using System.Collections.Generic; // Importa la clase para trabajar con listas genéricas.

using UnityEngine; // Importa la clase para interactuar con Unity.

using UnityEngine.UI; // Importa las clases necesarias para trabajar con UI en Unity.

using System.Linq; // Importa la librería LINQ para operaciones de consultas sobre colecciones.

public class paneles : MonoBehaviour

{

    // Definición de una clase Grafica que contiene información sobre los gráficos

    [System.Serializable]

    public class Grafica

    {

        public RectTransform panel; // Panel donde se dibujará la gráfica.

        public List<float> tiempo = new List<float>(); // Lista que almacena los valores del tiempo.

        public List<float> transversalAnteb = new List<float>(), frontalAnteb = new List<float>(), sagitalAnteb = new List<float>(); // Listas para los diferentes ejes de datos de la primera parte del cuerpo.

        public List<float> transversalEscap = new List<float>(), frontalEscap = new List<float>(), sagitalEscap = new List<float>(); // Listas para los diferentes ejes de datos de la segunda parte del cuerpo.

```
    public List<float> emg = new List<float>(); // Lista para los datos de EMG (electromiografía).


    // Constructor de la clase que inicializa las listas.
    public Grafica()
    {
        tiempo = new List<float>();
        transversalAnteb = new List<float>();
        frontalAnteb = new List<float>();
        sagitalAnteb = new List<float>();
        transversalEscap = new List<float>();
        frontalEscap = new List<float>();
        sagitalEscap = new List<float>();
        emg = new List<float>();
    }
}

public List<Grafica> graficas; // Lista que contiene todos los objetos Grafica.
public GameObject puntonegro; // Prefab para el punto negro.
```

```
    public GameObject puntorojo; // Prefab para el punto rojo.

    public GameObject puntoazul; // Prefab para el punto azul.

    public GameObject etiquetaPrefab; // Prefab para las etiquetas.

    public GameObject lineaPrefab; // Prefab para las líneas (utiliza LineRenderer).



    private bool tomandoDatos = false; // Bandera para saber si se están capturando datos.

    private List<List<float>> datosAlmacenados = new List<List<float>>(); // Lista que almacena los datos capturados.



    // Método para iniciar la captura de datos.

    public void IniciarCaptura()

    {

        tomandoDatos = true; // Activa la captura de datos.

        datosAlmacenados.Clear(); // Limpia los datos almacenados.

        for (int i = 0; i < 8; i++)

            datosAlmacenados.Add(new List<float>()); // Crea 8 listas vacías para almacenar los datos.

    }



    // Método para detener la captura de datos.
```

```
public void DetenerCaptura()

{

    tomandoDatos = false; // Desactiva la captura de datos.

    ActualizarDatos(datosAlmacenados); // Actualiza los gráficos con los datos almacenados.

}



// Método que recibe los datos y los almacena si está capturando datos.

public void RecibirDatos(List<float> datos)

{

    if (tomandoDatos && datos.Count == 8) // Verifica si se están capturando datos y si la lista tiene 8 elementos.

    {

        for (int i = 0; i < 8; i++) // Itera a través de las 8 listas.

            datosAlmacenados[i].Add(datos[i]); // Agrega los datos a la lista correspondiente.

    }

}



// Método para actualizar las gráficas con los nuevos datos.
```

```
public void ActualizarDatos(List<List<float>> nuevosDatos)
{
    Debug.Log("Actualizando la gráfica con nuevos datos.");
    if (nuevosDatos.Count != 8) // Verifica que los nuevos datos tengan 8 listas.
    {
        Debug.LogError("Cantidad incorrecta de listas en nuevosDatos.");
        return; // Sale si el número de listas no es el correcto.
    }

    // Se asignan los datos a las gráficas correspondientes.
    graficas[0].tiempo = new List<float>(nuevosDatos[7]);
    graficas[0].transversalAnteb = new List<float>(nuevosDatos[0]);
    graficas[0].frontalAnteb = new List<float>(nuevosDatos[1]);
    graficas[0].sagitalAnteb = new List<float>(nuevosDatos[2]);

    graficas[1].tiempo = new List<float>(nuevosDatos[7]);
    graficas[1].transversalEscap = new List<float>(nuevosDatos[3]);
    graficas[1].frontalEscap = new List<float>(nuevosDatos[4]);
    graficas[1].sagitalEscap = new List<float>(nuevosDatos[5]);
```

```
    graficas[2].transversalAnteb = new List<float>(nuevosDatos[0]);
    graficas[2].frontalAnteb = new List<float>(nuevosDatos[1]);
    graficas[2].sagitalAnteb = new List<float>(nuevosDatos[2]);

    graficas[3].transversalEscap = new List<float>(nuevosDatos[3]);
    graficas[3].frontalEscap = new List<float>(nuevosDatos[4]);
    graficas[3].sagitalEscap = new List<float>(nuevosDatos[5]);

    graficas[4].tiempo = new List<float>(nuevosDatos[7]);
    graficas[4].emg = new List<float>(nuevosDatos[6]);

    DibujarTodasLasGraficas(); // Llama al método para dibujar todas las gráficas.
}

// Método para dibujar todas las gráficas.
private void DibujarTodasLasGraficas()
{
    if (graficas.Count < 5) // Verifica que haya al menos 5 gráficos en la lista.
    {
```

```
        Debug.LogError("No hay suficientes paneles asignados en la lista de graficas.");

        return; // Sale si no hay suficientes paneles.

    }


    // Dibuja las diferentes gráficas.

    DibujarGrafica(graficas[0].panel, graficas[0].tiempo, new List<(List<float>, GameObject)>

    {

        (graficas[0].transversalAnteb, puntonegro),

        (graficas[0].frontalAnteb, puntorojo),

        (graficas[0].sagitalAnteb, puntoazul)

    });


    DibujarGrafica(graficas[1].panel, graficas[1].tiempo, new List<(List<float>, GameObject)>

    {

        (graficas[1].transversalEscap, puntonegro),

        (graficas[1].frontalEscap, puntorojo),

        (graficas[1].sagitalEscap, puntoazul)
```

```
});

DibujarGrafica(graficas[2].panel, graficas[2].transversalAnteb, new List<(List<float>, GameObject)>
{
    (graficas[2].frontalAnteb, puntorojo),
    (graficas[2].sagitalAnteb, puntoazul)
});

DibujarGrafica(graficas[3].panel, graficas[3].transversalEscap, new List<(List<float>, GameObject)>
{
    (graficas[3].frontalEscap, puntorojo),
    (graficas[3].sagitalEscap, puntoazul)
});

DibujarGrafica(graficas[4].panel, graficas[4].tiempo, new List<(List<float>, GameObject)>
{
    (graficas[4].emg, puntonegro)
});
```

```
    }

    // Método para dibujar un gráfico en un panel específico.
    private void DibujarGrafica(RectTransform panel, List<float> datosX, List<(List<float>, GameObject)> datosYPrefabs)
    {
        if (panel == null) // Verifica que el panel no sea null.
        {
            Debug.LogError("El panel es null.");
            return; // Sale si el panel no está asignado.
        }

        // Elimina los hijos actuales del panel antes de dibujar la nueva gráfica.
        foreach (Transform child in panel)
            Destroy(child.gameObject);

        float width = panel.rect.width; // Obtiene el ancho del panel.
        float height = panel.rect.height; // Obtiene la altura del panel.
        if (datosX.Count == 0) return; // Si no hay datos, no hace nada.
```

```
float minX = Mathf.Min(datosX.ToArray()); // Encuentra el valor mínimo de X.

float maxX = Mathf.Max(datosX.ToArray()); // Encuentra el valor máximo de X.

float minY = float.MaxValue, maxY = float.MinValue; // Inicializa los valores mínimos y máximos de Y.

foreach (var (lista, _) in datosYPrefabs)

{

    if (lista.Count > 0)

    {

        minY = Mathf.Min(minY, Mathf.Min(lista.ToArray())); // Encuentra el valor mínimo de Y.

        maxY = Mathf.Max(maxY, Mathf.Max(lista.ToArray())); // Encuentra el valor máximo de Y.

    }

}

if (minY == maxY) maxY += 0.1f; // Si no hay variación en Y, ajusta un poco el valor máximo.



float margenX = 0.1f * (maxX - minX); // Calcula el margen en el eje X.

float margenY = 0.1f * (maxY - minY); // Calcula el margen en el eje Y.

minX -= margenX; // Ajusta el valor mínimo de X.
```

```
        maxX += margenX; // Ajusta el valor máximo de X.

        minY -= margenY; // Ajusta el valor mínimo de Y.

        maxY += margenY; // Ajusta el valor máximo de Y.



        DibujarCuadrilla(panel, minX, maxX, minY, maxY, width, height, etiquetaPrefab);
// Dibuja la cuadrícula en el panel.



        // Dibuja los puntos en la gráfica.

        foreach (var (lista, prefab) in datosYPrefabs)

        {

            for (int i = 0; i < datosX.Count; i++)

            {

                // Calcula la posición X e Y en el panel según los datos.

                float posX = ((datosX[i] - minX) / (maxX - minX)) * width - (width / 2);

                float posY = ((lista[i] - minY) / (maxY - minY)) * height - (height / 2);

                GameObject punto = Instantiate(prefab, panel); // Instancia el prefab para
el punto.

                punto.GetComponent<RectTransform>().anchoredPosition = new
Vector2(posX, posY); // Asigna la posición al punto.

            }
```

```
    }

}


// Método para dibujar la cuadrícula en el panel.

private void DibujarCuadrilla(RectTransform panel, float minX, float maxX, float minY, float maxY, float width, float height, GameObject prefabEtiqueta)

{

    int divisiones = 5; // Número de divisiones en la cuadrícula.

    for (int i = 0; i <= divisiones; i++)

    {

        float x = i * (width / divisiones) - (width / 2); // Calcula la posición en X.

        CrearLinea(new Vector2(x, -height / 2), new Vector2(x, height / 2), panel, Color.gray, 0.25f); // Dibuja la línea vertical.

        float valorX = minX + i * ((maxX - minX) / divisiones); // Calcula el valor de X.

        CrearEtiqueta(new Vector2(x + 62, -height / 2 - 20), valorX.ToString("F2"), panel, prefabEtiqueta); // Dibuja la etiqueta para X.

        float y = i * (height / divisiones) - (height / 2); // Calcula la posición en Y.

        CrearLinea(new Vector2(-width / 2, y), new Vector2(width / 2, y), panel, Color.gray, 0.25f); // Dibuja la línea horizontal.

        float valorY = minY + i * ((maxY - minY) / divisiones); // Calcula el valor de Y.
```

```
            CrearEtiqueta(new Vector2(-width + 530, y - 5), valorY.ToString("F2"), panel, prefabEtiqueta); // Dibuja la etiqueta para Y.

        }

    }



    // Método para crear una línea entre dos puntos.

    private void CrearLinea(Vector2 start, Vector2 end, RectTransform parent, Color color, float width)

    {

        GameObject linea = Instantiate(lineaPrefab, parent); // Crea una nueva línea.

        LineRenderer lr = linea.GetComponent<LineRenderer>(); // Obtiene el componente LineRenderer.

        lr.startColor = color; // Asigna el color de inicio.

        lr.endColor = color; // Asigna el color de fin.

        lr.startWidth = width; // Asigna el grosor de la línea.

        lr.endWidth = width; // Asigna el grosor de la línea.

        lr.SetPosition(0, start); // Establece la posición inicial de la línea.

        lr.SetPosition(1, end); // Establece la posición final de la línea.

    }
```

```
    // Método para crear etiquetas de texto.

    private void CrearEtiqueta(Vector2 position, string text, RectTransform parent, GameObject prefab)

    {

        GameObject etiqueta = Instantiate(prefab, parent); // Instancia la etiqueta.

        etiqueta.GetComponent<RectTransform>().anchoredPosition = position; // Asigna la posición.

        etiqueta.GetComponent<Text>().text = text; // Asigna el texto de la etiqueta.

    }

}
```