

ABOUT ME



이름 : 신재성

직무 : Unity 개발자

연락처 : 01055878970

이메일: gotodongyang@naver.com

git : <https://github.com/sinjaesung>

Site : <https://sinjaesung.github.io/SINJAweb/intro>

Youtube : <https://www.youtube.com/@sinja-g8q>

목표: Unity 및 관련 기술에 대한 전문지식을 활용하여 혁신적이고 지속적인 게임 및 VR/AR 프로젝트 생산에 기여합니다.

안녕하세요. 신재성입니다. 특정 한 분야에만 만족하며 멈추지 않고, 계속하여 노력&연구하며 그래픽스분야에서 뿐만 아니라 컴퓨터Science기술을 기반으로 하여 가능한 범위에서 다양한 첨단 분야(생명,전자,AI) 와의 콜라보레이션 기술공부&확보를 통해 지속적으로 성장하고자하는 사람이 되고 싶습니다. 우선 AI,ComputerGraphics,Computer Science에 집중할것입니다. 더 나아가서 세상에 도움이 될 수 있는 사람이 되고자 합니다.

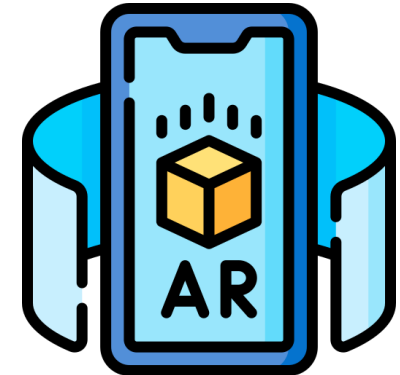
History

- 17.02 | 동양미래대학 컴퓨터정보공학 졸업
- 16.12 ~ 18.01 | NHN ACE 기업 근무(데이터지원팀)
- 18.01 ~ 19.03 | 개인프로젝트,수학스터디,그림,웹쟁이 작업활동 등 자기계발
- 19.04 ~ 19.10 | 그린컴퓨터아카데미 웹퍼블리셔 과정 수료
- 20.02 ~ 22.05 | (주)프라프 기업 근무(SI업체) 웹개발자(fullStack) 근무
- 22. 02 ~ 22.06 | (주)프라프 CLLLAP 모바일기반 영상편집웹앱제작 프리랜서형태로 마무리
- 22.10 ~ 23.01 | 프리랜서 활동 진행(웹개발,웹퍼블리싱 위주)
- 22.11 ~ 23.04 | 'SINJA' 프로젝트 관련된 blender(3d모델링) 에셋,애니메이션 제작
- 23.01 ~ 23.06 | 자체개발 threejs 웹게임 'SINJA' 개인 프로젝트 진행
- 23.07 ~ 23.09 | 웹에이전시 비닷컴크리에이티브 웹퍼블리셔 근무(과장급)
- 23.09 ~ 24.04 | 유니티 개발 작업진행(유니티게임 클라이언트개발)
- 24.05 ~ 현재 | 유니티 네트워크 포함 전반적 개발진행(유니티 관련 모든 콘텐츠)
- 24.06 ~ 25.02 | 방송정보국제교육원 AI활용한 실감형 콘텐츠 제작과정 교육(유니티VRAR게임 콘텐츠 제작)
- 자체적 블랜더 제작 캐릭터&애니메이션 6~8여종 존재.
- 하단에서 소개되겠지만 총 40~50종의 유니티 게임콘텐츠 제작 이력 존재

개발자로서의 기본기반 및 이해

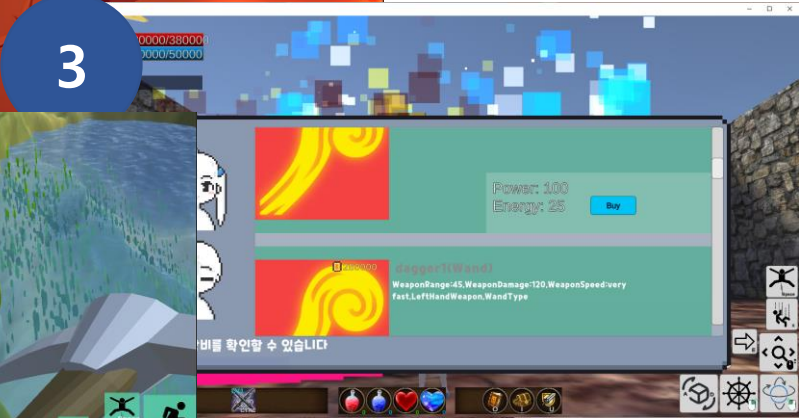


- 학력 : 동양미래대학교 컴퓨터정보공학과 전문학사
- 보유 기술 : 1.Unity에 익숙함(C#스크립팅,게임 메커니즘,물리,UI/UX)
- 2. 3D 모델링 및 애니메이션을 위한 Blender에 대한 지식
- 버전 제어 시스템(Git)에 대한 지식
- 3. 게임 디자인 원칙 및 알고리즘에 대한 이해
- 4. VR/AR 개발프레임워크(ARKIT,ARCore,Vuforia)에 대한 기본 지식
- 5. 관련 포트폴리오 결과물 링크 : <https://sinjaesung.github.io/SINJAweb/intro>



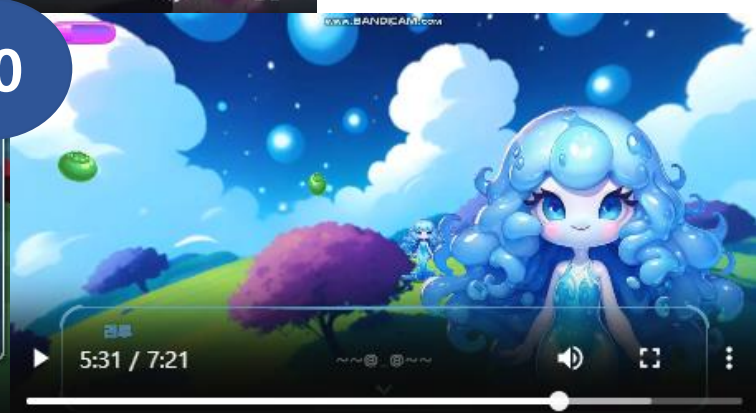
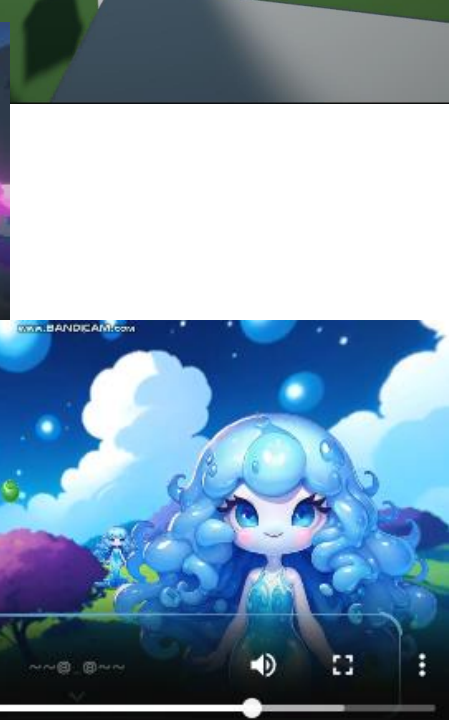
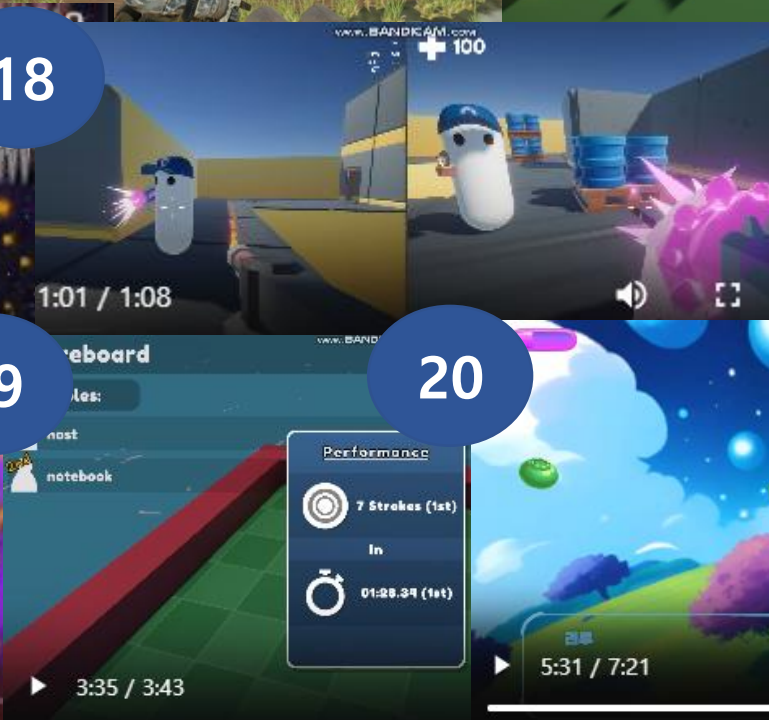
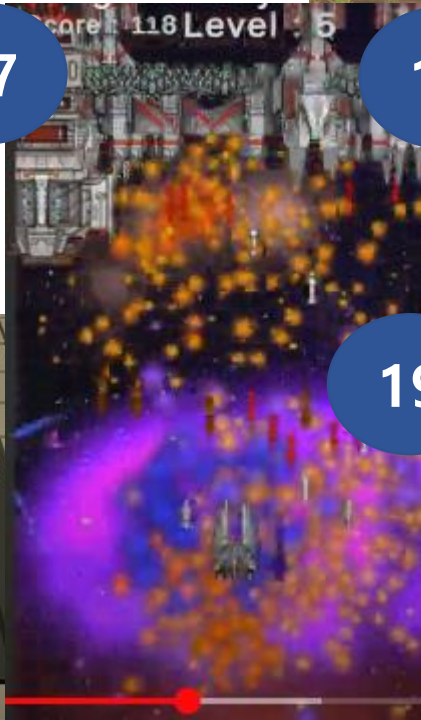
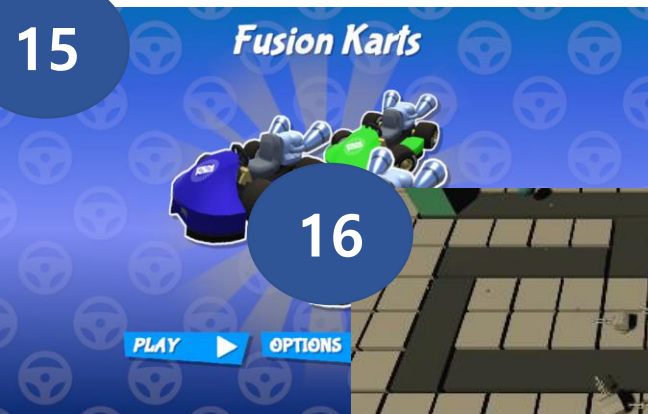
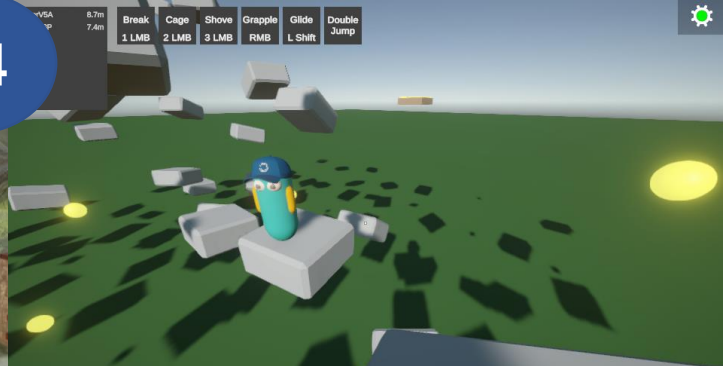
즉시 실무에 활용할 수 있는 기술 및 프로젝트 경험

- 1.멀티플레이어 simple fps 게임 (개인)
- 2.멀티플레이어 Tanknarok 게임 (개인)
- 3.싱글플레이어 SinjaUnityStory 게임 (개인)
- 4.싱글플레이어 SurvivalGame (개인)
- 5.간단한 상점시스템 포함 QuadActionRpg (개인)
- 6.오브젝트 풀링 기반 ShootRifle 몬스터 대전게임 (개인)
- 7.GTA확장형 RPG 게임 콘텐츠 (개인)
- 8.다대다 fps형식의 TDMClone 콘텐츠 (개인)
- 9.맵을 자유도있게 돌아다니는 파쿠르액션 게임 (개인)
- 10.GTAClone 개발 (개인)



- 11.BackgroundAdventure 자유도 탐험 콘텐츠 (개인)
- 12.Three.js 게임 개발 (개인)
- 13.멀티플레이어 총 게임 개발 (개인)
- 14.멀티플레이어 등반게임 개발 (개인)
- 15.멀티플레이어 카트경주 게임 개발 (개인)
- 16.싱글플레이어 타워디펜스 게임 개발 (개인)
- 17.싱글플레이어 2D Shooting 게임 개발 (개인)
- 18.멀티플레이어 Photonfusion_ProjectileAdvanced (개인)
- 19.멀티플레이어 골프게임 (개인)

20.싱글플레이어 러브 인 유니버스(2D 러브
시뮬레이션 게임) (팀플)



- 21.멀티플레이어 2d 비행기 슈팅게임 HostMode (개인)
- 22.멀티플레이어 2d 비행기 슈팅게임 SharedMode (개인)
- 23.멀티플레이어 2d 플랫폼머 게임 (개인)
- 24.멀티플레이어 스타터 프로젝트 웨어모드 (개인)
- 25.멀티플레이어 스타터 프로젝트 호스트모드 (개인)
- 26.멀티플레이어 보이스퀴즈 게임 (개인)
- 27.멀티플레이어 BR200 대규모 슈팅대전게임 (개인)
- 28.멀티플레이어 보이스 인터랙션 기반 마피아게임 (개인)
- 29.싱글플레이어 어쌔신크리드 캐릭터즈 파쿠르액션 (개인)
- 30.싱글플레이어 어쌔신크리드 미션 퀘스트 버전 (개인)

21



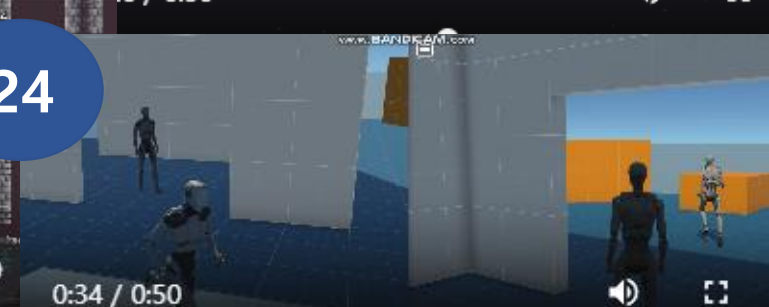
22



23



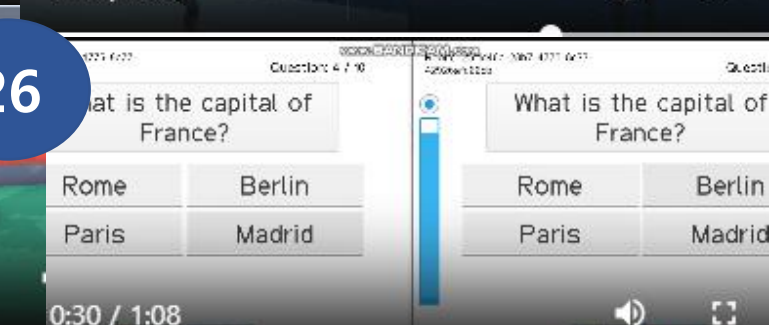
24



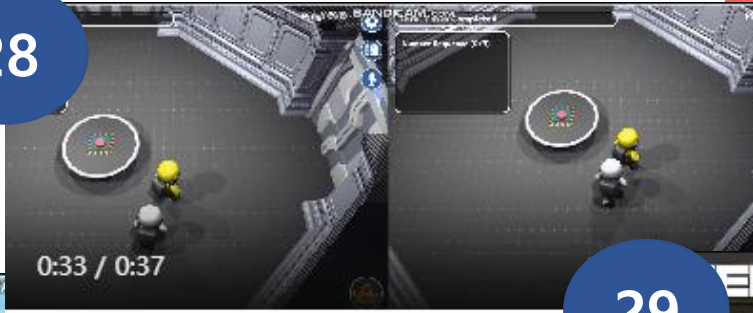
25



26



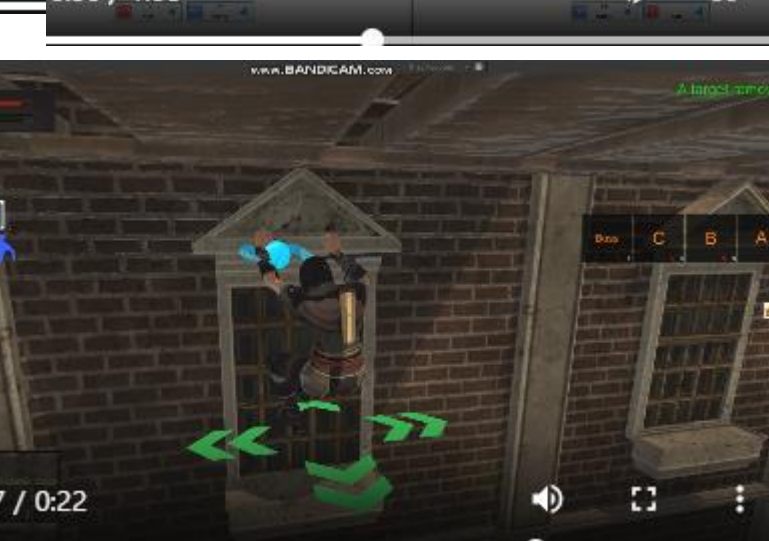
28



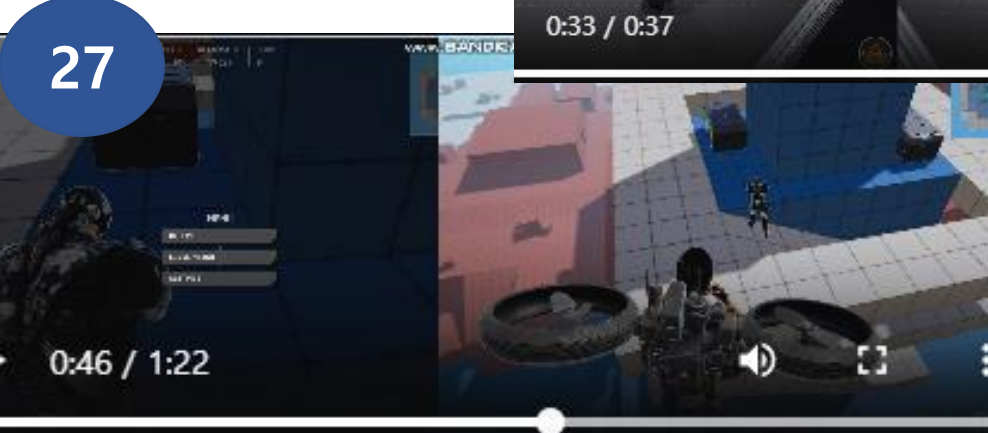
29



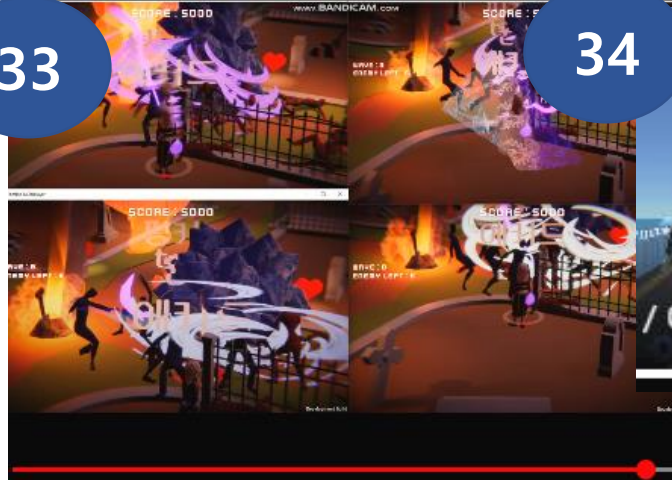
30



27



- 31.싱글플레이어 어쌔신 크리드 조선시대버전 (팀플)
- 32.멀티플레이어 MultiClimb_Characters (개인)
- 33.멀티플레이어 좀비대전 게임 (개인)
- 34.멀티플레이어 카트경주게임 캐릭터 커스텀 버전 (개인)
- 35.멀티플레이어 탱크대전게임 캐릭터 커스텀 버전 (개인)
- 36.멀티플레이어 Simplefps 캐릭터 커스텀 버전 (개인)
- 37.멀티플레이어 캐릭터 슈팅 대전 게임 (개인)
- 38.멀티플레이어 캐릭터,Titan 대전 게임 (팀플)



- 39.싱글플레이어 호러게임 (개인)
- 40.멀티플레이어 보이스캐릭터 스타일러 콘텐츠 (개인)
- 41.멀티플레이어 캐릭터 슈팅대전 시스템확장버전1(pvp,pve) (개인)
- 42.멀티플레이어 캐릭터 슈팅대전 시스템확장버전2(pvp,pve) (개인)
- 43.멀티플레이어 몬스터 레이드 게임(pvp,pve) (팀플)

39



43



40



41



42

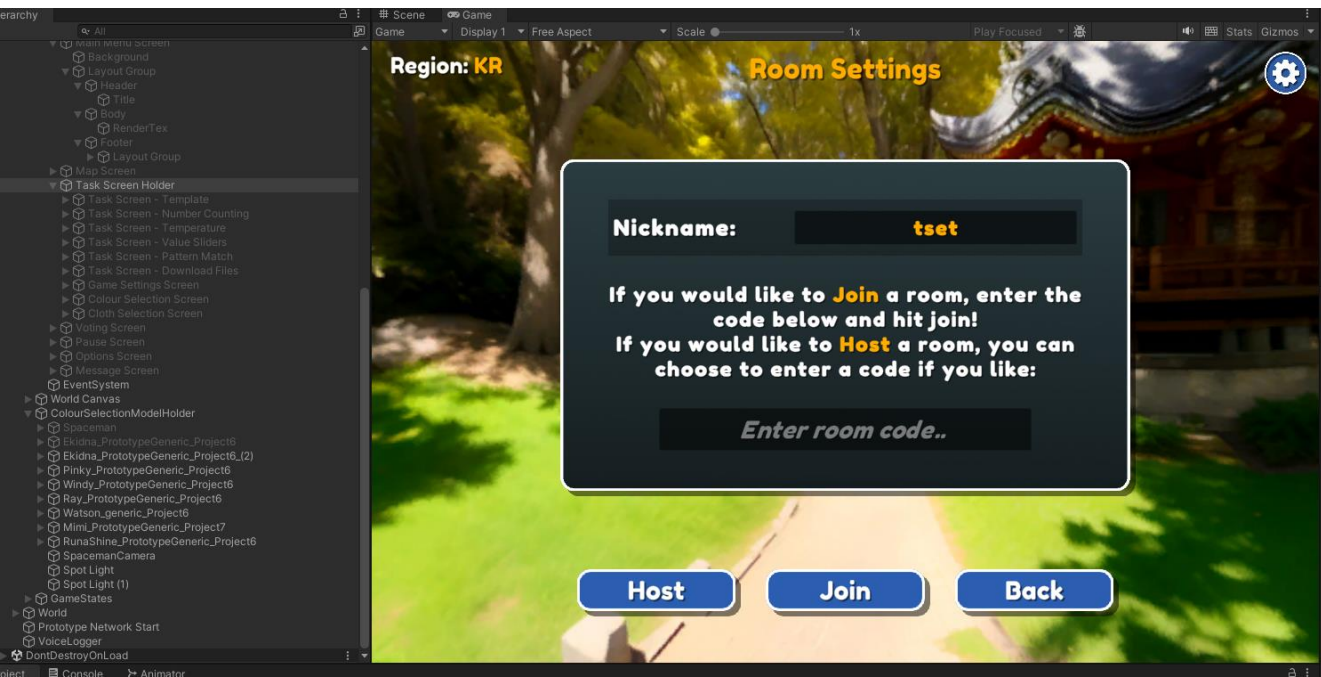


프로젝트1. 멀티플레이어 보이스 캐릭터 스타일러

개요: 플레이어별 랜덤한 서로 다른 캐릭터로 접속가능하며, 각 캐릭터별 고유한 자신만의 의상을 바꿀 수 있으며, 캐릭터의 눈&헤어 색상 또한 바꿀수 있다. 캐릭터의 스타일을 여러가지로 꾸며볼 수 있는 멀티플레이 컨텐츠.

작업 기간: 2024.12.16~2024.12.30 (2주)

작업 분담: 1인개발(개발 총괄)



TASK 부분

주요한 기능&스크립트 요약

```
using UnityEngine;

/// <summary>
/// Base class for different tasks.
/// </summary>
/// Unity 스크립트 참조 18개
public abstract class TaskBase : MonoBehaviour
{
    public TaskUI taskUI;
    참조 7개
    public abstract string Name { get; }

    참조 9개
    public abstract void ResetTask();
    참조 5개
    public virtual void Completed()
    {
        AudioManager.Play("taskCompleteUI", AudioManager.MixerTarget.UI);
        GameManager.Instance.CompleteTask();
        taskUI.Complete();
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Fusion;

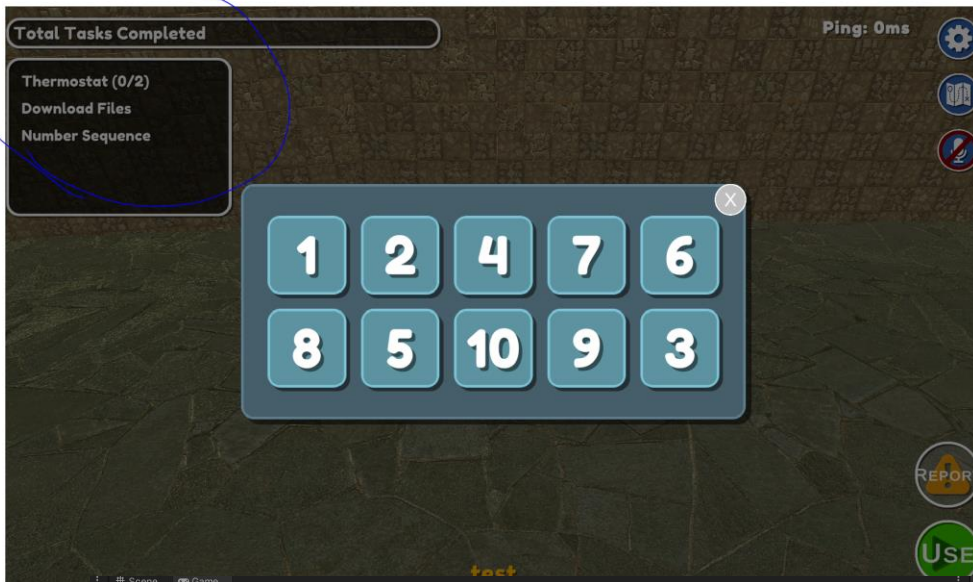
/// <summary>
/// Interactable that starts tasks for the local player.
/// </summary>
/// Unity 스크립트(자산 참조 28개) 참조 17개
public class TaskStation : Interactable
{
    public TaskUI taskUI;

    [Networked, Capacity(10)]
    참조 0개
    public NetworkLinkedList<PlayerRef> allowedPlayers => default;

    참조 4개
    public override bool CanInteract(PlayerMovement player)
    {
        return !player.IsSuspect && player.tasks.Contains(this);
    }

    참조 5개
    public override void Interact(NetworkObject interactor)
    {
        if (!interactor.HasInputAuthority)
            return;

        taskUI.Begin();
        Debug.Log("Began task");
    }
}
```



TaskBase > 상속

DownloadTasks, NumberSequenceTask, PatternMatchTask, SlidersTask, TemperatureTask

플레이어별 랜덤하게 부여되는 임무에 관한 클래스요소들

TaskStation

플레이어별 부여된 Task 관련하여 플레이어와 Interactable 관련 처리를 담당하는 클래스



캐릭터 스타일

주요한 기능&스크립트 요약

```
using Helpers.Collections;
using System.Collections;
using UnityEngine;

/// <summary>
/// Behaviour that handles non-networked information for players such as animation location on
/// </summary>
/// Unity 스크립트(자산 참조 14개) | 참조 2개
public class CharacterVisual : MonoBehaviour

{
    public Renderer[] modelParts; //dynamic change

    public Material HairMaterial;
    public Material EyeMaterial;

    public GameObject[] TopCloths;
    public GameObject[] BotCloths;

    public bool isExecuted = false;

    @Unity 메시지 | 참조 0개
    private void Awake()
    {
        for (int e = 0; e < TopCloths.Length; e++)
        {
            TopCloths[e].SetActive(false);
        }
        IF (TopCloths.Length > 0)
            TopCloths[0].SetActive(true);

        for (int e = 0; e < BotCloths.Length; e++)
        {
            BotCloths[e].SetActive(false);
        }
        IF (BotCloths.Length > 0)
            BotCloths[0].SetActive(true);
    }

    참조 2개
    public void UpdateRenderCoroutine()
    {
        StartCoroutine(UpdateRender());
    }

    참조 1개
    private IEnumerator UpdateRender()
    {
        while (PlayerObject.Local)
        {
            isExecuted = true;
        }
    }
}
```

CharaterVisual

캐릭터별 소속된 스타일메쉬들&메터리얼들 관련하여 색상변경, 메쉬변경 등의 스타일 관련된 부분을 처리하는 관련 클래스

```
HairMaterial = PlayerObject.Local.GetComponent<PlayerData>().HairMaterial;
EyeMaterial = PlayerObject.Local.GetComponent<PlayerData>().EyeMaterial;
//각각의 서버 플레이어 생성일것이다. Clone된 BodySuit메터리얼 이걸 적용하며 Suit에 해당하는 Mat를 가진 개별적 개체들은 이 mat를 공유한다.
//BodyMaterial = Instantiate(GameManager.rm.playerBodyMaterial); //생성 시점에 외부 리소스 공통용 Material을 각 자신의 모델파트에 적용한다.
//TrimMaterial = Instantiate(GameManager.rm.playerTrimMaterial);
//VisorMaterial = Instantiate(GameManager.rm.playerVisorMaterial);

modelParts.ForEach(ren =>
{
    Material[] mats = ren.sharedMaterials;
    for (int i = 0; i < mats.Length; i++)
    {
        if (mats[i] == HairMaterial)
        {
            Debug.Log("CharacterVisual HairMaterial modelParts" + mats[i] = PlayerObject.Local.GetComponent<PlayerData>().HairMaterial);
        }
        if (mats[i] == EyeMaterial)
        {
            Debug.Log("CharacterVisual EyeMaterial modelParts" + mats[i] = PlayerObject.Local.GetComponent<PlayerData>().EyeMaterial);
        }
        else
        {
            Debug.LogWarning($"(ren) is using an unmanaged material");
        }
    }
    ren.sharedMaterials = mats;
});
Debug.Log($"Finished initializing materials", gameObject);

/// The base class used for all interactables such as the color kiosk, tasks,
/// </summary>
/// Unity 스크립트 | 참조 16개
public abstract class Interactable : MonoBehaviour

{
    public bool isInteractionInstant = false;
    public bool isGhostAccessible = true;

    참조 9개
    public abstract bool CanInteract(PlayerMovement player);

    참조 10개
    public abstract void Interact(NetworkObject interactor);
}
```

```
for(int e=0; e<CharacterLocalVisual.Length; e++)
{
    CharacterLocalVisual[e].SetActive(false);
}
CharacterLocalVisual[PlayerObject.Local.Ref.AsIndex - 1].SetActive(true);
CharacterLocalVisual[PlayerObject.Local.Ref.AsIndex - 1].GetComponent<CharacterVisual>().UpdateRenderCoroutine();

for (byte c = 1; c < GameManager.rm.playerColours.Length; c++)
{
    ColorOption colorOption = Instantiate(optionPrefab, selectionHolder);
    colorOption.Init(toggleGroup, c);
    options.Add(c, colorOption);
}

/*dummyRenderers.ForEach(ren =>
{
    Material[] mats = ren.sharedMaterials;
    for (int i = 0; i < mats.Length; i++)
    {
        if (mats[i] == PlayerObject.Local.GetComponent<PlayerData>().HairMaterial)
        {
            mats[i] = PlayerObject.Local.GetComponent<PlayerData>().HairMaterial.Instance;
        }
        else if (mats[i] == PlayerObject.Local.GetComponent<PlayerData>().EyeMaterial)
        {
            mats[i] = PlayerObject.Local.GetComponent<PlayerData>().EyeMaterial.Instance;
        }
    }
    ren.sharedMaterials = mats;
});
```

```
/// <summary>
/// Interactable used for opening up the player color selector.
/// </summary>
/// Unity 스크립트(자산 참조 4개) | 참조 0개
public class ClothKiosk : Interactable
{
    public ClothSelectionUI clothUI;

    참조 4개
    public override bool CanInteract(PlayerMovement player) => true;

    참조 5개
    public override void Interact(NetworkObject interactor)
    {
        if (interactor.HasInputAuthority)
            return;

        clothUI.Open();
    }
}

/// <summary>
/// Unity 스크립트(자산 참조 2개) | 참조 1개
public class ClothSelectionUI : MonoBehaviour

{
    public Transform selectionHolder;
    public ToggleGroup toggleGroup;
    public ClothOption optionPrefab;

    public GameObject[] CharacterLocalVisual;

    readonly Dictionary<int, ClothOption> options = new Dictionary<int, ClothOption>();
    bool hasInit = false;

    참조 1개
    public void Init()
    {
        if (hasInit) return;

        for (int e = 0; e < CharacterLocalVisual.Length; e++)
        {
            CharacterLocalVisual[e].SetActive(false);
        }
        CharacterLocalVisual[PlayerObject.Local.Ref.AsIndex - 1].SetActive(true);
        CharacterLocalVisual[PlayerObject.Local.Ref.AsIndex - 1].GetComponent<CharacterVisual>().UpdateRenderCoroutine();

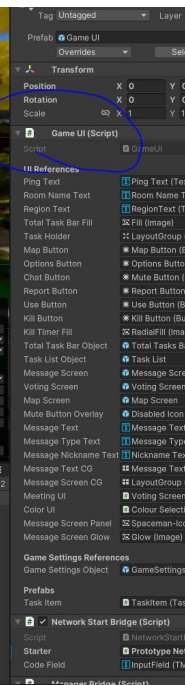
        for (int c = 0; c < PlayerObject.Local.GetComponent<PlayerData>().TopCloths.Length; c++)
        {
            ClothOption clothOption = Instantiate(optionPrefab, selectionHolder);
            clothOption.Init(toggleGroup, c);
            options.Add(c, clothOption);
        }

        /* dummyRenderers.ForEach(ren =>
        {
            Material[] mats = ren.sharedMaterials;
            for (int i = 0; i < mats.Length; i++)
            {
                if (mats[i] == PlayerObject.Local.GetComponent<PlayerData>().HairMaterial)
                {
                    mats[i] = PlayerObject.Local.GetComponent<PlayerData>().HairMaterial.Instance;
                }
                else if (mats[i] == PlayerObject.Local.GetComponent<PlayerData>().EyeMaterial)
                {
                    mats[i] = PlayerObject.Local.GetComponent<PlayerData>().EyeMaterial.Instance;
                }
            }
            ren.sharedMaterials = mats;
        });
```

ClothKiosk, ClothOption, ClothSelectionUI, ColorKiosk, ColorOption, ColorSelectionUI

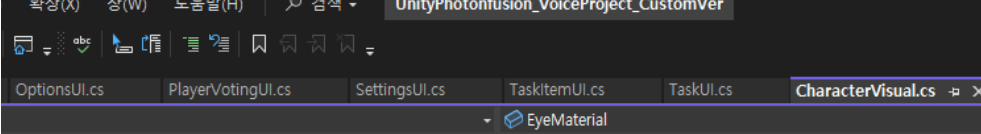
캐릭터별 헤어&눈색상, 의상Swap 등의 관련된 조작을 명령하는 관련 클래스

시스템 전체 스크립트 구조

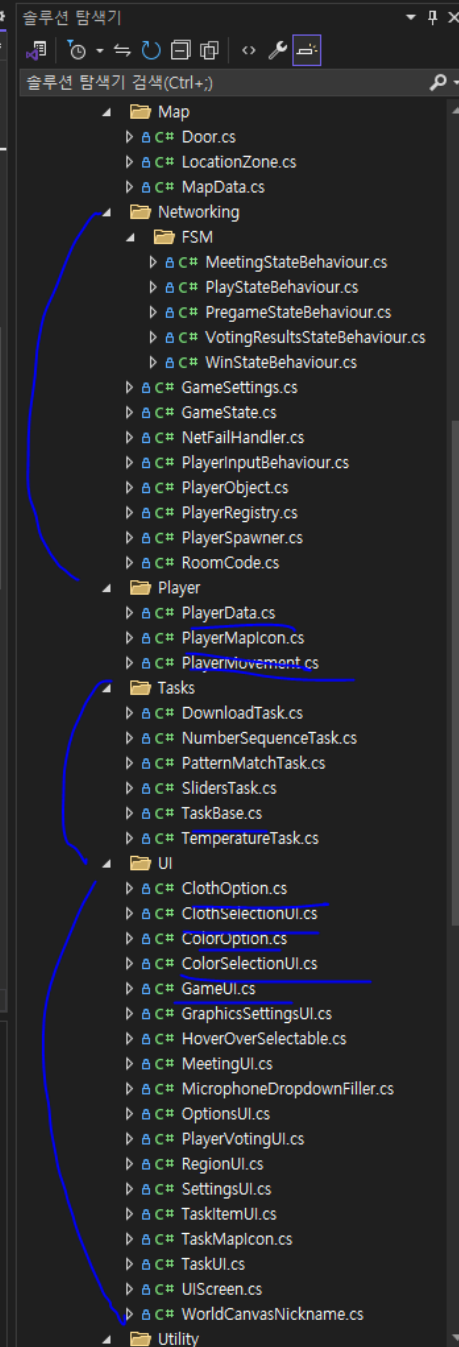


```
/// <summary>
/// Class that maintains the game's UI and its various references.
/// </summary>
/// <summary>
/// Unity 스크립트(자산 참조 1개) 참조 1개
public class GameUI : MonoBehaviour

[Header("UI References")]
public TMP_Text pingText;
public TMP_Text roomNameText;
public TMP_Text regionText;
public Image totalTaskBarFill;
public Transform taskHolder;
public Button mapButton;
public Button optionsButton;
public Button chatButton;
public Button reportButton;
public Button useButton;
public Button killButton;
public Image killTimerFill;
public GameObject totalTaskBarObject;
public GameObject taskListObject;
public GameObject messageScreen;
public GameObject votingScreen;
public GameObject mapScreen;
public GameObject muteButtonOverlay;
public TMP_Text messageText;
public TMP_Text messageTypeText;
public TMP_Text messageNicknameText;
public CanvasGroup messageTextOG;
public CanvasGroup messageTextOG;
public CanvasGroup messageScreenOG;
public MeetingUI meetingUI;
public ColorSelectionUI colorUI;
public Image messageScreenPanel;
public Image messageScreenGlow;
[Header("Game Settings References")]
public GameObject gameSettingsObject;
```



해당하는 Mat를 가진 개별적 개체들은 이 mat를 공유한다.
리소스 공통용 Material을 각 자신의 모델파트에 적용한다. (RenderSharedMaterials: 공유메터리얼매시)



주요한 기능&스크립트 요약(카테고리)

- Audio
 - AudioDestroyer.cs
 - AudioManager.cs
 - UIAudio.cs
- Bridge
 - HUDBridge.cs
 - ManagerBridge.cs
 - NetworkStartBridge.cs
 - NicknameSetter.cs
- Interactables
 - ClothKiosk.cs
 - ColorKiosk.cs
 - DeadPlayer.cs
 - EmergencyButton.cs
 - Interactable.cs
 - SettingsKiosk.cs
 - TaskStation.cs

오디오관련, 허드 & 매니저 & 네트워크 브릿지, 닉네임 설정자, 의상 & 색상 스타일 키오스크, 투표버튼, 인터랙터블 상위 클래스, 셋팅 키오스크, 과업 스테이션

- Managers
 - GameManager.cs
 - InterfaceManager.cs
 - ResourcesManager.cs
 - VoiceManager.cs
- Map
 - Door.cs
 - LocationZone.cs
 - MapData.cs

게임, 인터페이스, 리소스, 보이스 매니저, 맵 데이터, 맵 인터렉션 환경 오브젝트

- Player
 - PlayerData.cs
 - PlayerMapIcon.cs
 - PlayerMovement.cs
- Tasks
 - DownloadTask.cs
 - NumberSequenceTask.cs
 - PatternMatchTask.cs
 - SlidersTask.cs
 - TaskBase.cs
 - TemperatureTask.cs

플레이어 데이터, 플레이어 위치 미니맵 표시, 플레이어 조작, TaskBase 기반 과업 리스트 정의(과업종류별 플레이어별 랜덤 할당)

- Networking
 - FSM
 - MeetingStateBehaviour.cs
 - PlayStateBehaviour.cs
 - PregameStateBehaviour.cs
 - VotingResultsStateBehaviour.cs
 - WinStateBehaviour.cs
 - GameSettings.cs
 - GameState.cs
 - NetFailHandler.cs
 - PlayerInputBehaviour.cs
 - PlayerObject.cs
 - PlayerRegistry.cs
 - PlayerSpawner.cs
 - RoomCode.cs

회의 상태, 플레이 상태, 프리 게임 상태, 투표 결과, 승자 상태 행동자 FSM, 게임 셋팅, 게임 상태, 플레이어 입력 행동자, 플레이어 등록, 생성자, 플레이어 오브젝트

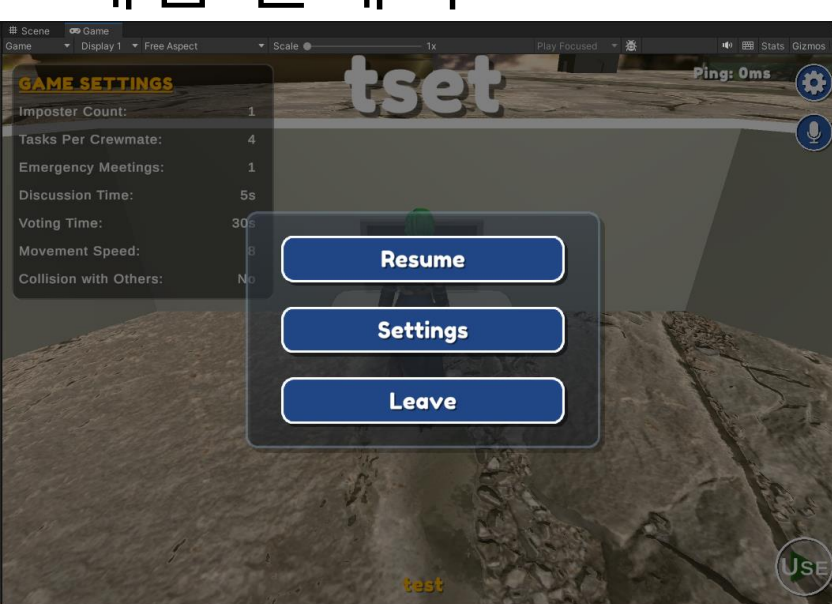
- UI
 - ClothOption.cs
 - ClothSelectionUI.cs
 - ColorOption.cs
 - ColorSelectionUI.cs
 - GameUI.cs
 - GraphicsSettingsUI.cs
 - HoverOverSelectable.cs
 - MeetingUI.cs
 - MicrophoneDropdownFiller.cs
 - OptionsUI.cs
 - PlayerVotingUI.cs
 - RegionUI.cs
 - SettingsUI.cs
 - TaskItemUI.cs
 - TaskMapIcon.cs
 - TaskUI.cs
 - UIScreen.cs
 - WorldCanvasNickname.cs

의상 옵션, 의상 선택, 컬러 옵션, 컬러 선택, 게임 UI, 그래픽 셋팅, 보이 스미팅 UI, 헤드폰 보이 스 셋팅, 투표 UI, 세션 지역 설정, 게임 셋팅 UI, 과업 아이템 UI, 과업 리스트 UI, UIScreen 등의 UI 관련

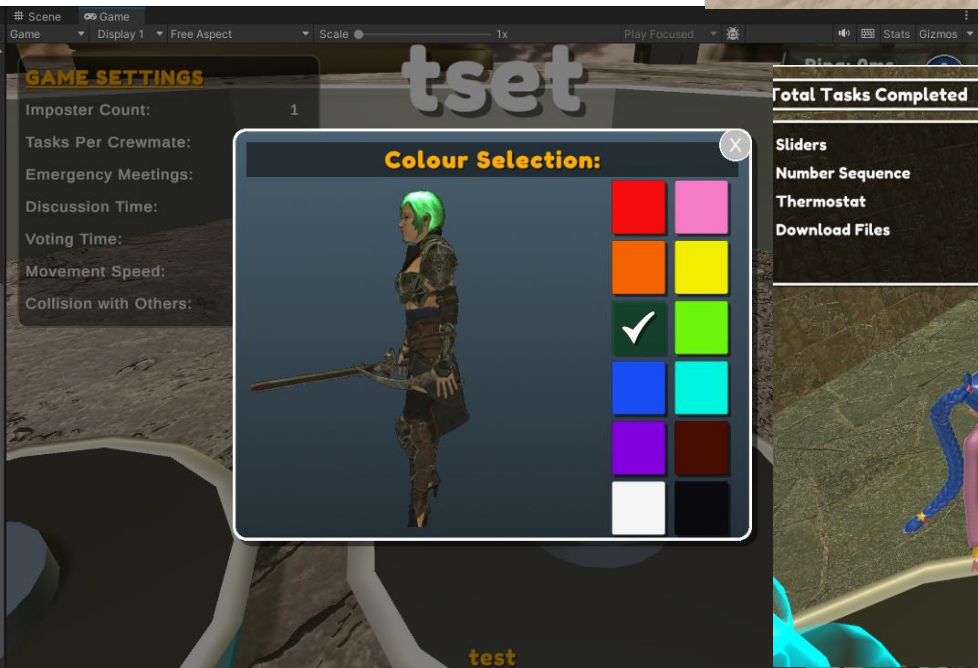
- Utility
 - Attribute
 - ReadOnlyAttribute.cs
 - Helpers
 - BitUtil.cs
 - CollectionsUtil.cs
 - CommonUtil.cs
 - PhysicsUtil.cs
 - ActionOnSelect.cs
 - CameraFollow.cs
 - CanvasFader.cs
 - FollowTarget.cs
 - FullscreenToggle.cs
 - MapCamera.cs
 - RandomIconSelector.cs
 - ScalePulse.cs
 - ScrollingRawImage.cs
 - Spin.cs
 - VersionText.cs
- CharacterVisual.cs
- SceneMover.cs

게임 오브젝트 속성 표현자, 유틸리티 헬퍼 클래스, 선택시 액션, 카메라 플레이어 추적, UICanvas 페이더, 맵 카메라, 스크롤 이미지 맵, 스핀, UI 크기 효과, 비주얼 캐릭터 스타일 뷰 등

• 게임 플레이



• 게임 플레이

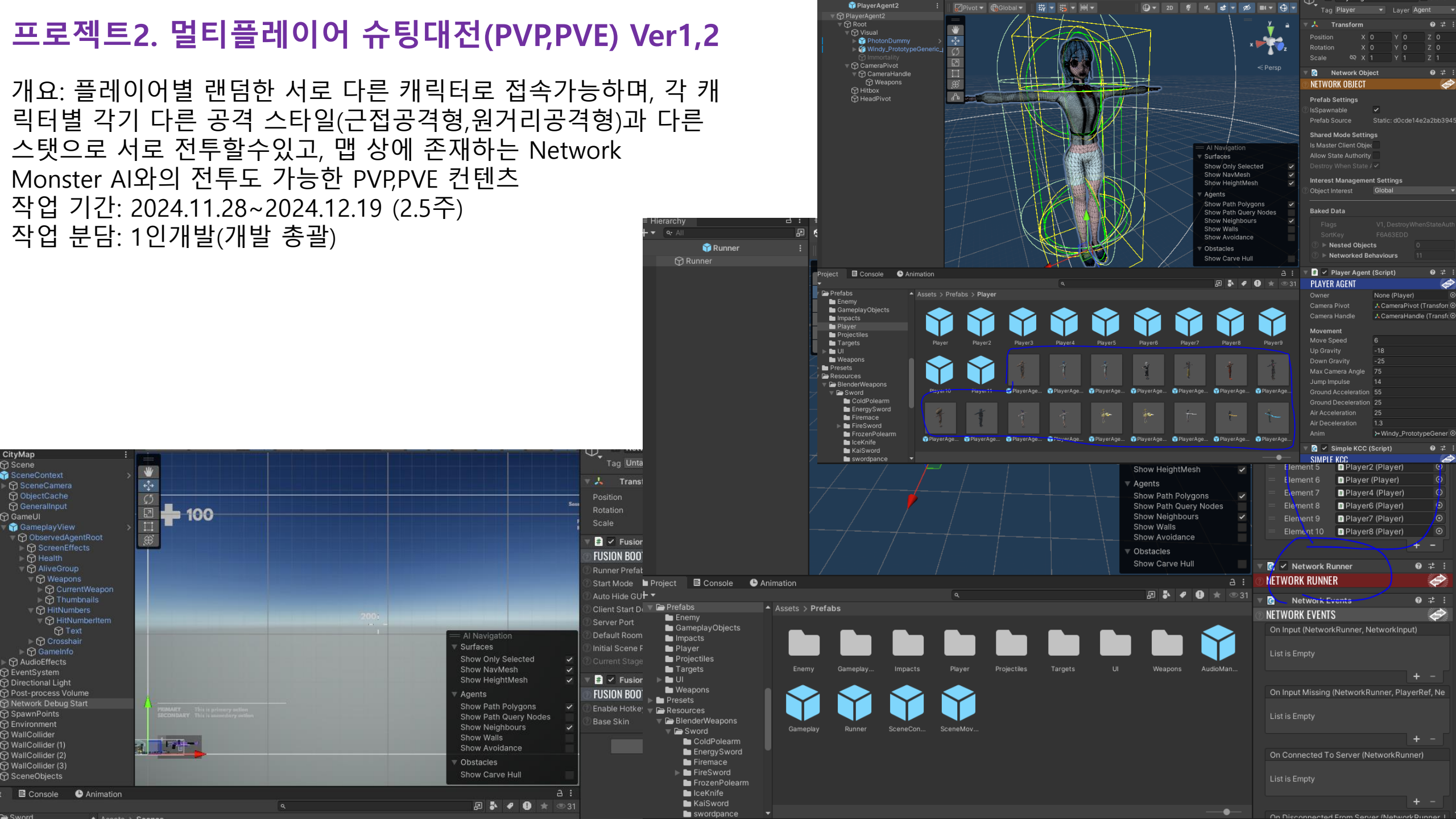


프로젝트2. 멀티플레이어 슈팅대전(PVP,PVE) Ver1,2

개요: 플레이어별 랜덤한 서로 다른 캐릭터로 접속가능하며, 각 캐릭터별 각기 다른 공격 스타일(근접공격형,원거리공격형)과 다른 스탯으로 서로 전투할수있고, 맵 상에 존재하는 Network Monster AI와의 전투도 가능한 PVP,PVE 콘텐츠

작업 기간: 2024.11.28~2024.12.19 (2.5주)

작업 분담: 1인개발(개발 총괄)



체력,데미지처리

주요한 기능&스크립트 요약

```
using System;
using Fusion;
using UnityEngine;

namespace Projectiles
{
    /// <summary>
    /// Handles synchronization of health state and hits (hit reactions) between client and server.
    /// </summary>
    /// Unity 스크립트 (자산 참조 23개) | 참조 16개
    public class Health : ContextBehaviour, IHitTarget, IHitInstigator
    {
        // PUBLIC MEMBERS

        public event Action<HitData> HitTaken;
        public event Action<HitData> HitPerformed;
        public event Action<HitData> FatalHitTaken;

        참조 26개
        public bool IsAlive => CurrentHealth > 0f;
        참조 2개
        public bool IsImmortal => _immortalCooldown.ExpiredOrNotRunning(Runner) == false;
        참조 2개
        public float MaxHealth => _maxHealth;

        [Networked]
        참조 12개
        public float CurrentHealth { get; private set; }

        // PRIVATE MEMBERS

        [SerializeField]
        private float _maxHealth = 100f;
        [SerializeField]
        private Transform _headPivot;
        [SerializeField]
        private Transform _bodyPivot;
        [SerializeField]
        private Transform _groundPivot;
        [SerializeField]
        private Hitbox _bodyHitbox;
    }
}
```

```
/// <summary>
/// A utility that encapsulates common approach of handling hits.
/// </summary>
참조 13개
public static class HitUtility
{
    // PUBLIC METHODS

    참조 1개
    public static void GetAllTargets(NetworkRunner runner, List<IHitTarget> targets)
    {
        targets.Clear();

        var healths = ListPool.Get<Health>(targets.Count);

        // TODO: Best would be to get IHitTargets directly, but that is not possible
        runner.GetAllBehaviours(healths);

        if (onlyActive == true)
        {
            for (int i = 0; i < healths.Count; i++)
            {
                var target = healths[i] as IHitTarget;

                if (target.IsActive == true)
                {
                    targets.Add(target);
                }
            }
        }
        else
        {
            targets.AddRange(healths);
        }

        ListPool.Return(healths);
    }

    참조 7개
    public static HitData ProcessHit(PlayerRef instigatorRef, Vector3 direction,
    {
        var target = GetHitTarget(hit.Hitbox, hit.Collider);
        if (target == null)
        {
            return default;
        }
    }
}
```

Health

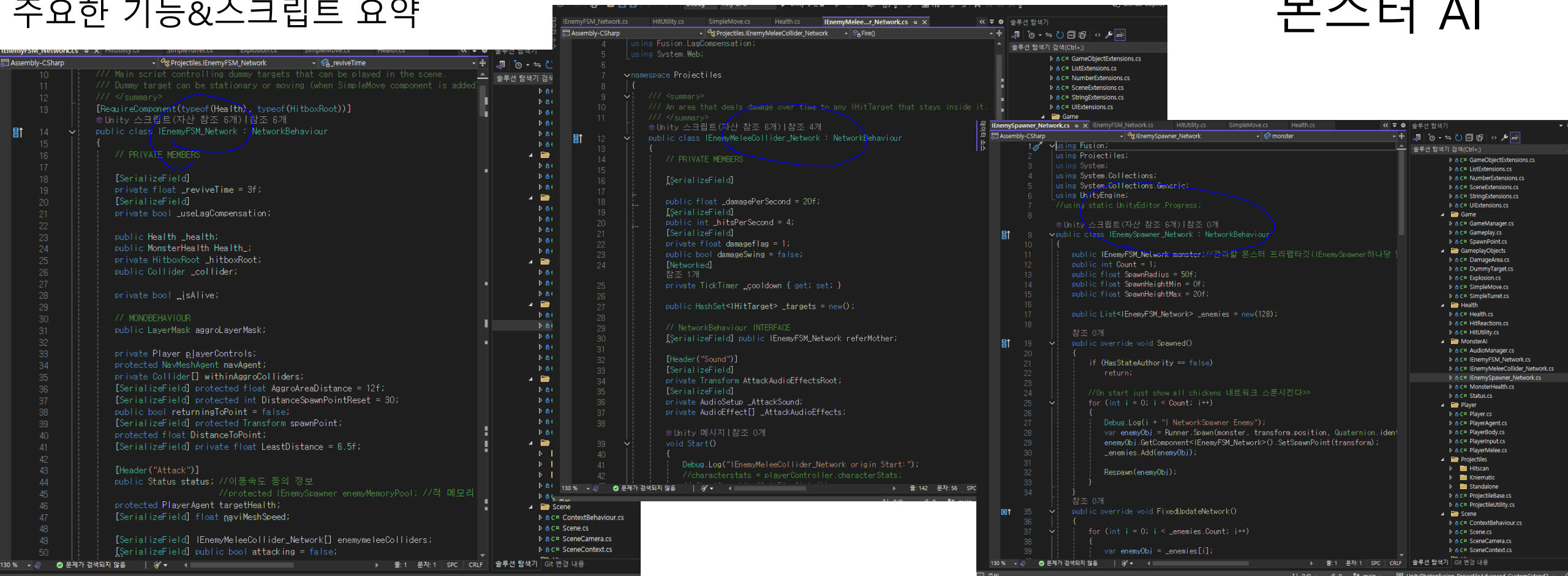
플레이어&몬스터의 체력관련 정보 처리 스크립트

HitUtility

플레이어의 데미지 피격 처리등의 데이터를 처리하는 관련 스크립트

주요한 기능&스크립트 요약

몬스터 AI



IEnemyFSM_Network

몬스터 AI 행동 결정 시스템
대기,추적,공격 관련 행동 의사결정
및 행동형태 네트워크 동기화

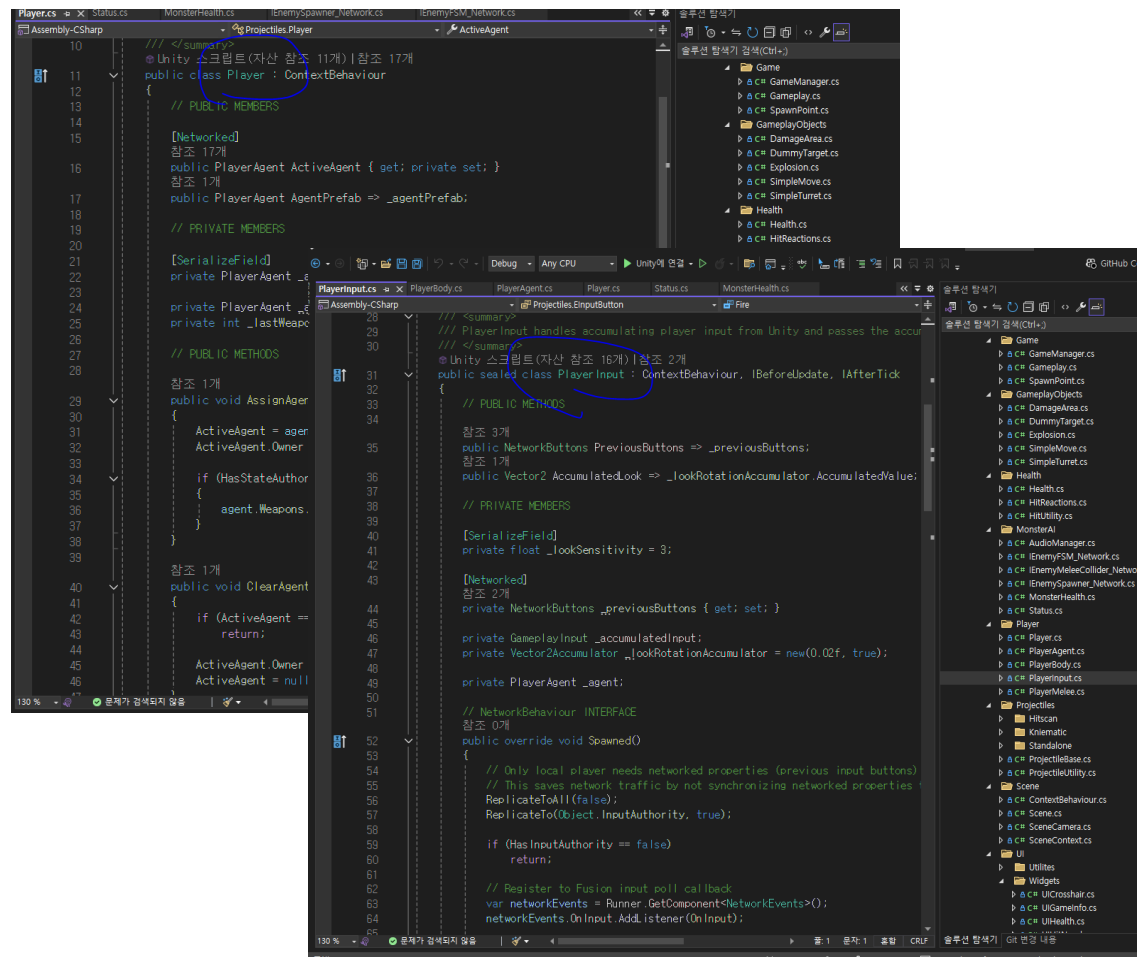
IEnemyMeleeCollider_Network

몬스터 AI 근접 공격 관련 피격처리

IEnemySpawner_Network

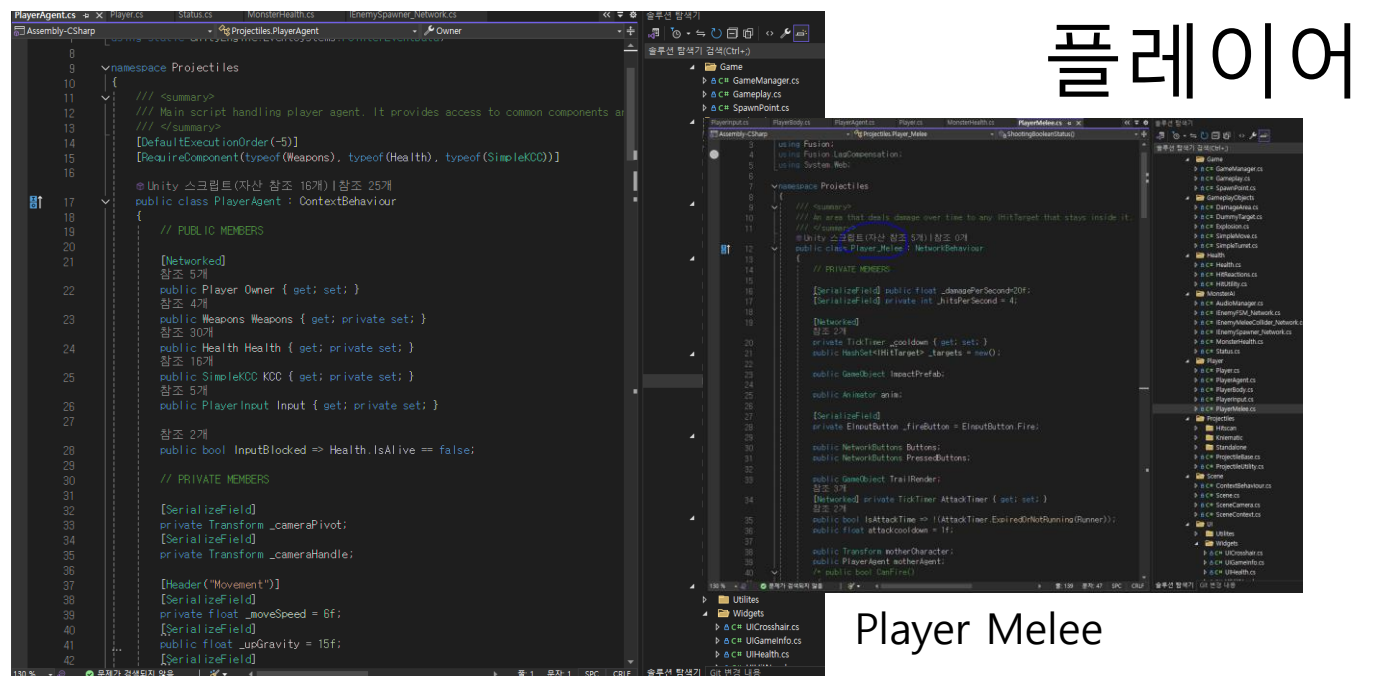
몬스터 AI 주기적 생성&삭제 관련
관리 클래스

주요한 기능&스크립트 요약



Player & PlayerInput

플레이어 Globe데이터 관리 및 플레이어 네트워크 입력 관련 처리

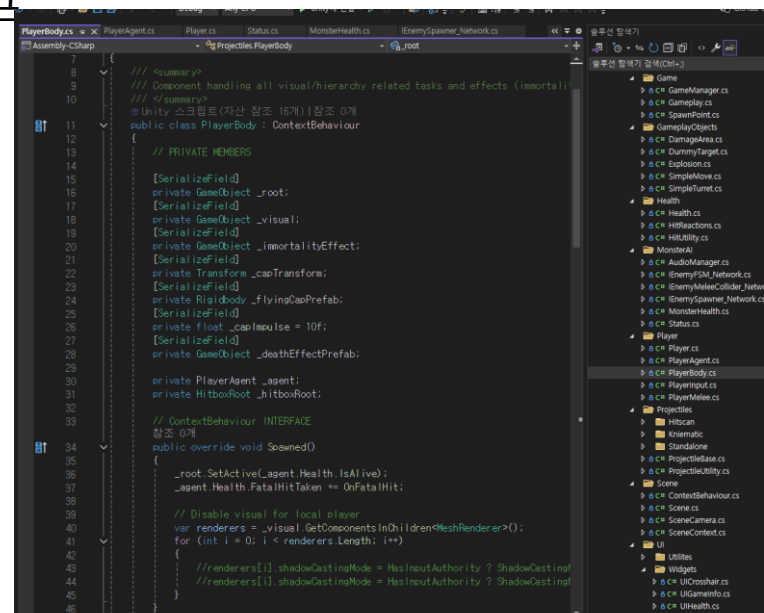


Player Agent

플레이어 애니메이션, 이동&점프, 공격 등의 입력관련 처리 등의
조작 주체적 처리

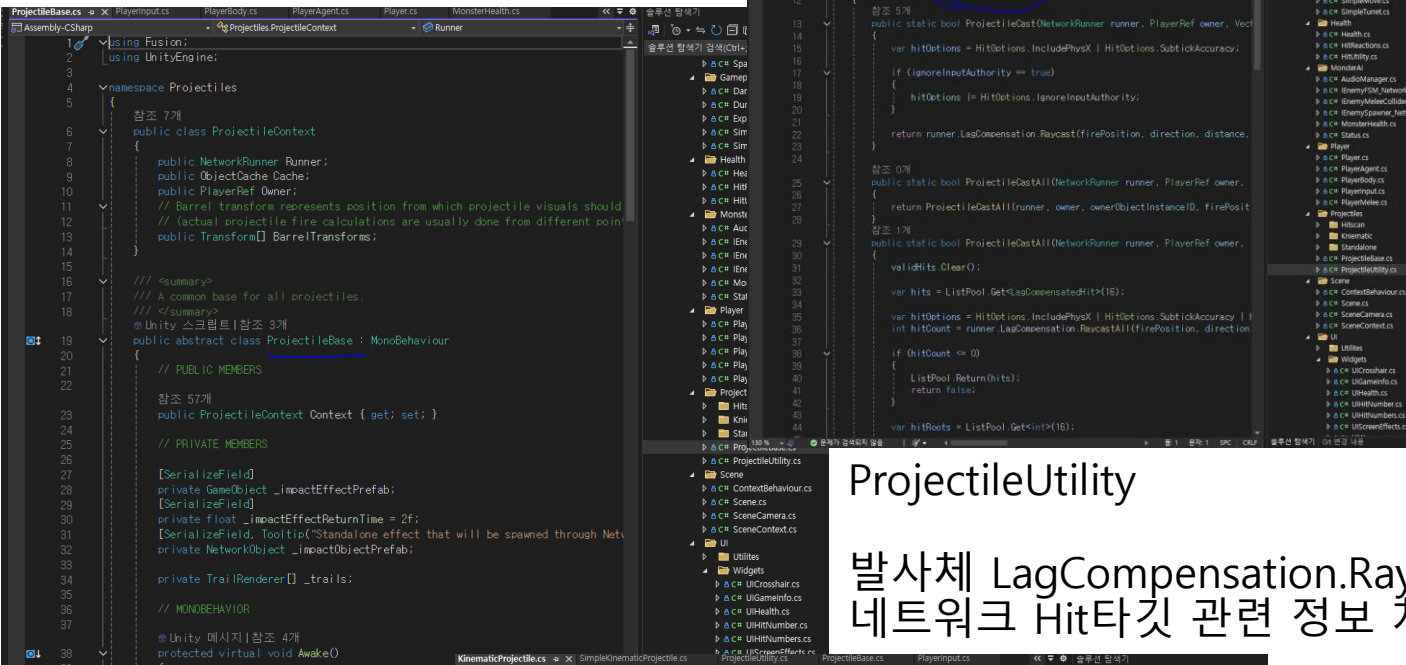
PlayerBody

테러이러 외형 관련 관리



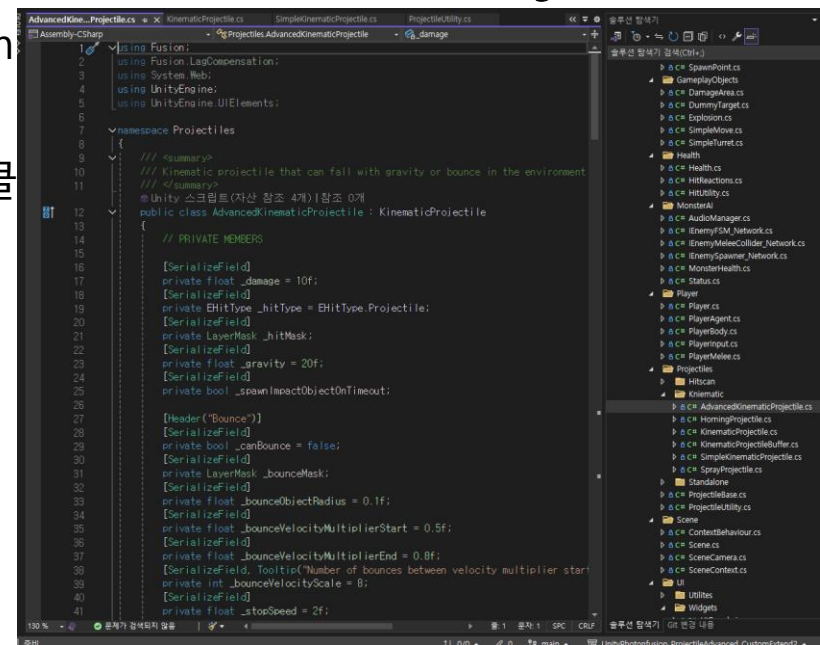
Projectiles

주요한 기능&스크립트 요약



AdvancedKinematicProjectile

키네마틱 상속클래스, 기본형에 더해서 반사, 바운스 효과 추가

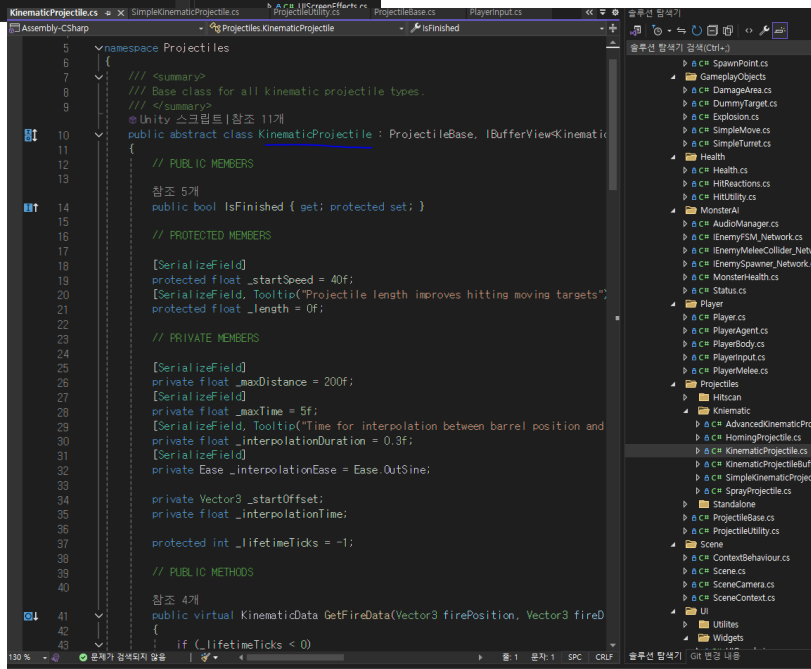


ProjectileUtility

발사체 LagCompensation.Raycast 네트워크 Hit타킷 관련 정보 처리

ProjectileBase

프로젝타일(발사체) 관련 기본 슈퍼클래스

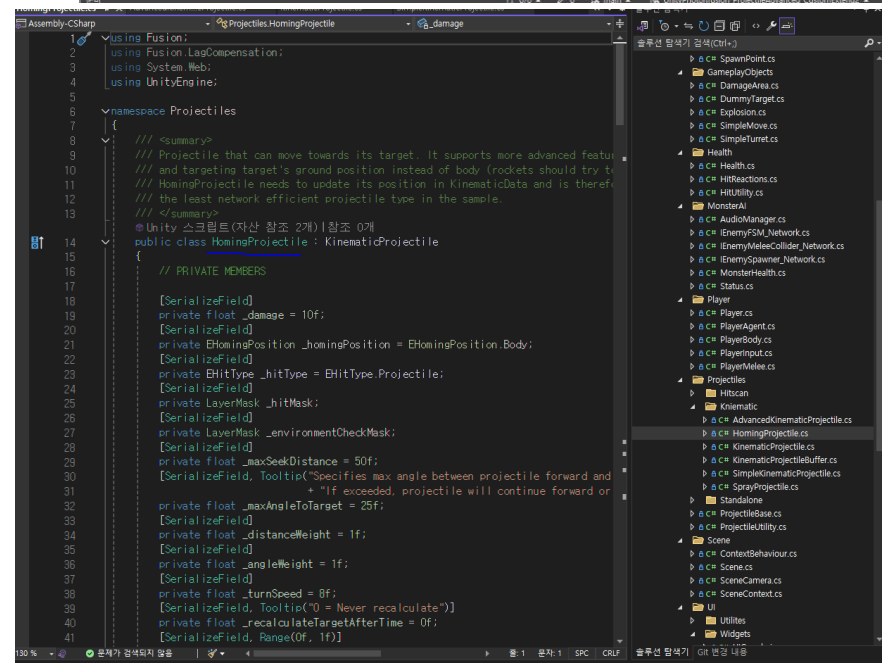


KinematicProjectile

키네마틱 프로젝트일 기본형 클래스

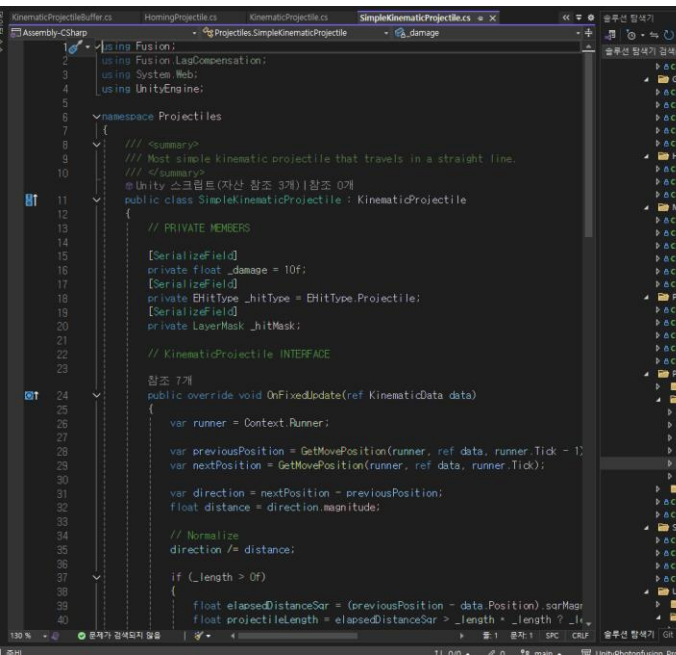
HomingProjectile

키네마틱 상속클래스, 타킷추적유도미사일&타킷위치추정 등의 기능 추가



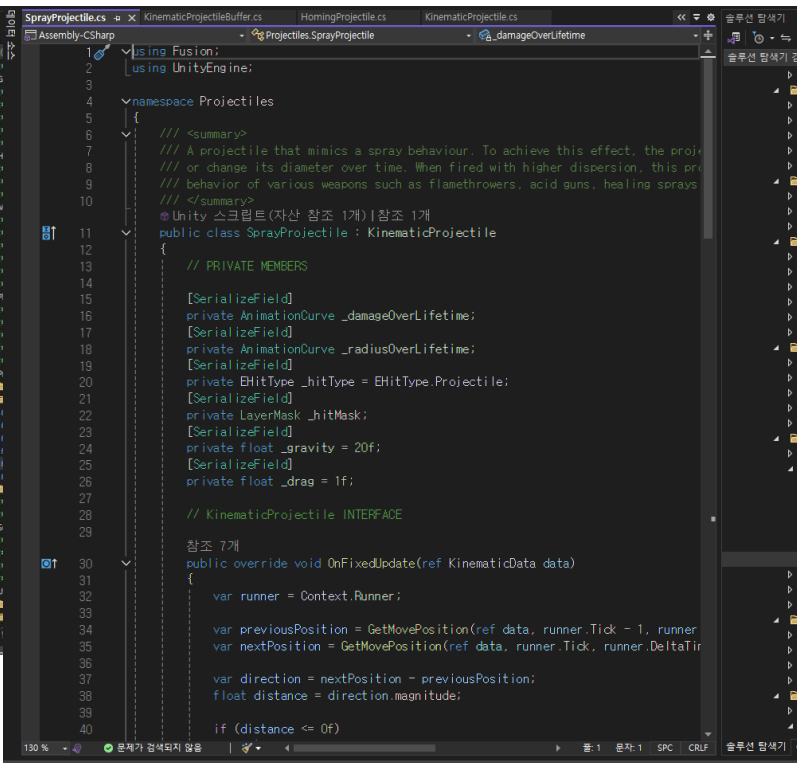
Projectiles

주요한 기능&스크립트 요약



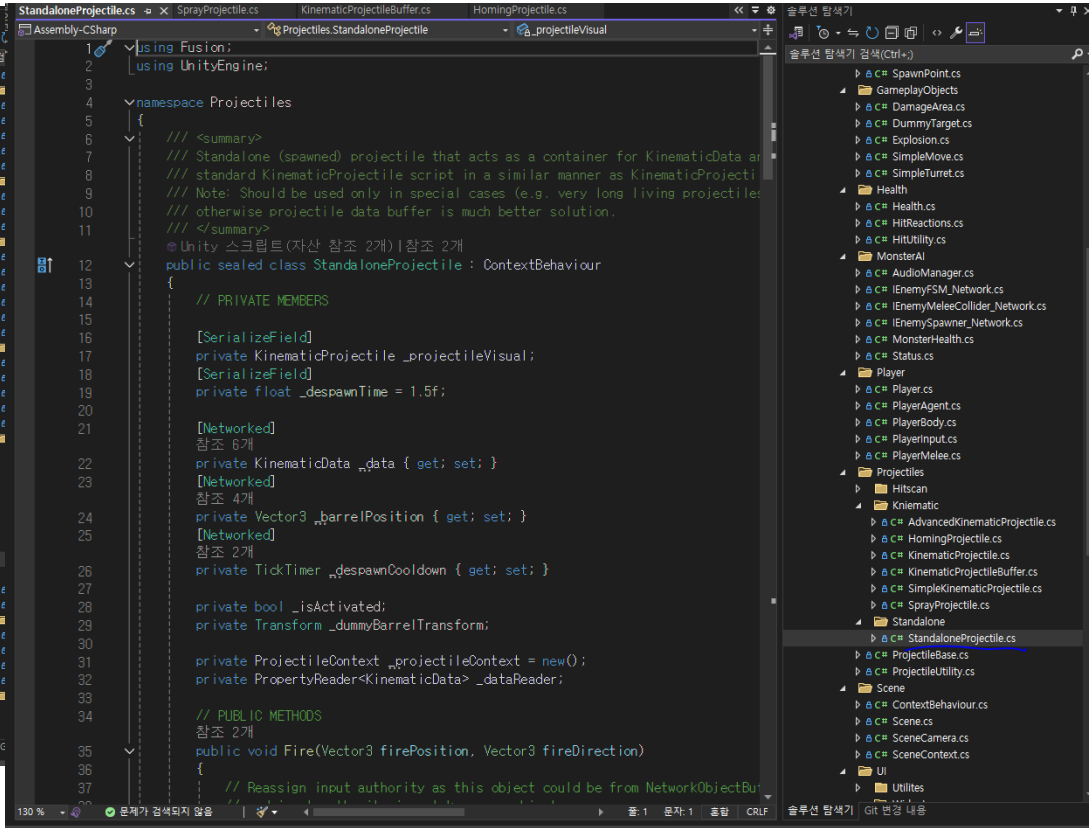
SimpleKinematicProjectile

키네마틱 상속체, 가장 기본적인 키네마틱 선형 발사체 구조



SprayProjectile

화염방사기, 레이저빔 등의 분사 형태의 CircleCast 기반 MultiHit Event 기반 키네마틱 상속체



StandaloneProjectile

플레이어의 입력에 따라서 그때그때 발사& 생성 하는 형태가 아닌, 플레이어의 최초입력 or AI시스템에 의해 생성된 StandAloneProjectile이 자체적으로 자식 ProjectileType을 생성하여 독립적으로 운영 하는 방식

Projectiles

주요한 기능&스크립트 요약

HitscanProjectile

발사직후에 발사된 궤적에 따라서 바로 피격처리하는 방식의 기본 발사체 방식

```
using Fusion;
using UnityEngine;

namespace Projectiles
{
    /// <summary>
    /// Base class for all hitscan projectile types.
    /// </summary>
    /// Unity 스크립트 | 참조 8개
    public abstract class HitscanProjectile : ProjectileBase, IBufferViewHitscanData
    {
        // PUBLIC MEMBERS

        참조 5개
        public bool IsFinished { get; protected set; }

        // PRIVATE MEMBERS

        [SerializeField]
        private float _damage = 10f;
        [SerializeField]
        private EHittType _hitType = EHittType.Projectile;
        [SerializeField]
        private LayerMask _hitMask;
        [SerializeField]
        protected float _maxDistance = 200f;

        // PUBLIC METHODS

        참조 12개
        public virtual HitscanData GetFireData(Vector3 firePosition, Vector3 fireDirection)
        {
            var data = new HitscanData()
            {
                FirePosition = firePosition,
                FireDirection = fireDirection,
            };

            if (ProjectileUtility.ProjectileCast(Context.Runner, Context.Owner, data, _maxDistance, _hitMask, _hitType))
            {
                HitUtility.ProcessHit(Context.Owner, data.FireDirection);
            }
        }
    }
}
```

```
using UnityEngine;

namespace Projectiles
{
    /// <summary>
    /// Hitscan projectile with dummy flying visual.
    /// </summary>
    /// Unity 스크립트 (자산 참조 1개) | 참조 0개
    public class FlyingHitscanProjectile : HitscanProjectile
    {
        // PRIVATE MEMBERS

        [SerializeField]
        private float _speed = 40f;

        private Vector3 _startPosition;
        private Vector3 _targetPosition;

        private float _time;
        private float _duration;

        // HitscanProjectile INTERFACE

        참조 4개
        public override void Activate(ref HitscanData data)
        {
            base.Activate(ref data);

            _startPosition = Context.BarrelTransforms[data.BarrelIndex].position;
            _targetPosition = (Vector3)data.ImpactPosition != Vector3.zero ? data.ImpactPosition : Context.Owner.transform.position;

            transform.position = _startPosition;
            transform.rotation = Quaternion.LookRotation(data.FireDirection);

            _duration = Vector3.Magnitude(_targetPosition - _startPosition) / _speed;
            _time = 0f;
        }

        참조 6개
        public override void Render(ref HitscanData data, ref HitscanData fromData, float alpha)
        {
            base.Render(ref data, ref fromData, alpha);
        }
    }
}
```

```
base.Activate(ref data);

var startPosition = Context.BarrelTransforms[data.BarrelIndex].position;

transform.position = startPosition;
transform.rotation = Quaternion.LookRotation(data.FireDirection);

SpawnImpactVisual(data.ImpactPosition, data.ImpactNormal);

if (_trail != null)
{
    _trail.SetPosition(0, startPosition);
    _trail.SetPosition(1, (Vector3)data.ImpactPosition != Vector3.zero ? data.ImpactPosition : Context.Owner.transform.position);
}

_time = 0f;

참조 6개
public override void Render(ref HitscanData data, ref HitscanData fromData, float alpha)
{
    base.Render(ref data, ref fromData, alpha);

    _time += Time.deltaTime;

    IsFinished = _time >= _visibleTime;

    if (_trail != null)
    {
        float progress = _time / _visibleTime;

        _trail.colorGradient = LerpGradient(_startTrailGradient, _fadeOutTrailGradient, progress);
        _trail.widthMultiplier = Mathf.Lerp(_startTrailWidthMultiplier, _fadeOutTrailWidthMultiplier, progress);
    }
}

// MONOBEHAVIOUR

Unity 메시지 | 참조 3개
protected override void Awake()
{
    _visibleTime = 1f;
}
```

FlyingHitscanProjectile

샷건등에서 사용되는, Hitscan 투사체의 상속 클래스

InstantHitscanProjectile

발사궤적, 흔적 등을 남기는 스나이퍼건에서 사용되는 Hitscan 투사체의 상속 클래스

Projectiles

주요한 기능&스크립트 요약

```
/// Projectile data buffer for Kinematic projectiles.
/// <summary>
/// Unity 스크립트 (자산 참조 16개) | 참조 2개
public class KinematicProjectileBuffer : NetworkDataBuffer<KinematicData, KinematicProjectile>
{
    // PRIVATE MEMBERS

    [SerializeField]
    private KinematicProjectile[] _projectilePrefabs;

    private ProjectileContext _context;

    // PUBLIC METHODS

    참조 2개
    public void AddProjectile(KinematicProjectile projectilePrefab, Vector3 firePos)
    {
        int prefabIndex = _projectilePrefabs.IndexOf(projectilePrefab);

        if (prefabIndex < 0)
        {
            Debug.LogError($"Projectile {projectilePrefab} not found. Add it in the prefabs list.");
            return;
        }

        // Temporarily assign correct context in case it will be needed in GetFireData
        projectilePrefab.Context = _context;
        var data = projectilePrefab.GetFireData(firePos, direction);
        projectilePrefab.Context = null;

        data.FireTick = Runner.Tick;
        data.PrefabIndex = (byte)prefabIndex;
        data.BarrelIndex = barrelIndex;

        if (InheritedVelocity != Vector3.zero)
        {
            data.Spray.InheritedVelocity = inheritedVelocity;
        }

        AddData(data);
    }

    참조 1개
}
```

```
namespace Projectiles
{
    참조 15개
    public struct HitscanData : INetworkStruct
    {
        public byte PrefabIndex;
        public byte BarrelIndex;
        public Vector3Compressed FirePosition;
        public Vector3Compressed FireDirection;
        public Vector3Compressed ImpactPosition;
        public Vector3Compressed ImpactNormal;
    }

    /// <summary>
    /// Projectile data buffer for hitscan projectiles.
    /// </summary>
    /// Unity 스크립트 (자산 참조 16개) | 참조 2개
    public class HitscanProjectileBuffer : NetworkDataBuffer<HitscanData, HitscanProjectile>
    {
        // PRIVATE MEMBERS

        [SerializeField]
        private HitscanProjectile[] _projectilePrefabs;

        private ProjectileContext _context;

        // PUBLIC METHODS

        참조 1개
        public void AddProjectile(HitscanProjectile projectilePrefab, Vector3 firePos)
        {
            int prefabIndex = _projectilePrefabs.IndexOf(projectilePrefab);

            if (prefabIndex < 0)
            {
                Debug.LogError($"Projectile {projectilePrefab} not found. Add it in the prefabs list.");
                return;
            }

            // Temporarily assign correct context in case it will be needed in GetFireData
            projectilePrefab.Context = _context;

            AddData(data);
        }
    }
}
```

```
/// <summary>
/// Unity 스크립트 (자산 참조 1개) | 참조 1개
public class NetworkObjectBuffer : NetworkBehaviour
{
    // CONSTANTS

    public const int CAPACITY = 8;

    // PRIVATE MEMBERS

    [SerializeField]
    private NetworkObject _prefab;
    [SerializeField, Range(1, CAPACITY)]
    private int _bufferSize = CAPACITY;

    [Networked]
    private NetworkArray<NetworkObject> _buffer { get; set; }
    [Networked]
    private int _bufferHead { get; set; }

    private NetworkObject[] _localBuffer = new NetworkObject[CAPACITY];

    // PUBLIC METHODS

    참조 1개
    public T Get<T>(Vector3 position, Quaternion rotation, PlayerRef inputAuthor)
    {
        var instance = Get(position, rotation, inputAuthor);
        return instance != null ? instance.GetComponent<T>() : null;
    }

    참조 1개
    public NetworkObject Get(Vector3 position, Quaternion rotation, PlayerRef inputAuthor)
    {
        var instance = _buffer[_bufferHead];

        if (instance == null)
            return null;
    }
}
```

KinematicProjectileBuffer

키네마틱 발사체등의 생성&삭제
&렌더링&위치&피격데이터 처리
등을 직접적으로 관리하는 클래스

NetworkObjectBuffer

StandAloneProjectile의 생성 및
파기,위치&렌더링 처리 등의 관리

HitscanProjectileBuffer

Hitscan 발사체등의 생성&삭제
&렌더링&위치&피격데이터 처리
등을 직접적으로 관리하는 클래스

NetworkDataBuffer

키네마틱, Hitscan 버퍼클래스의 상위
클래스로, 모든 프로젝트일의 생성&
삭제&렌더링&위치&피격데이터 관리
최상위 클래스

```
/// <summary>
/// Unity 스크립트 (자산 참조 3개)
public abstract class NetworkDataBuffer<TData, TView> : ContextBehaviour
where TData : INetworkStruct
where TView : IBufferView<TData>
{
    // PROTECTED MEMBERS

    [Networked]
    참조 4개
    protected int _dataCount { get; set; }

    // PRIVATE MEMBERS

    private Dictionary<int, TView> _views = new(256);
    private List<int> _finishedViews = new(128);

    private int _viewCount;

    private ArrayReader<TData> _dataBufferReader;
    private PropertyReader<int> _dataCountReader;

    // PUBLIC METHODS

    참조 2개
    public void AddData(TData data)
    {
        int dataIndex = _dataCount % DataBuffer.Length;
        DataBuffer.Set(dataIndex, data);

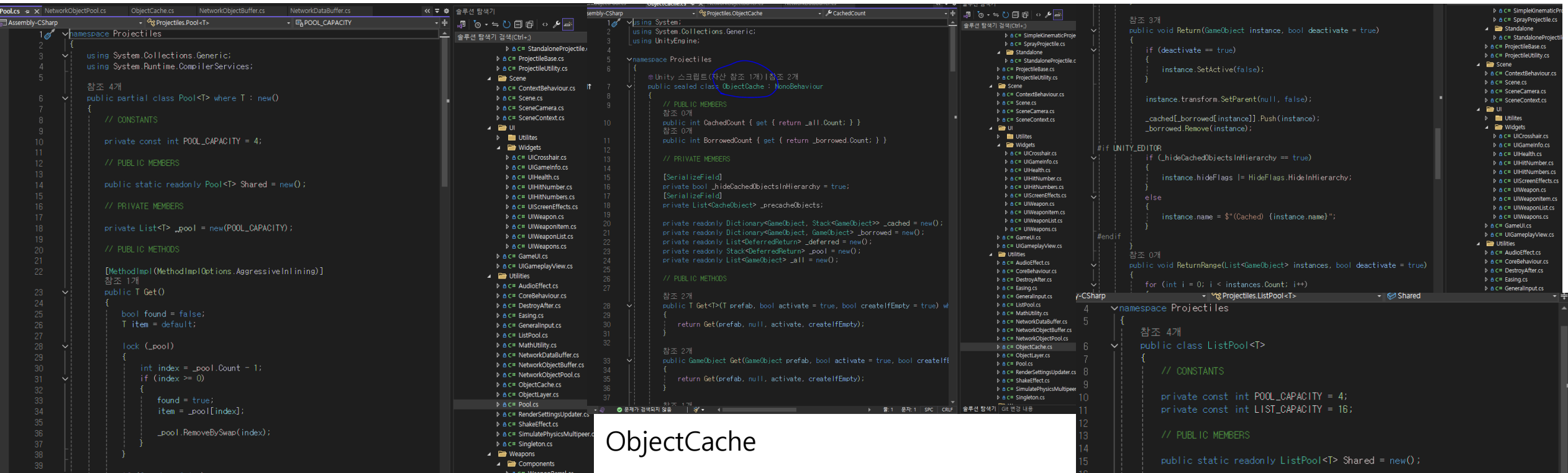
        _dataCount++;
    }

    // NetworkBehaviour INTERFACE

    참조 4개
    public override void Spawned()
    {
        _viewCount = _dataCount;

        _dataBufferReader = GetArrayReader<TData>(nameof(DataBuffer));
        _dataCountReader = GetPropertyReader<int>(nameof(DataCount));
    }
}
```

주요한 기능&스크립트 요약



ObjectCache

모든 종류의 프로젝타일등의 오브젝트 생성&파기등의 데이터를 주로 관리하는 클래스

ListPool

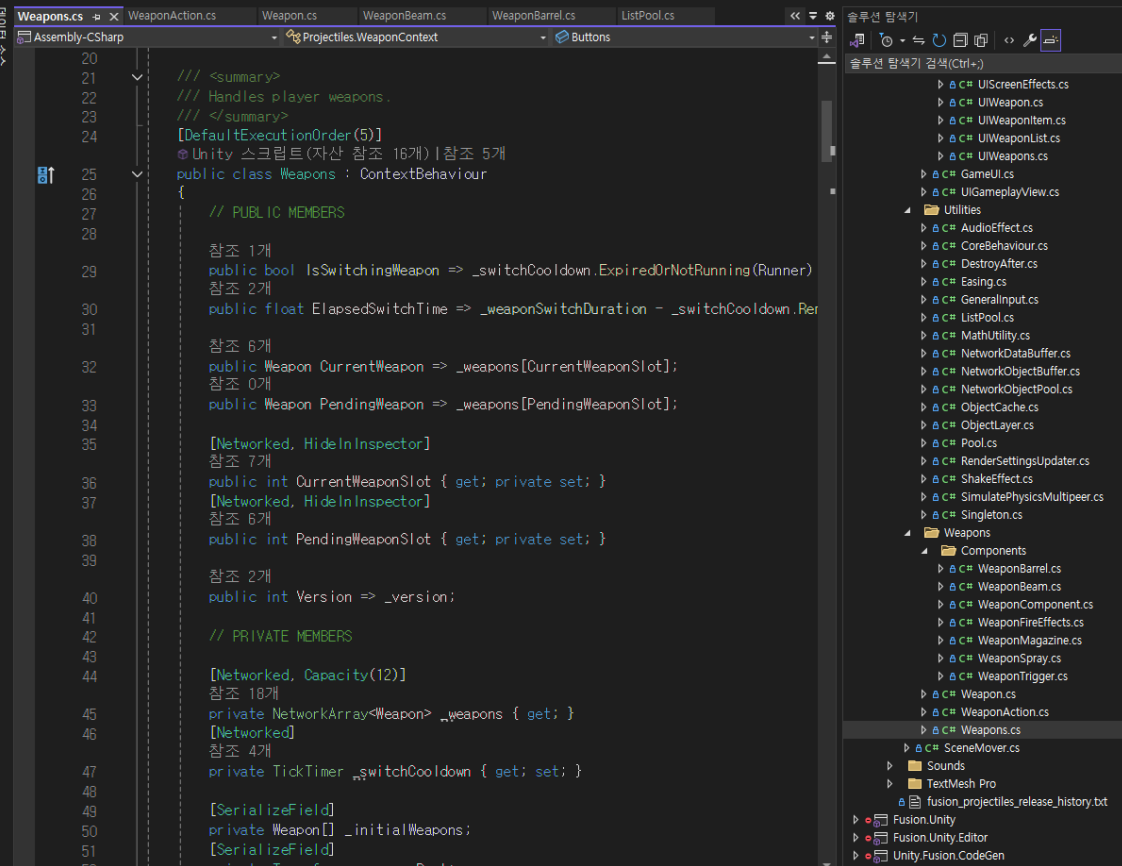
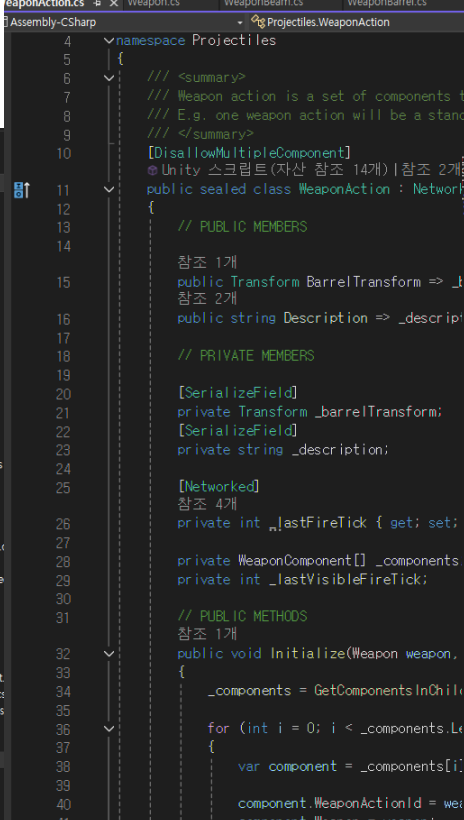
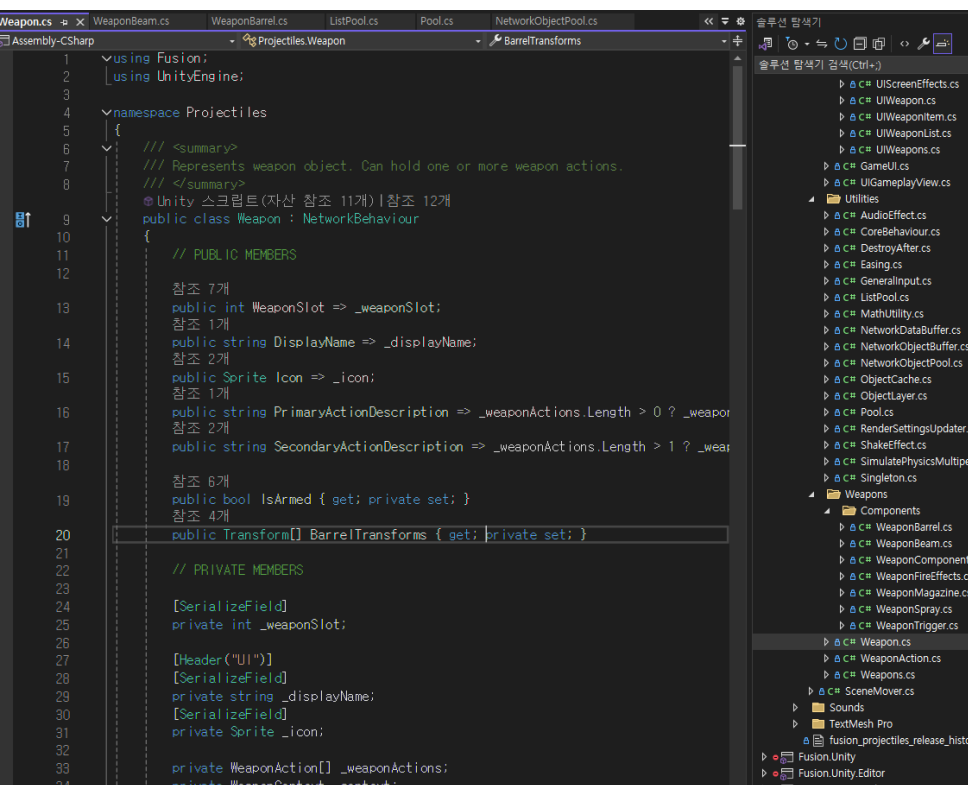
모든 리스트형태의 추상데이터의 생성&파기등의 데이터 관리 클래스

Pool

NetworkDataBuffer, ShakeEffect 클래스 등에서 사용되는 오브젝트 풀링제공 클래스

Weapons

주요한 기능&스크립트 요약



Weapon

유저의 현재 활성화되어있는 무기 타입에 대한 해제/장착 발사액션 컨트롤 관련 처리

WeaponAction

하위자식 WeaponComponent 관리 및 컴포넌트 FireRender, TryFire 등 관리

Weapons

유저의 개별적 웨폰리스트 관리, 웨폰 스왑 (Prev, Next), 무기 장착 관련 액션, 애니메이션 처리, 공격 처리, 웨폰 Add, Remove 관련 액션 수행

Weapons

주요한 기능&스크립트 요약

```
namespace Projectiles
{
    /// <summary>
    /// A base for all weapon components. Holds references needed for proper weapon.
    /// </summary>
    /// <summary>
    /// Unity 스크립트 참조 8개
    public abstract class WeaponComponent : ContextBehaviour
    {
        // PUBLIC MEMBERS

        참조 4개
        public byte WeaponActionId { get; set; }
        참조 8개
        public Weapon Weapon { get; set; }
        참조 8개
        public Transform BarrelTransform { get; set; }
        참조 8개
        public NetworkContext NetworkContext { get; set; }

        참조 2개
        public NetworkButtons Buttons => NetworkContext.Buttons;
        참조 1개
        public NetworkButtons PressedButtons => NetworkContext.PressedButtons;
        참조 12개
        public Transform FireTransform => NetworkContext.FireTransform;

        참조 4개
        public virtual bool IsBusy => false;

        // PUBLIC METHODS

        참조 3개
        public virtual bool CanFire() => true;
        참조 6개
        public virtual void Fire() {}
        참조 3개
        public virtual void FireRender() {}
    }
}
```

WeaponTrigger

무기별 공격, 발사 쿨타임 조
율 및 발사가능여부 등 조율

```
private int _fireTicks;

// WeaponComponent INTERFACE

참조 3개
public override bool IsBusy => _fireCooldown.ExpiredOrNotRunning(Runner) == false;

참조 2개
public override bool CanFire()
{
    if (Weapon.IsBusy() == true)
        return false;

    if (_fireCooldown.ExpiredOrNotRunning(Runner) == false)
        return false;

    return _fireOnKeyDownOnly == true ? PressedButtons.IsSet(_fireButton) : false;
}

참조 2개
public override void Fire()
{
    _fireCooldown = TickTimer.CreateFromTicks(Runner, _fireTicks);
}

// NetworkBehaviour INTERFACE

참조 0개
public override void Spawned()
{
    base.Spawned();

    float fireTime = 60f / _cadence;
    _fireTicks = (int)System.Math.Ceiling(fireTime / (double)Runner.DeltaTime);
}
```

```
using UnityEngine;

namespace Projectiles
{
    /// <summary>
    /// Weapon component that is responsible for firing projectiles.
    /// </summary>
    /// <summary>
    /// Unity 스크립트(자산 참조 12개) 참조 0개
    public class WeaponBarrel : WeaponComponent
    {
        // PRIVATE MEMBERS

        [SerializeField]
        private ProjectileBase _projectilePrefab;
        [SerializeField]
        private StandaloneProjectile _standaloneProjectilePrefab;
        [SerializeField]
        private NetworkObjectBuffer _standaloneProjectileBuffer;

        [SerializeField]
        private int _projectilesPerShot = 1;
        [SerializeField]
        private float _dispersion = 1f;
        [SerializeField, Tooltip("When firing ballistic projectiles (grenades) it will add a random pitch to the projectile's trajectory.")]
        private float _additionalFirePitch = 0f;

        // WeaponComponent INTERFACE

        참조 2개
        public override void Fire()
        {
            if (_dispersion > 0f)
            {
                Random.InitState(Runner.Tick * unchecked((int)Object.Id.Raw));

                var fireRotation = FireTransform.rotation * Quaternion.Euler(-_additionalFirePitch, 0, 0);

                for (int i = 0; i < _projectilesPerShot; i++)
                {
                    var projectileRotation = fireRotation;

                    if (_dispersion > 0f)
                    {
                        float randomPitch = Random.Range(-_dispersion, _dispersion);
                        projectileRotation *= Quaternion.Euler(0, 0, randomPitch);
                    }
                }
            }
        }
    }
}
```

WeaponBarrel

발사체 직접 발사 및 생성(히트스캔타입, 키네마틱파입, standalone타입)
Projectiles 발사 컨트롤

WeaponComponent

무기클래스 하위의 무기컴포넌트 관련 기본적인 액션 관련 추상클래스

주요한 기능&스크립트 요약

```
using UnityEngine;
using Projectiles;

namespace Projectiles
{
    /// <summary>
    /// Weapon component that is responsible for all fire effects
    /// - camera shake, weapon knockback, fire sound, fire particles.
    /// </summary>
    /// Unity 스크립트 (자신 참조 12개) | 참조 0개
    public class WeaponFireEffects : WeaponComponent
    {
        // PRIVATE MEMBERS

        [Header("Muzzle")]
        [SerializeField]
        private GameObject _fireParticle;

        [Header("Sound")]
        [SerializeField]
        private Transform _fireAudioEffectsRoot;
        [SerializeField]
        private AudioSource _fireSound;

        [Header("Camera")]
        [SerializeField]
        private ShakeSetup _cameraShakePosition;
        [SerializeField]
        private ShakeSetup _cameraShakeRotation;

        [Header("Kickback")]
        [SerializeField]
        private Transform _kickbackTransform;
        [SerializeField]
        private Kickback _positionKickback = new(0.06f, 60f, 20f);
        [SerializeField]
    }
}
```

```
using System;
using UnityEngine;

namespace Projectiles
{
    /// <summary>
    /// Handles weapon action ammo.
    /// </summary>
    /// Unity 스크립트 (자신 참조 14개) | 참조 0개
    public class WeaponMagazine : WeaponComponent
    {
        // PUBLIC MEMBERS

        참조 1개
        public bool IsReloading => _isReloading;
        참조 0개
        public float ReloadProgress => IsReloading == true ? _reloadCooldown.Remainder : 0;
        참조 0개
        public bool HasMagazine => _hasMagazine;
        참조 0개
        public bool HasUnlimitedAmmo => _hasUnlimitedAmmo;
        참조 0개
        public int MagazineAmmo => _magazineAmmo;
        참조 0개
        public int WeaponAmmo => _weaponAmmo;

        // PRIVATE MEMBERS

        [SerializeField]
        private int _initialAmmo = 150;
        [SerializeField]
        private bool _hasMagazine = true;
        [SerializeField]
        private int _maxMagazineAmmo = 30;
        [SerializeField]
        private int _maxWeaponAmmo = 120;
        [SerializeField]
        private bool _hasUnlimitedAmmo;
        [SerializeField]
        private float _reloadTime = 2f;

        [Networked]
    }
}
```

WeaponFireEffects

무기 공격시에 카메라 흔들림, 파티클 발생등의 시각적 효과 관련 처리

WeaponMagazine

무기 탄창수 관련 데이터 처리

```
using UnityEngine;

namespace Projectiles
{
    /// <summary>
    /// Weapon component that handles firing of spray projectiles and showing spray.
    /// </summary>
    /// Unity 스크립트 (자신 참조 1개) | 참조 0개
    public class WeaponSpray : WeaponComponent
    {
        // PRIVATE MEMBERS

        [SerializeField]
        private SprayProjectile _projectilePrefab;
        [SerializeField]
        private float _dispersion = 1f;
        [SerializeField, Range(0f, 1f)]
        private float _moveVelocityMultiplier = 0.3f;

        [Header("Effects")]
        [SerializeField]
        private ParticleSystem _fireParticle;
        [SerializeField]
        private ParticleSystem _sparkParticle;
        [SerializeField]
        private AudioSource _fireSound;
        [SerializeField]
        private ShakeSetup _cameraShakePosition;
        [SerializeField]
        private ShakeSetup _cameraShakeRotation;

        private float _fireEffectsCooldown;
        private float _maxVolume;

        // WeaponComponent INTERFACE
        참조 2개
        public override void Fire()
        {
            var projectileDirection = FireTransform.forward;
        }
    }
}
```

```
using System;
using System.Web;
using Fusion;
using Fusion.LagCompensation;
using UnityEngine;

namespace Projectiles
{
    /// <summary>
    /// Weapon component that fires a laser beam that damages targets on hit.
    /// </summary>
    /// Unity 스크립트 (자신 참조 1개) | 참조 0개
    public class WeaponBeam : WeaponComponent
    {
        // PRIVATE MEMBERS

        [SerializeField]
        private float _damage = 10f;
        [SerializeField]
        private RaycastHit _hitType = RaycastHit.Projectile;
        [SerializeField]
        private LayerMask _hitMask;
        [SerializeField]
        private float _maxDistance = 50f;
        [SerializeField]
        private float _beamRadius = 0.2f;
        [SerializeField, Tooltip("Number of raycast rays fired. First is always in center")]
        private int _raycastAmount = 5;
        [SerializeField]
        private WeaponTrigger _weaponTrigger;

        [Header("Beam Visuals")]
        [SerializeField]
        private GameObject _beamStart;
        [SerializeField]
        private GameObject _beamEnd;
        [SerializeField]
        private LineRenderer _beam;
        [SerializeField]
        private float _beamEndOffset = 0.5f;
        [SerializeField]
    }
}
```

Weapons

WeaponSpray

분사형 무기(화염방사기형) 관련 클래스

WeaponBeam

빔형 무기(레이저빔) 관련 클래스

주요한 기능&스크립트 요약(카테고리)

- Extensions
 - AudioEffectExtensions.cs
 - AudioSourceExtensions.cs
 - ComponentExtensions.cs
 - GameObjectExtensions.cs
 - ListExtensions.cs
 - NumberExtensions.cs
 - SceneExtensions.cs
 - StringExtensions.cs
 - UIExtensions.cs
- Game
 - GameManager.cs
 - Gameplay.cs
 - SpawnPoint.cs

오디오관련,컴포넌트,게임오브젝트,리스트,넘버,썬,문자열,UI관련 익스텐션 및 게임매니저&게임플레이,스폰 포인트 정의

- GameplayObjects
 - DamageArea.cs
 - DummyTarget.cs
 - Explosion.cs
 - SimpleMove.cs
 - SimpleTurret.cs

게임플레이 데미지영역,Dummy몬스터타깃,폭발이펙트형구체 데미지 네트워크처리,더미몬스터 주기이동,심플터렛 등

- Health
 - Health.cs
 - HitReactions.cs
 - HitUtility.cs
- MonsterAI
 - AudioManager.cs
 - IEnemyFSM_Network.cs
 - IEnemyMeleeCollider_Network.cs
 - IEnemySpawner_Network.cs
 - MonsterHealth.cs
 - Status.cs

캐릭터,몬스터 공용 체력처리,더미 타깃 히리액션,히데미지 처리관리자,적군AI시스템 및 적 Health에 따른 시각애니메이션 동기화

- Player
 - Player.cs
 - PlayerAgent.cs
 - PlayerBody.cs
 - PlayerInput.cs
 - PlayerMelee.cs

플레이어 세션고유 데이터,플레이어 주요조작,플레이어 외형,플레이어 입력,근접공격형 캐릭터 무기 구현

- Projectiles
 - Hitscan
 - FlyingHitscanProjectile.cs
 - HitscanProjectile.cs
 - HitscanProjectileBuffer.cs
 - InstantHitscanProjectile.cs
 - Kinematic
 - AdvancedKinematicProjectile.cs
 - HomingProjectile.cs
 - KinematicProjectile.cs
 - KinematicProjectileBuffer.cs
 - SimpleKinematicProjectile.cs
 - SprayProjectile.cs
 - Standalone
 - StandaloneProjectile.cs
 - ProjectileBase.cs
 - ProjectileUtility.cs

모든 발사체의 베이스클래스,발사체 피격궤도계산 유틸리티,키네마틱,향상키네마틱(튕김,반사,바운스), 홈키네마틱(유도미사일),키네마틱 자원제공버퍼, 기본형키네마틱, 방사형키네마틱, 히스캔발사체,히스캔 발사체 버퍼제공자,날르는 히스캔 발사체,궤적표현 히스캔 발사체,독립적 행동발사체

주요한 기능&스크립트 요약(카테고리)

```

└─ Utilities
  ├── AudioEffect.cs
  ├── CoreBehaviour.cs
  ├── DestroyAfter.cs
  ├── Easing.cs
  ├── GeneralInput.cs
  ├── ListPool.cs
  ├── MathUtility.cs
  ├── NetworkDataBuffer.cs
  ├── NetworkObjectBuffer.cs
  ├── NetworkObjectPool.cs
  ├── ObjectCache.cs
  ├── ObjectLayer.cs
  ├── Pool.cs
  ├── RenderSettingsUpdater.cs
  ├── ShakeEffect.cs
  ├── SimulatePhysicsMultipeer.cs
  └── Singleton.cs

```

오디오효과,핵심행동자,이징,일반
입력,리스트풀,네트워크 데이터버
퍼(히트스캔&키네마틱버퍼 상위 관
리자:프로젝타일 모든 행동관리
자),네트워크 오브젝트버퍼(스탠
드언론발사주체) 버퍼&행동제공
자,오브젝트 캐시관리자,풀,렌더셋
팅,카메라흔들림 효과,네트워크 물
리 시뮬레이션,싱글톤

```

└─ Scene
  ├── ContextBehaviour.cs
  ├── Scene.cs
  ├── SceneCamera.cs
  └── SceneContext.cs
└─ UI
  ├── Utilities
  │   ├── UIBehaviour.cs
  │   ├── UIFader.cs
  │   ├── UIList.cs
  │   └── UIListItem.cs
  ├── Widgets
  │   ├── UICrosshair.cs
  │   ├── UIGameInfo.cs
  │   ├── UIHealth.cs
  │   ├── UIHitNumber.cs
  │   ├── UIHitNumbers.cs
  │   ├── UIScreenEffects.cs
  │   ├── UIWeapon.cs
  │   ├── UIWeaponItem.cs
  │   ├── UIWeaponList.cs
  │   └── UIWeapons.cs
  ├── GameUI.cs
  └── UIGameplayView.cs

```

씬 컨텍스트,씬카메라,UI관련 유니
티리티,UI 위젯관련 컴포넌트 클래
스,UI 관련

```

└─ Weapons
  └─ Components
    ├── WeaponBarrel.cs
    ├── WeaponBeam.cs
    ├── WeaponComponent.cs
    ├── WeaponFireEffects.cs
    ├── WeaponMagazine.cs
    ├── WeaponSpray.cs
    └── WeaponTrigger.cs
  ├── Weapon.cs
  ├── WeaponAction.cs
  ├── Weapons.cs
  └── SceneMover.cs

```

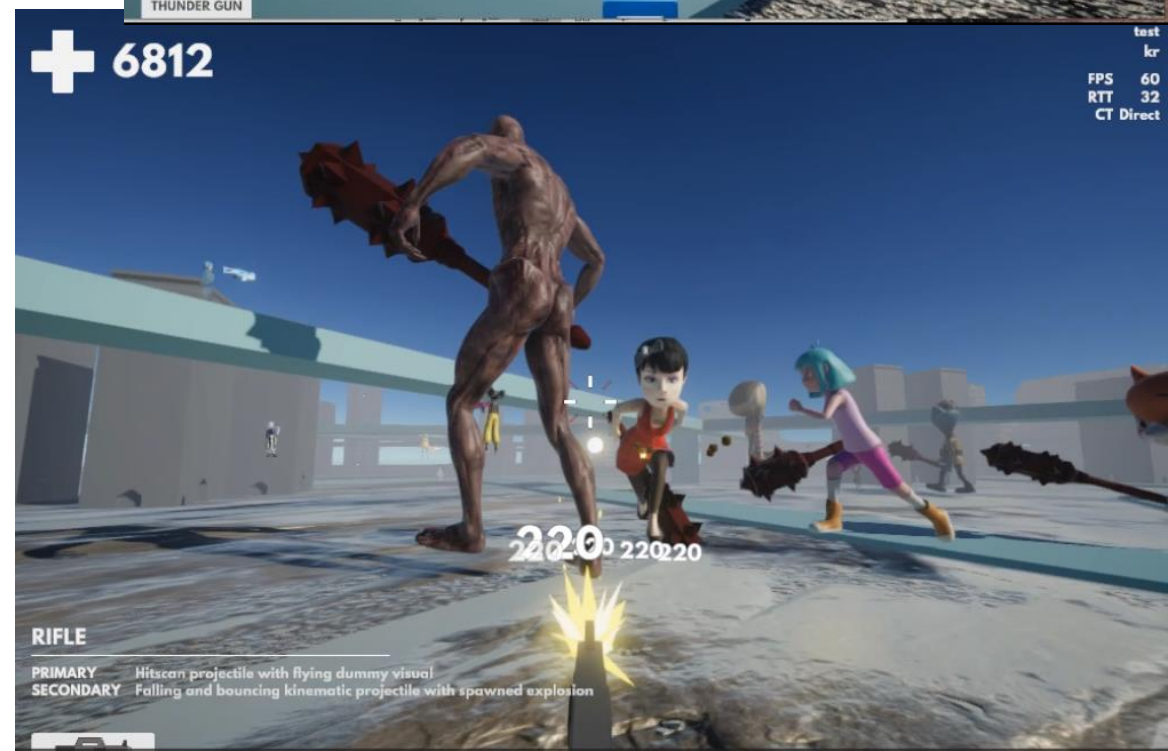
무기 발사체 공격 컨트롤,무
기빔,무기구성요소,무기발사
효과,무기 탄창,방사형무기,
무기 공격 쿨타임,무기리스
트,무기액션 등 무기관련



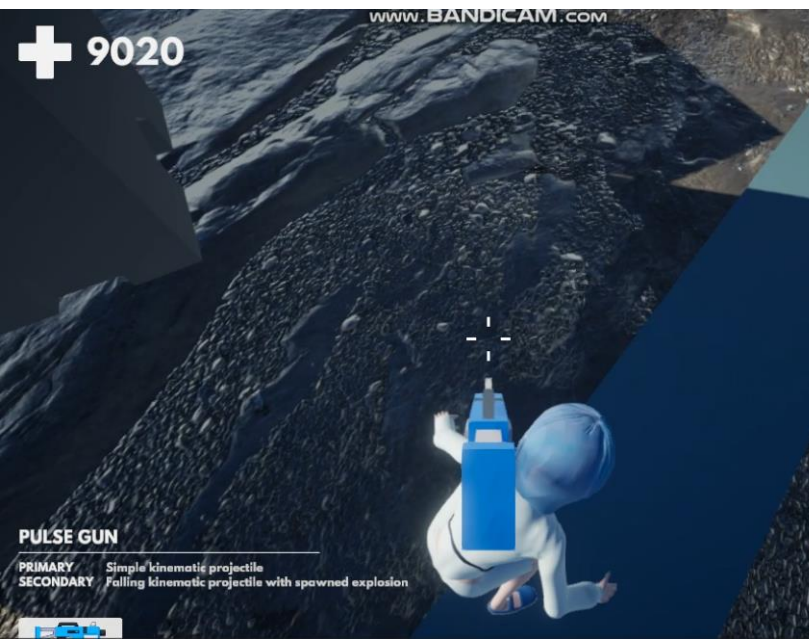
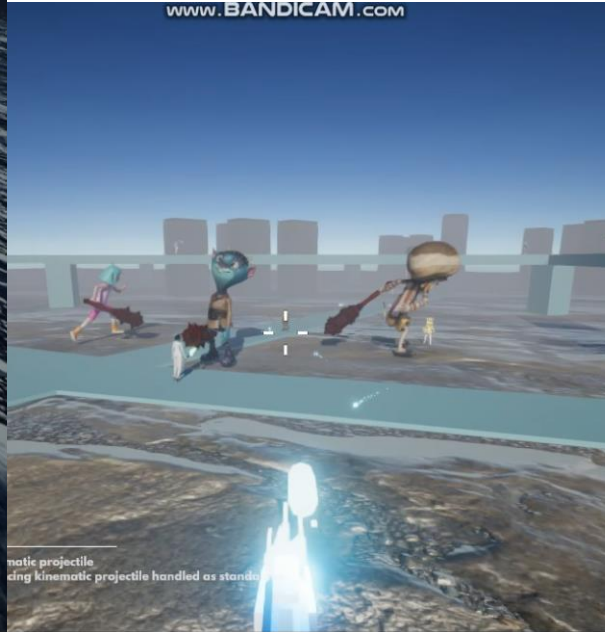
게임플레이





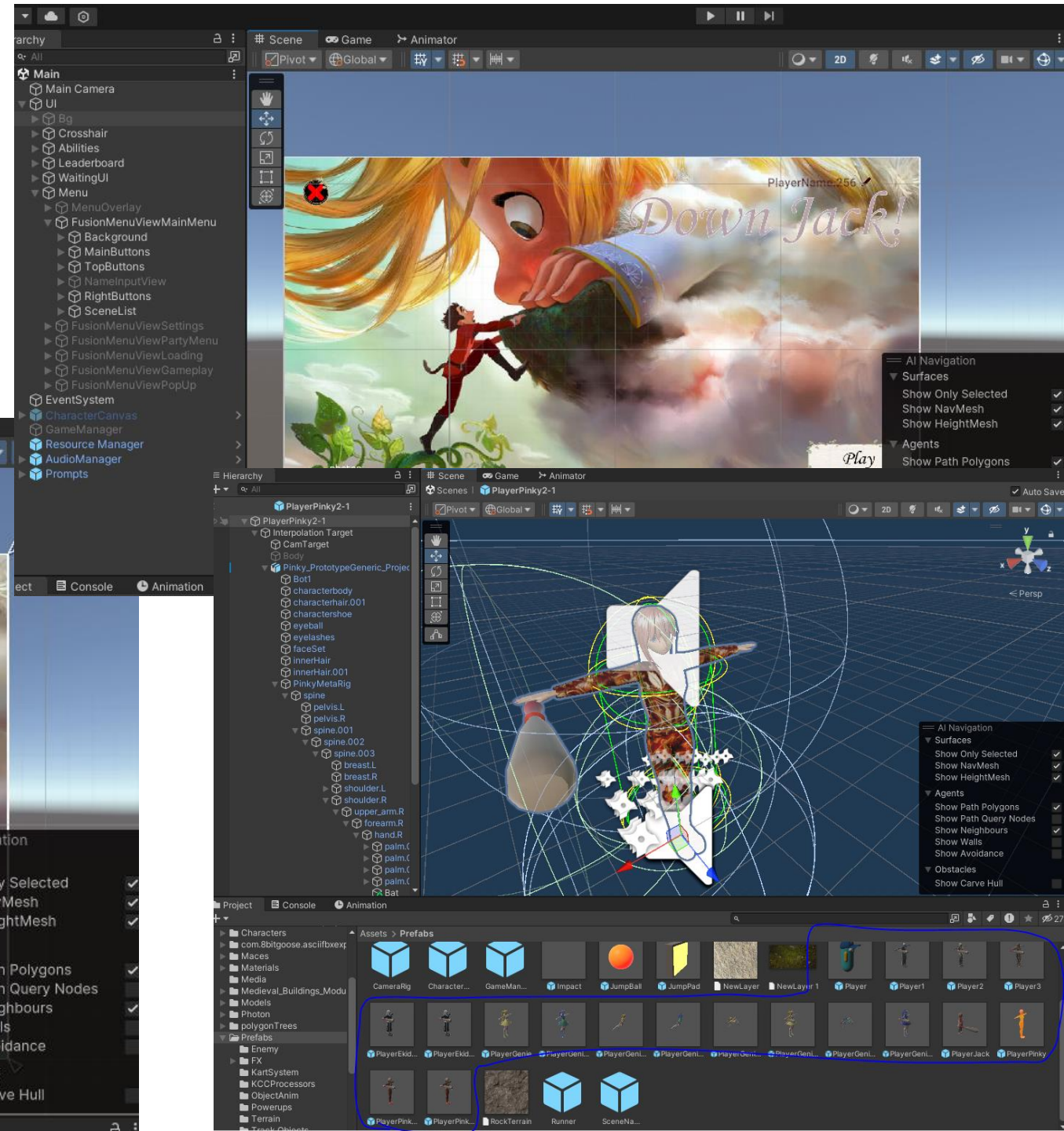
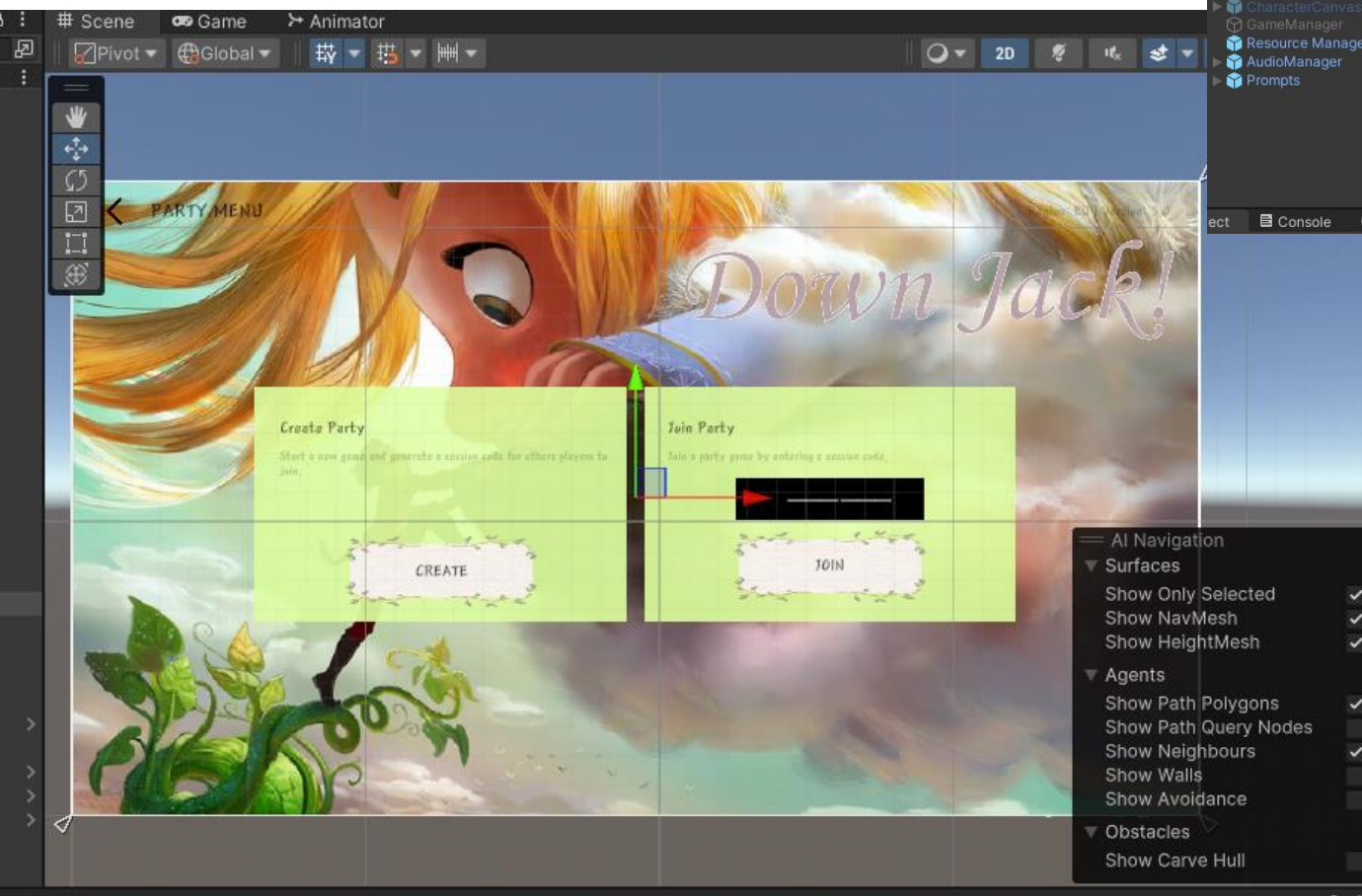




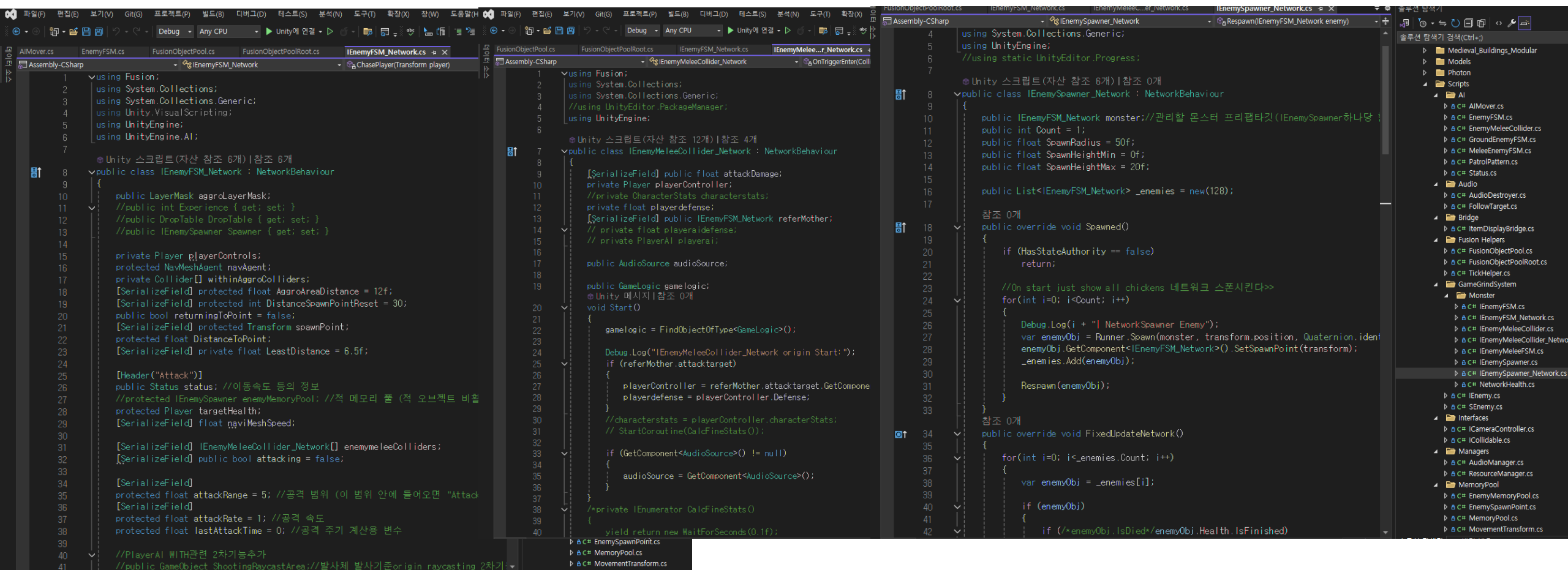


프로젝트3. 멀티플레이어 Titan레이드(PVP,PVE)

개요: 플레이어별 랜덤한 서로 다른 캐릭터로 접속 가능하며,
캐릭터마다 다른 근접공격 스타일의 무기로 서로 전투할 수
있고, 맵 상에 존재하는 Network Monster AI와의 전투도
가능한 PVP,PVE 콘텐츠
작업 기간: 2024.12.19~2024.12.30 (11일)
작업 분담: 4인개발(개발1,아트2,기획1)
본인 분담: 개발총괄,레벨&배경디자인
기여&소통: 아트파트에게 개발에 필요한 리소스 요청 및 커뮤니케이션,
기획파트와 기술&일정적으로 가능한 콘텐츠 기능구현 조율



주요한 기능&스크립트 요약



IEnemyFSM_Network

몬스터 AI 행동 결정 시스템
대기,추적,공격 관련 행동 의사결정
및 행동형태 네트워크 동기화

IEnemyMeleeCollider_Network

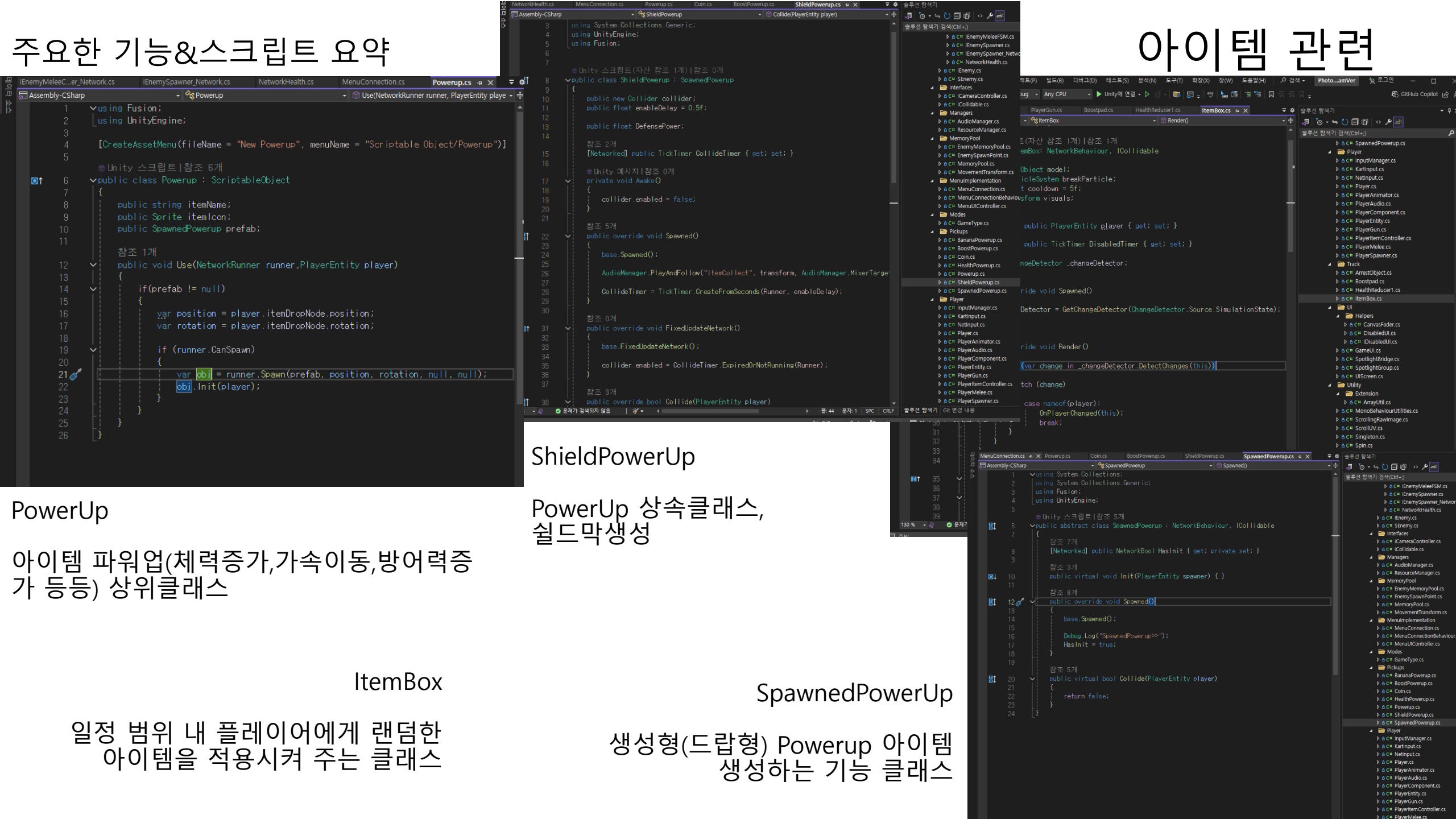
몬스터 AI 근접 공격 관련 피격처리

IEnemySpawner_Network

몬스터 AI 주기적 생성&삭제 관련
관리 클래스

아이템 관련

주요한 기능&스크립트 요약



PowerUp

아이템 파워업(체력증가,가속이동,방어력증가 등등) 상위클래스

ItemBox

일정 범위 내 플레이어에게 랜덤한
아이템을 적용시켜 주는 클래스

ShieldPowerUp

PowerUp 상속클래스,
실드막생성

SpawnedPowerUp

생성형(드랍형) Powerup 아이템
생성하는 기능 클래스

플레이어

주요한 기능&스크립트 요약

```
1 using Fusion;
2 using Fusion.Addons.KDC;
3 using System;
4 using UnityEngine;
5 using UnityEngine.Events;
6 using UnityEngine.InputSystem;
7 using UnityEngine.Windows;
8
9 @Unity 스크립트 (자산 참조 18개) | 참조 2개
10 public class Player : MonoBehaviour
11 {
12     [SerializeField] private KDC kdc;
13     [SerializeField] private KDCProcessor glideProcessor;
14     [SerializeField] private Transform camTarget;
15     [SerializeField] private AudioSource source;
16     [SerializeField] private float maxPitch = 85f;
17     [SerializeField] private float lookSensitivity = 0.15f;
18     [SerializeField] private Vector3 jumpImpulse = new(0f, 10f, 0f);
19     [SerializeField] private float doubleJumpMultiplier = 0.75f;
20     [SerializeField] private float grappleCD = 2f;
21     [SerializeField] private float doubleJumpCD = 1.2f;
22     [SerializeField] private float grappleStrength = 12f;
23     [SerializeField] private float maxGlideTime = 2f;
24     참조 1개
25     [field: SerializeField] public float AbilityRange { get; private set; } = 25f;
26
27     참조 1개
28     public float GrappleCDFactor => (GrappleCD.RemainingTime(Runner) ?? 0f) / grappleCD;
29     참조 1개
30     public float GlideCDFactor => (GlideCD.RemainingTime(Runner) ?? 0f) / glideCD;
31     참조 1개
32     public float DoubleJumpCDFactor => (DoubleJumpCD.RemainingTime(Runner) ?? 0f) / doubleJumpCD;
33     참조 2개
34     public double Score => Math.Round(transform.position.y, 1);
35     public bool IsReady; // Server is the only one who cares about this
36     public bool IsArrive;
37
38     private bool CanGlide => !kcc.Data.IsGrounded && GlideCharge > 0f;
```

Player

플레이어의 상태(스핀아웃, 방어력, 속박여부, 가속속도상태) 및 조작을 전반적으로 담당하는 클래스

PlayerComponent

PlayerEntity의 상위 클래스

```
1 using Fusion;
2 using UnityEngine;
3 using UnityEngine.InputSystem.XR;
4
5 @Unity 스크립트 (자산 참조 20개) | 참조 2개
6 public class PlayerAnimator : MonoBehaviour
7 {
8     //public ParticleSystem[] backfireEmitters;
9     public ParticleSystem[] boostEmitters;
10    //public ParticleSystem[] driftEmitters;
11    //public ParticleSystem[] driftTierEmitters;
12    //public ParticleSystem[] tireSmokeEmitters;
13
14    [SerializeField] private NetworkMecanimAnimator _nma;
15    [SerializeField] private Animator _animator;
16
17    public Player controller;
18
19    public GameObject DefensePowerup;
20    public GameObject ArrestBox;
21
22    @Unity 메시지 | 참조 0개
23    private void Start()
24    {
25        //controller = player;
26    }
27
28    @Unity 메시지 | 참조 0개
29    private void Update()
30    {
31        //controller = player;
32    }
33
34    참조 0개
35    public void AllowDrive()
36    {
37        controller.RefreshAppliedSpeed();
38    }
39
40    참조 0개
41    public override void Spawned()
```

PlayerAnimator

플레이어의 현재 데이터 상태에 따른 시각적 애니메이션 관련 처리 클래스

PlayerAudio

플레이어의 현재 데이터 상태에 따른 소리 VFX 관련 처리 클래스

```
1 using Fusion;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5
6 @Unity 스크립트 (자산 참조 16개) | 참조 2개
7 public class PlayerAudio : MonoBehaviour
8 {
9     public AudioSource StartSound;
10    public AudioSource IdleSound;
11    public AudioSource RunningSound;
12    public AudioSource ReverseSound;
13    public AudioSource Drift;
14    public AudioSource Boost;
15    //public AudioSource Offroad;
16    //public AudioSource Crash;
17    public AudioSource Horn;
18    [Range(0.1f, 1.0f)] public float RunningSoundMaxVolume = 1.0f;
19    [Range(0.1f, 2.0f)] public float RunningSoundMaxPitch = 1.0f;
20    [Range(0.1f, 1.0f)] public float ReverseSoundMaxVolume = 0.5f;
21    [Range(0.1f, 2.0f)] public float ReverseSoundMaxPitch = 0.8f;
22    [Range(0.1f, 1.0f)] public float IdleSoundMaxVolume = 0.8f;
23
24    [Range(0.1f, 1.0f)] public float DriftMaxVolume = 0.5f;
25
26    public Player controller;
27
28    @Unity 메시지 | 참조 0개
29    private void Start()
30    {
31        //controller = player;
32    }
33
34    @Unity 메시지 | 참조 0개
35    private void Update()
36    {
37        //controller = player;
38    }
39
40    참조 0개
```

```
1 using Fusion;
2 using UnityEngine;
3
4 @Unity 스크립트 | 참조 8개
5 public class PlayerComponent : NetworkBehaviour
6 {
7     참조 1개
8     public Player player { get; private set; }
9     참조 8개
10    public PlayerEntity playerEntity { get; private set; }
11    @Unity 메시지 | 참조 0개
12    private void Awake()
13    {
14        Debug.Log("PlayerComponent Awake");
15        player = GetComponentInParent<Player>();
16    }
17
18    참조 1개
19    public virtual void Init(PlayerEntity entity)
20    {
21        Debug.Log("PlayerComponent entity" + entity.name);
22        playerEntity = entity;
23    }
24
25    /// <summary>
26    /// Called on the tick that the race has started. This method is tick-aligned.
27    /// </summary>
28    public virtual void OnRaceStart() {}
29
30    /// <summary>
31    /// Called when this kart has crossed the finish line. This method is tick-aligned.
32    /// </summary>
33    public virtual void OnLapCompleted(int lap, bool isFinish) {}
34
35    /// <summary>
36    /// Called when an item has been picked up. This method is tick-aligned.
37    /// </summary>
38    참조 3개
39    public virtual void OnEquipItem(Powerup powerup, float timeUntilCanUse) {}
40
41    참조 3개
```

플레이어

주요한 기능&스크립트 요약

```
using Fusion;
//using Fusion.Addons.Physics;
using UnityEngine;
using UnityEngine.UIElements;

// Unity 스크립트 (자산 참조 16개) | 참조 23개
public class PlayerEntity : PlayerComponent
{
    //public static event Action<PlayerEntity> OnKartSpawned;
    //public static event Action<PlayerEntity> OnKartDespawned;

    public event Action<int> OnHeldItemChanged;
    public event Action<int> OnCoinCountChanged;

    참조 1개
    public PlayerAnimator Animator { get; private set; }
    //public KartCamera Camera { get; private set; }
    참조 21개
    public PlayerController { get; private set; }
    참조 0개
    public InputManager Input { get; private set; }
    // public KartLapController LapController { get; private set; }
    참조 2개
    public PlayerAudio Audio { get; private set; }
    참조 2개
    public GameUI Hud { get; private set; }
    참조 2개
    public PlayerItemController Items { get; private set; }
    //public NetworkRigidbody3D Rigidbody { get; private set; }

    참조 6개
    public Powerup HeldItem =>
        HeldItemIndex == -1
        ? null
        : ResourceManager.Instance.powerups[HeldItemIndex];

    [Networked]
    참조 8개
    public int HeldItemIndex { get; set; } = -1;

    [Networked]
    참조 3개
}
```

PlayerEntity

플레이어의 관련된 상태(애니메이션, 컨트롤러, 오디오, 아이템)등의 컴포넌트 상태관리 클래스

```
using Fusion;
using UnityEngine;

참조 34개
public enum InputButton
{
    Jump,
    Grapple,
    Glide,
    W,
    S,
    A,
    D,
    F,
    Ctrl
}

참조 13개
public struct NetInput : INetworkInput
{
    public NetworkButtons Buttons;
    public Vector2 Direction;
    public Vector2 LookDelta;
}
```

NetInput

NetInput 네트워크 인풋 Structure

```
InputSystem;
//참조 1개 | 참조 3개
Manager : SimulationBehaviour, IBeforeUpdate, INetworkRunnerCallBa

LocalPlayer;

! AccumulatedMouseDelta => mouseDeltaAccumulator.AccumulatedValue;

it accumulatedInput;
! Accumulator mouseDeltaAccumulator = new() { SmoothingWindow = 0.0f;
resetInput;
! Keyboard = Keyboard.current;
! mToggle = false;

! mouse scroll delta
! _scrollWheelDelta;

! xerty to access the scroll wheel delta
! _scrollWheelDelta => _scrollWheelDelta;

! LeftMouseButtonDown => _isLeftMouseButtonDown;

! Helpers
! CanvasFader.cs
! DisabledUI.cs
! DisabledUI.cs
! GameUI.cs
! SpotlightBridge.cs
! SpotlightGroup.cs
! UIScreen.cs

! Utility
! Extension
! ArrayUtil.cs
! MonoBehaviourUtilities.cs
! ScrollingRawImage.cs
! Singleton.cs
! Spin.cs
! AdvancedPlayer.cs

참조 0개
public float MouseX => Input.GetAxis("Mouse X");

public LayerMask HitMask; //마우스 위치 레이캐스터 관련 감지
public RaycastHit mouseForIdhit; //마우스 위치 관련 변수

Unity 메시지 | 참조 0개
private void OnDestroy()
{
    DisposeInputs();
}
```

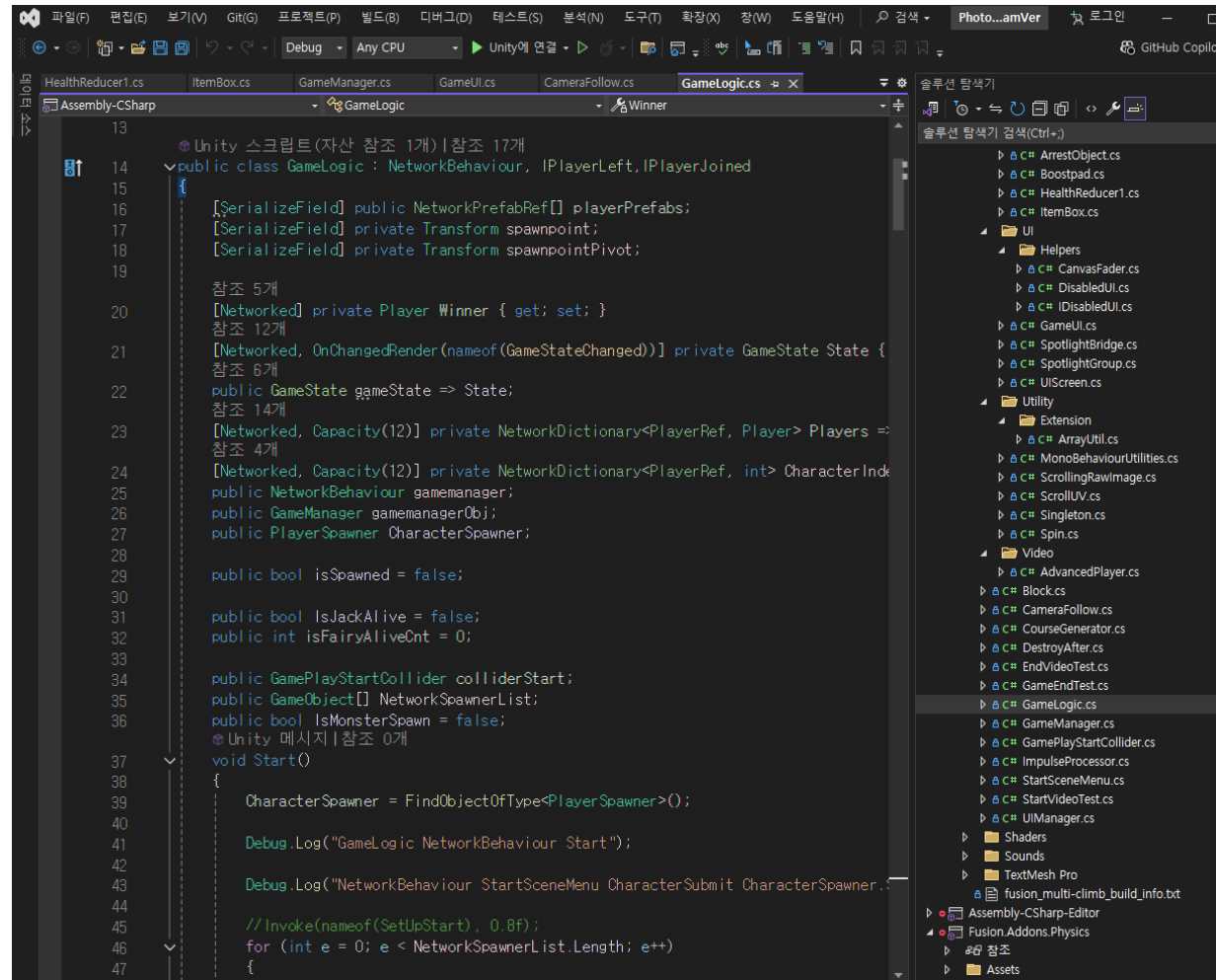
InputManager

네트워크 플레이어의 입력 관련 처리를 담당하는 클래스

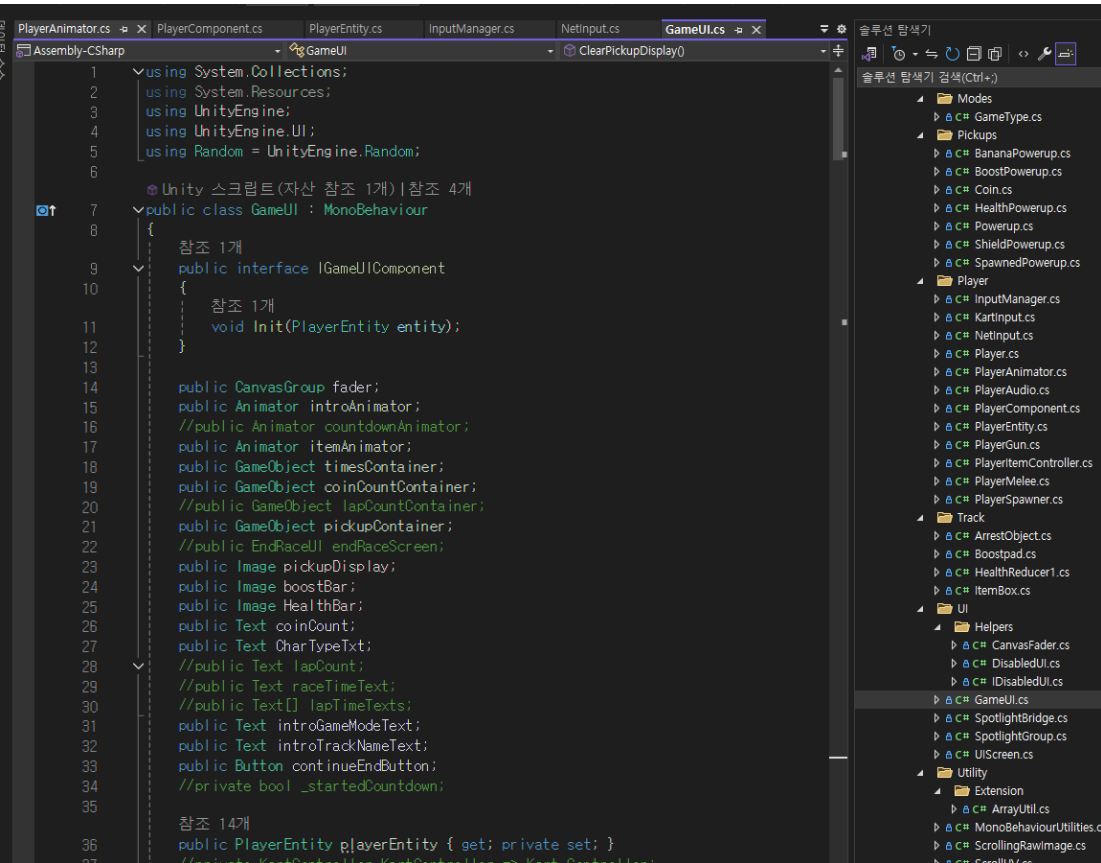
게임로직,게임UI

GameLogic

플레이어의 Add,Left관련 처리 및 접속 캐릭터(목숨1개)의 생존여부에 따른 게임결말(엔딩영상) 관련 상태처리



주요한 기능&스크립트 요약



GameUI

로컬플레이어,게임 관련 모든 정보를 표현,처리하는 클래스

주요한 기능&스크립트 요약(카테고리)

- Interfaces
 - ICameraController.cs
 - ICollidable.cs
- Managers
 - AudioManager.cs
 - ResourceManager.cs

카메라,충돌체 인터페이스,오디오&리소스 매니저

- MenuImplementation
 - MenuConnection.cs
 - MenuConnectionBehaviour.cs
 - MenuUIController.cs
- Modes
 - GameType.cs
- Pickups
 - BananaPowerup.cs
 - BoostPowerup.cs
 - Coin.cs
 - HealthPowerup.cs
 - Powerup.cs
 - ShieldPowerup.cs
 - SpawnedPowerup.cs

메뉴접속,접속행동 컨트롤,세션접속UI 컨트롤러,게임타입,아이템 픽업&관리

- Monster
 - IEnemyFSM.cs
 - IEnemyFSM_Network.cs
 - IEnemyMeleeCollider.cs
 - IEnemyMeleeCollider_Network.cs
 - IEnemyMeleeFSM.cs
 - IEnemySpawner.cs
 - IEnemySpawner_Network.cs
 - NetworkHealth.cs
- IEnemy.cs
- SEnemy.cs

몬스터 AI 행동 패턴 정의, 대기&추적&공격 리스폰주기(생성/해제) 관리

- Player
 - InputManager.cs
 - KartInput.cs
 - NetInput.cs
 - Player.cs
 - PlayerAnimator.cs
 - PlayerAudio.cs
 - PlayerComponent.cs
 - PlayerEntity.cs
 - PlayerGun.cs
 - PlayerItemController.cs
 - PlayerMelee.cs
 - PlayerSpawner.cs

플레이어 네트워크 입력제어,NetInput 정의,플레이어 조작,플레이어 애니메이션,오디오,플레이어 구성요소,플레이어 엔티티,원거리&근접형 공격 구현,아이템 컨트롤러

- Track
 - ArrestObject.cs
 - Boostpad.cs
 - HealthReducer1.cs
 - ItemBox.cs

속박아이템,부스트패드,지뢰,아이템박스 등의 트랙&맵 상호작용 오브젝트

- UI
 - Helpers
 - CanvasFader.cs
 - DisabledUI.cs
 - IDisabledUI.cs
 - GameUI.cs
 - SpotlightBridge.cs
 - SpotlightGroup.cs
 - UIScreen.cs

UI캔버스 페이더,UI아이템 활성화/비활성 인터페이스,게임UI,Spotlight유닛,UIScreen 정의

- Audio
 - AudioDestroyer.cs
 - FollowTarget.cs
- Bridge
 - ItemDisplayBridge.cs
- Fusion Helpers
 - FusionObjectPool.cs
 - FusionObjectPoolRoot.cs
 - TickHelper.cs

오디오해제자,오디오 3D 위치 추적,아이템브릿지,퓨전 오브젝트 캐싱 헬퍼 클래스

- Utility
 - Extension
 - ArrayUtil.cs
 - MonoBehaviourUtilities.cs
 - ScrollingRawImage.cs
 - ScrollUV.cs
 - Singleton.cs
 - Spin.cs
- Video
 - AdvancedPlayer.cs
- Block.cs
- CameraFollow.cs
- CourseGenerator.cs
- DestroyAfter.cs
- EndVideoTest.cs
- GameEndTest.cs
- GameLogic.cs
- GameManager.cs
- GamePlayStartCollider.cs
- ImpulseProcessor.cs
- StartSceneMenu.cs
- StartVideoTest.cs
- UIManager.cs

Array유틸리티,UI호버액션,스크롤링 이미지,UV,싱글톤 등 유틸리티 비디오플레이어,코스제네레이터,플레이어 카메라 추적할당,게임로직시스템,UI매니저 등의 구조

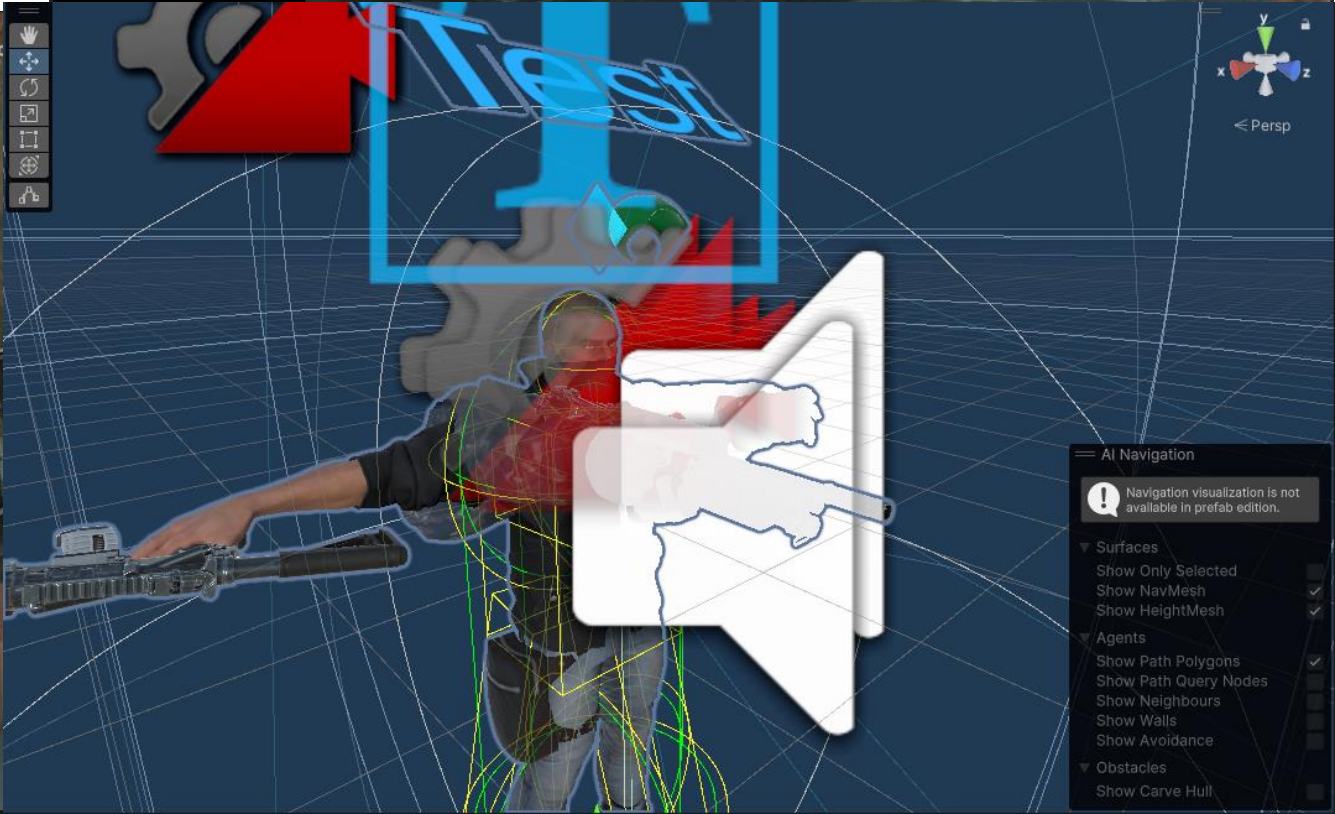
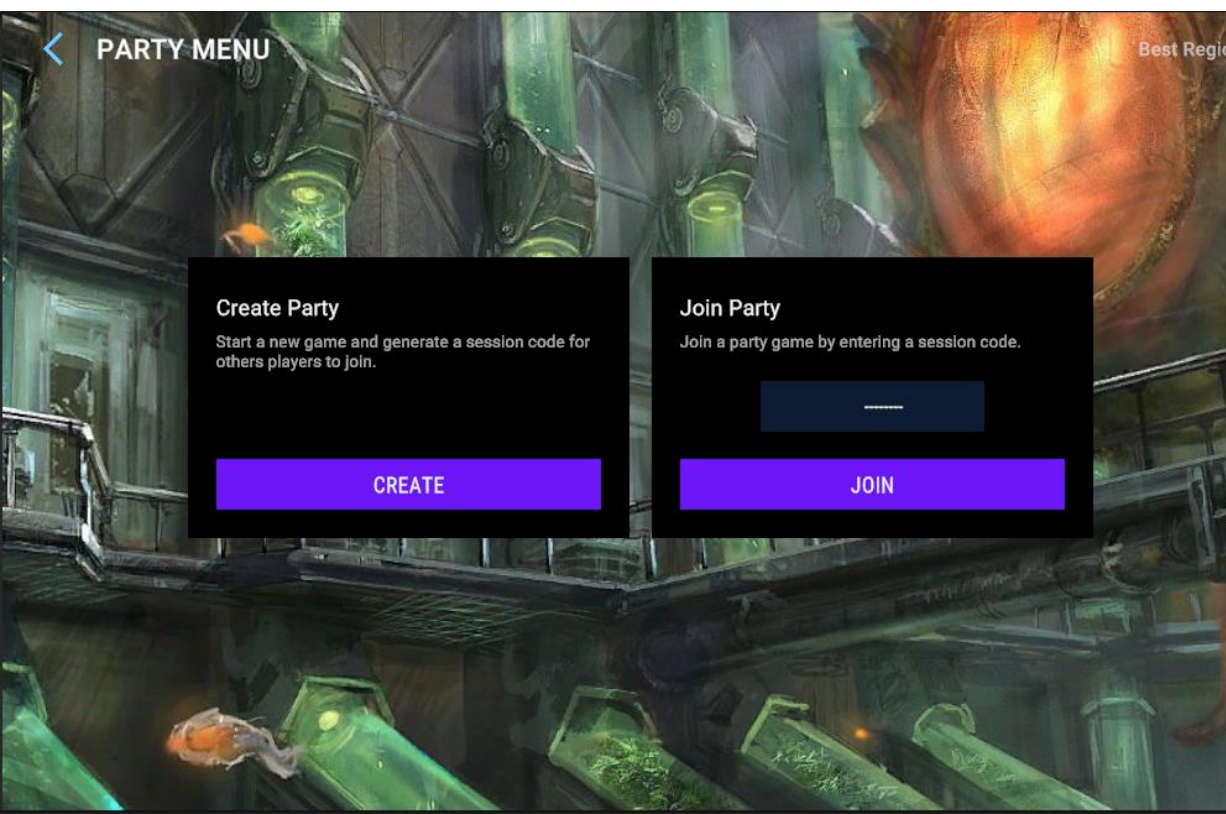


게임플레이



프로젝트8. 멀티플레이어 몬스터레이드(PVP,PVE)

개요: 멀티플레이어 1인칭 fps PVP,PVE 콘텐츠, 스테이지별 존재하는 Network Monster AI와의 전투가 가능한 PVP,PVE 콘텐츠
작업 기간: 2025.01.08~2025.01.17 (9일)
작업 분담: 4인개발(개발1,아트2,기획1)
본인 분담: 개발총괄
기여&소통: 아트파트에게 개발에 필요한 리소스 요청 및 커뮤니케이션, 기획파트와 기술&일정적으로 가능한 콘텐츠 기능구현 조율



주요한 기능&스크립트 요약(카테고리)

```
Gameplay
▸ C# Gameplay.cs
▸ C# HeadGearPickup.cs
▸ C# HealthPickup.cs
▸ C# MentalPickup.cs
▸ C# RotateObject.cs
▸ C# SceneObjects.cs
▸ C# SpawnPoint.cs
▸ C# SprayDecal.cs
▸ C# WeaponPickup.cs
```

게임플레이,헤드기어,hp물약,정신력 물약,오브젝트 회전 유틸리티,씬 오브젝트 관리,캐릭터 스폰,스프레이 데칼,무기류 아이템 픽업 등 정의

```
Menu
▸ C# MenuConnectionBehaviour.cs
▸ C# MenuConnectionPlugin.cs
▸ C# MenuUIController.cs
```

메뉴접속,접속행동컨트롤,세션접속UI컨트롤러

```
MonsterAI
▸ C# IEnemyFSM_Network.cs
▸ C# IEnemyMeleeCollider_Network.cs
▸ C# IEnemySpawner_Network.cs
▸ C# MonsterHealth.cs
▸ C# Status.cs
```

몬스터 AI 행동 패턴 정의, 대기&추적&공격 리스폰 주기(생성/해제) 관리

```
Player
▸ C# BodyHitbox.cs
▸ C# Footsteps.cs
▸ C# Health.cs
▸ C# Player.cs
▸ C# PlayerInput.cs
▸ C# PlayerManager.cs
▸ C# WorldNickname.cs
```

바디 히트박스,풋 스텝,체력&정신력데이터,플레이어 조작, 플레이어 네트워크 입력제어,플레이어 세션 접속관리,닉네임

게임UI,메뉴UI,조준점,접속해제 뷰,게임오버 뷰,게임정보,게임플레이 뷰,체력&정신력 UI,킬링데이터 UI피드백,UI메뉴 뷰,UI플레이어 뷰,UI점수판,UI설정 뷰,UI웨폰즈

```
Weapons
▸ C# ProjectileVisual.cs
▸ C# Weapon.cs
▸ C# Weapons.cs
▸ C# Stage1Start.cs
```

발사체 표현&관련데이터처리,개별무기특성 정의,무기리스트 관리

```
UI
▸ C# GameUI.cs
▸ C# MenuUI.cs
▸ C# UICrosshair.cs
▸ C# UIDisconnectedView.cs
▸ C# UIGameOverView.cs
▸ C# UIGameplayInfo.cs
▸ C# UIGameplayView.cs
▸ C# UIHealth.cs
▸ C# UIKillFeed.cs
▸ C# UIKillFeedItem.cs
▸ C# UIMenuView.cs
▸ C# UIPlayerView.cs
▸ C# UIScoreboard.cs
▸ C# UIScoreboardRow.cs
▸ C# UISettingsView.cs
▸ C# UIWeapons.cs
```




게임플레이



프로젝트4. 멀티플레이어 카트 경주

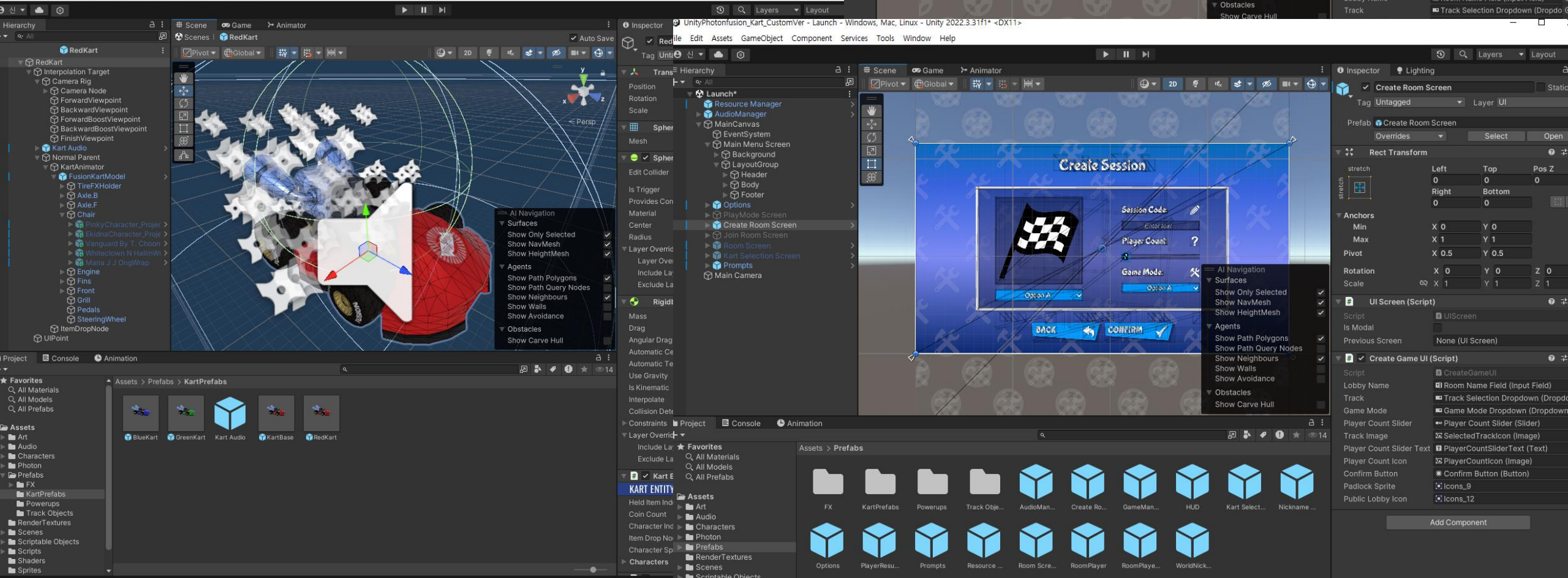
개요: 접속자 별 다른 종류의 차량,캐릭터 외형을 선택할 수 있으며, 접속한 자동차로 선택한 맵 에서 경주하는 게임

작업 기간: 2024.11.03~2024.11.11 (8일)

작업 분담: 1인개발(개발총괄)

UnityPhotonFusion_Kart_CustomVer - Launch - Windows, Mac, Linux - Unity 2022.3.31f1 <DX11>

File Edit Assets GameObject Component Services Tools Window Help



주요한 기능&스크립트 요약(카테고리)

- ▶ **Kart**
 - ▶ **C#** KartAnimator.cs
 - ▶ **C#** KartAudio.cs
 - ▶ **C#** KartCamera.cs
 - ▶ **C#** KartCharacter.cs
 - ▶ **C#** KartComponent.cs
 - ▶ **C#** KartController.cs
 - ▶ **C#** KartEntity.cs
 - ▶ **C#** KartInput.cs
 - ▶ **C#** KartItemController.cs
 - ▶ **C#** KartLapController.cs

카트 애니메이션, 오디오, 카메라, 카트 캐릭터, 카트 구성요소, 카트 입력, 카트 아이템, 카트 경주 등 카트 관련 기능

- ▶ **Managers**
 - ▶ **C#** AudioManager.cs
 - ▶ **C#** GameManager.cs
 - ▶ **C#** InterfaceManager.cs
 - ▶ **C#** LevelManager.cs
 - ▶ **C#** ResourceManager.cs
- ▶ **Modes**
 - ▶ **C#** GameType.cs

오디오, 게임, 인터페이스, 레벨, 리소스 매니저, 게임 모드 등

- ▶ **Networking**
 - ▶ **C#** ClientInfo.cs
 - ▶ **C#** GameLauncher.cs
 - ▶ **C#** RoomPlayer.cs
 - ▶ **C#** ServerInfo.cs
- ▶ **Pickups**
 - ▶ **C#** BananaPowerup.cs
 - ▶ **C#** BoostPowerup.cs
 - ▶ **C#** Coin.cs
 - ▶ **C#** Powerup.cs
 - ▶ **C#** SpawnedPowerup.cs
- ▶ **Scriptable Objects**
 - ▶ **C#** KartDefinition.cs
 - ▶ **C#** TrackDefinition.cs

클라이언트정보, 서버정보, 네트워크 게임런처, 룸 유저, 아이템픽업&관리, 카트&트랙정보 등

- ▶ **Track**
 - ▶ **C#** Boostpad.cs
 - ▶ **C#** CameraTrack.cs
 - ▶ **C#** Checkpoint.cs
 - ▶ **C#** FinishLine.cs
 - ▶ **C#** ItemBox.cs
 - ▶ **C#** Track.cs
 - ▶ **C#** TrackStartSequence.cs
- ▶ **Triggers**
 - ▶ **C#** RespawnTrigger.cs
 - ▶ **C#** SplashZone.cs

부스트패드, 카메라 트랙, 경주 체크포인트, 경주 완료라인, 아이템박스, 트랙, 트랙 시작 컷씬 형태 애니메이션 정의

- ▶ **Attributes**
 - ▶ **C#** LayerAttribute.cs
- ▶ **Audio**
 - ▶ **C#** AudioDestroyer.cs
 - ▶ **C#** FollowTarget.cs
- ▶ **Bridge**
 - ▶ **C#** CountdownBridge.cs
 - ▶ **C#** ItemDisplayBridge.cs
- ▶ **Debug**
 - ▶ **C#** LatencyCode.cs
- ▶ **Fusion Helpers**
 - ▶ **C#** FusionObjectPool.cs
 - ▶ **C#** FusionObjectPoolRoot.cs
 - ▶ **C#** TickHelper.cs
- ▶ **Interfaces**
 - ▶ **C#** ICameraController.cs
 - ▶ **C#** ICollidable.cs

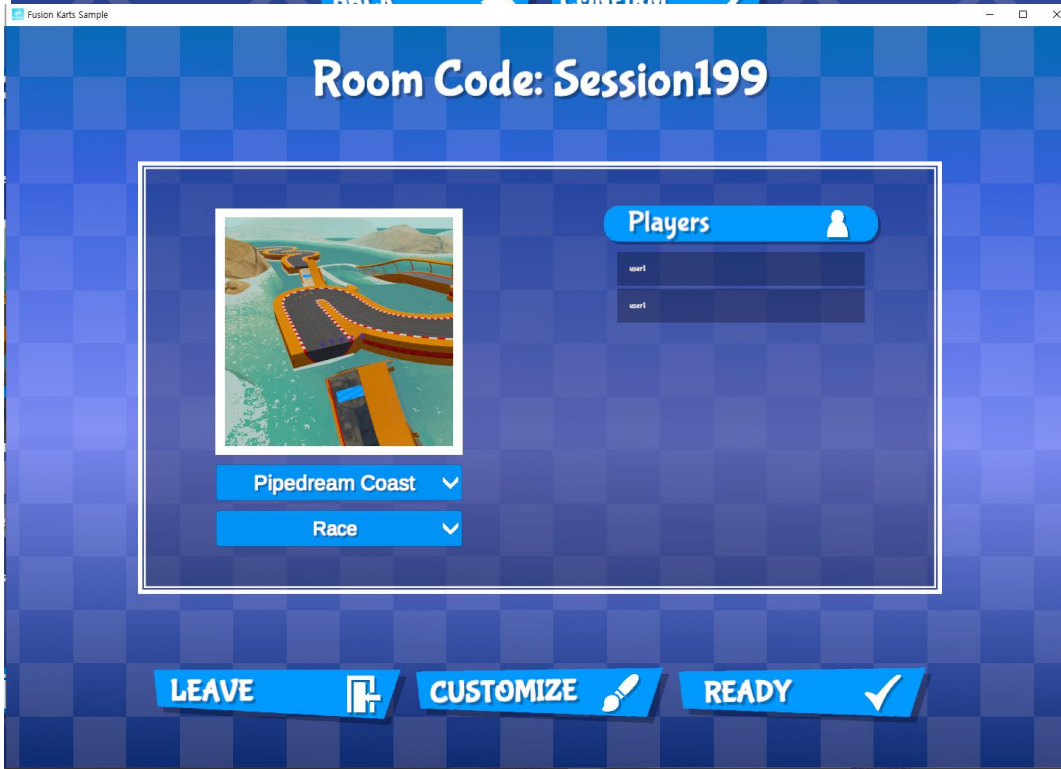
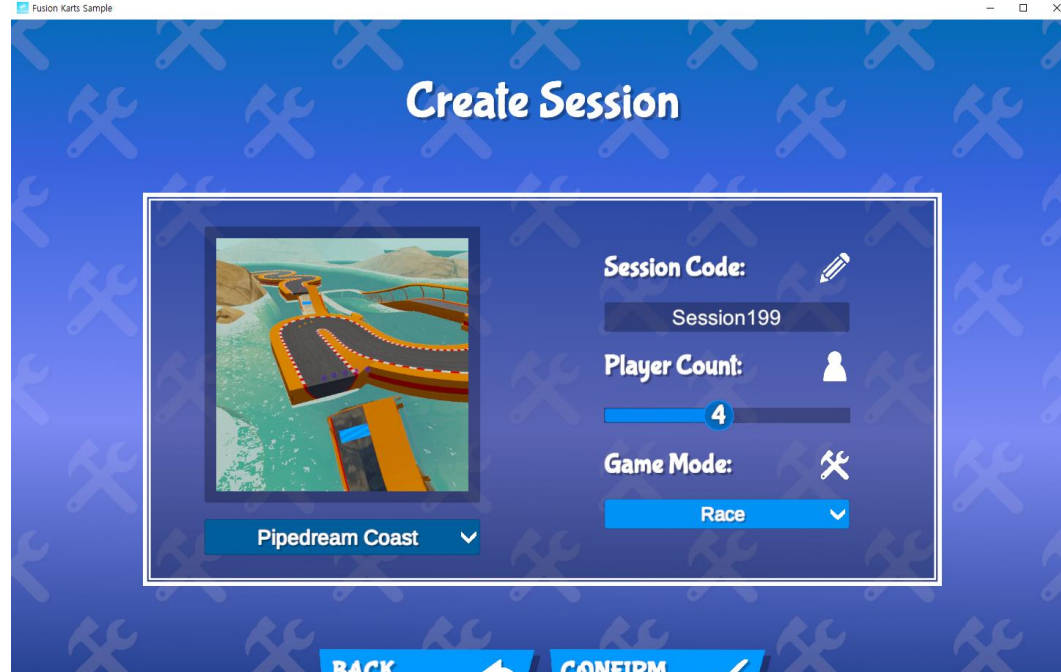
오디오해제, 오디오 객체 타깃 3D 위치 추적, 레이어속성, 카운트다운&아이템 2D롤렛형 브릿지 제공, 퓨전 오브젝트캐싱 헬퍼클래스, 카메라&충돌체 인터페이스 유틸리티

- ▶ **UI**
 - ▶ **Helpers**
 - ▶ **C#** CanvasFader.cs
 - ▶ **C#** DisabledUI.cs
 - ▶ **C#** IDisabledUI.cs
 - ▶ **C#** CreateGameUI.cs
 - ▶ **C#** DisconnectUI.cs
 - ▶ **C#** EndRaceUI.cs
 - ▶ **C#** GameUI.cs
 - ▶ **C#** GraphicsSettingsUI.cs
 - ▶ **C#** HoverAction.cs
 - ▶ **C#** JoinGameUI.cs
 - ▶ **C#** KartSelectUI.cs
 - ▶ **C#** LoadingScreenHints.cs
 - ▶ **C#** LobbyItemUI.cs
 - ▶ **C#** LobbyUI.cs
 - ▶ **C#** MainUIScreen.cs
 - ▶ **C#** NicknameUI.cs
 - ▶ **C#** PlayerResultItem.cs
 - ▶ **C#** ProfileSetupUI.cs
 - ▶ **C#** RegionUI.cs
 - ▶ **C#** RoomPlayerItemUI.cs
 - ▶ **C#** SpotlightBridge.cs
 - ▶ **C#** SpotlightGroup.cs
 - ▶ **C#** UIAudio.cs
 - ▶ **C#** UIScreen.cs
 - ▶ **C#** VolumeSlider.cs
 - ▶ **C#** WorldUINickname.cs

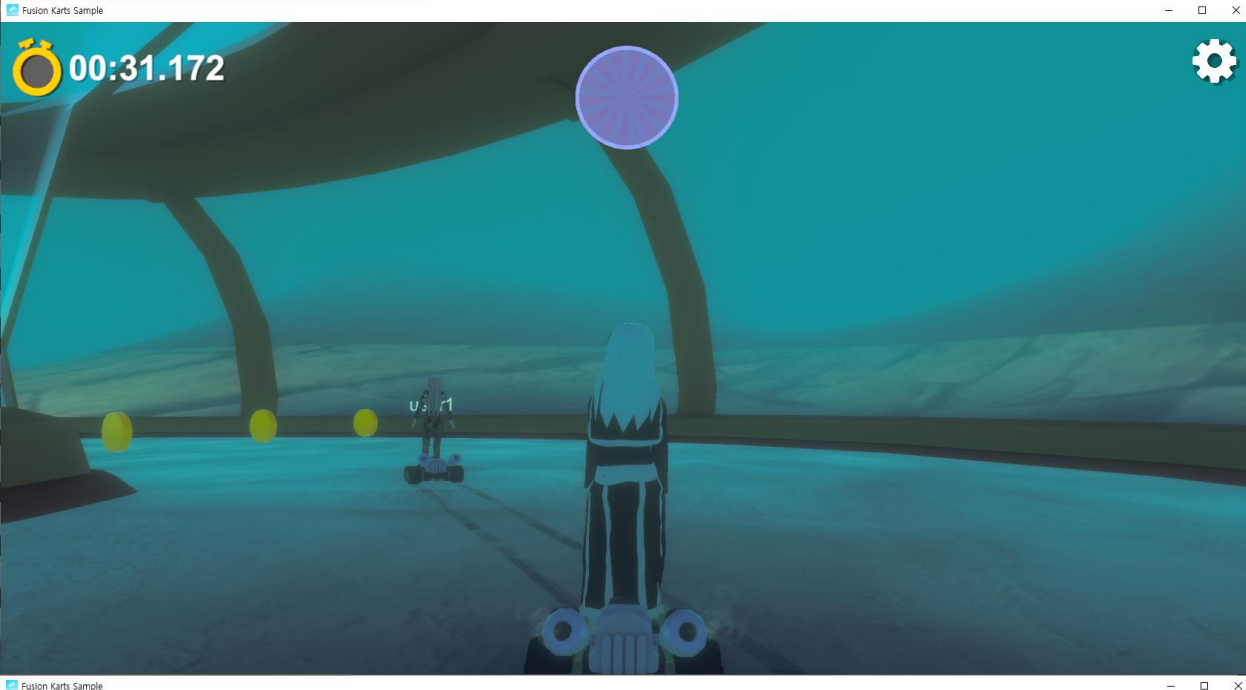
UI페이드, UI활성/비활성 인터페이스, 게임생성, 접속해제, 경주완료, 게임UI, 그래픽셋팅, 조인룸, 카트선택, 로딩스크린, 로비 세션아이템, 로비 UI, 메인UIScreen, 닉네임설정, 접속 세션지역 설정 등 게임접속 관련 인터페이스 제공

- ▶ **Utility**
 - ▶ **Extension**
 - ▶ **C#** ArrayUtil.cs
 - ▶ **C#** HoverOverButton.cs
 - ▶ **C#** InvokeOnEnable.cs
 - ▶ **C#** MonoBehaviourUtilities.cs
 - ▶ **C#** NetworkButton.cs
 - ▶ **C#** ScrollingRawImage.cs
 - ▶ **C#** ScrollUV.cs
 - ▶ **C#** Singleton.cs
 - ▶ **C#** Spin.cs

Array유틸리티, UI호버액션, 스크롤링 이미지, UV, 싱글톤 등 유틸리티



게임
플레이



프로젝트5. 달그림자

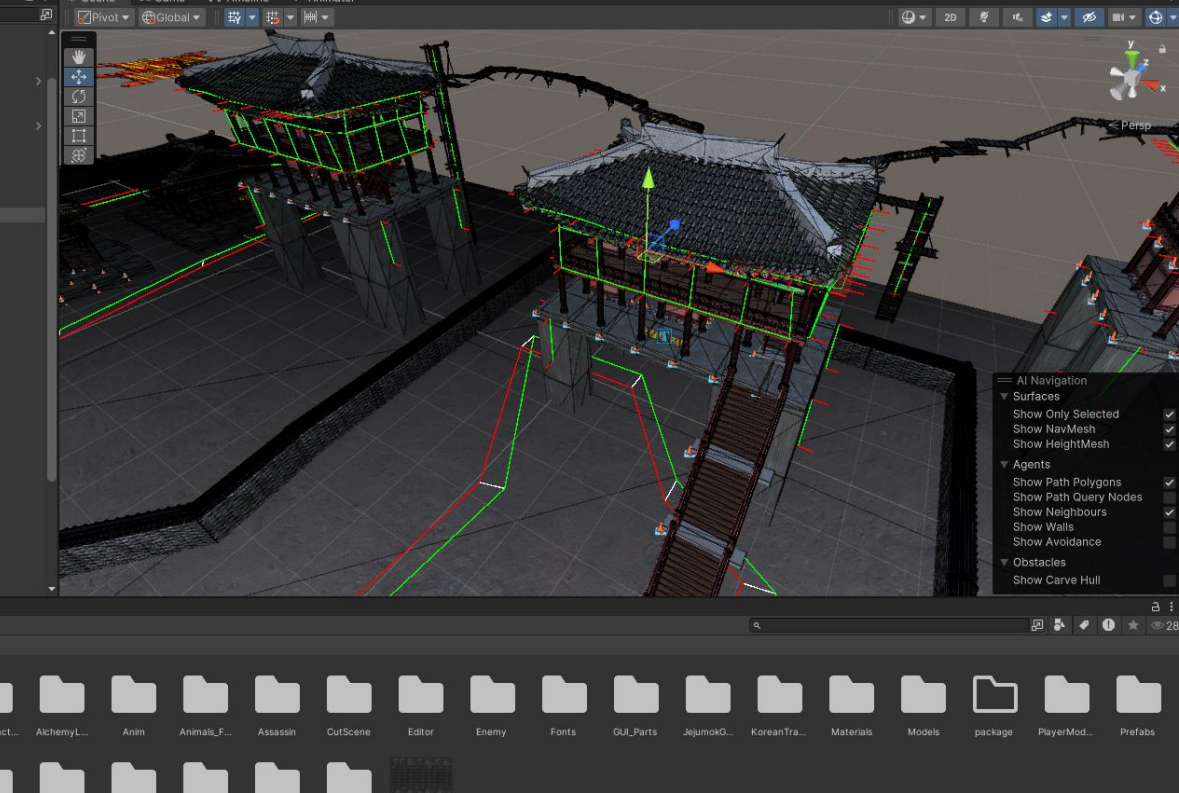
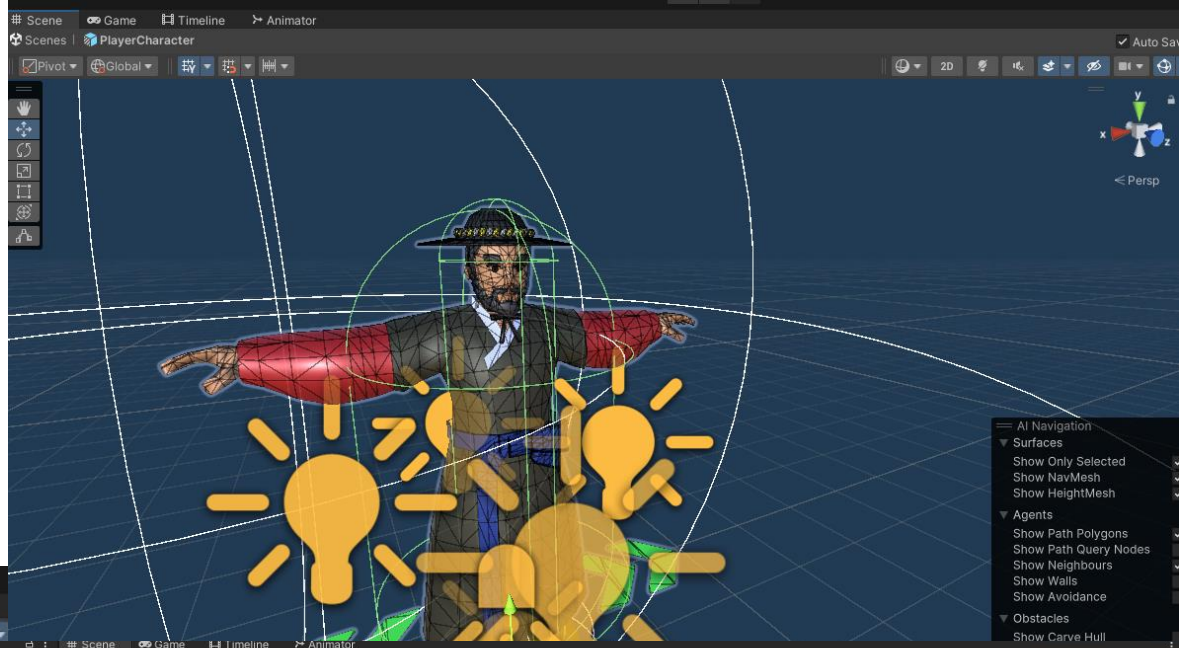
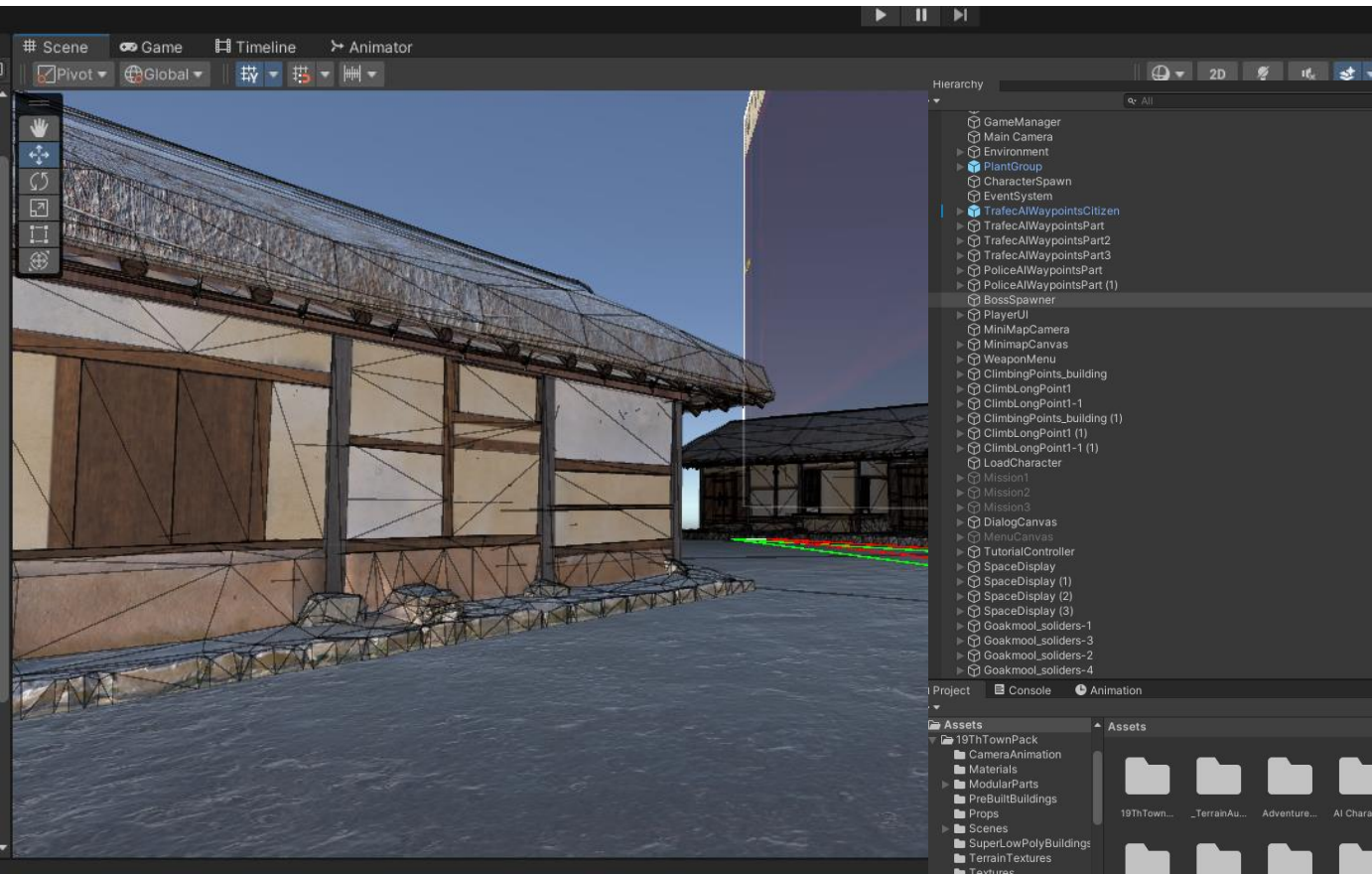
개요: 어쌔신크리드 기반 조선시대를 배경으로 한 파쿠르 잠입액션 게임, 미션&다이아로그 시스템

작업 기간: 2024.10.08~2024.10.22 (2주)

작업 분담: 4인개발(개발총괄)

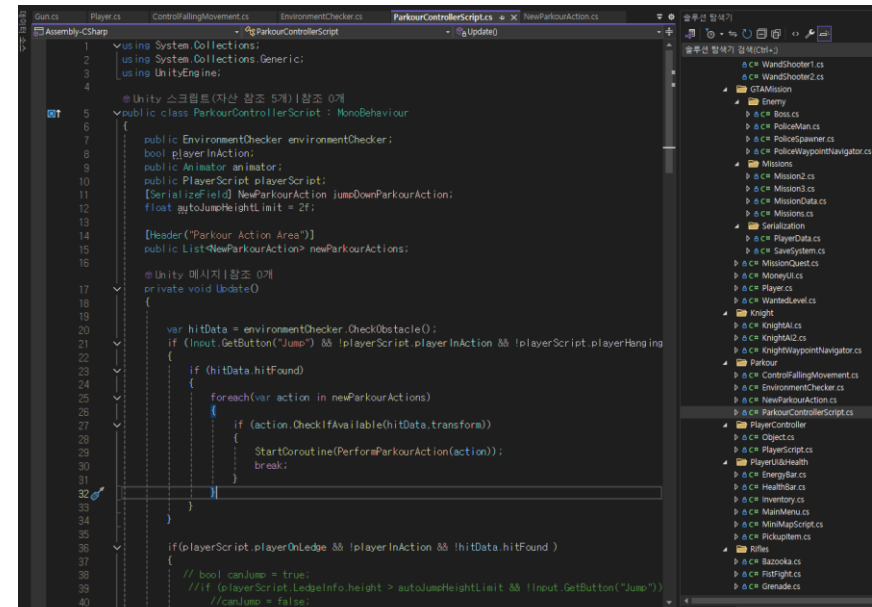
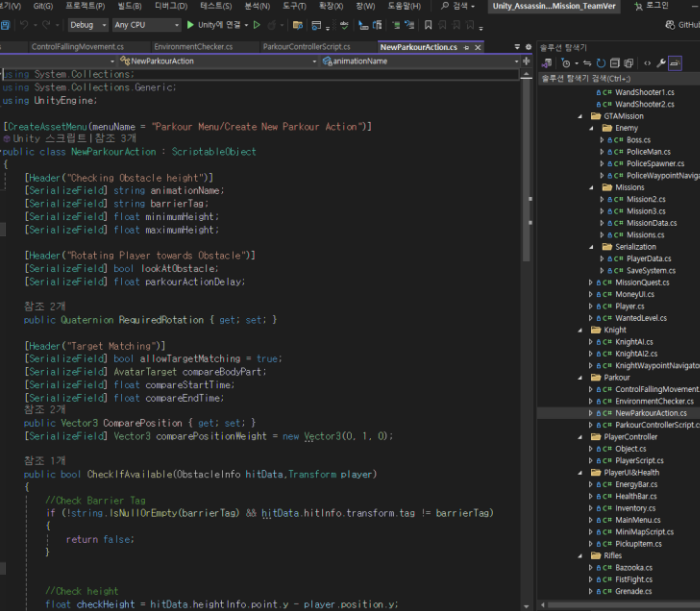
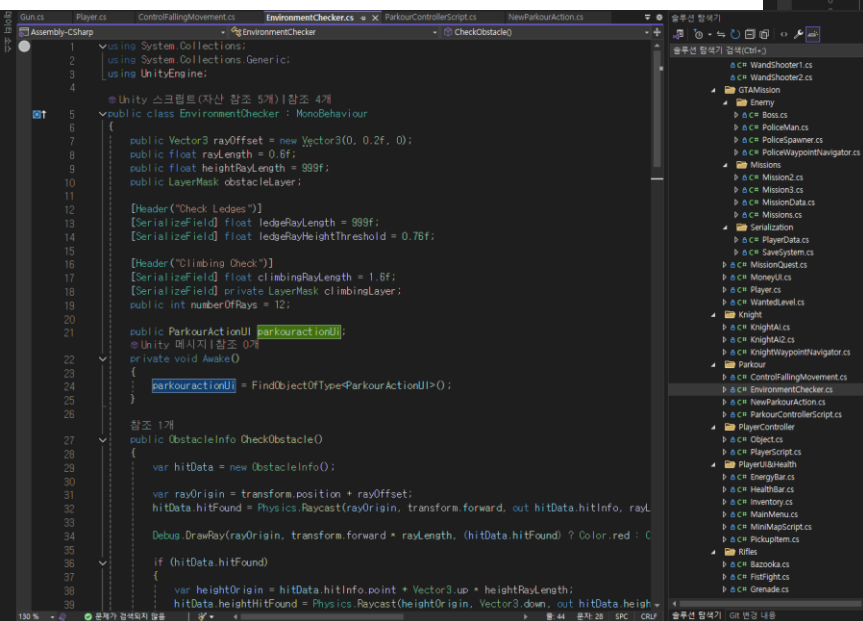
업부 분담: 개발1,아트2,기획1

기여&소통: 아트파트에게 개발에 필요한 리소스 요청 및 커뮤니케이션,기획파트와 기술&일정적으로 가능한 콘텐츠 기능구현 조율



파쿠르 시스템

주요한 기능&스크립트 요약



NetParkourAction

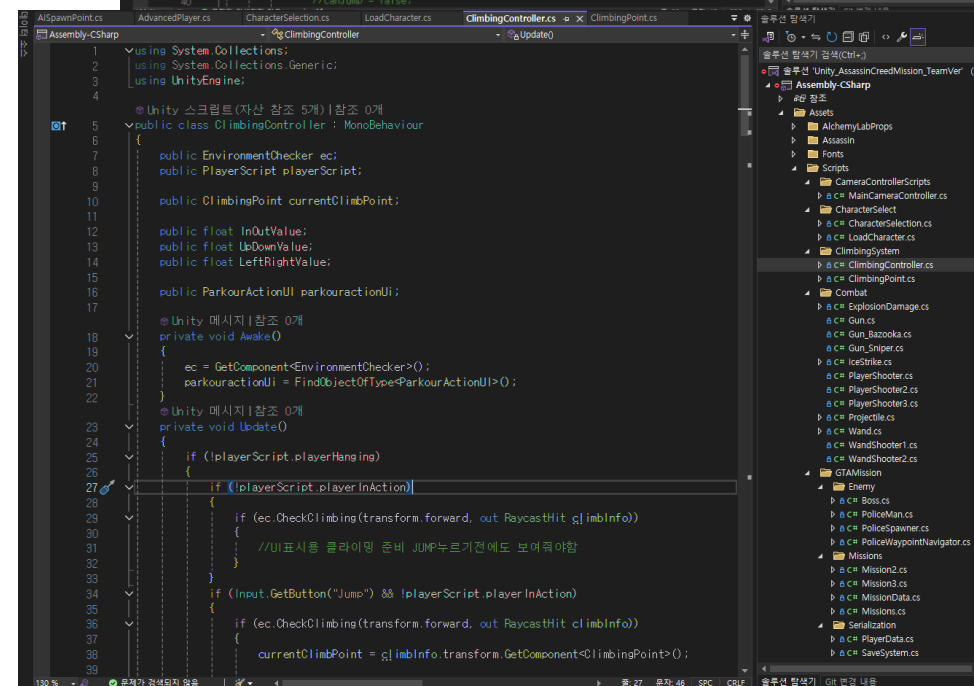
ScriptableObject 파쿠르 관련 객체

EnvironmentChecker

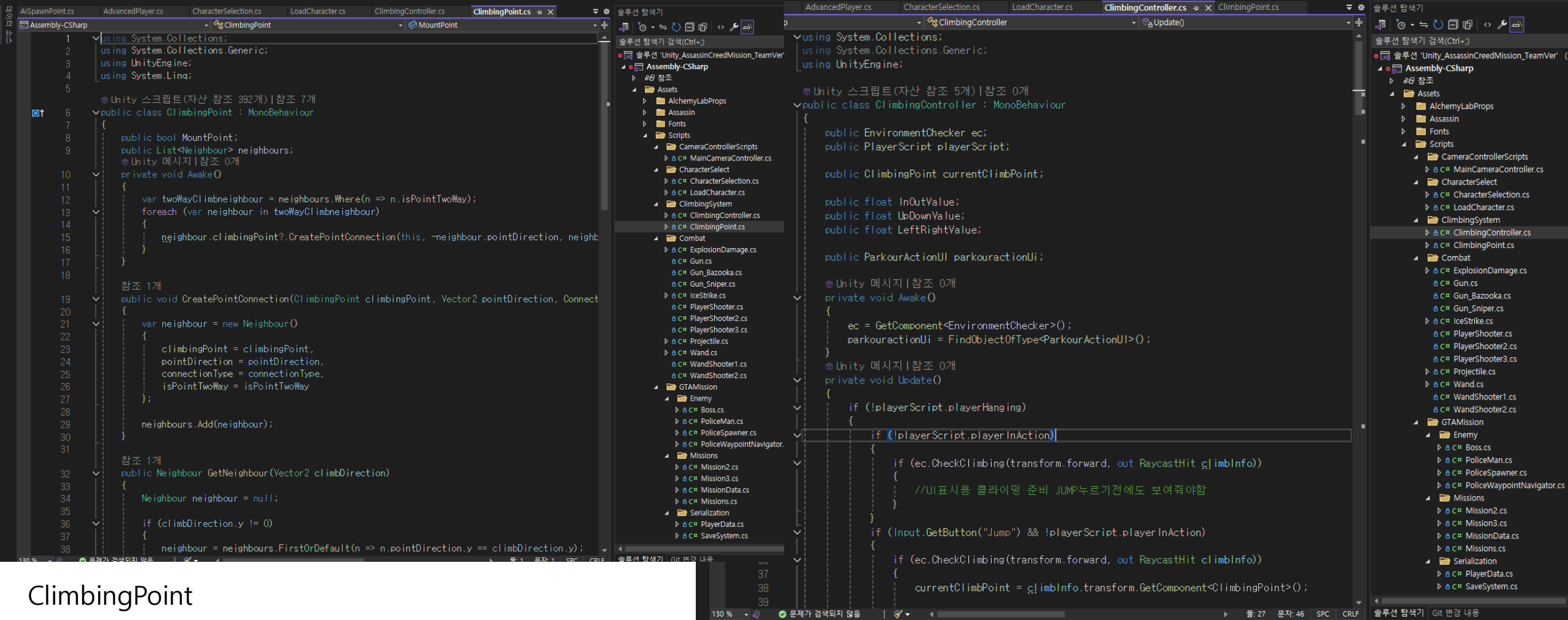
파쿠르 가능 지형지물&오브젝트 여부 검사(물리,충돌체 검사 등)

ParkourControllerScript

파쿠르 가능구간 여부 검사후 파쿠르액션 및 애니메이션&Transform 관련 처리 클래스



주요한 기능&스크립트 요약



ClimbingPoint

클라이밍 가능한 앵커 포인트 지점 정의

ClimbingController

파쿠르 벽타기 관련 액션 수행

주요한 기능&스크립트 요약(카테고리)

```
CameraControllerScripts
└─ C# MainCameraController.cs

CharacterSelect
└─ C# CharacterSelection.cs
└─ C# LoadCharacter.cs
```

카메라 컨트롤러,캐릭터 로드

```
ClimbingSystem
└─ C# ClimbingController.cs
└─ C# ClimbingPoint.cs
```

파쿠르 벽 등반 액션 기능구현,등반 가능 앵커포인트 정의

```
Combat
└─ C# ExplosionDamage.cs
└─ C# Gun.cs
└─ C# Gun_Bazooka.cs
└─ C# Gun_Sniper.cs
└─ C# IceStrike.cs
└─ C# PlayerShooter.cs
└─ C# PlayerShooter2.cs
└─ C# PlayerShooter3.cs
└─ C# Projectile.cs
└─ C# Wand.cs
└─ C# WandShooter1.cs
└─ C# WandShooter2.cs
```

무기형태 종류별 대전 스타일 정의 스크립트,총무기,바주카무기,저격총무기,얼음마법무기,발사체형정의,화염마법무기 등

```
TrafecAI
└─ C# AISpawner.cs
└─ C# BossSpawner.cs
└─ C# CharacterNavigatorScript.cs
└─ C# Waypoint.cs
└─ C# WaypointNavigator.cs
```

백성,NPC 등의 순찰&배회&이동 네비게이션 기능

```
Knight
└─ C# KnightAI.cs
└─ C# KnightAI2.cs
└─ C# KnightWaypointNavigator.cs
```

적군AI행동양식 정의,기능구현(순찰,대기,추적,공격)

```
GTAMission
└─ Enemy
└─ C# Boss.cs
└─ C# PoliceMan.cs
└─ C# PoliceSpawner.cs
└─ C# PoliceWaypointNavigator.cs
└─ Missions
└─ C# Mission2.cs
└─ C# Mission3.cs
└─ C# MissionData.cs
└─ C# Missions.cs
└─ Serialization
└─ C# PlayerData.cs
└─ C# SaveSystem.cs
└─ C# MissionQuest.cs
└─ C# MoneyUI.cs
└─ C# Player.cs
└─ C# WantedLevel.cs
└─ C# AISpawnPoint.cs
└─ C# CombatActionUI.cs
└─ C# Crosshair.cs
└─ C# CutSceneEnd.cs
└─ C# DamageIndicator.cs
└─ C# DayAndNight.cs
└─ C# GameManager.cs
└─ C# ParkourActionUI.cs
└─ C# WeaponMenu.cs
```

GTA미션 시스템 스크립트 참조, 미션 관련 데이터 시스템,플레이어데이터,저장시스템,현상금 시스템

```
Parkour
└─ C# ControlFallingMovement.cs
└─ C# EnvironmentChecker.cs
└─ C# NewParkourAction.cs
└─ C# ParkourControllerScript.cs
└─ PlayerController
└─ C# Object.cs
└─ C# PlayerScript.cs
```

플레이어 이동 조작 및 파쿠르 인터렉션 정의 스크립트, 파쿠르 기능 구현

```
PlayerUI&Health
└─ C# EnergyBar.cs
└─ C# HealthBar.cs
└─ C# Inventory.cs
└─ C# MainMenu.cs
└─ C# MiniMapScript.cs
└─ C# PickupItem.cs
└─ Rifles
└─ C# Bazooka.cs
└─ C# FistFight.cs
└─ C# Grenade.cs
└─ C# GrenadeThrower.cs
└─ C# Rifle.cs
└─ C# SingleMeleeAttack.cs
└─ C# SingleMeleeAttack2.cs
```

에너지바,체력바,인벤토리,메인메뉴,미니맵메뉴,아이템픽업,바주카,수류탄,라이플,근접형공격 정의

```
TutorialDialogue
└─ DialogSystem
└─ C# ArrowBlink.cs
└─ C# DialogSystem.cs
└─ C# FadeEffect.cs
└─ Tutorial
└─ C# TutorialActive.cs
└─ C# TutorialBase.cs
└─ C# TutorialCondStatus.cs
└─ C# TutorialController.cs
└─ C# TutorialDestroyTagObjects.cs
└─ C# TutorialDialog.cs
└─ C# TutorialFadeEffect.cs
└─ C# TutorialMovement.cs
└─ C# TutorialSample.cs
└─ C# TutorialSelection.cs
└─ C# TutorialTargeting_boolParam.cs
└─ C# TutorialTrigger.cs
└─ Video
└─ C# AdvancedPlayer.cs
```

튜토리얼 대화창 시스템 구현,튜토리얼 미션수행 컨트롤러 시스템,씬간 이동시에 관련 필요 영상 재생

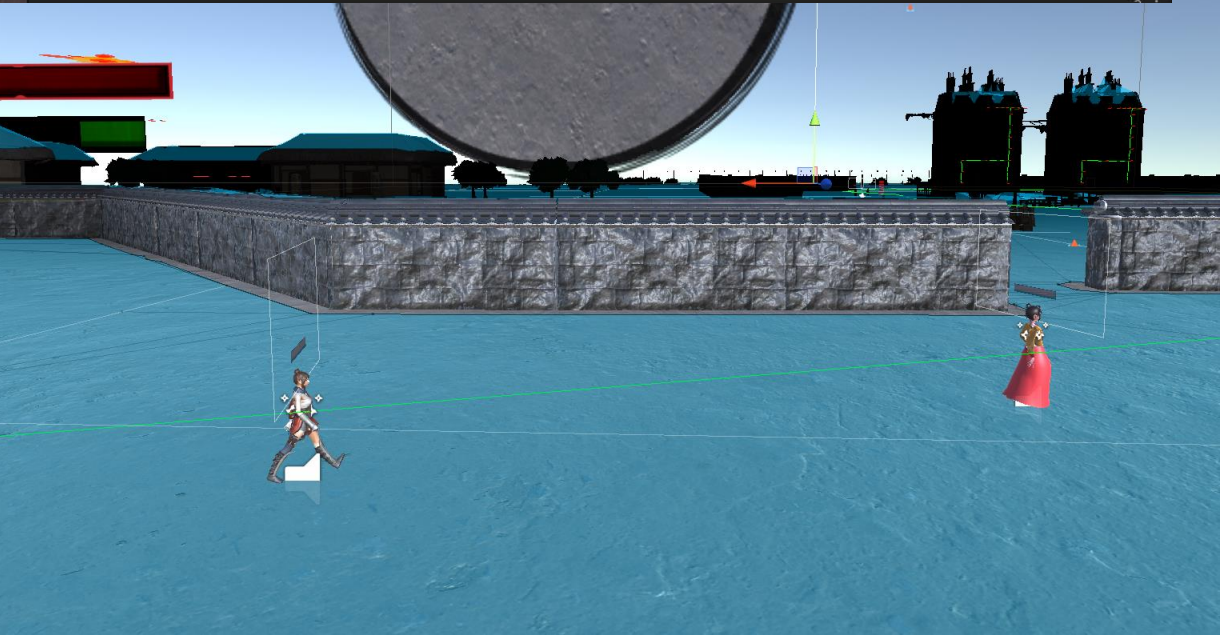
첫 번째 추적 대상: 윤성룡



두 번째 추적 대상: 윤성룡



게임
플레이



프로젝트6. 러브인유니버스

개요: 다이아로그,튜토리얼 시스템 기반 러브게임

작업 기간: 2024.08.14~2024.08.28 (2주)

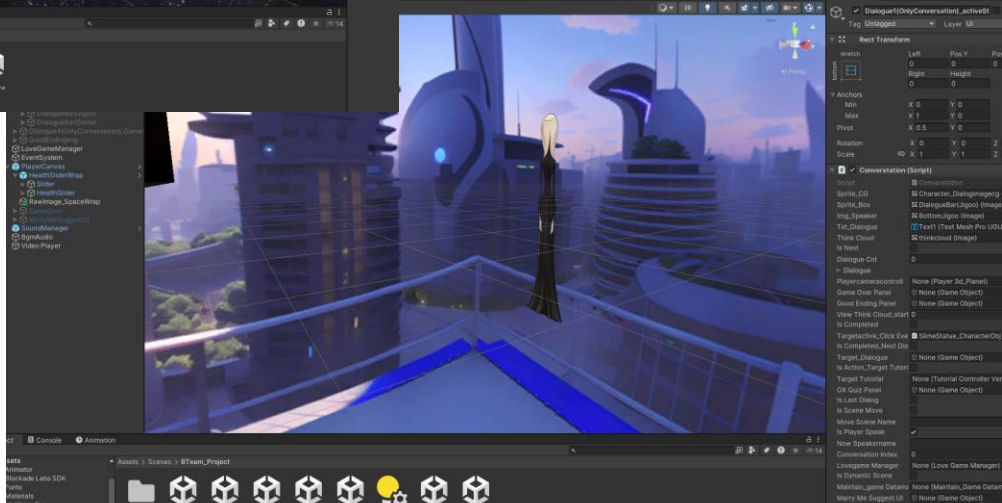
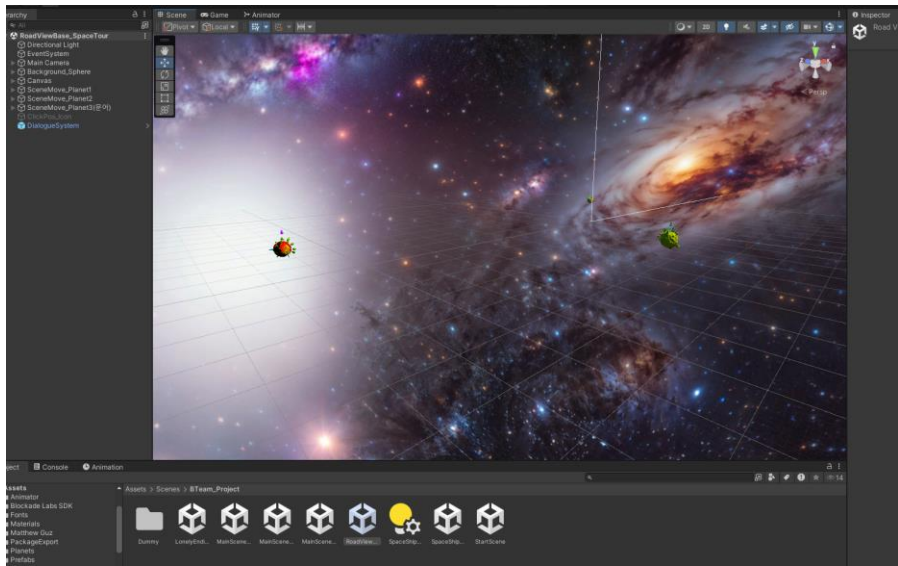
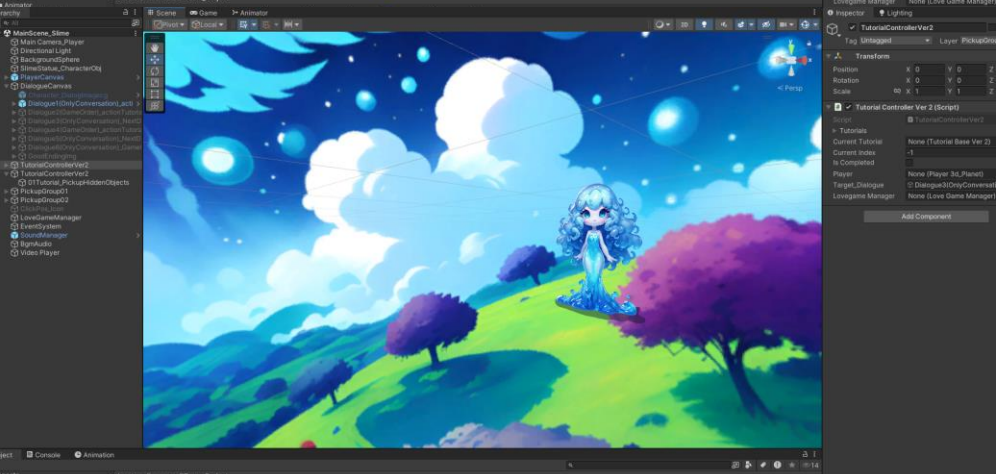
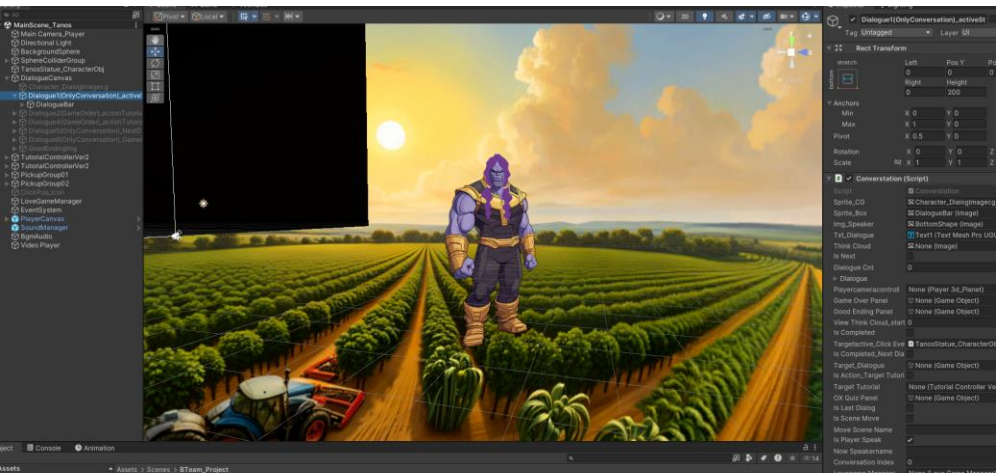
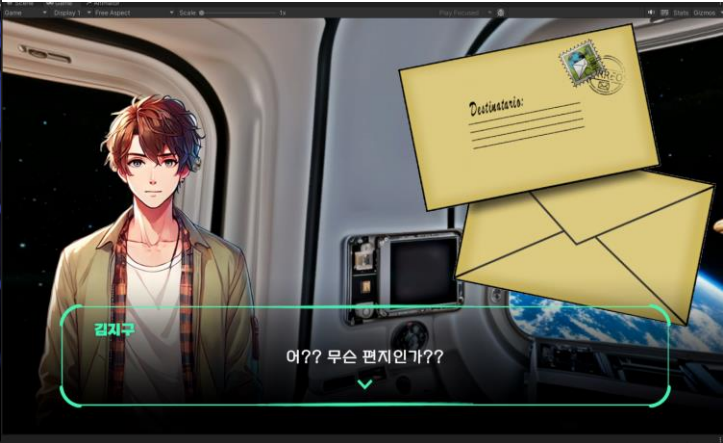
작업 분담: 4인개발(개발총괄)

업부 분담: 개발1,아트2,기획1

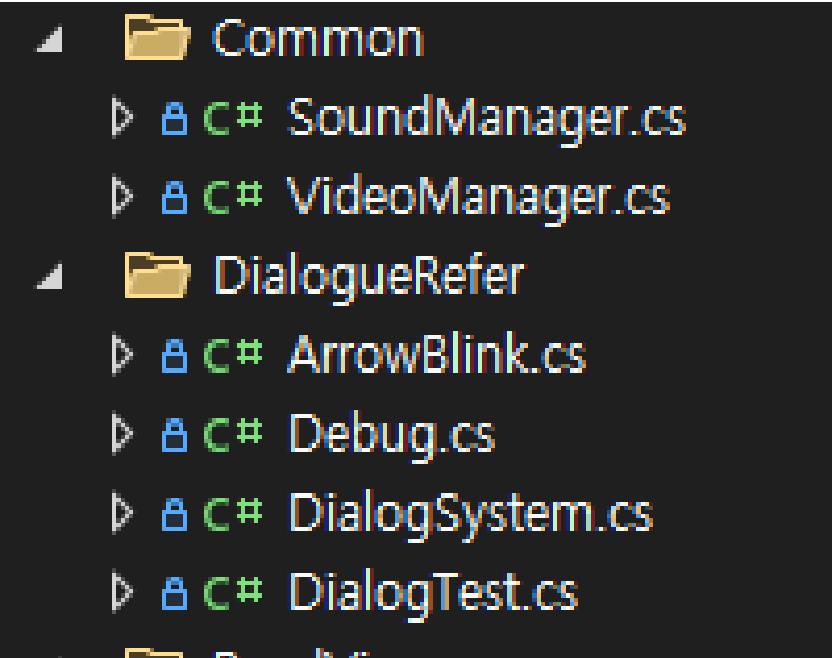
기여&소통: 아트파트에게 개발에 필요한 리소스

요청 및 커뮤니케이션,기획파트와 기술&일정적

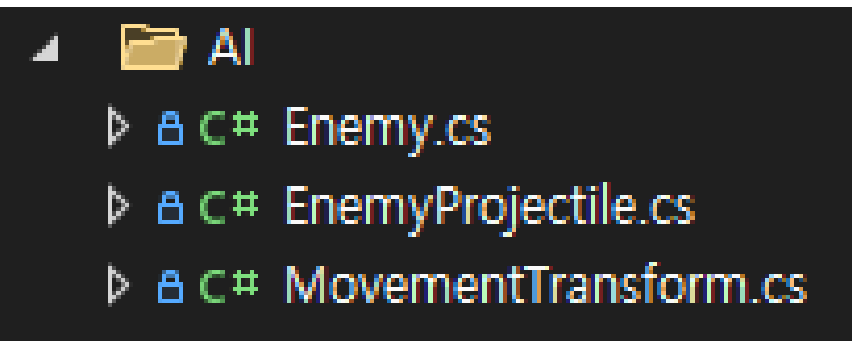
으로 가능한 콘텐츠 기능구현 조율



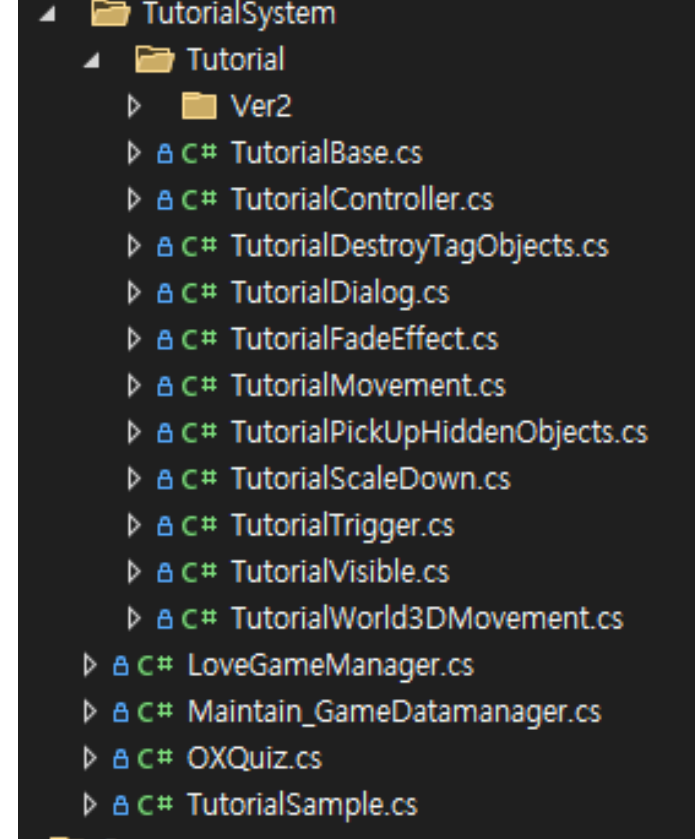
주요한 기능&스크립트 요약(카테고리)



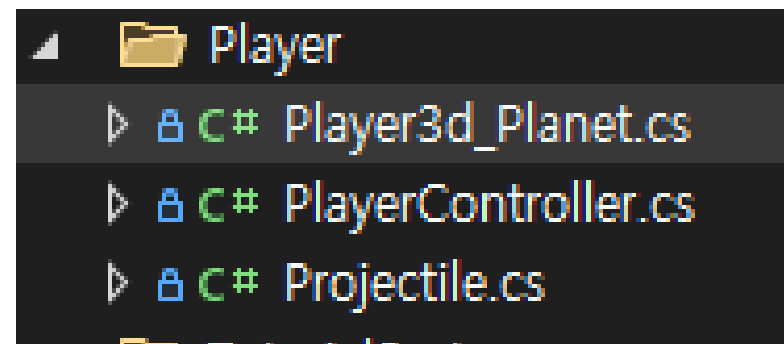
사운드,비디오매니저,다이아로그(대화창)시스템 레퍼런스



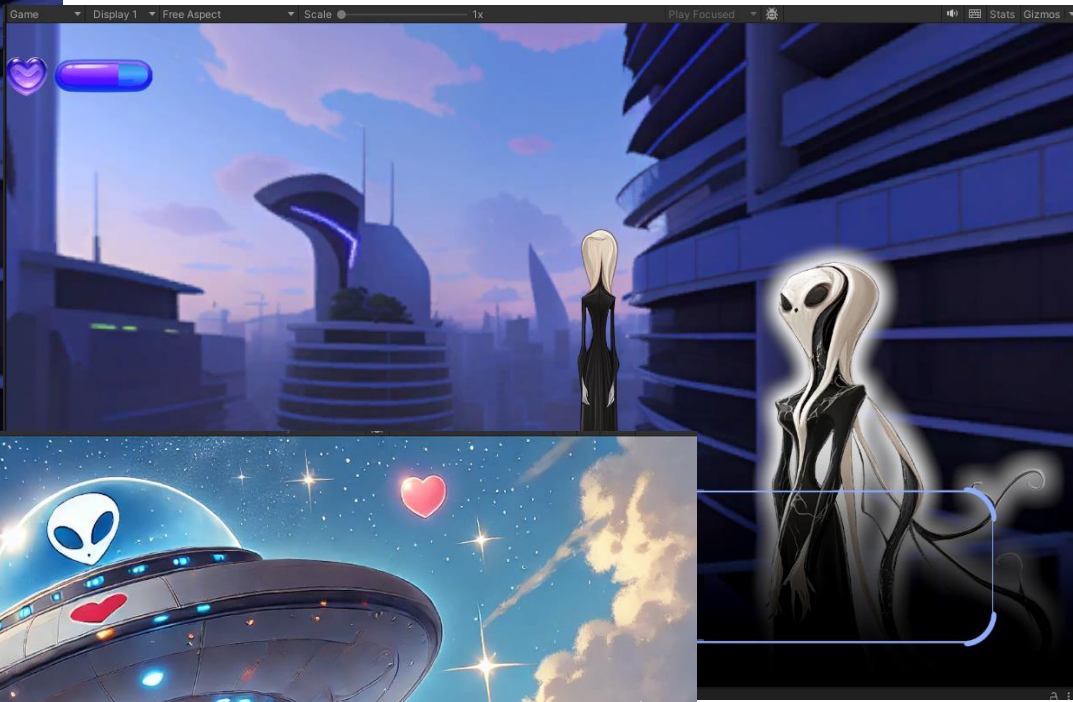
게임 내 미니게임 콘텐츠 내의 적군 AI시스템 ,적군 발사체 등



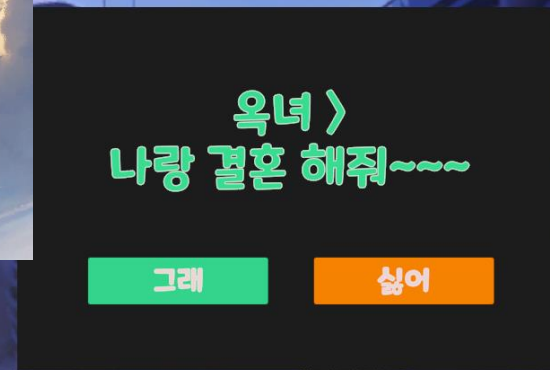
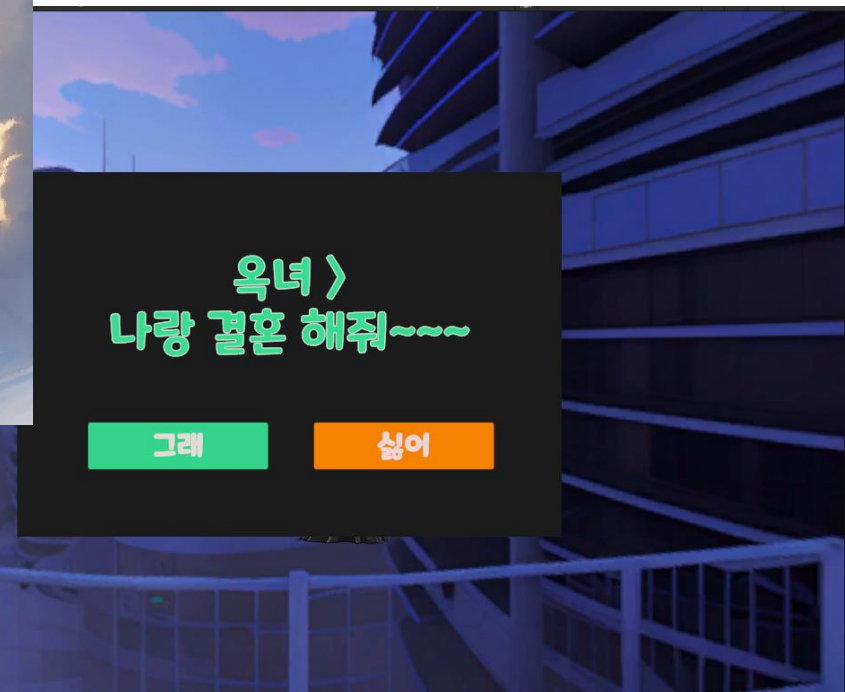
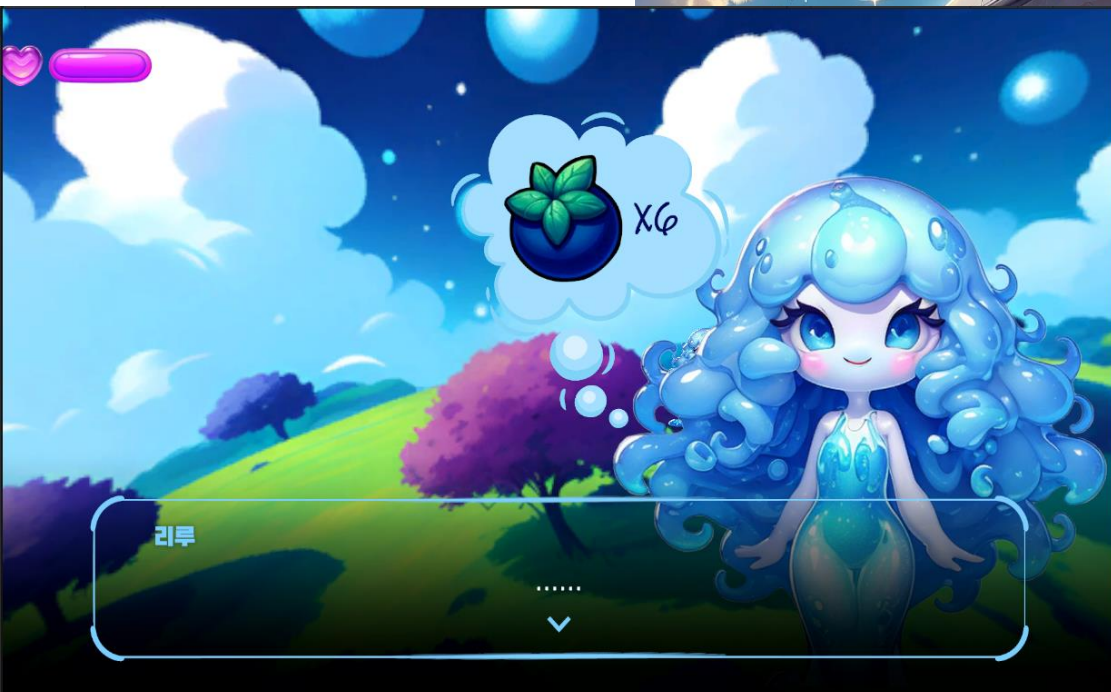
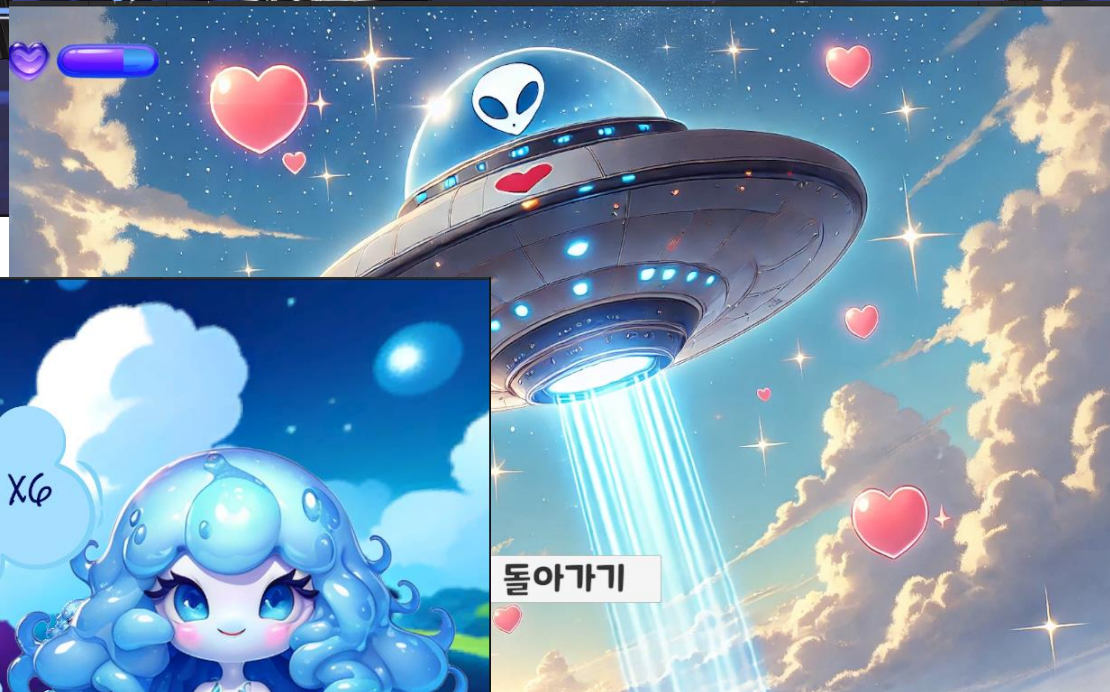
튜토리얼&대화&미션 시스템,튜토리얼 관련 상위클래스,게임매니저,게임싱글톤데이터 관리,OXQuiz미니게임 등



플레이어 관련 조작 스크립트(사용자 시점,이동 등)



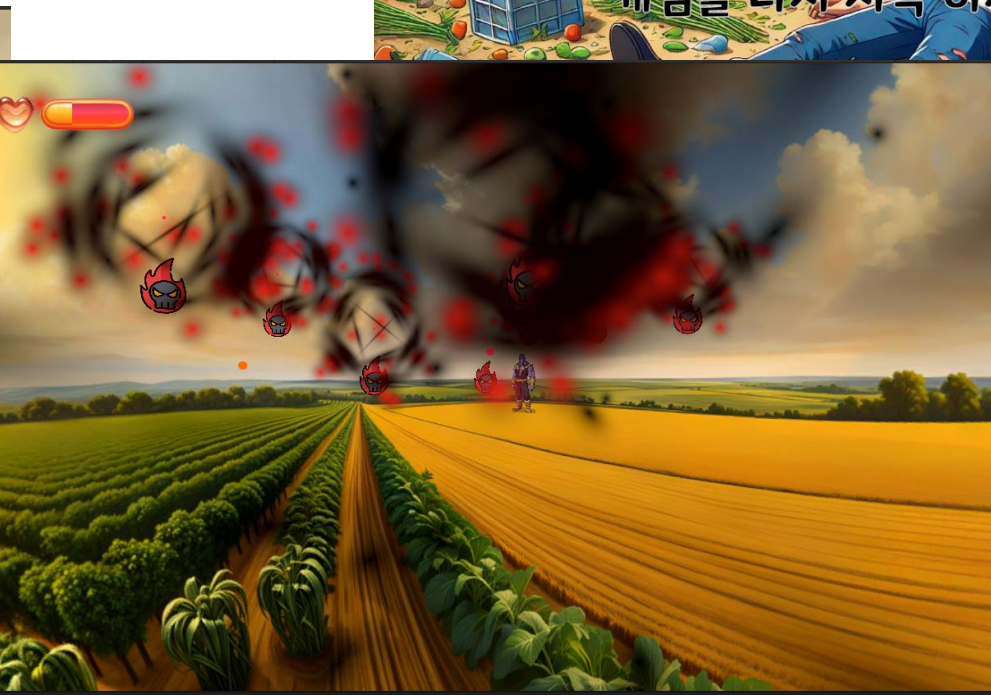
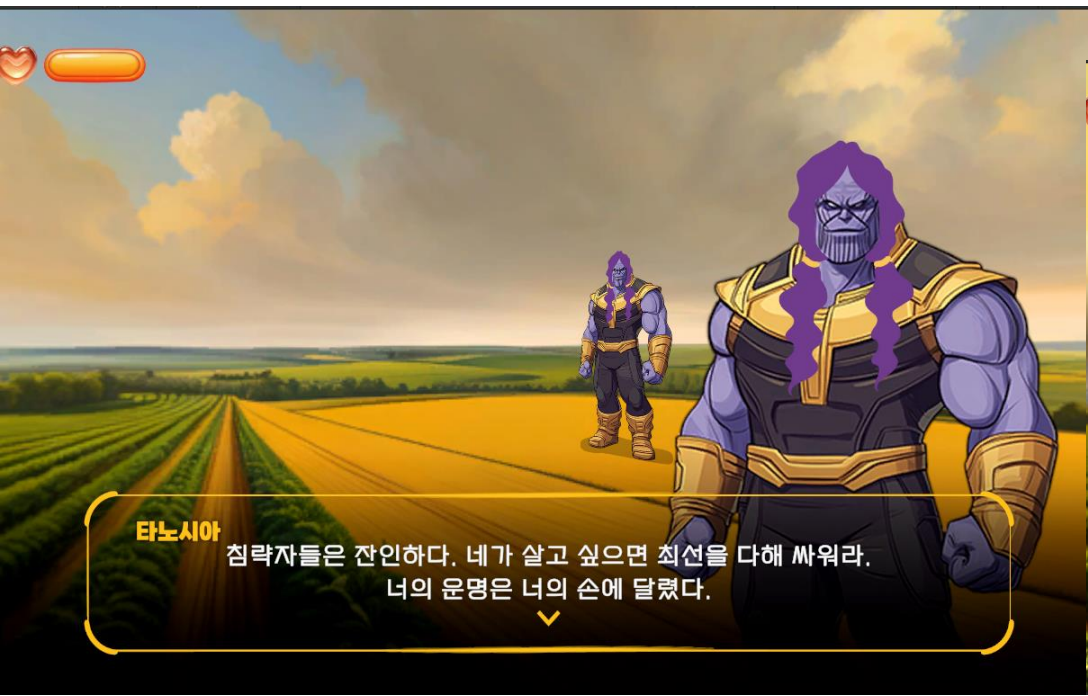
게임 플레이



옥녀 >
나랑 결혼 해줘~~~

그래

싫어

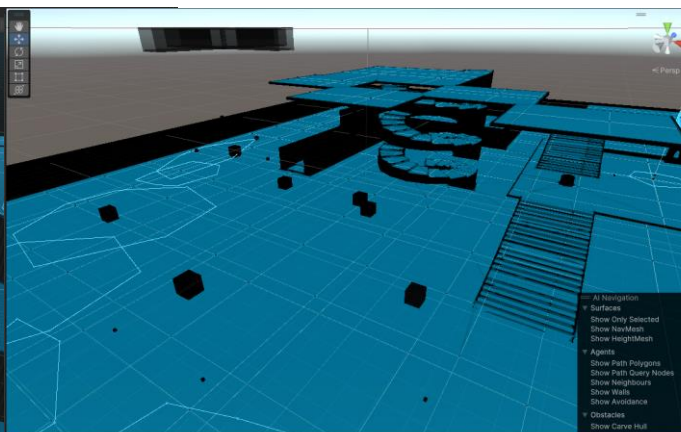
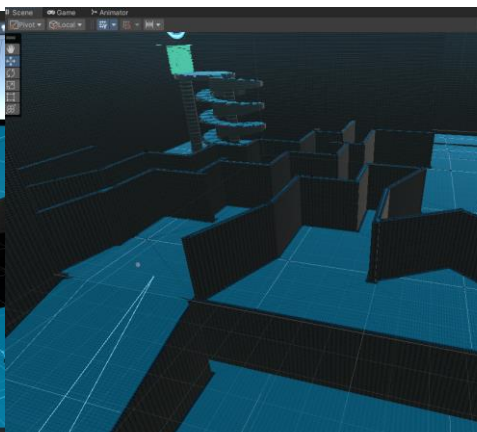
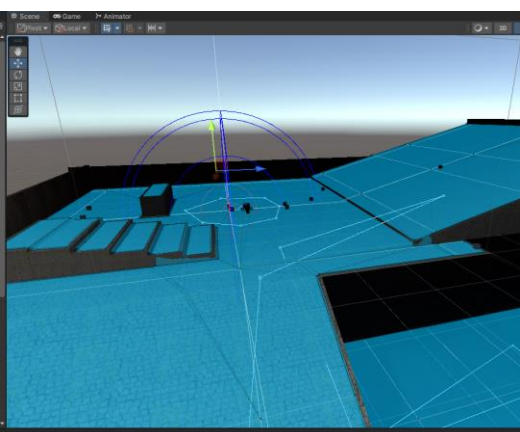
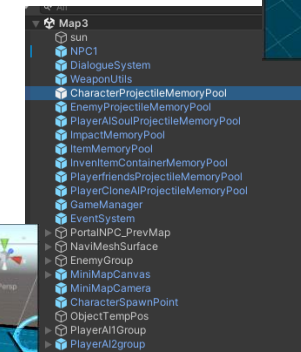
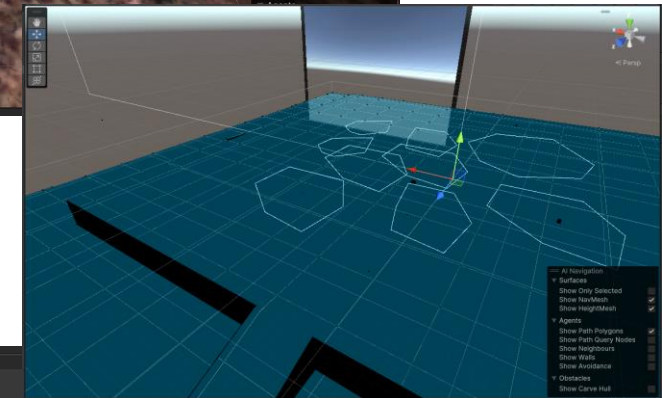
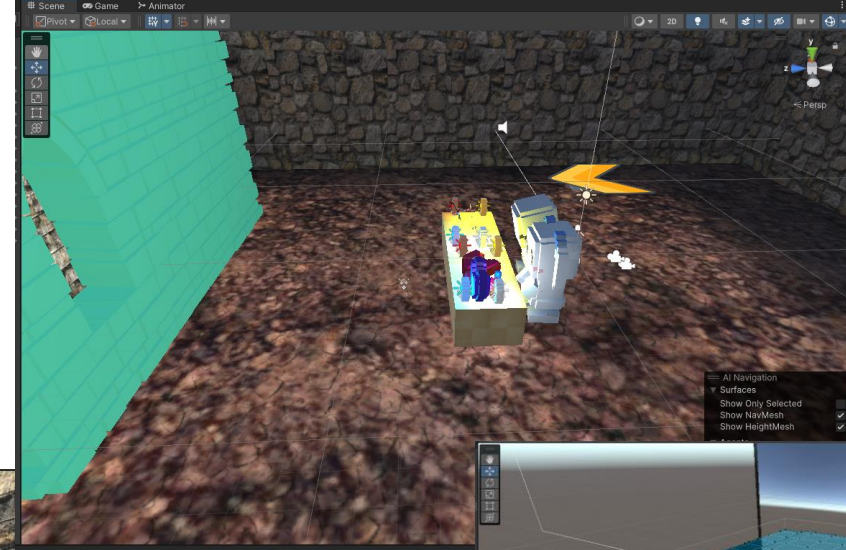
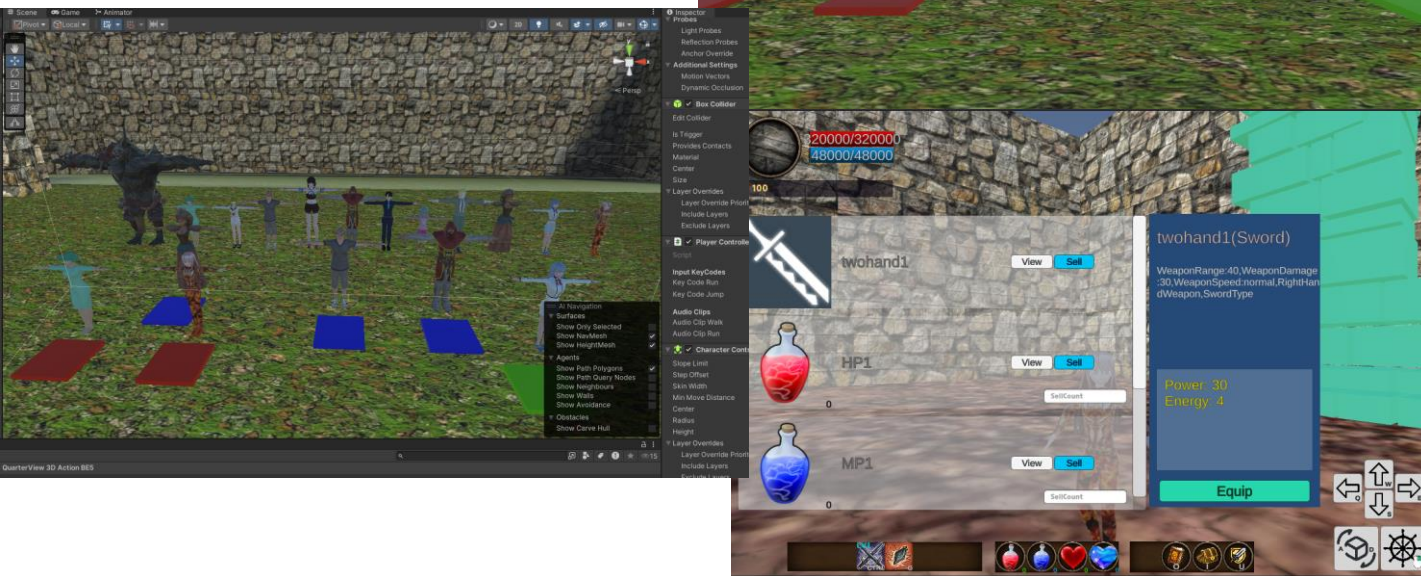


프로젝트7. SinjaUnityStory

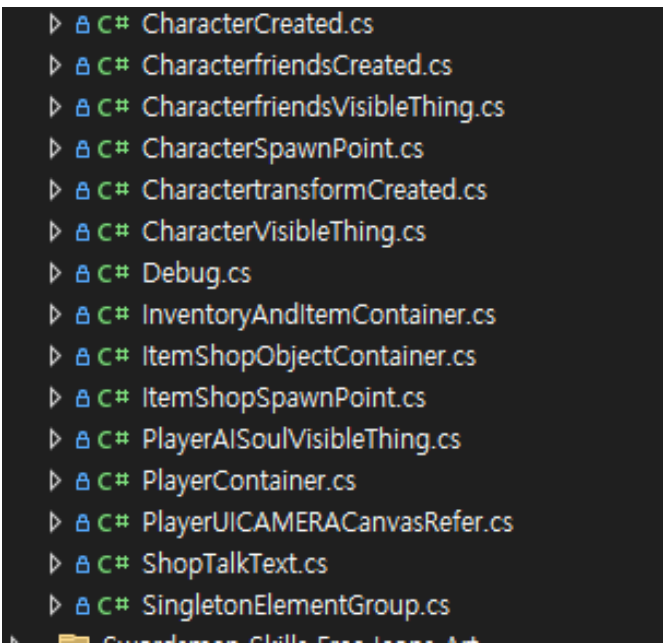
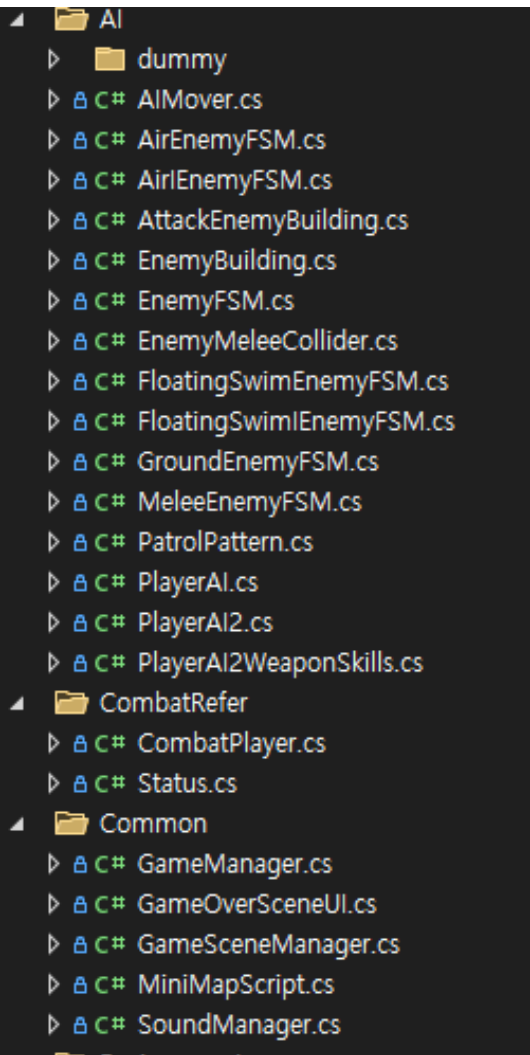
개요: 싱글 중규모 RPG 대전 게임

작업 기간: 2023.11~2024.03

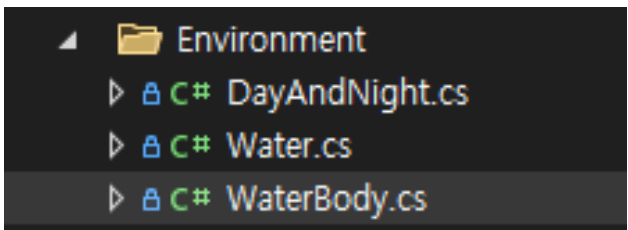
작업 분담: 1인개발(개발총괄)



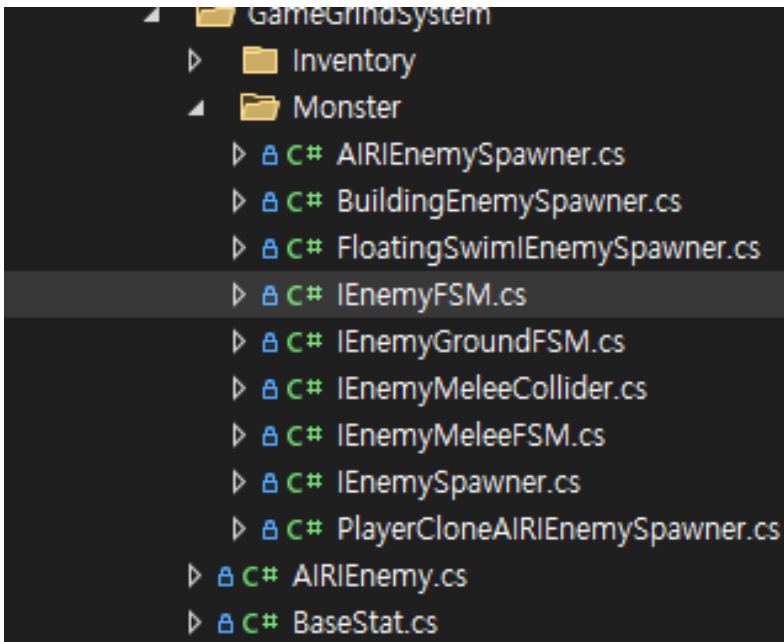
중규모 RPG 시스템 사용코드



캐릭터(레벨업에 따른 초월),캐릭터별 정령(초월) 의 초월효과의 따른 시각이펙트부여,상점Container,상점NPC,싱글톤 엘리먼트 그룹



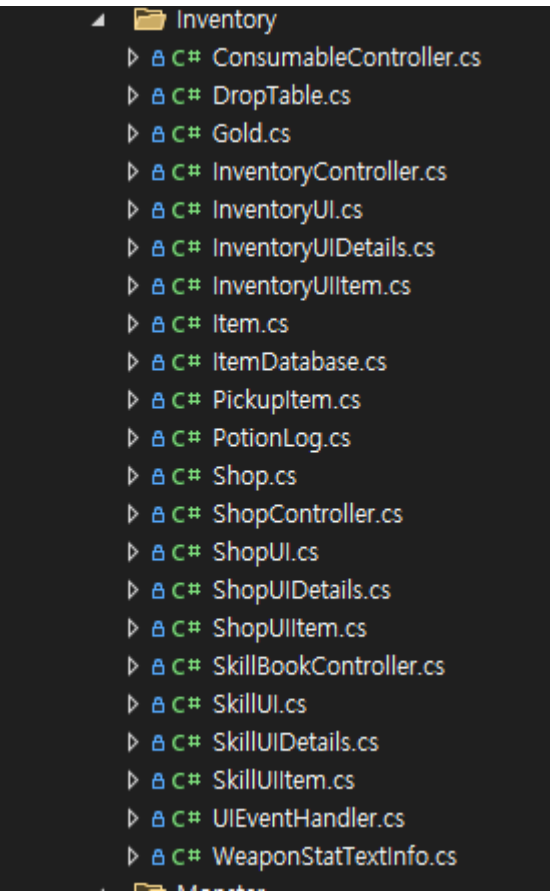
낮과 밤 기능, 물속 수영모드 시스템 구현,캐릭터 초월시에(레벨1000이상 도달) 걷기->비행 가능 모드로 전환



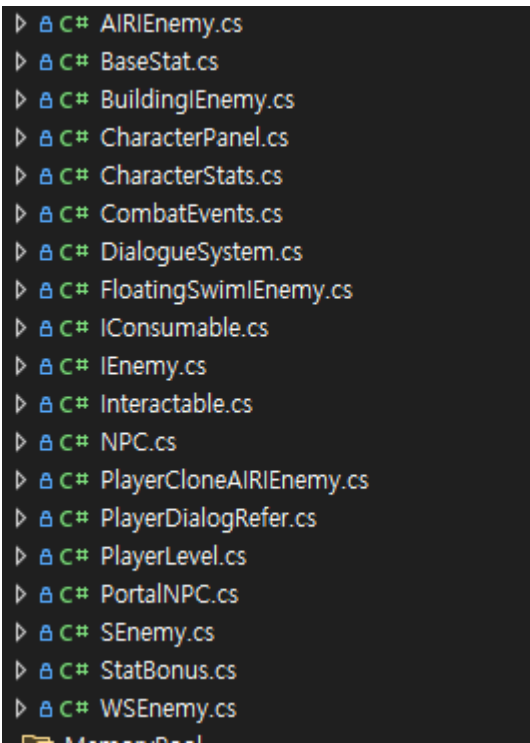
대기&추적형 AI시스템 구현
몬스터의 행동패턴에 따라서 크게
1.순찰형 2.대기&추적형 으로 나뉘어 있다. 몬스터의 지상군,바다군,공중군 모두 존재한다. Area에 따라서 행동양식이 조금씩 달라진다.

육상형,물속형,하늘형,순찰형,대기&추적형 AI시스템
오브젝트풀링기반 Enemy시스템, 적 가건물 시스템 구현.
적 가건물 파괴시에 몬스터 주기적 생산 종료 구현.
게임매니저,사운드관리,미니맵,유저 Combat 처리

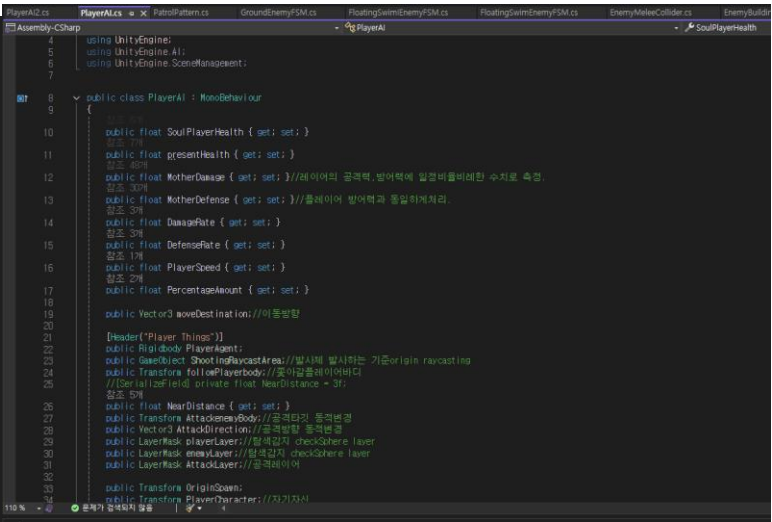
중규모 RPG 시스템 사용코드



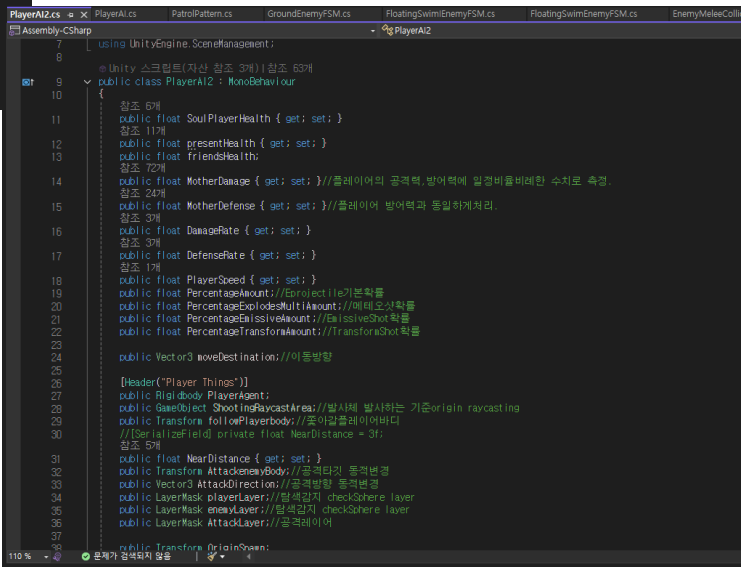
몬스터 아이템 드랍 테이블구현, 아이템소비 구현,골드,인벤토리 시스템,아이템시스템,아이템데이터베이스,아이템 픽업,포션,상점,상점 컨트롤러,스킬북 ui, 스킬상세,레벨에 따른 스킬 학습 관련 기능구현



캐릭터스탯,캐릭터패널,대화 다이아로그시스템,NPC시스템,레벨시스템,포탈맵이동 등의 RPG 관련 시스템

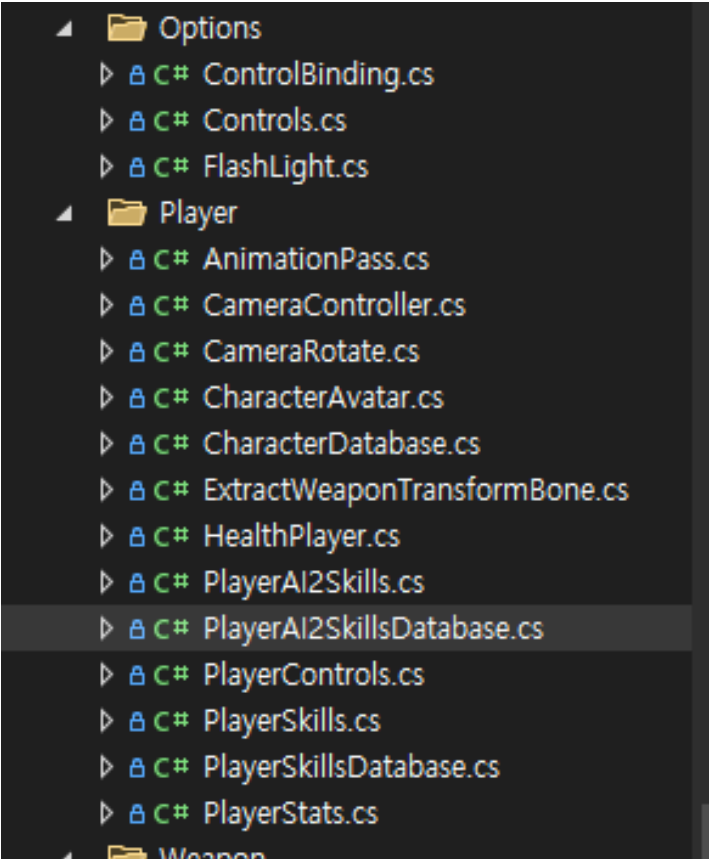


PlayerAI는 캐릭터별 캐릭터의 상징과도 같은 정령캐릭터가 항상 캐릭터 주변을 맴돌며,캐릭터와 같이 전투한다. 캐릭터 레벨에 따라서 정령의 수와 종류 등이 변경된다.

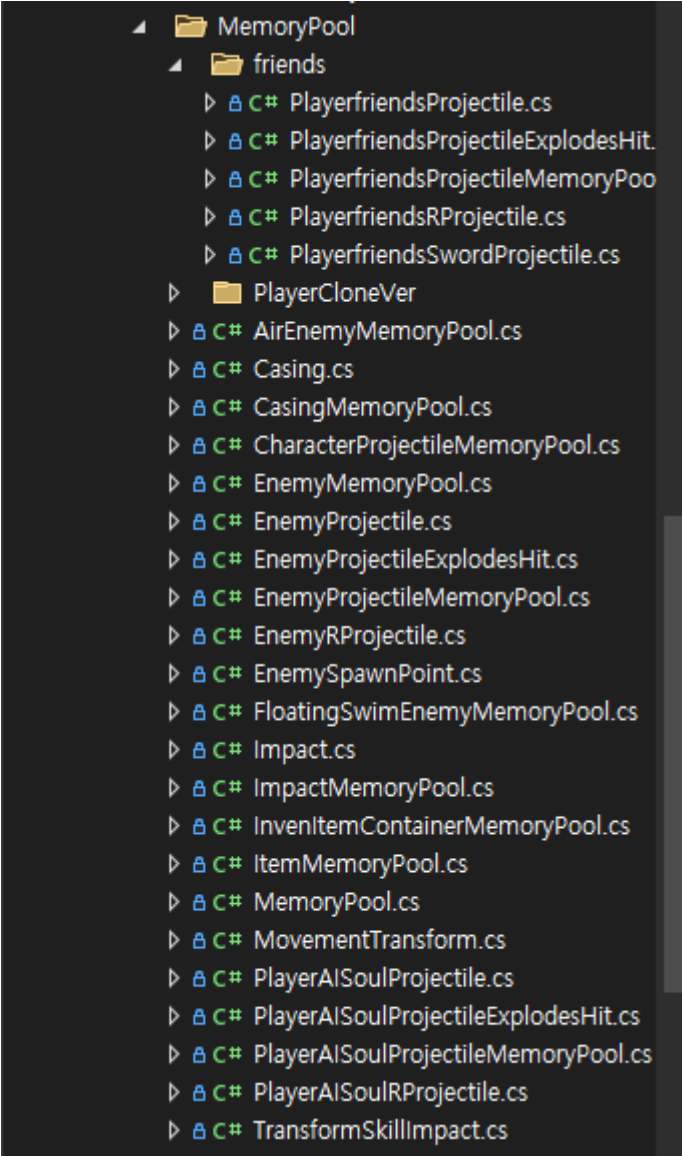


PlayerAI2는 캐릭터의 동료 캐릭터로서 플레이어와 같이 전투하며 같이 레벨업하며,플레이어의 레벨상태에 따라서 동료AI캐릭터 또한 레벨업하며, 랜덤하게 FSM 형태로 스킬을 알아서 자동으로 쓰며 플레이어와 전투한다.

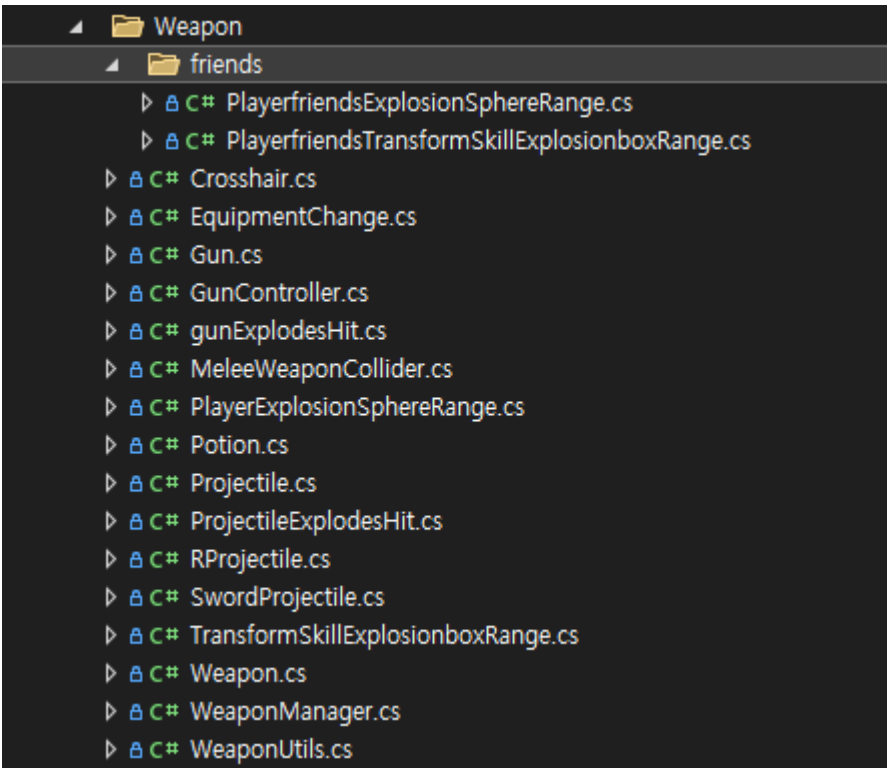
중규모 RPG 시스템 사용코드



플레이어 카메라,캐릭터아바타,캐릭터유,동료AI캐릭터스킬트리,플레이어조작,플레이어스킬트리,플레이어 스탯 관련 처리



동료AI캐릭터 발사 ProjectilesType별 메모리풀링,공중형적,바다형적,지상형적 메모리풀,캐릭터 발사체메모리풀,적발사체메모리풀,발사체피격이펙트효과처리,임팩트메모리풀,인벤토리&아이템 메모리풀,메모리풀 슈퍼클래스,플레이어정령 발사체 메모리풀 등



동료AI캐릭터 범위스킬 피격데미지 처리,프렐이어 장비관련제어,포션,플레이어 발사체&발사체피격 데미지처리,무기클래스,무기매니저,무기유틸 클래스 등











THANK

YOU