# 네트워크
# 게임 프로그래밍

# 텀프로젝트 계획서

# 총잽이 난투

2020180016 민경원 2020180009 김용주 2020180021 송승호

# 목차

# 클라이언트 기획

▶ 게임 이름: 총잽이 난투(Gun Mayham 모작)

▶ 교과목: 네트워크 게임 프로그래밍을 위한 자체제작 게임

▶ 게임 장르: 2D Platform Shooting

▶ 게임 소개: 적을 총으로 격추시켜 맵밖으로 밀어내서
승리를 쟁취

▶ 조작키

▷ 이동: 방향키

▷ 총 발사: A

▶ 게임 목표

▷ 모든 플레이어는 중력의 영향을 받아 맵과
충돌 중이 아닌 경우 아래로 떨어짐

▷1:1 전투로, 플레이어는 다른 플레이어를 총으로
격추시켜 맵 밖으로 밀어내야 함

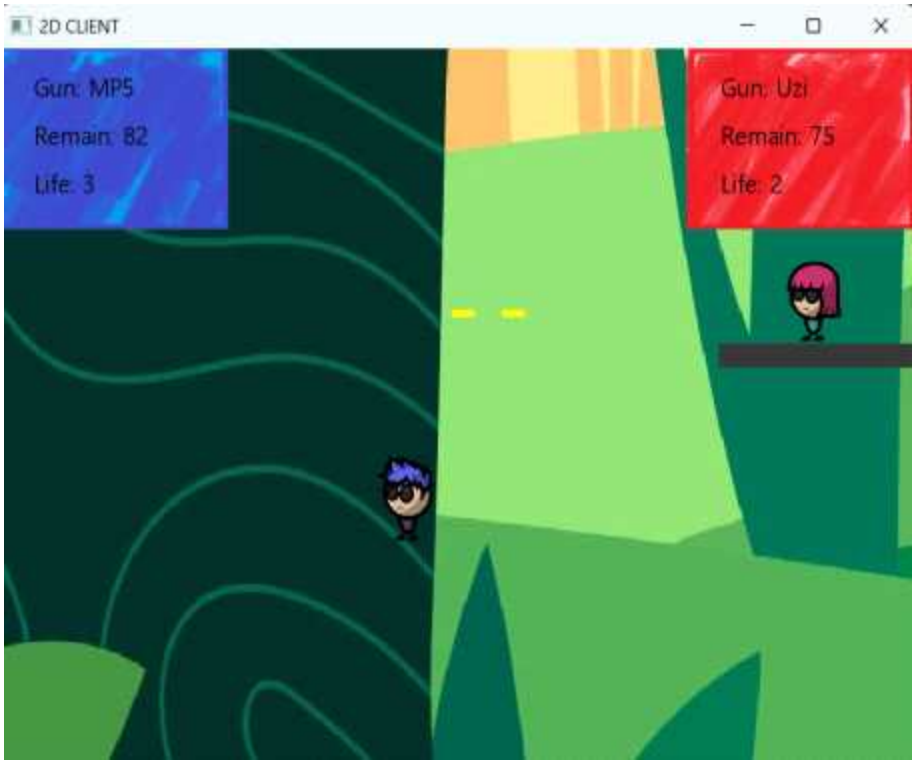
▷각 플레이어는 3의 체력이 있고, 맵 밖으로 떨어질 경우
체력이 -1

▷맵 밖으로 떨어진 후 1초 뒤 공중에서 부활 후 맵 위로
떨어짐

▷현재 사용중인 총을 다른 총으로 바꿔주는 아이템이
주기적으로 공중에서 생성

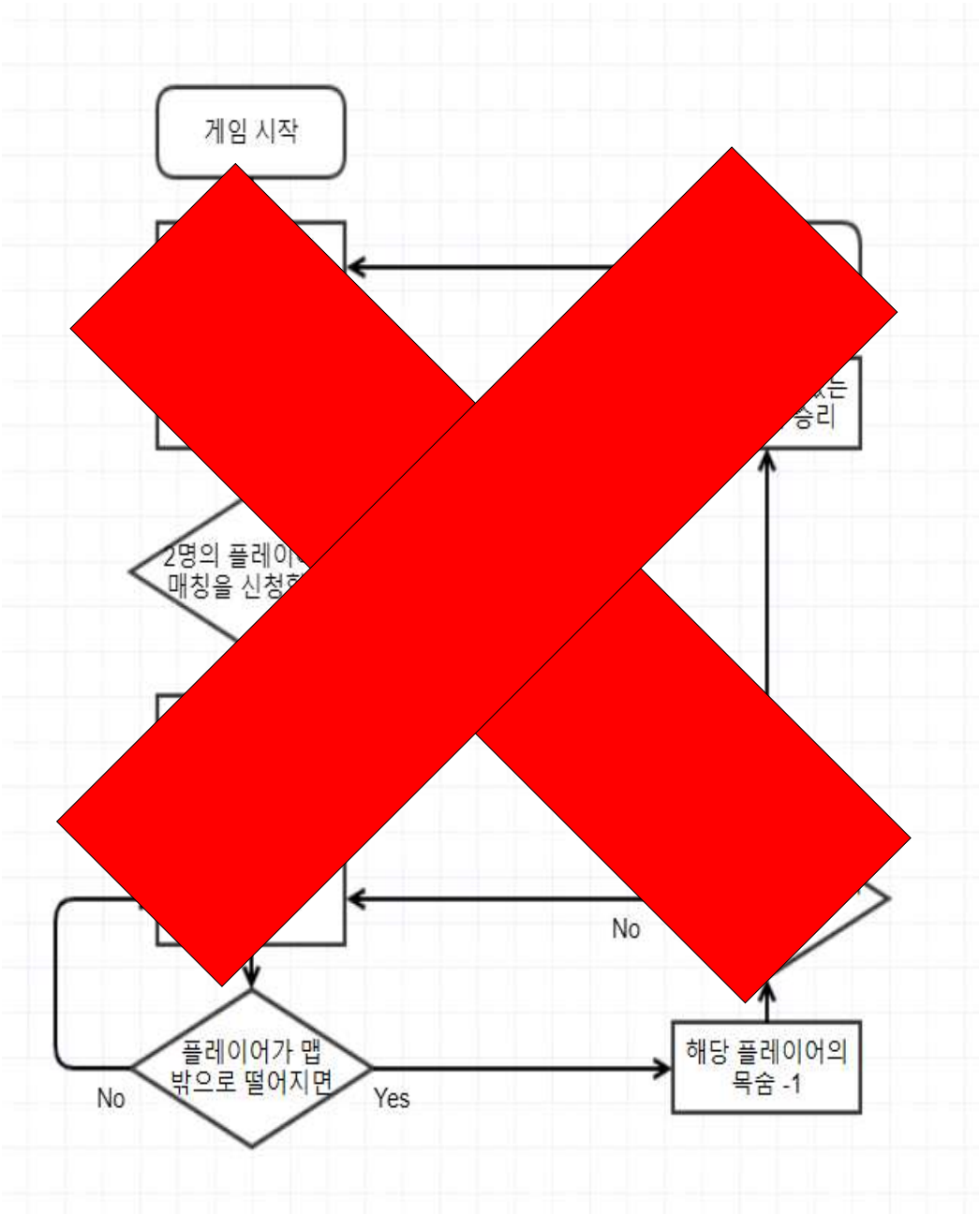
▷한 명의 플레이어라도 체력이 0이 되면 해당 게임은 종료

## 인 게임 스크린 샷


2D CLIENT
Gun: MP5
Remain: 82
Life: 3
Gun: Uzi
Remain: 75
Life: 2


2D CLIENT
Gun: M4A1
Remain: 75
Life: 3
Gun: MP5
Remain: 27
Life: 2

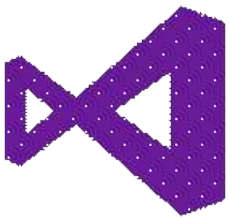
## 클라이언트 흐름 플로우차트



## 개발 환경

▶ 운영체제: Windows

▶ 컴파일러: Visual Studio



▶ 클라이언트 라이브러리: SFML


▶ 통신 프로토콜: TCP/IP


▶ 형상관리 프로그램: Git Hub

# High-Level Design

## 서버 구조

1. 서버 실행

▶ 클라이언트의 연결 대기: accept

2. 매칭

▶ 매칭을 시도하는 플레이어가 2명이면 매칭을 잡고 매칭 완료

3. 충돌 처리

▶ 플레이어 캐릭터와 총알과의 충돌 처리

▶ 플레이어 캐릭터와 아이템과의 충돌 처리

4. 클라이언트로 패킷 전송

▶ 각 플레이어간 필요한 정보를 클라이언트로 전송

## 클라이언트 구조

1. 클라이언트 실행

▶ 클라이언트 실행과 동시에 서버에 연결: Connect

▶ 게임 타이틀에서 매칭 신청 버튼을 눌러 매칭 시작

2. 매칭 완료

▶ 매칭이 완료되었다는 알림창을 3초 띄운 후 게임 시작

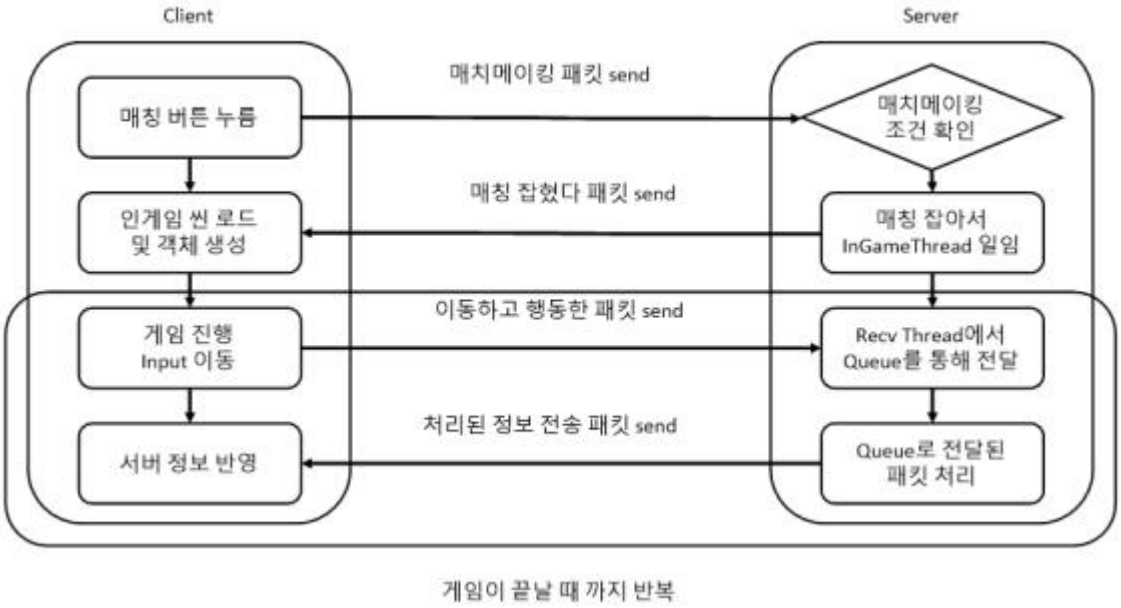
3. 플레이어 이동

▶ 클라이언트에서 플레이어의 이동을 계산한 후 서버로 전송

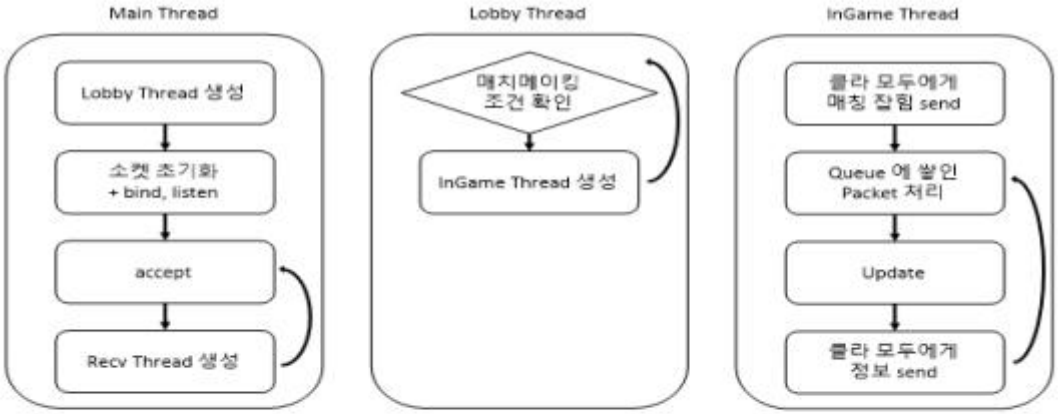
4. 서버로 패킷 전송

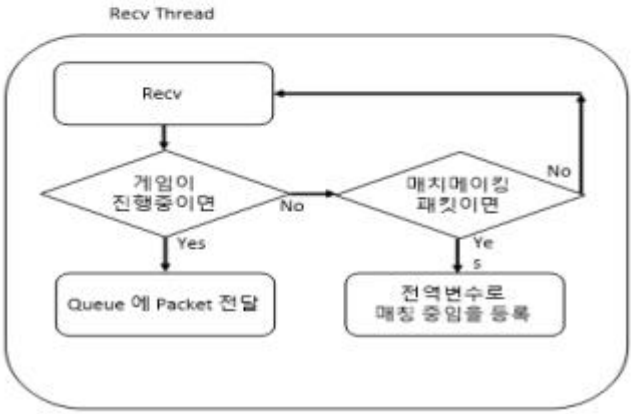
▶ 각 플레이어간 필요한 정보를 서버로 전송

## 클라이언트 – 서버 통신 플로우 차트



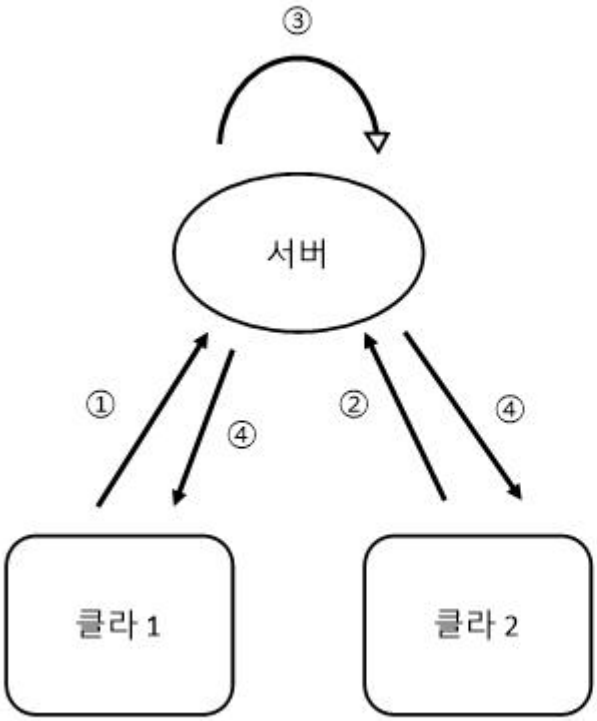

## 서버 구조 플로우 차트



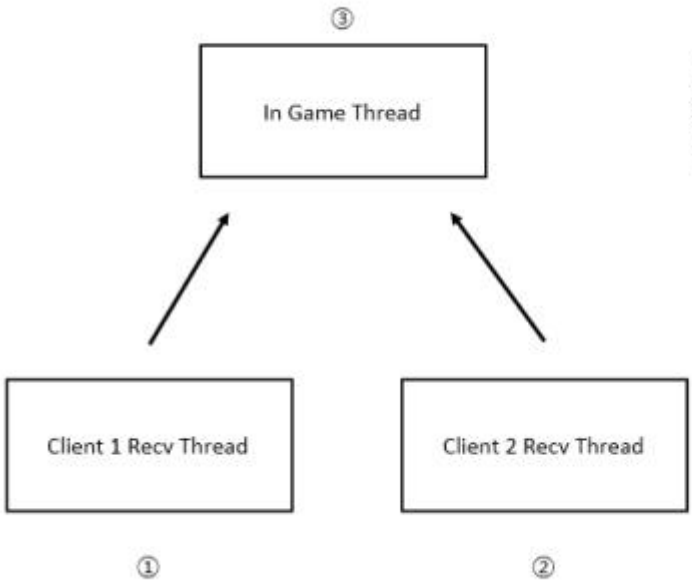
# 동기화 방법

▶ 네트워크 동기화



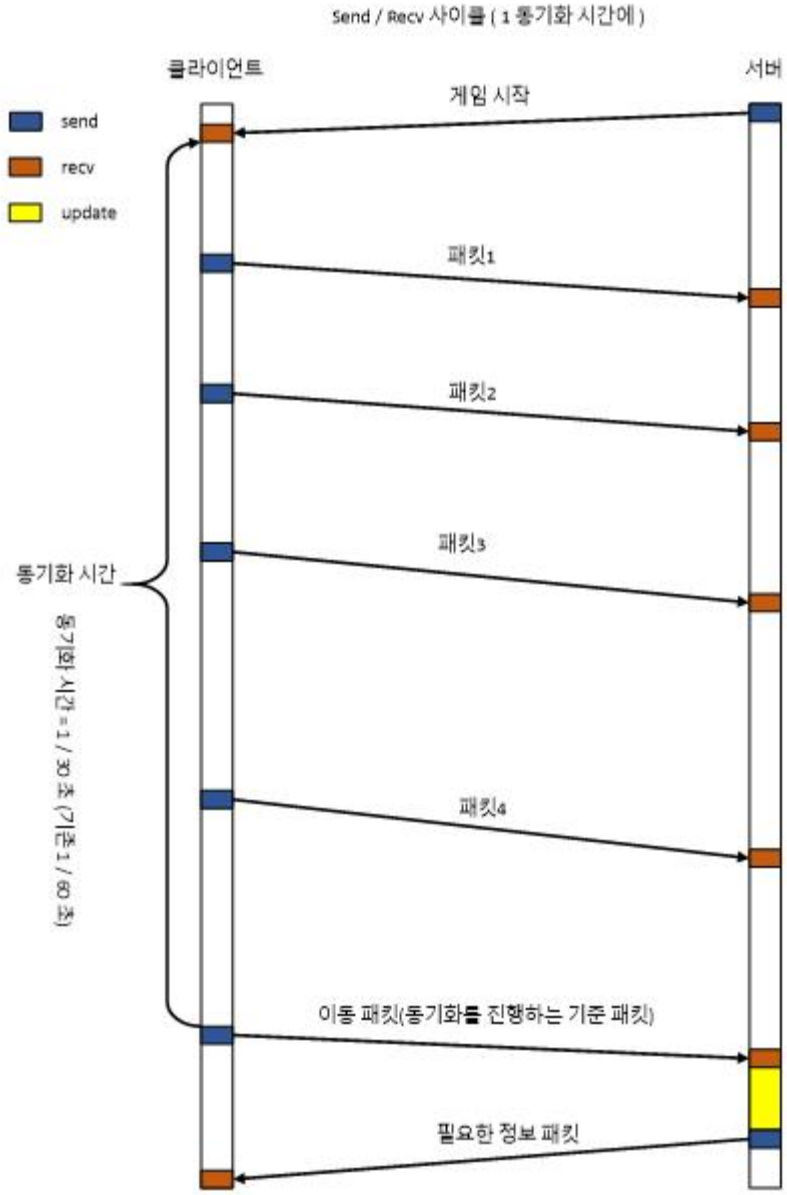

① 클라1의 정보를 서버로 send
② 클라2의 정보를 서버로 send
③ 서버 Update 처리
④ 처리된 정보를 클라로 send

▶ 이벤트로 스레드 간 순서 제어




이벤트 흐름
① 클라1 로부터 recv 다음 스레드
② 클라2 로부터 recv 다음 스레드
③ 패킷 처리 후 클라들에게 send

▶ 이동 패킷을 기준으로 동기화를 진행



# Low-Level Design

### 클라이언트 to 서버

| 구조체 | 역할 |
| --- | --- |
| struct BASE_PACKET<br>{<br>uint8_t size;<br>uint8_t id;<br>}; | 패킷에 공통된 헤더로<br>패킷이 가질 size와 id를 알려준다 |
| struct CS_MATCHCATCH_PACKET : public BASE_PACKET<br>{<br><br>}; | 플레이어가 매칭을 잡으려고 한다는 것을 알려주는 패킷 |
| struct CS_MOVE_PACKET : public BASE_PACKET<br>{<br>uint32_t p_id;<br>float posX, posY;<br>bool dir;<br>~~bool moving;~~<br>}; | 플레이어 위치와 방향, 움직이는지 아닌지를 알려주는 패킷 (1/60초 마다 보낼 예정) |
| struct CS_FIRE_PACKET : public BASE_PACKET<br>{<br>uint32_t b_id;<br>float posX, posY;<br>bool dir;<br>uint32_t type;<br>chrono::milliseconds fire_t;<br>}; | 해당 위치에서 총알이 발사되었음을 알리는 패킷,<br>발사 위치 및 방향, 총알의 타입 등 정보를 포함<br>chrono의 시간은 총알의 위치 보간을 위해 사용할 예정 (TM에서) |

**서버 to 클라이언트**

| 구조체 | 역할 |
| --- | --- |
| struct SC_MATCHMAKING_PACKET : public BASE_PACKET<br>{<br>bool succ;<br>+uint32_t p_id;<br>}; | 매치 메이킹 성공 실패 여부를 알리는 패킷 |
| struct SC_MOVE_PACKET : public BASE_PACKET<br>{<br>uint32_t p_id;<br>float posX, posY;<br>+bool dir;<br>}; | 플레이어 위치를 서버에서 클라이언트에 알려주는 패킷 |
| struct SC_PLAYER_DAMAGE_PACKET : public BASE_PACKET<br>{<br>float damage;<br>+bool dir;<br>}; | 해당 플레이어가 총알에 맞아 데미지를 받았음을 알리는 패킷 |
| struct SC_FIRE_PACKET : public BASE_PACKET<br>{<br>uint32_t b_id;<br>float posX, posY;<br>bool dir;<br>uint32_t type;<br>chrono::milliseconds fire_t;<br>}; | 상대 클라의 총알 발사를 알려주는 패킷 |
| struct SC_BULLET_REMOVE_PACKET : public BASE_PACKET<br>{<br>uint32_t p_id;<br>uint32_t b_id;<br>}; | 해당 총알이 서버에서 처리(소모) 되어 게임 씬에서 사라져야 함을 알리는 패킷 |
| struct SC_ITEM_CREATE_PACKET : public BASE_PACKET<br>{<br>uint32_t i_id;<br>float posX, posY;<br>}; | 해당 위치에 아이템이 생성되었음을 알리는 패킷 (이미지 처리를 위해) |

| | |
|---|---|
| struct SC_ITEM_REMOVE_PACKET : public BASE_PACKET<br>{<br>uint32_t i_id;<br>}; | 해당하는 아이템의 ID의 객체가 소모되어 게임 씬에서 사라져야 함을 알리는 패킷 |
| struct SC_GUN_UPDATE_PACKET : public BASE_PACKET<br>{<br>+uint32_t p_id;<br>uint32_t g_id;<br>}; | 해당 클라이언트의 플레이어의 총기가 변경됨을 알리는 패킷 |
| struct SC_LIVE_UDPATE_PACKET : public BASE_PACKET<br>{<br>uint32_t p_id;<br>}; | 해당 플레이어가 목숨이 감소했음을 알리는 패킷 (그게 클라이언트 플레이어 ID라면 사망 처리) |
| struct SC_GAMEOVER_PACKET : public BASE_PACKET<br>{<br><br>}; | 게임 종료 조건이 만족되어 게임이 종료되었음을 알리는 패킷 (타이틀 씬으로 복귀) |

# 예상 구현

**클라이언트**

```
// 패킷을 받아 queue에 등록 후 정보 전송(인자: 없음)
DWORD WINAPI ::workerRecv(LPVOID arg);

class NetworkManager {
private:
        Handle socket;    // 소켓
        Handle thread;    // 스레드
        ~~queue<Buffer[MAX_SIZE]> process_queue;    // 스레드 전달 큐~~
        int ClientID;        // 클라이언트 ID
        Handle recvEvent; // 스레드 동기화를 위한 이벤트
        Handle processEvent;        // 스레드 동기화를 위한 이벤트
        +bool playing;              // 게임중인지를 나누는 변수
        +array<char, INET_ADDRSTRLEN> addr;       // 주소 연결에 사용
public:
        NetworkManager();           // 생성자
        ~NetworkManager();          // 소멸자
        +void Init()                // 초기화
        void Connect();             // Connet();
        void CreateRecvThread();    // Recv() 스레드 생성
        void PushBuffer(char buf[MAX_SIZE]);          // enQueue
        void SendPacket(char buf[MAX_SIZE]);          // 패킷 전송

        +HANDLE GetRecvEvent();
        +HANDLE GetProcessEvent();
        +SOCKET GetSocket();
        +int GetClientID();

        +void ProcessPacket();
        +void ProcessPlayerMove(myNP::SC_MOVE_PACKET* move_packet);
        +void ProcessMatchMaking(myNP::SC_MATCHMAKING_PACKET* matchmaking_packet);
        +void ProcessFirebullet(myNP::SC_FIRE_PACKET* fire_packet);
        +void ProcessLifeUpdate(myNP::SC_LIFE_UPDATE_PACKET* life_packet);
        +ProcessGunUpdate(myNP::SC_GUN_UPDATE_PACKET* gun_packet);
        +ProcessCreateItem(myNP::SC_ITEM_CREATE_PACKET* item_create_packet);
        +ProcessRemoveItem(myNP::SC_ITEM_REMOVE_PACKET* item_remove_packet);
        +ProcessRemoveBullet(myNP::SC_BULLET_REMOVE_PACKET* bullet_remove_packet);
        +ProcessPlayerDamage(myNP::SC_PLAYER_DAMAGE_PACKET* player_damage_packet);
        +ProcessGameover(myNP::SC_GAMEOVER_PACKET* gameover_packet);
        ~~void Update();     // Update~~
        ~~void Draw();       // Draw~~
}
```

```
class SceneManager {
private:
        +SceneManager();
        +~SceneManager();

        vector<Scene> scenes;    // Scene들의 벡터
        Scene currentScene;      // 현재 Scene
public:
        void ChangeScene();      // Scene 전환
        +std::shared_ptr<Scene> LoadTitleScene();
        +std::shared_ptr<Scene> LoadGameScene(uint32_t p_id);

        +std::shared_ptr<Scene> GetActiveScene() const;

        void Update();   // Scene Update
        void Draw();     // Scene Draw
}
```

**서버**

```
// 지난 시간 만큼 객체의 상태를 update(인자: 이벤트 핸들 update)
DWORD WINAPI ::workerUpdate(LPVOID arg);
// 패킷을 받아 queue에 등록 후 정보 전송(인자: 이벤트 핸들 recv)
DWORD WINAPI ::workerRecv(LPVOID arg);
// 매치메이킹 검사(인자: 없음)
DWORD WINAPI ::workerLobby(LPVOID arg);

class NetworkManager {
private:
        ~~Handle socket;    // 소켓~~
        +SOCKET listenSocket;
        ~~Handle thread;    // 스레드~~
        +HANDLE updateThread;
        +HANDLE lobbyThread;
        ~~vector<queue<Buffer[MAX_SIZE]>> process_queue;    // 스레드 전달 큐 벡터~~
        +array<queue<std::array<char, MAX_SIZE>>, 2> processQueue; // 스레드 전달 큐
        array<Handle, 2> recvEvent;          // 스레드 동기화를 위한 이벤트
        array<Handle, 2> processEvent;       // 스레드 동기화를 위한 이벤트

        +int       nextId;
        +bool      playing;

        +bool doSend(SOCKET sock, const BufferType& buffer) const;
public:
        NetworkManager();           // 생성자
        ~NetworkManager();          // 소멸자
        void Init();          // 초기화
        void CreateLobbyThread(); // 로비 스레드 생성
        void CreateUpdateThread();// Update 스레드 생성
        void CreateRecvThread(HANDLE socket);       // Recv() 스레드 생성
        ~~void PushBuffer(Buffer[MAX_SIZE] buf);          // 버퍼 Push~~
        +void setProcessQueue(const QueueType& queue_, const int client_id);
        ~~void SendPacket(char buf[MAX_SIZE]);          // 패킷 Send()~~
        +void SendPacket(SOCKET sock, Args... args) const;
        +void NetworkInit();
        +void EventInit();
        +void Accept();
        +bool DoRecv(SOCKET sock, BufferType& buffer) const;
        +void ProcessPackets();
        +int GetNextId();
        +void DecreaseNextID();
        +bool IsPlaying() const;
        +void setPlaying(const bool value);
        +setProcessQueue(const QueueType& queue_, const int client_id);
```

```
        +setSocketArr(SOCKET sock, const int client_id);

        +void SetRecvEvent(const int c_id);
        +void WaitforRecvEvent();

        +void SetProcessEvent();
        +void WaitforProcessEvent(const int c_id);

        queue<Buffer[MAX_SIZE]> GetQueue();          // Getter
}




class World {
private:
        vector<Object> objects;    // 객체
        +TimerManager* tm;                  // TimerManager

        +Player p1;
        +Player p2;

        +std::list<Bullet> b1;          // p1이 쏜 총알
        +std::list<Bullet> b2;          // p2이 쏜 총알

        +list<Item> items;              // item들 정보를 담고 있는 자료구조
        +system_clock::time_point itemMakingTime;   // item을 생성하는데 필요한

        +Level* level;       // 플랫폼들 (레벨)
public:
        +World();
        void Update()       // 오브젝트 Update
        void Process();     // 오브젝트 Process
        void Recv();        // 오브젝트 Recv
        +void CollisionCheck();       // World의 요소들을 충돌처리 함다
}
```

**공용**

```
class TimerManager {
private:
        +chrono::system_clock::time_point startTime;
        chrono::system_clock::time_point oldTime;     // 이전 업데이트 프레임 기록
        chrono::milliseconds deltaTime;       // deltaTime
        +int64_t deltaTime;          // deltaTime
        +int64_t syncTime;
public:
        void Init();          // 초기화
        void Update();     // Update
        bool isSyncTime();// 동기화 시간인지 알려줌
        chrono::milliseconds epochToMillis();// epoch 시간으로부터의 차이
        chrono::milliseconds timeGap(chrono::milliseconds bullet_fire_t); // 시차

        +long long getDeltaTime() const;              // deltaTime을 가져온다
}
```

# 역할 분담

## 민경원

▶ 클라이언트 네트워크 기능 및 스레드 생성

▶ Scene Class 제작 및 객체 관리

▶ Server to Client Packet의 처리

## 김용주

▶ 서버 네트워크 기능 및 스레드 생성

▶ Client to Server Packet 송신 및 처리

▶ Title Scene 생성 및 전환

▶ UI ~~및 Animation~~ 제작

## 송승호

▶ Protocol 생성

▶ 서버 내 World 정보 관리

▶ Timer Class 제작

▶ Server to Client Packet의 송신

# 개발 일정

## 10 주차

| 날짜<br>팀원 | 월<br>(11/4) | 화<br>(11/5) | 수<br>(11/6) | 목<br>(11/7) | 금<br>(11/8) | 토<br>(11/9) | 일<br>(11/10) |
|---|---|---|---|---|---|---|---|
| 민경원 | Client Network Mgr() | C_NM Init() | C_NM Connect() | C_NM Push Buffer() | ::Worker Recv() | 디버깅, 머지, 오류 수정, 부족한 코드 개선 | |
| 김용주 | Server Network Manager 선언 및 정의 | S_NM init() | S_NM Push Buffer() | S_NM Get Queue() | S_NM Send Packet() | | |
| 송승호 | Protocol Enum, 상수 정의 | Protocol 패킷 선언 및 연동 | World Class 선언 및 정의 | Object Class 선언 및 정의 | Server Player 구현 | | |

## 11 주차

| 날짜<br>팀원 | 월<br>(11/11) | 화<br>(11/12) | 수<br>(11/13) | 목<br>(11/14) | 금<br>(11/15) | 토<br>(11/16) | 일<br>(11/17) |
|---|---|---|---|---|---|---|---|
| 민경원 | C_NM Create Recv Thread() | C_NM Process Queue() | SC_matchmaking 처리 | SC_move 처리 | Scene 정의 및 선언 | 디버깅, 머지, 오류 수정, 부족한 코드 개선 | |
| 김용주 | ::Worker Lobby() | ::Worker Update() | ::Worker Recv() | Create Thread's | SC_move 보내기 | | |
| 송승호 | Server Bullet 객체 구현 | Server Item 객체 구현 | Server World Update() | World 객체 내 충돌 감지 | 객체 충돌 처리 | | |

# 개발 일정

## 12 주차

| 날짜<br>팀원 | 월<br>(11/18) | 화<br>(11/19) | 수<br>(11/20) | 목<br>(11/21) | 금<br>(11/22) | 토<br>(11/23) | 일<br>(11/24) |
|---|---|---|---|---|---|---|---|
| 민경원 | 기존 객체 구조 변경 | Scene 에서 객체 관리 | Scene update() Draw() | Scene Manager update() Draw() | Scene Manager Change Scene() | 디버깅, 머지, 오류 수정, 부족한 코드 개선 | |
| 김용주 | SC_matchmaking 보내기 | 클라이언트 Send() | CS_MOVE 보내기 | Server CS_MOVE 처리 | CS_FIRE_PACKET 보내기 | | |
| 송승호 | Timer Manager 정의 및 선언 | TM init() | TM isSyncTime() | TM epochToMillis, timeGap | World에 Time 기능 적용 | | |

## 13 주차

| 날짜<br>팀원 | 월<br>(11/25) | 화<br>(11/26) | 수<br>(11/27) | 목<br>(11/28) | 금<br>(11/29) | 토<br>(11/30) | 일<br>(12/1) |
|---|---|---|---|---|---|---|---|
| 민경원 | SC_FIRE_ 처리 | SC_GUN_UPDATE_ 처리 | SC_LIVE_UPDATE 처리 | SC_ITEM_CREATE 처리 | SC_ITEM_LOCATION 처리 | 디버깅, 머지, 오류 수정, 부족한 코드 개선 | |
| 김용주 | Server CS_FIRE_PACKET 처리 (11/27) | Server processEvent 관련 구현 | Server recvEvent 관련 구현 | Client processEvent 관련 구현 | Client recvEvent 관련 구현 | | |
| 송승호 | SC_FIRE_PACKET 전송 | SC_GUN_UPDATE_PACKET 전송 | SC_LIVE_UDPATE_PACKET 전송 | SC_ITEM_CREATE 전송 | SC_ITEM_LOCATION 전송 | | |

# 개발 일정

## 14 주차

| 날짜<br>팀원 | 월<br>(12/2) | 화<br>(12/3) | 수<br>(12/4) | 목<br>(12/5) | 금<br>(12/6) | 토<br>(12/7) | 일<br>(12/8) |
|---|---|---|---|---|---|---|---|
| 민경원 | SC_ITEM_REMOVE 처리 | SC_BULLET_REMOVE 처리 | SC_PLAYER_DAMAGE 처리 | 게임 종료 구현 | 디버깅, 머지, 오류 수정, 부족한 코드 개선 | | |
| 김용주 | 타이틀 Scene 구현 | Scene 전환 적용 | SC_GAMEOVER 처리, 전송 | 게임 종료 시 처리 | | | |
| 송승호 | SC_ITEM_REMOVE 전송 | SC_BULLET_REMOVE 전송 | SC_PLAYER_DAMAGE 전송 | 게임 종료 시 처리 | | | |