네트워크 게임 프로그래밍

텀프로젝트 기획서

-배틀 탱크-

2017180007 김지은

2018182008 김수민

2018180037 전혜진

목차

[1. 게임 개요 5](#_Toc57678173)

[A. 게임 컨셉 5](#_Toc57678174)

[B. 게임 정보 5](#_Toc57678175)

[C. 조작 방법 5](#_Toc57678176)

[D. 게임 플레이 5](#_Toc57678177)

[2. HIGH-LEVEL 6](#_Toc57678178)

[A. 로비 (Lobby Scene) 6](#_Toc57678179)

[A-1. 서버 6](#_Toc57678180)

[A-2. 클라이언트 6](#_Toc57678181)

[B. 대기실 (Waitingroom Scene) 7](#_Toc57678182)

[B-1. 서버 7](#_Toc57678183)

[B-2. 클라이언트 7](#_Toc57678184)

[C. 인게임 (Stage Scene) 8](#_Toc57678185)

[C-1. 서버 9](#_Toc57678186)

[C-2. 클라이언트 9](#_Toc57678187)

[A. 로비 (Lobby Scene) & 대기실 (Waitingroom Scene) <수정> 10](#_Toc57678188)

[A-1. 서버 10](#_Toc57678189)

[A-2. 클라이언트 10](#_Toc57678190)

[B. 인게임 (Stage Scene) <수정> 11](#_Toc57678191)

[B-1. 서버 12](#_Toc57678193)

[B-2. 클라이언트 12](#_Toc57678194)

[3. LOW-LEVEL 13](#_Toc57678195)

[4. 스레드 구조 17](#_Toc57678196)

[5. 스레드 동기화 19](#_Toc57678197)

[6. 팀원 별 역할분담 19](#_Toc57678198)

[7. 개발 일정 20](#_Toc57678199)

1. 게임 개요
2. 게임 컨셉

* 탱크를 조종하여 상대 탱크와 싸우는 멀티 게임

1. 게임 정보

|  |  |
| --- | --- |
| 제목 | 배틀 탱크 |
| 장르 | 슈팅 |
| 플랫폼 | PC |
| 인원 | 3인 |
| 개발 환경 | WinAPI  Visual Studio 2019  TCP/IP  Windows Socket 프로그래밍 |

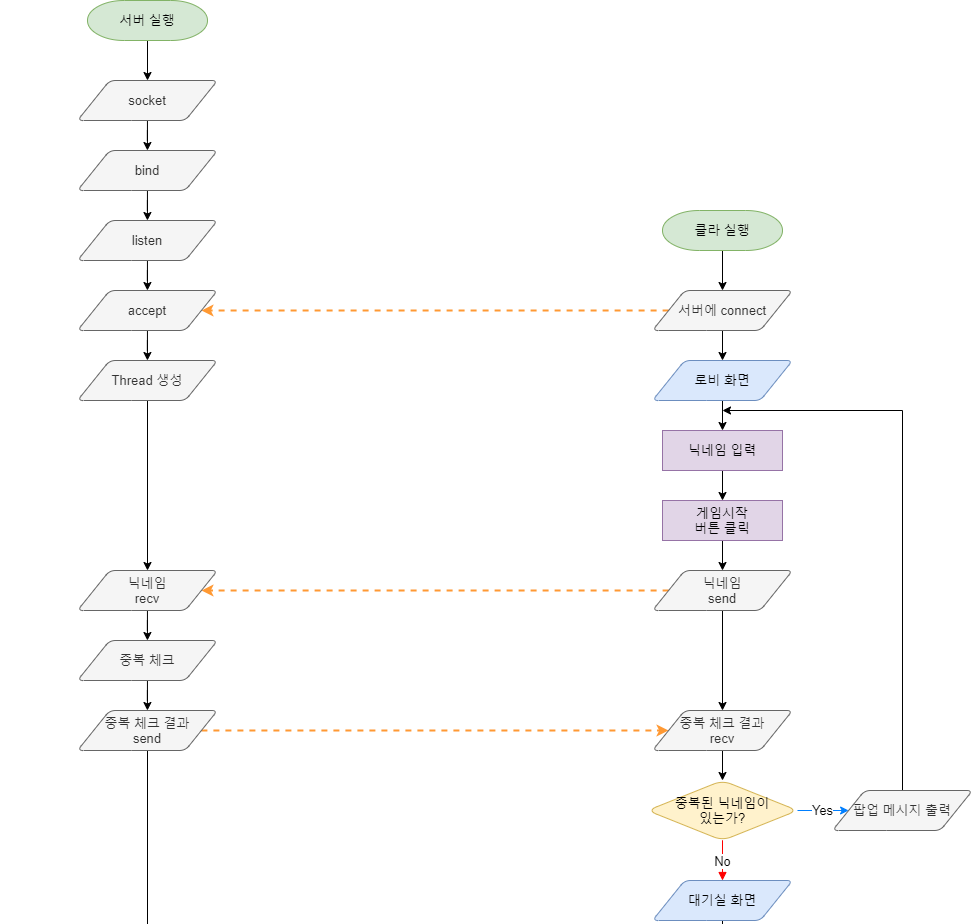
1. 조작 방법

|  |  |
| --- | --- |
| WASD | 상하좌우 이동 |
| 마우스 이동 | 총구 회전 |
| 마우스 좌클릭 | 총탄 발사 |

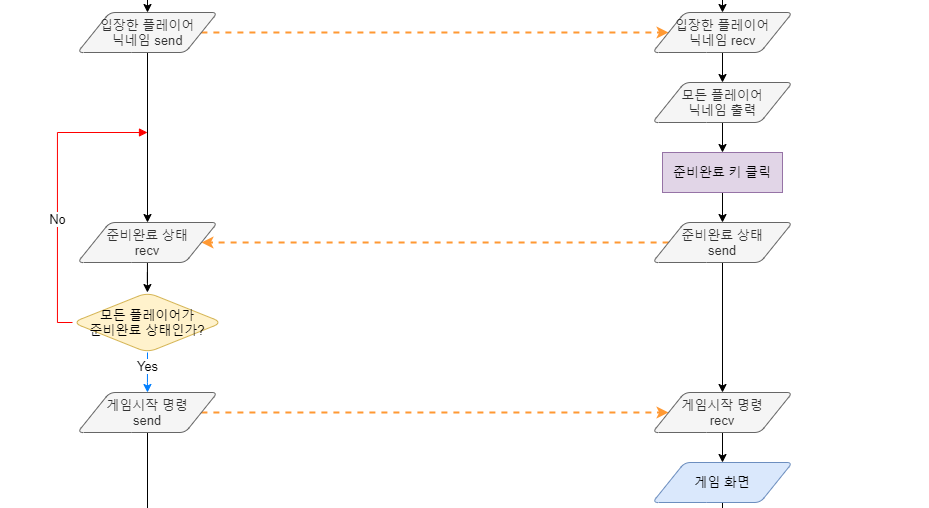
1. 게임 플레이

* 플레이어 3인이 게임을 진행한다.
* WASD로 움직이고 마우스를 움직여 회전할 수 있으며, 마우스 좌클릭을 하면 탱크가 향한 방향으로 총탄을 발사한다.
* 총탄은 화면 밖으로 나가거나 플레이어나 장애물에 충돌하면 사라진다.
* 플레이어가 상대방의 총탄에 닿으면 HP가 감소하며 HP가 0이 되면 죽게 된다.

1. HIGH-LEVEL
2. ~~로비 (Lobby Scene) <기존>~~



1. ~~서버~~
2. ~~서버를 실행 후, listening 상태에서 클라이언트의 connect 요청을 받으면 accept를 하고 스레드를 생성한다.~~
3. ~~클라이언트에서 닉네임 데이터를 받아서 중복되어 있는지 확인하고 그 결과값을 다시 클라이언트에 전송한다.~~
4. ~~클라이언트~~
5. ~~클라이언트를 실행 후, 서버에 connect한다.~~
6. ~~로비화면에서 닉네임을 입력하고 게임시작 버튼을 클릭하면 입력한 닉네임 데이터를 서버에 전송한다.~~
7. ~~서버에서 닉네임 중복 체크 결과값을 받아서 중복된 닉네임이 있을 경우 팝업 메시지를 출력하고 없을 경우 대기실 씬으로 전환한다.~~
8. ~~대기실 (Waitingroom Scene) <기존>~~

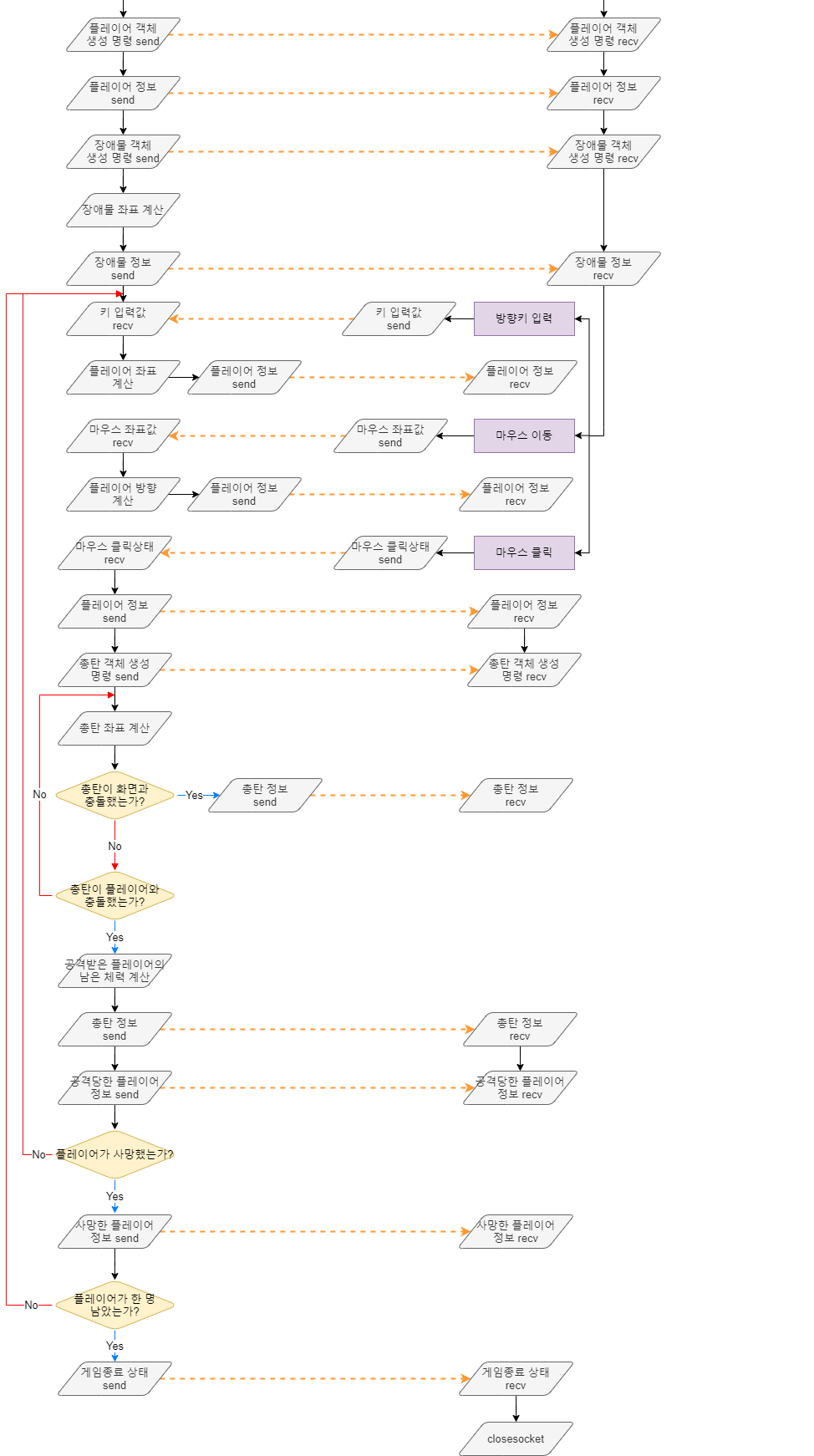


## ~~서버~~

1. ~~새로 입장한 플레이어가 있을 경우 모든 플레이어의 닉네임 데이터를 모든 클라이언트에 전송한다.~~
2. ~~클라이언트에서 플레이어의 준비 상태값을 받아서 모든 플레이어가 준비완료 상태이면 클라이언트에 게임시작 명령을 전송한다.~~

## ~~클라이언트~~

1. ~~서버에서 모든 플레이어의 닉네임 데이터를 받아서 출력한다.~~
2. ~~준비완료 키를 클릭하면 해당 플레이어의 현재 준비 상태값을 서버에 전송한다.~~
3. ~~서버에서 게임시작 명령을 받으면 Stage 씬으로 전환한다.~~
4. ~~인게임 (Stage Scene) <기존>~~

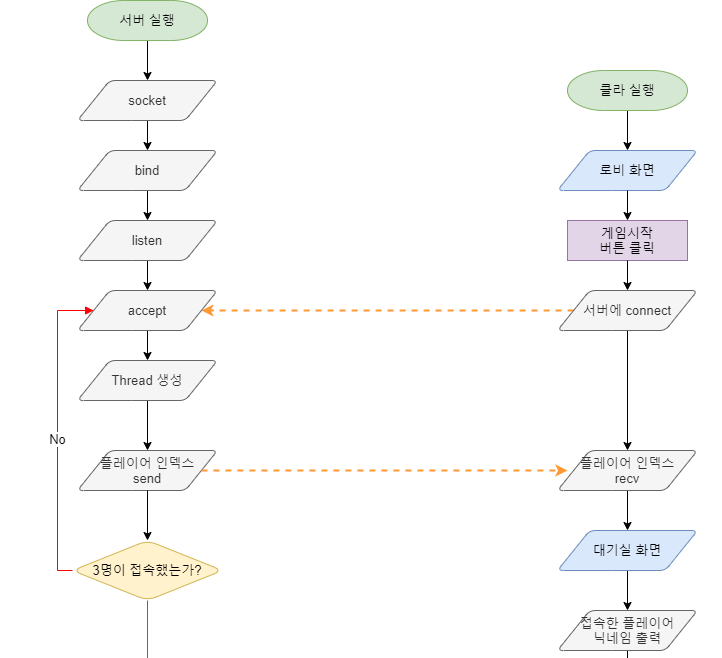


## ~~서버~~

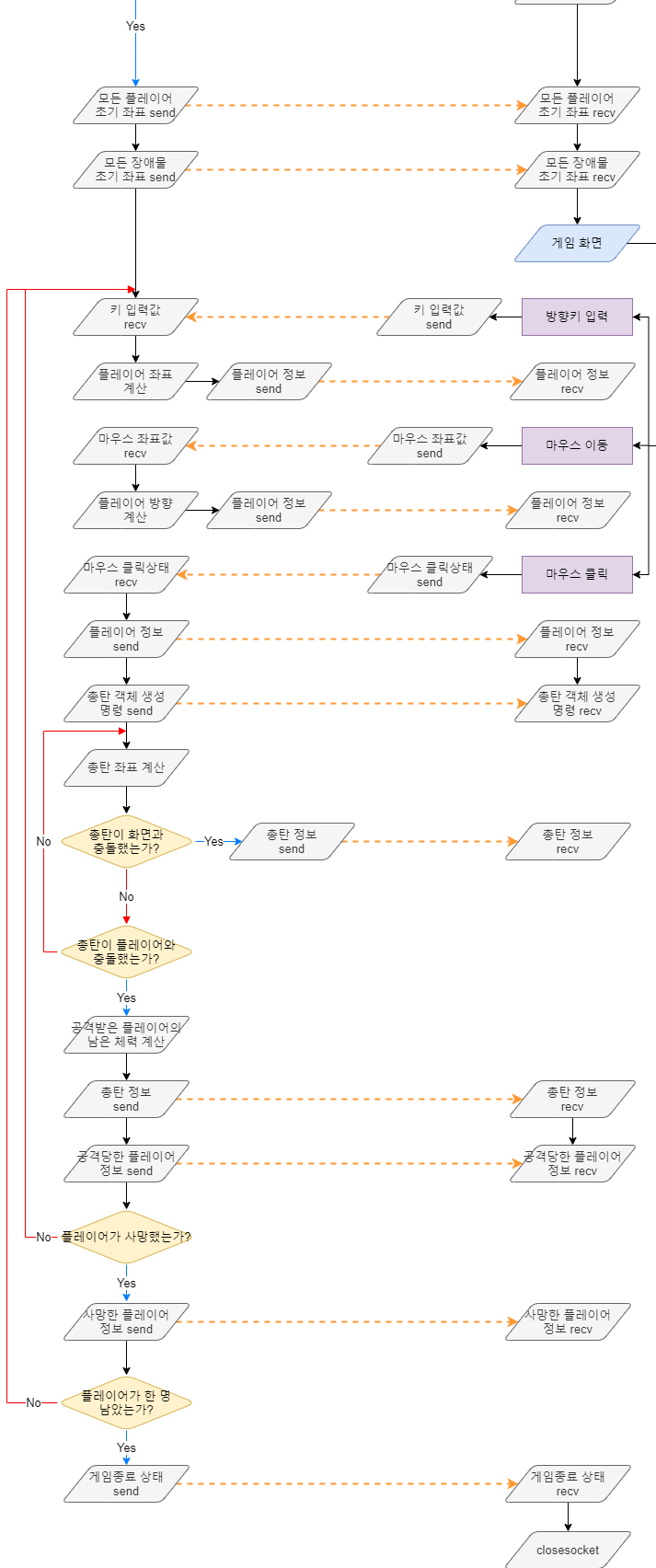
1. ~~초기값이 들어있는 모든 플레이어와 장애물 정보 구조체를 모든 클라이언트에 전송한다.~~
2. ~~클라이언트에서 입력값을 받아서 플레이어의 좌표나 방향 등을 연산 후, 동기화된 플레이어 정보 구조체를 모든 클라이언트에게 다시 전송한다.~~
3. ~~마우스 클릭 상태일 경우 초기값이 들어있는 총탄 정보 구조체를 모든 클라이언트에 전송한다. 총탄이 플레이어나 화면에 충돌하기 전까지 계속 총탄의 좌표를 연산 후, 동기화된 총탄 정보 구조체를 모든 클라이언트에게 전송한다.~~
4. ~~2-3번의 과정을 반복하고, 플레이어가 한 명 남으면 클라이언트에 게임종료 명령을 전송한다.~~

## ~~클라이언트~~

1. ~~서버에서 모든 플레이어와 장애물 정보 구조체를 받아서 객체를 생성하고 초기화한다.~~
2. ~~유저가 키보드와 마우스로 입력한 값을 서버에 전송한다.~~
3. ~~동기화된 플레이어 정보 구조체를 받아서 다시 그려준다.~~
4. ~~동기화된 총탄 정보 구조체를 받아서 다시 그려준다.~~
5. ~~서버에서 게임종료 명령을 받으면 일정 시간 후~~ ~~게임을 종료한다.~~
6. 로비 (Lobby Scene) & 대기실 (Waitingroom Scene) <수정>



1. 서버
2. 서버를 실행 후, listening 상태에서 클라이언트의 connect 요청을 받으면 accept를 하고 스레드를 생성한다.
3. 새로 입장한 플레이어가 있을 경우 모든 플레이어의 index를 모든 클라이언트에 전송한다.
4. 3명이 클라이언트가 accept할 경우 플레이어와 장애물의 초기 좌표를 계산한다.
5. 클라이언트
6. 클라이언트를 실행 후 로비화면에서 게임시작 버튼을 클릭하면 서버에 connect한다.
7. 서버에서 플레이어의 고유 index를 받고 대기실 씬으로 전환한다.
8. 대기실 씬에서 플레이어가 새로 접속할 때마다 접속한 모든 플레이어들의 번호를 출력한다.
9. 인게임 (Stage Scene) <수정>



## 서버

1. 초기값이 들어있는 모든 플레이어와 장애물 정보 구조체를 모든 클라이언트에 전송한다.
2. 클라이언트에서 입력값을 받아서 플레이어의 좌표나 방향 등을 연산 후, 동기화된 플레이어 정보 구조체를 모든 클라이언트에게 다시 전송한다.
3. 마우스 클릭 상태일 경우 초기값이 들어있는 총탄 정보 구조체를 모든 클라이언트에 전송한다. 총탄이 플레이어나 화면에 충돌하기 전까지 계속 총탄의 좌표를 연산 후, 동기화된 총탄 정보 구조체를 모든 클라이언트에게 전송한다.

## 클라이언트

1. 서버에서 모든 플레이어와 장애물 정보 구조체를 받아서 객체를 생성하고 초기화한다.
2. 유저가 키보드와 마우스로 입력한 값을 서버에 전송한다.
3. 동기화된 플레이어 정보 구조체를 받아서 다시 그려준다.
4. 동기화된 총탄 정보 구조체를 받아서 다시 그려준다.
5. 플레이어가 한 명 남을 경우 승리한 플레이어를 출력한다.
6. LOW-LEVEL
7. 구조체

~~[방향 구조체]~~

~~struct Direction {~~

~~float dirx;~~

~~float diry;~~

~~}DIR;~~

[고정 길이 정보 구조체]

typedef struct DataInfo {

char infoindex; //패킷 타입

char datasize; //패킷 크기

};

[입력 키 정보 구조체]

typedef struct KeyInfo {

char pIndex; //플레이어를 구분하기 위한 인덱스

bool w; //w 이동

bool a; //a 이동

bool s; //s 이동

bool d; //d 이동

};

[마우스 정보 구조체]

typedef struct MouseInfo {

char pIndex; //플레이어를 구분하기 위한 인덱스

float mx; //마우스 x좌표

float my; //마우스 y좌표

bool isClicked; //마우스 클릭 여부(총탄 발사)

};

[플레이어 정보 구조체]

typedef struct PlayerInfo {

char pIndex; //플레이어를 구분하기 위한 인덱스

~~char inputkey; //입력한 키 값~~

~~float mx; //마우스의 X좌표~~

~~float my; //마우스의 Y좌표~~

~~BOOL isClicked; //마우스의 클릭 상태~~

float x; //플레이어의 X좌표

float y; //플레이어의 Y좌표

char nowHp; //플레이어의 남은 체력

char maxHp; //플레이어의 최대 체력

~~float atk; //플레이어의 공격력~~

float gunx1; //총구의 왼쪽 X좌표

float guny1; //총구의 왼쪽 Y좌표

float gunx2; //총구의 오른쪽 X좌표

float guny2; //총구의 오른쪽 Y좌표

}

[총탄 정보 구조체]

typedef struct BulletInfo {

char pIndex; //플레이어를 구분하기 위한 인덱스

char bindex; //총탄을 구분하기 위한 인덱스

bool isDead;

float x; //총탄의 X좌표

float y; //총탄의 Y좌표

float dx; //총탄의 X방향 벡터

float dy; //총탄의 Y방향 벡터

};

[장애물 정보 구조체]

typedef struct ObstacleInfo{

float x; //장애물의 X좌표

float y; //장애물의 Y좌표

}

1. 함수
2. 서버

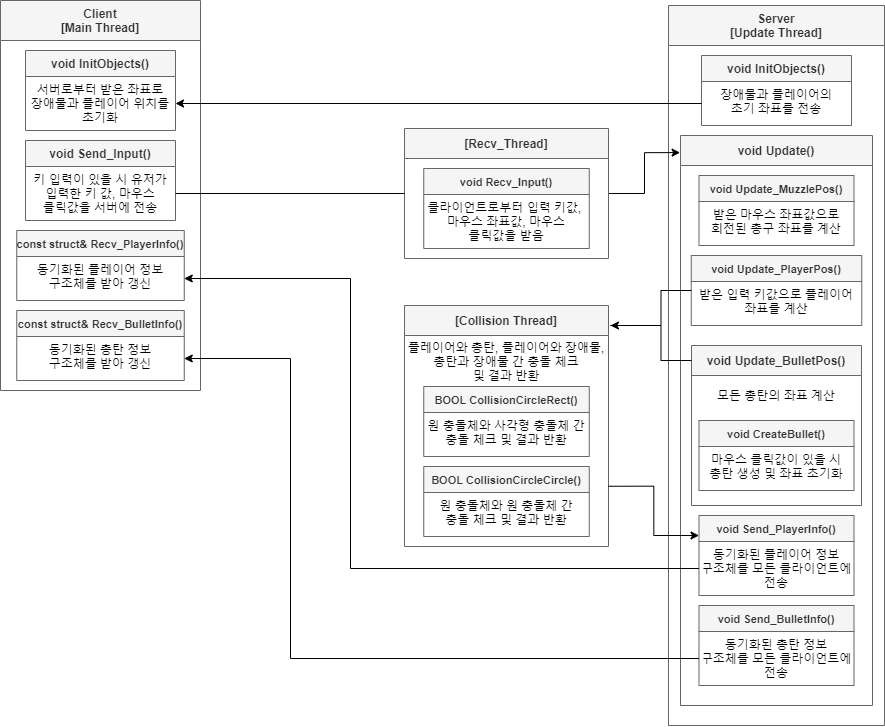
* int InitServer() : 소켓을 생성하고 소켓의 지역 IP 주소와 지역 포트번호를 결정한 후 포트 상태를 LISTENING으로 변경한다.
* void RunServer() : 접속한 클라이언트의 전용 소켓을 생성하고 접속한 클라이언트에게 플레이어 고유 인덱스를 전송한다. 클라이언트가 3명 접속하면 플레이어와 장애물 패킷에 초기 좌표를 넣어주고 모든 클라이언트에 전송한다.
* void InitObjects() : 플레이어와 장애물의 초기 좌표를 모든 클라이언트에게 전송한다.
* static void Send\_PlayerInfo() : 모든 클라이언트에게 플레이어 정보 패킷을 전송한다.
* static void Send\_ObstacleInfo() : 모든 클라이언트에게 장애물 정보 패킷을 전송한다.
* static void Send\_CreateBullet(BulletInfo bInfo) : 모든 클라이언트에게 생성할 총탄 정보 패킷을 전송한다.
* static void Send\_BulletInfo(BulletInfo bInfo) : 모든 클라이언트에게 총탄 정보 패킷을 전송한다.
* static void Recv\_Info(LPVOID arg) : 클라이언트로부터 정보 패킷을 받는다.
* static void UpdatePlayerPos(KeyInfo kInfo) : 입력된 키 값에 따라 플레이어 좌표를 계산한다.
* static void UpdateMuzzleInfo(MouseInfo mInfo) : 입력된 마우스 값에 따라 총구 좌표를 계산한다.
* static void CreateBullet(char pIndex) : 생성할 총탄의 좌표를 계산한다.
* static void UpdateBulletInfo(BulletInfo bInfo) : 총탄이 이동된 좌표를 계산한다.
* static void CollisionBullet(BulletInfo bInfo) : 총탄과 플레이어, 총탄과 장애물 간 충돌을 검사한다.
* static void CollisionPlayerObstacle(char cIndex) : 플레이어와 장애물 간 충돌을 검사한다.
* DWORD \_\_stdcall FrameThread(LPVOID arg) : 프레임과 타이머를 갱신한다.
* static DWORD \_\_stdcall UpdateThread(LPVOID arg) : 클라이언트로부터 데이터를 받고 해당 데이터를 갱신한 후 패킷 정보를 클라이언트에 전송한다.
* ~~char\* Recv\_UserID(): 클라이언트에서 보낸 닉네임 데이터를 받는다.~~
* ~~BOOL CheckUserID(char\* n) : 닉네임이 중복되어 있는지 확인한다.~~
* ~~void Send\_CheckedResult() : 닉네임이 중복되었을 경우 입력한 닉네임에 넘버링을 하여 클라이언트에 전송한다.~~
* ~~void Send\_AllUserIDs\_ToAll() : 모든 플레이어의 닉네임 데이터를 모든 클라이언트에 전송한다.~~
* ~~void GameSceneChange() : 씬을 전환하라는 명령을 모든 클라이언트에 전송한다. (게임시작, 게임종료, 씬 전환)~~
* ~~DWORD WINAPI CollisionThread(LPVOID arg) : 이벤트 함수를 통해 플레이어와 총탄, 플레이어와 장애물, 총탄과 장애물 간 충돌을 검사하고 후처리를 진행한다.~~

1. 클라이언트

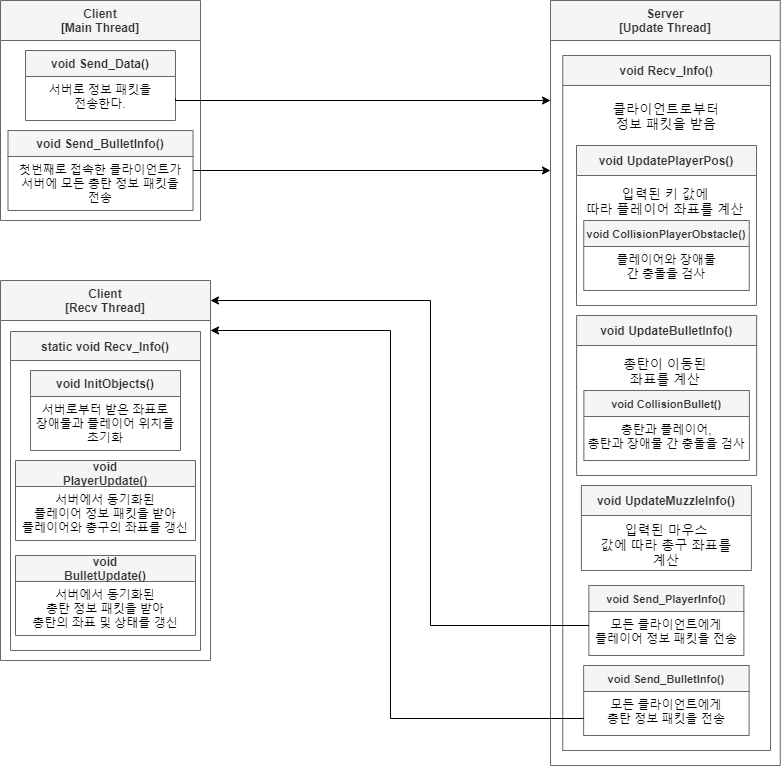
* int WsaStart() : 윈속을 초기화한다.
* void CreateSocket() : 소켓을 생성한다.
* void ConnectToServer() : 서버와 연결한다.
* void CreateRecvThread() : 서버와 통신할 RecvThread를 생성한다.
* void SetServerIP() : 서버의 IP주소를 입력한다.
* void SendData(char dataindex, LPVOID packet) : 서버로 정보 패킷을 전송한다.
* static void RecvInfo() : 서버로부터 패킷 정보를 받는다.
* void DisConnectServer() : 서버와의 연결을 끊는다.
* void PlayerUpdate(PlayerInfo pInfo) : 서버에서 동기화된 플레이어 정보 패킷을 받아 플레이어와 총구의 좌표를 갱신한다.
* void BulletUpdate(BulletInfo bInfo) : 서버에서 동기화된 총탄 정보 패킷을 받아 총탄 좌표 및 상태를 갱신한다.
* void Send\_BulletInfo() : 첫번째로 접속한 클라이언트가 서버에 모든 총탄 정보 패킷을 전송한다.
* int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPWSTR lpCmdLine, \_In\_ int nCmdShow) : 클라이언트의 메인 함수로, 모든 객체들을 갱신하고 렌더링한다.
* static DWORD \_\_stdcall RecvFromServer(void\*) : 서버로부터 정보 패킷을 받는다.
* ~~void Send\_UserID(const char\* n) : 입력한 닉네임 데이터를 서버에 전송한다.~~
* ~~BOOL Recv\_CheckedResult(): 서버에서 보낸 닉네임 중복 체크 결과값을 받는다.~~
* ~~void Recv\_AllUserIDs() : 서버에서 보낸 모든 플레이어의 닉네임 데이터를 받는다.~~
* ~~void SceneChange() : 서버에서 씬 전환 명령을 받고 씬을 전환한다.~~

1. 스레드 구조

<기존 구조>



<수정된 구조>



1. 스레드 동기화

<클라이언트>

MainThread에서 모든 객체를 Update할 때 각 객체의 상태를 검사한 후 해당 객체를 삭제하는 과정에서 RecvThread에서 총탄 객체를 받아오는 과정과 충돌하여 총탄이 제대로 삭제가 되지 않고, 인덱스가 밀리는 현상이 발생하였습니다.

이 문제를 해결하기 위해 Mutex를 사용하여 하나의 스레드에서 작업 중일 시 다른 스레드에서 해당 객체에 접근하지 못하도록 하였습니다.

<서버>

플레이어와 장애물, 총탄 정보를 전역 변수로 설정하는데 각 클라이언트의 UpdateThread에서 데이터를 계산 및 업데이트하는 과정, 즉 한 프로세스에 속한 여러 스레드가 해당 공유자원에 접근하기 때문에 임계영역을 이용해 동기화를 하는 것이 알맞다고 생각하였습니다.

~~충돌 스레드는 총탄 객체가 존재하지 않거나 플레이어의 키 입력이 없을 때는 충돌 검사를 할 필요가 없기 때문에 플레이어와 총탄 정보에 변화가 있을 때만 이벤트 함수를 이용해 깨워서 사용하는 것이 더 효율적이라고 판단하였습니다.~~

모든 총탄 객체의 정보를 서버에서 가지고 있지 않고 클라이언트에서 넘겨줄 때마다 좌표 계산 및 충돌 검사를 하여 바로 모든 클라이언트에게 다시 보내는 구조이기 때문에 서버에서 충돌 스레드를 생성하지 않았습니다.

1. 팀원 별 역할분담

* 김지은 : 클라이언트 제작 및 서버와 통신
* 김수민, 전혜진 : 게임 서버 제작 및 클라이언트와 통신

1. 개발 일정

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 월 | 화 | 수 | 목 | 금 | 토 | 일 |
|  |  |  |  | 10/29 | 10/30 | 10/31 | 11/01 |
| 김지은 |  |  |  | 프로젝트 기획서 작성 | | | |
| 김수민 |  |  |  |
| 전혜진 |  |  |  |
|  | 11/02 | 11/03 | 11/04 | 11/05 | 11/06 | 11/07 | 11/08 |
| 김지은 | 리소스  제작 | 클라이언트 프레임워크 설계 | | 클라이언트 프레임워크 제작 | |  | 중간 점검 |
| 김수민 | 서버 프레임워크 설계 | | 서버 프레임워크 제작 | |
| 전혜진 |
|  | 11/09 | 11/10 | 11/11 | 11/12 | 11/13 | 11/14 | 11/15 |
| 김지은 | 클라이언트  객체/구조체 정리 | 프로젝트 기획서 수정 |  |  | 클라이언트 connect |  | 스레드 구조, 구조체 수정 |
| 김수민 | 서버  객체/구조체 정리 |  |
| 전혜진 | 서버 accept |
|  | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 | 11/21 | 11/22 |
| 김지은 | 플레이어 인덱스 recv  키 입력 값 send  플레이어 패킷 recv |  | 마우스 좌표 값 send  플레이어 패킷 recv | |  |  | 중간 점검 |
| 김수민 | 플레이어 좌표 계산 | 플레이어, 장애물 초기 좌표 계산 | |
| 전혜진 | 플레이어 인덱스 send  키 입력 값 recv  플레이어 패킷 send | 마우스 좌표 값 recv  총구 좌표 계산 및 회전  플레이어 패킷 update, send | |
|  | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 | 11/28 | 11/29 |
| 김지은 | 총탄 객체 생성  총탄 패킷 send  장애물 패킷 recv  프레임 고정 | |  | 대기실 씬 구현  총탄 패킷 recv  총탄 객체 삭제 | |  | 중간 점검 |
| 김수민 |  | | 총탄, 장애물 충돌 체크 | |
| 전혜진 | 총탄 패킷 recv, update, send  장애물 패킷 send  총탄 좌표 계산  프레임 고정 | | 플레이어, 장애물 /  플레이어, 총탄 충돌 체크  플레이어, 총탄 패킷 update, send | |
|  | 11/30 | 12/1 | 12/2 | 12/3 | 12/4 | 12/5 | 12/6 |
| 김지은 | 최종 기획서 수정 |  | 테스트 | 버그 수정 | |  | 중간 점검 |
| 김수민 |
| 전혜진 |
|  | 12/07 | 12/08 |
| 김지은 | 레포트 작성 | 결과 제출 및 시연 |
| 김수민 |
| 전혜진 |