입 사 지 원 서

입사 후 포부 및 각오	**소통하고, 새로운 것을 습득하는 것을 두려워하지 않는 개발자가 되도록 하겠습니다.**				
지원 부문	게임 클라이언트	희망 연봉	회사 내규에 따름	입사 구분	신입
휴대폰	010 2473 8815		E-mail	yeonjindo98@gmail.com	

성명	한글	**도 연 진**	영문	Do Yeon Jin
생년월일	1998년 08월 15일		성별	여
주소	서울특별시 노원구 공릉동 408			
포트폴리오	- Youtube https://youtu.be/2MPlEhzdaec - Blog https://yeonjingameprogramming.tistory.com/ - GitHub https://github.com/yeonjinDo/UE4_RPG/tree/master			

	입학년월	졸업년월	출신학교 및 전공	구분(졸업여부)
학력사항	2017.03	2023.02	대진대학교 컴퓨터공학과	학사 졸업
	2014.03	2017.02	영신여자고등학교	졸업

교육 사항	교육 과목	게임 엔진 프로그래머 양성과정
	교육 기관	서울게임아카데미
	기간	2022-11-16 ~ 2023-08-29
	교육 내용	- Unreal Engine - DirectX - C / C++ - 자료구조 / 알고리즘 - STL

자 기 소 개 서

자기소개	해당 업무에 지원한 본인만의 역량과 강점, 지원동기를 소개해주세요
	집중력 있고 적극적인 태도로 임하는 개발자가 되겠습니다. 컴퓨터 공학과에서 진학한 뒤 프론트엔드 Html, Css를 배우면서 개발이 재미있다고 처음 느꼈습니다. 프론트엔드가 재미있었던 이유는 코드를 적는 대로 바로 화면에 보여졌기 때문이었습니다. 하지만 코드의 단순함과 디자인 감각의 필요성으로 인해 금방 지루함을 느끼게 되었습니다. 그 이후 전공과목을 수강하며 게임 프로그래밍을 접하게 되었는데 직접 프로그래밍한 모든 내용들이 게임 세상 내에서 구현이 되고 있다는 점, 개발을 할수록 기능이 완성되어 가는 것이 눈에 보인다는 점이 매력 있게 느껴졌습니다. 또한 수학적 지식을 공부하고 적용하는 과정에서 큰 성취감을 느낄 수 있었습니다. 졸업 후에 게임 개발자로서 역량을 습득하기 위해 게임 엔진 프로그래밍 과정을 수료하였습니다. 이 과정에서 게임 개발에 기초적인 수학과 영어, 자료구조와 알고리즘을 꾸준히 공부하고 정리가 필요한 부분은 블로그에 정리하며 개념을 이해하는 데 집중했습니다. 입사 후에 새로운 환경에 빠르게 적응하고 실무를 배워나갈 수 있도록 노력하겠습니다. 가지고 있는 역량을 최대한 발휘하여 적극적인 태도로 업무에 임하는 개발자가 되겠습니다.
강점	자신의 성격 중 강점은 무엇인가요?
	첫 번째로 긍정적입니다. 저는 밝은 에너지를 가진 사람으로 주변 사람들에게도 긍정적인 영향력이 되었습니다. 주변 사람들에게 도움과 위로가 되어주기도 하고 긍정적인 소통을 통해 분위기를 밝게 만드는 사람입니다. 호기심이 많아서 하고 싶은 일이 많은 성격인데, 이 긍정적인 마인드가 그 일들에 두려움 없이 도전하는 도전 정신과 용기가 되기도 합니다. 프로그래밍을 공부하면서도 지치는 순간에 빠르게 회복할 수 있었던 것도 긍정적인 마인드가 뒷받침되었기 때문이라고 생각합니다. 두 번째로 마음먹은 일을 끝까지 해내는 힘입니다. 하고자 하는 일이 생겼을 때 목표를 명확히 하고 계획을 탄탄히 세우는 성격입니다. 계획을 바탕으로 꾸준히 노력하여 성과를 내는 것에 자신 있습니다. 대학교 시절에 영어 공부를 하겠다는 목표를 가지고 학기 중 다양한 원어민 수업에 참여하고, 휴학 중에 해외에서 공부하여 토익 만점에 가까운 점수를 얻는 성과를 냈습니다. 이 때 노력해서 얻은 영어 실력은 개발을 공부하면서 영어로 된 자료를

	참고할 때 많은 도움이 되었습니다. 또한 체력을 길러야겠다는 필요성을 느끼고 꾸준히 운동하여 근육량과 기초 체력에서 뚜렷한 성장을 보았습니다. 끝까지 해내는 힘은 특히 복잡하고 시간이 많이 소요되는 게임 개발 작업에 필수적이라고 생각합니다. 지치지 않고 꾸준히 공부하며 열정을 가진 개발자가 되기 위해서 이러한 성격을 더욱 키워 완성도 있는 개발에 도움이 되도록 노력하겠습니다.
개발자로서의 목표	개발자로서 이루고 싶은 목표가 무엇이고 그 목표를 이루기 위해 어떠한 노력을 할 계획 인가요?
	단기적인 목표로는 실력이 탄탄한 개발자가 되고 싶습니다. 개발 특성상 개발 도구나 환경이 항상 동일하지 않을 수 있기 때문에 무엇보다도 중요한 것이 기본기라고 생각합니다. 따라서 업무를 수행하며 경험을 쌓고 기본기에 충실하여 완성도 있는 게임을 만드는 개발자가 되고 싶습니다. 이를 위해서 다양한 프로젝트에 적극적이고 능동적인 자세로 참여하여 현업에 대한 이해도를 높이겠습니다. 또한 다양한 직군과 협력하여 진행되는 게임 개발을 위해 항상 소통하고 긍정적인 자세로 원활한 협업에 도움이 되겠습니다. 장기적인 목표로는 보람을 느끼는 개발자가 되고 싶습니다. 해내는 결과물에 만족을 느끼고 개발에 참여한 게임이 이용자들에게 좋은 평가를 받아 보람을 느끼는 개발자가 되고 싶습니다. 업무에 책임감을 가지고 적극적인 태도로 팀원들과 협력하여 목표를 달성해내겠습니다. 게임을 즐기고 지치지 않는 개발을 하며 게임에 대한 이해도를 높이겠습니다.
직무 준비성	이 직무에 지원하기 위해서 어떤 준비를 해왔으며 어떠한 결과를 얻었나요?
	대학교에서 여러 가지 전공과목을 수강하고 주변 선배들의 진로를 보고 들으며 게임 프로그래밍에 관심을 가지게 되었습니다. 그래픽스 과목을 수강하며 OpenGL을 공부하고 그래픽 프로그래밍 관해서 공부했습니다. 그 후, 유니티 관련 서적을 읽으며 게임 엔진을 처음 접했고, 이를 참고하여 졸업 작품을 개발했습니다. C와 C++로 콘솔 창에 미니 게임을 만들었고, 그 후 엔진을 이해하기 위해 Window API와 DirectX를 사용한 게임 개발을 공부하고 2D게임을 제작했습니다. 이후 국비 과정으로 게임 엔진 프로그래밍을 수강하게 되면서 3D게임을 공부했습니다. 이때 Unreal 엔진을 처음 접하게 되었고 Blueprint와 Unreal C++을 활용한 게임 개발을 배우고 개인 프로젝트에 적용해 보며 이해도를 높였습니다.

	게임 개발에 필수적인 자료구조를 통해 메모리에 대해서 알아가고 코딩 테스트 사이트를 통해 다양한 알고리즘을 공부하며 문제 해결 능력을 길렀습니다. 또한 수업 시간 외에도 같은 반 동기들과 그날 수업에서 배운 것들을 나누고 과제를 해결하며 프로그래머로서 원활한 소통 능력을 향상시켰습니다. 같은 문제를 서로 다른 방식으로 구현한 것에 대해서 나누고 창의적인 문제 해결 방법을 생각해내는 것을 즐겼습니다.

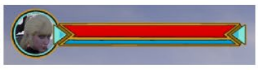
포 트 폴 리 오

소개

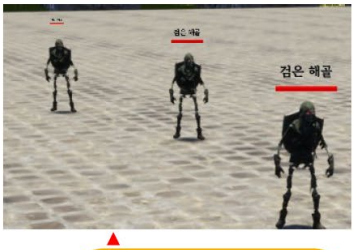
개발자	도연진
장르	액션 RPG
개발 환경	Unreal 4.26
개발 기간	12주 (24.1.1 ~ 24.3.1)
영상 링크	https://youtu.be/2MPlEhzdaec
블로그	https://yeonjingameprogramming.tistory.com/

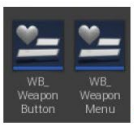
UI

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

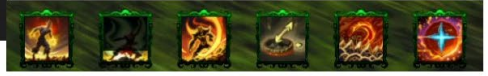




Weapon 선택 메뉴는 각각 Button으로 구현되어있고 Clicked/Hovered/UnHovered로 관리된다.
WeaponMenu에서는 클릭된 Button의 정보를 가져와 실제 무기의 장착/해제를 관리한다

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

    if (CurrentTime >= MaxCoolTime)
        bStartTick = false;

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

스킬 쿨타임은 Tick에서 SetPercent함수를 사용하여 구현했다



Enemy와의 거리에 따른 Enemey 체력바의 크기는 Enemy 클래스에서 PlayerCameraManager를 가져와 Target과의 거리를 비교하여 크기를 정한다.
일정 거리 이상 멀어지면 숨겨지도록 구현했다.

목차

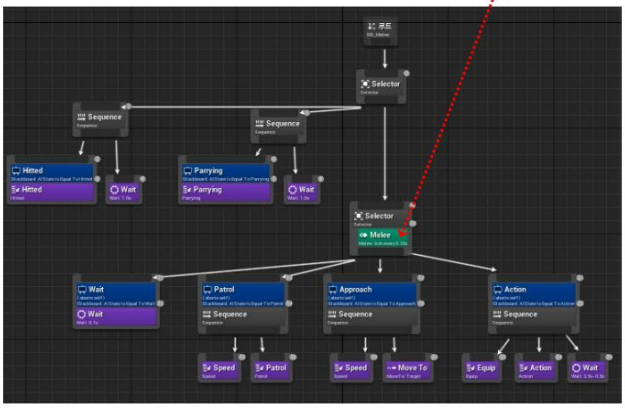
Page

Enemy AI

Behavior Tree

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

Service에서 Enemy 상태에 따른 Blackboard Key 설정



· 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

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 내적한 값이 가장 작은 25%을 판별

타겟의 후방으로 이동하도록
Vector(Target, Enemy)과
Vector(Target, Item)을 내적하여
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

Sword Data Asset

Player
- BindAction() 으로 WeaponComponent::DoAction 호출

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

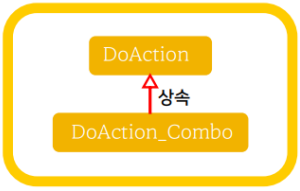
WeaponAsset
- WeaponAsset만 WeaponData에 접근 -> friend class로 열어줌
- 각 무기마다 OnEquip, OnAttachment, OnCollision 등의 이벤트 연결

WeaponData
- Asset 공유 문제 발생 방지 -> WeaponData에 생성 객체 분리

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

Weapon

Combo



- Anim_NotifyState로 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 hitted된 캐릭터가 여러 개라면 카메라의 방향을 플레이어의 전방 방향에서 가장 가까운 캐릭터에게 고정

전방 방향에서 가장 가까운 캐릭터는 내적을 통해 구한다

cos 0° = 1이므로 플레이어의 정면에 가까울수록 1에 가까운 값이 나온다

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

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

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

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

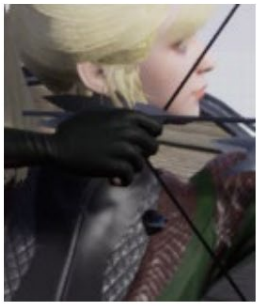
Weapon

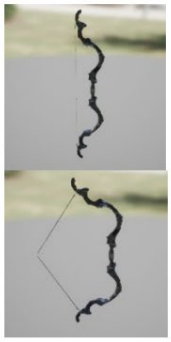
Bow

· 활의 휨은 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

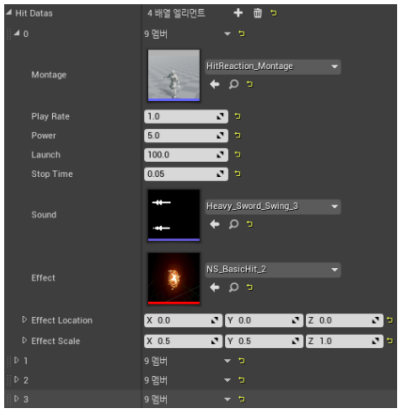
· hit된 물체가 없을 경우
spawnDirection = end - 화살 spawn 위치

```
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 FHitData를 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 대미지를 전달한다

Defend

Parrying, Dodge

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_M	1.000000
3	3	Hammer	AnimMontage'/Game/Character/Montages/Hammer/Hammer_AvoidAtta	1.000000

DT_AvoidAttack

· player는 tab키로 방어 모드로 전환 가능

· Anim_Notify_State로 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

```
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

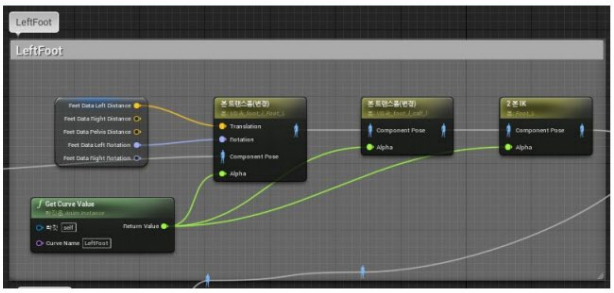
MontagesComponent에서 Montage를 재생하면 DataTable에 들어가있는 애니메이션이 재생된다

회피기 또한 DataTable로 만들어져 무기에 맞는 회피기가 자동으로 재생된다

IK

Feet IK

왼발 IK

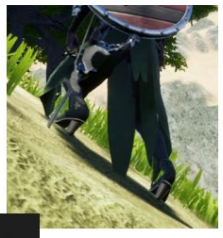


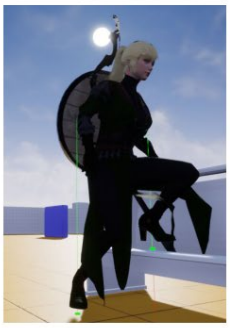
· 애니메이션을 프로그래밍으로 수정하기 위해서
로컬 공간을 컴포넌트 공간으로 변경

· 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



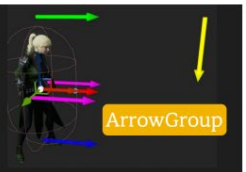


두 점 사이의 상대 좌표(x, y)를 받아 절대각을
반환하는 Atan2함수로 cos값을 알아내 발 각도를
지면에 맞게 조정한다

```
//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);

//파쿠르 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 LineTrace를 수행

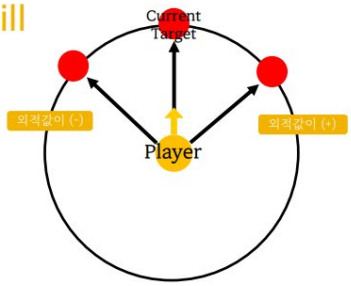
· LineTrace의 HitResult를 체크하여 파쿠르 수행 가능 여부와
장애물의 유형을 구분

· 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( Xx: 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 direction벡터를 외적하여 hitCharacter가 player보다 왼쪽에 있는지, 오른쪽에 있는지 구한다

이 외적된 값과 player의 up벡터를 내적하여 map의 key값에 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

```
PosA = FVector(Start.X, Start.Y, In2:200);
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
);
```

매개변수에 위의 초기화한 PosA~D가 들어간다

```
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

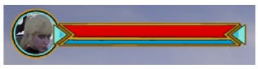
· player의 미래 위치를 예상하여 움직여 보스의 난이도 조절

```
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

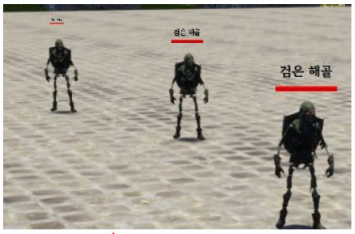


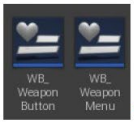
· player 체력바에는 Hp, Mana가 있음

· 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



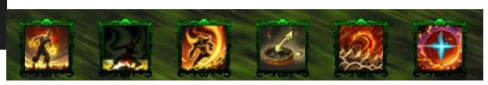


Weapon 선택 메뉴는 각각 Button으로 구현되어있고 Clicked/Hovered/UnHovered로 관리된다.
WeaponMenu에서는 클릭된 Button의 정보를 가져와 실제 무기의 장착/해제를 관리한다

Enemy와의 거리에 따른 Enemey 체력바의 크기는 Enemy 클래스에서 PlayerCameraManager를 가져와 Target과의 거리를 비교하여 크기를 정한다.
일정 거리 이상 멀어지면 숨겨지도록 구현했다.

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

    if (CurrentTime >= MaxCoolTime)
        bStartTick = false;

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

스킬 쿨타임은 Tick에서 SetPercent함수를 사용하여 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.

작업 후기	C++ 포트폴리오를 개발하기 전, 블루프린트 포트폴리오를 개발하면서 가장 크게 느낀 점은 '튼튼한 구조가 중요하다' 였습니다. 처음 시작할 때 기능 구현하는 것에만 신경쓰다 보니 결국 구조적으로 지저분해지고 수정이 어려워 결과적으로 시간이 두 배로 걸리는 상황이 오고는 했습니다. 따라서 이번에 C++포트폴리오를 시작할 때는 전체적으로 구현하고 싶은 기능들을 크게 정해놓고 구조를 짜는데 더 많은 시간을 들였습니다. 그 결과 나중에 추가하고 싶은 스킬이나 작은 수정사항이 생겨도 빠르게 보완할 수 있었습니다. 어려웠던 점 중 또 하나는 마지막까지 집중하는 것 이었습니다. 포트폴리오가 점점 완성되어가는 것이 눈에 보일수록 집중력이 떨어져 계획했던 일정이 점점 연기되었습니다. 이를 극복하기 위해 저는 완성이 되어갈수록 일정을 더욱 타이트하게 잡는 방법을 선택했습니다. 몸은 힘들었지만 완성을 하고 나니 마음이 뿌듯했습니다.