Dungeon In Fighter

한융희

목차

1. 개발환경 및 게임소개

게임 엔진	Unreal Engine (4.26.1)
IDE	Visual Studio 2019
주요 에셋들	언리얼 마켓플레이스 믹사모

게임명	Dungeon In Fighter
장르	쿼터뷰 형식의 RPG
타겟플랫폼	PC
플레이 방법	마우스로 캐릭터 이동 및 공격, 키보드 키로 각종 상호작용 가능

2. 개발기간 (2021-03-01 ~ 2021-03-26)

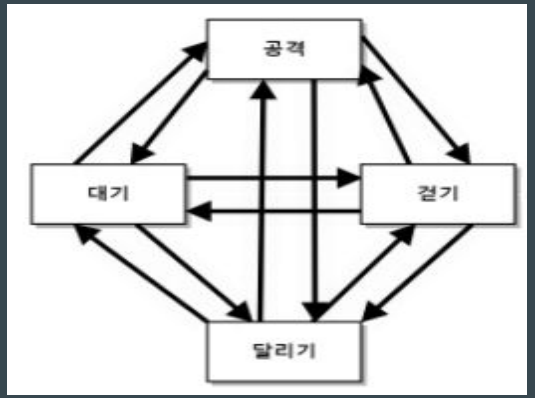
프로토 타입 기간 (03.01 ~ 03.07)	상태기계를 이용한 플레이어와 적의 주요 애니메이션들 구현
알파 버전 기간 (03.08 ~ 03.14)	TraceByChannel을 이용한 피격,타격 판정 구현 및 적의 인공지능 구현 게임의 UI 외관 구현
베타 버전 기간 (03.15 ~ 03.21)	상속을 이용한 아이템을 만들어 다형성 구현 배열을 이용한 인벤토리와 퀵슬롯 기능구현
최종 버전 기간 (03.22 ~ 03.26)	Drag and Drop을 이용한 슬롯(인벤토리,퀵슬롯)간 교환 구현 대쉬 기능과 쿨타임바 구현

3-1 상태기계를 이용한 플레이어와 적의 주요 애니메이션들 구현

FSM(Finite State Machine, 유한상태기계)이란?

: 유한 상태 기계는 유한개의 상태를 가지고,

전이조건에 따라 다른 상태로 변화하는 모델

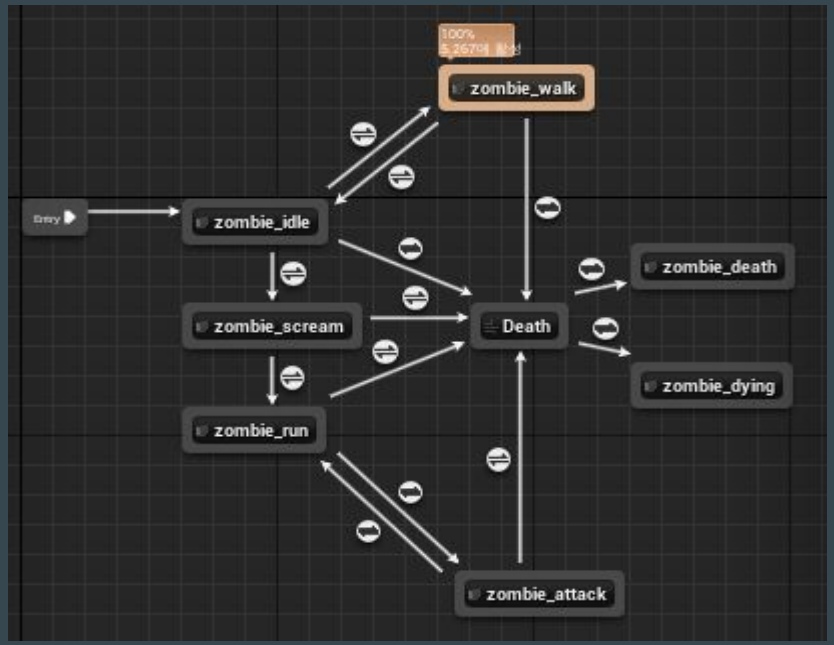


FSM을 사용하는 이유?

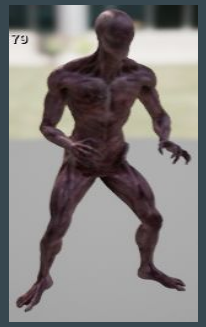
직관적	→	코드나 아닌 도표로 표현되므로, 프로그래머가 아닌 기획자 혹은 디자이너와의 협업이 쉬워진다.
코드 수정 용이	→	정해진 룰이 있기때문에 상태의 추가/삭제가 용이

3-1 상태기계를 이용한 플레이어와 적의 주요 애니메이션들 구현

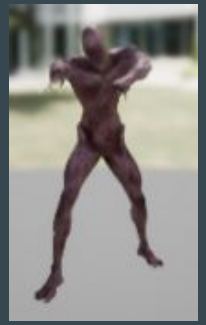
Unreal Engine에서 제공하는 애니메이션 블루프린트를 이용함



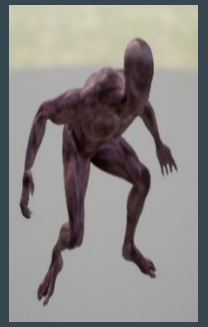
[적의 애니메이션 블루프린트]



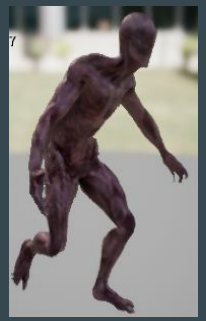
[idle 상태]



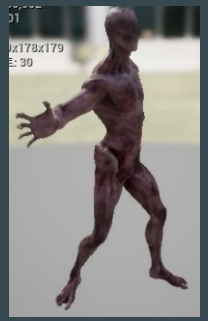
[walk 상태]



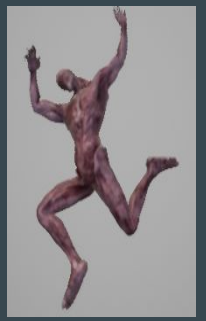
[scream 상태]



[run 상태]

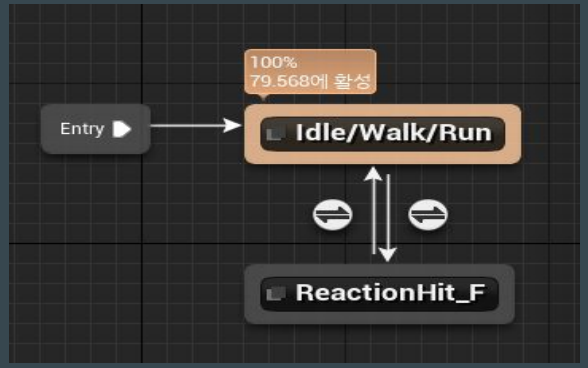


[attack 상태]

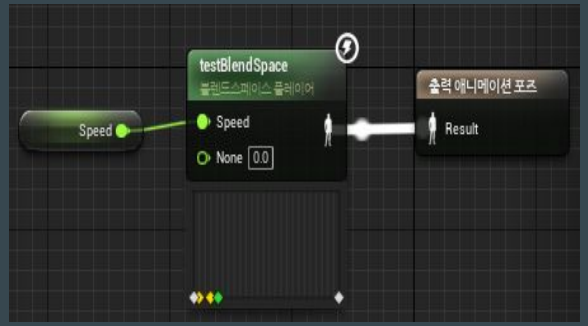


[death 상태]

3-1 상태기계를 이용한 플레이어와 적의 주요 애니메이션들 구현

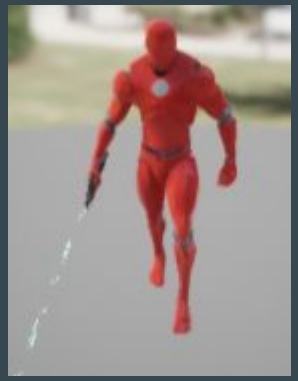


[플레이어의 애니메이션 블루프린트]

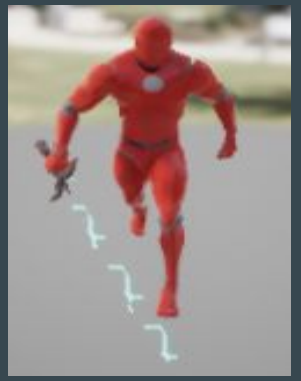


[블렌드스페이스를 이용]

[Idle 상태]
Speed 값 : 0근처



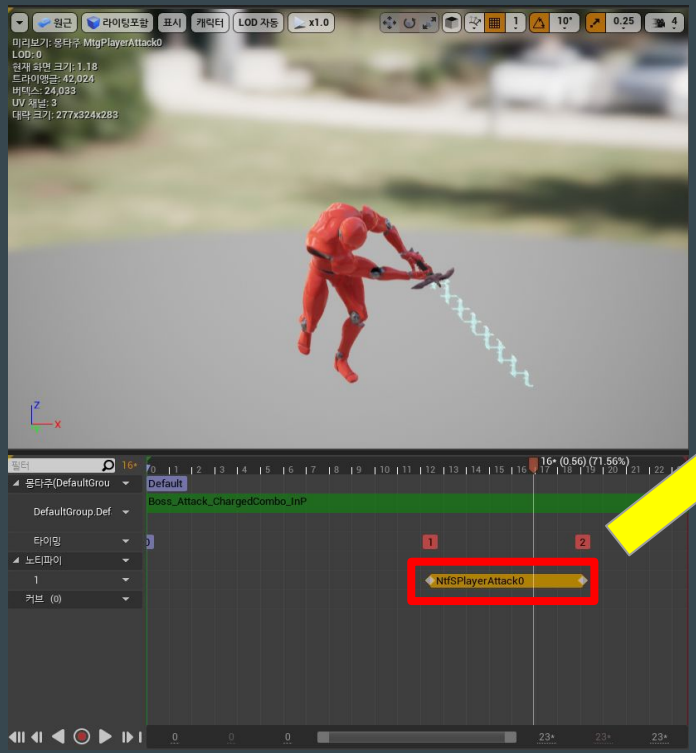
[Walk 상태]
Speed 값 : 120근처



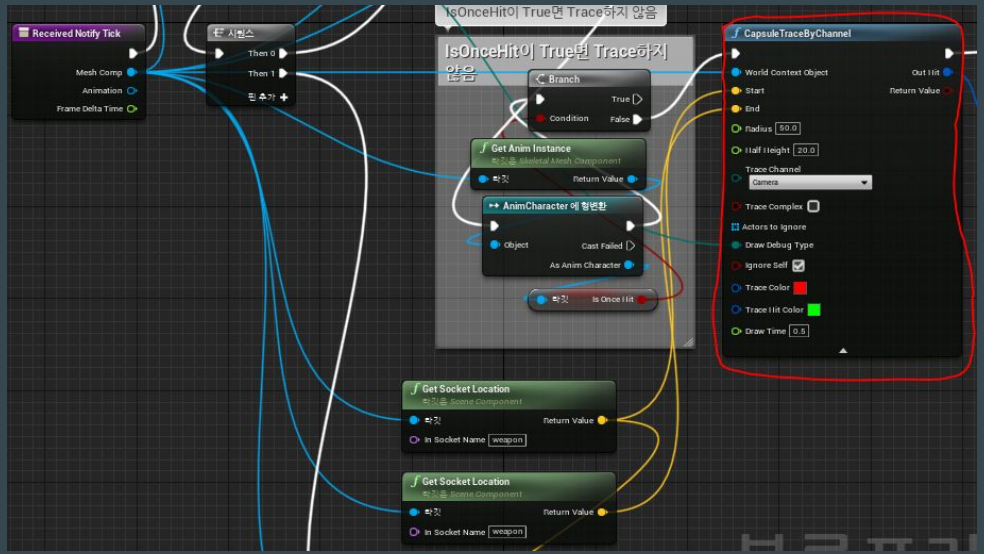
[Run 상태]
Speed 값 : 360근처

전이조건을 bool값이 아닌 float값(Speed)의 변화에 따라 상태를 변화시킬수 있음

3-2 TraceByChannel을 이용한 피격,타격 판정 구현 및 적의 인공지능 구현



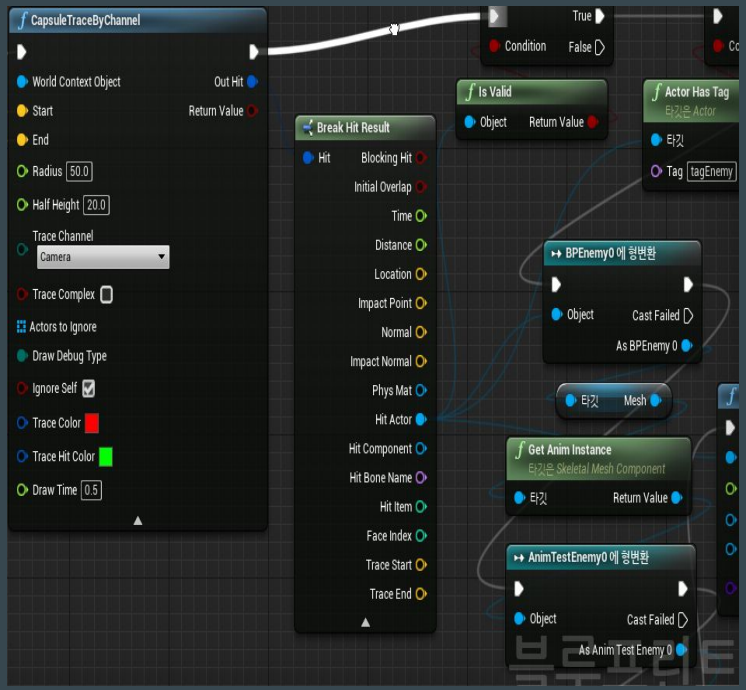
[플레이어 공격 몽타주]



[플레이어 공격 몽타주의 Notify State]

Notify State는 애니메이션 특정 구간의 상태를 얻음
ex) 공격 모션의 검을 휘두르는 구간만 충돌체크

3-2 TraceByChannel을 이용한 피격,타격 판정 구현 및 적의 인공지능 구현



CapsuleTraceByChannel을 이용해 tick마다 캡슐형태의 충돌감지를 함

Hit(충돌)된 Actor를 얻고 그 Actor의 태그가 tagEnemy인지 감지함 (아닌경우 계속감지)

충돌체의 태그가 tagEnemy인경우 ApplyDamage를 통해 데미지를 주는 이벤트 발생

3-2 TraceByChannel을 이용한 피격,타격 판정 구현 및 적의 인공지능 구현

Enemy의 Tick 이벤트에서 SphereTraceByChannel 로 감지범위 만듦 (랜덤으로 이동중)

플레이어 감지시 Scream 상태로 전이

Scream상태가 끝난후 플레이어 위치로 이동(Run상태)후 Attack 상태로 전이

※ 감지범위를 TraceByChannel로 만든 이유 :

단순히 적 Actor에 Collision물체를 달아서 충돌감지를 하는것이 Tick단위로 Trace하는 것보다 성능향상이 될거같지만 어느정도의 성능을 포기하고 TraceByChannel의 감지범위를 보이고 안보이게 하는 등의 디버그 기능이 더 좋은거 같아서 사용했다.

3-3 상속을 이용한 아이템을 만들어 다형성 구현

Item
(추상크래스, 부모클래스)

ItemStructure : 아이템정보구조체
(변수)

- 아이템명
- 중첩가능?
- 아이템 이미지
- 최대중첩수
- Item클래스

UseItem : 아이템사용
(함수)

Interact : 플레이어가 상호작용
(인터페이스)

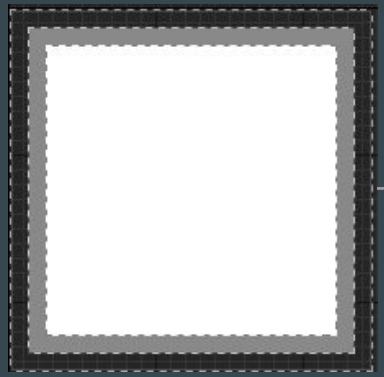
체력 포션
(자식클래스)

마나 포션
(자식클래스)

※ 상속을 사용한 이유 :

1. 코드중복을 막아 코드관리 용이
2. 새 클래스를 추가시키기 쉬워 생산성을 높여줌

3-4 배열을 이용한 인벤토리와 퀵슬롯 기능구현

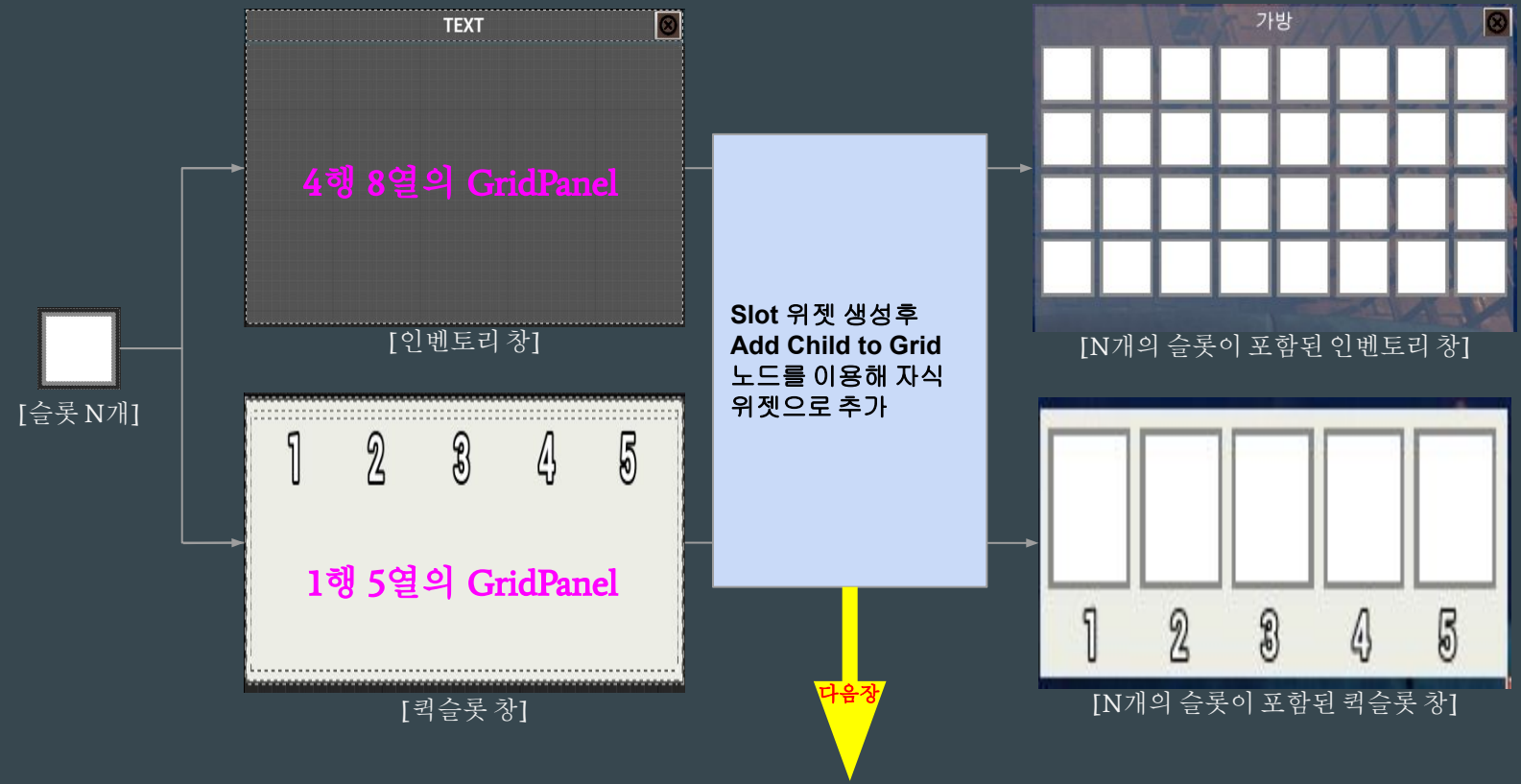


[Slot 위젯]

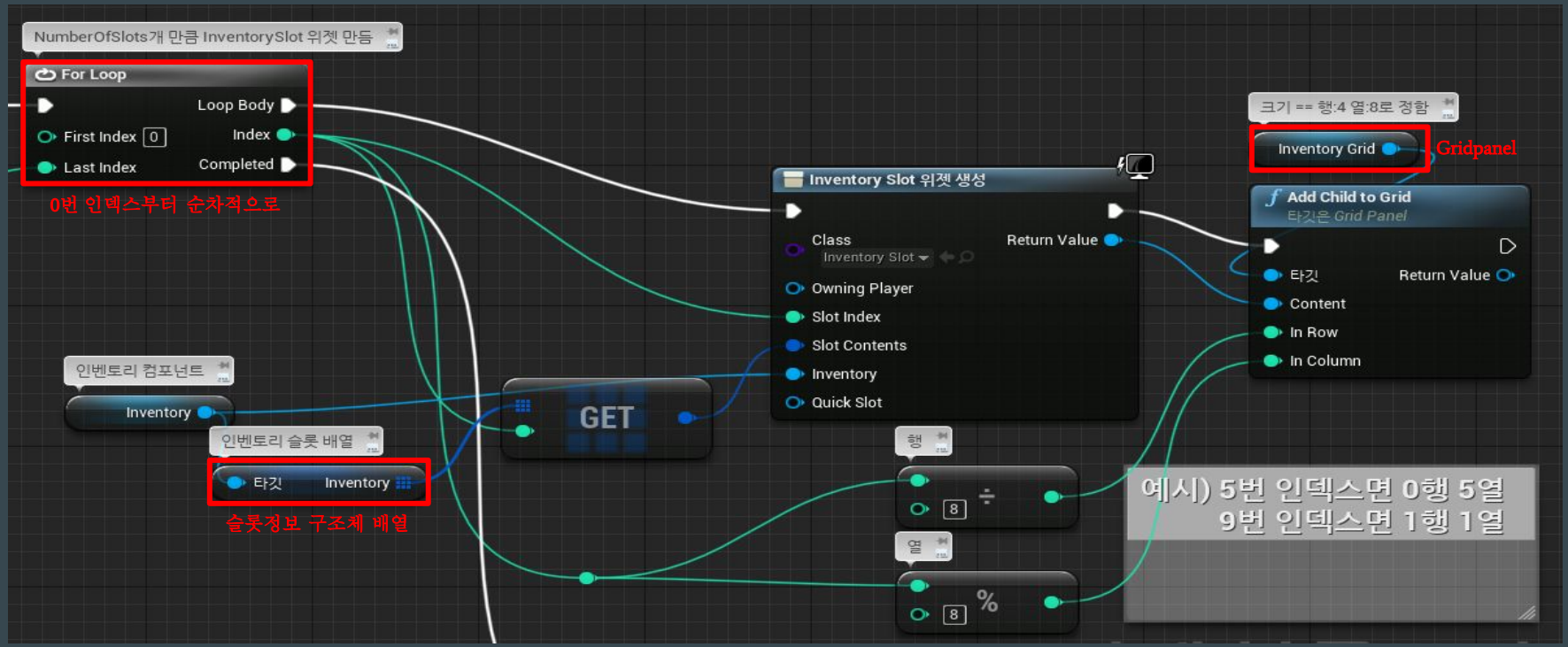
- 아이템 이미지
- 슬롯의 인덱스값
- 슬롯의 정보 구조체
  - 아이템 정보 구조체
  - 아이템 수량

인벤토리와 퀵슬롯은 N개의 슬롯으로 이루어짐

3-4 배열을 이용한 인벤토리와 퀵슬롯 기능구현

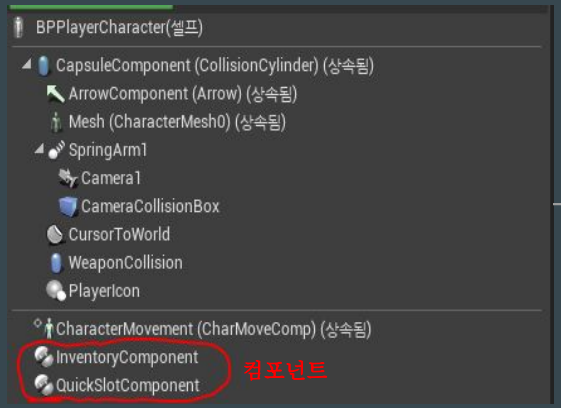


3-4 배열을 이용한 인벤토리와 퀵슬롯 기능구현

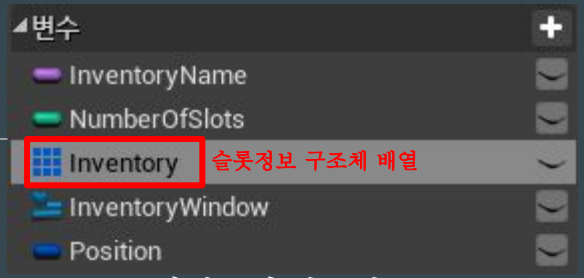


[인벤토리 슬롯 생성과정]

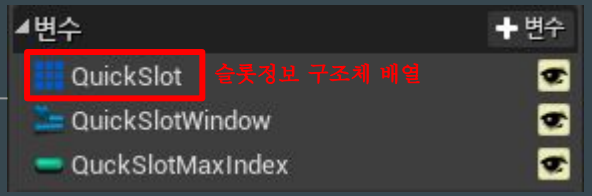
3-4 배열을 이용한 인벤토리와 퀵슬롯 기능구현



[플레이어캐릭터 블루프린트]



[인벤토리 컴포넌트]



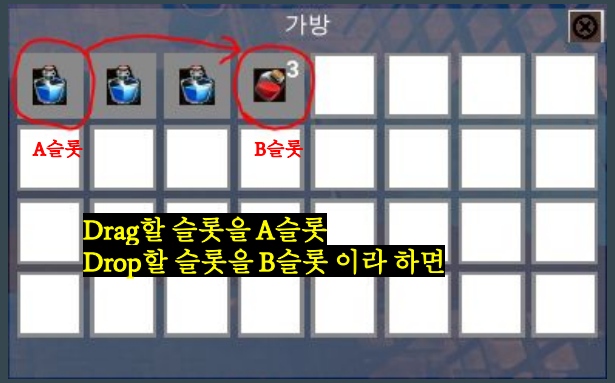
[퀵슬롯 컴포넌트]

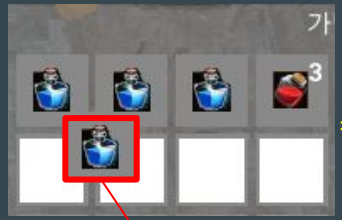
※ 슬롯의 정보를 배열로 구현한 이유 :

1. 인벤토리나 퀵슬롯의 행열은 정해진 크기다. (가변필요X)
2. 아이템과 상호작용해 획득시 0번 인덱스 부터 순차로 채워나감 (중간값 삽입이 없음)
3. 인덱스를 통해 특정 슬롯에 접근가능(자료의 접근,저장 빠름)

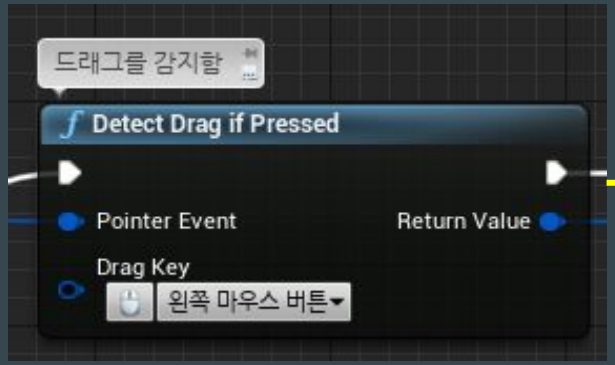
3-5 Drag and Drop을 이용한 슬롯(인벤토리,퀵슬롯)간 교환 구현

3-5 Drag and Drop을 이용한 슬롯(인벤토리,퀵슬롯)간 교환 구현

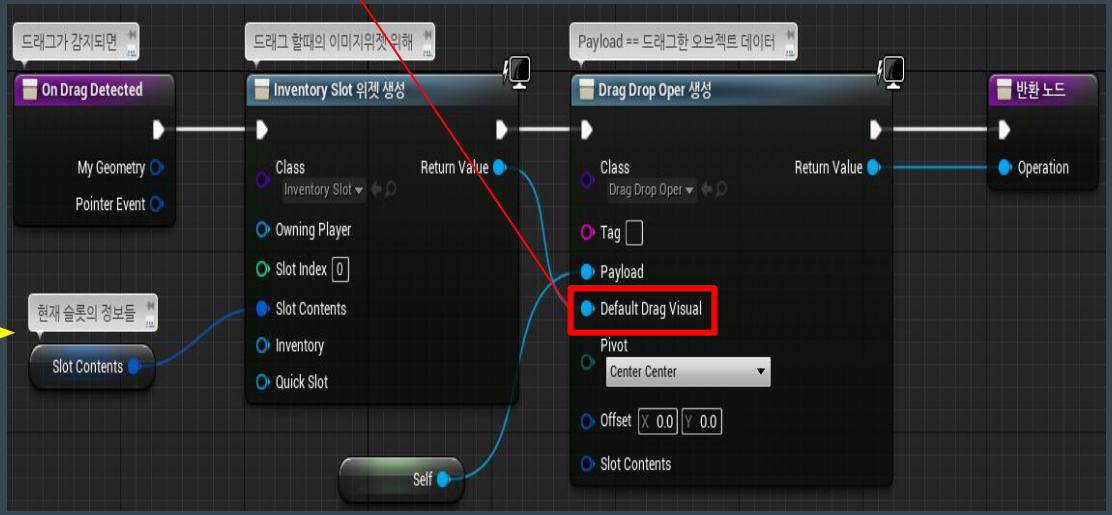




Default Drag Visual
: 드래그할때 보여질 위젯

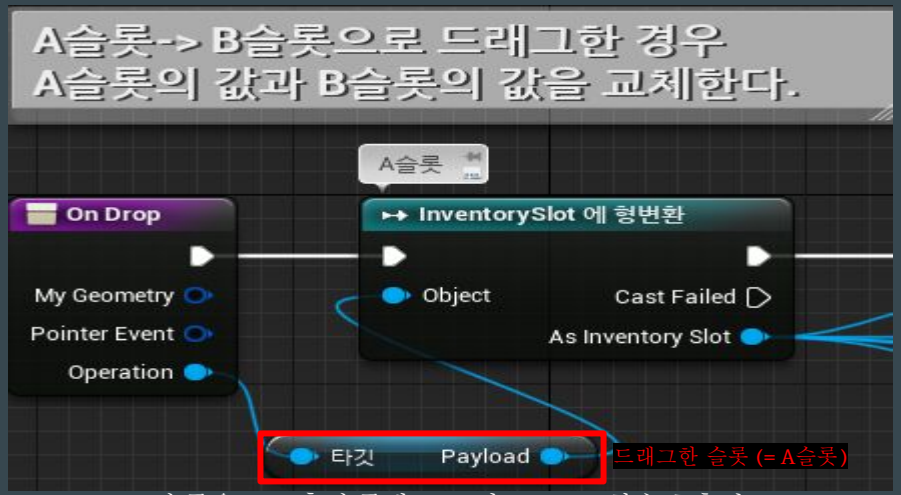


[슬롯 클릭후 드래그]



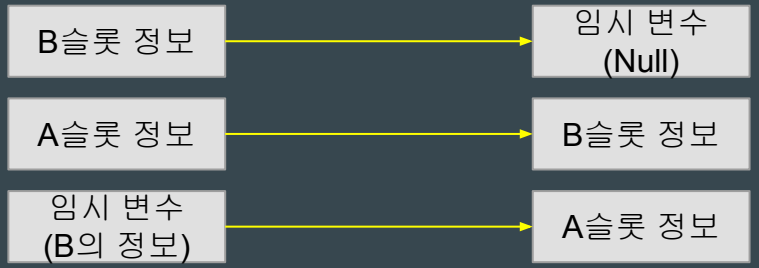
[On Drag Detected 함수호출됨]

3-5 Drag and Drop을 이용한 슬롯(인벤토리,퀵슬롯)간 교환 구현



[슬롯을 Drag후 슬롯에 Drop시 On Drop함수 호출됨]

슬롯의 인덱스값은 그대로 두고
슬롯의 정보구조체값(아이템정보,수량 등)만을 교환함



Swap(교환) 완료

4. 시연 영상

https://youtu.be/oGrgirYspDE

유튜브링크

감사합니다