



FACULTAD DE INGENIERÍA
UNIVERSIDAD DE BUENOS AIRES

DEPARTAMENTO DE COMPUTACIÓN
75.07 ALGORITMOS Y PROGRAMACIÓN III

CURSO 2

1^{er} CUATRIMESTRE 2022

GPS CHALLENGE

Autores:

Facundo BAEZ (97733)
fbaez@fi.uba.ar

Cristian TORALES (95549)
ctorales@fi.uba.ar

Agustín PIPERNO (96857)
apiperno@fi.uba.ar

Matías VIÑAS (99705)
mvinas@fi.uba.ar

5 de julio de 2022

Índice

1. Introducción	2
2. Supuestos	2
3. Desarrollo	3
3.1. Método de Desarrollo	3
3.2. Clases utilizadas	4
3.3. Movimiento de vehículos	5
3.4. Pilares de POO	5
3.5. Patrones utilizados	6
3.5.1. Singleton	6
3.5.2. Strategy	6
3.5.3. Double dispatch	6
3.6. Excepciones	6
3.7. Interfaz gráfica	6
3.7.1. Menús de Opciones	7
3.7.2. Juego y grilla	8
3.7.3. Barra Desplegable	9
3.7.4. Sonidos y musica	9
3.8. Diagramas UML	10
3.8.1. Diagramas de Packages	10
3.8.2. Diagramas de Clase	11
3.8.3. Diagramas de secuencia	16
3.9. Testeo del código	20
4. Conclusiones	21
5. Bibliografía	21

1. Introducción

Durante este Trabajo Practico N°2 se desarrolló una aplicación denominada GPS CHALLENGE, un juego de estrategia por turnos, donde el escenario es una ciudad y el objetivo es guiar a un vehículo a la meta en la menor cantidad de movimientos posibles.

El mismo se implementó con un lenguaje de tipado estático (Java), y fue llevado a cabo utilizando el paradigma de programación orientada a objetos (POO), TDD (Test Driven Development), e integración continua mediante el uso de GitHub Actions.

Se desarrolló la aplicación por completo, incluyendo el modelo de clases e interfaz gráfica. La aplicación es acompañada por pruebas unitarias e integrales y documentación de diseño.

2. Supuestos

El jugador podrá optar por tres diferentes tipos de vehículos:

- moto
- auto
- 4x4

Al atravesar una cuadra el jugador se podrá encontrar con alguno de los siguientes obstáculos:

- Pozos: Le suma 3 movimientos de penalización a autos y motos. Para una 4x4 penaliza en 2 movimientos luego de atravesar 3 pozos.
- Piquete: Autos y 4x4 deben pegar la vuelta, no pueden pasar. Las motos pueden pasar con una penalización de 2 movimientos.
- Control Policial: Para todos los vehículos la penalización es de 3 movimientos, sin embargo la probabilidad de que el vehículo quede demorado por el control y sea penalizado es de 0,3 para las 4x4, 0,5 para los autos y 0,8 para las motos.

También, cuando un vehículo este en circulación se podrán encontrar diferentes tipos de sorpresas:

- Sorpresa Favorable: Resta el 20 % de los movimientos hechos.
- Sorpresa Desfavorable: Suma el 25 % de los movimientos hechos.
- Sorpresa Cambio de Vehículo: Cambia el vehículo del jugador. Si es una moto, la convierte en auto. Si es un auto lo convierte en 4x4. Si es una 4x4 la convierte en moto.

El juego se desarrolla en un escenario que representa a una ciudad con sus calles y esquinas. En cada turno el usuario elige una de las cuatro direcciones para avanzar con su vehículo. Las sorpresas y los obstáculos que afectan al vehículo se encuentran en las calles que unen a las esquinas.

El jugador no puede visualizar más de dos manzanas a la redonda de la posición de su vehículo y la bandera que marca la meta. El resto del mapa permanecerá en sombras. Esto sucede individualmente para cada jugador, en su turno correspondiente. Esto quiere decir que el rango visible cambia en cada turno, según la posición de cada jugador, si los jugadores están lo suficientemente alejados, se podrá notar mejor la diferencia del rango visible.

El tamaño del escenario no es fijo: pueden definirse sus dimensiones al comienzo del juego. Por otro lado, el escenario tiene un punto de partida y una meta. Y las sorpresas y los obstáculos se generan de manera aleatoria por distintos puntos del escenario.

Se decidió que al pasar por una sorpresa, ya sea favorable o no, que la misma se mantenga existente en la misma posición de la grilla.

En los casos de que un auto o 4x4 quiera pasar por un piquete, se señalizará con una pantalla emergente que el movimiento no se puede realizar, el turno del jugador se mantendrá hasta que se realice un movimiento válido, después del movimiento válido, se pasará el turno al siguiente jugador.

En el caso de una moto quiera pasar por un piquete y en el medio tenga una sorpresa favorable que haga el cambio de vehículo a auto, el movimiento no será permitido y se esperará que se realice otro movimiento para continuar.

Los jugadores comienzan todos desde la misma posición, desde la posición medio de la grilla a la izquierda. Cada jugador tiene un color de fondo que los diferencia.

El juego comienza siempre con cada jugador teniendo una moto, se podrá cambiar a auto o 4x4 cuando se encuentre una sorpresa de cambio de vehículo.

Las cajas de sorpresas son indistinguibles, es decir, que no se saben que contienen o si son favorables o no hasta pasar por las sorpresas.

La meta se encuentra en la columna final, en posición aleatoria dentro de esa columna.

El juego finaliza cuando uno de los jugadores llega a la meta, en la tabla de puntajes sólo se tendrá en cuenta el puntaje del jugador que llegó a la meta. En el ranking que se muestra de mejores puntajes, el mejor jugador del listado es el que tiene menor cantidad de puntos.

3. Desarrollo

La aplicación se organizó según el patrón MVC, por lo que los principales paquetes en el código útil son modelo, controlador y vista. El modelo tiene su paquete de testeos en la carpeta de tests.

3.1. Método de Desarrollo

El desarrollo se realizó dividiendo por módulos para obtener tareas que puedan ser trabajadas de forma individual, esto se logró utilizando branches de github y luego realizando un cuidadoso merge entre los trabajos realizados. Y además buena parte del código se realizó trabajando de a pares, esto es: mientras un programador codea el, otro lo guía.

Además el desarrollo se realizó de forma iterativa con cinco entregas, una por semana, más una entrega cero donde se realizó la configuración del entorno de programación:

- Repositorio de código creado en Github
- Servidor de integración continua configurado con Github Actions
- Al menos un commit realizado por cada integrante, actualizando el README.md del repositorio.

Mientras que para las entregas 1 y 2 la cátedra proporcionó una serie de test que el modelo debía cumplir, a los que se debió adicionar otros creados propios.

Para las entregas 3, 4 se tuvo el modelo del videojuego terminado y la interfaz gráfica en desarrollo.

Finalmente para la entrega 5 se agrego sonido y se en embelleció la interfaz gráfica.

A su vez el desarrollo fue incremental, esto es que el código se fue mejorando, complejizando y añadiéndole más funciones a medida que pasaban las entregas.

Para la escritura del código de la aplicación se utilizaron prácticas ágiles tales como TDD (test driven Development), refactorización e integración continua. Para ello primero se implementaron los tests unitarios y se desarrollaron las clases para poder hacer pasar los tests. Luego se refactorizó el código, para aportar mayor legibilidad, mantenibilidad y compatibilidad con los pilares de POO. Todo el proceso descrito se realiza de forma iterativa. Además, se agregaron los tests por clase.

Para los test unitarios se utilizo el framework de mockito, con el fin de que para las clase que poseían una fuerte dependencia puedan ser testeadas por separado. Por último se realizaron pruebas integrales que prueben que estas dependencias funcionen correctamente.

3.2. Clases utilizadas

Las siguientes clases son las forman parte de la solución de los requerimientos de la aplicación GPS CHALLENGE:

- Juego: Representa al juego en sí, posee el listado de jugadores y administra sus turnos, además cuenta con un listado de los jugadores históricos con el puntaje más alto. Interactúa con la interfaz gráfica para mover de posición a los vehículos de los jugadores.
- PuntajesAltos: Es un listado de jugadores que se encuentra ordenado por puntaje, cada vez que se agrega un nuevo jugador este listado se reordena. Es capaz de exportar e importar un archivo .JSON para conservar los datos una vez cerrada la aplicación.
- Jugador: Posee su nickname configurable al inicio de la partida, su puntaje y un Vehículo el cual podrá mover hacia distintas direcciones con el fin de llegar a la meta
- Grilla: Es la representación del mapa de la ciudad, conformada por una matriz de posiciones y un listado de Ubicables (Punto de Partida, Meta, Sorpresas y Obstáculos) con su posición, estos ubicables pueden ser generados de forma aleatoria en una posición también aleatoria. Cada vez que un vehículo se mueve se verifica si "piso.^Igún ubicable y si esto sucedió se le aplica al vehículo.
- Vehículo: Cada uno de los jugadores posee uno. Puede cambiar de posición relativa según cada dirección proporcionada. Contiene un cantidad de movimientos acumulados y tiene como atributo a un TipoDeVehiculo y delega en ella como le afectan los distintos tipos de ubicables que se pueden encontrar por la grilla.
- TipoDeVehiculo: Clase abstracta, puede ser tanto una Moto, un Auto o una Cuatro por Cuatro, en alguna de estas clases herederas delega el vehículo su comportamiento cuando se topa con un Obstáculo.
- Moto: Es un tipo de vehículo, cuando el vehículo se tope con Pozo se calculará un penalización de 3 movimientos, para un Control Policial habrá un gran probabilidad de ser atrapado (porque siempre van sin casco) y penalizado con 3 movimientos, es el único capaz de pasar por un piquete, aunque con una penalización de 2 movimientos. El siguiente tipo de vehículo es un Auto.
- Auto: Es un tipo de vehículo, cuando el vehículo se tope con Pozo se calculará un penalización de 3 movimientos, para un Control Policial habrá un probabilidad de ser atrapado y penalizado con 3 movimientos, en el caso de intentar pasar por un piquete se lanzara una excepción avisando que esto no posible. El siguiente tipo de vehículo es una cuatro por cuatro.
- CuatroPorCuatro: Es un tipo de vehículo, cuando el vehículo se tope con Pozo a diferencia de los demás no recibirá una penalización, a menos que haya pasado por 2 piquetes previamente, para un Control Policial habrá un probabilidad de ser atrapado y penalizado con 3 movimientos, en el caso de intentar pasar por un piquete se lanzara una excepción avisando que esto no posible. El siguiente tipo de vehículo es una moto.
- Ubicable: Es una interfaz por lo que no tiene estado (no posee atributos). Las clases que implementen a la interfaz deben implementar serEncotradoPor (unVehiculo).
- Obstaculo: Es una interfaz por lo que no tiene estado (no posee atributos). Son elementos que afectan negativamente a un vehículo
- ControlPolicial: Posee una Posición, puede ser encontrado por un vehículo, tiene una probabilidad de afectar negativamente la cantidad movimientos del vehículo, dependiendo del tipo de vehículo.
- Pozo: Posee una Posición, puede ser encontrado por un vehículo, afecta negativamente a la cantidad de movimientos, a excepción de 4x4 según que condiciones.
- Piquete: Posee una Posición, puede ser encontrado por un vehículo, no permite que el vehículo pase por el arrojando una excepción, a menos que sea una moto.
- Sorpresa: Es una interfaz por lo que no tiene estado (no posee atributos). Son elementos que afectan a un vehículo de forma aleatoria, no se sabe de que tipo es hasta encontrarla.

- SorpresaFavorable: Es una implementación de sorpresa, posee una Posición y el vehículo que la encuentra vera reducida su cantidad de movimientos en un 20 por ciento.
- SorpresaNoFavorable: Es una implementación de sorpresa, posee una Posición y el vehículo que la encuentra vera aumentada su cantidad de movimientos en un 25 por ciento.
- CambioDeVehiculo: Es una implementación de sorpresa, posee una Posición y el vehículo que la encuentra cambiara de tipo de vehículo según su tipo actual.
- Posición: Se trata de una coordenada en X y otra en Y, puede obtener la suma de 2 posiciones y comparar posiciones.
- Dirección: Es una interfaz por lo que no tiene estado (no posee atributos). Las clases que implementan esta interfaz son Derecha, Izquierda, Arriba y Abajo, y están obligadas a implementar posicionRelativa que devuelve una posición que sumada a la posición de un vehículo se obtendrá el desplazamiento del mismo.

3.3. Movimiento de vehículos

Para el movimiento de los vehículos se decidió que los mismos realizarán movimientos de dos posiciones de la Grilla y esto representa a un movimiento del mismo en el juego. Esto se realizó de esta manera debido a que antes de que el Vehículo llegue a la posición siguiente, tiene que atravesar un elemento ubicable, por ejemplo un obstáculo o sorpresa.

Como al atravesar las posiciones de elementos ubicables, podría suceder el caso en que no tenga ningún elemento, para esto se utilizó el patrón de diseño "Null Object" que permite crear un objeto nulo que represente a esos elementos que no deberían afectar.

Para evitar respetar los principios de encapsulamiento, al momento de realizar comparaciones se realizaron comparaciones entre clases. Con esto se evitó la utilización de getters para comparaciones. Por ejemplo: se utilizó esta estrategia para comparar posiciones y puntajes, para el caso de los puntajes se realizaron las comparaciones entre jugadores ya que éstos tienen como atributo a los puntajes.

3.4. Pilares de POO

Los pilares del paradigma que se emplearon para modelar las clases fueron: abstracción, encapsulamiento, polimorfismo, delegación y herencia.

La abstracción expresa las características esenciales de un objeto, las cuales distinguen al objeto de los demás. Además de distinguirlos, provee límites conceptuales. Este enfoque consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan para un estudio específico y considerando solo las propiedades esenciales para dicho análisis. Durante todo el desarrollo del código se aplicó este pilar, por ejemplo al poder representar a la ciudad (el escenario) como una Grilla con Posiciones, que para el modelo bastaba pero para otros tipos de videojuegos posiblemente será representada de otra forma completamente distinta.

El encapsulamiento es el ocultamiento de la implementación. Cada objeto es responsable de responder a los mensajes que recibe, sin que quien le envía el mensaje tenga que saber cómo lo hace. Las ventajas que conlleva este enfoque es que pueden haber implementaciones alternativas para una misma operación y, por otro lado, se pueden realizar cambios en la implementación sin afectar al cliente que utiliza el servicio. Por ejemplo al comparar dos Posiciones distintas la grilla no viola el encapsulamiento obteniendo las coordenadas de la Posición, si no que utiliza el método IgualA de Posicion ya que cada una de ellas sabe como compararse contra objetos de la misma clase; este mismo mecanismo se utiliza también en otras clases como en Jugador.

La delegación es el mecanismo por el cual un objeto delega una funcionalidad a otros objetos para luego tomar una decisión. Un objeto debe conocer al objeto que le va a delegar una funcionalidad. En la delegación, generalmente, se cumple la relación “contiene”, “hace referencia” y “es parte de”. En ocasiones, se aplica como sustitución de la herencia. La delegación es conveniente si se reutiliza la interfaz sin mantenerla. Un ejemplo de ello es cuando los usuarios interactúan con Juego para mover su vehículo, Juego luego delega en un objeto de Clase Jugador según corresponda y Jugador en el Vehículo que posea y a su vez Vehículo generará un Movimiento y delegara en el para cambiar su posición.

La herencia es una relación entre clases, por la cual se define que una clase puede ser un caso particular de otra (llamada clase hija). Cuando hay herencia, todas las instancias de la clase hija son también instancias de la clase madre. Esta característica permite reutilizar código dado que al definir un comportamiento en la clase madre y usar ese comportamiento sin tener que repetir el código en cada clase hija. La herencia es conveniente si se reutiliza la interfaz tal cual está. En otras palabras, la herencia permite programar por diferencia. En la herencia, generalmente, se cumple la relación “es un”. Este principio fue muy utilizado durante el desarrollo del código, son Auto, 4x4 y Moto que son TipoDeVehiculo (clase abstracta), esta relación de herencia permitió ahorrar escribir mucho código.

El polimorfismo es la capacidad de respuesta que tienen distintos objetos de responder de maneras diferentes a un mismo mensaje. Esto permite aumentar de la reutilización, hacer el código más legible y ocultar detalles de la implementación al interactuar con un grupo de clases diferentes a través de una interfaz común. Se hizo gran uso del polimorfismo por ejemplo, utilizando la interfaz Ubicable quienes la implementan son, Meta, Sorpresa y Obstaculo (sus herederas), están obligados a entender el mensaje serEncontradoPor (Vehiculo unVehiculo), pero cada clase concreta tendrá su implementación resultando en comportamiento complemento distinto.

3.5. Patrones utilizados

3.5.1. Singleton

El patrón de diseño Singleton sirve para que al crear una clase se garantice que solo tenga una instancia y proporcione un punto de acceso global a ella. Las clases Juego y Grilla usan este patrón debido a que son únicos durante todo el juego, y deben ser accedidos desde diversas partes de la aplicación.

3.5.2. Strategy

El patrón Strategy propone una solución al problema de tener una clase que hace algo en particular de muchas formas diferentes y colocar esos algoritmos en clases separadas, de esta manera si se agregan nuevos métodos no hace falta modificar ni la clase original ni las clases de estrategia.

En la clase Vehículo gracias a que se dividió que se iba cambiar de Moto a Auto y a 4x4 se comprendió que era necesario utilizar este patrón. Por lo que se aisló el comportamiento de TipoDeVehiculo en una clase (Abstracta) aparte para facilitar las modificaciones este comportamiento, y se la agrego como atributo dentro de la clase Vehículo.

3.5.3. Double dispatch

El patrón de Double Dispatch permite el uso de la vinculación dinámica junto a métodos sobrecargados.

Se utilizó principalmente en el método de calcularPenalización() de las clases que heredan de TipoDeVehiculo, así se pudo realizar el cálculo de cada penalización según que obstáculo se pasaba por parámetro.

3.6. Excepciones

En el código se utilizaron las siguientes excepciones:

- PosicionFueraDeLimite: Lanzada cuando la grilla detecta que se quiere ir a una posición fuera de esta.
- VehiculoNoPuedePasar: Lanzada por Auto y CuatroPorCuatro cuando intentan calcular la penalización de un obstáculo que resulta ser un piquete.

3.7. Interfaz gráfica

Para el desarrollo de la interfaz gráfica se utilizó el framework JAVAFX, con el se logró mostrar al usuario los menús, el juego en sí, y se permitió que el jugador pueda interactuar libremente y de forma intuitiva con el juego. Además se añadió música y sonidos a los menús y al juego.

El videojuego GPS Challenge La Matanza Edition está ambientado de los barrios bajos del conurbano Bonaerense de la República Argentina.

3.7.1. Menús de Opciones

En los distintos menús del videojuego se le ofrece al usuario la posibilidad de elegir entre las distintas modalidades de juego mediante el uso del click izquierdo del mouse en distintos tipos de botones. Al iniciar la aplicación se aparecerá una ventana principal y se reproducirá la musica principal del juego, y no pasara nada más hasta que usuario presionen en "iniciar juego".

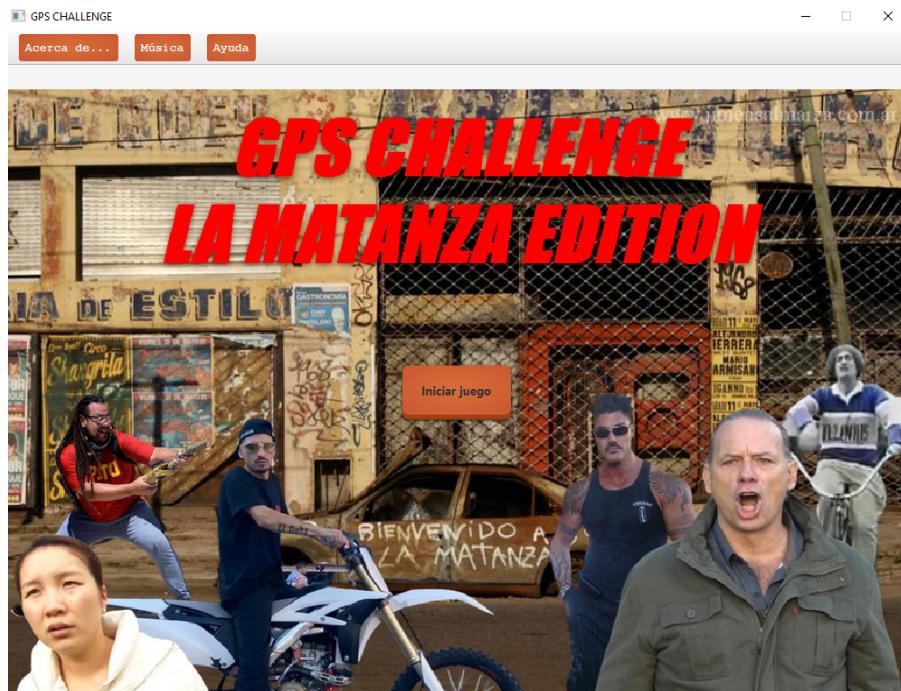


Figura 1. Pantalla del menú principal

En primer lugar se le permite seleccionar entre 3 distintos tamaños de mapas chico, mediano y grande. El juego se escalará según esta decisión mostrando luego menor o mayor cantidad de posiciones, sorpresas y ubicables. En el segundo menú se podrá elegir la cantidad de jugadores, si se le eligió el mapa chico solo se ofrece la opción de jugarlo Singleplayer o de a dos, en cambio para mapas medianos se puede jugar de hasta 3 y para mapas grandes de hasta 4 jugadores.

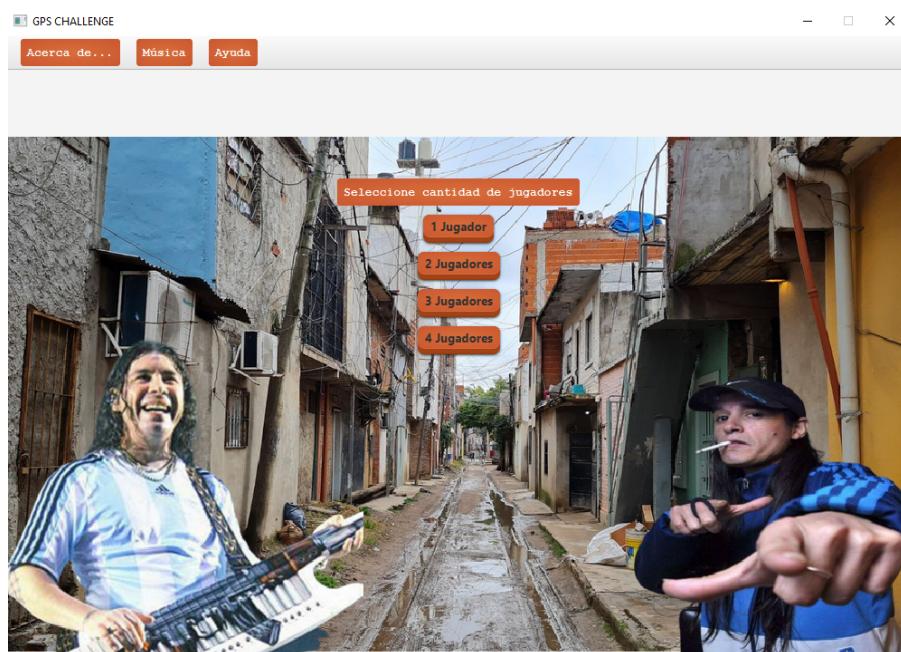


Figura 2. Pantalla del menú de selección de cantidad de jugadores para un mapa grande

Una vez seleccionada la cantidad de jugadores presentes en la partida, aparecerá un tercer menú donde se permitirá escribir de forma ordenada los nombres de cada uno de los jugadores (nicknames), para ello se debe escribir en cada uno de los cuadros de textos y presionar la tecla ENTER. Si se deja un cuadro de texto vacío y se presiona ENTER, el videojuego informará que un nickname vacío no es válido y esperará a que se ingrese uno válido.

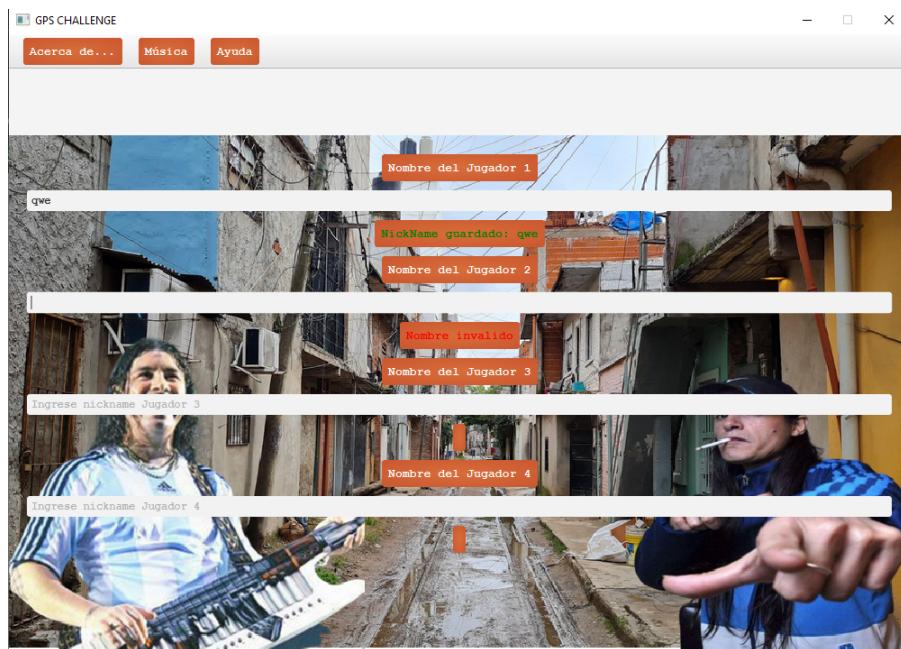


Figura 3. Pantalla del menú de selección indicando un error al ingresar un nombre vacío

3.7.2. Juego y grilla

Luego de haber realizado la configuración de la partida, el juego mostrará de forma gráfica a la grilla que representa a la ciudad, a los vehículos (colocados en principio en la misma posición), a las sorpresas y a los obstáculos. A una distancia mayor de 2 casillas del vehículo que tiene turno actual no se podrán ver los obstáculos y sorpresas, pero si la meta que siempre estará visible.

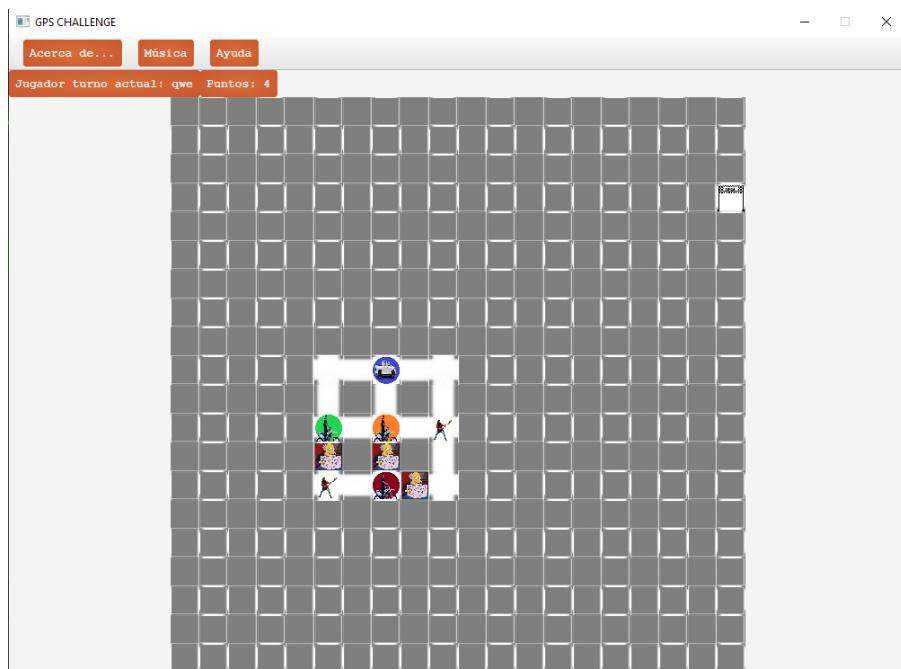


Figura 4. Pantalla del juego transcurriendo con normalidad en la grilla

Se podrá conocer quien es el jugador que posea el turno actual por un cuadro texto en la parte superior de la pantalla que informa el nickname y el puntaje que corresponde. El usuario podrá mover su vehículo utilizando las teclas W, A, S, y D (arriba, izquierda, abajo y derecha respectivamente), una vez presionada una de estas teclas se dará lugar al siguiente jugador en la cola, a excepción de que el jugador haya realizado un movimiento invalido, como irse fuera de la grilla o pasar por un piquete. En cualquiera de estos casos se lanzara una ventana emergente (popup) informado de lo acontecido. Adicional a esto cada jugador tendrá el icono de su vehículo y color para distinguirse de los demás.

Luego de que la partida prosiguió con normalidad, y uno de los participantes llego a la meta se informara con una nueva pantalla de quien fue el jugador ganador. También se mostrara un tabla con los puntajes históricos más bajos. Por ultimo aparecerá un botón para poder jugar nuevamente.

3.7.3. Barra Desplegable

En cada una de las pantallas del videojuego ofrecerá al usuario una barra con items desplegables donde se encontrara:

- Acerca de...: Indica quienes fueron los desarrolladores de la aplicación.
- Ayuda: Da información de cual es el objetivo del juego y de los elementos que se pueden encontrar en el.
- Musica: En este ítem se puede parar la musica o volver a reproducirla.

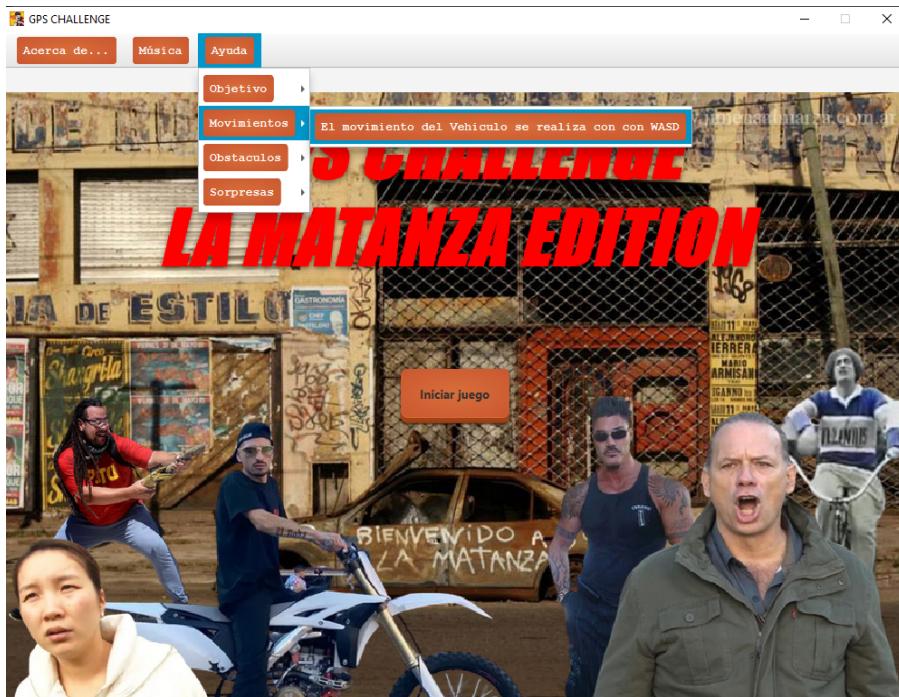


Figura 5. Pantalla del menu de inicio con barra de ayuda desplegada

3.7.4. Sonidos y musica

Además de la musica que acompaña al juego, y con motivo de añadir mayor información de los sucesos del juego que sean fácilmente interpretables de forma audible, se agregaron sonidos para cada uno de los botones de los menús del juego, como también sonidos para los distintos eventos, como cuando un vehículo se encuentra con alguna de las distintas sorpresas o obstáculos como así cuando se intenta realizar un movimiento invalido.

3.8. Diagramas UML

3.8.1. Diagramas de Packages

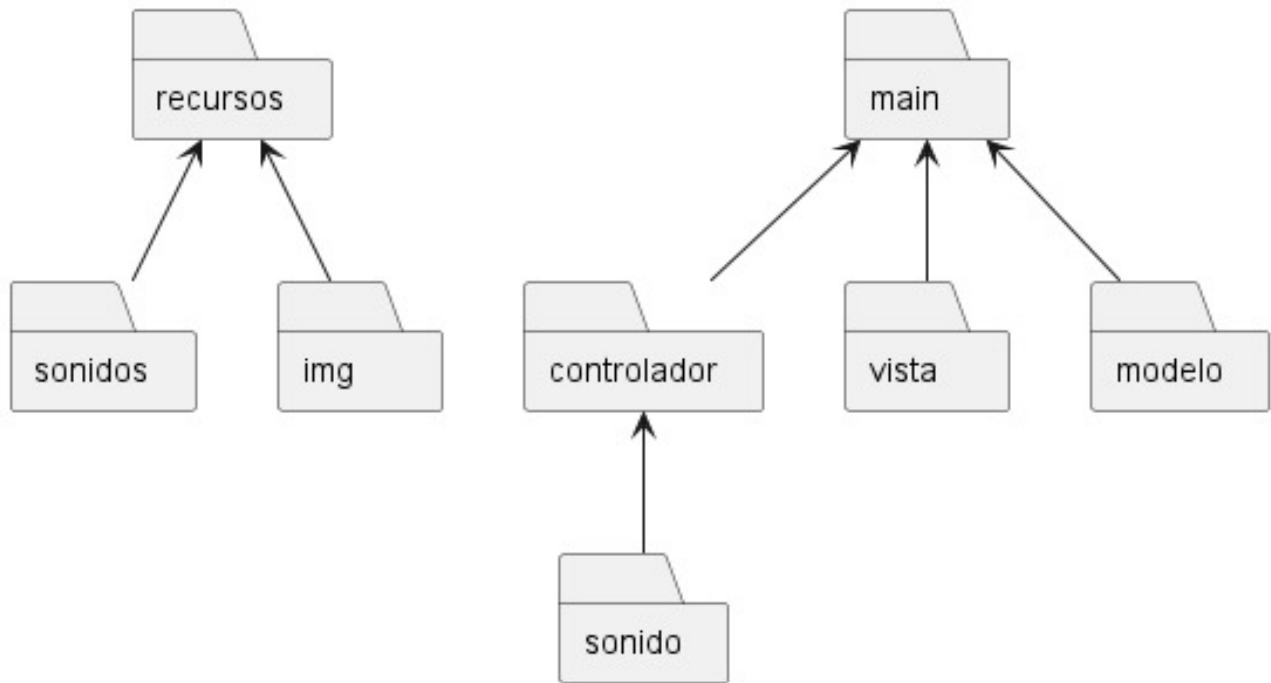


Figura 6. Diagrama de Packages general simplificado

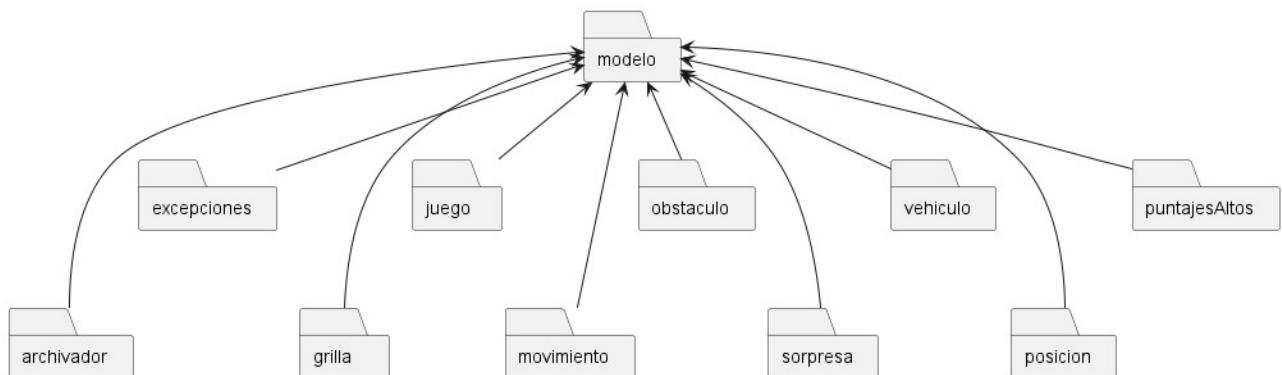


Figura 7. Diagrama de Packages del modelo

3.8.2. Diagramas de Clase

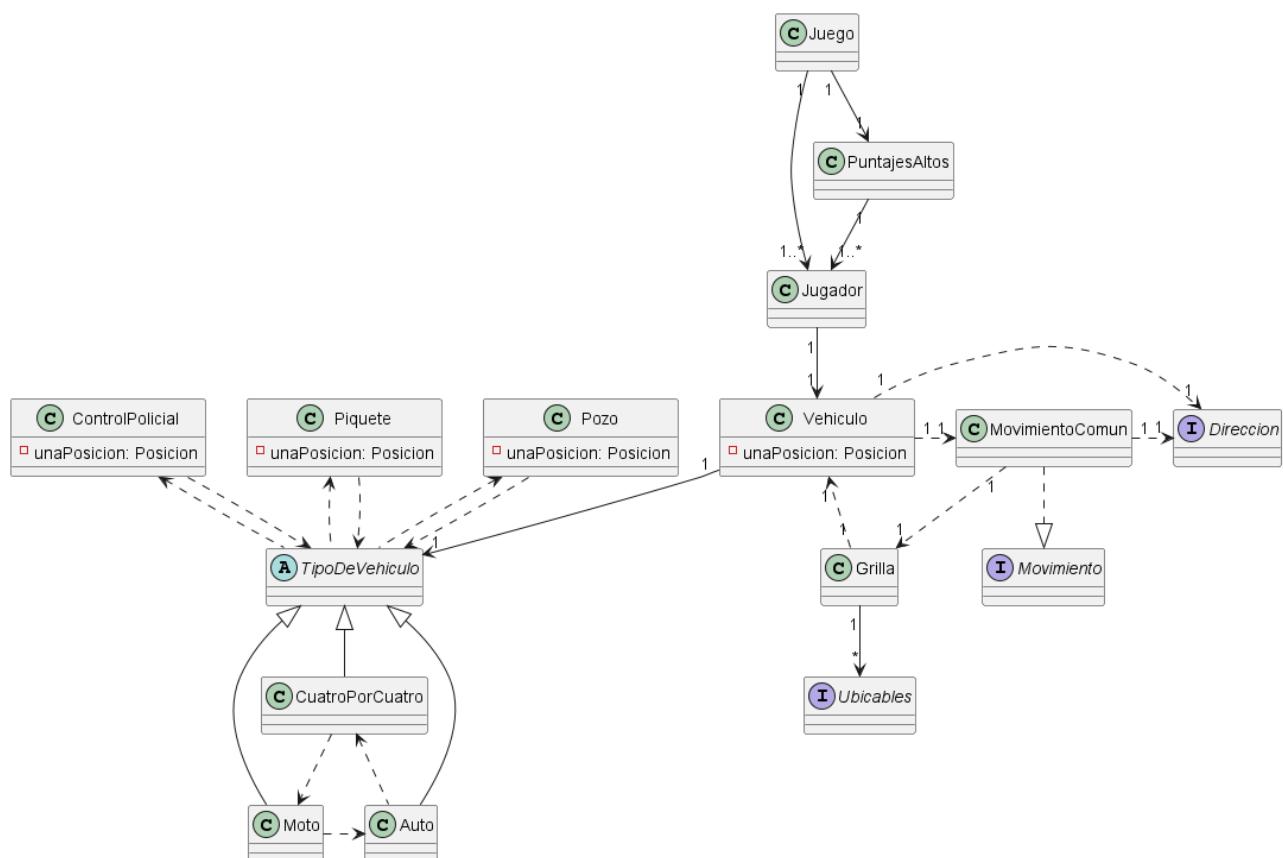
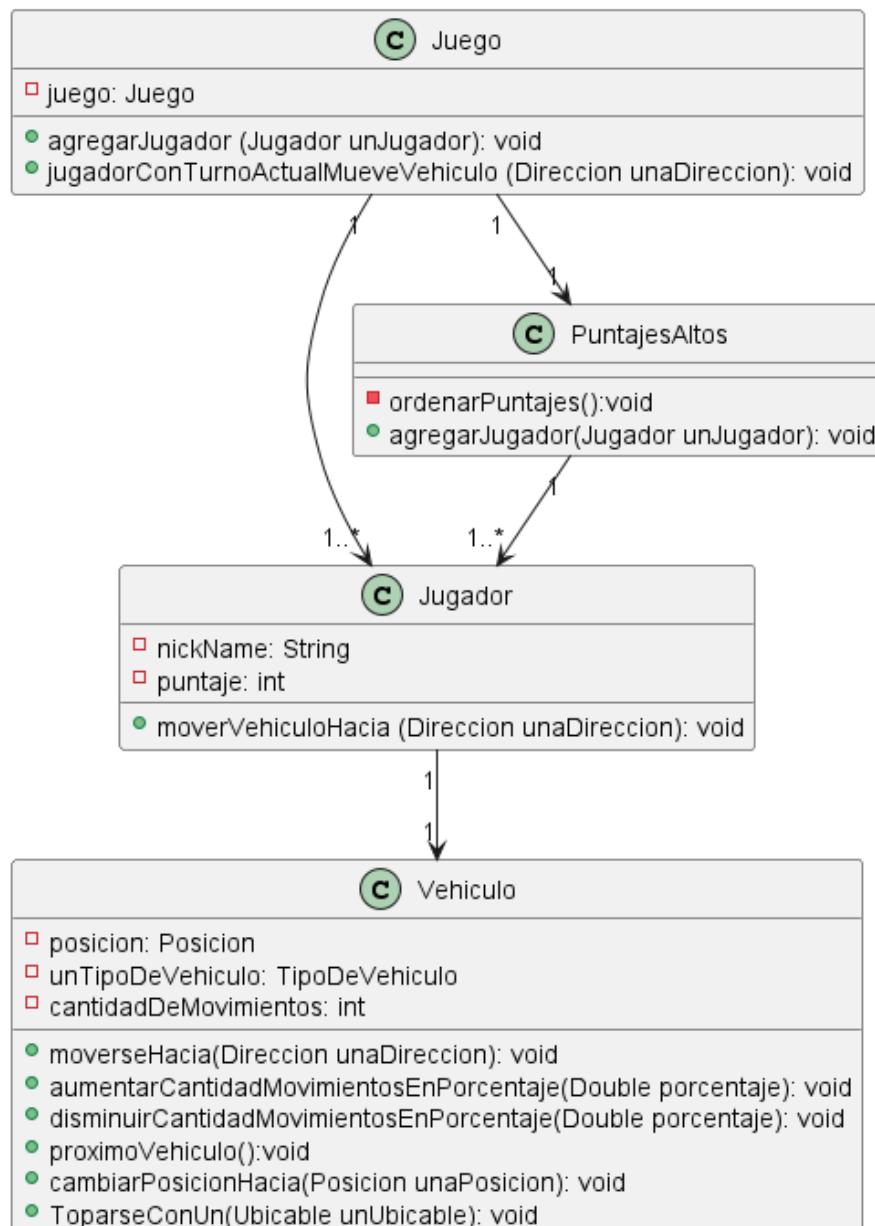


Figura 8. Diagrama de clases general simplificado

Diagrama General**Figura 9.** Diagrama De Clases - singleton Juego

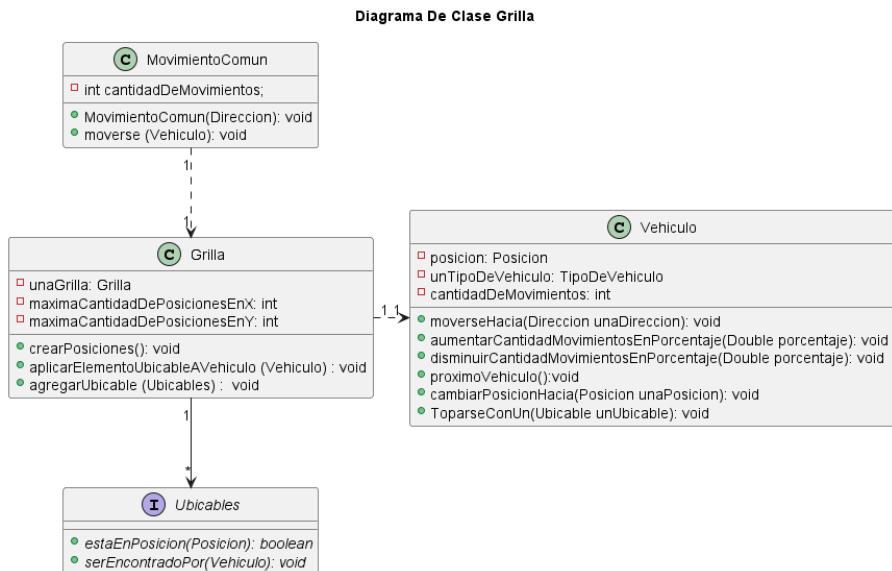


Figura 10. Diagrama De Clases - singleton Grilla

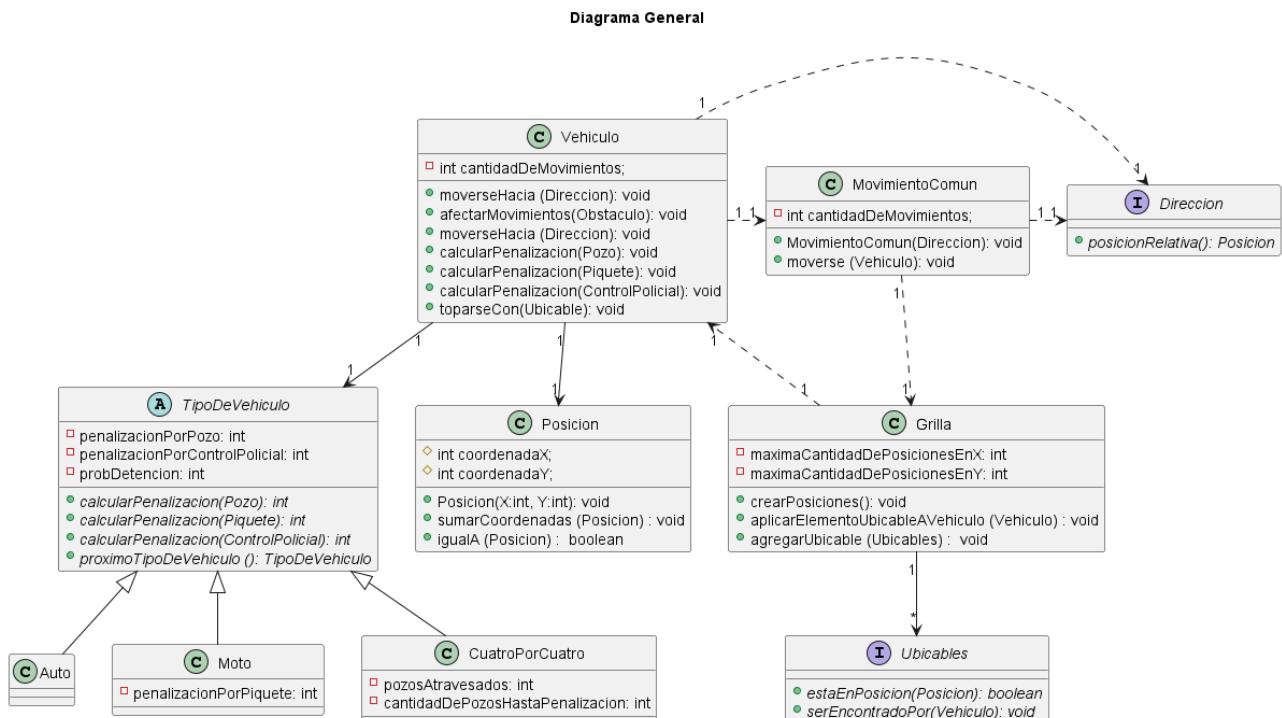


Figura 11. Diagrama De Clases - Vehículo

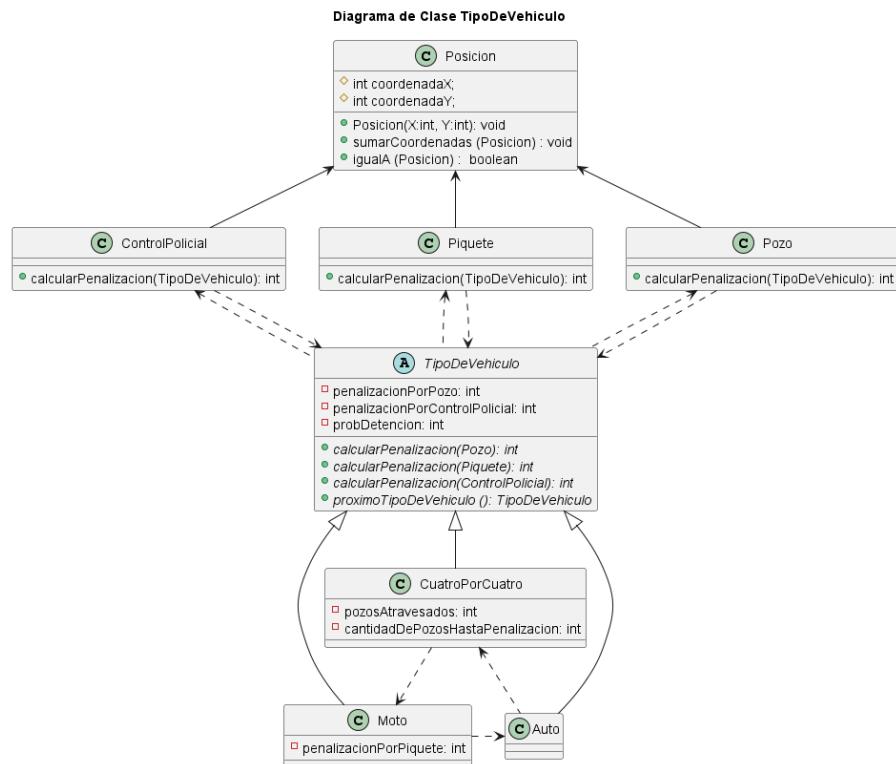


Figura 12. Diagrama De Clases - Tipo de Vehículo

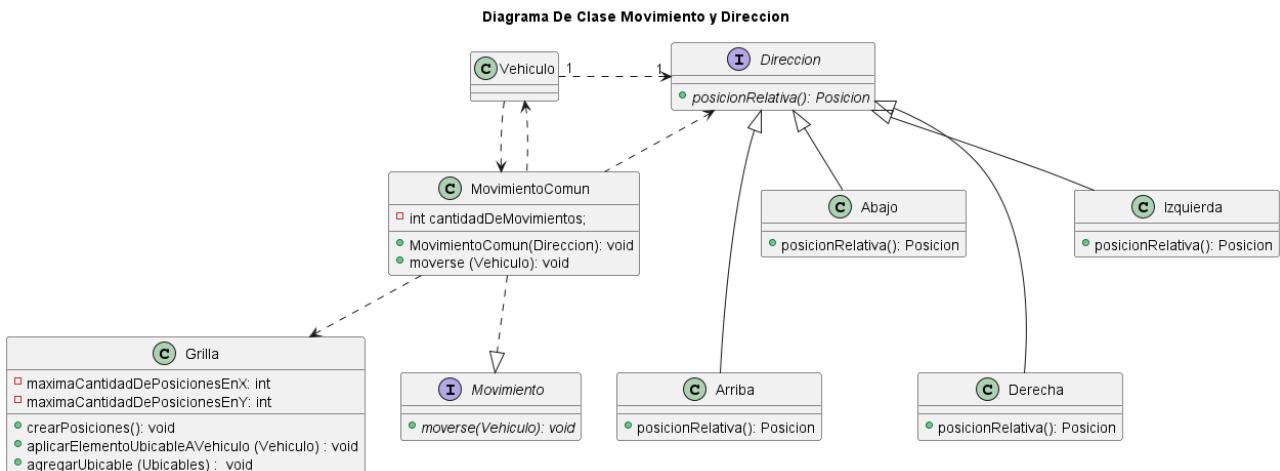


Figura 13. Diagrama De Clases - Movimiento y Dirección

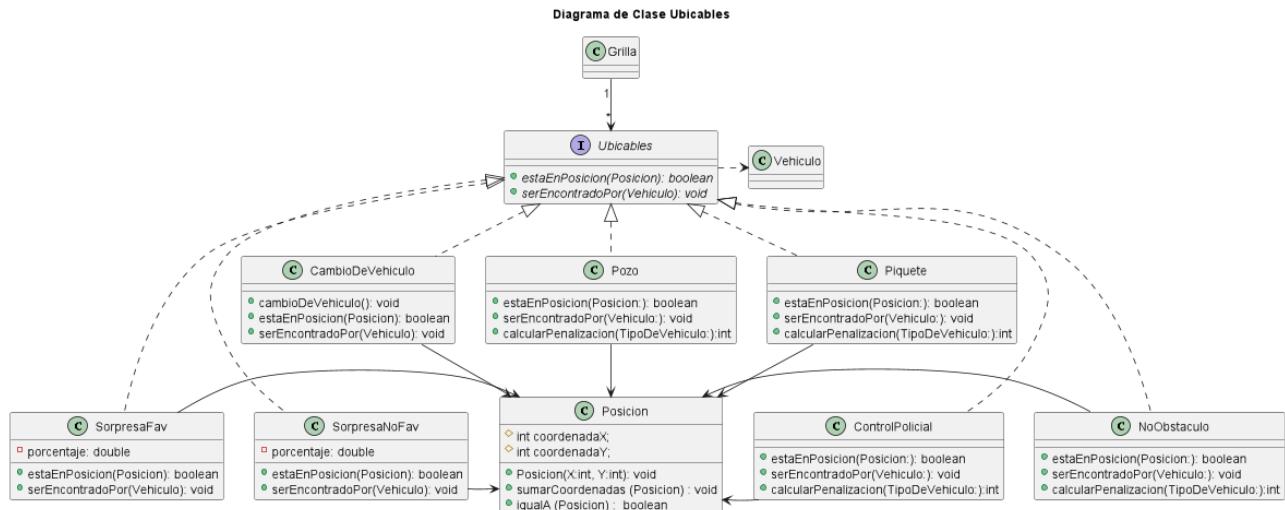


Figura 14. Diagrama De Clases - Ubicable

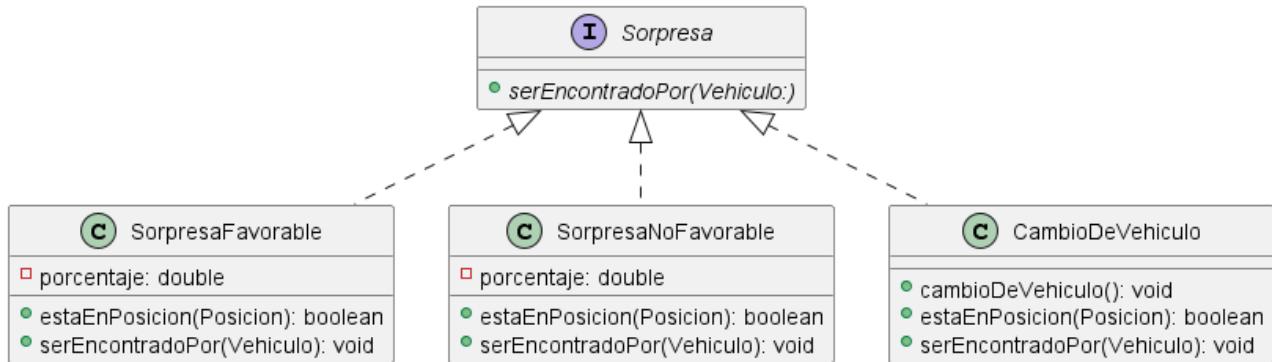


Figura 15. Diagrama De Clases - Sorpresa

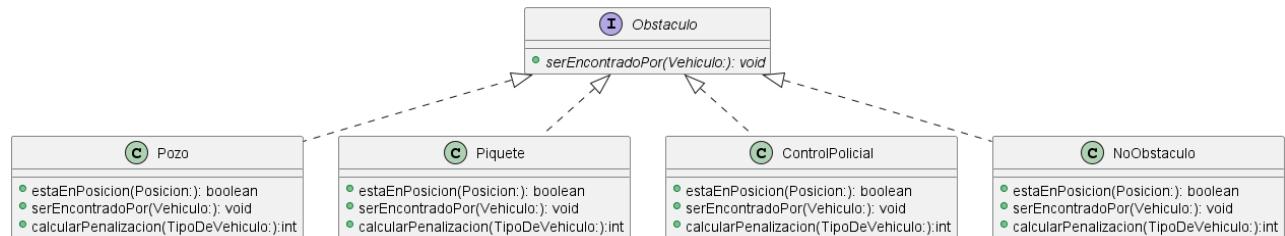


Figura 16. Diagrama De Clases - Obstaculo

3.8.3. Diagramas de secuencia

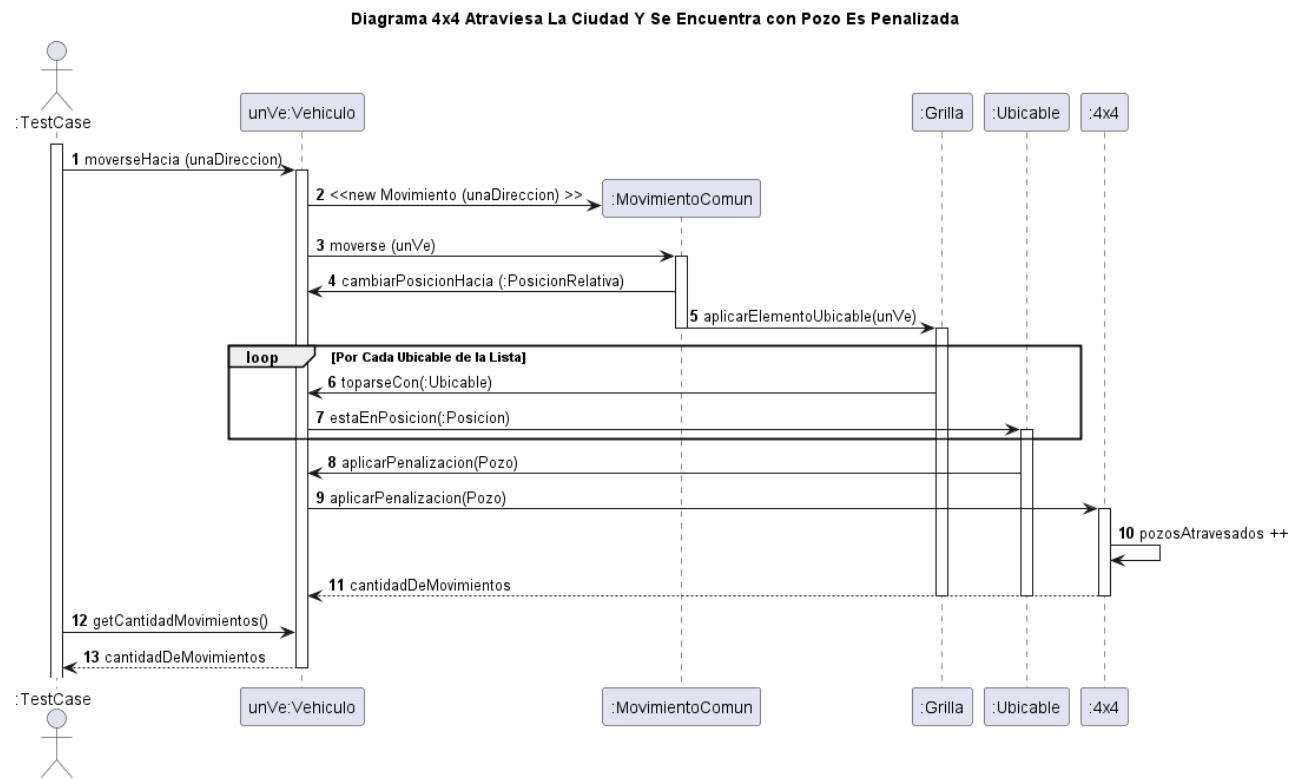


Figura 17. Diagrama De Secuencias - Una 4x4 atraviesa 3 pozos y es penalizada con 3 movimientos

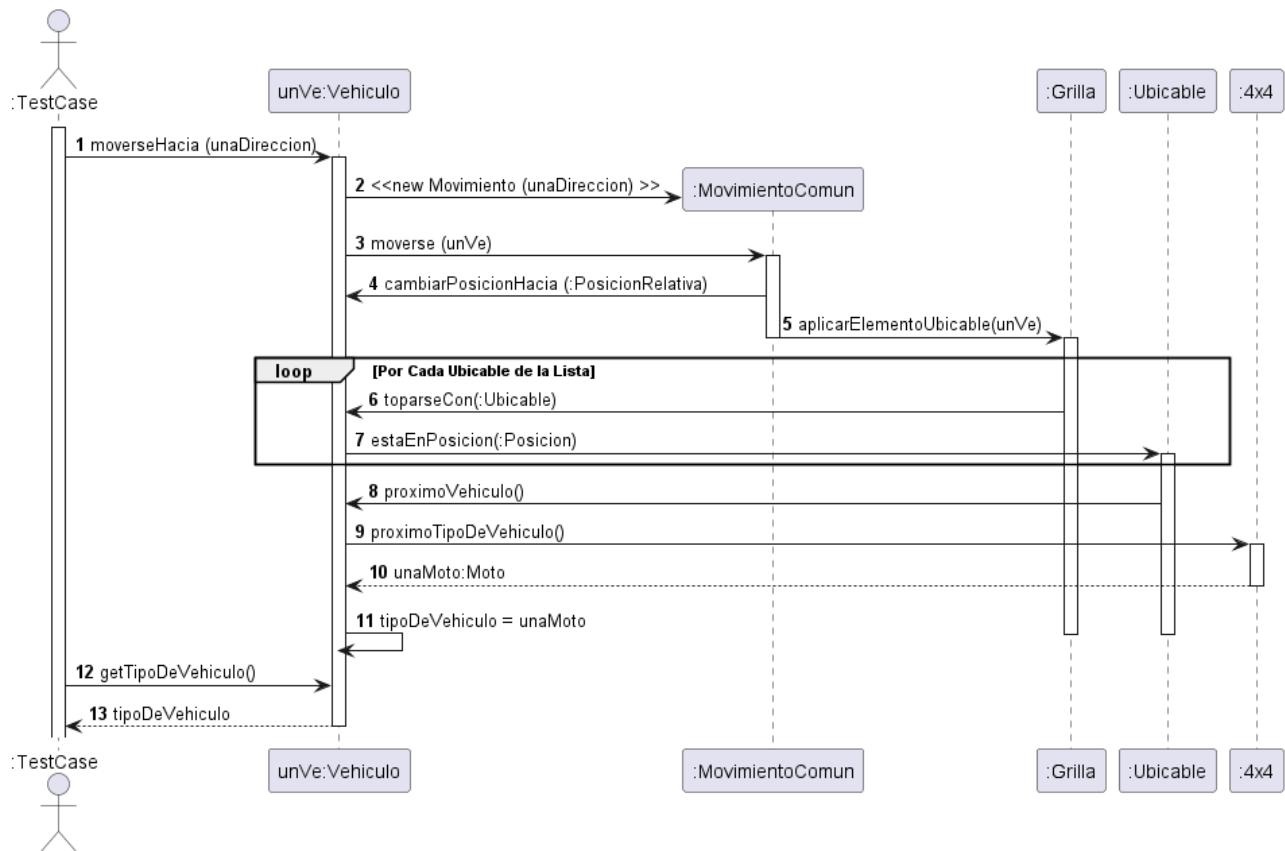
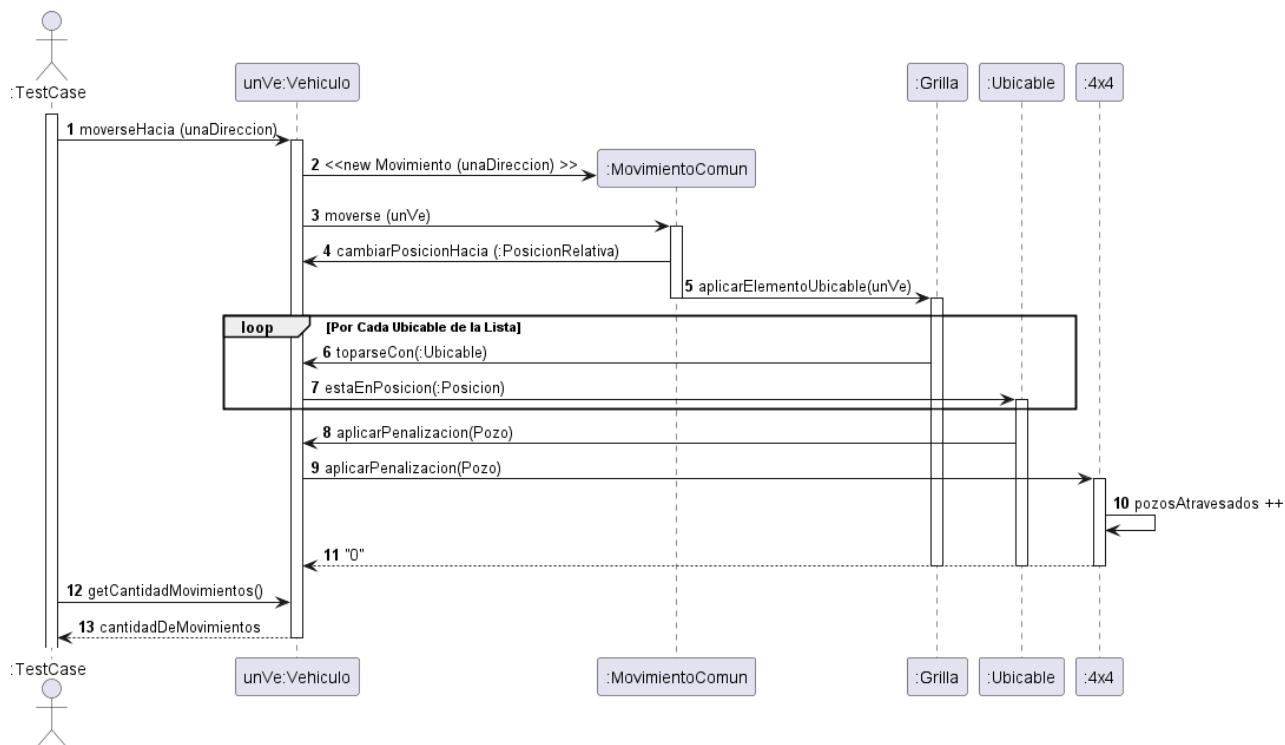
Diagrama 4x4 Atraviesa La Ciudad Y Se Encuentra con Sorpresa Cambio De Vehiculo**Figura 18.** Diagrama De Secuencias - Una 4x4 atraviesa la ciudad y encuentra una sorpresa de cambio de vehículo**Diagrama 4x4 Atraviesa La Ciudad Y Se Encuentra con Pozo No Es Penalizada****Figura 19.** Diagrama De Secuencias - Una 4x4 atraviesa la ciudad y se encuentra con un pozo no es penalizada

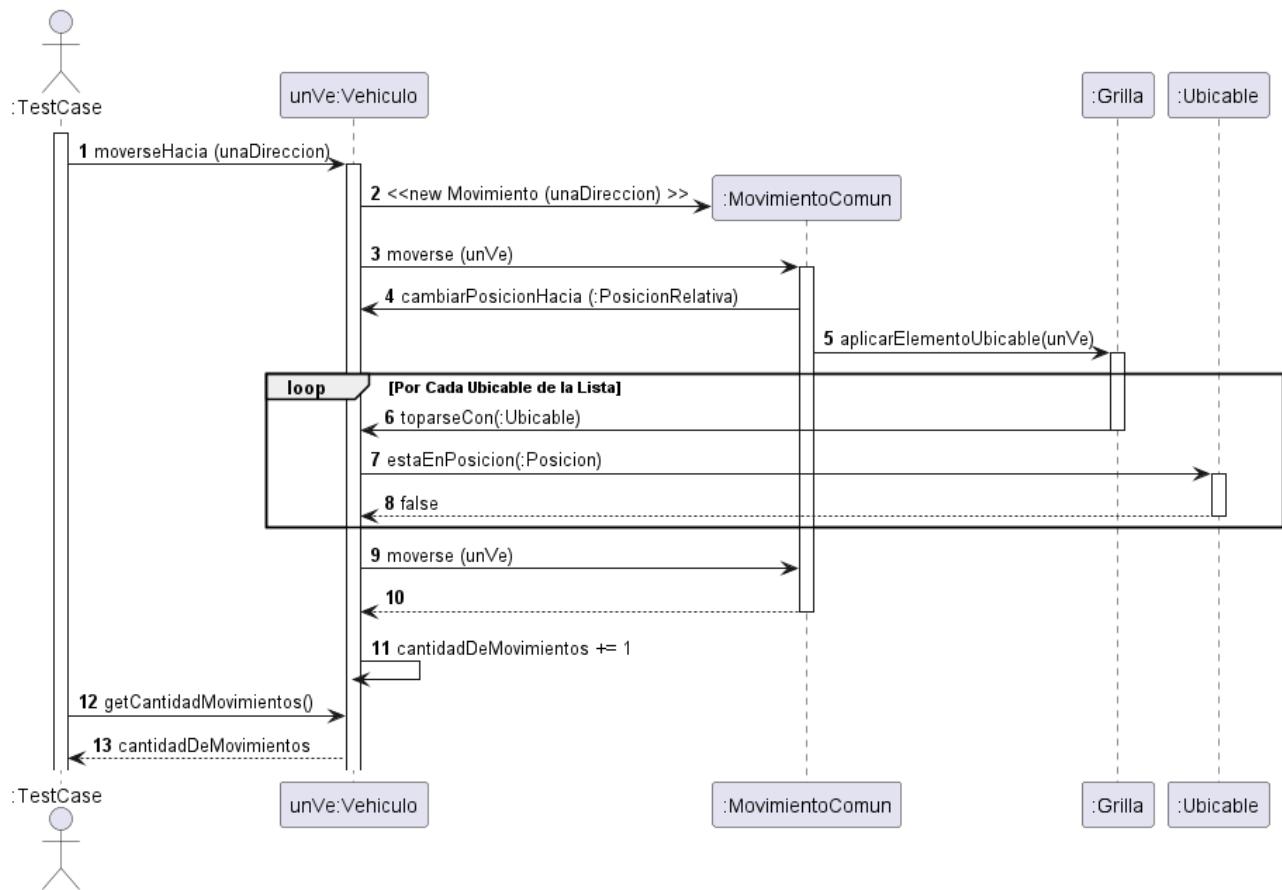
Diagrama Moto Atraviesa La Ciudad Y Sin Obstaculos La Cantidad De Movimientos es X**Figura 20.** Diagrama De Secuencias - una moto atraviesa la ciudad sin obstáculos y la cantidad de movimientos es X

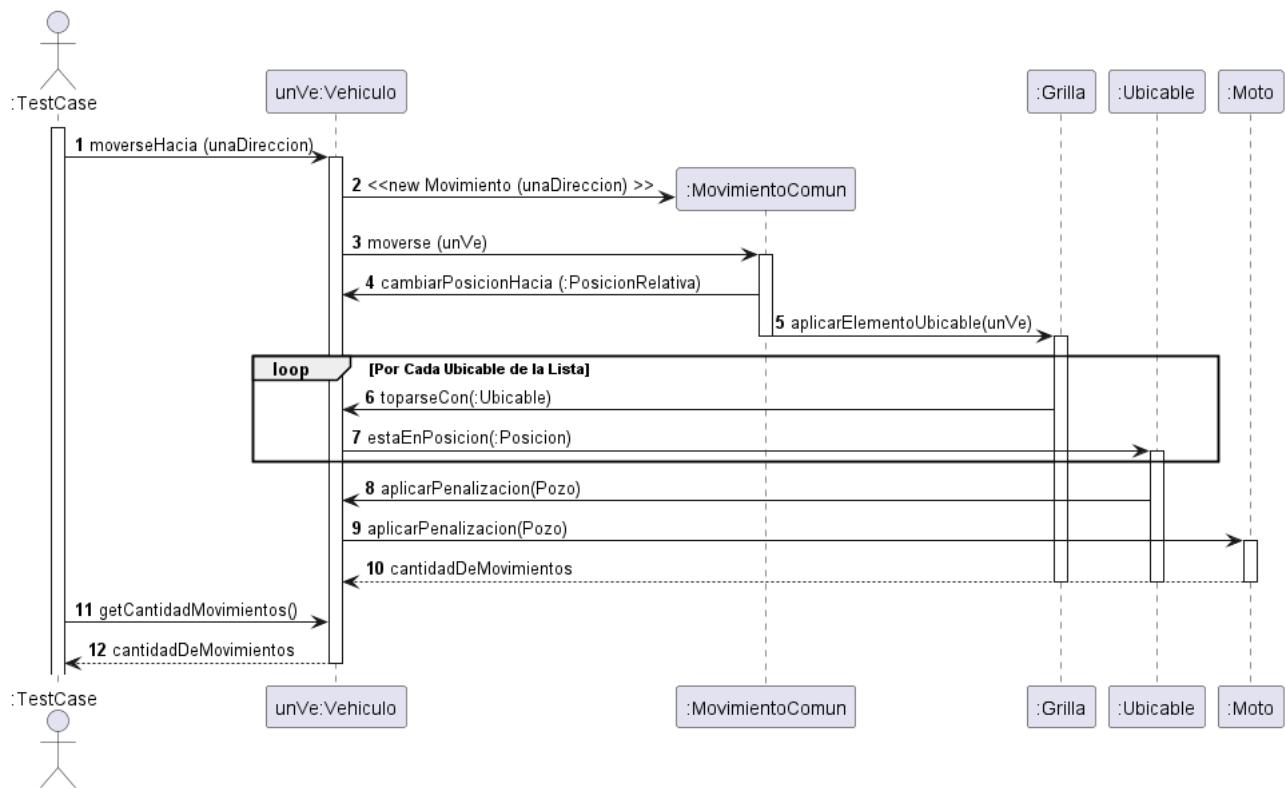
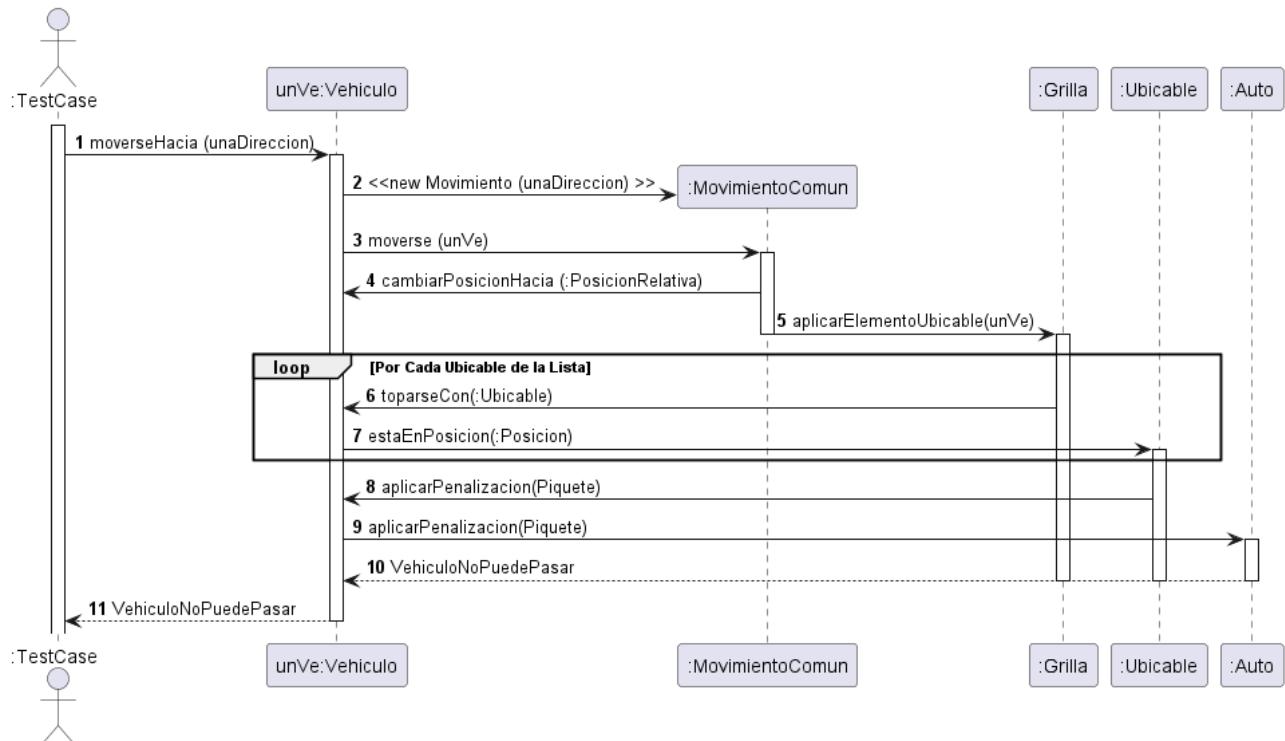
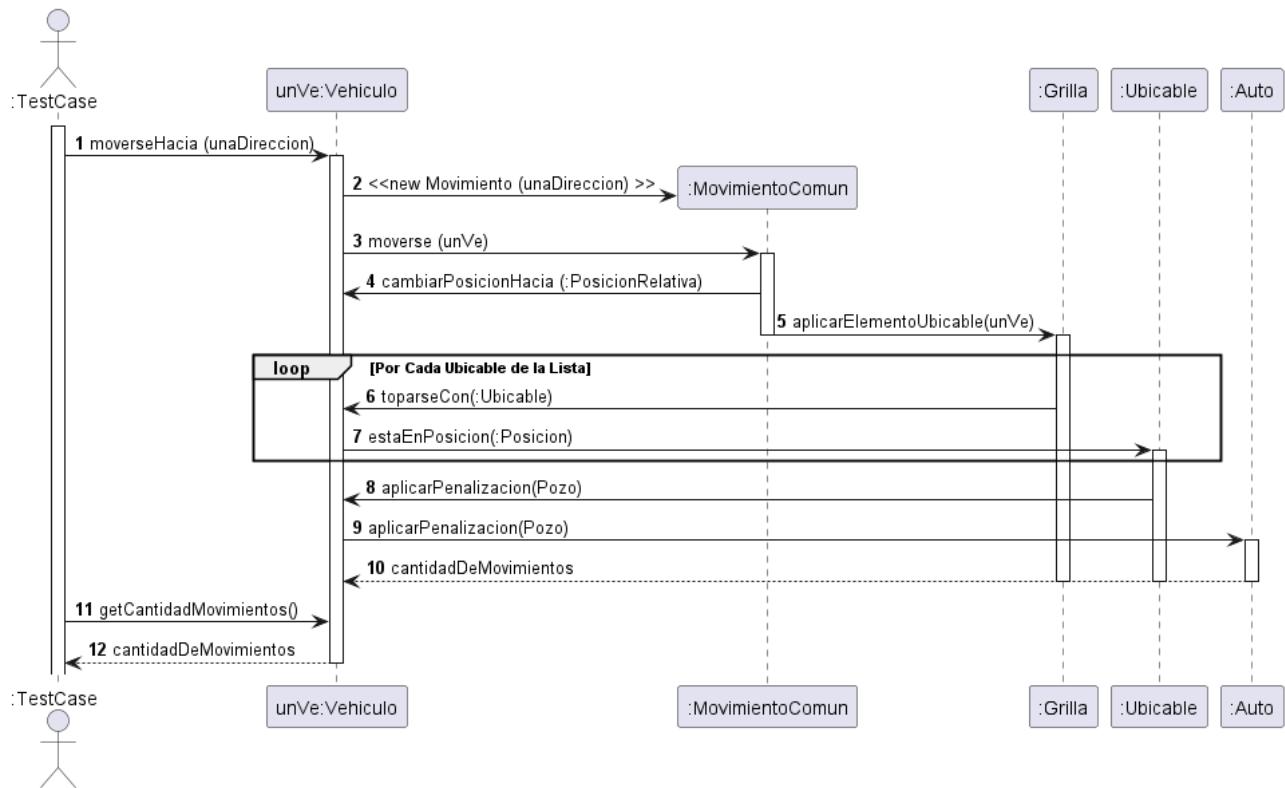
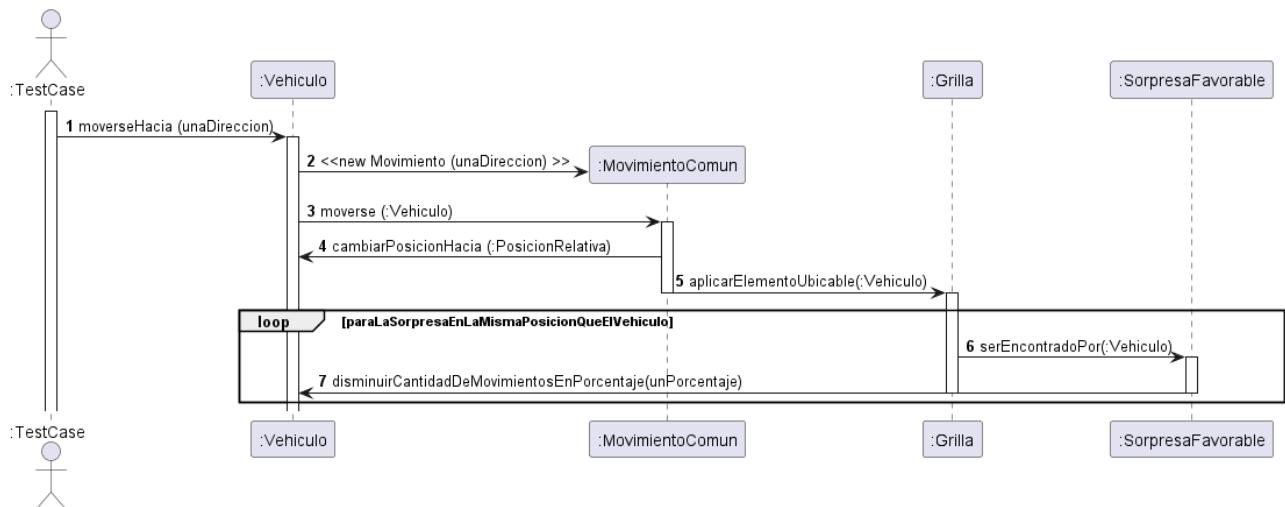
Diagrama Auto Atraviesa La Ciudad Y Se Encuentra con Pozo Penalizada En Tres Movimientos**Figura 21.** Diagrama De Secuencia - una moto atraviesa la ciudad y se encuentra con un pozo es penalizada en tres movimientos**Diagrama Auto Atraviesa La Ciudad Y Se Encuentra con Piquete No Puede Pasar En Tres Movimientos****Figura 22.** Diagrama De Secuencia - un Auto atraviesa la ciudad y se encuentra con un piquete no puede pasar

Diagrama Auto Atraviesa La Ciudad Y Se Encuentra con Pozo Penalizada En Tres Movimientos**Figura 23.** Diagrama De Secuencia - Un Auto Atraviesa la ciudad y se encuentra con un pozo es penalizada en tres movimientos**Diagrama Vehiculo Se Encuentra Con Sorpresa Favorable Que Esta En La Grilla****Figura 24.** Diagrama De Secuencia - Un vehículo se encuentra con una sorpresa favorable

3.9. Testeo del código

Se realizaron Test unitarios para cada una de las clases del Trabajo Practico, con el fin de aislar a las clases y probar que las instancias tengan el comportamiento esperado, esto es que cumplan con el contrato.

En el caso del uso no esperado/no deseado de una clase también se realizan pruebas para estos casos y se verifican que se lancen las excepciones correspondientes.

4. Conclusiones

La separación mediante el patrón MVC ayudó a desacoplar la vista del modelo y se permitió que el código sea mucho más organizado. Esto permitió que se termine primero el modelo y luego se haya continuado con la vista.

Los patrones de diseño han ayudado a resolver varios problemas durante el trabajo práctico, se pudo comprobar su importancia y necesidad ya que ayudan a que se cumplan los principios de POO y SOLID.

Se comenzó a trabajar con TDD lo permitió implementar el software realizando incrementos pequeños en la complejidad del código, con el fin de no introducir bugs y además de permitir organizar el trabajo. La refactorización para realizar mejoras del código sin cambiar contrato/funcionalidad resultó fundamental.

5. Bibliografía

"Growing object-oriented software, guided by tests " - Freeman Pryce - 2011

Programación orientada a objetos y técnicas avanzadas de programación - Carlos Fontela - 2003