# C++

## 1. 컴포넌트

##### CreateDefaultSubobject<T>()

UObject 클래스를 상속받는 모든 클래스에서 사용가능한 함수이다. 하위 오브젝트나 컴포넌트를 생성할 때 사용되는 함수. 이 함수는 T의 포인터(T\*) 타입을 반환한다.

### UTextRenderComponent

설정된 텍스트를 3D 공간 상에 렌더링하는 컴포넌트이다. 글자 색, 크기, 폰트, 정렬 등을 설정할 수 있으며 액터 등에 컴포넌트로 덧붙여서 사용할 수 있다.

##### TextRenderComponent -> SetHorizontalAlignment(EHTA\_Center);

렌더링되는 텍스트의 수평 정렬을 설정하는 함수이다. 정렬 방식은 EHTA\_Center, EHTA\_Left, EHTA\_Right가 있다.

##### TextRenderComponent->SetWorldSize(100.0f);

렌더링되는 텍스트의 월드에서의 크기를 설정하는 함수이다.

##### TextRenderComponent->SetText(TEXT("TEXT"));

렌더링되는 텍스트의 문자열 내용을 설정하는 함수이다.

### USceneComponent

트랜스폼을 가지고 있고 다른 컴포넌트를 이 컴포넌트에 덧붙이는(Attachment) 것을 지원하지만 충돌 같은 물리적 효과를 지원하지 않고 렌더링 기능이 없다.

##### SubComponent->SetupAttachment(RootComponent);

SetupAttachment() 함수는 컴포넌트를 다른 컴포넌트의 아래 계층으로 붙이는데 사용된다. 위의 예시 코드에 따르면 SubComponent는 계층적으로 자식 컴포넌트가 되고 RootComponent는 부모 컴포넌트가 되는 것이다.

##### **SubComponent->SetRelativeLocation(FVector(-250.0f, 0.0f, 250.0f));**

현재 컴포넌트가 부모를 기준으로 얼마나 떨어진 위치에 있을지 정한다.

##### **SubComponent->SetRelativeRotation(FRotator(-45.0f, 0.0f, 0.0f));**

컴포넌트가 부모를 기준으로 얼마나 회전된 상태인지 정한다.

##### **SubComponent** ->SetRelativeLocationAndRotation(FVector(.0f, .0f, 50.f), FRotator(-60.f, .0f, .0f));

컴포넌트가 부모를 기준으로 위치, 회전을 더하여 정한다.

##### SubComponent->GetComponentScale();

GetComponentScale() 함수는 월드 스페이스에서의 컴포넌트 크기를 가져온다.

##### SubComponent->SetWorldScale3D(FVector(0.0f, 0.0f, 0.0f));

SetWorldScale3D() 함수는 월드 스페이스에서의 컴포넌트 크기를 수정한다.

### UStaticMeshComponent

액터에 덧붙일 수 있는 스태틱 메시 컴포넌트이다. 월드에 렌더링된다.

##### StaticMeshComponent->SetStaticMesh(SphereVisualAsset.Object);

스태틱 메시 컴포넌트의 스태틱 메시를 설정하는 함수

### UParticleSystemComponent

액터에 파티클 시스템을 덧붙일 수 있는 컴포넌트

##### ParticleSystemComponent->bAutoActivate = true;

파티클 시스템이 생성되자마자 자동으로 켜질지에 대한 변수

##### ParticleSystemComponent->SetTemplate(ParticleAsset.Object);

파티클 시스템 컴포넌트의 파티클을 설정하는 함수

##### ParticleSystemComponent->ToggleActive();

파티클을 켜고 끄는 함수

### USphereComponent

액터에 구형 충돌 물리 효과를 줄 수 있는 컴포넌트

##### SphereComponent->InitSphereRadius(40.0f);

스피어 컴포넌트의 반지름은 설정하는 함수

##### SphereComponent->SetCollisionProfileName(TEXT("Pawn"));

콜리전의 프로필을 설정하는 함수. [프로젝트 세팅>엔진>콜리전] 하단에 Preset을 열어보면 각 콜리전 프로필마다 어떤 물리 설정을 가지고 있는지 확인할 수 있다.

### UPawnMovementComponent

Pawn의 움직임을 업데이트하는데 사용되는 컴포넌트

##### PawnOwner;

이 컴포넌트를 소유하고 있는 폰

##### UMovementComponent::UpdatedComponent;

UPawnMovementComponent의 부모 클래스인 UMovementComponent 클래스에 속하는 변수로 이 무브먼트 컴포넌트가 이동시키고 업데이트 해야할 컴포넌트

##### UMovementComponent::ShouldSkipUpdate(DeltaTime);

이동된 컴포넌트가 이동할 수 없거나 렌더링되지 않은 경우인지를 판별하여 알려주는 함수

##### ConsumeInputVector();

대기중인 입력을 반환하고 다시 0으로 설정하는 함수

##### SafeMoveUpdatedComponent(DesiredMovementThisFrame, UpdatedComponent->GetComponentRotation(), true, Hit);

언리얼 엔진 피직스를 이용해서 입체 장애물을 피해서 폰 무브먼트 컴포넌트를 이동시키는 함수

##### SlideAlongSurface(DesiredMovementThisFrame, 1.0f - Hit.Time, Hit.Normal, Hit);

컴포넌트가 이동하다가 충돌이 발생했을 때, 제자리에 멈추는 대신 충돌체의 표면을 타고 미끄러지듯이 이동하도록 도와주는 함수

##### AddInputVector(Vector);

## 2. 입력

## 3. 트랜스폼

##### SetWorldRotation(FRotator(Pitch, Yaw, Roll + 10.0f));

##### SetRelativeLocation(FVector(0.0f, 0.0f, 150.0f));

##### SetWorldSize(150.0f);

## 4. 충돌

##### Virtual void NotifyHit(UPrimitiveComponent\* MyComp, AActor\* Other, UPrimitieComponent\* OtherComp, bool bSelfMoevd, FVector HitLocation, Fector HitNormal, FVVector NormalImpulse, const FHitResult& Hit) override;

충돌 알림

##### virtual void NotifyActorBeginOverlap(AActor\* OtherActor) override;

두개의 액터 모두 bGenerateOverlapEvents = true; 이어야 한다.

## 5. 카메라

### UCameraComponent

액터에 덧붙일 수 있는 카메라 컴포넌트이다.

##### CameraComponent->FieldOfView = 60.0f;

원근감 모드(Projection Mode)에서의 수평 시야각을 Field of view라고 한다. 수평 시야각이 넓어지면 물체가 확대되어서 보이기 때문에 주로 FPS게임에서 저격 소총의 줌 효과에 주로 사용된다.

### USpringArmComponent

##### USpringArmComponent\* SpringArmComponent;

부모 오브젝트와 자식 오브젝트 사이에 일정한 거리를 유지하게 도와주는 컴포넌트. 충돌이 있는 경우라면 유연하게 부모와 자식 사이의 거리를 좁혔다가 충돌이 사라지면 다시 원래대로 돌아가게하는 기능을 제공한다.

##### SpringArmComponent->TargetArmLength = 400.0f;

아무런 충돌이 없을 때, 스프링 암의 자연적인 거리를 정할 수 있는 변수

##### SpringArmComponent->bEnableCameraLag = true;

true인 경우, 카메라가 목표 위치보다 뒤떨어져서 따라가도록 한다.

##### SpringArmComponent->CameraLagSpeed = 3.0f;

bEnableCameraLag가 true인 경우, 카메라가 목표 위치에 도달하는 속도를 제어한다.

### APlayerCameraManager

##### APlayerCameraManager\* camManager = GetWorld()->GetFirstPlayerController()->PlayerCameraManager;

현재 플레이어 컨트롤러가 사용하고있는 카메라 정보가 필요할 때 사용하는 클래스이다.

## 6. 클래스

### Actor

##### AActor::InputComponent

입력이 활성화된 액터에 대한 입력을 처리하는 컴포넌트이다.

##### InputComponent->BindAction("Action", IE\_Pressed, this, &AMyActor::ActionProcess);

액션 매핑에 처리 함수를 바인딩하는 함수다.

첫 번째 매개변수는 바인딩할 액션 매핑의 이름이다.

두 번째 매개변수는 처리할 키 이벤트다. 기본적으로 사용되는 이벤트는 키가 눌렸을 때를 뜻하는 IE\_Pressed와 눌린 키가 떼졌을 때를 뜻하는 IE\_Released가 있다.

세 번째 매개변수는 입력을 바인딩하는 오브젝트이다.

네 번째 매개변수는 입력이 들어왔을 때 입력을 처리하는 함수이다.

##### InputComponent->BindAxis("Axis", this, &AMyPawn::AxisProcess);

축 매핑에 처리 함수를 바인딩하는 함수다.

첫 번째 매개변수는 바인딩할 축 매핑의 이름이다.

두 번째 매개변수는 입력을 바인딩하는 오브젝트이다.

세 번째 매개변수는 입력이 들어왔을 때 입력을 처리하는 함수이다.

##### AActor::GetActorLocation()

액터의 월드 스페이스 상의 위치를 가져오는 함수이다.

##### AActor::SetActorLocation()

액터의 월드 스페이스 상의 위치를 정하는 함수이다.

##### GetActorRotation();

액터의 현재 회전을 반환하는 함수

##### SetActorRotation(FRotator());

액터의 회전을 설정하는 함수

##### GetActorForwardVector(); , GetActorRightVector();

액터의 Forward 방향을 구하는 함수, 액터의 Right 방향을 구하는 함수

##### GetWorld();

UWorld 객체를 가져오는 함수. UWorld는 액터나 컴포넌트들을 포함하는 맵이나 샌드박스의 최상위 객체이다.

### Pawn

Pawn 클래스는 플레이어나 AI가 컨트롤할 수 있는 모든 액터의 베이스 클래스다.

##### APawn::AutoPossessPlayer

레벨이 시작되거나 폰이 생성되었을 때, 플레이어 컨트롤러가 있다면 어떤 플레이어 컨트롤러가 자동으로 이 폰을 소유해야 되는지에 대한 변수다.

## 7. 수학, 구조체, 변수 자료형

### FMath

##### Clamp(Value, Min, Max)

Value값을 Min과 Max 사이로 보정하여 리턴한다.

##### Lerp(ValueA, ValueB, Factor);

선형 보간 함수이다. ValueA와 ValueB 사이의 Factor(0.0~1.0)값의 위치에 해당 하는 값을 구해준다.

### FVector

##### FVector Vector;

언리얼 엔진에서 3D 상의 위치나, 속도를 나타내는데 쓰이는 구조체

##### Vector.GetClampedToMaxSize(Value);

길이가 Value인 이 벡터의 복사본을 만들어서 반환하는 함수

##### Vector.IsNearlyZero();

지정된 허용오차 내에서 벡터의 길이가 0에 근접하는지 확인하는 함수

### FVector2D

### FColor

### 숫자 유형

**short**, **int**, **long** 와 같은 기본적인 유형도 플랫폼마다 크기가 각기 다를 수 있기에, UE4 에서는 대안으로 사용해야 하는 유형을 다음과 같이 제공하고 있습니다:

**int8**/**uint8**: 8 비트 부호 있는/없는 정수형

**int16**/**uint16**: 16 비트 부호 있는/없는 정수형

**int32**/**uint32**: 32 비트 부호 있는/없는 정수형

**int64**/**uint64**: 64 비트 부호 있는/없는 정수형

부동 소수점 역시 표준 **float** (32 비트) 및 **double** (64 비트) 유형으로 지원됩니다.

### 문자열

텍스트, 모니터, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

##### FString

FString MyStr = TEXT("Hello, Unreal 4!").

FString 가변(mutable) 문자열로, std::string 과 유사합니다. FString 에는 스트링 작업을 수월하게 하기 위한 대규모의 메서드 모음이 들어있습니다. 새로운 FString 생성을 위해서는, TEXT() 매크로를 사용합니다.

##### FText

FText MyText = NSLOCTEXT("Game UI", "Health Warning Message", "Low Health!")

FText 는 FString 과 유사하나, 현지화 텍스트 용입니다. 새로운 FText 생성을 위해서는, NSLOCTEXT 매크로를 사용합니다. 이 매크로는 네임스페이스, 키, 기본 언어 값을 받습니다:

##### FName

FName 은 자주 반복되는 문자열을 식별자로 저장하여 그에 대한 비교를 할 때 메모리와 CPU 시간을 절약하는 데 쓰입니다. 이를 참조하는 모든 오브젝트에 전체 문자열을 매번 반복 저장하기 보다, FName 은 보다 작은 저장 용량을 차지하는 Index 를 사용하여 주어진 문자열에 매핑시킵니다. 그러면 문자열 내용을 한 번만 저장하여, 여러 오브젝트에 사용할 때의 메모리를 절약합니다. NameA.Index 가 NameB.Index 와 같은지 검사해 보면 그 문자열 안의 글자를 하나하나 검사할 필요 없이 두 문자열을 빠르게 비교할 수 있습니다.

##### TCHAR

TCHAR 는 플랫폼마다 다를 수 있는 캐릭터 세트와 무관하게 캐릭터를 저장하기 위한 방편으로 사용됩니다. 내부적으로, UE4 문자열은 TCHAR 배열을 사용하여 UTF-16 인코딩으로 데이터를 저장합니다. TCHAR 를 반환하는 레퍼런스 해제 오버로드 연산자를 사용하여 원(raw) 데이터에 접근할 수 있습니다.

### 컨테이너

Container (컨테이너)는 그 주요 함수가 데이터 모음을 저장하는 데 쓰이는 클래스입니다. 이 클래스 중 가장 일반적인 것은 TArray, TMap, TSet 입니다. 이들 각각은 크기가 동적으로 변하며, 원하는 만큼 확장됩니다.

##### TArray

세 가지 컨테이너 중 언리얼 엔진 4 에서 가장 많이 쓰이는 컨테이너는 TArray (배열)인데, std::vector 와 매우 유사한 방식으로 작동하나, 훨씬 많은 함수 기능이 제공됩니다. 일반적인 것 몇 가지는 다음과 같습니다:

Num(), Add(NewActor), AddUnique(NewActor)**,** Remove(NewActor)**,** RemoveAt(Index), RemoveAtSwap(Index), Empty()

##### TMap

TMap (맵)은 키-값의 짝으로 된 모음으로, std::map 과 비슷합니다. TMap 에는 키를 가지고 요소를 검색, 추가, 제거하기 위한 빠른 메서드가 있습니다. 키에는 GetTypeHash 함수가 정의되어 있기만 하다면, 어떤 유형이든 사용할 수 있습니다.

Contains(), Add(), Remove(), Empty(), GetIndex(), Reset()

반복자

CreateIterator(), RemoveCurrent()

##### TSet

TSet (세트)는 고유한 값 모음을 저장하며, std::set 와 비슷합니다. TArray (배열)이 AddUnique 및 Contains 로 세트같은 동작을 지원하기는 하지만, TSet 에는 이 작업에 대한 더욱 빠른 구현이 있고, 복수 추가도 방지됩니다.

Add(), Contains(), Revmove(), Empty(), Array(),

반복자

CreateIterator(), RemoveCurrent(), GetIndex(), Reset()

### UENUM

UENUM(BlueprintType)

Eum class EMyColor : uint8

{

Red UMETA(DisplayName = “Red”)

Green UMETA(DisplayName = “Green”)

Blue UMETA(DisplayName = “Blue”)

};

## 8. 기타

### FTimerHandle

#include "TimerManager.h" 포함시키기..

타이머를 구별할 수 있는 유일한 핸들이다. 타이머를 생성하는 함수는 타이머를 생성할 때, 타이머의 핸들을 돌려주는데, 이 핸들을 가지고 있어야 생성한 타이머를 중지시킬 수 있다.

##### GetWorldTimerManager().SetTimer(TimerHandle, this, &ACountdown::AdvenceTimer, 1.0f, true);

첫 번째 매개변수는 지금 생성되는 타이머의 핸들이다. 위에서 설명했듯이 이 핸들을 가지고 있어야 나중에 타이머를 종료할 수 있다.

두 번째 매개변수는 타이머 함수를 호출하는 오브젝트이다.

세 번째 매개변수는 타이머가 발동할 때마다 호출될 함수이다.

네 번째 매개변수는 타이머가 호출될 시간이다. 만약 값을 1로 두면 1초에 한 번씩 함수가 호출된다.

다섯 번째 매개변수는 타이머의 반복 여부이다. 만약 값이 false라면 타이머는 반복되지 않고 정해진 시간에 한 번만 호출된다.

##### GetWorldTimerManager().ClearTimer(TimerHandle);

돌아가고 있는 타이머를 중지시키고 해당 핸들을 무효화시키는 함수이다.

### ConstructorHelpers

생성자에서만 사용해야하는 클래스

##### static ConstructorHelpers::FObjectFinder<T> Asset(TEXT("AssetPath"));

프로젝트에서 필요한 콘텐츠나 리소스, 에셋을 불러오는데 쓰이는 클래스

##### static ConstructorHelpers::FClassFinder<T> BpClass(TEXT("AssetPath"));

프로젝트에서 블루프린트 클래스 불러오는데 쓰이는 클래스

##### Asset.Succeeded();

에셋을 불러오는데 성공했는지를 반환하는 함수

##### Asset.Object;

불러온 에셋을 담고 있는 변수

### FHitResult

##### FHitResult Hit;

충돌에 대한 정보를 담고 있는 구조체

### 클래스 이름 접두사

언리얼 엔진에서는 빌드 프로세스 도중 코드를 생성하는 툴을 제공합니다. 이러한 툴에는 클래스 이름에 기대하는 작명 규칙이 몇 가지 있는데, 이름이 그 기대와 일치하지 않을 경우 경고나 오류가 나게 됩니다. 툴에서 기대하는 클래스 접두사에 대한 상세 설명은 아래 목록과 같습니다.

Actor (액터) 파생 클래스 접두사는 A 입니다. 예: AController.

Object (오브젝트) 파생 클래스 접두사는 U 입니다. 예: UComponent.

Enum (열거형) 접두사는 E 입니다. 예: EFortificationType.

Interface (인터페이스) 클래스 접두사는 보통 I 입니다. 예: IAbilitySystemInterface.

Template (템플릿) 클래스 접두사는 T 입니다. 예: TArray.

SWidget (슬레이트 UI) 파생 클래스 접두사는 S 입니다. 예: SButton.

그 외의 접두사는 letter F 입니다. 예: FVector.

### 형변환

AClass\* class = Cast<AClass>(OtherActor)

If(class != nullptr)

### 로그

##### UE\_LOG(LogTemp, Log, TEXT(“내용”))

**GEngine->AddOnScreenDebugMessage(-1, 2.0f, FColor::Green, FString::Printf(TEXT("deltaTime : %f"), DeltaTime));**

첫번째 파라미터 -1 == 기존 로그 아래 줄에 생성, -1 != 기존 로그 지우고 생성

두번째 파라미터 로그가 화면에 남아있는 시간

### TSubClassOf<T>

UClass 타입 안정성을 보장하는 템플릿 클래스. TSubClassOf에 전달된 인수가 템플릿 인자로 받은 타입과 일치하거나 템플릿 인자로 받은 타입을 상속받은 타입인지를 런타임 중에 확인하도록 도와주는 클래스이다.

### UUserWidget

##### UUserWidget\* UserWidget;

Widget Blueprint를 통해서 확장할 수 있는 사용자 위젯.

##### UserWidget->AddToViewport();

유저 위젯을 뷰 포트에 추가하는 함수.

##### UserWidget->RemoveToViewport();

유저 위젯을 뷰 포트에서 제거하는 함수.

##### CreateWidget(GetWorld(), newWidget);

위젯을 생성하는 함수

### APlayerController::SetInputMode()

##### SetInputMode(FInputModeGameAndUI());

플레이어 컨트롤러의 입력 모드를 설정하는 함수. Game 입력만 받을지, UI 입력만 받을지, 아니면 둘 다 받을지를 정할 수 있다.

### UPROPERTY()

##### UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = "UMG\_Game")

클래스의 멤버 변수에 붙이는 매크로. 에디터의 노출 수준을 정할 수 있다.

EditAnywhere는 에디터 디테일 패널에 노출이 가능하다.

BlueprintReadOnly는 블루프린트에서 수정을 불가능하고 읽기만 가능하다.

Category는 블루프린트에서 불러올 때, 카테고리를 분류해서 찾기 쉽게 만들어준다.

### UFUNCTION()

##### UFUNCTION(BlueprintCallable, Category = "UMG\_Game")

함수에 붙이는 매크로.

BlueprintCallable은 블루프린트에서 호출만 가능하다는 의미이다.

# 질문

PlayerInputComponent InputComponent 차이

EInputEvent 종류

플레이어 컨트롤러란?

AutoPossessPlayer = EAutoReceiveInput::Player0; 이게 뭔지?

한번은 코드를 튜토리얼에서 똑같이 작성을 했는데도 동작이 안되어서 로그를 띄워서 확인을 하려 했는데 로그를 새로 작성한 클래스에서 띄웠는데도 tick이나 생성자 등에서 아무데에서도 로그가 뜨지 않았는데, 기존에 있던 클래스 안의 함수에서 작성했는데 기존에 새로 작성한 클래스에서의 로그도 함께 뜨고, 동작이 안되었던 기능들도 잘 되었던 적이 있었습니다. 어쩔 수 없는건가요?

비주얼 어시스턴트 사용?

어떤 표기법 사용?

### Map 관련

##### UGameplayStatics::OpenLevel(GetWorld(), FName(\*TransferLevelName));

맵을 전환할 때 사용하는 함수

### Widget 관련

##### 텍스트의 투명도 변경

UTextBlock\* PressRKeyTextBlock = 목표 위젯->WidgetTree->FindWidget<UTextBlock>("PressRKey");

if (PressRKeyTextBlock != nullptr)

{

PressRKeyTextBlock->SetOpacity(0.2f);

}