Team 2-bit Dev Log

Week 9: 13/11/23 - 17/11/23

Michael Hayes

Overview:

This week we have been working on our current sprint 5, refactoring our prototype code and systems into a new project, focusing on SOLID principles to create an efficient and effective workflow. I have also started a blender course as we are unable to find a visual artist for our project although it is taking more time than I first expected.

Agile Sprint Update:

Sprint 5: Alpha Build

Create a feature complete alpha build that has all the core components and features present.

Links

Link to Minutes and Agenda: CS2 - Minutes and Agenda
Link to Team's Jira Scrum board:
https://cs2mr.atlassian.net/jira/software/projects/CS2LVR/boards/2/backlog
Miro board overview: https://miro.com/app/board/uXjVMj-Nye0=/?share_link_id=77318451600

Table of Contents:

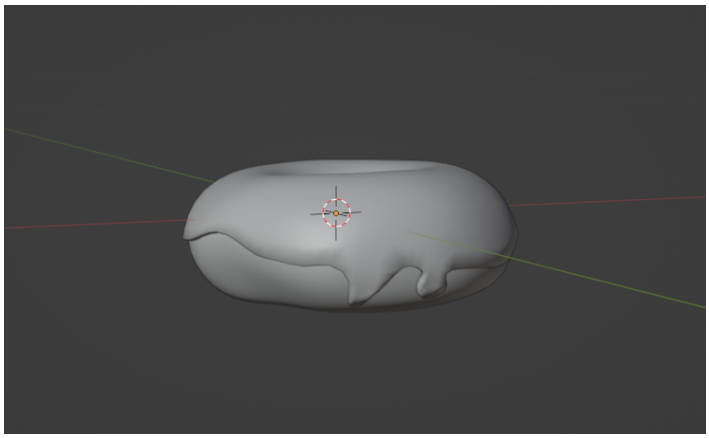
This week’s completed tasks:

Sprint 5 Tasks:

Learning Blender

Following a youtube tutorial creating a donut from scratch, I am currently learning the basics of blender and how to use the tools effectively which I hope to transfer to making a target and weapon for the game.

My donut result so far



Refactoring Code

As the project up to this point has been a prototype, we have not been too thoughtful with the code and how it interacts with the rest of the code base. As we want to keep clean code, avoiding technical debt and creating a more efficient workflow, we want to try to keep to the SOLID principles where it makes sense to do so.

Refactoring the Laser Gun script

The laser gun script in the prototype project was a larger script that contained all the functionality of the laser guns within the game. This is not following the SOLID principles and is becoming hard to manage, and for other team members to interact with.
I decided to separate the large class into multiple smaller classes responsible for a single purpose.

The input script is an example of this, and now is solely responsible for activating the VR device and setting inputs for the primary and secondary hands. When an input is pressed from the right or left hand controller, we call the Fire method to activate the ray cast logic.

Tooltips are also used where other team members will have to interact, to express what the field's purpose is.

Input Script

```
6   public class PlayerInput : MonoBehaviour
7   {
8       [Tooltip("Attach primary and secondary hand game objects located on the VR avatar, here")]
9       [SerializeField] private GameObject primaryHand, secondaryHand;
10      private LaserGunRayCast rayCast;
11
            Unity Message | 0 references
12      private void Start()
13      {
14          rayCast = gameObject.GetComponent<LaserGunRayCast>();
15      }
16
            Unity Message | 0 references
17      private void Update()
18      {
19          var rightHandInput = GetInput(VRInputDeviceHand.Right); // Primary Hand
20          var leftHandInput = GetInput(VRInputDeviceHand.Left); // Secondary Hand
21
22          if (rightHandInput != null)
23          {
24              if (rightHandInput.GetButtonDown(VRButton.One))
25              {
26                  //Debug.Log("Call right hand input");
27                  rayCast.Fire(primaryHand.transform);
28              }
29          }
30
31          if (leftHandInput != null)
32          {
33              if (leftHandInput.GetButtonDown(VRButton.One))
34              {
35                  //Debug.Log("Call left hand input");
36                  rayCast.Fire(secondaryHand.transform);
37              }
38          }
39      }
40
            2 references
41      private IVRInputDevice GetInput(VRInputDeviceHand hand)
42      {
```

When the input is pressed the ray cast logic is executed (shown in the screenshot below). This method takes in a transform position as a parameter so we know which gun to cast the raycast from.

As this class is only responsible for drawing the ray cast and checking if the raycast has hit a collider or not, I used a Unity event called On Laser Hit that other classes can subscribe to and listen for. When the event is invoked, if a collider was hit with the raycast, subscribed methods will be called.

Ray cast script

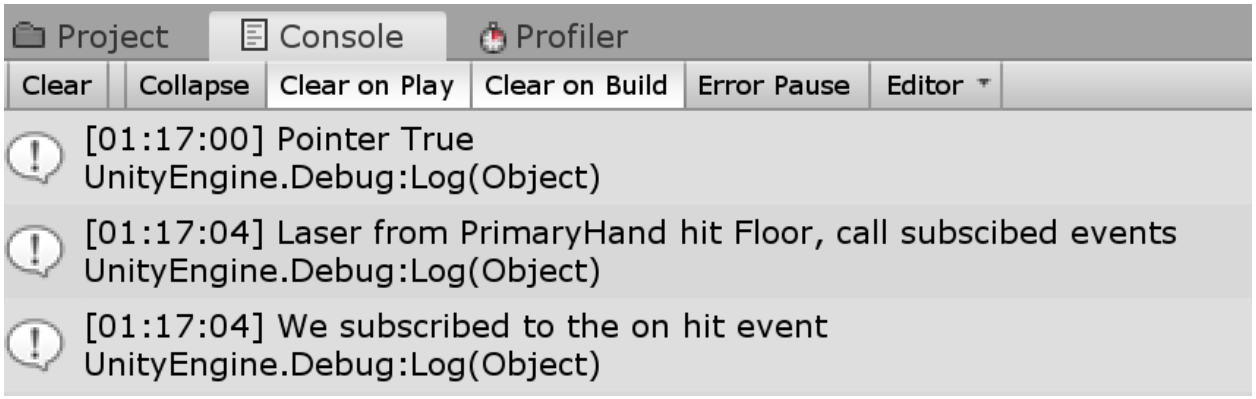
```
using UnityEngine;

Unity Script | 4 references
public class LaserGunRayCast : MonoBehaviour
{
    public delegate void HitAction();
    public static event HitAction OnLaserHit;

    2 references
    public void Fire(Transform rayOrigin)
    {
        RaycastHit hit;
        Ray laserGunRay = new Ray(rayOrigin.position, rayOrigin.forward);
        float rayDuration = 1f;
        Debug.DrawRay(rayOrigin.position, rayOrigin.forward, Color.red, rayDuration);

        if(Physics.Raycast(laserGunRay, out hit))
        {
            if (hit.collider != null && OnLaserHit != null)
            {
                Debug.Log("Laser from " + rayOrigin.gameObject.name +
                    " hit " + hit.collider.gameObject.name +   ", call subscibed events!");
                OnLaserHit();
            }
        }
    }
```

To test this I created a test script that would show a line on the console when the raycast hit.

Event test script

```
using UnityEngine;

Unity Script | 0 references
public class EventTest : MonoBehaviour
{
    Unity Message | 0 references
    void OnEnable() => LaserGunRayCast.OnLaserHit += Test;
    Unity Message | 0 references
    void OnDisable() => LaserGunRayCast.OnLaserHit -= Test;

    2 references
    private void Test()
    {
        Debug.Log("We subscribed to the on hit event");
    }
}
```

Project console



Only matching colour gun can destroy target

First I need to add a way for targets to select a colour when they spawn. To do this I created a bool that would randomly be set to true or false, if true then the target would select material 1 and if false it would select the second colour. This is done through a enum and state machine.

Target type class

```
Unity Message | 0 references
private void Start()
{
    bool randColour = Random.Range(0, 2) == 1;
    if (randColour) targetMaterial = TargetMaterial.Primary;
    else targetMaterial = TargetMaterial.Secondary;

    switch (targetMaterial)
    {
        case TargetMaterial.Primary:
            gameObject.GetComponent<Renderer>().material = primaryMaterial;
            break;

        case TargetMaterial.Secondary:
            gameObject.GetComponent<Renderer>().material = secondaryMaterial;
            break;
    }
}
```

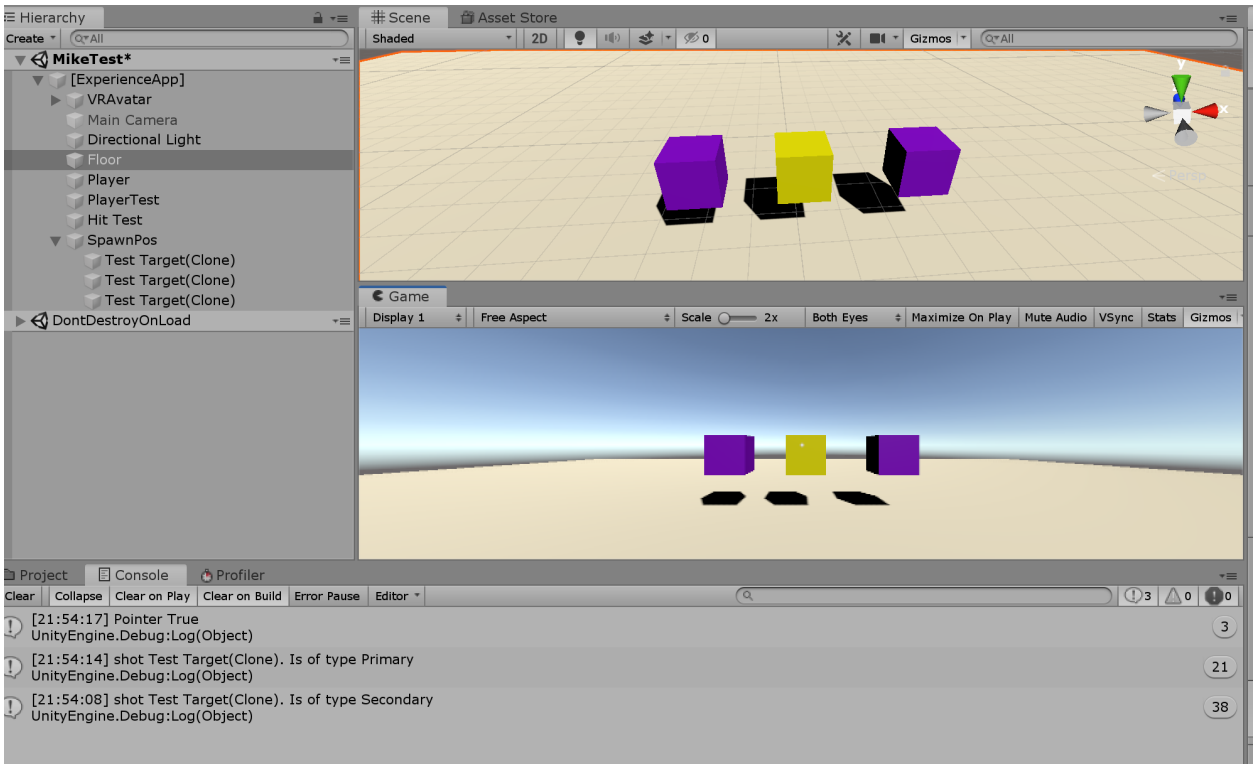
I then created a getter so the ray cast script can see which type the target it

```
1 reference
public TargetMaterial GetTargetType()
{
    return targetMaterial;
}
```

Then I created a reference to the getter within the ray cast script

```
2 references
public void Fire(Transform rayOrigin, int gunIndex)
{
    RaycastHit hit;
    Ray laserGunRay = new Ray(rayOrigin.position, rayOrigin.forward);
    DebugRayCast(rayOrigin);

    if(Physics.Raycast(laserGunRay, out hit))
    {
        if (hit.collider != null && OnLaserHit != null &&
            hit.collider.gameObject.GetComponent<TargetType>())
        {
            TargetType.TargetMaterial targetType =
                hit.collider.gameObject.GetComponent<TargetType>().GetTargetType();

            Debug.Log("shot " + hit.collider.gameObject.name + ". Is of type " +  targetType);
        }
    }
}
```

Results from testing the above code



Now that I know the target colour can be found, I check to see if the correct gun is being used to hit the correct target before calling the target hit methods.

```
10  public void Fire(Transform rayOrigin, int gunIndex)
11  {
12      RaycastHit hit;
13      Ray laserGunRay = new Ray(rayOrigin.position, rayOrigin.forward);
14      DebugRayCast(rayOrigin);
15
16      if(Physics.Raycast(laserGunRay, out hit))
17      {
18          if (hit.collider != null && OnLaserHit != null &&
19              hit.collider.gameObject.GetComponent<TargetType>())
20          {
21              TargetType.TargetMaterial targetType =
22                  hit.collider.gameObject.GetComponent<TargetType>().GetTargetType();
23
24              Debug.Log("shot " + hit.collider.gameObject.name + ". Is of type " +  targetType);
25
26              if (gunIndex == 1 && targetType == TargetType.TargetMaterial.Primary) PrimaryTargetHit(hit);
27
28              if (gunIndex == 2 && targetType == TargetType.TargetMaterial.Secondary) SecondaryTargetHit(hit);
29          }
30      }
31  }
```

```
32      private void PrimaryTargetHit(RaycastHit hit)
33      {
34          //spawner.despawnTarget(hit.collider.gameObject);
35          //OnLaserHit();
36          Debug.Log("We destroyed a primary colour target");
37      }
        1 reference
38      private void SecondaryTargetHit(RaycastHit hit)
39      {
40          Debug.Log("We destroyed a secondary colour target");
41      }
```

As it is hard to see where the laser gun is pointing in emulator mode, I added back the gizmos. Instead of manually placing in the transform positions of the laser guns, I used the inputs laser gun positions as references to keep everything uniform. I had one problem with the code where I was using a vector 3 on the Draw line method. This resulted in the line not moving with the controller. Changing the vector 3 to a transform fixed this and now it works as expected.

```
Unity Script | 0 references
public class LaserGunGizmos : MonoBehaviour
{
    private Transform primaryOrigin;
    private Transform secondaryOrigin;
    Unity Message | 0 references
    private void Start()
    {
        primaryOrigin = GetComponent<PlayerInput>().GetOriginPosPrimary();
        secondaryOrigin = GetComponent<PlayerInput>().GetOriginPosSecondary();
    }
    Unity Message | 0 references
    public void OnDrawGizmos()
    {
        Gizmos.color = Color.black;
        float distance = 100f;
        if (primaryOrigin != null) DrawLine(primaryOrigin, distance);
        if (secondaryOrigin !=null) DrawLine(secondaryOrigin, distance);
    }
    2 references
    private void DrawLine(Transform startPos, float distance)
    {
        Gizmos.DrawLine(startPos.position, startPos.forward * distance);
    }
}
```

Other

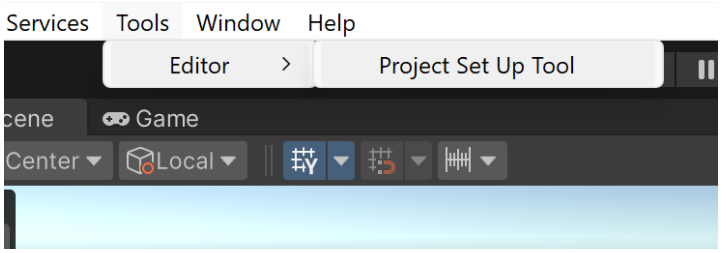
Creating Tools in Unity (Lesson)

Using Unity Editor and the Attribute Menu Item, we can make a custom menu in the unity project window and then attach functionality to it.

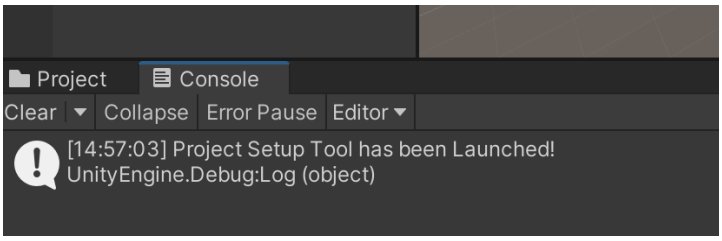
Menu script

```
using UnityEngine;
using UnityEditor;

namespace Tools
{
    0 references
    public class EditorMenus
    {
        // Directory of the added menu
        [MenuItem("Tools/Editor/Project Set Up Tool")]
        0 references
        public static void InitSetupTool()
        {
            Debug.Log("Project Setup Tool has been Launched!");
        }

    }
}
```

Unity project top bar



Console



To create the pop out window for once you click the menu button, we create a new script inheriting from the Unity editor window. Then using the get window & show function we create a new menu and have it show when called. The namespace will help prevent potential crashes.

Pop out window script

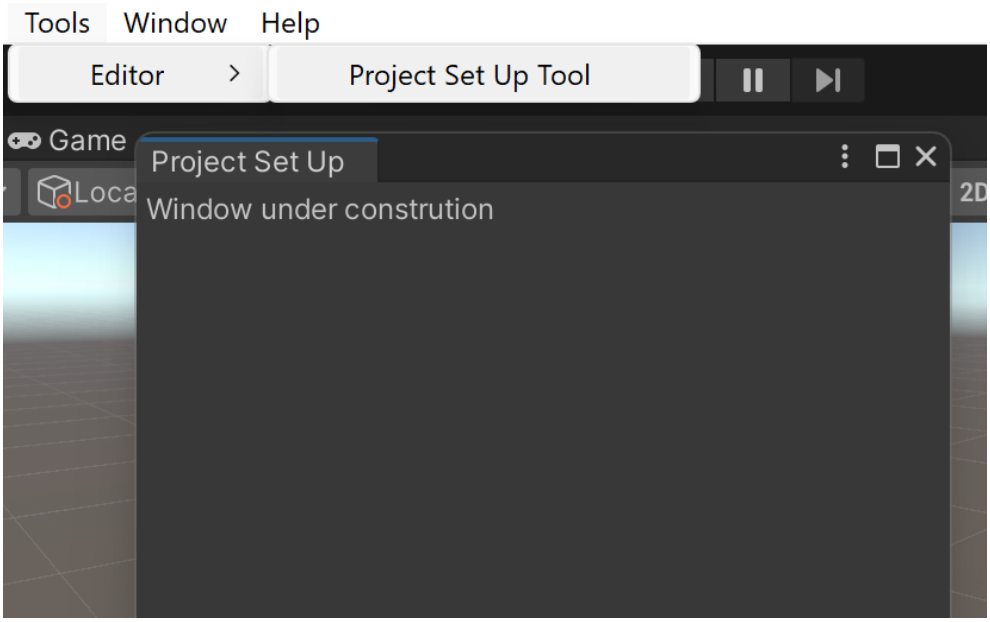
```
using UnityEditor;

namespace Tools
{
    // Inherit from Unity Editor > Editor Window
    Unity Script | 3 references
    public class ProjectSetupWindow : EditorWindow
    {
        static ProjectSetupWindow win;

        // Create a pop out window using Editor Window
        1 reference
        public static void InitWindow()
        {
            // Type of window inside <> & name of window inside ()
            win = EditorWindow.GetWindow<ProjectSetupWindow>("Project Set Up");
            win.Show();
        }

        // Edit the window's UI
        Unity Message | 0 references
        private void OnGUI()
        {
            EditorGUILayout.LabelField("Window under constrution");
        }
    }
}
```

On the editor menu script we call the pop out window function.

Menu script calling pop out window function

```
1   using UnityEditor;
2
3   namespace Tools
4   {
        0 references
5       public class EditorMenus
6       {
7           // Directory of the added menu
8           [MenuItem("Tools/Editor/Project Set Up Tool")]
            0 references
9           public static void InitSetupTool()
10          {
11              ProjectSetupWindow.InitWindow();
12          }
13      }
14  }
```

Pop out window in the Unity project



Team Members:

Robin Pound - Co-Lead:

Individual works:

- Continued work on refactoring Spawn system

Next Week's Goals:

- Complete the outstanding tasks for sprint 5 Alpha build for testing next week

Feedback and Comments: