第二次实验： 拓展编辑器

学号：22920212204392 姓名：黄勖

一、 实验目的

- 熟悉 Unity 的编辑器拓展 API 接口
- 学习修改系统自带编辑器
- 学习定义自己的编辑器

二、 实验条件

- 系统环境：Windows 10 21H2
- 软件环境：Unity 3D 2021.3.14f1c1

三、 实验内容

- 在 Scene 视图中增加一个全局坐标系的辅助元素
  - 三个轴分别用红绿蓝三根直线表示，尾端加一个立方体
  - 创建一个自定义菜单用于显示和隐藏全局坐标系
- 延伸任务：物体被选中时，Inspector 视图的 Transform 组件增加设置辅助坐标系的坐标轴长度、尾端立方体大小的属性

四、 实验步骤：

（注：代码工程第一次打开需要先在菜单打开全局坐标系）

为了实现实验内容，主要的工作是编写 GlobalCoordinateSystem.cs 编辑器拓展脚本，主要内容与注释如下：

- 初始化变量与准备工作（使用 InitializeOnLoadMethod() 方法，调用 UnityEditor 实现，实现在 Scene 视图中增加一个全局坐标系的辅助元素）

```
Assets > Editor > GlobalCoordinateSystem.cs > GlobalCoordinateSystem
1   using UnityEditor; //用于在编辑器中自定义菜单和编辑器界面
2   using UnityEngine;
3   using System.Reflection; //System.Reflection命名空间可以在运行时动态获取程序集信息
4
5   [InitializeOnLoad] //在Unity编辑器启动时加载
        0 references
6   public class GlobalCoordinateSystem : EditorWindow
7   {
            3 references
8       private static bool _isEnabled = false; //表示全局坐标系是否启用
            7 references
9       private static GameObject _axes; //全局坐标系的GameObject对象
            0 references
10      private static Transform _selectedAxis; //选中的轴的Transform对象
            8 references
11      private static float _axisLength = 5f; //坐标轴长度，默认为5
            11 references
12      private static float _cubeSize = 0.5f; //尾端立方体大小，默认为0.5
```

- 定义菜单项，用于启用和关闭全局坐标系

```
14      [MenuItem("拓展全局坐标系_黄勖/Toggle")] //定义菜单项，用于启用和关闭全局坐标系
            0 references
15      public static void Toggle()
16      {
17          _isEnabled = !_isEnabled; //切换全局坐标系的启用状态
18          Debug.Log("Toggle"); //测试
19          if (_isEnabled)
20          {
21              CreateAxes(); //启用全局坐标系
22          }
23          else
24          {
25              DestroyAxes(); //关闭全局坐标系
26          }
27      }
```

- 创建全局坐标系

```
29      private static void CreateAxes() //创建全局坐标系
30      {
31          _axes = new GameObject("Global Coordinate System"); //创建Global Coordinate System的GameObject
32
33          //创建X轴
34          var x = new GameObject("X");
35          x.transform.SetParent(_axes.transform); //设置X的父物体为全局坐标系
36          var lineX = x.AddComponent<LineRenderer>(); //添加LineRenderer组件
37          lineX.material = new Material(Shader.Find("Sprites/Default")); //设置材质球
38          lineX.startColor = Color.red; //设置线段起始颜色
39          lineX.endColor = Color.red; //结束颜色
40          lineX.startWidth = 0.1f; //线段起始宽度
41          lineX.endWidth = 0.1f; //线段结束宽度
42          lineX.SetPosition(0, Vector3.zero); //线段起始点为(0,0,0)
43          lineX.SetPosition(1, new Vector3(_axisLength, 0f, 0f)); //线段结束点为(_axisLength,0,0)
44          Selection.activeObject = lineX; //将当前对象设置为选中的对象
45          var cubeX = GameObject.CreatePrimitive(PrimitiveType.Cube); //创建方体
46          cubeX.transform.localScale = new Vector3(_cubeSize, _cubeSize, _cubeSize);//设置立方体缩放大小
47          cubeX.transform.SetParent(x.transform);//将立方体设置为X轴的子物体
48          cubeX.transform.localPosition = new Vector3(_axisLength, 0f, 0f);//设置立方体位置
```

```
        //创建Y轴
        var y = new GameObject("Y");
        y.transform.SetParent(_axes.transform);
        var lineY = y.AddComponent<LineRenderer>();
        lineY.material = new Material(Shader.Find("Sprites/Default"));
        lineY.startColor = Color.green;
        lineY.endColor = Color.green;
        lineY.startWidth = 0.1f;
        lineY.endWidth = 0.1f;
        lineY.SetPosition(0, Vector3.zero);
        lineY.SetPosition(1, new Vector3(0f, _axisLength, 0f));
        var cubeY = GameObject.CreatePrimitive(PrimitiveType.Cube);
        cubeY.transform.localScale = new Vector3(_cubeSize, _cubeSize, _cubeSize);
        cubeY.transform.SetParent(y.transform);
        cubeY.transform.localPosition = new Vector3(0f, _axisLength, 0f);

        //创建Z轴
        var z = new GameObject("Z");
        z.transform.SetParent(_axes.transform);
        var lineZ = z.AddComponent<LineRenderer>();
        lineZ.material = new Material(Shader.Find("Sprites/Default"));
        lineZ.startColor = Color.blue;
        lineZ.endColor = Color.blue;
        lineZ.startWidth = 0.1f;
        lineZ.endWidth = 0.1f;
        lineZ.SetPosition(0, Vector3.zero);
        lineZ.SetPosition(1, new Vector3(0f, 0f, _axisLength));
        var cubeZ = GameObject.CreatePrimitive(PrimitiveType.Cube);
        cubeZ.transform.localScale = new Vector3(_cubeSize, _cubeSize, _cubeSize);
        cubeZ.transform.SetParent(z.transform);
        cubeZ.transform.localPosition = new Vector3(0f, 0f, _axisLength);

        Selection.activeGameObject = _axes;
    }
```

- 销毁坐标轴的游戏对象

```
    2 references
    private static void DestroyAxes() //销毁坐标轴的游戏对象
    {
        if (_axes != null)
        {
            DestroyImmediate(_axes);
        }
    }
```

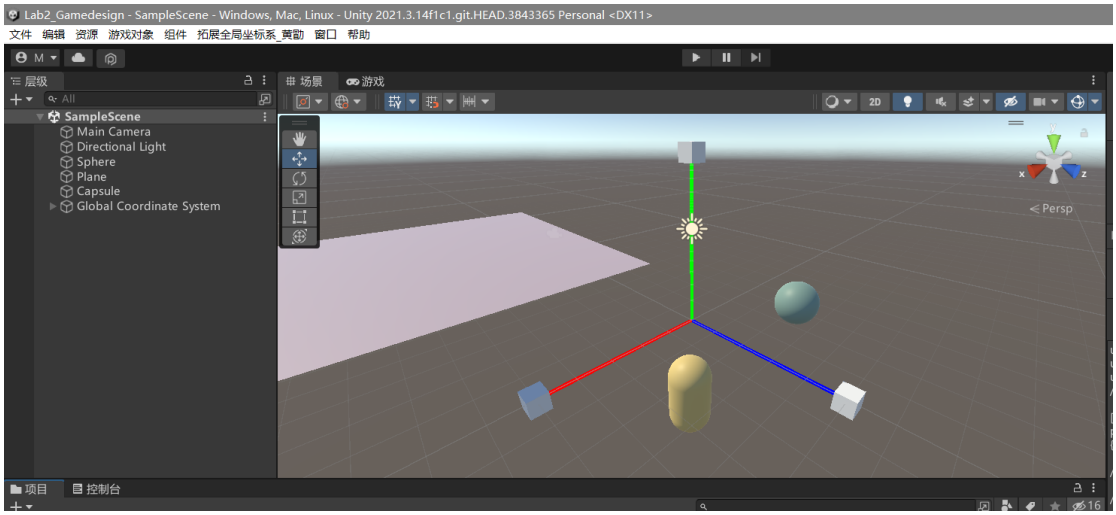
- 使用[CustomEditor(typeof(Transform))]，在 Inspector 视图的 Transform 组件增加设置辅助坐标系的坐标轴长度、尾端立方体大小的属性

```
//自定义 Transform 组件
[CustomEditor(typeof(Transform))]
0 references
public class GlobalCoordinateSystem_inspect : Editor
{
    2 references
    private Editor m_Editor;//存储基础的 Transform 组件的编辑器
    // 当激活该编辑器时，创建一个基础的 Transform 组件的编辑器
    0 references
    void OnEnable()
    {
        m_Editor = Editor.CreateEditor(target,
            Assembly.GetAssembly(typeof(Editor)).GetType("UnityEditor.TransformInspector", true));
    }
```

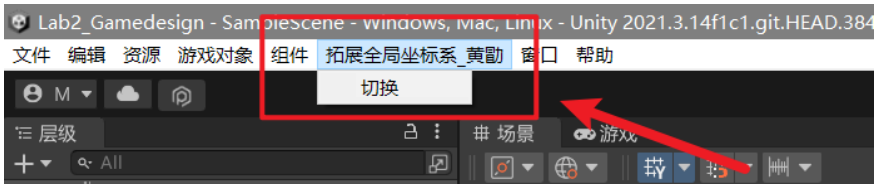
```
105         // 在编辑器中绘制自定义的 Transform 组件的属性
            16 references
106         public override void OnInspectorGUI()
107         {
108             // 绘制自定义标签
109             GUILayout.Label("拓展Transform : 修改全局坐标系");
110             GUILayout.Label("22920212204392 黄勖");
111
112             // 使用 EditorGUILayout 绘制一个可以编辑的浮点数，用于控制坐标轴长度
113             EditorGUI.BeginChangeCheck();
114             var newAxisLength = EditorGUILayout.FloatField("坐标轴长度", _axisLength);
115
116             // 用于控制尾端立方体的大小
117             var newCubeSize = EditorGUILayout.FloatField("尾端立方体大小", _cubeSize);
118
119             // 如果用户编辑了任意一个值，重新生成坐标轴
120             if (EditorGUI.EndChangeCheck())
121             {
122                 _axisLength = newAxisLength;
123                 _cubeSize = newCubeSize;
124
125                 // 销毁旧的坐标轴
126                 DestroyAxes();
127
128                 // 生成新的坐标轴
129                 CreateAxes();
130             }
131
132             // 调用 Unity 默认的 Transform 组件的编辑器
133             m_Editor.OnInspectorGUI();
```

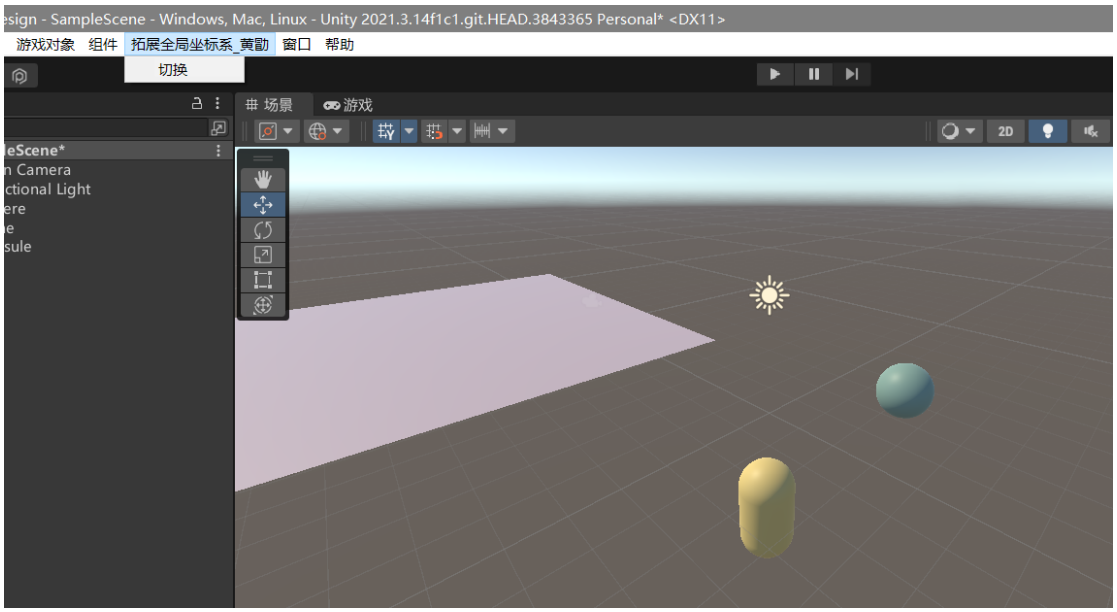
结果展示：

- 在 Scene 视图中增加一个全局坐标系的辅助元素
  - 三个轴分别用红绿蓝三根直线表示，尾端加一个立方体

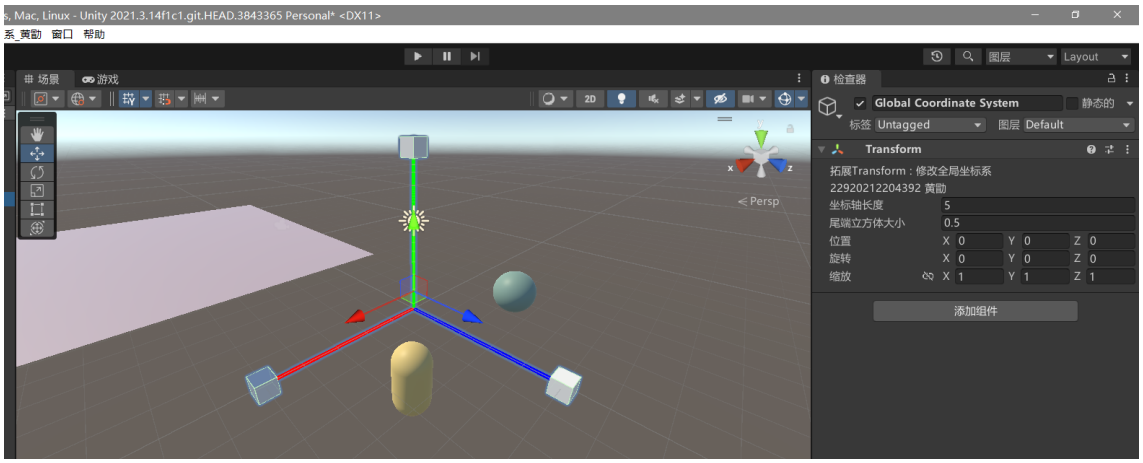


- 创建一个自定义菜单用于显示和隐藏全局坐标系

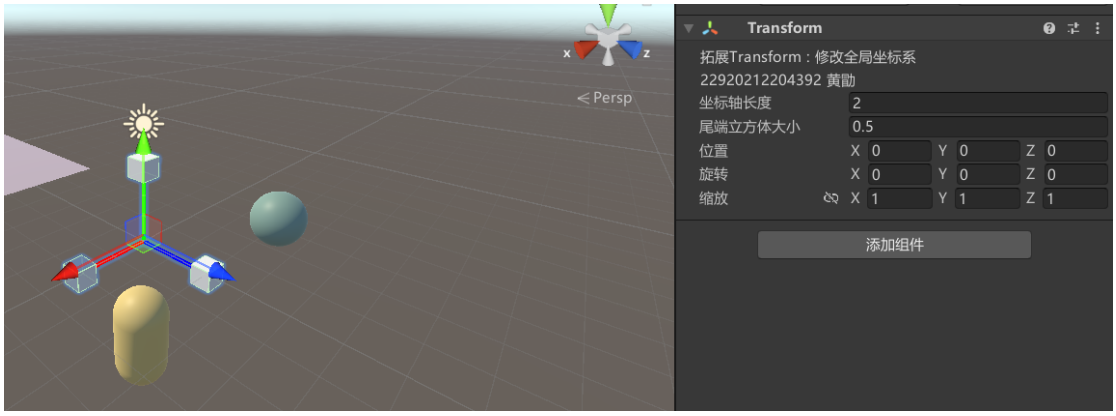




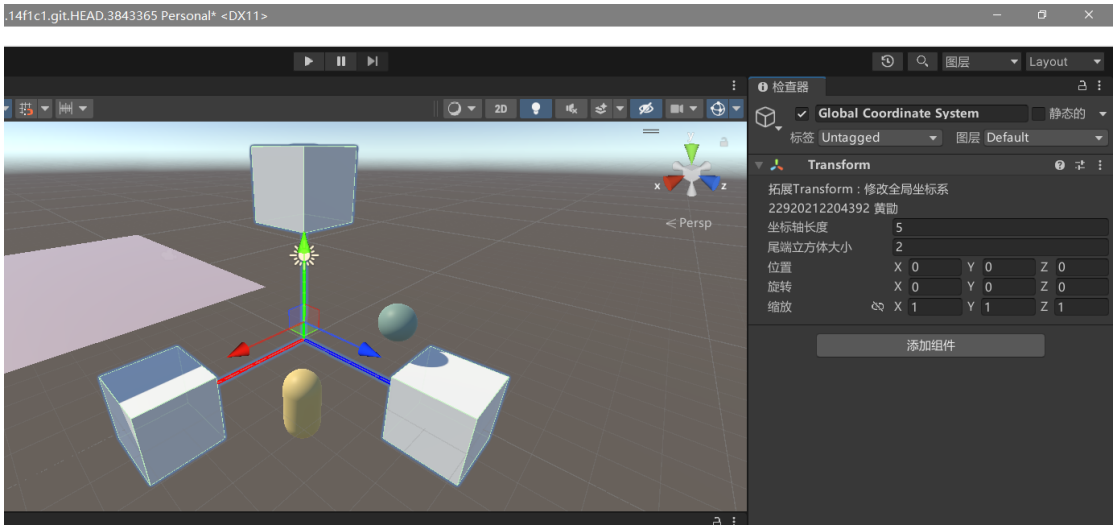
- 延伸任务：物体被选中时，Inspector 视图的 Transform 组件增加设置辅助坐标系的坐标轴长度、尾端立方体大小的属性



可以将长度修改为 2，此时坐标轴变短：



也可以修改立方体大小：



五、 实验心得总结：

本次实验的主要目的是了解 Unity 的编辑器扩展功能，以及如何使用 Unity API 创建自定义的编辑器工具。在本次实验中，我使用 InitializeOnLoadMethod()方法和 UnityEditor API 来创建一个全局坐标系的辅助元素，并且添加了一些属性和自定义菜单来改变全局坐标系的行为。下面是本次实验的总结：

1. 使用 InitializeOnLoadMethod()方法可以在 Unity 加载时调用某个静态方法，这个方法可以用来初始化编辑器扩展。我们可以在这个方法中创建编辑器工具，添加菜单项，或者在场景中创建游戏对象等。
2. 在编辑器扩展中，可以使用 UnityEditor 命名空间下的类来创建自定义菜单、工具栏按钮和 Inspector 窗口等。其中，CustomEditor 和 CustomPropertyDrawer 是两个最常用的类，它们分别用于自定义组件的 Inspector 界面和属性的显示方式。
3. 使用 Handles 类可以在 Scene 视图中绘制自定义的图形元素，例如线段、立方体和球体等。可以使用 Gizmos 类来绘制一些基本的图形元素，例如边框框、箭头和文本等。

本次实验让我了解了如何使用 Unity 编辑器扩展来创建自定义的工具，使得学习设计变得更加高效和方便。在今后的开发中，我将继续使用这些技巧来提高我的效率。