UE4 입력 시스템과 UE5 Enhanced Input의 차이점

1. UE4 입력 시스템 (Legacy Input)

UE4에서 사용되던 기존 입력 시스템은 상대적으로 단순한 구조를 가지고 있습니다.

특징

- **Action Mapping / Axis Mapping:**
  - `Project Settings > Input`에서 키 바인딩을 설정할 수 있음.
  - `Action Mapping`: 버튼을 누르거나 떼는 등의 이진 입력 (예: 점프, 공격).
  - `Axis Mapping`: 연속적인 값(예: 이동, 카메라 회전).
- **Blueprint 및 C++에서 직접 바인딩**
  - `InputComponent->BindAction()`을 사용해 입력을 직접 코드에서 바인딩.
- **입력 우선순위**
  - 플레이어 컨트롤러(PlayerController) → 폰(Pawn) → 컴포넌트(Component) 순으로 입력이 전달됨.

단점

1. **입력 컨텍스트 부족**
   - 다른 상태(예: 차량 운전, 1인칭 슈팅 모드, 메뉴 조작 등)에서 다른 입력을 사용하려면 복잡한 조건문이 필요함.
2. **복잡한 입력 리매핑**
   - 실행 중 키를 변경하는 기능이 제한적이며, 특정 UI를 만들어야 함.
3. **멀티플랫폼 지원 부족**
   - 키보드, 컨트롤러, VR 등의 입력을 통합적으로 처리하는 기능이 미흡.
4. **듀얼 입력 지원 미흡**
   - 키보드와 컨트롤러를 동시에 사용하거나, 특정 키 조합을 쉽게 처리하는 기능이 부족.

2. UE5 Enhanced Input 시스템

UE5에서는 기존 입력 시스템을 대체하는 **Enhanced Input**이 도입되었습니다.

특징

- **입력 컨텍스트** (Input Context)
  - 상황(예: 캐릭터 조작, UI 인터페이스 등)에 맞춰 동적으로 입력을 변경할 수 있음.
- **멀티플랫폼 입력 지원 강화**
  - 키보드, 게임패드, VR, 터치스크린 등 다양한 입력 장치를 통합적으로 지원.
- **입력 매핑 런타임 수정 가능**
  - 플레이어가 게임 내에서 직접 키 바인딩을 변경할 수 있도록 설계됨.
- **입력 이벤트 기반 처리**
  - `Started`, `Triggered`, `Canceled`, `Completed` 등의 이벤트를 활용하여 입력을 세밀하게 조작 가능.
- **Input Mapping Context 활용**
  - `Input Mapping Context`를 사용해 특정 상황에서 입력을 활성화/비활성화 가능.

장점

1. **상황별 입력 처리 가능**
   - 캐릭터가 차량을 탈 때, 전투 모드에 들어갈 때 등 자동으로 입력을 변경할 수 있음.
2. **입력 이벤트 기반 동작**
   - 기존 `Pressed`, `Released`만 제공하던 방식에서 더욱 세분화된 이벤트 제공.
3. **플레이어별 개별 설정 가능**
   - 동일한 컨트롤러를 사용하는 두 명의 플레이어가 각자 다른 입력 설정을 가질 수 있음.
4. **블루프린트와 C++ 통합 지원**
   - 블루프린트에서도 쉽게 사용할 수 있으며, C++에서도 확장 가능.
5. **스케일러블** (Scalable)
   - 모바일, 콘솔, PC 등 다양한 플랫폼에서 효율적으로 작동.

UE4 입력 시스템과 UE5 Enhanced Input의 차이점 (상황별 예시)

두 입력 시스템의 차이를 보다 직관적으로 이해할 수 있도록, **FPS 게임에서 "무기 조준(Aim)과 발사(Fire)"를 구현하는 예제**를 통해 비교해보겠습니다.

🟢 UE4 입력 시스템 (Legacy Input)

예제: FPS 게임에서 **오른쪽 마우스 버튼**을 누르면 **조준(Aim)** 상태가 되고, **왼쪽 마우스 버튼**을 누르면 **발사(Fire)** 기능을 실행.

UE4 방식 구현 (기본 입력 시스템 사용)

1. **입력 바인딩 설정** (`Project Settings > Input`)
    - `Action Mapping`
        - `"Aim"` → Right Mouse Button
        - `"Fire"` → Left Mouse Button

2. C++ 에서 입력 처리

```cpp
void AMyCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    PlayerInputComponent->BindAction("Aim", IE_Pressed, this, &AMyCharacter::StartAiming);
    PlayerInputComponent->BindAction("Aim", IE_Released, this, &AMyCharacter::StopAiming);

    PlayerInputComponent->BindAction("Fire", IE_Pressed, this, &AMyCharacter::FireWeapon);
}

void AMyCharacter::StartAiming()
{
    bIsAiming = true;
    // 조준 애니메이션 시작
}

void AMyCharacter::StopAiming()
{
    bIsAiming = false;
    // 일반 상태로 복귀
}

void AMyCharacter::FireWeapon()
{
    if (bIsAiming)
    {
        // 조준 사격 로직
    }
    else
    {
        // 일반 사격 로직
    }
}
```

UE4 방식의 문제점

1. **입력 상태(Context)에 대한 별도 관리 필요**
    - `bIsAiming` 같은 변수를 직접 관리해야 함.
    - 캐릭터가 다른 상태(예: 차량 운전 모드, 스나이퍼 모드 등)로 변할 때 상태 관리가 복잡해짐.
2. **새로운 입력을 추가할 때 번거로움**
    - 예를 들어, "조준 중에는 특수한 발사 방식이 활성화됨" 등의 변경이 필요하면 기존 코드를 많이 수정해야 함.
3. **여러 입력 장치에 대한 대응이 부족**
    - 키보드 & 마우스 외에도 컨트롤러, 터치스크린에서 다르게 동작해야 하는 경우, 추가적인 분기 처리가 필요함.

🔵 UE5 Enhanced Input 시스템

Enhanced Input을 사용하면 **입력 컨텍스트**(Input Context)와 **이벤트 기반 처리**가 가능하여 더 유연한 입력 처리가 가능합니다.

UE5 방식 구현 (Enhanced Input 사용)

1. Enhanced Input 시스템 설정
   - `Input Mapping Context (IMC_FPSCharacter)`
   - `Input Action (IA_Aim, IA_Fire)`
2. 입력 바인딩 설정
   - `IA_Aim` → Right Mouse Button
   - `IA_Fire` → Left Mouse Button

3. Enhanced Input 방식의 입력 처리

```cpp
void AMyCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    // Enhanced Input을 사용하기 위한 Enhanced Input Component 가져오기
    UEnhancedInputComponent* EnhancedInput = Cast<UEnhancedInputComponent>(PlayerInputComponent);
    if (EnhancedInput)
    {
        EnhancedInput->BindAction(AimAction, ETriggerEvent::Triggered, this, &AMyCharacter::OnAim);
        EnhancedInput->BindAction(FireAction, ETriggerEvent::Triggered, this, &AMyCharacter::OnFire);
    }
}

void AMyCharacter::OnAim(const FInputActionValue& Value)
{
    bool bIsAiming = Value.Get<bool>();
    // 조준 상태 토글
    bAiming = bIsAiming;
}

void AMyCharacter::OnFire(const FInputActionValue& Value)
{
    if (bAiming)
    {
        // 조준 사격 로직
    }
    else
    {
        // 일반 사격 로직
    }
}
```

UE5 Enhanced Input 방식의 장점

1. **입력 컨텍스트(Context) 활용 가능**
   - 캐릭터가 차량을 탈 때는 `IMC_Driving`(운전 컨텍스트)로 변경,
     캐릭터 상태가 달라질 때 별도 분기문 없이 동적 입력 변경 가능.
2. **입력 이벤트 기반 처리**
   - `ETriggerEvent::Triggered`, `Started`, `Completed` 같은 다양한 이벤트 사용 가능.
   - 기존 `Pressed`, `Released`보다 더 정밀한 조작이 가능함.
3. **멀티플랫폼 대응이 쉬움**
   - 키보드 & 마우스뿐만 아니라 컨트롤러, VR 기기 등도 손쉽게 지원 가능.
4. **동적 키 리매핑 가능**
   - 게임 내에서 UI를 통해 키를 변경할 수 있음.

🔵 추가적인 상황 예시

🎮 예제 1: 캐릭터가 차량을 탈 때 입력 변경

UE4 방식

- `bIsDriving` 같은 플래그를 만들어서 `if (bIsDriving) { 차량 조작 } else { 캐릭터 조작 }`처럼 조건문을 추가해야 함.
- 상태가 많아질수록 코드가 복잡해짐.

UE5 방식

- `Input Mapping Context (IMC_Character, IMC_Driving)`를 사용하여
  - 캐릭터가 차량에 탑승하면 `IMC_Driving`을 활성화
  - 차량에서 내리면 다시 `IMC_Character` 활성화
    → 코드 수정 없이 입력 방식이 자동으로 변경됨.

🔫 예제 2: 조준 중 발사 속도 변경

UE4 방식

- `bIsAiming`을 확인하고, 조건문을 추가하여 발사 속도를 변경해야 함.

UE5 방식

- `IA_Fire`에 `Modifiers (Input Modifiers)`를 추가하여
  - 조준 중에는 발사 속도를 0.5배로 조정
  - 일반 상태에서는 1.0배 유지
    → **별도의 조건문 없이 입력 데이터 자체를 변경 가능.**

💡 결론

비교 항목	UE4 Legacy Input	UE5 Enhanced Input
입력 관리 방식	`if-else` 기반 수동 관리	입력 컨텍스트 기반 자동 관리
입력 이벤트	`Pressed`, `Released`	`Triggered`, `Started`, `Completed` 등 다양한 이벤트 지원
키 리매핑	실행 중 변경 어려움	런타임 중에도 쉽게 변경 가능
상황별 입력 변경	수동으로 `bIsDriving` 같은 변수를 사용해야 함	`Input Mapping Context`를 통해 자동 변경
멀티플랫폼 지원	키보드 & 컨트롤러 분기 처리 필요	다양한 입력 장치 자동 지원

UE5의 **Enhanced Input 시스템**은 **더 유연하고 직관적인 입력 관리**가 가능하며,
플레이어 경험을 향상시키고, **개발자의 코드 유지보수 부담을 줄여주는** 강력한 시스템입니다! 🚀