UNITY

Intermediate

New Input System

기존의 Input System

기존에 우리가 사용하던 입력방식은 다음과 같았습니다.

If (Input.GetKeyDown(KeyCode.A)) ...

또는 이렇게 사용하신 분들도 있을겁니다.

Input.GetAxis(“Horizontal”)

그런데 이 입력방식은 다중 기기에 대해 할당할 때 불편한 부분도 있고, 고급 사용자들을 위한 기능도 부족한 점이 있었습니다. 일관성을 맞추기 위해 필요없는 값도 계산되기도 했죠. 이런 여러 문제점들을 보완하기 위해 새로 등장한 입력 방법을 소개하겠습니다!

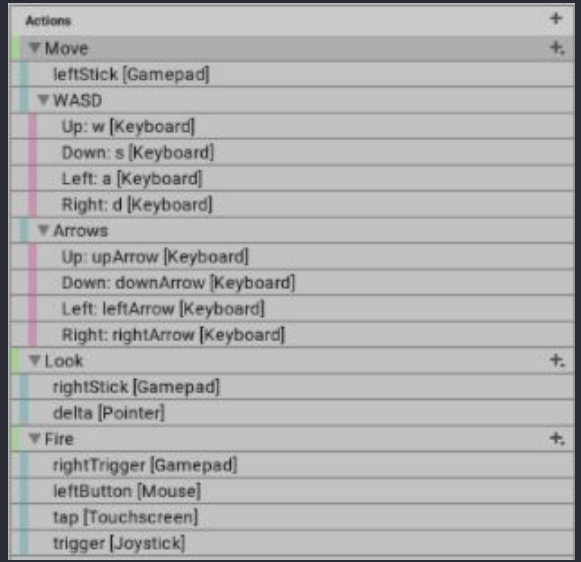
Input System 2.0

2.0이라 하긴 했지만 진짜 2.0라고 부르지는 않습니다. 대신 원래 있던 입력방식이 Input System (Old) 라고 되었죠.

Input System은 아까 말한 것과 같은 단점을 보완하기 위해 만들어졌으며, 사용자가 직접 입력받는 기기의 입력값을 조정해야 한다는 번거로움이 있지만 그만큼 기능이 강력해서 커스텀의 방식이 무궁무진합니다.

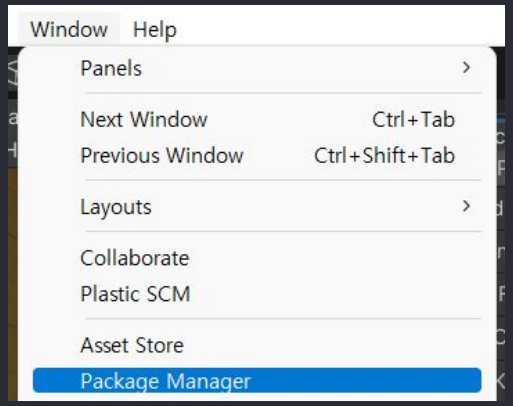
[키보드] 로부터 직접 입력받 수도 있고, 아니면 오른쪽 사진과 같이 Input Action이란 에셋을 만들어서 사용할 수도 있고 또는 코드상에서 Input Action을 동적으로 생성해줄 수도 있습니다.

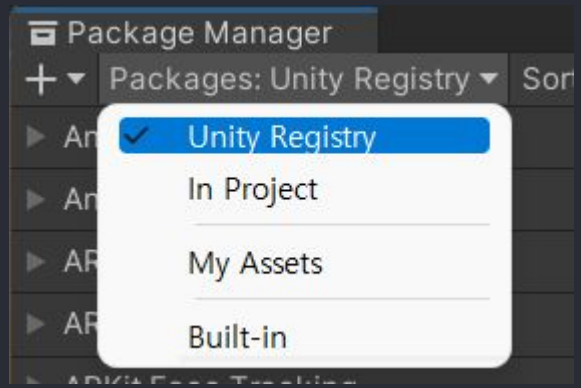
이 강의에서는 오른쪽의 Input Action처럼 UI를 사용하는 방식으로 진행하겠습니다.

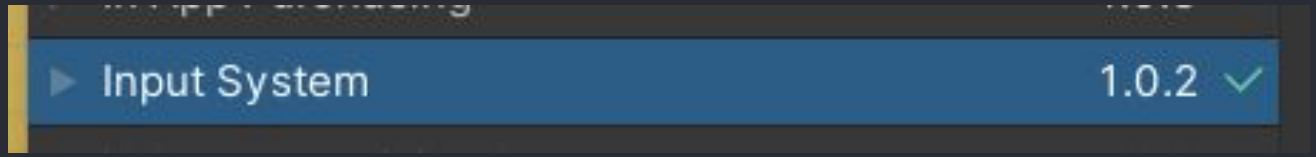


Installation

우선, Input System을 사용하기 위해서는 Package Manager에서 Input System을 설치해야합니다.

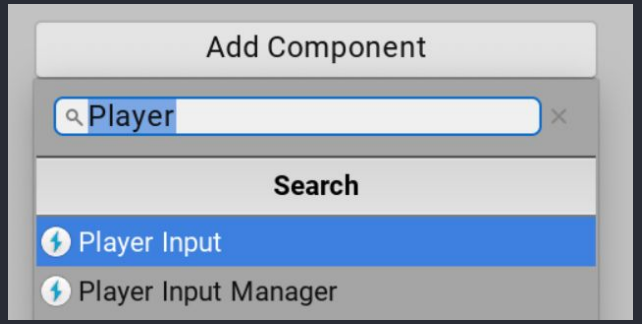


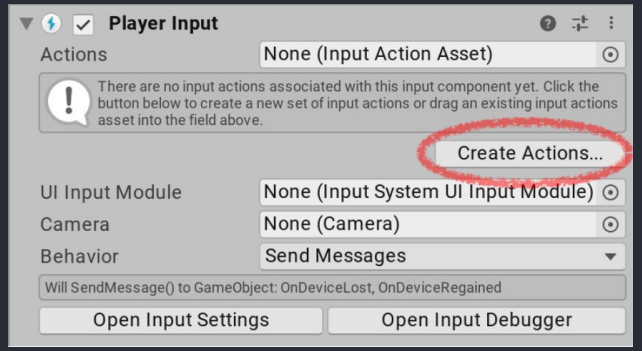




Player Input Component

설치가 끝났다면, 플레이어가 될 게임 오브젝트에 [Player Input] 컴포넌트를 추가해야 합니다. 그리고 Create Actions... 를 누르면 새 Input Action을 생성하기 위한 경로지정 창이 나옵니다. 원하는 장소에 저장해 주시고, 더블클릭으로 실행합니다.

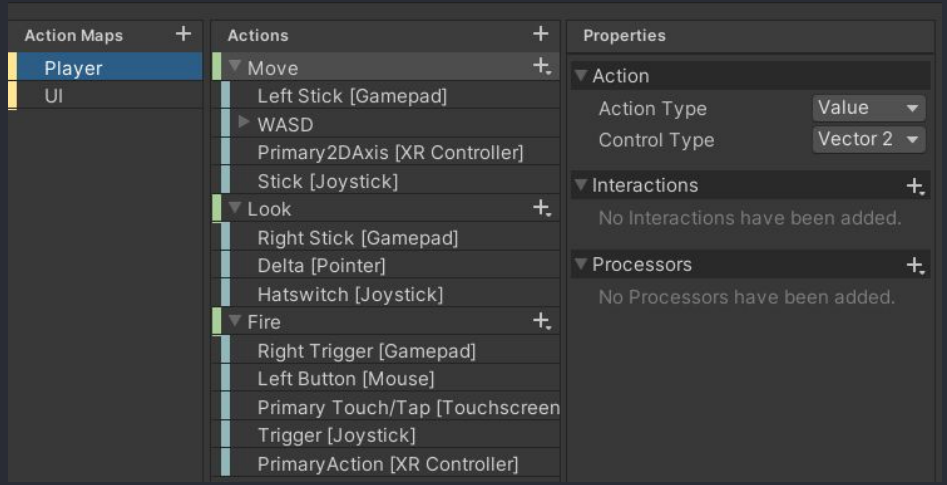




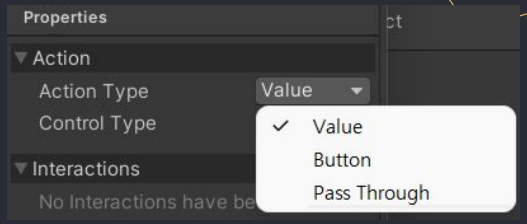
Input Actions

그럼 다음과 같은 화면이 나오는데, 여기서 Action Maps는 특정 상황에 맞춰서 입력방식을 변화시킬 때 사용되며 주로 게임조작과 UI조작으로 나뉩니다.

Actions는 실제 받을 입력값인데, 기본으로 있는 Action으로는 Move, Look, Fire가 있습니다. 오른쪽 위의 +버튼을 눌러 추가할 수 있으며, 오른쪽 Properties 창에서 Action Type을 변경할 수 있습니다. 또한, 각 Action의 안쪽을 보면 키보드 뿐만 아니라 게임패드, 조이스틱 등 다양한 입력을 하나로 처리하는 걸 볼 수 있는데 이는 생성된 Action의 오른쪽에 있는 +버튼을 눌러 추가할 수 있습니다.



Action Types



Action Type은 해당 입력이 동작하는 방식을 설정하며 3가지가 존재합니다.

- **Value**

연속적인 입력이 존재할 때 사용됩니다. 값이 변화될때마다 값을 전달하며,
액션에 여러 기기의 입력이 존재할 때 가장 많은 행동이 존재하는 기기가 선택되고,
그 기기의 입력만을 받습니다(driven). 그래서 가장 많은 행동을 하는 기기 하나에게서만 입력을 받으며 이를 Disambiguation 이라고 합니다.

- **Button**

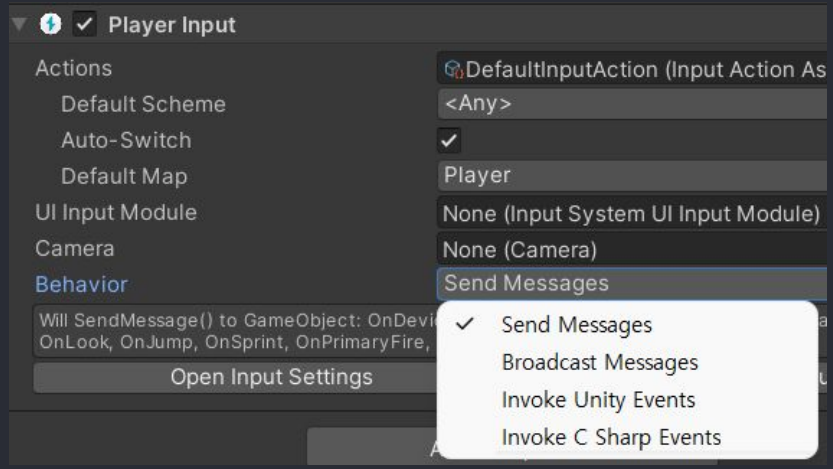
입력을 한 순간이 필요하다면 사용됩니다. 누른 순간에만 값을 전달합니다.

- **Pass Through**

Value와 똑같지만, Disambiguation을 하지 않습니다.

Behavior

Behavior는 Player Input 컴포넌트의 속성으로, 입력값을 어떻게 전달할 지에 대한 설정을 할 수 있습니다. 총 4가지 방법이 존재하는데 Send Messages와 Invoke Unity Events에 대해 알아보도록 하겠습니다.

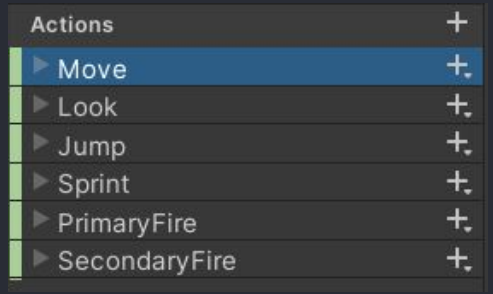


Send Messages

현재 Player Input이 적용된 GameObject의 모든 MonoBehavior 클래스들에게 특정 이름의 함수를 호출합니다. 여기서 함수 이름은 Input Action에 정의된 Action의 이름 앞에 On을 붙인 형태입니다.

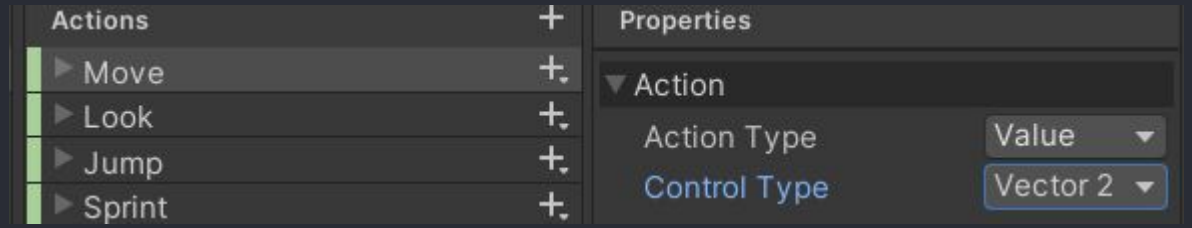
Will SendMessage() to GameObject: OnDeviceLost, OnDeviceRegained, OnControlsChanged, OnMove, OnLook, OnJump, OnSprint, OnPrimaryFire, OnSecondaryFire

```
public void OnMove(InputValue value)
{
    MoveInput(value.Get<Vector2>());
}

참조 0개
public void OnLook(InputValue value)
{
    if(cursorInputForLook)
    {
        LookInput(value.Get<Vector2>());
    }
}

참조 0개
public void OnJump(InputValue value)
{
    JumpInput(value.isPressed);
    Debug.Log("Jump " + value.isPressed);
}
```



Send Messages

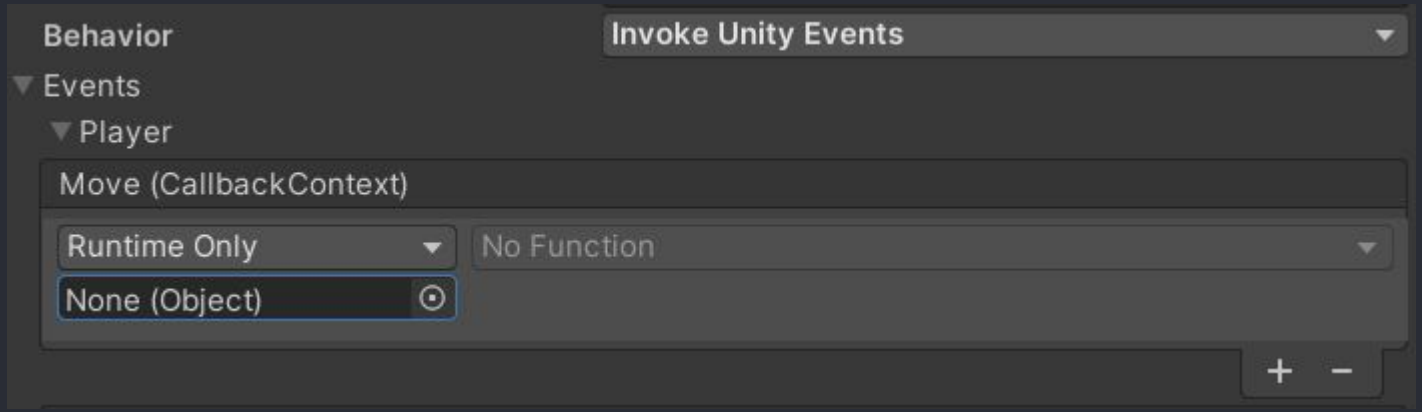
그리고 해당 함수는 InputValue 타입을 매개변수로 입력받는데, 여기에 Get<T>() 메서드를 사용하여서 원하는 타입의 입력값을 받을 수 있습니다. 여기서 T(타입) 에 들어가는 값은 Input Action에서 설정해준 타입과 동일해야합니다.




```
public void OnMove(InputValue value)
{
    MoveInput(value.Get<Vector2>());
}
```

Invoke Unity Events

이는 UI의 Button의 OnClick() 의 처리와 동일한데, 유니티 이벤트를 사용합니다.



안에 게임오브젝트를 넣으면, 해당 게임오브젝트가 가진 모든 컴포넌트에 접근할 수 있습니다. 이를 이용해서 입력값을 처리하기 위해 만들어준 메서드를 호출하게 하면 끝납니다. 잘 모르겠다면 유니티 UI의 버튼의 OnClick() 을 참고해보세요. 참고로, 이는 Unity Event를 다루는 방법과 동일합니다.

Input Handler

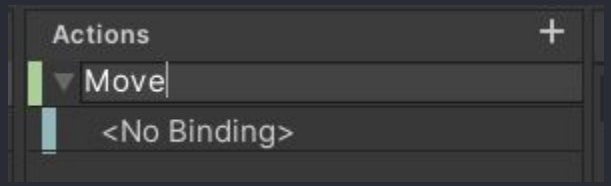
개인마다 입력을 처리하는 방법은 다르지만, 저와 그리고 대부분의 사람들이 InputHandler 라는 입력처리를 담당하는 클래스를 만들어서 처리합니다. InputHandler는 키마다 입력값을 받아와서 변수(필드)에 저장해 놓고, 입력값이 필요할 때는 다른 클래스에서 InputHandler 컴포넌트를 가져와서 필드에 접근합니다.

```
public class PlayerInputHandler : MonoBehaviour
{
    [Header("Character Input Values")]
    public Vector2 move;
    public Vector2 look;
    public bool jump;
    public bool sprint;
    public bool primaryFire;
    public bool secondaryFire;
```

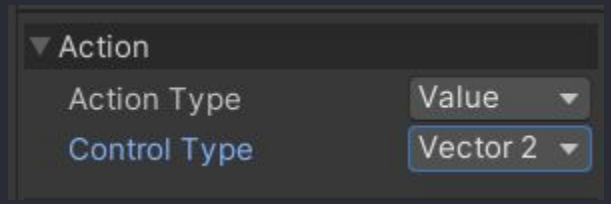
```
public class PlayerManager : MonoBehaviour
{
    PlayerInputHandler _input;
    Unity 메시지 | 참조 0개
    void Start()
    {
        _input = GetComponent<PlayerInputHandler>();
    }

    Unity 메시지 | 참조 0개
    void Update()
    {
        if (_input.primaryFire)
            return;              // 로직...
    }
}
```

실습하기

게임에 이동, 시야 조정, 점프를 넣으려고 합니다. 유니티로 직접 입력받는 방법을 알아봅시다.

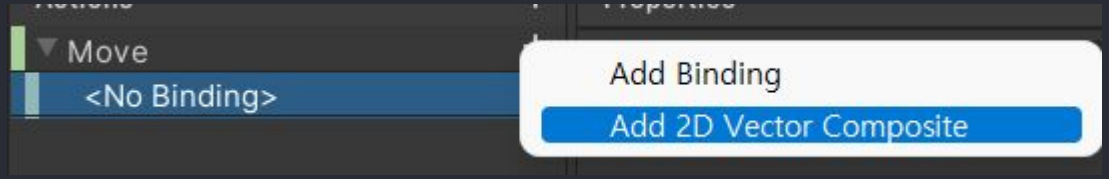


일단 Actions에 이미 있던것들을 지우고, +를 눌러 새로 추가한 뒤 Move라는 이름으로 바꿔줍니다.

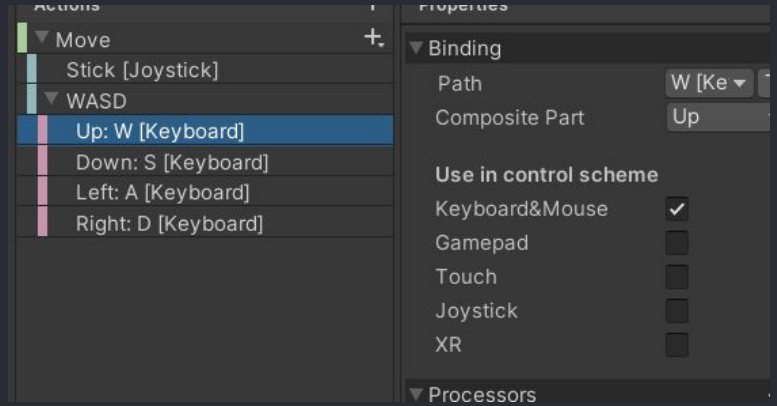


그 후에, Properties에서 위와 같이 설정을 바꿔줍시다. 이동은 왼쪽/오른쪽과 위/아래 를 가지고 있기 때문에 Vector2로 처리합니다. 왼쪽/오른쪽 구분은 -1, 1과 같이 해주면 되기 때문에 값이 2개만 있으면 충분합니다.

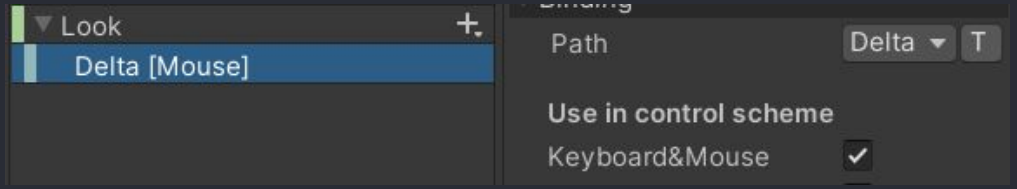
키보드 WASD 입력



Move 오른쪽의 +를 누르고, [Add 2D Vector Composite] 를 선택합니다. Add Binding의 경우, 입력값이 애초부터 Vector2 타입을 쓰는 것으로 게임패드의 스틱과 같은 입력이 해당됩니다. 하지만 우리는 WASD라는 서로 다른 키기 때문에 2D Vector Composite로 WASD를 벡터의 각 값에 대응할 것입니다.

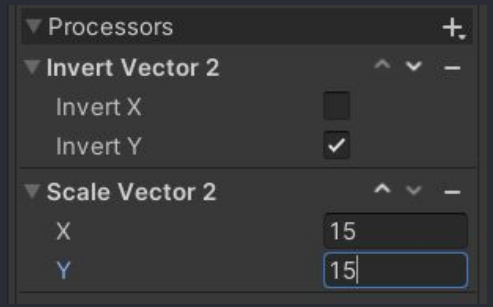


마우스 입력

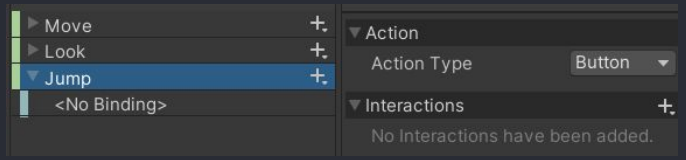


Look이라는 새 Action을 만들고 Move와 같이 Value, Vector2로 설정해주었습니다. 그리고 안에는 Delta [Mouse] 를 넣어주었습니다. 이는 Binding이며 자체적으로 Vector2의 값을 넘겨주는데 x로 마우스의 x 변화율, y로는 마우스의 y 변화율을 줍니다.

여기에 추가로 이후에 만들 코드를 위해 Properties에 있는 Processors에 다음과 같은 요소들을 추가해주겠습니다. Invert는 부호를 반전해 주는 기능이고 Scale는 크기에 곱해주는 역할을 합니다. 기본으로 들어오는 값은 1정도로 작기 때문에 키워서 사용하는 것이죠



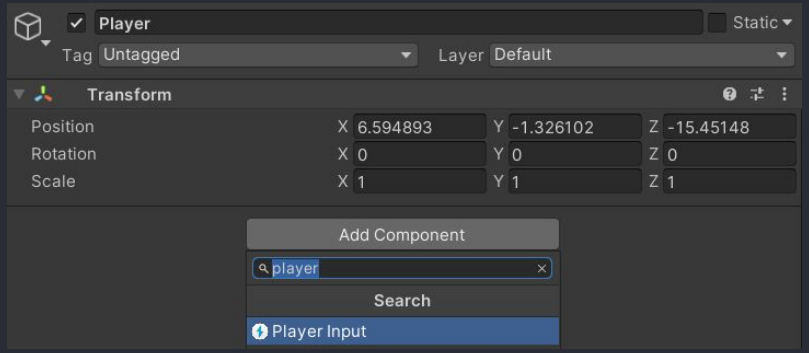
점프 입력



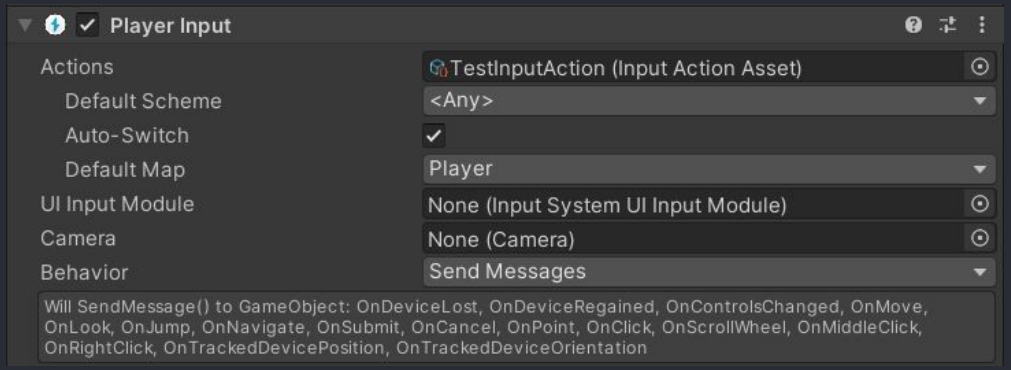
점프는 입력이 되었는지가 중요하기 때문에 Button으로 쓰겠습니다. Value를 이용하면 눌렸을 때 true, 뗐을 때 false가 되는 것을 더 쉽게 할 수 있지만 Button의 사용방법도 익힐 겸 Button을 사용하겠습니다.

전부 다 끝났다면 위의 Save Asset를 누르고 창을 닫습니다.

GameObject 생성

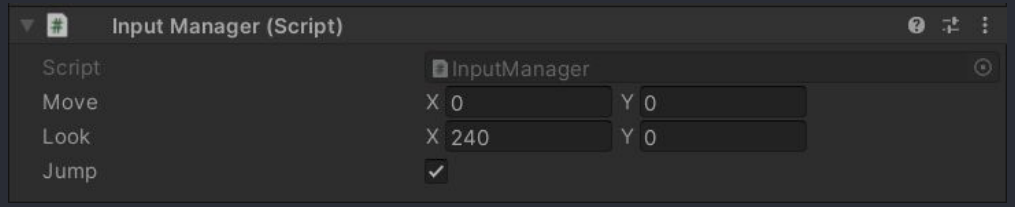


Create Empty로 새 게임 오브젝트를 만든 뒤, [Player Input] 컴포넌트를 추가합니다. 그리고 방금 전에 만든 Input Action을 “Actions” 속성 안에 넣어줍니다. 저는 Behavior로 Send Messages를 사용하겠습니다.



코딩 작업

다음과 같이 코드를 작성하고, 플레이어 게임오브젝트에 컴포넌트로 InputManager를 넣어줍니다. 그리고 시작하고 테스트해보면 WASD를 누를 때마다 Move의 값이 바뀌고, 마우스 이동값에 따라 Look의 값이 바뀐다는 것을 알 수 있습니다. 그리고 SpaceBar를 누르면 Jump가 true로 되는데, 다시 false로 바뀌지는 않는다는 것을 알 수 있습니다.




```
using UnityEngine.InputSystem;

Unity 스크립트 | 참조 0개
public class InputManager : MonoBehaviour
{
    public Vector2 move;
    public Vector2 look;
    public bool jump;
    참조 0개
    public void OnMove(InputValue val)
    {
        move = val.Get<Vector2>();
    }
    참조 0개
    public void OnLook(InputValue val)
    {
        look = val.Get<Vector2>();
    }
    참조 0개
    public void OnJump(InputValue val)
    {
        jump = val.isPressed;
    }
}
```

사용할 때는?

```
public class PlayerMove : MonoBehaviour
{
    InputManager _input;
    Unity 메시지 | 참조 0개
    void Start()
    {
        _input = GetComponent<InputManager>();
    }
```

위와 같이 플레이어 이동을 관리하는 클래스를 작성하고, InputManager를 GetComponent<>로 가져오게 됩니다.

```
        _fallTimeoutDelta -= Time.deltaTime;
    }

    // if we are not grounded, do not jump
    _input.jump = false;
}
```

그리고 InputManager의 jump값은 임의로 false로 지정해줘야 한다는 것을 잊지 마세요.

더 알아보기

Input System | Input System | 1.0.2 (unity3d.com)

공식 위키 문서를 참고하면 코드 내에서 Input Action 을 생성하거나 입력기기로부터 바로 입력을 받는 등의 다양한 기능도 배울 수 있습니다.