FACULTAD DE INGENIERIA
Universidad de Buenos Aires

# Trabajo Práctico 2 — Catan

Paradigmas de Programación
2C - 2025


| | |
|---|---|
| GAZCÓN, Felipe | 110933 |
| SALADINO, Joaquín | 111220 |
| SANTILLÁN, Thomás | 109574 |
| CILIA, Tomás Eugenio | 110819 |

# Índice

**1. Introduccion** **2**

**2. Supuestos** **2**

**3. Diagramas de Clase** **2**
3.1. Tablero . . . 2
3.2. Terrenos . . . 3
3.3. Banco y Recurso . . . 4
3.4. Recurso extendido . . . 5
3.5. Comercio . . . 5
3.6. Puertos . . . 6
3.7. Desarrollo . . . 6
3.8. Estructuras . . . 7
3.9. Jugador . . . 8
3.10. Juego . . . 9
3.11. General . . . 9

**4. Diagramas de Secuencia** **11**
4.1. Tablero . . . 11
4.2. Producción (tiro de dados) . . . 13
4.3. Comercio . . . 13

**5. Diagrama de Paquetes** **15**

**6. Detalles de Implementacion** **16**
6.1. Double Dispatch y Sobrecarga . . . 16
6.2. Strategy ” State . . . 16
6.3. Observer . . . 16
6.4. Null Pattern . . . 16
6.5. Factory . . . 16
6.6. Herencia, Polimorfismo y Delegacion . . . 16
6.7. Principios de diseño aplicados . . . 17
6.8. Excepciones . . . 17

# 1. Introduccion

El siguiente informe contiene todo lo necesario para comprender nuestra implementacion del juego Catan para el trabajo practico 2, de la materia Paradigmas de la Programacion. La misma es el resultado de mas de un mes de trabajo en Java aplicando los conceptos enseñados en la materia.

# 2. Supuestos

1. Se asume que el orden de los jugadores en el juego sera el que se el orden el que se los registra. Es decir, este no cambiara como en el juego real en el que se tira un dado para ver quien empieza.

2. Los intercambios con el banco seran siempre 4 a 1, 3 a 1, o 2 a 1. Es decir, no podra cambiar 8 por 2 en un solo intercambio, se tendran que ejecutar varios intercambios seguidos.

# 3. Diagramas de Clase

## 3.1. Tablero

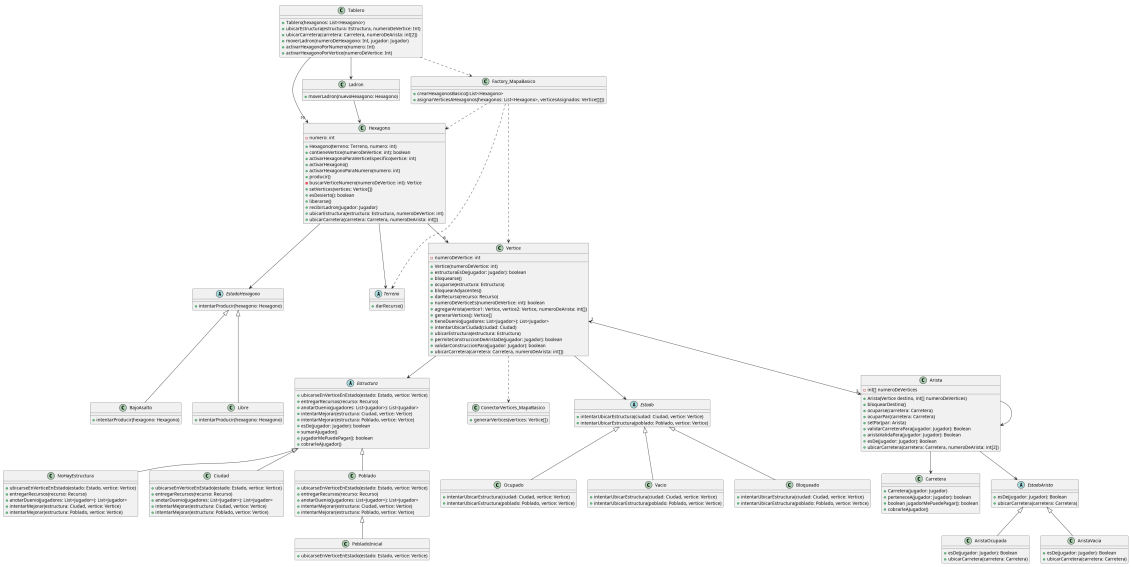


Figura 1: Diagrama de clases del Tablero

El Tablero del Catan esta constituido por 1 Ladron y 19 Hexagonos, que tienen 6 Vertices los cuales son compartidos con sus vecinos. Los Vertices tienen 2-3 Aristas cada uno, las cuales conectan 2 Vertices. Los Vertices tambien pueden tener o no una Estructura (Ciudad o Poblado), mientras que las Aristas son donde se ponen las Carreteras. Para el control de todas estas partes, usamos multiples Estados, que se encargaran de reaccionar a distintos eventos a su manera. Para la creacion controlada del Tablero y sus componentes, se usaron Factory_MapaBasico y ConectorGrafos_MapaBasico.

## 3.2. Terrenos

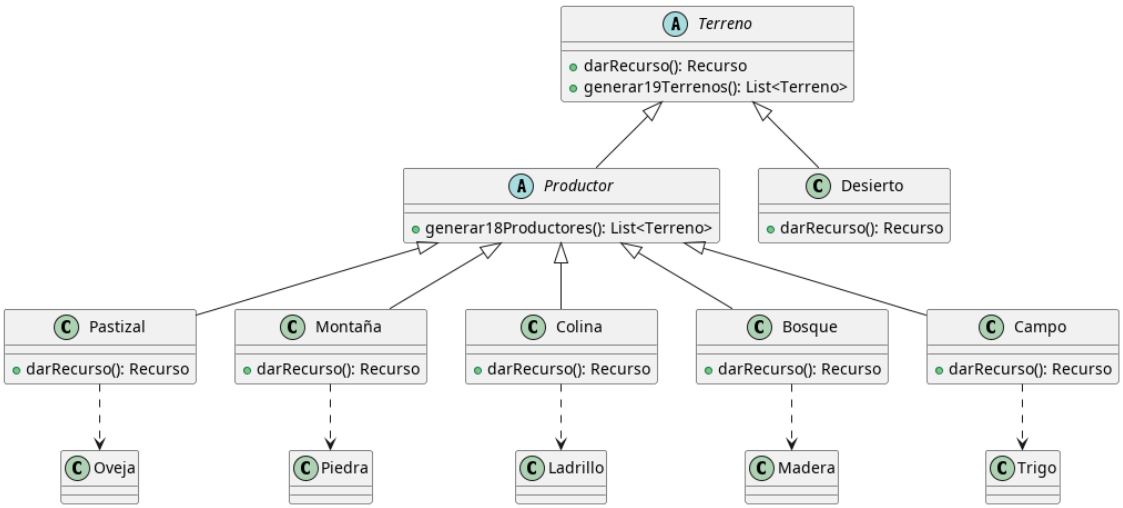


Figura 2: Diagrama de clases del Terreno

Los distintos tipos de Terrenos que tienen los Hexagonos del Tablero, son los que se ven en este diagrama de clases. En un principio existe la diferenciacion entre Terrenos Productores y Desierto, ya que mientras los Productores dan beneficios por ronda, el Desierto es inactivo todo el juego. Cada Productor da un tipo de Recurso distinto.

## 3.3. Banco y Recurso

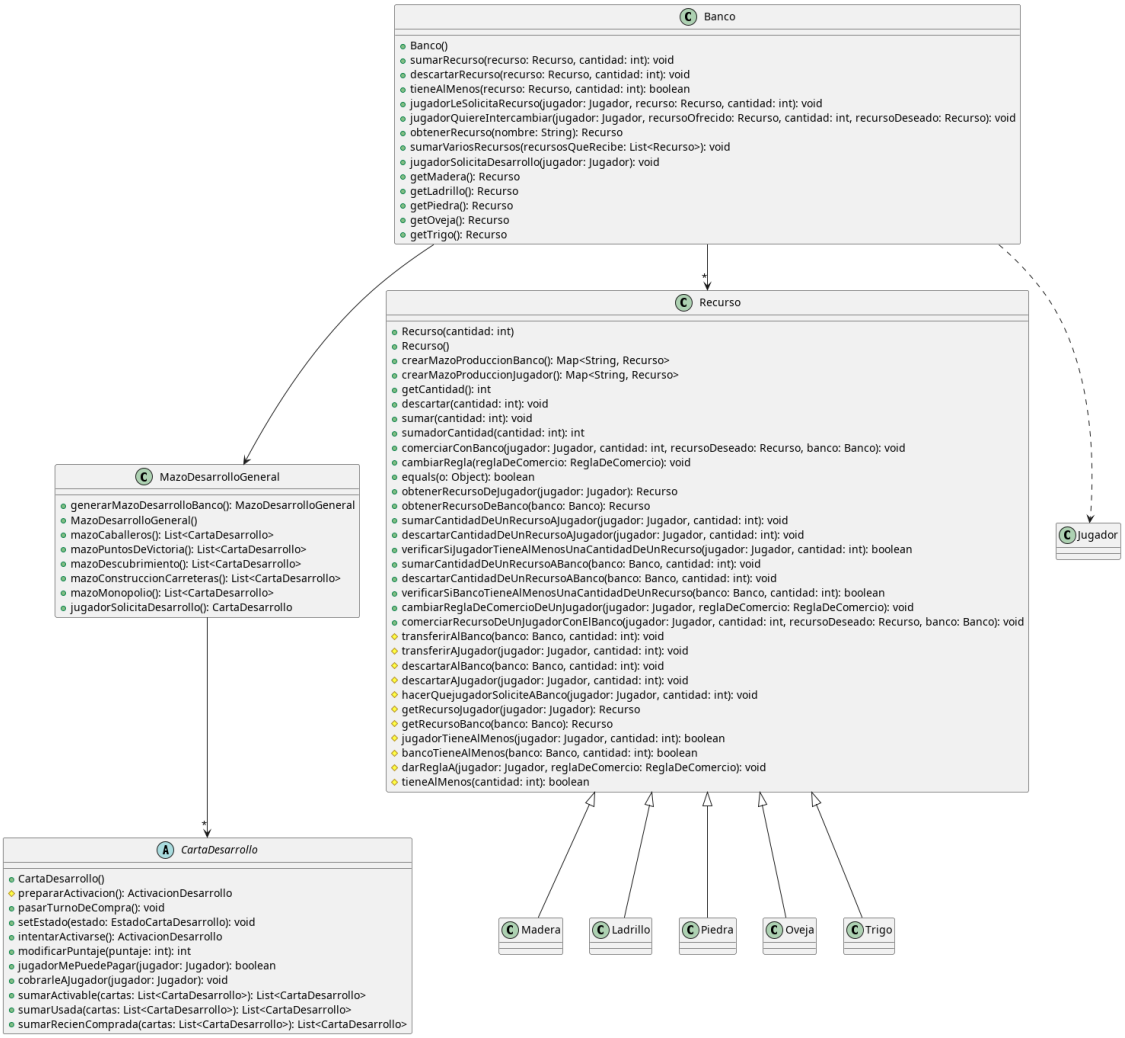


Figura 3: Diagrama de clases del Banco

Este grafico de clases muestra la clase Banco y sus colaboradores. Ademas, enseña por primera vez las acciones de los Recursos, aunque dejamos afuera los metodos reescritos de las hijas de Recurso para evitar que se extienda demasiado el grafico. El unico Banco de la partida tiene como labor dar y recibir Recursos y vender las Cartas de Desarrollo, las que almacena en MazoDesarrolloGeneral. El Banco puede hacer uso del Jugador si este le es pasado como parametro.

## 3.4. Recurso extendido

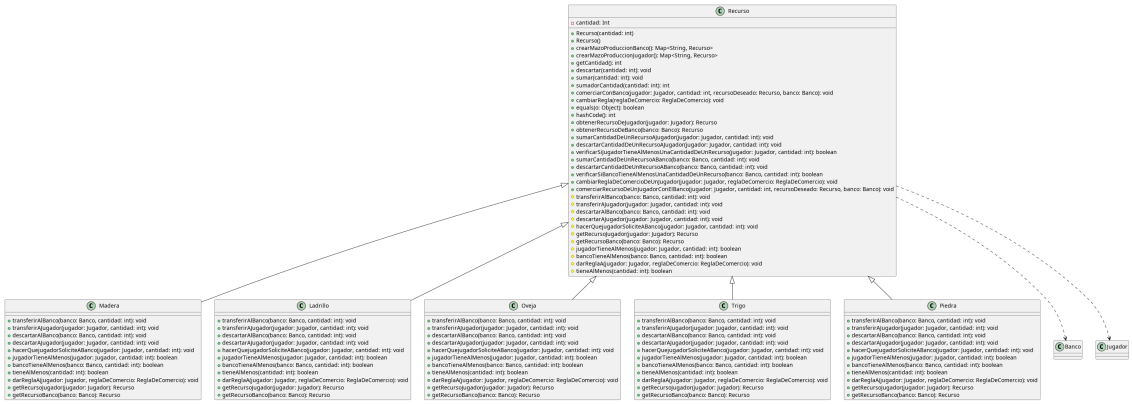


Figura 4: Diagrama de clases de los Recursos

Aca se puede apreciar mejor como los Recursos e hijas se encargan de gestionarse a si mismas y a sus pares mediante Polimorfismo. Haciendo uso de Jugador y Banco como parametros, estas pueden modificar los valores de los Recursos de cada identidad para movilizar la economia del juego. Hay muchos metodos redeclarados en este segmento, y eso es para poder aplicar Double Dispatch y sobrecarga para poder aislar el comportamiento de los recursos, reduciendolo a mas mensajes que se suman para lograr una accion.

## 3.5. Comercio

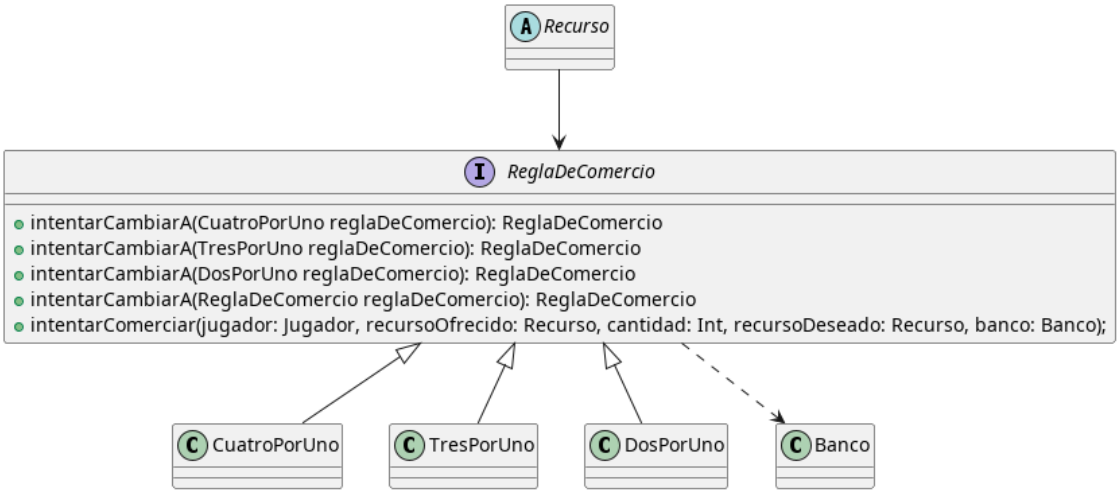


Figura 5: Diagrama de clases del Comercio

Ademas, los Recursos de cada Jugador cuentan con una Regla de Comercio, que sera utilizada cuando un Jugador quiera comerciar con el Banco. Mediante los puertos (proximo grafico), los Recursos de los Jugadores pueden adquirir mejores Comercios.

## 3.6. Puertos

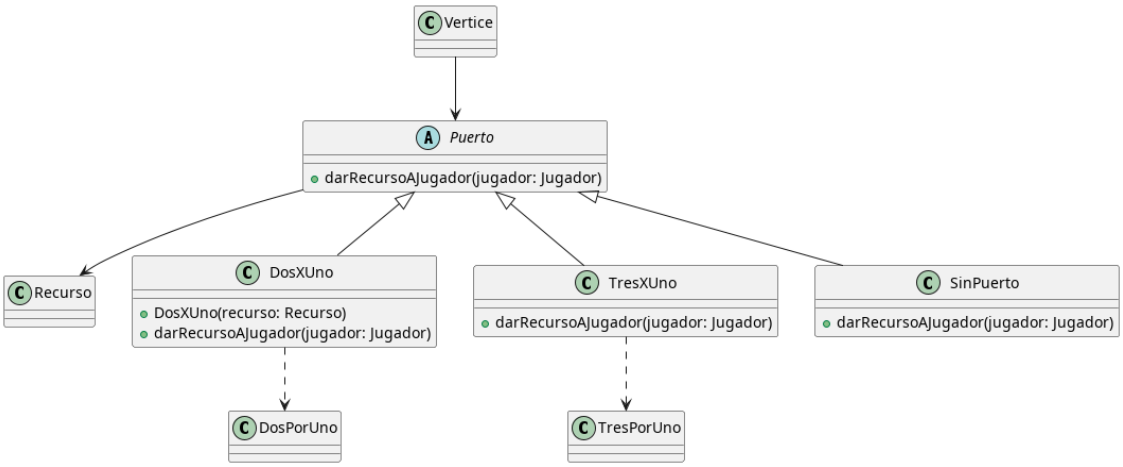


Figura 6: Diagrama de clases de los Puertos

En el tablero, todos los Vertices tienen un atributo Puerto (este puede ser SinPuerto, que reemplaza el nulo). Cuando un jugador ocupa el vertice con un poblado, le pide a su puerto que le de la Regla correspondiente al Recurso correspondiente del Jugador, para que cuando este quiera negociar con X Recurso, este tenga una nueva Regla.

## 3.7. Desarrollo

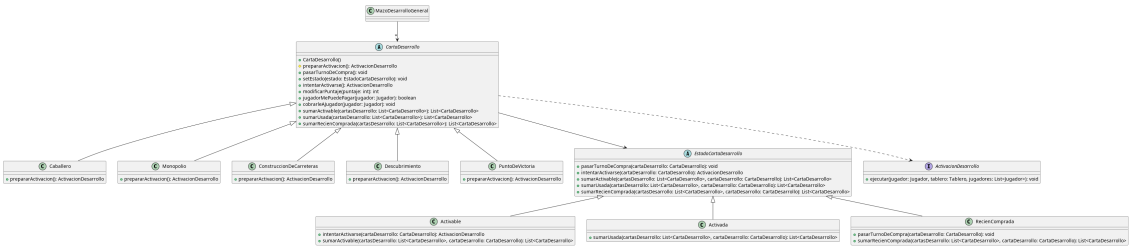

Figura 7: Diagrama de clases de las cartas de desarrollo

Breve descripcion del funcionamiento de las cartas de desarrollo. Estas cuentan con un estado, que se actualiza despues del primer turno y al ser usadas. De este depende si pueden ser activadas. Hay 5 tipos de carta de desarrollo, los 5 retornan un ejecutable como unica accion.

## 3.8. Estructuras

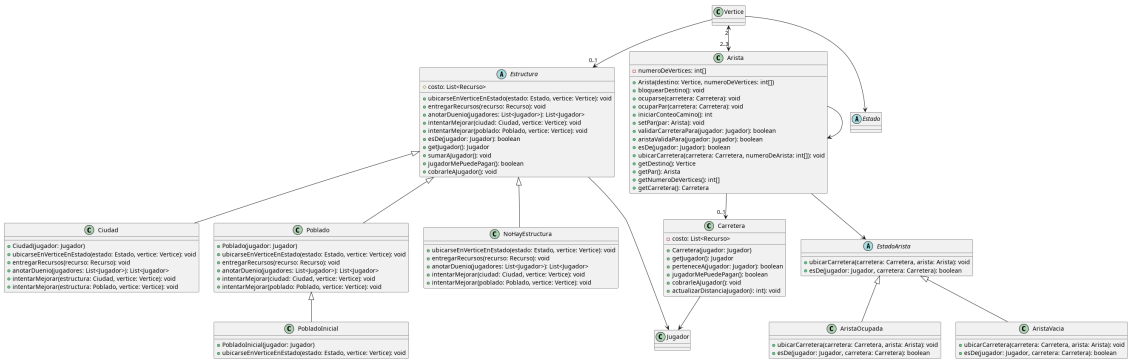


Figura 8: Diagrama de clases de los construibles del juego

Lo previamente mencionados Vertice y Arista pueden tener una Estructura o Carretera almacenada. Estas son las que los Jugadores compran para expandir su territorio productivo y ganar puntos para ganar. Para gestionar las opciones de los vertices y aristas, se usaban los Estados y EstadoArista (ambos explicados en el diagrama de tablero), y estos se van a encargar de permitir la ubicacion de las construcciones en sus dueños. Los distintos tipos de Estructura van a producir cantidades diferentes, y permitir o no ser reemplazadas por otro tipo de Estructura. Lo caminos son solo para expandir el territorio en el que el Jugador puede construir, y tambien para alcanzar la ruta mas larga. Ambos construibles pueden cobrarle al Jugador que los posee, y lo retienen como atributo para atriburile sus recompensas mas adelante.

## 3.9. Jugador

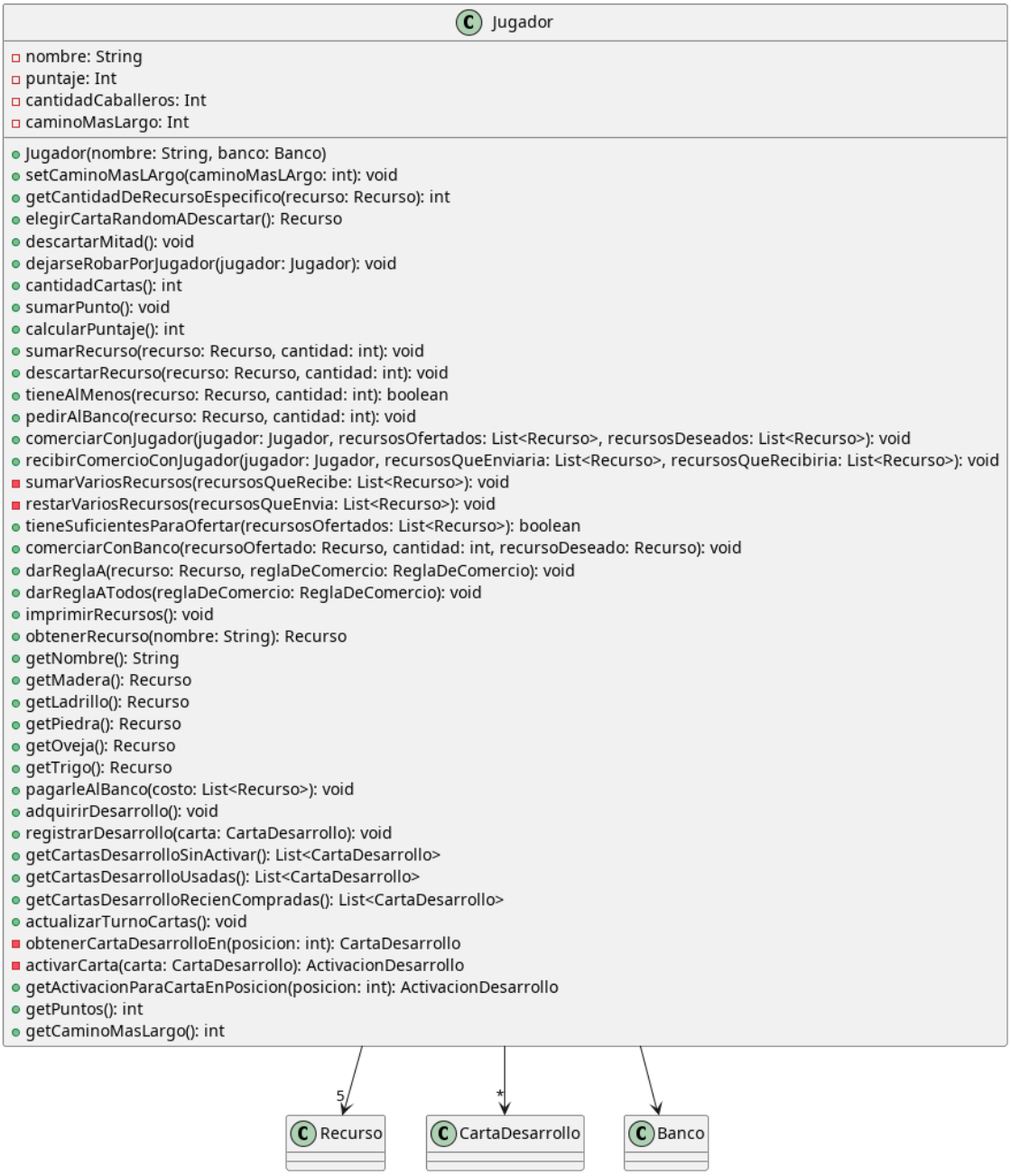


Figura 9: Diagrama de clases del Jugador

El Jugador es el protagonista del juego, por ende tambien es la clase mas cargada de todas. en el reside la responsabilidad de gestionar todos los puntos, Recursos y demas valores internos que se usan para darle forma al juego, como el camino mas largo. Varios de estos metodos estan para darle sentido a la estructura polimorfica de Recurso, y varios otros para darle acceso a la interfaz grafica.

## 3.10. Juego

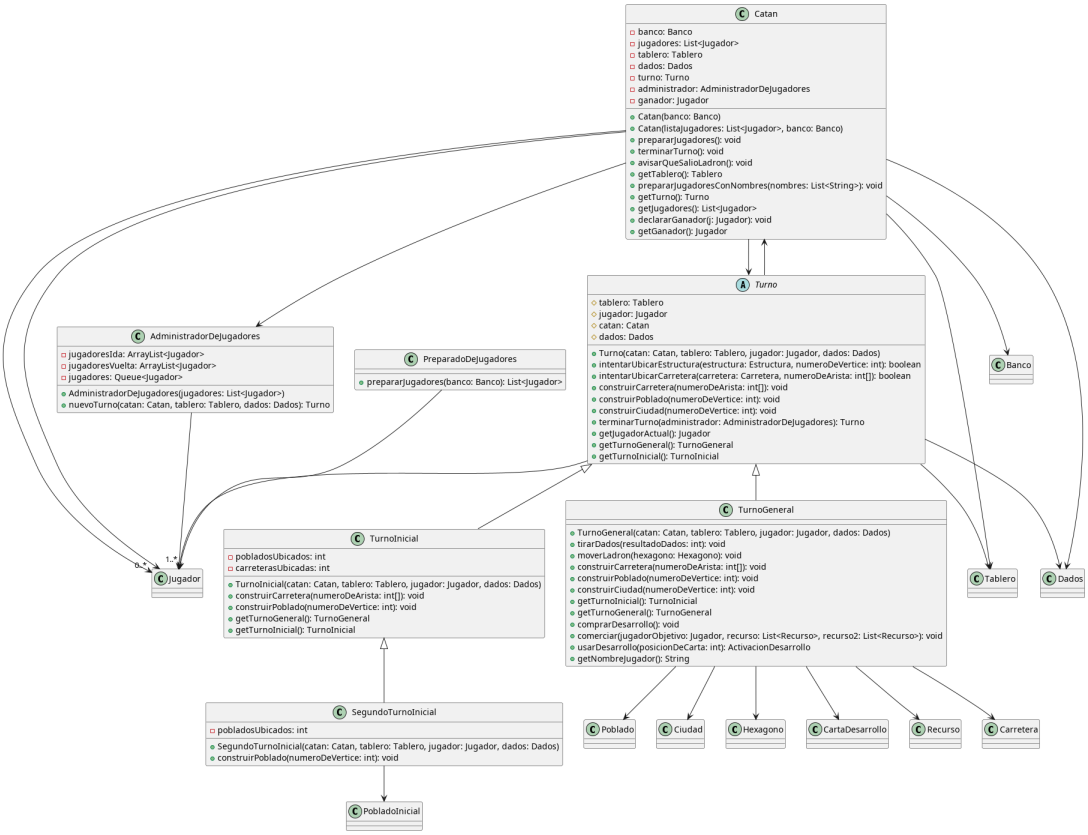


Figura 10: Diagrama de clases del Juego

La estructura del Juego es la siguiente: Un Catan que tiene un Turno, el cual dependiendo de la etapa del juego puede ser de distintas clases. El tipo de Turno que tendra Catan (y por ende la etapa del juego) sera determinada por AdministradorDeJugadores, y cada tipo de turno tendra metodos unicos dependiendo de las acciones que se pueden hacer en este. Por ejemplo, el SegundoTurnoInicial es un tipo de TurnoInicial (o sea que se puede ubicar un poblado sin regla de adyacencia), y encima este poblado produce inmediatamente.

## 3.11. General

Aqui un diagrama que reune todos los puntos recorridos previamente.

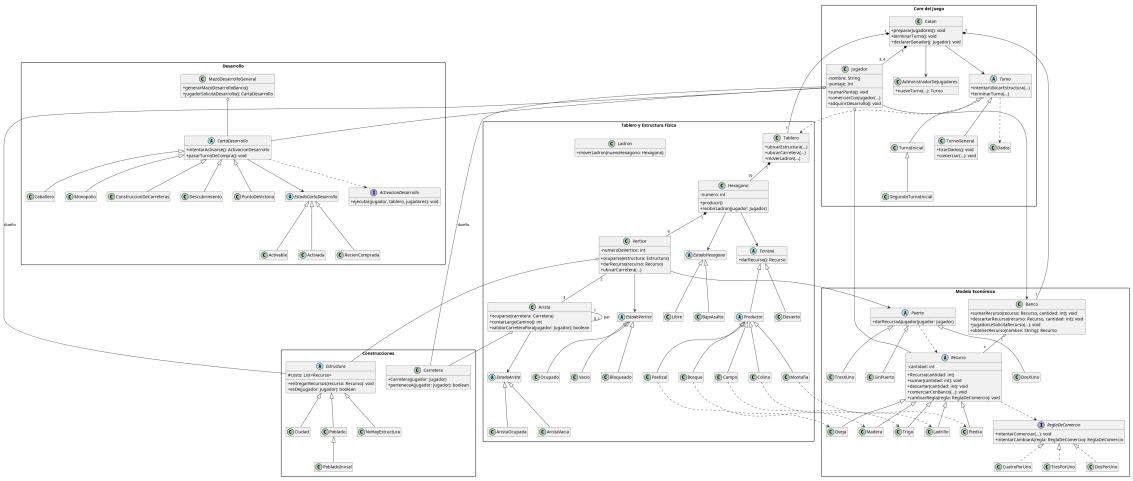


Figura 11: Diagrama de clases general

# 4. Diagramas de Secuencia

## 4.1. Tablero

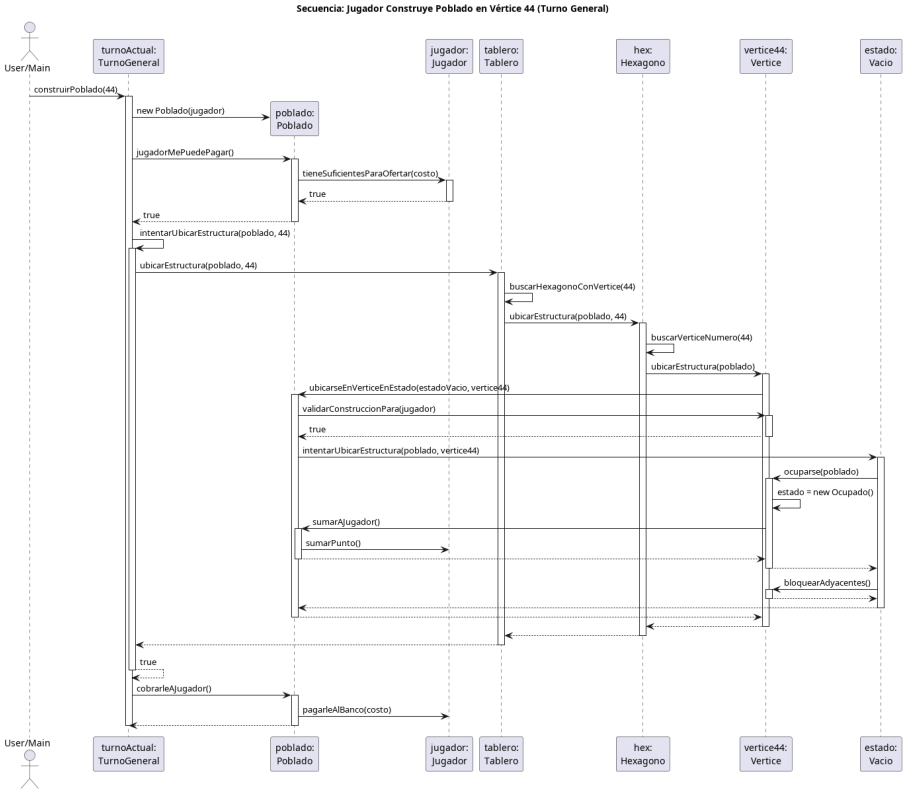


Figura 12: Diagrama de secuencia: se pone un poblado

Ejemplo exitoso de como se ubica un poblado.

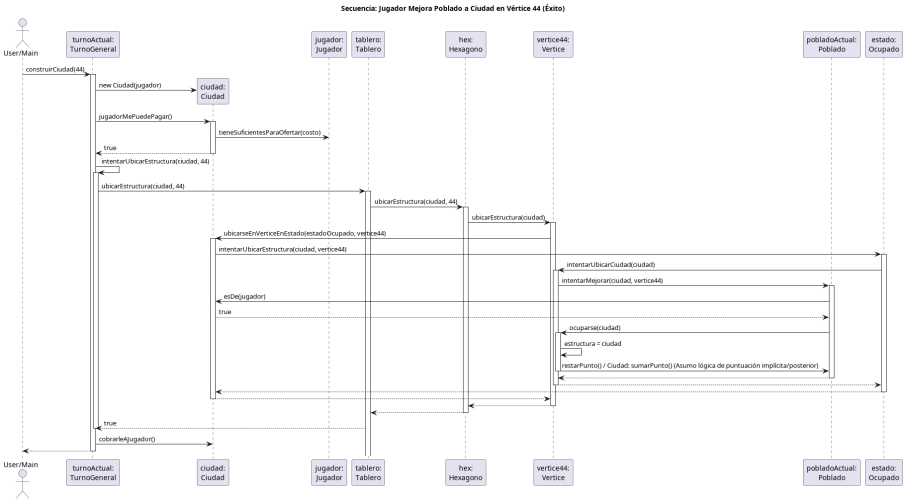


Figura 13: Diagrama de secuencia: se pone una ciudad

Ejemplo exitoso de como se ubica una ciudad.

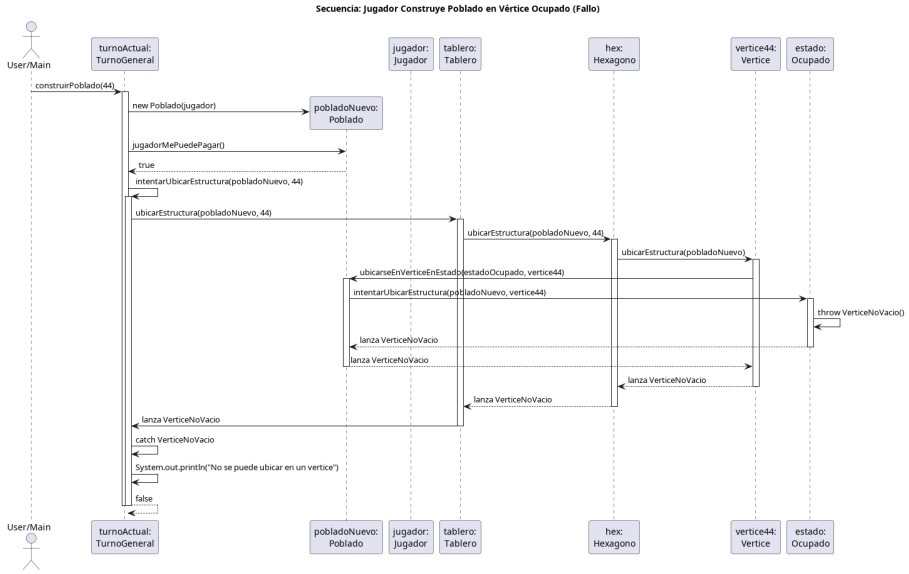


Figura 14: Diagrama de secuencia: se pone un poblado donde ya hay

Ejemplo de excepción donde se intenta colocar un poblado donde ya hay.

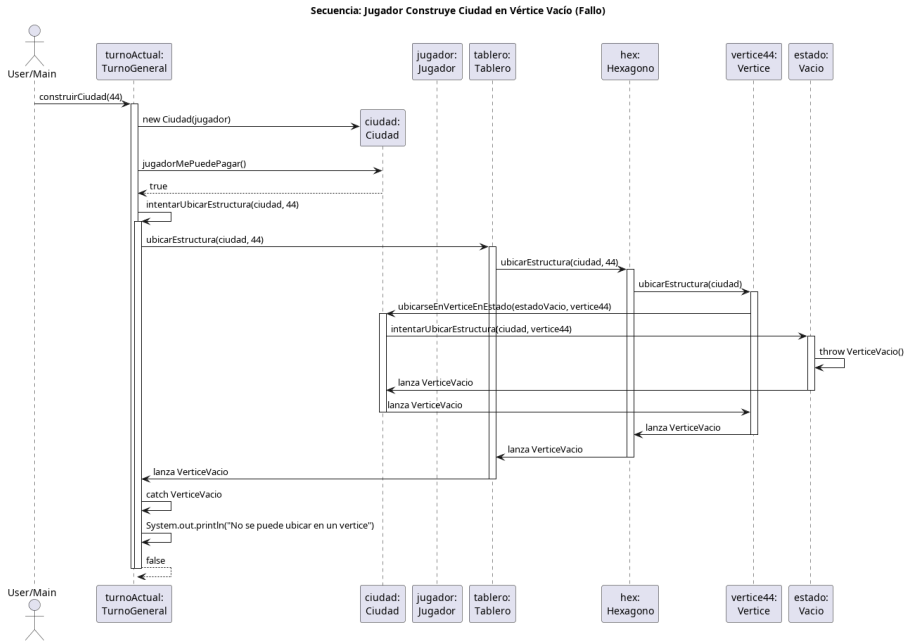


Figura 15: Diagrama de secuencia: una ciudad donde no hay poblado

Ejemplo de excepción donde se intenta colocar una ciudad sin un poblado previo.

## 4.2. Producción (tiro de dados)

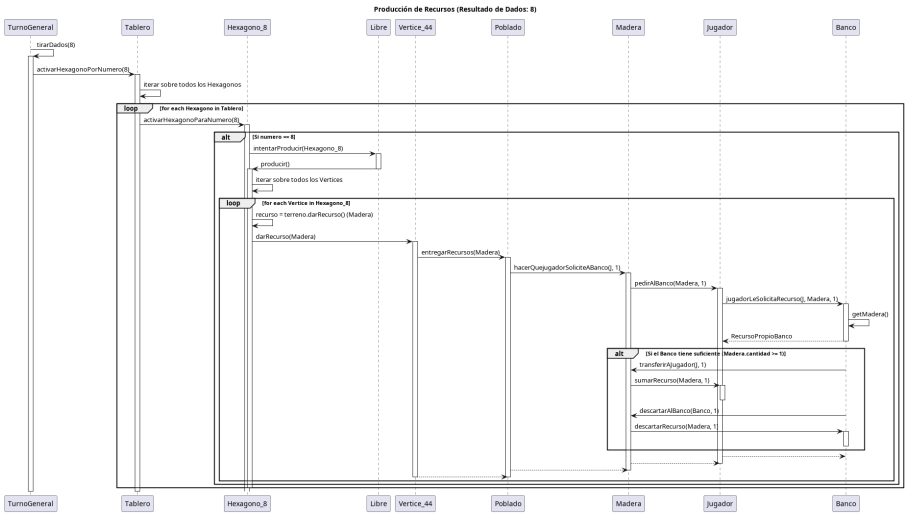


Figura 16: Diagrama de secuencia: ejemplo de produccion general

Ejemplo generalizado de como se producen los recursos mediante el tablero.

## 4.3. Comercio

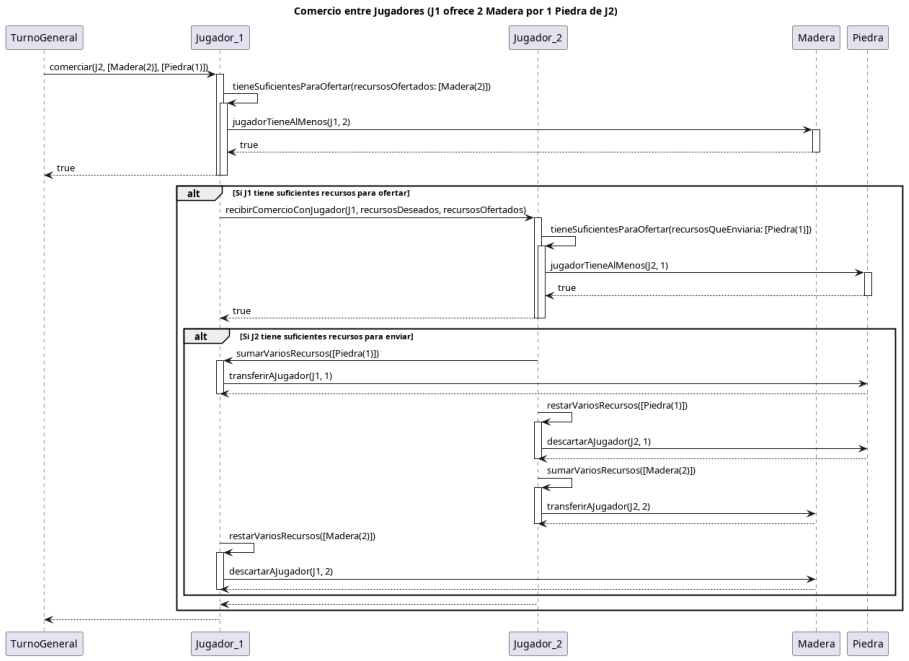


Figura 17: Diagrama de secuencia: Dos jugadores comercian

Ejemplo exitoso en donde dos jugadores van a comerciar.

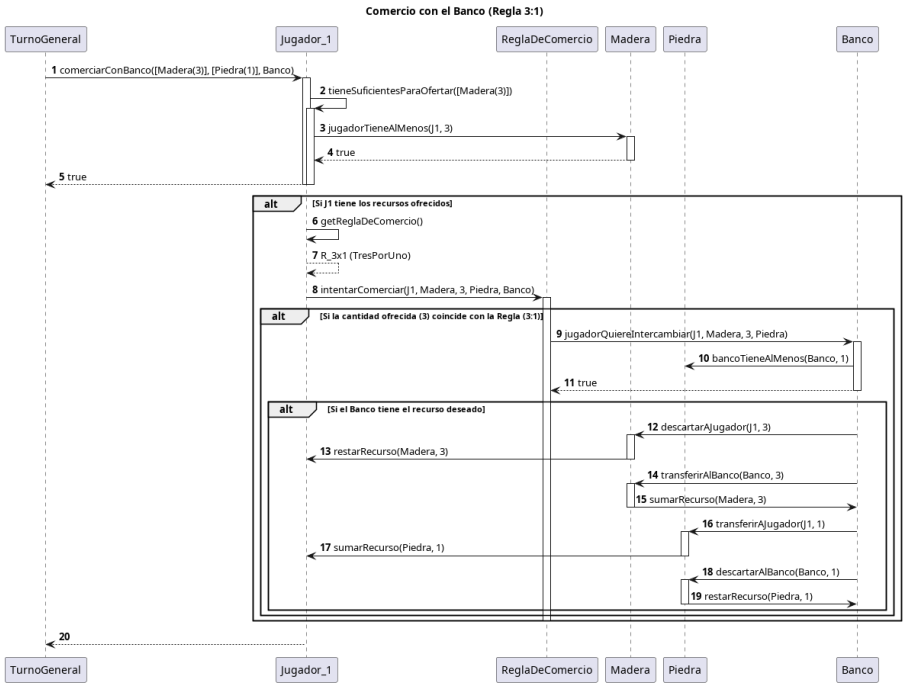


Figura 18: Diagrama de secuencia: comercio con el banco

Ejemplo especifico donde el jugador intenta comerciar con el banco (con puerto de por medio).

# 5. Diagrama de Paquetes

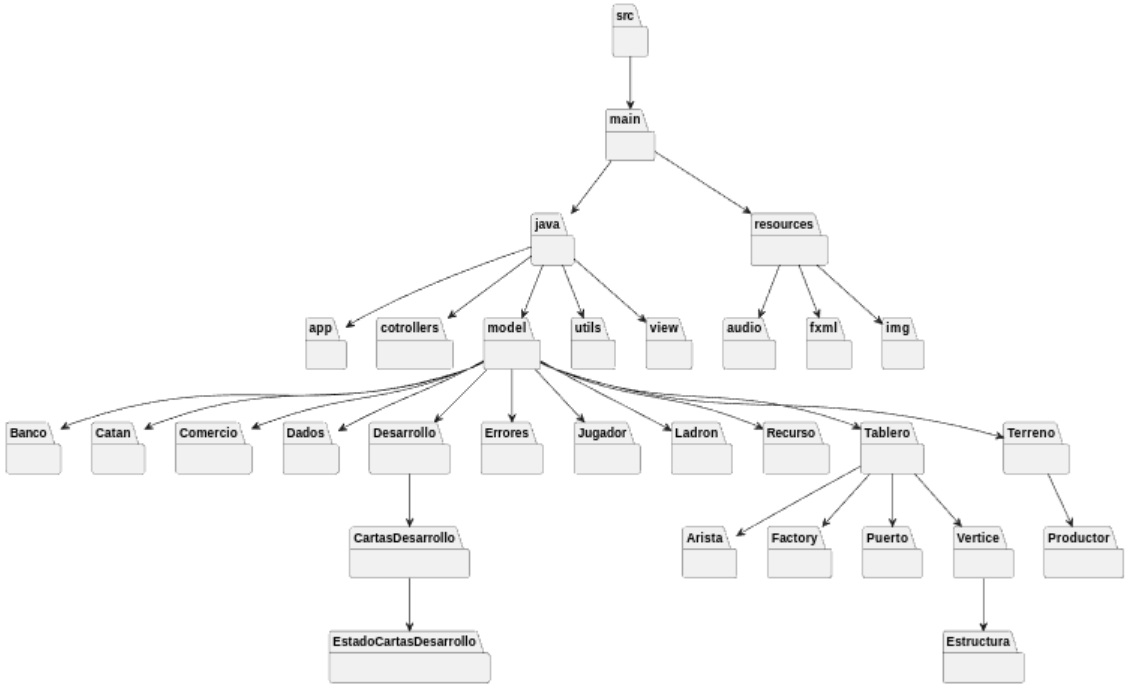


Figura 19: Diagrama de Paquetes del proyecto

Este diagrama de paquetes representa la estructura del proyecto, incluyendo modelo, controladores, y demás recursos necesarios. El paquete de tests va por fuera del src, por lo que no fue incluido. Si bien varias de las subdivisiones de los paquetes fueron hechas por una necesidad de organizacion, lo cierto es que estos paquetes encapsulan aspectos especificos, lo que facilita la expansion y coordinacion del juego.

# 6. Detalles de Implementacion

## 6.1. Double Dispatch y Sobrecarga

Uno de los puntos mas dificiles del TP, fue crear un sistema que pudiera gestionar la economia del juego, es decir, de los Recursos. Para esto teniamos que hacer algo que pudiera ser en cierto modo .ªlmacenado", transferido y comparado, tres cosas que parecen naturalmente opuestas al paradigma. Para solucionar esto, optamos por pensar los Recursos del juego como çantidades"del Recurso, es decir una Madera no representa una unica Madera, sino la cantidad de Maderas que tiene el Jugador. Para poder manipularlas, combinamos Double Dispatch y Sobrecarga para que cada Recurso pudiera identificarse entre los Recursos del Jugador y le delegase a este que hacer, por ejemplo descartar cierta cantidad. Esto se demuestra en uno de los diagramas de secuencia.

## 6.2. Strategy ’’ State

Por otra parte, notaran tambien que el Tablero cuenta con Estados en varios componentes, esto para implementar cosas basicas del juego como la regla de las distancias o de adyacencias. En estos casos usamos una combinacion entre Strategy y State, para que el Vertice (por ejemplo) pudiera delegarle la reaccion a un mensaje externo a su estado, y este se comunicara de nuevo con el Vertice para modificarlo como consecuencia. Hay estados con este patron en Vertice, Hexagono, Arista y CartaDesarrollo.

## 6.3. Observer

Tambien, usamos los principios basicos del patron Observer, de mucha utilidad para la interfaz grafica. Si bien no se usa para toda la interfaz, se usa para actualizar algunos de los factores que afectan como se ve la interfaz en dado momento, sin tener que forzar una asociacion incomoda entre interfaz y modelo.

## 6.4. Null Pattern

En algunas partes de nuestro modelo, se presentan situaciones donde una clase puede tener o no otra como atributo. Si bien en alguna excepcion habremos usado null, preferimos usar el Null Pattern, para evitar comparaciones del tipo equals(null). No es una aplicacion exacta del patron, pero se aplica en partes como Estructura o Comercio, adaptando la idea del patron al modelo y sus necesidades.

## 6.5. Factory

Si bien el mapa requerido es de solo un tipo, sabemos de la existencia de expansiones para el juego. Por esto mismo, y para mayor facilidad al testear el juego, decidimos crear un Factory del tablero, que por mas que solo tiene una implementacion, esta armado de manera que tolera nuevas formas y distribuciones.

## 6.6. Herencia, Polimorfismo y Delegacion

Las Herencias mas obvias se encuentran en Recurso, en Turno, y en CartaDesarrollo. Sin embargo, se usa a lo largo de toda la implementacion para limitar el uso de los condicionales y facilitar el trabajo en conjunto. Por ejemplo, se usa de manera un tanto obvia en ReglaDeComercio, donde una interfaz declara como una Regla debe actuar, y las hijas de esta Regla son las que terminan siendo usadas por el Recurso para comerciar. Todos los paquetes del juego, sin excepcion, usan Herencia y Polimorfismo de alguna manera. Por otro lado, la delegacion se cumple tambien en todo el juego, donde dejamos por ejemplo, que un estado se encargue de decidir si una ciudad puede o no ser construida, y que actue dependiendo de esto. Tambien se aplica cuando un Jugador

quiere comprar un Camino, por lo que le pide al Camino que le diga a sus Recursos que descartar, y el Camino le delega esto a los mismos Recursos.

## 6.7. Principios de diseño aplicados

El principio que se trabaja mas en el paradigma es el de Single Responsibility Principle. Si bien no fue aplicado perfectamente a lo largo de todo el modelo (hay clases como Jugador y Recurso que se encargan de varias cosas), logramos que las clases menos protagonistas como por ejemplo Terreno y ReglaDeComercio reduzcan sus existencias a una unica funcionalidad. Este principio se volvio una necesidad cuando el modelo empezo a escalar enormemente, y necesitabamos descongestionar las clases para que se entienda mejor la secuencia de cada operacion.

Otro de los principios mas utilizados fue el Open/Close, el cual fue mas facil de implementar, ya que no toma mas que una interfaz bien planificada. Nuevamente aplica el ejemplo de ReglaDeComercio y otros como Estructura, los varios Estados y tipos de Terreno. A la hora de expandir el modelo para que opere con muchso tipos que hacen lo mismo, el O/C resulta muy accesible y aprovechable.

Por ultimo, con frecuencia utilizamos el dependency inversion principle, aunque quiza sin saberlo. Esto debido a que utilizamos varias clases abstractas e interfaces (de la mano del Open/Close), lo que indirectamente nos llevo a una estructura controlada generalmente por abstracciones e interfaces, pero mas que nada abstracciones.

## 6.8. Excepciones

El modelo cuenta con 11 Excepciones en total. Brevemente repasaremos todas.

1. AristaEstaOcupada: Se lanza esta excepcion cuando un Jugador intenta construir una Carretera en una Arista con un EstadoArista del tipo AristaOcupada. Se usa con un try catch previo para no cobrarle al jugador si no se pudo colocar correctamente.

2. AristaFueraDeAlcance: Se lanza si un Jugador intenta construir una Carretera en una Arista a la cual no puede llegar, es decir, si no tiene una Carretera o Estructura adyacentes a la Arista en cuestion.

3. BancoNoTieneRecurso: Esta excepcion se utiliza cuando en la etapa de produccion, es decir cuando tiramos los Dados, el Banco se quedo sin Recursos de tal tipo para repartirle a los Jugadores que correspona. Es un error interno, que no tiene impacto por lo que podria ni estar, solo es para señalar.

4. DesiertoNoProduceNada: Cuando un Jugador pone su segundo poblado, se activan todos lo Hexagonos del juego para ese Vertice en especifico. Cuando este Vertice esta adyacente a un Desierto, este va a intentar producir, por lo que lanzara esta excepcion.

5. HexagonoBajoAsalto: Si un Hexagono tiene el ladron y sale su numero, la produccion sera detenida y se lanzara este error, puesto que un Hexagono con ladron no debe producir.

6. NoQuedanMasDesarrollo: Contrario a los Recursos, las Cartas de Desarrollo se pueden agotar, puesto que no retornan al Banco. Cuando un Jugador intenta comprar una Carta de Desarrollo y el Banco ya no tiene, este lanza esta excepcion.

7. VerticeFueraDeAlcance: Parecido a AristaFueraDeAlcance, si un Vertice intenta construir pero esta fuera del rando del Jugador, lanza esta excepcion.

8. VerticeNoVacio: Se lanza cuando un Jugador intenta construir un Poblado en un Vertice Ocupado

9. VerticeOcupadoPorAlguienMas: Se lanza si un Jugador quiere construir una Ciudad en un Poblado ajeno.

10. VerticeOcupadoPorCiudad: Se lanza si un Jugador quiere construir otra Ciudad encima de una Ciudad propia para generar mas puntos que no deberia tener.

11. VerticeVacio: Se lanza si un Jugador quiere construir una Ciudad en un Vertice sin ninguna Estructura, es decir sin Poblado previo. Del 7 al 11 inclusive, se usan para no cobrarle al Jugador y no ubicar su Estructura.