FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2
AlgoPoly

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

Alumno	Padrón	Email
Simizu, Nicolás Martín	94556	nicosimi993@gmail.com
Quino, Julian	94224	julian.quino2@gmail.com
Choque Scipione, Franco	98756	franco.choquescipione@gmail.com

Índice

1. Introducción

El presente trabajo tiene la finalidad de aplicar los conceptos aprendidos durante el curso, para la resolucion de problemas, para ello, se tiene que desarrolar una aplicacion que implemente el clasico juego del MonoPoly

2. Supuestos

Durante la implementación, se contemplaron situaciones que no son provistas en las indicaciones de la aplicación. Una de ellas, por ejemplo, es la situación cuando el jugador cae sobre el casillero de salida. Dado que no se explicita que debería suceder, se optó por dejar el casillero sin ninguna acción a realizar.

También se considera que los casilleros no van a cambiar su estado a lo largo del juego, refiriéndose a que una propiedad permanece como tal desde que inicia el juego hasta su fin.

Otra situación donde se trabaja sobre un supuesto es el funcionamiento de construir casas u hoteles. Se asumió que sólo se le permite al jugador construir edificios en el casillero donde se encuentra al empezar el turno.

3. Modelo de dominio

Para la implementación de la aplicación se utiliza una entidad que dirija su desarrollo y coordine a las demás y otra que funcione como interfaz de usuario. La primera se la denominó Tablero y la segunda, algoPoly.

Para desarrollar las entidades que componen al juego se utilizan las siguientes entidades:

Tablero : Esta entidad representa el tablero del juego, conteniendo los casilleros, referencias a los jugadores y estructuras(Casas y hoteles)

Casillero : Esta entidad representa a cada casillero del juego y depende de su posición en el tablero, cambiará su comportamiento.

Jugador : Esta entidad representa a un participante en la aplicación.

Dado : Esta entidad representa un dado del juego y tiene una única operación, obtener un número aleatorio del 1 al 6.

Casa y Hotel : Estas entidades representan las construcciones posibles para los casilleros que sean propiedades.

Con estas clases presentes, se busca proveer al usuario una manera sencilla de interactuar con la aplicación.

4. Diagramas de clase

En la figura 1 se presenta la relación entre las diferentes clases que componen la aplicación.

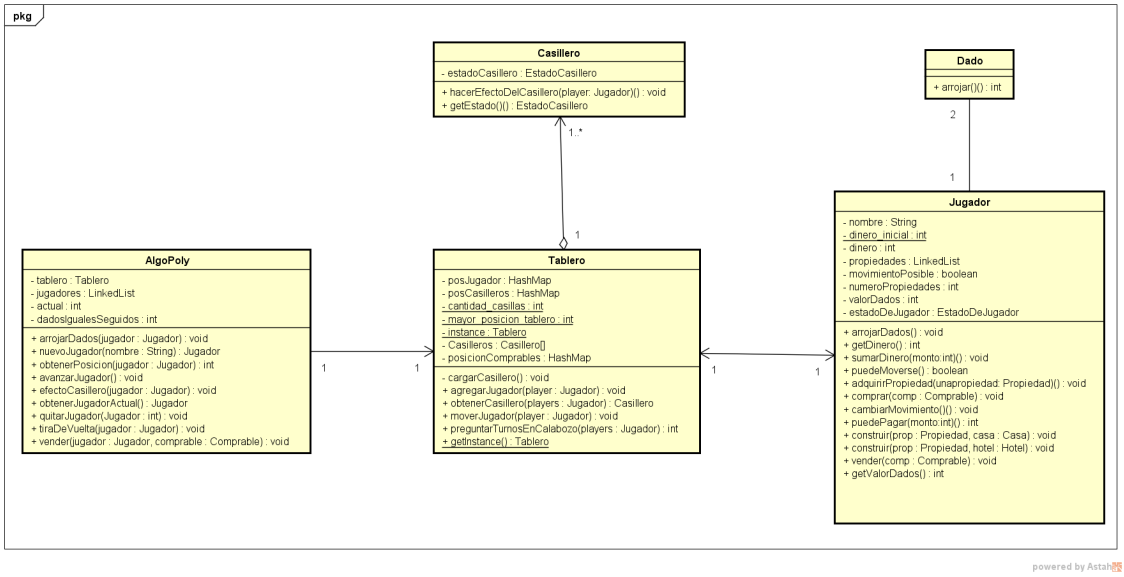

Figura 1: Diagrama de clases de algoPoly.

5. Detalles de implementación

5.1. Comportamiento de los casilleros

En la implementación de los diferentes comportamientos que puede tener las instancias de Casillero, se decidió por utilizar el patrón State, para simplficar como deberían enviarse los mensajes para ejecutar los efectos de cada casilleros.

Como se muestra en la figura 2, se tienen categorías de comportamientos de casilleros: están divididos en Comprables y No comprables. Los comportamientos Comprables heredan de EstadoCasillero y contienen los métodos y atributos necesarios para que un casillero pueda comprarse. Así mismo, de Comprables heredan las subclases Propiedades y Servicios, ya que los casilleros de Propiedades permiten construir edificios(Casas u hoteles, según corresponda el comportamiento) mientras que los Servicios no.

En las especificaciones de la aplicación también se pide que las casillas comprables puedan intercambiarse, si bien esto es posible hacerlo agregando las funciones necesarias en la interfaz, las pruebas pedidas para la entrega N1 de la aplicación no exigían este comportamiento.

También hay dos categorías que no se incluyen, Cárcel y Policía: son dos casilleros que no pueden integrarse. El comportamiento de policía es simplemente mover el jugador a la casilla Cárcel. Por su parte, Cárcel lo único que hace es retringirle el movimiento al jugador hasta que pague la fianza o llegue a estar 3 turnos sin moverse, luego le devuelve la capacidad de moverse al jugador.

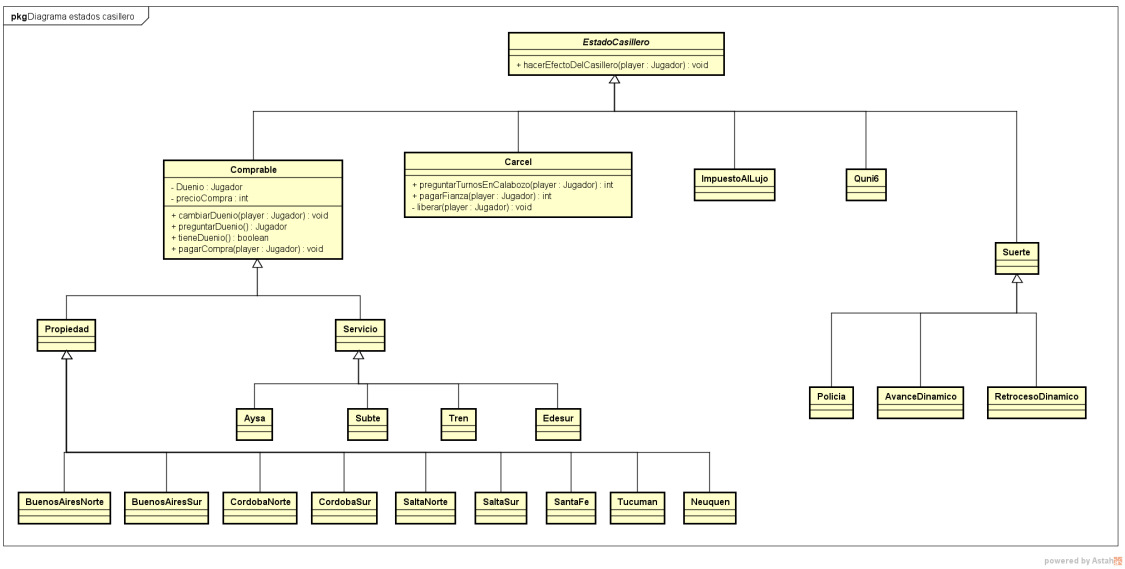

Figura 2: Diagrama de clases para el comporamiento del casillero.

5.2. Comportamiento de barrios.

En la figura 3 se muestra la implementación de los barrios con la clase Propiedad a la que se le hizo un Strategy. La manera de decidir entre cual de los comportamientos se encuentra es a través del tipo y cantidad de edificios en el barrio.

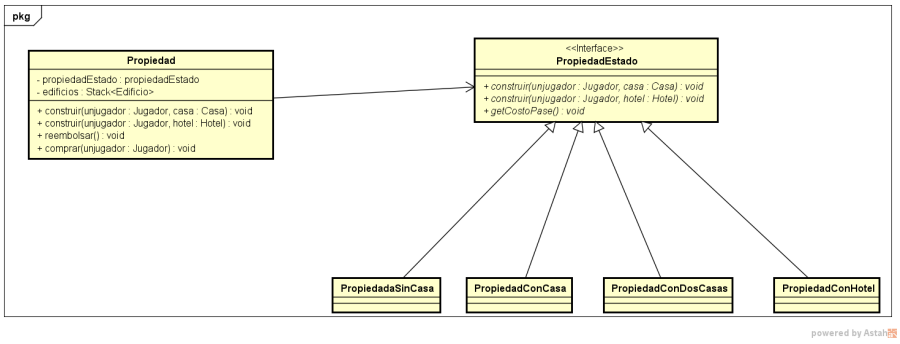

Figura 3: Implementacion del patron Strategy en propiedad.

5.3. Comporamiento de jugador.

En la figura 4 se muestra la implementación del turno del jugador, utilizando un Strategy. La clase jugador contiene un atributo estadoJugador que hace referencia a una clase que implemente la interfaz EstadoDeJugador que contiene el comportamiento requerido.

Inicializado el juego, el jugador que empieza tiene su estado en jugadorEmpezandoTurno. Cambia al estado jugadorTiroDados cuando se llama al método arrojarDados. Del estado jugadorTiroDados cambia al estado jugadorSinTurno luego de realizar el efecto del casillero donde se encuentre el jugador.

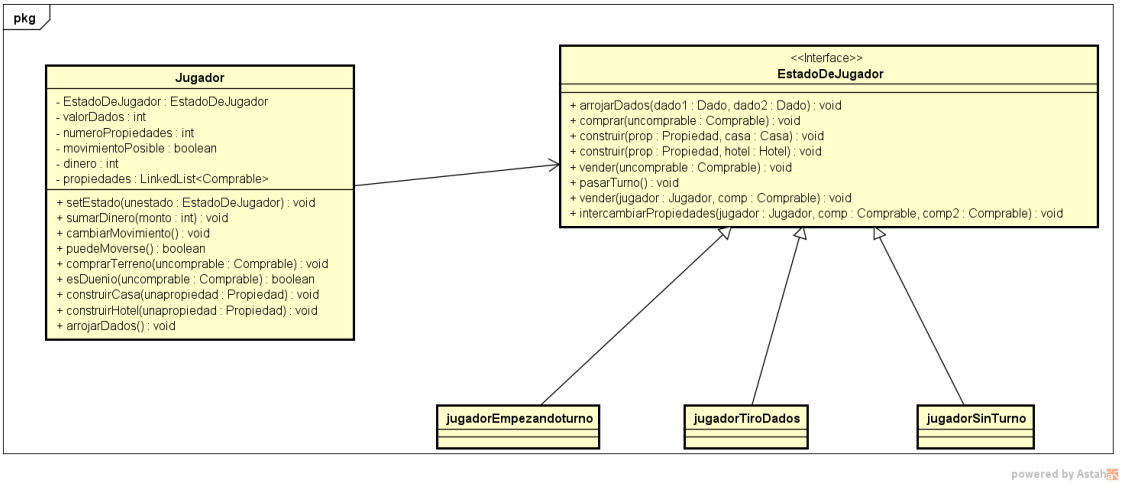

Figura 4: Implementacion del patron Strategy en jugador.

6. Excepciones

DineroInsuficiente Esta excepción se creó para los casos en que un jugador intenta pagar y no dispone de suficiente dinero.

TurnosEnCalabozoInvalido Esta excepción se creó para los casos en que los turnos que el jugador permaneció en la carcel no son válidos para realizar la acción.

JugadorYaTiroDados Esta excepción se creó para los casos en que al jugador no se le permite volver a tirar los dados.

JugadorNoTiroDados Esta excepción se creó para los casos en que al jugador no se le permite terminar su turno sin antes tirar los dados.

NoEsTurnoDelJugador Esta excepción se creó para los casos en que la acción no sea legal en el estado del jugador.

CasasInsuficientes Esta excepción se creó para los casos en que el jugador quiera construir un hotel, pero no se cumple el requisito de casas en los barrios.

JugadorEstaPreso Esta excepción se creó para los casos en que un jugador quiera moverse y se encuentra en la cárcel sin poder moverse.

JugadorNoEsPropietario Esta excepción se creó para los casos en un jugador quiera construir una casa en un barrio que no posee.

JugadorNoPoseeTodosLosBarrios Esta excepción se creó para los casos en que un jugador quiera construir una casa y no posea los barrios requeridos

NoPuedeConstruirMasCasas Esta excepción se creó para los casos en que un jugador quiera construir más casas de las permitidas por el barrio.

NoPuedeConstruirMasHoteles Esta excepción se creó para los casos en que un jugador quiera construir más hoteles de los permitidos por el barrio.

CasasInsuficientes Esta excepción se creó para los casos en que un jugador quiera construir un hotel y los barrios no tengan la máxima capacidad de casas construidas.

FinDelJuego Esta excepción se creó para poder lanzar detener el juego en caso de que solo quede un jugador.

NombreInvalidoException Esta excepción se creó para poder lanzar una ventana de error al validar los nombres de los jugadores en la interfaz gráfica.

7. Diagramas de estado

7.1. Estados de AlgoPoly

En la figura 5 se muestran los estados de AlgoPoly. Si bien el diagrama muestra pocos estados posibles, se programó la aplicación teniendo en mente que la clase AlgoPoly sera usada como una fachada para permitirle al usuario interactuar, de modo que facilite las decisiones del usuario en base a las opciones disponibles.

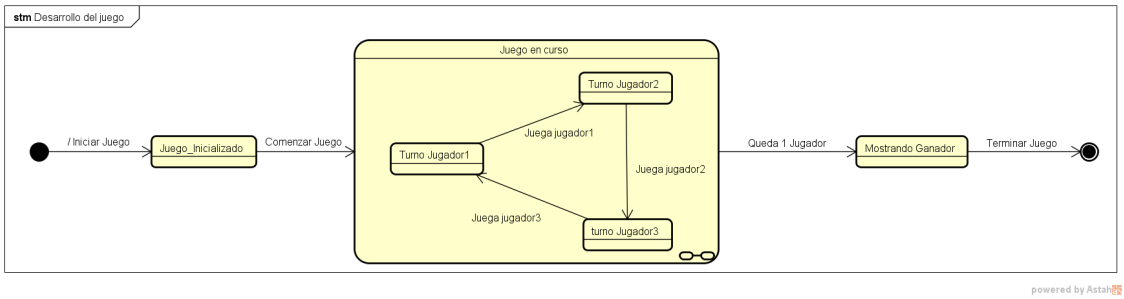

Figura 5: Estados por los que pasa AlgoPoly.

7.2. Estados de Jugador

En la figura 6 se muestran los estados del turno de la clase Jugador. Saber el estado en el cual se encuentra el jugador permite a la aplicación determinar si una acción es o no posible, por ejemplo construir casas.

Inicializada la aplicación, el jugador comienza el turno en el estado Empezando turno; mas avanzado el juego, se pregunta si el estado del jugador es Bancarrota(que representa que el jugador no pudo pagar un gasto y perdió). En el caso de ser bancarrota, se termina su turno y pasa al siguiente jugador.

En el estado EmpezandoTurno el jugador puede comprar o vender edificios e intercambiar propiedades. Si decide tirar los dados, el estado cambia a TiroDados.

En el estado TiroDados el jugador solo puede vender propiedades o edificios en caso de no disponer de efectivo suficiente para afrontar un gasto. Luego cambia al estado Sin Turno.

En el estado Sin turno, termina el turno del jugador actual y se cambia el estado del jugador siguiente a Empezando turno.

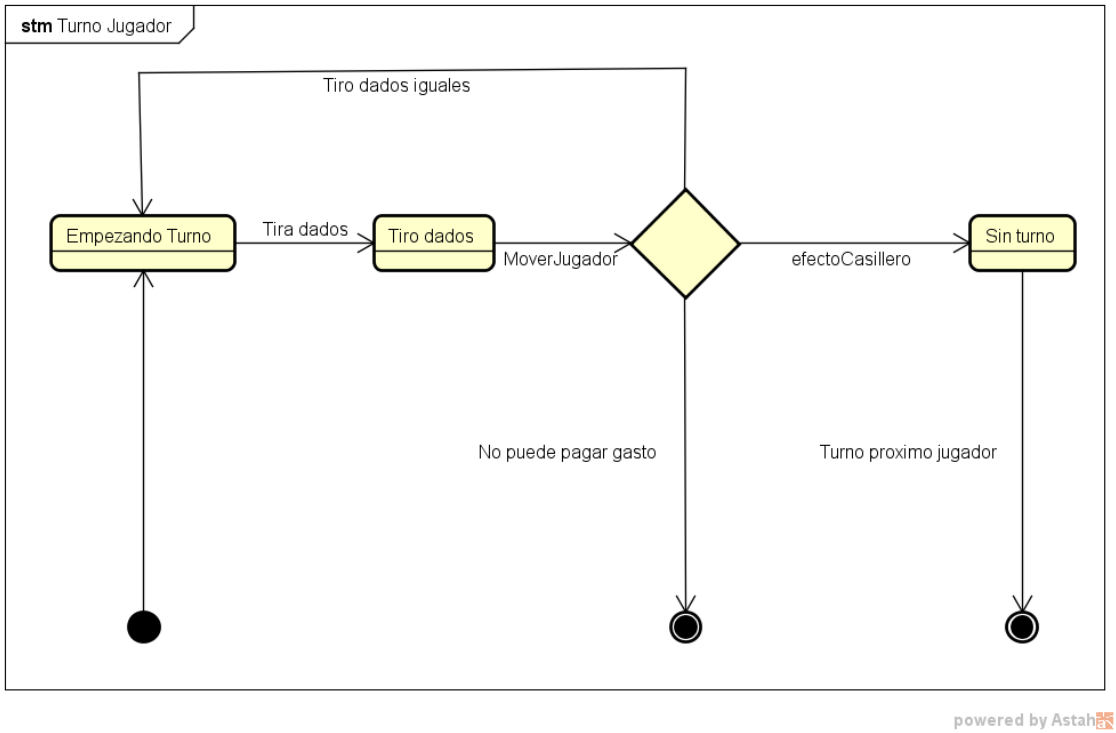

Figura 6: Estados por los que pasa el turno del jugador

7.3. Estados de barrios

En la figura 7 se muestran los estados de edificación de la clase Propiedad. El estado en que se encuentra la clase se determina según el tipo y la cantidad de edificios.

Inicializada la aplicación, siempre se encuentra el barrio en el estado Barrio sin Casas, ya que no pueden construirse edificios hasta que un jugador lo compre y cumpla con los requisitos de construcción. En ese caso, el jugador construye una casa y cambia el estado a Barrio con una casa.

En el estado Barrio con una casa, el barrio puede cambiar a Barrio con dos casas, si el jugador cumple los requisitos y construye una casa o volver a Barrio sin casa, si decide vender la casa.

En el estado Barrio con dos casas, el barrio puede cambiar a Barrio con hotel si cumple los requisitos o volver a barrio sin casa.

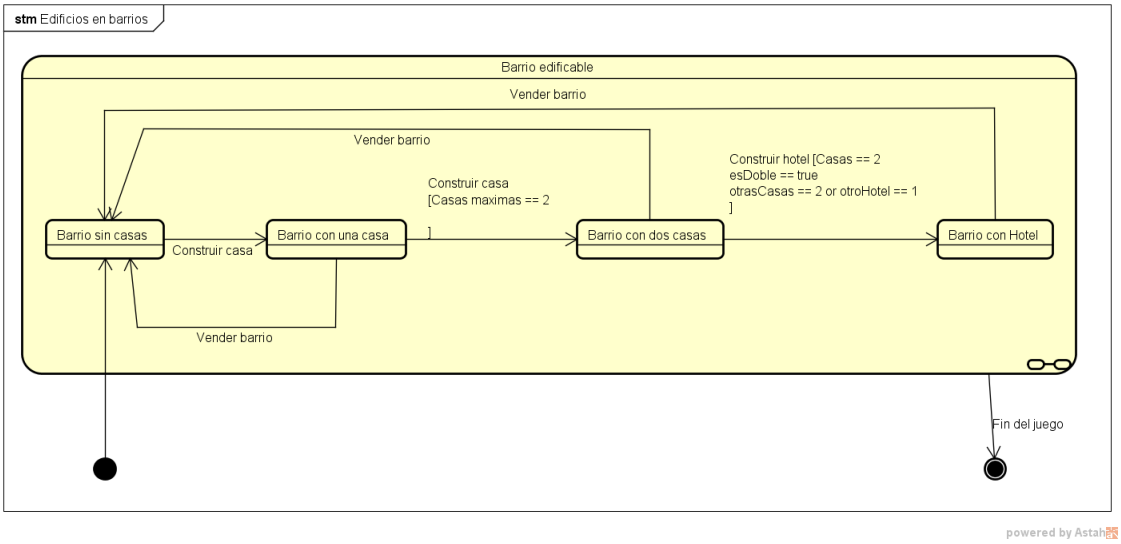

Figura 7: Estados por los que pasan los barrios edificables

8. Diagramas de paquetes

En la figura 8 se muestra el diagrama de paquetes de la aplicación. Se ajustó la distribución de las entidades requeridas por la aplicación según el patron MVC, que tiene por separado la vista, el modelo y el controlador.

La lógica del programa, las excepciones y como se implementan las clases para la aplicación se encuentran bajo el paquete Implementacion.programa. Este paquete representa la parte de modelo del MVC.

El paquete interfaz.usuario contiene los comandos que se le permiten ejecutar al usuario para modificar el modelo.

El paquete vista contiene todos los archivos de imagen y sonido requeridos para dar la representación gráfica del modelo.

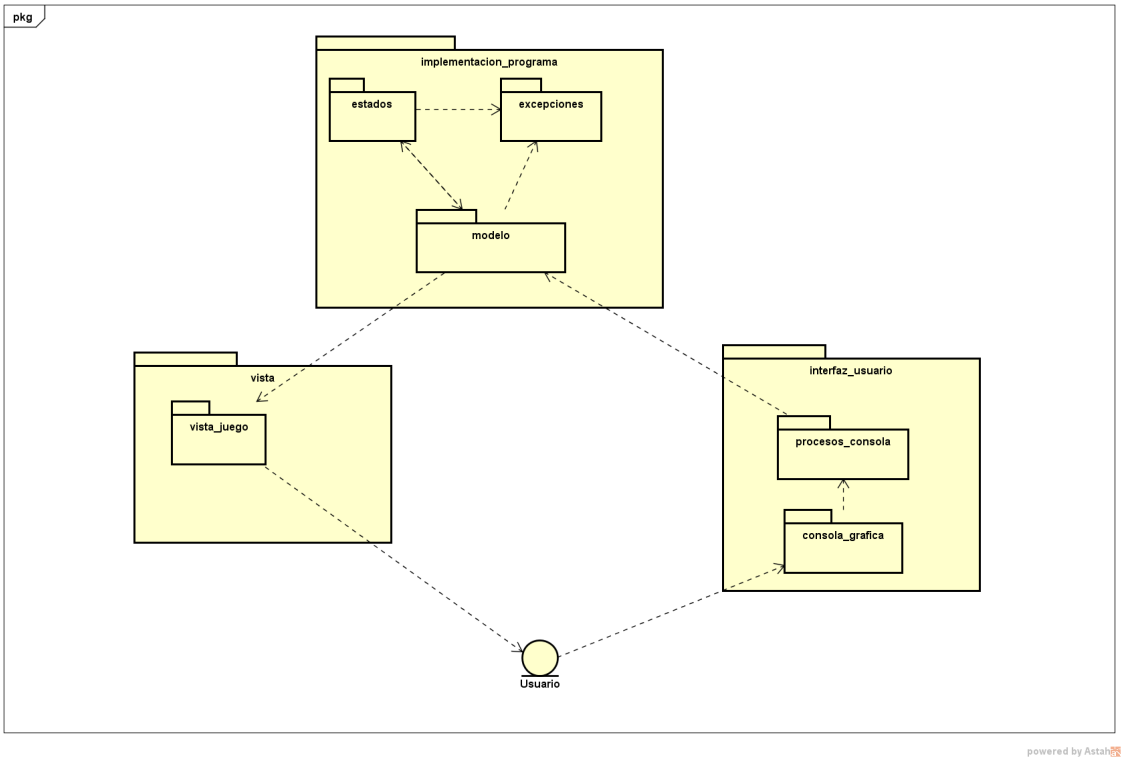

Figura 8: Diagrama de paquetes de algoPoly

9. Diagramas de secuencia

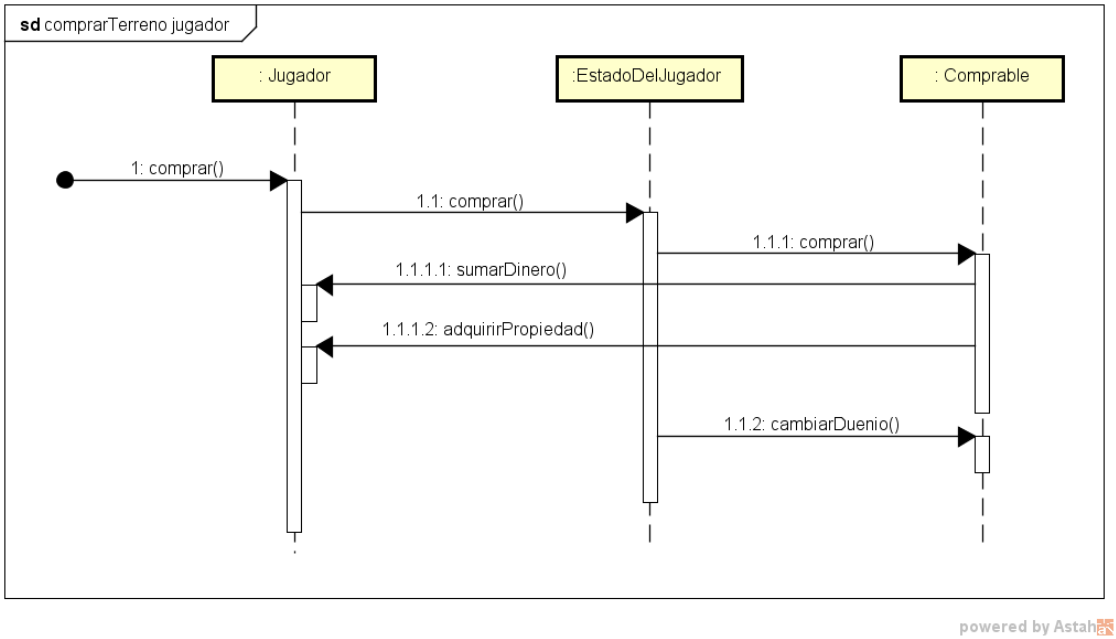

Figura 9: Secuencia del metodo comprar de Jugador.

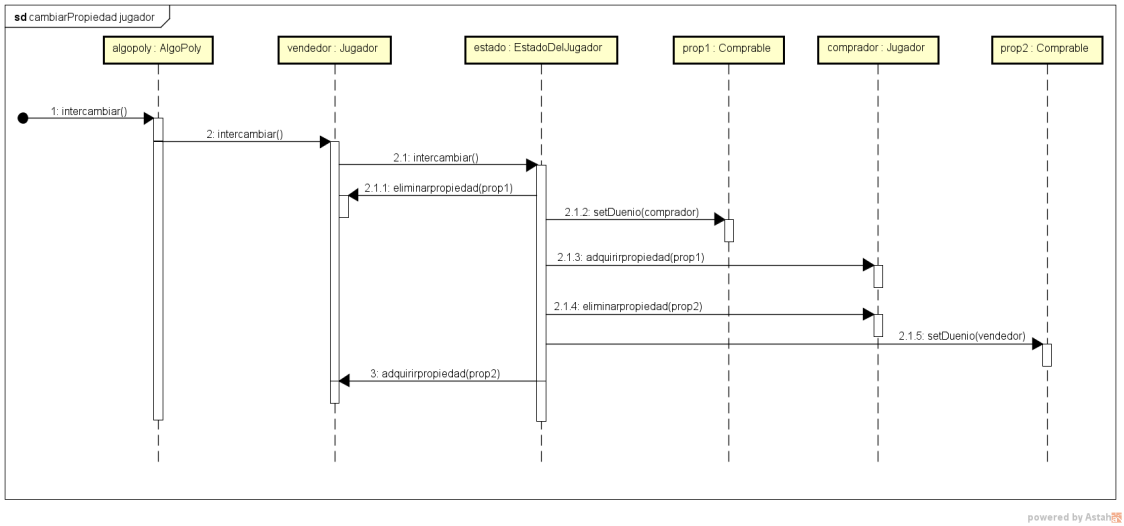

Figura 10: Secuencia del metodo intercambiarPropiedad de Jugador.

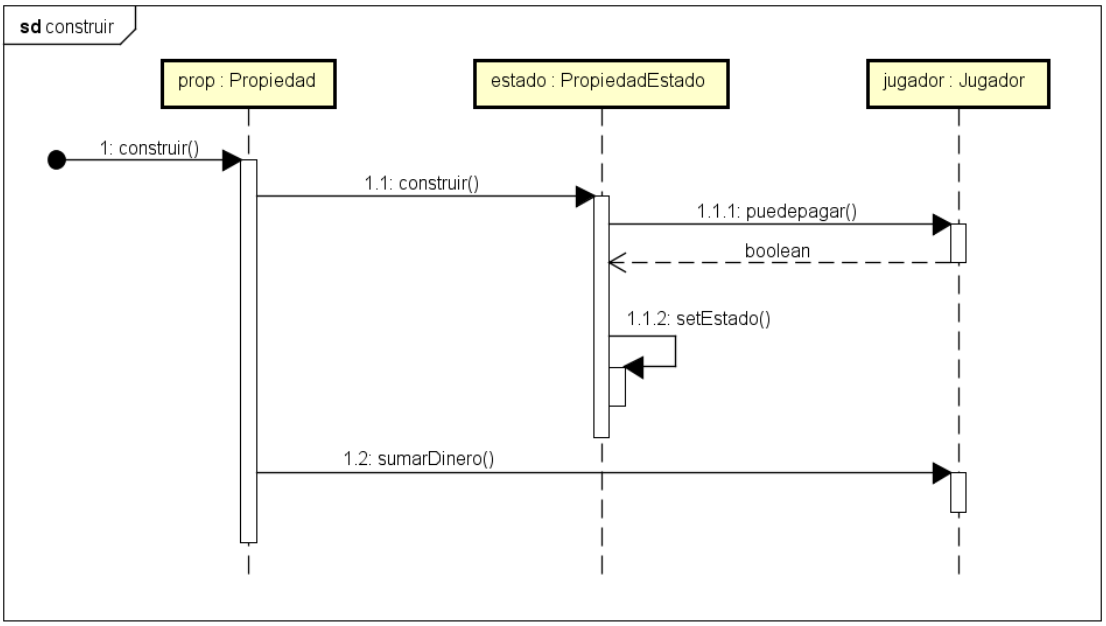

Figura 11: Secuencia del metodo construir de Propiedad.

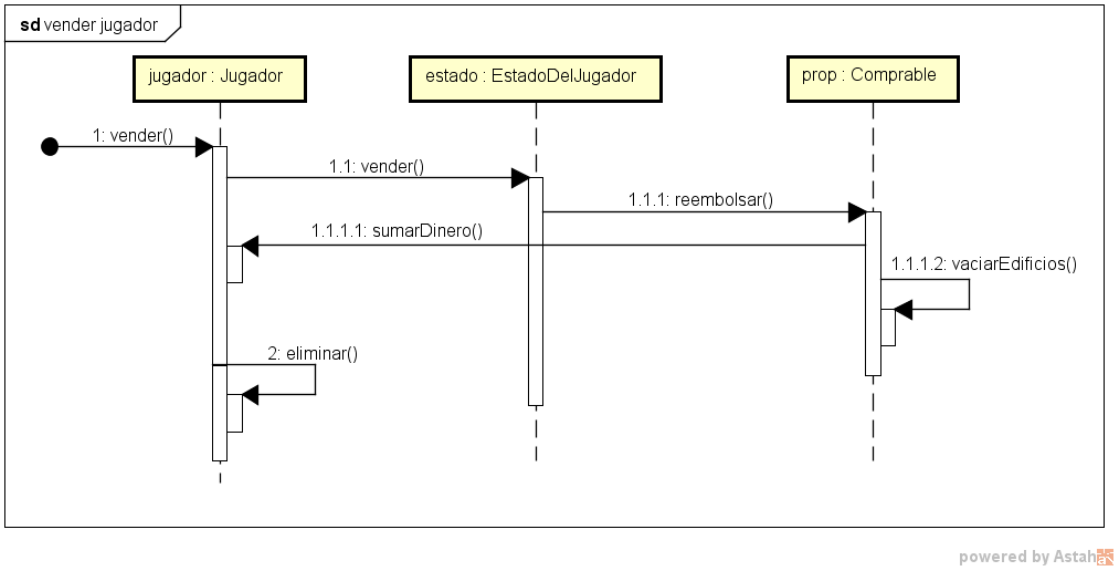

Figura 12: Secuencia del metodo vender de Jugador.

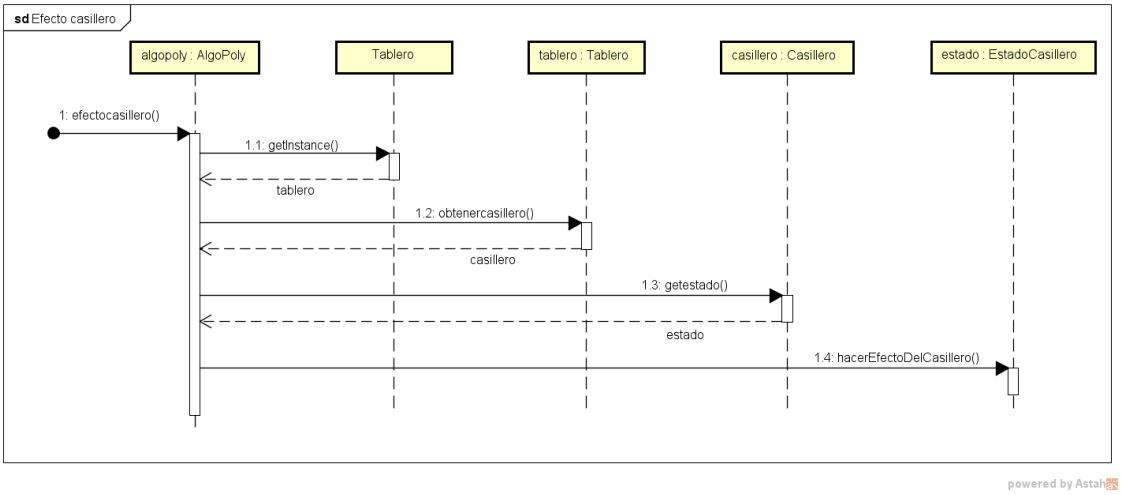

Figura 13: Secuencia del metodo hacerEfecto de AlgoPoly.

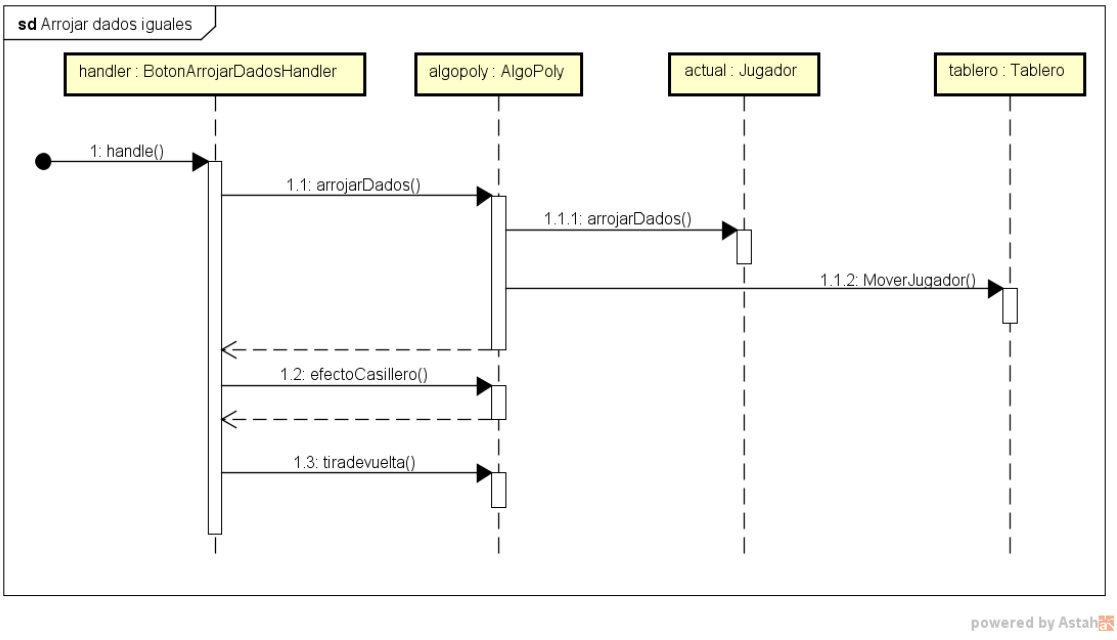

Figura 14: Secuencia del escenario arrojar dados iguales.

En la figura 14 se muestra la secuencia del turno del jugador cuando arroja los dados y ambos tienen el mismo valor. Solo se muestran las interacciones entre los objetos del modelo, dejando de lado las interacciones con las vistas y efectos de los casilleros.

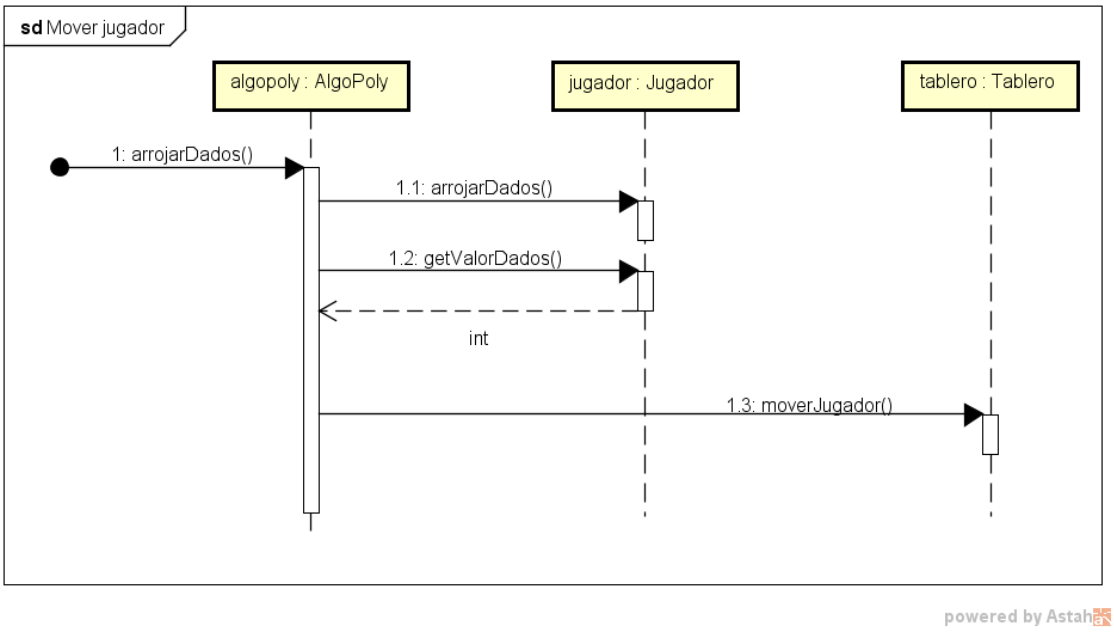

Figura 15: Secuencia del metodo arrojar dados de AlgoPoly.