Reporte

Partes

- La aplicacion cuenta con dos partes :
  - Una `dll (Logica.domino.dll)` donde esta implementada toda la logica de la aplicación.
  - Una aplicaion de consola encargada de mostrar lo que ocurre en el juego asi como permitir la modificacion de los aspectos personalizabes del mismo.

Estructura de carpetas

- Domino
  - Consola : Contiene todo lo relacionado con la ineterfac grafica del proyecto
  - Logica.Domino.dll : Contiene todo lo relacionado con la parte logica del proyecto. Cada subcarpeta de Logica.Domino.dll tiene asignado el nombre segun la funcionalidad que almacena, pues para casos como la carpeta Reglas no seria conveniente relacionar su contenido con el contenido de otra funcionalidad debido a la extension del mismo. Algunos fichero no estan contenidos en una subcarpeta debido a que no tienen nigun otro fichero relacionado con su funcionalidad; tal es el caso de `Mesa.cs`.

Logica.domino.dll

- #### Interfaces y clases que las implementan

  Las interfaces permiten imponer propiedades y las clases sirven para agrupar las caracteristicas y propiedades de un tipo.

  - IModo : Cada Modo deteremina la cantidad de partidas a jugar, ya sea por puntos o simplemente por la cantidad de partidas jugadas. Tambien determina el ganador del juego.

```
public interface IModo
{
    int CantidadJugadores{get;}//da la cantidad de jugadores en la partida
actual
    bool TerminoModo(int ganador, List<int> puntosAcumulados);
    void TerminoUnaPratida(int ganador, List<int> puntosAcumulados);
    (int, int) GetGanador(bool EnEquipo);
}
```

  - Clases que implementan IModo
    - `Amistoso` : Se juega una sola vez. Gana el que se oegue o tenga menos fichas (ya sea en equipo o individualmenet).
    - `Match` : se juegan n partidas a ganar n - k partidas (n, k son enteros positivos, k < n).
    - `HatsaX` : Se juegan las partidas acumulando puntos hasta llegar a X puntos (X > 0). \

- IReglas

```
public interface IReglas : IAccionDespuesDeLaJugada, IFinalizarJugada,
IGanador,IProximoJugador, IRepartir, IValidarJugada
{
    bool equipo { get; }//dice si se esta jugando en equipo o no
    bool invertido{get;}//dice en que sentido se esta jugando(a favor o en
contra de las manesillas del relloj)
    int CantFichasTotalJuego();
    int CantFichasPorJugador();
    int JugadorInicial();
    int CantidadJugadores{ get; }
    (int,int) DimensionTablero { get; }
    IContarPuntos contarPuntos { get; }
    ICalculaPuntos calculaPuntos { get; }
}
```

Cada regla esta definida por el compotamiento de determinados aspectos del juego. Dichos aspectos estan encapsulados en interfaces que implementa IReglas y a su vez cada interfaz que encapsula un aspecto del juego esta implentada por clases referentes a este aspecto. Cada regla hereda de una clase comun general (`ClaseComunReglas`) contenedora de los argumentos comunes a las clases que implementan IReglas.

- Interfaces que definen tales comportamientos. (Dichas interfaces solo contienen un metodo por lo genral que describe el comportamineto de ese aspecto del juego)
  En la siguiente lista se veran las interfaces mencionadas y las clases que implementan dichas interfaces.
  - `IRepatir`
  - `IProximoJugador` :
    - *Aleatorio* : Escoge aleatoriamente.
    - *Clasico* : Escoge el sucesor a favor de las manesillas del reloj.
  - `IValidarJugada` :
    - *ValidarJugada_Clasica* : Si las fichas coinciden se puede jugar.
    - *ValidarJugada_Menor* : Si la parte de la ficha con que se va a jugar es menor o igual que la parte de la ficha ppor donde se quiere jugar entonces la jugada es valida.
    - *ValidarJugada_Mayor* : Si la parte de la ficha con que se va a jugar es mayor o igual que la parte de la ficha ppor donde se quiere jugar entonces la jugada es valida.
  - `IAccionDespuesDeLaJugada` : Define que medida tomar luego de cada jugada
    - *AccionDespuesDeLaJugada_Quincena* : Si al jugar las nuevas partes de las fichas por donde debera jugar el proximo jugador suman un multiplo de 5 ent al jugador actual de le suman los valores de las partes de las fichas por donde debara jugar el proximo jugador.
    - *AccionDespuesDeLaJugada_Clasico* : No hace nada.
    - *AccionDespuesDeLaJugada_InvertirJugadores* : Si el jugador actual no lleva se invierte el sentido del juego.
  - `IFinalizarPartida` :

        - *FinalizarJugada_Llegue100* : La partida se acaba cuando se acumulan 100 puntos por algun equipo o jugador.
        - *FinalizarJugada_Clasico* : La partida se acaba cuando alguien se pega o cuando la suma de los pases consecutivos es igual a la cantidad de jugadores.
    - `IContarPuntos` : Determina como se cuantan los puntos de las fichas que le daan puntos al jugador o equipo ganador.
        - *ContarPuntos_Clasico* : Cada ficha tiene su valor tradicional.
        - *ContarPuntos_DobleDoble* : Los dobles tienen el doble de si valor tradicional.
        - *ContarPuntos_ManoDura* : El valor de la mano de un jugador (las fichas que le quedan) se multiplica por la cantidad de fichas que tiene ese jugador
    - `ICalcularPuntos` : Determina cuales son las fichas que le daran puntos al jugador o equipo ganador.
        - *CalcularPuntosGanoJugador_Clasico* : Las fichas seran aquellas que no pertenezcan al jugador o integrantes del equipo ganador.
        - *CalcularPuntosGanoJugador_SoloYo* : Las fichas seran las que le quedan al ganador luego de haber terminado la partida.
        - *CalcularPuntosGanoJugador_Comunista* : Las fichas de todos los jugadores daran puntos al ganador, pero este solo se quedara con la cuarta parte de los puntos acumulados.
        - *CalcularPuntosGanoJugador_Capitalista* : Todas las fichas cuantan.
        - *Quincena* : Los puntos de todos cuentan. Si la suma
          \

- IEstrategias : Las clases que implementan estrategias contiene un metodo que de un conjunto de fichas elige cual es la mejor candidata para la jugada actual una vez luego de haberse hecho la primera jugada en la partida.

```
public interface IEstrategias
{
    (Ficha, int) Jugar(ref List<Ficha> Mano, ParteFicha izquierda,
ParteFicha derecha, IReglas reglas, int jugadorActual);//devuelve una ficha
y 0 si juega pos la primera ficha q recibe y 1 si juega x la 2da ficha q
recibe
}
```

- Clases que implentan IEstrategias :
    - `EAleatorio` : Escoge una ficha de forma aleatoria.
    - `EBotagorda` : Escoge la ficha de mayor valor para jugar.
    - `EHumano` : Usuario.
    - `ELeo` : Juga en dependencia de la ficha que mas lleve.
    - `EMatematico` : Juega de forma tal que en la mesa la suma de las fichas por donde se puede jugar es un multiplo de 5. En caso de no poder lograr esto juega como Botagorda.
    - `Pasador` : Juega fijandose en el historial de los pases que se han dado en el juego y trata de pasar su secesor jugador.
      \

- EstrategiasSalir

```
public interface IEstrategiasSalir
{
    Ficha Jugar(ref List<Ficha> Mano, IReglas reglas);//devuelve una ficha y
0 si juega pos la primera ficha q recibe y si juega x la 2da ficha q recibe
}
```

- Clases que implentan IEstrategias :
  - `ESBotagorda` : Sale con la ficha de mayor valor.
  - `ESMatematico` : Sale con la ficha que mayor multiplo de 5 sumen sus valores por partes.
  - `ESLeo` : La que mas tenga.
  - \

- IJuegaConMesa : En dependecia del historial de jugadas decide la forma de jugar

```
public interface IJuegaConMesa
{
    public void juegaConMesa(Mesa mesa, (Ficha, int) ultimaJugada,bool
huboJugada);
}
```

- Clases que implementan IJuegaConMesa : Son los jugadores.
  - *Pasador* : Trata de pasar al proximo jugador.
    \

Clases

- `Arbitro` : Cada partida necesita un `Arbitro` y su vez el arbitro necesita unas regls y las coleccion de jugadores(y otros datos) para poder desarrollar la partida.
  - Metodos :
    - `Jugar`

      ```
      public Ficha Jugar(bool esLaPrimeraJugada)
      ```

      Devuelve la ficha que va juagr el jugador actual(esta solo se utiliza para saber si se paso o no).
      De forma general elige al proximo jugador, juega, pone la ficha en la mesa y actualiza la fichas por donde puede jugar el proximo jugador. Luego llama al metodo `AccionDespuesDeLaJugada()` de reglas.

    - `TerminoPartida`

      ```
      public bool TerminoPartida()
      ```

      Dice si se acabo la partida o no(depende de las reglas).

    - `GetGanador`

      ```
      public (int, List<int>) GetGanador()
      ```

Da el ganador y una lista con los puntos acumulados por cada jugador en la partida actual.
\

Tenemos implementadas unas estructuras que son la base del juego :

- `Ficha` : Clase que encapsula la el concepto de ficha. Cuenta con 2 ParteFicha y un valor. El valor representa el valor de las fichas. Tiene tambien un nombre. Redefine el metodo `ToString()` para poder ser impresas en consola.
    - `ParteFicha` : Clase que tiene un argumento que determina el valor de esa parte de la ficha y un argumento parte. Redefine el metodo `Equals()`.
- `ParametrosDefinenGanador` : Enumerable que identica los parametros necesarios para determinar el ganador de una partida.
\

Consola

La aplicacion de consola es la encargada arrancar el proyecto. Basandose en la interaccion con el usuario crea el modo, las reglas y los jugadores. Permite al usuario modificar el juego (esto se hace por cada aspecto del juego que se pueda personalizar).

Luego de creados el modo y el arbitro se comienza la primera partida del modo. Por cada partida se llama al metodo `Jugar` del arbitro mientras la partida no haya terminado. Al terminar cada partida se actualiza el estado del modo permitiendo así que si no son necesarias más partidas el modo se finalice y de el ganador.
\

Ciclo de flujo del programa

Antes de comenzar una partaida es necesario especificar cada aspecto personalizable de esta; el modo, las reglas, las estrategias de cada jugador. Esto se hace en la aplicación de consola, donde luego de elegir el número de jugadores y decidir si se juega en equipo mediente funciones sencillas, se crea el modo de juego y comienza el mismo; desarrollandose en el método

```
public static void DesarrollarModo(IModo modo, bool equipo){...}
```

`Método`

```
DesarrollarModo (IModo modo, bool equipo)
```

Debido a la variedad en número de los juegos necesarios para terminar un `IModo`, cada paratida se crea y desarrolla de forma cíclica hasta llegar a la cantidad de partidas requeridas por el modo.

```
...
 while (!modo.TerminoModo("jugador actual", "puntos acumulados por cada
jugador"))
 {
        IDomino domino = IniciaDomino();//Crea las fichas con las que el usuario
desea jugar
        Arbitro arbitro = CrearArbitro(modo.CantidadJugadores, domino,
equipo);//(1)
        ...
 }
```

(1)

```
#region Crear Arbitro
    static Arbitro CrearArbitro(int cantJugadores, IDomino domino, bool
EnEquipo)//crear un arbitro
    {
        ...
        IReglas reglas = IniciaRegla(cantJugadores, domino, EnEquipo,
FichasDomino);
        List<Jugador> jugadores = IniciaJugadores(cantJugadores, reglas, domino,
FichasDomino);
        return new Arbitro(cantJugadores, EnEquipo, reglas,
jugadores);//provisional
    }
#endregion
```

En el método anterior se inicializan las reglas a utilizar en el juego y las estrategias de cada jugador mediante funciones que tienen como principal componente el fragmento de código siguiente, donde `creando` es un IEnumerable que contiene un objeto de cada tipo que implementa la interfaz que encapsula el aspecto a personalizar en cuestión. Luego, en dependencia de la selección del usuario se escoge un objeto u otro :

```
Console.WriteLine("Aspecto a personalizar");
var creando = from t in Assembly.GetAssembly(typeof(Interfaz que encapsula el
aspecto a modificar)).GetTypes()
                                    where
t.GetInterfaces().Contains(typeof(Interfaz que encapsula el aspecto a
modificar))
                                    && t.GetConstructor(Type.EmptyTypes) != null
                                    select Activator.CreateInstance(t) as
(Interfaz que encapsula el aspecto a modificar);
  int x = 1;
        foreach (var item in creando)
        {

            System.Console.WriteLine(x+". "+item.ToString()
[(eliminar.Length+".Ganador_".Length)..]);
            x++;
        }
        ``(Interfaz que encapsula el aspecto a modificar)`` ganador = new
``(Interfaz que encapsula el aspecto a modificar)``();
        string resp = Console.ReadLine();
```

```
        try
        {
            int r = int.Parse(resp);
            int index = 0;
                foreach (var item in creando)
                {
                    if(index == r) break;
                    ganador = item;
                    index ++;
                }
        }
        catch { }
```

\

```
DesarrollarModo (IModo modo, bool equipo) (continuacion)
```

Luego de creado el arbitro se comienza la partida

```
    while(!arbitro.TerminoPartida())
        {
            System.Console.WriteLine();
            Ficha fichaActual = arbitro.Jugar(numJugada);
            if(fichaActual is null)
                System.Console.WriteLine("El jugador "+arbitro.GetJugadores()
[arbitro.JugadorActual].nombre+" no lleva.");
            else
            {
                System.Console.WriteLine("Jugó el jugador {0} ",
arbitro.GetJugadores()[arbitro.JugadorActual].nombre);//dice el jugador que va a
jugar
                arbitro.ImprimirMesa();
            }
            numJugada = false;

                ...
            }
```

Para saber si una partida ha acabado, el arbitro poseedor de las reglas, verifica si en estas la partida sigue o se finaliza.
En caso de continuar la partida el arbitro es el encargado de elegir el proximo jugador (apoyandose en las reglas) y de brindar la ficha jugada por el mismo.
Esto se hace en el método

```
public Ficha Jugar(bool esLaPrimeraJugada)
```

de la clase `Arbitro`. En este método se le pide a las reglas el próximo jugador y se le ordena a este que juegue. Luego en dependecia de la ficha a poner en mesa esta se reordena para poder darle inequívoca ubicación en `Mesa`.
\

Por convenio cuando la ficha entregada por el jugador para poner en la mesa es `null` se asume que el jugador `no lleva`, hecho que se muestra en pantalla, en caso contrario se muestra la ficha seleccionada por el jugador.
\
Luego se verifica la acción a realizar en pos de la jugada en dependencia del estado del juego y se retorna la ficha que se puso en la mesa en ese turno.
\

Al terminar la partida se muestran las fichas que no fueron jugadas pertenecientes a cada jugador y se anuncia el resultado de la partida (ganador y puntos).

`Este ciclo se repite hasta que no queden partidas por jugar.`

Luego se anuncian los resultados del `Modo` (ganador y puntos) y concluye el juego.