using System.Collections.Generic; // Importa las clases necesarias para manejar listas genéricas.

using UnityEngine; // Importa UnityEngine para el manejo de objetos y gráficos en Unity.

using UnityEngine.UI; // Importa UnityEngine.UI para el manejo de UI como botones y texto en Unity.

using System.Linq; // Importa LINQ para trabajar con consultas sobre colecciones.

public class panelesbasededatos : MonoBehaviour // Define la clase principal que hereda de MonoBehaviour, permitiendo usarla en Unity.

{

[System.Serializable] // Hace que esta clase sea serializable para poder ser vista y editada en el editor de Unity.

public class Grafica // Define una clase llamada "Grafica" que almacenará los datos gráficos.

{

public RectTransform panel; // El panel que contiene la gráfica.

public List<float> tiempo = new List<float>(); // Lista para almacenar los tiempos de la gráfica.

public List<float> transversalAnteb = new List<float>(), frontalAnteb = new List<float>(), sagitalAnteb = new List<float>(); // Listas para almacenar los datos de cada dimensión del antebrazo.

public List<float> transversalEscap = new List<float>(), frontalEscap = new List<float>(), sagitalEscap = new List<float>(); // Listas para los datos de cada dimensión de la escápula.

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


    public Grafica()  // Constructor que inicializa las listas.
    {
        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 todas las gráficas.
public GameObject puntonegro;  // Prefab de punto negro usado en las gráficas.
public GameObject puntorojo;  // Prefab de punto rojo usado en las gráficas.
```

```
    public GameObject puntoazul;  // Prefab de punto azul usado en las gráficas.

    public GameObject etiquetaPrefab;  // Prefab de las etiquetas en las gráficas.

    public GameObject lineaPrefab;  // Prefab para crear líneas usando LineRenderer.



    private bool tomandoDatos = false;  // Bandera que indica si se están capturando datos.

    private List<List<float>> datosAlmacenados = new List<List<float>>();  // Lista para almacenar los datos durante la captura.



    public void IniciarCaptura()  // Método que inicia la captura de datos.

    {

        tomandoDatos = true;  // Establece la bandera como true.

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

        for (int i = 0; i < 8; i++)  // Inicializa listas vacías para almacenar datos.

            datosAlmacenados.Add(new List<float>());

    }



    public void DetenerCaptura()  // Método que detiene la captura de datos.

    {

        tomandoDatos = false;  // Establece la bandera como false.
```

```
        ActualizarDatos(datosAlmacenados);  // Actualiza los datos en las gráficas.

    }


    public void RecibirDatos(List<float> datos)  // Método que recibe datos y los almacena si está capturando datos.

    {

        if (tomandoDatos && datos.Count == 8)  // Verifica si se están recibiendo 8 datos.

        {

            for (int i = 0; i < 8; i++)  // Almacena los datos en las listas correspondientes.

                datosAlmacenados[i].Add(datos[i]);

        }

    }


    public void ActualizarDatos(List<List<float>> nuevosDatos)  // Actualiza los datos de las gráficas.

    {

        Debug.Log("Actualizando la gráfica con nuevos datos.");

        if (nuevosDatos.Count != 8)  // Verifica que los datos recibidos sean de la cantidad esperada.

        {
```

```
    Debug.LogError("Cantidad incorrecta de listas en nuevosDatos.");
    return;
}

// Asigna 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();  // Dibuja todas las gráficas con los nuevos datos.

    }


    private void DibujarTodasLasGraficas()  // Dibuja todas las gráficas asignadas.

    {

        if (graficas.Count < 5)  // Verifica que haya suficientes gráficas.

        {

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

            return;

        }


        // Dibuja cada una de las gráficas con sus respectivos datos.
```

```
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[3].frontalEscap, puntorojo),
```

```
            (graficas[3].sagitalEscap, 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)

        });

    }


    private void DibujarGrafica(RectTransform panel, List<float> datosX,
List<(List<float>, GameObject)> datosYPrefabs)  // Dibuja una gráfica en un panel
determinado.

    {
```

```
        if (panel == null)  // Verifica que el panel no sea nulo.

        {

            Debug.LogError("El panel es null.");

            return;

        }



        foreach (Transform child in panel)  // Elimina todos los elementos hijos del panel (limpia la gráfica).

            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 dibuja nada.



        // Calcula los valores mínimo y máximo de X e Y para la escala de la gráfica.

        float minX = Mathf.Min(datosX.ToArray());

        float maxX = Mathf.Max(datosX.ToArray());

        float minY = float.MaxValue, maxY = float.MinValue;

        foreach (var (lista, _) in datosYPrefabs)

        {
```

```
    if (lista.Count > 0)
    {
        minY = Mathf.Min(minY, Mathf.Min(lista.ToArray()));
        maxY = Mathf.Max(maxY, Mathf.Max(lista.ToArray()));
    }
}
if (minY == maxY) maxY += 0.1f;  // Asegura que haya un rango en Y.

// Agrega un margen a los valores de X e Y.
float margenX = 0.1f * (maxX - minX);
float margenY = 0.1f * (maxY - minY);
minX -= margenX;
maxX += margenX;
minY -= margenY;
maxY += margenY;

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

```
        // Dibuja los puntos en la gráfica.
        foreach (var (lista, prefab) in datosYPrefabs)
        {
            for (int i = 0; i < datosX.Count; i++)
            {
                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.
                punto.GetComponent<RectTransform>().anchoredPosition = new Vector2(posX, posY);  // Asigna la posición del punto.
            }
        }
    }

    private void DibujarCuadrilla(RectTransform panel, float minX, float maxX, float minY, float maxY, float width, float height, GameObject prefabEtiqueta)  // Dibuja la cuadrícula de la gráfica.
    {
        int divisiones = 5;  // Número de divisiones en la cuadrícula.
```

```
        for (int i = 0; i <= divisiones; i++)  // Dibuja las líneas verticales de la cuadrícula.

        {

            float x = i * (width / divisiones) - (width / 2);

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

            float valorX = minX + i * ((maxX - minX) / divisiones);

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



            float y = i * (height / divisiones) - (height / 2);  // Dibuja las líneas horizontales de la cuadrícula.

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

            float valorY = minY + i * ((maxY - minY) / divisiones);

            CrearEtiqueta(new Vector2(-width + 530, y - 5), valorY.ToString("F2"), panel, prefabEtiqueta);

        }

    }



    private void CrearLinea(Vector2 start, Vector2 end, RectTransform panel, Color color, float width)  // Crea una línea entre dos puntos.

    {
```

```
        GameObject linea = Instantiate(lineaPrefab, panel);  // Instancia el prefab de la línea.

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

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

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

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

        lr.endColor = color;  // Establece el color final.

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

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

    }


    private void CrearEtiqueta(Vector2 pos, string texto, RectTransform panel, GameObject prefabEtiqueta)  // Crea una etiqueta con texto.

    {

        GameObject etiqueta = Instantiate(prefabEtiqueta, panel);  // Instancia el prefab de la etiqueta.

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

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

    public void LimpiarGraficas()
```

```
{
    // Destruye todos los elementos instanciados en cada panel de gráfica
    foreach (Grafica g in graficas)
    {
        foreach (Transform child in g.panel)
        {
            Destroy(child.gameObject);  // Elimina los elementos hijo (líneas, puntos, etiquetas, etc.)
        }
        // Opcional: reinicia las listas de datos
        g.tiempo.Clear();  // Limpia la lista de tiempo
        g.transversalAnteb.Clear();  // Limpia la lista de datos del ángulo transversal del antebrazo
        g.frontalAnteb.Clear();  // Limpia la lista de datos del ángulo frontal del antebrazo
        g.sagitalAnteb.Clear();  // Limpia la lista de datos del ángulo sagital del antebrazo
        g.transversalEscap.Clear();  // Limpia la lista de datos del ángulo transversal de la escápula
        g.frontalEscap.Clear();  // Limpia la lista de datos del ángulo frontal de la escápula
        g.sagitalEscap.Clear();  // Limpia la lista de datos del ángulo sagital de la escápula
```

```
        g.emg.Clear();  // Limpia la lista de datos de EMG

    }

}
```