FACULTAD DE INGENIERIA
Universidad de Buenos Aires

FIUBA - 75.07

Algoritmos y programación III

Trabajo práctico 2: Al-Go-Oh!

Curso 2

1er cuatrimestre, 2018

Alumno	Padrón	Mail
Demian Poggio	93665	demianpoggio@gmail.com
Vallejo Juan Pablo	101133	vallejojuanp@gmail.com
Matias Ramundo	100327	matiramundo@gmail.com
Claros Castro Elvis	99879	elvisclaros@icloud.com

Índice

1. Introducción

El presente informe reune la documentación del trabajo práctico número 2 en Java "Al-Go-Oh!".

2. Supuestos

DragonDeOjosAzulesDefinitivo Se lo tuvo que tratar como un monstruo común ya que se hacía muy complicado y engorroso tratarlo como en el juego real. El dragon definitivo es una carta como cualquier otra y tenes que tenerla en la mano para invocarla.

Jinzo7 Ataca siempre a los puntos de vida

InsectoComeHombres Suponemos que el efecto del insecto come hombres se activa luego de haber recibido el ataque, por lo cual, este puede destruirse si lo ataca un monstruo con mayor ataque que su defensa.

CartasCampo Suponemos que el efecto de las cartas campo se activará solamente durante la fase de ataque y sera revertido al finalizar esta misma.

3. Modelo de dominio

Inicialmente comenzamos realizando un diagrama de clases con el que pudiéramos observar como se desarrollaría el juego en formas generales. Dicho diagrama contenía las **clases Jugador, Zona y Carta**,

que permitía observar como un jugador se relacionaba con sus Zonas, con las que interactuaría durante el transcurso de la Partida y como una Zona se relacionaría con las Cartas que contendría. Luego para modelar las cartas decidimos crear las **clases CartaCampo, CartaMagica, CartaTrampa y Monstruo** que heredan de la clase Carta para manejar los distintos comportamientos que puede tener cada carta en particular. Luego, para resolver la problemática de los estados de las cartas, decidimos crear las clases EstadoSinEstado, EstadoBocaAbajo, EstadoBocaArriba, EstadoAtaque, EstadoDefensaBocaAbajo y EstadoDefensaBocaArriba que implementan la **interfaz Estado**. Las ultimas tres clases antes mencionadas también implementan la **interfaz EstadoMonstruo**, permitiendo así manejar el ataque de los Monstruos.

Ya teniendo el modelo avanzado, decidimos incorporar las **clases Mazo y Cementerio**, que permitieron linkear las Cartas con los Jugadores, agregando al juego el flujo necesario para desarrollar un turno completo del juego, ya que permite obtener y destruir cartas. Junto con estas clases se incorporaron las **clases FaseInicial, FasePreparacion, FaseAtaque y FaseFinal**, que implementan la interfaz **Fase** y permiten indicar las acciones permitidas en el momento actual del turno de un Jugador.

Posteriormente decidimos incorporar la **clase Juego**, la cual contiene 2 Jugadores y coordina los turnos de cada uno. Interactuando con esta clase se puede desarrollar una Partida, ya que la misma además de manejar los turnos se encarga de interactuar con las distintas Fases que

puede tener el turno de un Jugador y puede determinar si existe algún ganador.

Con el modelo completo comenzamos a desarrollar la interfaz gráfica. Para la misma utilizamos la librería **JavaFx**. Una de las primeras cosas a resolver fue la comunicación entre el modelo y la vista. Para este ítem decidimos utilizar la **clase Controlador**, la cual se encarga de recibir todas las peticiones que realiza el usuario final mediante la interfaz grafica y relacionarlo con el modelo en sí de manera adecuada. Luego para la vista decidimos ir desde lo mas pequeño a lo mas grande. Desarrollamos en primer lugar los Botones necesarios para la interfaz, seguido por los Paneles que contendrían dichos botones.

Finalmente, creamos una **clase PantallaBatalla**, que se encarga de ubicar los paneles y mostrar la interfaz completa del juego, y una clase Vista que se encarga de lanzar el Juego y cargar la pantalla anteriormente mencionada.

3.1. Descripción de las clases:

Paquete Modelo:

- Juego: Va a tener la responsabilidad de ser el nexo entre los dos Jugadores. Y decidir cuál de los jugadores es el ganador.
- Jugador: Tiene la mayor responsabilidad de delegar a otras clases, y contiene las zonas monstruo y mágica, un mazo, una mano, una

carta campo.y un cementerio.

- Carta: Es la clase padre de las clases; CartaMagica, CartaTrampa, Monstruos y CartaCampo.
- Monstruo: Es heredera de la clase Carta, y es la clase ancestra de todas las cartas monstruos del juego.
- CartaMagica: Es heredera de la clase Carta, y es la clase ancestra de todas las cartas mágicas (Fisura, Olla de la Codicia,etc).
- CartaTrampa: Es heredera de la clase Carta, y es la clase ancestra de todas las cartas trampa (Cilindro Mágico, etc).
- CartaCampo:Es heredera de la clase Carta, y es la clase ancestra de todas las cartas campo (Wasteland, Sogen).
- Zona: Es la clase padre de las clases; ZonaHechizos, ZonaMonstruo.
- ZonaHechizos: Va a contener 5 cartas de la clase CartaMagica y CartaTrampa.
- ZonaMonstruo: Va a contener 5 cartas de la clase Monstruo.
- Cementerio: Va a contener a todas las cartas mágicas, trampas que se activen y los monstruos que son destruidos o sacrificados.
- Mano: Va a contener todas las cartas del Jugador que van a poder ser jugadas en su turno.
- Mazo: Contiene 40 cartas las cuales van a estar mezcladas, tiene la responsabilidad de proveer al jugador cartas para la mano.

- EstadoMonstruo: Es una interfaz. Cada estado tiene la responsabilidad de cambiar el comportamiento de las cartas monstruo. Por ejemplo, EstadoAtaque va a tener la responsabilidad de habilitar el ataque hacia otro monstruo.
- Estado: Es una interfaz. Tiene dos implementaciones EstadoBocaArriba y EstadoBocaAbajo, esto sirve para todas las cartas, tanto mágicas,trampa,campo y monstruo. Esta clase sirve más que nada para la interfaz gráfica y para la activación de los efectos.

Paquete Vista:

- PantallaBatalla : Su función más importante es el de actualizar la vista del usuario.
- PanelMano : Panel donde se almacenaran los botones para las cartas de la mano de ambos jugadores cuando sean el "jugador activo"
- PanelManoEscondida : Panel que sirve para representar la mano del "jugador oponente"
- PanelMonstruos : Panel que se utilizara para representar la zona de monstruo y almacenar los botones de monstruos correspondientes. Cada jugador tendrá su panel, y cada panel se encargara de buscar las cartas dentro de la ZonaMonstruo.
- PanelMonstruosObjetivo : Panel que se abrirá para seleccionar un monstruo al momento de realizar un ataque. Representara la ZonaMonstruo del jugador oponente.

- PanelHechizos : Panel que se utilizar para representar la zona de hechizos y almacenar los botones de cartas mágicas o de trampa correspondientes. Cada jugador tendrá su panel, y cada panel se encargar de buscar las cartas dentro de la ZonaHechizos.
- PanelCementerio : Panel que se abrira al presionar dentro del boton de cementerio. Sera el encargado de mostrar todas las cartas que se encuentran en el cementerio.
- PanelSacrificios : Panel que se abrirá al momento de invocar un monstruo que requiera sacrificios. Representara una copia de la ZonaMonstruo del jugador activo.
- BotonCarta : Clase abstracta que sirve para representar un botón que contendrá una carta. Heredaran de ella, los demás Botones.
- BotonCartaCampo : Botón que almacenara una carta de campo. Activara los siguientes efectos según la fase o lugar donde se encuentre:
- Mano FasePreparacion : Permitirá activar la carta de campo desde la mano
- BotonCartaMagica : Boton que almacenara una carta mágica. Activara los siguientes efectos según la fase o lugar donde se encuentre:
    - Mano FasePreparacion : Permitirá activar la carta magica desde la mano o colocarla boca abajo en el campo de hechizos

  - Campo FasePreparacion : Permitirá activar la carta desde el campo
  - Campo FaseFinal : Mismo efecto que el anterior.

- BotonCartaTrampa : Botón que almacenara una carta trampa. Activara los siguientes efectos según la fase o lugar donde se encuentre:
  - Mano FasePreparacion : Permitirá colocar desde la mano la carta en el campo de hechizos.

- BotonMonstruo : Boton que almacenara una carta de monstruo. Activara los siguientes efectos segun la fase o lugar donde se encuentre:
  - Mano FasePreparacion : Permitirá invocar el monstruo en posición de ataque o colocar el monstruo boca abajo en posición de defensa. En el caso de que el monstruo necesite sacrificios para ser invocados, sera el controlador el que maneje la excepción y abra la ventana de sacrificios correspondiente.
  - Campo FasePreparacion : Permitirá cambiar el estado del monstruo a ataque o a defensa pero siempre boca arriba.
  - Campo FaseAtaque : Permitirá abrir una ventana para seleccionar el monstruo al cual se quiere atacar siempre y cuando el atacante se encuentre en posición de ataque.
  - Objetivo FaseAtaque : Permitirá seleccionar al monstruo como un objetivo para un ataque. Ademas sera el efecto que cierre la ventana abierta previamente.

  - Sacrificios : Permitirá seleccionar al monstruo como un objetivo de sacrificio para invocar otro monstruo.

- BotonSinCarta : boton utilizado para representar un boton que no contiene una carta. No tiene ningun efecto.
- BotonPasarFase : boton que llamara al controlador para avanzar la fase
- BotonPuntosDeVida : boton que se abrira cuando no haya monstruos en el campo adversario. Permitira al monstruo atacante, atacar a los puntos de vida del oponente.
- BotonReiniciarJuego : boton que llamara al controlador para reiniciar el juego.
- BotonTerminarTurno : boton que llamara al controlador para terminar el turno del jugador activo.
- ParametrosBoton : Clase que guardara variables de configuracion para el tamaño de los botones.

4. Excepciones

- ZonaVaciaException: Se utiliza para indicar que la Zona en cuestión no posee ninguna carta
- MazoVacioException: Se utiliza para indicar que el objeto mazo no posee ninguna carta.

- RequiereSacrificioException: Se utiliza para indicar que se han querido sacrificar (o no) monstruos de manera incorrecta.
- MonstruoDeFusionException: Se utiliza para indicar que los monstruos que se van a sacrificar no son los necesarios ya que el monstruo a invocar requiere de una fusión.
- InvocacionNoPosibleException: Se utiliza para indicar que el Jugador en cuestión no puede invocar a un monstruo, ya sea porque ya ha superado la cantidad de invocaciones por turno o porque no cuenta con los monstruos necesarios a sacrificar para la invocación.
- FaseIncorrectaException: Se utiliza para indicar que la acción que quiere realizar el Jugador no puede ser realizada en la Fase actual de la partida.
- DetenerAtaqueException: Se utiliza para indicar que se debe detener un ataque en curso.

5. Detalles de implementación

En un principio, encontramos inconvenientes a la hora de plantear cómo manejar los Estados de las cartas. Para resolver esta problemática, se utilizaron enums para implementar los estados de los monstruos ya que éstas responden a un u otro comportamiento dependiendo de la posición que tenían (ataque, en posición de ataque boca arriba o en posición de defensa). Luego de revisar lo anterior con nuestros correctores se optó por utilizar la interfaz Estado, con esto podemos quitar los enum

anteriormente utilizador y permitió aplicar un patrón de diseño. Para resolver la problemática de cómo conectar al Jugador con sus zonas decidimos crear una clase Tablero para cumplir esta responsabilidad. Al revisar la clase anteriormente creada, notamos, en conjunto con nuestros correctores, que no tenía ningún sentido incluir dicha clase, ya que lo único que hacía era llamar métodos de otras clases. Por este motivo, decidimos quitar dicha clase del modelo y que todos los métodos que manejaba Tablero pasaran a ser llamados por Jugador. Con esto la clase Jugador pasó a contener demasiadas responsabilidades (Jugador conoce a casi todas las demás clases y termina siendo una clase central). Pero dado que en este caso es una clase clave del sistema, no nos pareció mal que conozca a todas las zonas.

Para las Cartas decidimos utilizar el patrón **Composite**. El mismo nos permitió poder modelar sin ningún inconveniente la Mano y el Mazo de un Jugador, ya que, al *Mazo* y a la *Mano* no le interesaba saber la categoría de las cartas que tenía, sino solamente estar seguros de que las mismas cartas. Para ello, creamos la clase abstracta Carta la cual tiene los métodos necesarios para ser obtenida del *Mazo* y ser colocada desde la Mano en su Zona correspondiente. Para los estados de las cartas (*BocaArriba*, *BocaAbajo*, etc.), se utilizó el patrón de comportamiento **State**, ya que las cartas van cambiando su estado de carta y su comportamiento a lo largo del juego y no se puede estar redefiniendo la clase carta para cambiar su estado y su comportamiento. Para terminar con la lógica, decidimos crear las clases *Juego* y *Jugador*. Cada Jugador

posee una Mano, un Mazo, una *ZonaMonstruos*, una *ZonaHechizos*, un *Cementerio* y una *CartaCampo*. A su vez, también contiene un atributo que indica sus puntos de vida. Por el lado del *Juego*, para el mismo decidimos utilizar el patrón **Singleton**, ya que el mismo necesita ser accesible globalmente. Luego procedimos a realizar la interfaz gráfica. Este apartado fue realmente novedoso para el grupo, ya que ninguno había tenido oportunidad de realizar anteriormente ninguna interfaz gráfica. Para comenzar la interfaz, decidimos investigar sobre JavaFx para no encontrarnos con ninguna sorpresa cuando ya tuviéramos la mitad de la vista implementada. Desde un inicio, teníamos en claro que debíamos utilizar el patrón **MVC** para organizar la aplicación. Para poder llevar a cabo este patrón, creamos la clase Controlador, que tiene como objetivo comunicar la vista con la lógica. Dado a nuestra poca experiencia creando interfaces, todo nuestro juego se basó en una única pantalla (*PantallaBatalla*) que se va actualizando al realizarse algún evento.

6. Diagramas

- Diagramas de Clase: Hay 7 diagramas de clase. Uno corresponde al modelo del Juego, uno al de Carta, uno al de los Estados, uno al de las Fases, uno al Jugador, Uno a los Monstruos y también un diagrama corresponde al modelo de la Vista.
- Diagramas de Estado: Hay 1 diagrama de estados que corresponde al estado de las fases durante el turno de un jugador

- Diagramas de Secuencia: Hay 16 diagramas de secuencia. Dos corresponden a la carta AbismoReluciente, cuatro corresponden a la activación de cartas trampa, dos a la activación de cartas mágicas, dos al flujo de las fases, tres al comportamiento del AgresorOscuro, uno al InsectoComeHombres y uno a la invocación del DragonDefinitivo.
- Diagramas de Paquetes: Hay 1 diagrama de paquetes que muestra la relación de los paquetes dentro del modelo.