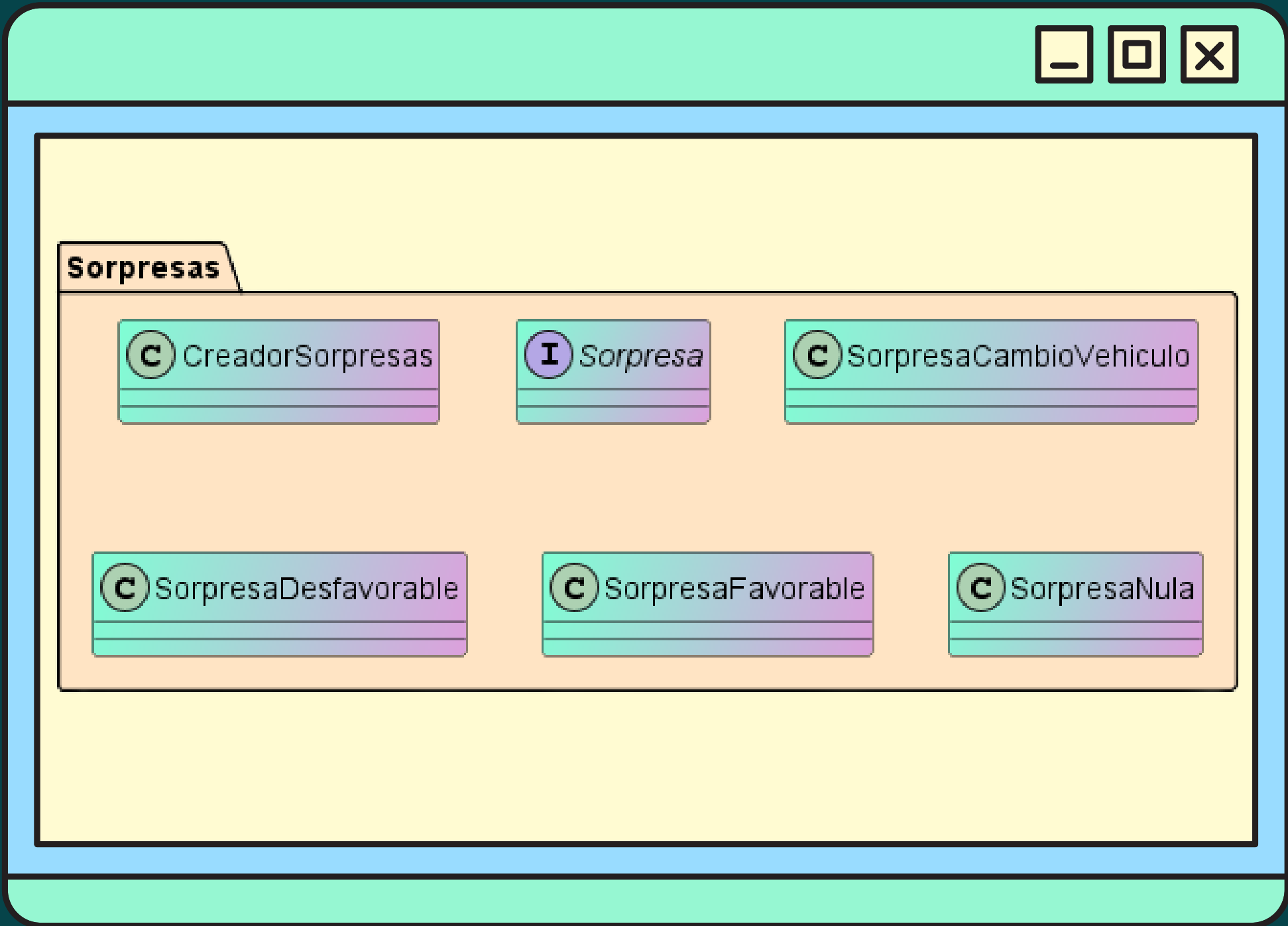
Subpaquetes de Vista



# DIAGRAMA DE PAQUETES

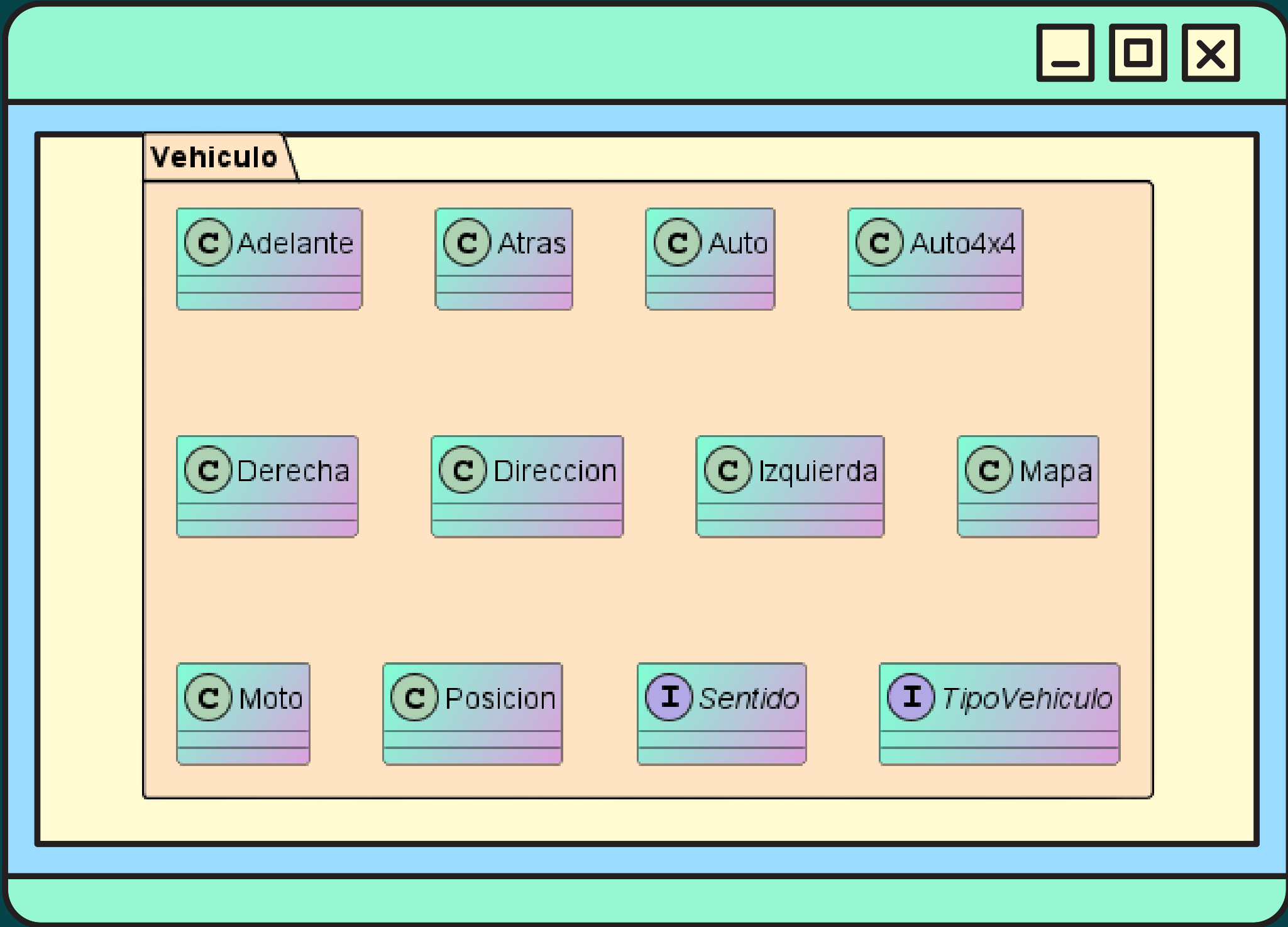


Subpaquete de Obstaculos



Subpaquetes Sorpresas

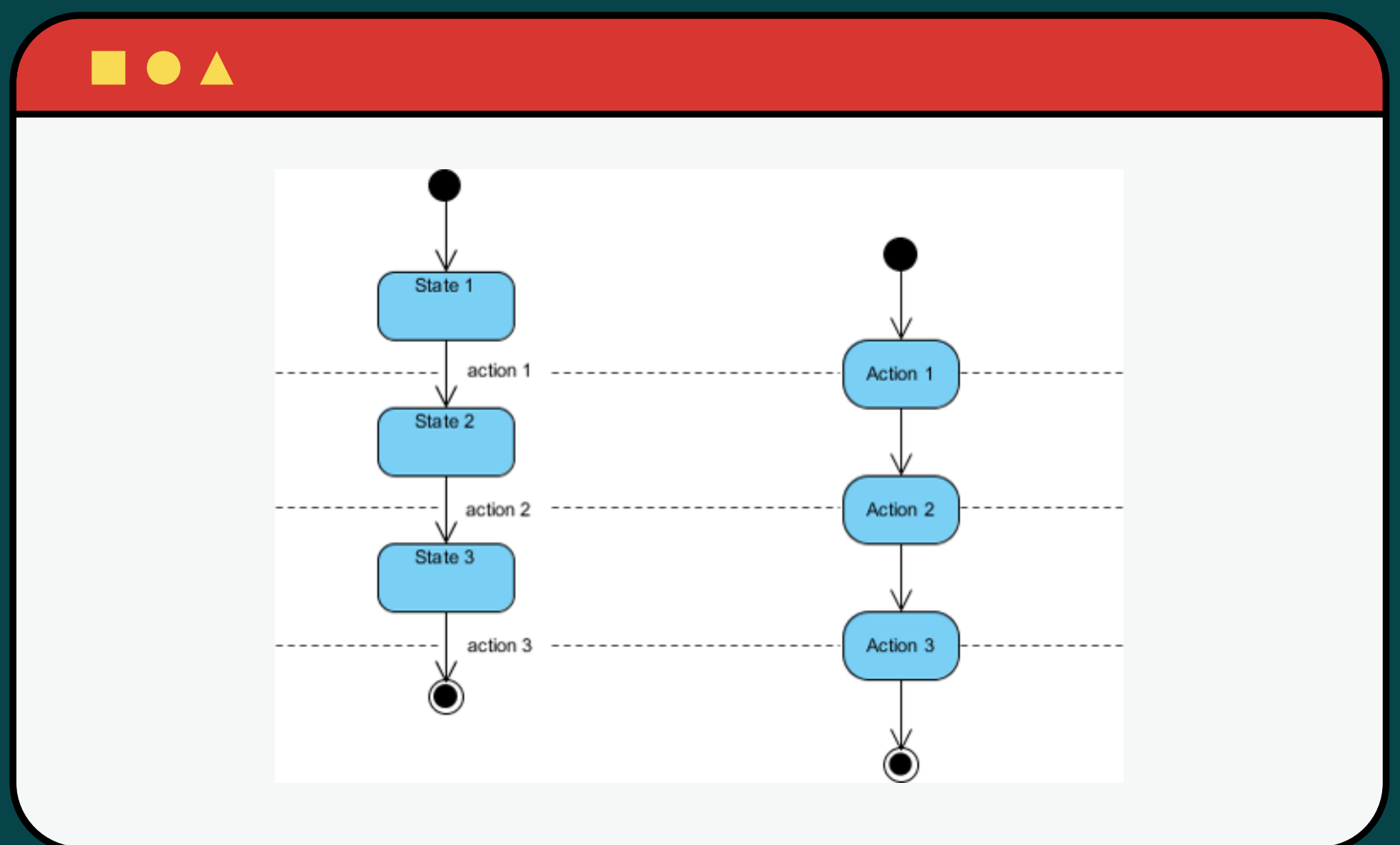
# OTAGRAMA DE PAQUETES



Subpaquete de Vehiculo

# DIAGRAMA DE ESTADO

El diagrama de estados de UML es una herramienta que sirve para modelar cómo afecta un escenario a los estados que un objeto toma, en conjunto con los eventos que provocan las transiciones de estado.

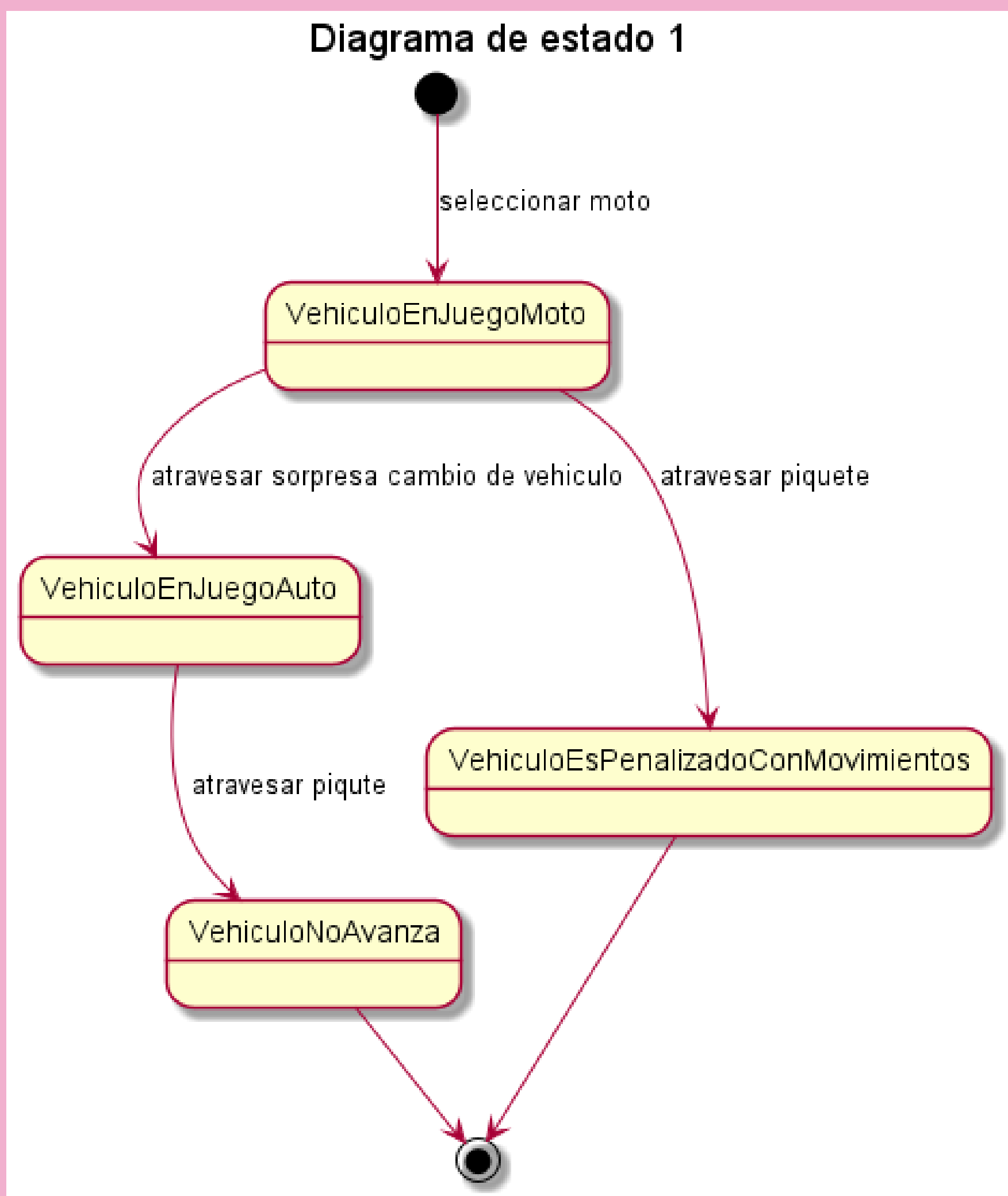


En este apartado se muestra los diagramas de estado.

# Diagrama de Estado de

# Integrando

# Estados



# DE TRATAMIENTO DE PATRONES

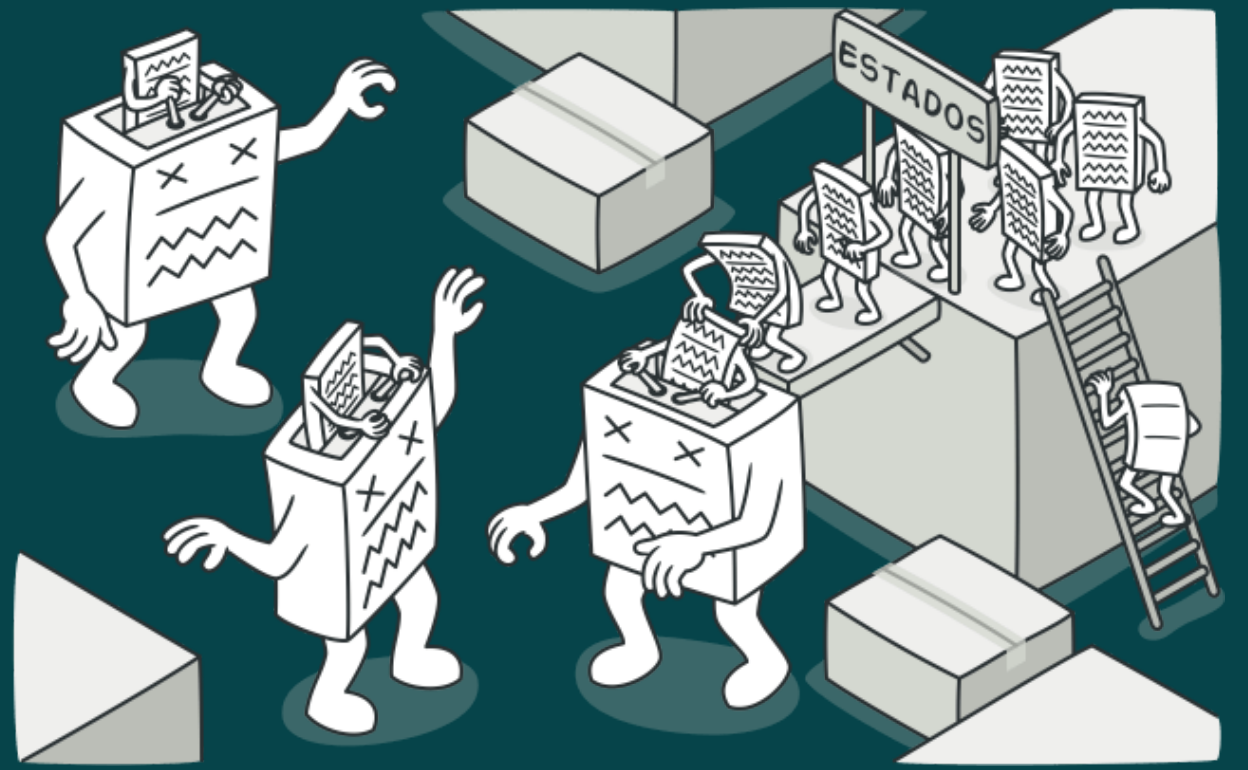
Para este apartado mostraremos los distintos patrones de diseños que fueron implementados con el objetivo de solucionar las distintas problemáticas que surgieron.



Los patrones de diseño nos indican cómo utilizar clases y objetos de formas conocidas y estudiadas, de modo de adaptarlos a la resolución de parte de un problema o a un escenario particular

# PATRÓN DE ESTADO

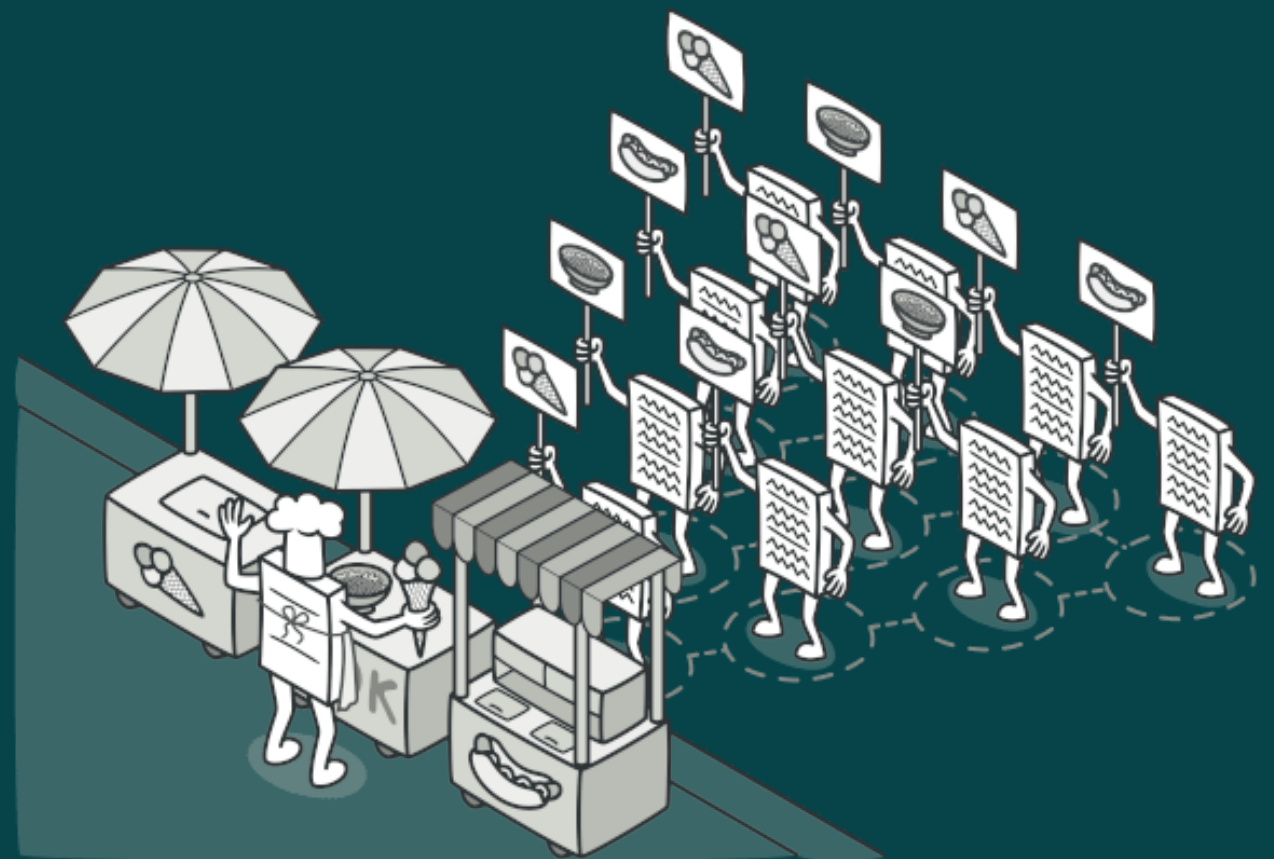
## STATE



State es un patrón de diseño de comportamiento que permite a un objeto alterar su comportamiento cuando su estado interno cambia. Parece como si el objeto cambiara su clase.

Aplicamos este patrón para diseñar el funcionamiento de la SorpresaCambioVehículo. El vehículo en juego será siempre el mismo que aloja un atributo TipoVehículo dentro suyo, imitando un estado, que cambiará al momento de pasar por dicha sorpresa. Con esto respetamos los principios SOLID de responsabilidad única (al organizar código relacionado con estados particulares en clases separadas) y de abierto/cerrado (al introducir nuevos comportamientos sin crear nuevas clases de estado ya existentes).

## VISITOR



Visitor es un patrón de diseño de comportamiento que te permite separar algoritmos de los objetos sobre los que operan.

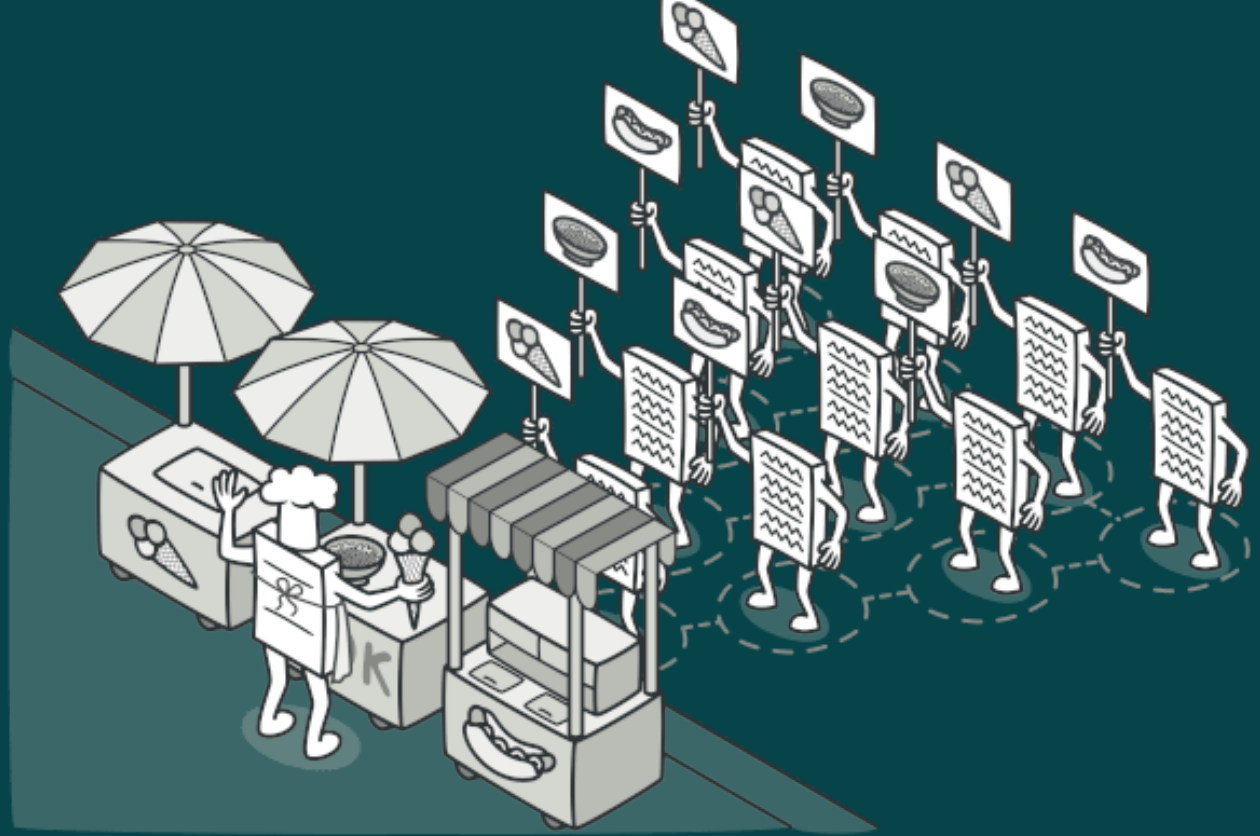
Pensamos los obstáculos como visitors que mediante la ayuda del patrón Double Dispatch ejecutan el método adecuado sobre un objeto sin complicados condicionales. Como el vehículo conoce sus propias clases de tipo, podrá elegir un método adecuado en el visitante más fácilmente. “Aceptan” un visitante y le dicen qué método visitante debe ejecutarse. Es especialmente útil al trabajarse con múltiples objetos que aplicarán este comportamiento repetidas veces.



# PATRONES DE

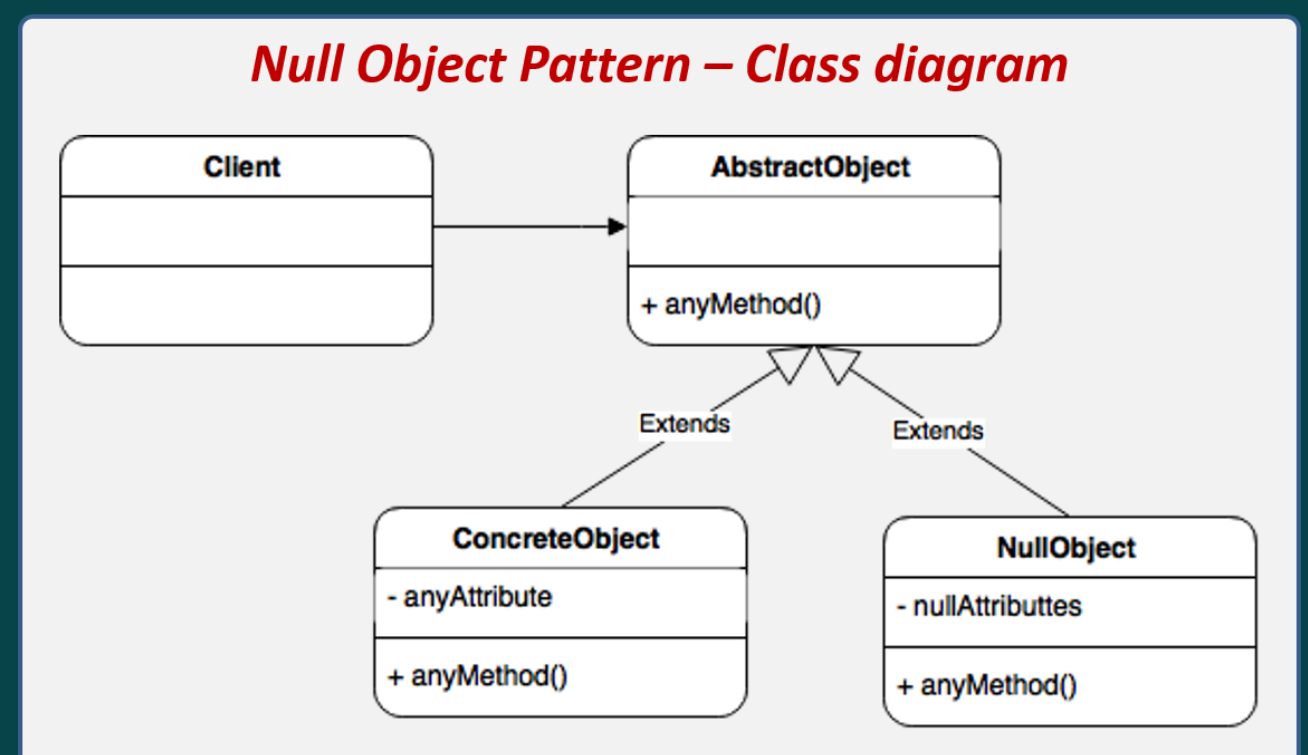
# TRANSITO

## DOUBLE DISPATCH



Como se menciona anteriormente, este patrón se utiliza con el propósito de evitar condicionales de manera similar a lo que logra el polimorfismo. En nuestro caso lo aplicamos junto al Visitor para que cada tipo de vehículo sea penalizado de la manera correspondiente a su tipo.

## NULL OBJECT



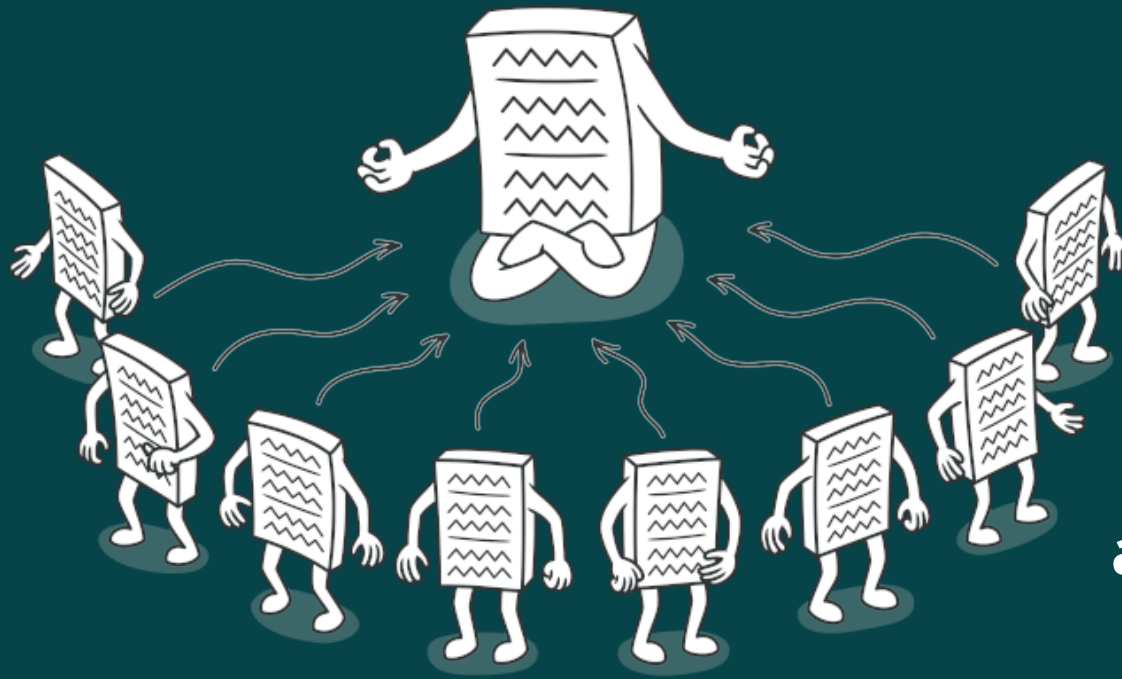
Null Object nace de la necesidad de evitar los valores nulos que puedan originar error en tiempo de ejecución. Fue necesaria la creación de objetos nulos o ficticios que simulen el comportamiento de los reales. En nuestro caso el mapa tendrá obstáculos y sorpresas nulas ubicadas que al pasar por ellas no activarán ninguna acción, es decir, tendrán comportamiento nulo. Con este patrón nos deshacemos de tratar nulls con ifs.

## MULTITON

Lo utilizamos para las posiciones de obstáculos y sorpresas mezclado con el uso de Singleton y HashMaps. Este patrón nos permite generar múltiples instancias de Posición (key del hash) que estarán asignadas a los diversos objetos (value del hash) pero a la vez nos garantiza que no pueda haber diversas instancias de una misma posición. Por ejemplo, la posición (1,1) podrá ser instanciada una única vez y si se la intenta crear de nuevo se devuelve la referencia a la instancia previamente creada.

# DE LOS PATRONES

## SINGLETON



Singleton es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia. En nuestra implementación utilizamos el patrón Singleton para Gameplay

## MVC

MVC (Model-View-Controller) Pattern. Este patrón lo aplicamos para dividir los problemas de la aplicación.



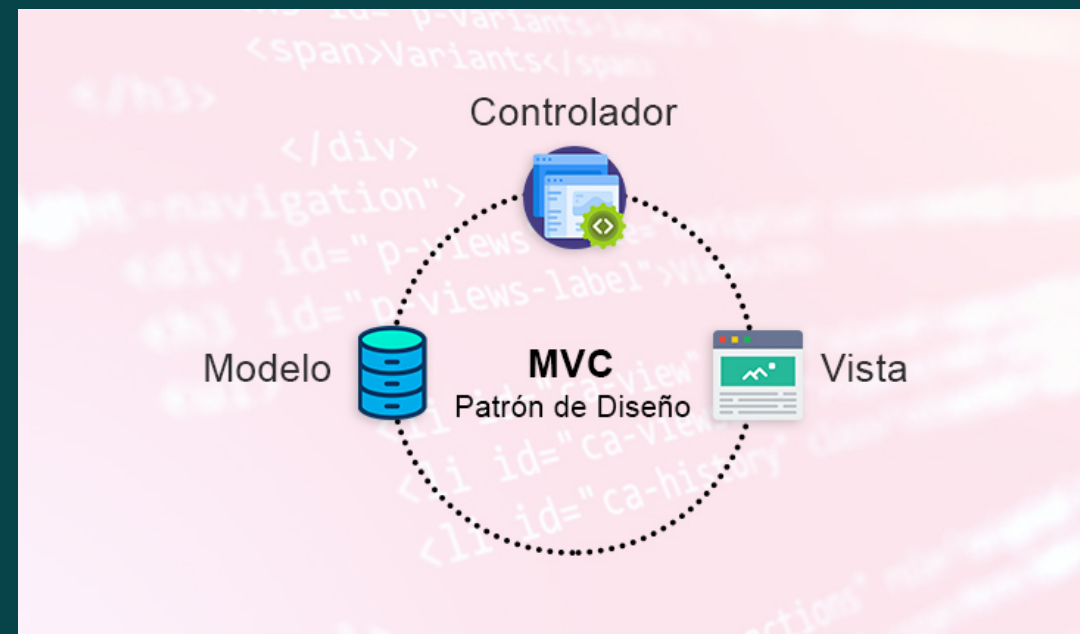
El controlador actúa sobre el modelo y la vista. Controla el flujo de data hacia el modelo y actualiza la vista cuando la data cambia. Mantiene vista y modelo separados.



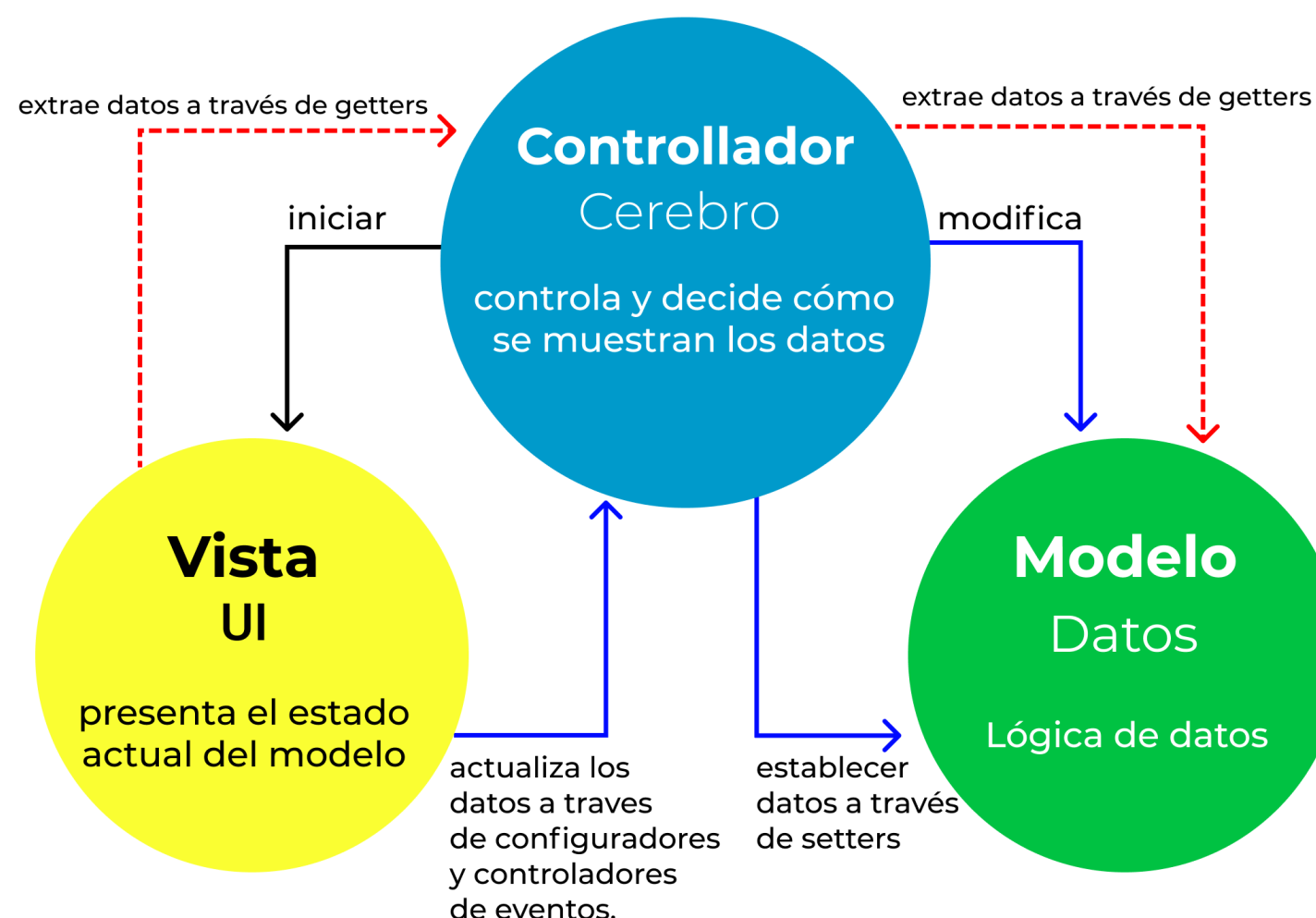
La vista representa la visualización gráfica de la información que contiene el modelo.



El modelo representa un objeto que transporta la data. Contiene la lógica para actualizar el controlador en caso de que la data cambie.



## Patrones de Arquitectura MVC



# SENZONEXCEPTIONS

Una excepción es un objeto que el receptor de un mensaje envía a su cliente como aviso de que el propio cliente no está cumpliendo con alguna precondición de ese mensaje. En este caso el juego presenta las siguientes excepciones:

