프로젝트 TDShooting

남정웅

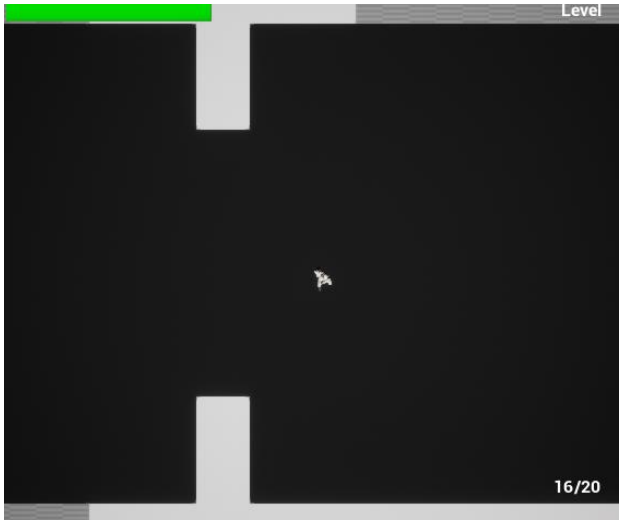
다운로드(GitHub)주소 : https://github.com/hehza90/-Unreal-TDShooting.git

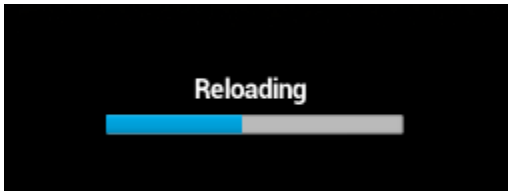
다른 언리얼프로젝트WaveDefense (GitHub)주소 :
https://github.com/hehza90/-Unreal-WaveDefense.git

1. 게임 소개
2. C++와 Blueprint의 사용처
3. BP_Room 제어
4. UInterface 활용
5. UI와 최적화

1. 게임 소개

1) 조작



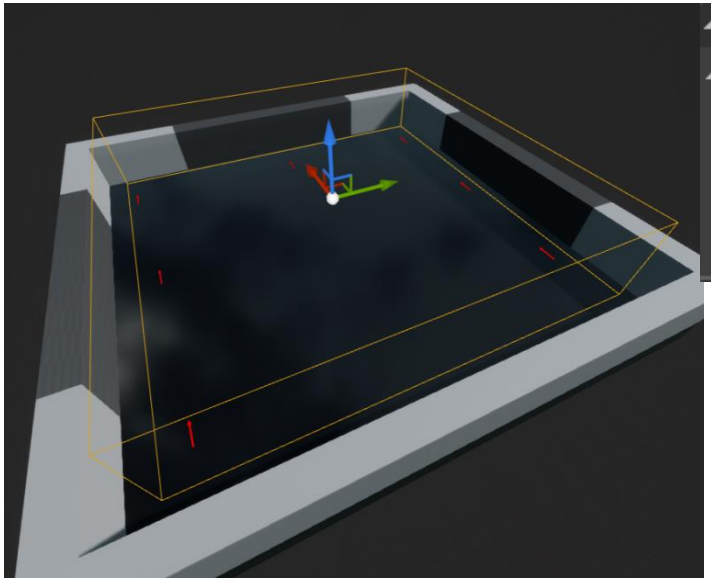


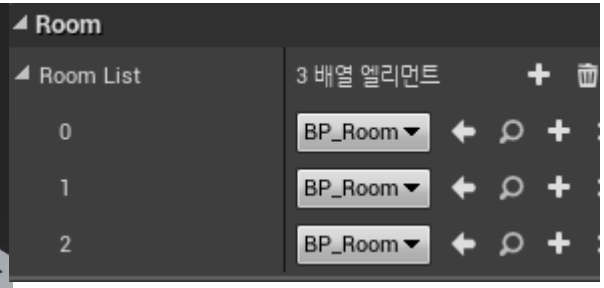
(내려다 보는 시점의 슈팅게임으로 WASD로 이동, 마우스로 조준 및 발사 입니다.)

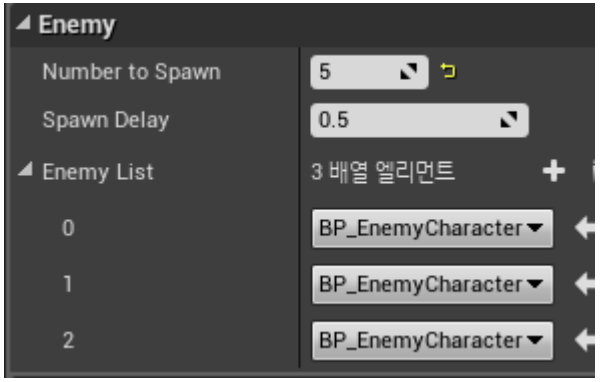
탄창을 한발이라도 소비 했을 때 R키를 누르거나 탄창을 모두 소비 했을 때 자동으로 재장전을 시작합니다. 재장전 할 때 아래에 게이지가 나타납니다
(오버워치의 재장전을 생각하면 되겠습니다)

2) 진행

BP_Room의 Trigger(BoxCollision)에 플레이어가 다가가면 BP_TDShootingGameModeBase에서 다음 레벨을 시작하여 붉은 화살표가 있는 위치 중 랜덤하게 적이 하나씩 등장합니다. (현재 등장횟수는 5회로 설정 했습니다)







BP_TDShootingGameModeBase 에서 여러게의 다른 BP_Room의 자손을 할당하여 랜덤으로 방을 생성 할 수 있습니다.(코드에서Tarray<TSubclassOf<ARoom>>) 나타나는 적의 종류도 마찬가지 입니다.

모든 적을 처치하면 3개의 검은색 벽 중 하나가 사라지면서 다음 방으로 가는 문이 열립니다. 방을 넘어 갈수록 적은 더 강해 집니다.

2. C++와 Blueprint의 사용처

C++	Blueprint
프로그래머가 당장 결정 할 수 없는 미션 완료나 게임 클리어 등의 세세한 부분을 제외한 거시적인 부분을 구현 할 때	미션 완료 되었을 때 특정 문이 열린다거나 아이템이 생기는 등 언제든지 바뀔 수 있는 세세한 부분
수학적인 공식이나 해당 프로젝트만을 위한 기능을 구현 할 때	UMG나 Behaviour Tree등 언리얼 엔진의 기능과 간편하게 연동이 가능한 부분
한 종류의 액터의 공통적인 행동을 정의 할 때	게임에 있는 단 하나의 액터의 행동만 정의 할 때
캐릭터 이동, 사운드나 이펙트에 대한 대략적인 제어만 할 때	캐릭터 이동, 사운드나 이펙트 등에 대한 에 대한 많은 옵션을 제어 할 때
성능을 최대한 올리고 싶을 때	프로그래밍에 익숙하지 않은 인원이 많을 때

3. BP_Room 제어

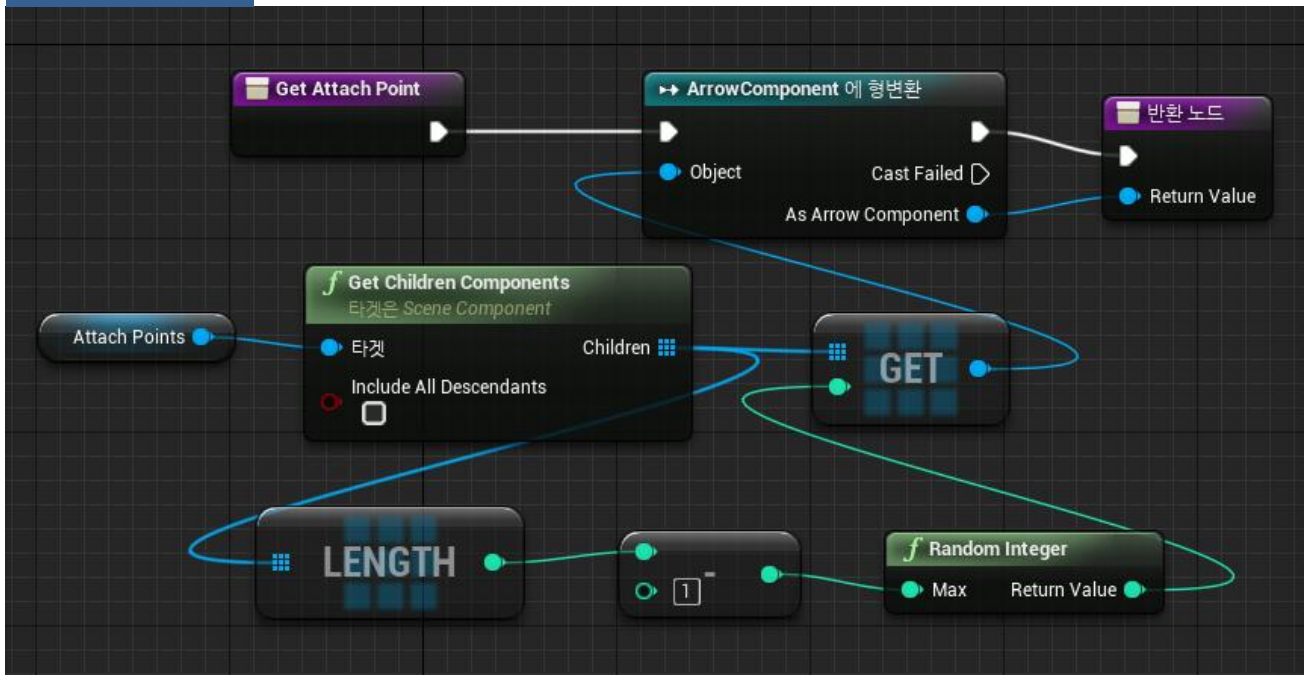
블루프린트 에디터에서 작업한 컴포넌트들을 제어 하기 위해선 시각적인 작업이 동반되고 코딩 중에 정확한 동작을 예측하기 힘듭니다.

Room.h

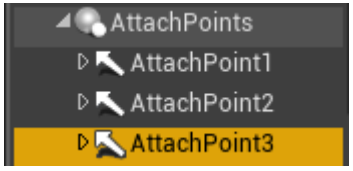
```
//다음 Room이 생성될 위치 받아오기(이하 블루프린트에서 작성)
UFUNCTION(BlueprintImplementableEvent, Category = "Room")
    UArrowComponent* GetAttachPoint();
```

다음 BP_Room이 생성될 위치들 중 하나를 랜덤으로 받아오는 함수

BP_Room



이러한 함수들을 코드에서 BlueprintImplementableEvent 정의만 한 후 블루프린트 에디터에서 추가한 (코딩 중에는 예측하기 힘든) ArrowComponent를 받아 오기 위해 블루프린트에서 함수를 재정의 했습니다.



블루프린트 에디터에서 AttachPoint들을 추가하거나 뷰포트를 통해 의도한 데로 Transform를 코드보다 직관적으로 편집 할 수 있습니다

4. UInterface 활용

새 C++클래스에서 UserInterfaceSettings 부모로 하는 Damageable 인터페이스를 만듭니다

Damageable.h

```
UFUNCTION(BlueprintNativeEvent, Catego
    void Damaged(float Amount);
```

BaseCharacter.h

```
void ABaseCharacter::Damaged_Implementation(float Amount)
{
    CalculateHealth(-Amount);
}
```

Obstacle.h

```
void AObstacle::Damaged_Implementation(float Amount)
{
    CalculateDurability();
}
```

Damaged함수를 class에 따라 (BaseCharacter, Obstacle 외 다른 추가된 class) 각각 다른 동작을 정의 할 수 있습니다.

BP_Projectile이 다른 오브젝트와 충돌 할 때 실행되는 함수

Projectile.h

```
void AProjectile::OnHit_Implementation(UPrimitiveComponent* Overla
{
    //데미지를 받을 수 있는 대상인지 확인
    IDamageable* DamageableActor = Cast<IDamageable>(OtherActor);
    if (DamageableActor != nullptr) {
        DamageableActor->Execute_Damaged(OtherActor, Damage);
    }
    Destroy();
}
```

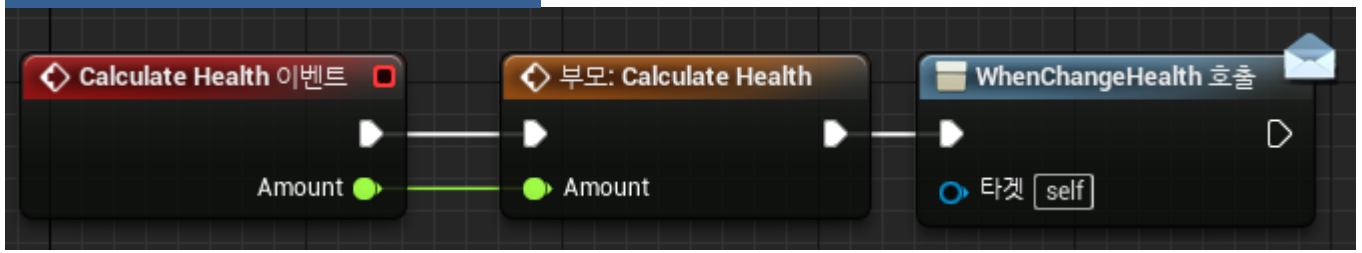
class Projectile에서 Damageable 인터페이스를 상속한 class라면 어떤 class이든지 Damaged 함수를 호출 할 수 있습니다.
인터페이스 함수는 호출하는 방법이 다르므로 유의합니다. (Execute_Damaged(...))

5. UI와 최적화

HP Bar의 경우 (현재체력/최대체력)
퍼센트를 구해야 합니다.

여기서 연산이 많이 추가 되어 매 프래임 실행 된다면 성능의 저하가 일어 날 수 있으므로 HP가 바뀔 때만 HP Bar의 정보를 갱신 하도록 만듭니다.

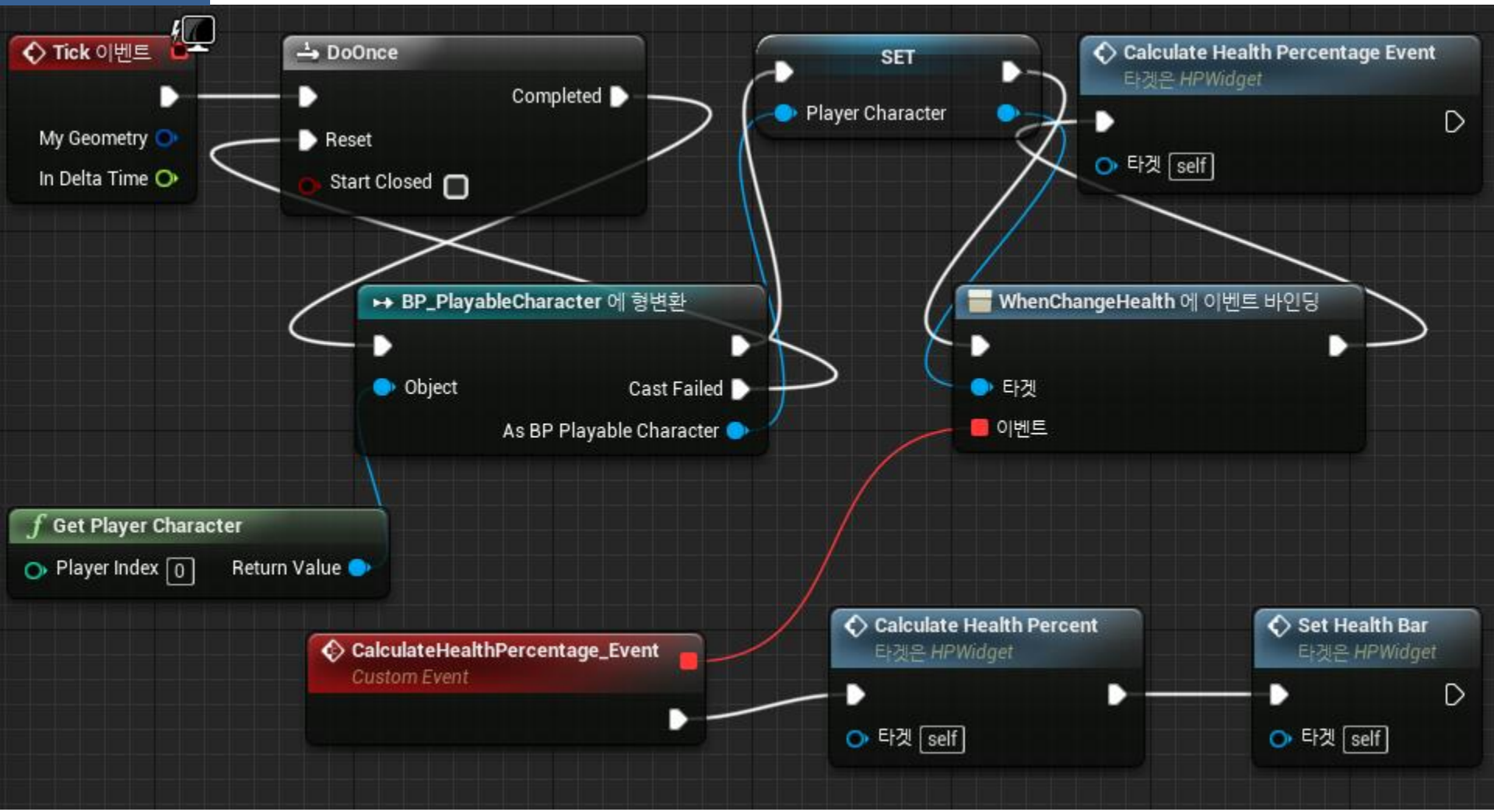
BP_PlayeableCharacter



WhenChangeHealth 이벤트 디스패처를 생성하고 HP가 바뀔 때마다 실행시켜 줍니다.
나중에 WhenChangeHealth에 HP의 퍼센테이지를 구하고 HP Bar를 갱신하는 함수를 등록 할 것입니다.

UI에 정보를 반영 할 때 값을 읽어 오는 것 외에도 연산이 필요 하다면(여기서는 HP의 Percentage를 구할 때의 Divide연산) 위와 같이 이벤트 디스패쳐를 이용해 CallBack 으로 반영하여 값이 변할 때만(값이 변하지 않을 때는 매 프래임마다 연산을 하지 않게) 연산하게 합니다.

HPWidget



Calclulate Health Percent : HealthPercent = 현재체력/최대체력
Set Health Bar : HealthPercent의 값을 UI에 반영

1. WhenChangeHealth 에 이벤트 바인딩
2. Calculate Health Percent의 Divede 연산은 매프레임이 아닌 HP가 바뀔 때만 실행