유니티 프로그래밍 강의

이준

3D 닷지 플레이어 제작

닷지 플레이어 제작

- PlayerController 스크립트를 완성했지만 몇 가지 문제가 있음.

1. **조작이 게임에 즉시 반영되지 않습니다.**
   리지드바디 컴포넌트의 AddForce() 메서드는 힘을 추가합니다. 누적된 힘으로 속도를 점진적으로 증가시키기 때문에 속도가 충분히 빨라질 때까지 시간이 걸립니다. 또한 이동 중에 반대 방향으로 이동하려는 경우 관성에 의해 힘이 상쇄되어 방향 전환이 금방 이루어지지 않습니다.
2. **입력 감지 코드가 복잡합니다.**
   방향키를 감지하는 데 if 문을 네 개 사용했습니다. 이것을 좀 더 쉽고 간결한 코드로 개선하고 싶습니다.
3. **playerRigidbody에 컴포넌트를 드래그&드롭으로 할당하는 것이 불편합니다.**
   6.6.5절 'PlayerController 컴포넌트 설정하기'에서 인스펙터 창에서 PlayerController 컴포넌트의 Player Rigidbody 필드로 리지드바디 컴포넌트를 직접 드래그&드롭했습니다. 변수에 컴포넌트를 직접 드래그&드롭하는 방식은 불편하며, 잘못된 값을 할당할 위험이 있습니다. 따라서 변수에 컴포넌트의 참조를 할당하는 과정을 코드로 실행하고 싶습니다.

닷지 플레C

- Start( ) 메서드 수정
  - Start( ) 메서드를 이용하여 게임이 시작될 때 변수 playerRigidbody에 리지드바디 컴포넌트의 참조를 할당하도록 수정하겠음.

GetComponent < 컴포넌트타입> () ;

▶ Player Rigidbody 필드가 사라짐

【과정 01】 리지드바디 컴포넌트를 코드에서 할당하기

① 변수 playerRigidbody와 Start( ) 메서드 부분을 다음과 같이 수정

```
// using 문 생략

public class PlayerController : MonoBehaviour {
    private Rigidbody playerRigidbody; // 이동에 사용할 리지드바디 컴포넌트
    public float speed = 8f; // 이동 속력

    void Start() {
        // 게임 오브젝트에서 Rigidbody 컴포넌트를 찾아 playerRigidbody에 할당
        playerRigidbody = GetComponent<Rigidbody>();
    }

    void Update() {
        // Update() 메서드 내용 생략
    }

    public void Die() {
        // 자신의 게임 오브젝트를 비활성화
        gameObject.SetActive(false);
    }
}
```

닷지 플레이어 제작

- 조작감 개선하기
  - 기존 Update( ) 메서드를 수정하여 코드를 더 간결하게 만들고 조작이 이동 속도에 즉시 반영되도록 개선함.

[과정 01] PlayerController의 기존 Update ( ) 메서드 개선

① PlayerController 스크립트의 Update ( ) 메서드를 다음과 같이 수정

```
void Update() {
    // 수평축과 수직축의 입력값을 감지하여 저장
    float xInput = Input.GetAxis("Horizontal");
    float zInput = Input.GetAxis("Vertical");

    // 실제 이동 속도를 입력값과 이동 속력을 사용해 결정
    float xSpeed = xInput * speed;
    float zSpeed = zInput * speed;

    // Vector3 속도를 (xSpeed, 0, zSpeed)로 생성
    Vector3 newVelocity = new Vector3(xSpeed, 0f, zSpeed);
    // 리지드바디의 속도에 newVelocity 할당
    playerRigidbody.velocity = newVelocity;
}
```

- 수평축과 수직축의 입력값을 감지
- 속도를 나타낼 새로운 Vector3를 생성
- 리지드바디 컴포넌트의 속도를 변경

닷지 플레이어 제작

- GetAxis( ) 메서드
  - Input.GetAxis( ) 메서드는 어떤 축에 대한 입력값을 숫자로 반환하는 메서드임.

```
float Input.GetAxis(string axisName);
```

  - Input.GetAxis( ) 메서드는 축Axis의 이름을 받음. 그리고 다음 경우에 따라 감지된 입력값을 반환함.

- 축의 음의 방향에 대응되는 버튼을 누름 : -1.0
- 아무것도 누르지 않음 : 0
- 축의 양의 방향에 대응되는 버튼을 누름 : +1.0

닷지 플레이어 제작

Horizontal 축의 경우

- Horizontal(수평) 축에 대응되는 키
  - 음의 방향 : ←(왼쪽 방향키), A 키
  - 양의 방향 : →(오른쪽 방향키), D 키
- Input.GetAxis("Horizontal")의 출력값
  - ← 또는 A 키를 누름 : -1.0
  - 아무것도 누르지 않음 : 0
  - → 또는 D 키를 누름 : +1.0

Vertical 축의 경우

- Vertical(수직) 축에 대응되는 키
  - 음의 방향 : ↓(아래쪽 방향키), S 키
  - 양의 방향 : ↑(위쪽 방향키), W 키
- Input.GetAxis("Vertical")의 출력값
  - ↓ 또는 S 키를 누름 : -1.0
  - 아무것도 누르지 않음 : 0
  - ↑ 또는 W 키를 누름 : +1.0

- ← 또는 A 키 : xInput = -1.0
- 아무것도 누르지 않음 : xInput = 0
- → 또는 D 키 : xInput = +1.0
- ↓ 또는 S 키 : zInput = -1.0
- 아무것도 누르지 않음 : zInput = 0
- ↑ 또는 W 키 : zInput = +1.0

닷지 플레이어 제작

- 속도 계산하기
  - xInput 값과 zInput 값을 기반으로 X와 Z 방향의 속도를 각각 계산하고 새로운 Vector3 데이터를 생성함.

```
float xInput = Input.GetAxis("Horizontal");
float zInput = Input.GetAxis("Vertical");

float xSpeed = xInput * speed;
float zSpeed = zInput * speed;
```

닷지 플레이어 제작

- ← 또는 A 키를 누른 경우
  - xInput = -1.0
  - xSpeed = (-1.0) x speed = -speed (왼쪽 이동)
- 아무것도 누르지 않은 경우
  - xInput = 0
  - xSpeed = 0 x speed = 0 (정지)
- → 또는 D 키를 누른 경우
  - xInput = +1.0
  - xSpeed = (+1.0) x speed = speed (오른쪽 이동)

- ↓ 또는 S 키를 누른 경우
  - zInput = -1.0
  - zSpeed = (-1.0) x speed = -speed (뒤쪽 이동)
- 아무것도 누르지 않은 경우
  - zInput = 0
  - zSpeed = 0 x speed = 0 (정지)
- ↑ 또는 D 키를 누른 경우
  - zInput = +1.0
  - zSpeed = (+1.0) x speed = speed (앞쪽 이동)

닷지 플레이어 제작

- Vector3는 원소 x, y, z를 가지는 타입입니다. 위치, 크기, 속도, 방향 등을 나타낼 때 주로 사용함.
- x, y, z에 대응하는 값을 넣으면 됨.

```
Vector3 vector = new Vector3(x, y, z);
```

```
Vector3 vector = new Vector3(100f, 100f, 100f);
```

닷지 플

- 완성된 스크립트 확인(2차)
  - 지금까지 완성된 PlayerController 스크립트는 다음과 같음.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController : MonoBehaviour {
    private Rigidbody playerRigidbody; // 이동에 사용할 리지드바디 컴포넌트
    public float speed = 8f; // 이동 속력

    void Start() {
        // 게임 오브젝트에서 Rigidbody 컴포넌트를 찾아 playerRigidbody에 할당
        playerRigidbody = GetComponent<Rigidbody>();
    }

    void Update() {
        // 수평축과 수직축의 입력값을 감지하여 저장
        float xInput = Input.GetAxis("Horizontal");
        float zInput = Input.GetAxis("Vertical");

        // 실제 이동 속도를 입력값과 이동 속력을 사용해 결정
        float xSpeed = xInput * speed;
        float zSpeed = zInput * speed;

        // Vector3 속도를 (xSpeed, 0, zSpeed)로 생성
        Vector3 newVelocity = new Vector3(xSpeed, 0f, zSpeed);
        // 리지드바디의 속도에 newVelocity 할당
        playerRigidbody.velocity = newVelocity;
    }

    public void Die() {
        // 자신의 게임 오브젝트를 비활성화
        gameObject.SetActive(false);
    }
}
```

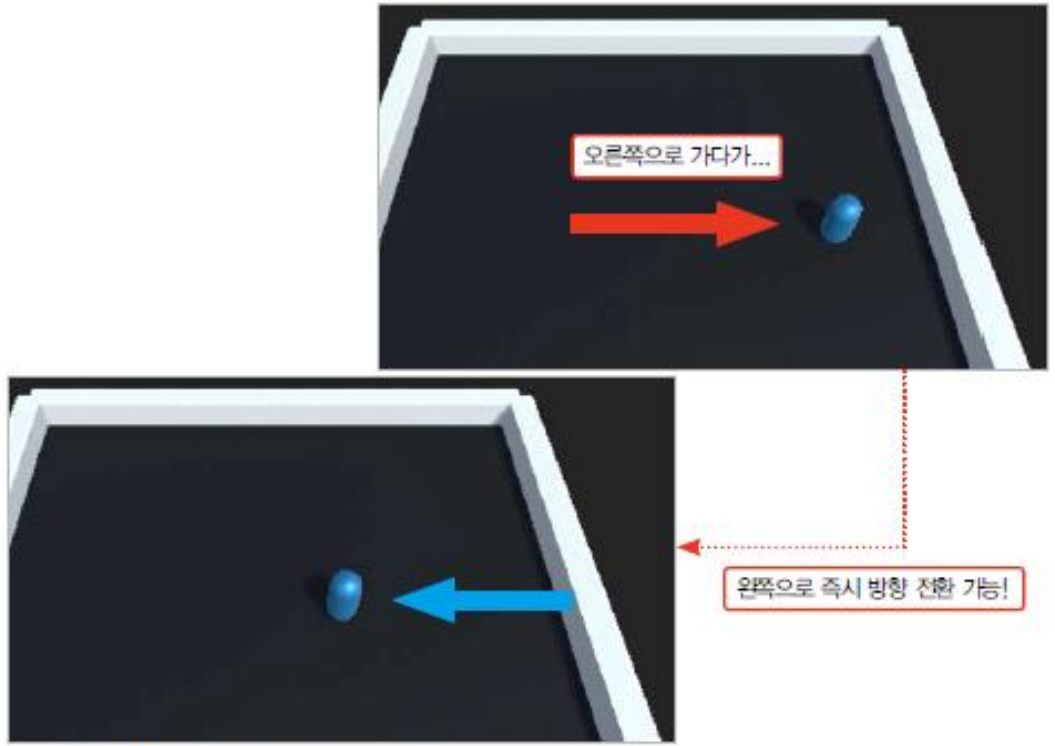
닷지 플레이어 제작

[과정 01] 테스트하기

① 스크립트를 저장하고 유니티 에디터로 돌아가기

② **플레이 버튼**을 눌러 씬 테스트

테스트를 해보면 이번에는 관성을 무시하고 즉시 플레이어의 이동 방향을 전환할 수 있습니다. 조작이 전반적으로 이전보다 더 빠르게 반영된다고 느껴질 겁니다.



▶ 관성을 무시하고 즉시 방향 전환 가능

닷지 플레이어 제작

- GetAxis( ) 메서드와 입력축
  - 처음에는 다음과 같이 Input.GetKey( )를 사용하여 키보드의 특정 키 입력을 직접 검사했음.

```
if (Input.GetKey(KeyCode.UpArrow) == true) {
    playerRigidbody.AddForce(0f, 0f, speed);
}
```

  - 특정 키를 지목하는 방식은 조작 키를 실시간으로 변경할 수 없는 단점이 있음

```
Input.GetKey(KeyCode.UpArrow) → Input.GetKey(KeyCode.W)
```

  - '입력 이름'을 거쳐 가는 방식을 사용함.

닷지 플레이어 제작

- 입력 이름
  - 유니티에서 사용하는 축이 바로 위에서 설명한 입력 이름임.
  - GetAxis( ) 메서드에서 사용한 Horizontal과 Vertical도 축임.
  - 축은 축에 대응하는 버튼을 가짐. 따라서 축을 사용하면 사용할 입력키를 직접 명시하지 않아도 됨.
  - 축은 유니티의 입력 매니저에서 관리함

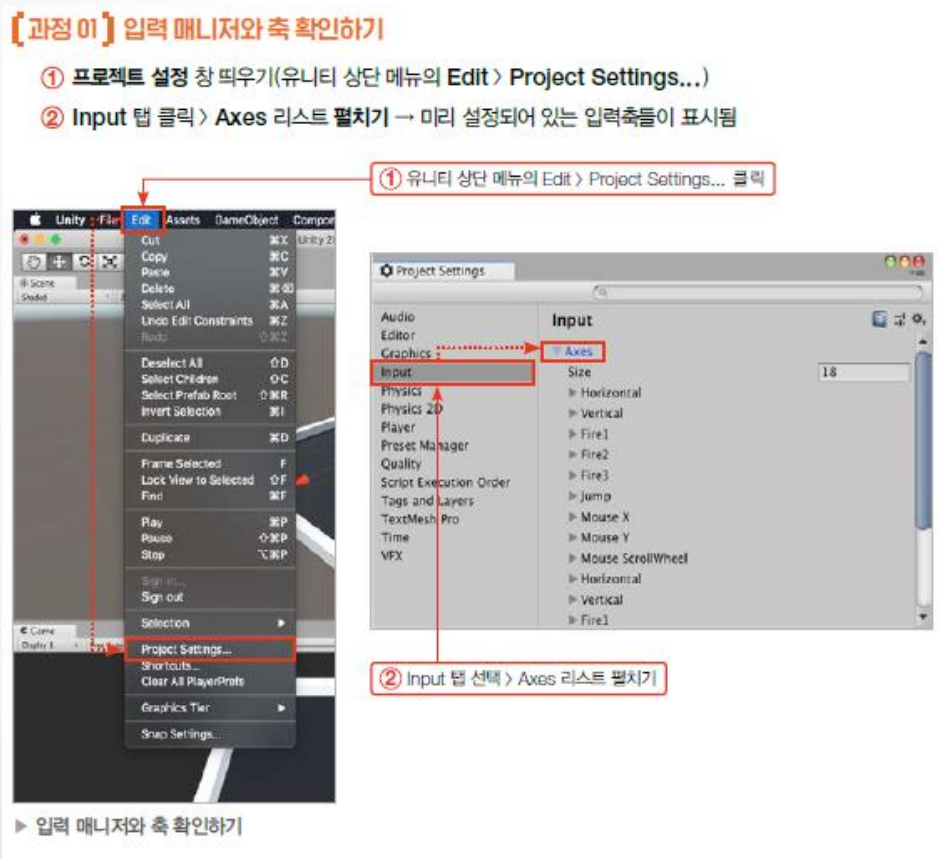
```
if (마우스 왼쪽 버튼을 누름) {
    // 총 발사
}
```

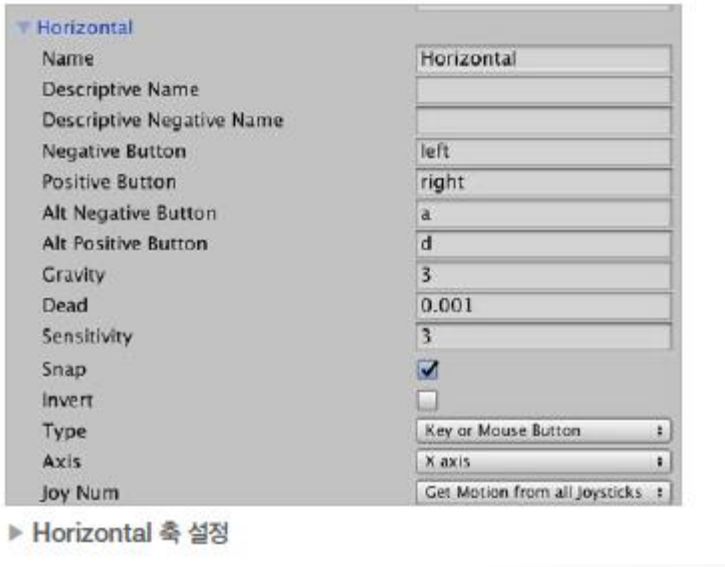
```
if ("발사"에 대응되는 버튼을 누름) {
    // 총 발사
}
```

코드 (실제 발사 기능) ⇔ 입력 이름 ("발사") ⇔ 입력 장치 (마우스 왼쪽 버튼)

닷지 플레이어 제작

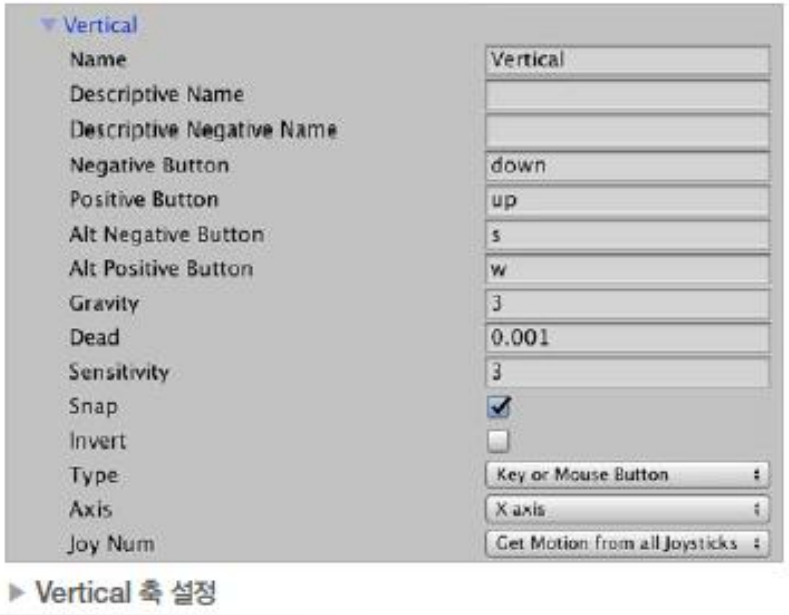
- 입력 매니저 설정 창
  - 입력 매니저에서 미리 설정된 축들을 확인함.





- 음의 방향 버튼(Negative Button) : `left`(왼쪽 방향키)
- 양의 방향 버튼(Positive Button) : `right`(오른쪽 방향키)
- 음의 방향 보조 버튼(Alt Negative Button) : `a`
- 양의 방향 보조 버튼(Alt Positive Button) : `d`

닷지 플레이어 제작



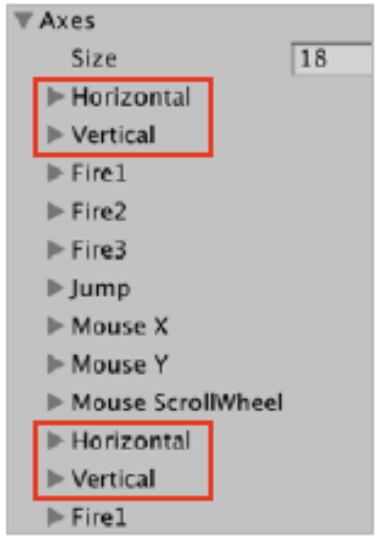
- **음의 방향 버튼(Negative Button)** : down(아래쪽 방향키)
- **양의 방향 버튼(Positive Button)** : up(위쪽 방향키)
- **음의 방향 보조 버튼(Alt Negative Button)** : s
- **양의 방향 보조 버튼(Alt Positive Button)** : w

[Input.GetAxis("Horizontal")이 실행될 때 다음 과정으로 입력값이 감지됨]

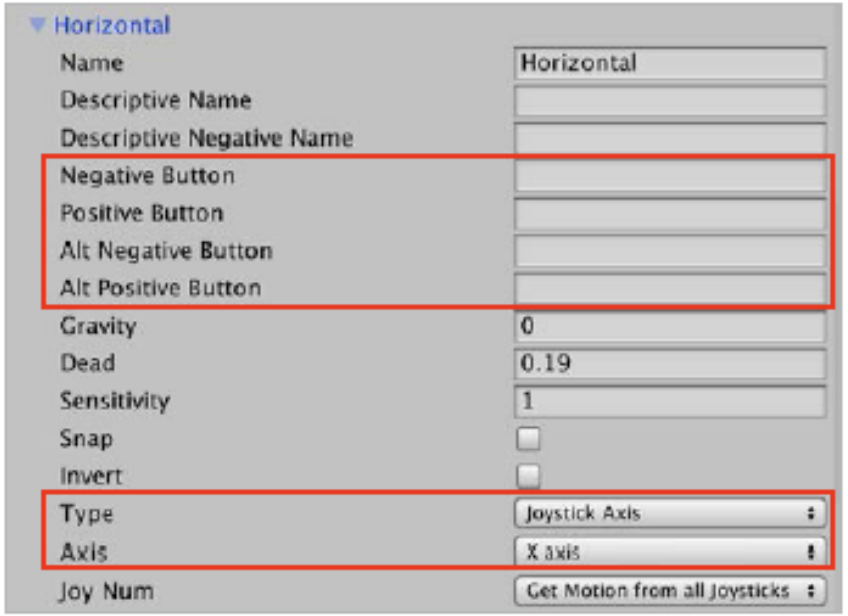
1. 입력 매니저에서 Horizontal 축을 찾음
2. Horizontal 축에 대응되는 버튼(←, a ,→ ,d)들로 현재 입력을 검사 › 감지된 입력값 반환

닷지 플레이어 제작

- 멀티플랫폼 입력 지원
  - Horizontal 축과 Vertical 축이 각각 하나씩 더 있음.
  - 두 번째 Horizontal 축과 Vertical 축은 엑스박스와 같은 콘솔 게임기 게임 패드의 조이스틱(아날로그 스틱)에 대응됨.



▶ Horizontal과 Vertical 축이 하나씩 더 있다

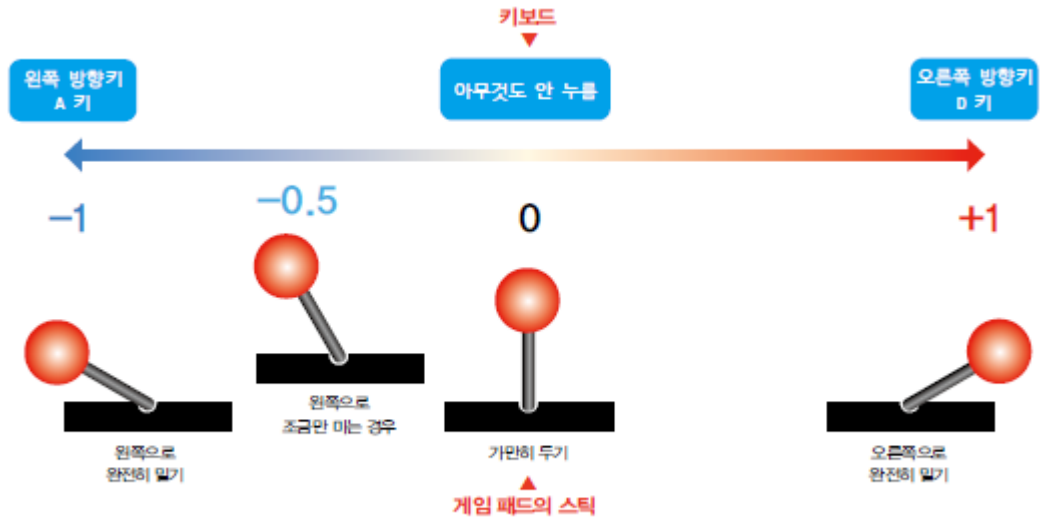


▶ 두 번째 Horizontal 축의 설정

닷지 플레이어 제작

- 입력을 숫자로 받는 이유
  - 입력축은 아날로그 스틱에도 대응된다는 사실에서 입력값을 true나 false가 아닌 숫자로 반환하는 이유를 알 수 있음.

▶ 게임 패드는 아날로그 스틱을 살짝 미는 것이 가능



▶ 키보드와 게임 패드에서 Horizontal 동작의 차이

키보드를 사용할 때는 다음과 같이 입력값이 감지됐습니다.

- 왼쪽 방향키 : -1.0
- 아무것도 누르지 않으면 : 0
- 오른쪽 방향키 : +1.0

마찬가지로 게임 패드의 스틱을 다음 방향으로 '완전히' 밀면 다음과 같이 입력값이 감지됩니다.

- 스틱을 왼쪽으로 완전히 밀기 : -1.0
- 스틱을 가만히 내버려 두기 : 0
- 스틱을 오른쪽으로 완전히 밀기 : +1.0