New Input System
(Chapter 3)

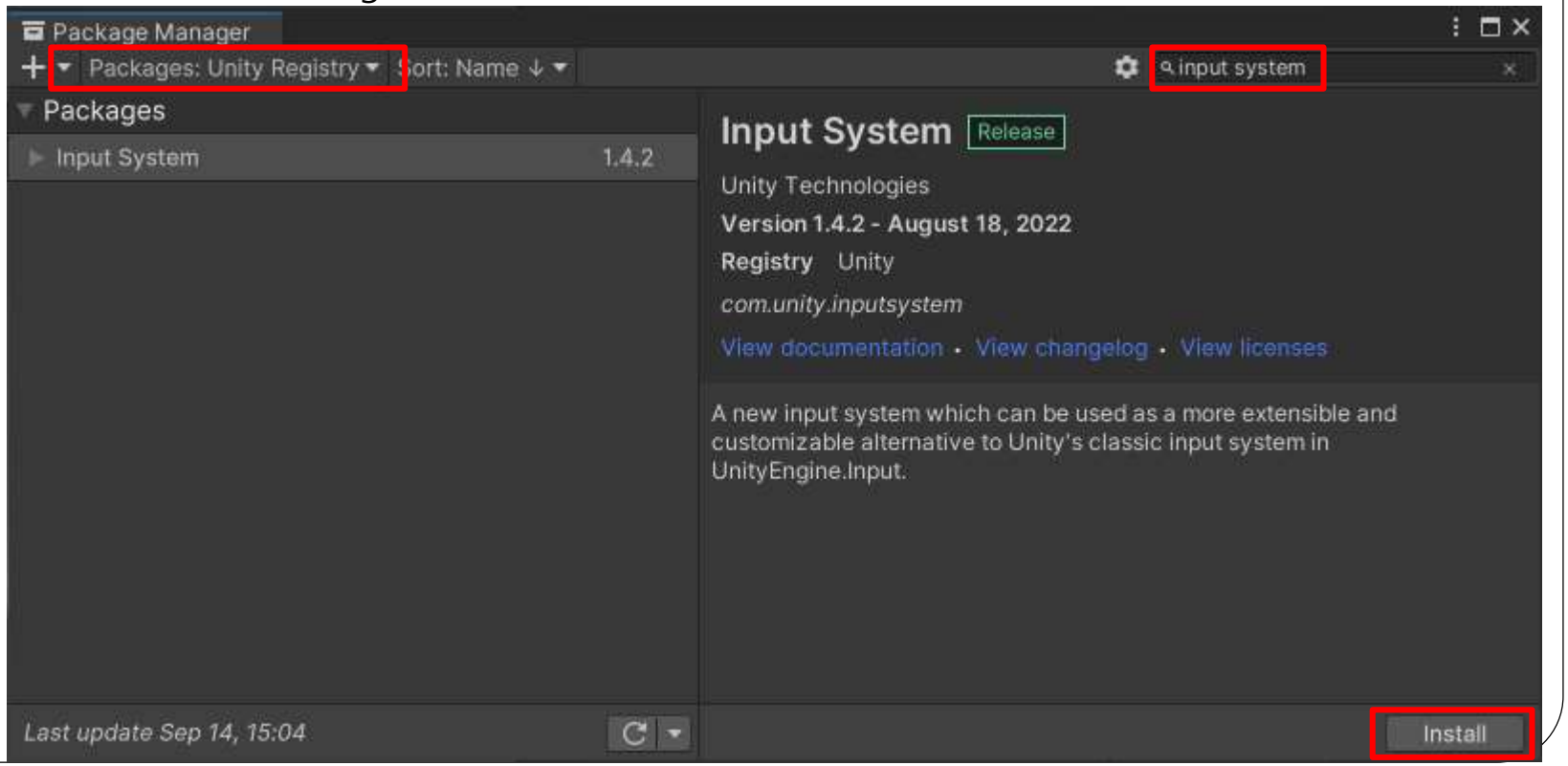
Jin-Mo Kim

jinmo.kim@hansung.ac.kr

그냥 신기술 소개이고,
참고로만 보면 될듯

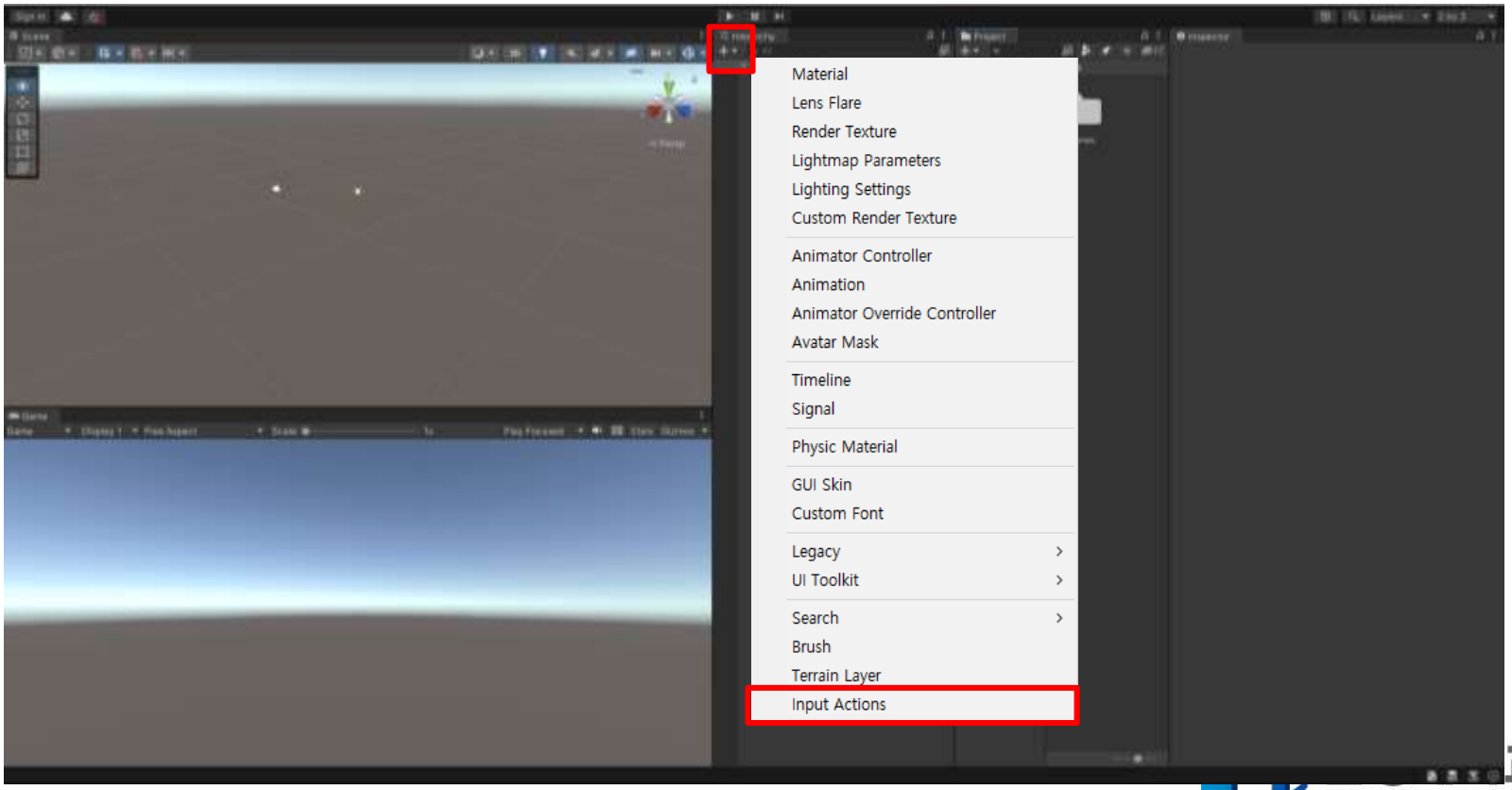
1. Package Manager

- Input System 설치
  - Window → Package Manager
    - Packages: Unity Registry 변경
    - Input System 검색 및 설치
      - 경고(Warning) 창 활성화 시 Yes 선택



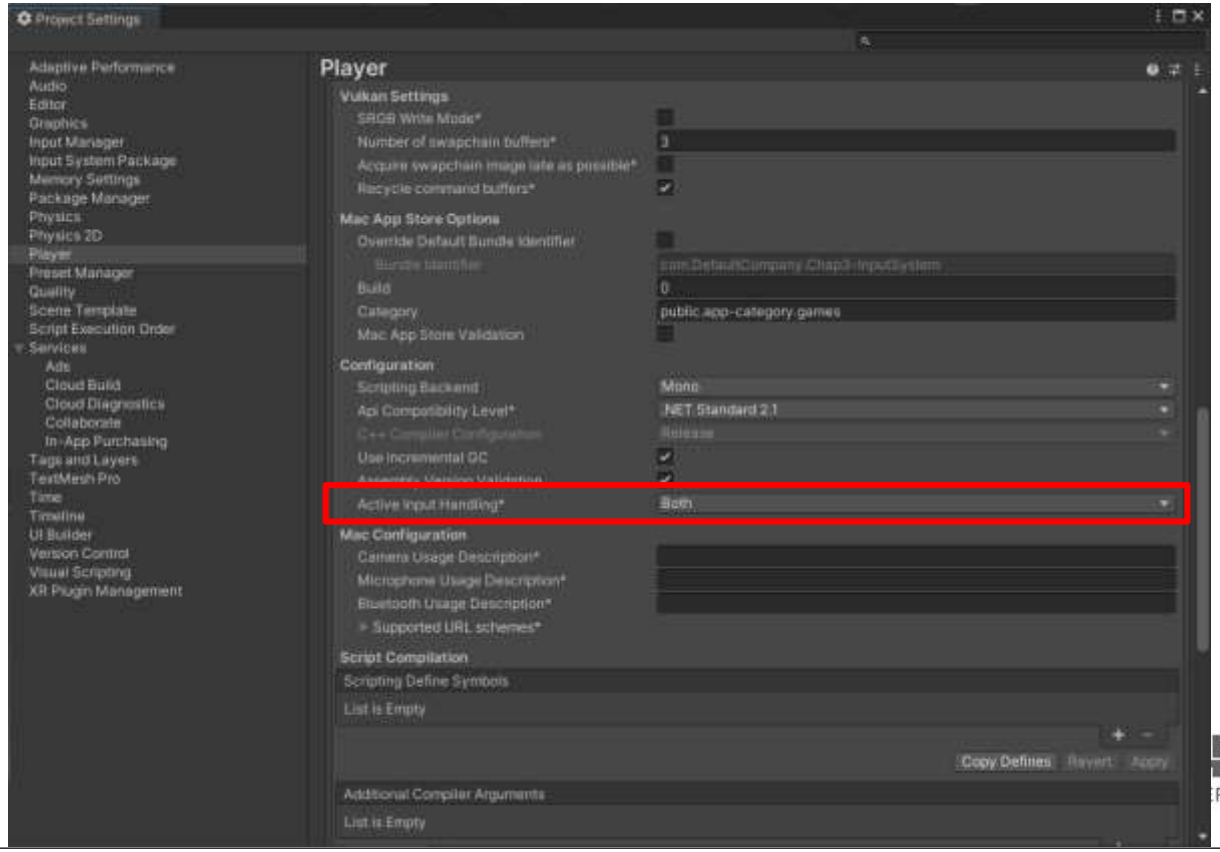
1. Package Manager

- Input System 설치
  - 유니티 재실행 후 활성화
    - Project → Input Actions 항목 확인



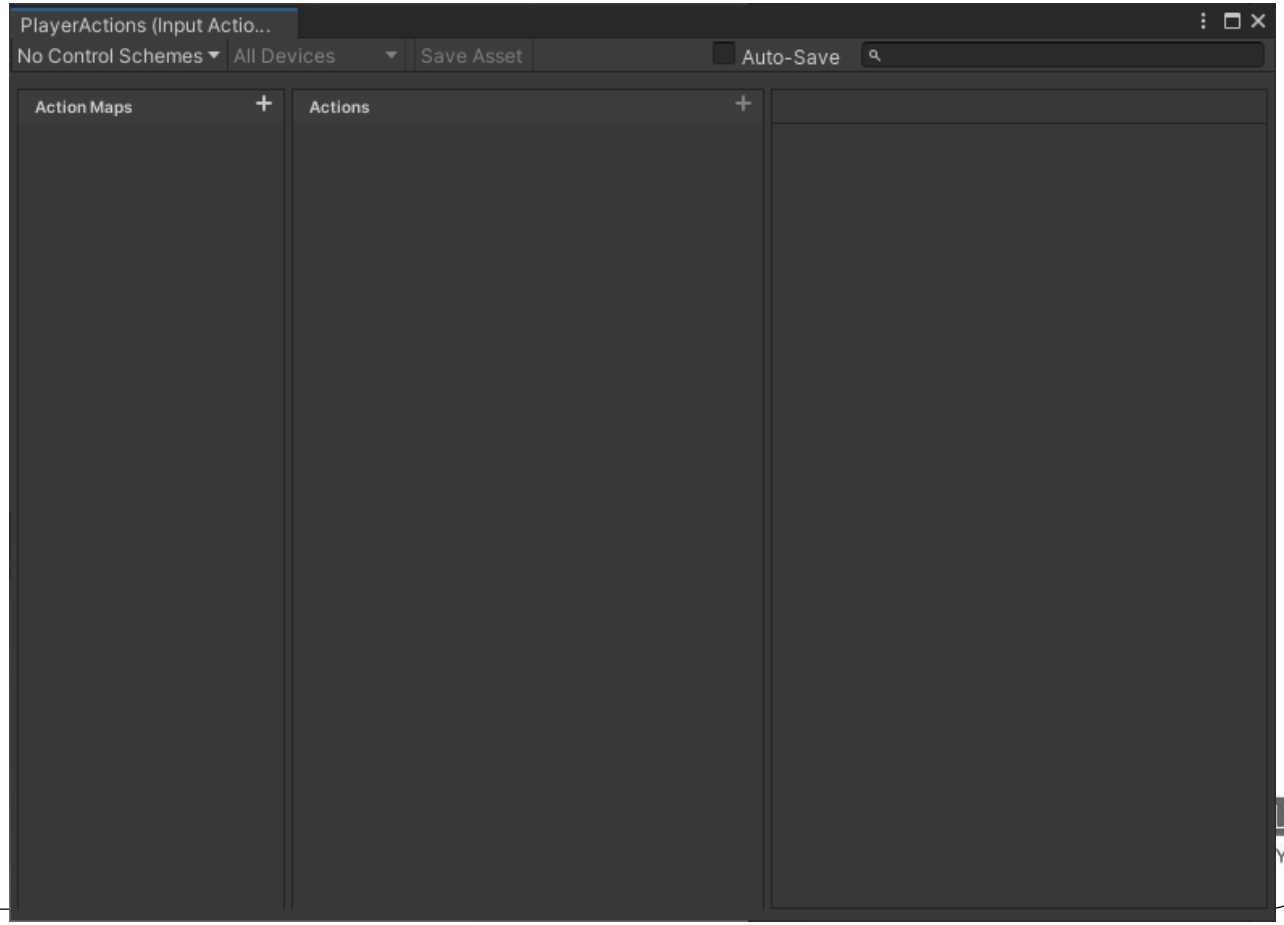
2. Project Settings

- 설정 확인
  - Edit → Project Settings → Player 선택
    - Other Settings → Active Input Handling*
      - Input System Package (New) / Input Manager (Old) / Both



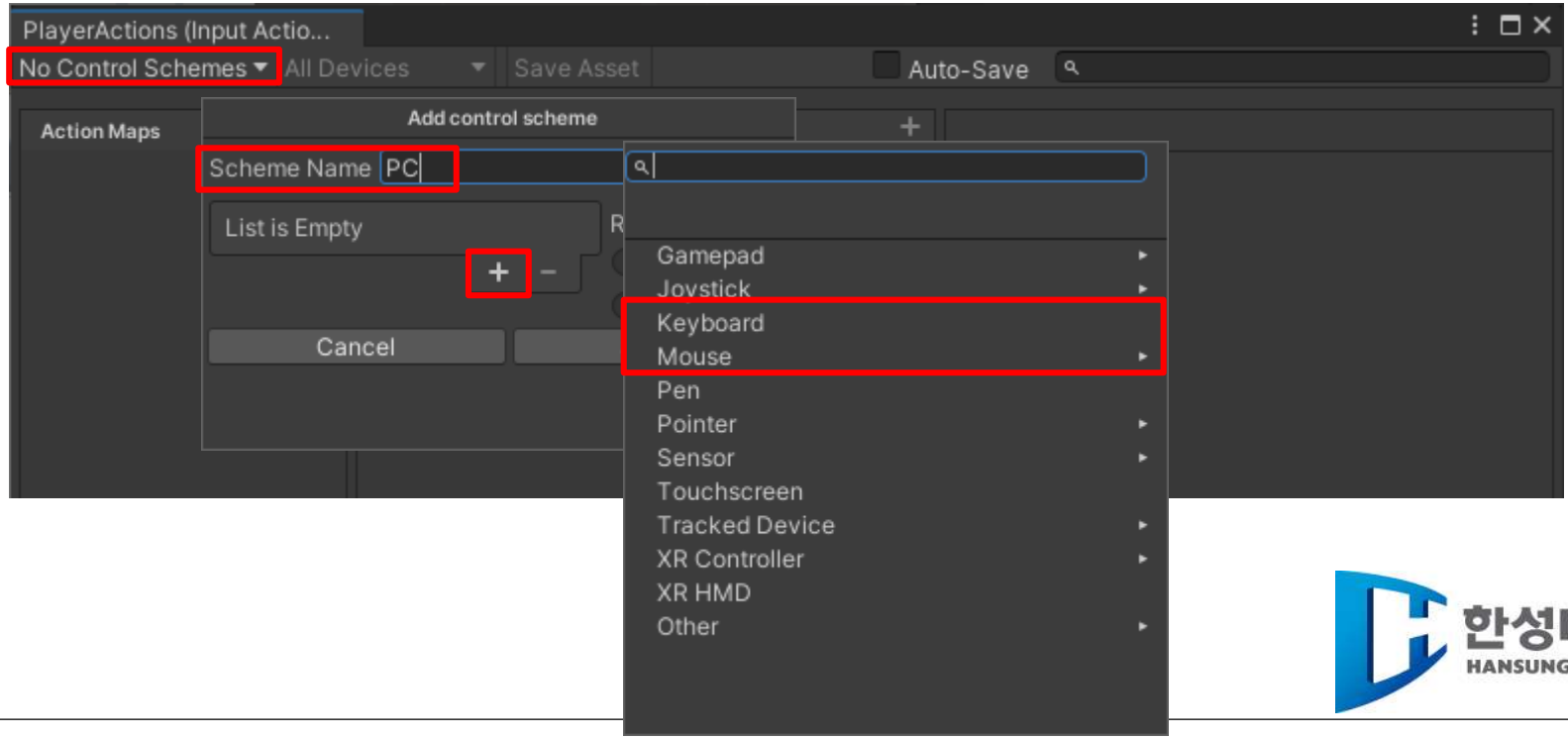
3. Input Actions

- Input Actions 설정
  - Project → Input Actions
    - 이름: PlayerActions



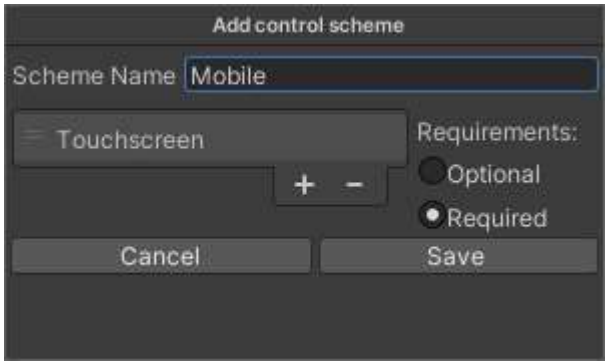
3. Input Actions

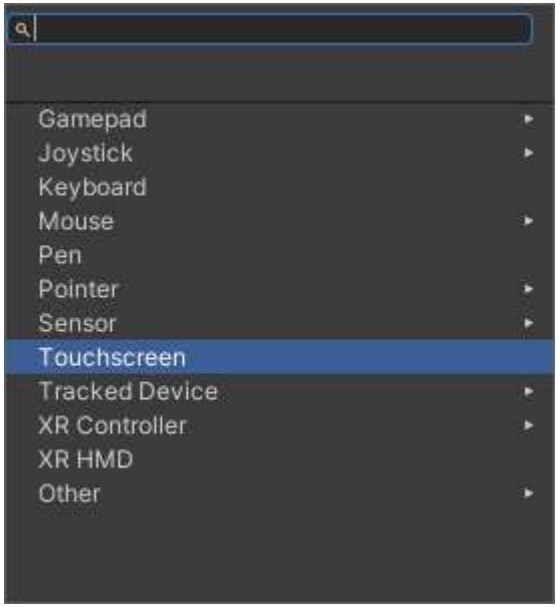
- Input Actions 설정
  - Control Schema 설정
    - 특정 플랫폼(PC, 모바일 등)에서 특정 내용들을 사용할지에 대한 설정
    - Add Control Scheme..
      - Scheme Name: PC
      - List → Keyboard, Mouse 추가



3. Input Actions

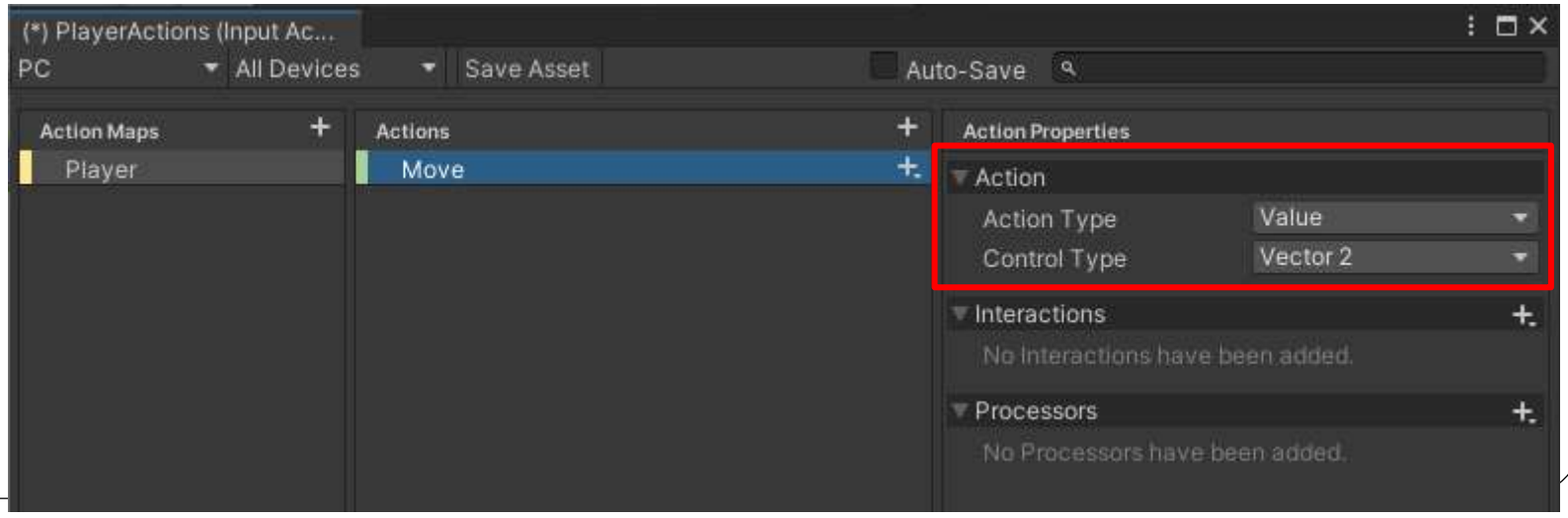
- Input Actions 설정
  - Control Schema 설정
    - Add Control Scheme..
      - Scheme Name: Mobile
      - List → Touchscreen 추가





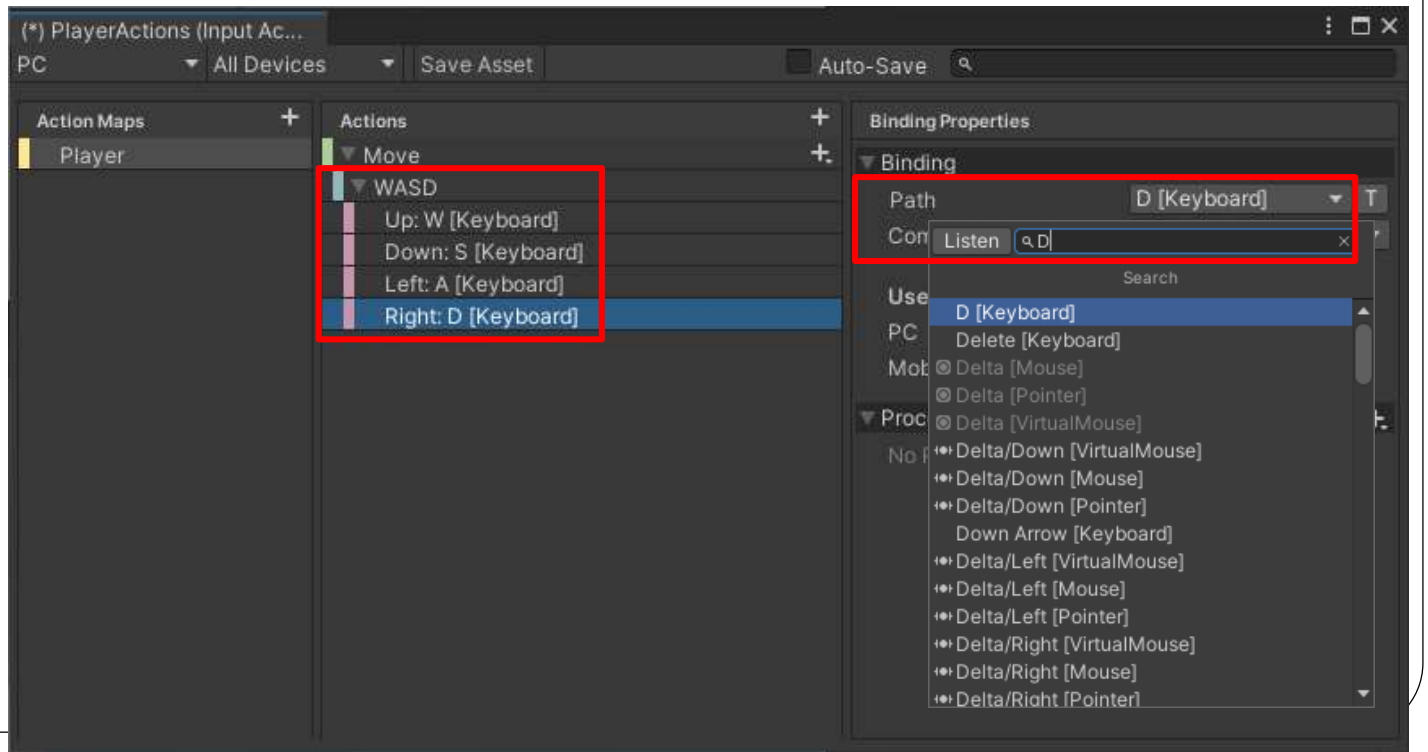
4. Action

- Action 설정
  - Action Maps
    - 캐릭터, 객체 등에 적용할 Action Map을 설정
      - Player 생성
  - Actions
    - 캐릭터, 객체의 구체적인 행동을 정의
      - Move로 이름 수정
      - 하위의 No Binding은 삭제
        - Action Type: Value
        - Control Type: Vector2 → 2차원 벡터 값으로 입력 설정



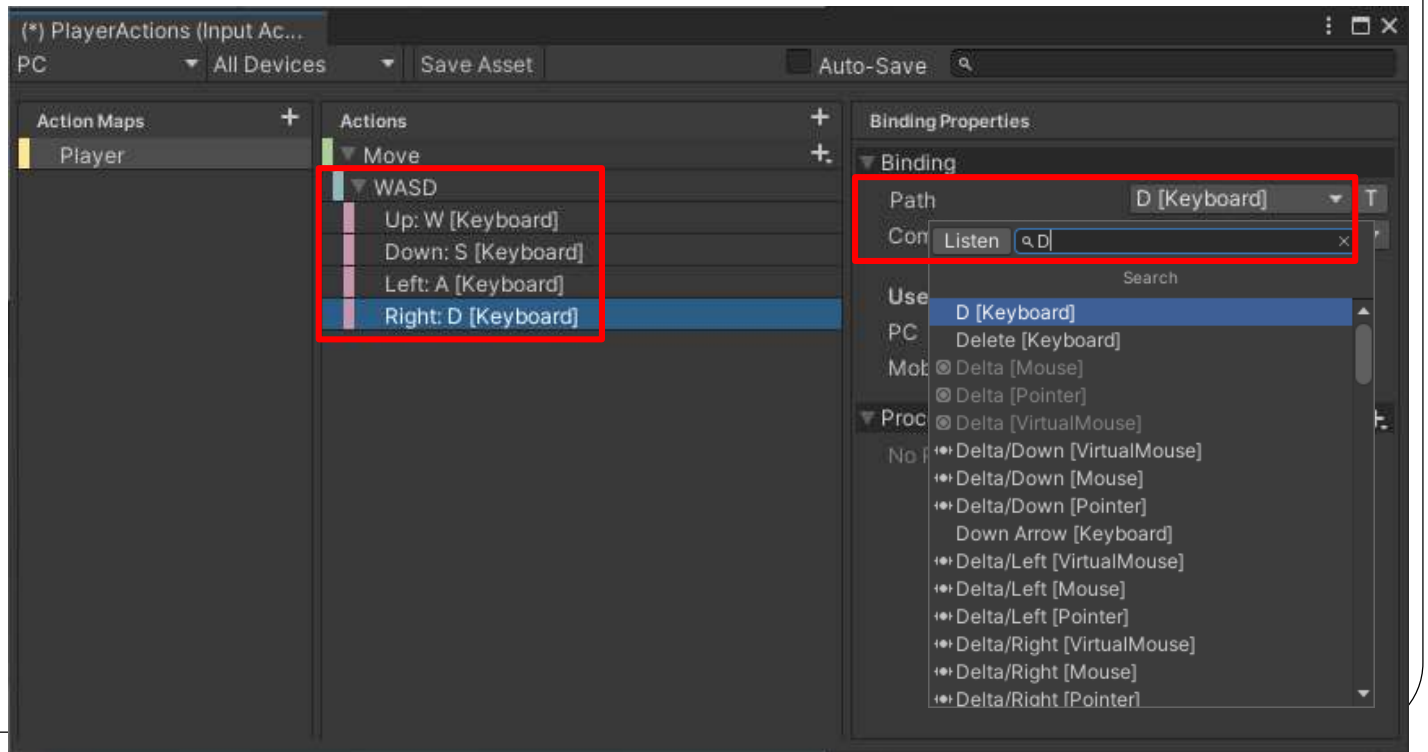
4. Action

- Action 설정
  - Actions
    - Add Up₩Down₩Left₩Right Composite 추가
      - WASD 또는 방향키를 사용하여 이동을 구현
      - 이름: WASD
        - Path에 적절한 키를 찾아 바인딩



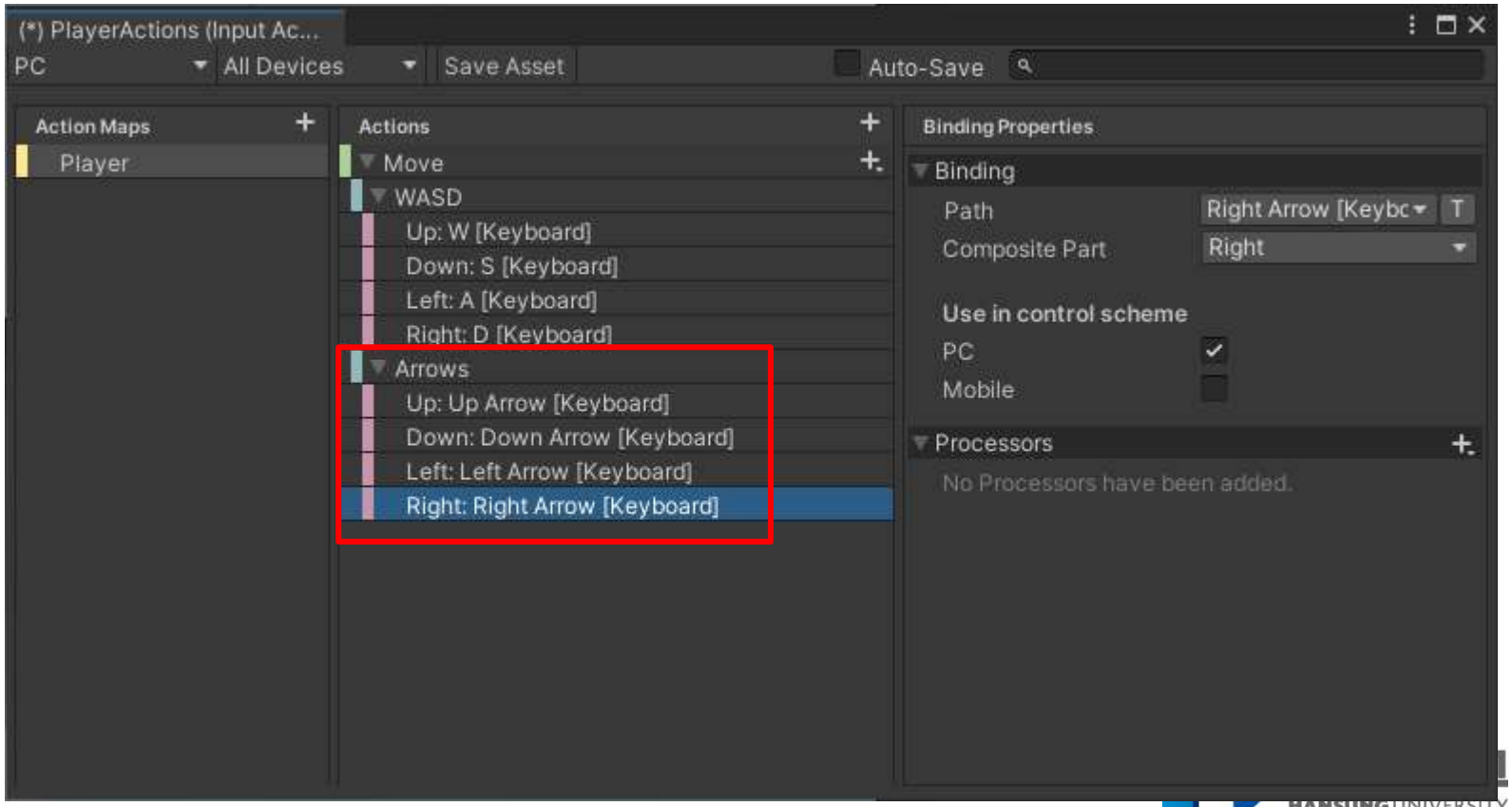
4. Action

- Action 설정
  - Actions
    - Add Up₩Down₩Left₩Right Composite 추가
      - WASD 또는 방향키를 사용하여 이동을 구현
      - 이름: WASD
        - Path에 적절한 키를 찾아 바인딩



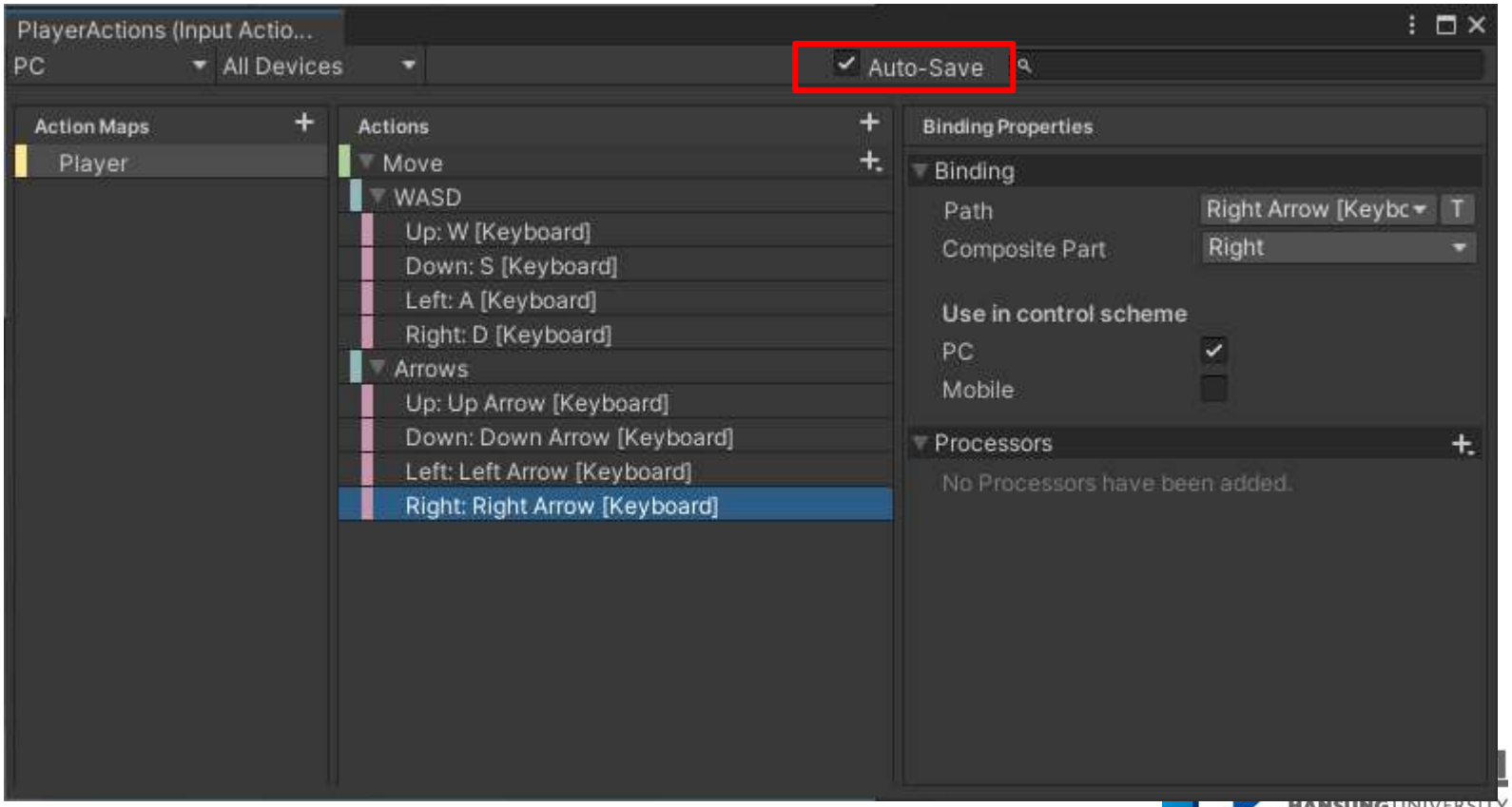
4. Action

- Action 설정
  - Actions
    - 같은 방법으로 방향키 Action (Arrows) 을 추가



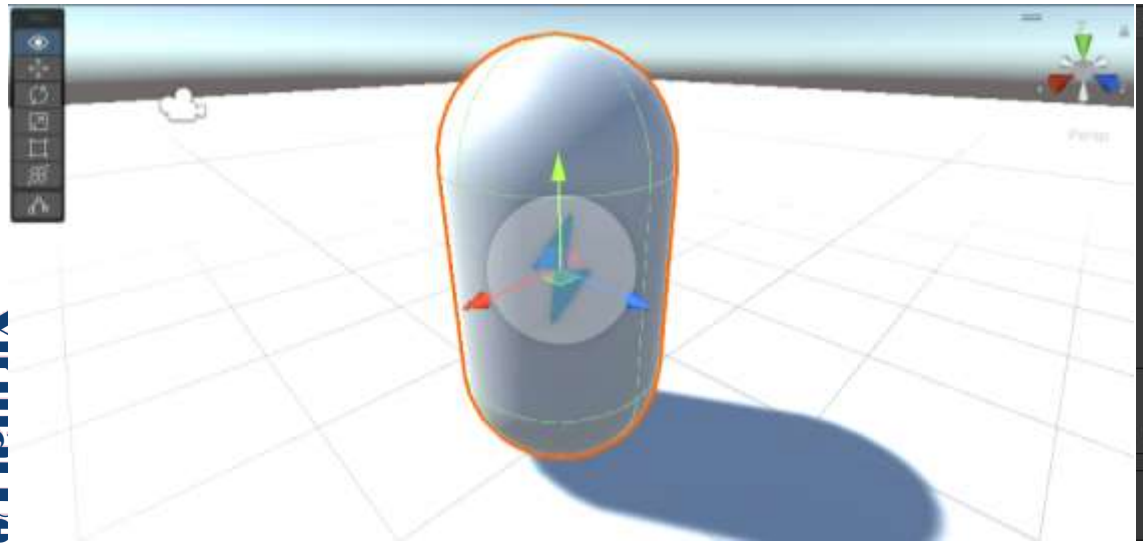
4. Action

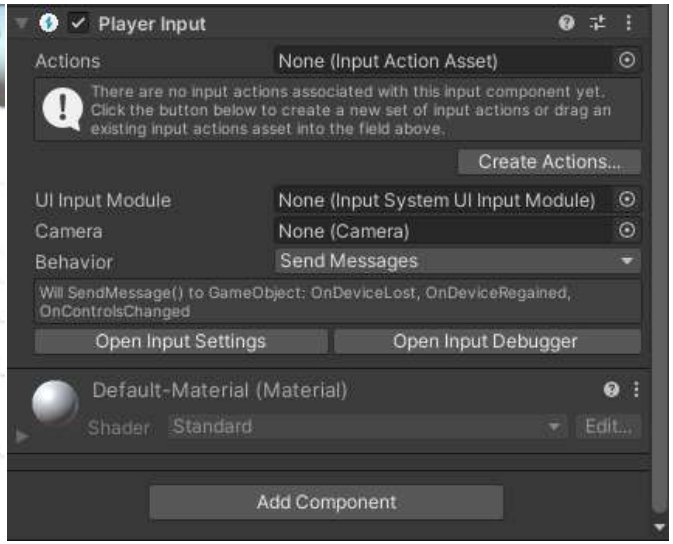
- Action 설정
  - 저장 설정
    - 우상단 Auto-Save 체크



5. Player Input Component

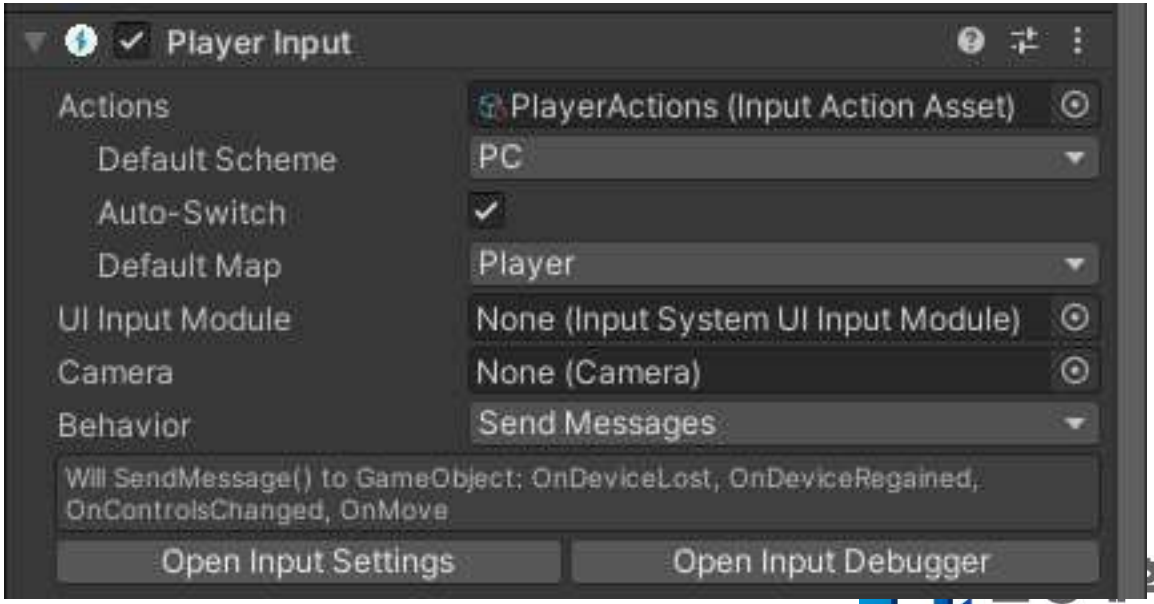
- Player Input Component 설정
  - 캐릭터 선택 → Add Component
    - Player Input 컴포넌트 추가
      - Player Input 컴포넌트 관련 속성 추가





5. Player Input Component

- Player Input Component 설정
  - Player Input
    - Actions: PlayerActions 선택
    - Default Scheme: PC
      - Any: 자동으로 설정
    - Default Map: Player



5. Player Input Component

- Player Input Component 설정
  - Player Input
    - Behavior: Send Messages
      - Send Messages, Broadcast Massages
        - 유니티의 Send Message 기능을 사용하여 함수를 호출
      - Invoke Unity Events, Invoke C Sharp Events
        - 유니티 또는 C#의 이벤트 처리 기능을 사용하여 구현

6. 스크립트 제어

- 스크립트 생성
  - Send Messages 기능을 사용하여 이벤트 처리
    - Broadcast Message: 하위 계층의 객체들까지 제어 가능
    - 함수 명은 "On + Actions name"으로 구성

```
using UnityEngine.InputSystem;

public class cshPlayer : MonoBehaviour
{
    private Vector3 moveDir;
    private float moveSpeed = 5.0f;

    void Update()
    {
        if(moveDir != Vector3.zero)
        {
            transform.rotation = Quaternion.LookRotation(moveDir);
            transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
        }
    }

    void OnMove(InputValue value) {
        Vector2 input = value.Get<Vector2>();
        if(input != null)
        {
            moveDir = new Vector3(input.x, 0.0f, input.y);
            moveDir.Normalize();
        }
    }
}
```

6. 스크립트 제어

- 스크립트 생성
  - Invoke Unity Events 기능을 사용하여 이벤트 처리
    - 유니티 이벤트 호출 방식을 사용
    - 스크립트를 Invoke Unity Events 방식으로 수정
      - 함수의 접근지정자는 public
        - 에디터에서 접근할 수 있음

```
public void OnMove(InputAction.CallbackContext context)
{
    Vector2 input = context.ReadValue<Vector2>();
    if (input != null)
    {
        moveDir = new Vector3(input.x, 0.0f, input.y);
        moveDir.Normalize();
    }
}
```

6. 스크립트 제어

- 스크립트 생성
  - Invoke Unity Events 기능을 사용하여 이벤트 처리
    - Player Input 컴포넌트 수정
      - Behavior: Invoke Unity Events 수정
      - Events → Player
        - Move CallbackContext 추가