Fierce Combat Online (2023)

- **프로젝트 내용** : 플레이어끼리 서로 싸워 쓰러뜨리는 게임이다. 콤보, 스킬, 필살기 등을 적절히 활용해서 상대에게 승리하는 것이 필요합니다.
- **사용 툴 및 기술** : C#, Unity, Photon
- **제작 기간** : 2023.11.04 ~ 2023.12.13 (약 1.5개월)
- **제작 인원** : 3명 (프로그래밍 3명/그래픽 1명/기획 1명)
- **제작 동기** : 기존에 제작하던 로컬 멀티플레이어 게임을 클라우드 서버를 이용하여 온라인 게임으로 변환시키기 위해 제작한 프로젝트입니다.
- **목표** : 온라인 게임의 주요 기능을 구현하고, 게임 속 캐릭터와 스킬, 체력 등을 서버 상에서 동기화하여 게임의 유저들이 게임을 즐기는 데 이상이 없도록 하는 것입니다.

Unity 프로젝트

[맡은 역할]
- 프로그래밍(메인), 그래픽, 기획
    - 로비 리스트 관리, 채팅 서버 분리, 플레이어 데이터, 스킬 등의 네트워크 동기화, 맵 내 스킬 아이템 제작 등을 담당함.

(어려웠던 점) : 로비에서의 채팅과 게임 룸에서의 채팅을 어떻게 구분할지 고민하였고, 별도의 서버를 구현하는 방식으로 해결했습니다.

[구현 내용]
- 로비, 게임 룸, 게임 스테이지, 플레이어 간 채팅, PVP 등

- 스크린샷

GitHub 링크

유튜브 링크

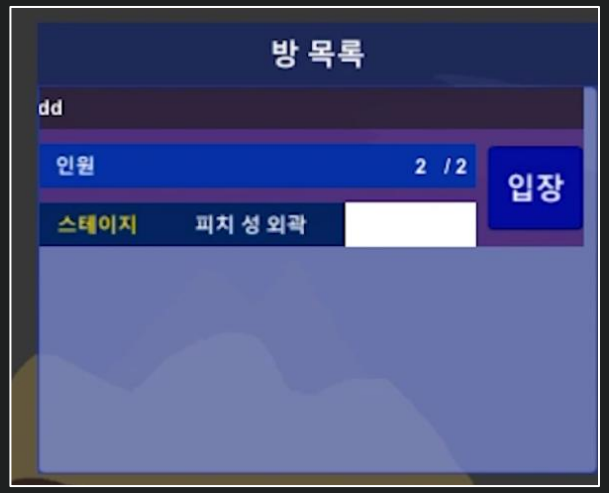
Unity 프로젝트

코드 링크

[게임 구조 : 로비 시스템] - Mingyu_PhotonLobby (1)

- 로비의 룸 리스트를 관리하고, 로비에서 특정 룸에 접속하거나 룸을 생성하는 기능을 포함한 클래스입니다.
- Awake 함수에서 게임 서버에 자동 접속합니다.
- OnRoomListUpdate 함수에서 실시간으로 서버에서 룸 리스트를 받아와 방 목록에 룸들을 추가하고, 서버의 룸 리스트에서 제거된 룸의 경우 삭제합니다.

```
[02:25:50] 마스터 서버에 접속중입니다.
UnityEngine.Debug:Log (object)
```




```
Unity 메시지 | 참조 0개
private void Awake()
{
    PhotonNetwork.GameVersion = "FierceFight 1.0";
    PhotonNetwork.ConnectUsingSettings();

    Debug.Log("마스터 서버에 접속중입니다.");
}
```

```
public override void OnRoomListUpdate(List<RoomInfo> roomList)
{
    foreach (RoomInfo item in roomList)
    {
        if (!item.RemovedFromList)
        {
            Debug.Log("방 생성");

            #region 룸 리스트 데이터 업데이트
            // 룸 리스트에 존재하지 않는다면, 추가해준다.
            if (!roomData_List.Contains(item))
                roomData_List.Add(item);

            // 룸 리스트에 존재한다면, roomList가 존재하는 리스트의 인덱스를 가져와서 데이터를 넣어준다.
            else
                roomData_List[roomData_List.IndexOf(item)] = item;
            #endregion

            Transform earlyRoom = roomListView.transform.Find(item.Name);
            RectTransform roomTemp;
            if (earlyRoom != null)
            {
                roomTemp = earlyRoom.GetComponent<RectTransform>();
            }
            else
            {
                roomTemp = GameObject.Instantiate(roomListItem).GetComponent<RectTransform>();
                roomTemp.SetParent(roomListView.transform);
                roomTemp.localScale = new Vector3(1, 1, 1);
            }
            Mingyu_RoomCtrl sync = roomTemp.GetComponent<Mingyu_RoomCtrl>();
            if (sync)
            {
                sync.name = item.Name;
                sync.roomNameT.text = item.Name;
                sync.playerNumberT.text = item.PlayerCount.ToString();

                sync.Set_RoomIndex = roomData_List.IndexOf(item);

                if (item.CustomProperties["stageName"] != null)
                {
                    Debug.Log(item.CustomProperties["stageName"].ToString());
                    sync.StageNameT.text = item.CustomProperties["stageName"].ToString();
                    Debug.Log("AA");
                }
                else
                    Debug.Log("이름 없음");
            }
            if (item != null)
                roomItems.Add(sync);
        }
```

Unity 프로젝트

코드 링크

[게임 구조 : 로비 시스템] - Mingyu_Photon_Lobby (2)

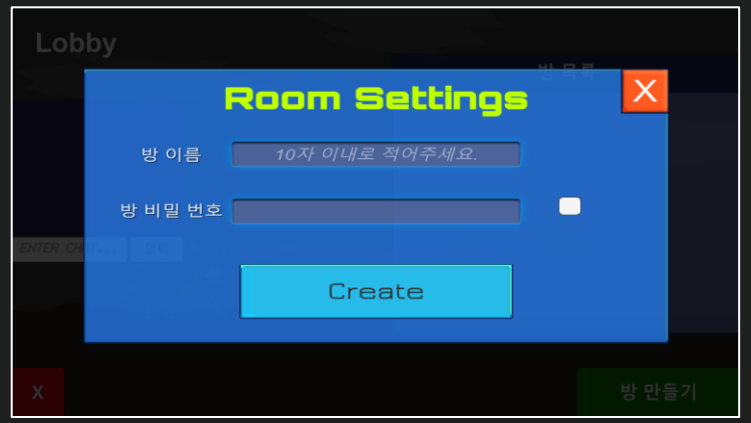
- BtnEvent_MakeRoomBtn 함수에서 방 생성 창을 띄우고, BtnEvent_MakeOk 함수에서 방의 생성과 관련된 로직을 처리합니다.
  - 헤시 테이블을 이용해 룸의 스테이지 명과 비밀번호를 설정함.

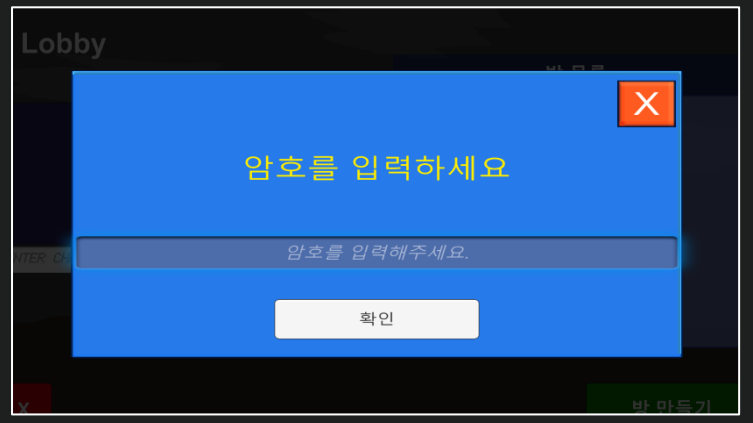
- EnterRoomWithPW 함수에서 패스워드가 있는 방의 경우 패스워드를 정확히 입력했을 경우에만 방에 접속 가능하게 했습니다.

```
// 패스 워드가 있는 방에 들어갔을 때, 비번이 맞고 틀린 유무에 따라
참조 1개
public void EnterRoomWithPW(int roomIndex)
{
    if ((string)roomData_List[roomIndex].CustomProperties["password"] == pw_CheckInput.text)
    {
        pw_Panel.SetActive(false);

        roomName = roomData_List[roomIndex].Name;

        StartCoroutine(Enter_WaitRoom(roomName, isCreateRoom));
    }
    else
        StartCoroutine("ShowPwWrongMsg");
}
```






```
// 방 만들기 입력
참조 0개
public void BtnEvent_MakeRoomBtn()
{
    Debug.Log("방 생성 버튼 클릭");

    makeRoom_Panel.SetActive(true);
}

// 방 만들기를 끝낸 후
참조 0개
public void BtnEvent_MakeOk()
{
    roomOptions = new RoomOptions();
    roomOptions.MaxPlayers = 2;
    Debug.Log(pw_Toggle.isOn);

    string default_StageName = "피치 성 외각 ";

    if (pw_Toggle.isOn)
    {
        string InputPWord = makeRoom_Panel.transform.Find("RoomPassword_InputField").
                            GetComponent<InputField>().text;

        roomOptions.CustomRoomPropertiesForLobby = new string[] { "password" };
        roomOptions.CustomRoomPropertiesForLobby = new string[] { "stageName" };

        roomOptions.CustomRoomProperties = new HashTable()
        {
            { "password", InputPWord },
            { "stageName", default_StageName }
        };
    }
    else
    {
        roomOptions.CustomRoomPropertiesForLobby = new string[] { "stageName" };
        roomOptions.CustomRoomProperties = new HashTable()
        {
            { "stageName", default_StageName }
        };
    }

    roomName = makeRoom_Panel.transform.Find("RoomName_InputField").
                            GetComponent<InputField>().text;


    isCreateRoom = true;
    makeRoom_Panel.SetActive(false);
    //PhotonNetwork.LeaveRoom();

    // 추가한 부분
    StartCoroutine(Enter_WaitRoom(roomName, isCreateRoom));
}
```

Unity 프로젝트

코드 링크

[게임 구조 : 로비 시스템] - Chat Service

- 게임 룸과 로비에서의 채팅을 분리하기 위해, Photon의 일반 서버와 Chat 서버를 동시에 연결하였습니다.

- 로비에서의 채팅은 Chat 서버에서 진행됩니다.
  - Chat Service 클래스에서 로비 내 채팅을 관리하며, Update에서 채팅 서버 내 클라이언트의 Service 함수를 호출해 채팅 내용을 주기적으로 업데이트함.

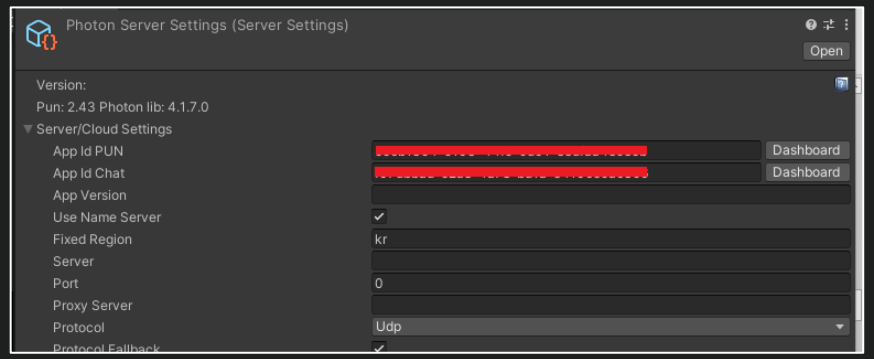
  - 로비에서의 대화는 Chat 서버에서 동작하도록 하여 게임 룸에서는 로비에 있는 유저와 채팅을 주고 받을 수 없음.

```
Unity 메시지 | 참조 0개
private void Update()
{
    if (chatClient != null)
    {
        chatClient.Service(); // 클라이언트의 서버 연결 상태 체크 및 갱신
```

```
참조 2개
public void OnConnected()
{
    chatClient.Subscribe(new string[] { "Lobby" }, 10); // Lobby 채널을 구독하여 메시지를 받을 수 있도록 함.
}


참조 1개
public void Connect() // Chat Server에 접속하기
{
    Application.runInBackground = true;
    chatClient = new ChatClient(this);

    chatClient.UseBackgroundWorkerForSending = true;
    chatClient.Connect(PhotonNetwork.PhotonServerSettings.AppSettings.AppIdChat, "1.0", new AuthenticationValues(PhotonNetwork.NickName));
}
참조 0개
public void OnEnterSend() // 엔터로 채팅 입력
{
    if (Input.GetKey(KeyCode.Return) || Input.GetKey(KeyCode.KeypadEnter))
    {
        SendChatMessage(InputFieldChat.text);
        InputFieldChat.text = "";
    }
}
참조 0개
public void OnClickSend() // 버튼 클릭으로 채팅 입력
{
    if (InputFieldChat != null)
    {
        SendChatMessage(this.InputFieldChat.text);
        InputFieldChat.text = "";
    }
}
```

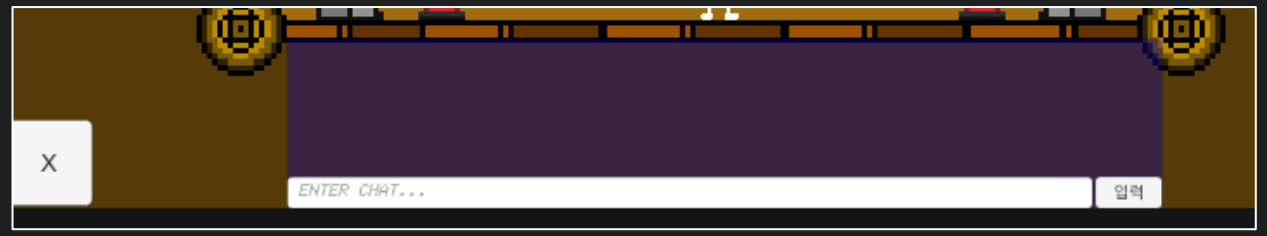


Unity 프로젝트

코드 링크

[게임 구조 : 플레이어 생성] - PhotonPlayerNetwork

- 게임 스테이지 또는 대기 룸에서의 플레이어 생성과 룸의 설정 변경 등이 구현된 클래스입니다.
- 클래스 내 멤버 변수 Is Lobby는 대기 룸에서는 true 값입니다.
  - 이를 통해 대기 룸에선 별도로 대기 룸 전용 캐릭터를 생성함.
  - 대기 룸에서는 캐릭터가 공격이 불가능하고 이모티콘 사용과 간단한 이동만 가능.
- 대기 룸 내에서의 채팅은 RPC(Remote Procedure Call)를 이용하였습니다.




```
#region 룸 채팅 부분 코딩
참조 0개
public void Chatting_Room(InputField inputChatting)
{
    ChatMessage = PhotonNetwork.NickName + ": " + inputChatting.text;
    inputChatting.text = string.Empty;

    pv.RPC("ChatInfo", RpcTarget.All, ChatMessage);
}
```

```
Unity 메시지 | 참조 0개
private void Awake()
{
    if (!isLobby)
    {
        if (PhotonNetwork.IsMasterClient)
        {
            spawn_P1 = PhotonNetwork.Instantiate(Player1Level.name, stage.playerFirstLocations[0].position, Quaternion.identity);
            StartCoroutine(Set_ReadyOption(spawn_P1));
        }
        else
        {
            spawn_P2 = PhotonNetwork.Instantiate(Player2Level.name, stage.playerFirstLocations[1].position, Quaternion.identity);
            StartCoroutine(Set_ReadyOption(spawn_P2));
        }
    }
}
```

```
참조 2개
IEnumerator Set_ReadyOption(GameObject spawnPlayer)
{
    // 생성 후, 3초 카운트 동안 플레이어의 상태를 무적으로 만듦
    spawnPlayer.GetComponent<Entity>().DamageBlock = Entity.DefenseStatus.Invincible;
    Ready_UI.SetActive(true);

    yield return new WaitForSecondsRealtime(1.5f);
    StartCoroutine(GameStart(spawnPlayer));
}

참조 1개
IEnumerator GameStart(GameObject spawnPlayer)
{
    spawnPlayer.GetComponent<Entity>().DamageBlock = Entity.DefenseStatus.Nope;
    Ready_UI.SetActive(false);
    Go_UI.SetActive(true);

    yield return new WaitForSecondsRealtime(0.5f);
    Go_UI.SetActive(false);
}

Unity 메시지 | 참조 0개
private void Start()
{
    if(player1Name && PhotonNetwork.PlayerList != null && PhotonNetwork.PlayerList.Length > 1)
        foreach (Player pl in PhotonNetwork.PlayerList)
        {
            if(pl.IsMasterClient)
                player1Name.text = pl.NickName;
            else
                player2Name.text = pl.NickName;
        }
    pool = PhotonNetwork.PrefabPool as DefaultPool;
    Debug.Log("실행");

    if (pool != null && !pool.ResourceCache.ContainsKey(PlayerLobby.name))
    {
        pool.ResourceCache.Add(PlayerLobby.name, PlayerLobby);
        pool.ResourceCache.Add(Player1Level.name, Player1Level);
        pool.ResourceCache.Add(Player2Level.name, Player2Level);

        // PhotonNetwork.InRoom으로 방에 있을 때만 실행되도록 수정
        if (PhotonNetwork.InRoom && SceneManager.GetActiveScene().name == "WaitingRoom")
        {
            StartCoroutine(WaitSpawn(pool));
        }
        else
        {
            // 방에 들어갈 때까지 대기하는 코루틴을 실행
            StartCoroutine(WaitForJoinRoom(pool));
        }
    }
}
```

Unity 프로젝트

코드 링크

[게임 구조 : 대기 룸] - WaittingRoomCtrl

- 대기 룸에서 게임 스테이지를 변경할 수 있게 하고, 변경 내용을 로비에 반영하는 기능을 담당하는 클래스입니다.
- 플레이어 모두가 준비 상태가 되었을 때 게임을 시작하는 기능도 포함하고 있습니다.


```
참조 0개
public void BtnClick_StageSelect()
{
    if (PhotonNetwork.CurrentRoom.CustomProperties["stageName"] != null)
    {
        ExitGames.Client.Photon.Hashtable customRoomProperties = new ExitGames.Client.Photon.Hashtable();
        customRoomProperties.Add("stageName", select_StageName);
        PhotonNetwork.CurrentRoom.SetCustomProperties(customRoomProperties);
    }
}
```

```
Unity 메시지|참조 0개
private void Update()
{
    if (!PhotonNetwork.IsMasterClient)
        return;

    // 각 양 옆의 버튼 상태 실시간 체크
    isColl_LButton = L_Button.GetComponent<ButtonCtrl>().Is_CollPlayer;
    isColl_RButton = R_Button.GetComponent<ButtonCtrl>().Is_CollPlayer;

    if (isGameStart == false && isColl_LButton && isColl_RButton)
    {
        isGameStart = true;

        if (PhotonNetwork.IsMasterClient)
        {
            if(select_StageName == "피치 성 외각")
                PhotonNetwork.LoadLevel("Peach Castle");
        }
    }
}
```

Unity 프로젝트

코드 링크

[게임 구조 : 플레이어 네트워크 동기화] - PhotonPlayer

- 플레이어를 네트워크 상에서 동기화 시켜주는 클래스입니다.

- 동기화 방법
  - 애니메이션 : RPC 이용
    - 추가적으로 애니메이션 이벤트를 활용해 스킬, 컷신 등을 재생함.
  - 위치 값 : 포톤 뷰로 x,y 값 동기화
  - 회전 값 : 좌우만 필요하니 포톤 뷰로 Bool 값 전달.
  - 콤보 횟수 표시 : RPC 이용

```
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (stream.IsWriting)
    {
        stream.SendNext(posX);
        stream.SendNext(posY);
        stream.SendNext(rot);
        stream.SendNext(combo);
    }
    else
    {
        posX = (float)stream.ReceiveNext();
        posY = (float)stream.ReceiveNext();
        rot = (bool)stream.ReceiveNext();
        combo = (int)stream.ReceiveNext();
    }
}
```

```
[PunRPC]
참조 0개
public void ShowNickname(string name)
{
    ft.NameUpdate(name);
}

[PunRPC]
참조 0개
public void ChangeSkill(int value, bool SkillValue)
{
    SkillActive[value] = SkillValue;
}

[PunRPC]
참조 0개
public void SetMp(int value)
{
    entity.SetMpNetwork(value);
}

[PunRPC]
참조 0개
public void ShowEmoticon(int value)
{
    entity.emoticon.SetValue(value);
}

[PunRPC]
참조 0개
public void ShowEffect(Vector2 pos)
{
    GameObject temp = Instantiate(effect);
    temp.transform.position = pos;
    temp.GetComponent<HitCollider>().owner = gameObject.GetComponent<Entity>();
}


[PunRPC]
참조 0개
public void ShowCombo()
{
    ComboView cv = Instantiate(entity.ComboUI);
    ComboView.owner = entity;
}

[PunRPC]
참조 0개
public void Network_Thrust(float thrustValue)
{
    if (thrustValue < 0)
        transform.localEulerAngles = new Vector3(0, 0, 0);
    else
        transform.localEulerAngles = new Vector3(0, 180, 0);
    mv.SetThrustForceX(thrustValue);
}


[PunRPC]
참조 0개
public void Network_Trigger(string name)
{
    if(am)
        am.Network_SetTrigger(name);
}
```

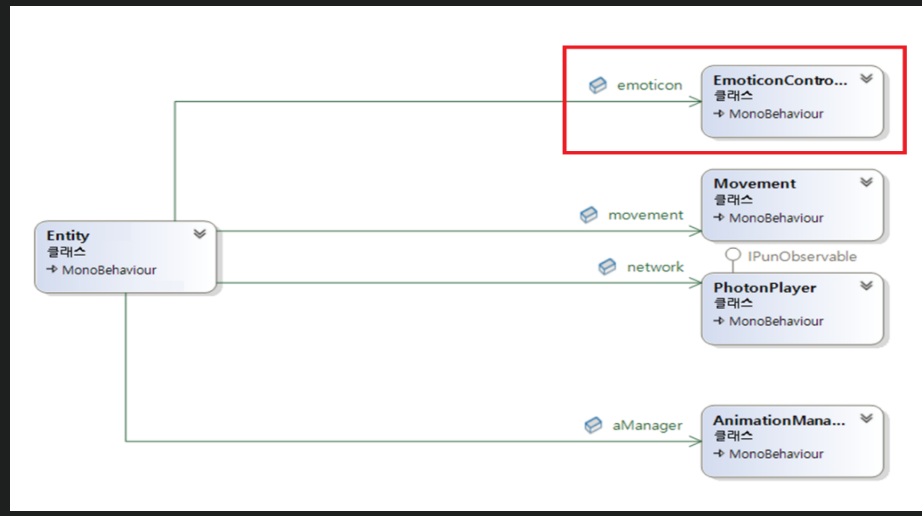
Unity 프로젝트

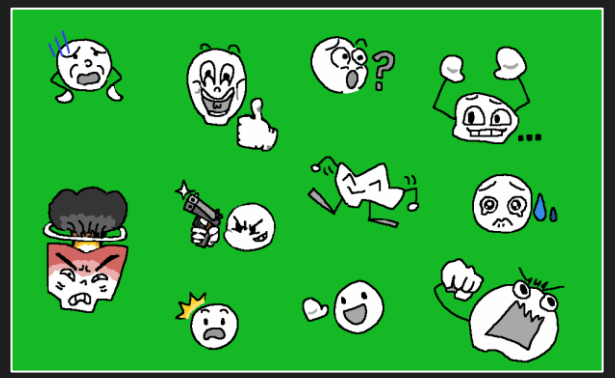
unity

코드 링크

[게임 구조 : 플레이어 네트워크 동기화] - EmoticonController

- 플레이어의 이모티콘 사용을 도와주는 클래스입니다.
- 플레이어의 기능을 종합적으로 관리하도록 생성한 Entity 클래스에서 사용합니다.
  - F1 ~ F11 까지 버튼으로 각종 감정 표현이 가능하게 구현함.
  - RPC를 통해 플레이어 캐릭터 위에 숨은 오브젝트에서 이모티콘 애니메이션을 실행함.





```
if (emoticon && network && network.pv.IsMine)
    foreach (KeyCode key in keyValues)
    {
        if (Input.GetKeyDown(key))
        {
            if ((int)key >= 282 && (int)key <= 292)
            {
                int keyValue = (int)key - 281;
                emoticon.SetValue(keyValue);
                if(network)
                    network.pv.RPC("ShowEmoticon", RpcTarget.Others, keyValue);
                break;
            }
        }
    }
```

Unity 프로젝트

코드 링크

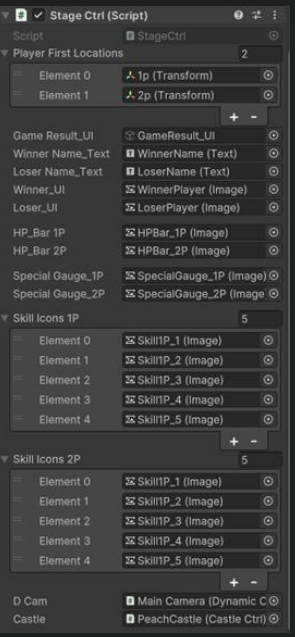
[게임 구조 : 플레이어 네트워크 동기화] - StageCtrl

- 전투 중 체력, MP, 보유 스킬, 플레이어 닉네임 표시 등을 네트워크 상에서 관리하고, 게임의 승패를 결정하는 클래스입니다.
- CreateUIManager 함수를 통해 UI를 관리하는 UIManager 객체를 생성해 각 플레이어와 연관된 오브젝트들을 연결시킵니다.
- 코루틴 PlayerCheck를 실행해 실시간으로 플레이어의 체력을 체크해 승패 결과를 체크합니다.


```
Unity 메시지 | 참조 0개
void Update()
{
    if (PlayerList == null || PlayerList.Length < 2)
    {
        PlayerList = FindObjectsOfType<Entity>(); //Entity를 가진 객체로 리스트를 만듦
    }
    if (PlayerList != null)
    {
        if (PlayerList.Length > 1)
        {
            if (!stageSettingEnd)
            {
                stageSettingEnd = true; // 다중 실행 방지
                if (castle)
                {
                    castle.enabled = true;
                }
                dCam.enabled = true;
                playerNumber = PlayerList.Length; //처음은 살아있으므로 리스트에 넣음
                superPlayer = PlayerList[0]; //임시로 superPlayer 설정

                CreateUIManager(); //UIManager 생성
                                   //Debug.Log("플레이어 수" + playerNumber);
                                   //Debug.Log("플레이어1" + PlayerList[0].name);
                                   //Debug.Log("플레이어2" + PlayerList[1].name);

                StartCoroutine(PlayerCheck());
            }
        }
    }
}
```




```
참조 1개
void CreateUIManager()
{
    GameObject UIObj = new GameObject("UIManager");
    UICtrl myUICtrl = UIObj.AddComponent<UICtrl>();

    myUICtrl.HP_Bar1P = HP_Bar1P;
    myUICtrl.HP_Bar2P = HP_Bar2P;
    myUICtrl.SpecialGauge_1P = SpecialGauge_1P;
    myUICtrl.SpecialGauge_2P = SpecialGauge_2P;
    myUICtrl.SkillIcons1P = SkillIcons1P;
    myUICtrl.SkillIcons2P = SkillIcons2P;

    foreach (Entity player in PlayerList)
    {
        if (player.network.pv.IsMine)
        {
            if (PhotonNetwork.IsMasterClient)
                myUICtrl.Player1 = player;
            else
                myUICtrl.Player2 = player;
        }
        else
        {
            if (PhotonNetwork.IsMasterClient)
                myUICtrl.Player2 = player;
            else
                myUICtrl.Player1 = player;
        }
    }
}
```

```
//player 체력 체크 후 승패 체크
참조 1개
IEnumerator PlayerCheck()
{
    while (playerNumber > 0 && !isDie)
    {
        for (int i = 0; i < PlayerList.Length; i++)
        {
            if (PlayerList[i].GetHp() >= superPlayer.GetHp())
                superPlayer = PlayerList[i];
        }

        //승리 조건
        if (superPlayer.isDie == true)
        {
            result = GameResult.Draw;
            Debug.Log("Draw");
        }
        else if (PlayerList[1].GetHp() <= 0)
        {
            result = GameResult.Win;
            Debug.Log(PlayerList[0].name + " Win");

            StartCoroutine(View_GameResult(PlayerList[0]));
        }
        else if (PlayerList[0].GetHp() <= 0)
        {
            result = GameResult.Lose;
            Debug.Log(PlayerList[1].name + " Win");

            StartCoroutine(View_GameResult(PlayerList[1]));
        }

        //Debug.Log("수빠 플레이어 : " + superPlayer.name);
        //Debug.Log("Player1 HP :" + PlayerList[0].GetHp() + "Player2 HP :" + PlayerList[1].GetHp());
        Debug.Log(PlayerList[1].isDie);
        yield return new WaitForSecondsRealtime(0.5f);
    }
}
```

Unity 프로젝트

- 성과
  - 로비와 룸에서의 대화가 서로 분리되도록 서버를 분리하는데 성공해 게임 룸에 있는 사람끼리 대화가 가능하도록 구현하였습니다.
  - 온라인 게임의 핵심 요소인 방 생성 및 방 정보 동기화, 채팅 및 이모티콘 사용을 게임에서 제공했습니다.
  - 플레이어 데이터를 실시간으로 공유하고, 게임 내 컷신 재생 또한 동기화에 성공하였습니다.