**네트워크 게임 프로그래밍**

**프로젝트 결과 보고서**

**-Shooting Nemo-**

**게임공학과 2015182042 최현욱**

**게임공학과 2016182027 이미륵**

**목차**

1. **애플리케이션 기획**

* 게임 개요
* 게임 방법
* 게임 진행 흐름
* 개발 목적

1. **High-Level**

* Flow Chart
* Server
* Client

1. **Low-Level**

* Server
* Client

1. **개발환경**

* Platform
* Protocol

1. **팀원 별 역할분담**
2. **개발 일정**
3. **애플리케이션 기획**

* **게임 이름**

Shooting Nemo

* **장르**

슈팅, 대전

* **게임 방법**

**진행 방식**

각 플레이어가 게임을 실행하면 무기와 서버 주소를 입력하는 콘솔 창이 나옵니다. 입력 후 접속하면 인게임 화면이 나오고 준비를 하며 두 플레이어가 접속해 준비 완료가 될 때까지 기다립니다. 준비 완료가 되면 게임이 바로 시작됩니다. 게임이 시작되면 각 플레이어가 양끝에서 시작하고 2명의 플레이어가 서로를 향해 총알을 발사하여 상대방 플레이어를 맞추는 게임입니다. 만약 플레이어가 상대방 총알에 맞는다면 그 플레이어의 HP가 감소하며 HP가 0이 된 플레이어는 사망하고 게임에서 패배하게 됩니다.

**조작키**

**W**

**A**

**S**

**D**

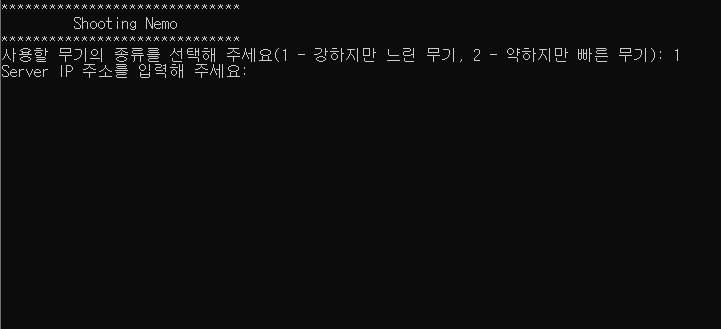


총알발사

W / A / S D : 플레이어 이동 **( 기존 : 방향키 )**

마우스 왼쪽버튼 클릭: 공격 **( 기존 : A키 )**

* **게임 진행 흐름 (스크린 샷 사용)**

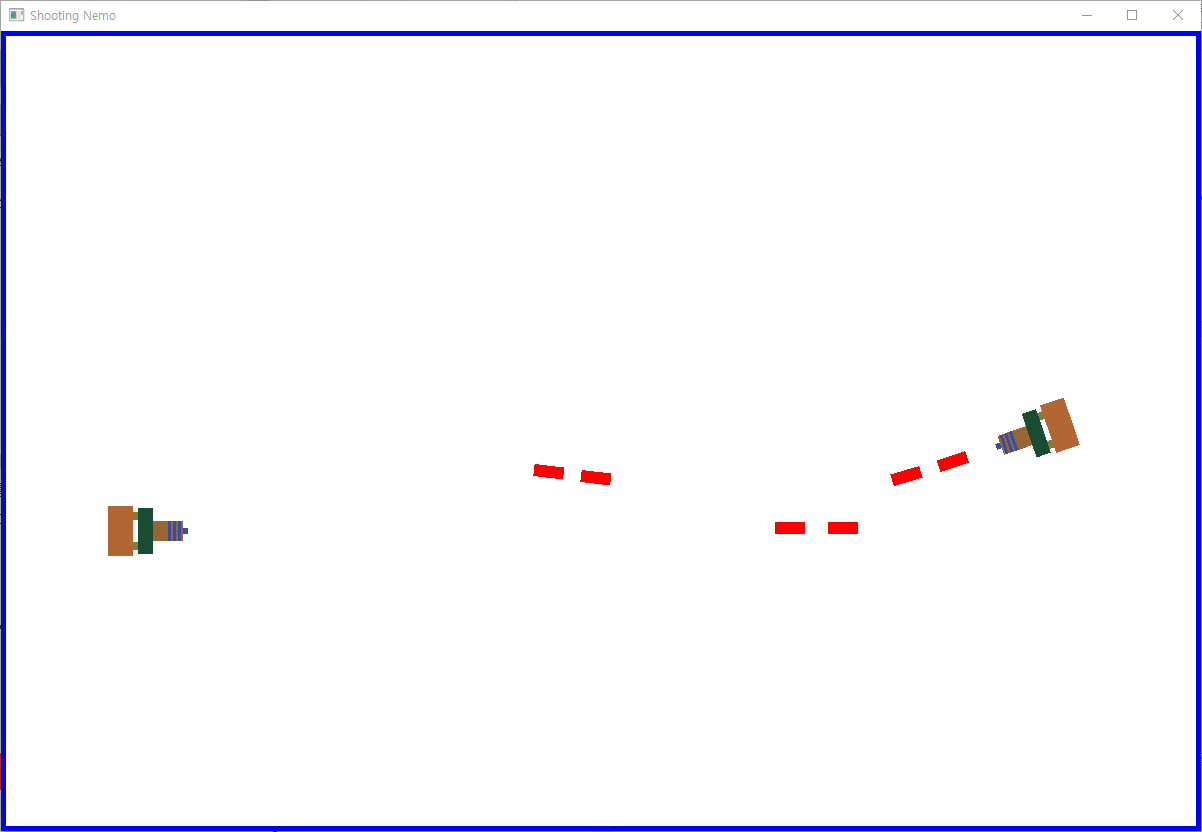
**title 화면**게임을 실행하면 콘솔 화면이 나옵니다. 처음에 무기의 종류를 선택할 수 있으며 선택 후 Server IP 주소를 입력하면 인게임으로 들어갑니다.

**대기 화면**



두 플레이어가 들어올 때까지 움직일 수 없으며 ‘r’ 키를 눌러 준비 완료를 하고, 두 플레이어가 들어와 준비 완료가 된다면 게임이 시작됩니다.

**게임 플레이**



두 명의 플레이어가 양 끝에서 시작하며 서로 상대방을 향해 총알을 쏘아 상대방의 HP가 0이될 때까지 게임이 진행됩니다.

**결과 화면**



한쪽의 플레이어가 HP가 0이 되면 게임은 끝나게 되며 결과 화면에 승패결과를 보여줍니다. 결과 확인 후 ‘r’키를 누르면 다시 준비 화면으로 돌아갑니다..

* **개발 목적**

2인용 슈팅 대전 게임 개발

MultiThread 활용 능력 향상

서버/클라이언트 동기화 능력 향상

1. **High-Level 디자인**

* **Flow Chart**

WSAStartup()

CreateSocket()

bind()

listen()

accept

WSAStartup()

CreateSocket()

connect()

send()

recv()

closesocket()

closeHandle()

closesocket()

WSACleanup()

WSACleanup()

CreateThread

RecvThread

Update And

Send Thread

CreateThread

NO

YES

send()

physics

Hp가 0인가?

게임 종료

접속 종료?

NO

NO

YES

YES

recv()

두 클라이언트가 접속을 했는가?

게임 시작

게임 진행 FlowChart

게임 시작

주소 입력

무기 선택

NO

YES

NO

결과 화면

YES

두 클라이언트가

접속을 했는가?

플레이어의 HP가 0이 되었는가?

무기 선택

대기 화면

플레이

R키 입력

기존 : 두 플레이어가 접속 후 무기 선택을 합니다. 결과 화면후 대기화면으로 전환됩니다.

변경: 전과 달리 무기 선택을 주소입력하는 곳에서 같이합니다. 또한 결과 화면이 나오면 ‘r’키를 통해 다시 대기화면이 됩니다.

* **Server**

서버 실행 순서

1. 서버 실행 후, listen()을 통해 클라이언트의 접속을 받을 수 있는 상태로 전환한다.
2. 클라이언트의 connect요청이 들어오면 해당 요청을 accept 한 후 RecvThread를 생성한다.
3. 2개의 클라이언트가 접속하고 클라이언트들의 준비상태를 확인한다. 모두 준비상태가 되면 플레이어의 초기 위치를 클라이언트들에게 전송한 후, UpdateAndSendThread를 생성한다.
4. RecvThread는 클라이언트에게서 패킷을 받는다. 패킷의 타입에 따라 패킷을 처리한 후 처리 결과를 모든 클라이언트들에게 전송한다.
5. UpdateAndSendThread는 총알의 위치를 업데이트하고 총알과 플레이어의 충돌을 확인하고 처리한 뒤 그 결과를 클라이언트들에게 전송한다.
6. 한 명의 플레이어가 남을 때까지 4~5의 과정을 진행한다. 한 명의 플레이어가 남으면 해당 결과를 모든 클라이언트에게 보내준다.

* **Client**

클라이언트 실행 순서

1. 클라이언트 실행 후 서버 주소 입력 및 원하는 무기 선택
2. 서버에 connect 한다.
3. 게임 시작
4. Input data를 서버에 전송
5. 서버에서 업데이트 된 후의 데이터(플레이어 위치, 총알의 위치, hp, 충돌처리 등)를 받는다.
6. 한 플레이어의 hp가 0이 될 때까지 4~5 반복하고, 0이되면 결과를 띄워준다.
7. **Low-Level 디자인**

**■ Server**

**- Server.cpp**

**void CollisionPlayerwithMap(int id)**

플레이어와 맵과의 충돌을 처리하는 함수입니다.

**void CollisionBulletwithPlayer(int id)**

총알과 상대 플레이어와의 충돌처리 하는 함수, Bullet의 데미지만큼 Hp를 감소하며 플레이어의 승패를 결정해줍니다.

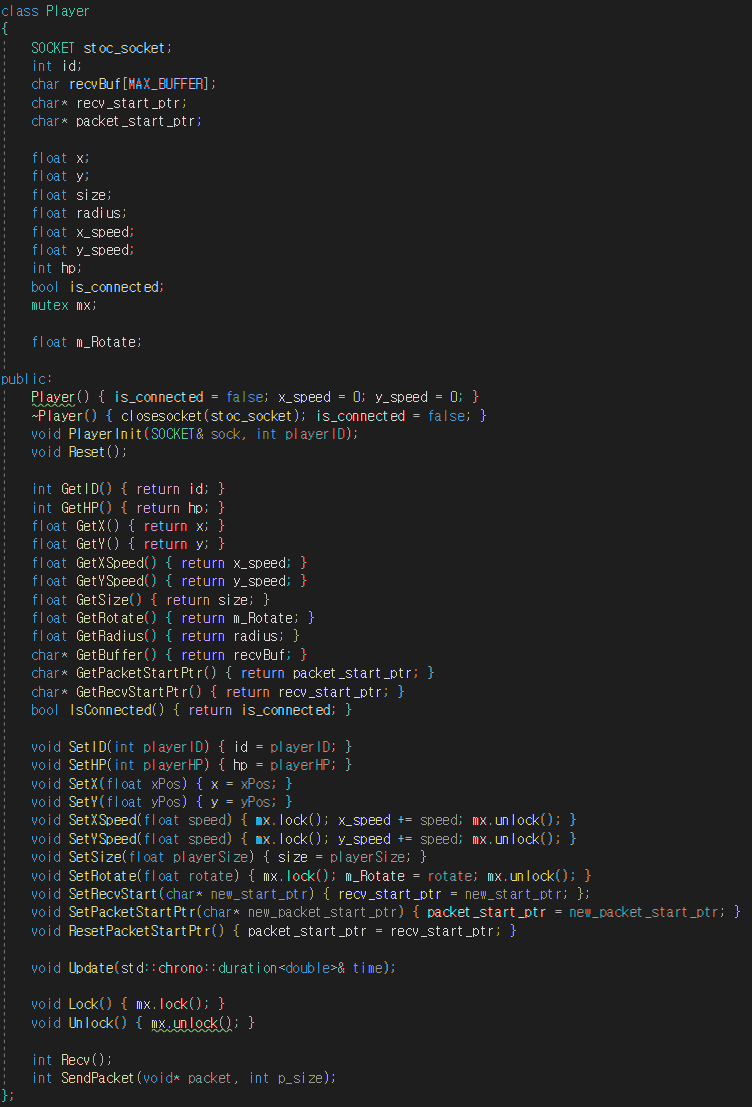
**void process\_packet(int id)**

**void UpdateAndSendThread()**

게임 월드의 상태(플레이어의 정보, 총알의 위치 등)을 업데이트하고 모든 클라이언트에게 전송합니다.

**void RecvThread(int id)**

클라이언트마다 하나씩 생성되며, 해당 클라이언트가 보내는 데이터를 수신합니다. 받은 데이터를 처리하고 모든 클라이언트에게 처리한 데이터를 전송합니다.

**- PlayerClass**

플레이어의 위치 정보, 사이즈, hp 등등 을 멤버 변수로 가집니다.

멤버 변수의 값을 추출할 수 있는 Get\*()함수를 가집니다.

멤버 변수의 값을 변경할 수 있는 Set\*()함수를 가집니다.

PlayerInit() 함수로 플레이어의 초기값을 정해줍니다.

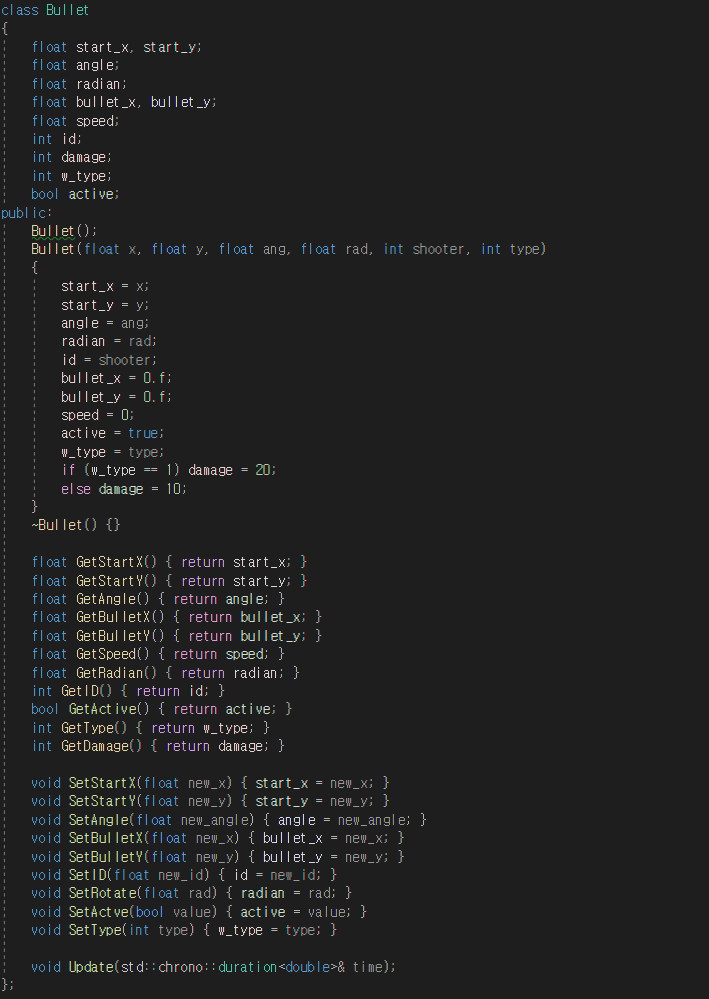
Rest() 함수로 게임을 재시작 했을 때 위치값과 hp값을 초기화 해줍니다.

Update() 함수로 플레이어의 속도와 시간을 곱하여 위치를 갱신해줍니다.

Recv() 함수로 ..???

SendPacket() 함수로 ..???

**- BulletClass**



Update() 함수로 총알의 위치를 계속해서 갱신해줍니다.

**------------------------지울 것---------------------------------**

**• 플레이어의 위치 정보(x, y), 체력(hp)을 멤버 변수로 가진다.**

**• 멤버 변수의 값을 추출할 수 있는 Get\*()함수를 멤버 함수로 가진다.**

**• 멤버 변수의 값을 변경할 수 있는 Set\*()함수를 멤버 함수로 가진다.**

**- bullet class: 총알의 정보를 담을 Class**

**• 총알의 위치 정보(x, y), 총알의 공격력을 멤버 변수로 가진다.**

**• 멤버 변수의 값을 추출할 수 있는 Get\*()함수를 멤버 함수로 가진다.**

**• 멤버 변수의 값을 변경할 수 있는 Set\*()함수를 멤버 함수로 가진다.**

**• 플레이어와 충돌을 확인하는 CollisionCheck()함수를 멤버 함수로 가진다.**

**• 총알의 위치 정보를 Update하기 위한 Update()함수를 멤버 함수로 가진다.**

• **player 구조체의 경우 게임이 시작될 때 플레이어들의 초기 위치를 전송하는데 사용된다.**

**- RecvThread: 클라이언트마다 하나씩 생성되며, 해당 클라이언트가 보내는 데이터를 수신한다. 받은 데이터를 처리하고 모든 클라이언트들에게 처리한 데이터들을 전송한다.**

**• 모든 패킷 구조체의 2번째 변수는 패킷의 타입이므로, 수신한 데이터를 저장하는 배열의 1번 인덱스 값을 확인, 타입에 따라 분기하여 패킷을 처리한다.**

**• send\_\*\_packet()함수를 호출하여 클라이언트로 보낼 패킷에 값을 채워 넣는다. 이후 send\_packet()함수를 호출하여 실질적으로 클라이언트에게 전송한다.**

**- UpdateAndSendThread: 게임 월드의 상태(총알의 위치 등)을 업데이트 하고 클라이언트들에게 전송한다.**

**• 모든 총알 객체들에 대하여 Update()함수를 호출하여 총알의 위치 정보를 갱신한다.**

**• 모든 총알 객체들에 대하여 CollisionCheck()함수를 호출하여 플레이어와 충돌을 검사한다.**

**• 플레이어와 총알이 충돌했으면 플레이어의 hp를 총알의 공격력만큼 감소시킨다.**

**• 총알의 위치정보와 플레이어의 정보 변화를 클라이언트에게 전송한다.**

**------------------------지울 것---------------------------------**

**■ Protocol**

///////////////////////////////////////////////////////////////////////////////

#define ctos\_login 20

#define ctos\_shoot 21

#define ctos\_move 22

#define ctos\_ready 24

#define ctos\_weapon\_select 25

#define stoc\_new\_client 26

#define stoc\_bullets 27

#define stoc\_move 28

#define stoc\_hit 29

#define stoc\_ready 30

#define stoc\_gamestart 31

#define stoc\_weapon\_select 32

#define stoc\_world\_state 33

#define stoc\_login\_ok 34

#define ctos\_rotate 35

#define stoc\_gameend 36

///////////////////////////////////////////////////////////////////////////////

#define DIR\_UP 10

#define DIR\_DOWN 11

#define DIR\_LEFT 12

#define DIR\_RIGHT 13

#define DIR\_UP\_UP 14

#define DIR\_DOWN\_UP 15

#define DIR\_LEFT\_UP 16

#define DIR\_RIGHT\_UP 17

#define STRONG\_WEAPON 18

#define NORMAL\_WAEPON 19

///////////////////////////////////////////////////////////////////////////////

#define SERVER\_PORT 5959

#define MAX\_BUFFER 5120

///////////////////////////////////////////////////////////////////////////////

#define PLAYER\_X\_SPEED 300.f

#define PLAYER\_Y\_SPEED 300.f

#define BULLET\_SPEED 500.f

#define STRONG\_BULLET\_SPEED 350.f

#define BULLET\_SIZE\_X 15.f

#define BULLET\_SIZE\_Y 3.f

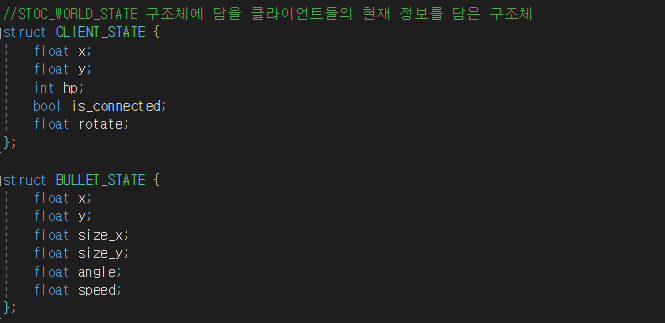
#define STRONG\_BULLET\_SIZE\_X 30.f

#define STRONG\_BULLET\_SIZE\_Y 6.f

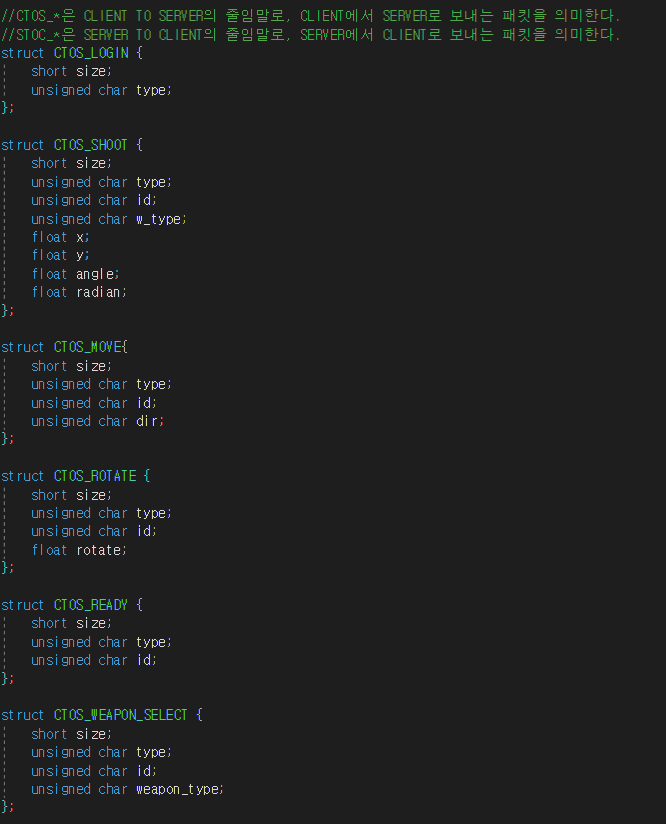
#define PLAYER\_SIZE 25.f

///////////////////////////////////////////////////////////////////////////////

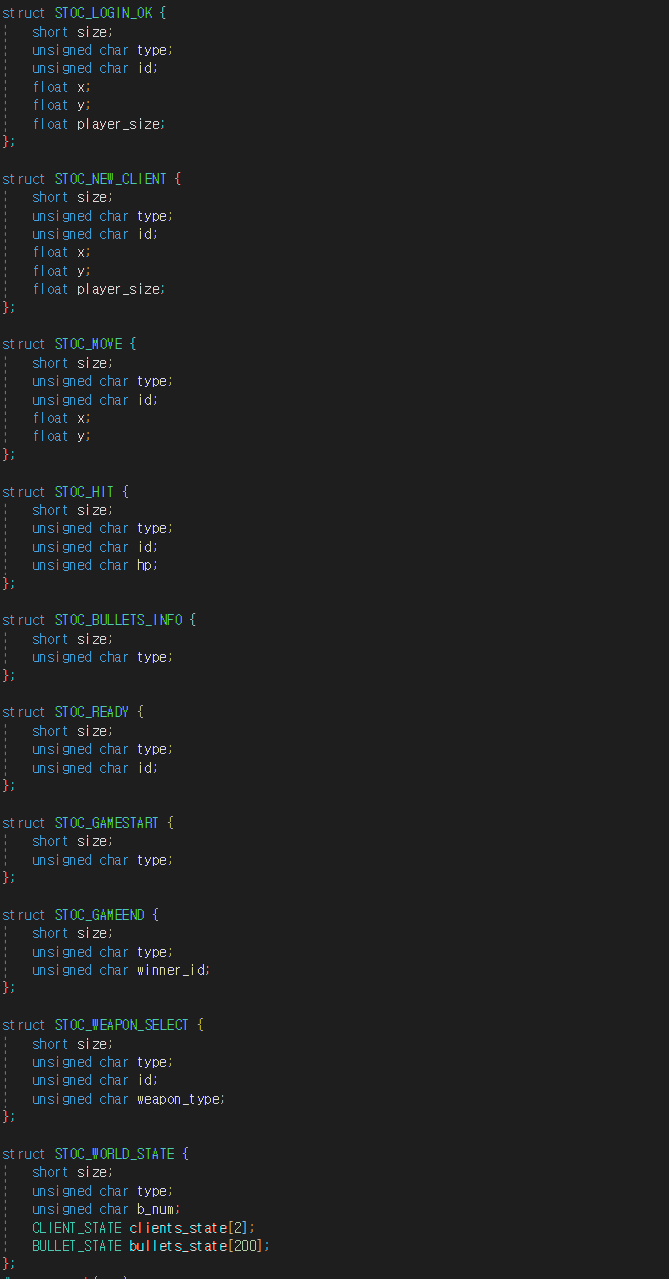
**구조체**



**Client To Server**



**Server To Client**



**■ Client**

**- Client.cpp**

**void process\_packet(char\* packet)**

패킷의 종류에 따라서 처리를 해주는 함수입니다.

**void procees\_recv(int recv\_bytes)**

서버로부터 오는 패킷을 받는 함수입니다.

**void Ready()**

ready를 그려주는 함수입니다.

**void Win()**

win을 그려주는 함수입니다.

**void Lose()**

lose를 그려주는 함수입니다.

**SenceClass: 타이틀, 준비, 인게임, 결과 화면에 관한 클래스**

**Struct ClientInfo: 클라이언트의 정보를 담을 구조체**

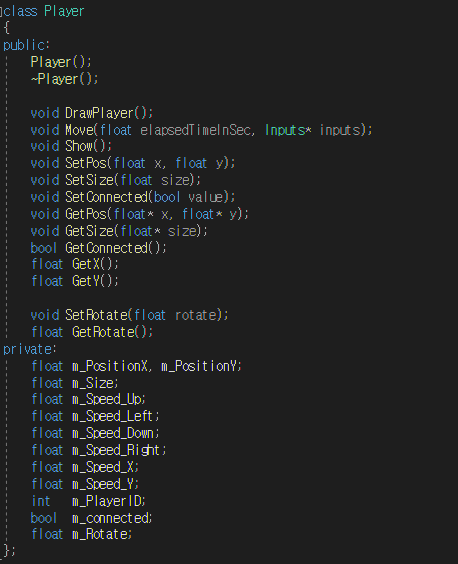
서버에게 전송할 데이터의 변수가 있는 구조체입니다. 서버와 클라이언트가 동일한 구조체를 가집니다.

**void SendDatatoServer(): 클라이언트의 정보를 서버에 전송하는 함수**

클라이언트에서 이루어진 작업들을 서버에 전송 처리를 하는 함수입니다.

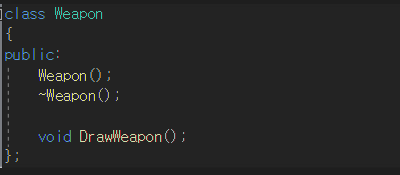
**void RecvDatatoServer(): 서버로부터 데이터를 얻는 함수**

서버로부터 오는 데이터를 받는 함수입니다. 기본적으로 플레이어의 좌표값, 충돌처리 등을 받습니다.



**PlayerClass: 플레이어 정보가 담긴 클래스**

**플**레이어 객체 생성을 위한 클래스이며 기본적으로 플레이어의 위치값, 사이즈, 스피드 등등 멤버 변수가 있습니다. 플레이어 이동이나, 총알 발사, 플레이어에 관한 처리들은 이 클래스 멤버 함수에 포함되어있습니다. 또한 플레이어의 외형을 그려주는 함수 또한 포함되어있습니다.

**WeaponClass: 무기 정보가 담긴 클래스**

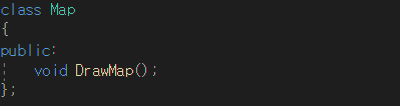
웨폰 클래스에는 총을 그려주는 함수가 있습니다.

**BulletClass: 총알 클래스**



총알 클래스에는 총알의 위치, 스피드, 각도 등이 변수로 있습니다. 화면에 총알을 그려줄 때 위치값을 사용하고 충돌처리 하는 과정에 있어서 필요로 합니다. 스피드 및 각도는 마우스를 화면에 클릭하여 각도를 구해 스피드 만큼 그 방향을 향하여 총알을 쏩니다.

**MapClass: 맵 클래스**



플레이어가 게임을 하는 맵을 관리하는 클래스입니다. 이 클래스안에는 맵을 그려주는 함수가 있습니다.

1. **개발 환경**

* **Platform**

**• Windows PC**

* **사용 API**

**• Open GL**

**• 윈도우 소켓 API**

* **프로토콜**

**• TCP**

1. **팀원 별 역할 분담**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 내용 | 최현욱 | 이미륵 |
| 기획 | 기획서 작성 | O | O |
| Server | Protocol.h | O | X |
|  | 서버 메인루프 작성(main함수) | O | O |
|  | RecvThread() | O | O |
|  | UpdateAndSendThread() | O | O |
|  | 충돌체크 및 처리 | O | O |
| Client | 맵 구현 | X | O |
|  | PlayerClass 구현 | O | O |
|  | WeaponClass 구현 | X | O |
|  | SecneClass 구현 | O | O |
|  | SendDatatoServer() | O | X |
|  | RecvDatatoServer() | O | X |

1. **개발 일정**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **월** | **화** | **수** | **목** | **금** | **토** | **일** |
|  | **11/02** | **11/03** | **11/04** | **11/05** | **11/06** | **11/07** | **11/08** |
| **최현욱** | 기획서 수정 | | | 기획서  검수 | 서버 메인 함수 작성 | 클라이언트 프레임 제작 | 개인 시간 |
| **이미륵** | 개인 시간 |
|  | **11/09** | **11/10** | **11/11** | **11/12** | **11/13** | **11/14** | **11/15** |
| **최현욱** | 클라이언트 서버 접속 처리 | Recvthread  구현 | 개인 시간 | updateandsendThread()  구현 | | 개인 시간 | Player  class |
| **이미륵** | 개인 시간 | 개인 시간 |
|  | **11/16** | **11/17** | **11/18** | **11/19** | **11/20** | **11/21** | **11/22** |
| **최현욱** | Player  class | SendDatatoServer | | RecvDatatoServer | | 개인 시간 | 중간  점검 |
| **이미륵** | 맵 구현 | | 무기 구현 | | 개인 시간 |
|  | **11/23** | **11/24** | **11/25** | **11/26** | **11/27** | **11/28** | **11/29** |
| **최현욱** | 개인 공부 | 플레이어 이동  동기화 | | 클라이언트  마무리 작업 | | 개인 시간 | 개인 시간 |
| **이미륵** | 개인 공부 | 개인 시간 | 개인 시간 |
|  | **11/30** | **12/01** | **12/02** | **12/03** | **12/04** | **12/05** | **12/06** |
| **최현욱** | 총알 위치  동기화 | | 충돌 확인 및 처리  오류 검사 | | | 개인 시간 | 개인 시간 |
| **이미륵** | 개인 시간 | 개인 시간 |
|  | **12/07** | **12/08** | **12/09** | **12/10** |  |  |  |
| **최현욱** | 마무리  작업 | 보고서  작성 | | 최종  검수 |  |  |  |
| **이미륵** |  |  |  |