**Unity编辑器结构**

**大致被分为三大部分**

**菜单栏：File、Edit、GameObject等菜单项**

**工具栏：移动缩放、播放等操作**

**视图部分(5大视图)：**

**层级视图**

**场景视图**

**游戏视图**

**检视视图**

**工程/项目视图**

**建模**

**常用的一些操作:**

**Alt + 鼠标左键 可以移动场景视图中的视角**

**Alt + 鼠标右键，可以拉近和拉远场景视图中的视角**

**鼠标滚轮 可以拉近和拉远场景视图中的视角**

**鼠标右键 可以调整场景视图中的视角**

**鼠标右键 + W(A、S、D)可以调整场景视图中的视角**

**使场景视图视角与游戏视图的视角保持一致的操作**

**先选中相机 然后 同时按住Cmd + Shift + F 就可以（Windows ： Ctrl + Shift + F）**

**Tank 资源—>游戏对象**

**Cube 游戏对象->资源**

**预设体的好处：**

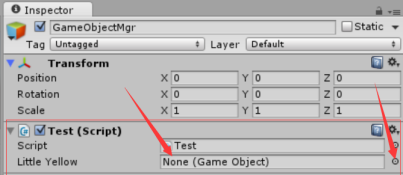
**批量操作游戏对象，创建、修改**

**下午**

**创建脚本(最好要分类管理这些脚本)**

**1 在工程视图下，右键->Create->C# Script**

**2 给脚本中的公共成员变量赋值方式可以在检视视图中操作：**

****

**点了之后**

****

**双击Capsule，就可以给LittleYellow变量赋值**

**脚本知识点**

**周期函数：**

**Awake Start**

**Update FixedUpdate LateUpdate**

**OnEnable**

**OnDisable**

**OnDestroy**

**等周期函数**

**作业：周期函数的调用顺序如何？**

**正确答案**

**刚加载完 Awake->OnEnable->Start->FixedUpdate->Update->LateUpdate**

**销毁 OnDisable-> OnDestroy**

**隐藏 OnDisable**

**FixedUpdate :以固定频率调用的函数**

**Update ：以非固定的频率调用的函数，该频率与设备的性能和程序自身有关**

**LateUpdate：每当Update调用完之后立马调用**

**OnDisable ：游戏对象消失(销毁/隐藏)的那一刻调用**

**OnDestroy ：游戏对象销毁的那一刻调用**

**扩展： Reset 构造函数(在脚本编译完之后，构造函数自动调用)**

**Reset 在脚本第一次挂载到游戏对象上的时候自动调用一次**

**构造函数 : 只要脚本重新编译，就会调用**

**Time类：**

**Update中的Time.deltaTime的值每次都不一定相同**

**LateUpdate中的Time.deltaTime的值与Update的是相同的**

**FixedUpdate中的Time.deltaTime的值每次都是相同的**

**组件：**

**1》Transform this.transform**

**2》普通脚本 (Test脚本)**

**问题：**

1. **不用Rotate/Translate 让物体进行旋转/平移运动**

**每帧改变坐标/角度即可，让坐标或者角度按照某个规律进行递增或递减，然后将角度/坐标值赋值给游戏对象即可**

**2 当多个游戏对象的Tag值一样，那FindWithTag获取到的是哪个游戏对象？**

**根据具有相同Tag值得游戏对象的添加顺序，最后被添加的游戏对象先被获取到**

**类的内部**

**方法体外**

**内存划分：**

**堆 栈 全局区/静态区 代码区 常量区**

**插值**

**Mathf.Lefp**

**Vector3.Lerp**

**2016/9/26**

**如何给一个游戏对象添加组件**

**1> 通过代码的方式添加组件： AddComponent<组件类型>();**

**2> 手动添加：在检视面板中 AddComponent ->选择想要添加的组件;**

**如果是脚本组件，可以直接将脚本拖入到游戏的检视面板中；**

**菜单栏Component->Add（Ctr + Shift + A）->选择想要添加的组件**

**如何获取一个游戏对象的组件**

**1> 组件和某个脚本挂在同一个游戏对象上，可以在该脚本中直接用GetComponent<组件类型>()**

**2>如果脚本与某个组件不在同一个游戏对象上，可以定义一个public的组件类型的变量，通过检视面板拖动赋值**

**3>如果脚本与某个组件不在同一个游戏对象上，先获取游戏对象，再通过游戏对象获取其组件**

**碰撞体**

**游戏对象之间碰撞的条件：。。。。。**

**触发器**

**2016/9/27**

**相机：**

**其功能主要是渲染场景中的游戏对象**

**正交相机**

**一般用于2d的游戏对象，忽略z轴的大小，不管z多大(在视野区域内)，游戏对象的大小始终不变**

**透视相机**

**一般用于3d内的游戏对象，不忽略z轴的大小，离相机越远，游戏对象显示出来的越小**

**参数说明：**

**Culling Mask ：显示过滤，通过调节此参数，可以有选择性的渲染某些层的游戏对象**

**Field Of View(FOV) ：视野区域，用于透视相机**

**Size ：视野，用于正交相机**

**Target Textrue ：可以将相机的渲染画面投射到此图片中**

**NGUI**

**UI Camera:用于处理NGUI中的手势识别（单机、双击、拖拽等手势）**

**Sprite(精灵) 显示图片**

**创建方式：Create->Sprite**

**具体操作：创建完精灵之后，指定Atlas(图集)【点击Atlas，找到需要的图集】,在图集选择之后就可以为精灵控件指定Sprite参数了，Sprite是图集中的一张小图**

**特别提醒：千万不要试图通过缩放控件来调整控件的大小，要尽量保证控件的缩放值为Vector3.one，通过修改控件的Size参数来调整控件显示的大小比例**

**扩展：精灵的帧动画**

**原理：将一系列的具有相同的图片组合在一起，然后用精灵控件显示这些图片，显示图片具有一定的规律，这样的可以达到一个动画的效果(精灵帧动画)**

**操作：**

**添加精灵控件->添加Sprite Animation脚本**

**【**

**参数说明：FrameRate 播放的频率**

**Name Prifix 每帧动画的名称的前缀**

**Lopp 是否循环**

**Pixel Snap：不改变原始图片大小**

**】**

**Label(文字标签，用于显示文字)**

**创建方式：Create->Lable**

**具体操作：需要指定Font(字体)，才能显示文字，指定字体：检视面版中 NGUI/Unity->Font**

**参数说明：**

**Modifier：字母的大小写设置**

**Overflow：溢出处理方式**

**Alignment:字体的对齐方式**

**Spacing ：调整字体的间距**

**MaxLines：最大行数**

**0代表不限制**

**其他大于0的数值n 代表限制在n行内**

**字体控件扩展**

**让一段文字显示不同的颜色 可以使用以下方式**

**[-][颜色的16进制数值]文字内容[-]**

**例如：改变3323的颜色为红色（红色的16进制表示为FF 00 00 FF）**

**[-][FF0000FF]3323[-] 3323就会显示为红色**

**Texture(纹理控件) 显示图片**

**与精灵控件相比，纹理控件只能显示一张图片，而精灵能单独显示一张图片中包含的某张小图**

**创建方式：Create->Texture**

**具体操作：指定好要显示的图片，一般情况下，纹理控件是用来显示一张大的背景图片的**

**参数说明**

**Textrue:要显示的图片纹理目标**

**Material：可以对纹理图片做一些特殊的操作，比如将图片变成圆的等各种形状**

**Widget 一般用于管理控件的，作为控件的容器**

**创建方式：Create->Widget**

**参数说明：**

**Alpha ：控制自身以及其子节点的透明度**

**作用：起到 统一管理其下的子节点**

**Panel 用于管理控件的，作为控件的容器**

**创建方式:Create->Panel**

**作用：起到 统一管理其下的子节点，相对于Widget而言，Panel还能统一的去提升和降低子控件的深度值**

***NGUI第二部分***

**高级控件**

**ScrollView 滚动视图**

**创建方式：Create->ScrollView**

**问题：如何让滚动视图能够滑动？**

**第一种方式：**

**给ScollView控件Attach一个Collider，然后再添加一个脚本组件(Drag Scroll View),这样就可以让滚动视图滚动起[实际添加的Collider包围了内容]，这种方式的弊端是子控件的所有触摸都被父节点ScrollView屏蔽了**

**第二种方式：**

**给SrollView的所有子控件Attach一个Collider，然后再给所有子控件添加一个脚本(Drag Scroll View)，这种方式的弊端是子控件都得重复添加Collider和Drag Scroll View脚本。**

**Grid 网格视图**

**创建方式：Create->Grid**

**作用：可以对网格视图下的子控件进行某些规律的布局**

**参数说明：**

**Arrangement：排列方式**

**Cell Height:单元格高度**

**Cell Width：单元格宽度**

**Column Limit（在有水平排列方式下可以修改） ：限制单元格的列数(0代表不限制)**

**Row Limit（在有垂直排列方式下可以修改）：限制单元格的行数(0代表不限制)**

**Table 表格视图**

**创建方式：Create->Table**

**参数说明：**

**。。。**

**Attach菜单项**

**Collider**

**创建方式：Attach->Collider**

**Button Script 按钮脚本**

**创建方式：创建一个空对象->Attach->Button Script**

**作用：能够让控件在接受到手势的作用下而发生一些插值动画(颜色的变化)**

**参数说明：**

**Target Tween ：指定的控件**

**Colors：**

**不同状态下的颜色**

**Sprites：**

**不同状太下的显示的精灵**

**Pixel Snap：控件的尺寸来适应精灵所用的图片的尺寸**

**Toggle：开关脚本**

**创建方式: 创建一个空对象->Attach->Toggle Script**

**能够实现开关的条件：**

**1>首先需要接收到点击事件，所以应该有Collider**

**参数说明：**

**Group：组号**

**Starting State：开始状态**

**State Transition(状态切换相关设置)**

**Sprite：开/关状态的精灵**

**Invert Sprite：反转**

**规律：**

**如果 Invert Sprite 是 false的情况下**

**Sprite 需要被指定为靠最前的精灵**

**Starting State如果为false 那么一开始 上面的精灵就是隐藏的，**

**如果是true，上面的精灵是显示的**

**Slider : 滑动条**

**创建方式：创建一个空对象->Attach->Slider Script**

**参数说明：**

**Value：滑动条的值**

**Alpha：滑动条的透明度**

**Steps：步长**

**Foreground：前景图，显示滑动条的大小**

**Background：背景图**

**Thumb ：滑块**

**扩展：声音(2d)控制**

**声源：发声的地方 AudioSource**

**监听：监听声源发出的声音 AudioListener**

**ScrollBar：滚动条**

**创建方式：创建一个空对象->Attach->ScrollBar Script**

**参数说明：**

**Size：滚动条前景图的显示的尺寸，该尺寸一般与内容有关，即：如果没有内容是可以自行调节的，如果有内容，那么跟内容相关**

**其他同Slider类似**

**制作技巧：**

**创建一个空对象->Attach->ScrollBar Script**

**该空对象添加两个精灵，作为背景以及前景，背景和前景图的尺寸先保持一致**

**ProgressBar：进度条**

**创建方式：创建一个空对象->Attach->Progress Bar Script**

**参数说明：**

**与Slider参数一样**

**Slider 、Scroll Bar、Progress Bar的对比**

|  |  |  |  |
| --- | --- | --- | --- |
| **控件名称** | **特殊参数(与Slider对比)** | **是否能够接受手势** | **用途** |
| **滑动条(Slider)** | **无** | **能** | **控制游戏种的音量大小等** |
| **滚动条(Scroll Bar)** | **多了一个Size参数，该参数与滚动内容相关** | **能** | **显示滚动区域** |
| **进度条(Progress Bar)** | **无** | **不能** | **显示加载进度，或者血条，蓝条** |

**PopupList：弹出列表视图控件**

**创建方式：创建一个空对象->Attach->PopupList Script**

**参数说明：**

**Options：选项，由字体构成**

**Default：默认选项**

**Position：往下/往上弹出列表**

**Alignment：排序方式**

**OpenOn：选项被选中的手势**

**Atlas**

**Background：选项背景**

**HighLight：选项被选中高亮显示的背景**

**Font**

**字体文件，需要指定字体文件才能显示选项，否则无法显示**

**InputField：字体输入控件**

**创建方式：**

**参数说明：**

**作业：聊天窗口的制作？**

**Play Tween Script：让控件能够控制一组Tween动画的播放**

**创建方式：选中某个控件(按钮、开关等)->Attach->Play Tween Script**

**参数说明：**

**Tween Target：被控制的Tween动画的游戏对象**

**Include Children：是否包含子物体的运动(经测试两者没有太大区别)**

**Tween Group：控制的动画组号**

**Trigger condition:触发Tween动画播放的条件(以何种手势触发)**

**Play Direction：播放方式(开关形式、顺着动画播放、逆运动播放)**

**If target is disabled：如果被控制的游戏对象是disabled应采取的方案**

**On activation:包含有 继续从当前进度播放、重置动画从头开始、播放完毕后回到起始状态**

**When fineshed：动画播放完后。。。**

**On finished：动画播放完后回调**

**Tween 相关脚本（用于给控件添加补间动画）**

**Alpha：可以给具有Alpha参数得控件添加，让其具有在Alpha上的动画**

**参数说明：**

**from：动画的起点数值**

**to：动画的终点数值**

**Tweener**

**Player：动画的循环模式(once:不循环，只有一次 Loop： A-B循环 PingPong ：A-B-A循环)**

**Animation Curve：动画曲线**

**Duration：动画播放时长**

**Start Delay：开始播放动画的延迟**

**Tween Group：动画组号**

**Ignore TimeScale：是否忽视TimeScale**

**On Finished：**

**监听补间动画播放结束**

**NGUI第三部分 工具使用**

**Atlas Maker：图集制作工具**

**做图集的意义何在？**

**1》资源管理:一般情况下，游戏会按照功能对图片进行规划，然后将具有相同功能模块上用到的图片放到一个图集中去，而对于频繁使用的图片，我们会将其放入公共图集中，方便游戏的维护管理。**

**2》能够适当减小游戏的DrawCall(Batches)数值：减小DrawCall数值的规则是，NGUI会对相邻[深度值相差0/1]的控件采取特殊处理，如果图集相同，DrawCall就会只有1次，如果不相邻的控件就无法采取特殊处理，会多出一次DrawCall，如下图所示(A C 同一个图集，B是另外一个图集)**

A、C不相邻 DrawCall + 3

A C相邻 DrawCall + 2

**Font Maker：字体制作**

**静态字体：**

**动态字体：**

**NGUI第四部分 NGUI多屏幕适配**

**适配原则是**

**为了不改变背景图片原始比例，一般宽高的缩放比例要相同，对于背景图片的适配，拿横板游戏来说，一般优先采取的是对宽进行适配，设计图片时，尽量采取宽高比，比较大的方案进行设计(1280\*720 = 1.7多？为何用这个比例呢？)，对于不同的屏幕而言，尽管有很多屏幕，但是相对于宽而言，背景图片永远会填充完，而高要尽量留出一点黑边，黑边也可以使用一张相对比较大的图片进行填充即可(当然有些公司就直接留出这个黑边不管)。**

**对于控件布局，在不同屏幕的设备上，往往控件的布局也会被打乱，甚至有些控件被遮挡、有些控件溢出屏幕外，这样的现象显而易见是不满足用户体验的，NGUI中有一个锚点控件，我们使用锚点对控件进行布局，在设计UI的时候，按照功能，给UI设计不同的挂点(其实就是一个父节点)，用锚点对其挂点进行位置锁定，这样UI的布局才不会乱。**

**扩展：**

**用单例封装窗口工具类**

**单例特点：内存中只有一个单例，不能继续创建或者拷贝单例**

**public class 类名**

**{**

**private static 类名 instance;**

**public static 类名 GetInstance()**

**{**

**if(instance == null)**

**instance = new 类名();**

**return instance;**

**}**

**}**

**外部访问单例方式：**

**类名.GetInstance();**

**监听控件的手势作用：**

**<1> 手动拖：创建的脚本可以挂在任何游戏对象上(最好是在控件父节点上，便于管理)，回调函数的命名没有限制**

**<2> OnClick：创建的脚本必须挂在控件上，点击该控件才会有所反应，而且回调的函数必须是OnClick命名，该函数在控件点击后自动回调**

**<3>UIEventListener：通过委托的方式给控件添加点击回调**

**<4>UIEventTrigger :通过给控件添加UIEventTrigger组件来实现点击回调，具体方式如下**

**控件的游戏对象Btn**

**EventDelegate de = new EventDelegate(脚本实例,”方法名”);**

**de.parameters[0] = Btn;//传给方法的第一个参数，第二个参数是de.parameters[1] 依次类推**

**Btn.AddComponent<UIEventTrigger>().onClick.Add(de);**

**NGUI中的控件要想能对手势做出反应，必要的条件是Collider**

射线

SendMessage

**EasyTouch**

**1 插件导入**

**2 EasyTouch菜单栏**

**Add EasyTouch for C# : 添加EasyTouch游戏对象，该游戏对象挂载有EasyTouch脚本组件(C#)**

**Add EasyTouch for javascript :添加EasyTouch游戏对象，该游戏对象挂载有EasyTouch脚本组件(JS)**

**Extensions**

**Adding a new joystick :**

**Adding a new button :**

**EasyJoystick：**

**NGUI控件永远的显示在用OnGUI写的虚拟摇杆的后面，除非改底层，否则无法扭转这种现象。**

**如何在点击了NGUI的按钮后，不触发虚拟摇杆的触摸响应？**

**XML格式化字符串解析**

**如果XML文件以TextAsset类型存在，那么该XML需要遵循Unity的命名规范，既必须以.txt后缀名结尾的命名。**

**XML：标记性扩展语言**

**特点：以成对的节点方式，且具有根节点，根节点也可以具有子节点，而每个子节点又可以有自己的子节点，每个节点可以有属性，有值，需要注意的是根节点有且仅有一对，如下所示。**

**例：**

**<?xml version="1.0" encoding="utf-8"?> ========>XML 一些信息**

**<LevelConfig> ========>根节点**

**<level>**

**<Number>1</Number>**

**<Lock>0</Lock>**

**<Star>0</Star>**

**<Time>30</Time>**

**<TargetScore>4</TargetScore>**

**</level>**

**<level>**

**<Number>2</Number>**

**<Lock>1</Lock>**

**<Star>0</Star>**

**<Time>25</Time>**

**<TargetScore>7</TargetScore>**

**</level>**

**<level>**

**<Number>3</Number>**

**<Lock>1</Lock>**

**<Star>0</Star>**

**<Time>24</Time>**

**<TargetScore>7</TargetScore>**

**</level>**

**<level>**

**<Number>4</Number>**

**<Lock>1</Lock>**

**<Star>0</Star>**

**<Time>23</Time>**

**<TargetScore>8</TargetScore>**

**</level>**

**<level>**

**<Number>5</Number>**

**<Lock>1</Lock>**

**<Star>0</Star>**

**<Time>22</Time>**

**<TargetScore>8</TargetScore>**

**</level>**

**<level>**

**<Number>6</Number>**

**<Lock>1</Lock>**

**<Star>0</Star>**

**<Time>21</Time>**

**<TargetScore>9</TargetScore>**

**</level>**

**<level>**

**<Number>7</Number>**

**<Lock>1</Lock>**

**<Star>0</Star>**

**<Time>21</Time>**

**<TargetScore>9</TargetScore>**

**</level>**

**<level>**

**<Number>8</Number>**

**<Lock>1</Lock>**

**<Star>0</Star>**

**<Time>18</Time>**

**<TargetScore>10</TargetScore>**

**</level>**

**<level>**

**<Number>9</Number>**

**<Lock>1</Lock>**

**<Star>0</Star>**

**<Time>16</Time>**

**<TargetScore>10</TargetScore>**

**</level>**

**<level>**

**<Number>10</Number>**

**<Lock>1</Lock>**

**<Star>0</Star>**

**<Time>15</Time>**

**<TargetScore>10</TargetScore>**

**</level>**

**<level>**

**<Number>11</Number>**

**<Lock>1</Lock>**

**<Star>0</Star>**

**<Time>14</Time>**

**<TargetScore>10</TargetScore>**

**</level>**

**</LevelConfig> =======>根节点**

**XML解析步骤:**

**<1> 创建XML解析的工厂 XMLDocument**

**<2> 向解析工厂加载XML文件/XML格式化字符串**

**<3> 获取解析的信息**

**获取根节点 =>doc.SelectSingleNode("根节点名称")**

**获取节点的名称=>node.Name**

**获取节点的值=>node.InnerText**

**如果需要对修改的信息进行保存，需要将XML字符串拷贝到可读写(Application.persistentDataPath)的文件中，每次解析都是在解析可读写的文件。**

**File IO(流)操作**

**WriteAllText() =====>将内容写入到文件中**

**GetAllText() =====>获取文件中的内容**

**Exist() =====>判断路径下的文件是否存在**

**Copy() =====>将原文件中内容拷贝到目标文件中**