Readied Bullet

게 임 공 학 과

2013182020
2013182021

박 인 혁
박 장 호

Contents

Readied Bullet

장르 : 진행형 슈팅 액션

시점 : 3인칭

기술 스택 : Unreal, C++, IOCP server

개발 기간 : 2020.06~2021.01 (약 7개월)

GitHub : https://github.com/jangho-park-dev/MyReadiedBullet

Youtube : https://youtu.be/PBkXwB52L8w

게임 화면 예시

플레이어의 주 장비는 총이 아닌 총알

총알의 변화로 게임의 특색을 살림

플레이어가 직접 만드는 총알

플레이어가 직접 자신이 쏠 총알을 디자인

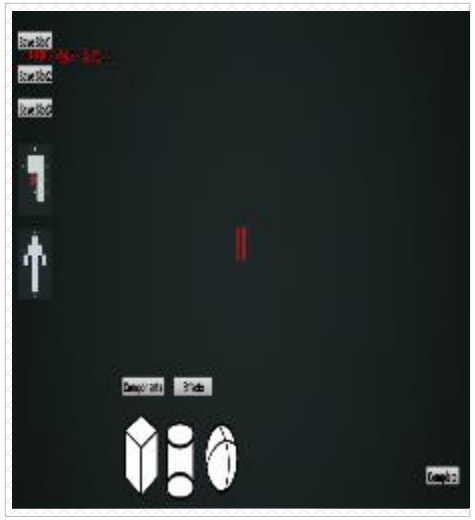
|특징|

최대 6인 멀티플레이

최대 6명까지 멀티플레이가 가능
멀티플레이시 3:3 대전

총알의 모양과 힘을 주는 방향에 따라 다른 궤적

플레이어가 직접 디자인한 총알은 모양과 힘을 주는 방향에 따라
다르게 날아간다. (같은 모양, 같은 힘이라면 동일한 궤적)



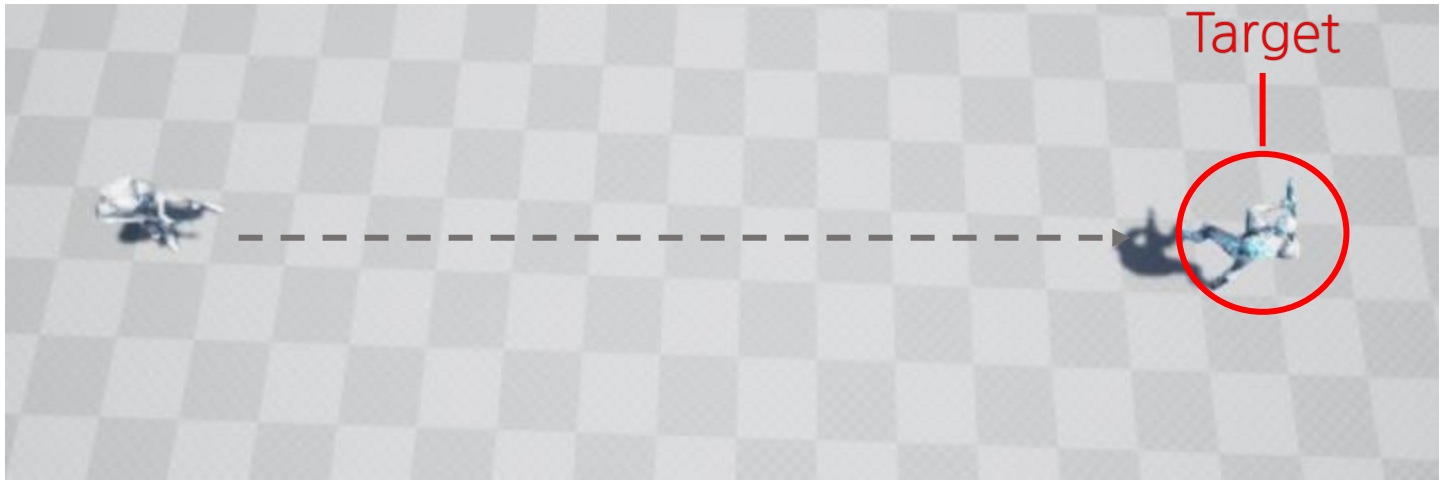
자신이 사용할 **총알**을
커스터마이즈 한다

끊임 없이 나오는 적들을 헤치며
열쇠를 찾는다.

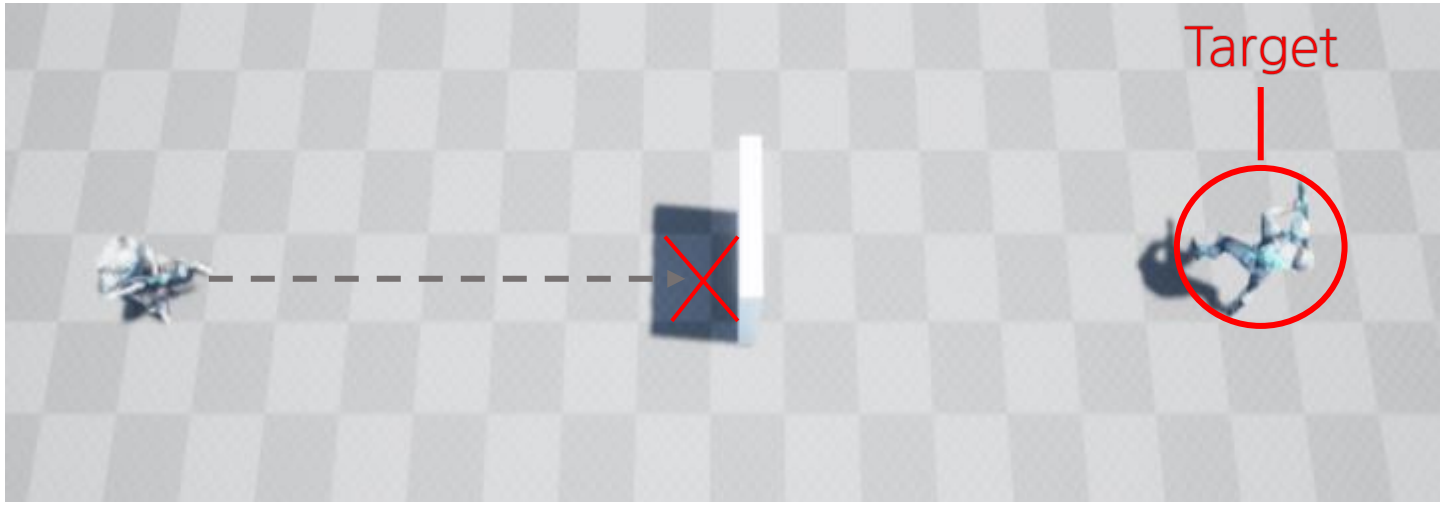
찾은 열쇠로 보스 방 문을 열고
보스를 잡는다.

다음 스테이지로 이동 및
총알 커스터마이즈

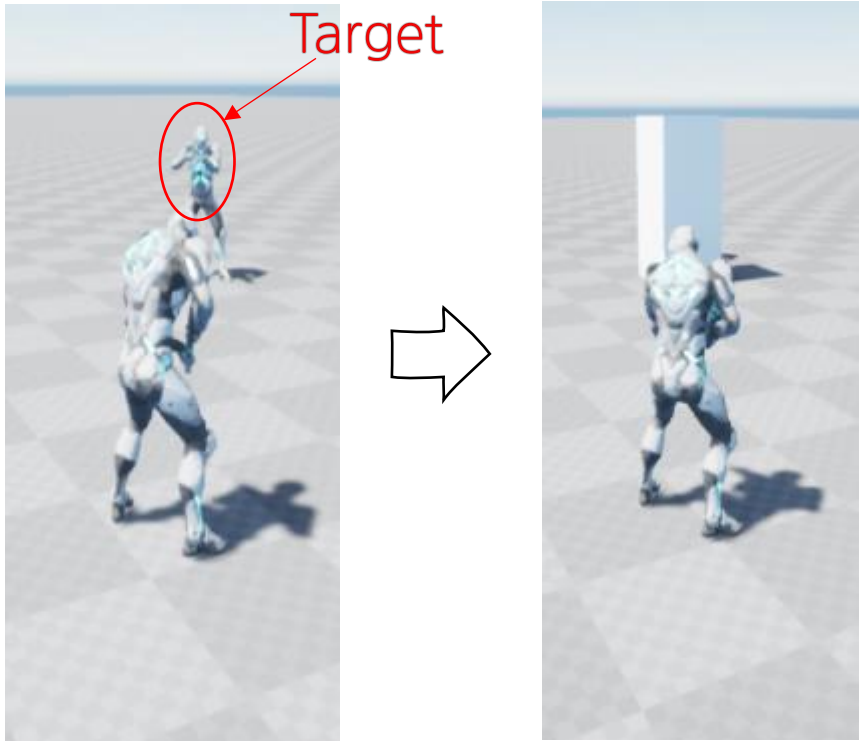
	박인혁	박장호	비고
리소스 수집	플레이어, 자연 맵, 무기, 파티클	적 NPC, 도시 맵, 유적 맵	
게임 로직	총알 커스터마이즈, 모양에 따른 총알 궤적 계산, 충돌처리	캐릭터 애니메이션 BP, 적 NPC 애니메이션 BP, 적 NPC AI (비헤이비어 트리)	
맵 제작 및 레벨 디자인	X	도시, 유적, 자연 맵 제작 및 레벨 디자인	
UI	커스터마이즈 UI, 캐릭터 UI (Hp, Ammo), NPC UI (HP)	X	
서버	X	전반적인 패킷 통신	
사운드	O	X	



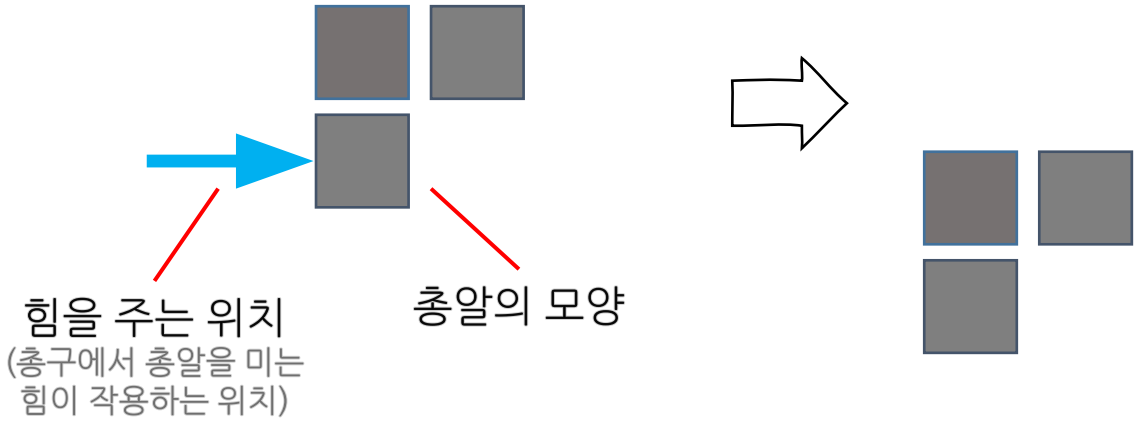
쉽게 표적을 맞출 수 있음

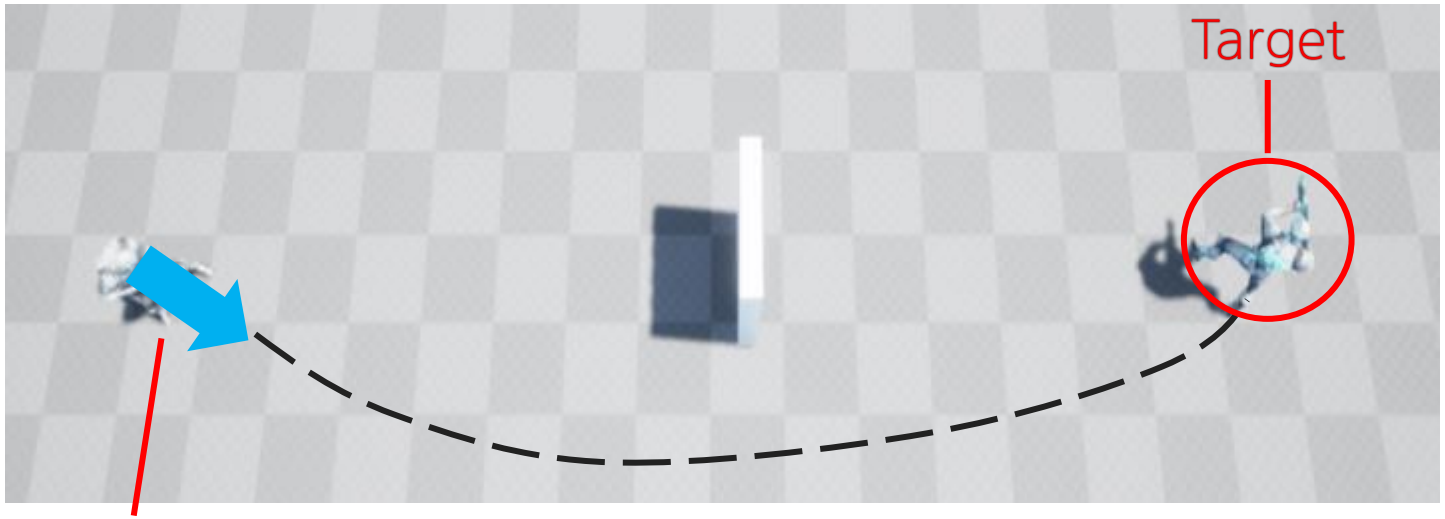


일반적으로 맞추는 것 불가능



은폐, 엄폐한 적을
맞출 수 있는
총알을 만들자!

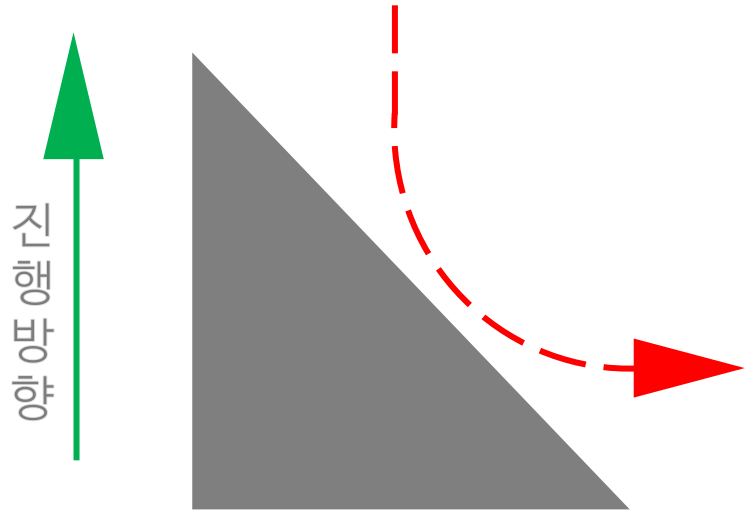




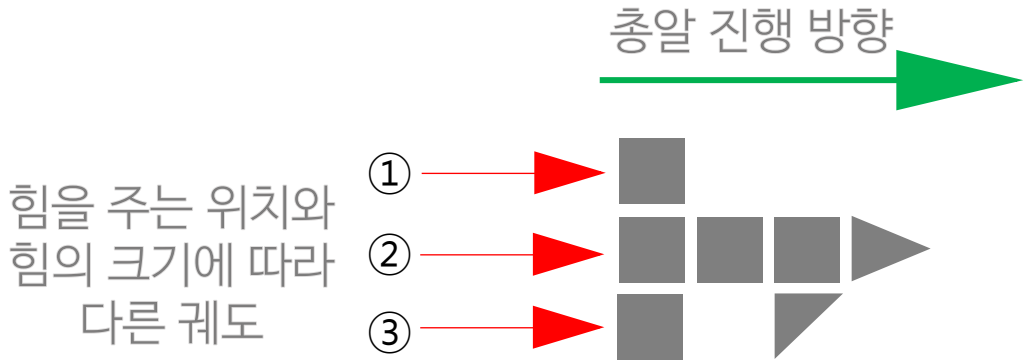
동일한 모양, 동일한 힘,
동일한 힘의 위치라면
항상 같은 궤적을 그리며 날아간다.

360도 회전시켜가며 원하는 디자인의 모양을 이어 붙인다.

총알 커스터마이즈 화면 예시



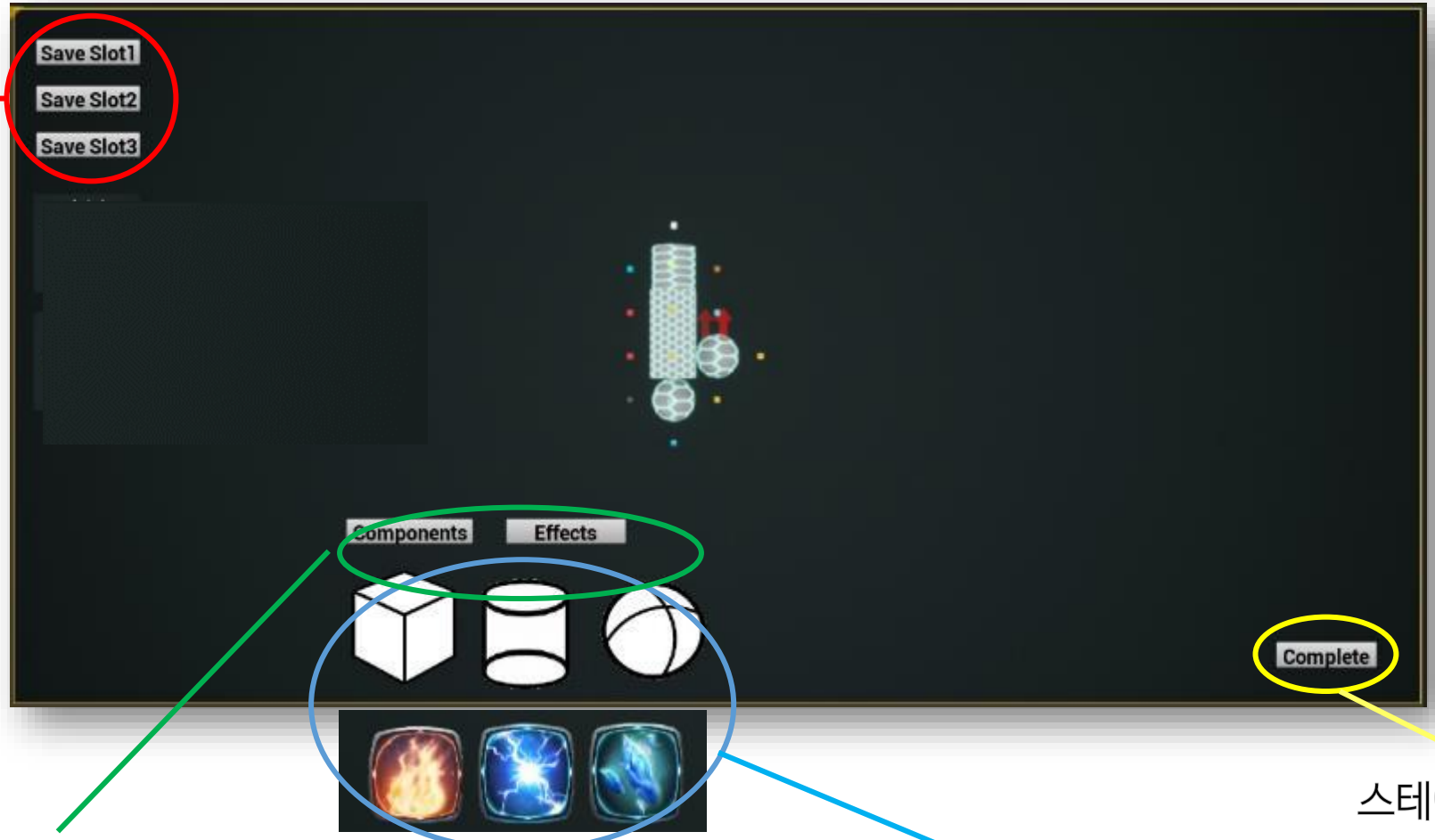
표면에 따른 속도와 궤도 변화



총알의 모양에 따라 다르게 날아가는 총알

(동일한 모양, 동일한 힘의 위치, 동일한 힘 = 같은 궤적)

만든 총알을
세이브 할 수 있는
세이브 슬롯 버튼



스테이지로 이동

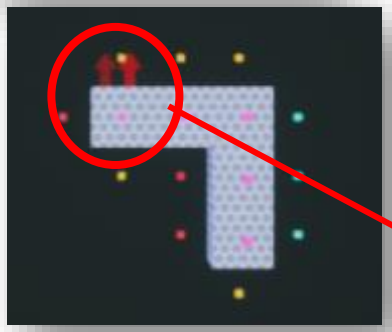
Components 버튼 - 총알을 만들 부품에 모양 탭으로 전환
Effects 버튼 - 총알에 넣을 효과 탭으로 전환

Components 버튼과 Effects 버튼에 따른 변화 탭

커스터마이즈 한 총알 모양

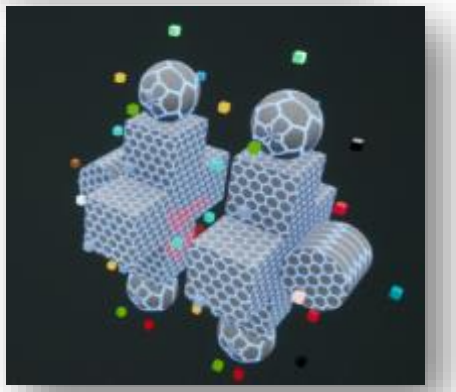
커스터마이즈 된 총알 그대로 날아가는 모습

화살표 모양



부메랑 모양
힘을 주는 위치에 따라
회전이 생기고 그에 따라
오른쪽으로 휘어져서 날아감

GIF 파일

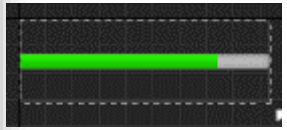


아무리 복잡해도 좌우상하
대칭이라면 일직선으로 날아감

애니메이션 블루프린트 제작
- Aim Offset
- Reload
- Idle, Run, Fire, Zoom Crouch 등 상태 전환

사운드
- 격발, 재장전
  No Ammo 상태 격발
- 총알 Effect에 따라 다른 효과음

플레이어 UI
- HP, Ammo



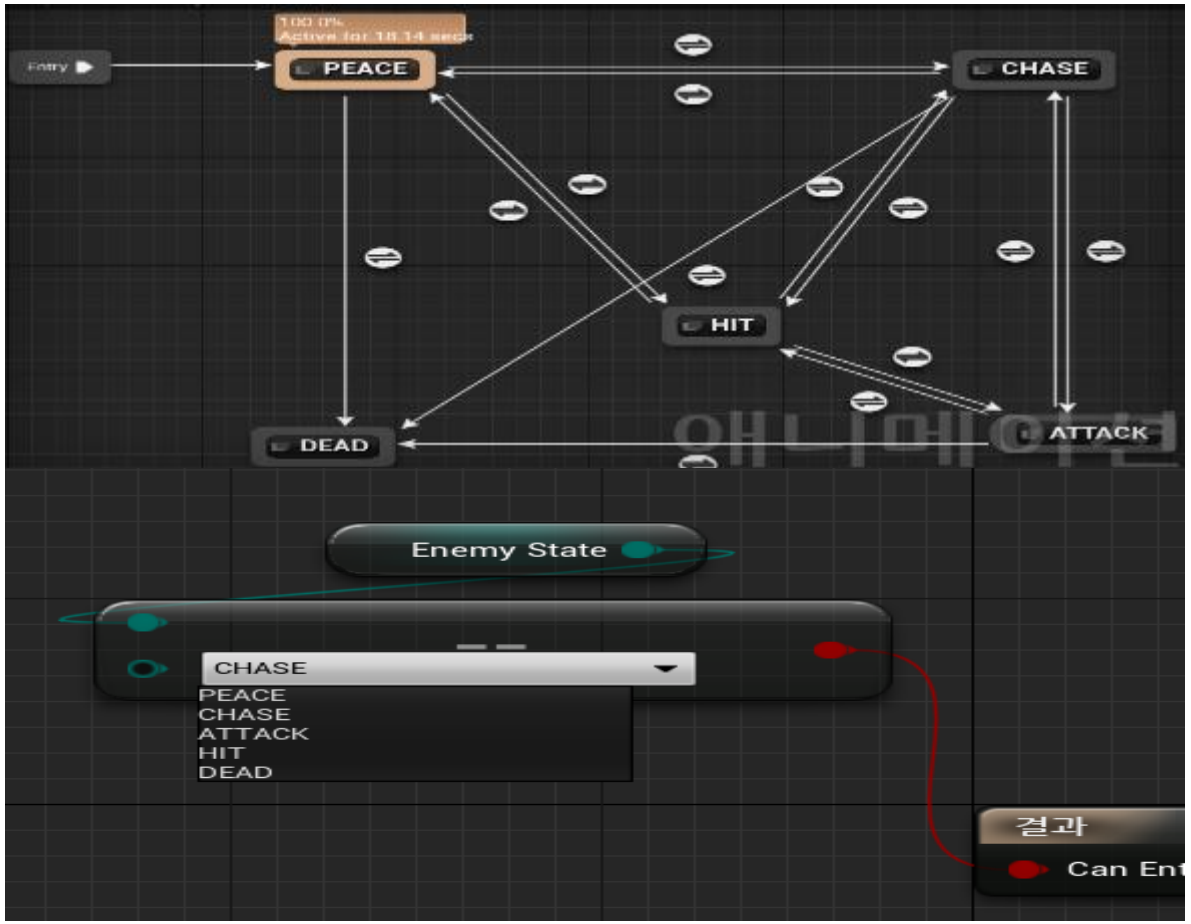
Griffon

Dragon

Wolf

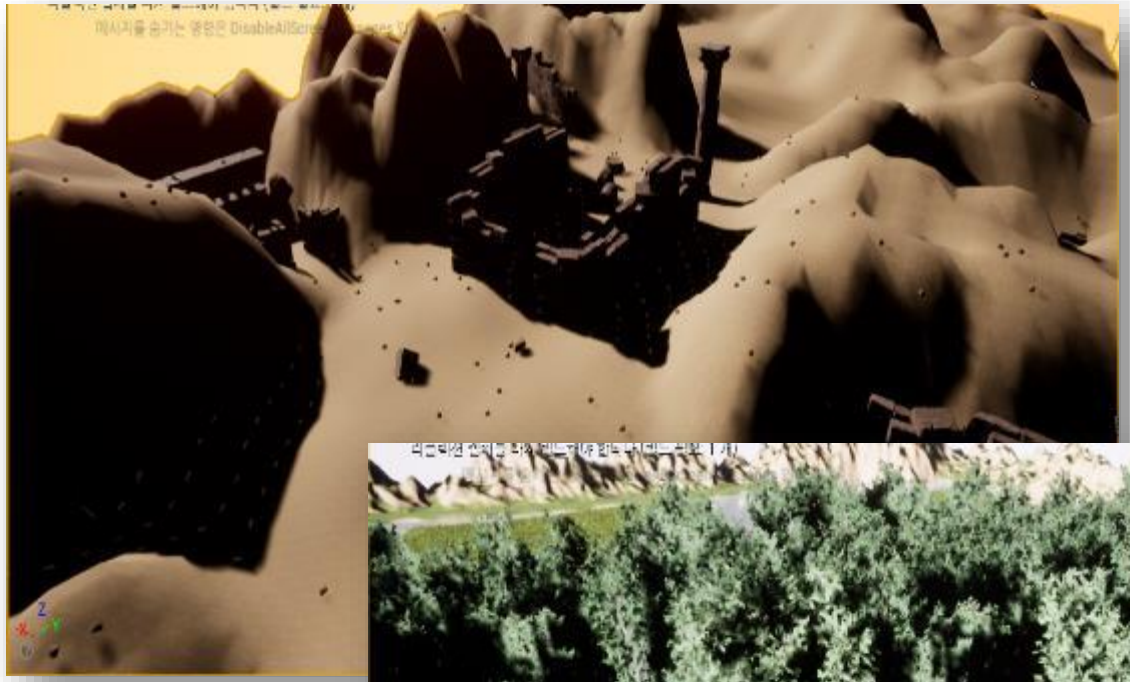
Centaur

- 몬스터 4마리 구현
- 각 맵의 테마에 맞게 배치
- AI Blueprint → C++ Code
- 특정 몬스터 소켓에 장비 착용
- 애니메이션 상태 머신화
- 행동 트리 제작



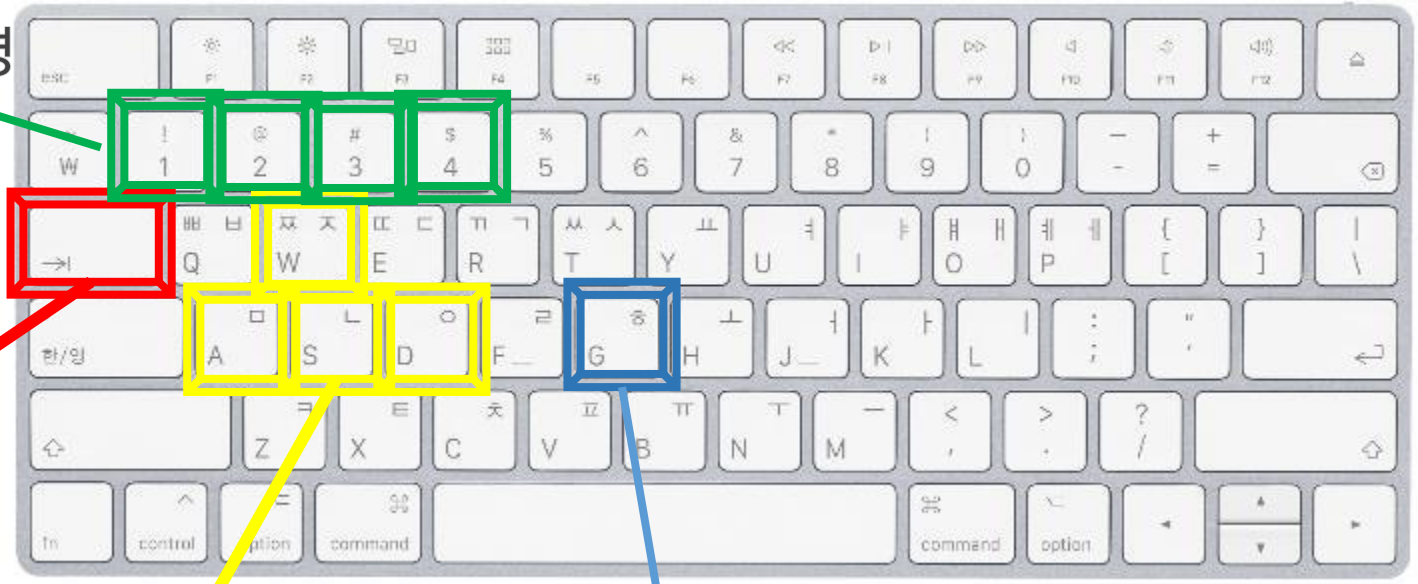
상태 머신

상태	내용
PEACE	Idle 상태(idle 애니메이션)에서 일정 쿨 타임마다 Walk상태로 전환함. Walk 상태는 정찰. 걸어 다니는 애니메이션을 취하며 정찰함.
CHASE	적 캐릭터의 상태가 CHASE로 바뀌면 플레이어를 따라 달림.
ATTACK	적 캐릭터의 상태가 ATTACK으로 바뀌면 플레이어 앞에서 공격함.
HIT	적 캐릭터의 상태가 HIT로 바뀌면 HIT 애니메이션을 취함.
DEAD	적 캐릭터의 체력이 0이 되면 DEAD 애니메이션을 취함.



테마별 3개의 스테이지
(자연, 유적, 도시)

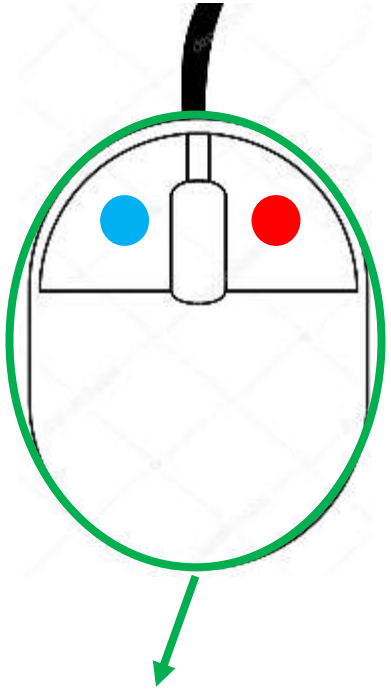
숫자 키 : 총알 변경



Tab : 인벤토리

WASD : 이동

G : 폭탄



Left Button : 공격

Right Button : 조준

마우스 이동 : 방향 전환

```cpp
void ARBNetwork::InitClientSocket()
{
    WSADATA wsa;

    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
        return;

    m_ClientSocket = WSASocket(AF_INET, SOCK_STREAM,
        0, NULL, 0, WSA_FLAG_OVERLAPPED);

    if (INVALID_SOCKET == m_ClientSocket)
        return;

    m_WSARecvBuf.buf = m_RecvBuf;
    m_WSARecvBuf.len = MAX_BUFFER;
    m_WSASendBuf.buf = m_SendBuf;
}
```

```cpp
void ARBNetwork::Connect()//const char* serverIP)
{
    SOCKADDR_IN serveraddr;
    ZeroMemory(&serveraddr, sizeof(SOCKADDR_IN));

    serveraddr.sin_family = AF_INET;
    serveraddr.sin_port = htons(SERVER_PORT);
    serveraddr.sin_addr.s_addr = inet_addr("127.0.0.1");//(serverIP);

    int retval = connect(m_ClientSocket, (SOCKADDR*)&serveraddr, sizeof(serveraddr));
    UE_LOG(LogTemp, Error, TEXT("%d"), retval);
    if (retval == SOCKET_ERROR)
    {
        return;
    }
    m_IsRun = true;

    u_long sock_on = 1;
    retval =ioctlsocket(m_ClientSocket, FIONBIO, &sock_on);

    return;
}
```

클라이언트에서는 먼저 소켓을 초기화하고 서버 정보를 입력해 접속하도록 합니다.

```
struct SOCKETINFO
{
    //WSAOVERLAPPED m_RecvOverlapped; //1개만써야함 ,  recv는1개지만 send용Overlapped는 그때그때 new로 할당해주어서 사용 콜백이불리면 delete
    //WSABUF m_WSASendBuf; //다중 송신을위해서는 여러개 써야함.     위와마찬가지로 new로 할당, 콜백이불리면 delete
    //WSABUF m_WSARecvBuf; // 1개만써야함
    SOCKET m_Socket;
    OverlappedEx* m_SendOverlappedEx;
    OverlappedEx* m_RecvOverlappedEx;

    //char m_SendBuf[MAXBUFFER]; //다중 송신을위해서는 여러개 써야함.   위와마찬가지로 new로 할당, 콜백이불리면 delete
    //char m_RecvBuf[MAXBUFFER]; // 1개만써야함.
    //Position m_PlayerPosition;
    char m_CompletePacketBuf[MAXBUFFER];
    int m_PacketPrevSize;
    int recvBytes = 0;
    int sendBytes = 0;
    Player m_Player;

    SOCKETINFO()
    {
        m_RecvOverlappedEx = new OverlappedEx;
        memset(&(m_RecvOverlappedEx->m_Overlapped), 0x00, sizeof(WSAOVERLAPPED));
        m_RecvOverlappedEx->m_WSABuf.buf = m_RecvOverlappedEx->m_DataBuf;
        m_RecvOverlappedEx->m_WSABuf.len = MAXBUFFER;

    }
    ~SOCKETINFO() { if (m_SendOverlappedEx != nullptr) delete m_SendOverlappedEx; delete m_RecvOverlappedEx; }
};
```

전 페이지와 이어집니다.

소켓 정보를 담을 구조체입니다.

```
void ARBNetwork::BeginPlay()
{
    Super::BeginPlay();
    InitClientSocket();
    Connect();
    GetWorldTimerManager().SetTimer(m_SendTimer, this, &ARBNetwork::SendMyTransform, 0.016f, true);
}
```

앞서 작성한 함수들은 BeginPlay()에서 사용합니다.
게임이 시작될 때 네트워크에 접속할 수 있도록 합니다.

```cpp
void IOCPServer::mainLoop()
{
    WSADATA WSAData;
    WSAStartup(MAKEWORD(2, 2), &WSAData);

    m_ListenSock = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);

    SOCKADDR_IN ServerAddr;
    memset(&ServerAddr, 0x00, sizeof(SOCKADDR_IN));
    ServerAddr.sin_family = AF_INET;
    ServerAddr.sin_port = htons(SERVERPORT);
    ServerAddr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);

    ::bind(m_ListenSock, reinterpret_cast<sockaddr*>(&ServerAddr), sizeof(ServerAddr));

    listen(m_ListenSock, SOMAXCONN);

    //iocp 생성
    m_iocp = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, NULL, 0);

    // 엑셉트도 iocp로 받기위해 등록.
    CreateIoCompletionPort(reinterpret_cast<HANDLE>(m_ListenSock), m_iocp, 999, 0);

    //동기 accpet랑 다르게 미리 소켓을 만들어놓고 이를이용해 클라이언트와 통신한다.
    SOCKET Client_Sock = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
    EXOVER Accept_over;
    ZeroMemory(&Accept_over.Over, sizeof(Accept_over.Over));
    Accept_over.Op = OP_ACCEPT;
    AcceptEx(m_ListenSock, Client_Sock, Accept_over.IO_Buf, NULL, sizeof(sockaddr_in) + 16, sizeof(sockaddr_in) + 16, NULL, &Accept_over.Over);
```

서버에서는 iocp 서버를 생성하고, 클라이언트의 데이터 수신을 위해 accep() 작업을 진행합니다.

```
while (true)
{
    DWORD iobytes;
    ULONG_PTR key;
    WSAOVERLAPPED* over;
    GetQueuedCompletionStatus(m_iocp, &iobytes, &key, &over, INFINITE);
    //완료한 결과를 담고 있는 overlapped구조체를 확장 overlapped구조체로 탈바꿈
    EXOVER* exover = reinterpret_cast<EXOVER*>(over);
    int user_id = static_cast<int>(key);
    CLIENT& Client = m_Clients[user_id];

    //Accept시 id를 999로
    if(user_id == 999)
        m_Clients[user_id].m_Player.m_ID = user_id;
    switch (exover->Op)
    {
```

```
struct EXOVER
{
    WSAOVERLAPPED Over;
    ENUMOP Op;
    char IO_Buf[MAXBUFFER];
    WSABUF WSABuf;
};
struct CLIENT
{
    SOCKET m_Sock;
    EXOVER m_RecvOver;
    int m_PrevSize;
    char m_PacketBuf[MAXPACKETSIZE];

    //게임콘텐츠정보
    char name[MAX_ID_LEN + 1];

    Player m_Player;
};
```

서버에서 ProcessPacket()을 이용해
IO 버퍼에 데이터를 수신합니다.

이는 메인 루프에서 이루어지고 있고,
exover 변수의 Op값으로 분기를 나눠
수신한 데이터를 처리합니다.

```
case OP_RECV:
{//recv가 완료됐으면 패킷처리

    if (iobytes == 0)
    {
        Disconnect(user_id);
    }
    else
    {
        char* ptr = exover->IO_Buf;
        static size_t in_packet_size = 0;
        while (0 != iobytes)
        {
            if (0 == in_packet_size)
                in_packet_size = ptr[0];
            if (iobytes + Client.m_PrevSize >= in_packet_size)
            {
                memcpy(Client.m_PacketBuf + Client.m_PrevSize, ptr, in_packet_size - Client.m_PrevSize);
                ProcessPacket(user_id, Client.m_PacketBuf);      // 이부분입니다!
                ptr += in_packet_size - Client.m_PrevSize;
                iobytes -= in_packet_size - Client.m_PrevSize;
                in_packet_size = 0;
                Client.m_PrevSize = 0;
            }
            else
            {
                memcpy(Client.m_PacketBuf + Client.m_PrevSize, ptr, iobytes);
                Client.m_PrevSize += iobytes;
                iobytes = 0;
                break;
            }
        }

        ZeroMemory(&Client.m_RecvOver.Over, sizeof(Client.m_RecvOver.Over));
        DWORD flags = 0;

        WSARecv(Client.m_Sock, &Client.m_RecvOver.WSABuf, 1, NULL, &flags, &Client.m_RecvOver.Over, NULL);
    }
}
break;
```

```
case OP_SEND:
{//센드가 완료됐으면 오버랩구조체 메모리를 반환
    if (iobytes == 0)
        Disconnect(user_id);
    delete exover;
}
break;
case OP_ACCEPT:
{
    int id = m_current_user_id++;

    CreateIoCompletionPort(reinterpret_cast<HANDLE>(Client_Sock), m_iocp, id, 0); //key값으로 id를 준다.

    m_current_user_id = m_current_user_id;// % MAXPLAYER; //아이디가 초과하는것을 방지하기위함.
    CLIENT& newClient = m_Clients[id];
    newClient.m_Player.m_ID = id;
    newClient.m_PrevSize = 0;
    newClient.m_RecvOver.Op = OP_RECV;
    ZeroMemory(&newClient.m_RecvOver.Over, sizeof(newClient.m_RecvOver.Over));
    newClient.m_RecvOver.WSABuf.buf = newClient.m_RecvOver.IO_Buf;
    newClient.m_RecvOver.WSABuf.len = MAXBUFFER;
    newClient.m_Sock = Client_Sock;

    /* ... */

    // accpet하고난뒤에 초기화할 정보들을 이곳에 추가.
    cout << "Client" << id << " 접속" << endl;

    DWORD flags = 0;
    WSARecv(newClient.m_Sock, &newClient.m_RecvOver.WSABuf, 1, NULL, &flags, &newClient.m_RecvOver.Over, NULL);

    //계속해서 Accept를 받기위함.
    Client_Sock = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
    ZeroMemory(&Accept_over.Over, sizeof(Accept_over.Over));
    AcceptEx(m_ListenSock, Client_Sock, Accept_over.IO_Buf, NULL, sizeof(sockaddr_in) + 16, sizeof(sockaddr_in) + 16, NULL, &Accept_over.Over);

}
break;
```

exover 변수의 OP가 SEND일 때는
오버랩 구조체 메모리를 반환합니다.

exover 변수의 OP가 ACCEPT일 때는
클라이언트가 서버에 접속할 수 있도록
소켓을 할당해줍니다.

계속해서 Accept할 수 있도록
다음 클라이언트의 소켓과 Accept정보를
초기화한 후 Accept를 시작합니다.

```cpp
void IOCPServer::Send_Packet(int user_id, void* p)
{
    char* buf = reinterpret_cast<char*>(p);
    CLIENT& user = m_Clients[user_id];

    //별도의 센드용 확장오버랩구조체를 생성한다.
    EXOVER* exover = new EXOVER;
    exover->Op = OP_SEND;
    ZeroMemory(&exover->Over, sizeof(exover->Over));
    exover->WSABuf.buf = exover->IO_Buf;
    exover->WSABuf.len = buf[0];
    memcpy(exover->IO_Buf, buf, buf[0]);

    WSASend(user.m_Sock, &exover->WSABuf, 1, NULL, 0, &exover->Over, NULL);
}
```

서버에서 클라이언트로 패킷을 보낼 때,
확장 오버랩 구조체를 사용하여 보내기 위한 함수입니다.

```cpp
void IOCPServer::Disconnect(int user_id)
{
    m_Clients.erase(user_id);
    cout << "[Disconnect] user_id: " << user_id << endl;

    sc_packet_leavePacket p{};
    p.m_id = user_id;
    p.size = sizeof(p);
    p.type = e_PacketType::e_LeavePacket;
    for (auto& client : m_Clients)
    {
        if (client.second.m_Player.m_ID == 999) continue;
        Send_Packet(client.second.m_Player.m_ID, &p);
    }
}
```

클라이언트의 연결이 끊겼을 때 사용하는 함수입니다.
모든 클라이언트에게 접속이 끊긴 플레이어의 정보를 알립니다.

```
enum class e_PacketType : uint8_t
{
    e_LoginOK,
    e_Select_GameMode,
    e_LeavePacket,
    e_ReadyPacket,
    e_StartPacket,
    e_PlayerInfoPacket,
    e_EnterPacket,
    e_myCharacterPacket,
    e_BulletSpawnPacket,
    e_BulletSlotPacket,
    e_BulletMovePacket,
    e_BulletRotPacket,
    e_CharacterDeadPacket,
    e_CharacterReloadPacket,
    e_CharacterLightPacket,
};
```

```
void IOCPServer::ProcessPacket(int user_id, char* buf)
{
    e_PacketType packetType = static_cast<e_PacketType>(buf[1]);

    switch (packetType)
    {
    case e_PacketType::e_Select_GameMode:
    {
        cs_packet_selectGameMode* packet = reinterpret_cast<cs_packet_selectGameMode*>(buf);
        m_Clients[user_id].m_Player.m_gamemode = packet->mode;

        sc_packet_loginOK p{};
        p.m_id = user_id;
        p.size = sizeof(p);
        p.type = e_PacketType::e_LoginOK;

        Send_Packet(p.m_id, &p);
    }
    break;
    case e_PacketType::e_LeavePacket:
    {
        cs_packet_leavePacket* packet = reinterpret_cast<cs_packet_leavePacket*>(buf);

        Disconnect(packet->m_id);
    }
```

클라이언트에게 수신받은 정보를 패킷 별로 분리하여 처리하는 함수입니다.
먼저 Select GameMode는 로그인된 플레이어의 정보를 조립해 해당 클라이언트에 보내줍니다.
솔로용 게임 모드를 확인하기 위함입니다.
LeavePacket을 받으면 해당 클라이언트와 연결을 끊습니다.

```
case e_PacketType::e_BulletRotPacket:
{
    cs_packet_bulletRotPacket* packet = reinterpret_cast<cs_packet_bulletRotPacket*>(buf);

    sc_packet_bulletRotPacket p{};
    p.m_id = packet->m_id;
    /* ... */

    m_Clients[packet->m_id].m_Player.slot1 = packet->slot1;
    m_Clients[packet->m_id].m_Player.slot2 = packet->slot2;
    m_Clients[packet->m_id].m_Player.slot3 = packet->slot3;

    /* ... */

    /* ... */
}
break;
```

숨겨진 부분은 디버깅 메시지입니다.

```
struct BulletSlotData
{
    float x;
    float y;
    float z;
};
```

```
struct cs_packet_bulletRotPacket
{
    char size;
    e_PacketType type;
    int m_id;
    BulletSlotData slot1;
    BulletSlotData slot2;
    BulletSlotData slot3;
};
```

플레이어가 만든 총알의 정보(slot1/slot2/slot3)를 받았다면,
그 정보를 서버의 클라이언트 배열에 저장해 관리합니다.
모든 클라이언트가 정보를 받아 동기화 해야 하기 때문입니다.

```
case e_PacketType::e_ReadyPacket:
{
    cs_packet_readyPacket* packet = reinterpret_cast<cs_packet_readyPacket*>(buf);

    m_Clients[user_id].m_Player.m_isReady = packet->isReady;
    int readycnt = 0;
    for (auto c : m_Clients)
    {
        if (c.second.m_Player.m_ID == 999) continue;
        if (c.second.m_Player.m_isReady) ++readycnt;
    }

    int team1cnt = 0;
    int team2cnt = 0;
```

클라이언트에게 준비 정보를 받습니다.
서버에서 관리하는 클라이언트 정보에 준비 완료 상태를 저장하고, 준비가 완료됐다면 게임을 시작할 수 있게 StartPacket을 보냅니다.

시연 영상과 코드는 간결성을 위해 2인 기준으로 작성되었습니다. 이 점 양해해주시면 감사합니다.

```
int team1cnt = 0;
int team2cnt = 0;
if (readycnt == 2)   // 2인 기준
{
    for (auto& c : m_Clients)
    {
        if (c.second.m_Player.m_ID == 999) continue;
        if (team1cnt % 2 == 0)
        {
            c.second.m_Player.m_PlayerInfo.m_Position = PlayerPosition{ 5000.f + 200 * team1cnt++, 310.f, 42500.f };
            c.second.m_Player.m_PlayerInfo.m_Rotation = PlayerRotation{ 0.f, 90.f, 0.f };
        }
        else
        {// 6200 9500 250
            c.second.m_Player.m_PlayerInfo.m_Position = PlayerPosition{ 5200.f + 200 * team2cnt++, 310.f, 42500.f };
            c.second.m_Player.m_PlayerInfo.m_Rotation = PlayerRotation{ 0.f, -90.f, 0.f };
        }
    }

    for (auto& c : m_Clients)
    {
        if (c.second.m_Player.m_ID == 999) continue;

        sc_packet_startPacket p{};
        p.size = sizeof(p);
        p.type = e_PacketType::e_StartPacket;
        p.m_id = c.second.m_Player.m_ID;
        p.pos = c.second.m_Player.m_PlayerInfo.m_Position;
        p.rot = c.second.m_Player.m_PlayerInfo.m_Rotation;

        Send_Packet(c.second.m_Player.m_ID, &p);
    }
```

```
for (auto& c : m_Clients)
{
    if (c.second.m_Player.m_ID == 999) continue;
    // c는 본인 클라, cl은 타 클라
    for (auto& cl : m_Clients)
    {
        if (cl.second.m_Player.m_ID == 999) continue;

        if (cl.second.m_Player.m_ID != c.second.m_Player.m_ID)
        {
            // 내 클라 슬롯 정보를 다른 클라에게
            sc_packet_bulletRotPacket rp{};
            rp.m_id = c.second.m_Player.m_ID;
            rp.size = sizeof(rp);
            rp.type = e_PacketType::e_BulletRotPacket;
            rp.slot1 = m_Clients[c.second.m_Player.m_ID].m_Player.slot1;
            rp.slot2 = m_Clients[c.second.m_Player.m_ID].m_Player.slot2;
            rp.slot3 = m_Clients[c.second.m_Player.m_ID].m_Player.slot3;

            Send_Packet(cl.second.m_Player.m_ID, &rp);

            // 내 클라 정보를 다른 클라에게
            sc_packet_enterPacket sp{};
            sp.m_id = c.second.m_Player.m_ID;
            sp.pos = c.second.m_Player.m_PlayerInfo.m_Position;
            sp.rot = c.second.m_Player.m_PlayerInfo.m_Rotation;
            sp.size = sizeof(sp);
            sp.type = e_PacketType::e_EnterPacket;

            Send_Packet(cl.second.m_Player.m_ID, &sp);

            // ...
        }
    }
}
```

```
        for (auto& c : m_Clients)
        {
            sc_packet_enterPacket mp{};
            mp.m_id = c.second.m_Player.m_ID;
            mp.pos = c.second.m_Player.m_PlayerInfo.m_Position;
            mp.rot = c.second.m_Player.m_PlayerInfo.m_Rotation;
            mp.size = sizeof(mp);
            mp.type = e_PacketType::e_myCharacterPacket;

            Send_Packet(c.second.m_Player.m_ID, &mp);
        }
    }
}
break;
```

전 페이지와 이어집니다.

모든 클라이언트가 정보를 동기화할 수 있도록
BulletRotPacket, EnterPacket,
myCharacterPacket을 보내고 받게 합니다.

```cpp
case e_PacketType::e_PlayerInfoPacket:
{
    cs_packet_playerInfo* packet = reinterpret_cast<cs_packet_playerInfo*>(buf);

    m_Clients[packet->m_id].m_Player.m_PlayerInfo = packet->info;

    sc_packet_playerInfo p{};
    p.info = packet->info;//m_Clients[packet->m_id].m_Player.m_PlayerInfo;
    /* ... */

    /* ... */

    p.m_id = packet->m_id;
    p.size = sizeof(p);
    p.type = e_PacketType::e_PlayerInfoPacket;

    // 위에서 보내온 클라의 정보를
    // 서버가 타 클라들에게 쏴준다.
    for (auto& c : m_Clients)
    {
        if (c.second.m_Player.m_ID == 999) continue;
        if (c.second.m_Player.m_ID == packet->m_id) continue;

        Send_Packet(c.second.m_Player.m_ID, &p);
    }
}
break;
```

서버

```cpp
case e_PacketType::e_PlayerInfoPacket:
{
    sc_packet_playerInfo* packet = reinterpret_cast<sc_packet_playerInfo*>(m_PacketBuf);
    // ...
    /* ... */
    // ...

    auto pos = packet->info.m_Position;
    auto rot = packet->info.m_Rotation;
    auto vel = packet->info.m_Velocity;

    if (m_OtherPlayers[packet->m_id] != nullptr)
    {
        m_OtherPlayers[packet->m_id]->SetActorLocation(FVector(pos.x, pos.y, pos.z));
        m_OtherPlayers[packet->m_id]->SetActorRotation(FRotator(rot.Pitch, rot.Yaw, rot.Roll));
        m_OtherPlayers[packet->m_id]->AddMovementInput(FVector(vel.vx, vel.vy, vel.vz));
    }
}
break;
```

클라이언트

플레이어 정보 패킷을 받습니다.
받은 정보는 서버에 저장하고, 그대로 타 클라이언트에게 전송합니다.

```
case e_PacketType::e_BulletSpawnPacket:
{
    cs_packet_bulletSpawnPacket* packet = reinterpret_cast<cs_packet_bulletSpawnPacket*>(buf);
    sc_packet_bulletSpawnPacket p{};
    memcpy(&p, packet, sizeof(p));

    for (auto& c : m_Clients)
    {
        if (c.second.m_Player.m_ID == 999) continue;
        //if (c.second.m_Player.m_ID == packet->m_id) continue;

        Send_Packet(c.second.m_Player.m_ID, &p);
    }
}
break;
```

서버

```
case e_PacketType::e_BulletSpawnPacket:
{
    cs_packet_bulletSpawnPacket* packet = reinterpret_cast<cs_packet_bulletSpawnPacket*>(m_PacketBuf);


    FVector pos{ packet->pos.x, packet->pos.y, packet->pos.z };
    FRotator rot{ packet->rot.Pitch, packet->rot.Yaw, packet->rot.Roll };

    UE_LOG(LogTemp, Error, TEXT("e_BulletSpawnPacket id : %d"), packet->m_id);
    UE_LOG(LogTemp, Error, TEXT("e_BulletSpawnPacket Pos : x %f / y %f / z %f"), packet->pos.x, packet->pos.y, packet->pos.z);
    UE_LOG(LogTemp, Error, TEXT("e_BulletSpawnPacket Rot : Pit %f / Yaw %f / Rol %f"), packet->rot.Pitch, packet->rot.Yaw, packet->rot.Roll);


    // 타 클라의 총알을 스폰하기 위해 타클라 정보 받아왔음 (슬롯이 뭔지도 알아야지)
    // 슬롯이 뭔지 어케 알지
    bulletSpawnID = packet->m_id;

    //URBGameInstance* GameInstance = Cast<URBGameInstance>(UGameplayStatics::GetGameInstance(GetWorld()));

    switch (packet->bulletType)
    {
    case e_bulletType::e_Bullet1:
    {
        if (m_myCharacter->m_ID != packet->m_id)
        {
            m_OtherPlayers[packet->m_id]->SelectSlot1(packet->m_id);
            GetWorld()->SpawnActor<AProjectile>(BPProjectile, pos, rot, FActorSpawnParameters{});
        }
        else
        {
            m_myCharacter->SelectSlot1(packet->m_id);
            GetWorld()->SpawnActor<AProjectile>(BPProjectile, pos, rot, FActorSpawnParameters{});
        }
    }
    break;
```

클라이언트

총알의 스폰 정보를 받습니다.
타 클라이언트에게 이 정보를 전송합니다.

```
case e_PacketType::e_BulletSlotPacket:
{
    cs_packet_bulletSlotPacket* packet = reinterpret_cast<cs_packet_bulletSlotPacket*>(buf);
    sc_packet_bulletSlotPacket p{};
    memcpy(&p, packet, sizeof(p));

    for (auto& c : m_Clients)
    {
        if (c.second.m_Player.m_ID == 999) continue;
        if (c.second.m_Player.m_ID == packet->m_id) continue;

        Send_Packet(c.second.m_Player.m_ID, &p);
    }
}
break;
```

서버

```
case e_PacketType::e_BulletSlotPacket:
{
    sc_packet_bulletSlotPacket* packet = reinterpret_cast<sc_packet_bulletSlotPacket*>(m_PacketBuf);

    // 타 클라의 슬롯을 여기서 set했다.
    switch (packet->bulletType)
    {
    case e_bulletType::e_Bullet1:
    {
        m_OtherPlayers[packet->m_id]->SelectSlot1(packet->m_id);
    }
    break;
    case e_bulletType::e_Bullet2:
    {
        m_OtherPlayers[packet->m_id]->SelectSlot2(packet->m_id);
    }
    break;
    case e_bulletType::e_Bullet3:
    {
        m_OtherPlayers[packet->m_id]->SelectSlot3(packet->m_id);
    }
    break;
    }
}
break;
```

클라이언트

서버에서 보낸 총알 슬롯 정보를 클라이언트에서 저장합니다.

```
case e_PacketType::e_CharacterDeadPacket:
{
    cs_packet_deadPacket* packet = reinterpret_cast<cs_packet_deadPacket*>(buf);
    sc_packet_deadPacket p{};
    memcpy(&p, packet, sizeof(p));

    for (auto& c : m_Clients)
    {
        if (c.second.m_Player.m_ID == 999) continue;
        if (c.second.m_Player.m_ID == packet->m_id) continue;

        Send_Packet(c.second.m_Player.m_ID, &p);
    }
}
break;
case e_PacketType::e_CharacterReloadPacket:
{
    cs_packet_reloadPacket* packet = reinterpret_cast<cs_packet_reloadPacket*>(buf);
    sc_packet_reloadPacket p{};
    memcpy(&p, packet, sizeof(p));

    for (auto& c : m_Clients)
    {
        if (c.second.m_Player.m_ID == 999) continue;
        if (c.second.m_Player.m_ID == packet->m_id) continue;

        Send_Packet(c.second.m_Player.m_ID, &p);
    }
}
break;
```

서버

```
case e_PacketType::e_CharacterDeadPacket:
{
    sc_packet_deadPacket* packet = reinterpret_cast<sc_packet_deadPacket*>(m_PacketBuf);

    auto animinstance = Cast<URBAnimInstance>(m_OtherPlayers[packet->m_id]->GetMesh()->GetAnimInstance());
    animinstance->IsDead = true;
}
break;
case e_PacketType::e_CharacterReloadPacket:
{
    sc_packet_deadPacket* packet = reinterpret_cast<sc_packet_deadPacket*>(m_PacketBuf);

    m_OtherPlayers[packet->m_id]->Reload();
}
break;
```

클라이언트

서버에서 판단한 정보(dead, reload)를 클라이언트에 전송합니다.
수신한 클라이언트는 적절한 처리를 진행합니다.

```
case e_PacketType::e_CharacterLightPacket:
{
    cs_packet_lightPacket* packet = reinterpret_cast<cs_packet_lightPacket*>(buf);
    sc_packet_lightPacket p{};
    memcpy(&p, packet, sizeof(p));

    for (auto& c : m_Clients)
    {
        if (c.second.m_Player.m_ID == 999) continue;
        if (c.second.m_Player.m_ID == packet->m_id) continue;

        Send_Packet(c.second.m_Player.m_ID, &p);
    }
}
break;
default:
    cout << "Unknown Packet Type Error";
    DebugBreak();
    exit(-1);
}
```

서버

```
case e_PacketType::e_CharacterLightPacket:
{
    sc_packet_lightPacket* packet = reinterpret_cast<sc_packet_lightPacket*>(m_PacketBuf);

    m_OtherPlayers[packet->m_id]->LightOnOff();
}
break;
}
```

클라이언트

CharacterLight은 캐릭터가 들고 있는 무기의 헤드라이트를 의미합니다.
그 상태를 서버에 전송, 서버에서는 타 클라이언트에게 그 정보들을 전송합니다.

```
void ARBNetwork::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    RecvPacket();
}
```

```
void ARBNetwork::RecvPacket()
{
    DWORD RecvBytes = 0;
    DWORD flags = 0;

    // ...
    WSARecv(m_ClientSocket, &m_WSARecvBuf, 1, &RecvBytes, &flags, NULL, NULL);


    int iobytes = RecvBytes;
    ProcessPacket(iobytes, m_RecvBuf);
}
```

클라이언트에서는 RecvPacket() 함수를 Tick()에서 호출하여,
매 프레임마다 정보를 수신할 수 있습니다.

```
bool ARBNetwork::Get_Ready()
{
    return m_IsReady;
}

void ARBNetwork::Set_Ready(bool setValue)
{
    m_IsReady = setValue;
}

void ARBNetwork::Send_Ready()
{
    cs_packet_readyPacket p{};

    if (m_IsReady)
        p.isReady = true;
    else
        p.isReady = false;

    p.m_id = m_ID;
    p.size = sizeof(p);
    p.type = e_PacketType::e_ReadyPacket;

    DWORD SentBytes = 0;
    DWORD flags = 0;

    memcpy(m_SendBuf, &p, sizeof(p));
    m_WSASendBuf.len = sizeof(p);
    int retval = WSASend(m_ClientSocket, &m_WSASendBuf, 1, &SentBytes, flags, NULL, NULL);
}
```

클라이언트의 Send_Ready() 함수 및
준비 정보의 Get/Set 함수입니다.

ReadyPacket을 보낼 수 있습니다.

```cpp
void ARBNetwork::SendMyTransform()
{
    PlayerPosition pos{};
    PlayerRotation rot{};
    PlayerVelocity vel{};


    if (m_myCharacter != nullptr)
    {
        auto apos = m_myCharacter->GetActorLocation();
        pos.x = apos.X;
        pos.y = apos.Y;
        pos.z = apos.Z;

        auto arot = m_myCharacter->GetActorRotation();
        rot.Pitch = arot.Pitch;
        rot.Roll = arot.Roll;
        rot.Yaw = arot.Yaw;

        auto avel = m_myCharacter->GetVelocity();
        vel.vx = avel.X;
        vel.vy = avel.Y;
        vel.vz = avel.Z;

        PlayerInfo info{ pos,rot,vel };

        cs_packet_playerInfo p{};
        p.m_id = m_myCharacter->m_ID;
        p.size = sizeof(p);
        p.type = e_PacketType::e_PlayerInfoPacket;
        p.info = info;

        DWORD SentBytes = 0;
        DWORD flags = 0;

        memcpy(m_SendBuf, &p, sizeof(p));
        m_WSASendBuf.len = sizeof(p);
        int retval = WSASend(m_ClientSocket, &m_WSASendBuf, 1, &SentBytes, flags, NULL, NULL);
    }
}
```

클라이언트의 SendMyTransform() 함수입니다.

이 함수가 호출될 때 자신의 Transform 정보를 서버에 보냅니다.

```
void ARBNetwork::SendProjectileSpawn(FVector loc, FRotator rot)
{
    PlayerPosition Pos{};
    PlayerRotation Rot{};

    if (m_myCharacter != nullptr)
    {
        Pos.x = loc.X;
        Pos.y = loc.Y;
        Pos.z = loc.Z;

        Rot.Pitch = rot.Pitch;
        Rot.Yaw = rot.Yaw;
        Rot.Roll = rot.Roll;

        URBGameInstance* GameInstance = Cast<URBGameInstance>(UGameplayStatics::GetGameInstance(GetWorld()));
        GameInstance->SelectSlot[m_ID];

        cs_packet_bulletSpawnPacket bp{};
        bp.m_id = m_myCharacter->m_ID;
        bp.pos = Pos;
        bp.rot = Rot;
        bp.bulletType = (e_bulletType)GameInstance->SelectSlot[m_ID];
        bp.size = sizeof(bp);
        bp.type = e_PacketType::e_BulletSpawnPacket;

        DWORD SentBytes = 0;
        DWORD flags = 0;

        memcpy(m_SendBuf, &bp, sizeof(bp));
        m_WSASendBuf.len = sizeof(bp);
        int retval = WSASend(m_ClientSocket, &m_WSASendBuf, 1, &SentBytes, flags, NULL, NULL);
    }
}
```

클라이언트의 SendProjectileSpawn() 함수입니다.

플레이어가 총을 발사했을 때 호출되며, 서버에 총알의 위치 및 회전 정보를 전송할 수 있도록 합니다.

```
void ARBNetwork::SendBulletType(e_bulletType type)
{
    // 내가 바꾼 슬롯을 다른 클라에게 알리기 위해
    if (m_myCharacter != nullptr)
    {
        cs_packet_bulletSlotPacket bp{};
        bp.m_id = m_myCharacter->m_ID;
        bp.bulletType = type;
        bp.size = sizeof(bp);
        bp.type = e_PacketType::e_BulletSlotPacket;

        DWORD SentBytes = 0;
        DWORD flags = 0;

        memcpy(m_SendBuf, &bp, sizeof(bp));
        m_WSASendBuf.len = sizeof(bp);
        int retval = WSASend(m_ClientSocket, &m_WSASendBuf, 1, &SentBytes, flags, NULL, NULL);
    }
}
```

클라이언트의 SendBulletType() 함수입니다.

플레이어가 총알 슬롯을 바꿨을 때 호출되며, 바꿨다는 정보를 서버에 전송합니다.

```
void ARBNetwork::SendCharacterDeadState(int id)
{
    if (m_myCharacter != nullptr)
    {
        cs_packet_deadPacket dp{};
        dp.m_id = id;
        dp.size = sizeof(dp);
        dp.type = e_PacketType::e_CharacterDeadPacket;


        DWORD SentBytes = 0;
        DWORD flags = 0;

        memcpy(m_SendBuf, &dp, sizeof(dp));
        m_WSASendBuf.len = sizeof(dp);
        int retval = WSASend(m_ClientSocket, &m_WSASendBuf, 1, &SentBytes, flags, NULL, NULL);
    }
}
```

클라이언트의 SendCharacterDeadState() 함수입니다.

플레이어가 죽었을 때 서버에 그 정보를 서버에 전송합니다.

```
void ARBNetwork::SendCharacterReloadState(int id)
{
    if (m_myCharacter != nullptr)
    {
        cs_packet_reloadPacket rp{};
        rp.m_id = id;
        rp.size = sizeof(rp);
        rp.type = e_PacketType::e_CharacterReloadPacket;


        DWORD SentBytes = 0;
        DWORD flags = 0;

        memcpy(m_SendBuf, &rp, sizeof(rp));
        m_WSASendBuf.len = sizeof(rp);
        int retval = WSASend(m_ClientSocket, &m_WSASendBuf, 1, &SentBytes, flags, NULL, NULL);
    }
}
```

클라이언트의 SendCharacterReloadState() 함수 입니다.

캐릭터가 총을 재장전할 때 그 정보를 서버에 전송합니다.

```
void ARBNetwork::SendCharacterLightState(int id)
{
    if (m_myCharacter != nullptr)
    {
        cs_packet_lightPacket lp{};
        lp.m_id = id;
        lp.size = sizeof(lp);
        lp.type = e_PacketType::e_CharacterLightPacket;


        DWORD SentBytes = 0;
        DWORD flags = 0;

        memcpy(m_SendBuf, &lp, sizeof(lp));
        m_WSASendBuf.len = sizeof(lp);
        int retval = WSASend(m_ClientSocket, &m_WSASendBuf, 1, &SentBytes, flags, NULL, NULL);
    }
}
```

클라이언트의 SendCharacterLightState() 함수입니다.

플레이어가 총의 라이트를 켜고 있는지 아닌지에 대한 정보를 서버에 전송합니다.

```cpp
void ARBNetwork::SendBulletRotData()
{
    cs_packet_bulletRotPacket rp{};

    rp.m_id = m_ID;
    rp.size = sizeof(rp);
    rp.type = e_PacketType::e_BulletRotPacket;

    // 게임 인스턴스에 저장되어 있는 총알의 회전 방향을 패킷에 저장해 보낸다.
    URBGameInstance* GameInstance = Cast<URBGameInstance>(UGameplayStatics::GetGameInstance(GetWorld()));
    rp.slot1.x = GameInstance->SaveSlot1_InstanceX[m_ID];
    rp.slot1.y = GameInstance->SaveSlot1_InstanceY[m_ID];
    rp.slot1.z = GameInstance->SaveSlot1_InstanceZ[m_ID];

    rp.slot2.x = GameInstance->SaveSlot2_InstanceX[m_ID];
    rp.slot2.y = GameInstance->SaveSlot2_InstanceY[m_ID];
    rp.slot2.z = GameInstance->SaveSlot2_InstanceZ[m_ID];

    rp.slot3.x = GameInstance->SaveSlot3_InstanceX[m_ID];
    rp.slot3.y = GameInstance->SaveSlot3_InstanceY[m_ID];
    rp.slot3.z = GameInstance->SaveSlot3_InstanceZ[m_ID];

    DWORD SentBytes = 0;
    DWORD flags = 0;

    memcpy(m_SendBuf, &rp, sizeof(rp));
    m_WSASendBuf.len = sizeof(rp);
    int retval = WSASend(m_ClientSocket, &m_WSASendBuf, 1, &SentBytes, flags, NULL, NULL);
}
```

클라이언트의 SendBulletRotData() 함수입니다.

플레이어의 총알 슬롯 정보를 서버에 전송합니다.

감사합니다.