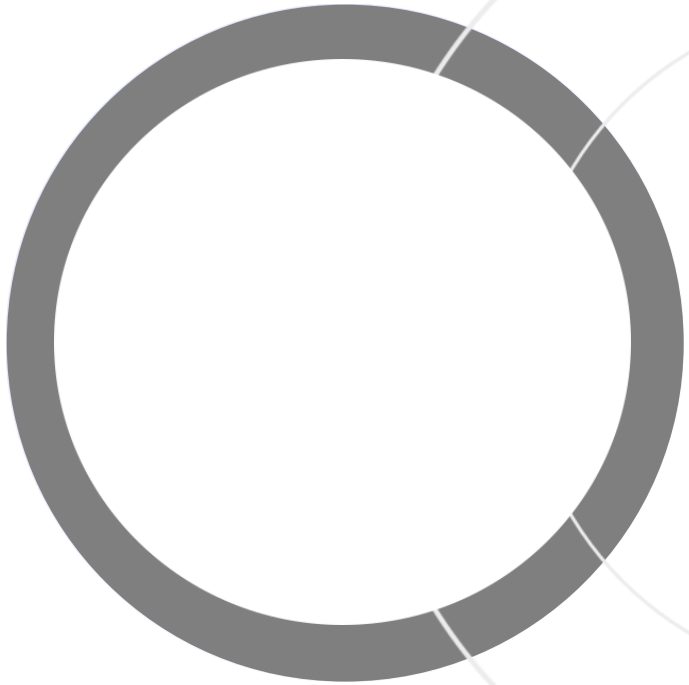


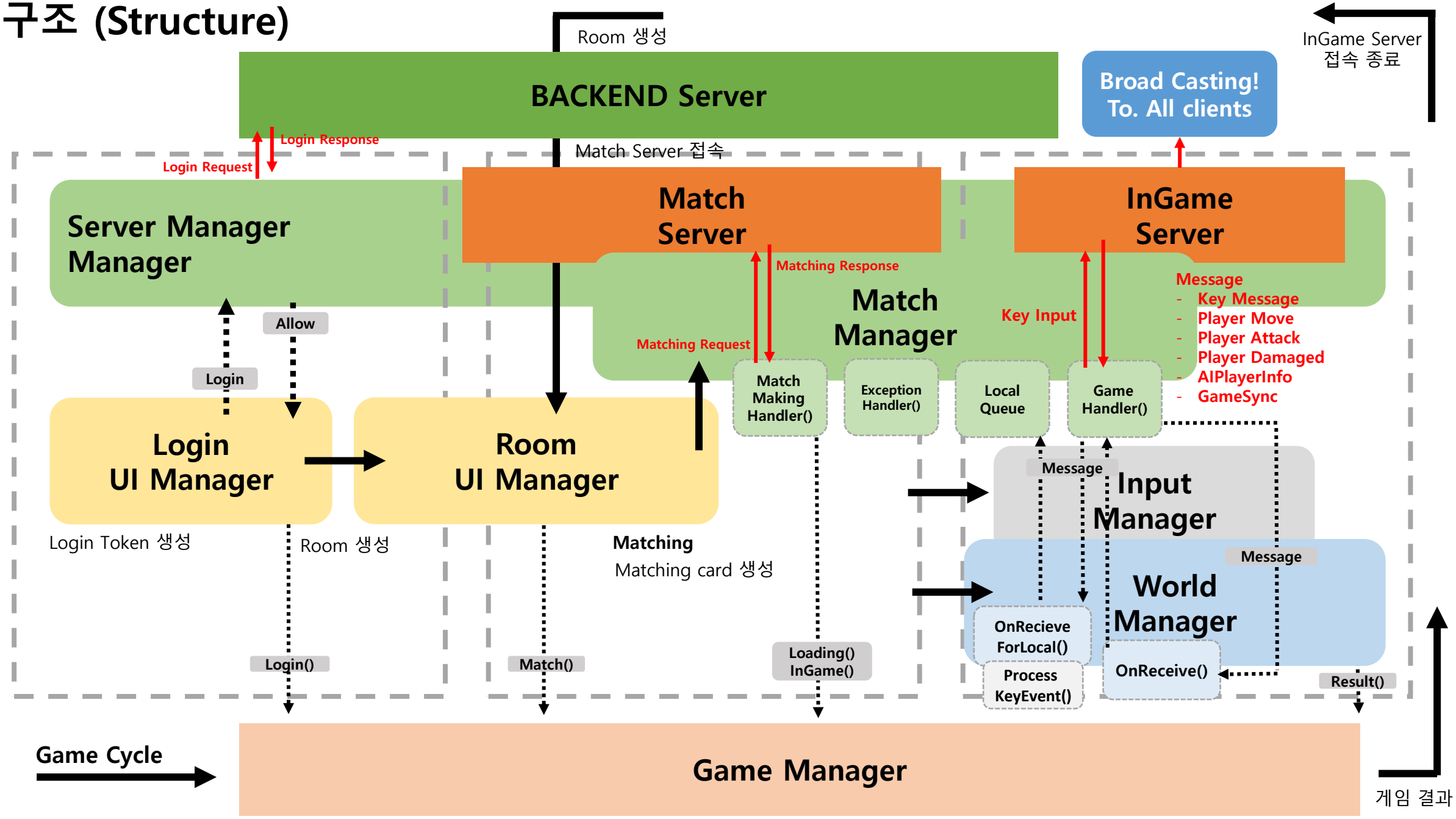
PRESENTATION

KEON GI KIM

CLIENT DEVELOPER

CONTENTS

1. 구조 (Structure)



2. Game Manager

Game Manager.cs

Introduce

현재 진행중인 게임의 상태를 저장하고 처리하며, 상태 핸들러를 통해 Scene을 전환하는 매니저. 모든 매니저는 인스턴스화 후, 하나의 인스턴스를 가진다. (Signtone Pattern)

실행주기
어플리케이션 시작과 동시에 실행되며, 어플리케이션이 종료되기 전까지 종료되지 않는다.

동작방식
시작과 동시에 각 Scene에서 사용될 **EventHandler**들을 초기화하며, Scene전환 함수가 호출 되면 Scene을 전환하고 해당 Scene에 필요한 GameManager의 State정보를 변경한다.

EventHandler

OnGameReady, InGame, AfterInGame, OnGameOver, OnGameResult, OnGameReconnet 모두 Game Manager가 State가 변경 될 때 이벤트가 추가되며, 이벤트 호출이 일어 날 때 호출되는 이벤트 핸들러이다.

이벤트 핸들러는 게임 시작시 호출 되는 GameStart()와 게임 재연결시도시 호출 되는 GameReconnect()로 초기화 된다.

Awake()

Application.targetFrame을 통해 프레임을 고정후, 주석과 동일하게 게임 슬립모드를 해제한다.

InGameUpdateCoroutine은 InGame핸들러에서 InputManager의 값을 1초 단위로 가져오기 위한 Coroutine이다.

DontDestroyOnLoad를 통해 GameManager Object는 Scene이 전환되어도 종료되지 않는다.

```
namespace PvpTank
{
    public class GameManager : MonoBehaviour
    {
        private static GameManager instance;
        private static bool isCreate = false;

        #region Scene_name
        private const string LOGIN = "0. Login";
        private const string LOBBY = "1. Match";
        private const string READY = "2. LoadRoom";
        private const string INGAME = "3. InGame";
        #endregion

        #region Actions-Events
        public static event Action OnGameReady = delegate { }; // Play User Ready UI? (Action? Data?)
        public static event Action InGame = delegate { }; // Decrease Heart point? or Player Action Data? // INIT IN GAME
        public static event Action AfterInGame = delegate { }; // Result Window? // INIT IN GAME
        public static event Action OnGameOver = delegate { };  // INIT IN GAME
        public static event Action OnGameResult = delegate { }; // INIT IN GAME
        public static event Action OnGameReconnect = delegate { };

        private string asyncSceneName = string.Empty;
        private IEnumerator InGameUpdateCoroutine; // Not Exist Update() Function!

        public enum GameState { Login, MatchLobby, Ready, Start, InGame, Over, Result, Reconnect };
        private GameState gameState;
        #endregion

        public static GameManager GetInstance()
        {
            if (instance == null)
            {
                Debug.LogError("Do not exist GameManager Object Instance.");
                return null;
            }
            return instance;
        }

        void Awake()
        {
            if (!instance)
            {
                instance = this;
            }

            Application.targetFrameRate = 60; // 60프레임 고정

            Screen.sleepTimeout = SleepTimeout.NeverSleep; // 게임중 화면슬립모드 해제
            InGameUpdateCoroutine = InGameUpdate(); // Update() => Coroutine

            DontDestroyOnLoad(this.gameObject); // GameManager is can not Destory
        }
```

2. Game Manager

GameManager.cs

ChangeState()

State를 통해 현재 gameState를 업데이트하고, 현재 state에서 일어날 Event를 찾아 호출한다.
MatchManager 혹은 에서 각 Scene의 UIManager 및 InGame의 WorldManager에서 호출된다.
InGame의 경우 앞서 말한 것처럼, StartCoroutine을 호출 시켜 Update()를 대신한다.

ChangeScene()

위에서 정의한 상수형 문자열들과 변경되어야 할 scene을 판단 후, SceneManager에게 변경되어야할 Scene으로 이동을 명령한다.

```
public void ChangeState(GameState state, Action<bool> func = null)
{
    gameState = state;
    switch (gameState)
    {
        case GameState.Login:
            Login();
            break;
        case GameState.MatchLobby:
            MatchLobby(func);
            break;
        case GameState.Ready:
            GameReady();
            break;
        case GameState.Start:
            GameStart();
            break;
        case GameState.Over:
            GameOver();
            break;
        case GameState.Result:
            GameResult();
            break;
        case GameState.InGame:
            // 코루틴 시작
            StartCoroutine(InGameUpdateCoroutine);
            break;
        case GameState.Reconnect:
            GameReconnect();
            break;
        default:
            Debug.Log("Unknown State. Please Confirm current state");
            break;
    }
}
public GameState GetGameState()
{
    return gameState;
}
public bool IsLobbyScene()
{
    return SceneManager.GetActiveScene().name == LOBBY;
}

private void ChangeScene(string scene)
{
    if (scene != LOGIN  && scene != INGAME && scene != LOBBY && scene != READY)
    {
        Debug.Log("Unknown Scene");
        return;
    }
    Debug.Log("CURRENT SCENE :: " + scene);
    SceneManager.LoadScene(scene);
}
```

3. Server Manager

Server Manager.cs

Introduce

서버와 클라이언트간 로그인 접속 및 회원가입을 위한 **API**, 현재 서버 접속 상태를 확인할 수 있다.

실행 주기
어플리케이션 시작과 동시에 실행되며, 어플리케이션이 종료되기 전까지 종료 되지 않는다.

동작방식
Login 버튼 클릭시 사용자의 ID와 PW가 서버에 전달되어 로그인 요청이 이루어지고, 로그인이 활성화 되었을 경우 **토큰**을 반환하여 어플리케이션이 종료되더라도 일정시간동안 재로그인 없이 **로그인활성화**가 가능하다. 마찬가지로, 회원 가입의 경우도 **SignUp** 버튼 클릭 시 사용자의 ID와 PW 그리고 닉네임이 서버에 전달되어 회원 가입 요청이 이루어진다.

loginSuccesFunc

loginSuccessFunc은 서버에 로그인 요청 시 콜백을 가리키는 대리자로, 해당 대리자를 통해 로그인 성공 여부와 errMessage를 통해 로그인 방법을 알 수 있다.

Start()

Backend.Initialize를 통해 뒤끝 로그인 서버와의 연결을 시작한다. 만약, 성공적으로 서버와 연결이 되었다면, 사용자 정보를 불러오기 전 OnPrevBackendAuthorized()를 호출하여 isLogin = true로 사전 작업 후 OnBackendAuthorized()를 호출하여 userNickName, userIndate를 업데이트 한다.

```
using System;
using System.Collections.Generic;
using UnityEngine;
using BackEnd;
using static BackEnd.SendQueue;

using UnityEngine.SocialPlatforms;
namespace PvpTank
{
    public class ServerManager : MonoBehaviour
    {
        private static ServerManager instance;   // 인스턴스
        public bool isLogin { get; private set; }   // 로그인 여부

        private string tempNickName; // 설정할 닉네임 (id와 동일)
        public string userNickName { get; private set; } = string.Empty;  // 로그인한 계정의 닉네임
        public string userIndate { get; private set; } = string.Empty;    // 로그인한 계정의 inDate
        private Action<bool, string> loginSuccessFunc = null;

        private const string BackendError = "statusCode : {0}₩nErrorCode : {1}₩nMessage : {2}"; // Error Message

        void Awake()
        {
            if (instance != null)
            {
                Destroy(instance);
            }
            instance = this;

            DontDestroyOnLoad(this.gameObject);
        }

        public static ServerManager GetInstance()
        {
            if (instance == null)
            {
                Debug.LogError("Not exist ServerManager instnace.");
                return null;
            }

            return instance;
        }

        // Start is called before the first frame update
        void Start()
        {
            isLogin = false;

            var bro = Backend.Initialize(true);

            if (bro.IsSuccess())
            {
                Debug.Log("Success to Initialize Backend : " + bro);
                #if UNITY_ANDROID
                    Debug.Log("GoogleHash = " + Backend.Utils.GetGoogleHash());
```

3. Server Manager

Server Manager.cs

Update()

서버에서 비동기 함수의 콜백은 비동기 IO에서 실행되지 않고 SendQue에 저장된다 그러므로 프레임마다 backend.AsycPool()사용해 주기적으로 큐에 저장된 콜백함수를 풀링해야한다.

GetVersionInfo()

Backend.Utils.GetLatestVerion을 통해 현재 서버에 업로드된 어플리케이션의 최신 버전 정보를 확인한다. 만약 해당 콜백에 대한 버전 값이 클라이언트에 버전과 동일하다면 0, 0이상이면 최신 업데이트 정보가 있기 때문에 LoginUI의 OpenUpdatePop()을 호출하여 메시지를 띄운다.

```
                    Debug.Log("GoogleHash - " + Backend.Utils.GetGoogleHash());
        #endif
    }
    else
    {
        Debug.LogError("Fail to Initialize Backend : " + bro);
    }
}

// Update is called once per frame
void Update()
{
    Backend.AsyncPoll();
}

private void GetVersionInfo()
{
    Enqueue(Backend.Utils.GetLatestVersion, callback =>
    {
        if (callback.IsSuccess() == false)
        {
            Debug.LogError("Failed to get the version information.₩n" + callback);
            return;
        }

        var version = callback.GetReturnValuetoJSON()["version"].ToString();

        Version server = new Version(version);
        Version client = new Version(Application.version);

        var result = server.CompareTo(client);
        if (result == 0)
        {
        // 0 이면 두 버전이 일치
            return;
        }
        else if (result < 0)
        {
        // 0 미만이면 server 버전이 client 이전 버전
            return;
        }
        else
        {
        // 0보다 크면 server 버전이 client 이후 버전
            if (client == null)
            {
                Debug.LogError("클라이언트 버전정보가 null 입니다.");
                    return;
            }
        }

    LoginUI.GetInstance().OpenUpdatePopup();
    });
}
```

3. Server Manager

Server Manager.cs

BackendTokenLogin()

서버 로그인과 동시에 서버는 클라이언트에게 토큰을 발급한다. 만약, 로그인 버튼 클릭 시 토큰이 있을 경우 ServerManager는 해당 함수를 통해 토큰의 유무와 토큰의 정당성을 파악하는데 토큰이 정당할 경우 callback.IsSuccess() = true가 된다. 만약 토큰이 없거나 정당하지 못할 경우에 토큰 로그인이 실패한다.

CustomeLogin()

입력된 ID와 PW를 통해 서버에 로그인을 요청하고 실패/성공을 콜백하는 함수이다. Backend.Bmember.CustomLogin을 통해 요청하고 콜백 함수를 통해 로그인 성공 유무를 판단한다. 성공할 경우 사용자 사전 작업을 진행하고, 실패할 경우 실패의 이유를 콜백한다.

```
public void BackendTokenLogin(Action<bool, string> func)
{
    Enqueue(Backend.BMember.LoginWithTheBackendToken, callback =>
    {
        if (callback.IsSuccess())
        {
            Debug.Log("Success Token Login");
            loginSuccessFunc = func;

            OnPrevBackendAuthorized();
            return;
        }

        Debug.Log("Failed Token Login₩n" + callback.ToString());
        func(false, string.Empty);
    });
}

// 커스텀 로그인
public void CustomLogin(string id, string pw, Action<bool, string> func)
{
    Enqueue(Backend.BMember.CustomLogin, id, pw, callback =>
    {
        if (callback.IsSuccess())
        {
            Debug.Log("Success Custom Login");
            loginSuccessFunc = func;

            OnPrevBackendAuthorized();
            return;
        }

        Debug.Log("Failed Custom Login₩n" + callback);
        func(false, string.Format(BackendError,
            callback.GetStatusCode(), callback.GetErrorCode(), callback.GetMessage()));
    });
}
```

3. Server Manager

Server Manager.cs

OnBackendAuthorized()

서버에게 해당 사용자 정보를 불러오도록 요청하는 함수이다.
Backend.Bmember.GetUserInfo를 통해 요청하고 콜백 받는다. 만약 !callback.IsSuccess() = false 라면 callback을 loginSuccessFunc대리자를 통해 저장하고 return한다.
만약 그렇지 않다면 userNickName과 userIndate를 최신화하고, 매칭 리스트를 초기화한다.

```
// 유저 정보 불러오기 사전작업
private void OnPrevBackendAuthorized()
{
    isLogin = true;

    OnBackendAuthorized();
}

// 실제 유저 정보 불러오기
private void OnBackendAuthorized()
{
    Enqueue(Backend.BMember.GetUserInfo, callback =>
    {
        if (!callback.IsSuccess())
        {
            Debug.LogError("Failed to get the user information.₩n" + callback);
            loginSuccessFunc(false, string.Format(BackendError,
                callback.GetStatusCode(), callback.GetErrorCode(), callback.GetMessage()));
            return;
        }
        Debug.Log("UserInfo₩n" + callback);

        var info = callback.GetReturnValuetoJSON()["row"];
        if (info["nickname"] == null)
        {
            LoginUI.GetInstance().ActiveNickNameObject();
            return;
        }

        userNickName = info["nickname"].ToString();
        userIndate = info["inDate"].ToString();

        if (loginSuccessFunc != null)
        {
            MatchManager.GetInstance().GetMatchList(loginSuccessFunc);
        }
    });
}
```

4. Login UI Manager

LoginUI.cs

Introduce

서버에 작업을 요청하는 API를 실행 시킬 수 있는 모든 UI들을 관리하는 매니저이다.

실행 주기
어플리케이션 시작과 동시에 실행되며, Match Scene으로 이동하는 순간 종료된다..

동작방식
서버에 요청하는 API를 실행뿐만아니라 모든 UI의 상호작용이 일어날 경우 LoginUIManager에 의해 처리 된다. 초기 화면에서 Touch를 누를 때, Login 버튼을 누를 때, SignUp 버튼을 누를 때, 팝업을 닫을 때, 모두 LoginUIManager에 의해 처리 된다.

Start()

초기 화면에 필요한 UI Setting을 진행하고, 각 InputField를 설정해준다. TouchStart()를 호출하는 touchStart Object를 활성화하고, mainTitle Object를 활성화한다. 나머지는 모두 비활성화한다. 여기서, Fade라는 객체는 fadeIn/ fadeOut을 처리하는 객체로 Scene전환과 동일하게 부드러운 화면 전환을 위해 사용된다.

OpenUpdatePopup()

ServerManager에서 GetVersionInfo()를 통해 호출 되는 함수이며, updateObject를 활성화 시킨다. updateObject에는 상수형 문자열로 마켓 주소가 저장 되어있다. (기능만 구현)

```
void Start()
{
    //Setting Init UI Objects
    errorText = errorObject.GetComponentInChildren<TextMeshProUGUI>();
    touchStart.SetActive(true);
    mainTitle.SetActive(true);

    loginObject.SetActive(false);
    customLoginObject.SetActive(false);
    signUpObject.SetActive(false);
    errorObject.SetActive(false);
    nicknameObject.SetActive(false);

    loginField = customLoginObject.GetComponentsInChildren<TMP_InputField>();
    signUpField = signUpObject.GetComponentsInChildren<TMP_InputField>();
    nicknameField = nicknameObject.GetComponentInChildren<TMP_InputField>();

    subTitle.GetComponentInChildren<TextMeshProUGUI>().text = string.Format(VERSION_STR, Application.version);

    loadingObject = GameObject.FindGameObjectWithTag("Loading");
    loadingObject.SetActive(false);


    var fade = GameObject.FindGameObjectWithTag("Fade");
    if (fade != null)
    {
        fadeObject = fade.GetComponent<FadeAnimation>();
    }

}
public void OpenUpdatePopup()
{
    updateObject.SetActive(true);
}
```

4. Login UI Manager

LoginUI.cs

TouchStart()

초기 화면에서 Touch Obejct를 누르게 되면 호출 되는 함수로, Login 화면으로 전환되게 된다. 만약, BackendTokenLogin()을 통해 정당한 토큰이 있다면 로그인 화면을 생략하고 MatchScene으로 전환되며 MatchManager의 Handler를 Setting한다. 그렇지만, 정당한 토큰이 존재하지 않는다면, touch Object를 비활성화 후 loginObject를 활성화하여 Login화면으로 전환한다.

```
public void TouchStart() // Touch Start Button
{
    loadingObject.SetActive(true); // Loading Object Enable
    ServerManager.GetInstance().BackendTokenLogin((bool result, string error) =>
    {
        Dispatcher.Current.BeginInvoke(() =>
        {
            if (result)
            {
                ChangeLobbyScene();
                MatchManager.GetInstance().SettingHanddler();
                return;
            }
            loadingObject.SetActive(false); // Loading object Disable

        if (!error.Equals(string.Empty))
            {
                errorText.text = "Failed to get User Data₩n₩n" + error;
                errorObject.SetActive(true);
                return;
            }
            touchStart.SetActive(false);
            loginObject.SetActive(true);

        });
    });
}
```

4. Login UI Manager

LoginUI.cs

Login()

사용자의 ID와 PW를 모두 입력 후, Login Button을 누르면 호출 되는 함수로, 서버에 로그인을 요청하는 API를 실행 한다. 만약 로그인에 실패할 경우 로그인 에러 메시지 팝업을 활성화하며, 에러 내용 callback.errMessage를 표시한다. 만약 로그인에 성공하는 경우 MatchScene으로 전환되며 MatchManager는 각 상황에서 사용 될 핸들러를 Setting한다.

Dispatcher?

Unity의 경우 싱글 스레드 기반으로, 서버 요청에 의한 응답이 동시에 UI에 접근하게 되면 멀티 쓰레드 요류가 발생한다. Dispatcher는 분리되어 작업되는 쓰레드의 내용을 순차적으로 하나씩 직렬처리 하도록 한다.

```
public void Login() // Touch User Login Button
{
    if (errorObject.activeSelf)
    {
        return;
    }
    string id = loginField[ID_INDEX].text;
    string pw = loginField[PW_INDEX].text;

    if (id.Equals(string.Empty) || pw.Equals(string.Empty))
    {
        errorText.text = "ID 혹은 PW 를 먼저 입력해주세요.";
        errorObject.SetActive(true);
        return;
    }

    loadingObject.SetActive(true);
    ServerManager.GetInstance().CustomLogin(id, pw, (bool result, string error) =>
    {
        Dispatcher.Current.BeginInvoke(() =>
        {
            if (!result)
            {
                loadingObject.SetActive(false);
                errorText.text = "로그인 에러₩n₩n" + error;
                errorObject.SetActive(true);
                return;
            }
            ChangeLobbyScene();
            MatchManager.GetInstance().SettingHanddler();
        });
    });
}
```

5. MatchManager

MatchManager.cs

Introduce

서버와 클라이언트간 매칭과 인게임 메세지 전달을 위한 **API**, 유저가 만든 매칭 룸간 매칭이 이루어지며 접속 된 매칭 서버에서 메세지를 주고 받는다.

실행주기

ServerManager와 동일하게 어플리케이션 시작과 동시에 실행되며 어플리케이션 종료 전까지 종료되지 않지만 Match Scene 이후 동작한다.

동작방식

Login 버튼클릭 이후에 **MatchManager**는 이벤트핸들러를 설정하며 본격적으로 실행 된다. **Matching** 버튼 클릭시 유저는 매칭 룸을 생성하게 되며, 동시에 유저는 매칭 서버와 접속을 요청하게 된다. 접속이 정상 처리 될 경우, 매칭 룸의 정보가 UI로 나타나며 현재 게임 옵션을 확인할 수 있다. **Start** 버튼 클릭시 유저는 매칭 신청을 서버에게 요청하며, 매칭 서버는 만들어진 매칭 룸끼리 매칭을 이루어준다. 매칭이 정상 처리 될 경우, **Ingame Scene**으로 전환되고 유저들이 보내는 메세지를 받아 처리 한다. **InputManager**에 의해 변경되는 정보가 있을 경우에 메세지를 전달한다. 만약, 게임이 종료 되어 종료 요청이 들어오면 게임 결과창을 UI에 나타내며 동시에 매칭 서버와 접속을 종료한다.

구성

MatchManager.cs
- 요청에 대한 응답 값 핸들러 정의 및 비호스트/호스트 설정

BackendMatch.cs
- Match Server연결 및 Match Room 생성 신청 API

BackendInGame.cs
- InGame Server 연결 API

OnApplicationQuit()

어플리케이션이 종료 될 경우, 현재 매치 서버와 연결 되어 있다면 매치 서버와의 연결을 종료하는 LeaveMatchServer()를 호출한다.

```
public class MatchInfo
{
    public string title;              // 매칭 명
    public string inDate;             // 매칭 inDate (UUID)
    public MatchType matchType;       // 매치 타입
    public MatchModeType matchModeType; // 매치 모드 타입
    public string headCount;          // 매칭 인원
    public bool isSandBoxEnable;      // 샌드박스 모드 (AI매칭)
}

public List<MatchInfo> matchInfos { get; private set; } = new List<MatchInfo>();  // 콘솔에서 생성한 매칭 카드들의 리스트
public List<SessionId> sessionIdList { get; private set; }  // 매치에 참가중인 유저들의 세션 목록
public Dictionary<SessionId, MatchUserGameRecord> gameRecords { get; private set; } = null;  // 매치에 참가중인 유저들의 매칭 기록
public SessionId hostSession { get; private set; }  // 호스트 세션
private ServerInfo roomInfo = null;            // 게임 룸 정보
private string inGameRoomToken = string.Empty;  // 게임 룸 토큰 (인게임 접속 토큰)
public bool isReconnectEnable { get; private set; } = false;
public bool isConnectMatchServer { get; private set; } = false;
private bool isConnectInGameServer = false;
private bool isJoinGameRoom = false;
public bool isReconnectProcess { get; private set; } = false;
public bool isSandBoxGame { get; private set; } = false;
private int numOfClient = 2; // 매치에 참가한 유저의 총 수

Host

void Awake()
{
    if (instance != null)
    {
        Destroy(instance);
    }
    instance = this;
}

void OnApplicationQuit()
{
    if (isConnectMatchServer)
    {
        LeaveMatchServer();
        Debug.Log("ApplicationQuit - LeaveMatchServer");
    }
}

// Start is called before the first frame update
void Start()
{
    GameManager.OnGameReconnect += OnGameReconnect;
}
```

5. MatchManager

MatchManager.cs

SettingHanddler()

로그인 서버와 연결이 되면, MatchManager는 핸들러를 Setting하기 위해 SettingHanddler()를 호출한다. Match Server와 연결하고 요청하며 받는 콜백함수들을 추가적으로 커스터마이징 하고, InGame Server와 연결하고 요청하며 받는 콜백들을 추가적으로 커스터마이징할 수 있다.

SetHostSession()

게임이 시작 되기 전, 호스트 세션을 정하는 함수로 서버에서 게임 시작 패킷을 전송해 올 경우 GameSetup()메소드에 의해 호출 된다. sessionIdList를 입장순서로 정렬한 후, 서버에서 정한 superGamer인지 그리고 자신이 superGamer인지 판단 한다. 만약 자신이 superGamer 즉, 호스트라면 로컬 큐를 생성하고 매치 서버와 접속을 끊는다.

```
public void SettingHanddler()
{
    MatchMakingHandler();
    GameHandler();
    ExceptionHandler();
}

public bool IsMySessionId(SessionId session)
{
    return Backend.Match.GetMySessionId() == session;
}

public string GetNickNameBySessionId(SessionId session)
{
    return gameRecords[session].m_nickname;
}

private bool SetHostSession()
{
    // 호스트 세션 정하기
    // 각 클라이언트가 모두 수행 (호스트 세션 정하는 로직은 모두 같으므로 각각의 클라이언트가 모두 로직을 수행하지만 결과값은 같다.)

    Debug.Log("Enter to Host Session Setting.");
    // 호스트 세션 정렬 (각 클라이언트마다 입장 순서가 다를 수 있기 때문에 정렬)
    sessionIdList.Sort();
    isHost = false;

    // 내가 호스트 세션인지
    foreach (var record in gameRecords)
    {
        if (record.Value.m_isSuperGamer == true)
        {
            if (record.Value.m_sessionId.Equals(Backend.Match.GetMySessionId()))
            {
                isHost = true;
            }
            hostSession = record.Value.m_sessionId;
            break;
        }
    }

    Debug.Log("Are you Host? " + isHost);

    // 호스트 세션이면 로컬에서 처리하는 패킷이 있으므로 로컬 큐를 생성해준다
    if (isHost)
    {
        localQueue = new Queue<KeyMessage>();
    }
    else
    {
        localQueue = null;
    }

    // 호스트 설정까지 끝나면 매치서버와 접속 끊음
```

5. MatchManager

MatchManager.cs

SetSubHost()

GameHandler에 의해 추가 된, Backend.Match.OnChangeSuperGamer()의 응답 값으로 SetSubHost()가 호출 된다. 만약, 호스트가 접속이 끊겼을 경우 서브 호스트를 설정할 경우, 비호스트의 설정을 업데이트 하기위해서 호출 된다. 서버에서 응답값으로 새로운 SuperGamerSessionId를 발행하고, 해당 SessionId와 유저의 전체 SessionId를 확인하며 같을 경우 해당 SessionId의 supergamer설정을 true로 변경한다. 이후 로직은 SetHostSession()과 동일하다.

```
private void SetSubHost(SessionId hostSessionId)
{
    Debug.Log("Enter to Sub Session Setting.");
    // 누가 서브 호스트 세션인지 서버에서 보낸 정보값 확인
    // 서버에서 보낸 SuperGamer 정보로 GameRecords의 SuperGamer 정보 갱신
    foreach (var record in gameRecords)
    {
        if (record.Value.m_sessionId.Equals(hostSessionId))
        {
            record.Value.m_isSuperGamer = true;
        }
        else
        {
            record.Value.m_isSuperGamer = false;
        }
    }
    // 내가 호스트 세션인지 확인
    if (hostSessionId.Equals(Backend.Match.GetMySessionId()))
    {
        isHost = true;
    }
    else
    {
        isHost = false;
    }

    hostSession = hostSessionId;

    Debug.Log("Are you Host? " + isHost);
    // 호스트 세션이면 로컬에서 처리하는 패킷이 있으므로 로컬 큐를 생성해준다
    if (isHost)
    {
        localQueue = new Queue<KeyMessage>();
    }
    else
    {
        localQueue = null;
    }

    Debug.Log("Complete Host setting.");
}
```

5. MatchManager

MatchManager.cs

MatchMakingHandler()

Match Server와의 API 응답 값을 핸들링하는 핸들러이다.
만약, 매칭 서버에 접속요청을 하게 되면 콜백 응답값에 ProcessAccessMatchMakingServer()를 추가한다. 이를 통해 접속이 정상적으로 이루어졌는지 로그를 출력한다.
만약, 매칭 신청 작업요청을 하게 되면 콜백 응답값에 ProcessMatchMakingResponse()를 추가한다. 해당 응답을 추가함으로, 어떤 이유로 접속이 불가능한지 로그를 출력한다.

GameHandler()

MatchMakingHandler와 비슷하게 InGame Server와의 API 응답 값을 핸들링하는 핸들러이다.
인게임 서버에 접속을 요청을 하게 될 경우 응답값에 현재 재접속 상태인지, 정상적으로 연결되어있는지 상태인지 파악후 AccessInGameRoom()를 호출하는 동작을 추가한다. 이를 통해 유저의 접속 상태에 따라 Backend.Match.JoinGameRoom()을 통해 정상적으로 인게임 룸에 접속할 수 있다.

```
private void MatchMakingHandler()
{
    Backend.Match.OnJoinMatchMakingServer += (args) =>
    {
        Debug.Log("OnJoinMatchMakingServer : " + args.ErrInfo);
    // 매칭 서버에 접속하면 호출
    ProcessAccessMatchMakingServer(args.ErrInfo);
    };
    Backend.Match.OnMatchMakingResponse += (args) =>
    {
        Debug.Log("OnMatchMakingResponse : " + args.ErrInfo + " : " + args.Reason);
    // 매칭 신청 관련 작업에 대한 호출
    ProcessMatchMakingResponse(args);
    };
```

```
private void GameHandler()
{
    Debug.Log("Successfully Connencted GameHandler on MatchManager");
    Backend.Match.OnSessionJoinInServer += (args) =>
    {
        Debug.Log("OnSessionJoinInServer : " + args.ErrInfo);
    // 인게임 서버에 접속하면 호출
    if (args.ErrInfo != ErrorInfo.Success)
        {
            if (isReconnectProcess)
            {
                if (args.ErrInfo.Reason.Equals("Reconnect Success"))
                {
                //재접속 성공
                GameManager.GetInstance().ChangeState(GameManager.GameState.Reconnect);
                    Debug.Log("Reconnect Success");
                }
                else if (args.ErrInfo.Reason.Equals("Fail To Reconnect"))
                {
                    Debug.Log("Fail To Reconnect");
                    JoinMatchServer();
                    isConnectInGameServer = false;
                }
```

6. LobbyUIManager

LobbyUI.cs

Introduce

서버에 매칭 카드 생성 및 매칭 요청 작업을 요청하는 API를 실행 시킬 수 있는 모든 UI들을 관리하는 매니저이다.

실행 주기

Match Scene으로 전환 시 생성되며, InGame Scene으로 전환시 종료된다.

동작방식

서버에 요청하는 API를 실행뿐만아니라 모든 UI의 상호작용이 일어날 경우 LobbyUIManager에 의해 처리 된다. Matching 버튼을 누를 때, Start 버튼을 누를 때, 팝업을 닫을 때, 모두 LobbyUIManager에 의해 처리 된다.

구성

LobbyUI.cs
– Lobby Scene에서 일어나는 MatchServer 연결, 재연결, 재요청 UI의 모든 상호작용 작업

RoomUI.cs
– 연결 된 Match Server를 통한 매칭 룸 요청 UI 상호작용

Start()

MatchManager 오브젝트가 실행 중이면 유저의 닉네임을 Setting하는 SetNickName()를 호출시킨다. Loading Obecjt, errorText, macthInfoList를 설정하고 UI들을 초기 세팅한다.
ChangeTab()을 통해 활성화 시킬 수 있는 매치 모드를 파악한다. (현재 세팅 1 = 샌드박스)

SetNickName()

ServerManager에 저장되어 있는 userNickName을 통해 Nickname Text Object를 설정한다. 만약 빈 userNickName을 불러오면 에러로그를 표시하고 test123 name으로 대신한다.

```
void Start()
{
    if (MatchManager.GetInstance() != null)
    {
        SetNickName();
    }

    errorText = errorObject.GetComponentInChildren<Text>();

    loadingObject = GameObject.FindGameObjectWithTag("Loading");
    loadingObject.SetActive(false);

    var fade = GameObject.FindGameObjectWithTag("Fade");
    if (fade != null)
    {
        fadeObject = fade.GetComponent<FadeAnimation>();
    }

    matchInfotabList = tabObject.GetComponentsInChildren<TabUI>();
    int index = 0;
    foreach (var info in MatchManager.GetInstance().matchInfos)
    {
        matchInfotabList[index].SetTabText(info.title);
        matchInfotabList[index].index = index;
        index += 1;
    }

    for (int i = MatchManager.GetInstance().matchInfos.Count; i < matchInfotabList.Length; ++i)
    {
        matchInfotabList[i].gameObject.SetActive(false);
        Debug.Log("Disabled");
    }

    ChangeTab();
    modelObject.SetActive(true);
    errorObject.SetActive(false);
    requestProgressObject.SetActive(false);
    matchDoneObject.SetActive(false);
    reconnectObject.SetActive(false);
    readyRoomObject.SetActive(false);
}

private void SetNickName()
{
    var name = ServerManager.GetInstance().userNickName;
    if (name.Equals(string.Empty))
    {
        Debug.LogError("닉네임 불러오기 실패");
        name = "test123";
    }
    TextMeshProUGUI nickname = nickNameObject.GetComponent<TextMeshProUGUI>();
    RectTransform rect = nickNameObject.GetComponent<RectTransform>();

    nickname.text = name;
    rect.sizeDelta = new Vector2(nickname.preferredWidth, nickname.preferredHeight);
}
```

6. LobbyUIManager

LobbyUI.cs

ChangeTab()

현재 매치 룸의 정보를 나타내는 탭을 변경 할 때 사용 되는 UI 함수이다. 초기 매칭 정보는 개인전 모드로 MatchManager의 matchInfos[]의 index에 해당하는 UI tab.text를 활성화시킨다.

MatchReusestCallback()

매친 신청 작업을 요청하는 API의 응답 값으로 추가 된 ProcessMatchMakingResponse()을 통해 호출 된다. 만약 신청 작업이 실패 될 경우, requestProgressObject를 활성화 시킨다. 반대로, 신청 작업이 성공할 경우 requestProgressObject를 비활성화 후 return한다.

MatchDoneCallback();

ServerManager에 저장되어 있는 userNickName을 통해 Nickname Text Object를 설정한다. 만약 빈 userNickName을 불러오면 에러로그를 표시하고 test123 name으로 대신한다.

```
public void ChangeTab()
{
    int index = 0;
    foreach(var tab in matchInfotabList)
    {
        if (tab.IsOn() == true)
        {
            break;
        }
        index += 1;
    }
    var matchInfo = MatchManager.GetInstance().matchInfos[index];
    matchInfoText.text = string.Format(matchInfoStr, matchInfo.headCount, matchInfo.isSandBoxEnable.Equals(true) ? "활성화" : "비활성화",
        matchInfo.matchType, matchInfo.matchModeType);
}

참조 0개
public void RequestCancel()
{
    if (loadingObject.activeSelf || errorObject.activeSelf || matchDoneObject.activeSelf)
    {
        return;
    }
    MatchManager.GetInstance().CancelRegistMatchMaking();
}

참조 7개
public void MatchRequestCallback(bool result)
{
    if (!result)
    {
        requestProgressObject.SetActive(false);
        modelObject.SetActive(true);
        return;
    }

    requestProgressObject.SetActive(true);
    modelObject.SetActive(false);
}

참조 2개
public void MatchDoneCallback()
{
    requestProgressObject.SetActive(false);
    matchDoneObject.SetActive(true);
}
```

6. LobbyUIManager

RoomUI.cs

CreateRoomResult()

Lobby의 Matching 버튼을 클릭하면, 매칭 서버와 연결을 시도하며 대기 방 생성을 요청한다. MatchMakingHandler에 의한 Backend.Match.OnMatchMakingRoomCreate의 콜백을 통해서 CreateRoomResult()가 호출된다. 만약 성공한다면 RoomObject를 활성화 시켜 RoomUI를 활성화한다.

LeaveReadyRoom()

룸의 Cancel 버튼 클릭시 호출 되는 함수로, MatchManager의 LeaveMatchLoom()함수를 호출시켜 Match Server와의 연결을 종료한다.

```
public void OpenRoomUI()
{
    // 매치 서버에 대기방 생성 요청
    Debug.Log("Pushed OpenRoomUI Button");
    if (MatchManager.GetInstance().CreateMatchRoom() == true)
    {
        SetLoadingObjectActive(true);
        Debug.Log("Success create matching room");
    }
}

public void CreateRoomResult(bool isSuccess, List<MatchMakingUserInfo> userList = null)
{
    // 대기 방 생성에 성공 시 대기방 UI를 활성화
    if (isSuccess == true)
    {
        readyRoomObject.SetActive(true);
        Debug.Log("Open Room UI");
        if (userList == null)
        {
            SetReadyUserList(ServerManager.GetInstance().userNickName);
        }
        else
        {
            SetReadyUserList(userList);
        }
        SetLoadingObjectActive(false);
    }

    // 대기 방 생성에 실패 시 에러를 띄움
    else
    {
        SetLoadingObjectActive(false);
        SetErrorObject("대기방 생성에 실패했습니다.₩n₩n잠시 후 다시 시도해주세요.");
    }
}

public void LeaveReadyRoom()
{
    MatchManager.GetInstance().LeaveMatchLoom();
}
```

7. WorldManager

WorldManager.cs

Introduce

유저의 업데이트 정보를 API를 통해 서버로 발신하거나, 수신 된 메세지를 통해 인게임 정보를 수시로 업데이트하는 매니저.

실행 주기

Loading Scene에서 InGame Scene으로 전환되면서 실행되고, 인게임이 종료됨과 동시에 종료된다.

동작 방식

생성과 동시에 게임을 초기화한다. 플레이어 정보(체력, 위치, 공격력, 닉네임 등), StartCountSetting, GameManager의 InGame.EventHandler 등
게임이 진행되면, MatchManager에 의해 정해진 호스트는 비호스트가 불러오는 메시지를 OnReceive()를 통해 처리하지 않고 LocalQue에 담아놓고 처리한다.

OnReceive()

GameHandler로 추가한 Backend.Match.OnMatchRelay()의 응답 값으로 호출 되며, 브로드 캐스트한 메세지들을 처리하는 부분이 이에 해당한다.

OnRecieve는 메시지의 정보가 없거나, playe의 정보가 없거나, HOST가 아닌 경우 자기 자신이 보낸 브로드 캐스트 메시지는 무시한다.

메시지는 Type별로 다음과 같이 나뉘어 처리한다.
StartCount : InGameUIManger의 SetStartCount()를 호출해라
GameStart : 현재 GameManager의 상태를 InGame으로 변경해라
GameEnd : 게임 결과창을 출력하고, GameManager의 상태를 Over로 변경해라.
Key : ProcessKeyEvent()를 실행하라
PlayerMove ~ : TypeMessage의 Type에 따라 ProcessPlayerData()을 호출해라.

```
public void OnRecieve(MatchRelayEventArgs args)
{

    if (args.BinaryUserData == null)
    {
        Debug.LogWarning(string.Format("빈 데이터가 브로드캐스팅 되었습니다.\n{0} - {1}", args.From, args.ErrInfo));
        // 데이터가 없으면 그냥 리턴
        return;
    }
    Message msg = DataParser.ReadJsonData<Message>(args.BinaryUserData);
    if (msg == null)
    {
        //Debug.Log("CHECKING_POINT#1");
        return;
    }
    // Host가 아닐 때 내가 보내는 패킷은 받지 않는다.
    if ( !MatchManager.GetInstance().IsHost() && args.From.SessionId == userPlayerIndex)
    {
        return;
    }

    if (players == null)
    {
        Debug.LogError("Players 정보가 존재하지 않습니다.");
        return;
    }
    switch (msg.type)
    {
        case Protocol.Type.StartCount:
            StartCountMessage startCount = DataParser.ReadJsonData<StartCountMessage>(args.BinaryUserData);
            Debug.Log("wait second : " + (startCount.time));
            InGameUIManager.GetInstance().SetStartCount(startCount.time);
            break;
        case Protocol.Type.GameStart:
            InGameUIManager.GetInstance().SetStartCount(0, false);
            GameManager.GetInstance().ChangeState(GameManager.GameState.InGame);
            break;
        case Protocol.Type.GameEnd:
            GameEndMessage endMessage = DataParser.ReadJsonData<GameEndMessage>(args.BinaryUserData);
            SetGameRecord(endMessage.count, endMessage.sessionList);
            GameManager.GetInstance().ChangeState(GameManager.GameState.Over);
            break;

        case Protocol.Type.Key:
            KeyMessage keyMessage = DataParser.ReadJsonData<KeyMessage>(args.BinaryUserData);
            ProcessKeyEvent(args.From.SessionId, keyMessage);
```

7. WorldManager

WorldManager.cs

OnReceiveForLocal()

수신 된 메시지를 로컬에서 처리하기 위한 함수로, WorldManager의 Update()에서 호출되어 로컬 큐에 저장 되어있는 메시지를 처리한다. 즉, 호스트가 메시지를 처리할 경우 사용된다.

ProcessKeyEvent()

수신된 KeyMessage를 처리하기 위한 이벤트 함수로, OnReceive()와 OnReceiveForLocal()에서 호출 된다. 그러나 ProcessKeyEvent()는 호스트에 의해 처리 된다.

ProcessKeyEvent()는 호스트만이 사용하는 함수이며, 호스트의 InputManager에서 입력 된 메시지만 로컬 큐로 받아 처리하거나 비호스트들이 보낸 KeyMessage들을 통해 타 유저의 탱크를 동작을 처리하고, TypeMessage로 재가공하여 보내게 된다.

해당 비호스트들은 브로드캐스트를 통해 TypeMessage을 받게 되며OnReceive()로 타 유저의 탱크를 동작시킨다.

ProcessKeyEvent는 다음과 같은 동작을 처리한다.
KeyMessage의 Key값이 MOVE일 경우 해당 index(sessionId)값을 가진 message의 담긴 값을 정규화한 moveVector를 통해 playe의 SetMoveVector(moveVector)를 호출 후, TypeMessage로 재가공하여 브로드 캐스트한다.

KeyMessage의 Key값이 ATTACK일 경우 해당 index(sessionId)값을 가진 player의 Attack(attackPos)함수를 호출하며 해당 TypeMessage로 재가공하여 브로드 캐스트한다.

KeyMessage의 Key값이 NO_MOVE인 경우isNoMove = true로 변경하고, 해당 index(sessionId)를 가진 player의 SetMoveVector(Vector3.zero)를 호출 후, TypeMessage로 재가공하여 브로드 캐스트한다.

```
public void OnRecieveForLocal(KeyMessage keyMessage)
{
    ProcessKeyEvent(userPlayerIndex, keyMessage);
}
참조 2개
private void ProcessKeyEvent(SessionId index, KeyMessage keyMessage)
{
    if (MatchManager.GetInstance().IsHost() == false)
    {
        //호스트만 수행
        return;
    }
    bool isMove = false;
    bool isAttack = false;
    bool isNoMove = false;

    int keyData = keyMessage.keyData;

    Vector3 moveVecotr = Vector3.zero;
    Vector3 attackPos = Vector3.zero;
    Vector3 playerPos = players[index].GetPosition();
    if ((keyData & KeyEventCode.MOVE) == KeyEventCode.MOVE)
    {
        players[index].EngineMovementAudio();
        moveVecotr = new Vector3(keyMessage.x, keyMessage.y, keyMessage.z);
        moveVecotr = Vector3.Normalize(moveVecotr);
        isMove = true;
    }
    if ((keyData & KeyEventCode.ATTACK) == KeyEventCode.ATTACK)
    {
        attackPos = new Vector3(keyMessage.x, keyMessage.y, keyMessage.z);
        players[index].Attack(attackPos);
        isAttack = true;
    }

    if ((keyData & KeyEventCode.NO_MOVE) == KeyEventCode.NO_MOVE)
    {
        //players[index].EngineIdleAudio();
        isNoMove = true;
    }

    if (isMove)
    {
        players[index].SetMoveVector(moveVecotr);
        PlayerMoveMessage msg = new PlayerMoveMessage(index, playerPos, moveVecotr);
        MatchManager.GetInstance().SendDataToInGame<PlayerMoveMessage>(msg);
    }
```

8. InputManager

InputManager.cs

Introduce

유저의 조이스틱 정보를 서버에 전송시키거나, 유저의 탱크를 알맞게 업데이트 시키기 위한 매니저이다.

실행 주기

WorldManager와 동일하게 InGame Scene으로 전환 되면서 실행되고, 인게임이 종료됨과 동시에 종료 된다.

동작 방식

GameManager의 Ingame() 델리게이트를 호출하는 **GameManager**의 **InGameUpdate** 코루틴에 의해 매 프레임 호출 된다. 인풋 입력이 있는 순간마다 서버에게 **SendDataToInGame()**를 통해 **KeyMessage**를 전달케 한다.

MobileInput()

InGame() += MobileInput()을 통해 EventHandler에 추가시킨 MobileInput()은 GameManager의 Coroutine에 의해 .1초에 한번씩 호출 된다. virtualStick의 입력이 있을 경우에만 진행되며, 입력이 없을 경우에 isMove = false로 설정 후 return 된다.

만약, 입력이 있을 경우 virtualStick의 Horizontal, Vertical 값을 통해 Vector3를 만들어 movector로 저장한다. 해당 moveVector는 virtualStick의 방향이며, 유저의 탱크가 바라보는 방향과 일치하다.

유저 본인이 호스트일 경우 LocalQue에 메시지를 저장시키기 위해 AddMstToLocalQueue()을 호출한다. 만약, 비호스트이라면 MatchManager의 SendDataToInGame()통해 메시지를 브로드캐스팅하여 호스트 유저가 읽도록 한다.

```
void MobileInput()
{
    if (!virtualStick)
    {

        return;
    }

    int keyCode = 0;
    isMove = false;

    if (!virtualStick.isInputEnable)
    {
ITY_EDITOR
    isMove = false;


        return;
    }

    isMove = true;

    keyCode |= KeyEventCode.MOVE;
    Vector3 moveVector = new Vector3(virtualStick.GetHorizontalValue(), 0, virtualStick.GetVerticalValue());
    moveVector = Vector3.Normalize(moveVector);


    if (keyCode <= 0)
    {
        return;
    }

    KeyMessage msg;
    msg = new KeyMessage(keyCode, moveVector);
    if (MatchManager.GetInstance().IsHost())
    {
        MatchManager.GetInstance().AddMsgToLocalQueue(msg);
    }
    else
    {
        MatchManager.GetInstance().SendDataToInGame<KeyMessage>(msg);
    }
}
```

9. Player

Player.cs

SetMoveVector()

moveVector(유저 탱크가 움직여야하는 방향)을 vecto로 최신화하고, 만약 vecto가 (0,0,0)이라면 isMove = false로 설정 그렇지 않다면 isMove = true로 설정하여 탱크가 움직이는 중인지 스테이트 정보를 최신화한다.

Move()

Player class의 변수 moveVector를 이용한 Move(Vector3 var)함수 이다.
호스트가 메시지를 통해 받은 vector값을 적용하여 호스트가 로컬내에서 적 위치를 업데이트 할 때 사용 되며 Player의 Update()에 의해 호출 된다.

Move(Vector3 var)

직접적인 Direction이 올 경우 해당 var(Direction)을 통해 유저 탱크를 움직인다.
유저의 탱크는 Rigidbody.AddForce()를 사용하여 moveSpeed만큼 var방향으로 이동한다.
만약, 스테이트 정보인 isLive가 false라면 죽은 것이므로 return;한다.

Rotatae()

MoveVector가 Vector3.zero라는 것은 VirtualStick이 중앙에 위치함을 나타내며, 움직이지도 않고, 회전하지도 않다는 것과 동일하다. 그러므로 isRotate = false로 설정 후 return한다. 또한, VirtualStick의 Horizontal, Vertical 값이 kEpsilon값 이하여도 isRotate = false로 설정 후 return한다.(NoMoveMessage의 전송을 위함)

만약 유저가 위 두조건을 충족하지 않는다면 유저 탱크는 회전하는 것이므로,
playerModelObject 즉, 유저 탱크를 Lerp하게 회전시킨다.

```
#region 이동관련 함수
public void SetMoveVector(Vector3 vector)
{
    moveVector = vector;

    if (vector == Vector3.zero)
    {
        isMove = false;
    }
    else
    {
        isMove = true;
    }
}

public void Move()
{
    Move(moveVector);
}

public void Move(Vector3 var)
{
    if (!isLive)
    {
        return;
    }

    if (var.Equals(Vector3.zero))
    {
        isRotate = false;
    }

    else
    {
        // 회전 단위벡터가 Quaternion.kEpsilon 이상이면
        if (Quaternion.Angle(playerModelObject.transform.rotation, Quaternion.LookRotation(var)) > Quaternion.kEpsilon)
        {
            isRotate = true;
        }
        else
        {
            isRotate = false;
        }
    }

    if (!isLive)
    {
        return;
    }
    m_Rigidbody.AddForce(var * moveSpeed * Time.fixedDeltaTime, ForceMode.VelocityChange);
}

private void Rotate()
{
    if (moveVector.Equals(Vector3.zero))
    {
        isRotate = false;
        return;
    }
    if (Quaternion.Angle(playerModelObject.transform.rotation, Quaternion.LookRotation(moveVector)) < Quaternion.kEpsilon)
    {
        isRotate = false;
        return;
    }

    playerModelObject.transform.rotation = Quaternion.Lerp(playerModelObject.transform.rotation, Quaternion.LookRotation(moveVector), Time.deltaTime * rotSpeed);
}
```

REFERENCE

@Reference BACKEND SDK Tutorial https://developer.thebackend.io/unity3d/realtime/matchMake/tutorial/
@Reference UNITY Tutorial https://learn.unity.com/project/tanks-tutorial

※ 해당 프로젝트는 위 두 프로젝트를 참고하였으며 구조 파악 이후, 수정을 거쳐 완성 된 프로젝트임을 명시합니다.