using System.Collections.Generic; // Importa la librería System.Collections.Generic, que permite el uso de colecciones genéricas, como listas y diccionarios.

using UnityEngine; // Importa el espacio de nombres UnityEngine, necesario para utilizar funcionalidades básicas de Unity, como el sistema de logueo (Debug).

using TMPro; // Importa el espacio de nombres TextMeshPro, necesario para utilizar componentes de texto en Unity.

public class DropdownManager : MonoBehaviour // Declara la clase DropdownManager que hereda de MonoBehaviour, lo que le permite ser un componente de Unity.

{

    public TMP_Dropdown dropdownEjercicios; // Referencia pública a un componente TMP_Dropdown (dropdown de ejercicios) en la interfaz de usuario.

    public TMP_Dropdown dropdownSeries; // Referencia pública a un componente TMP_Dropdown (dropdown de series) en la interfaz de usuario.

    public secuenciaManager secuenciaManager; // Referencia pública a un script 'secuenciaManager' que contiene la lógica para manejar los datos de los ejercicios.

    public paneles panelesScript; // Referencia pública a un script 'paneles' que se encarga de actualizar los paneles de la interfaz.

    public TextMeshProUGUI textoResumenRepeticiones; // Referencia pública a un componente de texto en la interfaz para mostrar las repeticiones realizadas.

    private Dictionary<string, List<int>> seriesPorEjercicio = new Dictionary<string, List<int>>(); // Diccionario que mapea cada ejercicio a una lista de series asociadas a ese ejercicio.

private List<string> listaEjercicios = new List<string>(); // Lista que contiene los nombres de los ejercicios disponibles.

void Start() // Método que se ejecuta al inicio del ciclo de vida del objeto (cuando se inicializa).

{

dropdownEjercicios.onValueChanged.AddListener(delegate // Agrega un listener para detectar cuando el valor del dropdown de ejercicios cambie.

{

ActualizarDropdownSeries(); // Llama al método para actualizar las opciones de series cuando se seleccione un ejercicio.

ActualizarTextoResumenRepeticiones(); // Actualiza el texto que muestra las repeticiones realizadas.

});

dropdownSeries.onValueChanged.AddListener(delegate // Agrega un listener para detectar cuando el valor del dropdown de series cambie.

{

GraficarDatos(); // Llama al método para graficar los datos correspondientes a la serie seleccionada.

ActualizarTextoResumenRepeticiones(); // Actualiza el texto que muestra las repeticiones realizadas.

});

ActualizarDropdownEjercicios(); // Llama a un método para actualizar la lista de ejercicios disponibles al inicio.

}

public void ActualizarDropdownEjercicios() // Método que actualiza las opciones disponibles en el dropdown de ejercicios.

{

dropdownEjercicios.ClearOptions(); // Limpia las opciones actuales en el dropdown de ejercicios.

dropdownSeries.ClearOptions(); // Limpia las opciones actuales en el dropdown de series.

listaEjercicios.Clear(); // Limpia la lista de ejercicios.

seriesPorEjercicio.Clear(); // Limpia el diccionario de series por ejercicio.

var datos = secuenciaManager.ObtenerDatosPorEjercicio(); // Obtiene los datos de los ejercicios del 'secuenciaManager'.

if (datos.Count == 0) // Si no hay datos, muestra un mensaje de error y termina el método.

{

Debug.Log("No hay datos en ObtenerDatosPorEjercicio(). Dropdown quedará vacío.");

return;

}

foreach (var kvp in datos) // Itera sobre cada par clave-valor (ejercicio y datos asociados) en el diccionario de datos.

{

string ejercicio = kvp.Key; // La clave es el nombre del ejercicio.

Dictionary<int, List<List<float>>> infoEjercicio = kvp.Value; // Los datos asociados al ejercicio.

listaEjercicios.Add(ejercicio); // Añade el nombre del ejercicio a la lista de ejercicios.

if (!seriesPorEjercicio.ContainsKey(ejercicio)) // Si el ejercicio no tiene series asignadas aún, lo añade al diccionario.

seriesPorEjercicio[ejercicio] = new List<int>();

foreach (var serie in infoEjercicio.Keys) // Itera sobre las series asociadas a ese ejercicio.

{

if (!seriesPorEjercicio[ejercicio].Contains(serie)) // Si la serie aún no está en el diccionario, la añade.

seriesPorEjercicio[ejercicio].Add(serie);

}

}

```
        dropdownEjercicios.AddOptions(listaEjercicios);    // Añade las opciones de ejercicios al dropdown de ejercicios.

        ActualizarDropdownSeries();    // Actualiza las opciones de series según el ejercicio seleccionado.

    }


    void ActualizarDropdownSeries()  // Método que actualiza las opciones de series en el dropdown de series.

    {

        dropdownSeries.ClearOptions();  // Limpia las opciones actuales en el dropdown de series.

        if (listaEjercicios.Count == 0)  // Si no hay ejercicios en la lista, termina el método.

            return;


        int indice = dropdownEjercicios.value;    // Obtiene el índice del ejercicio seleccionado.

        string ejercicioSeleccionado = listaEjercicios[indice];  // Obtiene el nombre del ejercicio seleccionado.


        if (!seriesPorEjercicio.ContainsKey(ejercicioSeleccionado))  // Si no existen series para el ejercicio seleccionado, muestra una advertencia.

        {

            Debug.LogWarning("No hay series para el ejercicio: " + ejercicioSeleccionado);
```

```
            return;

        }

        List<int> series = seriesPorEjercicio[ejercicioSeleccionado];  // Obtiene la lista de series asociadas al ejercicio seleccionado.

        List<string> opcionesSeries = new List<string>();  // Crea una lista de opciones de series.

        opcionesSeries.Add("Selecciona Serie");  // Agrega una opción por defecto.

        opcionesSeries.AddRange(series.ConvertAll(s => "Serie " + s));  // Añade cada serie a la lista de opciones de series.

        dropdownSeries.AddOptions(opcionesSeries);  // Añade las opciones de series al dropdown de series.

        dropdownSeries.value = 0;  // Establece el valor por defecto de la serie como 0.

    }

    void GraficarDatos()  // Método que grafica los datos correspondientes al ejercicio y serie seleccionados.

    {

        if (listaEjercicios.Count == 0 || dropdownSeries.options.Count <= 1)  // Si no hay ejercicios o no se ha seleccionado una serie válida, termina el método.

            return;
```

int indiceEjercicio = dropdownEjercicios.value; // Obtiene el índice del ejercicio seleccionado.

string ejercicioSeleccionado = listaEjercicios[indiceEjercicio]; // Obtiene el nombre del ejercicio seleccionado.

if (dropdownSeries.value == 0) // Si no se ha seleccionado una serie válida, muestra un mensaje de advertencia.

{

Debug.Log("Por favor, selecciona una serie válida.");

return;

}

int serieSeleccionada; // Variable para almacenar el número de la serie seleccionada.

if (!int.TryParse(dropdownSeries.options[dropdownSeries.value].text.Replace("Serie ", ""), out serieSeleccionada)) // Intenta convertir el texto de la opción seleccionada en un número.

{

Debug.LogError("Error al convertir la serie a número."); // Si la conversión falla, muestra un mensaje de error.

return;

}

```
        var datos = secuenciaManager.ObtenerDatosPorEjercicio();  // Obtiene los datos del 'secuenciaManager'.

        if (!datos.ContainsKey(ejercicioSeleccionado) || !datos[ejercicioSeleccionado].ContainsKey(serieSeleccionada))  // Si no se encuentran datos para el ejercicio o la serie seleccionada, muestra un mensaje de error.

        {

            Debug.LogError("No se encontraron datos para ese ejercicio/serie.");

            return;

        }



        List<List<float>> rawData = datos[ejercicioSeleccionado][serieSeleccionada];  // Obtiene los datos sin procesar para el ejercicio y serie seleccionados.

        List<List<float>> translatedData = DataTranslator.TranslateData(rawData, ejercicioSeleccionado);  // Traduce los datos crudos a un formato más adecuado para la visualización.

        if (panelesScript != null)  // Si el script de paneles no es nulo, actualiza los paneles con los datos traducidos.

            panelesScript.ActualizarDatos(translatedData);

    }



    public void ActualizarTextoResumenRepeticiones()  // Método que actualiza el texto de resumen de repeticiones en la interfaz.
```

```
    {

        if (secuenciaManager == null || textoResumenRepeticiones == null)  // Si no hay un 'secuenciaManager' o un texto para las repeticiones, termina el método.

            return;



        int indiceEjercicio = dropdownEjercicios.value;  // Obtiene el índice del ejercicio seleccionado.

        if (indiceEjercicio < 0 || indiceEjercicio >= listaEjercicios.Count)  // Si el índice es inválido, muestra "0" como repeticiones.

        {

            textoResumenRepeticiones.text = "Repeticiones: 0";

            return;

        }



        string ejercicioSeleccionado = listaEjercicios[indiceEjercicio];   // Obtiene el nombre del ejercicio seleccionado.

        int rep = secuenciaManager.GetRepeticionesPorEjercicio(ejercicioSeleccionado);   // Obtiene el número de repeticiones realizadas para ese ejercicio.

        textoResumenRepeticiones.text = "Repeticiones: " + rep;  // Actualiza el texto de resumen con el número de repeticiones.

    }

}
```