Game Audio: Final Project

Dave Maynard

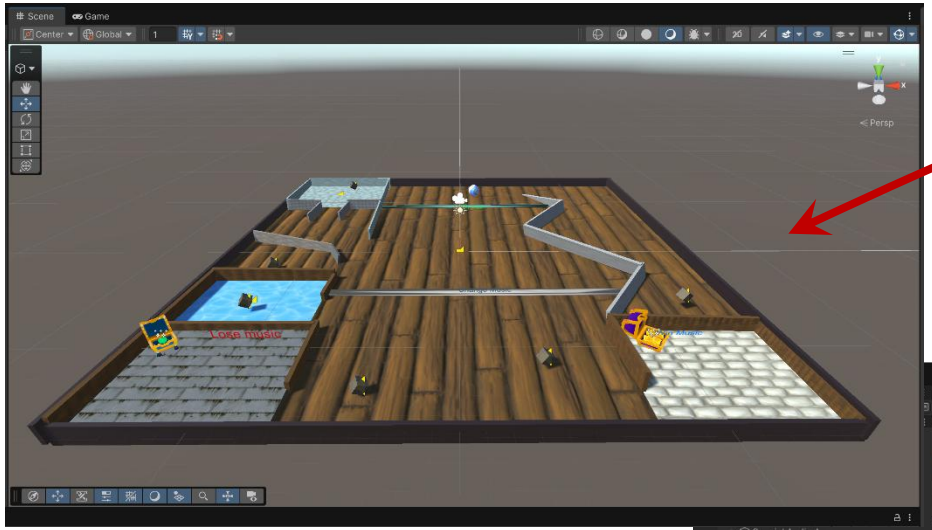
Medieval Encounters: The Quest for Nerk

“1950’s Aliens Invade the Kingdom”

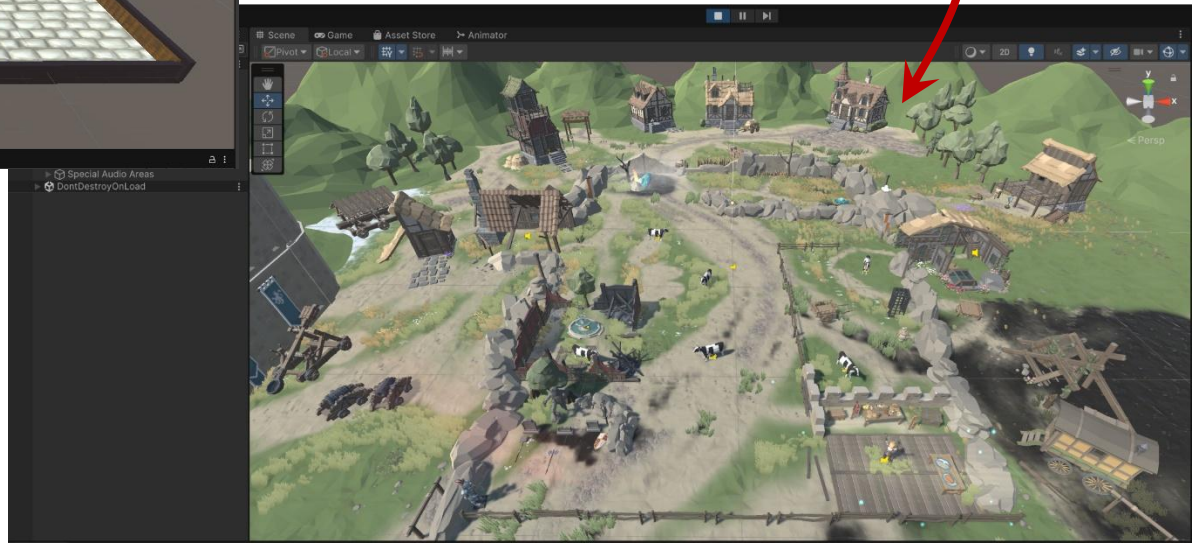
Custom Midjourney AI Generated Artwork

Game Audio: Final Project

Dave Maynard



Turned this into that ☺
in 2 Weeks



Game Audio: Final Project

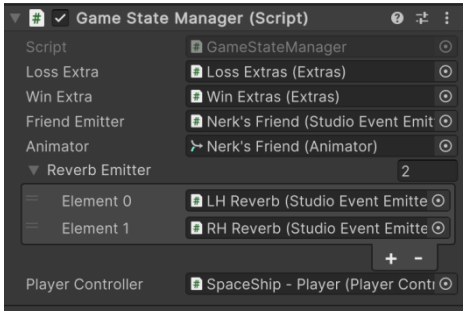
Dave Maynard

NOTE

You can hit the [ESCape] key to exit this assignment program.
You can <u>also</u> hit the [1] key to reset the level without exiting.

These are merely code snippets; the complete code encompasses much more than what is presented here, simplified for the sake of clarity.

```
Unity Message | 0 references
void Update()
{
    if (Input.GetKeyUp(KeyCode.Escape))
    {
        Application.Quit();
    }

    if (Input.GetKeyUp(KeyCode.Alpha1))
    {
        ResetLevel();
    }
}
```



I've incorporated a Singleton Game State Manager class into the project, which is now part of the game's hierarchy. It's designed to detect when the [Escape] key is pressed and ensures the program exits gracefully. It's also designed to detect when the [1] key is pressed to reset the level without exiting the program.

```
public void ResetLevel()
{
    foreach (var emitter in reverbEmitter)
    {
        emitter.Stop();
    }

    playerController.StopFMOD();
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
```

```
public void StopFMOD()
{
    musicEv.stop(FMOD.Studio.STOP_MODE.IMMEDIATE);
    flyingEv.stop(FMOD.Studio.STOP_MODE.IMMEDIATE);
}
```

For those who might be curious, these methods are how I ensure that all FMOD audio, including the music and the reverb zones, are forced to stop and the level reloads.

Game Audio: Final Project

Story of “1950’s Aliens Invade the Kingdom”

Our player lands on a planet with a rich Kingdom. He’s looking for his buddy “Nerk” who came here last Wednesday.

He’s sad to find his buddy crashed and his ship is on fire! But maybe he’s around somewhere.

But YAY there are cows to abduct on this planet. He’s happy again!

And there are lots of places to explore!

There are also places throughout that are special nooks that give everyday sounds a playful twist!

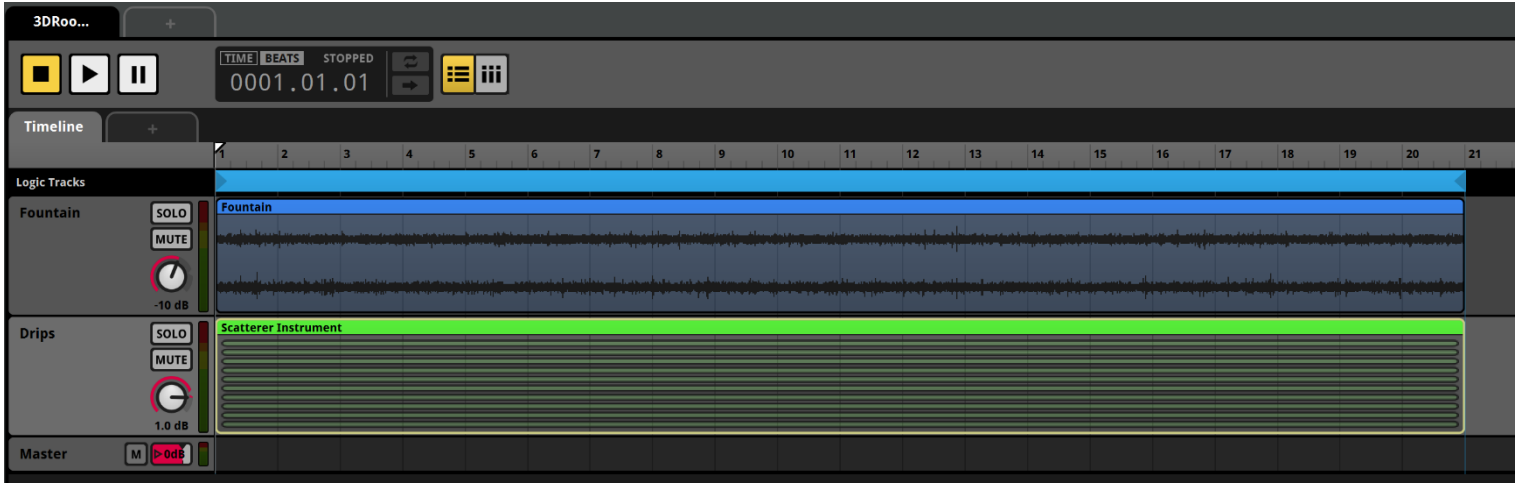
Game Audio: Final Project

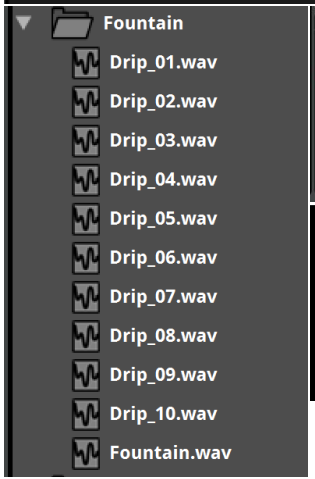
Story of “1950’s Aliens Invade the Kingdom”

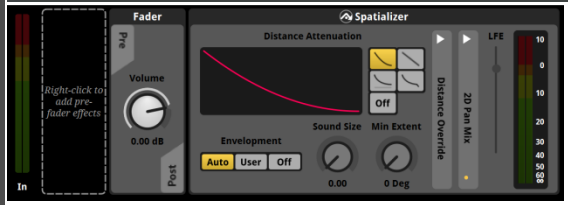
If our player finds Nerk’s friend with all the snacks he WINS !

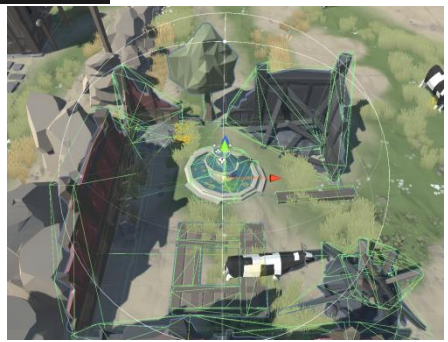
If our player finds the graveyard he is locked in and LOSES!

Game Audio: Final Project: FMOD Creation & Integration





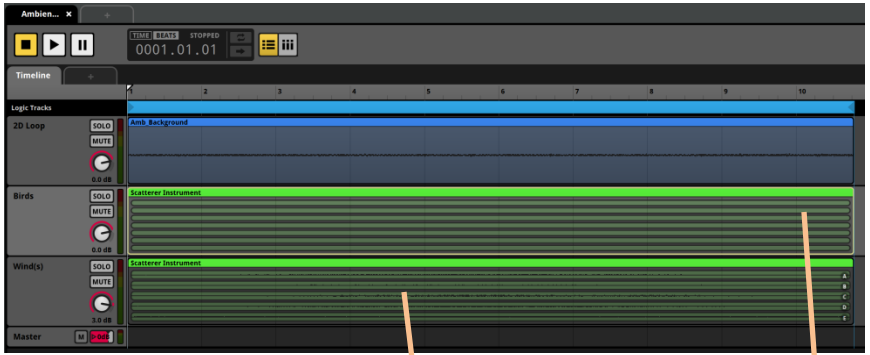




The 3D room features a fountain that plays a looping water stream sound. Additionally, there are 10 sporadic dripping sounds that are played by the scatterer instrument.

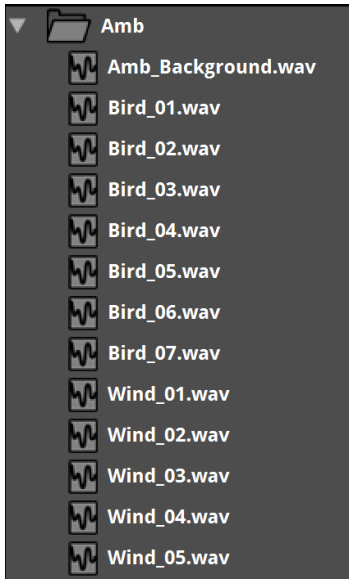
I adjusted the emitter distances in Unity by using the override attenuation feature.

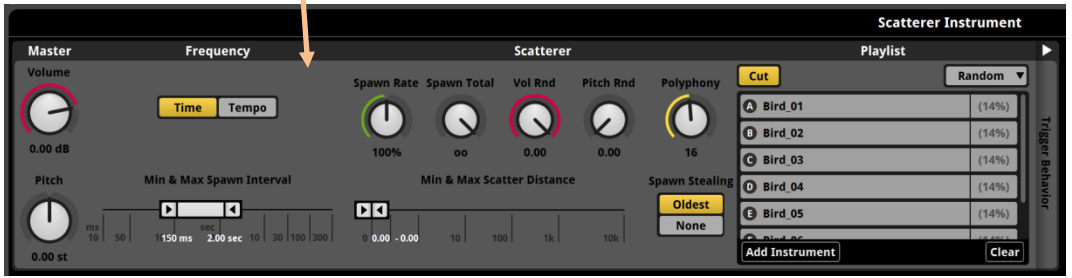
Game Audio: Final Project: FMOD Creation & Integration



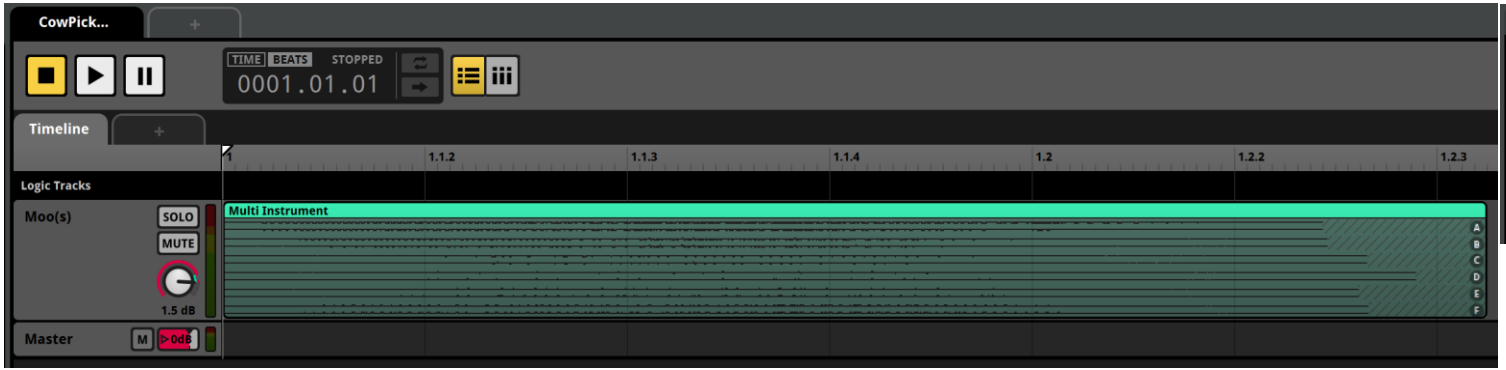
The ambient timeline includes a continuous loop of park sounds with no human voices. Additionally, it features two scatterer instruments

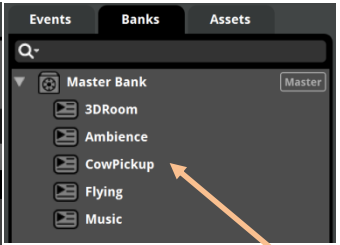
One randomly plays seven distinct bird calls, and another that randomly emits five varieties of wind sounds. I have fine-tuned the parameters of the scatterer instruments, such as polyphony, minimum and maximum spawn intervals, and minimum and maximum scatter distances, to achieve a blend that I thought best fit the scene.



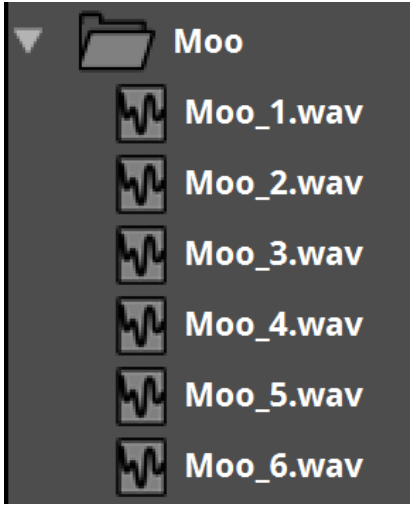


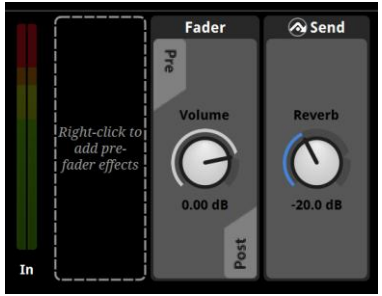
Game Audio: Final Project: FMOD Creation & Integration





Made sure the "CowPickup" event was in the Master Bank.



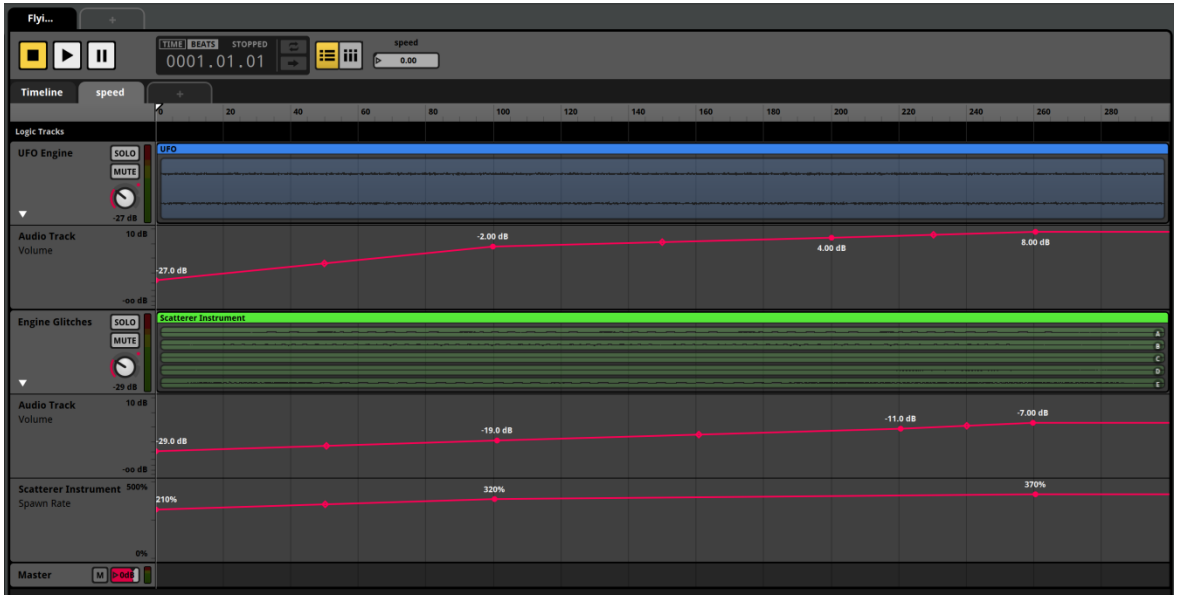


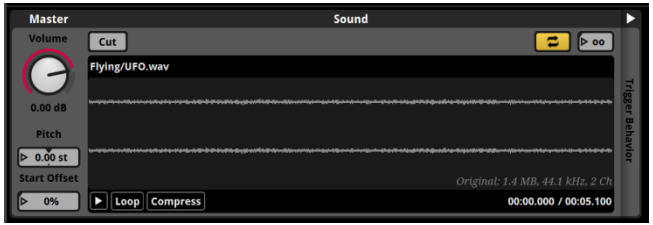
Master Track

Yes, that is me making the "moo sounds" for the cow ☺ Utilizing the multi-instrument, I incorporated six distinct "moo" sounds and combined them with a single teleport sound that features three varied "whoosh" effects.

I did rename this Event timeline, the details of which, including the rationale, are discussed in the "Extra" section of this write-up.

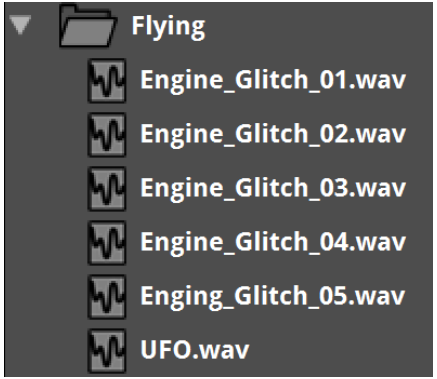
Game Audio: Final Project: FMOD Creation & Integration

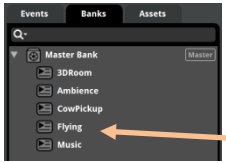




Yes, that is also me making the main engine sounds using my mouth ☺ along with a few other sounds baked in. I employed the scatterer instrument to integrate five unique engine glitching sounds.

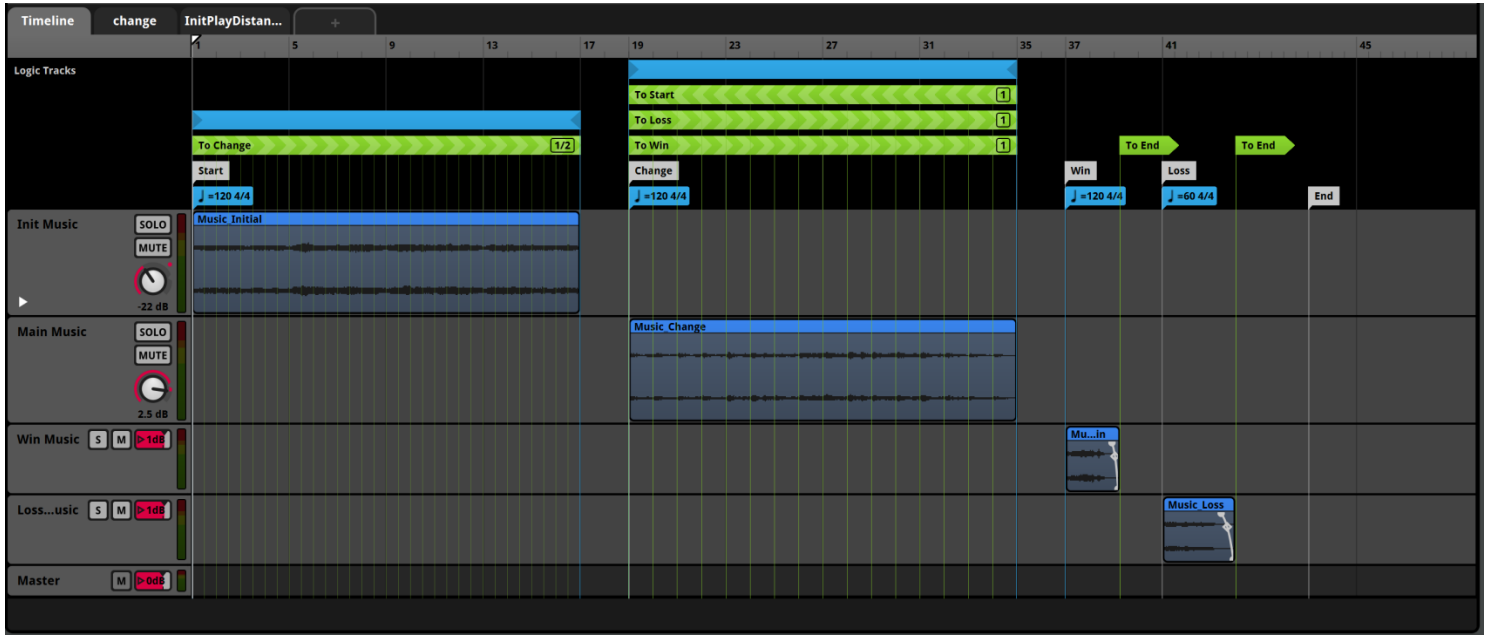
By using the ship's speed data from the game engine, I automated the adjustment of the overall volume(s) and the spawn rate of the engine glitches.





Made sure the “Flying” event was in the Master Bank.

Game Audio: Final Project: FMOD Creation & Integration



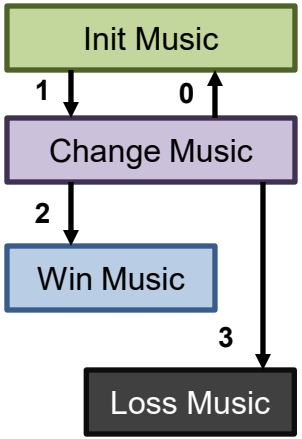
Following the specifications of the assignment, I created an Initial Music piece, Change Music, and both Win and Loss Stingers. My approach to the compositions was influenced by the style of 1950s science fiction, with a comedic twist.

I deviated slightly from the original transition markers and logic requirements. This was primarily because the layout of the game area made it impossible to transition directly from the "initial" music to the win or loss zones; one must pass through the "change" music area to enter the win or loss areas.

Game Audio: Final Project: FMOD Creation & Integration

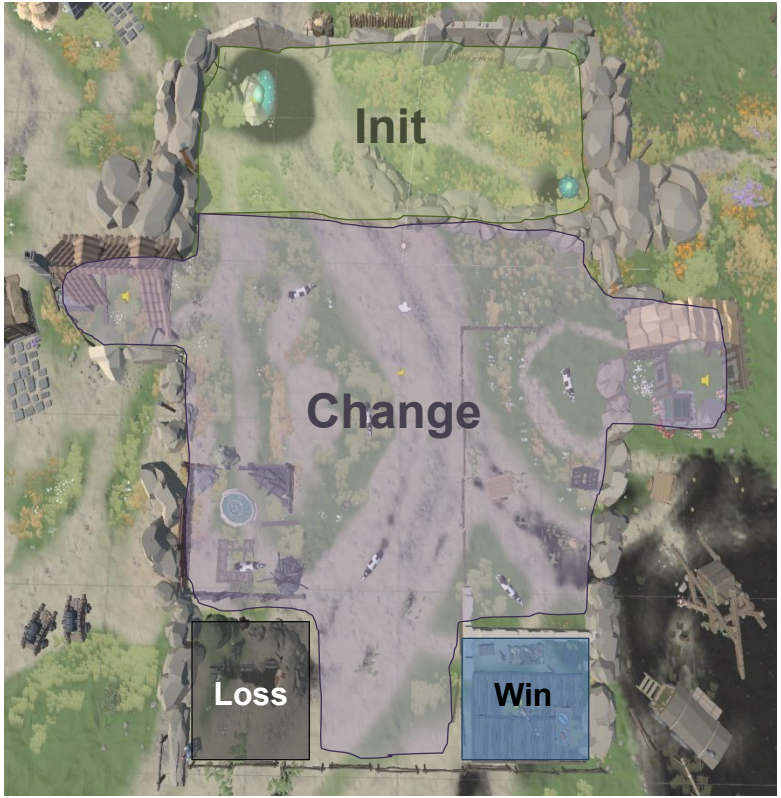
Logic Map for Music Transitions

 # = ‘change’ parameter value



Modified to Start Immediately

Logic going back to Init Music modified via Extra Logic described on Next Slide



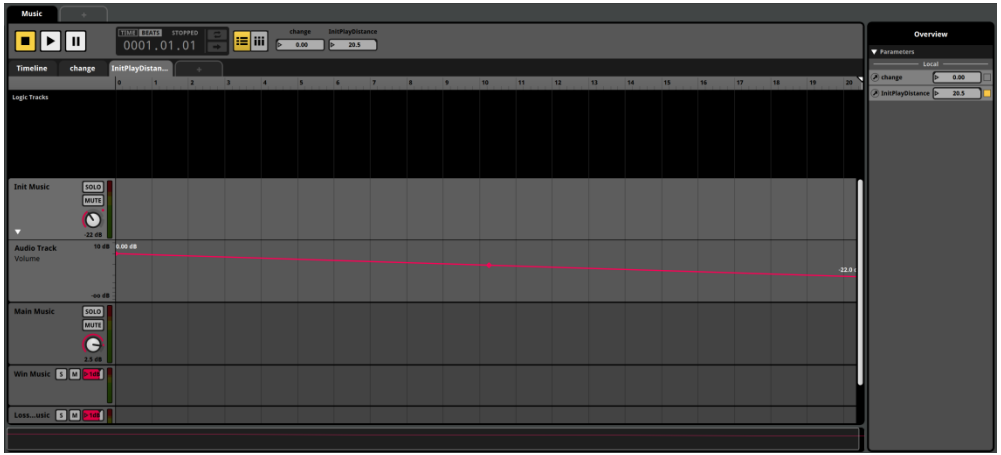
Very Rough Overlay

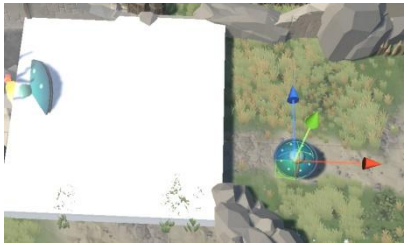
Via this style of layout, the player must pass through the ‘Change’ area before they can reach the ‘Win’ or ‘Loss’ areas.

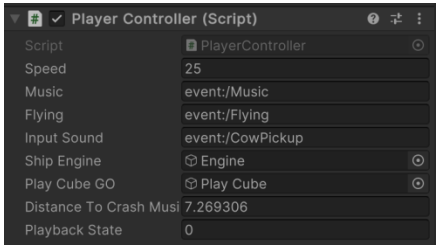
Due to this logical mapping, the 'Win' or 'Loss' stingers cannot be triggered from the 'Init' area.

Consequently, the logic presented in this and the previous FMOD slide adheres to the established flow of the game’s physical layout.

Game Audio: Final Project: FMOD Creation & Integration





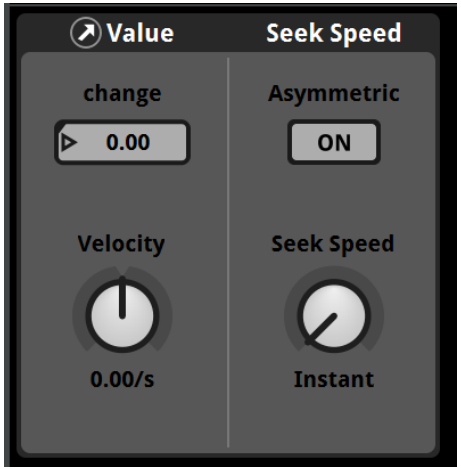


I intentionally added a new floating-point distance parameter in FMOD to craft a dramatic crescendo for the 1950s-style crash site discovery. Although I'm aware that this effect could potentially be achieved using a 2D/3D timeline and emitter setup, I chose to understand this challenge through coding within this assignment.

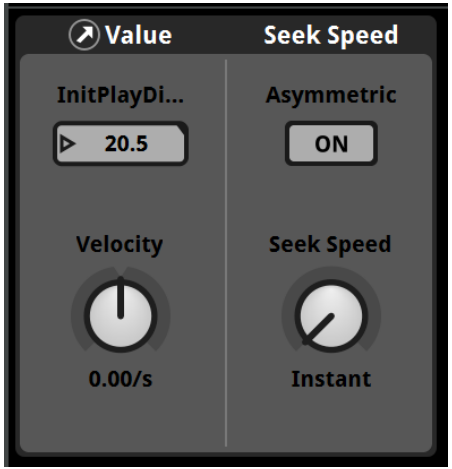
The resulting audio effect is that as the player's ship approaches the crash site, the volume of the dramatic "initial" music increases, and it decreases as the distance grows. Furthermore, the code I developed allows for a seamless transition back to the dramatic "initial" music from the "change" music, contingent on the player's interaction with a collider cube and their proximity to the crash site.

The code is discussed in the "Extra" section of this write-up.

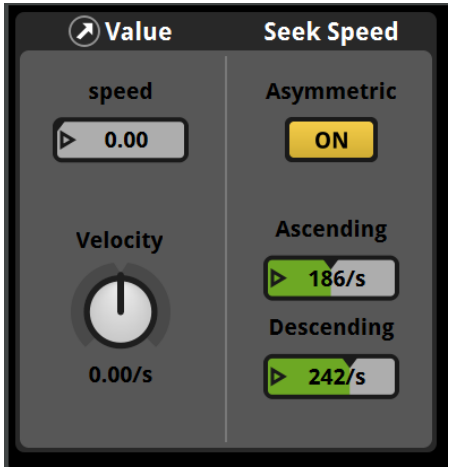
Game Audio: Final Project: FMOD Creation & Integration



“change” Parameter



“InitPlayDistance” Parameter



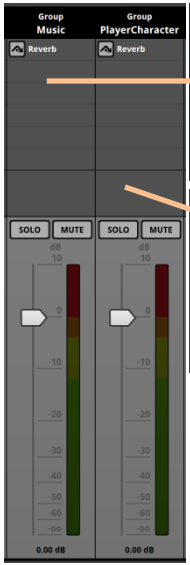
“speed” Parameter

The seek speeds for the "change" and my custom "InitPlayDistance" parameter must be instantaneous. For the "change" parameter, this is because it represents a discrete event, shifting from one numeric value to another, which in turn alters the music state. As for the "InitPlayDistance" parameter, it's a real-time value critical for adjusting the volume of the "initial" music associated with the crash site.

Conversely, the "speed" parameter required a bit more flexibility in its values to provide a sense of acceleration and deceleration in the ship's engine sounds, allowing for a fast but gradual increase or decrease in volume.

Game Audio: Final Project: FMOD Creation & Integration
Reverb Snapshot (Left House)

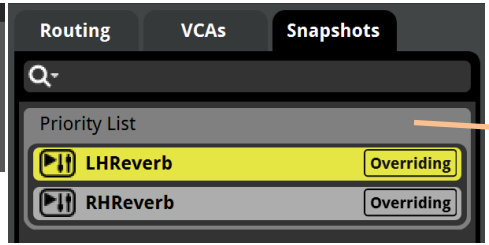
Please note that both the ship's flying audio and the music are affected by the reverb zones. Since moving the ship is required to hear the effect, adding the music was thought to enhance the demonstration of the reverb zone's impact.



Please note that the reverb settings are intentionally exaggerated to demonstrate the "reverb snapshot" requirement of the assignment.

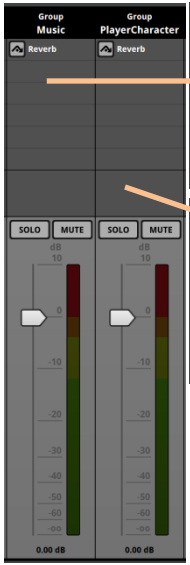
For the "Left House," the reverb zone encompasses the entire area of the destroyed house.

For each “group” they have the “Wet Level” set to ‘off’ when the snapshots are not active.



Game Audio: Final Project: FMOD Creation & Integration
Reverb Snapshot (Right House)

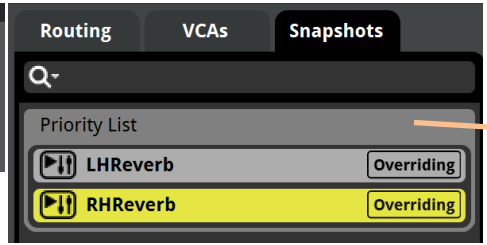
Please note that both the ship's flying audio and the music are affected by the reverb zones. Since moving the ship is required to hear the effect, adding the music was thought to enhance the demonstration of the reverb zone's impact.



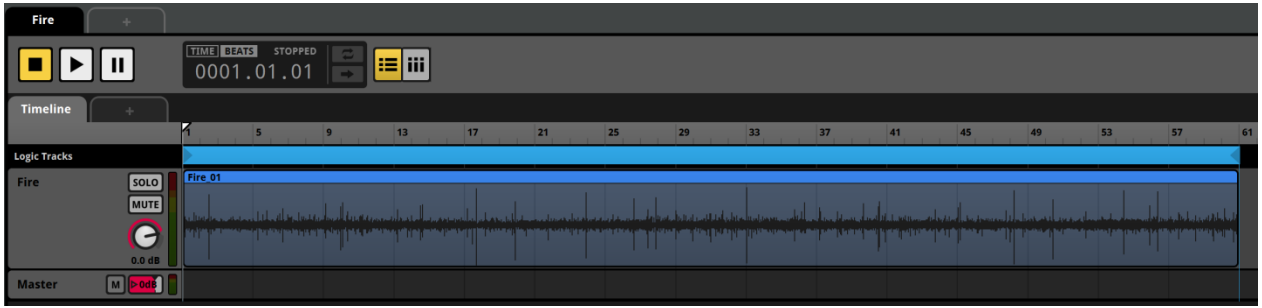
Please note that the reverb settings are intentionally exaggerated to demonstrate the "reverb snapshot" requirement of the assignment.

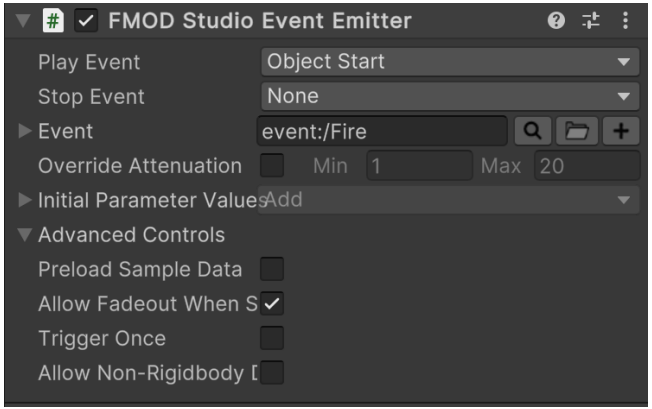
For the "Right House," the reverb zone is limited to the area under the roof, demonstrating how the ship can enter the open area to become reverb-free again.

For each “group” they have the “Wet Level” set to ‘off’ when the snapshots are not active.



Game Audio: Final Project: FMOD Creation & Integration
Fire at the Crash Site (FMOD 3D Timeline)

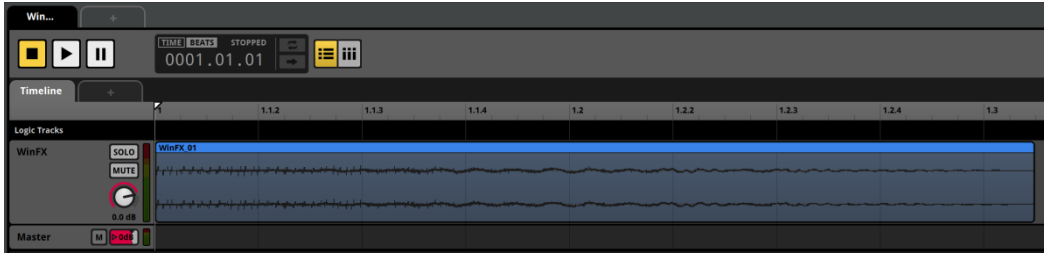


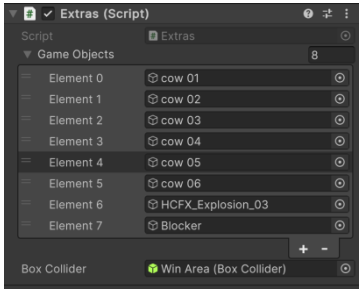


This was simply an extra 3D Audio Emitter placed in world with a looping fire sound.

Game Audio: Final Project: FMOD Creation & Integration
One-Shot SFX Played when Win Occurs

Win





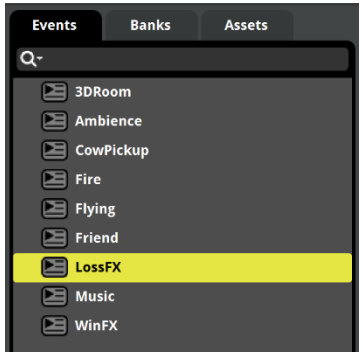

```
Unity Script | 2 references
public class Extras : MonoBehaviour
{
    [SerializeField]
    private List<GameObject> gameObjects;
    [SerializeField]
    private BoxCollider boxCollider;

    2 references
    public void Engage()
    {
        boxCollider.enabled = false;

        foreach (GameObject go in gameObjects)
        {
            go.SetActive(true);
        }

        GameStateManager.instance.GoIdle();
    }
}
```

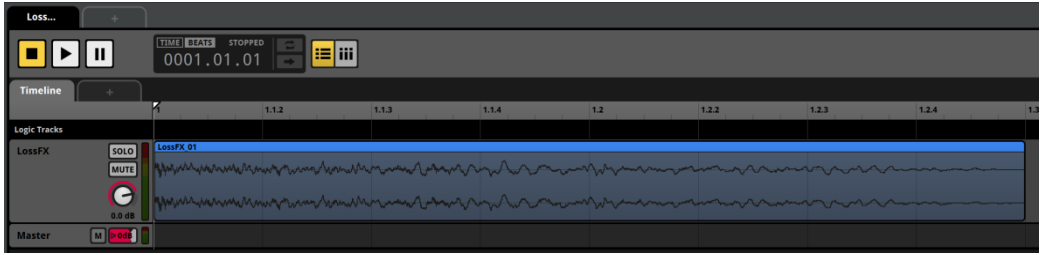
```
if (other.gameObject.CompareTag ("wincube"))
{
    playbackState = 2;
    FMODUnity.RuntimeManager.PlayOneShot(winSound);
    musicEv.setParameterByName("Change", playbackState);
    GameStateManager.instance.winExtra.Engage();
}
```

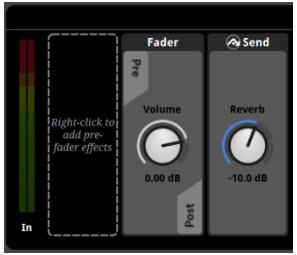
In summary, the updated code in the player controller triggers the one-shot SFX audio, activates the particle effect, disables a box collider to prevent multiple triggers, sets all GameObjects in the list to active (the cows and a collision blocker), and transitions "Nerk's Friend's" animation to an idle state, stopping her from waving and making her stand idle.

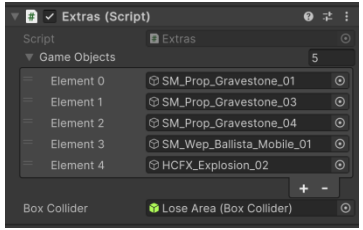
Game Audio: Final Project: FMOD Creation & Integration
One-Shot SFX Played when Loss Occurs

Loss







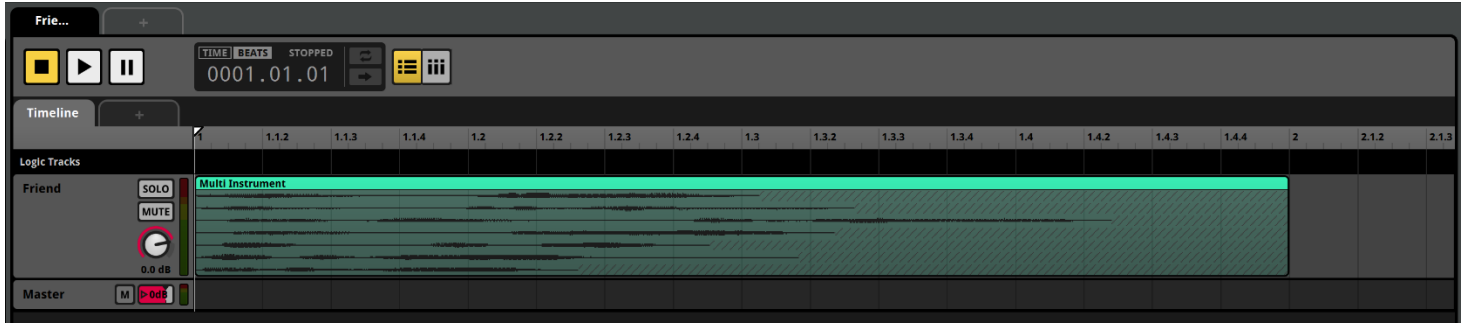



```
Unity Script | 2 references
public class Extras : MonoBehaviour
{
    [SerializeField]
    private List<GameObject> gameObjects;
    [SerializeField]
    private BoxCollider boxCollider;

    2 references
    public void Engage()
    {
        boxCollider.enabled = false;

        foreach (GameObject go in gameObjects)
        {
            go.SetActive(true);
        }

        GameStateManager.instance.GoIdle();
    }
}
```

```
if (other.gameObject.CompareTag ("losecube"))
{
    playbackState = 3;
    FMODUnity.RuntimeManager.PlayOneShot(lossSound);
    musicEv.setParameterByName("Change", playbackState);
    GameStateManager.instance.lossExtra.Engage();
}
```

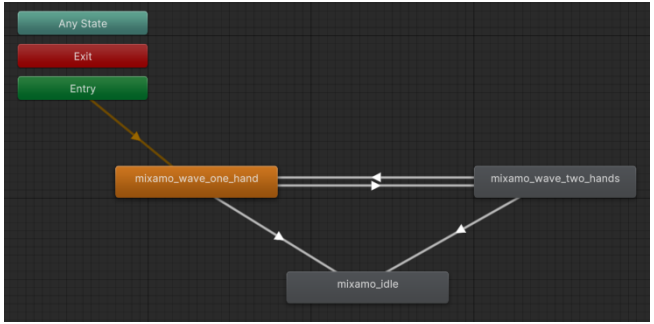
In summary, the updated code in the player controller triggers the one-shot SFX audio, activates the particle effect, disables a box collider to prevent multiple triggers, sets all GameObjects in the list to active (the gravestones, ballista and a collision blocker), and for completeness transitions "Nerk's Friend's" animation to an idle state, stopping her from waving and making her stand idle.

Game Audio: Final Project: FMOD Creation & Integration
“Nerk’s Friend” (Code Triggered FMOD 3D Timeline)



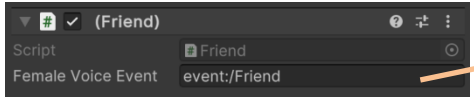
I navigated the Mixamo animation pipeline to integrate animations with my LowPoly Characters successfully.

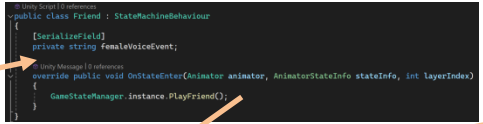
Admittedly, the process is somewhat complex.

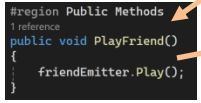


Eleven Labs

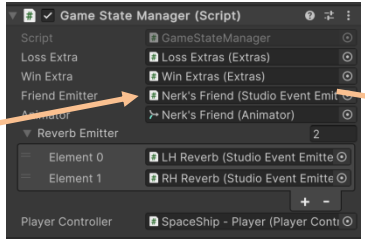
I utilized my license for ElevenLabs to create an AI-powered female voice for the lines above.



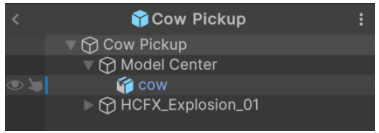


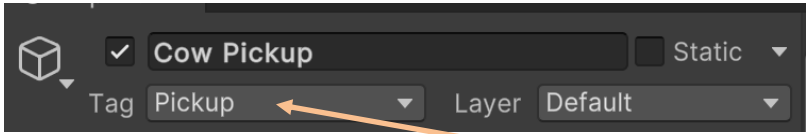


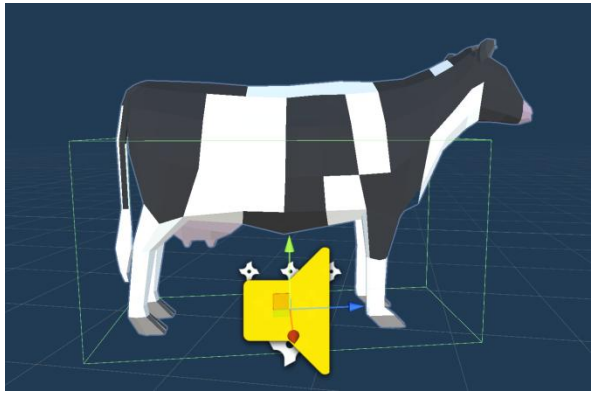
Essentially, whenever the animation switches to a two-handed wave, a call is made to the GameStateManager, triggering the FMOD Event Emitter on the "Nerk's Friend" GameObject.



Game Audio: Final Project: Complete the Final Project: Extras: Cow Pickup



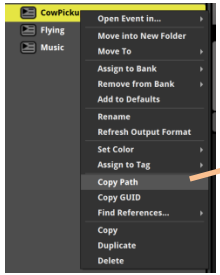


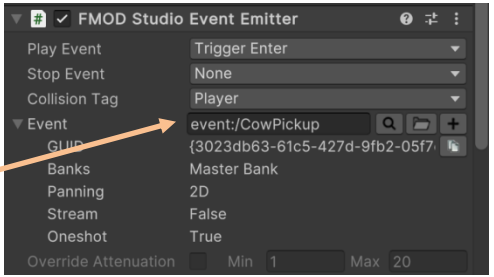


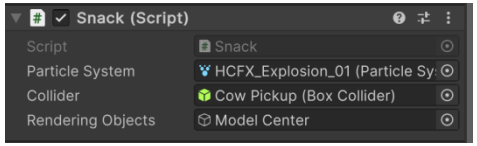
```
if (other.gameObject.CompareTag ("Pickup"))
{
    other.GetComponent<Snack>().RemoveSnack();
}
```

I created a new prefab of a cow that has the same tag as the project included code so that all I had to do was trigger my new code from the original code in the “PlayerController” class.

I won't delve into the specifics of the code below, but in essence, when the RemoveSnack() method is invoked, it disables the collider, turns off the renderer for the cow model, and then initiates a particle effect that coincides with the GameObject's destruction.







The path of the new event timeline needed to align with the specifications set in the "play event" ‘trigger enter’ within the FMOD Studio Event Emitter component.

Whenever there are changes of this nature, Unity provides a scaner for updates following a complete build in FMOD. This procedure is quite straightforward and is facilitated by a pop-up user interface within the project.

```
using System.Collections;
using UnityEngine;

#pragma warning disable CS0108
Unity Script | 1 reference
public class Snack : MonoBehaviour
{
    [SerializeField]
    private ParticleSystem particleSystem;
    [SerializeField]
    private BoxCollider collider;
    [SerializeField]
    private GameObject renderingObjects;

    1 reference
    public void RemoveSnack()
    {
        collider.enabled = false;
        renderingObjects.SetActive(false);
        particleSystem.Play();
        StartCoroutine(WaitForParticleSystem());
    }

    1 reference
    private IEnumerator WaitForParticleSystem()
    {
        yield return new WaitForSeconds(particleSystem.main.duration + 0.1f);

        if (!particleSystem.isPlaying)
        {
            Destroy(gameObject);
        }
    }
}
```

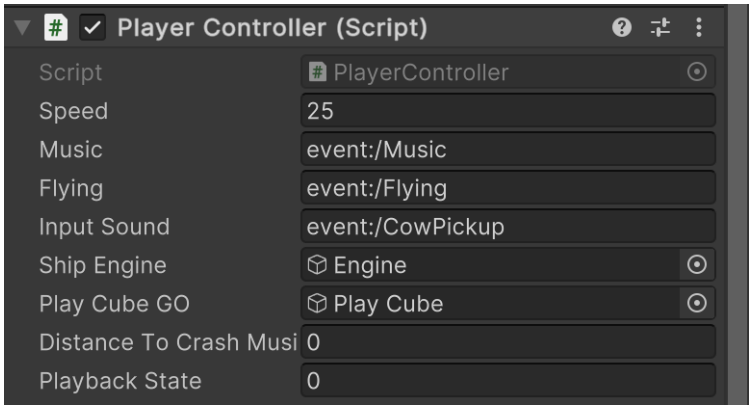
Game Audio: Final Project: Extras: “Initial” Music Work Product

```
// DSM
[SerializeField]
private GameObject shipEngine;
[SerializeField]
private GameObject playCubeGO;
private float tempVel = 0f;
public float distanceToCrashMusic = 0f;
private FMOD.Studio.PLAYBACK_STATE play_state;
public int playbackState = 0;
```

I added and modified the location of a few variables.

```
musicEv = FMODUnity.RuntimeManager.CreateInstance(music);
flyingEv = FMODUnity.RuntimeManager.CreateInstance(flying);
flyingEv.setParameterByName("speed", 0);
flyingEv.start();
```

Here I’m making sure the flyingEv is setup properly and the initial speed is set to 0 and started.




```
FMODUnity.EventReference Flying;
public string flying = "event:/Flying";
FMOD.Studio.EventInstance flyingEv;
```

Modified the name of the EventInstance from rolling to flying for clarity. FYI the EventReference isn’t used nor needed.

```
// DSM
shipEngine.transform.localScale = new Vector3((tempVel / 100f), (tempVel / 100f), (tempVel / 100f));
distanceToCrashMusic = Vector3.Distance(this.transform.position, playCubeGO.transform.position);
musicEv.setParameterByName("InitPlayDistance", distanceToCrashMusic);
if (distanceToCrashMusic <= 9.37)
{
    if (playbackState != 0)
    {
        playbackState = 0;
        musicEv.setParameterByName("Change", playbackState);
        musicEv.start();
    }
}
```

Inside the FixedUpdate() method, I inserted code that essentially utilizes the speed, stored in the variable ‘**tempVel**’ for temporary velocity, to determine the size of the particle effect underneath the spaceship. This code also assigns the distance to a new variable ‘**distanceToCrashSite**’ that I pass to FMOD. I introduced another variable named ‘**playbackState**’ to store the value corresponding to the "change" parameter in FMOD. This setup allows me to revert to the "initial" music from the "change" music, contingent on the FMOD “change” logic, the ‘playbackState’ and the ship's distance from the crash site, which is defined by the location of the original play cube.

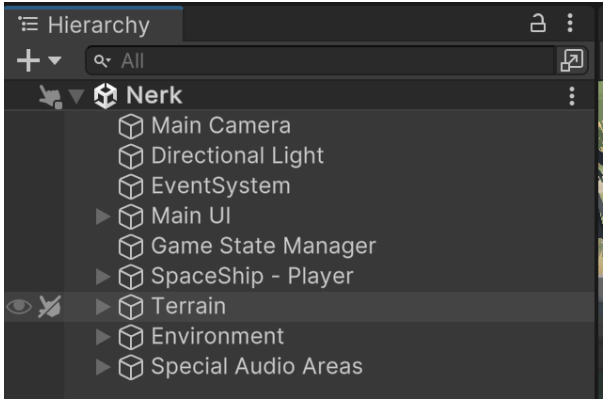
```
if (other.gameObject.CompareTag ("ChangeCube"))
{
    playbackState = 1;
    musicEv.setParameterByName("Change", playbackState);
}
```

Here is the slightly modified “ChangeCube” logic but with the playbackState being utilized.

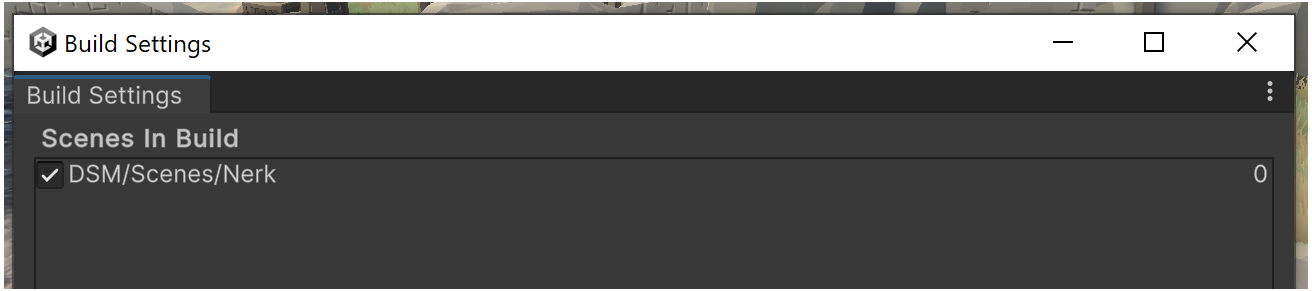
There were a few other items I changed both in code and the scene’s hierarchy but nothing worth noting as ultra important.

Game Audio: Final Project: Extras: Game's Hierarchy

Cleaned up the Game's Hierarchy and Created a New Scene (from previous)

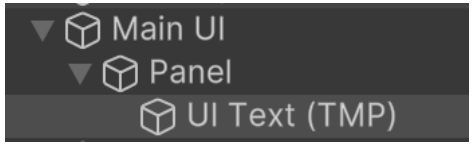


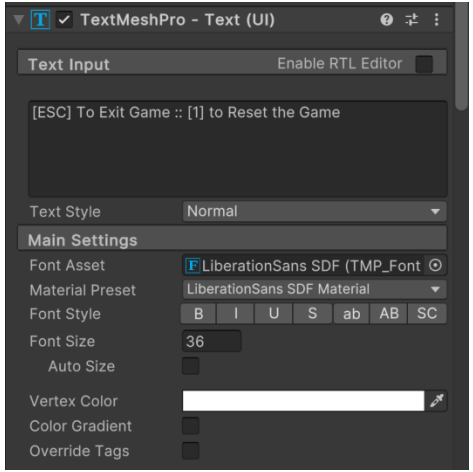
The game's hierarchy was restructured according to the collections of GameObjects and their overall purpose.

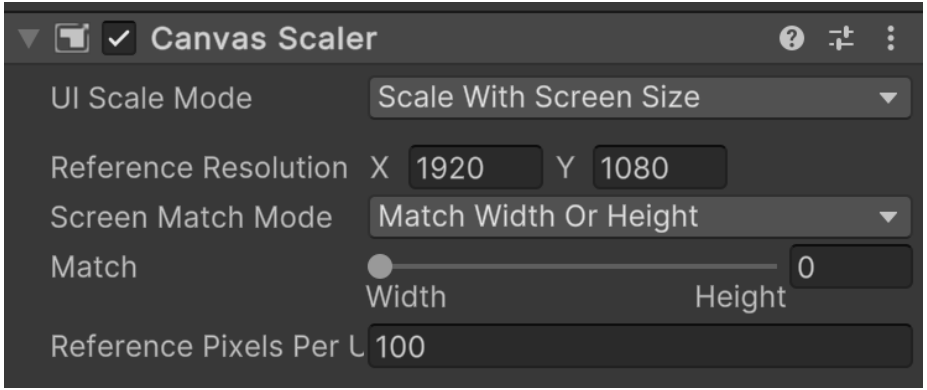


Made sure the correct scene was included in the build settings for appropriate build procedures.

Game Audio: Final Project: Extras: Very Simple UI



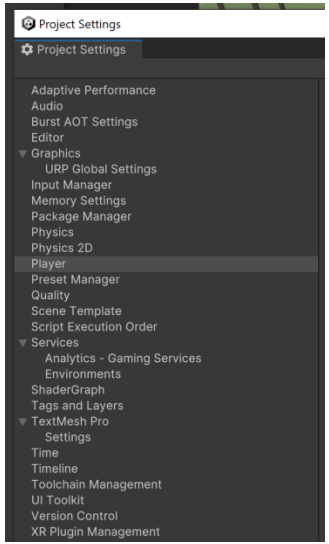




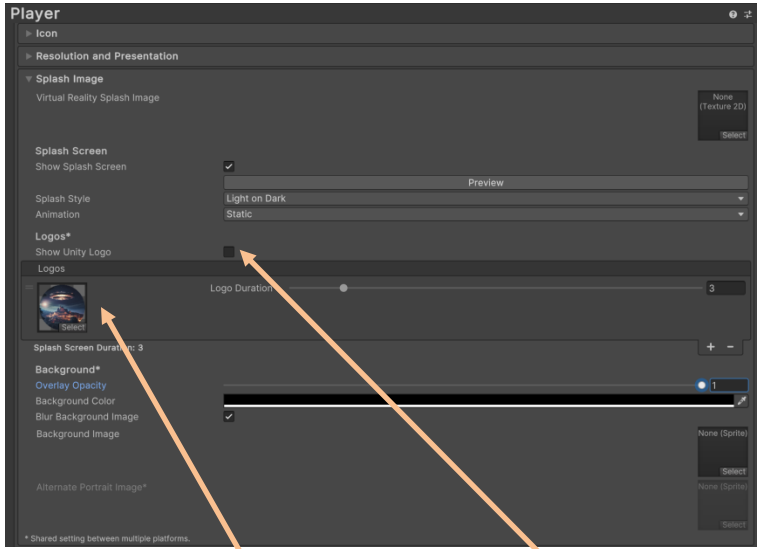
The UI scales based on the runtime resolution. The "Reference Resolution" serves precisely as its name suggests: it's what Unity uses as a benchmark when I constructed the very simple UI, ensuring it knows how and what to scale accurately.

Game Audio: Final Project: Extras: Splash Screen

Please note that a "Pro" license for Unity is required to perform some of the actions described here.

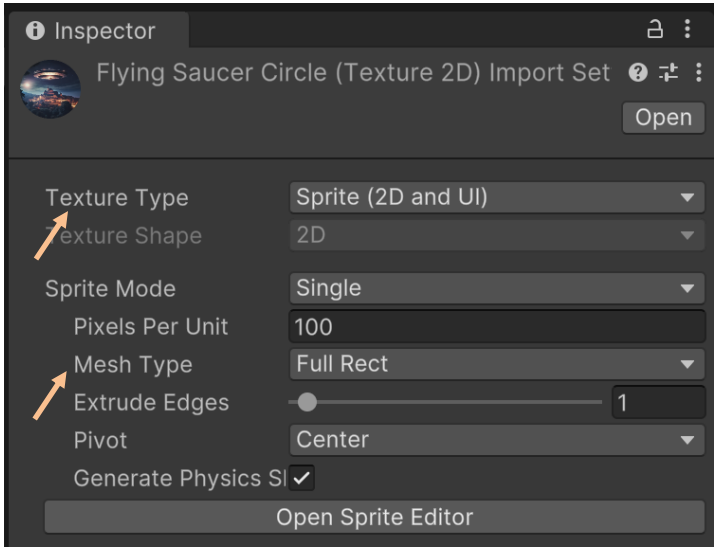


Edit to “Project Settings” then “Player”



Replace this area with your loading image.

Disabling the "Unity Logo" option is likely only available with "Pro" licenses.



Set the logo image's "Texture Type" to "Sprite (2D and UI)," and under "Sprite Mode," the "Mesh Type" should be adjusted to "Full Rect."