主交互程序设计文档

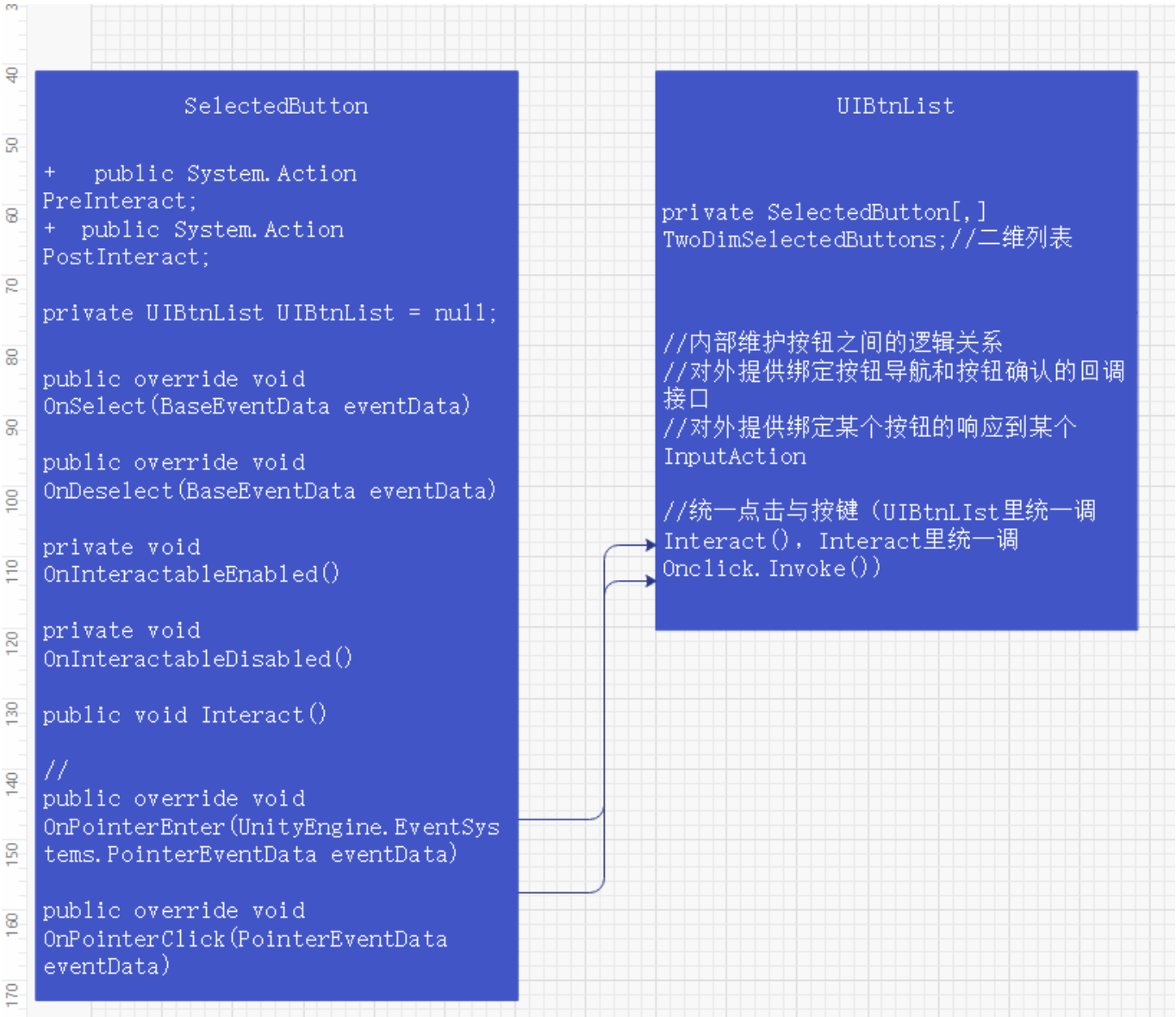
背景描述

目前完成了开始界面菜单，选项菜单，玩家最底层UI和加载界面UI的简单界面拼接，代码目前还差玩家最底层UI，但是所有代码仍有大量问题需要更改。

大致思路

1.开始界面菜单，选项菜单，玩家最底层UI中的按钮

这三个界面都有一个相同点，就是包含大量的按钮，目前使用的是SelectedButton。但是存在大量重复代码的问题 所以需要编写BtnList组件管理按钮



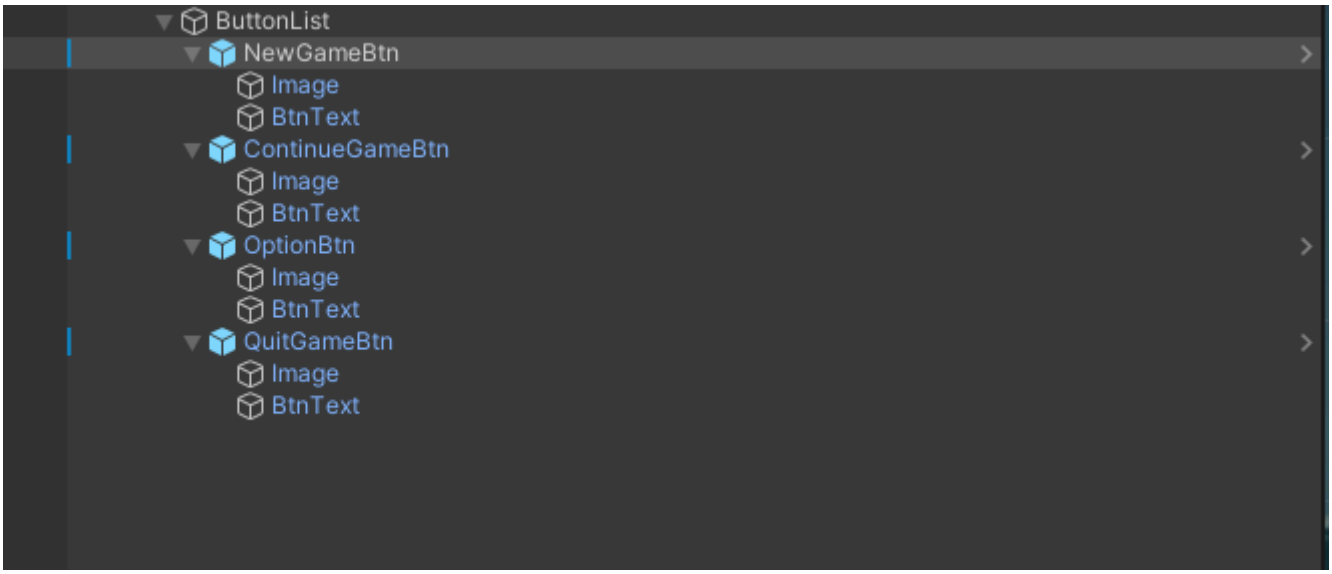
按钮文本描述json的载入并初始化

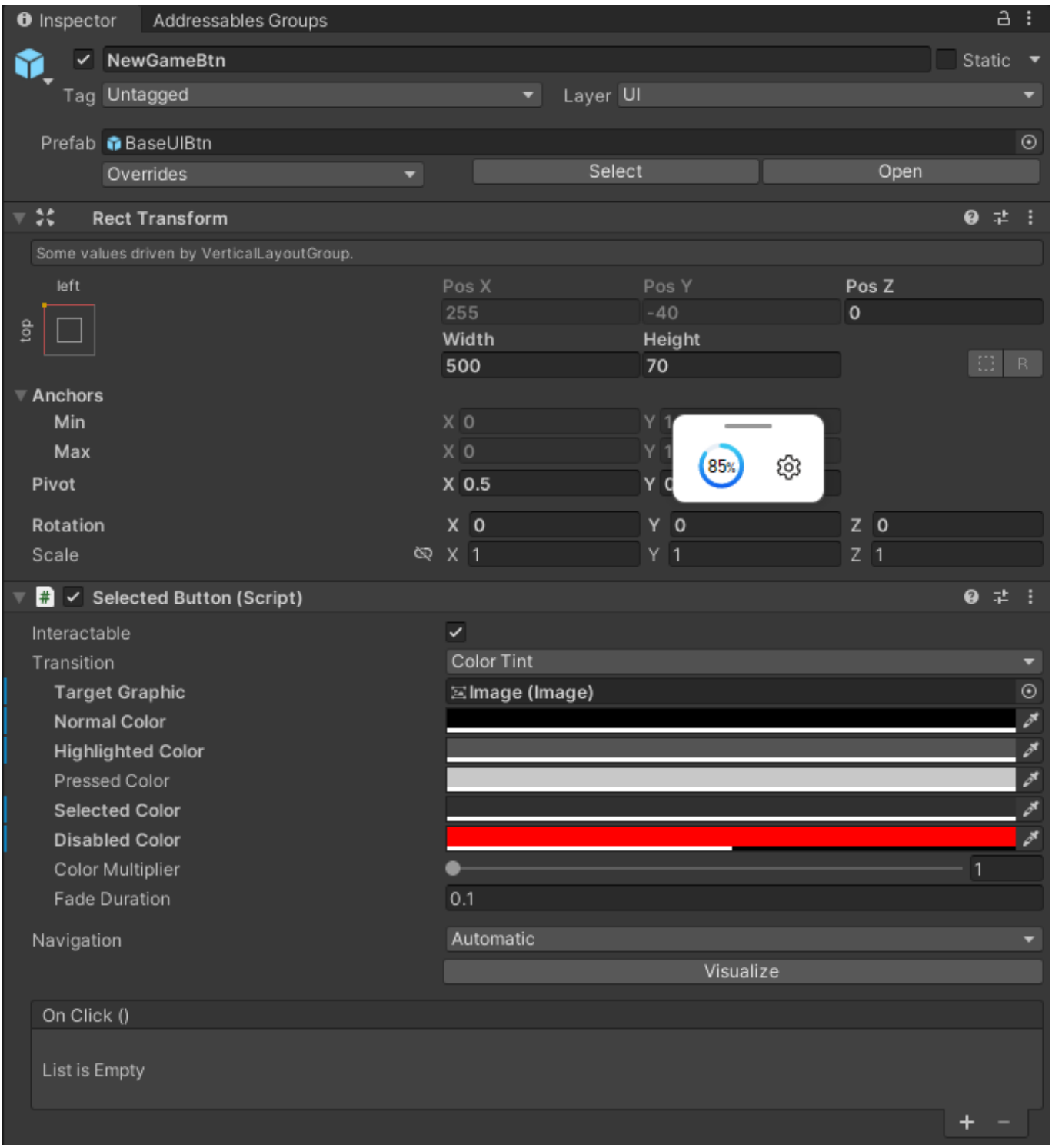
```
[System.Serializable]
3 个引用
public struct OptionPanelStruct
{
    public ML.Engine.TextContent.TextContent TopTitle;

    public TextTip[] Btns;

    //BotKeyTips
    public KeyTip Confirm;
    public KeyTip Back;
}
```

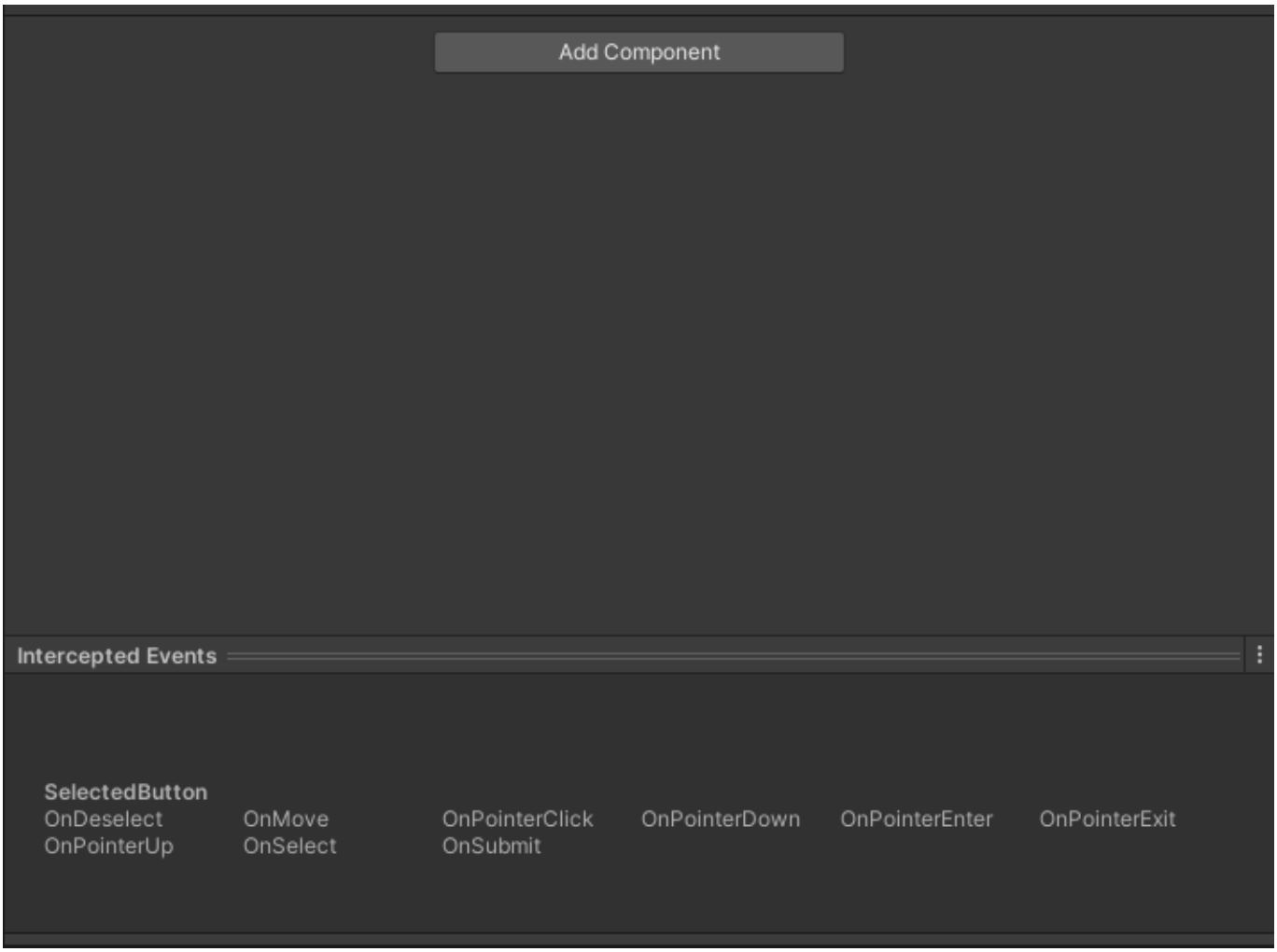
```
3 个引用
protected override void OnLoadJsonAssetComplete(OptionPanelStruct datas)
{
    base.OnLoadJsonAssetComplete(datas);
    InitBtnData(datas);
}
2 个引用
protected override void InitTextContentPathData()
{
    this.abpath = "ML/Json/TextContent";
    this.abname = "OptionPanel";
    this.description = "OptionPanel数据加载完成";
}
private Transform btnList;
private UIBtnList UIBtnList;
1 个引用
private void InitBtnData(OptionPanelStruct datas)
{
    foreach (var tt in datas.Btns)
    {
        this.UIBtnList.SetBtnText(tt.name, tt.description.GetText());
    }
}
#endregion
```

BtnList组件的初始化

BtnList组件获取整个ui中的按钮是在其构造器中进行，只需在配置预制体时配置好即可 按钮物体必须挂上SelectedButton脚本，并且物体名就是按钮名，之后在代码中可以用这个按钮名访问其对应的SelectedButton


ButtonList
NewGameBtn
Image
BtnText
ContinueGameBtn
Image
BtnText
OptionBtn
Image
BtnText
QuitGameBtn
Image
BtnText


Inspector
Addressables Groups
NewGameBtn
Static
Tag Untagged
Layer UI
Prefab BaseUIBtn
Overrides
Select
Open
Rect Transform
Some values driven by VerticalLayoutGroup.
left
top
Pos X 255
Pos Y -40
Pos Z 0
Width 500
Height 70
Anchors
Min X 0
Max X 0
Pivot X 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Selected Button (Script)
Interactable
Transition Color Tint
Target Graphic Image (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier 1
Fade Duration 0.1
Navigation Automatic
Visualize
On Click ()
List is Empty



BtnList组件构造器的参数

```
private bool isWheel = false;
/// <summary>
/// parent: 按钮父物体 limitNum: 一行多少个按钮 hasInitSelect:是否有初始选中 isWheel: 是否为轮转按钮 OnSelectedEnter: 选中回调 OnSelectedExit: 选出回调
/// </summary>
5 个引用
public UIBtnList(Transform parent, int limitNum = 1, bool hasInitSelect = true, bool isWheel = false, Action OnSelectedEnter = null, Action OnSelectedExit = null)
{
```

UIPanel中如何应用BtnList组件

首先需要在UIPanel申明一个BtnList UIBtnList,在UIPanel.OnEnter中调用BtnList的构造器 在UIPanel.Enter和UIPanel.Exit中分别调用UIBtnList.EnableBtnList()和UIBtnList.DisableBtnList()

```
#region Override
18 个引用
public override void OnEnter()
{
    UIBtnList = new UIBtnList(parent: btnList, limitNum: gridLayout.constraintCount);
    base.OnEnter();
}

14 个引用
protected override void Enter()
{
    this.UIBtnList.EnableBtnList();
    base.Enter();
}

14 个引用
protected override void Exit()
{
    this.UIBtnList.DisableBtnList();
    base.Exit();
}
```

BtnList组件的输入响应

情况一：BtnList组件中的按钮的响应不需要与InputAction有关，比如开始菜单中的按钮，每个按钮的响应不应与选择按钮与确定响应按钮的InputAction发生联系，所以可以调用

```
/// <summary>
/// 该函数功能为绑定按钮导航InputAction的回调函数
/// </summary>
4 个引用
public void BindNavigationInputAction(InputAction NavigationInputAction, BindType bindType, Action preAction = null, Action postAction = null)
{
    this.NavigationPreAction = preAction;
    this.NavigationPostAction = postAction;
    this.NavigationInputAction = NavigationInputAction;
    this.NavigationBindType = bindType;

    switch (bindType)
    {
        case BindType.started:
            this.NavigationInputAction.started += isWheel ? this.RingNavigation : this.GridNavigation;
            break;
        case BindType.performed:
            this.NavigationInputAction.performed += isWheel ? this.RingNavigation : this.GridNavigation;
            break;
        case BindType.canceled:
            this.NavigationInputAction.canceled += isWheel ? this.RingNavigation : this.GridNavigation;
            break;
    }
}
```

```
/// <summary>
/// 该函数功能为绑定按钮确认InputAction的回调函数
/// </summary>
4 个引用
public void BindButtonInteractInputAction(InputAction ButtonInteractInputAction, BindType bindType, Action preAction = null, Action postAction = null)
{
    this.ButtonInteractPreAction = preAction;
    this.ButtonInteractPostAction = postAction;
    this.ButtonInteractInputAction = ButtonInteractInputAction;
    this.ButtonInteractBindType = bindType;

    switch (bindType)
    {
        case BindType.started:
            this.ButtonInteractInputAction.started += this.ButtonInteract;
            break;
        case BindType.performed:
            this.ButtonInteractInputAction.performed += this.ButtonInteract;
            break;
        case BindType.canceled:
            this.ButtonInteractInputAction.canceled += this.ButtonInteract;
            break;
    }

}
```

这两个函数可以将按钮导航和按钮确定响应的Inputaction的回调绑定，绑定的函数写在该组件中。导航回调分为RingNavigation和GridNavigation，分别对应格子导航和轮盘UI导航。按钮确认回调为ButtonInteract。

情况二：BtnList组件中的按钮的响应需要与InputAction有关，此时的需求是需要InputAction去触发某个按钮的响应函数。比如通知UI中的弹窗UI 可以调用

```
/// <summary>
/// 该函数功能为绑定按钮 对应哪个InputAction触发
/// </summary>
1 个引用
public void BindInputAction(string btnName, InputAction InputAction, BindType bindType, Action preAction = null, Action postAction = null)
{

    //统一点击与按键 并且加入preAction 与 postAction
    SelectedButton btn = GetBtn(btnName);
    btn.SetPreAndPostInteract(preAction, postAction);
    Action<InputAction.CallbackContext> buttonClickAction = (context) => { preAction?.Invoke(); btn.onClick.Invoke(); postAction?.Invoke(); };

    switch (bindType)
    {
        case BindType.started:
            InputActionBindDic.Add((InputAction, BindType.started), buttonClickAction);
            InputAction.started += buttonClickAction;
            break;
        case BindType.performed:
            InputActionBindDic.Add((InputAction, BindType.performed), buttonClickAction);
            InputAction.performed += buttonClickAction;
            break;
        case BindType.canceled:
            InputActionBindDic.Add((InputAction, BindType.canceled), buttonClickAction);
            InputAction.canceled += buttonClickAction;
            break;
    }

}
```

该函数可以让InputAction的响应与按钮的响应联系起来

2.PlayerUIBotPanel中的菜单

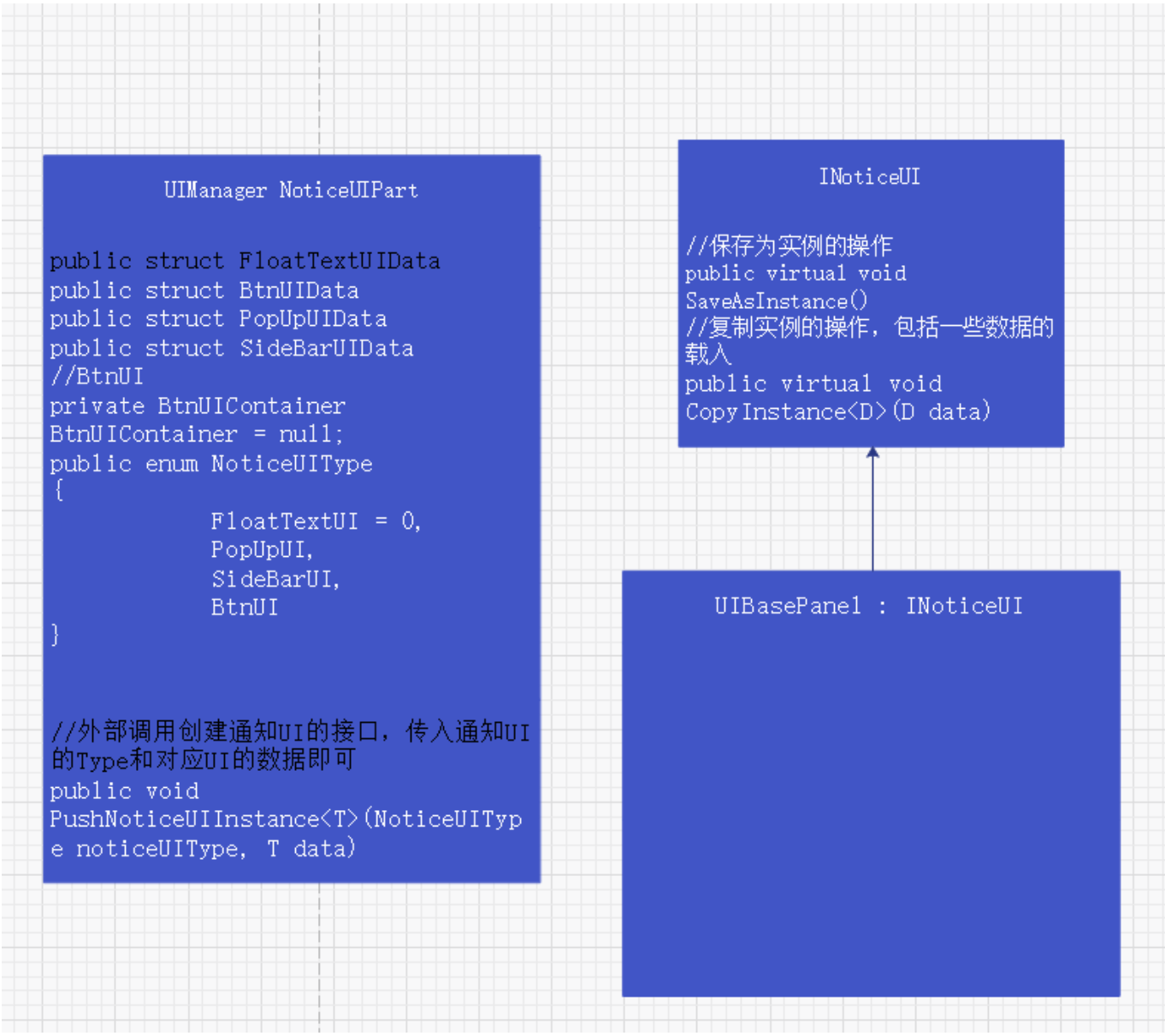
对于每个功能选项，考虑用上述的UISelectedButtonComponent实现，并将每个功能选项制作成预制体。同时需要将一个能够装载八个功能选项预制体的面板也制作成预制体。 应该需要一个维护当前所拥有功能的列表，每次打开菜单时，通过检测该列表实例化功能选项预制体，每八个建立一个预制体面板。内部翻页可以用一个下标来选择激活哪个预制体面板。

3.LoadingScenePanel的载入时机

LoadingScenePanel需要在切换场景开始时载入，切换场景完成后消失。 但是LoadingScenePanel并不进入UI栈，手动调其OnEnter与OnExit函数实现push与pop 利用GameManager.Instance.LevelSwitchManager.LoadSceneAsync函数切换场景 并用preCallback与postCallback实现载入与载出

```
this.NewGameBtn.OnInteract += () =>
{
    UIBasePanel panel = null;
    System.Action<string, string> preCallback = (string s1, string s2) =>
    {
        GameManager.Instance.EnterPoint.GetLoadingScenePanelInstance().Completed
+= (handle) =>
        {
            // 实例化
            panel = handle.Result.GetComponent<LoadingScenePanel>();

panel.transform.SetParent(GameManager.Instance.UIManager.GetCanvas.transform,
false);
            panel.OnEnter();
        };
    };
    System.Action<string, string> postCallback = async (string s1, string s2) =>
    {
        await System.Threading.Tasks.Task.Run(() =>
        {
            while (panel == null) ;
        });
        panel.OnExit();
    };

GameManager.Instance.StartCoroutine(GameManager.Instance.LevelSwitchManager.LoadSc
eneAsync("GameScene", preCallback, postCallback));
    this.OnExit();
};
```

4.通知UI



外部需要弹出通知UI时，调用public void PushNoticeUIInstance(NoticeUIType noticeUIType, T data)接口，传入通知UI类型和对应的数据即可

四种UI数据结构如下： … public struct FloatTextUIData { public string msg;

```
// 构造函数
public FloatTextUIData(string message)
{
    msg = message;
}
```

}

public struct BtnUIData { public string msg; public UnityAction action;

```
// 构造函数
public BtnUIData(string message, UnityAction act)
```

```
    {
        msg = message;
        action = act;
    }
```

}

public struct PopUpUIData { public string msg1; public string msg2; public List spriteList; public UnityAction action;

```
    // 构造函数
    public PopUpUIData(string message1, string message2, List<Sprite> sprites,
    UnityAction act)
    {
        msg1 = message1;
        msg2 = message2;
        spriteList = sprites;
        action = act;
    }
```

}

public struct SideBarUIData { public string msg1; public string msg2;

```
    // 构造函数
    public SideBarUIData(string message1, string message2)
    {
        msg1 = message1;
        msg2 = message2;
    }
```

} …

目前BtnUI仍需迭代，缺少手柄的交互