FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2

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

Grupo T10

Nombre	Padrón	Mail
Perez Ondarts, Flavio	96786	perezflavio94@gmail.com
Piersantolini, Matías	98299	matias.piersantolini@gmail.com
Ontiveros, Juan	98425	ontiverosfuertes@yahoo.com.ar
Aguirre, Ariel	100199	ariedro@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 completa con interfaz gráfica de una versión simplificada del Monopoly, utilizando los conceptos del paradigma de la orientación a objetos vistos en el curso.

2. Supuestos

Ésta entrega fue pensada como la primera en tres que se desarrollarán en un trabajo a largo plazo, por lo tanto resulta evidente que si bien algunos funcionamientos pueden parecer innecesarios, los razonamientos que llevaron a que se implementen fue una vista a futuro para que el modelo luego sea más acopable a los cambios que se realicen.

También debido a esto, surgieron algunos comportamientos que no fueron contemplados para desarrollarlos en éste momento, y por lo tanto fueron solucionados de maneras particulares, ya que después con el modelo completo se delegará de una mejor forma.

Enumerando algunos ejemplos:

- En la toma de decisión de que un jugador deba comprar una propiedad, ya que el modelo de MVC todavía no fue implementado, las pruebas unitarias tuvieron que asumir posibles interacciones del jugador que no deberían ser contempladas en el modelo.
- El funcionamiento de Avance y Retroceso Dinámico fue en parte delegado al Jugador para garantizar que no se pierda la singularidad de la interfaz que ésta implementaba, esto en un futuro será responsabilidad de una clase Tablero que aún no fue necesario desarrollar.

3. Modelo de dominio

El modelo general que se ideó fue el de una interacción entre 4 clases principales, éstas son: AlgoPoly, Jugador, Casillero y Dado. Debido a la generalidad de Casillero, éste delega la responsabilidad del tipo en una interfaz llamada Encasillable con un método de acción, de la cual cada clase lo implementa de diferentes maneras.

Además, Jugador contiene propiedades que el mismo posee, las mismas se referencian con la interfaz Comprable, que implementan los barrios y/o servicios para poder referenciarlos de manera polimorfa.

4. Diagramas de clase

A continuación se mostrarán dos diagramas de clase que ayudarán a representar el funcionamiento del programa.

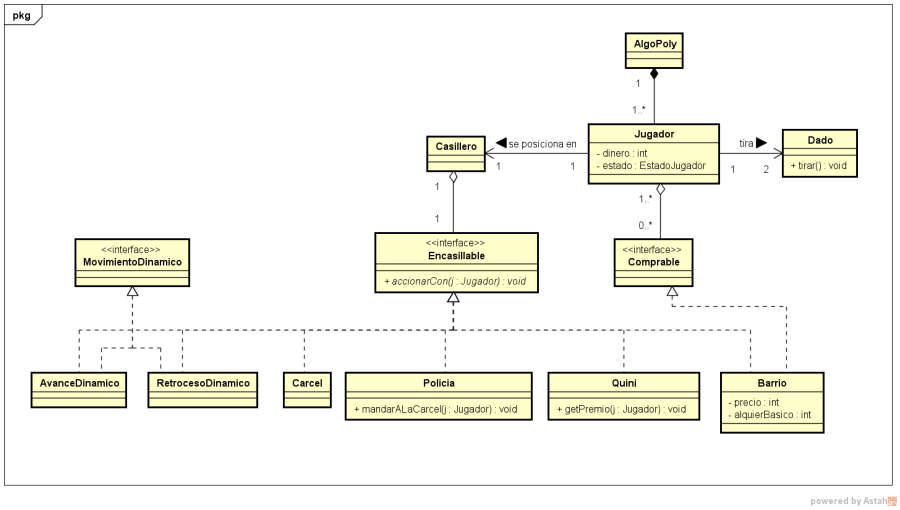

Figura 1: Diagrama general de clases.

Se puede ver cómo el uso de la interfaz Encasillable asegura el polimorfismo para que todos los casilleros puedan accionar sobre los jugadores. Y la interfaz Comprable es el nexo entre las propiedades de los barrios y lo que adquieran los jugadores.

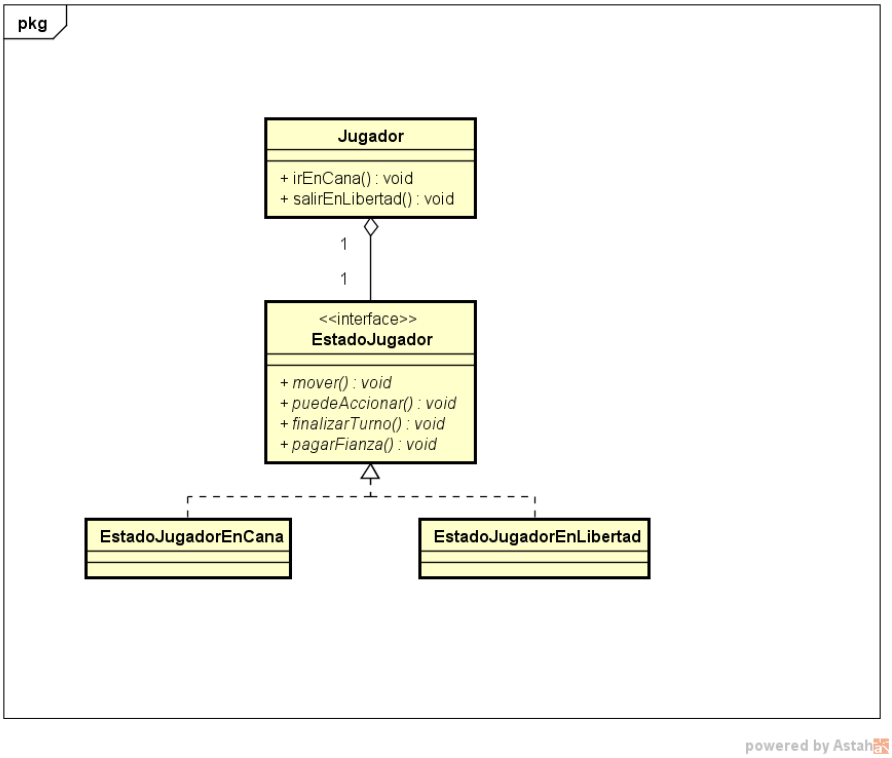

Figura 2: Diagrama de los diferentes estados que implementa el jugador.

5. Diagramas de secuencia

Se ilustrarán diferentes situaciones en las que el Jugador interactuaría con los diferentes Encasillables al momento de que los mismos se accionen.

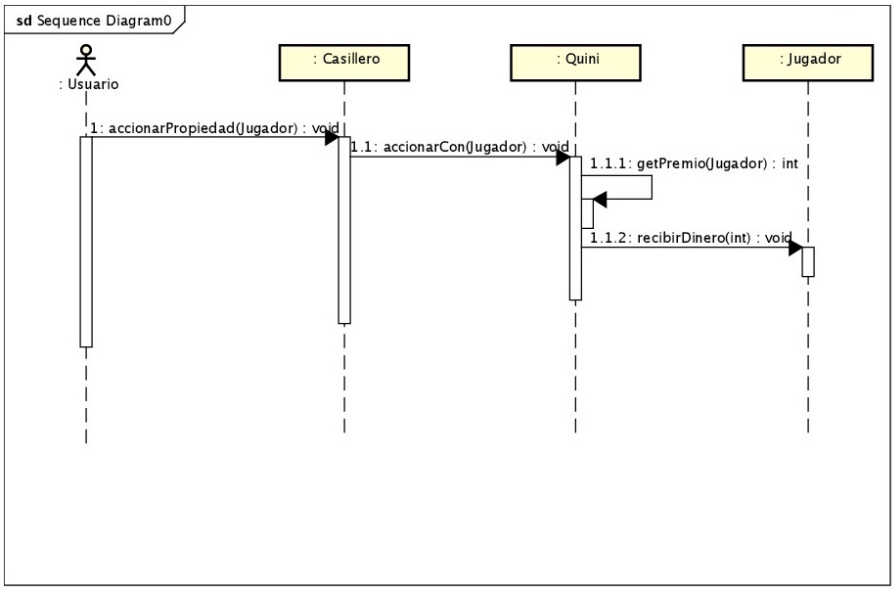

Figura 3: Funcionamiento de Quini 6

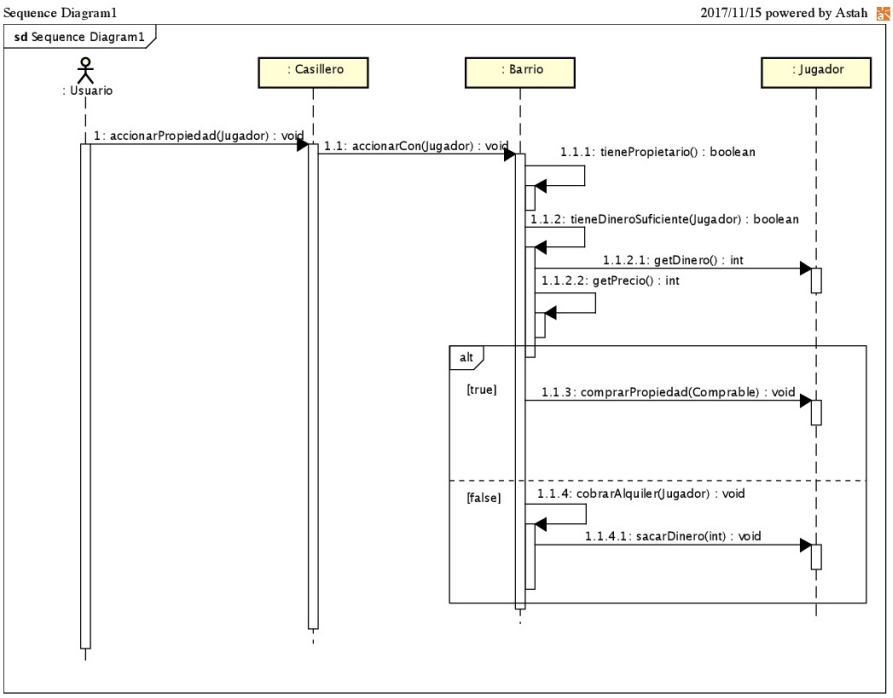

Figura 4: Funcionamiento de Barrio

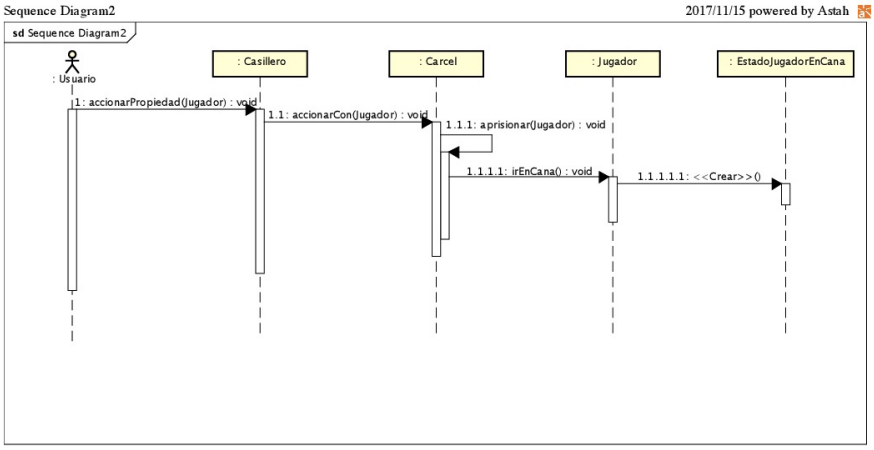

Figura 5: Funcionamiento de Cárcel

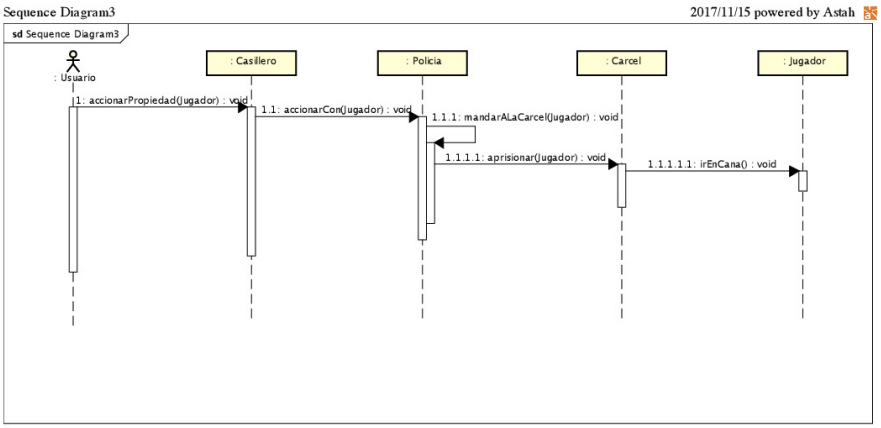

Figura 6: Funcionamiento de Policía

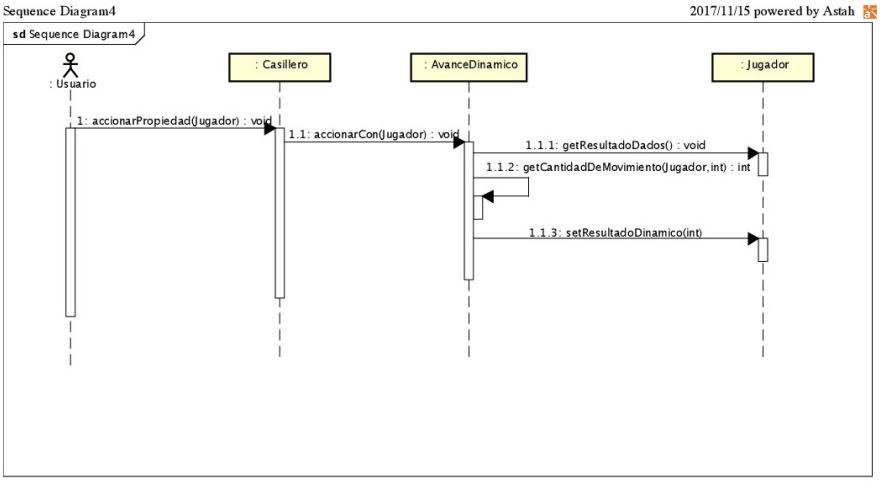

Figura 7: Funcionamiento de Avance Dinámico

6. Diagrama de paquetes

Ésta es una primera visión de cómo se manejaron los paquetes en el programa.

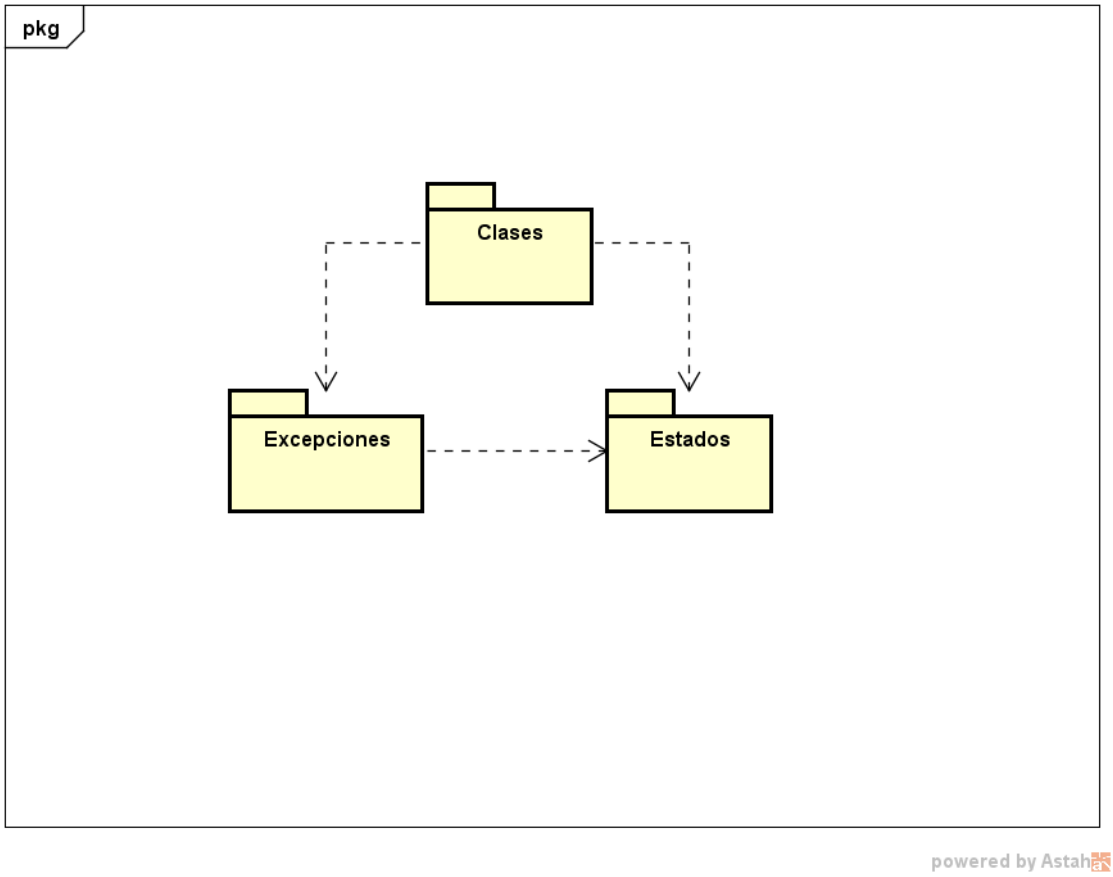

Figura 8: Paquetes

7. Diagramas de estado

A continuación se ilustrará los diferentes estados que implementan la clase Jugador.

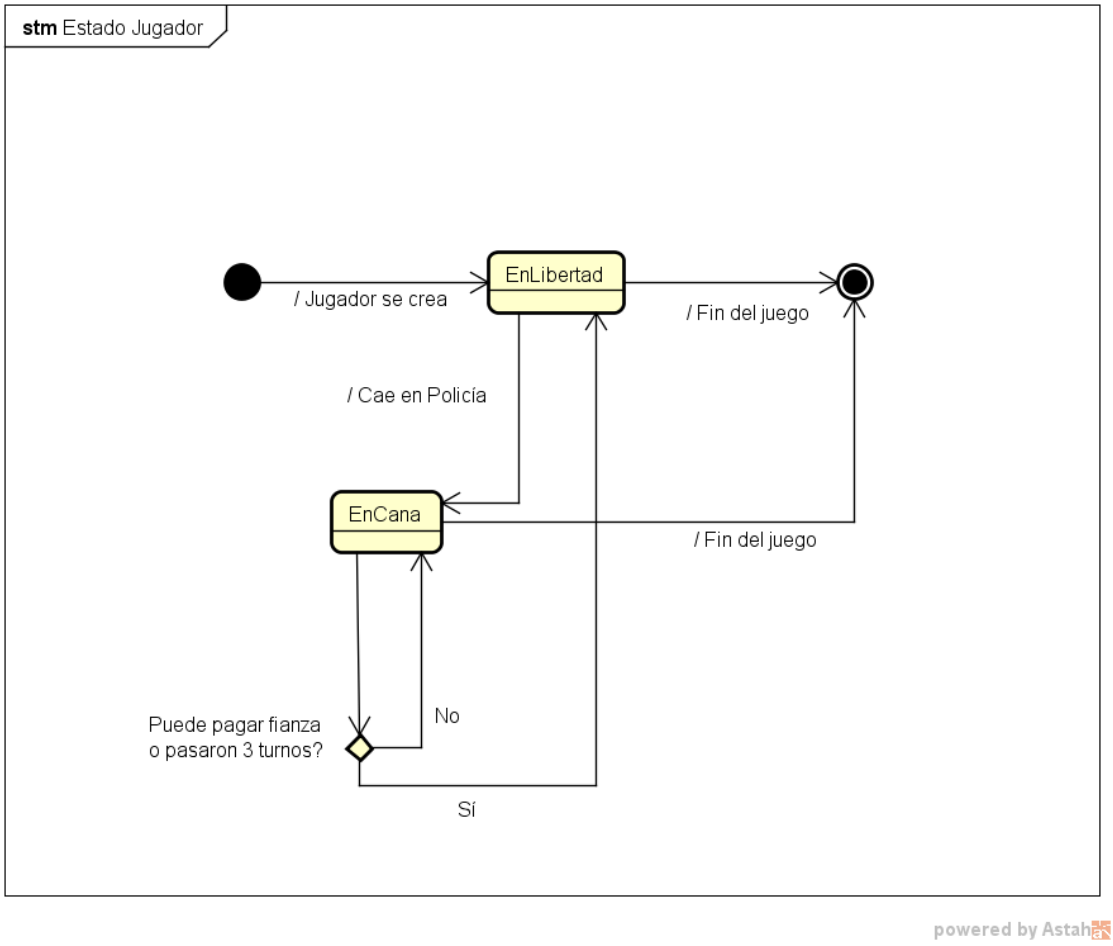

Figura 9: Estados de Jugador en el funcionamiento de la aprisión.

8. Detalles de implementación

8.1. Propiedades

Un jugador adquiere diferentes propiedades, las cuales son referenciadas por su interfaz Comprable, ya que ésta es la misma entidad que implementa el Encasillable, se puede ver que los jugadores se adueñan de los elementos que contienen los casilleros. De esta manera, en un futuro cuando un jugador caiga en un Barrio que posea otro jugador, será fácilmente verificable el alquiler que se deba pagar y a quién.

```
public class Barrio implements Encasillable, Comprable{

    private Jugador propietario;
    private int precio;
    private int alquilerBasico;

    public void accionarCon(Jugador unJugador) {
        if (!this.tienePropietario() && this.tieneDineroSuficiente(unJugador)) {
            unJugador.comprarPropiedad(this);
            this.propietario = unJugador;
        }
        else this.cobrarAlquiler(unJugador);
    }
```

8.2. Cárcel

Cómo se ilustró anteriormente, para ahorrarse verificaciones innecesarias, para el movimiento del jugador se aplicó el patrón State, es decir que se delegó el movimiento del Jugador dependiendo del estado de libertad en que se encuentre.

9. Excepciones

JugadorEstaEnCanaException Ésta excepción se lanza cuando un jugador trata de hacer una acción estando en la cárcel. Se implementará con una ventana de exclamación.

JugadorNoPuedePagarFianzaException Ésto ocurre cuando un jugador intenta pagar la fianza pero, tanto porque no le alcance el dinero o porque no esté en la cárcel, no puede hacerlo, se manejará igual que la excepción anterior.