Universidad de los Andes > INGENIERÍA

Proyecto: Hearthstone
Programación orientada a objetos

Profesor: Ignacio Parada
Ayudante: Diego Beckdorf
Francisco Moreira
Gianluca Fenzo
Gianluca Troncossi

Entrega 4:
Modelo relacional - Hearthstone
CLASES:

MainWindow:
Para la última entrega, nuestro main se ubica en la clase MainWindow, la cual aloja la interfaz de usuario principal, esta clase muestra el tablero y es donde los jugadores llevan a cabo el juego.
Atributos:

- Int resp#: Existen 8 integers con nombre resp, estos atributos sirven para generar el cambio de cartas en la mano inicial para los jugadores, siendo que el primer jugador puede cambiar tres cartas y el segundo jugador puede cambiar 4.
- Tablerini: El tablero del juego.
- TAClick1 (Tablero aliado click 1): Da a conocer quien ataca en el tablero aliado (con aliado nos referimos al jugador que esté en su turno).
- TEClick1(Tablero enemigo click 1): Da a conocer el enemigo al que se está atacando.

Genera el link entre jugador y héroes y da inicio a la partida, aunque antes de ejecutarse la MainWindow, se ejecuta MenuHeroe (se explicará más adelante).
Métodos:

- InicioJuego(object sender, RoutedEventArgs e): Cómo main de nuestro programa, en esta clase se generan todos los objetos necesarios para el juego, tablero, jugadores, manager(lleva mensajes al historial), las cartas, los mazos, manos etc… Dentro de este método se ejecuta InitializeComponent(), el cual ejecuta la interfaz entre otros métodos propios de la interfaz (ej: showdialog()). También abre las dos ventanas para que ambos jugadores cambien sus cartas.
- CambioImagenes(Heroe ally, Heroe enemy, String Ally, String Enemy): Revisa tanto para el aliado como el enemigo, los héroes que ambos eligieron y les asigna la imagen del heroe y la imagen del poder del heroe.
- HabilidadAliado_Click(object sender, RoutedEventArgs e): Asigna el click del boton de la habilidad de heroe a su poder respectivo.
- Save(Tablero tablero): Guarda la partida.
- Load(): Carga una partida guardada.
- ActGui(): Método el cual llama a otros cuatro métodos para actualizar la vista de la interfaz ante cualquier cambio. Llama a ActAli(), ActEne(), CambioImagenes(), ActManoAli().
- ActManoAli(List<Cartas> Cartitas): Actualiza las imágenes de la mano del jugador.
- ActTabAli(List<Cartas> Cartitas): Actualiza la vista del tablero aliado.
- ActTabEne(List<Cartas> Cartitas):Actualiza la vista del tablero enemigo.
- BajarCartaM(object sender, RoutedEventArgs e): Produce el cambio de imágenes al bajar una carta de la mano al tablero.
- FinTurno_Click(object sender, RoutedEventArgs e): Asigna el método FinTurno(), al botón fin turno de la interfaz.

- TAClick(object sender, RoutedEventArgs e): Inicializa todos los botones enemigos cuando se quiere atacar con un minion aliado.
- TEClick(object sender, RoutedEventArgs e): Se fija que en el ataque los botones(donde esta el atacante y el atacado) no esten vacios.
- Ataque(): Lleva a cabo el ataque, independientemente de si es entre minions, heroes, entre minions a/o héroes, con héroes con/sin armadura.
- Button_Click(object sender, RoutedEventArgs e): Asigna los botones.

static class ThreadSafeRandom y static class MyExtensions genera números aleatorios y crean el método Shuffle(), que baraja los mazos.

MenuHeroes:
Muestra dos ventanas donde los jugadores, eligen su nombre y heroe.
Atributos:
- int Heroe: Cada heroe tiene un int.
- Int j1:Une el heroe con el jugador.
- string nombre: Nombre del jugador.

Métodos:
- ClickHeroe(object sender, RoutedEventArgs e): Asigna el heroe con el jugador.
- Listo(object sender, RoutedEventArgs e): Cierra la ventan al seleccionar el heroe y el nombre.

CambioCartas:
Ventanas que se abren para que los jugadores puedan cambiar sus cartas iniciales, para el primero son 3 y el segundo son 4.
Atributos:
- int i#: Existen 4 de estos ints, que se usan para ver las posiciones de las cartas que se quieren cambiar.
- Jeque: Se le da un jugador que es quien está cambiando sus cartas.

Métodos:
- CambioCartas(object jugador): Hace el cambio de cartas. Ocupa una checkbox con el método: CheckBox_Checked(object sender, RoutedEventArgs e).
- Button_Click(object sender, RoutedEventArgs e): Cierra la ventana al estar listo.

Tablero:
El tablero es donde se crea el juego y sobre el cual los jugadores llevan la partida. Tiene una relación de asociación con la clase Jugador.
Atributos:
Jugador J1: Asigna al jugador 1 al tablero.
Jugador J2: Asigna al jugador 2 al tablero.
int Turno: Contador que muestra el turno en que está la partida.
Manager manager: Da mensajes al historial.
Random rdm = new Random(): Genera un número aleatorio, se usa a través del programa para por ejemplo generar los totems.
List<Cartas> Historial = new List<Cartas>(): Historial que muestra de forma escrita que se hizo el último turno.

Hechizos Moneda = new Hechizos("The Coin", 0): Creación de la moneda para el segundo jugador.
Boolean J1Jugando: Booleano que especifica en los métodos si el jugador que se le dio como parámetro es el que está jugando (es aliado).

Métodos:

- Partir(): Método con el que se inicia el juego, determina al jugador que parte, entrega las respectivas cartas de la mano a cada jugador y les ofrece cambiar cartas. utilizando el método CambiarCarta de la clase Jugador y luego llama a InicioTurno.
- CambioCartas3(Jugador J, int re1, int re2, int re3): Cambia las cartas para el jugador que va primero.
- CambioCartas4(Jugador J, int re1, int re2, int re3, int re4):Cambia las cartas para el jugador que va segundo.
- InicioTurno(Jugador J) :Recibe como parámetro al jugador que tiene que jugar, le muestra su mano y le pregunta qué desea hacer, con opciones que llaman a distintos métodos tales como: BajarCarta(), UsarCarta(), Conceder(), FinTurno() y UsarHabilidad().
- UsarCarta(Jugador J): Método que muestra la mano del jugador actual y le permite elegir una para bajarla al tablero.
- Ataque(int x, Jugador J, int y): Ejecuta el ataque en el juego, ya sea entre minion/minion, minion/heroe.
- AtaqueHeroe (Jugador J, int y):Ejecuta el ataque en el juego, ya sea entre heroe/minion y heroe/heroe.
- UsarHabilidad(Jugador J): Método que permite que el jugador utilice la habilidad especial de su héroe, llama al método de mismo nombre que se encuentra en Jugador.
- FinTurno(Jugador J): Método que finaliza el turno de un jugador y inicia el turno de el otro mediante el método InicioTurno().
- Memorias(): Lleva el número de minions que hay en juego.
- Conceder(): Método que modifica la vida del jugador que lo utiliza por 0 y luego finaliza el turno, para que el jugador pierda.

Jugador:
Clase que contiene la información del jugador, tiene una relación de asociación con las clases Mano, Mazo y Tablero. Además utiliza la interfaz Jugar la cual se relaciona con tablero y mano, además tiene una clase hija Héroe. Además genera cartas nuevas como los totems, silverhand y wicked knife de las clases shaman, paladin y rogue.
Atributos:

- String Nombre: Nombre del jugador.
- List<Cartas> Mano = new List<Cartas>(): Mano del jugador.
- Mazo Mazo: Mazo del jugador.
- List<Cartas> Tablero = new List<Cartas>(): Lista que guarda los minions en le tablero.
- Boolean ID: booleano true para jugador 1 y false para jugador 2.
- Manager manager:Manager que da los mensajes a historial.

- Random rdm = new Random(): Generador de números aleatorios para los totems del heroe shaman.
- int vida = 30: Vida del jugador.
- int ManaOrig: Mana inicial.
- int Mana: Mana del jugador.
- int Armor: Armadura del jugador (en la clase warrior).
- Heroe Heroe: Heroe del jugador.
- int VarAuxDmg = 0: VarAuxDmg, HabMej y UsoHab son variables auxiliares que se ocupan para implementar la habilidad de héroes
- bool HabMej: Booleano que da a conocer si tiene una habilidad mejorada o no (no se ocupa en nuestro programa ya que por tiempo no se implementó de forma extra la cartas Justicar TrueHeart.
- bool UsoHab: Indica si el jugador usó su habilidad o no.
- Armas Arma: Arma del jugador.
- int Daño = 0:
- int totem#count: Integers que van indicando que totems se han generado para que así el random del tótem pueda funcionar y generar distintos totems cada vez.

Métodos.

- BajarCarta(int z): Método que remueve una carta de la Mano y la mueve a la lista Memoria (Jugador 1 o jugador 2) del tablero.
- Revmano(): Método el cual revisa que un jugador no pueda tener más de 10 cartas en su mano en el caso de robar con 10 cartas en mano la carta robada se “quema”, desaparece del mazo y la mano y pasa al historial.
- CambiarCarta(int z): Método que toma una carta de la Mano y la devuelve al Mazo, luego saca una carta de Mazo y la agrega a Mano, sólo funciona si el turno es 0.
- Conceder():
- RobarCarta(): Método que saca una carta de Mazo y la agrega a Mano.
- UsarCarta(int z): Método que utiliza el ataque de un minion o arma. Además resta el costo de la carta (en mana) a la cantidad de maná disponible del jugador.
- UsarHabilidad(): Método que aplica la habilidad especial del jugador(Héroe).
- Fatiga(): Método el cual baja la vida del jugador cuando este debe robar una carta y no posee más cartas en su mazo, el valor de la vida dañada va aumentando con el número de veces que se activa el método.
- GenTotem#(): Métodos que crean el tótem que se eligió mediante UsarHabilidad() para la clase shaman.

Héroe:

Clase hija de la clase Jugador, que define el héroe que se está utilizando.

Atributos:

- NombreHeroe: String el cual contiene el nombre del Héroe.
- Clase: Contiene un String que representa el tipo de héroe. (Importante al momento de ver la habilidad del Héroe, notar que no se puede usar el nombre del héroe para este propósito ya que existen héroes adicionales que presentan las mismas habilidades, ejemplo de esto puede ser Tyrande y Anduin, ambos curan en dos a unos de sus esbirros o a sí mismos, así también Morgl y Thrall generan totems aleatorios.)

- Manager: Recibe avisos y entrega las respuestas del usuario.

Métodos.

- Mensaje(): Muestra mensajes (pre-hechos) que un jugador puede mostrarle al adversario.

IJugar:

Interfaz que contiene todos los métodos necesarios para jugar.

Métodos:

- RobarCarta(): Método que saca una carta de Mazo y la agrega a Mano.
- BajarCarta(): Método que remueve una carta de la Mano y la mueve a la lista Memoria (Jugador 1 o jugador 2) del tablero.
- CambiarCarta(): Método que toma una carta de la Mano y la devuelve al Mazo, luego saca una carta de Mazo y la agrega a Mano, sólo funciona si el turno es 0.
- UsarCarta(): Método que utiliza el ataque de un minion o arma. Además resta el costo de la carta (en mana) a la cantidad de maná disponible del jugador.
- UsarHabilidad(): Método que aplica la habilidad especial del jugador(Héroe).
- Fatiga(): Método el cual baja la vida del jugador cuando este debe robar una carta y no posee más cartas en su mazo, el valor de la vida dañada va aumentando con el número de veces que se activa el método.
- RevMano(): Método el cual revisa que un jugador no pueda tener más de 10 cartas en su mano en el caso de robar con 10 cartas en mano la carta robada se “quema”, desaparece del mazo y la mano y pasa al historial.
- Conceder(): Método que cambia la vida del jugador a 0 y pierde el juego.

Manager:

Clase conectada con Jugador y Tablero la cual lleva todos los avisos y respuestas del juego. Creada para abstraer la implementación del modelo de la utilización de consola, esto para no tener problemas al momento de desarrollar la interfaz gráfica.

Atributos:

string Aviso1.

Métodos:

Aviso(): Método por el cual se da aviso de todos los métodos y avisos que se están ejecutando.

RecibirRespuesta(): Da las respuestas de los usuarios para rellenar los datos del juego.

Cartas:

Es una clase abstracta con tres clases hijas: Armas, Hechizos y Minions. Compone a la clase Mazo.

Atributos:

- Nombre: contiene un String con el nombre de la carta.
- Costo: Tiene un número entero con el costo de maná de la carta.

OBSERVACIÓN: Para esta entrega, encontramos un problema al querer acceder a los atributos de la clase minions cuando estos se encontraban dentro de la lista tablero, y la consola arrojaba un error, donde informaba que se estaba asumiendo que el elemento al que queríamos acceder era un minion, cuando la lista es de cartas, por lo que para poder

arreglar el problema se pasaron los atributos, de la clase minion a cartas, en un futuro se pretende arreglar la clase minions, (cuando se pidan además cartas y armas)

- Ataque: Contiene un número entero el cual representa el ataque del minion.
- Vida: Contiene un número entero el cual representa la vida del minion.
- Memoria: Contiene un número entero el cual representa el número de turnos que un esbirro permanece vivo en el tablero.
- Estado: Booleano que da el estado de activo o inactivo a una carta que esté sobre el tablero (leer método a continuación para más detalle).

Métodos:

- AplicarEfecto(): Aplica la habilidad especial que tiene la carta. (Para la entrega 2 no existe ningún minion u otra carta con un efecto, por lo que solo se deja expresado, para en una futura entrega evaluar si se agrega el método/s.)

Armas:

Clase para las cartas de tipo “Arma”, es una clase hija de la clase “Cartas”.
Atributos:

- Ataque: Contiene un número entero el cual representa el ataque del arma
- Durabilidad: Contiene un número entero el cual representa la durabilidad del arma

Esta clase no contiene métodos.

Hechizos:

Clase para las cartas de tipo “Hechizo”, es una clase hija de la clase “Cartas”.
Observación: Las cartas de hechizo solo poseen un valor de maná (guardado en la clase abstracta padre) y su efecto.
Esta clase no contiene métodos.

Minions:

Clase para las cartas de tipo “Minion”, es una clase hija de la clase “Cartas”.
Atributos:

- Ataque: Contiene un número entero el cual representa el ataque del minion.
- Vida: Contiene un número entero el cual representa la vida del minion.
- Memoria: Contiene un número entero el cual representa el número de turnos que un esbirro permanece vivo en el tablero.
- Estado: Booleano que da el estado de activo o inactivo a una carta que esté sobre el tablero (leer método a continuación para más detalle).

Esta clase no contiene métodos.

Mazo:

Clase la cual guarda las cartas del “mazo” de cada jugador. Tiene una relación de asociación con la clase Jugador y utiliza la interfaz IJugar, además tiene una relación de composición con Cartas.
Atributos:

- miMazo[]: Lista miMazo[] la cual guarda las cartas del mazo.
- NombreMazo: String el cual contiene el nombre del mazo.

- FatigaCount: int el cual lleva el valor en que se disminuye la vida del jugador cuando se activa fatiga, va aumentando de acuerdo al número de veces que se activa el método fatiga.

Métodos:

- Fatiga(): Método el cual, cuando no quedan cartas en el mazo, reduce la vida del jugador cuando este roba una carta o bien mediante algún efecto de otra carta se produce que el jugador robe una carta.

Modelo Relacional:

Del modelo relacional de la entrega 1, no se realizaron grandes cambios, principalmente se trabajó en las clases tablero y jugador. Se decidió mover las zonas de tablero a la clase jugador ya que este baja las cartas al mismo, a ambas clases se les agregaron atributos y métodos para poder completar la entrega. El cambio con respecto a la clase cartas a la cual se le asignan los atributos de la clase minions, debe ser tomado como un cambio temporal, el cual se va a retomar y optimizar para la entrega 3, por ahora la clase minions no toma parte real de la funcionalidad del programa y queda propuesta (como un camino avanzado) para la entrega 3.

Supuestos:

A continuación describiremos los supuestos que hacemos para nuestro juego Hearthstone.

En esta entrega se debe entregar el código c# el cual contenga el juego (la base de este), dentro del código agregamos funcionalidades extras a las que se piden en el enunciado con el fin de tener un juego más completo, algunas funcionalidades se dejaron propuestas(a modo de tener trabajo hecho para futuras entregas, en el caso que se pidan ya se tendra parte del codigo listo.). Se actualizó el modelo relacional, el cual se anexará en un archivo visio, se recomienda ver el modelo relacional antes de evaluar el código.

1. Se supone que en consola se ingresarán solo opciones válidas (Por tiempo se irán agregando mensajes “Opcion invalida” en las siguientes entregas).
2. El programa actualmente está como público en todas sus clases, esto para no perder tiempo cuando se encuentran con problemas en el código.
3. Suponemos que siempre juegan sólo dos jugadores.
4. Los jugadores utilizan las misma de cartas, sin embargo, cada uno posee su lista(mazo) con ellas ordenadas de manera aleatoria .
5. No existen clases de minions, se supondrá que no van a existir clases tipo mech, beasts, totem, dragon, murloc, pirate, demon, etc… (En el caso que se pida este tipo de clase solo basta agregarla como atributo a la clase minion.)
6. Se suponen que la entrega de este programa no contempla los últimos cambios al juego (Un’goro expansión) como un nuevo tipo de clase de cartas tipo Quest, la cual actúa como un “secreto” el cual tras cumplir sus condiciones genera nuevas cartas y/o hechizos.
7. Se asume que por cada turno se le agrega una unidad de maná a cada jugador, hasta llegar al máximo de 10, habilidades especiales de cartas que otorguen

cristales de mana o refresquen cristales de mana al jugador solo podrá hacerlo hasta llegar a un máximo de 10 (el ya establecido), en el caso de algunas cartas de héroe druida que aumentan en un cristal de maná vacío, si se llega al límite de 10 cristales y se juega una carta de este estilo se sigue como lo hace el jugo y el efecto de esa carta cambia a robar una carta. La carta “Moneda”, se creará en el momento que se pida por enunciado, por el momento el segundo jugador no cuenta con esta medida, y por lo tanto parte con una desventaja.

8. Al ser dos jugadores con el mismo mazo, y al ser pedido por enunciado que solo deben jugar dos jugadores suponemos que no se ocupará la función de silenciar en juego de hearthstone (click derecho sobre el héroe oponente).
9. Los Hechizos del subtipo Secretos, se aplican como efectos pero en si son hechizos por lo que no ameritan una clase separada de ésta y no se animaran con un signo ?, sino que se supone que el oponente sabe que se jugó un secreto.
10. UsarCarta(), también contempla el uso de armas.
11. La clase héroe es hija de la clase jugador, esto quiere decir que jugador seria el usuario de battlenet(blizzard) osea la cuenta del jugador que se instancia mediante la elección de un héroe (Gul’dan, Thrall, Jaina etc…). La creación de clase héroe no está demás, ya que un jugador puede existir en el juego sin un héroe asignado, y al momento de entrar a una partida este debe elegir un héroe.
12. Fatiga() : La vida del jugador al robar la primera carta (con el mazo vacío) al jugador se le reducirá su vida en 1 al jugador y en 1 adicional cada vez que se active Fatiga(), es decir si se activa 3 veces la primera dañara en 1 al héroe luego en 2 y finalmente en 3.
13. Cada turno dura un máximo de 75 segundos. La cuerda que se activa mediante el método TiempoTurno(), el cual es un método propuesto y se activa cuando al jugador le quedan 20 segundos por jugar. Esta parte del programa no se implementó ya que finalmente se decidió no pedirla dentro de las entregas.
14. Los efectos individuales de cada carta se guardan y aplican con AplicarEfecto() en la clase Cartas. (Este método queda propuesto ya que para esta entrega no se pide.)
15. El juego puede realizar todos los tipos de jugadas que se piden implementar, bajo algunos casos especiales se puede forzar a que el juego se caiga, pero siempre intentando realizar jugadas que no se pueden hacer en el juego, por lo que si un jugador sabe jugar el juego no debería tener problemas en la jugabilidad de este.
16. La generacion de totems en la habilidad de heroe de la clase shaman se logró implementar tal cual como lo hace en el juego con una excepción, en el juego la habilidad del shaman siempre debe generar distintos totems, es decir hasta que los 4 tipos de totems no hayan salido al tablero no se pueden repetir los totems, excepto si estos totems se mueren, esta ultima parte no la considera el método que genera los totems de shaman, independientemente si los totems se encuentran vivos o muertos siempre se generarán tótems que no hayan salido. ej:

Se usa la habilidad y sale el tótem 1 luego el tótem 2 , despues matan a ambos totems, al usar la habilidad después de este caso solo se pueden generar de manera aleatoria el tótem 3 y el tótem 4.