

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
- Unity 20주년 게임잼 참가

목차

01 DucoTown

02 PentaShield

MISSING
ASSET
HUNTER

04 REVENGER

05 HUNVERSE

06 RacingAttack

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 툴 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 본 관계를 고려한 Quaternion 계산으로 각 관절의 회전을 계산하고, ConcurrentQueue로 비동기 콜백 데이터를 메인 스레드로 전달합니다.

계산된 회전값을 아바타에 적용하고, Fusion의 NetworkTransform으로 Transform 회전값을 동기화합니다.

<모바일 작업 >

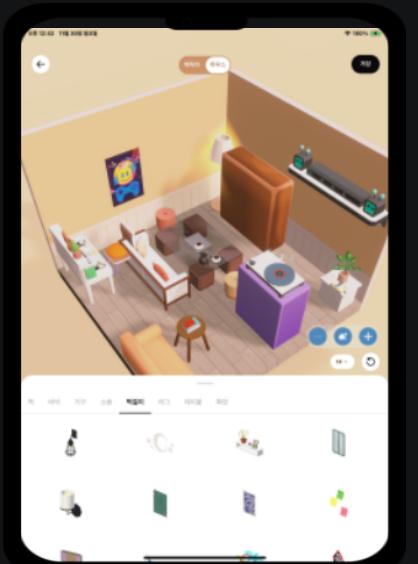
> 모바일 환경에 맞추어 유니티 빌드세팅을 진행하였습니다.

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

> 모바일에서 생기는 프레임 저하가 발생하면 Deep Profile를 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.

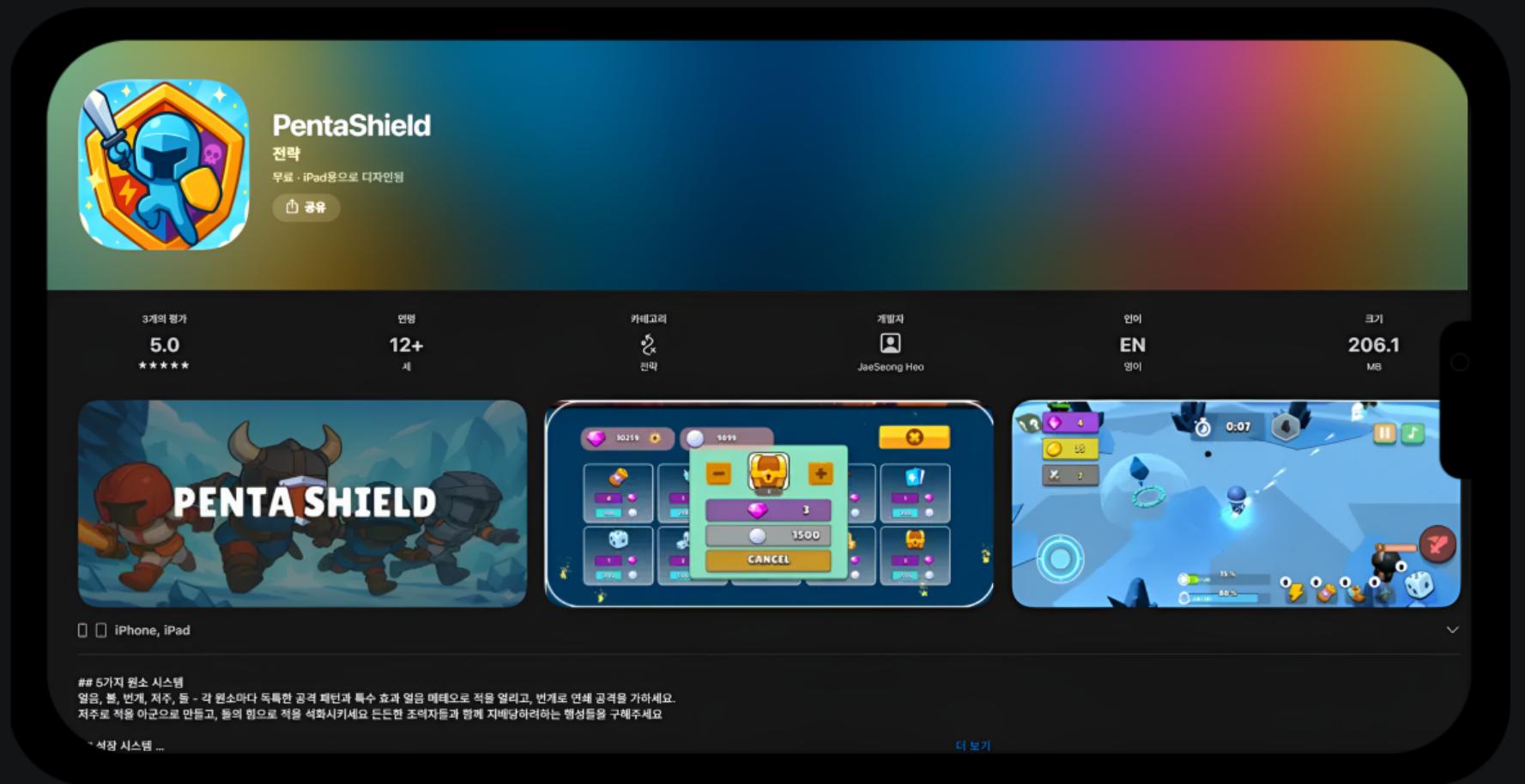
선택한 가구나 패치 정보를 서버로 부터 아이템 키를 받아와 Unity로 전달받고, 해당 위치에 배치하거나 아이템을 적용해 화면에 반영합니다.

아바타의 경우 본 구조에 바인딩하고, MaterialPropertyBlock로 스타일을 적용합니다. 여러 패치는 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, 몰려오는 적들로 부터 플레이어와 수정체를 방어하며 최종 웨이브까지 도달하여 보스를 처치해야합니다.

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

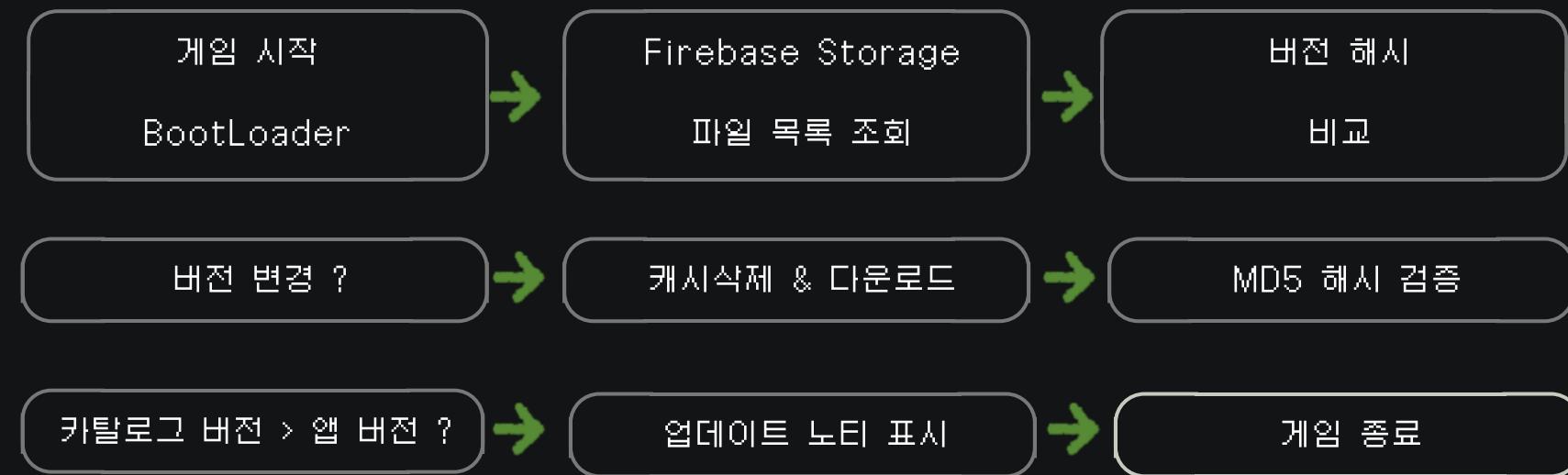
개발 인원	2인
프로젝트명	PentaShield
장 르	3D 쿼터뷰 액션 디펜스
담당 작업	<ul style="list-style-type: none">- 전체 기획 및 디자인- iOS, Android 빌드 포팅 및 스토어 운영- 에셋 파이프라인 관리 및 버전 관리- 아이템 구매, 뽑기 및 출석 보상 구현- 전투 아이템 능력 및 효과 구현- Apple 계정 연동- 전투 및 라운드 시스템 설계 및 구현- 게임 리소스 (UI/UX, Vfx, Sfx) 디자인 및 삽화작성
플랫폼	iOS, Android (정식 출시)
환경	<ul style="list-style-type: 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)
참고 링크	 Link Link Link Link

2. PentaShield

<부팅 및 리소스 관리 >



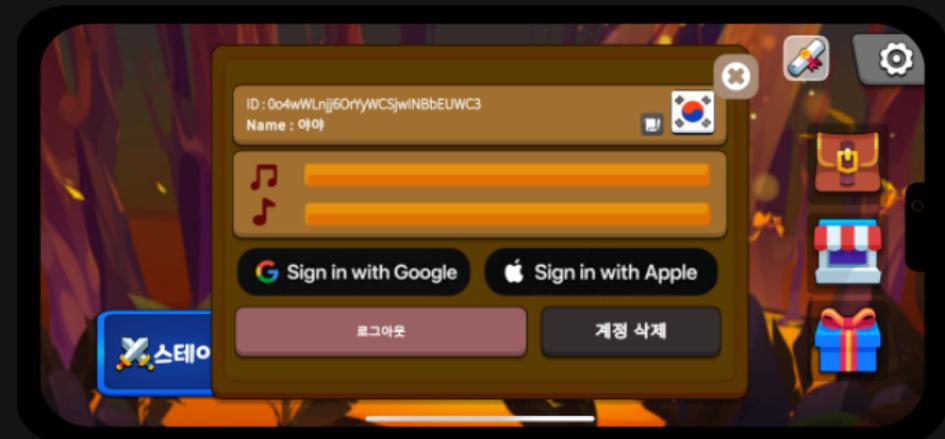
- Firebase Storage에서 다운로드한 Addressable 에셋의 버전이 앱 번들 버전과 불일치할 경우 게임 진행 불가 문제 해결
- MD5 해시 기반 버전 관리 시스템으로 파일 목록 변경 감지 및 캐시 자동 정리 구현
- 다운로드된 각 파일의 MD5 해시 검증을 통한 파일 무결성 보장
- 카탈로그 버전과 앱 버전을 Major.Minor.Patch 형식으로 비교하여 업데이트 필요 시 게임 진행 차단
- 버전 불일치 시 사용자에게 업데이트 알림 표시 후 게임 진행 방지로 크래시 및 데이터 손상 방지



```
// 1. 버전 해시 생성 및 비교
string currentVersion = GenerateVersionHash(response.items);
string lastVersion = LoadLastKnownVersion();
if (currentVersion != lastVersion) {
    ClearLocalCache(); // 캐시 삭제
}
// 2. 파일 무결성 검증
string downloadedHash = CalculateMD5(localPath);
if (downloadedHash != fileItem.md5Hash) {
    File.Delete(localPath); // 검증 실패 시 삭제
    throw new Exception("파일 무결성 검증 실패");
}
// 3. 카탈로그-앱 버전 비교
var appVersion = Application.version;
var catalogVersion = downloadManager.GetDownloadedCatalogVersion();
if (!IsCatalogNewer(catalogVersion, appVersion)) {
    needUpdateNoti.gameObject.SetActive(true); // 업데이트 알림
    return; // 게임 진행 차단
}
```

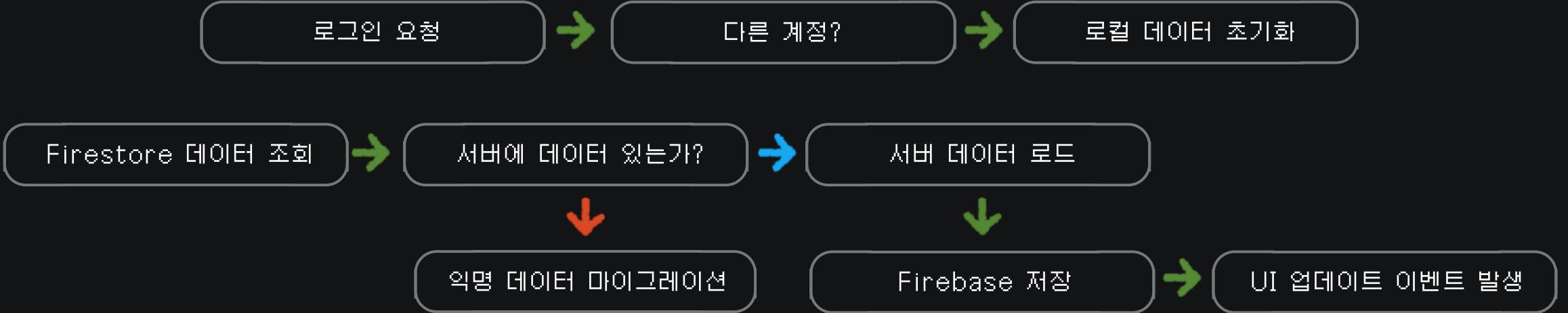


2. PentaShield



< 소셜 로그인 >

- 로그인/로그아웃/계정 삭제 시 **UserData 동기화 타이밍 문제 및 다른 계정 데이터 잔존 문제** 해결
- 다른 계정으로 로그인 시 이전 계정 데이터를 자동 초기화하고 서버 데이터와 동기화하는 마이그레이션 로직 구현
- 계정 삭제 시 Firestore 문서 삭제 → 랭킹 제거 → 로컬 초기화 → Auth 삭제 순서로 안전하게 처리
- 모든 데이터 변경 시 OnDataUpdated 이벤트를 발생하여 UI가 즉시 반영 되도록 이벤트 기반 아키텍처 적용



```
// 1. 다른 계정 감지 및 초기화
if (Data != null && Data.Id != firebaseUser.UserId) {
    ClearData(); // 이전 계정 데이터 삭제
}

// 2. 서버 데이터 확인
DocumentSnapshot snapshot = await userDocRef.GetSnapshotAsync();

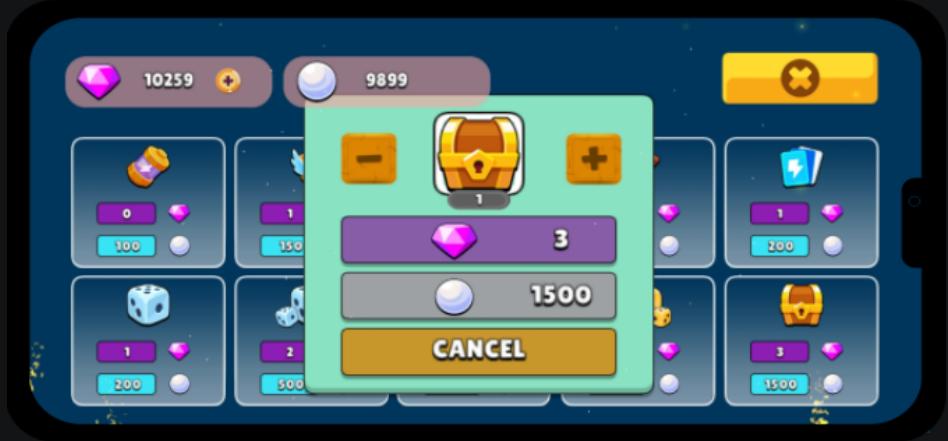
if (snapshot.Exists) {
    // 기존 유저: 서버 데이터로 덮어쓰기
    this.Data = snapshot.ConvertTo<UserData>();
} else {
    // 신규 유저: 익명 데이터 마이그레이션
    this.Data.Id = firebaseUser.UserId;
    this.Data.Name = firebaseUser.DisplayName;
}

// 3. 저장 및 UI 업데이트
await FirebaseSaveUserData();
NotifyDataUpdated(); // 이벤트 발생
```



2. PentaShield

<비동기 데이터에 대한 UI 동기화 >



- 비동기 데이터 로드와 UI 업데이트 타이밍 불일치로 인해, 데이터가 없는 상태에서 참조가 발생하여 Null 예러나 빈 화면이 노출되는 이슈가 있었습니다.

- IsInitialized 플래그 기반의 단계별 초기화 패턴을 적용하여, 데이터 로드와 검증이 완전히 완료된 후에만 UI가 접근하도록 실행 순서를 제어하여 안정성을 확보했습니다.

- 옵저버 패턴(Observer Pattern) 기반의 이벤트 아키텍처를 도입함으로써, 클라우드 데이터가 변경될 때마다 구독 중인 모든 UI가 즉각적으로 자동 업데이트되도록 구현했습니다.



```
// 1. 초기화 완료 대기
protected async virtual UniTask LateStart() {
    await UniTask.WaitUntil(() =>
        UserDataManager.Shared != null &&
        UserDataManager.Shared.IsInitialized == true);
}

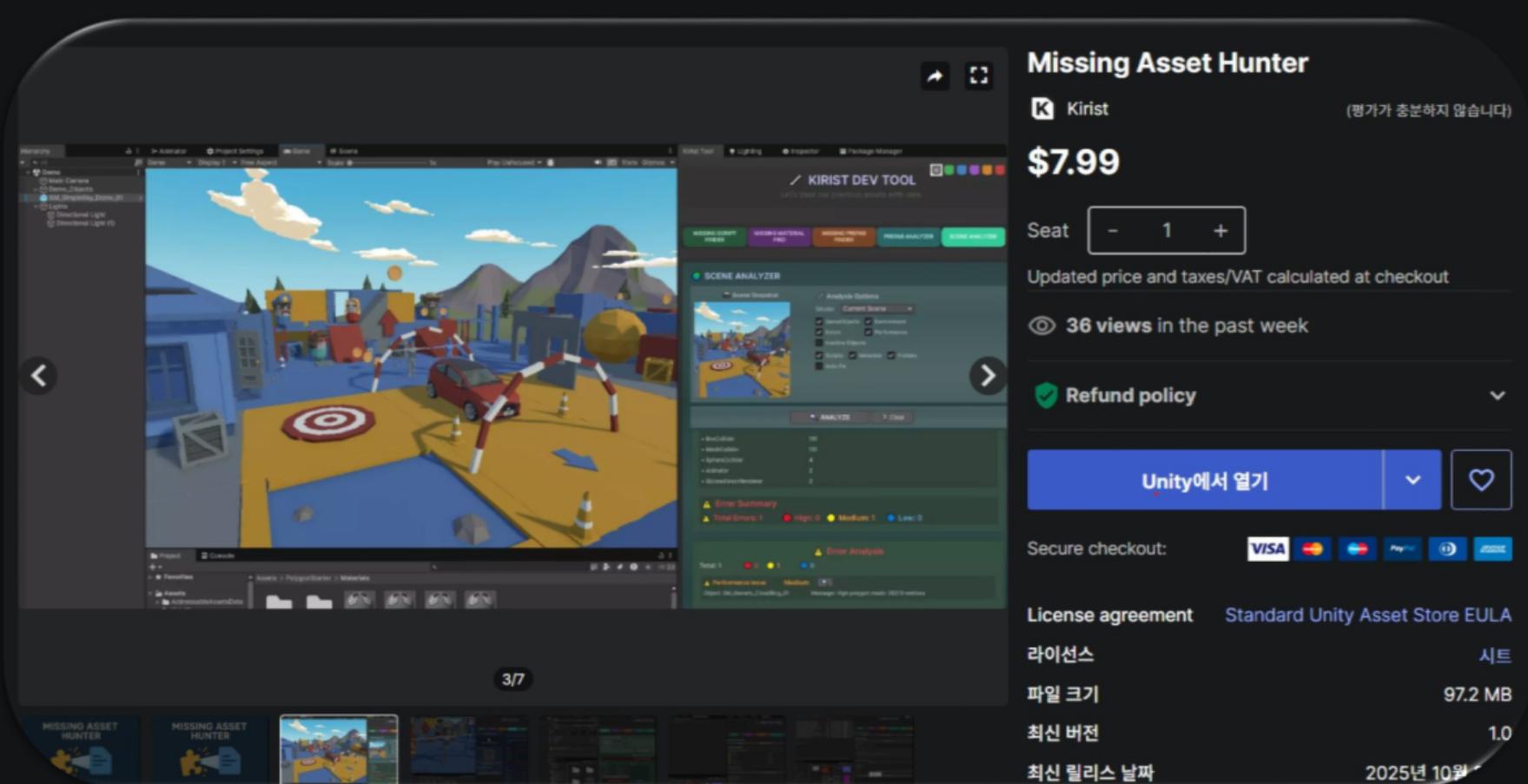
// 2. Observer Pattern: 이벤트 구독
UserDataManager.Shared.OnDataUpdated += HandleUserDataUpdated;

// 3. 데이터 변경 시 이벤트 발행
public void NotifyDataUpdated() {
    if (Data == null) return;
    OnDataUpdated?.Invoke(Data); // 모든 구독자에게 알림
}

// 4. 데이터 로드 보장 및 초기 반환
private async void OnEnable() {
    await RankingDataInit(forceRefresh: true);
    if (cachedRankings == null) return; // 초기 반환
    UpdateView(cachedRankings); // 데이터 로드 후 UI 업데이트
}
```

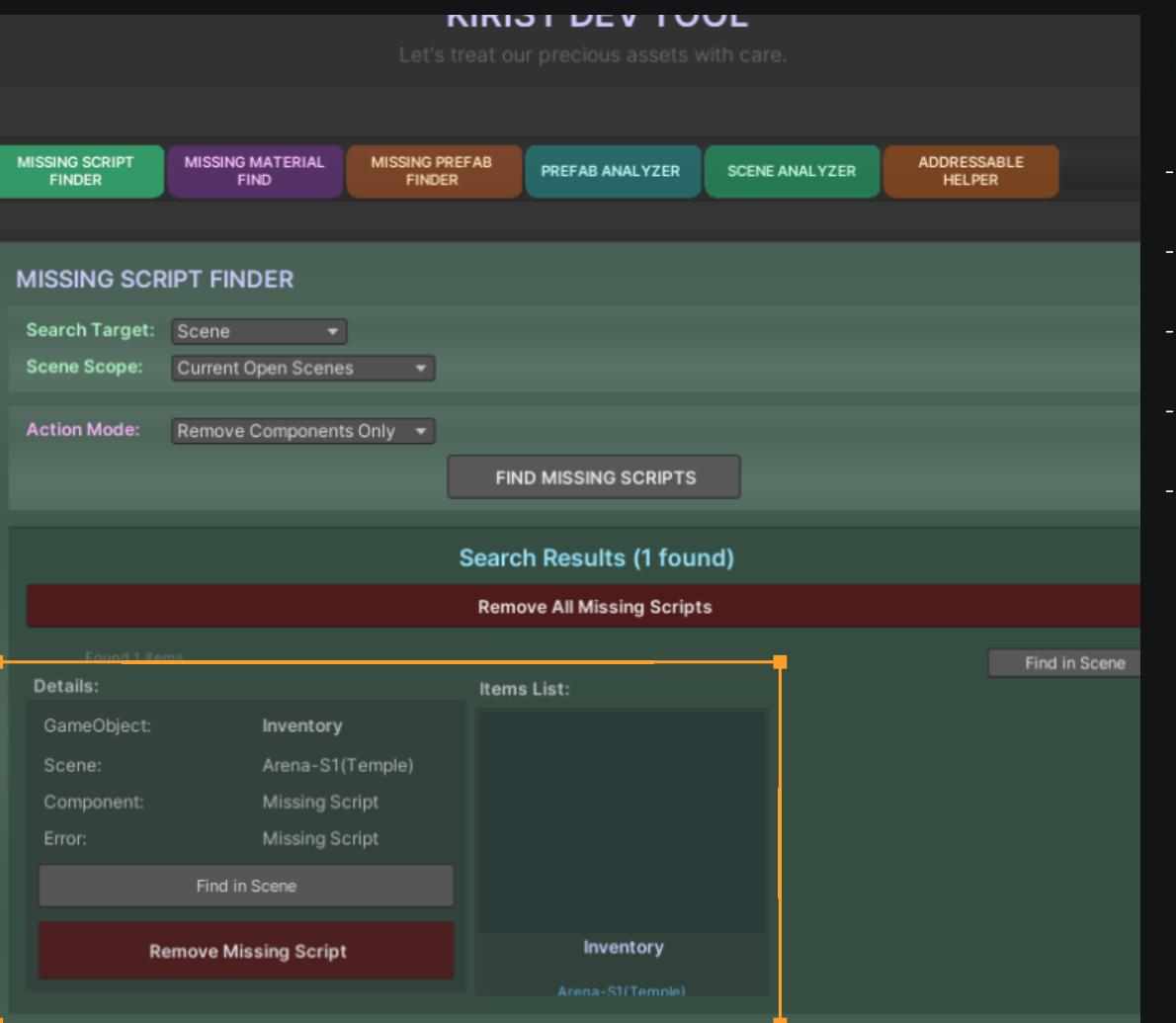
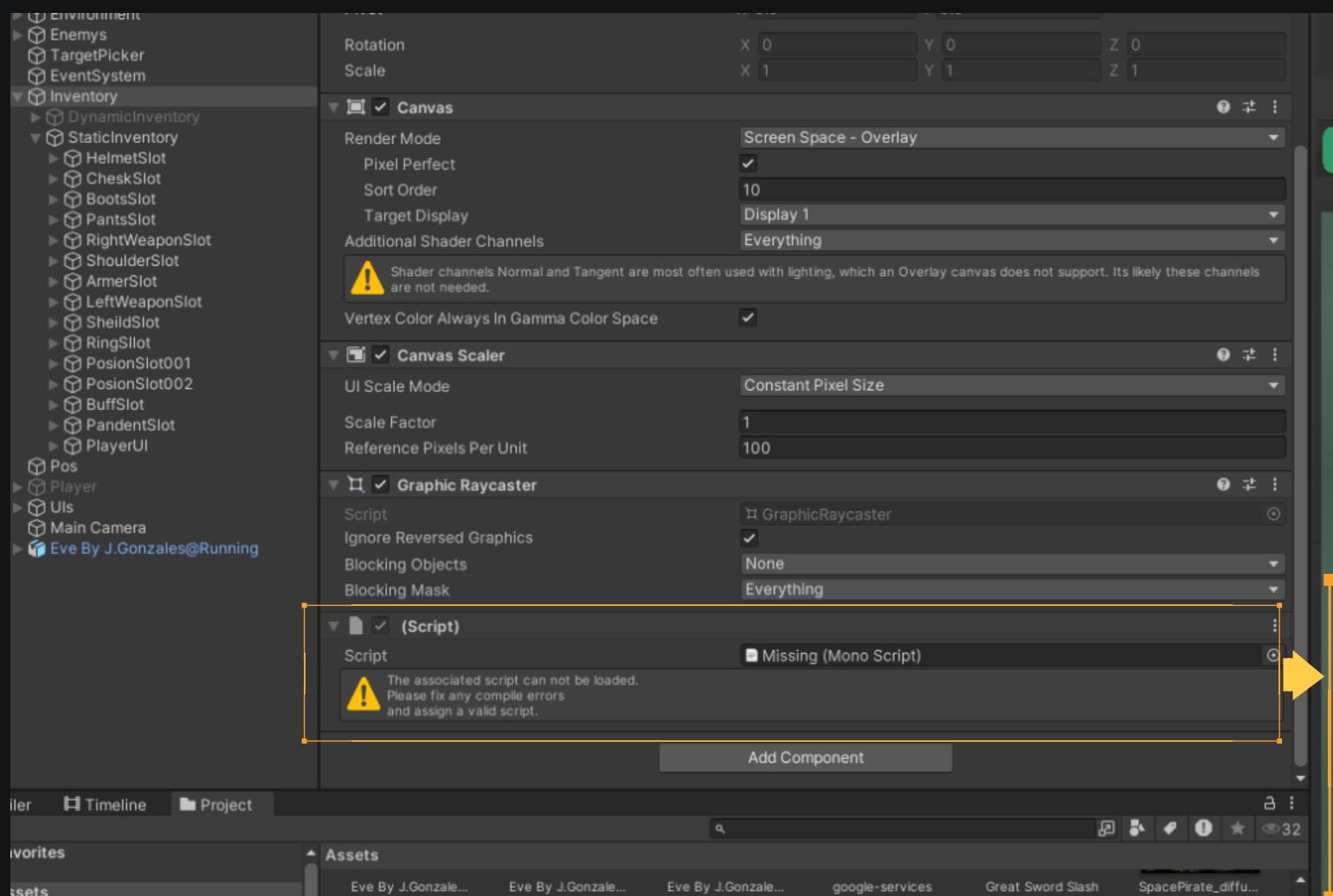


3. MISSING ASSET HUNTER



개발 인원	1인
프로젝트명	Missing Asset Hunter
장 르	개발 유ти리티
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  LINK

3. MISSING ASSET HUNTER



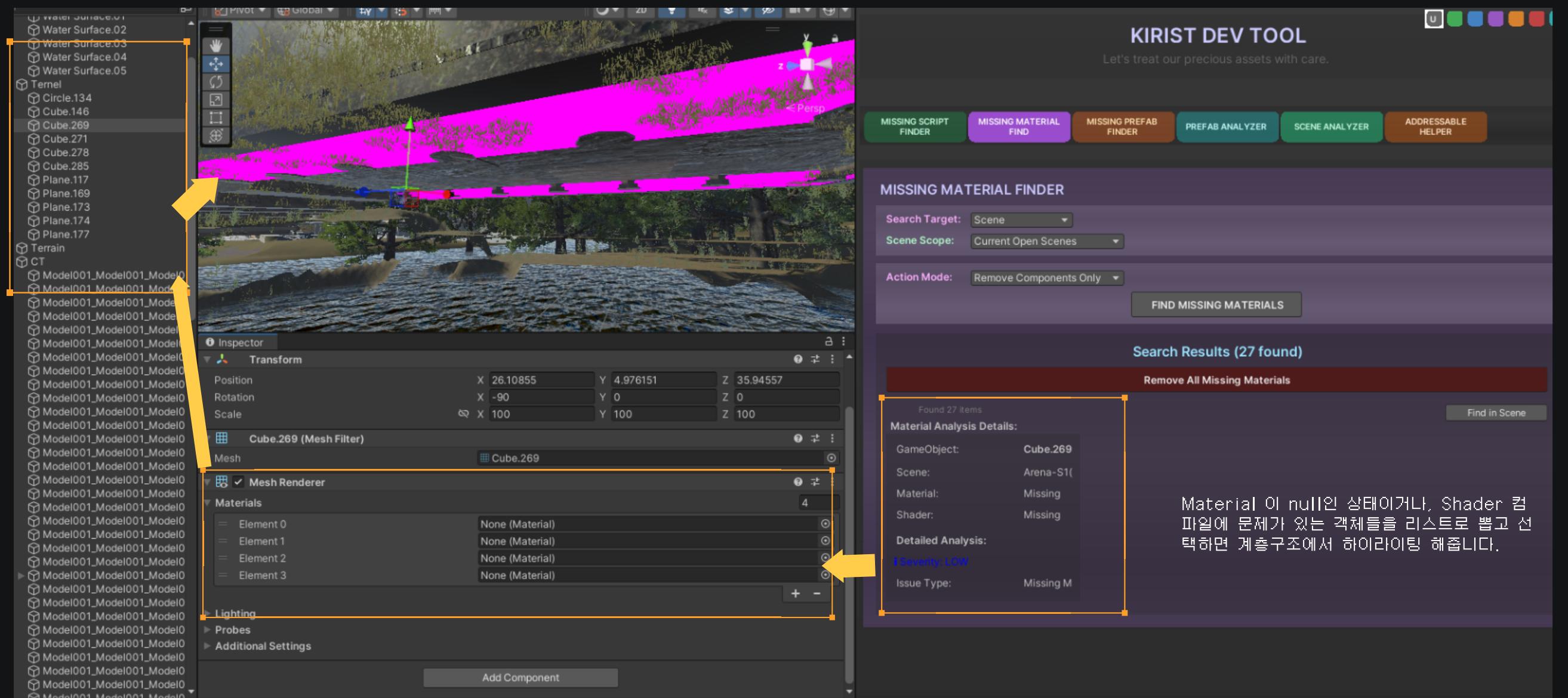
참조가 끊긴 컴포넌트를 즉시 계층 구조에서 탐색 및 삭제 가능하게 해줍니다.

참조 애러 스크립트 탐색

- 탐색 시작점 및 순회 구조 : 최상위 루트부터 재귀적 자식 노드 순회
- 컴포넌트 배열 분석 : Unity 직렬화 시스템을 활용한 컴포넌트 슬롯 추출
- Fake Null 상태 감지 : 스크립트 삭제/클래스명 변경으로 인한 참조 끊김 감지
- 프리팹 인스턴스 검증 : 원본-인스턴스 연결 관계 및 씬 오버라이드 검사
- 검사 결과 수집 및 분류 : 구조화된 에러 정보 및 위치 추적

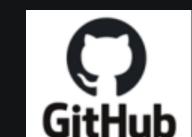


3. MISSING ASSET HUNTER



불안정한 쉐이더 탐색

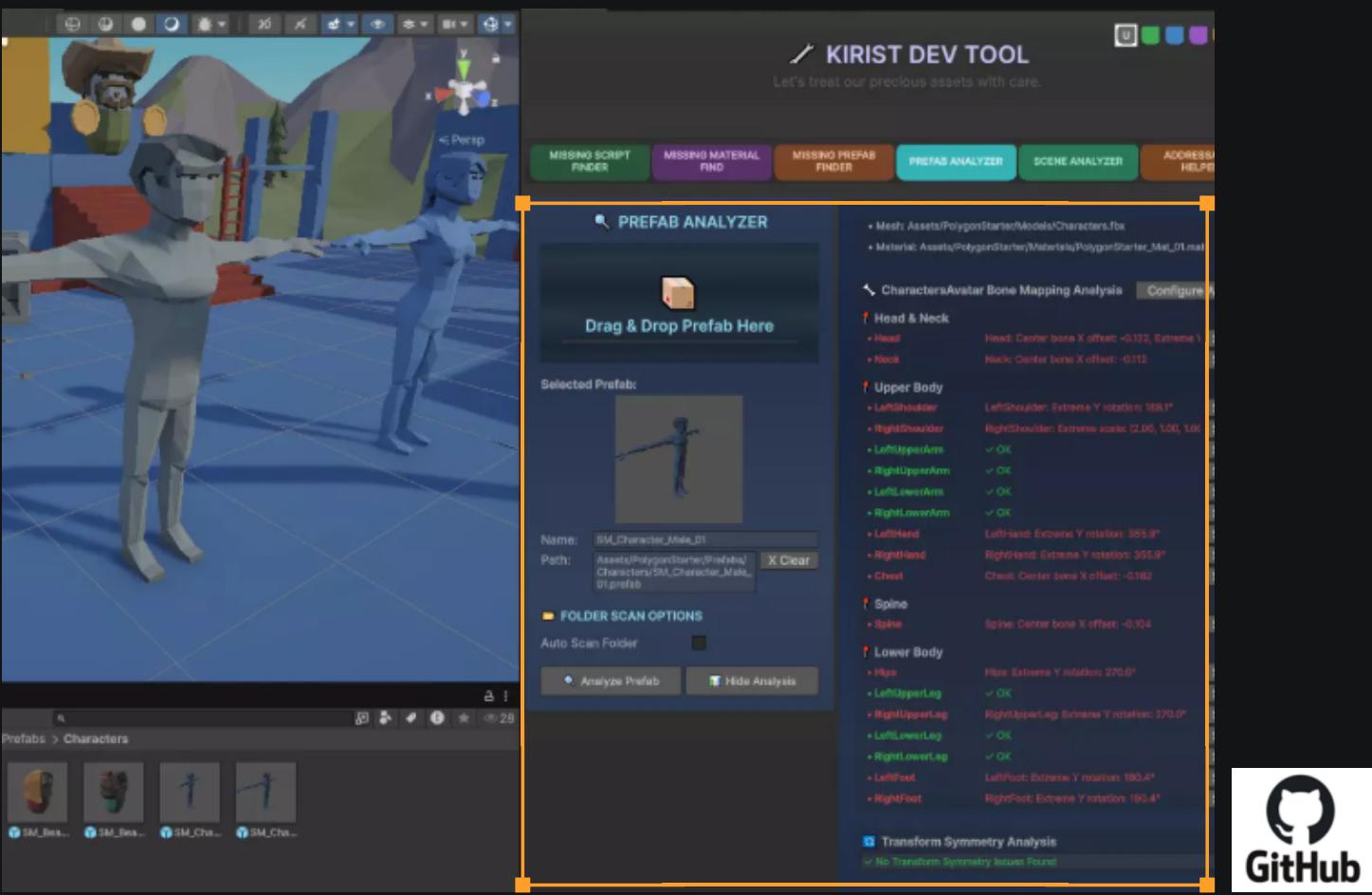
- 재귀적 렌더러 컴포넌트 탐색 : 모든 렌더러 타입 수집
 - IsErrorMaterial 정밀 검증 : 종합적인 머티리얼 상태 검증
 - 마젠타(Magenta) 이슈 감지 : Hidden/InternalErrorShader 우선 감지
 - 렌더 파이프라인 호환성 검사 : URP/HDRP/Built-in 파이프라인 불일치 감지
 - Custom 셰이더 깊은 검증
- > 1차: 런타임 상태 확인 (isSupported, passCount)
- > 2차: 소스 코드 구조 분석 (Regex 파싱, 함수명 불일치, 블록 구조 검증)



3. MISSING ASSET HUNTER

Scene 분석

Prefab 분석



- 재귀적 순회 및 Missing Assets 검사 : Missing Scripts, Materials, Prefabs 분석
- SerializedObject를 활용한 참조 관계 추적 : 컴포넌트 필드 순회 및 오브젝트 참조 추출
- AssetDatabase를 활용한 사용처 분석 : 씬 및 다른 프리팹에서의 사용처 추적
- Animator 뼈 매핑 검증 : Humanoid Avatar 검증 및 뼈 Transform 검증
- Transform 대칭성 검증 : 좌우 대칭 쌍 검증 및 비대칭성 감지



- 재귀적 GameObject 순회 및 Missing Assets 검사 : 최상위 루트부터 모든 자식 노드를 재귀적으로 순회하며 Missing Scripts, Materials, Prefabs, Error Shader 분석
- 환경 요소 분석 : Lighting, Camera, Terrain, PostProcessing 등 씬의 환경 설정 요소를 수집하고 설정 정보 분석
- 단계별 비동기 분석 : EditorApplication.update를 활용해 GameObjects → Components → Environment → Errors → Snapshot 순서로 단계별 비동기 분석 수행
- 성능 이슈 감지 : 모든 MeshFilter의 버텍스 개수를 검사하여 고폴리곤 메시를 성능 이슈로 분류
- 에러 통계 및 결과 정리 : 에러 유형별 및 심각도별로 분류하여 통계를 집계하고 최종 분석 결과 생성

4. REVENGER [졸업작품]

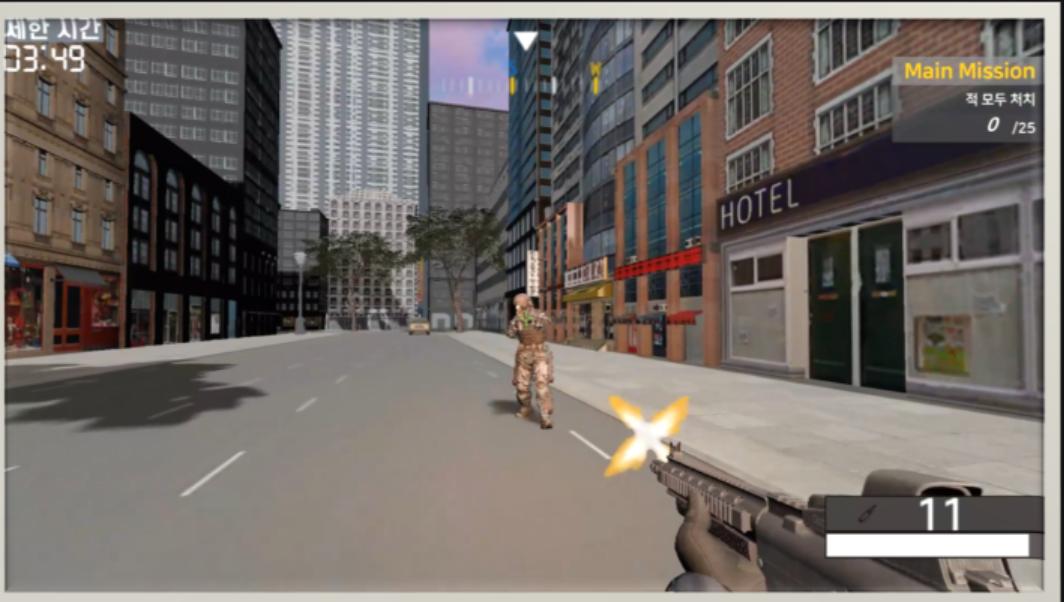


개발 인원	3인 (서버 1, 서버 및 기획 1, 클라 1)
프로젝트명	REVENGER
장르	3D FPS 멀티 협동 대전 게임
역할	클라이언트 전 과정 - 스테이지 로드 - 게임 컨텐츠 구현 - 사운드 - UI/UX - 그림자 - 애니메이션 - 객체 효과 및 연출 - 서버 패킷 상호작용 - 빌드 작업
환경	- Visual Studio 2022, DirectX 12, FMOD, Git hub, IOCP, DXTex - C++ 11
개발 기간	2022.08 ~ 2023.07
참고 링크	 LINK LINK LINK

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.

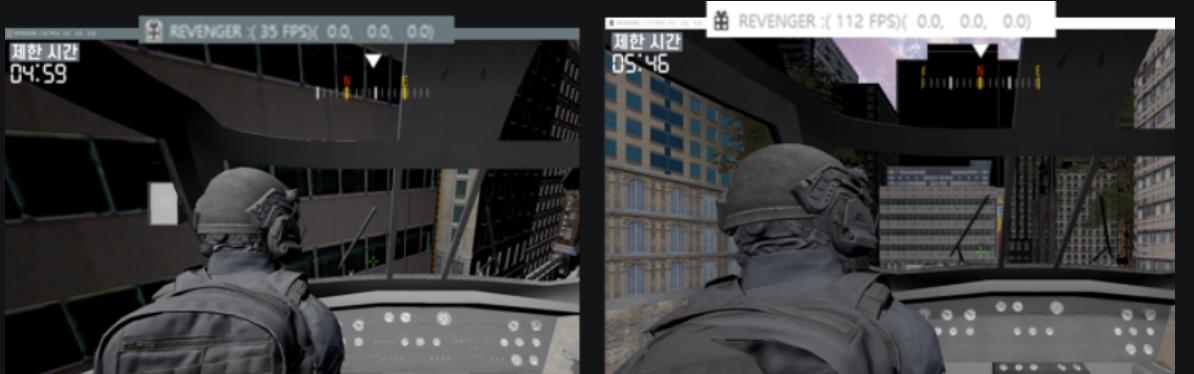
헬리콥터 격추 시 분해된 부품들과 피해 처리를 해주었습니다. 같은 방식으로 여부를 판단했고, 파편들이 모든 플레이어들에게 같은 방향과 무게로 동기화되어, 파편들과 충돌을 해주어야 했기에, 동일한 방향과 중력을 적용해주었습니다.

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 Bounding Box가 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, 뷔 행렬을 계산합니다.

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

멀티 렌더 타겟 뷔를 생성하고, 깊이-스텐 실 버퍼를 생성 및 초기화하고, 셰이더 변수를 생성합니다.

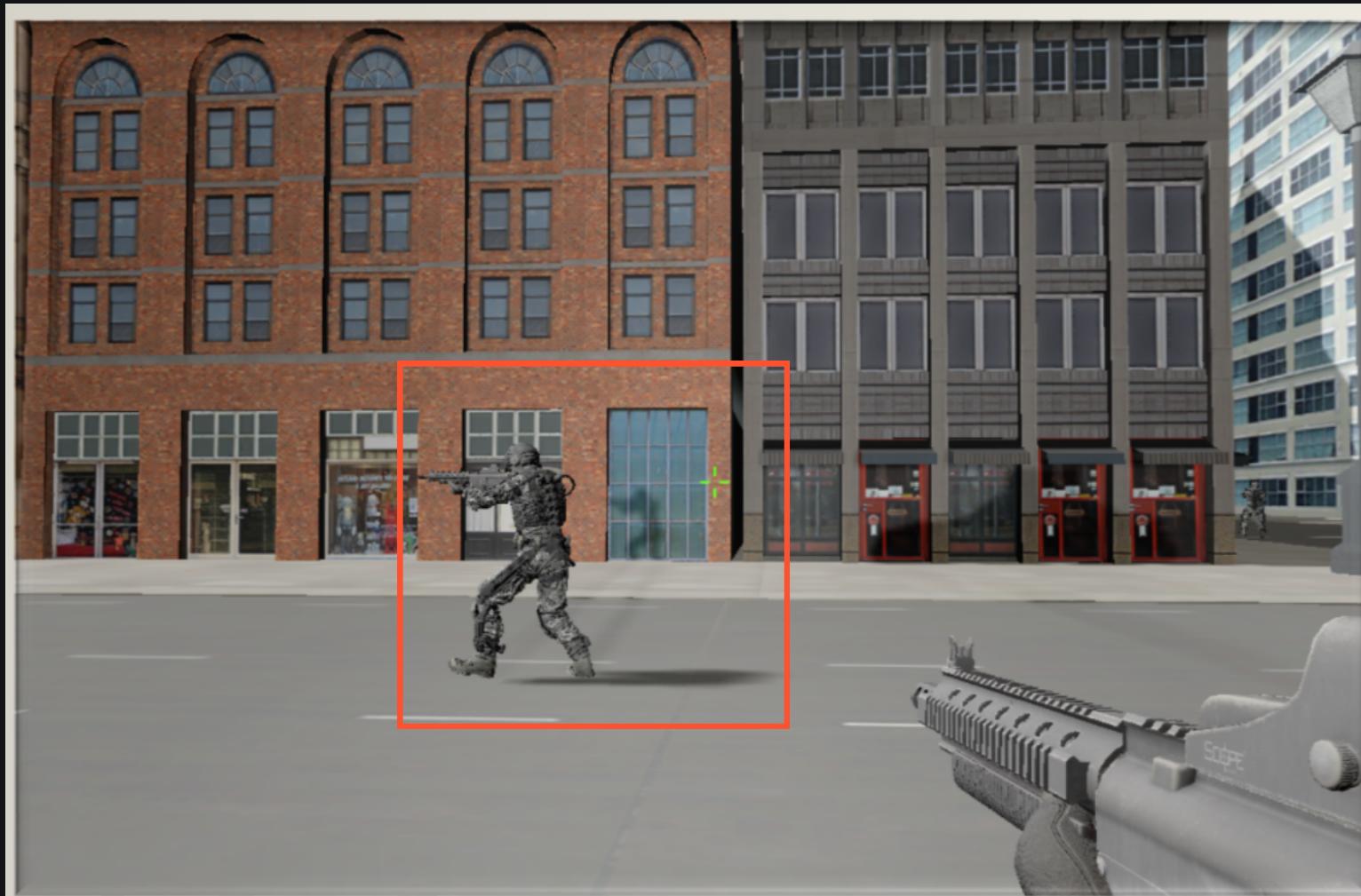
조명을 계산 시, PCF 방식을 사용하여 그림자 팩터를 계산하여 그림자를 더 부드럽게 만들게 됩니다.

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.

```
...
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;
        }
...
}
```

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. HUNTV ERSE [제작 중]



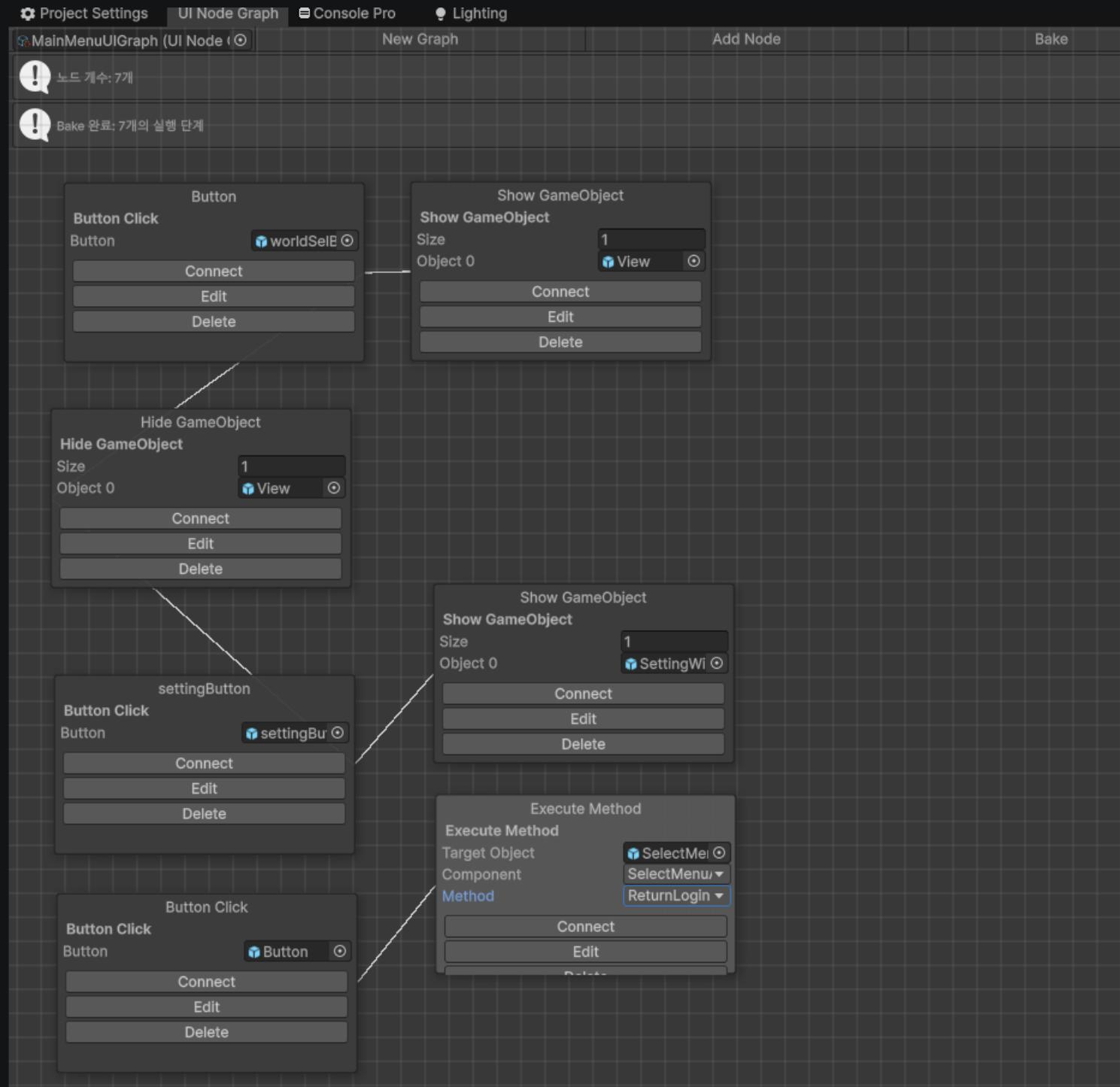
개발 인원	2인 개발 (서버1, 클라1)
프로젝트명	Huntverse
장르	2D 흥스크롤 액션 RPG
담당 역할	<p>메인 클라이언트</p> <ul style="list-style-type: none"> - 프로젝트 기획 및 디자인 - 게임 컨텐츠 개발 - UI/UX - 리소스 제작 (+AD) - 서버-클라이언트 데이터 동기화
플랫폼	PC (Steam)
개발 기간	2025.11.20 ~ 제작 중
환경	<ul style="list-style-type: none"> - Unity6+, vs2022, protobuf - Steamworks.NET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

5. HUNTVRSE [제작 중]



UI Node Graph System

기존의 파편화된 스크립트와 복잡한 인스펙터 방식은 UI 간 인과관계 파악이 어렵고 유지보수 효율이 떨어지는 문제점이 있었습니다.

이를 해결하기 위해 AI를 통해 노드 그래프 에디터를 빠른 시간 내에 직접 제작하여, 코드 수정 없이 UI 시퀀스와 레이어 그룹을 직관적으로 제어할 수 있는 제작 환경을 구축했습니다.

특히 'Bake' 시스템을 통한 고유 ID 정적 바인딩을 구현하여, 런타임 시 객체 탐색(Find) 부하를 제거하고 메모리 및 실행 성능을 극대화했습니다.

버튼을 누름으로써 원하는 객체들에 대한 활성 및 비활성화, 토글 대상 지정은 물론 지연(Delay), 메서드 실행 등 특수 노드 연출들도 지정할 수 있습니다.

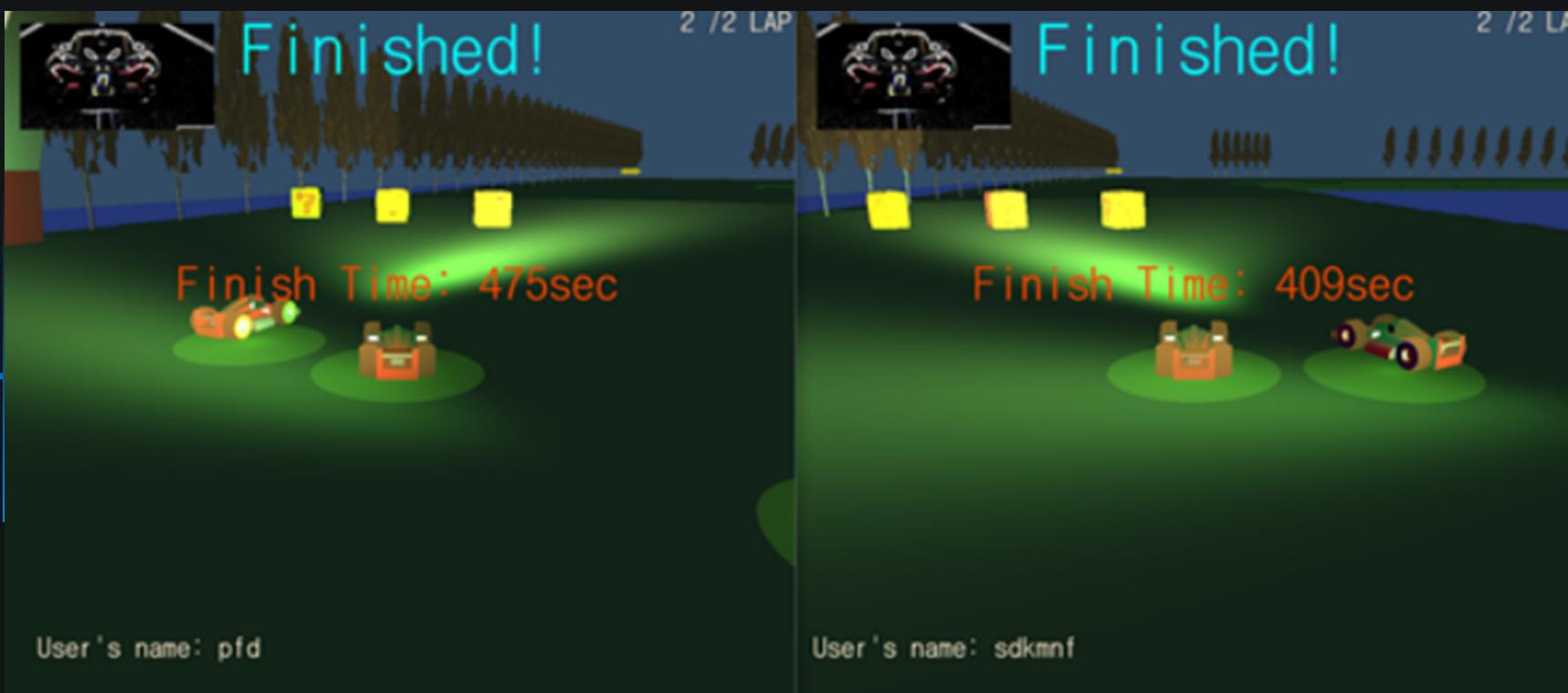
또한 사용자 정의 노드 추가가 가능한 확장 구조를 설계하여 입맛에 맞는 기능을 추가할 수도 있게하여, 복잡한 로직의 디버깅 편의성과 개발 생산성을 향상시켰습니다.



6. RacingAttack

[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	 GitHub LINK

6. RacingAttack

[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
```

클라이언트로부터 받은 이름이 이미 접속중인 다른 유저의 이름이 아닌지 확인합니다

```
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;
```

계정이 이미 접속 중 또는 존재하지 않는 이름일 때, 서버 상의 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(recved_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_pack에 START_DENY_ALREADYUSED 입력합니다.

6. RacingAttack

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 갖고 있는 아이템이 2개 미만일때만 새로운 아이템을 획득할 수 있도록 하였습니다.

작동하지 않는 미사일, 자신이 생성한 미사일, 피격 모션 중인 플레이어는 Ignore 대상으로 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]



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, 총들)을 활용한 제작물을 산출
참고 링크	LINK LINK



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



환경	vs2022, Unity6, WebGL
제작 목표	Unity 20주년 GameJam 참가
참고 링크	LINK



환경	vs2022, Unity2023
제작 목표	3d 컨텐츠 제작 공간
참고 링크	LINK

감사합니다