Project P - 포트폴리오 문서

1. 프로젝트 개요

Project P 는 과거 모바일 게임 '성지키기 온라인'에서 영감을 받아 Unity 기반으로 제작된 실시간 멀티플레이 4 인 협동형 2D 디펜스 게임입니다.
플레이어들은 각자 개성 있는 직업을 선택하여 몰려오는 몬스터의 웨이브로부터 중앙의 음식을 방어해야 하며, 카드 선택을 통해 능력치를 강화해 나가는 로그라이크 요소를 가지고 있습니다.

GooglePlay Store 및 Steam 에 출시를 목표로 하고 있습니다.

- 개발 기간: 2025 년 6 월 9 일 ~ 진행 중
- 개발 인원: 2 명 (프로그래머 1, 아트 1)
- 플랫폼: Android, PC
- 기술 스택: Unity, C#, ASP.NET Core (REST API, WebSocket), MySQL

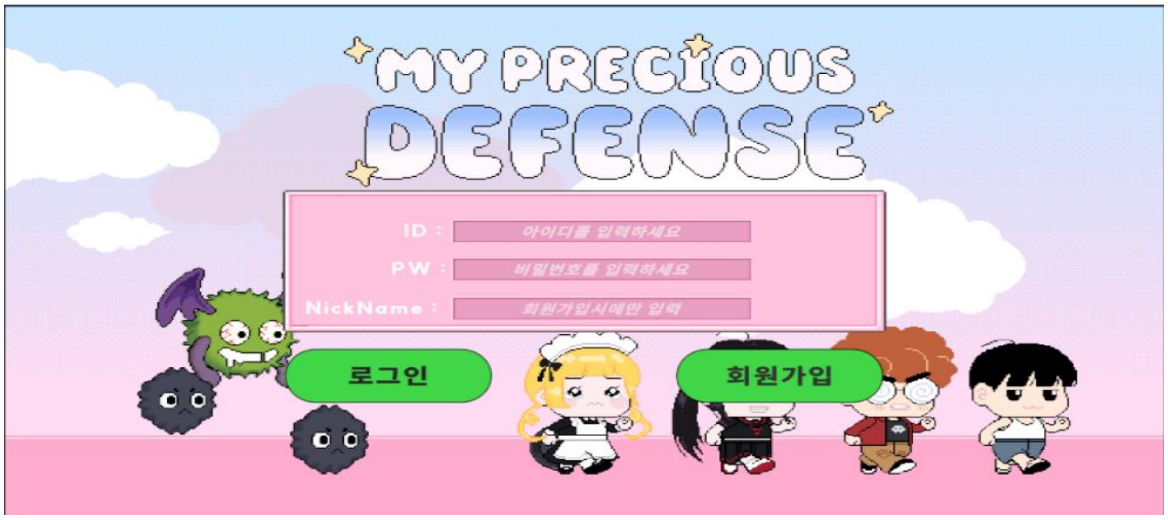
2. 아웃게임 시스템

- 회원가입 및 로그인 기능을 구현하여 사용자의 정보를 저장하고 로그인 후 게임 이용이 가능하도록 구성했습니다.
- RESTful API 서버 (ASP.NET Core) 와 MySQL 을 연동하여 사용자 정보를 안전하게 저장하고 인증 기능을 처리합니다.
- 로비 및 모드 선택 화면을 통해 빠른 매칭(Quick Match) 및 사용자 지정(Custom Match) 모드를 선택할 수 있습니다.
- 방 생성 후 캐릭터 선택 UI 가 제공되며, 가운데 캐릭터를 스와이프 방식으로 선택할 수 있도록 구현했습니다.
- 채팅 시스템을 도입하여 게임 시작 전 유저 간 커뮤니케이션이 가능합니다.
- 방장 전용 기능으로 "게임 시작", "유저 강퇴" 기능을 구현하여 매끄러운 룸 관리가 가능합니다.

- Room 기반 멀티플레이 구조를 도입하여 각 방에서 독립적으로 게임이 진행되며, 서버는 각 방의 상태를 메모리에서 관리합니다.

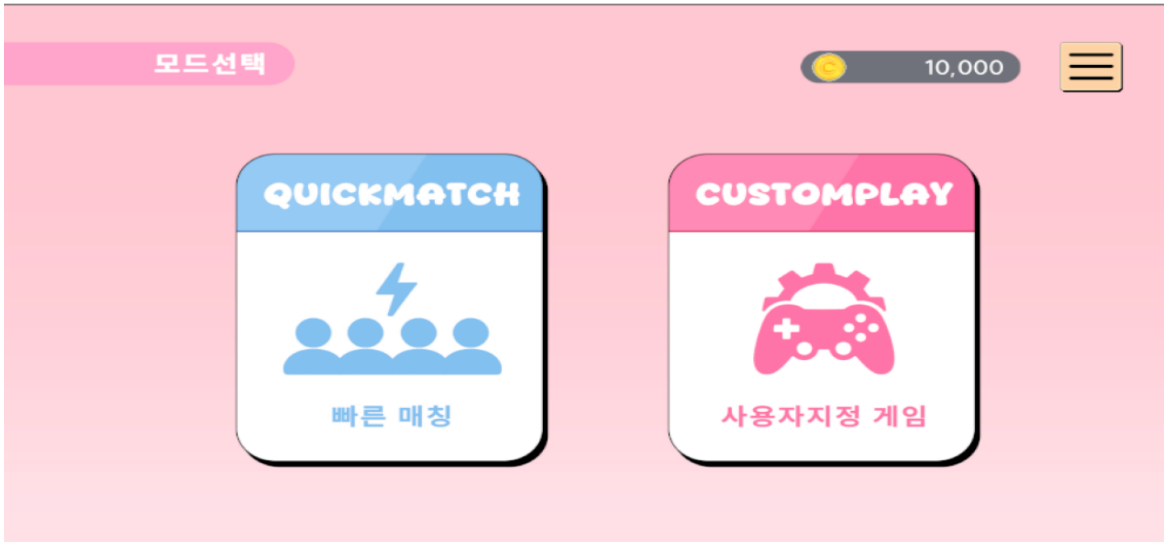
아웃게임 UI

로그인 및 회원가입



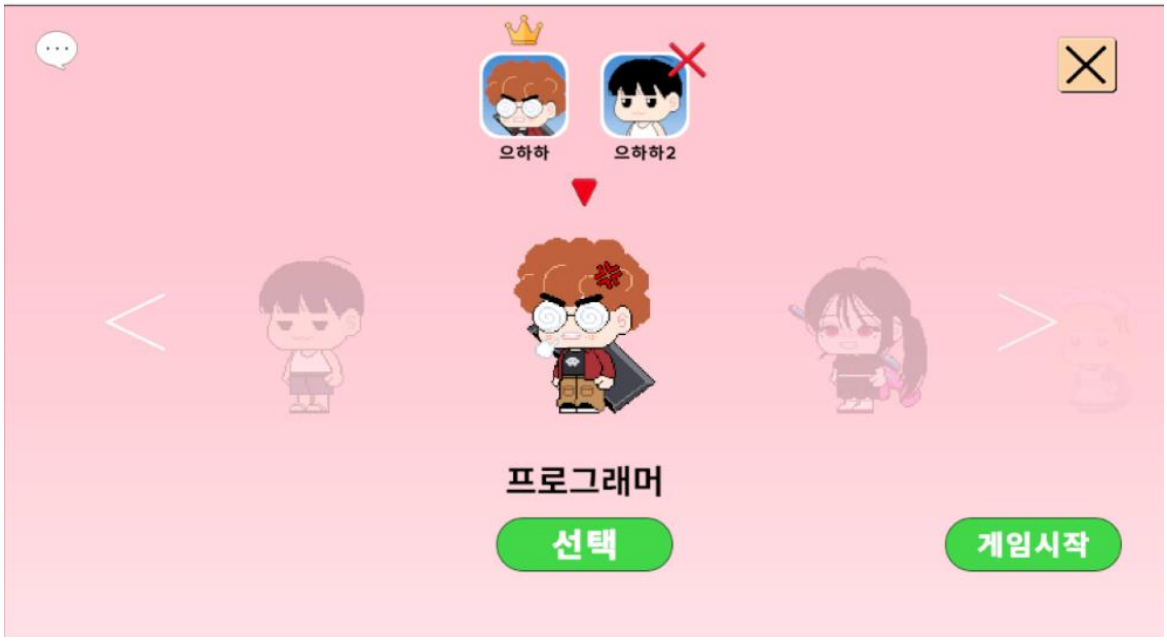

```
참조 1개
IEnumerator Login()
{
    if (ui_lock) yield break;
    ui_lock = true;
    var data = new Dictionary<string, string>
    {
        { "username", usernameInput.text },
        { "password", passwordInput.text }
    };

    yield return ApiManager.Instance.Post(
        "auth/login",
        data,
        onSuccess: (res) =>
        {
            var loginRes = JsonUtility.FromJson<ApiResponse.LoginResponse>(res);
            UserSession.Set(loginRes.userId, loginRes.nickname);
            SceneLoader.Instance.LoadScene("LobbyScene");
        },
        onError: (err) =>
        {
            showMessage($"로그인 실패: {err}");
        }
    );
    ui_lock = false;
}
```

모드 선택 화면




```
참조 1개
IEnumerator CreateRoom()
{
    if(ui_lock) yield break;
    ui_lock = true;
    var data = new Dictionary<string, string>
    {
        { "userId", UserSession.UserId }
    };

    yield return ApiManager.Instance.Post(
        "room/create",
        data,
        onSuccess: (res) =>
        {
            var parsed = JsonUtility.FromJson<ApiResponse.CreateRoomResponse>(res);
            RoomSession.Set(parsed.roomCode, parsed.hostId);
            var msg = new NetMsg
            {
                type = "create_room",
                playerId = UserSession.UserId,
                roomCode = parsed.roomCode,
                nickName = UserSession.Nickname,
            };
            string json = JsonConvert.SerializeObject(msg);
            WebSocketClient.Instance.Send(json);
        },
        onError: (err) =>
        {
            Debug.Log($"방 생성 실패: {err}");
        }
    );
    ui_lock = false;
}
```

캐릭터 선택 및 방장 기능




```
void TryGameStart()
{
    var message = new
    {
        type = "start_game",
        playerId = UserSession.UserId,
        roomCode = RoomSession.RoomCode,
    };
    string json = JsonConvert.SerializeObject(message);
    WebSocketClient.Instance.Send(json);
}
참조 1개
IEnumerator TryGameStartCoroutine()
{
    var data = new Dictionary<string, string>
    {
        { "roomcode", RoomSession.RoomCode},
    };

    yield return ApiManager.Instance.Post(
        "room/status",
        data,
        onSuccess: (res) =>
        {
        },
        onError: (err) =>
        {
            Debug.Log($"방 상태확인 실패: {err}");
        }
    );
    ui_lock = false;
}
```

```
IEnumerator TryOutRoom()
{
    var data = new Dictionary<string, string>
    {
        { "userId", UserSession.UserId },
        { "roomcode", RoomSession.RoomCode},
    };

    yield return ApiManager.Instance.Post(
        "room/out",
        data,
        onSuccess: (res) =>
        {
            var roomStatusResponse = JsonUtility.FromJson<ApiResponse.RoomOutResponse>(res);
            string hostId = roomStatusResponse != null ?  roomStatusResponse.hostId : null;
            var message = new
            {
                type = "out_room",
                playerId = UserSession.UserId,
                roomCode = RoomSession.RoomCode,
                hostId = hostId ?? ""
            };
            string json = JsonConvert.SerializeObject(message);
            WebSocketClient.Instance.Send(json);
        },
        onError: (err) =>
        {
            Debug.Log($"방 나가기 실패: {err}");
        }
    );
}
```

```
//입장시에 기본 플레이어 아이콘 생성
참조 1개
private void SetUpPlayerIcon()
{
    //startObj 활성화
    StartObj.SetActive(UserSession.UserId == RoomSession.HostId);

    if (RoomSession.RoomInfos.Count != players.Count)
    {
        // 삭제된 유저 정리
        foreach (var playerId in players.Keys.ToList())
        {
            if (!RoomSession.RoomInfos.Any(x => x.playerId == playerId))
            {
                Destroy(players[playerId].gameObject);
                players.Remove(playerId);
            }
        }
    }
    for (int i = 0; i < RoomSession.RoomInfos.Count; i++)
    {
        //업데이트
        if (players.ContainsKey(RoomSession.RoomInfos[i].playerId)) continue;

        //신규 플레이어 아이콘 생성
        GameObject playerIconObj = Instantiate(PlayerIconPrefab, PlayerIconParent);
        PlayerIcon icon = playerIconObj.GetComponent<PlayerIcon>();
        if (icon != null)
        {
            icon.SetInfo(RoomSession.RoomInfos[i]);
            players[RoomSession.RoomInfos[i].playerId] = icon;
        }
    }
}
```

채팅 시스템


```
private void OnClickChattingSend()
{
    if (ui_lock) return;
    ui_lock = true;
    ChattingInput.text = ChattingInput.text.Trim();
    if (ChattingInput.text != null & ChattingInput.text.Length > 0)
    {
        var message = new
        {
            type = "chat_room",
            playerId = UserSession.UserId,
            nickName = UserSession.Nickname,
            roomCode = RoomSession.RoomCode,
            message = ChattingInput.text
        };
        string json = JsonConvert.SerializeObject(message);
        WebSocketClient.Instance.Send(json);

        ChattingInput.text = null;
    }
    else
    {
        // 메시지를 스페이스바 혹은 공백으로 입력한 경우
        ChattingInput.text = null;
    }
    ui_lock = false;
}
```

3. 인게임 시스템

- 2D 멀티플레이 디펜스 게임으로, 최대 4 명이 함께 실시간으로 협력하여 적의 공격을 막는 구조입니다.

- 플레이어는 각각 직업(탱커, 딜러, 프로그래머)을 선택할 수 있으며, 직업별 고유 스킬과 역할이 다릅니다.

- WebSocket 기반의 실시간 통신을 통해 적 생성, 이동, 공격 및 플레이어 간 상호작용이 동기화됩니다.

- 서버는 각 방의 웨이브 상태를 주기적으로 스케줄링하여 적 소환 및 보스 등장 이벤트를 관리합니다.

- 적은 FSM(Finite State Machine)으로 구현되어, 이동, 추적, 공격, 죽음 상태를 자연스럽게 전환합니다.

- 플레이어는 전투 중 카드를 선택하여 캐릭터를 강화할 수 있으며, 카드 등급과 효과는 랜덤으로 등장합니다.

- 보스 몬스터는 별도의 FSM 과 패턴 데이터를 기반으로 공격을 수행하며, 체력 공유형 보스 메커니즘을 포함하고 있습니다.

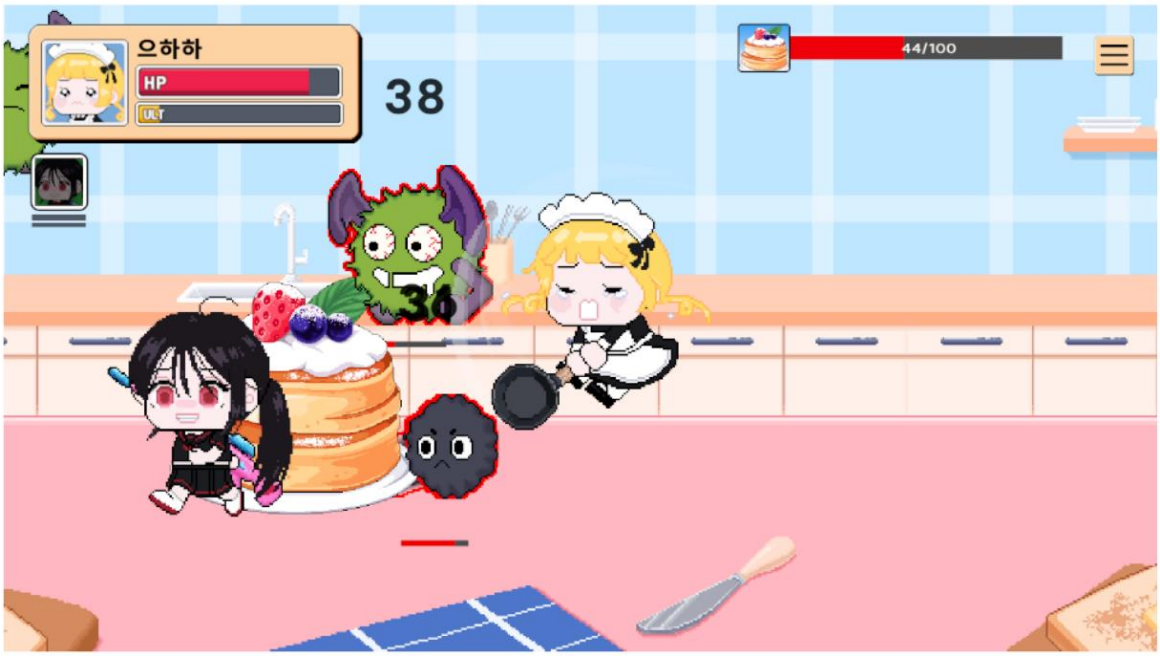
- 게임이 종료되면 서버는 해당 방의 리소스를 정리하고, 결과 데이터를 클라이언트에 전송합니다.

인게임 UI

카드 선택 UI


```
참조 1개
public void Init(List<CardData> cardList, float duration, int alivePlayerCount)
{
    this.duration = duration;

    Debug.Log($"[CardSelectPopup] 살아있는 플레이어: {alivePlayerCount}");

    // 카드 슬롯 초기화
    for (int i = 0; i < cardList.Count; i++)
    {
        var slot = cardSlots[i];
        slot.Init(cardList[i], OnCardClicked);
        slot.StartCoroutine((i + 1) * 0.1f);
        cardSlotMap[cardList[i].id] = slot;
    }

    confirmButton?.onClick.AddListener(OnConfirmClicked);
    confirmButton.interactable = false;
    timeBarImage.fillAmount = 1f;

    // 기존 Ready 슬롯 정리
    foreach (Transform child in readySlotTransform)
        Destroy(child.gameObject);

    // 서버에서 받은 살아있는 플레이어 수만큼 Ready 슬롯 생성
    for(int i = 0; i < alivePlayerCount; i++)
    {
        var slot = Instantiate(readyPrefab, readySlotTransform);
        readySlots.Add(slot.GetComponent<ReadyIcon>());
    }
}
```

실시간 전투 상황




```
Unity 메시지 | 참조 8개
protected virtual void Start()
{
    idleState = new IdleState(this);
    moveState = new MoveState(this);
    jumpState = new JumpState(this);
    attackState = new AttackState(this);
    deathState = new DeathState(this);

    ChangeState(idleState);
}

Unity 메시지 | 참조 0개
protected virtual void Update()
{
    // 사망 상태일 때는 입력 처리 안함
    if (isDead) return;

    // 본인 캐릭터일 때만 상태 업데이트 및 서버로 위치 전송
    if (!IsMyPlayer) return;

    SendMoveToServer();
    currentState?.Update();

    // 부활 입력 체크
    CheckRevivalInput();
}
```

관전자 모드


```
}
// 사망 처리 메서드
참조 1개
public virtual void Die()
{
    if (isDead) return;

    isDead = true;
    deathPosition = transform.position;
    // 무적 상태 해제
    StopInvulnerability();
    // 부활 이펙트 중단 (혹시 진행 중이었다면)
    if (RevivalEffectManager.Instance != null)
    {
        RevivalEffectManager.Instance.StopRevivalEffect(playerGUID);
    }

    //관전자 모드
    SpectatorManager.Instance.OnPlayerDied(playerGUID);

    if (IsMyPlayer)
    {
        ChangeState(deathState);
        SpectatorManager.Instance.StartSpectating();
    }
}
```

보스 등장 연출


```
private IEnumerator PlayBossIntro(float hp)
{
    GameManager.Instance.PauseGame();
    GameObject introUI = UIManager.Instance.getIntroUICanvas();
    CanvasGroup canvasGroup = introUI.GetComponent<CanvasGroup>();
    TextMeshProUGUI titleText = introUI.transform.Find("BossTitle").GetComponent<TextMeshProUGUI>();
    TextMeshProUGUI descText =  introUI.transform.Find("BossDesc").GetComponent<TextMeshProUGUI>();
    if(canvasGroup == null || titleText == null || descText == null) yield break;
    canvasGroup.alpha = 0f;

    UIManager.Instance.setActiveGameUICanvas(false);
    UIManager.Instance.setActiveIntroUICanvas(true);

    //카메라 팔로우
    // 1. 발 → 줌인
    yield return StartCoroutine(CameraFollow.Instance.MoveCamera(footPos.position, 2f, 0.8f));

    // 2. 얼굴 → 유지
    yield return StartCoroutine(CameraFollow.Instance.MoveCamera(facePos.position, 2f, 0.8f));

    // 3. 전체 → 줌아웃
    yield return StartCoroutine(CameraFollow.Instance.MoveCamera(bodyPos.position, 4f, 1.2f));

    yield return StartCoroutine(CameraFollow.Instance.MoveCamera(textShowPos.position, 5f, 0.2f));
    yield return StartCoroutine(PlayBossIntroText(canvasGroup, titleText, descText));

    //resume 하면 카메라도 알아서 update 됨.
    GameManager.Instance.ResumeGame();
    UIManager.Instance.setActiveGameUICanvas(true);
    UIManager.Instance.setActiveIntroUICanvas(false);
    UIManager.Instance.setActiveBossHpUI(true);
    GameManager.Instance.UpdateBossHPBar(hp,hp);
}
```

보스 패턴


```
public class BossDustWarningHandler : INetworkMessageHandler
{
    private readonly Dictionary<string, GameObject> bossDict = new ();
    참조 2개
    public string Type => "boss_dust_warning";
    참조 1개
    public BossDustWarningHandler(Dictionary<string, GameObject> bossDict)
    {
        this.bossDict = bossDict;
    }
    참조 3개
    public void Handle(NetMsg msg)
    {
        var bossObj = bossDict["boss"];
        if (bossObj == null) return;
        BossController bossController = bossObj.GetComponent<BossController>();
        if (bossController == null) return;
        bossController.PlayDustSummon();

        var spawnPositions = msg.spawnPositions;
        //TODO:마법진 생성 애니메이션 재생
        foreach (var spawnPosition in spawnPositions)
        {
            Debug.Log($"곧 먼지가 소환될 예정 x좌표 : {spawnPosition.Item1} , y좌표 : {spawnPosition.Item2} ");
            bossController.PlayDustSummonEffect(spawnPosition.Item1, spawnPosition.Item2 - 0.5f);
        }
    }
}
```

```
public class SpawnEnemyHandler : INetworkMessageHandler
{
    private readonly Dictionary<string, GameObject> Enemies = new ();
    private readonly WaveManager waveManager;

    참조 2개
    public string Type => "spawn_enemy";

    참조 1개
    public SpawnEnemyHandler(
        Dictionary<string, GameObject> Enemies,
        WaveManager waveManager)
    {
        this.Enemies = Enemies;
        this.waveManager = waveManager;
    }

    참조 3개
    public void Handle(NetMsg msg)
    {
        var pid = msg.enemyId;
        if (!Enemies.ContainsKey(pid))
        {
            var enemy = waveManager.SpawnEnemy(pid, msg.spawnPosX,msg.spawnPosY, msg.enemyDataId);
            Enemies[pid] = enemy;
        }
    }
}
```

4. 시스템

- APIManager.cs(웹서버 통신)

```
string GetUrl()
{
    var config = Resources.Load<ServerConfig>("ServerConfig");
    if (config == null)
    {
        Debug.LogError("ServerConfig.asset이 Resources 폴더에 없습니다!");
        return "";
    }
    string url = config.GetApiServerIP();
    return url;
}
참조 7개
public IEnumerator Post(string endpoint, Dictionary<string, string> formData, Action<string> onSuccess, Action<string> onError)
{
    WWWForm form = new WWWForm();
    foreach (var kv in formData)
    {
        form.AddField(kv.Key, kv.Value);
    }

    string url = $"{BaseUrl}/{endpoint}";
    UnityWebRequest www = UnityWebRequest.Post(url, form);
    www.useHttpContinue = false;
    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
    {
        onError?.Invoke(www.error);
    }
    else
    {
        onSuccess?.Invoke(www.downloadHandler.text);
    }
}
```

- WebSocketClient.cs (웹소켓 연결 및 통신)

```
public async Task TryConnect()
{
    string url = getUrl();
    Debug.Log($"웹소켓 연결 시도: {url}");
    websocket = new WebSocket(url);

    websocket.OnOpen += () =>
    {
        Debug.Log("WebSocket connected.");
    };

    websocket.OnError += (e) =>
    {
        Debug.LogError($"WebSocket error: {e}");
    };

    websocket.OnClose += (e) =>
    {
        Debug.Log("WebSocket closed.");
    };

    websocket.OnMessage += (bytes) =>
    {
        string message = System.Text.Encoding.UTF8.GetString(bytes);
        OnMessageReceived?.Invoke(message);
    };

    try
    {
        await websocket.Connect();
        Debug.Log("웹소켓 연결 완료!");
    }
```

```
참조 11개
public async void Send(string message)
{
    if (websocket != null && websocket.State == WebSocketState.Open)
    {
        if (websocket.State == WebSocketState.Open)
        {
            await websocket.SendText(message);
        }
        else
        {
            Debug.LogWarning("WebSocket not connected.");
        }
    }
    else
    {
        Debug.LogError("websocket 인스턴스가 null입니다.");
    }
}
```

- GameDataManager.cs (데이터 csv 파일 캐싱)

```
참조 1개
public void LoadAllData()
{
    //table 추가시에 여기다가 작업
    _tableDict["card_data"] = CsvLoader.Load<CardData>("DataExcels/card_data");
    _tableDict["player_data"] = CsvLoader.Load<PlayerData>("DataExcels/player_data");
    _tableDict["enemy_data"] = CsvLoader.Load<EnemyData>("DataExcels/enemy_data");
    _tableDict["wave_data"] = CsvLoader.Load<WaveData>("DataExcels/wave_data");
    _tableDict["shared_data"] = CsvLoader.Load<SharedData>("DataExcels/shared_data");
    _tableDict["bullet_data"] = CsvLoader.Load<BulletData>("DataExcels/bullet_data");
}

참조 3개
public Dictionary<int, T> GetTable<T>(string tableName)
{
    if (!_tableDict.TryGetValue(tableName, out var tableObj))
    {
        Debug.LogError($"[GameDataManager] 테이블 {tableName} 없음");
        return null;
    }

    var table = tableObj as Dictionary<int, T>;

    if (table == null)
    {
        Debug.LogError($"[GameDataManager] 테이블 {tableName} 타입 오류");
        return null;
    }

    return table;
}
```

- TextManager.cs (로컬라이징)

```
참조 4개
public enum Language
{
    Korean,
    English,
    Japanese
}

public Language CurrentLanguage = Language.Korean;

참조 1개
public void LoadLanguageFile(Language lang)
{
    string langFileName = lang switch
    {
        Language.Korean => "textkey",
        _ => "textkey"
    };

    TextAsset csvFile = Resources.Load<TextAsset>($"Localization/{langFileName}");

    if (csvFile == null)
    {
        Debug.LogError($"Localization file not found: {langFileName}");
        return;
    }

    _localizedTexts.Clear();

    using (StringReader reader = new StringReader(csvFile.text))
    {
        string line;
        bool isFirstLine = true;

        while ((line = reader.ReadLine()) != null)
        {
            if (isFirstLine)
            {
                isFirstLine = false; // skip header
                continue;
            }

            // , 로 split
            string[] tokens = line.Split(',');

            if (tokens.Length >= 2)
            {
                string key = tokens[0].Trim();
                string value = tokens[1].Trim();

                if (!_localizedTexts.ContainsKey(key))
                    _localizedTexts.Add(key, value);
            }
        }
    }
}
```

- PopupManager.cs

```
참조 2개
public void SetLoading(bool loading)
{
    isLoadingScene = loading;
    if (!isLoadingScene)
    {
        // 로딩 끝나면 보류된 팝업 실행
        while (deferredPopups.Count > 0)
        {
            var action = deferredPopups.Dequeue();
            action();
        }
    }
}

참조 5개
public T ShowPopup<T>(GameObject popupPrefab) where T : BasePopup
{
    if (isLoadingScene)
    {
        deferredPopups.Enqueue(() => CreatePopup<T>(popupPrefab));
        return null;
    }
    return CreatePopup<T>(popupPrefab);
}

참조 2개
private T CreatePopup<T>(GameObject prefab) where T : BasePopup
{
    if (popupRoot == null)
    {
        Debug.LogError("[PopupManager] popupRoot가 할당되지 않아 팝업을 띄울 수 없습니다!");
        return null;
    }
    var obj = Instantiate(prefab, popupRoot);
    var popup = obj.GetComponent<T>();
    popup.Open();
    return popup;
}
```

- InputHelper.cs

```
참조 3개
public static class MovementHelper
{
    참조 2개
    public static void Move(Rigidbody2D rb, float direction, float speed)
    {
        Vector2 velocity = rb.linearVelocity;
        velocity.x = direction * speed;
        rb.linearVelocity = velocity;
    }

    참조 1개
    public static void Jump(Rigidbody2D rb, float jumpForce)
    {
        Vector2 velocity = rb.linearVelocity;
        velocity.y = jumpForce;
        rb.linearVelocity = velocity;
    }
}

참조 4개
public static class InputManager
{
    public static FixedJoystick joystick;

    참조 3개
    public static float GetMoveInput()
    {
#if UNITY_EDITOR || UNITY_STANDALONE
        return Input.GetAxisRaw("Horizontal"); // 키보드 입력
#elif UNITY_ANDROID || UNITY_IOS
    if (joystick != null && Mathf.Abs(joystick.Horizontal) > 0.01f)
    {
        return joystick.Horizontal; // 조이스틱 입력
    }
    return 0f; // 입력 없음
#else
    return 0f;
#endif
```

5. 깃허브 주소

- 클라이언트 : https://github.com/smsm8087/DefenseGame
- 웹소켓서버 : https://github.com/smsm8087/DefenseGameWebSocketServer
- API 서버 : https://github.com/smsm8087/DefenseGameWebServer