게임엔진프로그래밍응용

12. 네트워크 협동 게임 1

청강문화산업대학교 게임콘텐츠스쿨

반 경 진

공지사항

공지사항

- 2023학년도 1학기 방역 및 학사운영 방안
  https://www.ck.ac.kr/archives/193175
- 2023학년도 1학기 국가공휴일 및 대학 행사 수업 대체 일정 공지
  https://www.ck.ac.kr/archives/193109

온라인 수업 저작권 유의 사항

온라인 수업 저작권 유의 사항

온라인수업 저작권 유의사항 안내

강의 저작물을 다운로드, 캡처하여
교외로 유출하는 행위는
불 법 입 니 다

저작권자의 허락 없이 저작물을 복제, 공중송신 또는 배포하는 것은
저작권 침해에 해당하며 저작권법에 처벌받을 수 있습니다.

강의 동영상과 자료 일체는 교수 및 학교의 저작물로서 저작권이 보호됩니다.
수업자료를 무단 복제 또는 배포, 전송 시 민형사상 책임을 질 수 있습니다.

Index

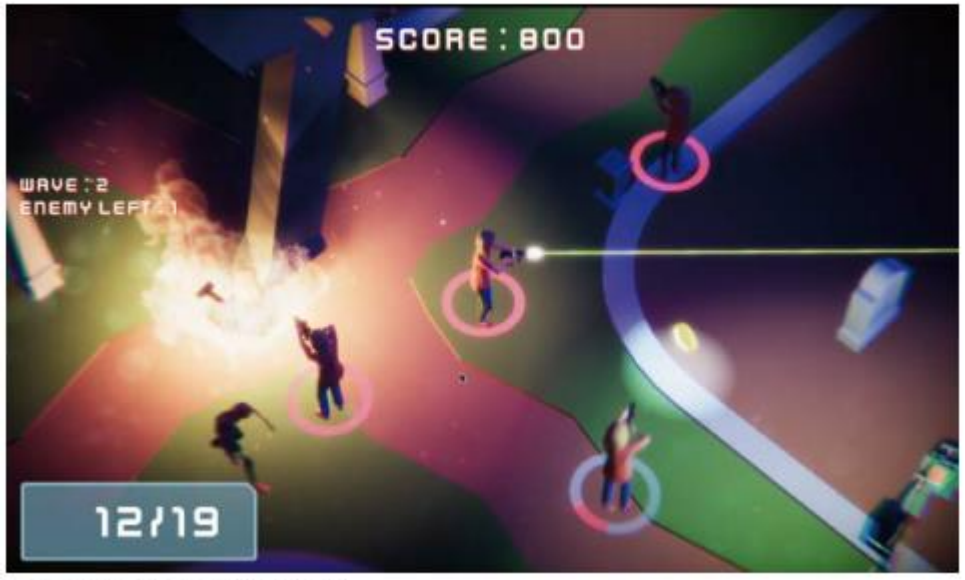
네트워크 게임 기능

네트워크 게임 기능

- 기본 게임 플레이 구성은 싱글 플레이어 버전과 동일
- 최대 4인의 멀티 플레이어
- 사망한 플레이어는 5초 뒤에 다시 부활
- 메치메이킹 시스템
  로비에서 인터넷을 통해 자동으로 빈 룸을 찾아 다른 플레이어의 게임에 참가

네트워크 게임 기능

최대 4명의 플레이어가 서로 협동하여 좀비를 학살하라!



▶ 한 룸에 접속한 4명의 플레이어

▶ 사망한 플레이어 캐릭터

네트워크 게임 기능

▶ 매치메이킹 로비

조작법

- 캐릭터 회전 : ←, → 또는 A, D
- 캐릭터 전진/후진 : ↑, ↓ 또는 W, S
- 발사 : 마우스 왼쪽 버튼
- 재장전 : R
- 룸 나가기 : Esc

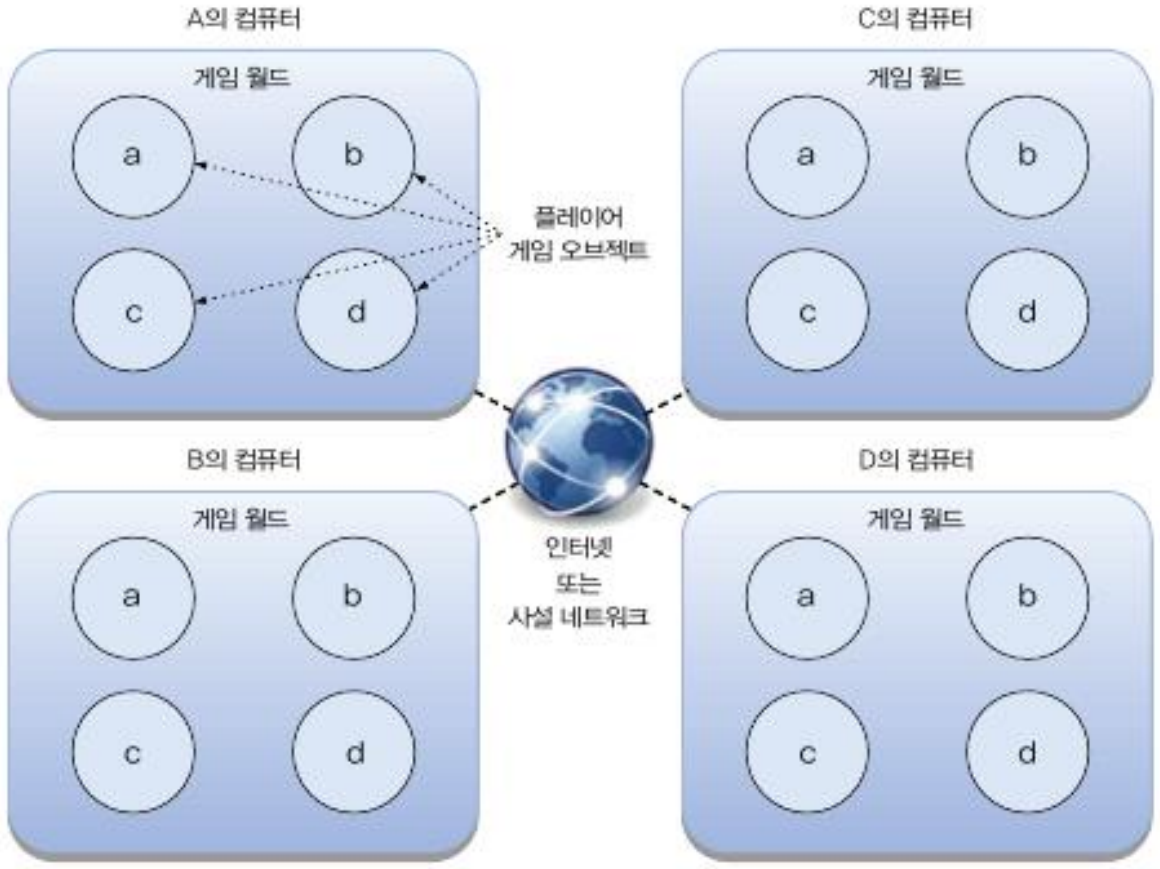
게임 네트워크

게임 네트워크

- Warrors of the Net(출처: https://www.warriorsofthe.net/)

게임 네트워크

- 동기화



▶ 4인 멀티플레이어

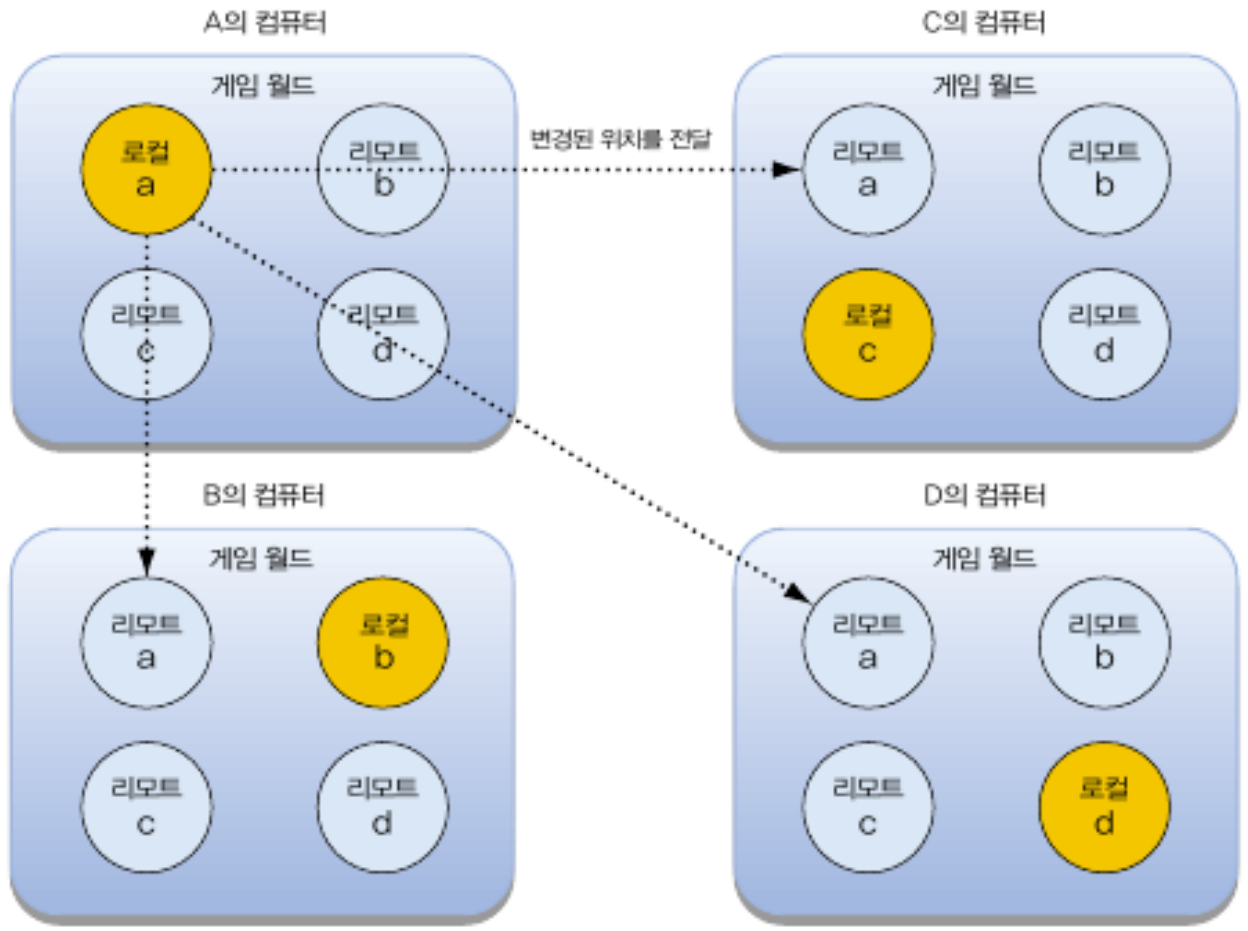
• 플레이어 캐릭터 a(클라이언트 A) ← 동기화 → 플레이어 캐릭터 a(클라이언트 B)

게임 네트워크

▶ 로컬과 리모트의 구분

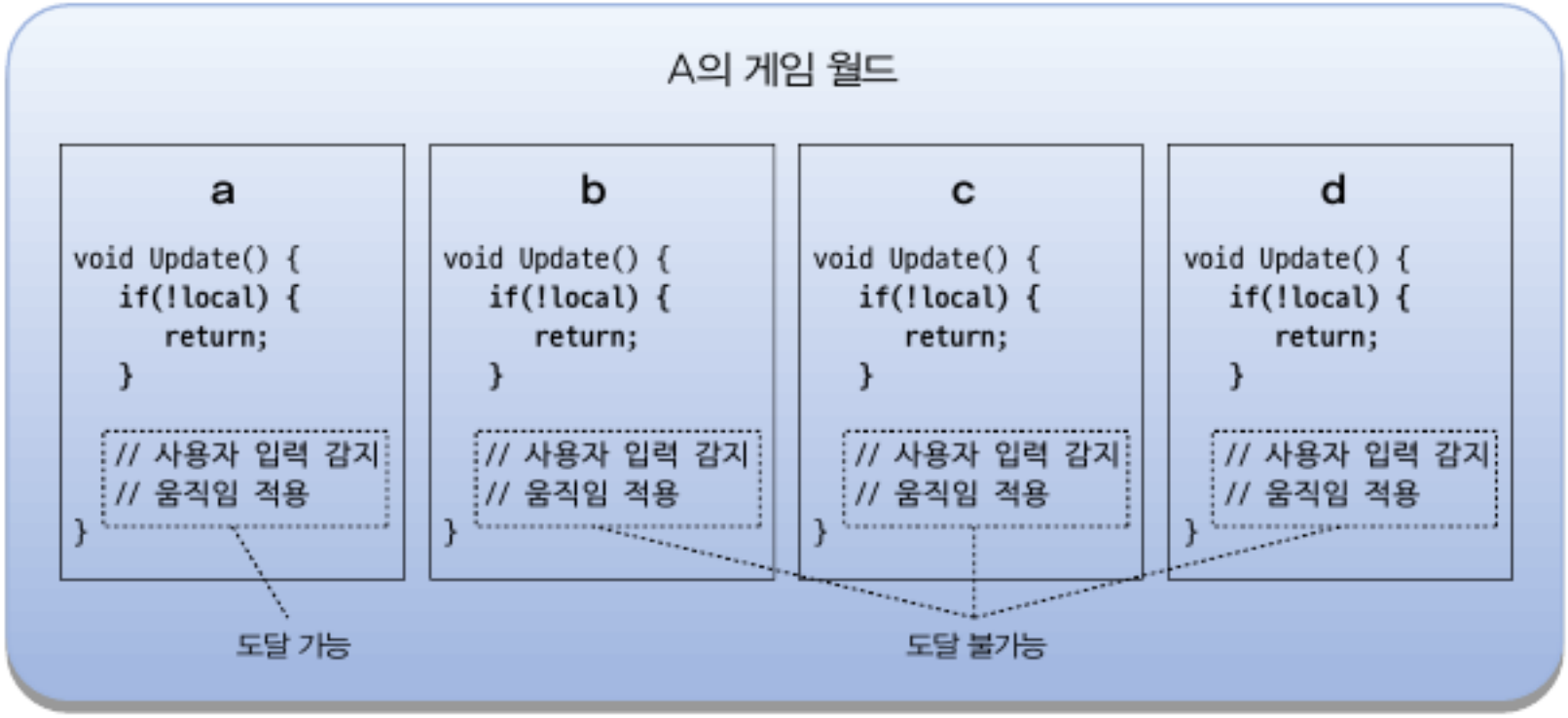
- **로컬 오브젝트** : 주도권이 자신에게 있음
- **리모트 오브젝트** : 주도권이 네트워크 너머의 타인에게 있음

게임 네트워크



▶ 위치 동기화

게임 네트워크



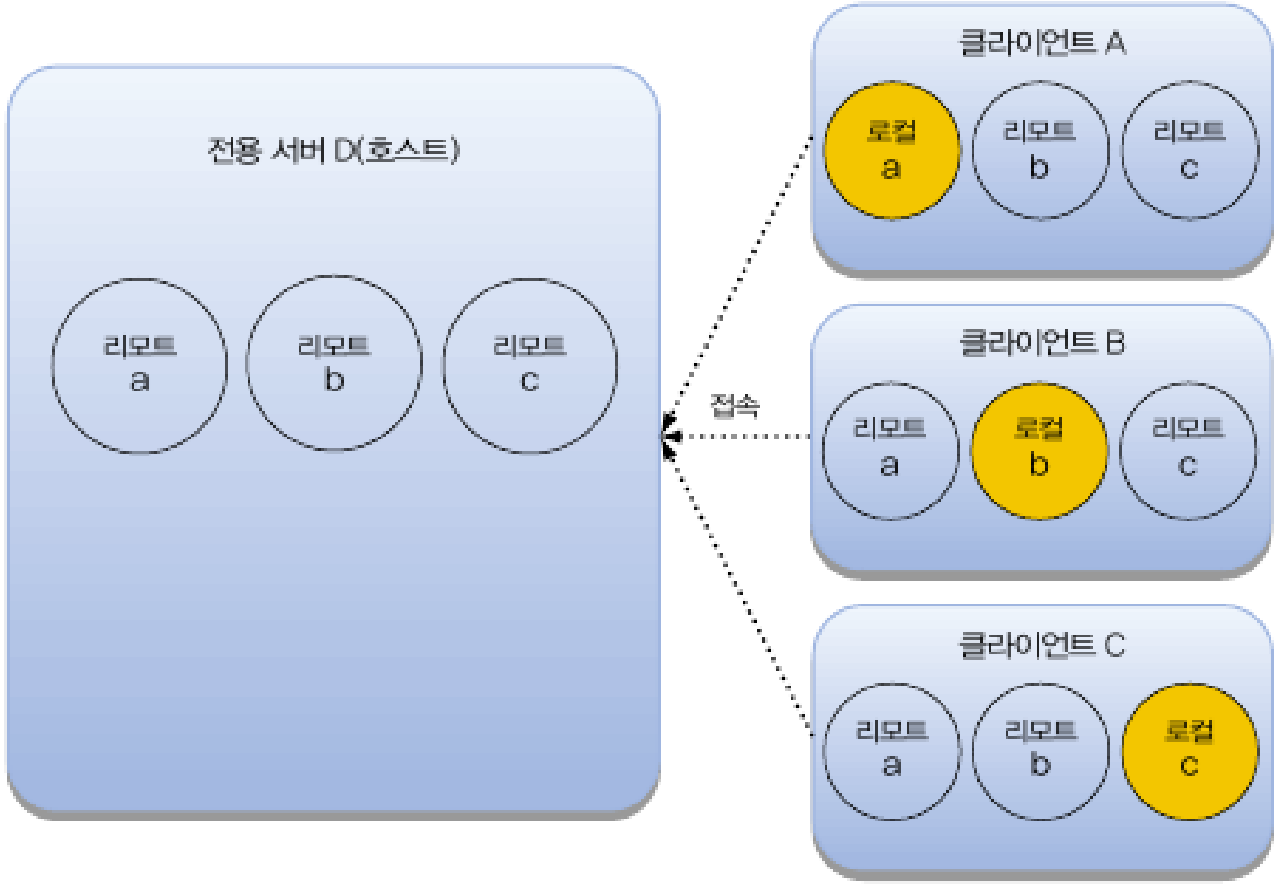
▶ 로컬 권한 검사

게임 네트워크

- 게임 서버의 종류
  - 전용 서버 (Dedicated Server)
  - 리슨 서버 (Listen Server)
  - P to P (Peer-to-Peer)

게임 네트워크

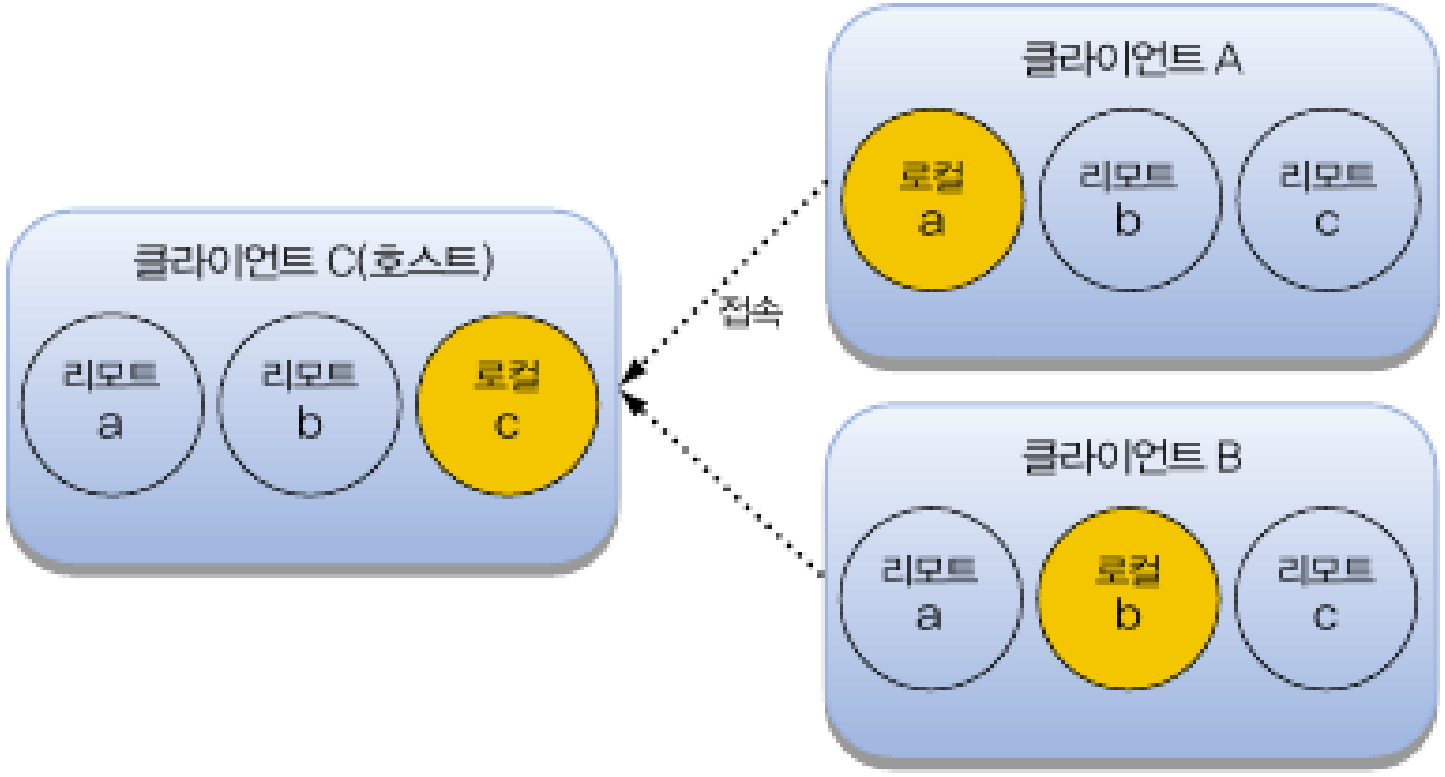
- **전용 서버 (Dedicated Server)**
  - 서버의 모든 자원인 온전히 네트워크 서비스를 유지하는 데 사용
  - 서버가 플레이어로서 게임에 직접 참가하지 않는다.
  - 고성능 고비용



▶ 전용 서버 방식

게임 네트워크

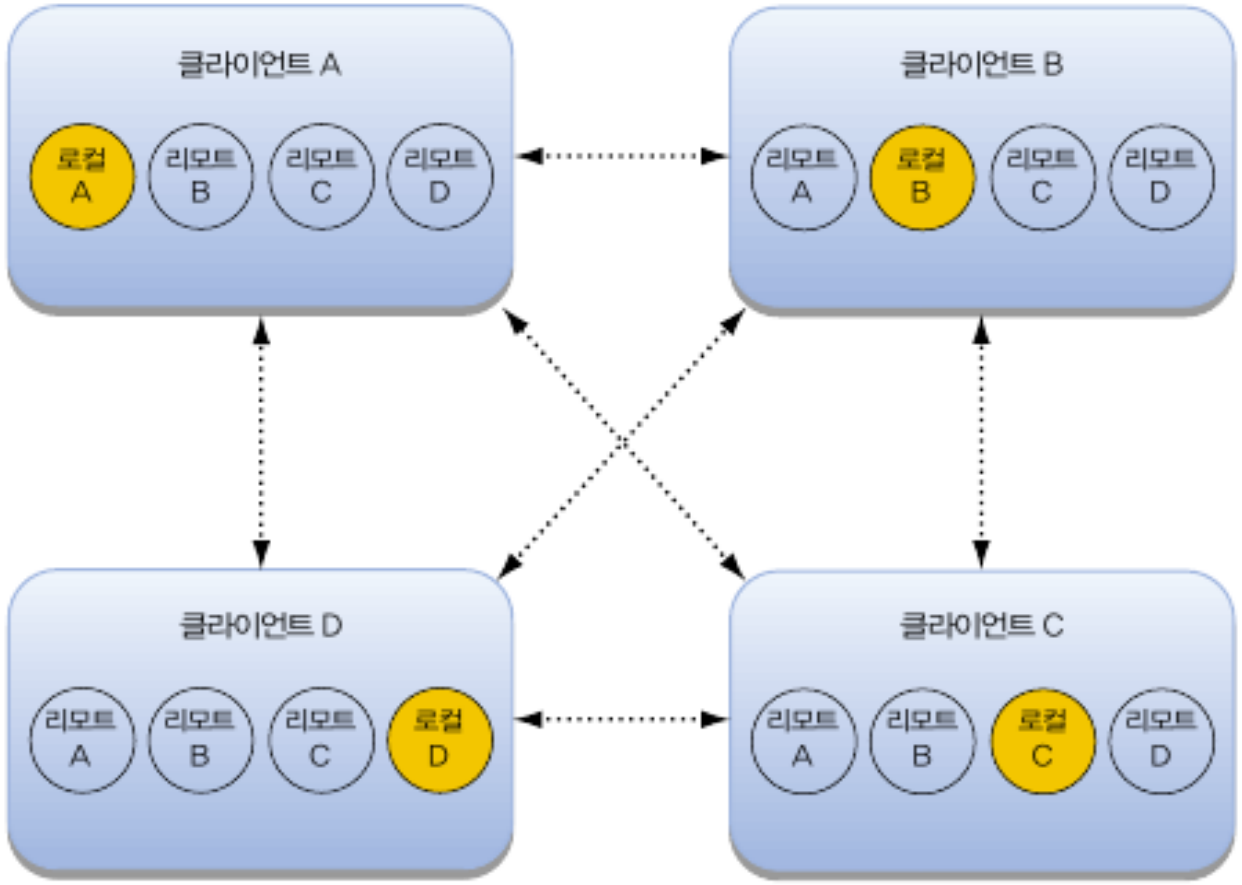
- **리슨 서버 (Listen Server)**
  - 클라이언트 중 하나가 서버 역할
  - 리슨 서버가 플레이어로서 게임에 직접 참가
    (방장, 호스트, 마스터)
  - 적은 유지비용
  - 리슨 서버에 따라 네트워크 품질 차이
  - 호스트에 연산 부담
  - 호스트 종료 시 새로운 호스트 선정



▶ 리슨 서버 방식

게임 네트워크

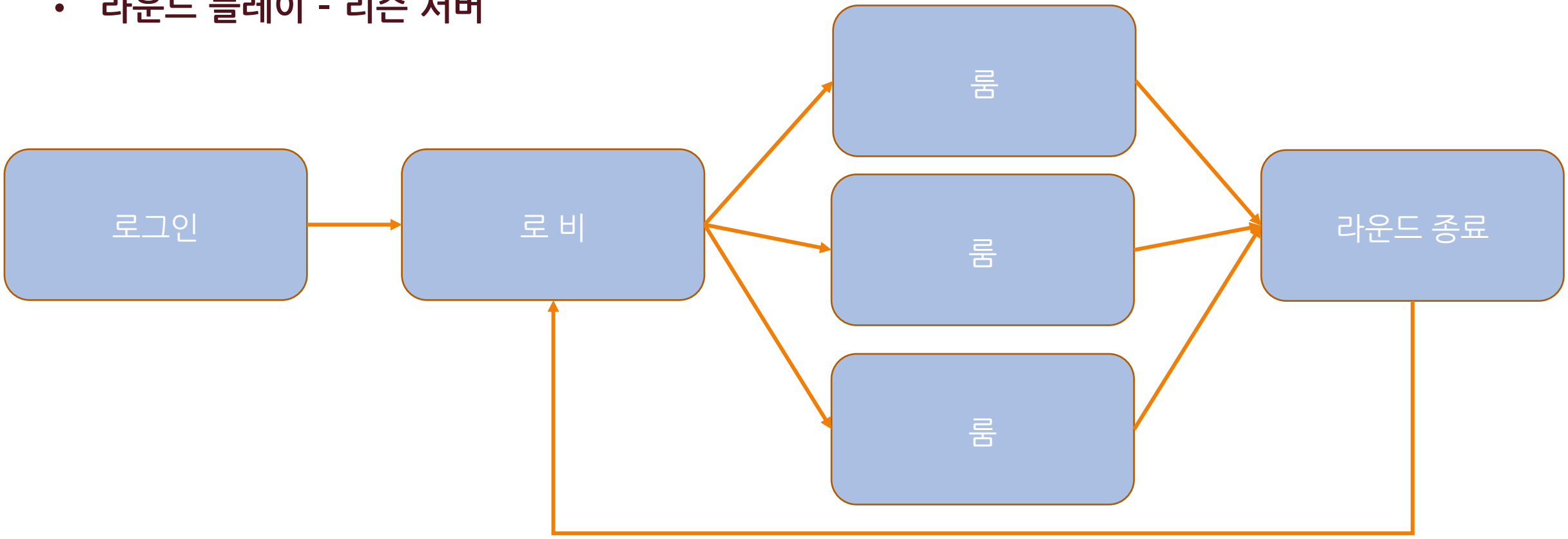
- P to P (Peer-to-Peer)
  - 참가한 클라이언트들 모두가 호스트 역할
  - 특정 호스트가 없기 때문에 클라이언트가 각자 자신의 월드에 자신의 담당 연산을 실행하고 전파
  - 적은 유지비용
  - 참가자가 증가할 수록 반응속도가 느려짐
  - 수치 변조에 취약



▶ P2P 방식

게임 네트워크

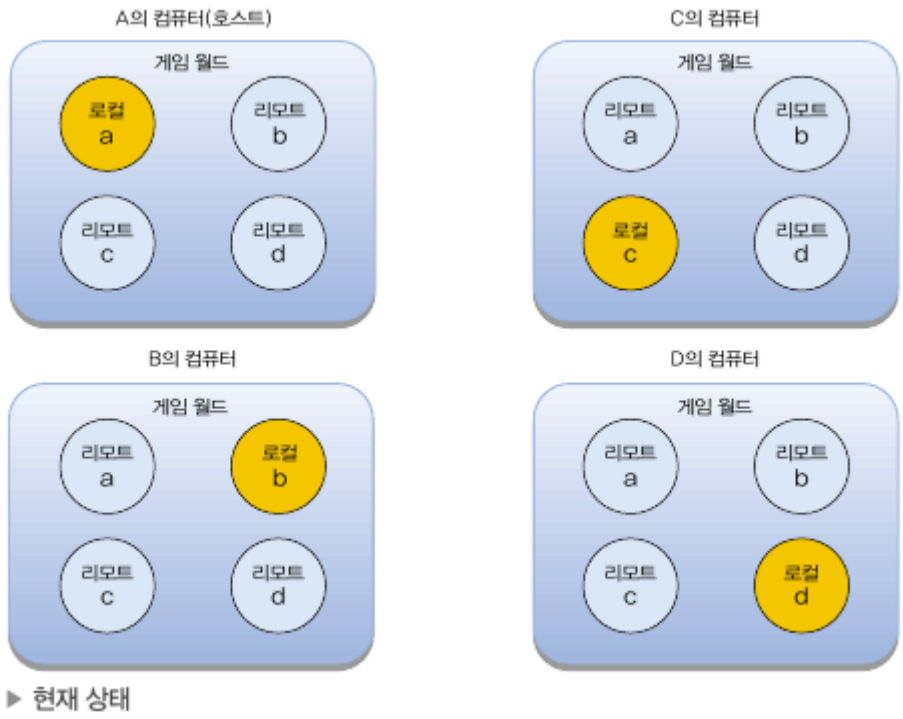
- 전용 서버 (Dedicated Server) + 리슨 서버 (Listen Server)
  - 매치메이킹 서버 - 전용서버
  - 라운드 플레이 - 리슨 서버



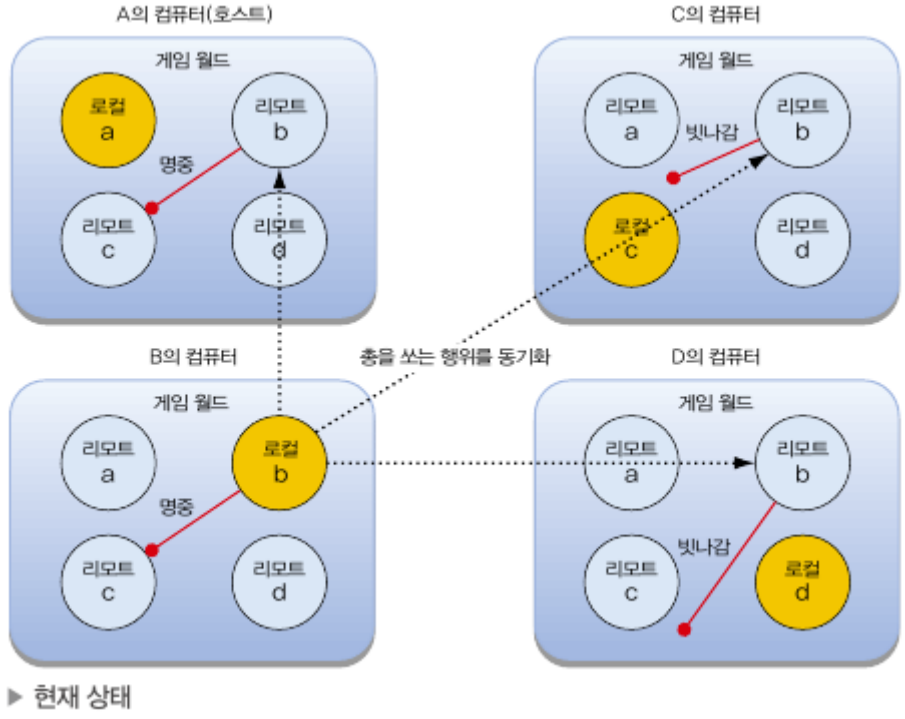
게임 네트워크

- 네트워크 권한 분리
  - 중요한 연산은 모두 서버(호스트)에 위임 - 보안성
  - 동기화에 오차가 존재하는 경우 기준이 되는 월드를 정하기 위해
  - 클라이언트의 변조나 위조 행위를 막기 위해

게임 네트워크



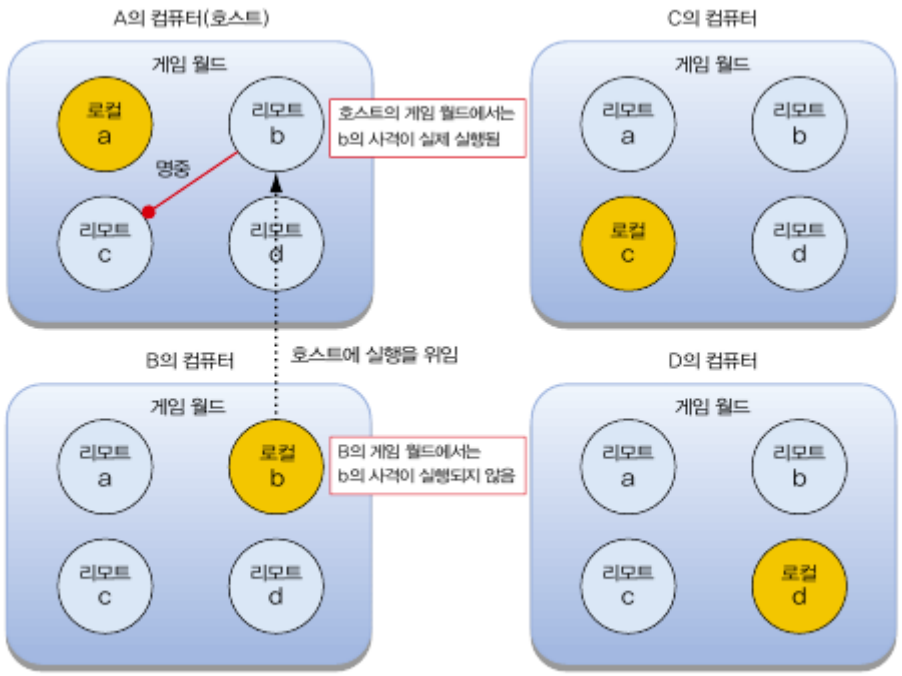
▶ 현재 상태



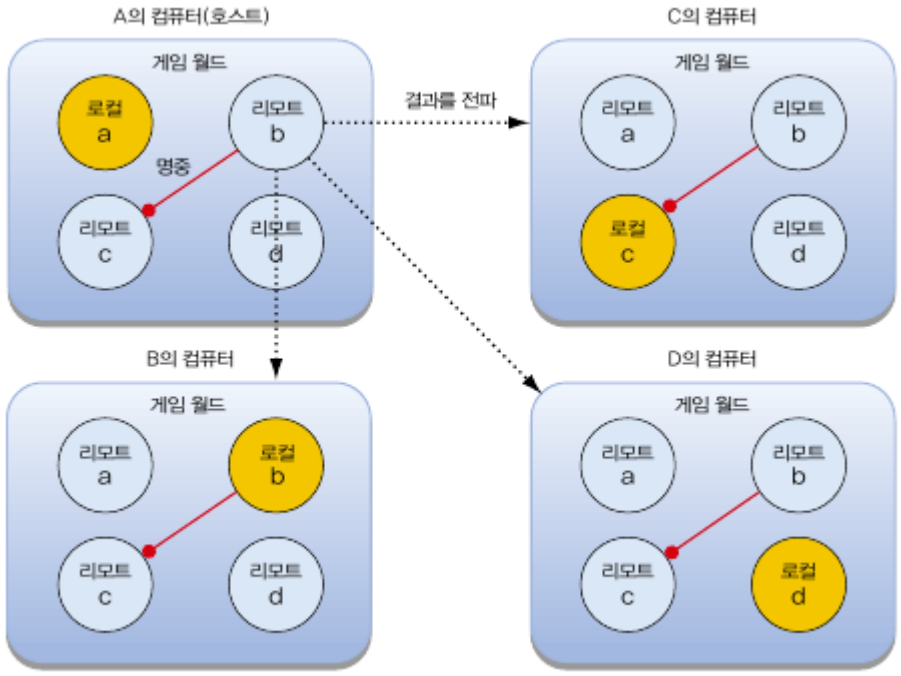
▶ 현재 상태

사용자 B가 마우스를 클릭하여 b가 c를 향해 총을 쐈다고 가정

게임 네트워크



▶ 사격 실행을 호스트에 위임



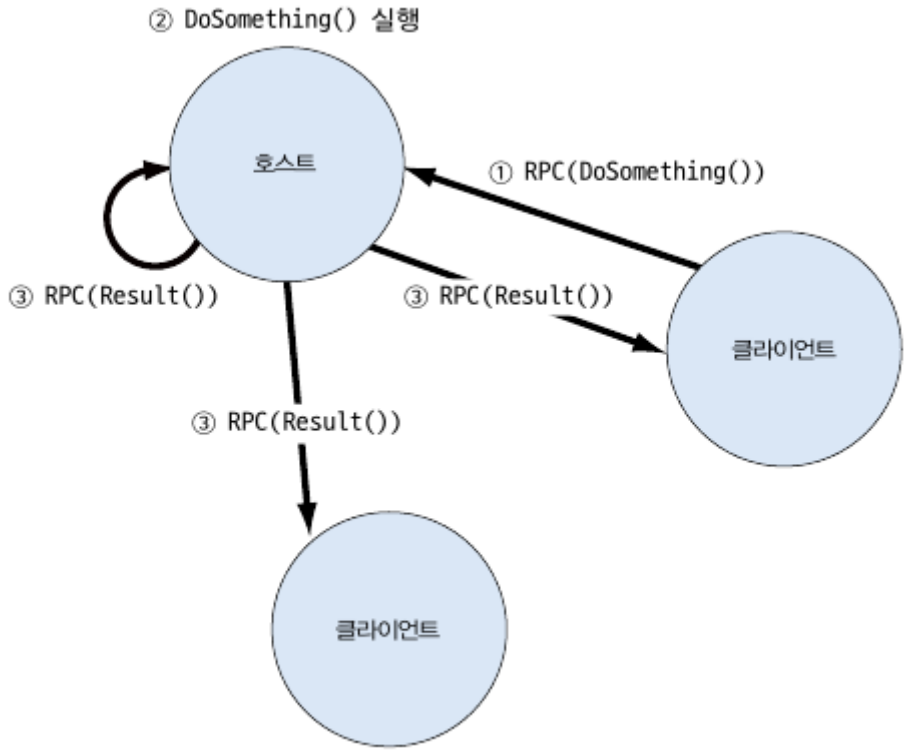
▶ 사격 결과를 전파

게임 네트워크

- RPC(remote procedure call)
  - 어떤 메서드나 처리를 네트워크를 넘어서 다른 클라이언트에서 실행

1. 사용자 B가 발사 버튼 누름
2. 클라이언트 B → 호스트 A로 RPC(`b.Shot()`) 전달
3. 호스트 A에서 `b.Shot()` 실행

1. 호스트 A → 클라이언트 A, B, C, D에 RPC(`b.ShotEffect()`) 전달
2. 클라이언트 A, B, C, D의 각 게임 월드에서 게임 오브젝트 b가 `ShotEffect()`를 실행하여 사격 효과를 재생



▶ 호스트와 클라이언트 사이의 실행 흐름

포톤 소개

포톤 소개

- **포톤(photon)**
  https://www.photonengine.com/ko-KR/PUN
  https://assetstore.unity.com/packages/tools/network/pun-2-free-119922

- **서버리스 (게임) 아키텍처**
  분산 서버 설계 패턴들을 미리 설계 구현해 놓고, 서버 프로그래머들은 이 위에다가 자기가 필요한 비즈니스 (게임) 로직을 만들어 넣음

- 장점
  대용량 서버를 쉽게 만들 수 있다.
  개발기간이 단축된다.
  안정성이 높다
  비용 절감

- 단점
  특정 서버리스 아키텍처 제공업체를 사용하면 다른 제공업체로 전환하기 힘들다.
  서버리스 아키텍처 인프라 자체에 문제가 생기면 컨트롤하기 힘들다.
  대용량 사용자 처리는 적합하나. 실시간 처리는 속도가 미흡하다.

포톤 소개

[과정 01] 프로젝트 열기

① **Zombie Multiplayer** 프로젝트를 유니티로 열기

[과정 02] PUN 2 직접 임포트하기(1) - 필수 아님

① 유니티 **에셋 스토어**(https://assetstore.unity.com)에 접속하여 로그인

② **PUN 2 - Free** 검색 > **구매**(무료)

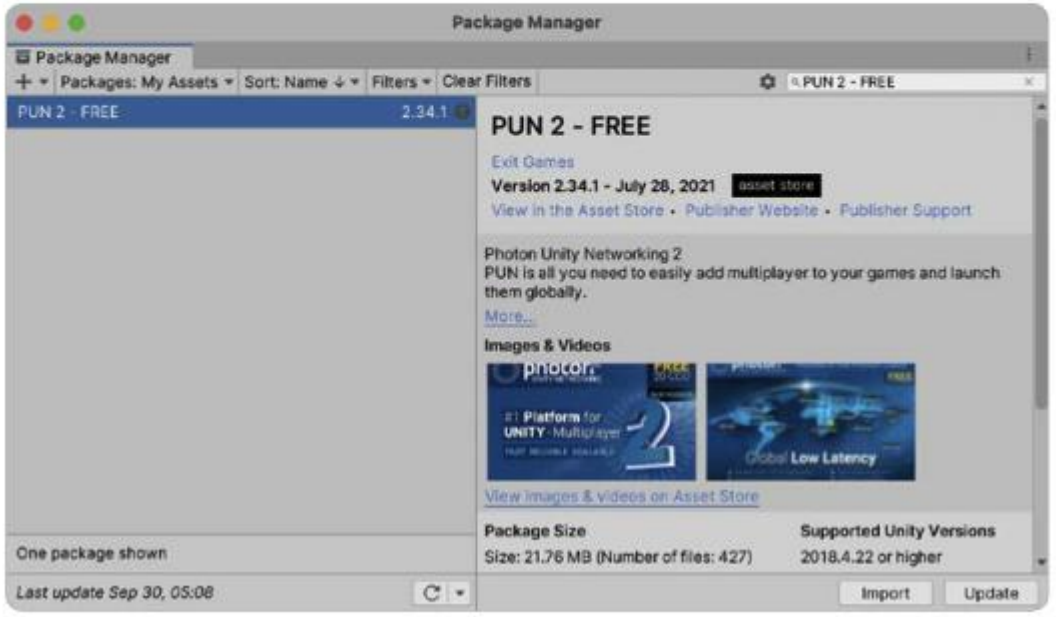
▶ PUN 2 직접 임포트하기(1)

[과정 03] PUN 2 직접 임포트하기(2) - 필수 아님

① 유니티 **패키지 매니저** 열기(**Window** > **Package Manager**)

② 패키지 **목록 필터**를 **My Assets**로 변경

③ **PUN 2 - FREE**를 찾아 선택 > **Download** 버튼 클릭 > 다운로드 후 **Import** 버튼 클릭



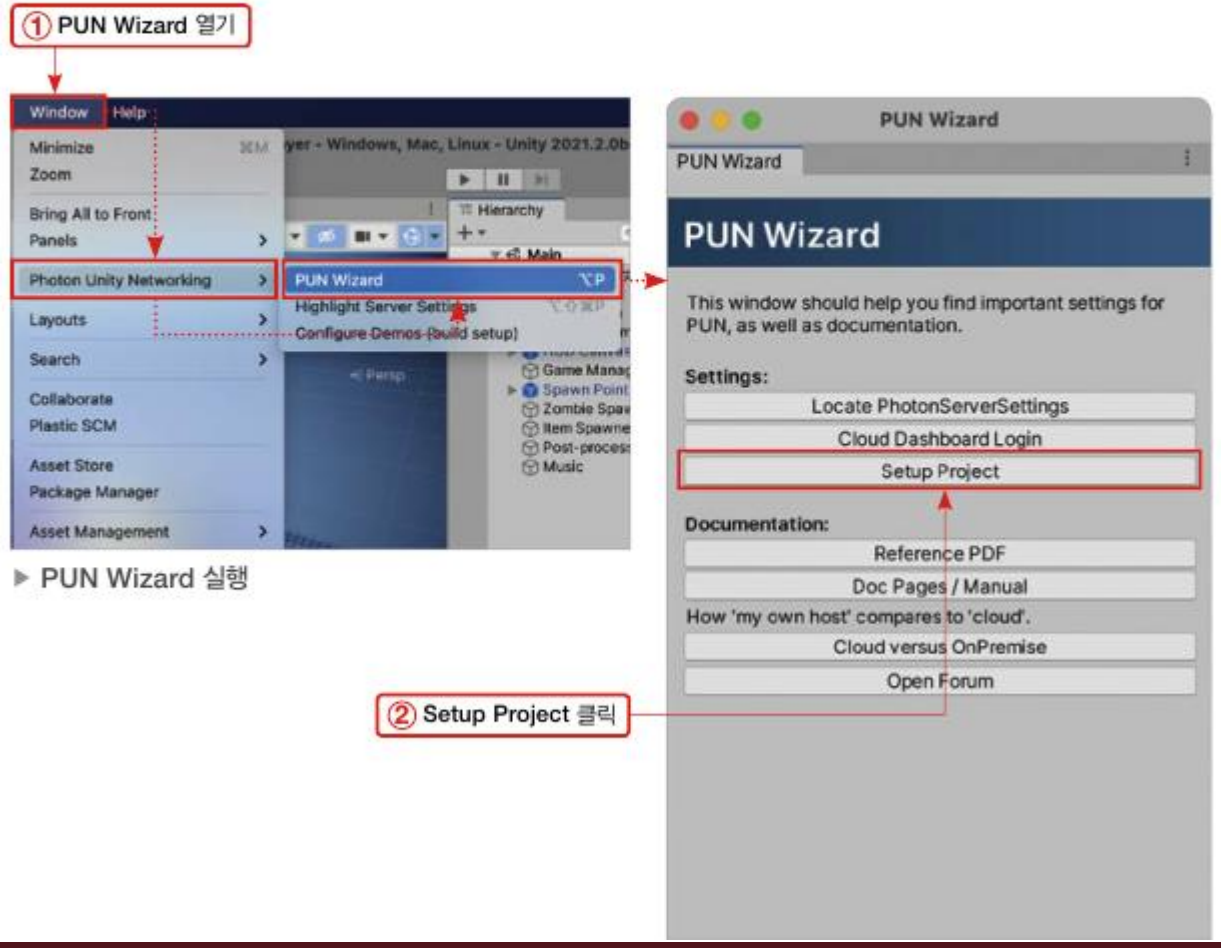
▶ PUN 2 직접 임포트하기(2)

포톤 소개

[과정 04] PUN Wizard 실행

① PUN Wizard 열기(Window > Photon Unity Networking > PUN Wizard)

② Setup Project 클릭

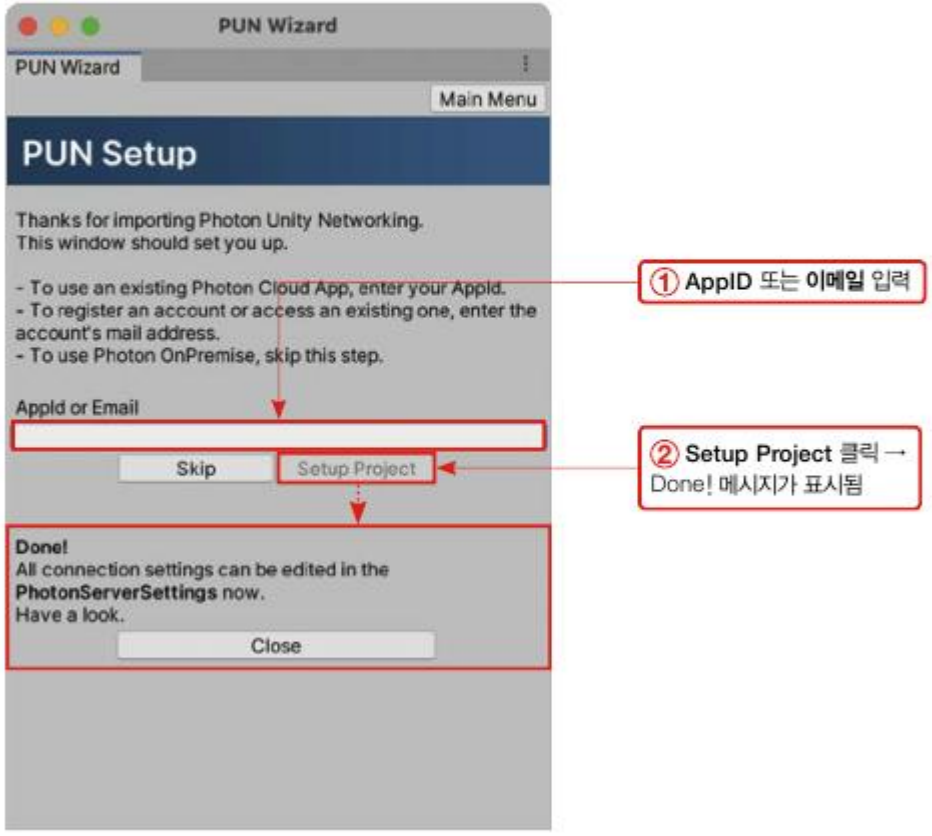


▶ PUN Wizard 실행

[과정 05] AppID 설정하기

① PUN Wizard에 AppID 또는 이메일 입력

② Setup Project 클릭 → Done! 메시지가 표시됨



▶ AppID 설정하기

포톤 소개

▶ 포톤 공식 웹에서 확인한 AppID

로비 구현

로비 구현

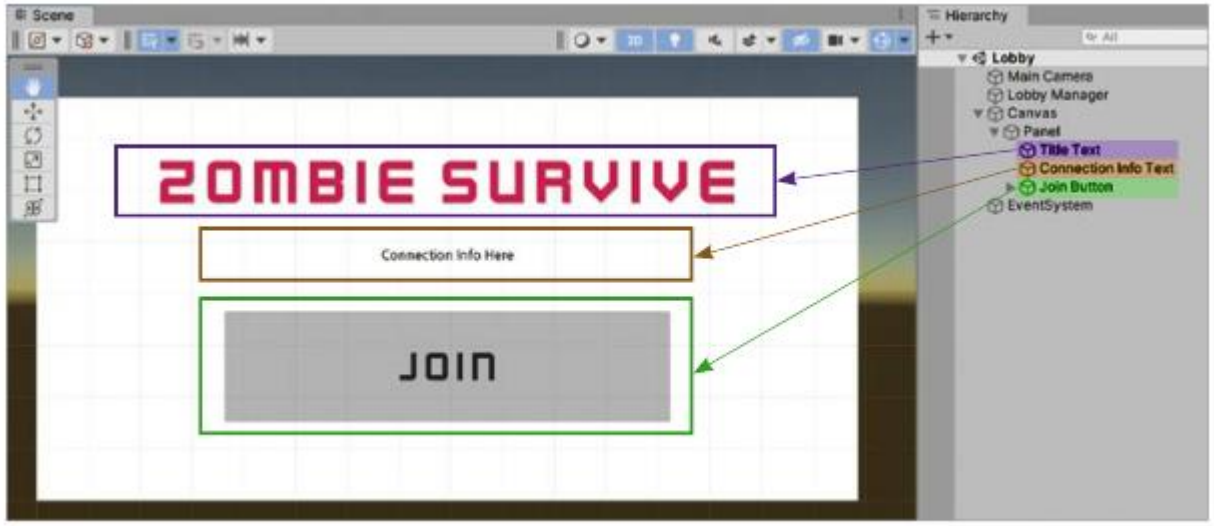
[과정 01] 로비 씬 열기

① **프로젝트** 창에서 **Scenes** 폴더의 **Lobby** 씬 열기

▶ 로비 씬 열기

Lobby 씬의 모습

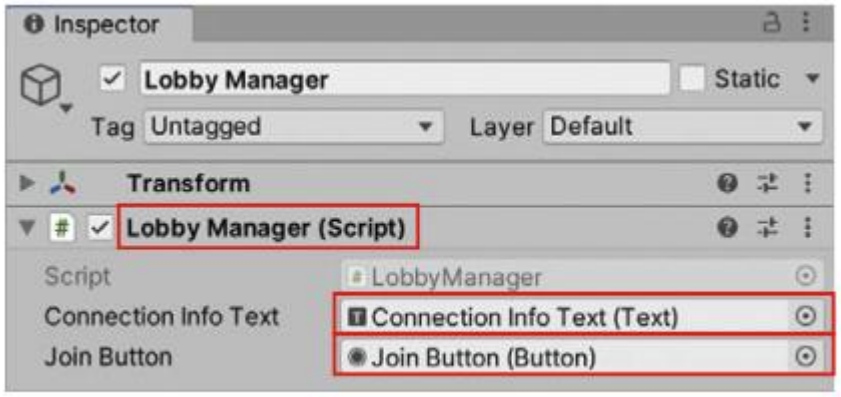
① Scenes 폴더의 Lobby 씬 열기



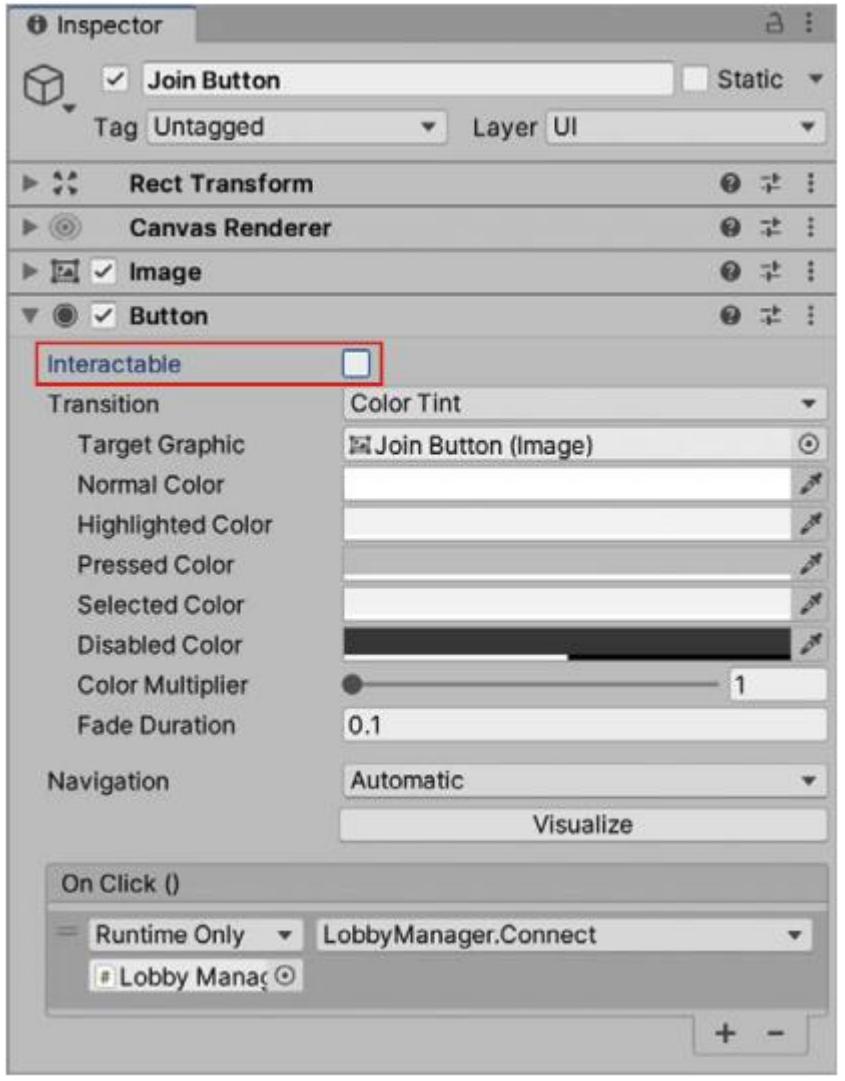
▶ Lobby 씬 구성

- **Main Camera** : 카메라
- **Lobby Manager** : 네트워크 로비 관리자
- **Canvas** : UI 캔버스
  - Panel : 단순 배경 패널
    - Title Text : 단순 제목 텍스트
    - Connection Info Text : 네트워크 접속 정보 표시 텍스트
    - Join Button : 룸 접속 시작 버튼
- **EventSystem** : UI 이벤트 관리자

로비 구현



▶ Lobby Manager의 모습



▶ Join Button 게임 오브젝트

로비 구현

[과정 01] LobbyManager의 Start() 메서드 완성하기

① Start() 메서드를 다음과 같이 완성

```
void Start() {
    // 접속에 필요한 정보(게임 버전) 설정
    PhotonNetwork.GameVersion = gameVersion;
    // 설정한 정보로 마스터 서버 접속 시도
    PhotonNetwork.ConnectUsingSettings();

    // 룸 접속 버튼 잠시 비활성화
    joinButton.interactable = false;
    // 접속 시도 중임을 텍스트로 표시
    connectionInfoText.text = "마스터 서버에 접속 중...";
}
```

[과정 01] LobbyManager의 OnConnectedToMaster() 메서드 완성하기

① OnConnectedToMaster() 메서드를 다음과 같이 완성

```
public override void OnConnectedToMaster() {
    // 룸 접속 버튼 활성화
    joinButton.interactable = true;
    // 접속 정보 표시
    connectionInfoText.text = "온라인 : 마스터 서버와 연결됨";
}
```

로비 구현

[과정 01] LobbyManager의 OnDisconnected ( ) 메서드 완성

① OnDisconnected ( ) 메서드를 다음과 같이 완성

```
public override void OnDisconnected(DisconnectCause cause) {
    // 룸 접속 버튼 비활성화
    joinButton.interactable = false;
    // 접속 정보 표시
    connectionInfoText.text = "오프라인 : 마스터 서버와 연결되지 않음\n접속 재시도 중...";

    // 마스터 서버로의 재접속 시도
    PhotonNetwork.ConnectUsingSettings();
}
```

[과정 01] LobbyManager의 Connect ( ) 메서드 완성하기

① Connect ( ) 메서드를 다음과 같이 완성

```
public void Connect() {
    // 중복 접속 시도를 막기 위해 접속 버튼 잠시 비활성화
    joinButton.interactable = false;

    // 마스터 서버에 접속 중이라면
    if (PhotonNetwork.IsConnected)
    {
        // 룸 접속 실행
        connectionInfoText.text = "룸에 접속...";
        PhotonNetwork.JoinRandomRoom();
    }
    else
    {
        // 마스터 서버에 접속 중이 아니라면 마스터 서버에 접속 시도
        connectionInfoText.text = "오프라인 : 마스터 서버와 연결되지 않음\n접속 재시도 중...";
        // 마스터 서버로의 재접속 시도
        PhotonNetwork.ConnectUsingSettings();
    }
}
```

로비 구현

[과정 01] LobbyManager의 OnJoinRandomFailed ( ) 메서드 완성하기

① OnJoinRandomFailed ( ) 메서드를 다음과 같이 완성

```
public override void OnJoinRandomFailed(short returnCode, string message) {
    // 접속 상태 표시
    connectionInfoText.text = "빈 방이 없음, 새로운 방 생성...";
    // 최대 4명을 수용 가능한 빈 방 생성
    PhotonNetwork.CreateRoom(null, new RoomOptions {MaxPlayers = 4});
}
```

[과정 01] LobbyManager의 OnJoinedRoom ( ) 메서드 완성하기

① OnJoinedRoom ( ) 메서드를 다음과 같이 완성

```
public override void OnJoinedRoom() {
    // 접속 상태 표시
    connectionInfoText.text = "방 참가 성공";
    // 모든 룸 참가자가 Main 씬을 로드하게 함
    PhotonNetwork.LoadLevel("Main");
}
```

캐릭터 움직임 개선

캐릭터 움직임 개선

- 방향키 방향으로 이동
- 항상 이동 방향 바라보기
- PlayerMovement 컴포넌트에서 선택하여 플레이 가능

캐릭터 움직임 개선

- PlayerMovement.cs

```
using UnityEngine;

// 플레이어 캐릭터를 사용자 입력에 따라 움직이는 스크립트
public class PlayerMovement : MonoBehaviour
{
    public enum MovingType
    {
        MoveAndRotate,
        MoveForward,
    };

    public MovingType movingType = MovingType.MoveAndRotate;

    public float moveSpeed = 5f; // 앞뒤 움직임의 속도
    public float rotateSpeed = 180f; // 좌우 회전 속도


    private PlayerInput playerInput; // 플레이어 입력을 알려주는 컴포넌트
    private Rigidbody playerRigidbody; // 플레이어 캐릭터의 리지드바디
    private Animator playerAnimator; // 플레이어 캐릭터의 애니메이터
```

캐릭터 움직임 개선

```
private void Start()
{
    // 사용할 컴포넌트들의 참조를 가져오기
    playerInput = GetComponent<PlayerInput>();
    playerRigidbody = GetComponent<Rigidbody>();
    playerAnimator = GetComponent<Animator>();
}

/// <summary>
/// MoveForward 이동시 방향 저장
/// </summary>
private Vector3 _dir;

/// <summary>
/// MoveForward 이동시 방향 계산
/// </summary>
private void Update()
{
    if (movingType == MovingType.MoveForward)
    {
        _dir.x = playerInput.move;
        _dir.z = -playerInput.rotate;   // A키가 +z축으로 세팅
```

캐릭터 움직임 개선

```
            _dir.Normalize();

            // 뒤로 이동하는 애니메이션을 방지 하기 위해 절대값 사용
            playerAnimator.SetFloat("Move", Mathf.Abs(_dir.x) + Mathf.Abs(_dir.z));
        }
    }

    // FixedUpdate는 물리 갱신 주기에 맞춰 실행됨
    private void FixedUpdate()
    {
        // 물리 갱신 주기마다 움직임, 회전, 애니메이션 처리 실행
        switch (movingType)
        {
            case MovingType.MoveAndRotate:
                MoveAndRotate();
                break;

            case MovingType.MoveForward:
                MoveForward();
                break;
        }

    }
```

캐릭터 움직임 개선

```
/// <summary>
/// 키보드 방향으로 이동
/// </summary>
private void MoveForward()
{
    Vector3 moveDistance =
        _dir * moveSpeed * Time.deltaTime;

    playerRigidbody.MovePosition(playerRigidbody.position + moveDistance);
    transform.LookAt(playerRigidbody.position + _dir);
}

/// <summary>
/// 이동키와 회전키를 이용한 이동
/// </summary>
private void MoveAndRotate()
{
    Rotate();
    Move();

    playerAnimator.SetFloat("Move", playerInput.move);
}
```

캐릭터 움직임 개선

```
    // 입력값에 따라 캐릭터를 앞뒤로 움직임
    private void Move()
    {
        Vector3 moveDistance =
            playerInput.move * transform.forward * moveSpeed * Time.deltaTime;
        playerRigidbody.MovePosition(playerRigidbody.position + moveDistance);
    }

    // 입력값에 따라 캐릭터를 좌우로 회전
    private void Rotate() {
        float turn = playerInput.rotate * rotateSpeed * Time.deltaTime;
        playerRigidbody.rotation =
            playerRigidbody.rotation * Quaternion.Euler(0, turn, 0);
    }
}
```

캐릭터 움직임 개선

- 가상 조이스틱과 연동

캐릭터 움직임 개선

- PlayerInput.cs

```
using UnityEngine;

// 플레이어 캐릭터를 조작하기 위한 사용자 입력을 감지
// 감지된 입력값을 다른 컴포넌트들이 사용할 수 있도록 제공
public class PlayerInput : MonoBehaviour
{
    public string moveAxisName = "Vertical"; // 앞뒤 움직임을 위한 입력축 이름
    public string rotateAxisName = "Horizontal"; // 좌우 회전을 위한 입력축 이름
    public string fireButtonName = "Fire1"; // 발사를 위한 입력 버튼 이름
    public string reloadButtonName = "Reload"; // 재장전을 위한 입력 버튼 이름
    public string changeWeaponButtonName = "Change";  // 무기 변경을 위한 입력 버튼 이름

    // 값 할당은 내부에서만 가능
    public float move { get; private set; } // 감지된 움직임 입력값
    public float rotate { get; private set; } // 감지된 회전 입력값
    public bool fire { get; private set; } // 감지된 발사 입력값
    public bool reload { get; private set; } // 감지된 재장전 입력값
    public bool changeWeapon { get; private set; } // 감지된 무기 변경 입력값
```

캐릭터 움직임 개선

```
[SerializeField] private Joystick joystick;

// 매프레임 사용자 입력을 감지
private void Update()
{
    // 게임오버 상태에서는 사용자 입력을 감지하지 않는다
    if (GameManager.instance != null && GameManager.instance.isGameover)
    {
        move = 0;
        rotate = 0;
        fire = false;
        reload = false;
        changeWeapon = false;
        return;
    }


    if (joystick != null && joystick.gameObject.activeSelf)
    {
        // move에 관한 입력 감지
        move = joystick.Vertical;
        // rotate에 관한 입력 감지
```

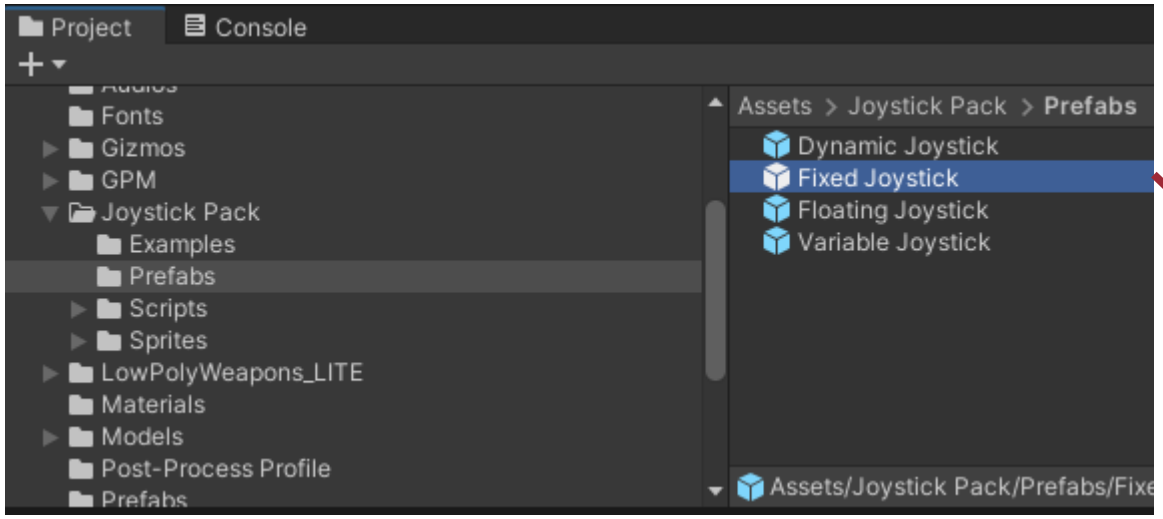
캐릭터 움직임 개선

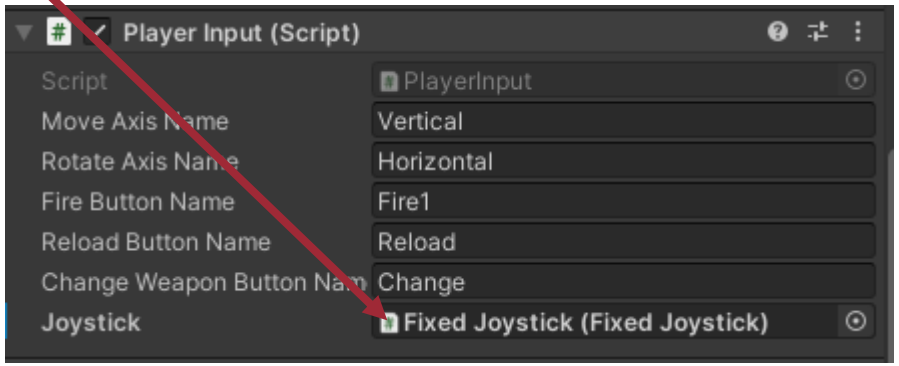
```
            rotate = joystick.Horizontal;
        }
        else
        {
            // move에 관한 입력 감지
            move = Input.GetAxis(moveAxisName);
            // rotate에 관한 입력 감지
            rotate = Input.GetAxis(rotateAxisName);
            // fire에 관한 입력 감지
            fire = Input.GetButton(fireButtonName);
            // reload에 관한 입력 감지
            reload = Input.GetButtonDown(reloadButtonName);
            // change Weapon에 관한 입력 감지
            changeWeapon = Input.GetButton(changeWeaponButtonName);
        }

    }

    public void SetFire(bool fire)
    {
        this.fire = fire;
    }
```

캐릭터 움직임 개선

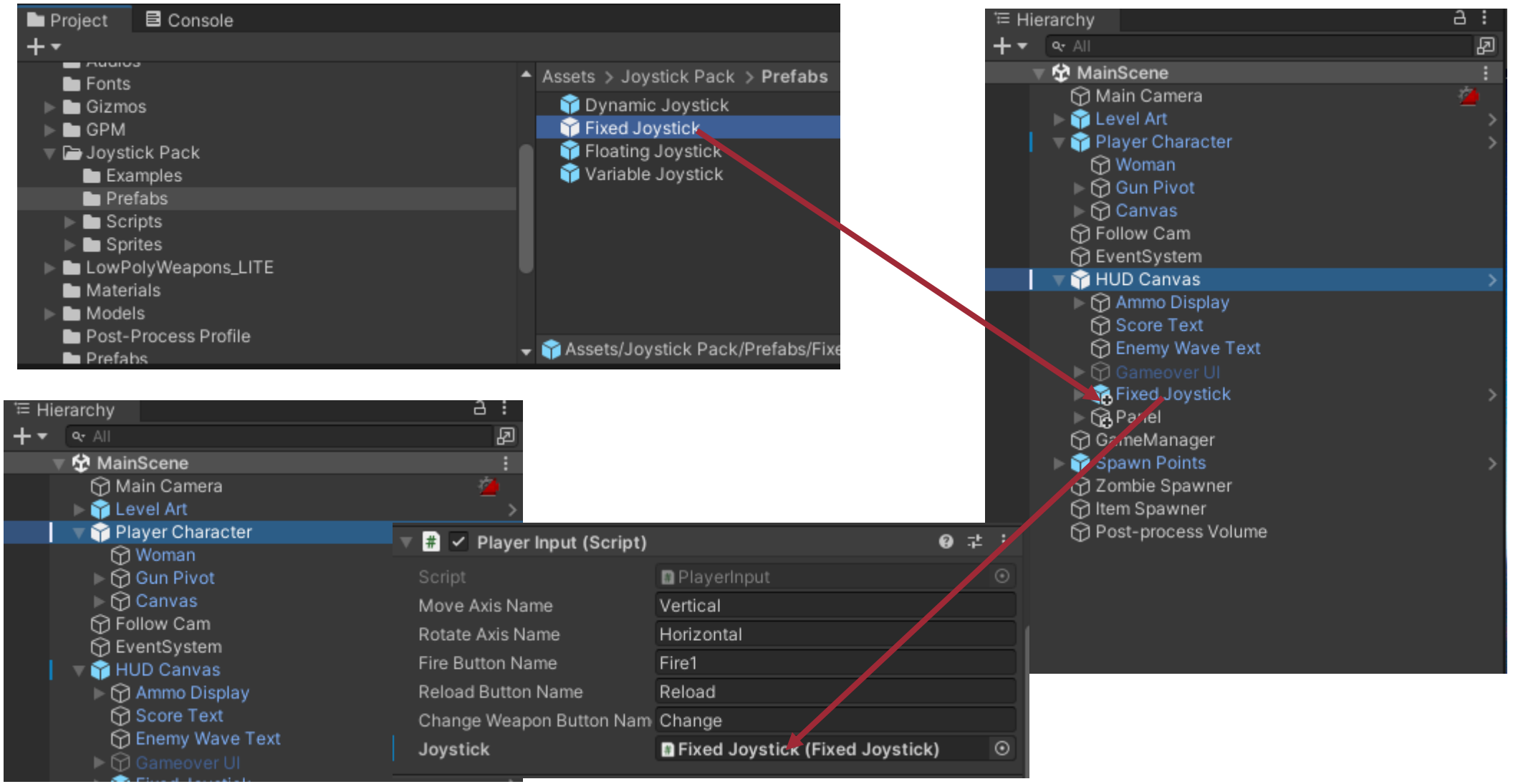
```
    public void SetReload(bool reload)
    {
        this.reload = reload;
    }

    public void SetChangeWeapon(bool changeWeapon)
    {
        this.changeWeapon = changeWeapon;
    }
}
```

캐릭터 움직임 개선

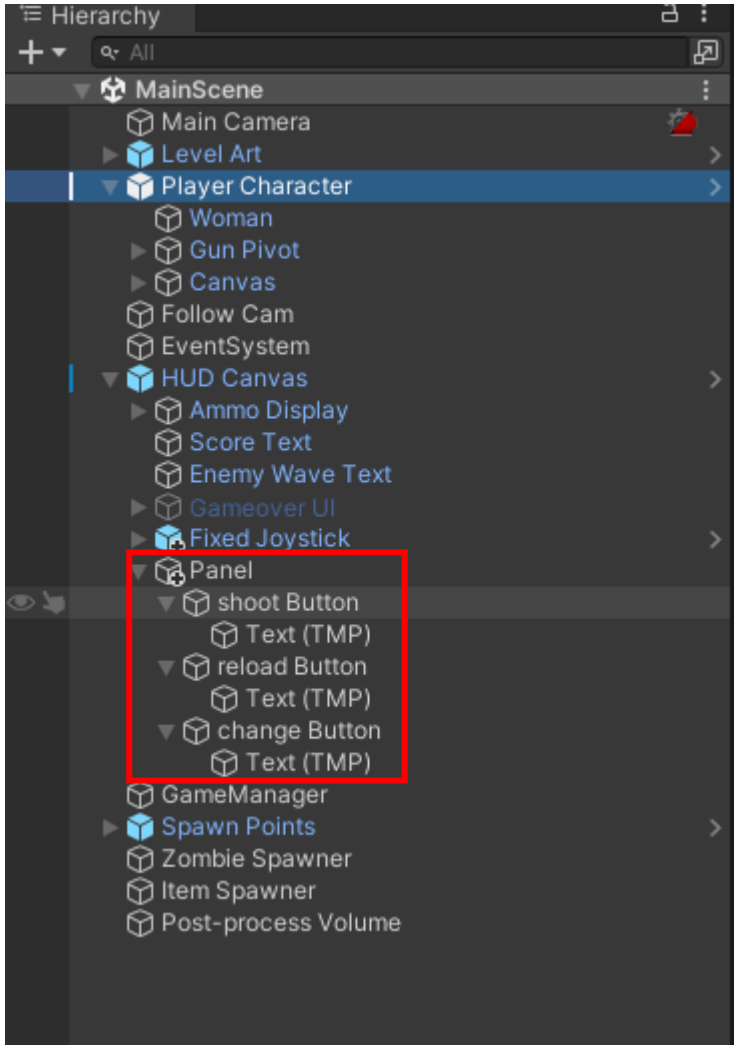

Project
Console
Fonts
Gizmos
GPM
Joystick Pack
Examples
Prefabs
Scripts
Sprites
LowPolyWeapons_LITE
Materials
Models
Post-Process Profile
Assets > Joystick Pack > Prefabs
Dynamic Joystick
Fixed Joystick
Floating Joystick
Variable Joystick
Assets/Joystick Pack/Prefabs/Fixe

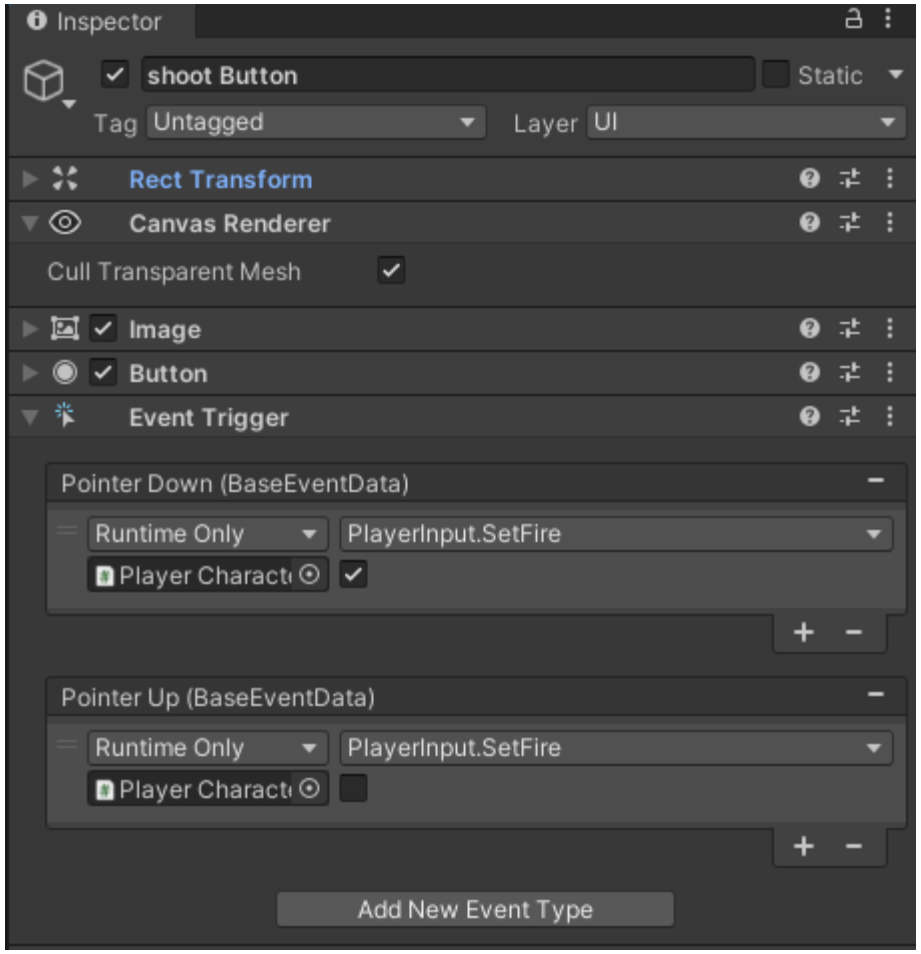

Player Input (Script)
Script
PlayerInput
Move Axis Name
Vertical
Rotate Axis Name
Horizontal
Fire Button Name
Fire1
Reload Button Name
Reload
Change Weapon Button Nam
Change
Joystick
Fixed Joystick (Fixed Joystick)

캐릭터 움직임 개선


Project
Console
Fonts
Gizmos
GPM
Joystick Pack
Examples
Prefabs
Scripts
Sprites
LowPolyWeapons_LITE
Materials
Models
Post-Process Profile
Assets > Joystick Pack > Prefabs
Dynamic Joystick
Fixed Joystick
Floating Joystick
Variable Joystick
Assets/Joystick Pack/Prefabs/Fixe
Hierarchy
All
MainScene
Main Camera
Level Art
Player Character
Woman
Gun Pivot
Canvas
Follow Cam
EventSystem
HUD Canvas
Ammo Display
Score Text
Enemy Wave Text
Gameover UI
Fixed Joystick
Spawn Points
Zombie Spawner
Item Spawner
Post-process Volume
Player Input (Script)
Script
PlayerInput
Move Axis Name
Vertical
Rotate Axis Name
Horizontal
Fire Button Name
Fire1
Reload Button Name
Reload
Change Weapon Button Nam
Change
Joystick
Fixed Joystick (Fixed Joystick)

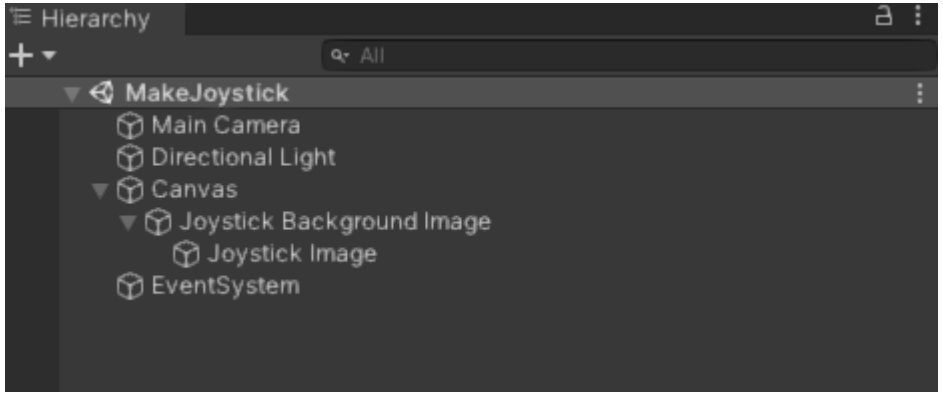
캐릭터 움직임 개선

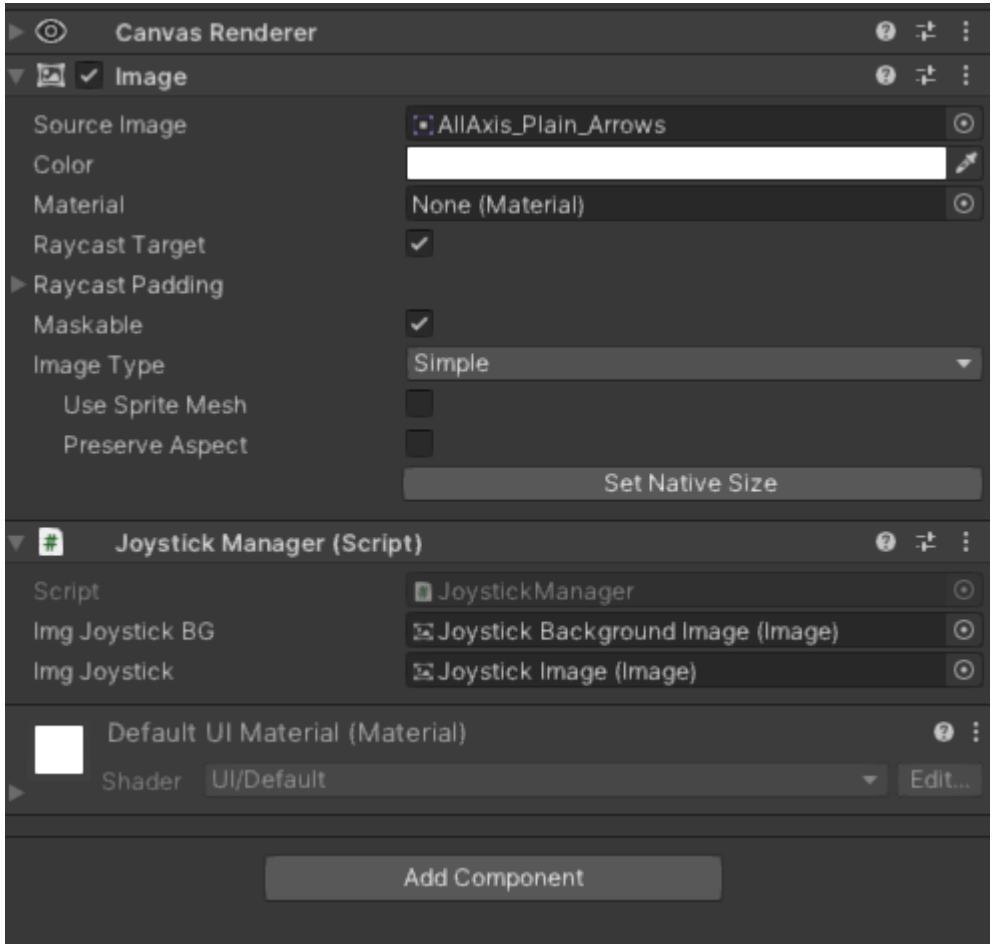

Hierarchy
All
MainScene
Main Camera
Level Art
Player Character
Woman
Gun Pivot
Canvas
Follow Cam
EventSystem
HUD Canvas
Ammo Display
Score Text
Enemy Wave Text
Gameover UI
Fixed Joystick
Panel
shoot Button
Text (TMP)
reload Button
Text (TMP)
change Button
Text (TMP)
GameManager
Spawn Points
Zombie Spawner
Item Spawner
Post-process Volume


Inspector
shoot Button
Static
Tag Untagged
Layer UI
Rect Transform
Canvas Renderer
Cull Transparent Mesh
Image
Button
Event Trigger
Pointer Down (BaseEventData)
Runtime Only
PlayerInput.SetFire
Player Charact
Pointer Up (BaseEventData)
Runtime Only
PlayerInput.SetFire
Player Charact
Add New Event Type

가상 조이스틱 만들기

가상 조이스틱 만들기


Hierarchy
All
MakeJoystick
Main Camera
Directional Light
Canvas
Joystick Background Image
Joystick Image
EventSystem


Canvas Renderer
Image
Source Image
AllAxis_Plain_Arrows
Color
Material
None (Material)
Raycast Target
Raycast Padding
Maskable
Image Type
Simple
Use Sprite Mesh
Preserve Aspect
Set Native Size
Joystick Manager (Script)
Script
JoystickManager
Img Joystick BG
Joystick Background Image (Image)
Img Joystick
Joystick Image (Image)
Default UI Material (Material)
Shader
UI/Default
Edit...
Add Component

가상 조이스틱 만들기

```
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

/// <summary>
/// 조이스틱 핸들 움직이고 2d 위치(-1 ~ 1) 반환
/// </summary>
public class JoystickManager : MonoBehaviour, IDragHandler, IPointerUpHandler, IPointerDownHandler
{
    /// <summary>
    /// 조이스틱 핸들 이미지
    /// </summary>
    [SerializeField]
    private Image imgJoystick;

    /// <summary>
    /// 조이스틱 바탕 이미지
    /// </summary>
    [SerializeField]
    private Image imgJoystickBackground;

    private Vector2 posInput;
```

가상 조이스틱 만들기

```
/// <summary>
/// 드래그 이벤트 핸들러
/// </summary>
/// <param name="eventData"></param>
public void OnDrag(PointerEventData eventData)
{
    if (RectTransformUtility.ScreenPointToLocalPointInRectangle(
                                imgJoystickBackground.rectTransform,
                                eventData.position,
                                eventData.pressEventCamera,
                                out posInput))
    {
        //Debug.Log(posInput.ToString());

        // 좌표를 imgJoystickBackground.rectTransform 크기에 비례(-1 ~ 1)
        posInput.x = posInput.x / (imgJoystickBackground.rectTransform.sizeDelta.x / 2);
        posInput.y = posInput.y / (imgJoystickBackground.rectTransform.sizeDelta.y / 2);

        // 바탕이미지를 벗어나지 않게
        if (posInput.magnitude > 1.0f)
        {
            posInput = posInput.normalized;
        }
```

가상 조이스틱 만들기

```
            //Debug.Log(posInput.ToString());

            // 조이스틱 핸들 이동
            //imgJoystick.rectTransform.anchoredPosition = new Vector2(posInput.x, posInput.y);
            imgJoystick.rectTransform.anchoredPosition =
                new Vector2(posInput.x * imgJoystickBackground.rectTransform.sizeDelta.x / 2,
                            posInput.y * imgJoystickBackground.rectTransform.sizeDelta.y / 2);
        }
    }

    public void OnPointerDown(PointerEventData eventData)
    {

    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="eventData"></param>
    public void OnPointerUp(PointerEventData eventData)
    {
        posInput = Vector2.zero;
```

가상 조이스틱 만들기

```
            imgJoystick.rectTransform.anchoredPosition = posInput;
        }

        public float InputHorizontal
        {
            get
            {
                return posInput.x;
            }
        }

        public float InputVertical
        {
            get
            {
                return posInput.y;
            }
        }
    }
```