Sprint 02

Assignment 03 : Level Design Tools

Step 01: Setup

- We setup everything in the Coding Part.
- We create an Example Script in the Scripts > Level > Editor.
- We use the "***using*** **UnityEngine**" namespace.
- And add a "**[MenuItem("NIIT/Level Editor")]**" line which adds a Sub-Menu in the Menu Bar of Unity.

Step 02: Script & Workflow

- The Scripts workflow is like this:
    - LevelEditor > AssetsCustomEditor & FoldOutObject.

- Rest of the scripts, GUIEditor, LayoutEditor, FoldOutEditor are Example Scripts.

❖ **FoldOutObject.**

- This script performs the action of adding a Dropdown Menu.
- It is further used in the "*AssetsEditorCustom*" wherever and whenever it is required.

```
AssetEditor.cs U    FoldOutObject.cs U ×    AssetsEditorCustom.cs U    LevelEditor.cs U
Assets > Scripts > Level > Editor > FoldOutObject.cs > FoldOutObject > FoldOutObject(string name, int no1, int no2)
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// Put this type of scripts in the Editor Folder as they are not used in Runtime. That is, when the Game runs into the GameLogic or something.
9 references
public class FoldOutObject
{
    8 references
    public string name;
    6 references
    public int no1;
    5 references
    public int no2;
    8 references
    public bool expanded;

    2 references
    public FoldOutObject(string name, int no1, int no2)
    {
        this.name = name;
        this.no1 = no1;
        this.no2 = no2;
    }

}
```

❖ AssetsEditorCustom.

- This script is the main script that handles all the functionalities.
- First, we create button that selects & focuses the "*EscapePoint*" GameObject from the Hierarchy.
- Secondly, we create another button that updates the NavmeshSurface Build.
- For this we access the NavMeshSurfcae gameobject in the scene and Build in each time the Button is pressed.

AssetsEditorCustom.cs U × | AssetEditor.cs U | FoldOutC

Assets > Scripts > Level > Editor > AssetsEditorCustom.cs > Assets

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
using UnityEngine.AI;
using System;
using System.IO;

2 references
public class AssetsEditorCustom
{
    3 references
    private GameObject escapePoint;
    4 references
    private GameObject navMesh;
    3 references
    private NavMeshSurface navMeshSurface;
    0 references
```

```
    1 reference
    public AssetsEditorCustom()
    {
        escapePoint = GameObject.Find("EscapePoint");

        navMesh = GameObject.Find("NavMesh");
        navMeshSurface = navMesh.GetComponent<NavMeshSurface>();    //Created the Surface of
        prefabInstances = new List<AssetInstance>();
        groupList = new List<FoldOutObject>();
        groupCount = 0;

        prefabPaths = new List<string>();
        prefabPaths.Add("-- Select Asset Folder --");
        prefabPaths.Add("Barriers");
        prefabPaths.Add("Props");
        prefabPaths.Add("Terrain");
    }

```

```
public void Render()
{
    GUILayout.BeginHorizontal();

        if(GUILayout.Button("Select EsacapePoint"))     //Escape Point
        {
            Selection.activeObject = escapePoint;
            SceneView.lastActiveSceneView.LookAt(escapePoint.transform.position);
            Debug.Log(Selection.activeObject.name);
        }

        if(GUILayout.Button("Updtae NavMesh"))      //Updated & Build NavMeshSurface
        {
            Selection.activeObject = navMesh;
            SceneView.lastActiveSceneView.LookAt(navMesh.transform.position);

            if(navMeshSurface != null)
            {
                navMeshSurface.BuildNavMesh();
                Debug.Log("NavMesh Updated!");
            }
        }

```

- Then, we begin with the implementation of the Grouping Part.
- We create a Button and a TextField that Creates an Empty GameObject with the Name entered in the String.
- We do these things in the "*Render*()" function.
- Then we from further, Shift into the "*DisplayGroup*()" function
- Also, we add a Tag called "AssetsGroup" to each new group created for simplification purposes.

```
private List<string> assetsPaths;
0 references
private List<string> assetNames;
5 references
private string groupName;
2 references
private GameObject newGroup;
27 references
private List<FoldOutObject> groupList;
3 references
private int groupCount;
```

```
        GUILayout.EndHorizontal();

        /////-------------Creating New Group-----------------------//////
        GUILayout.BeginHorizontal();

            GUILayout.Label("New Group :", GUILayout.Width(80));

            //TextField: Make a single-line text field where the user can edit a string.
            groupName = GUILayout.TextField(groupName);

            //Create Group Button.
            if (GUILayout.Button("Add Group", GUILayout.Width(100)) && groupName != null && GameObject.Find(groupName) == false)
            {
                newGroup = new GameObject(groupName);  //Create Empty GameObject with Name.
                newGroup.tag = "AssetGroup";           //Adding Tag.
                groupList = new List<FoldOutObject>();
            }

        GUILayout.EndHorizontal();

        DisplayGroup();

    }

    1 reference
    private void DisplayGroup()     //Creating & Deleting Group(s).
    {
        GameObject[] AssetGroupObjects = GameObject.FindGameObjectsWithTag("AssetGroup"); //Finds AssetGroup gameObjects with Tag "AssetGroup"
        if (groupCount != AssetGroupObjects.Length)
        {
            groupList = new List<FoldOutObject>();
            groupCount = AssetGroupObjects.Length;

            foreach (GameObject group in AssetGroupObjects)
            {
                groupList.Add(new FoldOutObject(group.name, 0, 0));  //Takes String - Name & Int - Number.
            }
        }
```

- Then, we add code to add a Sub-Group within the Parent-Group.
- We also add a condition that if we have One Sub-Group Open, and if we Open another Sub-Group, then the Previous One collapses automatically.

```
        //g = Group & sb = Sub-Group.
        for (int g = groupList.Count - 1; g >= 0; g--)
        {
            groupList[g].expanded = EditorGUILayout.Foldout(groupList[g].expanded, groupList[g].name);
            if (groupList[g].expanded)
            {
                //This Line Closes a Sub-Group, when another Group is Expanded.
                for (int sb = groupList.Count - 1; sb >= 0; sb--)
                {
                    if(groupList[sb].name != groupList[g].name && groupList[sb].expanded == true)
                    {
                        groupList[sb].expanded = false;
                        //prefabInstances = new List<AssetInstance>();
                    }
                }

                //Creating the Sub-Group.
                GUILayout.BeginVertical();
                {
                    GUILayout.BeginHorizontal();
                    {
                        groupList[g].name = GUILayout.TextField(AssetGroupObjects[g].name);
                        AssetGroupObjects[g].name = groupList[g].name;

                        if (GUILayout.Button("Delete Group", GUILayout.Width(100)))   //Delete
                        {
                            GameObject.DestroyImmediate(AssetGroupObjects[g]);
                            groupList.Remove(groupList[g]);
                            //prefabInstances = new List<AssetInstance>();
                        }
                    }
                    GUILayout.EndHorizontal();
```

- Next, we move ahead to Get all the Prefabs from their respective folder.
- For that we first Add the Title of the Folders and then add their path or location and finally the specific Prefab.

```
GUILayout.BeginHorizontal();
{
    //Get the Path of All the Assets present in the Particular Folder (Ex. Barrires).
    //prefabPaths = Constsits of 3 Assets Groups decalred above.
    groupList[g].no1 = EditorGUILayout.Popup(groupList[g].no1, prefabPaths.ToArray());   // Dropdown - 3 Assets Groups.

    prefabNames = new List<string>();   //Bcoz - We need to create an Instance of it before we use it.

    if (groupList[g].no1 > 0)
    {
        string[] files = Directory.GetFiles("Assets/Prefabs/Environment/" + prefabPaths[groupList[g].no1], "*.prefab", SearchOption.AllDirectories);
        foreach (string file in files)
        {
            prefabNames.Add(Path.GetFileName(file));
            //Debug.Log(prefabNames);
        }
    }

    //Getting the Each Individual Prefab present inside the Folders (Ex. Barrires/abc.prefab).
    groupList[g].no2 = EditorGUILayout.Popup(groupList[g].no2, prefabNames.ToArray());     // Dropdown - Individual Assets.

    if (GUILayout.Button("+", GUILayout.Width(50)) && groupList[g].no2 > 0)
    {
        string prefabPath = "Assets/Prefabs/Environment/" + prefabPaths[groupList[g].no1] + "/" + prefabNames[groupList[g].no2];
        GameObject initialPrefab = AssetDatabase.LoadAssetAtPath<GameObject>(prefabPath);
        GameObject child = (GameObject)PrefabUtility.InstantiatePrefab(initialPrefab);
        child.transform.parent = AssetGroupObjects[g].transform;    //Parenting Child into the Specific Group.
        //prefabInstances = new List<AssetInstance>();
    }
}
GUILayout.EndHorizontal();
```

- Next, we get Each Prefab from the List and then implement further functionalities.
- So, we first get the individual prefab and under it Add and Dropdown list and inside it, add functionalities like "*Select*", "*Duplicate*" and "*Delete*".

```
//IndividualAssetOptions();
int AssetCount = AssetGroupObjects[g].transform.childCount;     //Get Count of Assets Under Group from Hierarchy.

for (int i = 0; i < AssetCount; i++)    //Adds Group into AssetInstance.
{
    prefabInstances.Add(new AssetInstance(AssetGroupObjects[g].transform.GetChild(i).gameObject));
}

for (int i = 0; i < AssetCount; i++)    //Get Individual Assets from the AssetInstance List.
{
    GUILayout.BeginVertical(EditorStyles.helpBox);
    {
        prefabInstances[i].expanded = EditorGUILayout.Foldout(prefabInstances[i].expanded, prefabInstances[i].instance.name);
        if (prefabInstances[i].expanded)
        {
            //Collapses the Active Asset, if we open any other Asset.
            for (int k = 0; k < AssetCount; k++)
            {
                if (prefabInstances[k].instance.name != prefabInstances[i].instance.name && prefabInstances[k].expanded == true)
                {
                    prefabInstances[k].expanded = false;
                }
            }

            prefabInstances[i].instance.name = GUILayout.TextField(AssetGroupObjects[g].transform.GetChild(i).name); //Asset TextBox
            AssetGroupObjects[g].transform.GetChild(i).name = prefabInstances[i].instance.name;
```

```
                GUILayout.BeginHorizontal();
                {
                    if (GUILayout.Button("Select"))
                    {
                        if (Selection.activeGameObject != null)
                        {
                            Debug.Log(Selection.activeGameObject.name);
                        }
                        SceneView.lastActiveSceneView.LookAt(AssetGroupObjects[g].transform.GetChild(i).position);
                        Selection.activeGameObject = AssetGroupObjects[g].transform.GetChild(i).gameObject;
                        Debug.Log(Selection.activeGameObject.name);
                    }

                    if (GUILayout.Button("Duplicate"))
                    {
                        GameObject prefabRoot = PrefabUtility.GetCorrespondingObjectFromSource(AssetGroupObjects[g].transform.GetChild(i).gameObject);
                        string prefabPath = PrefabUtility.GetPrefabAssetPathOfNearestInstanceRoot(prefabRoot);
                        GameObject initialPrefab = AssetDatabase.LoadAssetAtPath<GameObject>(prefabPath);
                        GameObject child = (GameObject)PrefabUtility.InstantiatePrefab(initialPrefab);
                        child.transform.parent = AssetGroupObjects[g].transform;
                        //prefabInstances = new List<AssetInstance>();
                    }

                    if (GUILayout.Button("Delete"))
                    {
                        GameObject.DestroyImmediate(AssetGroupObjects[g].transform.GetChild(i).gameObject);
                        //prefabInstances = new List<AssetInstance>();
                        //prefabInstances.Remove(AssetGroupObjects[g].transform.GetChild(i).gameObject);
                    }
                }
                GUILayout.EndHorizontal();
            }
        }
        GUILayout.EndVertical();

    }
    GUILayout.EndVertical();

}
}
}
}
}
}
```

- Meanwhile, within the script we create Horizontal and Vertical Divisions by using the "*Begin*" and "*End*" Horizontal and Vertical functions.
- And then we end the script and call it and their respective functions in the "*LevelEditor*" script.

❖ LevelEditor.

- This is the Main Script that controls or implements the Functionalities created in other sub scripts; "*FoldOutObject*" and "*AssetsEditorCustom*".
- First, we create a function called "*ShowEditor*()" that shows or displays the Editor in the Scene.
- Then we add the "*OnEnable*()" function, where we instantiate all the scripts that we have created.

```
LevelEditor.cs U ×   AssetsEditorCustom.cs U   AssetEditor.cs U   FoldOutObject.cs U
Assets > Scripts > Level > Editor > LevelEditor.cs > LevelEditor > OnEnable()
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;

3 references
public class LevelEditor : EditorWindow
{

    2 references
    private Vector2 scroll;
    3 references
    private int switchEditor;
    2 references
    private GUIEditor guiEditor;
    2 references
    private FoldOutEditor foldOutEditor;
    2 references
    private LayoutEditor layoutEditor;
    2 references
    private AssetEditor assetEditor;
    2 references
    private AssetsEditorCustom assetsEditorCustom;


    [MenuItem("NIIT/Level Editor")]
    0 references
    static void ShowEditor()
    {
        LevelEditor window = (LevelEditor)EditorWindow.GetWindow(typeof(LevelEditor));  //Also can be written as:- LevelEditor window = EditorWindow.GetWindow<LevelEditor>();
        window.Show();  //Shows the Window.
    }

    /// <summary>
    /// This function is called when the object becomes enabled and active.
    /// </summary>
    0 references
    void OnEnable()
    {
        guiEditor = new GUIEditor();
        foldOutEditor = new FoldOutEditor();
        layoutEditor = new LayoutEditor();
        assetEditor = new AssetEditor();
        assetsEditorCustom = new AssetsEditorCustom();
    }

```

- Then, we create another function called "*OnGUI*()" which constantly updates the Interface we created.
- And in that, we add a Dropdown Menu that consists of all the Custom Editors we created.
- Then we use a "*Switch*" function to switch between the Editors.
- And in the "*Cases*" of the Switch function, we add the "*Render*()" common function which we created in each Editor Script, from its instantiated references.

```
    /// <summary>
    /// OnGUI is called for rendering and handling GUI events.
    /// This function can be called multiple times per frame (one call per event).
    /// </summary>
    0 references
    void OnGUI()
    {
        scroll = EditorGUILayout.BeginScrollView(scroll);   //Start Scroll

            switchEditor = EditorGUILayout.Popup("Editors", switchEditor, new string[] {"GUI", "FoldOut", "Layout Editor", "Asset Editor", "Assets Editor Custom"});

            switch(switchEditor)
            {
                case 0:
                guiEditor.RenderGUI();
                break;

                case 1:
                foldOutEditor.RenderFoldOut();
                break;

                case 2:
                layoutEditor.Render();
                break;

                case 3:
                assetEditor.Render();
                break;

                case 4:
                assetsEditorCustom.Render();
                break;

                default:
                break;
            }

        EditorGUILayout.EndScrollView();    //End Scroll
    }
}

```

----------THE END----------