Multi-User Contents
(Chapter 8)

Jin-Mo Kim

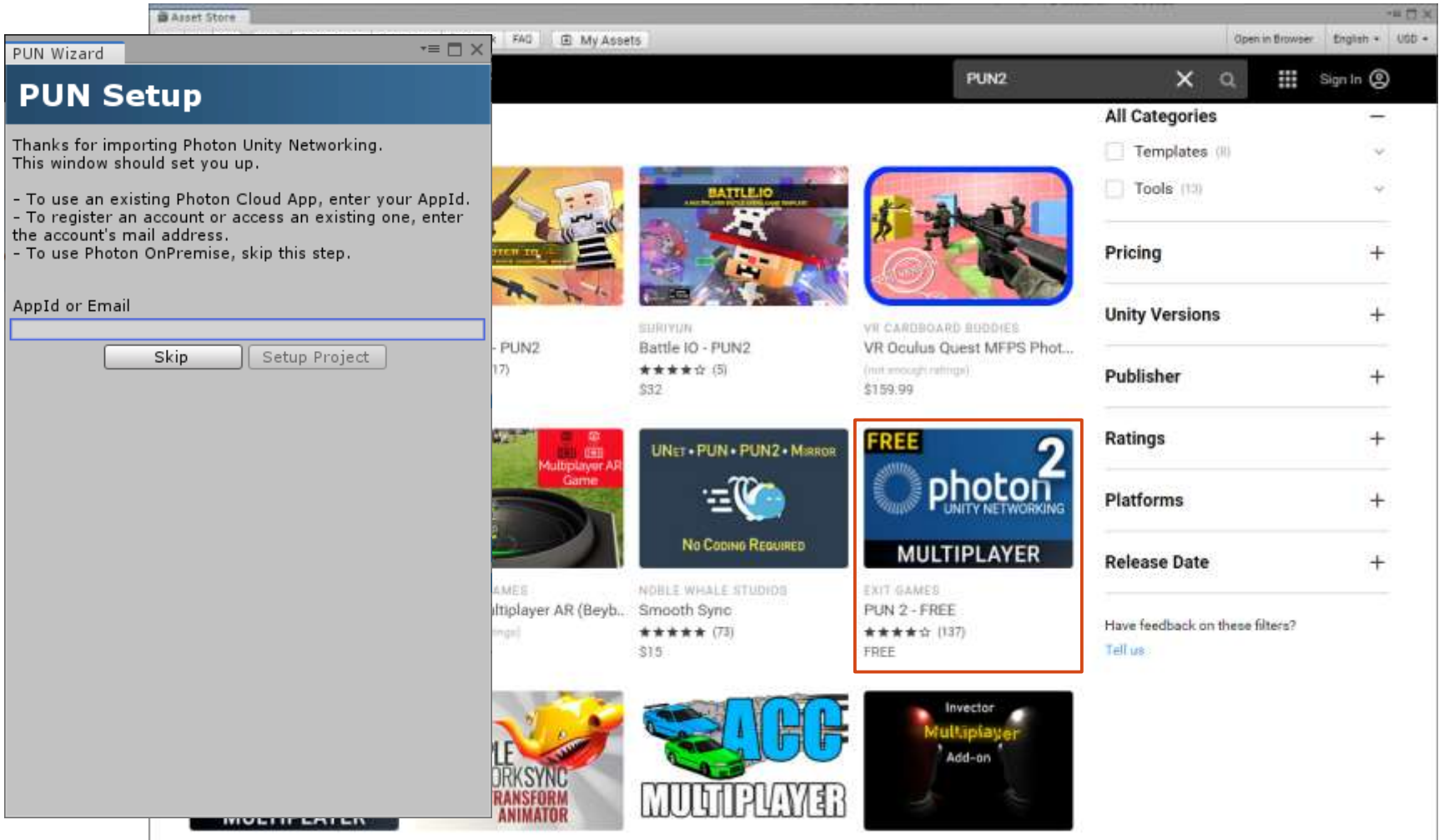
jinmo.kim@hansung.ac.kr

다중 사용자 체험을 위한 네트워크

- 2인 이상 사용자 참여가 가능한 프로젝트
  - Photon Unity Networking (PUN)
    - 멀티플레이어 콘텐츠 유니티 패키지 활용
    - 유연한 매치 메이킹
    - 플레이어와 객체들이 네트워크를 통해 동기화
      - RPC, PhotonView와 같은 속성들을 제공
    - 전용 Photon 서버를 통해 이루어지므로 클라이언트들은 1 대 1 연결이 필요 없는 이점

다중 사용자 체험을 위한 네트워크

- 프로젝트 생성 및 PUN 2 에셋 다운로드



다중 사용자 체험을 위한 네트워크

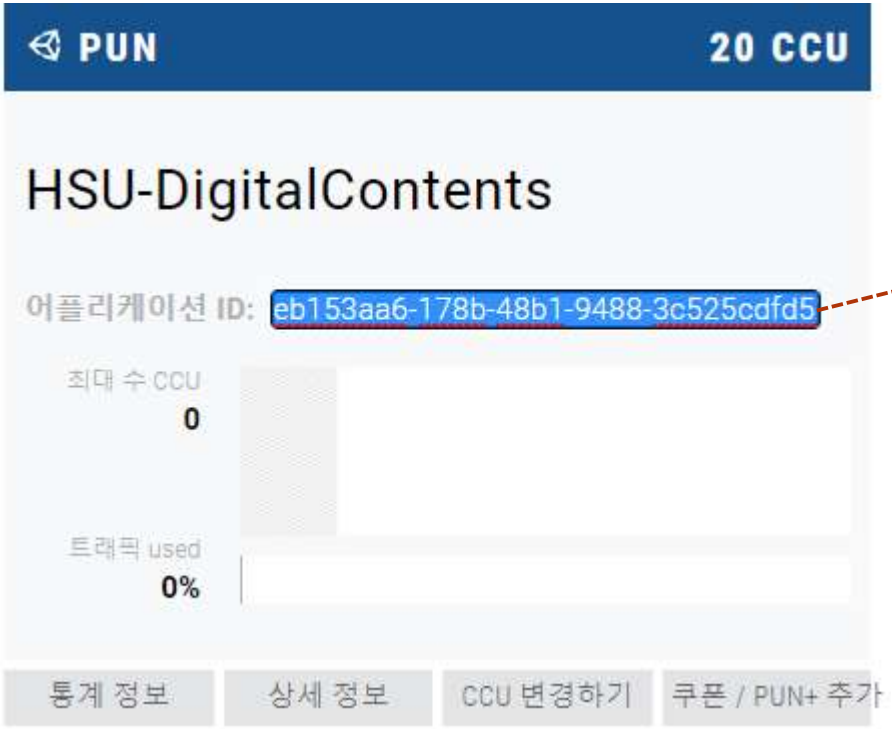
- Photon 엔진 등록 및 어플리케이션 활성화
  - https://www.photonengine.com/ko-KR/Photon
  - 회원 가입 후 로그인

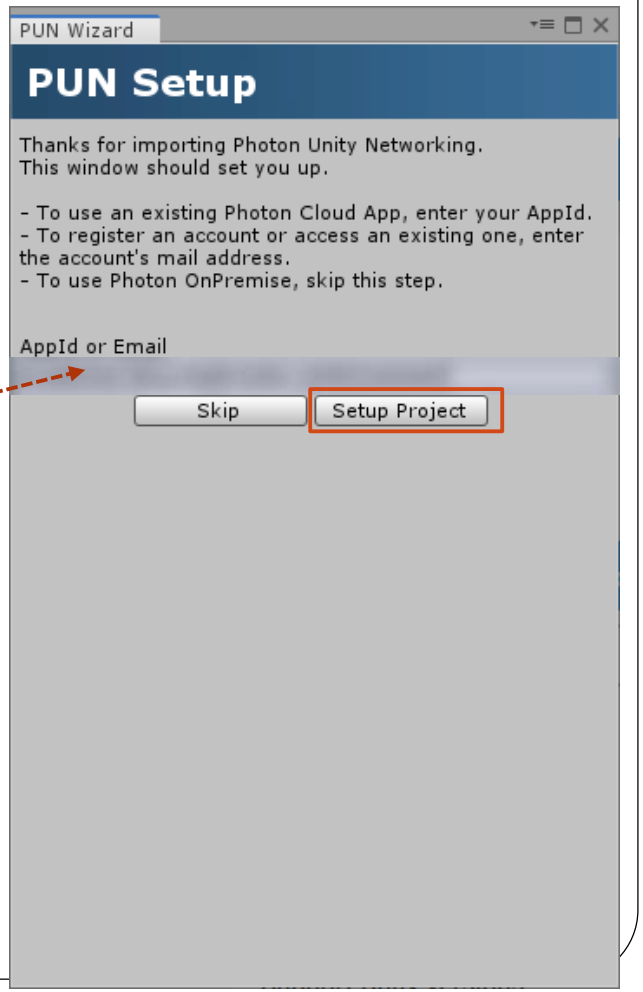
다중 사용자 체험을 위한 네트워크

- Photon 엔진 등록 및 어플리케이션 활성화
  - 새 어플리케이션 만들기
    - 종류: Photon PUN, 이름: 자유롭게 지정

다중 사용자 체험을 위한 네트워크

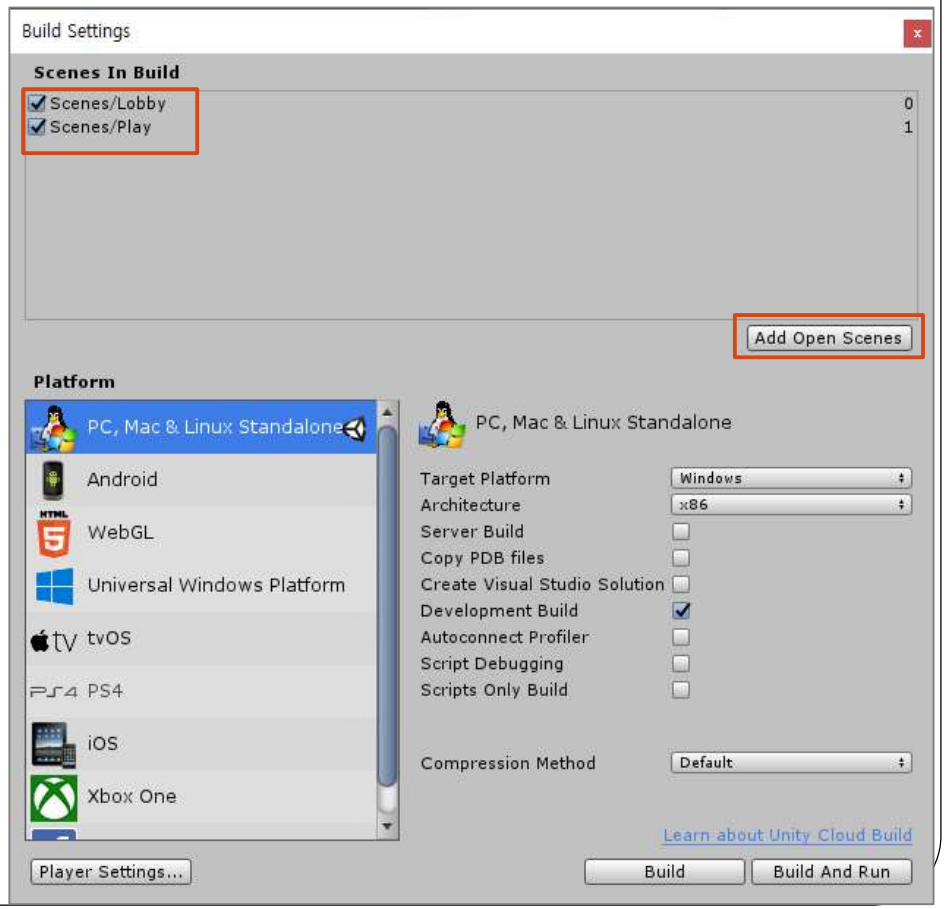
- Photon 엔진 등록 및 어플리케이션 활성화
  - 생성된 어플리케이션 → 어플리케이션 ID 선택 후 복사
  - Unity3D → PUN Wizard에 붙여넣기





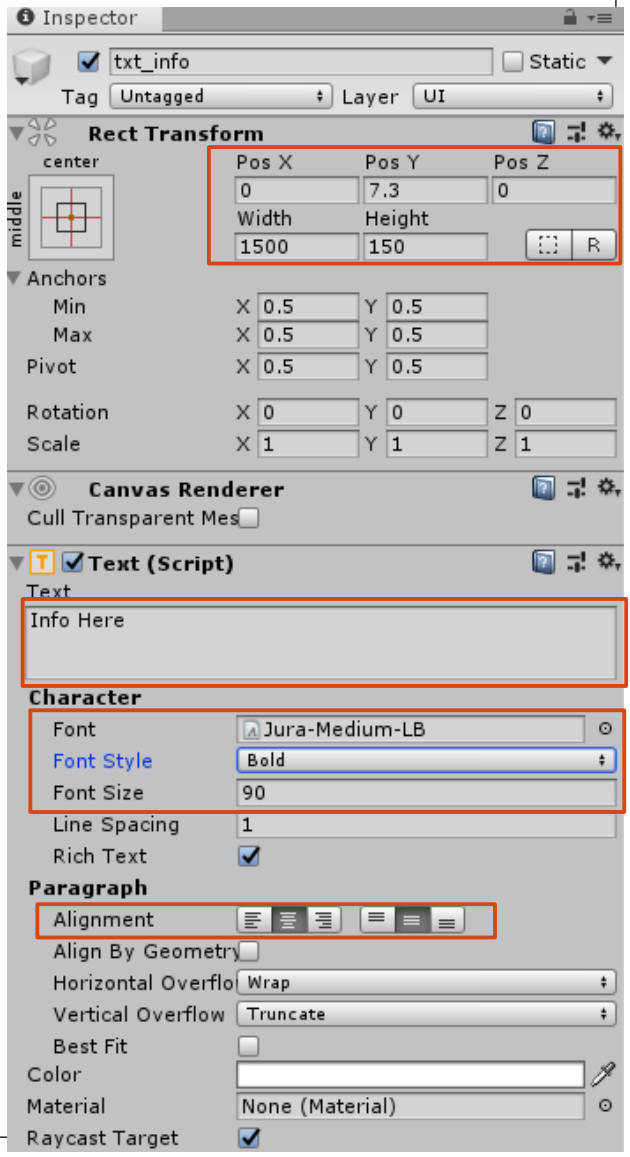
다중 사용자 체험을 위한 네트워크

- 룸 생성 및 참여를 위한 장면(Scene) 설정
  - File → New Scene → Save As → Lobby
  - File → New Scene → Save As → Play
  - File → Build Settings
    - Lobby, Play 장면을 각각 추가



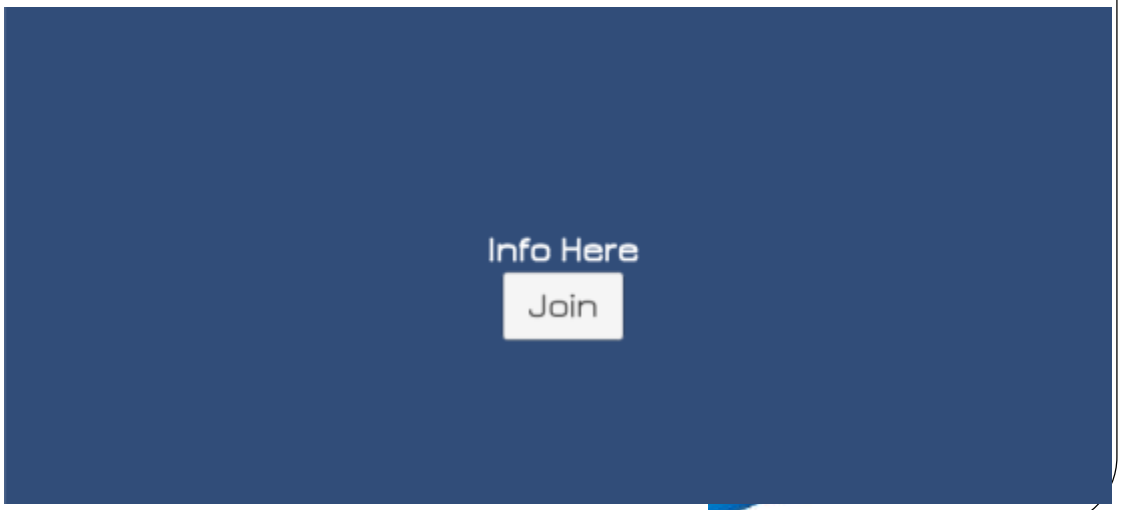
다중 사용자 체험을 위한 네트워크

- 룸 설정을 위한 Lobby 제작: Lobby scene
  - Main Camera → Clear Flags → Solid Color
  - 룸 입장과 정보 출력을 위한 UI 생성
    - Hierarchy → Create → Text
      - 이름: txt_info
      - (PosX, PosY, PosZ): (0, 7.3, 0)
      - (width, height): (1500, 150)
      - 중앙 정렬



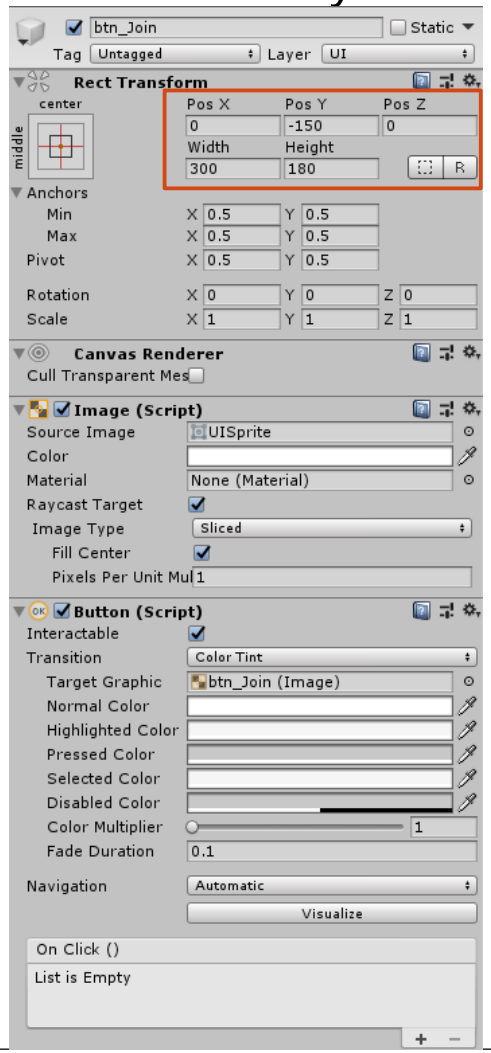
다중 사용자 체험을 위한 네트워크

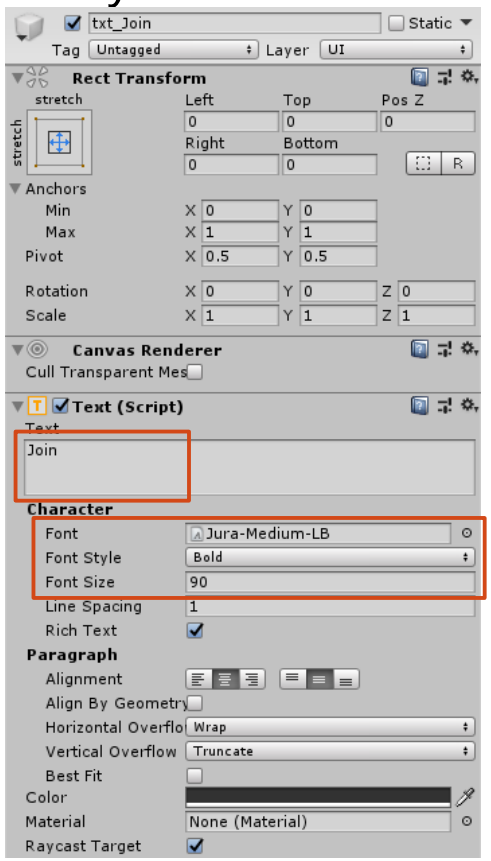
- 룸 설정을 위한 Lobby 제작: Lobby scene
  - Main Camera → Clear Flags → Solid Color
  - 룸 입장과 정보 출력을 위한 UI 생성
    - Hierarchy → Create → Button
      - 이름: btn_Join
      - (PosX, PosY, PosZ): (0, -150, 0)
      - (width, height): (300, 180)
    - Button → Text
      - 이름: txt_Join
      - Font, Font Size 변경



다중 사용자 체험을 위한 네트워크

- 룸 설정을 위한 Lobby 제작: Lobby scene





Digital Contents Design & Production

한성대학교 HANSUNG UNIVERSITY

다중 사용자 체험을 위한 네트워크

- 플레이 장면으로의 이동을 위한 룸 관리자
  - Hierarchy → Create → Create Empty
    - 이름: LobbyManager
  - Project → Create → Folder
    - 이름: Scripts
    - Project → Create → C# Script
      - 이름: cshLobbyManager
      - 스크립트를 LobbyManager 객체에 등록

다중 사용자 체험을 위한 네트워크

- 플레이 장면으로의 이동을 위한 룸 관리자

```
using Photon.Pun;       // Photon 의 여러 기능을 포함 라이브러리를 Unity에서 컴포넌트로 사용 가능
using Photon.Realtime;  // Realtime Network 게임 개발 c# 라이브러리
using UnityEngine.UI;

public class cshLobbyManager : MonoBehaviourPunCallbacks // PUN 구현할때 override 사용해 코드 작성해야됨
{
    private string gameVersion = "1"; // 같은 버전끼리 매칭하기 위해 string 사용 숫자뿐만 아닌 다른 것도 사용 가능

    public Text connectionInfoText;
    public Button joinButton;

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

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

    public override void OnConnectedToMaster()
    {
        joinButton.interactable = true;
        connectionInfoText.text = "온라인: Master 서버와 연결됨";
        //base.OnConnectedToMaster();
    }
```

다중 사용자 체험을 위한 네트워크

- 플레이 장면으로의 이동을 위한 룸 관리자

```
public override void OnDisconnected(DisconnectCause cause)
{
    joinButton.interactable = false;
    connectionInfoText.text = "오프라인: Master 서버와 연결되지 않음₩n접속 재시도 중...";

    PhotonNetwork.ConnectUsingSettings();
    //base.OnDisconnected(cause);
}

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

    // Master 서버에 접속 중이라면
    if (PhotonNetwork.IsConnected)
    {
        connectionInfoText.text = "Room에 접속...";
        PhotonNetwork.JoinRandomRoom();
    }
    else
    {
        connectionInfoText.text = "오프라인: Master 서버와 연결되지 않음₩n접속 재시도 중...";
        PhotonNetwork.ConnectUsingSettings();
    }
}
```

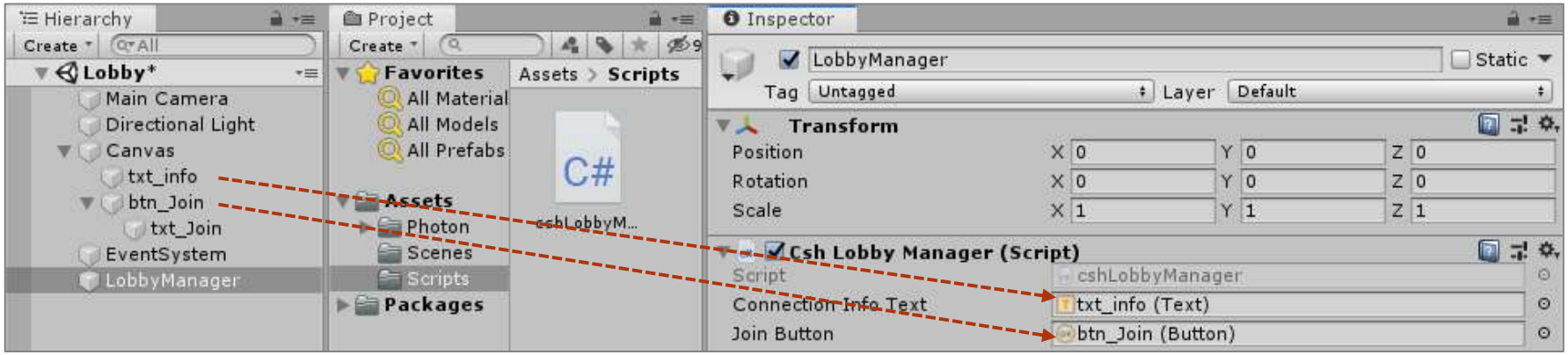
다중 사용자 체험을 위한 네트워크

- 플레이 장면으로의 이동을 위한 룸 관리자

```
    public override void OnJoinRandomFailed(short returnCode, string message)
    {
        connectionInfoText.text = "빈 방이 없음, 새로운 방 생성...";

        // 새로운 방을 만들며 (방의 Name, 방의 옵션 설정)
        PhotonNetwork.CreateRoom(null, new RoomOptions { MaxPlayers = 4 });

        //base.OnJoinRandomFailed(returnCode, message);
    }

    public override void OnJoinedRoom()
    {
        connectionInfoText.text = "방 참가 성공";
        // 모든 룸 참가자가 Main 씬을 로드하게 함
        // Unity 에서 제공하는 SceneMangaer.LoadScene() 은 이전 씬의 모든 게임 오브젝트를 삭제 및 네트워크 정보 유지가 되지 않음
        PhotonNetwork.LoadLevel("Play");

        //base.OnJoinedRoom();
    }

}
```

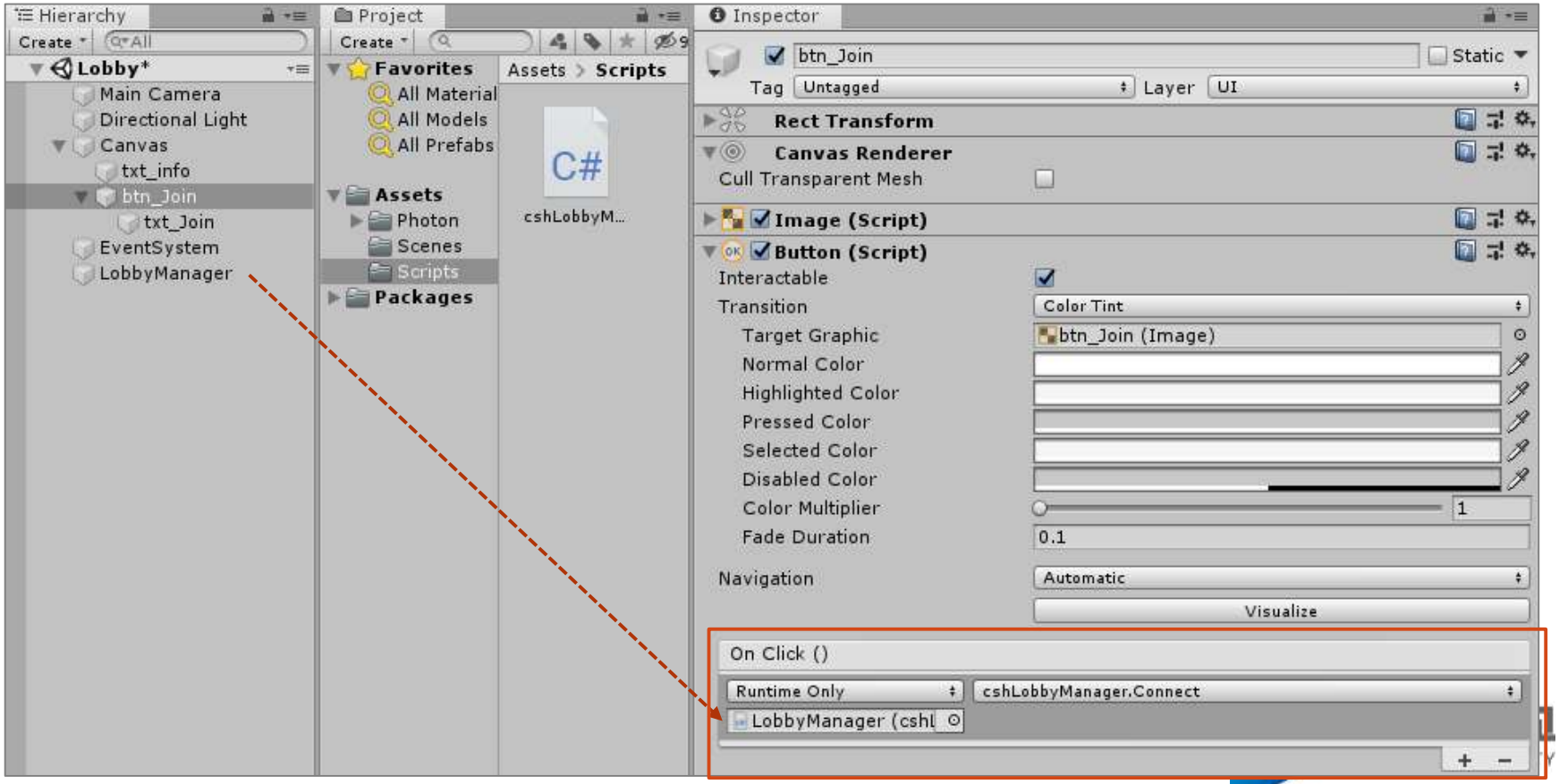
다중 사용자 체험을 위한 네트워크

- 플레이 장면으로의 이동을 위한 룸 관리자
  - LobbyManager에 변수 등록



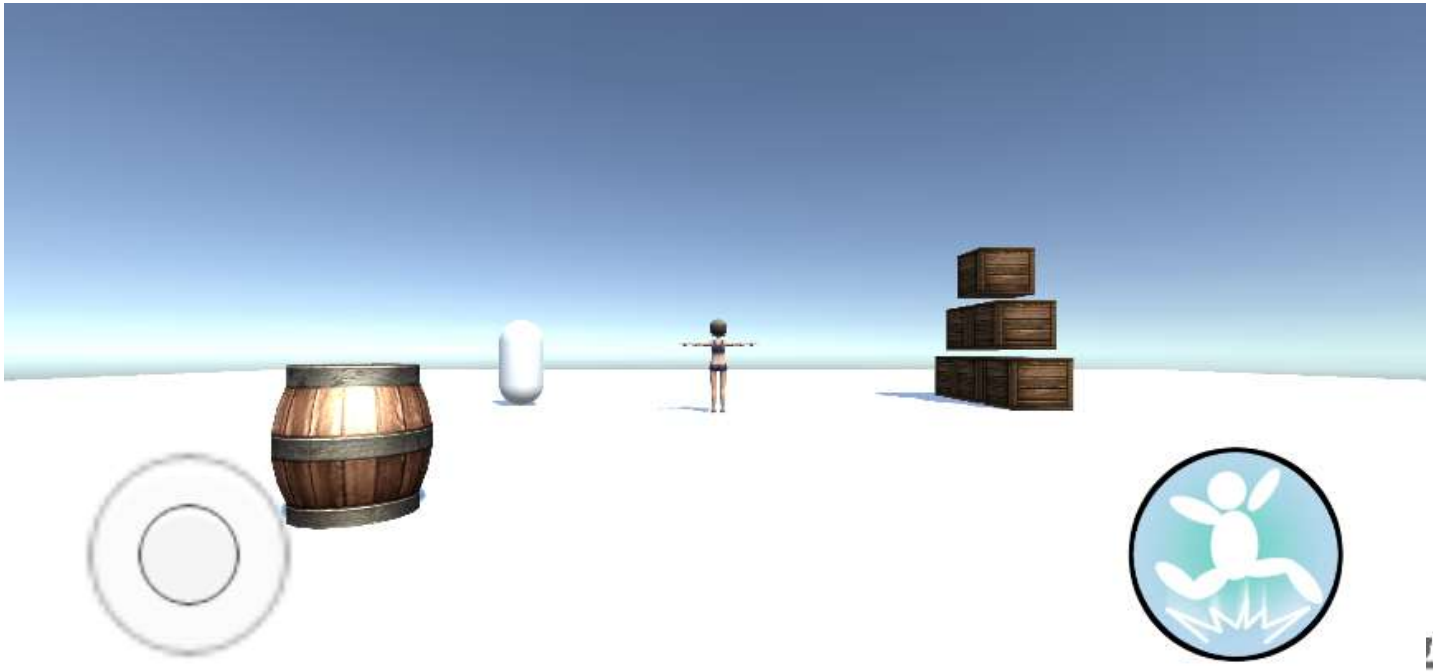
다중 사용자 체험을 위한 네트워크

- 플레이 장면으로의 이동을 위한 룸 관리자
  - 룸 입장 버튼 이벤트 설정
    - btn_Join 버튼에 클릭 이벤트 추가



다중 사용자 체험을 위한 네트워크

- 콘텐츠가 진행되는 Play 장면 구성
  - 템플릿에서 제공하고 있는 Play 장면을 활용 및 수정
  - 플레이어(Fighter)와 UI를 하나의 움직임으로 처리하기 위하여 UI를 플레이어의 자식 객체로 등록

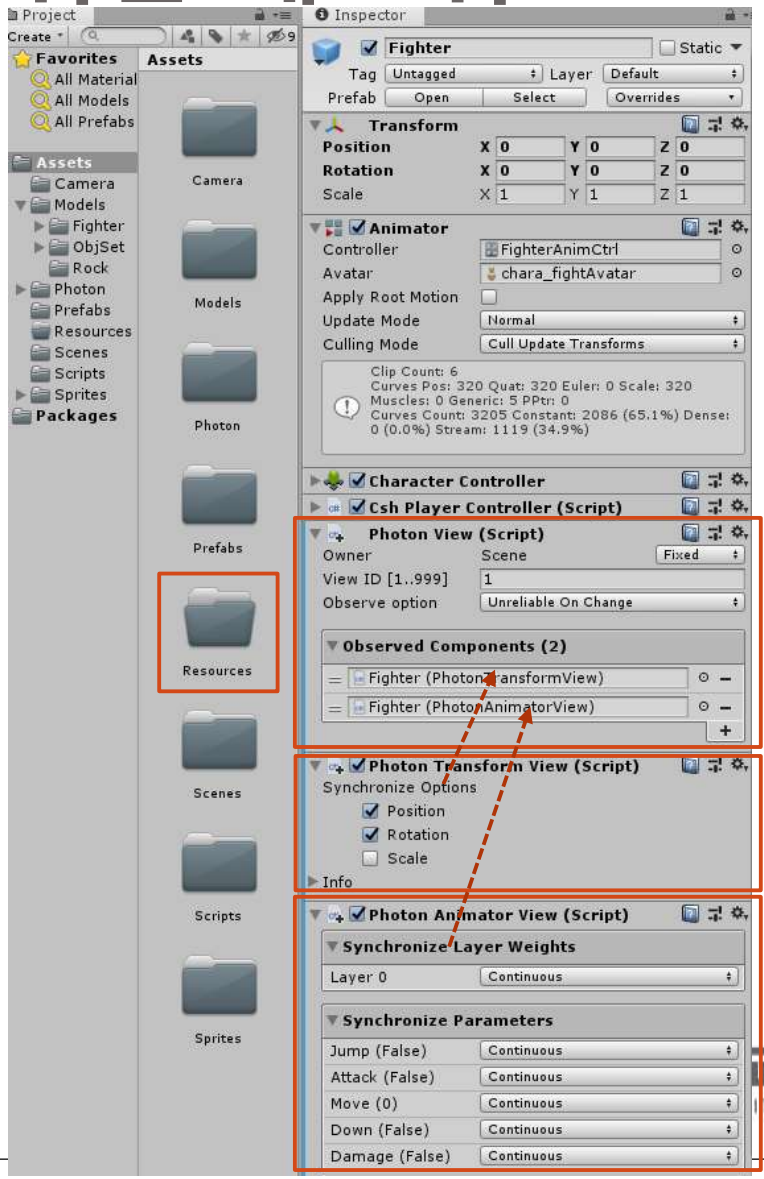


다중 사용자 체험을 위한 네트워크

- 콘텐츠가 진행되는 Play 장면 구성
  - 다중 사용자 참여를 위한 캐릭터 수정
    - Fighter 객체
      - Photon View, Photon Transform View, Photon Animator View
        - 속성 각각 추가
        - 속성 수정 및 등록
      - Project → Create → Folder
        - 이름: Resources
        - 네트워크를 통해 사용자가 룸에 접속하는 순간 생성되는 캐릭터의 복제(instantiate)를 위해서 프리팹은 Resources 폴더에 등록되어 있어야 함
        - Resources로 이름 되어 있는 폴더는 외부로 접근 가능한 폴더의 기능을 가짐

다중 사용자 체험을 위한 네트워크

- 콘텐츠가 진행되는 Play 장면 구성



다중 사용자 체험을 위한 네트워크

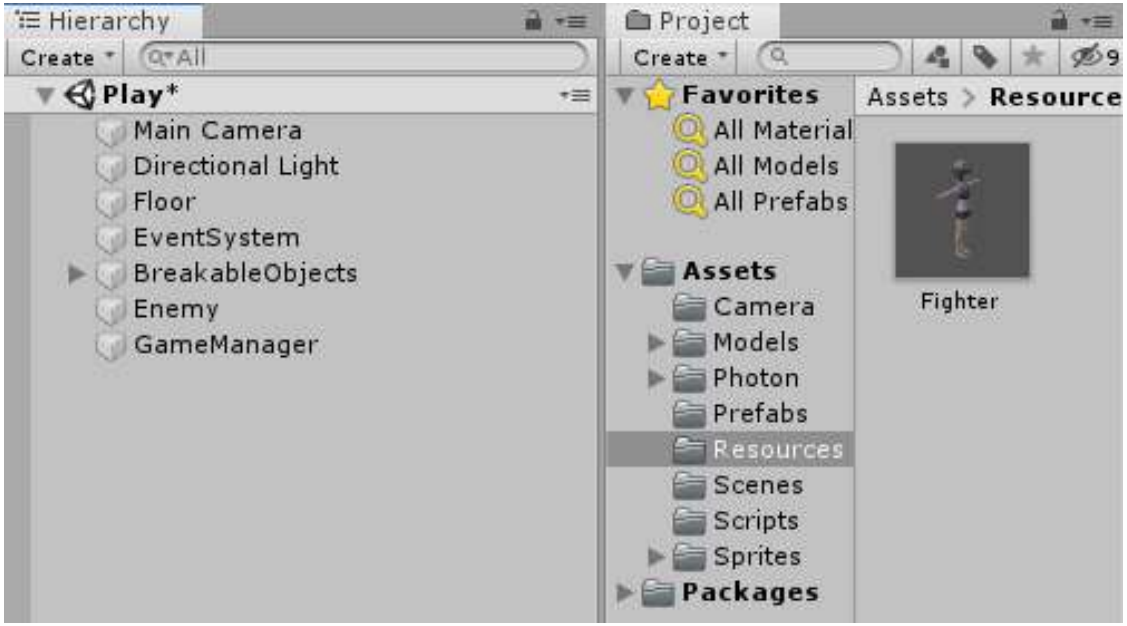
- 콘텐츠가 진행되는 Play 장면 구성
  - 사용자만 제어할 수 있는 독립된 캐릭터 설정을 위한 스크립트 수정
    - Fighter → cshPlayerController 스크립트

```
using Photon.Pun;

public class cshPlayerController : MonoBehaviourPun

 void Update()
 {
     if (!photonView.IsMine)
     {
         m_UI.SetActive(false);
         return;
     }
     ...
 }
```

추가

다중 사용자 체험을 위한 네트워크

- 콘텐츠가 진행되는 Play 장면 구성
  - Fighter 캐릭터를 Resources 폴더의 프리팹으로 등록
    - 등록 후 Hierarchy의 Fighter 캐릭터는 삭제



다중 사용자 체험을 위한 네트워크

- 콘텐츠가 진행되는 Play 장면 구성
  - 룸에 접속 시 사용자의 캐릭터를 생성시키는 게임 매니저 구현
  - Hierarchy → Create → Create Empty
    - 이름: GameManager
    - Photon View, Photon Transform View 속성 추가
      - 다중 사용자가 존재하는 룸 안에서 게임 관리자는 한 명만 설정하기 위해 추가
    - Hierarchy → Create → Create Empty
      - 이름: PlayerSpawnPos
      - 룸 접속 시 캐릭터가 생성될 초기 위치
  - Project → Create → C# Script
    - 이름: cshGameManager
      - 룸 접속 시 지정된 위치에 사용자가 제어 가능한 캐릭터 생성

다중 사용자 체험을 위한 네트워크

- 콘텐츠가 진행되는 Play 장면 구성

```
using Photon.Pun;
using UnityEngine.SceneManagement;

public class cshGameManager : MonoBehaviourPun // 점수와 게임 오버 여부 및 게임 UI를 관리하는 게임 매니저 스크립트
{
    public static cshGameManager instance // 외부에서 싱글톤 오브젝트를 가져올때 사용할 프로퍼티
    {
        get
        {
            // 만약 싱글톤 변수에 아직 오브젝트가 할당되지 않았다면
            if (m_instance == null)
            {
                // 씬에서 GameManager 오브젝트를 찾아 할당
                m_instance = FindObjectOfType<cshGameManager>();
            }

            // 싱글톤 오브젝트를 반환
            return m_instance;
        }
    }

    private static cshGameManager m_instance; // 싱글톤이 할당될 static 변수

    public GameObject PlayerPrefab; // 생성할 VR 플레이어 캐릭터
    public GameObject SpawnPosPrefab; // 생성할 VR 플레이어 캐릭터의 위치
```

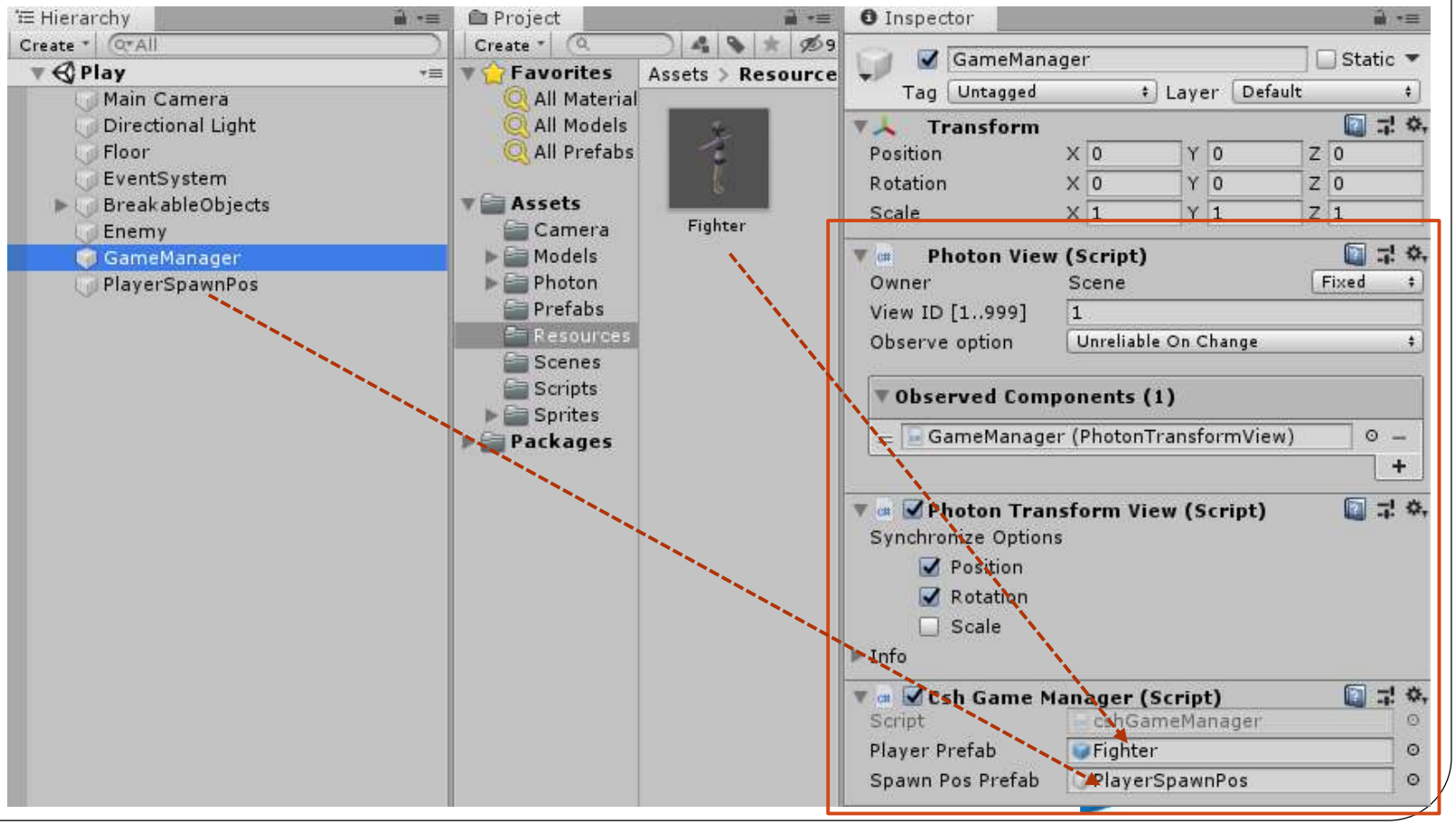
다중 사용자 체험을 위한 네트워크

- 콘텐츠가 진행되는 Play 장면 구성

```
    private void Awake()
    {
        // 씬에 싱글톤 오브젝트가 된 다른 GameManager 오브젝트가 있다면
        if (instance != this)
        {
            // 자신을 파괴
            Destroy(gameObject);
        }
    }

    private void Start()
    {
        // 생성할 랜덤 위치 지정
        Vector3 randomSpawnPos = SpawnPosPrefab.transform.position;//Random.insideUnitSphere * 5f;
        // 네트워크상의 모든 클라이언트에서 생성 실행
        // 해당 게임 오브젝트의 주도권은 생성 메서드를 직접 실행한 클라이언트에 있음
        PhotonNetwork.Instantiate(PlayerPrefab.name, randomSpawnPos, Quaternion.identity);
    }
}
```

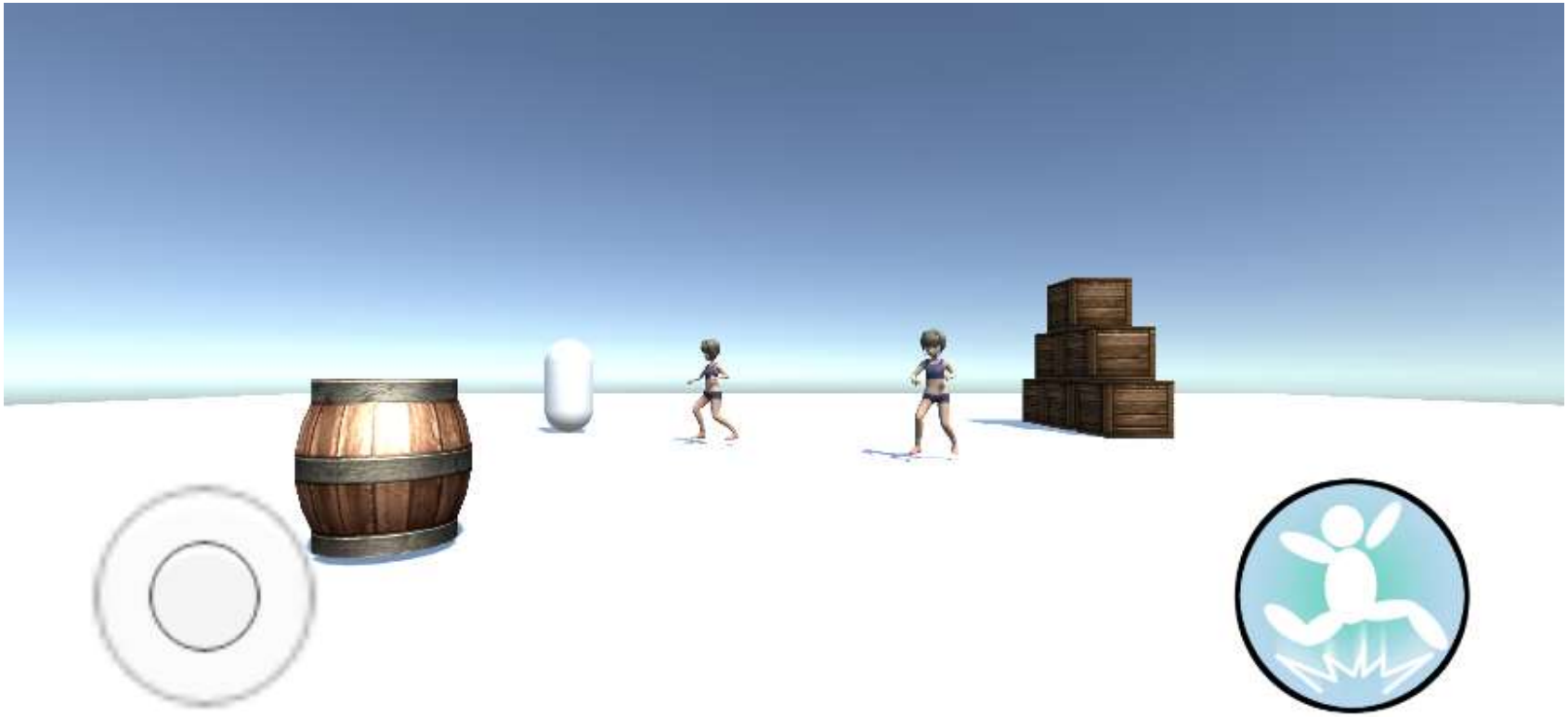
다중 사용자 체험을 위한 네트워크

- 콘텐츠가 진행되는 Play 장면 구성
  - 구현된 스크립트를 GameManager 객체에 등록
    - 변수 설정



다중 사용자 체험을 위한 네트워크

- 콘텐츠가 진행되는 Play 장면 구성
  - Lobby 장면에서 콘텐츠를 실행하여 룸으로 접속

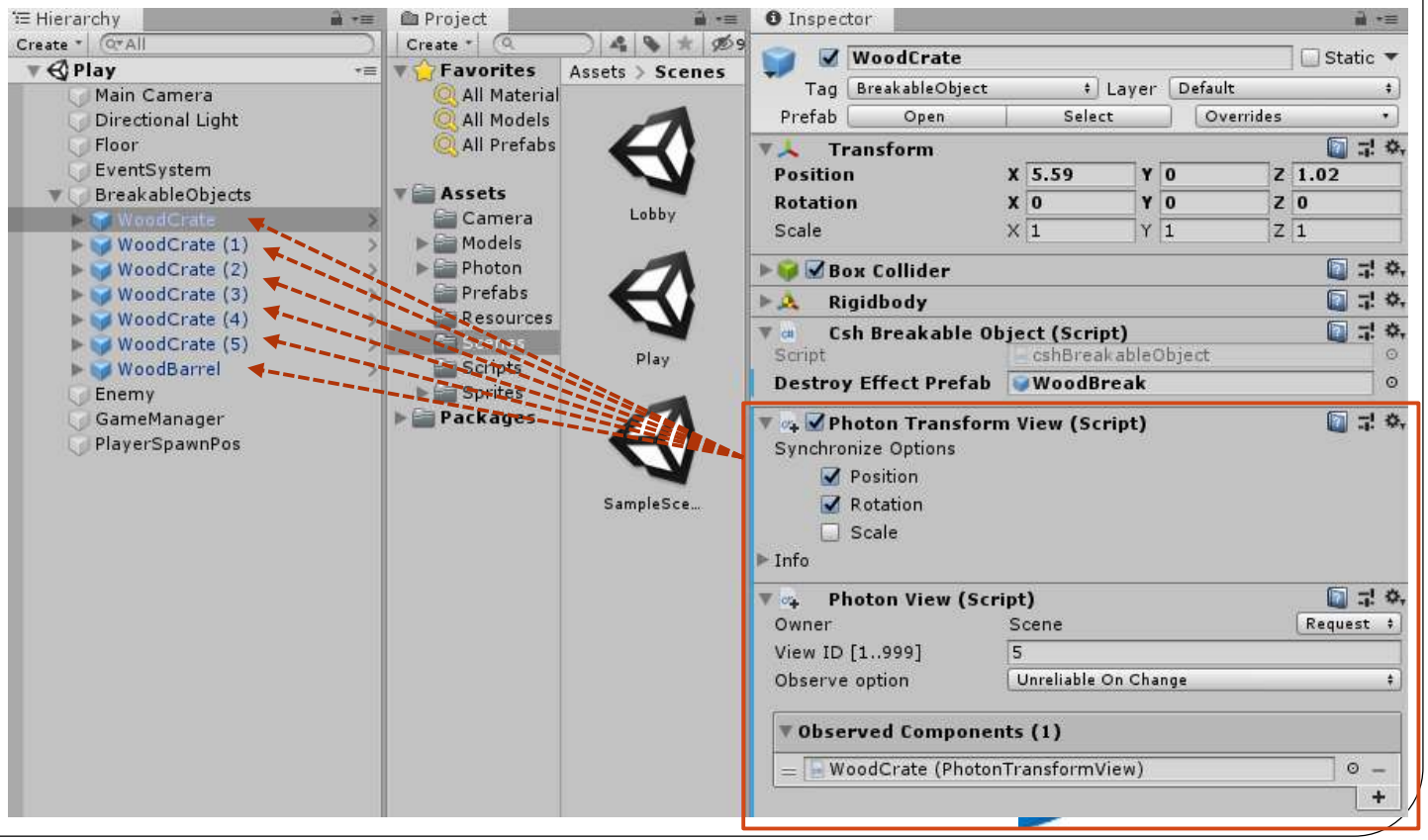


다중 사용자 체험을 위한 네트워크

- RPC를 활용한 동기화 구현
  - RPC: Remote Procedure Call 원격 프로시저 호출
    - 네트워크 사용자에게 현재 내가 수행한 결과를 원격으로 호출하거나 받을 수 있는 기능
  - BreakableObjects 하위의 객체들이 제거될 때 동기화되지 않는 문제
    - 나의 화면에서는 사라지지만 다른 클라이언트 화면에는 남아 있는 상황
    - 객체 제거 시 파티클이 나의 화면에만 나타나는 상황
    - 이런 문제를 해결하기 위한 방법
      - PhotonView 속성을 통한 동기화 객체 설정
      - RPC를 통한 상태 정보를 다른 클라이언트와 공유

다중 사용자 체험을 위한 네트워크

- RPC를 활용한 동기화 구현
  - BreakableObjects 하위 객체들 → PhotonView 속성 추가
    - 외부 접근이 가능하도록 속성 설정
      - Owner: Request



다중 사용자 체험을 위한 네트워크

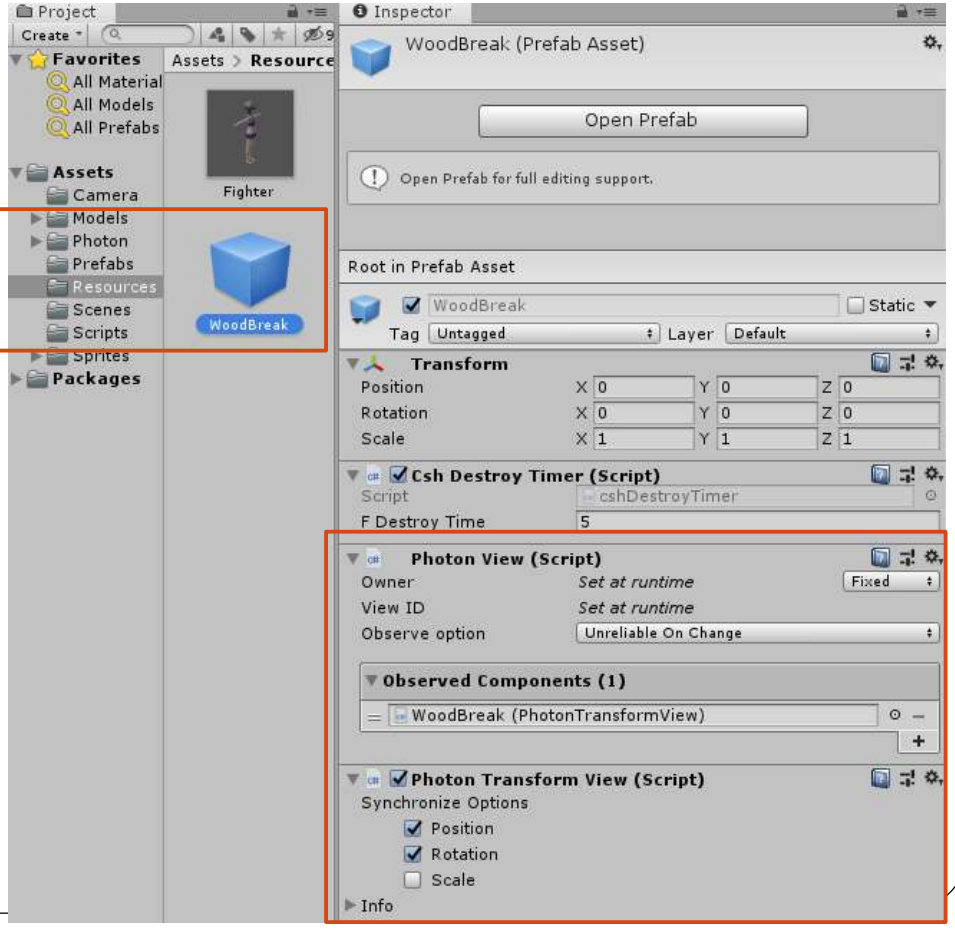
- RPC를 활용한 동기화 구현
  - 스크립트 수정
    - cshPlayerController.cs

```
public void OnVirtualPadAttack()
{
    for (int i = 0; i < m_attackArea.colliders.Count; ++i)
    {
        ...
        var enemy = collider.GetComponent<cshEnemyController>();
        if (enemy != null) {
            enemy.Damage();
            if(enemy.GetHP() <= 0) m_attackArea.colliders.Clear();
        }
        else {
            int id = collider.gameObject.GetComponent<PhotonView>().ViewID;
            photonView.RPC("RPCDestroy", RpcTarget.All, id);
            //Destroy(collider.gameObject);
        }
    }
    if(cntBreak > 0) m_attackArea.colliders.Clear();


    center /= cnt;
    center.y = transform.localPosition.y;
    transform.LookAt(center);

}
```

```
[PunRPC]
public void RPCDestroy(int id)
{
    Destroy(PhotonView.Find(id).gameObject);
}
```

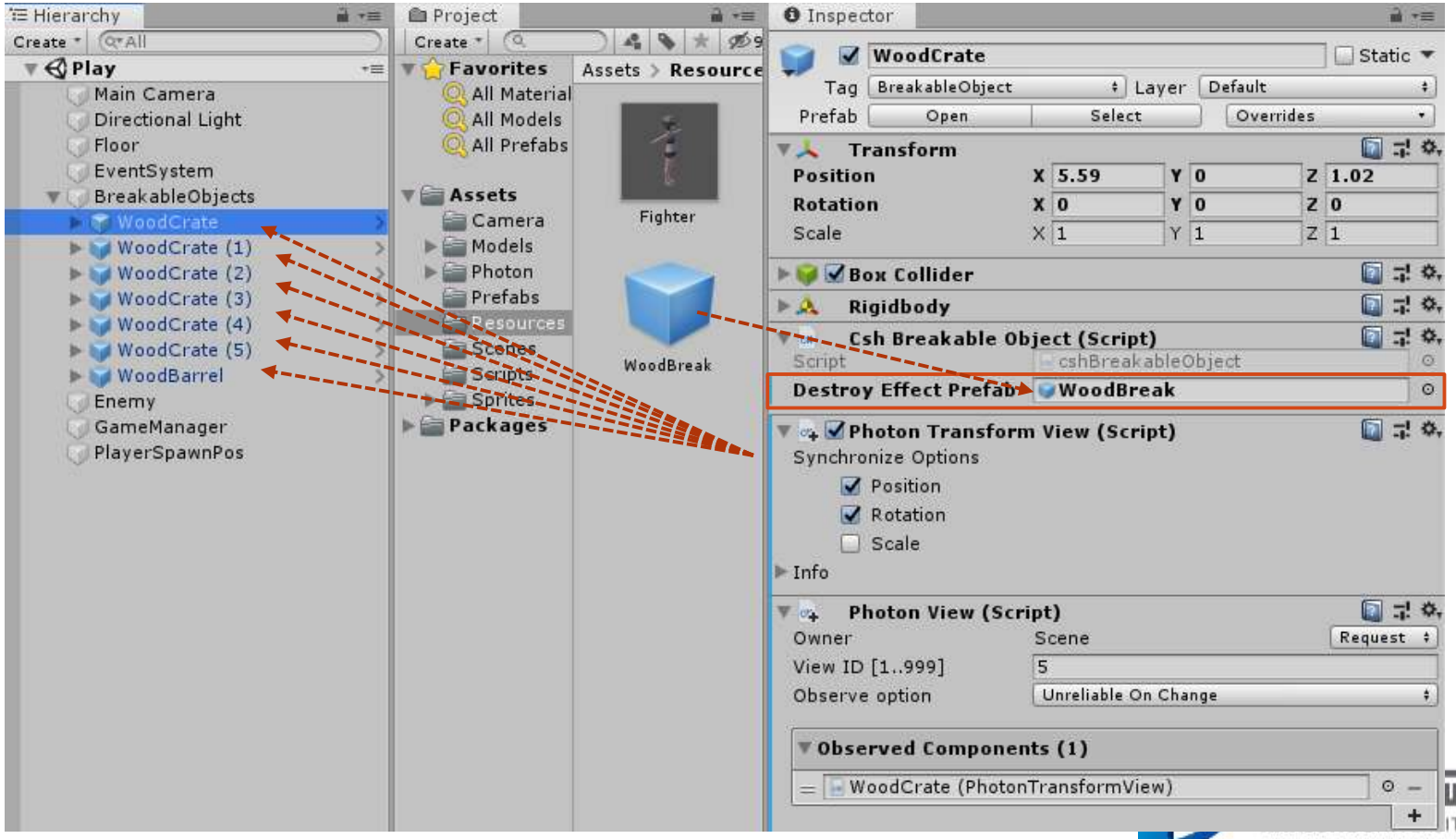
다중 사용자 체험을 위한 네트워크

- RPC를 활용한 동기화 구현
  - 객체 제거 시 생성되는 파티클에 대한 동기화 처리
    - PhotonView 속성 추가 후 Resources 폴더에 프리팹 등록



다중 사용자 체험을 위한 네트워크

- RPC를 활용한 동기화 구현
  - BreakableObjects 하위 객체들 → 파티클 객체 변경



다중 사용자 체험을 위한 네트워크

- RPC를 활용한 동기화 구현
  - 스크립트 수정
    - cshBreakableObject.cs

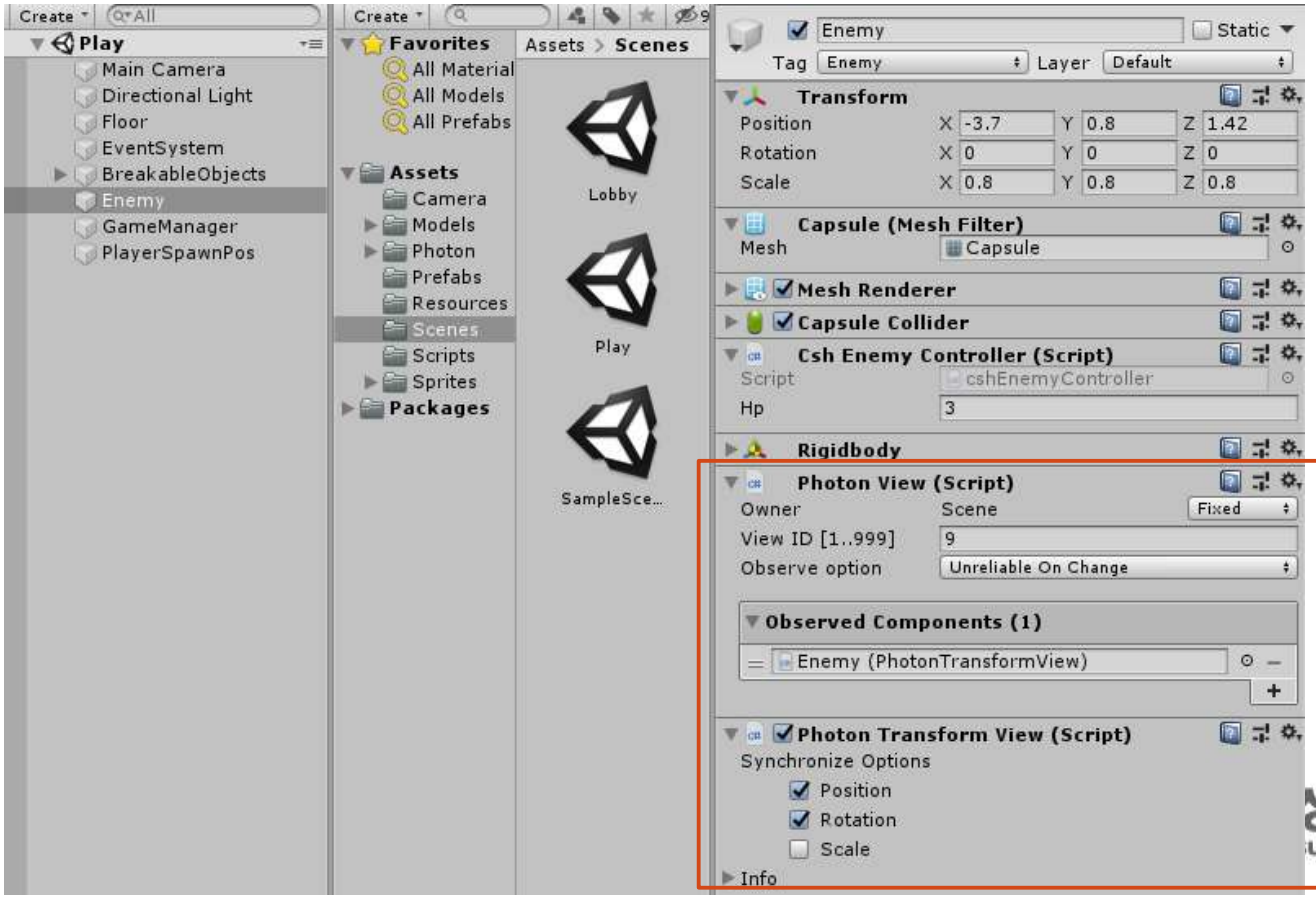
```
using Photon.Pun;

public class cshBreakableObject : MonoBehaviourPun
{
    public GameObject destroyEffectPrefab;

    public void PlayEffect()
    {
        //Instantiate(destroyEffectPrefab, transform.localPosition, Quaternion.identity);
        PhotonNetwork.Instantiate(destroyEffectPrefab.name, transform.localPosition, Quaternion.identity);
    }
}
```

다중 사용자 체험을 위한 네트워크

- RPC를 활용한 동기화 구현
  - 적 객체에 대한 동기화 처리
  - 문제점
    - 사용자의 관점에서 Enemy의 HP가 1씩 줄어드는 걸 확인할 수 있지만, 다른 사용자에게는 반영되지 않음
    - 즉, 사용자의 실행 결과가 다른 사용자에게 알려지지 않았기 때문에 이를 위한 처리과정이 필요 → RPC

다중 사용자 체험을 위한 네트워크

- RPC를 활용한 동기화 구현
  - Enemy 객체에 네트워크 속성 추가
    - PhotonView



다중 사용자 체험을 위한 네트워크

- RPC를 활용한 동기화 구현
  - 스크립트 수정: RPC 코드 추가
    - cshEnemyController.cs

```
using Photon.Pun;

public class cshEnemyController : MonoBehaviourPun
{
    ...
    [PunRPC]
    public void RPCDamage(int attack)
    {
        hp = hp - attack;
        if (hp <= 0)
        {
            Destroy(gameObject);
        }
    }
    ...
}
```

다중 사용자 체험을 위한 네트워크

- RPC를 활용한 동기화 구현
  - cshPlayerController 스크립트에 RPC 호출 수정
    - RPC 호출을 통한 동기화로 모든 사용자가 같은 결과를 공유

```
if (enemy != null) {
    //enemy.Damage();
    PhotonView pv = enemy.gameObject.GetComponent<PhotonView>();
    pv.RPC("RPCDamage", RpcTarget.All, 1);

    if (enemy.GetHP() <= 0) m_attackArea.colliders.Clear();
}
```