PROYECTO DE PROGRAMACIÓN II

de: Daniel Abad Fundora y Javier Rodríguez Sánchez

Facultad de Matemática y Computación

El dominó es un popular juego de mesa en el cual, un conjunto de jugadores se turnan para concatenar fichas bajo ciertas reglas.

Existen disímiles variantes de este juego, por ejemplo, en la zona oriental de nuestro país, el juego se compone de 28 fichas y 4 jugadores, tomando cada jugador 7 fichas al iniciar el juego. Por otro lado, en la capital, el juego se compone de 55 fichas, de las cuales se le asignan 10 a cada jugador y 15 permanecen boca abajo en la mesa.

Con este proyecto pretendemos simular distintos juegos de dominó, permitiendo variar las reglas y las diferentes posibles estrategias de los jugadores.

ESTRUCTURA:

Nuestro proyecto consiste en una aplicación web basada creada mediante el uso de Blazor, y utiliza como lenguajes HTML5, CSS y principalmente C#, sobre el cual se basa la parte lógica de nuestra aplicación, mediante un complejo diseño de clases agrupados en 4 namespaces:

-DominoGame: Agrupa el desarrollo lógico del juego. Contiene las clases Escena<T>, Game<T> y Torney<T>.

-DominoTable: Agrupa diferentes abstracciones del juego inherentes a cualquier tipo de juego. Contiene las clases Table<T>, Piece<T>, Pase<T> y Jugada<T>.

-DominoPlayer: Concentra lo relativo a un jugador y a la inteligencia artificial del juego. Contiene Player<T> y a las diferentes aplicaciones de la interfaz IEstrategia

-DominoRules: Aquí se regula las posibles variaciones de las reglas del juego, lo que permite diversificar la modalidad y flexibiliza nuestro código. Compuesto por la clase Rules<T>, DominoPieces<T> y las diferentes interfaces que permite el mismo.

El núcleo de nuestra aplicación viene dado en la clase Piece<T> como abstracción del concepto de ficha del juego, como un conjunto de valores de tipo T, y tiene como clase principal la clase Game<T> anteriormente mencionada, la cual representa una partida de dominó. Esta clase pertenece la interface IEnumerable<Escena<T>>, que permite recorrer un objeto de este tipo con un ciclo foreach y ver las escenas del juego. Escena<T> constituye el nexo entre Game<T> y la interfaz gráfica. Se trata de una "foto" del estado del juego con toda la información que se necesita para representarlo en pantalla.(Digase la mesa, las manos de los jugadores, los scores de los equipos, el jugador actual, etc).

Game<T> recibe como parámetros una clase de tipo Rules<T> y un array de Player<T> (con estructuras posteriormente explicadas)

La mesa del juego viene representada mediante la clase Table<T>, la cual memoriza el historial del juego en un IEnumerable de Jugada<T>(Encargada de guardar la información relacionada con cada jugada) y en otro de Pase<T>(Encargado de almacenar lo relacionado con cada pase ocurrido durante el juego), además de llevar la cuenta de las casillas que estan disponibles para jugar. Las fichas que no han sido repartidas(que no estan en las manos de los jugadores ni en la mesa) se guardan en DominoPieces<T>.

Con el objetivo de simular todo un torneo de Dominó(refierase a un conjunto de partidas concatenadas), se ha creado la clase Torney<T>, la cual implementa la interface IEnumerable<Game<T>>, que permite recorrer a través de los diferentes juegos de dominó para poder observar así todo el desarrollo del torneo y llevar scores para decidir el equipo ganador. Recibe los mismos parametros que Game<T>.

Player<T> representa a un jugador de dominó, este recibe como parámetro un string con el nombre del jugador, un int que representa el equipo del jugador y una interface IEstrategia con el algoritmo que seguirá el jugador a la hora de jugar. Esta interfaz posee el método Play que recibe un conjunto de datos del juego y devuelve la jugada ideal según la estategia.

Entre las diferentes implementaciones de IEstrategia esta:

-"El Borracho"(estrategiaBorracho<T>): Juega la primera ficha que ve

-"El BotaGordas"(estrategiaBotaGorda<T>): Evalua cada ficha ficha y juega la de mayor valor posible

-"El Agachado"(BotaMasRepetida<T>): Analiza que tan reemplazable es cada ficha de su mano y juega la más prescindible posible bajo la eurística de "Mientras más opciones tenga, menos me voy a pasar"

-"El Pro"(Pro<T>): Analiza el historial del juego, tiene en cuenta las fichas que sabe que no lleva

tanto sus aliados como sus oponentes, además analiza que fichas ha jugado cada uno de ellos con el objetivo de seleccionar la ficha ideal a jugar. Basado en las eurísticas de "no matar la del tuyo", "matar la del rival", "matar la que el tuyo no lleva", "poner la que el enemigo no lleva", "evitar quitar una que el rival no lleve" y "no poner la que el tuyo no lleva".

"El Impredecible"(EstrategiaCambiante<T>): Juega a partir de alguna de las estrategias anteriores aleatoriamente

Rules<T> representa las reglas del juego. Recibe un conjunto de interfaces que especifican cada regla del juego, entre las que podemos mencionar:

-IPieceEvaluator<T>: Posee el método Evaluar, el cual recibe una ficha y devuelve un entero (su valor asignado) Entre las diferentes formas implementadas de evaluar una ficha, podemos mencionar: la suma de sus caras(SumEvaluator), el producto de las mismas(MultiplicatoryEvaluator), su máximo(MaxEvaluator), asi como la norma euclideana asociada a la misma si la consideramos un vector de Rn(VectorialEvaluator), un evaluador que devuelve el mismo valor para toda ficha(EquitativeEvaluator<T>) y un evaluador que duplica el valor de los dobles(DoubleDoublesEvaluator). Tambien se ha implementado un evaluador de char, el cual le asigna a cada letra su posición en el alfabeto latino y devuelve la suma de estos valores(CharEvaluator).

-IWinCondition<T>: Determina cuando se da por concluido un juego, estre las diferentas formas de tomar tal decisión tenemos: Al trancarse el juego(FinPorTranque<T>), cuando la suma de los valores de las fichas en la mesa es mayor a 150(Mesa150puntos<T>) y al ocurrir dos pases seguidos, todas además de que un jugador se pegue(DosPases<T>)

-IWinner<T>: Al finalizar un juego, determina quién es el ganador del mismo. Como criterios para determinar esto tenemos: El equipo con el jugador con la menor mano(menor suma de los valores de las fichas en la mano)(finalPorPuntos<T>) y el equipo con las menores manos(menor suma de los valores de las fichas en todas las manos del equipo)(TeamPoints<T>)

-IGenerator<T>: Al iniciar el juego, genera las fichas con las cuales se jugará, el metodo Generate recibe un entero que indica cuantos valores diferentes puede tener cada cara. Entre las diferentes formas de generar las fichas, podemos mencionar: todas las combinaciones de numeros posibles desde 0 a n(ClassicGenerator), asi como un generador que crea fichas al azar de valores válidos(CrazyGenerator) y uno encargado de crear las combinaciones de las n+1 primeras letras del alfabeto latino(CharGenerator)

-ILegalPLay<T>: Proporciona un criterio de si una jugada es o no legal. Entre estos criterios

tenemos: Si la cara de la ficha a jugar es igual a la cara de la ficha por donde se jugará(RegularLegalPlay<T>), asi como una modalidad del criterio anterior donde siempre se pueden usar dobles(AlwaysCanUseDobles<T>). Tenemos una variación donde siempre se puede jugar un blanco o el inmediato superior al valor por donde se jugará(Escareromino). Otra forma posible permite a los jugadores jugar por una cara que sea igual al antecesor o al sucesor de la ficha en mesa(Piramiomino).

-IPieceDistributer<T>: Se encarga de repartir las fichas a cada jugador al iniciar el juego. Recibe como parámetro la cantidad de fichas que puede tener un jugador como máximo. Se ha creado una modalidad que le da a cada jugador el máximo de fichas al iniciar(IPieceDistributer<T>) y uno que le da una cantidad aleatoria entre 1 y el máximo(distribucionRandom<T>).

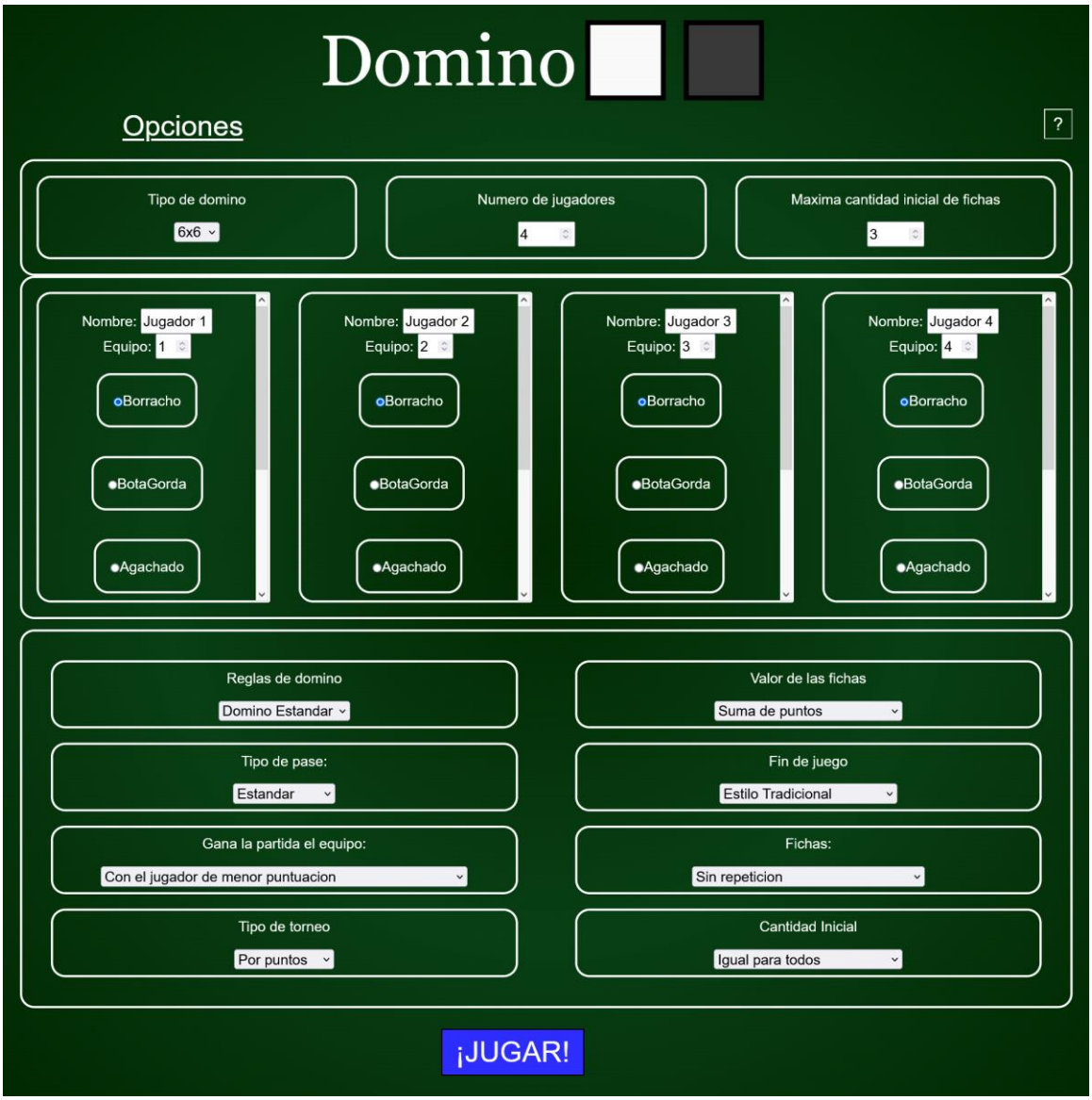
-ITurn<T>: Establece las acciones a realizar durante un turno, rigiendo el proceso del juego. Se ha creado una modalidad en la cual cada jugador simplemente juega una ficha por turno(NormalTur<T>), otra en la que, cada vez que un jugador se pasa, roba una ficha y una en la que si un jugador se pasa(Robadito<T>), además este de robar una ficha, se invierte la mesa, el jugador que comienza el juego puede jugar una ficha además de la salida, y si juegas un doble, puedes volver a jugar(Ciclomino<T>).

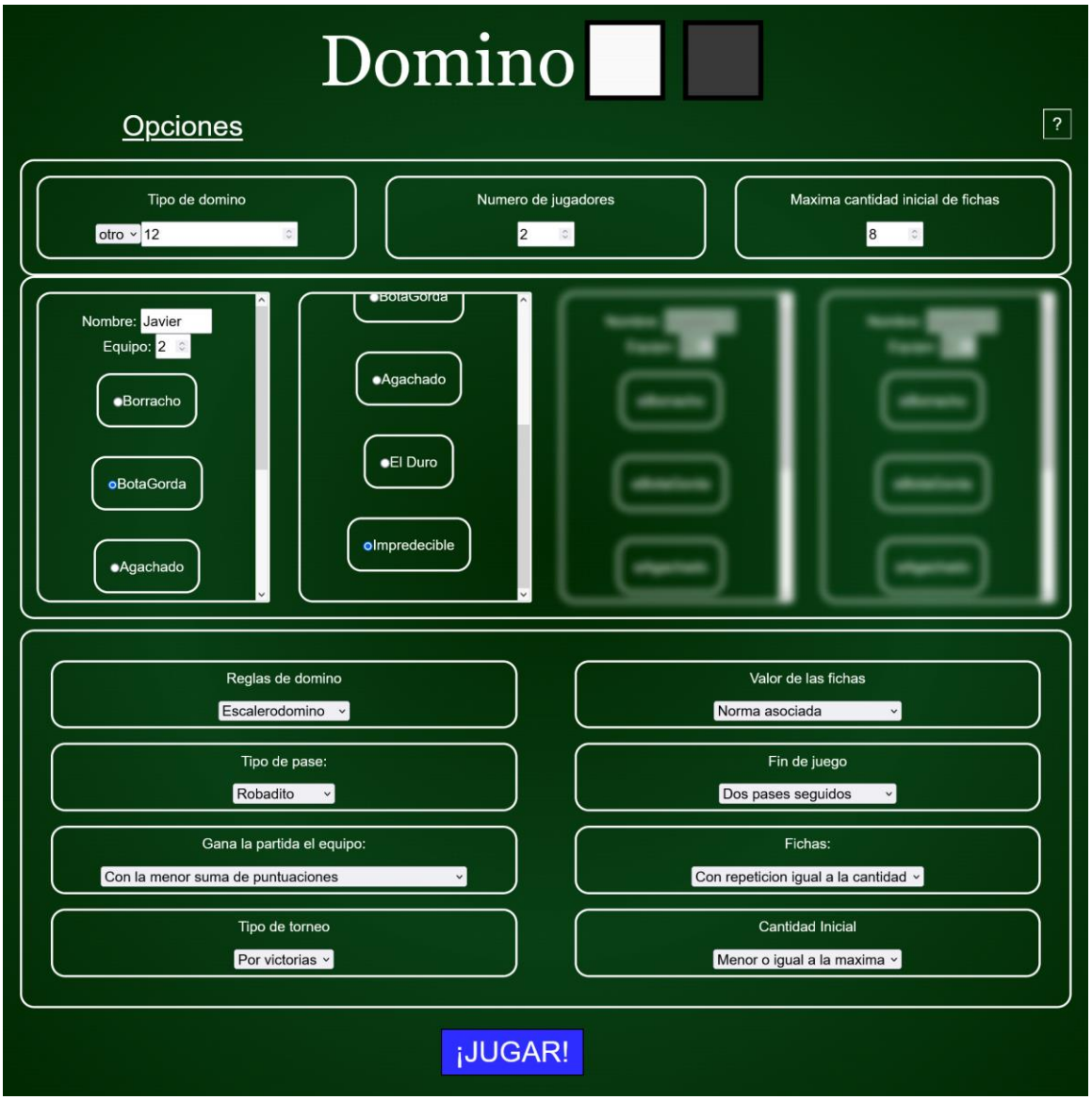
-ITorn<T>: Establece el funcionamiento de un torneo, la forma de asignar puntuaciones a los equipos y las condiciones de finalización del mismo. Se ha implementado una modalidad de torneo en la cual gana el primer jugador en alcanzar 3 victorias gana(TornPorVictorias<T>), y uno en el cual, al acabar un juego, se le asigna al equipo ganador una puntiacion igual a la suma de los valores de las manos de los jugadores del equipo perdedor, ganado el primer equipo en obtener 100 puntos(TornPorPuntos<T>).

INTERFAZ GRÁFICA

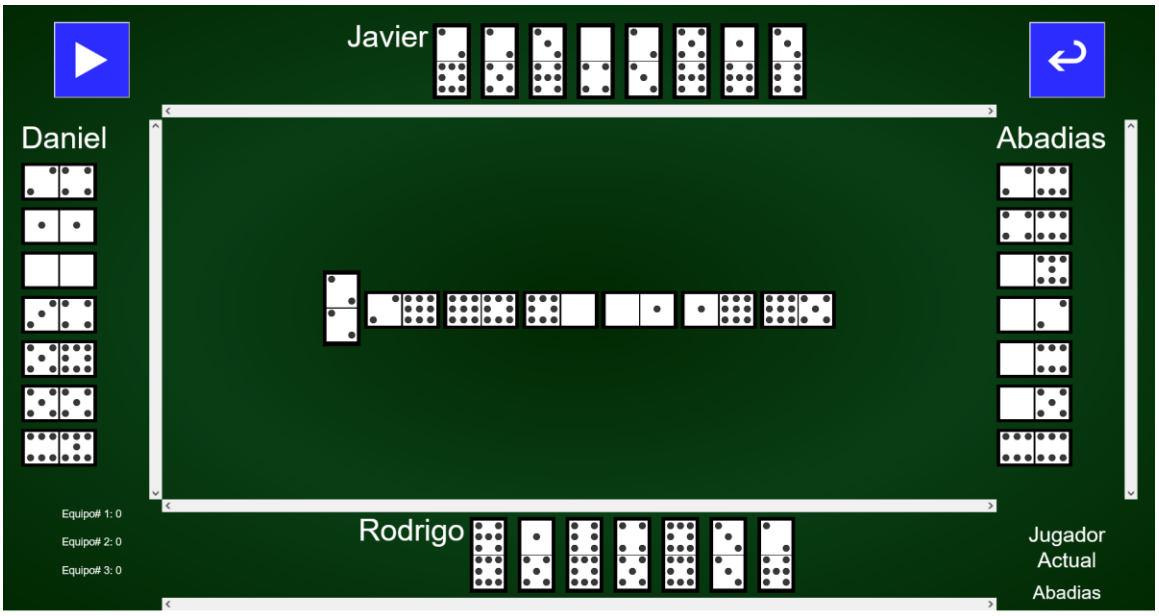
Como fue mencionado anteriormente, la interfaz gráfica fue realizada en Blazor, y los detalles sobre como funciona se encuentran dentro de una sección de esta(presionando el boton ?). Consiste en dos páginas web:

-Index: encargada de permitir a los usuarios regular las reglas del juego, disponiendo de todas las diferentes mecánicas a variar, asi como poder seleccionar y personalizar los jugadores y sus respectivos equipos:


Domino
Opciones
?
Tipo de domino
6x6
Numero de jugadores
4
Maxima cantidad inicial de fichas
3
Nombre: Jugador 1
Equipo: 1
Borracho
BotaGorda
Agachado
Nombre: Jugador 2
Equipo: 2
Borracho
BotaGorda
Agachado
Nombre: Jugador 3
Equipo: 3
Borracho
BotaGorda
Agachado
Nombre: Jugador 4
Equipo: 4
Borracho
BotaGorda
Agachado
Reglas de domino
Domino Estandar
Valor de las fichas
Suma de puntos
Tipo de pase:
Estandar
Fin de juego
Estilo Tradicional
Gana la partida el equipo:
Con el jugador de menor puntuacion
Fichas:
Sin repeticion
Tipo de torneo
Por puntos
Cantidad Inicial
Igual para todos
¡JUGAR!

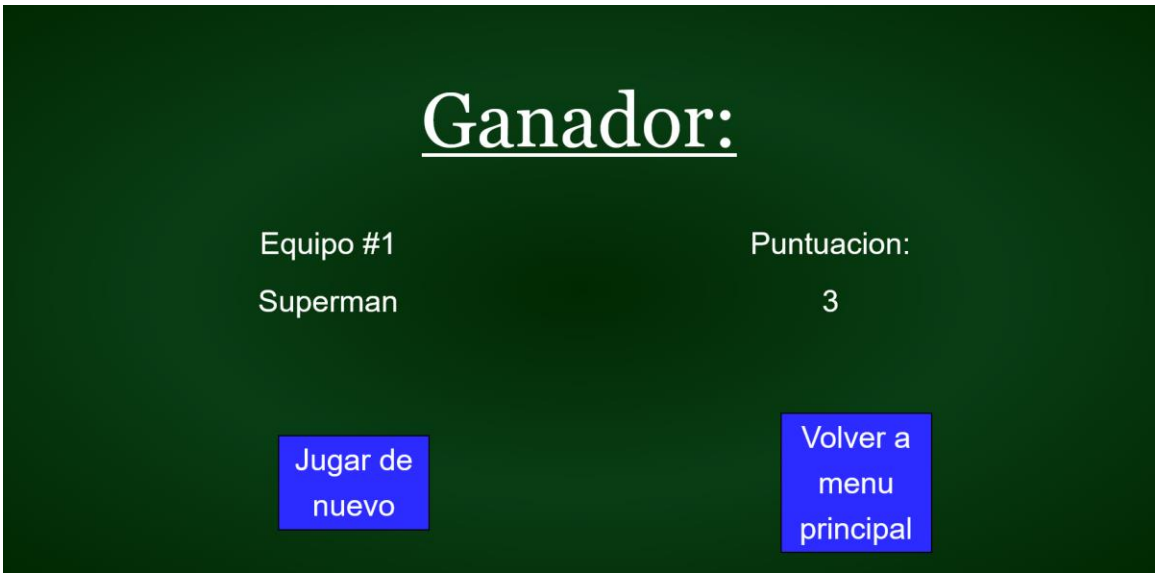


-Game: muestra las escenas de la partida creada ilustrando el desarrollo y funcionamiento del Engine de acuerdo a las opciones seleccionadas.



juego tipico

fin de una ronda



fin de un juego

Cabe mencionar que esta interfaz demuestra la versatilidad de nuestro código y su facilidad para ajustarse a las reglas seleccionadas, mas no soporta todas las diferentes variaciones del juego, no por un problema de capacidad, sino de "estética". Por ejemplo, solo se pueden seleccionar de 2 a 4 jugadores, ya que aunque el Engine lo permite, es "engorroso" mostrar una partida de una cantidad mayor de cuatro jugadores.

Por ultimo, cabe mencionar a la clase estática Bridge es la encargada de enlazar la parte visual con la lógica del programa, creando el torneo con la variante elegida y accediendo a las escenas del juego.

CONCLUSIONES

El equipo desarrollador asegura haber creado una aplicación extensible y mantenible, respetando cada uno de los principios SOLID y cumpliendo las condiciones minimas del proyecto, con un código correctamente comentado y organizado. Se espera que este a la altura de poder ilustrar muchas más permutaciones de reglas y sea capaz de abarcar todas las formas de juego a partir de nuetra estructura de clases.