Dommie Report

Preliminares :

Este proyecto está orientado a cumplir con la evaluación anual de la asignatura Programación en el segundo año de Ciencias de la Computación de la Universidad de la Habana. Su objetivo es desarrollar una aplicación que simule el juego de domino, además de contener ciertos aspectos que van a influir en la mecánica de un juego clásico de domino. El principal propósito es demostrar los conocimientos adquiridos durante el curso que responden a un buen desarrollo de software, que sea mantenible y extensible, que cumpla en su mayor medida los principios del SOLID y otros principios de la programación en general; todo ello en parte, mediante el uso de interfaces y delegados.

Para ello hemos desarrollado una librería de clases `DominoLibrary` y una aplicación de consola `ConsoleApp`.

La librería de clases es la encargada de controlar toda la lógica del juego, mientras que la aplicación de consola mostrará al usuario los resultados del juego que haya decidido simular, y le permitirá ser partícipe durante la ejecución.

En la carpeta raiz está ubicado el archivo `README.md`, con la información necesaria para la ejecución del proyecto.

El objetivo de este reporte es documentar al usuario de Dommie para que se familiarice con los términos de nuestro juego y pueda entender el funcionamiento del código.

Aspectos variables :

Max Token:

Este aspecto se refiere al número del doble máximo que estará en el juego. Este parámetro influirá en la posterior generación de las fichas con que se va a jugar. Por ejemplo, la alternativa clásica del doble-9, va a generar 55 fichas donde la mayor de todas será el doble-9, esta sería la ficha máxima. Su implementación viene dada como un parámetro de tipo `int`, que es pasado a las clases y métodos tras su selección.

Variantes:

- double - 6
- double - 7
- double - 8
- double - 9
- double - 10
- double - 11

- double - 12

De momento han sido las variantes implementadas, pero si fuera necesario se pudiera extender a más opciones en el futuro.

Number of Players:

De igual manera que el aspecto anterior, damos la libertad de elegir cuantos jugadores el usuario quiere sentar en la mesa, no todos tenemos solo cuatro amigos (yo tengo unos poquitos más). Hemos establecido un rango de entre 2 a 6 jugadores, tampoco queremos que la partida dure lo que un monopoly.

Hand Out:

No es más que las distintas formas en las que se pueden repartir las fichas del juego. Es el criterio que define la manera en la que las fichas se asignan a los jugadores. No queremos que siempre se jueguen todas las fichas sobre la mesa, por tanto no las vamos a repartir todas; para ello hemos decidido que se repartan siempre el **71%** de las fichas del juego, dejando una parte fuera de partida, así complicamos un poco las posibles estrategias de jugadores más aventajados. Esta mecánica queda encapsulada en el delegado `HandOut`, que se encuentra, junto a las implementaciones en la clase `HandOuts.cs`.

Variantes:

- Random: Esta variante es la clásica forma de repartir las fichas del juego. No es mas que ¨dar agua¨, frase típica referida a la acción realizada por los jugadores para mezclar las fichas que luego elegirán para jugar la partida.
- Bigger First Unfair: Aquí rompemos la tradicional regla de repartir la misma cantidad de fichas por jugador. Dejamos al azar decidir la cantidad de fichas que tendrán los mismos, eso sí, garantizamos siempre que al menos una ficha tendrán. Para hacerlo un poco más interesante no solo escogemos los jugadores de forma aleatoria, sino que las fichas a repartir se eligirán por la mayor puntuación de la misma.

Over Round Condition:

Es el criterio que se consulta para determinar si una ronda del juego ya llegó a su fin. Esta mecánica queda encapsulada en el delegado `OverRound`, que se encuentra, junto a las implementaciones en la clase `RoundOvers.cs`.

Variantes:

- Classic: Con esta variante te sentirás familiarizado. La partida llegará a su fin una vez que uno de los jugadores haya puesto en mesa todas sus fichas de la mano, o simplemente haya sido jugada una ficha capaz de pasar a todos los jugadores en mesa.

- Crazy Token: Una ficha entre las fichas de los jugadores será escogida al azar. Los participantes del juego no tendrán conocimiento de la misma. En caso de haber sido puesta en juego, la partida habrá llegado a su fin. Sin embargo, es posible que todos los jugadores de la mesa se pasen sin haber sido jugada la ficha, por lo que de igual modo la partida habrá llegado a su fin.

Winners Getter Judgment:

Se refiere a al criterio de escoger el ganador de una ronda una vez esta ha terminado. Esta mecánica queda encapsulada en el delegado `WinnerRoundGetter`, que se encuentra, junto a las implementaciones en la clase `RoundWinners.cs`.

Variantes:

- Classic: El ganador será aquel jugador que haya puesto todas sus fichas en mesa. En caso de no haber sido posible lo anterior y todos los jugadores tengan al menos una ficha en su poder, serán contados los puntos de las fichas y obtendrá la victoria aquel jugador que obtenga la menor puntuación. De haber dos jugadores con la mínima puntuación posible el juego será declarado empate, es decir, ninguno de los participantes habrá alcanzado la victoria.
- Random: Una vez más dejamos al azar decidir tu suerte. Independientemente de tu puntuación o de haber sido capaz de poner todas las fichas en mesa, el jugador será elegido de forma aleatoria. Ahora bien, miremos el lado positivo, si durante la partida no te está yendo demasiado bien, no todo está perdido, aún puedes ser tú quien obtenga la victoria.
- Smallest 5 Multiple: Si decides jugar escogiendo esta variante debes combinar muy bien tus jugadas, pues solo podrás obtener la victoria si eres capaz de lograr alcanzar en tu puntuación el menor múltiplo de cinco distinto de cero al final de cada partida. En caso de no haber algún jugador que cumpla la condición, el juego será declarado empate.

Points Getter Judgment:

Se refiere al criterio mediante el que se computa una puntuación que será asignada al jugador ganador de cada ronda una vez este se conoce y ya ha terminado la ronda. Esta mecánica queda encapsulada en el delegado `PointsGetter`, que se encuentra junto a las implementaciones en la clase `PointsWinner.cs`.

Variantes:

- Classic: Si eres capaz de alcanzar la victoria lograrás obtener una puntuación igual a la suma de las puntuaciones de cada uno de los restantes jugadores. En caso de haber empate los jugadores no obtendrán ningun tipo de puntuación.

- 5 Multiples: Jugar con esta modalidad te permitirá alcanzar una puntuación con mayor rapidez o no, solo depende de tus habilidades para dejar al contrario con la mayor cantidad de fichas en mano. Tu puntuación en este caso será cinco puntos por cada una de las fichas que hayan quedado en manos de los adversarios.

Inner Selector Judgment:

Con un delegado abstraemos el criterio con que se elige el jugador que va a comenzar jugando en una partida. Quedan definidos en el archivo `InnerPlayer.cs`. Esta mecánica queda encapsulada en el delegado `InnerGetter`, que se encuentra junto a las implementaciones en la clase `InnerPlayer.cs`.

Variantes:

- Random: De manera aleatoria se decide cual de los jugadores en mesa será el primero en colocar una de sus fichas al inicio de cada partida.
- Bigger Token: Si eres uno de los jugadores de los cuales no le gusta la ficha doble-9, la cual de seguro conoces como "la gorda", tendrás la oportunidad de ser el iniciador del juego y poder deshacerte de ella rápidamente.
- Min Double: Esta variación del juego es muy semejante a la anterior, con la diferencia de que al iterar por cada una de las fichas de los jugadores, el jugador seleccionado para ser el iniciador del juego será aquel que tenga en su poder el menor doble entre las fichas repartidas a los jugadores. En caso de no exisitir dobles en mesa, o sea, que ningun jugador tenga en su mano una ficha de esta característica, lo cual es muy poco probable, el iniciador del juego será elegido de forma aleatoria.
- Max Data: Hemos querido que el iniciador del juego también pueda ser aquel que tiene la "mejor data", o sea, la mayor cantidad de caras repetidas entre las fichas de su mano. Si existen dos jugadores con la cantidad máxima, el elegido será aquel que haya sido encontrado primero.

Game Mode:

Hemos facilitado que el usuario no solo simule una simple partida de un juego de domino, sino que además pueda jugar torneos, o sea, un conjunto de varias partidas que al ganarlas un jugador, va acumulando puntos y la victoria se alcanzará una vez se haya obtenido una cantidad de puntos que puede ser especificada por el usuario antes de comenzar.

Teams Play:

Otro aspecto que hemos decidido extender es el juego en equipo ya que no hay un buen torneo de domino que no se juegue en parejas; por ellos hemos decidido

llevar esta modalidad a Dommie. Nuestra aplicación no se limita a que solo sea mediante parejas, sino que también deja libertades al usuario respecto a la manera de formar los equipos, todo ello completamente personalizable desde los menús de configuración inicial, que bien explicamos en su sección. Para una mejor adaptación a la lógica, consideramos un solo jugador como un equipo.

Human Player:

SÍ!!!, es los que estás pensando, ahora tú eres partícipe de nuestro juego. Te damos la opción de ser parte de Dommie e involucrarte a nuestros jugadores virtuales. Traza tus propias estrategias, demuestra tu destreza en el domino e intenta llevarte las mayor cantidad de partidas a tu palmarés.

Los jugadores :

HumanPlayer:

Implementación de IPlayer que permite al usuario ser partícipe del juego.

SingleStrategyPlayer:

Esta clase define a los jugadores que utilizan una única estrategia preestablecida para jugar.

Strategies:

- ***Botagorda***: El famoso "bota-gorda" tiene como estrategia jugar la ficha de mayor puntuación entre las posibles fichas válidas (dígase ficha válida aquella que cumple que uno de sus extremos coincide con alguno de los extremos de la mesa) en su mano.
- ***Mosaic***: Con esta estrategia el jugador trata de mantener un rango de fichas en su mano para poder acertar tantos números como sea posible. Si todas tus fichas tienen palos similares, te quedarás atrapado si eso es todo lo que está disponible en el tablero.
- ***Random***: Jugar de forma aleatoria no es más que seleccionar una ficha entre las posibles fichas válidas a jugar. Ten en cuenta que usar esta estrategia es impredecible. Con ella puedes o no alcanzar la victoria.

Cómo se elije una plantilla :

Una vez que el juego le da la bienvenida al usuario, se muestran una serie de menús que permiten la elección de las variantes que modifican el juego. Una vez determinado si el propio usuario va a participar en la experiencia de poner fichas sobre la mesa, y el tipo de competición, el motor del juego carga una serie de **plantillas** que el usuario podrá elegir para jugar directamente. Quedan explicadas a continuación:

- ***Classic double-9 (Teams)***: Es el clásico juego en parejas que se juega en las esquinas. Las partidas se juegan entre 4 jugadores, donde los que se sientan a lados opuestos de la mesa conforman los equipos. Se juega con una data máxima de doble-9 donde se reparten 10 fichas a cada jugador de manera pseudoaleatoria. Una ronda se acaba cuando un jugador pone todas sus fichas sobre la mesa o ninguno tiene una ficha capaz de encajar con los extremos de la mesa. ¿quién gana?, pues el jugador que se quedó sin fichas, y en caso de tranque, el que tenga la mano con menos puntos, donde los puntos los conforman la suma de ambos lados de todas las fichas. Una vez hay un ganador, los puntos que este recauda es la suma de las manos de todos los jugadores que se consideren contrarios. En caso de torneo, se ganará la competición cuando se reúnan 100 puntos.
- ***Classic double-9 (Single Player)***: Configuración muy similar a la anterior, solo que esta vez no habrá equipos, es una modalidad Deathmatch pero sobre la mesa.
- ***Classic double-6***: Siguiendo el hilo de una partida clásica, esta vez se juega con una data máxima de doble-6. Con el compañero del frente en el mismo equipo, y respetando la convención antes explicada de la cantidad de fichas a repartir, los jugadores buscan ganar puntos como lo juegan los clásicos chinos.
- ***Crazy Token***: Modalidad propia de Dommie, traemos esta modalidad donde las cosas se salen un poco de lo habitual. Para empezar esta vez se sentarán en la mesa 6 jugadores y con 10 fichas cada uno, aunque esta vez hasta el doble-12; con fichas obtenidas de manera pseudoaleatoria, intentarán ganar una partida quedándose sin fichas en mano o teniendo la menor cantidad de puntos. ¿equipos?, por supuesto, pero esta vez serán dos tríos donde nunca juegan dos compañeros de manera consecutiva. Pero ahora, una peculiaridad es que durante todo el juego, existe una ficha muy particular que será repartida a uno de los jugadores de manera asegurada. Nadie conoce esta ficha ni quien la porta, pero una vez que la ficha se ponga sobre la mesa, la partida terminará inmediatamente. Puede pasar que un jugador nunca llegue a poner la ficha y ninguno, ni siquiera él, lleve una ficha que pueda jugar por uno de los extremos, si esto ocurre la partida acabará igualmente. Los puntos obtenidos para el torneo se computan de igual manera que el juego clásico.
- ***Customize your own template:*** Permite al usuario acceder a un nuevo menú de personalización del juego, a través del cual podrá combinar a placer, todas las opciones de personalización anteriores y hacer del juego de dominó una experiencia más que preferida.

Navegación por los menús :

Para los menús del juego ha sido necesario la creación de clases que permitan un funcionamiento los más amigable posible con el usuario. Durante la navegación por la interfaz gráfica el usuario interactúa con varios tipos de menús. La navegación por los menús se hace mediante las flechas de dirección del teclado,

presionando la tecla **Enter** para confirmar selecciones. Para seleccionar o deseleccionar la opción `Continue`, en los tipos de menú que cuentan con ella, la navegación es mediante las flechas **LeftArrow** y **RightArrow**.

El más común es el `SingleSelectionmenu<T>`, como su nombre indica es un menú de selección simple, que tiene un método llamado `Show()` que no termina hasta que el usuario confirma una selección. Sus opciones de tipo genérico (`GenericOption<T>`) son almacenadas en una lista y durante la navegación muta constantemente un valor que contiene la selección actual. Como una extensión de este menú, se decidió implementar sobre él mismo una opción que mediante un valor booleano, habilita una opción extra llamada `Continue`, que en caso de usarla, es quien da la salida del método. También intervienen otras clases como `QuickScreen`, que muestra un mensaje en consola durante un tiempo determinado.

Quizás el más complejo sea el menú de personalización de plantilla: `CustomizeGame`. Este menú carga unos valores de los parámetros personalizables por defecto para cubrir el caso en que el jugador inicie la partida sin establecer todos los valores. Conociendo ya si el usuario va a poner fichas en juego y el modo de juego (partida o torneo), la configuración iniciaría de la siguiente manera:

Players: 4 (Generados de manera aleatoria)

Teams: No

Max Token: double - 6

Hand Out Judgment: Random (clásico)

Inner Selector: Random

Over Round Condition: Classic

Get Winner Judgment: Classic

Points Getter Judgment: Classic

Tournament Win Score: 100 points

Volvemos a habilitar la opción de escoger si se jugará solo una ronda, o el usuario prefiere un torneo, así que, si al ver la extensión que brindamos para modificar el juego tienes ganas de más, te dejamos cambiarlo.

En el menú de personalización de jugadores no solo se eligen la cantidad de jugadores que rodearán la mesa, sino también las diferentes estrategias que estos van a adoptar durante todo el juego. Eso sí, si el usuario como jugador humano, decide poner fichas en el tablero, entonces no le permitiremos saber cuales serán las estrategias de sus contrincantes.

El menú de los equipos permite elegir la cantidad de equipos que estarán presentes en el juego, teniendo en cuenta cuantos jugadores van a participar. Conociendo siempre cuantos jugadores quedan sin equipo asignado, y la cantidad

de equipos que restan sin jugadores, hacemos del juego que se completen automaticamente los equipos si se acerca a uno de los límites. Además, si juegan solo dos jugadores, pues no tiene sentido la opción de los equipos; ni tampoco permitimos formar la misma cantidad de equipos que de jugadores.

En los menús para elegir las implementaciones de **Hand Out Judgment**, **Inner Selector**, **Over Round Condition**, **GetWinner Judgment** y **Points Getter Judgment**, se utiliza un mismo tipo de menú, donde la opción seleccionada es el delegado.

Para el menú de establecer los puntos con los que se ganará un torneo, utilizamos el `WriteMenu.cs`, el cual computa el `string` introducido por el usuario y lo valida solo si el `string` puede representar a un número entero mayor que cero, y hasta el límite que permite la definición del tipo `int`.

En base a las selecciones anteriores el usuario puede modificar los aspectos del juego. Una vez establecido todo, se selecciona la opción `Continue` y tras confirmar, comienza el juego.

Detalles de la implementación lógica del juego :

Una vez configurada por completo una plantilla, el siguiente paso sería ejecutar el IGame, donde IGame podría ser tanto un torneo como una ronda del juego.

La Ronda:

Ubicada en `Round.cs`. Una ronda es una secuencia de las acciones que toman los jugadores que intentan ganarla. Para la creación de esta es necesario la entrada de cierta información, esto ocurre a través de su constructor que recibe como parámetro un objeto de tipo `RoundSetting`. En el constructor se asignan las fichas con las que los jugadores comenzarán la ronda. Una vez creada la instancia, la manera de ejecutarse la ronda es mediante el método `NextMove()`, que devuelve la colección de diferentes estados de juego por los que atraviesa la misma. Estos estados son definidos por el objeto `GameStatus`, y los mismos son devueltos en la medida en que ocurren las jugadas utilizando `yield return`. Cada jugada crea un nuevo estado de juego que se actualiza respecto al anterior con los nuevos cambios que esta implica. Entonces para generar las jugadas es necesario iterar por los diferentes jugadores; como estos están recogidos en el tipo `CircularList<T>`, la forma en que se definió el `Enumerator` de esta colección de datos hace que el `foreach` sobre ella solo acabe cuando termine la ronda.

La jugada es obtenida por el método `GetPlay()`, encargado de pedir la jugada al jugador que le corresponda el turno. Solo se deja a un miembro de la mesa jugar si tiene al menos una ficha válida de poner por alguno de los extremos. En caso contrario es detectado que el jugador no tiene opción de juego alguna y no se le cede el turno, por tanto la jugada queda predeterminada como un pase. La jugada se maneja como un objeto que implemente `IPlay`, de esta manera una

jugada válida es tanto un pase como una ficha que encaje en los extremos de la mesa. Si un jugador intenta poner 3 veces una ficha inválida en la mesa, su turno se cederá al siguiente jugador y su jugada durante la ronda será un pase.

Una vez devuelta la jugada se actualiza el estado del juego y se comprueba si con esta el juego llega a su fin mediante el criterio establecido, en cuyo caso se devuelve un nuevo estado de juego con una actualización extra del mismo.

El Torneo:

Ubicado en `Tournament.cs`. Un torneo no es más que una secuencia de rondas en la que los diferentes equipos buscan sacar las mejores puntuaciones y así ganar la competición. Su implementación está en el archivo `Tournament.cs`.

Una instancia de `Tournament`, recibe como parámetro del constructor, un objeto de tipo `TournamentSetting`, que contiene los datos necesarios para la inicialización del mismo. Como es propio de los `IGame`, tiene un método llamadado `NextMove()`, el cual se encarga de iterar por las diferentes rondas del torneo y generar estados del juego que se van a ir devolviendo utilizando `yield return`. De esta menera se devuelven los estados de juego que genera cada ronda, además de un estado de juego al finalizar el torneo. Estos estados están definidos por la clase `GameStatus.cs`. Además en ella están definidos métodos propios y específicos del torneo: `IsOver()` determina si dada la puntuación actual y la puntuación necesaria de victoria, el torneo ha terminado; `GetWinner()` extrae el ganador del torneo por las puntuaciones obtenidas.

Detalles de la implementación de la interfaz gráfica , el flujo:

La interfaz gráfica está en la carpeta ConsoleApp. La ejecución del juego comienza con el método `Main()` de la clase `Program.cs` . Su función es darle la bienvenida al usuario y luego llamar a la clase `ConsoleApp.cs`, que inicia con el flujo del juego.

El flujo comienza en el constructor donde se muestran los dos primeros menús, para que el usuario decida si va a participar en el juego, y la modalidad de juego que quiere tomar. Luego se muestra el menú de selección de plantillas (`TemplateMenu`), en este están las plantillas precargadas con todas las configuraciónes necesarias para que el usuario elija una y comience a jugar; donde la última opción permite personalizar tu propia plantilla. La opción seleccionada se devuelve por el `TemplateMenu` y la recibe nuestra aplicación de consola. Llegado a este punto, todo está listo para mostrase al usuario, y se ejecuta el tipo de juego elegido a través de los métodos `RunRound()` o `RunTournament()`.

El método `RunRound()` es el encargado de mostrar al usuario la información de cada estado de juego de la ronda. Los métodos puramente de impresión quedan reunidos en la clase estática `ConsolePrinters.cs`. De esta manera se imprimen los datos relevantes en cada momento del juego, así como cada jugada

de la ronda a medida de que esta va ocurriendo, o el ganador una vez este se concreta.

De manera similar funciona `RunTournament()`. Solo que éste va a tener estados de juego que son puramente de un torneo, y en todo momento muestra la información relevante del mismo. En el caso de los estados de juego que corresponden a las rondas internas del torneo, se muestran de la misma manera que ocurría en `RunRound()`.

Terminado el juego, damos la opción de que, manteniendo la ejecución se pueda volver al inicio y comenzar una nueva partida a través de `PlayAgainMenu()`.

También forma parte de `ConsoleApp.cs` un método que es el encargado de llegar al jugador humano en caso de que exista este. Dicho método es recogido en el delegado `HumanPlayerMenu` y pasado como parámetro al jugador humano para que lo ejecute y permite que el usuario interactúe con el juego. En él se recoge la ficha elegida de la mano del jugador y se permite escoger el lado de la mesa por donde se desea jugar . Utiliza la clase `PlaySelectorMenu.cs` destinada a mostrar al usuario la mano y permitirle escoger la ficha que se desea poner en mesa. La interfaz de esta clase de cara al usuario, la hemos adaptado a que sea lo más cómoda posible para que una persona juegue. En la misma mostramos todas las fichas del jugador, las fichas que están puestas sobre la mesa en forma consecutiva para que encajen de forma adecuada, la lista de los jugadores de la partida, así como sus fichas, sin poder ver sus valores (simula las fichas de cara hacia abajo), y un historial de las últimas jugadas ocurridas (para los que siempre se pierden).

Otras clases que están presentes durante el juego :

`GameStatus.cs`

Es un contenedor de los datos que representan el estado de un juego, ya sea este una simple ronda o un torneo. Contiene todas las propiedades necesarias para brindarle a la interfaz gráfica una forma adecuada de mostrar los resultados al usuario. Tiene los métodos `UpdateRoundStatus()` y `UpdateTournamentStatus()` que se encargan de actualizar la información creando una instancia nueva de su mismo tipo y copiando los parámetros anteriores pero ya actualizados.

`CircularList.cs`

Esta clase es una implementación genérica de una estructura de datos que funciona de manera similar a una lista enlazada mediante nodos. Su principal razón de ser es que implementa la interfaz `IEnumerable` de manera que pasándole un `IEnumerator` por constructor nos permite variar la forma de recorrerla. La idea de su creación fue para realizar el ciclo `foreach` sobre los jugadores de manera circular, es decir, que tras el último jugador aparezca el primero en el ciclo y solo se termine el mismo al llamar a `break`.

`Node.cs`

Es el contenedor de datos de `CircularList<T>`, tal como los nodos de las `LinkedList`

`Enumerator.cs`

Contiene dos implementaciones específicas de `IEnumerator` para que sea utilizado por la `CircularList`. Una implementación clásica que va desde el primer elemento hasta el último, y una implementación circular, que enlaza al último elemento con el primero, como es propio de la lista circular.

`Utils.cs`

Contiene métodos útiles para ciertos momentos de la ejecución:

- `AssignColors()`: Dado un array con los jugadores, devuelve un diccionario donde a cada jugador se le asigna un color de una lista preestablecida de ellos.
- `Lapse()`: Dado un entero que representa una cantidad de segundos, a los ojos del usuario su función es detener el tiempo de ejecución con la cantidad indicada.
- `RandomTokenGenerator()`: Dada una lista de fichas, elige y devuelve una al azar.
- `GetAllTokens()`: Dado un diccionario con jugadores, y por cada jugador una lista de fichas, devuelve una lista nueva con las fichas de todos juntas.
- `GenerateTokens()`: Genera todas las fichas de un juego de domino, dado el doble máximo con que se va a jugar.
- `DecideTokensPerPlayer()`: Devuelve por convención el porciento de las fichas que se decidió.
- `SetTeamsScore()`: Dada una lista de equipos, devuelve un diccionario donde a cada equipo se le asigna una puntuación inicial en 0.

`Token.cs`

Es la clase que engloba el concepto de ficha.

`Token_onBoard.cs`

Define a una ficha una vez puesta en la mesa, una ficha que ahora tiene dueño, una orientación, y el lugar de la mesa por donde fue jugada.

`Pass.cs`

Implementación de `IPlay` que representa una jugada donde el jugador no tiene opciones de ficha válida.

`Team.cs`

Define lo que es un equipo con una lista de jugadores que lo conforman.

`Judge.cs`

Esta clase es la que contiene los delegados que el juego ejecuta para hacer cumplir las reglas. Además tiene un método `IsValid()` que determina si una jugada es válida.

`Setting.cs`

Esta clase es un simple contenedor de información que guarda los datos que necesitan `Round.cs` y `Tournament.cs` a través de `RoundSetting` y `TournamentSetting` que heredan de la clase abstracta `Setting`.

`WinnerStatus.cs`

Contenedor de la información de un ganador de ronda o torneo.

`BoardInfo.cs`

Contiene los datos necesarios sobre el tablero para que los jugadores puedan ejecutar una jugada. En su constructor recibe un `GameStatus` del cual copia los parámetros que necesita a sus propiedades.