Informe del 2do Proyecto de Programación

Facultad de Matemática y Computación

Ciencias de la Computación

Tema: Emulador de Torneos para Juegos de Mesa

Jordan Plá González

C111

Curso: 2019 - 2020

Informe del Proyecto de Programación

Facultad de Matemática y Computación

¿Cuál es el flujo para crear un Torneo?

El torneo es genérico, por lo que se debe crear un Torneo en función de un Juego de Mesa, para ello cada torneo recibe como parámetro un Juego de Mesa (siendo precisamente el mismo) del cual será ese Torneo y una lista de Jugadores que juegan ese mismo Juego, independientemente de si el Juego de Mesa se jugará por Equipos o no. El torneo posee una propiedad *Estado*, que representa el Estado del Torneo (*iniciando*, *en progreso*, *finalizó*) que deberá actualizarse dependiendo del momento en que se encuentre el torneo. Finalmente, para crear un nuevo tipo de torneo tendrá que heredar de la clase genérica y abstracta antes dicha y deberá, además de actualizar la propiedad *Estado,* implementar, dos métodos:

```
string NotificaTorneo();
```

que se encarga de Notificar el torneo mediante el Estado (*propiedad*) en que se encuentra y,

```
IEnumerator<Partida<JM>> GetEnumerator();
```

que se encarga de enumerar las partidas del mismo tipo de juego creadas en este nuevo tipo de torneo, aquí se presentará el orden en que jugarán los participantes y se deberá comprobar si la partida finalizó para pasar a la siguiente, en caso que la funcionalidad de este torneo necesite de partidas anteriores para realizar partidas posteriores, y obviamente la partida puede establecer un ganador que no necesariamente tiene que ser el que mayor puntuación obtenga o puede resultar en empate; según el torneo que se desee crear puede obtener las puntuaciones de la partida actual, conocer cuál fue el ganador de la misma, si resultó en empate o establecer sus propias puntuaciones.

¿Cuáles son los pasos para ejecutar un Torneo?

Para la correcta ejecución de un torneo, se creó una clase sellada *Emulador* que también es genérica, siempre y cuando el tipo sea un Juego de Mesa, esta clase recibe como parámetro un Torneo que su Juego de Mesa tiene que ser del mismo tipo del Emulador. Esta clase se encarga de iterar cada partida del torneo, a su vez por cada partida itera todos sus juegos, y por todos sus juegos itera cada jugada, para ello tiene un método que retorna un `Enumerator<string>` en el cual se recoge todas las notificaciones ofrecidas por el torneo, sus partidas, sus juegos y jugadas, y muestra las puntuaciones una vez finalizado el Torneo. Esta clase contiene las propiedades pertinentes que se modifican en dependencia de si el usuario desea que se le muestre las Notificaciones de las Partidas, de los Juegos y de las Jugadas. Los Juegos de Mesa contienen propiedades tales como, si es un Juego que se puede jugar por Equipos, y en tal caso la cantidad de jugadores que pueden tener los mismos, si el Juego otorga puntuaciones, la capacidad máxima y mínima de Jugadores que puede tener. Estas propiedades son fundamentales para crear un torneo ya que ofrece información necesaria para saber si el juego es compatible con el tipo de torneo que se quiere crear, es importante usarlas.

Informe del Proyecto de Programación

Facultad de Matemática y Computación

Este será el formato que tendrá la aplicación, seleccionar el botón Juegos, Jugadores o Torneos, y por supuesto en dependencia del Juego escogido se mostrará los Jugadores o Torneos compatibles con el mismo. Si aun así presenta dudas en cómo proceder, en el extremo superior derecho hay un botón de ayuda que le indicará como configurar correctamente el emulador; y una vez finalizada la configuración con todo lo escogido y seleccionar el botón de *finalizar* saldrá una nueva pantalla que servirá para ejecutar el torneo seleccionado. Si quiere ejecutar correctamente un torneo a partir de la .dll solo debe crear una instancia del emulador con el mismo juego de mesa del torneo, pasarle su torneo como parámetro, y pedirle las notificaciones.

Tipos de Torneos implementados

- *Calificación Individual*

Este tipo de torneo presenta algunas particularidades, tales como que no se pueden presentar Equipos, el Juego de Mesa tiene que tener una capacidad mínima de hasta dos jugadores y el juego debe otorgar puntuación a los mismos. Para hacer partidas más justas, si el Juego de Mesa no se puede jugar en solitario este torneo ofrece un jugador virtual ¨Jordan¨ que se enfrentará con cada jugador del torneo, así las puntuaciones que se obtengan serán las más adecuadas, si no es el caso, y el Juego de Mesa se puede jugar en solitario, entonces cada Jugador jugará dicho juego sin la necesidad del jugador virtual.

- *Título*

A diferencia del anterior torneo, el Juego de Mesa no puede ser jugado por una única persona, el juego debe ser jugado por dos Jugadores o dos Equipos, en este torneo no importa si el Juego de Mesa no ofrece puntuaciones, lo que importa es quien gane la partida o si resulta en empate.

Se escogen dos Jugadores o dos Equipos al azar y los pone a competir, el ganador se convierte en campeón temporal y es retado con otro adversario al azar que no haya jugado anteriormente, y el que gane se convierte nuevamente en campeón temporal y así hasta que hayan competido todos los jugadores o equipos y finalmente quedará un ganador que será el Campeón del Torneo, en caso de que la partida resultara en empate se mantendrá el campeón anterior.

- *Dos a Dos*

Este torneo es similar al de título en cuanto a que el Juego de Mesa no puede ser jugado por una única persona como su nombre bien indica es de a dos contrincantes (ya sean dos Jugadores o dos Equipos), la diferencia radica en que en este torneo sí importa si el Juego de Mesa ofrece puntuaciones, como ya había aclarado anteriormente quien tenga mayor puntuación no necesariamente es el ganador de la partida. Finalizado este torneo se muestra los Jugadores y sus puntuaciones obtenidas en cada partida, otra diferencia radica en que para los *n* jugadores o equipos deberán enfrentarse con los *n – 1* restantes.

Jugadores implementados

1. *Jugador Aleatorio*

Es un jugador genérico ya que puede jugar cualquier juego, implementa la interfaz genérica `IJugador<JM>` que lo convierte en jugador, producto a esta interfaz implementa el método `Jugada Estatregia (IEstado<JM> estadoJuego)` que recibe un estado (*tableros*, *fichas*, *currentActual*, *etc*.) del Juego que se está jugando, y dependiendo de ello realiza su jugada. Para su apoyo, cuando es construido este tipo de jugador se le pasa como parámetro un evaluador del mismo juego, este evaluador contiene un método que dado un estado del Juego retorna jugadas válidas, y ya luego de haber evaluado este estado, dicho Jugador escoge aleatoriamente una jugada entre las disponibles retornándola en este método. En fin, este jugador se encarga de ofrecer una jugada aleatoria entre las jugadas válidas disponibles.

2. *Jugador Goloso*

Este otro jugador también es genérico ya que puede jugar cualquier juego; como implementa la interfaz genérica `IJugador<JM>` entonces implementa el método `Jugada Estatregia (IEstado<JM> estadosJuego)` que recibe un estado del mismo Juego para realizar su jugada. También se le pasa un evaluador del mismo juego con sus características como jugador, este evaluador en su método de `Evalua()` retorna jugadas válidas y valores según la prioridad de la jugada, luego este Jugador escoge aleatoriamente una jugada entre las jugadas con mayor valor, retornándola en este método. En fin, este jugador se encarga de ofrecer una jugada de mayor prioridad entre las jugadas válidas de mayor prioridad disponibles.

Lo que hace especial a estos jugadores, además de su estrategia es el evaluador de cada juego que contienen cada uno. Este evaluador ofrece características especiales en cada juego que implementa como evaluador, por ejemplo, en el *Dominó* el *goloso* se encarga de jugar la ficha válida más alta que tenga, en el *Tic Tac Toe* se encarga de ocupar las esquinas para ganar lo más rápido posible, así como otras particularidades dependiendo del juego, algo similar sucede con el *aleatorio* que también tiene sus particularidades.

¿Cómo crear un nuevo Jugador?

Depende del jugador que desee crear, el proyecto le ofrece varias alternativas:

- *Jugador Genérico*

Si desea crear un nuevo jugador genérico como los anteriormente implementados (*Jugador Aleatorio* y *Jugador Goloso*) que juegan cualquier Juego de Mesa, solo tiene que implementar la interfaz `IJugador<JM>` donde ese juego es del mismo tipo que el del jugador que se quiere crear, a su vez tendrá que implementar el método que contiene esta interfaz que ya explicamos anteriormente, recibe como parámetro un estado del Juego y el jugador mediante este método ofrece una jugada para ese Juego de Mesa. Este proyecto le da la sugerencia, de crear un evaluador para ese nuevo jugador que implemente los evaluadores de los juegos que desee jugar y pasárselo como parámetro en el constructor de este Jugador, así sería más sencillo evaluar un juego que se conoce de antemano, y realizar las estrategias pertinentes según este jugador para ese Juego.

```
public interface IJugador<JM> where JM : JuegosdeMesa
{  Jugada Estrategia(IEstado<JM> estadoJuego);  }
```

```
public class JugadorNuevo<JM> : IJugador<JM> where JM : JuegosdeMesa
{
   public JugadorNuevo(string nombre, int identificador, IEvaluador<JM> evaluador)
   {
       public Jugada Estrategia(IEstado<JM>estadoJuego)
       {
           ///Líneas de Código
       }
   }
}
```

```
public class EvaluadorJugadorNuevo : IEvaluador<JuegoDeseado>
{
   public List<Jugada,int> Evalua(IEstado<JuegoDeseado>estadoJuego)
   {
       ///Líneas de Código
   }
}
```

OJO!!! EL Evaluador debe implementarse para los Juegos que ese Jugador vaya a Jugar

- *Jugador Específico*

Si desea crear un nuevo jugador que juegue un Juego de Mesa en específico, sería más sencillo aún, solo tiene que implementar la interfaz `IJugador<JuegoDeseado>` donde ese juego es el que jugará el Jugador creado, por lo que tendrá que implementar el método que contiene esta interfaz que ya se explicó anteriormente, pero esta vez será en este Juego en específico, recordemos que recibe como parámetro un estado, en este caso de este Juego, y el jugador mediante este método ofrece una jugada. Para este jugador no es necesario crear un evaluador ya que conocemos de antemano que juego es el que jugará, por lo en el método *Estrategia* el jugador podrá hacerlo directamente.

```
public class JugadorNuevo : IJugador<JuegoDeseado>
{
   public JugadorNuevo(string nombre, int identificador)
   {
       public Jugada Estrategia(IEstado<JuegoDeseado>estadoJuego)
       {
            ///Líneas de Código
       }
   }

}
```

¿Cómo añadir un nuevo Juego?

Antes de pasar a la explicación de cómo crear un nuevo Juego de Mesa, es necesario aclarar algunos puntos para la fácil comprensión del mismo.

Es de total importancia qué tipo de *Jugada* es la que se realizará en el nuevo Juego de Mesa que se desea crear, por ejemplo, en el *Tic Tac Toe* una jugada es una *Posición*, en el *Dominó* es una *Ficha*, así que lo mejor será saber previamente que tipo de jugada se realizará en su juego. Una vez aclarado esto, tendrá que implementar su nuevo tipo de Jugada en caso de que no esté previamente creado, heredar de la clase abstracta *Jugada* para dar constancia de lo que es, e implementar su método `ToString()` que será lo que se mostrará en dicho juego.

También es importante saber qué es lo que ofrecerá el nuevo Juego de Mesa a los jugadores en su *Estado* temporal. O sea, cada juego debe tener un estado que será una clase que implementará la interfaz `IEstado<JuegoDeseado>` donde podrá ofrecerle al jugador todo lo que necesite para realizar una jugada, por ejemplo, el estado del *Tic Tac Toe* es el tablero y el índice del jugador así sabrá si juega *0* o *X*, el del *Dominó* es una *Lista de Fichas* que posee el jugador, así como las que están puestas en la mesa; por lo que será mejor tener presente que es lo que le ofrecerá a los jugadores para jugar su nuevo juego.

Luego de las aclaraciones anteriores podrá crear un nuevo Juego de Mesa, para ello deberá conocer que *Juego de Mesa* es una clase abstracta y que los nuevos juegos deberán heredar de ella. Una vez heredado deberá implementar unas propiedades para su juego que facilitará la compatibilidad con los torneos, como es el caso de si su nuevo Juego de Mesa podrá ser jugado en Equipo, en tal caso cuál es la capacidad de jugadores, si el juego ofrece puntuación, entre otras, pero lo más importante es que debe crear un Árbitro asociado a su juego que es el que se encargará de darle orden a los jugadores, que no hagan trampas, de ofrecer las puntuaciones en sus juegos y partidas, de realizarlas y de notificarlas, entre otros elementos que contiene la interfaz `IArbitro`.

A continuación, un ejemplo de cómo crear un nuevo Juego de Mesa con todo lo relacionado a él:

```
public class JuegoNuevo : JuegosdeMesa
{
   public override bool Equipos => true o false (para poder jugar por equipo)
   public override bool DaPuntuacion => true o false (para saber si ofrece puntuación)
   public override int CantJugadoresPorEquipos => # (número de jugadores por equipo en tal caso)
   public override int CapacidadMaxima => # (capacidad máxima de jugadores que permite)
   public override int CapacidadMinima => # (capacidad mínima de jugadores que permite)
   public override IArbitro CrearArbitro<T>(List<T> participantes)
   {
        ///Líneas de Código
   }
}

public class ArbitroJuegoNuevo : IArbitro
{
   ///Métodos Auxiliares
   public int IndexGanador { get; private set; }
   public int[] Puntuaciones { get; private set; }
   public EstadoPartida EstadoPartida { get; private set; }
   public EstadoJuego EstadoJuego { get; private set; }

   public string NotificarPartida(bool seMuestra)
   { … } ///Notificar partida según el EstadoPartida
   public string NotificarJuego(bool seMuestra)
   { … } ///Notificar juego según el EstadoJuego
   public string NotificarJugada(bool seMuestra)
   { … } ///Notificar jugada según la Jugada Actual

   public Juego RealizarJuego()
   {
       ///Líneas de Código
   }

   public Jugada RealizarJugada()
   {
       ///Líneas de Código
   }
}

public class EstadoJuegoNuevo : IEstado<JuegoNuevo>
{
   ///Líneas de Código
}
```

IMPORTANTE!!! Recuerde implementar el tipo de Jugada que desea retornar

OJO!!! Lo que necesita el Jugador para Jugar

Se crea el estado del Juego en el momento y se le pasa al jugador