FIUBA – 75.07
Algoritmos y programación 3
Trabajo práctico 2: GPS Challenge
2do cuatrimestre, 2013
(Trabajo grupal)

Integrantes del **grupo 3**:

Nombre	**Padrón**	**Mail**
Nikita Dyatlov	94441	nexodrone@gmail.com
Manuel Iglesias	93774	m.iglesias91@gmail.com
Jorge Cabrera	93310	cabrerajjorge@gmail.com
Cristian Gonzalez	94719	cristian3629@gmail.com

Fecha de entrega final: **12 dic. 2013**

Tutor: **Pablo Matías Rodríguez Massuh**

Nota Final:

Tabla de contenidos

Introducción

Objetivo del trabajo

Aplicar los conceptos enseñados en la materia a la resolución de un problema, trabajando en forma grupal y utilizando un lenguaje de tipado estático (Java).

Consigna general

Desarrollar la aplicación completa, incluyendo el modelo de clases e interface gráfica. La aplicación deberá ser acompañada por pruebas unitarias e integrales y documentación de diseño. En la siguiente sección se describe la aplicación a desarrollar.

Descripción de la aplicación a desarrollar

Contexto

La empresa Algo Ritmos SA dedicada al desarrollo de video juegos ha decidido contratar a un grupo de programadores para implementar el juego GPS Challenge.

Objetivo del juego

GPS es 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.

Dinámica del juego

El juego se jugará por turnos, y en cada turno el usuario decide hacia cuál de las 4 esquinas posibles avanzará.

Vehículos

El jugador podrá optar por tres diferentes tipos de vehículos.

- moto
- auto
- 4x4

Obstáculos

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, pero no afecta a las 4x4.
- Piquete: Autos y 4x4 deben pegar la vuelta, no pueden pasar. Las motos puede 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 ya que nunca llevan el casco puesto.

Sorpresas

También se podrán encontrar diferentes tipos de sorpresas:

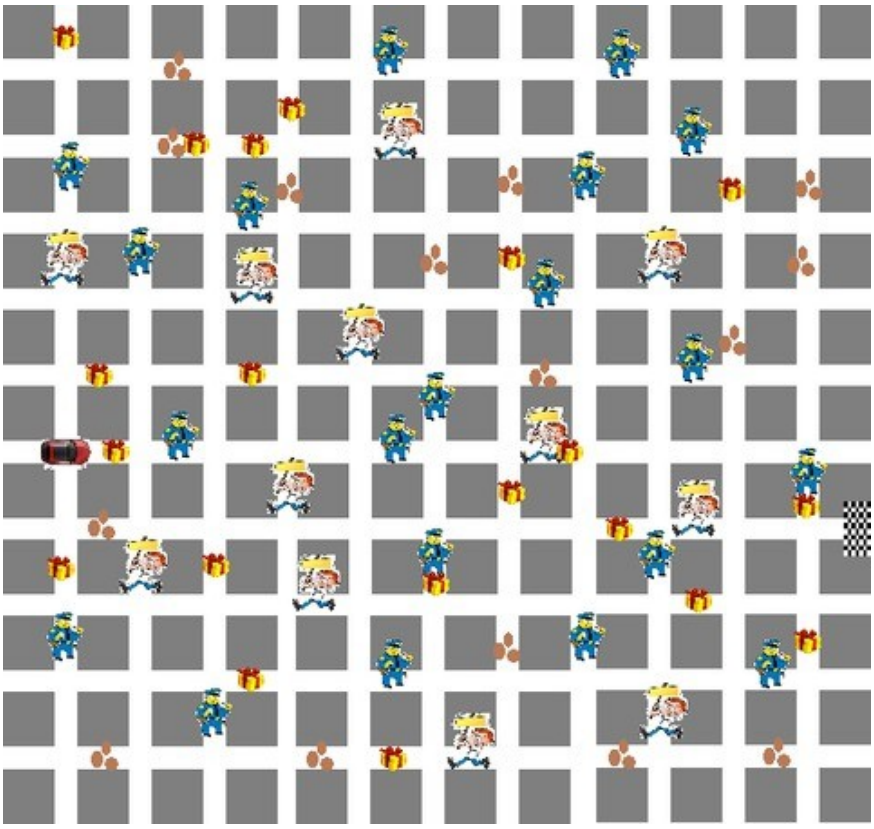
- Sorpresa Favorable: Resta el 20% de los movimientos hechos.
- Sorpresa Desfavorable: Suma el 25% de los movimientos hechos.
- Sorpresa Cambio de Vehículo: Cambia el vehículo del jugador. Si es una moto, la convierte en auto. Si es un auto lo convierte en 4x4. Si es una 4x4 la convierte en moto.

Las sorpresas figuraran en el mapa como un regalo y no se sabrá que es hasta que el vehículo la accione.

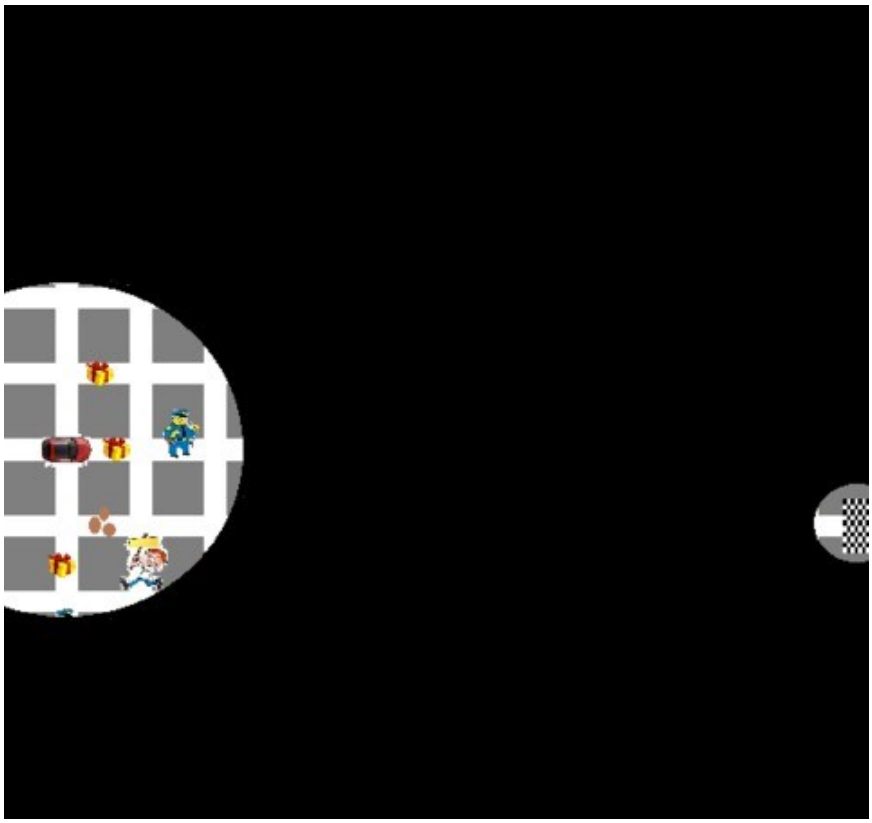
Escenario

Para hacerlo más interesante y jugable, el jugador no podrá ver más que dos manzanas a la redonda de la posición de su vehículo y la bandera a cuadros que marca la meta. El resto del mapa permanecerá en sombras.

El tamaño del escenario no será fijo, y tendrá un punto de partida y una meta.



Ejemplo de escenario



Ejemplo de escenario como lo visualiza el jugador

Puntajes altos

Se debe almacenar un ranking donde figuren los mejores puntajes asociados a un nickname que indique el usuario.

Entregables

- Código fuente de la aplicación completa, incluyendo también: código de la pruebas, archivos de recursos
- Script para compilación y ejecución (ant)
- Informe, acorde a lo especificado en este documento

Forma de entrega

A coordinar con el docente asignado.

Fechas de entrega

Se deberá validar semanalmente con el docente asignado el avance del trabajo. El docente podrá solicitar ítems específicos a entregar en cada revisión semanal.
La entrega final deberá ser en la semana del 13 de diciembre, en la fecha del curso en que se está inscripto.

Extensión del enunciado

Consigna general

Agregar al trabajo práctico la interfaz gráfica descripta en el presente enunciado, como así también los conceptos de persistencia explicados en clase. Cabe destacar que las siguientes imágenes son a modo orientativo de cómo organizar la interfaz gráfica y qué pantallas tienen que estar. Nada aclara y queda a libre criterio de los alumnos, la elección de combinación de colores a utilizar, imágenes gráficas para los objetos, imágenes de fondo, texturas, sonidos, etc.

Niveles y Puntajes

El juego deberá poder jugarse en hasta 3 niveles:

- Fácil
- Moderado
- Difícil

Siendo distintos la configuración de los mapas y cantidad de obstáculos y sorpresas en cada uno. Cada nivel tendrá una cantidad de movimientos límite para ser ganado. Por ejemplo, si para el nivel fácil se necesitan no más de 80 movimientos y el jugador llega a la meta en 50 entonces obtendrá 30 puntos por haber ganado ese nivel de esa forma.

- Se contabilizará 1 punto por cada movimiento "sobrante" en fácil.
- Se contabilizarán 2 puntos por cada movimiento "sobrante" en moderado.
- Se contabilizarán 3 puntos por cada movimiento "sobrante" en Difícil.

La configuración de los mapas, los obstáculos y las sorpresas deberán ser extraídos de archivos xml. No así, la posición de inicio del jugador y la meta, que serán aleatorias.

Persistencia

Se almacenará tanto los jugadores, sus puntajes, y la última partida guardada de cada uno.

Interfaz gráfica mínima.

Al iniciar la aplicación la misma debería preguntarnos si somos un jugador existente o vamos a jugar por primera vez.

Si soy nuevo:

Si ya existo:

Luego veremos:

Si elegimos puntajes:

Pablo S.	70
Pablo M.	60
Nico	30
Gabriel	10
Diego	0

○	Fácil	○	Moto
○	Moderado	○	4x4
◉	Dificil	◉	Auto

Jugar

Hola Nico

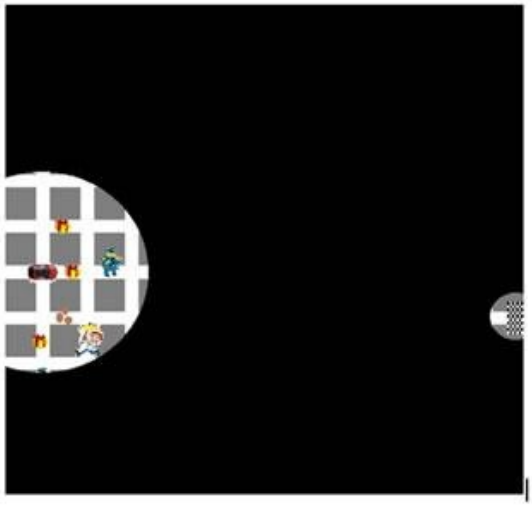
Partida Difícil

Volver

Mov Límites: 35

Mov actuales: 1

Vehículo actual: Auto



Puntaje: 102

Guardar

Hola Nico

Partida Difícil

Volver

¡ Ganaste !

Te sobraron 8 Movimientos

Obtuviste un puntaje de 24 puntos

Jugar otra vez

¡ Perdiste !

Jugar otra vez

Informe

Supuestos

- Al pasar por una calle con un obstáculo y una sorpresa, el vehículo primero interactúa con el obstáculo, y luego con la sorpresa.
- Pasar por calle sin obstáculos demora un movimiento. Sí la calle tiene un obstáculo y una sorpresa, primero se aplica la penalización del obstáculo (si hay), luego el porcentaje de la sorpresa (si hay), y se suma 1 movimiento.
- El redondeo de movimientos según el porcentaje es al número entero más próximo.
- La posición ganadora siempre va a estar en el tablero.
- En cada calle hay como máximo un obstáculo y/o una sorpresa.
- Las sorpresas se activan una única vez, al pasar por ésta.

Modelo de dominio

Para el desarrollo del Juego, decidimos que el tablero sea una matriz de bocacalles, en donde cada bocacalle guarda una referencia a las cuatro calles posibles: norte, sur, este y oeste. Las bocacalles vecinas tendrán calles compartidas. Así mismo las calles contendrán las sorpresas y los obstáculos, que se le aplicarán al coche.

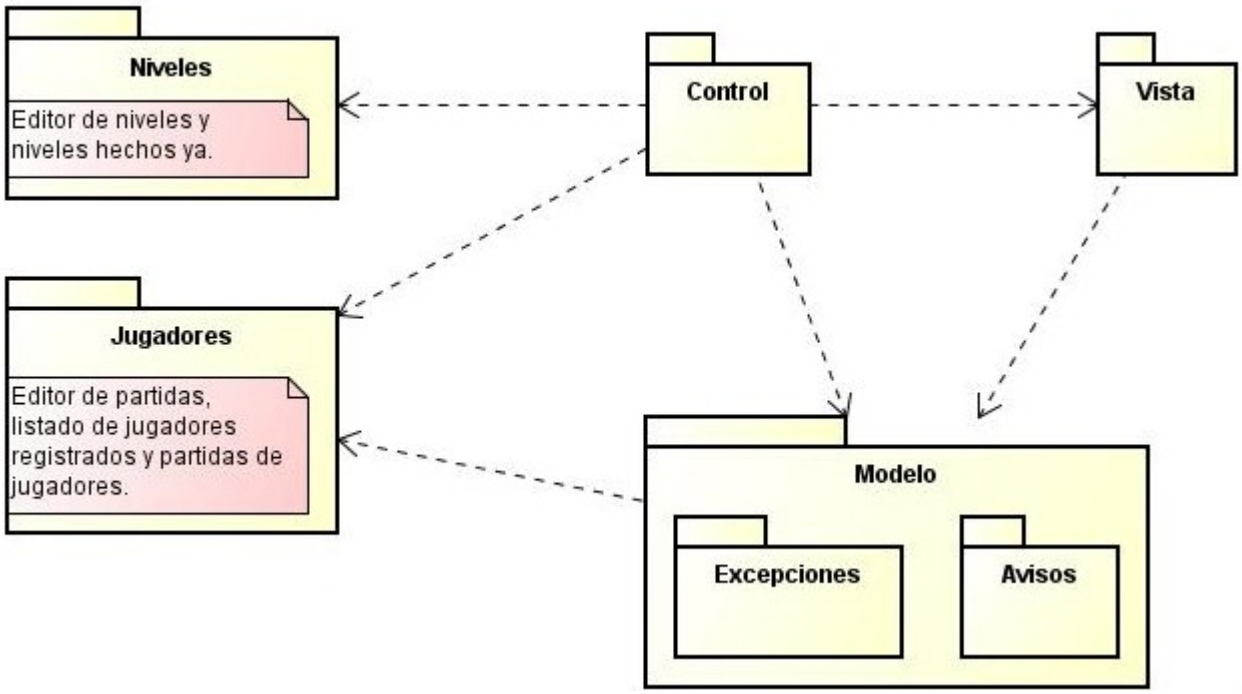
Con respecto al movimiento del vehículo en el tablero, como éste es una matriz de bocacalles, el vehículo se mueve de bocacalle a bocacalle.

Viendo que cuando se empieza a correr la aplicación siempre hay una instancia de Juego y lo variante son las Partidas, teniendo en cuenta esto se decidió diseñar el Juego con el patrón de diseño Singleton, el cual te permite acceder a la única instancia disponible desde cualquier clase.

Esto también se aplica en Observador, cuya única responsabilidad es notificar al Juego que hubo un cambio de Vehiculo, y el que notifica al Observador, puede ser cualquiera de las instancias de SorpresaCambioDeVehiculo, siendo no necesaria mas de una instancia de esta clase.

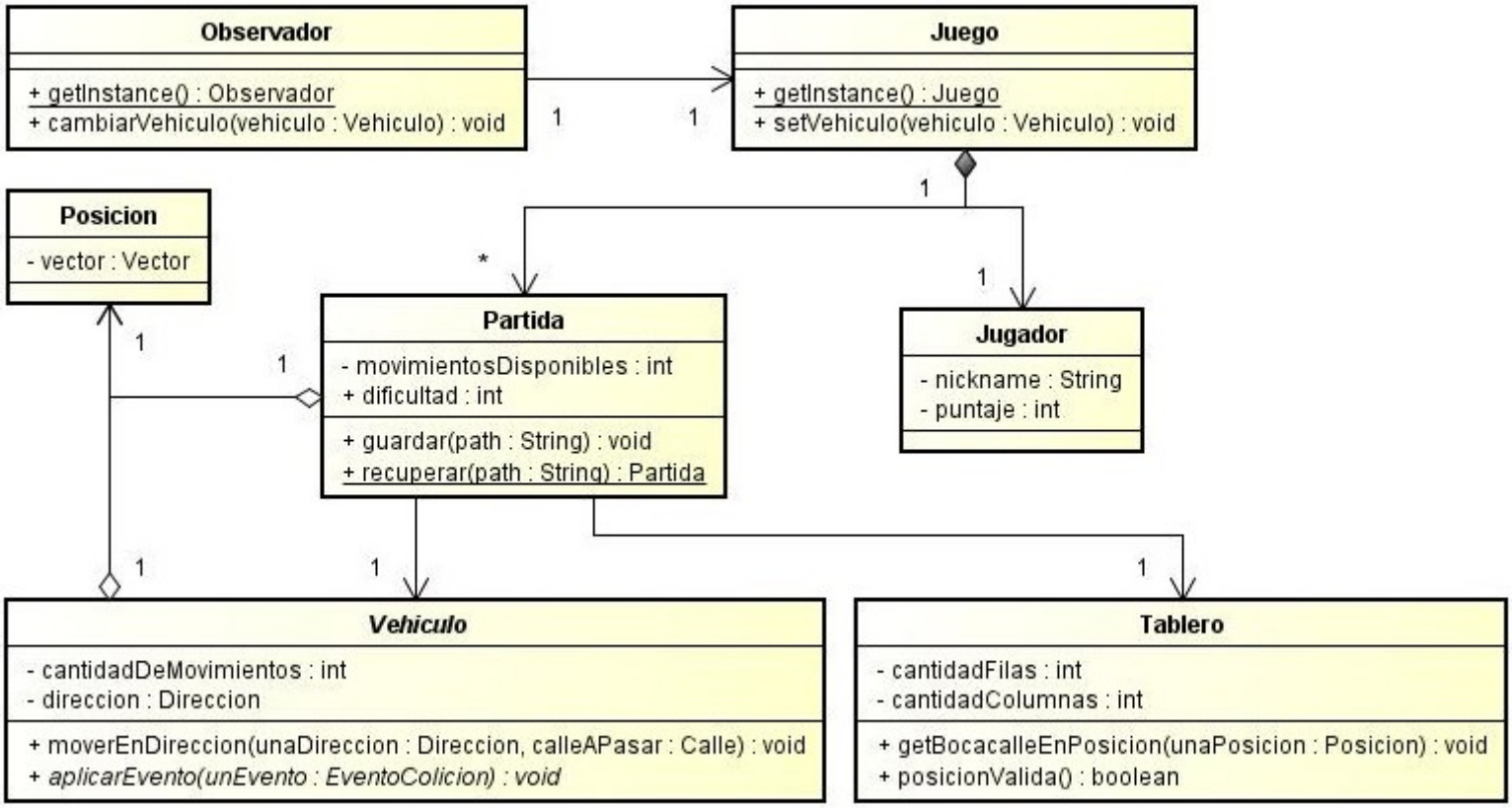
Para resolver el problema de interacción Vehiculo-Sorpresa y Vehiculo-Obstaculo, se decidió implementar el patrón de diseño Doble Dispatch , y para volver aun mas genérica la solución, se obliga a todos los objetos que interactúen con Vehiculo, implementen la interfaz EventoColicion.

Diagrama de paquetes

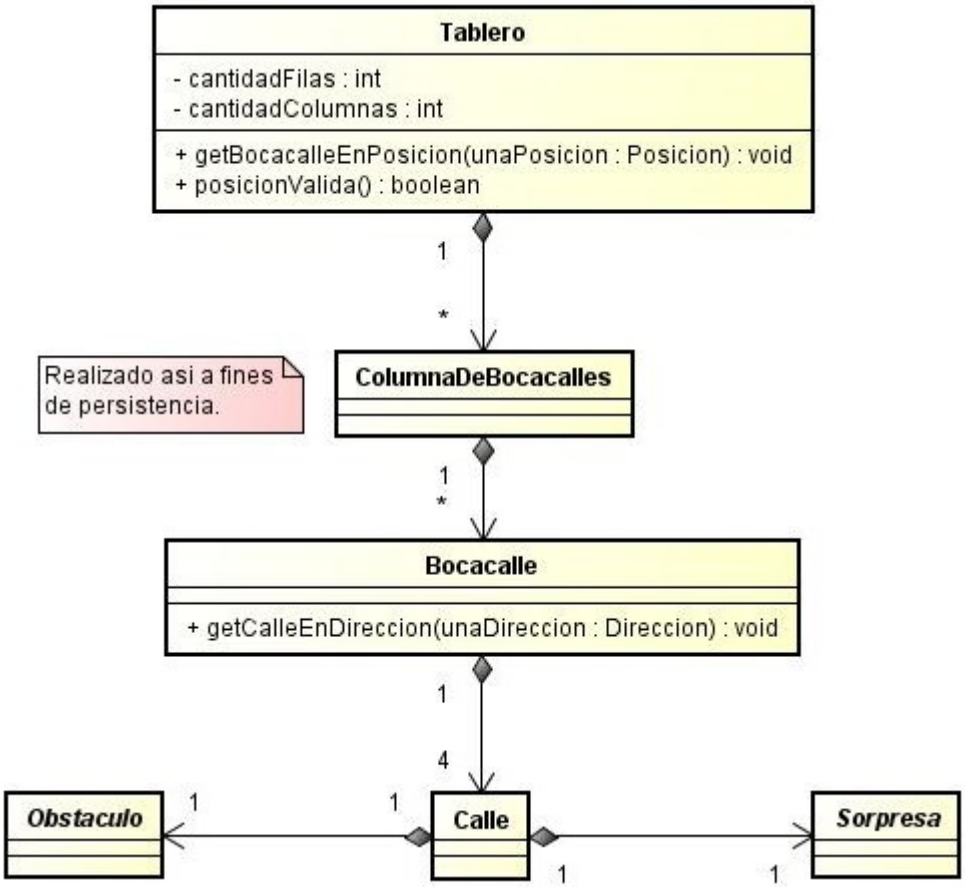


Diagramas de clases

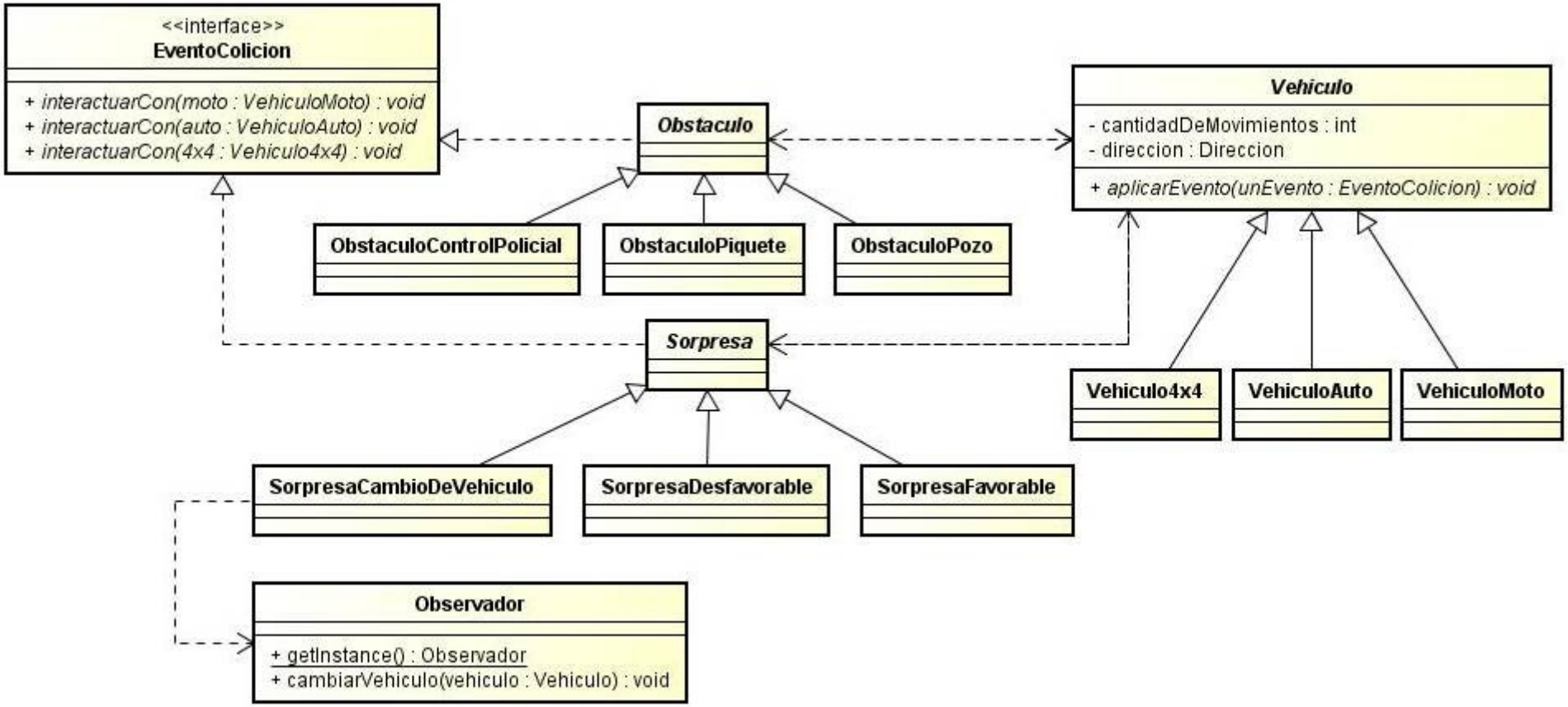
Juego



Tablero y sus componentes



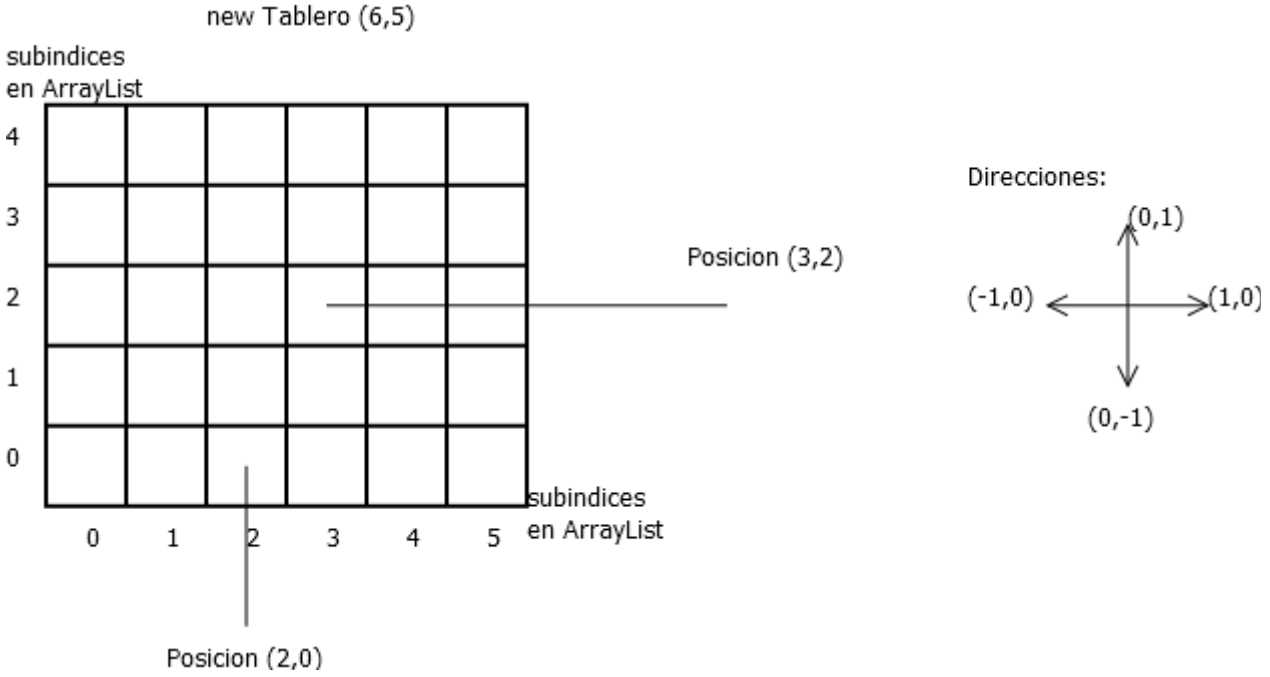
Vehiculo


<<interface>>
EventoColicion
+ interactuarCon(moto : VehiculoMoto) : void
+ interactuarCon(auto : VehiculoAuto) : void
+ interactuarCon(4x4 : Vehiculo4x4) : void
Obstaculo
Vehiculo
- cantidadDeMovimientos : int
- direccion : Direccion
+ aplicarEvento(unEvento : EventoColicion) : void
ObstaculoControlPolicial
ObstaculoPiquete
ObstaculoPozo
Sorpresa
Vehiculo4x4
VehiculoAuto
VehiculoMoto
SorpresaCambioDeVehiculo
SorpresaDesfavorable
SorpresaFavorable
Observador
+ getInstance() : Observador
+ cambiarVehiculo(vehiculo : Vehiculo) : void

Detalles de implementación

Movimientos

En primer lugar como el Tablero fue pensado como una matriz de Bocacalles, nos pareció apropiado poder tener una clase Vector que nos indique dirección y sentido de cómo vamos a movernos dentro del tablero. Para esto creamos las clases Dirección y Posición, que usan a la clase Vector.
La clase Posición es usada tanto por el Vehiculo para su posición actual, como por el Tablero para su posición ganadora. La clase Dirección es usada para indicar hacía donde se quiere mover al Vehículo. Para ello adoptamos la siguiente convención de signos:



Los índices comienzan desde cero, y el primer parámetro del constructor del tablero refiere a la cantidad de columnas mientras que el siguiente a la cantidad de filas.

No se consideraron válidas las posiciones cuyo número de columna o fila exceda el rango del tablero. Dirección siempre es en un sentido y unitaria, es decir, solo se puede desplazar en el eje x (columnas) o en el eje y (filas), y solo una calle por movimiento. Lo más habitual para nosotros resulto usar siguientes direcciones:

- Norte: (0,1)

- Sur: (0,-1)
- Este: (1,0)
- Oeste: (-1,0)

El movimiento que el jugador desee realizar no será válido si está en el límite del tablero, y la dirección de movimiento conduce a una posición que esta fuera del rango del tablero.

Eventos

Creamos una interfaz llamada "Evento", que tiene los métodos:

- interactuarCon(VehiculoMoto unaMoto)
- interactuarCon(VehiculoAuto unAuto)
- interactuarCon(Vehiculo4x4 una4x4)

Se pensó para que las clases Sorpresa y Obstáculo la implementen y sobrecarguen los métodos. Se pensó así debido a que tanto Sorpresa como Obstáculo actúan diferente según el tipo de vehículo con el que interactúen.

Para aplicar el evento correctamente, se hace doble dispatch, y simplemente cuando un coche intenta atravesar alguna calle con algún obstáculo u sorpresa le enviamos el mensaje intereactuarCon(this).

Persistencia

Para el manejo de la persistencia de utilizo la librería externa "SimpleXML serialization". El estado de los objetos a serializar se guarda en archivos .xml, por lo que estos estados se pueden modificar desde fuera de la aplicación.

Para el guardado y cargado de una Partida, se crearon los métodos "guardar" y "recuperar". El método "guardar" lo que hace es serializar al objeto que lo llamó, es decir a la Partida actual, persistiendo todos sus componentes (Vehículo, Tablero, Posición Ganadora y Cantidad de Movimientos). El método "recuperar" es un método de clase. Al llamarlo, se busca el archivo .xml de la partida guardad, se crea una nueva instancia de la clase Partida, y se la devuelve con el estado cargado. En ambos métodos se deben indicar los directorios donde se guardará o se buscará al archivo.

El listado de jugadores se guarda en un archivo, el cual contiene todos los datos del jugador (nombre y puntaje). Esta lista se recupera cuando se quiere empezar a jugar con un jugador ya registrado, y se muestran por pantalla para que se seleccione el jugador.

En caso de querer jugar con un jugador nuevo, el mismo es agregado inmediatamente al final de la lista, con su puntaje inicializado en 0 (cero).

Cuando un jugador ganada su partida, se recupera el puntaje del jugador, luego se le suma el puntaje obtenido y por último se lo vuelve a persistir en la lista de jugadores, con su puntaje actualizado.

Log de la partida

Para la creación del log de la partida, se implementó un logger (Logger.java) y una interfaz Listener. El logger contiene una lista de Listener, que serán las áreas de texto donde se escribirán todos los mensajes del log. En nuestro caso, esta lista contiene solamente a un objeto de la clase LogPartida, que implementa la interfaz Listener. Para agregar un mensaje en el log de la partida, se procede como cualquier logger: se pide la instancia del logger desde cualquier parte de la aplicación, y se agrega el mensaje.

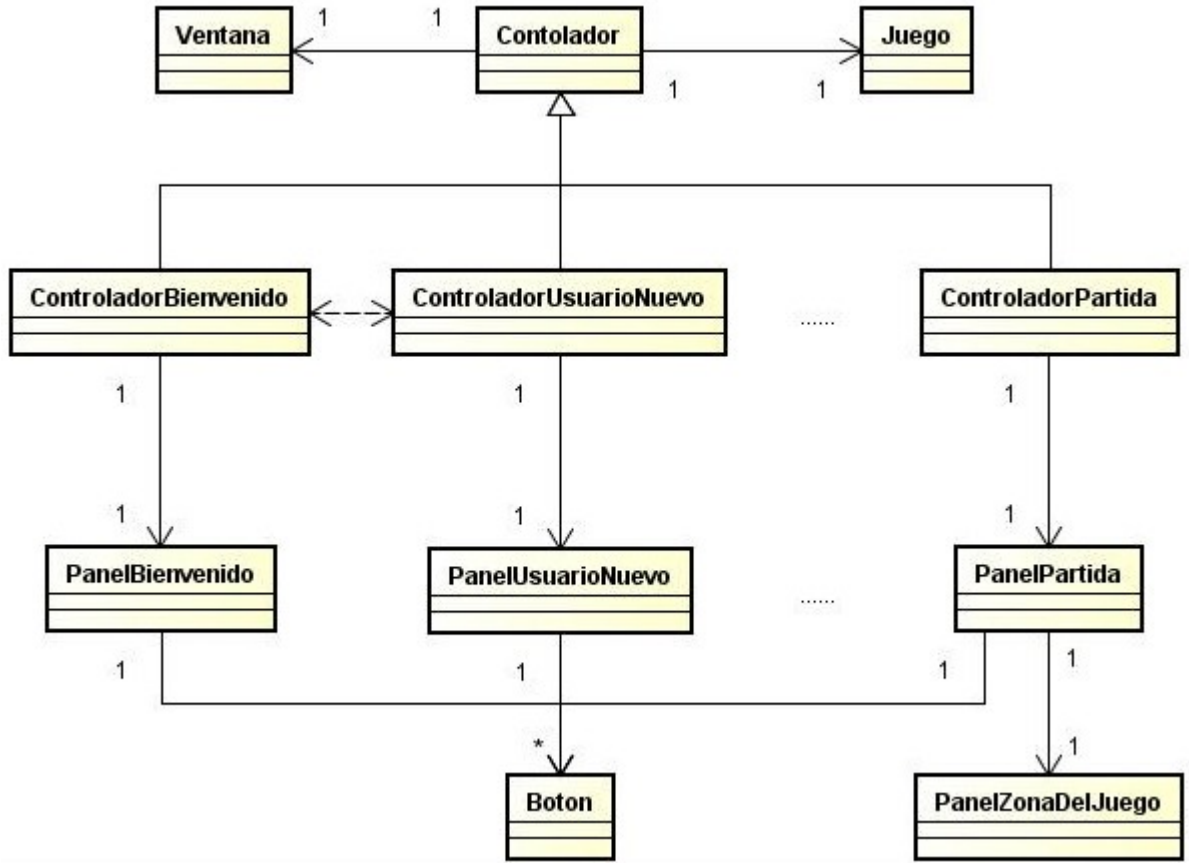
Pérdidas y ganancias

- Si después de jugada, la cantidad de movimientos sobrantes es negativa, la partida se considera perdida.
- Si después de jugada, la cantidad de movimientos sobrantes es igual a 0 y la posición resultante no es ganadora, la partida se considera perdida.
- Si después de jugada, la cantidad de movimientos sobrantes es igual a 0 y la posición resultante es ganadora, la partida se considera ganada, aunque puntaje será 0.
- Si después de jugada, la cantidad de movimientos sobrantes es positiva y la posición resultante es ganadora, la partida se considera ganada.
- Si después de jugada, la cantidad de movimientos sobrantes es positiva y la posición resultante no es ganadora, eso era una jugada común.

Vista y control

La parte visual de la aplicación consiste en una única ventana (JFrame) y paneles, que se controlan a través de sus respectivos controladores. Todos los controladores heredan de la clase Controlador y comparten el mismo juego y ventana (estáticos). Los paneles heredan directamente de JPanel, ya que no tienen nada en común. PanelZonaDelJuego es una particularidad, ya que está contenido en PanelPartida y representa visualización del tablero en tiempo de jugada. Entre la mayoría de los controladores hay una dependencia débil, causada por la posibilidad de navegación por menús del programa. La clase Botón es una adaptación de JButton para nuestra vista. Al correr aplicación, se

arranca con el PanelBienvenido. De allá el usuario puede llegar a los demás paneles, según los botones que elija.



Cada panel que ve el usuario está organizado por "layouts". En muchos casos, dentro del panel hay otros paneles con sus propios "layouts". Esa estructura anidada garantiza la alineación automática del contenido y un mayor orden.

Excepciones

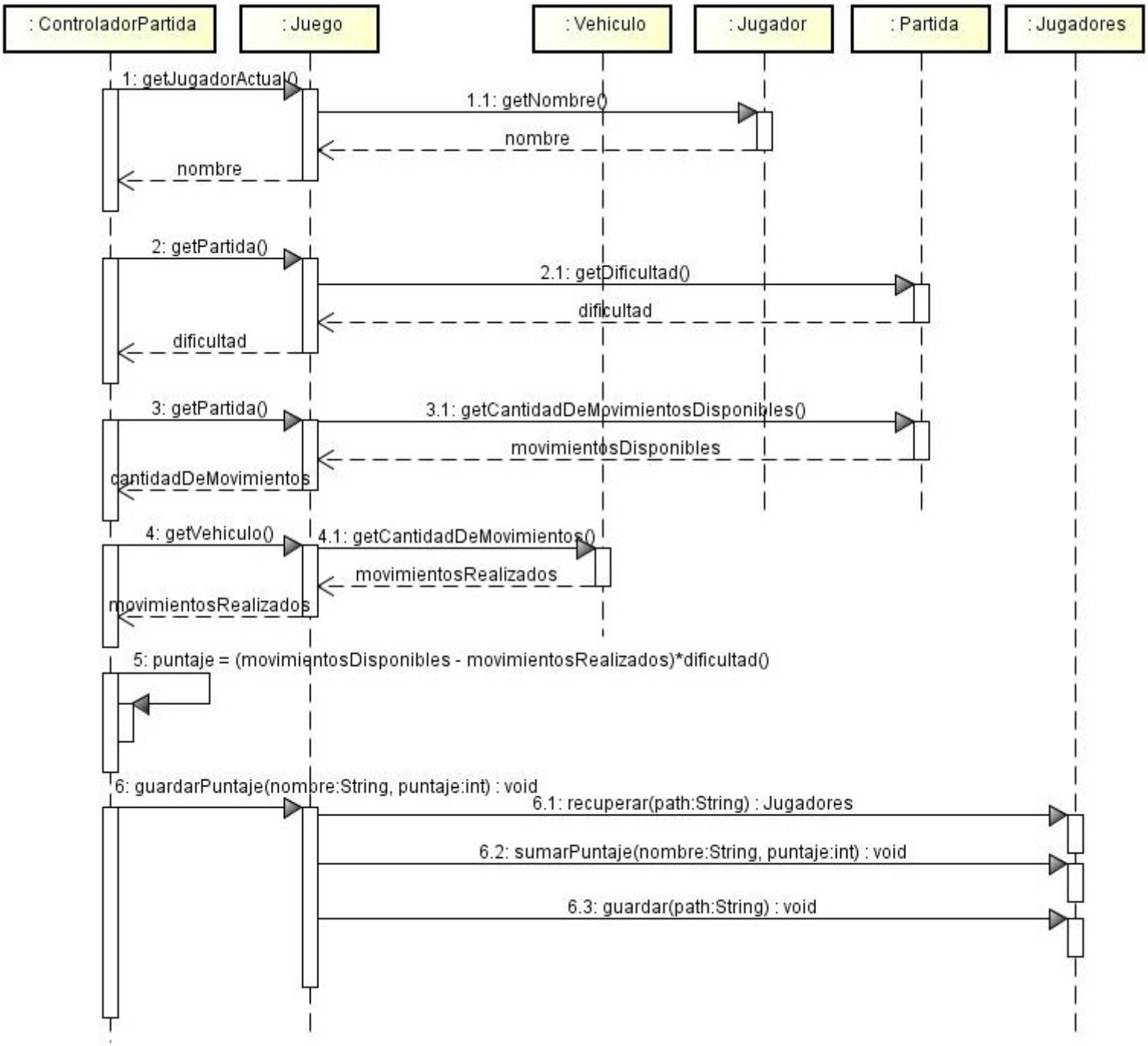
- Siempre que el jugador intente moverse hacía una posición inválida, se lanzara la excepción "MovimientoInvalido", lo que hará que se

vuelva a realizar la jugada, sin haber modificado el estado del juego. Se informará en el log de la partida.

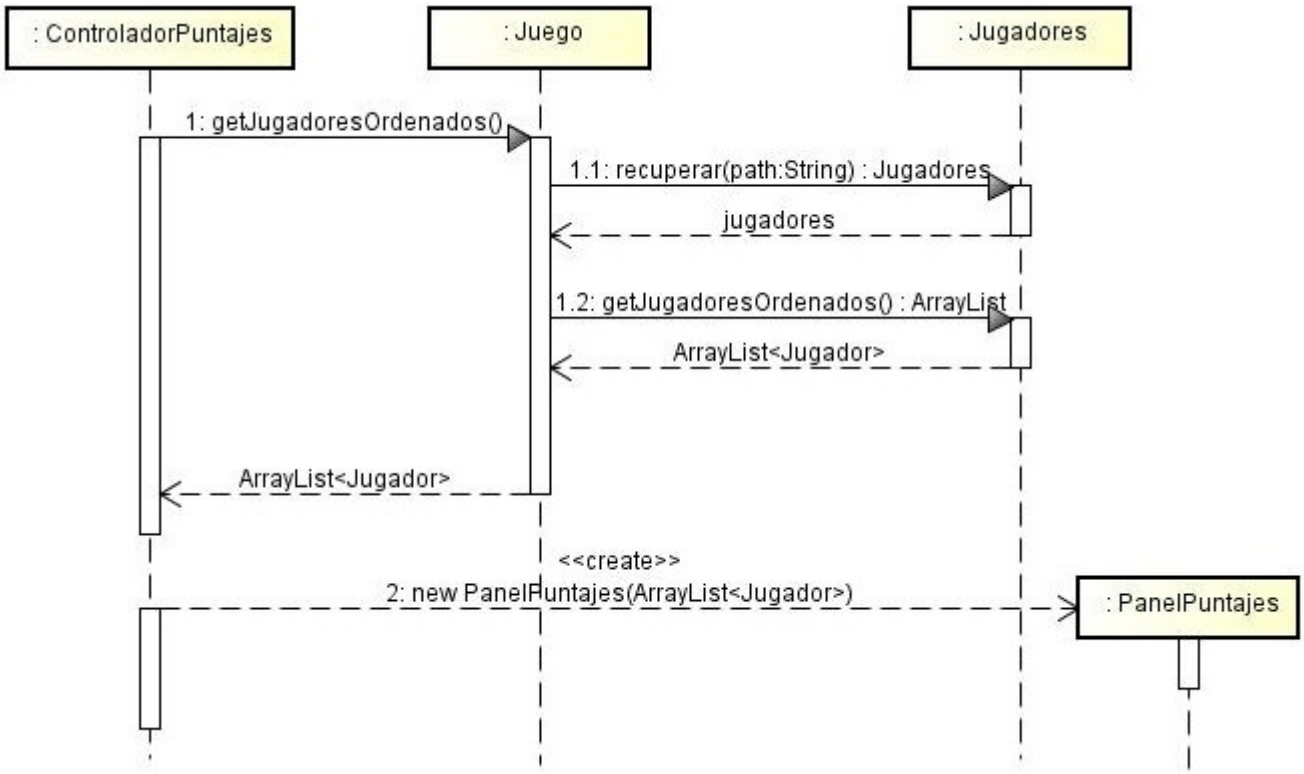
- Siempre que el jugador intente moverse por una calle donde hay un Piquete, se lanzara la excepción “CalleBloqueadaPorPiquete”, lo que hará que se vuelva a realizar la jugada, sin haber modificado el estado del juego. Se informará en el log de la partida.
- Cuando se intente cargar una partida que no este guardada en el directorio indicado, se lanzara la excepción “PartidaInexistente”. El jugador deberá ingresar nuevamente el directorio de la partida guardada.
- Cuando se quiera recuperar la lista de jugadores de un directorio inexistente, se lanza la excepción “ErrorArchivoJugadores”. Se le avisará al jugador que hubo un error con el archivo que contiene a la lista de jugadores, y se le pedirá que se registre con un usuario nuevo. Este nuevo usuario se almacenará en una nueva lista, en el directorio actual donde no se encontró el archivo.
- Cuando se quiera ver los puntajes, pero no haya ningún jugador registrado, se lanzará la excepción “NoHayUsuariosCreados”. Esto solo ocurrirá en caso de que el primer jugador del juego quiera registrarse con un usuario existente. Se le informará al jugador que no hay ningún jugador creado, y se le pedirá que cree uno.
- Cuando un jugador quiera crear un usuario con un nombre ya existente, se lanzará la excepción “UsuarioExistente”. Se le informará al jugador que ya existe un usuario con ese mismo nombre, y se le pedirá que se registre con otro nombre.
- En caso de que accidentalmente se modifique el nombre del jugador mientras se está jugando, se lanzará la excepción “UsuarioInexistente”. Se le informará al jugador que no se pudo encontrar al usuario, y que no se podrá guardar su puntaje.

Diagramas de secuencia

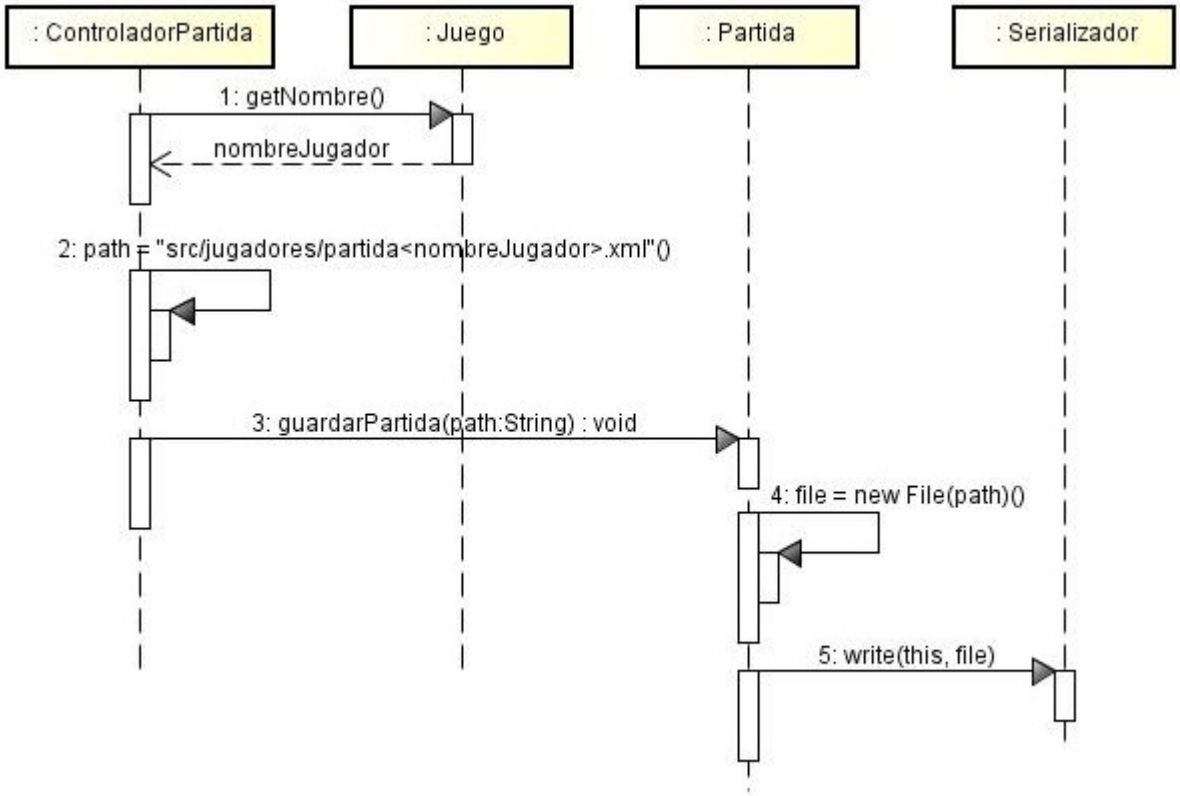
Calcular y guardar puntaje



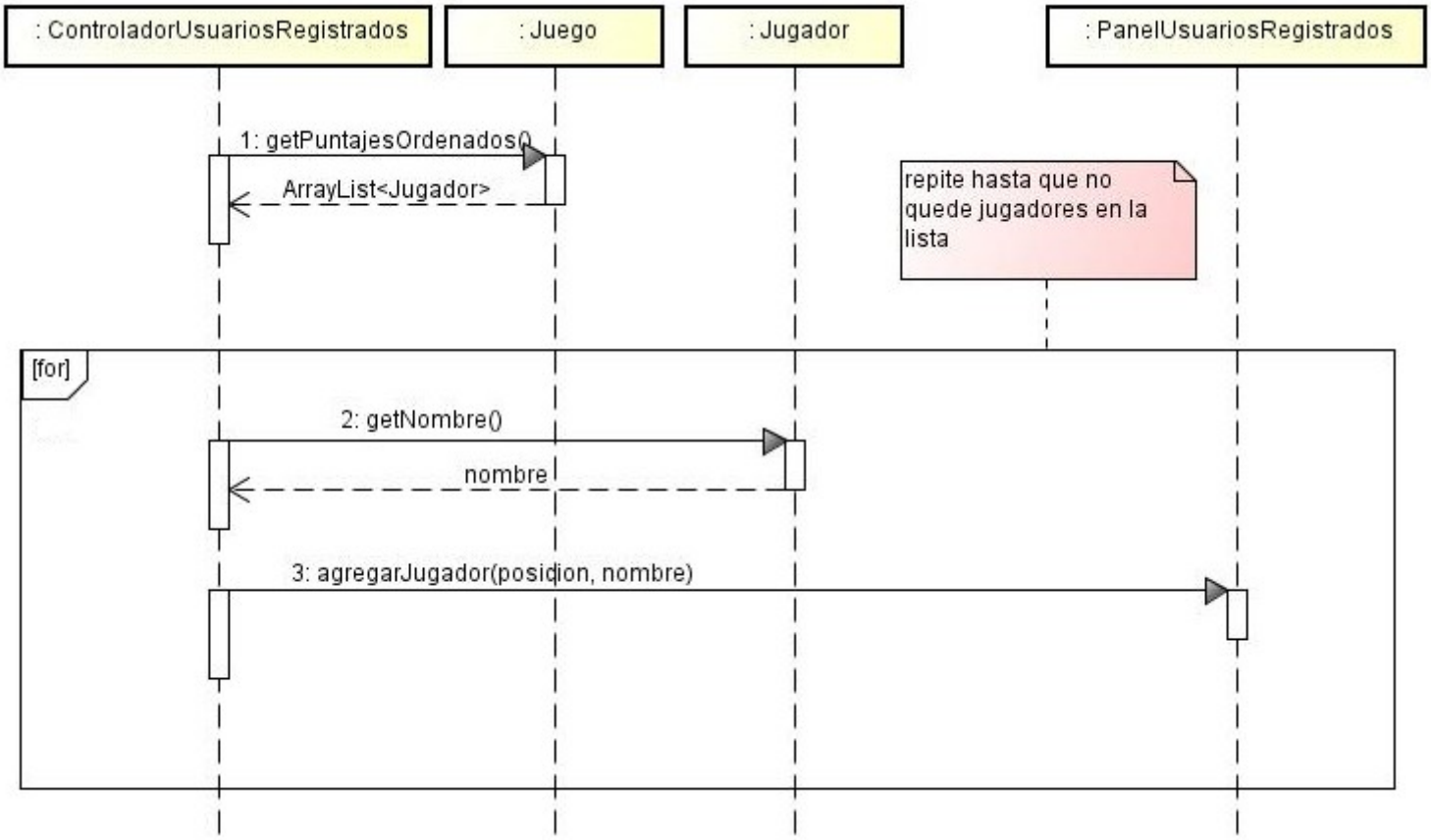
Cargar puntajes



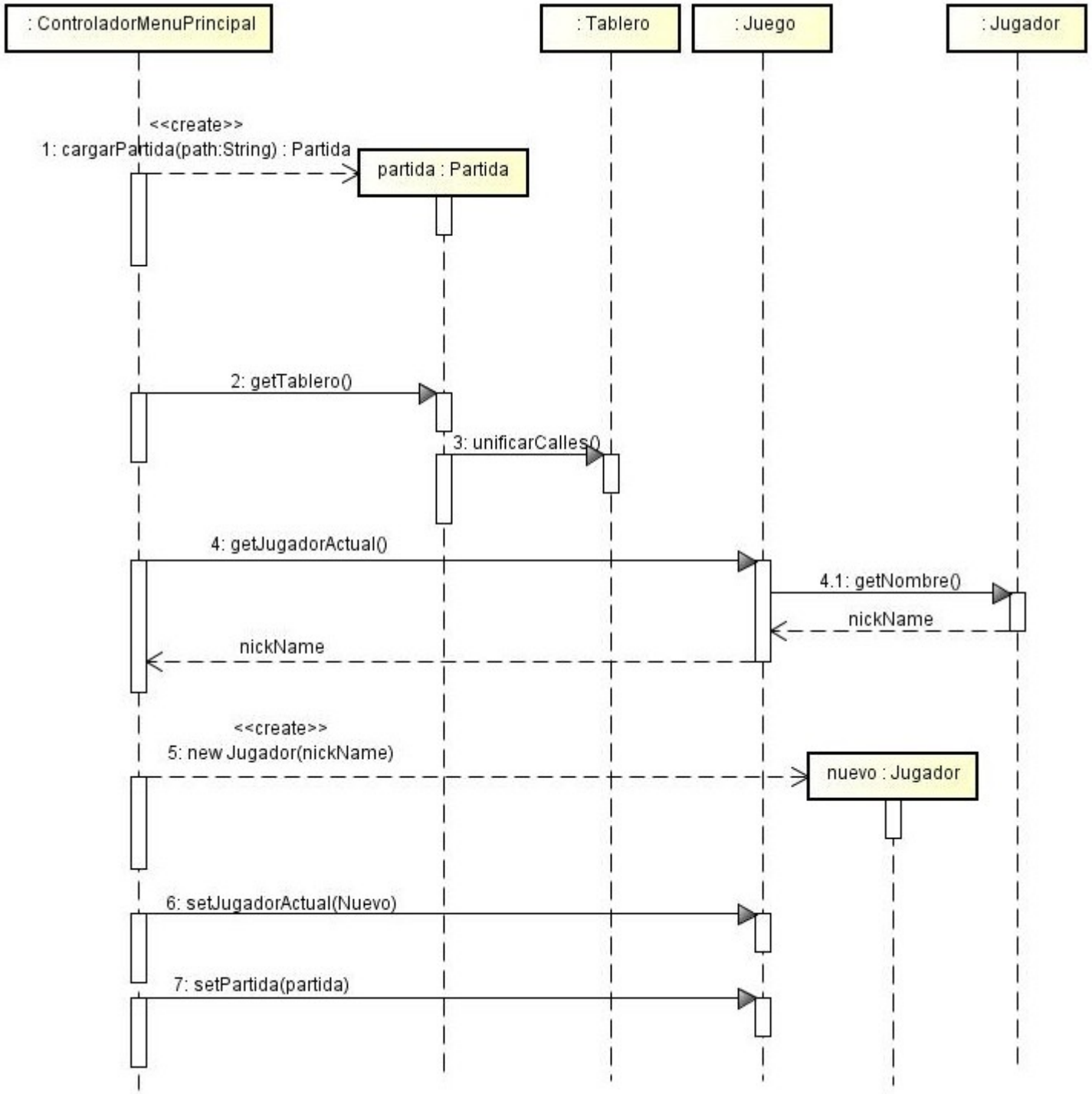
Guardar partida



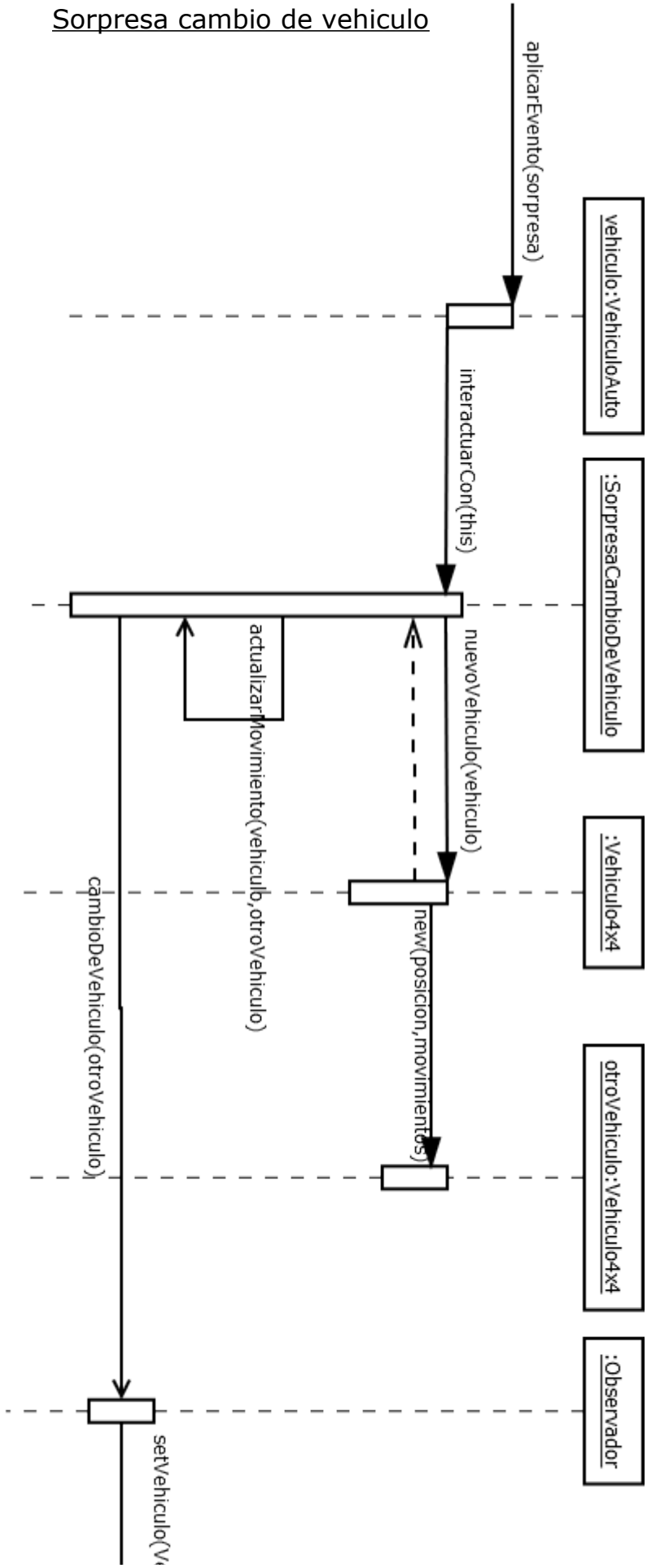
Jugadores registrados



Retomar partida


: ControladorMenuPrincipal
: Tablero
: Juego
: Jugador
<<create>>
1: cargarPartida(path:String) : Partida
partida : Partida
2: getTablero()
3: unificarCalles()
4: getJugadorActual()
4.1: getNombre()
nickName
nickName
<<create>>
5: new Jugador(nickName)
nuevo : Jugador
6: setJugadorActual(Nuevo)
7: setPartida(partida)

Sorpresa cambio de vehiculo



Dibujar sorpresa en tablero

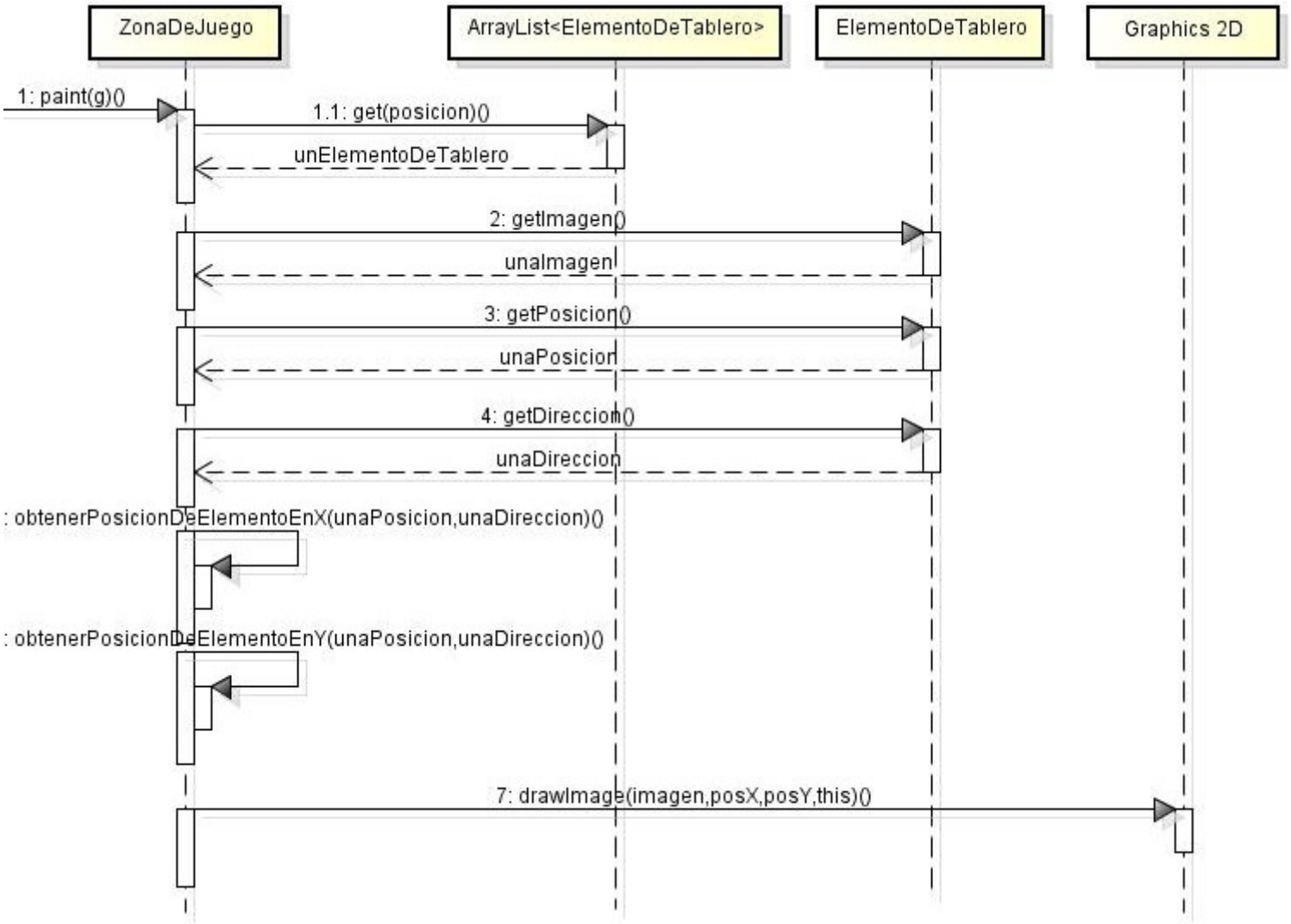
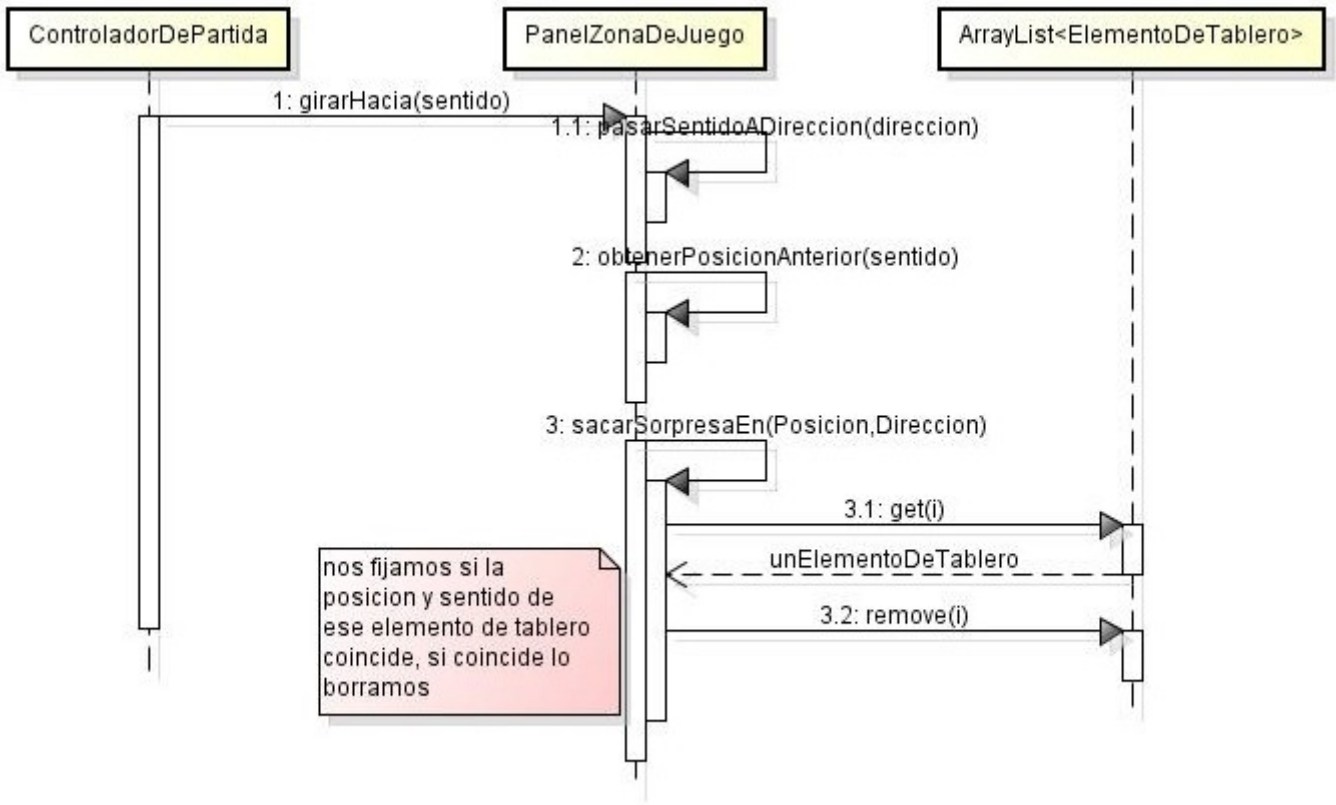
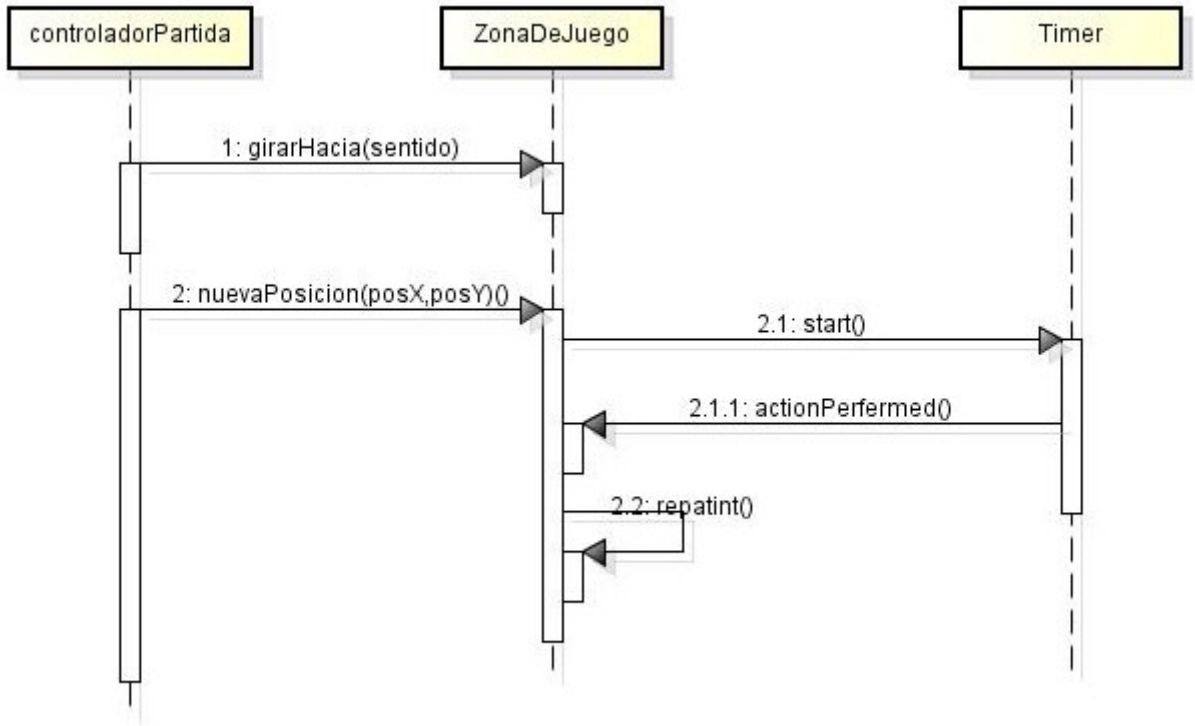


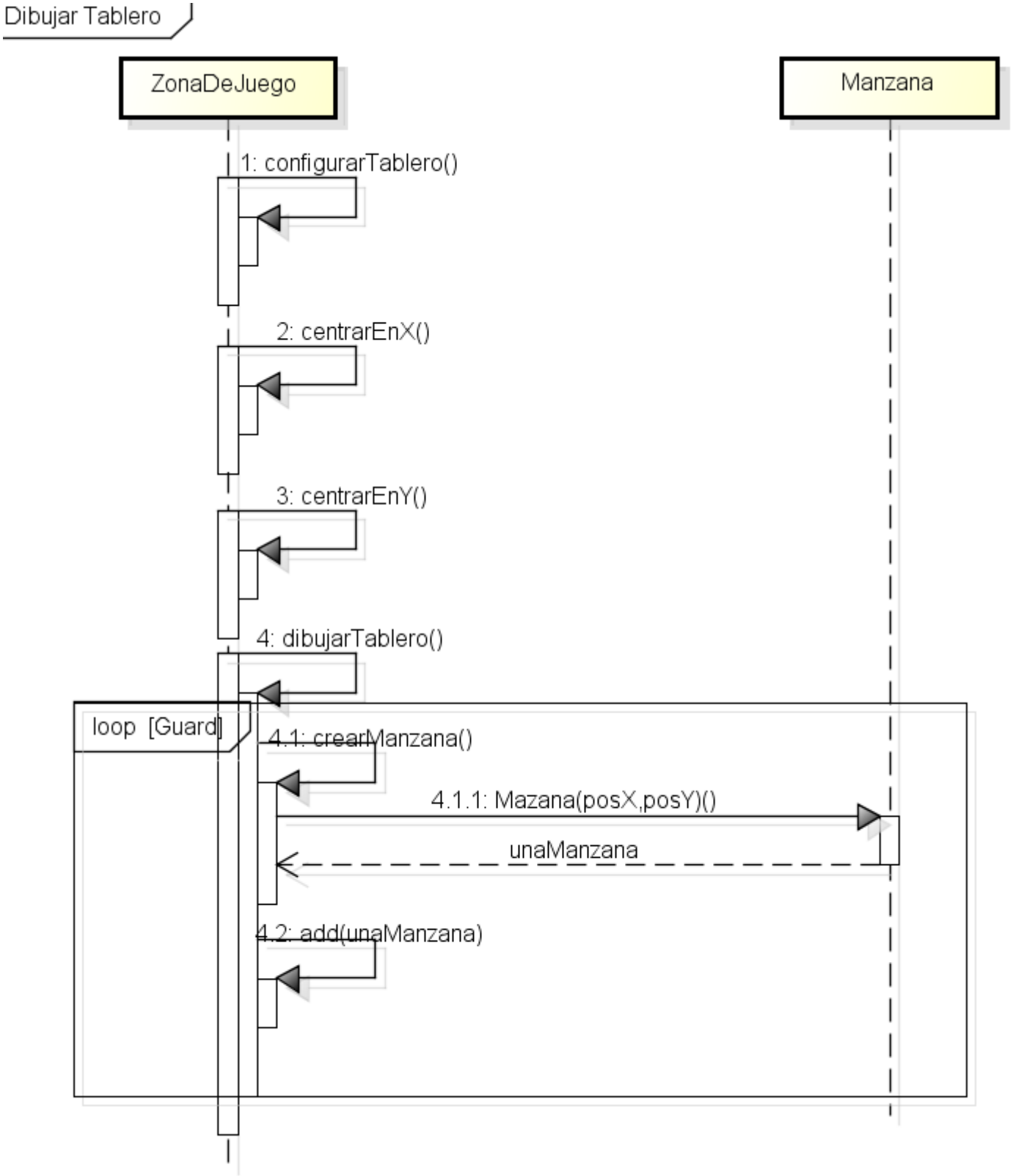
Diagrama eliminar sorpresa del tablero



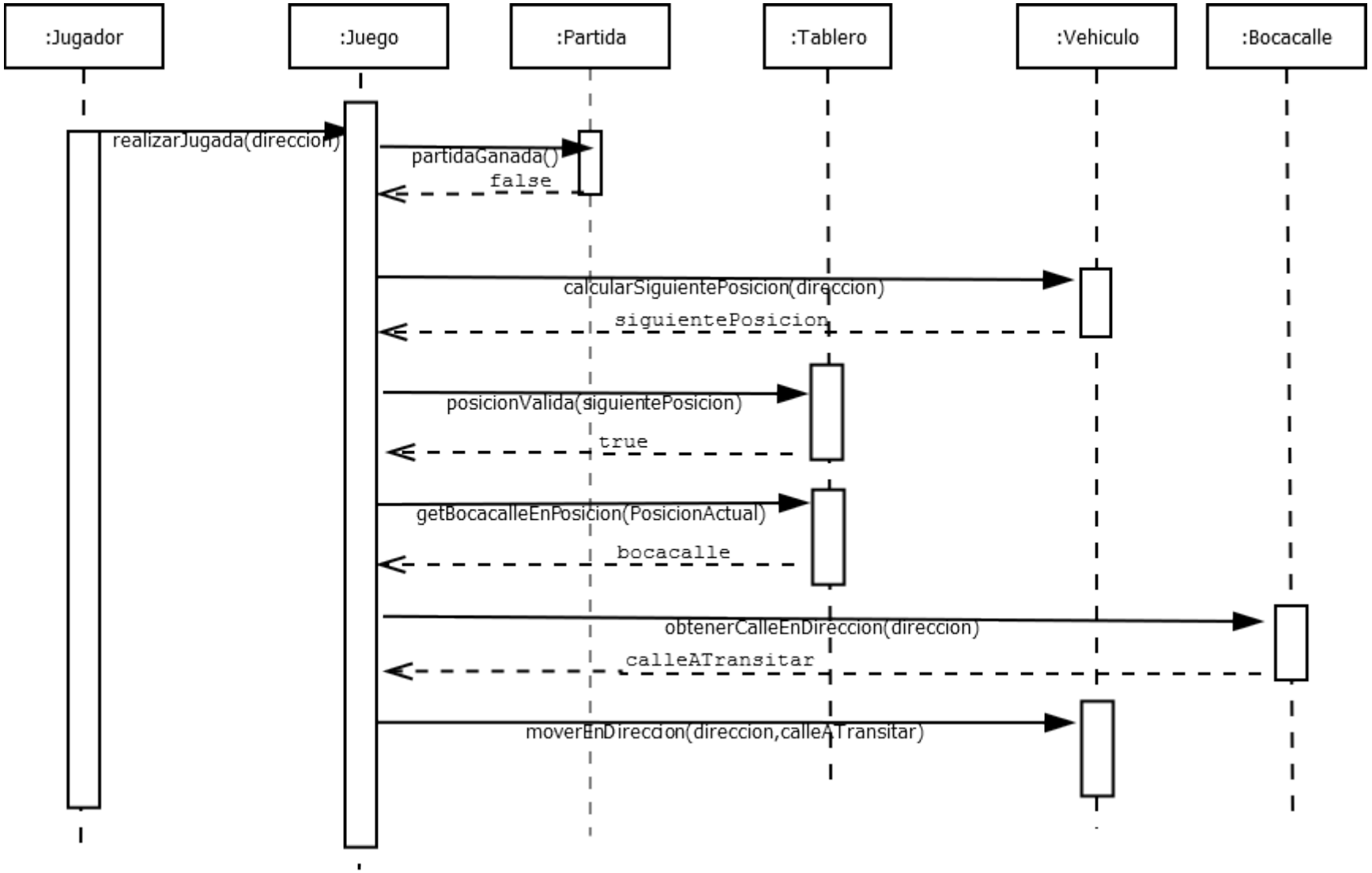
Dibujar vehículo en movimiento



Dibujar tablero



Jugada valida


:Jugador
:Juego
:Partida
:Tablero
:Vehiculo
:Bocacalle
realizarJugada(direccion)
partidaGanada()
false
calcularSiguientePosicion(direccion)
siguientePosicion
posicionValida(siguientePosicion)
true
getBocacalleEnPosicion(PosicionActual)
bocacalle
obtenerCalleEnDireccion(direccion)
calleATransitar
moverEnDireccion(direccion,calleATransitar)

Checklist de corrección

Esta sección es para uso exclusivo de la cátedra, por favor no modificar.

Carpeta

Generalidades

- ¿Son correctos los supuestos y extensiones?
- ¿Es prolija la presentación? (hojas del mismo tamaño, numeradas y con tipografía uniforme)

Modelo

- ¿Está completo? ¿Contempla la totalidad del problema?
- ¿Respeta encapsulamiento?
- ¿Hace un buen uso de excepciones?
- ¿Utiliza polimorfismo en las situaciones esperadas?

Diagramas

Diagrama de clases

- ¿Está completo?
- ¿Está bien utilizada la notación?

Diagramas de secuencia

- ¿Está completo?
- ¿Es consistente con el diagrama de clases?
- ¿Está bien utilizada la notación?

Diagrama de estados

- ¿Está completo?
- ¿Está bien utilizada la notación?

Código

Generalidades

- ¿Respeta estándares de codificación?
- ¿Está correctamente documentado?