FIUBA - 75.07

Algoritmos y programación III

Trabajo práctico 2: AlgoPoly

2do cuatrimestre, 2017

Alumnos:

Nombre	Padrón	Mail
Harfuch, Mateo	95049	mharfuch@gmail.com
Martin, Julian	97242	Julianmp09@gmail.com
Ricaldone, Matias	98519	matias.ricaldone@gmail.com
Zugna, Federico	95758	fede.zugna@gmail.com

Fecha de entrega final: 11 de Diciembre de 2017

Tutor:

Comentarios:

INFORME

SUPUESTOS

- El dinero de cada jugador siempre será un entero.
- Si el jugador al tirar los dados saca un doble (los dos dados valen lo mismo) y el jugador cae en la cárcel, éste no vuelve a tirar.
- Antes de intercambiar dos propiedades, las construcciones se demuelen.
- Si un jugador está en la cárcel y no dispone el dinero para pagar la fianza, no podrá vender ninguna propiedad en el caso que disponga una o más.

MODELO DE DOMINIO

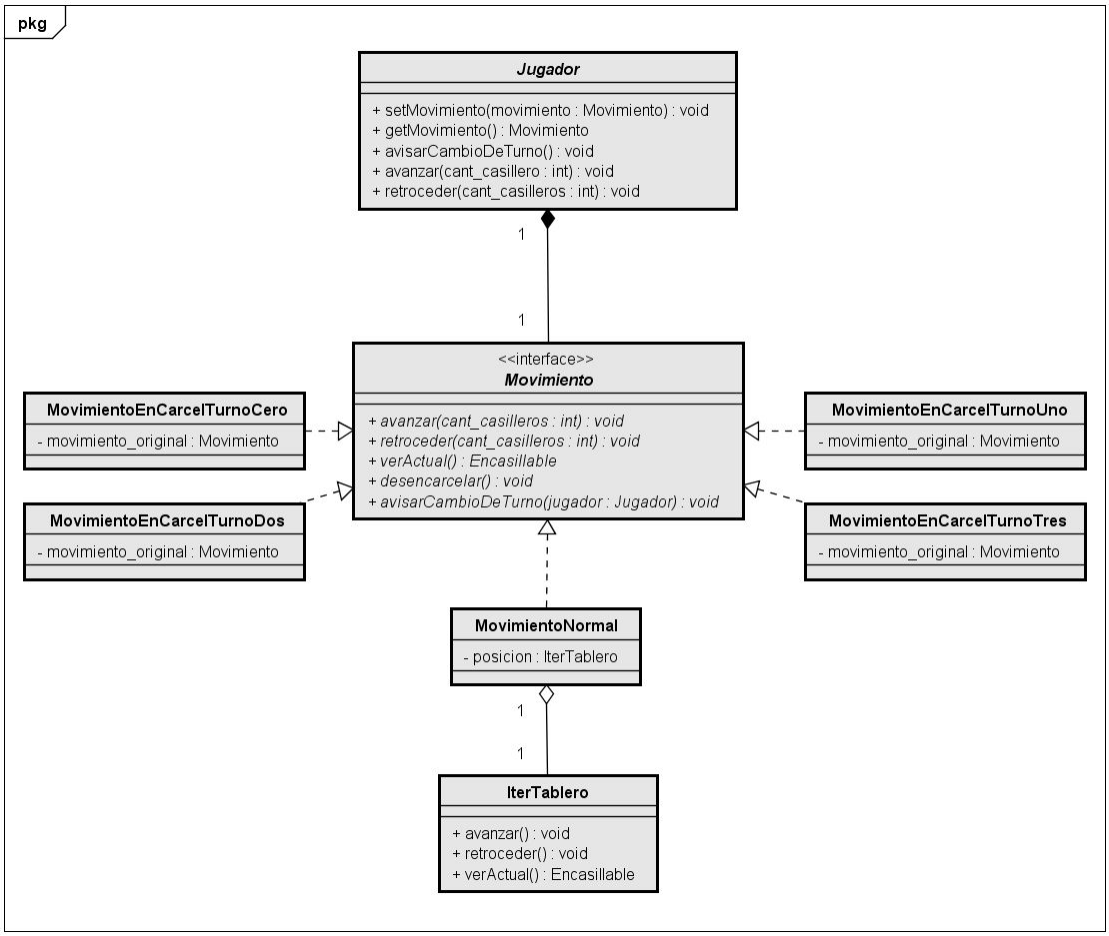
El juego consiste en un Tablero que tiene Casilleros con Encasillables (pueden ser Comprables o no) y con Jugadores, que en este caso serán 3 (tres). Cada Jugador comienza la partida con una suma de cien mil pesos ($100.000). Los Encasillables que no pueden comprarse como propiedad, tienen un efecto, el cual se aplica a cada jugador que al tirar los dados y avanzar quede situado en dicha Casilla. En cuanto a las propiedades que pueden ser compradas, tenemos dos clases: Servicios y Terreno. En los servicios se crean las clases de los distintos servicios y solo puede adquirir dicha propiedad y cobrar el impuesto correspondiente al jugador que se sitúe en él. En los terrenos se crean clases de los distritos donde se puede adquirir y construir casas y/o hoteles y donde se le cobrarán alquileres según corresponda.

DETALLES DE IMPLEMENTACIÓN - MODELO

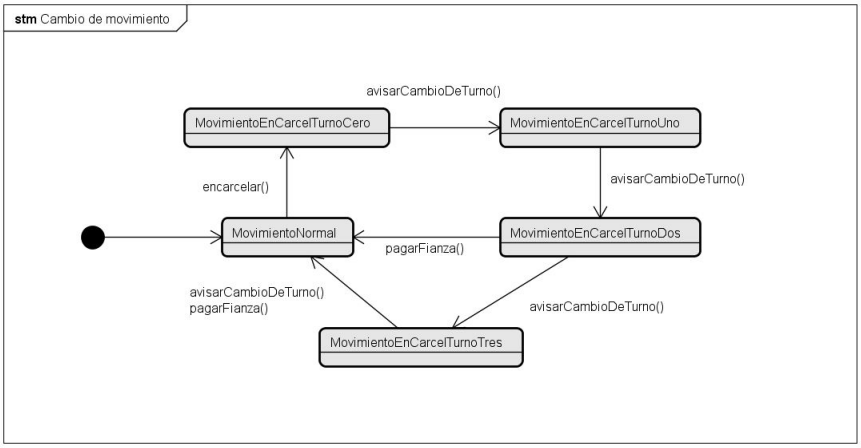
CÁRCEL Y POLICÍA

La cárcel tiene la responsabilidad de modificar el movimiento del jugador. Es decir, si el jugador cae en este casillero (aplica el efecto) no puede moverse por el tablero durante 3 turnos consecutivos a no ser que pague la fianza en el turno 2 o 3.

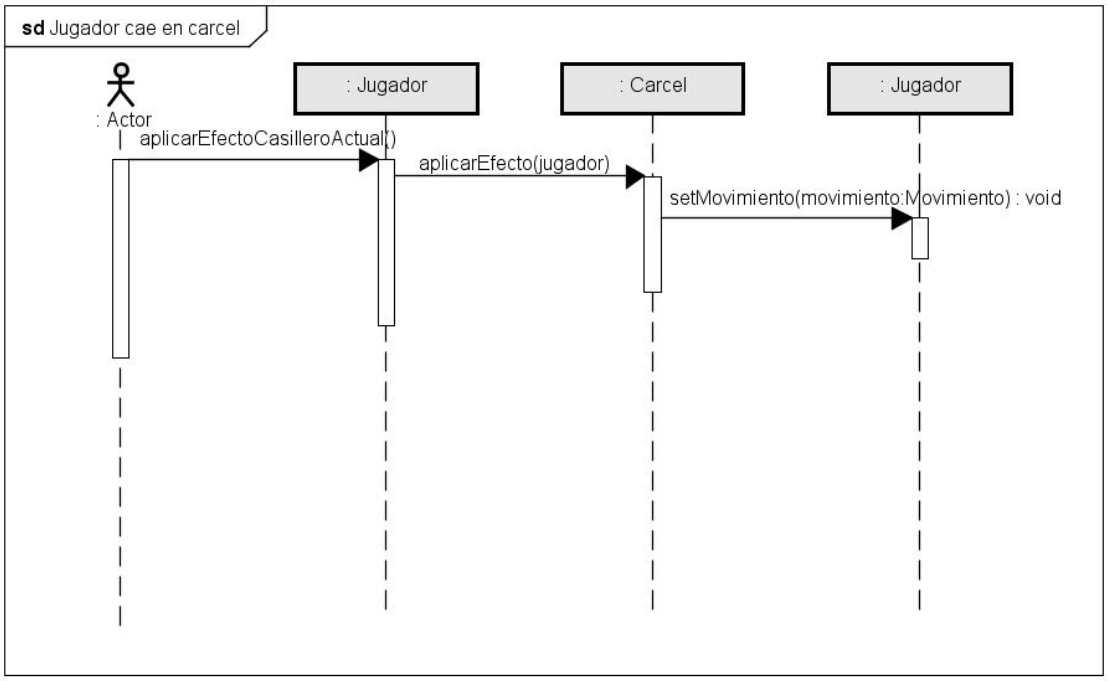
Para modelar esto se utilizó el patrón state donde el movimiento es un estado dentro del jugador que cambiará a medida que avanzan los turnos. Además el movimiento tiene la responsabilidad de avanzar sobre el tablero.

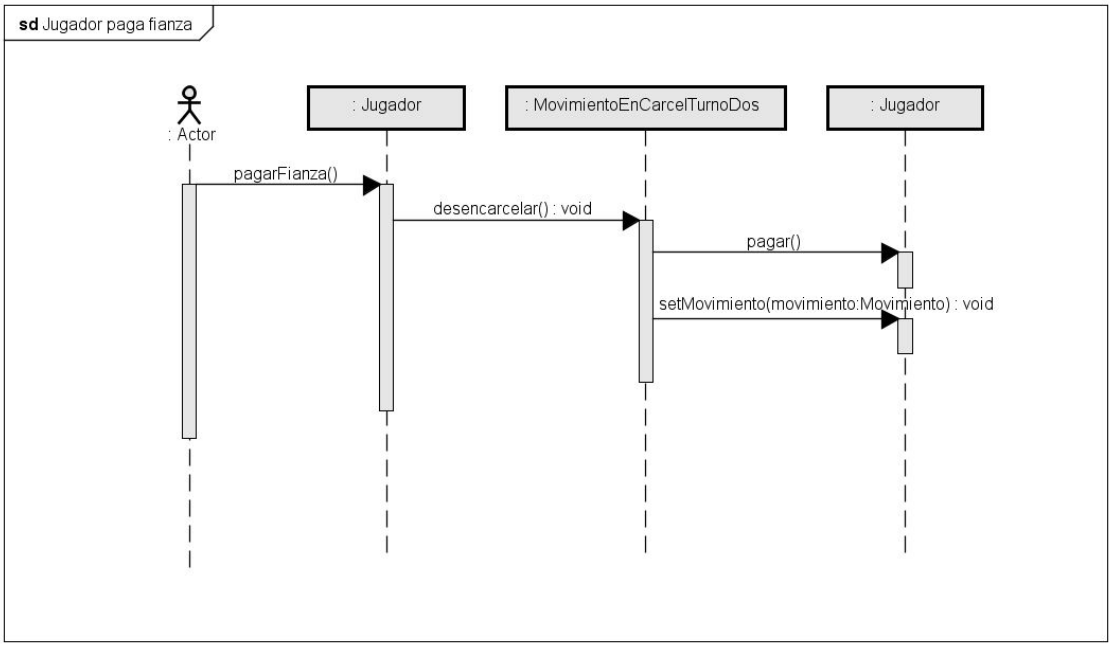


El movimiento normal contiene un iterador de tablero en el que delega todas las acciones de movimiento. Los movimientos en cárcel en general contienen un atributo donde conservan el estado del movimiento normal a fin de poder recuperarlo en caso de salir de la cárcel. En particular los turnos cero y uno responden con una excepción al intentar desencarcelar, pero los turnos dos y tres le cobran al jugador un monto de 45000 y restauran su movimiento. Otra forma de restaurar el movimiento es dejando que pasen los turnos. En cada cambio de turno se le debe avisar al jugador que debe cambiar su estado de movimiento. En el caso del movimiento normal nunca cambiará, pero en el caso de los turnos en cárcel siempre cambia el estado del movimiento al siguiente. Finalmente el movimiento en cárcel turno tres cambia el estado al normal nuevamente. El siguiente diagrama de estados representa todo esto:



Dicho todo esto podemos hablar del casillero cárcel. Una vez el jugador cae en la cárcel aplica el efecto. Este efecto cambia el estado de movimiento del jugador a cárcel turno cero y luego ocurre todo lo antes mencionado.



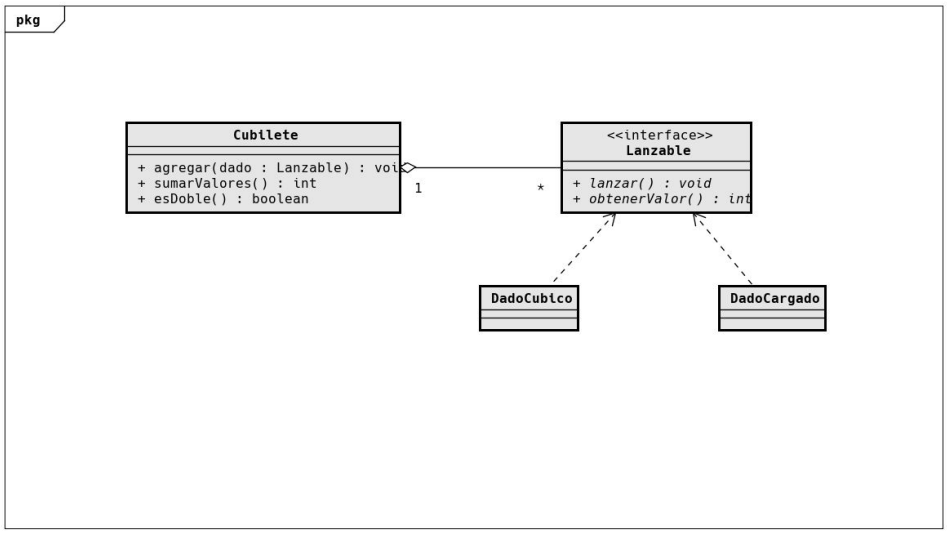


Adicionalmente existe un casillero policía que lleva automáticamente al jugador a la cárcel. Este casillero guarda una referencia al casillero de la cárcel (Esto ocurre dentro del factory), de manera que cuando se aplica el efecto del casillero policía automáticamente el jugador se mueve hacia la cárcel y se aplica el efecto de la cárcel.

DADOS Y CUBILETE

Los dados utilizados en un juego normal son dados de 6 caras (tienen 6 resultados posibles), siempre cumplen la invariante de mostrar un valor válido (números del 1 al 6) incluso desde su creación. Además poseen un método lanzar que cambia la cara (o valor mostrado) de manera aleatoria.

El cubilete almacena los dados, de esta forma nos abstraemos de la cantidad de dados que se utilizan en el juego, por lo tanto el cubilete posee métodos para agregar un dado, para sumar sus valores, para lanzarlos y para verificar que los mismos tengan el mismo valor.

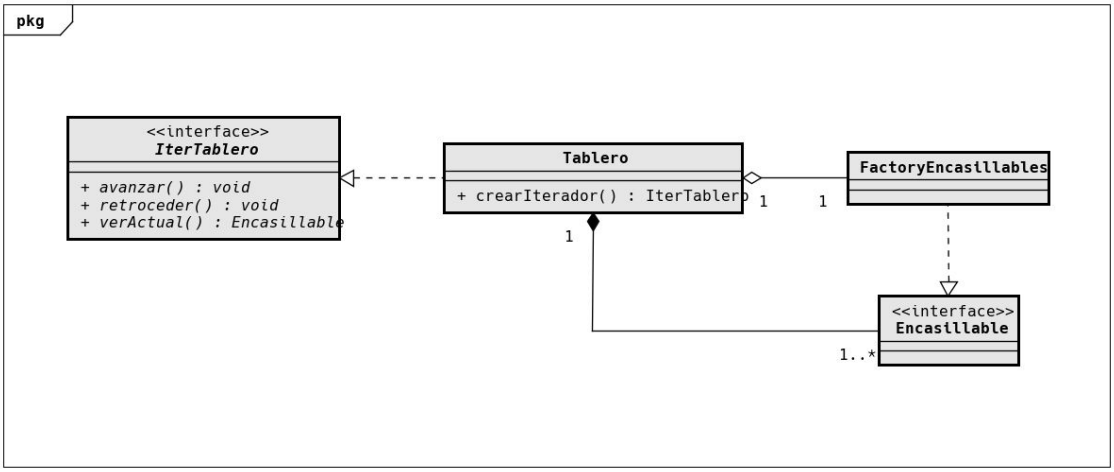


TABLERO

El tablero es responsable de almacenar los casilleros de manera ordenada. Además posee un iterador propio que permite recorrerlo. Este iterador es un *singleton* y es creado únicamente por el tablero.

Los casilleros también son *singleton*, pero cada uno tiene una forma particular de ser construido, por lo tanto la construcción de todos los casilleros están encapsulados en un *factory* que es responsable de crearlos respetando la particularidad de cada uno.

Dentro del *factory* encontramos encasillables que deben ser emparejados luego de construirlos y antes de ser entregados al usuario de la clase. Se debe destacar que el patrón *factory* permite un acceso global de los encasillables.



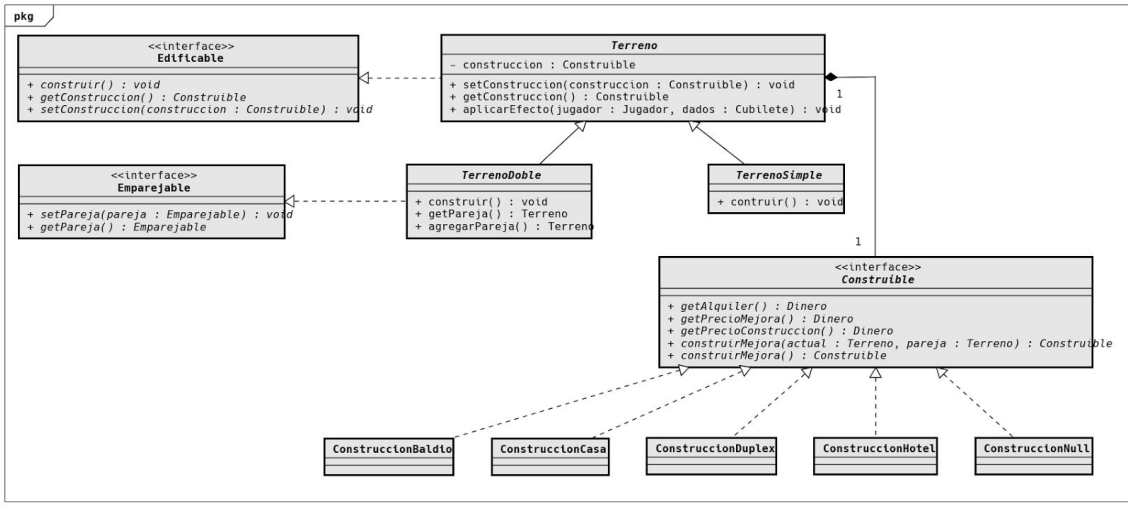
TERRENOS

Los terrenos son propiedades en las cuales se pueden construir casas u hoteles. El jugador que aplica el efecto sobre un terreno le pagará a su propietario un monto que se calcula en función de la construcción existente en ese terreno.

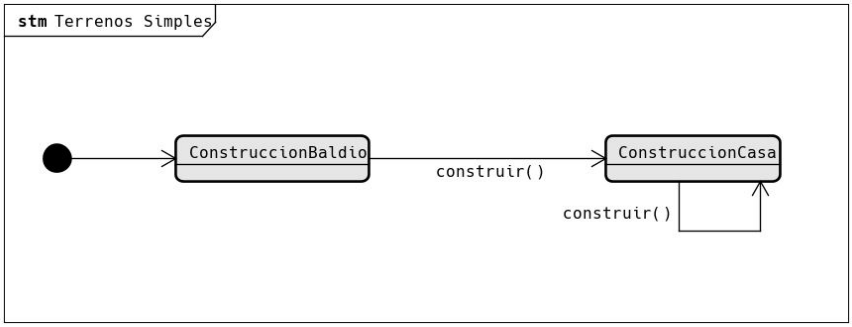
Las construcciones posibles son “Baldío”, “Casa”, “Duplex”, “Hotel”. Se observa fácilmente que estos se pueden modelar como estados del terreno (o mejor dicho estado de construcción) por lo tanto resulta trivial el uso del patrón “State”.

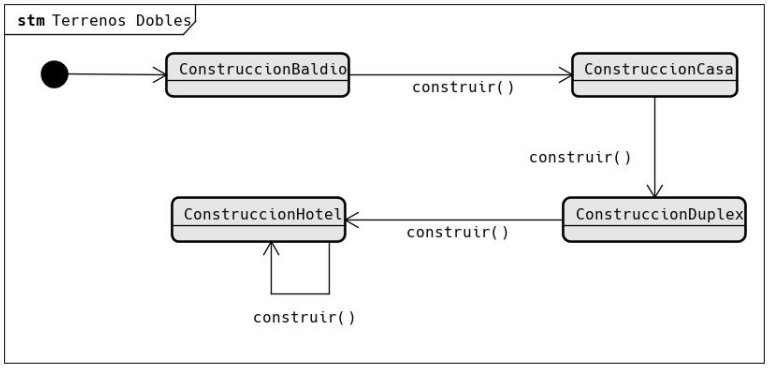
A los distintos estados de las construcciones los llamaremos construibles.

Adicionalmente cabe mencionar que existen dos tipos de terrenos, los simples y los emparejables. La diferencia sustancial entre estos es la validación en el momento de construir.



En el constructor de los terrenos se define la secuencia de construcciones que el terreno utilizará. Por ejemplo, para un terreno simple el baldío tiene un alquiler y conoce su próxima mejora que es la casa. La casa tiene un costo de construcción y no tiene próxima mejora. Sin embargo para los terrenos dobles la casa conoce a su próxima mejora que es el duplex y el duplex al hotel pero el hotel no tiene mejora.





Por otro lado para los terrenos dobles existen ciertas reglas para la construcción.

1. El propietario del terreno pareja debe ser el mismo para poder construir cualquier construible.

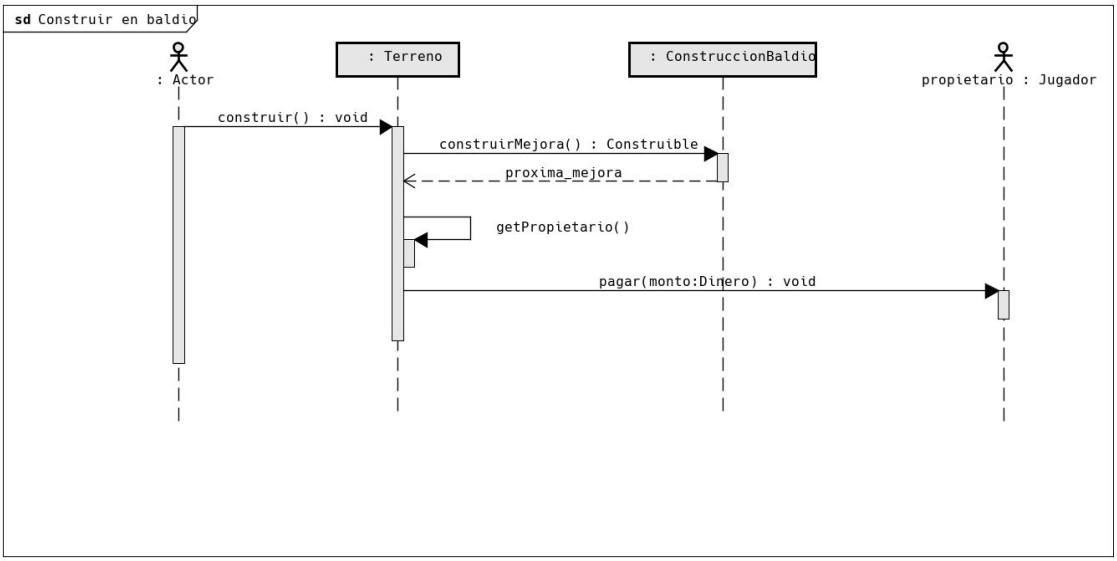
2. Para construir un hotel la pareja debe tener dúplex en ambos terrenos.

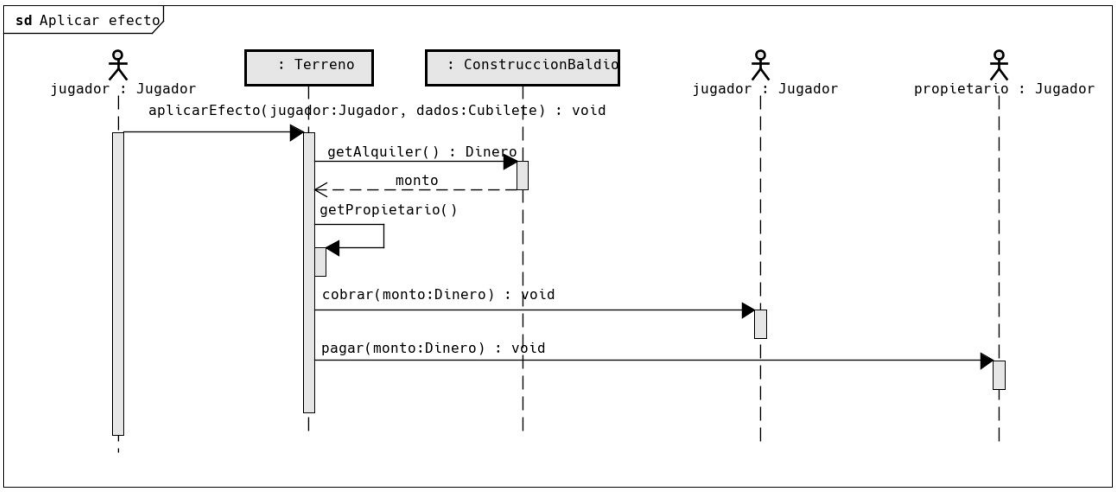
Esto puede puede ser modelado con excepciones. Para construir el terreno actual le “avisa” a su pareja que quiere construir, la pareja chequea que el propietario sea el mismo. Si el propietario no es el mismo lanzará una excepción.

Para construir el hotel cada terreno le avisa a su pareja que se desea construir un hotel. Si la construcción de la pareja es un baldío o una casa se lanza una excepción.

Para ayudar al modelado también se utilizó el patrón *null object*. Existe un construible que actúa como nulo y responde los mismos mensajes que cualquier otro construible pero con las siguientes particularidades:

- El precio de construcción y mejora es cero.
- El precio de alquiler es el precio de su antecesor.
- Construir siguiente mejora lanza una excepción.
- Si su pareja quiere construir dependerá de su antecesor.


sd Construir en baldio
: Actor
: Terreno
: ConstruccionBaldio
propietario : Jugador
construir() : void
construirMejora() : Construible
proxima_mejora
getPropietario()
pagar(monto:Dinero) : void

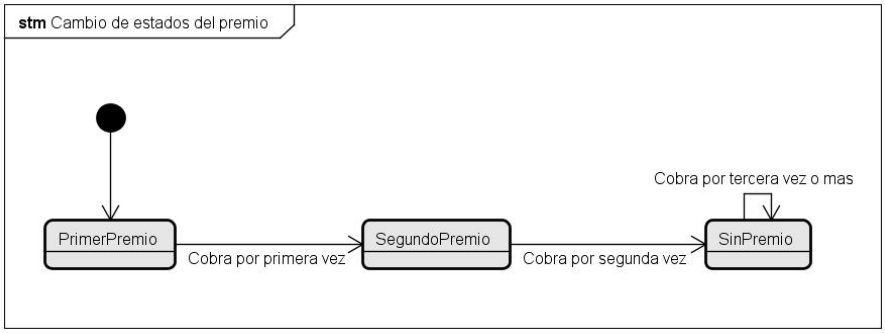

sd Aplicar efecto
jugador : Jugador
: Terreno
: ConstruccionBaldio
jugador : Jugador
propietario : Jugador
aplicarEfecto(jugador:Jugador, dados:Cubilete) : void
getAlquiler() : Dinero
monto
getPropietario()
cobrar(monto:Dinero) : void
pagar(monto:Dinero) : void

QUINI 6

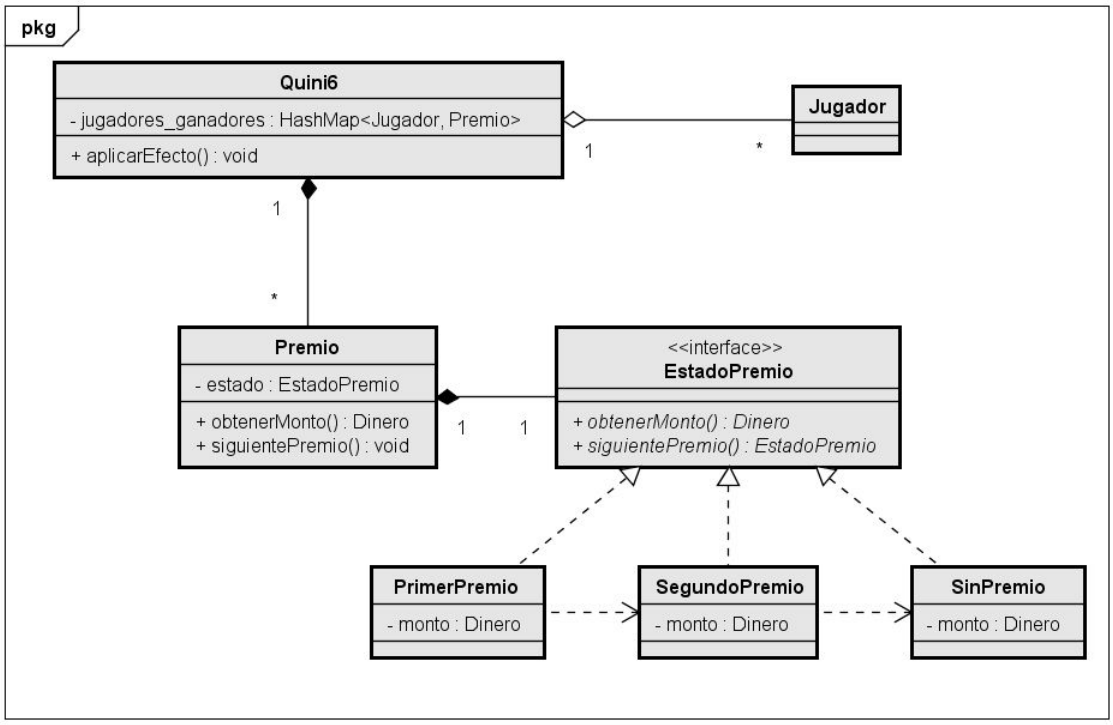
Para la solución a este requerimiento se utilizó el patrón “state” que permite que un objeto altere completamente su comportamiento en función de un cambio de estado de manera de comportarse como una clase completamente diferente.

Entonces, enfocándonos en el problema, necesitamos que el (casillero) Quini6 entregue un premio diferente según el jugador haya caído en el casillero una, dos o más veces.

Por lo tanto el objeto (casillero) Quini6 conoce a los jugadores que cayeron en él y le asigna un premio a cada uno de ellos. Si el jugador cae por primera vez el Premio se comportará como un *“PrimerPremio”*, si cae por segunda vez se comportara como un *“SegundoPremio”* y a la tercera vez se comportará como un *“SinPremio”*.

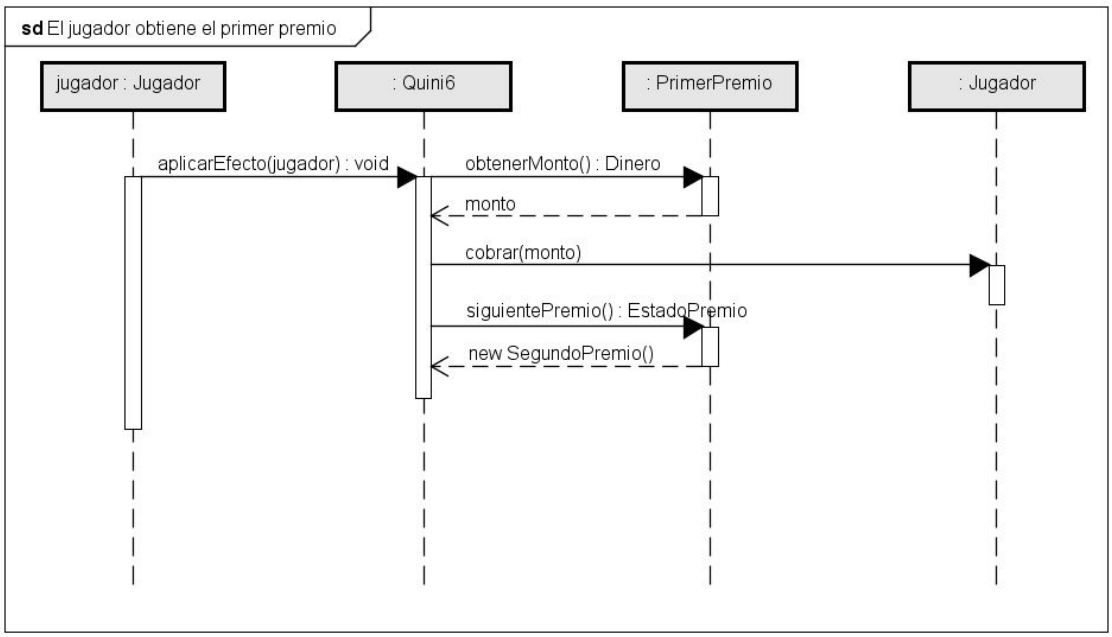


Entonces, siguiendo el patrón elegido, se creó una clase *“Premio”* que contiene un *“EstadoPremio”*. El *“EstadoPremio”* es una interfaz que es implementada por las clases concretas *“PrimerPremio”*, *“SegundoPremio”*, *“SinPremio”*.



Cada estado de premio es capaz de generar al siguiente premio. Por ejemplo, el primer premio tiene un método concreto que crea al segundo y el segundo al sin premio.

Entonces cada jugador al cobrar el primer premio llama al siguiente premio. El último premio en instanciarse es “SinPremio”. Para esta clase, el método siguiente premio devolverá una referencia a sí mismo.

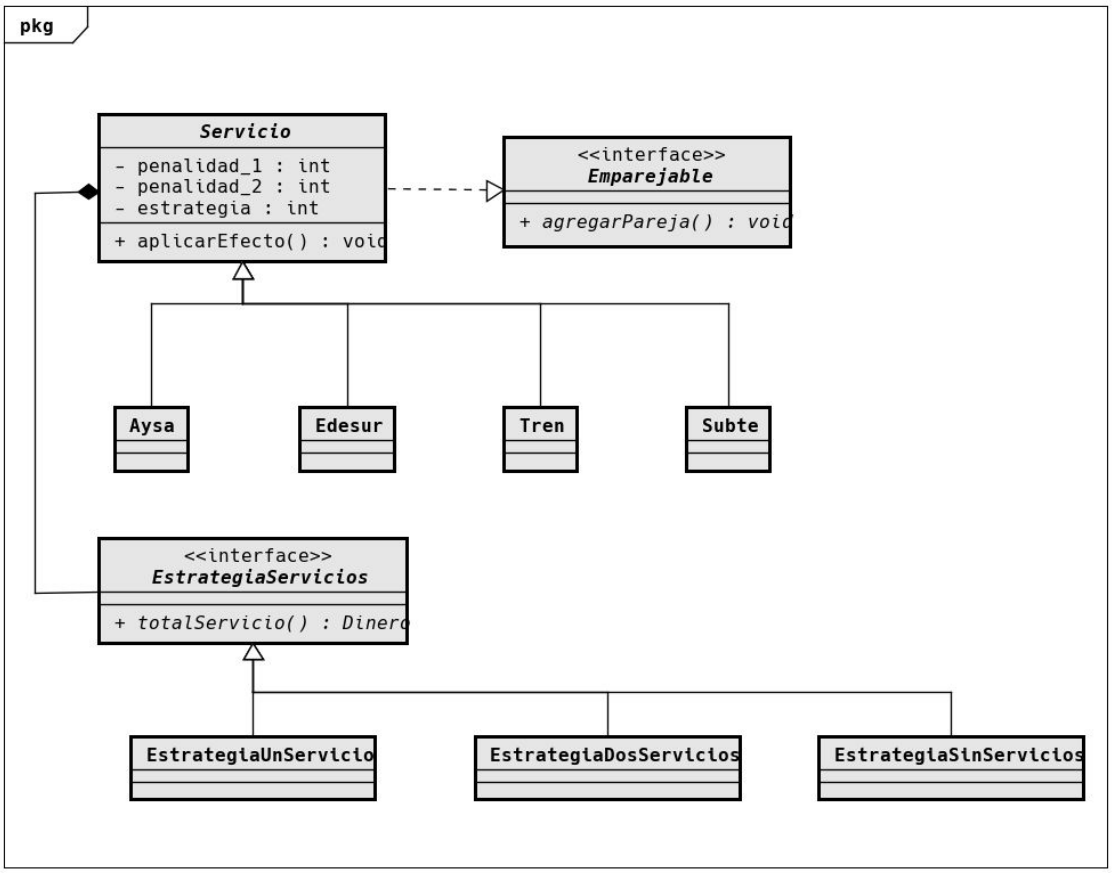


SERVICIOS

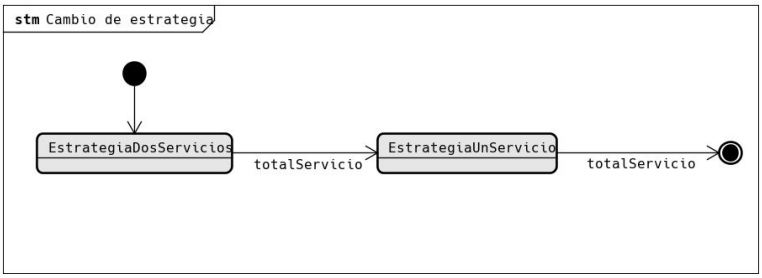
Un jugador, al caer en un servicio que tiene propietario, debe pagar un monto relativo al número que obtuvo en los dados. Además el monto también dependerá de si el propietario también posee un servicio hermano.

Entonces, los servicios son un tipo de propiedad en los que no se puede construir pero que sí pueden ser emparejados. Con emparejar nos referimos a que un servicio puede tener otro servicio hermano como pareja.

Por lo tanto se creó la clase abstracta "S*ervicio"* que contiene la lógica para aplicar el efecto relativo al cobro de la penalidad al jugador que cae en ese casillero (*servicio*). Además implementa la interfaz *"Emparejable"* que provee al método necesario para agregar a su casillero pareja. Este casillero pareja puede ser cualquier casillero que pueda tener un propietario por lo tanto es del tipo "*Propiedad*".

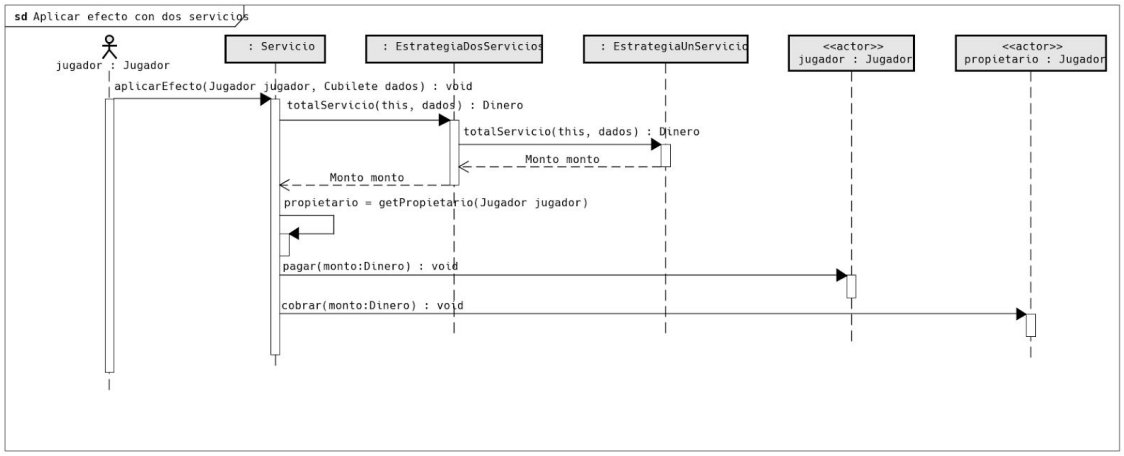


Para cobrar el servicio se solicita el propietario a su pareja. Si el propietario del servicio coincide con el propietario de su pareja se ejecuta el cálculo con dos servicios. Sino se realiza el cálculo con un único servicio. Si el propietario de este servicio no existe entonces no se realiza ningún cobro al jugador.



Esta lógica está modelada a partir del patrón “strategy”. Para esto se creó la interfaz “EstrategiaCalculoServicio” de la cual derivan las distintas estrategias de cálculo.

Cada estrategia almacena información sobre cuánto debe cobrarse a un jugador que cae en el casillero “*Servicio*” y además conoce a la siguiente estrategia que será aplicada si esta falla. De esta manera queda modelado un *Switch* pero con objetos.



EXCEPCIONES

Se crearon las siguientes excepciones:

BancaRotaException: esta excepción se creó para manejar el hecho que el jugador no dispone de dinero para pagar un alquiler, comprar una propiedad o pagar la fianza de la cárcel y tampoco dispone de ninguna propiedad para vender o la venta de las propiedades no alcanza para los costos afrontados.

DineroInsuficienteException: esta excepción se creó para manejar el hecho que el jugador no dispone del dinero para comprar una propiedad, pagar la fianza de la cárcel.

DineroNegativoException: esta excepción hereda de RUNTIME porque solo debería suceder si se crea un objeto Dinero con saldo negativo.

FaltaAdquirirParejaException: esta excepción se creó para manejar el hecho que no podrá construir un jugador en las propiedades dobles (que contienen Norte y Sur) al no ser dueño de las dos.

FaltanCasasException: esta excepción está para el caso que se quiera construir un hotel y no estén todas las casas de cada barrio construidas.

LimiteDeConstruccionesException:

MaximoDeConstruccionesAlcanzadoException:

NoHayConstruccionesParaDemolerException: esta excepción está para manejar el hecho que se quiera vender o intercambiar una propiedad y no tenga ninguna construcción hecha.

NoHayJugadoresException: esta excepción se creó para manejar los turnos y el jugador actual o el siguiente.

NoHayMasMejorasExcepion: esta excepción está para manejar el hecho de las construcciones. Después de construir un hotel, no se puede construir mas nada.

NoPuedePagarFianzaException: esta excepción está para manejar el hecho que el jugador no dispone del dinero cuando está encarcelado.

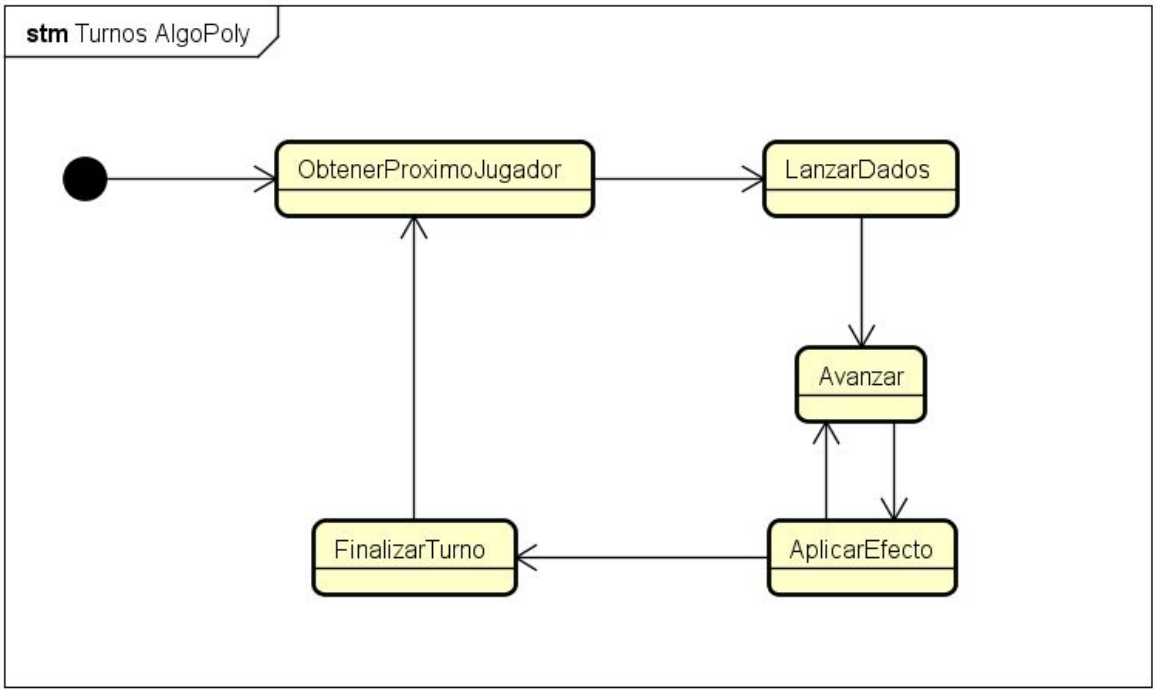
YaTieneParejaException: esta excepción está para manejar el hecho del casillero si tiene la pareja (Norte y sur) o no.

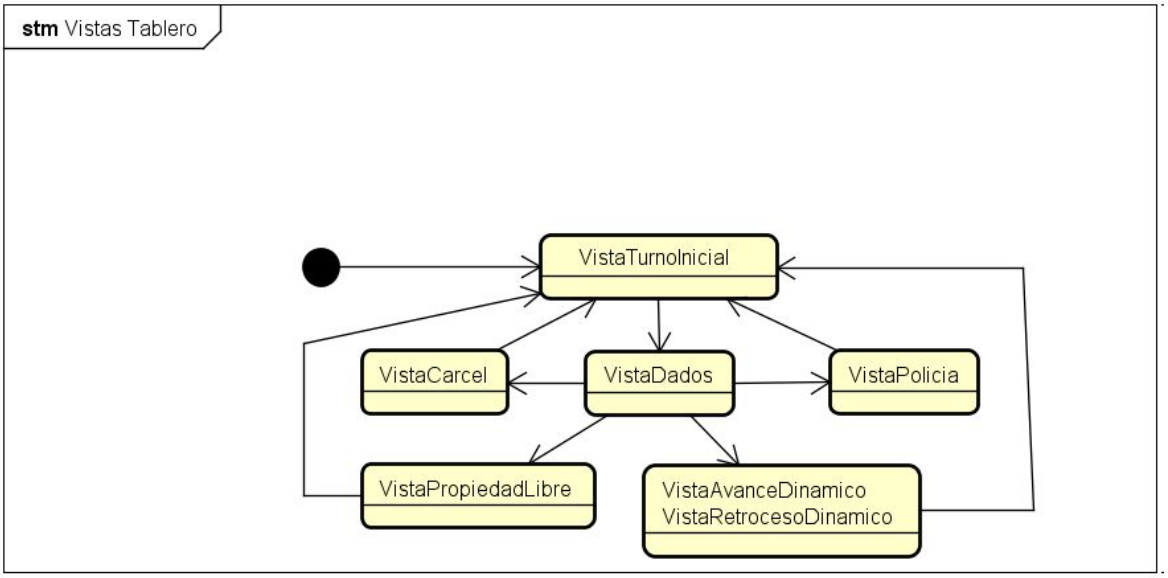
YaTienePropietarioException: esta excepción está para saber si una propiedad tiene dueño o no.

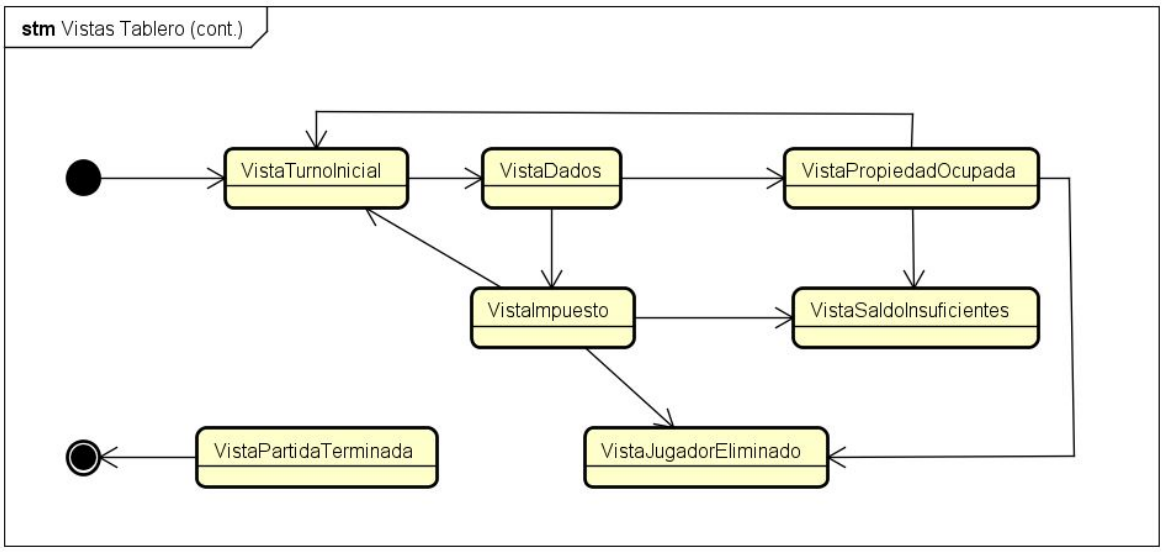
DETALLES DE IMPLEMENTACIÓN - GUI

Para el desarrollo de la interfaz de usuario se decidió utilizar las librerías de JavaFX. A grandes rasgos, se buscó aplicar el patrón de diseño MVC para lograr la mayor separación posible entre el modelo y la representación gráfica que se eligió. No se realizaron pruebas automáticas de interfaz de usuario, priorizando las pruebas manuales de verificación de jugabilidad. A continuación, un breve resumen de los detalles más importantes:

- Excepciones: los controladores se encargan de manejar las excepciones que reciben del modelo para mostrar en pantalla los mensajes adecuados en cada caso.
- Patrón Observer: tanto para mostrar la información de los jugadores como de los casilleros, se decidió usar el patrón Observer a través de la implementación que Java proporciona con la clase Observable y la interfaz Observer. De esta forma, los controladores actualizan las vistas individuales cada vez que:
    - Jugador: cambia su dinero
    - Propiedad: cambia el dueño
    - Construible: se construye o demuele algo


stm Turnos AlgoPoly
ObtenerProximoJugador
LanzarDados
Avanzar
FinalizarTurno
AplicarEfecto


stm Vistas Tablero
VistaTurnoInicial
VistaCarcel
VistaDados
VistaPolicia
VistaPropiedadLibre
VistaAvanceDinamico
VistaRetrocesoDinamico



VISTAS

[Pantalla Inicial]

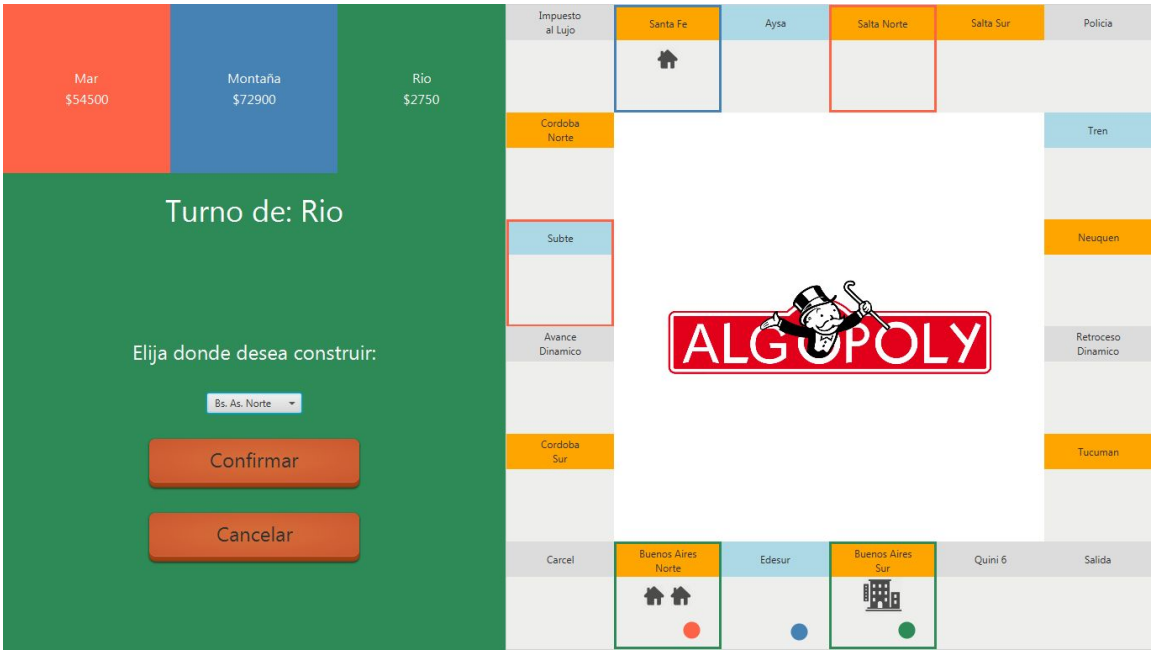
[Pantalla Nueva Partida]

[Pantalla Partida]

[Pantalla Tirar Dados]

[Pantalla Efecto Impuesto al Lujo]

[Pantalla Propiedad Libre]



[Pantalla Propiedades con dueño y construidas]