2do Proyecto de Programación

Integrantes:

Maday Mircia Suarez Velázquez CC-211

Kevin Majim Ortega Alvarez CC-212

El objetivo de este PDF es explicar la estructura de clases detrás del proyecto dominó, todas las variaciones hechas y la motivación que dió origen a cada una de ellas. No trataremos el tema de cómo usarlo puesto que este punto se toca en otro documento. Sin más dilatación comienzo.

El proyecto cuenta con varias bibliotecas de clases en las cuales están recogidos los puntos más importantes del proyecto. Estas bibliotecas son (cada una se explicara con más detalles abajo):

1. Interfaces
2. Pieces
3. Players
4. Table
5. Condiciones
6. Formas de juego
7. Repartir
8. Juego
9. Painter

1 Interfaces:

En esta biblioteca se guardan todas las interfaces que se usan en el proyecto. El resto de bibliotecas utilizan a esta como referencia para su funcionamiento. En principio no la necesitamos, pero encontramos ciertos problemas a la hora de variar, por tanto es más fácil agrupar las variaciones para su posterior uso. Cada parámetro usado en los métodos de las interfaces se explicaran en cada una de sus implementaciones, por tanto aquí solo haremos mención a ellos.

Las Interfaces usadas en esta biblioteca son:

- IPlayer<T>

Esta interface agrupa los métodos necesarios que cada nuevo jugador debe tener para que sea jugador los cuales son:

  - Puntuación

    Esta propiedad se encarga de devolver la suma de los valores de cada ficha que posea el jugador

  - Number

    La propiedad Number cuenta las fichas de la mano del jugador.

  - Strategy

    El método Strategy sería el encargado de variar la forma de jugar de cada uno de los jugadores. Recibe como parámetro un objeto de tipo IMesa<T>.

  - Add

    El método Add se encarga de llenar la mano de cada jugador. Se le da entrada un parámetro de tipo T.

  - Pase

    El pase le indicara al jugador que hacer cuando un jugador se pasa.

- IWin<T>

En esta interface se agrupan los métodos necesarios para futuras variaciones de la forma de ganar, los cuales son:

  - WinnerConditions

    Aquí viene la condición de victoria específica. Recibe dos parámetros, un int y un array de IPlayers, devuelve el índice del jugador vencedor.

  - Tranco

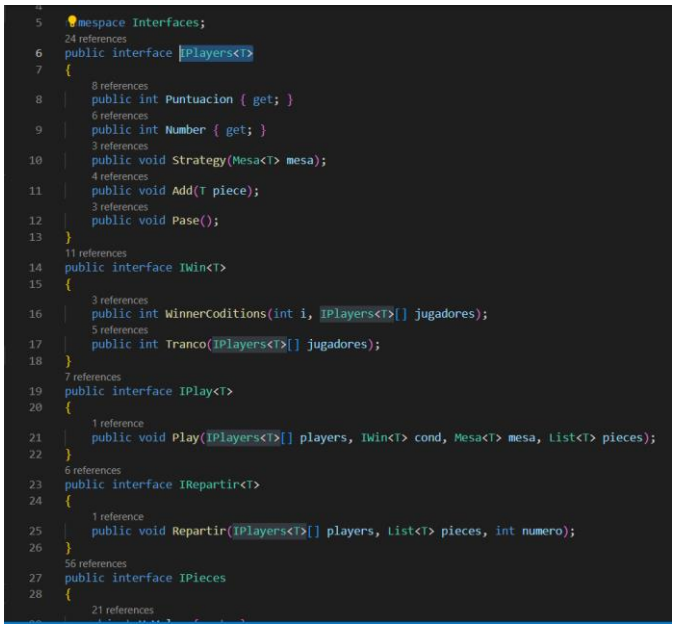
    Este método se encarga de calcular la puntuación si cada jugador se ha pasado al menos una vez. Recibe un array de IPlayers y devuelve un int con el índice del vencedor

- IPlay<T>

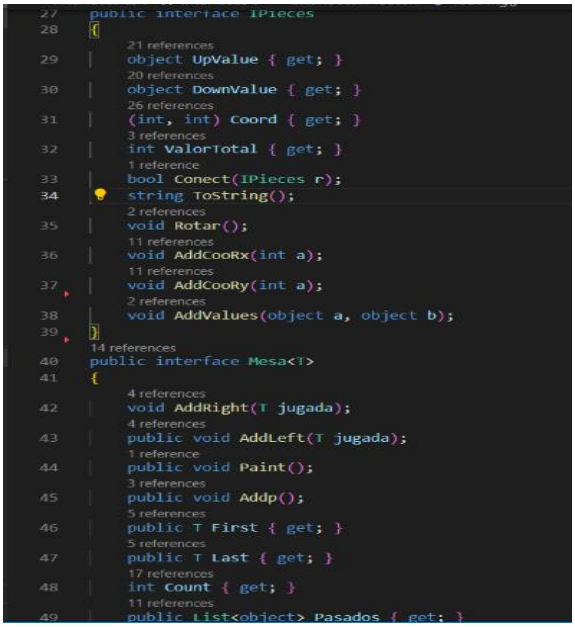
Esta interface se encarga de variar toda la forma de jugar. Posee un solo método; Play recibe como parámetros una IWin<T>, un array de IPlayers<T>, una IMesa<T> y una List<T>. Este método no tiene retorno

- IRepartir<T>

Su función es repartir las fichas a los jugadores y para eso tiene un método Repartir el cual recibe como parámetros un array de IPlayers<T>, una List<T> y un int.



```
mespace Interfaces;
24 references
public interface IPlayers<T>
{
    8 references
    public int Puntuacion { get; }
    6 references
    public int Number { get; }
    3 references
    public void Strategy(Mesa<T> mesa);
    4 references
    public void Add(T piece);
    3 references
    public void Pase();
}
11 references
public interface IWin<T>
{
    3 references
    public int WinnerCoditions(int i, IPlayers<T>[] jugadores);
    5 references
    public int Tranco(IPlayers<T>[] jugadores);
}
7 references
public interface IPlay<T>
{
    1 reference
    public void Play(IPlayers<T>[] players, IWin<T> cond, Mesa<T> mesa, List<T> pieces);
}
6 references
public interface IRepartir<T>
{
    1 reference
    public void Repartir(IPlayers<T>[] players, List<T> pieces, int numero);
}
56 references
public interface IPieces
{
    21 references
```



```
public interface IPieces
{
    21 references
    object UpValue { get; }
    20 references
    object DownValue { get; }
    26 references
    (int, int) Coord { get; }
    3 references
    int ValorTotal { get; }
    1 reference
    bool Conect(IPieces r);
    string ToString();
    2 references
    void Rotar();
    11 references
    void AddCooRx(int a);
    11 references
    void AddCooRy(int a);
    2 references
    void AddValues(object a, object b);
}
14 references
public interface Mesa<T>
{
    4 references
    void AddRight(T jugada);
    4 references
    public void AddLeft(T jugada);
    1 reference
    public void Paint();
    3 references
    public void Addp();
    5 references
    public T First { get; }
    5 references
    public T Last { get; }
    17 references
    int Count { get; }
    11 references
    public List<object> Pasados { get; }
```

- IPieces

Esta interface es de las mas importantes pues permite variar las fichas con las que vamos a jugar. Posee:

  - UpValue
  - DownValue

  Estas propiedades se encargan de devolver los valores de la ficha, tanto el de arriba como el de abajo

  - Coord

  Esta propiedad devolverá las coordenadas de las fichas para poder imprimirlas en la interfaz visual

- ValorTotal

  Esta propiedad calcula el valor total de las fichas, puesto que cada una debe tener un valor

- Conect

  Este método se encarga de ver si alguno de los valores de las fichas son iguales. Recibe como parámetro otra IPieces y devuelve un bool

- ToString

  Este método tal y como dice convierte las fichas en string para poder visualizarlas

- Rotar

  Este método intercambia los valores de la ficha permitiendo que encajen

- AddCoordx
- AddCoordy

  Estos metodos son los que le añaden las coordenadas a las fichas

- AddValues

  Este se encarga de añadir los valores

- IMesa<T>

Esta Interface no se hizo con el objetivo de añadir modificaciones al código, sino para que pudiera usarse el tablero en cualquiera del resto de las bibliotecas. Sus métodos son:

- AddRight
- AddLeft

  Estos metodos se encargan de añadir a la mesa los valores por la derecha o por la izquierda

- AddP

Este método se encarga de en caso de que un jugador se haya pasado, añadir a la lista si no los tiene, los valores por los cuales este se pasó

- First
- Last

Estas propiedades devuelven las fichas de más a la izquierda (derecha) de la mesa.

- Count

Devuelve la cantidad de fichas que hay en la mesa.

- Pasados

En esta lista se agrupan todos los valores por los cuales se han pasado los jugadores.

2 **Pieces:**

En esta biblioteca se encuentran las implementaciones del tipo IPieces. Todas las clases implementan de esa interface por tanto utilizan sus metodos.

Aquí podemos encontrar dos clases esenciales (las cuales corresponden a las variaciones de las fichas), la clase Fichas y la clase Letrichas, Ambas como implementan la misma interface pues utilizan los mismos métodos.

- UpValue
- DownValue
- Coord

Son Propiedades que se explican bien en la interface IPieces

- ValorTotal

Esta Propiedad es propia de cada implementación, puesto que fichas lo que tiene como valores son números por eso es fácil calcular ese valor total, pero en letrichas usamos el abecedario como referencia y le dimos valor a las letras según su posición en el abecedario.

- Conect

  Conect recibe un tipo IPieces y compara sus valores para así poder revisar cual es la ficha que más conexiones tiene. Es importante para uno de los jugadores de ahí su creación.

- ToString
- Rotar

  Este método cambia los valores de las fichas permitiendo que encajen unas con otras. El principal problema a resolver con este método es que si dos fichas encajan por los valores de arriba, necesito cambiar los valores de una, hacia abajo, para que el valor de la cambiada este debajo y pueda encajar con el valor de arriba y el juego continúe sin desperfectos.
- AddValues
- AddCoorx
- AddCoory

  Estos métodos se explican bien en la interface IPieces

3 **Players:**

En esta biblioteca se guarda todo lo referente a los jugadores del proyecto, cada implementación de ellos se ha hecho aquí. Fue una de las primeras en nacer puesto que para lograr que el proyecto funcionara de forma básica se necesitaba una interpretación de los jugadores.

En ella encontramos una clase Players<T>, de la que heredaran todos los jugadores, la T será del tipo IPieces para su correcto funcionamiento. La clase posee además los métodos que luego necesitaran el resto de jugadores.

- Add

  Add recibe como parámetro un tipo T que será de IPieces y lo añade a la mano del jugador.

- ValidFirst
- ValidLast

Estos métodos reciben como parámetros una IMesa y un T, retornaran true si el T (el cual es un IPieces) se puede colocar en la mesa, para esto verifica si el valor de arriba encaja con el de abajo, en caso de encajar con el de arriba, se rota la ficha con el método Rotar de ella.

- Posibilities

Posibilities es una lista que se crea con todas las jugadas válidas que posee el jugador en el momento en que se llama. Recibe una IMesa para verificar las fichas de la mano con los valores de la mesa.

- Pase

Pase se encarga de llamar a la clase estática Painter y pasarle una ficha vacía. En esta clase se explicara el porqué de este llamado.

Pero como bien dijimos al principio cada jugador realiza esas funciones básicas, la verdadera diferencia de los jugadores está en el método Strategy el cual es diferente para cada uno. Este método recibe una IMesa para que los jugadores pongan la ficha en la mesa, ellos no pueden modificar los valores de la mesa, solo pueden jugar una ficha.

- RandomPlayer
  El random, en su estrategia, de todas las fichas que él puede jugar el elegirá una al azar y la jugara.
- LightPlayer
  La estrategia de este jugador es quedarse lo más blanco posible, o sea entre todas las que puede jugar el siempre jugara la que mayor valor tenga.
- TrollPlayer
  El troll, su función es siempre jugar la ficha por la que los jugadores se han pasado incluso el, no le importa; para ello, utiliza la lista Pasados de la mesa y busca la ficha que tenga el valor para jugarlo.
- BankPlayer
  La función de este jugador es nunca quedarse en blanco, o sea siempre poder jugar una ficha. Gracias al método conect de las fichas, el verifica cuál es la que más conexiones tiene y esa es la que juega.

Nota: Si usted quiere implementar un jugador tramposo y que haga forro, y además jugar con él, lo puede hacer, claro que esto no será justo para algunos, pero hay formas de juego que no son justas y aun así las hemos hecho. Queda bajo su responsabilidad utilizarlo o no.

4 **Table:**

En esta biblioteca guardamos la mesa, el lugar donde se almacenan las fichas jugadas y se le añaden las coordenadas para su posterior visualización. Para almacenar las fichas se ha usado una lista. La mesa posee los siguientes métodos y propiedades

- First
- Last
- Pasados

  De estas propiedades hablamos en la interface IMesa así que profundizaremos en los siguientes métodos
- AddLeft
- AddRight

  AddLeft y AddRight son los métodos más importantes de la mesa pues son los que permiten añadir una ficha, y dependiendo de la posición en la que se añadió, colocan sus coordenadas. Si es la primera ficha la ponen en la coordenada del centro, ya luego si se ponen a la derecha o la izquierda se suma uno o se resta uno a la coordenada anterior. Además llaman a la clase Painter, al método Add para guardar en su lista estos valores. Pero como todo el orden de las fichas va cambiando por eso también llaman al método rotar de la propia mesa para que este rote las piezas solo en la visualización de la aplicación.
- Paint

  Este método es simplemente para comprobar si las implementaciones estaban bien hechas antes de tener un visual.
- Rotar

Este método va a crear una nueva ficha con los valores intercambiados y las coordenadas que le corresponden para llevársela al Painter.

Como vemos la clase Painter tiene una gran función dentro del código, pero de ella, hablaremos después.

5 **Condiciones:**

La biblioteca condiciones guarda las implementaciones de las condiciones necesarias para la victoria, ella actúa como un juez así que si quieren poner el tramposo, aquí podemos rápidamente verificar y si hizo algo indebido pues pierde.

Los métodos se explicaron en la interface IWin por tanto aquí haremos énfasis en las modificaciones que se hicieron con estos métodos. Primero decir que WinnerConditions recibe un int con la posición del jugador que acaba de jugar y un array de IPlayers a los cuales verificar las reglas. Tranco solo recibe a los jugadores pues lo único que hace es contar la puntuación de estos.

- ClasicC
  La condición clásica si alguien se pegó gana, para esto se verifica si la propiedad number del jugador es 0, igual que si se tranca cuenta y el que menos puntuación tenga pues es el vencedor. Para todo esto es necesario los jugadores.
- Inverter
  Una nueva forma, en la que el que se pegue en realidad pierde, pues se cuentan los puntos de cada jugador, y aquel que mayor puntuación logre tener es el que gana, evidentemente el que se pegó tiene puntuación 0. Esta forma no le hace justicia a los LightPlayer pues ellos siempre botan la más grande, pero nadie ha dicho que el mundo es justo
- Pasado
  El pasado posee un método privado llamado TrancoI. Este método nos retorna como ganador al jugador con menos puntos.
  Pasado verifica si algún jugador tiene exactamente la misma cantidad de fichas que el que le sigue, puesto que si esto es cierto entonces se pasó, ya luego se llama a TrancoI el cual verifica quien es que menos

puntos tiene, si ese es el que se pasó pues entonces, se vuelve a contar con el método Tranco y el segundo con menos puntos gana. Si el que el paso no es el que menos puntos tiene entonces gana ese con menos puntos.

6 **Formas de juego:**

La biblioteca se encarga de variar las formas que se tiene de jugar. El método Play recibe como parámetros un IWin el cual se encarga de verificar la condición de victoria, un array de IPlayers con todos los jugadores que están en la partida, una IMesa que sera donde se almacenen las fichas del juego, y una lista con las piezas que quedaron después de repartir.

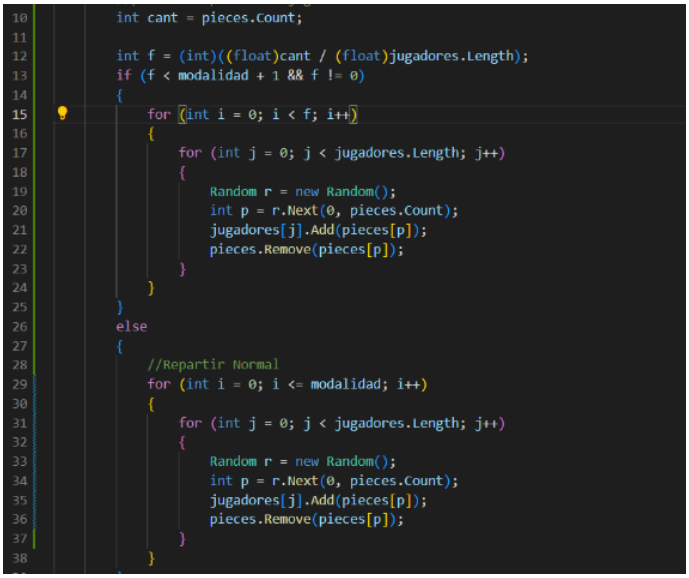
- ClassicF
  Esta forma es la clásica en la que cada jugador juega una sola vez cuando le toca, y si no lleva pues no juega. Cuando se le hace el llamado al método Strategy de los jugadores es cuando se juega. Para verificar si alguien se pasó se verifica que la mesa contenga la misma cantidad de fichas que antes del llamado al método Strategy
- KeepPlaying
  La forma KeepPlaying, como su nombre lo indica verificara que cada jugador si jugo pues seguirá jugando hasta que no lleve más, esto se logra con un método do while. Cuando se pase entonces le tocará al otro
- Thief
  La forma Thief es el clásico juego del robadito, en el cual, si un jugador no lleva ninguna de las fichas de la mesa, puede robar del resto que quedo al repartir, hasta que juegue una ficha.

7 **Repartir:**

En esta biblioteca se guardan todas las formas que se han implementado de repartir. El método recibe como parámetro las fichas que se van a repartir y los jugadores a los cuales se le van a repartir. Se han hecho dos variaciones:

- ClassicR

En esta forma se reparte de forma clásica, se reparten una ficha más que la modalidad que se va a jugar (ejemplo: si se juega la modalidad de 6, se reparten 7 fichas), claro está, que si añaden más jugadores entonces, se divide la cantidad de jugadores entre la cantidad de fichas que tenemos y si es menor a la cantidad que normalmente se reparte, entonces reparte esa nueva cantidad



```
int cant = pieces.Count;

int f = (int)((float)cant / (float)jugadores.Length);
if (f < modalidad + 1 && f != 0)
{
    for (int i = 0; i < f; i++)
    {
        for (int j = 0; j < jugadores.Length; j++)
        {
            Random r = new Random();
            int p = r.Next(0, pieces.Count);
            jugadores[j].Add(pieces[p]);
            pieces.Remove(pieces[p]);
        }
    }
}
else
{
    //Repartir Normal
    for (int i = 0; i <= modalidad; i++)
    {
        for (int j = 0; j < jugadores.Length; j++)
        {
            Random r = new Random();
            int p = r.Next(0, pieces.Count);
            jugadores[j].Add(pieces[p]);
            pieces.Remove(pieces[p]);
        }
    }
```

- All

  En esta forma de repartir, se reparten todas las fichas aleatoriamente, puede ser que a algún jugador le toque jugar con mas fichas que a otros. No tiene mucho sentido combinar esta forma de repartir con el Thief de los juegos puesto que no quedarán fichas para robar si alguien se pasa (Igual si quieren jugar así no habrá ningún problema)

8 **Juego:**

En esta biblioteca se agrupa todo lo que una vez iniciado la aplicación se instancia, con la finalidad de que a partir de aquí sea que empiece a correr. Posee dos clases una que llamamos Judge y otra a la que llamamos domino.

- Judge

  Judge al inicio no lo necesitamos, de hecho esta clase es n poco prescindible pues lo único que hace es agrupar todas las condiciones y entrar al constructor de Domino, así que su finalidad es sencillamente esa, no tener muchos parámetros sueltos.

- Domino

  Al inicio domino se encargaba de todo pero a medida que fuimos ampliando las modificaciones y nos dimos cuenta de cuantas variaciones fuimos haciendo, esta clase se fue haciendo cada vez más

pequeña. Recibe en su constructor un int modalidad, un Judge, y un array de IPlayers. Su método Play lo que hace es llamar al método Play de la interface IPlay que posee el Judge. Repartir tambien busca la forma de repartir que posee el Judge. Paint Es un método de comprobación, gracias a este método se fue comprobando todas las implementaciones.

9 **Painter:**

Esta biblioteca guarda una clase estática fundamental a la hora de unir el motor (en si todo el juego) con la parte visual, puesto que esta clase posee dos listas en las cuales se van guardando las fichas jugadas y sus coordenadas, y otra en la que se almacena el estado del juego (Quien se pasó, quien jugó y quien ganó) todo gracias a los métodos:

- Add
  Add añade una ficha a la lista de las fichas, si la ficha esta vacia eso indica que ese jugador se pasó
- Finish
  Finish guarda en la lista de string una oración con el jugador que ha ganado
- Devolver
  Este método devuelve las listas una vez estén actualizadas
- Actualización
  Actualización guarda en la lista de string una oración con el jugador que jugó o una oración con el jugador que se ha pasado si encuentra una ficha vacía.