|  |
| --- |
| *Development of 3D game and reinforced learning* |

*2021-1st Semester*

*Interdisciplinary Project*

|  |  |
| --- | --- |
| Student ID | 20151028 |
| Name | 권상민 |
| School | ECE |
| 1 track | CSE |
| 2 track | MGE |
| Advisor 1 | Sung-Ahn Ko |
| Advisor 2 | Gi-Soo Kim |

I. Introduction

1) Topic and Purpose of Research

Recently, with the rise of metaverse technology, interest and demand for Unreal Engine and Unity3D are increasing. Also, I want to be a Game Programmer. So, I will make a 3D game wtih Unreal Engine. the reason why i choose Unreal Engine platform (not the Unity3D) is that Unreal Engine has various advantages such as :

* The Graphic Quality of the Unreal Engine is amazing and workable.
* The usability of the program is very much in demand.
* The user interface of Unreal Engine keeps on updating with the latest and newer tools and options.
* It has simple codes and uses nodes called Blueprints. These nodes help the users to create video games and other High-end games without writing scripts and codes.
* This uses the [C++ programming language](https://www.educba.com/c-programming-language-basics/), which itself makes the program a developer’s first choice program.

|  |
| --- |
| Unreal Engine Sample Image |
|  |

II. Main Subject

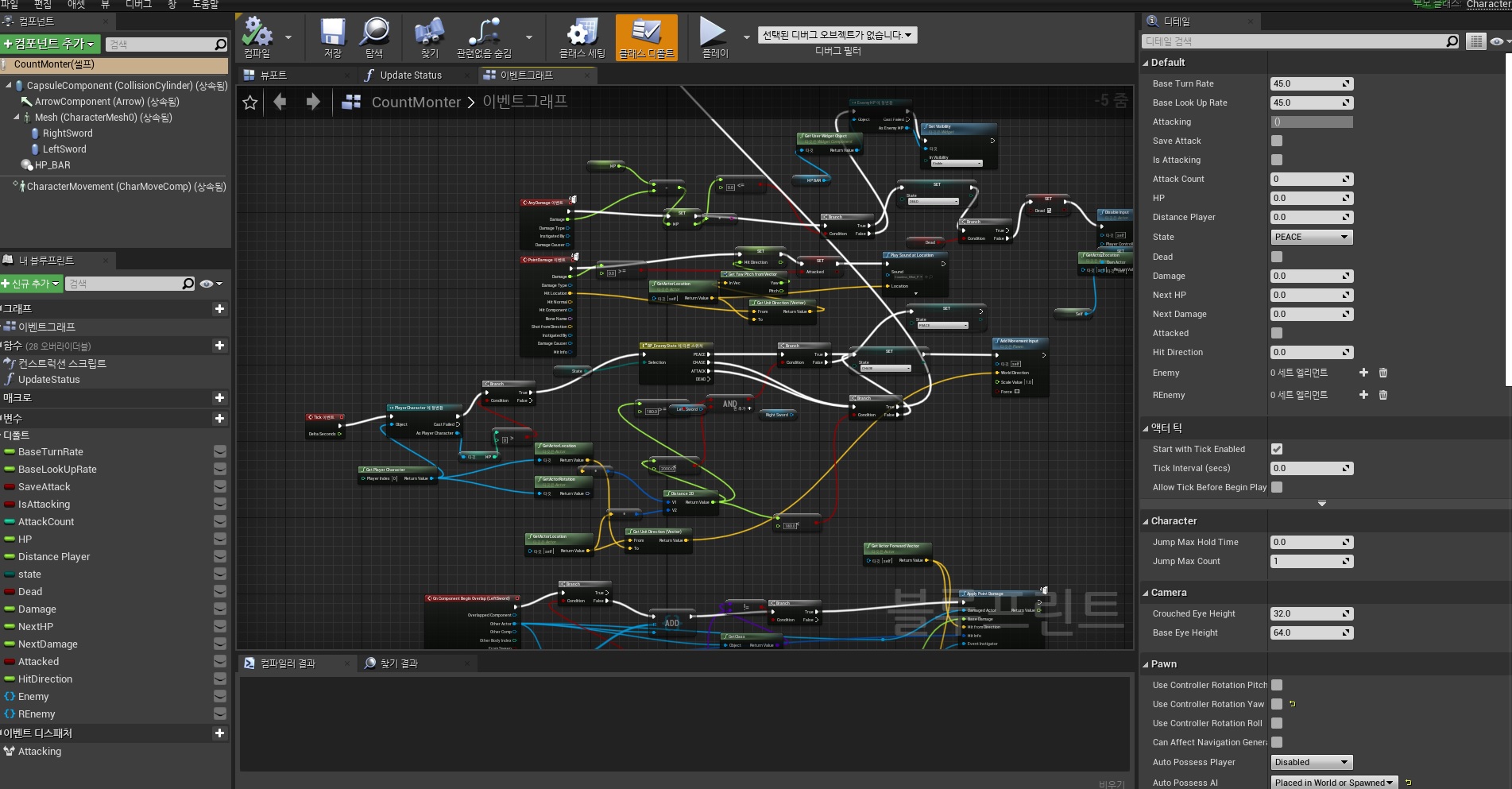
1) Research Planning

1. Learning Unreal Engine

Since I had never used Unreal Engine before, I started studying the features by referring to the official Unreal Documentation.

I will introduce 2 major system which i learn from Unreal Documentation

* Blueprint System is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within Editor. As with many common scripting languages, it is used to define object-oriented (OO) classes or objects in the engine.



* C++ Programming with VisualStudio is a very convenient system. Unreal Engine 4 provides two toolsets for programmers which can also be used in tandem to accelerate development workflows. New gameplay classes, Slate and Canvas user interface elements, and editor functionality can be written with C++, and all changes will be reflected in Unreal Editor after compiling with either Visual Studio.

2. 3D game production

To make a 3D game, I needed a lot of assets such as animations, meshes, skeleton meshes, audio, interfaces, monsters, projectiles, environments, and so on. To get those one, I use epic games marketplace. They provided some free assets such as meshes, brick actors, skeleton meshes, sounds and animations.With those assets, I will create a 3D action game in which Paragon characters appear as monsters and player characters in the medieval era.

3. Making Monster AI

To make Monster AI, I will use the Unreal Engine Blueprint System.

This AI will search the neighborhood, looking for player characters, and when they find them, they will chase and attack the player. If a player dies, they will go back to their place and wait for another player to appear.

III. Conclusion and Discussion

1) Research Results

Synchronizing Input and Animation

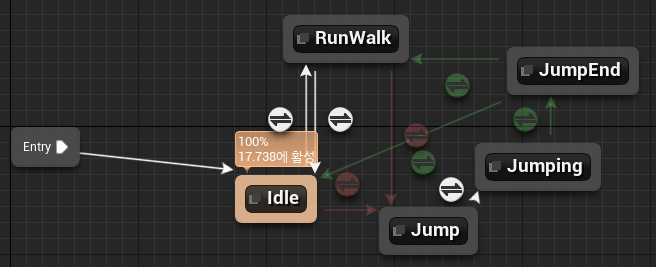
* My Player Character Actor can get 4 type Input.
  + 1. Movement Input (W,A,S,D, Space bar)
    2. Attack Input (Mouse Left Button, Mouse Right Button)
    3. Run/Walk Input (Left Shift)
    4. Camera Movement Input (Mouse Movement)

Axis Input : W,A,S,D, Space bar, Mouse Movement

Controller Components of Unreal Engine Class take inputs from User or AI. But, the actual objective component which I want to move is Character Actor. Thus, I have to pass the input value to the Character Actor.

|  |
| --- |
| // Called to bind functionality to input void APlayerCharacter::SetupPlayerInputComponent(UInputComponent\* PlayerInputComponent) {  Super::SetupPlayerInputComponent(PlayerInputComponent);  InputComponent->BindAxis("MoveForward", this, &APlayerCharacter::MoveForward); }  void APlayerCharacter::MoveForward(float Axisvalue) {  if (Controller != nullptr && Axisvalue != 0.f)  {  FVector Direction = FRotationMatrix(GetActorRotation()).GetUnitAxis(EAxis::X);  Direction.Z = 0;  Direction.Normalize();  AddMovementInput(Direction, Axisvalue);  } } |

This is sample code for moving Character Actor.



with the above Direction, Axis Value and private transition rules of the State machine, I can update the animation to play.

**IN GAME OUTPUT With W, D inputs**



Action Input : Mouse Left Button, Mouse Right Button, Left Shift

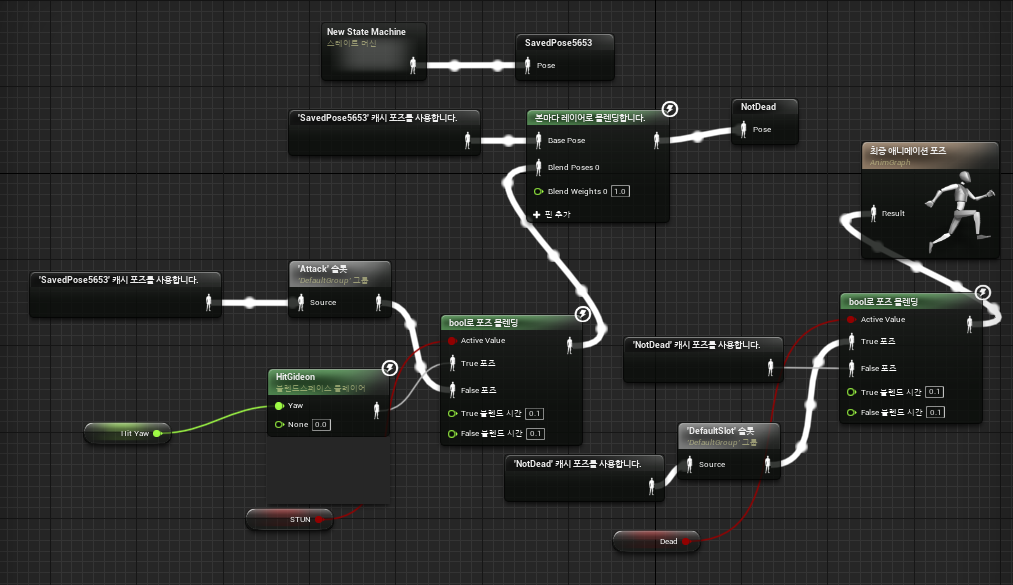
|  |
| --- |
| void APlayerCharacter::SetupPlayerInputComponent(UInputComponent\* PlayerInputComponent) {  InputComponent->BindAction("Attack", IE\_Pressed, this, &APlayerCharacter::LMB);  InputComponent->BindAction("BlackHole", IE\_Pressed, this, &APlayerCharacter::BlackHole);  InputComponent->BindAction("BlackHole", IE\_Released, this, &APlayerCharacter::CanFire); } void APlayerCharacter::AttackEnd() {  IsAttack = false;  resetcombo(); } void APlayerCharacter::Combo() {  if (CanNextCombo)  {  UE\_LOG(LogTemp, Warning, TEXT("Combo with Can Next Combo"));  Attack();  }  else  UE\_LOG(LogTemp, Warning, TEXT("Combo withOut Can Next Combo")); } void APlayerCharacter::Fire() {  FVector EyeLocation;  FRotator EyeRotation;  GetActorEyesViewPoint(EyeLocation, EyeRotation);    FActorSpawnParameters SpawnParams;  SpawnParams.Owner = this;  SpawnParams.Instigator = this;   UWorld\* World = GetWorld();  if (CurrentCombo != 2)  {  EyeLocation = GetMesh()->GetSocketLocation(TEXT("weapon\_soc"));    }  else  EyeLocation = GetMesh()->GetSocketLocation(TEXT("hand\_l"));  auto BULLET = World->SpawnActor<ABullet>(ABullet::StaticClass(), EyeLocation, EyeRotation, SpawnParams);  if (BULLET)  {  FVector LaunchDirection = EyeRotation.Vector();  BULLET->Player = this;  }  else {  UE\_LOG(LogTemp, Log, TEXT("BULLET CREATE FAIL"));  } } void APlayerCharacter::resetcombo() {  UE\_LOG(LogTemp, Warning, TEXT("ResetCombo"));  IsAttack = false;  CanNextCombo = false;  CurrentCombo = 0;  FIREBlackHole = false; } void APlayerCharacter::BlackHole() {  if (IsAttack == false && !RMBullet && !stun)  {  auto Instance = GetMesh()->GetAnimInstance();  IsAttack = true;  Instance->Montage\_Play(AttackMontage);  Instance->Montage\_JumpToSection(FName("RMB"), AttackMontage);  RMB = true;  } } void APlayerCharacter::BlackHoleCreate() {    FVector EyeLocation;  FRotator EyeRotation;  GetActorEyesViewPoint(EyeLocation, EyeRotation);  channum = 0;  FActorSpawnParameters SpawnParams;  SpawnParams.Owner = this;  SpawnParams.Instigator = this;    FVector LaunchDirection = EyeRotation.Vector();  LaunchDirection.Normalize();  EyeLocation += LaunchDirection \* 100;  UWorld\* World = GetWorld();    RMBullet = World->SpawnActor<ARMBBullet>(ARMBBullet::StaticClass(), EyeLocation, EyeRotation, SpawnParams);  } void APlayerCharacter::CanFire() {  if(RMBullet||RMB)  FIREBlackHole = true; } void APlayerCharacter::RMBFire() {  if (RMBullet)  {  auto Instance = GetMesh()->GetAnimInstance();  channum = 0;  Instance->Montage\_Play(AttackMontage);  Instance->Montage\_JumpToSection(FName("RMBfire"), AttackMontage);  FVector EyeLocation;  FRotator EyeRotation;    RMBullet->Player = this;  RMBullet->FireInDirection();  RMBullet = nullptr;  RMB = false;  }  else {  UE\_LOG(LogTemp, Log, TEXT("RMBullet CREATE FAIL"));  } } |

If you look closely at the code, you will find that it binds the input and the function using a different method than the movement input. The reason why it is different is that Movement input is a continuous input. Therefore, the MoveForward function has a variable that has information about how long a specific input value was pressed as a function parameter. Contrary to this, in the case of the attack button, a specific function is executed whenever the user presses the mouse, so that the function is executed only once even if a specific input is input for a long time.

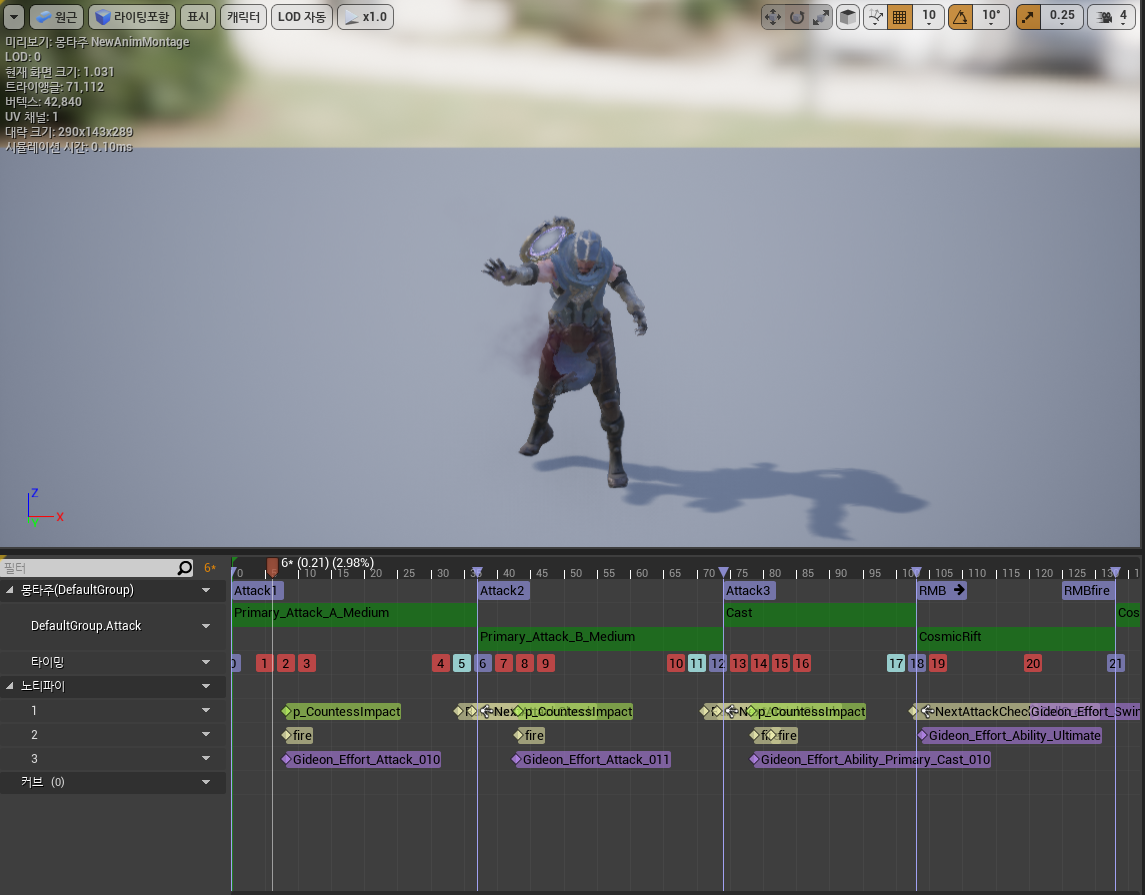
Due to the characteristics of the action input described above, if there is no restriction on the attack function, the attack operation is executed every time the mouse is clicked, and the attack animation is re-executed from the beginning each time. Therefore, in order to provide a delay between attacks as in a normal game, it is necessary to make it impossible to execute the attack function during the attack animation execution time. And in order to smooth this process and implement a combo attack, the Animation Blend and Animation Montage functions provided by Unreal were used.

Animation Blend provides a function that connects different Animations naturally, and Animation Montage provides a function similar to an array that stores multiple Animations.

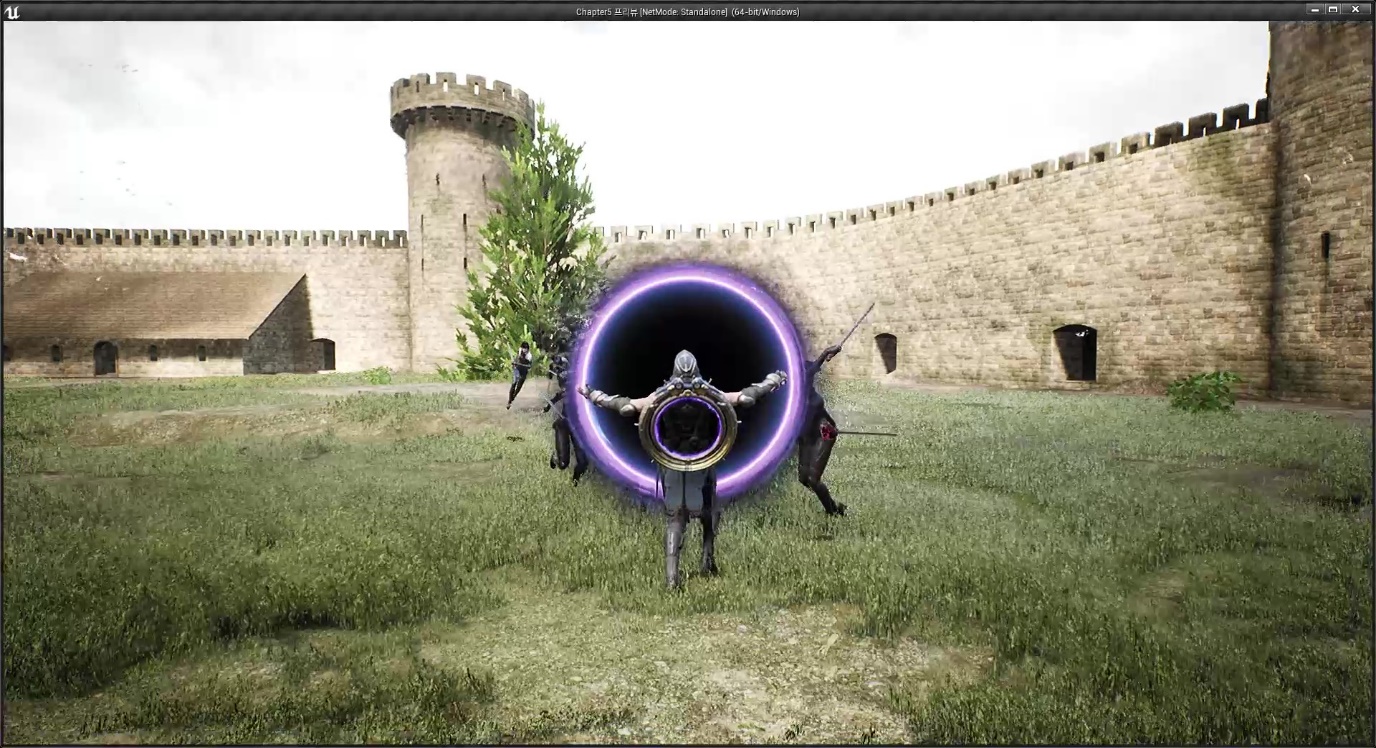
Animation Blueprint : This blend different animations by referencing socket, variables.



Animation Montage : As you can see in the picture below, there are several slots and different animations are executed in each slot. And notify is set for each specific frame of each animation. Synchronization was implemented using these notify.



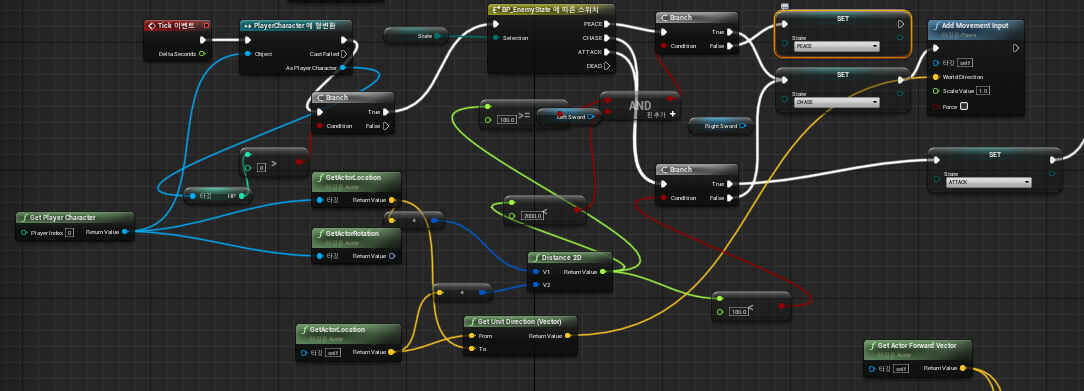
IN GAME OUTPUT WITH Attack Inputs



Making Monster and Monster AI

When creating a monster, I must follow the same process for creating the player character above. So, in this section, I am going to talk about Monster AI work.

I implemented Monster AI using Blueprint. Below Picture is naïve my Monster AI architecture.



Monsters have 4 status values: PEACE, CHASE, ATTACK, and DEAD.

Basically, it maintains the PEACE state, and if the player comes within a certain distance, then the monster identifies the player and chases it and performs an attack. During this process, the monster utilizes the Sphere-shaped invisible components set on the weapon and static mesh component to detect Overlap Events and Hit Events and inflict damage to or damage from the player. When HP falls below 0, it waits for a while and is respawned with a slightly higher Status at a random point. The reason for reusing the actor without destroying it is that reusing the actor consumes less computer resources than creating a new one.



Chase and Peace State



Attack and Chase State : You can see that the monster approaches and attacks within a certain distance. In addition, it can be confirmed that the attack operation is successfully transferred to the player character and the HP decreases.

Demo video :

[](https://www.youtube.com/embed/3821_dTLer4?feature=oembed)

2) Discussion

Although we have successfully implemented all the components of a 3D action game, the content is still insufficient to be released as a real game. Therefore, it is necessary to increase the diversity of contents by obtaining additional assets and creating more diverse monsters and player characters.

Also, sometimes animation does not run normally due to frame drops and errors, so it seems necessary to move the functions implemented in Blueprint to C++.

※ References

<https://www.educba.com/what-is-unreal-engine/>

https://docs.unrealengine.com/4.26/ko/