Gwent-pro

Guía de desarrollo

Antes de leer este documento se recomienda leer la Guía de juego, que se encuentra en este mismo directorio, para entender mejor las reglas sobre las cuales el funcionamiento de este juego está construido.

Introducción

Gwent-pro es una juego de cartas digitales para dos jugadores que enfrenta a dos ejércitos en un campo de batalla. Es una aplicación, desarrollada con tecnología .NET Core 8.0,usando Unity como framework para la interfaz gráfica, y en el lenguaje C#.

Descripción general del juego

Gwent-pro es un juego en 2D, con una escena principal: el campo de batalla donde se lleva a cabo la partida.

En cada turno, los jugadores colocan una carta en el campo de batalla. Al final de cada ronda, se suma la fuerza total de todas las cartas de unidad desplegadas en el campo de batalla de cada bando. El jugador con la fuerza total más alta gana la ronda. El primer jugador que gane dos rondas gana la partida.

Para su ejecución, la aplicación se ha dividido en tres componentes fundamentales a nivel de desarrollo: la lógica del juego que se implementó en una biblioteca de clases separada de el segundo componente, que sería la interfaz gráfica. El tercer componente sería un conjunto de scripts a los que nos referiremos como “Managers”, que actúan como “puente” para la comunicación entre la lógica del juego y la interfaz gráfica.

La lógica del juego maneja la información de las cartas, sus posiciones en el tablero, los datos del jugador, los puntos en cada turno, los efectos y cómo interactúan con las cartas, etc. La interfaz gráfica muestra estos datos al jugador y le permite interactuar con las cartas y el tablero según las reglas del juego.

Implementación de la Lógica del Juego

La lógica del juego se implementó en una biblioteca de clases que se encuentra en la ruta “Assets/GwentLibrary”.

GwentLibrary implementa clases para las cartas, el tablero y los jugadores.

La Carta

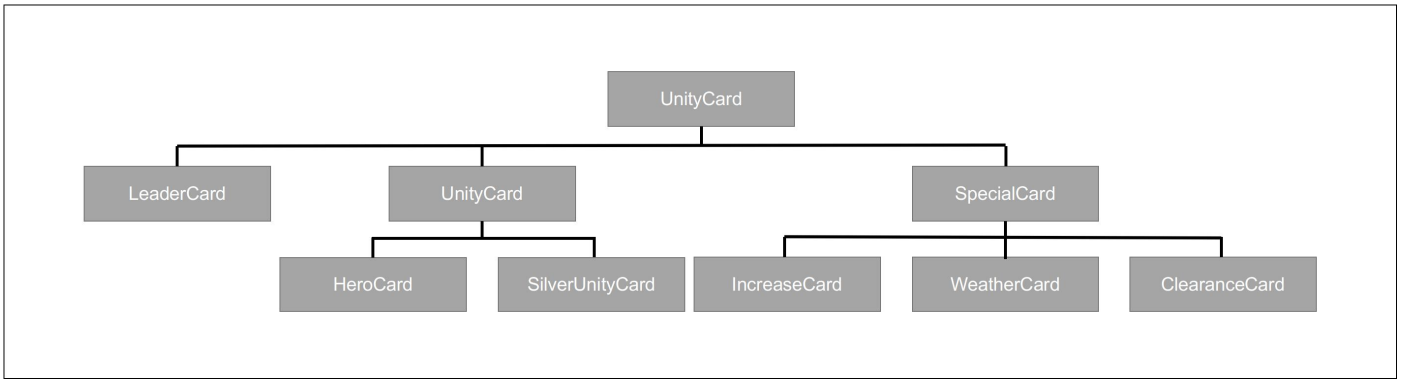
La carta es el elemento principal del juego.

Su información (nombre, puntos, descripción, facción, etc) se encuentra almacenada en archivos de texto plano (txt) en la ruta “Assets/Resources”. Cada archivo representa una carta, y la información en su interior tiene la siguiente distribución:

< letra que identifica el tipo de carta > ‘\n’
< nombre del personaje > ‘\n’
< nombre de la faccion a la que pertence > ‘\n’
< posicion en el campo de batalla > ‘\n’
< valor de fuerza > ‘\n’
< descripcion del efecto de la carta> ‘\n’
< numero correspondiente al efecto > ‘\n’
< descripcion del personaje > ‘\n’
< cita del personaje> ‘\n’

La clase CardsCreator es la encargada de leer la información contenida en cada archivo de texto y la transforma a una estructura de datos que será utilizada en el constructor de la carta como objeto.

La clase Card es la clase padre de la jerarquía de todos los tipos de carta que existen en el juego. La jerarquía de los tipos de cartas tiene la siguiente estructura:



Las cartas son creadas en la clase CardsCollection la cual representa la colección de todas las cartas que existen en el juego. Este es uno de los primeros pasos en la ejecución del juego, ya que de esta colección de cartas se extraerán las cartas para el mazo del jugador y posteriormente para su mano.

En el interior del constructor de la clase CardsCollection las cartas se crean desde las clases finales, es decir desde el nivel más bajo de la jerarquía de la herencia, y son almacenadas en una lista de cartas que tiene un objeto CardsCollection como propiedad.

El mazo

El mazo de un jugador está compuesto por un conjunto de cartas que deben pertenecer a la misma facción que el líder o ser neutrales. Un mazo debe tener al menos 25 cartas para poder jugar con él.

La clase DeckCreator representa al mazo mientras está siendo creado. Es aquel mazo al que le puedes añadir, retirar cartas y duplicar las cartas que son Unidades de Plata. Una vez se ha terminado de conformar el mazo la lista de cartas representada por la propiedad CardDeck es el mazo que utilizará el jugador durante la batalla.

La mano

La mano del jugador está representada en la clase Hand, la cual contiene dos propiedades principales PlayerHand que representa la lista de cartas en la mano del jugador, y GameDeck que es el mazo desde el cual la mano obtiene las cartas. Este último es tomado de la propiedad CardDeck mencionada anteriormente en la sección del mazo.

El campo de batalla

El campo de batalla está representado en la clase PlayerBattlefield, sin embargo, esta clase solo representa a la parte del campo sobre la que puede operar el jugador, es decir: las filas de combate Cuerpo a Cuerpo (M), combate a Distancia (R) y combate de Asedio (S) y las casillas de los aumentos correspondientes a cada fila mencionada anteriormente. En el caso de la fila de los climas, esta está representada por una propiedad estática, ya que ambos jugadores poseen exactamente la misma fila de climas y resulta conveniente que los cambios que haga un jugador sobre la ella, se vean reflejados en ambos lados del campo de batalla.

El jugador

El jugador es una parte fundamental de la lógica del juego, es quien tiene acceso a modificar y manipular la mayoría de los objetos que hemos mencionado hasta ahora. El jugador está representado en la clase Player.

Patrón de diseño Observer

El patrón de diseño Observer fue implementado para permitir la comunicación entre la lógica del juego y la parte visual.

El patrón de diseño Observer es un patrón de comportamiento que permite a un objeto conocido como sujeto, mantener una lista de sus dependientes, llamados observadores, y notificarles automáticamente sobre cualquier cambio de estado, generalmente mediante la llamada a uno de sus métodos. Es también conocido como Publicación-Suscripción o Modelo-patrón.

La idea principal es que el sujeto, no necesita saber quiénes son los observadores, solo que deben ser modificados. Esto desacopla los objetos y permite que interactúen sin formar dependencias rígidas.

Este patrón fue muy útil en este proyecto, donde un cambio en el estado del tablero debe desencadenar cambios en otros objetos, y especialmente cuando no se sabe de antemano, cuántos objetos necesitarán cambiar ante un movimiento del jugador.

En nuestro caso, el **sujeto** es lo que denominamos Game Manager, que en términos simplistas es un objeto vacío en la escena de el campo de batalla con un Script titulado GameManager.

El Game Manager es el objeto central que mantiene una lista de observadores (de los que se detallará un poco más adelante). El Game Manager gestiona los movimientos realizados por el jugador, y en dependencia de cuáles sean, se encarga de notificar a todos los observadores, además de manejar el sistema de turnos y rondas en el juego.

La clase GameManager hereda de una clase llamada Subject, cuya implementación está accesible en la ruta "Assets/Scripts/Assembly/ObserverPattern/Subject.cs".

Los **observadores** son los elementos de la interfaz gráfica, separados y controlados mediante dos Scripts denominados UIAudible y UIVisual que se encargan de manejar el sistema de audio y gran parte de la interfaz visual durante la partida respectivamente. Estos objetos son notificados sobre las acciones que realiza el Game Manager, y actúan acorde a estas para mostrar al usuario aquello que ocurre durante la partida. Los observadores implementan una interfaz denomidada IObserver (puede encontrarla en la ruta "Assets/Scripts/Assembly/ObserverPattern/IObserver.cs".

En términos generales, los observadores se registran con el sujeto para recibir actualizaciones. El juego está en un estado de "quietud" hasta que el jugador activo en ese momento hace un movimiento válido. En ese momento el Game Manager realiza el movimiento correspondiente en la lógica del juego y gestiona las consecuencias de dicho movimiento, luego de esto llama a un método de notificación que a su vez invoca un método específico para los observadores registrados. Llevando el resultado final a los ojos del usuario.

El objetivo de la implementación del patrón fue mantener el juego actualizado sin la necesidad de verificar constantemente el estado en cada frame, con el objetivo de ahorrar recursos y mejorar el rendimiento del juego.

Los scripts mencionados en esta sección se encuentran en la ruta "Assets/Scripts/Assembly/" junto al Script que transporta los datos de los jugadores de una escena a otra durante el juego.

Integración con Unity

El resto de los scripts que manejan la interfaz visual durante el juego están en la ruta “GwentPro/Assets/Scripts/UI scripts”. Estos no serán detallados en esta guía, pues sus aportes al funcionamiento del juego, si bien son importantes, son bastante simples.

Sí es necesario mencionar que el “desencadenante” de las acciones que realiza el Game Manager son los movimientos del jugador, validados por los Scripts que manejan el sistema de Drop de las cartas, el botón de pasar y un componente botón añadido al lugar del líder en el tablero.