Lab: Dynamic Sound Effects and Music Transitions in Unity

Introduction

In this lab, we'll create two Unity scenes that demonstrate dynamic audio implementation. The first part will focus on playing different sound effects based on collision with various surfaces, while the second part will showcase smooth transitions between music tracks.

Prerequisites

- Unity 2022.3 LTS with URP
- Basic familiarity with C# scripting in Unity
- Basic knowledge of Unity audio system.

Part I: Dynamic Surface Sound Effects

Project Setup:

1. Create a new URP 3D project in Unity.
2. Download and Import the Free `FOOTSTEPS - ESSENTIALS` sound effects pack from the Unity Asset Store into your project. https://assetstore.unity.com/packages/audio/sound-fx/foley/footsteps-essentials-189879
3. (Optional): If you do not wish to use the above SFX pack, source a variety of sound effects to use for this lab. You will need at least four (4) SFX and two (2) Music tracks.

> **Asset Selection**
>
> The Unity Asset Store has several free sound effects packs, as does various free sound effects websites.

Scene Setup:

1. In the `Hierarchy` window, `right-click > 3D Object > Plane`. Rename it to "Ground".
2. Right-click in the `Hierarchy` again, select `Create Empty`, and name it "AudioManager".
3. Right-click in the `Hierarchy` once more, select `3D Object > Sphere`, and name it "Bouncer".

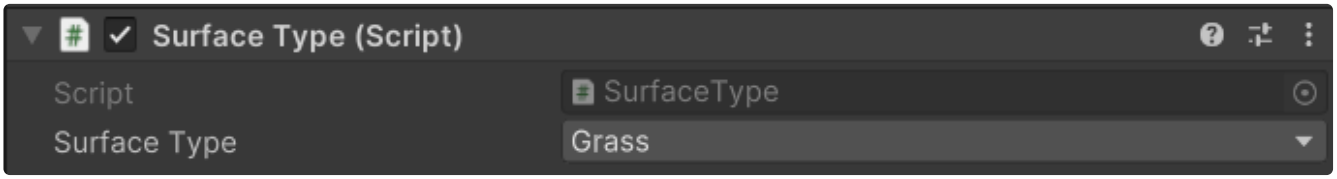
Step 1: Create the Surface Types

1. In the `Project window`, right-click and select `Create > C# Script`. Name it "SurfaceType".
2. Double-click the script to open it in your code editor.

3. Replace the contents with the following code:

```
using UnityEngine;

public enum SubstrateType
{
    Grass,
    Wood,
    Stone,
    Metal
}

public class SurfaceType : MonoBehaviour
{
    public SubstrateType surfaceType;
}
```

1. Save the script and return to Unity.
2. In the `Hierarchy`, select the Ground object.
3. In the `Inspector`, click `Add Component`.
4. Type "Surface Type" and select the script you just created.



Info

Enums are useful for creating a fixed set of options. In this case, we're using it to define our different surface types.

Step 2: Create the Audio Manager

1. In the `Project window`, right-click and select `Create > C# Script`. Name it "AudioManager".
2. Double-click the script to open it in your code editor.
3. Replace the contents with the following code:

```
using UnityEngine;

public class AudioManager : MonoBehaviour
{
    public AudioClip[] grassSounds;
```

```
    public AudioClip[] woodSounds;
    public AudioClip[] stoneSounds;
    public AudioClip[] metalSounds;

    private AudioSource audioSource;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }

    public void PlaySurfaceSound(SubstrateType surfaceType)
    {
        AudioClip[] currentSounds = null;

        switch (surfaceType)
        {
            case SubstrateType.Grass:
                currentSounds = grassSounds;
                break;
            case SubstrateType.Wood:
                currentSounds = woodSounds;
                break;
            case SubstrateType.Stone:
                currentSounds = stoneSounds;
                break;
            case SubstrateType.Metal:
                currentSounds = metalSounds;
                break;
        }

        if (currentSounds != null && currentSounds.Length > 0)
        {
            AudioClip randomSound = currentSounds[Random.Range(0,
currentSounds.Length)];
            audioSource.PlayOneShot(randomSound);
        }
    }
}
```

ⓘ Code Explainer: Switch Statement

A switch statement is a control flow mechanism used when you have multiple possible execution paths based on the value of a single variable.

How it works:

1. It evaluates an expression (in this case, `surfaceType`).
2. It compares the result with the values specified in each `case` clause.

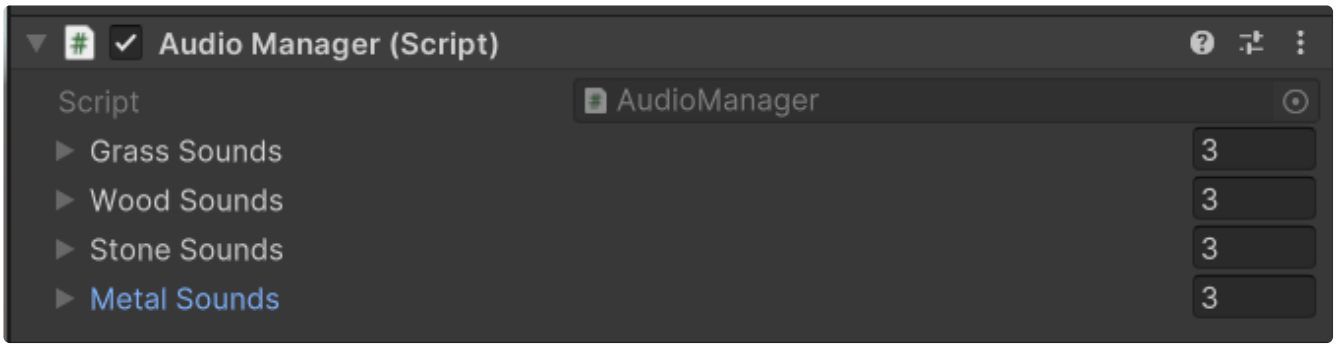
3. When a match is found, it executes the corresponding code block.
4. The `break` statement ends each case to prevent fall-through to the next case.

Use cases:

- When you have multiple conditions based on a single variable.
- As a more readable alternative to multiple if-else statements.

In this script:
We use the switch statement to select the appropriate array of sound effects based on the surface type. This allows us to easily extend our code in the future by adding new surface types without changing the overall structure of the method.

1. Save the script and return to Unity.
2. In the `Hierarchy`, select the AudioManager GameObject.
3. In the `Inspector`, click `Add Component`.
4. Type "Audio Manager" and select the script you just created.



Tip

Using arrays for each surface type allows us to have multiple sound variations, making the audio more dynamic and less repetitive.

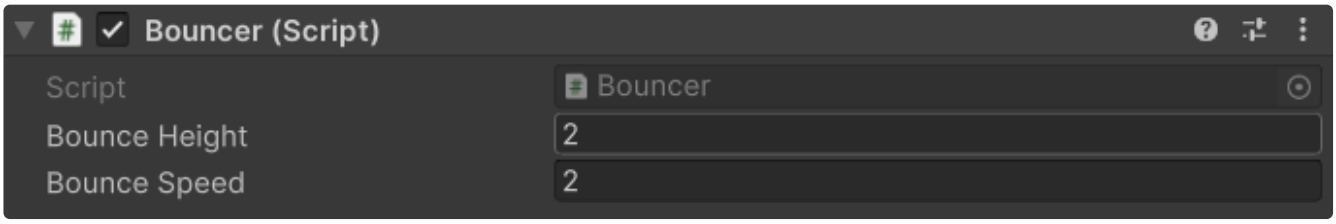
Step 3: Create the Bouncer Script

1. In the `Project window`, right-click and select `Create > C# Script`. Name it "Bouncer".
2. Double-click the script to open it in your code editor.
3. Replace the contents with the following code:

```
using UnityEngine;
```

```
public class Bouncer : MonoBehaviour
{
    public float bounceHeight = 5f;
    public float bounceSpeed = 1f;

    private Vector3 startPosition;
    private AudioManager audioManager;
    private bool hasTouchedSurface = false;

    void Start()
    {
            startPosition = transform.position;
        audioManager = FindObjectOfType<AudioManager>();
    }
    void Update()
    {
    float newY = startPosition.y + Mathf.Abs(Mathf.Sin(Time.time * bounceSpeed)) *
bounceHeight;

    transform.position = new Vector3(transform.position.x, newY,
transform.position.z);

    if (Physics.Raycast(transform.position, Vector3.down, out RaycastHit hit,
bounceHeight))
        {
                SurfaceType surfaceType = hit.collider.GetComponent<SurfaceType>();
        if (surfaceType != null)
            {
                    if (transform.position.y <= startPosition.y + 0.1f)
                {
                        if (!hasTouchedSurface)
                    {
                audioManager.PlaySurfaceSound(surfaceType.surfaceType);
                        hasTouchedSurface = true;
                    }
                }
                else
                {
                    hasTouchedSurface = false;
                }
            }
        }
    }
}
```

1. Save the script and return to Unity.
2. In the `Hierarchy`, select the Bouncer GameObject.
3. In the `Inspector`, click `Add Component`.
4. Type "Bouncer" and select the script you just created.

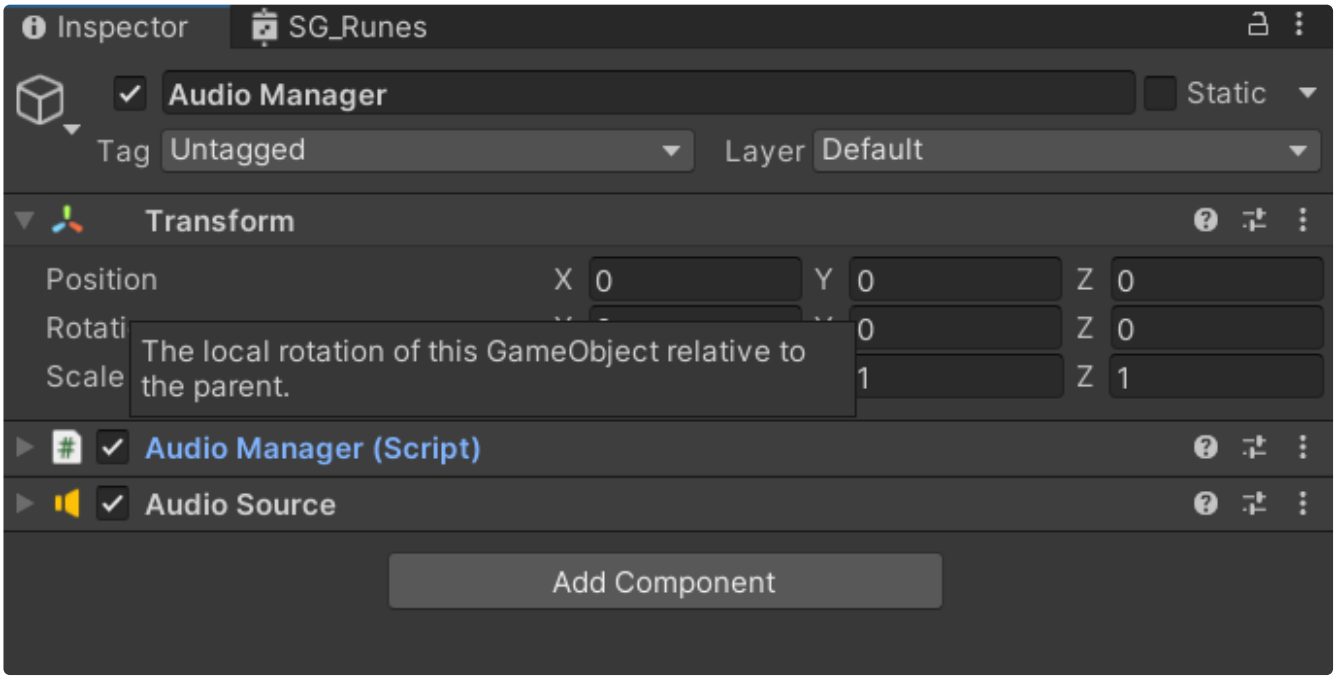


> **Info**
>
> We use a sine wave to create a smooth up-and-down motion for the bouncer. The Raycast is used to detect when the bouncer is close to the ground, triggering the sound effect.

Step 4: Set Up the Scene

4.1 Add AudioSource to AudioManager

1. In the `Hierarchy`, select the AudioManager GameObject.
2. In the `Inspector`, click `Add Component`.
3. Type "Audio Source" in the search bar and select it to add the component.

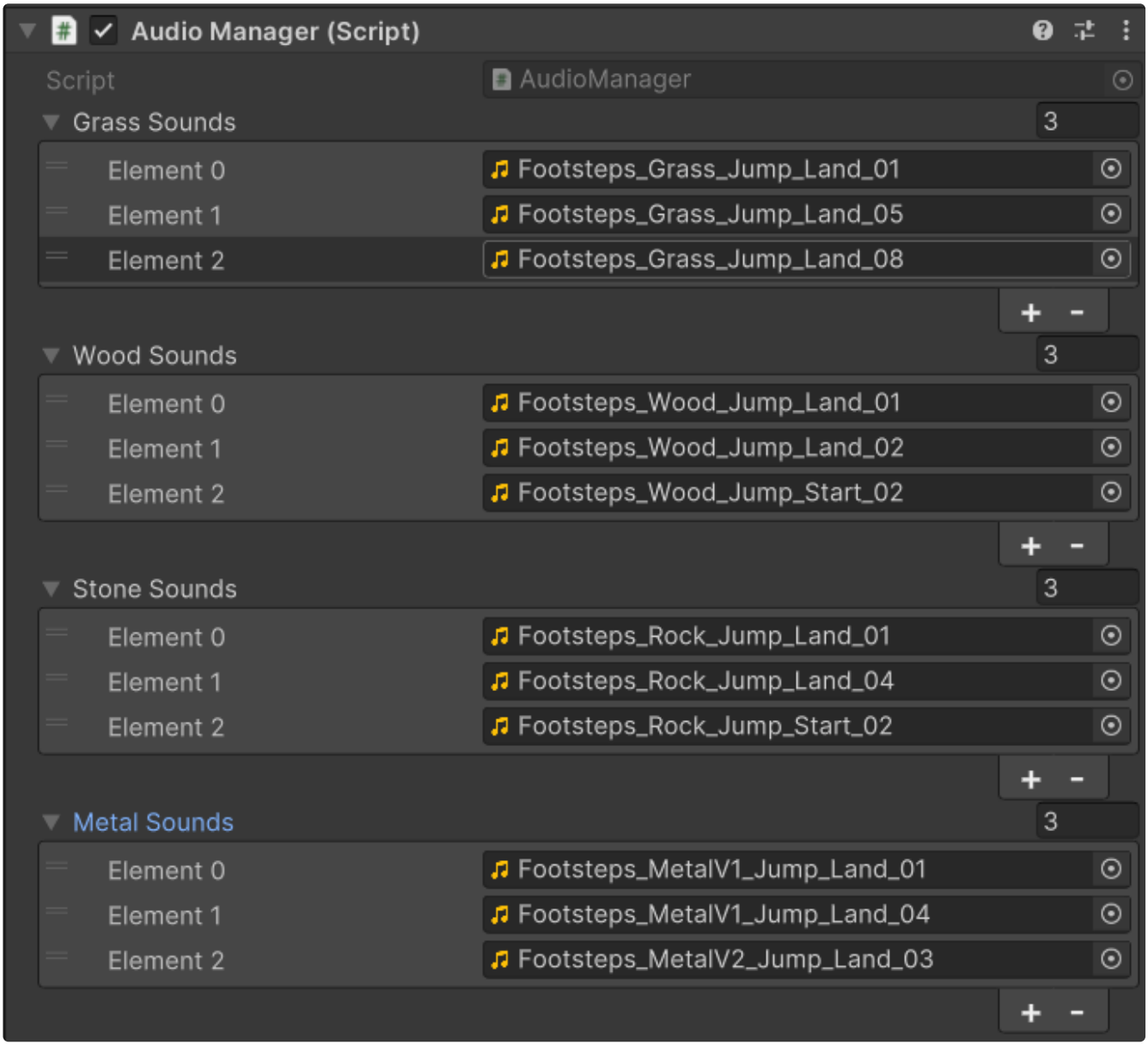


4.2 Assign Sound Effect Clips

> **Optional Step**
>
> Creating the audio folder below may be optional depending on where your sourced audio assets come from. However, it is always a good organizational practice to break assets into folders and

sub-folders.

1. In your `Project window`, right-click and select `Create > Folder`. Name it "Audio".
2. Find some free sound effects online (e.g., from Freesound.org) for grass, wood, stone, and metal impacts. Download at least 2-3 for each type. *Or use the recommended Unity Store Asset at the top of the lab*.
3. In Unity, drag and drop your audio files into the Audio folder you just created.
4. In the `Hierarchy`, select the AudioManager GameObject.
5. In the `Inspector`, find the AudioManager script component.
6. For each array (grassSounds, woodSounds, stoneSounds, metalSounds):
   a. Set the Size to the number of clips you have for that surface type.
   b. Drag and drop each audio clip from the Project window to the corresponding element in the array. Or use the small Target icon to select assets.



> **Tip**
>
> Make sure your audio clips are short and punchy for the best effect.

4.3 Adjust Bouncer Properties

1. In the `Hierarchy`, select the Bouncer GameObject.
2. In the `Inspector`, find the Bouncer script component.
3. Locate the "Bounce Height" field and set it to 2.
4. Find the "Bounce Speed" field and set it to 1.

4.4 Create Dynamic Material Script

1. In the `Project window`, right-click and select `Create > C# Script`. Name it "DynamicMaterial".
2. Double-click the script to open it in your code editor.
3. Replace the contents with the following code:

```
using UnityEngine;

public class DynamicMaterial : MonoBehaviour
{
    public Color grassColor = new Color(0.1f, 0.8f, 0.1f);
    public Color woodColor = new Color(0.6f, 0.4f, 0.2f);
    public Color stoneColor = new Color(0.5f, 0.5f, 0.5f);
    public Color metalColor = new Color(0.8f, 0.8f, 0.8f);

    private Material material;
    private SurfaceType surfaceType;

    void Start()
    {
        material = new Material(Shader.Find("Universal Render Pipeline/Lit"));
        GetComponent<Renderer>().material = material;
        surfaceType = GetComponent<SurfaceType>();
        UpdateMaterial();
    }

    public void UpdateMaterial()
    {
        switch (surfaceType.surfaceType)
        {
            case SubstrateType.Grass:
                material.color = grassColor;
                break;
            case SubstrateType.Wood:
                material.color = woodColor;
```

```
                break;
            case SubstrateType.Stone:
                material.color = stoneColor;
                break;
            case SubstrateType.Metal:
                material.color = metalColor;
                break;
        }
    }
}
```

4. Save the script and return to Unity.

4.5 Update SurfaceType Script

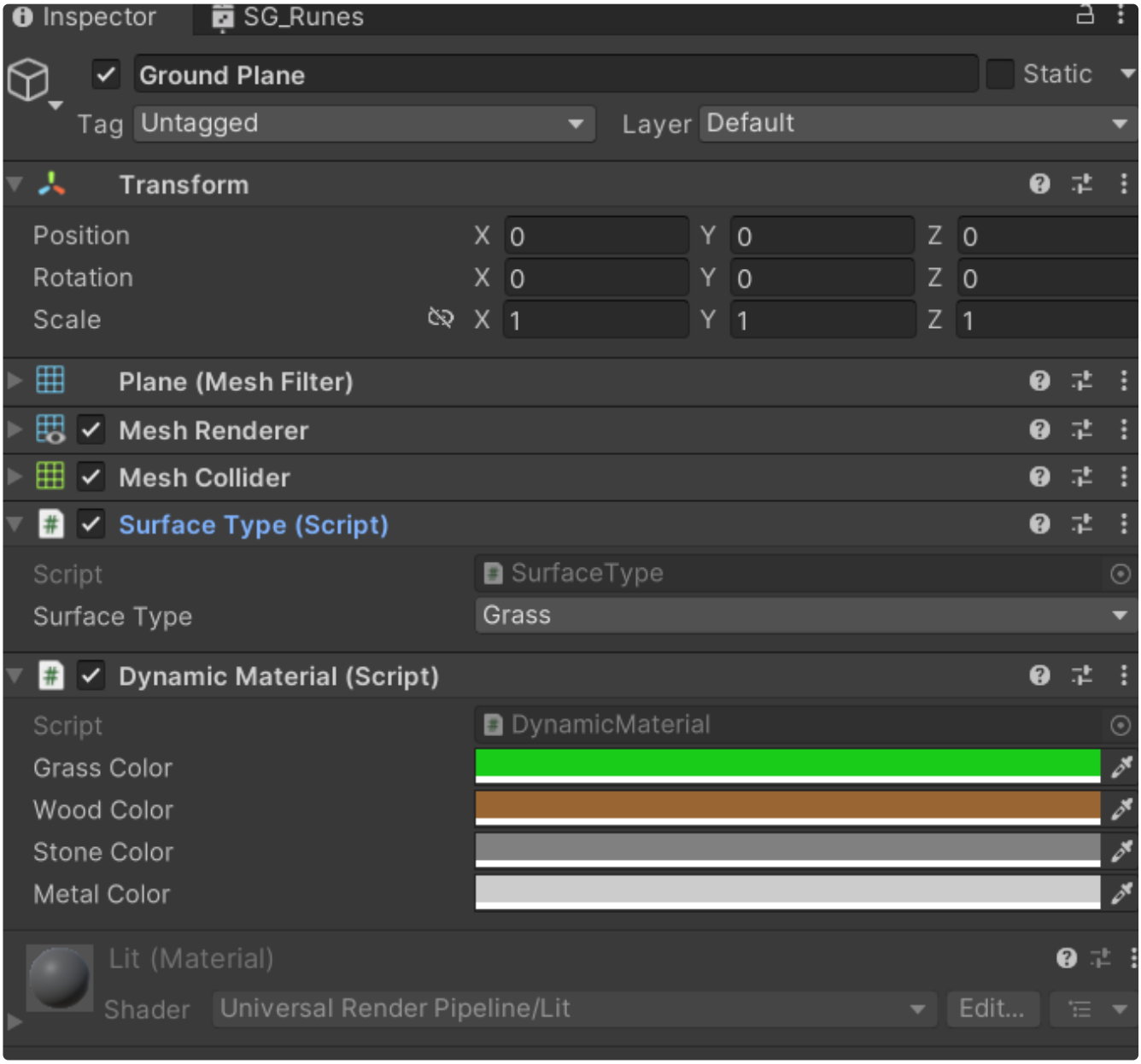
1. Open the SurfaceType script in your code editor.
2. Update the script to include a method for changing the surface type:

```
using UnityEngine;

public enum SubstrateType
{
    Grass,
    Wood,
    Stone,
    Metal
}

public class SurfaceType : MonoBehaviour
{
    public SubstrateType surfaceType;
    private DynamicMaterial dynamicMaterial;

    void Start()
    {
        dynamicMaterial = GetComponent<DynamicMaterial>();
    }

    public void ChangeSurfaceType(SubstrateType newType)
    {
        surfaceType = newType;
        if (dynamicMaterial != null)
        {
            dynamicMaterial.UpdateMaterial();
        }
    }
}
```

3. Save the script and return to Unity.

4.6 Apply Scripts to Ground

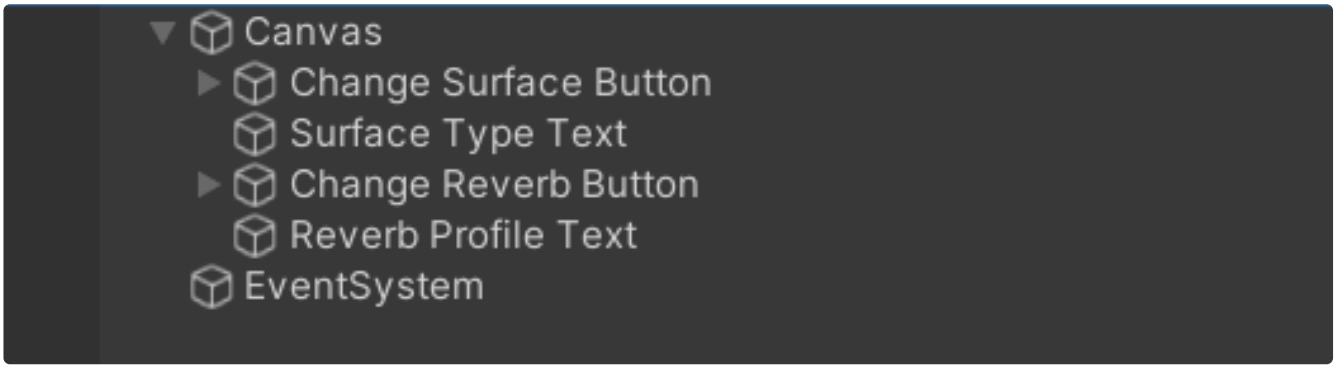
1. In the `Hierarchy`, select the Ground GameObject.
2. In the `Inspector`, click `Add Component` and add the DynamicMaterial script.
3. Ensure that the SurfaceType script is still attached to the Ground GameObject.



4.7 Create UI for Surface Type Switching

1. In the `Hierarchy`, right-click and select `UI > Button-TextMeshPro`. This will create a Canvas with a Button if it doesn't exist already.
2. Rename the Button to "ChangeSurfaceTypeButton".

3. Select the Button in the Hierarchy and in the Inspector, change the Text component to say "Change Surface".
4. In the `Hierarchy`, right-click on the Canvas and select `UI > Text-TextMeshPro`. Name this "SurfaceTypeText".
5. Position the Button and Text as desired in the Scene view.



4.8 Create UI Manager Script for Surface Type

1. In the `Project window`, right-click and select `Create > C# Script`. Name it "UIManager".
2. Double-click the script to open it in your code editor.
3. Replace the contents with the following code:

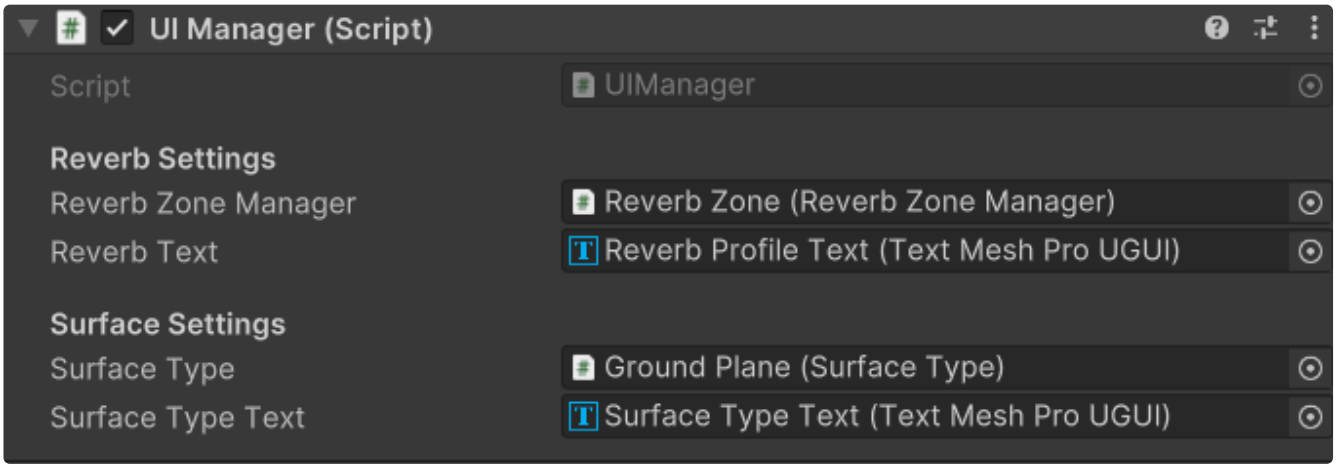
```
using UnityEngine;
using TMPro;

public class UIManager : MonoBehaviour
{
    [Header("Surface Settings")]
    public SurfaceType surfaceType;
    public TMP_Text surfaceTypeText;

    private int currentSurfaceTypeIndex = 0;

    void Start()
    {
        UpdateSurfaceTypeText();
    }

    public void ChangeSurfaceType()
    {
        currentSurfaceTypeIndex = (currentSurfaceTypeIndex + 1) %
System.Enum.GetValues(typeof(SubstrateType)).Length;
        SubstrateType newType = (SubstrateType)currentSurfaceTypeIndex;
        surfaceType.ChangeSurfaceType(newType);
        UpdateSurfaceTypeText();
    }
```

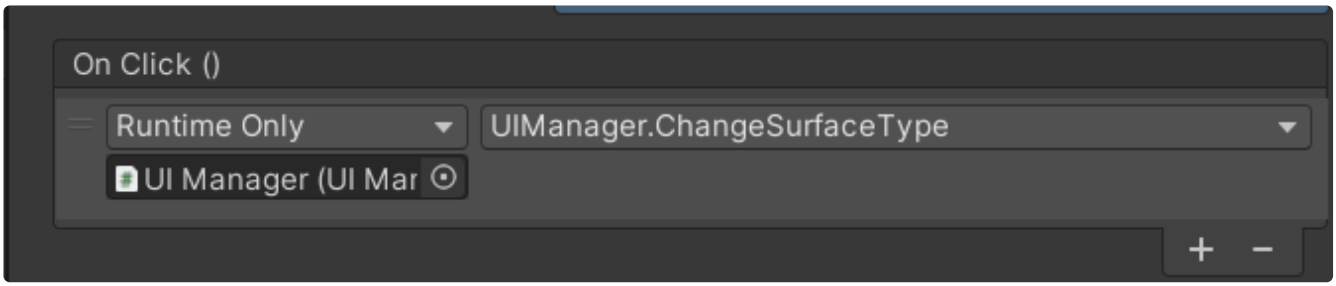
```
    private void UpdateSurfaceTypeText()
    {
        surfaceTypeText.text = "Current Surface: " +
surfaceType.surfaceType.ToString();
    }
}
```

4. Save the script and return to Unity.

4.9 Set Up Surface Type UI

1. Create an empty GameObject named "UI Manager" and add the UIManager script to it.
2. In the `Inspector` for UI Manager:
   a. Drag the Ground GameObject to the Surface Type field.
   b. Drag the SurfaceTypeText GameObject to the Surface Type Text field.
3. Select the ChangeSurfaceTypeButton in the Hierarchy.
4. In the `Inspector`, find the `OnClick()` section of the Button component.
5. Click the `+` button, drag the SurfaceUIManager GameObject to the object field.
6. In the function dropdown, select `UIManager > ChangeSurfaceType()`.





Tip

This setup allows for real-time changes to both surface types and materials. In a game, you could trigger these changes based on various events or player actions, creating a more dynamic and

interactive environment.

4.10 Test Dynamic Surface Types and Materials

1. Press Play in Unity.
2. Click the "Change Surface" button to cycle through surface types.
3. Observe how the material color changes dynamically with each surface type.
4. Ensure that the bouncing ball plays the correct sound for each surface type.

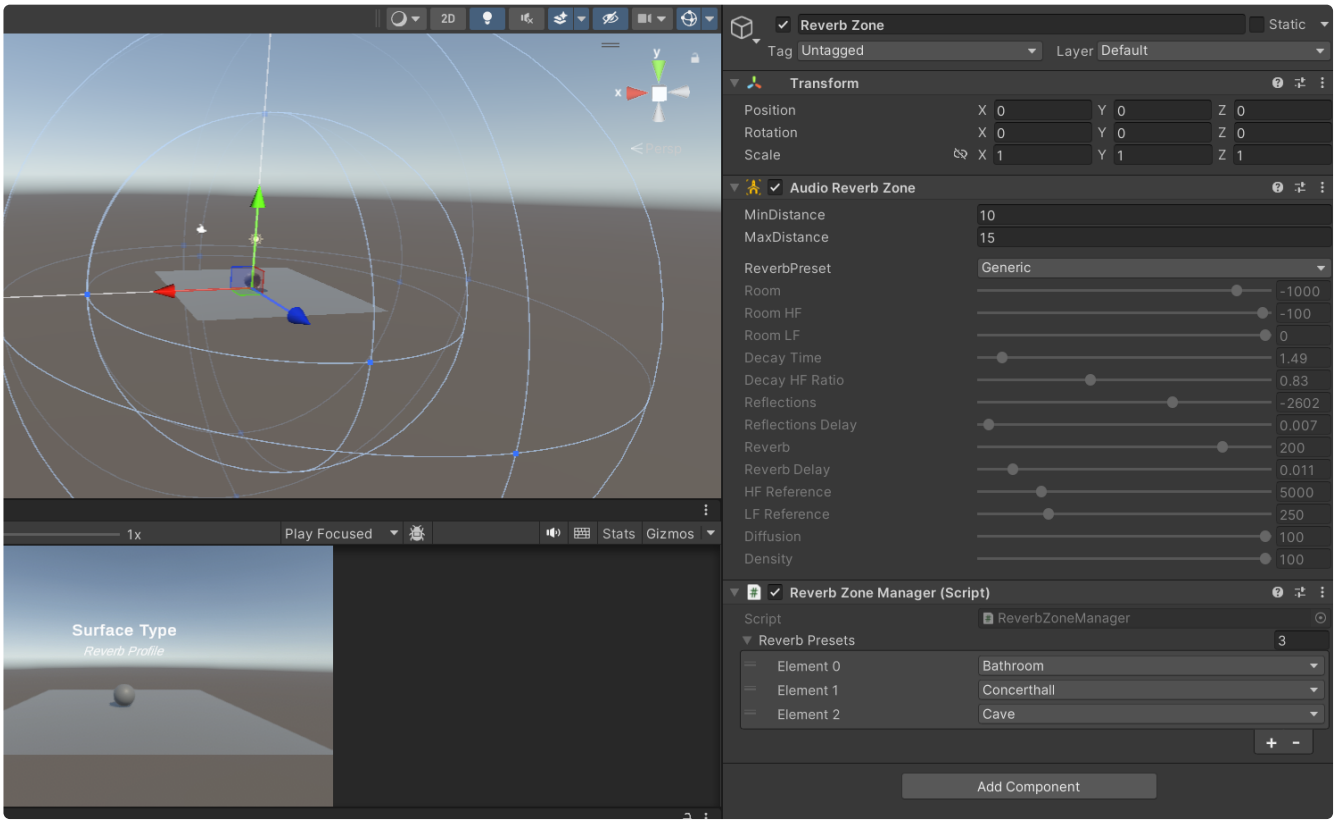
Step 5: Add Reverb Zones

5.1 Create Reverb Zone GameObject

1. In the `Hierarchy`, right-click and select Create Empty. Name it "ReverbZone".
2. With ReverbZone selected, in the Inspector, click `Add Component`.
3. Type "Audio Reverb Zone" and select it to add the component.

5.2 Create Reverb Presets

1. In the `Inspector` of the ReverbZone GameObject, find the Audio Reverb Zone component.
2. Click on the "Preset" dropdown and explore different presets.
3. Choose three presets you like (e.g., "Small Room", "Large Hall", "Outdoor"). *Remember these for later*.



5.3 Create ReverbZoneManager Script

1. In the `Project window`, right-click and select `Create > C# Script`. Name it "ReverbZoneManager".
2. Double-click the script to open it in your code editor.
3. Replace the contents with the following code:

```
using UnityEngine;

public class ReverbZoneManager : MonoBehaviour
{
    public AudioReverbPreset[] reverbPresets;
    private AudioReverbZone reverbZone;

    void Start()
    {
        reverbZone = GetComponent<AudioReverbZone>();
    }

    public void ChangeReverbZone(int index)
    {
        if (index >= 0 && index < reverbPresets.Length)
        {
            reverbZone.reverbPreset = reverbPresets[index];
        }
```

```
    }
}
```

4. Save the script and return to Unity.
5. Select the ReverbZone GameObject in the Hierarchy.
6. In the `Inspector`, click `Add Component` and add the ReverbZoneManager script.

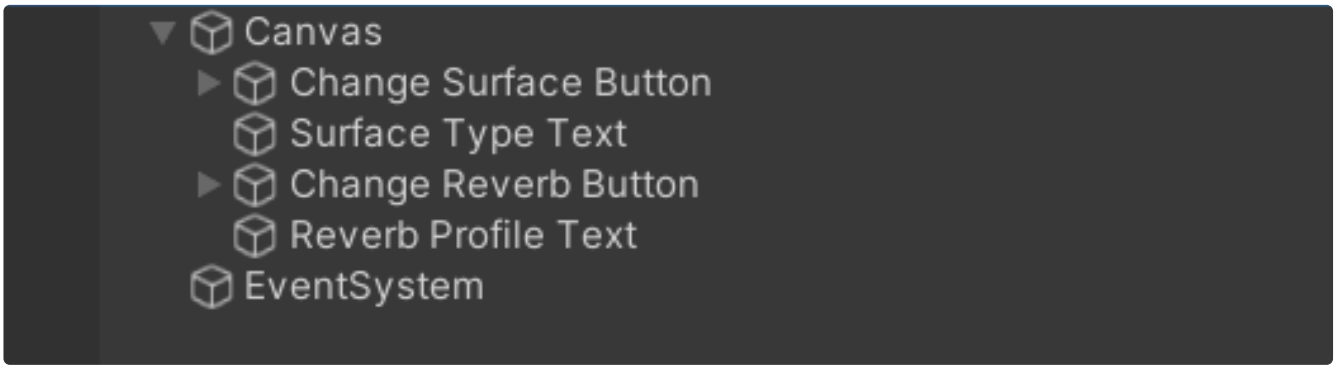
5.4 Set Up Reverb Presets

1. With the ReverbZone GameObject selected, find the ReverbZoneManager component in the Inspector.
2. Set the Size of the Reverb Presets array to 3 (or however many presets you chose).
3. For each element in the array, select one of the presets you chose earlier from the dropdown.

> **Tip**
>
> Reverb zones can greatly enhance the sense of space in your game audio. Experiment with different settings to find what works best for your scene.

5.5 Create UI for Reverb Switching

1. In the `Hierarchy`, right-click and select `UI > Button-TextMeshPro`. This will create a Canvas with a Button.
2. Select the Button in the `Hierarchy` and in the `Inspector`, change the Text component to say "Change Reverb".
3. In the Hierarchy, right-click on the `Canvas` and select `UI > Text-TextMeshPro`. This will add a Text object to display the current reverb.
4. Position the Button and Text as desired in the Scene view.



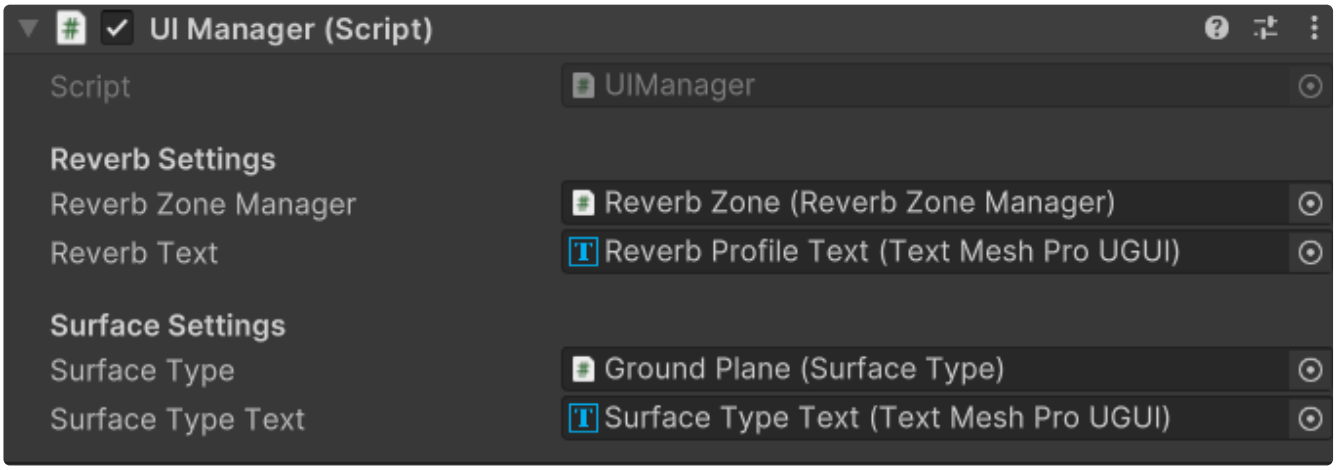
5.6 Create UI Manager Script

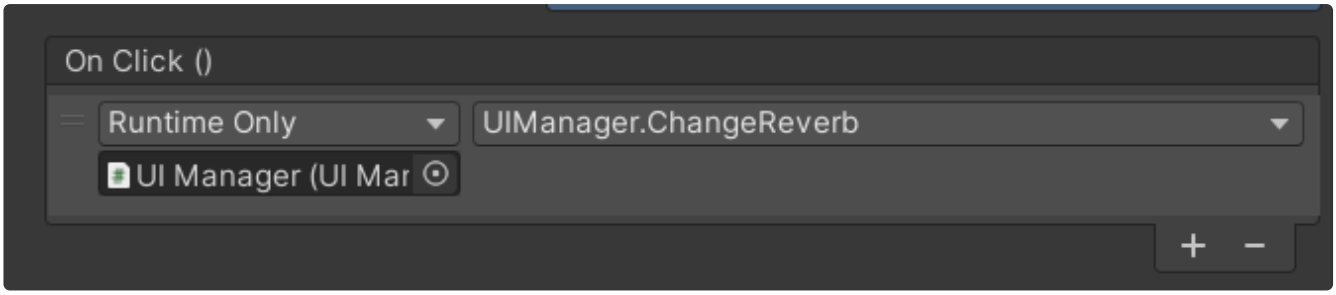
1. In the `Project window` locate the "UIManager" script.

2. Double-click the script to open it in your code editor.
3. Update the script with the following code:

```
using UnityEngine;
using TMPro;

public class UIManager : MonoBehaviour
{
    //Same code from earlier...

    [Header("Reverb Settings")]
    public ReverbZoneManager reverbZoneManager;
    public TMP_Text reverbText; // Changed to TMP_Text

    private int currentReverbIndex = 0;

    void Start()
    {
        //Same code from earlier...
        UpdateReverbText();
    }

        //Same code from earlier...

    public void ChangeReverb()
    {
        currentReverbIndex = (currentReverbIndex + 1) %
reverbZoneManager.reverbPresets.Length;
        reverbZoneManager.ChangeReverbZone(currentReverbIndex);
        UpdateReverbText();
    }

    private void UpdateReverbText()
    {
        reverbText.text = "Current Reverb: " +
reverbZoneManager.reverbPresets[currentReverbIndex].ToString();
    }
}
```

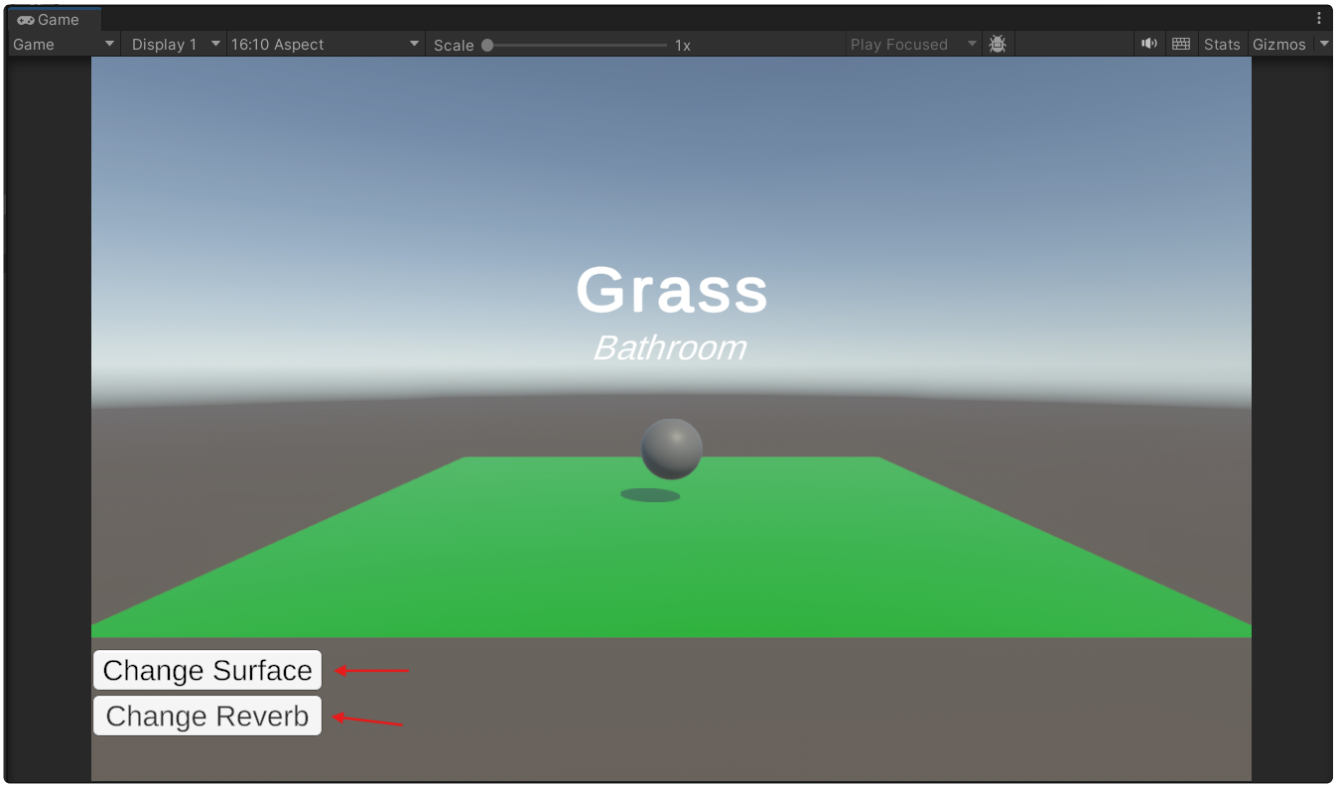
1. Save the script and return to Unity.
2. In the `Inspector` for UIManager, drag the ReverbZone GameObject to the Reverb Zone Manager field.
3. Drag the Text GameObject you created to the Reverb Text field.
4. Select the Button in the `Hierarchy`, find the `OnClick()` section in the Inspector.
5. Click the `+` button, drag the UIManager GameObject to the object field.
6. In the function dropdown, select `UIManager > ChangeReverb()`.





5.7 Test Reverb Switching

1. Press Play in Unity.
2. Click the "Change Reverb" button to cycle through reverb presets.
3. Listen for the change in reverb as you click.



Part II: Music Transitions

Setup

1. In Unity, go to `File > New Scene`. Name it "MusicTransitionScene".
2. In the `Hierarchy`, right-click and select `Create Empty`. Name it "MusicManager".
3. You will need some free music tracks for this lab. One that is calmer "Exploration" and one that is more high paced "combat" music. Try this pack from the Asset Store: https://assetstore.unity.com/packages/audio/music/25-fantasy-rpg-game-tracks-music-pack-240154

Step 1: Create the Music Manager

1.1 Create MusicManager Script

1. In the `Project window`, right-click and select `Create > C# Script`. Name it "MusicManager".
2. Double-click the script to open it in your code editor.
3. Replace the contents with the following code:

```
using UnityEngine;
using System.Collections;

public class MusicManager : MonoBehaviour
{
    public AudioSource musicSource;
    public AudioClip explorationMusic;
    public AudioClip combatMusic;

    public float transitionTime = 2.0f;

    private bool inCombat = false;

    void Start()
    {
        musicSource.clip = explorationMusic;
        musicSource.Play();
    }

    public void ToggleCombatMusic()
    {
        StartCoroutine(TransitionMusic());
    }

    private IEnumerator TransitionMusic()
    {
        float timeElapsed = 0;
        float startVolume = musicSource.volume;
```

```
        AudioClip targetClip = inCombat ? explorationMusic : combatMusic;

        while (timeElapsed < transitionTime)
        {
            musicSource.volume = Mathf.Lerp(startVolume, 0, timeElapsed /
transitionTime);
            timeElapsed += Time.deltaTime;
            yield return null;
        }

        musicSource.Stop();
        musicSource.clip = targetClip;
        musicSource.Play();

        timeElapsed = 0;
        while (timeElapsed < transitionTime)
        {
            musicSource.volume = Mathf.Lerp(0, startVolume, timeElapsed /
transitionTime);
            timeElapsed += Time.deltaTime;
            yield return null;
        }

        inCombat = !inCombat;
    }
}
```

4. Save the script and return to Unity.

ⓘ Code Explainer: Coroutines

Coroutines in Unity are a powerful way to write asynchronous code that can be paused and resumed.

How they work:

1. Coroutines run parallel to the main update loop.
2. They can be paused using `yield` statements.
3. They resume execution on the next frame or after a specified time.

Key features:

- Start with `StartCoroutine()`
- Use `IEnumerator` as the return type
- Can yield return `null`, `WaitForSeconds`, or other custom conditions

Use cases:

- Animations over time
- Delayed actions
- Asynchronous operations (like loading)
- Gradual changes (like our music transition)

In this script:
We use a coroutine for smooth music transition. It allows us to gradually change the volume over time, creating a seamless blend between tracks without blocking the main game loop.

1.2 Set Up MusicManager GameObject

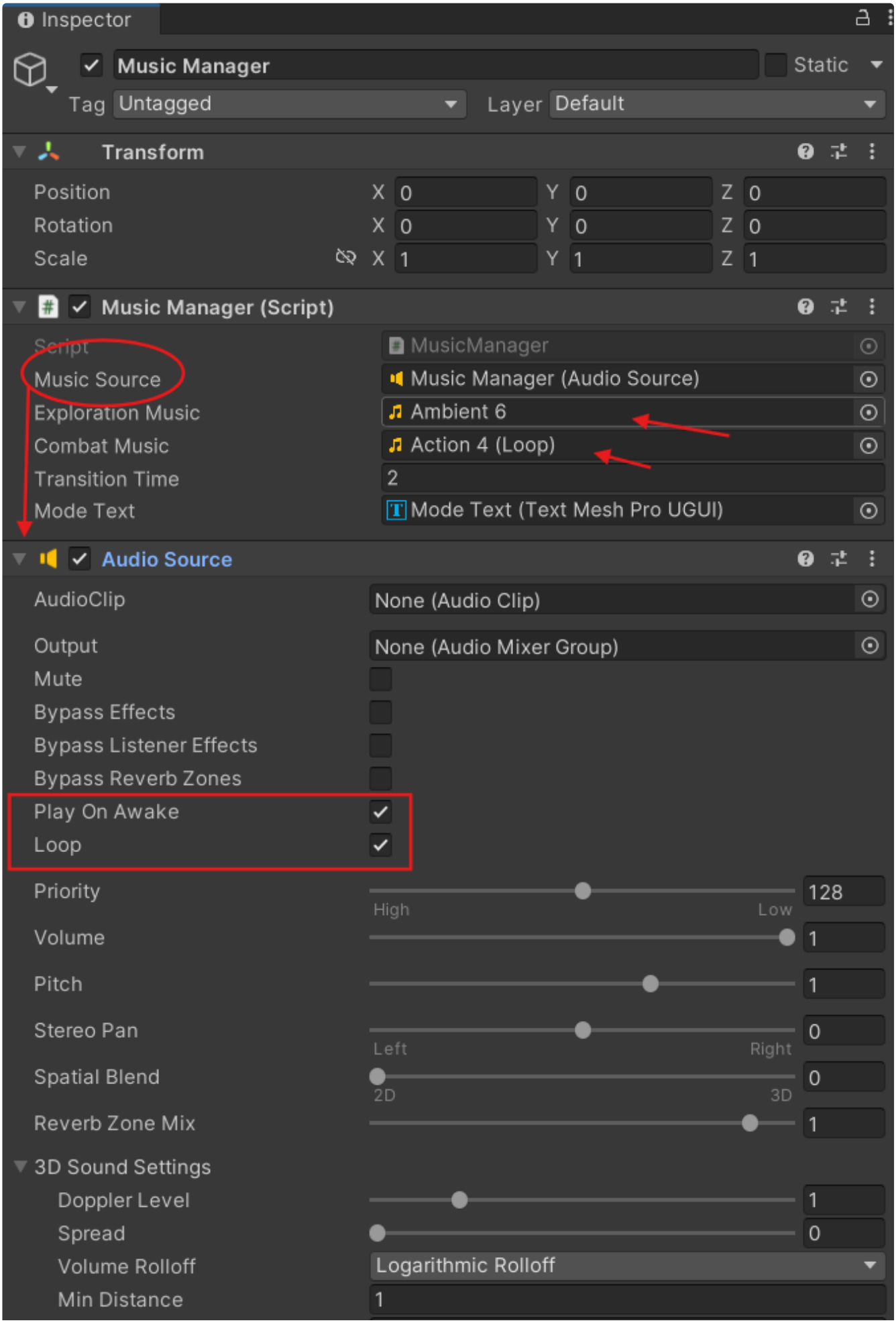
1. In the `Hierarchy`, select the MusicManager GameObject.
2. In the `Inspector`, click `Add Component` and add the MusicManager script.
3. Click `Add Component` again and add an Audio Source component.

1.3 Assign Audio Clips

⚠ Optional Step

Creating the music folder below may be optional depending on where your sourced audio assets come from. But again, it is good organizational practice.

1. In your Project window, create a new folder called "Music".
2. Find two music tracks online: one for exploration/peaceful gameplay and one for combat. Make sure they are in a format Unity supports (e.g., MP3 or WAV).
3. Drag and drop these music files into the Music folder you just created in Unity.
4. In the `Hierarchy`, select the MusicManager GameObject.
5. In the `Inspector`, find the MusicManager script component.
6. Drag the peaceful music clip from the `Project window` to the "Exploration Music" field in the Inspector.
7. Drag the combat music clip from the `Project window` to the "Combat Music" field in the Inspector.
8. In the `Audio Source component`, make sure `Play On Awake` is checked and `Loop` is checked.


Inspector
Music Manager
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Music Manager (Script)
Script MusicManager
Music Source Music Manager (Audio Source)
Exploration Music Ambient 6
Combat Music Action 4 (Loop)
Transition Time 2
Mode Text Mode Text (Text Mesh Pro UGUI)
Audio Source
AudioClip None (Audio Clip)
Output None (Audio Mixer Group)
Mute
Bypass Effects
Bypass Listener Effects
Bypass Reverb Zones
Play On Awake
Loop
Priority 128
High Low
Volume 1
Pitch 1
Stereo Pan 0
Left Right
Spatial Blend 0
2D 3D
Reverb Zone Mix 1
3D Sound Settings
Doppler Level 1
Spread 0
Volume Rolloff Logarithmic Rolloff
Min Distance 1

Free Music on Asset Store

There are several free music packs on the Unity Asset store that will work perfectly well for this lab. IF you are having trouble finding one try `25 Fantasy RPG Game Tracks Music Pack` The key is to find two tracks that are diverse enough to hear the transition.

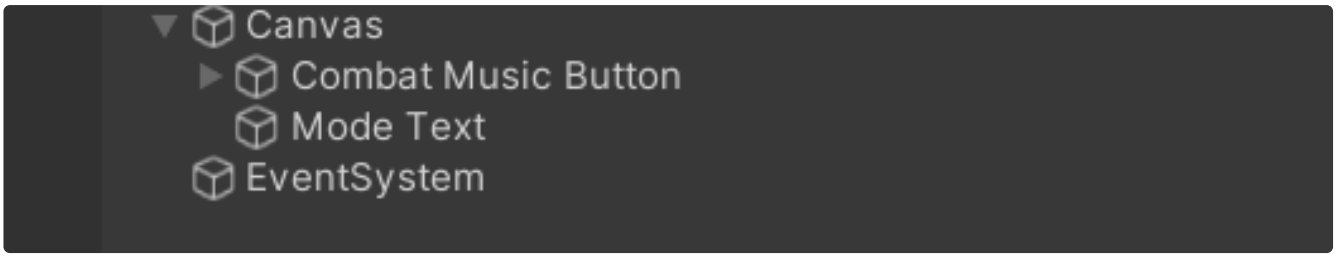
Step 2: Create UI for Music Transition

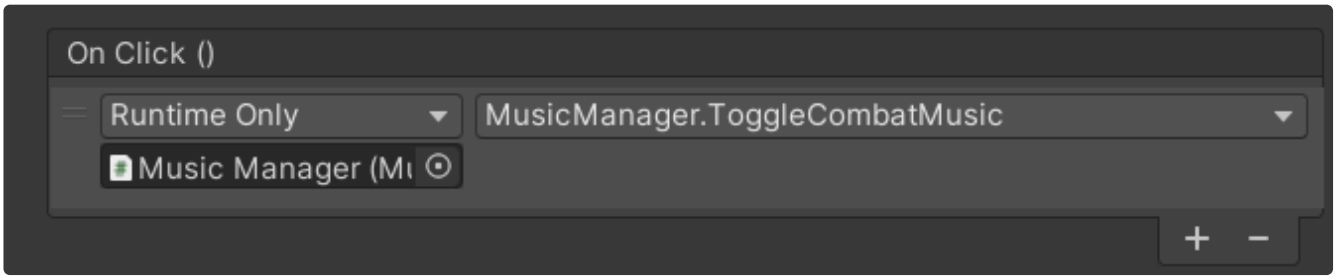
2.1 Create UI Button

1. In the Hierarchy, right-click and select `UI > Button-TextMeshPro`. This will create a Canvas with a Button.
2. Select the Button in the Hierarchy.
3. In the `Inspector`, find the `Text` component (child of the Button).
4. Change the text to "Toggle Combat Music".
5. Position the Button where it fits best.

Step 3: Connect UI to Music Manager

1. In the `Hierarchy`, select the Button you created.
2. In the `Inspector`, find the Button component.
3. In the `OnClick()` section, click the `+` button to add a new event.
4. Drag the MusicManager GameObject from the `Hierarchy` to the object field in this new event.
5. In the function dropdown that appears, select `MusicManager > ToggleCombatMusic()`.





Step 4: Test Your Music Transition

1. In the Unity editor, press the `Play` button to enter Play Mode.
2. You should hear the exploration music start playing automatically.
3. Click the "Toggle Combat Music" button.
4. Listen as the music smoothly transitions from the exploration track to the combat track.
5. Click the button again to transition back to the exploration music.

> **Tip**
>
> If you can't hear any music, check that your system volume is up and that the Audio Source component on the MusicManager has its volume set above 0.

Step 5: (Optional) Bonus Refine

1. Try adjusting the "Transition Time" value in the MusicManager component to make the transition faster or slower.
2. Experiment with different music tracks to see how they transition.
3. Consider adding a text element that displays whether you're currently in "Exploration" or "Combat" mode.

5.1 Add Mode Display Text

1. In the `Hierarchy`, right-click on the Canvas and select `UI > Text-TextMeshPro`.
2. Rename this new Text object to "ModeText".
3. Position it at the top of the screen using the `Rect Transform` component.
4. Set its text to "Current Mode: Exploration".

5.2 Update MusicManager Script

1. Open the MusicManager script in your code editor.
2. Add the Following using statement the top of the script

```
using TMPro;
```

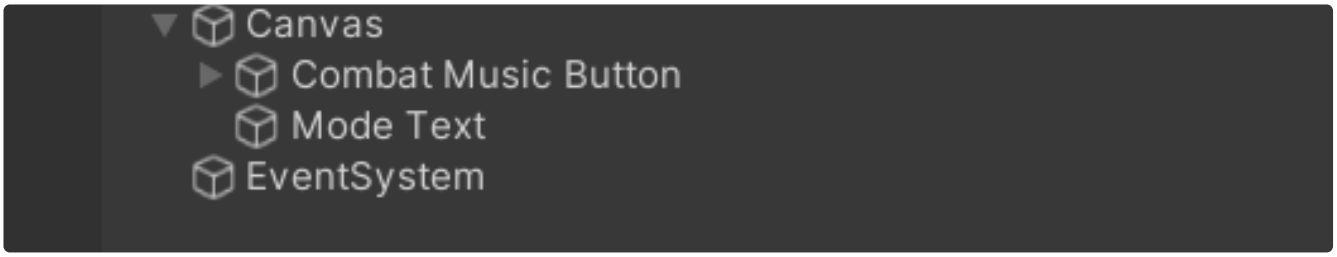
1. Add the following public variable at the top of the class:

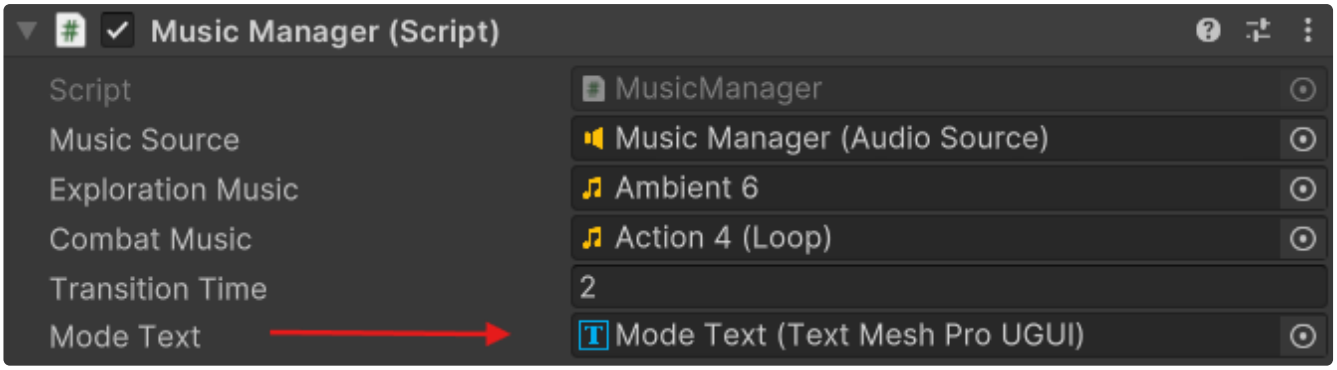
```
public TMP_Text modeText;
```

3. At the end of the `Start()` method and the `TransitionMusic()` coroutine (IEnumerator), add:

```
modeText.text = "Current Mode: " + (inCombat ? "Combat" : "Exploration");
```

1. Save the script and return to Unity.
2. In the Inspector for the MusicManager, drag the ModeText object to the new "Mode Text" field.





Conclusion

✓ You've Completed the Audio Lab

You've successfully created two dynamic audio systems in Unity:
Part I: Dynamic Surface Sound Effects

> - Implemented a system that plays different sound effects based on surface types
> - Implemented a reverb zone system with real-time switching

Part II: Smooth Music Transitions

> - Created a music manager that smoothly transitions between different tracks
> - Utilized coroutines for gradual volume changes