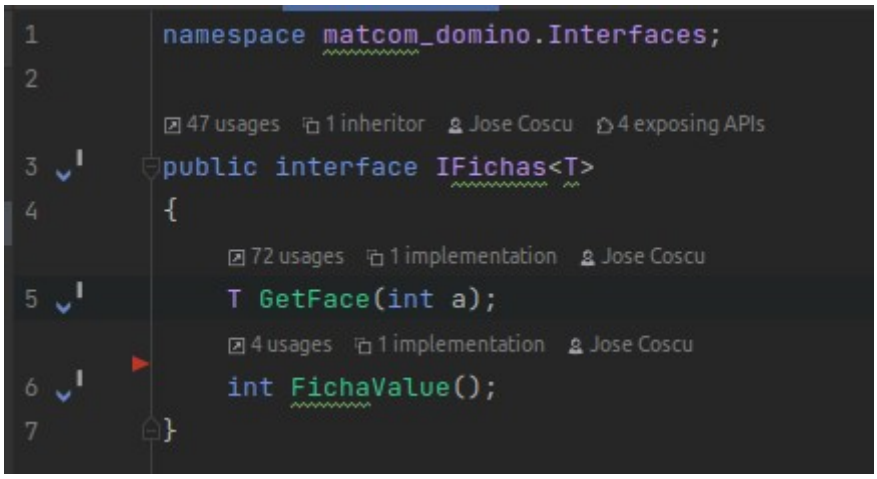
Introducción

Este proyecto tiene como objetivo modelar un juego de Domino lo suficientemente flexible como para que se pueda crear el domino mas extravagante que se pueda imaginar. Para esto nos hemos valido de las interfaces que ofrece C#.

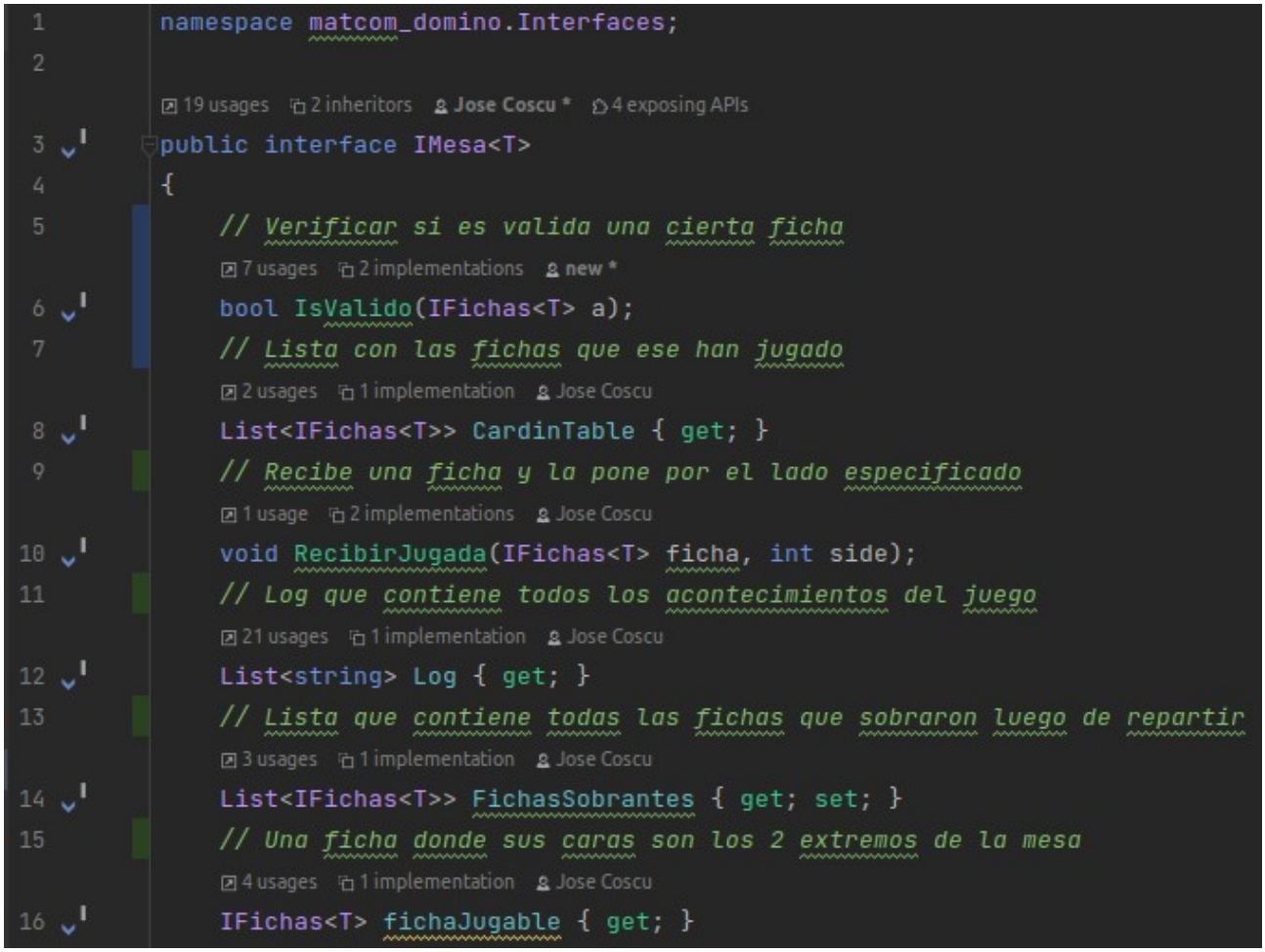
Sobre las Interfaces y Clases

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



La clase que implementa esta interfaz es la de Ficha9. En la que GetFace(int a) devuelve el valor de la cara indicada y FichaValue() devuelve la suma 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.

La interfaz IMesa.cs de igual forma se encarga de definir como va a comportarse todas las mesas que implementen esta interfaz. Para esto se ha implementado la clase Mesa (No te esperabas ese nombre ehh??). Esta clase, su función principal es verificar si una ficha cualquiera puede ser jugada en esta mesa con el método IsValid(IFicha ficha) que recibe la ficha a validar. Ademas tiene una lista con las fichas que se vallan tirando a la mesa, para mostrarlas al jugador y que este vea todo el desarrollo del juego. Contiene ademas un Log con toda la información de lo que va sucediendo en el juego. Esto sirve para mostrarle al jugador lo que hicieron sus compañeros jugadores. Tiene también una ficha jugable que sus caras son los dos extremos de la mesa. Un método RecibirJugada que acomoda la ficha que recibe en la mesa especificando por que lado se quiere pone y por ultimo las fichas sobrantes que no se repartieron a ningún jugador. Existe otra implementación de mesa que es la clase MesaDobleSupremo, que hereda de la mesa anterior descrita pero que en esta implementación las fichas dobles siempre son validas.



```
namespace matcom_domino.Interfaces;

19 usages  2 inheritors  Jose Coscu *  4 exposing APIs
public interface IMesa<T>
{
    // Verificar si es valida una cierta ficha
    7 usages  2 implementations  new *
    bool IsValido(IFichas<T> a);
    // Lista con las fichas que ese han jugado
    2 usages  1 implementation  Jose Coscu
    List<IFichas<T>> CardinTable { get; }
    // Recibe una ficha y la pone por el lado especificado
    1 usage  2 implementations  Jose Coscu
    void RecibirJugada(IFichas<T> ficha, int side);
    // Log que contiene todos los acontecimientos del juego
    21 usages  1 implementation  Jose Coscu
    List<string> Log { get; }
    // Lista que contiene todas las fichas que sobraron luego de repartir
    3 usages  1 implementation  Jose Coscu
    List<IFichas<T>> FichasSobrantes { get; set; }
    // Una ficha donde sus caras son los 2 extremos de la mesa
    4 usages  1 implementation  Jose Coscu
    IFichas<T> fichaJugable { get; }
```

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 intefaz 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 parametros para asi garantizar las variaciones del juego.

El DominoClasico es el juego clasico 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.

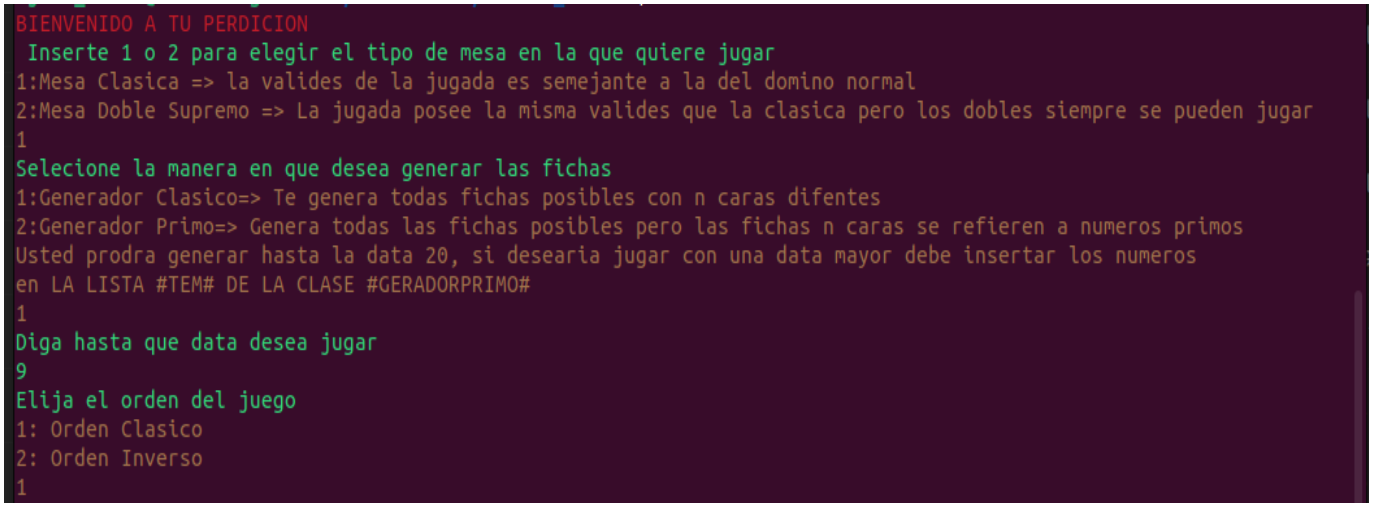
Como se puede apreciar en la imagen están comentado las funcionalidades de cada linea que aparece en la interfaz. Para implementar esta interfaz se ha creado una clase Player, que es lo que seria el Player que controla el usuario y otras clases que heredan de esta para especificar los comportamientos 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 que hace trampa. Este jugador siempre juega aunque no lleve.

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.

```
3   public interface IPlayer<T>
4   {
5       // Campo que acumula la puntuacion del player
            5 usages  1 implementation  Jose Coscu
6       int player_score { get; set; }
7
8       // Campo para saber si el jugador se paso
            4 usages  1 implementation  Jose Coscu
9       bool Pasarse { get; set; }
10
11      // Lista que contiene las fichas del jugador
            27 usages  1 implementation  Jose Coscu
12      List<IFichas<T>> ManoDeFichas { get; }
13
14      // Decide como cada jugador selecciona la ficha a jugar
            2 usages  5 implementations  Jose Coscu
15      void SelectCard();
16
17      // Campo para saber si el jugador esta en turno
            5 usages  1 implementation  Jose Coscu
18      public bool in_turn { get; set; }
19
20      // Campo que da nombre al jugador
            9 usages  1 implementation  Jose Coscu
21      string name { get; }
22
23      // Metodo que juega una Ficha
            5 usages  1 implementation  Jose Coscu
24      void Play(IFichas<T> ficha, int side = -1);
25
26      // Campo que dice las veces que se ha pasado el jugador
            1 usage  1 implementation  Jose Coscu
27      int time_passed { get; set; }
28  }
```

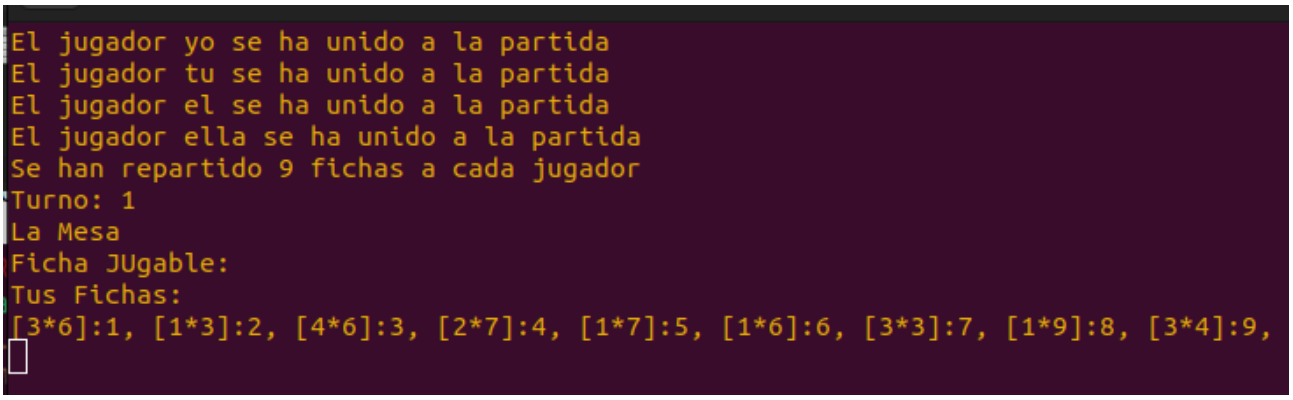
Sobre la Interfaz de Usuario

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 numero 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 tranke 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.