기술소개서

신정호

Tel +82-10-2507-8645
E-mail sjh98091@gmail.com

프로필 - 신정호

생년월일	1998.09.17
전화번호	+82-10-2507-8645
이메일	sjh98091@gmail.com
주요기술	C++, DirectX 9/11, Unreal 4/5, VR 및 모션캡쳐, HLSL
경력	1년 이상 (약 1년 5개월)

Git Hub Link

소개할 프로젝트

25.05 ~ 25.09	Unreal 5.4	Bip and Bop 출시
25.03 ~ 25.04	DirectX 11	Window Kill 모작

그 외 프로젝트

21.11 ~ 22.01	DirectX 11	It Takes Two 모작
21.09 ~ 21.11	DirectX 11	Dungeon Defenders 2 모작
21.06 ~ 21.07	DirectX 9	Ever Space 모작
21.04 ~ 21.05	DirectX 9	Skul 모작
21.02 ~ 21.03	Win API	소닉 3 모작

포트폴리오 소개 목차

1. Bip and Bop - Unreal 5.4 소개 영상

- 2025.06.01 ~ 2025.09.15
- 4인 개발
- 주요 역할: 빌드 관리, 멀티플레이 기능 및 유틸리티 구현
- 페이지 바로가기

2. Window Kill 모작 - DirectX 11, Win API 소개 영상

- 2025.02.27 ~ 2025.04.04
- 1인 개발
- 주요 역할: 자체 엔진 아키텍처, 좌표계 변환 시스템, 수학
- 페이지 바로가기

Bip and Bop

스팀 출시, 4인 개발

2인 협동, 멀티 플레이, 퍼즐, 무료

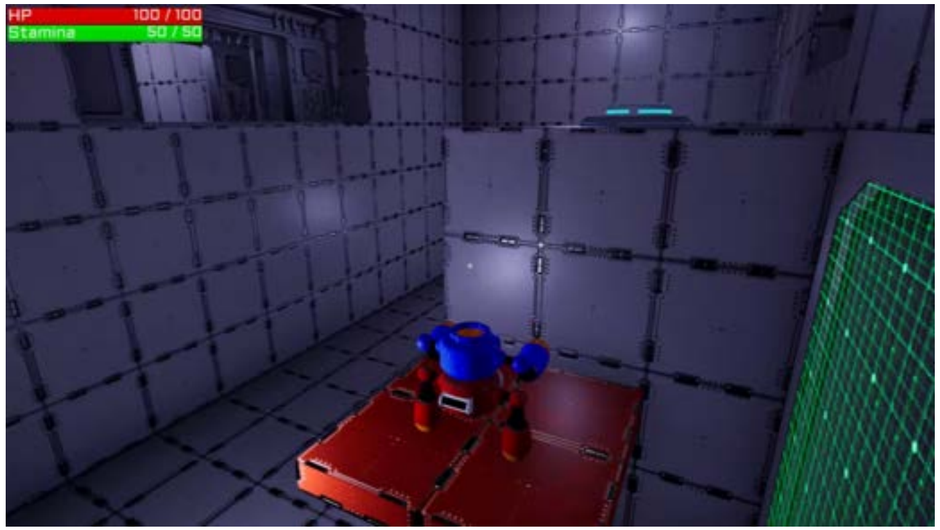
개발 이력

- 개발 기간 - 2025.06.01 ~ 2025.09.15
- 개발 인원 - 4명
- 사용 기술 - Unreal Engine 5.4, C++
- 개발 도구 - Visual Studio 2022
- 버전 관리 - Tortoise SVN
- 일정 관리 - Notion, Discord

포트폴리오 소개 영상 링크

상점페이지 링크

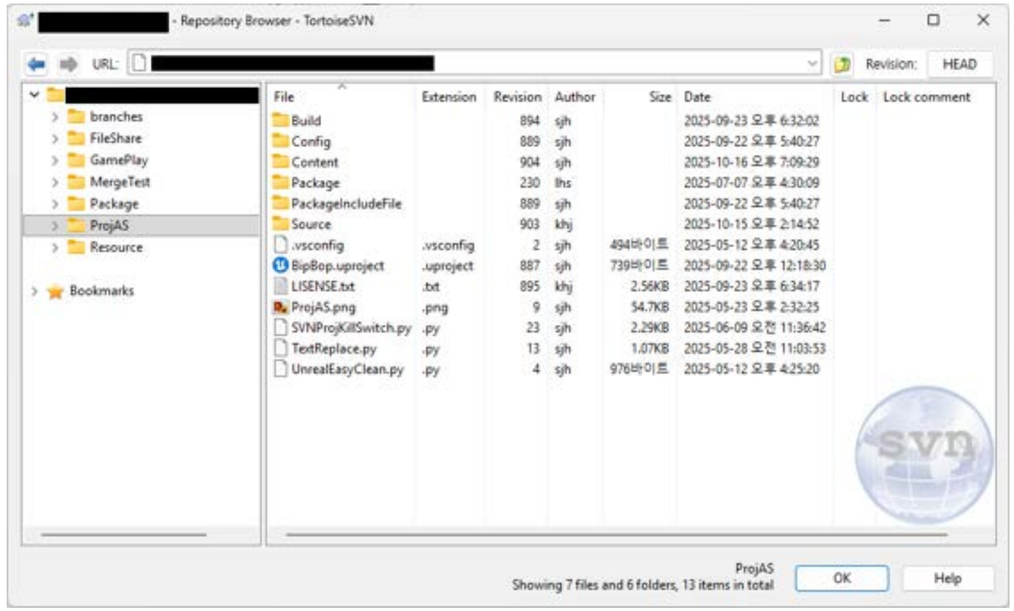
목차

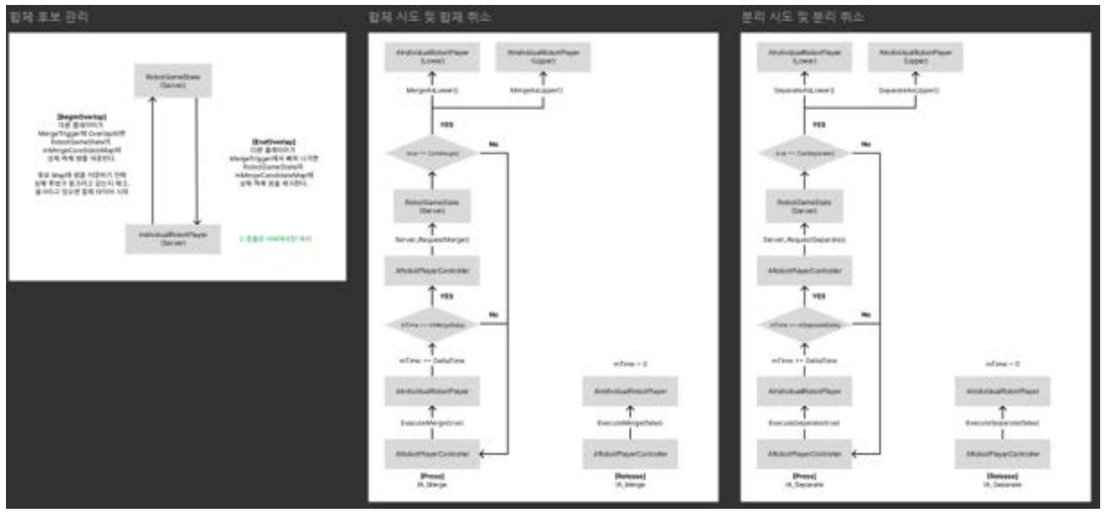


일정 및 버전 관리

- SVN을 이용해 버전 관리
- Notion을 통해 회의록 및 작업사항 확인
- Figma 시각화 및 다이어그램 표시용으로 사용

회의 내용	날짜	참석자
게임 컨셉 구체화 및 맵 에셋 선정	2025년 6월 10일	정 정호 신
@2025년 5월 28일 회의 Part2	2025년 5월 29일	정 정호 신
중간 점검 1회차	2025년 6월 10일	럴수 이
중간 점검 2회차	2025년 6월 17일	럴수 이
중간 점검 3회차	2025년 6월 23일	럴수 이
GameAify 대회 및 언리얼 페스타		
중간 점검 4회차	2025년 7월 1일	럴수 이
중간 점검 5회차	2025년 7월 14일	럴수 이
출시 전 점검 1	2025년 8월 14일	정 정호 신





OSS 기반 세션 관리 시스템 - 핵심 구현 내용 (1/4)

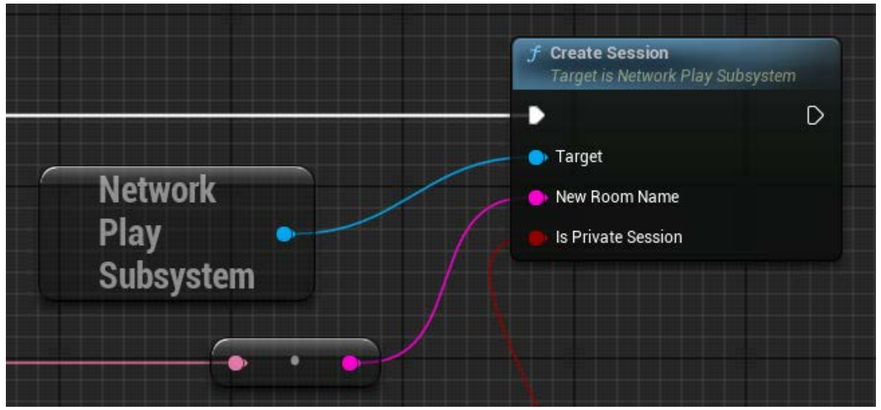
- Subsystem 아키텍처 도입
  - UGameInstanceSubsystem으로 모듈화
  - **유지보수성 및 코드 재사용성** 증대

- OSS(Online Subsystem) 추상화 및 래핑(Wrapping)
  - Steam, LAN 등 **플랫폼 통합 인터페이스**(Create, Find, Join) 구현
  - 복잡한 OSS의 델리게이트 및 핸들 처리 로직을 내부로 숨김
  - 외부(UI)에는 **단순화된 Dynamic Multicast Delegate만 노출**하여 **결합도(Coupling) 최소화**

- 비동기 예외 처리 시스템 (Async Error Queue)
  - 네트워크 오류(연결 끊김 등)는 UI로 즉시 표현할 수 없는 문제를 해결하기 위해 **TQueue 기반의 메시지 큐 시스템 구현**

- 커스텀 세션 데이터 관리
  - FOnlineSessionSettings를 활용하여 '초대 코드', '방 이름', '세션 상태 (Lobby/Ingame)' 등의 **메타데이터를 세션에 동기화**

Revision	Actions	Author	Date	Message
612		sjh	2025년 8월 21일 목요일 오후 6:09:32	*TestSteamSDK에 main 병합(rv.599)
591		sjh	2025년 8월 15일 금요일 오후 8:08:20	1. DebugGame을 실행 했을 때 CDO 및 S
590		sjh	2025년 8월 15일 금요일 오후 8:06:05	1. LoadingScreen 구조 수정 - 자동 Loadi
588		sjh	2025년 8월 14일 목요일 오후 6:30:13	test용 패키징 빌드 파일 삭제
587		sjh	2025년 8월 14일 목요일 오후 6:26:32	1. TreeConflict 수정 - UI/MainMenu 폴더
585		sjh	2025년 8월 14일 목요일 오후 6:06:37	1. 패키징 에러로 인해 구버전 WBP_Main
584		sjh	2025년 8월 14일 목요일 오후 5:48:57	*TestSteamSDK에 main 병합(rv.583)
576		sjh	2025년 8월 14일 목요일 오후 2:48:33	빌드 추가
568		sjh	2025년 8월 13일 수요일 오후 11:25:21	1. NetworkPlaySubsystem 수정 - 리팩토
567		sjh	2025년 8월 13일 수요일 오후 4:13:52	TestSteamSDK에 main.rv.566 병합
565		sjh	2025년 8월 13일 수요일 오후 4:01:54	1. steam연동 시에도 LAN모드 가능한 ba
560		sjh	2025년 8월 11일 월요일 오후 11:56:55	1. Steam User이름 예외처리 추가
556		sjh	2025년 8월 3일 일요일 오후 11:49:59	1. DefaultEngine.ini 수정 - Steam SDK 연



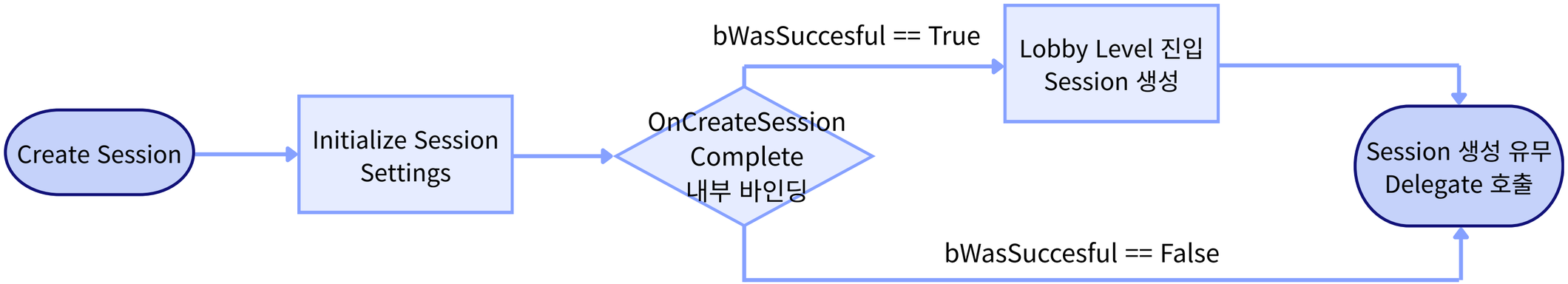
OSS 기반 세션 관리 시스템 - 핵심 구현 내용 (2/4)

특징: UI와 네트워크 로직의 분리

NetworkPlaySubsystem을 통해 **UI가 OSS의 존재를 전혀 모르도록 설계**
Subsystem내부에서 OSS의 델리게이트를 받아 처리
비동기 처리를 위해 Network관련 함수는 Delegate를 호출하는 구조로 설계
외부 클래스에서 접근가능한 Delegate와 OSS접근용 내부 Delegate로 분리

UI로는 성공/실패 유무만 반환, 내부적으로 다른 Level로 이동 등을 수행

```
/// 방 생성 델리게이트 이벤트
void UNetworkPlaySubsystem::OnCreateSessionComplete(FName SessionName, bool
{
    if (false == mSessionInterface.IsValid())
    {
        ULogHelperLibrary::DebugMessage(TEXT("SessionInterface is Invalid")
        return;
    }

    if (bWasSuccessful)
    {
        ULevelFlowSubsystem* LevelFlowSubsys = GetLevelFlowSubsystem();
        if (IsValid(LevelFlowSubsys))
        {
            LevelFlowSubsys->OpenListenServer(
                LevelFlowSubsys->GetDefaultLevelRow(ELevelType::Lobby));
        }
    }

    if (mOnCreateSessionCompleteDelegate.IsBound())
        mOnCreateSessionCompleteDelegate.Broadcast(bWasSuccessful);
```



OSS 기반 세션 관리 시스템 - 핵심 구현 내용 (3/4)

특징: OSS(Online Subsystem) 추상화, Session 관리 통합

멀티플레이 시 EMatchMode에 따라 플랫폼 환경 확인
UI같은 외부에서 호출 시, **동일한 함수사용**
내부적으로 EMatchMode에 따라 Session을 다양하게 생성

휴먼에러 방지를 위해 열거형과 클래스의 멤버변수 적극 활용

```
public:
    UFUNCTION(BlueprintCallable)
    2개의 청사진 참조
    void ServerStartGame(bool IsForcedStart);

    UFUNCTION(BlueprintCallable)
    2개의 청사진 참조
    void CreateSession(const FString& NewRoomName, bool IsPrivateSession = false);

    UFUNCTION(BlueprintCallable)
    2개의 청사진 참조
    void FindSession();

    UFUNCTION(BlueprintCallable)
    2개의 청사진 참조
    void JoinSessionRoomIndex(int32 RoomIndex);
```

```
void UNetworkPlaySubsystem::SetMatchMode(EMatchMode MatchMode)
{
    // 사용중인 세션이 있다면 파괴
    LeaveSession(false);

    mMatchMode = MatchMode;

    FName MatchModeName = NAME_None;

    switch (MatchMode)
    {
    case EMatchMode::Auto:
    {
        if (false == (FParse::Param(FCommandLine::Get(), TEXT("nosteam"))))
        {
            MatchModeName = SessionDefine::Online::LAN;
            mMatchMode = EMatchMode::LAN;
        }
        else
        {
            MatchModeName = SessionDefine::Online::STEAM;
            mMatchMode = EMatchMode::Steam;
        }
    }
        break;
    case EMatchMode::LAN:
        MatchModeName = SessionDefine::Online::LAN;
        break;
    case EMatchMode::Steam:
        MatchModeName = SessionDefine::Online::STEAM;
        break;
    default:
        break;
    }
```

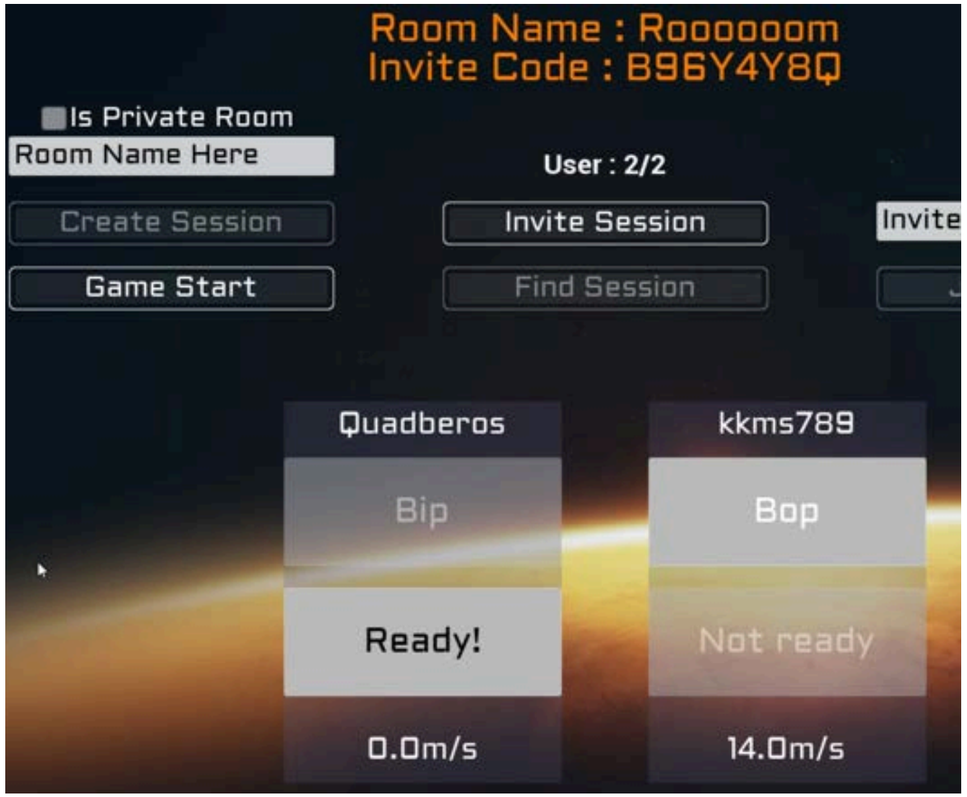
OSS 기반 세션 관리 시스템 - 핵심 구현 내용 (4/4)

특징: 커스텀 세션 데이터 동기화, 초대 코드 시스템

FOnlineSessionSettings를 활용하여 단순히 플레이어를 연결하는것 뿐 아니라
방제목, 비공개 세션, 초대 코드 등의 메타데이터를 세션에 동기화

별도의 친구 초대 UI 없이도 접속할 수 있도록 **독자적인 Invite Code 생성**
Session 입장 시 검증 로직을 구현하여 비공개 방 또는 Session List에 노출되지 않아도 Session 입장을 지원

```
bool UNetworkPlaySubsystem::CheckSessionInviteCode(const FString& InviteCode)
{
    if (mSessionSearch.IsValid())
    {
        const int32 Size = mSessionSearch->SearchResults.Num();

        for (int32 Index = 0; Index < Size; ++Index)
        {
            const FOnlineSessionSearchResult& SearchResult = mSessionSearch->SearchResults[Index];
            FString SessionInviteCode = {};

            SearchResult.Session.SessionSettings.Get(
                SessionDefine::Key::INVITE_CODE, SessionInviteCode);

            if (InviteCode == SessionInviteCode)
            {
                // 찾은경우 함수 종료
                JoinSessionRoomIndex(Index);
                return true;
```



통합 이펙트 스폰 시스템 - 핵심 구현 내용 (1/4)

- 이펙트/사운드 출력을 하나로 묶은 라이브러리
  - 파편화된 엔진의 **이펙트/사운드 시스템을 하나의 데이터 구조로 통합**
  - **생명주기를 자동화**한 유틸리티
  - 반복되는 작업을 기피해 **생산성 향상, 코드 중복 제거**

- UBlueprintFunctionLibrary을 상속해, 어디서나 접근 가능
  - FSpawnFXData를 통한 **데이터 중심 설계**
  - USpawnedFXHandle을 통한 **상태 제어 분리**
  - EPlayFXType(재생 타입)을 정의해 **상태 제어**
  - 내부적으로 BitwiseFX 클래스를 통해 **이펙트의 속성 정의 최적화**

- 외부에서 호출 가능한 public 함수는 모두 private 속성의 Spawn함수에서 이펙트/사운드를 생성하고 반환 하는 구조
  - 생성 함수 통합으로 **유지보수 용이**

```
class PROJAS_API USpawnFXLibrary : public UBlueprintFunctionLibrary
{
    GENERATED_BODY()

public:
    USpawnFXLibrary();

public:
    UFUNCTION(BlueprintCallable, Category = "SpawnFXLibrary|Spawn", meta = (WorldCo
    static void SpawnFX(UObject* WorldContextObject,
        const FSpawnFXData& SpawnFXData);

    UFUNCTION(BlueprintCallable, Category = "SpawnFXLibrary|Spawn", meta = (WorldCo
    static USpawnedFXHandle* SpawnFXWithHandle(UObject* WorldContextObject,
        const FSpawnFXData& SpawnFXData,
        EPlayFXType PlayFXType = EPlayFXType::RestartIfPlaying);

    UFUNCTION(BlueprintCallable, Category = "SpawnFXLibrary|Spawn", meta = (WorldCo
    static USpawnedFXHandle* CreateFXWithHandle(UObject* WorldContextObject,
        const FSpawnFXData& SpawnFXData,
        EPlayFXType PlayFXType = EPlayFXType::RestartIfPlaying);

private:
    UFUNCTION()
    0개의 청사진 참조
    static USpawnedFXHandle* Spawn(UObject* WorldContextObject,
        const FSpawnFXData& SpawnFXData,
        bool NeedSpawnedFXHandle,
        EPlayFXType PlayFXType = EPlayFXType::RestartIfPlaying);
```

통합 이펙트 스폰 시스템 - 핵심 구현 내용 (2/4)

특징: 통합 데이터 구조체 - FSpawnFXData

FSpawnFXData 구조체를 통해 생성할 FX를 정의
다양한 상황(이펙트만, 이펙트+위치, 사운드 포함 등)에 맞춰 생성자를 다중 정의,
호출부(Client Code)의 가독성을 극대화

```cpp
void ASoundTrigger::Multicast_PlaySound_Implementation()
{
    if (mSoundTableData.IsBackgroundMusic)
    {
        UGameInstanceAS* GameInst = GetGameInstance<UGameInstanceAS>();
        if (IsValid(GameInst))
        {
            GameInst->PlayPersistent2DSound(mSoundTableData.Sound);
        }
    }
    else
    {
        USpawnFXLibrary::SpawnFX(this, FSpawnFXData(mSoundTableData.Sound,
            mSoundTableData.SoundType, GetTransform()));
    }
}
```

```cpp
FSpawnFXData(UNiagaraSystem* Niagara,
             ENiagaraType NaiagaraType,
             const FTransform& EffectTransform,
             ENiagaraRenderType RenderType = ENiagaraRenderType::ShouldCull)
    : FSpawnFXData(Niagara, NaiagaraType, RenderType)
{
    SetTransform(EffectTransform);
}

FSpawnFXData(class UParticleSystem* Cascade,
             const FTransform& EffectTransform)
    : ParticleEffect(Cascade)
{
    SetTransform(EffectTransform);
}

FSpawnFXData(USoundBase* Sound, ESoundType SoundType,
    const FTransform& EffectTransform,
    float VolumeMultiplier = 1.f, float PitchMultiplier = 1.f, float StartPoi
{
    SetSoundEffect(Sound, SoundType);
    SetSoundData(VolumeMultiplier, PitchMultiplier, StartPoint);
    SetTransform(EffectTransform);
}
```

통합 이펙트 스폰 시스템 - 핵심 구현 내용 (3/4)

특징: 상태 제어 분리 - USpawnedFXHandle

USpawnedFXHandle 클래스를 통해
스폰된 이펙트와 사운드 컴포넌트들을 멤버 변수로 관리

엔진의 AutoDestroy에만 의존하지 않고, **별도의 LifeTime타이머**를 통해
이펙트와 사운드의 수명을 강제로 동기화 및제어 (Self-Destroy 구현)

EPlayFXType 열거형에 따라 **이펙트의 재생 정책(Policy) 지원**
AlwaysPlay:기존 이펙트는 유지하고 **겹쳐서 재생**
RestartIfPlaying: 기존 것을 **끄고 다시 재생**
IgnoreIfPlaying: 이미 **재생 중이면 무시**

```
mFXHandle = USpawnFXLibrary::CreateFXWithHandle(this,
    SpawnFXData, EPlayFXType::IgnoreIfPlaying);
```

```
void AProjectileBase::PlayFX()
{
    if (IsValid(mFXHandle))
    {
        mFXHandle->Play();
    }
```

```
void USpawnedFXHandle::ReadyDestroy()
{
    ClearTimer();
    Stop();
    DestroyFX();

    // 가비지 컬렉션 대상으로 명시적으로 표시
    MarkAsGarbage();
}
```

```
void USpawnedFXHandle::Play()
{
    if (EPlayFXType::AlwaysPlay == mPlayFXType)
    {
        if (IsPlayingAnyFX())
        {
            FSpawnFXData DisposableSpawnFXData = mSpawnFXData;

            if (mSpawnFXData.OnFinishedNiagaraFX.IsBound()) { ... }

            if (mSpawnFXData.OnFinishedParticleSystemFX.IsBound()) {

            if (mSpawnFXData.OnFinishedSoundFX.IsBound()) { ... }

            USpawnFXLibrary::SpawnFX(this, DisposableSpawnFXData);
        }
    }
    else if (EPlayFXType::RestartIfPlaying == mPlayFXType)
    {
        Stop();
    }
    else if (EPlayFXType::IgnoreIfPlaying == mPlayFXType)
    {
        if (IsPlayingAnyFX())
        {
            return;
        }
    }
```

통합 이펙트 스폰 시스템 - 핵심 구현 내용 (4/4)

특징: 데이터 패킹과 메모리 최적화

FSpawnFXData의 멤버변수인 int32 AssetOption로 정의
AssetOption int32 자료형에 BitMask 사용, 언리얼의 다양한 이펙트를 유연하게 확장

추후 옵션이 늘어나도 **구조체 크기 변화 없이 비트 마스크만 추가해 확장성 용이**
다수의 boolean 플래그를 4Bytes(int32) 하나로 압축

```
USTRUCT(BlueprintType)
파생된 블루프린트 클래스 0개
struct FSpawnFXData
{
    void SetNiagaraType(ENiagaraType NiagaraType)
    {
        BitwiseFX::Set(AssetOption, NiagaraType);
    }

    void SetNiagaraRenderType(ENiagaraRenderType NiagaraRenderType)
    {
        BitwiseFX::Set(AssetOption, NiagaraRenderType);
    }

    void SetSoundType(ESoundType SoundType)
    {
        BitwiseFX::Set(AssetOption, SoundType);
    }
```

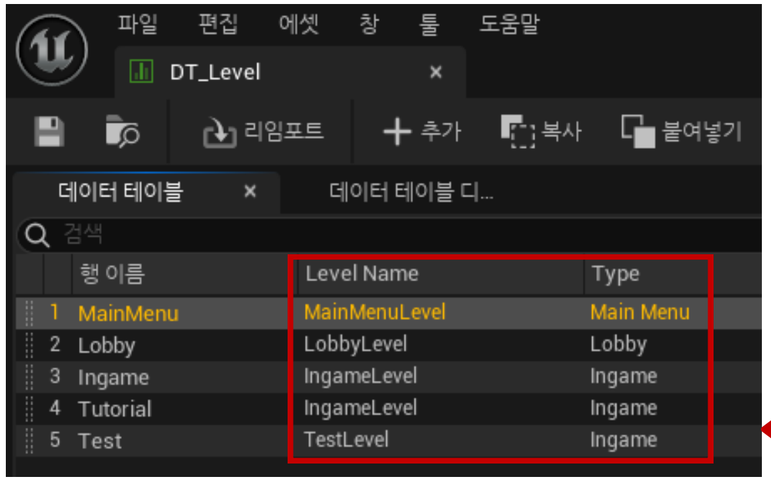
```
class BitwiseFX
{
    friend struct FSpawnFXData;
    friend class USpawnFXLibrary;

    static const uint8 SHIFT_NAIGARA_TYPE = 1;
    static const uint8 SHIFT_NAIGARA_RENDER_TYPE = 2;
    static const uint8 SHIFT_SOUND_TYPE = 3;

    static void Set(int32& AssetOption, ENiagaraType BoolEnum)
    {
        _set(AssetOption, static_cast<uint8>(BoolEnum), SHIFT_NAIGARA_TYPE);
    }

    static void _set(int32& assetOption, uint8 bit, uint8 bitPosition)
    {
        assetOption &= ~(1 << bitPosition);
        assetOption |= (bit << bitPosition);
    }

    static bool _isInclude(int32 assetOption, uint8 bit, uint8 bitPosition)
    {
        int32 mask = (1 << bitPosition);
        int32 extractBit = (assetOption & mask);

        if (bit == 1)
            return extractBit != 0;
        else
            return extractBit == 0;
    }
};
```

레벨 상태 관리자 - 핵심 구현 내용 (1/3)

- 전반적인 레벨 이동을 관리하는 **InstanceSubsystem 상속**
  - **레벨의 상태 및 정보 관리** (CurrentLevelInfo, PrevLevelInfo)
  - **멀티플레이어 환경(ServerTravel, ClientTravel) 고려**

- DataTable을 활용한 **데이터 주도형 레벨 관리**
  - “CheckSplitLevelName” 함수 등을 통해 “Level”이라는 접미사 유무 확인으로 **유연하게 RowName을 찾아내도록 문자열 파싱 처리**

- 레벨 이동 시 호출되는 **로딩 스크린 구현**
  - 레벨 전환 시(Travel)에도 로딩 위젯이 파괴되지 않도록 구현



LevelName: 레벨의 경로
Type: 레벨의 상태

```cpp
bool ULevelFlowSubsystem::FindLevelTable(FLevelTableRow& FindTable, FName RowName) const
{
    FindTable = {};
    bool Result = false;

    UWorld* World = GetWorld();
    UGameInstance* GameInst = IsValid(World) ?
        World->GetGameInstance() : nullptr;

    UTableSubsystem* TableSubsys = IsValid(GameInst) ?
        GameInst->GetSubsystem<UTableSubsystem>() : nullptr;

    if (IsValid(TableSubsys))
    {
        RowName = CheckSplitLevelName(RowName.ToString());

        FLevelTableRow* LevelRow = TableSubsys->FindTableRow<FLevelTableRow>(
            UTableData::TableName::LEVEL, RowName);

        if (LevelRow)
        {
            Result = true;
            FindTable = *LevelRow;
        }
        else
        {
```

DataTable의 Row 순회
LevelName으로 Level Load

레벨 상태 관리자 - 핵심 구현 내용 (2/3)

특징: InstanceSubsystem 의존성 관리

LevelFlowSubsystem은 DataTable을 필요로 하기 때문에,
TableSubsystem을 먼저 초기화하도록 의존성 추가

```cpp
void ULevelFlowSubsystem::Initialize(FSubsystemCollectionBase& Collection)
{
    // UTableSubsystem먼저 Intialize를 호출한다
    Collection.InitializeDependency<UTableSubsystem>();

    Super::Initialize(Collection);

#if WITH_EDITOR 활성 전처리기 블록
#endif

    UGameInstanceAS* GameInstAS = Cast<UGameInstanceAS>(GetGameInstance());
    if (IsValid(GameInstAS))
    {
        GameInstAS->mOnPreLoadMapDelegate.AddDynamic(
            this, &ULevelFlowSubsystem::OnPreLoadMap);
        ③
        GameInstAS->mOnPostLoadMapDelegate.AddDynamic(
            this, &ULevelFlowSubsystem::OnPostLoadMap);
    }
    else
    {
        FCoreUObjectDelegates::PreLoadMap.AddUObject(
            this, &ULevelFlowSubsystem::OnPreLoadMap);

        FCoreUObjectDelegates::PostLoadMapWithWorld.AddUObject(
            this, &ULevelFlowSubsystem::OnPostLoadMap);
    }
}
```

Level이 Load되는 시점을 GameInstance와 분리하기 위해,
UGameInstanceAS의 Delegate에 바인드해 순서를 보장

```cpp
void UGameInstanceAS::Init()
{
    Super::Init();

    ①
    FCoreUObjectDelegates::PostLoadMapWithWorld.AddUObject(
        this, &UGameInstanceAS::OnPostLoadMap);

    FCoreUObjectDelegates::PreLoadMap.AddUObject(
        this, &UGameInstanceAS::OnPreLoadMap);
}

void UGameInstanceAS::OnPostLoadMap(UWorld* LoadedWorld)
{
    ②
    if (mOnPostLoadMapDelegate.IsBound())
        mOnPostLoadMapDelegate.Broadcast(LoadedWorld);
}
```

GameInstance에 **종속적인 InstanceSubsystem**이기 때문에,
호출 순서 또한 **GameInstance의 Delegate를 사용하도록 구현**

레벨 상태 관리자 - 핵심 구현 내용 (3/3)

특징: 레벨 상태 정의 및 로딩 스크린

LevelName을 이용해 현재 레벨의 이름, 레벨 타입등 상태를 저장
이로인해, MainMenu, Lobby, Ingame등 **레벨의 상태 구별 가능**

```cpp
void ULevelFlowSubsystem::OnPreLoadMap(const FString& LevelName)
{
    FString SplitLevelName = CheckSplitLevelName(LevelName);

    mPrevLevelInfo = mCurrnetLevelInfo;
    FindLevelTable(mCurrnetLevelInfo, *SplitLevelName);

    ShowLoadingScreen();
}
```

ShowLoadingScreen
Viewport에 직접적으로 **Widget을 주입, 레벨 전환간 유지**

```cpp
USTRUCT(BlueprintType)
파생된 블루프린트 클래스 0개
struct FLevelTableRow : public FTableRowBase
{
    GENERATED_BODY()
public:

    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    0 Blueprints에서 변경됨
    FString LevelName;

    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    0 Blueprints에서 변경됨
    ELevelType Type = ELevelType::End;

    UPROPERTY(EditAnywhere, BlueprintReadOnly)
    0 Blueprints에서 변경됨
    FString Option = {};
```

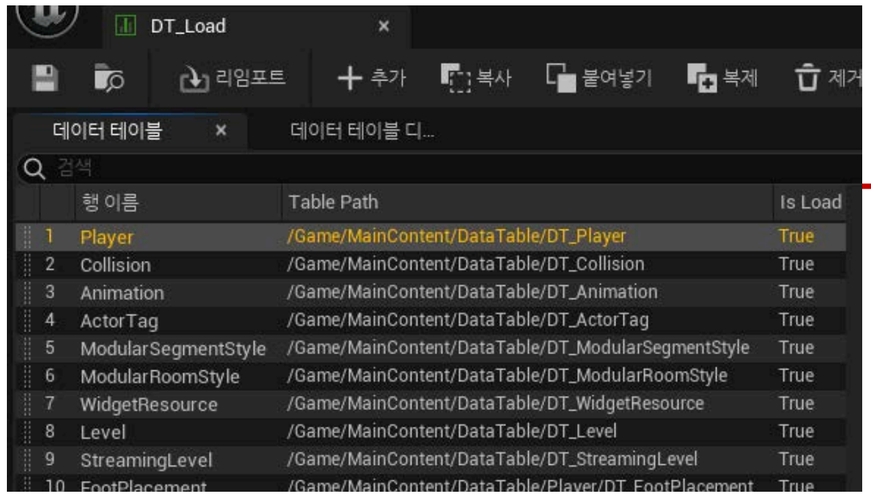
```cpp
void ULevelFlowSubsystem::ShowLoadingScreen(
    bool IsAutoControlLoadingScreen /*= false*/)
{
    if (false == IsEnableShowLoadingScreen())
        return;

    if (IsAleadyShowingLoadingScreen())
        return;

    mLoadingWidget = CreateWidget<UUserWidget>(
        GetGameInstance(), mLoadingWidgetClass);

    GEngine->GameViewport->AddViewportWidgetContent(
        mLoadingWidget->TakeWidget(), INT32_MAX);

    mIsAutoControlLoadingScreen = IsAutoControlLoadingScreen;
}
```

데이터 테이블 관리자 - 핵심 구현 내용 (1/4)

- **메타 테이블(Meta Table)** 구조
  - 수십 개의 테이블 경로를 코드에서 분리하기 위해, 테이블 목록을 관리하는 **마스터 테이블 시스템을 구축, 유지보수 향상**
  - 다양한 구조체의 데이터 테이블을 타입 안전(Type-Safe)하게 조회 가능하도록 **템플릿 함수 추가, 생산성 향상**

- **최적화 및 리소스 관리**
  - 에셋 위주의 데이터 테이블의 경우 TSoftObjectPtr 등을 사용, **과도한 메모리 사용 방지**

- **에디터 환경에서 접근 가능한 상호보완 라이브러리 추가**



Table을 관리하는 **UTableSubsystem**

```
bool UTableSubsystem::LoadTable()
{
    UDataTable* TablePaths = TABLE_LOAD(UTableData::TablePath::LOAD_TABLE_PATH);

    if (!IsValid(TablePaths))
        return false;

    bool Result = true;
```

반복문을 이용해 Data Table Load

```
    TablePaths->ForeachRow<FLoadTableRow>
        (TEXT("Not Found Table Row..!"),
            [this, &Result](const FName& Key, const FLoadTableRow& Value)
            {
                if (Value.IsLoad)
                {
                    UDataTable* LoadTable = TABLE_LOAD(Value.TablePath);
                    if (true == IsValid(LoadTable))
                        mTableMap.Add(Key, LoadTable);
                    else
                        Result = false;
                }
            });

    return Result;
}
```

데이터 테이블 관리자 - 핵심 구현 내용 (2/4)

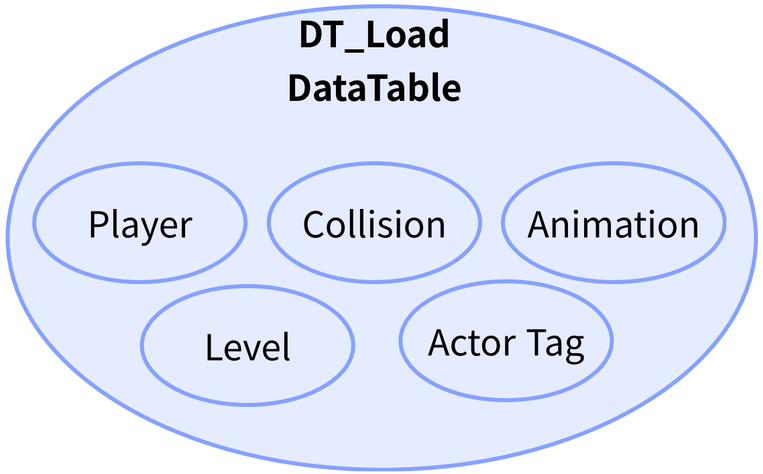
특징: 마스터 테이블 구조

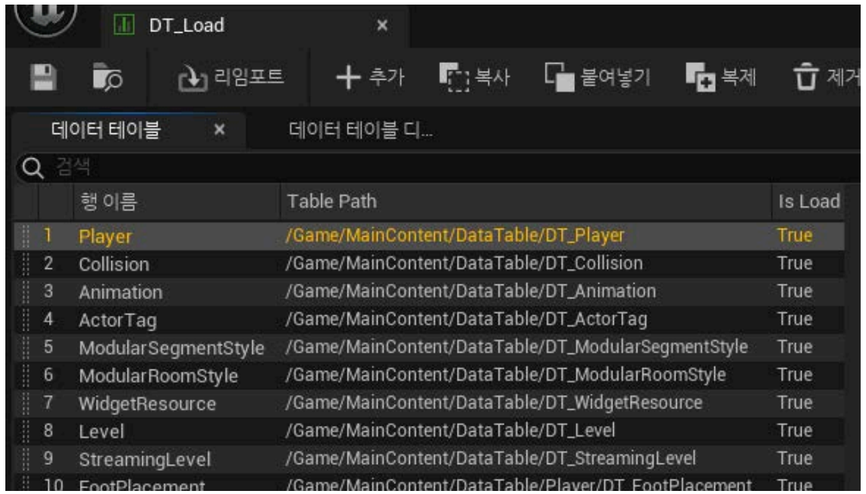
다양한 구조체의 데이터 테이블을 타입 안전하게 조회 가능하도록,
템플릿 함수를 도입해 생산성 향상

```
public:
    UDataTable* FindTable(FName Name)
    {
        return mTableMap.Contains(Name) ?
            *mTableMap.Find(Name) : nullptr;
    }

    template<typename T>
    T* FindTableRow(FName TableName, FName RowName)
    {
        UDataTable* Table = FindTable(TableName);

        if (nullptr == Table)
            return nullptr;

        return Table->FindRow<T>(RowName,
            TEXT("FindTableRow Failed. Not Found Table Row"));
    }
```

DT_Load 데이터 테이블에서 게임 구성에 필요한 데이터 테이블을 Load하는 방식
해당 **마스터 테이블에 행만 추가**하면 새로운 데이터 테이블 사용 가능, **협업 효율성 향상**

DT_Load 다이어그램





	행 이름	Table Path	Is Load
1	Player	/Game/MainContent/DataTable/DT_Player	True
2	Collision	/Game/MainContent/DataTable/DT_Collision	True
3	Animation	/Game/MainContent/DataTable/DT_Animation	True
4	ActorTag	/Game/MainContent/DataTable/DT_ActorTag	True
5	ModularSegmentStyle	/Game/MainContent/DataTable/DT_ModularSegmentStyle	True
6	ModularRoomStyle	/Game/MainContent/DataTable/DT_ModularRoomStyle	True
7	WidgetResource	/Game/MainContent/DataTable/DT_WidgetResource	True
8	Level	/Game/MainContent/DataTable/DT_Level	True
9	StreamingLevel	/Game/MainContent/DataTable/DT_StreamingLevel	True
10	FootPlacement	/Game/MainContent/DataTable/Player/DT_FootPlacement	True

데이터 테이블 관리자 - 핵심 구현 내용 (3/4)

특징: 지연 로딩, 리소스 관리

일부 DataTable은 **Asset을 사용**하기 때문에, **소프트 레퍼런스(TSoftObjectPtr)로 TableRow를 구성**
지연 로딩(LoadSynchronous) 사용해 에셋을 로드한 다음, **일반적인 형태로 재구성**

UI의 리소스와 같이 **에셋을 주로 사용하는 경우**의 데이터에만 적용

```
USTRUCT(BlueprintType)
파생된 블루프린트 클래스 0개
struct FWidgetResourceTableRow : public FTableRowBase
{
    GENERATED_BODY()

    FWidgetResourceTableRow() {}
    ~FWidgetResourceTableRow() {}

    UPROPERTY(EditAnywhere, BlueprintReadWrite,
        0 Blueprints에서 변경됨
        meta = (AllowedClasses = "Texture2D, MaterialInterface"))
    TArray<TSoftObjectPtr<UObject>> TextureList = {};

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<TSoftObjectPtr<class USoundCue>> SoundList = {};
};
```

지연 로드 이후, 재구성

```
USTRUCT(BlueprintType)
파생된 블루프린트 클래스 0개
struct FWidgetResourceInfo
{
public:
    GENERATED_BODY()

    FWidgetResourceInfo() {}

    UPROPERTY(EditAnywhere, BlueprintReadWrite,
        0 Blueprints에서 변경됨
        meta = (AllowedClasses = "Texture2D, MaterialInterface"))
    TArray<TObjectPtr<UObject>> TextureList = {};

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    0 Blueprints에서 변경됨
    TArray<TObjectPtr<class USoundCue>> SoundList = {};
};
```

데이터 테이블 관리자 - 핵심 구현 내용 (4/4)

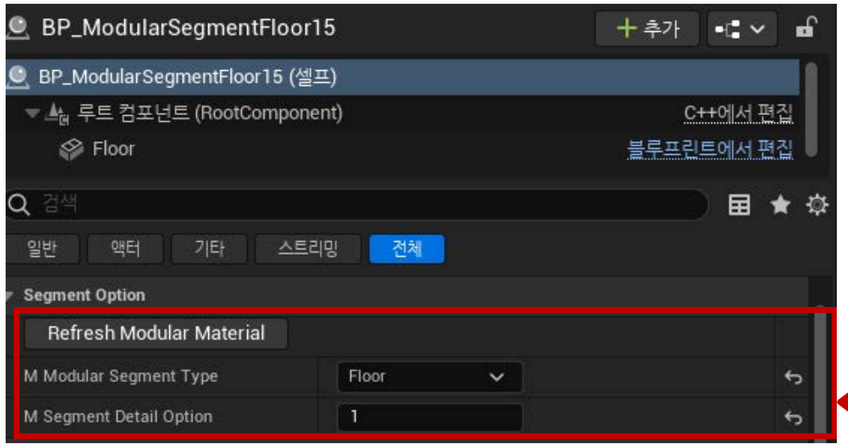
특징: **에디터 상태의 환경에서 접근 가능**

에디터 편집 중에 테이블을 사용해야 하는 예외사항
Subsystem은 GameInstance에 종속적이며, 런타임이어야 접근 가능
이를 보완하기 위해, **에디터 전용 테이블접근 라이브러리 구현**

에디터/런타임 데이터 접근 이원화

주로, 레벨 배치, 데이터 검증 등에 사용

레벨에 배치된 바닥 액터의 재질(Material)을 변경하고, 즉각 적용




```
class PROJAS_API UEditorTableLoadLibrary : public UBlueprintFunctionLibrary
{
    GENERATED_BODY()

public:
    UFUNCTION(BlueprintCallable)
    0개의 청사진 참조
    static bool IsOnlyEditor()
    {
        return GIsEditor && !GIsPlayInEditorWorld && !IsRunningCommandlet();
    }
```

에디터 유무 확인

```
#if WITH_EDITOR
public:
    // UTableData::TableName
    UFUNCTION(BlueprintCallable)
    0개의 청사진 참조
    static UDataTable* FindTable(FName TableName)
    {
        UDataTable* LoadTable = Cast<UDataTable>(
            StaticLoadObject(UDataTable::StaticClass(), nullptr,
                *UTableData::TablePath::LOAD_TABLE_PATH));

        if (false == IsValid(LoadTable))
            return nullptr;

        FLoadTableRow* LoadRow = LoadTable->FindRow<FLoadTableRow>(
            TableName, TEXT("Find Failed"));

        if (nullptr == LoadRow || false == LoadRow->IsLoad)
            return nullptr;

        UDataTable* TargetTable = Cast<UDataTable>(
            StaticLoadObject(UDataTable::StaticClass(), nullptr,
                *LoadRow->TablePath));

        return TargetTable;
    }
```

에디터에서 테이블 접근

인게임 기능 구현 - 플랫폼(발판) 무브먼트 (1/4)

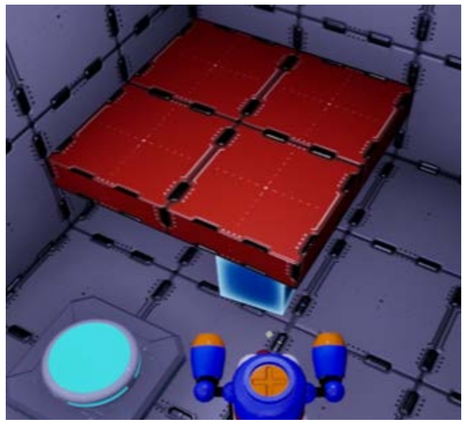
플랫폼(발판) 무브먼트 구현: **UPlatformMovementComponent**

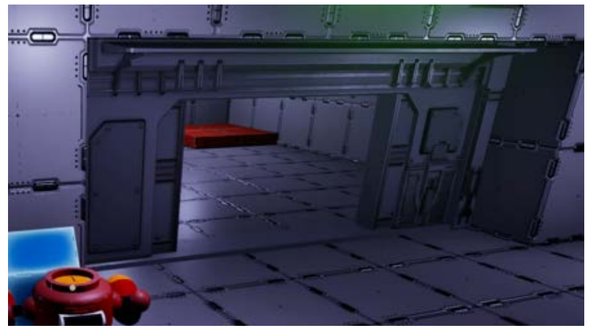
기반 클래스 확장: 엔진이 제공하는 보간 이동(UInterpToMovementComponent)을 그대로 쓰지 않고, **게임 로직에 필요한 기능을 추가하기 위해 상속받아 확장**

플랫폼 전용 동작: 일반적인 액터 이동이 아니라, 플레이어가 밟거나 장애물에 막히는 등 “움직이는 발판”에 필요한 **물리적 상호작용과 대기 로직이 포함**

네트워크 동기화: 서버가 이동을 시뮬레이션하고, 클라이언트는 서버의 CurrentTime을 받아 부드럽게 보간(Interpolate)하여 보여주는 구조

```
UCLASS(Blueprintable, ClassGroup = Movement, meta = (BlueprintSpawnableComponent))
파생된 블루프린트 클래스 0개
class PROJAS_API UPlatformMovementComponent : public UInterpToMovementComponent,
    public IMovementInterface
{
    GENERATED_BODY()

public:
    UPlatformMovementComponent();

#pragma region VirtualFunc

protected:
    virtual void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const override;
    virtual void HandleImpact(const FHitResult& Hit, float TimeSlice = 0.f, const FVector& MoveDelta = F
```



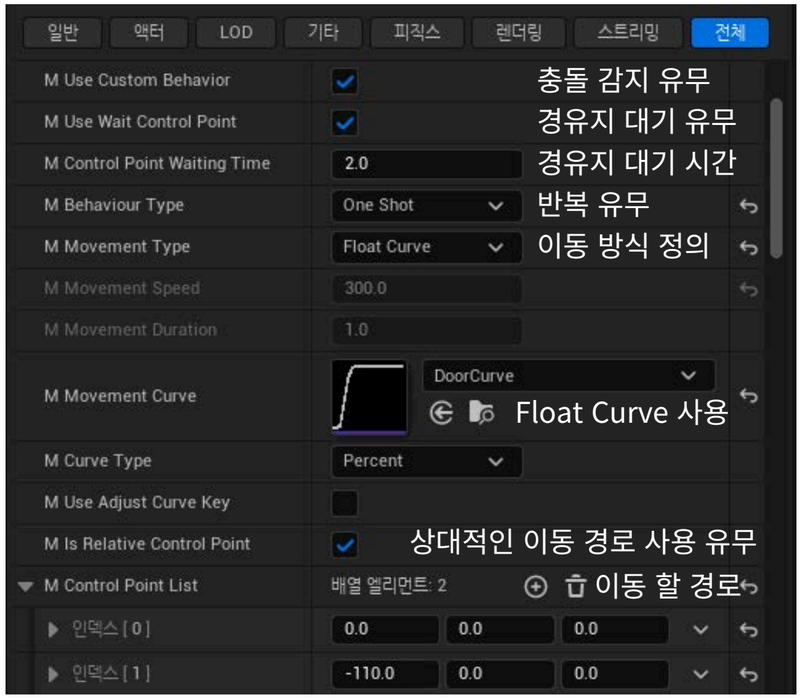


Component로 구현한 기능이기 때문에
플랫폼의 움직임 및 게이트의 열림/닫힘으로도 활용

인게임 기능 구현 - 플랫폼(발판) 무브먼트 (2/4)

특징: **기반 클래스 확장**

EPlaftomMovementType에 따라, 플랫폼의 이동모드를 설정
일정한 속도로 이동하는 **SpeedPerSeconds** (속도 기반),
거리에 상관없이 **정해진 시간에 도착**하는 **InDuration** (시간 기반),
UCurveFloat를 사용하여 특별한 이동을 부여하는 **FloatCurve** (커브 기반) 등을 지원



플랫폼/게이트의 동작을 정의
이동방식, 경유지 대기 유무 등

```
UENUM(BlueprintType)
enum class EPlatformMovementType : uint8
{
    SpeedPerSeconds,
    InDuration,
    FloatCurve,
};
```

```
void UPlatformMovementComponent::InitMovementOption()
{
    mPrevControlPoint = 0;
    mPrevMoveDirection = CurrentDirection;
    BehaviourType = mBehaviourType;
    SetControlPointList(mControlPointList, mIsRelativeControlPoint);

    switch (mMovementType)
    {
    case EPlatformMovementType::SpeedPerSeconds:
        SetMovementSpeed_Implementation(mMovementSpeed);
        break;
    case EPlatformMovementType::InDuration:
        SetDuration(mMovementDuration);
        break;
    case EPlatformMovementType::FloatCurve:
        SetTimeLine(mMovementCurve);
        break;
    }
}
```

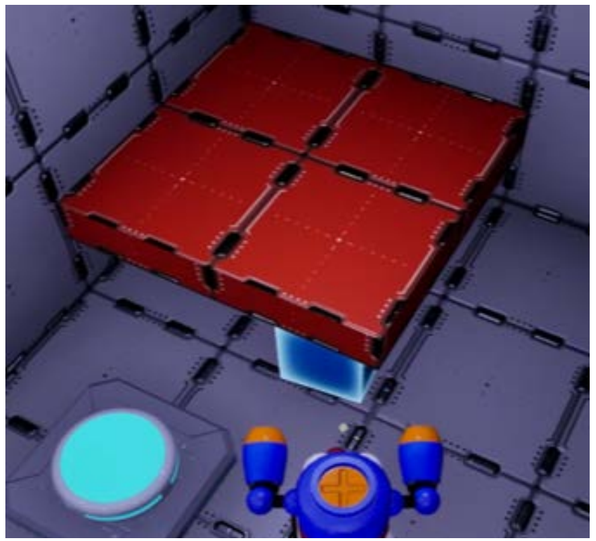
인게임 기능 구현 - 플랫폼(발판) 무브먼트 (3/4)

특징: **플랫폼 전용 동작**(물리적 상호 작용)

플랫폼 이동 시 **장애물 유무를 판단** (CalculateSweepBehaviorType)
무시하고 장애물을 밀고 **나갈 것인지, 대기 할 것인지 움직임을 제어**

플랫폼의 **이동 방향(Direction)에 Z(높이)가 없는 방향의 경우 Ignore**
Z가 음수(DownVector, 낙하)인 방향은 NeedImpact

위 아래로 움직이는 빨간색 플랫폼이
밑의 액터를 감지해, 이동을 중단



```cpp
bool UPlatformMovementComponent::MoveUpdatedComponentImpl(
    const FVector& Delta, const FQuat& NewRotation, bool bUseSweep,
    FHitResult* OutHit, ETeleportType Teleport)
{
    bool Result = false;

    UPrimitiveComponent* UpdatedPrimitiveComp = GetUpdatedPrimitive();
    if (mUseCustomBehavior && IsValid(UpdatedPrimitiveComp))
    {
        Result = Super::MoveUpdatedComponentImpl(Delta, NewRotation, bUseSweep, OutHit, Teleport);

        EPlatformSweepBehaviorType PlatformSweepBehaviorType = CalculateSweepBehaviorType(OutHit);

        switch (PlatformSweepBehaviorType)
        {
        case EPlatformSweepBehaviorType::Ignore:
            bUseSweep = false;
            Result = false;
            Teleport = ETeleportType::None;
            break;
        case EPlatformSweepBehaviorType::NeedImpact:
            bUseSweep = true;
            break;
        default:
            bUseSweep = false;
            break;
        }
    }

    if (false == Result)
        Result = Super::MoveUpdatedComponentImpl(Delta, NewRotation, bUseSweep, OutHit, Teleport);

    return Result;
}
```

장애물 유무를 판단

Ignore: 장애물을 무시하고 진행
NeedImpact: 장애물을 인식하고 정지

인게임 기능 구현 - 플랫폼(발판) 무브먼트 (4/4)

특징: **네트워크 동기화**

플랫폼의 이동 경로 완료 시, **Server는 정지 함수 실행**
Multicast로 실행된 정지 함수에는 **Server와 Client의 핑 차이를 계산**
정지 할 시간을 TimerManager에게 전달

```
void UPlatformMovementComponent::Multicast_SendStopWaitControlPoint_Implementation(
    float StoppedTime, float StoppedDirection, bool IsStopped)
{
    CurrentTime = StoppedTime;
    CurrentDirection = StoppedDirection;
    bStopped = IsStopped;
    Velocity = FVector::ZeroVector;

    if (bStopped)
    {
        UWorld* World = GetWorld();
        if (nullptr == World)
            return;

        FTimerManager& TimerManager = World->GetTimerManager();
        if (TimerManager.IsTimerActive(mTimerWaitControlPoint))
        {
            TimerManager.ClearTimer(mTimerWaitControlPoint);
        }

        TimerManager.SetTimer(mTimerWaitControlPoint, mDelegateWaitControlPoint,
            mControlPointWaitingTime - GetPing(false), false);
    }
}
```

```
float UPlatformMovementComponent::GetPing(bool IsMilliSecond /*= true*/) const
{
    float Ping = 0.f;
    UWorld* World = GetWorld();

    APlayerController* PlayerController = World->GetFirstPlayerController();
    if (false == IsValid(PlayerController))
        return Ping;

    APlayerState* PlayerState = PlayerController->GetPlayerState<APlayerState>();
    if (false == IsValid(PlayerState))
        return Ping;

    Ping = PlayerState->ExactPing;

    if (false == IsMilliSecond)
        Ping *= 0.001f;

    return Ping;
}
```

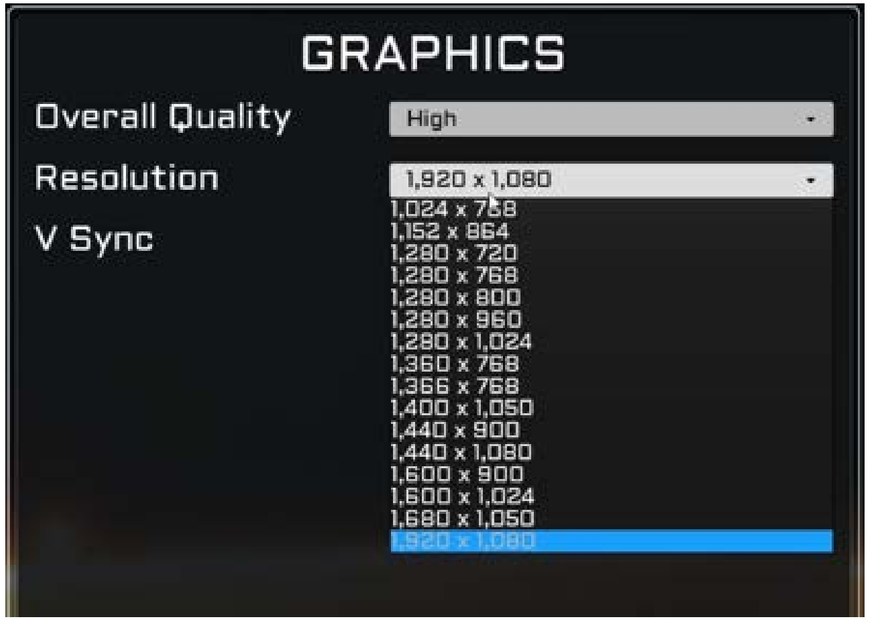
엔진의 기능을 이용해 Server와 Client간의 핑 Get
밀리초 단위를 초 단위로 바꿔서 Timer설정

환경설정 - 핵심 구현 내용(1/2)

환경설정: **UGameUserSettings 확장**
언리얼의 표준 설정 저장 방식을 상속받아 구현, **엔진 친화적인 구조 유지**

DefaultEngine.ini를 제어해 **그래픽 옵션을 변경**하거나,
사운드의 볼륨 조절 기능 구현

매직 스트링 방지를 위해 **ConfigSection과 ConfigKey**라는 **내부 클래스로 상수화**



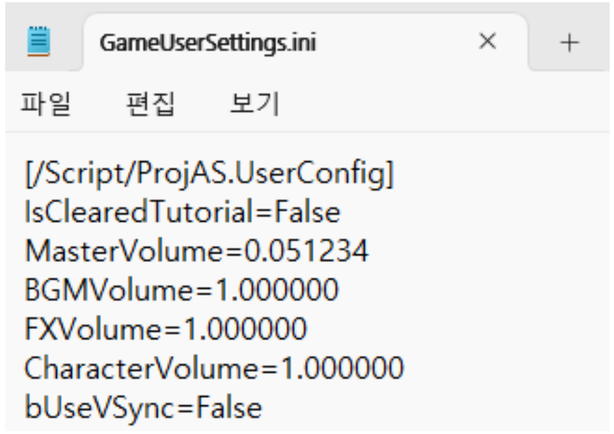

```
UCLASS(BlueprintType)
파생된 블루프린트 클래스 0개
class PROJAS_API UUserConfig : public UGameUserSettings
{
    GENERATED_BODY()

public:
    class ConfigSection { ... };

    class ConfigKey { ... };

public:
    bool InitUserConfig(UObject* WorldContextObject);

#pragma region Config Save/Load/Apply

public:
    UFUNCTION(BlueprintCallable)
    2개의 청사진 참조
    void ApplyAllSettings();

    virtual void ApplySettings(bool bCheckForCommandLineOverrides) override;

    virtual void LoadSettings(bool bForceReload = false) override;

    virtual void SaveSettings() override;

    virtual void ValidateSettings() override;
```

환경설정 - 핵심 구현 내용(2/2)

특징: USoundMix 활용한 **능동적인 오디오 제어**

UGameplayStatics::SetSoundMixClassOverride를 사용,
엔진의 오디오 믹싱 단계에서 볼륨을 제어

SoundClass 단위로 **볼륨 일괄 통제 기능 구현**

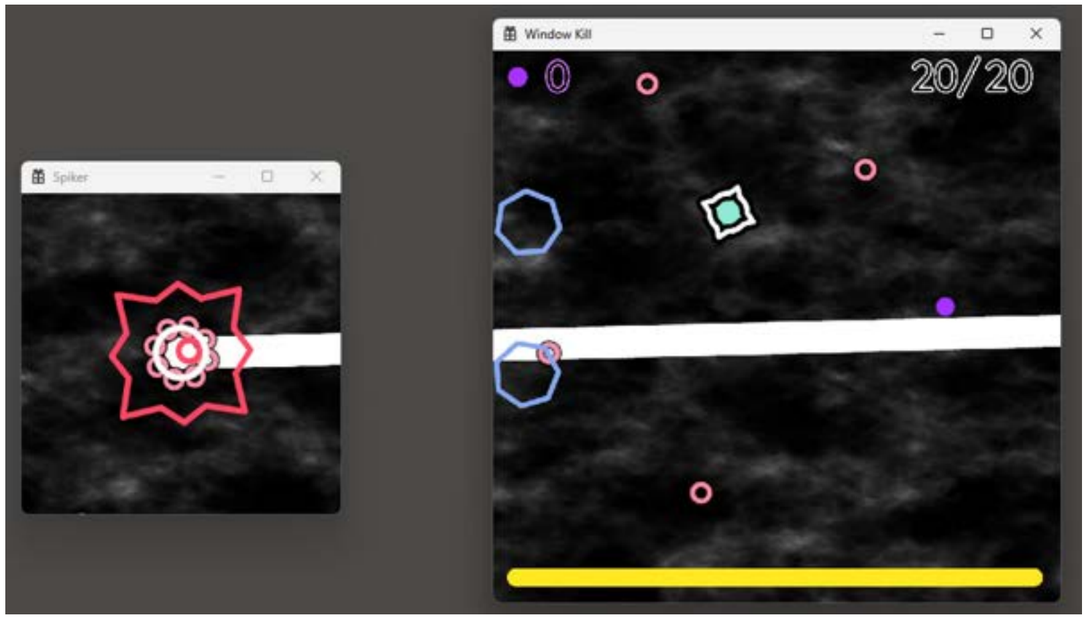



```cpp
void UUserConfig::ApplySoundMix(UObject* WorldContextObject, bool IsInit)
{
    if (false == IsValid(mSoundMix) ||
        nullptr == WorldContextObject)
        return;

    TArray<float> VolumeList = GetAllVolume();

    int32 Size = mSoundMix->SoundClassEffects.Num();

    for (int32 Index = 0; Index < Size; ++Index)
    {
        if (false == mSoundMix->SoundClassEffects.IsValidIndex(Index) ||
            false == VolumeList.IsValidIndex(Index))
            continue;

        if (IsInit)
        {
            mSoundMix->SoundClassEffects[Index].VolumeAdjuster = VolumeList[Index];
        }
        else
        {
            USoundClass* SoundClass = mSoundMix->SoundClassEffects[Index].SoundClassObject;

            if (nullptr != SoundClass)
            {
                UGameplayStatics::SetSoundMixClassOverride(WorldContextObject, mSoundMix,
                    SoundClass, VolumeList[Index]);
            }
        }
    }

    if (IsInit)
        UGameplayStatics::PushSoundMixModifier(WorldContextObject, mSoundMix);
}
```

Window Kill 모작

모작, 1인 개발

탄막 슈팅, 실시간 윈도우 리사이징 & 이동



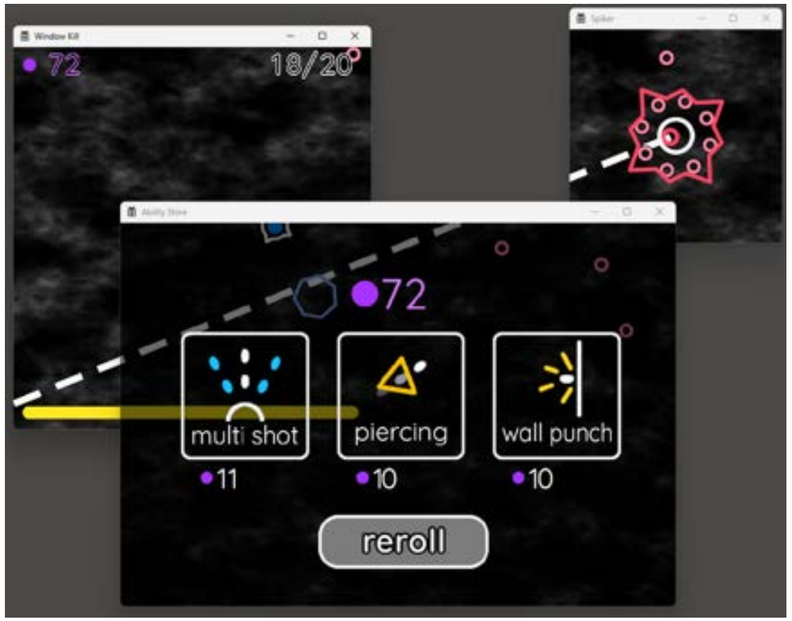
개발 이력

- 개발 기간 - 2025.02.27 ~ 2025.04.04
- 개발 인원 - 1명
- 사용 기술 - DirectX 11, Win32 API, HLSL
- 개발 도구 - Visual Studio 2022
- 버전 관리 - Git, Source Tree
- 일정 관리 - Notion

모작영상 링크

원본영상 링크

목차



일정 및 버전 관리

- Git을 이용해 버전관리를 사용

- Notion을 통해 일정관리와 작업일지를 작성
  - Notion의 캘린더 기능을 이용해 일정관리

보드 보기 | 표 | 캘린더 | 1개 더 보기 | 새로 만들기

↑ 날짜 | 상태 | + 필터

Aa 이름	날짜	상태	태그
리소스 분류	2025년 2월 28일 → 2025년 2월 28일	완료	리소스
멀티 윈도우 최적화	2025년 3월 1일 → 2025년 3월 4일	미분류	프레임워크
윈도우 창 관련 처리	2025년 3월 5일 → 2025년 3월 6일	완료	컨텐츠
윈도우 메시지 처리	2025년 3월 7일 → 2025년 3월 10일	완료	컨텐츠 프레임워크
플레이어 구현	2025년 3월 11일 → 2025년 3월 12일	완료	컨텐츠 데이터 테이블
몬스터 4종 구현	2025년 3월 12일 → 2025년 3월 14일	완료	컨텐츠 데이터 테이블
몬스터 육각형, 칠각형 버그	2025년 3월 13일 → 2025년 3월 13일	완료	버그
중간 점검	2025년 3월 15일	완료	릴리즈_테스트

03.12 구현

- StatusComponent 추가
- Movement Component에 LookTarget추가
  - 해당 방향으로 또는 오브젝트로 바라보는 기능
- Monster 추가 중
  - 부모 클래스 MonsterCharacter 추가
  - 삼각형 몬스터 Triangle 추가
    - 플레이어 추격 추가
- Texture Asset의 크기로 Mesh의 Scale을 조절하는 함수 추가
  - StaticMeshComponent 적용
  - Sprite Component 적용
- 마우스 좌표 Offset추가
  - 윈도우 영역과 클라이언트 영역의 갭 연산 추가

03.12 수정 및 보완

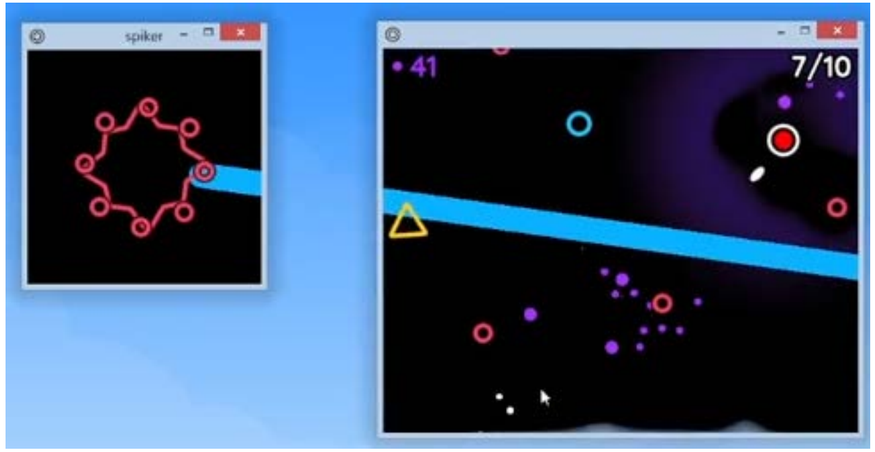
- SceneObject의 미사용 함수 제거
- GameManager 초기화 부분 수정
  - 기타 Manager의 초기화 순서 조절

좌표계 고정 및 충돌 - 핵심구현 내용 (1/4)

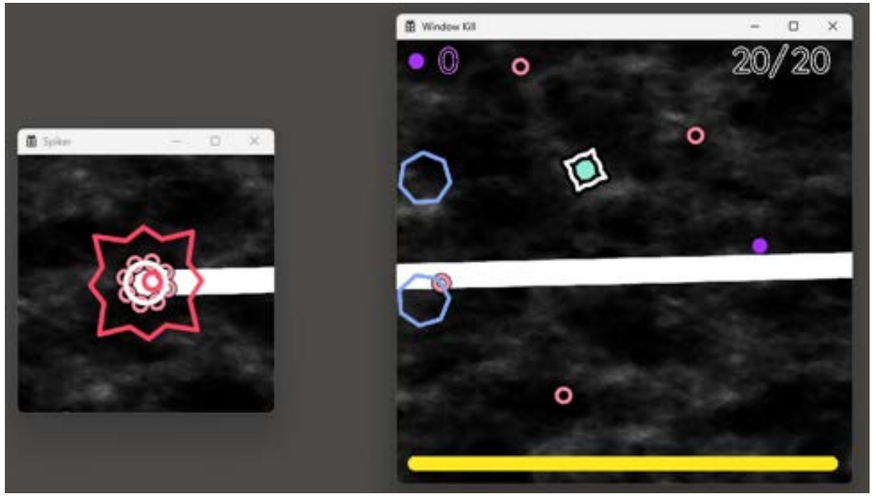
- **비대칭 투영 행렬**(창 이동에 따른 좌표계 동기화)
  - **그래픽스 파이프라인의 좌표계 변환을 이해하고 응용**
  - 게임 특성 상 **모니터의 해상도를 게임의 스테이지**처럼 활용
  - 창이 이동되어도 **원점은 고정**되어 있어, **충돌 감지 가능**

- **GameObject 충돌 예외처리**
  - 렌더되는 Object와 **실제 충돌의 위치는 다르다**
  - **원하는 모니터에서 게임을 실행, 렌더링 시 오프셋을 주어 해당 모니터에서 렌더링**

- **단일 투영 파이프라인 & UI 월드 좌표 역보정**
  - **비대칭 투영 행렬은 그대로 사용**
  - **UI는 윈도우에 고정되어 렌더링하기 위해 역보정 추가**



위: **원작** WindowKill 게임 플레이
아래: **모작** WindowKill 게임 플레이



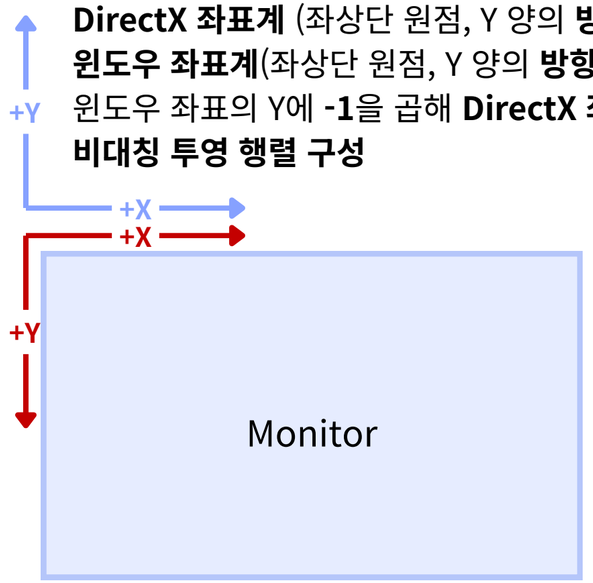
좌표계 고정 및 충돌 - 핵심구현 내용 (2/4)

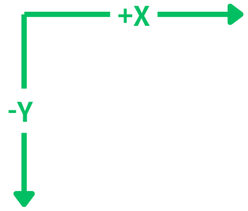
특징: **비대칭 투영 행렬**(창 이동에 따른 좌표계 동기화)

"모니터를 하나의 거대한 월드"로 간주하고,
윈도우 창을 월드를 들여다보는 '움직이는 카메라(Viewport)'로 변환

카메라의 물리적 이동(View Matrix의 단위행렬화) **대신,**
투영 행렬의 오프셋(Projection Matrix)**을 제어하는 방식을 채택.**
(월드의 원점과 뷰 행렬의 원점은 동일)

DirectX 좌표계 (좌상단 원점, Y 양의 **방향이 위,** 푸른색)
윈도우 좌표계(좌상단 원점, Y 양의 **방향이 아래,** 붉은색)
윈도우 좌표의 Y에 **-1**을 곱해 **DirectX 좌표계로 변환 후 역보정을 통해,**
비대칭 투영 행렬 구성





게임에 적용되는 **최종 좌표계**
(좌상단 원점, 녹색)

```
void CMultiWindow::MakeProjMatrix()
{
    RECT ClientRect;
    RECT WinRect;

    if (GetClientRect(mhWnd, &ClientRect) && GetWindowRect(mhWnd, &WinRect))
    {
        --- 중략 ---
        mWindowInfo.Location.x = ClientRect.left;
        mWindowInfo.Location.y = ClientRect.top;

        const float HalfSize = 0.5f;
①       const float ClientWidthHalf = (float)ClientRect.right * HalfSize;
        const float ClientHeightHalf = (float)ClientRect.bottom * HalfSize;
②       const float WinRectCenterX = (float)WinRect.left + ClientWidthHalf;
        const float WinRectCenterY = ((float)WinRect.top + ClientHeightHalf) * -1.f;

③       FMatrix MatProj = DirectX::XMMatrixOrthographicOffCenterLH(
            WinRectCenterX - ClientWidthHalf, WinRectCenterX + ClientWidthHalf,
            WinRectCenterY - ClientHeightHalf, WinRectCenterY + ClientHeightHalf,
            VIEW_NEAR, VIEW_FAR);

        CSceneManager::GetInst()->GetCameraManager()->SetProjMatrix(MatProj);
        if (mCBuffer)
        {
            mCBuffer->SetProj(MatProj);
        }
    }
}
```

①: 윈도우 창의 절반 크기
②: 모니터 상의 윈도우 창 좌표
③: 모니터 공간에서의 윈도우 중심점
윈도우가 움직일 때 렌더링 영역을 역보정

좌표계 고정 및 충돌 - 핵심구현 내용 (3/4)

특징: **단일 투영 파이프라인 & UI 좌표 역보정**

UI를 화면에 고정하기 위해, Window 좌표를 이용해 역보정.

World의 **Location을 직접적으로 제어**하기 때문에,
버튼을 비롯한 충돌과 GameObject의 같은 충돌 좌표 사용 가능

역보정을 제외한 **행렬 변환 과정은 GameObject와 동일**

$$FinalPos = Local \times World(Monitor) \times View(Identity) \times Proj$$

```cpp
void CMultiWindow::UpdateWidgetLocation(float DeltaTime)
{
    --- 중략 ---
    RECT WinRect;
    GetWindowRect(mhWnd, &WinRect);

    FVector2D Location;
    Location.x = (float)WinRect.left;
    Location.y = (float)WinRect.top * -1.f;

    int Size = (int)mExclusiveWidgetList.size();
    for (int i = 0; i < Size; ++i)
    {
        mExclusiveWidgetList[i]->SetRelativeLocation(Location);
    }
}
```

역보정을 통해, UI가 **윈도우에 고정**되어 렌더링

구분	Game Object	UI, HUD
좌표 기준	**Monitor Space** (절대 좌표)	**Window Space** (상대 좌표)
적용 방식	좌표 변환 없음	윈도우 위치만큼 **오프셋 추가** (WinRect)
결과	투영 행렬에 의해, **모니터에 고정**	투영 행렬의 이동을 **상쇄, 창에 고정**

좌표계 고정 및 충돌 - 핵심구현 내용 (4/4)

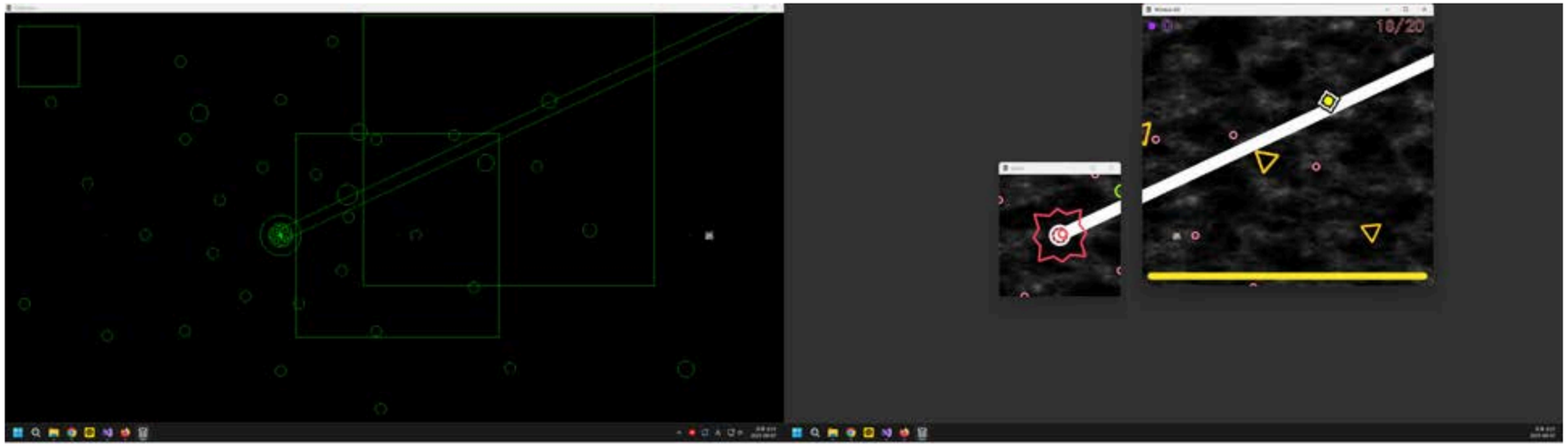
특징: **GameObject 충돌 예외처리**

윈도우 좌표계 특성상, 모니터의 해상도 만큼 좌표계의 최대값이 증가

쉐이더의 ConstantBuffer에 전달하기 직전,
모니터 해상도만큼의 Offset을 World행렬의 위치에 적용

결과: 원하는 모니터에서 게임을 실행 가능, 렌더링시 해당 모니터에서 실행, **충돌 로직은 주 모니터에서 충돌 감지**

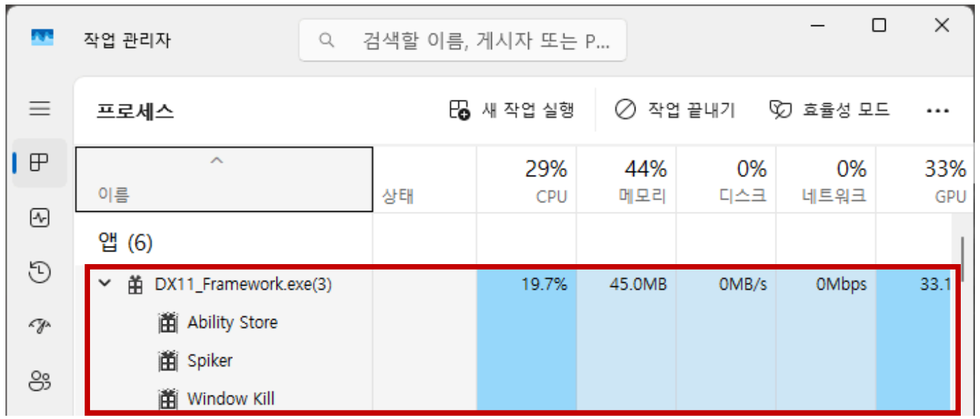
```
void CMeshComponent::Render()
{
    __super::Render();

#if USE_FORCED_MONITOR_OFFSET
    FVector2D Offset = CMultiWindowManager::GetInst()->GetMonitorOffset();
    mRenderWorldMatrix = GetWorldMatrix();
    mRenderWorldMatrix.v[3].x += Offset.x;
    mRenderWorldMatrix.v[3].y += Offset.y;

    mCBuffer_Transform->SetWorldMatrix(mRenderWorldMatrix);
#else
    mCBuffer_Transform->SetWorldMatrix(GetWorldMatrix());
#endif // USE_FORCED_MONITOR_OFFSET

    mCBuffer_Transform->SetViewMatrix(mScene->GetCameraManager()->GetViewMatrix());
    mCBuffer_Transform->SetProjMatrix(mScene->GetCameraManager()->GetProjectionMatrix());

    mCBuffer_Transform->UpdateBuffer();
}
```



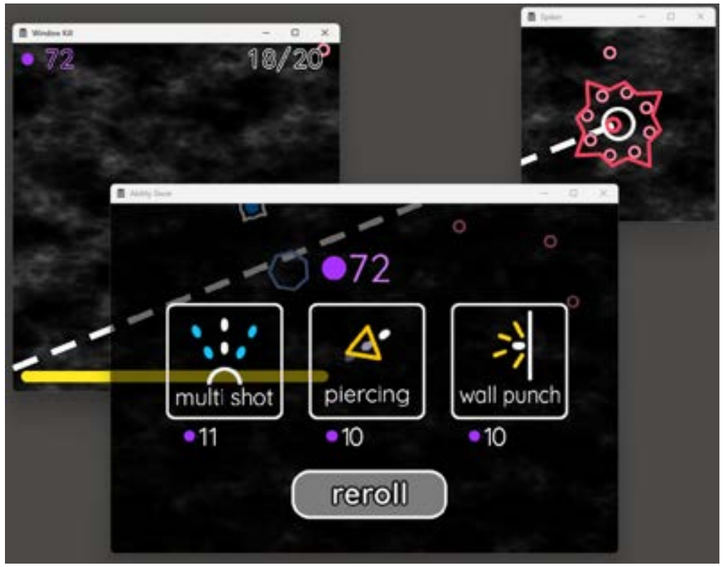
오른쪽 서브 모니터 WindowKill 플레이
충돌은 왼쪽의 메인 모니터의 좌표에서 수행
(충돌범위는 녹색의 WireFrame으로 표현)

다중 윈도우 - 핵심구현 내용 (1/5)

- 게임 특성상 **실시간으로 윈도우 창이 변화, 여러개의 윈도우 창 생성**
  - Main과 Sub Window로 분리
- 여러개의 윈도우 창을 관리하기 위해, **윈도우마다 고유한 타입**을 지정
- 윈도우 창의 최소화, 최대화, 닫기 버튼은 **구별된 WinProc 함수를 설정**
  - 윈도우마다 해당 **버튼에 대한 다른 로직**을 수행



작업 관리자에서 창이 3개 활성화 된 것을 확인 가능



다중 윈도우 - 핵심구현 내용 (2/5)

특징: **윈도우 클래스와 관리자 클래스 구현**

DirectX 11 환경에서 **단일 프로세스로 여러 개의 윈도우를 생성 및 렌더링 제어**
싱글톤 패턴을 적용해 매니저 클래스 단일화 및 접근성 향상

Window Update 코드 예시

```
void CMultiWindow::UpdateWindowArea(float DeltaTime)
{
    if (!mWindowArea || !mhWnd)
        return;

    RECT RC = mWindowArea->GetRect();
    FLong2D Offset = CMultiWindowManager::GetInst()->GetUsingMonitorStartPos();

    SetWindowPos(mhWnd, HWND_NOTOPMOST,
        RC.left + Offset.x, RC.top + Offset.y,
        RC.right, RC.bottom,
        NULL);

    if (mCBuffer)
    {
        FWinRect WinRect;
        WinRect.Location.x = RC.left;
        WinRect.Location.y = RC.top;
        WinRect.Resolution.x = RC.right;
        WinRect.Resolution.y = RC.bottom;

        mCBuffer->SetWindowRect(WinRect);
    }
}
```

mWindowArea 멤버 변수를 기반으로
윈도우의 위치(모니터 기준)로 이동 및 창 크기 Update

```
void CMultiWindowManager::Update(float DeltaTime)
{
    DeltaTime = CTimer::GetDeltaTime();

    mMainWindow->Update(DeltaTime); 메인 윈도우 Update

    _MultiWindowList::iterator iter = mSubWindowList.begin();
    _MultiWindowList::iterator iterEnd = mSubWindowList.end();
    while (iter != iterEnd)
    {
        if (!(*iter)->IsActivate() ||
            (*iter)->GetWindowHandle() == nullptr)
        {
            iter = mSubWindowList.erase(iter);
            iterEnd = mSubWindowList.end();
            continue;
        }

        (*iter)->Update(DeltaTime); 서브 윈도우 Update

        if ((*iter)->GetWindowType() == EWindowType::InGameUI ||
            (*iter)->GetWindowType() == EWindowType::OutGameUI)
            (*iter)->SetWindowZOrder();

        ++iter;
    }
}
```

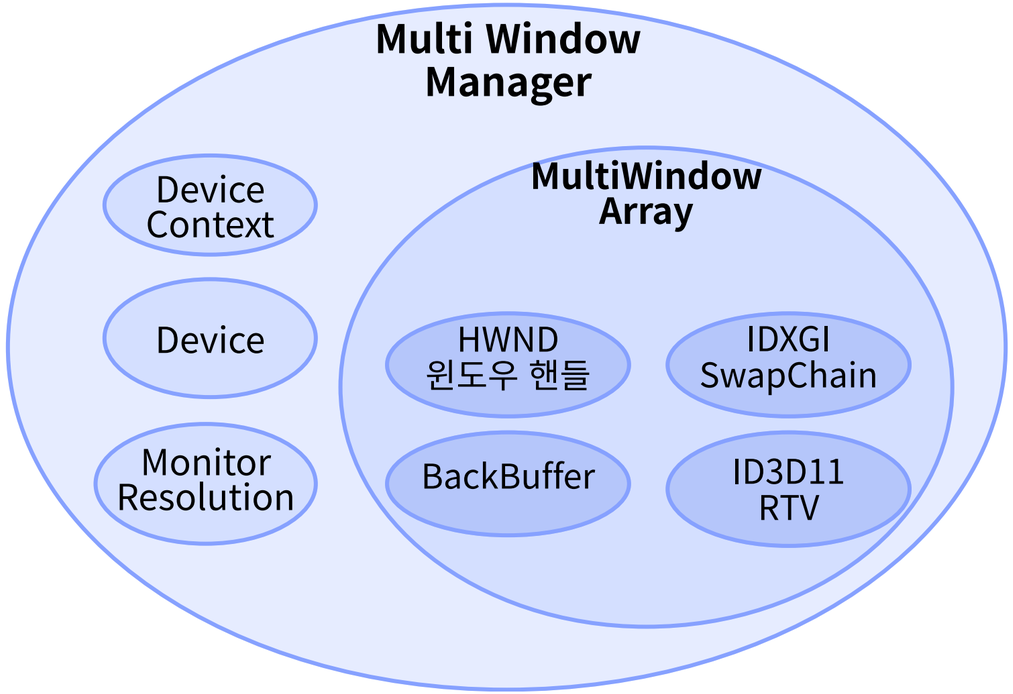
하나의 메인 윈도우와 여러 개의 서브 윈도우를 관리 및 루프

다중 윈도우 - 핵심구현 내용 (3/5)

특징: **실시간 윈도우 생성**
상점 윈도우, 보스 몬스터 표시용 윈도우 등 **실시간으로 윈도우 창 생성 및 관리**

Win32 API의 윈도우 생성과 DX11의 Device를 사용한 초기화는 동일
하지만, Device와 Context는 미리 생성된 MultiWindowManager에게 접근.

모든 윈도우는 Context를 공유하기 때문에 **메모리 절약과 리소스 공유 가능**
즉, **MultiWindow는 렌더링에 필요한 초기화**가 주를 이룬다



새로운 멀티 윈도우를 생성하는 함수

```
HRESULT CMultiWindow::CreateMultiWindow()
{
    if (FAILED(_Ready_Window()))
        return E_FAIL;

    ID3D11Device* Device = CMultiWindowManager::GetInst()->GetDevice();

    if (FAILED(_Ready_SwapChain(Device)))
        return E_FAIL;

    if (FAILED(_Ready_RenderTargetView(Device)))
        return E_FAIL;

    if (FAILED(_Ready_DepthStencilView(Device)))
        return E_FAIL;

    return S_OK;
}
```

MultiWindowManager의 **Device를 사용한 초기화가 필수**
그 외의 **초기화 과정은 일반적인 버퍼 초기화와 동일**

```
HRESULT CMultiWindow::_Ready_SwapChain(ID3D11Device* Device)
{
    if (!Device)
        return E_FAIL;
HRESULT CMultiWindow::_Ready_RenderTargetView(ID3D11Device* Device)
{
    if (!Device)
        return E_FAIL;
```

다중 윈도우 - 핵심구현 내용 (4/5)

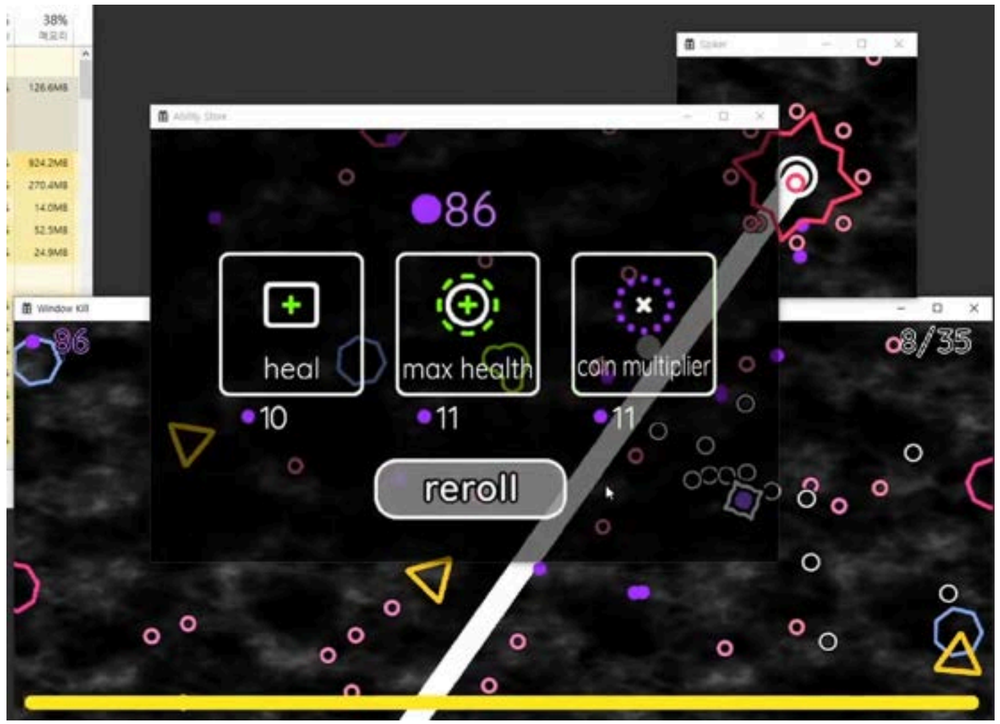
특징: **렌더링 파이프라인**

각 윈도우는 렌더링에 필요한 버퍼들을 소유

하나의 DeviceContext로 렌더링, 렌더 할 버퍼(SwapChain, BackBuffer)는
윈도우마다 다르기 때문에 **개별적인 화면 출력이 가능**

SetTarget과 RSSetViewports를 호출하여,
그리는 대상(백 버퍼)을 실시간으로 스위칭 하며 렌더링 → 윈도우 창이 겹쳐도 무방

```
void CMultiWindowManager::RenderCall()
{
    --- 중략 ---
    mMainWindow->ClearRenderTargetView(mContext, ClearColor);
    mMainWindow->ClearDepthStencilBuffer(mContext, Depth, Stencil);
    mMainWindow->SetTarget(mContext);

    mContext->RSSetViewports(1, &ViewPort);

    mMainWindow->MakeProjMatrix();
    mMainWindow->RenderSRV(mContext, mRenderTarget->GetSRV());
    mMainWindow->Render();
    ------------------------------------------------------
    _MultiWindowList::iterator iter = mSubWindowList.begin();
    _MultiWindowList::iterator iterEnd = mSubWindowList.end();
    while (iter != iterEnd)
    {
        (*iter)->ClearRenderTargetView(mContext, ClearColor);
        (*iter)->ClearDepthStencilBuffer(mContext, Depth, Stencil);
        (*iter)->SetTarget(mContext);
```



보스의 레이저가 여러 윈도우창에 렌더링 중

그려질 윈도우가 여러개일 뿐,
Main과 Sub의 렌더링 과정은 동일

다중 윈도우 - 핵심구현 내용 (5/5)

```
void CPlayerCharacter::ShowStore()
{
    CUserWidget* StoreWidget = mScene->GetUIManager()->FindWindowWidget(STORE_WIDGET);

    if (!StoreWidget)
        return;

    if (!StoreWidget->IsEnable())
    {
        FWinRect WinRect = {};
        WinRect.Location = CMultiWindowManager::GetInst()->GetMonitorResolution();

        WinRect.Location.x = WinRect.Location.x / 2 - WinRect.Resolution.x / 2;
        WinRect.Location.y = WinRect.Location.y / 2 - WinRect.Resolution.y / 2;

        CMultiWindow* Window = CMultiWindowManager::GetInst()->CreateSubWindow(
            WinRect, WINDOW_STORE_TITLE, EWindowType::InGameUI);

        Window->AddExclusiveWidgetList(StoreWidget);
    }
    else
    {
        CMultiWindow* Window = CMultiWindowManager::GetInst()->FindMultiWindow(
            WINDOW_STORE_TITLE);

        if (!Window)
            return;

        Window->Destroy();
    }
}
```

특징: **윈도우 버튼 반응 분리**

상점 윈도우, 보스 몬스터 표시용 윈도우 등 타입에 맞는 반응 구현
EWindowType로 작동하는 윈도우 프로시저를 분리

MainWndProc: 최소화, 최대화 불가, **닫기 클릭 시 게임 종료**
SubTargetWndProc: 최소화, 최대화, 닫기 불가
SubWndProc: 최소화, 최대화 불가**, 닫기 시 해당 윈도우만 삭제, 게임 일시정지**

```
HRESULT CMultiWindow::_Ready_Window()
{
    --- 중략 ---

    switch (mWinType)
    {
    case EWindowType::Main:
    case EWindowType::MasterRender:
        wcex.lpfnWndProc = MainWndProc;
        break;
    case EWindowType::Target:
        wcex.lpfnWndProc = SubTargetWndProc;
        break;
    case EWindowType::InGameUI:
    case EWindowType::OutGameUI:
    case EWindowType::End:
    default:
        wcex.lpfnWndProc = SubWndProc;
        break;
    }
```

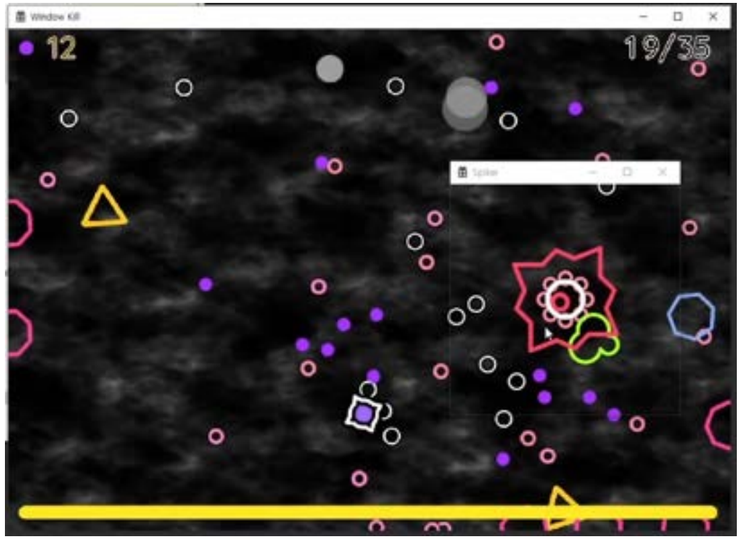
윈도우 리사이징 및 이동 - 핵심구현 내용 (1/2)

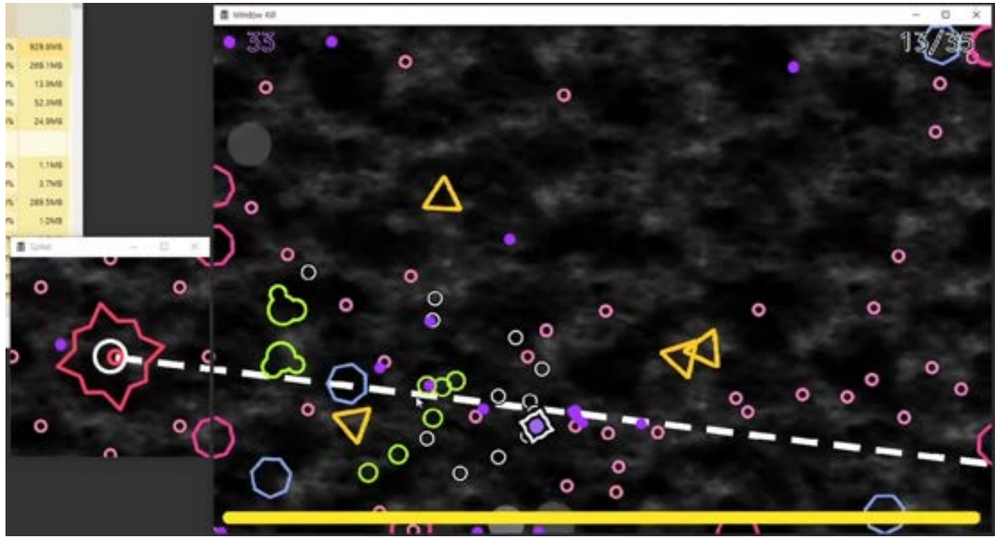
- **윈도우 창 실시간 변화**
  - **플레이어의 활동 영역**: 플레이어는 윈도우 창을 벗어나지 못한다.
  - **윈도우 리사이징**: 플레이어의 총알과 충돌 시, 변화
  - **윈도우 이동**: 플레이어의 총알과 충돌 시, 밀려나듯 이동

- **게임의 로직과 연결하기 위해 SceneObject를 상속한 객체를 이용**
  - **윈도우 크기와 동일**한 AABB 객체 구성
  - **AABB의 크기 및 이동에 맞춰 윈도우 변화**

```
float CWindowArea::TakeDamage(float Damage,
    CSceneObject* DamageCauser, CSceneComponent* DamageCauserComponent,
    const FDamageAdditiveInfo& DamageAdditiveInfo)
{
    if (nullptr == DamageCauser)
        return 0.f;

    mColliderScalingSpeed += DamageAdditiveInfo.WallPunchForce;
    mColliderMoveSpeed = DamageAdditiveInfo.WallMoveSpeed;

    FVector3D CauserLocation = DamageCauserComponent == nullptr ?
        DamageCauser->GetLocation() : DamageCauserComponent->GetWorldLocation();

    mColliderMoveDir = Math::Normalize(CauserLocation - GetLocation());
    --- 중략 ---
    return 0.0f;
}
```

윈도우 리사이징 및 이동 관련,
변수 초기화





윈도우 리사이징 및 이동 - 핵심구현 내용 (2/2)

특징: **윈도우 창 실시간 변화 (위치 및 크기)**

① **속도 변화:** 현재 속도의 일정 비율만큼 감소
② **크기 증가:** 크게 증가 후, 완만하게 감소
③ **크기 복원: 선형보간 사용**, 기본 윈도우 크기로 되돌아옴

```
void CWindowArea::UpdateCollider(float DeltaTime)
{
    if (mColliderMoveSpeed >= mColliderMoveSpeedMin)
    {
①       AddLocation(mColliderMoveDir * (mColliderMoveSpeed * DeltaTime));
        mColliderMoveSpeed -= mColliderMoveSpeed * (mColliderDecreaseMoveSpeedRatio * DeltaTime);
    }
    else
        mColliderMoveSpeed = 0.f;

    FVector2D BoxSize = mCollider->GetBoxSize();
    if (mColliderScalingSpeed > mColliderScalingSpeedMin)
    {
②       BoxSize += mRectSizeDir * (mColliderScalingSpeed * DeltaTime);
        mColliderScalingSpeed -= mColliderScalingSpeed * mColliderDecreaseScaleRatio * DeltaTime;
    }
    else
    {
③       mColliderScalingSpeed = 0.f;
        BoxSize = Math::Lerp(BoxSize, mDefaultBoxSize, DeltaTime);
    }

    mCollider->SetBoxSize(BoxSize);
    SetScale(BoxSize.x, BoxSize.y);
```

AABB 충돌체 정보를 **윈도우 좌표계로 변환**하는 함수

```
RECT CWindowArea::GetRect() const
{
    RECT Rect = {};
    FAABB2D BoxInfo = mCollider->GetBoxInfo();

    Rect.left = (LONG)ceilf(BoxInfo.Min.x);
    Rect.top = (LONG)ceilf(BoxInfo.Max.y) * -1;

    Rect.right  = (LONG)ceilf(mCollider->GetBoxSize().x);
    Rect.bottom = (LONG)ceilf(mCollider->GetBoxSize().y);

    return Rect;
}
```

```
void CMultiWindow::UpdateWindowArea(float DeltaTime)
{
    if (!mWindowArea || !mhWnd)
        return;

    RECT RC = mWindowArea->GetRect();
    FLong2D Offset = CMultiWindowManager::GetInst()->GetUsingMonitorStartPos();

    SetWindowPos(mhWnd, HWND_NOTOPMOST,
        RC.left + Offset.x, RC.top + Offset.y,
        RC.right, RC.bottom,
        NULL);
        //SWP_NOZORDER);
```

WindowArea객체의 **AABB 충돌체 정보를 기반**으로
WindowPos 재설정

범용 이벤트 처리기 - 핵심구현 내용 (1/2)

- **언리얼 엔진의 Delegate를 모방**한 구현
  - **Bind, Execute, Erase**(바인딩, 실행, 삭제)**등의 기능 지원**
  - **가변인자 템플릿 사용**으로, **다양한 타입 지원**
  - **람다 캡처 → std::function이라는 공통 타입**으로 저장
  - **일 대 다 바인드로 멀티캐스트 모방 구현**
  - **커플링 최소화**

EventFuncHandler 클래스 일부 설명

1: 이벤트 바인드, FuncName으로 특정 함수 제거 가능

2: 이벤트 실행, 바인드 된 **모든 함수 호출**

3: 바인드된 객체 관리
void*: 바인드한 객체의 주소

변수 선언 예시

```
protected:
    CEventFuncHandler<int> mEventHandlerCoinInfo;
    CEventFuncHandler<int, int> mEventHandlerHealthInfo;
    CEventFuncHandler<float> mEventHandlerLaserInfo;
    CEventFuncHandler<bool> mEventHandlerPlayerDead;
```

```
template <typename... Args>
class CEventFuncHandler
{
public:
    template<typename T>
    void BindEvent(T* Obj, const std::string& EventName,void(T::* Func)(Args...))
①   {
        EventFuncHandler::FBindInfo<Args...> BindInfo;

        BindInfo.FuncName = EventName;
        BindInfo.Func = [Obj, Func](Args... args)
            {
                (Obj->*Func)(std::forward<Args>(args)...);
            };

        mEventCallback.emplace_back(std::make_pair(Obj, BindInfo));
    }

②   void ExecuteEvent(Args... args)
    {
        auto iter    = mEventCallback.begin();
        auto iterEnd = mEventCallback.end();

        while (iter != iterEnd)
        {
            iter->second.Func(std::forward<Args>(args)...);
            ++iter;
        }
    }
protected:
    // 가변 인자를 받을 수 있도록 변경
③   std::list<std::pair<
        void*, EventFuncHandler::FBindInfo<Args...>>> mEventCallback;
```

범용 이벤트 처리기 - 핵심구현 내용 (2/2)

이벤트 처리기 실행 흐름 정리

①: Bind - Player에게 접근해 함수 바인딩

②: Execute - Player의 소지금 변동 함수 호출 → Delegate 실행

③: 바인드된 함수 호출 - 소지금 UI 변경, UI 색상효과 처리

소지금 UI 표시량 변화
(좌상단 보라색)

①
```
bool CCoinWidget::Init()
{
    --- 중략 ---
    CPlayerCharacter* Player = mScene->FindObjectFromName<CPlayerCharacter>("Player");
    if (Player)
    {
        Player->GetEventHandlerCoinInfo()->BindEvent<CCoinWidget>(
            this, GetName(), &CCoinWidget::UpdatePlayerCoin);
    }
```

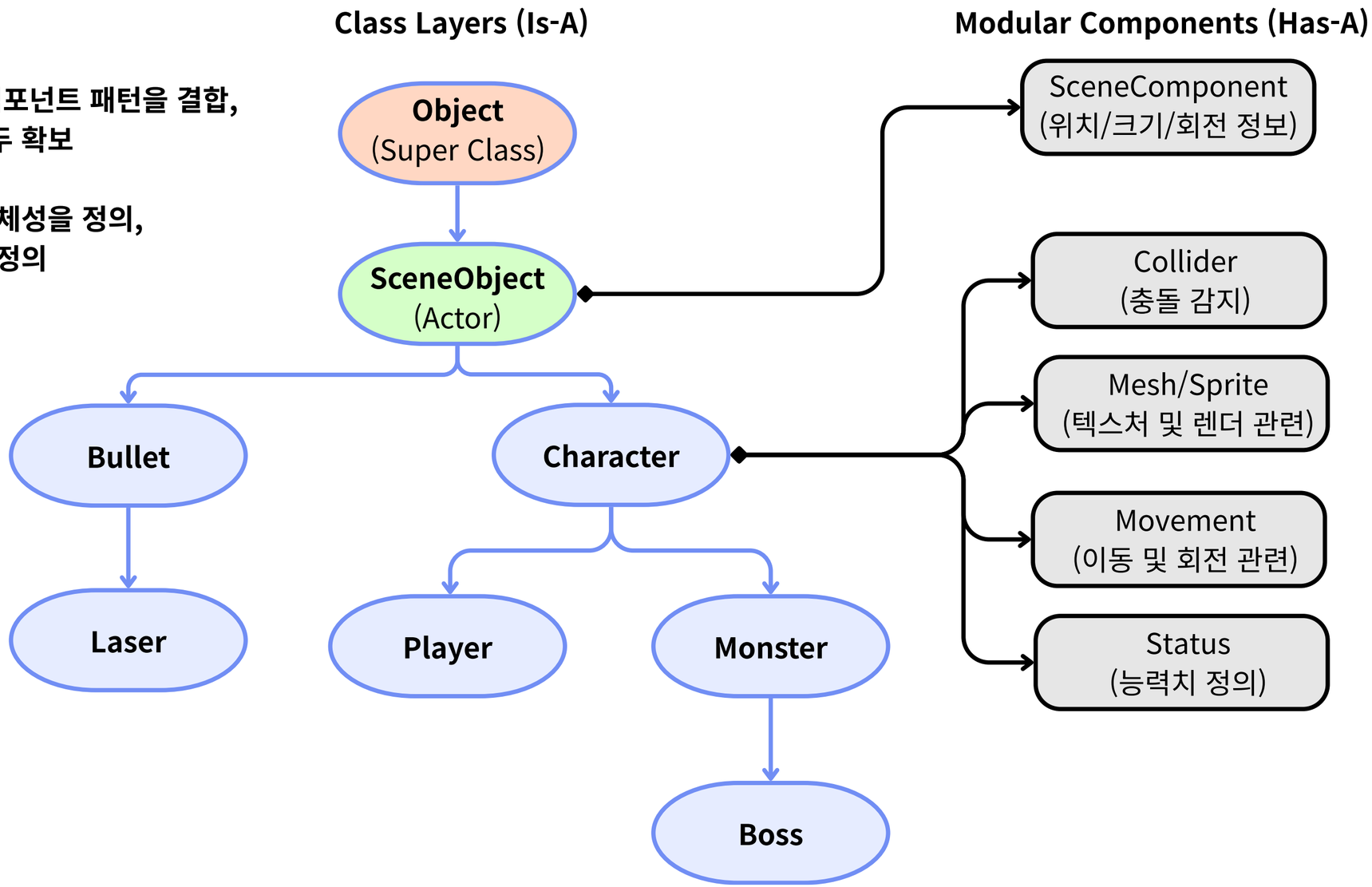
②
```
void CPlayerCharacter::AddGold(float Gold)
{
    __super::AddGold(Gold);
    mEventHandlerCoinInfo.ExecuteEvent((int)mStatusComp->GetGold());
    CPlayerDataManager::GetInst()->AddPlayerResultCoinCollected((int)Gold);
}

void CPlayerCharacter::DecreaseGold(float Gold)
{
    __super::DecreaseGold(Gold);
    mEventHandlerCoinInfo.ExecuteEvent((int)mStatusComp->GetGold());
}
```

③
```
void CCoinWidget::UpdatePlayerCoin(int Coin)
{
    mPlayerCoin->SetNumber(Coin);

    if (mCoinPrev < Coin)
    {
        mCurrentCoinStrokTint = _FColorRGBA::RGBA_Yellow;
        mCurrentCoinFillTint = _FColorRGBA::RGBA_Yellow;
    }
    mCoinPrev = Coin;
}
```

게임 오브젝트 설계 - 핵심구현 내용 (1/1)

상속을 통한 계층 설계와 컴포넌트 패턴을 결합,
확장성과 유지보수성을 모두 확보

클래스의 상속은 객체의 **정체성을 정의,**
컴포넌트는 객체의 **기능을 정의**



충돌 프로파일 - 핵심구현 내용 (1/1)

충돌 프로파일 구현

- **언리얼의 프로파일 모방**
- **Ignore**(충돌 무시), **Collision**(언리얼의 오버랩)기능 제공
- 충돌 채널 간 **필요한 충돌만 하도록 최적화**

충돌 채널간의 충돌 필터링 표 (충돌이 가능한 경우 O로 표시)

Collision Channel	Player	Monster	Player Bullet	Monster Bullet
Player	-	**O**	-	**O**
Monster	**O**	-	**O**	-
Player Bullet	-	**O**	-	-
Monster Bullet	**O**	-	-	-

```
void CCollisionManager::Collision(float DeltaTime)
{
    size_t  Size = mColliderList.size();
    for (size_t i = 0; i < Size - 1;)
    {
        CColliderComponent* Src = mColliderList[i];
        // early return: Src 충돌체 비활성화 유무 확인

        FCollisionProfile* SrcProfile = Src->GetProfile();
        for (size_t j = i + 1; j < Size;)
        {
            CColliderComponent* Dest = mColliderList[j];
            // early return: Dest 충돌체 비활성화 유무 확인

            //상호 간 무시(Ignore) 설정인지 확인
            else if (SrcProfile->Interaction[DestProfile->Channel] == ECollisionInteraction::Ignore ||
                DestProfile->Interaction[SrcProfile->Channel] == ECollisionInteraction::Ignore)
            {
                ++j;
                continue;
            }

            // 충돌 진행
            FVector3D   HitPoint;
            if (Src->Collision(HitPoint, Dest))
            {
                // 최초 충돌 유무
                if (!Src->CheckCollisionObject(Dest))
```

최적화 고려

각 충돌체의 **비활성화 유무로 Early Return** 수행
상호간 충돌 유무로 **필터링 추가**, **필요한 충돌**만 수행

HLSL - 핵심구현 내용 (1/2)

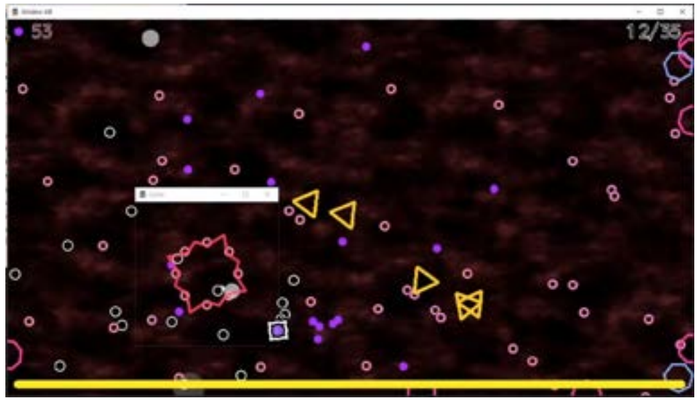
백그라운드 HLSL

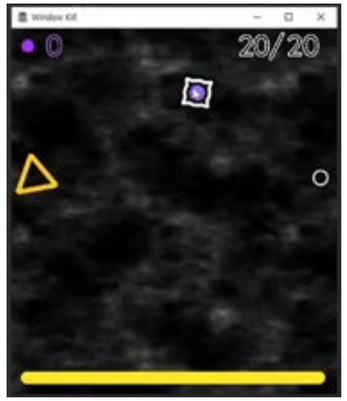
- 3장의 **노이즈 텍스처를 하나의 색상정보로 합성**
- **텍스처 각각의 UV를 서로 다른 속도 스크롤링, 절차적인 움직임 구현**
- 보스의 패턴 표시 FX로 **백그라운드 색상 변경**

CPU 에서 Tint값 조절 → GPU로 전송

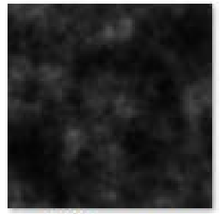
```
void CBackGround::LerpTint(float DeltaTime)
{
    mBackGroundSprite->SetTint(mBackGroundTint.x, mBackGroundTint.y, mBackGroundTint.z);

    mBackGroundTint = Math::Lerp(mBackGroundTint, _FColorRGBA::RGBA_White,
        DeltaTime * mLerpScale);
}
```

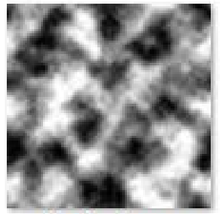
인게임 캡처

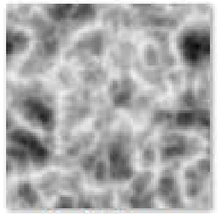





```
PS_OUTPUT_SINGLE_COLOR PS_PointSpriteBackGround(GS_OUTPUT_BACKGROUND input)
{
    PS_OUTPUT_SINGLE_COLOR output = (PS_OUTPUT_SINGLE_COLOR) 0;

    float4 Diffuse = gBaseTexture.Sample(gBaseSampler, input.DiffuseUV);
    Diffuse *= gNoise1Texture.Sample(gBaseSampler, input.Noise1UV);
    Diffuse *= gNoise2Texture.Sample(gBaseSampler, input.Noise2UV);

    Diffuse *= 2.f;

    output.Color = Diffuse * gSpriteTint;

    return output;
}
```


Diffuse.png


Noise1.png

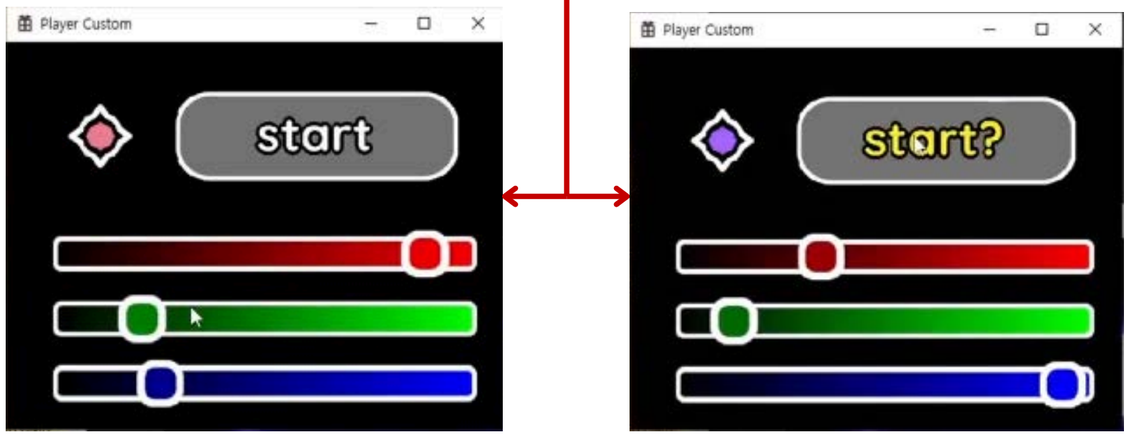

Noise2.png

노이즈 텍스처를 활용한 **절차적(Procedural) 배경 생성**
Time 기반의 **UV 스크롤링(UV Scrolling)으로 동적 연출**

HLSL - 핵심구현 내용 (2/2)

```
void CPlayerSelectWidget::Update(float DeltaTime)
{
    __super::Update(DeltaTime);

    mPlayerTint.x = mColorSlideRed->GetSlideValue();
    mPlayerTint.y = mColorSlideGreen->GetSlideValue();
    mPlayerTint.z = mColorSlideBlue->GetSlideValue();

    mPlayerPreviewColored->SetTint(mPlayerTint.x, mPlayerTint.y, mPlayerTint.z);
}
```

메인메뉴 레벨
플레이어 색상 변경

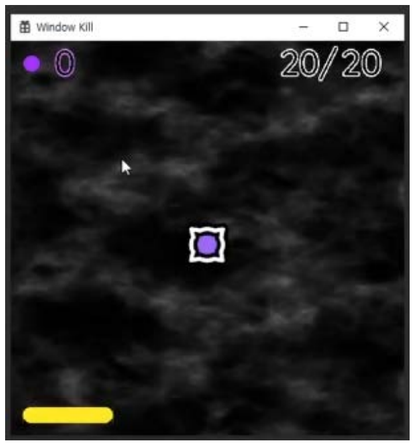


캐릭터 색상 커스텀

- SlideUI를 이용해, **RGB 색상 정보를 정의**
- PlayerDataManager를 통해 **색상 정보 캐싱**
- **인게임의 플레이어에게 RGB 색상 적용**

```
bool CPlayerCharacter::PreInit()
{
    --- 중략 ---

    _FColorRGB Tint = CPlayerDataManager::GetInst()->GetPlayerTint();
    mSpriteBack->SetTint(Tint);
    AttachChildComponent(mSpriteBack);
```

인게임 레벨 적용



커스텀 수학 라이브러리 - 핵심구현 내용 (1/1)

SIMD 가속 기반 커스텀 수학 라이브러리

- **가독성**: FVector, FMatrix 등 **직관적인 구조체 제공**
- **성능: DirectXMath (SIMD) 기반의 고속 병렬 연산**
- **사용성: 길이, 각도, 정규화, 보간, 외적 등 함수 구현**
- **변환:** Convert, XMLoad, XMStore 함수를 통해 이 둘을 연결

MathLibrary 정의 일부

```
namespace Math
{
    INLINE DirectX::XMVECTOR Convert(const FVector2D& v);
    INLINE DirectX::XMVECTOR Convert(const FVector3D& v);
    INLINE DirectX::XMVECTOR Convert(const FVector4D& v);

    INLINE float AngleDir(const FVector2D& LookDir, const FVector2D& TargetDir);
    INLINE float AngleDir(const FVector3D& LookDir, const FVector3D& TargetDir);

    INLINE float Angle(const FVector2D& TargetLocation, const FVector2D& Location,
    INLINE float Angle(const FVector3D& TargetLocation, const FVector3D& Location,

    INLINE int  Clamp(int Value, int Min, int Max);
    INLINE float Clamp(float Value, float Min, float Max);

    INLINE float Length(const FVector2D& v);
    INLINE float Length(const FVector3D& v);
    INLINE float Length(const FVector4D& v);

    INLINE FVector2D Normalize(const FVector2D& v);
    INLINE FVector3D Normalize(const FVector3D& v);
    INLINE FVector4D Normalize(const FVector4D& v);
```

MathLibrary 내부에서
XMVECTOR로 변환, SIMD 가속 사용

```
DirectX::XMVECTOR Math::Convert(const FVector3D& v)
{
    DirectX::XMVECTOR float3 = DirectX::XMLoadFloat3((DirectX::XMFLOAT3*)&v);
    return float3;
}
```

```
struct FVector3D
{
    static FVector3D ZeroVector;
    static FVector3D OneVector;
    static FVector3D Axis[EAxis::End];

    float   x = 0.f;
    float   y = 0.f;
    float   z = 0.f;
```

3차원 벡터, 4x4 행렬 등 **구조체 지원**,
연산자 오버로딩으로 직관적인 사용
(FMatrix 메모리 정렬)

```
__declspec(align(16)) union FMatrix
{
    static FMatrix Identity;

    DirectX::XMMATRIX   m;

    struct
    {
        float   _11, _12, _13, _14;
        float   _21, _22, _23, _24;
        float   _31, _32, _33, _34;
        float   _41, _42, _43, _44;
    };

    FVector4D   v[4];
```

```
    FVector3D operator * (const FVector3D& v)   const;
    FVector3D operator * (float Value)  const;
    FVector3D operator * (double Value) const;
    FVector3D operator * (int Value)    const;

    // *=
    const FVector3D& operator *= (const FVector3D& v);
    const FVector3D& operator *= (float Value);
    const FVector3D& operator *= (double Value);
    const FVector3D& operator *= (int Value);
```

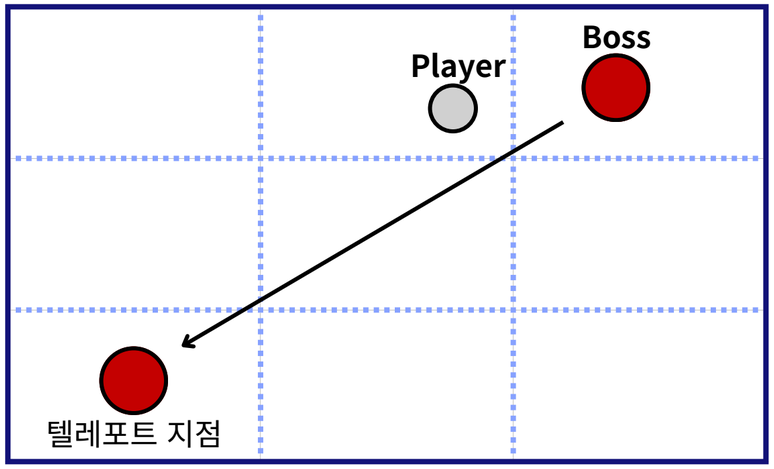
인게임 구현 (보스 순간이동) - 핵심구현 내용 (1/1)

보스 AI: 공간 분석 기반 텔레포트

①: 보스는 모니터 전체 해상도를 전장으로 사용,
모니터의 해상도를 3x3, 총 9개의 영역으로 분할

②: **3x3으로 분할된 영역의 중앙 위치를 기준**으로,
플레이어와 가장 먼 위치 탐색.
원거리 저격 패턴을 사용하기 유리한 위치 점유

③: 찾은 위치에서 **랜덤한 오프셋 추가**,
자연스러운 움직임 구현



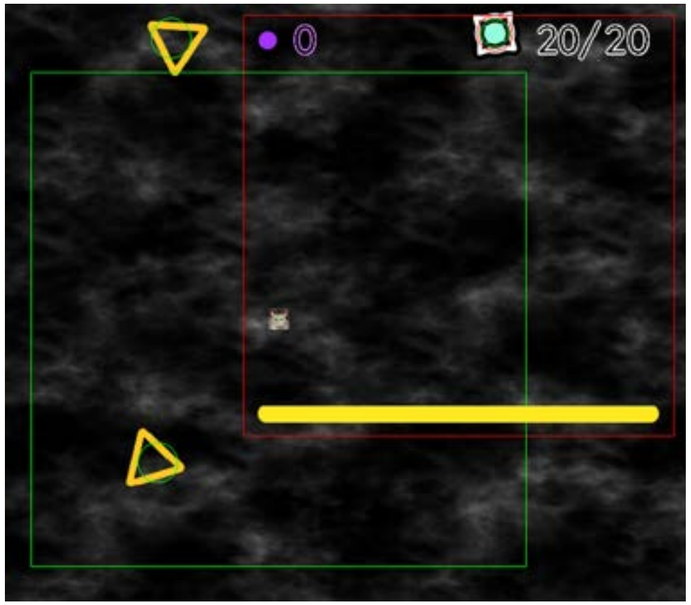

```
void CBossSpiker::UpdateTeleport(float DeltaTime)
{
    --- 중략 ---
①   FLong2D Resolution = CMultiWindowManager::GetInst()->GetMonitorResolution();
    FVector2D ConvertedResolution = FVector2D((float)Resolution.x, (float)Resolution.y * -1.f);

    //화면을 3x3분할,
    FVector2D SplitSize = FVector2D(ConvertedResolution.x / (float)MONITOR_SPLIT_X,
        ConvertedResolution.y / (float)MONITOR_SPLIT_Y);
    FVector2D SplitCenter = SplitSize * 0.5f;
    FVector2D TargetLocation = mTargetObject->GetLocation();

②   // 플레이어와 가장 먼 위치 탐색
    float FarMonitorDist = FLT_MIN;
    FVector3D FarMonitorArea;
    for (int i = 0; i < MONITOR_SPLIT_Y; ++i)
    {
        for (int j = 0; j < MONITOR_SPLIT_X; ++j)
        {
            FVector2D MonitorArea = SplitCenter;
            MonitorArea += SplitSize * FVector2D((float)j, (float)i);

            float Dist = Math::Distance(MonitorArea, TargetLocation);
            if (Dist >= FarMonitorDist)
            {
                FarMonitorArea = MonitorArea;
                FarMonitorDist = Dist;
            }
        }
    }

③   // 가장 먼 위치에서 보스 텍스처 스케일의 절반만큼 플레이어 쪽으로 이동
    FVector2D TextureSize = mMeshComp->GetTextureSize(0);
    FarMonitorArea += Math::Normalize(TargetLocation - FarMonitorArea) * TextureSize;
    // 랜덤 오프셋 추가
    int RandomRange = (int)TextureSize.x / 2;
    FarMonitorArea.x += (float)((rand() % RandomRange) - (RandomRange / 2));
    RandomRange = (int)TextureSize.y / 2;
    FarMonitorArea.y += (float)((rand() % RandomRange) - (RandomRange / 2));

    SetLocation(FarMonitorArea);
    SetState(ESpikeState::Combat);
}
```

인게임 구현 (몬스터 스포너) - 핵심구현 내용 (1/1)

유연한 몬스터 스폰 시스템

①: **ESpawnLoopType** 스폰 타입 지정
②: 스폰 주기에 **난수 적용으로 랜덤한 간격 생성**
③: **스폰 영역 내에서, 가우스 분포를 활용해 몬스터 위치 지정**

스폰 영역의 범위를 WireFrame으로 표시 (녹색)



①
```
enum class ESpawnLoopType : unsigned char
{
    Loop,           // 생성된 오브젝트 상관없이 계속 생성
    OnceDestroy,    // 생성되고 스폰포인트 제거
    CountDestroy,   // 일정 수량만큼 생성하고 스폰포인트 제거
    End,
};
```

②
```
void CMonsterSpawner::SetSpawningOption(float ActivateSpawnTime,
    float SpawnCycleMax, float SpawnCycleRandomInterval, FVector2D SpawnRange)
{
    mActivateSpawnTime = ActivateSpawnTime;
    mSpawnCycleMax = SpawnCycleMax;
    mSpawnCycleRandomInterval = SpawnCycleRandomInterval;
    mSpawnAreaRange = SpawnRange;

#ifdef _DEBUG
    mSpawnArea->SetBoxSize(mSpawnAreaRange);
#endif
}
```

③
```
void CMonsterSpawner::Spawn(float DeltaTime)
{
    --- 중략 ---
    FVector2D RandomLocation = {};
    RandomLocation.x = CRandomGenerator::GetRandom(-mSpawnAreaRange.x, mSpawnAreaRange.x);
    RandomLocation.y = CRandomGenerator::GetRandom(-mSpawnAreaRange.y, mSpawnAreaRange.y);

    FVector3D Location = GetLocation();
    Location.x += RandomLocation.x;
    Location.y += RandomLocation.y;
    Monster->SetLocation(Location);
}
```

경청해 주셔서 감사합니다.

신정호

시행착오를 두려워 하지않고, 어떠한 기술적 난관도 흔들림 없이 해결하는
뿌리가 튼튼한 프로그래머가 되겠습니다.
감사합니다.

Git Hub Link

Tel +82-10-2507-8645
E-mail sjh98091@gmail.com