8주차 정리

개요

▸네트워킹과 멀티플레이어 게임 개발 학습

▸네트워크 매니저 설정 및 연결 처리 실습

▸기본 멀티플레이어 게임 요소 구현

▼ 네트워킹과 멀티플레이어 게임 개발 학습

고수준 스크립팅 API

Unity 네트워킹은 “고수준” 스크립팅 API를 제공합니다. 이는 HLAPI라고 부르며, 사용하게 되면 “로우 레벨” 구현 세부 사항에 대해 신경쓰지 않고도 멀티 유저 게임에 일반적으로 필요한 대부분의 요구 사항을 만족시킬 수 있는 커맨드에 접근할 수 있습니다. 아래는 HLAPI를 활용할 수 있는 방법입니다.

- “네트워크 관리자”를 사용하여 게임의 네트워크 상태를 제어.
- 호스트가 플레이어 클라이언트이기도 한 “클라이언트가 호스트하는” 게임 운영.
- 범용 시리얼라이저를 사용하여 데이터 직렬화.
- 네트워크 메시지 송수신.
- 네트워크 커맨드를 클라이언트에서 서버로 송신.
- 서버에서 클라이언트로 원격 프로시저 호출. (RPC)
- 서버에서 클라이언트로 네트워크 이벤트 전송.

엔진과 에디터 통합

Unity 네트워킹은 엔진과 에디터에 통합되어 있으므로, 멀티플레이어 게임을 빌드할 수 있도록 컴포넌트와 시각적 툴을 사용하여 작업할 수 있습니다. 다음은 몇 가지 기능입니

다.

- 네트워크 오브젝트용 NetworkIdentity 컴포넌트.
- 네트워크 스크립트용 NetworkBehaviour.
- 설정 가능한 오브젝트 변환의 자동 동기화.
- 스크립트 변수의 자동 동기화.
- Unity 씬에서 네트워크 오브젝트 배치 지원.
- Network 컴포넌트

▼ 네트워크 매니저 설정 및 연결 처리 실습

NetworkManager 설정

- 새로운 게임 오브젝트를 씬에 추가하고 “NetworkManager”로 이름을 변경합니다.
- **NetworkManager** 컴포넌트를 “NetworkManager” 게임 오브젝트에 추가합니다.
- 게임 오브젝트에 **NetworkManagerHUD** 컴포넌트를 추가합니다. 이 컴포넌트는 네트워크 게임 상태를 관리하는 디폴트 UI를 제공합니다.

자세한 내용은 NetworkManager 사용를 참조하십시오.

플레이어 프리팹 설정

- 게임에서 플레이어 게임 오브젝트용 프리팹을 찾거나 플레이어 게임 오브젝트에서 프리팹을 생성해야 합니다.
- **NetworkIdentity** 컴포넌트를 플레이어 프리팹에 추가합니다.
- NetworkIdentity의 LocalPlayerAuthority 상자에 체크합니다.
- NetworkManager의 **Spawn Info** 섹션에 `playerPrefab`를 플레이어 프리팹으로 설정합니다.
- 씬에 플레이어 게임 오브젝트 인스턴스가 있는 경우 인스턴스를 제거해야 합니다.

자세한 내용은 플레이어 오브젝트를 참조하십시오.

플레이어 이동

- NetworkTransform 컴포넌트를 플레이어 프리팹에 추가합니다.
- `isLocalPlayer`에 따라 입력 및 제어 스크립트를 업데이트합니다.

- 스폰된 플레이어 및 `isLocalPlayer`를 사용하도록 카메라를 수정합니다.

예를 들어, 아래 스크립트는 로컬 플레이어용 입력만을 처리합니다.

```
using UnityEngine;
using UnityEngine.Networking;

public class Controls : NetworkBehaviour
{
    void Update()
    {
        if (!isLocalPlayer)
        {
            // exit from update if this is not the local player
            return;
        }

        // handle player input for movement
    }
}
```

▼ 기본 멀티플레이어 게임 요소 구현

기본 플레이어 게임 상태

- 중요한 데이터를 포함하는 스크립트는 MonoBehaviours 대신 NetworkBehaviours에 포함되도록 해야 합니다.
- 중요한 멤버 변수는 SyncVars에 포함되도록 해야 합니다.

상태 동기화를 참조하십시오.

네트워크 액션

- 중요한 액션을 포함하는 스크립트는 MonoBehaviours 대신 NetworkBehaviours에 포함되도록 해야 합니다.
- 중요한 플레이어 액션을 수행하는 함수는 커맨드로 업데이트해야 합니다.

ServerCore(서버) / Client(클라이언트) 두 개의 스크립트를 작성한다.

ServerCore

```
static void Main(string[] args)
{
    //DNS (Domain Name System)
    //ip주소를 하드코딩을 하지 않음.
    string host = Dns.GetHostName();
    IPHostEntry ipHost = Dns.GetHostEntry(host);
    IPAddress ipAddr = ipHost.AddressList[0];
    IPEndPoint endPoint = new IPEndPoint(ipAddr, 7777);

    _listener.Init(endPoint, OnAcceptHandler);
    Console.WriteLine("Listening...");

    while (true)
    {

    }
}
```

```
string host = Dns.GetHostName();
IPHostEntry ipHost = Dns.GetHostEntry(host);
IPAddress ipAddr = ipHost.AddressList[0];
IPEndPoint endPoint = new IPEndPoint(ipAddr, 7777);
```

host : ip주소가 아닌 도메인주소로 사용하도록 한다.

AddressList : IP주소 리턴

IPEndPoint : IP주소와 포트번호를 통해 생성

Listener.class

```
참조 1개
public void Init(IPEndPoint endPoint, Action<Socket> OnAcceptHandler)
{
    _listenSocket = new Socket(endPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
    _OnAcceptHandler += OnAcceptHandler;
    //문지기 교육.
    _listenSocket.Bind(endPoint);

    //영업 시작.
    //backlog : 최대 대기수
    _listenSocket.Listen(10);

    SocketAsyncEventArgs args = new SocketAsyncEventArgs();
    args.Completed += new EventHandler<SocketAsyncEventArgs>(OnAcceptCompleted); //pending이 true가되면 콜백.
    RegisterAccept(args);
}
```

매개변수 OnAcceptHandler : Accept이 성공했을 때 불리는 콜백

_listenSocket.Bind(endPoint); : 넘겨 받아온 IPEndPoint의 바인드 처리

_listenSocket.Listen(10); : Listen 시작 클라이언트의 연결 요청을 대기상태

SocketAsyncEventArgs : 비동기 소켓의 관련한 패턴을 사용할 때? 쓰임

args.Completed += new EventHandler<SocketAsyncEventArgs>(OnAcceptCompleted); : 작업이 완료되면 OnAcceptCompleted을 호출

RegisterAccept(args); : 최조 한번 등록

```
private void RegisterAccept(SocketAsyncEventArgs args)
{
    args.AcceptSocket = null;

    bool pending = _listenSocket.AcceptAsync(args);
    if (pending == false) //즉시 완료됐을때.
        OnAcceptCompleted(null, args);
}
```

args.AcceptSocket = null; : 전에 사용했던 args이기 때문에 null 초기화 작업

bool pending = _listenSocket.AcceptAsync(args); : 완료되면 false를 리턴

즉시 완료되면 OnAcceptCompleted 실행

```
private void OnAcceptCompleted(object sender, SocketAsyncEventArgs args)
{
    if (args.SocketError == SocketError.Success)
    {
        _OnAcceptHandler.Invoke(args.AcceptSocket);
    }
    else
    {
        Console.WriteLine(args.SocketError.ToString());
    }

    RegisterAccept(args);
}
```

_OnAcceptHandler.Invoke(args.AcceptSocket); : 연결 성공시 송수신 처리

RegisterAccept(args); : 다시 Accept 처리.

Init()에서 최초 Accept 처리 -> Accept 성공 완료 콜백 OnAcceptCompleted 에서 Accept 처리 -> Accept처리 ...

로 계속 반복해서 Accept을 시도한다.

```
static Listener _listener = new Listener();
참조 1개
static void OnAcceptHandler(Socket socket)
{
    try
    {
        //받는다.
        byte[] recvBuff = new byte[1024];
        int recvByte = socket.Receive(recvBuff);
        string recvData = Encoding.UTF8.GetString(recvBuff, 0, recvByte);
        Console.WriteLine($"[From Client]{recvData}");

        //보낸다.
        byte[] sendBuff = Encoding.UTF8.GetBytes("Welecome to MMORPG Server !");
        socket.Send(sendBuff);

        //쫓아낸다.
        socket.Shutdown(SocketShutdown.Both);
        socket.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex);
    }
```