

Informe del proyecto de programación por parte de:

Alejandro Barrios Real

Bruno Jesús Pire Ricardo

Grupo: C-212

Este será el orden de los aspectos a tratar en el informe:

- 1. Estrategia de los jugadores***
- 2. Ficha***
- 3. Reglas***
- 4. El juego***
- 5. Interfaz Visual***

1era Parte: Estrategia de los jugadores

En la siguiente figura mostraremos como implementamos los jugadores y sus estrategias:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Dominó
8  {
9      //Interfaz que contiene un métodos para la funcionalidad del jugador
10     // 22 references
11     public interface IPlayer...
12
13     //Jugador Bota-Gorda
14     // 2 references
15     public class HighScoreDropperPlayer ...
16
17     //Jugador Bota-Flaca
18     // 2 references
19     public class LowScoreDropperPlayer ...
20
21     //Jugador Random
22     // 2 references
23     public class RandomPlayer ...
24 }
25
```

Lo primero que implementamos fue una **interface IPlayer** la cual tiene en su cuerpo una serie de propiedades que van a ser parte del jugador. Estas son:

```
9      //Interfaz que contiene un métodos para la funcionalidad del jugador
10     // 22 references
11     public interface IPlayer
12     {
13         //Este método indica la cantidad de veces que el jugador se ha pasado
14         // 6 references
15         public int GetContinuesTimesPassed...
16
17         //Este método devuelve el número del jugador
18         // 13 references
19         public int GetPlayerNumber...
20
21         //Este método devuelve las fichas de la mano del jugador
22         // 12 references
23         public List<Token> GetHand...
24
25         //Este método recibe como parámetro al campo con las fichas
26         // 4 references
27         Token Play(List<Token> field);
28 }
29
```

- I. GetContinuesTimesPassed, esta propiedad se encarga de decirnos cuantas veces el jugador se ha pasado

- II. `GetPlayerNumber`, bien, pues todos los jugadores tienen identificadores, en nuestro caso van a ser números, cada jugador va a tener un número designado. (Ej: player 1, player 2,)
- III. `GetHand`, esta propiedad es la encargada de dar acceso a la mano del jugador
- IV. `Token Play(List<Token> field)`, este método es el encargado de hacer jugar al jugador. Este recibe como parámetros una lista de tipo *Token* que será el tablero jugado hasta el momento y devolverá una ficha en caso de poder jugar bajo las condiciones del juego, de lo contrario pasará el turno

Como podemos apreciar en la figura anterior implementamos 3 tipos de jugadores:

- I. Bota-Gorda
- II. Bota-Flaca
- III. Random

Dentro de los jugadores, estos van a tener tres campos:

- I. `int continuesTimesPassed`
- II. `int playerNumber`
- III. `List<Token> hand`

Las funciones de estas propiedades serán asignadas por las propiedades vistas anteriormente en la *interface IPlayer* .

¿Cómo piensan nuestros jugadores?

El Bota-Gorda y el Bota-Flaca hacen prácticamente lo mismo, el primero lo que hace es buscar la ficha que puntúe mayor entre las que tiene disponibles para jugar y la juega y el segundo hace lo mismo, pero busca la que puntúa menor.

El Random como bien dice su nombre, lo que hace es que, de las fichas disponibles a jugar, escoge una al azar y la juega.

Estas implementaciones van dentro del método `public Token Play(List<Token> field)` de cada jugador.

Los 3 jugadores implementan los mismos métodos, donde único difieren es que el Bota-Gorda y el Bota-Flaca tienen un método más, el cual es `public static void SortHand(List<Token> hand)` que en el caso del Bota-Gorda ordena las fichas de mayor puntuación a menor y en el del Bota-flaca las ordena de menor a mayor. Esto es para que cuando vayan a jugar, recorran sus fichas y la primera que encuentren que puedan jugar, jueguen esa. Así en determinados casos recorren menos veces la mano.

`public void TurnsPassed()`, este método lo que hace es contabilizar cuando un jugador se pasa o no juega.

`public void AssignToken(Token token)`, este método reparte las fichas a la mano de los jugadores.

2da Parte: Ficha

En la siguiente figura mostraremos como implementamos Ficha y sus características:

```

10 public class Token
11 {
12     //esta va a tener solo dos valores o dos caras que es lo mismo
13     int leftBack;
14     int rightBack;
15
16
17     2 references
18     public Token(int leftBack, int rightBack) {...}
19
20
21     16 references
22     public int LeftBack {...}
23     16 references
24     public int RightBack {...}
25
26
27     //Score va a devolver la suma de los valores de la cara de la ficha
28     6 references
29     public int Score {...}
30
31
32     //Higher va a devolver el valor máximo entre los valores de la cara de la ficha
33     3 references
34     virtual public int Higher() {...}
35
36
37     //Lower va a devolver el valor mínimo entre los valores de la cara de la ficha
38     1 reference
39     virtual public int Lower() {...}
40
41
42     //Rotate va a intercambiar los valores de la cara de la ficha
43     6 references
44     public void Rotate() {...}
45
46
47     //Contains va a preguntar si cierto valor esta contenido en la ficha
48     3 references
49     virtual public bool Contains(int value) {...}
50
51
52     //ToString esta redefinido en este caso para que imprima la ficha con sus características
53     2 references
54     public override string ToString() {...}
55
56
57 }

```

Lo primero que implementamos fue **public class Token** que va a ser nuestro tipo Ficha, el cual va a tener 3 campos, un elemento izquierdo (por detrás), uno derecho (por detrás), los cuales van a ser asignados cuando se cree la ficha, y un score que va a ser la suma de ambos elementos. Ficha va tener una serie de propiedades que van a ser útiles dentro del juego y de la estrategia de los jugadores.

Estas son:

- I. **public int LeftBack**, esto dará un valor por detrás al elemento de la izquierda de la ficha

- II. `public int RightBack`, esto dará un valor por detrás al elemento de la izquierda de la ficha
- III. `public int Score`
- IV. `public int Higher()`
- V. `public int Lower()`
- VI. `public void Rotate()`
- VII. `public bool Contains(int value)`
- VIII. `public override string ToString()`

Desde el III hasta el VIII están explicados en los comentarios del código, los cuales voy a mostrar en la siguiente foto:

```
32 //Score va a devolver la suma de los valores de la cara de la ficha
33 6 references
34 public int Score
35 {
36     get { return leftBack + rightBack; }
37 }
38 //Higher va a devolver el valor máximo entre los valores de la cara de la ficha
39 2 references
40 public int Higher()
41 {
42     if (leftBack < rightBack) return rightBack;
43     return leftBack;
44 }
45 //Lower va a devolver el valor mínimo entre los valores de la cara de la ficha
46 0 references
47 public int Lower()
48 {
49     if (leftBack > rightBack) return rightBack;
50     return leftBack;
51 }
52 //Rotate va a intercambiar los valores de la cara de la ficha
53 6 references
54 public void Rotate()
55 {
56     int aux = leftBack;
57     leftBack = rightBack;
58     rightBack = aux;
59 }
60 //Contains va a preguntar si cierto valor esta contenido en la ficha
61 2 references
62 public bool Contains(int value)
63 {
64     return leftBack == value || rightBack == value;
65 }
66 //ToString esta redefinido en este caso para que imprima la ficha con sus valores
67 2 references
68 public override string ToString()
69 {
70     return "[" + leftBack + " | " + rightBack + " ]";
71 }
```

Luego implementamos un tipo Animal `public class Animal`, el cual la utilizamos para aportar variedad y tener al menos dos tipos de fichas en nuestro dominó. Mostraremos en la siguiente figura como esta implementado el tipo Animal:

```

80 //Aquí definimos la clase Animal
81 public class Animal
82 {
83     //este va a ser el nombre de dicho animal
84     string name;
85
86     public Animal(string name)
87     {
88         this.name = name;
89     }
90
91     //ToString esta redefinido en este caso para que devuelva el nombre del animal
92     public override string ToString()
93     {
94         return name;
95     }
96 }

```

Este tendrá un campo *name* el cual, como bien su nombre lo indica, va a ser el nombre de dicho animal. Va a tener una propiedad `public override string ToString()`, la cual la sobrescribimos para que cuando preguntemos por ella imprima el nombre del animal.

Ahora bien, cuando nosotros creamos las fichas decidimos ponerle un valor numérico por detrás a cada cara, de esa forma siempre que preguntáramos si era igual a la ficha en el campo, lo hacíamos por esos valores y no por los de la cara, de esta forma tenemos un sistema de valores únicos para cualquier tipo de ficha que se cree y no importa que valor tenga la cara, así sean colores, canciones o incluso fotos. También decidimos hacer dos variantes en la cantidad de fichas; una será con el tipo de ficha “doble 6” y el otro “doble 9”. Para el caso de los enteros en la primera variante van a ser números del 0 al 6 y en el segundo caso van a ser del 0 al 9. Para la ficha de animales creamos un array de tamaño 9 donde van a haber 10 tipos de animales distintos, y dado el caso que escojan una variante u otra con este tipo de fichas simplemente los valores de los animales van a estar dados por su posición en el array. (Ej: `guide = new Animal[] { new Animal("Lion"), new Animal("Snake"), new Animal("Dog") }`),

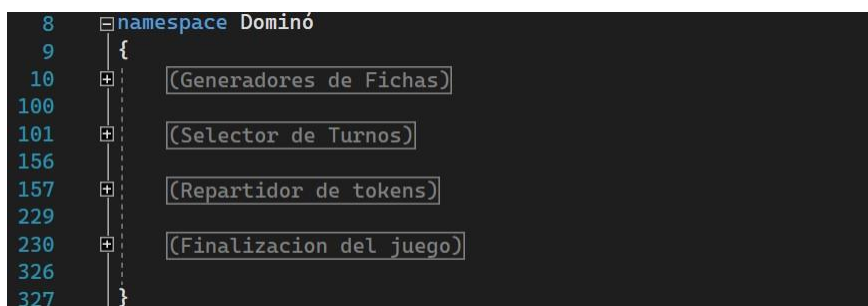
en este caso el “Lion” va a tener valor 0, el “Snake” valor 1 y el “Dog” valor 2)

Luego de haber explicado nuestros dos tipos principales, *Animal* y *Token*, y nuestra filosofía vamos explicar cómo se utilizarán en las fichas. Creamos dos tipos de fichas que heredan de *Token*:

- I. `public class InToken : Token`, la cual tendrá en sus caras valores dados por *LeftBack* y *RightBack* de *Token*. Para esta ficha como son valores de enteros utilizamos los mismos valores que guardan por detrás las caras, de esta forma en este tipo de ficha nos facilitábamos en trabajo.
- II. `public class AnimalToken : Token`, este tiene un campo `Animal[] guide`, el cual va a ser el array que explicamos previamente y una propiedad `public override string ToString()`, la cual esta redefinida para que imprima la ficha con los nombres de los animales.

3era Parte: Reglas

En la siguiente figura mostraremos cuales son las reglas de nuestro juego:



Ahora pues, cada regla tiene dos implementaciones, estas serán *interfaces* las cuales tendrán en su interior un método que será el que varíe para aportar las dos implementaciones de cada regla. A

continuación, explicaremos sobre que tratan y cuáles son las diferentes implementaciones de cada una.

- I. Generadores de Fichas va a ser la que genere según la elección del usuario las fichas. Estas pueden ser de enteros o de animales, y puede variar la cantidad de fichas, siendo estas 28 (doble 6) o 55 (doble 9). Dentro de esto lo más importante va a ser una *interface* `public interface TokenAmountGenerator` que en su interior va a contener un método `public List<Token> GenerateTokens()`, el cual va a devolver una lista según la elección del usuario, con todas las fichas del juego

```
12 //Interfaz que va a contener un método que generará una lista de fichas
13 7 references
14 public interface TokenAmountGenerator
15 {
16     5 references
17     public List<Token> GenerateTokens();
18 }
19
20 //Esta clase va a generar las fichas del tipo de juego "doble 6"
21 1 reference
22 public class DoubleSixInToken ...
23
24 //Esta clase va a generar las fichas del tipo de juego "doble 6" pero en vez de números van a ser animales
25 1 reference
26 public class DoubleSixAnimalToken ...
27
28 //Esta clase va a generar las fichas del tipo de juego "doble 9"
29 1 reference
30 public class DoubleNineInToken ...
31
32 //Esta clase va a generar las fichas del tipo de juego "doble 9" pero en vez de números van a ser animales
33 1 reference
34 public class DoubleNineAnimalToken ...
```

- II. Selector de Turnos va a ser la opción donde el usuario escoja si quiere que el primer jugador empiece de manera aleatoria y los que estén a su derecha en un sistema circular sean los siguientes en ese orden, o si quiere que cada jugador sea elegido aleatoriamente. Cabe destacar aquí la utilización de una *interface* `public interface TurnOrderSelector` la cual tendrá en su interior un método `public void GetTurnOrder(List<IPlayer> players)` el cual recibirá como parámetro la lista con los jugadores donde el orden de

esta será modificado según la elección del usuario

```
103 //Interfaz que va a contener un método que recibe como parámetro la lista de jugadores
104 public interface TurnOrderSelector
105 {
106     3 references
107     public void GetTurnOrder(List<IPlayer> players);
108 }
109 //Esta clase va a escoger el primer jugador random y los siguientes son los que esten a su derecha en un sistema circular
110 public class GetRandomFirstPlayer ...
113
114 //Esta clase va a escoger el orden de todos los jugadores de manera random
115 public class GetAllPlayersRandom ...
```

- III. Repartidor de Tokens(fichas) va a ser la opción que de al usuario a escoger si quiere repartir las fichas a los jugadores de manera aleatoria o si quiere repartir manualmente las fichas viéndoles sus caras, esta última opción puede valer en caso de que el usuario quiera probar ciertas estrategias entre los jugadores con determinadas fichas. La *interface* en este caso será **public interface TokensDistributor** y en su interior tendrá un método **public void DistributeTokens(List<Token> tokens, List<IPlayer> players)** el cual recibirá la lista con todas las fichas del juego y la lista con los jugadores para posteriormente asignarle a la mano de los jugadores sus fichas.

```
159 //Interfaz que va a contener un método que recibirá como parámetros una lista de jugadores y una lista con todas las fichas del juego
160 public interface TokensDistributor
161 {
162     3 references
163     public void DistributeTokens(List<Token> tokens, List<IPlayer> players);
164 }
165 //Esta clase va a repartir las fichas de manera aleatoria a los jugadores
166 public class RadomTokenDistributor ...
167
168 //Esta clase va a permitir que el usuario escoja que fichas asignar a cada jugador
169 public class SelectedTokenDistributor ...
```

- IV. Finalización del juego va a ser la opción donde el usuario debe escoger entre el método de finalización común, el cual es cuando un jugador se quede sin fichas (o se haya se pegado) o cuando todos los jugadores se pasen en una misma ronda o vuelta (o sea que se tranque el juego) y el método de finalización por expulsión, que es cuando un jugador se pasa dos veces seguidas

entonces es eliminado del juego y si el juego no se acaba antes por el método de finalización común entonces el último en quedar en juego es el que gana. La *interface* será **public interface GameEnd** cuyo interior contendrá un método **public int CheckIfTheGameIsOver(List<IPlayer> players)** que retornará un el numero del jugador victorios

```
233 //Interfaz que va a contener un método que recibe como parámetro una lista de jugadores
234 public interface GameEnd
235 {
236     3 references
237     public int CheckIfTheGameIsOver(List<IPlayer> players);
238 }
239 //Esta clase va a contener la finalización del juego común (cuando un jugador se quede sin fichas o se tranque el juego)
240 1 reference
241 public class CommonEnd ...
242
243 //Esta clase lo que hace es que si un jugador se pasó dos veces seguidas en el juego, este es eliminado, y el último en quedar es el ganador
244 1 reference
245 public class ExpelEnd ...
```

4ta Parte: El juego

```
3 //Esta clase va a contener un método encargado de moverse por el menú y seleccionar la opción deseada
4 4 references
5 public class Selector
6 {
7     10 references
8     public int FancySelector(string initialMessage, string[] states)...
9 }
10
11 //Esta clase va a contener la funcionalidad de juego con todos sus aspectos
12 3 references
13 public class Game
14 {
15     List<IPlayer> players;
16     TokenAmountGenerator tokenGenerator;
17     TurnOrderSelector turnSelector;
18     TokensDistributor tokenDistributor;
19     GameEnd gameEnd;
20     List<Token> allTokens;
21     List<Token> Table;
22     Token lastTokenPlayed;
23     bool gameOver;
24     int turn;
25
26     1 reference
27     public Game(TokenAmountGenerator tokenGenerator, TurnOrderSelector turnSelector, TokensDistributor tokenDistributor, GameEnd gameEnd, List<IPlayer> players)...
28
29     1 reference
30     public bool GameOver...
31
32     2 references
33     public string TableStatus()...
34
35     1 reference
36     public string PrintHand()...
37
38     1 reference
39     public string Run()...
```

El juego va a estar regido por una clase **public class Game** y una clase **public class Selector** esta última va a tener un método en su interior **public int FancySelector(string initialMessage, string[] states)** el cual va a recibir en sus parámetros un mensaje que va a ser todo lo que este impreso en consola en ese momento y unos *states* que son las opciones por las que el usuario se va a mover y

va a devolver un entero que va a ser el número de la opción en la que el usuario decidió entrar.

En la clase *Game* van a haber una serie de campos que son de los que el juego va a estar compuesto, estos son:

```
54 public class Game
55 {
56     List<IPlayer> players;
57     TokenAmountGenerator tokenGenerator;
58     TurnOrderSelector turnSelector;
59     TokensDistributor tokenDistributor;
60     GameEnder gameEnder;
61     List<Token> allTokens;
62     List<Token> Table;
63     Token lastTokenPlayed;
64     bool gameOver;
65     int turn;
```

- *allTokens* va a ser la lista donde se guarden todas las fichas del juego
- *Table* va a ser la lista con todas las fichas del tablero
- *lastTokenPlayed* va a ser la ficha jugada en cada momento por el jugador
- *gameOver* va a ser una variable booleana que cuando cambie de estado indicara que el juego se ha acabado
- *turn* va a indicar el turno en el que están

Dentro de las propiedades de *Game* encontramos:

- I. `public string TableStatus()` método encargado de imprimir en cada momento el tablero
- II. `public string PrintHand()` método encargado de imprimir en cada momento la mano de cada jugador
- III. `public bool GameOver` método encargado de indicar cuando se acaba el juego

- IV. `public string Run()` va a ser el encargado de devolver en cada momento el estado de la partida, o sea todo lo que le interfaz muestra en cada momento dentro del juego una vez iniciado.

4ta Parte: Interfaz Visual

La última parte de nuestro proyecto y no menos importante es la interfaz visual, esta va a estar contenida por:

- I. `class VisualInterface`
- II. `class Printer`

Empezando por el primer punto, *VisualInterface* va a estar compuesto por varios campos, estos son:

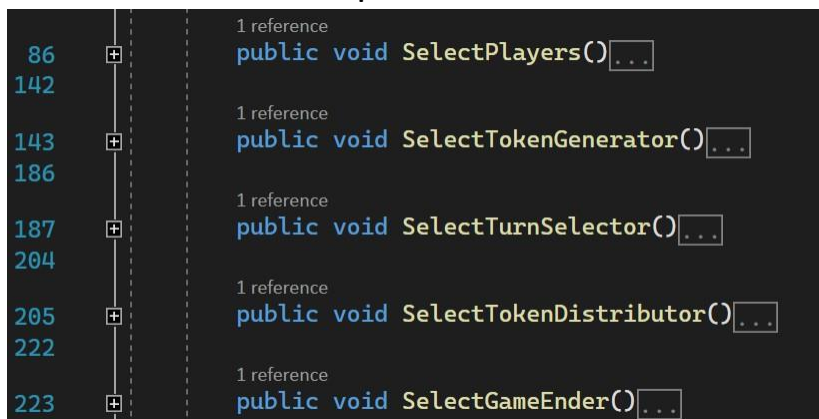
```
List<IPlayer> players = new List<IPlayer>();
TokenAmountGenerator tokenGenerator;
TurnOrderSelector turnSelector;
TokensDistributor tokensDistributor;
GameEnder gameEnder;
List<Token> allTokens;
Selector selector = new Selector();
string[] message = new string[6];
Game game;
```

El tamaño de *message* es 6 pues son la cantidad de opciones a escoger dentro del visual, como mostraremos en la siguiente figura

```
Lista de jugadores:
-
Tipo de ficha:
-
Tipo de juego:
-
Método para asignar los trunos:
-
Manera de repartir las fichas:
-
Finalización del juego:
-
```

También va a contener una serie de propiedades, que estas son:

- `public void RunVisual()` va a ser el encargado de correr, mientras el juego no se haya acabado, la propiedad *Run* de *Game* vista anteriormente
- `public void InitializeGame()` es la propiedad encargada de imprimir el cuadro de opciones antes de empezar el juego. Además de recoger las opciones que elige el usuario necesarias para iniciar el juego. Estas se archivan dentro de *message*
- A continuación, las siguientes propiedades cumplirán las mismas funciones en sus respectivas funcionalidades:



```
86  + 1 reference  
142 public void SelectPlayers()...  
  
143 + 1 reference  
186 public void SelectTokenGenerator()...  
  
187 + 1 reference  
204 public void SelectTurnSelector()...  
  
205 + 1 reference  
222 public void SelectTokenDistributor()...  
  
223 + 1 reference  
public void SelectGameEnder()...
```

una vez seleccionada la opción, moverse dentro y seleccionar una de las dos implementaciones de cada una.

- `public string PrintMessage()` propiedad encargada de darle sentido a la impresión de *message*, o sea, imprime una lista con las opciones del juego junto a la modalidad de cada una que escogió el usuario

Y por último *Printer* se encarga de correr el juego, o sea crear una instancia de *VisualInterface* y *correr el visual*.

¡Esperamos que haya sido de ayuda nuestro informe para el entendimiento de nuestro proyecto!