Intro to Unity Part 1

Short Outline

1. **Overview of what the workshop will cover**
   - We will be making a simple 2D game where the player directs a 2D sprite around the screen to collect “food” which adds points to an on-screen score counter.
2. **What is Unity?**
3. **Concepts to understand**
4. **How to get Unity**
5. **Creating a new project**
6. **Tour of Unity interface**
7. **Creating a square**
8. **Script Machines and Graphs**
9. **Adding Barriers**
10. **Adding Collectible**
11. **Writing Collectible Script**
12. **Creating a Random Spawn for Collectible**
13. **Audio portion (2 parts)**

   - Part 1a: Introduction to Unity's Audio System:
     - Understanding audio sources, listeners, and spatial audio.
     - Working with audio clips and mixing audio in Unity.
     - Implementing background music, sound effects, and ambient audio.
       - Include mono, stereo and ambisonic mixing examples.

   - Part 1b: Audio Integration and Interactivity with Visual Scripting:
     - Triggering audio based on in-game events.
       - Sounds FX based on collisions and proximity.
     - Creating dynamic audio responses to player actions.
       - Use player motion to generate sound.
     - Syncing audio with animations and gameplay elements.

   - Part 2a: C# Scripting for Audio in Unity:

    - Recap on Unity's scripting language (C#) based on Drew's session.
    - Scripting audio-related functionalities: accessing the mixer and effects parameters via Scripting. Addressing game objects, attributes using scripting.
    - Scripting game mechanics and behaviors using user input.

- Part 2b: Advanced Techniques (if we have time)
    - Implementing audio filters and effects.
    - Utilizing third-party audio plugins.
    - Optimizing audio performance in Unity projects.

Full Instructions

1. **Overview**
    - **a.** We will be making a simple 2D game where the player directs a 2D sprite around the screen to collect "food" which adds points to an on-screen score counter.

2. **What is Unity?**
    - a. Unity is a game engine that can be used to create 2D or 3D games, VR experiences, training simulations, interactive movies, animations, etc.
    - b. It is free to use for most people

3. **How to get Unity**
    - a. Go to Unity's website to install Unity Hub
        - i. unity.com/download
        - ii. Unity Hub is lightweight launcher program
    - b. From Unity Hub install a version of the actual Unity application and set up your Unity account
        - i. Since Unity is regularly updated, there are many versions to pick from that have different features

1. Choose one of the LTS (Long Term Support) installs as those are the most stable builds and receive support for longer
2. Make sure you pick a version of Unity that is 2021.1 or newer

4. **Important Concepts for this Lesson**
   a. Unity offers two different methods for adding functionality to games:
      i. C# programming language
      ii. Visual Scripting
   b. We will be learning about Visual Scripting today
   c. Everything you work with in Unity can be boiled down to:
      i. Objects
         1. The physical things that make up the virtual world you create
         2. Objects are made up of components: pre-made scripts
      ii. Scripts
         1. A set of instructions that tell Objects what to do
      iii. Show illustrations of objects and scripts to explain these concepts

5. **Creating a New Project**
   a. New Project → 2D → Project Name → Location → Create project
   b. Like all other digital media production processes, good organization is vital to a good workflow

6. **Tour of the Unity Interface**
   a. Project tab
      i. Everything in the project tab are assets you can use in your game, similar to video editors
      ii. Scenes
         1. I find it helpful to think of scenes as levels in a game
   b. Hierarchy

      i. Shows every object in your scene
      ii. Camera is where the player will see from when your game is completed
   c. Scene view
      i. Allows you to freely move around the scene and manipulate the objects in it
   d. Play view
      i. Shows the view from the camera object
      ii. When you press play to test the scene, you will be able to play the game from this view
   e. Inspector
      i. Allows you to look at the components of objects
         1. Remember that components are just scripts that tell the object what to do
      ii. Create an empty game object to show that objects without components will do nothing
         1. The only component that all objects have is a transform
         2. The transform describes the location, rotation, and size of an object (where and how it exists in physical space)

7. **Creating a Square**
   a. Add a <u>square</u> in the Hierarchy
   b. Name it **Player**
   c. Set its position to **0**
   d. Go over its various components and describe what they do
   e. Look at the square through the Play view

8. **Adding a Script Machine and Creating a Graph**
   a. Add the <u>Script Machine</u> component to the Player
      i. A machine is a component that runs the instructions from a script

b. Create an assets folder in the Project tab and call it **VisualScripts**

c. Click <u>New</u> on the Script Machine

d. Name the new graph **Movement** and save it to the new folder

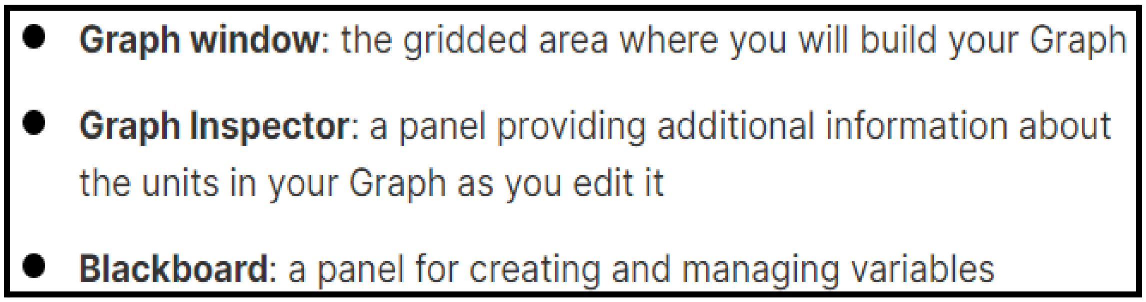
  i. A graph is the name for the scripts that scripts machines run

e. Select <u>Edit Graph</u> in the Inspector

f. Show how to dock windows in Unity and dock the Script Graph window to the bottom of Unity

g. Separate Graph Inspector from Blackboard using this button:

h. There are three parts of this window:



- **Graph window**: the gridded area where you will build your Graph
- **Graph Inspector**: a panel providing additional information about the units in your Graph as you edit it
- **Blackboard**: a panel for creating and managing variables

9. Writing a Graph/Script

a. Start

  i. Is a method that is triggered once at the beginning of the scene

b. Update

  i. Is a method that is triggered on each screen update (or frame)

  ii. Basically, it runs continuously

c. Note that both methods have a green banner across the top, signaling that they are the triggering event in the flowchart

d. Select and delete <u>Start</u>

e. Right click the graph window and search for <u>Input Get Key</u> in the fuzzy finder

  i. Place <u>Input Get Key</u> under <u>On Update</u>

  ii. This function listens for the user to press down a specific key and outputs a Boolean value

    1. Booleans can either be true or false

        2. When the user is not pressing down the designated key the function outputs false, vice versa when the user is pressing the key

    iii. Select Up Arrow for the chosen key

f. Search and add **If** to the graph

    i. If takes a Boolean value and decides what to do based upon if it is true or false

    ii. If we connect the flow to If we're telling the flowchart to move to that function next

        1. On Update → If

    iii. And if we connect the pink/purple dots, that will pass the Boolean value Input Get Key is creating to If which will decide what happens next

        1. It branches the flow of the graph

g. Search and add **Transform: Translate (X, Y, Z)**

    i. If: True → Translate

    ii. We don't want anything to happen if the Get Key is false so we don't connect it to anything

    iii. Set the Y to **0.1**

    iv. Translate changes the object's position every frame by a certain amount, essentially teleporting it

h. Press play to test out the scene

    i. The square moves and goes out of view

    ii. Next we'll want to add movement in the other directions and add barriers to keep the square on screen

10. Adding Barriers

a. Create another square
    i. Name it **Wall**
    ii. Set its position to **0, 5, 0**
    iii. Set its scale to **16, 1, 1**
    iv. Set its color to gray
    v. Duplicate it and reverse its position

b. Duplicate the second wall
    i. Change its scale to **1, 10, 1**
    ii. Change its position to **8, 0, 0**
    iii. Duplicate this wall and reverse its position

c. Now the walls need something to give them hardness, otherwise the player will just move through them

d. Select all of the walls and add a Box Collider 2D
    i. Colliders allow objects to collide with each other and physically interact

e. Add another Box Collider 2D to the Player

11. Adding Additional Movement

a. In order for some of the later systems we want to add to work, we'll want to make use of Unity's physics system (it's also good practice!)

b. Add a Rigidbody 2D to the Player
    i. This allows the object to interact with the physics system in Unity
    ii. All of these different elements impact how the object moves, feel free to mess around with them once you have your input system finished
    iii. Set:
        1. Mass to **0.1**
        2. Linear Drag to **2**

        3. <u>Gravity Scale</u> to **0**
        4. <u>Constraints</u> → <u>Freeze Rotation</u>

   c. Go back to the Player’s graph and replace:
      i. <u>On Update</u> with <u>On Fixed Update</u>
         1. This is very similar to Update but works better with the physics system

      ii. <u>Translate</u> with <u>Rigidbody 2D Add Force</u>
         1. Set it to **0,1**
         2. This method function uses the physics system to push an object with a Rigidbody as opposed to teleporting it every update
   d. Select the four units and duplicate them three times
      i. Change the duplicates to be:
         1. <u>Down Arrow</u> → **0, -1**
         2. <u>Right Arrow</u> → **1, 0**
         3. <u>Left Arrow</u> → **-1, 0**
   e. Press play to test out the scene

12. Cleaning Up Movement

   a. This code feels a bit clunky. I bet there’s a way to streamline some of it.
   b. An important concept to understand is that this (x, y) coordinate in Unity are called <u>Vector 2</u>
      i. What each group of nodes is doing is its pushing the object in that direction by a certain amount, in this case by 1
   c. Delete everything but the top <u>On Fixed Update</u> and <u>Add Force</u>
      i. Connect <u>On Fixed Update</u> → <u>Add Force</u>
   d. Add <u>Input Get Axis</u>

        i. This method listens for the typical movement keys (arrows and WASD) on a specific axis, and if they're pressed it will create a value for them of either 1, 0, or -1
        ii. Type **Horizontal** into Input Get Axis (Axis Name)
            1. If we press the left or right arrows (A or D) it will create a value of -1 for left and 1 for right (0 for neither)
            2. This replaces the previous system where we checked for an a key and then set a number value
    e. Duplicate Input Get Axis and type **Vertical**

    f. Add <u>Vector 2 Create (X, Y)</u>
        i. Connect:
            1. Horizontal → X
            2. Vertical → Y
            3. Vector 2 Create → Add Force (Force)
        ii. What this does is merge the horizontal and vertical values together into one coordinate and then feeds it into the Add Force method
    g. That's all we need!

13. Adding Collectible

- a. We now want to add something for the player square to pick up
- b. Create a <u>circle</u> in the Hierarchy and name it **Collectible**
    - i. Set the the position to **-4, 0**
    - ii. Add:
        1. <u>Circle Collider 2D</u>
        2. <u>Script Machine</u>
    - iii. Create a new graph and name it **Collectible**

14. Writing Collectible Script

a. Edit the Collectible graph
b. The triggering event we want for this object will not be either On Update or On Start, so we will delete both
c. Add:
    i. <u>On Collision Enter 2D</u>
    ii. <u>Game Object Compare Tag</u>
    iii. <u>If</u>
d. On Collision Enter is triggered anytime this object has a collision with another object that has a collider component
    i. It is called once per collision and at the beginning of the collision
e. Connect:
    i. On Collision Enter (Collider) → Compare Tag (Game Object)
    ii. Every object has a tag, so what this function will do is check the tag of the game object that collided with the Collectible (that's why we have to pass on the collider information)
f. Type **Player** into the Tag portion of Compare Tag
g. Connect:
    i. Compare Tag (Boolean) → If (Boolean)
    ii. Like the Get Key method earlier, the Compare Tag function will produce a true or false result and the If will decide what to do with each result
h. Go to the Player object and set its tag to <u>Player</u>
i. What do we want to happen when the Player collides with the Collectible?

15. Creating a Random Spawn for Collectible

a. We are going to want to have the collectible move to a new position after being "picked up" by the Player

        i. To do this, we'll want to have the game pick a random point within the play area

b. Add the Random Range node twice
    i. The first one will be to pick a random X and the second will be to pick a random Y
    ii. For the X one, type:
        1. Min: **-7**
        2. Max: **7**
    iii. For the Y one, type:
        1. Min: **-3.5**
        2. Max: **3.5**
    iv. Both of these nodes will pick a random number between the minimum and maximum we just set
        1. We chose these values by looking at the size of the play area
c. Connect the Random Range nodes together
d. Now, we'll want to pass these new numbers onto our Collectible object and tell it to teleport to the new location
e. Add:
    i. Vector 2 Create (X, Y)
    ii. Transform Set Position
f. Connect:
    i. If (True) → Set Position
g. What Vector 2 Create does is take two numbers and creates a Vector 2 with them, i.e. creating a coordinate with them
    i. We'll connect the blue outputs from the two Random Range nodes to the X and Y inputs of the Create node (make sure the X goes to X and the Y to Y)

h. Now that we have a new coordinate, we'll connect that to the bottom input of Set Position which will instruct the object to set its position to that new coordinate
i. Play the scene

Intro to Unity Part 2

Set-up

- Download the example sounds from Google Drive
- Add them to your Unity project assets folder
    - I like to create a subdirectory called "audio" for good hygiene.

Build out in Unity

- Create Generic Empty GameObject to act as a **"SoundManager"**
    - Helps to keep everything in one place!

- Add **Script Machine**
- Add a new script called **"Collectible".**
    - The script name needs to match between object Script Graphs (SG).

- Search and add an **Aot List** (for coin sound effects; it's an array list) called "Collectible Sounds".

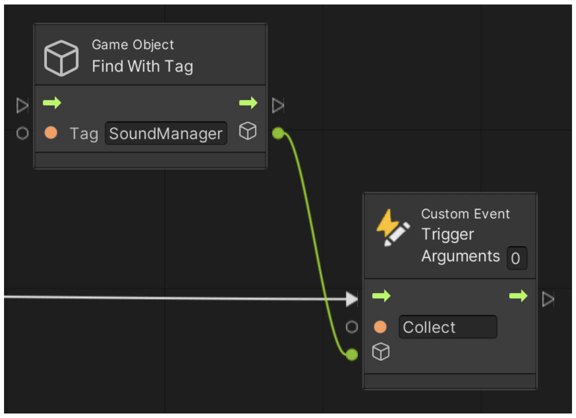
- Add three variables to your Aot List and set each to an **Audio Clip.**
- Drag and drop your SoundManager objects Variables (Aot List).

- Add an Audio Source to the SoundManager. This is required to playback the audio clips (you'll need to ensure Unity audio playback properties are configured to work with your current device).

- Create a "SoundManager" tag for visual scripting custom trigger events and apply it to your SoundManager object.

- Enter the SG for Collectible and add a **Sequence node** to leverage Drew's trigger event for one-to-many relationships like triggering sounds and debug logging.

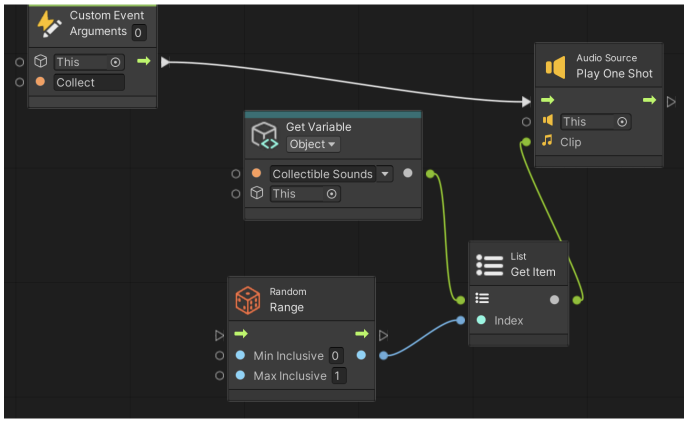
- Add a Game Object ("Find with Tag") and Trigger Custom Event.
    - Name the former, "SoundManager"
    - Name the latter, "Collect".



- Finally, build out the SoundManager Script Graph.
    - Add a Custom Event
        - Configure to receive action via the Collectible script.
        - The final output stage is an **Audio Source (Play One Shot).**
        - **Select a single audio clip to test**

How can we develop it?

- Check to make sure you've added your audio files to your SoundManager objects Variables (Aot List).
- Drag and drop the list of sounds onto the VSG area.



How do we leverage randomness?

- Add List Node (“Get Item”)
- Add a Random node with a range of 0 to 1 to randomly select from multiple coin collection sounds.

Then, if we have time.

- Add B format audio source to a Generic Empty GameObject called Ambience.
- We can demo the difference between 2D/3D by adding an audio listener to the player and reverb zone.
    - It doesn’t make sense to have spatialization in a 2D game unless it is an important part of the game’s navigation system.

Intro to Unity Part 3

<u>**Outline**</u>

1. Intro to CREATE
2. Overview of what the workshop will cover
3. Overview of game made in last workshop
4. Clean up the collision
5. Change player input
6. Change collectible transparency Pt. 1
7. Change collectible transparency Pt. 2
8. Change collectible transparency Pt. 3
9. Add game manager
10. Update score from collectible
11. Add obstacle (Cut for time)
12. Create prefab (Cut for time)

<u>**Full Instructions**</u>

4. Clean up the collision

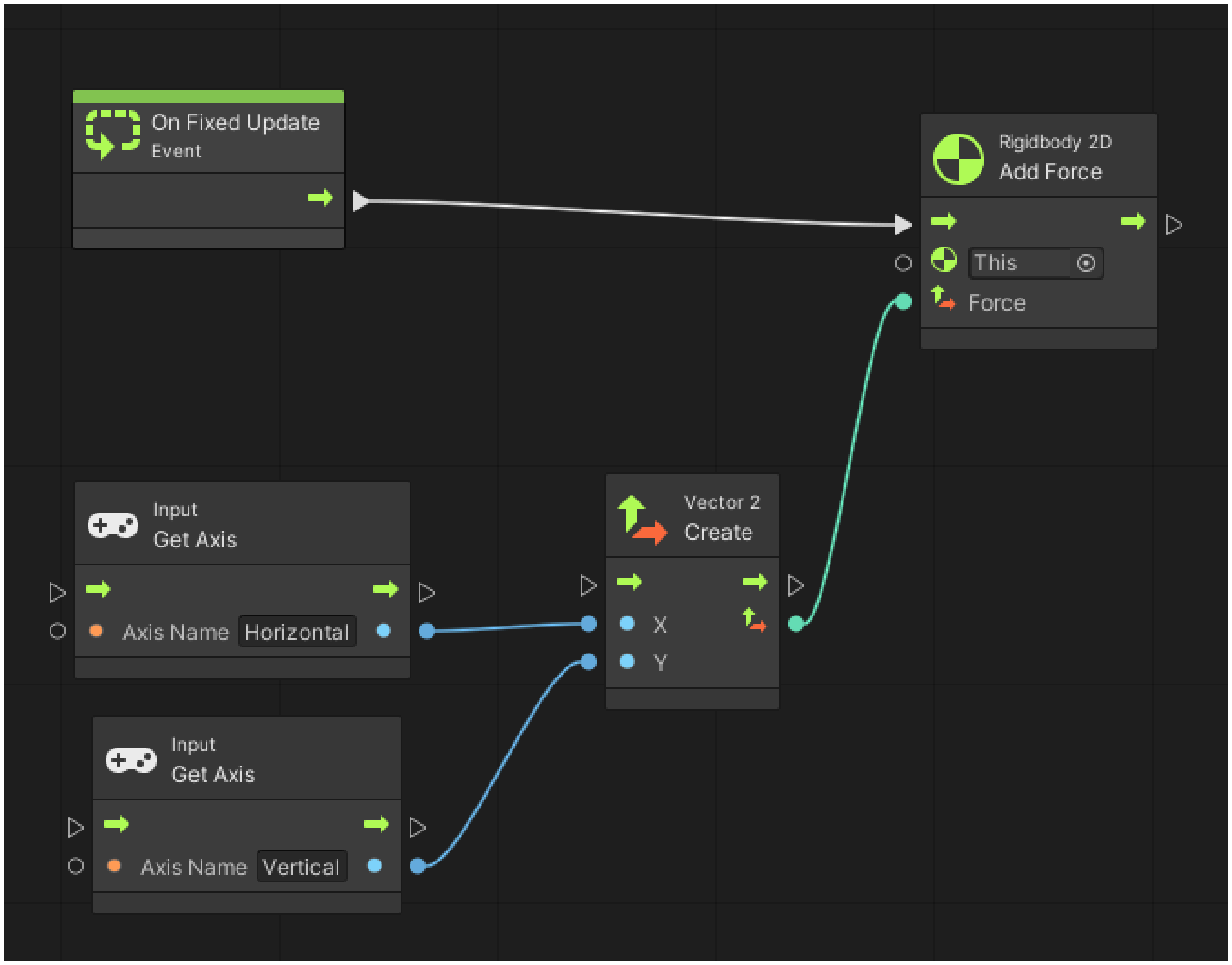
A. This might play better if the Player didn't stop each time it hits the Collectible object. We can fix this by making the Collectible a trigger
B. On the Collectible's Circle Collider, check **Is Trigger**
C. On the Collectible's Graph, replace **On Collision Enter** with **On Trigger Enter 2D**
D. Making a collider a trigger makes it so that it won't physically interact with other object's colliders (i.e. exert force against them) but it will still register collisions

5. Change player input

A. This will streamline the player input system
B. Delete three of the four rows of player input
C. Connect the last **On Fixed Update** -> **Rigidbody 2D: Add Force**
D. Add *two* **Input: Get Axis** nodes
E. Add **Vector 2: Create (X, Y)**
F. Label one axis <u>Horizontal</u> and one <u>Vertical</u>
G. Connect **Input: Get Axis** -> **Vector 2: Create**
    a. <u>Horizontal</u> -> X
    b. <u>Vertical</u> -> Y

H. Connect **Vector 2: Create** -> **Rigidbody 2D: Add Force**

I. Should look like this:



6. Change collectible transparency Pt. 1

A. To add some challenge, we're going to make the collectible only visible when the player object is close to it.

B. We'll begin by adding back the **On Start** node

C. Add **Sprite Renderer: Set Color**

D. Connect **On Start** -> **Sprite Renderer: Set Color**

E. Clock on the color bar on **Sprite Renderer: Set Color** and set the Alpha (A) value to 0

F. This step isn't entirely necessary but it is good practice

G. What we've done is set the collectible object's transparency to 0 at the start of the game

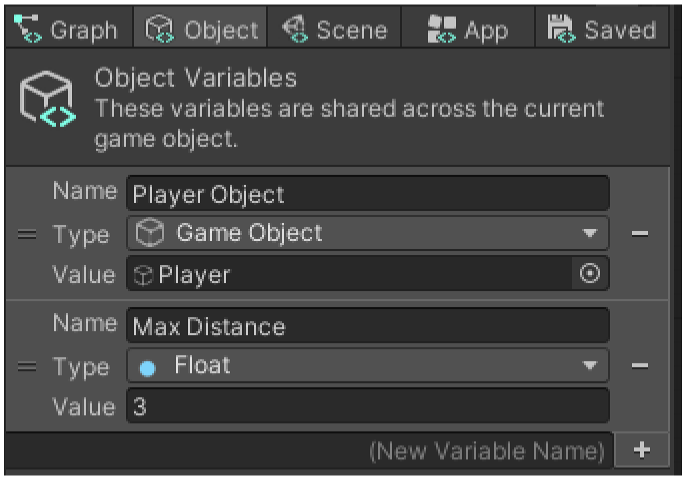
7. Change collectible transparency Pt. 2

A. To actually make the transparency change over time, we'll need to use what is called a Lerp function
    a. Lerp is short of linear interpolation
    b. Lerp "eases" the transition between values over time
B. We'll start by adding in **On Update** and another **Sprite Renderer: Set Color**
    a. Connect **On Update** -> **Sprite Renderer: Set Color**
C. Now we will add **Color: Lerp**
    a. This will have two color variable inputs and a float value for combining color A and B
    b. This Lerp function uses a scale between 0 and 1
        i. 0 is all color A and 1 is all color B
    c. Set the Alpha of Color B to 0

8. Change collectible transparency Pt. 3

A. Now we need to calculate the distance between the player and the collectible and use that to feed a value into the Lerp
B. Add two Object Variables to the collectible
    a. Variable one:
        i. <u>Name:</u> Player Object
        ii. <u>Type:</u> Game Object
        iii. <u>Value:</u> Player (game object)
    b. Variable two:
        i. <u>Name:</u> Max Distance
        ii. <u>Type:</u> Float
        iii. <u>Value:</u> 3
C. Click and drag the <u>Player Object</u> variable onto the graph (this will add a pre-filled **Get Variable** node)
D. Add two **Transform: Get Position** nodes
    a. Connect one's input to the <u>Player Object</u> variable node
    b. Leave the other as set on <u>This</u>
E. Add **Vector 2: Distance (A, B)**
F. Connect the two **Transform: Get Position** nodes -> A and B in **Vector 2: Distance**
    a. This function will calculate the distance between the two position values
G. Click and drag the <u>Max Distance</u> variable onto the graph

H. Add **Mathf: Inverse Lerp** and connect:
   a. **Vector 2: Distance** -> Value
   b. Max Distance -> B
I. This will scale the distance value to be between 0 and 1, essentially meaning that if the player is 3 or closer then the collectible will be visible. Otherwise, any distance greater than 3 will result in the full B value in Lerp
J. Connect **Mathf: Inverse Lerp** -> T in **Color: Lerp**
K. The additions to the Collectible graph should look like this:

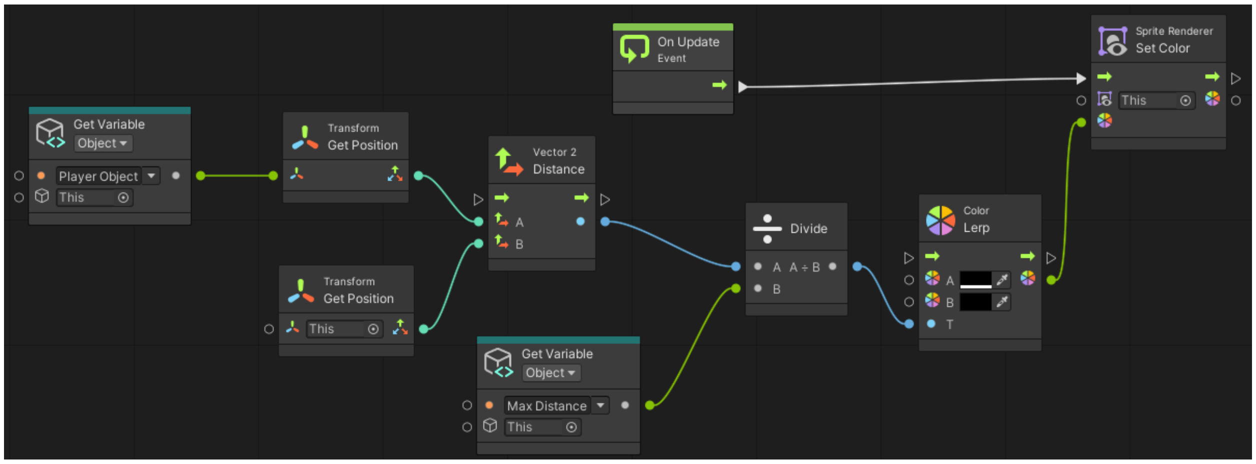


Graph | Object | Scene | App | Saved

Object Variables
These variables are shared across the current game object.

Name	Player Object
Type	Game Object
Value	Player
Name	Max Distance
Type	Float
Value	3

(New Variable Name) +



9. Add game manager

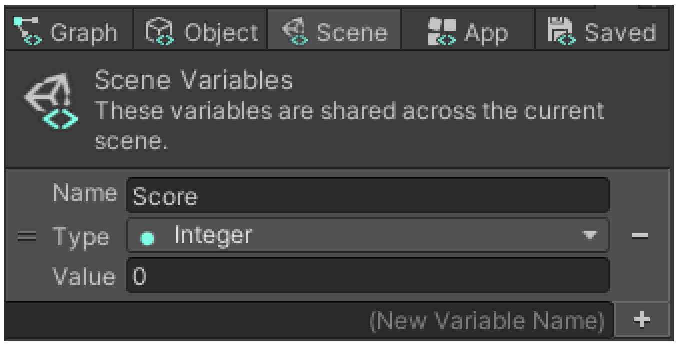
A. To help with Ricky's part, let's add an invisible score counter and game manager for keeping track of other parts of our game
B. Like the SoundManager, we'll create an empty object in the scene's hierarchy
   a. Name it GameManager
C. Add a **Script Machine** component and create a new graph named GameManager
D. Edit the graph

E. Delete **On Update**
F. Create a Scene Variable:
    a. <u>Name:</u> Score
    b. <u>Type:</u> Integer
    c. <u>Value:</u> 0
G. Add a **Custom Event** node to the graph and call it <u>AddScore</u>
H. Add **Set Scene Variable** and connect **AddScore** -> **Set Variable**
    a. Set the Variable to the <u>Score</u> variable
I. Now we'll want to add to the score each time this function is called
J. Drag the <u>Score</u> variable onto the graph
K. Add an **Integer** node and an **Add** node
L. Set the **Integer** node to 1 and connect it -> **Add**
M. Connect <u>Score</u> -> **Add**
N. Connect **Add** -> New Value input of **Set Variable**
O. Just to make sure everything works, we'll want to add a **Debug: Log** node
    a. Connect **Set Variable** -> **Debug: Log**
    b. Connect the <u>Score</u> variable node -> Message input of **Debug: Log**
    c. This will display the score variable in the Console each time it is updated

10. Update score from collectible

A. Now let's call this custom function from the collectible object each time it is picked up
B. Add another Object Variable to the Collectible script
    a. <u>Name:</u> GameManager
    b. <u>Type:</u> Game Object
    c. <u>Value:</u> GameManager (game object)
C. Add a 3rd step to the **Sequence** node
D. Add **Trigger Custom Event** and connect **Sequence** -> **Trigger Custom Event**
E. Drag the <u>Game Manager</u> variable onto the graph and connect it -> Target input of **Trigger Custom Event**
F. Type <u>AddScore</u> for the name of the **Trigger Custom Event**
G. If you would like, you can use this same method for the SoundManager. You can replace the **Find with Tag** node with a variable you created and it will do the same thing

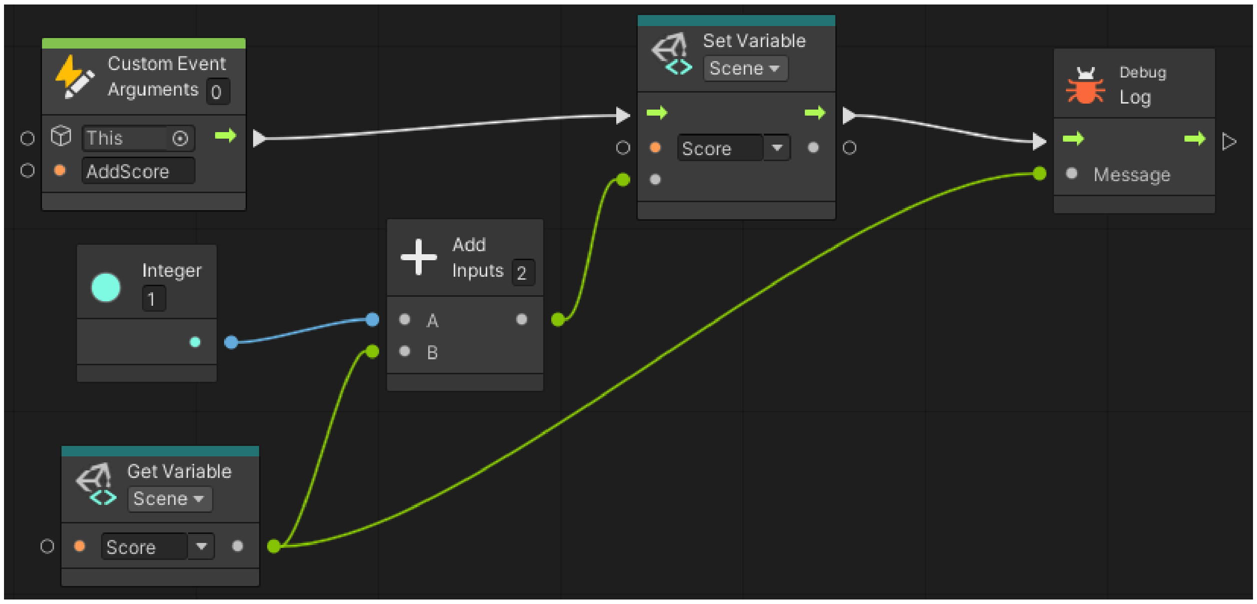
H. The GameManager script should look like this:



Scene Variables
These variables are shared across the current scene.

Name: Score
Type: Integer
Value: 0

(New Variable Name)



11. Add obstacles

A. Create the obstacle

B. Set its location randomly at the start

C. Alter its transparency based on proximity

12. Create prefab

A. Turn obstacles into prefab

B. Spawn multiple obstacles with the game manager

Intro to Unity Part 4

Objective / Stretch Goal

We want to leverage Unity spatial audio features to improve the game's accessibility for impaired learners. One approach is to use the perceived location of sound sources to guide the player's movement throughout the 2D grid to find the collectible. The player can move around freely and as they approach a collectible, they'll hear some form of music being generated from a particular direction. Once the collectible has been collected, the music audio sample will loop continuously, indirectly generating game music for the overall experience.

In this iteration we will focus on implementing two things:

1) Parent-child interactions between audio loops whereby the parent maintains synchronicity between the beat and the other audio loops (pad, melody, and bass loops)
2) Create a relationship between a score counter and the way the game music is performed.

Example of Game Mechanics

We want to use amplitude and spatial location to find the sounds. For example, a player may experience the following, “It sounds like the beat is coming from the left. It gets louder as I move to the left.”

Setup

Before we begin, let's add some better-sounding music loops to our Assets > Audio folder. There are four new audio samples:

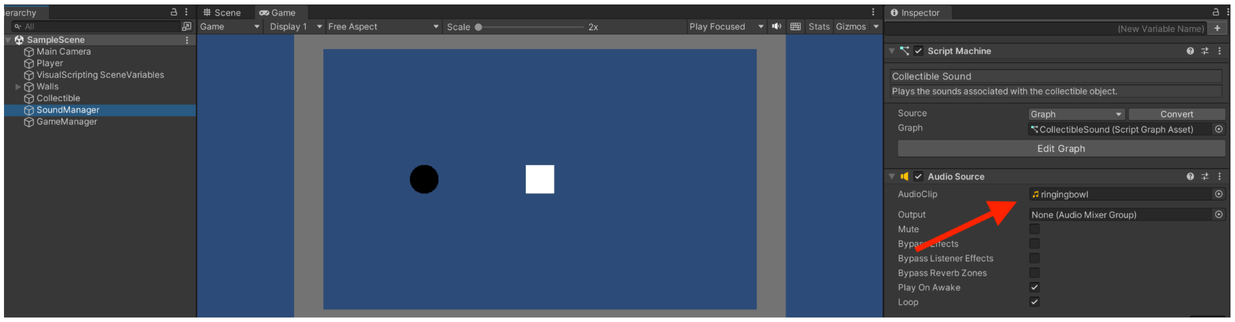
1. pad.wav
2. beat.wav
3. bass.wav
4. melody.wav

You can ignore the finalmix.wav for now. It's supplied only as a reference. These samples are purposely shorter samples to avoid retriggering problems later on.

Also, make sure the custom C# script is placed in the assets folder and that you follow the instructions at the end of this document for Unity visual scripting regeneration for the project.

Steps

Step 1: Clean up!
Let's remove the percussive background audio sample from the scene since we want to introduce some new sound elements into our game. You can delete the audio sample listed in the AudioClip field of the Audio Source under the SoundManager game object. Audio Source is required to find audio samples on the SoundManager game object, so don't delete that component from SoundManager. You can just delete the audio sample or disable playback.

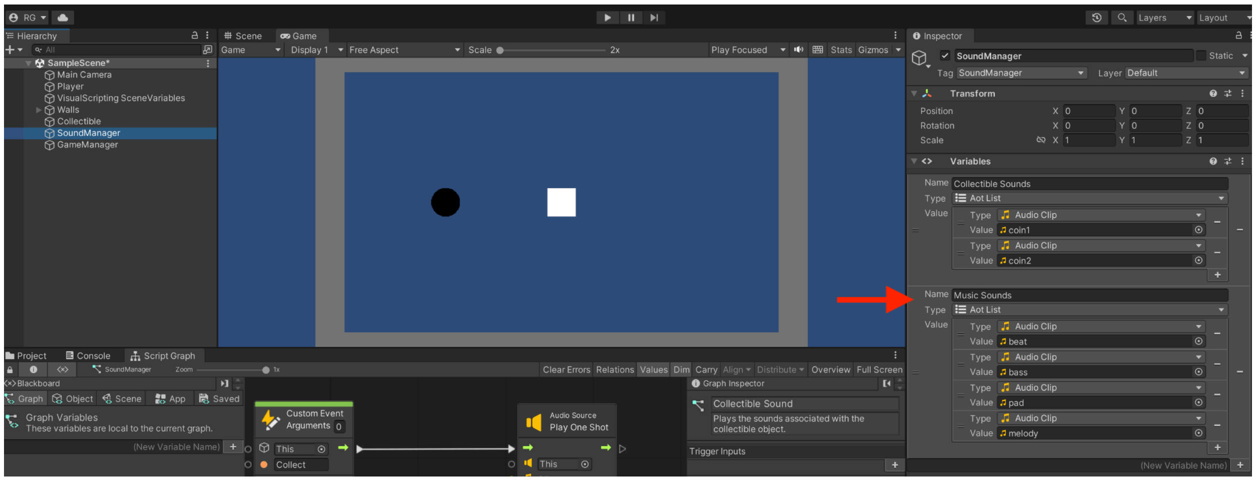


Script Machine
Collectible Sound
Plays the sounds associated with the collectible object.
Source: Graph
Graph: CollectibleSound (Script Graph Asset)

Audio Source
AudioClip: ringingbowl
Output: None (Audio Mixer Group)
Mute: ☐
Bypass Effects: ☐
Bypass Listener Effects: ☐
Bypass Reverb Zones: ☐
Play On Awake: ✓
Loop: ✓

Click on it and press delete.

Step 2: Preparing Logic to Store, Play, and Loop Music Loops
In the last workshop, we used a list to store our coin collectible sounds. We will use the same structure here but modify how they are played and called. We will start by adding another Variable to the Variables Component called "Music Loops" of type "Aot List" and add our audio samples in the order they should be played. We'll start with the beat, then the bass, then the pad, and finally the melody.



SoundManager
Tag SoundManager Layer Default

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

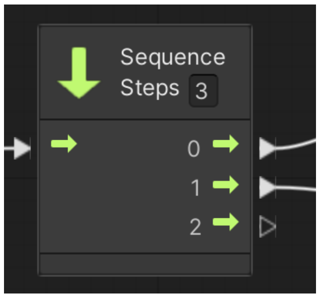
Name: Collectible Sounds
Type: Aot List
Value:
- Type: Audio Clip, Value: coin1
- Type: Audio Clip, Value: coin2

Name: Music Sounds
Type: Aot List
Value:
- Type: Audio Clip, Value: beat
- Type: Audio Clip, Value: bass
- Type: Audio Clip, Value: pad
- Type: Audio Clip, Value: melody

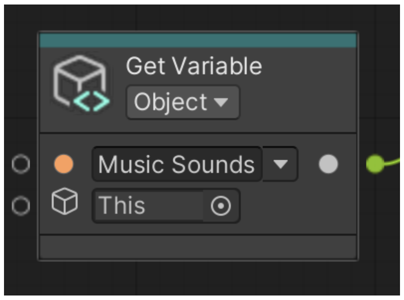
Step 3: Playback a music loop when you collect a collectible.

Let's get some logic in our script graph to get some sounds playing back and test our new music sounds Aot List. We need to do the following:

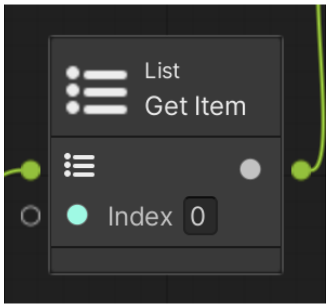
a) Use the Sequence Steps to allow one-to-many mapping of our collectible Custom Event.



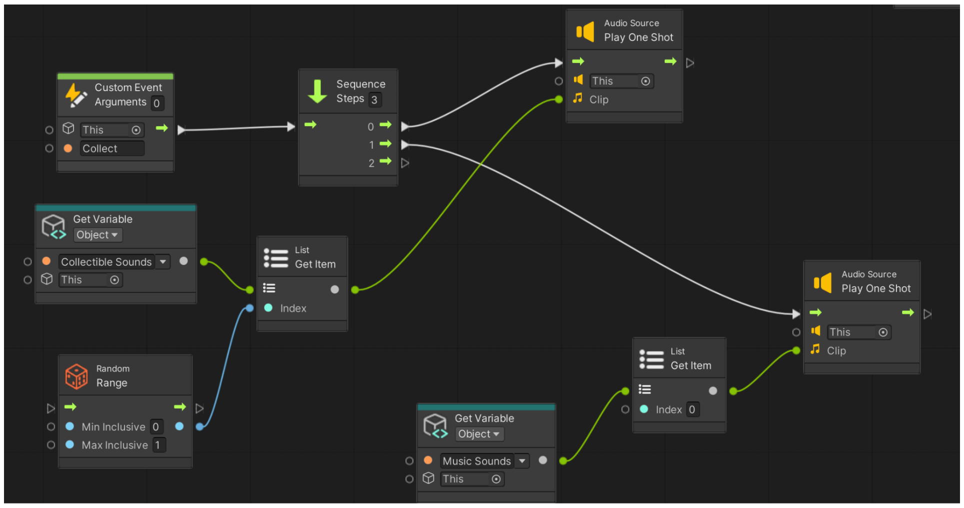
b) Use the Get Variable node to access our new variable list, "Music Sounds."



c) Use a new List node to determine which music loop to play by offsetting the index integer value.

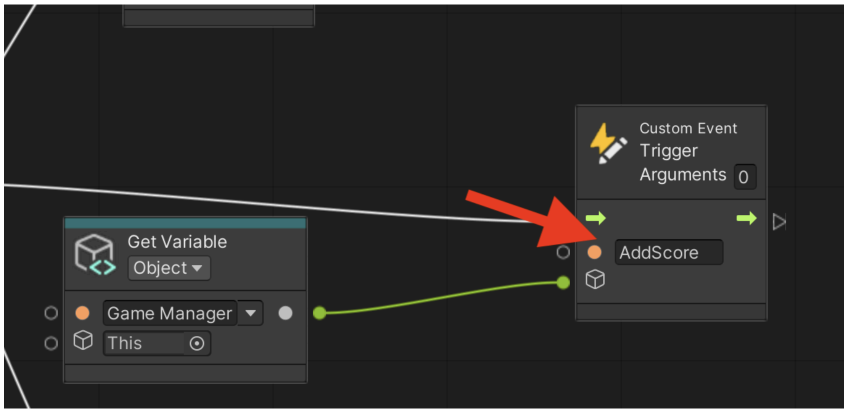


d) Create a new Audio Source (Play One Shot) node to play back the music samples.

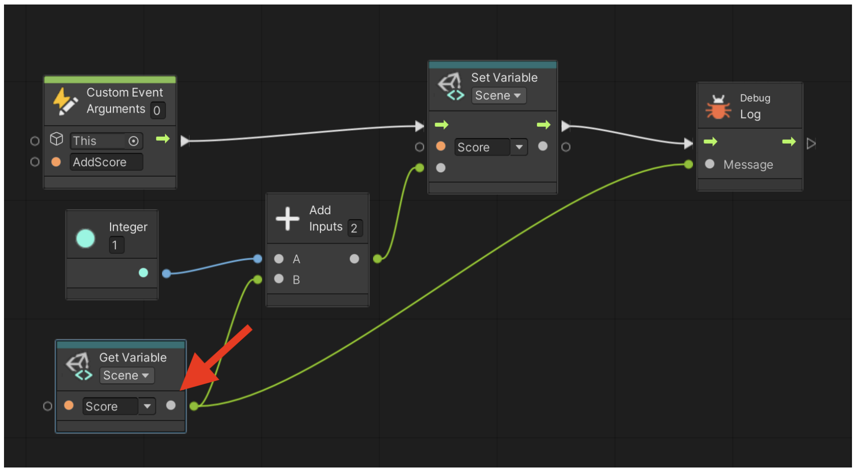


Step 4: Use the score counter to play our music loops in order.

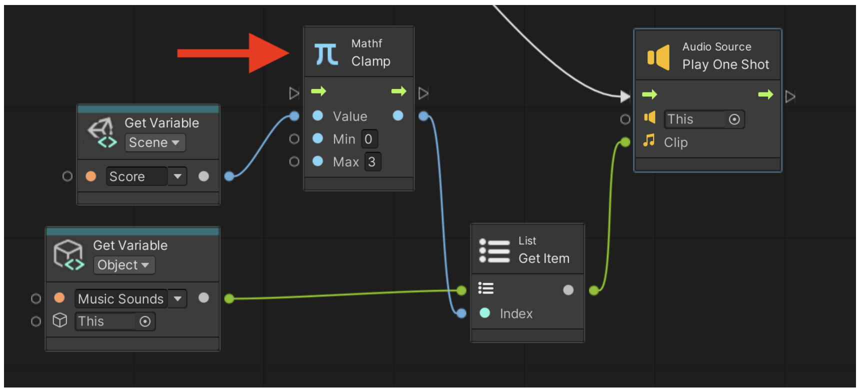
If we look at the Collectible Game Object in our Hierarchy, we notice new logic to add the cumulative player score (Add Score).



We can use the Score value to trigger each of the audio samples in order. Let’s get that working first and then implement the finer points. We can do this by simply pulling the Score value using Get Variable in our SoundManager, similar to how the score is printed in the GameManager game object in our hierarchy.



Like so! We’ll use Math’s Clamp to ensure we don’t exceed bounds (0-3).



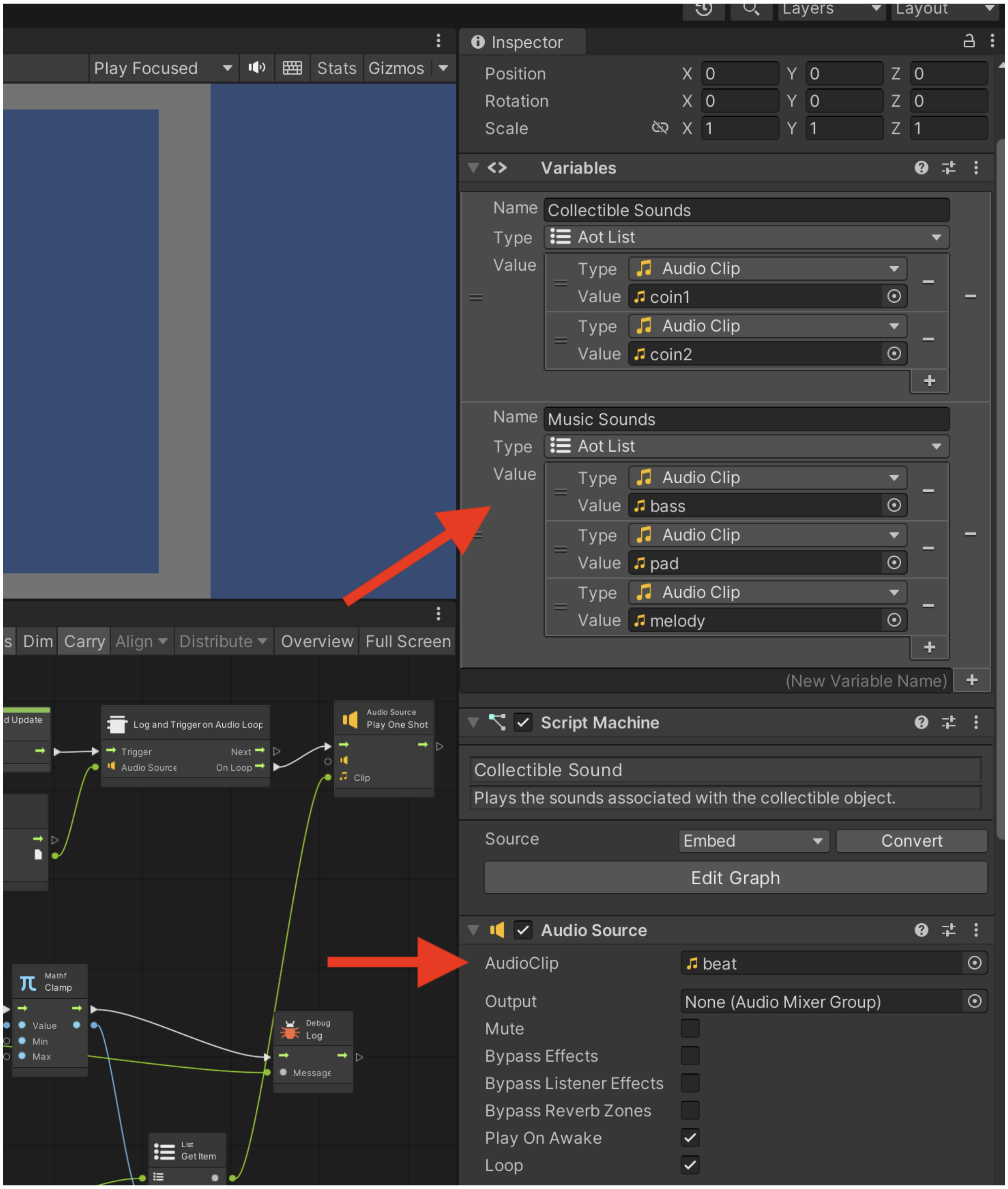
Check Point:
Run the game and note the difference in audio playback. What problem do we run into with audio playback synchronicity?

Step 5: Wait! The timing is off. Don't play the next loop until the current loop has finished.

To ensure that the next audio sample is in sync with its predecessor, we will need to incorporate a way to track when the audio clip has finished playing before triggering the next one. "Play One Shot" doesn't have an inbuilt mechanism to detect when it has finished so we'll take a different tack with playback and change out a few objects to ensure synchronization.

Step 1: Main Clock
We can use the main **beat** as our main synchronization reference point or clock. Use one AudioSource to play the first looped audio clip. This will be your master sync track, so make sure it's set to **loop.**

Add beat.wav to the Audio Source and remove it from Aot, "Music Sounds."



Inspector

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Variables

Name: Collectible Sounds
Type: Aot List
Value:
- Type: Audio Clip; Value: coin1
- Type: Audio Clip; Value: coin2

Name: Music Sounds
Type: Aot List
Value:
- Type: Audio Clip; Value: bass
- Type: Audio Clip; Value: pad
- Type: Audio Clip; Value: melody

(New Variable Name)

Script Machine

Collectible Sound
Plays the sounds associated with the collectible object.

Source: Embed
Convert
Edit Graph

Audio Source

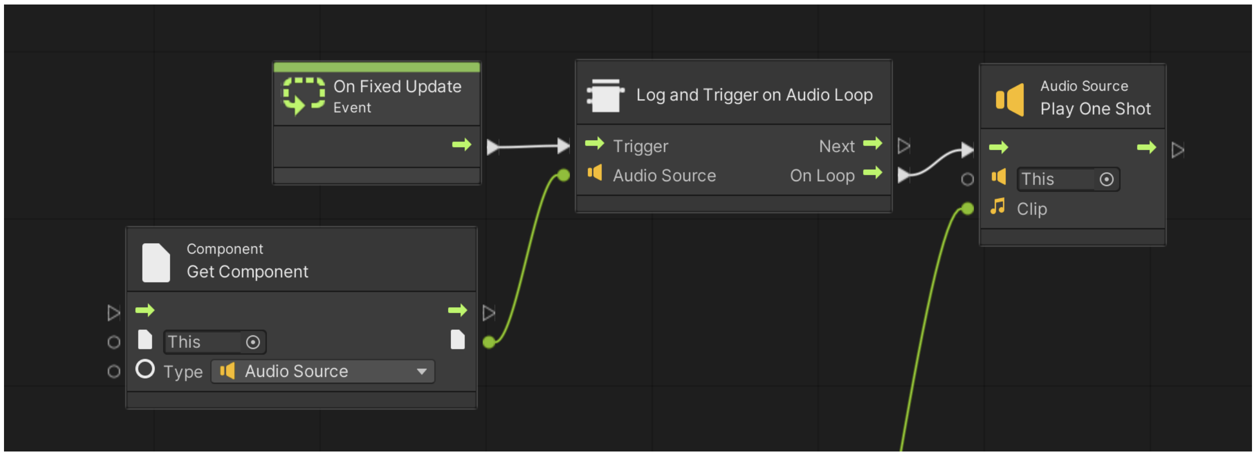
AudioClip	beat
Output	None (Audio Mixer Group)
Mute	☐
Bypass Effects	☐
Bypass Listener Effects	☐
Bypass Reverb Zones	☐
Play On Awake	✓
Loop	✓

a) We will need an AudioSource to access the sample position of our beat loop.

b) We will request the position using FixedUpdate, as used earlier in the workshop.

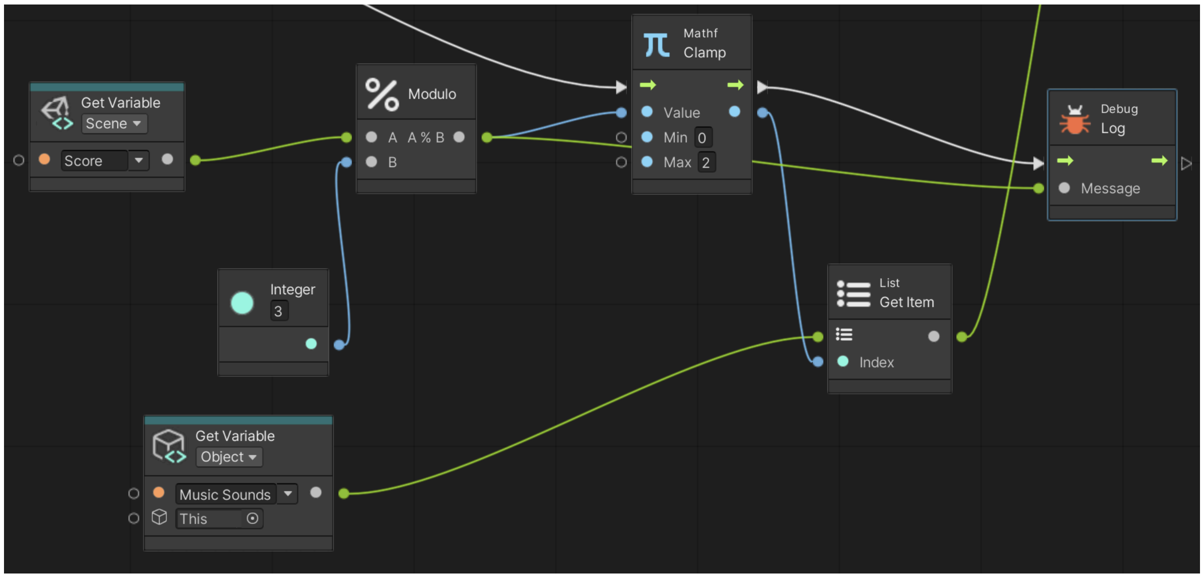
c) We will load our first Custom Node ("Log and Trigger on Audio Loop"). This will output a trigger every time our beat loop restarts.

i) We will take a quick look at the logic. I have included a guide at the end of this tutorial if you'd like to dig deeper into custom nodes using C#.

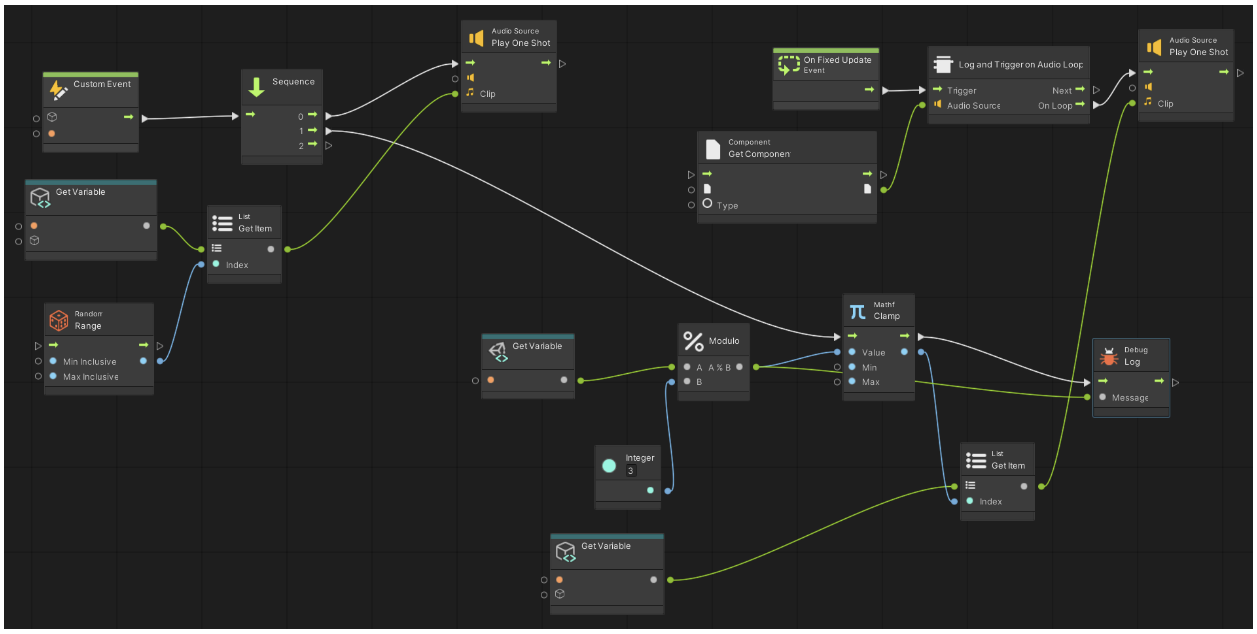


Step 2: Change it up! Incremental or Quasi-Generative Game Score

Use Modulo to keep our index within bounds so we can keep playing the game and pick an audio sample dependent on the current score.



Step 3: Make Final Connections and Play!



We use our ListItem to determine the next sample. We can use DebugLog to keep track of the current score.

End of Session Reflection:

How does the music impact the game?

Supplemental Resource: How to Write Custom Nodes
This could be written more succinctly using a custom node.

Store the script in your Assets folder.

- Go to Edit > Project Settings.
- Navigate to Visual Scripting.
- Find the section related to Node Library and click the Regenerate button.

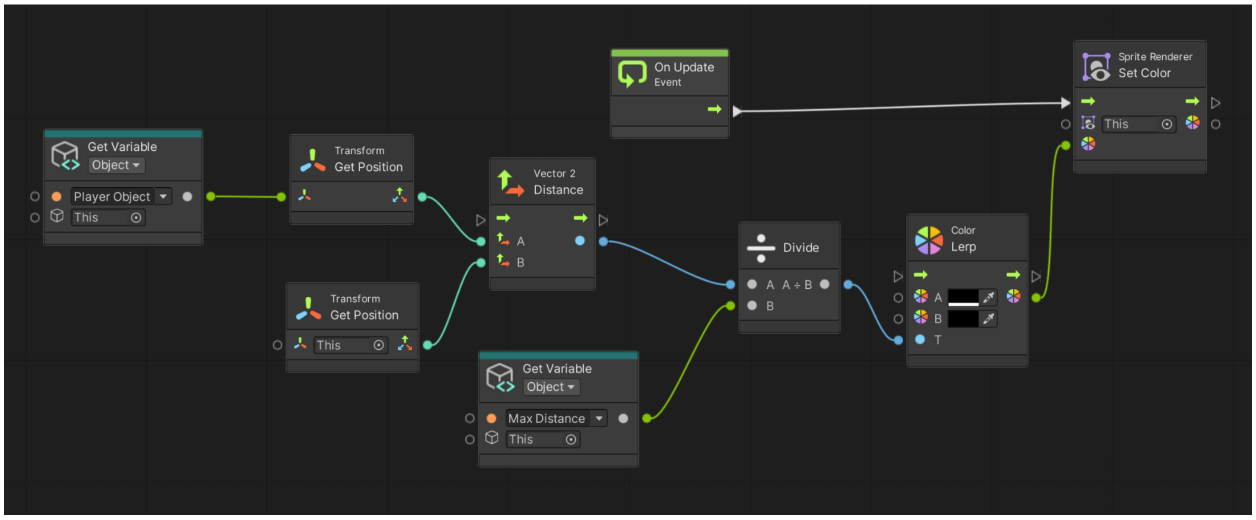
With the Node Library updated, your custom node should now appear in the node browser within the Visual Scripting graph editor.

Extra Time: Design Challenge | Use Music to Guide Your Collectible Experience

If we have time: Self-Directed Design Challenge!

We will cover this in the third and final part of this workshop. But go ahead and give it a go on your own.

- How could you apply the same logic used for collectible object transparency to gradually fade in and out music loops relative to player position?



- How could we use the player listener position to have repeating music elements guide the user to find the collectible?

- Could the music samples fade in gradually and playback relative to the player object position on the 2D grid?

How to get started? Hint! Let's make the music loops respect the player's position on the 2D horizontal stereo plane.
Activate the spatial attributes of the Audio Listener by removing the AudioListener from SoundManager and adding one to the Player game object. This will allow for audio to playback from a spatial location relative to the Player game object position on the 2D grid.

Other Design Challenges Coming Up in Workshop 3 (Part 6):

1. How can you create logic to avoid retriggering the same audio sample over and over again if the user input never changes? Hint: look at get and set variable nodes in addition to the comparison node.

2. How can we attach our music loops to the randomized position of each collectible?

3. How can you create logic to fade in and out an audio sample that is currently not playing and then loop it one time without fades only if the player collectible is discovered?