언리얼 프로그래밍 Part1-2

제목: 언리얼 C++ 코딩 표준

** 강의 목표
- 프로그래밍을 시작하기 전에 알아두어야 할 언리얼 c++ 코딩 표준을 이해
- 언리얼 c++표준 코딩 표준에서 주의할 점의 확인

**코딩 표준(Coding Standard)
-프로그래밍을 작성하는데 지켜야 하는 프로그래밍 이름 규칙, 작성 방법 등을 지정한 가이드라인
-코딩 스타일(Coding Style), 코딩 컨벤션(Coding Convention)이라고도 함.
-프로그래밍 하는데 있어서 표준은 매우 중요함

**좋은 코딩 표준
-절대적으로 좋은 코딩 표준이란 없음.
->이전에 내가 사용한 코딩 표준이 항상 옳은 것이 아님.

-중요한 것은 코딩 표준을 정하고 잘 따르는 것
->이미 프로젝트에 코딩 표준이 있다면 그대로 따라야 함.

-프로젝트의 모든 코드는 한 사람이 만든 것처럼 보여져야 함.

-좋은 소프트웨어 회사들은 자신만의 코딩 표준이 있음.

-어떤 프로젝트의 모든 코드가 마치 한사람이 만든것처럼 보여줘야 한다. 즉 좋은 소프트웨어 회사들은 자신만의 코딩 표준이 존재한다.

-언리얼 엔진은 자체 표준이 존재하기 때문에 기존의 c++에서 유용한 코딩 표준이 있을지라도 문서에 명시된 언리얼 엔진 코딩 표준을 따라야 하며 권고사양이 아닌 반드시 지켜야 하는 규칙이다.

-언리얼 엔진의 소스코드는 코딩 표준을 따라 작성이 되어 있는데 우리의 게임 로직이 다른 표준으로 작성되어 있다면 읽는데에 불편할 수 있다. 즉 기존의 c++ 좋은 표준이 있을지라도 언리얼 엔진의 공식 코딩 표준을 꼭 따라야 한다.

-통일된 방식으로 코딩을 작성해야 코드를 분석하는데 있어서 소비 시간을 줄일 수 있고 이는 프로젝트의 효율과 직결된다.

**unreal c++ 코딩표준
- 링크:언리얼 엔진을 위한 에픽 C++ 코딩 표준 | 언리얼 엔진 5.4 문서 | Epic Developer Community (epicgames.com)

- 내용

1.언리얼 엔진에서 코딩 규칙이 프로그래머에게 중요한 이유
(1)소프트웨어의 총 수명 비용 중 80%는 유지보수에 소모됩니다.
(2)최초 작성자가 그 소프트웨어의 수명이 다할 때까지 유지보수하는 경우는 거의 없습니다.
(3)코딩 규칙을 사용하면 소프트웨어의 가독성을 향상하여 엔지니어가 새로운 코드를 더 빠르고 완벽히 이해할 수 있습니다.
(4)에픽에서 소스 코드를 모드 개발자 커뮤니티에 공개할 경우 코딩 규칙을 알고 있으면 이해하기 더 쉽습니다.
(5)대다수의 코딩 규칙이 크로스 컴파일러 호환성에 필요합니다.

2. 클래스 체계
클래스는 작성자보다는 읽는 사람을 염두에 두고 조직되어야 합니다. 읽는 사람은 대부분 클래스의 퍼블릭 인터페이스를 사용할 것이므로, 퍼블릭 인터페이스에서 먼저 선언한 후 클래스의 프라이빗 구현이 뒤따라야 합니다.

ex)
```
class MYPROJECT_API UMyGameInstance : public UGameInstance
{
	GENERATED_BODY()

public:

private:

};
```
-->탬플릿이 주어지면 우선 퍼블릭과 private를 선언해놓고 시작하라는 소리임

3. 저작권 고지
배포용으로 에픽에서 제공한 모든 소스 파일(.h, .cpp, .xaml, etc.)은 파일 첫 번째 줄에 저작권 고지를 포함해야 합니다. 저작권 고지의 포맷은 다음과 정확히 일치해야 합니다.
(에픽 게임즈가 작성한 소스코드에는 저작권 고지가 있기 때문에 임의 대로 수정하지 말고 고지해야한다.)

4. 명명 규칙 (일반 적인 c++ 코딩과 다르게 언리얼 엔진은 특별한 명명 규칙이 있어 중요하다.)

-언리얼 엔진은 파스칼 케이싱을 따라야 한다.
(1) 파스칼 케이싱(Pascal Casing) : 합성어의 첫 글자를 대문자를 사용해 명명 (언리얼 엔진)
ex)UnrealEngine

(2) 카멜(낙타) 케이싱(Camel Casing) : 첫 합성어는 소문자로 나머지는 대문자를 사용해 명명
ex)unrealEngine

(3) 스네이크 케이싱(Snake Casing) : 합성어 사이에 언더바(_)를 사용해 명명 (일반적인 c++ 언어)
ex)unreal_engine

-타입 이름에는 추가적으로 대문자로 이루어진 접두사를 포함하여 변수 이름과 구분합니다. 예를 들어 FSkin 은 타입 이름이고, Skin 은 FSkin 의 인스턴스입니다.

(1)템플릿 클래스에는 접두사 T를 포함합니다.

(2)UObject 에서 상속받는 클래스에는 접두사 U를 포함합니다.

(3)AActor 에서 상속받는 클래스에는 접두사 A를 포함합니다.

(4)SWidget 에서 상속받는 클래스에는 접두사 S를 포함합니다. (특정 UI에서 사용 됨)

(5)추상적 인터페이스인 클래스에는 접두사 I를 포함합니다.

(6)에픽의 개념이 유사한 클래스 타입(TModels 타입 특성에 첫 번째 아규먼트로 사용)에는 접두사 C를 포함합니다.

(7)열거형에는 접두사 E를 포함합니다.

(8)부울 변수는 접두사 b를 포함합니다(예: bPendingDestruction 또는 bHasFadedIn).

(9)그 외 대부분의 클래스는 접두사 F를 포함합니다. 그러나 일부 서브시스템은 다른 글자를 사용하기도 합니다. [즉 언리얼 오브젝트로부터 상속 받지 않는 클래스와 구조체들은 F를 사용한다]

(10) 탬플릿과 관련된 내용은 우리가 크게 사용할일이 없지만 숙지하도록 하자.

-부울을 반환하는 모든 함수는 IsVisible() 또는 ShouldClearBuffer() 등의 true/false 질문을 해야 합니다. (부울 반환 함수는 질의형으로 만들자, 명사형으로 만들면 모호해질 수 있다.)

-필수 사항은 아니지만, 함수 파라미터가 참조로 전달된 후 함수가 그 값에 쓸 것으로 예상되는 경우 이름 앞에 접두사 'Out'을 추가할 것을 권장합니다. 이렇게 하면 이 아규먼트에 전달되는 값이 함수로 대체된다는 것을 확실히 알 수 있습니다. [인자가 밖으로 나와서 함수가 끝난 이후에도 사용될 것인지 입력으로만 사용될것인지 둘다 사용될 것인지 써줘야 한다.]

-In 또는 Out 파라미터도 부울인 경우 bOutResult 와 같이 In/Out 접두사 앞에 'b'를 붙입니다.

-인종, 민족, 종교 포용성
(1)편견을 강화하는 은유나 직유를 사용하지 마세요.

(2)여기에는 blacklist / whitelist 와 같이 흑백이 대비되는 표현도 포함됩니다.

(3)역사상의 트라우마나 실제 차별받았던 경험을 연상시키는 단어를 사용하지 마세요.

(4)여기에는 slave, master, nuke 등이 포함됩니다.

(5)우리는 글로벌로 깃허브등을 통해 코딩을 공유하기 때문에 문제의 소지가 있는 단어들은 사용하지 말아야 한다.

-포터블 C++ 코드
(1) 우리는 int를 쓸때에도 int32와 같이 정확한 용량을 지정해주는 코드를 써야한다.
(많이 쓰는 자료형: uint8, int 32)

(2) 문자열에 대해서도 char이 아닌 TCHAR를 사용하여 문자를 표현해주어야 한다.

(3) c++ 규약에서 int는 크기를 보장하지 않는다 하지만 규약에서는 최소로 32비트로 보장해줘야 한다. (어쨋든 불확실 하기 때문에 32비트인지 64비트인지 보장해줘야 한다.) [최고의 퍼포먼스를 내기 위해서는 자료형이 가지고 있는 크기를 파악하는 것이 매우 중요하다.]

(4) 자세한건 다음 강의에서 다룸

- 표준라이브러리
(1) 언리얼 엔진이 개발될 때는 c++ 표준 라이브러리가 정착되지 않아 자체적으로 멀티 플랫폼에서 안정적으로 동작되는 표준 라이브러리를 자체적으로 구축하였다.

(2)현재 표준 라이브러리는 더욱 안정적이고 완성도가 높아졌다. 하지만 표준 라이브러리는 범용적으로 설계되었기 때문에 게임에 끌어들이면 컴파일 시간이나 복잡도들이 증가하게 돼서 팀마다 결정해야할 문제다.

(3)언리얼 엔진에서 구현이 도움이 되지 않는다면 모든 사용을 표준 라이브러리로 이주하도록 결정할수도 있다. (아직 결정된 사항도 안되고, 이미 언리얼 엔진 코딩 생태계에 너무 익숙해져있고, 너무 잘 돌아가기 때문에 굳이 표준 라이브러리를 이주해서 다시 새로운 문제점을 만들어낼 이유가 없다. 하지만 염두는 해두자.)

(4)즉 c++ 표준 라이브러리는 사용하지 않는다가 기본 원칙이다.

(5) 헤더가 키워드가 겹치는 것이 있어 혼합하지 않도록 주의해야 한다.(memcpy(), memset() 같은거 언리얼엔진이 직접 만들어 제공하고있음) [홈페이지 참고하자]

-코멘트
(1) 코멘트 작성은 한번씩 읽어보자

-Const 정확도
(1) consts선언되는 변수는 불변하다라고 지시하는건데 가급적이면 const를 사용가능하면 다 사용하자. (컴파일 타임에서 사전에 체크하기 때문에 코드의 정확성이 높아진다.)

(2) 루프를 돌 때 값이 변경되면 전체 루프가 깨지기 때문에 웬만해서는 const로 지정해주는게 좋다.

(3) 포인터 자체를 const, 즉 포인터의 증감 연산자를 사용하면 안됨을 지시할려면 포인터 자체에 const를 둘수도 있다.

(4) 레퍼런스의 경우엔 레퍼런스 자체를 선언하는 것만으로 const가 지정되어 있고 레퍼런스에 대해서 const를 지정하는 것 자체가 문법적으로 의미가 없고 괴장한 이상한 문법이 된다. (레퍼런스에는 const를 지정해주면 안된다.)

(5) const가 포인터에 지정되어 있다면 포인터 연산자에 대해서 증감 연산같은 것을 사용할 수 없다. 하지만 이것이 가리키는 연산자는 const가 아니기 때문에 이 포인터가 가리키는 데이터는 우리가 변경할 수 있다.

(6) 배열 반환이 레퍼런스로 지정되지 않으면 복사가 일어나기 때문에 나쁘다.

(7) 함수를 통해서 반환된 값을 레퍼런스로 const로 받은 경우 복사가 일어나지 않고 사용할 수 있고. Array안에 있는 데이터를 고치지 않기 때문에 좋은 예이다.

(8) 포인터에 Const가 선언되지 않으면 증감연산자 사용이 되며, 이 포인터가 가르키는 데이터는 바뀌면 안된다.

(9) 예시 잘 살펴보자.

-예시 포맷

(1) 코딩을 작성할 때 규칙에 맞춰 작성 하면 javaDoc을 사용하여 자동으로 문서를 만들 수 있다.

(2) 나중에 주석을 취합하여 문서화 시키기 좋은 프로그램이 있어 도움을 얻을 수 있다.

-최신 C++ 언어 문법
(1) C++17까지 사용이 가능하다.

-override , final [c++11 규약] --> 사용을 강력히 권함

-nullptr
(1) NULL값의 경우 c의 NULL이 아니라 nullptr로 사용해야 한다.

-'auto'키워드
(1) 굉장히 편리하지만 어떤 게임 엔진은 절대 금지하는 엔진도 있다.
(2) 언리얼 엔진도 왠만하면 사용하지 말라고 되어 있다.
(3) 사용이 가능한 상황들
    -> 변수에 람다를 바인딩해야 하는 경우. 람다 타입은 코드로 표현할 수 없기 때문입니다.
    ->이터레이터 변수의 경우. 단, 이터레이터 타입이 매우 장황하여 가독성에 악영향을 미치는 경우에 한합니다. (너무나 뻔한 구문에는 사용해도 괜찮다.)
    -> 템플릿 코드에서 표현식의 타입을 쉽게 식별할 수 없는 경우. 고급 사용 사례입니다.

-범위 기반 for(c++ 11규약)
(1) for문을 세게의 영역으로 쓰는 것을 두 개의 영역으로 줄여서 사용이 가능하다.

(2) 인덱스 값들을 사용 못하고 처음부터 끝까지 순회하는 방식에서만 사용 가능하다.

-람다 함수.
(1) 람다 함수도 우리가 어떻게 사용하는지 명확히 인지한 상태에서 명시적으로 사용해야한다.

-열거형 Enum
(1) 11규약 이후부터는 enum class를 사용하여 열거형 사용이 강화되었다.
(2) 여전히 11규약 이전의 열거형 이 존재할 수 있다.
(3) 홈페이지 참고하면 11규약 이전, 이후 차이점을 볼 수 있다.

-자료구조
(1) c++에서 자료구조 사용할 때 메모리 복사없이 바로 옮길 수 있는 문법도 지원한다.(스탠다드 라이브러리의 move사용)
(2) 언리얼 엔진에서는 MoveTemp를 유사한 함수를 사용하여 move기능을 사용할 수 있다.

-디폴트 멤버 이니셜라이저
(1) 언리얼엔진의 U오브젝트에서 생성자 구문에서 초기화 하는 것이 좋다.
(2) 어떤 객체가 가지고 있는 기본값에 대해서는 언리얼 오브젝트의 규칙을 이해하고 지정해주는 것이 좋다.

-서드파티 코드
(1) 서드파티 코드의 경우에는 '서드파티 코드'라고 명시해줘야 한다.

-중괄호
(1) 에픽게임즈에서는 새 줄에 중괄호를 사용하는 것이 오래된 관행이다.

-탭 및 들여쓰기
(1) 탭이 당현히 들여쓰기이다. (탭 크기는 4글자로 설정)

-Switch 문
(1) 가급적이면 Switch문에 대해서 아무것도 모르는 사람들이 사용할 수 있다는 전제하에 정확히 명시해주자.(홈페이지에 명시 방법 나와 있음.)

-네임스페이스
(1)게임 코드의 경우 네임스페이스를 지정하고 사용할일이 별로 없지만, 우리가 어떤 외부적인 툴을 만들 때 신경써주자.

-파일 이름
(1) 파일 이름은 클래스 이름과는 다르게 접두사를 빼는게 좋다.

(2) 모든 헤더는 #pragma once를 지시어로 작성하여 중복된 헤더를 복사하지 않도록 해야한다.

(3) 헤더에 무언가를 선언할 때에는 헤더의 모든 내용을 컴파일 단계에서 복사하기 때문에 헤더가 많을수록 컴파일 시간이 오래걸리게 된다. 가급적 include는 세밀하게 설계하여 지정해줘야 한다. (전방 선언:forward declaration을 사용할 수 있도록 하자.) 헤더는 최대한 적게!

-캡슐화
(1) 왠만하면 private로 설정하고 getter, setter 함수로 접근할수 있게 선언하자.
(2) final이란 명확하게 더 이상 파생시킬 클래스가 아니면 여기서 잠가줘야 할 때 써주자.

-일반적인 스타일 문제
(1) 의존성이나 종속성, 즉 어떤 클래스가 있을 때 다른 클래스에 영향을 주지 않게끔 설계하는 것이 기본적인 원칙이다.
(2) 스트링 리터럴 주변에는 항상 TEXT()매크로를 사용할 수 있도록 하자.
(3) 코드가 길어지더라도 보기 편하도록 해야한다.
(4) 포인터를 선언할 때 FShaderType* PTR과 같이 타입뒤에 바로 포인터 레퍼런스 붙이고 한칸 띄고 변수이름
(5) 애매모호한 변수명은 길어도 좋으니 항상 명확하게 작성하자.
(6) 헤더에 적용된 static 변수를 선언하면 헤더를 참조하는 모든 인스턴스들이 컴파일 되기 때문에 빼줘야 한다. (실제 메모리에 할당 내용들은 cpp 파일에 이정도가 있다.! 정도만 해야하며, 그것이 무엇인지는 cpp에서 구현해야 한다.)

-API디자인 가이드
(1) Bool형태를 인자로 집어넣는 것을 피하고 카테고리 지정시 enum과 같은 열거형을 지정해준다.

(2) 열거형의 숫자조합이 늘어나면 비트플래그를 사용하여 조합하는 형태로도 지정 가능하다.

(3) 너무 함수 파라미터를 길게 사용하면 안되고 이런 경우 다 합쳐 구조체를 선언하여 전달하자.

(4) 인터페이스에 대한 문법은 없기 Eoansd 멤버변수가 있을수도 있다. 하지만 멤버 변수가 있으면 인터페이스 변수에 맞게 멤버 변수를 쓰지 말자

(5) 오버라이딩 할때는 override 키워드를 반드시 명시하자.

-플랫폼별 코드
(1) 나중에 어떤 플랫폼에 특정된 코드를 구현 할 때 일반 기능에 영향을 미치지 않도록 설계를 해야한다. (게임 작업시엔 큰 영향이 없지만 게임을 만들고 특정 플랫폼에 옮길 때 알아야 한다.)

*정리

언리얼 엔진 코딩 표준

1. public 에서 private로 이어지는 클래스 체계(Organization)를 준수.
2. 명명 규칙
   1. 파스칼 케이싱(Pascal Casing)을 사용한다.
   2. 소문자를 가급적 사용하지 않고 공백 및 언더스코어(_)가 없음
   3. 모든 클래스와 구조체에는 고유한 접두사가 있음
3. 코드의 명확성
   1. 파라미터에 가급적 In과 Out 접두사를 사용해 명시
   2. const 지시자(directive)의 적극적인 활용
   3. 레퍼런스를 통한 복사 방지
   4. auto 키워드는 가급적 자제
4. Find In Files의 활용
5. 헤더 파일 및 #include 구문은 의존성을 최소화시켜 주의 깊게 다룰 것

기존의 C++ 코딩을 사용해 때