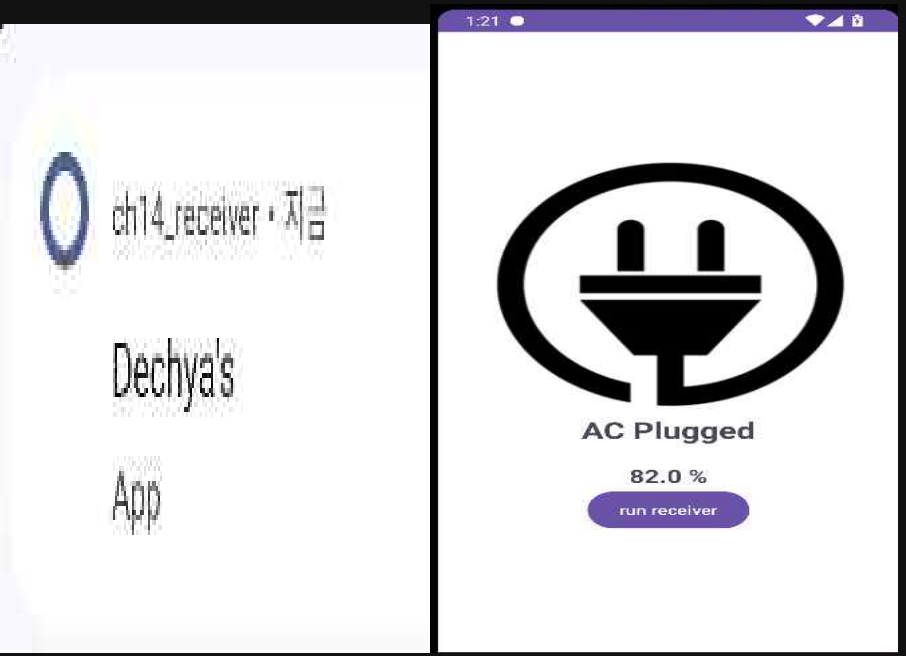
PORTFOLIO

寄稿者：キム・ヨンハン
携帯電話番号：010-9045-8663(KR)
生年月日：1999. 05. 27
e-mail : apple0426052@naver.com

目次

Dechya’s App

Moblie | Application | Android Studio



- アプリ紹介

このアプリは充電状態を画面に表示し、ユーザーがボタンを押すと「充電完了」や「充電中」などの通知が表示され、直感的なUIで誰でも簡単に充電状態を確認できます。

- 開発内容

1人開発

2025-02〜2024-03（5週間）

- プロジェクト本人の役割

プログラマー / 企画

主な実装

Script for Implement

バッテリーの充電状態を検出して通知を送信する機能スクリプト
MainActivity.kt

BroadcastReceiverを利用した通知(Notification)転送機能を持つスクリプト
MyReceiver.kt

MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        //add.....................
        val permissionLauncher = registerForActivityResult(
            ActivityResultContracts.RequestMultiplePermissions()
        ){
            if(it.all { permission -> permission.value == true}) {
                val intent = Intent( packageContext: this, MyReceiver::class.java)
                sendBroadcast(intent)
            }else{
                Toast.makeText( context: this, text: "permission denied...", Toast.LENGTH_SHORT).show()
            }
        }

        registerReceiver( receiver: null, IntentFilter(Intent.ACTION_BATTERY_CHANGED))!!.apply {
            when(getIntExtra(BatteryManager.EXTRA_STATUS, defaultValue: -1)){
                BatteryManager.BATTERY_STATUS_CHARGING -> {
                    when(getIntExtra(BatteryManager.EXTRA_PLUGGED, defaultValue: -1)){
                        BatteryManager.BATTERY_PLUGGED_USB -> {
                            binding.chargingResultView.text = "USB Plugged"
                            binding.chargingImageView.setImageBitmap(BitmapFactory.decodeResource(
                                resources, R.drawable.usb
                            ))
                        }
                        BatteryManager.BATTERY_PLUGGED_AC -> {
                            binding.chargingResultView.text = "AC Plugged"
                            binding.chargingImageView.setImageBitmap(BitmapFactory.decodeResource(
                                resources, R.drawable.ac
                            ))
                        }
                    }
                }else -> {
                    binding.chargingResultView.text = "No Plugged"
                }
            }

            val level = getIntExtra(BatteryManager.EXTRA_LEVEL, defaultValue: -1)
            val scale = getIntExtra(BatteryManager.EXTRA_SCALE, defaultValue: -1)
            val batteryPct = level / scale.toFloat() * 100
            binding.percentResultView.text = "$batteryPct %"
        }

        binding.button.setOnClickListener{
            if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU){
                if(ContextCompat.checkSelfPermission(
                        context: this,
                        permission: "android.permission.POST_NOTIFICATIONS"
                    ) == PackageManager.PERMISSION_GRANTED){
                    val intent = Intent( packageContext: this, MyReceiver::class.java)
                    sendBroadcast(intent)
                }else {
                    permissionLauncher.launch(
                        arrayOf("android.permission.POST_NOTIFICATIONS")
                    )
                }
            }else {
                val intent = Intent( packageContext: this, MyReceiver::class.java)
                sendBroadcast(intent)
            }
        }
    }
```

コードの説明

アプリが起動すると、現在の充電状態（USB、AC、未接続）とバッテリー残量が画面に表示され、ユーザーがボタンを押すとBroadcastReceiver（MyReceiver）を介して通知を送信します。

Android 13（TIRAMISU）以降では、通知権限の要求機能を含む、ユーザーが承認したかどうかに応じて通知を表示できるように設計されています。

このプロジェクトでは、計画とプログラミングを担当し、リアルタイムのバッテリー状態検出、BroadcastReceiverの活用、通知システムの実装、最新のAndroid権限処理方式の適用などの機能を開発しました。

MyReceiver.kt

```
package com.example.ch14_receiver

import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.media.AudioAttributes
import android.media.RingtoneManager
import android.net.Uri
import android.os.Build
import android.util.Log
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.NotificationCompat

class MyReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        Log.d( tag: "kkang", msg: "MyReceiver.............")
        val manager = context.getSystemService(AppCompatActivity.NOTIFICATION_SERVICE) as NotificationManager
        val builder: NotificationCompat.Builder

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            //26버전 이상
            val channelId = "one-channel"
            val channelName = "My Channel One"
            val channel = NotificationChannel(
                channelId,
                channelName,
                NotificationManager.IMPORTANCE_DEFAULT
            ).apply {
                //채널에 다양한 정보 설정
                description = "My Channel One Description"
                setShowBadge(true)
                val uri: Uri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION)
                val audioAttributes = AudioAttributes.Builder()
                    .setContentType(AudioAttributes.CONTENT_TYPE_SONIFICATION)
                    .setUsage(AudioAttributes.USAGE_ALARM)
                    .build()
                setSound(uri, audioAttributes)
                enableVibration( vibration: true)
            }
            //채널을 NotificationManager에 등록
            manager.createNotificationChannel(channel)

            //채널을 이용하여 builder 생성
            builder = NotificationCompat.Builder(context, channelId)
        } else {
            //26버전 이하
            builder = NotificationCompat.Builder(context)
        }

        builder.run {
            //알림의 기본 정보
            setSmallIcon(android.R.drawable.ic_notification_overlay)
            setWhen(System.currentTimeMillis())
            setContentTitle("Dechya's")
            setContentText("App")
        }
        manager.notify( id: 11, builder.build())
    }
}
```

コードの説明

アプリが特定の**Broadcast（ボタンをクリックするなど）**を受信すると、通知を作成してユーザーに表示します。

Android 8.0（Oreo）以降では、通知チャンネル（Notification Channel）を生成し、サウンド、振動、バッジ（Badge）などの設定が可能です。

ユーザーが通知を受信すると、アプリ名やメッセージなどの通知が表示されます。

本プロジェクトでは企画・プログラミングを担当し、

BroadcastReceiverの活用、通知(Notification)システムの実装、Androidの最新バージョン対応(Notification Channel適用)、ユーザーインターフェースの改善などの機能を開発しました。

プロジェクトを進行しながら感じたこと

今回のプロジェクトにより、KotlinとAndroid Studioを活用したアプリ開発経験を積むことができました。当初はKotlinの文法とAndroid開発環境が見慣れていましたが、UIを直接設計し、機能を実装しながら慣れてきました。特に、BroadcastReceiverを活用したNotificationシステムとデータストレージ管理機能を実装する過程が印象的であり、これによりAndroidアプリケーションの核心的な構造と開発フローを深く理解することができました。

Jetpack Composeを参考にして、より効率的なUI設計を学びながら、Kotlinを活用した現代的なAndroid開発方法を学ぶことができました。プロジェクトを進めながら、Android Studioの多様な機能を積極的に活用して開発生産性を高める方法も体得でき、これをもとに今後さらに多様な機能を備えたAndroidアプリケーションを開発してみたいという動機付けになりました。

Tower：Hero’s Journey

PC | マルチアクションRPG | Unity3D

プレイビデオリンク
https://www.youtube.com/watch?v=49m9B6BOzqk

ゲーム紹介

戦略的戦闘とミッションを遂行し、タワーを探索し、選択によるストーリー変化、NPCとの相互作用が特徴です。また、サーバーベースのマルチプレイで協力と競争が可能で、ピクセルスタイルの3Dグラフィックが戦略性を最大化するゲームです。

開発内容

4人開発

2024-09〜2024-11（9週間）

G-Star 2024参加

プロジェクト本人の役割

プログラマー / 企画 / チーム長

プレーヤーの機能と制御 / 武器システム / サーバー連携 / プロジェクト企画 / スケジュール管理

主な実装

Script for Implement

プレイヤーの入力処理と移動、対話、武器変更、UI連携するスクリプト

PlayerInputCheck.cs

ゲーム管理、ネットワーク同期、シンローディング、敵スポーン関連ロジックをカバーするスクリプト

GameManager.cs

ネットワークを利用したマルチプレイヤーキャラクターの状態を管理するスクリプト

PlayerState.cs

Mirror Networkingを活用したカスタムネットワークマネージャスクリプト

CustomNetworkManager.cs

Mirror Networkingを使用して敵をスポーンする機能を担当し、各ステージ（Stage）ごとに敵リストとスポーン位置を管理するスクリプト

EnemySpawnManager.cs

武器とアイテムデータを管理するスクリプト

WeaponBase.cs

PlayerInputCheck.cs

WeaponCollection.cs | PlayerInputCheck.cs | WeaponBase.cs* | Weapon.cs | PlayerState.cs | PlayerIntroAnim

기타 파일 — PlayerInputCheck

```
416     public void InputInteraction(InputAction.CallbackContext con)
417     {
418         if (!con.started) return;
419
420         if (npc_InRange == true)
421         {
422             animator.SetFloat("MoveX", 0);
423             animator.SetFloat("MoveY", 0);
424             PM.IsDialoging(true);
425             PlayerInteraction.SetDialogManager(dialogManager, dialogBox, npcData, isCollector);
426             PlayerInteraction.ui = Ui;
427         }
428         if (lootChest_InRange)
429         {
430             animator.SetFloat("MoveX", 0);
431             animator.SetFloat("MoveY", 0);
432
433             if (currentLootChest == null)
434             {
435                 Debug.LogWarning("currentLootChest가 null입니다.");
436                 return;
437             }
438
439             LootChest lootChest = currentLootChest.GetComponent<LootChest>();
440
441             if (lootChest == null)
442             {
443                 Debug.LogWarning("LootChest 컴포넌트를 currentLootChest에서 찾을 수 없습니다.");
444                 return;
445             }
446
447             if (lootChest.gameObject.name == "TutorialChest")
448             {
449                 if (questManager != null && questManager.IsQuestAccepted("t_Quest_001") && !questManager.IsQuestCompleted("t_Quest_001"))
450                 {
451                     lootUIManager.Initialize(PM.InventoryManager); //인벤토리 매니저 설정
452                     lootUIManager.OpenLootingUI(lootChest); //루팅 UI 열기
453                 }
454                 else
455                 {
456                     Debug.Log("t_Quest_001이 수락 중이 아니거나 완료되어 상호작용할 수 없습니다.");
457                 }
458             }
459             else
460             {
461                 lootUIManager.Initialize(PM.InventoryManager); //인벤토리 매니저 설정
462                 lootUIManager.OpenLootingUI(lootChest); //루팅 UI 열기
463             }
464
465         }
466     }
```

```
private void ItemUse(WeaponBase Base,int index)
{
    switch(Base.Item)
    {
        case ItemClass.Hpotion:
            CmdUseHpP(Base._Amount, PM.PS);
            if (quickSlotManager.inventory.QuickSlots[index].myItem.Amount > 1)
            {
                quickSlotManager.inventory.QuickSlots[index].myItem.DisplayAmount(-1);
            }
            else if(quickSlotManager.inventory.QuickSlots[index].myItem.Amount >= 1)
            {
                quickSlotManager.inventory.QuickSlots[index].myItem.DisplayAmount(-1);
                Destroy(quickSlotManager.inventory.QuickSlots[index].myItem.gameObject);
                quickSlotManager.inventory.QuickSlots[index].myItem = null;
                quickSlotManager.inventory.QuickSlots[index].CurrentItem = null;
            }
            break;
        case ItemClass.Mpotion:
            CmdUseMpP(Base._Amount, PM.PS);
            if(quickSlotManager.inventory.QuickSlots[index].myItem.Amount > 0)
            {
                quickSlotManager.inventory.QuickSlots[index].myItem.DisplayAmount(-1);
            }
            else if (quickSlotManager.inventory.QuickSlots[index].myItem.Amount >= 1)
            {
                quickSlotManager.inventory.QuickSlots[index].myItem.DisplayAmount(-1);
                Destroy(quickSlotManager.inventory.QuickSlots[index].myItem.gameObject);
                quickSlotManager.inventory.QuickSlots[index].myItem = null;
                quickSlotManager.inventory.QuickSlots[index].CurrentItem = null;
            }
            break;
    }
}
[Command]
public void CmdUseHpP(int amount,PlayerState PS)
{
    if(PS.CurrentHp + amount > PS.MaxHP)
    {
        PS.CurrentHp = PS.MaxHP;
    }
    else
    {
        PS.CurrentHp += amount;
    }
}
```

コードの説明

CmdUpdateInput、InputInteraction、CmdUseHpPは、ネットワーク同期を介してプレイヤーの入力、対話、フィットネス回復を処理する機能です。CmdUpdateInputは移動入力をサーバーに渡し、InputInteractionはオブジェクトとの対話を処理し、CmdUseHpPは体力回復アイテムの使用をサーバーで反映するように設計しました。

```
// 움직임
[Client]
public void OnMove(InputAction.CallbackContext context)
{
    if (!isAttack && !InventoryOpen && !QuestOpen && !dialogManager.isUpgrade && !PM.isDialog && !interactionInputcheck)
    {
        if(context.started || context.performed)
        {
            moveInput = context.ReadValue<Vector2>();
            PM.PS.SetSpeed(2f,this);
            PM.CmdUpdateInput(moveInput); // Command 호출
        }

        if(context.canceled)
        {
            moveInput = Vector2.zero;
            PM.PS.SetSpeed(0f,this);
            PM.CmdUpdateInput(moveInput); // Command 호출
        }
    }
    else
    {
        moveInput = Vector2.zero;
        PM.PS.SetSpeed(0f, this);
        PM.CmdUpdateInput(moveInput);
    }

}
```

GameManager.cs

```
public class GameManager : NetworkBehaviour
{
    public static GameManager Instance;
    public EnemySpawnManager ESM;
    public LootManager Loot;

    Camera Maincam;

    // NetworkIdentity 대신 netId를 SyncDictionary로 관리
    public readonly SyncDictionary<uint, uint> spawnedWeapons2 = new SyncDictionary<uint, uint>();

    public readonly SyncDictionary<string, List<uint>> SpawnEnemys = new SyncDictionary<string, List<uint>>();

    [SerializeField]
    public GameSctors gameSctors;

    // player관련
    private List<PlayerManager> Players = new List<PlayerManager>();

    public GameObject PlayerPrefab;

    public WeponCollection weponCollection;

    public EffectManager Em;

    public ResourcesManager RM;
    public LootManager lootManager;
    [SerializeDictionary("Round", "LootPoint")]
    public SerializedDictionary<int, List<LootPoint>> lootPointList = new SerializedDictionary<int, List<LootPoint>>();

    public TestSpawner TS;
    public BossSpawner BS;
```

```
    [Server]
    public void EnemySpawnInNewRound(string Round)
    {
        if(!isServer) return;

        StartCoroutine(CheckScene(Round));
    }

    IEnumerator CheckScene(string Round)
    {
        Scene scene = SceneManager.GetSceneByName(Round);

        while(!scene.isLoaded)
        {
            yield return null;
        }

        switch (Round)
        {
            case "1Round":
                ESM.SpawnEnemy(Stage.Stage2, "1Round");
                BS.Spawnboss();
                break;
            case "2Round":
                ESM.SpawnEnemy(Stage.Stage3, "2Round");
                break;
        }
    }
```

コードの説明

EnemySpawnInNewRound関数は、特定のラウンドの開始時にシーンがロードされるのを待ってから敵とボスをスポーンする役割を果たします。 SyncDictionaryは、サーバーとクライアント間の武器と敵データを同期させるために使用され、netIdをキーとして管理されます。SyncScenesWithClient関数は、クライアントが接続したときに、そのプレイヤーの進行状況に合わせてシーンを同期して一貫したゲーム環境を提供するように設計しました。

```
public void SyncScenesWithClient(NetworkConnection conn)
{
    foreach(var player in NetworkServer.spawned.Values)
    {
        if (player.gameObject.CompareTag("Player"))
        {
            if (player.GetComponent<PlayerState>().Round != "Tutorial")
            {
                Debug.Log("튜토리얼이 아니라 다른 씬 생성");
                TargetLoadScene(conn, player.GetComponent<PlayerState>().Round);
                break;
            }
            else if(player.GetComponent<PlayerState>().Round == "Tutorial")
            {
                gameSctors.SyncSectorStates(conn, "Tutorial");
            }
        }
    }
}
```

PlayerState.cs

```
public void animsetstate(PlayerManager PM)
{
    if (PM.isLocalPlayer) // 클라이언트가 권한이 있을 경우만 실행
    {
        if (PM.PS.WeaponEquip == false)
        {
            PM.PS.WeaponEquip = true;
            PM.PS.CurrentWeaponOBJ.SetActive(true);
            PM.PS.CurrentWeaponBackOBj.SetActive(false);
            PM.PS.CurrentAura.SetActive(true);
        }
        else if (PM.PS.WeaponEquip == true)
        {
            PM.PS.WeaponEquip = false;
            PM.PS.CurrentWeaponOBJ.SetActive(false);
            PM.PS.CurrentWeaponBackOBj.SetActive(true);
            PM.PS.CurrentAura.SetActive(false);
        }
        PM.PS.SetAnimState(PM); // 서버에 상태 변경 요청
    }
}

[Command]
public void SetAnimState(PlayerManager Pm)
{
    if(Pm.PS.WeaponEquip == false)
    {
        Pm.PS.WeaponEquip = true;
        Pm.PS.CurrentWeaponOBJ.SetActive(true);
        Pm.PS.CurrentWeaponBackOBj.SetActive(false);
        Pm.PS.CurrentAura.SetActive(true);
    }
    else if(PM.PS.WeaponEquip == true)
    {
        Pm.PS.WeaponEquip = false;
        Pm.PS.CurrentWeaponOBJ.SetActive(false);
        Pm.PS.CurrentWeaponBackOBj.SetActive(true);
        Pm.PS.CurrentAura.SetActive(false);
    }
}
}
```

```
[Command]
private void ChangeRound(PlayerState PS,string Round)
{
    PS.Round = Round;
    CmdChangeStage(PS,PM, PS.Round);
}

public void CmdChangeStage(PlayerManager Pm,string Round)
{
    GameManager.Instance.CmdUnloadScene(Pm,Round);
}

public void ChestSpawnInNewRound()
{
    CmdUnloadScene(this,PM);
}
[Server]
private void CmdUnloadScene(PlayerManager PM)
{
    Scene NextRound = SceneManager.GetSceneByName(PM.PS.Round);
    GameObject[] gameObjects = NextRound.GetRootGameObjects();
    switch (PM.PS.Round)
    {
        case "1Round":
            if (GameManager.Instance.lootManager.lootPoints.TryGetValue(2, out List<LootPoint> list))
            {
                GameManager.Instance.lootManager.SpawnChest(2);
            }
            break;
        case "2Round":
            if (GameManager.Instance.lootManager.lootPoints.TryGetValue(3, out List<LootPoint> list2))
            {
                GameManager.Instance.lootManager.SpawnChest(3);
            }
            break;
    }
}
```

コードの説明

SetState、ChangeRound、animsetstate関数は、プレイヤーの状態を初期化し、ラウンドを変更し、武器装着の有無に応じてアニメーション状態を更新する役割を果たす。 SetStateはプレイヤーの基本ステータスと初期武器状態を設定し、ChangeRoundは特定の条件を満たせばラウンドを変更して新しいステージに移動するようにし、animsetstateは武器装着の有無に応じてアニメーション状態を変更して視覚的反映を担当するように設計しました。

```
[SyncVar]
private bool isInit = false;

#region 스탯 초기 설정
public void SetState(PlayerState ps)
{
    if(isInit) return;

    ps.MaxHP = 100f;
    ps.MaxMP = 100f;
    ps.MaxXP = 100;
    ps.CurrentLevel = 1;


    ps.CurrentWeaponID = 0;
    ps.CurrentAtk = 10;
    ps.CurrentAttackSpeed = 1f;
    ps.CurrentZSkillATK = 0;
    ps.CurrentXSkillATK = 0;
    ps.CurrentZSkillCoolTime = 0;
    ps.CurrentXSkillCoolTime = 0;
    ps.WeaponEquip = false;
    ps.AnimEnd = true;

    ps.CurrentHp = 100f;
    ps.CurrentMp = 100f;
    ps.CurrentXp = 0;
    ps.Round = "Tutorial";
    isInit = true;
}

[Command]
public void SetSpeed(float speed,PlayerInputCheck PIC)
{
    CurrentMoveSpeed = speed;
}

#endregion
```

CustomNetworkManager.cs

```
public override void OnStartServer()
{
    base.OnStartServer();
    GameManager Ins = Instantiate(GameManagerPrefabs);
    NetworkServer.Spawn(GameManager Ins);
}

public override void OnStartClient()
{
    base.OnStartClient();
}

public override void OnStopClient()
{
    base.OnStopClient();
}
```

CustomNetworkManager.cs

기타 파일

```
// 클라가 접속, OnServerAddPlayer가 호출되기 전에 호출
public override void OnClientConnect()
{
    base.OnClientConnect();

}
// 서버


public override void OnServerAddPlayer(NetworkConnectionToClient conn)
{
    Debug.Log("플레이어 접속 ");
    // 클라이언트 준비 상태 설정

    base.OnServerAddPlayer(conn);

    if(conn.identity != null)
    {
        Debug.Log("플레이어 리스트에 추가");
        SceneManager.MoveGameObjectToScene(conn.identity.gameObject, SceneManager.GetSceneByName("Tutorial"));
        GameManager.Instance.gameSctors.player.Add(conn.identity.gameObject);
        GameManager.Instance.gameSctors.m_playerList.Add(conn.identity.netId);
    }

}
```

コードの説明

OnStartServerはサーバー起動時にGameManagerを作成し、ネットワークにスポーンし、OnServerAddPlayerはプレイヤーが接続したときにそのプレイヤーをゲームシーンに追加し、リストに登録してマルチプレイヤー環境を構築する役割をするように設計しました。

EnemySpawnManager.cs

```
[Server]
public void SpawnEnemy(Stage stage,string SceneName)
{
    Debug.Log("SpawnEnemy");
    List<uint> SpawnList = new List<uint>();
    spawnCount = 0;
    if(GameManager.Instance.gameActors.player.Count > 0)
    {
        if (GameManager.Instance.SpawnEnemys.TryGetValue(SceneName, out List<uint> list2))
        {
            Debug.Log("A");
            return;
        }
        else
        {
            for (int j = 0; j < GameManager.Instance.gameActors.player.Count; j++)
            {
                if (GameManager.Instance.gameActors.player[j].gameObject.scene.name == SceneName)
                {
                    SpawnPoint.TryGetValue(SceneName, out List<Transform> point2);
                    for(int i=0;i<point2.Count; i++)
                    {
                        if (spawnCount >= point2.Count)
                        {
                            GameManager.Instance.SpawnEnemys.Add(SceneName, SpawnList);
                            break;
                        }
                        Transform spawnPosition = null;
                        int index = Random.Range(0, point2.Count);
                        spawnPosition = point2[index];

                        var enemy = Instantiate(m_SpawnList[stage][Random.Range(0, m_SpawnList[stage].Count)], spawnPosition.position, Quaternion.identity);
                        Debug.Log(enemy.GetComponent<NetworkIdentity>().sceneId);

                        NetworkServer.Spawn(enemy.gameObject);
                        SpawnList.Add(enemy.netId);
                        spawnCount++;
                        Debug.Log("spawned");

                    }
                }
            }
        }
    }
    else if("Tutorial" == SceneName)
    {
        SpawnPoint.TryGetValue(SceneName, out List<Transform> point);
        for (int i = 0; i < point.Count; i++)
        {
            if (spawnCount >= point.Count)
            {
                GameManager.Instance.SpawnEnemys.Add(SceneName, SpawnList);
                break;
            }
            Transform spawnPosition = null;
            int index = Random.Range(0, point.Count);
            spawnPosition = point[index];

            var enemy = Instantiate(m_SpawnList[stage][0], spawnPosition.position, Quaternion.identity);

            if (enemy.CompareTag("Enemy"))
            {
                NetworkServer.Spawn(enemy.gameObject);
                SpawnList.Add(enemy.netId);
                Debug.Log("Spawn Tutorial Enemy");
                spawnCount++;
            }
        }
    }
}
```

コードの説明

SpawnEnemy関数は特定のステージとシーンで敵を生成してネットワーク上で同期する役割を果たし、ReSpawn関数は既に生成された敵を一定時間後に再度活性化する機能を実行する。どちらの関数も敵の配置と持続的な登場に関連しており、SpawnEnemyが初期生成の役割を担い、ReSpawnは消えた敵を一定時間後に再活性化してゲームの進行を維持する役割をするように設計しました。

```
public void ReSpawn(GameObject g , float time)
{
    StartCoroutine(Re(g,time));
    Debug.Log("REspawn");
}
private IEnumerator Re(GameObject g,float time)
{
    yield return YieldCache.WaitForSeconds(time);
    g.SetActive(true);
    ClientReSpawn(g);
}
[ClientRpc]
private void ClientReSpawn(GameObject g)
{
    g.SetActive(true);
}
```

WeaponBase.cs

```
public IEnumerator ChangeWeaponCoroutine(string weaponType, string weaponName)
{
    if (currentWeapon != null)
    {
        Destroy(currentWeapon);
    }

    GameObject newWeapon = Resources.Load<GameObject>($"Weapons/{weaponName}");

    if (newWeapon != null)
    {
        currentWeapon = Instantiate(newWeapon, weaponHolder.position, weaponHolder.rotation, weaponHolder);
        Debug.Log($"{weaponName} 무기가 장착되었습니다.");
    }
    else
    {
        Debug.LogWarning("해당 이름의 무기를 찾을 수 없습니다.");
    }

    yield return null;
}
```

```
public int quantity = 0;

public Sprite WeaponSprite;

public string itemDes_Text;

public int price;

public WeaponBase myItem { get; set; }

public InventorySlot activeSlot { get; set; }

public int _WeaponID => WeaponID;

public GameObject Weapon => WeaponPrefab;

public GameObject BackWeapon => BackWeaponPrefab;

public attribute Attribute => attribute;

public WeaponType Type => weaponType;

public ItemClass Item => itemClass;

public ItemType GetItemType => ItemType;

public int Atk
{
    get => amount;
    set => amount = value;
}
```

コードの説明

ChangeWeaponCoroutine関数は、既存の武器を削除し、新しい武器を装備し、Atkプロパティを介して攻撃力を調整することができます。さらに、DecreaseQuantity関数は武器の数量を管理し、変更時にイベントを呼び出してUIの更新などを処理するように設計しました。

```
    public event Action OnQuantityChanged;

    public void DecreaseQuantity(int amount)
    {
        quantity = amount;
        OnQuantityChanged?.Invoke(); // 수량이 변경될 때 이벤트 호출
    }
}
```

プロジェクトを進行しながら感じたこと

クライアントとサーバー間のデータ同期を維持することは予想よりも困難でした。特に、複数のプレイヤーが同時に接続する環境では、データミスマッチ問題（Desync）が頻繁に発生した。これを解決するために、Mirror NetworkingのSyncVar、SyncDictionary、Command / Rpcメソッドを活用し、クライアントとサーバー間のデータフローを明確に整理することが重要であることを学びました。機能を実装する際に必要な要素を決定する過程で、自ら開発すべきか、既存のソリューションを活用すべきかを慎重に判断することの重要性を実感し、それを通じて問題解決能力が一層向上しました。

最初はサーバーからすべてのデータを直接管理してクライアントに配信する方法で実装したが、多数のプレイヤーが接続するほど**ネットワーク負荷(Network Overhead**)が大きくなり、サーバーの応答速度が遅くなった。これを解決するために、サーバーで不要なメッセージ送受信を減らし、ローカル予測（Local Prediction）やデルタ同期（Delta Sync）などを適用する方法を試してみました。最適化されたネットワーク構造を設計することは、シームレスなマルチプレイ環境を構築するために不可欠であることに気付きました。

Lantern in Darkness

VR | ホラー放出ゲーム | Unity3D

プレイビデオリンク
https://www.youtube.com/watch?v=bm0iKNtl5d4

ゲーム紹介

VRを活用してホテル、学校、病院、墓地を脱出し、それぞれの場所でパズル要素を見つけて克服し、プレイを進行するホラー放出ゲームです。

開発内容

4人開発

2023-08〜2023-11（10週間）

G-Star 2023参加

紹介ビデオリンク：

プロジェクト本人の役割

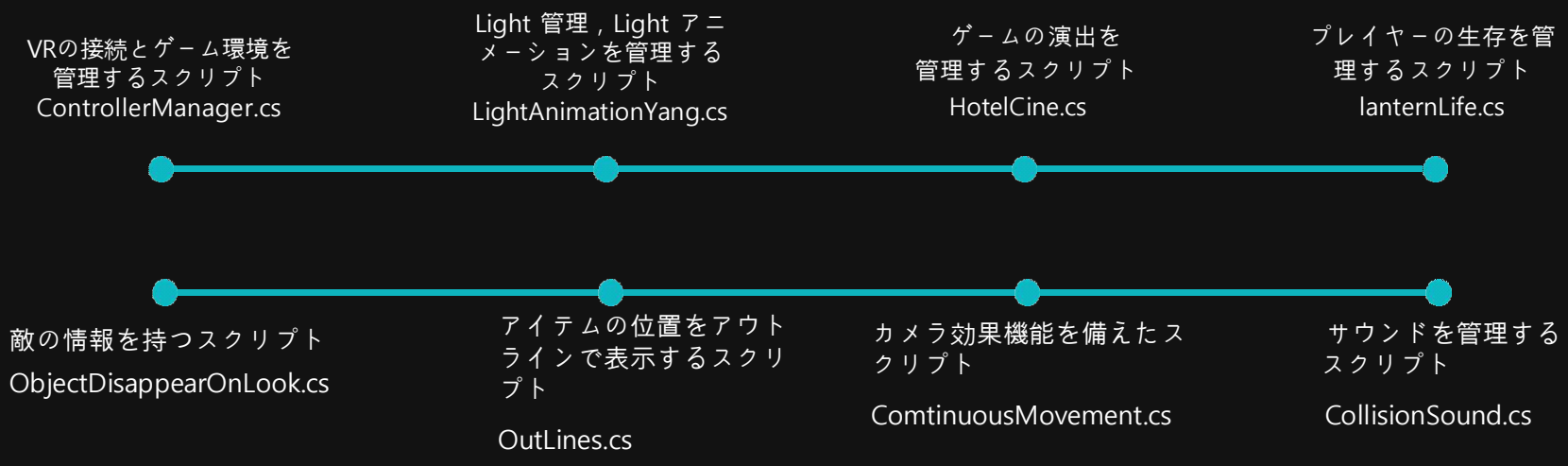
プログラマー / 企画 / チーム長

ユーザーインターフェース（UI） / ステージ管理 / VR連動 / 敵システム

プロジェクト企画 / 日程管理

主な実装

Script for Implement



lanternLife.cs

```
public class lanternLife : MonoBehaviour
{
    public Light lanternLight;
    private float initialIntensity;
    private float targetIntensity = 0.0f;
    [SerializeField] private float durationWithoutCollision = 90.0f;
    [SerializeField] private float durationWithCollision = 2.0f;
    private float elapsedTime = 0.0f;
    private bool collisionOccurred = false;

    void Start()
    {
        initialIntensity = lanternLight.intensity;
    }

    void Update()
    {
        elapsedTime += Time.deltaTime;
        float t;

        if (collisionOccurred)
        {
            t = elapsedTime / durationWithCollision;
            lanternLight.intensity = Mathf.Lerp(initialIntensity, 1.0f, t);
        }
        else
        {
            t = elapsedTime / durationWithoutCollision;
            lanternLight.intensity = Mathf.Lerp(initialIntensity, targetIntensity, t);
        }

        if (t >= 1.0f)
            lanternLight.intensity = collisionOccurred ? 1.0f : targetIntensity;
            enabled = false;

        if(lanternLight.intensity <=0f)
        {
            Debug.Log("Game Over");
        }
    }
```

```
void OnCollisionStay(Collision collision)
{
    if (collision.gameObject.tag == "Candle")
    {
        targetIntensity = 1.0f;
        collisionOccurred = true;
        elapsedTime = 0.0f;
    }
}
```

コードの説明

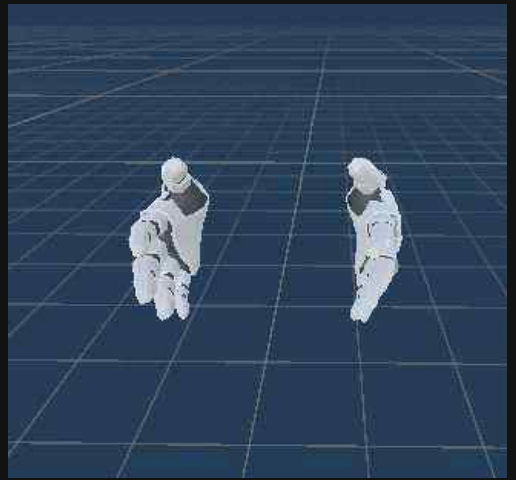
ランタンの光が徐々に減少する効果を実現し、「Candle」タグのオブジェクトと衝突した場合、光が再び明るくなるように設計されています。衝突の有無と経過時間に基づいて光の強度を調節し、光が完全に消えたら「Game Over」メッセージを出力してゲームの状態を表示しました。

ControllerManager.cs

```
public class ControllerManager : MonoBehaviour
{
    public LineRenderer leftControllerTrail;
    public LineRenderer rightControllerTrail;

    public Transform leftControllerTransform;
    public Transform rightControllerTransform;
    public Material customMaterial;
    public float trailWidth = 0.1f;

    void Start()
    {

        leftControllerTrail = leftControllerTransform.gameObject.AddComponent<LineRenderer>();
        leftControllerTrail.material = customMaterial;
        customMaterial.color = Color.yellow;
        leftControllerTrail.startWidth = trailWidth;
        leftControllerTrail.endWidth = 0;

        rightControllerTrail = rightControllerTransform.gameObject.AddComponent<LineRenderer>();
        rightControllerTrail.material = customMaterial;

        rightControllerTrail.startWidth = trailWidth;
        rightControllerTrail.endWidth = 0;
    }

    void Update()
    {
        Show();
    }

    void Show()
    {
        leftControllerTrail.SetPosition(0, leftControllerTransform.position);
        Vector3 leftPreviousPosition = leftControllerTransform.position - leftControllerTransform.forward * 0.1f;
        leftControllerTrail.SetPosition(1, leftPreviousPosition);

        rightControllerTrail.SetPosition(0, rightControllerTransform.position);
        Vector3 rightPreviousPosition = rightControllerTransform.position - rightControllerTransform.forward * 0.1f;
        rightControllerTrail.SetPosition(1, rightPreviousPosition);
    }
}
```

コードの説明

VR環境でユーザーのコントローラーが動く軌跡を視覚的に表示します。 LineRendererを使用して各コントローラの現在位置と前位置との間に線を描き、ユーザーが設定した材質と幅を介してカスタマイジングが可能になるように設定しました。



ObjectDisappearOnLook.cs

```
public class ObjectDisappearOnLook : MonoBehaviour
{
    private Camera vrCamera;
    public float maxDistance = 10f;
    private GameObject firsthitObject;

    private void Start()
    {
        vrCamera = Camera.main;
    }

    private void Update()
    {
        RaycastHit hit;
        Debug.DrawRay(vrCamera.transform.position, vrCamera.transform.forward, Color.red, maxDistance);
        if (Physics.Raycast(vrCamera.transform.position, vrCamera.transform.forward, out hit, maxDistance))
        {
            GameObject hitObject = hit.collider.gameObject;

            if (hitObject.name == "StairDoll")
                hitObject.GetComponent<DollRun>().SetDestinationToTarget();

            if(hitObject.name == "SneakDoll")
                hitObject.GetComponent<DOTweenAnimation>().DOPlayBackwards();

            if (hitObject.name == "WindowDollTrigger")
                hitObject.transform.GetChild(0).gameObject.SetActive(true);

            if (hitObject.name == "CreatureTrigger")
                hitObject.transform.GetChild(0).GetComponent<Animator>().enabled = true;
        }
    }
}
```

コードの説明

検出されたオブジェクトに応じてさまざまなアクションを実行し、各オブジェクトは移動またはアニメーションを再生し、特定の子オブジェクトをアクティブにしました。

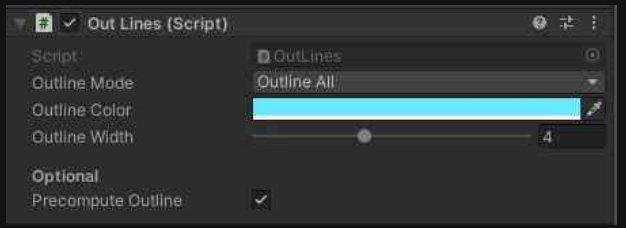
OutLines.cs

```
public class OutLines : MonoBehaviour {
    private static HashSet<Mesh> registeredMeshes = new HashSet<Mesh>();

    void UpdateMaterialProperties() {

        outlineFillMaterial.SetColor("_OutlineColor", outlineColor);

        switch (outlineMode) {
            case Mode.OutlineAll:
                outlineMaskMaterial.SetFloat("_ZTest", (float)UnityEngine.Rendering.CompareFunction.Always);
                outlineFillMaterial.SetFloat("_ZTest", (float)UnityEngine.Rendering.CompareFunction.Always);
                outlineFillMaterial.SetFloat("_OutlineWidth", outlineWidth);
                break;

            case Mode.OutlineVisible:
                outlineMaskMaterial.SetFloat("_ZTest", (float)UnityEngine.Rendering.CompareFunction.Always);
                outlineFillMaterial.SetFloat("_ZTest", (float)UnityEngine.Rendering.CompareFunction.LessEqual);
                outlineFillMaterial.SetFloat("_OutlineWidth", outlineWidth);
                break;

            case Mode.OutlineHidden:
                outlineMaskMaterial.SetFloat("_ZTest", (float)UnityEngine.Rendering.CompareFunction.Always);
                outlineFillMaterial.SetFloat("_ZTest", (float)UnityEngine.Rendering.CompareFunction.Greater);
                outlineFillMaterial.SetFloat("_OutlineWidth", outlineWidth);
                break;

            case Mode.OutlineAndSilhouette:
                outlineMaskMaterial.SetFloat("_ZTest", (float)UnityEngine.Rendering.CompareFunction.LessEqual);
                outlineFillMaterial.SetFloat("_ZTest", (float)UnityEngine.Rendering.CompareFunction.Always);
                outlineFillMaterial.SetFloat("_OutlineWidth", outlineWidth);
                break;

            case Mode.SilhouetteOnly:
                outlineMaskMaterial.SetFloat("_ZTest", (float)UnityEngine.Rendering.CompareFunction.LessEqual);
                outlineFillMaterial.SetFloat("_ZTest", (float)UnityEngine.Rendering.CompareFunction.Greater);
                outlineFillMaterial.SetFloat("_OutlineWidth", 0f);
                break;
        }
    }
    public enum Mode...

    public Mode OutlineMode...
```

```
void LoadSmoothNormals()
{
    foreach (var meshFilter in GetComponentsInChildren<MeshFilter>())
    {

        if (!registeredMeshes.Add(meshFilter.sharedMesh))
        {
            continue;
        }

        var index = bakeKeys.IndexOf(meshFilter.sharedMesh);
        var smoothNormals = (index >= 0) ? bakeValues[index].data : SmoothNormals(meshFilter.sharedMesh);

        meshFilter.sharedMesh.SetUVs(3, smoothNormals);

        var renderer = meshFilter.GetComponent<Renderer>();

        if (renderer != null)
        {
            CombineSubmeshes(meshFilter.sharedMesh, renderer.sharedMaterials);
        }
    }
    foreach (var skinnedMeshRenderer in GetComponentsInChildren<SkinnedMeshRenderer>())
    {
        if (!registeredMeshes.Add(skinnedMeshRenderer.sharedMesh))
        {
            continue;
        }

        skinnedMeshRenderer.sharedMesh.uv4 = new Vector2[skinnedMeshRenderer.sharedMesh.vertexCount];

        CombineSubmeshes(skinnedMeshRenderer.sharedMesh, skinnedMeshRenderer.sharedMaterials);
    }
}

List<Vector3> SmoothNormals(Mesh mesh)...
```

コードの説明

メッシュの表面を滑らかにして材質を適用するなど、さまざまな視覚効果を実現し、暗所でアイテムの位置を確認できるように設計しました。

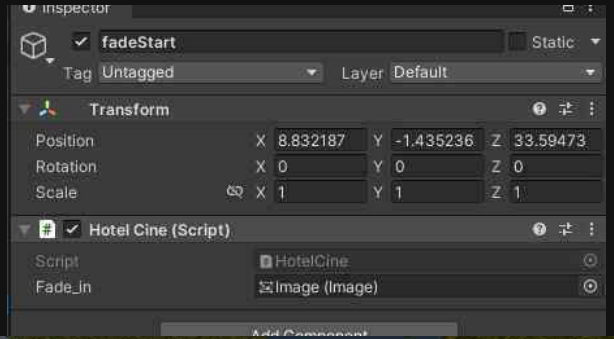


HotelCine.cs

```
public class HotelCine : MonoBehaviour
{
    public Image fade_in;

    void Start()
    {
        StartCoroutine(GoToHotel());
    }

    IEnumerator GoToHotel()
    {
        yield return new WaitForSeconds(3f);
        float targetAlpha = 1.0f;
        float duration = 2.0f;
        float elapsedTime = 0.0f;


        Color color = fade_in.color;
        color.a = 0.0f;
        fade_in.color = color;

        while (elapsedTime < duration)
        {
            elapsedTime += Time.deltaTime;
            float t = Mathf.Clamp01(elapsedTime / duration);
            color.a = Mathf.Lerp(0, targetAlpha, t);
            fade_in.color = color;
            yield return null;
        }

        yield return new WaitForSeconds(1.3f);
        SceneManager.LoadScene("Hotel");
    }
}
```

コードの説明

画面を2秒間徐々に暗くした後、「Hotel」シーンに切り替える機能を実装します。フェードイン効果とともにシーン切り替えが自然につながるように設定しました。



プロジェクトを進行しながら感じたこと

VR開発に初めて触れる中で、未知の環境に多くの挑戦がありましたが、実際の実装プロセスを通じて有益な経験と深い学びを得ることができました。この過程は、私に新しい技術への興味を抱かせる契機となりました。

機能を実装する際に必要な要素を決定する過程で、自ら開発すべきか、既存のソリューションを活用すべきかを慎重に判断することの重要性を実感し、それを通じて問題解決能力が一層向上しました。

チームメンバーと協力しながら効率的にエラーを克服することで、デバッグに対する深い理解を培うことができました。このような協力経験は、私の開発スキル向上に大きく貢献しました。

Goupong Logistics

PC | シミュレーションゲーム | Unity2D

プレイビデオリンク
https://www.youtube.com/watch?v=szpZOQXXap0

ゲーム紹介

シミュレーションを通じて全国各地を巡り、各地域の特産品を購入・販売して最高の利回りを見つけてスコアを記録するゲームです。

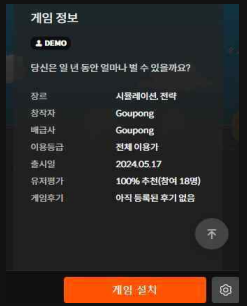
開発内容

3人開発

2024-03〜2024-05（7週間）

PlayX4 2024に参加

2024-05 Stoveリリース



プロジェクト本人の役割

プログラマー / チーム長

ユーザーインターフェース（UI）/ 取引システム / ゲームマネージャ管理 / ランキングシステム

プロジェクト企画 / 日程管理

主な実装

Script for Implement

全体的なゲームフローを管理するスクリプト
GameManager.cs

チャレンジ実績の達成可否を管理するスクリプト
AchievementManager.cs

ゲームのスコアを管理するスクリプト
Ranking.cs

ゲーム内での取引システムを管理するスクリプト
tradeGoods.cs

ソング情報とサウンドを管理するスクリプト
SoundManager.cs

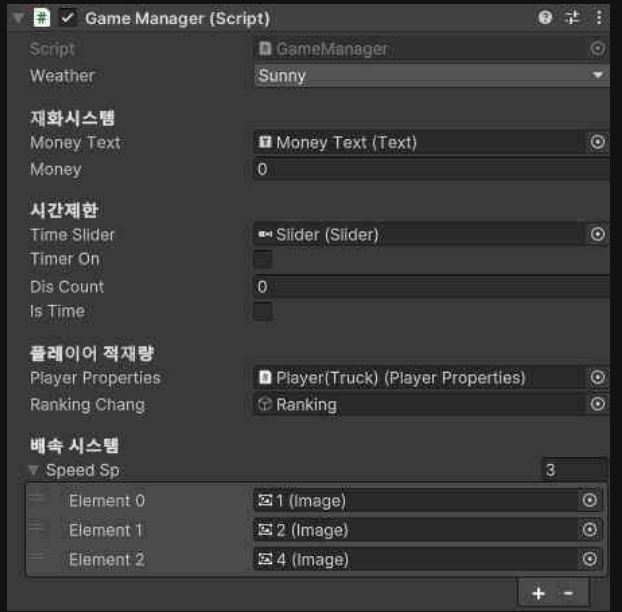
GameManager.cs

```
public class GameManager : MonoBehaviour
{
    public static GameManager Instance;

    public enum Weather{...}
    public Weather weather;

    public void TimeTest()
    {
        if (TimerOn == true && !coroutineRunning && isTime == true)
        {

            if (timeSliderval < maxvalue)
            {
                CancelInvoke("DisMoney");
                if(playerProperties.player == Player.Truck)
                {
                    timeSliderval += ((playerProperties.currentPlayerSpeed / playerProperties.currentPlayerSpeed) * (Time.deltaTime * 0.4f));
                    timeSlider.value = timeSliderval;
                }
                if (playerProperties.player == Player.Ship)
                {
                    timeSliderval += ((playerProperties.currentPlayerSpeed / playerProperties.currentPlayerSpeed) * (Time.deltaTime * 0.5f));
                    timeSlider.value = timeSliderval;
                }
                if (playerProperties.player == Player.Airplane)
                {
                    timeSliderval += ((playerProperties.currentPlayerSpeed / playerProperties.currentPlayerSpeed) * (Time.deltaTime * 1f));
                    timeSlider.value = timeSliderval;
                }
                DisCount = 0;
                dismoney = 2000;
            }
            else
            {
                coroutineRunning = true;
                Dis = true;
                InvokeRepeating("DisMoney", 0f, 3f);
                StartCoroutine(Wait());
            }
        }
    }

    [Header("재화시스템")]
    public Text moneyText;
```

```
public void MoneyChange()
{
    var animator = moneyText.GetComponent<Animator>();
    animator.Play("재화", 0, 0);
    SoundManager.Instance.PlaySFX("돈소리");
}

IEnumerator Wait()
{
    yield return null;
    curMonth += 1;
    timeSliderval = 0;
    timeSlider.value = timeSliderval;
}

public void TimeVal()
{
    timeSliderval = 0;
    timeSlider.value = timeSliderval;
}

public void Ranking()
{
    if(curMonth == 13)
    {
        Debug.Log("Ranking");
        RoadMap.Instance.chang[0].gameObject.SetActive(false);
        RoadMap.Instance.chang[1].gameObject.SetActive(false);
        Time.timeScale = 0f;
        RankingChang.SetActive(true);
        RandomImage.Instance.RanCreate();
        curMonth = 1;
    }
}

public void RankingExit()
{
    RankingChang.SetActive(false);
    RandomImage.Instance.count = 0;
}
```

コードの説明

このスクリプトはゲームの全体的な状態を管理し、プレイヤーの通貨、タイムアウト、天気の変化、倍速調整、ランキングの表示など、さまざまなゲームメカニズムを実装しています。各機能はUIと対話し、プレイヤーにゲームの進行状況を示すように設計しました。



AchievementManager.cs

```
void Dojun()
{
    if (tradeCount == 100 && achievement0 == 0)
    {
        achievement0 = 1;
        ActiveDojun(0);
        ReturnImage(0);
        StartCoroutine(Setting());
        AchievementSystem.Instance.achievementList.AddEntry(achievement0, achievement1, achievement2, achievement3, achievement4, achievement6);
        AchievementSystem.Instance.achievementList.SaveAchievements();
    }
    if (lisenceCount == 6 && achievement1 == 0)
    {
        achievement1 = 1;
        ActiveDojun(1);
        ReturnImage(1);
        StartCoroutine(Setting());
        AchievementSystem.Instance.achievementList.AddEntry(achievement0, achievement1, achievement2, achievement3, achievement4, achievement6);
        AchievementSystem.Instance.achievementList.SaveAchievements();
    }
    if (achievement2 == 1 && achievement2Active == false)
    {
        achievement2Active = true;
        AchievementSystem.Instance.achievementList.AddEntry(achievement0, achievement1, achievement2, achievement3, achievement4, achievement6);
        AchievementSystem.Instance.achievementList.SaveAchievements();
    }
    if (GameManager.Instance.curMonth < 9 && Unlock.instance.AirBT.gameObject.activeSelf && achievement3 == 0)
    {
        achievement3 = 1;
        ActiveDojun(3);
        ReturnImage(3);
        StartCoroutine(Setting());
        AchievementSystem.Instance.achievementList.AddEntry(achievement0, achievement1, achievement2, achievement3, achievement4, achievement6);
        AchievementSystem.Instance.achievementList.SaveAchievements();
    }
    if (clickCount == 12 && achievement4 == 0)
    {
        achievement4 = 1;
        ActiveDojun(4);
        ReturnImage(4);
        StartCoroutine(Setting());
        AchievementSystem.Instance.achievementList.AddEntry(achievement0, achievement1, achievement2, achievement3, achievement4, achievement6);
        AchievementSystem.Instance.achievementList.SaveAchievements();
    }
    if (GameManager.Instance.DisCount == 10 && achievement5 == 0)
    {
        achievement5 = 1;
        ActiveDojun(5);
        ReturnImage(5);
        StartCoroutine(Setting());
```

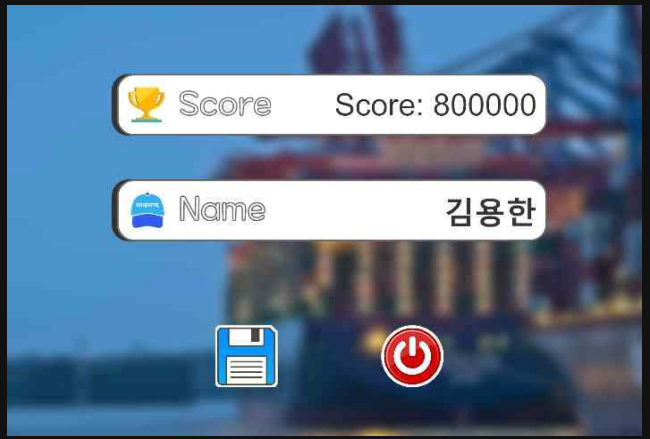
```
public void ReturnImage(int index)
{
    achievementImage[index].GetComponent<Image>().color = new Color(1f, 1f, 1f, 1f);
    achievementtext[index].gameObject.SetActive(true);
}

public void ActiveDojun(int index)
{
    popupWindow.SetActive(true);
    anim.SetBool("6Second", false);
    popupAchievement[index].SetActive(true);
}

public IEnumerator Setting()
{
    yield return new WaitForSecondsRealtime(6f);
    anim.SetBool("6Second", true);
    yield return new WaitForSecondsRealtime(1f);
    popupWindow.SetActive(false);
    for (int i = 0; i < popupAchievement.Length; i++)
    {
        popupAchievement[i].SetActive(false);
    }
}
```

コードの説明

ゲームのパフォーマンスシステムを管理し、プレイヤーが特定の目標を達成したときにそれを記録し、UIに表示する役割を果たします。成果達成のためのカウントを管理し、UIに成果を視覚的に表示しました。

Ranking.cs

```
public void SaveScore()
{
    string playerName = playerNameInput.text;
    int playerScore = GameManager.Instance.money;

    List<PlayerScore> rankingList = LoadRankingList();

    rankingList.Add(new PlayerScore(playerName, playerScore));

    rankingList = rankingList.OrderByDescending(ps => ps.Score).Take(10).ToList();

    SaveRankingList(rankingList);
}

private List<PlayerScore> LoadRankingList()
{
    string jsonString = PlayerPrefs.GetString("rankingList", "[]");
    return JsonUtility.FromJson<PlayerScoreList>(jsonString).PlayerScores;
}

private void SaveRankingList(List<PlayerScore> rankingList)
{
    PlayerScoreList playerScoreList = new PlayerScoreList { PlayerScores = rankingList };
    string jsonString = JsonUtility.ToJson(playerScoreList);
    PlayerPrefs.SetString("rankingList", jsonString);
}

void ShowRanking()
{
    List<PlayerScore> rankingList = LoadRankingList().Take(3).ToList();

    for (int i = 0; i < rankingTexts.Length; i++)
    {
        rankingTexts[i].text = "";
    }

    for (int i = 0; i < rankingList.Count; i++)
    {
        var score = rankingList[i];
        rankingTexts[i].text = $"{i + 1}. {score.PlayerName} - {score.Score}";
    }
}
```



コードの説明

プレイヤーの名前とスコアを記録し、ランキングを更新してUIに表示する機能を実行し、PlayerPrefsを通じてスコアを永久保存し、上位3人のスコアを画面に表示してプレイヤーの達成感を高め、ランキングシステムはユーザーが入力した名前とスコアに基づいて動的に更新されるように設計しました。

tradeGoods.cs

```
public class tradeGoods : MonoBehaviour
{
    public static tradeGoods Instance;
    public SoundManager soundManager;
    public goodsDB goodsDB;

    //계산용 변수
    private int totalCost;
    private int purCost;
    private int saleCost;

    public UnityEvent onCargoWindowEnabled;

    private string selectedProvince;

    [Header("배송 지역 특산물")]
    [SerializeField] private GameObject preview;

    [Header("각 창에 생성된 프리팹 리스트")]
    public List<GameObject> instantiatedPrefabs = new List<GameObject>();
    public List<GameObject> newgoods = new List<GameObject>();
    public List<GameObject> inventories = new List<GameObject>();

    [Header("처음 재고 저장 할 배열")]
    public SavedStocks[] Sstock = new SavedStocks[90];

    [Header("지역화물")]
    [SerializeField] private GameObject TradeChang; // 지역화물 프리팹이 생성될 창
    [SerializeField] private GameObject LocalTradePrefab; // 지역화물 프리팹

    [Header("거래화물")]
    [SerializeField] private GameObject PurchaseChang; // 거래화물 프리팹이 생성될 창
    [SerializeField] private GameObject purchasePrefab; // 거래화물 프리팹
    public GameObject salePrefab; // 거래창에 생성되는 판매 프리팹
    public GameObject costSum; // 구매, 판매가격 합산

    [Header("소유화물")]
    public GameObject inventransfrom; // 소유화물 프리팹이 생성될 창
    public GameObject invenPrefab; // 소유화물 프리팹

    [Header("적재량 슬라이더")]
    public Slider capacitySlider;
    [HideInInspector] public float haveCapacity;
    [HideInInspector] public bool disCount = false;
```

```
//희귀도별로 가격 업데이트
private Dictionary<string, bool> isPriceAdjusted = new Dictionary<string, bool>();

public void UpdateInventoryPrices()
{
    float CalculateAdjustedPrice(float basePrice, string rarity, float originalPrice, string goodsName)
    {
        float multiplier = GetMultiplierByRarity(rarity);

        if (!isPriceAdjusted.ContainsKey(goodsName))
            isPriceAdjusted[goodsName] = false;

        if (!isPriceAdjusted[goodsName] && basePrice <= 2 * originalPrice)
        {
            isPriceAdjusted[goodsName] = true;
            return basePrice * multiplier;
        }
        else
            return basePrice;
    }
    for (int i = 0; i < inventories.Count; i++)
    {
        string goodsName = inventories[i].transform.GetChild(0).GetComponent<Text>().text;
        string rarity = GetRarityForGoods(goodsName);
        float originalPrice = float.Parse(inventories[i].transform.GetChild(3).GetComponent<Text>().text);
        float adjustedPrice = CalculateAdjustedPrice(originalPrice, rarity, originalPrice, goodsName);

        Text costText = inventories[i].transform.GetChild(3).GetComponent<Text>();
        costText.text = adjustedPrice.ToString();

        GameObject prefabWithGoodsName = instantiatedPrefabs.Find(prefab => prefab.transform.GetChild(0).GetComponent<Text>().text == goodsName);

        if (prefabWithGoodsName != null)
        {
            Text localPriceText = prefabWithGoodsName.transform.GetChild(3).GetComponent<Text>();
            localPriceText.text = adjustedPrice.ToString();
        }
    }
}
```

コードの説明

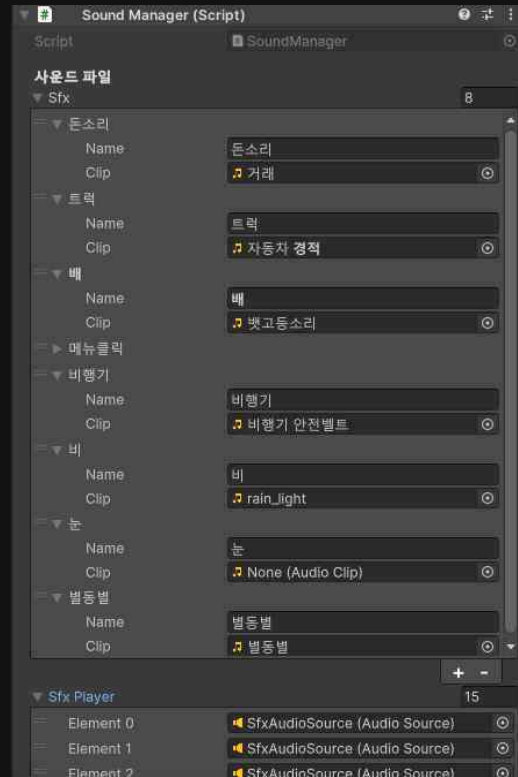
ローカル貨物および取引貨物システムを管理するために必要なすべての機能が含まれています。　ユーザーが選択した特産品の在庫を管理し、それを購入または販売するためのUIを提供し、価格調整と在庫更新を行い、またプレイヤーの現在の容量と重量を管理することができます。

SoundManager.cs

```
public class SoundManager : MonoBehaviour
{
    public static SoundManager Instance;

    [Header("사운드 파일")]
    [SerializeField] public Sound[] sfx = null;
    [SerializeField] AudioSource[] sfxPlayer = null;

    [Header("배경음악")]
    [SerializeField] public Sound bgm = null;
    [SerializeField] AudioSource bgmPlayer = null;

    public int click = 0;

    private void Awake()
    {
        if(Instance == null)
            Instance = this;
        else
            Destroy(gameObject);

        SoundManager.Instance.PlayBGM("배경음악");
    }

    public void PlayBGM(string bgmName)
    {
        if (bgmName == bgm.name)
        {
            bgmPlayer.clip = bgm.clip;
            bgmPlayer.Play();
        }
    }

    public void SoundCtrl()
    {
        SoundManager.Instance.PlaySFX("메뉴클릭");
        click++;

        if (click == 1)
        {
            for(int i = 0; i < sfxPlayer.Length; i++)
            {
                sfxPlayer[i].volume = 0.0625f;
                bgmPlayer.volume = 0.0625f;
            }
        }
        if (click == 2)
        {
            for (int i = 0; i < sfxPlayer.Length; i++)
            {
                sfxPlayer[i].volume = 0f;
                bgmPlayer.volume = 0f;
            }
        }
        if (click > 2)
        {
            for (int i = 0; i < sfxPlayer.Length; i++)
            {
                sfxPlayer[i].volume = 0.125f;
                bgmPlayer.volume = 0.125f;
            }
            click = 0;
        }
    }

    public void PlaySFX(string p_sfxName)
    {
        // Debug.Log("효과음 재생");
        for (int i = 0; i < sfx.Length; i++)
        {
            if (sfx[i].name == p_sfxName/*|| ransfx[i].name == p_sfxName*/)
            {
                for (int x = 0; x < sfxPlayer.Length; x++)
                {
                    if (!sfxPlayer[x].isPlaying)
                    {
                        sfxPlayer[x].clip = sfx[i].clip;
                        sfxPlayer[x].Play();
                        return;
                    }
                }
                Debug.Log("모든 오디오 플레이어가 재생중입니다.");
                return;
            }
        }
        Debug.Log(p_sfxName + "이름의 효과음이 없습니다.");
    }
```

コードの説明

さまざまなサウンドを追加してゲームの没入感を高め、ユーザーが望む方法でサウンドを調節する機能を提供し、全体的なゲーム体験を向上させるように設計しました。



プロジェクトを進行しながら感じたこと

ゲーム開発を進める中で、ビジュアルスクリプティングとレベルデザインの重要性を実感しました。これら二つの要素は、ゲーム全体の品質やプレイヤーの体験に大きな影響を与えるものであると考えるようになり、それを習得するために多くの時間と努力を費やしました。その過程で、より深い知識を身につけることができました。

ビジュアルスクリプティングを活用することで直感的なゲームロジックを構築し、レベルデザインを通じてプレイヤーの興味を引く環境を作ることの重要性を学びました。この経験を基に、より優れたゲーム開発者となるために、今後も継続的に学習と成長を続けていきたいと考えております。

CSの文法的な面で自身の不足している部分を補う貴重な時間となり、知識や経験を共有することで、チームワークの重要性を改めて認識するとともに、個人としての成長も実感することができました。

고군분투 2.5D

PC | ランニングアクションゲーム | Unity2.5D

- ゲーム紹介

プレイヤーが様々な障害物や敵を超えて目標を達成する2.5Dドットグラフィックを活用したプラットフォーマーゲームです。

- 開発内容

1人開発

2022-06~2022-09（10週間）

- プロジェクト本人の役割

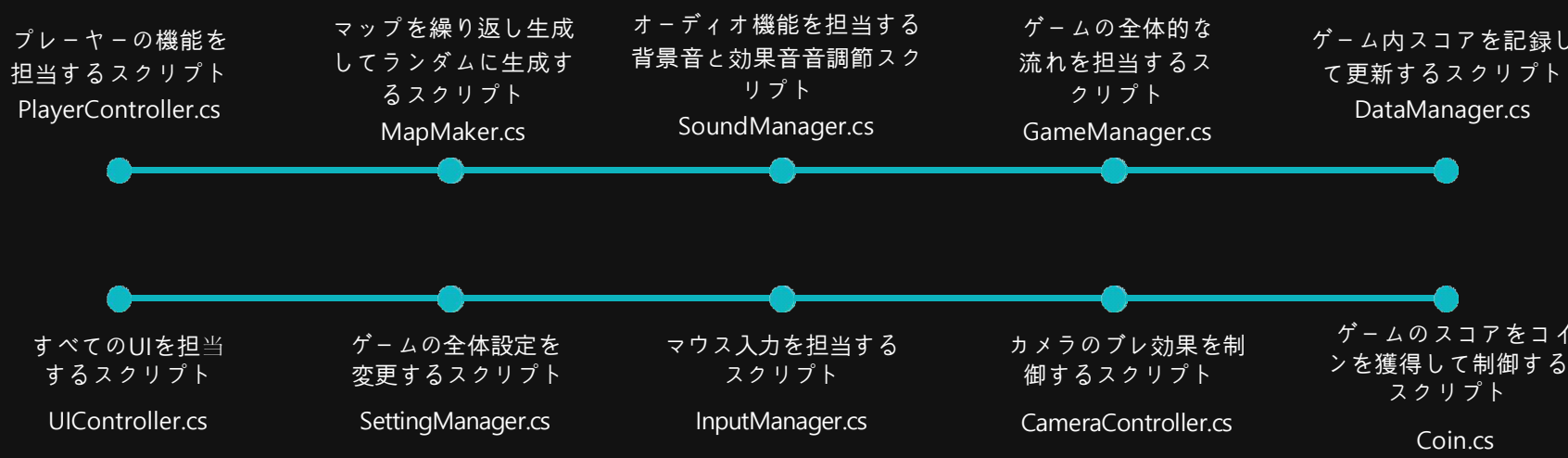
プログラマー / 企画

プレーヤー機能とコントロール / マップ＆障害物管理 / ユーザーインターフェース

プロジェクト企画 / スケジュール管理/サウンド管理 / 障害物の実装 / 演出

主な実装

Script for Implement



PlayerController.cs

```
private void Update()
{
    if (DataManager.Instance.isDead)
    {
        if(isDead)
            return;
        rigid.velocity = Vector3.up * 40f;
        GetComponent<CapsuleCollider>().enabled = false;
        transform.position = new Vector3(transform.position.x, transform.position.y, -1);
        playerSprite.SetAnimation(PlayerAnimation.Dead);
        isDead = true;
        return;
    }

    if (InputManager.Instance.GetJumpKey())
    {
        if (OnGrounded())
        {
            currentHand.localPosition = jumpHandPos;
            StartCoroutine("Jumping");
            playerActive.Jump();
            playerSprite.SetAnimation(PlayerAnimation.Jump);
        }
        else
        {
            playerActive.JumpShot();
            playerSprite.SetAnimation(PlayerAnimation.JumpShot);
        }
    }
    if (wire.OnWire && !isWire)
    {
        currentHand.localPosition = wireHandPos;
        isWire = true;
        playerActive.OnWire();
        playerSprite.SetAnimation(PlayerAnimation.Wire);
        camera.ZoomInCamera();
    }
    else if (wire.OnWire)
    {
        transform.up = (wire.transform.position - transform.position).normalized;
    }
    else if (!wire.OnWire && isWire)
    {
        transform.rotation = Quaternion.identity;
        isWire = false;
        playerSprite.SetAnimation(PlayerAnimation.Rolling);
    }
    if(OnGrounded() && !isJump)
    {
        playerSprite.SetAnimation(PlayerAnimation.Run);
        camera.ZoomOutCamera();
        if (DeadCast() && !wire.OnWire)
        {
            gameManager.GameOver();
        }
    }
}
```

```
private IEnumerator Jumping()
{
    isJump = true;
    yield return new WaitForSeconds(0.1f);
    isJump = false;
}

private bool OnGrounded()
{
    return Physics.BoxCast(transform.position, new Vector3(1.5f, 0.2f, 1f) * 0.5f, Vector3.down, Quaternion.identity, 1.5f, jumpableGround);
}

private bool DeadCast()
{
    return !Physics.Raycast(transform.position + new Vector3(-0.25f,0,0), Vector3.down, 4f, jumpableGround);
}
```

コードの説明

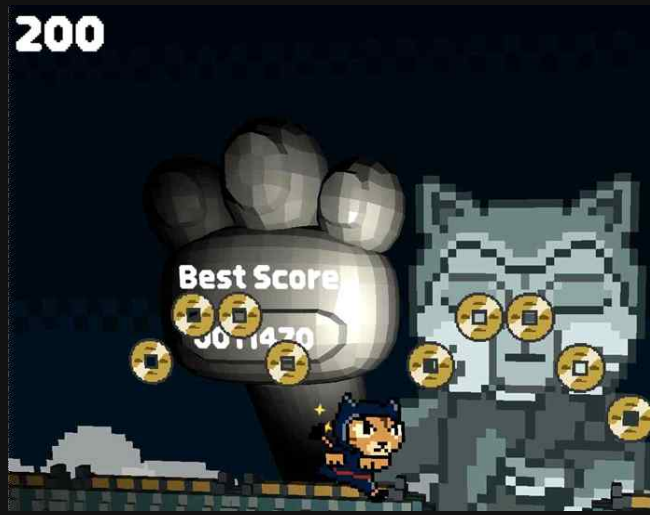
レイヤーの多様な行動や状態を制御し、ゲームの流れをスムーズに保つことに貢献し、ゲーム内の物理的な相互作用やアニメーション管理、ユーザー入力処理を行いました。

Coin.cs

```
public class Coin : MonoBehaviour
{
    [SerializeField] private AudioClip clip;
    [SerializeField] private GameObject coinEffect;

    private void Start()
    {
        transform.localPosition += Vector3.forward * -0.1f;
    }

    private void MakeEffect()
    {
        GameObject effect = Instantiate(coinEffect, transform.position, Quaternion.identity);
        effect.transform.parent = transform.parent.parent.parent;
        effect.GetComponent<ParticleSystem>().Play();
    }

    private void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.CompareTag("Player") == true)
        {
            if(transform.tag == "Coin")
            {
                DataManager.Instance.Score += 5;
            }
            else if(transform.tag == "Gold_Coin")
            {
                DataManager.Instance.Score += 50 - (5 * DataManager.Instance.Stage);
                transform.parent.parent.parent.GetComponent<MapSetting>().currentGoldCoins++;
            }

            SoundManager.Instance.SFXPlay(transform.tag, clip);
            MakeEffect();
            gameObject.SetActive(false);
        }
    }
}
```

コードの説明

プレイヤーがコインに触れるとスコアを更新し、視覚的効果やサウンドを再生してコイン収集の満足感を高め、また、コインの種類によってスコアを異なるように設定してゲームの進行状況に合った報酬を提供するように設計しました。



SettingManager.cs

```
public class SettingManager : MonoBehaviour
{
    [SerializeField] private GameObject settingPanel;
    [SerializeField] private Slider BGMSlider;
    [SerializeField] private Slider SFXSlider;
    [SerializeField] private AudioClip settingPanelSound;

    public void OnSetting()
    {
        BGMSlider.value = PlayerPrefs.GetFloat("BGMScale");
        SFXSlider.value = PlayerPrefs.GetFloat("SFXScale");
        settingPanel.SetActive(true);
        StartCoroutine("ShowSetting");
    }

    public void OffSetting()
    {
        StartCoroutine("CloseSetting");
    }

    public void SetBGM()
    {
        float back = BGMSlider.value;
        back = Mathf.Clamp(back, 0f, 1f);
        PlayerPrefs.SetFloat("BGMScale", back);
    }

    public void SetSFX()
    {
        float play = SFXSlider.value;
        play = Mathf.Clamp(play, 0f, 1f);
        PlayerPrefs.SetFloat("SFXScale", play);
    }

    private void SetComponent(Transform trans, bool b)
    {
        for(int i=0;i< trans.childCount; i++)
        {
            trans.GetChild(i).gameObject.SetActive(b);
        }
    }

    private IEnumerator ShowSetting()
    {
        RectTransform panelTransform = settingPanel.transform.GetChild(0).GetComponent<RectTransform>();
        SettingPanel panel = settingPanel.GetComponent<SettingPanel>();

        float yPos = -1000;
        float panelSpeed = 10f;
        SetComponent(panelTransform.transform, false);
        panelTransform.position = new Vector3(0, yPos, 0);

        while (true)
        {
            yield return null;
            if(yPos < 0)
            {
            yPos = Mathf.Lerp(yPos, 0 + 10, Time.deltaTime * panelSpeed);
            panelTransform.localPosition = new Vector3(0, yPos, 0);
            }
            else
            {
                SoundManager.Instance.SFXPlay("Paper", settingPanelSound);
                panel.OpenAnim();
                yield return new WaitForSeconds(0.3f);
                SetComponent(panelTransform.transform, true);

                yield break;
            }
        }
    }

    private IEnumerator CloseSetting()
    {
        RectTransform panelTransform = settingPanel.transform.GetChild(0).GetComponent<RectTransform>();
        SettingPanel panel = settingPanel.GetComponent<SettingPanel>();

        float yPos = 0;
        float panelSpeed = 10f;

        SoundManager.Instance.SFXPlay("Paper", settingPanelSound);
        panelTransform.localPosition = new Vector3(0, yPos, 0);
        SetComponent(panelTransform.transform, false);
        panel.CloseAnim();
        yield return new WaitForSeconds(0.3f);
        while (true)
        {
            yield return null;
            if (yPos > -1000)
            {
                yPos = Mathf.Lerp(yPos, -1000 - 10, Time.deltaTime * panelSpeed);
                panelTransform.localPosition = new Vector3(0, yPos, 0);
            }
            else
            {
                settingPanel.SetActive(false);
                yield break;
            }
        }
    }
```

コードの説明

ゲーム設定パネルの表示および非表示機能を管理し、ユーザーが背景音楽や効果音の音量を調節できるようにし、また、設定値を保存して読み込む機能を通じてユーザーの設定を維持できるようにし、パネルの開閉の過程でアニメーション効果を適用してより滑らかな効果を実現しました。

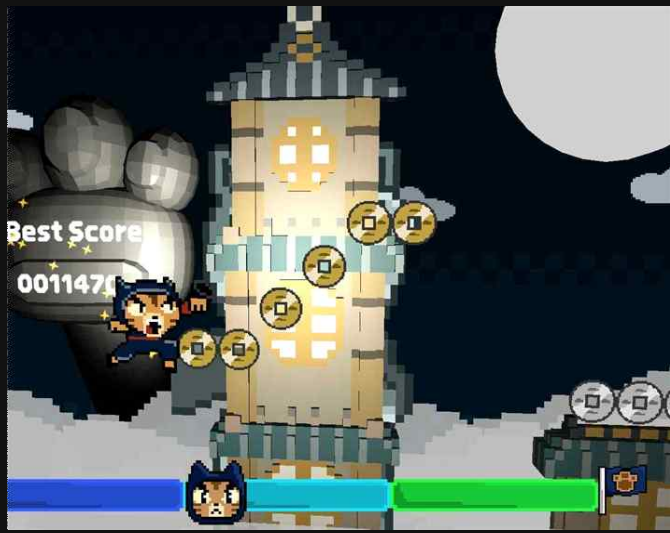
MapMaker.cs

```
public class MapMaker : MonoBehaviour
{
    [Serializable] public class Map
    {
        public List<GameObject> map = new List<GameObject>();
    }

    public List<Map> stageMap = new List<Map>();
    [SerializeField] private GameObject noneMap;

    private MapMove mapMove;

    private void Awake()
    {
        mapMove = GetComponent<MapMove>();
    }

    private void Update()
    {
        if (mapMove.currentDistance > mapMove.maxDistance)
        {
            if(DataManager.Instance.isEnd)
            {
                GameObject go1 = Instantiate(noneMap);

                mapMove.distance += mapMove.maxDistance;
                mapMove.currentDistance -= mapMove.maxDistance;
                mapMove.maxDistance = go1.GetComponent<MapSetting>().Distance;
                go1.transform.position = transform.localPosition + Vector3.right * mapMove.distance;
                go1.transform.parent = transform;
                return;
            }
            int random = UnityEngine.Random.Range(0, stageMap[DataManager.Instance.Stage].map.Count);
            GameObject go = Instantiate(stageMap[DataManager.Instance.Stage].map[random]);

            mapMove.distance += mapMove.maxDistance;
            mapMove.currentDistance -= mapMove.maxDistance;
            mapMove.maxDistance = go.GetComponent<MapSetting>().Distance;
            go.transform.position = transform.localPosition + Vector3.right * mapMove.distance;
            go.transform.parent = transform;
        }
    }
}
```

```
[SerializeField] private GameManager gameManager;
[SerializeField] private float currentMapSpeed;
[SerializeField] private float moveSpeed = 14;
[SerializeField] private float changeSpeed = 6f;
[SerializeField] private List<float> stageTime;

public int MaxStage => stageTime.Count;

public float distance = 0;
public float currentDistance = 0;
public float time = 0;
public float maxTime;

public float maxDistance = 0;

private void Start()
{
    maxTime = stageTime[DataManager.Instance.Stage];
}

private void Update()
{
    if (DataManager.Instance.isDead)
        return;

    currentMapSpeed = moveSpeed + (DataManager.Instance.Stage * changeSpeed);
    currentDistance += Time.deltaTime * currentMapSpeed;
    transform.position += Vector3.left * Time.deltaTime * currentMapSpeed;

    time += Time.deltaTime;

    if (time > maxTime && DataManager.Instance.Stage + 1 < MaxStage)
    {
        UpdateStage();
        time -= maxTime;
        maxTime = stageTime[DataManager.Instance.Stage];
    }
    else if(time > maxTime && DataManager.Instance.Stage + 1 >= MaxStage)
    {
        gameManager.Ending();
    }
}

private void UpdateStage()
{
    gameManager.SpeedUp();
}
```

コードの説明

ゲームステージに応じて様々なマップオブジェクトを動的に生成し、継続的にプレイヤーに新しい環境を提供するように設定しました。



CameraController.cs

```
private void UpdateCameraZoom()
{
    nomalSize = saveNomalSize + (DataManager.Instance.Stage * changeSize);
    zoomSize = saveZoomSize + (DataManager.Instance.Stage * changeSize);
}

private IEnumerator ZoomIn()
{
    while (true)
    {
        transZ = Mathf.Lerp(transZ, zoomZ, Time.deltaTime * zoomSpeed);
        fieldOfView = Mathf.Lerp(fieldOfView, zoomSize, Time.deltaTime * zoomSpeed);

        transform.position = Vector3.Lerp(transform.position, new Vector3(player.position.x, player.position.y, transZ), Time.deltaTime * zoomSpeed);
        camera.fieldOfView = fieldOfView;
        yield return null;
    }
}


private IEnumerator ZoomOut()
{
    transZ = nomalZ;
    while (true)
    {
        fieldOfView = Mathf.Lerp(fieldOfView, nomalSize, Time.deltaTime * zoomSpeed);

        transform.position = Vector3.Lerp(transform.position, offset, Time.deltaTime * zoomSpeed);
        camera.fieldOfView = fieldOfView;
        yield return null;
    }
}
```

```
public class CameraController : MonoBehaviour
{
    [SerializeField] private Transform player;
    [SerializeField] private float nomalSize = 60f;
    [SerializeField] private float nomalZ = -11f;
    [SerializeField] private float zoomSize = 70f;
    [SerializeField] private float zoomZ = -9;
    [SerializeField] private float zoomSpeed = 5f;
    [SerializeField] private float changeSize = 2f;
    [SerializeField] private Vector3 offset;

    private float saveNomalSize;
    private float saveZoomSize;

    private Camera camera;
    private float fieldOfView;
    private float transZ;
    private bool isZoomOut = false;

    private void Awake()
    {
        camera = GetComponent<Camera>();
    }

    private void Update()
    {
        UpdateCameraZoom();
    }

    private void Start()
    {
        fieldOfView = nomalSize;
        transZ = nomalZ;
        saveNomalSize = nomalSize;
        saveZoomSize = zoomSize;
    }

    public void ZoomInCamera()
    {
        StopCoroutine("ZoomOut");
        isZoomOut = false;
        transform.position = offset;
        StartCoroutine("ZoomIn");
    }
    public void ZoomOutCamera()
    {
        StopCoroutine("ZoomIn");
        if (!isZoomOut)
        {
            isZoomOut = true;
            StartCoroutine("ZoomOut");
        }
    }
```

コードの説明

プレイヤーの行動に応じてカメラのズーム機能を動的に調整してゲームの没入感を高め、各ステージの進行状況に応じてカメラの視野が変化し、プレイヤーにより良い視覚体験を提供するようにしました。

プロジェクトを進行しながら感じたこと

コードを作成する中で直面したさまざまな課題やエラーは、開発プロセスにおける成長へとつながり、問題を解決するたびに達成感を感じました。このような経験は、今後のプロジェクトにも大いに役立つと確信しております。

サウンド管理、プレイヤーコントロール、カメラ制御、マップ生成など、さまざまなスクリプトを通じて、それぞれの要素がどのように相互作用するのかを深く理解することができました。

スクリプトの細部を調整しながら、デバッグの重要性や最適化の必要性を痛感し、さらにビジュアルスクリプティングやレベルデザインの重要性も改めて実感いたしました。

Dechya room

PC | 3D放出ゲーム | Unreal3D

- ゲーム紹介

アンリアルエンジンの高品質グラフィックスで、細かく没入感のある環境を具現化し、防脱出のユニークな雰囲気を体験するゲームです。

- 開発内容

1人開発

2024-06〜2024-08（7週間）

- プロジェクト本人の役割

プレーヤー機能 / カメラ演出 / ユーザーインターフェース / プロジェクト企画 / カレンダー / サウンド管理 / 演出

主な実装

Script for Implement

プレイヤーの移動、状態を管理するスクリプト
MyCharacter.cpp

すべてのUIを担当するスクリプト
UIManager.cpp

プレーヤーカメラの機能を実行するスクリプト
PlayerCameraController.cpp

オーディオ機能を担当する背景音と効果音音調節スクリプト
AudioManager.cpp

ゲームの全体的な流れを担当するスクリプト
GameManager.cpp

MyCharacter.cpp

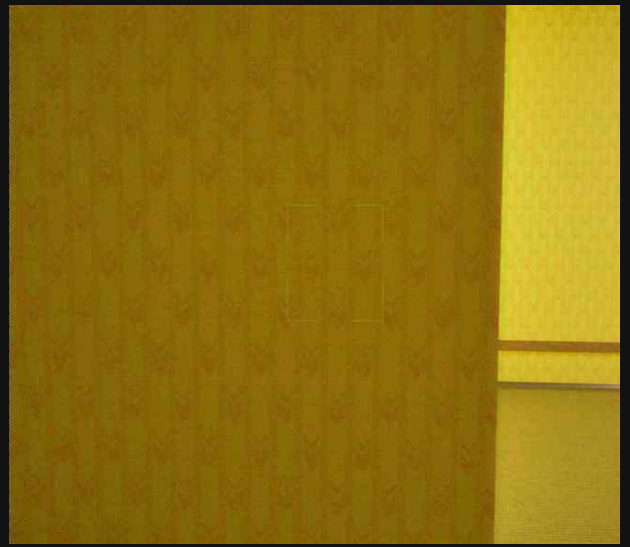
```h
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Character.h"
#include "MyCharacter.generated.h"

UCLASS()
class YOURPROJECT_API AMyCharacter : public ACharacter
{
    GENERATED_BODY()

public:
    AMyCharacter();

protected:
    virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;

    void MoveForward(float Value);
    void MoveRight(float Value);
    void TurnAtRate(float Rate);
    void LookUpAtRate(float Rate);

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Camera)
    float BaseTurnRate;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Camera)
    float BaseLookUpRate;
};
```

```cpp
#include "MyCharacter.h"

AMyCharacter::AMyCharacter()
{
    BaseTurnRate = 45.0f;
    BaseLookUpRate = 45.0f;
}

void AMyCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    PlayerInputComponent->BindAxis("MoveForward", this, &AMyCharacter::MoveForward);
    PlayerInputComponent->BindAxis("MoveRight", this, &AMyCharacter::MoveRight);
    PlayerInputComponent->BindAxis("Turn", this, &AMyCharacter::TurnAtRate);
    PlayerInputComponent->BindAxis("LookUp", this, &AMyCharacter::LookUpAtRate);
}
```

```cpp
void AMyCharacter::MoveForward(float Value)
{
    if (Value != 0.0f)
    {
        AddMovementInput(GetActorForwardVector(), Value);
    }
}

void AMyCharacter::MoveRight(float Value)
{
    if (Value != 0.0f)
    {
        AddMovementInput(GetActorRightVector(), Value);
    }
}

void AMyCharacter::TurnAtRate(float Rate)
{
    AddControllerYawInput(Rate * BaseTurnRate * GetWorld()->GetDeltaSeconds());
}

void AMyCharacter::LookUpAtRate(float Rate)
{
    AddControllerPitchInput(Rate * BaseLookUpRate * GetWorld()->GetDeltaSeconds());
}
```

コードの説明

プレイヤーキャラクターがゲーム内で自由に移動して回転できるようにするための基本的な構造を設計しました。



UIManager.cpp

```cpp
#include "UIManager.h"
#include "Blueprint/UserWidget.h"
#include "CameraStatusWidget.h"

AUIManager::AUIManager()
{
    PrimaryActorTick.bCanEverTick = false;
}

void AUIManager::BeginPlay()
{
    Super::BeginPlay();

    if (CameraStatusWidgetClass)
    {
        CameraStatusWidget = CreateWidget<UCameraStatusWidget>(GetWorld(), CameraStatusWidgetClass);
        if (CameraStatusWidget)
        {
            CameraStatusWidget->AddToViewport();
        }
    }
}

void AUIManager::UpdateCameraStatus(const FString& StatusMessage, float FPS)
{
    if (CameraStatusWidget)
    {
        CameraStatusWidget->UpdateStatus(StatusMessage, FPS);
    }
}
```

```cpp
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "UIManager.generated.h"

UCLASS()
class YOURPROJECT_API AUIManager : public AActor
{
    GENERATED_BODY()

public:
    AUIManager();

protected:
    virtual void BeginPlay() override;

public:
    void UpdateCameraStatus(const FString& StatusMessage, float FPS);

private:
    UPROPERTY(EditAnywhere)
    TSubclassOf<class UUserWidget> CameraStatusWidgetClass;

    UCameraStatusWidget* CameraStatusWidget;
};
```

コードの説明

FPSを計算してUIを更新し、RECモードとFPS情報を表示するように設計しました。

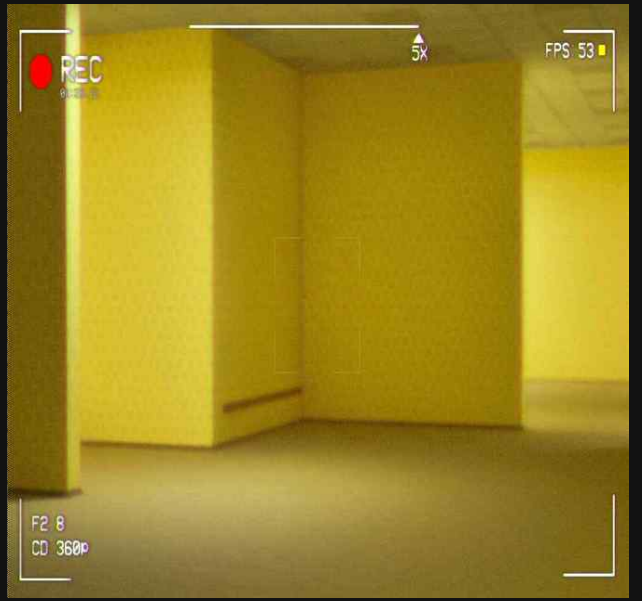
PlayerCameraController.cpp

```h
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/SpringArmComponent.h"
#include "Camera/CameraComponent.h"
#include "GameFramework/Character.h"
#include "PlayerCameraController.generated.h"

UCLASS()
class YOURPROJECT_API APlayerCameraController : public ACharacter
{
    GENERATED_BODY()

public:
    APlayerCameraController();

protected:
    virtual void BeginPlay() override;
    virtual void Tick(float DeltaTime) override;

public:
    virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;

private:
    void Turn(float Value);
    void LookUp(float Value);

    UPROPERTY(VisibleAnywhere)
    USpringArmComponent* SpringArm;

    UPROPERTY(VisibleAnywhere)
    UCameraComponent* Camera;

    float MouseSensitivity;
};
```

```cpp
#include "PlayerCameraController.h"
#include "GameFramework/SpringArmComponent.h"
#include "Camera/CameraComponent.h"

APlayerCameraController::APlayerCameraController()
{
    PrimaryActorTick.bCanEverTick = true;

    SpringArm = CreateDefaultSubobject<USpringArmComponent>(TEXT("SpringArm"));
    SpringArm->SetupAttachment(RootComponent);
    SpringArm->TargetArmLength = 300.0f;
    SpringArm->bUsePawnControlRotation = true;

    Camera = CreateDefaultSubobject<UCameraComponent>(TEXT("Camera"));
    Camera->SetupAttachment(SpringArm);
    Camera->bUsePawnControlRotation = false;

    MouseSensitivity = 1.0f;
}

void APlayerCameraController::BeginPlay()
{
    Super::BeginPlay();
}

void APlayerCameraController::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);
}

void APlayerCameraController::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    PlayerInputComponent->BindAxis("Turn", this, &APlayerCameraController::Turn);
    PlayerInputComponent->BindAxis("LookUp", this, &APlayerCameraController::LookUp);
}

void APlayerCameraController::Turn(float Value)
{
    AddControllerYawInput(Value * MouseSensitivity);
}

void APlayerCameraController::LookUp(float Value)
{
    AddControllerPitchInput(Value * MouseSensitivity);
}
```

コードの説明

プレイヤーカメラの回転をマウス入力に応じて調節する構造を設計し、スプリングアームを使用してカメラの位置を維持し、カメラコンポーネントがプレイヤーの視点に従うように設定しました。



AudioManager.cpp

```cpp
#include "AudioManager.h"
#include "Components/AudioComponent.h"
#include "Sound/SoundMix.h"

AAudioManager::AAudioManager()
{
    PrimaryActorTick.bCanEverTick = false;

    BackgroundAudioComponent = CreateDefaultSubobject<UAudioComponent>(TEXT("BackgroundAudioComponent"));
    EffectAudioComponent = CreateDefaultSubobject<UAudioComponent>(TEXT("EffectAudioComponent"));
}

void AAudioManager::BeginPlay()
{
    Super::BeginPlay();

    BackgroundAudioComponent->Play();
}

void AAudioManager::SetBackgroundVolume(float Volume)
{
    if (BackgroundAudioComponent)
    {
        BackgroundAudioComponent->SetVolumeMultiplier(Volume);
    }
}

void AAudioManager::SetEffectVolume(float Volume)
{
    if (EffectAudioComponent)
    {
        EffectAudioComponent->SetVolumeMultiplier(Volume);
    }
}
```

```cpp
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/Actor.h"
#include "AudioManager.generated.h"

UCLASS()
class YOURPROJECT_API AAudioManager : public AActor
{
    GENERATED_BODY()

public:
    AAudioManager();

protected:
    virtual void BeginPlay() override;

public:
    UFUNCTION(BlueprintCallable, Category = "Audio")
    void SetBackgroundVolume(float Volume);

    UFUNCTION(BlueprintCallable, Category = "Audio")
    void SetEffectVolume(float Volume);

    UPROPERTY(EditAnywhere, Category = "Audio")
    class USoundMix* SoundMix;

private:
    UPROPERTY()
    class UAudioComponent* BackgroundAudioComponent;

    UPROPERTY()
    class UAudioComponent* EffectAudioComponent;
};
```

コードの説明

ゲームで背景音と効果音の再生と音量調整を管理する機能を担当し、独自の設計を行い、それぞれのオーディオコンポーネントを作成し、オーディオ管理を簡便にしてゲームの没入感を高めることに貢献するように設計しました。

AbilityManager.cs

```h
#pragma once

#include "CoreMinimal.h"
#include "GameFramework/GameModeBase.h"
#include "GameManager.generated.h"

UCLASS()
class YOURPROJECT_API AGameManager : public AGameModeBase
{
    GENERATED_BODY()

public:
    AGameManager();

    virtual void BeginPlay() override;

    UFUNCTION(BlueprintCallable, Category = "Game Flow")
    void StartGame();

    UFUNCTION(BlueprintCallable, Category = "Game Flow")
    bool IsGameRunning() const;

private:
    bool bIsGameRunning;
};
```

```cpp
#include "GameManager.h"

AGameManager::AGameManager()
{
    PrimaryActorTick.bCanEverTick = false;
    bIsGameRunning = false;
}

void AGameManager::BeginPlay()
{
    Super::BeginPlay();
}

void AGameManager::StartGame()
{
    bIsGameRunning = true;
    UE_LOG(LogTemp, Warning, TEXT("Game Started!"));
}

bool AGameManager::IsGameRunning() const
{
    return bIsGameRunning;
}
```

コードの説明

GameManagerを介してUnreal Editorでゲームが開始されるか終了したかを判定し、メモリの浪費を防ぐように設計しました。

プロジェクトを進行しながら感じたこと

Unreal EngineでC++を使用して開発するのは初めてだったため、最初は多くの困難がありました。特に、メモリ管理やオブジェクト指向プログラミングの概念を理解するのに時間がかかりましたが、次第に慣れていくことで、より深いレベルでのプログラミングが可能となりました。

Unreal Engineを用いた開発の過程で感じた自分の未熟さは、今後さらに成長したいという強い動機となり、より良い成果物を目指して学び続け、成長し続ける決意を固めることができました。

プロジェクトを進める中で、さまざまな技術的な問題に直面しました。これらの問題を解決するために、多くの資料を参考にし、コミュニティや先輩方と交流する中で、問題解決能力を向上させることができました。