


## 2단계\_02. 캐릭터와 입력 시스템

📅 수강일	@ 2023/06/28
➦ 이름	 전영재
🔍 멘토	Min-Kang Song, 현웅 최
👥 멘티	
✨ 작성 상태	Done
🕒 단계	2단계
☑ 강의 시청 여부	<input checked="" type="checkbox"/>
☑ 이수 여부	<input type="checkbox"/>

### Contents



- 캐릭터와 입력 시스템
  - [1] 액터와 컴포넌트
    - [2] C++액터에서 컴포넌트 생성
      - 컴포넌트의 생성
      - 컴포넌트의 등록
    - [3] 캐릭터의 제작
      - 폰의 기능과 설계
    - [4] 입력시스템의 개요
      - 입력시스템의 동작 방식
      - Enhanced Input System
  - 0n. 캐릭터와 입력 시스템 강의 과제

## 캐릭터와 입력 시스템



### 강의 목표

- 액터와 컴포넌트 개념의 이해
- 블루프린트로 확장 가능한 프로퍼티 설계
- 언리얼 엔진의 폰과 캐릭터 시스템의 이해
- 언리얼엔진5의 EnhancedInput 시스템 활용

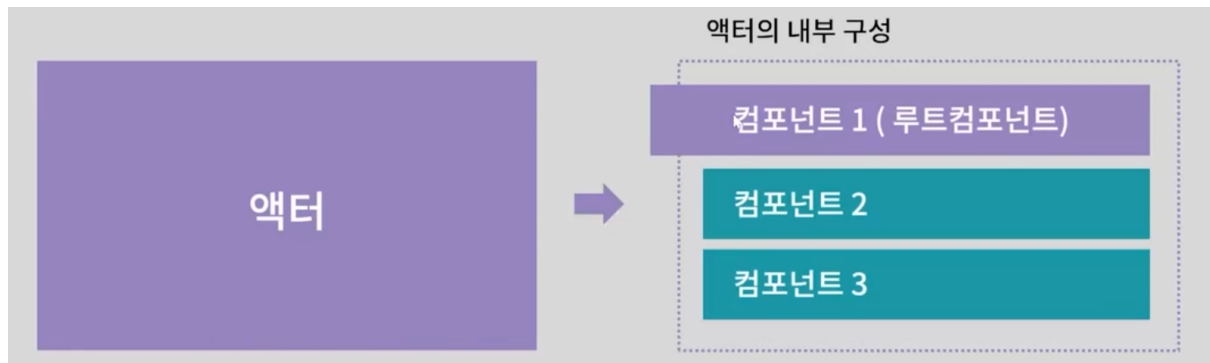
## [1] 액터와 컴포넌트



액터 = 월드에 속한 콘텐츠의 기본 단위

사실 액터는 논리적 개념일 뿐 컴포넌트를 감싸는 포장 박스에 불과하다.

즉, 중요한건 액터의 실질적인 기능을 구성하는 **컴포넌트**라고 할 수 있다.



## [2] C++액터에서 컴포넌트 생성

### 컴포넌트의 생성

컴포넌트는 언리얼 오브젝트이기 때문에 `UPROPERTY` 매크로를 설정한다.

이렇게 `UPROPERTY`로 설정하여 생성한 컴포넌트는 에디터 및 블루프린트로 확장하여 편집할 수 있어 편의성을 제공하며 `UPROPERTY` 지정자를 통해 필요에 따른 설정이 가능하다.

헤더에서 언리얼 오브젝트를 선언할 때는 일반 포인터가 아닌 `TObjectPtr`로 포인터를 선언한다.



#### 지정자의 예시

- **Visible / Edit** : 객체(Visible)타입과 값(Edit)타입으로 사용
- **Anywhere / DefaultsOnly / InstanceOnly** : 에디터에서 편집 가능 영역  
(위 Visible/Edit 과 Anywhere/DefaultsOnly/InstanceOnly를 합성하여 사용한다)
- **BlueprintReadOnly / BlueprintReadWrite** : 블루프린트 확장 시, 읽기 혹은 쓰기 권한 부여
- **Category** : 에디터 Detail 내의 카테고리 지정

### 컴포넌트의 등록

CDO(생성자)에서 생성한 컴포넌트는 **자동으로** 월드에 등록된다. 기본적으로 이 방법을 자주 사용하며 `UPROPERTY`와 `TObjectPtr`을 사용하는 것을 잊지말자. (UE4 기준에서 class를 통한 전방선언에 익숙해져서 손에 잘 익진 않더라..)

만약 런타임에서 `newObject`를 통해 컴포넌트를 생성한다면 **반드시 등록절차**를 거쳐야 한다. (예. `RegisterComponent`)

```
AABFountain.h
UCLASS()
class ARENABATTLE_API AABFountain : public Actor
{
    ...

public:
    UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Mesh)
    TObjectPtr<class UStaticMeshComponent> Body;
    UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = Mesh)
    TObjectPtr<class UStaticMeshComponent> Water;
}
```

위처럼 헤더파일에 컴포넌트의 포인터를 선언했다면, cpp파일 내부에서 `CreateDefaultSubobject`를 사용해 직접 생성해주면 된다.

```
AABFountain.cpp
AABFountain::AABFountain()
{
    PrimaryActorTick.bCanEverTick = true;

    Body = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Body"));
    Water = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Water"));

    RootComponent = Body; //이처럼 RootComponent로 지정해줄수 있다.
    Water->SetupAttachment(Body); //SetupAttachment를 통해 특정 컴포넌트 하에 부착시킬 수 있다.
```

```
static ConstructorHelpers::FObjectFinder<UStaticMesh> BodyMeshRef(TEXT("경로명")); //앞서 컴포넌트의 포인터를 생성해 준것이기에, 실질적인 Mesh를
if(BodyMeshRef.Object)
{
    Body->SetStaticMesh(BodyMeshRef.Object);
}
}
```

## [3] 캐릭터의 제작

### 폰의 기능과 설계

폰은 액터를 상속받은 액터이며, **플레이어가 빙의해 입출력**을 처리하는데 특화되어 있다.

크게 Collision컴포넌트, Mesh컴포넌트, Movement컴포넌트라는 세 가지 주요 컴포넌트로 구성되며

이러한 컴포넌트 중, **Transform정보를 소유하는지**에 따라 Actor컴포넌트, Scene컴포넌트라고 구분짓는다.

캐릭터는 폰 중에서도 **인간형 폰**을 구성하는데 특화된 전문 폰 클래스이다.

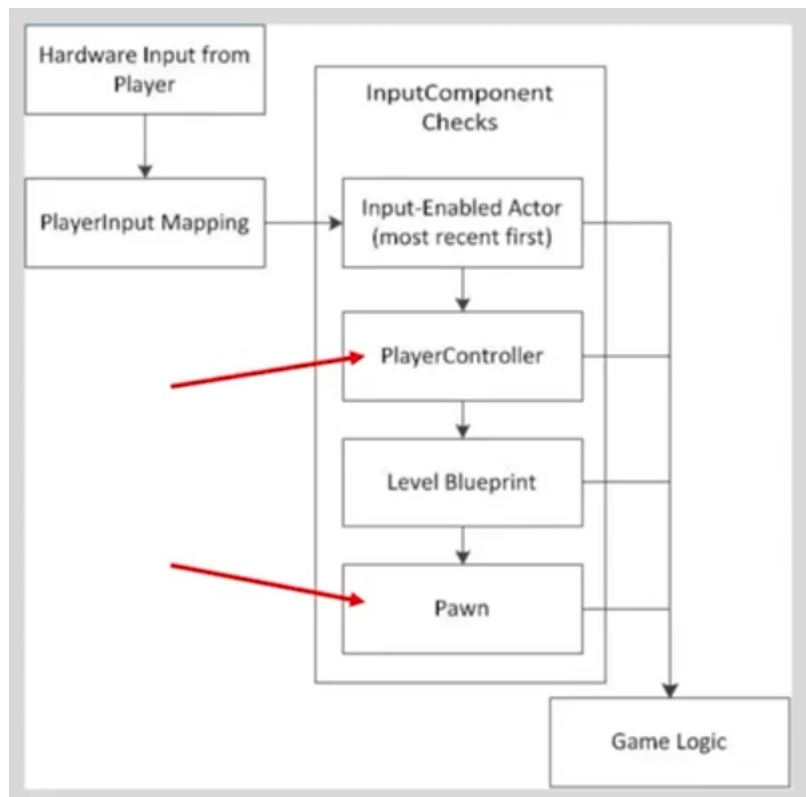
## [4] 입력시스템의 개요

### 입력시스템의 동작 방식

플레이어의 입력은 하드웨어(컨트롤러)를 통해 InputMapping으로 입력되고, PlayerController액터에게 전달된다. 또한 PlayerController액터는 Level블루프린트에게 이러한 입력을 전달하고, Level 블루프린트는 월드 내의 Pawn에게 전달하는 식으로 동작한다.

즉, 1차적으로 PlayerController에서 전달을 담당하고 2차적으로 Pawn에서 담당한다 생각하자.

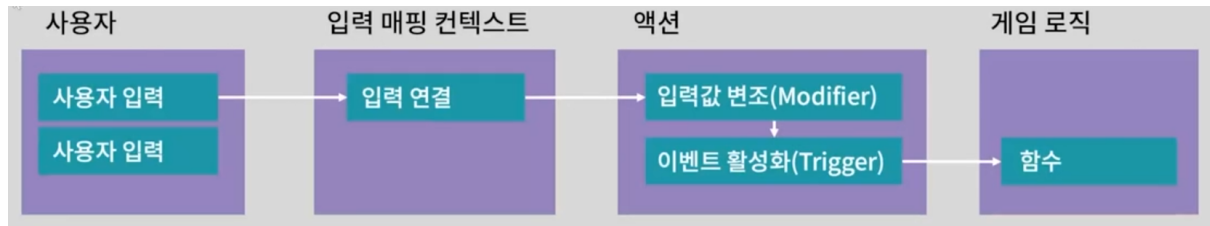
이런 경우, GTA같은 다양한 사물에 빙의하는 게임의 경우, 해당 Pawn에서 자신의 입력로직을 만드는 것이 코드가 분산되어 효율적이다.



### Enhanced Input System

그러나 기존 입력시스템의 경우, 플레이어의 입력을 설정하는 과정에서 Controller또는 Pawn이 처리하기 전에 Input Mapping을 처리하게 되므로, 고정된 Mapping을 사용해야만 했고, Mapping 변경에 관해 유연하게 대처할 수 없었다.

이러한 문제를 해결하기 위해, 플레이어의 입력을 최종단계인 Game Logic부에서 Mapping하도록 진행하는 Enhacend Input System이 도입되었다. (이것도 UE5에서 새로 나왔다.)



```
AABCharacterPlayer.h
UCLASS
class ARENABATTLE_API AABCharacter : public AABCharacterBase
{
public:
    AABCharacterPlayer();
    ...

protected:
    virtual void BeginPlay() override;

public:
    virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;

protected:
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, Meta = (AllowPrivateAccess = "true"))
    TSharedPtr<class UInputMappingContext> DefaultMappingContext;
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, Meta = (AllowPrivateAccess = "true"))
    TSharedPtr<class UInputAction> JumpAction;
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, Meta = (AllowPrivateAccess = "true"))
    TSharedPtr<class UInputAction> MoveAction;
    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input, Meta = (AllowPrivateAccess = "true"))
    TSharedPtr<class UInputAction> LookAction;

    void Move(const FInputActionValue& Value);
    void Look(const FInputActionValue& Value);
}
```

위와 같이, 헤더에서 입력연결을 다루는 InputMappingContext와 각각의 InputAction을 생성하고, 입력의 기능을 구현하는 함수를 생성하여 Mapping해주면 된다.

생성자 단계에서 입력의 할당을, BeginPlay에서는 MappingContext를 PlayerController의 SubSystem에 연결, SetupPlayerInputComponent에서 각 입력과 기능함수를 Mapping시켜줄 것이다.

```
AABCharacterPlayer.cpp
AABCharacterPlayer::AABCharacterPlayer()
{
    ...
    // Input
    // 코드에서 할당해줘도 되지만, 당연히 BP에서 할당해도 된다.
    static ConstructorHelpers::FObjectFinder<UInputMappingContext> InputMappingContextRef(TEXT("/Script/EnhancedInput.InputMappingContext"));
    if (nullptr != InputMappingContextRef.Object)
    {
        DefaultMappingContext = InputMappingContextRef.Object;
    }

    static ConstructorHelpers::FObjectFinder<UInputAction> InputActionMoveRef(TEXT("/Script/EnhancedInput.InputAction'/Game/ArenaBattle/"));
    if (nullptr != InputActionMoveRef.Object)
    {
        MoveAction = InputActionMoveRef.Object;
    }

    static ConstructorHelpers::FObjectFinder<UInputAction> InputActionJumpRef(TEXT("/Script/EnhancedInput.InputAction'/Game/ArenaBattle/"));
    if (nullptr != InputActionJumpRef.Object)
    {
        JumpAction = InputActionJumpRef.Object;
    }

    static ConstructorHelpers::FObjectFinder<UInputAction> InputActionLookRef(TEXT("/Script/EnhancedInput.InputAction'/Game/ArenaBattle/"));
    if (nullptr != InputActionLookRef.Object)
    {
        LookAction = InputActionLookRef.Object;
    }
}
```

```

}

void AABCharacterPlayer::BeginPlay()
{
    Super::BeginPlay();

    APlayerController* PlayerController = CastChecked<APlayerController>(GetController());
    if (UEnhancedInputLocalPlayerSubsystem* Subsystem = ULocalPlayer::GetSubsystem<UEnhancedInputLocalPlayerSubsystem>(PlayerController))
    {
        Subsystem->AddMappingContext(DefaultMappingContext, 0);
        //Subsystem->RemoveMappingContext(DefaultMappingContext);
    }
}

void AABCharacterPlayer::SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    UEnhancedInputComponent* EnhancedInputComponent = CastChecked<UEnhancedInputComponent>(PlayerInputComponent);

    EnhancedInputComponent->BindAction(JumpAction, ETriggerEvent::Triggered, this, &ACharacter::Jump);
    EnhancedInputComponent->BindAction(JumpAction, ETriggerEvent::Completed, this, &ACharacter::StopJumping);
    EnhancedInputComponent->BindAction(MoveAction, ETriggerEvent::Triggered, this, &AABCharacterPlayer::Move);
    EnhancedInputComponent->BindAction(LookAction, ETriggerEvent::Triggered, this, &AABCharacterPlayer::Look);
}

```

#### ▼ Move와 Look함수 구현코드

```

void AABCharacterPlayer::Move(const FInputActionValue& Value)
{
    FVector2D MovementVector = Value.Get<FVector2D>();

    const FRotator Rotation = Controller->GetControlRotation();
    const FRotator YawRotation(0, Rotation.Yaw, 0);

    const FVector ForwardDirection = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);
    const FVector RightDirection = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);

    AddMovementInput(ForwardDirection, MovementVector.X);
    AddMovementInput(RightDirection, MovementVector.Y);
    //IA애셋에서 Modifier를 통해 입력의 축을 YXZ로 변환해주었기에, 굳이 코드부에서 X,Y를 반대로 넣을필요가 없더라
}

void AABCharacterPlayer::Look(const FInputActionValue& Value)
{
    FVector2D LookAxisVector = Value.Get<FVector2D>();

    AddControllerYawInput(LookAxisVector.X);
    AddControllerPitchInput(LookAxisVector.Y);
}

```

#### Summary

- 액터는 컴포넌트를 담는 **포장상자**라고 이해하면 편하다. 실질적인 기능은 컴포넌트에서 구현되지만, 이것들을 묶어 하나의 개체로서 표현할때는 액터로 사용하는 것이 편리하기 때문이다.
- UE5에서 도입된 **EnhancedInputSystem**을 사용해보자. 기존 입력과 함수의 Mapping을 하드코딩해줘야 했던 문제를 해결하는 방법이다.

## 0n. 캐릭터와 입력 시스템 강의 과제

### Q1. 자신의 게임 내에서 사용할 입력에 대해 정리하시오.

입력	내용
이동 ( WASD )	탐류시점의 게임으로 XY축으로 이동하게 된다.
무기발사 ( Left Click )	마우스 왼쪽 클릭을 통해 마우스 커서의 위치로 총알을 발사한다.
무기변경 ( 1, 2, 3 )	숫자 입력을 통해, 해당 슬롯의 무기로 변경한다.

입력	내용
건설 UI 토글 ( B )	B 버튼을 입력하여 접혀진 UI를 활성화시키거나 비활성화 시킨다.
건설 확인 ( Left Click )	건설UI에서 특정 건물을 선택하거나, 선택건물의 생성 위치를 설정할때 사용한다.

## Q2. 이러한 입력 처리를 어디서 구현할지 정리하시오. (PlayerController, Character)

이동, 무기발사, 무기변경은 Character에서 구현해야 한다. 해당 캐릭터의 특징이자 그 캐릭터만 가져야하는 조작법이기 때문이다.

건설UITo글, 건설확인은 PlayerController에서 구현해야 한다. 모든 플레이어가 공통적으로 가져야하는 기능이며 Pawn에서 UI와 게임시스템(건설)에 접근하는 것은 부적절하다 생각한다.