**PROYECTO “PROELIUM”**

***MANUAL DE USUARIO*** (aquí va la descripción de la interfaz gráfica, cómo se utiliza, así como en qué consiste el juego)

Proelium, cuya traducción al español es “batalla”, es un juego basado en el famoso Yu-hi-oh! Se trata de un enfrentamiento entre dos jugadores y cuyo objetivo llevar a 0 los puntos de vida del adversario, o dejar a este sin cartas que robar en su mazo personal. Para ello utilizarán durante los enfrentamientos cartas de tipo monstruo y mágicas.

Cada turno está compuesto por una serie de fases: robo, fase principal, fase de batalla y fase final, dispuestas en ese mismo orden. En la primera tiene lugar, de forma automática, el robo de una carta del mazo individual. En la segunda el jugador puede elegir invocar cartas que tenga en su mano y/o activar los efectos de las cartas que estén en el campo. Téngase en cuenta que existe un límite de cartas a invocar en el campo, y que las cartas mágicas tienen una duración efímera en el tablero, es decir, una vez invocadas se activa automáticamente su efecto y acto seguido es enviada al cementerio. En la fase de batalla se realizan los ataques a través de las cartas de monstruo. El jugador activo puede elegir si realiza o no un ataque a una carta de monstruo. En caso de que el oponente no tenga invocadas cartas de tipo monstruo, el jugador activo podrá realizar ataques directos y así restar puntos de vida del adversario. En la fase final se puede invocar cartas y/o activar el efecto de algunas de las barajas que estén invocadas y cuyo efecto no haya sido activado previamente en el turno. En cualquier fase el jugador activo puede decidir finalizar la fase actual o su turno.

(explicar la dinámica de los efectos de las cartas, algunas de las que estén previamente creadas y cómo crear nuevos efectos)

El juego finaliza una vez que alguno de los jugadores (o los dos) se haya quedado sin puntos de vida o sin cartas en su mazo individual en el momento de robar. El 1er jugador en esta situación resultará ser el perdedor y el otro el vencedor. El juego se declara empate si ambos jugadores se quedan sin vidas al mismo tiempo.

**ESTRUCTURA INTERNA DE PROELIUM:**

Proelium utiliza en su interior una biblioteca de clases recogidas en los namespaces ProeliumEngine y (…). El primero contiene todo lo relacionado en el funcionamiento interno del juego como las reglas, estructura de un turno, el tablero, además del jugador virtual, y el segundo (…).

***ProeliumEngine***

Este namespace está compuesto por dos secciones: clases e interfaces. Estas últimas tienen el objetivo de establecer un “contrato” a través de los métodos y propiedades que, como mínimo, deben poseer las clases que las implementen. Además, junto con las interfaces también se encuentran los enums que se emplearon en distintas partes del proyecto.

Tanto las clases como las interfaces pertenecientes a *ProeliumEngine* están agrupadas, para una mejor legibilidad del código, en 3 regiones fundamentales: *Game*, que contiene las clases, interfaces y enums relacionados el tablero, estado del juego, reglas, acciones a realizar, así como el ciclo del juego hasta su fin; *Cards*, que recoge lo relacionado con las cartas que interviene directamente en la dinámica del juego; y por último se encuentra la región *Player* que contiene lo que respecta al jugador virtual y su interacción con el juego.

*REGION GAME;*

*Table:*

La clase *Table* implementa la interfaz *ITable* y representa una abstracción de lo que físicamente debe poseer un tablero utilizado para este tipo de juego: las cartas que han sido invocadas por los jugadores, las del cementerio, la carta de campo, así como los mazos de ambos jugadores. La interfaz garantiza que, para cualquier tablero que se cree para este juego, debe cumplir, al menos, con las características mencionadas previamente.

*Rules:*

La clase *Rules* que implementa la interfaz *IRules*, representa el conjunto de reglas que cada jugado debe seguir para el funcionamiento correcto del juego. La interfaz establece que, como mínimo, un conjunto de reglas adecuadas para Proelium debe considerar los siguientes aspectos: máximo número de cartas por mazo, cartas en la mano de cada jugador, y de puntos de vida; asimismo debe implementar los métodos que indiquen cuándo son válidas una jugada o alguna acción en específico teniendo en cuenta el estado actual del juego (el cual se explicará más adelante); además de implementar los métodos que indiquen cuándo a finalizado el juego y declarar a un ganador, o el empate.

*Actions:*

La clase *Actions* implementa la interfaz *IActions* y contiene todas las posibles acciones que los jugadores pueden realizar durante su turno. Dichas acciones están representadas mediante métodos que se encargan de modificar el estado actual del tablero. Esta interfaz contiene las principales acciones que deben realizarse en este juego: robar una carta del mazo, invocar una carta, activar un efecto, atacar a una carta o a los puntos de vida del adversario, barajar el mazo, descartar una carta de la mano, poner fin a la fase actual o terminar el turno.

*State:*

*State* es una estructura que guarda todos los datos que se necesitan saber del juego, los cuales incluyen desde la información almacenada en el tablero hasta los jugadores que participan, sus “manos”, sus puntos de vida, el número del turno actual, además de la fase del turno en la que se encuentra el jugador activo.

*Game:*

*Game* es la clase en la que tiene lugar el ciclo de un juego entero, la cual implementa un *IEnumerable<State>* por lo que representa una secuencia de estados del juego. Cada nuevo elemento se añade a esta colección una vez que el estado actual es modificado por alguna de las acciones previamente explicadas. Por cada cambio que se efectúe sobre el estado las reglas verifican si se trata del estado final del juego, y de ser así no se añaden más elementos a la secuencia, se pone fin a la partida y se declara un vencedor (o empate).

*REGION CARDS:*

*Cards:*

*Cards* es una clase abstracta de la cual heredan los diferentes tipos de cartas. Contiene la propiedad *Name* (nombre de la carta) y el método *EffectExecute* que deben implementar sus herederos.

*MonsterCard:*

*MonsterCard* hereda de *Cards* y representa a las cartas de tipo “monstruo”, las cuales poseen propiedades adicionales como la vida, el ataque y la defensa.

*MagicCard:*

Esta clase hereda también de *Cards* y representa al tipo de carta “mágica”. Esta contiene dos propiedades adicionales relacionadas con su efecto.

*FieldCard:*

Esta clase hereda igualmente de *Cards* y representa a las cartas de campo, un tipo de cartas mágicas cuyos hechizos tienen efecto sobre todas las cartas presentes en el campo. Para ello posee también dos propiedades más que las de su padre relacionadas con el efecto que realiza.

*REGION PLAYERS:*

*Move:*

Esta clase representa el tipo de la jugada que realiza un jugador. Tiene por propiedades el nombre de la acción a realizar, así como la lista de las cartas que intervienen en dicha acción.

*Player:*

Esta clase representa a un jugador virtual cualquiera, el cual se va caracterizar por un nombre, un ID y una lista de *IStrategy* que representarán las distintas estrategias que dicho jugador empleará para decidir la jugada a realizar.

*Greedy:*

Esta clase implementa la interfaz *IStrategy*, interfaz que se contiene un método *Play* que devuelve un objeto de tipo *Move* y que debe contener en su interior el “análisis” que efectuará para devolver una jugada. En el caso de *Greedy*, como su nombre lo indica, tratará de hacer todas las acciones posibles por cada fase respetando siempre las reglas del juego.

*ENUMS:*

*PhasesEnum:*

Contiene los nombres de todas las posibles fases durante un turno.

*ActionsEnum:*

Posee los nombres de todas las posibles acciones a realizar durante un turno completo.

*MYEXCEPTIONS:*

Esta clase, aunque no está incluida en ninguna de las regiones previamente mencionadas, pertenece a *ProeliumEngine*. La misma contiene métodos que verifican el cumplimiento de varios de los requisitos a cumplir para el correcto funcionamiento del código, de lo contrario lanzará alguna de las excepciones propias de este namespaces. Estas excepciones heredan de la clase *Exception* de C# y están creadas en el mismo .cs que la clase *MyExceptions*.

***Otros Namespaces (…)***