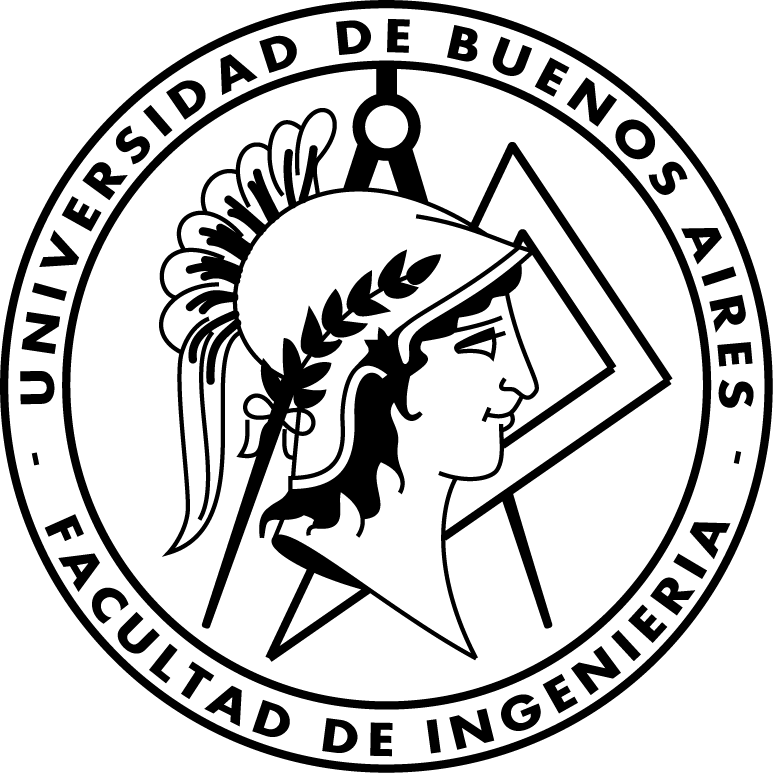
******

UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

**75.07 Algoritmos y Programación III**

**Trabajo Práctico Nro. 2**

**Al-Go-Oh!**

**Primer Cuatrimestre de 2018**

**Alumno Padrón E-Mail**

|  |  |  |
| --- | --- | --- |
| Bravo Arroyo, Victor Manuel | 98882 | alqace@gmail.com |
| Botter Brun, Juan Bautista | 101505 | juan.botter@gmail.com |
| Calvani, Sergio Alejandro | 98588 | sergiocalvani@outlook.com |
| Rack, Lucas Alexis | 99425 | lucas\_rack@live.com.ar |

***Índice***

**1. Introducción 3**

**2. Supuestos 3**

**3. Modelo de Dominio 3**

**4. Diagrama de Clases 6**

4.1. Yugioh,Tablero y Jugador 6

4.2. Cartas 7

4.3. Carta Monstruo 7

4.4. Carta Mágica, Trampa y Campo 8

4.5. Diagrama Completo 9

**5. Diagrama de Secuencia 10**

5.1. Activar Carta de Campo 10

5.2. Activar Carta Fisura 11

**6. Diagrama de Estados 12**

**7. Diagrama de Paquetes 12**

**8. Detalles de Implementación 13**

**9. Excepciones 13**

1. ***Introducción***

En el siguiente informe se explicará de manera detallada todo lo relacionado a la realización del Trabajo Práctico Nro. 2. Primero se enunciarán los supuestos que tomamos para la realización del trabajo, luego el Modelo de Dominio, el cual tiene la descripción de cada una de las clases que utilizamos y el porqué de su utilización. Le sigue el Diagrama de Clases, cuyo objetivo es el de explicar mediante un diagrama UML las relaciones que hay entre las clases explicadas anteriormente. Luego sigue el Diagrama de Secuencia de los métodos que creemos que pueden resultar confusos, seguido de esto se encuentra el Diagrama de Estado, el cual muestra cómo se comporta el flujo de la aplicación. A continuación, estará el Diagrama de Paquetes para que se pueda observar la dinámica de estos y, por último, se continuará con las Excepciones utilizadas durante el programa, así mismo a que se refiere cada una, el porqué de su creación y el cuándo y dónde se lanzará.

1. ***Supuestos***

A continuación, se van a enumerar los supuestos que hemos tenido en cuenta para la realización del Trabajo:

* Por defecto cuando se coloca una carta monstruo, se encuentra en posición de ataque
* Las Cartas Mágicas y de Trampa se agregan por defecto boca abajo
* En el mazo puede haber cartas repetidas
* Los errores que pueden llegar a surgir durante la ejecución del programa serán avisados me manera sonora mediante un sonido de error

1. ***Modelo De Dominio***

En el siguiente apartado se explicarán con detalle cada una de las clases que se pretenden utilizar a lo largo de la realización del trabajo:

• ***Yugioh:*** Clase principal del Juego, es la que se encarga de crear el tablero y los jugadores. A su vez, verifica quien fue el ganador y/o perdedor de la partida.

• ***Tablero:*** Clase que representa el tablero de Juego, contiene los lados de cada Jugador. Es el que permite la interacción entre ambos lados del tablero.

• ***Lado:*** Es la clase que representa el lado de cada Jugador, así mismo dentro de ella se encuentran las zonas que conforman cada lado. Tiene como atributos las zonas de cada Tipo de carta y también se encarga de agregar una carta en donde sea correspondiente. Así mismo, permite atacar con un monstruo a otro monstruo del lado enemigo.

***• Jugador:*** Clase que representa a un jugador de una partida, en este caso solamente va a haber dos. Tiene puntos de vida, los cuales, si uno de ellos llega a cero, ese jugador perdió el juego.

• ***FabricaDeCartas:*** Clase que se encarga de crear las cartas que van a estar en el mazo.

***• Mano:*** Es la clase que contiene las cartas las cuales el jugador obtuvo del maso. Son las que el jugador puede poner en el tablero (obviamente, según la zona se corresponda). Posee cartas (5 en este caso) y permite agregar cartas a sí misma sacándola desde el mazo. También se encarga de verificar que contenga las 5 partes del Éxoda, se poseerlas, el jugador con dicha mano resultará ganador.

***• Mazo:*** Representa a una colección de cartas de la cual el jugador puede sacar de a una para tenerlas en su mano. Tiene como máximo 40 cartas y en el caso de que un jugador se quede sin cartas, ese jugador resulto como perdedor del juego.

***• Carta:*** Clase que representa una carta del juego. Posee un nombre, un efecto y una posición (estas últimas dos se explicarán con más detalle a continuación). Los métodos que se encuentran presentes por el momento son los de obtener el nombre de la carta, el de asignarle una posición, el de saber si la carta esta boca abajo y también voltear. A su vez de ella hereden 4 clases más las cuales luego se detallarán, pero utilizamos este criterio debido a que el comportamiento de cada tipo de carta es similar, salvo algo muy específico, y también cumple la regla de “*es un*”.

***• CartaDeCampo:*** Representa a una carta de tipo Campo. Tiene como atributo el Tablero, ya que este tipo de carta afecta a ambos lados por igual, y su dueño. Sus métodos son los que le permiten aumentar los puntos de ataque y defensa según sea el caso necesario. De ella hereden Wasteland y Sogen, las cuales su comportamiento es igual, lo único que varía es a qué lado se le aumenta el ataque y a cuál la defensa.

***• CartaDeTrampa:*** Representa a una carta de tipo Trampa. Tiene un atributo que es el D

ueño de esa carta y sus métodos son los que permiten activarse para utilizar su efecto. De ella heredan Cilindro Mágico y Reinforcements, ya que como sucede con las Cartas De Campo, su comportamiento es similar, lo único que varía es como se aplica su efecto.

***• CartaMágica:*** Representa a una carta Mágica. Tiene como atributo el lado en el cual se encuentra y su método es el que le permite activar su efecto. De ella heredan Fisura, Olla de la Codicia y Agujero Negro, debido a que como sucede en los dos casos anteriores, lo único que varía entre ellas es su manera de utilizar el efecto en cuestión.

***• CartaMonstruo:*** Representa una carta tipo Monstruo y, sin dudas, es la que más diferencia tiene con respecto a los demás tipos de cartas. Presenta mayor cantidad de atributos, ya que posee puntosDeAtaque, puntosDeDefensa, un estado y un nivel. El estado se explicará en breve, mientras que el nivel representa, como lo dice su nombre, al nivel de cada carta de tipo monstruo, ya que se necesita por el hecho de que, si se quiere insertar una carta en el tablero con tal nivel, puede requerir distintos tipos de sacrificios. Sus métodos son los que se encargar de agregar una carta en base a si su Nivel, luego de aumentar sus puntos de ataque o de defensa según sea necesario y la de permitir atacar a otro monstruo. De ella heredan algunas cartas que presentan algún efecto en particular, tal como es el caso del Jinzo #7 y el Insecto Come-Hombres. Luego están las partes de la Éxoda o del Dragón de Ojos Azules cuyo comportamiento es un poco distinto.

***• Nivel:*** Representa a el nivel de una Carta Monstruo, sus métodos son los que permiten crear el nivel según el número de estrellas correspondiente, el de sacrificar según el nivel necesario y el de destruir. De ella heredan NivelBajo, NivelMedio y NivelSuperior, esto se debe a que según el tipo de estrellas que tenga la carta, su colocación puede requerir más cantidad de sacrificios o no.

***• Estado:*** Se encarga de representar el estado que puede tener la carta de tipo Monstruo. Con estado nos estamos refiriendo a si se encuentra en modo Ofensivo o modo Defensivo. Su método es el que, según el estado en el que se encuentre, se realiza el ataque.

1. ***Diagrama de Clases***

Ahora se mostrará el modelo de clases que se tomó en cuenta para la realización del Trabajo. Se decidió dividir el diagrama en dos partes para mejorar su comprensión, luego se mostrará el diagrama completo para que se pueda observar con claridad las relaciones entre todas las clases.

***4.1 Yugioh, Tablero y Jugador***

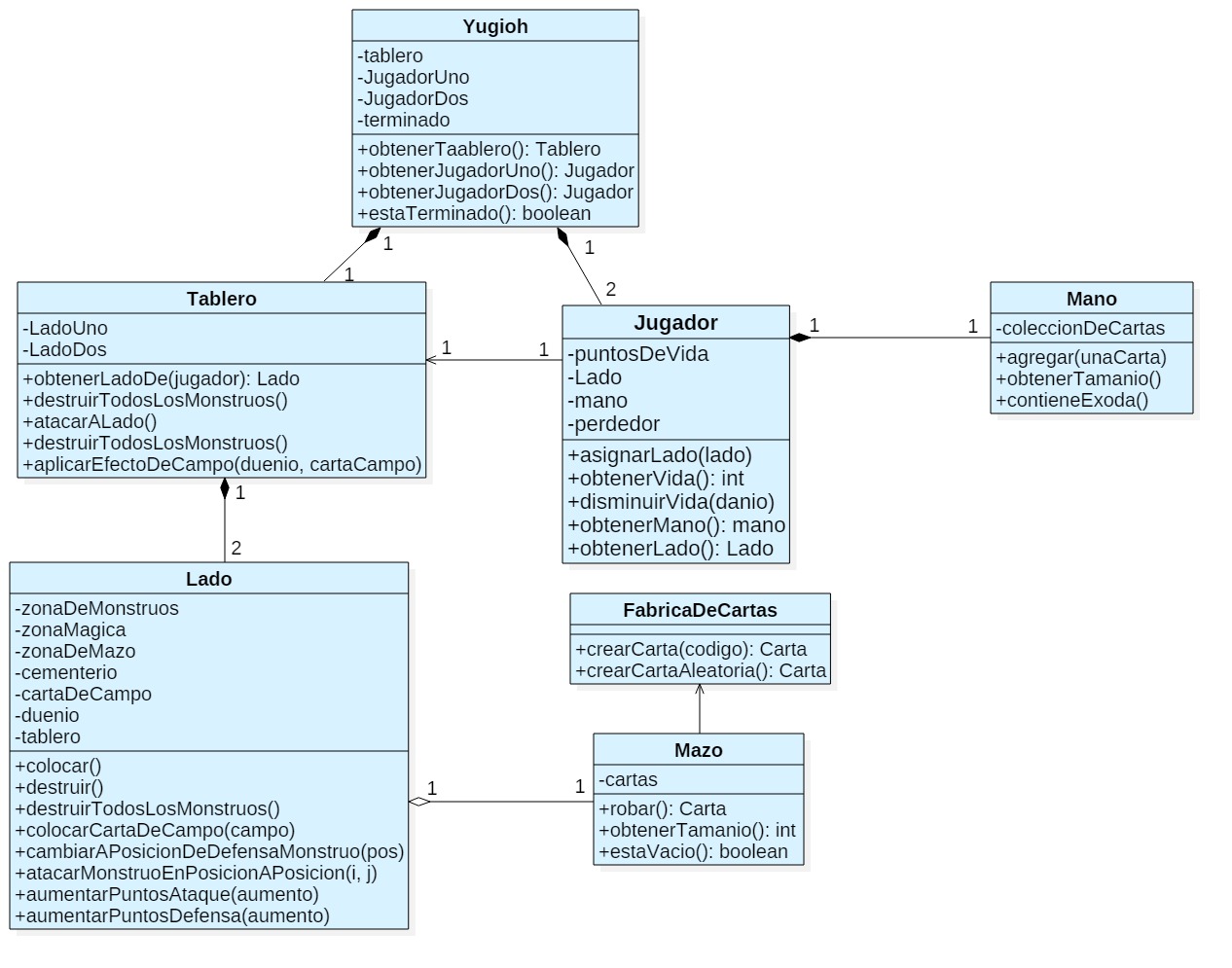


Figura 4.1: Diagrama de Jugador y Tablero

***4.2. Carta***

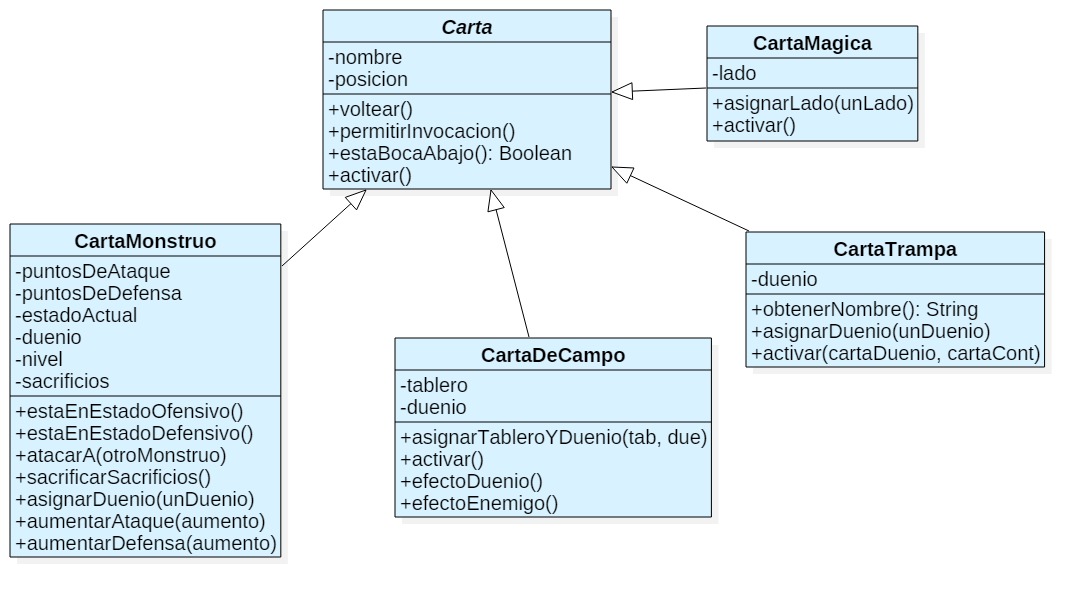


Figura 4.2: Diagrama de Carta

***4.3. Carta Monstruo***

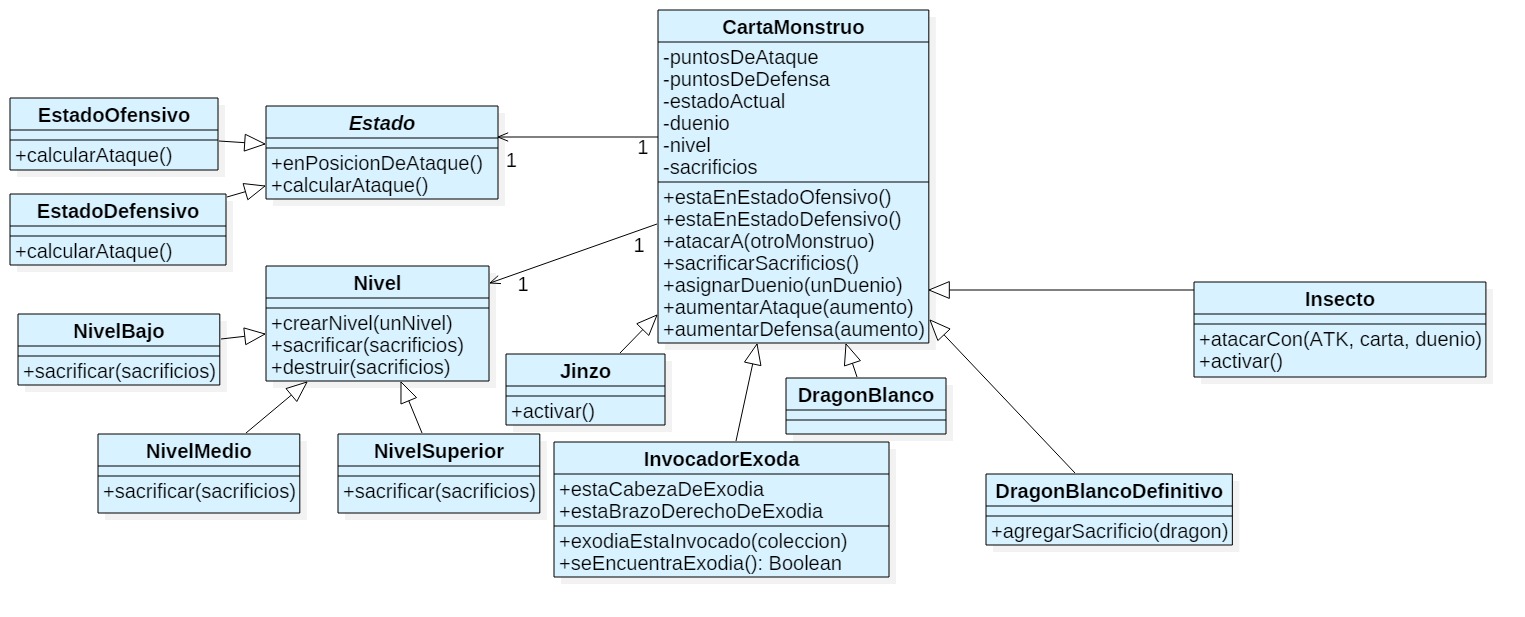


Figura 4.3: Diagrama de Carta Monstruo

***4.4. Carta Trampa, Mágica y de Campo***

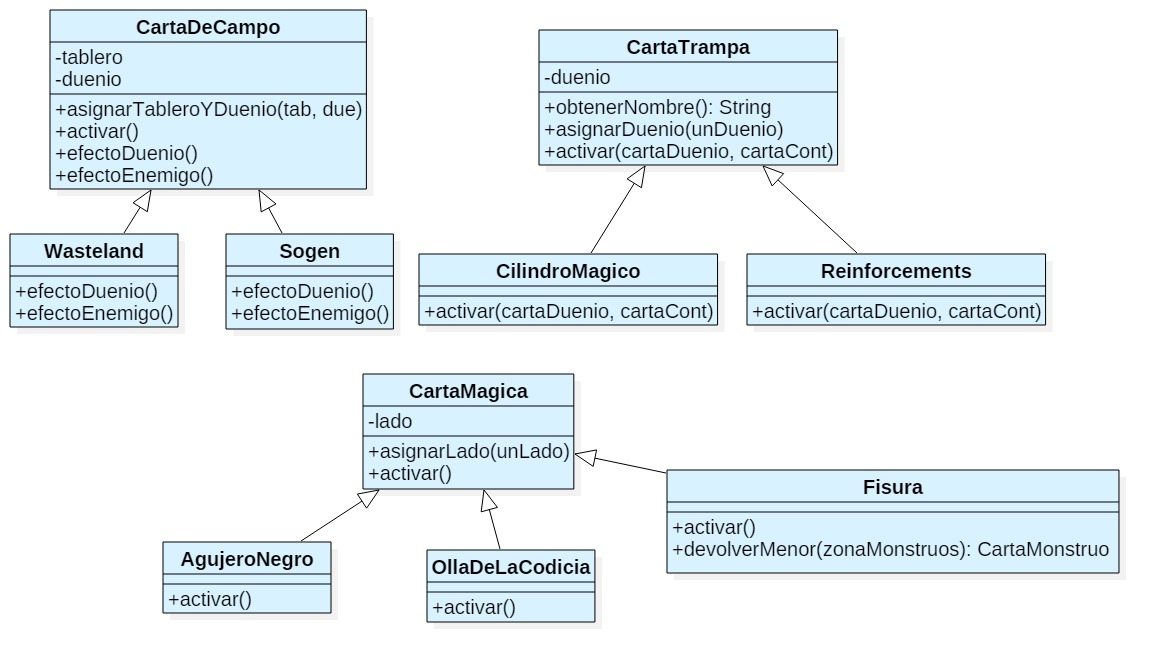


Figura 4.4: Diagrama de Carta de Campo, Trampa y Mágica

***4.5. Diagrama Completo***

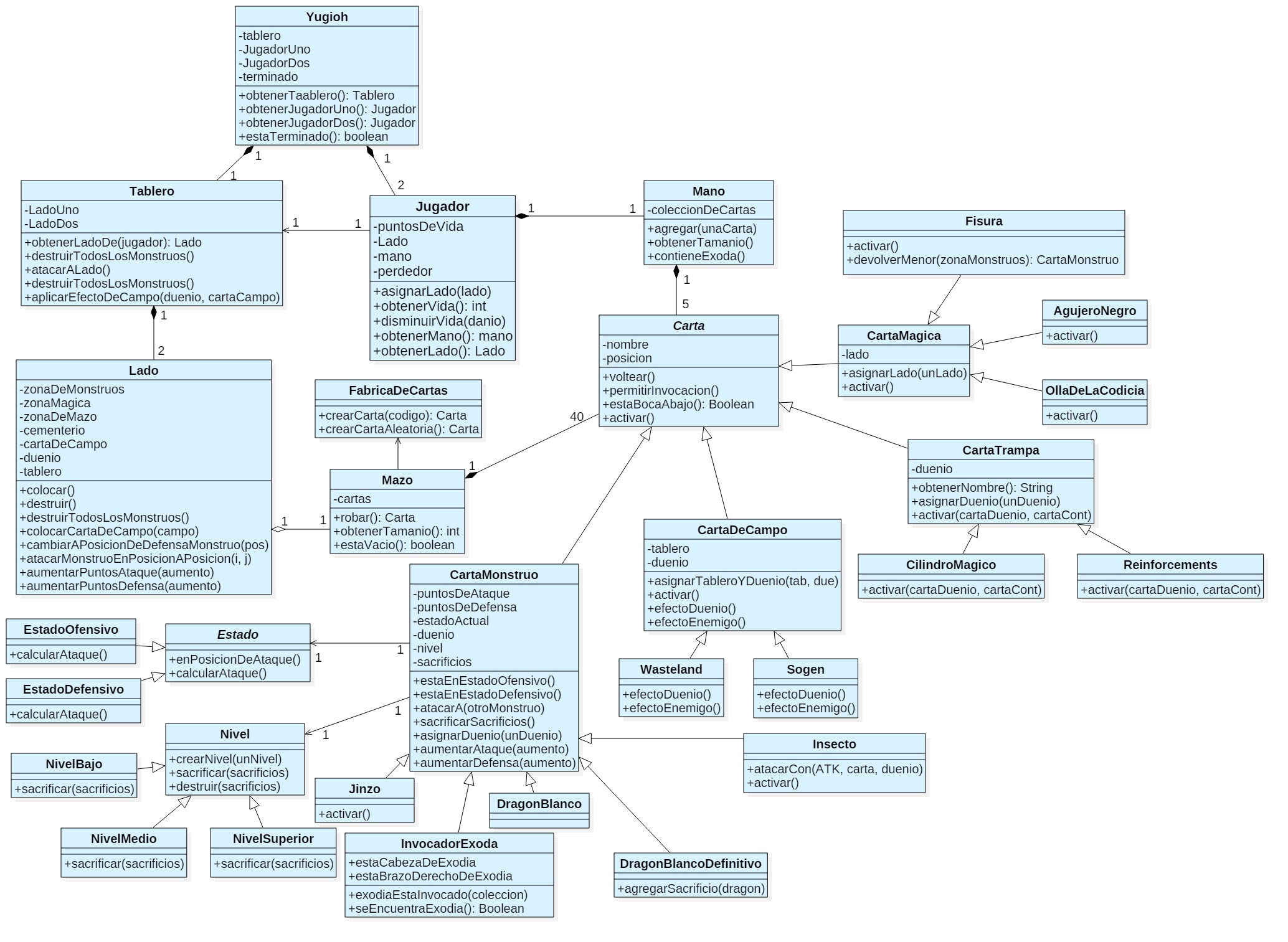


Figura 4.5: Diagrama de Clase Completo

1. ***Diagrama de Secuencia***

A continuación, se mostrarán algunos diagramas de secuencia de algunos métodos para entender de mejor manera su interacción con los otros objetos.

***5.1. Activar Carta De Campo***

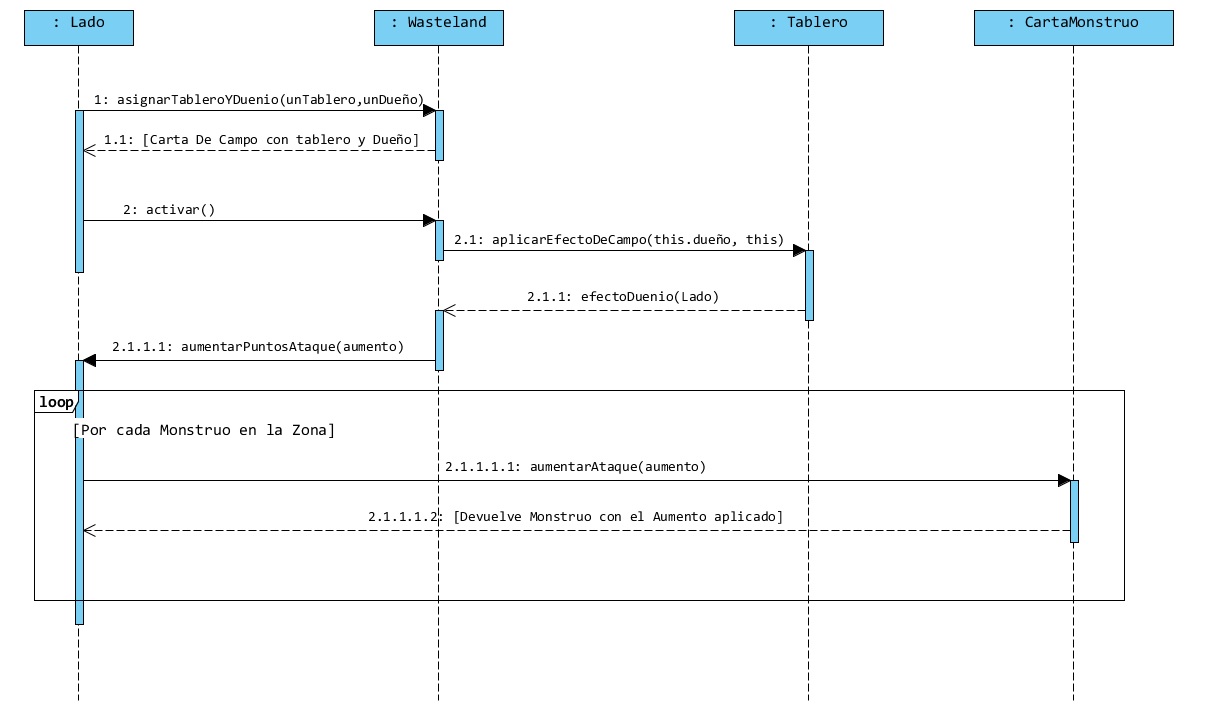


Figura 5.1: Diagrama de Secuencia de Activar Carta de Campo

***5.2. Activar Carta Fisura***

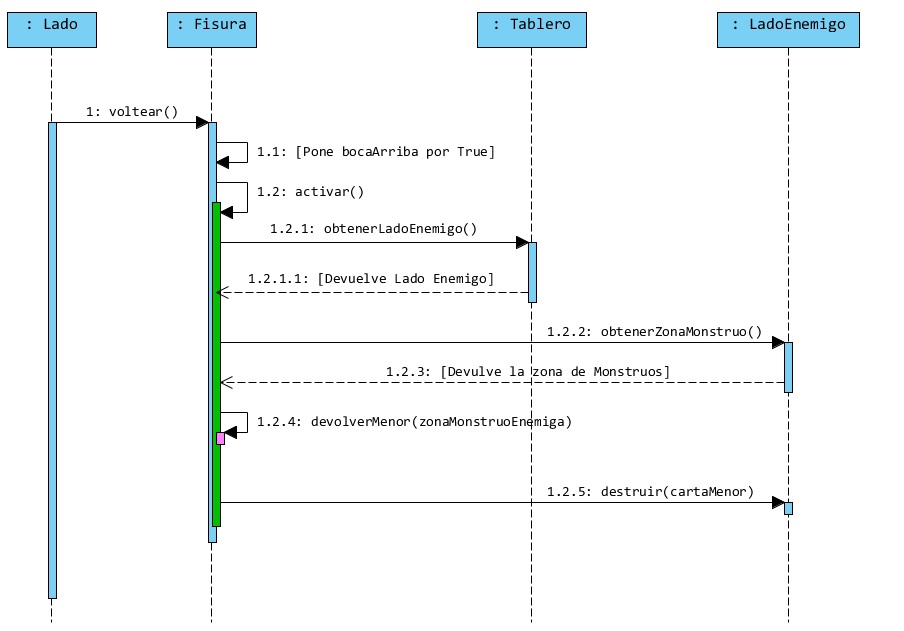


Figura 5.2: Diagrama de Activar Carta Mágica “Fisura”

***5.2. Ataque de un monstruo a otro en estado defensa.***

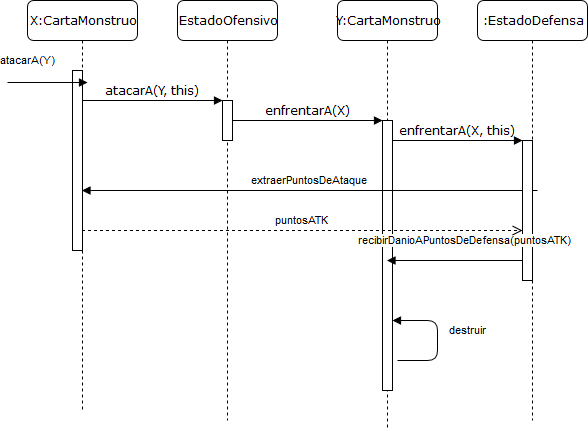


Figura 5.3: Diagrama de Activar Carta Mágica “Fisura”

Como se puede apreciar en la figura 5.2, el ataque entre dos monstruos depende de sus estados, en este caso, el atacante está en estado de ataque y el atacado en defensa. Se inicia llamando a atacarA(Y) de X el cual delega en el estado, el cual al ser ofensivo llamará a la carta monstruo atacada, en cambio si hubiese sido estado de defensa se lanzaría una excepción, puesto que no puede atacar en ese estado. Luego, Y recibe el mensaje enfrentara(X) y también lo delega en su estado el cual extrae los puntos de ataque del atacante y envía el mensaje a Y con los puntos. SI hubiese estado Y en estado de ataque, además se le enviaría el mensaje a X de recibirDanioAPuntosDeAtaque y X actuara de manera acorde. Finalmente, suponiendo que el ataque fue mayor a puntos de defensa, la carta Y se destruye.

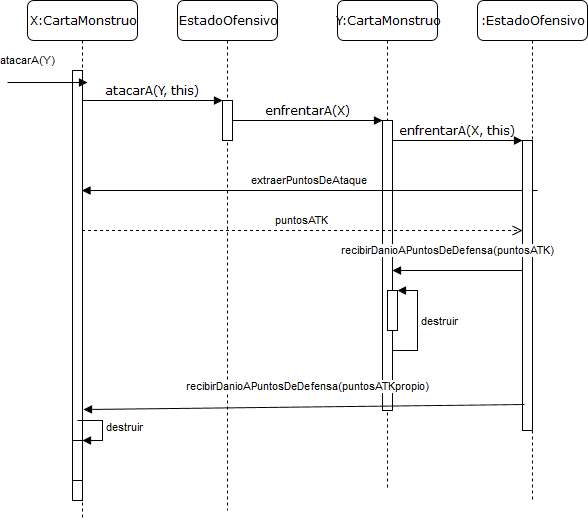


Figura: 5.4 Ataque en el cual ambos se destruyen

1. ***Diagrama de Estados***

A continuación, se mostrará un diagrama de los distintos estados que suceden desde la ejecución del programa hasta la finalización del mismo

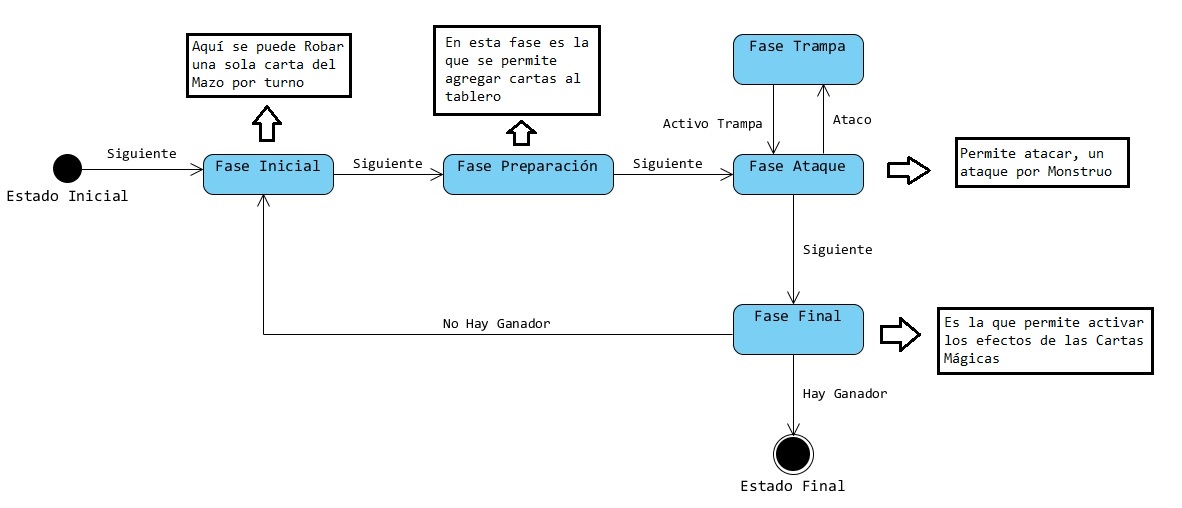


Figura 6.1: Diagrama de Estado

1. ***Diagrama de Paquetes***

Ahora se mostrará el diagrama de paquetes en función de lo que hemos implementado

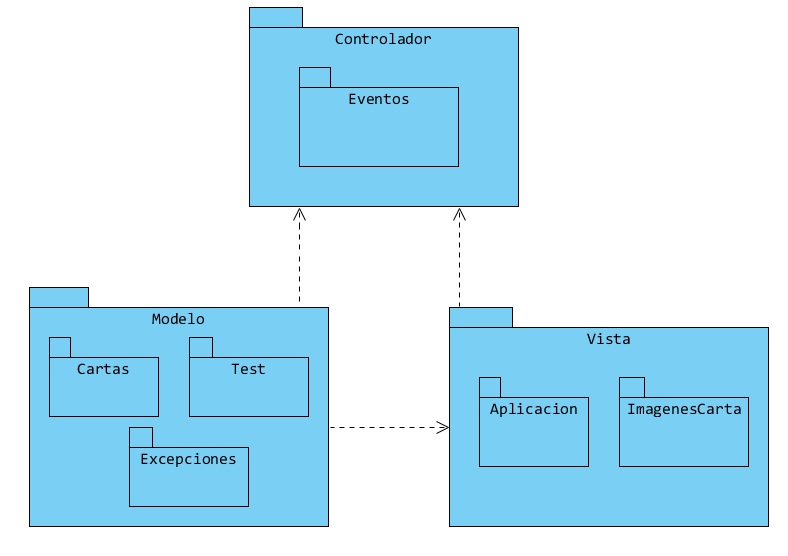


Figura 7.1: Diagrama de Paquete

1. ***Detalles de Implementación***

En el siguiente apartado se explicarán algunos métodos que creemos a nuestro criterio que pueden llegar a resultar confusos.

* ***Colocar Carta de Campo:*** Este método lo que realiza es lo siguiente, una vez que se coloca la carta de campo en el tablero, a dicha carta se le asigna su dueño y el tablero del juego. Luego se activa, lo cual le envía un mensaje al tablero para saber qué lado la colocó, una vez que tiene esa información, procede a aplicar el efecto correspondiente para cada Jugador.
* ***Activar Carta de Trampa:*** Cuando un Jugador ataca a otro Jugador, si el Jugador atacado tiene una carta de trampa, esta se activa y aplica el efecto según sea correspondiente.
* ***Activar Efecto Fisura:*** Esta carta mágica es la que, en caso de activarse, elije del lado contrario la de menor ataque y la destruye, para ello, primero como tiene como atributo el lado al que pertenece, obtiene el tablero y, por consiguiente, obtiene el lado del contrario. Luego obtiene la zona de Monstruos del contrario y a partir de allí, elije la carta de menor ataque y le envía un mensaje a lado enemigo que destruya esa carta en cuestión.
* ***Niveles:*** La cantidad de cartas sacrificadas requeridas para invocar a un monstruo en el campo depende de su nivel, pudiendo ser Bajo en cuyo caso no requiere sacrificios, Medio solo 1 y Superior con 2. Para abarcar el problema nos basamos en el patrón *Strategy*, en el cual implementamos el método invocar en cada una de las hijas de la clase abstracta *Nivel* y se encarga de lanzar las respectivas excepciones que ocurran al no contar con la cantidad de sacrificios suficientes.
* ***Creación de cartas:*** La creación de cartas es requerida al iniciar el juego y muchas veces al probar distintas funcionalidades en las pruebas unitarias, por lo tanto se hace necesario contar con una manera fácil de crear cartas sin tener que memorizar constantemente los constructores de cada clase y para ello se crea la clase *Fabrica de cartas* basada en el patrón *Abstract Factory* que se encarga de la creación de cartas mediante un código asociado a cada una y donde el usuario conoce que recibió una instancia de la clase abstracta *Carta*, con lo cual nos acercamos la regla de programar contra interfaces en vez de clases concretas.
* ***Ataque y Defensa:*** Una carta monstruo puede estar en estado de ataque o defensa al ser ubicada en la zona de monstruos y dependiendo tendrá distintas respuestas ante ataques oponentes pudiendo simplemente destruirse, atacar de vuelta, o destruir ambos monstruos. La solución encontrada se basa en el patrón *State*, en el cual cada carta delega su respuesta ante ataques en ella y responde de manera acorde a lo anteriormente dicho.
* ***Double dispatch:*** Podemos ver el caso anterior de otro modo, en vez de ubicarnos desde la vista del monstruo atacado podemos alejarnos un poco más y darnos cuenta que los ataques no depende del estado de solo uno si no que del de ambos. Un monstruo en estado de defensa no puede atacar, en cambio si está en estado de ataque si puede, pero el resultado varía según el estado del monstruo atacado y por lo tanto al tener dos factores que cambian los resultados nos basamos en el patrón *Double Dispatch* en el cual mediante mensajes entre las clases involucradas “recolectamos” el estado de cada una de ellas sin preguntarles explícitamente y luego se procede de manera acorde.

* ***Fases de juego:*** Como se observó anteriormente, las fases del juego sirven como su estado, pasando de uno a otro según las acciones del jugador y por lo tanto resulta claro que el patrón en el cual nos basamos fue *State pattern.*
* ***Cartas:*** Tenemos una gran cantidad de cartas y muchos métodos en común, por lo cual para evitar la repetición de código entre las clases hijas nos basamos en el *Template patter* en el ubicamos aquel código idéntico en la clase madre y dejamos los métodos específicos en las clases hijas.
* ***Invocacion de Exodia:*** Exodia es un monstruo especial que está distribuido en cinco cartas y que en caso de que el jugador las contenga a todas en su mano lo hace ganador automáticamente. La solución encontrada para este problema fue basarnos en el patrón *Visitor Pattern* en el cual tenemos a una clase *InvocadorExodia*  que visita las cartas de la mano y se fija si se encuentran las cinco partes y en el caso de encontrarse hace ganador al jugador.
* ***Modelo, vista y controlador:*** Como es visible en el diagrama de paquetes para manejar los aspectos visuales del juego nos basamos en el patrón MVC.

1. ***Excepciones***

Para finalizar, se va a describir las excepciones que pueden ocurrir a lo largo de la ejecución del programa, si bien algunas las utilizamos para que sean notificadas al usuario que esté utilizando esta aplicación, otras son para la utilización dentro del flujo del programa

***PosicionInvalidaException:*** Esta excepción es lanzada en el caso que cuando se quiera insertar una carta en una posición, pero la posición es inválida. Se encarga de lanzar esta excepción

***PosicionOcupadaException:*** Ocurre en el caso que cuando se quiera insertar una carta en una posición y dicha posición está ocupada. Se encarga de lanzar esta excepción

***NivelInvalidoException:*** Esta excepción ocurre en el caso de que se ingresa un nivel de Carta Monstruo mayor a 10 o menor a 1. Se encarga de lanzar esta excepción la clase Nivel cuando se está creando el nivel.

***CantidadDeSacrificiosInvalidaException:*** Ocurre cuando se quiere colocar una carta que requiere sacrificios, pero no hay una cantidad necesaria de ellos. Se encarga de lanzarla la clase NivelMedio o NivelSuperior, debido a que depende del nivel que sea la carta, y se lanzará en el método que quiere sacrificar.

***NoSePuedeAtacarDirectamenteException:*** Esta excepción ocurre en el caso de que se quiera atacar a los puntos de Vida del Oponente directamente, pero en su zona de Monstruos tiene una Carta. La clase Lado es la encargada de lanzarla