Unity Editor

需要使用编辑器类代码时，需要在脚本前面加上 using UnityEditor 命名空间。

一. Editor 文件夹

Editor 文件夹可以放在工程项目的任何文件夹下，一个工程项目中可以存在多个"Editor"文件夹。编辑器扩展相关的脚本都要放在该文件夹内，该文件夹中的脚本只会对 Unity 编辑器起作用。

在项目打包的时候，Editor 文件夹中的内容不会被打包到项目中。如果与 Unity 编辑器相关脚本不放在该文件夹中，打包项目时可能会出错。如果非要有些编辑器相关脚本不放在该文件夹中，需要在该类的前后加上 UNITY_EDITOR 的宏定义。

二. Editor Default Resources

该文件夹需要放在 Assets 根目录下，用来存储编辑器所需要的图片等资源，书写的时候需要注意中间有空格隔开。在进行项目打包时，此文件夹也不会被打包，访问方法为：EditorGUIUtility[1].Load()。

也可以在 Editor 文件夹内创建一个 Resources 文件夹，将相关资源放在该文件夹内，通过 Resources.Load()获取文件夹中的资源，也是可以的。

只能有一个 Editor Default Resources 文件夹，且必须将其放在项目的根目录；直接位于 Assets 文件夹中。将所需的资源文件放在此 Editor Default Resources 文件夹内或其中的子文件夹内。如果资源文件位于子文件夹中，请始终在传递给 EditorGUIUtility.Load 函数的路径中包含子文件夹路径。

三. Gizmos 文件夹

Gizmos 文件夹也需要放在 Assets 根目录下，可以用来存放 Gizmos.DrawIcon()的图片资源。Gizmos 允许将图形添加到 Scene 视图，以帮助可视化不可见的设计细节。Gizmos.DrawIcon 函数在场景中放置一个图标，作为特殊对象或位置的标记。必须将用于绘制此图标的图像文件放在名为 Gizmos 的文件夹中，这样才能被 DrawIcon 函数找到。

只能有一个 Gizmos 文件夹，且必须将其放在项目的根目录；直接位于 Assets 文件夹中。将所需的资源文件放在此 Gizmos 文件夹内或其中的子文件夹内。如果资源文件位于子文件夹中，请始终在传递给 Gizmos.DrawIcon 函数的路径中包含子文件夹路径。

四. MenuItem

MenuItem 属性允许我们添加菜单项到主菜单和检视面板的上下文菜单。 (该属性把任意静态函数变为一个菜单命令。仅静态函数能使用这个 MenuItem 属性。)

注意：MenuItem 是编辑器类，所以需要引用 using UnityEditor; 命名空间，且一般我们的类也不是集成自 MonoBehaviour 的。

要将新的菜单项添加到 Unity 的菜单栏，我们只需在项目中的任何类中添加一个方法，并使用属性 static 对其进行装饰。MenuItem 方法必须为 static，但可以 private。下面的示例将展示提供带有菜单项的新菜单游戏辅助工具。

PS：

```
[MenuItem("Tools/游戏/游戏辅助工具", false, 101)]
        static void Init()
        {
            GameHelperEditorWindow window =
(GameHelperEditorWindow)EditorWindow.GetWindow(typeof(GameHelperEditorWindow),
```

[1] Utility adj.有用的，多功能的 n.应用程序；实用

```
false, "游戏辅助工具");
            window.Show();
        }
```

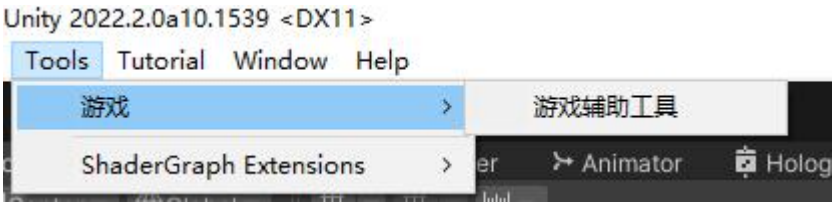

图 1.　　使用 MenuItem 构建的辅助工具菜单

1. 工作原理

当 Unity 重新编译脚本时，它会搜索项目中的所有脚本并查找使用该 MenuItem 属性修饰的方法。当它找到一个时，它会创建一个具有给定路径的新菜单项，并在您单击该菜单项时执行方法中的代码。

2. 嵌套

MenuItem 是可以进行嵌套的。要嵌套菜单项，我们只需用 ”/”在菜单路径中添加一个，它就会为您的菜单添加另一个子菜单！

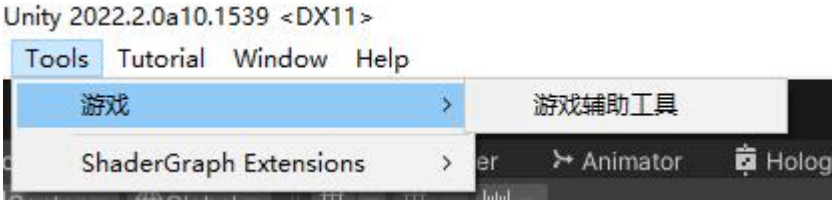

图 2.　　使用 MenuItem 构建的辅助工具菜单

如上图所示，我就是在 Tools 这个大类下，新增了一个子菜单，即游戏。在游戏下面又新增了一个子菜单，即游戏辅助工具。

具体可以嵌套多少个，我目前还没有去进行尝试。

五. 自定义快捷键

指令	含义
%	Windows 上为 Ctrl 键，macOS 上为 cmd
#	Shift 键
&	Alt 键
LEFT/RIGHT/UP/DOWN	方向键的上下左右
F1-F12	键盘上的 F1——F12
HOME/END/PGUP/PDDN	对应键盘的 Home/End/PageUp/PageDown
_+字母（字母和其他进行组合时不需要下划线）	表示单一按键

PS：

```
[MenuItem("Tools/New Option %#a"]//CTRL 键+SHIFT 键+A 键
privte static void NewMenuOption()
{
}
[MenuItem("Tools/Item %g")]//CTRL 键+G 键
private static void NewNestedOption()
{
```

```
}
[MenuItem("Tools/Item2 _g")]//G 键
private static void NewOptionWithHotKey()
{
}
```

六. UnityEditorWindow

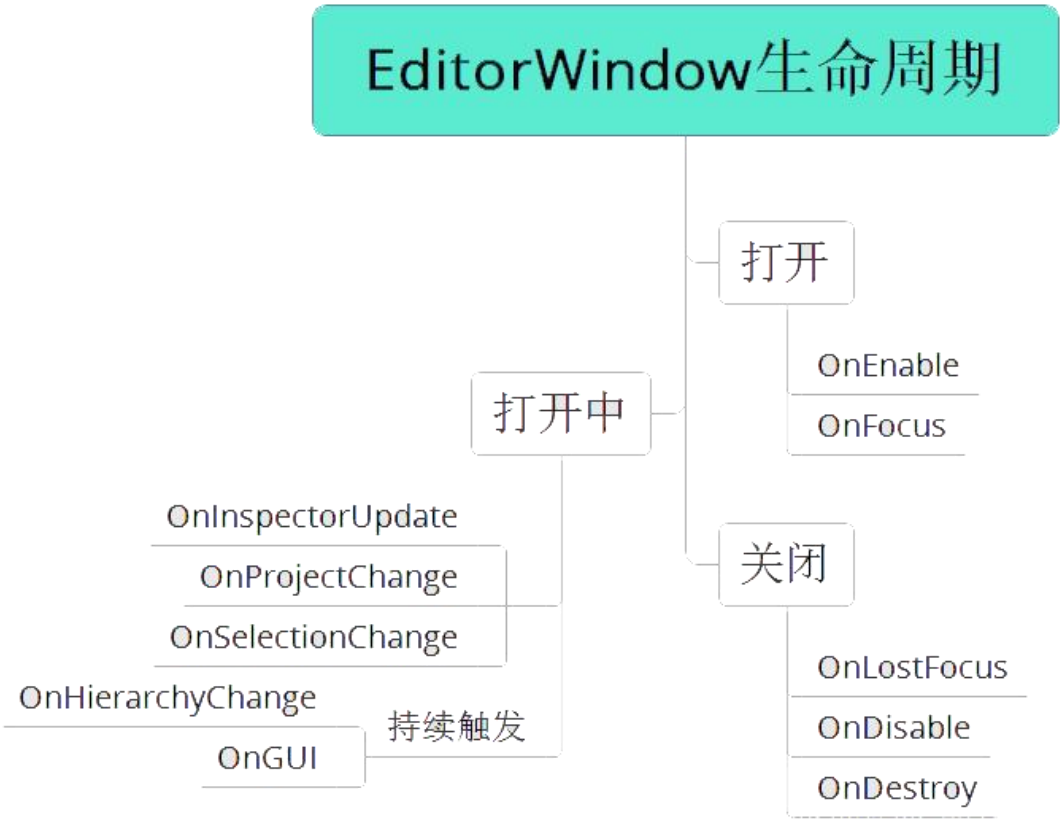

图 3. EditorWindow 生命周期

1. EditorWindow 生命周期

OnEnable()	当打开界面的时候调用
OnFocus()	当该窗口被聚焦（点击该窗口）
OnGUI()	当渲染 UI 的时候调用
OnSelectionChange()	当选择发生更改时调用，选中的可选项（在 Project 和 Hierarchy 视图中）
OnLostFocus()	从该窗口离开时调用（点击非窗口外其他地方）
OnInspectorUpdate()	当属性界面更新时，几乎一直在更新
OnHierarchyChange()	Hierarchy 界面改变（增加、减少物体）
OnProjectChange()	在 Project 视图删除、增加文件
OnDisable()	隐藏的时候调用
OnDestroy()	销毁的时候调用
OnValidate()	拖拽式赋值时调用

七. UnityEditorInternal 命名空间

1. 可重新排序的列表-ReorderableList

如果我们在代码中直接使用 List，那么 List 里面元素的位置是无法在 Inspector 窗口中进行修改的。

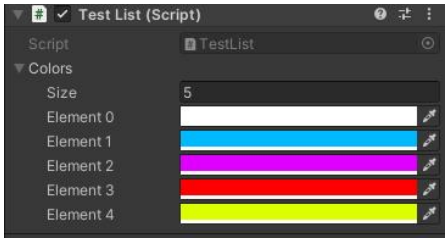

图 4. 默认的 List 我们无法任意改变元素位置

在 UnityEditorInternal 命名空间下提供 ReorderableList 类，它可以实现通过拖曳来重新排列列表元素的目的。我们所需要做的，就是编写一个自定义的编辑器来使用它。

图 5. 在 Editor 中利用 ReorderableList，实现可任意拖动的 List

1) 构造函数的参数

参数名称	功能描述
dragable	是否可拖曳排序
displayHeader	是否显示 Header
displayAddButton	是否显示添加元素按钮
displayRemoveButton	是否显示移除元素按钮

2) 回调方法

ReorderableList 具有多个回调函数，我们可以在 Editor 中使用这些回调函数，自定义事件完成后的行为。

回调函数名称	功能描述
drawHeaderCallback	绘制表头时的回调
drawFooterCallback	绘制尾部时的回调
drawElementCallback	绘制元素时的回调
drawElementBackgroundCallback	绘制元素背景时的回调
onReorderCallback	元素发生重新排序时的回调
onSelectCallback	选中元素时响应的回调
onAddCallback	新增一个元素时响应的回调
onRemoveCallback	移除选择的元素时执行的回调
onMouseUpCallback	选择元素后，鼠标抬起时的回调
onCanRemoveCallback	是否显示可移除按钮回调

onChangedCallback	列表数据发生改变时响应的回调

八. Unity Undo 详解

Ctrl + z 这对按键组合应该为广大计算机使用者所熟知，即用来做撤销操作。在没有热键屏蔽的情况下(搜狗输入法，QQ 等软件可能会抢占某些常用按键组合的优先使用权从而导致我们按下按钮发现没有反应)，Unity 也可以通过 Ctrl+z 的组合来进行撤销。

比如我们在场景中新建一个 Cube，按下 Ctrl+ z（或者菜单栏的 Edit->Undo），新建的这个 Cube 就从场景中消失了。

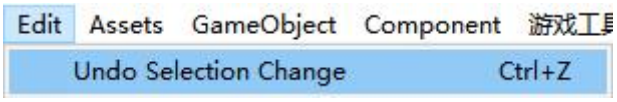

图 6. 使用 Unity 里的 Edit 选项卡进行撤销

1. Undo 的机制

Unity 中 Undo 使用的数据结构基于 Stack（栈），采用 LIFO（Last In First Out）的策略，越晚进栈的越早被弹出。

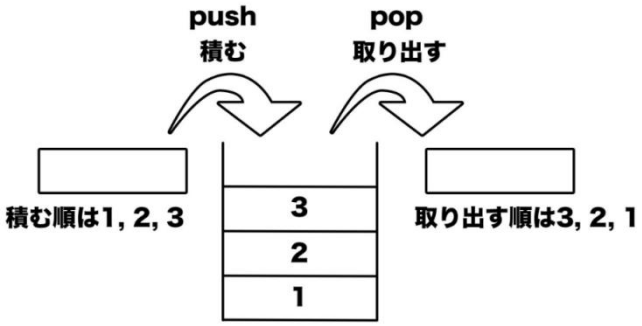

图 7. Undo 堆栈

2. 创建一个用于 Undo 的对象

下面的代码是用于生成立方体。

PS：

```
using UnityEngine;
using UnityEditor;
public class Example
 {
     [MenuItem("Example/创建立方体")]
     static void CreateCube ()
      {
          GameObject.CreatePrimitive (PrimitiveType.Cube);
      }
}
```

当我们按照之前的方式进行撤销时，发现 Ctrl+z 无法撤销我们的创建操作。由于我们单纯地用代码来创建这么一个 Cube，Undo 的栈并没有将这一操作记录进去，所以这个操作就无法撤销。

针对创建新 GameObject 的操作，我们使用 Undo.RegisterCreatedObjectUndo 来记录之前

的状态。

现在，我们重新创建一个新的场景（为了避免使用之前的 Undo 栈）

PS：

```
using UnityEngine;
using UnityEditor;
public class Example
 {
     [MenuItem("Example/创建立方体")]
     static void CreateCube ()
      {
          var cube = GameObject.CreatePrimitive (PrimitiveType.Cube);
          Undo.RegisterCreatedObjectUndo (cube, "Create Cube");
      }
 }
```

我们使用上面的代码来创建新的 Cube，这次使用 Undo.RegisterCreatedObjectUndo (cube, "Create Cube");记录了状态，所以可以撤销创建新 Cube 的操作了。

3. 撤销游戏对象属性的变化

下面给出一段旋转 GameObject 的代码，Example/Random Rotate 可以让物体旋转一个随机的角度。

PS：

```
using UnityEngine;
using UnityEditor;
public class Example
 {
     [MenuItem("Example/Random Rotate")]
     static void RandomRotate ()
      {
          var transform = Selection.activeTransform;
          if (transform)  {
               transform.rotation = Random.rotation;
          }
      }
 }
```

上面的代码里没有 Undo 相关的代码，所以并不能撤销随机角度旋转的操作。

这里我们使用 Undo.RecordObject 来记录 Object 的属性。

PS：

```
using UnityEngine;
using UnityEditor;
public class Example
 {
     [MenuItem("Example/Random Rotate")]
     static void RandomRotate ()
      {
          var transform = Selection.activeTransform;
```

```
            if (transform)  {
                Undo.RecordObject (transform, "Rotate " + transform.name);
                transform.rotation = Random.rotation;
            }
        }
    }
```

在执行了旋转之后，我们再次按下 Ctrl+z！这一次，立方体的旋转操作被撤销了！

4. 撤销 UnityEngine.Object

下面来讲讲继承自 UnityEngine.Object 对象的撤销（继承自 UnityEngine.Object 的对象都是可序列化的）

经常 Undo 的对象包括下面三种：

① 游戏对象

② Component（也包括 MonoBehaviour）

③ ScriptableObject

当要 Undo 被 System.Serializable 修饰的属性的时候，执行 Example/Change 操作，PlayerInfo 被撤销回到之前的状态了。

PS：

```
[System.Serializable]
public class PlayerInfo
 {
    public string name;
    public int hp;
}
```

示例 Player

```
using UnityEngine;
public class Player : MonoBehaviour
 {
    [SerializeField]
    PlayerInfo info;
}
using UnityEngine;
using UnityEditor;
public class Example
 {
    [MenuItem("Example/Change PlayerInfo")]
    static void ChangePlayerInfo ()
     {
        var player = Selection.activeGameObject.GetComponent<Player> ();
        if (player)  {
            Undo.RecordObject (player, "Change PlayerInfo");
            player.info = new PlayerInfo {
                name = "New PlayerName",
                hp = Random.Range(0,10)
            } ;
```

```
            }
        }
    }
```

5. Revert

Revert 是和 Record 相反的操作，相当于 Ctrl+z。

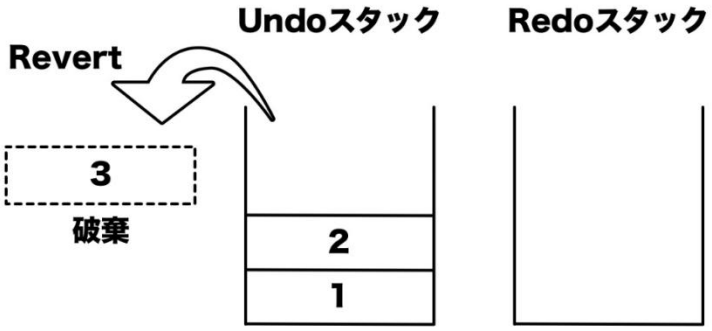

图 8. Revert 图解

6. Undo 的类型

RecordObject（s）	记录 Object 的状态
AddComponent	要添加 Component 的时候使用，可以撤销掉新加的 Component
RegisterCreatedObjectUndo	新建 Object 的时候使用，可以撤销新建的物体
DestroyObjectImmediate	要删除 Object 的时候使用，可以撤销删除操作
SetTransformParent	修改 Transform 的 Parent 的时候使用，可以恢复层次结构

九. Unity GUI 系统

1. GUI 系统

命名空间 UnityEngine 属于运行时 API，也可用于编辑器，只能单纯的绘制控件，没有布局效果，需要手写每个控件的位置和大小。

PS：

```
private void OnGUI()
{
    GUI.Button(new Rect(0, 0, 100, 30), "Button");
    GUI.Label(new Rect(100, 0, 100, 30), "Label");
    GUI.PasswordField(new Rect(0, 30, 100, 30), "Password", '$');
    GUI.Toggle(new Rect(100, 30, 100, 30), true, "Toggle");
}
```

2. GUILayout 系统

同 GUI，命名空间 UnityEngine 属于运行时 API，也可用于编辑器，相对于 GUI 系统，它自带布局系统，可以很好的控制布局和位置。

PS：

```
private void OnGUI()
{
    GUILayout.BeginHorizontal("Box"); //开始一个水平布局
    GUILayout.Button("Button");
```

```
        GUILayout.Label("Label");
        GUILayout.PasswordField("Password", '$');
        GUILayout.Toggle(true, "");
        GUILayout.FlexibleSpace();          //创建一个自适应的空白区域，也即是填满本次布局中的这部分空间
        GUILayout.EndHorizontal();          //结束一个水平布局
    }
```

3. EditorGUI 系统

命名空间 UnityEditor 属于编辑器 API，只能在编辑器下使用，无法在运行时使用。绘制的控件，没有布局效果，需要手写每个控件的位置和大小，同 GUI。但是比 GUI 系统的功能稍微多一些，开发自由度高一些。

PS：

```
private void OnGUI()
{
    EditorGUI.ColorField(new Rect(0, 0, 100, 30), Color.red);
    EditorGUI.DoubleField(new Rect(0, 30, 100, 30), 10);
    EditorGUI.ProgressBar(new Rect(0, 60, 100, 30), 0.5f, "ProgressBar");
}
```

4. EditorGUILayout 系统

命名空间 UnityEditor 属于编辑器 API，只能编辑器下使用，无法运行时使用。自带布局效果，同 GUILayout。拥有很多编辑器专属空间，开发自由度高。

PS：

```
private void OnGUI()
{
    EditorGUILayout.BeginHorizontal("Box");     //开始一个水平布局
    EditorGUILayout.LabelField("LabelField");
    EditorGUILayout.PasswordField("PasswordField");
    EditorGUILayout.RectField(Rect.zero);
    EditorGUILayout.EndHorizontal();            //结束一个水平布局
}
```

总结：Editor 和运行时使用不同的 GUI，Layout 版本的 GUI 带有布局系统。另外编辑器下，以上四个系统可以混合使用。但总得来说，带有 Layout 的使用居多。