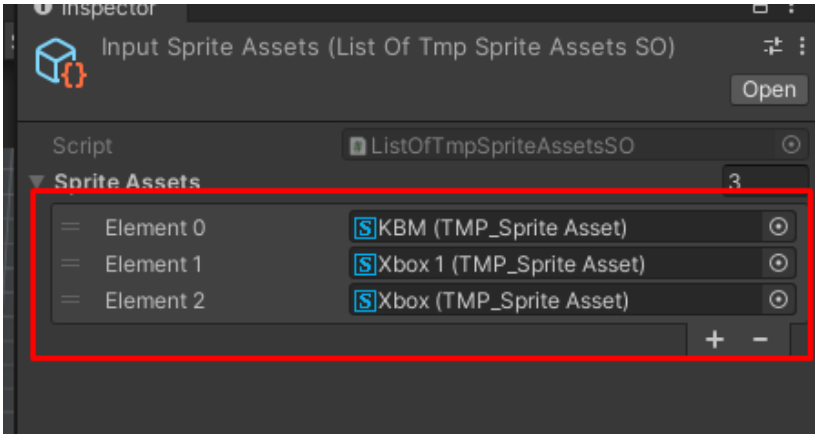
Dynamic Input Switching Documentation

How to run demo scene:

You need to import unity new input system package. If you haven't imported package manager

- Go to Window-> Package Manager -> select packages to Unity registry
- Find Input System and Installed it.
- Go to 2D Assets-> Sprite Assets copy the content of the file and paste it onto TextMeshPro -> Resources -> Sprite Assets
- Open InputSpriteAsset data and add KBM and Xbox sprite asset

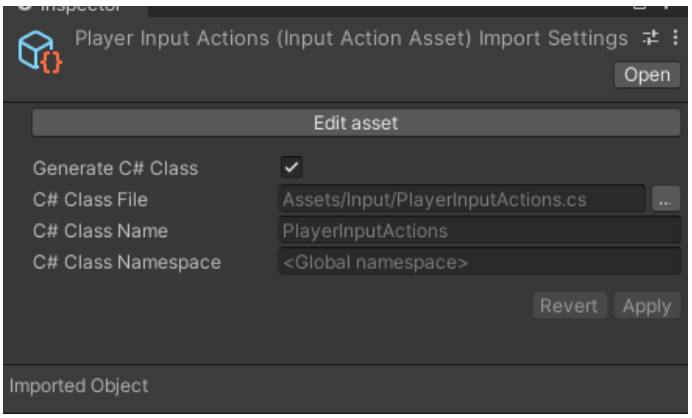


- After installing you can run Demo scenes by going through the Scenes folder

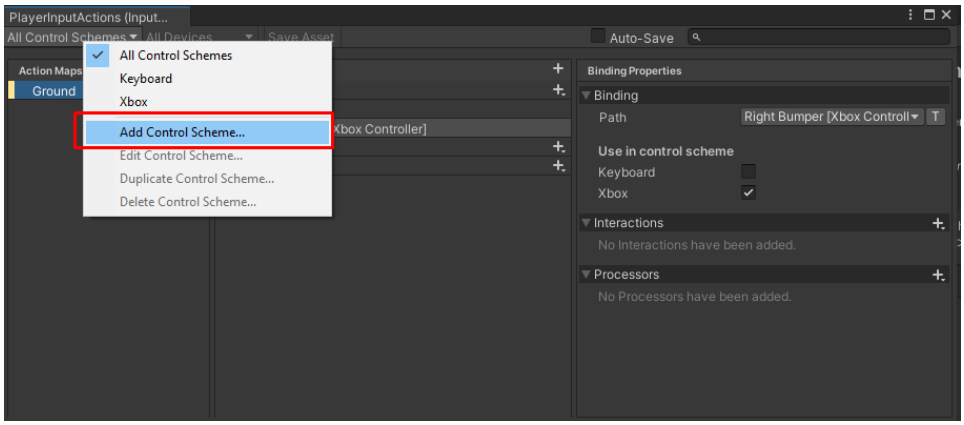
How to run on your own project:

- Install unity new input system If you haven't

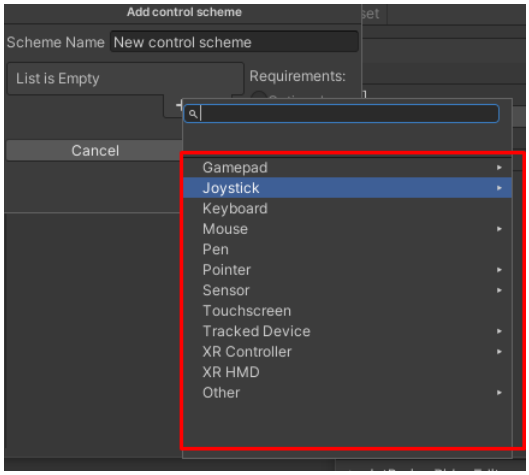
- Create Input Actions on any folder and generate C# class



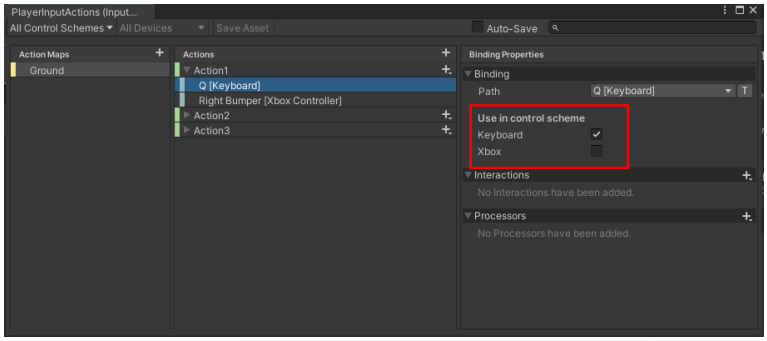
- After creating input actions open the input action and create control scheme



- Name the control scheme and add device list item as per as your project required

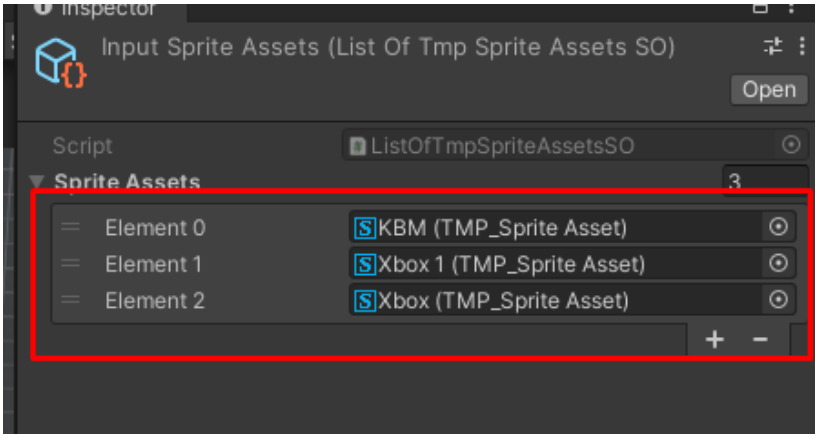


- After creating control scheme create action maps and actions as per as your project required
- Select the control scheme as per as your action meet the device requirements and save asset



- Go to 2D Assets-> Sprite Assets copy the content of the file and paste it onto TextMeshPro -> Resources -> Sprite Assets

- Open InputSpriteAsset data and add KBM and Xbox sprite asset

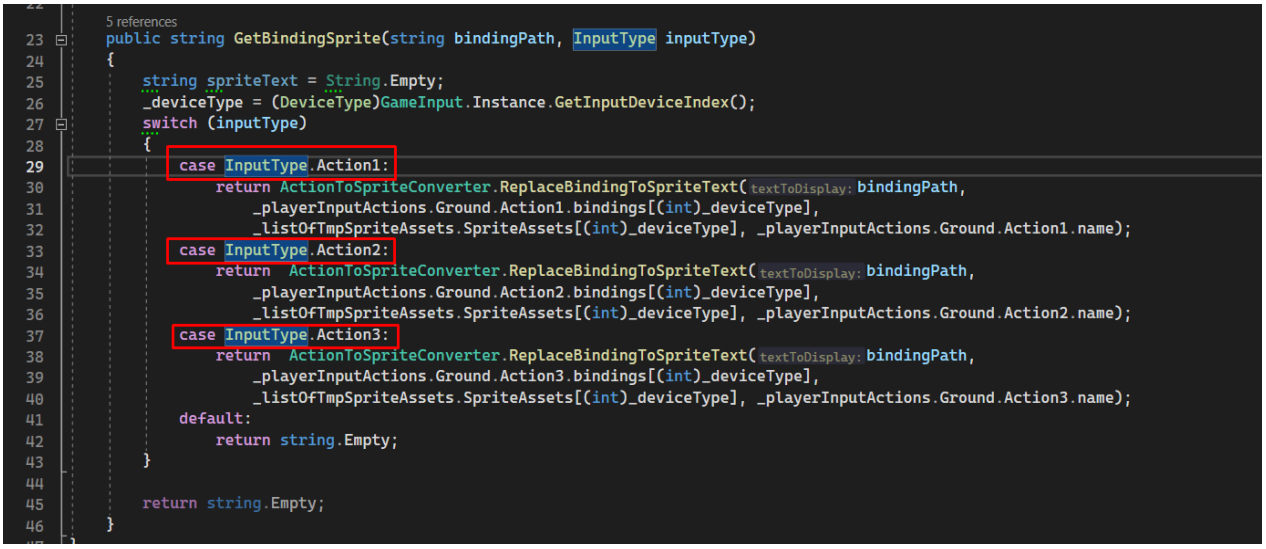


- Go to Enums.cs and add actions enum value as per as you've added InputActions

```
public enum InputType
{
    Action1, //Add any type of action required
    Action2,
    Action3
}

2 references
public enum DeviceType
{
    Keyboard = 0,
    Gamepad = 1,
    Xbox = 2,
    PS = 3
}
```

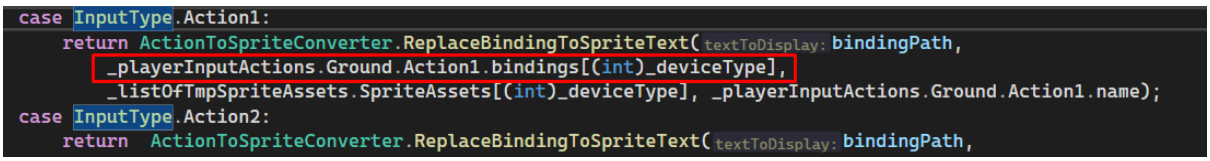
- Add DeviceType enum value as your project may runs on devices.

- Go to BindingDisplayHandler.cs and go to the GetBindingSprite() method and add actions you've added in Enum.cs



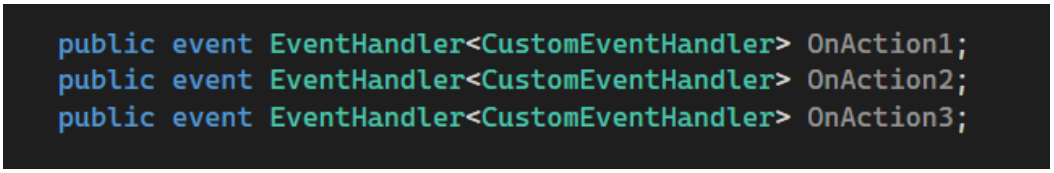
```
5 references
public string GetBindingSprite(string bindingPath, InputType inputType)
{
    string spriteText = String.Empty;
    _deviceType = (DeviceType)GameInput.Instance.GetInputDeviceIndex();
    switch (inputType)
    {
        case InputType.Action1:
            return ActionToSpriteConverter.ReplaceBindingToSpriteText(textToDisplay: bindingPath,
                _playerInputActions.Ground.Action1.bindings[(int)_deviceType],
                _listOfTmpSpriteAssets.SpriteAssets[(int)_deviceType], _playerInputActions.Ground.Action1.name);
        case InputType.Action2:
            return  ActionToSpriteConverter.ReplaceBindingToSpriteText(textToDisplay: bindingPath,
                _playerInputActions.Ground.Action2.bindings[(int)_deviceType],
                _listOfTmpSpriteAssets.SpriteAssets[(int)_deviceType], _playerInputActions.Ground.Action2.name);
        case InputType.Action3:
            return  ActionToSpriteConverter.ReplaceBindingToSpriteText(textToDisplay: bindingPath,
                _playerInputActions.Ground.Action3.bindings[(int)_deviceType],
                _listOfTmpSpriteAssets.SpriteAssets[(int)_deviceType], _playerInputActions.Ground.Action3.name);
        default:
            return string.Empty;
    }

    return string.Empty;
}
```

- Add action maps and binding according to your own action maps and bindings



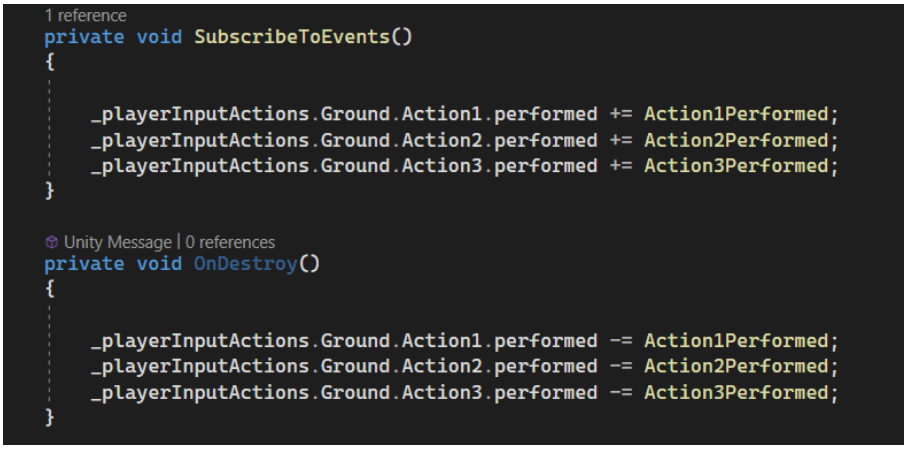
```
case InputType.Action1:
    return ActionToSpriteConverter.ReplaceBindingToSpriteText(textToDisplay: bindingPath,
        _playerInputActions.Ground.Action1.bindings[(int)_deviceType],
        _listOfTmpSpriteAssets.SpriteAssets[(int)_deviceType], _playerInputActions.Ground.Action1.name);
case InputType.Action2:
    return  ActionToSpriteConverter.ReplaceBindingToSpriteText(textToDisplay: bindingPath,
```

- Go to GameInput.cs and add events as per as your actions needed.



```
public event EventHandler<CustomEventHandler> OnAction1;
public event EventHandler<CustomEventHandler> OnAction2;
public event EventHandler<CustomEventHandler> OnAction3;
```

- Now register and unsubscribe your own action events by editing SubscribeToEvents() method and OnDestroy() method



```
1 reference
private void SubscribeToEvents()
{

    _playerInputActions.Ground.Action1.performed += Action1Performed;
    _playerInputActions.Ground.Action2.performed += Action2Performed;
    _playerInputActions.Ground.Action3.performed += Action3Performed;
}

Unity Message | 0 references
private void OnDestroy()
{

    _playerInputActions.Ground.Action1.performed -= Action1Performed;
    _playerInputActions.Ground.Action2.performed -= Action2Performed;
    _playerInputActions.Ground.Action3.performed -= Action3Performed;
}
```

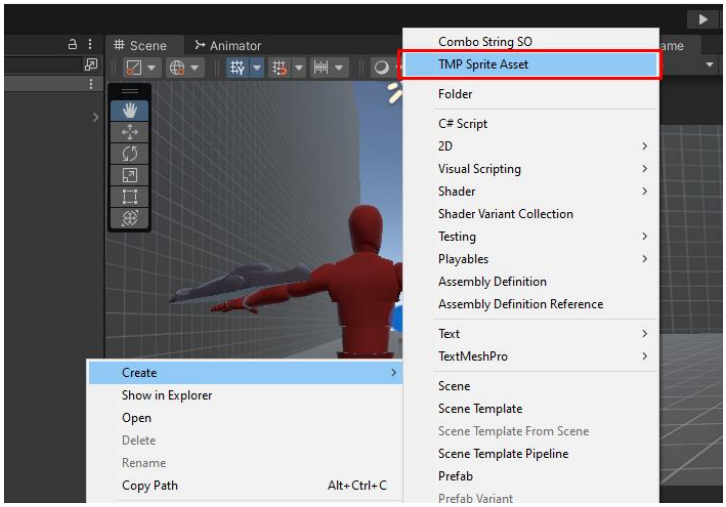
- Change actions callback method according to your own action callback method

```
2 references
private void Action2Performed(InputAction.CallbackContext obj)
{
    OnAction2?.Invoke(sender: this, e: new CustomEventHandler
    {
        DisplayString = GetBindingText(InputType.Action2)
    });
    EventActionHandler eventActionHandler = GetComponent<EventActionHandler>();
    if (eventActionHandler != null)
    {
        eventActionHandler.HandleInputAction(obj.action.name);

    }
}
2 references
```

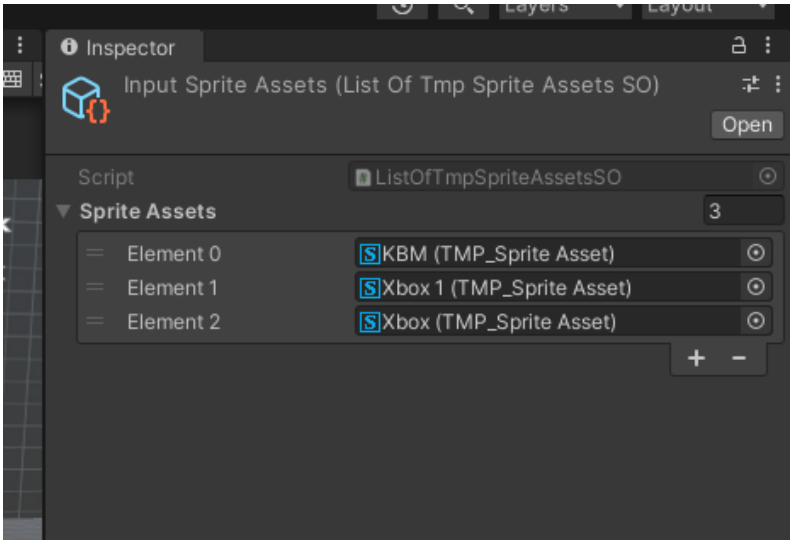
- Change your own action name in GetBindingText() method

```
8 references
public string GetBindingText(InputType inputType)
{
    switch (inputType)
    {
        case InputType.Action1:
            return GetDeviceBoundInputControl(_playerInputActions.Ground.Action1);
        case InputType.Action2:
            return GetDeviceBoundInputControl(_playerInputActions.Ground.Action2);
        case InputType.Action3:
            return GetDeviceBoundInputControl(_playerInputActions.Ground.Action3);
    }

    return $"Null";
}
```

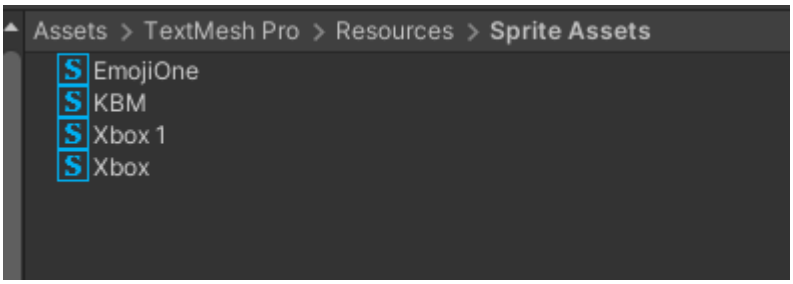
- Now create an empty gameobject onto the scene and add BindingDisplayHandler script.
- Create InputSprite Asset at any folder and add Sprite Assets onto the InputSpriteAsset



- Sprite Assets **must be placed on to the Textmeshpro Resources Folder.**



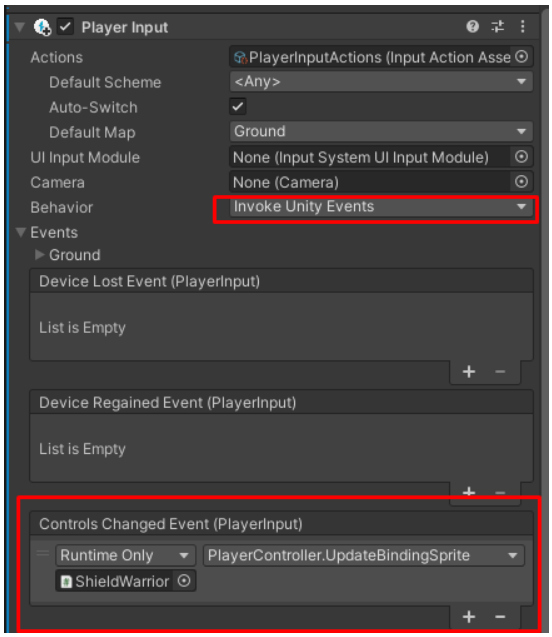
- Move Sprite Asset to the Textmeshpro -> Resources -> Sprite Assets folder



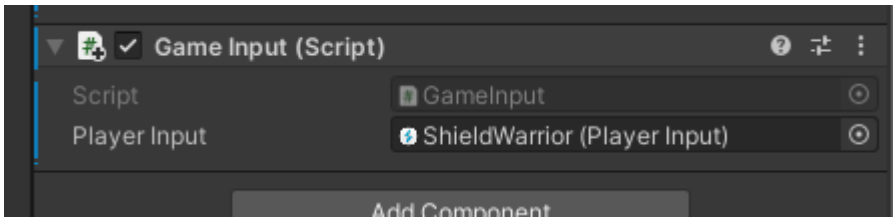
- In your controller class where input action will be defined you've to override the HandleInputAction() function. So inherit EventActionHandler abstract class and override the following function. The example code will be look like this:

```
//Define your actions here
4 references
public override void HandleInputAction(string contextName)
{
    Dictionary<string, Action> actions = new Dictionary<string, Action>()
    {
        { "Action1", () => {
            Debug.Log(message: "pressed action 1");
            _animator.SetTrigger(name: "Attack1");
        }},
        { "Action2", () => {
            Debug.Log(message: "pressed action 2");
            _animator.SetTrigger(name: "Attack2");
        }},
        { "Action3", () => {
            Debug.Log(message: "pressed action 3");

        }},
    };
    if (actions.ContainsKey(contextName))
    {
        actions[contextName].Invoke();
    }
    else
    {
        Debug.LogError(message: "Event action not defined");
    }
}
```

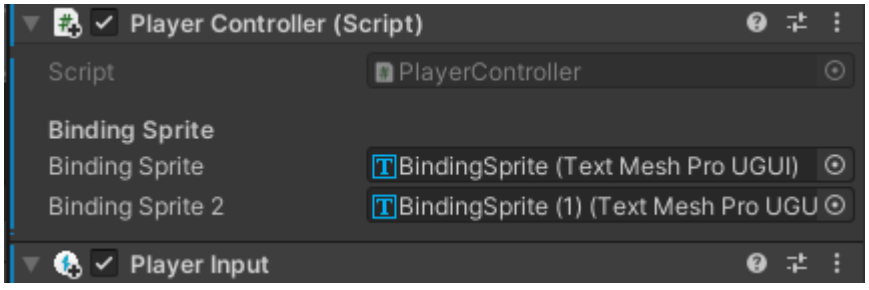
- Now your player gameobject add Player Input then add actions you've just created from scratch and change behaviour to Invoke Unity Events. Expand events dropdown and add playercontroller script and enable updatebindingsprite to the **ControlChangedEvent.**



- Add GameInput.cs class and attach playerInput to refer player input object.



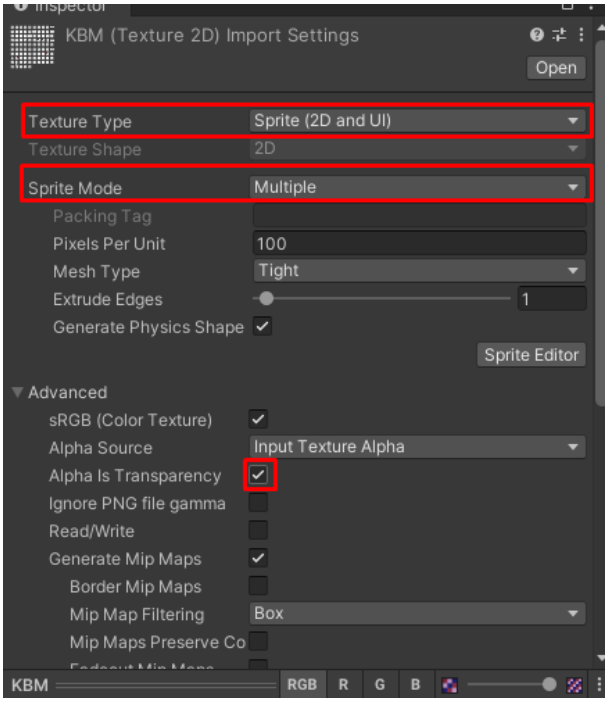
- Attach many textmeshpro text asset as your project prefers in order to show sprite asset in binding sprite.



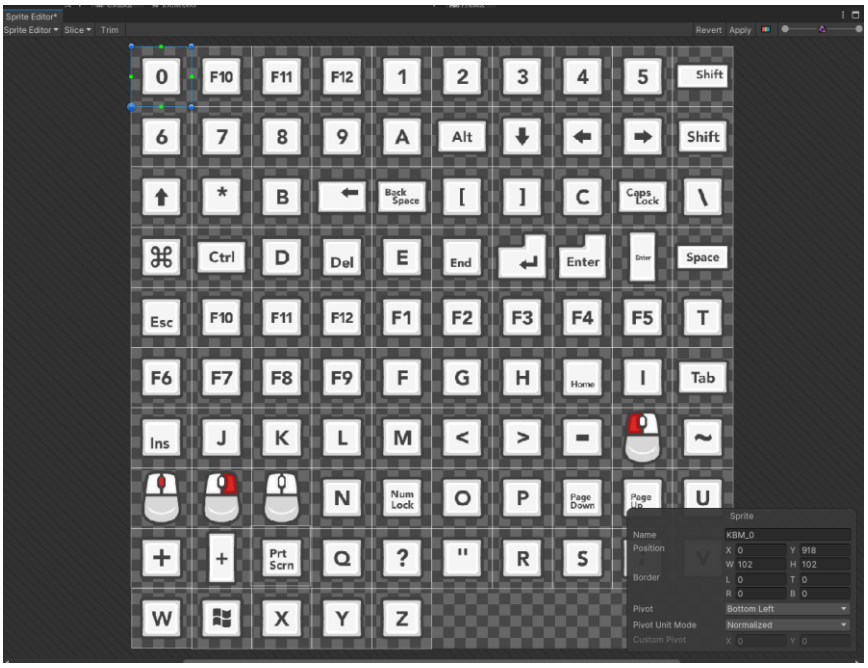
Make your own sprite assets:

You can make your own sprite assets and can be used as binding sprite. To do this

- Download any sprite sheet containing your device button layout.
- Change texture type to Sprite (2D and UI)
- Change Sprite mode to multiple
- Enable Alpha is Transparency to true



- Now open sprite editor and change pivot to centered and match the row and colums in which spirtes can be sliced perfectly.



- After slicing the spritesheet, then select the spritesheet and create textmeshpro sprite asset.
- Now you've to rename sprite sheet name as the input takes the name of sprite sheet on ActionToSpriteConverter class

```
1 reference
private static string RenameInput(string stringBtnName, string actionName)
{
    string inputDeviceName = GameInput.Instance.GetInputDeviceName();
    stringBtnName = stringBtnName.Replace(oldValue: $"[{inputDeviceName}]", newValue: string.Empty);
    stringBtnName = stringBtnName.Replace(oldValue: $"{actionName}:", newValue: string.Empty);
    stringBtnName = stringBtnName.Replace(oldValue: $"<Keyboard>/", newValue: $"Kb_");
    stringBtnName = stringBtnName.Replace(oldValue: $"<XInputController>/", newValue: $"Xbox_");
    stringBtnName = stringBtnName.Replace(oldValue: $"<DualShockGamepad>/", newValue: $"PS_");
    stringBtnName = stringBtnName.Replace(oldValue: $"[Xbox]", newValue: string.Empty);
    stringBtnName = stringBtnName.Replace(oldValue: $"[PS]", newValue: string.Empty);
    return stringBtnName;
}
```

- You can change the naming method on the RenameInput() method or change the name on the spritesheet. The naming should be for example:
    - Keyboard -> "Kb_a"
    - Xbox -> "Xbox_buttonSouth"