액션 시스템

저의 주요 프로젝트인 DefendTheDungeon의 CombatComponent 코드와 주요 기능 중 하나인 액션시스템에 대한 설명입니다.

이 시스템은 우선순위 기반으로 캐릭터의 다양한 액션들(공격, 스킬, 대시, 블록 등)을 체계적으로 관리하며, 멀티플레이어 환경에서 안정적으로 동작하도록 설계되었습니다.

▼ 내부 코드

```
bool UCombatComponent::TryPlayAction_Internal(FAction& Action)
{
    if (GetNetMode() != NM_ListenServer)
    {
        MY_LOG(LogTemp, Error, TEXT("TryPlayAction Called in Client"));
        return false;
    }

    if (!CheckValidAction(Action)) return false;

    if (Action.ActionLevel < 0) Action.ActionLevel = 0;
    else if (Action.ActionLevel > ActionMax) Action.ActionLevel = Action
Max;

    if (Action.CancelLevel < 0) Action.CancelLevel = 0;
    else if (Action.CancelLevel > ActionMax) Action.CancelLevel = Action
Max;

    if (!CanPlayAction(Action.ActionLevel))
    {
        MY_LOG(LogTemp, Log, TEXT("Try Action Level %d < Cur Cancel
Level %d, denied. Owner %s, ActionName %s, ActionType %s"), Actio
n.ActionLevel, CurAction.CancelLevel, *GetNameSafe(Action.Owner), *
Action.ActionName.ToString(), *UEnum::GetValueAsString(Action.Actio
nType));
        return false;
    }
```

```
    //전 액션과 동일한 경우, 건너뛴다.
    if (CurAction != Action)
    {
        //이전 액션 취소 함수 호출
        if (ActionCancelCalled.ExecuteIfBound()) ActionCancelCalled.Clea
r();

        //취소 함수 바인딩
        if (!Action.CancelFunctionName.IsNone())
            ActionCancelCalled.BindUFunction(Action.Owner, Action.Cancel
FunctionName);

        //취소 함수가 비워져 있을 땐, 기본 상태로 돌아간다 생각한다.
        else
        {
            ActionEnded.BindUFunction(this, "SetDefaultAction");
        }

        //엔드 함수 바인딩
        ActionEnded.Clear();
        if (!Action.EndFunctionName.IsNone())
            ActionEnded.BindUFunction(Action.Owner, Action.EndFunctionN
ame);

        //엔드 함수가 비워져 있을 땐, 기본 상태로 돌아간다 생각한다.
        else
        {
            ActionEnded.BindUFunction(this, "SetDefaultAction");
        }

        //새로운 액션 바인딩
        ActionCalled.Clear();
        ActionCalled.BindUFunction(Action.Owner, Action.PlayFunctionNa
me);
    }

    //액션 실행
    if (ActionCalled.ExecuteIfBound())
```

```
    {
        MY_LOG(LogTemp, Log, TEXT("Action Called, Owner %s, ActionName %s, ActionType %s"), *GetNameSafe(Action.Owner), *Action.ActionName.ToString(), *UEnum::GetValueAsString(Action.ActionType));
        CurAction = Action;
    }
    else
    {
        MY_LOG(LogTemp, Warning, TEXT("Action Call Failed, Owner %s, ActionName %s, ActionType %s"), *GetNameSafe(Action.Owner), *Action.ActionName.ToString(), *UEnum::GetValueAsString(Action.ActionType));
        return false;
    }

    return true;
}
```

문제 인식

기존 캐릭터 시스템은 입력과 상황 조건에 따라 액션이 경쟁적으로 즉시 실행되는 방식이었습니다.

하지만 게임 볼륨이 커지고 전투 상황이 다양해지면서 동시에 여러 행동이 겹치는 복잡한 문제들이 발생했습니다.

이를 해결하기 위해 모든 캐릭터 애니메이션 및 행동을 한 곳에서 관리하고, 각 행동에 **우선순위**를 부여해 동시에 여러 입력이나 이벤트가 들어와도 체계적으로 실행·취소·중단 로직이 동작하도록 하는 액션 관리 시스템을 설계하게 되었습니다

▼ 액션 타입 정의(Type)

```
UENUM(Blueprintable)
enum EActionType
{
    Normal,         // 일반 액션
    Dead,           // 사망 상태
    CrowdControl,   // 군중 제어 효과
```

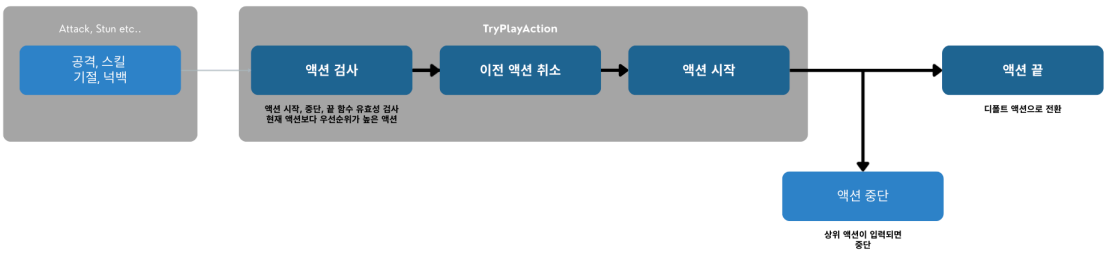
```
    SmallKnockback, // 소형 넉백
    Skill,          // 스킬 액션
    Attacking,      // 공격 중
    Block,          // 방어
};
```

▼ 액션 구조체(FAction)

```
USTRUCT(Blueprintable)
struct FAction
{
    GENERATED_BODY()

    //액션 소유 객체
    UPROPERTY()
    UObject *Owner;

    //액션 타입
    UPROPERTY()
    TEnumAsByte<EActionType> ActionType;

    //실행 함수
    UPROPERTY()
    FName PlayFunctionName;

    //취소 함수
    UPROPERTY()
    FName CancelFunctionName;

    //엔드 함수
    UPROPERTY()
    FName EndFunctionName;

    //실행 우선순위
    UPROPERTY()
    int32 ActionLevel;
```

```
    //취소 우선순위
    UPROPERTY()
    int32 CancelLevel;

    //액션 이름, ActionName이 달라도 다른 속성이 동일한 경우, 같은 Action으로 취급합니다.
    UPROPERTY()
    FName ActionName;

    //생성자, 연산자 정의 etc..
};
```

생명 주기



공격, 스킬 등의 플레이어 입력과, 스턴, 넉백 등의 외부 입력이 들어오면
입력 정보는 Action 구조체로 변환되어 CombatComponent의 액션 시스템이 처리합니다.

액션 시스템에서는 입력 받은 Action에 대해 다음 항목을 검사합니다.

- 행동 중 호출을 위한 함수가 유효한지
- 지금 이 행동을 할 수 있는지(우선순위 검사)

검사가 통과하면, 이전 실행하고 있는 Action의 중단 함수를 실행합니다.

중단 후, 새로운 Action의 실행 함수와 중단 함수, 엔드 함수를 바인딩하고, 실행 함수를 실행시킵니다.