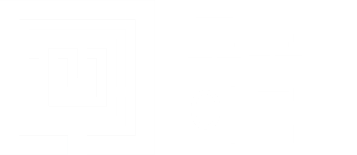
|  |
| --- |
| VR 프로그래밍 심화과정  with LiiYuu Senpai |
|  |
| 2021.09 ~ 2021.11  게임과 3학년  작성자: 함승호 |



# 1주차

# 강사소개, 수업계획, 입력매핑, 서버(이론)

# 2주차

# 총 스폰&장착, 데미지처리, 애니메이션

# 3주차

# SetUpWeapon, LineTrace, 카메라 회전 고치기

# 4주차

# 5주차

# 6주차

# 7주차

# 8주차

# 9주차

# 10주차

# 수업계획

# 여러가지 도구를 활용하는 네트워크 5대5 FPS 게임

# 패키징을 하여 배포까지 하는것이 목표.

# 참고게임 : RainbowSix Siege

# 

# 사용 버전 : UE4.27

# 입력 매핑

# 캐릭터 이동에 대한 Input 할당을 해줘야 한다.

# [세팅 → 프로젝트 세팅 → 입력]

# 텍스트, 하늘, 스크린샷이(가) 표시된 사진 자동 생성된 설명

# FPS 게임이므로 이동에 WASD 를 사용할 것이다.

# 텍스트, 스크린샷, 모니터, 화면이(가) 표시된 사진 자동 생성된 설명

# 왼쪽의 슬라이더바에서 ‘입력’ 카테고리로 이동한다.

# 텍스트, 전자기기이(가) 표시된 사진 자동 생성된 설명

# 축매핑 옆의 + 를 누르면 새로운 축 매핑을 생성한다.

# MoveForward 라는 이름의 축 매핑에 W, S 라는 키 값을 각각 W = 1, S = -1을 Scale 로 할당시킨다.

# 마찬가지로 MoveRight 축 매핑에 A = -1, D = 1 을 Scale 할당한다.

# 

# x, Y축 및 번호 - 로열티 프리 그래프 스톡 사진ㅁ

S

A

D

W

# 중앙에 캐릭터가 있다고 생각하면 Scale 할당이 이해 될 것이다.

# 캐릭터 C++ 클래스를 만든다.

# [파일→새로운C++클래스] 에서 부모클래스 Character 를 선택한다.

# 

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Character를 부모 클래스로 생성한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# ' BaseCharacter ' 라는 퍼블릭 클래스를 만든다.

# 클래스 생성을 누르면 C++코드 컴파일 후, Visual Studio 에서 로드가 필요하다 할 것이다. (자동으로 될 때도 있다)

# 모두 로드 한다.

# Visual Studio를 꺼버렸다면 [파일 → Visual Studio열기] 로 다시 켤 수 있다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 솔루션 탐색기에서BaseCharacter.cpp , BaseCharacter.h 파일이 있는지 확인하자.

# Games → Source → (프로젝트이름) → Public 에 에디터에서 생성한 BaseCharacter의 헤더 파일이 있다.

# class 프로젝트이름\_API ABaseCharacter : public ACharacter 클래스 내부에 설정한 입력 매핑을 함수로 선언한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 파라미터인 float형 변수AixsValue 에는 매핑 설정시 Scale 에 넣어둔 값이 할당된다.

# 헤더파일에 선언을 했으니 .cpp에 정의를 해줘야 한다.

# ※팁 : 초록색 물결줄이 그어지는 이유는 선언만 하고 정의가 되어있지 않은것. 포인터를 변수명 우에 두고 ctrl + . 를 누르면 정의 만들기를 간단하게 할 수 있다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# BaseCharacter.cpp 에 함수의 정의가 선언되었다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 이제 이동구현을 할 것이다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# ControlRotation : 캐릭터의 회전값을 추출

# WorldDirection : 회전값 만큼 회전한 캐릭터가 바라보고 있는 방향을 추출하여 저장한다.

# AddMovementInput : WorldDirection방향(바라보고있는방향)으로 Scale(AxisValue) 만큼 이동

# FrotationMatrix 로 ControlRotation을 회전 행렬로 바꿔준다.

# 현재 회전 값 상태에서 축을 얻어온다.

# 얻어온 회전 값 중 Y축에 해당하는 UnitVector(normalizedVector)를 WorldDirection으로 정해서 움직인다.

# Frotator , FVector 을 이해하기 위해 Yaw, Pitch, Roll 에 대한 지식이 필요하다.

# ※수학에서 사용하는 좌표계와 다르다!

# Yaw, Pitch, Roll 바로알기-스테이지 기초이론1

# 그림에 보이는대로

# Yaw는 Z 축을기준 좌/우 로 회전한다.

# Pitch 는 X축을 기준으로 위/아래 로 회전한다.

# Roll 은 Y 축을 기준으로 시계 / 반시계 방향으로 회전한다.

# 회전방향 : 양수, 역회전방향 : 음수

# ABaseCharacter클래스에 정의한 함수를 작성한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 저장하고 에디터에서 컴파일시킨다.

# C++클래스를 부모클래스로 하는 실질적으로 이동하는 캐릭터를 만들어준다.

# 

# 콘텐츠 브라우저의 빈 공간을 우클릭 후, 블루프린트 클래스를 선택한다.

# 

# 모든 클래스 버튼을 확장하여 BaseCharacter 를 검색하고 선택한다.

# 텍스트, 노란색, 스크린샷이(가) 표시된 사진 자동 생성된 설명

# ‘BP\_Character’ 라는 이름으로 만들어준다. 사람 형태의 엑터가 생성되었다.

# 캐릭터만 생성했다고 되는 것이 아닌, 이 캐릭터를 메인 캐릭터로 사용할 게임 모드를 만들어줘야한다.

# 다시 콘텐츠 브라우저의 빈 공간을 우클릭 후, 블루프린트 클래스를 선택한다.

# Game Mode Base 를 선택한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 텍스트, 장치, 측정기이(가) 표시된 사진 자동 생성된 설명

# ‘BaseGameMode’ 라는 이름을 주었다. 게임기 형태의 엑터가 생성되었다.

# 더블클릭하여 게임모드블루프린트를 연다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 우측의 디테일 패널에서 Default Pawn Class 를 우리가 생성한 BP\_Character로 설정한다.

# ※C++클래스인 BaseCharacter 가 아니다!

# 이 게임모드를 만드는 게임의 베이스게임모드로 설정해줘야한다.

# 텍스트, 스크린샷, 모니터, 화면이(가) 표시된 사진 자동 생성된 설명

# 프로젝트 세팅의 가장 상단 카테고리 맵&모드 의 Default Modes 를 생성한 ‘BaseGameMode‘ 로 설정한다.

# 

# 그리고 에디터의 블루프린트의 레벨 블루프린트와 월드 오버라이드의 Gamemode를 BaseGameMode 로 재정의한다.

# 

# 에디터 상단에 컴파일 버튼이 있다.

# 플레이 해서 이동을 확인한다.

# 이동은 하지만 마우스에 대한 시점 회전을 하지 않는다.

# 마우스 회전에 대한 축 매핑을 할당한다.

# LookUp 축 매핑에는 키 값 마우스 Y , Scale = -1 로 할당한다.

# Turn 축 매핑에는 키 값 마우스 X , Scale = 1 로 할당한다.

# 

# 이동때와 마찬가지로 헤더파일에 선언하고,

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# BaseCharacter.cpp에 정의한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Turn은 카메라를 좌/우로 회전시키므로 YawInput을

# LookUp은 카메라를 상/하로 회전시키므로 PitchInput 을

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# ABaseCharacter 클래스에 정의한 함수를 작성한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 저장하고 에디터에서 컴파일. 플레이해서 확인한다.

# Networking and Multiplayer

# UE4 멀티플레이는 Client – Server 모델을 기반으로 한다.

# 즉 중앙에 서버가 있고, 클라이언트들이 주변에서 접근을 하는 방식이다.

# ( client to server)

# 서버의 구조는 2가지 존재한다.

# Listen Server : 클라이언트이자 서버.

# → 서버의 역할도 하고 클라이언트의 역할도 하는 서버이다.(서버 상태로 플레이 가능)

# Dedicated Server : 서버의 역할만 하는 독립적인 서버.

# → 렌더링 관련 처리를 아예 하지 않는다. 별도 프로세스에서 서버 로직이 돈다.

# 소유권과 NetRole

# 스폰되어 있는 모든 엑터들은 NetRole 을 가지고 있다.

# ※Netrole 에는 4가지의 종류가 있다.(네트워크 상태)

# 1. Authority : 해당 엑터에 권한 O

# 2. Autonomous Proxy : 권한 X, 소유권 O

# 3. Simulated Proxy : 권한 X, 소유권 X

# 4. None : 지정 Role이 없다.

# 서버에서 생성된 액터들은 기본으로 Authority 속성을 가지게 된다.

# 각 클라이언트들은 빙의한 폰에 대해 소유권을 가지게 된다. // 설명 조금 부족

# (서버에서 생성한 캐릭터 폰에 클라이언트가 접속한다)

# /\*테스트해보기\*/

# [ Has Authority ]

# 

# Has Authority 함수를 이용해서 엑터에 대한 권한을 가지고 있는지 알 수 있다.

# Authority를 가지고있다면(true) “True” 를 가지고 있지 않다면(false) “False”를 출력하게 했다.

# 

# 

# 에디터의 플레이모드 세팅을 변경한다.

# 플레이어 수 : 2, 넷 모드 : Play As Client

# (넷모드 바군 이유 찾아바야댐!)

# 플레이를 한다면 클라이언트가 2개 나오고, 디버그가 출력된다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Server, Client1, Client2 모두 캐릭터를 두개 가지고 있지만, Server 에서만 HasAuthority 권한을 가지고 있음을 알 수 있다.

# [ Is Locally Controlled ]

# 

# Is Locally Contolled : 로컬에서는 조종중인지 여부를 bool 타입으로 받는다.

# 캐릭터에 빙의된 컨트롤러가 있고, 컨트롤러의 인스턴스가 클라이언트에 있을 때 true가 된다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Client1, 2 에 각각 하나의 컨트롤러가 들어가 있으므로 한번씩 출력된다.

# Q . 이런 서버에서 피격판정은 어떻게 일어나는가?

# A. Client A : 총알 발사 이벤트 실행 -> A 캐릭터의 [발사함] 로직실행

# -> 서버의 [A캐릭터 발사] 로직 실행 -> 누군가 hit -> 권한있음? -> 없다면(Client) 실행 X, 있다면(Server) 발사 이벤트 실행 O -> 실제로 대미지를 줌 -> hit된 대상 체력 감소

# 권한있음? 을 실행하는 이유 : 누군가 악의적으로 1 데미지를 1000데미지로 바꿀 수 있다.

# [ Authority ]

# 

# ApplyDamageToMe 라는 커스텀 이벤트를 만들었다.

# 이 이벤트는 HasAuthority 를 가지고 있는 객체에서 최초 1회 실행한다.

# 이제는 알 수도 있지만 Server에서만 호출이 된다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# (당구장기호) 이해가 잘 안되는 사람을 위해 : 간단하게 생각하면 Authority 를 가지고 있다면 서버, 가지고 있지 않다면 클라이언트 이다.

# [ Get Local Role ]

# NetRole 상태값들을 출력한다.

# NetRole 에는 Authority, Autonomous Proxy, Simulated Proxy 가 있다.(none은 논외로)

# 텍스트, 전자기기이(가) 표시된 사진 자동 생성된 설명

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Server에서는 Authority출력되고,

# Client에서는 AutonomousProxy, SimulatedProxy출력된다.

# 

# 이런식으로 된다.

# 텍스트, 스크린샷, 화면, 검은색이(가) 표시된 사진 자동 생성된 설명

# 에디터의 플레이모드 세팅을 변경한다.

# 플레이어 수 : 2, 넷 모드 : Play As Listen Server

# Listen Server 설정에서 실행할 경우 에디터에 보여지는 화면이 서버의 역할을 하게 된다.

# 텍스트, 스크린샷, 전자기기, 컴퓨터이(가) 표시된 사진 자동 생성된 설명

# 이 화면이 서버가 된다.

# BP\_Character의 Tick 이벤트에 간단한 함수를 추가한다.

# 액터의 위치에 게속하여 액터의 권한이 표시될 것이다.

# 

# #동기화 기법

# 동기화를 시키는 방법에는 두가지가 있다.

# 1.변수 Replication

# Replication. 즉 서버에서 값이 변경된다면 조건에 맞는 클라이언트에 복제된다.

# 2.RPC(Remote Procedure Call)

# 로컬에서 호출되지만 다른 머신에서 원격 실행되는 함수

# ※RPC 함수의 종류

# 1. Server RPC

# 2. Client RPC

# 3. Multicast RPC

# Tip : RPC 는 호출되는 위치와 호출하는 위치에 따라 실행이 안될 수 있다.

# example) 클라이언트1에서 A, 2에서 B 변수를 생성한다. 서버에서는 A, B의 값을 할당하고 클라이언트와 동기화한다. 그리고 마지막에 동기화한 값을 출력한다.

# 

# 하지만 순서가 A 생성 → 동기화 → B생성 → 출력 ( 위의 그림 ) 이 된다면, 출력에서 동기화한 변수 B에 해당하는 값은 None 이므로 정상적으로 작동하지 않을것이다.

# [ RPC 사용하기 - ServerRPC ]

# 

# ServerRPC는 Autonomous Proxy 이상의 권한에서 호출가능

# 커스텀 이벤트 ServerPrintString 을 만든다.

# Print string은 Call On Server를 출력하게 한다.

# 

# C++ 에서는 PRC 태그를 UFUNCTION 메타지정어를 통해 지정할 수 있다.

# 자료(https://docs.unrealengine.com/4.27/ko/InteractiveExperiences/Networking/Actors/RPCs/)

# 블루프린트에서는 이벤트의 디테일 패널에서 설정한다.

# 텍스트, 스크린샷, 모니터이(가) 표시된 사진 자동 생성된 설명

# 리플리케이트에 ‘서버에서 실행’ 을 선택한다.

# ServerPrintString 이벤트는 클라이언트에서 호출한다.

# 

# (Authority를 가지고 있지 않다 = 클라이언트)

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 클라이언트에서만 확인하면 된다.

# Play As Client모드에서 1개의 클라이언트만 생성되게 한다.

# 출력에는 Server : Call On Server 가 출력되고,

# 캐릭터 엑터 위치에는 ROLE Autonomous Proxy 가 계속해서 있을 것이다.

# ServerPrintString 이벤트를 서버에서 실행시킨다면 어떻게 될까?

# 된다. ServerRPC는 Authonomous Proxy 이상에서 실행이 되기 때문

# 

# [ RPC 사용하기 – ClientPRC ]

# ClientRPC : 클라이언트에서 서버로 호출하는 RPC

# 

# ClientRPC는 Authority 권한에서만 호출 가능

# 

# 서버에서 캐릭터A에 대한 함수를 실행하면, 캐릭터 A 에 대한 소유권을 가지고 있는 Client 1에서 실행이 된다.

# Client 2 에서는 캐릭터 A 에 대해 실행을 할 수 없다.

# [ Blueprint ]

# ClientPrintString 이라는 커스텀 이벤트를 만든다.

# ServerRPC 에서는 리플리케이트를 '서버에서 실행' 으로 했다면 이번에는 ' 소유중인 클라이언트에서 실행'으로 설정한다.

# 

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# HasAuthority 의 bool 함수 그래프를 True 로 할 것이다.

# (화살표)ClientPrintString을 실행할 클라이언트는 Authority를 가지고 있다.

# 텍스트, 실내, 전자기기이(가) 표시된 사진 자동 생성된 설명

# 실행하는 주체는 Client일 것이다. 하지만 Server에서 실행한다는 디버그다.

# 클라이언트가 접속하는데 시간이 필요하기 때문에 바로 실행한다면 제대로 되지 않는다.

# 

# Client 접속을 위해 1초 딜레이를 주어서 함수를 실행한다.

# (센빠이랑 디코 대화한거 보고 어떤 bool 타입 함수 쓸지 고민해야댐

# Has Authority vs Is Sever)

# 

# Client 1에서 함수를 실행한다는 디버그다.

# 

# RPC앞의 Server,Client 의 의미는 해당 머신에서 실행하겠다는 의미

# 클라이언트 RPC = 클라이언트면 무조건 통과? but 서버에서 클라이언트rpc호출시 그 캐릭터를 가지고 있는 클라이언트에 가서 실행

# [ Multicast RPC ]

# Multicast 는 확성기다.

# 

# Server 에서 A 인스턴스를 가지고 있는 모든 Client에게 함수를 호출한다.

# 이러한 경우 Multicast 를 사용한다.

# [ Blueprint ]

# MulticastPrintString커스텀이벤트를 만든다.

# 

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 리플리케이트 – 멀티캐스트

# Client 로드 시간이 필요하기에 1초의 딜레이를 주고, Authority를 가진 엑터가 아닌 서버 호스트에서 실행시키기 위해 Is Server 라는 bool함수를 검사한다.

# 

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# A 캐릭터가 존재하는 Client 뿐만 아니라 Server 에서도 호출된다.

# 조금 더 구분을 주기 위해서 함수를 실행하는 캐릭터의 이름을 출력한다.

# PrintString 의 In String노드를 밖으로 끌어 Append 노드를 추가한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 빈 곳을 우클릭 하고 셀프 레퍼런스를 가져온다.

# 

# Self 오브젝트에서 노드를 이어 Get Display Name 함수를 생성한다.

# 

# Display Name 을 A 에 연결하고 B 에 MulticastRPC 라고 써준다.

# A 뒤에 B 가 이어서 호출된다.

# 

# 플레이하면 Client1 는 BP\_Character, Client2 는 BP\_Character1 에 빙의되어 함수를 실행한것을 알 수 있다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 클라이언트 rpc – 클라이언트에서실행됨

# 멀티캐스트 rpc – 클라이언트에서 실행하면 소유권없어도 실행 됨

# (Authority 없을때 출력하는 브렌치 사진)

# 테이블이(가) 표시된 사진 자동 생성된 설명

# 클라이언트 에서 클라이언트RPC를 호출하면 어떤 경우던 호출하는 클라이언트에서 실행된다.

# 빙의하고 있지 않은 캐릭터에서 서버 RPC를 호출하려 하면 드롭된다.

# 

# [ 신뢰성(Reliable)]

# 이벤트의 리플리케이트에 ‘신뢰성’ 이라는 항목이 있다.

# 

# 체크를 한다 – 신뢰성 RPC 함수로 사용한다.

# 비신뢰성 RPC 함수들은 원격 머신에서 실행이 안될 수 있다.

# (패킷이 드랍되거나 네트워크 대욕폭이 꽉차거나)

# 확실하게 RPC 함수가 실행되기 위해서는 Reliable체크를 해줘야한다.

# Ex) 신뢰성함수 : 클라이언트에서 호출된 RPC

# 비신뢰성함수 : 연출관련된 RPC들

# TCP/UDP 의 신뢰성 차이와 비슷한 개념이다.

# 02 : 09 : 55 ~ Replication Condition 할거면 넣자

# 2주차

# [ 에셋 추가하기 ]

# AnimStarterPack 과 FPS\_Assult\_Pack을 프로젝트에 추가한다.

# 에픽게임즈 런처의 좌측 카테고리의 언리얼 엔진 (화살표) 마켓플레이스에 ‘애니메이션 스타터 팩’ 을 검색한다.

# 

# 이미 보유중이여서 받는 버튼이 없다.

# 처음 받는다면 이런 식으로 되어 있을 것.

# ㅁ4.26 이후의 버전을 사용하고 있다면 ㅁ

# 모든 프로젝트를 표시를 체크하고, 4.26 버전을 선택하면 프로젝트에 추가 시킬 수 있다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# [ 캐릭터 뷰포트 ]

# 

# 

# BP\_Character의 Mesh::SkeletalMesh 를 SK\_Mannequin 으로 설정한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 캐릭터의 바라보는 방향과 위치가 이상하기 때문에 위치:: z : -88 회전::z : -90으로 변경한다.

# 오렌지이(가) 표시된 사진 자동 생성된 설명

# Camera 컴포넌트를 추가하고, 캐릭터 시야에 맞춰준다. 카메라의 이름은 FP(First Person)Camera로 변경해준다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명실내, 자동장치이(가) 표시된 사진 자동 생성된 설명

# FPCamera의 하위 객체로 SkeletalMesh 컴포넌트를 추가한다.

# SkeletalMesh 를 받은 SK\_Custom\_Arms로 설정한다.

# SK\_Custom\_Arms 는 FPS\_Assult\_Pack에 포함되어있다.

# 이름을 ArmMesh로 변경한 뒤, Mesh의 팔 위치에 맞춰 설정한다.

# 바닥, 자동장치이(가) 표시된 사진 자동 생성된 설명

# [ Set Visiblity ]

# Mesh 와 ArmMesh를 Ctrl + 드래그 하여 블루프린트에 Get 노드를 가져온다.

# 텍스트, 전자기기이(가) 표시된 사진 자동 생성된 설명

# 렌더링 관련 Set Visibility 함수노드를 생성한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# BeginPlay 이벤트 노드에 Is Locally Controlled함수를 Branch 분기문에 연결한다.

# 

# Is Locally Controlled 라면 1인칭 ArmMesh를 렌더시키고, 아니라면 3인칭 Mesh를 보이게 한다.

# 

# [ 총 스폰, 장착 ]

# Gun 클래스를 부모로하는 블루프린트 액터를 만든다.

# 

# BP\_M4A1로 이름을 변경한다.

# SkeletalMesh를 추가하고, SK\_Rifle 매시를 넣는다.

# 

# 

# 클래스에 스폰시킬 엑터의 클래스 타입. BP\_M4A1을 할당한다.

# Spawn Transform : 생성위치, Make Transform 노드를 연결한다.

# Collision Handling Override : Always Spawn Ignore Collisions = 콜리전을 무시한다.

# Owner : 생성시킬 총기 엑터의 주인인 캐릭터, self 레퍼런스를 할당한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 스폰 Location이 (0, 0, 0)이기에 월드의 (0, 0, 0)에 총기가 스폰되었다.

# 하늘, 실외, 무기, 활주로이(가) 표시된 사진 자동 생성된 설명

# [ 부모 설정 ]

# AttachActorToComponent함수 : Target Actor에 대한 정보를 Parent 의 자식으로 만든다.

# 1인칭 상태에서는 1인칭 Mesh의 자식, 3인칭 상태에서는 3인칭Mesh의 자식으로 넣어준다.

# Mesh의 Skeleton에 특정 Bone의 Socket을 넣을 수 있다.

# 애니메이션이 동작할 때 같이 움직이게 할 때, Socket을 생성, 참조한다.

# 

# Socket은 확성기 모양이다.

# 프리뷰 애니메이션을 실행 해보면 같이 움직이는 것을 확인할 수 있다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Rule :

# Keep Relative : 배치시키고 나서, LocalLocation을 유지시킨다.

# Keep World : Attach 시켜도 월드 위치에 계속 배치시킨다.

# Snap To Target : 부모에 스냅시켜버린다 (0, 0, 0) 값을 준다.

# 

# ArmMesh의 애님블루프린트는 이미 만들어져 있는 것이 있다.

# AnimClass 를 BP\_Rifle\_Arms\_AnimBP로 설정한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# [ 카메라 위치 수정]

# 사람, 무기이(가) 표시된 사진 자동 생성된 설명

# 부자연스러운 팔의 위치를 수정한다.

# FMCamera가 ArmMesh의 자식이 되게 반전시킨다.-

# 

# FMCamera의 Use Pawn Control Rotation 옵션을 해제한다.

# 

# BP\_Character(self) 컴포넌트의 Use Controller Rotation Pitch 옵션을 킨다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 자식 컴포넌트가 된 FMCamera의 위치를 수정한다.

# 

# 조금이나마 자연스러운 모습이 된다.

# 하늘, 실외이(가) 표시된 사진 자동 생성된 설명

# [ 소켓 추가하기 ]

# 3인칭 캐릭터의 Skeleton 으로 간다.

# 이미 작업을 해놓았기에 오른손 소켓에 총이 있지만, 초기 상태라면 총이 없다.

# 텍스트, 스크린샷, 전자기기, 디스플레이이(가) 표시된 사진 자동 생성된 설명

# 오른손에 총기를 붙이므로 hand\_r에 소켓을 추가한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 소켓의 이름을 ArmMesh에 있던것과 같이 R\_GunSocket으로 변경한다.

# 블루프린트를 연결한다.

# LocationRule, RotationRule = Snap to Target,

# ScaleRule = Keep Relative

# SetVisibility의 Propagate to Children은 그 자식객체까지 Visibility옵션을 적용시킬건지의 여부 체크이다. True로 해준다.

# 텍스트, 실내이(가) 표시된 사진 자동 생성된 설명

# 하늘, 사람, 실외, 무기이(가) 표시된 사진 자동 생성된 설명

# 3인칭 Mesh 캐릭터에 BP\_M4A1이 붙어서 보인다.

# 캐릭터를 위 아래로 마우스 회전하면 3인칭에서 이상하게 보인다.

# BP\_Character(self)에서 설정한 옵션으로인해 그렇다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 이 액터의 Rotation을 컨트롤러 Rotation의 Pitch값을 따라간다는 의미이다.

# 1인칭 에서는 맞는 옵션이지만 3인칭에서는 이러한 문제가 생긴다.

# 해결을 위해 Set Use Controller Rotation Pitch 노드에서 1인칭에서는 True,

# 3인칭에서는 False값이 되게 설정한다.

# BP\_Character(self)의 Use Controller Rotation Pitch 옵션은 꺼준다.

# 

# [ Gun 방향 고치기 ]

# 3인칭 Mesh의 R\_GunSocket을 우클릭하여 프리뷰 에셋(SK\_Rifle)을 추가한다.

# 텍스트, 스크린샷, 점수판이(가) 표시된 사진 자동 생성된 설명

# 프리퓨 애니메이션 탭에서 Idle\_Rifle\_Hip 시퀀스를 실행하게 한다.

# 텍스트, 탑재, 점수판이(가) 표시된 사진 자동 생성된 설명

# 우측 하단의 정지버튼을 누르고 SK\_Rifle의 위치를 맞춰준다.

# 하늘, 실외, 장난감, 자동장치이(가) 표시된 사진 자동 생성된 설명

# 저장하고 실행하여 확인한다.

# 3인칭 캐릭터가 Idle애니메이션을 하고있지 않다면 BP\_Character의 3인칭 Mesh AnimClass가 있는지 확인하고 애님블루프린트를 넣어준다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 사람, 무기이(가) 표시된 사진 자동 생성된 설명

# [ 발사 ]

# 프로젝트세팅 -> 입력에서 액션 매핑 Fire를 추가한다.

# 

# BaseCharacter.h

# 텍스트이(가) 표시된 사진 자동 생성된 설명 PressedFire(), ReleasedFire() 를 선언한다.

# BaseCharacter.cpp

# 

# BindAction함수를 이용해 바인드한다.

# [ Gun 가져오기 ]

# Gun의 타입을 가져오기 위해 TsubClassOf클래스를 사용한다.

# EditDefaultsOnly 지정어를 이용해 기본값을 수정할 수 있게 한다.

# BaseCharacter.h

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# BaseCharacter.h 에 Gun헤더파일을 포함시킨다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 

# TSubClassOf : 변수에 들어갈 수 있는 타입은 < > 안의 타입부터 타입이하의 클래스타입만 넣을 수 있다.

# Ex:

# 

# Gun 클래스부터 하위 클래스까지 담을수 있는 안정성을 보장하는 탬플릿클래스

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Actor 타입 WeaponActor를 선언한다. 블루프린트에서도 사용가능하도록 BlueprintReadWrite지정자를 쓴다.

# BeginPlay에서 SpawnActor함수를 이용해 생성한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# EquipGunType : 클래스타입

# &Ftransform::Identitiy : 위치속성0 회전속성0, 제로벡터

# SpawnParameters :

# 텍스트, 전자기기, 스크린샷, 컴퓨터이(가) 표시된 사진 자동 생성된 설명

# SpawnActor 노드의 파라미터와 같은 속성을 갖게 해준다.

# Super::BeginPlay()보다 앞서 실행되도록 한다.

# BP\_Character

# C++ 코드에서 역할을 대신하므로 SpawnActor노드를 삭제시킨다.

# 텍스트, 점수판, 검은색이(가) 표시된 사진 자동 생성된 설명

# BP\_Character(self) 컴포넌트의 디테일 패널에서 EquipGunType을 설정한다.

# 

# AttackACtorToComponent노드에서 타깃을 WeaponActor로 설정한다.

# 

# [ PressedFire ]

# Fire() 가상함수를 선언한다.

# Gun.h

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Fire() 함수가 실행된다면 Gun클래스가 가지고 있는 발사 몽타주와 1인칭 매시의 발사 몽타주를 재생할 것이다.

# OnPressedFire() 함수를 선언한다. BlueprintImplementatbleEvent지정어를 이용해 블루프린트 내에서만 정의할 수 있게한다.

# BaseCharacter.h

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 선언한 OnPressedFire()를 PressedFire()실행 시 호출한다.

# BaseCharacter.cpp

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 2021 09 19 16 47 34

# 00 30 13

# Tip: 클라이언트와 서버 둘다 쓰이는 함수에 주석으로 [Client+Server]라고 표시를 해놓으면 코드읽기가 편하다.

# 3주차

# 클라이언트에서 작동할 SetupWeapon 를 작성한다.

# 서버에서도 SetupWeapon 을호출하게 한다.

# OnSetupWeapon()은 BlueprintImplementableEvent UFUNCTION지정어로 선언되어 있어서 Blueprint에서 이벤트 호출이 가능하다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 서버에서 WeaponActor 설정이 되고 난 후, Super::BeginPlay(); 가 호출 되고 나서 Authority를 가지고 있을 때 SetupWeapon을 실행하게 한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 멀티 기준으로 SetupWeapon이 서버와 클라이언트에서 같이 실행된다.

# 1, 3인칭의 Visibility를 처리하는것도 WeaponActor의 동기화가 진행된 후에 진행되도록 한다. (WeaponActort가 동기화 되지 않고 실행이 되 오류가 나는 것을 방지한다)

# 동기화 전에 이미 호출을 했을수도 있으니까 is Valid 로 null체크해준다.

# 텍스트, 실내이(가) 표시된 사진 자동 생성된 설명

# BaseCharacter.cpp의 BeginPlay에서 SetupWeapon을 실행시킨다.

# SetupWeapon 함수는 HasAuthorirty를 가지고 있을 때 실행시킨다.

# 라인을 쏴서 데미지를 주는 작업을 서버에서 진행할 것이다.

# ServerFire도 블루프린트에서 작업할 수 있도록 OnServerFire 함수를 선언하고

# UFUNCTION지정어 BlueprintImplementableEvent를 지정한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Gun.cpp 의 ServerFire() 에서 OnServerFire() 함수가 실행되게 한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Gun.h

# 총알이 발사될 위치를 반환할 Fvector형 함수 GetMuzzleLocation을 선언한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# BlueprintNativeEvent : 별도 하이클래스에서 재정의 하지 않으면, cpp에서 정의한 함수가 호출된다.

# Gun.cpp의 GetMuzzleLocation 정의부에 \_Implementation을 추가로 붙여쓴다.

# 반환값으로는 GetActorLocation (GunActor의 WorldLocation좌표)를 준다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# \_Implementation은 함수 명에 추가되지 않고 기본 내장 구현을 사용하기 위해 작성한다.

# 컴파일하면 GetMuzzleLocation 함수를 Gun 클래스를 부모클래스로 만든 BP\_M4A1에서 오버라이드 할 수 있다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# M4A1의 액터 위치는 손잡이 부분으로 되어있다.

# 무기, 어두운이(가) 표시된 사진 자동 생성된 설명

# 컴포넌트의 SkeletalMesh의 SK\_Rifle 을 더블클릭하여 스켈레톤 트리를 확장한다.

# 

# 

# MuzzleSocket 이 MuzzleLocation이 될것이다.

# M4A1 SkeletalMesh를 타깃으로 하는 Get Socket Transform 함수를 생성한다.

# GetSocketTransform : 스켈레탈 매시에 있는 특정 소켓의 트랜스폼을 가져오는 함수

# TrasnformSpace 를 RTS World 로 지정해준다.

# Return Value를 우클릭하여 구조체 핀 분할. Location 을 반환시킨다.

# 텍스트, 전자기기, 스크린샷이(가) 표시된 사진 자동 생성된 설명

# Gun.h

# Fire() 가상함수 하단에 OnFire() 함수를선언한다.

# OnServerFire()와 같은 BlueprintImplementableEvent 를 선언한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# OnFire() 를 Gun.cpp 의 Fire() 함수 내부에서 호출시킨다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# [ MuzzleLocation ]

# Gun.cpp의 ServerFire함수의 내부구현

# Line Trace를 이용해 MuzzleSocket 위치에서 정면으로 Ray를 쏴 피격 처리를 한다.

# 실제 발사 시작할 위치인 StartFireLocation 변수를 FVector형으로 선언한다.

# 끝 위치인 EndFireLocation변수를 선언한다.

# 발사 방향 벡터로 사용할 FireDirection변수를 선언한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 조건문 : Owner가 빈 값이 아닌지, 즉 Gun을 가지고 있는 BaseCharacter가 비어있지 않을 때 실행한다. (BaseCharacter.cpp 의 BeginPlay에서 파라미터로 Owner를 넣었다.

# EveLocation : 현재 Eye(BaseCharacter클래스의 경우 ActorLocation + Eye) 위치

# EyeRotation : 현재 Eye각도

# GetActorEyesViewPoint() : 엑터의 눈의 위치와 바라보고 있는 방향 정보를 가지고 LineTrace 시작지점과 끝지점을 정해준다. 파라미터에 EyeLocation(Eye위치), EyeRotation(Eye각도)가 들어간다.

# EyeLocation을 LineTrace시작위치로 설정.

# EyeRotation은 Rotator형이기 때문에 Vector로 변환시켜 FireDirection에 넣는다.

# 

# 끝점 = 시작점 + 발사 방향 벡터 \* 사정거리

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# Hit : LineTrace를 맞은 액터

# QueryParams : 설명필요!

# LineTrace할 때 발사하는 총 엑터와 총의 오너(캐릭터)를 제외시킨다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# ㅁLineTraceSingleByObjectType : 시작점과 끝점을 기준으로 검사를 하는데 Pawn 이라는 ColllisionChannel을 가지고 있는 액터를 대상으로 진행한다.

# (위에서 정의한 QueryParams으로 인해 총과 자신은 검사에서 제외된다)

# ㅁDrawDebugLine, DrawDebugSphere : LineTrace를 보이게 하고, LineTrace검사를 하며 걸린 엑터가 있다면 그 지점에 Spere를 출력한다.

# 맞은 대상이 있다면 빨간색 Line이, 없다면 파란색 Line이 출력된다.

# DrawDebug를 사용하기 위해 DrawDebugHelpers 헤더파일을 포함시킨다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 캐릭터의 눈의 위치가 이상하게 설정되어 있을 수 있다.

# BP\_Character의 FPCamera의 Z 위치값을 복사해서

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# BP\_Character(self) 디테일 패널의BaseEyeHeight에 붙여넣는다.

# 

# 에디터에서 플레이 중 ~ 기호를 누르면 콘솔 명령어를 입력가능하다.

# Show Collision 을 사용해 콜리전 범위를 확인할 수 있다.

# 텍스트, 하늘이(가) 표시된 사진 자동 생성된 설명

# ㅁBaseCharacter를 Actor로 캐스팅(형변환)한다.

# ㅁ형변환이 잘 되었다면 nullptr값이 아닐 것이다.

# ㅁUnrealEngine에서 지원하는 DamageEvent가 있다.

# Gun.h에 한 발의 데미지인 Int형 DamagePerBullet을 정의한다.

# 에디터에서 기본 데미지를 설정할 것이다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# ㅁPointDamage : 총 같은 한 점에 이벤트 발생

# ㅁRadialDamage: 데미지가 원형으로 방사형으로 이벤트 발생

# PointDamage를 사용한다.

# ㅁDamageEvent

# TypeClass 는 nullptr,

# Damage는 위에서 만든 DamagePerBullet 만큼,

# HitInfo : LineTrace수행에서 나온 HitResult,

# ShotDirection : 발사 방향, LineTrace의 발사 방향과 같다.

# ㅁBaseCharacter에서 공격을 받으므로 TakeDamage를 호출한다.

# TakeDamage(데미지, 데미지이벤트, \*선동자(데미지를 준 주최자)컨트롤러, 데미지캐서 (실질적으로 데미지를 줄 캐릭터) );

# [ Die ]

# 일정 데미지를 받으면 죽게 하기 위해 함수를 만든다.

# BaseCharacter.h 에 선언한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명 public:

# Die는 클라이언트에서 실행되고, OnDie는 서버에서도 호출될 것이다.

# BaseCharacter.cpp 에 정의한다. Die() 가 실행되면 OnDie()가 호출된다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# [ 체력 시스템 ]

# 데미지를 주기 위해 체력 시스템을 만든다.

# BaseCharacter.h 에 MaxHealth, CurrentHealth를 선언한다.

# Blueprint에서도 사용할 수 있도록 BlueprintReadWrite로 지정한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# BaseCharacter.cpp 의 생성자에서 체력의 초기값 설정을 한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# [ TakeDamage ]

# TakeDamage를 BaseCharcter.h에 override선언한다.

# public 단에 놔줘야 한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 

# BaseCharacter.cpp

# TakeDamage정의를 완성한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 현재 체력을 감소시키고, 0 이하일 시 Die() 함수를 실행한다.

# [ 캐릭터 상/하 카메라 회전 고치기 ]

# 카메라가 원점을 기준으로 회전하는 것이 아닌, 카메라 위치에서 회전을 하게 한다.

# BaseCharacter.cpp

# BeginPlay에 있는 IsLocallyControlled 조건문을 주석처리한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# BP\_Character의 FPCamera의 UsePawnControlRotation옵션을 켜준다.

# 

# 회전이 자연스러워진 것을 확인할 수 있다.

# 사람, 무기이(가) 표시된 사진 자동 생성된 설명

# 4주차

# [ 애니메이션 ]

# BlueprintImplementableEvent UFUNCTION 지정어를 이용해 블루프린트에서 사용 가능하게(?) 해놓은 함수들이 있다.

# BaseCharacter의 OnPresssedFire, Gun의OnFire

# [ Gun 작업 ]

# M4A1의 블루프린트의 빈 공간에서 OnFire이벤트를 생성한다.

# 

# 몽타주란.

# BP나 C++ 에서 PlayMonstage 를 이용해 몽타주를 넘겨주면 몽타주가 가지고있는 슬롯에 해당하는 에님블루프린트를 재생한다.

# OnFire 호출 시. M4A1의 ~ 몽타주를 실행할 것이다.

# [ 블루프린트 ]

# 몽타주를 재생시킬 SkeletalMesh 를 가져와 PlayMontage 노드를 실행시킨다.

# 텍스트, 전자기기, 스크린샷이(가) 표시된 사진 자동 생성된 설명

# 재생시킬 몽타주는 ‘A\_Rifle\_Shoot01’

# 재생속도, 시작위치, 시작시간을 지정해줄 수 있다.

# M4A1의 SkeletalMesh 의 AnimClass 설정이 되어있지 않다.

# BP\_Rifle\_Skeleton\_AnimBP AnimClass로 설정한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 실행시켜 OnFIre이벤트를 발생시킨다.(마우스 좌클릭)

# (쏘는 사진)

# 탄피가 나오고 총구섬광이 생긴다.

# OnFire 이벤트가 발사를 한 로컬에서만 호출이 이러나기 때문에 다른 캐릭터의 시점에서는 애니메이션이 실행되지 않는다.

# 자신 클라이언트 뿐만 아니라 다른 클라이언트 에게서도 발생하도록 한다.

# [ C++ ]

# Multicast RPC는 서버와 모든 클라이언트에서 실행한다. 이를 이용해 다른 클라이언트에서도 나타나게 한다.

# OnServerFIre 함수 아래에 작성해준다.

# 텍스트, 오렌지, 어두운이(가) 표시된 사진 자동 생성된 설명

# MulticastRPC를 사용하고, 신뢰성(Reliable)있는 함수를 선언한다.

# 

# 정의할때는 GetMuzzleLocation처럼 \_Implementation을 함수명 뒤에 붙여 정의한다

# .

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# (함수 내부의 의미 적어야댐)

# MulticastFire함수는 서버에서도 호출할 것이다.

# ServerFire 함수에서 호출시킨다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# [ 정리 ]

# Client1에서 Fire() 를 호출한다.

# Authority가 없는경우 서버에서 ServerFire() 가 호출된다.

# ServerFire()에서 MulticastFire()를 호출시켜 모든 클라이언트에게 MulticastFire()를 실행시킨다.

# MulticastFire()는 동기화 되는 타이밍에 서버에서도 실행된다.

# 

# Client1의 경우 Fire이 호출되고, MulticastFire에서 또다시 호출되는 문제점이 있다.

# 이런 경우를 막기 위해 MulticastFire()함수 내부에 조건문을 걸어놓은 것.

# 클라이언트에서 ServerRPC를 실행하면 드롭된다.

# A\_Riffle\_Shoot01 애님노드에 이어서 3인칭 캐릭터 Mesh 의 OnFire() 함수 호출시 애니메이션 몽타주도 실행한다.

# 3인칭 캐릭터는 기본 몽타주가 없기 때문에 새로 만든다.

# 몽타주를 만드는 법 :

# 몽타주로 지정할 애님시퀀스를 찾고, 애님시퀀스를 우클릭 하고 -> Create(생성) -> CreateAnimMontage(애님몽타주 생성) 하여 생성할 수 있다.

# 콘텐츠 브라우의 빈 공간을 우크릭 후 애니메이션 카테고리의 애니메이션 몽타주로 생성한다. 몽타주로 생성할 SkeletalMesh를 선택하고 사용할 시퀀스를 드래그해서 넣어준다.

# 텍스트, 표지판, 스크린샷, 액자이(가) 표시된 사진 자동 생성된 설명

# Fire\_Rifle\_Hip이라는 애니메이션시퀀스를 몽타주로 만든다.

# 텍스트, 스크린샷, 전자기기, 화면이(가) 표시된 사진 자동 생성된 설명

# Fire\_Rifle\_Hip\_Montage

# 텍스트, 실내, 스크린샷이(가) 표시된 사진 자동 생성된 설명

# BP\_Character의 3인칭Mesh를 타깃으로 하는 Fire\_Rifle\_Hip\_Montage를 실행한다.

# 이어서 BP\_Character의 1인칭 Mesh(ArmMesh)를 타깃으로 하는 A\_Arms\_Rifle\_Shoot01\_Montage를 실행한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 1인칭 화면에서는 Gun과 Arms의 애니메이션이 잘 보이지만, 다른 클라이언트의 시점에서는 Arms의 애니메이션은 보이지 않는다.

# Fire\_Rifle\_Hip\_Montage의 블렌드 옵션의

# Blend In Time을 0.0으로, Bleend Out Time을 0.05 로 설정한다.

# Fire 몽타주의 재생시간이 0.22초인데, 기본Blend Time이 그것보다 길다.

# 텍스트, 모니터, 스크린샷, 텔레비전이(가) 표시된 사진 자동 생성된 설명

# [ 애니메이션 ]

# 텍스트, 실내, 검은색, 장치이(가) 표시된 사진 자동 생성된 설명

# 3인칭 캐릭터 매쉬의 애님블루프린트

# DefaultSlot을 재생시키면, Source에 대한 애니메이션은 무시된다.

# 만약 걸으면서 총을 쏜다면 상체에서는 HipMontage를 하체에서는 걷는 애니메이션을 실행시킬 것이다.

# Fire\_Rifle\_Hip\_Montage 의 슬롯 편집 카테고리에서 슬롯 매니저를 추가한다.

# 

# UpperBody 라는 새 슬롯을 만든다.

# 

# DefaultSlot의 이름을 UpperBody로 변경한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 3인칭 캐릭터의 애님블루프린트의 DefaultSlot의 이름을 UpperBody로 한다.

# 텍스트, 실내, 디스플레이, 스크린샷이(가) 표시된 사진 자동 생성된 설명

# 저장을 해야 변경내용이 적용된다.

# 하늘, 사람, 실외이(가) 표시된 사진 자동 생성된 설명

# 다른 클라이언트에서도 잘 보인다.

# [ 레이어 블렌딩 ]

# ‘본마다 레이어로 블렌딩합니다. ‘ 노드를 애님블루프린트에 추가한다.

# 텍스트, 전자기기이(가) 표시된 사진 자동 생성된 설명

# 레이어 블렌딩 노드의 디테일 패널의 Layer Setup카테고리. BranchFilters의 BoneName을 spine\_02로 한다.

# 

# 3인칭 캐릭터의 Skeleton을 보면 spine\_02가 어느 위치인지 알 수 있다.

# 텍스트, 스크린샷, 다른이(가) 표시된 사진 자동 생성된 설명

# 현재 재생되고 있는 BasePose에 BoneName 아래로는 BlendPoses0에 해당하는 슬롯을 재생한다는 의미.

# 

# Locomotion 애니메이션에 해당하는 캐시 포즈를 저장할 수 있다.

# 

# ‘캐시 포즈 새로 저장’ 노드를 사용한다.

# 

# 저장한 캐시 포즈를 사용할 수 있다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# UpperBody의 Source Input에도 캐시 포즈 사용을 연결해준다.

# 슬롯의 애니메이션이 실행이 안될때는 Source에 해당하는 애니메이션이 Output으로 나온다.

# 

# (기울이면서 총 쏘기)

# 10월 2일차영상 1 : 01 : 21부터

# [ Die Animation ]

# 죽게된다면 OnDie 이벤트를 실행한다. – BaseCharacter.cpp

# 바로 Destory 되지 않게 하기 위해 Destory() 함수 실행을 주석처리한다.

# 

# 죽었을 경우 Physics 라는 것을 쓸 것이다.

# SkeletalMesh 탭의 Physics 탭에 가보면 피직스 모델이 있다.

# 시뮬레이트를 누르면 시뮬레이션 된다.

# 

# 이 피직스 모드를 키려면 3인칭 캐릭터 Mesh를 가져오고 SetSimulate Physics의 Simulate를 True로 만든다.

# 텍스트, 전자기기이(가) 표시된 사진 자동 생성된 설명

# [ Listen Server를 고려한 코드 수정 ]

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 조건문의 HsAuthority를 IsLocallyControlled 로 바꿔줄것이다.

# GetOwner가 엑터이기 때문에 폰으로 캐스팅해줘야한다.

# Apawn\* Pawn = Cast<Apawn>(GetOwner());

# ->Owner를 Pawn 타입으로 캐스팅 하겠다.

# 

# 이렇게 하면 ListenServer로 돌려도 Fire() 함수가 호출된다.

# 3인칭캐릭터 Mesh 의 CollisionEnalbed 가 Query Only 로 되어있다.

# 

# OnDie 이벤트 발생 시.

# Mesh 에서 노드를 이어 SetCollisionEnabled 함수를 생성한다.

# CollisionType을 CollisionEnabled로 설정한다.

# 

# Mesh 노드에서 PauseAnim 을 가져와 노드를 잇는다.

# 

# 죽게 되면 해당 폰에서 UnPossess 되게 한다.

# GetController 함수를 만들어서 ReturnValue에서 UnProsses 노드를 만든다.

# 텍스트, 스크린샷, 전자기기, 디스플레이이(가) 표시된 사진 자동 생성된 설명

# ~~OnDie이벤트에 연결한다.~~

# 제외시키고 하는 것이 테스트에 좋다.

# 텍스트, 전자기기이(가) 표시된 사진 자동 생성된 설명

# RPC : 특정 상황, 단발성 이벤트를 위한 동기화 기법

# 변수 동기화(Replicated) : 지속성 있는 동기화가 필요할 때

# [ C++]

# BaseCharacter.h

# CurrnetHealth 함수 아래에 bool타입 IsDead 함수를 선언한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# BlueprintReadOnly, Replicated UPROPERTY 지정어를 준다.

# BaseCharacter.cpp

# IsDead 함수도 Replicated 목록에 추가한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 체력이 0 이 되었을 때 IsDead를 true로 만든다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# OnDie를 클라이언트에서도 호출하려고 한다.

# 변수가 동기화 되었을 때 Notify를 실행한다(?)

# ReplicatedUsing을 사용한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# IsDead가 동기화 될 때 OnRep\_IsDead()가 실행된다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 3인칭 캐릭터Mesh를 OnDie이벤트 발생시 항상 보이게 한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# 1인칭 캐릭터 ArmMesh는 보이지 않게 한다.

# 텍스트, 전자기기이(가) 표시된 사진 자동 생성된 설명

# 테스트하여 확인한다.

# 사람, 무기이(가) 표시된 사진 자동 생성된 설명