

GAME CLIENT

2025

허재성
Heo Jae Seong

허재성 Heo Jae Seong



수상

- 게임 공학과 졸업작품 우수작 선정 (REVENGER) 2023.07
 - 한국공학대학교 주최

학력

- 한국공학대학교 졸업 2024.02
 - 게임 공학과 4년제 학사
- 평택 한광고등학교 졸업 2018.02

경력

- (주) 라이브몰로 2024.07 ~ 2025.09
 - Esper
 - Unity 컨텐츠 개발 및 WebGL빌드
 - DucoTown
 - Addressable 업로드 및 다운로드
 - 플랫폼별 모바일 빌드 및 최적화
 - 실시간 모션 캡쳐
 - 하우스 및 아바타 커스터마이징
 - Flutter-Unity 통신 작업

기술

- C++
- C#
- Unity (Unity 6)
- DirectX
- Xcode - Swift
- Github, Gitea, GitLens
- Figma
- Notion
- Jira
- GCP, Firebase

수강 과목

- C / C++
- STL
- 자료구조
- 운영체제
- 선형대수학
- 게임수학
- OpenGL
- DirectX12
- Unity
- 네트워크 게임 프로그래밍
- 쉐이더 프로그래밍

활동

- 게임공학과 학술 동아리 WARP 멘토 2022.12~2023.12
- 게임공학과 부학생회장 역임 2023
- 게임공학과 학생회 대외부 차장 역임 2019

목차

01 DucoTown

02 PentaShield

MISSING
ASSET
HUNTER

04 REVENGER

05 HUNVERSE

06 WALLZ

07 OTHERS

1. DucoTown

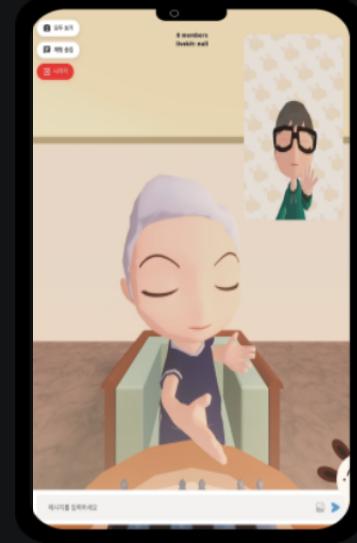


회사명	(주)라이브몰로
프로젝트명	듀코타운
장르	소셜 3D 아바타 메타버스
역할	<p>유니티 클라이언트 컨텐츠 개발 - 에셋 파일관리 관리 및 런타임 로드 - 아바타 및 하우스 품 커스터마이징 - 실시간 모션 캡쳐 - 모바일 리소스 최적화 및 성능체크</p>
플랫폼	IOS , Android
개발 기간	2024.12 ~ 2025.09.30 (회사 내부 사정으로 인한 개발 중단)
환경	<p>- Flutter, Unity6+,vs2022, vsCode, Xcode, Android Studio, Firebase, WebRtc, MediaPipe - C# 9.0, Dart, Swift, Kotlin</p>
참고 링크	 LINK

DUCO(듀코타운)는 Flutter와 Unity를 결합한 하이브리드 모바일 앱으로, 사용자가 3D 아바타를 커스터마이징하고 가상 공간에서 친구들과 실시간으로 소통할 수 있는 소셜 메타버스 플랫폼입니다.

사용자는 자신만의 가상 집을 꾸미고, 친구들을 초대하여 함께 시간을 보내거나, 랜덤 매칭을 통해 새로운 사람들과 만날 수 있습니다.

1. DucoTown



< 실시간 모션 캡쳐 >

실시간 라이브캠 촬영과 MediaPipe Holistic로 사람의 형태를 인식합니다.

T-Pose 캐싱과 부모-자식 분 관계를 고려한 Quaternion 계산으로 각 관절의 회전을 계산하고, ConcurrentQueue로 비동기 콜백 데이터를 메인 스레드로 전달합니다.

계산된 회전값을 아바타에 적용하고, **Fusion**의 NetworkTransform으로 Transform 회전값을 동기화합니다.

< 모바일 작업 >

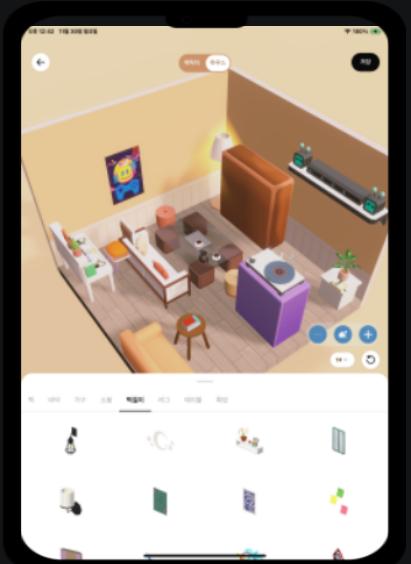
> 모바일 환경에 맞추어 유니티 빌드세팅을 진행하였습니다.

- 플랫폼 별 텍스처 압축 포맷 설정
- Stripping 레벨 및 백엔드(IL2CPP) 조정
- Toon Shader 모바일 설정
- **플랫폼 별 컨텐츠 다운로드 작업** (Addressables + Firebase Storage)

> iOS 포팅

- Flutter-Unity 통신을 위한 프로파티추가 (UnityAppController.h)
- Unity-Flutter 메시지 전달을 위한 C 메소드 추가 (UnityAppController.mm)
- Unity 메시지 핸들러 설정 (AppDelegate.swift)

> 모바일에서 생기는 프레임 저하가 발생하면 **Deep Profile**를 통해 부하가 걸리는 시점을 추적하였으며, 앱에 대한 전반적인 로깅이나 에러는 Android Studio, XCode를 통해서 확인하였습니다.



< 하우스 및 캐릭터 커스터마이징 >

Flutter UI에서 선택한 아이템을 유니티의 객체로 교체 및 생성하는 커스터 마이징 작업을 담당하여 진행하였습니다.

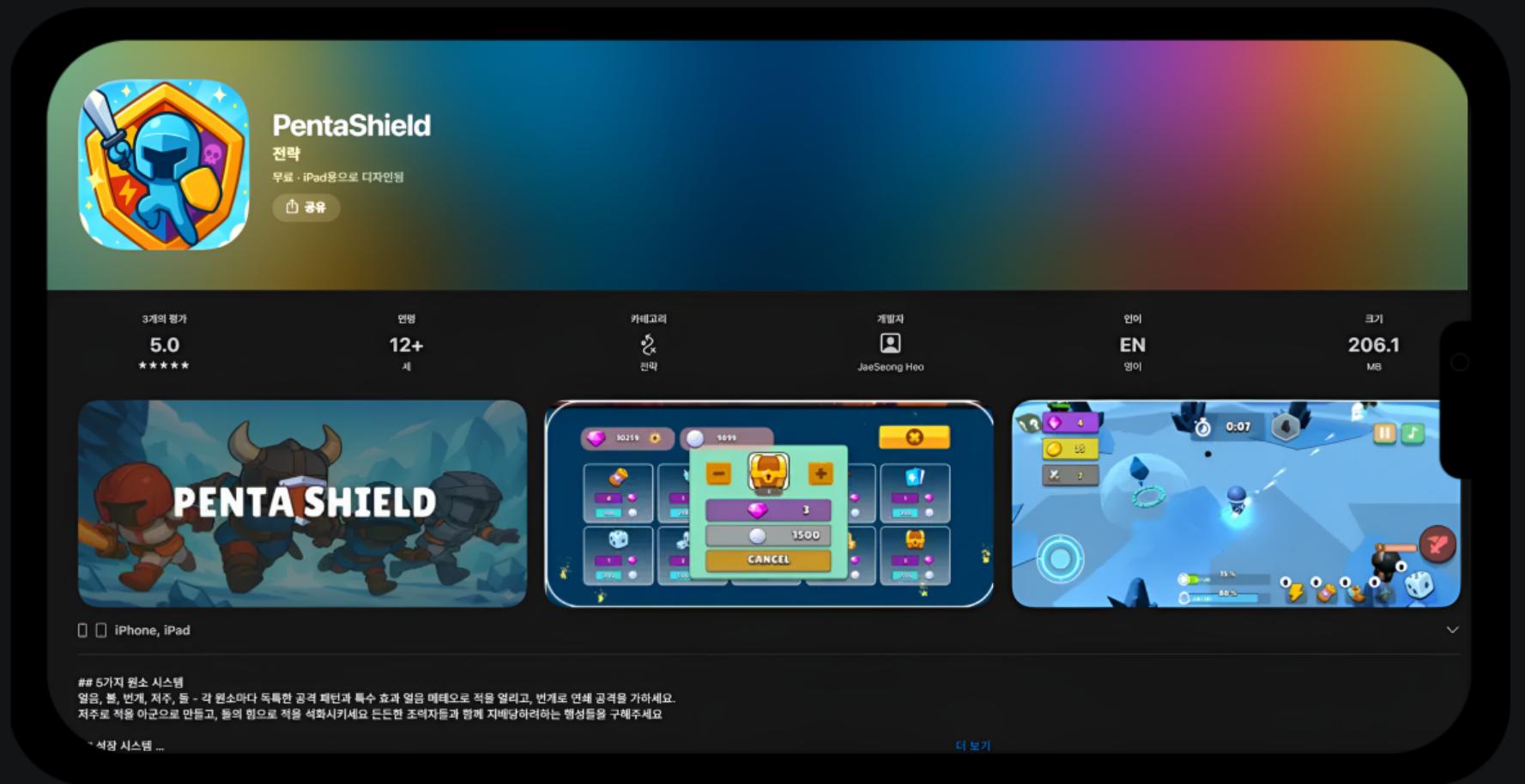
선택한 가구나 파츠 정보를 서버로부터 아이템 키를 받아와 Unity로 전달받고, 해당 위치에 배치하거나 아이템을 적용해 화면에 반영합니다.

아바타의 경우 본 구조에 바인딩하고, MaterialPropertyBlock로 스타일을 적용합니다. 여러 파츠는 병렬로 로드해 성능을 최적화합니다.

하우스는 격자(그리드) 기반으로 구성되며, 각 칸에 가구를 배치 및 수정, 삭제, 초기화 작업을 하였습니다.

또한 아트팀과의 커뮤니케이션을 위해, 모든 프리셋을 적용할 수 있도록 Tool Scene을 만들고 공유하도록 하였습니다.

2. PentaShield



PentaShield는 3D 쿼터뷰 액션 디펜스 모바일 게임입니다.

플레이어는 캐릭터를 직접 조작하며, 둘려오는 적들로 부터 플레이어와 수정체를 방어하며 최종 웨이브까지 도달하여 보스를 처치해야 합니다.

단순히 적을 처치하는 것을 넘어, 제한된 아이템을 언제 사용할지 판단하는 전략적 요소와 캐릭터를 성장시키는 주어진 스테이지를 클리어하는 재미가 결합된 게임입니다.

개발 인원	2인
프로젝트명	PentaShield
장르	3D 쿼터뷰 액션 디펜스
담당 작업	<ul style="list-style-type: none">- 전체 기획 및 디자인- iOS, Android 빌드 포팅 및 스토어 운영- 에셋 파이프라인 관리 및 버전 관리- 아이템 구매, 뽑기 및 출석 보상 구현- 전투 아이템 능력 및 효과 구현- Apple 계정 연동- 전투 및 라운드 시스템 설계 및 구현- 게임 리소스 (UI/UX, Vfx, Sfx) 디자인 및 삽화작성
플랫폼	iOS, Android
환경	<ul style="list-style-type: none">- Unity 2022.3.62f, vs2022, Xcode, Firebase, GCP, Github,- Android Studio, Apple Dev, Google Play Console- C# 9.0, Swift, C++
개발 기간	2025.02 ~ 2025.11 (10개월 - 라이브 서비스 중)
참고 링크	 Link Link Link Link

2. PentaShield

부팅 및 리소스 관리



Firebase Storage 기반 Addressables 다운로드/검증 파이프라인에서 구버전 캐시가 남거나 불필요한 전체 다운로드가 발생하는 이슈가 있어,

Storage REST API로 파일 목록을 조회한 뒤 정렬된 파일명 기준 MD5 버전을 생성하고 version.txt와 비교해 변경 시 로컬 캐시를 자동 삭제 후 재다운로드하도록 구성했습니다.

또한 Addressables.InternalIdTransformFunc로 persistentDataPath로 변환해 로드 실패를 방지했고, 다운로드 전후 로컬 MD5와 서버 해시를 비교/재검증하여 불일치 시에만 다운로드되도록 했습니다.

다운로드는 ConcurrentQueue로 최대 3개 병렬 처리하고, 검증 완료 파일은 HashSet에 저장해 동일 세션 중복 검증을 제거했으며, 검증 실패 시 파일 삭제 후 예외 처리로 자동 재다운로드되도록 했습니다.

마지막으로 catalog_*.json에서 버전을 추출해 앱 버전과 비교하고, 카탈로그 버전이 더 높으면 업데이트 알림 후 진행을 차단하도록 처리했습니다.

Jenkins와 같은 CI/CD 인프라는 서비스 규모 확장 시 도입을 검토할 예정입니다.



출석 보상



14일 사이클 연속 출석 보상에서 단말 시간 오차/조작으로 출석이 어긋나거나 중복 지급되는 이슈가 있어, 출석 판정을 UTC 날짜 + Firestore 서버 타임스탬프 기준으로 처리하고 마지막 출석일과 비교해 중복을 차단했습니다.

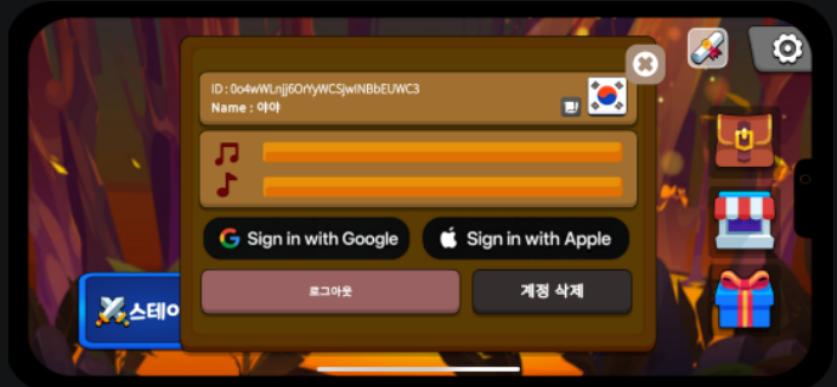
또한 보상 테이블 변경 시 앱 업데이트가 필요한 운영 이슈를 해결하기 위해 보상을 Realtime Database로 분리해 실시간 수정 가능하게 했고, 사이클 ID·버전 비교로 자동 동기화되도록 구성했습니다.

네트워크/로그인 불가 상황은 안내 UI로 처리하고, Firebase 호출 실패로 보상 지급이 중단되는 이슈는 기본 보상 대체 + 로컬 캐시 후 정상화 시 동기화, 저장 실패 시 룰백으로 무결성을 보장했습니다.



2. PentaShield

소셜 로그인



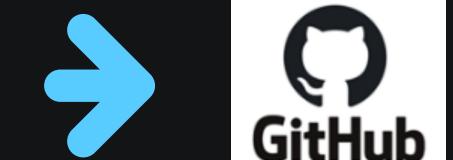
AppleAuth로 받은 iOS 네이티브 토큰을 Firebase OAuth로 넘기고, Google Sign-In도 같은 방식으로 ID 토큰을 Auth에 연결했습니다.

연속 로그인·로그아웃 꼬임은 Firebase Auth 초기화 후에만 로그인 루틴을 돌리고, Unity PlayerLoop Update에서 AppleAuth 콜백을 직렬 큐로 처리해 중복을 차단했으며, Auth 상태 변경 이벤트를 최종 커밋으로 UI/세션을 동기화해 해결했습니다.

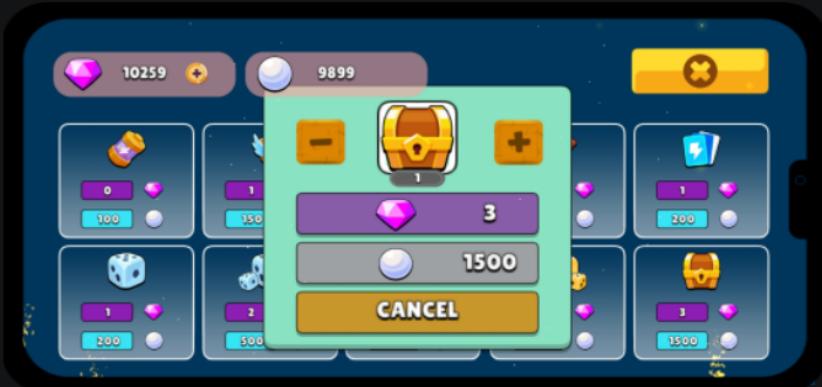
로그아웃은 로그인된 경우에만 버튼과 로직을 열고, 수행 시 Google Sign-In 세션까지 함께 정리합니다.

게스트가 있으면 우선 익명 계정에 Apple/Google 자격증명을 링크하고, 이미 사용 중인 계정 오류 시 기존 계정으로 로그인해 복구합니다.

로그인 성공 후 Firestore users/{uid}의 displayName을 읽어 UI에 즉시 반영하고, 계정 삭제 시 Firestore와 Firebase Auth를 함께 지웁니다.



상점 시스템



인게임 재화(Eli, Stone)를 통한 아이템 상점의 아이템은 Eli와 Stone으로 구매할 수 있으며, Stone으로 Eli 교환도 가능합니다.

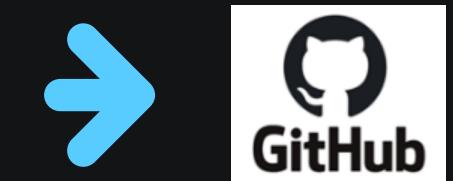
초기에는 결제 버튼 클릭 시 재화를 차감하고 서버에 SaveImportantAsync로 동기화하는 과정에서 응답을 기다리느라 UI 반영이 지연되어, 내 보관함 UI와 실제 보유 재화/아이템 상태가 잠시 어긋나는 문제가 있었습니다.

이를 해결하기 위해, 결제 시 보유 재화를 먼저 검증한 뒤 충분하면 즉시 로컬에서 재화를 차감하고 아이템을 지급한 다음, ItemData.OnItemCountChanged 이벤트를 통해 보관함 UI를 바로 갱신하도록 흐름을 재구성했습니다.

이후 비동기적으로 서버 동기화를 수행하되, 실패 시에만 롤백 로직을 태우는 방식으로 동작을 안정화했습니다.

구매 수량은 기본 수량 단위로 조절하며, 수량에 따라 가격을 자동 계산하고, 구매 성공 시 포뮬선 비행 이펙트를 재생해 시각적 피드백도 제공합니다.

특정 재화나 아이템에 종속되지 않도록 공통 베이스 클래스를 상속받아 구현하여, 추후 다른 재화 타입과 보상 구조로도 쉽게 확장할 수 있도록 했습니다.



2. PentaShield

각 아이템 별 능력 및 효과 구현



각 아이템은 스테이지 내에서 **독립적인 쿨타임**을 가지며, 사용 시 즉시 효과가 발동되도록 설계했습니다. 임시 아이템과 영구 아이템을 구분하여 관리하며, 피버타임 사용 시 플레이어 주위에 5가지 원소를 생성하여 적을 공격하도록 구현했습니다.



게임 내 아이템 시스템은 **서포트 아이템**과 **광역 스킬**로 구성됩니다. 서포트 아이템은 **헤이스트**(미동 속도 증가), **물약**(체력 회복), **피버타임**(임시 무적 및 원소 소환), **시간정지**(적 행동 정지) 4종이며, 광역 스킬은 **5가지 원소 특성**(얼음, 불, 돌, 번개, 저주)에 맞는 메테오, 레이디얼, 크래시 등으로 구현했습니다.

얼음 메테오는 사선 낙하하며 적 최대 체력의 30% 데미지를 주고, 불 메테오는 즉시 30% 데미지와 함께 그라운드 충돌 시 도트 데미지 영역을 생성합니다. 번개는 적 머리 위에서 수직 발사체를 생성하여 공격하고, 돌은 가속 낙하하며 석화 효과를 적용합니다. 저주는 범위 내 적들에게 저주를 적용하여 저주받은 적들이 서로 공격하도록 만듭니다.

추가로 레벨업을 하게되어 **성장아이템박스**를 맵에 랜덤한 위치에 스폰시켜, 획득 시 개수가 차감되지 않는 랜덤한 서포트 아이템을 사용할 수 있도록 하였습니다.

황금 랜덤 큐브를 통해 이 아이템들을 무작위로 뽑을 수 있도록 하였으며, **재화 뽑기 상자와 아이템 뽑기 카드**를 추가하여 재화 소비를 유도하고 랜덤하게 뽑혔다는 시각적 표현도 추가했습니다.

2. PentaShield

전투 및 라운드 시스템



발사체는 유도/직선, 가속/등속을 모두 지원하며 Thunder는 털김 연쇄로 데미지가 줄고, Flame은 도트 영역을 남기며, Stone은 넉백을, Curse는 적을 아군 공격 상태로 만듭니다.
유도탄은 탐지 반경 내 최단거리 적을 추적하고, 가속 모드에서는 속도가 점진적으로 상승하며

자동 타겟팅으로 가장 가까운 적을 향해 회전·발사하되 일부만 유도탄으로 배정해 섞어 사용하며, Elemental은 Guard 주위를 공전하다가 반경 내 적이 있을 때만 공격하고 동시에 활성 발사체 수를 제한해 관리합니다.

적은 Player/Guard를 랜덤으로 추적하며 사망 시 성장요소인 경험치와 코인을 무작위로 떨어뜨립니다.

Curse 이후 타겟이 사라져 미아가 되던 이슈가 있었기에 타겟 상실 시 즉시 재탐색·해제하도록 고쳐 연속 전투를 안정화했고, 많은 발사체로 인한 GC 부담은 풀링으로 완화했으며, GlobalItem 처리와 알림이 뒤엉키던 문제는 “상태 갱신 → 효과 계산 → 노티 발행” 순으로 고정하고, 동일 이벤트에 쿨타임을 두어 중복 트리거를 막았습니다.



라운드 시스템은 30초 시간 제한 기반으로 동작하며, 라운드 완료 시 게임을 일시정지하고 업그레이드 화면을 표시합니다.
업그레이드 선택 후 카운트다운을 거쳐 다음 라운드로 진행하며, 다음 라운드 데이터가 없으면 게임 클리어로 처리합니다.

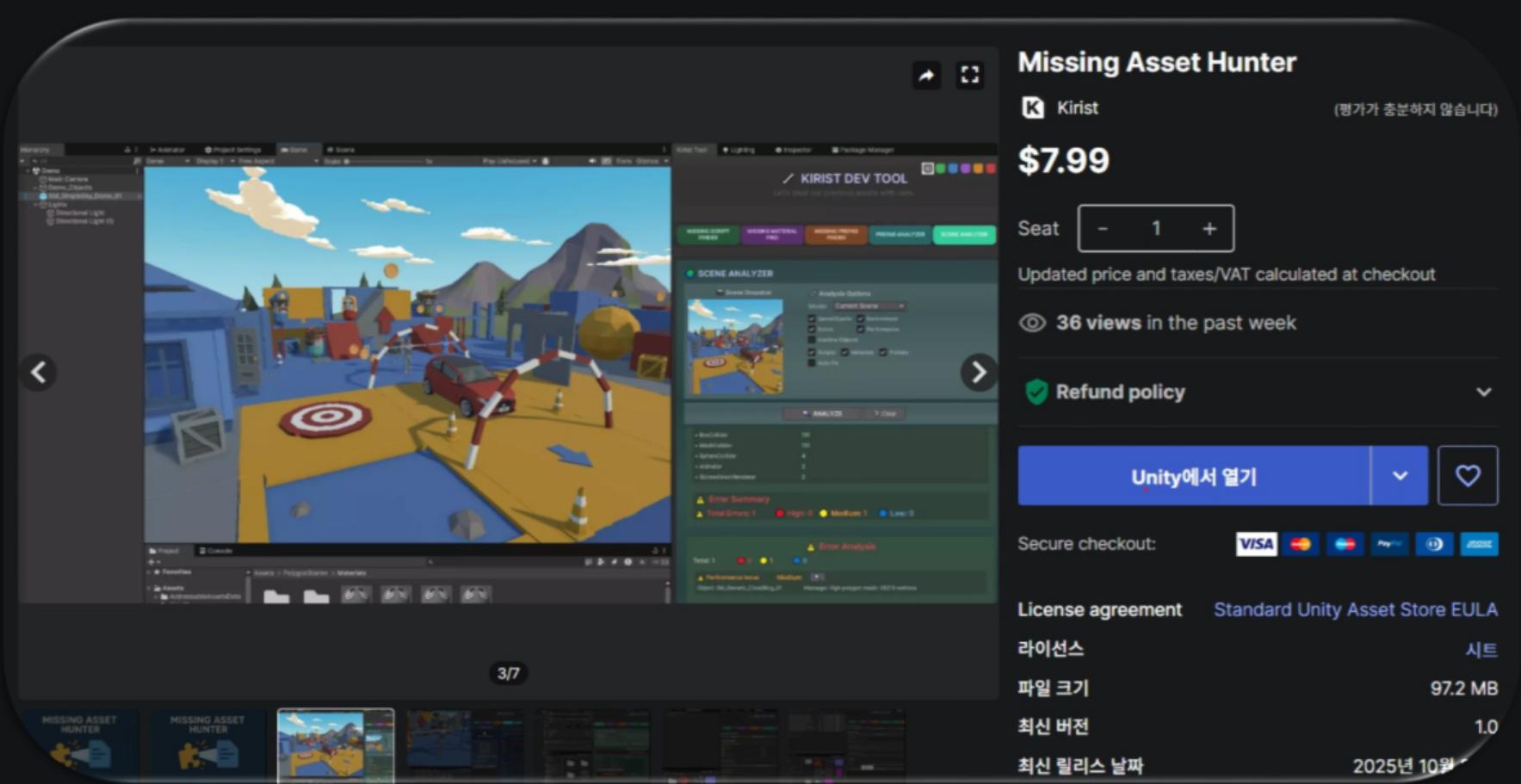
적 스판 시스템은, 라운드별 스판 테이블을 기반으로 타이머 기반 스판을 관리합니다. 라운드 종료 시 모든 스판된 적을 정리하고 스판 오퍼레이션을 초기화합니다.

게임 오버 또는 클리어 시 현재 라운드, 스테이지명, 점수를 StageData로 저장하고 최고 라운드 기록을 갱신하며, 익명 사용자는 로컬에만 저장하고 로그인 사용자는 Firebase에 동기화합니다.

스코어 획득은 시간 경과에 따른 **스코어 + 적 처치 시 부여된 고유한 스코어** 값을 반영하였으며, 플레이어가 사망하게 되어 스테이지가 종료되거나 클리어하게되면 Score에 비례해서 Stone을 일부 지급합니다.

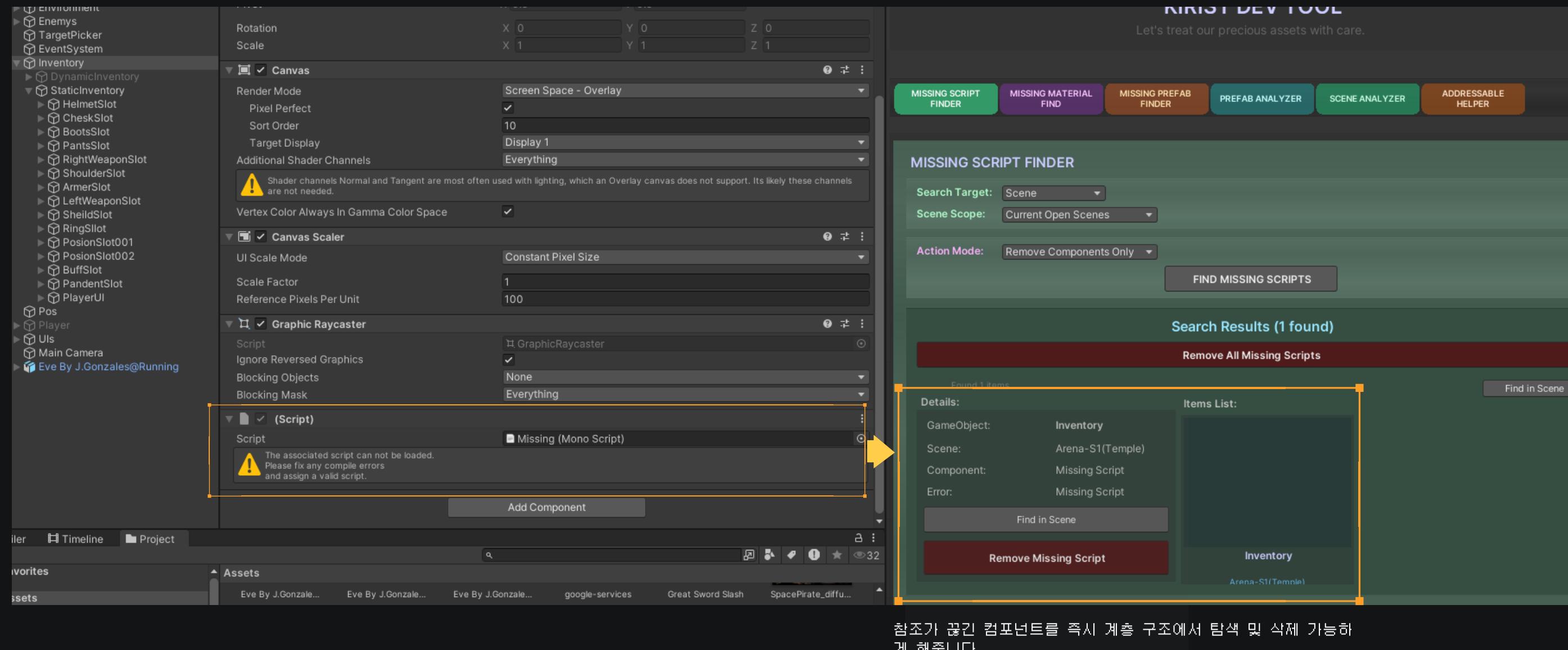


3. MISSING ASSET HUNTER



개발 인원	1인
프로젝트명	Missing Asset Hunter
장 르	개발 유틸리티
세부 기능	<ul style="list-style-type: none">- 참조가 끊어진 Script, Prefab를 탐색- 불안정한 상태의 Material(Shader)들을 Scene, Prefab에서 탐색하여 사용자에게 편리한 수정작업 서포트- Prefab 및 Scene 분석- Addressable Group 생성 도움 툴
환경	<ul style="list-style-type: none">- Unity6000.1.4f1, .NET 4.7.1, Unity Asset Store- C# 9.0
개발 기간	2025.09 ~ 2025.10
참고 링크	 LINK  GitHub LINK

3. MISSING ASSET HUNTER



참조 애러 스크립트 탐색

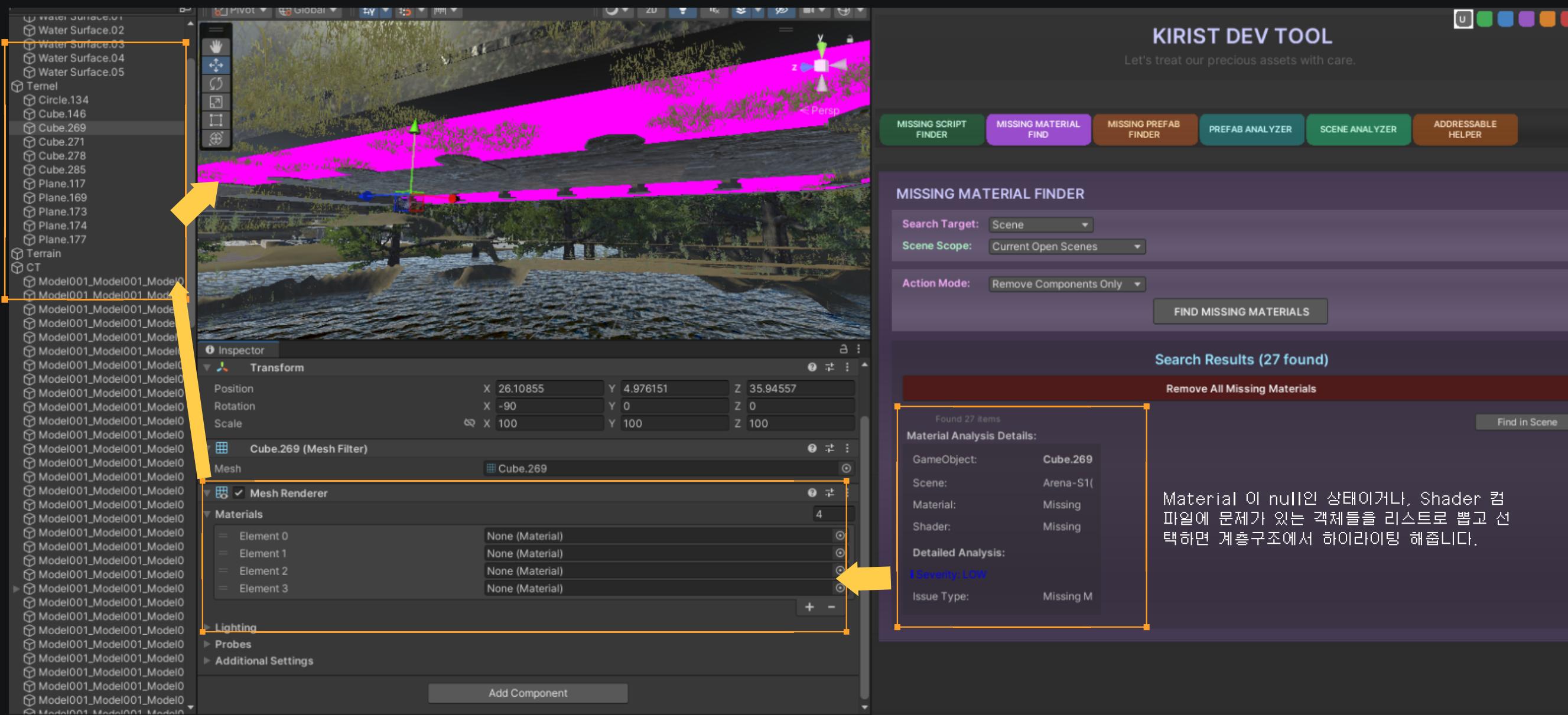
탐색이 시작되면 최상위 루트 GameObject 부터 시작해, 연결된 모든 컴포넌트 배열을 가져와 재귀적으로 자식 노드까지 순차 순회합니다.

이때 유니티의 직렬화(Serialization) 특성을 활용하여, 컴포넌트 솔루션은 존재하지만 실제 활동된 객체가 null인 상태를 감지합니다. 이를 통해 스크립트 파일이 삭제되었거나 클래스명이 변경되어 참조 연결이 끊긴 'Fake Null' 상태를 애러로 규정합니다.

특히 프리팹 검사 시에는 파일 자체뿐만 아니라 Scene에 배치된 인스턴스와의 연결 관계까지 추적하여, 원본은 정상이더라도 빈 데이터 모바일과 과정에서 깨진 케이스까지 검증해냅니다.



3. MISSING ASSET HUNTER



불안정한 쉐이더 탐색

탐색이 시작되면 최상위 루트 GameObject부터 재귀적(Recursive)으로 순회하며, 렌더러 컴포넌트에 할당된 머티리얼에 대해 **IsErrorHandler** 메서드를 호출하여 정밀 검증을 수행합니다.

우선 가장 치명적인 마젠타(Magenta) 이슈를 잡기 위해, 셰이더 이름(shader.name)이 Hidden/InternalErrorShader 같은 유니티 내부 셰이더와 일치하는지를 식별합니다.

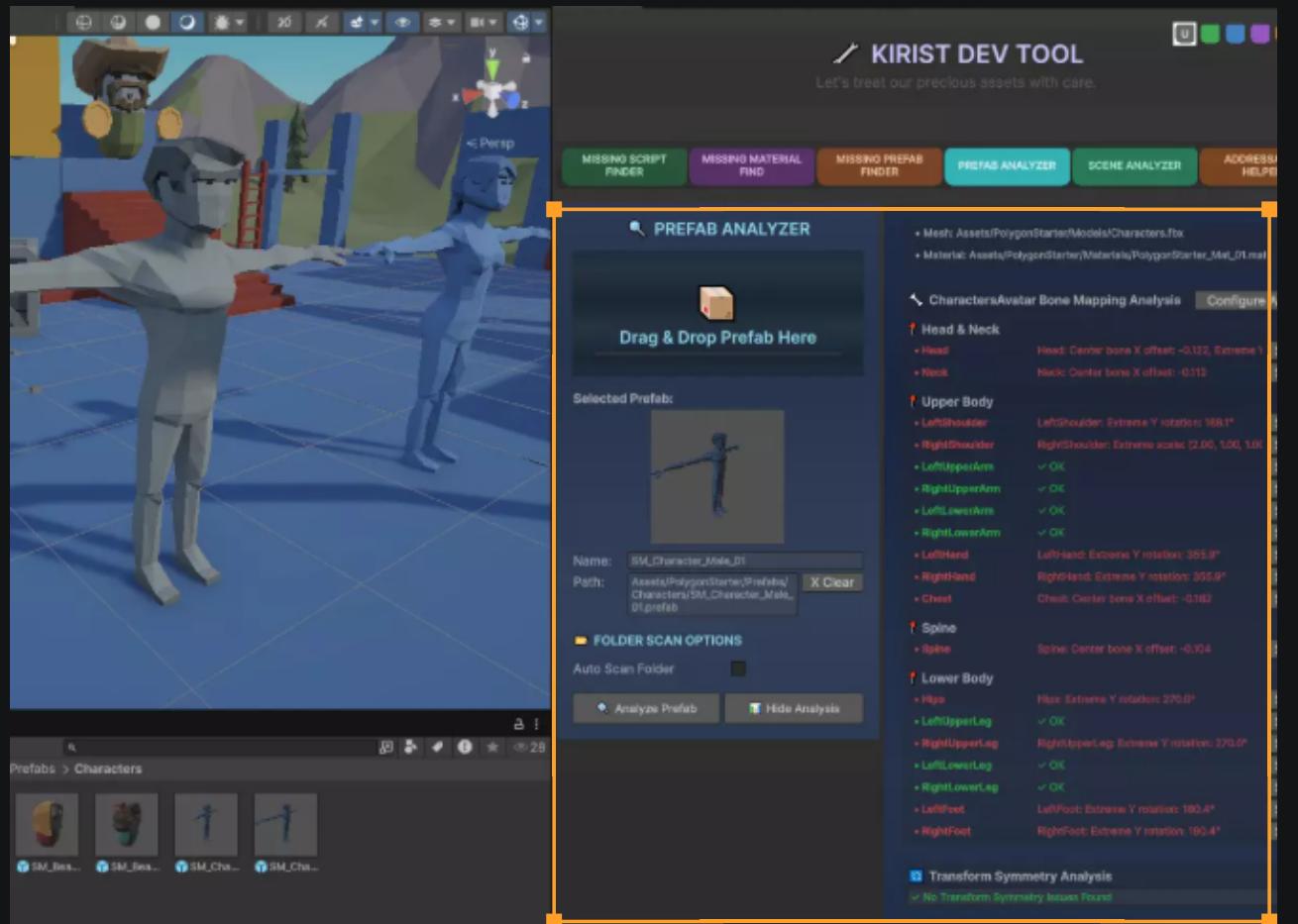
동시에 GraphicsSettings.currentRenderPipeline으로 현재 파이프라인(URP/HDRP/Built-in)을 확인하고, 셰이더 접두사(Universal Render Pipeline/, Standard 등)를 비교하여 호환되지 않는 셜이더(Pipeline Mismatch)를 찾아냅니다.

Custom 셜이더의 경우 더 깊은 단계의 검증을 수행합니다. **shader.isSupported** 및 **passCount**를 통해 컴파일 실패 여부를 1차적으로 확인하고, 소스 코드를 정규 표현식(Regex)으로 파싱하여 #pragma 선언 함수와 실제 구현부의 불일치 같은 구조적 결함을 검증합니다.



3. MISSING ASSET HUNTER

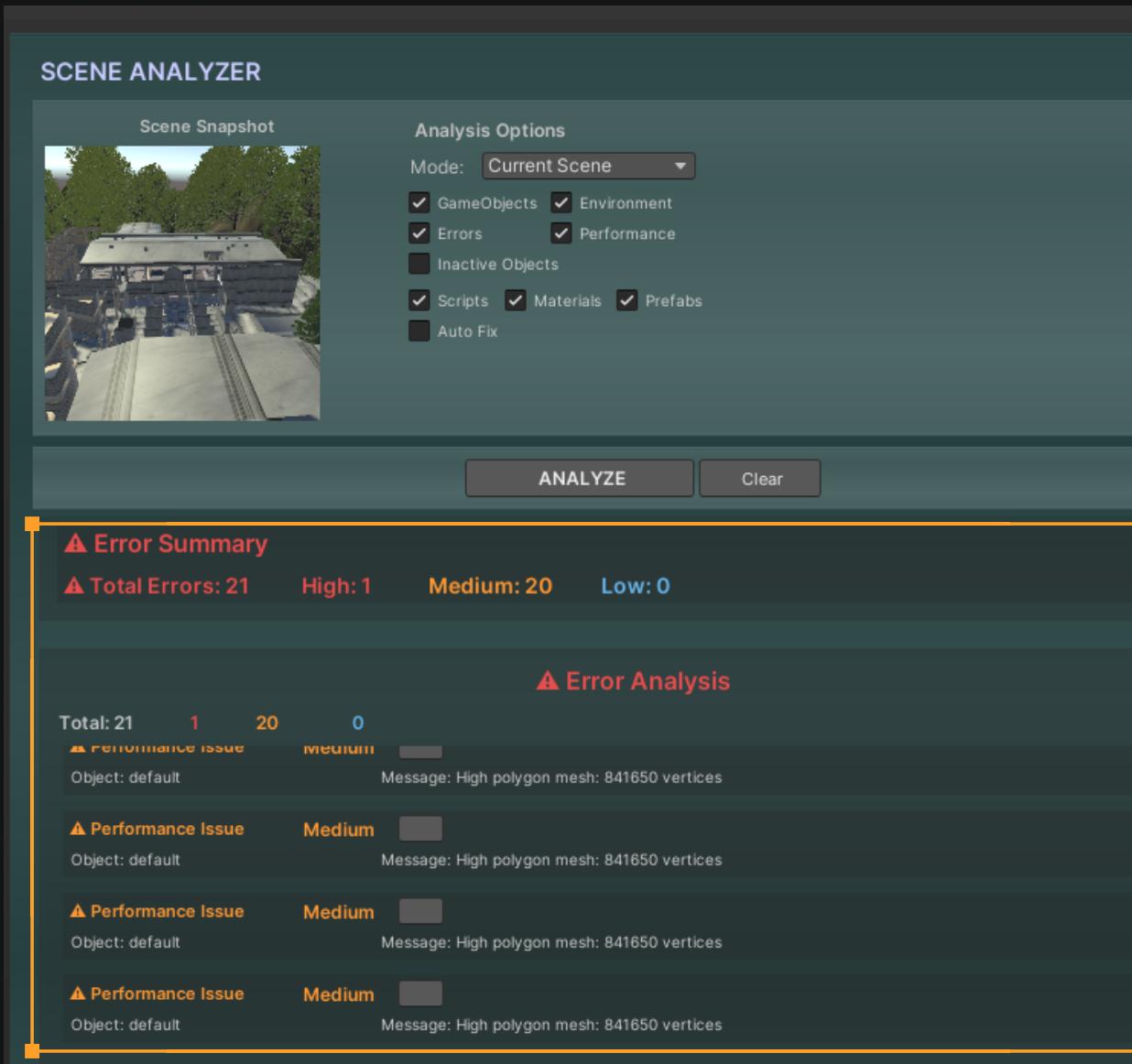
Prefab 분석



프리팹을 재귀적으로 순회하여 Missing Scripts, Materials, Prefabs를 검사하고, 세이더 소스 코드를 정규표현식으로 파싱하여 #pragma 선언과 구현부 불일치, Pass 블록 누락 등 구조적 결함을 검증합니다.

SerializedObject를 활용해 컴포넌트 참조 관계를 추적하고, AssetDatabase.GetDependencies로 씬 및 다른 프리팹에서의 사용처를 분석합니다. Animator 뼈 매핑과 Transform 대칭성도 검증합니다.

Scene 분석



씬의 모든 GameObject를 재귀적으로 순회하여 Missing Scripts, Materials, Prefabs, Error Shader를 검사하고, Lighting, Camera, Terrain, PostProcessing 등 환경 요소를 분석합니다.

EditorApplication.update를 활용해 GameObjects → Components → Environment → Errors → Snapshot 순서로 단계별 비동기 분석을 수행하며, RenderTexture를 통해 씬 스크린샷을 캡처하고 고폴리곤 메시 등 성능 이슈를 감지합니다.

4. REVENGER [졸업작품]

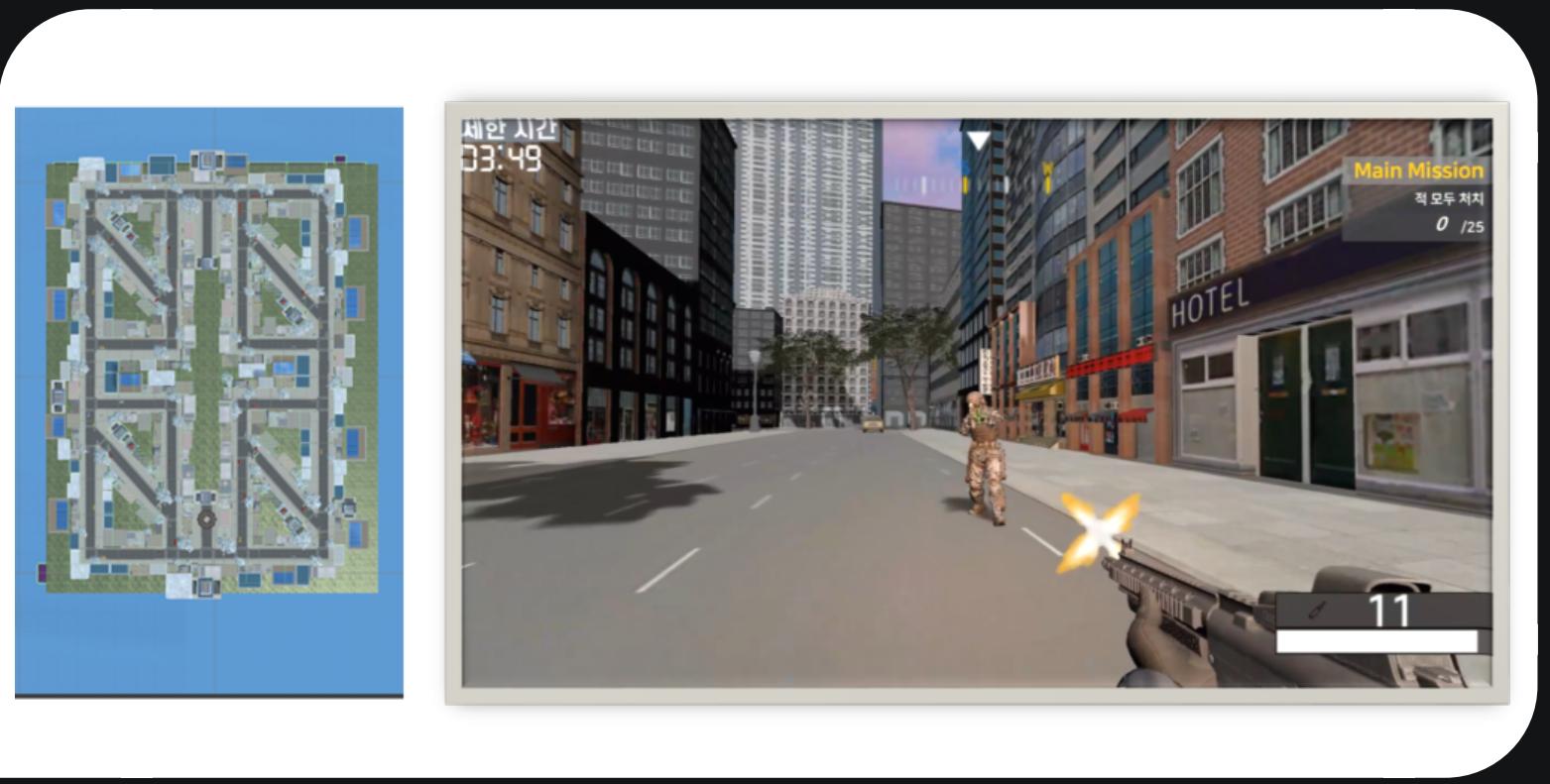


개발 인원	3인 (서버 1, 서버 및 기획 1, 클라 1)
프로젝트명	REVENGER
장르	3D FPS 멀티 협동 대전 게임
역할	클라이언트 전 과정 - 스테이지 로드 - 게임 컨텐츠 구현 - 사운드 - UI/UX - 그림자 - 애니메이션 - 객체 효과 및 연출 - 서버 패킷 상호작용 - 빌드 작업
환경	- Visual Studio 2022, DirectX 12, FMOD, Git hub, IOCP, DXTex - C++ 11
개발 기간	2022.08 ~ 2023.07
참고 링크	 LINK LINK LINK

REVENGER는 지상 플레이어 2명, 공중 플레이어 1명으로 구성된, 총 3명의 플레이어가 다같이 모여 방에 입장하고, 적군 지역에서 활보하는 적들을 정해진 시간 내에 소탕하고 전장을 점령하여 승리를 취하는 것을 목표로 하는 게임입니다.

4. REVENGER [졸업작품]

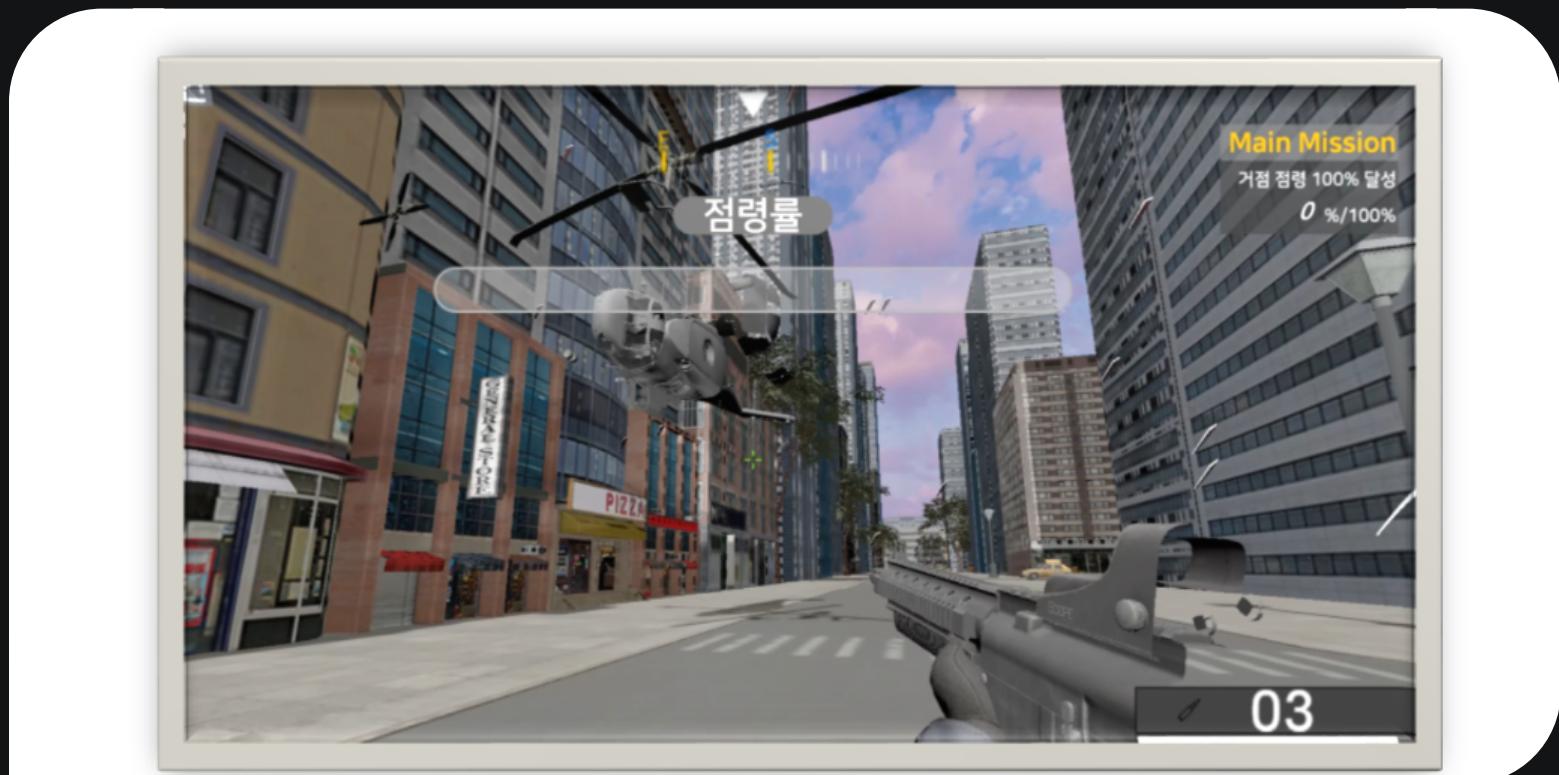
게임 맵 제작 및 로드



로드하는 과정을 최소화하기 위해, Unity에서 건물들과 도로, 나무를 배치시켜 하나의 객체로 Export하여, bin 파일로 변환 후 맵을 로드 Albedo, Normal, Metallic 3가지 Shader 정보를 담고 있습니다.

나무의 경우 맵의 구조에 맞게 따로 배치 후 추출하여, 알파블렌딩을 적용하여, 나뭇잎과 줄기를 표현하였습니다.

객체 간의 충돌



충돌 과정 중, NPC와 플레이어의 충돌, 파편들과 NPC, 플레이어들 간의 충돌 작업을 해주었습니다.

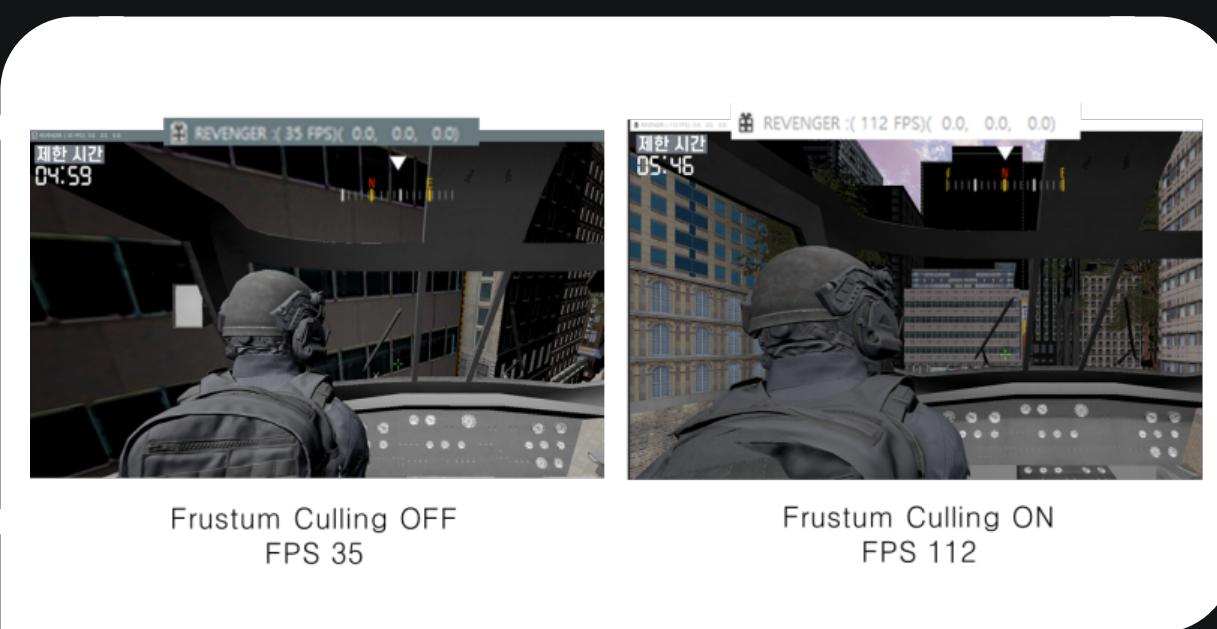
헬리콥터 격추 시 분해된 부품들과 피해 처리를 해주었습니다. 같은 방식으로 여부를 판단했고, 파편들이 모든 플레이어들에게 같은 방향과 무게로 동기화되어, 파편들과 충돌을 해주어야 했기에, 동일한 방향과 중력을 적용해주었습니다.

모든 클라이언트에서 파편의 이동 경로와 충돌 처리 결과가 동일하게 보이도록 구현했습니다.

각 클라이언트에서 충돌이 감지되면 해당 정보를 서버로 전달하고, 서버는 이를 모든 클라이언트에게 브로드캐스트하여 충돌 상태가 일관되게 동기화되도록 했습니다.

4. REVENGER [졸업작품]

절두체 컬링



절두체 영역을 계산하여, 객체의 위치와 크기를 고려하여 매 프레임마다 렌더링되는 객체의 BoundingBox가 절두체의 평면과 교차하는지 검사해줍니다.

절두체 컬링 과정을 통해, 절두체 영역에 속하지 않는 객체를 렌더링 대상에서 제외시켜, FrameRate의 성능을 향상시켜주었습니다.

그림자 렌더링



광원이 활성화 될 때, 광원의 유형에 따라 투영 행렬을 하고, 광원의 위치와 방향에 따른 뷰 행렬을 계산합니다.

계산된 투영 행렬과 뷰 행렬을 사용해서, 깊이 렌더링 카메라가 어떻게 3D 공간을 볼 것인지 카메라 설정을 업데이트합니다.

멀티 렌더 타겟 뷰를 생성하고, 깊이-스텐실 버퍼를 생성 및 초기화하고, 셰이더 변수를 생성합니다.

조명을 계산 시, PCF 방식을 사용하여 그림자 팩터를 계산하여 그림자를 더 부드럽게 만들게 됩니다.

```
for (int i = 0; i < MAX_LIGHTS; i++) {
    if (gLights[i].m_bEnable) {
        float fShadowFactor = 1.0f;
        #ifdef _WITH_PCF_FILTERING
        if (bShadow) fShadowFactor = Compute3x3ShadowFactor(uvs[i].xy / uvs[i].ww, uvs[i].z / uvs[i].w, i);
        #else
        if (bShadow) fShadowFactor = gtxtDepthTextures[i].SampleCmpLevelZero(gssComparisonPCFShadow,
            uvs[i].xy / uvs[i].ww, uvs[i].z / uvs[i].w).r;
        #endif
        if(gLights[i].m_nType == DIRECTIONAL_LIGHT) {
            cColor += DirectionalLight(i, vNormal, vToCamera) * fShadowFactor;
        }
    }
}
```

5x5의 커널 크기를 사용하여, 계단 현상을 더 줄이고자 했지만, 연산의 양이 너무 많고, 육안 상 큰 차이가 없었기에 3x3 커널의 크기를 사용하여 그림자를 필터링 하였습니다.

4. REVENGER [졸업작품]

그림자 - 동적 객체



```
Compute3x3ShadowFactor(float2 uv, float fDepth, uint nIndex) {
    float fPercentLit =
        gtxtDepthTextures[nIndex].SampleCmpLevelZero(gssComparisonPCFShadow,uv, fDepth).r;
    return(fPercentLit / 9.0f);
}

for (int i = 0; i < MAX_LIGHTS; i++) {
    if (gLights[i].m_bEnable) {
        float fShadowFactor = 1.0f;
#ifdef _WITH_PCF_FILTERING
        if (bShadow) fShadowFactor = Compute3x3ShadowFactor(uvs[i].xy / uvs[i].ww, uvs[i].z / uvs[i].w, i);
#else
        if (bShadow) fShadowFactor = gtxtDepthTextures[i].SampleCmpLevelZero(gssComparisonPCFShadow,
            uvs[i].xy / uvs[i].ww, uvs[i].z / uvs[i].w).r;
#endif
        if(gLights[i].m_nType == DIRECTIONAL_LIGHT) {
            cColor += DirectionalLight(i, vNormal, vToCamera) * fShadowFactor;
        }
    }
}
```

그림자에 해당하는 객체들을 관리하는 클래스에 담아둔 후,
광원으로 부터 그 객체들의 깊이 값을 구하여 그림자 맵을 생성합니다.

객체들의 계층구조 모델을 로드하는 과정에서 리깅을 수행하는 객체인지
아닌지를 상수버퍼 값으로 넘깁니다.

그림자 맵을 생성하는 정점 쉐이더에서 리깅을 수행할 경우, 정점을 본 행렬로 변환하게 됩니다.

4. REVENGER [졸업작품]

애니메이션 리깅 및 동기화



Mixamo에 있는 동작들을 Unity에서 리깅을 수행할 하나의 Prefab에 어떤 동작들을 수행할 것인지 스크립트를 통해 필요한 동작들을 넣어준 후 본의 정보를 가진 bin파일을 추출하게 됩니다.

모델을 로드 할 때, 애니메이션 세트, 키 프레임들의 변환 정보를 읽어와 애니메이션을 설정하게 됩니다.

애니메이션을 사용할 게임 객체의 생성자에서는, 트랙의 수에 따라 몇번 째 애니메이션 세트를 사용할 것인지, 설정해주고, 해당 애니메이션 트랙에서 사운드 콜백 함수를 통해 어떤 사운드를 재생시킬지 정의해줍니다.

걷기, 장전 등 Idle 상태로 부터, 객체들의 상태가 활성화 되어 동작을 수행할 땐, 각 트랙들의 활성 상태를 변경해주어 해당 상태에서의 동작을 실시합니다.

서버에서 모든 플레이어들에게 리깅 정보가 동기화되어야 하기 때문에, 서버와 주고 받는 상태들을 따져서, 어떤 객체가 어떤 동작들을 수행해야하는지, 함수들을 호출하게 됩니다.

```
// 3. 만약 죽어있는 상태면 캐릭터 조작이 불가능하게 막아야합니다.  
if(players_info[my_id].m_ingame_state == PL_ST_DEAD) MyPlayerDieMotion();  
  
MyPlayerDieMotion()  
{  
    If (m_ingame_role == R_RIFLE) {  
        ((CHumanPlayer*)((MainGameScene*)m_pScene)->m_pPlayer)->m_bDieState = true;  
        ((CHumanPlayer*)((MainGameScene*)m_pScene)->m_pPlayer)->DyingMotion();  
    }  
}
```

5. HUNTVRSE [제작 중]



개발 인원	2인 개발 (서버1, 클라1)
프로젝트명	Huntverse
장르	2D 흥스컬 액션 RPG
담당 역할	<p>메인 클라이언트</p> <ul style="list-style-type: none"> - 프로젝트 기획 및 디자인 - 게임 컨텐츠 개발 - UI/UX - 리소스 제작 (+AD) - 서버-클라이언트 데이터 동기화
플랫폼	PC (Steam)
개발 기간	2025.11.20 ~ 제작 중
환경	<ul style="list-style-type: none"> - Unity6+, vs2022, protobuf Steamworks.NET - NanoBanana pro, Ludo.ai - c# 9.0 , c++ 20, mysql
참고 링크	 LINK GitHub LINK

잠든 주인공이 알 수 없는 평행 세계에서 전사로 깨어나는 멀티플레이어 2D 액션 RPG를 제작 중입니다.

전투와 탐험을 통해 현실로 돌아갈 실마리를 찾아가는 플레이 중심의 게임으로, 빠른 전투 템포, 다양한 스킬 조합, 캐릭터별 개별 서사를 제공합니다.

플레이어는 이 세계의 비밀을 파헤치며 각 캐릭터의 결말을 직접 선택하게 됩니다.

6. WALLZ

[Unity 20th GameJam]



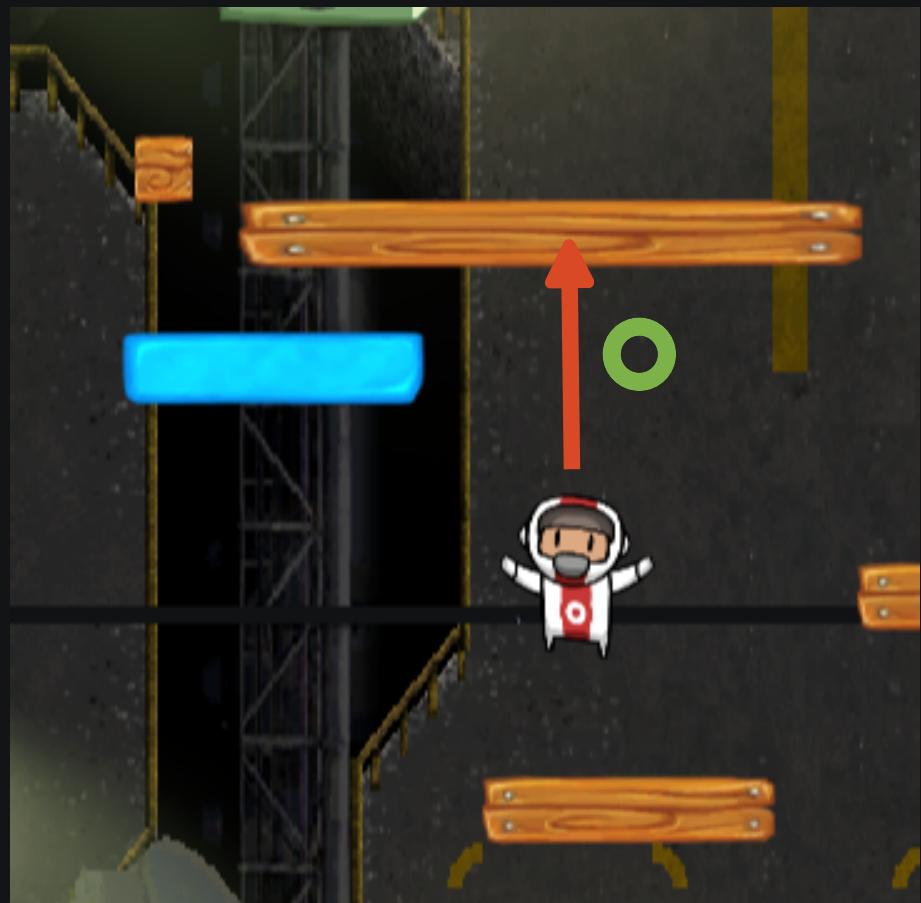
개발 인원	1인
프로젝트명	WALLZ
장르	2D 촘촘스크롤 플랫포머
플랫폼	WebGL
환경	- Unity6+, itch.io - C# 9.0
개발 기간	2025.11.10 ~ 2025.11.12
참고 링크	 LINK  LINK

무한한 시간 동안, 위를 향해 우주 정거장을 탈출하는 2D 촘촘스크롤 플랫포머 장르로 48시간 이내에 만들어야하는 Unity 20주년 게임잼 응모작입니다.

적들의 반격과 함정들을 이겨내면서 최대한 위로 많이 올라가는 간단하게 즐기기 좋은 게임입니다.

6. WALLZ

[Unity 20th GameJam]



발판은 One-Way Platform 방식으로 구성하였습니다.

PlatformEffector2D는 정직 플랫폼에 최적화 되어있다고 생각했고 발판의 타입도 Tilt,Fake,Normal 타입이 있어, 접촉 법선(Contact Normal)기반 충돌 판정을 사용하였습니다.

```
if (c.normal.y > 0.5f)
{
    isGrounded = true;
    // 발판 위에서만 지면으로 인식
}
```

위에서 밟기: Contact Normal (0, 1) → normal.y = 1.0 > 0.5 → 충돌

아래에서 올라가기: Contact Normal (0, -1) → normal.y = -1.0 < 0.5 → 통과

발사체나 발판,적 객체들은 Pool Size를 지정해두고 재사용하였습니다.

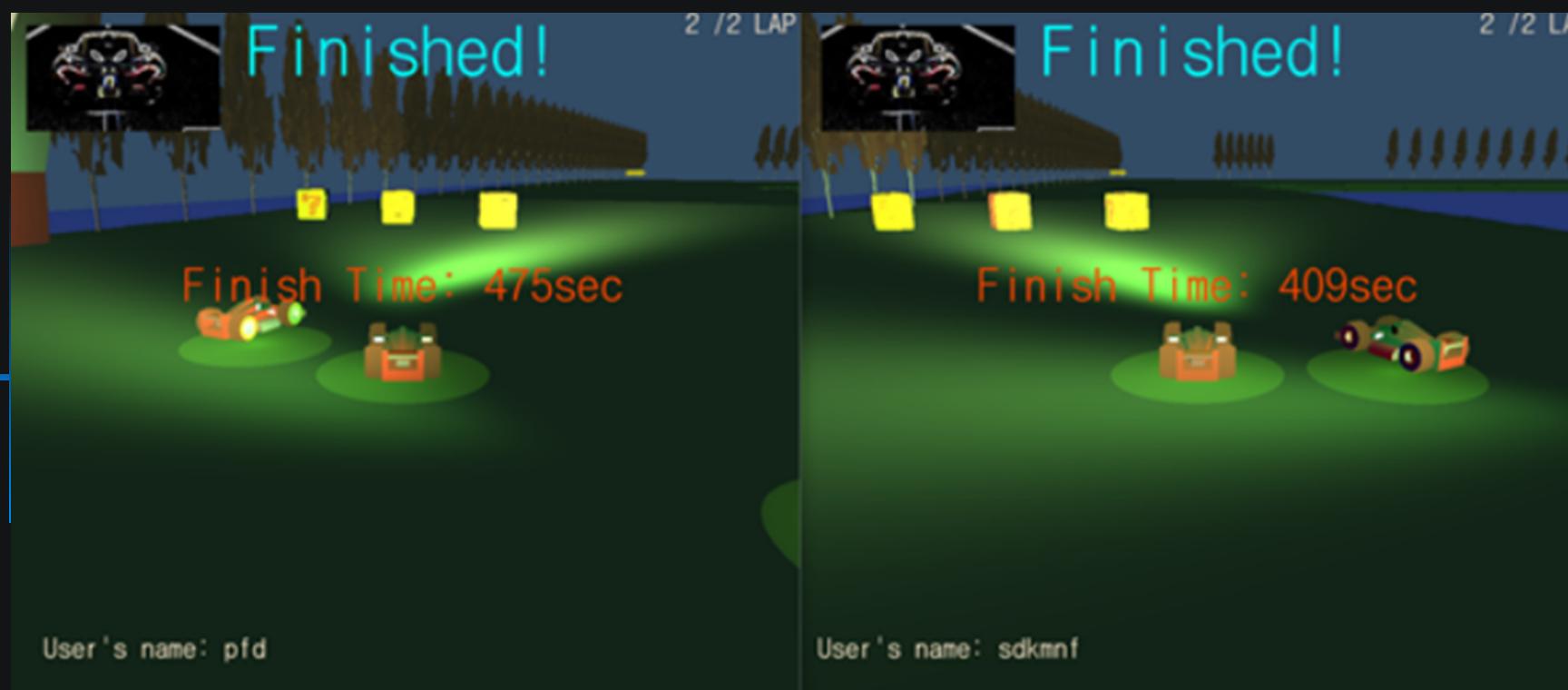
발판의 경우 최상단 발판의 위치에 따라서 겹침을 방지 하도록 하였습니다.

주어진 에셋으로 48시간 이내에 만들었지만 어색한 부분도 있고, 컨텐츠의 범위나 활용도도 부족해서 추후에 보강할 예정입니다.

7. Others

[NetworkGameProgramming]

RacingAttack



3명이 출발선에서 출발하여, 아이템 박스를 통해 아이템을 획득하고, 해당아이템을 사용하여 상대방의 완주를 방해하면서 먼저 3바퀴를 완주하면 승리하는 것을 목표로하는 프로젝트입니다.

개발 인원	3인
프로젝트명	Racing Attack
장 르	3D 레이싱 멀티 게임
담당 작업	- 클라이언트 전체 제작 - 로그인 패킷 송수신 - 총돌 판단 여부와 총돌 연출 시 동기화 작업
환경	- vs2022, Directx12, Github, TCP - C++ 11
개발 기간	2022/10 - 2022/12
참고 링크	 LINK LINK

7. Others

[NetworkGameProgramming]

로그인 - 유저 정보 확인

```
C2LS_LOGIN_PACKET login_pack;
retval = recv(client_sock, (char*) &login_pack,
              sizeof(C2LS_LOGIN_PACKET), MSG_WAITALL);
if (retval == SOCKET_ERROR) {
    err_display("recv()");
    break;
}

std::string recved_name = login_pack.name;
bool name_already_used = false;
for (int i = 0; i < MAX_USER; i++) {
    if (clients[i].getName().compare(recved_name) == 0) {
        name_already_used = true;
        break;
    } // if end
} // for end

LS2C_GAMESTART_PACKET start_pack;
start_pack.type = LS2C_GAMESTART;
if (approval && name_exist && !name_already_used) {
    start_pack.start = START_APPROVAL;
}
else if (!name_exist) {
    start_pack.start = START_DENY_UNKNOWNNAME;
}
else if (!approval) {
    start_pack.start = START_DENY_FULL;
}
else if (name_already_used) {
    start_pack.start = START_DENY_ALREADYUSED;
}
}

If (approval && name_exist) break;
```

클라이언트로부터 받은 이름이 이미 접속중인 다른 유저의 이름이 아닌지 확인합니다

계정이 이미 접속 중 또는 존재하지 않는 이름일 때, 서버상의 유저수가 포화상태일때 각 상태에 맞는 값을 패킷에 담아 클라이언트로 전달하는 과정입니다.

```
// 2. 계정 정보 데이터파일에 존재하는지 확인.
bool name_exist = false;
std::string saved_name;
while (fin >> saved_name) {
    std::cout << "Saved name: " << saved_name << std::endl;
    if (saved_name.compare(recv_name) == 0) {
        std::cout << "계정 [" << login_pack.name << "]이 확인되었습니다."
        << std::endl;
        name_exist = true;
        break;
    }
}
fin.close();
bool approval = true;
if (!name_exist) {
    std::cout << "존재하지 않는 계정입니다. 등록 후 다시 시도해주세요." << std::endl;
}
else if (name_already_used) {
    std::cout << "입력한 계정 [" << login_pack.name << "]은 이미 다른
    플레이어가 사용 중입니다." << std::endl;
}
else {
    // id 할당
    for (int i = 0; i < MAX_USER; i++) {
        if (clients[i].getState() == SESSION_EMPTY) {
            client_id = i;
            clients[client_id].;
            clients[client_id].setId(client_id); setState(SESSION_RUNNING)
            clients[client_id].setName(login_pack.name);
            std::cout << "Clients[" << clients[client_id].getId() << "]'s Name: "
            << clients[client_id].getName() << std::endl;
            break;
        }
    }
}

if (i == MAX_USER - 1 && clients[i].getState() == SESSION_RUNNING) {
    std::cout << "Max Users Exceeded!" << std::endl;
    approval = false;
} // if end
}// for end
} // else end
```

계정 정보가 데이터파일에 존재하는지 확인합니다. 존재하지 않는 계정이라면 새로 등록을 진행해달라고 클라이언트에 메시지를 띄우기 위해 start_pack의 start 메소드(시작 여부)에 START_DENY_UNKNOWNNAME를 입력하고, 이미 접속해있는 유저라면 start_pack의 START_DENY_ALREADYUSED 입력합니다.

7. Others

[NetworkGameProgramming]

충돌여부 판단 (객체 / 맵)

```
Void collisionscheck_Player2ItemBox(int client_id)
{
    for(int i=0 ; i< ITEMBOXNUM ; i++) {
        if( !ItemBoxArray[i].m_visible ) continue;
        ...
        if(ItemBoxArray[i].xoobb.
            Intersects(clients[client_id].xoobb)) {
            EnterCriticalSection(&ItemBoxArray[i].m_cs);
            ItemBoxArray[i].m_pos.y =
                ItemBoxArray[i].m_pos.y - 500;
            ItemBoxArray[i].m_visible = false;
            LeaveCriticalSection(&ItemBoxArray[i].m_cs);

            sendItemBoxUpdatePacket_toAllClient(i);

            EnterCriticalSection(&cs_timer_event);
            setServerEvent(EV_TYPE_REFRESH,5.0f,
            EV_TARGET_ITEMBOX, 0, i, 0, 0);
            LeaveCriticalSection(&cs_timer_event);

            if (clients[client_id].getHowManyItem() < 2) {
                srand(static_cast<unsigned int>(SERVER_TIME) * i);
                int new_item = rand() % 3;

                EnterCriticalSection(&clients[client_id].m_cs);
                clients[client_id].setItemQueue(new_item);
                LeaveCriticalSection(&clients[client_id].m_cs);

            } //for end
        }
    }
}
```

아이템 박스의 변경사항을 모든 클라이언트에게 전달합니다.
충돌한 아이템박스는 5초 후에 초기 상태로 돌아오며, 충돌한 플레이어는 갖고있는 아이템이 2개 미만일때만 새로운 아이템을 획득할 수 있도록 하였습니다.

작동하지않는 미사일, 자신이 생성한 미사일, 피격 모션 좋은 플레이어는 Ignore대상으로 처리했습니다.
제어권과 피격되었을 때 미사일을 삭제하며, 미사일 제거 패킷을 모든 클라이언트들에게 전달합니다.

```
void collisioncheck_Player2Missile(int client_id)
{
    for (int i = 0; i < MissileNum; i++) {
        if (!MissileArray[i].getRunning()) continue;
        if (MissileArray[i].getObjOwner() == client_id) continue;
        if (clients[client_id].getHitMotion()) continue;

        ...
        if (MissileArray[i].xoobb.Intersects(clients[client_id].xoobb)) {

            EnterCriticalSection(&clients[client_id].m_cs);
            clients[client_id].setLoseControl(true);
            clients[client_id].setHitMotion(true);
            MissileArray[i].returnToInitialState();
            LeaveCriticalSection(&clients[client_id].m_cs);

            GS2C_REMOVE_OBJ_PACKET rm_missile_packet;
            rm_missile_packet.type = GS2C_REMOVE_OBJ;
            rm_missile_packet.id = i;
            rm_missile_packet.objtype = OBJ_TYPE_MISSILE;
            for (int j = 0; j < MAX_USER; j++) {

                if (clients[j].getState() == CL_STATE_EMPTY) continue;
                clients[j].sendRemoveObjPacket(rm_missile_packet);
            } // for end

            EnterCriticalSection(&cs_timer_event);
            setServerEvent(EV_TYPE_HIT, HIT_MISSILE_DURATION, EV_TARGET_CLIENTS,
            0,client_id, 0, 0);
            LeaveCriticalSection(&cs_timer_event);

        } // if end
    } //for end
}
```

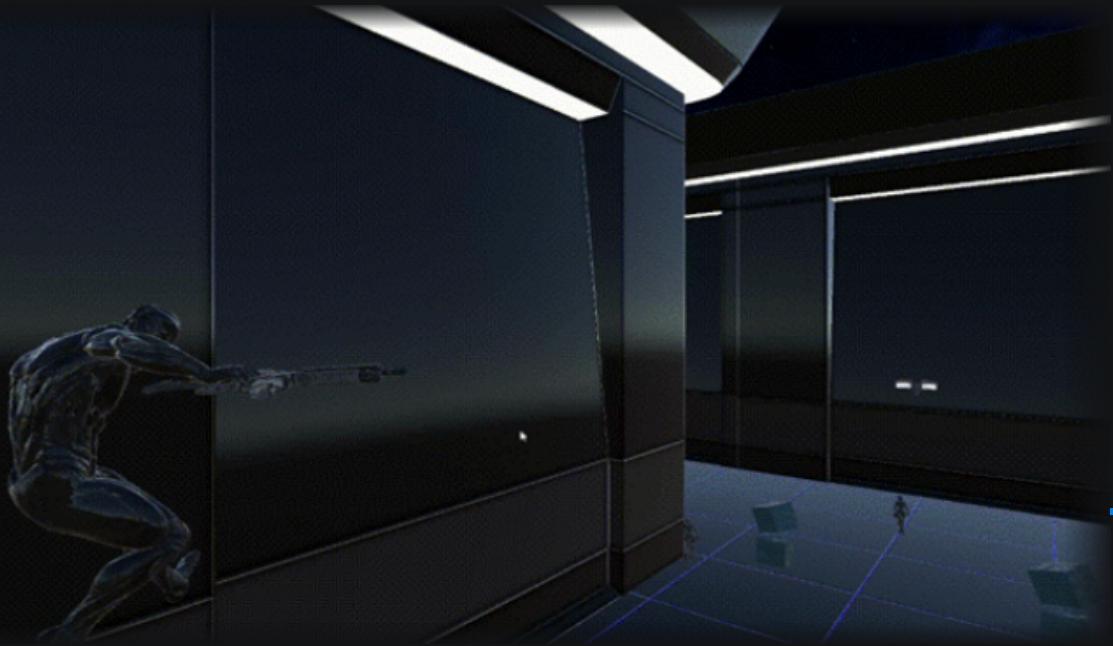
```
void collisioncheck_Player2Map(int client_id)
{
    MyVector3D cur_pos = clients[client_id].getPos();
    if (cur_pos.x > MAP_X_MIN + MAP_COLLISIONCHECK_RANGE
        && cur_pos.x < MAP_X_MAX - MAP_COLLISIONCHECK_RANGE
        && cur_pos.z > MAP_Z_MIN + MAP_COLLISIONCHECK_RANGE
        && cur_pos.z < MAP_Z_MAX - MAP_COLLISIONCHECK_RANGE)
        return;
    if (cur_pos.x < MAP_X_MIN) {
        cur_pos.x = MAP_X_MIN;
        EnterCriticalSection(&clients[client_id].m_cs);
        clients[client_id].setPos(cur_pos);
        LeaveCriticalSection(&clients[client_id].m_cs);
    }
    else if (cur_pos.x > MAP_X_MAX) {
        cur_pos.x = MAP_X_MAX;
        EnterCriticalSection(&clients[client_id].m_cs);
        clients[client_id].setPos(cur_pos);
        LeaveCriticalSection(&clients[client_id].m_cs);
    }
    if (cur_pos.z < MAP_Z_MIN) {
        cur_pos.z = MAP_Z_MIN;
        EnterCriticalSection(&clients[client_id].m_cs);
        clients[client_id].setPos(cur_pos);
        LeaveCriticalSection(&clients[client_id].m_cs);
    }
    else if (cur_pos.z > MAP_Z_MAX) {
        cur_pos.z = MAP_Z_MAX;
        EnterCriticalSection(&clients[client_id].m_cs);
        clients[client_id].setPos(cur_pos);
        LeaveCriticalSection(&clients[client_id].m_cs);
    }
}
```

맵의 끝으로부터 너무 멀리 떨어져 있다면 충돌체크 및 후처리를 진행하지 않으며, 맵의 최대,최소 범위를 넘어가게 되면 넘어가기 전 위치에 머물러있게 됩니다.

Client[]는 전역에 선언되어있기에 메소드에 Write 작업을 할 때 Date Race를 막기 위해 Critical Section으로 보호합니다.

7. Others

[Unity Learn]



개발 인원	1인
프로젝트명	HomanoidWar (Unity 수업 최종 과제물)
환경	vs2022, Unity2022 (URP), C# 9.0
개발 기간	2023/05 ~ 2023/06
제작 목표	한학기 동안 배운 Unity 기능들(애니메이션 블렌딩, 맵 제작, 물리엔진, 총돌)을 활용한 제작물을 산출
참고 링크	 LINK LINK

개발 인원	1인
환경	vs2022, Unity2022 (HDRP)
개발 기간	2023.11.10 ~ 2023.11.21
제작 목표	한학기 동안 배운 Unity 환경시스템을 이용한 자연배경 제작
참고 링크	 LINK

감사합니다