FIUBA - 75.07

Algoritmos y programación III

Trabajo práctico 2: AlgoPoly

2do cuatrimestre, 2017

(trabajo grupal de 4 integrantes)

Alumnos:

Nombre	Padrón	Mail
Casimiro Pastine	100017	casimiropastine@gmail.com
Jonathan Medina	100052	jonathanm_96@hotmail.com
Lautaro Manzano	100274	manzano.lautaro@gmail.com
Tomás Accini	99136	tomasaccini@gmail.com

Fecha de entrega final: 30 de Noviembre de 2017

Tutor: Tomás Bustamante

Repositorio: *https://github.com/pastine/AlgoPoly*

Comentarios:

Informe

El presente informe reúne la documentación de la solución del segundo trabajo práctico grupal de la materia Algoritmos y Programación III que consiste en desarrollar un juego basado en el Monopoly, desarrollado en Java utilizando los conceptos del paradigma de la programación orientada a objetos vistos hasta ahora en el curso.

1 Supuestos

En la presente sección se especifica todas las hipótesis y supuestos que fueron tomados como verdaderos resultado de una interpretación en grupo de las ambigüedades de la consigna:

- No existe la posibilidad de vender construcciones individuales, solamente se pueden vender propiedades completas, con todas sus construcciones.
- La cantidad de casilleros que se avanza o retrocede al caer en Avance Dinámico o Retroceso Dinámico se basa en la suma de los dados con los que llegó a ese casillero.
- Al sacar dos dados iguales, no se avanza. Si se saca 2 veces, pierde el turno y queda en el mismo casillero donde empezó.
- Si en avance dinámico la cuenta de los casilleros a avanzar da negativa, entonces se queda en el lugar, no retrocede.
- Si en retroceso dinámico la cuenta de los casilleros a retroceder da negativa, entonces se queda en el lugar, no avanza.

2 Modelo de dominio

A continuación explicaremos de forma concisa el diseño general del trabajo, analizando las clases principales con sus atributos, mensajes y responsabilidades.

2.1 Casilleros y asociados

- AvanceDinámico: extiende de MovimientoDinámico, aclara que casos de movimientos dinámicos quiere aplicar.
- Cárcel: extiende de Casillero. Tiene los métodos accionar (que apresa al jugador), apresarJugador, estaEnLaCarcel, turnosQueLeFaltan y cobrarFianza, que intenta cobrarle al jugador 45000 a cambio de liberarlo.
- Casillero: Puede albergar a los jugadores, un jugador al pisar el casillero acciona un efecto especial a definir por las clases que lo heredan (accionar).
- ImpuestoAlLujo: extiende de Casillero, cuando un jugador acciona a este casillero le cobra un 10% del dinero total.
- MovimientoDinámico: extiende de Casillero. Tiene los métodos mover, y los diferentes casos del Avance y Retroceso Dinámico. Es una clase abstracta. Tanto Avance como retroceso deben indicar qué casos quieren utilizar. Tiene la capacidad de mover al jugador ya sea avanzar o retroceder.
- Policía: extiende de Casillero. Tiene una referencia a la cárcel. Tiene dos métodos, apresarJugador, que llama a su vez al mismo método de Cárcel, y accionar, que llama al método anterior.
- Quini6:
- RetrocesoDinámico: al igual que AvanceDinámico extiende de MovimientoDinámico, y aclara que casos de movimiento dinámico quiere aplicar.
- Servicio: extiende de Propiedad. Representa los servicios, y posee el atributo de costo de servicio, que es el multiplicador para los dados con los que caíste en el casillero.
- TerrenoDoble: extiende la clase Terreno, son aquellos terrenos que tienen un “par”. Por ejemplo, Bs.As. Norte y Sur. Tiene una referencia a su terreno “par”. La diferencia mayor con Terreno es la verificación que tiene que hacer, es decir, si uno quiere construir en un TerrenoDoble debe verificar tener el hermano o que el hermano tenga la misma cantidad de casas.
- Terreno: extiende Propiedad. Representa los terrenos simples, que no tienen un terreno asociado a ellos.
- Propiedad: Clase abstracta que agrupa todo el comportamiento compartido entre Terrenos y Servicios.

2.2 Jugador y estados

- Jugador: Es la clase que representa a un jugador en el juego. Conoce sus propiedades, su dinero, su nombre y el casillero en el que se encuentra parado.
- EstadoQuiniJugador: Es una interfaz que implementan los diferentes estados de Jugador contiene el método darPremioAJugador.
- EstadoJugadorGanoQuiniCeroVeces: Implementa a EstadoQuiniJugador, le acredita 50000 pesos al jugador
- EstadoJugadorGanoQuiniUnaVez: Implementa a EstadoQuiniJugador, le acredita 30000 pesos al jugador.
- EstadoJugadorGanoQuiniDosVeces: Implementa a EstadoQuiniJugador.
- EstadoDeMovimientoDelJugador: Es una interfaz que implementan los diferentes estados de movimiento, obliga a implementar el metodo mover
- EstadoLibre: Implementa a EstadoDeMovimientoDelJugador. Permite mover al jugador en la dirección en la que debe hacerlo.

- EstadoPreso: Implementa a EstadoDeMovimientoDelJugador. No permite que el jugador se mueva, descontando un día de condena al jugador.

- EstadoDirección: Es una interfaz que obliga a implementar obtenerCasilleroAdyacente y cambiarDireccion a los estados avanzar y retroceder.
- EstadoDirecciónAvanzar: Implementa a EstadoDirección. Permite al jugador ir avanzando a través de casilleros adyacentes siguientes ir hacia adelante.
- EstadoDirecciónRetroceder: Implementa a EstadoDirección. Permite al jugador ir avanzando a través de casilleros adyacentes a ir hacia anteriores atrás.

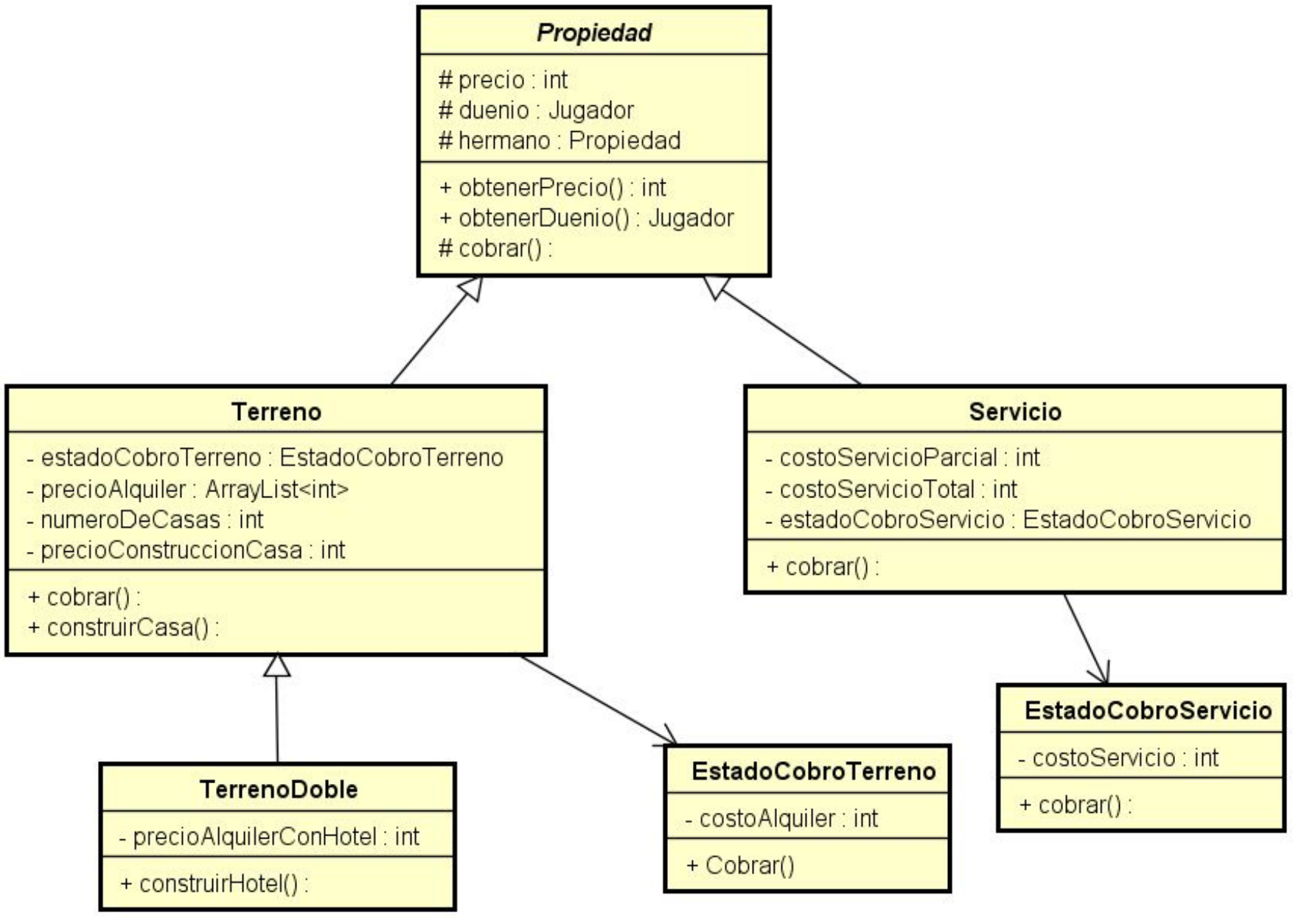
2.3 Otros

- AlgoPoly: Es la entidad encargada de manejar un turno de un jugador y de verificar si perdió luego de este.
- Dado: Es la clase que representa a un dado en el juego. Si bien se necesitan dos dados para tirarlos nosotros decidimos usar el mismo dado 2 veces.
- Tablero: Es el encargado de instanciar todos los casilleros particulares en el juego. Además es el responsable de saber que casillero viene después de cualquiera (o antes), para poder mover al jugador.
- Turnos: Es la clase que representa los turnos del juego, sabe en qué orden deben ir los jugadores. La usa AlgoPoly para abstraerse del concepto de turnos, le pide el jugador actual a turnos y lo usa.

3 Diagramas de clases

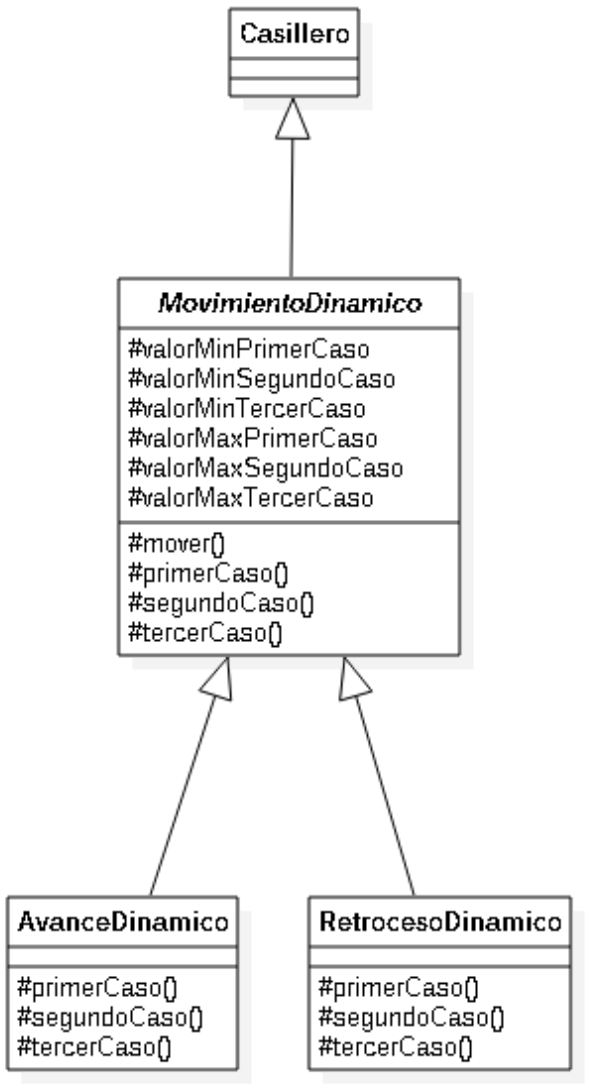
3.1 Diagrama de clase de Propiedad

En el siguiente diagrama de clases se muestra las relaciones principales de la clase Propiedad. Se observa que tanto Terreno como Servicio heredan de la clase Propiedad, ya que cumplen con la condición “es un”, y comparten mucho comportamiento propio de una propiedad. Sin embargo, la principal diferencia entre ambos es que sobre los terrenos se puede construir, mientras que en los servicios no existe esta posibilidad. A su vez, separamos los terrenos simples y los terrenos dobles en dos clases: Terreno y TerrenoDoble, que a su vez hereda de Terreno por compartir mucho comportamiento y atributos. La posibilidad que se le agrega a esta segunda clase es construir hoteles y tener un “hermano” o TerrenoDoble asociado. Al mismo tiempo, intentamos delegar la responsabilidad del cobro a dos estados, uno relacionado a los Terrenos y otro relacionado a los Servicios. Esta decisión nos pareció positiva por el hecho de que el monto a cobrar varía según el estado de la Propiedad (con o sin construcciones, si el hermano pertenece al mismo dueño).



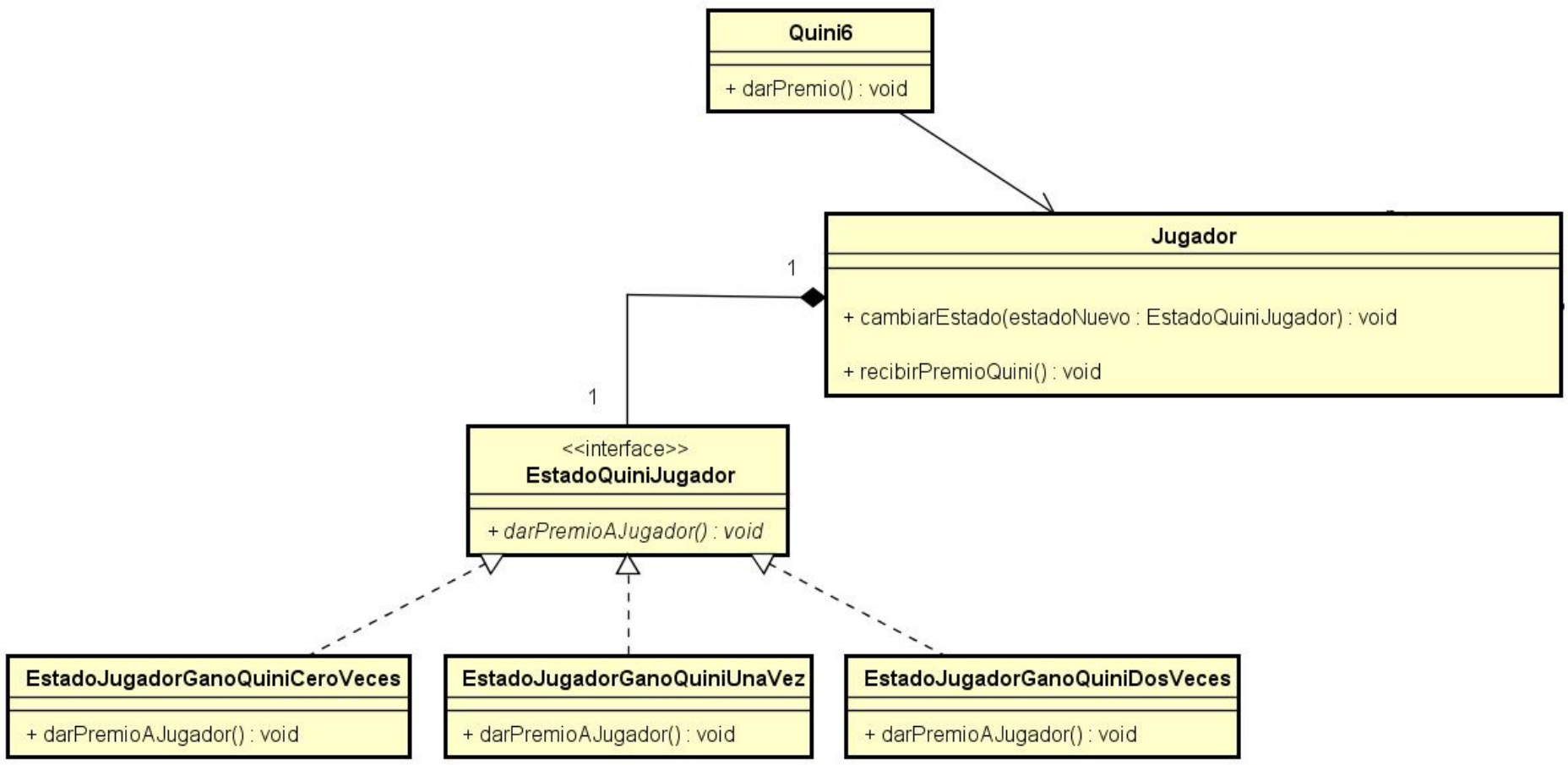
3.2 Diagrama de clase de los Movimientos dinámicos

Aquí se observa cómo fue resuelto la situación de los Movimientos Dinámicos. Estas dos clases tenían comportamientos similares, pero con las acciones invertidas, por lo que se optó en crear una clase abstracta llamada “MovimientosDinamicos”. MovimientosDinamicos tiene los tres casos posibles de Avance/Retroceso Dinámico, que deben ser redefinidos en cada clase hija.



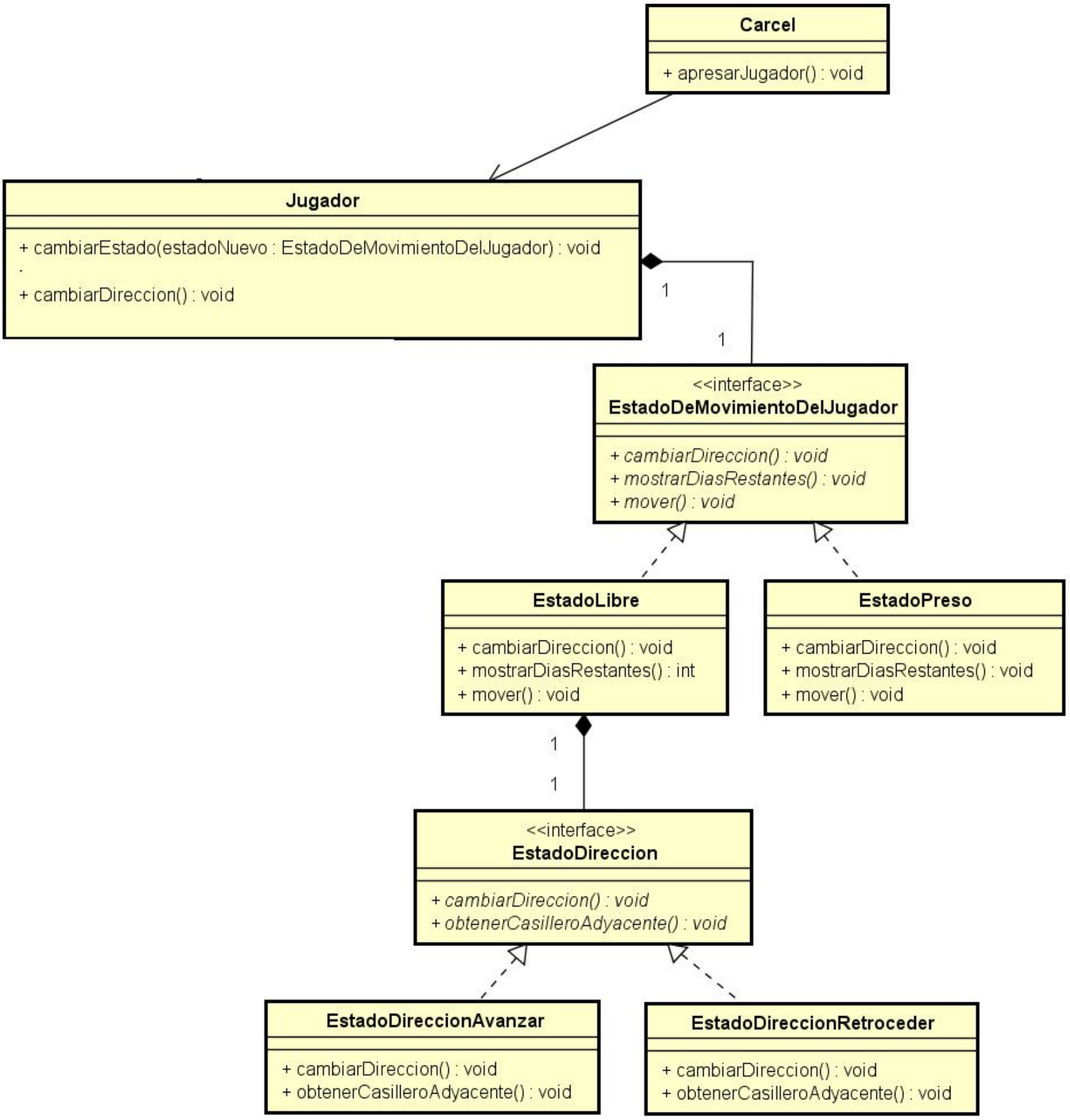
3.3 Diagrama de clase de Jugador y Quini:

En este diagrama se puede apreciar que Quini6 tiene una referencia a un Jugador, y el jugador contiene un estado de Quini, dependiendo cuantas veces se acciono el Quini6 se va modificando los diferentes estados.



3.4 Diagrama de clase de Jugador y Cárcel

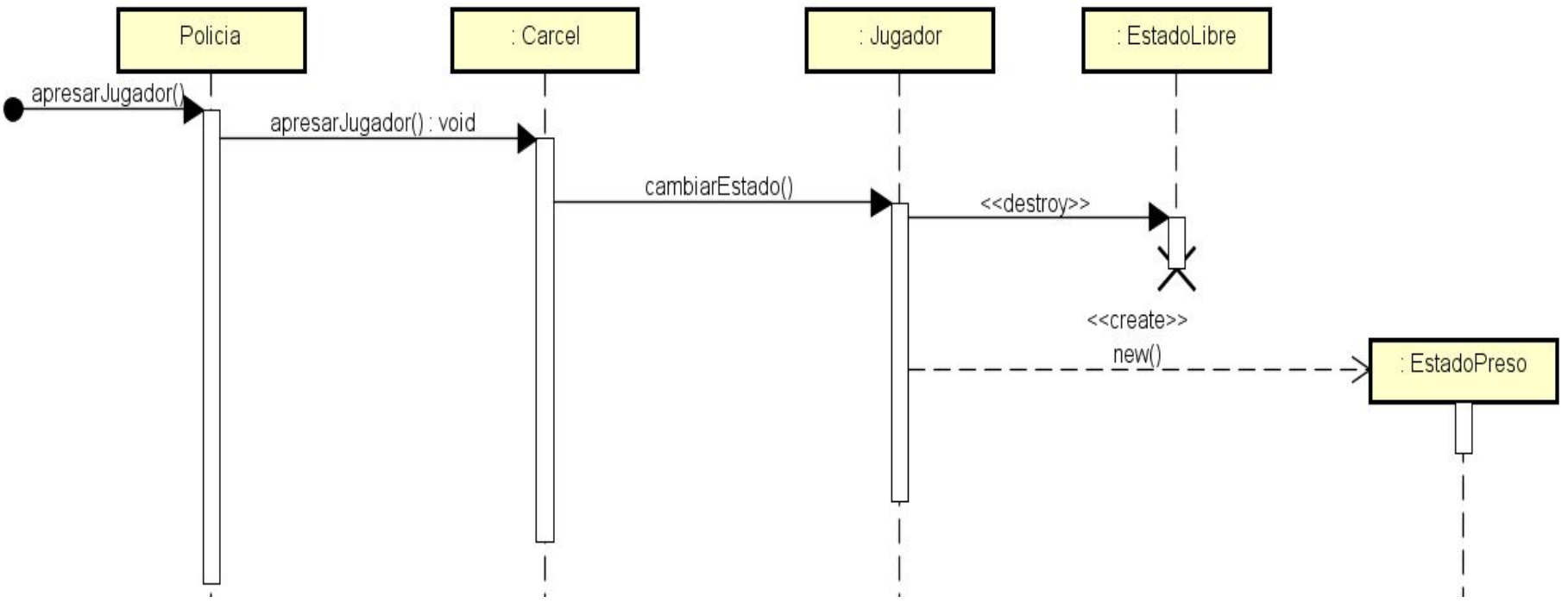
En este cuarto diagrama de clases se muestra cómo es la estructura de relaciones entre las clases Jugador, Cárcel y todos los Estados de jugador que se relacionan con su situación legal.



4 Diagramas de secuencia

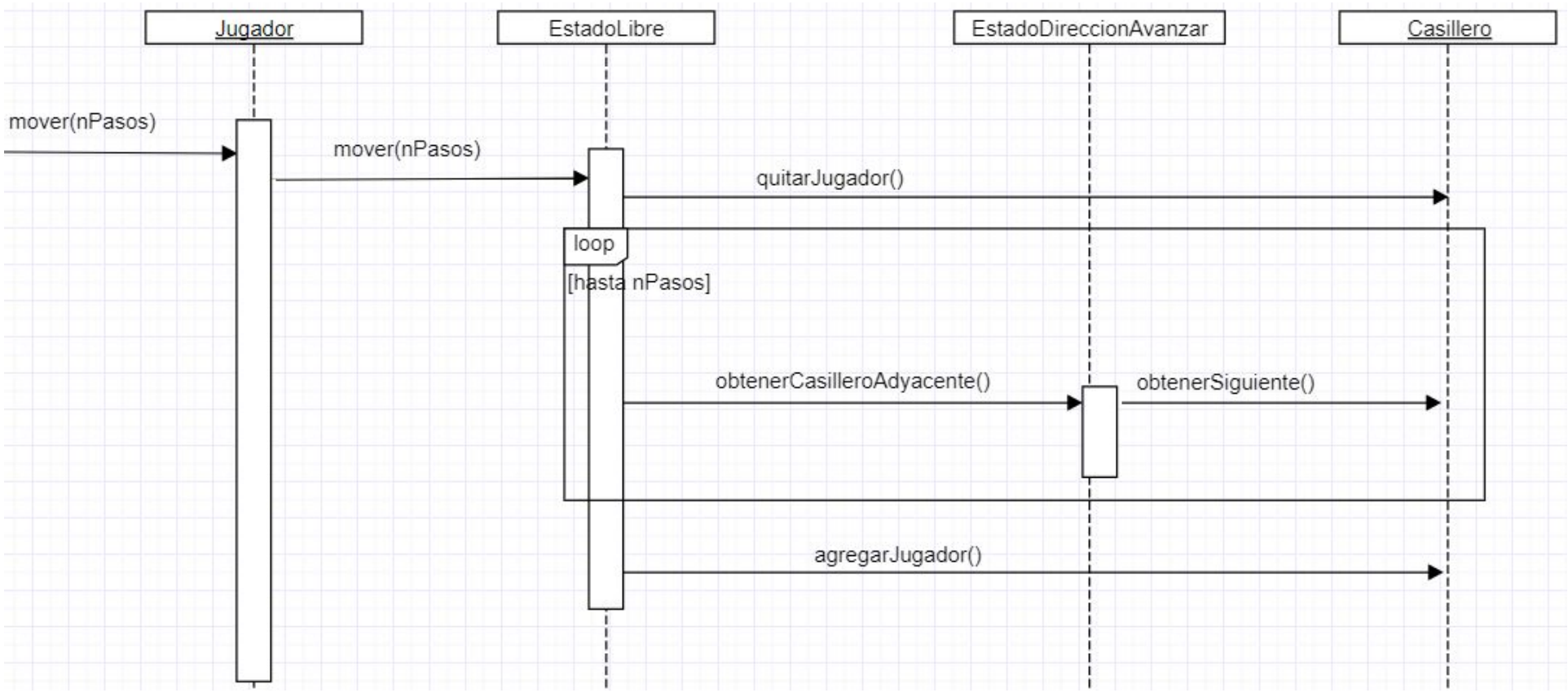
4.1 Diagrama de secuencia para apresar un Jugador

Se puede observar la secuencia desde que se le envía la orden de apresar al jugador al casillero Policía, llamando al método apresarJugador de Cárcel, cambiándole el estado al jugador de EstadoLibre a EstadoPreso.



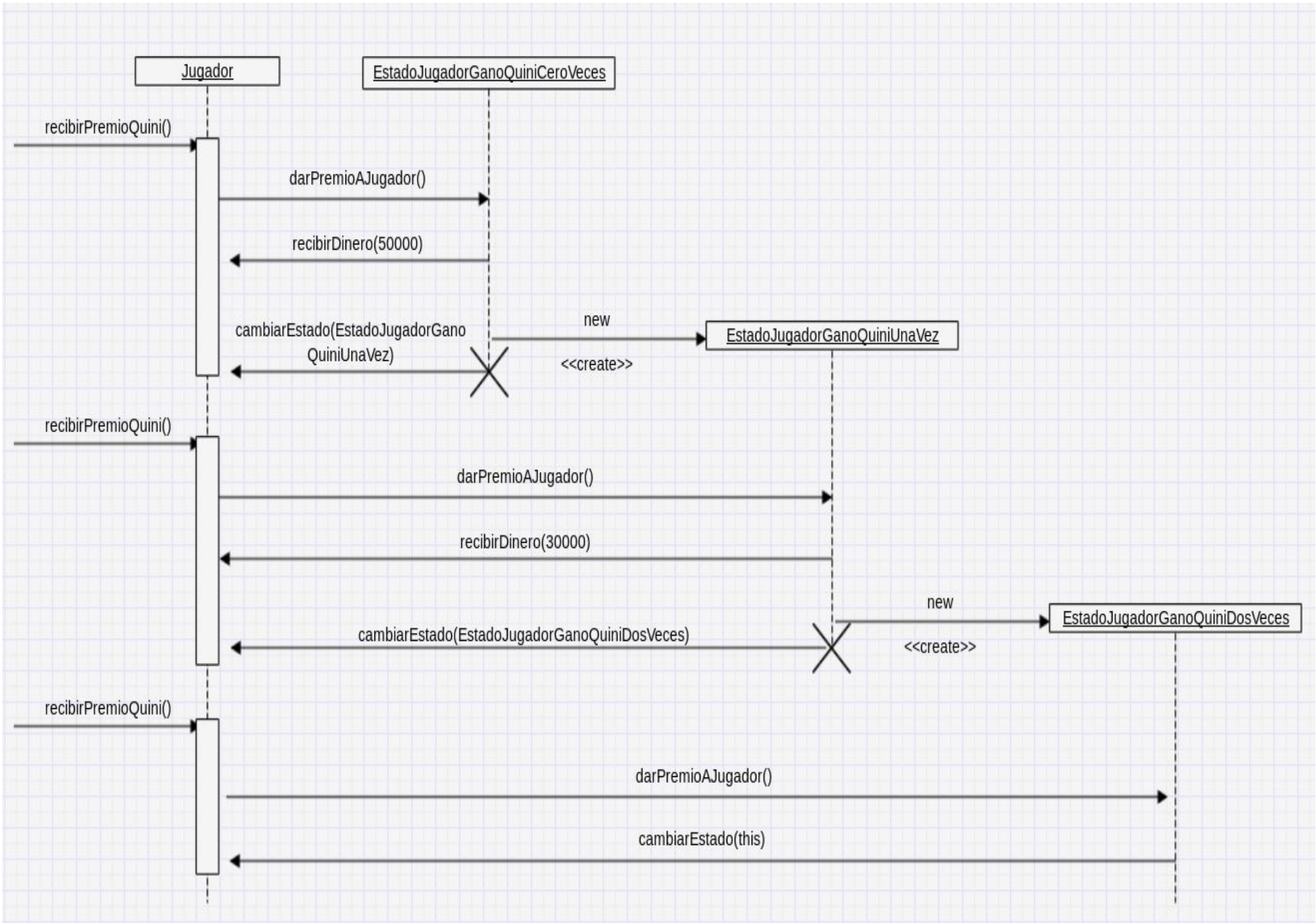
4.2 Diagrama de secuencia para mover a un jugador n casilleros

En este segundo diagrama de secuencia se aprecian los pasos que sigue el programa para avanzar a un jugador n casilleros. Empieza con un llamado al método mover(n) de la clase Jugador, el cual le manda el mensaje mover(n) al EstadoMovimiento asociado. En este diagrama en particular, se ve el EstadoLibre. Dentro de este método, se quita al jugador del casillero actual, se hace un loop en el cual se llama n veces al método obtenerCasilleroAdyacente de la clase EstadoDirecciónAvanzar, que a su vez obtiene el siguiente casillero al actual, y por último se agrega al jugador en el casillero encontrado.



4.3 Diagrama de secuencia para las diferentes caídas en Quini6

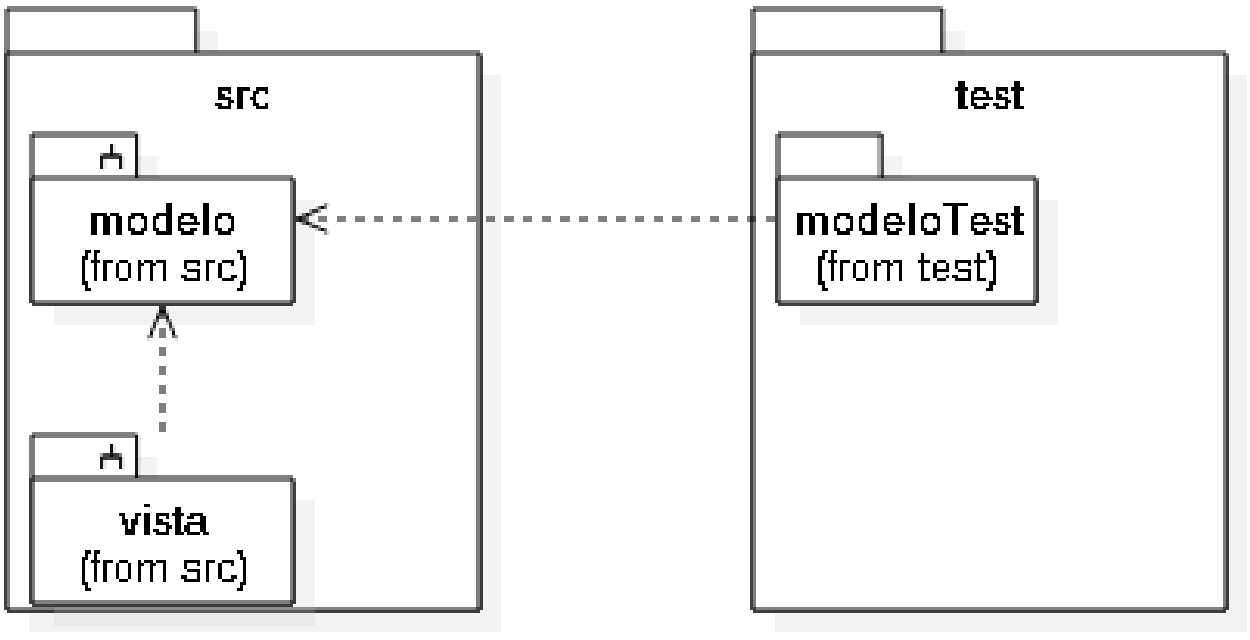
En el tercer diagrama de secuencia se observa cómo es el flujo del modelo a la hora de caer múltiples veces en el casillero Quini6. Se puede ver el patrón Strategy en acción, ya que cuando cae la primera vez, el propio estado actual llama al método cambiarEstado() enviándole el estado siguiente estado. Cada uno de los estados sabe cual es el premio que debe darle al jugador, y sabe cuál es el próximo estado al que debe cambiar.



5 Diagrama de paquetes

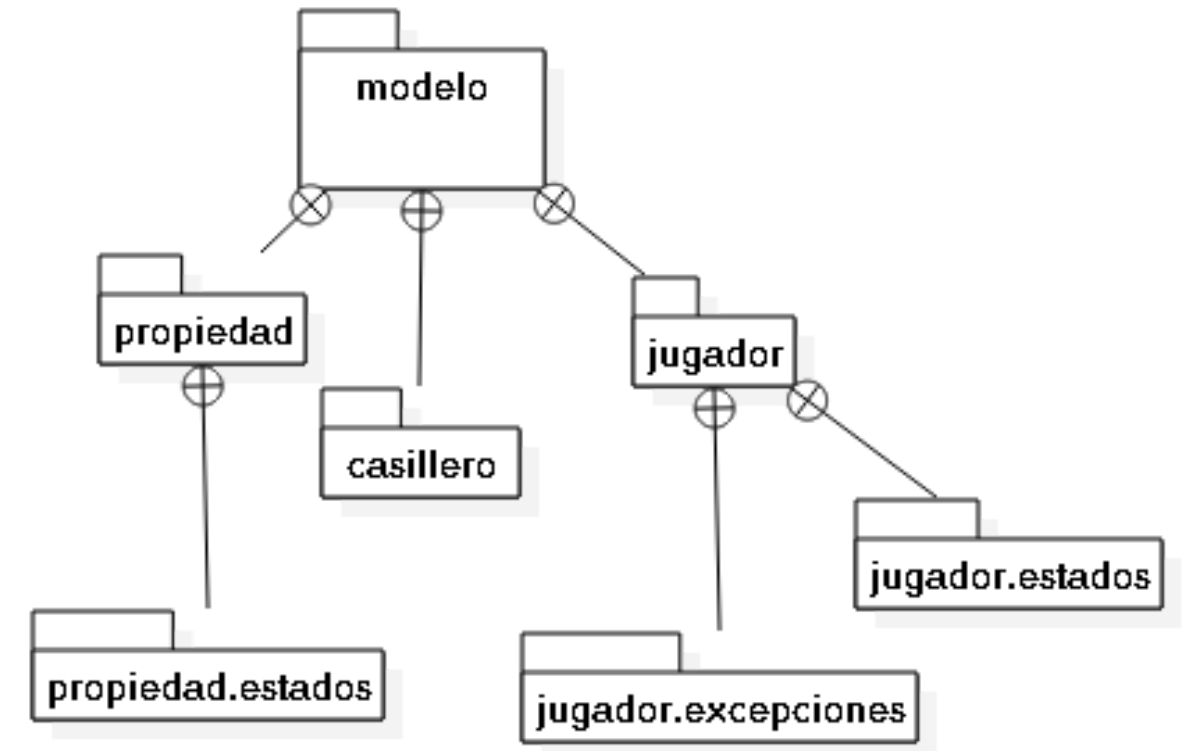
5.1 Diagrama de paquetes general mostrando dependencias

Este diagrama muestra cómo dependen los otros paquetes del paquete modelo, que se encuentra dentro de src.



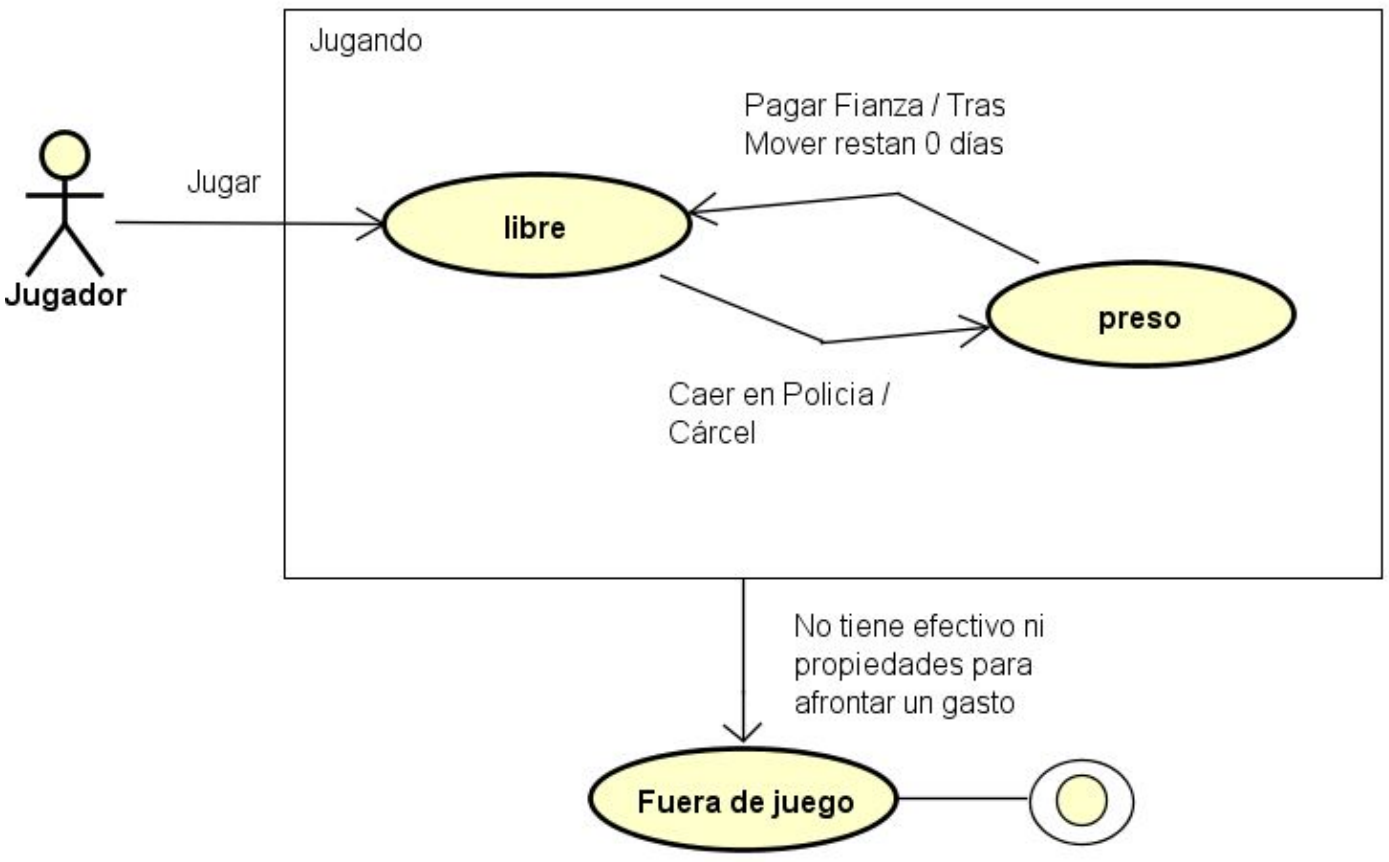
5.1 Diagrama de paquetes mostrando la estructura de paquetes en modelo

En este diagrama podemos ver cómo es la estructura general del paquete modelo, es decir, que paquetes están contenidos dentro de otros.



6 Diagramas de estado

En el siguiente diagrama de estado se muestran los diferentes estados relacionados a la cárcel por los que pasa un jugador a lo largo del juego.



7 Detalles de implementación

Quini 6:

Para poder hacer que el quini 6 devuelva diferente cantidad de dinero dependiendo cuantas veces el jugador lo acciono, lo que se hizo fue utilizar un patrón de strategy, el jugador tiene un Estado de Quni, y al accionar el casillero, el estado del jugador le da su correspondiente premio y pasa al siguiente estado hasta llegar al EstadoJugadorGanoQuiniDosVeces que se transforma en si mismo.

Avance Dinámico y Retroceso Dinámico:

Estas dos clases tenían comportamientos similares, pero con las acciones invertidas, por lo que se optó en crear una clase abstracta llamada

“MovimientosDinamicos”. MovimientosDinamicos tiene los tres casos posibles de Avance/Retroceso Dinámico. Tienen que especificar dependiendo qué número saquen se ejecuta determinada acción. El movimiento dinámico acciona el mover del jugador, cambiando la dirección para el caso de Retroceso Dinámico. Una vez ejecutada la acción se vuelve al estado original.

Diferentes Propiedades:

Agrupamos todo el comportamiento similar entre Terrenos y Servicios (cambiar y pedir dueño, comprar y saber si puede ser comprado, obtener valor de venta, etc). Además, obliga a todas sus clases hijas a implementar los métodos pedirSituacion, actualizarEstadoCobro y obtenerValorTotal.

Por su parte, Servicio tiene como particularidad que su estado de cobro cambia según el dueño del hermano (si un servicio y su hermano tienen el mismo dueño, entonces aumenta el valor por el que se multiplican los dados del jugador que cae en dichos casilleros). Por lo tanto, el método actualizarEstadoCobro se fija quién es el dueño del Servicio hermano. Al momento de cobrar, le delega esa responsabilidad a EstadoCobroServicio, que se encarga de calcular cuánto va a cobrarle al jugador según los dados obtenidos al momento de caer en ese Servicio.

A su vez, Terreno sigue la misma lógica que Servicio, pero esta vez el EstadoCobroTerreno modifica el monto a cobrar según las construcciones que haya en dicho terreno.

Tomamos la decisión de crear una cuarta clase, TerrenoDoble, que extiende la clase Terreno. TerrenoDoble agrega un nuevo atributo, que indica cuanto sale construir un hotel, y en el constructor recibe, además de todos los parámetros que recibía Terreno, el precio de construcción de un hotel y cuánto es el precio de alquiler del terreno con uno construido. También se agregó el método permiteConstruirHotel, y obtenerValorDeLasConstruccionesDelHermano, ya que al vender un TerrenoDoble se venden a su vez todas las construcciones del hermano.

Cárcel y movimiento de Jugador:

La clase Cárcel la empezamos representando lo más parecido a una cárcel de verdad, los jugadores entraban y tenía un método llamado restar día de condena, cuando su condena estaba en 0 el jugador salía.

Luego decidimos cambiar un poco la representación. Al jugador le agregamos un estado que se encarga del movimiento de este. Cuándo comienza el juego el jugador tiene un estado libre, puede moverse por el tablero normalmente. Pero cuándo el jugador cae en la cárcel su estado cambia a preso y cuando se llama al método mover de este estado le resta un día de condena.

Para lograr este comportamiento usamos el patrón de diseño Strategy, que te permite inyectar comportamiento dentro de una clase por un estímulo causado desde afuera.

Diferentes movimientos del Jugador:

El movimiento del jugador está representado en el EstadoLibre, ya que cuando el jugador se encuentra preso no va a poder moverse. El jugador puede moverse hacia adelante o hacia atrás (únicamente en retroceso dinámico). Para implementar eso nuevamente recurrimos al patrón de diseño Strategy. En EstadoLibre tenemos un estado llamado EstadoDirección que puede tomar dos direcciones, avanzar y retroceder. Si está en avanzar pide el casillero siguiente al actual y si está en retroceder pide el anterior.

Lo más común es que esté en avanzar ese estado, únicamente cambia a retroceder cuando cae en retroceso dinámico. Cuando esto ocurre el estado cambia al comienzo del movimiento del jugador y vuelve a cambiar cuando termina.

8 Excepciones

- **CasilleroNoEncontradoException:**
  - Se usa esta excepción en la navegación del tablero por los casilleros. Este tiene dos métodos, obtenerAnterior y obtenerSiguiente, que se van a usar para mover el jugador entre los casilleros. Si al buscar un casillero en el Array, este devuelve un index de -1, se lanza esta excepción. Es decir, si a cualquiera de estos métodos se le pasa un casillero que no existe, se va a lanzar la excepción.
- **JugadorEstaPresoException:**
  - Esta excepción fue creada para el momento en el que un jugador quiere realizar un movimiento pero está en la cárcel, entonces no se debería poder mover. Cuándo el jugador quiere moverse pero está en la cárcel, se le resta un día y luego se lanza la excepción para notificar que no se va a mover.
    La excepción se atrapa en la clase AlgoPoly únicamente, pero fué de mucha utilidad a la hora de hacer los tests unitarios de Cárcel. Ya que nos ayudó a determinar que un jugador estando en la cárcel no puede moverse, sin la necesidad de tener el concepto de casillero, tirando la excepción cuándo se llamaba al método mover.
- **SaldoInsuficienteException:**
  - Es lanzada cuando el jugador quiere hacer algún tipo de gasto mayor a su dinero actual. Se lanza en un método de Jugador quitarDinero.
    Esta también es atrapada por AlgoPoly, cuándo un jugador cae en una propiedad de otro y no tiene saldo suficiente como para pagar el alquiler. Lo que se hace es vender ciertas propiedades del jugador que no tiene dinero para poder afrontar el gasto, si este queda sin propiedades, pierde el juego.
- **PropiedadConDuenioException:**
  - Se lanza cuando un jugador quiere comprar o realizar una operación sobre una propiedad que le pertenece a otro jugador. Por ejemplo, la clase Terreno la lanza cuando un jugador quiere construir una casa sobre un terreno que no es de el.
    Si bién, en la vista, el botón para construir sobre un terreno o comprar una propiedad está habilitado únicamente para aquel jugador que pueda comprar o construir, esta excepción nos ayudó mucho a la hora de hacer los tests y verificar que nuestro código sea robusto independientemente de la parte gráfica.
- **ConstruccionNoPermitidaException:**

- Cuándo esta excepción es lanzada es porque el jugador quiso hacer una mejora o construcción en un terreno en el que no se puede todavía. Por ejemplo, un jugador tiene Buenos Aires Sur y Buenos Aires Norte, con 1 casa en BS.AS. Sur. Entonces, si el jugador decide construir otra casa en BS.AS sur, esta excepción va a ser lanzada.