FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 - Balatro

[7507/9502] Algoritmos y Programación III
Segundo Cuatrimestre de 2024

Alumno	Padron	Mail
Ladaga, Rodrigo	100188	rladaga@fi.uba.ar
Alvarez, Aníbal	102533	analvarez@fi.uba.ar
Apud Mora, Tomás Ezequiel	107968	tapud@fi.uba.ar
De Toffoli, Agustín	104514	adetoffoli@fi.uba.ar

Índice

1. Introducción **2**

2. Supuestos **2**

3. Diagramas de clase **3**

4. Diagramas de secuencia **6**

5. Diagrama de paquetes **9**

6. Detalles de implementación del modelo **10**
6.1. Clase Balatro 10
6.2. Clase Ronda, Descarte y Mano 10
6.3. Clase Tienda 10
6.4. Clase Carta 10
6.5. Clase Mazo 11
6.6. Clase Abstracta Joker 11
6.7. Clase Tarot 12
6.8. Clase PuntuacionTirada 12
6.9. Interfaz EstrategiaModificacion y Clase Abstracta Modificador 12
6.10. Clase Abstracta ManoDePoker 13
6.11. Clase EvaluadorMano 13

7. Otras implementaciones **14**
7.1. Lector de JSON 14
7.2. Interfaz GeneradorRandom 14

8. Excepciones **14**

1. Introducción

El presente informe reune la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar un juego similar a "Balatro."aplicando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

A continuacion enumeraremos los supuestos empleados en la implementación de las distintas funcionalidades de nuestro juego:

1. Las cartas de Tarot son de único uso y no se asignan al jugador como tal, modifican la carta o la mano deseada y se eliminan.
2. Una mano de poker se puede mejorar más de una vez, en caso de lograr volver a comprar la carta de Tarot en la tienda.
3. Se considera que una carta con Tarot aplicado en el multiplicador, además de modificar el multiplicador sumará sus puntos base.
4. Se considera que una carta con Tarot aplicado en los puntos, no sumara sus puntos base ya que sumara los brindados por el Tarot.
5. Una tirada tiene que estar compuesta indefectiblemente por 5 cartas.
6. Independiemente del tipo de mano tirado, se sumara a la puntuación tirada el valor de las 5 cartas presentes en esta.
7. La Tienda asignada a la ronda actual sera mostrada al superar con exito esta última, no antes.
8. No se limita al usuario a la utilización de una moneda para la compra en la tienda, todas las cartas presentes son de obtencion gratuita.
9. La cantidad mínima de cartas a descartar es 1 y la cantidad máxima es 5.

3. Diagramas de clase

En la siguiente sección, se presentan varios diagramas de clases mostrando las relaciones estáticas entre dichas clases en nuestro diseño. *Aclaración*: decidimos no agregar los getters y setters en algunas clases para facilitar la lectura de los diagramas.

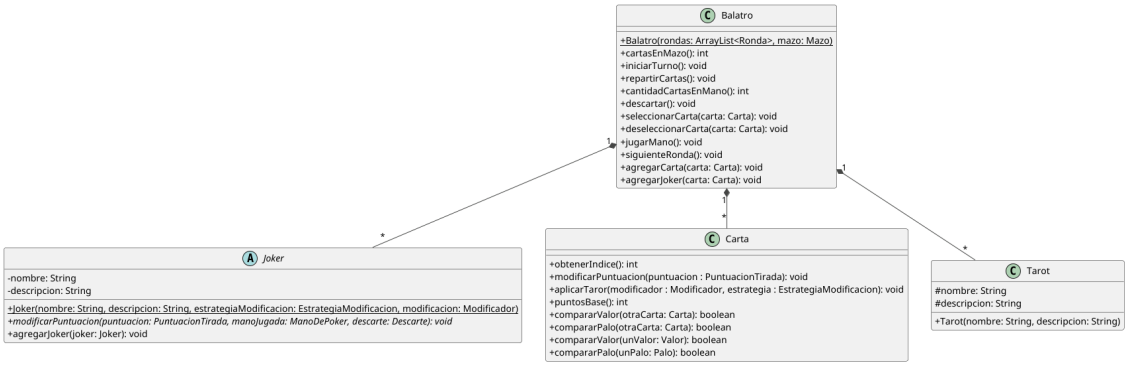

Figura 1: Diagrama de clases que muestra la relación entre Balatro con Carta, Joker y Tarot

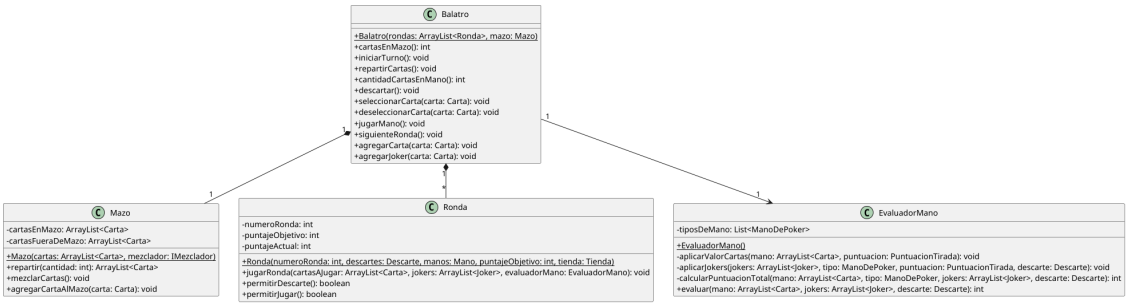

Figura 2: Diagrama de clases que muestra la relación entre Balatro con Mazo, Ronda y EvaluadorMano

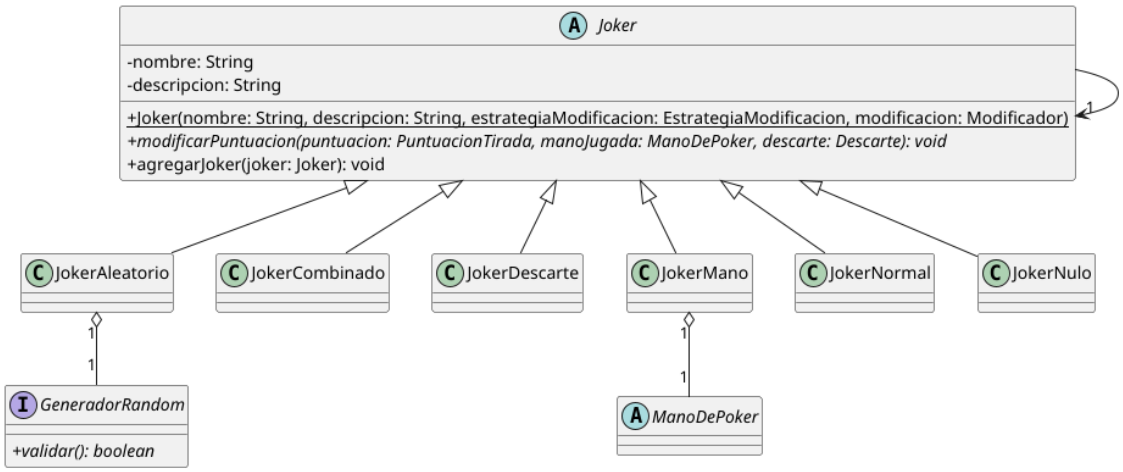

Figura 3: Diagrama de clases que muestra la relación entre la clase abstracta Joker y sus hijas

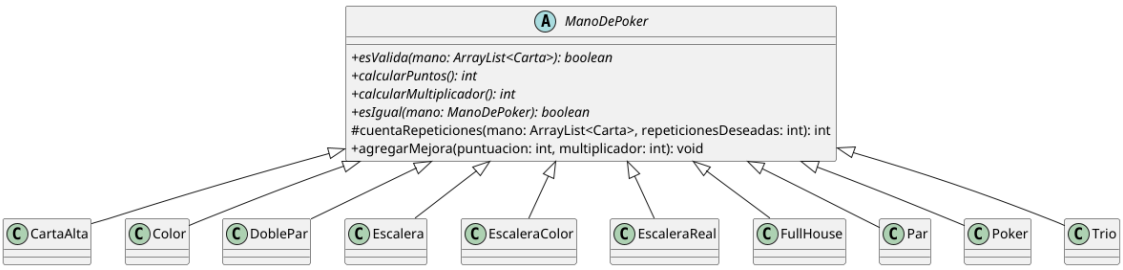

Figura 4: Diagrama de clases que muestra la relación entre la clase abstracta ManoDePoker y sus hijas

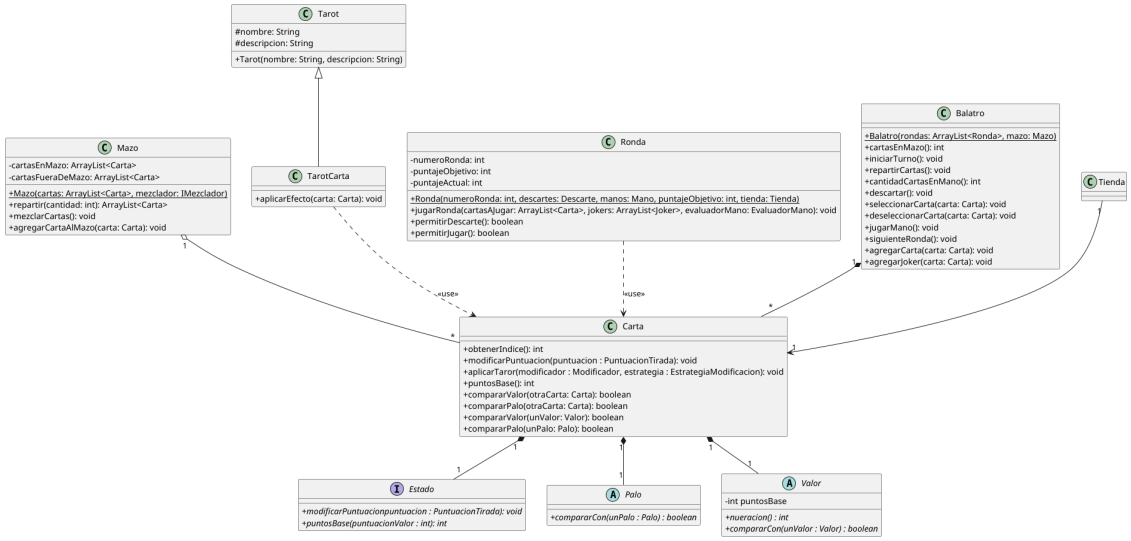

Figura 5: Diagrama de clases que muestra la relación entre Carta con distintas clases que la utilizan

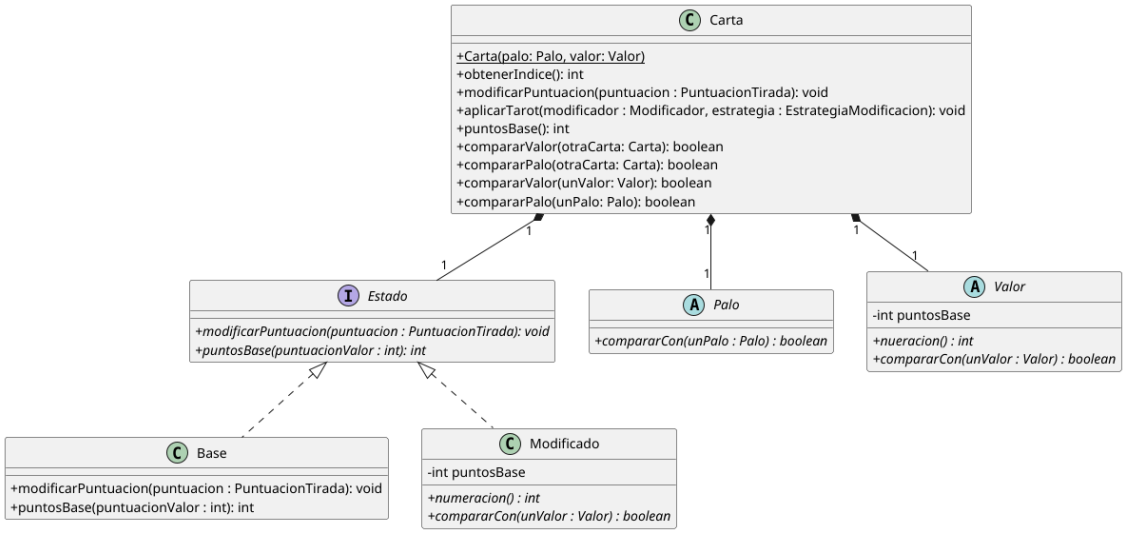

Figura 6: Diagrama de clases que muestra a la clase Carta con sus dependencias

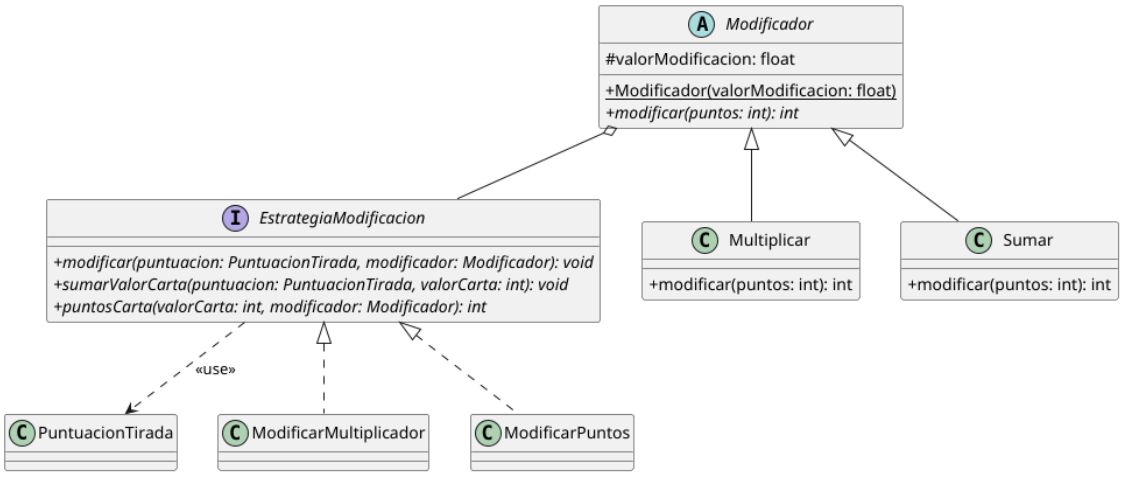

Figura 7: Diagrama de clases que muestra la aplicación del Patron Strategy

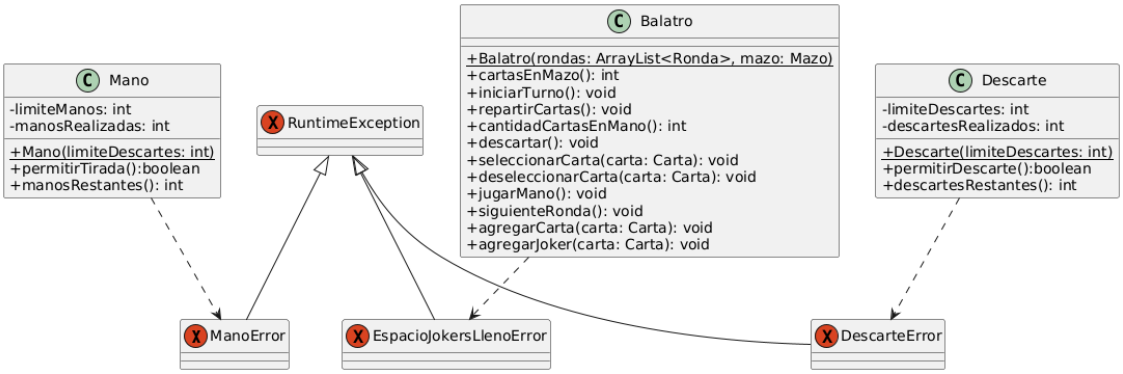

Figura 8: Diagrama de clases que muestra las clases con dependencia a excepciones

4. Diagramas de secuencia

En esta última sección de este informe, se muestran distintos diagramas de secuencia que nos parecieron importantes de presentar.

En la Figura 9 y 10, se observa como se lleva a cabo un ronda y evaluación de una mano Full House. Al principio, se parsea el archivo Balatro.json y devuelve una instancia de Balatro con las rondas y mazo inicializados. Luego, se seleccionan las cartas para arman una mano de Full House y se agregan a las manos a jugar, por consiguiente, se evalúa la mano, que finalmente da como resultado una puntuación de 360 puntos (no se aplica ningun joker por ser la primera ronda).

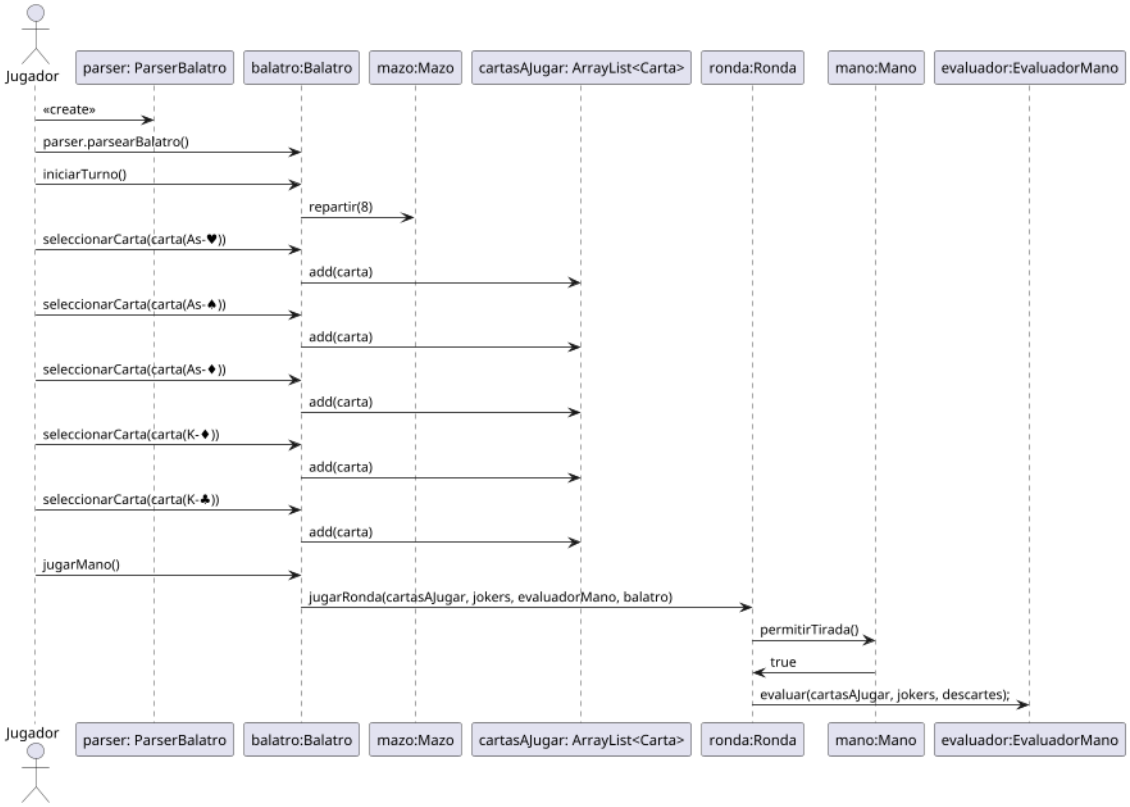

Figura 9: Diagrama de secuencia de evaluación de una mano FullHouse

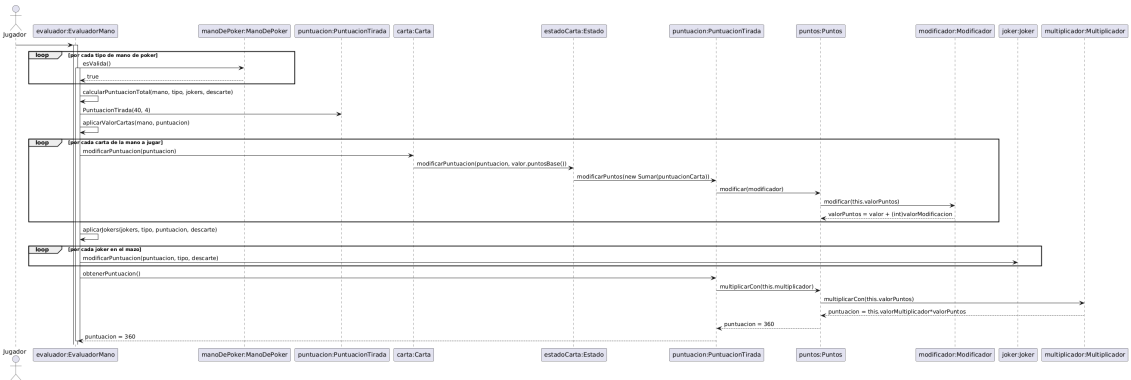

Figura 10: Diagrama de secuencia de evaluación de una mano FullHouse

En la figura 11, se inicializa como en la Figura 9 y se evalua una mano de Full House con un Joker que suma +8 al multiplicador de la mano jugada, de poseer multiplicador 4, se modifica a multiplicador 12.

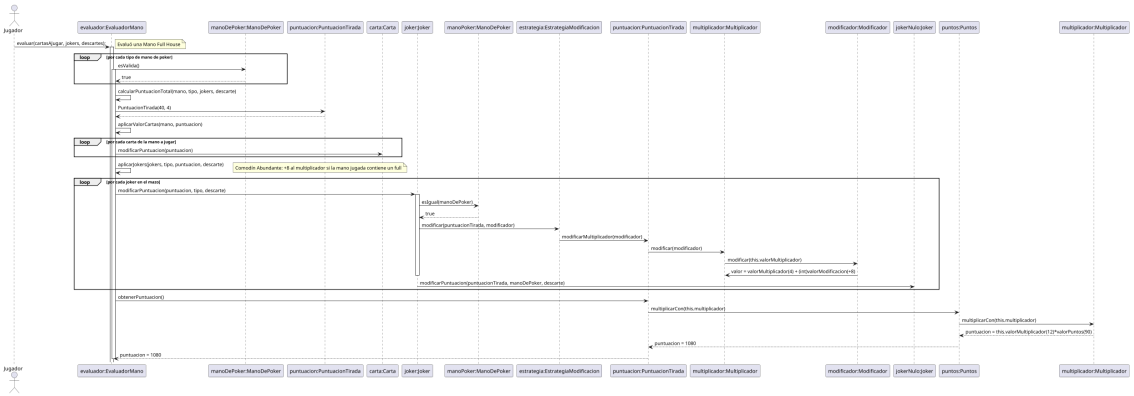

Figura 11: Diagrama de secuencia de evaluación de una mano FullHouse con un Joker que modifica la mano

5. Diagrama de paquetes

En esta sección, presentamos un diagrama de paquetes de nuestro modelo.

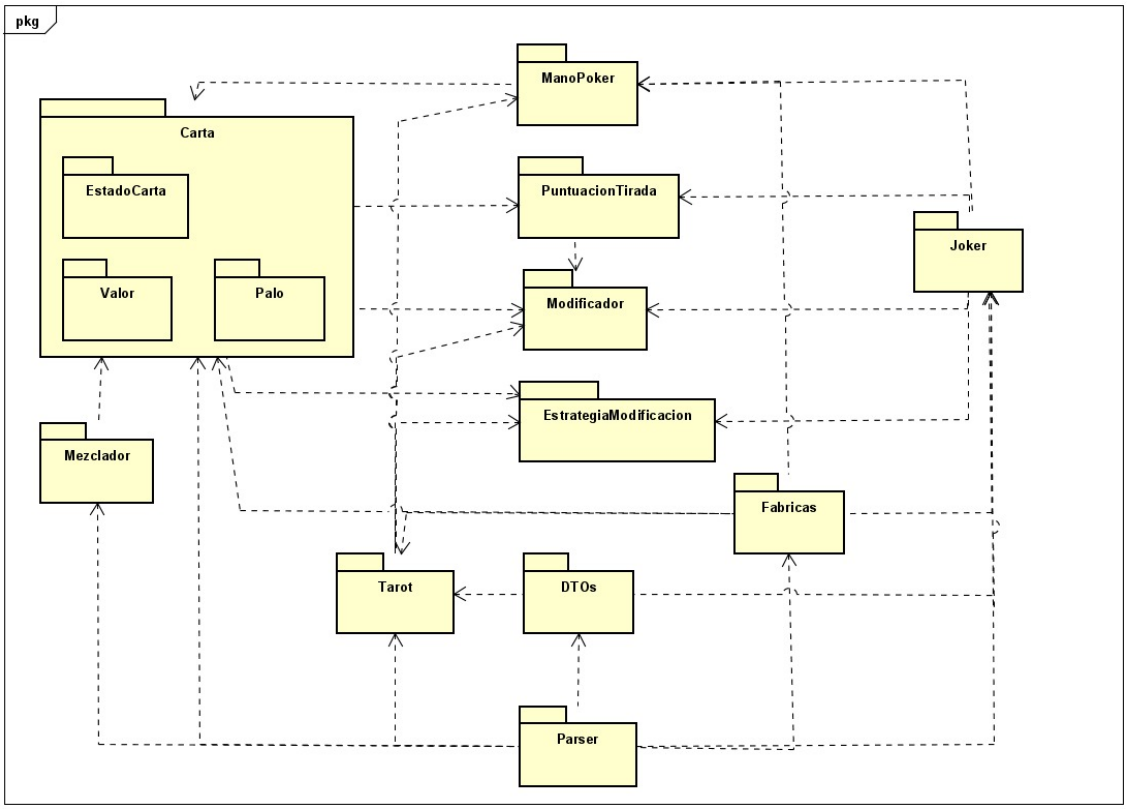

Figura 12: Diagrama de paquetes

6. Detalles de implementación del modelo

En el modelo se desarrolla un juego simíl Balatro. Se tiene como clase encargada de gestionar lo realizado por el usuario consumidos a la clase Balatro, la cual se encarga de almacenar e interactuar cada Ronda y resolverla, buscando superar el límite de puntos de esta y pasar a la siguiente. Cada Ronda tiene un objeto Descarte y Mano propio, los cuales tienen un limíte de descartes y tiradas previamente establecido, limitando al jugador en la cantidad de veces que puede realizar cada acción.

La Ronda termina al momento en que el jugador consigue superar la cantidad de puntos límite (caso victoria) o no lo consigue y se queda sin tiradas disponibles para realizar (caso derrota).

Al momento de superar con éxito la Ronda esta muestra la tienda al Jugador, con el cual se puede interactuar para la obtención de Comodines (Jokers), cartas de Tarot y cartas de poker comunes para agregar al mazo. El mazo esta contenido dentro de la misma clase Balatro.

6.1. Clase Balatro

Esta clase es la encargada de manejar el flujo del juego y efectuar la comunicacion entre las acciones realizadas entre el jugador (mendiante las vistas) y trasladarlas al resto del modelo.

Balatro se encargara de resolver cada Ronda, teniendo métodos para que el jugador pueda seleccionar las Cartas y luego jugarlas (o descartarlas). Además contendra los Jokers que el jugador va adquiriendo en la tienda a medida que avanzan las Rondas.

6.2. Clase Ronda, Descarte y Mano

Representa la Ronda del juego. Esta tiene como atributos un objeto Mano y otro objeto Descarte, además de un entero (puntajeObjetivo) el cual representa el puntaje a superar por parte del Jugador.

La Ronda esta en constante comunicación con Mano y Descarte, estos se encargan de gestionar el compartamiento de las acciones Jugar Mano y Descartar, ya que tienen el límite de veces que se puede realizar la acción, la cantidad de veces que el jugador la realizo y el poder volver a realizarla.

Además cada Ronda tiene guardada una Tienda, que se mostrara al ser superada con éxito.

6.3. Clase Tienda

La Tienda es una clase contenedora de Jokers, Tarots y una carta, la cual se mostrara al finalizar la Ronda al jugador. Esta tiene como objetivo el proveer al jugador de mejoras, pudiendo elegir este cualquier elemento de la Tienda y apropiarselo para su posterior uso durante la partida.

6.4. Clase Carta

La clase Carta representa las cartas de poker que el jugador puede emplear para realizar distintas manos y superar la puntuación de la ronda. Cada carta tiene como atributos otras clases llamada Palo y Valor.

Palo tiene como funcionalidad el representar el palo de la carta (Trebol, Pica, Diamante o Corazon), implementando un metodo para poder compararse con el Palo de otras cartas. Este se emplea con la utilización del **patrón de diseño Double Dispatch**.

El Valor por su parte tiene la funcionalidad de representar el número de carta de poker al que esta asignada, esta tiene como metodos el devolver su númeracion en el mazo de Poker, los puntos base que esta provee y el poder compararse con otro Valor.

```
public abstract class Valor{
public int puntosBase() {
    return puntosBase;
}

public abstract int numeracion();
}
```

Por otra parte la Carta puede ser Tarotizada, en la cual una carta Tarot le aplica un efecto y cambia el tipo de modificación que realiza la Carta a la PuntuacionTirada. Para esto se aplico a la carta el **patrón de diseño State**, en el cual se tienen dos estados, Base y Modificada.

El estado Base es con el que se instancia la Carta. Este modifica la PuntuacionTirada añadiendo a los puntos los puntosBase de la Carta. Al Tarot aplicar un efecto sobre la Carta, el mismo Tarot le envía el tipo de modificación que hace (si Suma o Multiplica) y la estrategia de modificación que realizara (si sobre los Puntos o el Multiplicador). La Carta cambia su Estado a Modificada y le envía lo anteriormente mencionado, delegando siempre en el Estado el modificar la PuntuacionTirada.

6.5. Clase Mazo

El Mazo es tal cual su nombre indica la clase contenedora de todas las Cartas que el jugador tendra a su disposición. Este tiene la capacidad de repartir cartas al jugador (siempre llegando a ocupar el límite de 8 cartas). Este tiene asociado una interfaz IMezclador la cual se encarga de mezclar las cartas en el mazo.

Se implemento aparte de las cartasEnMazo un ArrayList<Carta>llamado cartas fuera de mazo, lo cual nos facilita el momento de pasar de ronda y tener que volver a juntar todas las cartas nuevamente en cartasEnMazo.

6.6. Clase Abstracta Joker

Esta clase fue creada para la representación de los comodines que el jugador puede obtener para mejorar la puntuación de la tirada realizada. De esta clase heredan las clases JokerNormal, JokerAleatorio, JokerDescarte, JokerMano y JokerCombinado. Estos se diferencian en tener distintos tipos de activación, por ejemplo el JokeMano se activa (o no) dependiendo de la mano que el jugador tiró.

El metodo a implementar de la clase madre Joker es modificarPuntuación, el cual recibe una PuntuacionTirada y dependiendo el tipo de Joker que sea realizara una validación para verificar si tiene que realizar la modificación.

```
public void modificarPuntuacion(PuntuacionTirada puntuacion,
ManoDePoker mano, Descarte descartes){
// logica segun Joker
}
```

Para la implementación del JokerCombinado decidimos utilizar el **patron de diseño Decorator**, ya que la clase abstracta Joker tiene como atributo otro Joker, logrando así encadenar el comportamiento del Joker Combinado, el cual invoca el efecto de un Joker y luego este invoca el efecto de otro.

```
//atributo otroJoker, el cual nos permite encadenar Jokers entre
   si al modificar PuntuacionTirada
protected Joker otroJoker;
public void modificarPuntuacion(PuntuacionTirada puntuacionTirada,
   ManoDePoker manoDePoker, Descarte descarte) {
    // logica segun Joker
```

```
    this.otroJoker.modificarPuntuacion(puntuacionTirada,
        manoDePoker, descarte);
}
```

Los Jokers individuales en su creación instanciaran el atributo otroJoker con un Joker Nulo, el cual no realiza ninguna acción para la modificación de la PuntuacionTirada recibida.

6.7. Clase Tarot

El Tarot es una clase que modifica el comportamiento de una Carta (modificando su Estado) y además puede proveer una mejora de puntos y multiplicador a la ManoDePoker. De Tarot derivan dos clases, TarotCarta y TarotManoPoker, las cuales respectivamente aplican un efecto en la Carta o ManoDePoker deseada.

El TarotManoPoker tiene la particularidad de que se puede aplicar a una Mano en particular, a diferencia de TarotCarta que puede mejorar cualquier Carta a eleccion del jugador.

```
//TarotCarta aplicando efecto sobre Carta
public void aplicarEfecto(Carta carta){
    carta.aplicarTarot(modificador,estrategiaModificacion);
}
//TarotMano aplicando efecto sobre Mano
public void aplicarEfecto(ManoDePoker manoDePoker){
    if(mano.esIgual(manoDePoker)){
        manoDePoker.agregarMejora(puntuacion,multiplicador;
    }
}
```

6.8. Clase PuntuacionTirada

Esta es la puntuación que el jugador puede obtener al tirar una mano de poker. Esta tiene como atributos las clases Puntos y Multiplicador. Estas se van a modificar en tiempo de ejecución a medida que se van aplicando el valor de la mano de Poker, el valor de las cartas y los modificadores de los Joker.

Ambas, Puntos y Multiplicador, tienen un atributo el cual es el valor (tipo entero) que estas contienen. Ambas son modificables, pudiendo así variar su valor según sea necesario en el desarrollo de la tirada.

Un metodo muy importante de PuntuacionTirada es obtenerPuntuacion(), el cual provoca que Puntos se multiplique con Multiplicador y obtener el valor final de la tirada realizada.

6.9. Interfaz EstrategiaModificacion y Clase Abstracta Modificador

La Carta en su estado base tiene como objetivo modificar los Puntos, pero esta puede recibir un Tarot y de tal manera tener que modificar el Multiplicador de PuntuacionTirada. Esta misma problematica ocurria con los Jokers, modificado los Puntos o el Multiplicador. Para resolver esto se uso el **patrón de diseño Strategy**, creando una interfaz EstrategiaModificacion, implementada por ModificarMultiplicador y ModificarPuntos, recibiendo ambos la PuntuacionTirada y sabiendo que tienen que modificar de ella.

```
public interface EstrategiaModificacion {
    void modificar(PuntuacionTirada puntuacion, Modificador
        modificador);

    void sumarValorCarta(PuntuacionTirada puntuacion, int
        valorCarta);
```

```
    int puntosCarta(int valorCarta, Modificador modificador);
}
```

Como se observa a su vez esta tiene los métodos sumarValorCarta() y puntosCarta(), los cuales se emplean en caso que la Carta se encuentre Modificada.

Lo que no sabe hacer por si sola es como modificar ese valor ya que los Puntos o el Multiplicador pueden modificarse mediante una suma o una multiplicación. Por lo que creamos una nueva clase que es almacenada por la EstrategiaModificacion, volviendo a aplicar **Strategy** con una clase abstracta que llamamos Modificador. Las clases Sumar y Multiplicar heredan de Modificador, teniendo un atributo modificacion (de tipo float) y un metodo modificar, el cual realiza la modificación requerida sobre el valor recibido.

6.10. Clase Abstracta ManoDePoker

La ManoDePoker es una c lase que representa la traducción de la tirada de 5 cartas que realiza el Jugador. Este tiene como metodo principal esValida(ArrayList<Carta>mano); el cual verifica si la mano jugada corresponde a esa ManoDePoker que esta realizando la evaluación. Esto se realiza con distintos métodos, por ejemplo la clase Escalera tiene los metodos esEscalera() y esEscaleraSuperiorConAs(), en el cual verifica las numeraciones de las cartas para afirmar que sea Escalera.

También hay un metodo llamado CuentaRepeticiones, el cual como su nombre indica verifica las incidencias de un mismo Valor en una mano, así pudiendo determinar si es una mano Par, Trio, Poker, Doble Par o Full House.

Por último tenemos la verificación por Color (es decir, Palo), en la cual se verifica que las 5 cartas sean del mismo Palo, pudiendo así tener una tirada Color o, en caso de que además sea Escalera, una EscaleraColor.

Cada ManoDePoker tiene valores base para el Multiplicadr y los Puntos, por lo que siempre que se instancia una PuntuacionTirada como valor base de Puntos y Multiplicador tiene los que contenga la ManoDePoker que se jugo.

Estos valores base se pueden mejorar gracias a las cartas de Tarot, las cuales le agregan más puntos y más mutliplicador base a alguna mano en paticular. Para esta implementación creamos la clase MejoraMano, que la ManoDePoker guarda como atributo. Al tarotizar una mano lo que se hace es modificar el valor de la mejora (tanto en puntos como en multiplicador) sumando los que agrega la carta Tarot. Esto genera que una ManoDePoker pueda recibar varias mejoras, siendo estas acumulativas.

6.11. Clase EvaluadorMano

Esta clase se encarga de analizar la tirada realizada por le jugador, determinando así el tipo de ManoDePoker que este jugó. Lo hace guardando en un ArrayList<ManoDePoker>todas las manos disponibles y comparando si puede ser ella con el metodo esValida(). Es importante el orden en que se guardan en nuestro modelo ya que podriamos cometer el error de tirar una EscaleraColor pero que aplique la puntuación de una tirada Color (la cual es menor).

Además de ello, al determinar la ManoDePoker correspondiente a la tirada instancia un objeto PuntuacionTirada, al cual le aplica las modificaciones pertinentes, las cuales son agregar a los Puntos o al Multiplicador las modificaciones presentes en las Cartas que se usaron para la tirada y los Jokers que tiene en su podes el jugador.

Todo esto lo realiza mediante el metodo público

```
public int evaluar(ArrayList<Carta> mano, ArrayList<Joker>
    jokers, Descarte descarte)
```

el cual luego retorna un entero correspondiente a la puntuación obtenida finalmente en la PuntuaciónTirada antes mencionada.

7. Otras implementaciones

7.1. Lector de JSON

Para la lectura de los distintos JSON se empleo la libreria Gson, con la cual logramos traducirlo en las clases del modelo. Se nos proveyo de cuatro archivos JSON (Comodines, Mazo, Tarots y Balatro). Con los tres primeros se modelaron tres parsers individuales (ParserJoker, ParserMazo y ParserTarot) y fueron de utilidad para verificar el correcto funcionamiento de ellos, que luego se usarían en ParserBalatro, el cual lee el archivo Balatro.JSON para cargar el juego.

Para la instanciación de cada objeto generamos fábricas particulares. Además usando el patron Data Transfer Object creamos las clases JokerDTO y TarotDTO, lo que nos permitio darle a ambas fabricas todas las indicaciones que nos provee el JSON para su instanciación, y implementar la lógica en las mismas fábricas.

7.2. Interfaz GeneradorRandom

El JokerAleatorio instancia la clase NumeroAleatorio, la cual implementa esta interfaz, y tiene la función de proveernos un booleano verificador de manera random según la probabilidad que tiene el Joker de activarse.

```
public class NumeroAleatorio implements GeneradorRandom {
    private int probabilidad;
    private Random rand;

    public NumeroAleatorio(int probabilidad) {
        this.probabilidad = probabilidad;
        this.rand = new Random();
}

    @Override
    public boolean validar(){
      return rand.nextInt(probabilidad) + 1 == 1;
    }
}
```

La creación de esta interfaz fue de utilidad ya que nos permitio poder testear el comportamiento del JokerAleatorio con la utilización de la herramienta Mockito.

8. Excepciones

Explicación de cada una de las excepciones creadas y con qué fin fueron creadas:

DescarteError : Se lanza esta excepción cuando se quiere realizar más descartes de los asignados.

ManoError : Se lanza esta excepción cuando se quiere realizar más manos de las asignadas.

EspacioJokerLlenoError : Se lanza esta excepción cuando se quiere agregar más de cinco comodines al mazo.