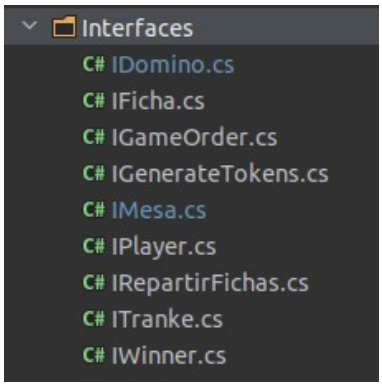
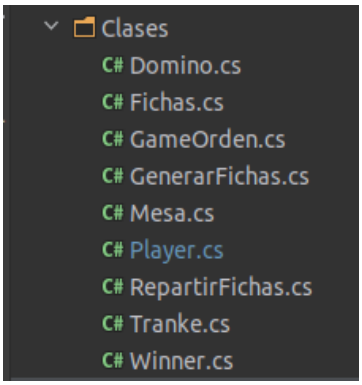
Aquí les dejamos un proyecto que tiene como objetivo modelar un juego de Dominó lo suficientemente flexible como para que se pueda crear el dominó más extravagante que se pueda imaginar. Para esto nos hemos valido de las interfaces que hemos creado y varias clases en las cuales hemos implementado estas interfaces. Agrupandolo todo en nuestras clases de Dominó

Sobre las Interfaces y Clases:





♦ La 1era interfaz que se ha creado es la interfaz IFicha.cs. En ella se define de forma abstracta lo que se entiende por Ficha en el Domino

```
namespace matcom_domino.Interfaces;

public interface IFichas<T>
{
   T GetFace(int a);
   int ValueFace(T a);
   int FichaValue();
}
```

Con nuestra interfaz quisimos definir lo que representa una ficha en el dominó,pues la ficha nos da la información de lo que hay en sus caras y sus respectivos valores en caso de que se tranque el juego. La clase que implementa esta interfaz es la de Ficha9. En la que GetFace(int a) devuelve lo que ocupa esa posición de la ficha en este caso debolbería un int pues se trabajó con el tipo int ,ValueFace(T a) devuelve el valor de esa cara y FichaValue() devuelve la suma del valor de las 2 caras juntas. Esto básicamente se usa para calcular la puntuación de un jugador por sus fichas en mano o las que halla jugado,cambiar la manera en la que se puede hacer corresponder a una ficha con la otra entre otras funcionalidades.

♦ Por otra parte tenemos a la interfaz IMesa.cs posee los metodos y propiedades propios de una mesa de dominó pues ,qué es lo que sucede en la mesa?

```
namespace matcom_domino.Interfaces;

public interface IMesa<T>
{

   bool IsValido(IFichas<T> a);
   List<IFichas<T>> CardinTable { get; }
   void RecibirJugada(IFichas<T> ficha, int side);
   List<string> Log { get; }
   List<IFichas<T>> FichasSobrantes { get; set; }
   IFichas<T> fichaJugable { get; }
}
```

Esta determina la manera en la que se hace corresponder una ficha con otra,posee un metodo en el cual ella organiza la ficha que el jugador decide poner en la mesa pues no se puede poner una ficha valida por la deracha en el lado izquierdo si este es invalidó,la mesa también posee las fichas que se han ido jugando durante el desarrollo del juego y las fichas sobrantes al repartirles las fichas a los jugadores(las que se ponen en la esquinita),y por ultimo posee una ficha jugable que no es más que las dos caras de las esquinas en una sola ficha ,teniendo un jugador cualquiera de las cara de esta ficha ,es jugable.

[3.4]-[4.6]-[6.6] =>ficha jugable [3.6]

De esta interfaz tenemos dos implementaciones. Una clase Mesa la cual se refiere al domino clasico ,y la clase MesaDobleSupremo en la cual cualquie ficha con las dos caras iguales siempre es valida ej:

[2.3]-[3.7]-[8.8]

♦ Contamos con la interfaz IGenerarFichas.cs crear el conjunto de fichas a jugar como desçseamos ,en nuestro caso implementamos dos clases las,una la cual genera todas las posibles fichas hasta el numero que usted desea y otra que te genera todas las posibles fichas hasta el numero que usted desea pero en las cara de las fichas van a haber numeros primos

```
.namespace matcom_domino.Interfaces;
public interface IGenerarFichas<T>
{
   List<IFichas<T>> GenerateCards(int a);

}
```

♦ La interfaz IGameOrden nos permite editar el orden del juego ,pues esta edita la lista de los jugadores ingresados al comiezo de la partida,hemos adoptado por crear dos estilos de juego ,en orden y en reversa.

```
namespace matcom_domino.Interfaces;

public interface IGameOrden<T>
{
   void OrdendelJuego(List<IPlayer<T>> jugadores);
}
```

♦ La interfaz IrepartirFichas esta creada con la idea de que las fichas creadas ustedes la reparta

```
namespace matcom_domino.Interfaces;

public interface IRepartirFichas<T>
{
   void Repartir(List<IFichas<T>> AllTokens, List<IPlayer<T>> PlayerList, int TokenQty);
}
```

a los jugadores de la manera que desee , nosotros creamos una repartición random , es decir, que a cada jugadador se le asigne un numero ramdon de fichas;también tenemos una repartición por orden esto significa que las fichas que le tocaron a cada jugador sean de una misma data prácticamente Ej:

```
fichas de jugador 1:[0.0][0.1][0.2][0.3]
fichas de jugador 2:[0.4][0.5][0.6][0.7]
fichas del jugador 3:[0.8][0.9][1.1][1.2]
```

y otro tipo de repartición donde solo se asignan fichas cuya suma suma valor sea

par .

♦ La interfaz ITranque genera un método en el cual sabemos cuando el juego se tranca

```
namespace matcom_domino.Interfaces;

public interface ITranke<T>

{
   bool Tranke(List<IPlayer<T>> PlayersList, IMesa<T> table);
}
```

en nuestras implementaciones esta el TranqueClásico que es cuando ningún jugador posee una ficha valida para jugar y también tenemos el DobleTranque la cual da posibilidad de que el juego se tranque cuando el hay un jugador que se paso dos veces.

♦ La interfaz IWinnner nos permite saber cual es el ganador del juego una ves que este finalice:

```
namespace matcom_domino.Interfaces;

public interface IWinner<T>
{
   IPlayer<T> PlayerWinner(List<IPlayer<T>> PlayerList);
}
```

en nuestras implementaciones se encuentra que el ganador es el que posea menos puntos en sus fichas restantes (HighScoreWinner) y el de mayor cantidad de puntos en sus fichas restantes(LowScoreWinner)

♦♦ La siguiente interfaz es la mas importante de todo el proyecto, la interfaz IDomino. Se dice esto porque es la que se encarga de desarrollar todo el juego. Para implementar esta interfaz se han desarrollado 2 clases DominoClasico y DominoRobaito. El constructor de estas clases recibe la mesa, ademas de que se indica hasta que data se desea generar las fichas, de que forma se generan, como se reparten a cada jugador y cuantas se reparten a cada uno de ellos. Se especifica con la interfaz ITranke cuando es que se tranca el juego, con la interfaz IWinner quien es el ganador del juego y el orden del juego. El constructor de la clase recibe todos estos parámetros para así garantizar las variaciones del juego.

```
namespace matcom_domino.Interfaces;

public interface IDomino<T>
{

   List<IFichas<T>> ConjuntodeFichas { get; }
   List<IPlayer<T>> Jugadores { get; }
   void Robar();
   ITranke<T> _tranke { get; }
   void AgregarJugador(IPlayer<T> a);
   IRepartirFichas<T> _repartirFichas { get; }
   void RepartirFichas(int k);

   IGenerarFichas<T> _generarFichas { get; }
   IGameOrden<int> orden { get; }
   void StartGame();
```

```
    bool EndGame();

    IWinner<T> Winner { get; set; }
}
```

El DominoClasico es el juego clásico que se conoce y el DominoRobaito es en el cual si algún jugador se pasa (no lleva) roba de la pila hasta que pueda jugar una ficha que robo. En el caso que se acaben las fichas de la pila le robara fichas al jugador que le toque a continuación. Esto se hace para garantizar el fin del juego sin que halla tranque ya que siempre alguien se va a quedar sin fichas.

♦♦ La Interfaz IPlayer.cs esta interfaz se encarga de definir el comportamiento de todos los jugadores de la siguiente forma.

```
namespace matcom_domino.Interfaces;

public interface IPlayer<T>
{
    // Campo que acumula la puntuacion del player
    int player_score { get; set; }

    // Campo para saber si el jugador se paso
    bool Pasarse { get; set; }

    // Lista que contiene las fichas del jugador
    List<IFichas<T>> ManoDeFichas { get; }

    // Decide como cada jugador selecciona la ficha a jugar
    void SelectCard();

    // Campo para saber si el jugador esta en turno
    public bool in_turn { get; set; }

    // Campo que da nombre al jugador
    string name { get; }

    // Metodo que juega una Ficha
    void Play(IFichas<T> ficha, int side = -1);

    // Campo que dice las veces que se ha pasado el jugador
    int time_passed { get; set; }
}
```

Como se puede apreciar en la imagen están comentado las funcionalidades de cada linea que aparece en la interfaz. Para implementar esta interfaz se ha creado una clase Player, que es lo que seria el Player que controla el usuario y otras clases que heredan de esta para especificar los comportamientos de cada tipo de jugador. Existen 4 tipos de jugadores diferentes:

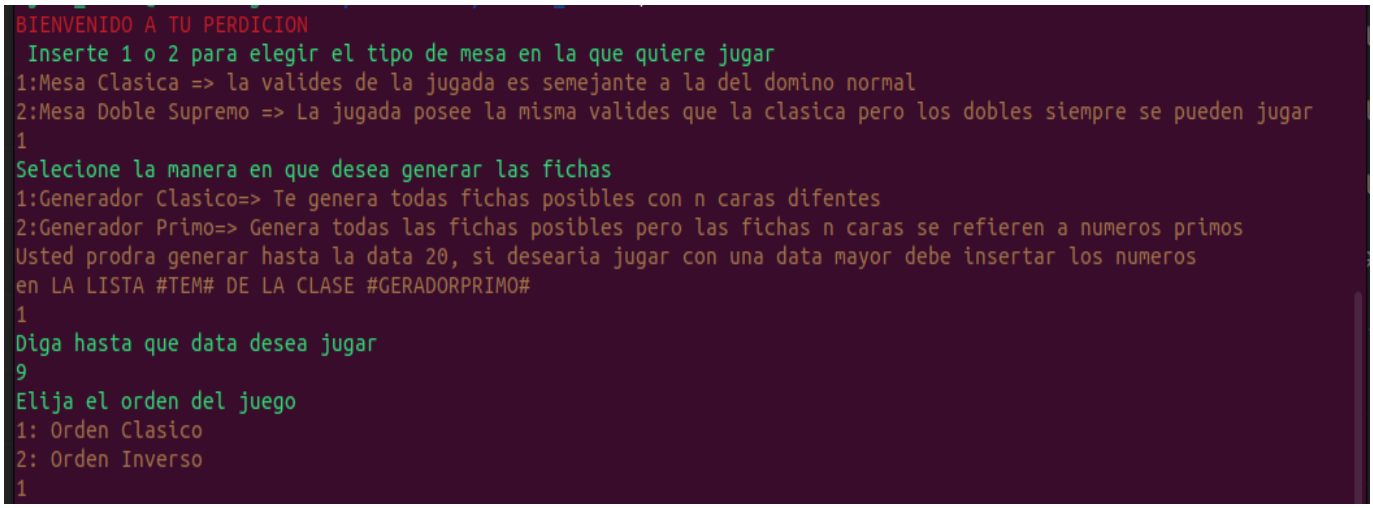
- El clásico bota gorda
- El jugador que juega una ficha aleatoria
- El jugador que juega la ficha que mas tenga (Para mantener todas las fichas posibles)
- El jugador supremo,entrenó en uno de los antiguos ¨tatami¨ de Japón,ha desarrollado un gran nivel de concentración,y por eso jugar contra el hace un poco difícil ganar.

Las interfaces restantes son las que le dan las variaciones al juego, ademas de las 2 mesas implementadas y los 2 tipos de domino. La interfaz ITranke

decide cuando se tranca el juego y de que forma según la implementación. La interfaz IGenerateTokens, decide la forma de generar las fichas. La interfaz IGameOrder, decide el orden del juego. La interfaz IWinner, decide el ganador del juego. La interfaz IRepartirFichas, decide la forma en que se reparten las fichas a los jugadores.

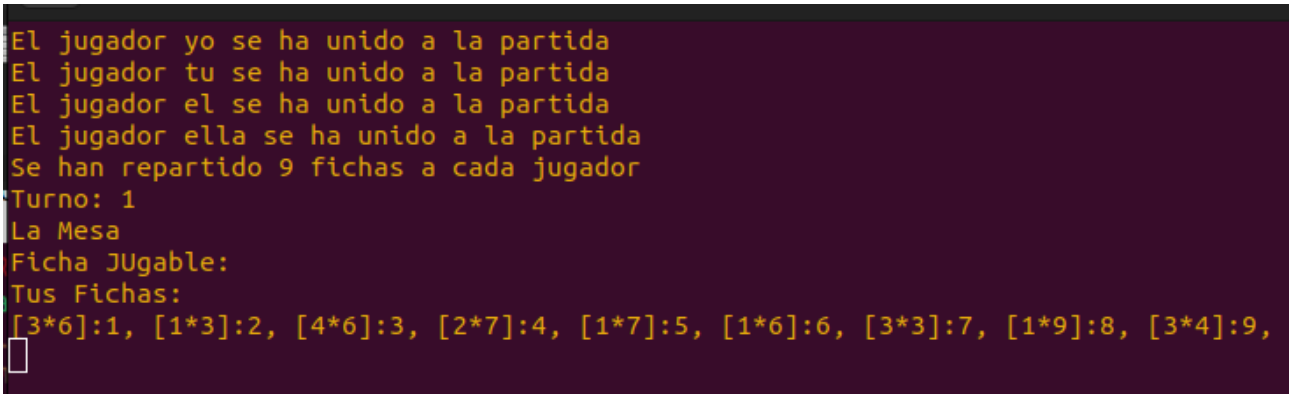
- Sobre la Interfaz de Usuario

Para este proyecto se ha decidido usar la consola básica para mostrar el juego, principalmente por falta de conocimiento. Aunque no es muy llamativa, se ha conseguido hacer llegar al usuario toda la información que necesita para jugar.



Al iniciar el juego se le indicara al usuario como configurar el juego de la forma que desee. Ingresando el número de la opción deseada y presionando enter podra avanzar en la configuración del juego hasta que este inicie.

Al comenzar la partida si elegiste controlar a algún jugador se mostrara así:



Cuando concluya la partida se mostrara todo el log del juego, para que el usuario pueda ver todo lo que ocurrió durante el juego. Si el usuario decidió no jugar, el juego se simulara automáticamente y se mostrara el log.

Conclusiones

Se cree que la aplicación cumple con el propósito del proyecto. Se han

creado 5 tipos diferentes de jugadores, 2 tipos de juego de domino, 2 tipos de mesas, 2 maneras de generar las fichas, 3 de repartir las mismas a los jugadores y 2 criterios de tranque del juego. Al crear interfaces para todos los objetos que se usan en el programa, se ha ganado la flexibilidad y sostenibilidad. Dando así capacidad a los futuros desarrolladores de crear el juego de domino mas extravagante que pueda ocurrir.