UMG系统（二）

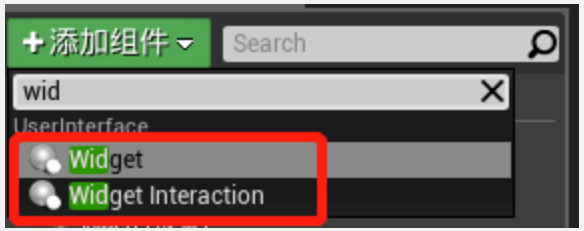
虚幻四高级程序开发专业

01

3D UI

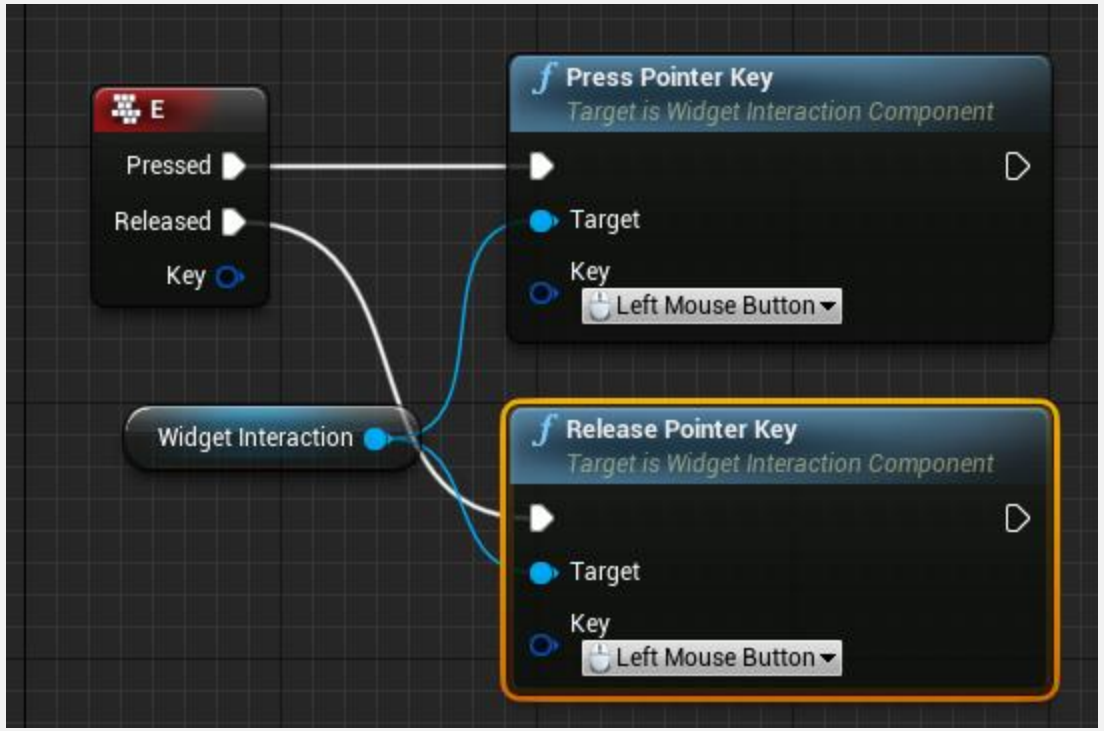
Widget组件

虚幻中提供了一种方便构建空间UI的解决方案，借助组件Widget用来将UMG控件显示在空间中（World和Screen）。借助组件Widget Interaction来完成3D交互



模拟点击事件

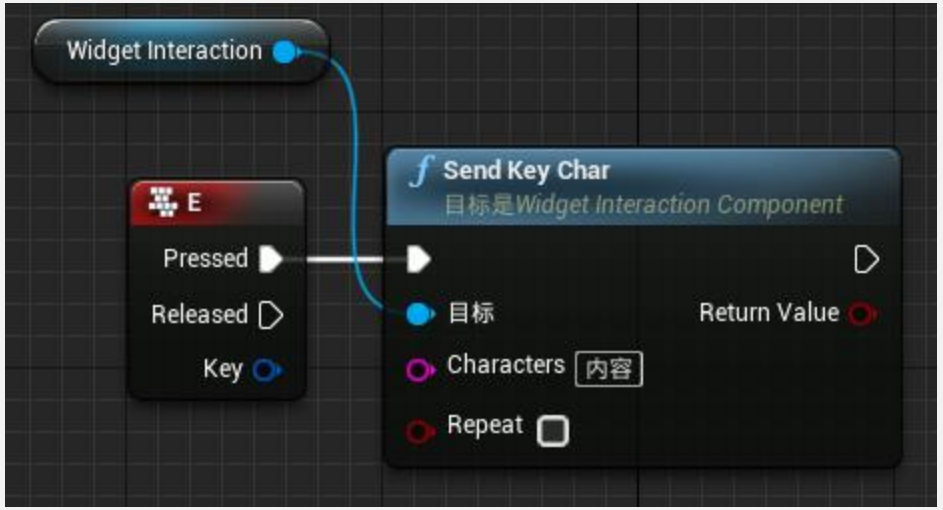
借助组件Widget Interaction进行模拟交互，完成点击事件交互。注意控件的响应一般是响应抬起事件

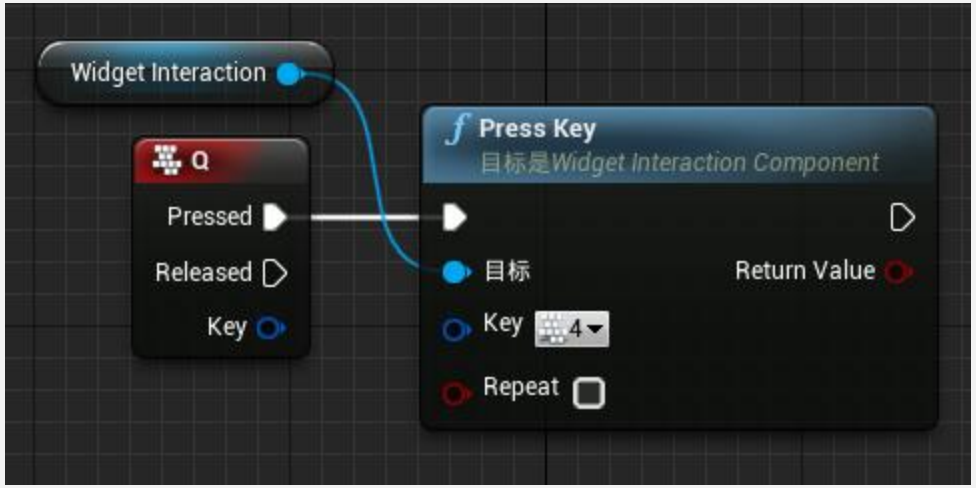


模拟键盘事件

按键输入模拟 Press Key/ Release Key 为输入框进行输入模拟 如同Send Key Char这不是当作敲击来使用，而是当作字符输入来使用

输入字符 Send Key Char 向界面输入文本内容





02

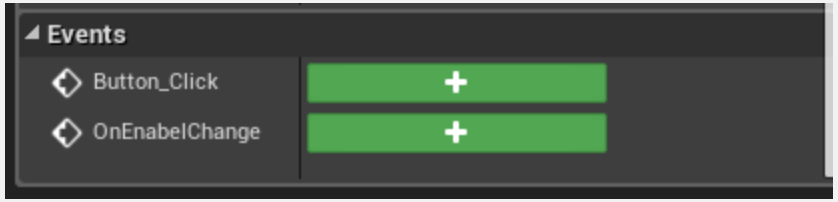
自定义控件交互

构建调度器暴露外部事件

在编写控件时，如果在控件中构建了调度器，那么当此控件被当做其他控件的子控件时，调度器将会被当做事件直接暴露到另一个控件中。可以直接在细节面板中选中实现控件。

A控件中的事件调度器

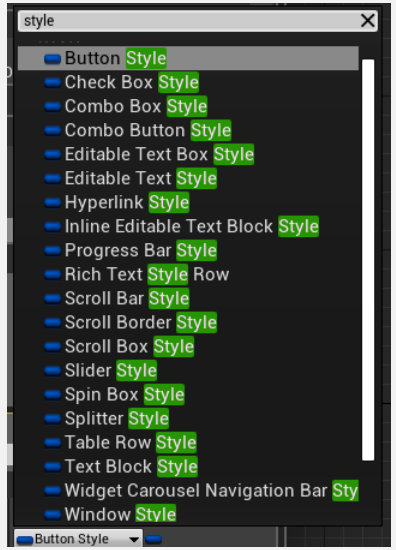
将A控件添加到B控件中，在B控件中查看A控件的细节面板，将会获得事件绑定快捷入口

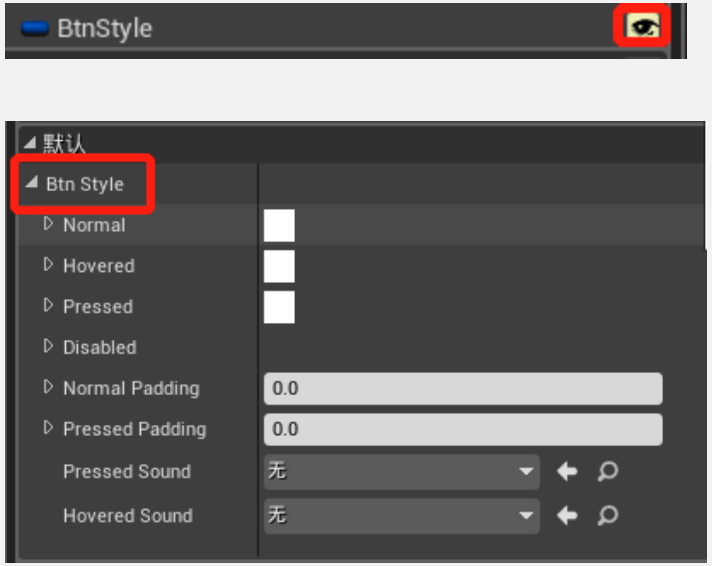


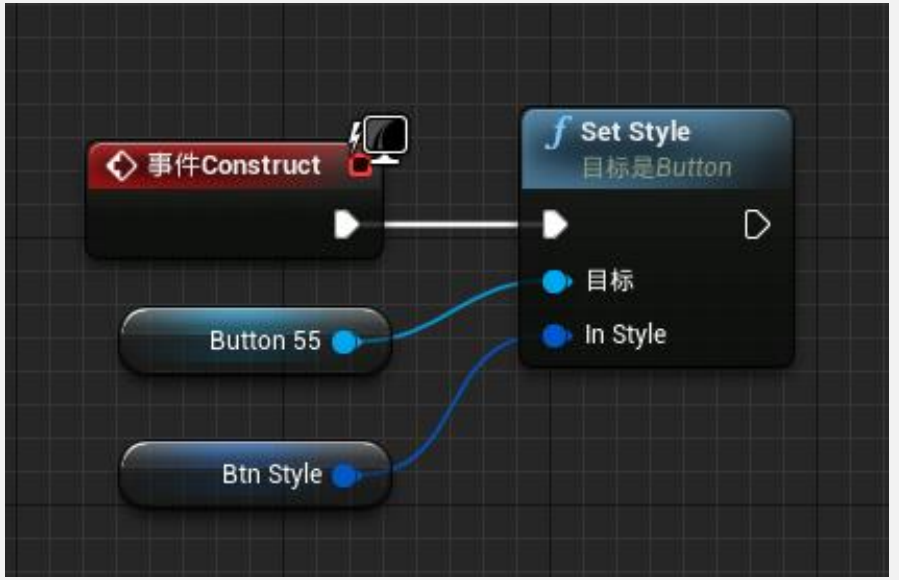
暴露样式

有时我们需要将自定义控件中的控件样式暴露到外部，让使用此控件的人来决定控件样式！那么我们可以在控件中直接构建**控件样式**属性，那么我们可以直接点击暴露属性。当此控件被添加到其他控件中时，属性将能在细节面板中看到。然后我们在自定义的控件中，将样式应用即可！

构建属性，**选择类型**，并将**属性暴露到外部**，当控件添加到其他面板时可以看到控件样式设定，**启动时设置样式（链接到Construct只有运行能看到效果，如果链接到Pre Construct上则可以在编辑时看到效果）**







03

UI动画

04

控件拖拽

拖拽逻辑

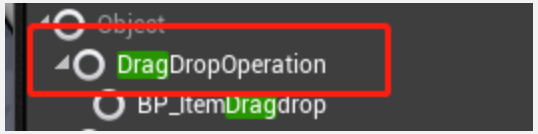
控件的拖拽，在游戏开发中非常常见。我们在运行游戏时经常需要将控件的位置（层级关系）进行调整，显而易见，拖拽操作是最好的解决方案！在虚幻中，完成拖拽非常的简单，我们只需要操作几步即可完成拖拽的操作。

- 构建拖拽数据对象
- 开启拖拽测试响应
- 响应抬起事件
- 检测是否存在拖拽
- 完成拖拽结果

构建拖拽数据

拖拽数据是用来将控件从已在位置向其他位置转移时的参考！控件在拖拽时并没有真正的离开父容器，而是只在抬起时决定是否离开（这是正常的软件设计必须要考虑的，否则用户误操作无法解决）。在拖拽过程中，拖拽数据帮助进行了有效的桥接。

创建蓝图类DragdropOperation，用于在拖拽过程中进行拖拽交互，拖拽交互数据均来自此类



开启拖拽检测

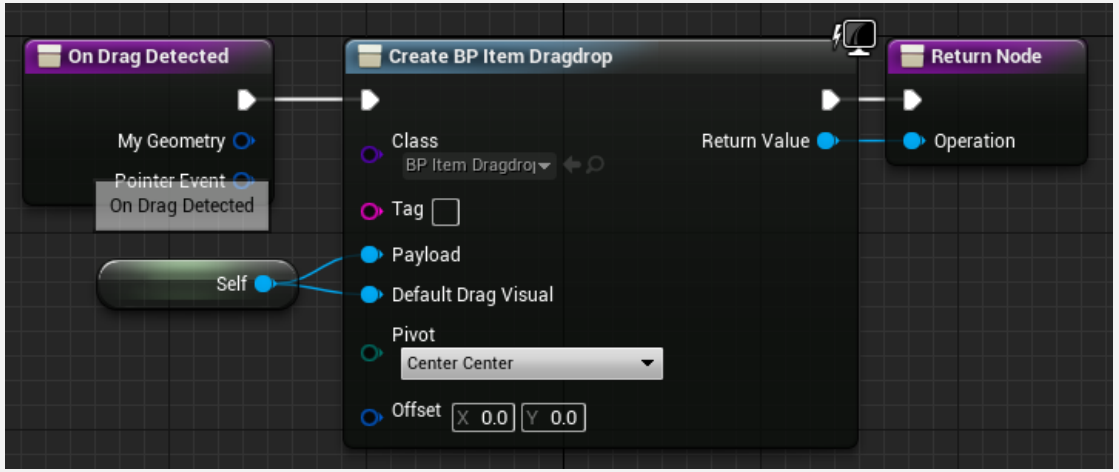
当鼠标或是其他事件发生时，可以在控件中调用监听拖拽产生，当产生拖拽时才开始执行对应的逻辑

在被拖拽的控件中重写OnMouseButtonDown

目的：当控件收到点击事件时，进行检测拖拽，如果使用的是鼠标左键，则响应拖拽事件

响应拖拽检测

当产生拖拽时，需要重写拖拽函数，完成响应，并且构建拖拽数据



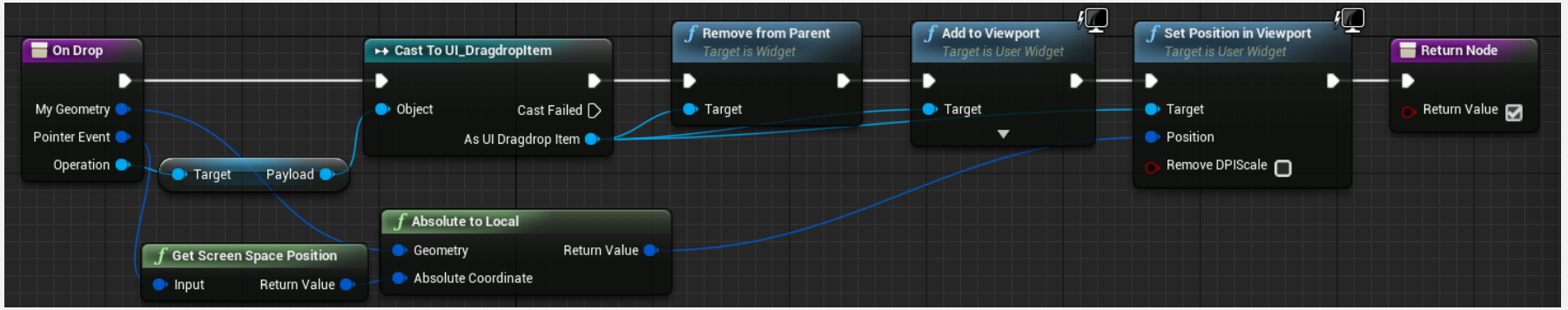
Payload 用于在响应抬起时，进行数据交互，Visual是拖拽时您想要显示的虚拟体

响应抬起

如果希望将控件拖到其他控件上，需要先将对方控件的上层容器可视开启，以便能够正常接收鼠标拖拽事件，并将事件进行传递给子控件

完成拖拽

在接受拖拽完毕的控件中，重写如下的函数



注意：由于DPI影响，GetScreenSpacePostion是有缩放的（基准比率是1920*1080，是1：1），所以屏幕小的时候，获取的位置会被放大，使用ATL转换到当前控件坐标，由于转换过程中没有忽略DPI，所以在设置的时候需要去掉Remove DPIScale选项。

05

C++操作UMG

获取编辑器创建控件

在编写逻辑时，我们可能需要在C++中完成逻辑的构建，在蓝图中完成UI的结构拼凑（这很灵活）。如果控件在蓝图中添加，我们需要在C++中使用时，我们可以通过以下两种办法完成获取。注意：获取方案需要控件是在C++中进行构建

1.通过名称获取控件

```
UButton* Btn = Cast<UButton>(GetWidgetFromName(TEXT("NativeBtn")));//获取蓝图中添加的一个控件名称是“NativeBtn”的Button
```

2.通过宏绑定控件（控件类型和名称必须和蓝图添加的一致，并且在蓝图中必须添加同名同类型控件，标记BlueprintReadWrite不是必须）

```
UPROPERTY(BlueprintReadWrite, meta=(BindWidget))
class UButton* NativeBtn;
```

获取编辑器创建动画（一）

通过绑定方式也可以将编辑器中创建的动画在C++中获取，需要完成以下标记

注意：必须添加标记Transient

```
UPROPERTY(Transient, meta=(BindWidgetAnim))
UWidgetAnimation* ShowAnim;
```

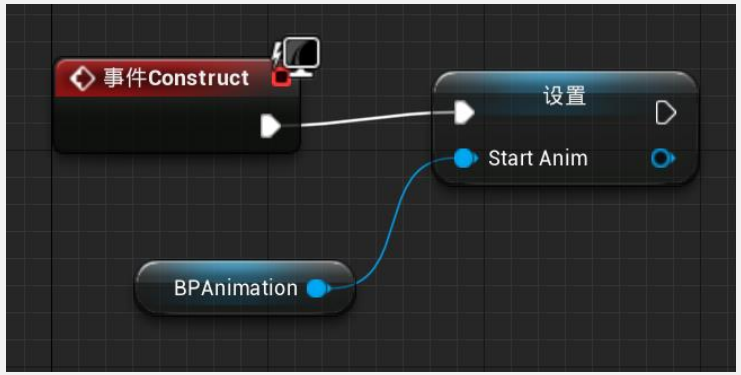
获取编辑器创建动画（二）

我们也可以通过序列化变量，然后到蓝图中设置（不建议用此方法，动画太多操作会越来越复杂），操作如下：

1.在C++中构建动画对象参数，并使用宏进行说明，在蓝图中进行设置

```
UPROPERTY(EditAnywhere, BlueprintReadWrite)
class UWidgetAnimation* StartAnim;
```

在蓝图中进行绑定（BPAnimation是在蓝图中构建）



获取编辑器创建动画

2.通过反射机制，直接读取蓝图中添加的成员参数

```
UProperty* Prop = GetClass()->PropertyLink;//获取当前类成员链
while (Prop != NULL)//遍历链
{
    if (Prop->GetClass() == UObjectProperty::StaticClass())//检查当前成员是否是属性
    {
        UObjectProperty* ObjectProp = Cast<UObjectProperty>(Prop);
        if (ObjectProp->PropertyClass == UWidgetAnimation::StaticClass())//检查当前成员属性是否是动画属性
        {
            UObject* Object = ObjectProp->GetObjectPropertyValue_InContainer(this);
            UWidgetAnimation* anim = Cast<UWidgetAnimation>(Object);//尝试转换类型
            if (anim)
            {
                UE_LOG(LogTemp, Log, TEXT("Property Name is %s"), *anim->GetDisplayName().ToString());
            }
        }
    }
    Prop = Prop->PropertyLinkNext;
}
```

感谢观看