Proyecto de Juego de Dominó

Introducción

En el presente trabajo se trata de crear una aplicación que simule un juego de Dominó. Este juego, además de popular, es muy versátil y existen muchas variantes en el mundo, unas más populares y conocidas, que otras. Entre su variabilidad se cuenta que se puede jugar con diversos números de fichas, diversas formas de definir un inicio, desarrollo del juego, varias variantes de finalizar partidas y juego, y además, diversos criterios para definir el equipo ganador; unido a esto, las múltiples y diferentes tácticas y estrategias, que se han utilizado en diferentes escenarios, hacen del Dominó uno de los juegos más complejos de automatizar y es, por lo tanto, muy difícil abarcar todas las combinaciones posibles.

Por todo lo anterior, se escogieron varias variantes de los diferentes aspectos del juego. El esquema general del juego consiste en tres etapas fundamentales:

1.- **Inicio del juego**: donde se define, de acuerdo a diversos criterios, quién o qué equipo comienza el juego, y además que tipo de ficha será la primera en tirarse. En esta etapa se repartirán un número de fichas a cada jugador, en dependencia del tipo de juego de dominó a efectuarse.

2. **Desarrollo del juego:** en esta etapa, de acuerdo al tipo de juego acordado, se realizará una o varias partidas para definir un ganador. Cada jugador tirará la ficha, que según su comportamiento o estrategia en el juego, le convenga, siempre y cuando posea al menos una posible de colocar en la mesa.

3. **Final de partida o juego:** en dependencia del tipo de juego, previamente acordado, se puede ejecutar una partida o varias, hasta que se cumpla un criterio de finalización del juego y/o de cada partida que defina el juego.

Debido al comportamiento del juego, la manera más factible es controlar cada una de estas etapas por separado e incorporarle diferentes comportamientos. Para ello se declararon clases abstractas, para cada una de ellas y diversos comportamientos diseñados como interfaces, para simular las diferentes variantes. Esta manera modular y generalizada permite hacer el diseño escalable, sin comprometer las anteriores opciones ya incorporadas.

Por la necesidad de la interacción entre los diferentes módulos se crearon dos clases que aglutinan los datos fundamentales y funciones, estas son **MESA**, donde se aglutinan los datos necesarios para el juego, como las fichas disponibles y las que se van tirando por cada jugador, así como, los datos e informaciones que se van generando en el juego y que son necesarios acceder y/o modificar. La otra clase es **DOMINO**, donde se declaran y controlan las diferentes etapas del juego y muestran los resultados parciales.

A continuación se mostrarán las diferentes variantes tomadas en cuenta para el desarrollo de la aplicación:

Tipos de juegos:

- **Doble 6**: El dominó se compone de 28 fichas rectangulares. En este juego pueden participar dos, tres o cuatro personas.. Cada ficha está dividida en 2 espacios iguales divididos con una rayita en los que aparece con forma de puntos una cifra de 0 o blanca hasta 6.
- **Doble 9**: En esta modalidad se utilizan 55 fichas (30 pares y 25 impares) cuyos valores van desde el blanco cero hasta el doble nueve y que en total suman 459 puntos.

Cantidad de jugadores:

Estos pueden ser a partir de 2.

Tipos de jugadores:

- **Bota gorda**: Se le dice a un jugador que saca fichas altas constantemente, sin importarle lo que lleve o no su pareja. Normalmente es el estilo de los novatos.

  (Este funciona ordenando las fichas posibles del jugador de forma descendente en cuanto a cantidad de números y escoge la primera, o sea la mayor)

- **Aleatorio**: Este juega eligiendo la primer ficha que vea y se pueda poner en la mesa

  (Funciona con una función de generación de números aleatorios que elige una ficha dentro de las posibles a jugar)

- **Tramposo**: Persona que juega con trampa, mira las fichas del jugador siguiente cuando este se despista.
- **Humano**: que permite el usuario jugar con los demás.

Inicio del juego:

- **Salida Aleatoria**: Elige un jugador al azar entre los participantes.
- **Mayor Ficha**: Solo para juegos en los que se reparten todas las fichas, entonces saldrá el jugador que tengo el mayor doble en este caso doble6 o doble 9.
- **Pares o nones**: Un equipo elige ser par y el otro impar, luego se escoge una ficha entre las sobrantes y si es par sale un jugador del equipi par, lo mismo en caso contrario.

Acción ante el Pase:

- **Pase usual**: No ocurre nada, solo imprime en consola q un jugador se pasó.
- **Robar ficha si no lleva**: Cuando un jugador no lleva ficha en vez de pasarse, lo que hace es robar una ficha entre las restantes.

Final de partida:

- **Fin usual**: Se acaba la partida cuando nadie lleva.
- **Fin doble pase**: Finaliza el juego cuando un jugador se pasa 2 veces seguidas (Se controlan los pases seguidos de los jugadores mediante un arreglo, cada vez que un

jugador se pase se suma 1 al número de cada jugador, y cuando juega se restablece en 0)

Final del juego:

- **Por número de partidas**: El usuario decide cuantas partidas desea jugar, gana el que mayor cantidad de partidas ganadas tenga, en caso de empate se juega de nuevo.

  (Se controlan los ganadores de cada partida mediante la clase Partidas que tiene la propiedad ganador y juego Actual y se guardan estos ganadores en una lista de tipo Partidas.

- **Acumulación de puntos**: Gana el equipo que obtenga puntos mayor o igual a 100.

  (Se guardan los puntos obtenidos por cada equipo en un arreglo y se va comprobando, al final de cada partida, si algún equipo sobrepaso los puntos, entonces este gana)

Sistema de puntaje:

- **Fichas por puntos**: Juego clásico, se suman todos los puntos que tiene cada ficha
- **Cantidad de fichas**: cada ficha cuenta como 1 punto, los dobles cuentan como 2 puntos.

Las opciones aleatorias, aunque es raro en un juego, no obstante se incorpora como variante por la facilidad de cálculo de la computadora.

Implementación

La programación se realizó en plataforma Windows con .NET6.0, con salida a consola, cada una de las variantes se corresponden con interfaces o clases abstractas, para encapsular comportamientos comunes y permitir incorporar nuevas variantes. En el caso de las interfaces obliga a la implementación de los métodos obligatorios para un buen comportamiento del juego.

Primero se crea la biblioteca de clases denominada **DominoBase** con la cual se logra contener todas las definiciones y tareas que se realiza en general dentro un juego de Dominó. Está constituida por clases abstractas, interfaces y estructuras de datos, que representan las diferentes acciones y los resultados, tanto parciales, como finales de un juego de Dominó.

Esta biblioteca sirve de base, para la implementación de muchas combinaciones y variantes de juegos.

Posteriormente se creó una biblioteca de cases denominada **DominoLib** que contiene la implementación de las variantes, para cada uno de los aspectos del juego, descritas en la introducción, de modo tal, que un usuario cualquiera tenga ya un número de variantes desarrolladas y le permita, a partir de ellas, **confeccionar nuevas opciones de juegos o crear nuevas funcionalidades,** tomando como ejemplo, las implementaciones ya realizadas y enriquecer así, la Biblioteca de clases **DominoLib.**

Por último se implementó un ejemplo de utilización de ésta última biblioteca, tratando de abarcar todas las variantes incorporadas a la misma**.**

Para mejor comprensión y acceso se dividió en diferentes espacios de trabajo (***namespace***) que contienen las clases necesarias en cada aspecto.

La distribución de los espacios de trabajo y sus contenidos se relacionan a continuación:

Biblioteca Base

1. NAMESPACE domino

   Clase: Domino

   Descripción: contiene las variables y la declaración de las principales clases tanto abstractas,, como referencias, que controlan el juego.
   Dependenciass: jugador, mesa, ficha,partidas, Desarrollo_del_Juego, Final_del_juego, interfaces, inicio, fin_de_partida, config, variantes;
   Propiedades:
   public static int CantidadJugadores;
   public static int NumeroMaximoFichas;
   public static int CantidadFichasARepartir;
   public static List<FICHA> fichas
   public static JUGADOR[] jugadores
   public static int[] equipos
   public static List<Partidas> partida
   public static bool Apuntos
   (funcionalidades)
   public static IInicioDelJuego inicio
   public static FinalPartida finalP
   public static IPases pase
   public static ISistemaDePuntaje pts
   public static IFinJuego finalJ
   public static IReparticion reparticion
   public static IMirar Verjuego
   Métodos:
   public static void RepartirFichas()
   public static void EliminarFichaDeLaMesa(FICHA)
   public static void MostrarFichas()
   public static void Play()

2. NAMESPACE **mesa**

   Clase: MESA

   Descripción: Clase que agrupa y permite el intercambio de datos generales para todas las funcionalidades y que permite el acceso desde cualquier parte**.**

   Dependencias: jugador, ficha, domino

   Propiedades:
   public static int[] CantidadPasesSeguidosPorJugador;

public static List<FICHA> Mesa
public static int jugadorActual
public static int NumeroPasesSeguidos
public static int[] puntosPorEquipos
public static List<int>participantes

Métodos:

public static int ProximoJugador()
static public FICHA Extremos()
static public void Colocar_Ficha(FICHA)
public static void Rotar(FICHA ficha)
public static void Mostrar()
public static void MostrarEstado(string)

3. NAMESPACE **Configuracion**

Clase abstracta: _Configuracion

Descripción: concentra los parámetros cuyos valores determinan las variantes en que se ejecutará el juego.

Construcción: _Configuracion()

Dependencias: jugador, ficha, domino, inicio, fin_de_partida, variantes

Propiedades:

public int numjugadores;
public int numNPartidass;
public int numFichas;
public int Inicio;
public int FinalPartida;
public int FinalJuego;
public int Puntaje;
public int Pases ;
public int CantidadPartidas;
public int MaxFicha;
public int ver;

Métodos:

Public abstract void DefinicionJuego() : Función que permite indicar a la clase Dominó el comportamiento. Permite el desarrollo de extensiones.

4. **NAMESPACE ficha**

Descripción: Clase que representa una ficha del juego con sus dos números.
Dependencias: **no;**
Clase: FICHA
Propiedades:

public int num1
public int num2

5. NAMESPACE **jugador**

Clase abstracta: **JUGADOR**

Descripción: Clase que recoge los datos y las acciones que realiza un jugador, mediante herencias de esta clase se pueden implementar los comportamientos como: "*Botagorga", Tramposo*", etc. Contenidos en **DominioLib.**

Dependencias: **mesa, ficha**

Propiedades:

```
public List<FICHA> FichasDelJugador;
public List<FICHA> Fichas_Tiradas;
public List<int> Pases;
public int numeroJugador;
 public int Equipo { get; set; }
 public int Numero { get; set; }
```

Constructor: JUGADOR()

Métodos comunes:

```
public void RecogerFicha(FICHA ficha)
public void TirarFicha(FICHA ficha)
public FICHA TirarFichaEscogida()
public  List<FICHA> fichas_que_Lleva_Jugador(FICHA extremos)
```

nota: "*FICHA extremos*" representa los números extremos de la mesa.

Método abstracto:

```
FICHA Escoger(List<FICHA> Fichas_posibles);
```

Nota: este es el método que cambia según tipo de jugador.

6. NAMESPACE **Desarrollo_del_Juego**

Clase: Juego

Descripción: esta clase posee los métodos para hacer fluir el juego y recoger el ganador final.

Dependencias: **jugador, domino, mesa, ficha**;

Propiedades:

```
public static bool Pase
public JUGADOR ganador
```

Métodos:

```
private void Siguiente()
public void jugar()
```

7. NAMESPACE **inicio**

Clase: interface IInicioDelJuego

Descripción: Esta interface obliga en su declaración a implementar la manera en que se determinará, quien y con cuál ficha se comenzará el juego. En **DominoLib**, se implementaron los inicios: *SalidaAleatoria, MayorDoble*, entre otros, para escoger.

Dependencias: domino, ficha, mesa;

Propiedades:

public static int jugadorInicial;

Métodos:

public void Inicio();

8. NAMESPACE fin_de_partida

Clase: FinalPartida (abstracta)

Descripción: Esta clase permite definir cómo se determina el fin de una partida y recoge el resultado del equipo ganador. La herencia de esta clase permite recrear las diversas variantes, en particular en DominoLib se implementaron las variantes: por pase general o *Fin doble pase.*

Dependencias: jugador, mesa, domino, variantes, interfaces;

Métodos:

public abstract bool Fin()
public JUGADOR Ganador()

9. namespace Final_del_juego

Clase: interface IFinJuego

Descripción: Esta clase permite definir cómo se determina el final del juego y recoge el resultado del equipo ganador. La implementación de esta interface permite recrear las diversas variantes, en particular en DominoLib se implementaron las variantes: "*Por número de partidas" y "Acumulación de puntos".*

Dependenciass: partidas, domino, Desarrollo_del_Juego, mesa

Métodos:

public bool Final();
public int EquipoGanador();
public string Tipo();
public void Accion(Juego j);

10. NAMESPACE **interfaces**

Descripción: donde se declaran una serie de interfaces auxiliares necesarias para poder realizar acciones en el juego:

Dependencias: No

- **Clase**: **interface IPases**

  Descripción: define el modo de actuar ante la ocurrencia de un pase. En DominoLib**: "***Pase usual" y "Robar ficha si no lleva***"**

  Método:

  void Adicionar();

- **Clase**: **interface ISistemaDePuntaje**

  Descripción: define el modo en se acumularán puntos, en DominoLib: *"PuntosSimples" y "PuntosDeLasFichas"*

Método:

public int DevolverPuntos(int jugador);

- **Clase**: public interface IMirar

  Método: public void Mirar();

- **Descripción:** permite definir el comportamiento del flujo del programa: paso a paso o continuo.

11. NAMESPACE repartirfichas

Clase: **interface IReparticion**

Descripción: define el modo en se repartirán las fichas, en DominoLib: *"*Repartir66*" y "*Repartir99*"*

Método:

public void RepartirFichas()

12. NAMESPACE **partidas**

Dependencias: jugador;

Clase: Partidas

Descripción: controla el flujo de las partidas

Propiedades:

public JUGADOR ganador
public int puntosObtenidos
public int juegoActual

Constructor: public Partidas(JUGADOR ganador)

Implementación de un juego

Para la implementación de un juego se deben seguir los siguientes pasos:

1. Crear un proyecto nuevo de tipo "Aplicación de Consola"
2. Incorporar la dependencia de éste con la biblioteca de clases **DominoBase** y **DominoLib**, si desea utilizar las imlementaciones ya hechas sobre la base.
3. Declarar la clase Configuración ;
4. Completar los parámetros de dicha clase (esto puede hacerse directamente utilizando las funciones admisibles encontradas en la biblioteca DominoLib o confeccionar un menú que permita escoger para cada parámetro su valor correspondiente). En el presente proyecto se presenta ésta última opción.
5. Finalmente llamar al método Play() de la clase tipo **Domino.**
6. A partir de este último paso, se ejecuta automáticamente la aplicación.