**窗口基础知识点**

Hierarchy层级窗口：显示一个场景中的所有对象对这些对象进行统一管理

Scene场景窗口：场景内对象显示模拟

Game游戏窗口：通过摄像头/屏幕玩家可以观察到的内容

Project工程窗口：资源管理

Unity支持的类型 ：

|-图像：jpg,png,tga

|-模型：fbx,max,maya

|-音效：wav,mp3,ogg

|-文本：txt,json,bytes

|­-视频：mp4

Inspector检查窗口：显示具体参数

Console控制台窗口：调试用 **Ctrl+Shift+C**

**反射：**程序正在运行时可以查看其它程序集或者自身的元数据

GameObject是最底层的抽象类

场景的本质是配置文件

在同一个Hierarchy下可以叠加显示多个场景，便于Object拷贝，**开发者房间惯用手法**

脚本的创建类名和文件名必须一致，否则不好挂载（与java类似）

通过[DisallowMultipleComponent]可以防止一个Asset挂载了多个脚本

Unity自带的固定生命周期函数：

只调用一次的：

Awake：自己的类被调用时触发，类似构造函数

Start：被创建时，第一帧之前调用

OnDestroy：对象被销毁时调用

与循环无关，仅和对象失活激活有关：

OnEnable：每次激活触发

OnDisable：每次失活触发

循环触发：

FixedUpdate：逻辑帧更新，每帧（引擎内独立计帧）执行，进行物理更新

更改：ProjectSetting-Time

Update：处理游戏核心逻辑更新的函数

LateUpdate：一般用于摄像机位置更新相关内容，因为Update与LateUpdate之间引擎会做一些渲染相关操作

时间顺序：

生-Awake-OnEnable-Start-FixedUpdate-Update-LateUpdate-OnDisable-OnDestroy-死

**可序列化：**

私有和保护成员无法显示编辑

但是通过在申明变量前加上**[SerializeField]**可以强制显示

通过加上**[HideInInspector]**可以强制隐藏

大部分变量都可以显示，自定义数据结构前需要加**[System.Serializable]**并using System

辅助参数：

格式：

[Header(“<分组说明>”)]

[Tooltip(“<悬停说明内容>”)]

[Space()] （增加序列显示间隔）

参数：

[Range(<最小值>,<最大值>)] （添加有取值范围的滚动条）

[Multiline(5)] （多行显示字符串，缺省3行）

[TextAera(<最小行>,<最大行>)] （默认显示<最小行>，超过最大行变成滚动条显示字符串）

[ContextMenuItem(“<说明>”,”<方法名>”)]（简易右键方法触发器）

函数：

[ContextMenu(“<说明>”)] （脚本可选择额外测试方法）

[CustomEditor(typeof(<ClassName>))] （局内Editor API）

public class A:Editor

提供方法

public override void OnInspectorGUI()

{

base.DrawDefaultInspector()

if(GUILayout.Button(“<ButtonName>”))

{//do something}

}

**MonoBehavior主要内容：**

this.transform 位置参数

this.enabled 激活状态

public <ClassName> A获取别的脚本对象，并通过A.gameObject获取相关信息

this.GetComponent(“<ScriptName>”) as <ClassName> 通过名字获取，可为Null **（不靠谱）**

this.GetComponent(typeof(<ClassName>)) as <ClassName> 通过类获取

this.GetComponent<ClassName>() 泛型获取

————得到自己挂载的其它脚本

<ClassName>[] array = this.GetComponents<ClassName>()

List<ClassName> list = new List<ClassName>()

————获取多个脚本(返回数组)

this.GetComponentInChildren<ClassName>(<缺省false，意味着不查询失活子对象>)

this.GetComponentsInChildren<ClassName>()

————获取子对象脚本(也会查询自己身上是否挂载)

this.GetComponentInParent<ClassName>(<缺省false，意味着不查询失活子对象>)

this.GetComponentsInParent<ClassName>()

————获取父对象脚本(也会查询自己身上是否挂载)

this.TryGetComponent<ClassName>(out A) 含bool返回值，更加安全

**最小单位GameObject：**

**GameObject成员变量：**

.name

.activeSelf是否激活

.isStatic是否静态

.tag标签

.layer层级

**静态方法：**

CreatePrimitive(PrimitiveType.XXX) 创建自带几何体，带GameObject返回值，可用于引用

查找对象（只能找到激活的对象）

GameObject.Find(“<Name>”) 全局查找，效率较慢

GameObject.FindGameObjectWithTag(“<TagName>”) 根据Tag寻找对象

找多个对象同理

GameObject.FindObjectOfType<ClassName>()找到挂载了ClassName的某Object(效率很低，勿用)

**克隆并实例化对象：**

**(GameObject.)Instantiate(GameObject A)** 可以是预设体，也可以是场景已有对象

（在下一帧异步）删除对象：

(GameObject.)Destroy(GameObject A,<arg> Delay\_time)

(GameObject.)Destroy(this) 可用于删除脚本

（立即删除：GameObject.DestroyImmediate(GameObject A)）

**防止在切换场景时删除对象：**

**GameObject.DontDestroyOnLoad(this.gameObject)**

**GameObject成员方法：**

创建空物体：GameObject obj = new GameObject()

GameObject obj = new GameObject(“NewObjectsName”, typeof(<ClassName>),…)

————创建新的空对象

**动态添加继承MonoBehavior的脚本：**

**obj.AddComponent<ClassName>() 有同脚本类的返回值用于引用**

（由于同祖宗继承，也具有和MonoBehavior子类相同的引用脚本的方法）

标签比较：

this.gameObject.CompareTag(“<TagName>”), 具有bool返回值

设置激活/失活

obj.SetActive(false/true)

次要方法：(不建议使用)

通过广播让自己或别人执行某些行为方法（执行所有脚本所有同名方法）：

this.gameObject.SendMessage(“<FunctionName>”, <args>…)（全局遍历，效率低）

this.gameObject.BroadcastMessage(……)让自己和自己的子对象执行

this.gameObject.SendMessageUpwards(……)自己和自己的父对象

**时间相关：**

**Unity自带方法类：Time**

Time.timeScale = 0(停止) 1(正常) 2(2倍速)

**帧间隔时间：最近的一帧用了多长时间（单位 秒）**

**——主要用于计算位移，每一次update进行间隔时间\*速度的位移**

受scale影响：Time.deltaTime

不受scale影响的：Time.unscaledDeltaTime

游戏开始到现在的时间：Time.time, Time.unscaledtime

物理帧间隔时间：

Time.fixedDeltaTime, Time.fixedUnscaledDeltaTime

累计帧数：Time.frameCount

**Transform和Vector3：**

**位移**

以一个Vector3表示一个向量或一个点

静态常用基础定义：

Vector3.zero, Vector3.right/.left, Vector3.forward(z轴)/back, Vector3.up(y轴)/down

算距离的静态方法：Vector3.Distance(v1,v2)

this.transfrom.position 绝对位置

this.transform.localPosition 相对位置

\*transform Vector3必须整体改变，不能单独更改某轴值

this.transform.forward/back/…… 获取对象目前第一视角位置

this.transform.Translate(direction \* speed \* deltatime, (Space.world/Self))

通过API位移，默认以自己坐标系

不要direction为自己坐标系参考，而又以Space.Self移动，会变化两次相对坐标，预期不一致。

**缩放**

this.transform.lossyScale 世界（只读不能写）

this.transform.localScale 相对（可以读写）

**入门34，38-40课暂时跳过！**

**鼠标与键盘输入：**

静态类：Input

鼠标位置：Input.mousePosition, 返回值Vector3

原点位于左下角，右是x正方向，上是y正方向，z一直是0

鼠标按下/抬起：

Input.GetMouseButtonDown(<arg>)

Input.GetMouseButtonUp(<arg>)

鼠标按住：

Input.GetMouseButton (<arg>)

<arg>:0左键1右键2中键

中键滚动：Input.mouseScrollDelta返回一个Vector2

Y：-1 下，0没滚，1上滚

键盘输入：

Input.GetKeyDown(KeyCode.<A-Z>)

Input.GetKeyDown(“q”)必须传入小写字符串

Input.GetKeyUp(KeyCode.<A-Z>)

Input.GetKey (KeyCode.<A-Z>)

检测默认轴输入（WSAD）：返回-1~1的值

(字符串映射可以在project setting-Inuput Manager控制)

GetAxis(“Horizontal”) 水平AD

GetAxis(“Vertical”) 前后WS

GetAxis(“Mouse X”)

GetAxis(“Mouse Y”)

GetAxisRaw(“…”)只会返回-1，0，1，不会返回中间值

**入门43-53课暂时跳过！**

**物理系统-碰撞检测：**

**碰撞产生的条件：1两个条件都有碰撞器Collider 2至少一个物体有刚体**

**刚体组件Rigidbody：**让一个物体会受到力的作用

属性：

Mass-质量（单位：千克）

Drag-空气阻力

Angular Drag-扭矩（旋转）对象时收到的空气阻力

Use Gravity-是否受重力

is Kinematic-如果启用，则不会被物理引擎驱动，只能通过Transform对其进行操作。

如果要动画化附加了HingeJoint的刚体，该会非常有用

**Interpolate-插值运算：物理帧过长时平滑动画**

None不运用/Interpolate根据前一帧变换来平滑变换/Extrapolate根据下一帧估计变换来平滑变换

**Collision Detection-碰撞检测模式：用以防止快速移动的对象穿过其它对象而不检测碰撞**

Discrete离散检测，默认，最不耗性能。若太快就会穿过，未被检测。性能消耗低­>高。

连续推测抬升所有至连续推测，优先于离散下降所有至离散，额外连续/连续碰撞为离散，其余连续

Continuous连续检测

Continuous Speculative连续推测检测

Continuous Dynamic连续动态监测

**Constraints约束：**

Freeze Position：停止沿轴位移

Freeze Rotation：停止沿轴旋转

**碰撞器Collider：**用于模拟物体受物理效果的体积（形状）

形状：盒状Cube，球状Ball，胶囊Capsule（性能高）/网格Mesh，轮胎Wheel，地形Terrain（准确度高）

共同参数：

Is Trigger-是否是触发器，如果是，则会用于触发事件并被物理引擎忽略

Material-材料材质，用以确定碰撞体和其它对象碰撞时的交互方式

Center-碰撞体在对象局部空间中的中心点位置

刚体对象的子对象碰撞器信息参与碰撞检测，于是只需要在父对象上加入刚体脚本

网格Mesh碰撞器：

Convex-启用后可以与其他Mesh Collider碰撞，最多255三角形面(如果加入刚体必须勾选)

Cooking Options-管理一些网格烹制选项

Mesh-引用需要用于碰撞的网格

轮胎Wheel碰撞器：用于赛车游戏，不会单独使用，车体必须加刚体

地形Terrain碰撞器：基于地形系统，因为效率不高所以很少用

**入门56-65课暂时跳过！**

根据路径加载Sprite：

Sprite sprite = Resources.Load<Sprite>($"Sprites/{id}");