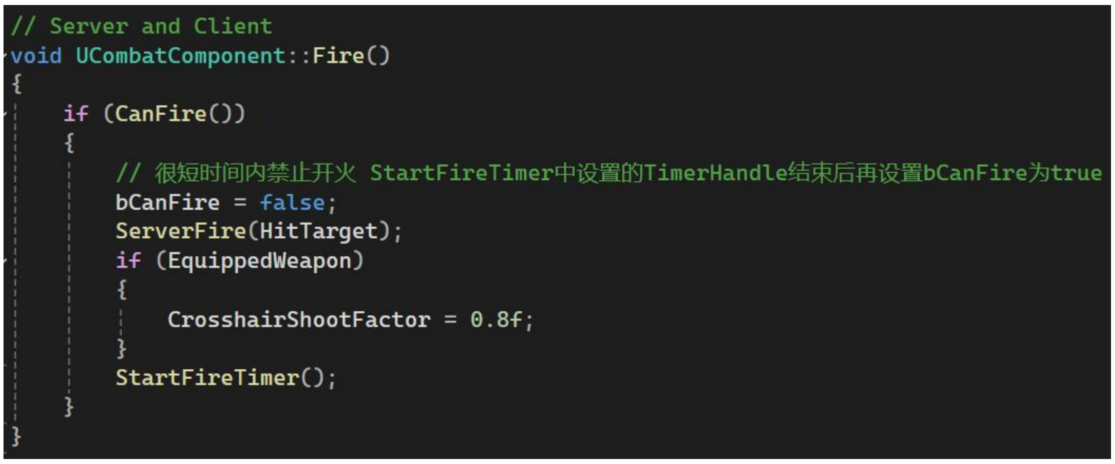
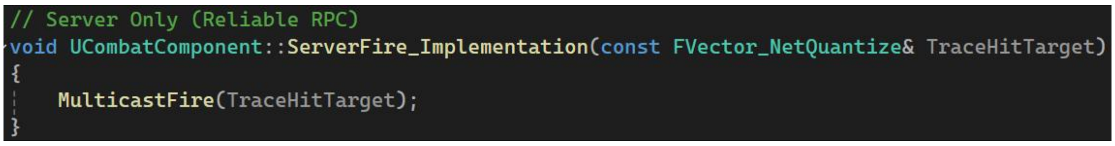
演示视频：

https://www.bilibili.com/video/BV1nDc8e5EMd/?spm_id_from=333.1387.homepage.video_card.click

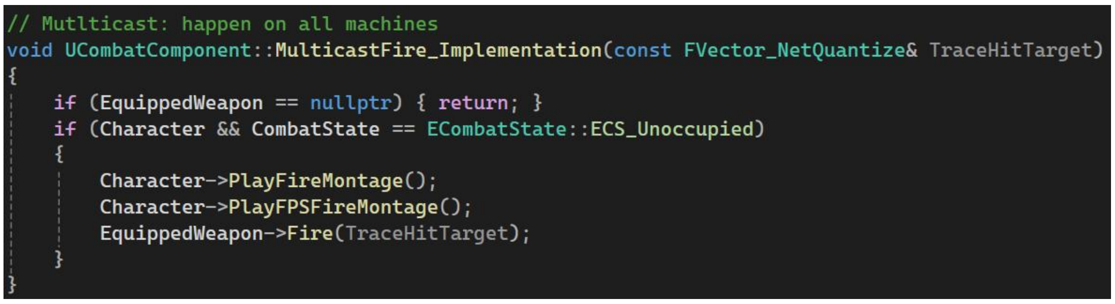
玩家发起开火，服务端创建子弹并广播：



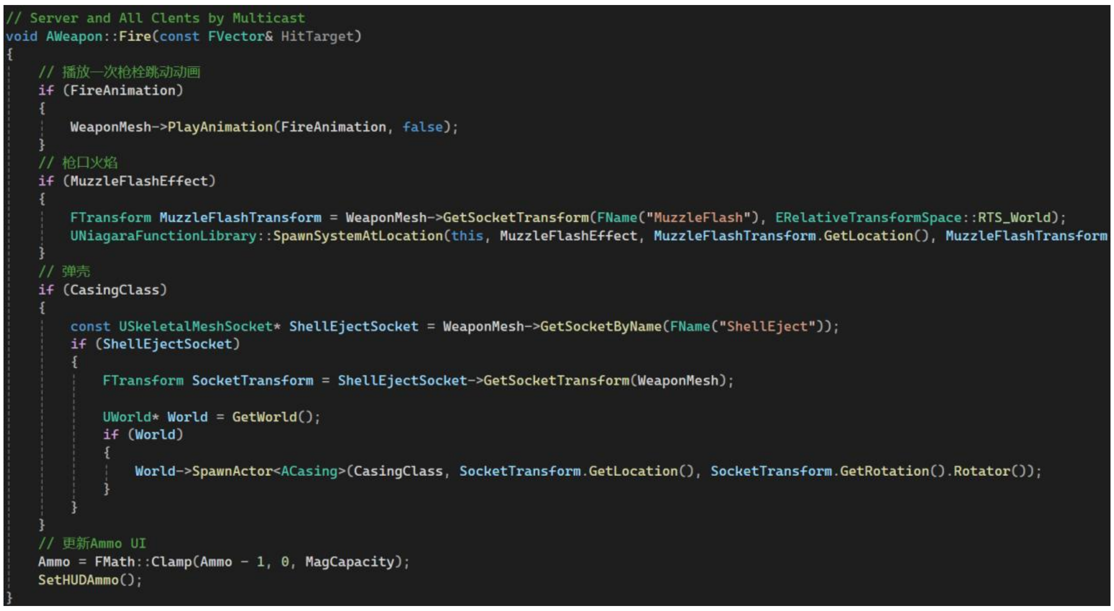
```cpp
// Server and Client
void UCombatComponent::Fire()
{
    if (CanFire())
    {
        // 很短时间内禁止开火 StartFireTimer中设置的TimerHandle结束后再设置bCanFire为true
        bCanFire = false;
        ServerFire(HitTarget);
        if (EquippedWeapon)
        {
            CrosshairShootFactor = 0.8f;
        }
        StartFireTimer();
    }
}
```



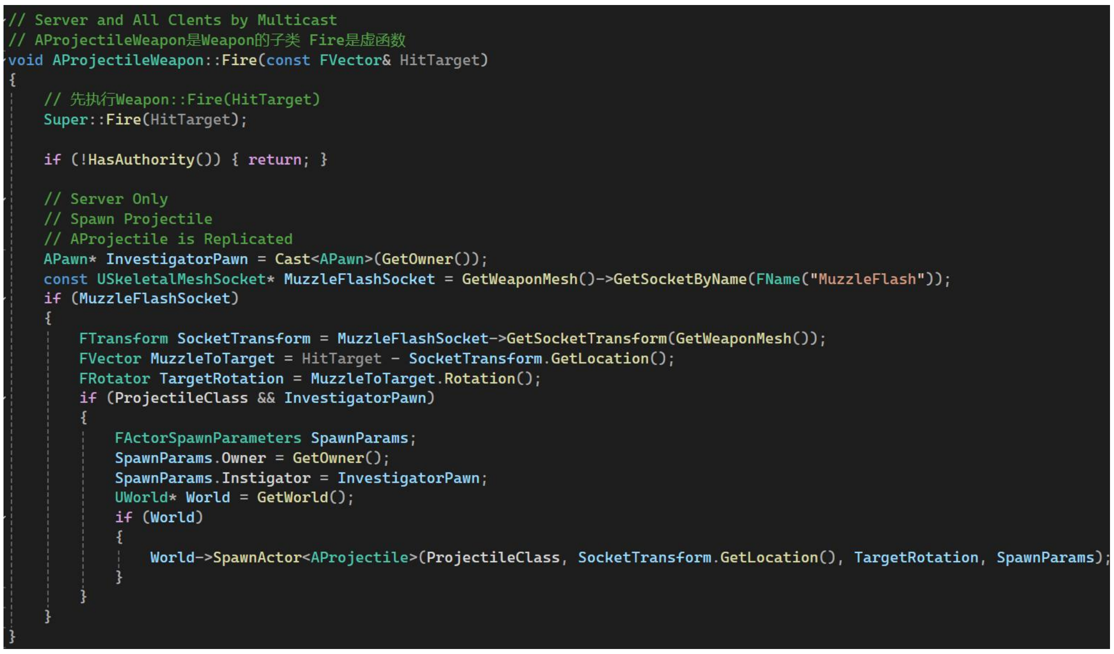
```cpp
// Server Only (Reliable RPC)
void UCombatComponent::ServerFire_Implementation(const FVector_NetQuantize& TraceHitTarget)
{
    MulticastFire(TraceHitTarget);
}
```



```cpp
// Mutlticast: happen on all machines
void UCombatComponent::MulticastFire_Implementation(const FVector_NetQuantize& TraceHitTarget)
{
    if (EquippedWeapon == nullptr) { return; }
    if (Character && CombatState == ECombatState::ECS_Unoccupied)
    {
        Character->PlayFireMontage();
        Character->PlayFPSFireMontage();
        EquippedWeapon->Fire(TraceHitTarget);
    }
}
```

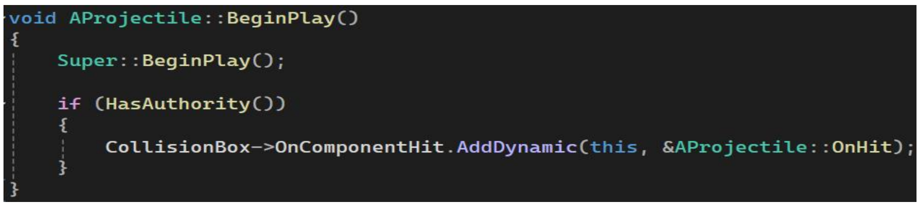


```cpp
// Server and All Clents by Multicast
void AWeapon::Fire(const FVector& HitTarget)
{
    // 播放一次枪栓跳动动画
    if (FireAnimation)
    {
        WeaponMesh->PlayAnimation(FireAnimation, false);
    }
    // 枪口火焰
    if (MuzzleFlashEffect)
    {
        FTransform MuzzleFlashTransform = WeaponMesh->GetSocketTransform(FName("MuzzleFlash"), ERelativeTransformSpace::RTS_World);
        UNiagaraFunctionLibrary::SpawnSystemAtLocation(this, MuzzleFlashEffect, MuzzleFlashTransform.GetLocation(), MuzzleFlashTransform
    }
    // 弹壳
    if (CasingClass)
    {
        const USkeletalMeshSocket* ShellEjectSocket = WeaponMesh->GetSocketByName(FName("ShellEject"));
        if (ShellEjectSocket)
        {
            FTransform SocketTransform = ShellEjectSocket->GetSocketTransform(WeaponMesh);

            UWorld* World = GetWorld();
            if (World)
            {
                World->SpawnActor<ACasing>(CasingClass, SocketTransform.GetLocation(), SocketTransform.GetRotation().Rotator());
            }
        }
    }
    // 更新Ammo UI
    Ammo = FMath::Clamp(Ammo - 1, 0, MagCapacity);
    SetHUDAmmo();
}
```

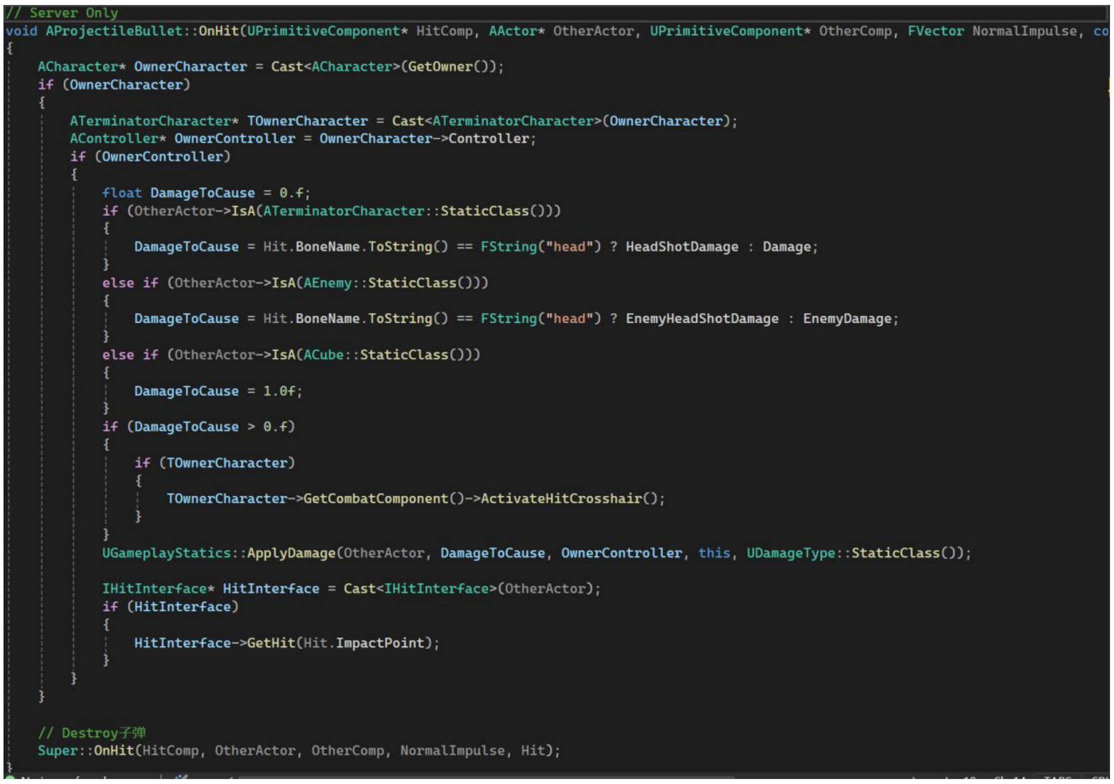


```cpp
// Server and All Clents by Multicast
// AProjectileWeapon是Weapon的子类 Fire是虚函数
void AProjectileWeapon::Fire(const FVector& HitTarget)
{
    // 先执行Weapon::Fire(HitTarget)
    Super::Fire(HitTarget);

    if (!HasAuthority()) { return; }

    // Server Only
    // Spawn Projectile
    // AProjectile is Replicated
    APawn* InvestigatorPawn = Cast<APawn>(GetOwner());
    const USkeletalMeshSocket* MuzzleFlashSocket = GetWeaponMesh()->GetSocketByName(FName("MuzzleFlash"));
    if (MuzzleFlashSocket)
    {
        FTransform SocketTransform = MuzzleFlashSocket->GetSocketTransform(GetWeaponMesh());
        FVector MuzzleToTarget = HitTarget - SocketTransform.GetLocation();
        FRotator TargetRotation = MuzzleToTarget.Rotation();
        if (ProjectileClass && InvestigatorPawn)
        {
            FActorSpawnParameters SpawnParams;
            SpawnParams.Owner = GetOwner();
            SpawnParams.Instigator = InvestigatorPawn;
            UWorld* World = GetWorld();
            if (World)
            {
                World->SpawnActor<AProjectile>(ProjectileClass, SocketTransform.GetLocation(), TargetRotation, SpawnParams);
            }
        }
    }
}
```

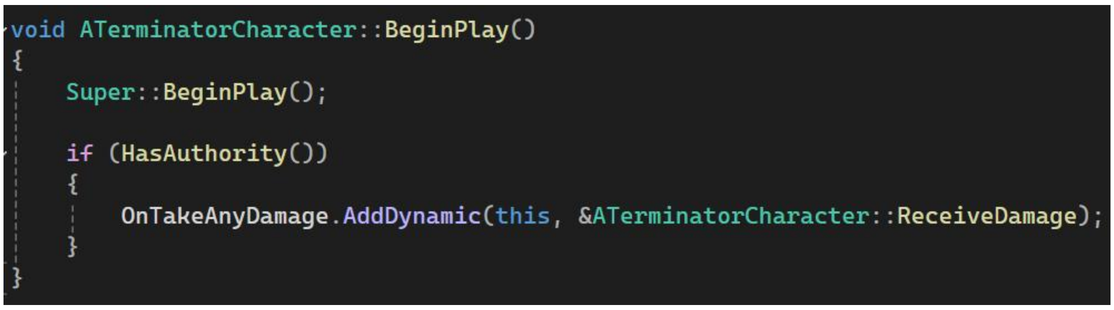
服务器实现子弹命中伤害逻辑，并同步角色血量变化：



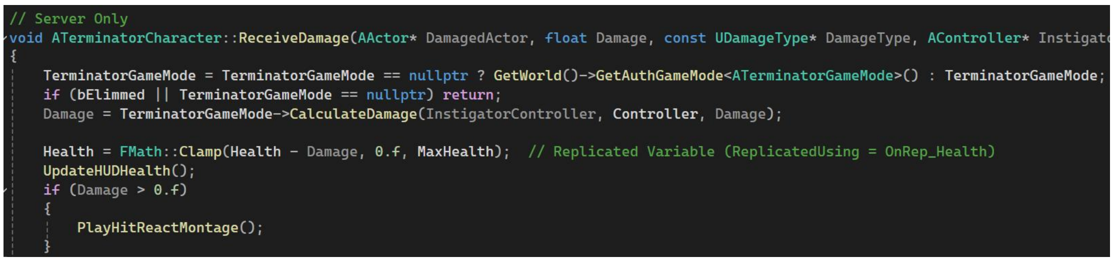
```cpp
void AProjectile::BeginPlay()
{
    Super::BeginPlay();

    if (HasAuthority())
    {
        CollisionBox->OnComponentHit.AddDynamic(this, &AProjectile::OnHit);
    }
}
```



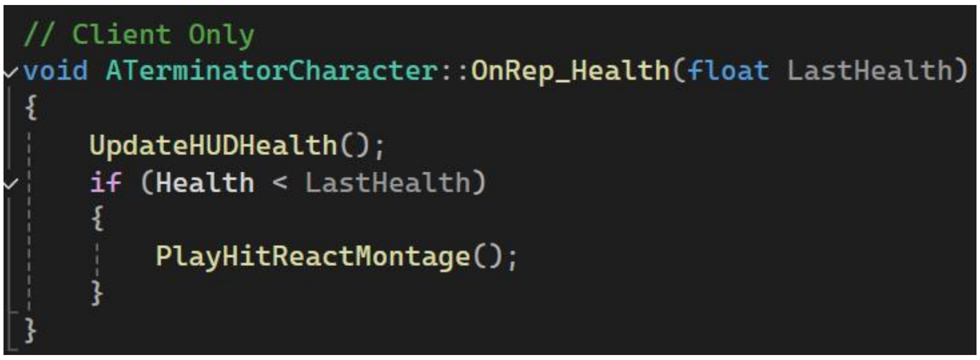
```cpp
// Server Only
void AProjectileBullet::OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, FVector NormalImpulse, co
{
    ACharacter* OwnerCharacter = Cast<ACharacter>(GetOwner());
    if (OwnerCharacter)
    {
        ATerminatorCharacter* TOwnerCharacter = Cast<ATerminatorCharacter>(OwnerCharacter);
        AController* OwnerController = OwnerCharacter->Controller;
        if (OwnerController)
        {
            float DamageToCause = 0.f;
            if (OtherActor->IsA(ATerminatorCharacter::StaticClass()))
            {
                DamageToCause = Hit.BoneName.ToString() == FString("head") ? HeadShotDamage : Damage;
            }
            else if (OtherActor->IsA(AEnemy::StaticClass()))
            {
                DamageToCause = Hit.BoneName.ToString() == FString("head") ? EnemyHeadShotDamage : EnemyDamage;
            }
            else if (OtherActor->IsA(ACube::StaticClass()))
            {
                DamageToCause = 1.0f;
            }
            if (DamageToCause > 0.f)
            {
                if (TOwnerCharacter)
                {
                    TOwnerCharacter->GetCombatComponent()->ActivateHitCrosshair();
                }
            }
            UGameplayStatics::ApplyDamage(OtherActor, DamageToCause, OwnerController, this, UDamageType::StaticClass());

            IHitInterface* HitInterface = Cast<IHitInterface>(OtherActor);
            if (HitInterface)
            {
                HitInterface->GetHit(Hit.ImpactPoint);
            }
        }
    }

    // Destroy子弹
    Super::OnHit(HitComp, OtherActor, OtherComp, NormalImpulse, Hit);
}
```



```cpp
void ATerminatorCharacter::BeginPlay()
{
    Super::BeginPlay();

    if (HasAuthority())
    {
        OnTakeAnyDamage.AddDynamic(this, &ATerminatorCharacter::ReceiveDamage);
    }
}
```

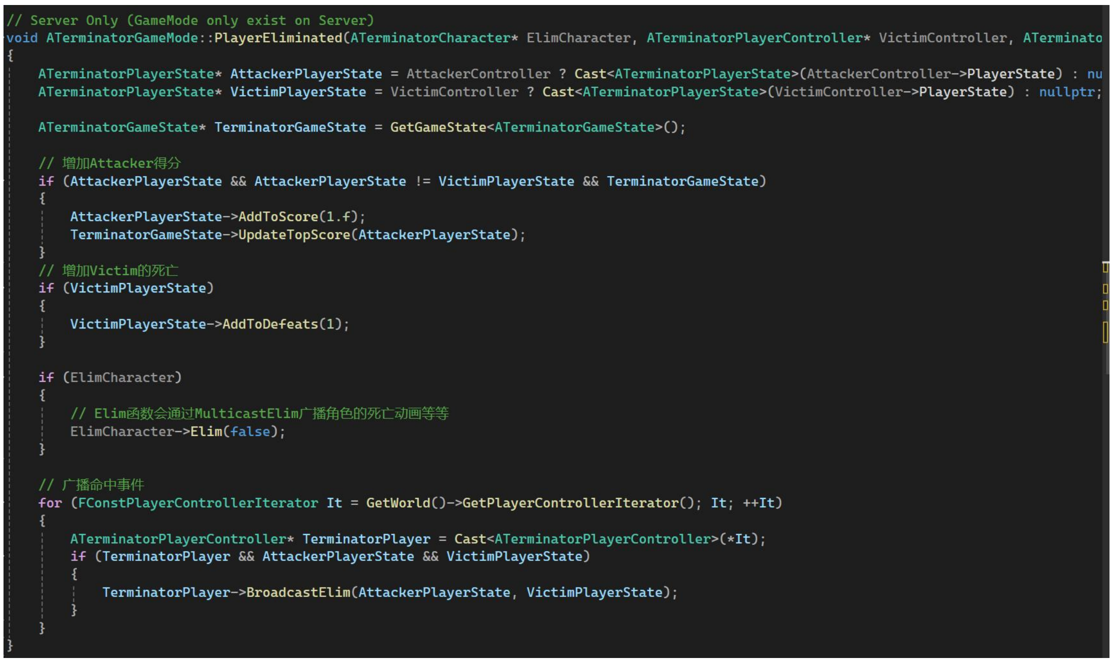


```cpp
// Server Only
void ATerminatorCharacter::ReceiveDamage(AActor* DamagedActor, float Damage, const UDamageType* DamageType, AController* Instigat
{
    TerminatorGameMode = TerminatorGameMode == nullptr ? GetWorld()->GetAuthGameMode<ATerminatorGameMode>() : TerminatorGameMode;
    if (bElimmed || TerminatorGameMode == nullptr) return;
    Damage = TerminatorGameMode->CalculateDamage(InstigatorController, Controller, Damage);

    Health = FMath::Clamp(Health - Damage, 0.f, MaxHealth);  // Replicated Variable (ReplicatedUsing = OnRep_Health)
    UpdateHUDHealth();
    if (Damage > 0.f)
    {
        PlayHitReactMontage();
    }
```

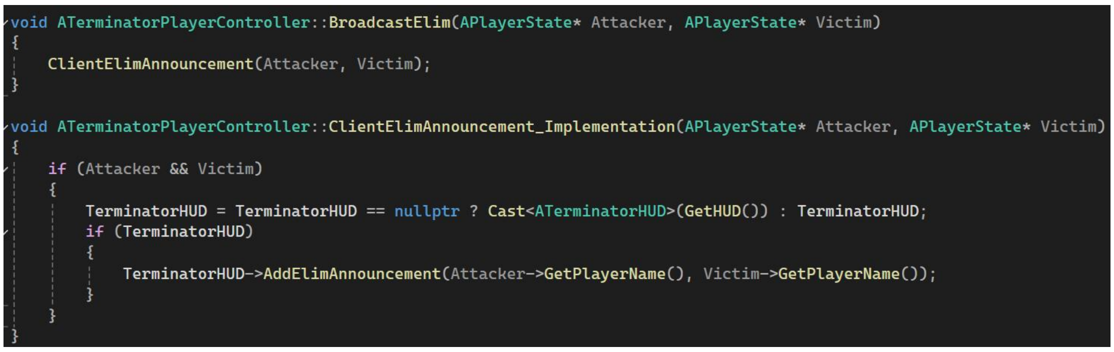


```cpp
// Client Only
void ATerminatorCharacter::OnRep_Health(float LastHealth)
{
    UpdateHUDHealth();
    if (Health < LastHealth)
    {
        PlayHitReactMontage();
    }
}
```

服务器实现子弹命中伤害逻辑，角色血量为 0 时广播命中事件：



```cpp
// Server Only (GameMode only exist on Server)
void ATerminatorGameMode::PlayerEliminated(ATerminatorCharacter* ElimCharacter, ATerminatorPlayerController* VictimController, ATerminato
{
    ATerminatorPlayerState* AttackerPlayerState = AttackerController ? Cast<ATerminatorPlayerState>(AttackerController->PlayerState) : nu
    ATerminatorPlayerState* VictimPlayerState = VictimController ? Cast<ATerminatorPlayerState>(VictimController->PlayerState) : nullptr;

    ATerminatorGameState* TerminatorGameState = GetGameState<ATerminatorGameState>();

    // 增加Attacker得分
    if (AttackerPlayerState && AttackerPlayerState != VictimPlayerState && TerminatorGameState)
    {
        AttackerPlayerState->AddToScore(1.f);
        TerminatorGameState->UpdateTopScore(AttackerPlayerState);
    }
    // 增加Victim的死亡
    if (VictimPlayerState)
    {
        VictimPlayerState->AddToDefeats(1);
    }

    if (ElimCharacter)
    {
        // Elim函数会通过MulticastElim广播角色的死亡动画等等
        ElimCharacter->Elim(false);
    }

    // 广播命中事件
    for (FConstPlayerControllerIterator It = GetWorld()->GetPlayerControllerIterator(); It; ++It)
    {
        ATerminatorPlayerController* TerminatorPlayer = Cast<ATerminatorPlayerController>(*It);
        if (TerminatorPlayer && AttackerPlayerState && VictimPlayerState)
        {
            TerminatorPlayer->BroadcastElim(AttackerPlayerState, VictimPlayerState);
        }
    }
}
```



```cpp
void ATerminatorPlayerController::BroadcastElim(APlayerState* Attacker, APlayerState* Victim)
{
    ClientElimAnnouncement(Attacker, Victim);
}

void ATerminatorPlayerController::ClientElimAnnouncement_Implementation(APlayerState* Attacker, APlayerState* Victim)
{
    if (Attacker && Victim)
    {
        TerminatorHUD = TerminatorHUD == nullptr ? Cast<ATerminatorHUD>(GetHUD()) : TerminatorHUD;
        if (TerminatorHUD)
        {
            TerminatorHUD->AddElimAnnouncement(Attacker->GetPlayerName(), Victim->GetPlayerName());
        }
    }
}
```