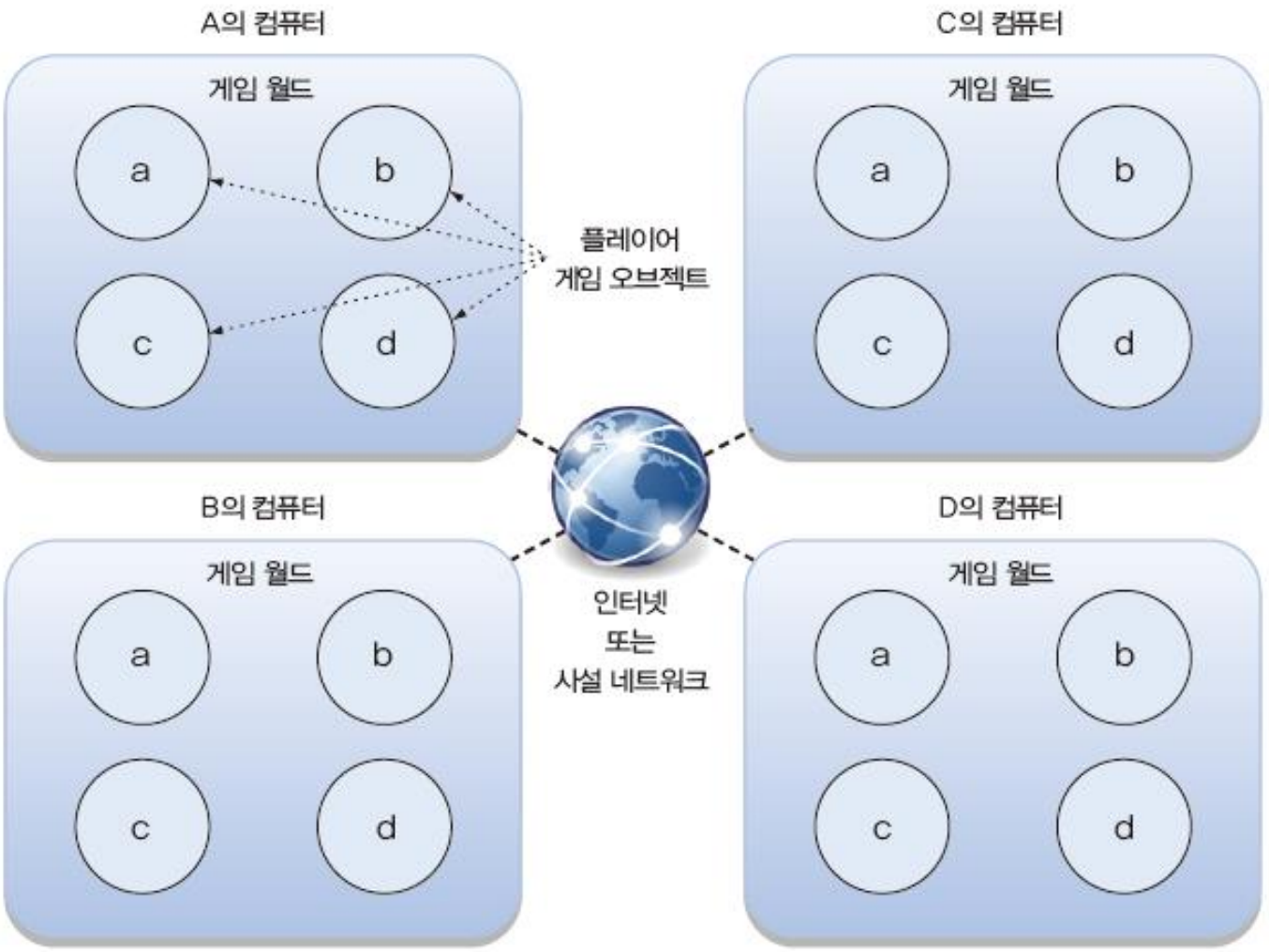
좀비 서바이버 멀티플레이어 : 네트워크 이론과 로비 구현

네트워크 동기화(1)

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



네트워크 동기화(2)

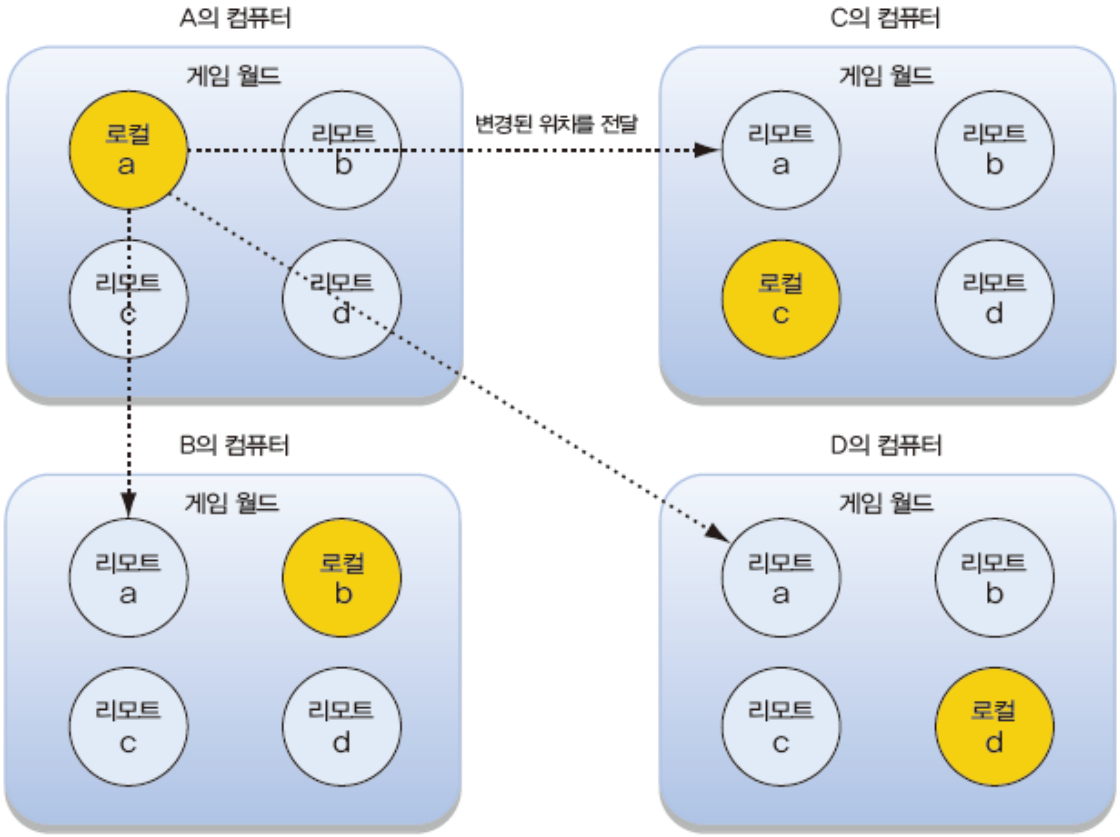
로컬과 리모트

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

네트워크 동기화(3)

동기화

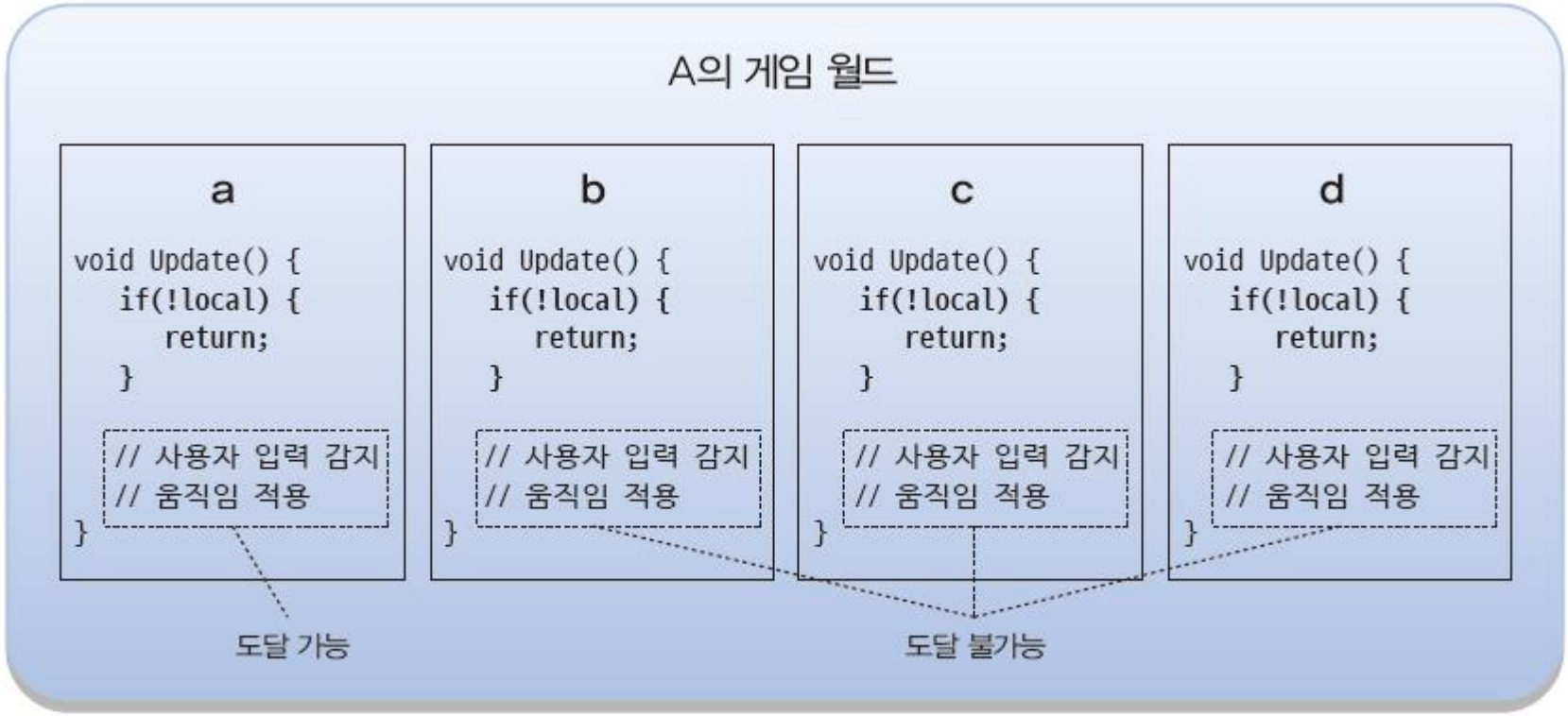
- 로컬 플레이어 캐릭터 a의 정보를 다른 월드에 있는 리모트 플레이어 캐릭터 a에 전달하여 로컬 플레이어 캐릭터 a의 변경 사항을 리모트 플레이어 캐릭터 a에 반영
- 이러한 방식으로 4개의 게임 월드를 끊임없이 같은 모습으로 동기화



네트워크 동기화(4)

로컬 권한 검사

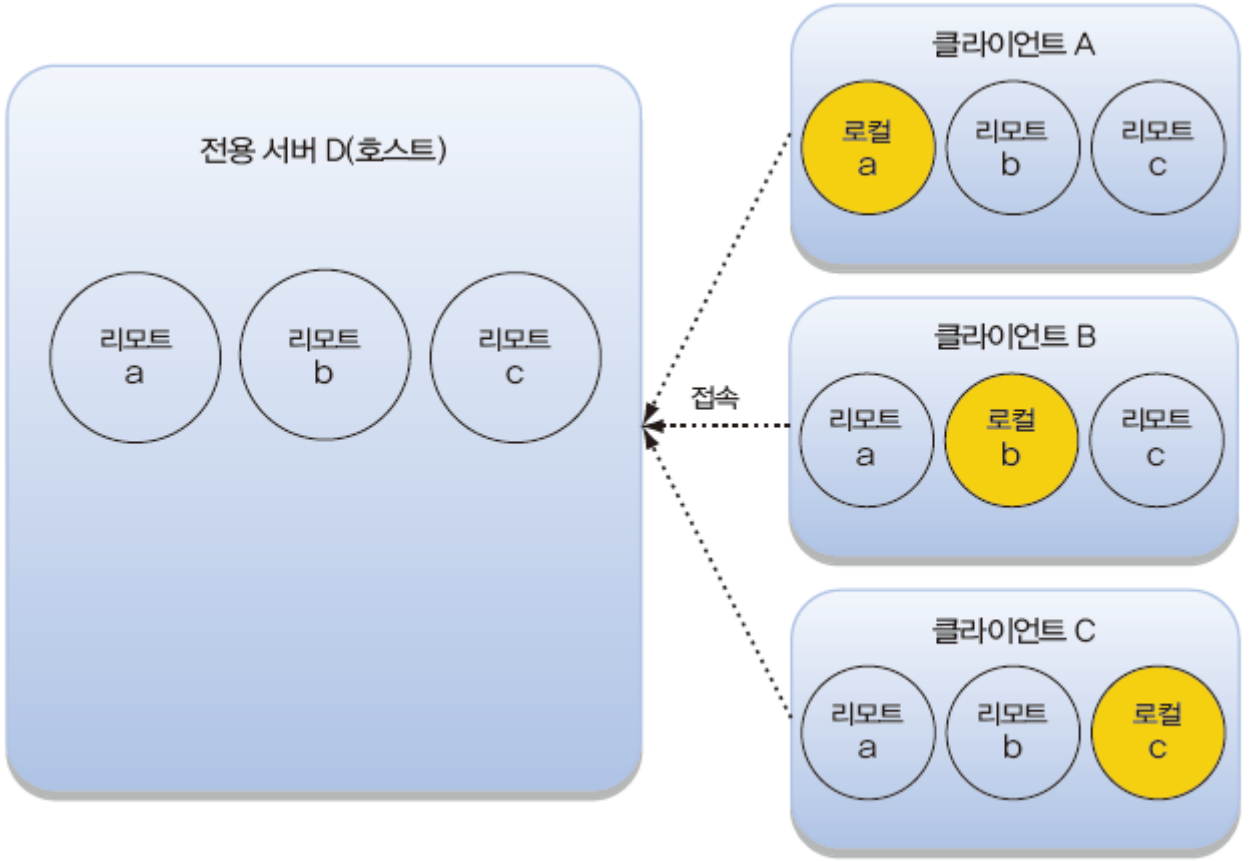
- 로컬과 리모트 플레이어 캐릭터 모두 사용자 입력을 받을 수 있음
- 오브젝트가 로컬 권한을 가지고 있는지 검사
  - 코드마다 if 문을 삽입하여 오브젝트 자신이 로컬 오브젝트라면 사용자 입력을 그대로 받고, 리모트라면 사용자 입력을 무시



게임 서버의 종류(1)

전용 서버

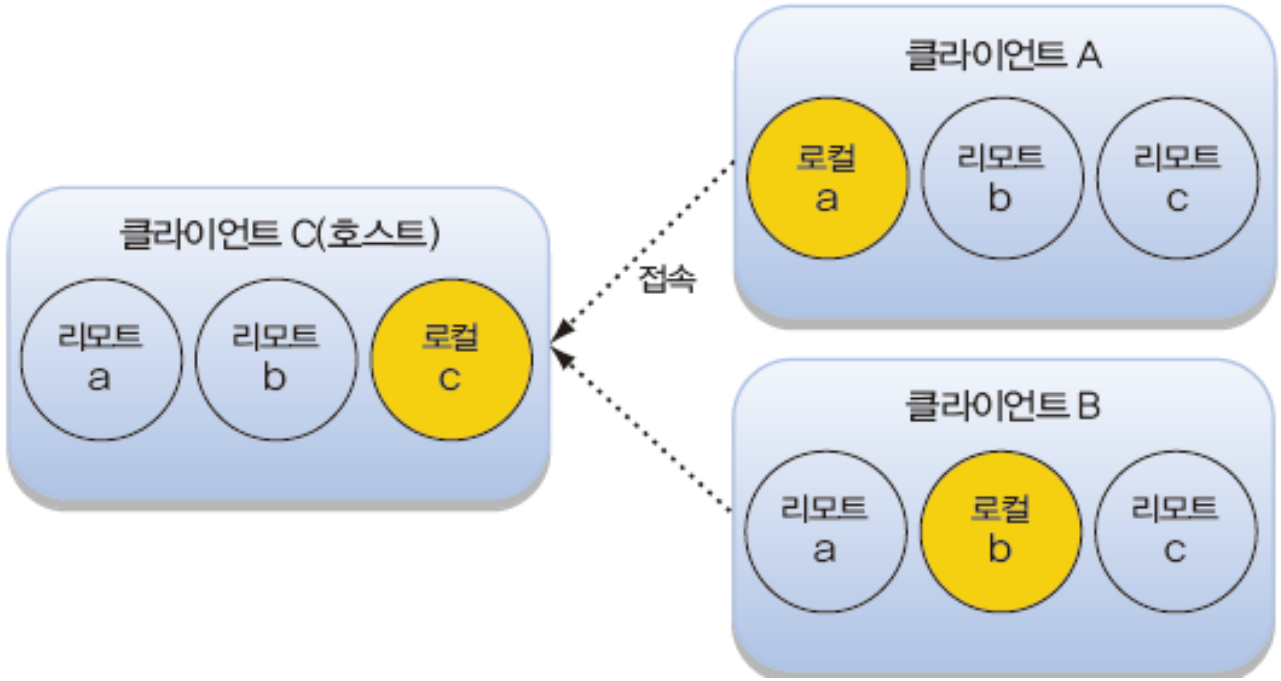
- 서버의 모든 자원이 온전히 네트워크 서비스를 유지하는 데 사용되며, 서버가 플레이어로서 게임에 직접 참가하지 않는 형태



게임 서버의 종류(2)

리슨 서버

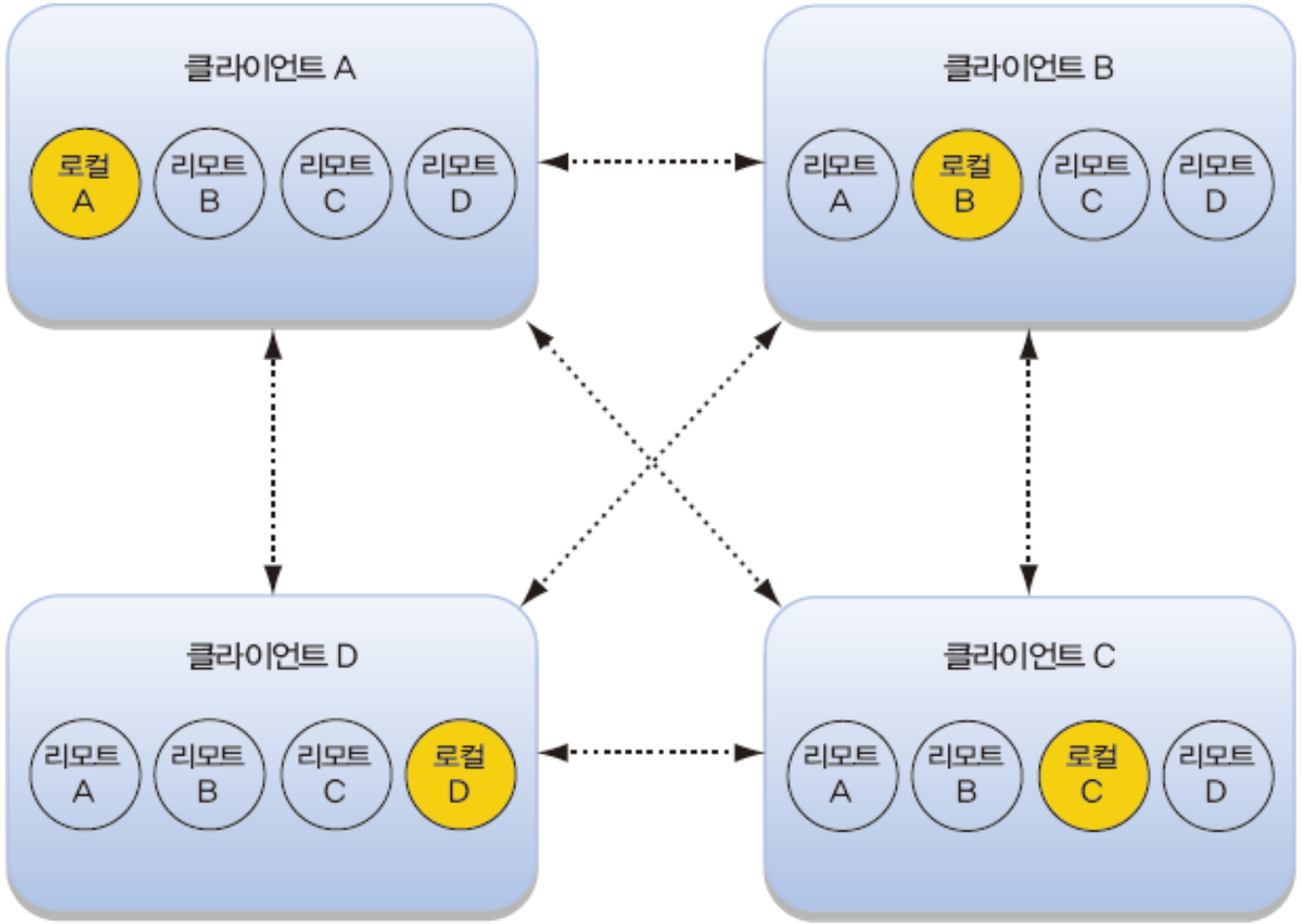
- 리슨 서버 방식은 전용 서버가 없는 대신 플레이어 클라이언트 중 하나가 서버 역할을 맡음



게임 서버의 종류(3)

P2P

- P2P 방식은 게임에 참가한 클라이언들 모두가 호스트 역할을 겸함
- 클라이언트들이 특정 단일 호스트에 참가하는 방식이 아니라 서로 직접 연결된 형태



게임 서버의 종류(4)

매치메이킹 서버

- 좀비 서바이버 멀티플레이어는 리슨 서버 방식을 사용하여 구현하겠음
  - 다만, 매치메이킹 과정에서는 포톤(Photon)에서 제공하는 전용 클라우드 서버를 사용
- 리슨 서버나 P2P 방식을 사용하더라도 참가할 클라이언트들이 서로를 찾아 방 하나에 모이는 과정에서 사용할 전용 서버가 필요 - 이러한 전용 서버를 매치메이킹 서버라고 함
- MMO RPG 게임에서는 대부분의 요소를 전용 서버로 처리
- 반면 플레이어들이 한 룸(세션)에 모여 레이드나 팀 데스매치를 진행하는 종류의 게임에서는 매치메이킹은 전용 서버를 사용하지만, 룸이 구성되고 라운드를 시작할 때는 플레이어 중 한 명을 호스트로 삼는 리슨 서버 방식을 많이 사용

게임 서버의 종류(5)

포톤 룸

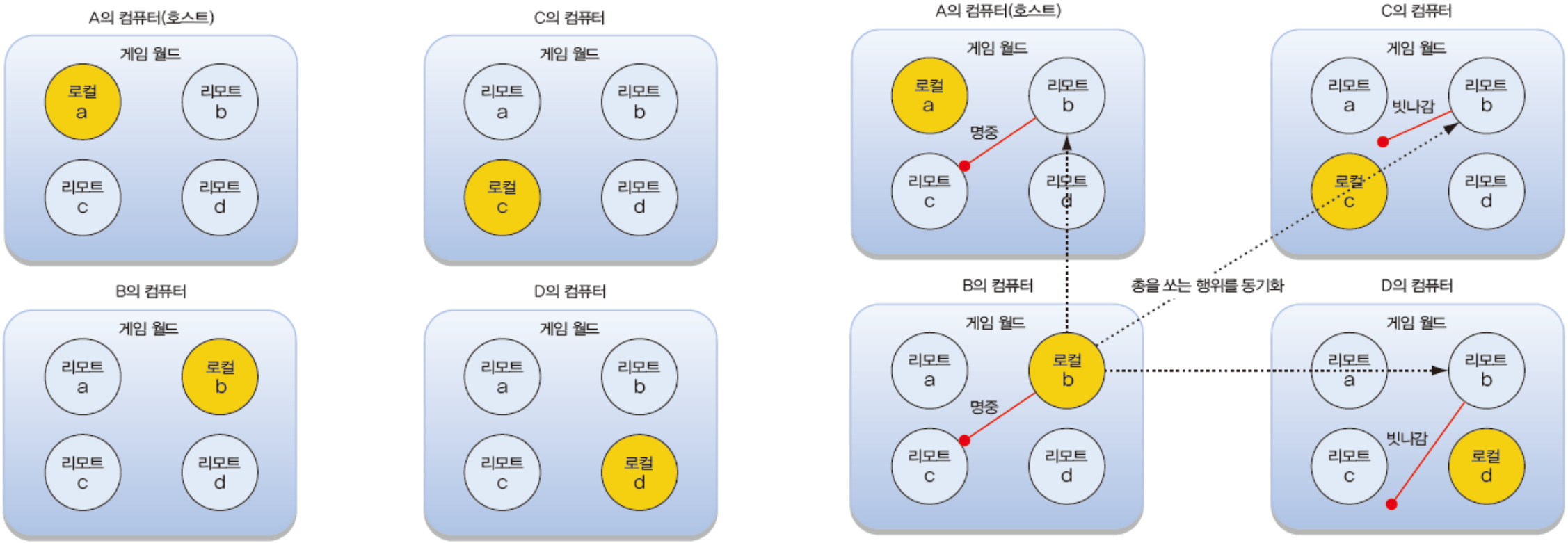
- 포톤은 여러 클라이언트가 모인 네트워크상의 가상의 공간을 룸이라는 단위로 부름
- 포톤의 룸은 유니티의 씬이 아니라는 점에 주의합니다. 포톤의 룸은 유니티의 씬과 다른 계층에서 동작
- 따라서 플레이어들이 같은 룸에 있지만 서로 다른 씬을 로드하는 것도 가능

네트워크 권한 분리(1)

호스트에 위임

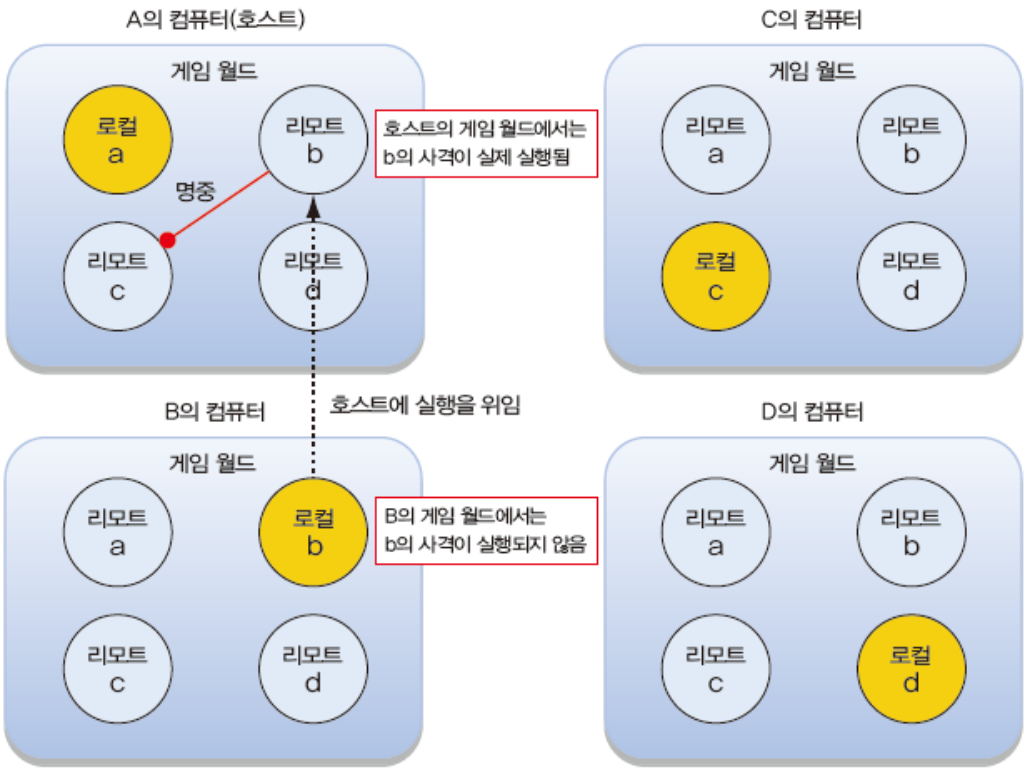
- 중요한 연산을 호스트에 위임하는 두 가지 대표적인 이유
  - 동기화에 오차가 존재하는 경우 기준이 되는 월드를 정하기 위해
  - 클라이언트의 변조나 위조 행위를 막기 위해
- 호스트에 중요한 연산을 위임하지 않는 방식으로 구현된 FPS 게임을 가정
  - 동기화에 오차가 생겨 동일한 행위에 대한 결과가 각자의 클라이언트마다 다를 경우 어떤 결과를 따라야 하는지 결정할 수 없는 문제가 발생
  - 특정 클라이언트의 비정상적인 행위나 수치 변조가 네트워크를 넘어 다른 클라이언트에도 그대로 적용되는 문제 발생

네트워크 권한 분리(2)

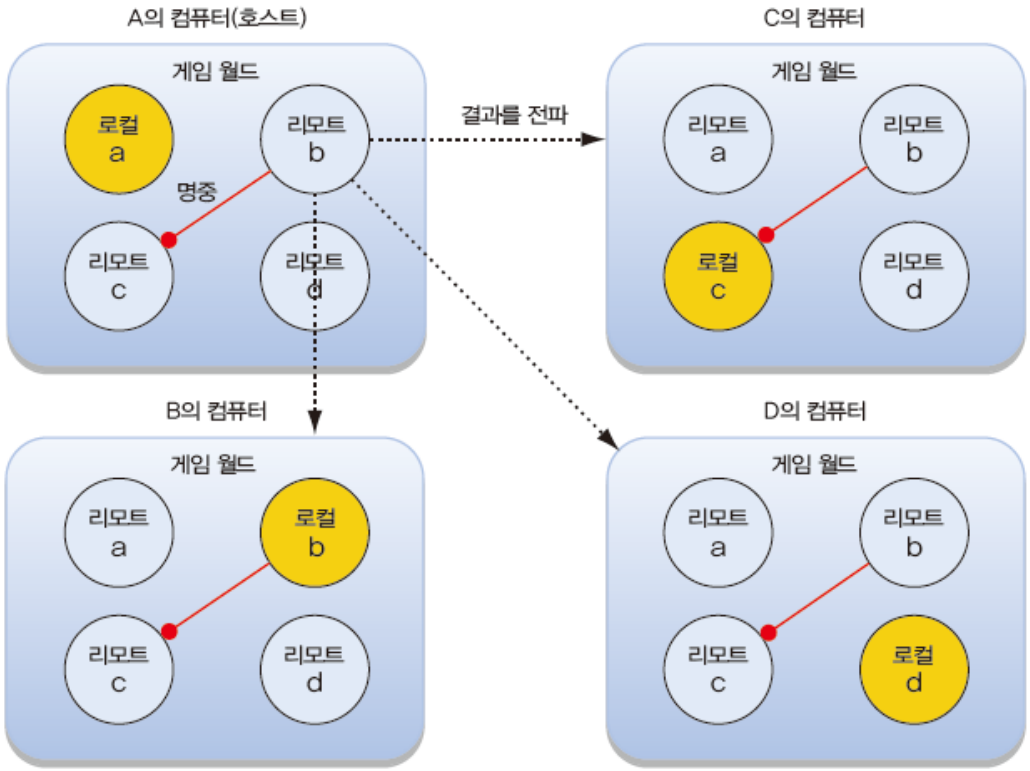


동기화에 오차가 생겨 동일한 행위에 대한 결과가 각자의 클라이언트마다 다르게 발생

네트워크 권한 분리(3)



▶ 사격 실행을 호스트에 위임



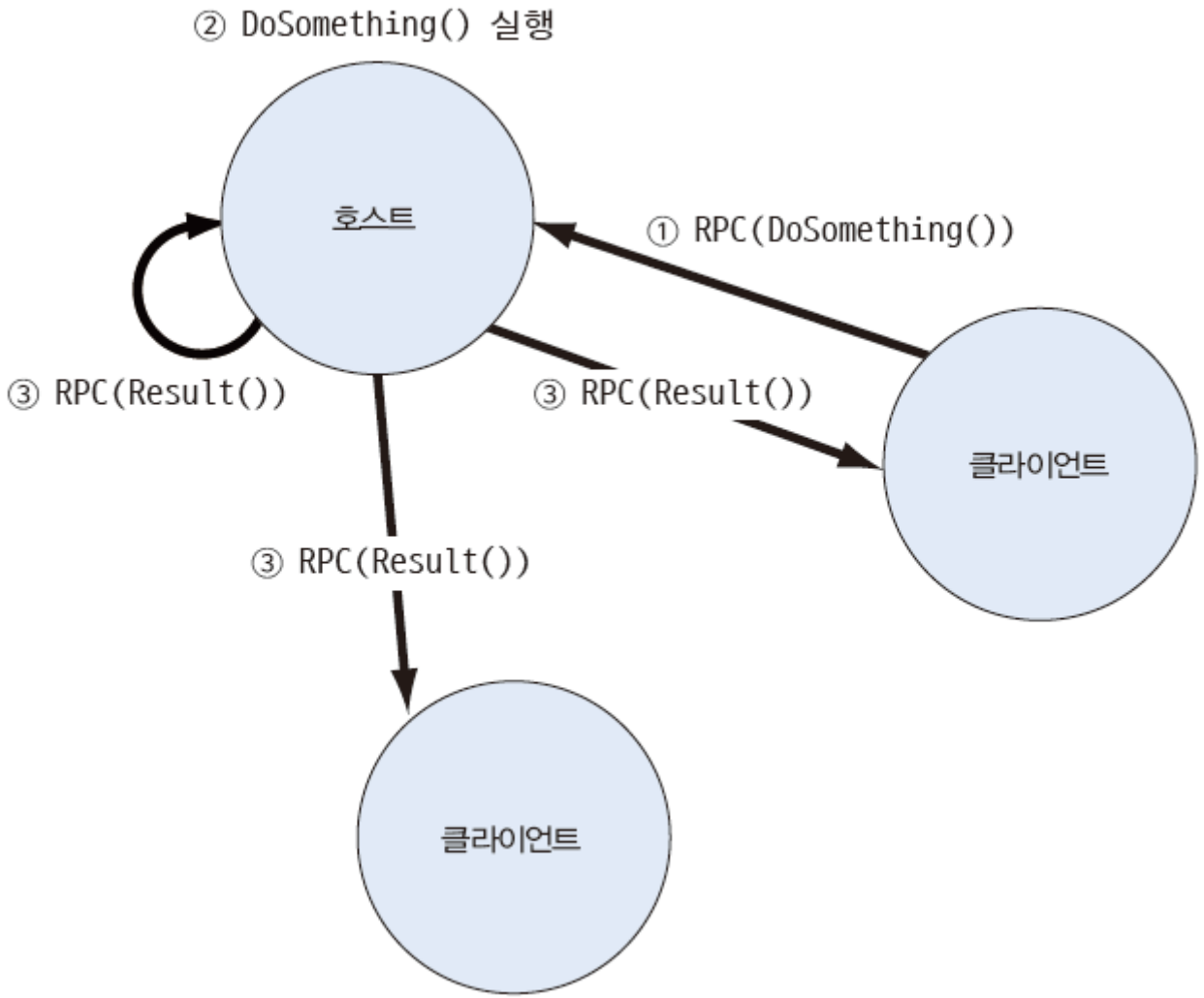
▶ 사격 결과를 전파

네트워크 권한 분리(4)

RPC

- 호스트에 처리를 위임하고, 호스트가 처리 결과를 클라이언트에 전파하려면 RPC(원격 프로시저 호출)를 구현
- RPC를 통해 호스트 A의 플레이어 캐릭터 b에서 Shot( )을 실행하도록 요청
  1. 사용자 B가 발사 버튼 누름
  2. 클라이언트 B → 호스트 A로 RPC(b.Shot( )) 전달
  3. 호스트 A에서 b.Shot( ) 실행
- 모든 클라이언트가 플레이어 캐릭터 b에서 ShotEffect( ) 메서드를 실행하여 발사 이펙트를 재생
  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

네트워크 권한 분리(5)



▶ 호스트와 클라이언트 사이의 실행 흐름

로비 만들기(1)

- Lobby 씬을 열고 네트워크 로비 제작을 시작

[과정 01] 로비 씬 열기

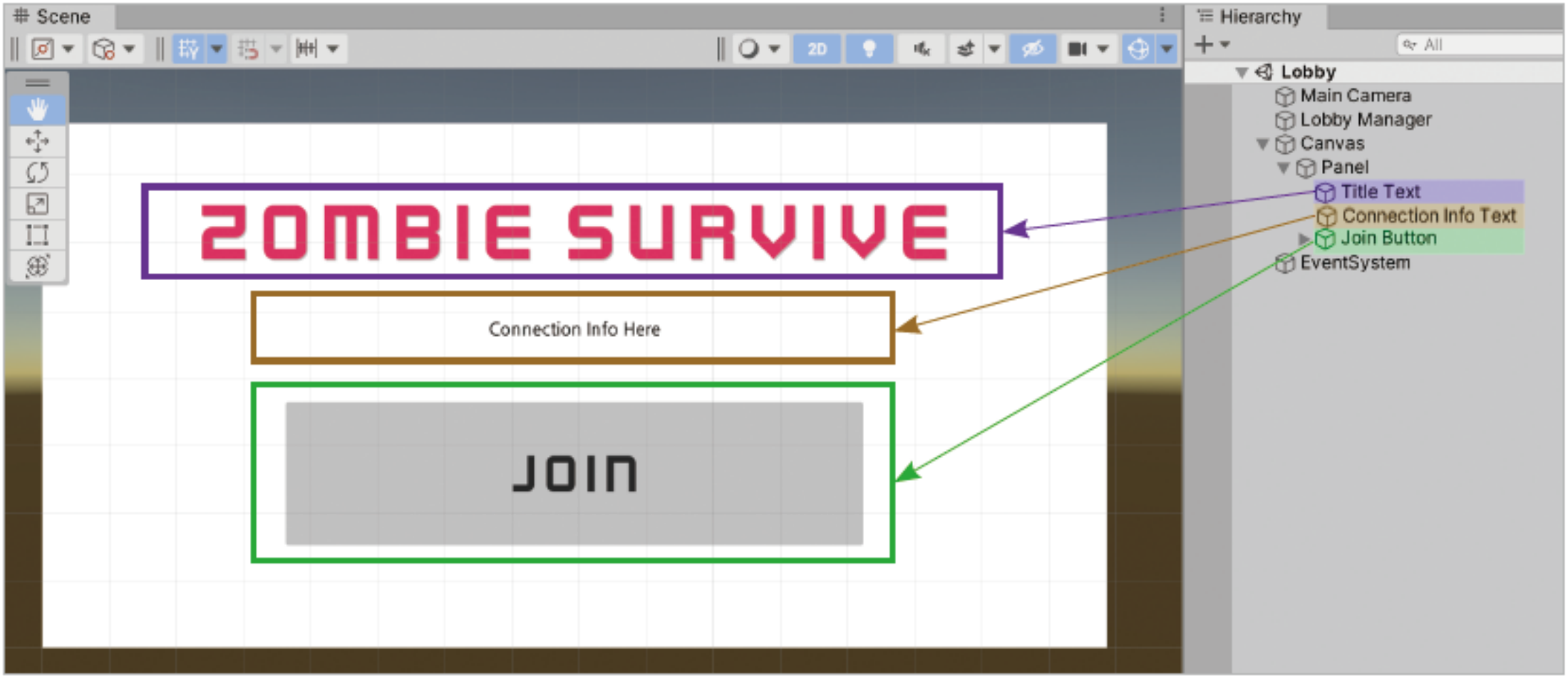
① 프로젝트 창에서 Scenes 폴더의 Lobby 씬 열기

로비 만들기(2)

Lobby 씬 살펴보기

- Lobby 씬은 다음과 같은 게임 오브젝트로 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

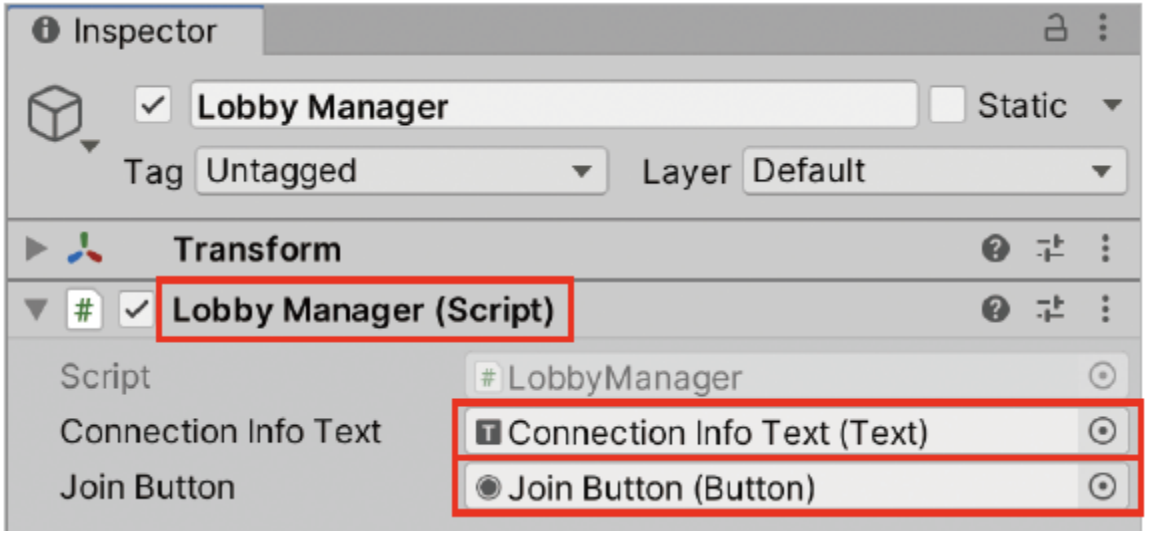
로비 만들기(3)



▶ Lobby 씬 구성

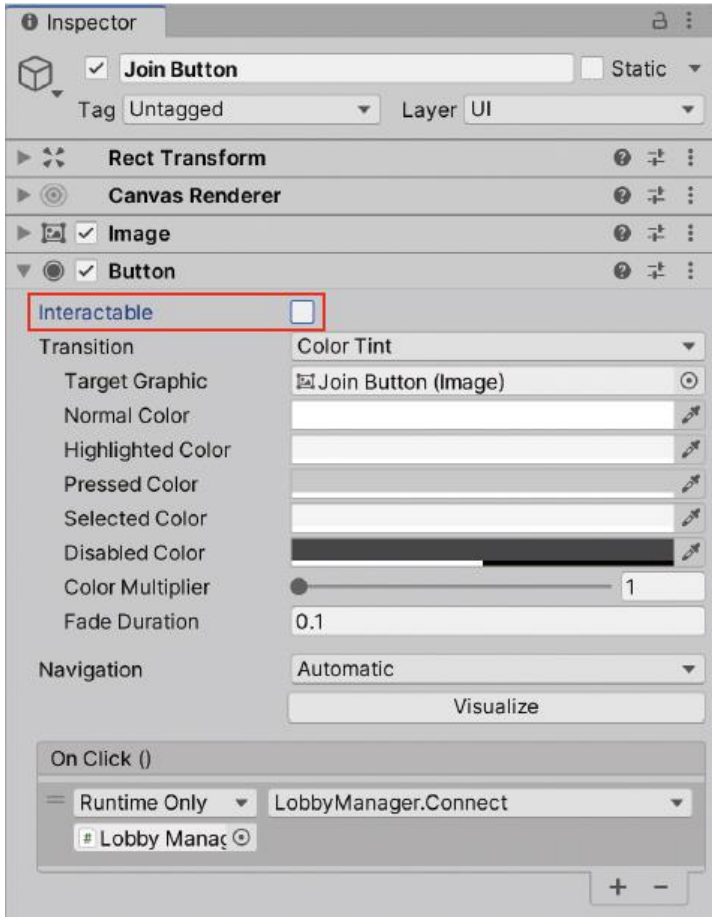
로비 만들기(4)

- Lobby Manager는 LobbyManager라는 스크립트만 컴포넌트로 추가된 빈 게임 오브젝트
  - Connection Info Text, Join Button이라는 필드가 존재하며,
    각각 Connection Info Text 게임 오브젝트의 텍스트 컴포넌트와 Join Button 게임 오브젝트의 버튼 컴포넌트가 할당



로비 만들기(5)

- Join Button 게임 오브젝트는 버튼 컴포넌트를 가진 UI 버튼
  - 플레이어가 Join Button을 누르면 매치메이킹 서버를 통해 룸에 접속하고 메인 게임으로 이동
  - Join Button 게임 오브젝트의 버튼 컴포넌트의 Interactable 체크가 해제되어 있음



로비 만들기(6)

LobbyManager 스크립트

[과정 01] LobbyManager 스크립트 열기

① Scripts 폴더의 LobbyManager 스크립트 열기

- using 문에서는 사용할 포톤 라이브러리를 가져옴

```
using Photon.Pun;
using Photon.Realtime;
```

- LobbyManager 클래스가 MonoBehaviourPunCallbacks를 상속

```
public class LobbyManager : MonoBehaviourPunCallbacks
```

- 사용할 변수들이 선언

```
private string gameVersion = "1";

public Text connectionInfoText;
public Button joinButton;
```

로비 만들기(7)

Start( ) 메서드

- LobbyManager의 Start( ) 메서드는 Photon 마스터 서버에 접속해 매치메이킹을 시도
- 접속하는 동안에는 룸 접속을 시도할 수 없도록 접속 버튼을 비활성화

[과정 01] LobbyManager의 Start ( ) 메서드 완성하기

① Start( ) 메서드를 다음과 같이 완성

- 마스터 서버에 접속을 시도

```
PhotonNetwork.GameVersion = gameVersion;
PhotonNetwork.ConnectUsingSettings();
```

- 마스터 서버에 접속을 시도하면서 동시에 joinButton의 interactable을 해제

```
joinButton.interactable = false;
```

- Connection Info Text의 텍스트 컴포넌트를 사용해 마스터 서버에 접속 중임을 표시하고 Start( ) 메서드가 종료

```
connectionInfoText.text = "마스터 서버에 접속 중...";
```

로비 만들기(8)

OnConnectedToMaster( ) 메서드

- OnConnectedToMaster( ) 메서드는 포톤 마스터 서버에 접속 성공한 경우 자동으로 실행
- LobbyManager의 OnConnectedToMaster( ) 메서드는 접속에 성공했다는 메시지를 표시하고, 접속 버튼인 Join Button이 상호작용 가능하도록 전환

[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 만들기(9)

OnDisconnected( ) 메서드

- OnDisconnected( ) 메서드는 마스터 서버 접속에 실패했거나, 이미 마스터 서버에 접속된 상태에서 어떠한 이유로 접속이 끊긴 경우 자동으로 실행됨

[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();
}
```

로비 만들기(10)

Connect( ) 메서드

- LobbyManager의 Connect( ) 메서드는 씬의 Join Button을 클릭했을 때 실행할 메서드
- 매치메이킹 서버(마스터 서버)를 통해 빈 무작위 룸에 접속을 시도

[과정 01] LobbyManager의 Connect ( ) 메서드 완성하기

① Connect( ) 메서드를 다음과 같이 완성

- Join Button 버튼의 interactable을 해제하여 Connect( ) 메서드의 룸 접속 처리가 끝나기 전에 버튼을 다시 클릭하여 Connect( ) 메서드가 실행되고 룸 접속이 중복 시도되는 상황을 방지

```
joinButton.interactable = false;
```

- 접속을 시도하는 예외 상황을 막기 위해 if 문으로 접속 상태 PhotonNetwork.IsConnected를 검사
  - 마스터 서버에 접속된 상태에서만 랜덤 룸 접속을 시도하고 접속 정보 텍스트의 내용을 갱신

```
if (PhotonNetwork.IsConnected)
{
    connectionInfoText.text = "룸에 접속...";
    PhotonNetwork.JoinRandomRoom();
}
```

로비 만들기(11)

- 만약 어떠한 이유로든 마스터 서버에 접속된 상태가 아니라면 룸 접속을 시도하지 않고 마스터 서버로의 재접속을 실행

```
else
{
    connectionInfoText.text = "오프라인 : 마스터 서버와 연결되지 않음\n접속 재시도 중...";
    PhotonNetwork.ConnectUsingSettings();
}
```

로비 만들기(12)

OnJoinRandomFailed( ) 메서드

- OnJoinRandomFailed( ) 메서드는 랜덤 룸 접속에 실패한 경우 자동 실행
  - 마스터 서버와의 연결이 끊긴 것이 아니라는 점에 주의

[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});
}
```

18.5 로비 만들기(13)

18.5.8 OnJoinedRoom( ) 메서드

- OnJoinedRoom( ) 메서드는 룸 참가에 성공한 경우 자동 실행
- 예제를 간단하게 하기 위해 좀비 룸에 접속하자마자 바로 메인 게임이 시작되도록 코드 완성

[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");
}
```

18.5 로비 만들기(14)

18.5.9 완성된 LobbyManager 스크립트

```
using Photon.Pun; // 유니티용 포톤 컴포넌트
using Photon.Realtime; // 포톤 서비스 관련 라이브러리
using UnityEngine;
using UnityEngine.UI;

// 마스터(매치메이킹) 서버와 룸 접속 담당
public class LobbyManager : MonoBehaviourPunCallbacks {
private string gameVersion = "1"; // 게임 버전

public Text connectionInfoText; // 네트워크 정보를 표시할 텍스트
public Button joinButton; // 룸 접속 버튼

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

▶ 다음 쪽에 이어짐

18.5 로비 만들기(15)

◀ 앞쪽에 이어

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

    // 마스터 서버 접속 성공 시 자동 실행
    public override void OnConnectedToMaster() {
        // 룸 접속 버튼을 활성화
        joinButton.interactable = true;
        // 접속 정보 표시
        connectionInfoText.text = "온라인 : 마스터 서버와 연결됨";
    }

    // 마스터 서버 접속 실패 시 자동 실행
    public override void OnDisconnected(DisconnectCause cause) {
```

▶ 다음 쪽에 이어짐

18.5 로비 만들기(16)

◀ 앞쪽에 이어

```
        // 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();
    }

    // 룸 접속 시도
    public void Connect() {
        // 중복 접속 시도를 막기 위해 접속 버튼 잠시 비활성화
        joinButton.interactable = false;
        // 마스터 서버에 접속 중이라면
        if (PhotonNetwork.IsConnected)
        {
```

▶ 다음 쪽에 이어짐

18.5 로비 만들기(17)

◀ 앞쪽에 이어

```
            // 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();
        }
    }

    // (빈 방이 없어) 랜덤 룸 참가에 실패한 경우 자동 실행
    public override void OnJoinRandomFailed(short returnCode, string message) {
        // 접속 상태 표시
        connectionInfoText.text = "빈 방이 없음, 새로운 방 생성...";
```

▶ 다음 쪽에 이어짐

18.5 로비 만들기(18)

◀ 앞쪽에 이어

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

    // (빈 방이 없어) 랜덤 룸 참가에 실패한 경우 자동 실행
    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});
    }

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