Informe de Champions Gwent

Ernesto Javier Govea Varona CC112

Índice

1. Golden_Card

El script `Golden_Card` maneja las acciones de las cartas doradas en el juego. Aquí están las funcionalidades clave:

Variables:

- Variables públicas para el nombre de la carta, el valor de ataque y la posición de la fila en la que se invocará.
- Una variable booleana para verificar si la carta ha sido invocada.
- Una variable para hacer referencia al componente MANOS.

Métodos:

- `Start()`: Método que se llama al inicio del juego. Asigna el componente MANOS del objeto padre a la variable manos.
- `OnMouseDown()`: Método que se llama cuando se hace clic en la carta.
  - Verifica si es el turno del jugador y si no ha jugado ninguna carta o si es posible convocar.
  - Llama al método `Invocar_GoldCard` en el componente manos, pasándose a sí misma como argumento.
  - Llama al método `ActualizarTextoSumaAtaque` en el componente manos para actualizar los puntos de ataque.
  - Incrementa el contador de cartas jugadas.
  - Si el nombre de la carta es "Messi", aplica un efecto especial llamando al método `Efecto_Messi`.

2. Silver_Card

El script Silver_Card maneja las acciones de las cartas de plata en el juego. Aquí están las funcionalidades clave:

Variables:

- `Name`: Nombre de la carta.
- `Atk`: Valor de ataque de la carta.
- `Posicion_Fila`: Posición de la carta en el campo de juego.
- `manos`: Referencia a la clase MANOS.
- `efecto_activado`: Booleano que indica si el efecto de la carta está activado.

Métodos:

- `Start()`: Inicializa la referencia a la mano del jugador (MANOS).
- `OnMouseDown()`: Maneja la interacción cuando la carta es clickeada. Invoca la carta si es el turno del jugador y aún no ha jugado ninguna carta o si tiene la posibilidad de convocar. Actualiza el texto con la suma de ataque y aumenta el contador de cartas jugadas.

3. Weather_Card

El script Weather_Card maneja las acciones de las cartas climáticas en el juego. Aquí están las funcionalidades clave:

Variables:

- `Name`: Nombre de la carta.
- `Posicion_Fila`: Posición de la carta en el campo de juego.

- `disminuye_puntos`: Puntos que disminuyen a las cartas afectadas.
- `cartas_afectadas`: Lista de cartas de plata afectadas por la carta climática.
- `manos`: Referencia a la clase MANOS.

Métodos:

- `Start()`: Inicializa la referencia a la mano del jugador (MANOS).
- `OnMouseDown()`: Maneja la interacción cuando la carta es clickeada. Invoca la carta si es el turno del jugador y aún no ha jugado ninguna carta o si tiene la posibilidad de convocar. Llama a efectos climáticos específicos según el nombre de la carta.

4. Aumento_Card

El script `Aumento_Card` maneja las acciones de las cartas de aumento en el juego. Aquí están las funcionalidades clave:

Variables:

- Variables públicas para el nombre de la carta y la posición de la fila en la que se invocará.
- Una lista de cartas afectadas por esta carta de aumento.
- Una variable para hacer referencia al componente MANOS.

Métodos:

- `Start()`: Método que se llama al inicio del juego. Asigna el componente MANOS del objeto padre a la variable manos.
- `OnMouseDown()`: Método que se llama cuando se hace clic en la carta.
  - Verifica si es el turno del jugador y si no ha jugado ninguna carta o si es posible convocar.
  - Llama al método `Invocar_Aumento_Card` en el componente manos, pasándose a sí misma como argumento.
  - Incrementa el contador de cartas jugadas.
  - Aplica efectos especiales según el nombre de la carta llamando al método `Efecto_Aumento` en el componente manos.
  - Si ya ha jugado una carta y no puede convocar más, muestra un mensaje de error en la consola.
  - Si no es el turno del jugador, muestra un mensaje de error en la consola.

5. Despeje_Card

El script Despeje_Card gestiona las acciones de las cartas de despeje en el juego. Aquí están las funcionalidades clave:

Variables:

- `Name`: Nombre de la carta.
- `Posicion_Fila`: Posición de la carta en el campo de juego.
- `manos`: Referencia a la clase MANOS.
- `invocada`: Booleano que indica si la carta ha sido invocada.
- `destruida`: Booleano que indica si la carta ha sido destruida.

Métodos:

- `Start()`: Inicializa la referencia a la mano del jugador (MANOS).
- `OnMouseDown()`: Maneja la interacción cuando la carta es clickeada. Si la carta ya ha sido destruida, muestra un mensaje. Si no ha sido invocada, la invoca, aplica el efecto de despeje y marca la carta como invocada. Aumenta el contador de cartas jugadas. Si no es el turno del jugador o ya ha jugado una carta y no puede convocar más, muestra los mensajes correspondientes.

6. Distancia_Fila

El script `Distancia_Fila` gestiona las posiciones y cartas en la fila de distancia. Aquí están las funcionalidades clave:

Variables:

- `public List<GameObject>position_distancia_fila = new List<GameObject>()`: Declara una lista pública de `GameObject` llamada `position_distancia_fila`. Esta lista se utiliza para almacenar las posiciones específicas en la fila de distancia en el campo de juego.
- `public List<GameObject>Cartas_Fila_Distancia = new List<GameObject>()`: Declara una lista pública de `GameObject` llamada `Cartas_Fila_Distancia`. Esta lista se utiliza para almacenar las cartas que se colocan en la fila de distancia.

Métodos:

- `public void ColocarEnDistanciaFila(GameObject carta)`: Declara un método público llamado `ColocarEnDistanciaFila`, que toma como parámetro un `GameObject` llamado `carta`. Este método se utiliza para colocar una carta en una posición específica en la fila de distancia.
  - La carta se coloca en la primera posición disponible en la fila de distancia.
  - Si no hay posiciones disponibles, se imprime un mensaje de error en la consola.
- `public void RemoverDeDistanciaFila(GameObject carta)`: Declara un método público llamado `RemoverDeDistanciaFila`, que toma como parámetro un `GameObject` llamado `carta`. Este método se utiliza para remover una carta de la fila de distancia.
  - La carta se remueve de la lista de cartas en la fila de distancia.
  - Si la carta no está en la lista, se imprime un mensaje de error en la consola.

7. Cementerio

El script `Cementerio` gestiona las cartas que han sido descartadas o destruidas y colocadas en el cementerio del juego. Aquí están las funcionalidades clave:

Variables:

- `Cartas_en_Cementerio`: Lista de `GameObject` que representa las cartas actualmente en el cementerio.

8. DECKS

El script `DECKS` se encarga de gestionar las manos de los jugadores en el juego de cartas. Aquí están las funcionalidades clave:

Variables:

- `Hand`: Referencia a la clase `MANOS`, que representa la mano del jugador.
- `deck`: Lista de `GameObject` que representa las cartas en el mazo.
- `listaBooleana`: Lista de booleanos que indica si una posición en la mano está ocupada o no.

Métodos:

- `Start()`: Se ejecuta al inicio del juego. Baraja el mazo y roba 10 cartas iniciales.
- `ShuffleDeck()`: Baraja las cartas en el mazo usando un algoritmo de intercambio aleatorio.
- `RobarCarta()`: Toma una carta del mazo y la añade a la mano en la primera posición libre disponible. Si todas las posiciones están ocupadas, muestra un mensaje de advertencia.

9. Mano_Controller

El script `MANOS` gestiona las manos del jugador. Aquí están las funcionalidades clave:
Variables:

- `public int CartasJugadas`: Un entero público llamado `CartasJugadas` que rastrea el número de cartas jugadas.
- `public bool es_mi_turno`: Un booleano público llamado `es_mi_turno` que indica si es el turno del jugador.
- `public void Invocar_GoldCard(Gold_Card goldCard)`: Declara un método público llamado `Invocar_GoldCard` que toma un parámetro de tipo `Gold_Card`. Este método invoca una carta dorada.
  - Si la carta ya ha sido invocada, muestra un mensaje de error en la consola.
  - Si la carta no ha sido invocada, la coloca en la fila correspondiente y marca la carta como invocada.
  - Llama a `ActualizarTextoSumaAtaque()` para actualizar el total de puntos de ataque.
- `public void Invocar_SilverCard(Silver_Card silverCard)`: Declara un método público llamado `Invocar_SilverCard` que toma un parámetro de tipo `Silver_Card`. Este método invoca una carta de plata.
  - Si la carta ya ha sido invocada, muestra un mensaje de error en la consola.
  - Si la carta no ha sido invocada, la coloca en la fila correspondiente y marca la carta como invocada.
  - Llama a `ActualizarTextoSumaAtaque()` para actualizar el total de puntos de ataque.
- `public void Invocar_WeatherCard(Weather_Card weatherCard)`: Declara un método público llamado `Invocar_WeatherCard` que toma un parámetro de tipo `Weather_Card`. Este método invoca una carta climática.
  - Si la carta ya ha sido invocada, muestra un mensaje de error en la consola.
  - Si la carta no ha sido invocada, la coloca en la fila correspondiente y marca la carta como invocada.
  - Llama a `Efecto_Climatico()` para aplicar el efecto climático de la carta.
- `public void Invocar_Aumento_Card(Aumento_Card aumentoCard)`: Declara un método público llamado `Invocar_Aumento_Card` que toma un parámetro de tipo `Aumento_Card`. Este método invoca una carta de aumento.
  - Si la carta ya ha sido invocada, muestra un mensaje de error en la consola.
  - Si la carta no ha sido invocada, la coloca en la fila correspondiente y marca la carta como invocada.
  - Llama a `Efecto_Aumento()` para aplicar el efecto de aumento de la carta.
- `public void Invocar_Despeje_Card(Despeje_Card despejeCard)`: Declara un método público llamado `Invocar_Despeje_Card` que toma un parámetro de tipo `Despeje_Card`. Este método invoca una carta de despeje.
  - Si la carta ya ha sido invocada, muestra un mensaje de error en la consola.

- Si la carta no ha sido invocada, la coloca en la fila correspondiente y marca la carta como invocada.
- Llama a `Efecto_Despeje()` para aplicar el efecto de despeje de la carta.

- `public void ActualizarTextoSumaAtaque()`: Declara un método público llamado `ActualizarTextoSumaAtaque`. Este método actualiza el texto que muestra la suma total de puntos de ataque.

10. Game_Manager

El script `Game_Manager` gestiona la mecánica del juego, incluido el control de rondas, turnos de jugadores, puntuación y determinación de ganadores. Aquí está el resumen de lo que hace:

Variables:

- El script declara referencias públicas a varios objetos y componentes del juego, incluyendo los mazos de jugador y rival, las manos de jugador y rival, y las zonas de juego (ataque, distancia y asedio).

Métodos:

- `Start()`: Esta función se llama al inicio del juego y se encarga de barajar los mazos de jugador y rival, además de establecer que el jugador tiene el primer turno.
- `Update()`: Esta función se ejecuta en cada frame y actualiza los puntos de ataque del jugador y del rival. Luego, verifica si ambas manos han pasado la ronda para determinar al ganador.
- `Cambio_Turno_Jugador1()` y `Cambio_Turno_Rival()`: Estos métodos cambian el turno entre el jugador y el rival, verificando si han jugado al menos una carta y si el otro jugador ha pasado la ronda.
- `Pasar_Ronda_Jugador1()` y `Pasar_Ronda_Rival()`: Estos métodos permiten a los jugadores pasar de ronda si cumplen ciertas condiciones, como no haber jugado ninguna carta o no tener cartas en la mano.
- `Ganador_Ronda()`: Este método determina el ganador de la ronda comparando los puntos de ataque del jugador y del rival, y luego incrementa el contador de rondas ganadas del jugador correspondiente.
- `Nueva_Ronda()`: Este método limpia el tablero, actualiza el marcador y reparte nuevas cartas a ambos jugadores para iniciar una nueva ronda.

Este script controla eficientemente la lógica del juego y asegura un flujo suave de juego entre las diferentes rondas y turnos.