Trabajo Práctico 2 — Algopoly

[7507/9502] Algoritmos y Programación III
Curso 1
Segundo cuatrimestre de 2017

Alumnos:	Aquino, Maria ; Bravo, Facundo
Números de padrón:	99871; 100151

Índice

1. Introducción **2**

2. Supuestos **2**

3. Modelo de dominio **2**

4. Diagramas de clase **3**

5. Detalles de implementación **6**
5.1. Utilización de patrón Strategy en Avance Dinámico y Retroceso Dinámico 6
5.2. Utilización de patrón State en Propiedad 6
5.3. Implementación de acciones posibles 7

6. Excepciones **7**

7. Diagramas de secuencia **8**

1. Introducción

El presente informe reúne la documentación de la solución del último trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar Algopoly, un juego similar al conocido Monopoly, en java.

2. Supuestos

Se adoptaron los siguientes supuestos para el desarrollo de la aplicación:

-Un jugador al caer en un barrio o compañía, la compra automáticamente.

-En caso de que el avance para avance dinámico dé negativo, el jugador retrocederá, análogamente pasará lo mismo con retroceso dinámico.

-Un jugador vende sus propiedades al juego y no a otro jugador.

3. Modelo de dominio

Las principales clases que conforman la aplicación son Juego y Jugador. Juego contiene los jugadores activos, los dados oficiales, los turnos, y el tablero, mientras que Jugador representa a un jugador que realiza las acciones que esta habilitado a realizar en un momento dado.

Los dados se pueden definir en un rango arbitrario, en este caso Juego utiliza dados de 1 a 6, que a su vez los agrupa en una tirada de dos dados.

Para el manejo de los turnos, Juego utiliza un objeto Turno que contiene a los jugadores, que decide quién es el jugador actual.

El tablero, al que se puede acceder solo a través de Juego, contiene a los casilleros, establece sus posiciones, y también los grupos de barrios y compañías.

Casillero es una clase abstracta de la que heredan todos los casilleros, contiene la posición y obliga a las subclases a implementar el método äccionar"que se encarga de aplicar los efectos de un casillero específico, cuando un jugador cae sobre el mismo.

Hay diferentes tipos de casilleros, uno de ellos son las propiedades, englobadas en la clase abstracta Propiedad que define a los casilleros adquiribles por un jugador, y cuya acción es ser comprado cuando no tiene dueño, o cobrar un monto a favor del dueño cuando lo tiene, estos casilleros son subclases de Barrio y Compañía. Los barrios modifican el monto a cobrar según las edificaciones que posean ,mientras que las compañías lo hacen de acuerdo a la tirada del jugador no dueño que cayó en dicho casillero sumado a si el propietario tiene el grupo de compañías completo (en este juego cada compañía tiene otra aparejada) . Se agrega como detalle de implementación que se creó una clase concreta por cada barrio y compañía : Buenos Aires, Santa Fe, Aysa, etc.

Cada jugador tiene un conjunto de acciones posibles en un momento dado, que definen qué puede hacerse en cada instante del juego. Cuando no es el turno de un determinado jugador, su conjunto de acciones posibles esta vacío.

La dinámica del juego de desarrolla pidiéndole al juego el jugador actual, preguntarle sus acciones posibles y ejecutando alguna de ellas, repitiendo este proceso de pregunta y ejecución de acciones hasta que se ejecute la acción de pasar turno, momento en el que se deberá volver a pedir el jugador actual a juego y repetir todo el proceso. Este ciclo termina cuando solo queda un jugador, coincidiendo con el fin del juego.

4. Diagramas de clase

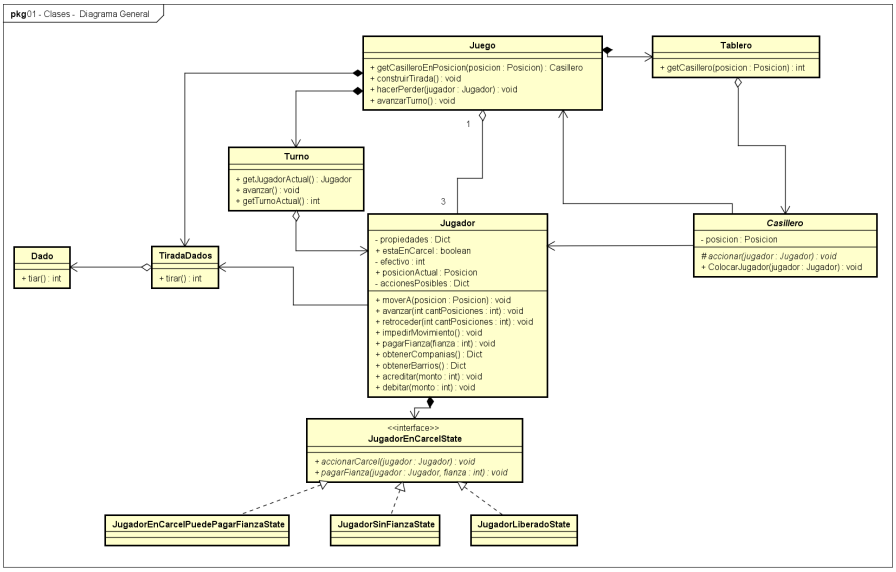

Figura 1: Diagrama general del juego Algopoly

En este diagrama (Figura 1) se detalla el diseño del juego en su totalidad. A Juego se le agregan instancias de Jugador y a éstas se les da una referencia de TiradaDados, ya que Juego es quien maneja las reglas de la tirada.Por otra parte, Jugador debe poseer una referencia de Juego,ya que cuando un jugador debe desplazarse una cantidad de posiciones determinada por la tirada de dados, lo debe hacer a través de Juego porque es quien conoce al tablero y obtiene los casilleros por su posición dentro de este último. Una de las responsabilidades importantes de Juego es la de avanzar turnos y obtener el jugador en el turno actual, para lograr esto delega la responsabilidad en la clase Turno.

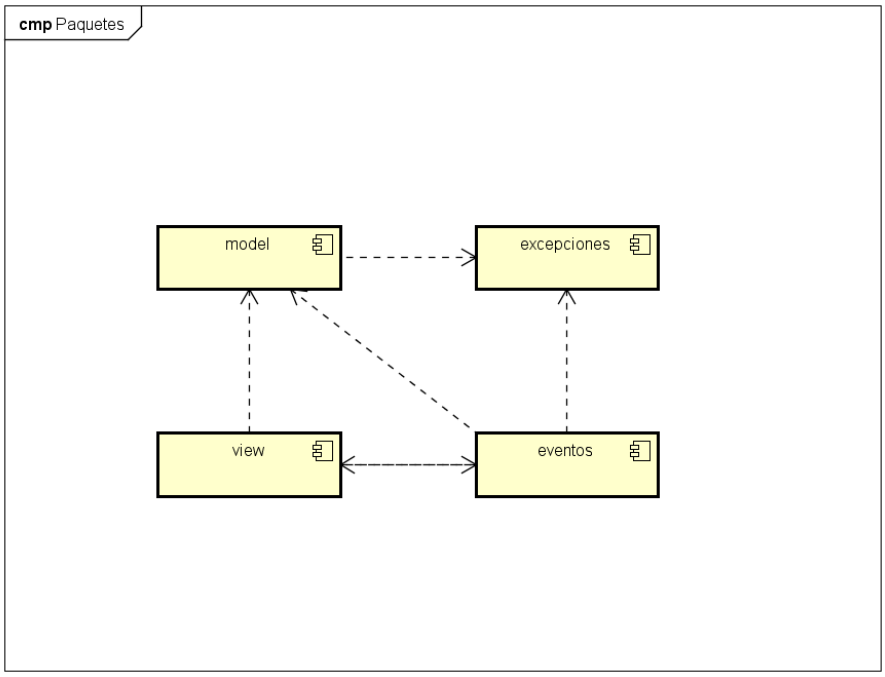

Figura 2: Diagrama de paquetes

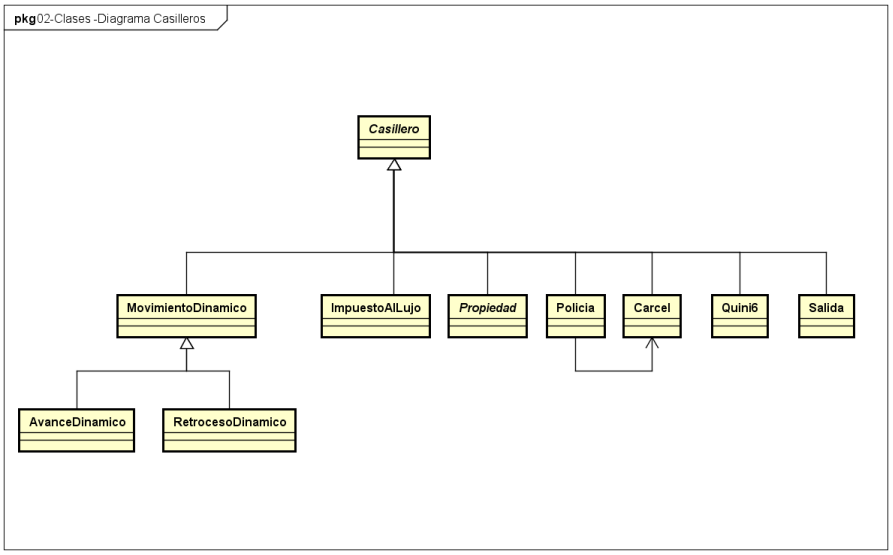

Figura 3: Diagrama general de los casilleros

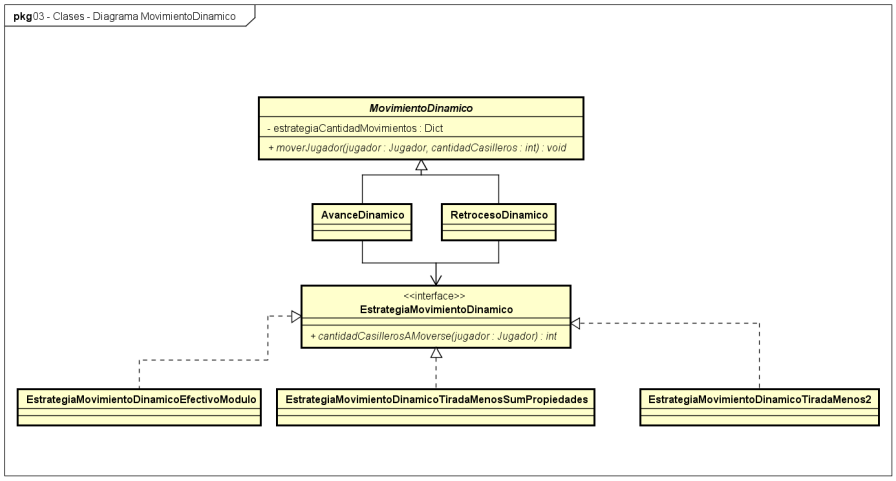

Figura 4: Diagrama de MovimientoDinamico

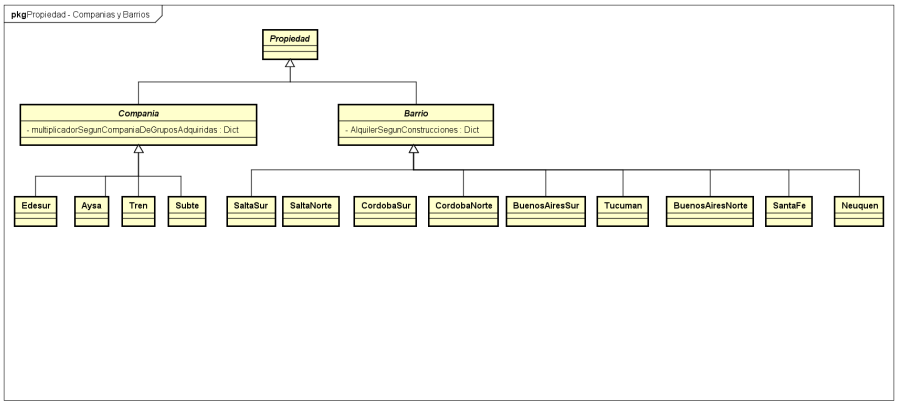

Figura 5: Diagrama general de Propiedades

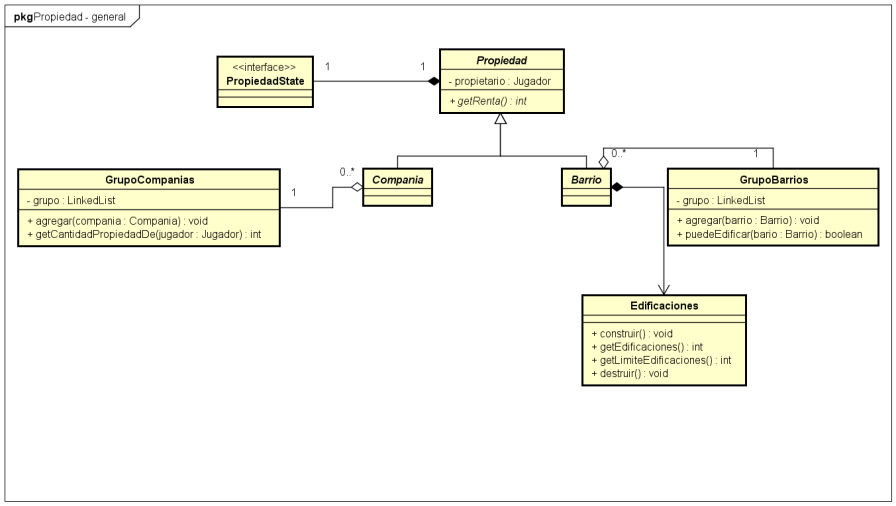

Figura 6: Diagrama de propiedades más detallado

Este último diagrama de clases (Figura 6) logra dar un entendimiento más cabal de como se trataron a los barrios y a las compañías: ambas son instancias de las clases abstractas Barrio y Compania respectivamente, pero además son subclases de Propiedad por el hecho de que comparten la misma transición de estados : sin propietario - con propietario y viceversa, además de aplicar el mismo accionar sobre el jugador que cae en ellas. Barrio y Compania tienen la referencia a una clase que contiene al grupo al que pertenecen y que es quien conoce cuáles son las propiedades asociadas. En el caso de Barrio, la clase GrupoBarrios se encarga de permitir la edificación mientras que la clase Edificaciones se encarga de tener el número total de edificaciones de Barrio y el valor de alquiler/venta de acuerdo al número de éstas.

5. Detalles de implementación

5.1. Utilización de patrón Strategy en Avance Dinámico y Retroceso Dinámico

En los casilleros AvanceDinamico y RetrocesoDinamico se utilizan las mismas estrategias de las cuales una de ellas es la elegida en tiempo de ejecución para aplicar un desplazamiento al jugador en el tablero, por lo tanto se decidió la utilización del patrón Strategy. En la superclase abstracta MovimientoDinamico , se tiene una instancia de Hashtable en el cual se cargan las estrategias según el numero de la tirada de dados, y también hay un método abstracto "moverJugador"que AvanceDinamico implementa utilizando el método .ªvanzar"de Jugador, mientras que RetrocesoDinamico lo implementa utilizando el método retroceder"de Jugador.

```
public abstract class MovimientoDinamico extends Casillero {

protected Hashtable<Integer, EstrategiaMovimientoDinamico> estrategiaCantidadMovimientos ;

public MovimientoDinamico(Posicion posicion) {
        ....
}

@Override
protected void accionar(Jugador jugador) {
...
int cantCasillerosAMoverse = this.estrategiaCantidadMovimientos.get(jugador.getUltimaTiradaDados(
        ...
this.moverJugador(jugador, cantCasillerosAMoverse);
}

public abstract void moverJugador(Jugador jugador, int cantCasilleros);
}
```

En cuanto a las estrategias, se menciona que cada una de ellas son una implementación de la interfaz EstrategiaMovimientoDinamico

```
public interface EstrategiaMovimientoDinamico {
int cantidadCasillerosAMoverse(Jugador jugador);
}
```

5.2. Utilización de patrón State en Propiedad

La clase Propiedad implementa un estado, siguiendo el patrón de diseño State, que decide su comportamiento dependiendo de si tiene o no dueño. Si no tiene dueño, la acción de la propiedad

pregunta al jugador que la accionó si quiere comprarla. Si tiene dueño, cobra la renta, cuando corresponde.

5.3. Implementación de acciones posibles

Las acciones posibles de cada jugador fueron implementadas como un Hashtable<String, Comando>, siendo comando una interfaz con un único método, ejecutar(). Las acciones implementadas ejecutan su acción con dicho método y son:

- TirarDadosYAvanzarComando
- TerminarTurnoComando
- PagarFianzaComando
- VenderPropiedadComando
- ComprarPropiedadComando
- EdificarComando

Todos los comandos son cargados en su creación con los parámetros necesarios (jugador al que afectan, propiedad que se compra/vende/edifica)

Jugador tiene un conjunto de acciones para cuando inicia el turno y otro para después de tirar los dados. Una vez terminado el turno, su conjunto de acciones esta vacío., hasta el inicio del siguiente turno del jugador. Pueden ser agregadas a jugador acciones externamente que desaparecen cuando termina el turno. También pueden ser agregadas y quitadas acciones extra para que jugador tenga disponible al inicio de cada turno.

6. Excepciones

Se implementaron las siguientes excepciones autoexplicativas:

- BarrioDobleNoPuedeEdificarHotelSinHaberEdificadoMaximaCantidadDeCasasException
- BarrioDobleNoPuedeEdificarSiPropietarioTieneUnaDeLasDosPropieadesException
- BarrioLimiteEdificacionesException
- JugadorEfectivoInsuficienteException
- JugadorLiberadoIntentaPagarFianzaException
- JugadorNoPuedePagarFianzaException
- NoPuedeVendersePropiedadSinDuenioException

7. Diagramas de secuencia

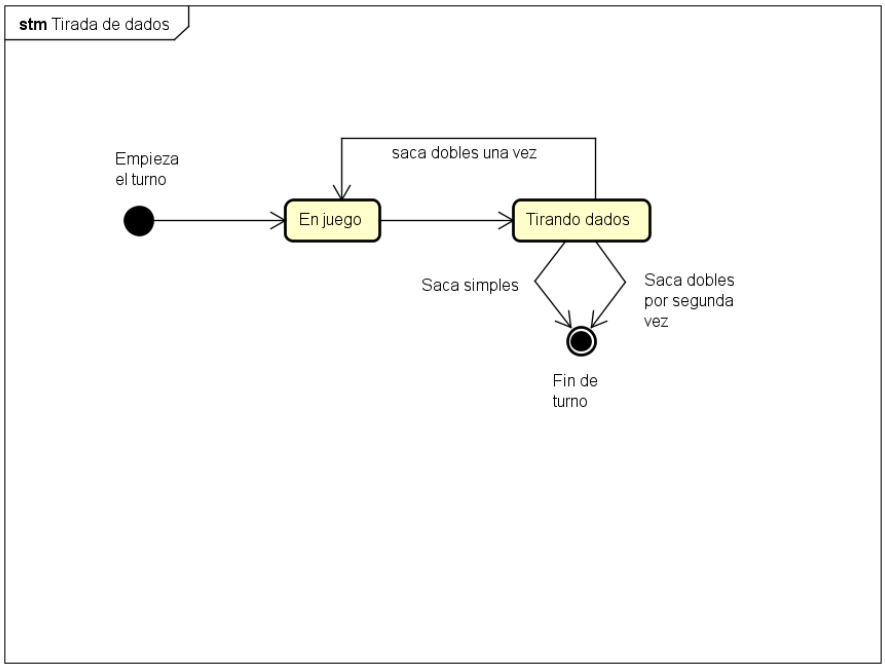

Figura 7: Tirada de dados

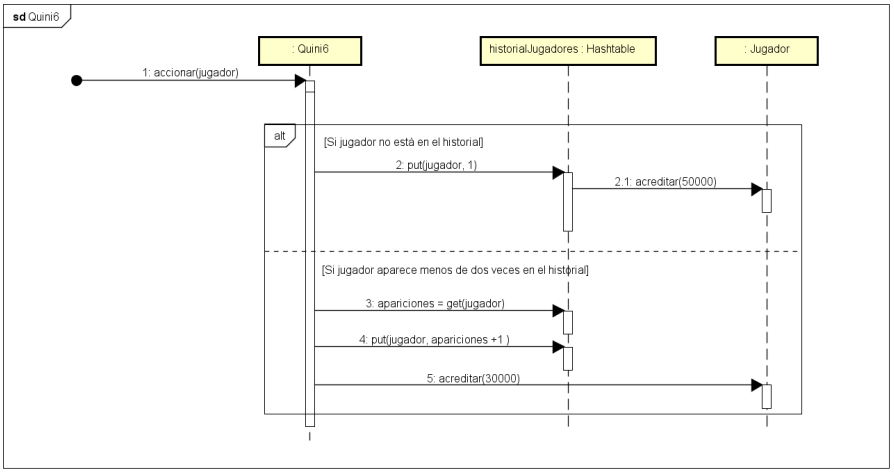

Figura 8: Caer en Quini6

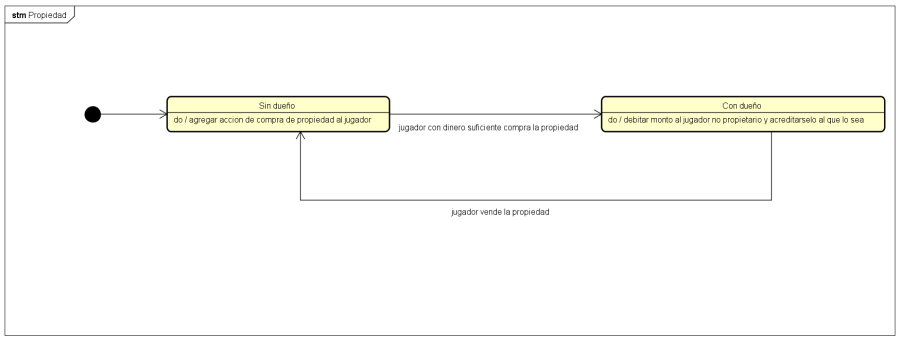

Figura 9: Diagrama de estado de las propiedades

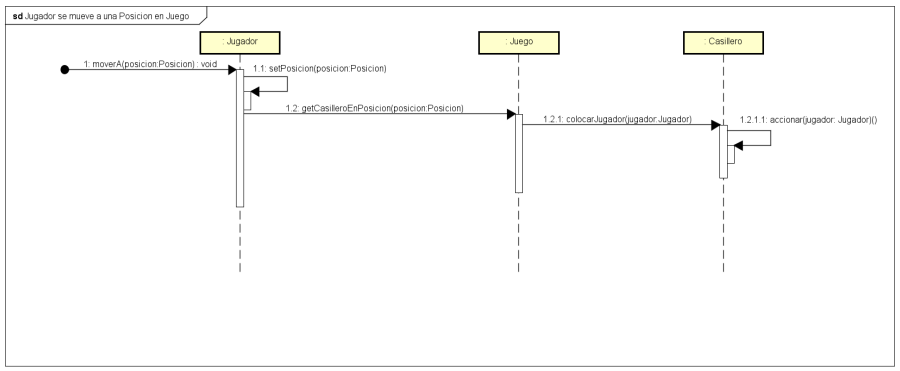

Figura 10: Secuencia de movimiento de jugador una vez tirados los dados

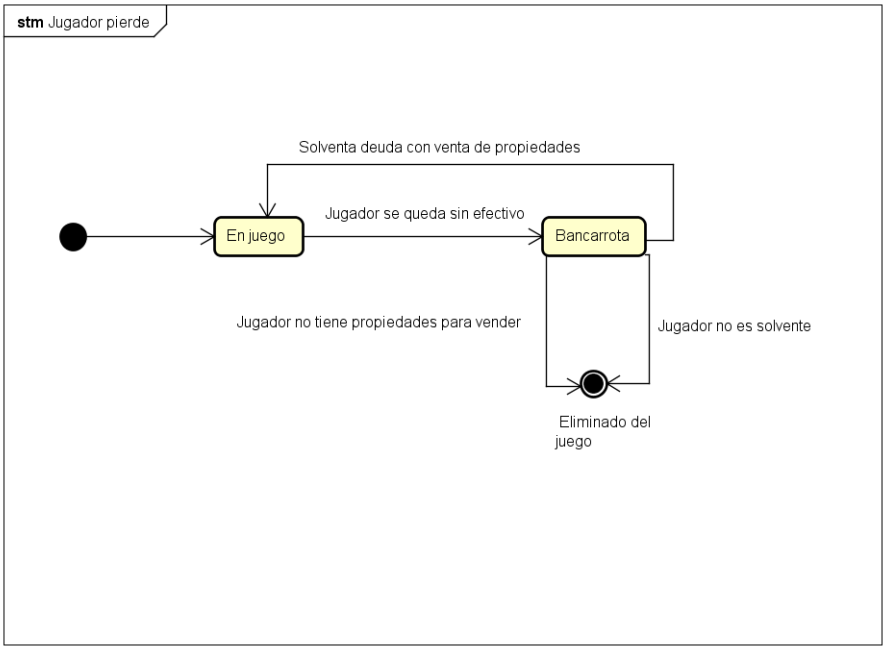

Figura 11: Secuencia de jugador perdiendo

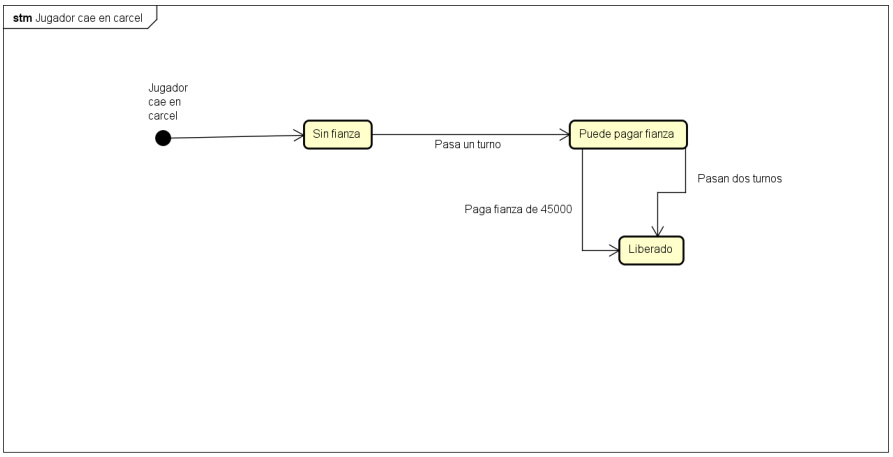

Figura 12: Secuencia de jugador cayendo en carcel

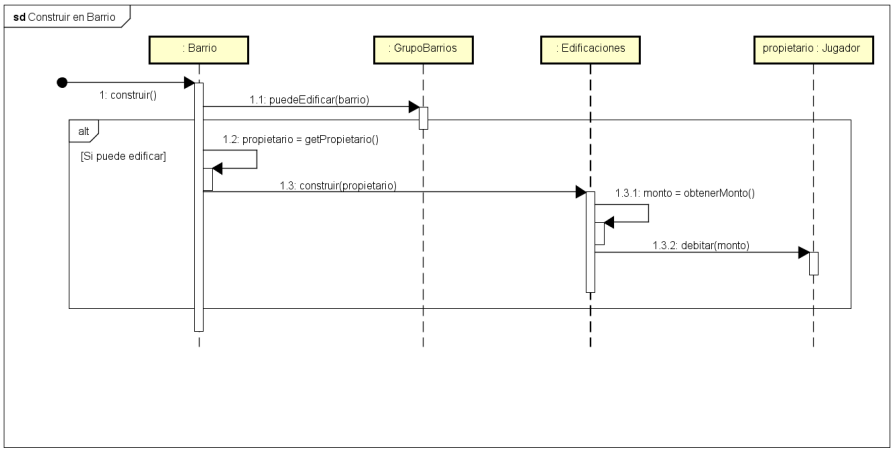

Figura 13: Secuencia de edificación en un barrio

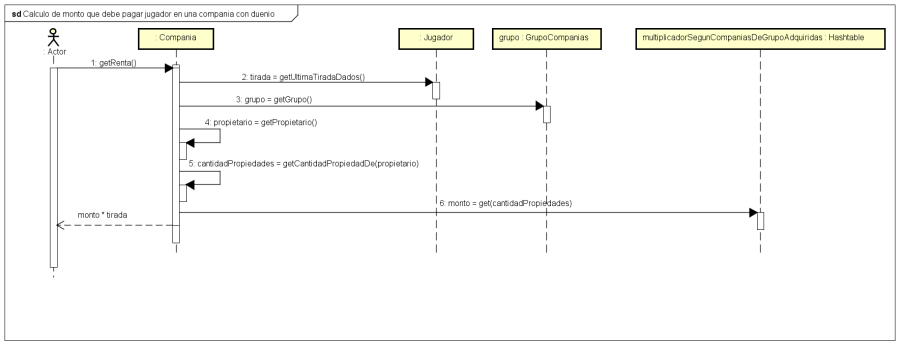

Figura 14: Calculo de monto a cobrar por una compañía cuando esta adquirida por un jugador distinto al que cae