Como se puede apreciar en el diagrama en nuestro proyecto todo parte de tres nodos principales que son Cards, Cards Engine y ConsoleApp que explicaremos detalladamente a continuacion:

1.Cards
Esta es la súper clase de la que heredan los dos tipos de carta protagonistas de nuestro juego (MonsterCards y PowerCards) las cartas de monstruos y las cartas de poder que no son mas que habilidades que tendrán esos monstruos.
Cada carta tiene las siguientes propiedades:
-Nombre

-Tipo: Explicar bonificación de ataque, adjuntar diagrama circulo

-Descripción pública: Es la descripción del efecto de la carta que se mostrara en el tablero, debe ser lo mas explicita y certera posible.

Las cartas de monstruo además de los campos anteriores incluyen:
-Estado: es el estado del monstruo…
-Puntos de ataque
-Puntos de vida

Las cartas de poder contarán con las siguientes propiedades:
-Código: es el código que deberá escribir el creador de la carta. Para su correcta definición debe ser escrito de la siguiente manera:

Condition{
condiciones que se deben cumplir para que la carta se pueda ejecutar}
Actions{
Acciones a realizar si se cumplen las condiciones}

Explicar a detalle las posibles condiciones y acciones.

-Energía de activación: el usuario debe contar con una cantidad igual o superior de energía para poder usar dicha carta.

Para el correcto funcionamiento del programa fue necesario implementar un método Clone() para estos objetos. Mas adelante explicaremos su uso.

2. Cards Engine
Deck
El objeto Deck no es mas que la colección de cartas de monstruo y de poder que usted elija para su juego. Puede ser creado con 3 monstruos y 4 habilidades para cada uno, 12 en total.

Board
Esta clase lleva se compone de dos propiedades:
-Monstruos en el campo: por definición de nuestro juego inicialmente cada jugador contara con tres monstruos siempre en su campo.
-Cartas en la mano: cada jugador comenzara el juego con una cantidad de 5 cartas en la mano y en cada turno robara dos cartas mas de su mazo de 12 habilidades. Cada jugador podrá tener como máximo 7 cartas en la mano.
Game
Esta clase es la que controla el flujo del juego, todo el contexto es almacenado en ella, por ello explicaremos detalladamente sus propiedades:

ADJUNTAR SCREEN DE CODIGODEL CONSTRUCTOR DE GAME

-Jugadores: Esta propiedad esta definida por un array booleano de tamaño n(cantidad de jugadores) que si en un su posición i-esima es true representa a un jugador real, en cambio si es falsa representa a un NPC. Esto es necesario para llevar el control y saber si el juego necesita interactuar con un jugador real o si simplemente ejecuta el código del NPC.

-Decks: Obviamente el juego no puede comenzar si no están definidos todos los decks de los participantes que son necesarios para construir el tablero.

-Turno: Un jugador tiene una sola oportunidad de jugar por turno, y en un turno culmina cuando todos los jugadores hayan jugado.

-Jugador Actual: es el jugador que esta o le toca jugar.

-Puntos de Energía: estos puntos son necesarios para jugar cada carta, a menos que el costo de la misma sea 0. Cada jugador tiene una cantidad de energía independiente. Se inicializa con una cantidad de 200 y cada turno se actualiza aumentando 30 de energía a cada jugador.

-Perdedores: esta propiedad es un array booleano que se inicializa en false y vuelve true la posicion i cuando el jugador i haya perdido.

-Board: esta propiedad ya fue explicada.

-Npcs: en una lista guardamos instancia de todos los npcs del juego.

Esta clase cuenta con métodos modificadores de sus propiedades como:
-UpdateTurn() : Actualiza el turno.
-NextPlayer() : Actualiza el jugador actual.
-TurnDraw() : Es llamado al principio de cada turno para agregar dos cartas nuevas a la mano del jugador acual.
-UpdateEnergy() : Actualiza la energía, es usado tanto cuando un jugador juega una carta como para actualizar la energía al comienzo del turno.
-UpdateLosers() : Actualiza los perdedores.

Otros dos metodos de la clase son:
CanPlay() : este método evalúa el costo de energía de la carta a jugar con la energía del jugador actual, si es posible hacer a jugada devuelve TRUE, de los contrario devuelve FALSE.
PlayCard() : Este metodo se ejecuta cada vez que un jugador decide que carta usar y contra quien. El monstruo objetivo se decide según la formación del jugador objetivo, el primer monstruo vivo sera seleccionado. Posteriormente se ejecuta el efecto de la carta, se verifica si el monstruo atacado murio, se actualiza la energia y se remueve la carta usada de la mano.

Npc
Los objetos de tipo Npc solo contarán con un indice para identificarlos.
PlayTurn() : es el método que se invoca cuando le toca jugar al Npc.

La jugada del Npc se procesa con los métodos:
Potencia() : este método recibe la mano del Npc como parámetro y crea todos los conjuntos potencia de cartas posibles, podando por el requerimiento de energía. Posteriormente es llamado el método…

Permutation() : este método permuta cada una de los conjuntos potencia y con cada una de las permutaciones llama al método...
(Ambos metodos Potencia() y Permutation() fueron implementados con BackTracking)
EvaluateBetterPlay() : aquí se clona la instancia de Game clonando cada una de sus propiedades y se juegan todas las cartas de la permutacion que se recibió, se evalúa el resultado y finalmente se escogerá la combinación de cartas que mas daño haya causado al jugador objetivo.

Engine
Esta clase es estática y fue necesaria para implementar métodos auxiliares.:
LoadCards() : esta función carga todas las cartas desde archivos tipo JSON y las almacena en dos listas, definidas también en esta clase.
SaveMonsterCard() y SavePowerCard() son utilizados para guardar permanentemente las cartas creadas por cualquier jugador.
GetInitialHand() : solo es utilizada para repartir la mano inicial de 5 cartas a cada jugador.
Draw() : simplemente toma una carta ramdon del deck.
PlayerWins() : verifica si un jugador gano, se ejecuta al final de cada turno.
PlayerLose() : determina si un jugador perdió, esto sucede cuando todos sus monstruos han muerto. Se ejecuta después de cada ataque.
MonsterDied() : determina si un monstruo murió, se ejecuta a la hora de realizar un ataque.

Métodos del lenguaje…
…