```
using System.Collections;              // Proporciona interfaces y clases para colecciones

using System.Collections.Generic;      // Proporciona clases de colecciones genéricas

using TMPro;                           // Biblioteca TextMeshPro para manejo de texto UI

using UnityEngine;                     // Framework principal de Unity

using UnityEngine.UI;                  // Componentes del sistema UI de Unity

using Newtonsoft.Json;                 // Biblioteca JSON para serialización/deserialización

using System;                          // Funcionalidad principal de C#

using System.Linq;                     // LINQ para manipulación de colecciones


public class secuenciaManager : MonoBehaviour // Clase principal que hereda de MonoBehaviour

{

    public string[] ejercicios;        // Array para almacenar los nombres de ejercicios


    // Paneles de animaciones tutoriales

    public GameObject PanelAnimacionesBolos; // Panel donde van las animaciones de bolos

    public GameObject PanelAnimacionesArco;  // Panel donde van las animaciones de arco

    public GameObject AnimacionPendulo;      // Objeto de animación péndulo
```

```
    public GameObject AnimacionRotacionE;    // Animación de rotación escapular

    public GameObject AnimacionRotacionEFI;  // Animación de rotación externa en abducción

    public GameObject AnimacionRotacionEA;   // Animación de rotación externa de arco

    public GameObject AnimacionRotacionIA;   // Animación de rotación interna de arco



    // Objetos de calibración

    public GameObject CalibracionPendulo;    // Objeto de calibración del péndulo

    public GameObject CalibracionFortaEscap; // Calibración de fortalecimiento escapular

    public GameObject CalibracionRotaAbdExt; // Calibración de rotación externa en abducción

    public GameObject CalibracionRotaExtArco;// Calibración de rotación externa de arco

    public GameObject CalibracionRotaIntArco;// Calibración de rotación interna de arco



    // VARIABLES BOLOS

    public TMP_InputField sesPendulo;        // Campo de entrada para sesiones de péndulo

    public TMP_InputField repPendulo;        // Campo de entrada para repeticiones de péndulo
```

```
    public TMP_InputField sesRotacionE;       // Campo de entrada para sesiones de rotación escapular

    public TMP_InputField repRotacionE;      // Campo de entrada para repeticiones de rotación escapular

    public TMP_InputField sesRotacionEFI;     // Campo de entrada para sesiones de rotación externa en abducción

    public TMP_InputField repRotacionEFI;     // Campo de entrada para repeticiones de rotación externa en abducción



    // Información de sesiones y valores de bolos

    private int sesionesPendulo;             // Número de sesiones de péndulo

    private int repeticionesPendulo;         // Número de repeticiones de péndulo

    private int sesionesRotacionE;           // Número de sesiones de rotación escapular

    private int repeticionesRotacionE;           // Número de repeticiones de rotación escapular

    private int sesionesRotacionEFI;          // Número de sesiones de rotación externa en abducción

    private int repeticionesRotacionEFI;         // Número de repeticiones de rotación externa en abducción



    // Videos de Bolos

    public GameObject video0;                // Objeto para el primer video
```

```
    public GameObject video1;              // Objeto para el segundo video

    public GameObject video2;              // Objeto para el tercer video


    // VARIABLES ARCO

    public TMP_InputField sesRotacionEA;      // Campo de entrada para sesiones de rotación externa de arco

    public TMP_InputField repRotacionEA;      // Campo de entrada para repeticiones de rotación externa de arco

    public TMP_InputField sesRotacionIA;      // Campo de entrada para sesiones de rotación interna de arco

    public TMP_InputField repRotacionIA;     // Campo de entrada para repeticiones de rotación interna de arco



    // Información de sesiones y valores de arco

    private int sesionesRotacionEA;          // Número de sesiones de rotación externa de arco

    private int repeticionesRotacionEA;          // Número de repeticiones de rotación externa de arco

    private int sesionesRotacionIA;          // Número de sesiones de rotación interna de arco

    private int repeticionesRotacionIA;      // Número de repeticiones de rotación interna de arco
```

```
    // Videos de Arco
    public GameObject video3;              // Objeto para el cuarto video
    public GameObject video4;              // Objeto para el quinto video

    // UI de Información
    public TextMeshProUGUI infoEjercicioText;  // Texto que muestra el ejercicio actual
    public TextMeshProUGUI infoSesionText;     // Texto que muestra la sesión actual
    public TextMeshProUGUI infoRepeticionText;     // Texto que muestra las repeticiones actuales

    // Canvas de Retroalimentación
    public TextMeshProUGUI infoCorrectoText;     // Texto para feedback correcto
    public TextMeshProUGUI infoIncorrectoText;    // Texto para feedback incorrecto
    public TextMeshProUGUI infoEjercicioPanelText;// Texto para panel de ejercicio
    public GameObject infoCorrectoPanel;         // Panel de ejercicio correcto
    public GameObject infoFinalPanel;            // Panel de categoría finalizada
    public GameObject EjercicioFinalizadoPanel;   // Panel de ejercicio finalizado
    public GameObject seriesPrefab;              // Prefab del panel con texto e imagen
    public Transform panelContainer;             // Contenedor para instanciar prefabs
```

```
    private List<GameObject> instantiatedPanels = new List<GameObject>(); // Lista de paneles creados

    private Vector3 posicionInicial = new Vector3(-133.5f, 50.2f, 0);     // Posición inicial para paneles

    private float offsetY = -20f;                // Desplazamiento vertical entre paneles



    // Variables Controladoras

    public string tipoEjercicio;                // Categoría (Bolos o Arco)

    public int sesionActual;                    // Sesión actual del ejercicio

    public int ejercicioActualIndex;            // Índice del ejercicio actual

    private bool ejercicioEnProgreso;           // Indica si hay un ejercicio en progreso

    // ---------- NUEVAS VARIABLES PARA SINCRONIZAR CON LOS DATOS ----------



    // Referencia al script de envío de datos (versión nueva)

    public enviodedatos envioDatosScript;     // Referencia al script que maneja el envío de datos



    // Estructura para almacenar los datos de cada serie para cada ejercicio.

    // La clave es el nombre del ejercicio y el valor es otro diccionario:

    //    llave: número de serie (int) – valor: datos procesados (List<List<float>> de 8 listas: pitch1, roll1, yaw1, pitch2, roll2, yaw2, emg, tiempo)
```

```
    private Dictionary<string, Dictionary<int, Dictionary<int, SerieData>>> datosPorEjercicio = new Dictionary<string, Dictionary<int, Dictionary<int, SerieData>>>();



    public InputField inputNota;              // Campo de entrada para la nota de firebase



    [System.Serializable]

    public class SerieData {                  // Clase para almacenar datos de una serie

        public List<List<float>> datos;                  // Estructura original usada en DropdownManager

        public string datosJson;              // Cadena JSON para enviar a Firebase

        public int repeticiones;              // Número de repeticiones de la serie



        public SerieData(List<List<float>> datos, int repeticiones) {  // Constructor de la clase

            this.datos = datos;

            // Serializamos exactamente la estructura recibida

            this.datosJson = Newtonsoft.Json.JsonConvert.SerializeObject(datos);

            this.repeticiones = repeticiones;

        }

    }
```

```
    void Start()  // Método ejecutado al iniciar el script

    {

        ejercicios = new string[] { "Péndulo", "Fortalecimiento escapular", "Rotación externa en aducción", "Rotación externa", "Rotación interna" };  // Inicializa el array de ejercicios

        ejercicioEnProgreso = false;          // Inicialmente no hay ejercicio en progreso

        ejercicioActualIndex = 0;             // Comienza con el primer ejercicio

        sesionActual = 1;                     // Comienza con la primera sesión



        // Suscripción al evento que dispara el script de envío de datos al finalizar la recopilación

        if (envioDatosScript != null)

        {

            // Se asume que enviodedatos (versión nueva) tiene un evento OnDatosProcesados de tipo Action<List<List<float>>>

            envioDatosScript.OnDatosProcesados += RecibirDatosSerie;  // Suscribe al método RecibirDatosSerie al evento

        }

    }
```

```
    // Guarda los valores ingresados para los ejercicios de Bolos

    public void GuardarValoresBolos()

    {

        int.TryParse(sesPendulo.text, out sesionesPendulo);            // Convierte el texto a entero para sesiones de péndulo

        int.TryParse(repPendulo.text, out repeticionesPendulo);        // Convierte el texto a entero para repeticiones de péndulo

        int.TryParse(sesRotacionE.text, out sesionesRotacionE);        // Convierte el texto a entero para sesiones de rotación escapular

        int.TryParse(repRotacionE.text, out repeticionesRotacionE);    // Convierte el texto a entero para repeticiones de rotación escapular

        int.TryParse(sesRotacionEFI.text, out sesionesRotacionEFI);    // Convierte el texto a entero para sesiones de rotación externa en abducción

        int.TryParse(repRotacionEFI.text, out repeticionesRotacionEFI);// Convierte el texto a entero para repeticiones de rotación externa en abducción

        Debug.Log($"Valores guardados: {sesionesPendulo} {repeticionesPendulo} {sesionesRotacionE} {repeticionesRotacionE} {sesionesRotacionEFI} {repeticionesRotacionEFI}");  // Registra los valores

    }



    // Guarda los valores ingresados para los ejercicios de Arco

    public void GuardarValoresArco()

    {
```

```
        int.TryParse(sesRotacionEA.text, out sesionesRotacionEA);       // Convierte el texto a entero para sesiones de rotación externa de arco

        int.TryParse(repRotacionEA.text, out repeticionesRotacionEA); // Convierte el texto a entero para repeticiones de rotación externa de arco

        int.TryParse(sesRotacionIA.text, out sesionesRotacionIA);     // Convierte el texto a entero para sesiones de rotación interna de arco

        int.TryParse(repRotacionIA.text, out repeticionesRotacionIA); // Convierte el texto a entero para repeticiones de rotación interna de arco

        Debug.Log($"Valores guardados: {sesionesRotacionEA} {repeticionesRotacionEA} {sesionesRotacionIA} {repeticionesRotacionIA}"); // Registra los valores

    }


    // Establece el tipo de ejercicio (Bolos o Arco)

    public void InformacionTipoEjercicio(string ejercicioInput)

    {

        tipoEjercicio = ejercicioInput;  // Guarda el tipo de ejercicio

    }


    // Inicia la secuencia de ejercicios según el tipo seleccionado

    public void IniciarEjercicios()

    {
```

```
        LimpiarPaneles02();  // Limpia los paneles existentes
        if (!ejercicioEnProgreso)  // Verifica que no haya un ejercicio en progreso
        {
            ejercicioEnProgreso = true;  // Marca el inicio de ejercicios
            if (tipoEjercicio == "Bolos")  // Si el tipo es Bolos
            {
                GuardarValoresBolos();  // Guarda los valores de los inputs
                ejercicioActualIndex = 0;  // Comienza con el primer ejercicio
                sesionActual = 1;  // Comienza con la primera sesión
                video0.SetActive(true);  // Activa el primer video
                video4.SetActive(false);  // Desactiva el video 4
                PanelAnimacionesBolos.SetActive(true);  // Activa el panel de animaciones
de bolos
            }
            else  // Si el tipo es Arco
            {
                GuardarValoresArco();  // Guarda los valores de los inputs
                ejercicioActualIndex = 3;  // Comienza con el cuarto ejercicio (índice 3)
```

```
            sesionActual = 1;  // Comienza con la primera sesión

            video4.SetActive(true);  // Activa el video 4

            video0.SetActive(false);  // Desactiva el primer video

            PanelAnimacionesArco.SetActive(true);  // Activa el panel de animaciones de arco

        }

        ActualizarEstadoUI();  // Actualiza la interfaz de usuario

        InstanciarPanelSesion(sesionActual);  // Crea el panel para la sesión actual

    }

    activarPaneles(ejercicioActualIndex);  // Activa los paneles correspondientes al ejercicio actual

}


// Valida si se ha completado la sesión actual y maneja la transición

public void ValidarSesionCompletada()

{

    if (!ejercicioEnProgreso)  // Verifica que haya un ejercicio en progreso

    {

        Debug.Log("No hay un ejercicio en progreso.");
```

```
            return;

        }


        int totalSesiones = ObtenerTotalSesiones(ejercicioActualIndex);  // Obtiene el total de sesiones para el ejercicio actual

        if (sesionActual >= totalSesiones)  // Si se ha alcanzado el total de sesiones

        {

            if ((ejercicioActualIndex == 2 && tipoEjercicio == "Bolos") || (ejercicioActualIndex == 4 && tipoEjercicio == "Arco"))  // Si se ha completado el último ejercicio

            {

                Debug.Log("Todos los ejercicios han sido completados!");

                infoFinalPanel.SetActive(true);  // Muestra el panel de finalización

                ejercicioEnProgreso = false;  // Finaliza el progreso

                return;

            }


            videoEjercicios(ejercicioActualIndex);  // Actualiza el video según el ejercicio

            LimpiarPaneles();  // Limpia los paneles existentes


            infoCorrectoPanel.SetActive(false);  // Oculta el panel de correcto
```

```
        EjercicioFinalizadoPanel.SetActive(true);  // Muestra el panel de ejercicio finalizado

        infoEjercicioPanelText.text = $"Ejercicio {ejercicios[ejercicioActualIndex]} completado.";  // Actualiza el texto informativo



        ejercicioActualIndex++;  // Avanza al siguiente ejercicio

        activarPaneles(ejercicioActualIndex);  // Activa los paneles para el nuevo ejercicio

        sesionActual = 1;  // Reinicia a la primera sesión

        InstanciarPanelSesion(sesionActual);  // Crea el panel para la nueva sesión

    }

    else  // Si no se ha completado todas las sesiones

    {

        ActivarImagenUltimoPanel();  // Activa la imagen del último panel (marca como completado)

        sesionActual++;  // Avanza a la siguiente sesión

        Debug.Log($"Continuando ejercicio: {ejercicios[ejercicioActualIndex]} - Serie {sesionActual}");

        InstanciarPanelSesion(sesionActual);  // Crea el panel para la nueva sesión

    }



    ActualizarEstadoUI();  // Actualiza la interfaz de usuario
```

```
}

// Controla qué videos se muestran según el ejercicio actual
private void videoEjercicios(int ejercicioIndex)
{
    switch (ejercicioIndex)
    {
        case 0:  // Para el primer ejercicio
            video3.SetActive(false);
            video2.SetActive(false);
            video1.SetActive(true);  // Activa solo el video 1
            break;
        case 1:  // Para el segundo ejercicio
            video1.SetActive(false);
            video2.SetActive(true);  // Activa solo el video 2
            video3.SetActive(false);
            break;
        case 3:  // Para el cuarto ejercicio
```

```
            video1.SetActive(false);
            video2.SetActive(false);
            video3.SetActive(true);  // Activa solo el video 3
            break;
    }
}

// Activa los paneles de animación y calibración según el ejercicio actual
private void activarPaneles(int ejercicioIndex)
{
    // Determinar qué paneles deben activarse
    GameObject panelAnimacion = null;
    GameObject panelCalibracion = null;

    switch (ejercicioIndex)
    {
        case 0:  // Péndulo
            panelAnimacion = AnimacionPendulo;
```

```
    panelCalibracion = CalibracionPendulo;
    break;
case 1:  // Fortalecimiento escapular
    panelAnimacion = AnimacionRotacionE;
    panelCalibracion = CalibracionFortaEscap;
    break;
case 2:  // Rotación externa en aducción
    panelAnimacion = AnimacionRotacionEFI;
    panelCalibracion = CalibracionRotaAbdExt;
    break;
case 3:  // Rotación externa (arco)
    panelAnimacion = AnimacionRotacionEA;
    panelCalibracion = CalibracionRotaExtArco;
    break;
case 4:  // Rotación interna (arco)
    panelAnimacion = AnimacionRotacionIA;
    panelCalibracion = CalibracionRotaIntArco;
    break;
```

```
}

// Desactivar todos los paneles de animación y calibración
desactivarTodosPaneles();

// Verificar qué panel principal activar basado en el índice del ejercicio
if (ejercicioIndex <= 2) // Bolos (índices 0, 1, 2)
{
    PanelAnimacionesBolos.SetActive(true);
    PanelAnimacionesArco.SetActive(false);
}
else // Arco (índices 3, 4)
{
    PanelAnimacionesBolos.SetActive(false);
    PanelAnimacionesArco.SetActive(true);
}

// Activar los paneles específicos de animación y calibración
```

```
    if (panelAnimacion != null) panelAnimacion.SetActive(true);
    if (panelCalibracion != null) panelCalibracion.SetActive(true);
}

// Desactiva todos los paneles de animación y calibración
private void desactivarTodosPaneles()
{
    // Desactivar todos los paneles de animación
    AnimacionPendulo.SetActive(false);
    AnimacionRotacionE.SetActive(false);
    AnimacionRotacionEFI.SetActive(false);
    AnimacionRotacionEA.SetActive(false);
    AnimacionRotacionIA.SetActive(false);

    // Desactivar todos los paneles de calibración
    CalibracionPendulo.SetActive(false);
    CalibracionFortaEscap.SetActive(false);
    CalibracionRotaAbdExt.SetActive(false);
```

```
        CalibracionRotaExtArco.SetActive(false);
        CalibracionRotaIntArco.SetActive(false);
    }

    // Obtiene el número total de sesiones para un ejercicio dado
    private int ObtenerTotalSesiones(int ejercicioIndex)
    {
        switch (ejercicioIndex)
        {
            case 0: return sesionesPendulo;          // Sesiones para péndulo
            case 1: return sesionesRotacionE;        // Sesiones para rotación escapular
            case 2: return sesionesRotacionEFI;      // Sesiones para rotación externa en aducción
            case 3: return sesionesRotacionEA;       // Sesiones para rotación externa de arco
            case 4: return sesionesRotacionIA;       // Sesiones para rotación interna de arco
            default: return 0;                       // Valor por defecto
        }
    }
```

```
    // Obtiene el número total de repeticiones para un ejercicio dado
    private int ObtenerTotalRepeticiones(int ejercicioIndex)
    {
        switch (ejercicioIndex)
        {
            case 0: return repeticionesPendulo;      // Repeticiones para péndulo
            case 1: return repeticionesRotacionE;        // Repeticiones para rotación escapular
            case 2: return repeticionesRotacionEFI;  // Repeticiones para rotación externa en aducción
            case 3: return repeticionesRotacionEA;   // Repeticiones para rotación externa de arco
            case 4: return repeticionesRotacionIA;   // Repeticiones para rotación interna de arco
            default: return 0;                       // Valor por defecto
        }
    }

    // Actualiza la información mostrada en la interfaz de usuario
    private void ActualizarEstadoUI()
```

```
    {
        if (ejercicioEnProgreso)  // Solo actualiza si hay un ejercicio en progreso
        {
            infoRepeticionText.text = $"{ObtenerTotalRepeticiones(ejercicioActualIndex)} Repeticiones";  // Muestra las repeticiones
            infoEjercicioText.text = $"Ejercicio: {ejercicios[ejercicioActualIndex]}"; // Muestra el nombre del ejercicio
            infoSesionText.text = $"Serie: {sesionActual}"; // Muestra la serie actual
        }
    }

    // Actualiza los textos en los paneles de retroalimentación
    public void ActualizarEstadoPaneles()
    {
        if (ejercicioEnProgreso)  // Solo actualiza si hay un ejercicio en progreso
        {
            Debug.Log(ejercicios[ejercicioActualIndex]);
            infoIncorrectoText.text = $"Serie: {sesionActual} fallida del ejercicio {ejercicios[ejercicioActualIndex]}";  // Texto para serie fallida
```

```
            infoCorrectoText.text = $"Serie: {sesionActual} del ejercicio {ejercicios[ejercicioActualIndex]} realizada con éxito";    // Texto para serie exitosa

        }

    }


    // Crea un nuevo panel para la sesión indicada

    private void InstanciarPanelSesion(int numeroSesion)

    {

        Vector3 nuevaPosicion = posicionInicial;  // Posición inicial por defecto


        if (instantiatedPanels.Count > 0)  // Si ya hay paneles, calcula la posición basada en el último

        {

            GameObject ultimoPanel = instantiatedPanels[instantiatedPanels.Count - 1];

            nuevaPosicion = new Vector3(ultimoPanel.transform.localPosition.x,

                                        ultimoPanel.transform.localPosition.y + offsetY,

                                        ultimoPanel.transform.localPosition.z);

        }


        GameObject nuevoPanel = Instantiate(seriesPrefab, panelContainer);     // Instancia un nuevo panel
```

```
        nuevoPanel.transform.localPosition = nuevaPosicion;  // Asigna la posición calculada



        // Desactivar la imagen inicialmente (el indicador de completado)

        Image imagenPanel = nuevoPanel.GetComponentInChildren<Image>();

        if (imagenPanel != null)

        {

            imagenPanel.gameObject.SetActive(false);

        }



        nuevoPanel.GetComponentInChildren<TextMeshProUGUI>().text = $"Serie {numeroSesion}";  // Establece el texto con el número de serie

        instantiatedPanels.Add(nuevoPanel);  // Añade el panel a la lista de paneles instanciados

    }



    // Activa la imagen del último panel para indicar que se completó

    private void ActivarImagenUltimoPanel()

    {

        if (instantiatedPanels.Count > 0)  // Si hay paneles instanciados

        {
```

```
            GameObject ultimoPanel = instantiatedPanels[instantiatedPanels.Count - 1];

            Image imagenPanel = ultimoPanel.GetComponentInChildren<Image>();

            if (imagenPanel != null)

            {

                imagenPanel.gameObject.SetActive(true);  // Activa la imagen que indica completado

            }

        }

    }


    // Limpia todos los paneles y oculta los paneles de información

    private void LimpiarPaneles02()

    {

        infoFinalPanel.SetActive(false);          // Oculta el panel final

        infoCorrectoPanel.SetActive(false);       // Oculta el panel de correcto

        EjercicioFinalizadoPanel.SetActive(false);// Oculta el panel de ejercicio finalizado

        LimpiarPaneles();                         // Limpia los paneles de sesiones

    }
```

```
    // Limpia los paneles de sesiones
    private void LimpiarPaneles()
    {
        foreach (GameObject panel in instantiatedPanels)  // Para cada panel instanciado
        {
            Destroy(panel);  // Destruye el objeto
        }
        instantiatedPanels.Clear();  // Limpia la lista de paneles
    }

    // ---------- MÉTODO NUEVO: Recibir datos procesados de enviodedatos ----------

    // Obtiene el número de repeticiones configuradas para el ejercicio actual
    public int GetRepeticionesActuales()
    {
        int rep = 0;
        if (tipoEjercicio == "Bolos")  // Si es tipo Bolos
        {
```

```
        if (ejercicioActualIndex == 0)
            int.TryParse(repPendulo.text, out rep);  // Péndulo
        else if (ejercicioActualIndex == 1)
            int.TryParse(repRotacionE.text, out rep);  // Rotación escapular
        else if (ejercicioActualIndex == 2)
            int.TryParse(repRotacionEFI.text, out rep);  // Rotación externa en aducción
    }
    else // Si es tipo Arco
    {
        if (ejercicioActualIndex == 3)
            int.TryParse(repRotacionEA.text, out rep);  // Rotación externa arco
        else if (ejercicioActualIndex == 4)
            int.TryParse(repRotacionIA.text, out rep);  // Rotación interna arco
    }
    return rep;  // Devuelve el número de repeticiones
}

// Obtiene el número de repeticiones para un ejercicio específico por nombre
```

```
public int GetRepeticionesPorEjercicio(string ejercicio)
{
    int rep = 0;
    if (tipoEjercicio == "Bolos")  // Si es tipo Bolos
    {
        if(ejercicio.Equals("Péndulo"))
            int.TryParse(repPendulo.text, out rep);  // Péndulo
        else if(ejercicio.Equals("Fortalecimiento escapular"))
            int.TryParse(repRotacionE.text, out rep);  // Rotación escapular
        else if(ejercicio.Equals("Rotación externa en aducción"))
            int.TryParse(repRotacionEFI.text, out rep);  // Rotación externa en aducción
    }
    else // Si es tipo Arco
    {
        if(ejercicio.Equals("Rotación externa"))
            int.TryParse(repRotacionIA.text, out rep);  // Rotación externa
        else if(ejercicio.Equals("Rotación interna"))
            int.TryParse(repRotacionEA.text, out rep);  // Rotación interna
```

```
    }

    return rep;  // Devuelve el número de repeticiones

}


// Recibe los datos de una serie capturada por enviodedatos

public void RecibirDatosSerie(List<List<float>> datosSerie)

{

    // Obtiene las repeticiones actuales para el ejercicio

    int repeticionesSerie = GetRepeticionesActuales();

    RegistrarSerie(datosSerie, repeticionesSerie);  // Registra la serie con los datos


    // Notifica a DropdownManager para que actualice su lista

    DropdownManager dm = FindObjectOfType<DropdownManager>();

    if (dm != null)

        dm.ActualizarDropdownEjercicios();

}


// Registra una serie de datos para el ejercicio y sesión actuales

public void RegistrarSerie(List<List<float>> datosSerie, int repeticionesSerie)
```

```
    {
        string ejercicioActual = ejercicios[ejercicioActualIndex];  // Obtiene el nombre del ejercicio actual


        // Crear la entrada para el ejercicio si no existe
        if (!datosPorEjercicio.ContainsKey(ejercicioActual))
        {
            datosPorEjercicio[ejercicioActual] = new Dictionary<int, Dictionary<int, SerieData>>();
        }


        // Crear la entrada para la sesión actual si no existe
        if (!datosPorEjercicio[ejercicioActual].ContainsKey(sesionActual))
        {
            datosPorEjercicio[ejercicioActual][sesionActual] = new Dictionary<int, SerieData>();
        }


        // El número de serie es el conteo actual + 1
        int serieNumber = datosPorEjercicio[ejercicioActual][sesionActual].Count + 1;
```

```
        // Almacena los datos en la estructura

        datosPorEjercicio[ejercicioActual][sesionActual][serieNumber] = new SerieData(datosSerie, repeticionesSerie);

        Debug.Log($"Registrada serie {serieNumber} para {ejercicioActual} en sesión {sesionActual} con {repeticionesSerie} repeticiones.");

    }



    // Permite reintentar la última toma de datos para el ejercicio y sesión actuales

    public void ReintentarToma()

    {

        // Usamos el ejercicio actual

        string ejercicio = ejercicios[ejercicioActualIndex];



        // Verificamos que exista la entrada para este ejercicio y la sesión actual

        if (datosPorEjercicio.ContainsKey(ejercicio) && datosPorEjercicio[ejercicio].ContainsKey(sesionActual))

        {

            // Buscamos la última toma registrada en esa sesión

            int ultimaToma = 0;
```

```
        foreach (var key in datosPorEjercicio[ejercicio][sesionActual].Keys)
        {
            if (key > ultimaToma)
                ultimaToma = key;  // Encuentra la clave más alta (última toma)
        }


        if (ultimaToma > 0)
        {
            // Elimina la última toma para permitir un reintento
            datosPorEjercicio[ejercicio][sesionActual].Remove(ultimaToma);
            Debug.Log($"Se eliminó la toma {ultimaToma} del ejercicio {ejercicio} en la serie {sesionActual} para reintento.");
        }
        else
        {
            Debug.Log("No se encontró ninguna toma para reintentar.");
        }
    }
```

```
        else

        {

            Debug.Log("No se encontró la información del ejercicio o de la serie actual para reintentar.");

        }

    }

    // Método que reorganiza los datos de los ejercicios para facilitar su uso en otras partes de la aplicación

    public Dictionary<string, Dictionary<int, List<List<float>>>> ObtenerDatosPorEjercicio()

    {

        // Creamos un diccionario donde:

        // - La clave es el nombre del ejercicio

        // - El valor es otro diccionario donde la clave es un ID de serie y el valor son los datos de esa serie

        var result = new Dictionary<string, Dictionary<int, List<List<float>>>>();



        // Iteramos por cada ejercicio almacenado en la estructura datosPorEjercicio

        foreach (var ejercicioEntry in datosPorEjercicio)

        {
```

```
        string ejercicio = ejercicioEntry.Key;

        // Creamos un diccionario para almacenar todas las series de este ejercicio con un nuevo contador

        var seriesCombinadas = new Dictionary<int, List<List<float>>>();

        int contadorSerie = 1; // Inicializamos un contador para asignar IDs únicos a las series



        // Iteramos por cada sesión de este ejercicio

        foreach (var sesionEntry in ejercicioEntry.Value)

        {

            // Iteramos por cada serie dentro de la sesión actual

            foreach (var serieEntry in sesionEntry.Value)

            {

                // Verificamos que no haya IDs duplicados incrementando el contador si es necesario

                while (seriesCombinadas.ContainsKey(contadorSerie))

                {

                    contadorSerie++;

                }
```

```
                // Añadimos la serie con su nuevo ID al diccionario de series combinadas

                // Copiamos los datos de la serie desde la propiedad "datos"

                seriesCombinadas[contadorSerie] = new List<List<float>>(serieEntry.Value.datos);

                contadorSerie++;

            }

        }

        // Añadimos el ejercicio con todas sus series combinadas al diccionario final

        result.Add(ejercicio, seriesCombinadas);

    }

    // Devolvemos el diccionario con todos los ejercicios y sus series reorganizadas

    return result;

}

    // Método que prepara todos los datos de la sesión en el formato adecuado para su almacenamiento en Firebase

public Dictionary<string, object> ObtenerDatosSesionCompletaParaFirebase()

{

    // Diccionario principal que contendrá toda la información de la sesión

    Dictionary<string, object> result = new Dictionary<string, object>();
```

```
// Añadimos los metadatos generales de la sesión

result.Add("fecha", DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss")); // Fecha y hora actual como timestamp

result.Add("nota", inputNota.text); // Nota del usuario sobre la sesión

result.Add("tipoJuego", tipoEjercicio); // Tipo de ejercicio o juego realizado



// Diccionario para almacenar todos los ejercicios de la sesión

Dictionary<string, object> ejerciciosDict = new Dictionary<string, object>();



// Recorremos cada ejercicio registrado en la estructura de datos

foreach (var ejercicioEntry in datosPorEjercicio)

{

    string ejercicio = ejercicioEntry.Key; // Nombre del ejercicio

    Dictionary<string, object> seriesDict = new Dictionary<string, object>(); // Diccionario para las series de este ejercicio



    // Recorremos cada sesión de este ejercicio (que representa conjuntos de series)

    foreach (var sesionEntry in ejercicioEntry.Value)

    {
```

```
            // Para cada sesión, recorremos sus series ordenadas por su clave
            foreach (var serieEntry in sesionEntry.Value.OrderBy(kvp => kvp.Key))
            {
                // Generamos una clave única para esta serie basada en el número de sesión
                string serieKey = $"Serie {sesionEntry.Key}";

                // Creamos un diccionario con los datos de la serie
                Dictionary<string, object> serieData = new Dictionary<string, object>
                {
                    // Usamos datosJson que ya contiene los datos serializados
                    { "datos", serieEntry.Value.datosJson },
                    // Añadimos el número de repeticiones realizadas
                    { "repeticiones", serieEntry.Value.repeticiones }
                };

                // Añadimos la serie al diccionario de series de este ejercicio
                seriesDict.Add(serieKey, serieData);
            }
```

```
            }


            // Solo añadimos el ejercicio si tiene al menos una serie registrada

            if (seriesDict.Count > 0)

            {

                // Estructuramos los datos del ejercicio con sus series

                ejerciciosDict.Add(ejercicio, new Dictionary<string, object> { { "series", seriesDict } });

            }

        }



        // Añadimos todos los ejercicios al resultado final

        result.Add("ejercicios", ejerciciosDict);

        return result; // Devolvemos el diccionario completo listo para Firebase

    }

        // Método que gestiona el proceso de guardar toda la sesión en la base de datos Firebase

    public void GuardarSesionCompletaEnFirebase()

    {

        // Buscamos el componente FirebaseManager en la escena
```

```
        FirebaseManager fbManager = FindObjectOfType<FirebaseManager>();



        // Verificamos que se haya encontrado el componente

        if (fbManager != null)

        {

            // Obtenemos los datos formateados para Firebase

            Dictionary<string, object> datosParaFirebase =
ObtenerDatosSesionCompletaParaFirebase();



            // Llamamos al método del FirebaseManager para almacenar los datos

            fbManager.AlmacenarSesionFirebase(datosParaFirebase);



            // Registramos en el log que la operación fue exitosa

            Debug.Log("Sesión completa guardada en Firebase.");



            // Nota: Aquí hay un comentario sobre la posibilidad de limpiar la estructura

            // después de guardar, pero está comentado para que no se ejecute

            // datosPorEjercicio.Clear();
```

```
        }
        else
        {
            // Si no se encuentra el FirebaseManager, registramos un error
            Debug.LogError("FirebaseManager no encontrado en la escena.");
        }
    }
}
```