Universidad De Santiago De Chile
Facultad de ingeniería
Departamento de ingeniería informática

Informe de Laboratorio 3: Juego Dobble

Nombre: Cristobal Francisco Marchant Osorio
Profesor: Roberto González I.
Asignatura: Paradigmas de programación

11 de julio del 2022

Tabla de contenidos

Pagina contenido

Introducción

El presente informe habla sobre el desarrollo del laboratorio N°4 de la asignatura de paradigmas de programación, este informe consiste en que gracias a la utilización de “Visual Studio” en su versión 17.2.5 podemos usar el lenguaje de programación C#, para la realización de modelos, interfaces y vistas. En este informe se plantean los objetivos de explicar brevemente el problema planteado, darle un análisis a este problema, generar un diseño a la solución de este, mostrar aspectos de la implementación, instrucciones y como hacer usos de estas, mostrar resultados, una autoevaluación y una conclusión acerca del trabajo, todo esto enfocado al paradigma orientado a objetos y dirigido por eventos, en donde como ya se mencionó se hará uso del lenguaje C# para realizar las modelos de clase “Carta (Card)”, “Jugador (Player)”, “Mazo (Dobble)” , “Tablero (DobbleGame)” y para la vistas. Con todo esto antes mencionado se deja invitado al lector a ver cada implementación y que pueda entender el modo de uso de estas.

Descripción del problema

El problema planteado consiste en desarrollar el juego de cartas “Dobble”, en donde cada carta tiene un elemento en común. Sobre este tema se tendrá la creación de los 4 modelos de clase anteriormente mencionadas, además de la creación de un menú (vista) en donde un jugador pueda interactuar en distintos modos de juegos.
Las 4 clases con interfaces que se implementaran son:

Jugador: Serán los jugadores que se registren en el juego, estos constarán de su nombre, puntaje y turno.

Carta: Constará de una lista de elementos, esta servirá para poder construir el mazo.

Mazo: Esta clase tendrá como atributos el número máximo de cartas y el número de elementos. Como métodos tendrá el poder eliminar cartas, ver si es igual otro mazo, etc. además de poseer un constructor que reciba los parámetros de entrada máximo de cartas, número de elementos y orden.

Tablero: Este tendrá como atributos una lista de los jugadores, un mazo de cartas, el estado del juego y la modalidad. Como métodos tendrá el poder registrar jugadores, poder jugar, obtener datos de los atributos, poder finalizar la partida, etc.

Por otra parte, las vistas serán:

Main: Es el menú inicial, en donde se elegirá el modo de juego.

Datos: Es la sección en donde se ingresan los datos del tamaño de la carta y la cantidad de jugadores

Register: Es la sección en donde se ingresan los nombres de los usuarios a registrar, cabe destacar que serán mínimo 2.

Play: En esta sección se podrá elegir entre hacer spotit, pasar turno y terminar la partida.

Finish: Mostrara los resultados de los ganadores

Descripción del paradigma

En este trabajo se trabajará bajo los estándares del paradigma orientado a objetos y el paradigma dirigido por eventos, en el cual el primero de estos se basa en la declaración de objetos, en donde estos objetos dependiendo de su abstracción poseen atributos y métodos que pueden interactuar con otros objetos, mientras que el paradigma dirigido por objetos se entiendo que a base de un suceso ocurrido en el sistema se determina la ejecución del programa.

Las Clases son definiciones de las características o también llamadas atributos y también son la definición de los métodos o comportamientos para un tipo de clase.

Los Objetos son “instancias de la clase”, esto quiere decir que durante la ejecución de un programa estas están activas en memoria.

Los Atributos antes mencionados son variables que caracterizan a un objeto, estos pueden ser tipo de datos primitivos u otras clases.

Los Métodos son un comportamiento que denota un objeto, pueden percibirse como función desde la vista de otro paradigma, estos métodos poseen un comportamiento que realiza sobre otros o sobre sí mismo.

Los Constructor nos ayudan a crear objetos basado en una clase, es decir, formar una nueva instancia de un objeto.

Las Vistas se utilizan para visualizar los registros del modelo.

Los Eventos son sucesos que permiten determinar el manejo y flujo del programa.

Existen relaciones entre clases, estas llamadas composición, agregación, asociación, dependencia y herencia.

Desarrollo

Análisis del problema

El desarrollo de esta actividad consta en la construcción de 4 Clases con sus respectivas interfaces y 1 vista interactivo controlada por el o los jugadores, las 4 clases están compuestas por la clase carta, jugador, mazo y tablero, mientras que las vistas serán “Main”, “Datos”, “Register”, “Play” y “Finish”.

CLASE CARTA (Card)

Se representa como una lista de objetos.

Posee el siguiente método fundamental.

Descripción	Parámetros de entrada	Retorno
Obtener (get)	()	Carta
Modificar (set)	1 carta	VOID
Comparar 2 cartas	2 cartas	Booleano
Convertir a String	1 carta	String

CLASE MAZO (Dobble)

Para entender la base del problema es necesario tener en cuenta los datos de entrada solicitados para el constructor, para la creación de las cartas se ingresa un número de elementos por carta, pero este número es un n+1, el n tiene que cumplir condiciones para que este sea válido, como es el caso de que este sea primo, porque si no fuera primo existirían cartas con más de 2 elementos en común o que no tengan ningún elemento en común y esto no es posible, ya que por la definición de las reglas del juego, cada carta tiene que tener un solo elemento en común.

Para la clase mazo se tienen los siguientes métodos:

Descripción	Parámetros de entrada	Retorno
Constructor	Entero (largo), Entero (Numero de cartas) y Boolean (cartas al azar)	Mazo
¿Es un mazo valido? (T or F)	Mazo de cartas	Boolean
Una carta en cierta posición	Mazo de cartas, la posición	Carta
El total de cartas	Una Carta	Entero
De un mazo recortado encontrar las cartas restantes	Un mazo recortado	Mazo
Obtener (get)	()	Mazo
Modificar (set)	1 mazo	VOID
Comparar 2 mazos	2 mazos	Booleano
Convertir a String	1 mazo	String

TDA JUGADOR (Player)

Jugador es el usuario que realice una acción dentro del tablero, es por lo que este posee como atributo un nombre, un puntaje y un turno.

Entonces tenemos los siguientes métodos:

Descripción	Parámetros de entrada	Retorno
Obtener nombre(get)	()	String (Nombre)
Modificar nombre (set)	1 mazo	VOID
Obtener puntaje (get)	()	Integer (Puntaje)
Modificar puntaje (set)	1 mazo	VOID
Obtener turno (get)	()	Integer (Turno)
Modificar turno (set)	1 mazo	VOID
Obtener (get)	()	Jugador
Modificar (set)	1 jugador	VOID
Comparar 2 jugadores	2 jugadores	Booleano
Convertir a String	1 jugador	String

CLASE TABLERO (DobbleGame)

Es donde se llevará a cabo el juego y que presenta los métodos para que un usuario pueda jugar.

Entonces tenemos los siguientes métodos:

Descripción	Parámetros de entrada	Retorno
Obtener para cada atributo (get)	()	Mazo || Lista de jugadores || Modalidad || Estado de juego
Modificar para cada atributo (set)	Mazo || Lista de jugadores || Modalidad || Estado de juego	VOID
Comparar 2 Tableros	2 tableros	Booleano
Convertir a String	1 tablero	String
Registrar jugador	Nombre del jugador	Integer
Ver el turno actual	()	String (nombre)
Jugar	Elemento	VOID
Resultado	()	VOID

Vistas (“Main”, “Datos”, “Register”, “Play” y “Finish”)

Se piensan en funciones void que permitan avanzar, regresar, salir, spotit, pasar turno, finalizar partida, volver a inicio y registrar, todas estas distribuidas según se requiera.

RELACIONES ENTRE CLASES E INTERFACES

Realización de un diagrama UML en donde cada flecha representa una relación entre 2 clases o interfaces o los modelos vistas, estas relaciones son composición, agregación, asociación, dependencia y herencia, de esta manera se tiene un Diagrama UML de entrada, es decir, que en el diseño de solución se planteara un Diagrama UML final.
(ver anexo, figura 1).

Diseño de solución

Para diseñar la solución al problema, se plateo la creación de la clase carta como una lista de objetos, por su parte la clase mazo seria una lista de cartas, la clase jugador posee los atributos que llevará cada jugador y la clase tablero tendrá de atributos la lista de jugadores, el mazo, la modalidad y el estado de juego, por su parte las vistas poseerán funciones de los modelos, especialmente del “DobbleGame”, para poder realizar acciones a base de sucesos.

CLASE CARTA (Card)

Se diseña simplemente como una lista de objetos. (ver anexo, figura 2)

En el **ToString** se utilizó bajo un override, y retornaba el contenido de la carta.

En **Equals** también se utilizó bajo un override y comparaba de manera en que, si la otra carta tiene los elementos desordenados, pero son los mismos elementos, este retorna true.

CLASE MAZO (Dobble)

A continuación, se explicarán como funcionarán las entradas para el **mazo de cartas**, para el constructor del mazo se tomarán en cuenta 5 aspectos de entrada:

El primer parámetro es el número de elementos por carta, este aplico de manera (((NElem - 1)^2) + NElem) es así como podíamos saber cuántas cartas tendríamos en total.
El segundo parámetro fue el máximo de cartas, esto se pensó en cuanto se tuviera el mazo de manera de acortar el mazo en base a este número.
El tercer parámetro es un booleano para que desordene el mazo.

Para el caso del método de “**isDobble()**” este se pensó como un conjunto de 3 funciones que controlan que el mazo cumpla ciertos requisitos, estos son:
1. Todas las cartas tienen el mismo largo.
2. Todas las cartas tienen únicamente un solo elemento en común.
3. Cada carta no puede tener elementos repetidos.

En **NthCard** se aplicó un get(i) sobre el mazo, tomando en cuenta que esta sea de 0 hasta n-1.

En el caso de **FindTotalCards** se pensó en contar la cantidad de elementos de la carta y aplicar ((n-1)**2)+n.

En el caso de **ElementsRequeridos,** al igual que el caso anterior esta se pensó en contar la cantidad de elementos de la carta y aplicar ((n-1)**2)+n.

En el **MissingCards** se implementó de manera en que se creara un nuevo mazo a partir de 1 carta del mazo principal, de esta manera las cartas en común se eliminan gracias al ".remove()".

En el **ToString** se utilizó bajo un override, y retornaba el contenido del mazo.

En **Equals** también se utilizó bajo un override y comparaba las cartas bajo su propio equals, además si el otro mazo también tiene las cartas revueltas, este retorna true.

CLASE JUGADOR (Player)

Para la creación de este, se implementaron los atributos que lo formaran, estos son el nombre del jugador, sus puntos (inicialmente 0) y su turno (inicialmente 0).

Este posee los siguientes métodos principales:

Get y Set aplicado a cada atributo que posee jugador.

En el **ToString** se utilizó bajo un override, y retornaba el contenido del jugador.

En **Equals** también se utilizó bajo un override y retornaba un true o un false.

CLASE TABLERO (DobbleGame)

Para la creación del constructor del constructor del **DobbleGame** se agregaron los atributos de lista de jugadores, número de jugadores, modo de juego, mazo y el estado del juego.

Para la creación del **RegisterUser** se basó en que se encontraba el número de jugadores y se le resta 1 a este, a su vez se agrega el jugador a la lista de jugadores con turno y puntos iguales a 0, en caso de que el nombre del jugador ya este registrado, este retorna una alerta y no lo agrega.

Para la creación del Selector **WhoseTurnIsIt**, esta se basó en buscar quien estaba de turno según la lógica que la lista siempre vaya de izquierda a derecha y cuando llegaba al final volvía al primero, entonces cuando la comparaba a 2 usuarios del juego se veía cual era mayor, entonces el menor de estos 2 estaba de turno, pero si eran iguales se compararán el segundo con el tercero y si todos son iguales el turno seria del primero de la lista.

Para la creación método **Play** consiste en que el elemento que ingrese las siguientes opciones:]-∞,-1] es para terminar el juego, 0 es para pasar turno y [1, ∞[realizar una acción dentro del juego.
El diagrama de cómo funciona la mesa de juego junto a otras funciones establecida hasta este punto se muestra a continuación: (ver anexo, figura 3).

Get y Set aplicado a cada atributo que posee el tablero.

En el **ToString** se utilizó bajo un override, y retornaba el contenido del tablero.

En **Equals** también se utilizó bajo un override y retornaba un true o un false.

Vistas (“Main”, “Datos”, “Register”, “Play” y “Finish”)

Avanzar: da paso a una nueva ventana; **Regresar:** da paso a una nueva ventana; **Salir**: genera una opción de cerrar el programa; **Spotit:** da la opción de ingresar un valor para play; **Pasar:** llama la función play, con pass; **Turno y Punto:** muestran los turnos y puntos actuales; **Finalizar partida:** llama a la función play y termina la partida, además abre paso a la ventana de resultados; **Volver a inicio:** botón que dirige a la ventana Main; **Registrar:** llama a registrar a todos los usuarios propuestos mediante la función register.

RELACIONES ENTRE CLASES E INTERFACES

Se manejo el uso de flechas en un diagrama UML en donde cada flecha representa una relación entre 2 clases o interfaces o los modelos vistas, estas relaciones son composición, agregación, asociación, dependencia y herencia, todas estas fueron utilizadas para la realización del siguiente diagrama UML.

(ver anexo, figura 4)

Aspectos de implementación

Compilador

Se utilizo el programa Visual Studio Versión 17.2.5 para la realización del Código y las vistas, bajo el UI framework WPF.

Estructura del proyecto

Para la estructura de este proyecto se realizó un uso similar al patrón MVC, el cual posee 4 carpeta en este caso: View y Model.
View contiene el menú que se muestra por interfaz grafica para que el usuario interactuare y las imágenes utilizadas en esta interfaz gráfica.
Model contiene las clases antes mencionadas (Card, Player, Dobble, DobbleGame), además de las interfaces de cada una de estas.

Instrucciones de uso

Existen 2 formas de uso, una de estas en teniendo Visual Studio en la versión 17.2.5. Luego cargaríamos la carpeta principal, para posteriormente ejecutar la solución “src”, esto nos mostrar 2 cosas, la carpeta con las clases e interfaces y la segunda la carpeta con las vistas.

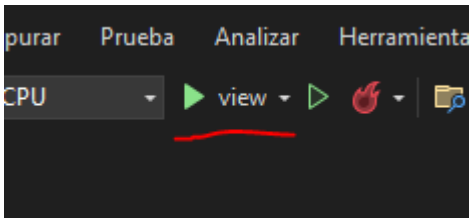

Figura 5, ejecutar desde main.

Resultados y autoevaluación

A partir de los resultados obtenidos cumple con éxito la totalidad de las funciones, en donde se pusieron a prueba los ejemplos para entregar este tipo de resultados.
A continuación, se presenta la autoevaluación.

Requerimientos Funcionales	Calificación
Clases y estructuras que forman el programa	1
Menu interactivo	1
Constructor de juegos	1
Register	1
Play	1
ToString	1
Equal(Object o)	1
User vs User	1
CPU vs CPU	1

Conclusión

Luego de terminar el proyecto se puede concluir que se cumplió con la totalidad de los objetivos planteados. Con respeto a los logros del proyecto también podemos afirmar que se cumplieron con éxito y se pretende trabajar de manera más ordenanza y mejor documentada, explicando un número mayor de ejemplos en próximos trabajos. Una de las complicaciones en relación con este trabajo fue el comprender el uso e implementación de las vistas en c# wpf en donde gracias a lo visto en clases y el material publicado en internet y por los profesores se pudo llegar a comprender e implementar, también cabe destacar que para algunas clases planteadas se requirió de una gran cantidad de tiempo, ya que estas fueron bastantes extensas y hubo confusión a la hora de implementar, pero fuera de estos tropiezos, se puede decir que se cumplió la totalidad del proyecto.

Sobre el paradigma orientado a objetos y el paradigma dirigido por eventos, gracias a la realización de este trabajo fue posible tener un mejor entendimiento sobre estos, y pensar de manera más rápida en una solución hacia un problema.

Sobre C# en Visual Studio no hubo complicaciones, ya que se facilitó el uso gracias a su fácil comprensión y sobre la plataforma git no hubo ningún inconveniente y se intentó hacer como mínimo un comit diario.

Se espera que para próximos trabajos trabajar con más tiempo, planear la idea de la implementación desde antes y trabajar de manera más ordenada.

Bibliografía

Roberto González, Víctor Flores, Miguel Trufa y Gonzalo Martínez , 2022 Proyecto semestral de laboratorio.
https://docs.google.com/document/d/1tCVZBGScJbXFspQfu6WlDl3PsyCJUcqIgos5__DQz7k/edit#

Roberto González, 2020 Material online Mostrado en clases.

Oscar Campos, 2011 Introducción a la programación dirigida por eventos
https://www.genbeta.com/desarrollo/introduccion-a-la-programacion-dirigida-por-eventos

Anexo

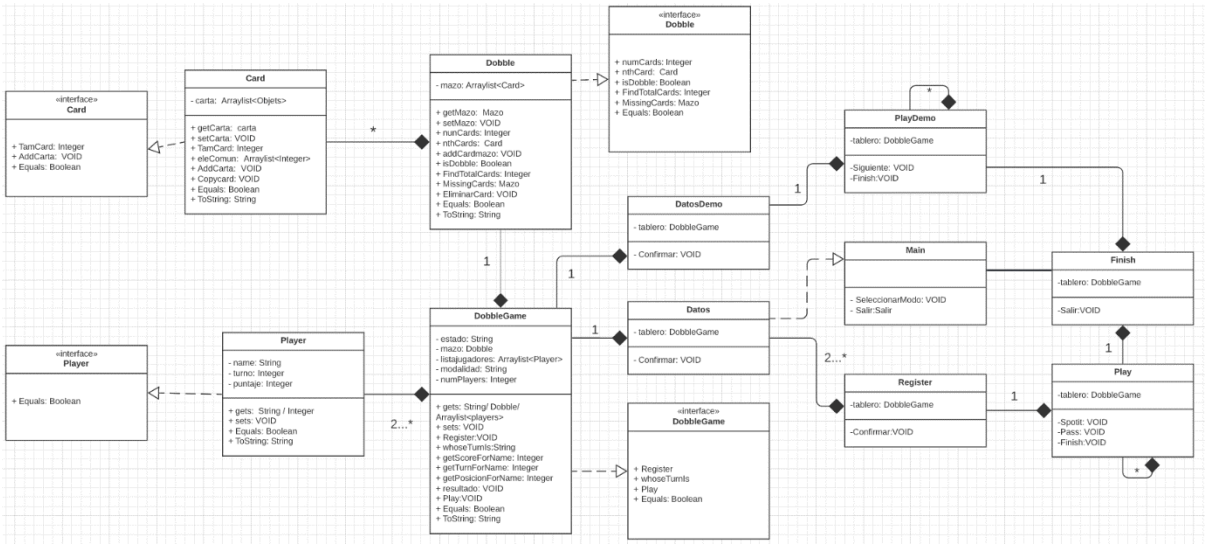

Figura 1, Diagrama UML previo.

```java
public class Card:Interface_Card
    {
    // Atributos
    private List<int> carta;

    // Metodo
    public Card(){
        this.carta = new List<int>();
    }
```

Figura 2, declaración carta.

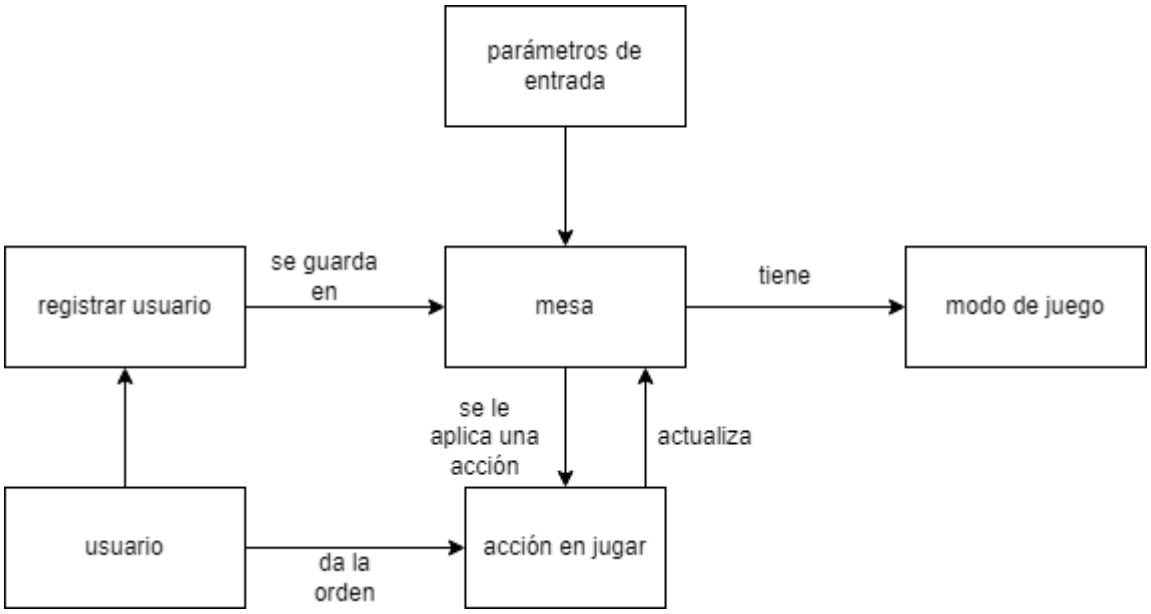

Figura 3, Diagrama del funcionamiento del game.

p

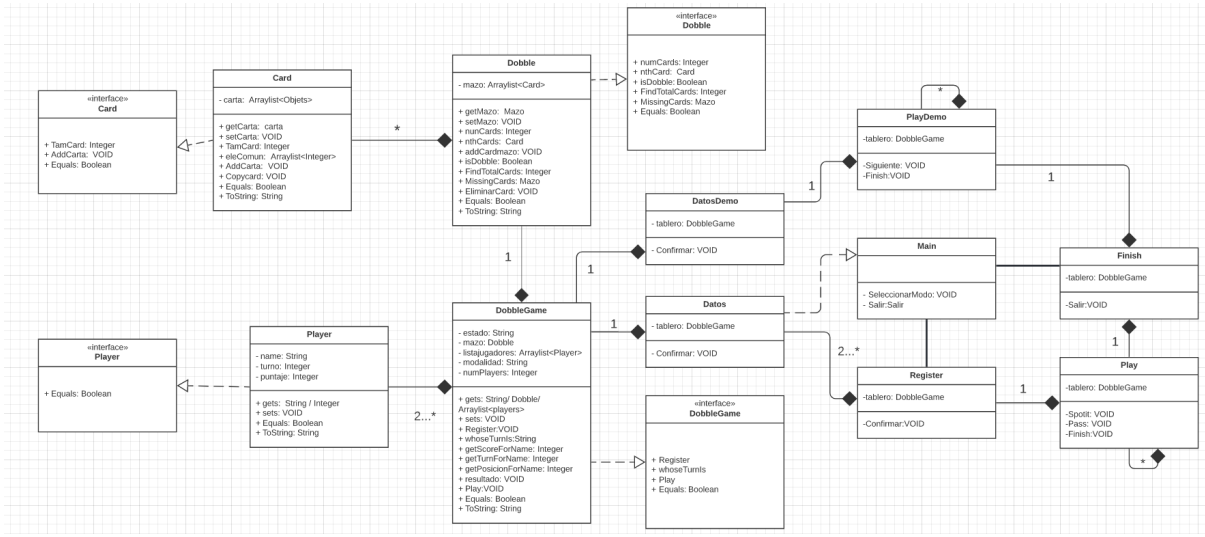

Figura 4, Diagrama UML posterior.