Informe

FIUBA - 75.07

Algoritmos y programación III

Trabajo práctico 2: Al-Go-Oh!

1er cuatrimestre, 2018

(trabajo grupal de 4 integrantes)

Alumnos:

Nombre	Padrón	Mail
Di Maria, Franco Martin	100498	franki.dimaria@gmail.com
Loguercio, Sebastian Ismael	100517	seba21log@gmail.com
Marchesini, Ariel	100625	arielmrchsn@gmail.com
Sanchez, Cristian	98112	sanchezcristiansebastian@gmail.com

Fecha de entrega final:

Tutor:

Comentarios:

Índice

Supuestos 2

Modelo de dominio 2

Diagramas de clases 9

Diagramas de secuencia 17

Diagrama de Paquetes 27

Diagramas de estado 29

Detalles de implementación 30

Detalles de la interfaz gráfica 36

Excepciones 37

Supuestos

- El usuario no elige que monstruos se sacrifican del campo. El campo sacrifica la cantidad necesaria (si es que la hay) de los monstruos de menor ataque.
- El cementerio es único, las cartas de cualquier jugador que son destruidas van todas al mismo cementerio.
- Los monstruos del juego sólo pueden atacarse entre sí. El único monstruo que puede atacar al jugador es Jinzo7.
- El monstruo insecto come hombre siempre destruye al monstruo que lo atacó cuando se encuentra boca abajo. Sólo se puede seleccionar a quien destruir cuando su efecto es activado en el turno del jugador a quien pertenece.
- Sólo puede haber una carta de campo activa por jugador. Al reemplazar una carta de campo por otra, en las cartas que ya estaban en el campo el nuevo efecto se acumula con el antiguo y en las futuras cartas a colocar sólo se aplicará el efecto de la carta de campo activa.

Modelo de dominio

Conjunto de entidades creadas, y sus responsabilidades:

Partida : Representa una partida. Tiene la responsabilidad de saber quien es el ganador y perdedor de la partida y de terminar la misma una vez establecido alguno de los dos.

Jugador : Representa a un jugador del juego. Tiene la responsabilidad de mostrar su vida y actualizarla al recibir daños y ataques. También se responsabiliza por ganar y perder, avisando a la partida sobre ello.

Mano : Representa una mano de Cartas. Tiene la responsabilidad de contener cartas, agregarlas y colocarlas en el campo según sea requerido. También se responsabiliza por ganar en caso de contener todas las partes de Exodia. Sabe cuántas cartas de cierto tipo se agregaron y siguen estando.

Mazo : Representa el mazo de cartas de un jugador. Tiene la responsabilidad de ir sacando y agregando cartas a alguna mano.

ListaMonstruos : Representa un conjunto de monstruos, tiene la responsabilidad de devolver sub-ListaMonstruos con los monstruos boca arriba. Puede devolver el monstruo con el menor ataque. También devuelve subconjuntos de monstruos de algún tipo específico.

Cementerio : Representa un cementerio. Tiene la responsabilidad de acumular las cartas que son destruidas.

Campo (Reiniciable) : Representa el conjunto de cartas colocadas en el campo. Administra la colocación y destrucción de distintos tipos de objetos Carta : CartaMonstruo (esta delegada en una ListaMonstruos), CartaMagica, CartaTrampa, CartaDeCampo, así como de saber si las contiene. Tiene la responsabilidad de realizar sacrificios, fusiones, de aumentar y disminuir el ataque y la defensa de los monstruos al setear efectos de campo propios y del enemigo. También se responsabiliza por desactivar efectos temporales en todos sus monstruos.

Carta (Interfaz) : responde a los mensaje colocarse (en Campo), colocarse (en Mano), colocarBocaArriba, colocarBocaAbajo, toString, imagen.

Botín : Representa el resultado de una batalla entre monstruos. Tiene la responsabilidad de saber los daños que le corresponden a cada Jugador y de saber los monstruos que fueron destruidos en la batalla. Inflige estos daños recibiendo los Jugadores a quienes corresponden dichos daños y destruye estos monstruos recibiendo los Campos donde deben ser destruidos.

Posición (Interfaz): responde a los métodos obtenerPuntos(...), atacar(), recibirAtaque(...), matar(...), enCaracteres().

PosAtaque (Posición) : Representa la posición de ataque de un Monstruo. Tiene la responsabilidad de calcular los puntos de ataque del Monstruo que compone. Tiene la responsabilidad de matar (setear en un Botín) al monstruo que está atacando o del cual está siendo atacado.

PosDefensa (Posición) : Representa la posición de defensa de un Monstruo. Tiene las mismas responsabilidades que PosAtaque, pero con distinto comportamiento: el metodo matar de la posición de defensa, no agregar ningún muerto al Botín. Además, la posición de defensa no inflige daño a ningún Jugador.

PosDormido (Posición) : Es la Posición inicial de cada Monstruo. Si un Monstruo con ésta Posición recibe el mensaje de atacar a otro, PosDormido lanza la excepción MonstruoNoPuedeAtacarError. Representa a los monstruo que no se han colocado en el Campo.

Monstruo : Representa un monstruo. Tiene la responsabilidad de atacar y recibir el ataque de otro monstruos, devolviendo un objeto Botín con el resultado de la batalla. Además se responsabiliza de aumentar/reducir su ataque y defensa según sea requerido. Administra sus aumentos (de ataque o defensa) temporales, y se encarga de eliminarlos una vez requerido. También tiene la responsabilidad de actualizar su posición (ataque/defensa) según sea requerido, en tiempo de ejecución.

Invocación (Interfaz) : responde al método activar sobre algún campo.

InvocacionSacrificio (Invocación) : Se inicia con una cantidad de estrellas, dependiendo de la misma se establece la cantidad de monstruos a sacrificar. Al activarse, le pide al campo que recibió,sacrificar tal cantidad de monstruos.

InvocacionFusion (Interfaz) : Se inicia con un carta de algún tipo, y la cantidad de cartas de dicho tipo a sacrificar. Al activarse, le pide al campo una fusión con dichos parámetros.

CartaMonstruo (Carta) : Representa una carta monstruo. Se responsabiliza de administrar su Boca cambiandola de arriba a abajo según sea requerido. También se responsabiliza de colocarse tanto en una Mano como en un Campo, asegurándose que se realicen los sacrificios correspondientes. Otra responsabilidad que tiene es la atacar y recibir el ataque de otra CartaMosntruo.

CartaMonstruoExodia : hereda de CartaMonstruo. Al colocarse en Mano,en caso de que estén las 5 partes de Exodia en la misma, se gana la partida inmediatamente.

CartaMonstruoJinzo7: hereda de CartaMonstruo. Es la única CartaMonstruo que puede atacar directamente a los puntos de vida del jugador.

CartaMonstruoComeHombres: hereda de CartaMonstruo. Al ser volteada de boca abajo a a boca arriba , puede destruir una CartaMonstruo del campo enemigo. En caso de ser atacada Boca Abajo, destruye la CartaMonstruo que lo atacó.

CartaMonstruoDragon (de ojos azules) : hereda de CartaMonstruo. Se utiliza como tipo para identificar las cartas necesarias para invocar al DragonDefinitivo (de ojos azules).

CartaMonstruoDragonDefinitivo (de ojos azules) : hereda de CartaMonstruo. Necesita del sacrificio de 3 CartaMonstruoDragon para ser invocada. Utiliza InvocacionFusion.

NoCartaMonstruo : hereda de CartaMonstruo. Se inicializa con un Monstruo cualquiera. Casi todos métodos fueron redefinidos para que se evite su uso elevando un excepción. Se utiliza como CartaMonstruo nula (NullPattern).

Magia (interfaz) : responde al método activar.

AgujeroOscuro (Magia) : Se inicializa con dos objetos Campo (del jugador y del enemigo). Al activarse destruye todos los monstruos en ambos campos.

Fisura (Magia) : Se inicializa con un objeto Campo correspondiente al enemigo. Al activarse destruye la CartaMonstruo de menor ataque que esté boca arriba.

OllaDeLaCodicia (Magia) : Se inicializa con un objeto Mano y Mazo, correspondientes a un mismo Jugador. Al activarse, saca dos cartas del Mazo y las coloca en la Mano.

CartaMágica (Carta) : Representa una carta mágica. Las cartas mágicas se inicializan con una magia que se activa al colocar la primera, boca arriba. Tiene la responsabilidad de activar o no activar su magia según esté boca arriba o boca abajo, respectivamente.

Trampa (Interfaz) : Responde al método activar (sobre dos objetos Monstruo, y un botín).

TrampaNula (Trampa) : al activarse no afecta el resultado de la batalla, y devuelve el mismo botín que recibió.

Refuerzos (Trampa) : al activarse aumenta el ataque del monstruo aliado, el cual se mantiene vigente hasta finalizar el turno, devolviendo un nuevo Botín con el resultado de hacer pelear a los monstruos recibidos con este nuevo aumento de ataque.

CilindroMagico (Trampa) : al activarse devuelve un nuevo Botín, infligiendo todo el daño del monstruo atacante en el Jugador atacante, sin dejar monstruos destruidos.

CartaTrampa (Carta) : Representa una carta trampa. Están compuestas por una Trampa. Tiene las responsabilidad de activar su Trampa sobre dos monstruos y un Botín afectando de alguna manera el resultado de la batalla, cuando sea requerido.

EfectoDeCampo : es una clase abstracta que se inicializa con dos Objetos Campo correspondientes al jugador y al enemigo. Posee metodo activar no definido

EfectoSogen : hereda de EfectoDeCampo. Al activarse aumenta la defensa de los monstruos del jugador en 500, y el ataque de los monstruos del enemigo en 200. Este efecto dura hasta que es reemplazado por otro, y aplica sobre nuevos monstruos colocados en el campo, también.

EfectoWasteland : hereda de EfectoDeCampo. Al activarse aumenta el ataque de los monstruos del jugador en 200, y la defensa de los monstruos del enemigo en 300. Mismas características que el efecto Sogen.

CartaDeCampo (Carta) : Se inicializa con un EfectoDeCampo. Tiene la responsabilidad de activarse al colocarse en un campo. Al activarse activa su EfectoDeCampo.

Efecto (Interfaz) : responde al método activar(sobre CartaMonstruo), y activar (sobre ListaMonstruos).

EfectoDestruirMonstruo (Efecto) : Se inicializa con un objeto Campo correspondiente al enemigo. Al activarse sobre una CartaMonstruo lo destruye. Al activarse sobre una ListaMonstruos, destruye todos los monstruos de dicha ListaMonstruos

EfectoDestruirElDeMenorAtaque : Se inicializa con un objeto Campo correspondiente al enemigo. Al activarse sobre una CartaMonstruo no hace nada. Al activarse sobre una ListaMonstruo, busca , a través de sus métodos, al monstruo boca arriba de menor ataque y lo destruye.

ConstructorDeCartas : Se responsabiliza de saber como crear todas las cartas con las que el usuario puede jugar. Tiene un método para crear cada carta en particular, por ejemplo: aitsu() crea una CartaMonstruo con cierta cantidad de daño y defensa, 5 estrellas; agujeroOscuro crea una CartaMagica compuesta de una Magia AgujeroOscuro que recibe los campos de ambos jugadores; y así sucesivamente con cada carta del juego, que se haya decidido implementar. Cada creación de carta, recibe parametros especiales para dicha carta.

RandomizadorCartas : Se utiliza para llenar un mazo con cartas de forma aleatoria. Se responsabiliza por construir cartas mediante el constructor de cartas, de forma aleatoria, y las va agregando al mazo. En particular, se responsabiliza por que no haya más de 5 CartaMonstruoExodia en el mazo, y cada una correspondiente a una parte distinta de Exodia.

Atacable (Interfaz) : responde al método recibirAtaque que recibe la CartaMonstruo atacante, el campo aliado. Se utiliza en CartaMonstruo y Jugador para poder incluirlos en un mismo ComboBox de la interfaz gráfica.

Alerta : se responsabiliza de abrir ventanas de error en la vista. Se utiliza para evitar código repetido.

Actualizable (interfaz) : responde al método actualizar

Controladores :
ControladorCampoMonstruos (Actualizable) : se inicializa con el campo y un Box donde se muestran las cartas monstruo de dicho campo. Posee un método actualizador que limpia el Box y lo vuelve llenar con las imágenes de las cartas monstruos del campo.

ControladorCampoMagicasTrampas (Actualizable) : se inicializa con el campo y un Box donde se muestran las cartas magicas y trampa de dicho campo. Posee un método

actualizador que limpia el Box y lo vuelve llenar con las imágenes de las cartas magicas y trampas del campo.

ControladorJugador (Actualizable) : se inicializa con un Jugador y un Box donde se muestra la vida del primero. Posee un método “actualizar” que va actualizando la barra de vida del jugador en la vista.

ControladorMonstruoComeHombres (Actualizable) : se inicializa con una CartaMonstruoComeHombres, el campo enemigo a dicha carta, y un actualizador de representaciones. Posee un método activar que abre una ventana que le permite al usuario para seleccionar un monstruo a destruir por la CartaMonstruoComeHombres. Se utiliza en el BotonCambiarBocaYPosicion cuando se voltea de boca abajo a boca arriba una CartaMonstruoComeHombres.

ControladorCartaDeCampo (Actualizable) : se inicializa con un objeto Campo. Posee un método “actualizar” que actualiza la imagen de la CartaDeCampo vigente en un determinado Campo.

ActualizadorDeReprentaciones : se encarga de llamar a los métodos “actualizar” de ciertos controladores que manejan la vista del juego.

Reiniciable (Interfaz) : responde al método reiniciar.

Botones :
BotonSacarCarta (Reiniciable): saca una carta del mazo y la coloca en la mano del jugador.

BotonVerMano : abre una ventana con todas las cartas que tiene el jugador en la mano.

BotonColocarCarta (Reiniciable) : abre una ventana con las cartas que se pueden colocar en el campo, desplegando luego opciones para colocarla boca arriba o boca abajo, y en posición de ataque o defensa si es una CartaMonstruo.

BotonCambiarDeBocaYPosicion : abre una ventana con las cartas monstruo que pueden ser cambiadas de boca y posición, con una serie de desplegables para tomar dichas decisiones.

BotonActivarMagia : abre una ventana con las cartas mágicas que pueden ser activadas desde el campo.

BotonAtacar : abre una ventana que muestra los monstruos con que el jugador puede atacar. Luego de elegir uno, muestra los atacables : monstruos y/o jugador(en el caso de

Jinzo7) a los que puede atacar. Tras estas elecciones, el botón hace pelear a los monstruos mediante la clase Combate, o hace que Jinzo7 ataque directamente al Jugador llegado al caso.

BotonSiguienteTurno : avanza al siguiente, activando los botones del contrincante y desactivando los propios, mediante la clase Turno.

Todos lo botones se inicializan con un mismo ActualizadorDeRepresentaciones, y tras cada operación que realizan, le piden a este actualizador que actualice la vista.

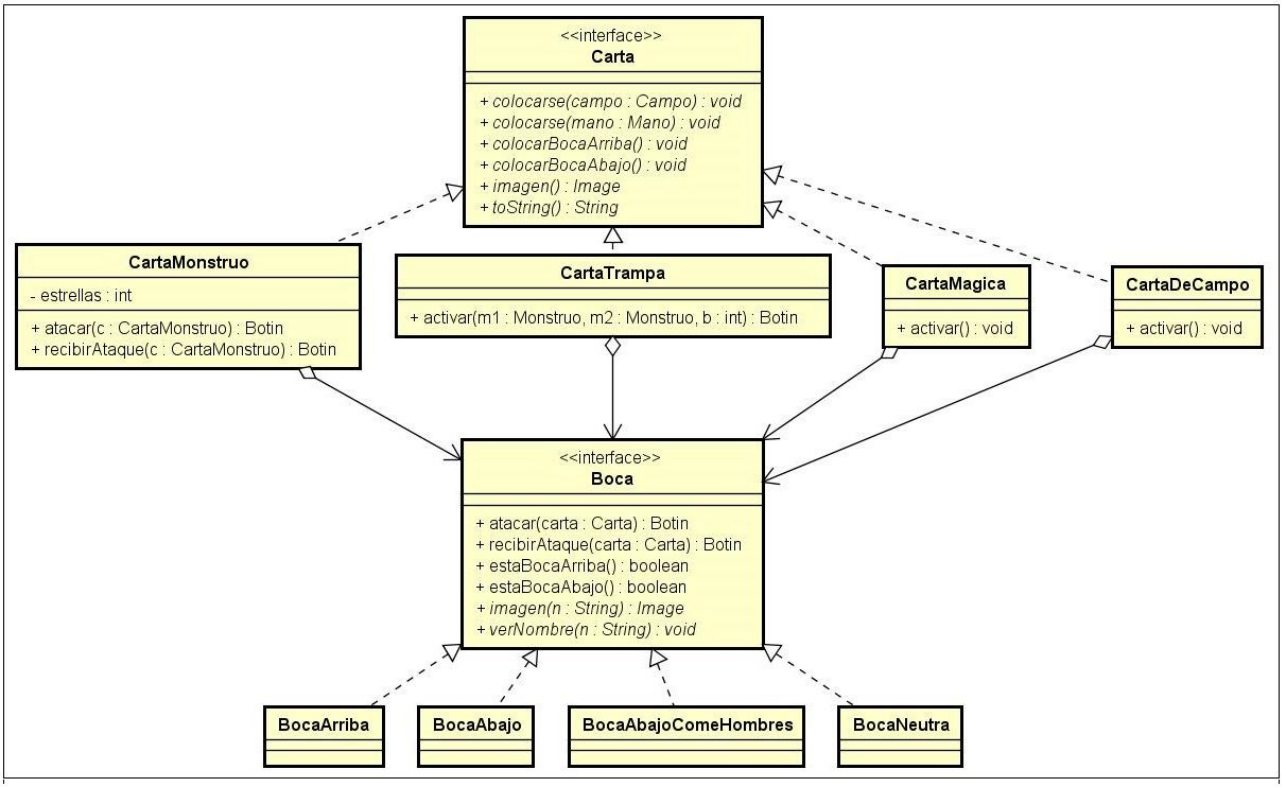
Turno: recibe el conjunto de botones de cada jugador, y objetos reiniciables, como por ejemplo son los campos que tienen monstruos con efectos temporales, o botones específicos que guardan información por turno. Se encarga de activar y desactivar los botones de cada jugador en el cambio de turno. También se responsabiliza por reiniciar ciertos objetos que necesitan volver a cierto estado inicial antes de iniciar el turno.

El modelo está basado en la idea de que las cartas son contenedores, por ejemplo, una CartaMonstruo contiene un monstruo, y una CartaMagica contiene una magia. Lo que nos permite hacer esto es diferenciar entre la acción que toma en combate una carta y el estado en el que se encuentra. Por ejemplo el Monstruo no sabe si la carta que lo contiene está boca arriba o boca abajo, él solo sabe atacar a otro monstruo, y la Carta, que si sabe la boca que tiene le dice al monstruo cuando atacar o cuando no. Además, si se agrega cualquier carta nueva y un comportamiento distinto, con este modelo es fácil de afrontar; Un ejemplo es el monstruo come hombres, está representado como una CartaMonstruoComeHombres que contiene a un Monstruo común, entonces al atacar (responsabilidad del monstruo) no afecta el hecho de que sea un ‘monstruo come hombres’ solo sus puntos de daño y defensa. En cambio, al pasar de estar boca abajo a estar boca arriba, tiene un efecto especial y de esto se encarga la CartaMonstruoComeHombres. Así pudimos solucionar el problema, agregando solo una entidad más al modelo

Diagramas de clases

Diagrama de clases Carta y Boca.

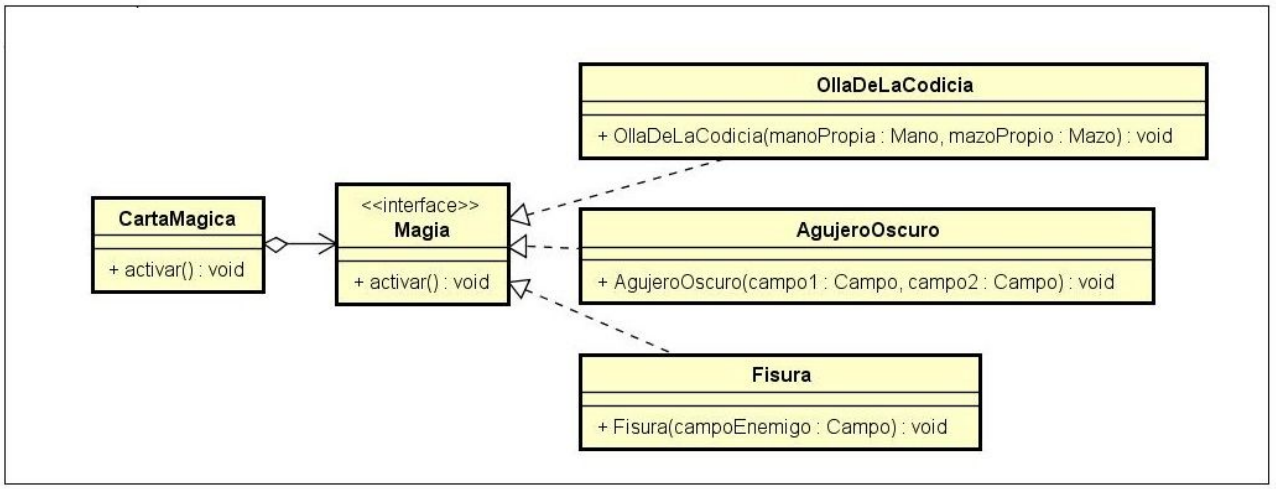
Diagrama 1:



Ver detalles en detalles de implementaciòn.

Diagrama de clase de CartaMágica

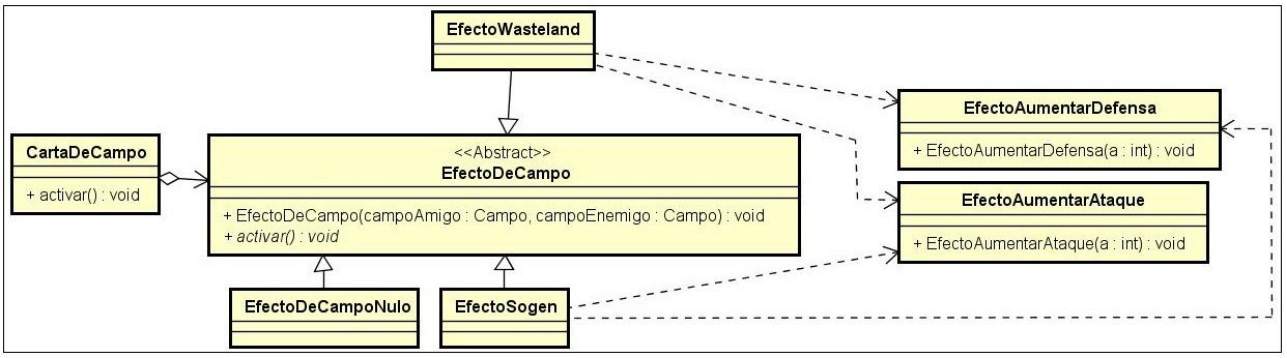
Diagrama 2



Ver detalles en detalles de implementaciòn.

Diagrama de clase de CartaDeCampo

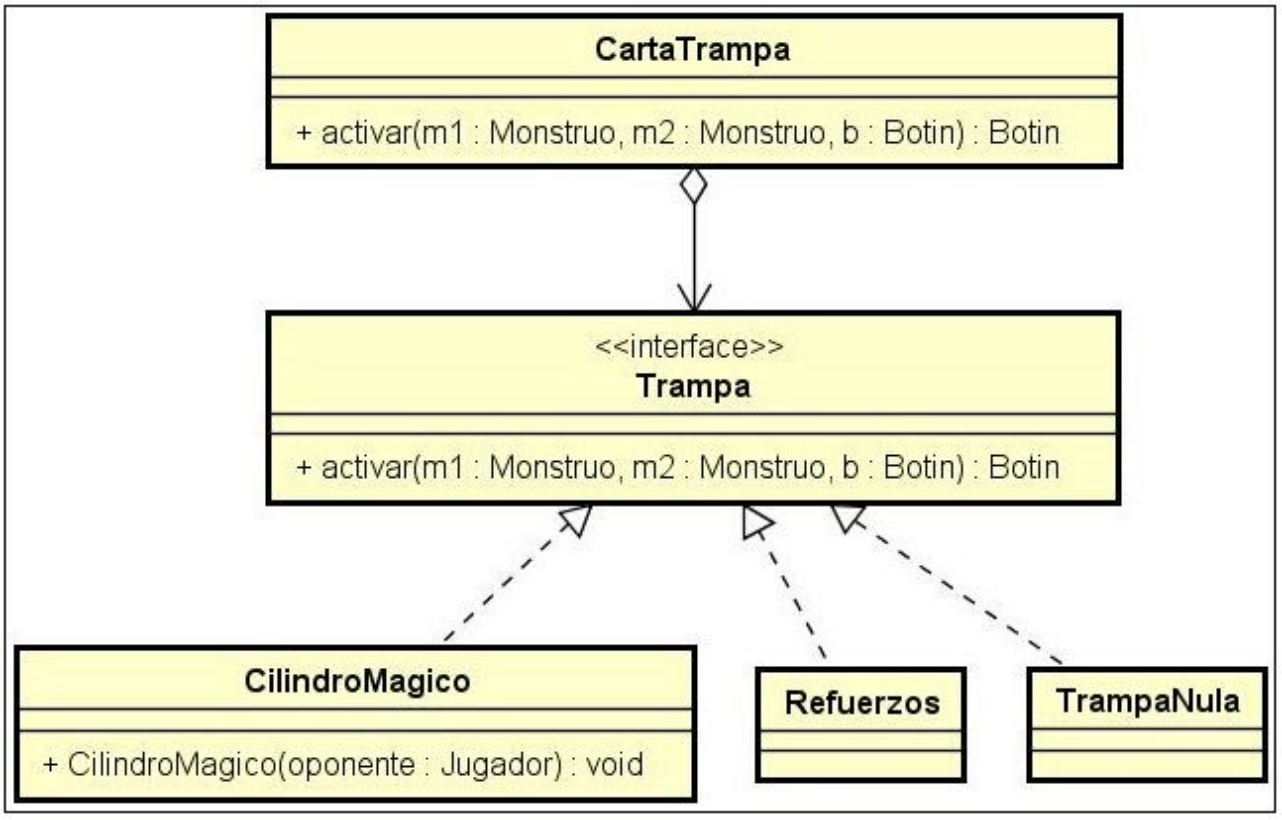
Diagrama 3



Ver detalles en detalles de implementaciòn

Diagrama de clase de CartaTrampa

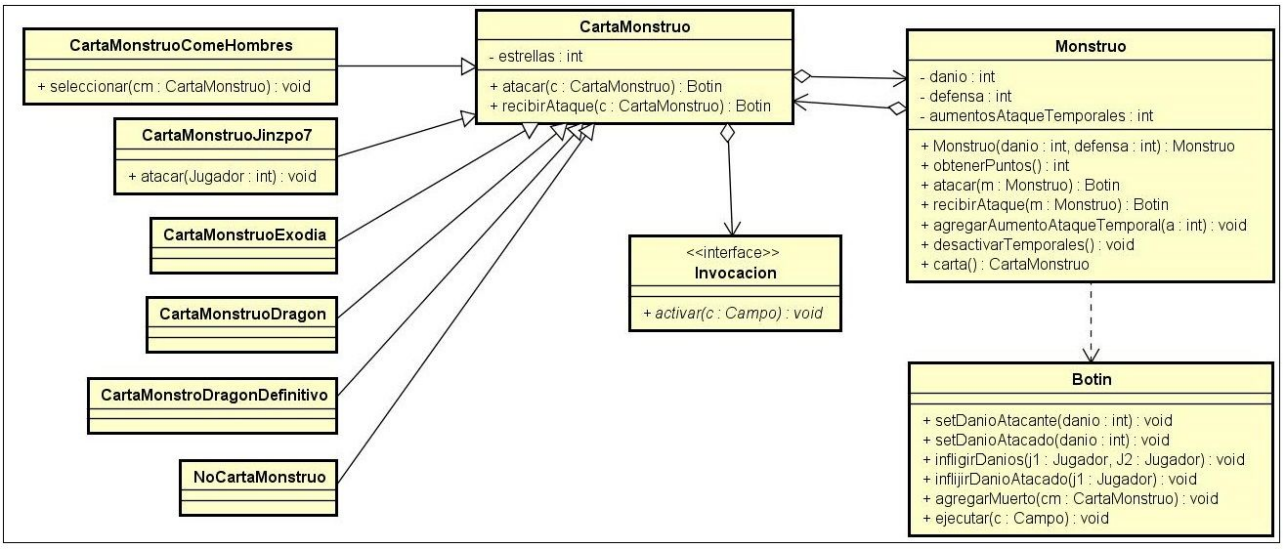
Diagrama 4:



Ver detalles en detalles de implementaciòn.

Diagrama de clase de CartaMonstruo

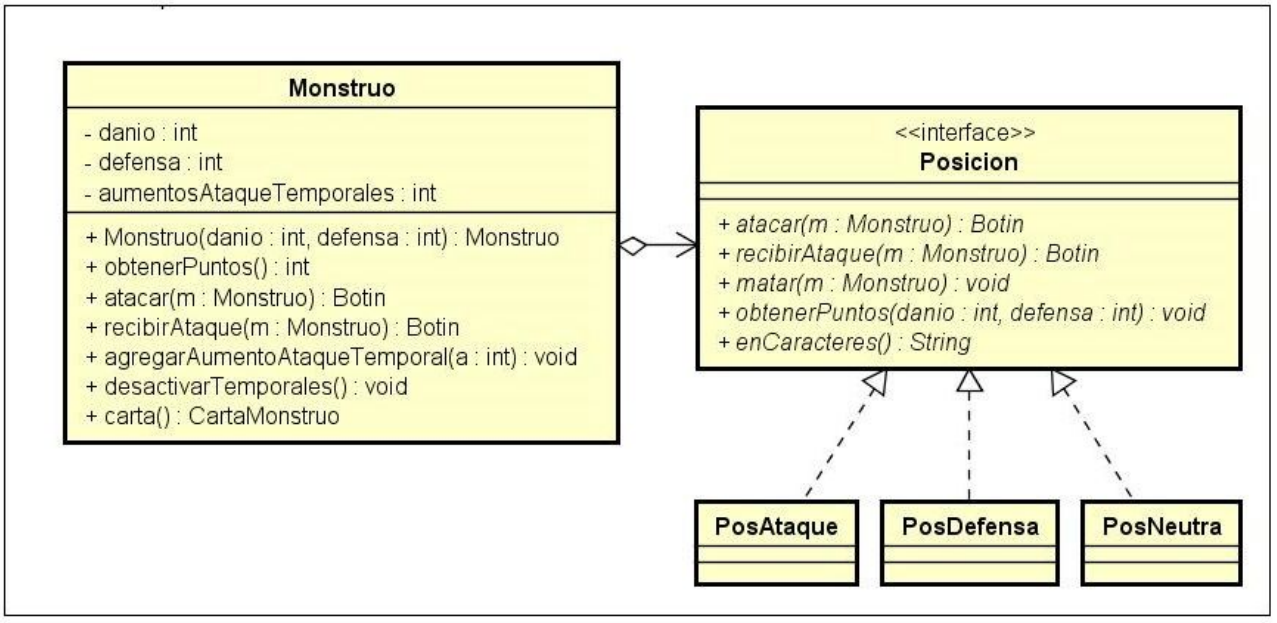
Diagrama 5



Ver detalles en detalles de implementaciòn.

Diagrama de clase de Monstruo

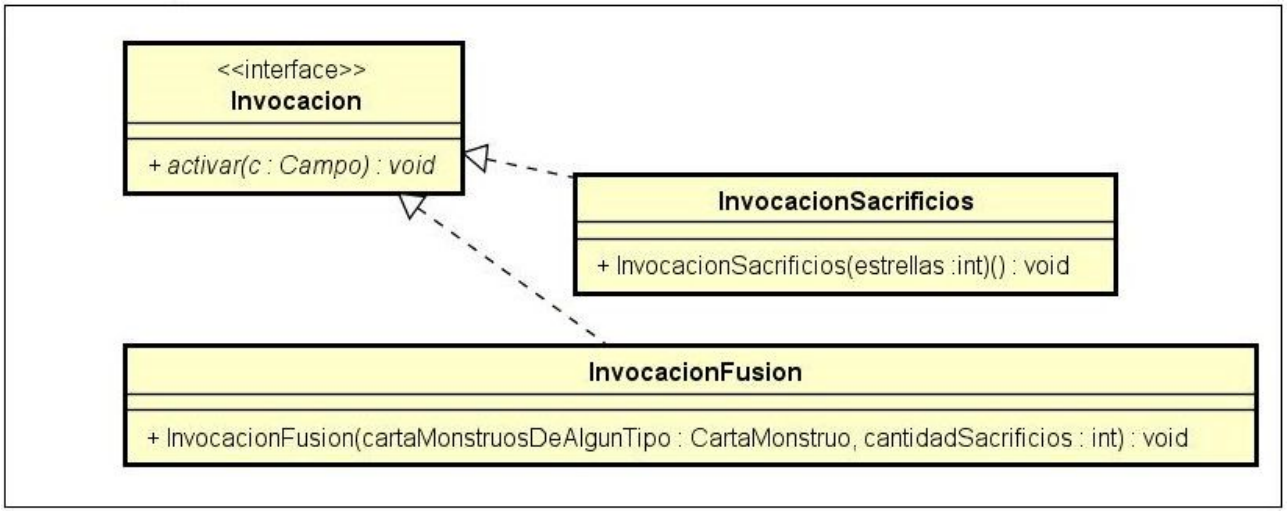
Diagrama 6



Ver detalles en detalles de implementaciòn.

Diagrama de la clases de tipo Invocacion

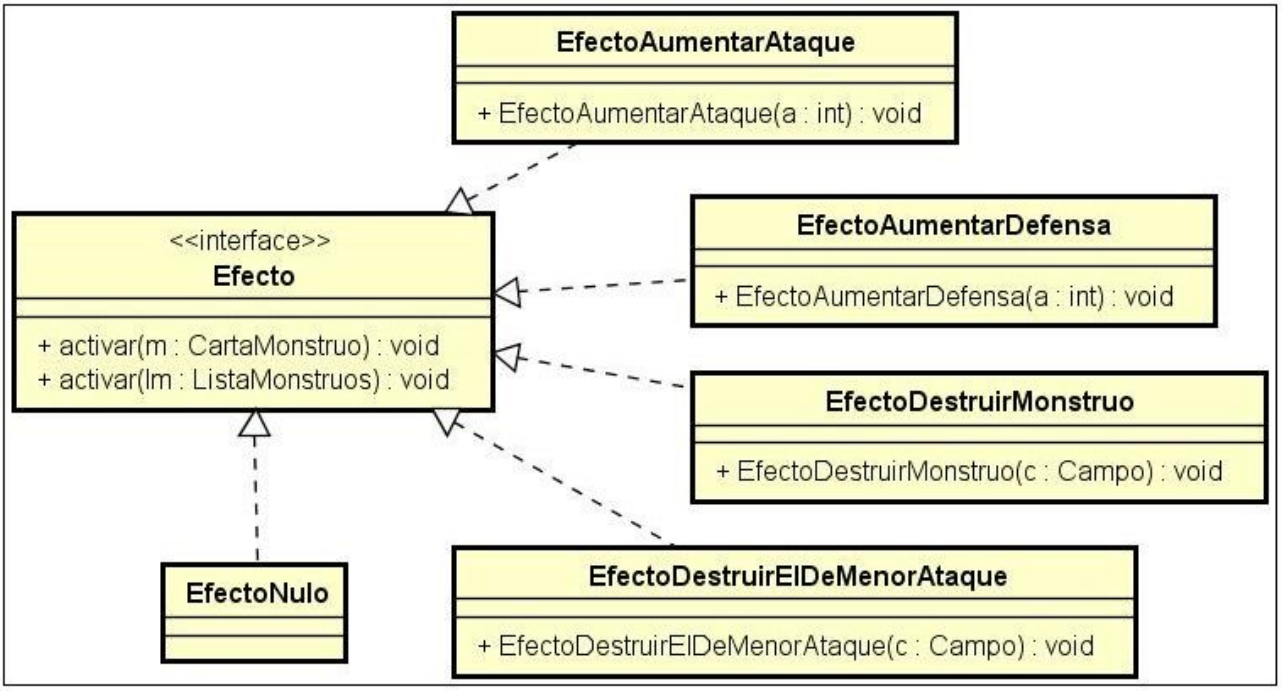
Diagrama 7:



Ver detalles en detalles de implementaciòn.

Diagrama de clase de objetos de tipo Efecto

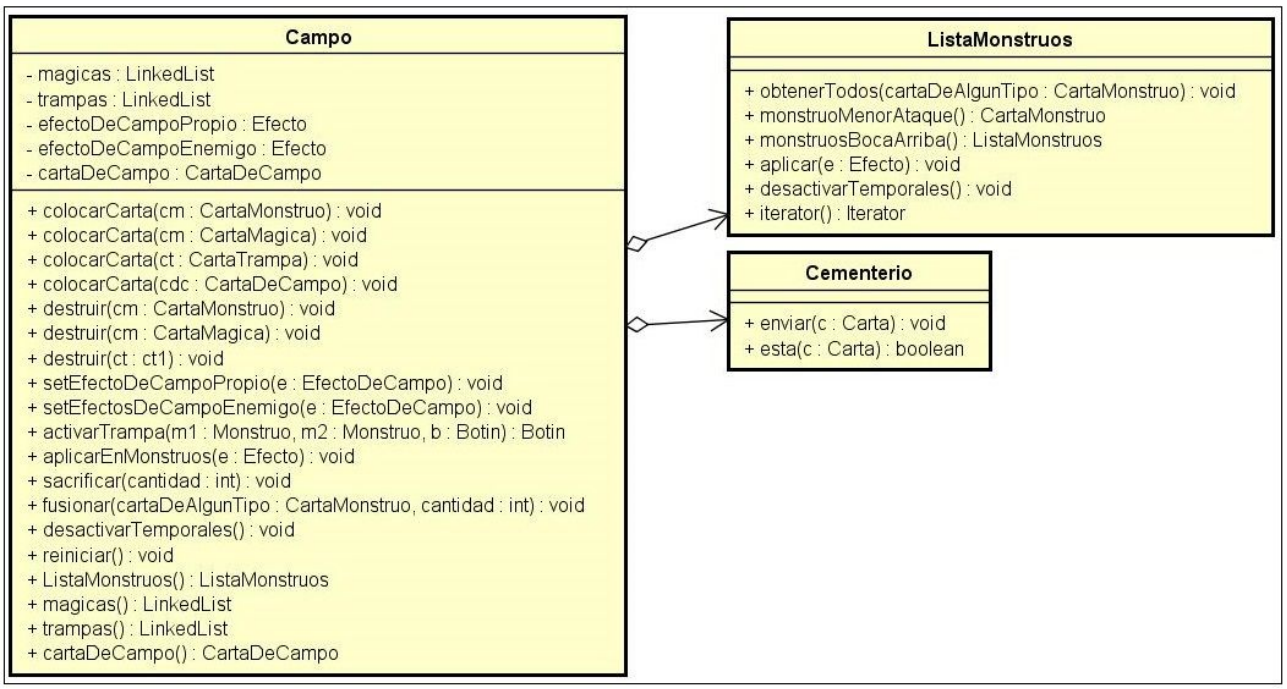
Diagrama 8



Ver detalles en detalles de implementación.

Diagrama de clase de Campo

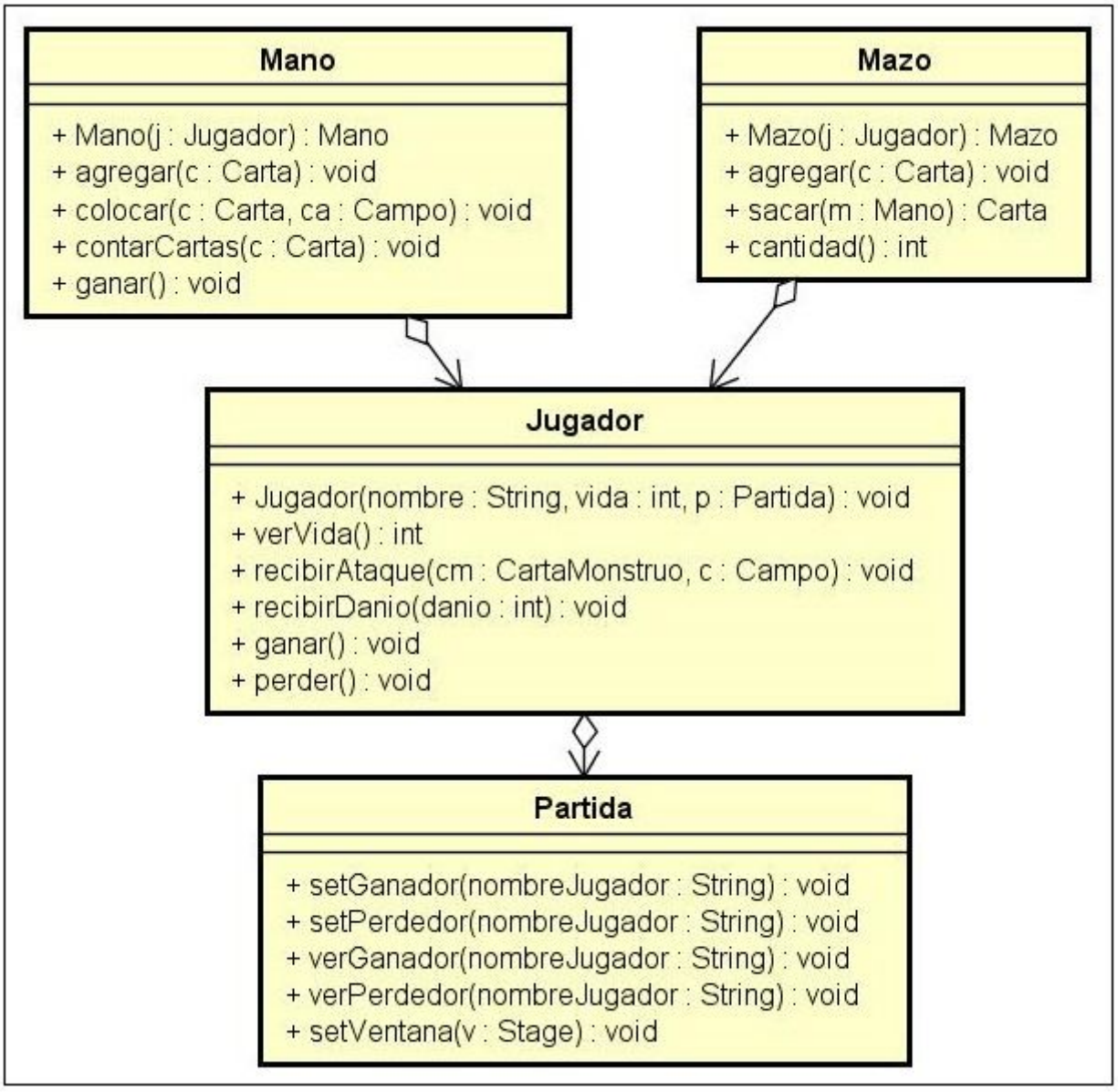
Diagrama 9



Ver detalles en detalles de implementación

Diagrama de clases de Mano, Mazo, Jugador y Partida

Diagrama 10



Ver detalles en detalles de implementación.

Diagrama de clases del ConstructorDeCartas y el RandomizadorCartas

Diagrama 11

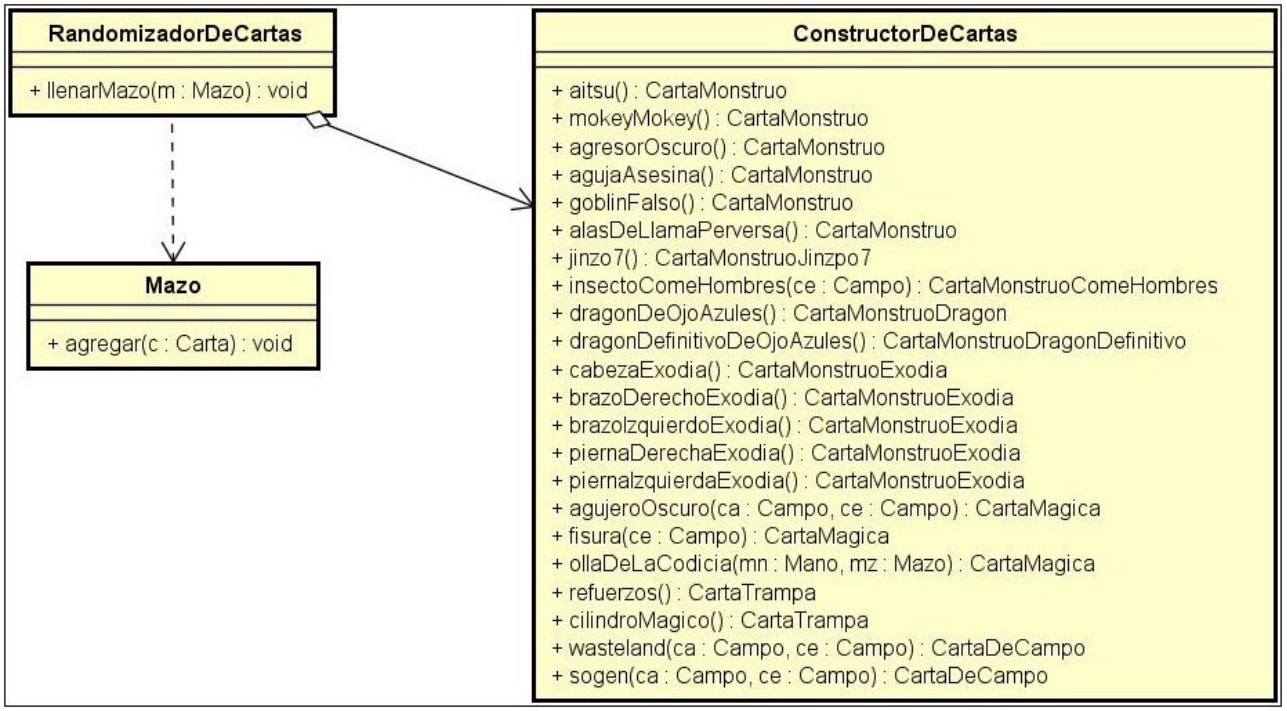
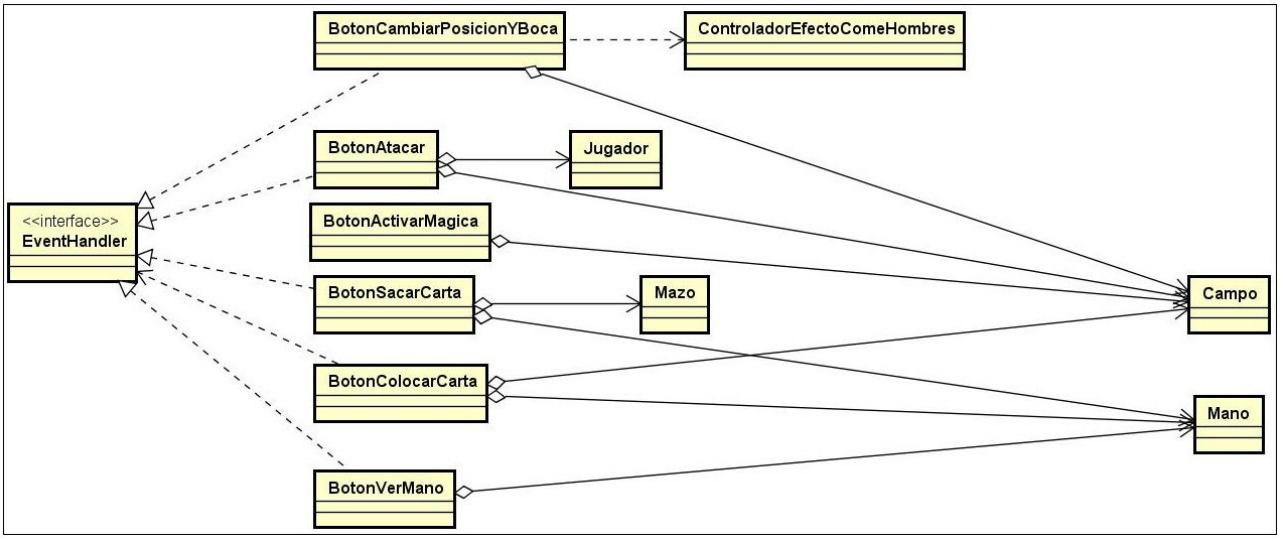


Diagrama de clases de Botones

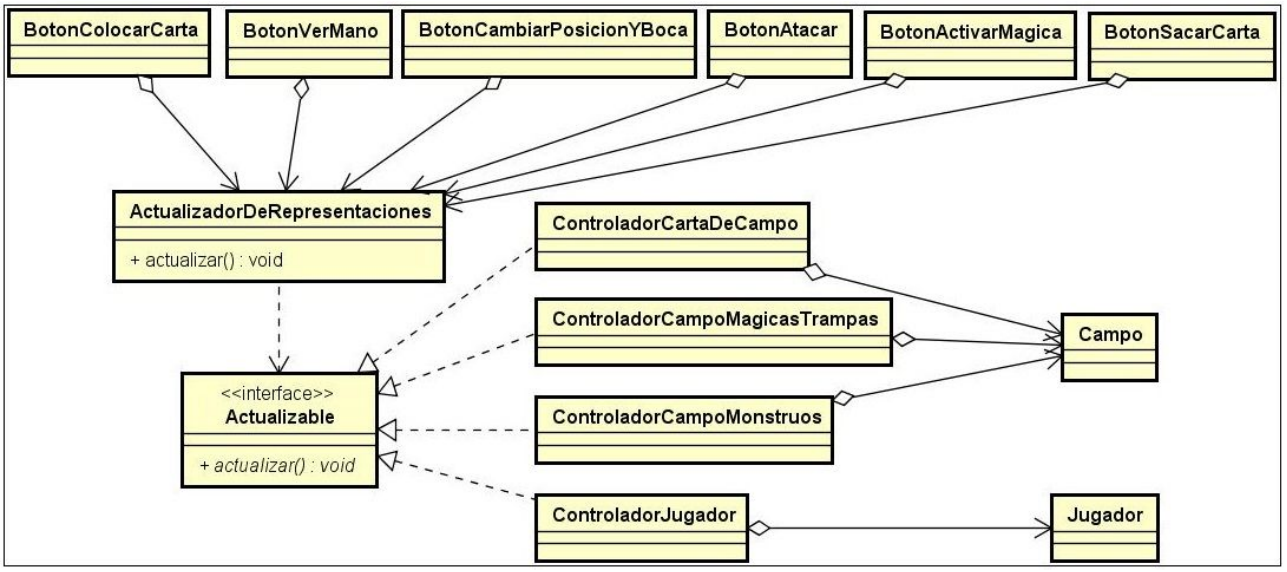
Diagrama 12



Ver detalles en detalles de implementación.

Diagrama de clases Controladores

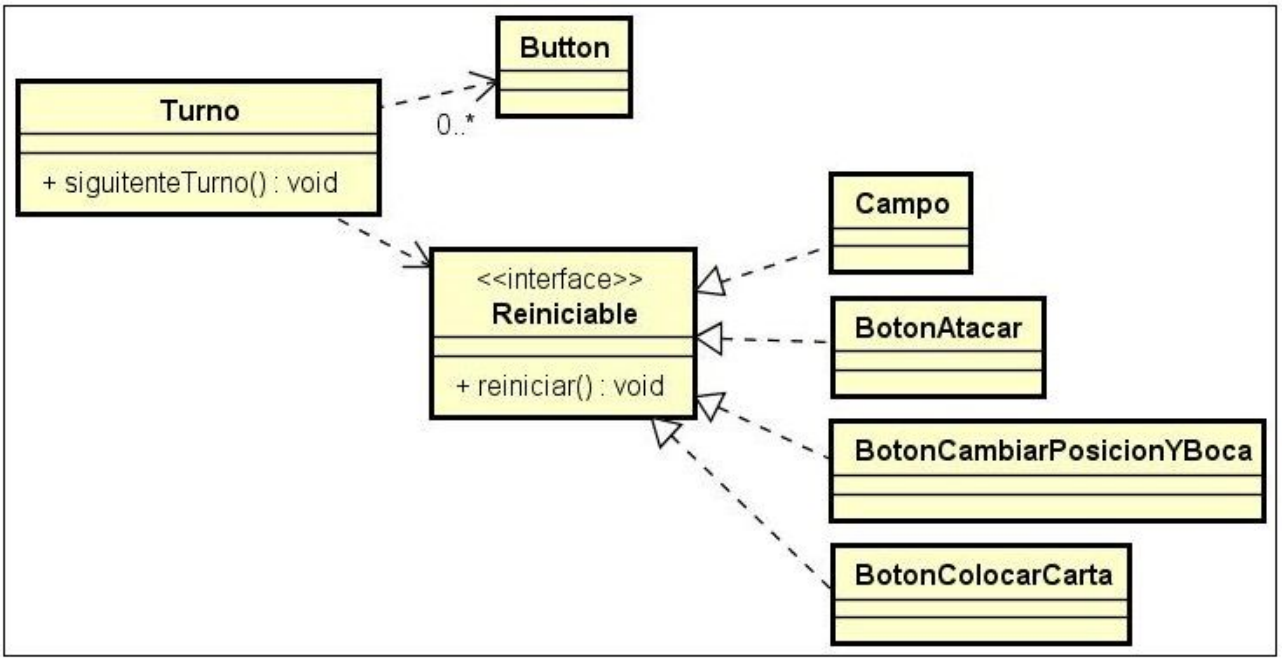
Diagrama 13



Ver detalles en detalles de implementación

Diagrama clases de turno

Diagrama 14

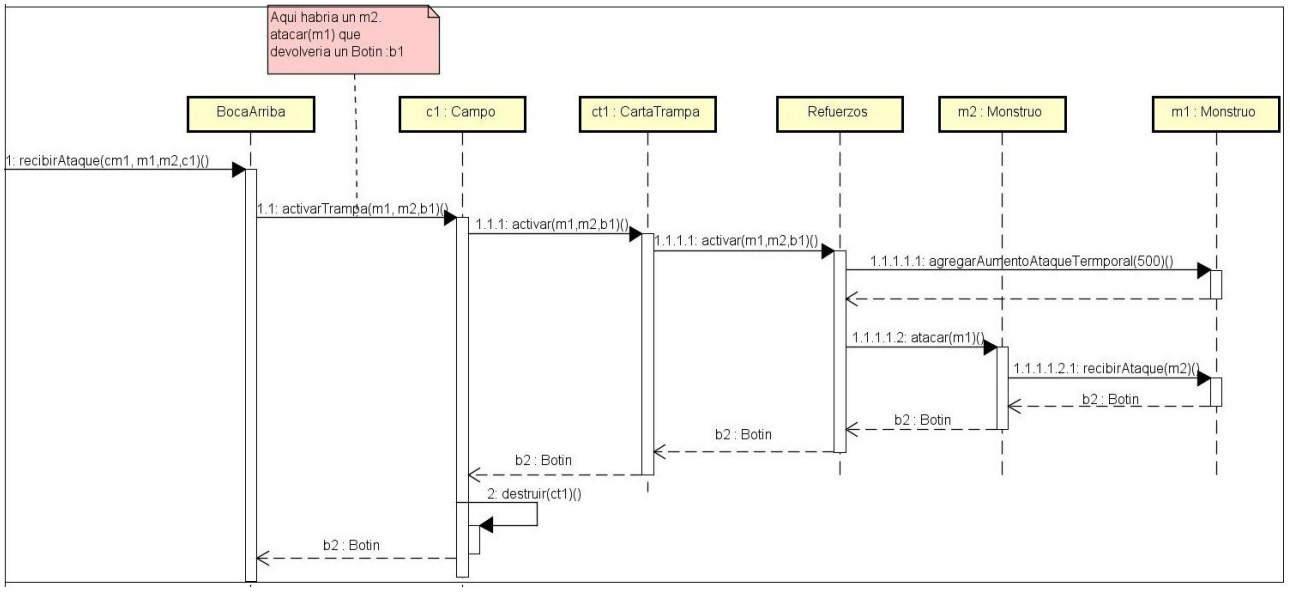


Ver detalles en detalles de implementación

Diagramas de secuencia

Secuencia 1: Una carta cualquiera recibe un ataque y delega el método recibirAtaque() a su Boca (en este caso BocaArriba) y esto hace activar la carta trampa Refuerzos colocada en el Campo donde está la carta que recibió dicho ataque.

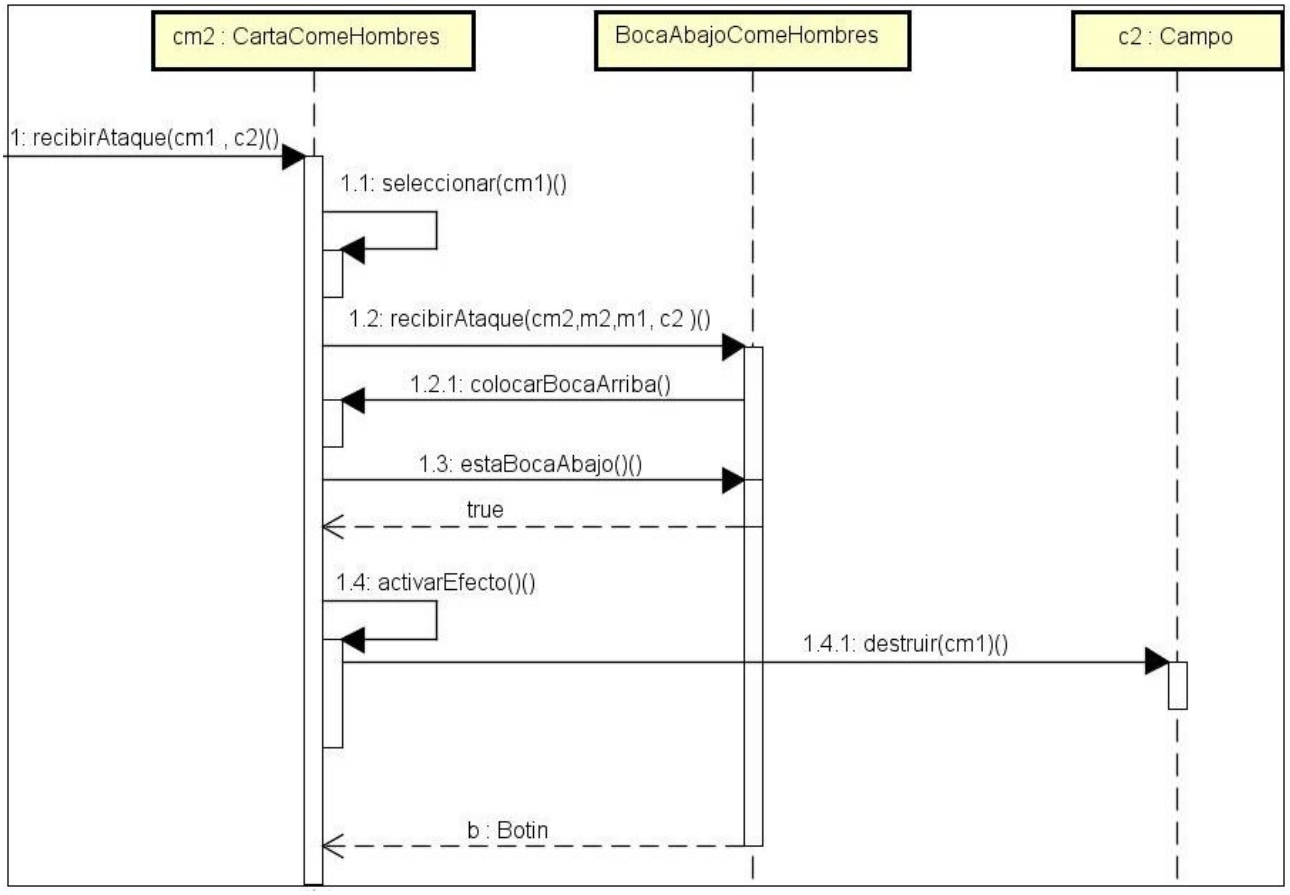
Diagrama 15



La BocaArriba recibe un ataque siendo CartaMonstruo (cm1) y su monstruos m1 los atacados, y el monstruo m2, el atacante.
Se hacen pelear a los monstruos: m2 ataca a m1 generando un botín inicial, y luego sobre estos mismo objetos se activa la trampa. (esta parte se obvió por falta de espacio, y porque no es lo que se quiere mostrar, pero se señalizo con la anotación al principio de la secuencia)
Se le pide al campo activar una carta trampa.
El campo activa una CartaTrampa sobre los monstruos y botín recibidos.
La CartaTrampa activa la Trampa Refuerzos por la que está compuesta, pasandole los mismo parámetros anteriores.
La trampa Refuerzos agrega un aumento de ataque temporal en el monstruo atacado (m1).
Luego se vuelven a hacer pelear a los monstruos con este nuevo ataque.
El campo destruye la carta trampa activada.

Secuencia 2: La carta CartaMonstruoComeHombres (posicionado boca abajo) recibe un ataque de una carta monstruo cualquiera y, por el efecto especial del Come Hombres, el monstruo atacante es destruido.

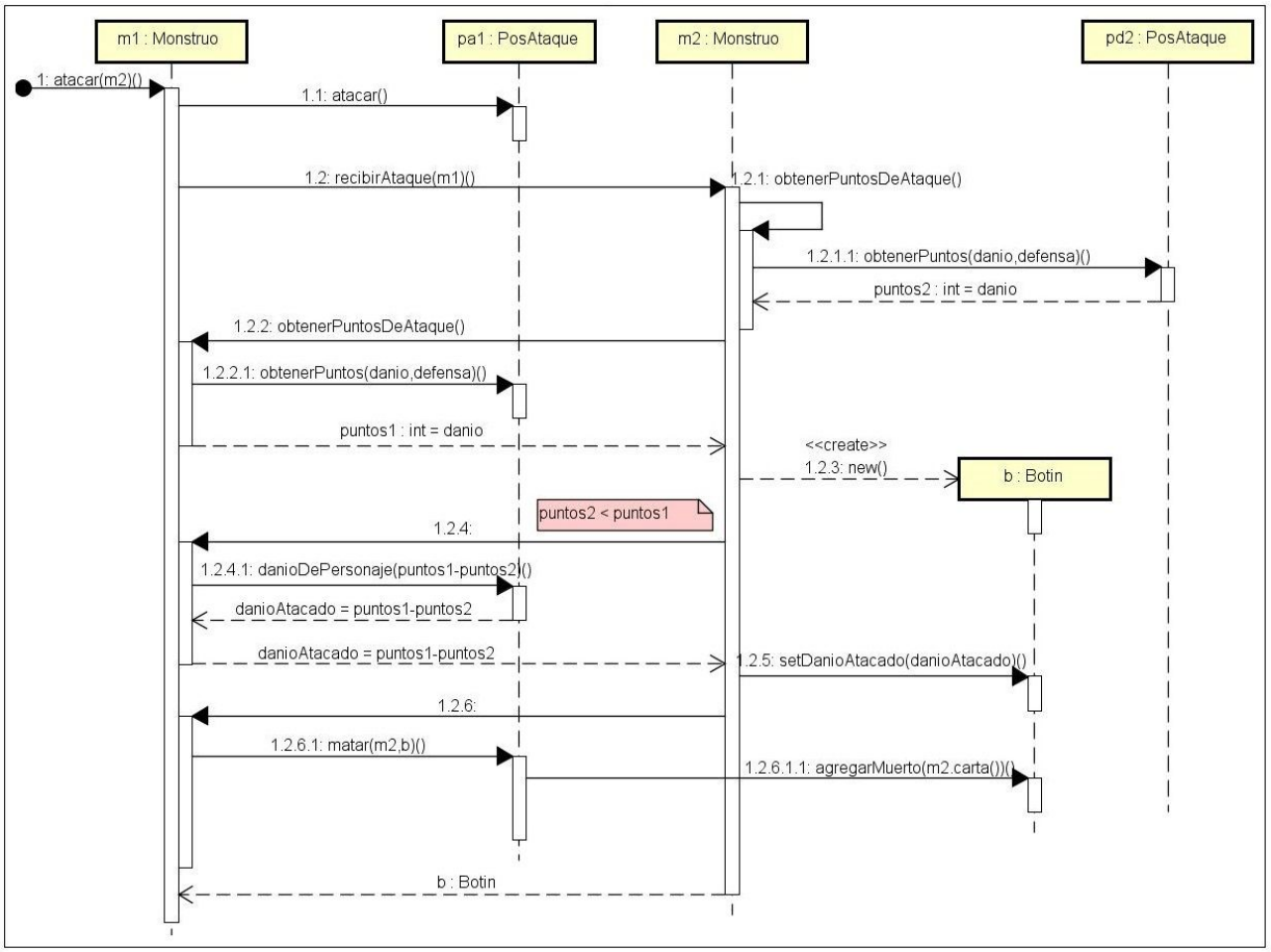
Diagrama 16



Las CartaMonstruoComeHombres recibe un ataque de otra CartaMonstruo.
Se selecciona la carta atacante, y se la guarda.
Se delega el recibir ataque en la Boca actual de la carta. En este caso, está boca abajo, y particularmente, la CartaMonstruoComeHombres tiene un estada particular BocaAbajoComeHombres.
La BocaAbajoComeHombres coloca a su carta boca arriba, cuyo método está implementado de tal manera de preguntar si la boca actual es boca abajo, y de ser así llama a un método interno activarEfecto.
El método activarEfecto, si se seleccionó una carta, hace que campo enemigo la destruya.
Habiéndose destruido la carta atacante, no se ejecuta una pelea entre monstruos, luego la BocaAbajoComeHombres devuelve un botín vacío.

Secuencia 3: El monstruo m1 en posición de ataque ataca a un monstruo m2 en posición de ataque, con menos puntos de ataque que el primero.

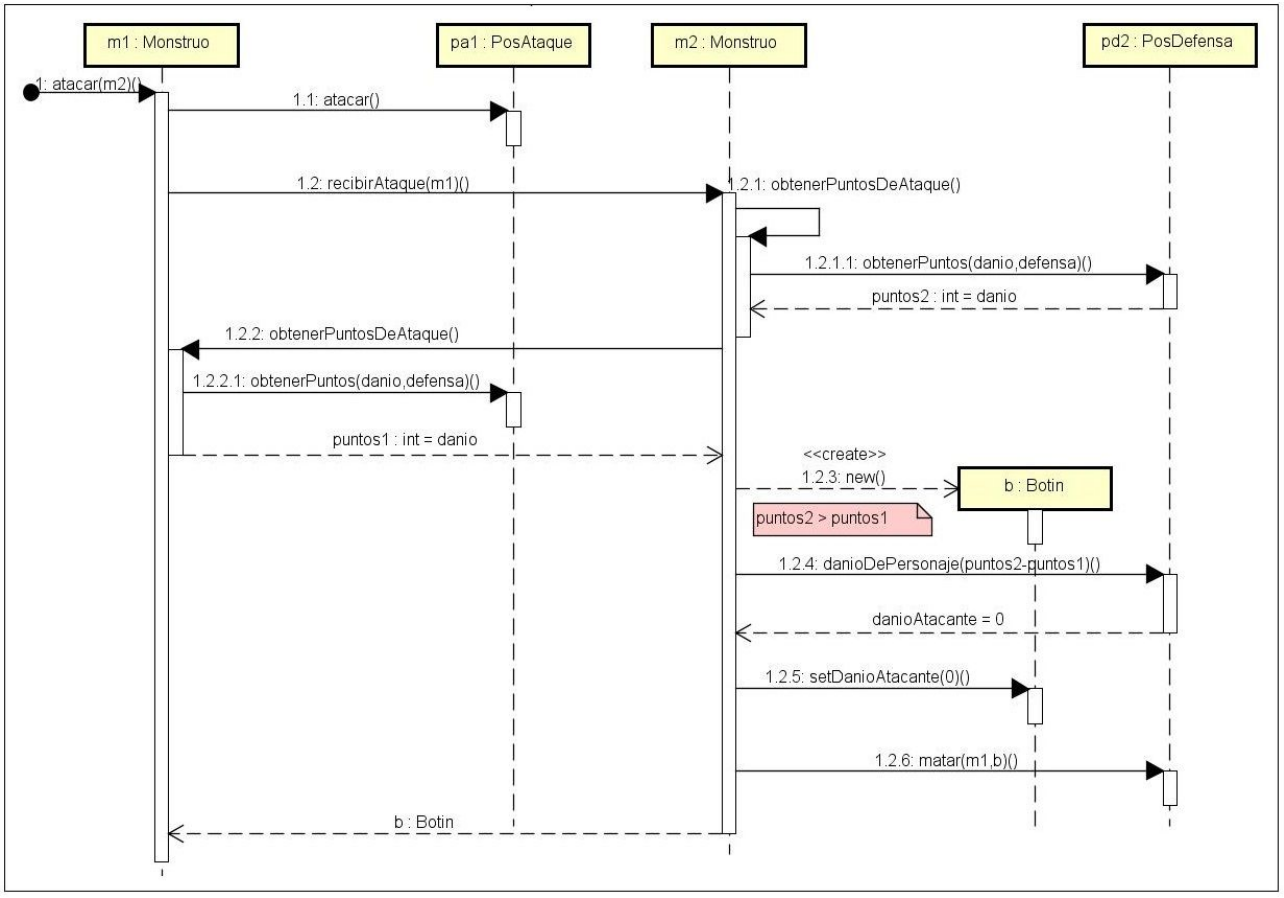
Diagrama 17



El monstruo atacante (m1) le dice a su posición atacar que no levanta ningun error pues puede atacar.
El monstruo atacado (m2) recibe el ataque de m1 .
Se obtienen los puntos de ataque de ambos monstruos, que son pedidos a sus posiciones, pasándose los datos de daño y defensa de cada uno. En ambos, estos puntos son el daño de cada monstruo.
Los puntos de ataque de m1 son mayores que los de m2. Luego se le pide a la posición p1 de m1 el daño que inflige al personaje (contrincante) pasandole y obteniendo la misma diferencia de puntos. Esto se puede realizar pues ambos objetos son de tipo monstruo y por ende conocen su estructura interna. Gracia a ello, se evita generar métodos de sobra en la clase monstruo.
Dicho daño al personaje atacado se guarda en el botín mediante el mètodo setDanioAtacado(...)
Luego se le dice a la posiciòn p1 matar al monstruo atacado, bajo la misma mecánica explicada en el párrafo anterior. El método agregarMuerto del botín recibe una CartaMonstruo que que está guardada en el monstruo a destruir que obtiene por el metodo carta() de Monstruo.
Luego el botín se devuelve.

Secuencia 4: El monstruo m1 en posición de ataque ataca a un monstruo m2 en posición de defensa que tiene más puntos puntos de defensa que puntos de ataque que el atacante m1.

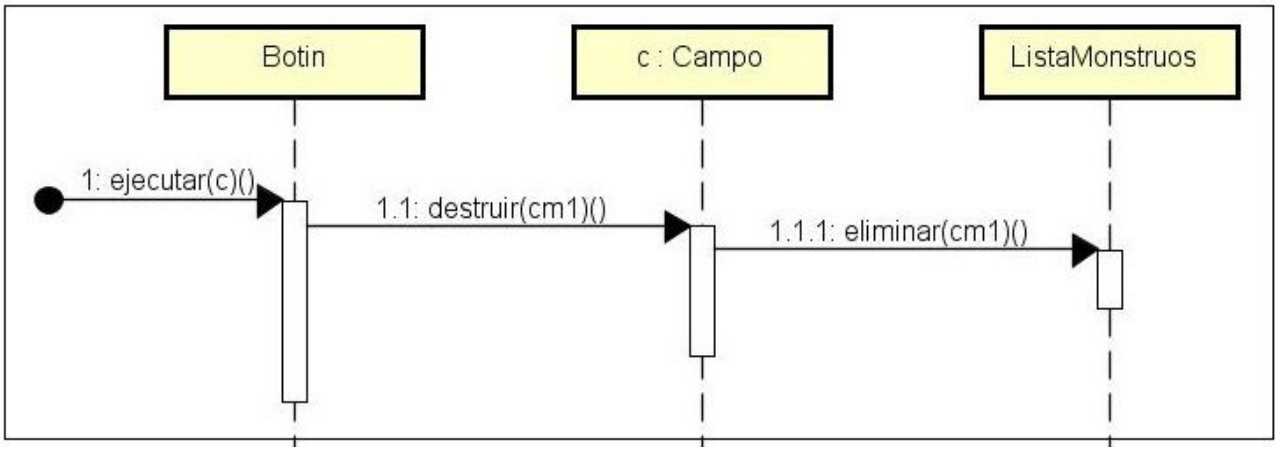
Diagrama 18



El monstruo atacante (m1) le dice a su posición atacar que no levanta ningun error pues puede atacar.
El monstruo atacado (m2) recibe el ataque de m1 .
Se obtienen los puntos de ataque de ambos monstruos, que son pedidos a sus posiciones, pasándose los datos de daño y defensa de cada uno.
La PosAtaque (p1) devuelve el valor de daño recibido.
La PosDefensa (p2) devuelve el valor de defensa recibido.
Con dichos valores se establece una diferencia de puntaje con cierto signo. En este caso, como los puntos de ataque del del atacado es mayor a los del atacante, se le pide a p2 el daño a inflijirle al jugador enemigo. Como la posicion p2 es PosDefensa, se devuelve daño cero. Este daño luego es establecido un nuevo Botin creado, como daña que debe recibir el atacante.
A continuación, se le pide a la posicion p2 que mate al monstruo atacante. Como la posicion p2 es PosDefensa, no mata al atacante, y no se agregar ningun muerto en el botin.
Luego el botín se devuelve con el resultado de la batalla.

Secuencia 5: Ejecución de un Botin. Destruye la/s carta/s que deben ser eliminadas luego de realizado un ataque. En este caso cm1 es la única carta a destruir.

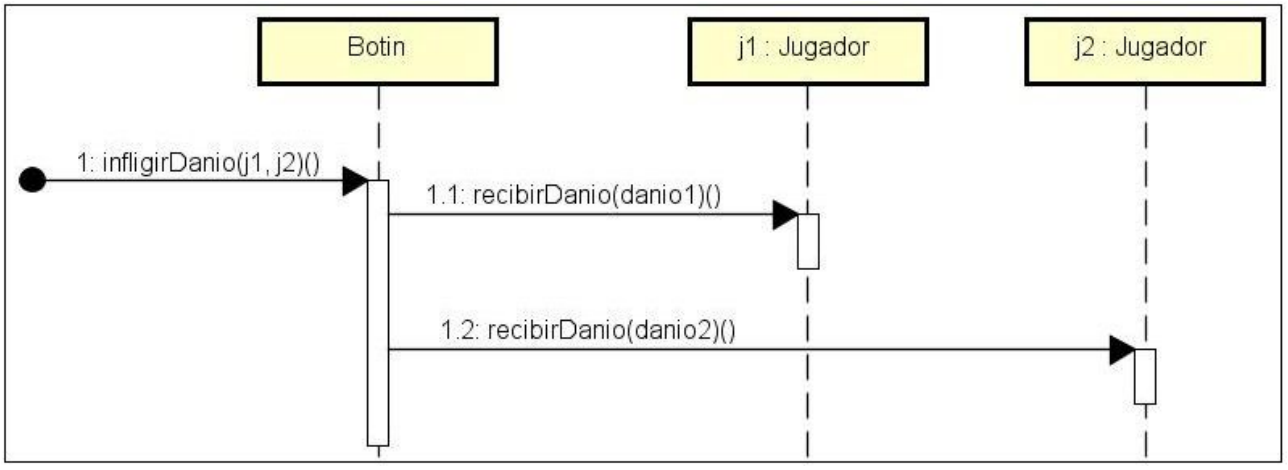
Diagrama 19



El método ejecutar recibe el campo donde se debe destruir la CartaMosntruo cm1.
A este campo se le dice que destruya la carta, quien luego delega su destrucción en la ListaMonstruos de la que está compuesto.

Secuencia 6: un botín inflige los daños guardados en los jugadores recibidos:
j1: un Jugador que realizó un ataque con un CartaMonstruo
j2 : otro Jugador que su CartaMonstruo recibió un ataque de la primer carta.

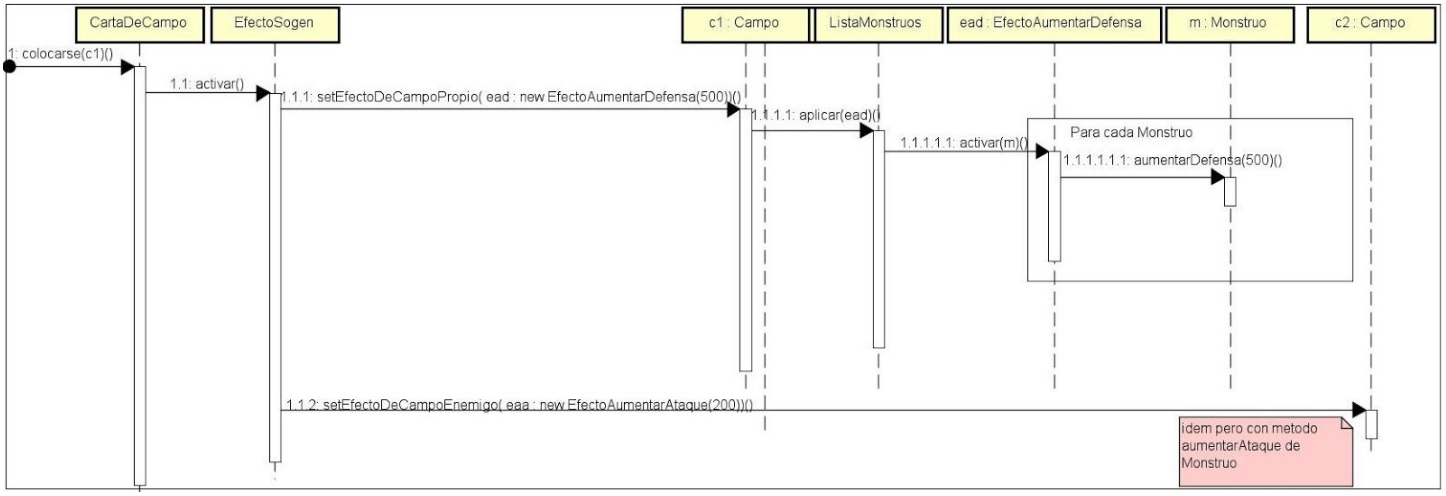
Diagrama 20



El botín llama a los métodos recibirDanio de los jugadores pasandoles el daño que le corresponde a cada uno.

Secuencia 7: Se coloca la carta de campo Sogen en el campo. De esta forma se activa su efecto (aumenta el ataque del campo enemigo (c2) y aumenta la defensa del campo propio (c1)).

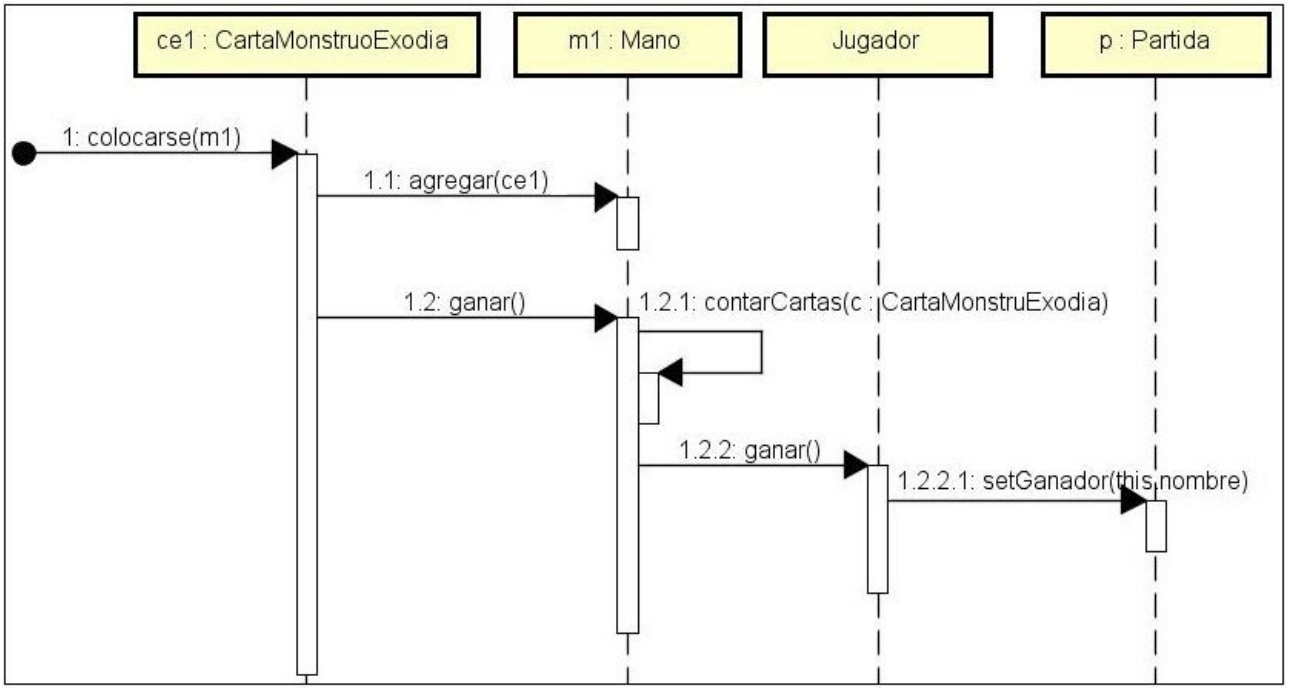
Diagrama 21



La CartaDeCampo se coloca en algún campo, activando su EfectoDeCampo que este caso es el EfectoSogen.
Al activarse el efecto sogen, le envia un mensaje al campo aliado para que establezca como efecto de campo propio a un nuevo EfectoAumentarAtaque
El campo se guarda dicho efecto, y le pide a su ListaMon
El campo pasa este efecto a su ListaMonstruos que lo compone para aplicarlos en los monstruos.
La ListaMonstruo activa dicho efecto en todo los monstruos que contiene.
Luego, el efecto se guarda en el campo como atributo, tal que, se aplique en los nuevos monstruos agregados, o para desaplicar posteriormente.
El procedimiento se repite con el campo enemigo, esta vez estableciendo efecto de campo enemigo al EfectoAumentarDefensa, que realiza en mismo seguimiento que en el caso anterior, solo que al final llama al método aumentarDefensa en cada monstruo. Esta parte no entro en el diagrama pero se señalizo con una anotación

Secuencia 8: Se coloca la última parte de Exodia en la mano del jugador ganando la partida.
ce1 : una CartaMosntruosExodia
m1 : una Mano
p : una Partida

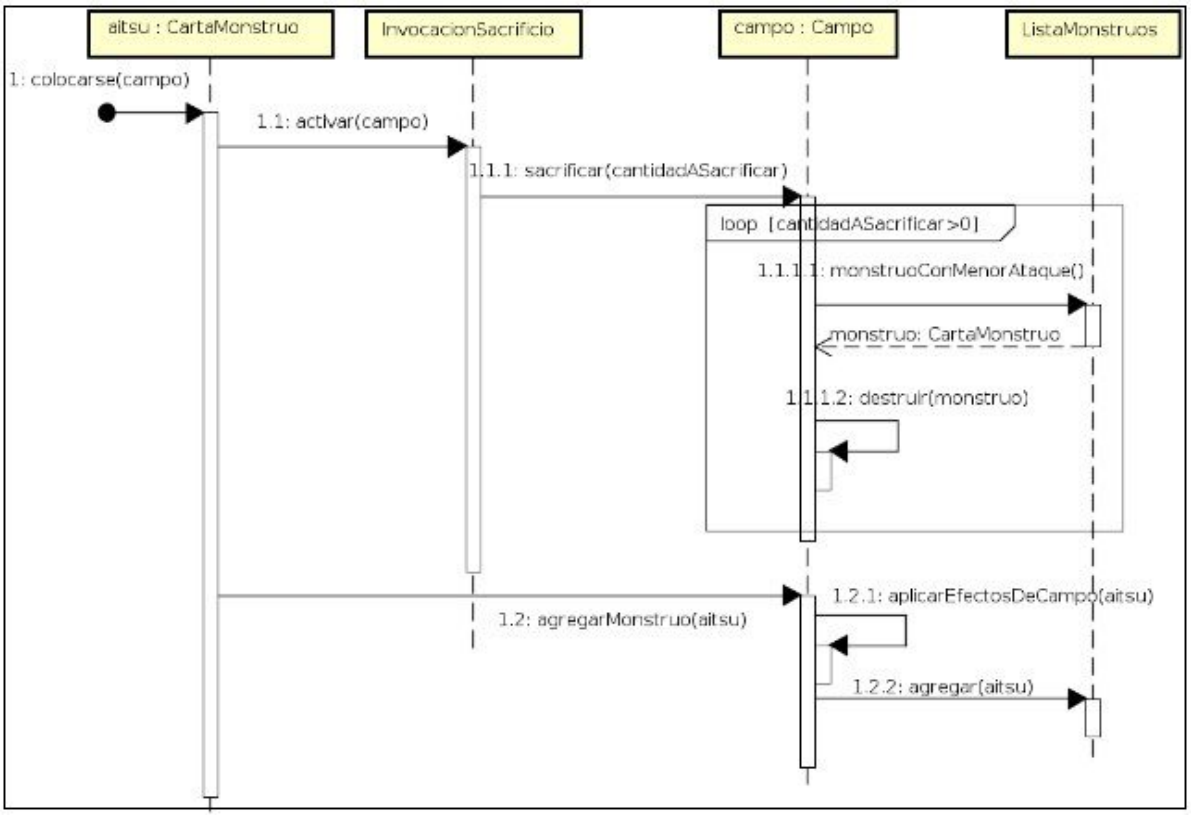
Diagrama 22



La CartaMonstruoExodia se coloca en la clase mano, agregandose en la misma.
Luego se llama al metodo “ganar” de la mano.
La mano cuenta las cartas del tipo CartaMonstruosExodia que hay en ella.
Si hay 5 CartaMontruoExodia (5 partes de Exodia) como expresa este caso, llama al metodo “ganar” de la clase Jugador que la compone.
En este ultimo metodo “ganar” se establece al jugador como ganador de la partida, avisando de ello a la clase Partida que lo compone.

Secuencia 9 : CartaMonstruo se coloca en el campo activando su invocación por estrellas.

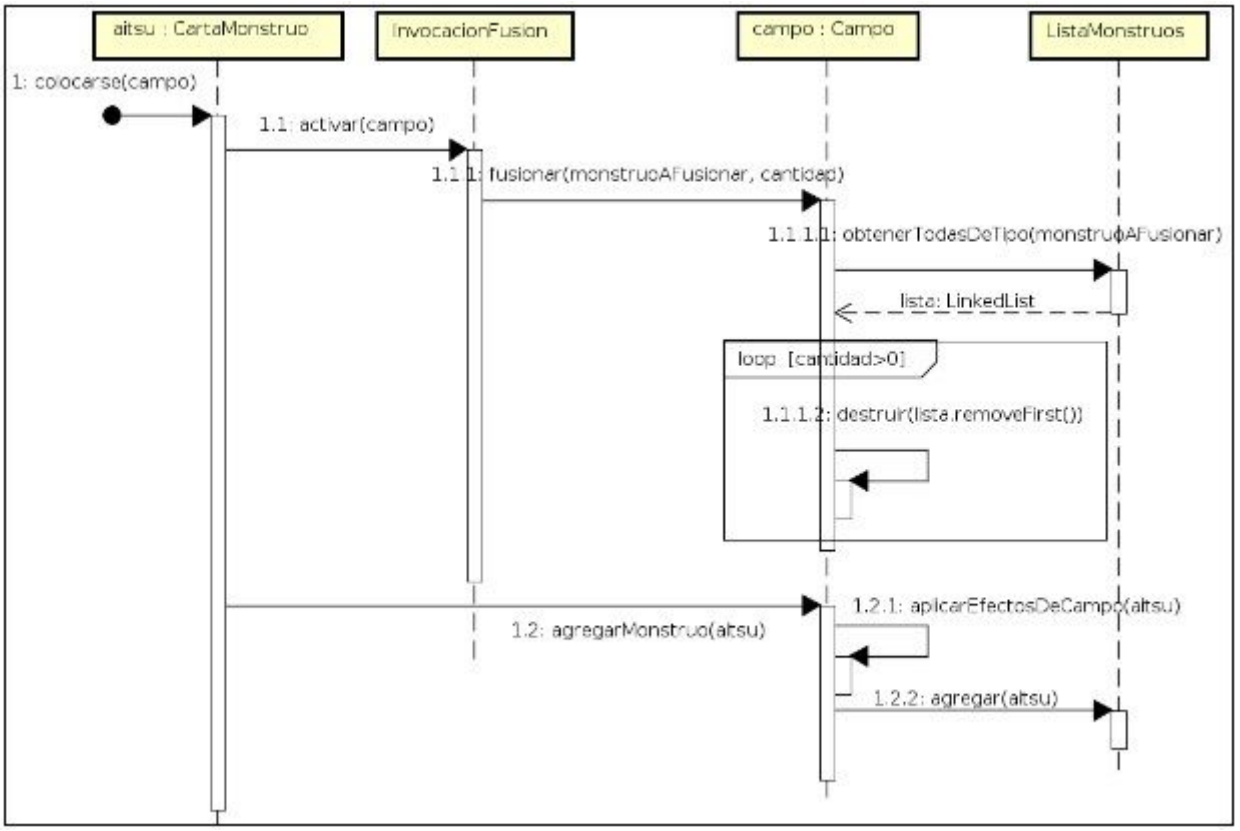
Diagrama 23



La CartaMonstruo recibe el método colocarse en un campo. Se llama al método invocar, que activa la InvocaciónSacrificio por la que está compuesta. Esta última llama al método sacrificar del campo, pasándole por parámetro la cantidad de sacrificios a realizar. El campo obtiene los monstruos a sacrificar pidiéndole el de menor ataque a su ListaMonstruos la cantidad de veces necesaria. Una vez concluida la invocación, se coloca en el campo.

Secuencia 10: CartaMonstruo se coloca en el campo activando su sacrificio por fusión.

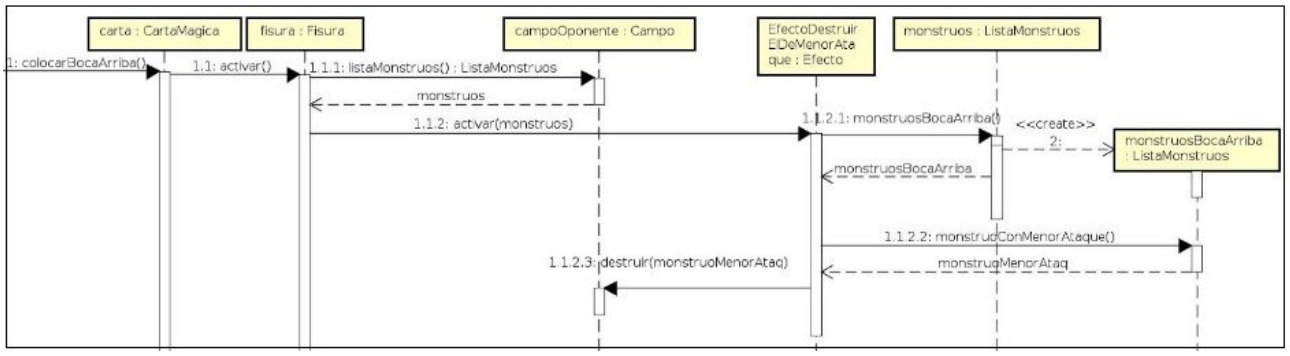
Diagrama 24



La CartaMonstruo recibe el método colocarse en un campo. Se llama al método invocar, que activa la InvocaciónFusion por la que está compuesta. Esta última llama al método fusionar del campo, pasándole por parámetro la cantidad y el tipo de monstruo a buscar. El campo realiza esta búsqueda pidiéndole a su ListaMonstruos una sublista con todos los monstruos del tipo a fusionar que se encuentren en el campo, luego destruye los necesarios a partir de la lista obtenida. Una vez concluida la invocación, se coloca en el campo.

Secuencia 11: Una carta mágica se coloca boca arriba y activa su magia Fisura, destruyendo el monstruo boca arriba con menor ataque del campo oponente.

Diagrama 25:



Luego de haber mirado el diagrama y las clases que lo componen se pensó que tal vez la clase EfectoDestruirElDeMenorAtaque está de más. Toda la responsabilidad de esa clase se podría pasar a la clase Fisura, ahorrandose de alguna manera tanto encapsulamiento sin romper principios de modelo. Sin embargo, no pareció necesaria esa refactorización ya que habiendo probado de esa forma, quedaba ilegible y confuso, asi que se decidió que separar una clase para este comportamiento simplificaba la lectura del código.

Secuencia 12: Una carta mágica se coloca boca arriba y activa su magia AgujeroOscuro, destruyendo todas las cartas monstruo de ambos campos.
La variable *e* escrita en el diagrama es una instancia de la clase EfectoDestruirMonstruo.

Diagrama 26:

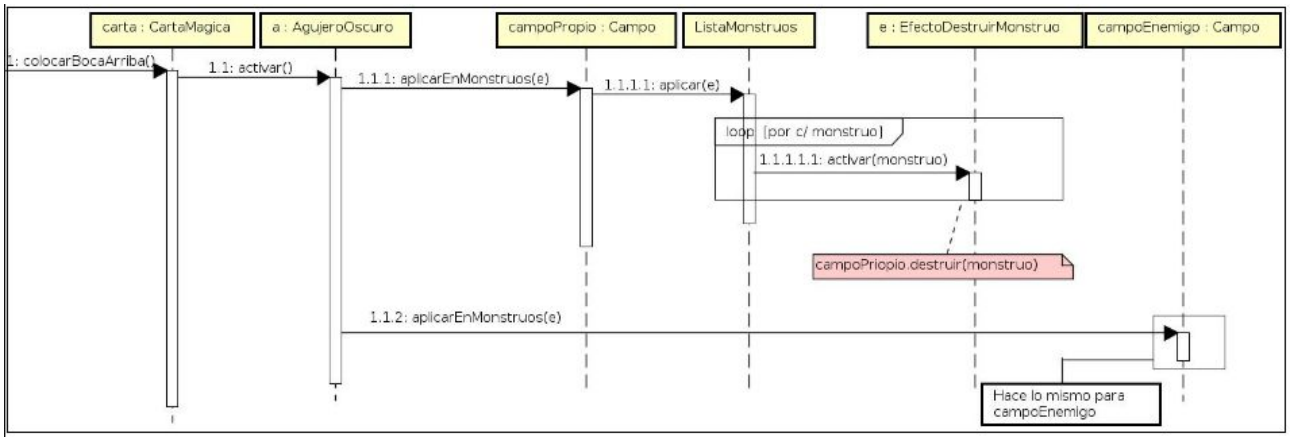
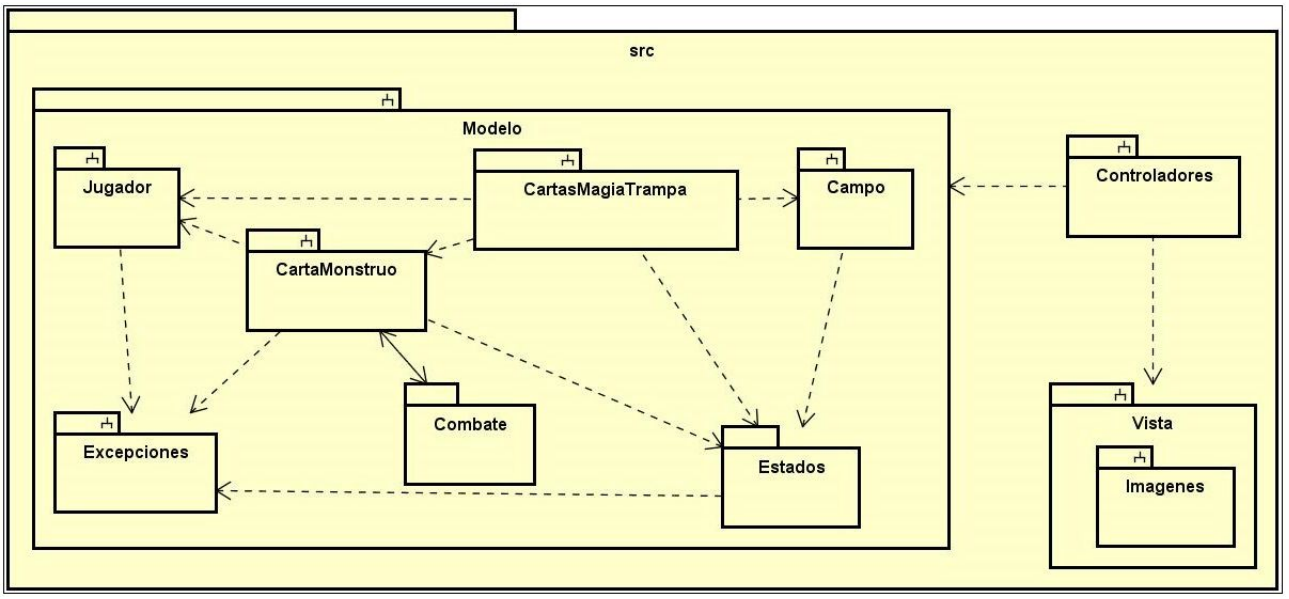


Diagrama de Paquetes

Diagrama 26:



El programa cuenta con dos directorio src y test. En este último se encuentran las pruebas. El directorio src se encuentra dividido en 3 paquetes.

Paquete Modelo:

Este paquete a su vez se encuentra dividido en sub paquetes:

- Sub-Paquete Jugador: contiene a las clases Jugador, Mano, y Mazo.
- Sub-Paquete CartaMonstruo: contiene a la CartaMonstruo y las clases que heredan de él: CartaMonstruoJinzo7, CartaMonstruoComeHombres, CartaMonstruoDragon, CartaMonstruoDragonDefinitivo, CartaMonstruoExodia, y NoCartaMonstruo. También contiene a la clase Monstruo, y la clases que implemetan la interfaz Invocacion, junto a la misma: InvocacionSacrifico, InvocacionFusion.
- Sub-Paquete Estados: Contiene los objetos que implementan las interfaces Boca y Posicion, junto con las mismas: BocaNeutra, BocaArriba, BocaAbajo, BocaAbajoComeHombres, PosDormido, PosAtaque, PosDefensa. Este paquete es conocido tanto por el Sub-Paquete CartasMonstruos, como el Sub-Paquete CartasTrampa, y Sub-Paquete Campo, pues en ellos estan las cartas que los usan.
- Sub-Paquete Excepciones: contiene las excepciones que levanta el programa: MonstruoNoPuedeAtacarError, NoHaySuficientesCartasError, NoCartaMonstruoError, NoHaySuficientesSacrificiosError. Este Sub-Paquete es conocido por los Sub-Paquetes Jugador, CartaMonstruo y Estados Pues son desde ellos donde se levantan.

- Sub-Paquete Combate: contiene la clase Combate que hace pelear a las CartaMonstruo y cobra el resultado Botín que también está en este Sub-Paquete. Por estas razones, este Sub-Paquete conocen a los Sub-Paquete Jugador, CartaMonstruo, Campo.

- Sub-Paquete Campo: Contiene a las clases Campo, CartaDeCampo, CartaDeCampoNula, EfectoDeCampo, EfectoSogen, EfectoWasteland, EfectoDeCampoNulo, EfectoAumentarAtaque, EfectoAumentarDefensa, EfectoNulo, Cementerio, ListaMonstruos. Conoce a los Sub-Paquete CartaMonstruos y CartaMagiaTrampa.

- Sub-Paquete CartasMagiaTrampa: Contiene las clases CartaMagica, CartaTrampa, los objetos que implementan las interfaces Magia, y Trampa, junto con las mismas: Refuerzos, CilindroMagico, TrampaNula, AgujeroOscuro, Fisura, OllaDeLaCodicia. Además, también están las clases EfectoDestruirMonstruo, EfectoDestruirElDeMenorAtaque.. Este SubPaquete conoce a los Sub-Paquetes Jugador, Campo, CartaMonstruos, Combate donde se encuentran objetos necesarios para que crear cada una de las Trampas y Magias.

- Además, fuera de estos Sub-Paquetes, se encuentran sueltas las interfaces Carta, Efecto, y Atacable que son usadas en varios Sub-Paquetes.

Paquete Controladores:
Este Paquete es formado por las clases BotonSacarCarta, BotonVerMano, BotonColocarCarta, BotonCambiarBocaYPosicion, BotonAtacar, BotonActivarMagica, ActualizadorDeRepresentaciones.

Objetos que implementán la interfaz Actualizable: ControladorJugador, ControladorCampoMonstruos, ControladorCampoMagicasTrampa, ControladorCartaDeCampo; y el ControladorCartaComeHombres también. Además, en este paquete también estan las clases Partida, ContructorDeCartas, RandomizadorCartas, Turno, Alerta, la interfaz Reiniciable que usan los botones y el Main.

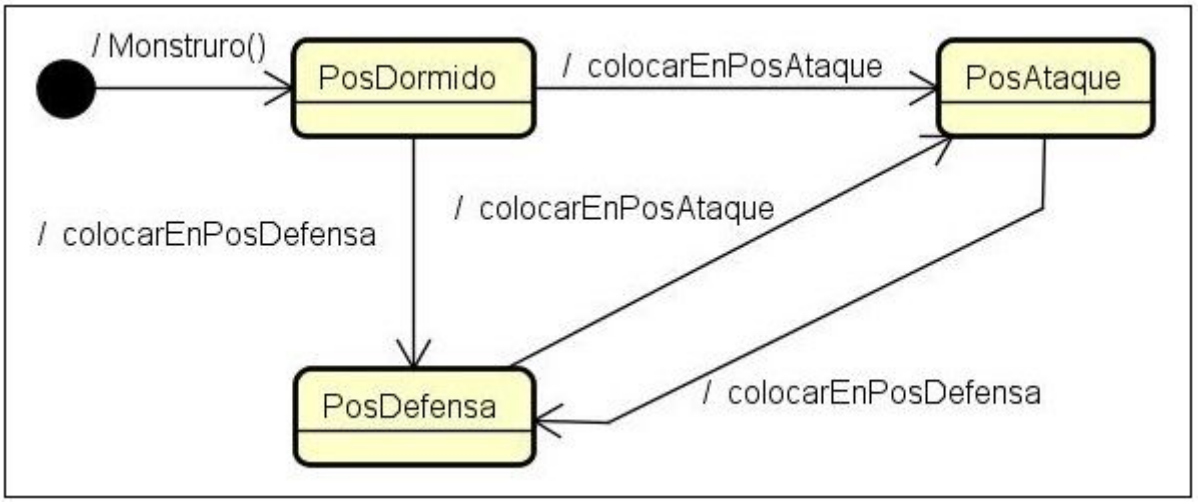
Este paquete contiene los objetos que utilizan al Modelo y manejan la Vista.

Paquete Vista:
Contiene un único Sub-Paquete con todas las imágenes que son utilizadas para la interfaz gráfica.

Diagramas de estado

Diagrama de estado de un Monstruo con respecto a su Posicion

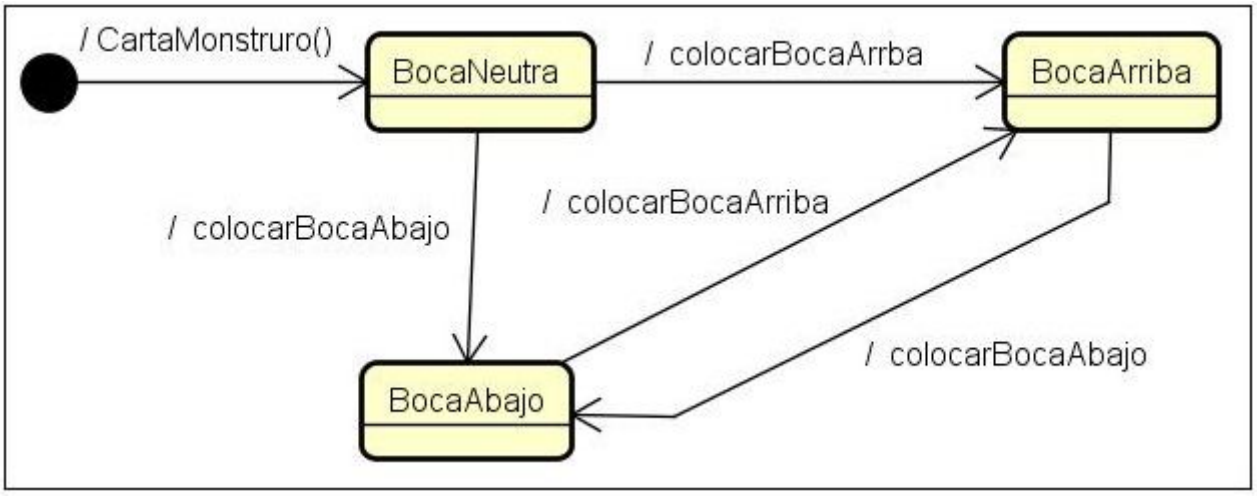
Diagrama 27



Los monstruos se inicializan con una posición dormido (neutra). Desde allí, puede posicionarse en ataque o defensa, y alternándose entre estos dos últimos.

Diagrama de estado de CartaMonstruo con respecto a su Boca

Diagrama 28



Los monstruos se inicializan con una boca neutra. Desde allí, puede colocarse boca arriba o abajo, y alternándose entre estos dos últimos.

Detalles de implementación

Detalles de implementación generales

Para simular la posición de ataque y de defensa de un Monstruo, junto a todo el comportamiento que tienen arraigados, se utilizó el patrón State. Un Monstruo al ser creado tiene un estado Dormido (similar a un estado nulo) y luego puede cambiar de estado a PosAtaque (posición de ataque) o a PosDefensa (posición de defensa). Lo mismo se utilizó para modelar que las cartas están boca abajo o boca arriba . Esto se realizó de esta manera, pues los comportamientos de una carta pueden variar en tiempo de ejecución al cambiar entre estos estados. Además, este patrón fue útil para poder delegar comportamiento a los distintos tipos de Posicion y Boca utilizando polimorfismo sin la necesidad de usar condicionales para cambiar parte del comportamiento de la Carta o del Monstruo.

Al intentar crear clases genéricas como CartaMonstruo, surgió el problema de las cartas más específicas que tienen efectos o propiedades especiales que no se pudieron hacer genéricamente. Para resolver esto, se decidió que éstas cartas particulares sean creadas aparte (de las más genéricas) y que reciban en el constructor las cosas necesarias para su activación del efecto o para su propiedad especial. Esto se puede ver en las clases CartaMonstruoComeHombres, CartaMonstruoJinzo7, entre otras. Sin embargo, estas mismas heredan de CartaMonstruo, por lo que lo único que se tuvo que hacer para modificar su comportamiento específico fue cambiar algún método de la clase madre o agregar otro.
Esta idea también se utilizó para los efectos, magias y trampas: cada clase recibe lo necesario para ser activada. Como EfectoSogen, que necesita de ambos campos de los jugadores para poder usarse, o como Fisura (Magia) que solo necesita el el campo enemigo.
Para el funcionamiento del juego, cada efecto, magia o trampa es colocada en su carta correspondiente (CartaMagica, CartaTrampa o CartaDeCampo).

Para unir todo, en principio se pensó hacer un Singleton que contuviese todo los objetos del modelo. Luego se le pediría al mismo que aplicase ciertas operaciones donde sea requerido. El problema que encontramos con esto fue que por cada nuevo requerimiento se necesitaría implementar un nuevo método, sobrepoblando la Clase.
Finalmente, no fue necesario un objeto gigante que manejara todo, sino que mediante la interfaz gráfica se logró una comunicación entre los objetos del modelo.

Para implementar ciertas reglas del juego, como que un monstruo no puede atacar más de una vez por turno, se aprovechó la interfaz visual para solo permitirle al usuario atacar con quien no haya atacado aun. De la misma manera, se evita que el usuario cometa algunos errores restringiendo las posibilidades de acción a solamente las correctas.

Detalles de clases

Clases Carta y Boca

(Ver diagramas 1: Carta y Boca)

Todas las cartas tienen un estado Boca que, dependiendo del mismo, define distintos comportamientos. Como este comportamiento sólo influye cuando las mismas están colocadas en el campo de batalla, al inicializarse lo hacen con un estado neutro : BocaNeutra que se mantiene mientras estén en el mazo o mano del jugador. Desde allí se pueden colocar boca arriba o boca abajo en el campo.

La mayor parte del comportamiento de las Bocas sirven para la lógica de atacar y para representar a la carta gráficamente. Por ejemplo, una carta monstruo boca abajo no puede atacar y se voltea boca arriba cuando es atacada. De esta manera, las bocas sirven de intermediario para verificar que se es posible el ataque antes de realizarlo. Esta última parte del ataque está delegada en la clase Monstruo que se detallará más adelante.

Las clase tipo Boca responden a métodos atacar, pues solo una carta boca arriba puede atacar. Los metodos imagen, y verNombre son usados para la interfaz gráfica. La carta boca abajo tiene una imagen donde no se revela la carta a la cual representan, y el método verNombre que recibe el nombre de la carta, en el caso de la BocaAbajo devuelve una String “?” que también oculta la identidad de la misma. En el caso de boca arriba , ambos métodos revelan toda la información de la carta.

La clase BocaAbajoComeHombres se utiliza en la CartaMonstruoComeHombres. Sirve para activar el efecto de la CartaComeHombres cuando es atacada boca abajo.

Los metodos estaBocaAbajo y estaBocaArriba se agregaron para funcionalidades específicas de ciertas cartas, que heredan de otras y que necesitan verificar estas condiciones. Se decidieron implementar para evitar agregar métodos que hacen “ruido” en las clases madre.

Las cartas mágicas activan sus magias al colocarse boca arriba, pero pueden colocarse boca abajo y estar a la espera de cambiar de boca.

Clase CartaMagica

(Ver diagrama 2: CartaMagica)

Las cartas mágicas pueden llegar a modificar cualquier aspecto en cualquier parte del juego. Para solucionar este problema se implementa un interfaz Magia. Cada objeto que implementa esta interfaz se inicializa con las partes del juego que necesita para activarse: Campo, Mano, Mazo, Jugador, etc. Cada Magia sabe que hacer al activarse. Esto se implementó de esta manera para evitar tener una clase “bolsa de informacion” que solo guarde las partes del juego que necesitan, sobrecargo de métodos en cada clase para aplicar cada magia en particular o tener que pasar dichas magias de un lado a otro para aplicar su efecto.

Clase CartaDeCampo
(Ver diagrama 3: CartaDeCampo)

Las cartas de campo se implementaron de manera similar a las cartas mágicas. Tienen un método activar que activa un objeto que hereda de la clase abstracta EfectoDeCampo. En este caso, se optó por una clase abstracta de la que heredan un constructor en común, pues todos los efectos de campo siempre aplican sobre los campos de ambos jugadores.

La clase CartaDeCampo es necesaria pues la misma siempre se activa cuando es colocada en el campo. Luego se necesita el tipado para poder identificarla y que el campo sepa cómo administrarla.

Los efectos de campo a implementar (Sogen y Wasteland) son muy parecidos en el sentido que siempre aumentan el ataque o la defensa de los monstruos en el campo aliado o enemigo. Por esta razón, se decidió implementar un EfectoDeCampo abstracto que se posee un constructor común que recibe dos campos . Además, ambos efectos utilizan los efectos EfectoAumentarAtaque/Defensa que se crean en tiempo de ejecuciòn. Estos efecto se explicarán más adelante.

Existe una CartaDeCampoNula que hereda de CartaDeCampo con un EfectoDeCampoNulo que sirve de NullPattern para iniciarlos atributos del objeto Campo. El EfectoDeCampoNulo no realiza ningún comportamiento al activarse pero es necesario por como está definida la clase madre.

(Ver secuencia 7, diagrama 18, para apreciar su aplicación).

Clase CartaTrampa
(Ver diagrama 4: CartaTrampa)
Así como en casos anteriores, las cartas trampa tienen un comportamiento posible muy abierto. Se implementa la interfaz Trampa para que cada una se inicialice con lo que necesita. La diferencia es que las cartas trampa solo se activan cuando una CartaMonstruo ataca a otra, específicamente afectan el resultado de una batalla (Botín) entre dos objetos de la clase Monstruo. De esta manera, el método activar de la CartaTrampa y de Trampa recibe el monstruo atacante, el monstruo atacado y el botín producto de su batalla.

Clase CartaMonstruo

Las cartas monstruos son un poco más complejas. Tienen varias responsabilidades que delegan en otras clases de las que está compuesta.(Ver diagrama 5: CartaMonstruo)

La responsabilidad de atacar y recibir un ataque de otra carta monstruo lo delega en la clase Monstruo. (Ver diagrama 6: Monstruo)

La clase Monstruo puede atacar a otra clase Monstruo devolviendo un objeto Botín que representa el resultado de la batalla. El mismo contiene los datos de la diferencia en daño entre los monstruos y a quien se los debería aplicar (jugador atacante o atacado).

La clases Monstruos están compuestas por una clase del tipo Posición, en la cual delega los distintos comportamientos de ataque y recibir ataque. Todos los monstruos se inicializa con una PosDormido, que corresponde a una posición neutra que no puede ni atacar ni recibir ataque.

La posición de ataque PosAtaque puede tanto atacar como recibir ataque. En el caso de recibir ataque de un Monstruo más fuerte la diferencia de ataque se guarda como daño que debe sufrir su jugador en el Botín. Además si el monstruo tiene tanto o mayor ataque al otro Monstruo, este "mata" al otro monstruo (lo anota como muerto en el Botín).

La posición de defensa PosDefensa no puede atacar pero si recibir ataque. Para este último, la posición de defensa impide que el jugador reciba daño alguno y no puede destruir al monstruo que lo atacó en caso de que su defensa sea mayor al ataque.
Las posiciones le dicen al monstruo con cuanto puntos pelean.

La responsabilidad de invocarse (realizar ciertos sacrificios al colocarse en un campo) es delegado en objetos de tipo Invocación. Aquí se aplica algo similar a las Trampas y Magias, pero en este caso solo se inicializa con lo que define cada Invocación. Todas las invocaciones tienen un método activar que recibe el campo donde se realiza la invocación, al cual se le dice qué hacer. (Ver diagrama 7: Invocacion)

Todos las cartas monstruo al invocarse se inicializan con un objeto de tipo Invocación: InvocacionSacrificio que se inicializa con las estrellas que la carta Monstruo recibe al iniciarse. Dependiendo de la mismas, la invocación se guarda cierta cantidad de CartaMonstruo a sacrificar.

La InvocacionFusion recibe una CartaMonstruo de algún tipo, y la cantidad de cartas de dicha clase a sacrificar. La InvocaciónFusion se usa para la CartaMonstruoDragonDefinitivo que necesita sacrificar 3 CartaMonstruoDragon para invocarse.

De la clase CartaMonstruo heredan otras clases más especiales como son las CartaMonstruoDragon y CartaMonstruoDragonDefinitivo ya mencionadas.

La CartaMonstruoJinzo7 que se caracteriza por poder atacar a un objeto de la clase Jugador directamente. Tiene un método para ello.

La CartaMonstruoExodia solo tiene de especial su tipado, que se usa para identificar las partes de Exodia del resto de las CartaMonstruos. Ademas su colocación implica la posibilidad de ganar el juego que se explicará más adelante en detalles de implementación de la clase Mano.
Luego está la CartaMonstruoComeHombres.

CartaMonstruoComeHombres

La carta monstruo come hombres se caracteriza por la habilidad especial donde si es volteada de boca abajo a boca arriba, puede seleccionar una CartaMonstruo del campo enemigo y destruirla. Específicamente, si esta carta es atacada boca abajo, se decidió tomar como supuesto que siempre se destruye la carta que lo atacó. Para ello se decidió implementar la BocaAbajoComeHombres que al recibir un ataque se voltea activando el efecto de esta carta,

destruyendo a la carta atacada y cortando la ejecución del ataque entre monstruos que se ejecuta después, devolviendo un botín vacío. (Ver secuencia 2, diagrama 15)

Para seleccionar cartas monstruo y destruirlas, tanto cuando es atacado boca abajo como cuando se voltea voluntariamente, se implementó un método “seleccionar”. El método recibe una CartaMonstruo y lo guarda bajo un atributo que se inicializa con una NoCartaMonstruo. Esta última se decidió implementar para evitar hacer uso del null, pero sí tener una CartaMonstruo nula que se puede usar como identificador antes de realizar operaciones como esta. Luego de activar el efecto de la carta come hombres, este atributo se vuelve a setear a NoCartaMonstruo.

Clases de tipo Efecto

(Ver diagrama 8)

Los objetos de tipo Efecto se pueden activar sobre una CartaMonstruo en particular o sobre una ListaMonstruos que se lo aplica a todos sus CartaMonstruos.

Los efectos aumentar ataque / defensa aumentan el ataque / defensa del Monstruo que está en la CartaMonstruo a través de sus métodos. El efecto destruir monstruo se inicializa con el campo donde se encuentran la/s CartaMonstruo a destruir. El efecto “destruir el de menor ataque” destruye el monstruo de menor ataque colocado boca arriba en el campo enemigo, por lo que recibe el campo del enemigo al inicializarse, y busca al monstruo de menor ataque boca arriba a través de los métodos de una ListaMonstruos que le da el campo.

Estos efectos surgieron en parte para reunir código repetido de otras clases de tipo Magia, Trampa, EfectoDeCampo, etc. Por ejemplo, efecto aumentar ataque/defensa se usan por todos los efectos de campo existentes. Otros efectos, se utilizan para delegar responsabilidades.

A diferencia de casos anteriores,estas clases reciben lo que necesitan para funcionar como parámetros a través de sus métodos

Clase Campo

(Ver diagrama 9)

El objeto Campo almacena distintos tipos de cartas, para lo cual cada uno tiene un método específico “colocar” para agregarlas al mismo. El campo tiene referencia a las cartas colocadas en el mismo. En el caso de CartaMagica y CartaTrampa guardan LinkedList separadas. En el caso de las CartaMonstruo, su almacenamiento es delegado en la clase ListaMonstruos que además de métodos para agregar y destruir posee métodos para buscar al monstruo de menor ataque y otro para devolver a los que estén boca arriba.

Las CartaDeCampo, dado que solo hay una por campo, se guardan como atributo específico. Con respecto a las mismas, cada una aplica un efecto permanente sobre las CartaMonstruo del campo por lo que este último posee dos atributos más donde se guardan los efectos (Efecto) que setean los efectos de campo, distinguiendo los que son aplicados por el jugador a quien pertenece dicho campo, de los que aplica el enemigo. Estos se hace de esta manera para facilitar los reemplazos con nuevas cartas de campo colocadas. Inicialmente, estos efectos propios y del enemigo están seteados con el EfectoNulo (Efecto) que funciona de NullPattern

De la misma forma que con colocar cartas, a la hora de destruir a un método distinto para cada tipo de carta, pues cada una se guarda de manera distintos. Sin embargo, a excepción del tipo de

carta que recibe, todos los métodos de colocar y destruir cartas tienen la misma sintaxis. Las cartas destruidas son enviadas a un cementerio común del cual estan compuesto ambos campos.

El campo posee un método para aplicar efectos en todos sus monstruos. Y un método para desactivar efectos temporales (que duran hasta el final del turno, por ejemplo) que puedan tener cada Monstruo en cada CartaMonstruo.

Otra responsabilidad que tiene el Campo es la de activar CartaTrampa. En dicho método recibe los parámetros necesarios para activar una CartaTrampa cualquiera. En caso de haber CartaTrampa colocadas toma siempre la primero que se colocó. De no haber, utiliza una CartaTrampa con TrampaNula que no realiza ninguna acción y devuelve el mismo Botín que recibió.

El campo además posee métodos para devolver las cartas mágicas, trampa , monstruo que se utilizan para la interfaz gráfica.

Clases Mano, Mazo, Jugador y Partida

(Ver diagrama 10)

Tanto el objeto Mano como el objeto Mazo se inicializan con el objeto Jugador al que pertenecen.
El mazo se responsabiliza por tener todos las cartas con que va a jugar un jugador. A medida que es requerido va sacando cartas en cierto orden, y la devuelve. En particular, el método sacar() se hace responsable de que al sacar la ultima carta del mazo, le avisa al jugador por el que está compuesto que perdió.

La mano se responsabiliza por contener las cartas que el jugador tiene la misma, y que luego son colocadas en el campo. Posee un método contarCartas que recibe una carta de algún tipo y cuenta cuántas cartas hay de dicho tipo en la mano. Luego posee un metodo ganar(), verifica a través de sus métodos si estan las 5 partes de Exodia en la mano, y de ser así le aviso al jugador por el que está compuesto que ganó. (Ver secuencia 8, diagrama 22).

La clase Jugador se inicializa con un objeto Partida. El jugador tiene una vida, métodos para mostrarla, y para recibir daño que afecta a dicha vida: recibirDanio() y recibirAtaque(). En el primer caso el daño recibido es numérico y resta vida directamente al jugador, en el segunda el daño lo recibe del ataque de un monstruo. Además de esto, el jugador tiene métodos para perder o ganar según sea requerido, que le avisan de ello a la clase Partida de quien está compuesta.

La clase Partida tiene la responsabilidad de terminar el juego en caso de que un Jugador se establecido como ganador o perder de ello. La Partida puede tener una ventana seteada externamente, que cierra una vez finalizado el juego, abriendo otra con la información del ganador o perdedor.

Trampa Refuerzos

(Ver secuencia 1, diagrama 15)

La carta trampa refuerzos es una trampa que aumenta los puntos de daño (ataque) del monstruo que está siendo atacado. Además, tiene la característica que dicho aumento dura hasta el final del turno. Para solucionar este problema se agrego un metodo agregarAumentoAtaqueTemporal en la clase Monstruos. Dicho método recibe el aumento de ataque a realizar, aumentando el daño del

monstruo y guardando dicho aumento en subtotal de aumentos que tiene el monstruo como atributo. Luego posee un método desactivarTemporales para restar todos esos aumentos de ataque temporales que pudo haber tenido, y que es usado al final de cada turno, cosa que el Monstruo no conoce (la existencia de turnos). No fue necesario complicar el modelo con un conjunto de efectos desactivables pues los únicos aumentos temporales que se requirieron fueron en el ataque de un monstruo y de carácter aditivo.

Para realizar esta operación de desactivarTemporales en forma masiva, se implementó un método del mismo nombre en la clase Campo. Luego se enmascara dicho método bajo un método reiniciar con tal de poder incluirlo en un conjunto de objetos que implementan la interfaz Reiniciable, que siempre se reinician de alguna manera en los cambios de turno.

Botín

Los objeto de tipo Botín se implementaron con el objetivo de separar lo que es la batalla entre dos monstruos, de las consecuencia que las mismas producen. Gracias a esta separación resultó más fácil modelar , diseñar e idear la forma en que dos monstruos combatieran, como se destruirán los mismos tras dicho combate, como se infligirán los daños en los jugadores a quienes pertenecieran, etc. Para lograr todo esto, se pensó al botín como un contenedor del resultado de una batalla, del la cual conoce: cuánto daño le corresponde a cada jugador (atacante y atacado), y que cartas monstruos deben destruirse. (Ver secuencia 3, diagrama 17; secuencia 4, diagrama 18, para apreciar cómo se crea un botín).

Luego, el botín tiene métodos para utilizar dichos datos guardados, recibiendo los objetos donde corresponda aplicarlos. (Ver secuencia 5, diagrama 19; secuencia 6, diagrama 20, para apreciar dichos métodos).

ConstructorDeCartas Y RandomizadorCartas

Para evitar llenar el código principal del programa con muchas líneas para iniciar cada tipo de carta en específico, se implementó un clase ConstructorDeCartas, cuyas métodos corresponden a la construcción de una carta en específico, cada uno. Cada uno de estos métodos recibe las partes del juego: campo, mano, mazo, jugador, que son necesarias construir cada carta.

Luego para llenar los mazo de cada jugador de forma aleatoria, se implementó una clase RandomizadorCartas cuya única responsabilidad es llenar un mazo de forma aleatoria con cartas que construye con el ConstructorDeCartas. Esto podria haber sido un método propio del mazo, pero el código quedaba muy desprolijo e ilegible.

(Ver diagrama 11)

Detalles de la interfaz gráfica

El usuario ver pantalla un conjunto de Vistas que simulan el juego de Yu-Gi-OH. Las vistas están manejadas por un conjunto de controladores que contienen el campo de monstruos, el campo de cartas mágicas y trampas, y la vida del jugador. Todos estos controladores implementan una interfaz Actualizable pues requieren ser actualizados manualmente. Para solucionar este problema, se implementó una clase ActualizadorDeRepresentaciones que puede contener objetos

de tipo actualizables, cuyo unica responsabilidad es decirle a cada uno que se actualice cuando sea requerido.

Luego, existe un conjunto de botones que el usuario puede accionar provocando cambios en las vista. Para que estos cambios se puedan reflejar en esta última, todos los botones están compuesto por un mismo objeto ActualizadorDeRepresentaciones al que le piden actualizar la vista con cada operación que realicen. (Ver diagrama 13)

Los botones son en realidad objetos que implementan la interfaz Eventhandler de Javafx. Los botones que se ven en la pantalla son objetos Button que al accionarse activan estos EventHandler. Cada uno de estos botones conoce una parte específica del juego (del modelo), sobre el cual realiza las operaciones correspondientes a cada botón. Estas acciones producen cambios en los estados del modelo, por lo que utilizan el ActualizadorDeRepresentaciones para reflejar esos cambios en la vista, como antes se habló.

Particularmente el BotonCambiarPosicionYBoca utiliza un ControladorCartaComeHombres para poder aplicar el efecto de la CartaMonstruoComeHombres visualmente. Además, este mismo boton esta compusto por la clase BotonColocarCarta, pues necesita saber que monstruos fueron colocados para en una determinada instancia para no permitirles cambiar su Boca o Posicion en el mismo turno que fueron colocados. (Ver diagrama 12)

Para simular los turnos, se implementó un clase Turno, cuya única responsabilidad es avanzar al siguiente turno, activando y desactivando los botones de cada jugador según a quien le toque jugar. Para ello, la clase turno se inicializa con los objetos Button a habilitar y deshabilitar, correspondientes a cada jugador. Además, también recibe al inicializarse un conjunto de objetos que implementan la interfaz Reiniciable, como son el campo y algunos botones en específico. Estos objetos guardan información que tras cada turno deben volver reiniciar. (Ver diagrama 14)

Excepciones

1. MonstruoNoPuedeAtacarError: Se levanta desde las clases Dormido y PosDefensa (de la interfaz Posición). Son lanzados en el método *atacar()* de las mismas ya que una carta monstruo no debería poder atacar si está en posición de defensa (PosDefensa) o no está en el campo del jugador (Dormido). Esta excepción es atrapada en la parte donde se controla la interfaz gráfica, en la clase BotonAtacar; para que, cuando el usuario elija un monstruo para atacar a otro, sea avisado que no es posible hacerlo (en caso de no estar en posición de ataque).

2. NoHayCartasError: Es lanzada por la clase Mazo en el método *sacar()*, el cual sirve para obtener una carta del mazo. Si el mazo no posee más cartas para dar, lanza la excepción. Fue creada ya que, si el mazo queda vacío y se intenta sacar otra carta, debería avisar de alguna forma que no es posible la acción sin devolver algo que no sea útil o no sea claro (como algún objeto nulo).

3. NoHaySufucienteSacrificiosError: Es lanzada por la clase Campo en los métodos *sacrificar(cantidad: int)* y en *fusionar(m: CartaMonstruo, cantidad: int)* cuando no se

encuentra la cantidad suficiente de cartas a sacrificar o las cartas específicas a fusionar. Esta excepción es atrapada en la parte donde se controla la interfaz gráfica, en la clase BotonColocarCarta cuando se quiere colocar una carta y ésta requiere de ciertos sacrificios o fusiones. Al atrapar ésta excepción, no se coloca la carta en el campo y se le avisa al usuario por medio de una ventana que no es posible colocarla por los motivos mencionados.

4. NoCartaMonstruoError : Es lanzada por la clase NoCartaMonstruo cuando se intenta realizar alguna operación de su madre CartaMonstruo. No debería levantarse ni atraparse en ninguna parte del programa, pues el mismo impide que este error ocurra. Aun así, el error existe como parte del NullPattern que intenta expresar esta clase.