FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2
Al-Go-Oh!

[7507/9502] Algoritmos y Programación III
Curso 1
Primer cuatrimestre de 2018

Alumno	Padrón	E-mail
Souto, Rodrigo Nicolás	97649	rnsoutob@gmail.com
García Villamor, Delfina	101154	delfinagarciav@gmail.com
Schischlo, Franco Daniel	100615	francoschlo@hotmail.com
Silvestri, María Carolina	99175	silconito@hotmail.com

Fecha entrega final: 9/7/2018
Tutor: Fontela, Carlos

Índice

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 el juego de cartas conocido como Yu-Gi-Oh!. Utilizando los conceptos del paradigma orientado a objetos vistos en el curso.

2. Supuestos

El trabajo inicialmente se diseñó pensando en la futura existencia de una interfaz gráfica que integre la totalidad de las clases. De este modo simplificamos el hecho de realizar una refactorización a gran escala. Aún así, se trabajó con las excepciones correspondientes para poder simular lo que la interfaz gráfica no nos hubiera permitido hacer.

El juego a pesar de que tenga 2 jugadores, los turnos van a ser jugados por una sola persona.

En cuanto a las cartas:

- Es obligatorio que en su turno el jugador tome una carta del mazo.
- La carta de campo no puede ser colocada boca abajo.
- Las cartas de trampa no pueden ser colocadas boca arriba.
- Las cartas de monstruo por default se colocan en modo ataque y, luego si se requiere se la puede pasar a modo defensa.
- Si la carta es mágica y se quiere activar su efecto, se debe colocarla boca arriba en el campo, ya que desde la mano no se puede activar.

Muchos de estos supuestos se tuvieron en cuenta para que concordaran con las reglas del juego.

3. Modelo de dominio

Las clases principales de nuestra implementación son las siguientes:

- Juego: Es la clase principal. Contiene a los dos jugadores del juego. Establece el inicio y el final del juego.
- Jugador: Representa al jugador con sus puntos de vida, se encarga de todo lo que este puede realizar y contiene una referencia a su propio tablero.
- Turno: Se encarga de controlar las distintas fases de los turnos de los jugadores.
- TableroJugador: Es el tablero propio de cada jugador. Posee las distintas zonas del tablero: el cementerio, el mazo y el campo. Se encarga de la interacción entre las cartas y el campo durante el duelo.
- Campo: Zona del tablero donde se posicionan las distintas cartas del campo estas son: Mágicas, Trampa y Monstruos. Se encarga colocarlas en el lugar correcto y quitarlas en caso de ser necesario.

Además se implementó la clase Carta con sus :

- Carta: Es una clase abstracta reune los atributos pertenecientes a cada carta (nombre, efecto, estado, en turno y jugador). Se encarga de todo lo que puede realizar una carta en general. Es la clase madre de las siguientes: Monstruo y NoMonstruo, este último incluye las cartas mágicas y de trampa.

- Monstruo y NoMonstruo: Estas clases se encargan de las tareas específicas de cada tipo de carta. Como se mencionó anteriormente, NoMonstruo incluye a las cartas mágicas y de trampa. Ambas contemplan los modos (estado) en los que se puede encontrar la carta.

Dentro de carta encontramos interfaces y clases de estado que nos permiten saber la disposición de las cartas:

- EstadoCarta: Es una interfaz que se encarga de informar si la carta se encuentra en juego o no.
- EnJuego y FueraDeJuego: una carta está en juego cuando se encuentra en el campo, lo que implica que la carta puede ser utilizada. En cambio, cuando la carta esta fuera de juego esta se encuentra en el cemeterio, en el mazo o en la mano y queda inhabilitada para su uso. Estas clases abstractas que implementan EstadoCarta se encargan de informar y verificar que las cartas que interactúan estén en juego.
- BocaArriba y BocaAbajo: Implentan EnJuego y son las encargadas de la lógica de las cartas en ese estado.
- ModoMonstuo: clase abstracta encagargada de derivar a sus hijas el modo de resolver la logica del ataque entre monstruos estas son: ModoAtaque y ModoDefensa.

Además las cartas possen efectos por lo tanto las clases que manejan esto son:

- EfectoCarta: Es una interfaz, corresponde al efecto de una carta.
- EfectoNulo: le corresponde a una carta sin efecto.

Los efectos de las distintas cartas implementan esta interfaz en combinación con la implementación de cada efecto correspondiente a cada carta.

Las cartas, a su vez poseen una interfaz que indica el estado en turno para saber, a que jugador le permite seleccionarla o si esa carta puede atacar. Esta interfaz se puede apreciar en el diagrama de estado de la sección 8.

A medida que avanzo el trabajo se notó que el jugador posee distintos estados durante su turno, para ello de desarrollo lo siguiente:

- EstadoJugador: interfaz que indica el estado del jugador.
- PreInvocacion: si el jugador se encuentra en esta fase solo podrá colocar un monstruo en el campo y robar una carta.
- PostInvoación: si el jugador se encuentra en esta fase solo realizará ataques, o activaciones de cartas.
- TurnoDelOponente: el jugador no esta jugando, pero su oponente (otro jugador) sí, por lo tanto tiene los mismos estados que nuestro primer jugador.

4. Diagramas de clase

Los siguientes diagramas son a baja escala y muy generales, en específico se tratan del juego a nivel general y de la clase carta.

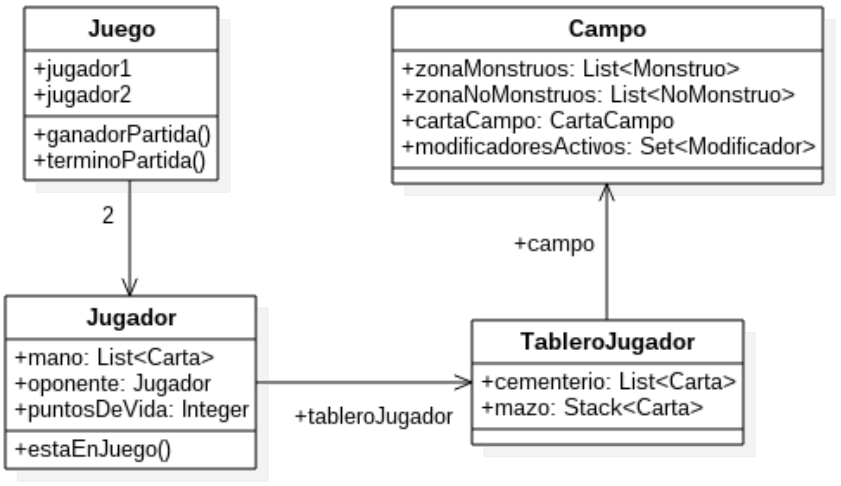

Figura 1: Diagrama de clases a nivel del juego.

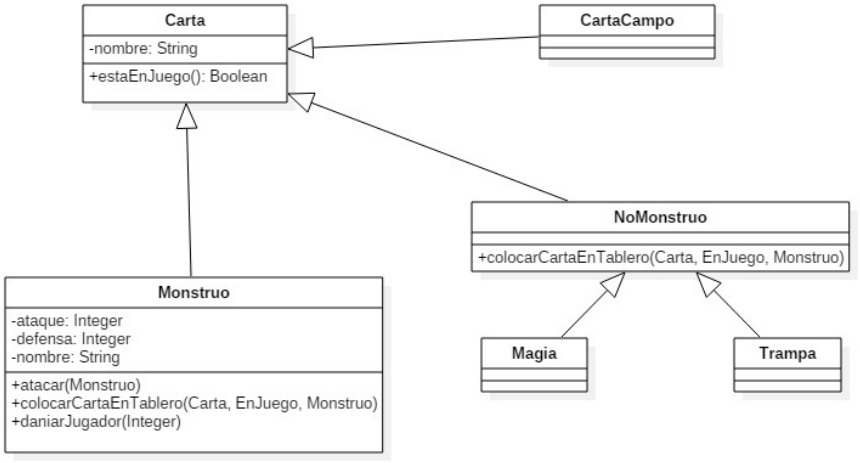

Figura 2: Diagrama de clase que muestra los distintos tipos de carta.

Para ver el diagrama completo se recomienda ver el anexo adjunto.

5. Detalles de implementación

5.1. Utilización de patrones

- En el sistema de turnos implementamos el patrón **singleton** a Turno para que sea conocido por todas las clases.
- Para que las cartas no puedan atacar en el turno del oponente, y que no puedan volver a atacar luego de haberlo hecho una vez, se implementó el patrón **observer**, siendo Turno el Observable, junto con el patrón state en el que van cambiando de estado según hayan atacado o haya terminado el turno.

- El Juego que contiene a los jugadores también es un **singleton** porque es único.
- El jugador conoce las cartas de su carta con la interfaz Carta, pero al colocarse en el campo se ubican en zonas distintos según sea el tipo de carta (Monstruo, Mágica, etc.). Para solucionar esto usamos el patrón **double dispatch** así el campo sabe donde colocar cada carta.
- El problema de que las cartas campo modifican las estádísticas de los monstruos en campo se solucionó agregando modificadores. A las cartas se le agregan o quitan modificadores dependiendo de si el efecto se mantiene en campo o se va. Esto sirvió también para que en la vista se vean los valores de ataque y defensa modificados y generados dinámicamente.
- La carta refuerzos hace uso de esos Modificadores, agregando 500 puntos de ataque al atacado y luego con el **patrón observer** al terminar el turno remueve el efecto.
- Según sea el tipo de carta, la forma en la que se coloque (BocaAbajo o BocaArriba) puede variar la funcionalidad al colocarse (por ejemplo si las cartas mágicas se colocan boca arriba se activan directamente sin colocarse en el campo). Para esto también usamos double dispatch para que una carta se coloque como se debe según la carta y el estado.
- Se utiliza el patrón **strategy** para delegar el sistema de sacrificios del monstruo, teniendo un atributo que se llama nivel. Este atributo es generado por la cantidad de estrellas del monstruo, usando el patrón factory para generar la instancia de Nivel correcta.
- Para controlar que un jugador pueda atacar sólo en su turno, y que no pueda colocar más cartas después de invocar un monstruo, se implementó el **patrón state junto con observer**.
- Se usa un **template** method en ModoMonstruo en el método recibirAtaque para no repetir código y sólo delegar lo necesario en la subclases ModoDeAtaque y ModoDeDefensa.
- Para modelar el estado de la carta a través del juego (en mazo, en mano, en juego, etc.) se usó el patrón **state**.

6. Excepciones

- CampoNoPermiteColocarCartaExcepcion: Si el campo esta lleno el campo no permite colocar más cartas.
- CartaInhabilitadaParaActivarseExcepcion: si se desea activar un efecto de una carta en un lugar que no es correspondiente entonces esta excepción es lanzada.
- CartaNoInicializadaEnDueloExcepcion: las cartas deben estar inicializadas de algún modo para poder luchar.
- CartasInsuficientesExcepcion: usualmente si el mazo queda en 0, el juego termina, aún así se implemento esta excepción para moldear el juego.
- EstrellasInvalidasExcepcion: los mosntruos no pueden tener 0 estrellas.
- FaltaObjetivoAAtacarExcepcion: si se decide atacar y no se selecciona a un monstruo en campo enemigo, esta excepción es lanzada.
- JugadorInhabilitadoParaColocarCartasExcepcion: esta excepción se lanza cuando el jugador ya pasó la fase de preinvocación, y, por lo tanto no puede agregar más cartas al campo.
- MonstruoInhabilitadoParaAtacarExcepcion: si el monstruo ya atacó, por reglamento no puede volver a atacar.

- NoSePuedeAtacarEnElPrimerTurnoExcepcion: las reglas indican que en el primer turno un jugador no puede atacar al otro por lo tanto, se lanza esta excepción.
- SacrificiosIncorrectosExcepcion: dado que en el juegos hay monstruos que requieren sacrificios, si no se selecciona la cantidad correcta, se lanza esta excepción.

7. Diagramas de secuencia

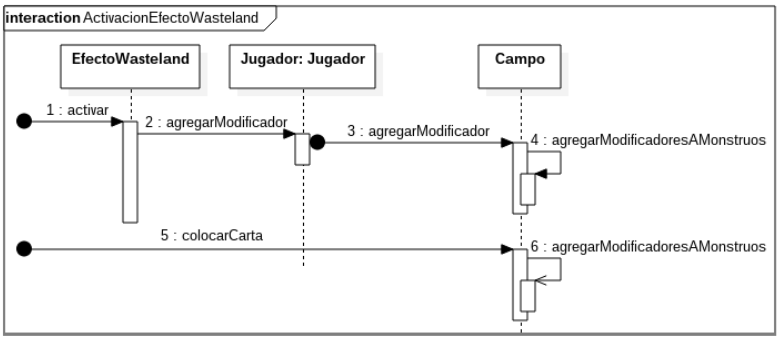

Figura 3: Diagrama que represanta la activación del efecto Wasteland.

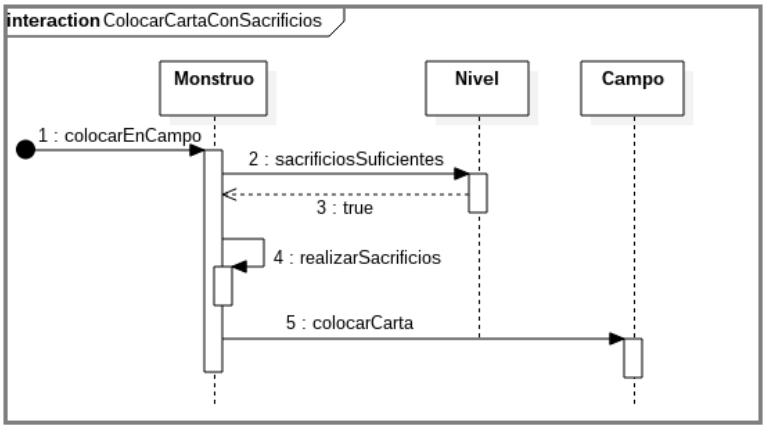

Figura 4: Diagrama de secuencia: colocación de monstruos que requieren sacrificios.

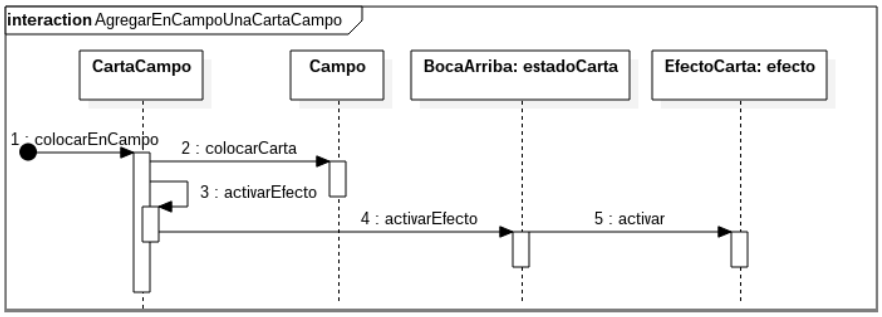

Figura 5: Diagrama de secuencia: colocación de cartas en campo.

8. Diagramas de estado

Figura 6: Diagrama de estado: distintos estados de un jugador dentro de su turno.

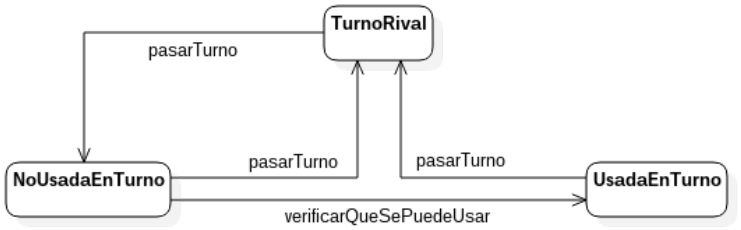

Figura 7: Diagrama de estado: distintos estados que adopta una carta durante el juego.