**第5章 UI界面**

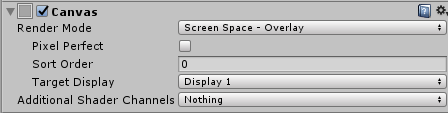
# 5.1 UI组件

UI系统能快速而直观地创建用户界面，该章节内容是对Unity UI系统的主要特性的介绍。

## 5.1.1 渲染组件

#### 1. Canvas（画布）

我们可以将Canvas理解成一个容器，而其他UI元素填充到该容器中，形成了我们看到的一个UI界面。因此可以设定游戏中用到Canvas的地方均可以作为一个独立的游戏界面，当容器的渲染属性发生改变时，所有子物体均会收到影响。



**【特性】**

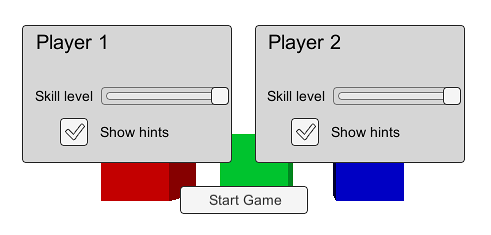
Canvas是UI渲染最主要的组件：

* 只有放在Canvas下的子物体才会参与UI的渲染。
* 形状大小取决于屏幕分辨率，我们可以看到创建出来的Canvas是一个矩形，我们可以修改**Game**窗口的分辨率选项修改矩形大小，窗口分辨率默认设置为FreeAspect，我们可以把切换它成1920X1080这样具体数值的分辨率，这样我们的界面就不容易出现变形。
* 子物体的渲染层级取决于UI元素在层次结构中出现的顺序，两个UI元素位置上重叠，层级结构下方的UI元素会遮挡上方的UI元素。

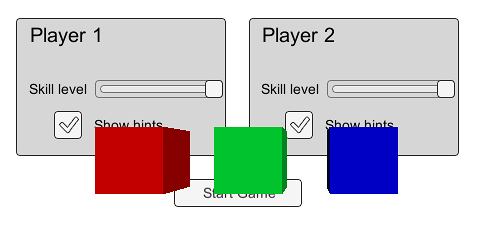
**【参数】**

如下表所示。

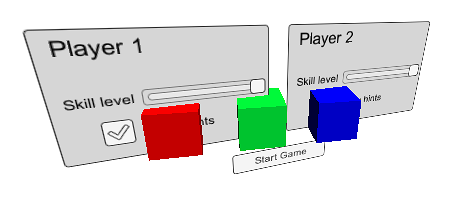
|  |  |
| --- | --- |
| **参数** | **功能** |
| Render Modes（渲染模式） | UI界面通过Canvas的渲染模式来设置自己渲染在屏幕空间还是世界空间 |
| Overlay（覆盖渲染模式） | 设置界面为屏幕空间，UI界面处于屏幕空间时将会被渲染在游戏窗口的最顶层 |
| Camera（相机渲染模式） | 和Overlay设置相似，但它的渲染取决于摄像机。当摄像机层级较低时，高层级的摄像机所见物体将会遮挡UI，这种特性经常用做UI界面上显示3D模型。同时摄像机若设置为Perspective透视选项，UI界面将会可以设置更加立体的效果，而不是一张只能铺在平面上的纸 |
| World Space（世界空间渲染） | 设置界面在世界空间中渲染，可以单独设置UI的位置、旋转、缩放的变换。可以理解为UI组件成为游戏世界中的一个物体，比如游戏中经常用到的血条、伤害扣血字，对话文本框 |
| Pixel Perfect (优化像素渲染) | 只能屏幕空间渲染中开启，打开后UI会进行抗锯齿渲染 |
| Sort Order（渲染排序） | 不同的Canvas根据渲染排序的层级设置，显示成不同的遮挡关系，详细可以查看‘UI进阶-渲染顺序’章节 |
| TargetDisplay（显示目标） | 在Overlay模式下将会出现，和多屏显示相关 |
| Additional Shader Channels（附加着色器通道） | 设置在创建画布网格时使用的附加着色器通道 |



Overlay（覆盖渲染模式）：



Camera（相机渲染模式）：

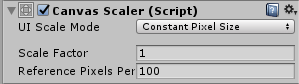


World Space（世界空间渲染）：

#### 2. Canvas Scaler（画布缩放组件）

画布缩放组件是用于控制整体界面的缩放和在画布上的UI元素的像素密度。这种缩放影响画布下的所有内容，包括字体大小和图像边框。

由于游戏会存在不同的分辨率模式，UI需要能在不同的分辨率下保持一个恰当的显示效果。因此我们需要让UI有一个可以适应的区域范围，比如由长方形的屏幕变成了正方形时，UI界面不会发生太严重的形变。这个时候就需要设置画布缩放组件。让调节我们的UI界面，始终保持屏幕对齐。



**【特性】**

* 控制整体界面UI元素的大小
* 能让UI界面维持适应屏幕的比例

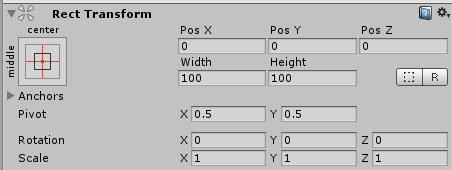
**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| UI Scale Mode(UI渲染模式) | 设置当前Canvas的渲染模式 |
| Constant Pixel Size（固定像素尺寸） | 使UI元素保持相同大小的像素，而不管屏幕大小。 |
| **Scale With Screen Size**（随屏幕缩放尺寸） | 使UI随着屏幕变大而变大 |
| **Constant Physical Size（固定物理空间尺寸）** | 使UI保持物理空间大小，不管分辨率和屏幕大小。 |
| **Scale Factor（缩放系数）** | 按照该比例缩放UI中的元素 |
| **Reference Pixels Per Unit（每单元参考像素大小）** | **如果Sprite勾选了**‘Pixels Per Unit’设置， 那么该设置将会影响一个UI单元占用多少图片像素。 |
| **Reference Resolution（参考分辨率）** | 屏幕大小与该分辨率大小比对，如果屏幕大，UI将会放大，反之缩小 |
| **Screen Match Mode（屏幕匹配模式）** | **屏幕匹配模式** |
| **Match Width or Height（匹配宽度或高度）** | 根据设置的高度宽度的权重匹配，若偏向Width，那么UI将会参考宽度变化的大小匹配界面。如果制作横屏手机游戏，一般以偏重Height大小 |
| **Expand（扩展）** | 扩展画布的水平或竖直区域。保持画布的大小永远不会小于参考分辨率 |
| **Shrink（收缩）** | 缩小画布的水平或竖直区域。保持画布大小永远不会大于参考分辨率 |

## 5.1.2布局组件

#### 1. Rect Transform（矩形变换）

因为UI物体的资源为Sprite像素图片，因此通常需要对Width和Height来对图片的进行形变（UI物体的形变通常不通过修改缩放值，因为UI修改缩放值拉伸后容易出现模糊精度丢失的问题），UI的经常会遇到排版和屏幕适应问题，因此需要锚点功能来做定位对齐。因此Unity对UI物体单独制作了RectTransform变换组件。



**【特性】**

RectTransform是记录UI空间属性的组件:

* 能修改UI元素的长宽，配合Sprite图片的九宫格设置，能以很小的像素图片创建UI背景板。
* 仅改变UI元素自身的长宽，并不会影响其他UI元素（如字体大小，其他图片的长宽等）
* 对齐功能，能够通过左上角十字框实现九个位置上的对齐和吸附功能（通过按Alt按键切换对齐吸附效果）

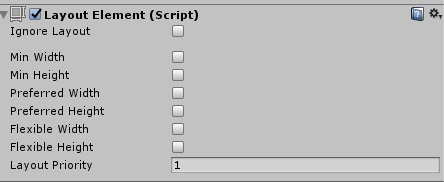


**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| Pivot（中心点） | 中心点影响UI元素的旋转、改变长宽及缩放的显示效果。当工具栏选中为Pivot Mode时，可以在场景中移动中心点的位置；在Rect Tranform组件中也可以修改Pivot的位置。 |
| Anchors（锚点） | 锚点影响父子节点的相对变化。父节点锚点影响子节点相对于该锚点的相对位置和缩放。调整锚点有三种方式：   * 直接在场景中拖拽锚点 * 在Rect Transform组件中使用预置的锚点位置 * 在Rect Transform组件中改变Anchors的值 |

### 2. Layout Element（布局元素）

如果要重定义布局元素的最小、最合适或自适应的大小，可以通过向游戏对象添加Layout Element来实现这一点。



**【特性】**

* 允许重写一个或多个布局属性的值
* 自动与其他布局元素计算大小，当首选大小不适配后，会选用额外适应的大小量自动调节UI元素大小

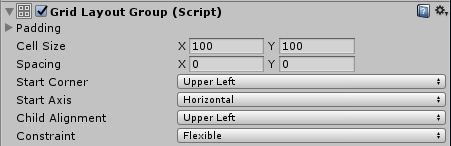
**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| Min Width（最小宽度） | 该布局下的最小宽度 |
| Min Height（最小高度） | 该布局下的最小高度 |
| Preferred Width（适宜宽度） | 首选最适宜宽度（有效宽度有空余情况下） |
| Preferred Height（适宜高度） | 首选最适宜高度（有效高度有空余情况下） |
| Flexible Width（灵活宽度） | 灵活宽度（有效宽度附加量） |
| Flexible Height（灵活高度） | 灵活长度（有效高度附加量） |

### 3. Layout Controllers（布局控制器）

Layout Controllers既可用于控制自身（即父物体）也可用于控制子物体的元素，包含 Horizontal Layout Group（横向），Vertical Layout Group（纵向）和 Grid Layout Group（网格状）组件。

这里我们主要介绍Grid Layout Group（网格状）组件，因为横向和纵向的参数功能为网格状组件的子集，了解了Grid Layout Group就了解了布局控制的核心。



**【特性】**

* 作为布局系统，具有全面的布局约束属性
* 计算设置自身的大小时候，参考自身包括自身子物体所有UI渲染的大小

**【参数】**

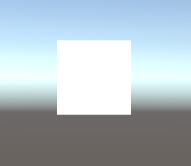
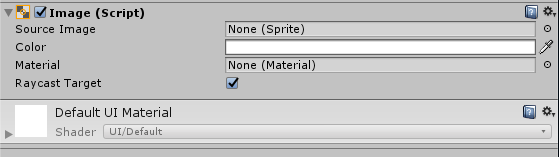
|  |  |
| --- | --- |
| **参数** | **功能** |
| Padding（边缘填充距离） | 四周边缘位置的填充距离 |
| Cell Size（组元素范围） | 组中（自身下一层子物体）元素排序约束的尺寸 |
| Spacing（间距） | 组中元素排序约束的间隔 |
| Start Corner（开始转角方向） | 决定新的元素放置于哪个元素行列上，有左上、左下、右上、右下四个轴向 |
| Start Axis（开始轴向） | 开始排布的轴向，决定竖直排布还是水平排布 |
| Child Alignment（组元素对齐） | 设置组元素排布对齐位置,相当于文本框的左对齐、右对齐、居中等九个排布位置。 |
| Constraint(约束) | 约束固定的行列数量，完成自动布局 |

## 5.1.3显示组件（Visual Components）

显示组件是组成UI界面的基本元素，主要用于在Canvas中显示文字和图片等信息。

### 1. Image（图片组件）

UI界面最基础的组成元素，一切需要渲染出图片信息的功能都会使用到Image组件。而其他的组件实质上是对Image组件的组合，例如一个滑动条，抛开功能逻辑，实质上是一个用做拖拽按钮的Image，一个显示进度的Image，一个底层背景Image的组合。

**【特性】**

作为显示组件，渲染依托于渲染组件，因此只要添加有显示组件同时Unity会自动添加上CanvasRenderer组件。显示的层级依赖与Canvas的设置。

* 用于图片信息的显示
* 不可交互，但可屏蔽遮挡点击事件。

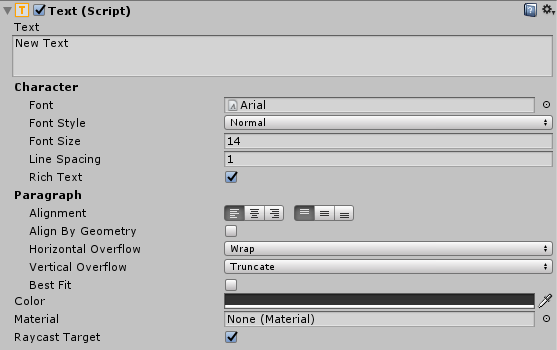
**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| SourceImage（源图片） | 只能添加Sprite格式的图片资源，Sprite为Unity为2D或UI类型设置的特殊格式。导入一张图片时，选择TextureType的类型为Sprite（2D and UI）后，点击Apply。这时Unity会修改图片为Sprite类型的图片。 |
| SetNativeSize（设置原始大小） | 将Sprite拖入SourceImage参数栏后会出现该选项，点击后会修改UI的RectTransform组件的长宽为Sprite的长宽。 |
| Color（颜色） | 设置Image组件的颜色，当不存在SourceImage资源时，设置的颜色就是显示的颜色，当存在SourceImage资源，颜色设置会与Sprite的像素颜色相乘。可以使用动画去动态Color的Alpha通道实现淡入淡出效果。 |
| Material（材质） | UI的渲染Unity是自己有一个Shader来计算渲染的，游戏开发中有时会想要UI制作更加高级的显示功能，就需要替换渲染方式。为了使用我们能自定义的渲染方式，UGUI提供了Material参数接口。 |
| RaycastTarget（射线目标） | 布尔值一般作为开关功能，因此该参数打开时，这个UI会接受射线。当我们点击到UI元素上时，EventSystem都会自动发射UI射线，射线接触到第一个打开“接受射线”的显示组件将会消失。所以界面响应不到点击事件，很有可能是上层出现了一个UI遮挡住了事件。 |
| ImageType（图片类型） | * Simple：等比缩放整张图片 * Sliced ：利用3×3的图片分割区域，当拉伸图片时，只有九宫格的中间区域会缩放，其余区域不会发生变化 * Tiled：与Sliced类似，但中间区域是平铺而非缩放 * Filled：与Simple类似，但在该模式下，可以按照特定的方向、方法和数量填充图片 |

### 2. Text（文本组件）

文本组件用于文字的显示，在该组件中可以很方便的改变文字的字体、样式、大小、对齐方式、颜色等属性。





**【特性】**

* 用于图片信息的显示
* 不可交互，但可屏蔽遮挡点击事件。

**【参数】**

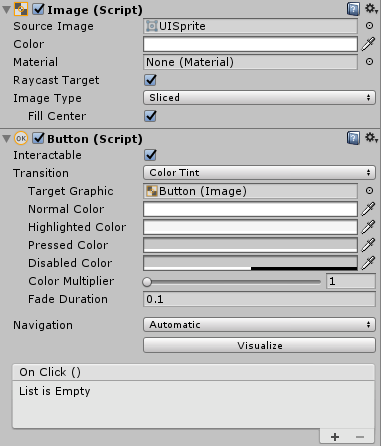
|  |  |
| --- | --- |
| **参数** | **功能** |
| Font（字体） | 字体设置，unity默认字体是Arial。可以从我的电脑中C：>Windows>Fonts中选取其他字体替换，也可以网上下载字体。甚至可以自己制作静态字体，利用BitMap工具制作字体的图集，这个方法广泛用于制作伤害扣血的UI显示效果中。 |
| FontStyle（字体风格） | 字体的加粗，倾斜等设置。 |
| FontSize（字体大小） | 字体大小设置，这里注意字体设置如果过大，超过了RectTransform设置的宽度或高度将不会显示字体(很多时候美术PS中的字体大小和unity的字体大小有区别的，应该统一用像素大小来统一)。 |
| LineSpacing（行间距） | 行间距，间隔的是你当前字体大小的倍数。 |
| RichText（富文本） | 富文本选项，该选项如果勾选，可以通过加入颜色命令字符来修改字体颜色（例：<color=#525252>变色的内容</color>）。游戏的公告的编辑就需要该功能。 |
| Paragraph（段落设置） | Alignment为设置文件上下左右居中等对齐效果  AlignByGeometry为几何对齐，图文混排的时候需要该功能配合  HorizontalOverflow和VerticalOverflow分别为水平和竖直换行，如果Wrap和Truncate选项，内容将会束缚在设置定宽度高度之内，如果选项为Overflow内容将会超出设定的边界。 |
| BestFit（完美适配） | 勾选这个选项，字体将会以RectTransform的宽度高度边界，动态修改字体大小让所有内容刚好填充满这个框。 |
| Color（颜色） | 字体颜色，若用了富文本修改颜色，则不会改变用到了富文本的字体颜色。 |

## 5.2.3交互组件

### 1. Button（按钮）

按钮组件用于相应玩家的点击输入，是游戏中玩家交互较多的组件之一。需要使用按钮的例子有：确认某一选项（如开始游戏、保存进度），开启其他界面，取消某一进程（取消下载更新）等。





**【特性】**

新创建Button物体上只有两个组件，一个组件是之前介绍过的Image组件，一个是Button组件。因为交互组件自身是没有实体的，它是依托于显示组件存在的功能，即使是一个隐形的按钮，也需要依托一个Alpha值为0的Image组件。

* 相应点击输入，从按下到释放按钮是一次完整的点击
* 一次完整的点击才会调用OnClick事件
* 动画集成（Animation Integration），可以利用其自带的动画系统表现各种UI组件的状态切换。若要使用动画切换，需要在对应的UI组件上添加Animator组件，当然也可以点击UI组件上的Auto Generate Animation按钮来实现。Unity已经提供了一些默认的切换动画，当然也可以根据需要自定义动画。

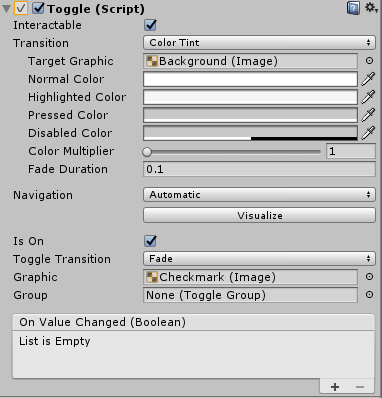
**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| TargetGraphic(目标图像) | Button组件绑定的Image组件，注意如果该项为空按钮点击事件将会失效，同时是有按钮绑定的Image组件勾选了RaycastTarget参数才能有点击效果。 |
| Interactable(可交互) | 是否开启按钮交互，若取消则按钮会变成DisabledColor选择的颜色，此时按钮不会响应点击操作。 |
| Transition(切换效果) | 按钮的点击效果类型，unity自带了3种类型，分别为Color Tint(颜色变化），Sprite Swap(图片切换),Animation (动画变化)。不同类型对应的Normal XXX，Highlighted XXX，Pressed XXX，Disabled XXX，分别为按钮不点击时效果、鼠标移动到按钮时效果、点击时效果和未激活时效果。 |
| OnClick(点击事件) | 点击事件，可以关联点击按钮后的行为至我们自己写的代码中。 |

### 2. Toggle（单选框组件）

单选框是控制某一选项开关切换的组件，可用于切换选项选中状态（例如游戏中背景音乐的播放/关闭）、使玩家确认相应的游戏条款等情况。





**【特性】**

* 用于切换某一选项的选中状态。
* 每当选中状态发生变化时，会调用OnValueChanged事件。

**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| **Interactable**(可交互) | 是否开启交互 |
| **Transition**(切换效果) | 点击效果类型，unity自带了4种类型，分别为None(无变化)，Color Tint(颜色变化），Sprite Swap(图片切换),Animation (动画变化)。 |
| **Navigation(受控方向)** | 决定组件可受鼠标键盘等输入设备控制的方向。分为None(不受任何方向上的输入影响)，Horizontal(受水平方向影响)，Vertical(受垂直方向影响)，Automatic(自动控制)，Explicit(自定义) |
| **Is On(选中)** | 设定单选框的选中状态 |
| **Toggle Transition(选项切换效果)** | 选项变化时的切换效果。分为None(选中效果直接显示/消失)和Fade(选中效果渐进/渐出) 。 |
| **Graphic(源图片)** | 选中效果的源图片 |
| **Group(所属组)** | 该组件所属的单选框组（如果存在的话） |
| On Value Changed(值变化) | 点击事件，会将现有状态的bool值传出 |

### 3. Toggle Group（单选框组）

单选框组是由多个单选框组件组成的，通常用于在一系列选项中挑选其中一个，例如剧情对话中的多个选项，性别选择，任务报酬可选项等





**【特性】**

* 非可视化组件，仅用来管理相关的单选框组件。
* 由多个单选框组成。
* 有且仅有一个单选框可以被选中。

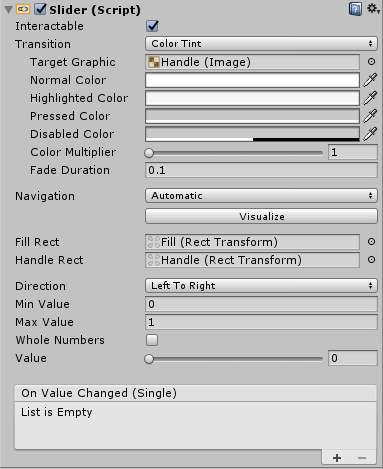
**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| Allow Switch Off(是否允许非选中) | 是否允许选项非选中。若允许，当点击某一已经被选中的选项时，会将其切换为非选中状态；若不允许，则点击已被选中的选项时不会发生任何变化。 |

### 4. Slider（滑动条组件）

滑动条组件是玩家在游戏中接触较多的另一种UI组件，常见于游戏设置（如改变画面亮度，鼠标灵敏度等设置），许多游戏带有的捏人功能中也会用到滑动条组件（如调整身高，胸围，腰围等身体数据）。





**【特性】**

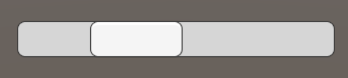
* 根据滑动块的拖拽距离改变值的大小。
* 每当值产生变化时会调用OnValueChanged事件。

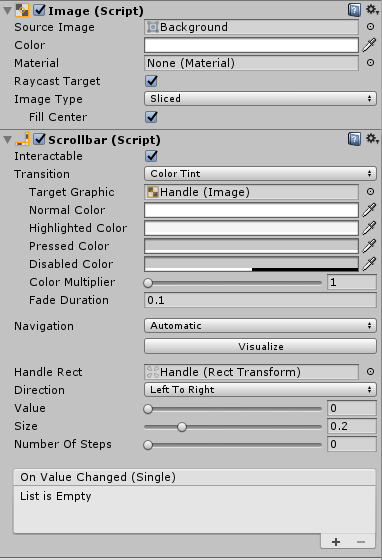
**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| **Interactable**(可交互) | 是否开启交互 |
| **Transition**(切换效果) | 点击效果类型，unity自带了4种类型，分别为None(无变化)，Color Tint(颜色变化），Sprite Swap(图片切换),Animation (动画变化)。 |
| **Navigation(受控方向)** | 决定组件可受鼠标键盘等输入设备控制的方向。分为None(不受任何方向上的输入影响)，Horizontal(受水平方向影响)，Vertical(受垂直方向影响)，Automatic(自动控制)，Explicit(自定义) |
| **Fill Rect(填充区)** | 滑动条的填充区域 |
| **Handle Rect(滑动块区)** | 滑动块信息 |
| **Direction(填充方向)** | 拖动滑动块时，滑动条的填充方向。分为Left To Right(从左向右填充)*,*Right To Left(从右向左填充)*,*Bottom To Top(从底到顶填充)和Top To Bottom(从顶到底填充)。 |
| **Min Value(最小值)** | 滑动块所能滑动到的最小值 |
| **Max Value(最大值)** | 滑动块所能滑动到的最大值 |
| **Whole Numbers(整数化)** | 是否将变化值限制为整数 |
| Value(滑动值) | 现阶段所滑动的值。若在Inspector中修改，会将其设为初始值，但在运行过程中会动态改变。 |
| On Value Changed(值变化) | 当滑动条的值发生变化时会被调用的事件。无论Whole Numbers是否被选中都只会传出float类型的值。 |

### 5. Scrollbar（滚动条组件）

滚动条组件通常用于查看超出可视范围的图片或界面，在游戏中常用于查看背包物品，查看书信等较长的文字信息，查看超出画面范围的菜单选项。





**【特性】**

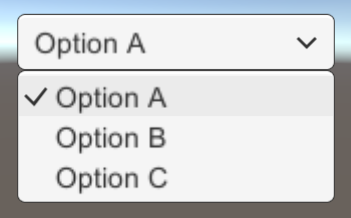
* 滚动块的大小会随着填充区域的大小变化。
* 滚动条值的大小与滚动块的长度，位置和滚动区域的大小有关，是一个百分比数值。
* 滚动块位置变化时会调用OnValueChanged事件。

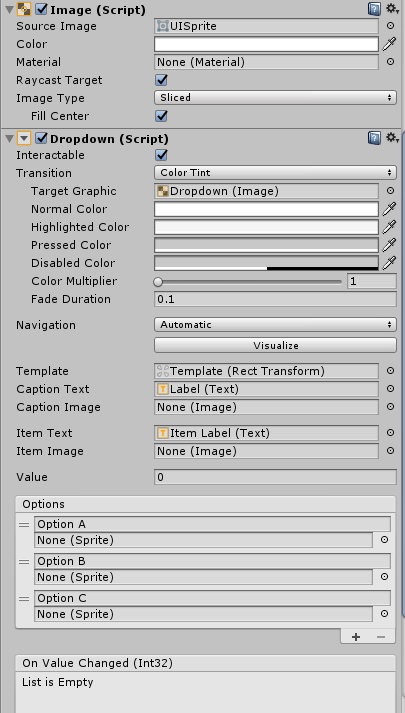
**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| **Interactable**(可交互) | 是否开启交互 |
| **Transition**(切换效果) | 点击效果类型，unity自带了4种类型，分别为None(无变化)，Color Tint(颜色变化），Sprite Swap(图片切换),Animation (动画变化)。 |
| **Navigation(受控方向)** | 决定组件可受鼠标键盘等输入设备控制的方向。分为None(不受任何方向上的输入影响)，Horizontal(受水平方向影响)，Vertical(受垂直方向影响)，Automatic(自动控制)，Explicit(自定义) |
| **Fill Rect(填充区)** | 滚动填充区域背景图像信息 |
| **Handle Rect(滚动块区)** | 滚动块图像信息 |
| **Direction(滚动方向)** | 滚动条的值变大时，滚动块的移动方向。分为Left To Right(从左向右移动)*,*Right To Left(从右向左移动)*,*Bottom To Top(从底到顶移动)*and*Top To Bottom(从顶到底移动)。 |
| Value(滚动块位置) | 滚动条的值，是一个相对的百分比，范围从0.0到1.0 |
| **Size(滚动块尺寸)** | 滚动块的填充范围，限制为0.0到1.0 |
| **Number Of Steps(步长)** | 滚动块可滚动的位置数量 |
| On Value Changed(值变化) | 当滚动条的值发生变化时会被调用的事件。会将float类型的值传出。 |

### 6. Dropdown（下拉菜单组件）

下拉菜单和单选框组作用类似，更节省视觉空间。游戏中的浏览任务/角色信息，游戏设置（如选择分辨率和画质）等经常会使用该组件。





**【特性】**

* 仅可选择提供的选项中的一项，当选中其中一项时会调用OnValueChanged事件。
* 下拉选项既可以是文字也可以是图片。
* 下拉菜单的位置取决于Template的锚点和中心点的位置，下拉菜单的弹出方向会根据组件位置改变，以防出现下拉菜单超出Canvas范围无法显示的情况。

**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| **Interactable**(可交互) | 是否开启交互 |
| **Transition**(切换效果) | 点击效果类型，unity自带了4种类型，分别为None(无变化)，Color Tint(颜色变化），Sprite Swap(图片切换),Animation (动画变化)。 |
| **Navigation(受控方向)** | 决定组件可受鼠标键盘等输入设备控制的方向。分为None(不受任何方向上的输入影响)，Horizontal(受水平方向影响)，Vertical(受垂直方向影响)，Automatic(自动控制)，Explicit(自定义) |
| **Template(选项存储位置)** | 下拉菜单内容的父节点 |
| **Caption Text(已选选项文字信息)** | 现有已选选项的文字信息(可选) |
| **Caption Image(已选选项图片信息)** | 现有已选选项的图片信息(可选) |
| **Item Text(下拉菜单选项文字信息)** | 下拉菜单选项的文字信息(可选) |
| **Item Image(下拉菜单选项图片信息)** | 下拉菜单选项的图片信息(可选) |
| **Value(选择位置)** | 现有已选选项的位置。0代表第一项，1代表第二项，以此类推 |
| **Options(选项列表)** | 选项列表。既可编辑文字信息也可编辑图片信息 |
| On Value Changed(值变化) | 点击下拉菜单选项时调用的事件 |

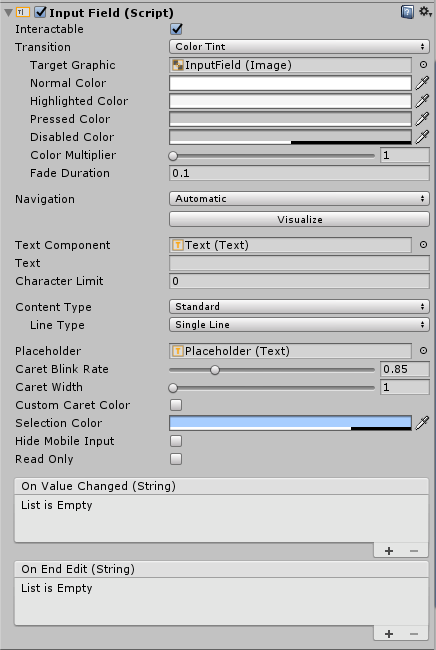
### 7. Input Field（输入框组件）

游戏中经常会有输入行为，比如登录输入账号密码，比如聊天栏输入信息。

这时候需要输入框组件来得到用户的输入数据，如果在手机上点击输入框，Unity会自动打开手机的数字键盘。

输入框可以对输入的数据做约束，Unity自带了各种设置格式类型，只能检测输入的数据是否满足设置的约束。





**【特性】**

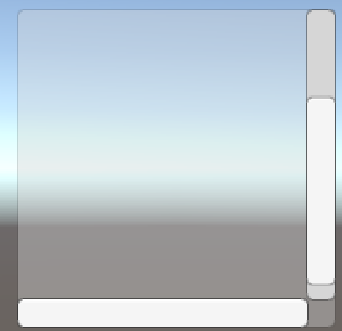
* 唯一的文本输入组件
* 自带约束类型，实现输入字符的屏蔽
* 可以定义显示效果，如颜色变换、图片变换、动画变换
* 字体显示效果和大小取决于依赖的文本组件

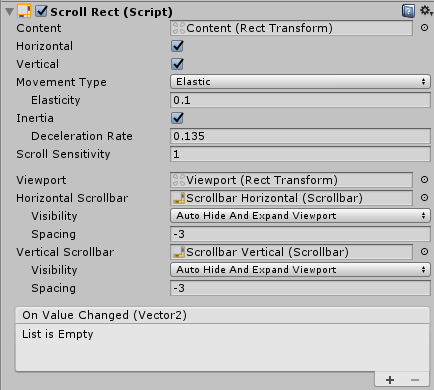
**【参数】**

|  |  |  |
| --- | --- | --- |
| **参数** | | **功能** |
| **Interactable（可交互的）** | | 决定该组件是否可以接受用户交互 |
| **Transition（过渡效果）** | | 决定组件触发Normal, Highlighted, Pressed、Disabled时的过渡效果。 |
| **Navigation（导航排布）** | | 决定空间的排布顺序 |
| **TextComponent（保存输入信息的文本组件）** | | 输入的文本将会显示在该参数的文本组件上 |
| **Text（起始文本）** | | 起始文本，编辑开始时显示的初始文本 |
| **Character Limit（字符限制）** | | 限制写入的最大字符字数 |
| **Content Type（内容约束类型）** | | 决定字段输入字符的类型 |
|  | **Standard（标准）** | 任何字符都可以输入 |
|  | **Autocorrected（自动校正）** | 输入未知单词时，建议用户选取更合适的候选替代。如果用户不重写字符，将会自动替换文本。 |
|  | **Integer Number（整形数字）** | 只允许输入整形的数字字符 |
|  | **Decimal Number（小数）** | 允许输入数字和小数点后的数字 |
|  | **Alphanumeric（字母数字）** | 允许数字和字母，无法输入符号 |
|  | **Name（名字）** | 自动让首字母大写 |
|  | **Email Address（邮箱地址）** | 允许输入最多包含一个@符号的字母数字字符串 |
|  | **Password\*（符号密码）** | 允许符号输入，自动将文字隐藏成输入的星号\* |
|  | **Pin（数字密码）** | 只允许数字输入，自动将文字隐藏成输入的星号\* |
|  | **Custom（自定义）** | 自定义约束 |
| **Line Type** | | 定义字符在文本中的显示格式 |
|  | **Single Line（单行）** | 只允许输入的文本显示在一行内 |
|  | **Multi Line Submit（多行提交）** | 允许使用多行显示，当有需要时显示新的一行 |
|  | **Multi Line Newline（多行换行）** | 允许文本使用多行。用户可以通过按回车键使用换行符 |
| **Placeholder（占位符）** | | 比如当无任何输入字符时，输入框组件会提示‘请输入’文本 |
| **Caret Blink Rate（**光标闪烁速率**）** | | 定义放置在行上的标记闪烁速率 |
| **Selection Color（选定部分颜色）** | | 选定文本部分的背景颜色 |
| **Hide Mobile Input** **(隐藏移动设备)** | | 在移动设备的屏幕键盘上隐藏已经输入的文本。 |

### 8. Scroll Rect / Scroll View（滚动区域组件）

进场用来做滑动界面，比如很多个道具的背包，需要滑动让界面显示更多的元素。Scroll View通常与Mask、和Scrollbars组合完成效果。





**【特性】**

适用性非常广泛，比如游戏中的公告长文本，比如背包系统元素，比如邮箱邮件显示。

**【参数】**

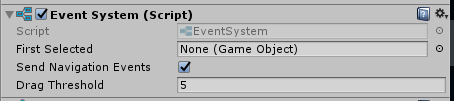
|  |  |
| --- | --- |
| **参数** | **功能** |
| Content（滚动内容物体） | 设定滚动内容区域，通常为一个空的UI物体，我们的UI元素将会放在Content物体层级下。 |
| Horizontal（水平滑动） | 是否开启水平滑动 |
| Vertical（竖直滑动） | 是否开启竖直滑动 |
| Movement Type（运动类型） | 滑动界面的效果：  设置为Elastic（弹性的）将会滑动界面到边缘时会反弹一定距离  设置为Unrestricted（无限制的）将会可以无限的把界面往滑动方向上移动  设置为Clamped（限制区域的）滑动区域将会一直保持在Content界面大小中 |
| Inertia（惯性的） | 当滑动松开时，界面将会根据惯性继续向滑动方向移动 |
| Scroll Sensitivity（滑动灵敏度） | 滑动滚轮和滑动条滑动事件的敏感性 |
| Viewport（视口） | 界面显示的物体，实现滑动区域遮罩屏蔽功能。剔除Scroll View区域外的UI显示 |
| Horizontal Scrollbar/ Vertical Scrollbar（竖直/水平滑动条） | 水平/竖直滑动轴 |
| Visibility（可见属性） | 可以调整滑动条是否自动隐藏、是否自动改变大小适应元素数量 |
| Spacing（间距） | 滑动条和滑动区域窗口的区域间隔 |

## 5.2.4事件功能

### 1. EventSystemManager（事件系统组件）

该组件只会在整个UI系统中存在一个，当创建第一个Canvas时会自动创建到层级窗口中。可以点击EventSystem名字的游戏对象查看组件。

该组件负责控制所有使用到UI事件的元素。EventSystem将会每一帧都会运行，当输入点击到UI元素时他会记录当前点击碰撞到的UI信息和位置信息。如果UI元素绑定了事件，事件系统将会处理该委托调用绑定的函数执行。



**【特性】**

* UI功能的核心组件，所有交互组件的响应事件都由该组件处理
* 该组件是Update中实时运行，因此会受到机器性能的影响

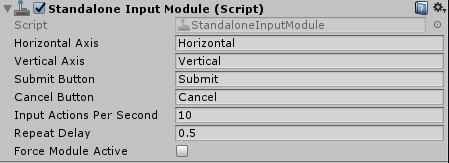
**【参数】**

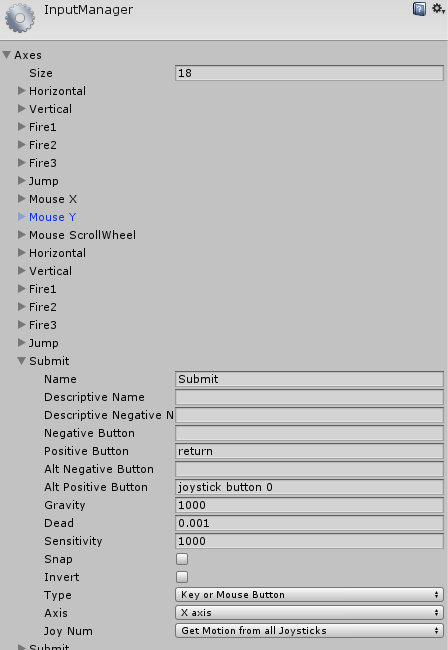
|  |  |
| --- | --- |
| **参数** | **功能** |
| First Selected（第一个选择对象） | 被选择的第一个UI对象（当刚进入游戏无任何输入的时候，默认选择元素） |
| Send Navigation Events（发送导航事件） | 是否允许事件系统可以发送 (move / submit / cancel)等导航席间 |
| Drag Threshold（推拽门槛） | 响应拖拽事件的最小像素区域 |

### 2. Standalone Input Module（独立输入模块组件）

该模块的功能在于，如果你想你的输入控制器，如手柄、鼠标，在UI交互组件上产生了滑轮、确定、取消等输入时，UI系统能够得到相应的事件处理。

它与Editor>ProjectSettings>Input中配置的字符变量对应，通关字符串变量匹配相应外置设备输入。





**【特性】**

使UI界面能够获得多种外设的输入信息，相当于输入信号的转换器

**【参数】**

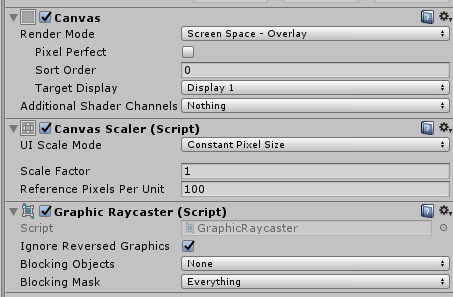
|  |  |
| --- | --- |
| **参数** | **功能** |
| **Horizontal Axis（水平轴）** | 指定水平移动输入绑定的InputManager中字符串变量名称 |
| **Vertical Axis(竖直轴)** | 指定竖直移动输入绑定的InputManager中字符串变量名称 |
| **Submit Button（确定按钮）** | 指定确定输入绑定的InputManager中字符串变量名称 |
| **Cancel Button（取消按钮）** | 指定取消输入绑定的InputManager中字符串变量名称 |
| **Input Actions Per Second（每秒指令数）** | 处理外设每秒输入的指令数量 |
| **Repeat Delay（重复响应延迟）** | 单位为秒，重复行为动作响应的延时 |
| **Force Module Active（强制该模块激活）** | 勾选此复选框以强制该独立输入模块处于实时活动状态 |

### 3. Graphic Raycaster（UI射线发射组件）

当创建Canvas时,自动添加到Canvas物体上。

单独作用射线发射到画布上（该射线不会影响其他场景物体），射线将会击中第一个阻塞它的UI元素时停止。

每一个画布如果有交互组件都应当有Graphic Raycaster，否则将无法得到输入响应。



**【特性】**

与canvas关联绑定，如果父Canvas拥有Graphic Raycaster，而子Canvas没有自己的Graphic Raycaster，那么子Canvas的UI响应将不会被触发。

**【参数】**

|  |  |
| --- | --- |
| **参数名** | **功能** |
| Ignore Reversed Graphics（反向图形忽略） | 射线是否忽略非面向屏幕的UI元素（UI会旋转） |
| Blocked Objects（阻挡对象组） | 射线将会响应的对象类型组（2D、3D或所有对象，需要有collider组件） |
| Blocking Mask（阻挡遮罩） | 会阻挡图形射线的Layer |

### 4. EventTrigger（事件触发器）

EventTrigger相当于Unity交互会用到的所有触发功能都集合在了这个组件之中。可以说Button组件本质一个集合体，集合处理‘从点击按下到抬起’的操作，和对显示组件二次处理（如动画）。因此利用EventTrigger后，我们可以单独将Image赋予可“点击”的响应的能力，而不用Button组件，因为EventTrigger中涵盖了“从点击按下到抬起”的事件触发。

**【特性】**

* 通过编辑界面，自定义交互功能
* 涵盖UI交互相关操作的触发功能
* 内容驱动的回调，无操作时不占用计算资源
* 可以为单个事件分配多个函数

**【参数】**

|  |  |
| --- | --- |
| **参数** | **功能** |
| OnPointerEnter | 当鼠标/触摸进入对象时调用 |
| OnPointerExit | 当鼠标/触摸退出对象时调用 |
| OnPointerDown | 当鼠标/触摸按下对象时调用 |
| OnPointerUp | 当鼠标/触摸抬起对象时调用 |
| OnPointerClick | 当鼠标/触摸按下释放对象时调用 |
| OnBeginDrag | 当开始推拽对象时调用 |
| OnDrag | 当拖拽正在发生时调用 |
| OnEndDrag | 当拖拽结束时调用 |
| OnDrop | 当对象成为拖动物体时被调用，表示推拽正在发生 |
| OnScroll | 鼠标滑轮滑动时调用 |
| OnUpdateSelected | 选中对象时，每一帧都会调用 |
| OnSelect | 当一个对象被选中时调用 |
| OnDeselect  OnMove | 当一个对象被取消选中时调用  当对象发生位移时被调用 |
| OnSubmit | 提交按钮被按下时被调用 |
| OnCancel | 当取消按钮被按下时调用 |

### 5. IEventSystemHandler（事件处理程序接口，非组件）

事件触发器能识别用户的输入，追其根本是Unity的事件处理程序的功能。因此如果我们需要代码中动态修改、添加、删除按钮的事件，就需要寻求代码上的接口来完成。因此对于UI来说，我们需要了解如何代码中去得到用户的操作。

**【特性】**

* 通过Interface接口完成事件功能的添加
* 需要有显示组件的支持，继承IEventSystemHandler类型的脚本类必须挂载在显示组件上才得以运行
* 作为代码类，更加灵活易用，支持自定义自己的事件监听器。

**【参数】**

IEventSystemHandler有许多子类接口，每一子类接口都有一个事件对应，与EventTrigger的触发功能一一对应。

* IPointerEnterHandler - OnPointerEnter
* IPointerExitHandler - OnPointerExit
* IPointerDownHandler - OnPointerDown
* IPointerUpHandler - OnPointerUp
* IPointerClickHandler - OnPointerClick
* IBeginDragHandler - OnBeginDrag
* IDragHandler - OnDrag
* IEndDragHandler - OnEndDrag
* IDropHandler - OnDrop
* IScrollHandler - OnScroll
* IUpdateSelectedHandler - OnUpdateSelected
* ISelectHandler - OnSelect
* IDeselectHandler - OnDeselect
* IMoveHandler - OnMove
* ISubmitHandler - OnSubmit
* ICancelHandler - OnCancel

**【代码示例】**

相当于EventTrigger功能，但由于EventTrigger的事件动态添加不方便，很多时候我们都是自定义一个自己的事件监听器。如果你需要监听按钮点击功能，就继承IPointerClickHandler接口，当用户触发该操作时，OnPointerClick函数将会被调用。

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.EventSystems;  using UnityEngine.Events;  using System;  public class EventTriggerListener : MonoBehaviour, IPointerClickHandler, IPointerUpHandler, IDragHandler, IBeginDragHandler ,IPointerDownHandler  {  public UnityAction onClick;  public UnityAction onPressDown;  public UnityAction<PointerEventData> onDrag;  public UnityAction<PointerEventData> onBeginDrag;  public UnityAction<PointerEventData> onPointerUp;  public void OnPointerClick(PointerEventData eventData)  {  if (onClick != null)  {  onClick();  }  }  public void OnBeginDrag(PointerEventData eventData)  {  if (onBeginDrag != null)  {  onBeginDrag(eventData);  }  }  public void OnPointerUp(PointerEventData eventData)  {  if (onPointerUp != null)  {  onPointerUp(eventData);  }  }  public void OnDrag(PointerEventData eventData)  {  if (onDrag != null)  {  onDrag(eventData);  }  }  public void OnPointerDown(PointerEventData eventData)  {  if (onPressDown != null)  {  onPressDown( );  }  }  } |

# 5.2 UI进阶

## 5.2.1图集

### 1. 什么是图集

在使用3D技术开发2D游戏或制作UI时（即使用GPU绘制），都会使用到图集 ，那么什么是图集呢？准确的说法图集是一张包含了多个小图的大图和一份记录了每个小图id、位置、尺寸等数据的数据文件 。

### 2. 为什么要用图集

在GPU已经成为PC、手机等设备的必备组件的现在，把所有显示的绘制操作交给专门处理图像的GPU显然比交给CPU更合适，这样空闲下来的CPU可以集中力量处理游戏的逻辑运算。

而GPU处理图像的做法和CPU是不一样的，在GPU中，我们要绘制一个图像需要提交图片（即纹理）到显存，然后在进行绘制（这个过程称为一次DrawCall），那么如果我们一帧要绘制100个就需要提交100次图片，如果使用包含了这100图片的图集，只需要一次提交即可，即一次DrawCall就搞定，处理效率上会有很大的提升。

另外使用图集也方便管理和归类各个模块的图片，可以通过一次加载和一次卸载完成多个图片的处理，同理，加载次数也下来了，可以提升运行效率。

### 3. CPU与GPU的限制

GPU一般具有填充率(Fillrate)和内存带宽(Memory Bandwidth)的限制，如果你的游戏在低质量表现的情况下会快很多，那么，你很可能需要限制你在GPU的填充率。

CPU一般被所需要渲染物体的个数限制，CPU给GPU发送渲染物体命令叫做DrawCalls。一般来说DrawCalls数量是需要控制的，在能表现效果的前提下越少越好。通常来说，电脑平台上DrawCalls几千个之内，移动平台上DrawCalls几百个之内。这样就差不多了。当然以上并不是绝对的，仅作一个参考。

### 4. 如何使用图集

在Unity中我们只要使用小图片即可，可以通过设置图片的Packing Tag来指定小图会被打包到的图集，比如2个小图的Packing Tag都叫“MyAtlas”，则Unity会将这两个小图打包到名为“MyAtlas”的图集中。

注意图片不能放在Resources文件夹下面，Resources文件夹下的资源将不会被打入图集。

是否打包图集的控制选项：Editor->Project Settings 下面有sprite packer的模式。Disabled表示不打包图集，Enabled For Builds 表示只有打包应用的时候才会打包图集，Always Enabled 表示始终会打包图集。

在Windows->Sprite Packer 里，点击packer 在这里你就可以预览到你的图集信息，图集文件被保存在和Assets文件夹同级的目录，Libary/AtlasCache里面。图集的大小还有图集的格式等等很多参数我们都是可以控制的，也可以通过脚本来设置。

通过设置我们就可以发现多个同一Packing Tag的小图放到场景中只会消耗一个DrawCall，表示我们的图集已经开始起作用了。

## 5.2.1图片格式

Unity3D引擎对纹理的处理是智能的：不论你放入的是PNG，[PSD](http://psd.knowsky.com/)还是TGA，它们都会被自动转换成Unity自己的Texture2D格式。

在Texture2D的设置选项中，你可以针对不同的平台，设置不同的压缩格式，如IOS设置成PVRTC4，Android平台设置成RGBA16等。嗯，非常的智能。

tga是无损支持透明的无压缩格式，dds有一点点压缩，png是无损压缩，对效率要求高的可以tga，dds，ui一般png，psd不可取，2的n次方幂天然被GPU接受。



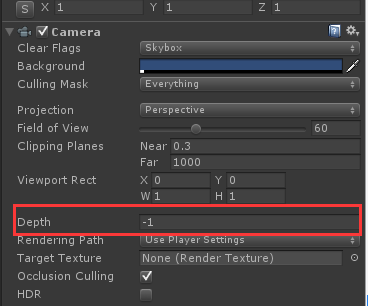
### 5.2.3渲染顺序

### 1. 为什么要关心渲染顺序

* 如果是2D游戏，渲染顺序关系着每个层次的显示先后，比如UI在游戏内容前面，游戏内容又有多层次。举一个简单的例子，在横版2D游戏中，经常会用到多层滚动的背景，把游戏物体分层管理起来，可以有效的减少出错几率，很好的控制显示效果。
* 对于3D游戏，游戏内容是3D的，UI一般是2D，有很多时候需要把某个模型啊，粒子特效啊，放在界面上，这样就有一个问题，3D物体和2D界面的先后关系，比如有些界面是在模型之上的，有些在下面，尝试过很多种办法，都能实现需求，但不是每种办法都是那么舒服的。

### 2. 控制渲染顺序的方式

Camera是unity中最优先的渲染顺序控制。depth越大，渲染顺序越靠后。



**sortingLayer 和 sortingOrder**

按照字面意思是层的排序，Canvas和Renderer都有这个属性。

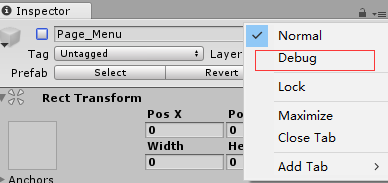
Canvas中的sorting layer可以控制Canvas的层级

Sorting order可以控制UI和粒子的层级

**RenderQueue**

这是unity中的一个概念，大致意思就是渲染顺序 。

所以一般设置材质的renderQueue或直接在shader中设置。



（可以在Debug模式下直接修改）

在ShaderLab中，有4个提前定义好的render queue，你可以设置更多的在他们之间的值：

Background ：表示渲染在任何物体之前

Geometry（default）：渲染大多数几何物体所用的render queue

AlphaTest：用于alpha测试

Transparent：用于渲染半透明物体

Overlay：渲染所有物体之上

**空间深度**

在摄像机坐标系下的Z轴，控制着该相机下的物体的深度，在fragment shader中进行深度测试，这样就控制了渲染到屏幕的顺序。

## 5.3.5实现转动圆盘效果

### 1. 圆盘转动数学知识

圆盘转动主要是对我么在圆盘上输入的位置点和之前的位置点做处理，让UI知道我们是往哪个方向转动的。曾经入门Unity的时候没有太多的经验，使用的是以圆盘中心划分四个象限，通关鼠标的偏移来做的旋转。现在来看，感叹当初游戏数学知识不过关，为了避免各位读者也走我之前的老路，我先普及一个数学概念。

**向量的点乘和叉乘**

点乘：

C://Users/Administrator/AppData/Local/YNote/data/qq9290E963D9D9CCC99F4283B9929E2D29/0f34f020632143089da2effad91cb8c0/clipboard.png

两个向量点乘得到一个标量 ，数值等于两个向量长度相乘后再乘以二者夹角的余弦值

公式里面有个余弦值，我们可以利用余弦的特性判断出：

若为 o，则表示两个向量之间角度是垂直关系， 。

若小于0  ，则两个向量夹角大于90°。

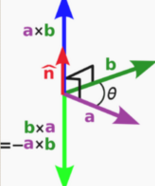
若结果 >0  ，则两个向量夹角小于 90°

**总结：**点乘通常用来判断角度，在判断物体是否转动到目标角度、虚拟摇杆在某个方向上的力度，或者shader里面对光线做处理都会用到这个。

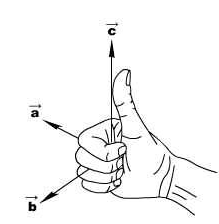
叉乘：

C://Users/Administrator/AppData/Local/YNote/data/qq9290E963D9D9CCC99F4283B9929E2D29/6b90ef184d804345afc0f8d34c3fb37f/clipboard.png

数学老师曾说过，两个向量叉乘确定一个平面，而平面是有方向的。



a X b不等于b X a,因为确定的平面方向是刚好相反的，比如a X b我们根据右手定则，得到的是向上的向量，若b X a我们的右手就是向着a向量方向收拢，得到是向下的向量。



**总结：**叉乘通常判断向量的方向，判断a在b向量的相对位置通过叉乘的正负判断是非常有效的。

### 2. 代码原型实现

我们只用获取到鼠标开始拖拽转盘UI时，每次鼠标偏移位置带来的角度变化，以及转盘左转还是右转，因为我们想让转动是单方向的，就直接转动为逆时针的时候直接return，如果要任意转动，则将return的地方改为angle=-angle。

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.EventSystems;  public class TurnAround :NoteLogic  {  /// <summary>  /// UI上的圆盘物体  /// </summary>  public RectTransform uiDisk;  /// <summary>  /// 对UI监听的一个重写（可以下载源码查看）  /// </summary>  public EventTriggerListener eventTrigger;  /// <summary>  /// 上一帧点击的方向  /// </summary>  Vector2 perTouchDir;  /// <summary>  /// 圆盘中心点位置  /// </summary>  Vector2 uiDiskPos;  /// <summary>  /// 当前转动角度记录  /// </summary>  float curAngle;  /// <summary>  /// 和上篇一样，通过重构虚函数，让操作部分实现不同Node不同操作。  /// </summary>  public override void OnJudgetOperation()  {  //将Ugui的开始拖拽事件和拖拽事件绑定功能函数  eventTrigger.onBeginDrag = BegionTurn;  eventTrigger.onDrag = Turn;  uiDiskPos = uiDisk.transform.position;  //协程判断时间是否超出，超出则直接判定当前分数  StartCoroutine(OnDelayCall(judgeTime));  }  IEnumerator OnDelayCall(float rTime)  {  yield return new WaitForSeconds(rTime);  JudgetStore();  }  /// <summary>  /// 开始拖拽UI的时候，储存一个点击点到转盘中心点的向量  /// </summary>  public void BegionTurn(PointerEventData rData)  {  perTouchDir = rData.position - uiDiskPos;  }  /// <summary>  /// 转动处理  /// </summary>  /// <param name="rData"></param>  public void Turn(PointerEventData rData)  {  //得到当前鼠标到转盘中心的向量  var curTouchDir = rData.position - uiDiskPos;  //上一帧向量和当前帧向量叉乘得到一个判断当前向量位置值  var crossValue = Vector3.Cross(perTouchDir, curTouchDir);  //获取两帧之间转动的角度（Unity自带的函数，原理其实可以用点乘来搞定）  float angle = -Vector3.Angle(perTouchDir, curTouchDir);  //如果逆时针旋转，不响应  if (crossValue.z > 0)  {  return;  }    //让ui旋转角度  uiDisk.transform.Rotate(new Vector3(0, 0, angle));  //保存当前帧向量  perTouchDir = curTouchDir;    curAngle -= angle;  //提示的圆圈UI根据转动的值由大变小  circleTipObj.gameObject.transform.localScale = Vector3.one \* Mathf.Lerp(10, 0, curAngle / targetValue);  //转动到目标值判定分数  if (curAngle > targetValue)  {  JudgetStore();  }  }  public void JudgetStore()  {  if (curState == eState.Over)  {  return;  }  if (curAngle > targetValue)  {  curScore = eScore.Perfect;  }  else if (curAngle > targetValue\*0.6f)  {  curScore = eScore.Good;  }  else  {  curScore = eScore.Fail;  }  StopCoroutine("OnDelayCall");  SetCurState(eState.Over);  }  } |

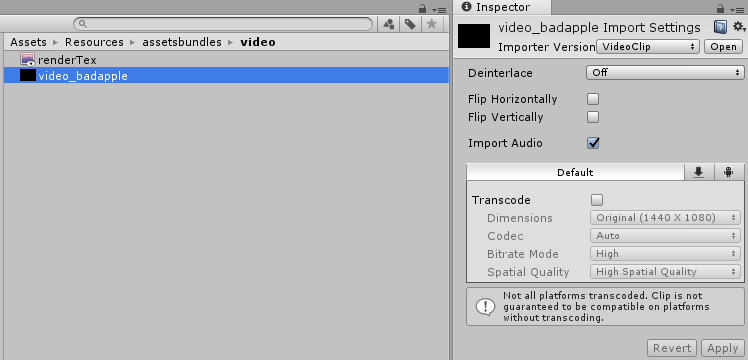
### 3. VideoPlayer视频播放

视频播放曾经是Unity的一个坑点，PC和手机平台的视频播放采用两种方式。电脑上是MovieTexture格式还得装QuickTime，手机上则是利用Handheld.PlayFullScreenMovie接口播放而且还是直接停止代码运行（怎么做功能啊喂！）。项目中为了用到视频最后采用了MobileMovieTexture插件，但是效果有点差，得用专门的格式，还得放在流文件夹，而且音效还得单独制作。

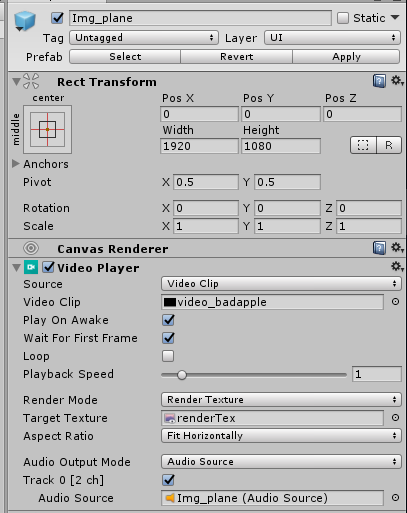
但是之前5.6更新的时候，新出的VideoPlayer功能搞定了以上的问题，PC端和手机端播放视频的方式都做了统一，而且支持摄像机镜头播放，UGUI播放等多种播放方式，同时还能直接搞定音频。简直太方便了！

实现流程：

① 放入视频文件，这里我将视频转换成了Unity支持的格式MP4。

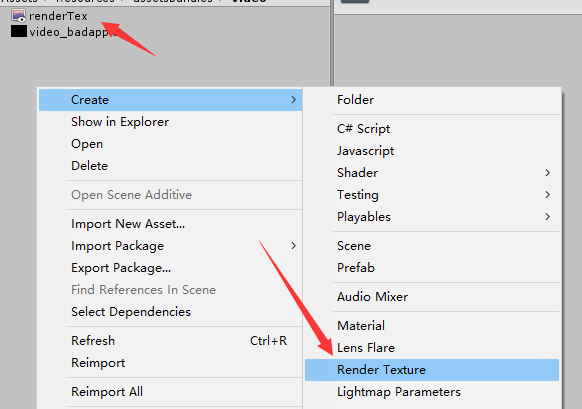


② 新建VideoPlayer组件，简单介绍下组件的功能。

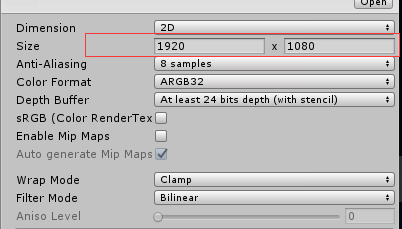


|  |  |
| --- | --- |
| **参数** | **功能** |
| Source | 可以选择是本地视频还是URL视频，如果是URL视频只用下方写上地址即可 |
| Wait For First Frame | 勾选PlayOnAwake后有效，避免加载视频过程中直接播放卡掉前几帧 |
| PlaybackSpeed | 视频播放速度 |
| RenderMode | 视频是哪里方式渲染的，目前有再摄像机上渲染，渲染到贴图上，渲染到材质球，和OnlyAPI（没对这个接口做研究） |
| AspectRatio | 长宽比，是宽度适配还是高度适配 |
| Audio OutPut Mode | 音频输出方式，这里选用通关AudioSource输出 |

③ 因为我们将视频渲染到UI上，因此需要一个目标渲染的贴图，创建一个渲染贴图。



可以直接设置贴图的像素。

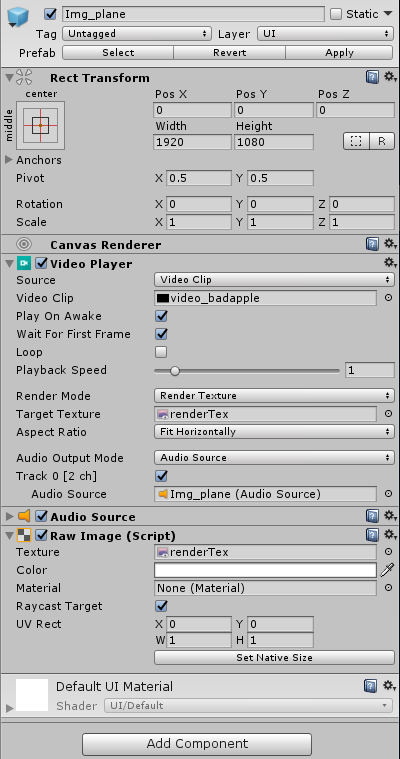


④ 创建一个UICanva当作视频播放的界面。

C://Users/Administrator/AppData/Local/YNote/data/qq9290E963D9D9CCC99F4283B9929E2D29/52921ba4acb54f42a9ef554b0b87ac09/clipboard.png

然后创建一个RawImage组件，挂载VideoPlayer组件，和音频组件

RawImage的贴图可以是任意类型，不一定是Sprite类型，因此我们没有选用Image组件。



这样我们就可以实现视频播放了，是不是特别简单？

### 4. 参数功能

|  |  |
| --- | --- |
| **VedioPlay事件名** | **功能** |
| errorReceived | 错误监听到时被执行 |
| frameDropped | 有丢帧发生时被执行 |
| frameReady | 新的一帧准备好时被执行 |
| loopPointReached | 播放结束或播放到循环的点时被执行 |
| prepareCompleted | 视频准备完成时被执行 |
| seekCompleted | 查询帧操作完成时被执行 |
| started | 在Play方法调用之后立刻调用 |