Práctica: Reproductor de Música en Unity

1. Introducción y Objetivo

En esta práctica de Unity aprenderemos:

- El sistema de UI de Unity (Canvas, TextMeshPro, Buttons, Sliders, Toggles).
- La gestión básica de Audio (AudioSource, AudioClip).
- La creación y conexión de scripts C# para controlar la lógica de la aplicación.

Al final de esta práctica, tendrás un reproductor como el que se muestra a continuación:

Nota: Esta práctica asume que ya tienes los assets necesarios descargados del Classroom (imágenes para los botones/fondos, archivos de audio .mp3, .wav u .ogg, y la fuente

pixelada si deseas replicar el estilo exacto).

2. Configuración Inicial del Proyecto

1. Crea un nuevo proyecto 2D en Unity Hub. Puedes llamarlo "MusicPlayer".
2. Importa los assets necesarios a tu proyecto:
   - Crea carpetas en la ventana `Project` (por ejemplo: `Audio`, `Sprites`, `Scripts`, `Fonts`).
   - Arrastra tus archivos de audio a la carpeta `Audio`.
   - Arrastra tus imágenes (botones, fondos, etc.) a la carpeta `Sprites`.
   - (Opcional) Arrastra tu fuente a la carpeta `Fonts`.
3. **Importante: TextMeshPro.** Si es la primera vez que usas UI en este proyecto, Unity te pedirá importar los "TMP Essentials". Haz clic en "Import TMP Essentials". Si no aparece, puedes ir a `Window` > `TextMeshPro` > `Import TMP Essential Resources`.
4. En la jerarquía (`Hierarchy`), crea un GameObject vacío y llámalo `MusicPlayerController`. Este objeto contendrá nuestro script principal más adelante.
5. Asegúrate de que tienes un `EventSystem` en tu escena. Si no lo tienes, créalo haciendo clic derecho en la jerarquía -> `UI` -> `Event System`. Es esencial para que la UI funcione.

Tu jerarquía inicial debería tener al menos `Main Camera`, `MusicPlayerController` y `EventSystem`.

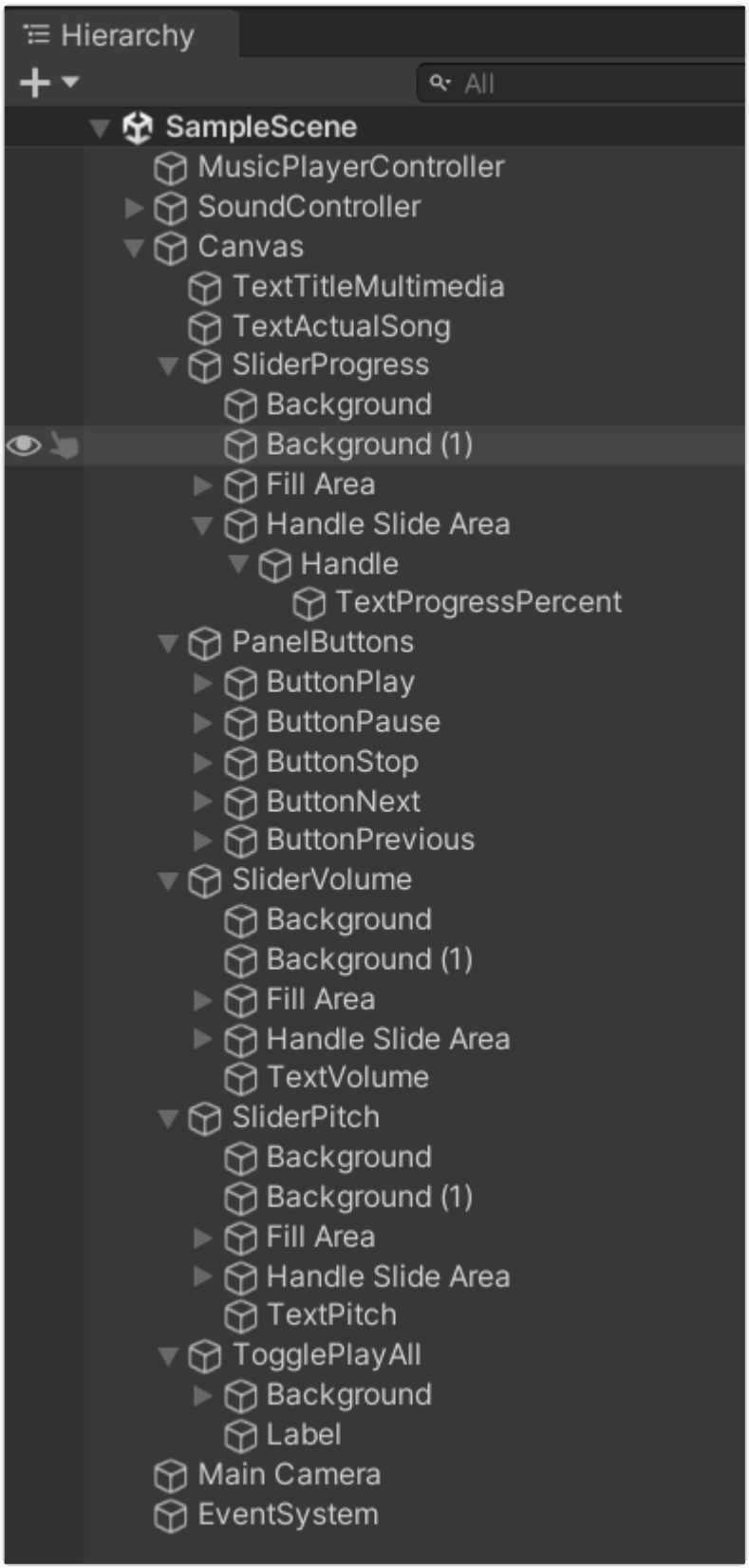
3. Creando la Base de la Interfaz (UI)

1. **Crear el Canvas:** Haz clic derecho en la Jerarquía -> `UI` -> `Canvas`. Renómbralo a `Canvas`.
   - Selecciona el `Canvas`. En el Inspector, busca el componente `Canvas Scaler`.
   - Cambia `UI Scale Mode` a `Scale With Screen Size`.
   - Establece una `Reference Resolution` (por ejemplo, 1920x1080 o una resolución que se ajuste a tu diseño).
   - Ajusta `Match` según prefieras (0 = Width, 1 = Height, 0.5 = mezcla).
2. **(Opcional) Añadir un Fondo:** Si tienes una imagen de fondo para el reproductor, haz clic

derecho en `Canvas` -> `UI` -> `Image`. Renómbrala a `BackgroundImage`, ajústala para que cubra el área deseada y asígnale tu sprite de fondo en el campo `Source Image` del componente `Image`. Asegúrate de que esté detrás de otros elementos reordenándola en la jerarquía (arrastrarla hacia arriba).

3. **Panel para Botones:** Para organizar los botones de control (Play, Pause, etc.), crea un panel. Haz clic derecho en `Canvas` -> `UI` -> `Panel`. Renómbralo a `PanelButtons`.
    - Puedes quitarle el componente `Image` si no quieres que tenga fondo visible, o ajustar su color y transparencia.
    - Posiciona y dimensiona este panel donde irán los botones centrales. Puedes usar `Layout Groups` (como `Horizontal Layout Group` o `Grid Layout Group`) dentro del panel para organizar los botones automáticamente si lo deseas.

Tu jerarquía ahora debería parecerse un poco a esto (sin todos los elementos aún):



4. Añadiendo Elementos de Texto (TextMeshPro)

Usaremos TextMeshPro para un mejor control visual del texto. Si no importaste los "TMP Essentials" antes, hazlo ahora (`Window` > `TextMeshPro` > `Import TMP Essential

Resources`).

1. **Título Multimedia:** Haz clic derecho en `Canvas` -> `UI` -> `Text - TextMeshPro`. Renómbralo a `TextTitleMultimedia`.
   - En el Inspector, escribe el título (ej: "MUSIC PLAYER").
   - Ajusta su posición, tamaño, fuente (si importaste una), tamaño de fuente, color y alineación.
2. **Texto Canción Actual:** Crea otro `Text - TextMeshPro` (hijo de `Canvas`). Renómbralo a `TextActualSong`.
   - Posiciónalo debajo del título. Este texto mostrará el nombre de la canción (ej: "Playing: Song 1"). Lo actualizaremos por script.
   - Ajusta su estilo como el título.
3. **Porcentaje Progreso:** Crea otro `Text - TextMeshPro` (hijo de `Canvas`). Renómbralo a `TextProgressPercent`.
   - Posiciónalo cerca del slider de progreso (que crearemos luego), normalmente al inicio o al final. Mostrará el tiempo (ej: "0:34 / 3:15").
   - Ajusta su estilo.
4. **Etiquetas Volumen y Pitch:** Crea dos `Text - TextMeshPro` más (hijos de `Canvas`). Renómbralos a `TextVolume` y `TextPitch`.
   - Escribe "VOLUME" y "PITCH" respectivamente.
   - Posiciónalos encima o cerca de sus respectivos sliders (que crearemos más adelante).
   - Ajusta su estilo.

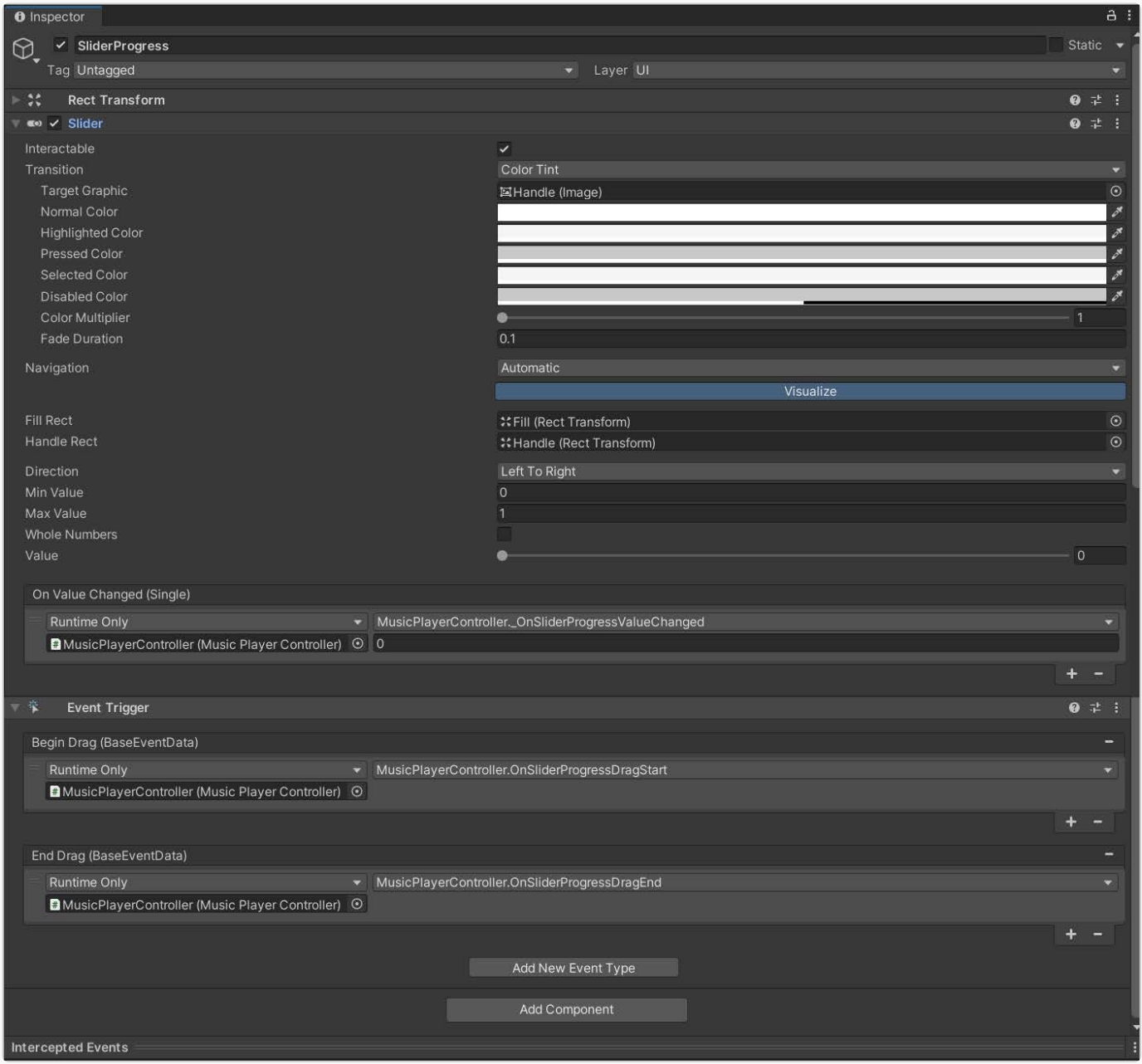
Organiza estos elementos en el `Canvas` para que coincidan con el diseño de la imagen de previsualización.

5. Creando el Slider de Progreso

1. Haz clic derecho en `Canvas` -> `UI` -> `Slider`. Renómbralo a `SliderProgress`.
2. Posiciona y dimensiona el slider debajo de `TextActualSong`.
3. **Configuración del Slider:** Selecciona `SliderProgress`. En el Inspector, busca el componente `Slider`.
   - Marca `Interactable` si quieres que el usuario pueda pinchar y arrastrar para buscar en la canción.
   - Asegúrate de que `Direction` sea `Left To Right`.
   - `Min Value` debe ser 0.
   - `Max Value` debe ser 1. Esto facilita el cálculo del progreso como un porcentaje (0 a 1).
   - `Value` inicial debe ser 0.
   - Puedes desmarcar `Whole Numbers`.
4. **Apariencia (Opcional pero recomendado):**
   - Expande `SliderProgress` en la jerarquía. Verás `Background`, `Fill Area`, y `Handle Slide Area`.

- Selecciona `Background` y `Fill Area` -> `Fill`. Asígnales los sprites o colores deseados en sus componentes `Image`. El `Fill` es la parte que se llena.
- Selecciona `Handle Slide Area` -> `Handle`. Asígnale el sprite o color deseado para el control deslizante. Puedes ajustar su tamaño. Si no quieres un handle visible, puedes desactivar el GameObject `Handle` o hacer su imagen transparente.

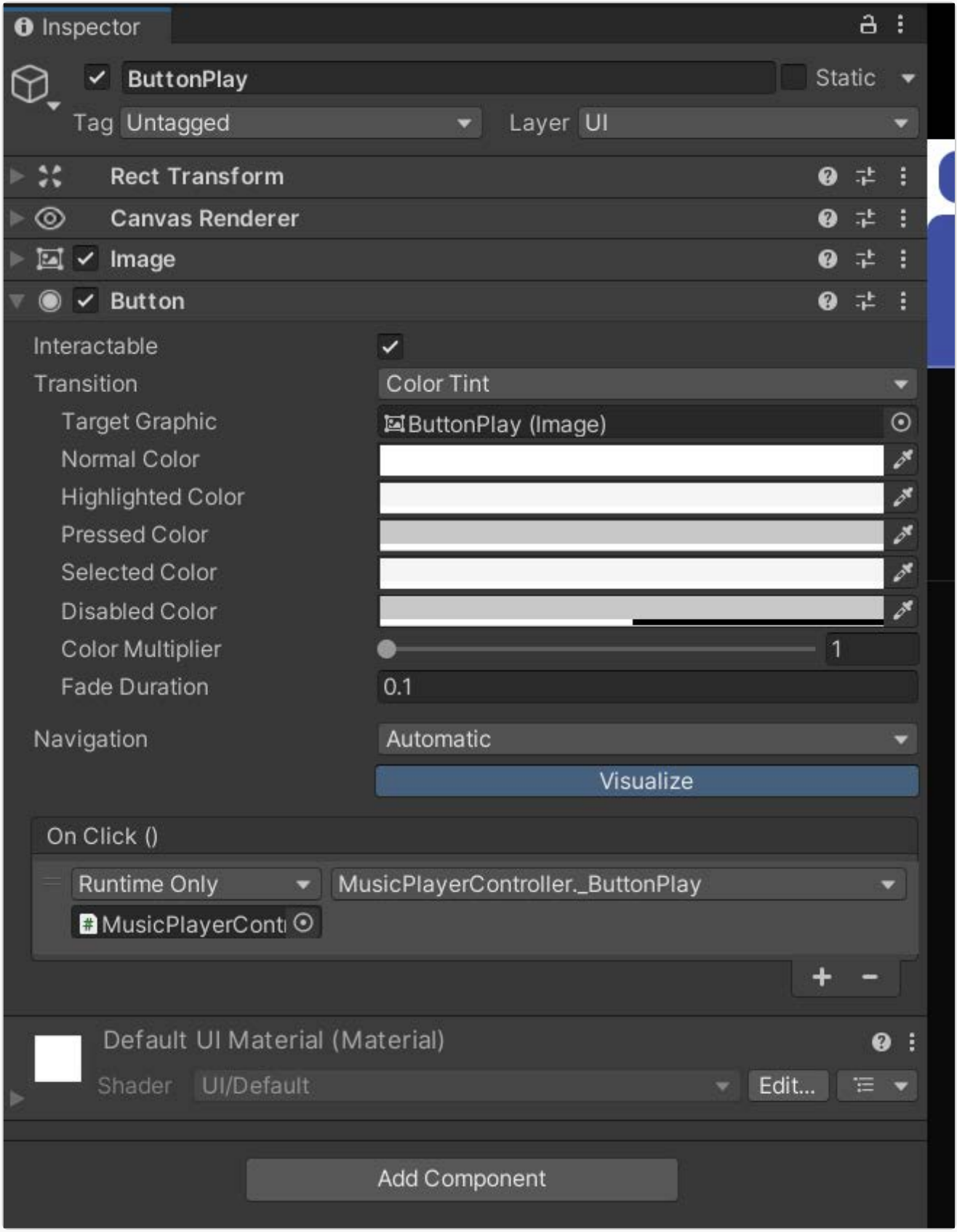
5. **Eventos (Importante):** En la sección 10 ("Conexión UI-Script"), conectaremos los eventos `On Value Changed`, `Begin Drag`, `Drag`, y `End Drag` para que el slider funcione correctamente con nuestro script.



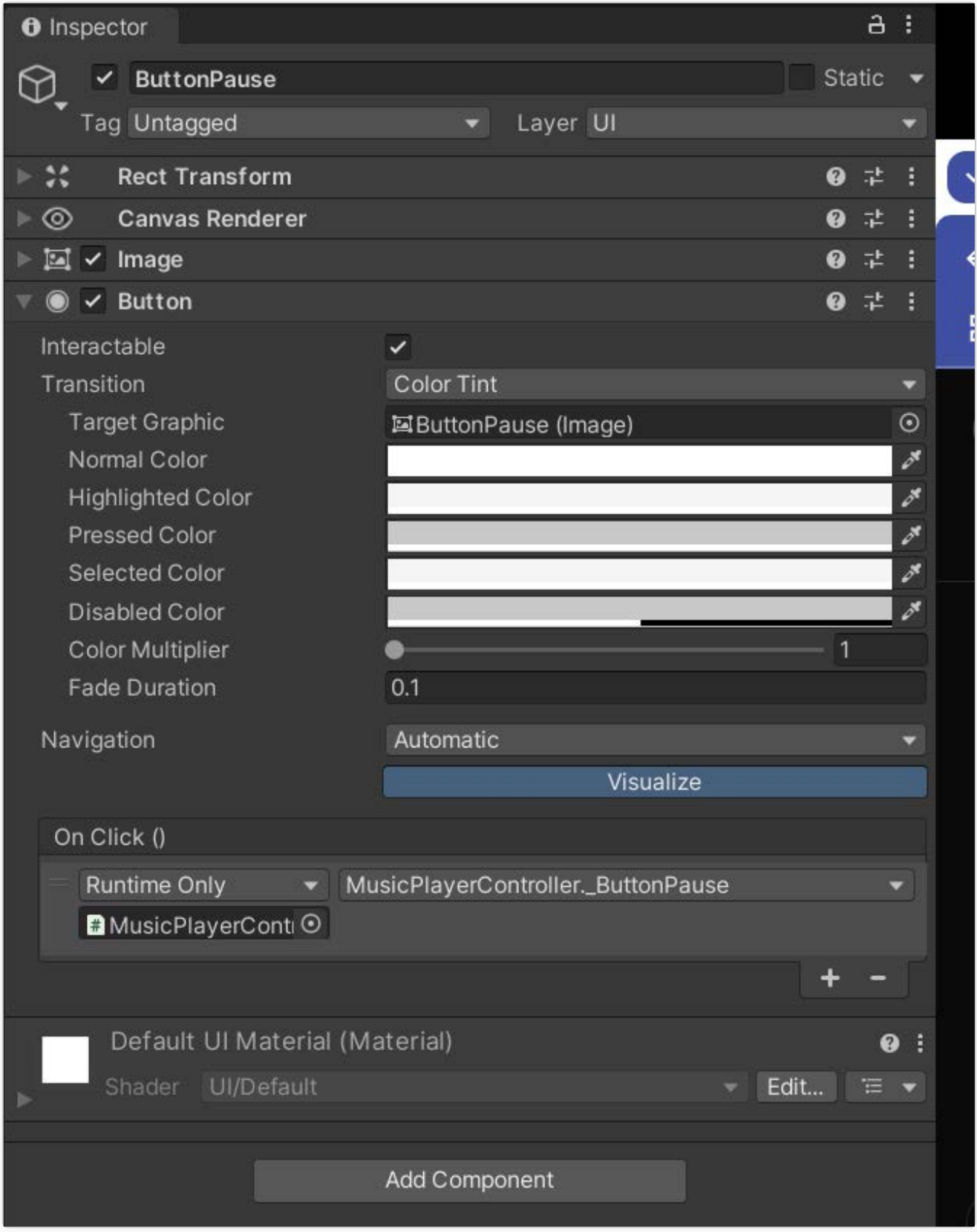
6. Creando los Botones de Control

Crearemos los botones dentro del `PanelButtons` que hicimos antes.

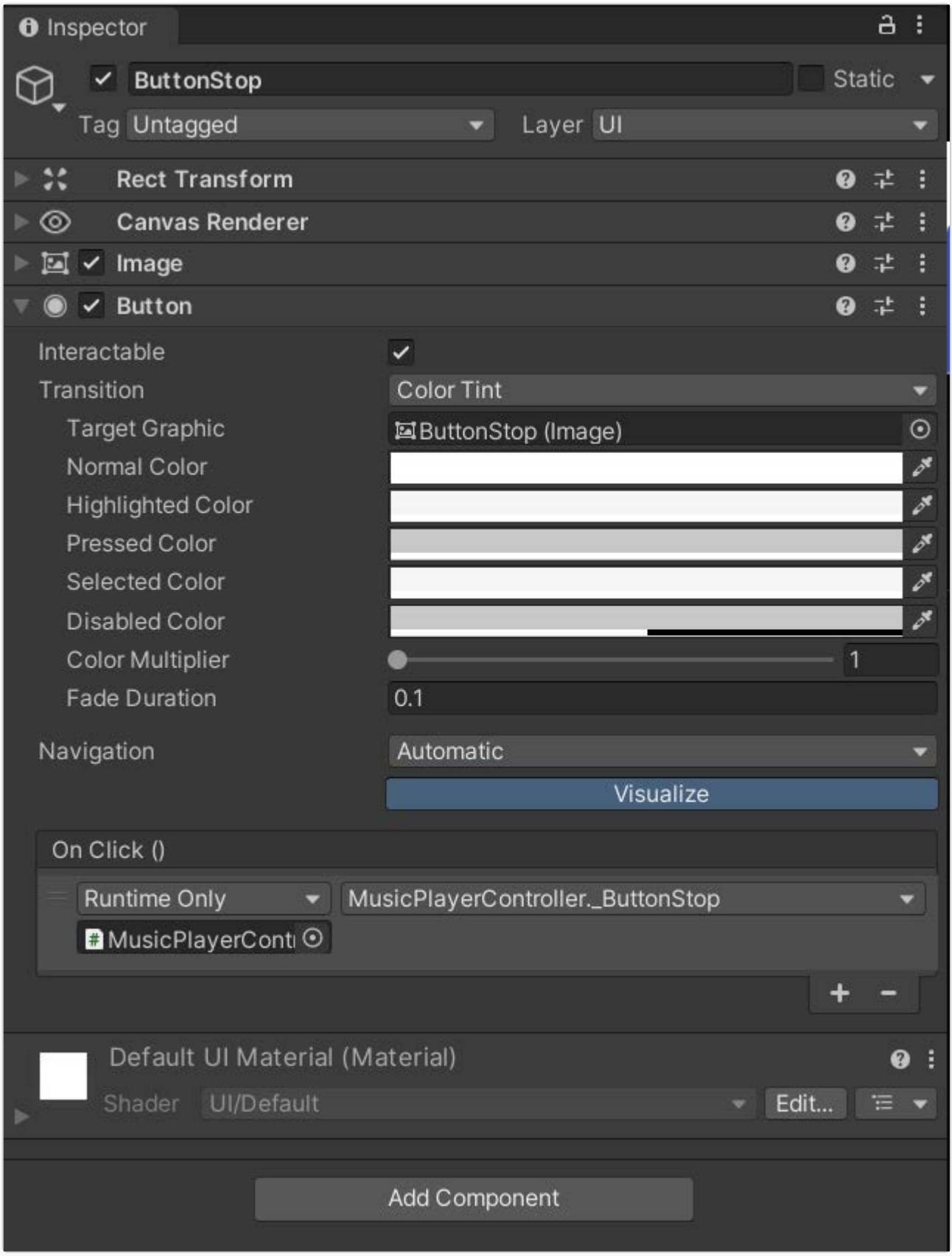
1. **Botón Play:**
    - Haz clic derecho en `PanelButtons` -> `UI` -> `Button - TextMeshPro`. Renómbralo a `ButtonPlay`.
    - Selecciona `ButtonPlay`. En el Inspector, busca el componente `Image` y asígnale el sprite deseado.
    - Expande `ButtonPlay`, selecciona el objeto `Text (TMP)` hijo y cambia su texto a "PLAY". Ajusta estilo.
    - Posiciona el botón.
    - **Evento OnClick:** Lo conectaremos en la sección 10.



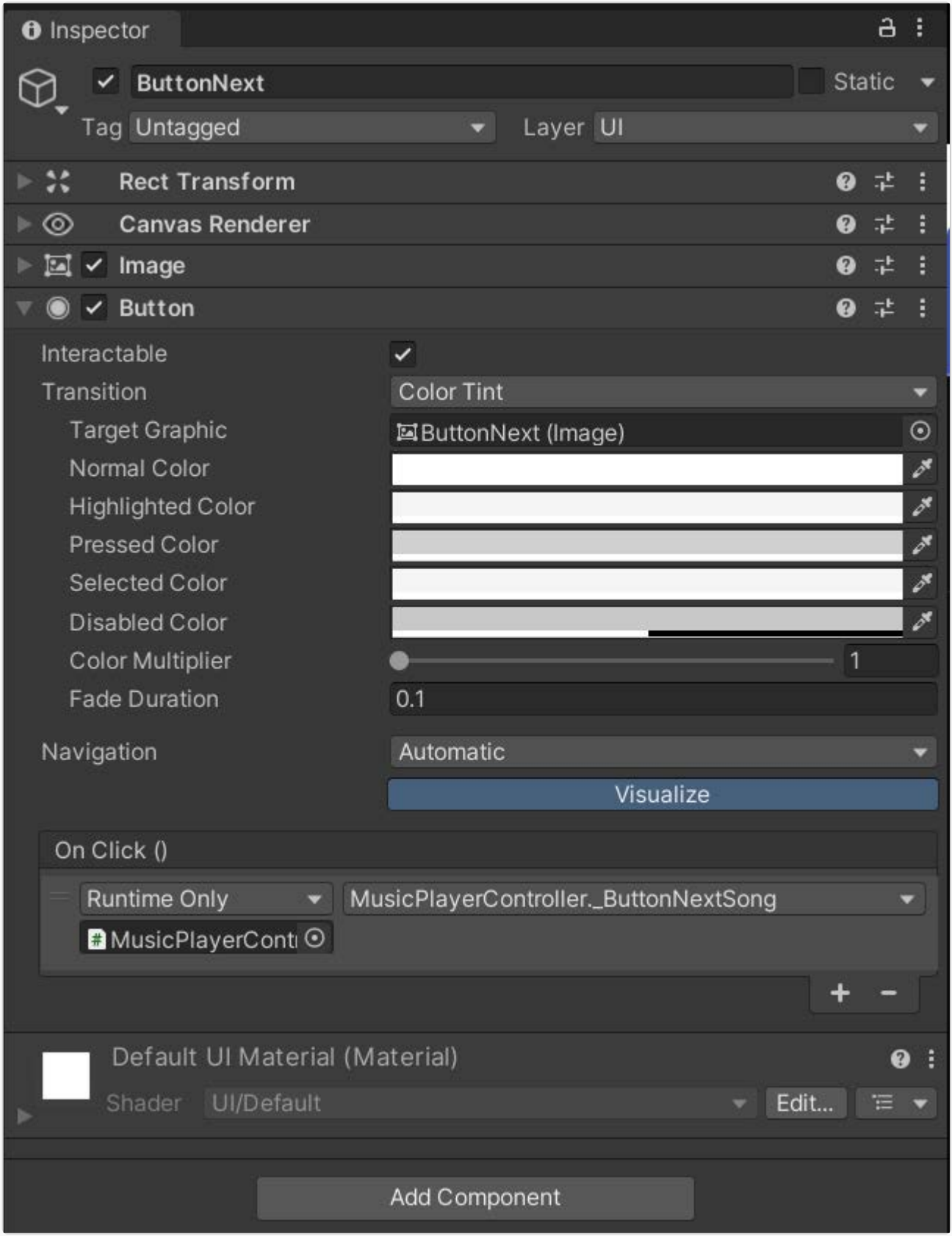
2. **Botón Pause:** Repite el proceso para `ButtonPause` ("PAUSE").



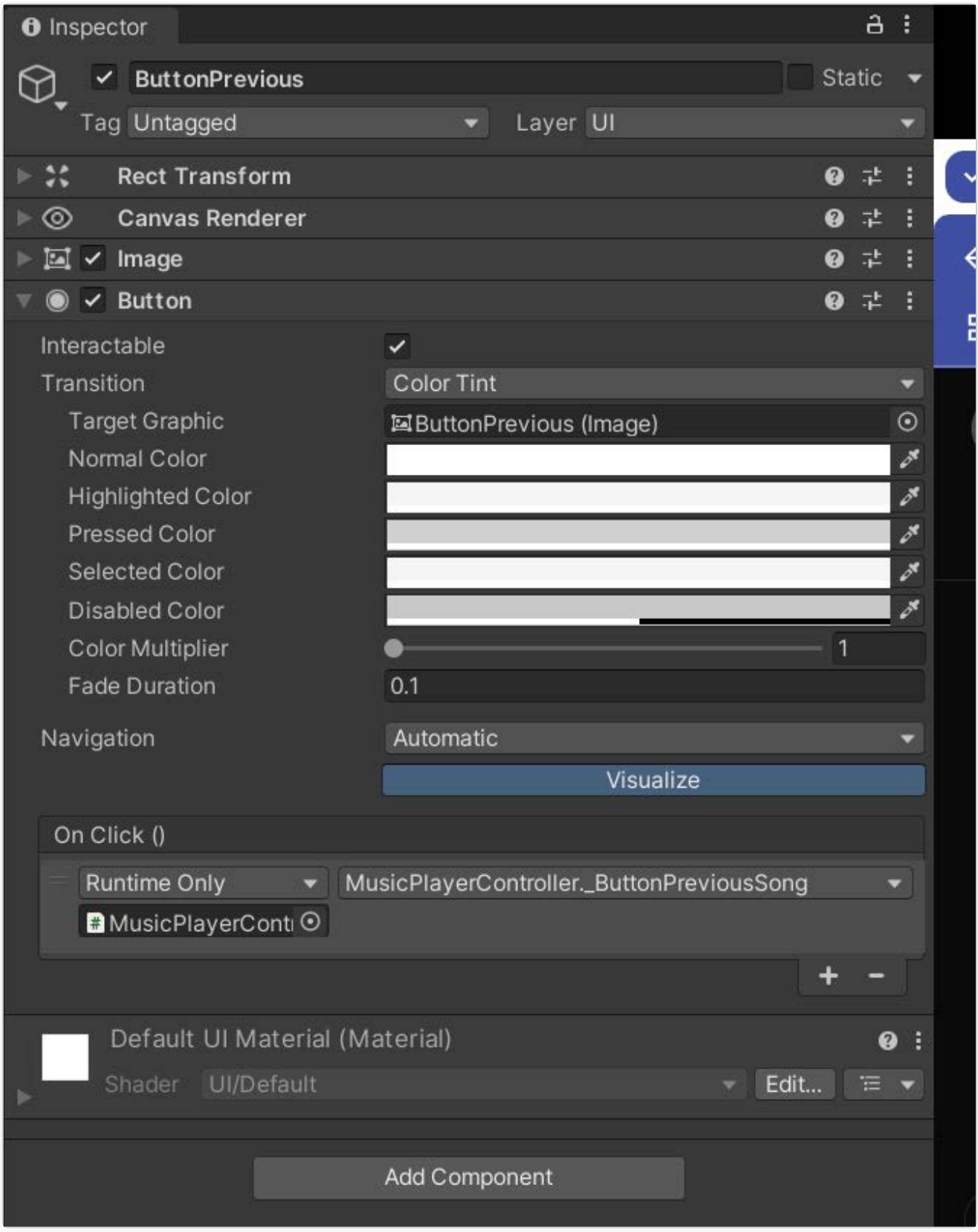
3. **Botón Stop:** Repite para `ButtonStop` ("STOP").



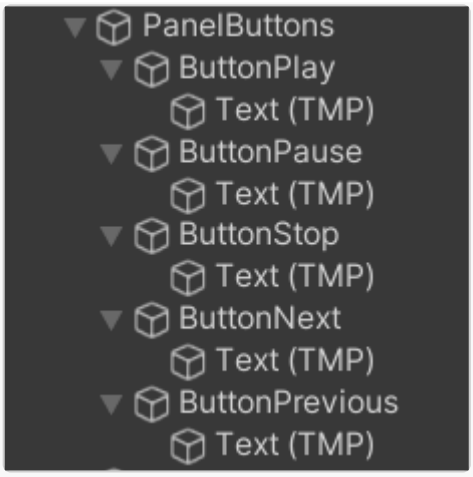
4. **Botón Next:** Repite para `ButtonNext` ("NEXT").



5. **Botón Previous:** Repite para `ButtonPrevious` ("PREV").



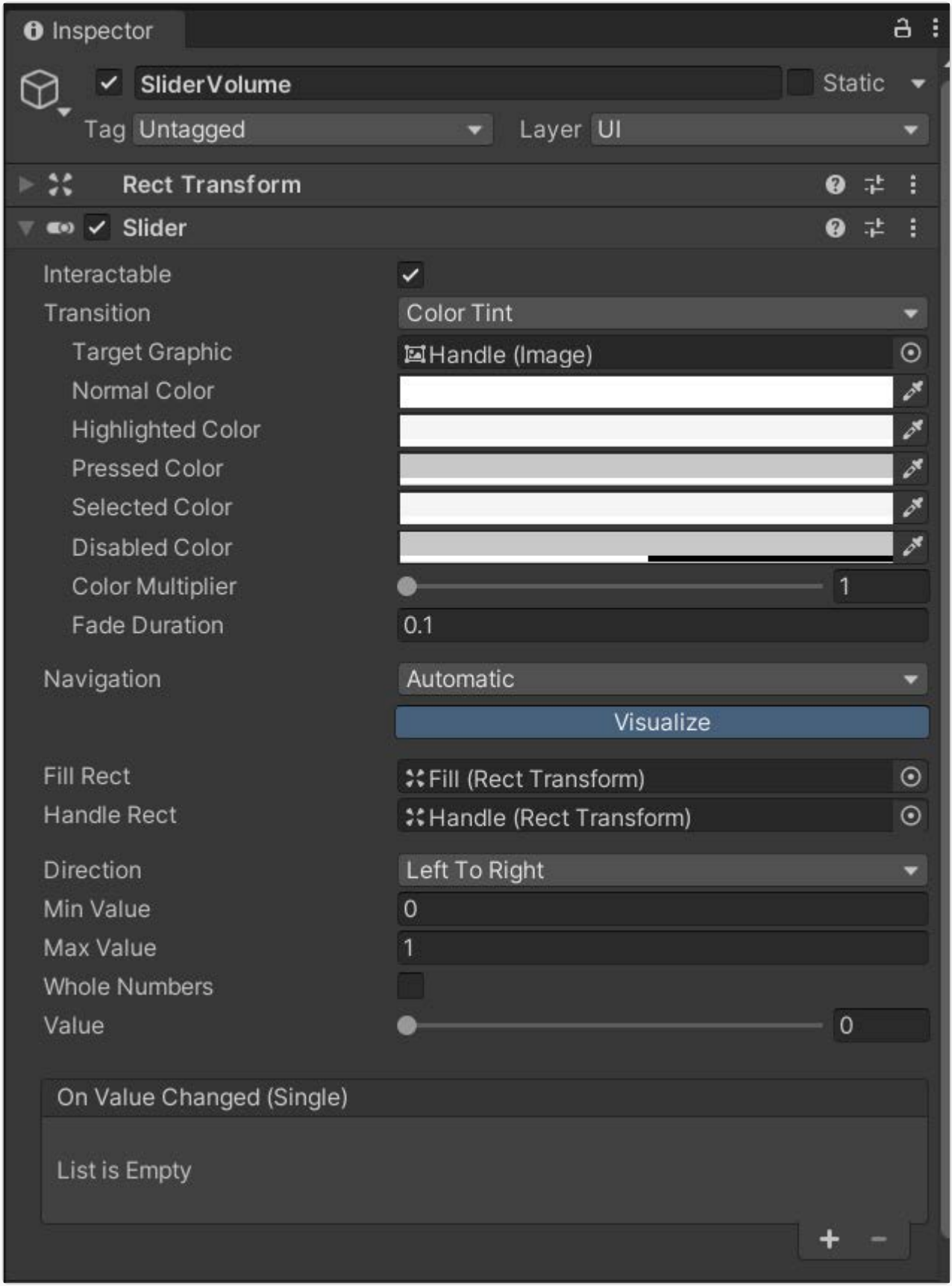
Organiza los botones dentro de `PanelButtons`.



7. Creando los Sliders de Volumen y Pitch

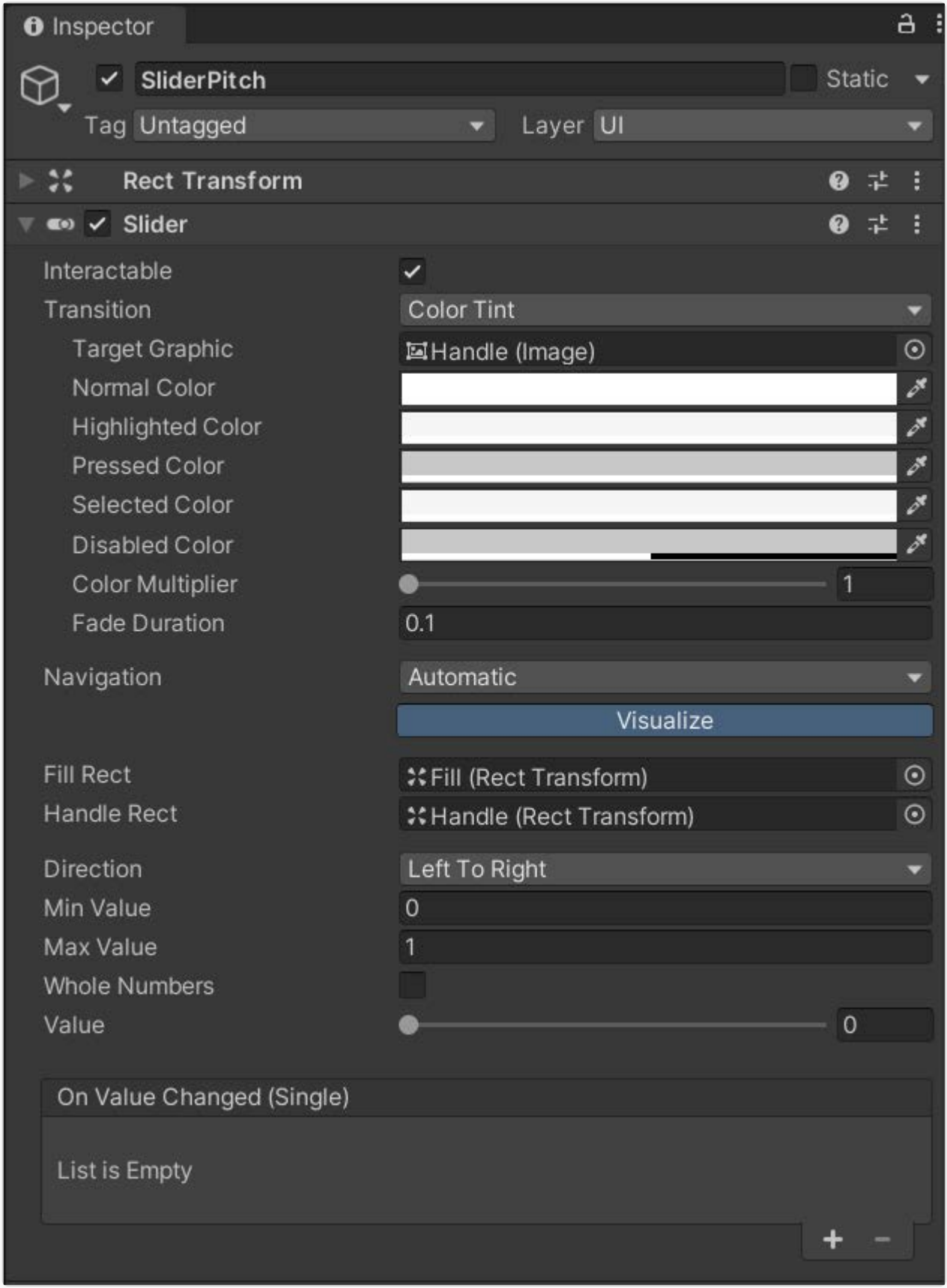
Estos sliders controlarán el volumen y el tono (pitch) del audio.

1. **Slider Volumen:**
   - Haz clic derecho en `Canvas` -> `UI` -> `Slider`. Renómbralo a `SliderVolume`.
   - Posiciónalo cerca de `TextVolume`.
   - Configura el `Slider`: `Interactable` (marcado), `Direction` (Left To Right), `Min Value` (0), `Max Value` (1), `Value` inicial (1), `Whole Numbers` (desmarcado).
   - Estiliza `Background`, `Fill` y `Handle`.
   - **Evento OnValueChanged:** Lo conectaremos en la sección 10.



2. **Slider Pitch:**
   - Crea otro `Slider` -> `SliderPitch`.
   - Posiciónalo cerca de `TextPitch`.
   - Configura el `Slider`: `Interactable` (marcado), `Direction` (Left To Right), `Min Value` (0.5), `Max Value` (2), `Value` inicial (1), `Whole Numbers` (desmarcado).

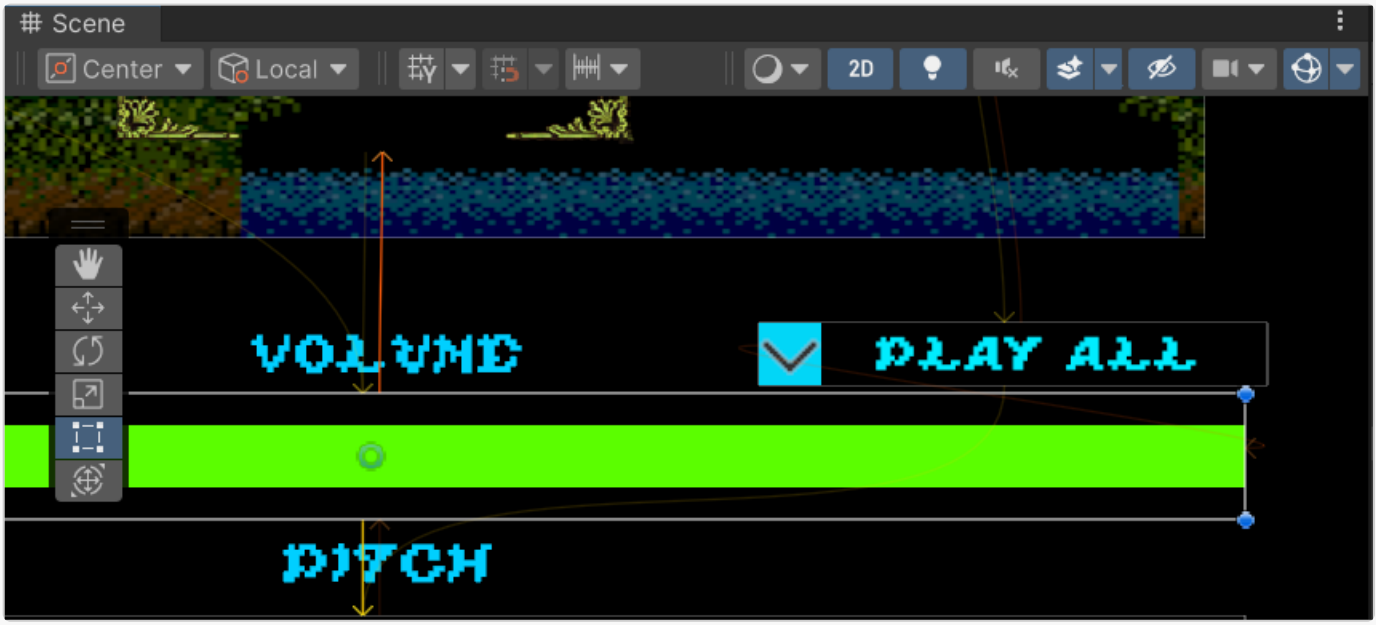
- Estiliza `Background`, `Fill` y `Handle`.
- **Evento OnValueChanged:** Lo conectaremos en la sección 10.

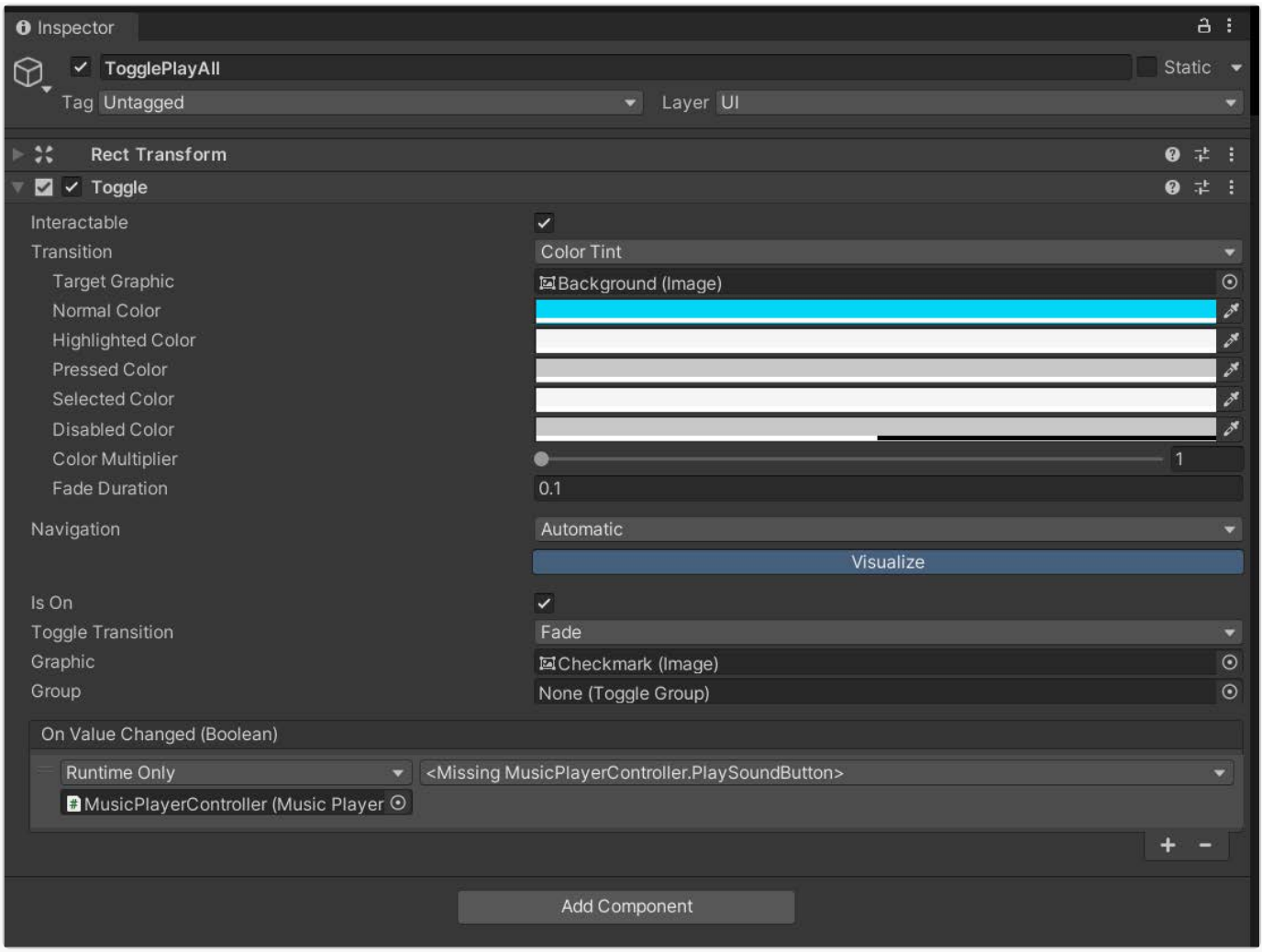


8. Creando el Toggle "Play All"

Este toggle permitirá activar/desactivar la reproducción continua (loop) de la lista de canciones.

1. Haz clic derecho en `Canvas` -> `UI` -> `Toggle`. Renómbralo a `TogglePlayAll`.
2. Posiciónalo en la interfaz.
3. Expande `TogglePlayAll`. Verás `Background`, `Checkmark` (dentro de Background) y `Label`.
    - Asigna sprites/colores a `Background` (estado apagado) y `Checkmark` (estado encendido).
    - Selecciona `Label` (TextMeshPro o Text). Cambia el texto a "PLAY ALL". Ajusta estilo.
4. **Configuración del Toggle:** Selecciona `TogglePlayAll`. `Interactable` (marcado), `Is On` (desmarcado por defecto).
5. **Evento OnValueChanged:** Lo conectaremos en la sección 10.





9. Creando el Script Controlador (MusicPlayerController.cs) - Paso a Paso

Ahora crearemos el cerebro de nuestro reproductor. En lugar de copiar todo el código de golpe, lo construiremos paso a paso para entender mejor cada parte.

1. En la ventana `Project`, ve a tu carpeta `Scripts`. Haz clic derecho -> `Create` -> `C# Script`. Nómbralo `MusicPlayerController`.
2. Haz doble clic en el script para abrirlo en tu editor de código (como Visual Studio). Borra el contenido por defecto (`Start` y `Update`) para empezar desde cero, dejando solo la definición de la clase vacía.

9.1 Estructura Básica, Interfaces y Namespaces (`using`)

Primero, necesitamos importar las herramientas (namespaces) que Unity nos da para trabajar con UI, TextMeshPro y eventos. También definiremos la clase e indicaremos que usaremos

interfaces para detectar cuándo el usuario arrastra el slider de progreso.

Añade el siguiente código al inicio de tu archivo `MusicPlayerController.cs`:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI; // Necesario para Button, Slider, Toggle, etc.
using TMPro; // Necesario para TextMeshProUGUI
using UnityEngine.EventSystems; // Necesario para las interfaces de Drag and Drop
(Arrastrar y Soltar)

// Añadimos las interfaces después de MonoBehaviour, separadas por comas.
// Esto le dice a Unity que nuestra clase SABE CÓMO manejar estos eventos.
public class MusicPlayerController : MonoBehaviour, IBeginDragHandler, IDragHandler,
IEndDragHandler
{
    // --- Aquí dentro irá el resto de nuestro código ---

} // Fin de la clase MusicPlayerController
```

Explicación:

- `using ...;`: Importa funcionalidades predefinidas de Unity y C#.
- `public class MusicPlayerController : MonoBehaviour`: Define nuestra clase principal que hereda de `MonoBehaviour` (lo que permite adjuntarla a GameObjects en Unity).
- `, IBeginDragHandler, IDragHandler, IEndDragHandler`: Declara que esta clase implementará los métodos necesarios para responder cuando el usuario empiece a arrastrar (`OnBeginDrag`), continúe arrastrando (`OnDrag`) y suelte (`OnEndDrag`) un elemento UI (en nuestro caso, el `SliderProgress`).

9.2 Variables para Audio

Necesitamos variables para guardar las referencias a los componentes `AudioSource` (los "reproductores" de sonido) y los `AudioClip` (los archivos de sonido).

Añade estas líneas **dentro** de las llaves `{}` de la clase `MusicPlayerController`:

```
    // --- Elementos de Audio ---
    [Header("Audio Sources")] // Ayuda a organizar en el Inspector
    public AudioSource AS_Music; // Para reproducir la música principal
    public AudioSource AS_HUD;   // Para sonidos de UI (clics, etc.)

    [Header("Audio Clips")]
    public AudioClip AC_Button_OK;     // Sonido para clic genérico
    public AudioClip AC_Button_Stop;     // Sonido específico para Stop
```

```
    public AudioClip AC_Slider_Effect; // Sonido al mover slider (opcional)
    public List<AudioClip> AC_Music; // Lista para guardar todas las canciones
```

Explicación:

- `public`: Hace que estas variables sean visibles en el Inspector de Unity para que podamos arrastrar los componentes y archivos correspondientes.
- `AudioSource`: Tipo de componente que reproduce sonido.
- `AudioClip`: Tipo de archivo que contiene datos de sonido.
- `List<AudioClip>`: Una lista que puede contener múltiples `AudioClip` (nuestras canciones).
- `[Header("...")]`: Crea un título en el Inspector para mejor organización.

9.3 Variables para Elementos de UI

Ahora, variables para guardar las referencias a todos los elementos de la interfaz (botones, textos, sliders, toggle) que creamos en la escena.

Añade estas líneas **dentro** de la clase, después de las variables de audio:

```
    // --- Elementos de UI (Referencias desde el Inspector) ---
    [Header("Elementos de UI")]
    public Button ButtonPlay;
    public Button ButtonPause;
    public Button ButtonStop;
    public Button ButtonNext;
    public Button ButtonPrevious;

    public TextMeshProUGUI TextActualSong;      // Texto para el nombre de la canción
    public TextMeshProUGUI TextProgressPercent; // Texto para el tiempo (0:00 / 3:15)
    public TextMeshProUGUI TextVolume;          // Opcional: para mostrar valor numérico
    public TextMeshProUGUI TextPitch;           // Opcional: para mostrar valor numérico
    public TextMeshProUGUI TextTitleMultimedia; // Referencia si quieres cambiar el título

    public Toggle TogglePlayAll;        // El checkbox para repetir lista

    public Slider SliderProgress;       // Barra de progreso de la canción
    public Slider SliderVolume;         // Slider de volumen
    public Slider SliderPitch;          // Slider de pitch
```

Explicación:

- Declara variables públicas para cada tipo de componente UI (`Button`, `TextMeshProUGUI`, `Toggle`, `Slider`) para poder conectarlas en el Inspector.

9.4 Variables Internas

Estas variables las usará el script para llevar la cuenta del estado actual (qué canción suena, si está reproduciendo, etc.). No necesitan ser `public` porque no las configuraremos desde el Inspector.

Añade estas líneas **dentro** de la clase, después de las variables de UI:

```
    // --- Variables Internas ---
    private int currentSongIndex = 0; // Índice de la canción actual en la lista AC_Music
    private bool isPlaying = false;   // ¿Está la música sonando ahora mismo?
    private bool playAll = false;     // ¿Está activado el modo "Play All" (loop de
lista)?
    private bool isDraggingProgressBar = false; // ¿Está el usuario arrastrando la barra
de progreso?
```

Explicación:

- `private`: Estas variables solo son accesibles desde dentro de este script.
- `currentSongIndex`: Guarda el número (posición) de la canción que está seleccionada o sonando.
- `isPlaying`: Un interruptor (verdadero/falso) para saber si `AS_Music.Play()` está activo.
- `playAll`: Guarda el estado del `TogglePlayAll`.
- `isDraggingProgressBar`: Importante para evitar que la barra se actualice automáticamente mientras el usuario la está moviendo manualmente.

9.5 Método `Start()` - Configuración Inicial

El método `Start()` se ejecuta automáticamente una sola vez cuando el juego (o la escena) empieza. Lo usamos para configurar el estado inicial del reproductor.

Añade este método completo **dentro** de la clase:

```
    // --- Métodos de Unity ---
    void Start()
    {
        // 1. Configurar la primera canción
        if (AC_Music != null && AC_Music.Count > 0) // Comprobar si hay canciones
        {
            AS_Music.clip = AC_Music[currentSongIndex]; // Cargar el primer clip (índice
0) en el AudioSource
            UpdateSongInfo(); // Actualizar el texto con nombre/número de canción (lo
crearemos luego)
```

```
        }
        else
        {
            Debug.LogError("MusicPlayerController: ¡No hay canciones asignadas en la lista AC_Music!");
        }

        // 2. Sincronizar la UI con el estado inicial
        UpdateButtonStates(); // Poner los botones Play/Pause/Stop en estado correcto (lo crearemos luego)
        SliderVolume.value = AS_Music.volume; // Ajustar slider volumen al valor actual del AudioSource
        SliderPitch.value = AS_Music.pitch;   // Ajustar slider pitch al valor actual del AudioSource
        TogglePlayAll.isOn = playAll;       // Ajustar el toggle al valor inicial de playAll (false)

        // 3. Conectar funciones a eventos de Sliders y Toggle POR CÓDIGO
        //    (Alternativa/complemento a hacerlo en el Inspector)
        SliderVolume.onValueChanged.AddListener(OnSliderVolumeChanged); // Llama a OnSliderVolumeChanged cuando cambie
        SliderPitch.onValueChanged.AddListener(OnSliderPitchChanged);   // Llama a OnSliderPitchChanged cuando cambie
        TogglePlayAll.onValueChanged.AddListener(OnTogglePlayAllChanged); // Llama a OnTogglePlayAllChanged cuando cambie

        // 4. Asegurarse que el texto de progreso empiece en 00:00 / Duración
         UpdateProgressUI(0, AS_Music.clip != null ? AS_Music.clip.length : 0); // Actualiza texto progreso (lo crearemos luego)
    }
```

Explicación:

- Carga la primera canción de la lista en el `AS_Music`.
- Llama a funciones (que definiremos más tarde) para poner la UI (textos, botones) en su estado inicial.
- Ajusta los sliders de volumen/pitch y el toggle a sus valores por defecto.
- `AddListener`: Es una forma de decir "cuando este evento ocurra (ej: el valor del slider cambie), ejecuta esta función".
- Muestra un error en la consola si no hemos añadido canciones a la lista.

9.6 Método `Update()` - Lógica Continua

El método `Update()` se ejecuta en cada fotograma del juego. Lo usaremos principalmente para actualizar la barra de progreso mientras la música suena y para detectar cuándo una canción termina.

Añade este método completo **dentro** de la clase:

```
    void Update()
    {
        // Solo actualizamos si la música está sonando Y el usuario NO está arrastrando la barra
        if (isPlaying && !isDraggingProgressBar && AS_Music.clip != null && AS_Music.clip.length > 0)
        {
            // Calcular progreso (tiempo actual / duración total) -> valor entre 0.0 y 1.0
            float progress = AS_Music.time / AS_Music.clip.length;
            SliderProgress.value = progress; // Actualizar la posición del slider
            UpdateProgressUI(AS_Music.time, AS_Music.clip.length); // Actualizar el texto MM:SS / MM:SS

            // Detectar fin de canción para pasar a la siguiente (si aplica)
            // Comprobamos si el tiempo actual está muy cerca del final
            if (!AS_Music.loop && AS_Music.time >= AS_Music.clip.length - 0.1f)
            {
                if (playAll) // Si está activado el repetir lista...
                {
                    _ButtonNextSong(); // ...pasamos a la siguiente automáticamente
                }
                else // Si no...
                {
                    _ButtonStop(); // ...simplemente paramos la reproducción
                }
            }
        }
         // Si no está sonando y no arrastramos, asegurar que el texto marca 0 al inicio
         else if (!isPlaying && !isDraggingProgressBar)
        {
             if (AS_Music.time == 0) UpdateProgressUI(0, AS_Music.clip != null ? AS_Music.clip.length : 0);
        }
    }
```

Explicación:

- Comprueba si `isPlaying` es verdadero y `isDraggingProgressBar` es falso.
- Calcula el progreso dividiendo el tiempo actual (`AS_Music.time`) por la duración total (`AS_Music.clip.length`).
- Actualiza el valor del `SliderProgress` y llama a `UpdateProgressUI` (que haremos luego).
- Comprueba si la canción ha llegado casi al final (`AS_Music.time >= ...`).
- Si `playAll` está activo, llama a `_ButtonNextSong()` (que haremos luego). Si no, llama a `_ButtonStop()`.

9.7 Métodos para Botones

Ahora crearemos las funciones públicas que se ejecutarán cuando hagamos clic en cada botón. Las llamamos desde el evento `OnClick` que configuraremos en el Inspector.

Añade estos métodos **dentro** de la clase:

```
    // --- Funciones para los Botones (Llamadas desde el Inspector) ---
    public void _ButtonPlay()
    {
        if (AC_Music == null || AC_Music.Count == 0) return; // No hacer nada si no hay canciones

        AS_Music.Play();          // Inicia o reanuda la reproducción
        isPlaying = true;         // Marcar que está sonando
        UpdateButtonStates();     // Actualizar estado botones Play/Pause/Stop
        PlaySound(AC_Button_OK); // Reproducir sonido de clic (función auxiliar que haremos)
    }

    public void _ButtonPause()
    {
        AS_Music.Pause();         // Pausa la reproducción
        isPlaying = false;        // Marcar que NO está sonando
        UpdateButtonStates();     // Actualizar botones
        PlaySound(AC_Button_OK); // Sonido de clic
    }

    public void _ButtonStop()
    {
        AS_Music.Stop();          // Detiene la reproducción
        isPlaying = false;        // Marcar que NO está sonando
        AS_Music.time = 0;         // Rebobinar al principio (tiempo = 0)
        SliderProgress.value = 0; // Poner el slider a 0
        UpdateProgressUI(0, AS_Music.clip != null ? AS_Music.clip.length : 0); // Actualizar texto a 00:00
        UpdateButtonStates();     // Actualizar botones
        PlaySound(AC_Button_Stop); // Sonido específico de Stop
    }

    public void _ButtonNextSong()
    {
        if (AC_Music == null || AC_Music.Count <= 1) return; // No hacer nada si no hay canciones o solo 1

        currentSongIndex = (currentSongIndex + 1) % AC_Music.Count; // Avanza índice y vuelve a 0 si llega al final
        ChangeSong(); // Llama a la función que cambia la canción (la haremos luego)
        PlaySound(AC_Button_OK);
    }

    public void _ButtonPreviousSong()
    {
         if (AC_Music == null || AC_Music.Count <= 1) return; // No hacer nada si no hay
```

```
canciones o solo 1

        currentSongIndex--; // Retrocede el índice
        if (currentSongIndex < 0) // Si es menor que 0...
        {
            currentSongIndex = AC_Music.Count - 1; // ...va a la última canción de la
lista
        }
        ChangeSong(); // Llama a la función que cambia la canción
         PlaySound(AC_Button_OK);
    }
```

Explicación:

- Cada función corresponde a un botón y realiza la acción esperada (`Play`, `Pause`, `Stop`).
- Actualizan la variable `isPlaying` y llaman a `UpdateButtonStates()` para reflejar el cambio en la UI.
- `Stop` además rebobina la canción (`AS_Music.time = 0`).
- `Next` y `Previous` calculan el nuevo `currentSongIndex`. El `% AC_Music.Count` (módulo) en `Next` asegura que el índice vuelva a 0 después de la última canción, creando un ciclo.
- Llaman a funciones auxiliares `PlaySound` y `ChangeSong` (que crearemos pronto).

9.8 Métodos para Sliders y Toggle

Estas funciones se ejecutarán cuando el usuario mueva los sliders de Volumen/Pitch o cambie el estado del Toggle "Play All". Las conectamos en `Start()` usando `AddListener`, o también se pueden conectar en el Inspector.

Añade estos métodos **dentro** de la clase:

```

    // --- Funciones para Sliders y Toggle (Llamadas desde Listener o Inspector) ---

    // Se ejecuta cuando el valor del SliderVolume cambia
    public void OnSliderVolumeChanged(float value)
    {
        AS_Music.volume = value; // Actualiza el volumen del AudioSource
        // Opcional: Actualizar texto si tienes uno para mostrar el valor
        if (TextVolume != null) TextVolume.text = $"VOLUME: {value:P0}"; // P0 formatea
como porcentaje sin decimales
        // PlaySound(AC_Slider_Effect); // Podrías poner un sonido aquí si quieres
    }

    // Se ejecuta cuando el valor del SliderPitch cambia
    public void OnSliderPitchChanged(float value)
    {
        AS_Music.pitch = value; // Actualiza el pitch (velocidad/tono) del AudioSource
```

```
        // Opcional: Actualizar texto
        if (TextPitch != null) TextPitch.text = $"PITCH: {value:F2}x"; // F2 formatea
como número con 2 decimales
        // PlaySound(AC_Slider_Effect);
    }

    // Se ejecuta cuando el valor del SliderProgress cambia INTERACTIVAMENTE
    // (Nota: Esta se conecta al OnValueChanged del SliderProgress en el Inspector)
    public void OnSliderProgressValueChanged(float value)
    {
        // Solo actualizamos el TEXTO si el usuario está arrastrando
        if (isDraggingProgressBar && AS_Music.clip != null)
        {
            float targetTime = value * AS_Music.clip.length; // Calcula a qué tiempo
corresponde el valor del slider
            UpdateProgressUI(targetTime, AS_Music.clip.length); // Actualiza el texto
MM:SS
            // ¡OJO! No cambiamos AS_Music.time aquí, eso lo hacemos SOLO cuando suelta
el ratón (OnEndDrag)
        }
    }

    // Se ejecuta cuando el estado del TogglePlayAll cambia
    public void OnTogglePlayAllChanged(bool isOn) // Recibe el nuevo estado (true si está
marcado)
    {
        playAll = isOn; // Actualiza nuestra variable interna
        // OJO: AS_Music.loop = playAll; solo funcionaría bien si quisiéramos repetir LA
MISMA canción.
        // Para repetir la lista, gestionamos el cambio de canción en Update() cuando
termina.
        PlaySound(AC_Button_OK);
    }
```

Explicación:

- Reciben el nuevo valor del control como parámetro (`float value` o `bool isOn`).
- Actualizan la propiedad correspondiente del `AS_Music` (`volume`, `pitch`) o la variable interna (`playAll`).
- `OnSliderProgressValueChanged` es un poco especial: solo actualiza el texto *mientras* se arrastra, pero no cambia el tiempo real de la canción hasta que se suelta el ratón.
- Opcionalmente, actualizan los textos `TextVolume` y `TextPitch` para mostrar el valor numérico.

9.9 Métodos para Arrastrar el Slider de Progreso (Interfaces)

Estos son los métodos que necesitamos implementar por haber añadido `IBeginDragHandler`, `IDragHandler` y `IEndDragHandler` a la definición de la clase. Se llamarán automáticamente

cuando el usuario interactúe con el `SliderProgress` (siempre que hayamos añadido un componente `Event Trigger` en el Inspector, como veremos en la sección 10).

Añade estos métodos **dentro** de la clase:

```
    // --- Gestión Dragging Slider Progreso (Implementación Interfaces) ---

    // Se llama UNA VEZ cuando el usuario EMPIEZA a arrastrar el objeto
    public void OnBeginDrag(PointerEventData eventData)
    {
        // Comprobamos si el objeto que se empezó a arrastrar es nuestro SliderProgress
        if (eventData.pointerDrag == SliderProgress.gameObject)
        {
            isDraggingProgressBar = true; // Marcamos que estamos arrastrando
            // PlaySound(AC_Slider_Effect); // Sonido opcional al empezar
        }
    }

    // Se llama CONTINUAMENTE mientras el usuario MANTIENE presionado y MUEVE el ratón
    public void OnDrag(PointerEventData eventData)
    {
        // El propio Slider ya llama a OnSliderProgressValueChanged mientras arrastramos,
        // así que ahí actualizamos la UI visualmente. Aquí podríamos hacer otras cosas
si quisiéramos.
        // Nos aseguramos de que el flag sigue activo por si acaso.
        if (!isDraggingProgressBar && eventData.pointerDrag == SliderProgress.gameObject)
        {
            isDraggingProgressBar = true;
        }
    }

    // Se llama UNA VEZ cuando el usuario SUELTA el botón del ratón
    public void OnEndDrag(PointerEventData eventData)
    {
        // Comprobamos si estábamos arrastrando y si soltamos sobre el SliderProgress
        if (isDraggingProgressBar && eventData.pointerPress == SliderProgress.gameObject)
        {
            isDraggingProgressBar = false; // Marcamos que ya NO estamos arrastrando
            if (AS_Music.clip != null)
            {
                // ¡AHORA SÍ! Aplicamos el cambio de tiempo en la reproducción
                AS_Music.time = SliderProgress.value * AS_Music.clip.length;
                // Actualizamos el texto MM:SS por última vez tras soltar
                UpdateProgressUI(AS_Music.time, AS_Music.clip.length);
            }
            // PlaySound(AC_Button_OK); // Sonido opcional al soltar
        }
    }
```

Explicación:

- `OnBeginDrag`: Pone `isDraggingProgressBar` a `true` para que `Update()` deje de mover el slider automáticamente.
- `OnDrag`: No hace mucho aquí porque `OnSliderProgressValueChanged` ya se encarga de actualizar el texto mientras se arrastra.
- `OnEndDrag`: Pone `isDraggingProgressBar` a `false`. Lo más importante: calcula el tiempo (`AS_Music.time`) correspondiente a la posición final del slider (`SliderProgress.value`) y lo aplica, haciendo que la música salte a ese punto.
- `PointerEventData eventData`: Contiene información sobre el evento del ratón/puntero. Lo usamos para asegurarnos de que el evento ocurrió sobre nuestro slider.

9.10 Métodos Auxiliares (Helpers)

Finalmente, creamos unas funciones privadas que nos ayudan a organizar el código y evitar repeticiones. Estas funciones son llamadas desde otros métodos que ya hemos creado.

Añade estos métodos **dentro** de la clase:

```
    // --- Métodos Auxiliares ---

    // Cambia la canción actual en el AudioSource
    void ChangeSong()
    {
        AS_Music.Stop(); // Para la canción anterior si estaba sonando
        AS_Music.clip = AC_Music[currentSongIndex]; // Carga el nuevo AudioClip
        UpdateSongInfo(); // Actualiza el texto del nombre de la canción
        SliderProgress.value = 0; // Resetea el slider de progreso
        UpdateProgressUI(0, AS_Music.clip != null ? AS_Music.clip.length : 0); // Resetea
el texto de progreso

        if (isPlaying) // Si el reproductor estaba en modo "Play" antes del cambio...
        {
            AS_Music.Play(); // ...inicia la nueva canción automáticamente
        }
         // Si no estaba en Play, no hacemos nada y se queda lista para darle al Play
manualmente.
         // En ambos casos, los botones se actualizarán porque Play/Stop llaman a
UpdateButtonStates.
         // Si quisiéramos asegurar actualización si NO estaba en Play: else {
UpdateButtonStates(); }
    }

    // Actualiza el texto que muestra el nombre de la canción y su número
    void UpdateSongInfo()
    {
        if (TextActualSong != null && AS_Music.clip != null && AC_Music.Count > 0)
        {
            // Usamos string interpolation ($"...") para construir el texto
```

```
            TextActualSong.text = $"Playing: {AS_Music.clip.name} ({currentSongIndex +
1}/{AC_Music.Count})";
        }
         else if (TextActualSong != null)
        {
             TextActualSong.text = "No songs loaded"; // Mensaje si no hay canciones
        }
    }

    // Habilita/deshabilita los botones Play/Pause/Stop según el estado
    void UpdateButtonStates()
    {
        // Si no tenemos referencias a los botones, no hacer nada
        if (ButtonPlay == null || ButtonPause == null || ButtonStop == null) return;

        // El botón Play SÓLO debe estar activo si NO está sonando
        ButtonPlay.interactable = !isPlaying;
        // El botón Pause SÓLO debe estar activo si SÍ está sonando
        ButtonPause.interactable = isPlaying;
        // El botón Stop debe estar activo si está sonando O si está pausado pero no al
inicio (tiempo > 0)
        ButtonStop.interactable = isPlaying || AS_Music.time > 0.01f; // Pequeño margen
por si acaso

        // (Opcional) Habilitar/deshabilitar Next/Prev si solo hay una canción
        bool canChangeTrack = AC_Music != null && AC_Music.Count > 1;
        if(ButtonNext != null) ButtonNext.interactable = canChangeTrack;
        if(ButtonPrevious != null) ButtonPrevious.interactable = canChangeTrack;
    }

     // Formatea y actualiza el texto del progreso (ej: "01:23 / 04:50")
    void UpdateProgressUI(float currentTime, float totalTime)
    {
        if (TextProgressPercent != null)
        {
            // TimeSpan es útil para convertir segundos a formato MM:SS
            System.TimeSpan current = System.TimeSpan.FromSeconds(currentTime);
            System.TimeSpan total = System.TimeSpan.FromSeconds(totalTime);

            // Formato "mm\:ss" asegura dos dígitos para minutos y segundos, con ':'
literal
            TextProgressPercent.text = $"{current:mm\\:ss} / {total:mm\\:ss}";
        }
         // Nota: La actualización del VALOR del slider se hace en Update() o OnEndDrag()
    }

    // Reproduce un sonido corto usando el AudioSource del HUD
    void PlaySound(AudioClip clip)
    {
        if (AS_HUD != null && clip != null)
        {
            // PlayOneShot es ideal para efectos de sonido, no interrumpe otros sonidos en
el mismo source
            AS_HUD.PlayOneShot(clip);
        }
```

```
    }
// ¡Asegúrate de que esta llave cierra la clase MusicPlayerController!
}
```

Explicación:

- `ChangeSong`: Contiene la lógica para detener la canción actual, cargar la nueva, actualizar la UI relacionada y opcionalmente empezar a reproducir la nueva.
- `UpdateSongInfo`: Actualiza el `TextActualSong` con el nombre del clip y el número de canción.
- `UpdateButtonStates`: Activa/desactiva los botones Play, Pause y Stop según tenga sentido (no puedes pausar si no está sonando, etc.). Usa la propiedad `interactable` de los botones.
- `UpdateProgressUI`: Toma los tiempos en segundos y los convierte a formato "minutos:segundos" para mostrarlos en `TextProgressPercent`.
- `PlaySound`: Una función simple para reproducir los efectos de sonido de la UI usando el `AS_HUD` y `PlayOneShot`.

¡Felicidades! Si has seguido todos los pasos, ahora tienes el script `MusicPlayerController.cs` completo y, lo más importante, has visto cómo se construye cada parte y para qué sirve. Guarda el archivo.

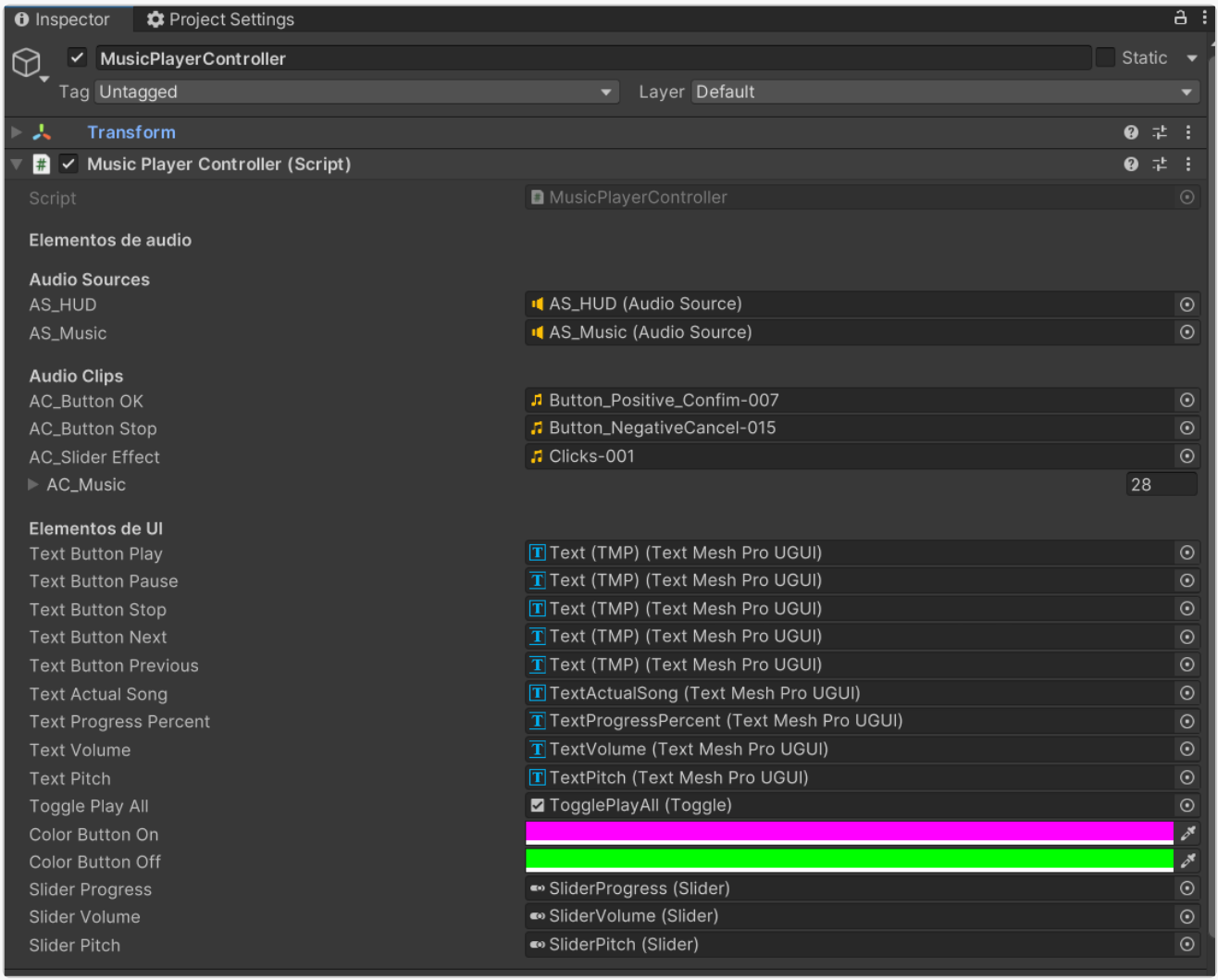
El siguiente paso es volver a Unity, asignar este script al GameObject `MusicPlayerController` y conectar todas las referencias y eventos en el Inspector, como se describe en la sección 10.

10. Conectando la UI al Script

Ahora es el momento de hacer que la UI interactúe con el script que acabamos de construir.

1. **Asignar Script:** Selecciona el GameObject vacío `MusicPlayerController` en la Jerarquía. Arrastra el script `MusicPlayerController.cs` desde la ventana Project hasta el Inspector de este GameObject.
2. **Conectar Referencias:** Verás todas las variables públicas del script en el Inspector (organizadas por los `[Header]`). Arrastra cada GameObject de la UI desde la Jerarquía al campo correspondiente en el script:
   - Arrastra `ButtonPlay` (el GameObject) al campo `Button Play` del script.
   - Arrastra `ButtonPause` al campo `Button Pause`.
   - ...y así sucesivamente para `ButtonStop`, `ButtonNext`, `ButtonPrevious`.

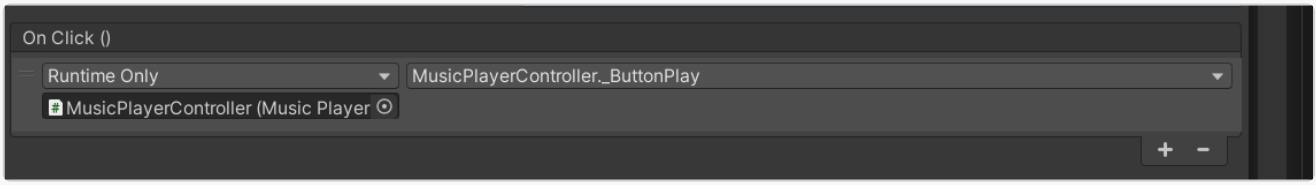
- Arrastra `TextActualSong` (el GameObject con TextMeshPro) al campo `Text Actual Song`.
- ...y así sucesivamente para `TextProgressPercent`, `TextVolume`, `TextPitch`, `TextTitleMultimedia`.
- Arrastra `TogglePlayAll` (el GameObject) al campo `Toggle Play All`.
- Arrastra `SliderProgress` (el GameObject) al campo `Slider Progress`.
- ...y así sucesivamente para `SliderVolume` y `SliderPitch`.
- *(Dejaremos las referencias de Audio para la siguiente sección).*



Asegúrate de que TODAS las referencias de UI estén asignadas. Si falta alguna, tendrás errores de "Null Reference Exception" al ejecutar.

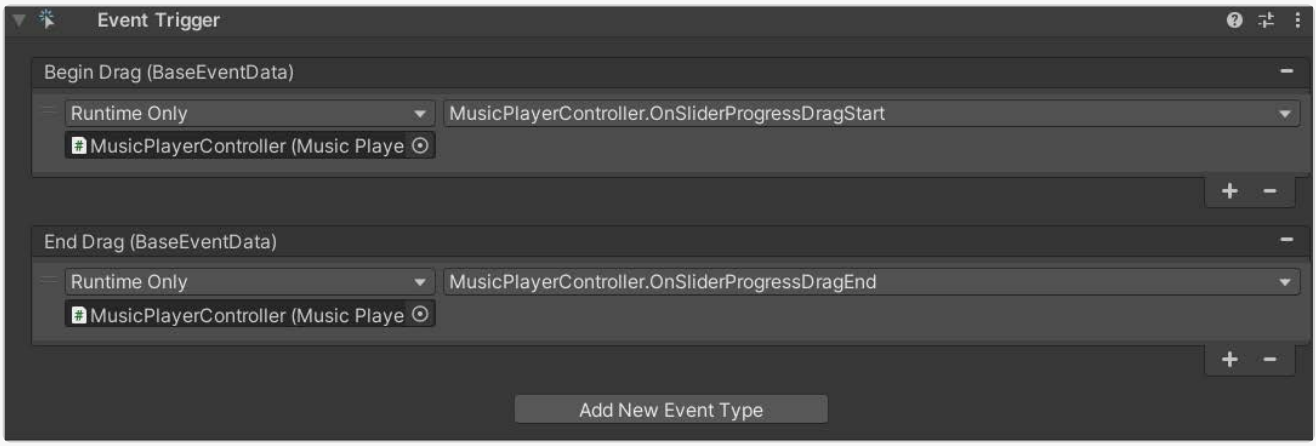
3. **Configurar Eventos OnClick de Botones:**
   - Selecciona `ButtonPlay` en la Jerarquía.
   - En el Inspector, busca el componente `Button` y la sección `On Click ()`.
   - Haz clic en el botón `+`.

- Arrastra el GameObject `MusicPlayerController` desde la Jerarquía al campo que dice "None (Object)".
- En el desplegable "No Function", selecciona `MusicPlayerController` -> `_ButtonPlay()`.
- Repite este proceso para los demás botones, conectándolos a sus funciones correspondientes:
  - `ButtonPause` -> `_ButtonPause()`
  - `ButtonStop` -> `_ButtonStop()`
  - `ButtonNext` -> `_ButtonNextSong()`
  - `ButtonPrevious` -> `_ButtonPreviousSong()`



4. **Configurar Eventos de Sliders y Toggle (Método Inspector):** Aunque añadimos listeners en `Start()` para Volumen, Pitch y Toggle, también podemos (o debemos, en el caso de Progress) hacerlo aquí.
   - Selecciona `SliderVolume`. En `On Value Changed (Single)`, haz clic `+`, arrastra `MusicPlayerController`, y selecciona `MusicPlayerController` -> `OnSliderVolumeChanged (dynamic float)`.
   - Haz lo mismo para `SliderPitch` -> `OnSliderPitchChanged (dynamic float)`.
   - Selecciona `TogglePlayAll`. En `On Value Changed (Boolean)`, haz clic `+`, arrastra `MusicPlayerController`, y selecciona `MusicPlayerController` -> `OnTogglePlayAllChanged (dynamic bool)`.
   - Selecciona `SliderProgress`. En `On Value Changed (Single)`, haz clic `+`, arrastra `MusicPlayerController`, y selecciona `MusicPlayerController` -> `OnSliderProgressValueChanged (dynamic float)`.
5. **Configurar Eventos Drag del SliderProgress (Event Trigger):** Esto es crucial para que la detección de arrastre funcione.
   - Selecciona `SliderProgress`.
   - Haz clic en "Add Component" y busca y añade "Event Trigger".
   - Haz clic en "Add New Event Type" y selecciona `BeginDrag`. En la nueva entrada, haz clic `+`, arrastra `MusicPlayerController` y selecciona `MusicPlayerController` -> `OnBeginDrag`.
   - Haz clic "Add New Event Type" -> `Drag`. Haz clic `+`, arrastra `MusicPlayerController` -> `OnDrag`.
   - Haz clic "Add New Event Type" -> `EndDrag`. Haz clic `+`, arrastra

`MusicPlayerController` -> `OnEndDrag`.

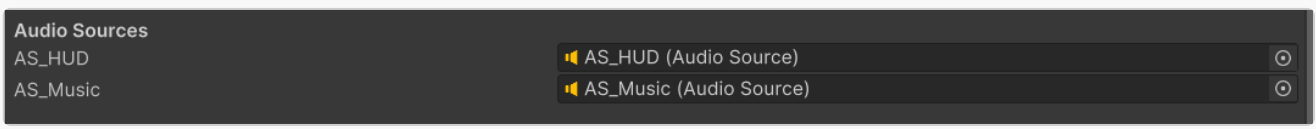


¡Ahora la interfaz gráfica sabe qué funciones llamar en nuestro script cuando interactuamos con ella!

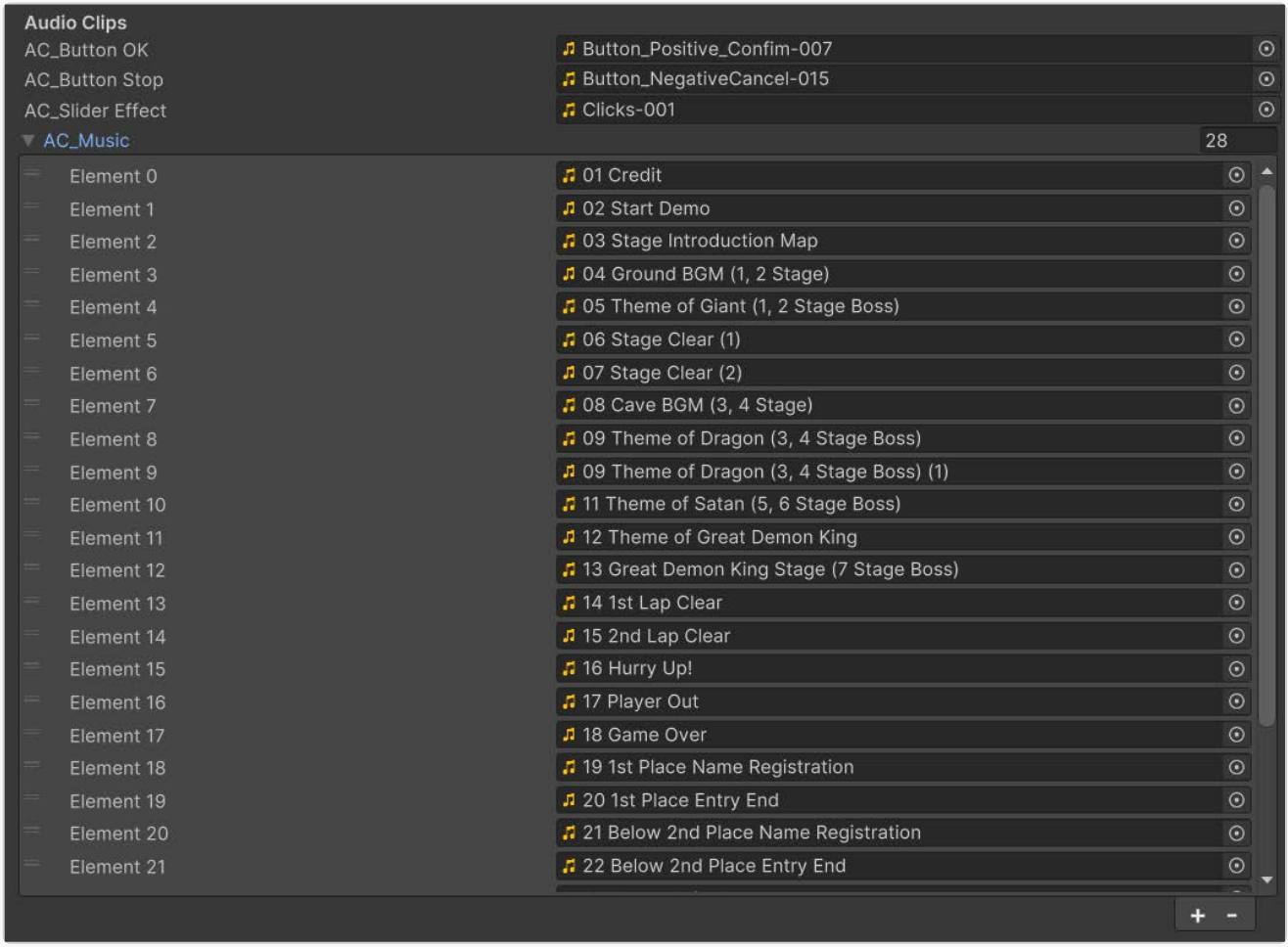
11. Configurando el Audio

Necesitamos configurar los AudioSource y asignar los clips de audio a nuestro script.

1. **Crear AudioSources:**
   - Selecciona el GameObject `MusicPlayerController` en la Jerarquía.
   - En el Inspector, haz clic en "Add Component" -> "Audio Source".
   - En este primer Audio Source: desmarca `Play On Awake`. Ajusta `Spatial Blend` a `2D`. Este será `AS_Music`.
   - Añade un SEGUNDO componente "Audio Source" al mismo GameObject.
   - En este segundo Audio Source: desmarca `Play On Awake`. Ajusta `Spatial Blend` a `2D`. Este será `AS_HUD`.
2. **Asignar AudioSources al Script:**
   - Con `MusicPlayerController` seleccionado, busca los campos `AS_Music` y `AS_HUD` en el script dentro del Inspector.
   - Arrastra el primer componente Audio Source que añadiste (el de música) al campo `AS_Music`.
   - Arrastra el segundo componente Audio Source (el de HUD) al campo `AS_HUD`.



3. **Asignar AudioClips al Script:**
    - Busca la sección "Audio Clips" en el script (Inspector).
    - Arrastra tus clips de sonido de botones desde la ventana Project a los campos `AC Button OK`, `AC Button Stop`, `AC Slider Effect` (si tienes uno).
    - Busca la lista `AC Music`. Cambia su `Size` al número de canciones que tengas (ej: si tienes 3 canciones, pon `Size` a 3).
    - Arrastra cada uno de tus archivos de música desde la ventana Project a los campos "Element 0", "Element 1", "Element 2", etc., que han aparecido en la lista `AC Music`.



4. **Configuración Adicional (Opcional):**
    - Puedes ajustar el `Volume` inicial de `AS_Music` y `AS_HUD` en sus respectivos componentes Audio Source (aunque los sliders los controlarán después).
    - Asegúrate de que `Loop` esté desmarcado en `AS_Music`.

12. ¡Prueba y Finalización!

1. **¡Ejecuta la Escena!** Haz clic en el botón Play de Unity.

2. **Prueba todas las funcionalidades:**
    - ¿Funciona Play? ¿Se actualiza el texto de la canción y el progreso?
    - ¿Funciona Pause? ¿Y Stop?
    - ¿Funcionan Next y Previous?
    - ¿Puedes ajustar Volumen y Pitch con los sliders?
    - ¿Puedes arrastrar el slider de Progreso para buscar en la canción?
    - ¿Funciona el toggle "Play All" (pasa a la siguiente canción automáticamente al terminar)?
    - ¿Se escuchan los efectos de sonido de los botones?
3. **Debugging:** Si algo no funciona, revisa la Consola de Unity (`Window` -> `General` -> `Console`) en busca de errores (rojos). Los más comunes:
    - `NullReferenceException`: Falta asignar una referencia en el Inspector (un botón, texto, slider, AudioSource o AudioClip en el `MusicPlayerController`). Revisa TODOS los campos públicos del script.
    - Errores de compilación: Si el script tiene errores de sintaxis, la Consola lo indicará. Haz doble clic en el error.
    - Eventos no conectados: Verifica que los `OnClick` de botones y los `OnValueChanged`/`Event Triggers` de sliders/toggle estén bien configurados en el Inspector (Sección 10).
4. **Pulido Final:** Ajusta la posición, tamaño y estilo de los elementos UI hasta que estés contento/a.

¡Felicidades! Has construido un reproductor de música funcional en Unity, aprendiendo sobre UI, Audio y Scripting paso a paso.

Posibles Mejoras:

- Visualización de espectro de audio.
- Cargar listas de reproducción externas.
- Guardar/cargar preferencias (`PlayerPrefs`).
- Botón Shuffle (aleatorio).
- Animaciones y efectos visuales.

Práctica Reproductor de Sonido en Unity