Audience Analysis

The organization I am writing these technical instructions for is the University of Maryland Immersive Media Design Department. They teach students about immersive art tools such as game development software, XR, and projection mapping. For this department, I am both a TA and a student. I TA for the introductory class where they learn the fundamentals of using the Unity Game Engine. The intended audience are the students taking the Immersive Media Design 101 class. Many want to be Immersive media design majors and are trying to learn more about the technologies available to them. They have a good amount of fundamental experience at this point in the semester. Therefore, by following my instructions, they will become more familiar with the different artistic possibilities available to create a game. They need to do the instructions to see the possibilities and limitations when creating a full Unity project.

Unity Engine Gesterial Recognition Game

Introduction:

An efficient way of learning new skills is by example. Students in the IMDM program at UMD need to constantly develop their skills in order to create effective, unique projects. These sets of instructions will help you learn how you can utilize other people's work to build off of your own projects. It will also show you how to develop progress for creating a multi-feature, interconnected game. We will be using the mediapipe library to use gesterial control in conjunction with a particle system for the player to interact with.

Warnings/Caution:

Throughout the development process of this project, make sure to continuously press play at the end of the steps to make sure that there are no errors. If you miss a step, this may be harder to find the further you progress into the steps.

Technical Background:

For these instructions, users should have a basic understanding of Unity's interface, including navigating the scene view, hierarchy, and inspector windows. They should have some proficiency in creating and manipulating gameobjects, creating C# scripts, and debugging their projects.

Users should have a grasp of C# programming language as Unity primarily uses C# for scripting. While it is not necessary to be a proficient coder, understanding basic programming concepts such as variables, conditionals, loops, and functions will be beneficial.

a fundamental understanding of computer vision and machine learning concepts would be beneficial, particularly for gesture recognition. While not necessary, familiarity with MediaPipe, an open-source framework for building computer vision pipelines, can provide the user with an insight into implementing gesture recognition functionalities within Unity.

Users with a basic understanding of game development principles, including game mechanics, level design, and player interaction, will retain the most from these instructions as they will be able to take it a step further and integrate the concepts into their own projects.

With these prerequisites in mind, users will be better equipped to follow instructions for creating a basic gesture control game in Unity Engine that responds to collisions with oscillating balls, enhancing the users ability to develop interactive and immersive gaming experiences.

Materials:

1. Unity Hub

a. Unity Hub helps manage projects, installations, and versions. It is how you access the Unity Editor to develop the game.

2. Visual Studio Code
    a. Visual studio code is an Integrated development environment(IDE) to code your C# scripts in. While any IDE that supports C# is sufficient, this is the one that will be used throughout the tutorial.
3. MediaPipe package
    a. Mediapipe is an open source framework for machine learning pipelines. We will be utilizing it for its gesture recognition capabilities. It is an ideal environment for writing and editing scripts in Unity as it has features such as color coded syntax, code completion, and an intuitive file system.
4. Computer
    a. You will need a computer with sufficient power, memory, GPU, and CPU. It should meet the systems requirements for using Unity and Visual Studio Code (VSCode)
    b. VSCode
        i. System Requirements
            1. Hardware requirements:
                a. 1.6 GHz or faster processor & 1 GB of RAM
            2. Software Supported Platforms:
                a. Windows 10 and 11 (64-bit)
                b. macOS versions with Apple security update support. This is typically the latest release and the two previous versions
                c. Linux (Debian): Ubuntu Desktop 20.04, Debian 10
                d. Linux (Red Hat): Red Hat Enterprise Linux 8, Fedora 36
    c. Unity
        i. System requirements:

Minimum requirements	Windows	macOS	Linux (Support in Preview)
Operating system version	Windows 7 (SP1+), Windows 10 and Windows 11, 64-bit versions only.	High Sierra 10.13+	Ubuntu 20.04, Ubuntu 18.04, and CentOS 7
CPU	X64 architecture with SSE2 instruction set support	X64 architecture with SSE2 instruction set support	X64 architecture with SSE2 instruction set support
Graphics API	DX10, DX11, and DX12-capable GPUs	Metal-capable Intel and AMD GPUs	OpenGL 3.2+ or Vulkan-capable, Nvidia and AMD GPUs.
Additional requirements	Hardware vendor officially supported drivers	Apple officially supported drivers	Gnome desktop environment running on top of X11 windowing system, Nvidia official proprietary graphics driver or AMD Mesa graphics driver. Other configuration and user environment as provided stock with the supported distribution (Kernel, Compositor, etc.)
	For all operating systems, the Unity Editor is supported on workstations or laptop form factors, running without emulation, container or compatibility layer.		

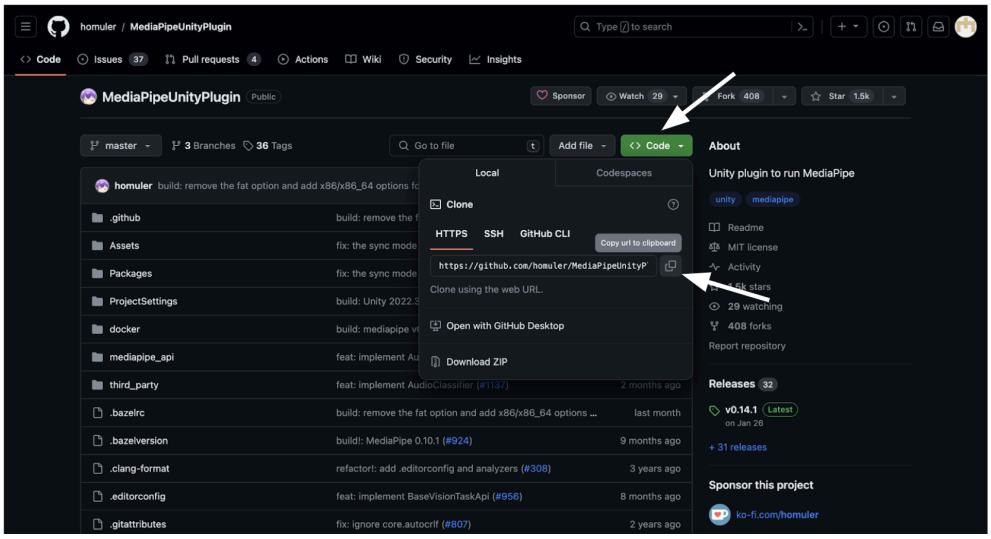
5. Github on terminal
    a. Github is a platform which allows for version control and collaboration on software development. We will use github on the terminal to smoothly download the project.

b. To check if you have github on the terminal:
    i. Open your terminal and run the command: git --version
    ii. If you do not have it installed, follow the instructions on: https://github.com/git-guides/install-git

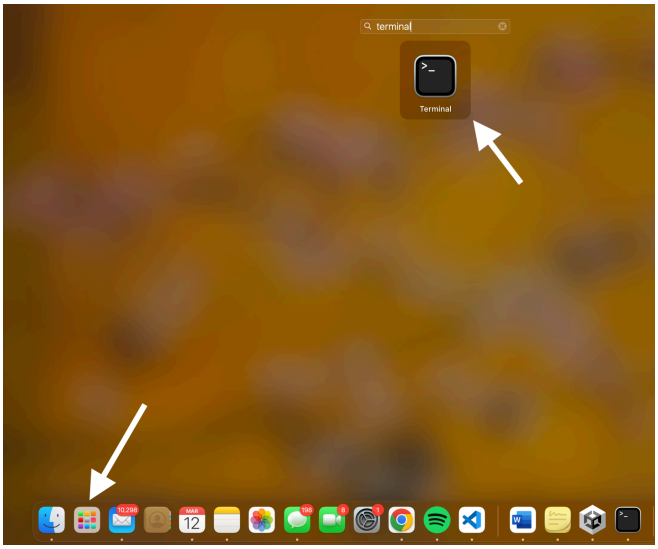
Instructions:

Setting up The Project Files:

1. **Get the project from Github**
    a. Open up the link to the github where the mediapipe gesterial recognition project is stored: https://github.com/MyunginLee/GesturalControl_Unity.git
    b. On the github click the "<> Code" button. Copy the HTTPS URL



    c. Open up your terminal
        i. On Mac:
            1. Open your launchpad on the dock
            2. Search for terminal

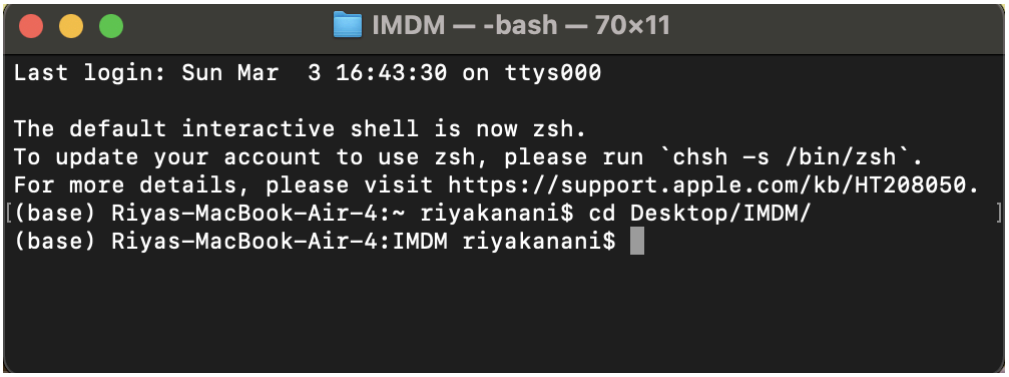


        ii. On Windows:
            1. Open the start menu

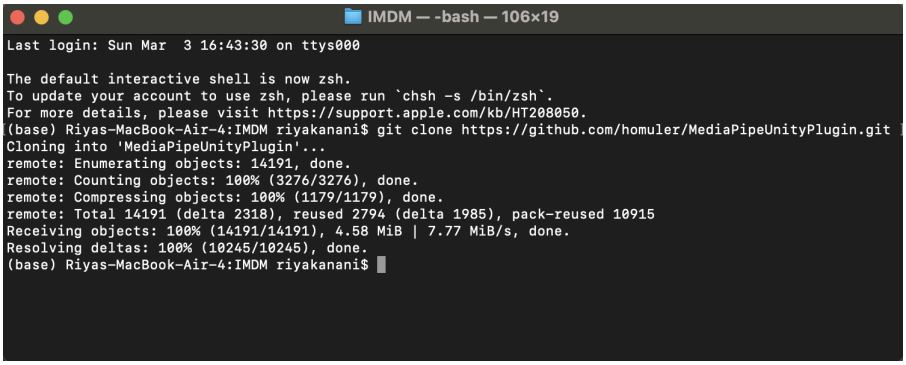
2. Search up “command prompt” or “powershell”

d. Type “git version” to check if you have github installed
   i. If you do not, follow the instructions on the website to install github: https://github.com/git-guides/install-git

e. Enter the file path where you store your Unity Projects through using the “cd” command. Type *cd [file path]*



f. Type *git clone [the URL you copied from github]* in the terminal



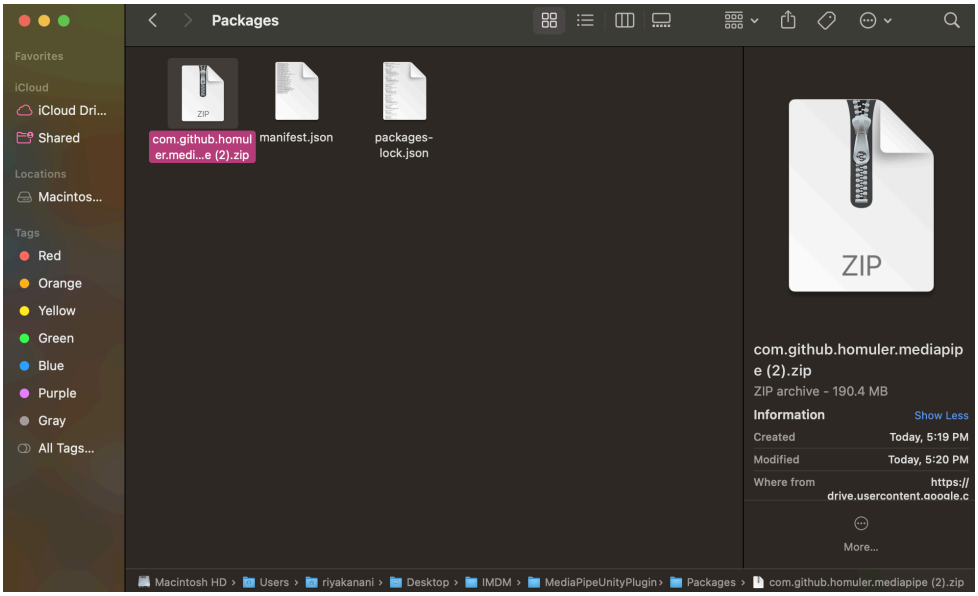
g. You can now close the terminal

2. Importing files into the project

a. Download the following link and unzip the file https://drive.google.com/file/d/1FHeuvVfdFxR-u5moOPuVg5mrkPM-Hl1L/view?usp=share_link

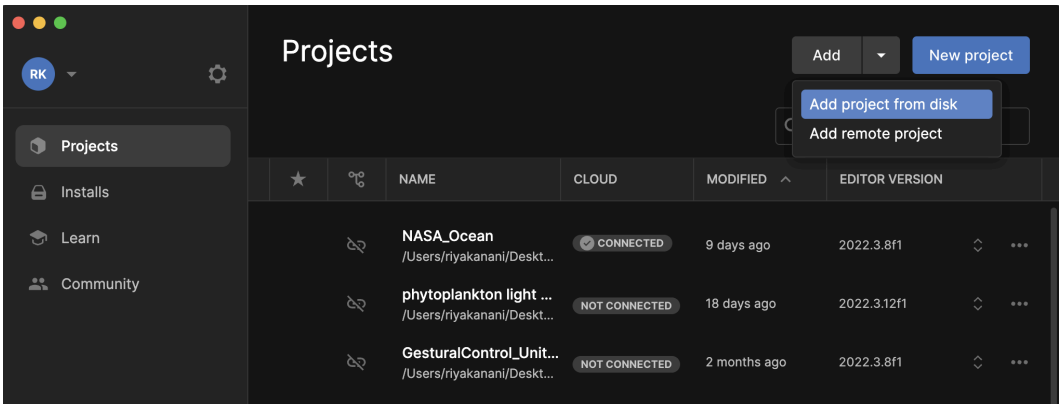
b. Open up the project in your file directory, open up the packages folder in the project.

c. Drag the file you downloaded into the packages folder

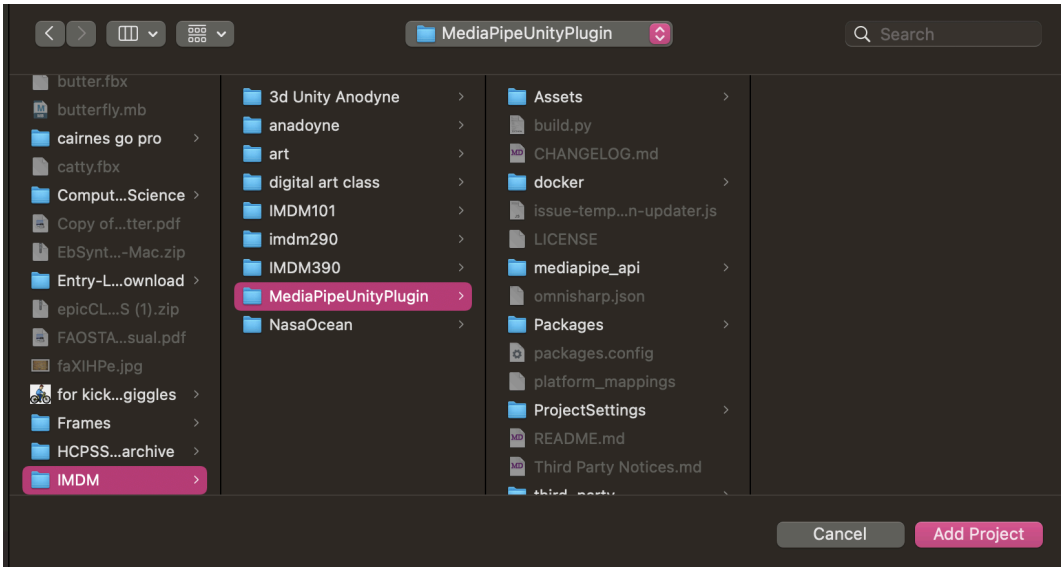


Setting up The Project Editor:

3. **Open up your Unity Hub application on your computer**
    a. Click on add project from disk

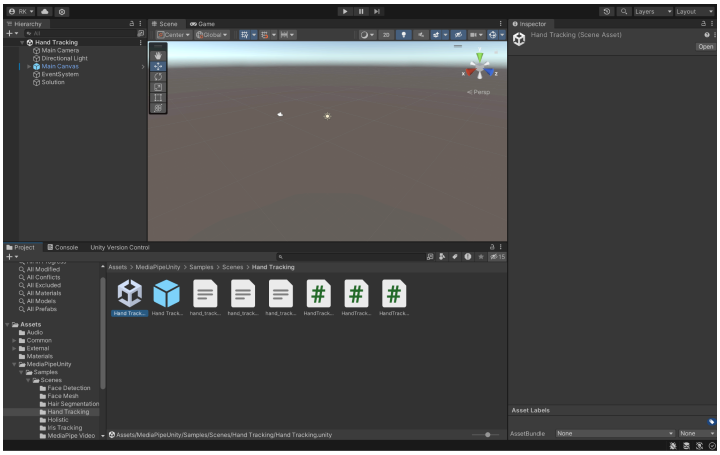


b. Locate where you downloaded the project in your file directory and add project

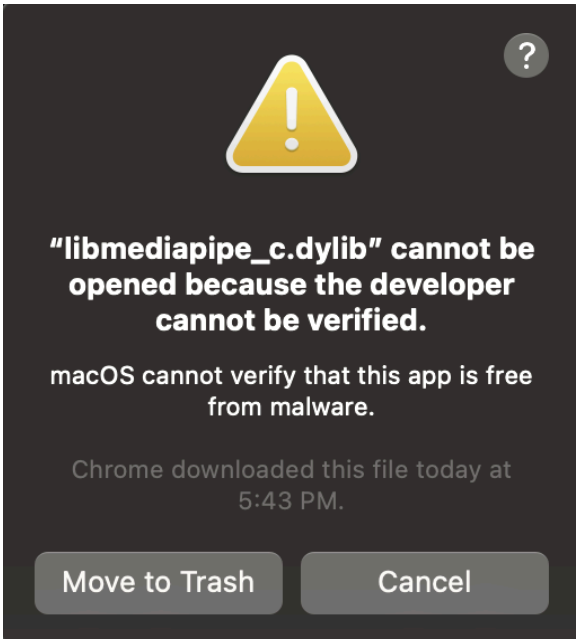


4. In the editor, go to Assets -> MediaPipeUnity -> Samples -> Scenes -> Handtracking

a. Open the handtracking scene and enter play mode

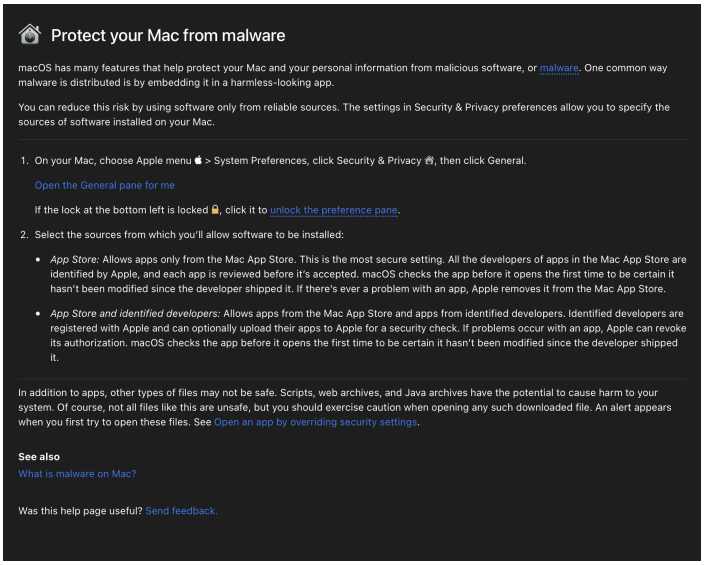


b. On macs if you get the following error:

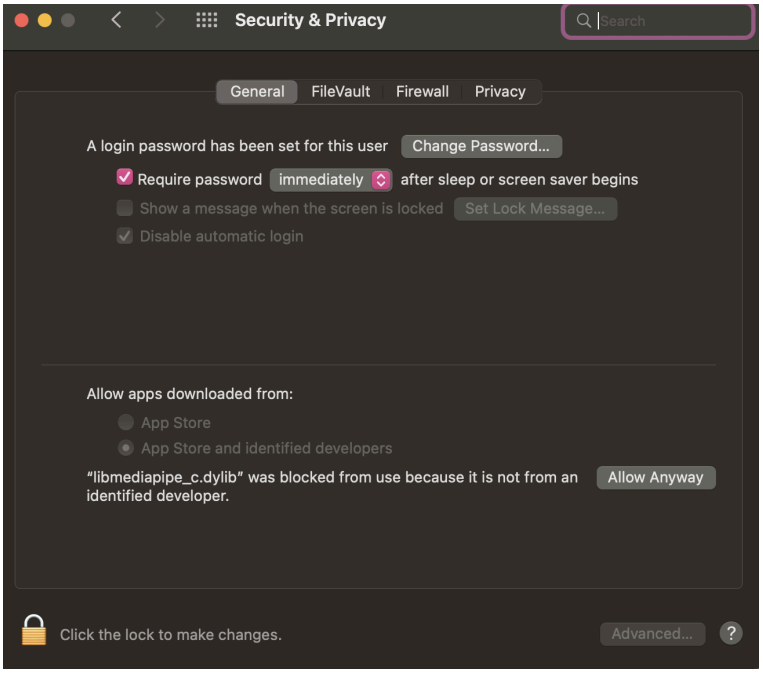


i. click the question mark

ii. Click on "open the general pane for me" button

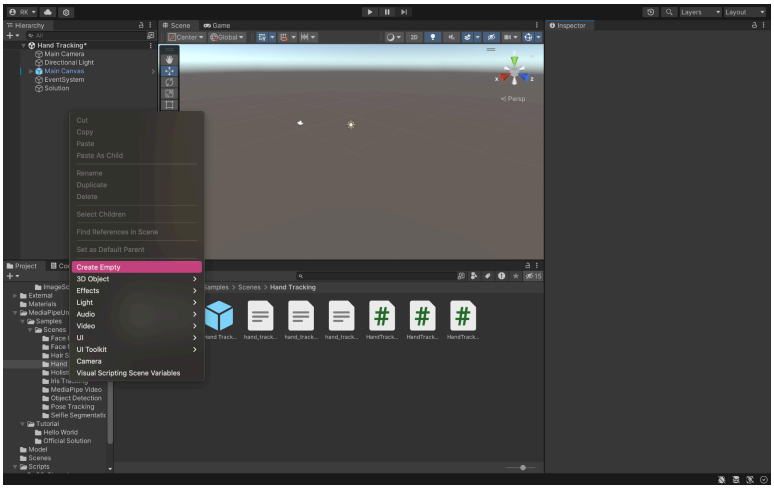


iii. Open the lock and then click “Allow Anyways”



Creating the Gestural Control:

5. **Creating finger objects**
   a. Create a new Empty Game object and name it “Genesis”



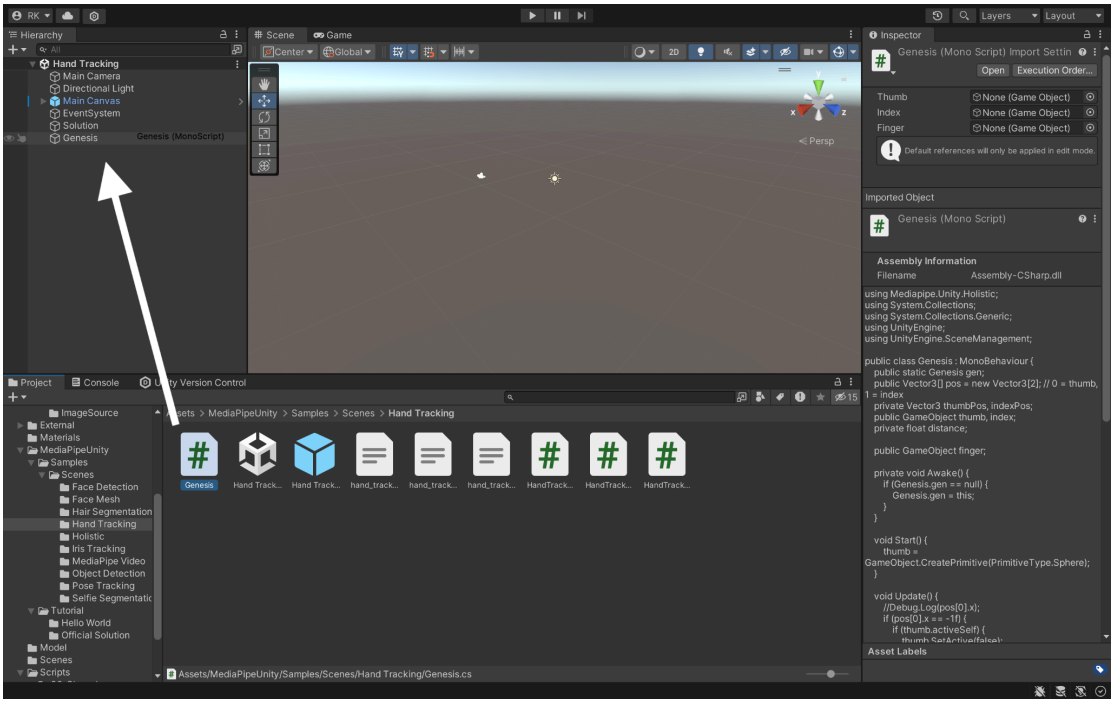
   b. In the “Hand Tracking” folder, create a new script and name it “Genesis”
      i. Add the following code into the Genesis script and save the changes:

```
using Mediapipe.Unity.Holistic;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class Genesis : MonoBehaviour {
  public static Genesis gen;
  public Vector3[] pos = new Vector3[2]; // 0 = thumb, 1 = index
  private Vector3 thumbPos, indexPos;
  private GameObject index, thumb;
  private float distance;

  private void Awake() {
     if (Genesis.gen == null) {
        Genesis.gen = this;
     }
  }

  void Start() {
     index = GameObject.CreatePrimitive(PrimitiveType.Sphere);
     thumb = GameObject.CreatePrimitive(PrimitiveType.Sphere);
  }

  void Update() {
     thumbPos.x = Mathf.Lerp(-9.5f, 4.4f, pos[0].x);
     thumbPos.y = Mathf.Lerp(4.8f, -3.4f, pos[0].y);
     indexPos.x = Mathf.Lerp(-9.5f, 4.4f, pos[1].x);
     indexPos.y = Mathf.Lerp(4.8f, -3.4f, pos[1].y);

     thumb.transform.position = thumbPos;
     index.transform.position = indexPos;

     distance = (thumbPos - indexPos).magnitude;
  }
```

```
}
```

c. Click and drag the Genesis script into the Genesis game object



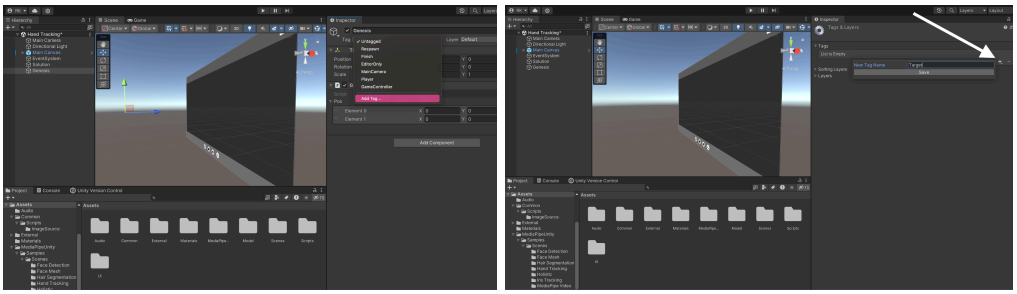
d. Open the HandTrackingSolution.cs script

i. Delete the code in the OnHandLandmarksOutput method and replace it with

```
private void OnHandLandmarksOutput(object stream,
OutputEventArgs<List<NormalizedLandmarkList>> eventArgs)
  {
    if (eventArgs.value != null) {
      Gen.gen.pos[0].x = (float)eventArgs.value[0].Landmark[4].X;
      Gen.gen.pos[0].y = (float)eventArgs.value[0].Landmark[4].Y;

      Gen.gen.pos[1].x = (float)eventArgs.value[0].Landmark[8].X;
      Gen.gen.pos[1].y = (float)eventArgs.value[0].Landmark[8].Y;

    } else {
      Gen.gen.pos[0].x = -1f;
    }
  }
```

Creating The Target:

6. **make a sphere that oscillates through scripting**
    a. Create a new tag called “Target”



    b. Create a new script and name it “Oscillator” in the Hand Tracking solutions folder.
        i. Paste the following code into the file:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Oscillator : MonoBehaviour
{
  public int particleCount = 3;
  public float speed = .5f;
  public float width = 5;
  public float height = 5;
  float timeCounter = 0;
  GameObject[] particles;

  void Awake()
  {
     particles = new GameObject[particleCount];

  }
  // Start is called before the first frame update
```

```
  void Start()
  {
     for (int i = 0; i < particleCount; i++)
     {
        particles[i] = GameObject.CreatePrimitive(PrimitiveType.Sphere);
        particles[i].transform.localScale = new Vector3(2, 2, 2);
        particles[i].tag = "Target";
        Color customColor = Color.HSVToRGB(Random.Range(0, 1f), Random.Range(0.8f, 1f),
Random.Range(0.5f, 1f));
        particles[i].GetComponent<Renderer>().material.SetColor("_Color", customColor);

     }
  }

  // Update is called once per frame
  void Update()
  {
     float angleStep = 360f / particleCount; // Angle step for even distribution

     for (int i = 0; i < particleCount; i++)
     {
        float angle = i * angleStep;
        float radians = Mathf.Deg2Rad * angle;

        timeCounter += Time.deltaTime * speed;
        float x = Mathf.Cos(timeCounter + radians) * width;
        float y = Mathf.Sin(timeCounter + radians) * height;
        float z = 0;
        if(particles[i] != null){
           particles[i].transform.position = new Vector3(x, y, z);
        }
```

```
    }
  }

}
```

c. Create another script called “SphereController” which will give each target a unique number. This helps in creating different outcomes from hitting each target.
    i. Paste the following code into the file

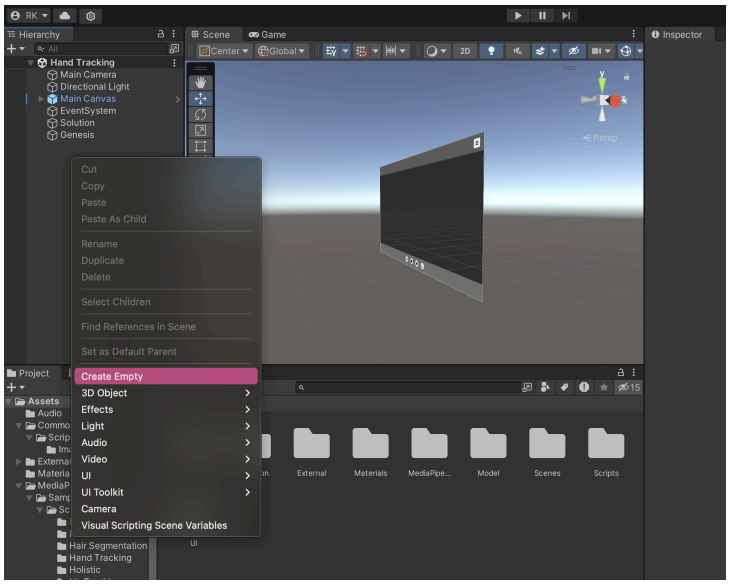
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SphereController : MonoBehaviour
{
  public int myNumber = 0;


}
```

d. Open up the oscillator script
    i. Add the highlighted code to the start method:

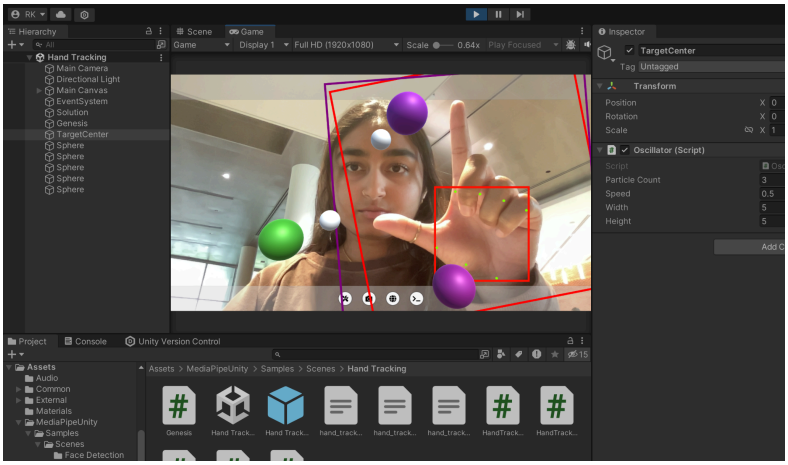
```
void Start()
  {
    for (int i = 0; i < particleCount; i++)
    {
      particles[i] = GameObject.CreatePrimitive(PrimitiveType.Sphere);
      SphereController sphereController = particles[i].AddComponent<SphereController>();
      sphereController.myNumber = i + 1;
      particles[i].transform.localScale = new Vector3(7, 7, 7);
      particles[i].tag = "Hint";
      Color customColor = Color.HSVToRGB(Random.Range(0, 1f), Random.Range(0.8f, 1f),
Random.Range(0.5f, 1f));
      particles[i].GetComponent<Renderer>().material.SetColor("_Color", customColor);
```

```
    }
}
```

7. **Place the target onto the game**
    a. Create a new empty game object called TargetCenter



    b. Paste the Oscillator script onto the Target game object
    c. Press play to make sure you have no compilation errors, your game should look like this on play



Putting The Parts together:

8. **Make the game recognize the collision**
    a. Open the Genesis script
        i. Add the highlighted code to the start method

```
void Start() {
    index = GameObject.CreatePrimitive(PrimitiveType.Sphere);
```

```
    index.tag = "Player";
    thumb = GameObject.CreatePrimitive(PrimitiveType.Sphere);
    thumb.tag = "Player";
  }
```

b. Create a new script in the hand tracking folder called Collision.cs

i. Paste the following code:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using Mediapipe.Unity;

public class Collision : MonoBehaviour
{
  public float interactionDistance = 0;

  void Update()
  {
    GameObject[] hintObjects = GameObject.FindGameObjectsWithTag("Target");
    GameObject[] fingerObjects = GameObject.FindGameObjectsWithTag("Player");
    if (hintObjects != null){
      // Check each child collider
      foreach (GameObject hintObject in hintObjects)
      {
        foreach (GameObject fingerObject in fingerObjects)
        {
            float distance = Vector3.Distance(fingerObject.transform.position,
hintObject.transform.position);
            if (distance <= interactionDistance)
            {
```

```
                    // Perform the action when a Hint object is within the distance of a RightHand
object
                    Destroy(hintObject);
                    return;
                }
            }
        }
    }
}
```

c. Click and drag the Collision script onto the TargetCenter game object

9. Create an action from hitting the object

a. In the collision script, in the update method, remove *Destroy(hintObject)* and replace it with *PerformAction(hintObject);*

b. Create another method under the update method called performAction that takes in a gameObject as a parameter

i. Paste the following code into this new method

```
void PerformAction(GameObject hint)
  {
    switch(hint.GetComponent<SphereController>().myNumber) {
      case 1:
        //do something
        break;
      case 2:
        //do something
        break;
      case 3:
        //do something
        break;
    }
    Destroy(hint);
  }
```

c. Insert your own actions into the cases such as scene switches

10. Test the Game

a. Press the play button
b. Use your index finger or thumb to hit the balls
c. Try to hit all of the balls until there are no more left

References

Aalam, A. (2021, September 26). *Mediapipe bracelet demo part - 1 | mediapipe | unity | Arham Aalam*. YouTube. https://www.youtube.com/watch?v=eZbi08dNCOU&list=PLRKJ1vRSUVCI4kZHpxkOhSabIfGXkvT_J&index=3&ab_channel=ArhamAalam

Boundless Studios. (2016, May 1). *Rotate around or orbit a game object in Unity 3D C# tutorial beginner*. YouTube. https://www.youtube.com/watch?v=3PHc6vEckvc&ab_channel=BoundlessStudios

Buckley, I. (2019, August 21). *Unity learn is the easiest way to master game development*. MUO. https://www.makeuseof.com/tag/unity-learn-game-development/

Carotenuto, A. (2023, November 14). *Unity 3D: Collisions basics*. binPress. https://www.binpress.com/unity-3d-collisions-basics/

GitHub, Inc. (n.d.). *Git guides*. GitHub. https://github.com/git-guides/install-git

Meer. (2023, May 11). VTUBER model and GUI using MediaPipe and unity. Medium. https://medium.com/@meerabahsan03/vtuber-model-and-gui-using-mediapipe-and-unity-1f49a5eac6a7

Microsoft. (2024, February 28). *Requirements for Visual Studio Code*. Visual Studio Code. https://code.visualstudio.com/docs/supporting/requirements#:~:text=VS%20Code%20is%20lightweight%20and,1%20GB%20of%20RAM

Nishida, J. (n.d.). *MediaPipe Unity Plugin*. Github. https://github.com/homuler/MediaPipeUnityPlugin

Srivastava, A. (2021, July 30). *Unity Game Engine: Why is it the better option for beginners?*. Medium. https://medium.com/sigma-xi-vit/unity-game-engine-why-it-is-the-better-option-for-beginners-3a192feabecb

Unity Technologies. (2024, February 21). *Introduction to collision*. Unity. https://docs.unity3d.com/Manual/CollidersOverview.html

Unity Technologies. (n.d.). *System requirements for unity 2021.1*. Unity. https://docs.unity.cn/2021.1/Documentation/Manual/system-requirements.html