Trabajo Práctico 2: AlgoPoly

[7507/9502] Algoritmos y Programación III
Curso 2 Segundo cuatrimestre de 2017

Nicolás Daniel Vazquez
100338
vazquez.nicolas.daniel@gmail.com

Eliana Gamarra
100016
elianagam2@gmail.com

Javier Albarracín
97568
jalbarracn@gmail.com

Camila Serra
97422
camilaserra5@gmail.com

Índice

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación que implemente un juego relacionado con el clásico juego de mesa Monopoly aplicando los conceptos enseñados en la materia a la resolución de un problema, trabajando en forma grupal y utilizando un lenguaje de tipado estático (Java).

2. Supuestos

- Cuando un jugador cae en **Avance Dinámico** o **Retroceso Dinámico** el mismo será movido hacía otro casillero. Al ser movido, el jugador visitará la nueva casilla. Es decir, si un jugador cae en **Avance Dinámico** y avanza 3 casilleros, caerá en **Impuesto de Lujo** y tendrá que pagar el impuesto.
- Cuando un jugador visita **Avance Dinámico** habiendo sacado 11 o 12, la cantidad de casilleros que debería avanzar se calcula: tirada - cantidadDePropiedades. Se asume que si el jugador tiene más propiedades que su tirada (11 o 12), no avanzará ningún casillero.
- Cuando un jugador visita **Retroceso Dinámico** habiendo sacado 2,3,4,5 o 6, la cantidad de casilleros que debería retroceder se calcula: tirada - cantidadDePropiedades. Se asume que si el jugador tiene más propiedades que su tirada (2,3,4,5 o 6), no retrocederá ningún casillero.
- La clase Provincia conoce conoce los valores de cuanto cuesta Comprar una propiedad ahí y los Edificios que se pueden construir en ella, con sus respectivos precios.
- Las Propiedades Regionales conocen a su vez la otra Provincia a la que están asociadas y su propietario solo puede construir un Edificio si tiene en su lista de propiedades su par.
- Las Propiedades tienen estados que sabrán realizar determinada acción, ya sea comprar, pagar alquiler o construir.
- Si el Jugador compra una propiedad simple puede construir una Casa en ella, en cualquier momento, por no necesitar de otra propiedad.
- Las pruebas de la segunda entrega con fecha 23-11-2017 fueron realizadas de forma genérica, es decir que no se crearon tests para cada Propiedad en particular, sino que se hicieron tests para las clases PropiedadSimple (representa a Neuquen, Tucuman, SantaFe) y PropiedadRegional (representa a Buenos Aires, Córdoba y Salta). De esta forma evitamos ser redundantes con los tests.
- Cuando el jugador debe afrontar un gasto y su capital no es suficiente, se venderán primero las propiedades, y luego las compañías que posea, hasta poder pagar el monto. En caso de no alcanzar, el jugador queda fuera de juego.
- Si el Jugador queda fuera de juego todas sus propiedades quedan disponibles para que otro jugador las compre y su capital pasa a ser nulo.

3. Modelo de dominio

En primer lugar, se creó la clase **Jugador**, para representar a un jugador del AlgoPoly. Esta clase se encarga de mantener el capital de un jugador, sus propiedades adquiridas, y además tiene diferentes estados.

Luego, se creo la interfaz **Casillero** para representar a cada uno de los casilleros del juego. Luego, los casilleros tendrán clases especificas que implementen dicha interfaz. Esto se hizo para unificar a todos los casilleros y que todos respondan al mismo mensaje: **recibirJugador**. Cada uno de los casilleros, sobrecargará el método y lo implementará de acuerdo a lo especificado.

Para los casilleros **Avance Dinámico** y **Retroceso Dinámico** se crearon las clases: **AvanceDinamico** y **RetrocesoDinamico**. Estas clases lo que hacen es, recibir al jugador, y en base a la última tirada del jugador, modificar su posición.

Por otro lado, se creó la clase **Quini6** para representar su casillero. Está clase lo que hace es incrementar el capital del jugador (pero sólo las primeras 2 veces).

Además, se crearon las clases **Carcel** y **Policia**. La clase **Carcel** representa su casillero, y lo que hace es modificar el estado del **Jugador** de forma que éste queda inhabilitado al visitar la **Carcel**. El jugador luego es el encargado de saber si se puede mover, si está habilitado para pagar la fianza, o si no puede hacer ninguna acción. La clase **Policia**, está relacionada con **Carcel**, ya que, al visitar policía, el jugador es enviado a la cárcel.

Para representar las propiedades, se creo la clase **Propiedad**. Esta clase tiene como atributo:

- propietario que indica que **Jugador** la posee.
- provinicia y, en caso de ser una propiedad regional, su provincia completento.
- edificio del cual obtener su alquiler cuando un jugador que no sea propietario tendra que pagar
- estado que sera responsable de comprar, cobrarAlquiler y construir casas u hoteles de diferente manera

Para las compañías (subte, tren, aysa, edesur) se creó la clase abstracta **Compania**. También, se crearon las clases Aysa, Edesur, Subte y Tren que extienden de la clase **Compania**. Además, se creo la interfaz **Estado**, de la cuál heredan **SinPropietario** y **ConPropietario**. Estas clases se encargan de recibir al jugador, y dependiendo el estado hacer lo que corresponda. La clase **Compania** tiene a su vez como atributo otra clase **Compania** para poder calcular el impuesto en caso de que el mismo jugador posea ambas.

4. Diagramas de clase

El diagrama muestra las relaciones entre las clases creadas.

En primer lugar vemos el diagrama del Jugador. Se puede ver que el jugador tiene un Estado que puede ser: Habilitado, Encarcelado o FueraDeJuego. También, se ve que el jugador tiene dos Dados, y también que conoce a la clase Casillero.

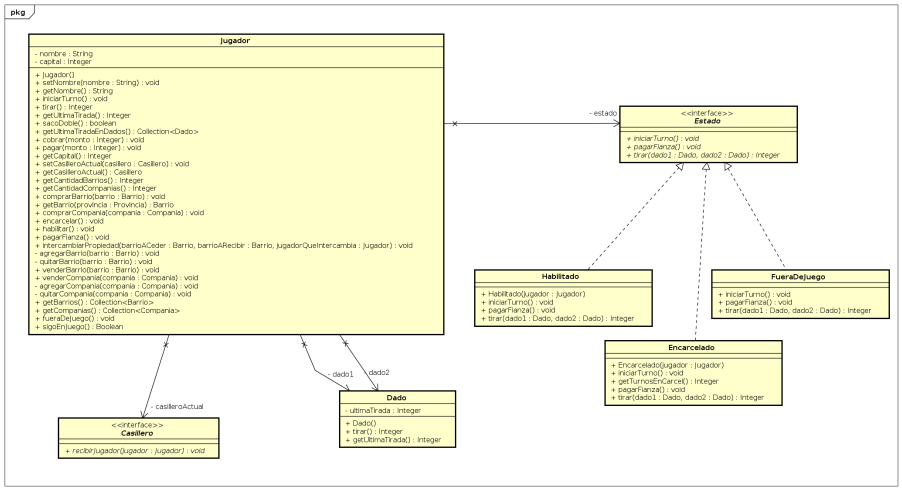

Figura 1: Diagrama del jugador.

Por otro lado, vemos en más detalle la implementación de los casilleros. Se creó la interfaz **Casillero**, que es implementada por todos los casilleros que conforman el Algopoly. Además vemos que la clase **Policia** conoce a la clase **Carcel** ya que debe enviar al jugador a la misma.

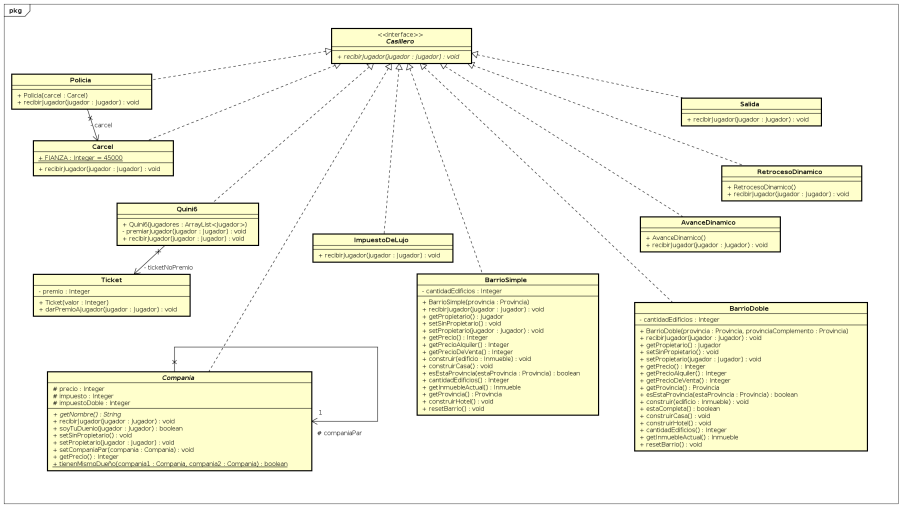

Figura 2: Diagrama de los casilleros.

Para integrar todo el juego se creo la clase **Tablero**. La misma tiene una lista de casilleros y una lista de jugadores. También tiene una lista de posiciones que tienen la ubicación de cada casillero.

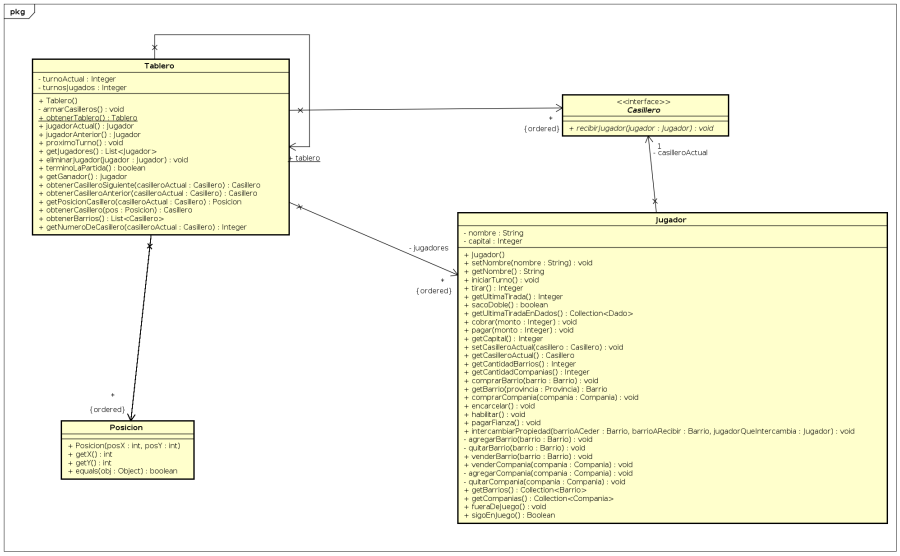

Figura 3: Diagrama del tablero.

Vemos en este diagrama lo explicado anteriormente. Existe la clase abstracta **Compania** de la cual hereda cada una de las compañías. También, la clase tiene un estado para identificar si tiene o no propietario, y en base a eso saber que hacer cuando recibe a un jugador.

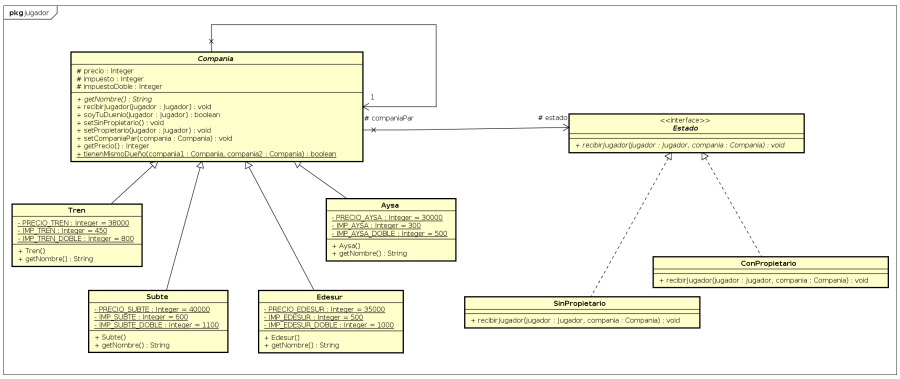

Figura 4: Diagrama de compañías

Detalles de implementacion de Propiedades

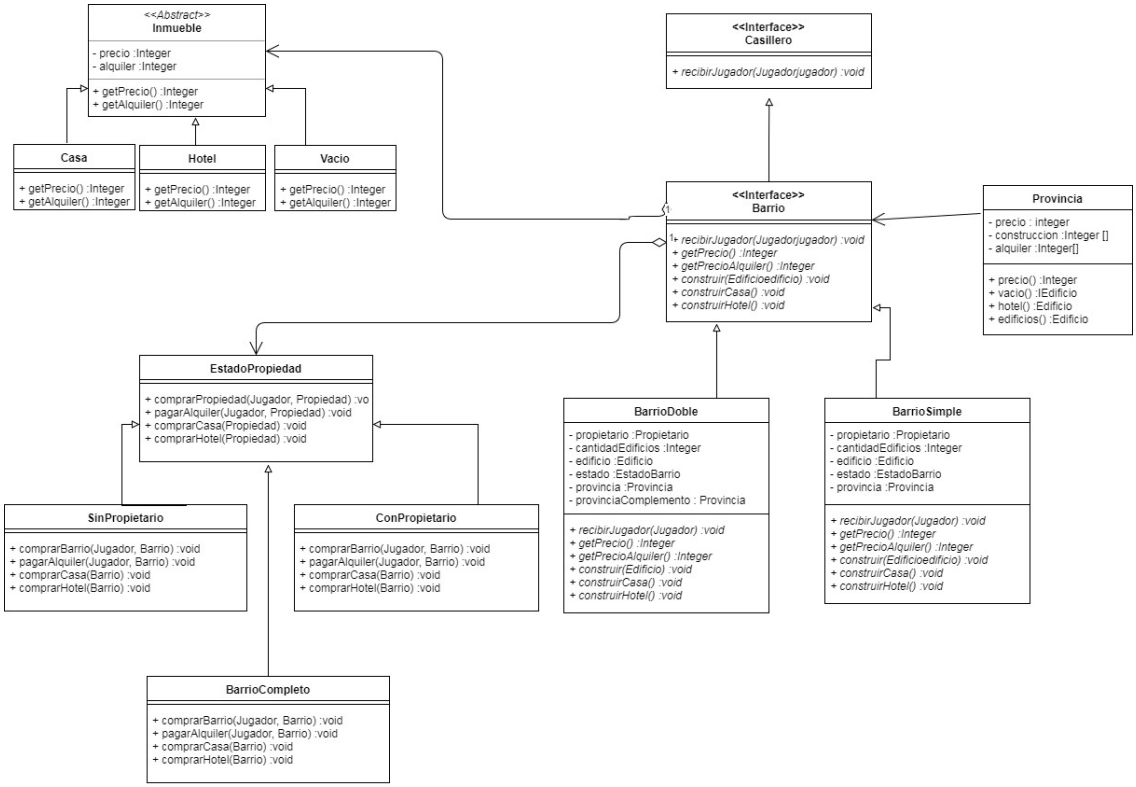

Figura 5: Diagrama de la Propiedad.

En el siguiente diagrama se muestra como funciona la implementación del patrón Factory aplicado a la creación de **Propiedades**. La clase **PropiedadFactory** posee dos métodos que al ser llamados devuelven una instancia de la Propiedad correspondiente. Decidimos utilizarlo para resolver el problema que nos plantearía tener una clase para cada propiedad del tablero, lo cual resulta innecesario teniendo herramientas como esta.

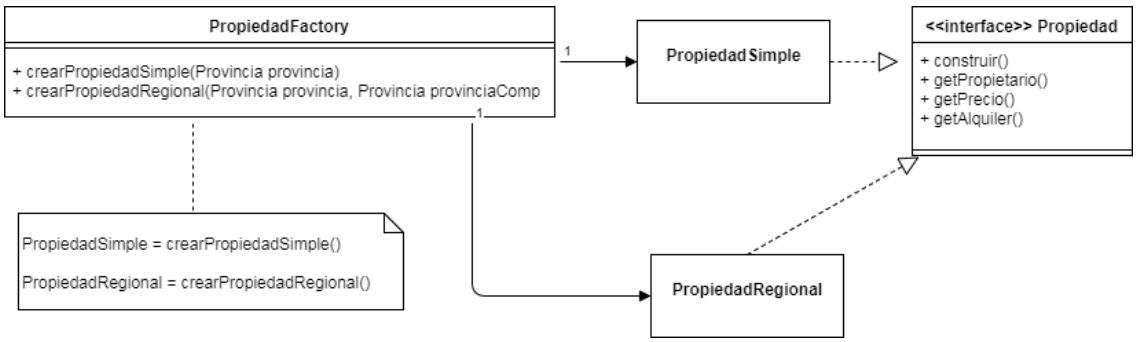

Figura 6: PropiedadFactory.

5. Detalles de implementación

5.1. Interfaz Casillero

La interfaz **Casillero** se creó para que todas las casillas sigan el contrato que se plantea, ya que, por más que cada casilla tenga una acción diferente sobre el jugador, representan lo mismo. Se decidió que el mensaje a sobrecargar sea **recibirJugador**, ya que cuando un jugador cae en la casilla, la casilla lo recibe y luego actúa sobre él.

```
public interface Casillero {
    void recibirJugador(Jugador jugador);
}
```

5.2. Calculo premio Quini6

Para asignarle un premio al jugador que visita la casilla **Quini 6**, se modeló un Ticket, el cual se encarga de sumar el dinero ganado al jugador que cae en este casillero.

```
public class Ticket {
    private Integer premio;
    public Ticket(Integer valor){
        this.premio = valor;
    }
    public void darPremioAJugador(Jugador jugador){
        jugador.cobrar(this.premio);
    }
}
```

Luego para determinar qué Ticket le corresponde al que visita el casillero, decidimos utilizar un **HashMap(Jugador, Queue<Ticket>)** . La cola se inicializa con dos tickets ganadores, y luego a medida que el jugador visita el casillero, se desencola un ticket y se encola un ticket no ganador. De esta forma la cola asociada al jugador siempre tiene dos elementos, y luego de las dos primeras visitas sólo quedan tickets sin premio, por lo cual no es necesario el uso de condicionales para determinar el monto de dinero que se le suma al jugador visitante.

```
    private void premiarJugador(Jugador jugador){
        Ticket ticket = this.premios.get(jugador).remove();

        ticket.darPremioAJugador(jugador);

        this.premios.get(jugador).add(this.ticketNoPremio);
    }
```

5.3. Interfaz Estado Jugador

Para manejar los estados del jugador se creó la interfaz **Estado**. Esta interfaz tiene los siguientes métodos:

```
public interface Estado {
    void iniciarTurno();
    void pagarFianza();
    Integer tirar(Dado dado1, Dado dado2);
}
```

En principio, se crearon las clases **Habilitado**, **Encarcelado** y **FueraDeJuego**, que implementan esta interfaz. Habilitado es el estado natural de un jugador. Luego, si el jugador cae en la carcel su estado pasa a ser Encarcelado, por tres turnos o hasta que pague la fianza. Por último, cuando un jugador debe afrontar un gasto y no tiene más capital, su estado pasa a ser FueraDeJuego.

5.4. Enum Provincia

Para manejar los valores numericos de cada Propiedad se creo un enum de Provincias que tiene el precio de la propiedad y casa una de las construcciones que soporta el barrio: **Vacio**, **Casa** y **Hotel** con sus respectivos valores de Precio y Alquiler.

```
    public enum Provincia {
// nombre (precio,[Casa,Hotel],[alqSim,alq1Casa,alq2Casa,Hotel])

BSAS_SUR (20000, new Vacio(0,2000), new Casa(5000, 3000), new Casa(5000, 3500), new Hotel(8000,50

BSAS_NORTE (25000, new Vacio(0, 2500), new Casa(5500, 3500), new Casa(5500, 4000), new Hotel(9000

    Provincia(Integer precio, Inmueble vacio, Inmueble casa1, Inmueble casa2, Inmueble hotel) {
this.precio = precio;
this.edificios = new ArrayList<>();

this.edificios.add( vacio );
this.edificios.add( casa1 );
if ( casa2 != null ) {
this.edificios.add( casa2 );
this.edificios.add( hotel );
}
}
```

5.5. BarrioDoble

Un Jugador no puede edificar en una BarrioDoble si no tiene su Barrio Complemento en su lista de propiedades. Para eso el constructor de BarrioDoble conoce su Provincia y la de su complemento y en el momento de construir una casa tendra que buscar en la lista de Barrios del jugador el otro Barrio y si la tiene cambiara su estado a BarrioCompleto.

```
public PropiedadRegional(Provincia provincia, Provincia provinciaComplemento){
     this.provincia = provincia;
     this.provinciaComplemento = provinciaComplemento;
        this.estado = new SinPropietario();
        this.cantidadEdificios = 0;
```

```
    }

    public boolean estaCompleta() {
     return this.propietario.getPropiedad(this.provinciaComplemento) != null;
    }

    @Override
public void construirCasa() {

if ( this.estaCompleta() && this.cantidadEdificios < 2) {
this.estado = new RegionCompleta();
this.estado.construirCasa(this);
}
}
```

5.6. Clase abstracta EstadoBarrio

Para manejar los estados de la propiedad se creó la clase abstracta **EstadoBarrio**. Esta tiene los siguientes métodos:

```
public abstract class EstadoBarrio {

abstract void comprarBarrio(Jugador jugador, Barrio barrio);
void pagarAlquiler(Jugador jugador, Barrio barrio) {
abstract void construirCasa(Barrio barrio);
abstract void construirHotel(Propiedad barrio);

}
```

Las clases **SinPropietario**, **ConPropietario** y **BarrioCompleto** la implementan. SinPropietario sobreescribe el metodo para asignarle un propietario a la propiedad y ademas un edificio vacio. ConPropietario y BarrioCompleto se comprortan de la misma manera cuando se llama a pagarAlquiler y solo BarrioCompleto podra comprarCasa y ConstruirHotel

```
public class BarrioCompleto extends EstadoBarrio {

@Override
public void comprarBarrio(Jugador jugador, Barrio barrio) {
}

@Override
public void construirCasa(Barrio barrio) {
Edificio edificio = propiedad.getProvincia().edificios().get(barrio.cantidadEdificios() + 1);

barrio.getPropietario().pagar(edificio.getPrecio() );
barrio.construir(edificio);
}

@Override
public void construirHotel(Barrio barrio) {
Edificio edificio = barrio.getProvincia().hotel();
barrio.getPropietario().pagar(edificio.getPrecio() );
barrio.construir(edificio);
}
```

```
}
```

5.7. Avance/Retroceso Dinámico

Para calcular la cantidad de casilleros que debe avanzar o retroceder un jugador al caer en cualquier de estas dos casillas se implementó un mapa que mapea los números del 1 al 12 con las funciones correspondientes.

```
private Map<Integer, Function<Jugador, Integer>> funciones;
public AvanceDinamico() {
    this.funciones = new HashMap<>();
    for (int i = 1; i <= 6; i++) {
        this.funciones.put(i, j -> j.getUltimaTirada() - 2);
    }
    for (int i = 7; i <= 10; i++) {
        this.funciones.put(i, j -> j.getCapital() % j.getUltimaTirada());
    }
    for (int i = 11; i <= 12; i++) {
        this.funciones.put(i, j -> j.getUltimaTirada() - j.getCantidadPropiedades());
    }
}
```

6. Diagramas de secuencia

En el siguiente diagrama se muestra como funciona la casilla **Avance Dinámico** cuando un jugador cae en ella habiendo sacado 8 con los dados. El jugador avanzará la cantidad de casilleros equivalentes a hacer el módulo entre su capital y su tirada.

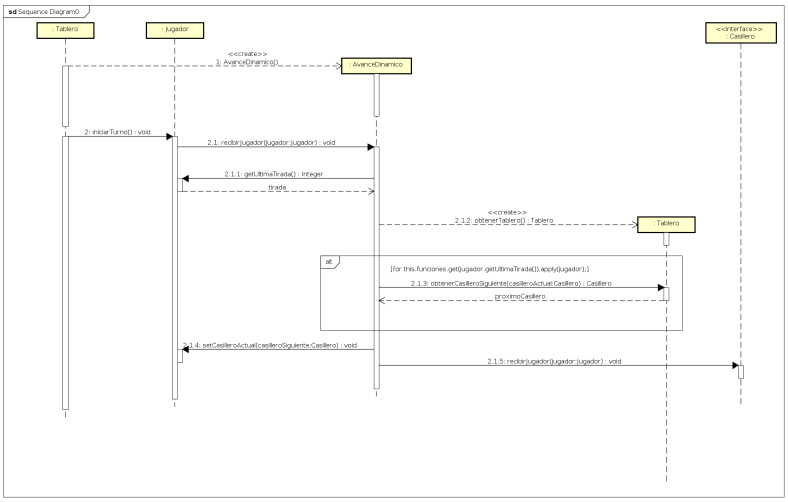

Figura 7: Avance Dinámico.

En este diagrama se muestra cómo funciona la casilla **Quini6** cuando un jugador cae en ella cuatro veces. Su capital sólo se verá afectado en la primer y segunda visita.

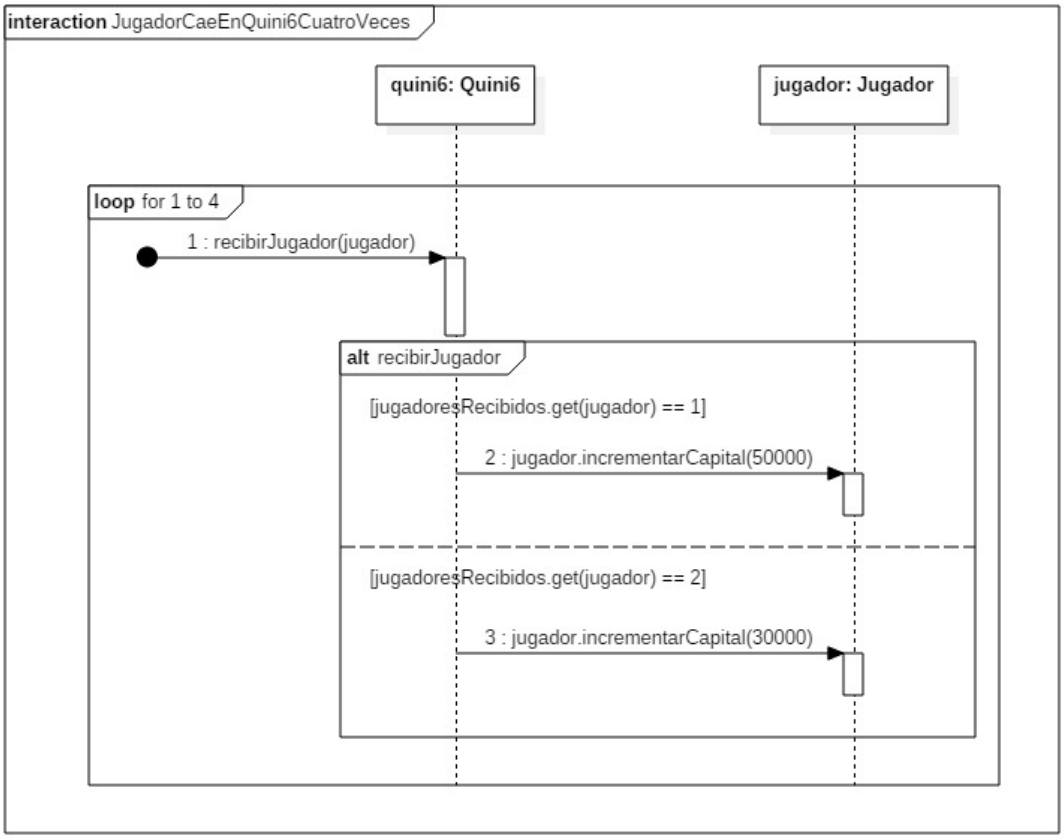

Figura 8: Quini6.

El siguiente diagrama muestra la interacción entre el **Jugador** y el casillero **Carcel** luego de caer en dicho casillero. El método encarcelar asigna una instancia de **Encarcelado** al estado del **Jugador**.

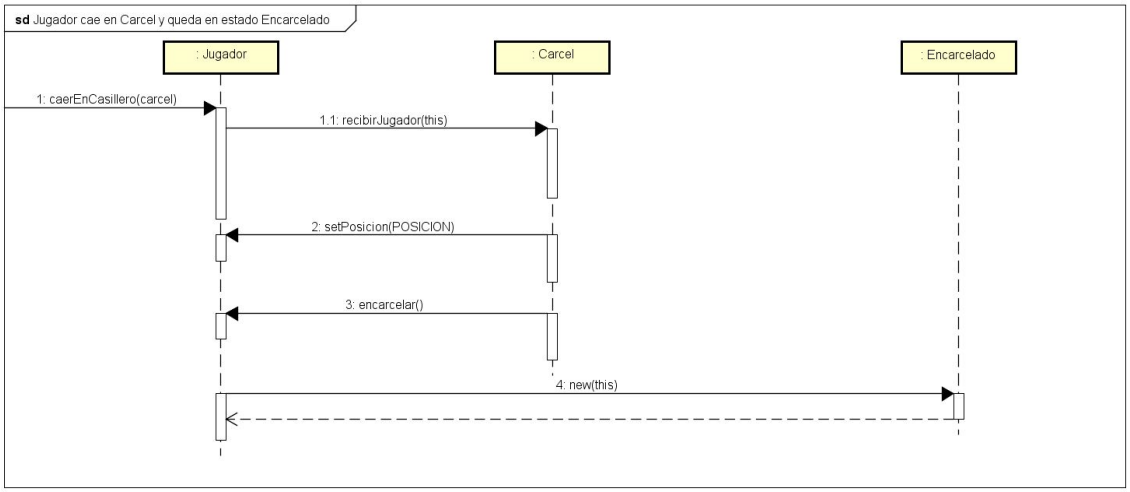

Figura 9: Jugador cae en Carcel.

El siguiente diagrama muestra como luego de pagar una fianza se modifica el estado del **Jugador** a **Habilitado**. El diagrama supone que el **Jugador** se encuentra en condiciones de pagar la fianza, esto es, transcurrió al menos un turno desde que fue encarcelado y tiene un capital mayor a 45000.

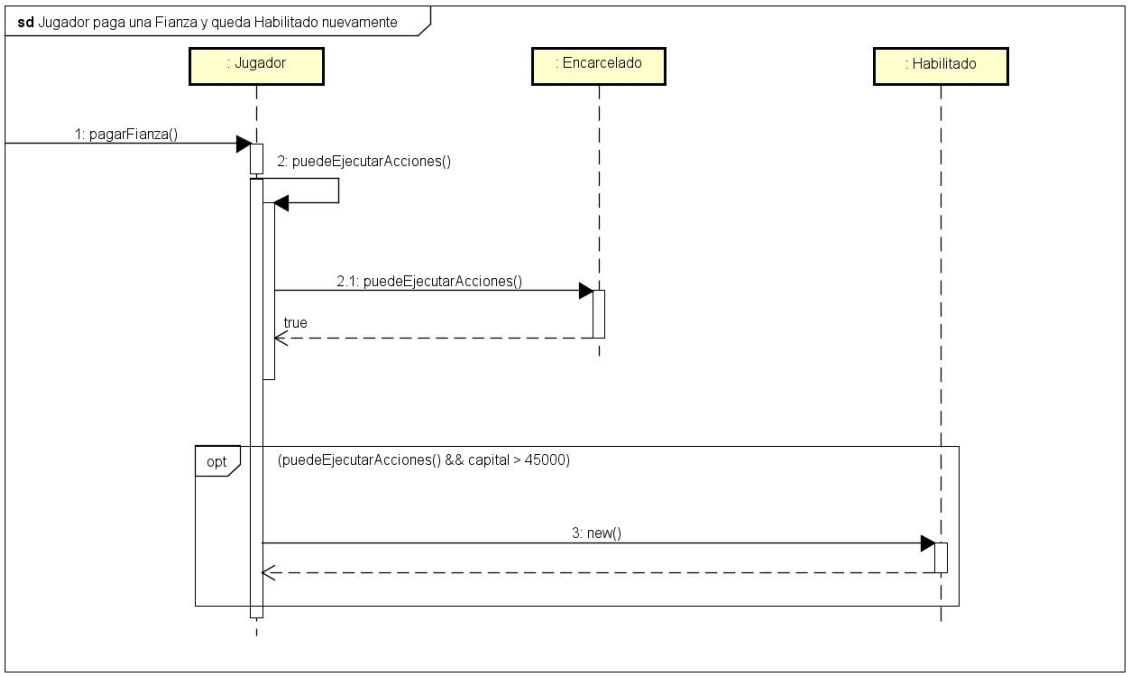

Figura 10: Jugador paga una fianza.

El siguiente diagrama muestra como luego de transcurridos cuatro turnos encarcelado se modifica el estado del **Jugador** a **Habilitado**. El diagrama supone que el método inicial iniciarTurno() está siendo invocado por cuarta vez desde que el **Jugador** fue encarcelado.

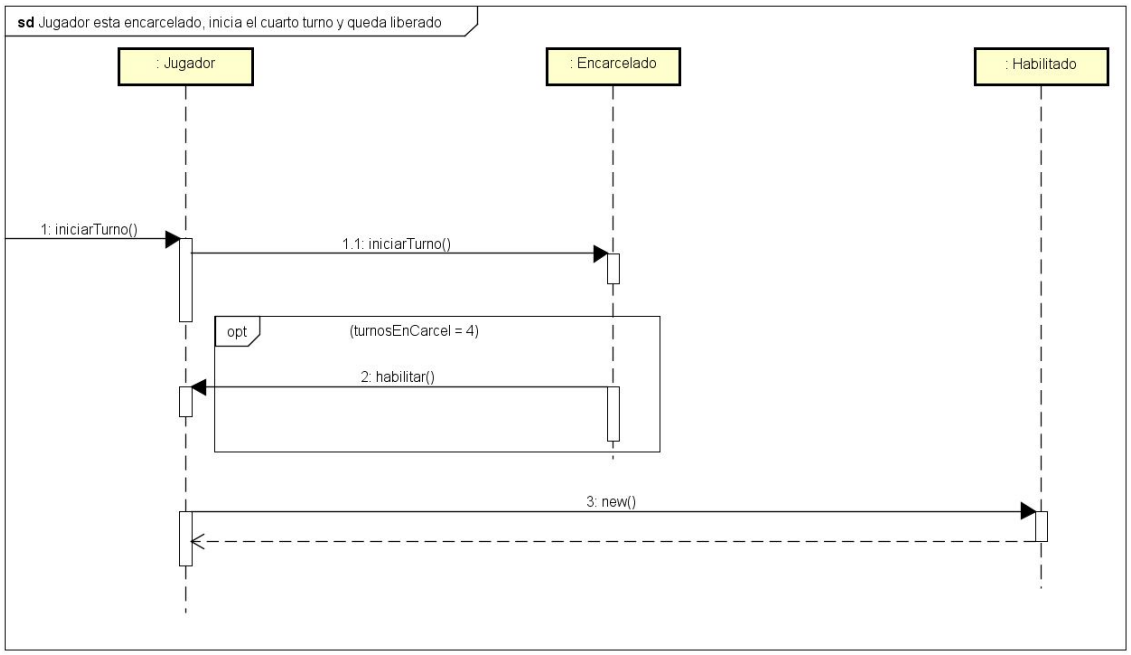

Figura 11: Jugador queda liberado luego de cuatro turnos.

En el siguiente diagrama se muestra como al recibir un jugador el estado de BarrioSimple, al comienzo SinPropietario recibe el jugador y el barrio para comprar el barrio y luego se cambia el estado a BarrioCompleto, en la misma funcion de recibir jugador, al estado se le envia el mensaje de pagar alquiler pero como es propietario y el jugador es la misma persona no hace nada

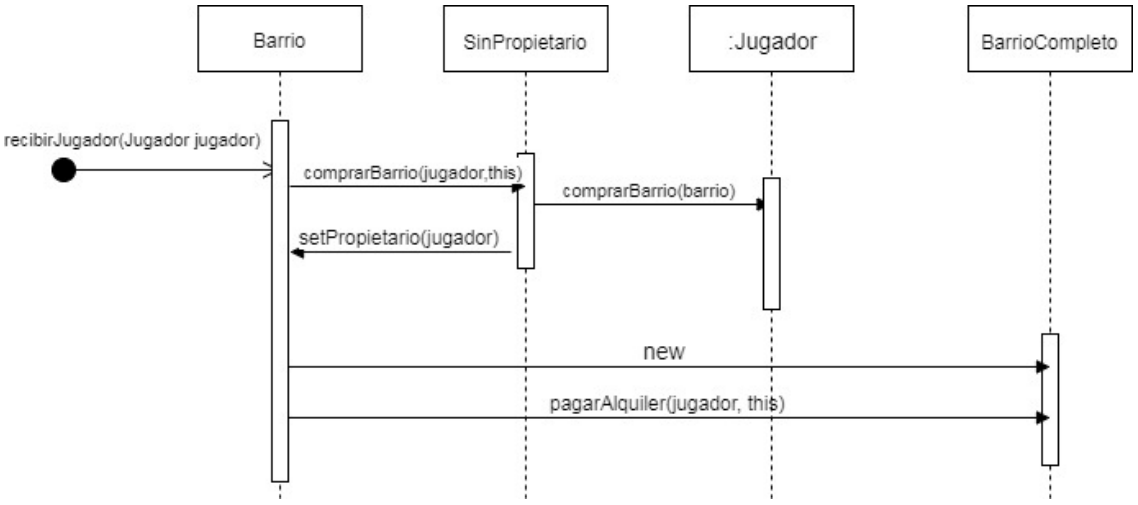

Figura 12: SetPropietario.

En el siguiente diagrama se muestra cómo se realiza la construcción de una casa en un BarrioDoble.

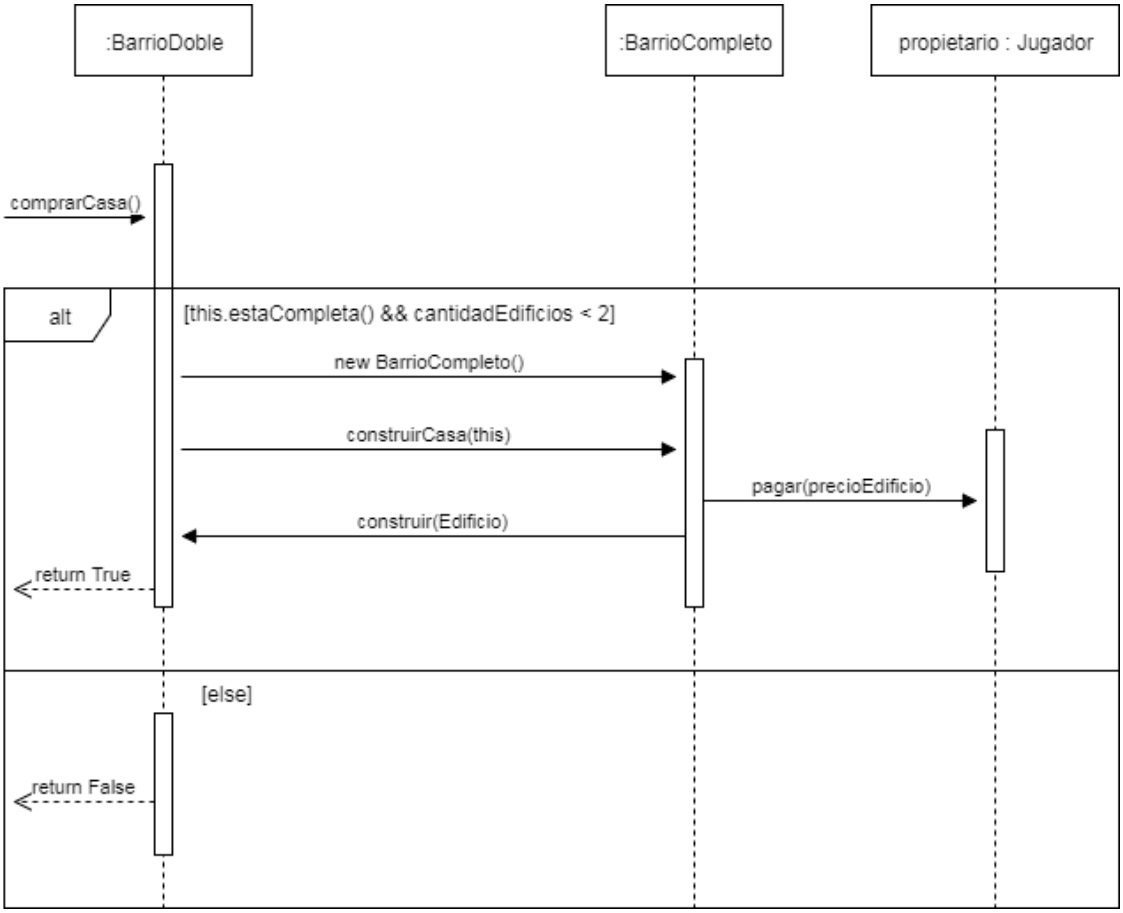

Figura 13: construirCasa.

En el siguiente diagrama se puede ver que pasa cuando un jugador cae en una compañía por primera vez, y luego otro diferente cae en la misma compañía.

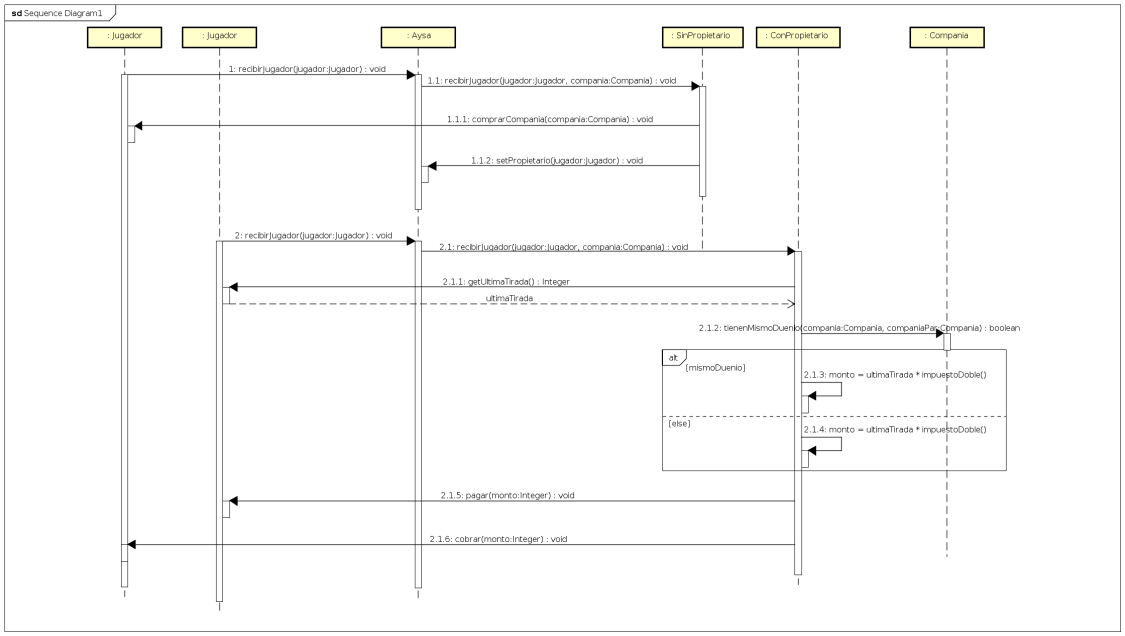

Figura 14: Compañías

En este diagrama se trató de mostrar como es el curso normal de un turno de un juegador.

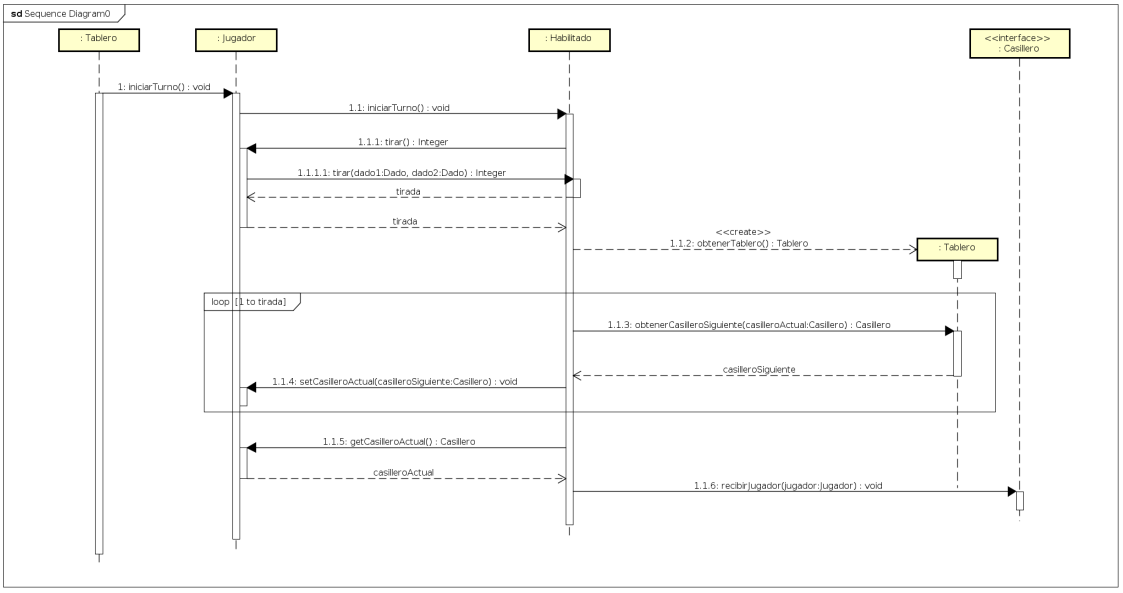

Figura 15: Turno de un jugador

En este diagrama se muestra que sucede cuando un jugador debe pagar algo y no le alcanza el capital.

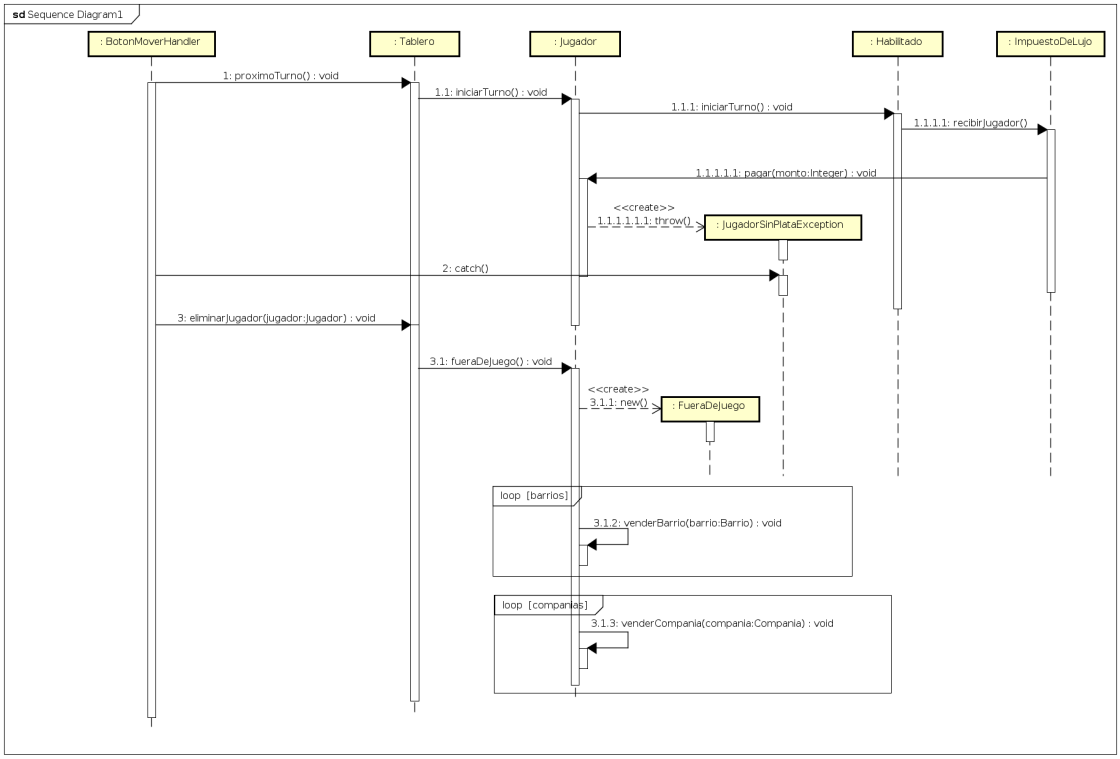

Figura 16: Fuera de juego

En este diagrama se muestra cómo funciona el casillero Impuesto de Lujo.

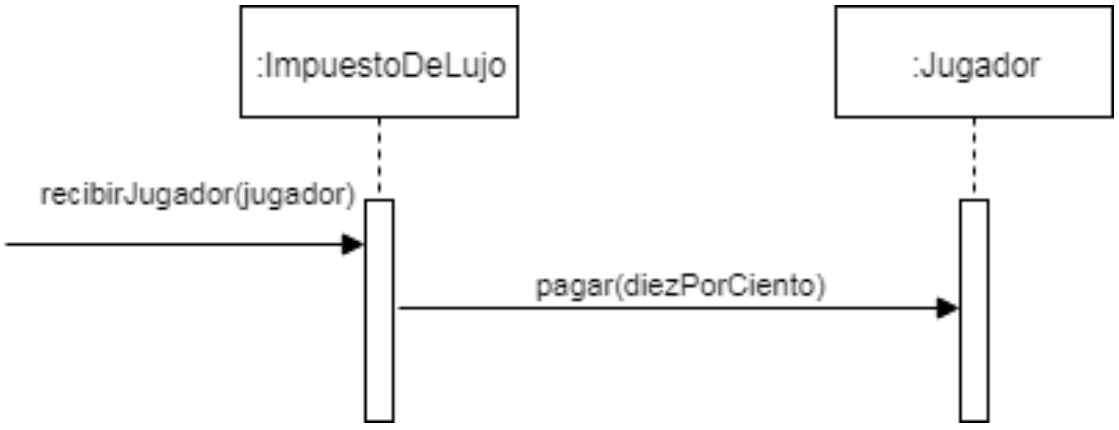

Figura 17: Impuesto de Lujo

7. Diagramas de paquete

En el siguiente diagrama se muestra como se realizó la estructura de los paquetes. En principio se creó el paquete algobay para agrupar todo. Después se dividió en modelos/vistas/controladores. Luego, dentro de modelos, se dividió en jugador y tablero. En el paquete jugador está la clase Jugador y sus posibles estados. En el paquete tablero, están todos los casilleros y como subdivision del paquete de casilleros se encunetras otros dos paquetes para barrios y companias.

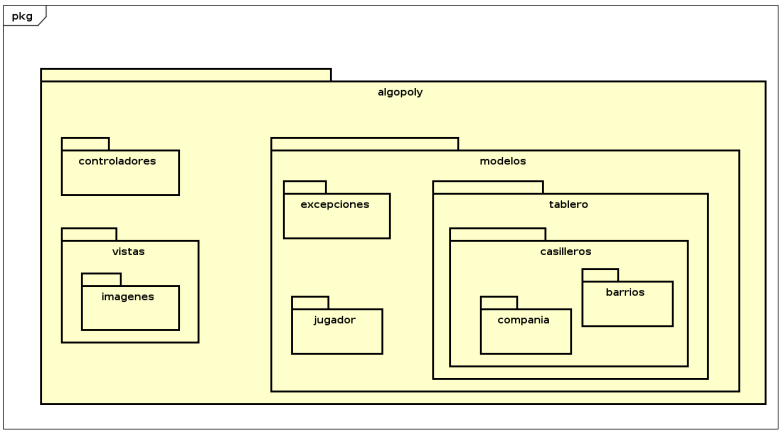

Figura 18: Diagrama de Paquetes.