


2단계_03. 캐릭터 컨트롤러 설정

📅 수강일	@ 2023/06/28
➦ 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



- 캐릭터 컨트롤러 설정
 - [1] 캐릭터 컨트롤 요소
 - Control Rotation
 - 스프링암의 컨트롤 옵션
 - 카메라의 컨트롤 옵션
 - 캐릭터 무브먼트의 이동옵션
 - [2] 데이터 애셋
 - [3] Input Mapping Context 전환
 - 0n. 캐릭터 컨트롤러 설정 강의 과제

캐릭터 컨트롤러 설정



강의 목표

- 캐릭터 컨트롤을 위한 각종 설정 옵션의 이해
- 데이터 애셋을 활용한 설정값의 체계적인 관리
- 입력매핑컨텍스트 활용법의 이해

[1] 캐릭터 컨트롤 요소



컨트롤러 + 폰 + 카메라 + 스프링암 + 캐릭터 무브먼트

Control Rotation

Control Rotation은 컨트롤러의 Rotation 프로퍼티 값으로 Pawn의 Rotation과는 **별개의 속성**이다.

앞선 강좌에서 구현한 `Look()` 함수를 살펴보면 현재 컨트롤러의 Control Rotation에 마우스 이동에 따라 새로운 Rotation을 더해주는 식으로 업데이트가 진행되며 `Move()`에서는 그렇게 얻어낸 Control Rotation의 정면벡터와 우측벡터를 통해서 캐릭터의 이동방향을 설정하게 된다.



콘솔 커맨드창(~키)에서 DisplayAll PlayerController ControlRotation을 통해 해당 프로퍼티를 볼 수 있다.

나아가 캐릭터의 Blueprint에 들어가 액터의 Detail패널에서 Pawn을 검색해보면, Use Controller Rotation Pitch/Yaw/Roll 속성을 확인할 수 있다.

▼ Pawn

Use Controller Rotation Pitch	<input type="checkbox"/>
Use Controller Rotation Yaw	<input type="checkbox"/>
Use Controller Rotation Roll	<input type="checkbox"/>

Desired Rotation = Control Rotation

일정 속도로 회전하는 것이 아닌 바로 동기화

여기서 해당 체크가 나온 이유가 등장한다.

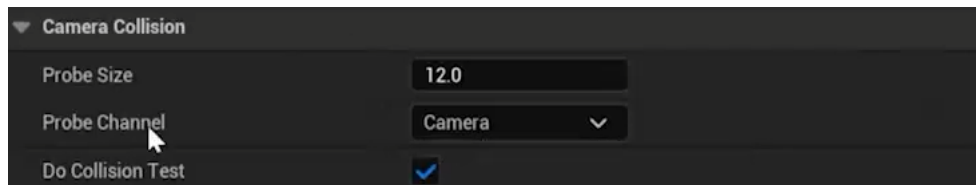
앞서 Control Rotation과 Pawn의 Rotation은 별개의 속성이라고 말했다. 그러나 Use Controller Rotation에 체크를 해준다면 해당 Pawn은 ControlRotation에 맞춰 곧바로 회전(동기화)하게 된다.

즉, 이 방법을 통해 캐릭터의 자연스러운 회전과 더불어 조작법(1인칭/3인칭 조작)을 구현할 수 있다.

! Detour Crowd AI Controller를 사용했을때, AI가 급격하게 변화하는 경로에 맞춰 회전하다보니 DesiredRotation이 수시로 바뀌어 마치 부르르 떨듯 회전하는 **Jerk현상**이 발생한다.
이때, Use Control Rotation의 체크를 해제해준다면 이 현상을 해결할 수 있지 않을까?

스프링암의 컨트롤 옵션

위와 동일하게 스프링암에도 Use Pawn Control Rotation이 존재한다. 1인칭 또는 3인칭과 같은(마치 배틀필드와 배틀그라운드의 시점 차이) 시점의 변화를 줄 수 있다.



스프링암에서 주목할만한 옵션은 **Do Collision Test**이다. 캐릭터와 카메라 사이에 장애물이 발생할때, 카메라를 벽앞으로 당겨주는 효과를 얻을 수 있다. 이때, Collision여부를 Trace채널을 통해서 관리한다.

카메라의 컨트롤 옵션

Use Pawn Control Rotation이 존재한다. 똑같이 Pawn의 ControlRotation에 동기화하는 기능으로 주로 1인칭 카메라 회전에 사용된다. 단, 카메라를 스프링암과 함께 사용한다면, 두 컴포넌트의 UsePawnControlRotation 관계를 고려해줘야 한다.

캐릭터 무브먼트의 이동옵션

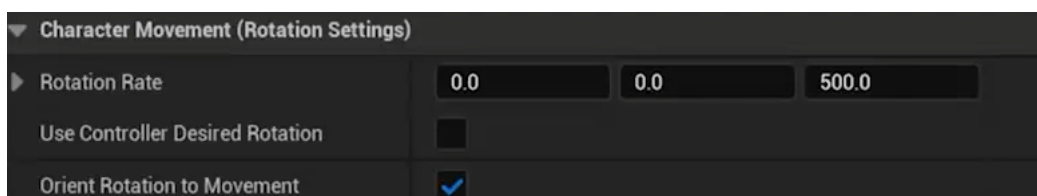
언리얼 엔진에서는 캐릭터의 이동상태에 관하여 다양한 모드를 제공한다.

모드	설명
None	걸지 않음
Walking	땅 위에서 걸음
Falling	땅 위에 존재하지 않음
Swimming	이름 그대로의 의미로, 사실 해당 위치에서 수영하거나 비행하기 보다는 해당 모드가 범용적이며 그런 모드에 대한 카테고리를 만들기 위해 사용한다
Flying	“
Navmesh Walking	환경지물과 모든 Collision을 무시하며 오직 NavigationMesh와만 상호작용하는 걷는 모드



해당 모드의 수치값을 목록화시킨 카테고리라고 생각하는 것이 더 좋을것 같다.

전에 물 속에서 수영하거나 하늘을 날 때 모두 Swimming 카테고리를 사용하여, 속도나 애니메이션 출력등을 손쉽게 다뤘던 적이 있다.



또한 UseControllerDesiredRotation과 RotationRate를 조정하여, Control Rotation을 DesiredRotation으로 사용할지 혹은 회전할때 어느정도 비율로 회전할지 등을 조절할 수 있다.

추가로 **Orient Rotation to Movement**를 주목해야하는데, 컨트롤러에서 입력을 통해 캐릭터를 이동시키려 할때, 그 이동방향으로 캐릭터의 Rotation을 회전시키는 기능이다.



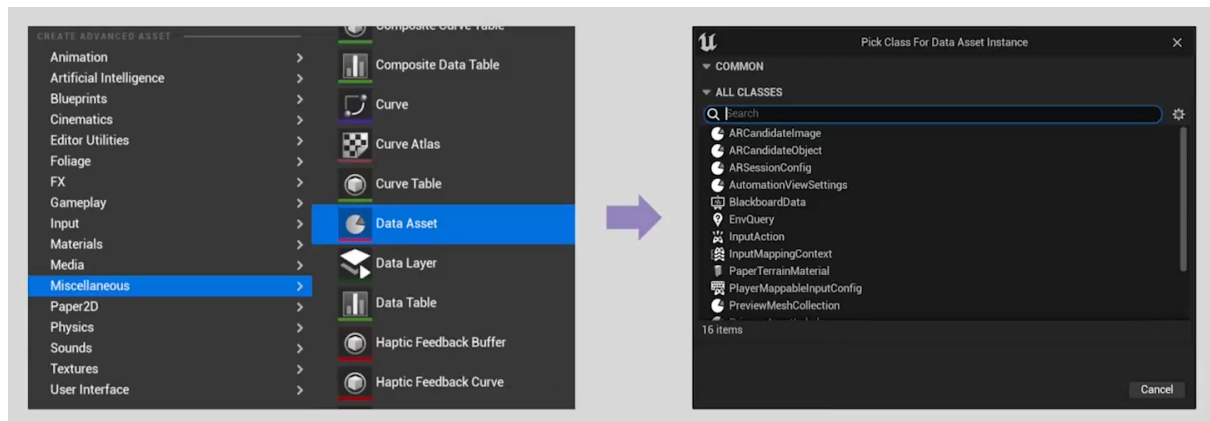
배틀그라운드에서 기본적으로 움직일때는 이동방향을 앞으로 보고 뛰지만, 조준을 하면 카메라 방향을 앞으로 보며 옆걸음 하는걸 떠올려보면 이해가 될 것이다.

[2]데이터 애셋

위 캐릭터 무브먼트 경우에만 봐도 하나의 컴포넌트 안에 설정해줘야 할 프로퍼티가 너무나도 많다.

이것들을 일일이 하나씩 설정해준다면 관리가 매우 힘들 것이다.

그렇기에 이러한 **데이터 옵션들을 에디터 내에서 '애셋'의 형태로 관리**할 수 있게 해주는 것이 **데이터 애셋**이다.



위 방법 뿐 아니라 UDataAsset클래스를 사용해 C++클래스로 생성할 수도 있다.

관리해야할 여러 프로퍼티에 UPROPERTY를 지정해주어 클래스를 생성하고, 이를 상속받는 에디터 내의 Data Asset파일로 만들어 사용하면된다.

```
UABCharacterControlData.h
UCLASS()
class ARENABATTLE_API UABCharacterControlData : public UPrimaryDataAsset
{
    GENERATED_BODY()

public:
    UABCharacterControlData();

    UPROPERTY(EditAnywhere, Category = Pawn)
    uint32 bUseControllerRotationYaw : 1;

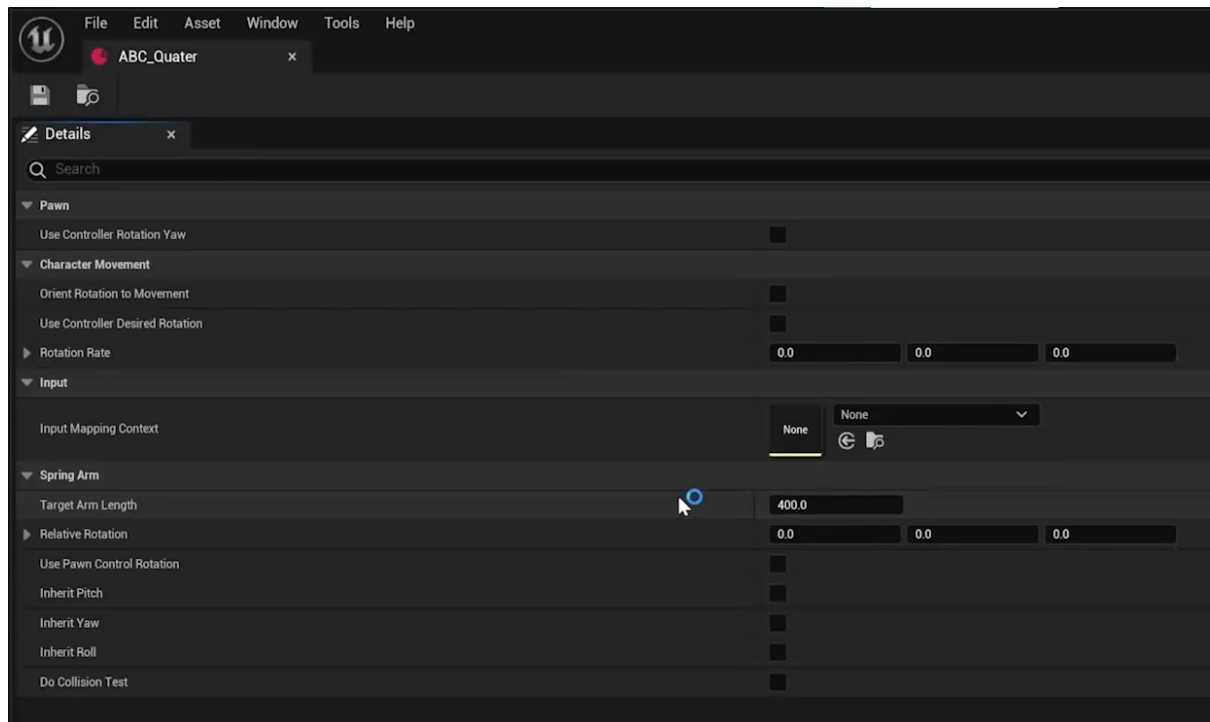
    UPROPERTY(EditAnywhere, Category = CharacterMovement)
    uint32 bOrientRotationToMovement : 1;

    UPROPERTY(EditAnywhere, Category = CharacterMovement)
    uint32 bUseControllerDesiredRotation : 1;

    UPROPERTY(EditAnywhere, Category = CharacterMovement)
    FRotator RotationRate;

    UPROPERTY(EditAnywhere, BlueprintReadOnly, Category = Input)
    TObjectPtr<class UInputMappingContext> InputMappingContext;

    UPROPERTY(EditAnywhere, Category = SpringArm)
    float TargetArmLength; //float값을 헤더에서 설정해주는 것은 효율적이지 못하기에 cpp생성자에 넣었다.
    ...
}
```



사진처럼 내부에서 설정이 가능하다.

이렇게 만들어낸 DataAsset을 설정해주는 함수를 캐릭터 클래스 내에서 만들어 할당해주면 된다.

```
void AABCharacterBase::SetCharacterControlData(const UABCharacterControlData* CharacterControlData)
{
    //DataAsset을 입력으로 받는 함수
    //DataAsset내의 프로퍼티를 옮긴다
    bUseControllerRotationYaw = CharacterControlData->bUseControllerRotationYaw;

    GetCharacterMovement()->bOrientRotationToMovement = CharacterControlData->bOrientRotationToMovement;
    GetCharacterMovement()->bUseControllerDesiredRotation = CharacterControlData->bUseControllerDesiredRotation;
    GetCharacterMovement()->RotationRate = CharacterControlData->RotationRate;
}
```

[3] Input Mapping Context 전환

입력을 통해 다른 Input Mapping Context(조작방법)으로 전환하는 방법을 알아보자.

사실 간단하다. Input Mapping Context를 바꿔주는 함수를 캐릭터 클래스 내에 구현하고, 하나의 InputAction을 추가해 해당 함수를 맵핑 해주면 끝난다.

본 강의에서는 해당 함수를 구현할때, 쿼터뷰 시점의 데이터애셋과 탑뷰 시점의 데이터애셋의 프로퍼티 또한 변환시켜주는 코드를 추가함으로써 손쉽게 InputMappingContext와 시점데이터를 동시에 변경할 수 있었다.

```
ABCharacterPlayer.cpp
...
void AABCharacterPlayer::SetCharacterControl(ECharacterControlType NewCharacterControlType)
{
    UABCharacterControlData* NewCharacterControl = CharacterControlManager[NewCharacterControlType];
    check(NewCharacterControl);

    SetCharacterControlData(NewCharacterControl);

    APlayerController* PlayerController = CastChecked<APlayerController>(GetController());
    if (UEnhancedInputLocalPlayerSubsystem* Subsystem = ULocalPlayer::GetSubsystem<UEnhancedInputLocalPlayerSubsystem>(PlayerController))
    {
        Subsystem->ClearAllMappings(); //기존의 맵핑을 clear해주고 아래서 새로 연결해주는 것이 포인트
        UInputMappingContext* NewMappingContext = NewCharacterControl->InputMappingContext;
        if (NewMappingContext)
        {
            Subsystem->AddMappingContext(NewMappingContext, 0);
        }
    }
}
```

```
CurrentCharacterControlType = NewCharacterControlType;
}
...
```

Summary

- Control Rotation과 Desired Rotation의 관계 이해
- 데이터를 애셋 형태로 관리하는 DataAsset의 이해
- Runtime 도중 InputMappingContext의 변경을 통한 조작법 변경

0n. 캐릭터 컨트롤러 설정 강의 과제

Q1. 자신의 게임에서 사용할 시점에 대해 정리하시오. (1인칭 또는 3인칭)

하늘에서 바라보는 쿼터뷰시점을 사용할 예정이다.

Q2. 자신의 게임에서 조작할 캐릭터의 움직임에 대해 정리하시오.

쿼터뷰이기에 컨트롤러의 Control Rotation을 사용하지 않고 XY축 입력에 맞춰 움직이게 된다.

Q3. 이를 구현하기 위해 어떤 값을 설정할지 정리하고 이 값들을 애셋으로 관리해보시오.

캐릭터는 컨트롤러의 Control Rotation과 관계없이 상하좌우로 이동할 것이며 캐릭터의 Rotation 또한 이동방향에 관계 없이 마우스 커서를 향해야 한다.

카메라 또한 Pawn의 Control Rotation에 관계 없이 각도로 화면을 비추어야 한다.

즉,

- Pawn :
 - Use Controller Rotation Yaw = false
- Character Movement :
 - Orient Rotation to Movement = false
- Camera :
 - Use Pawn Control Rotation = false

Reference

Purpose of Project Navmesh Walking?

In recent version 4.8 there is this feature available for character actors, saying in the release note: New: Added optional raycast to conform NavMesh walking closer to underlying geometry I would like to ask to please extend this explanation or point to documentation I can't find. How this feature affect performance?

 <https://forums.unrealengine.com/t/purpose-of-project-navmesh-walking/323506>

