Domino Game

El propósito de este proyecto es implementar una biblioteca de clases que permita modelar y experimentar con diferentes variantes del juego Dominó, así como diferentes estrategias de jugadores virtuales.

General

El dominó es un juego de mesa surgido en China hace miles de años a partir de los dados con seis caras y se han creado diversas variantes del juego mediante la adaptación de sus reglas. El modo clásico de juego conocido en Cuba inicia cuando los jugadores reciben sus fichas,y en un orden elegido cada uno pone por turno una ficha, si las lleva, donde el primero en vaciar la mano gana, si el juego se tranca,como se conoce cuando ningún jugador contiene en su mano fichas que se puedan enlazar con las opciones que hay en la mesa, se determina quien ganó de acuerdo con la regla asociado a ello, normalmente sería quien menos puntos acumule en su mano.

Dominos Game es una implementacion basada en la idea general de este juego común,en el que se le da al usuario la oportunidad de decidir con que regla quiere jugar. Además se crea con la idea de poder implementar nuevas variaciones de juegos cuando sea conveniente, de modo que la flexibilidad de la estructura permita hacer cambios sin afectar o tener si quiera que entender mucho sobre la implementación inicial.
Por lo general los aspectos variantes de un tipo de dominó a otro, así como de casi cualquier juego de mesa, vienen dados por reglas. estas reglas suelen ser:

Repartición

- modo de obtener las fichas(cartas,cualquier otro objeto con el que el usuario interactúa cuando hace una jugada), es decir el criterio de asignación, variando tanto la cantidad de fichas como de veces en que son repartidas, o cualquier otro modo de repartición que pueda ser imaginado. Casi siempre se inicia en un estado neutral,para que haya cierta igualdad de condiciones al comienzo, dejando a la suerte este error entre la mano que tiende a la victoria y la que no.

  En Domino Game para modelar esta regla se emplea la interfaz IShuffler:

```
public interface IShuffler

{

    public void Shuffle(Player player, GameInformation gi,
ref int index, Referee rf, [Optional] int cant);

}
```

Asi se garantiza que dado un repartidor determinado que debe implementa IShuffler se tome el modo de repartir que se desee. Para este aspecto se crearon dos repartidores, shuffler1 y shuffler2.

shuffler1:

shuffler1 es una implementación del repartidor clásico. Dado un jugador, la información del juego y el indice correspondiente a la última ficha repartida del mazo, asigna a cada jugador la cantidad correspondiente de fichas con la opción de dominó que se juega, en el caso de ser el dómino de 9 se le asignan 10 como se hace de forma convencional en el juego clásico.

```


public class shuffler1 : IShuffler
```

```
{

    public void Shuffle(Player player, GameInformation gi,
ref int index, Referee referee, [Optional] int cant)

    {
            ...
        }
}
```

shuffler2:

En esta variación se emplea la paridad, en el sentido si éste jugador tiene un índice par, cuando se le reparte o repartió el repartidor había repartido un número impar de veces hasta el momento entonces se le asignara entre sus fichas una suma total de puntos que corresponda con dicha paridad.

Controlador de Turnos

En esta variación se establece el control del orden de los jugadores. Se utiliza una interfaz ITurnPlayer, que define como se comportará el turno siguiente.

```
public interface ITurnPlayer

{

    public void Turn(int[] turns, int ind);

}
```

Finalización

- Cuando finaliza un juego, de acuerdo con el estado del tablero que condiciones hacen que ya se haya terminado la partida. Se utiliza la Interfaz IFinalized, que se asegura con la función `EndGame` de determinar la finalización de la partida.

```
public interface IFinalized

{

    public bool EndGame(GameInformation gm, Referee
referee,ref int max);

}
```

Se implementan las variacion de `clasicEnd` y `EndGameAbovePoints` .

En *clasicEnd* se determina finalización dado el modo convencional, es decir cuando alguno de los jugadores ya no tenga fichas en la mano o cuando ninguna de las fichas de los jugadores se pueda jugar en la mesa.

En *EndGameAbovePoints*
Variante de terminación cuya condición es que haya un número random de puntos en mesa, es decir dado un número entre 0 y total de punto entre todas las fichas que tienen los jugadores, finaliza cuando haya en mesa esa puntuación como suma de todas las fichas jugadas.

Ganador

-Determinación de que jugador es el que tiene condiciones mas favorables de acuerso al criterio de evaluación para devolverlo como ganador. De manera clásica el ganador es el que no tenga fichas o bien el que menos puntos tenga dada la suma de puntos de su mano. Se determina a partir de la interfaz `IWinner`:

```
public interface IWinner

{

    public string Win(Referee referee,GameInformation gm);

}
```

La función Win es la encargada de decolver el id del jugador victorioso.
Se implementan dos variaciones `clasicWinner` y `Winner2`, el primero adquiere como criterio el antes mencionado, es decir el ganador clásico, en Winner2 se basa en dado el número de fichas de los jugadores y el puntaje, gana el que tenga más baja la multiplicación entre estos parámetros.

Validador de jugadas.

-Determinacíon de si una jugada es válida o no, una jugada viene dada por la ficha seleccionada por el jugador y en donde la quiere jugar, por lo general llamamos forros a las fichas tiradas por un jugador que no se pueden enlazar con la opción por donde este decide jugarla. Se emplea la interfaz IValidator que implementa el método ValidPlay, booleano que nos dice si es un forro la jugada.

```
public interface IValidator

{

   public bool ValidPlay(jugada jugada, GameInformation gi
);

}
```

Se cuenta como validadores :

`validator1`

basado en la forma de validar jugada clasica, es decir solamente cuando dos fichas se pueden enlazar

`validatorEvenOdd`

Se tiene una ficha formada por las opciones de juego, es decir si las opciones en la mesa para jugar son 1 y 2 esta nueva tiene correspondiente a sus caras estas opciones, ahora si el peso de esta ficha es par(impar) entonces la jugada será valida si y solo si el peso de la ficha jugada es par(impar), claro estas ficha jugada debe poderse enlazar con alguna de las opciones de juego. En caso de que no se haya jugado ninguna ficha anteriormente entonces será válida cualquier jugada.

Estructura

Domino Game cuanta con una biblioteca de clases donde se encuentran todas las clases correspondiente a la implementación lógica.
Las siguientes clases son empleadas en esta implementación:

GameInformation

GameInformation tiene como objetivo gestionar lo referente a la información necesaria para trabajar la implementación, es decir, con esta se registran los cambios durante el juego.
Contiene:

- Todas las fichas que son jugadas durante la partida
- Opciones de actuales por las que se puede jugar
- Lista de todas las fichas barajeadas.
- delegado que se encargara de calcular peso de las fichas
- Almacenador de las jugadas de cada jugador.
- Almacenador de las fichas que no llevó en cierto turno cada jugador,los pases.

-Método auxiliar usado para la repartición de fichas `public List<Records> MakingRecords(int cant)` Esta función coloca las fichas en diferentes órdenes,como barajear ,asi poder tomarlas como particiones del conjunto de fichas, teniendo distintas distribuciones a seleccionar cada vez que se utilice este método.
-Función que se encarga de llevar el total de puntos que hay en el total de fichas de todos los jugadores, los puntos repartidos. `public int shuffledPoints(Referee referee)`
-Función para conocer el peso total que hay en el tablero entre todas las fichas. `public int PointsInGame()`

InformationForPlayer

InformationForPlayer es un reporte con todos los datos que necesita conocer un jugador en su turno para poder decidir jugada. Esta clase esta creada de modo que sea lo único a lo que el jugador tendrá acceso de modo que no pueda cambiar elementos del juego a conveniencia y que pueda seguir una estrategia adecuada.

Variations

Se implementan las variaciones de cada regla antes explicadas.

Records

En Records se define lo que es una ficha. Esta dada una lista de valores son asignados dichos valores a los elementos, siendo definidas como una serie de elementos con un valor asignado.

jugada

Donde se define una jugada como la ficha a jugar y la posición donde se desea colacar.

Rules

Reglas del juego que pueden variarse, aquí se encuentran una serie de interfaces que son las encargadas de variar los direfentes aspectos en Domino Game.

Players

En Player se encuentran las estrategias para los jugadores, donde podemos encontrar una clase player general, que define al jugador y el metodo principal que debe contener que es el de dado el estado del tablero determinar una jugada. Se pueden encontrar la implementación del jugador manual, que no es mas que un jugador humano, el jugador Random que explora entre las opciones que se podían enlazar en la mesa, obtenidas en el reporte que se les brinda, el jugador Greedy que siempre va a jugar la ficha que "óptima" en lo posible, y el jugador Data, que juega tratando de asegurar el dominio con el número más frecuente en sun mano(se le conoce como data).

Los diferentes tipos de jugadores son parten de una clase abstracta Player que cuenta con una propiedad identificación y un método `GiveMeRecords()` que recibe la información de la mesa y las reglas actuales para planear su estrategia y devuelve una `jugada` que como ya se planteó consta de una ficha y la posición en la que se va a jugar.
Se hace de esta manera ya que con el objetivo de hacer que el árbitro sea el que defina si la jugada que se propuso por parte del jugador es válida o no, permitiendo así que el juego identifique de los "forros del jugador".

```
public abstract class Player

{

    public  string id;
```

```
    public Player(string id)
    {
        this.id=id;
    }
    public abstract jugada
GiveMeRecords(InformationForPlayer info, Referee rf);
}
```

En cuanto a las estrategias, estas difieren en dependencia de las clases que las implementan, por ejemplo :

- La clase RandomPlayer es la más básica ya que genera la jugada seleccionando de manera random tanto la ficha a jugar como la posicion en la que ponerla.
- La clase GreedyPlayer tiene varios métodos auxiliares como `Rearrange` y `FindOption` para seleccionar la ficha con caracter "óptimo" a jugar (nótese que una ficha es óptima en dependencia del peso que tenga, y esto se define en correspondencia con las reglas activas en la partida en curso) (Ejemplo: En el dominó clásico se determina que si el juego se tranca, gana el jugador con menos valor acumulado en su mano, por tanto una ficha óptima es la que más valor o peso posee, y esa es la que se juega de ser posible).
- La clase DataPlayer es un poco similar a GreedyPlayer en el sentido que hereda de esta para tener acceso a su metodología, pero su estrategia es en cambio algo más compleja, ya que se basa en determinar el valor más prominente en su mano y tomarlo como la "data" para basar su juego alrededor de las fichas que contengan ese valor.

Referee

El árbitro es el encargado de modelar el funcionamiento del juego. Este recibe las reglas por las cuales se tratará la partida, almacena las fichas repartidas a cada jugador.
Por medio de `Play` hace los cambios a partir de la jugada del jugador,como registrar en GameInformation las nuevas opciones de juego y agregar al tablero la ficha en la posición indicada por el jugador.
Con `HavesARecord` determina si un jugador contiene alguna ficha en su mano que pueda enlazarse con las fichas en la mesa.
Se encarga de construir el reporte que se le dará a los jugadores en su turno para que puedan tener toda la información necesaria para decidir una jugada en `ProvidedInformation`.