


2단계_01. 언리얼 엔진 게임 제작 기초

📅 수강일	@2023/06/21
➦ 이름	 전영재
🔍 멘토	Min-Kang Song, 현웅 최
👥 멘티	
✨ 작성 상태	Done
🕒 단계	
☑ 강의 시청 여부	<input checked="" type="checkbox"/>
☑ 이수 여부	<input type="checkbox"/>

Contents



- 언리얼 엔진 게임 제작 기초
 - [1] 게임 프레임워크(Gameplay Framework)란?
 - 게임 콘텐츠의 구조
 - 게임 = 월드 + 모드 + 상태
 - 기믹 = 트리거 + 스폰 + 물리
 - 플레이어 = 입력 + 카메라 + HUD + 상태
 - 폰 = 이동+모션
 - [2] 게임의 5단계 제작 과정
 - 최종예제 확장 목표
 - [3] 모듈명 추가를 통해 Header 경로 입력 간소화
 - [4] 게임모드의 구성
 - 0n. 언리얼 엔진 게임 제작 기초 강의 과제

언리얼 엔진 게임 제작 기초



강의 목표

- 언리얼 게임 프레임워크를 활용한 게임 제작 방식의 이해
- 강의 예제에 사용되는 프로젝트와 기본 C++ 클래스 생성
- 향후 운영될 C++ 프로젝트 운영 규칙의 이해

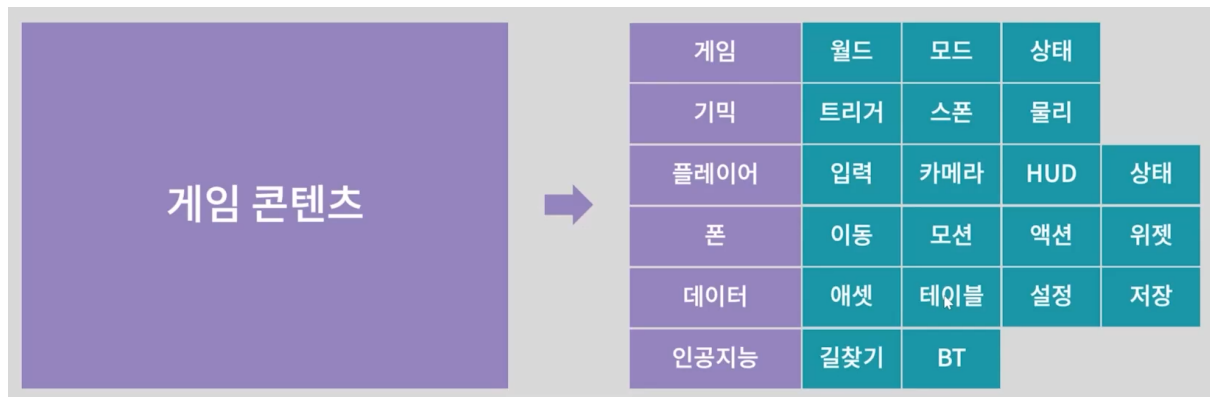
[1] 게임 프레임워크(Gameplay Framework)란?

게임 제작을 위해 필요한 구성 요소를 의미한다.

언리얼엔진에서는 자체적으로 설계한 프레임워크를 제공하고 있으며, 이를 파악하고 확장하여 게임을 제작하는 것을 권장하고 있으므로, 이번 시간에는 게임 콘텐츠의 구조와 구성요소를 파악하도록 해보자.

게임 콘텐츠의 구조

콘텐츠 = 게임 + 기믹 + 플레이어 + 폰 + 데이터 + 인공지능



게임 = 월드 + 모드 + 상태

월드 : 말 그대로 게임 콘텐츠를 담기 위해 제공되는 **가상의 공간**을 의미한다. 더 자세히 설명하자면, 공간(Transform), 진행(Tick), 시간(Time), 세계규칙(World Setting) 등을 관리한다.

월드	말 그대로 게임 콘텐츠를 담기 위해 제공되는 가상의 공간 을 의미한다. 자세히 설명하자면, 게임 내의 공간(Transform), 진행(Tick), 시간(Time), 환경설정(World Setting) 등을 관리한다.
모드	게임 규칙을 지정하고 판정하는 최고 관리자의 책임을 지닌다.
상태	게임의 현재 상태 (데이터) 등을 관리한다.

기믹 = 트리거 + 스폰 + 물리

게임 진행을 위한 이벤트를 발생시키는 사물객터를 의미한다. 주로 이벤트 발생을 위한 충돌 영역을 설정하는데, 이를 '**트리거**' 라고 한다. 게임은 트리거를 통해 캐릭터와 상호 작용하고, 월드에 액터를 스폰해 콘텐츠를 전개한다.

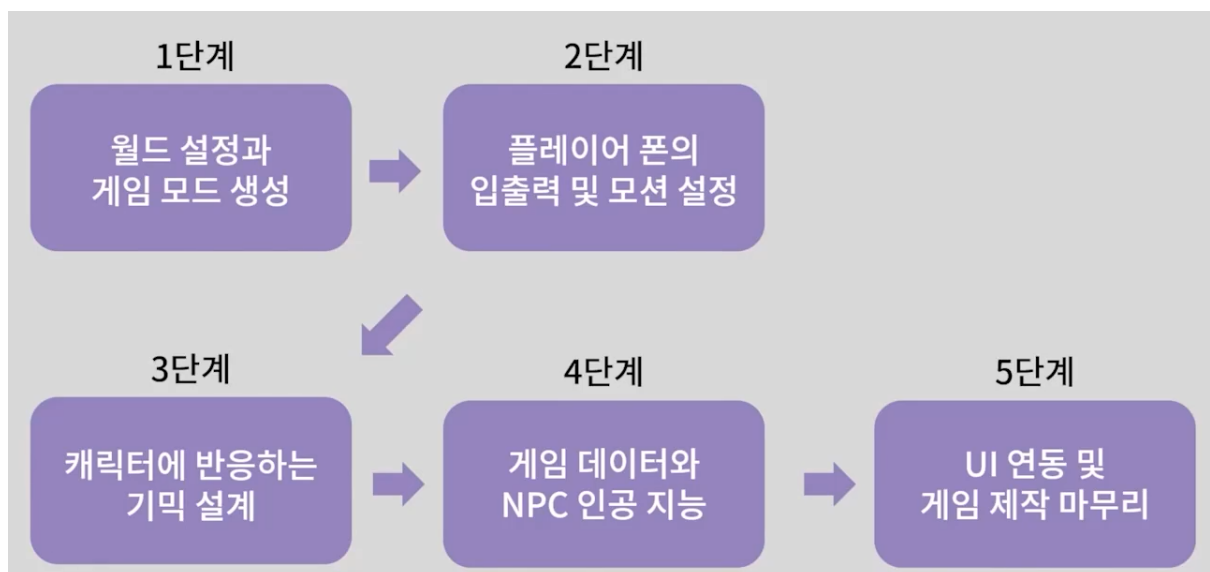
플레이어 = 입력 + 카메라 + HUD + 상태

게임에 입장한 사용자 액터를 의미한다. 게임 모드의 로그인 판정을 통해, 사용자가 게임 월드에 입장해야 플레이어가 생성된다. 사용자와 1:1로 대응되기에, 사용자와의 최종 커뮤니케이션을 담당하게 된다.

폰 = 이동+모션

플레이어가 빙의해 조종하는 액터를 의미한다. 빙의(Possess)를 통해 플레이어와 연결되며, 사용자 입력의 실제 처리를 맡는다.

[2] 게임의 5단계 제작 과정



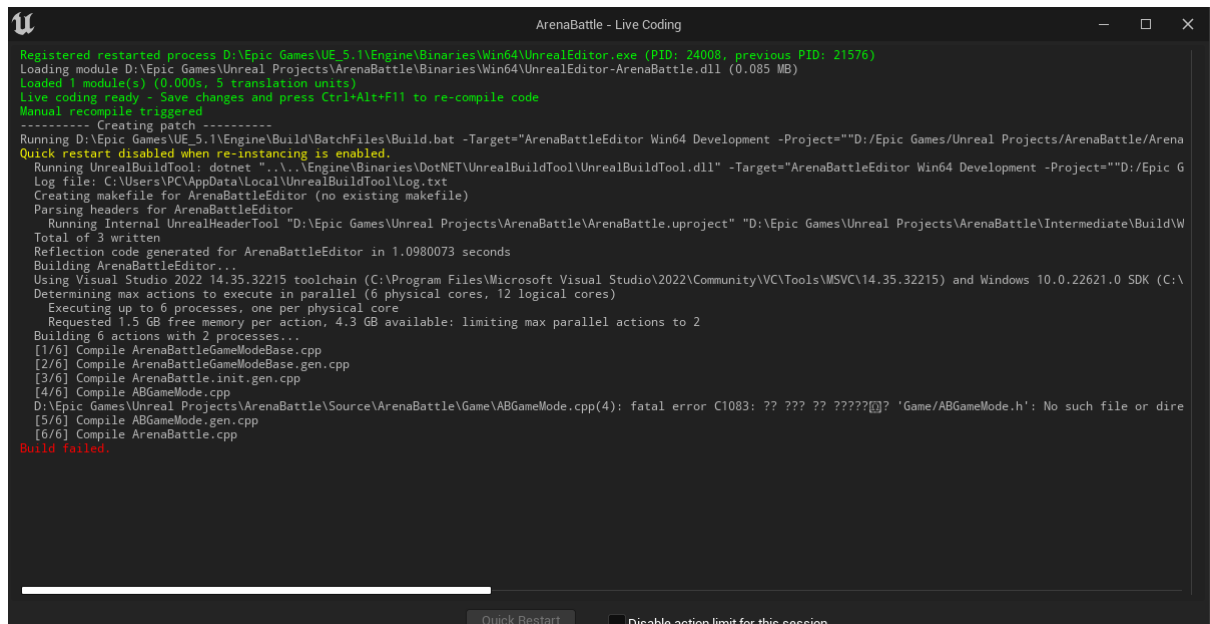
게임은 다음과 같이 5단계의 작업 과정을 통해 진행되며, 우리의 목표는 위의 5개 단계를 모두 C++를 사용해 제작하는 것이다.

최종예제 확장 목표

1. 기본 템플릿을 사용하지 않고 직접 제작
2. 두 가지 카메라 모드 제공 (솔더뷰, 탑뷰)
3. 액션 입력을 통한 콤보 공격 구현
4. 아이템 애셋을 활용한 무기 상자의 구현
5. NPC레벨 정보를 스프레드 시트로 관리
6. 플레이어와 전투하는 NPC 인공지능 구현
7. UI 연동과 게임의 마무리

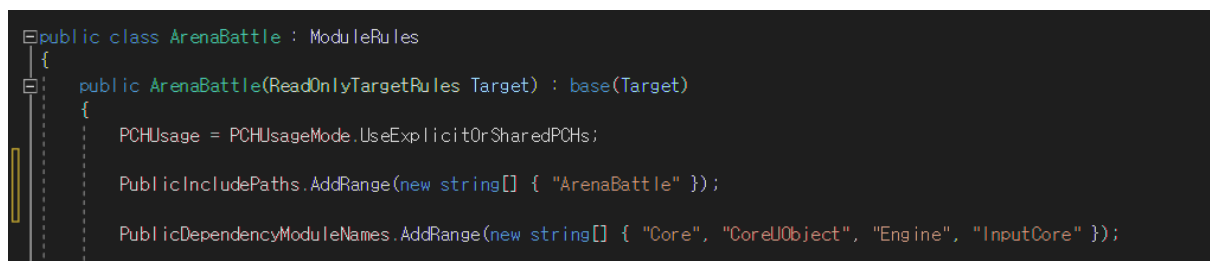
[3] 모듈명 추가를 통해 Header 경로 입력 간소화

C++클래스를 추가하면 다음과 같이 컴파일 과정에서 에러가 발생한다. 내용을 읽어보면 `Game/ABGameMode.h` 라는 헤더 경로를 읽지 못했다는 것. 이를 해결하기 위해서는 `ArenaBattle/Game/ABGameMode.h` 와 같이 해당 모듈이 존재하는 정확한 경로를 입력해야 한다. 그러나 매번 수작업을 거치는 것은 불편하므로, 해당 내용 **모듈의 경로를 프로젝트 Build.cs에 입력하여** 간편화 할 수 있다.



```
Registered restarted process D:\Epic Games\UE_5.1\Engine\Binaries\Win64\UnrealEditor.exe (PID: 24008, previous PID: 21576)
Loading module D:\Epic Games\Unreal Projects\ArenaBattle\Binaries\Win64\UnrealEditor-ArenaBattle.dll (0.085 MB)
Loaded 1 module(s) (0.000s, 5 translation units)
Live coding ready - Save changes and press Ctrl+Alt+F11 to re-compile code
Manual recompile triggered
----- Creating patch -----
Running D:\Epic Games\UE_5.1\Engine\Build\BatchFiles\Build.bat -Target="ArenaBattleEditor Win64 Development -Project=""D:\Epic Games\Unreal Projects\ArenaBattle\Arena
Quick restart disabled when re-instancing is enabled.
Running UnrealBuildTool: dotnet "D:\Epic Games\UE_5.1\Engine\Binaries\DotNET\UnrealBuildTool\UnrealBuildTool.dll" -Target="ArenaBattleEditor Win64 Development -Project=""D:\Epic G
Log file: C:\Users\PC\AppData\Local\UnrealBuildTool\Log.txt
Creating makefile for ArenaBattleEditor (no existing makefile)
Parsing headers for ArenaBattleEditor
Running Internal UnrealHeaderTool "D:\Epic Games\Unreal Projects\ArenaBattle\ArenaBattle.uproject" "D:\Epic Games\Unreal Projects\ArenaBattle\Intermediate\Build\W
Total of 3 written
Reflection code generated for ArenaBattleEditor in 1.0980073 seconds
Building ArenaBattleEditor...
Using Visual Studio 2022 14.35.32215 toolchain (C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.35.32215) and Windows 10.0.22621.0 SDK (C:\
Determining max actions to execute in parallel (6 physical cores, 12 logical cores)
Executing up to 6 processes, one per physical core
Requested 1.5 GB free memory per action, 4.3 GB available: limiting max parallel actions to 2
Building 6 actions with 2 processes...
[1/6] Compile ArenaBattleGameModeBase.cpp
[2/6] Compile ArenaBattleGameModeBase.gen.cpp
[3/6] Compile ArenaBattle.init.gen.cpp
[4/6] Compile ABGameMode.cpp
D:\Epic Games\Unreal Projects\ArenaBattle\Source\ArenaBattle\Game\ABGameMode.cpp(4): fatal error C1083: ?? ??? ?? ?????? 'Game/ABGameMode.h': No such file or dire
[5/6] Compile ABGameMode.gen.cpp
[6/6] Compile ArenaBattle.cpp
Build failed.
```

다음과 같이 `PublicIncludePaths.AddRange(new string[] { " " });` 를 추가하면, 더 이상 컴파일 에러를 보지 않아도 된다.



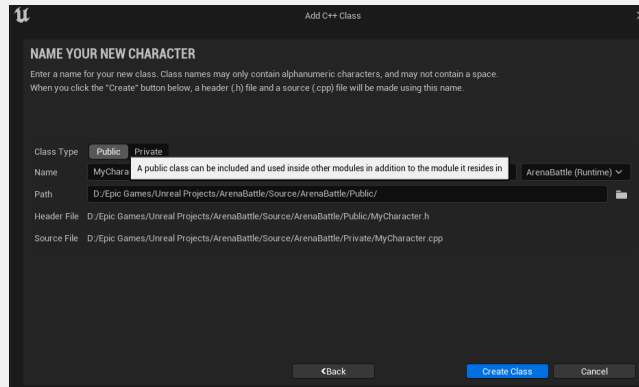
```
public class ArenaBattle : ModuleRules
{
    public ArenaBattle(ReadOnlyTargetRules Target) : base(Target)
    {
        PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;

        PublicIncludePaths.AddRange(new string[] { "ArenaBattle" });

        PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore" });
    }
}
```



이 방법 외에도, C++클래스 생성단계에서 Type을 Public 또는 Private으로 설정해준다면 위와 같은 문제를 해결할 수 있다. 정확한 원리는 몰랐지만, Public과 Private라는 모듈 하에 해당 클래스를 생성하는 방법으로 문제를 해결한 것 아닐까 추측한다.



[4] 게임모드의 구성

게임의 컨트롤러, 플레이어가 사용할 Pawn 등등 여러가지 규칙을 지정하는 클래스이며, 이를 설정하는 방법을 알아보자.

cpp의 생성자 단계에서 해당 내용들을 추가해 줄 수 있다.

주목할 점으로는 ControllerClass를 설정하는 과정에서 볼 수 있는데, 헤더 include를 통해 직접 해당 클래스에 대한 정보를 불러오지 않고 **ConstructorHelpers::FClassFinder**를 사용하여 직접 경로를 입력해줌으로써 정보를 불러오는 모습을 볼 수 있었다. 이는 헤더의 추가를 최소화하여 GameMode가 불필요하게 커지는 것을 예방한 것으로 보인다.

```
ABGameMode.cpp

#include "Game/ABGameMode.h"

AABGameMode::AABGameMode()
{
    //DefaultPawnClass =

    static ConstructorHelpers::FClassFinder<APlayerController> PlayerControllerClassRef(TEXT("/Script/ArenaBattle.ABPlayerController"));
    if (PlayerControllerClassRef.Class)
    {
        PlayerControllerClass = PlayerControllerClassRef.Class;
    }
}
```



Summary

1. 언리얼엔진에서는 게임 제작을 위한 프레임워크를 제공하고 있으며, 이를 활용하는 것을 권장한다.
2. 게임의 구성 요소를 폴더별로 분류하고, 헤더 참조를 최소화하는 규칙을 수립하는 것이 좋다. 만약 다른 폴더의 참조가 필요할 경우, **Interface**를 사용할길 추천한다.
3. `PublicIncludePaths.AddRange(new string[] { " " });` 를 통해, 해당 모듈에 관한 헤더 참조를 좀 더 간편하게 입력할 수 있다.

0n. 언리얼 엔진 게임 제작 기초 강의 과제



Q1. 현재 만들고자 하는 게임의 게임 프레임워크 요소를 정리해보시오.

현재 제작 중인 프로젝트는 포탑을 설치하여 몰려오는 적들을 막는다는 내용이다. 이를 구현하기 위해서 필요한 게임 프레임워크 요소를 생각해보자.

게임	- 건물을 설치할 수 있는 장소(월드) - 건물의 설치 여부와 게임의 승패 여부 등에 관한 판정을 내릴 수 있는 모드가 필요하다.
기믹	- 건물이 설치되는 또는 적들이 스폰되는 이벤트의 트리거를 작성해야 한다.
플레이어	- 플레이어의 입력을 다루는 컨트롤러와 이 모습을 탭뷰로 보여줄 수 있는 카메라 - 플레이어의 상태를 알려주는 HUD UI를 만들어야 한다.
폰	- 플레이어 캐릭터의 애니메이션과 위젯 - 에네미 캐릭터의 애니메이션, 건물의 애니메이션을 적용시켜줘야 한다.
데이터	- 설치하는 건물과 에네미 캐릭터의 정보를 담고있는 데이터 애셋을 만들어야 한다.
인공지능	- 적들을 감지하고 격파하는 포탑의 AI, 플레이어를 추적하고 공격하는 에네미의 AI가 필요하다.

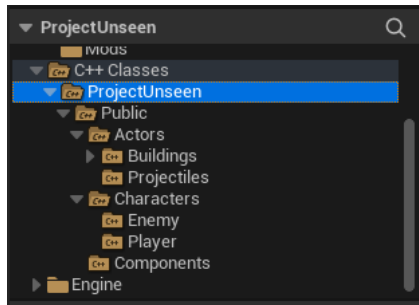
? Q2. Build.cs에 `PublicIncludePaths.AddRange(new string[] { "모듈이름" });` 설정을 추가하고, 소스코드를 최대한 폴더 단위로 묶어 보시오.

```

1 // Fill out your copyright notice in the Description page of Project Settings.
2
3 using UnrealBuildTool;
4
5 public class ProjectUnseen : ModuleRules
6 {
7     public ProjectUnseen(ReadOnlyTargetRules Target) : base(Target)
8     {
9         PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;
10
11         PublicIncludePaths.AddRange(new string[] { "ProjectUnseen" });
12
13         PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject", "Engine", "InputCore", "EnhancedInput", "UMG", "AIModule", "NavigationSystem",
14
15         PrivateDependencyModuleNames.AddRange(new string[] { });
16
17         // Uncomment if you are using Slate UI
18         // PrivateDependencyModuleNames.AddRange(new string[] { "Slate", "SlateCore" });
19
20         // Uncomment if you are using online features
21         // PrivateDependencyModuleNames.Add("OnlineSubsystem");
22
23         // To include OnlineSubsystemSteam, add it to the plugins section in your uproject file with the Enabled attribute set to true
24     }
25 }

```

본 강의를 듣기전에 프로젝트를 먼저 제작하기 시작했기에, 다른 방법으로 header경로 작성에 대한 문제를 해결했지만, 이처럼 간단히 모듈명을 넣어줘서 기능을 추가할 수 있었다.



다행히, 소스코드를 작성할 때에는, 강의에서 제시하는 방법과는 다르지만 내 나름의 카테고리를 나눠서 분류하고 있었다.

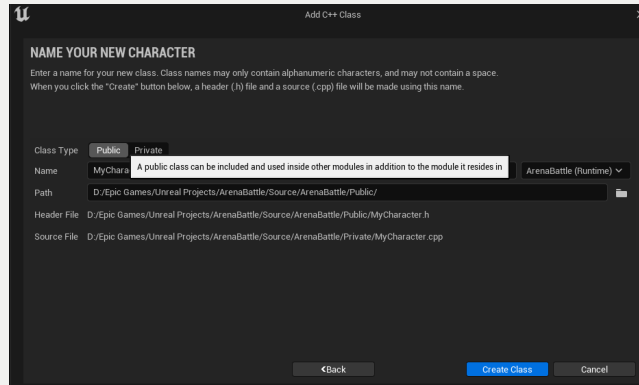


질문!

그동안 C++ 클래스 생성단계에서 ClassType을 Public으로 설정해줌으로써 `PublicIncludePaths.AddRange`를 사용한 것과 비슷한 효과를 얻을 수 있었습니다.

저는 이 방법이 언리얼엔진에서 기본적으로 Game폴더를 인식하듯, 기본적으로 인식하는 Public폴더 내에 해당 소스코드를 추가함으로써 상세한 경로명을 추가할 필요가 없는 것이라고 생각했는데 맞을까요?

그리고 ClassType을 사용하는 방법과 PublicIncludePaths를 사용하는 방법은 무슨 차이점이 있는지 궁금합니다.



Q3. cpp파일이 다른 폴더에 위치한 헤더 파일을 참조하고 있는지 확인해보고, 이를 없앨 수 있는 방법은 어떤 것이 있는지 생각해봅시오.

BaseTurret.cpp

```

#include "Actors/Buildings/Turrets/BaseTurret.h"
#include "Components/SphereComponent.h"
#include "Kismet/KismetMathLibrary.h"
#include "Kismet/GameplayStatics.h"
#include "Characters/Enemy/EnemyCharacter.h"
#include "Actors/Projectiles/BaseProjectile.h"
#include "Components/StatComponent.h"
#include "Components/BoxComponent.h"

ABaseTurret::ABaseTurret()
{
    ...
}

```

대분류로 3개의 카테고리를 나눠서 소스파일을 생성했는데, 놀랍게도 하나의 소스파일에서 3개의 카테고리에 대한 헤더를 모두 참조하고 있는 경우를 발견했다.

ClassFinder를 통해, 직접적으로 해당 소스코드에 대한 경로를 입력하는 방식으로 참조를 줄일 수 있을 것 같다.

강의 중에서는 Interface를 사용하는 방법도 제시하였으니, 이에 대한 내용을 좀 더 공부하고 사용하고 싶다.



Reference