**FIUBA - 75.07**

**Algoritmos y programación III**

*Trabajo práctico 2: AlgoPoly*

2do cuatrimestre, 2017

(trabajo grupal de 4 integrantes)

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Padrón** | **Mail** |
| Dan Tchechenistky | 100117 | dtcheche@gmail.com |
| Andres Visciglio | 100118 | atvisciglio@gmail.com |
| Tomás Sabao | 99437 | tomalon95@gmail.com |
| Luciano Ortiz | 100323 | lucianortiz97@gmail.com |

***Fecha de entrega final***: 30 de Noviembre de 2017

***Tutor***: Pablo Suárez, Matías Leal Bazterrica

***Comentarios***:

## 

## Introducción

El presente informe reúne la documentación para la solución del tercer trabajo práctico de la materia Algoritmos y Programación III, que consiste en desarrollar una aplicación del MonoPoly, con adaptaciones en sus reglas y dinámica de juego. El lenguaje utilizado es Java fueron utilizados los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

## Supuestos

Se tuvieron las siguientes consideraciones a la hora de implementar el comportamiento de AlgoPoly:

* En el caso en el que un jugador saca dobles, esa tirada no se contará y el jugador debe lanzar nuevamente los dados.
* El jugador que cumpla los requisitos necesarios para construir, solo podrá hacerlo si se encuentra en el casillero sobre el cual quiere construir.
* Avance Dinámico se activa automáticamente, situando al jugador en el casillero correspondiente según el algoritmo indicado. Lo mismo sucede con Retroceso Dinámico.
* Si un jugador debe pagar un alquiler y no cuenta con el efectivo suficiente, todas sus propiedades serán vendidas. Si luego de vender esas propiedades, aún no puede pagar el alquiler, el jugador es eliminado.
* Si un jugador fue eliminado por no pagar el alquiler, el pago al dueño de la propiedad se realiza de todas formas.

## 

## 

## Modelo de dominio

El modelo del programa está compuesto por total de X clases, con el fin de modelar el juego Monopoly. El siguiente listado menciona y detalla la responsabilidad de cada una de ellas.

* *Cárcel*: Encargada de modificar el movimiento del jugador y de cobrar la fianza.
* *Quini6*: Clase cuya responsabilidad es premiar al jugador dependiendo de sus recompensas anteriores.
* *Policía*: Su responsabilidad es enviar al *Jugador* a la *Cárcel*.
* *Caible*: Clase abstracta cuyo fin es agrupar *Cárcel*, *Quini6*, *Policía*, entre otros y unificar la recepción de un determinado mensaje, en este caso accionar( ).
* *Movimiento Normal Y Cárcel*: encapsulan la movilidad del jugador, de esta forma no es necesario que el jugador conozca la movilidad dentro del tablero.
* *RetrocesoDinámico*: Su responsabilidad es retroceder una cantidad determinada de casilleros al jugador que caiga en él, dependiendo de lo que saco en sus dados.
* *AvanceDinámico*: Su responsabilidad es avanzar una cantidad determinada de casilleros al jugador que caiga en él, dependiendo de lo que saco en sus dados.
* *BarrioNormal*: Clase que representa a aquellas propiedades que se encuentran divididas en Norte y Sur.
* *BarrioEspecial*: Clase que representa a las demás propiedades que no son compañías y que no pueden construir hoteles
* *Compania*: Clase que agrupa a los distintos servicios *Aysa*, *Edesur*,*Subtes* y *Trenes*, los cuales tienen mecánicas especiales para el cobro de alquiler y no pueden realizar construcciones.
* *Tablero*: Clase que representa el tablero de juego. Todos los posibles casilleros ya se encuentran inicializados en el mismo
* *EstadoConstruccion*: Representa a las construcciones que pueden realizarse en cada propiedad. Se encarga de devolver el alquiler correcto de la propiedad según el número de construcciones realizadas.
* *Partida*: Agrupa los elementos indispensables para el desarrollo del juego; *Jugador*, *Tablero*, *Turno*, entre otros.
* *Turno*: Se crea un nuevo turno para cada jugador y luego es descartado. Aquí se tiran los dados y se acciona el casillero en el cual el jugador cayó.
* *Propiedad*: Agrupa todos los bienes materiales que puede tener un jugador.
* *Caible*: Agrupa todos los distintos casilleros que tiene el juego.
* *EstadoDuenio*: Recurso que permite un accionar diferente en cada casillero, dependiendo del estado del casillero, *ConDuenio* o *SinDuenio*.
* *Tirador*: Utiliza Dados para obtener números al azar, aquí se verifica que los números no coinciden (ver Supuestos).

## Detalles de implementación

### Casilleros:

Al momento de plantear la implementación de los casilleros surgió la idea de hacer una clase abstracta Caible que tenga como atributo una posición en el tablero y un método que la accione. De esta clase heredarian entonces los diferentes tipos de casilleros. Por un lado estarían los casilleros que el jugador puede comprar, estos se representan con una clase abstracta Propiedad. Esta contiene los atributos generales a todas las propiedades. Y de ella heredan Barrio y Compañía. Por otro lado se encuentran los casilleros que no pueden ser comprados por el jugador como lo son por ejemplo Carcel, Policia, Avance Dinámico… Todos estos son representados por clases individuales ya que su lógica es independiente.

### Estados de propiedad

Para determinar el comportamiento de las propiedades que poseen o no un dueño, decidimos aplicar el state pattern. Se creó una clase abstracta “EstadoDuenio” de la cual se producen otras dos clases “ConDuenio” y “SinDuenio”. Dado que el comportamiento de las propiedades se encuentra determinado por si tienen o no un propietario, haciendo que al caer los jugadores se mande un mensaje directamente al estado de la propiedad nos ahorra la utilización de numerosos if, haciendo que el código sea más claro y legible, y al mismo tiempo nos otorga más flexibilidad ante cambios en los requisitos propuestos.

### Quini 6

Para la implementación del quini 6 utilizamos una clase abstracta premio, de la cual obtuvimos los 3 premios que pueden ocurrir en el transcurso de una partida: Premio50K

Premio30K y PremioNulo. El jugador guarda una instancia premio, y al momento de caer en Quini6 se cobra el premio actual. Una vez cobrado el premio, el mismo instancia un nuevo premio que corresponde al que le sigue( Premio 30k para Premio50K, PremioNulo para Premio30K) el cual es guardado por el jugador, reemplazando a su predecesor. Una vez que se llega al premio nulo, no se modifica màs el premio.

### Avance y retroceso dinámico

El avance y retroceso dinámico son muy similares. Al caer un jugador en este casillero se determina cuántas posiciones deberá avanzar/retroceder mediante if’s, luego cuentan con un método que hará respectivamente avanzar/retroceder esa cantidad de casilleros al jugador.

Estados de construcción

Se trata de una clase abstracta que modela las construcciones que pueden realizarse en las distintas propiedades. En función de la construcción que representen, cada una devuelve el actual costo de la renta que le corresponde al nivel de construcción. Con el fin de evitar sobrecargar de información a la clase que instancia los distintos estados de construcción, todos los EstadosConstruccion son creados en el constructor de la Propiedad correspondiente y agrupados en una lista. Y es debido a esta decisión, que la única forma de operar los estados es a través de un índice. De otra forma, por ejemplo que los estados sepan construirse uno a otro, no habría forma de recuperar la información sobre los costos de cada Construcción en particular, ya que éstos varían según el Barrio.

BarrioNormal

Se categoriza como BarrioNormal a aquel Barrio que cuente con un Barrio dupla/par. Es necesario, a lo largo del desarrollo del juego, acceder desde un BarrioNormal, a su Barrio dupla. Para solucionar este problema se opto por que cada barrio tenga un atributo *nombre* que lo identifique y otro con el nombre de su dupla. Se descarto la idea de que cada BarrioNormal tenga una referencia directa a su Barrio dupla debido a que al crear un BarrioNormal, éste debería recibir en el constructor a su Barrio dupla, pero esto generaría un loop. La opción de asignarle una referencia fuera del constructor fue descartada ya que el constructor dejaria al BarrioNormal en un estado inválido hasta que se le asigne su barrio dupla. Son estos los motivos por los cuales optamos por identificarlos con un nombre fijo.

## Excepciones:

Se implementaron las siguientes excepciones para cubrir posibles errores; MontoInsuficienteException, FianzaInhabilitadaException, JugadorEnBancarrotaExcepcion, RequisitosInsuficientesException

Como es de esperar en el desarrollo del juego, existe la posibilidad de que el jugador no tenga el dinero suficiente para pagar lo que él desea. Es por eso que MontoInsuficienteException aparece en esos casos, para evitar que se realicen operaciones inválidas y perjudiquen la integridad del juego.

Por otro lado tenemos FianzaInhabilitadaException, la cual surge en los casos en que el jugador intenta pagar la fianza no habiendo dejado pasar un turno obligatorio en la Cárcel. De esta forma se respetan las reglas establecidas por el juego.

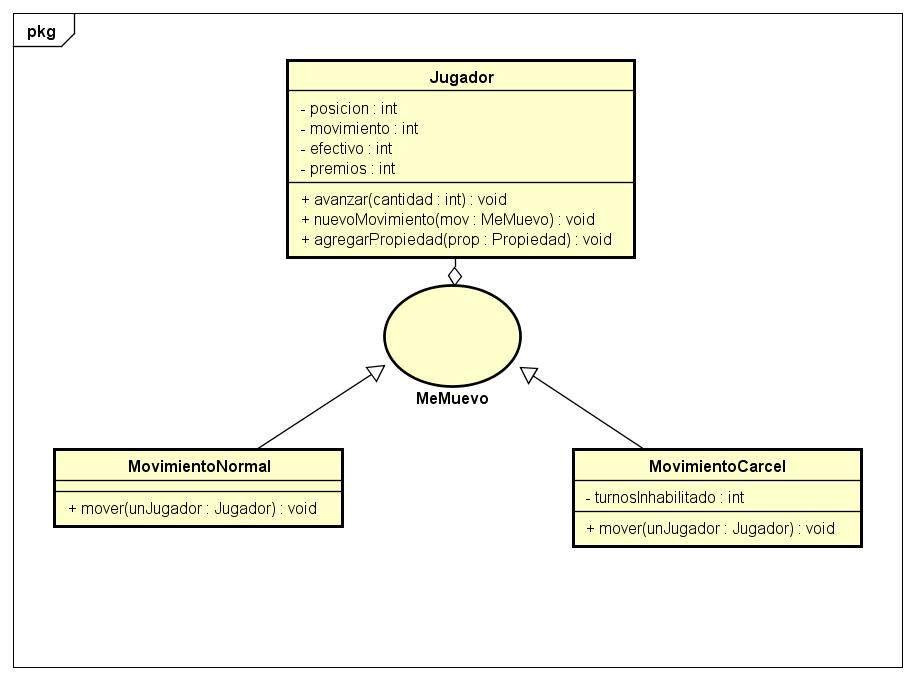
El momento en que un jugador queda eliminado del juego es el momento en que èste queda en bancarrota. JugadorEnBancarrotaExcepcion aparece en el momento en que se le intenta cobrar a un jugador, y el monto a cobrar es superior a lo que puede llegar a pagar usando todos sus recursos.

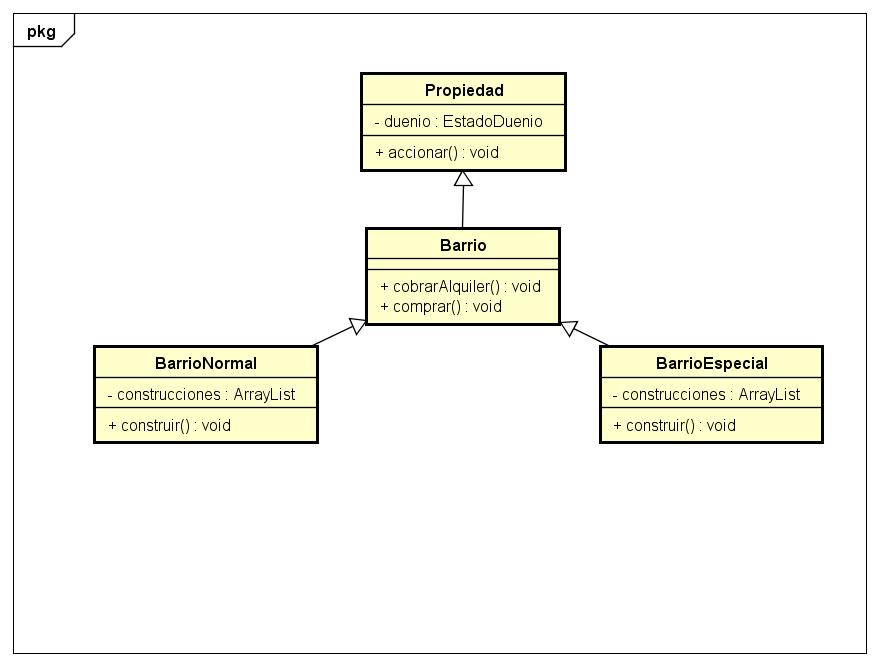
RequisitosInsuficientesException es una excepcion creada para la realizaciòn de las construcciones en los distintos barrios, y como su nombre lo indica se aplica cuando el jugador no posee los requisitos suficientes para comenzar dicha construcción. Aquellos barrios que poseen tanto una región norte como una región sur tienen como requisito para poder empezar a construir, ambas regiones deben estar en manos de un mismo jugador. Suponiendo el caso de que un jugador trate de construir sin cumplir este requisito, ésta excepción será lanzada.

Otras excepciones que no manejan errores también fueron creadas. Las mismas son utilizadas para especificar un comportamiento en la interfaz visual que interactúa con el usuario. Los distintos objetos “vistas” que componen dicha interfaz son quienes capturan estas excepciones y muestran distintos tipos de información necesaria para que el usuario tenga mayor conocimiento de lo sucedido en el transcurso del juego. Algunas de estas excepciones son NumeroDobleEnDadosException y PartidaFinalizadaException .

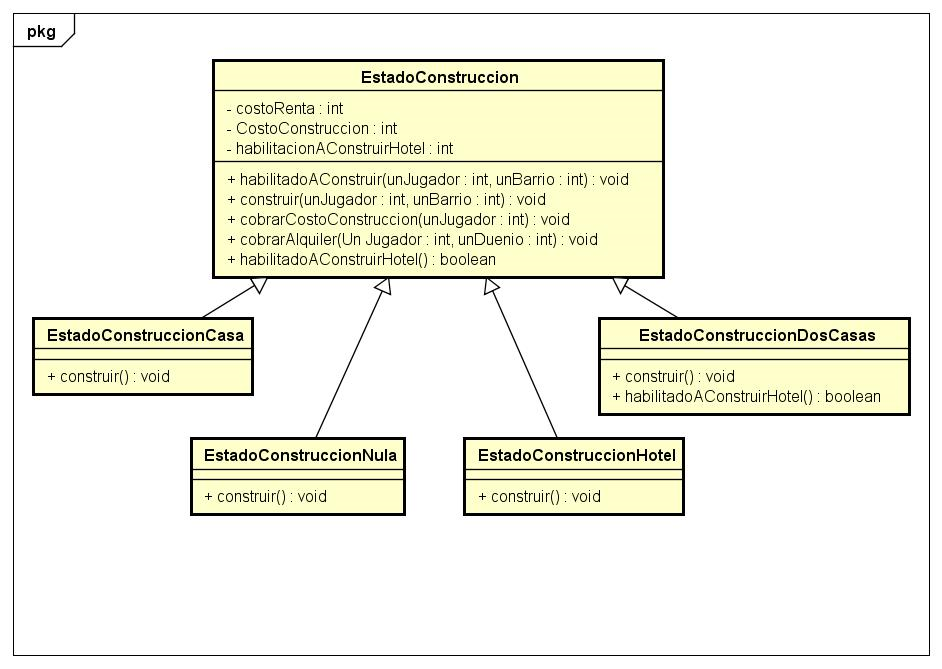
## Diagramas de clase

Diagrama que muestra el movimiento del jugador a través de una interfaz *MeMuevo*.

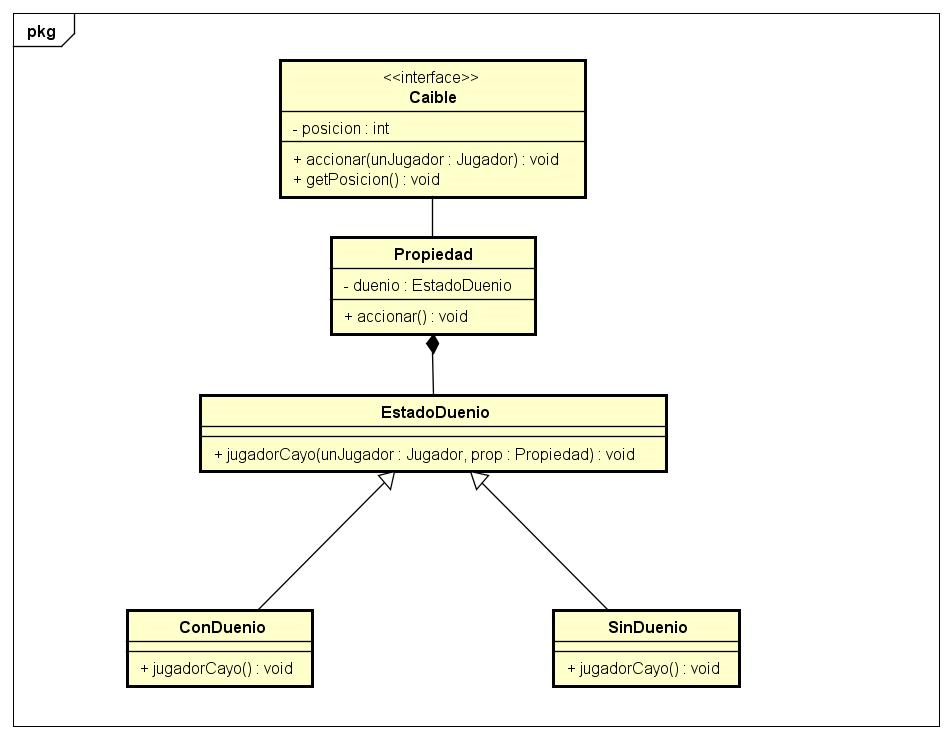


Jerarquía del esquema Barrio - Propiedad

EstadoConstruccion y sus derivados.

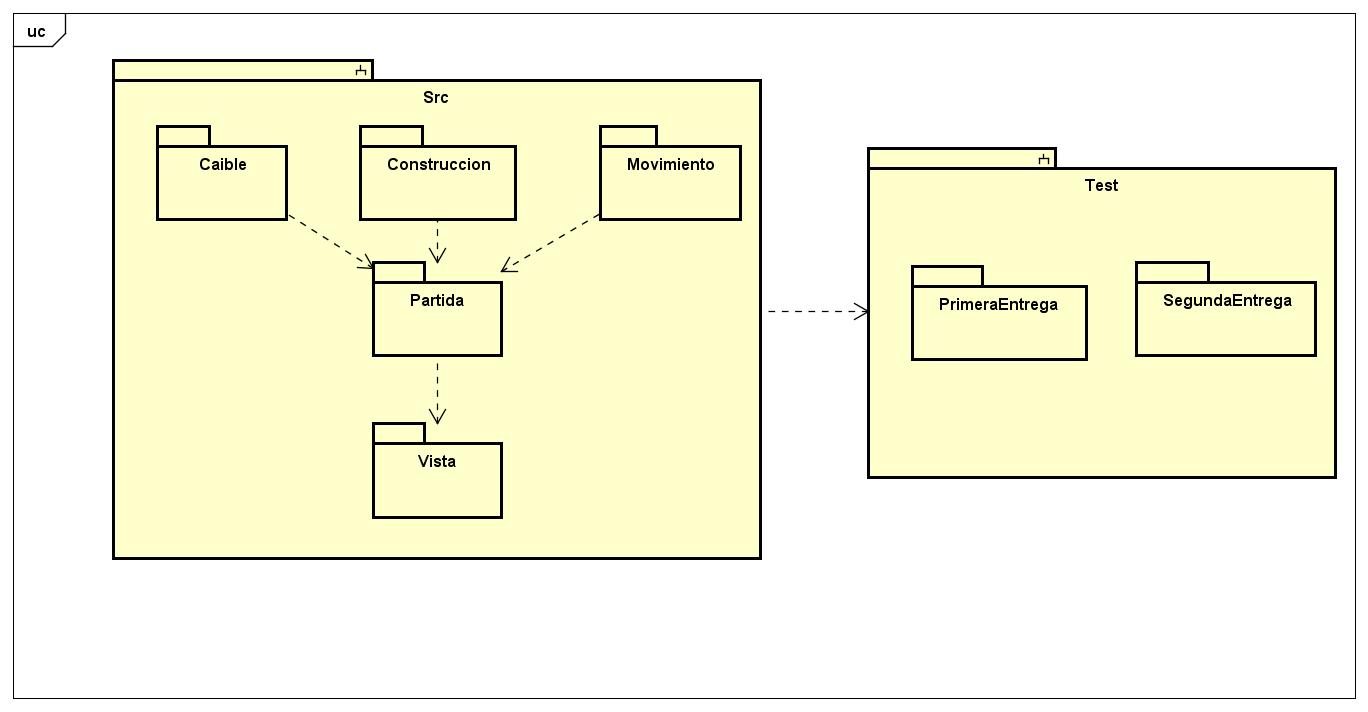


EstadoDuenio y su relación con Propiedades



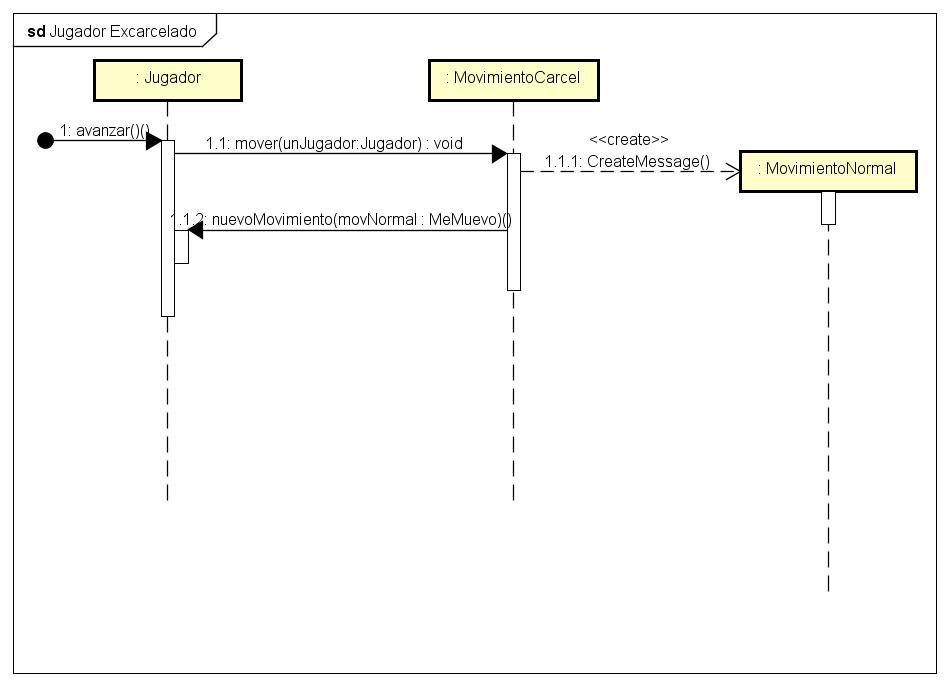
1. Diagrama de Paquetes

Conexión entre los paquetes del programa.

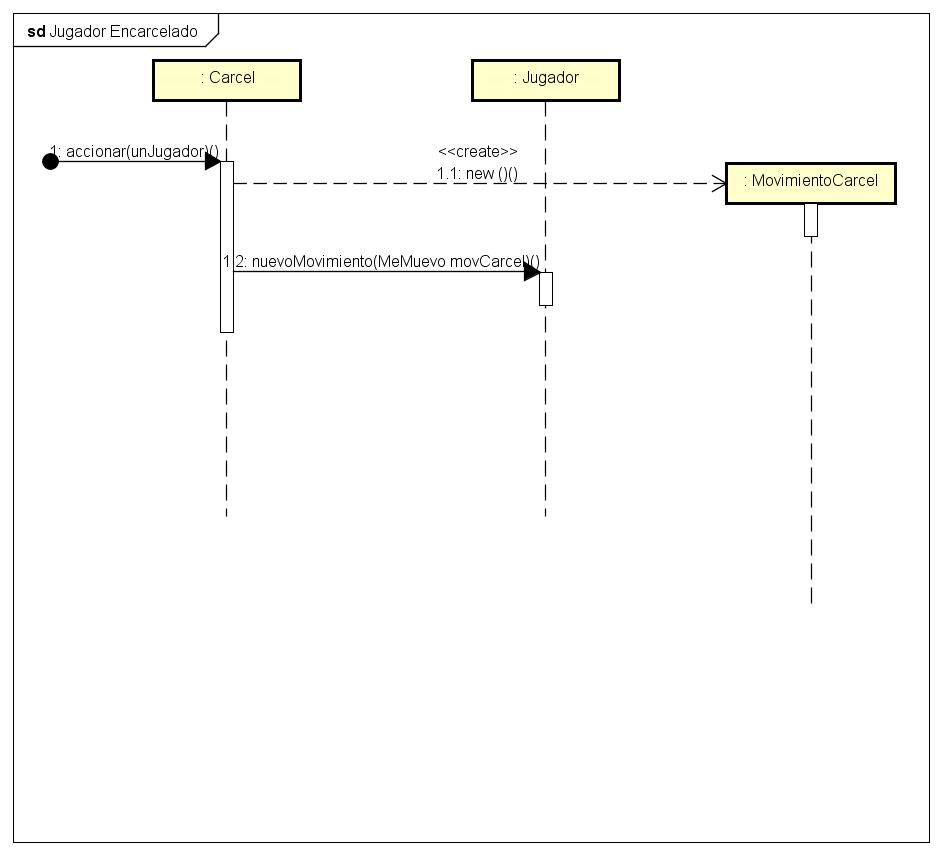


## Diagrama de secuencia

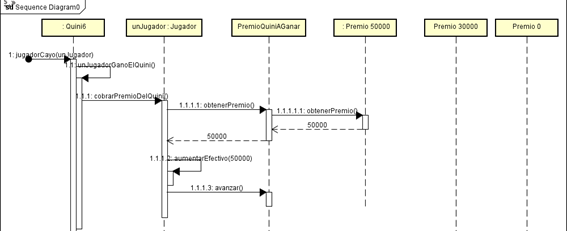
Luego de 4 turnos en la Cárcel, el jugador recupera su movilidad en el juego

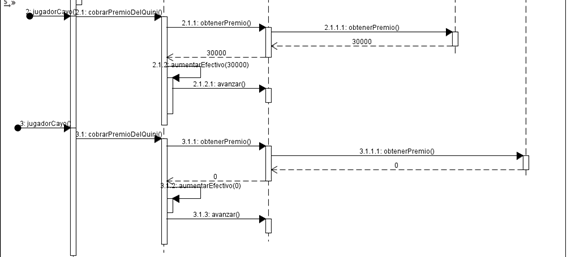


El Jugador es encarcelado y no puede avanzar

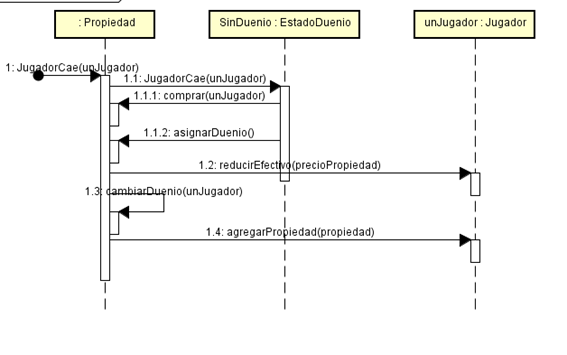


Jugador cae en quini6 3 veces

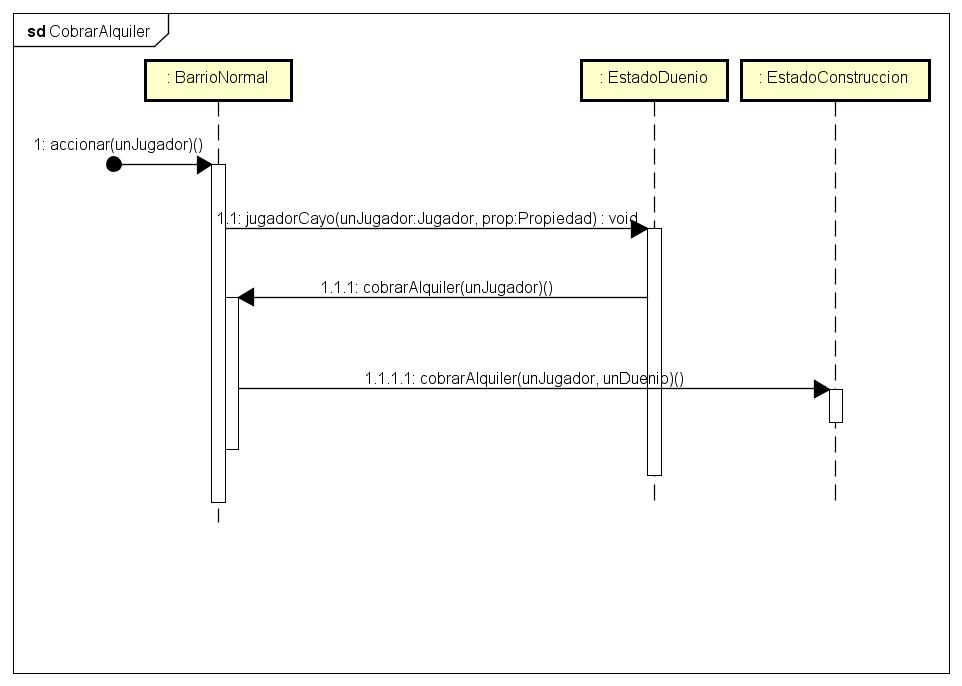




Jugador cae en una propiedad sin dueño y la compra. Él pasa a ser el dueño



Paso a paso, como se le cobra el alquiler a un jugador que cae en una propiedad ajena.



En el siguiente esquema se puede observar cómo funciona el chequeo de la habilitación para la construcción de un Hotel, en un BarrioNormal.

