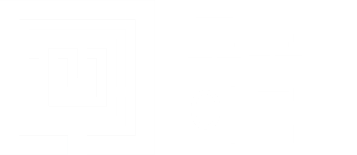
|  |
| --- |
| VR 프로그래밍 심화과정  with LiiYuu Senpai |
|  |
| 2021.09 ~ 2021.11  게임과 3학년  작성자: 함승호 |



# 1주차

# 강사소개, 수업계획, 입력매핑, 서버(이론)

# 2주차

# 냐옹

# 3주차

# 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

# 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 Gamemode들을 BaseGameMode 로 설정한다.

# 

# 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.

# #(비)신뢰성(.Reliable)

# 비신뢰성 RPC 함수들은 원격 머신에서 실행이 안될 수 있다.

# (패킷이 드랍되거나 네트워크 대욕폭이 꽉차거나)

# 확실하게 RPC 함수가 실행되기 위해서는 Reliable체크를 해줘야한다.

# (에디터에서 체크박스 사진)

# TCP/UDP 의 신뢰성 차이와 비슷한 개념이다.

# [ 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 ]

# 1: 35:00 언저리

# ClientPrintString 이라는 커스텀 이벤트를 만든다.

# ServerRPC 에서는 리플리케이트를 '서버에서 실행' 으로 했다면 이번에는 ' ~ ' 로 설정한다.

# (이벤트 사진)

# (이벤트 상세 설정 사진)

# BeginPlay 이벤트의 실행 함수를 ClientPrintString 으로 하고 실행하더라도 서버에서 실행한다고 할것이다. 클라이언트가 접속하는데 시간이 필요하기 때문에 바로 실행한다면 제대로 되지 않는다.

# (Server:Client 호출 사진)

# Client 접속을 위해 1초 딜레이를 주어서 함수를 실행한다.

# 그리고 딜레이 이후에 Is Server라는 함수를 Branch 앞에둔다 ( 수정 )

# (고친 블루프린트 사진)

# RPC 앞의 Server,Client 의 의미는 해당 머신에서 실행하겠다는 의미

# 2주차

# Gun.h 의 GunCalss 내부의 public 범위지정자? 내부에 Fire 가상함수를 선언한다.

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

# 3주차

# 냐옹

# 클라이언트에서 작동할 SetupWeapon 를 작성한다.

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

# 이렇게 한다면 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

# UFUNCTION(BlueprintImplementableEvent) 를 선언한다.

# (선언사진)

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

# (OnserverFire 실행사진)

# 총알이 발사될 위치를 반환할 Fvector형 const 함수 GetMuzzleLocation을 선언한다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

# (Gun.h ->AGun클래스 내부의 public: )

# BlueprintNativeEvent지정어 : 따른 하이클래스에서 재정의 하지 않으면, cpp에서 정의한 함수가 호출된다.

# Gun.cpp의 GetMuzzleLocation 정의부에 \_Implementation을 추가로 붙여쓴다.

# 텍스트이(가) 표시된 사진 자동 생성된 설명

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

# 반환값으로 현재 액터 위치를 반환시킨다.

# 컴파일하면 GetMuzzleLocation 함수를 BP\_M4A1에서 오버라이드 할 수 있다.

# 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