

언리얼 엔진 4 포트폴리오 기술 문서

도연진

소개



개발자

도연진

장르

액션 RPG

개발 환경

Unreal 4.26

개발 기간

12주 (24.1.1 ~ 24.3.1)

영상 링크

<https://youtu.be/2MPIEhzdaec>

블로그

<https://yeonjingameprogrammin.g.tistory.com/>

목차

Page

4	<u>Enemy AI Behavior Tree</u>	11	<u>Defend Parrying,Dodge</u>
5	<u>Enemy AI EQS</u>	12	<u>Feet IK</u>
6	<u>Weapon Structures</u>	13	<u>Parkour</u>
7	<u>Weapon Combo</u>	14	<u>Targeting</u>
8	<u>Bow</u>	15	<u>Boss Skill</u>
9	<u>Arrow</u>	16	<u>UI</u>
10	<u>Damage</u>	17	<u>개발 후기</u>

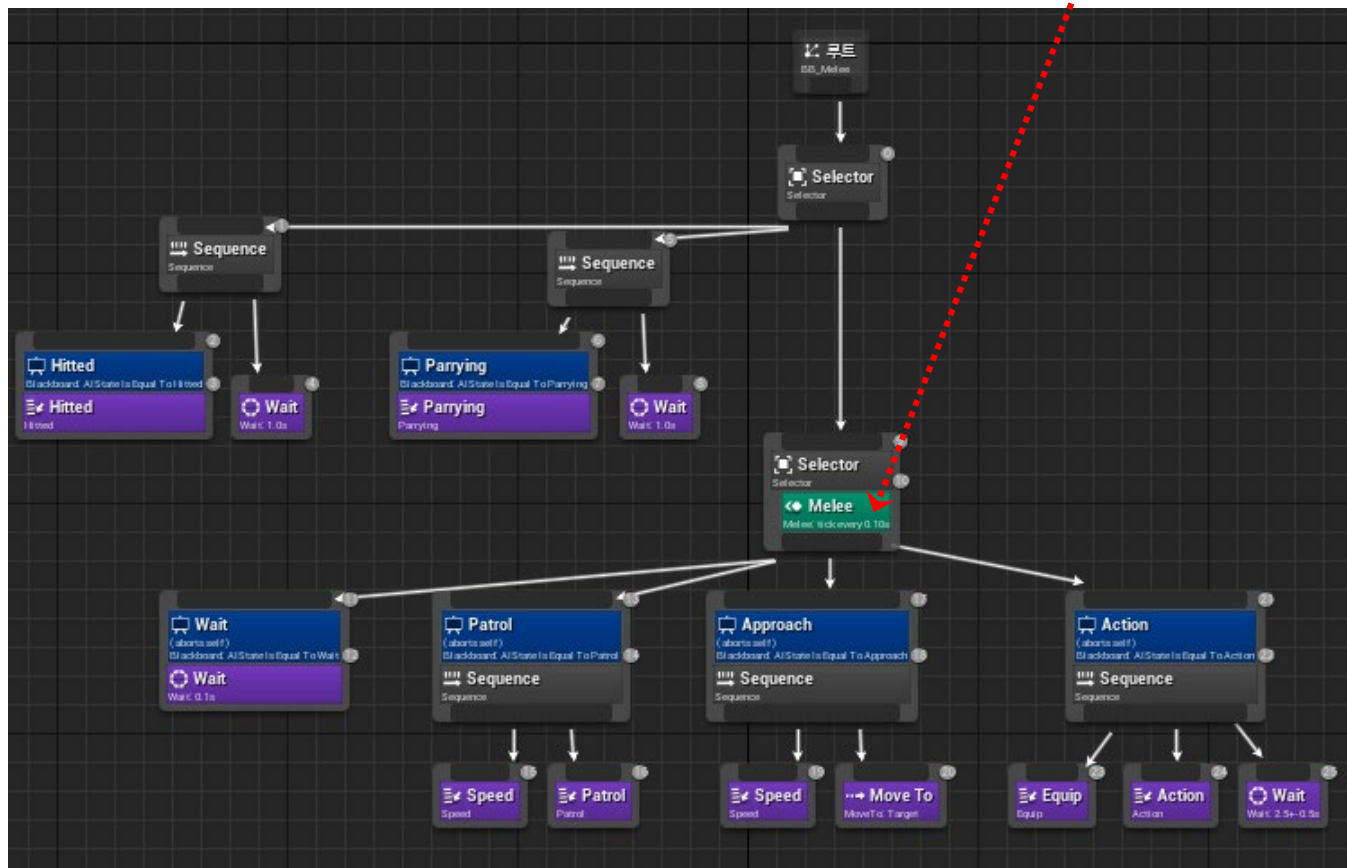
Enemy AI

Behavior Tree

- 일반 Enemy들은 Behavior Tree에 따라서 적을 인식하고 Enemy들의 상태에 맞는 행동을 일정 간격으로 수행
- Enemy는 Target으로 인식된 캐릭터와의 거리에 따라서 Wait, Patrol, Approach, Action(Attack) 등을 수행

Selector로 여러 행동 중 하나의 행동만 수행하도록 설정, Sequence로 행동을 위한 기능들을 연결

Service에서 **Enemy 상태**에 따른 Blackboard Key 설정



Enemy AI

EQS

- 궁수는 일반 몬스터와 같이 Behavior Tree로 행동
- Target이 일정거리 이하로 다가왔을 때 EQS 쿼리를 통해 순간 이동할 위치를 선택
- Scoring Equation을 Inverse Linear로 설정하여 Target과 Enemy, Target과 Item 을 내적인 값이 가장 작은 25%을 판별



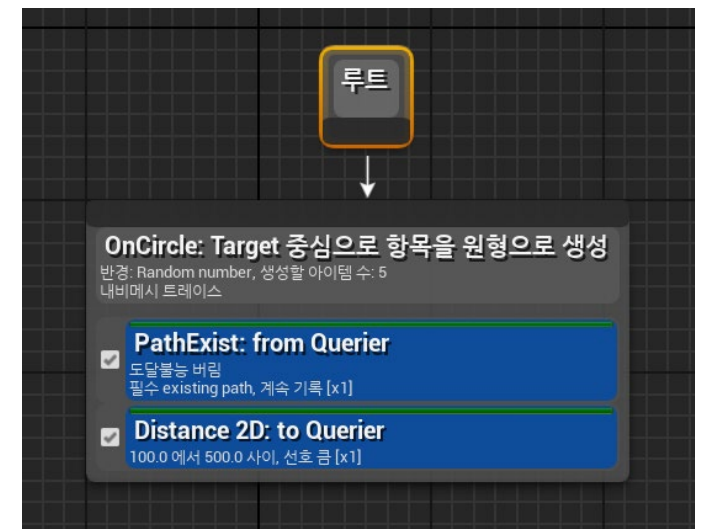
타겟의 후방으로 이동하도록
Vector(Target, Enemy)과
Vector(Target, Item)을 **내적**하여
판별된 상위 25% 아이템 중
랜덤으로 하나를 선택하여
해당 위치를 Blackboard에 넘겨줌

Enemy AI

EQS - Strafing

- EQS 를 사용하여 Strafing 구현
- 한 지점에만 계속 머무르지 않으며, Player 사이를 가로질러가지 않도록 양 옆 지점으로만 이동
- 옆걸음질 애니메이션 적용
- player가 일정거리 이상 가까이 오면 공격 시작

bool 변수를 주고 Strafing 중일때는 옆걸음질 Blendspace,
평소에는 장착중인 무기에 맞는 Blendspace 실행



Weapon Structures

Player

- BindAction() 으로 WeaponComponent::DoAction 호출

WeaponComponent

- BeginPlay()에서 WeaponAsset 배열에 **WeaponData** 를 넣어줌
- 현재 무기에 맞는 DoAction, SubAction, Equip 등의 함수 호출

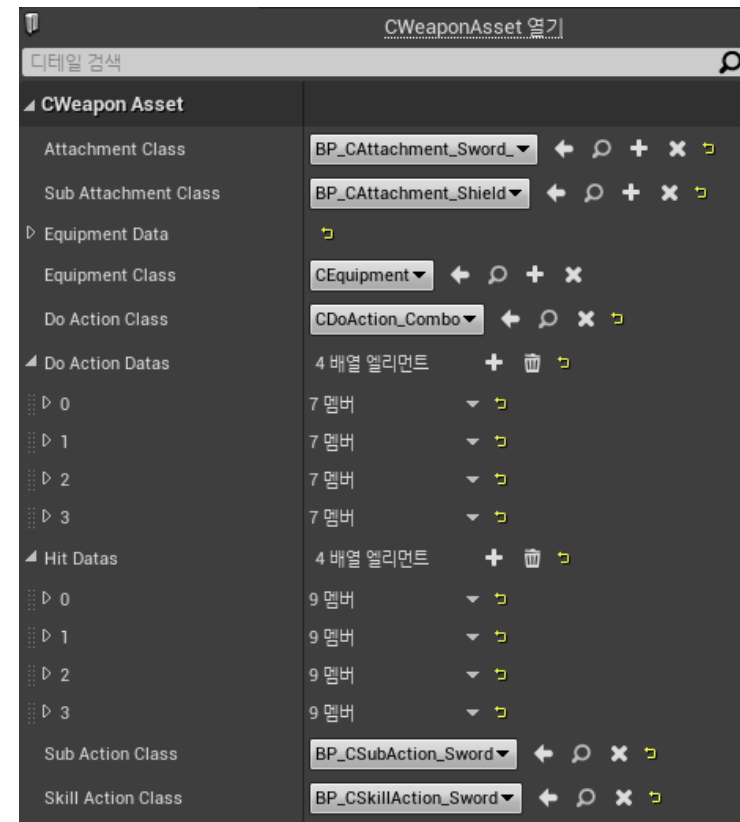
WeaponAsset

- WeaponAsset만 WeaponData에 접근 -> friend class로 열어줌
- 각 무기마다 OnEquip, OnAttachment, OnCollision 등의 **이벤트 연결**
- Asset 공유 문제 발생 방지 -> **WeaponData에 생성 객체 분리**

WeaponData

WeaponStructures

- FHitData, FEquipmentData, FDoActionData등의 **구조체 정의**
- Damage와 관련된 함수 정의



Sword Data Asset

Weapon

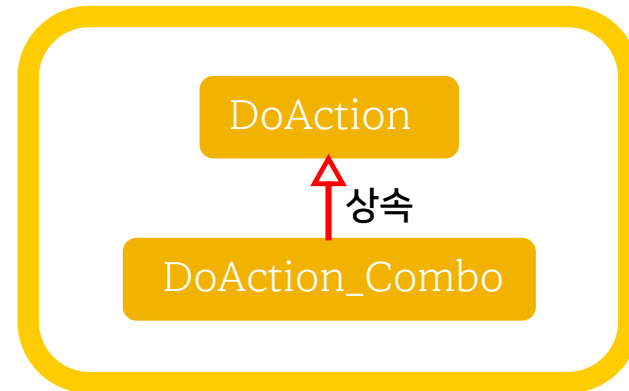
Combo

- **Anim_NotifyState**로 Combo구간에 시도 가능
- DoAction에서는 Attachment, Equipment, ActionData, HitData 생성
- DoAction_Combo 클래스에서는 Combo 가능 시 **ActionData의 index 증가**, Collision, Damage 전달, Enemy의 경우 랜덤으로 Combo실행

- 공격 후 hit된 캐릭터가 여러 개라면 카메라의 방향을 플레이어의 전방 방향에서 가장 가까운 캐릭터에게 고정

전방 방향에서 가장 가까운 캐릭터는 **내적**을 통해 구한다

$\cos 0^\circ = 1$ 이므로 플레이어의 정면에 가까울수록 1에 가까운 값이 나온다



```
//공격했을때 공격 방향으로 카메라 잡아준다
float angle = -2.0f;
ACharacter* candidate = nullptr;

for (ACharacter* hit: Hitted)
{
    FVector direction = hit->GetActorLocation() - OwnerCharacter->GetActorLocation();
    direction = direction.GetSafeNormal2D();

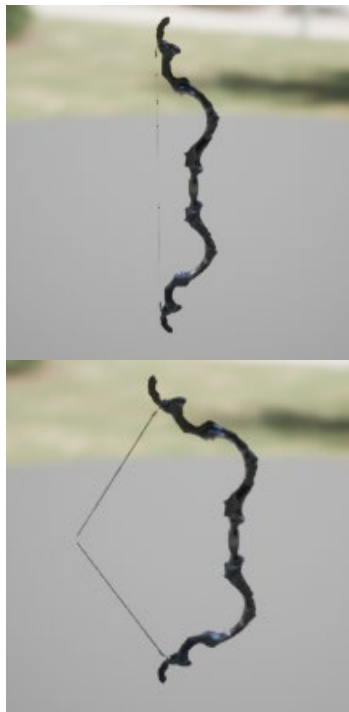
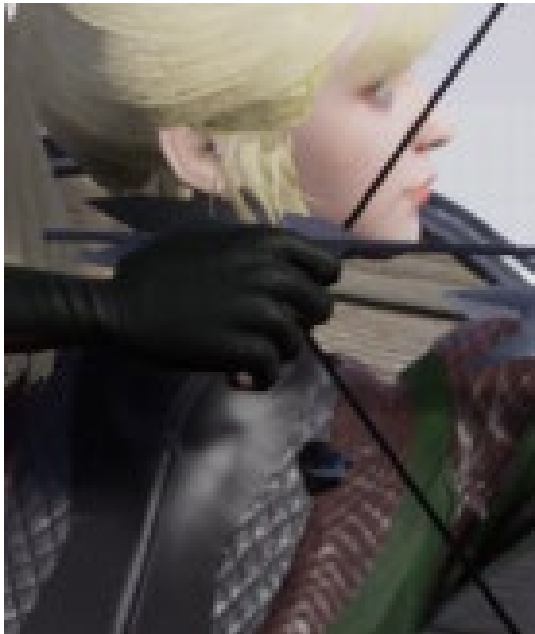
    //카메라와 전방방향 가장 가까운걸 찾는다
    FVector forward = FQuat(R: OwnerCharacter->GetControlRotation()).GetForwardVector();

    float dot = FVector::DotProduct(A: direction, B: forward);
    if(dot >= angle)
    {
        angle = dot;
        candidate = hit;
    } //가장 1에 가까운값이 나온다 = 정면
}
```


Weapon

Bow

- 활의 힘은 BlendSpace를 사용해 구부러지도록 구현
- 활 줄은 PoseableMesh->SetBoneLocationByName()함수를 사용해 캐릭터의 손에 붙도록 구현
- Aim시 zoom-in되어 조준이 편하도록 조정



```
//만들어놓은 Curve_Aim 실행
FOnTimelineVector timeline;
timeline.BindUFunction(InObject: this, InFunctionName: "OnAiming");

Timeline.AddInterpVector(Curve, timeline);
//커브를 0초~20초 구간으로 만들어놨다 -> 200배 빠르게 재생하면 0.1초 만에 재생되게 된다
Timeline.SetPlayRate(AimingSpeed);

ACAttachment_Bow* bow = Cast<ACAttachment_Bow>(InAttachment);

if (!!bow)
    Bend = bow->GetBend();
```

▲ Aim 할 때 만들어놓은 Curve에 따라서 zoom-in되도록 구현

OnAiming(FVector Output) 함수 ▶

```
Camera->FieldOfView = Output.X;

if (!!Bend)
    *Bend = Output.Y;
```

Weapon

Arrow

- 화살은 Projectile로 만들어졌고, Gravity를 없애고 원하는 곳으로 날아가도록 설정
- 화살은 Crosshair의 정중앙으로 날아가도록 설정

start : 카메라 위치
end : start + 전방 방향 * 거리

start ~ end 까지 LineTrace 수행

· hit된 물체가 있을 경우
spawnDirection
: hit된 물체 - 화살 spawn 위치

· hit된 물체가 없을 경우
spawnDirection = end - 화살 spawn 위치

```
if(!player)
{
    //Crosshair안에 화살을 날아가도록
    FVector start
        = UGameplayStatics::GetPlayerCameraManager(OwnerCharacter->GetWorld(), PlayerIndex:0)->K2_GetActorLocation();
    FVector end = start
        + UGameplayStatics::GetPlayerCameraManager(OwnerCharacter->GetWorld(), PlayerIndex:0)->GetActorForwardVector()
        * ScaleForwardVector;

    TArray<AActor*> ignores;
    ignores.Add(OwnerCharacter);

    FHitResult hitResult;

    UKismetSystemLibrary::LineTraceSingle(OwnerCharacter->GetWorld(), start, end,
        TraceChannel:ETraceTypeQuery::TraceTypeQuery2, bTraceComplex:false, ignores,
        EDrawDebugTrace::None, [&]hitResult, bIgnoreSelf:true);

    if (hitResult.bBlockingHit == true)
    {
        end = hitResult.ImpactPoint;
    }

    spawnLocation = OwnerCharacter->GetMesh()->GetSocketLocation("Hand_Bow_Right_Arrow");

    FVector spawnDirection = end - spawnLocation;
    spawnRotation = UKismetMathLibrary::MakeRotFromX(spawnDirection);

    FVector forward = UKismetMathLibrary::GetForwardVector(spawnRotation);

    arrow->Shoot(forward);
}
```

Weapon

Damage

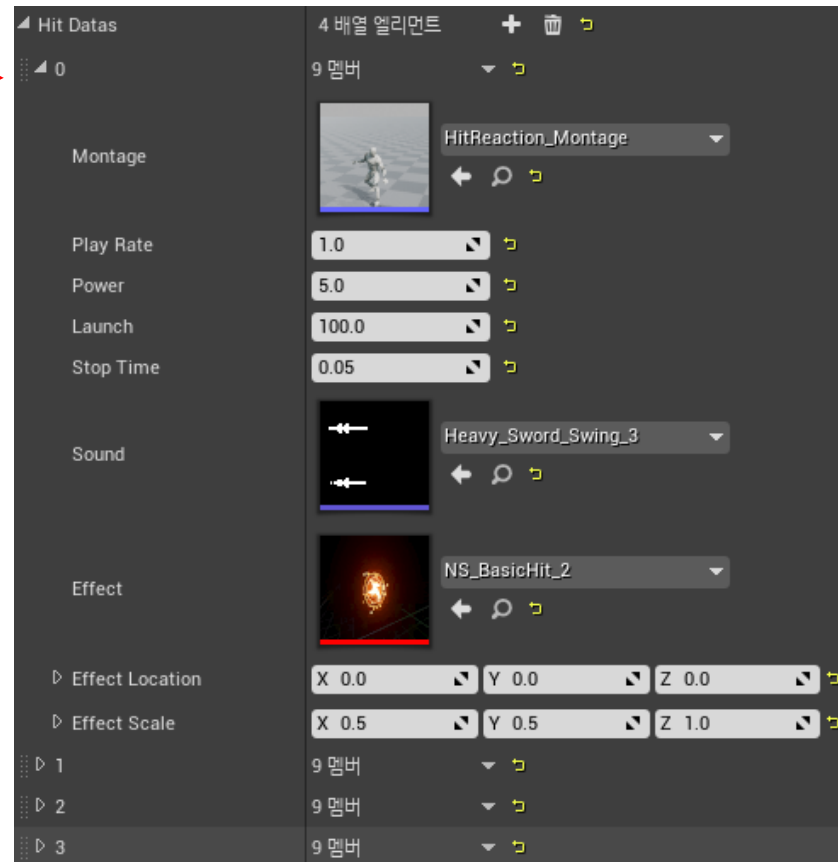
Data Asset에 들어가있는 HitData

- Damage관리는 WeaponStructures 클래스에서 구조체로 만들어진 **FHitData**를 통해 관리
- FHitData에는 변수로 Montage, PlayRate, Power, Launch, StopTime, Sound, Effect 등이 들어가있고 Editor에서 수정이 가능하도록 EditAnywhere로 직렬화
- 변수들의 설정과 관련된 함수도 FHitData에 들어가있음

```
public:
    void SendDamage(class ACharacter* InAttacker, AActor* InAttackCauser, class ACharacter* InOther);
    void PlayMontage(class ACharacter* InOwner);
    void PlayHitStop(UWorld* InWorld);
    void PlaySoundWave(class ACharacter* InOwner);
    void PlayEffect(UWorld* InWorld, const FVector& InLocation);
    void PlayEffect(UWorld* InWorld, const FVector& InLocation, const FRotator& InRotation);
```

```
void FHitData::SendDamage(ACharacter* InAttacker, AActor* InAttackCauser, ACharacter* InOther)
{
    FActionDamageEvent e;
    e.HitData = this;

    InOther->TakeDamage(Power, e, EventInstigator: InAttacker->GetController(), InAttackCauser);
}
```



SendDamage 함수에서는 매개변수로 공격한 캐릭터, 공격한 액터(무기), 맞은 캐릭터를 받고, 해당 매개변수들을 사용하여 **TakeDamage** 함수를 통해 데미지를 전달한다

Defend

Parrying, Dodge

- player는 tab키로 방어 모드로 전환 가능
- **Anim_Notify_State**로 만들어진 Parrying 구간에 방어키 입력 시 parrying성공
- parrying 구간에 방어 + 회피 시 회피기 발동
- 회피기는 현재 장착된 무기에 맞는 공격 동작 실행 -> 무기 미장착시 회피기 불가능
- parrying 성공 + 처형키 입력시 처형 가능

행	Type	Montage	Play Rate	
1	1	Fist	AnimMontage'/Game/Character/Montages/Fist/Fist_AvoidAttack_Monta	1.300000
2	2	Sword	AnimMontage'/Game/Character/Montages/Sword/Sword_AvoidAttack_N	1.000000
3	3	Hammer	AnimMontage'/Game/Character/Montages/Hammer/Hammer_AvoidAtta	1.000000

DT_AvoidAttack

```
TArray<FAvoidAttackData*> avoidAttackDatas;  
AvoidAttackDataTable->GetAllRows<FAvoidAttackData>(ContextString: "", [&]avoidAttackDatas);  
  
for (int32 i = 0; i < (int32)EWeaponType::Max; i++)  
{  
    for (FAvoidAttackData* data : avoidAttackDatas)  
    {  
        if ((EWeaponType)i == data->Type)  
        {  
            AvoidAttackDatas[i] = data;  
  
            continue;  
        }  
    }  
}
```

StateComponent에서 캐릭터들의 상태가 관리되고
특정 상태에 재생될 Montages들은
MontagesComponent에서 관리한다

MontagesComponent에서 Montage를 재생하면
DataTable에 들어가있는 애니메이션이 재생된다

회피기 또한 DataTable로 만들어져 무기에 맞는
회피기가 자동으로 재생된다

IK

Feet IK

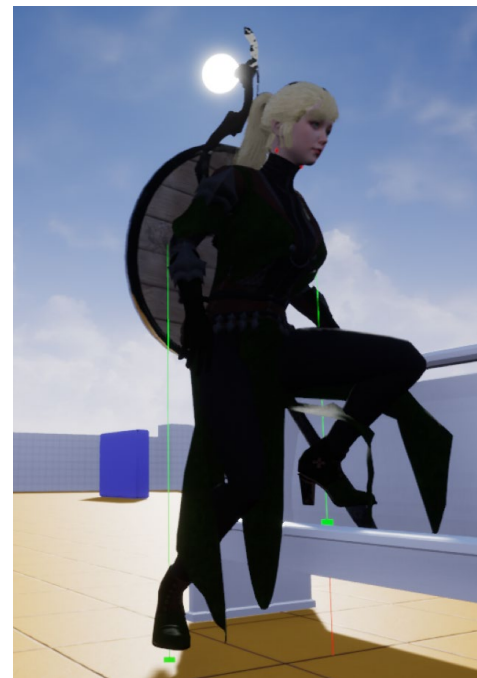
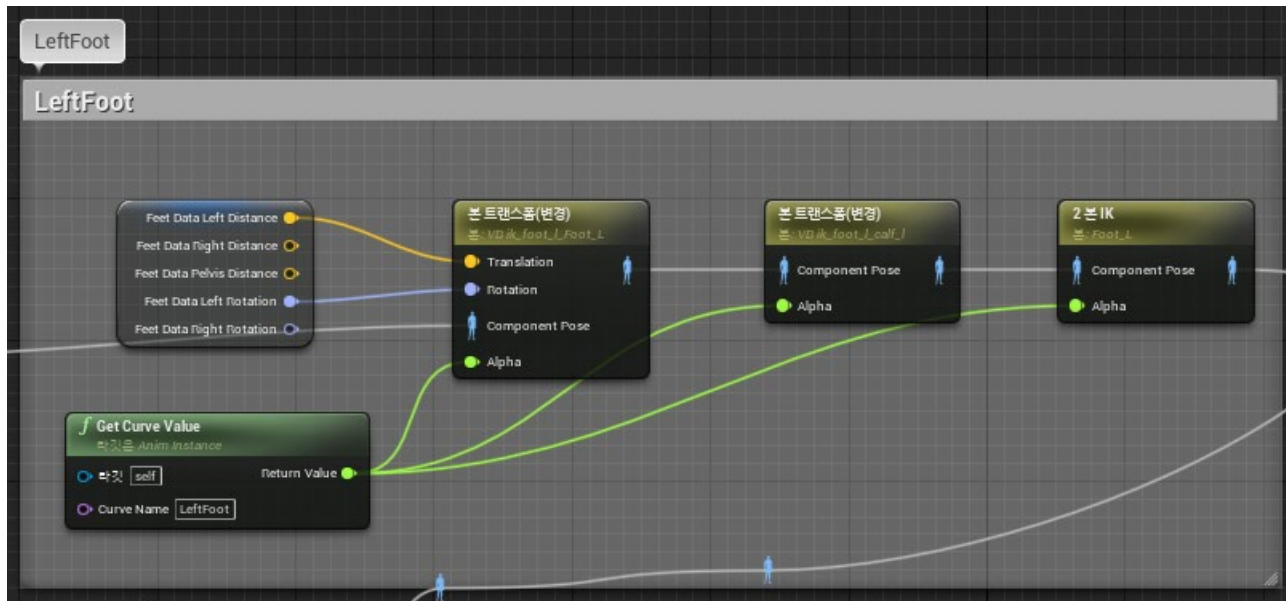
왼발 IK

- 애니메이션을 프로그래밍으로 수정하기 위해서 로컬 공간을 컴포넌트 공간으로 변경
- LineTrace를 사용해 발이 지면에 닿는 위치를 구한 뒤 자연스러움을 위해 **2 Bone IK** 적용
- 다리 위치가 변함에 따라 Pelvis까지 자연스럽게 조정
- IK 적용 후 다시 컴포넌트 공간에서 로컬 공간으로 변경

두 점 사이의 상대 좌표(x, y)를 받아 절대각을 반환하는 Atan2함수로 cos값을 알아내 **발 각도**를 지면에 맞게 조정한다

```
//발 각도
float roll = +UKismetMathLibrary::DegAtan2(hitResult.Normal.Y, x: hitResult.Normal.Z);
float pitch = -UKismetMathLibrary::DegAtan2(y: hitResult.Normal.X, x: hitResult.Normal.Z);

OutRotation = FRotator(pitch, InYaw: 0, roll);
```



Parkour

```
//데이터 테이블에서 데이터를 불러온다
TArray<FParkourData*> datas;
DataTable->GetAllRows<FParkourData>(ContextString: "", [&]datas);

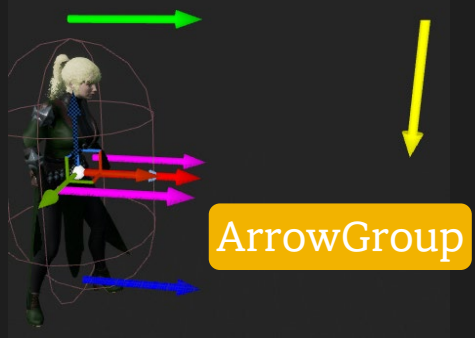
//파크루 타입에 맞게 데이터를 집어넣는 과정
for(int32 i = 0; i < (int32)EParkourType::Max; i++)
{
    TArray<FParkourData> temp;
    for(FParkourData* data: datas)
    {
        if (data->Type == (EParkourType)i)
            temp.Add(*data);
    }

    DataMap.Add((EParkourType)i, temp);
}

OwnerCharacter = Cast<ACharacter>(Src:GetOwner());
USceneComponent* arrow = CHelpers::GetComponent<USceneComponent>(OwnerCharacter, InName: "ArrowGroup");

TArray<USceneComponent*> components;
arrow->GetChildrenComponents(bIncludeAllDescendants: false, [&]components);

for (int32 i = 0; i < (int32)EParkourArrowType::Max; i++)
    Arrows[i] = Cast<UArrowComponent>(components[i]);
```



- center, left, right, ceil, land, floor 6개의 arrow를 만들고 이 arrow를 기반으로 **LineTrace**를 수행
- LineTrace의 HitResult를 체크하여 **파크루 수행 가능 여부**와 **장애물의 유형**을 구분
- ParkourData에 있는 조건에 맞는 Data를 불러와 파쿠르 수행

```
bool UCParkourComponent::Check_Obstacle()
{
    CheckNullResult(HitObstacle, false);

    //모서리가 아닐때를 체크(가운데, 양옆이 모두 hit되어야한다)
    bool b = true;
    b &= HitResults[(int32)EParkourArrowType::Center].bBlockingHit;
    b &= HitResults[(int32)EParkourArrowType::Left ].bBlockingHit;
    b &= HitResults[(int32)EParkourArrowType::Right ].bBlockingHit;
    CheckFalseResult(b, false);

    //3개의 normal 벡터가 같은지 판단
    FVector center = HitResults[(int32)EParkourArrowType::Center].Normal;
    FVector left  = HitResults[(int32)EParkourArrowType::Left ].Normal;
    FVector right = HitResults[(int32)EParkourArrowType::Right ].Normal;

    CheckFalseResult(center.Equals(left), false);
    CheckFalseResult(center.Equals(right), false);

    //전방방향(각도 체크)
    FVector start = HitResults[(int32)EParkourArrowType::Center].ImpactPoint;
    FVector end   = OwnerCharacter->GetActorLocation();
    float lookAt  = UKismetMathLibrary::FindLookAtRotation(start, Target:end).Yaw;

    FVector impactNormal = HitResults[(int32)EParkourArrowType::Center].ImpactNormal;
    float impactAt       = UKismetMathLibrary::MakeRotFromX(impactNormal).Yaw;

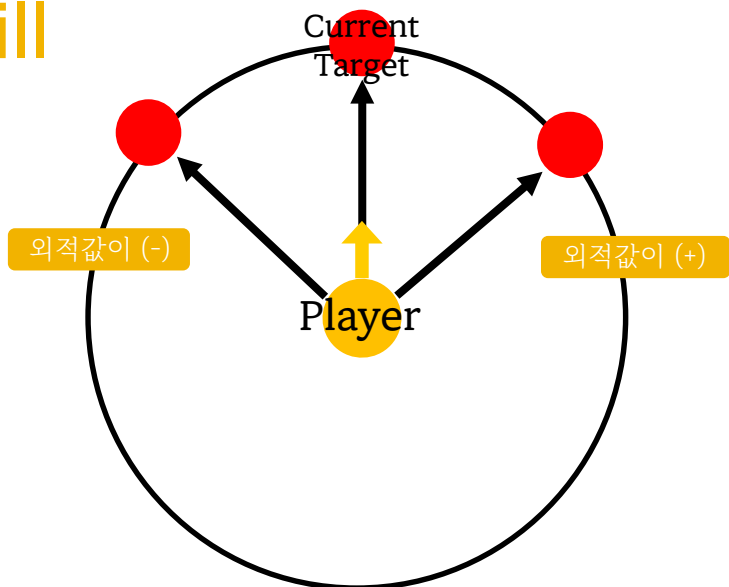
    float yaw = abs(_Xxx: abs(lookAt) - abs(impactAt));

    CheckFalseResult(yaw <= AvailableFrontAngle, false);

    return true;
}
```

Targeting

Skill



```
UKismetSystemLibrary::SphereTraceMultiByProfile(GetWorld(), start, end, Radius: TraceDistance,
    ProfileName: "Targeting", bTraceComplex: false, ignores, DebugType, [&]hitResults, bIgnoreSelf: true);

for (int32 i = 0; i <= hitResults.Num() - 1; i++)
{
    AActor* hit = hitResults[i].GetActor();

    ACharacter* hitCharacter = Cast<ACharacter>(hit);

    FVector direction = (hit->GetActorLocation() - OwnerCharacter->GetActorLocation()).GetSafeNormal2D(1e-04);
    FVector forward = UKismetMathLibrary::GetForwardVector(InRot: OwnerCharacter->GetControlRotation());
    FVector up = OwnerCharacter->GetActorUpVector();

    FVector cross = UKismetMathLibrary::Cross_VectorVector(A: forward, B: direction);
    float dot = UKismetMathLibrary::Dot_VectorVector(A: cross, B: up);

    map.Add(dot, hitCharacter);
}
```

forward 벡터와 direction 벡터를 외적하여 hitCharacter가 player보다 왼쪽에 있는지, 오른쪽에 있는지 구한다

이 외적된 값과 player의 up 벡터를 내적하여 map의 key 값에 넣어주고 최소 key를 가진 candidate가 가장 작은 각도 안에 있다는 뜻이므로 새로운 target이 된다

```
ACharacter* candidate = nullptr;
for (TTuple<float, ACharacter*> keys : map)
{
    key = keys.Key;

    if ((bInRight && key > 0) || (bInRight && key < 0))
    {
        float absKey = UKismetMathLibrary::Abs(key);

        if (absKey < minimum)
        {
            minimum = absKey;
            candidate = *map.Find(key);
        }
    }
}

ChangeTarget(candidate);
```

ChangeTarget_Right

ChangeTarget_Left

Boss

Skill

- Boss는 Patrol상태를 유지하다가 Target이 인식되면 공격 시작
- Boss는 총 5가지 공격을 정해진 확률에 따라 반복

불덩이가 떨어지는 Trail은 베지에 곡선으로 구현
PosA는 시작점(Boss의 머리 위치),
PosD는 불덩이가 도달할 목표지점,
PosB와 PosC는 그 사이의 랜덤값으로 잡아주었다



매개변수에 위의 초기화한 PosA~D가 들어간다

```
PosA = FVector(Start.X, Start.Y, InZ:200);
PosD = End;

PosB = PosA + FVector(
    DistanceFromStart * UKismetMathLibrary::RandomFloatInRange(Min:-1.0, Max:1.0) * Start.RightVector
    + DistanceFromStart * UKismetMathLibrary::RandomFloatInRange(Min:-1, Max:1.0) * Start.UpVector
    + DistanceFromStart * UKismetMathLibrary::RandomFloatInRange(Min:-1.0, Max:1) * Start.ForwardVector
);

PosC = PosD + FVector(
    DistanceFromEnd * UKismetMathLibrary::RandomFloatInRange(Min:-1.0, Max:1.0) * End.RightVector
    + DistanceFromEnd * UKismetMathLibrary::RandomFloatInRange(Min:-1.0, Max:1.0) * End.UpVector
    + DistanceFromEnd * UKismetMathLibrary::RandomFloatInRange(Min:-1, Max:1.0) * End.ForwardVector
);
```

```
float ACRandomObject::Trail(float a, float b, float c, float d)
{
    float t = CurrentTime / MaxTime;

    return    FMath::Pow(A:(1 - t), B:3) * a
    + FMath::Pow(A:(1 - t), B:2) * 3 * t * b
    + FMath::Pow(A:t, B:2) * 3 * (1 - t) * c
    + FMath::Pow(A:t, B:3) * d;
}
```


Boss

Skill – Hammer Jump

- player의 현재 이동 속도와 방향을 이용하여 일정 시간 이후 player의 위치를 예상
- 계산된 위치로

SuggestProjectileVelocity_CustomArc()함수를
사용하여 Launch

- player의 미래 위치를 예상하여 움직여 보스의 난이도 조절

```
FVector UCMovementComponent::GetFutureLocation(ACharacter* InCharacter, float InTime)
{
    //캐릭터의 현재 위치와 속도를 계산하여 InTime뒤의 캐릭터의 위치를 예상
    FVector velocity = InCharacter->GetVelocity();
    FVector location = InCharacter->GetActorLocation();

    FVector distance = velocity * FVector(InX:1, InY:1, InZ:0) * InTime;

    return distance + location; //InTime동안 움직일 거리 + 현재위치
}
```

```
//타겟의 위치로 점프하며 launch
FVector startPos = ai->GetActorLocation();
FVector targetPos = movement->GetFutureLocation(InCharacter: aiState->GetTarget(), ExpectTime);
FVector endPos = FVector(targetPos.X, targetPos.Y, InZ: targetPos.Z + 100);

FVector direction = endPos - startPos;

if(Debug)
    DrawDebugSphere(GetWorld(), endPos, Radius:100, Segments:12, FColor::Blue, bPersistentLines:false, LifeTime:1, DepthPriority:0,

UGameplayStatics::SuggestProjectileVelocity_CustomArc(GetWorld(), [&]Velocity, startPos, endPos);

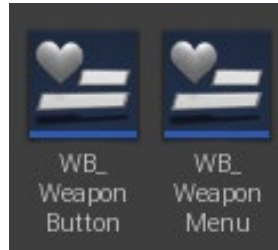
ai->SetActorRotation(direction.ToOrientationRotator());
ai->LaunchCharacter(Velocity, bXYOverride:true, bZOverride:true);
```



UI

Skill

- player 체력바에는 Hp, Mana가 있음
- Enemy 체력바는 player와 Enemy의 거리에 따라 크기가 다르게 표시됨
- Weapon선택 메뉴는 Button으로 구성, 클릭 시 player의 무기 장착 또는 해제 됨
- Skill 쿨타임 표시창은 스킬 사용시 0에서부터 시간이 지날수록 점점 차오름



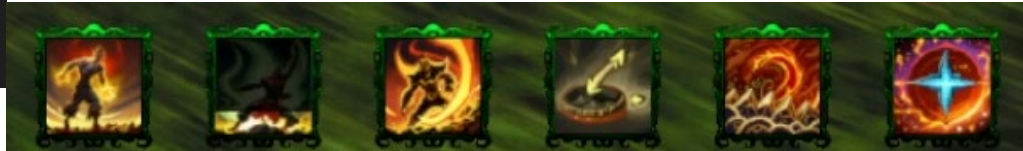
Weapon 선택 메뉴는 각각 **Button**으로 구현되어있고 Clicked/Hovered/UnHovered로 관리된다.
WeaponMenu에서는 **클릭된 Button의 정보**를 가져와 **실제 무기의 장착/해제**를 관리한다

스킬 쿨타임은 Tick에서 **SetPercent**함수를 사용하여 구현했다

```
if(bStartTick)
{
    CurrentTime += InDeltaTime;

    if (CurrentTime >= MaxCoolTime)
        bStartTick = false;

    CoolTime->SetPercent(CurrentTime / MaxCoolTime);
}
```



Enemy와의 거리에 따른 Enemy 체력바의 크기는 Enemy 클래스에서 **PlayerCameraManager**를 가져와 **Target과의 거리**를 **비교**하여 크기를 정한다.
일정 거리 이상 멀어지면 숨겨지도록 구현했다.

개발 후기

벡터의 외적과 내적을 직접적으로 사용해 볼 수 있는 기회였습니다. 적의 위치를 이용해 방향을 구하고 player의 forward벡터와 내적하여 적이 얼마나 가까운 각도 안에 있는지를 판별하고, 외적을 사용하여 적이 오른쪽에 있는지 왼쪽에 있는지 알아내며 벡터의 외적 내적을 정확하게 사용해볼 수 있었습니다.

Delegate를 사용하여 이벤트 처리를 했습니다. 하나의 Delegate를 적용하여 이벤트 발생시 여러가지 처리를 할 수 있었습니다. 간단하고 안전하게 콜백 함수를 호출하는 방식을 사용해 볼 수 있었습니다.

전체적으로 구조의 중요성을 알게되었습니다. 이후에 수정/추가를 위해서 지금은 시간이 더 걸리더라도 잘 짜여진 구조를 만들어가는 것이 나중에는 더 빠르게 완성할 수 있는 방법이라는 것을 알게되었습니다.