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
Matías Ramundo	100327	matiramundo@gmail.com
Claros Castro Elvis	99879	elvisclaros@icloud.com

1.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 *EstadoAtaque,* 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.

2.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.

3. 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 mismo es una carta como cualquier otra y debe estar 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.
- Las cartas campo se activan en la fase de ataque del turno de cada Jugador

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.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.