Reporte del Proyecto

Gwent Cards Game.

Partes Principales del Proyecto:

1-Jerarquia de Clases y Abstracciones Utilizadas

2-Interfaz Grafica

3-Interprete

4-Jugadores

Estudiantes:

-Carlos Manuel Garcia Rodriguez C113

-Carlos Antonio Breso Sotto C113

-Ernesto Rousell Zurita C113

A modo de Introducción me gustaría decir que este proyecto esta basado en un juego con el mismo nombre “Gwent The Witcher Cards Game”, esta desarrollado en el lenguaje C# y para la interfaz usamos el Motor Grafico de Unity.

No voy a entrar en detalles sobre las reglas del juego pero a groso modo es un juego que se basa en ganar dos de tres rondas a partir de puntos, puntos determinados por las cartas de tu campo, cada carta tiene un efecto que se activa al ser convocada.

**Jerarquía de Clases**

**. Class Card:** Una clase que define como tal un objeto de tipo carta, que es la base del juego, esta clase tiene las propiedades de las cartas según las reglas del juego:

-**Nombre y Descripción**: propiedades con único propósito de mostrarse en la interfaz para q el usuario las lea.

- **Poder**: una propiedad muy importante en la carta y en la q se basa el juego.

-**UrlImage**: una propiedad que almacena el path de la imagen de la carta.

-**EfectType**: un enum EfectType que almacena los tipos de efecto, que son: NoTarget, TargetEnemy y TargetAllie.

-**Efect:** almacena un string con el efecto de la carta escrito en un lenguaje definido por nosotros el cual el intérprete es el encargado de ejecutar.

**. Class Move:** Move es una clase que define una jugada del juego como tal, esta compuesta por un **Card** y duplas de entero que señalan posiciones en el campo, para hacer referencias a donde se quiere jugar una carta o que carta quieres afectar con un efecto especifico.

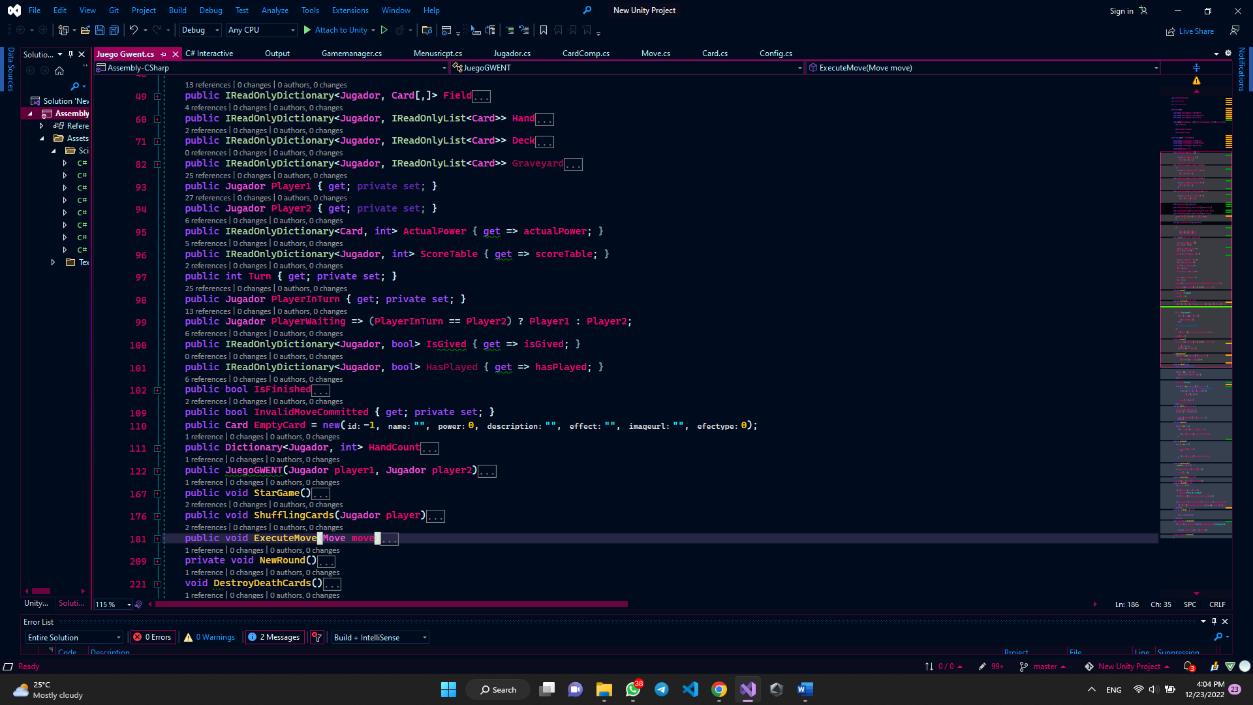
**. Class Jugador:** Una clase Abstracta que define como tal la abstracción de lo que es un jugador. Tiene un método Play, el cual es el encargado de decidir como el jugador juega. Este método Play devuelve un Objeto de tipo **Move**

De jugador heredan dos clases: **JugadorHumano** y **JugadorIA**, cada uno con su propia implementación del método Play.

El **JugadorHumano** como tal está ligado a Unity por lo que tiene elementos de la Interfaz, los cuales veremos más adelante.

El **JugadorIA** dentro del método Play tiene como tal una estrategia que se basa en las cartas en la mano, los puntos en el campo y los tipos de efecto de las cartas para decidir que es lo q va a jugar

**.** **Class JuegoGwent:** Esta clase es como tal lo principal del juego, es la abstracción q define el juego y lo controla, y el único que tiene permitido modificar su estado. Esta clase se construye con dos objetos de tipo **Jugador** y esta clase hereda de un IEnumerable<**Move**>. Donde su **GetEnumerator**, ósea cuando se pide la siguiente jugada esta dado por los métodos Play de los jugadores.



En la imagen anterior se muestran algunas de las propiedades de la clase JuegoGwent, su nombre deja bastante claro su funcionalidad, es decir en su mayoría son diccionarios que relacionan un jugador con su mano o con su mazo, booleanos que evalúan si el juego ya se terminó o que jugador es el que está en turno en ese momento etc.

Además de los métodos propios o necesarios para el juego, díganse:

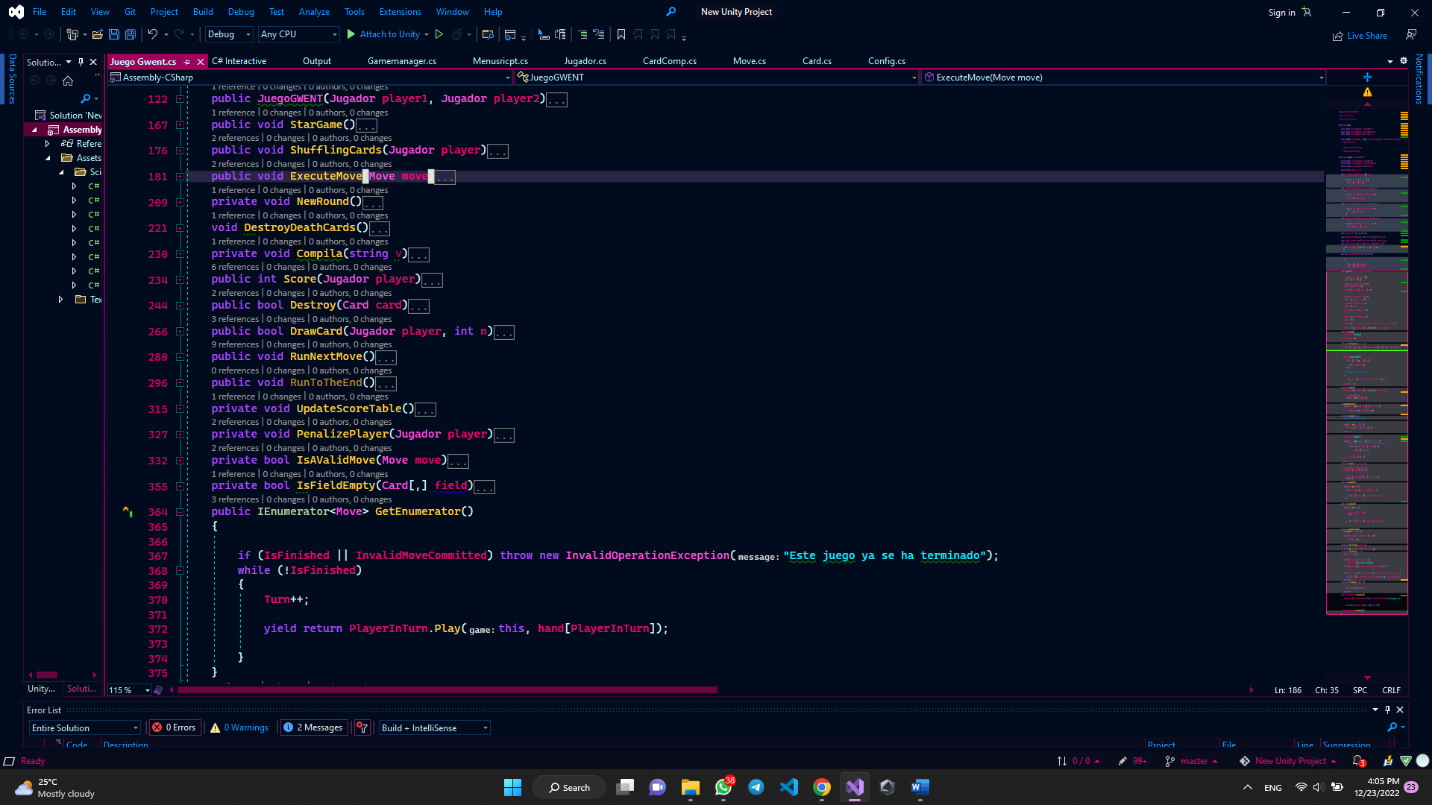
**DrawCard, DestroyDeathCards, ShufflingCards, New Round, StartGame,** etc. Sus funcionalidades creo que están bastantes claras también.

Entre dichos métodos los más destacados son:

**RunNextMove**: este método es el que se encarga de iterar por el innumerable.

**InvalidMoveCommited**: Revisa si la jugada devuelta por el jugador en turno en la siguiente iteración del juego es válida según las reglas.

Aquí están el resto de los métodos y el código del **GetEnumerator** del Juego.



**Interfaz Gráfica**

Como mencione anteriormente la Interfaz esta hecha con Unity, un motor de juego que para nuestra comodidad usa C#.

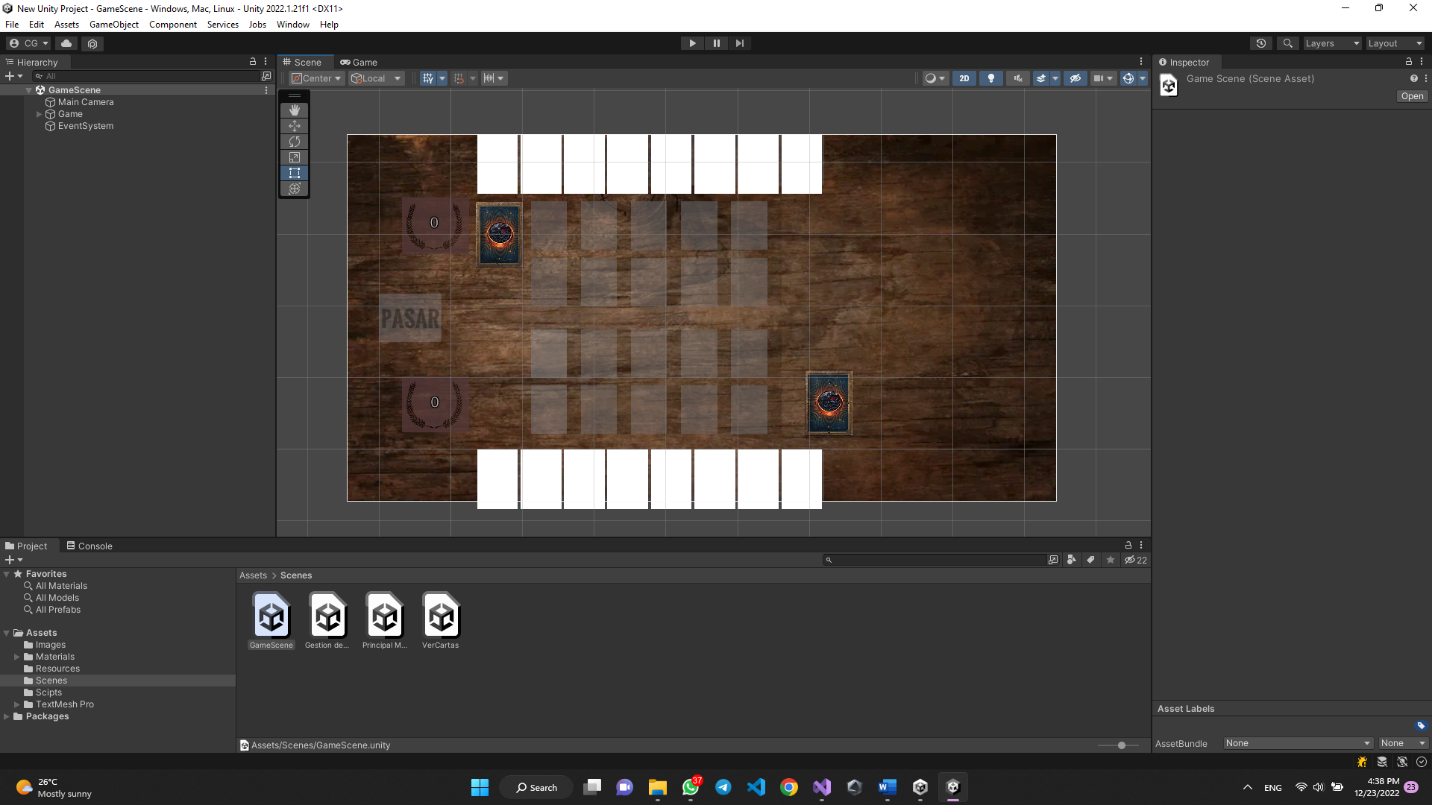
Primero que todo para la Interfaz cree 3 clases, no tienen ningún propósito de definir abstracciones, simplemente es para tener el código mas organizado. Estas son:

**GameManager**: la clase principal donde esta el cuerpo de la Interfaz.

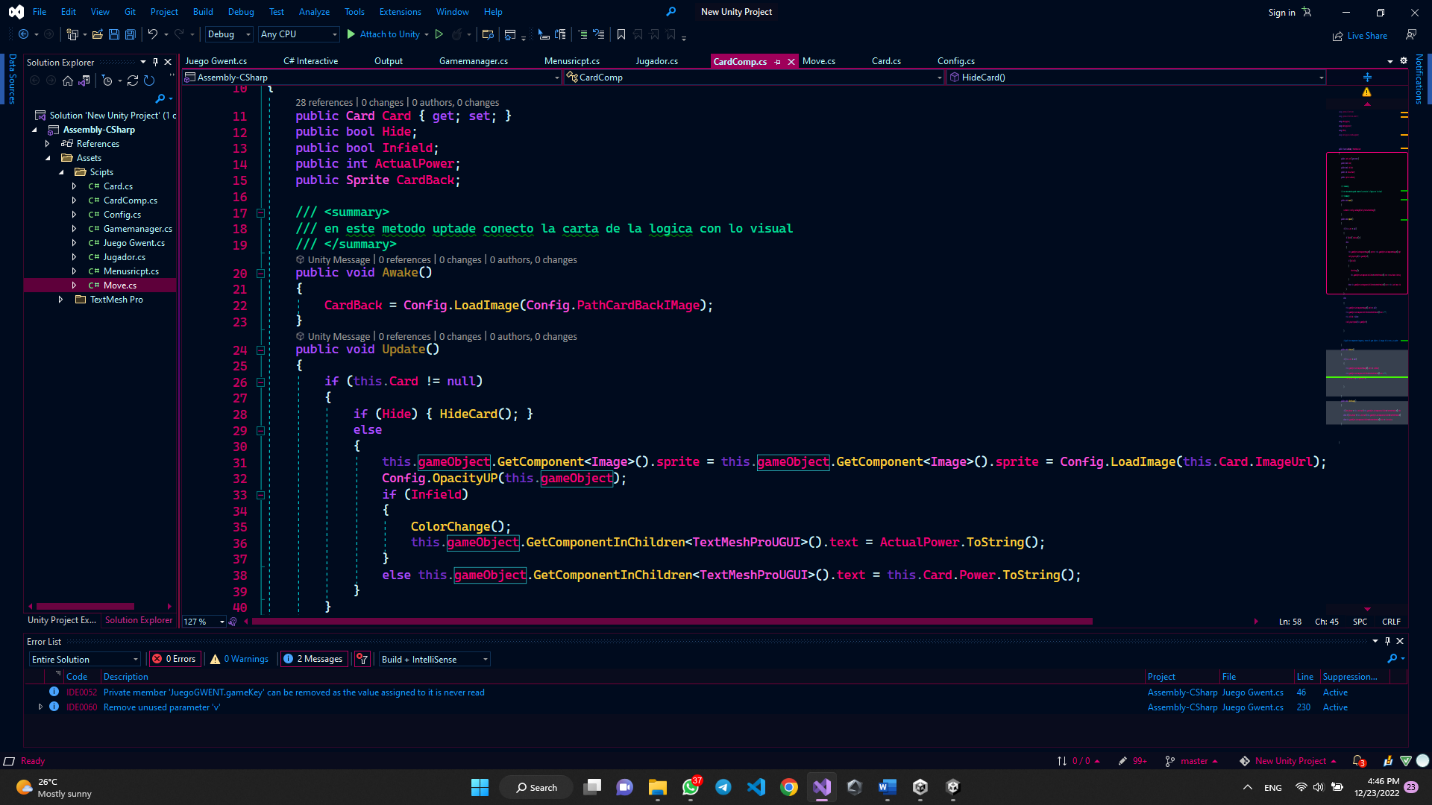
**Config**: Una clase menor donde tengo métodos que interactúan con la base de datos y objetos exteriores al proyecto.

**MenuScript**: Una clase que es la que ejecuta los métodos de navegación por la aplicación.

Y la clase mas importante para la implementación de la interfaz es **CardComp.**



Aquí podemos ver la Scene Principal del juego la jerarquía de la interfaz esta dada por un Canvas principal, un Menú de Control, un Menú de Información, el Battlefield, y las Manos, todos hijos del Canvas. Como tal el Battlefield y los Hands, son arrays de GameObject los cuales tienen como componentes un script de tipo **CardComp, el** cual los hace comportarse como un **Card.**



Este es parte del Código de **CardComp,** cada **GameObject** carta tiene un componente imagen y un hijo de componente TextMesh, este **CardComp** tiene uno de sus campos un objeto **Card** y en el método **Update** de este script simplemente igualo los componentes de este **GameObject** a las propiedades del objeto **Card** que representa este **GameObject**. Así puedo tener actualizado lo que se muestra en tiempo real indiferentemente de los cambios que se hagan.

Ahora GameManager es la clase principal de la interfaz, aquí es donde tengo declarado todos los **GameObjetcs**, Panels, Images, y Textos de la interfaz además de los métodos encargados de “**pintar**” el estado del juego.

Unity tiene tres métodos Principales para cada Script:

**Awake**: es lo primero q se ejecuta al inicializarse la scene.

**Start**: Al terminar de cargar la scene empiezan a ejecutarse los métodos **Start.**

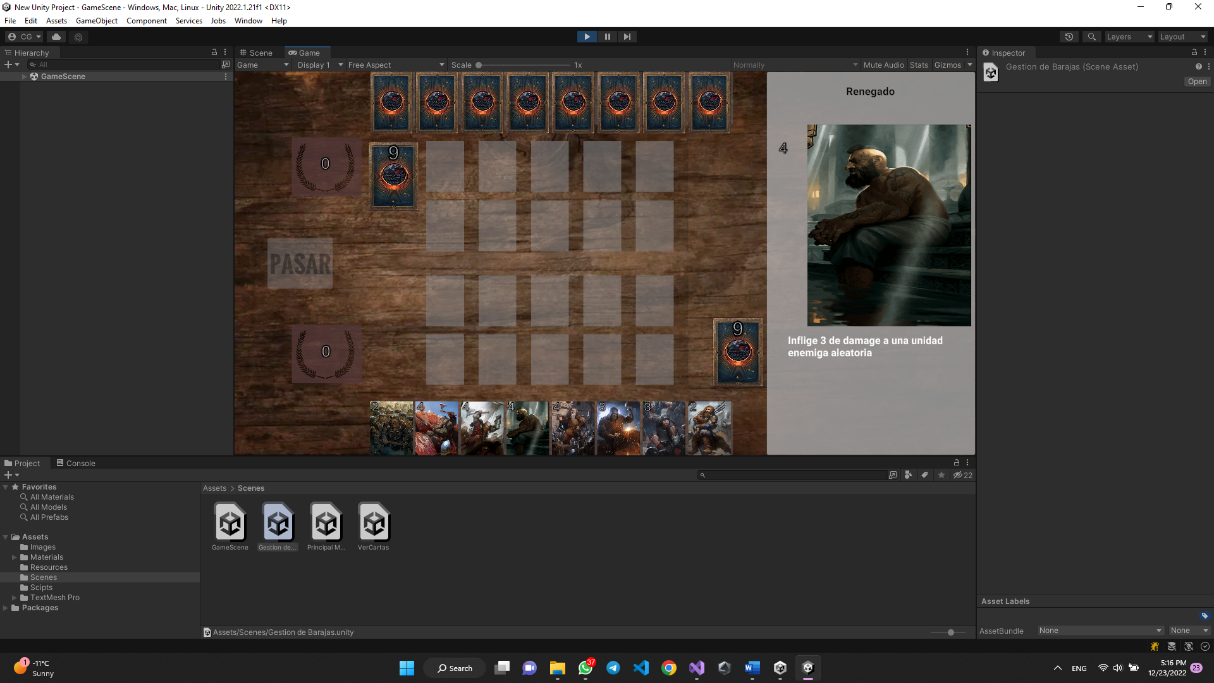
**Update**: es un método que se ejecuta en cada frame de la aplicación.

Entonces en **Gamemanager** en **Start** se crea una nueva instancia de JuegoGwent que es el juego que se va a jugar. En este proyecto la interfaz esta completamente desconectada de la lógica del juego para que los cambios en una u otra parte no afecten a lo demás. Entonces entre los métodos principales de **GameManager** están:

-**UpdateField**: Accede al campo de la instancia del juego e iguala el **CardComp** de los **GameObject** de la visual con las cartas del campo en la lógica, para mostrarlas.

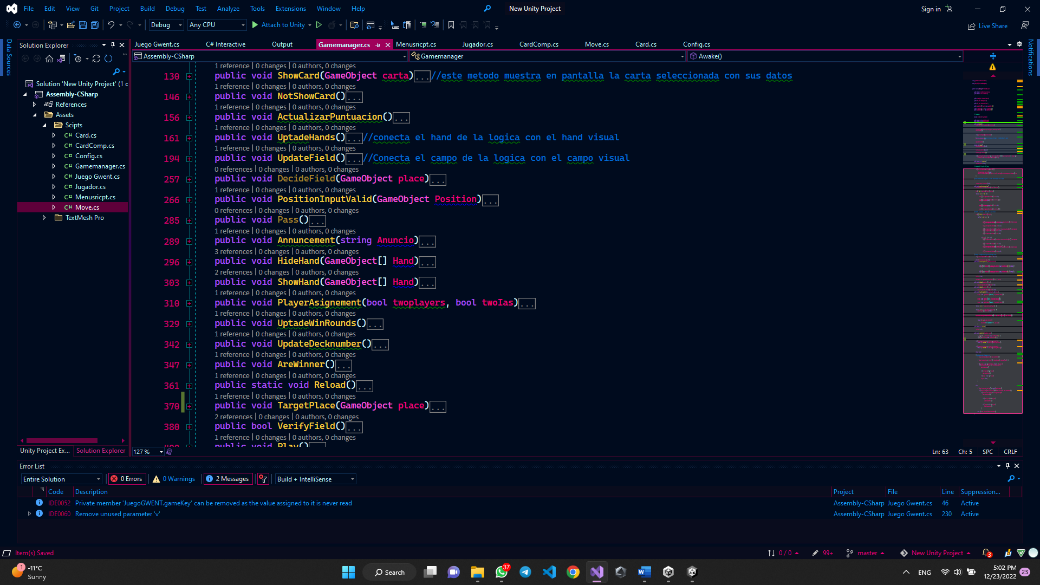
-**UpdateHands**: Hace exactamente lo mismo que **UpdateField,** pero con las manos de los jugadores.

-**DecideField**: este método es llamado cuando se toca un GameObject del **BattleField** visual, y dependiendo de si esta vacío o si tiene una carta ya es el código que se ejecuta.

**-ShowHand**: este método es puramente visual, si tocas una carta esta se mostrara en un Panel que normalmente esta desactivado para mostrarte en detalle la información de dicha carta como se muestra en la siguiente imagen: 

-**Select**: este método se activa también cuando se toca una carta de la mano, y almacena cual carta se selecciono para usar esa información después cuando el **JugadorHumano** vaya a ejecutar una jugada.

A continuación, el resto de los métodos de **GameManager**:



Ahora el **JugadorHumano** es el único elemento que tiene cosas de la interfaz. Y es que mediante las acciones del usuario se van llenando variables de la interfaz y cuando el **JugadorHumano** es llamado a jugar por el **GetEnumerator** de **JuegoGwent** construye un **Move** con esas variables y lo devuelve.

Por último, en la Interfaz cabe mencionar la clase Config, aquí hay pocos métodos importantes los cuales son:

-**LoadDeck**: implementa un método de la clase **JsonUtility** de Unity el cual es utilizado para cargar los mazos utilizados por los jugadores desde el archivo **json** donde se encuentran.

-**LoadImage**: este método es el que se encarga de cargar las imágenes a partir de su path para mostrarlas en el juego, usa un método propio de Unity que se llama **Load** y es de la clase **Resources** de Unity. Esto permite cargar cualquier archivo que este dentro de una carpeta determinada con el tipo que se desee.

**Crear Cartas**

Una de las ordenes del proyecto era que los usuarios fueran capaces de crear cartas con efectos que ellos decidieran en un lenguaje creado por nosotros. La forma de crear una carta nueva es accediendo a los **json** que están creados en la carpeta **Resources**, cada json tiene el nombre del mazo al que corresponden, ahí escriben las propiedades de la carta que quieran crear y la imagen que le quieran asignar a la carta la guardan en la carpeta **Resources** del proyecto y el campo de la carta ImageUrl es simplemente el nombre del archivo imagen, el campo **efect** lo deben escribir según los parámetros del lenguaje descrito en la sección del interprete más abajo. Por último, el parámetro efectype se debe llenar con un numero del 0 al 2 dependiendo si el efecto es TargetEnemy, TargetAllie o NoTarget respectivamente.

**Interprete**