

언리얼 프로그래밍 Part1-15

제목:언리얼 빌드시스템

****강의 내용 :** 언리얼 에디터의 동작 방식을 이해하고, 언리얼 엔진의 독특한 모듈 시스템을 기반으로 소스코드와 플러그인 구조를 직접 제작하기

****강의 목표**

강의 목표

- 언리얼 엔진의 프로젝트 구성과 에디터 동작 방식의 이해
- 언리얼 엔진의 모듈 시스템을 기반으로 소스 코드를 구성하는 방법의 학습
- 언리얼 플러그인 시스템을 활용한 효과적인 모듈 구성의 학습

Inflearn

1138208

이득우의 언리얼 프로그래밍 Part1 - 언리얼 C++의 이해
이득우

언리얼 엔진의 프로젝트 구성과

***언리얼 에디터 프로젝트 구성

언리얼 에디터 구성

- 게임 제작을 위해 에픽 게임즈가 제공하는 저작 도구
- 언리얼 엔진의 구성
 - 에디터 : 게임 제작을 위해 제공되는 응용 프로그램 (일반적으로 인식하는 언리얼 엔진)
 - 게임 빌드 : EXE 파일과 리소스로 이루어진 독립적으로 동작하는 게임 클라이언트
- 언리얼 에디터의 특징
 - 게임 개발 작업을 위해 다양한 폴더와 파일 이름 규칙이 미리 설정되어 있다.
 - 정해진 규칙을 잘 파악하고 프로젝트 폴더와 파일을 설정해야 함.
- 에디터에서 기획과 개발을 완료한 후, 게임 빌드를 통해 최종 게임 빌드를 제작하도록 설정



우리가 보통 게임 엔진이라고 하면

- 우리가 보통 게임 엔진이라고 하면 '언리얼 에디터'같은 것을 가르킨다.
- 언리얼 엔진은 크게 '에디터'와 '게임빌드' 두가지로 나뉘어진다.
- 언리얼 에디터의 특징

**언리얼 에디터의 동작

언리얼 에디터의 동작

- 프로젝트 폴더의 uproject 확장자를 더블클릭하면 에디터가 트리거 됨.
- 에디터의 실행 방식
 - uproject 확장자는 윈도우 레지스트리에 등록되어 있음.
 - 등록이 안되어 있다면 런처를 실행해 등록
 - UnrealVersionSelector 프로그램으로 프로젝트 정보가 넘겨짐
 - UnrealVersionSelector는 런처가 저장한 에디터 정보로부터 버전에 맞는 에디터를 실행함
- UnrealVersionSelector 소스는 에픽게임즈 GitHub에서 확인 가능
<https://github.com/EpicGames/UnrealEngine/tree/release/Engine/Source/Programs/UnrealVersionSelector>



프로젝트라고 이야기합니다



**에디터 버전 정보의 파악

에디터 버전 정보의 파악

- 프로젝트.uproject 텍스트 파일에 정해져있음.
- .uproject 확장자는 에디터를 띄우기 위한 명세서 역할을 함
- 버전 내용은 JSON 형식으로 구성되어 있음.
- 파일에 기록된 버전 정보를 바탕으로 에픽 런처가 지정한 정보를 찾아 에디터를 실행함.
 - ProgramData/Epic/UnrealLauncher 폴더에 관련 정보가 있음.
 - 이 역시 JSON 형식으로 설치된 언리얼 버전 정보가 기록되어 있음.



런처에 해당 정보가 들어가 있습니다

-UnrealBuildSystem.uproject를 만들어 실행하면 총5개의 폴더가 새로 생긴다

| | | | |
|-------------------|--------------------|-------------------------|-----|
| UnrealBuildSystem | 2024-09-03 오후 3:56 | Unreal Engine Projec... | 1KB |
| Intermediate | 2024-09-03 오후 3:57 | 파일 폴더 | |
| Saved | 2024-09-03 오후 3:57 | 파일 폴더 | |
| DerivedDataCache | 2024-09-03 오후 3:57 | 파일 폴더 | |
| Content | 2024-09-03 오후 3:57 | 파일 폴더 | |
| Config | 2024-09-03 오후 3:57 | 파일 폴더 | |

(1)config : 프로젝트 설정에 필요한 정보를 보관하는데 사용

(2)content : 에셋들을 보관하는데 사용

(3)DerivedDataCache : 우리가 사용한 에셋들의 중요 정보를 캐싱 하는데 사용

(4)Intermediate : 임시적으로 사용되는 중간 결과물들을 보관하는데 사용

(5)Saved : 임시로 무언가를 저장하는 용도로 활용 한다.

-여기서 Intermediate폴더는 용량을 보고 용량이 크면 언제든지 삭제해도 무방하다

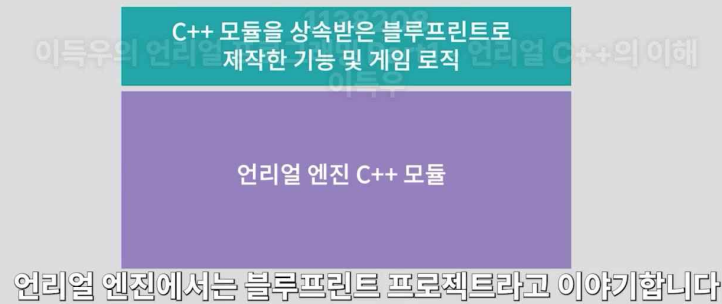
-Saved는 의도적으로 어떤 것을 저장하지 않으면 지워도 큰 문제가 없다.

-DerivedDataCache의 경우에도 데이터 캐시 값이 있다면 로딩을 빨리하는데 도움이 되지만, 당장 용량이 급하다 하면 동작에 문제가 없다.

**블루프린트 프로젝트

블루프린트 프로젝트

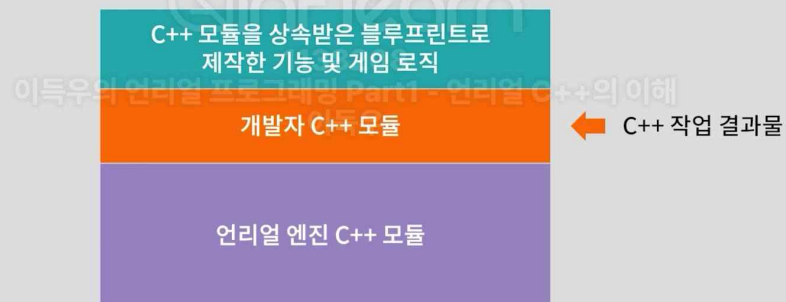
- C++ 코드가 없는 언리얼 프로젝트를 의미함.
- 언리얼 엔진이 제공하는 기본 기능을 활용해 게임을 제작하는 프로젝트
- 언리얼 엔진은 게임 제작에 필요한 기능을 모듈이라는 단위로 제공하고 있음.
- 언리얼 엔진의 모듈을 상속받아 블루프린트를 활용해 모든 기능과 로직을 구현하는 방식



**언리얼 C++ 프로젝트


언리얼 C++ 프로젝트

- 언리얼 엔진 C++ 모듈에 개발자가 추가로 자신만의 C++ 모듈을 추가할 수 있음.
- 언리얼 엔진 모듈과 개발자 모듈을 함께 사용하는 프로젝트



언리얼 C++ 모듈

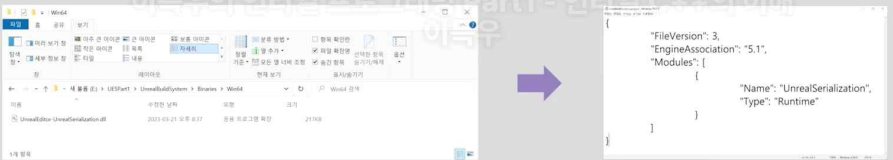
- 언리얼 엔진의 소스코드는 모두 모듈(Module) 단위로 구성되어 있음.
- 모듈을 컴파일함으로서 에디터 및 게임에 우리가 제작한 로직을 공급할 수 있음.
- 모듈 단위로 구성된 C++ 소스 코드를 컴파일한 결과물
 - 에디터 용으로 DLL 동적라이브러리
 - 게임 용으로는 정적 라이브러리
- 에디터 용 모듈은 언제나 UnrealEditor-[모듈이름].DLL 이름 규칙을 가지고 있음.
하나의 단위라고 생각하시면 될 것 같습니다



언리얼 엔진 설치 폴더에 가면 수많은 에디터 모듈이 설치된 것을 볼 수 있음

언리얼 C++ 모듈의 추가

- 기본 언리얼 모듈에 우리가 제작한 C++ 모듈을 추가해 에디터를 띄우고 싶은 경우.
- 우리가 만든 에디터 모듈(DLL 동적라이브러리)을 빌드 폴더에 넣어주어야 함.
 - Windows의 경우 Binaries/Win64 폴더에 해당 DLL을 넣어야 함
 - 빌드된 모듈 목록이 있는 UnrealEditor.modules 파일도 같은 폴더에 넣어주어야 인식됨.
- uproject 명세서에 모듈 이름을 지정하고 에디터를 실행.



```
{
  "FileVersion": 3,
  "EngineAssociation": "5.1",
  "Modules": [
    {
      "Name": "UnrealSerialization",
      "Type": "Runtime"
    }
  ]
}
```

그렇다면 우리도 당연히 저런 형태로 모듈을 만들면

- 실습을 위해 1-14의 Binaries 폴더를 복사하여 UnrealBuildSystem에 복사하자.
- uproject 파일의 내용을 사진과 같이 변경
- 실행 후 이거 로딩할 때 생성자 코드에 에셋이 없어서 경고 뜰거임

**모듈 C++코드의 관리

모듈 C++ 코드의 관리

- 언리얼 프로젝트가 소스 코드를 관리하는 규칙에 따라 소스 코드 구조를 구성해야 함.
- 소스 코드는 멀티 플랫폼 빌드 시스템을 지원하기 위해 특정 프로그램에 종속되어 있지 않음.
- 실제 빌드를 진행하는 주체 : Unreal Build Tool 이라는 C# 프로그램
- Source 폴더에 지정된 규칙대로 소스를 넣으면 플랫폼에 맞춰서 알아서 컴파일을 진행



그렇다면 이번에는 적절 모듈을 만들어서

-언리얼 엔진 소스코드의 특징은 멀티플랫폼 빌드 시스템을 지원하기에 특정 프로그램의 구조를 따르지 않으며, 언리얼만의 규칙을 가진다. 예를들어 우리가 작업할 소스코드는 비주얼 스튜디오와 100% 무관하게 구성된다.

-실제 빌드를 담당하는 주체가 있는데 중간에서 언리얼 엔진이 제공하는 언리얼 빌드 툴이라는 c# 프로그램이 있다.

-언리얼 엔진 빌드툴은 우리가 source라는 폴더에 지정된 규칙대로 소스를 넣으면 플랫폼에 맞춰 프로젝트를 만들고 빌드를 수행해준다.

**Source폴더의 구조

Source 폴더의 구조

- Source 폴더
 - 타겟 설정 파일
 - 모듈 폴더 (보통은 프로젝트 이름으로 모듈 이름을 지정)
 - 모듈 설정 파일
 - 소스 코드 파일 (.h 및 .cpp 파일들)
 - 타겟 설정 파일 : 전체 솔루션이 다룰 빌드 대상을 지정함.
 - {프로젝트이름}.Target.cs : 게임 빌드 설정
 - {프로젝트이름}Editor.Target.cs : 에디터 빌드 설정
 - 모듈 설정 파일 : 모듈을 빌드하기 위한 C++ 프로젝트 설정 정보
 - {모듈이름}.Build.cs : 모듈을 빌드하기 위한 환경 설정

C#이 가진 유연한 기능(compile on-the-fly)을 활용해
런타임에 cs 파일을 읽어 빌드 환경을 구축하고 컴파일을 진행함

-하나의 모듈을 만들기 위해 두가지 설정 파일들을 지정해줘야 하는데 이것들이 각각 '타겟 설정 파일'과 '모듈 설정 파일'이다.

-이런 설정파일들은 c# 코드들로 구성되어 있는데 이는 언리얼 빌드 툴이 c# 프로그램이기 때문이다. c#의 경우 실행중에 코드를 바로 컴파일하고 그 결과를 반영할 수 있는 기능이 있다.

게임 프로젝트의 소스

- 내가 만든 소스가 게임 프로젝트의 C++ 모듈이 되기 위해 필요한 것.
- 모듈(Module)을 구현한 선언한 헤더와 소스 파일이 있어야 함.
 - 주로 {모듈이름}.h와 {모듈이름}.cpp로 지정함
- 모듈의 뼈대를 제작
 - 매크로를 통해 기본 뼈대 구조를 제작
 - IMPLEMENT_MODULE : 일반 모듈
 - IMPLEMENT_GAME_MODULE : 게임 모듈
 - IMPLEMENT_PRIMARY_GAME_MODULE : 주 게임 모듈
- 일반적으로 게임 프로젝트는 주 게임 모듈을 하나 선언해야 함.

모든 준비가 완료되면 Generate Visual Studio project files. 메뉴를 선택

Intermediate 폴더에 프로젝트 관련 파일이 자동으로 생성됨.

Source 폴더를 규칙에 맞게 구성하면 Intermediate 폴더는 언제든지 재생성이 가능함
이를 위해서 모듈 이름으로 된 헤더와

-실습 도중 Target.cs파일은 프로젝트 단위로 관리가 되기 때문에 프로젝트에 단 하나만 존재한다.

-Source파일에 있는 UnrealBuildSystem는 모듈 이름이 된다.

-UnrealBuildSystem.Build.cs

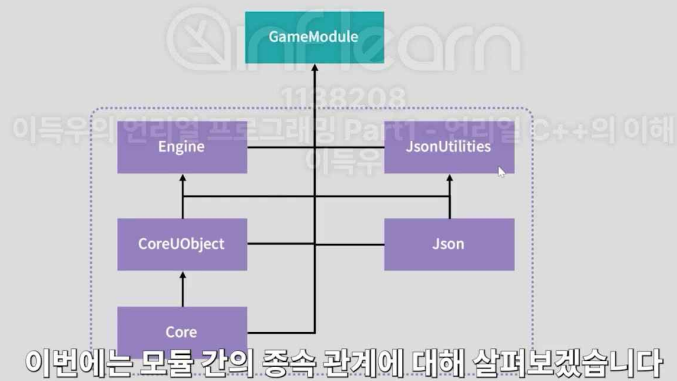
- (1)클래스 생성자에 우리 모듈이 사용하고 있는 여러 가지 설정 지정이 가능하다.
- (2) 기본적으로 우리가 사용할 코드는 게임과 관련이 있어 언리얼 엔진의 기본 기능을 사용해야 한다
- (3) 기본 기능은 Core, CoreUObject, Engine, InputCore라는 기본적인 모듈들이 있는데 게임 제작 과정에서 거의 필수로 사용한다.

-UnrealBuildSystem 헤더와 cpp 파일을 작성후 처음 빌드해도 UnrealBuildSystem에 대한 모듈은 찾아볼 수 없을 것이다. 이유는 우리가 만든 모듈에 언리얼 오브젝트가 없기 때문이다.(아무것도 없으면 표시하지 않도록 에디터가 설계됨.)

**모듈간의 종속 관계

모듈간의 종속 관계

- 모듈 사이에 종속 관계를 설정해 다양한 기능을 구현할 수 있다.
- 우리가 만드는 게임 모듈도 언리얼 엔진이 만든 모듈을 활용해야 한다.
- 언리얼 엔진이 제공하는 모듈 사이에도 종속 관계가 있음.



-언리얼 엔진의 소스는 결국 수많은 모듈의 집합이다. 우리가 만든 모듈도 결국에는 언리얼 엔진이 미리 만들어둔 모듈을 참고하여 다양한 언리얼 오브젝트를 생성해주어야 한다.

-우리가 참조하는 모듈도 서열이 있는데 가장 근본 모듈은 Core모듈이다.

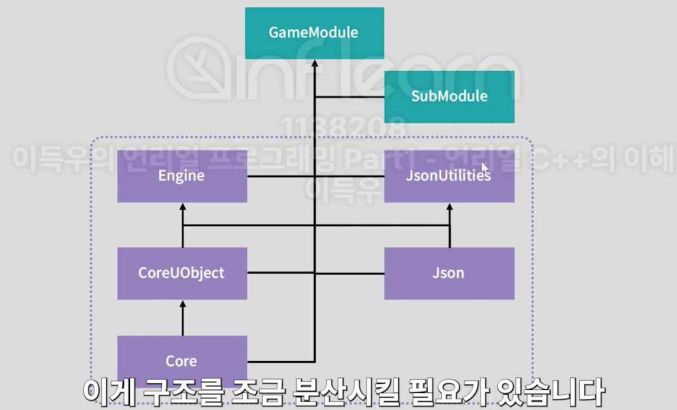
-Core다음은 CoreUObject모듈을 기반으로 언리얼 오브젝트가 빌드업 되고 있다.

-이러한 기능들이 모여 엔진이라고 하는 엔진이라고 하는 게임 콘텐츠 제작의 중심을 이루는 거대한 모듈이 엔진 내에서 만들어 지게 된다.

**새로운 모듈의 추가

새로운 모듈의 추가

- 하나의 모듈에 너무 많은 코드가 들어가면 언리얼 엔진은 빌드 방식을 변경함.
- 그렇기에 프로젝트가 커질 수록 모듈을 나누어서 관리하는 것이 유리.



-우리가 만드는 게임이 방대해지면 구조를 조금 분산시켜야 하는데 하나의 모듈에 너무 많은 코드가 들어가게 되면 해당 모듈의 컴파일 방식을 자동으로 변경하는데 제작 규모가 어느정도 커지면 게임 제작에 도움이 되는 기능은 별도의 모듈로 분리하는게 좋다.

-언리얼 엔진은 하나의 엔진에 여러개의 게임 모듈을 둘 수 있도록 설계 됨.

-다음 설명을 듣기전 GameModule(주게임 모듈), Submodule(서브모듈) 이라고 생각하자.

**모듈의 공개와참조

모듈의 공개와 참조

- 모듈 내 소스를 필요한 만큼만 공개해야 모듈 간 의존성을 줄이고 컴파일 시간을 최소화 할 수 있음.
- 공개할 파일은 모두 Public 폴더로
 - 예외) 예전 언리얼 엔진은 Classes 폴더가 있어 Public 폴더 역할을 하면서 언리얼 오브젝트를 관리했음
- 숨길 파일은 모두 Private 폴더로
- 외부로 공개할 클래스 선언에는 {모듈이름}_DLL 매크로를 붙일 것.
- 게임 모듈에서는 Build.cs 설정을 통해 참조할 모듈을 지정할 수 있음.

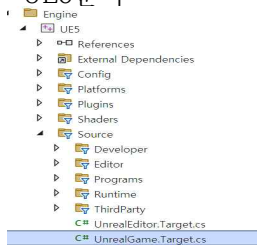


-주게임 모듈은 서버모듈이 제공하는 기능에 의존한다.

-서브모듈의 모든 기능을 참조가 아닌 필요한것만 참조하도록 설계하는게 좋다.(의존성 최소화) -->컴파일 시간이 줄어듬

-언리얼 엔진은 모듈내에 public과 private라는 두가지 폴더 규칙을 만듦

-UE5분석



(1)위에 보이는 UE5라고 되어있는 프로젝트가 언리얼엔진의 소스코드이다.

(2)Source폴더에 들어가면 구조가 동일한데 Target이라는 cs파일이 들어가있는 것을 볼 수 있다.

(3)Source에 여러 가지 폴더가 구분이 되어 있는데 이 폴더가 반드시 모듈을 의미하는 것은 아니다. 모듈의 경우엔 반드시 타겟 파일 아래에 위치해야할 필요는 없다. 하지만 우리가 모듈을 시작할때 모듈.Build.cs파일이 있으면 된다. Source안에 폴더들은 모듈들이 들어가 있는 것을 묶은 대표 폴더다.

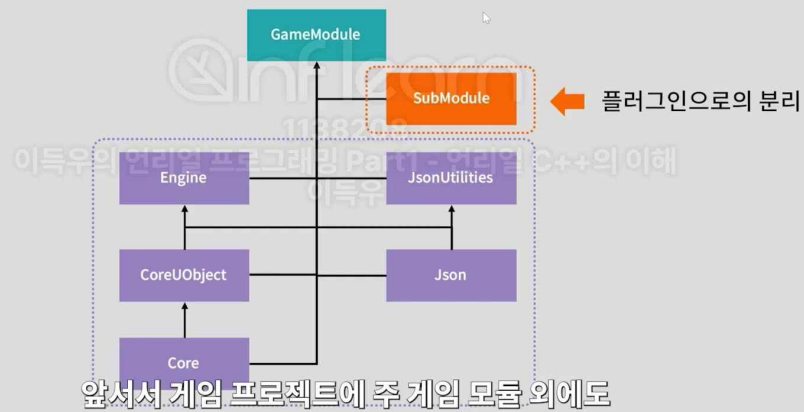
(4) 모듈들이 너무 많다 보니 성격에 따라 다시 폴더를 만들어준것이라고 생각해주면 된다.(RunTime 폴더 들어가면 굉장히 많은 모듈들이 폴더로 들어가 있고 이중에 앞서 언급한 Core 모듈이 있다.)

(5) 모듈은 항상 Build.cs파일이 있는곳이 모듈이 시작되는 곳이다. 또한 여기엔 Public폴더와 private 폴더가 있다. 또한 Public 폴더에는 또다시 외부로 노출되는 많은 헤더파일이 선언되어 있다. (Engine폴더속 Classes 폴더는 Public와 동일한 역할을 한다.)

**플러그인 시스템

플러그인 시스템

- 게임 프로젝트 소스에 모듈을 추가하는 방법은 분업이 어렵다는 단점이 있음.
- 모듈만 독립적으로 동작하는 플러그인 구조를 만들어 분업화하는 것이 바람직함



-게임프로젝트 소스에 모듈 추가하는 방법은 가능하지만, 코드를 함께 프로젝트에서 관리해야 해서 분업이 까다로움

-그래서 공용 기능들을 모아둔 모듈들은 프로젝트에서 분리시켜 스스로 동작 하게 설정 하는 걸 고려할 수 있는데 이것을 위해 '플러그인 시스템' 이란게 존재함

**플러그인 구조

플러그인 구조

- 플러그인은 다수의 모듈과 게임 콘텐츠를 포함하는 포장 단위
- 에디터 설정을 통해 유연하게 플러그인을 추가하거나 삭제할 수 있음
- 플러그인 구조
 - 플러그인 명세서 (uplugin 파일)
 - 플러그인 리소스 (Resource 폴더, 에디터 메뉴용 아이콘)
 - 콘텐츠
 - 모듈 폴더
- 이러한 플러그인은 마켓 플레이스 판매로도 이어질 수 있도록 여러 설정을 추가할 수 있음.

플러그인은 다수의 모듈과

-에디터 설정은 프로젝트에서 함.

-실습

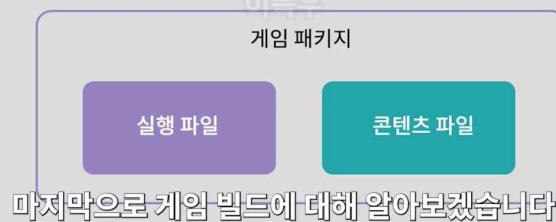
- (1)플러그인 이름은 우선 '게임유틸리티'로 지정
- (2)Plugins폴더를 만들고 그안에 GameUtility폴더를 만들고 그 안에 플러그인 명세서인 Uplugin을 하나 채워주자
- (3)강의에 나오는것처럼 플러그인 만들 때 수동으로 쳐야할 필요는 없고 에디터 메뉴를 통해 추가 가능함 다만, 구조파악을 위해 강의에서는 수동으로 작업함.
- (4) 모듈이 들어갈 Source파일을 만들고 CommonUtility모듈을 추가함(파일 만들면 됨.) 그리고 항상 모듈 안에는 Build.cs파일이 있어야함
- (5) CommonUtility.h와 CommonUtility.cpp파일을 만들고 재구성한다
- (6) CommonUtility.cpp의 모듈속성은 공용 기능이기 때문에 IMPLEMENT_MODULE만 사용한다.
- (7) 에디터내에서 Person과 중요한 기능이 담겨있다고 가정한 PersonImpl을 CommonUtility 모듈에다가 만든다.
- (8) 다시 CommonUtility 탐색기로 가서 public과 private로 구분해보자
- (9) 분류하고 나면 이 CommonUtility를 참고하는 다른 모듈들은 오로지 이 Person 유 오브젝트에 대해서만 사용이 가능하다.(변경후 솔루션 재생성 해야한다.)
- (10)PersonImpl 클래스는 내부에서만 사용하기에 외부로 노출할 필요가 없다. 헤더파일에서 모듈이름_API로 되어있는 매크로들을 삭제하여 안전하게 외부모듈이 참조할 수 없도록 지정하는게 좋다.
- (11) 주게임 모듈 같은 경우 주게임 모듈을 참조하는 다른 모듈이 있는건 설계에 좋지 않다. 따라서 주게임 모듈 안에서 굳이 퍼블릭인지, 프라이빗인지 나눌필요가 없다.
- (12) CommonUtility라는 모듈을 참조하면 Public폴더의 위치를 자동으로 참조하여 여기서 만들어진 라이브러리들을 자동으로 링크해 사용이 가능하다.
- (13) 플러그인시스템은 Edit-Plugins폴더에서 자유롭게 추가하거나 삭제가 가능하다.

**게임빌드

게임 빌드

- 게임 타겟 설정을 추가하면 게임 빌드 옵션이 추가됨.
- 게임 타겟으로 빌드된 모듈은 정적 라이브러리로 실행 파일에 포함됨.
- 게임이 실행되기 위해서는 실행 파일과 콘텐츠 애셋이 함께 있어야 함.
- 빌드 : 실행 파일을 생성하기 위한 컴파일
- 쿠킹 : 지정한 플랫폼에 맞춰 콘텐츠 애셋을 변환하는 작업
- 패키징 : 이들을 모두 모아서 하나의 프로그램으로 만드는 작업

이득우의 언리얼 프로그래밍 Part1 - 언리얼 C++의 이해
이득우



-실행파일 빌드와 쿠킹된 에셋을 묶어 하나의 프로그램을 만드는 것을 패키징이라고 한다.

-실습

(1)Source파일로 가서 새로운 타겟 파일을 만든다

(2)타겟 파일에서 Type정보에 .Game을 꼭 지정해야한다. (기본 세팅)

(3) Game타입을 지정후 재구성하면 빌드된 타겟이 더 늘어나는데 Shipping 빌드가 사용자한테 배포할 최종 게임의 코드를 만들어내는 작업이다.(모든 코드들이 최적화되어 들어간다.) 따라서 앞서 사용한 check()나 ensure()와 같은 어설션 매크로들은 Shipping빌드에서 제외된다.

(4)빌드하고 exe파일 실행하면 missing에러 다이얼로그가 뜨는데 실행파일은 만들었지만, 게임에 필요한 에셋들이 없어 프로그램 실행할 수가 없다는 뜻이다.

(5) 게임 에셋들을 대상 플랫폼으로 튜닝해서 묶어둔 에셋모음이 필요한데 이것이 쿠킹이다.

(6)에디터에서 상단 Platforms 메뉴에서 windows->shipping을 체크하고 패키지 프로젝트 메뉴를 선택후 Package폴더를 만들고 그안에 선택하면 쿠킹과 빌드를 동시에 수행해준다.

(7)E:\HighProject\UnrealBuildSystem\Package\Windows\UnrealBuildSystem\Binaries\Win64 여기에 전에 실행했던 100MB짜리 실행파일 있다.

(8)사용되는 에셋은 package폴더를 통해 탐색하다 나오는 content폴더에 있다.

(9)엔진에 관련된 기본 에셋은 Engine 폴더에 있다.

(10) 실행은 Package->Windows에 있는 exe파일로하자.

****정리**

언리얼 빌드 시스템

1. uproject 명세서를 사용한 언리얼 에디터 동작 원리
2. 언리얼 엔진의 모듈 시스템과 소스 코드 관리 방법
3. 모듈 작업 분리를 위한 플러그인 시스템
4. 언리얼 소스코드의 구조
5. 게임 빌드의 설정과 게임 패키징 과정

inflearn
1138208
이득우의 언리얼 프로그래밍 Part1 - 언리얼 C++의 이해
이득우

에디터의 동작 원리에 대해서 살펴보았습니다

***part1 : 언리얼 c++의 이해

Part1 : 언리얼 C++의 이해

1. 헬로 언리얼!
 2. 언리얼 C++ 코딩 규칙
 3. 언리얼 C++ 기본 타입과 문자열
 4. 언리얼 오브젝트 소개
 5. 언리얼 오브젝트 리플렉션 시스템 I
 6. 언리얼 오브젝트 리플렉션 시스템 II
 7. 언리얼 C++ 설계 I - 인터페이스
 8. 언리얼 C++ 설계 II - 컴포지션
 9. 언리얼 C++ 설계 III - 델리게이트
 10. 언리얼 컨테이너 라이브러리 I - Array와 Set
 11. 언리얼 컨테이너 라이브러리 II - 구조체와 Map
 12. 언리얼 엔진의 메모리 관리
 13. 언리얼 오브젝트 관리 I - 직렬화
 14. 언리얼 오브젝트 관리 II - 패키지
 15. 언리얼 빌드 시스템
- 기본적으로 알아야 되는 필수적 개념에 대해서 설명드렸습니다

-Part1은 언리얼 엔진으로 게임을 제작하기전 기본적으로 알아야 되는 필수적 개념에 대해 설명함.

