네트워크 게임 프로그래밍

term Project

-Agar.io-

**01 분반 (월 23 화 23)**

**게임공학과 2016182007 김동년**

**게임공학과 2016182020 소 준**

**목차**

1. **애플리케이션 기획**

* **게임 이름**
* **게임 개요**
* **게임 방법**
* **게임 진행 방식**
* **조작키**

1. **High-Level 디자인**

* **Flow Chart**
* **Client/Server 핵심요소**
* **Client 실행순서**
* **Server 실행순서**

1. **Low-Level 디자인**

* **Protocol**
* **Client**
* **Server**

1. **애플리케이션 기획**

* **게임 이름**

**Agar.io**



[출처: <http://agar.io/> ]

* **게임 개요**

.io의 **웹 게임**들 중 하나로 다운로드 받지 않고 즐길 수 있는 게임으로 웹 상에서 다른 유저들과 플레이할 수 있는 세포 키우기 게임

* **게임 방법**

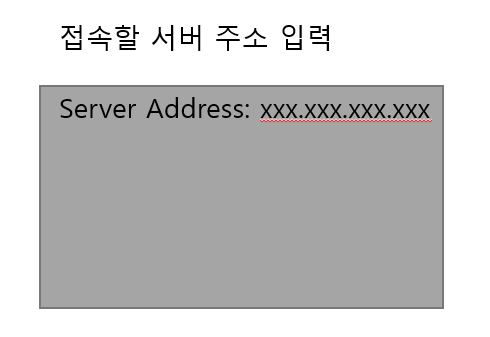
**플레이어**가 게임에 접속하면 **2D**의 맵 상 랜덤한 위치에 배치가 됩니다. 맵 상에는 이 전에 존재하던 플레이어가 있다면 플레이어들이 존재하고 이 외에 **먹이**, **랜덤 아이템**, **장애물**이 있습니다. 플레이어는 맵을 돌아다니면서 먹이를 먹고 몸집을 키울 수 있습니다. 또한 자신보다 작은 크기의 상대를 만날 경우에는 먹을 수 있고 상대의 몸집의 크기에 따라 자신의 몸집이 커지는 비율이 달라집니다. 물론 자신보다 몸집이 큰 상대를 만나면 도망가야 하며 상대에게 먹힐 수도 있습니다. 아이템을 먹으면 효과는 2개 중에 하나의 효과가 랜덤하게 실행됩니다. 첫번째는 속도 증가, 플레이어의 속도를 일정시간 증가시켜 줍니다. 두번째는 스턴, 플레이어의 움직임을 일정시간 멈추게 합니다. 또한 맵 곳곳에는 장애물이 있는데 장애물에 충돌할 경우 플레이어의 크기는 작아지게 됩니다. 또한 거대한 플레이어의 게임의 장악을 막기 위해 벽에 닿을 경우 몸집을 50% 감소시킵니다.

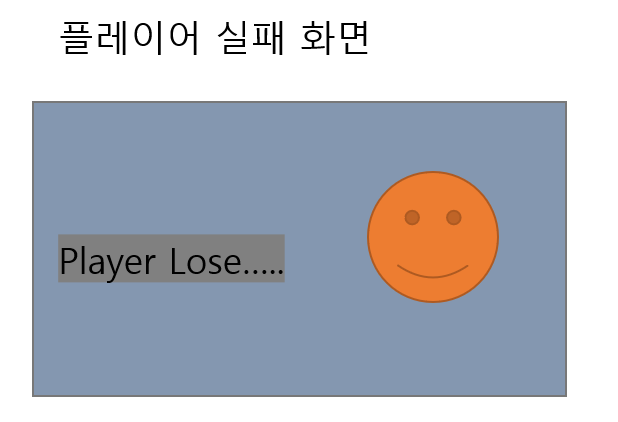
게임의 **승자는 따로 없으며** 자신보다 큰 상대 플레이어에게 **먹힐 경우 게임에서 패배**하게 됩니다.

따로 게임의 세션은 존재하지 않고 플레이어의 게임 **입장과 퇴장이 자유**롭습니다.

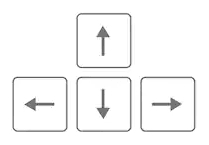
* **게임 진행 방식**

1. 플레이어는 **서버의 주소**를 입력하고 게임 입장 요청
2. 서버로부터 **데이터**를 받고 게임 시작
3. 접속하는 새로운 플레이어들과 **게임 플레이**
4. **먹히거나** **잡아먹거나** 둘 중 하나!





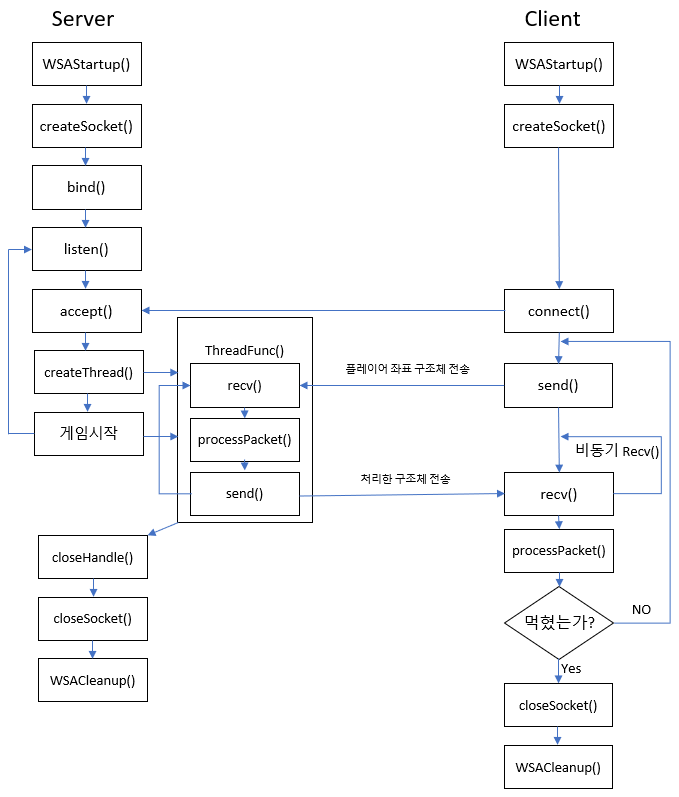
* **조작키**



방향키: 방향에 맞춰 자신의 캐릭터(원)이 이동

1. **High-Level 디자인**

* **Flow Chart**



* **Client/Server 핵심요소**

**Client**

**→클라이언트는 무슨 일만 할 것인가?**

그래픽(플레이어(본인, 상대들), 먹이, 아이템, 맵) 표시

실시간으로 움직이는 플레이어의 좌표 전송

서버에서 보내준 데이터 처리

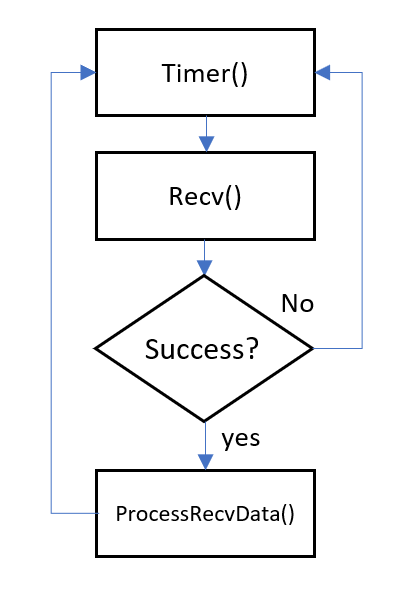
* 다른 플레이어 위치 및 크기
* 먹이, 아이템, 장애물 위치 변경

**→동기 클라이언트? 비동기 클라이언트?**

서버에서 실시간으로 계속 데이터를 보내기 때문에 클라이언트에서는 비동기로 Recv받아야 함

▶Timer를 돌면서 일정시간마다 Recv받도록 함

▶Non-Blocking 소켓 사용



**++추가 기능구현++**

**채팅 기능 추가!**

클라이언트에서 플레이어가 채팅창을 열면 새로운 채팅 전용 스레드가 생성되며 채팅과 별개로 게임이 동작하도록 함

채팅 내용을 입력 받고 전송 후에 해당 스레드를 종료

클라이언트에서는 큐로 채팅들을 관리하고 5개 이상일 경우 제일 먼저 온 채팅을 pop 시키면서 표시되는 채팅 수를 5개 미만으로 유지한다.

**Server**

**→서버에서는 무슨 일만 할 것인가?**

실시간 바뀌는 플레이어 좌표들을 저장

플레이어 좌표들을 브로드 캐스트

연산처리

* 플레이어와 먹이, 이벤트, 장애물, 맵 충돌처리
* 플레이어와 플레이어 충돌처리

먹이, 이벤트, 장애물 생성 및 재배치

실시간으로 변화하는 오브젝트 데이터를 브로드 캐스트

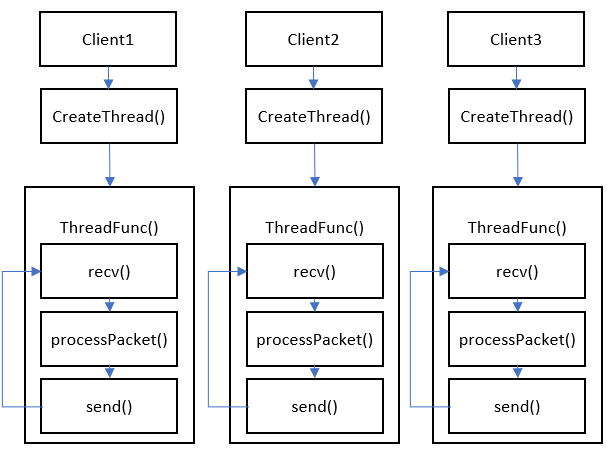
새로운 플레이어 접속관리

먹힌 플레이어 로그아웃 관리

**→멀티 클라이언트 접속 방법?**

각 클라이언트마다 스레드를 생성 후 독립적으로 처리

각 플레이어가 돌면서 충돌처리 사항을 자신의 스레드에서 처리



**→스레드 동기화 방법?**

임계영역을 사용하여 서로 겹치지 않은 공유 변수에 대해서는 다른 동기화 객체로 Lock을 함

+같은 배열내에서 다른 원소를 참조할 때 Lock이 필요한가?

**→어떻게 브로드 캐스트 할 것인가?**

1초에 20번~30번 모든 데이터를 보내는 것 보다 바뀐 데이터

만 보내는 것이 효율적

**++추가 기능구현++**

**채팅 기능 추가!**

클라이언트에서 넘어온 채팅을 모든 클라이언트에게 송신자의 ID와 첨부하여 전송

* **Client 실행순서**

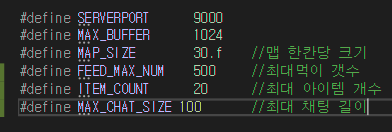
1. **클라이언트 프로그램 실행**
2. **접속할 서버 주소 입력**
3. **서버로부터 초기데이터(기존 플레이어, 먹이, 아이템, 장애물)을 받음**
4. **게임 시작(실행)**
5. **키 입력하여 플레이어의 좌표가 이동할 경우 서버에게 좌표 전송**
6. **timer가 호출되면서 Recv함수 실행**
7. **서버에서 보낸 데이터를 받고 오브젝트 업데이트**
8. **게임이 종료될 때까지 반복 및 조건에 따라 종료**

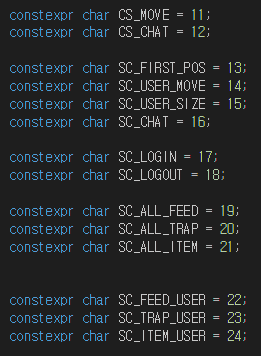
* **Server 실행순서**

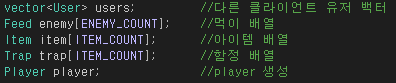
1. **서버를 실행**
2. **소켓을 생성 후 Listen 상태로 대기**
3. **클라이언트가 접속하면 Accept 후 새로운 소켓 생성**
4. **쓰레드를 생성 후 Accept한 소켓의 전용 스레드 생성**
5. **스레드 함수 실행**
6. **다시 메인 스레드는 Listen 상태로 대기**
7. **스레드 함수에서 기존의 플레이어, 먹이, 아이템, 장애물의 자료구조를 전송**
8. **플레이어의 위치를 받고 브로드 캐스트**
9. **연산 및 충돌처리 후 수정된 오브젝트 데이터에 대해 브로드 캐스트**
10. **상대에게 먹힌 플레이어에게 패배 메시지를 보내고 소켓 종료**
11. **위 절차를 반복하면서 서버는 계속 실행**
12. **Low-Level 디자인**

**-Protocol**

Define & Global Variable







**패킷**에서 무슨 패킷인지 알려주기 위한 패킷 type을 protocol로 정의

CS -> Client -> Server

SC -> Server -> Client

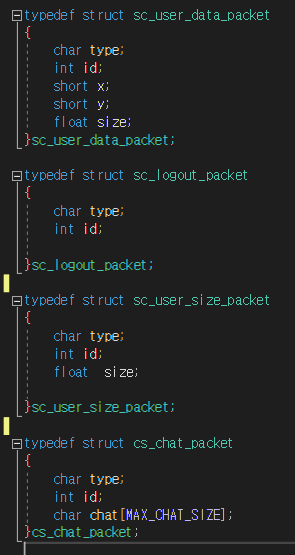
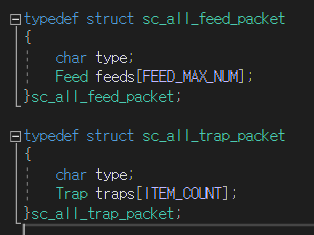
**유저**는 사용자가 정해진 것이 아니라 실시간으로 바뀌므로 벡터로 자료구조 설정

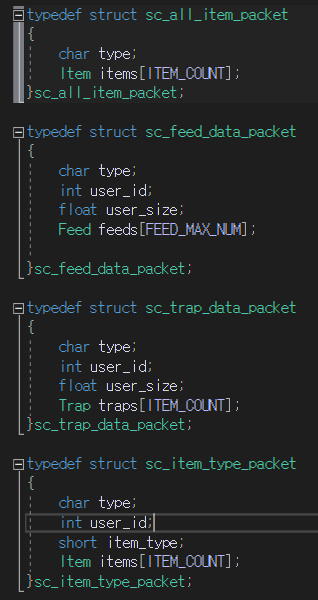
**먹이, 아이템 함정**은 모두 동일한 개수의 양이 맵에 유지됨으로 배열로 설정

**플레이어**(본인)는 관리 용이를 위해 따로 객체를 생성

플레이어 객체는 Client에만 존재

서버에서는 모두 User벡터 자료구조에서 관리

Packet



Client -> Server

**struct sc\_user\_data\_packet**

클라이언트에서 서버로 본인의 좌표와 크기를 전달해주는 패킷

++추가

**struct cs\_chat\_packet**

클라이언트에서 보낸 채팅 데이터와 보낸 ID를 담는 패킷

서버에서도 동일한 패킷을 이용하여 채팅을 클라이언트에게 전달한다.

Server -> Client

**struct sc\_logout\_packet**

클라이언트가 종료하거나 해당 플레이어가 질 경우 logout된 것을 알려주고 자신이 logout 패킷을 받을 경우 lose로 종료처리가 되고 다른 클라이언트는 해당 클라이언트가 종료됨을 처리함

**struct sc\_user\_size\_packet**

클라이언트들이 서로 충돌하였을 때 충돌처리를 하면서 바뀐 크기를 모든 클라이언트들에게 전송하기 위한 패킷

**struct sc\_all\_feed\_packet**

클라이언트가 초기 입장 시 현재 먹이 배열을 전송하기 위한 패킷

**struct sc\_all\_item\_packet**

클라이언트가 초기 입장 시 현재 아이템 배열을 전송하기 위한 패킷

**struct sc\_all\_trap\_packet**

클라이언트가 초기 입장 시 현재 함정 배열을 전송하기 위한 패킷

**struct sc\_feed\_data\_packet**

클라이언트와 먹이 원소가 충돌하였을 때 충돌한 클라이언트의 변경된 사이즈와 위치가 업데이트된 먹이 배열을 전송하기 위한 패킷

**struct sc\_trap\_data\_packet**

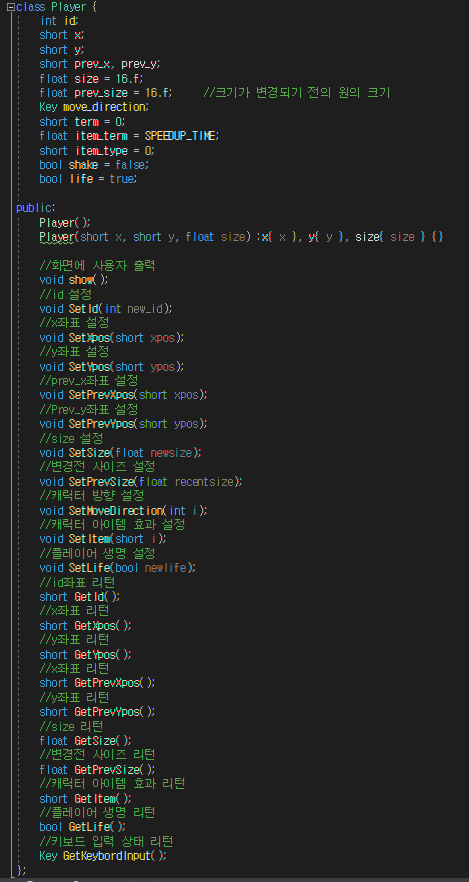
클라이언트와 장애물 원소가 충돌하였을 때 충돌한 클라이언트의 변경된 사이즈와 위치가 업데이트된 장애물 배열을 전송하기 위한 패킷

**struct sc\_item\_data\_packet**

클라이언트와 아이템 원소가 충돌하였을 때 충돌한 아이템의 속성과 위치가 업데이트된 아이템 배열을 전송하기 위한 패킷 클라이언트는 해당 아이템 속성을 받아 그에 맞는 처리함

**-Client**

**Player Class : 플레이어 클래스**



플레이어 본인의 객체 생성을 위한 클래스로 클라이언트에서만 존재함

x, y의 좌표 값과 현재의 크기를 저장하는 size 변수가 할당되고 일정크기 이상 작아지지 못하게 prev\_size라는 변수를 추가함.

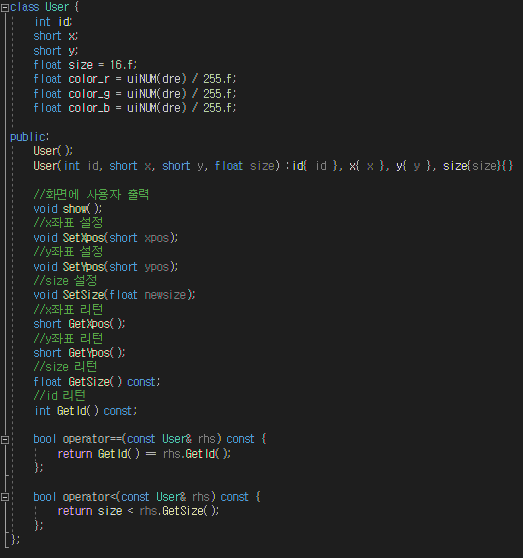
플레이어의 좌표는 클라이언트에서 사용자의 키 값을 받아 실시간으로 수정하고 size는 서버에서 받은 값으로 업데이트함

life 변수를 통해 죽었는지를 판별하며, 키보드 입력 상태를 받아 끊김없이 자연스러운 움직임을 구현함

item을 얻었을 때 서버로부터 받은 데이터에 따라 아이템 효과 (가속or스턴)를 구현함

Show()함수에서 캐릭터가 일정 거리만큼 움직일 때마다 서버로 데이터를 보내주며, 맵과 충돌처리 및 스턴 효과 등을 구현함

**User Class : 유저 클래스**

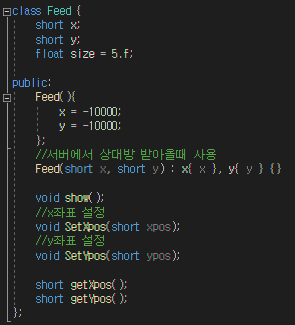


클라이언트에서 본인 이외의 유저들을 위한 클래스이며 서버로부터 받은 x, y 좌표와 크기 size을 초기화 및 수정하고 맵 위에 보여주는 함수 등을 정의

유저별로 색깔을 달리하여 구분할 수 있도록 함.

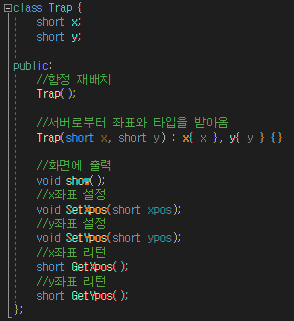
Operator연산자를 추가하여 정렬기능을 사용해 Rank를 알 수 있도록 함.

**Feed Class : 먹이 클래스**



서버에서 받아온 먹이들의 좌표들을 깜빡이는 색상으로 맵 위에 그려준다. 또한 서버에서 받아온 배열을 덮어쓰며 배열을 갱신함.

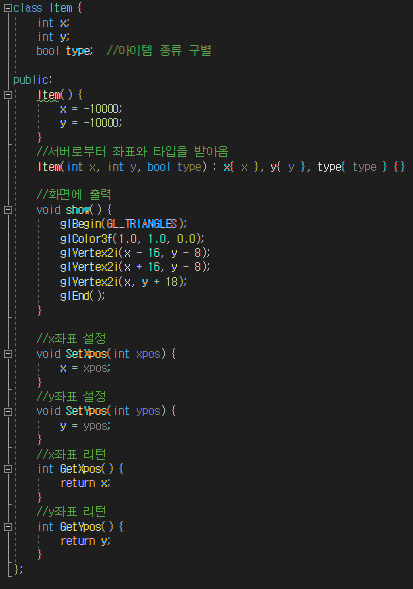
**Trap Class : 트랩 클래스**



서버에서 전송받은 함정의 좌표를 화면에 그려주고 업데이트를 하기 위한 함수들을 정의.

화면에 검정색 사각형으로 그려지며, 충돌 시 충돌한 플레이어의 크기가 일정 비율만큼 줄어듦.

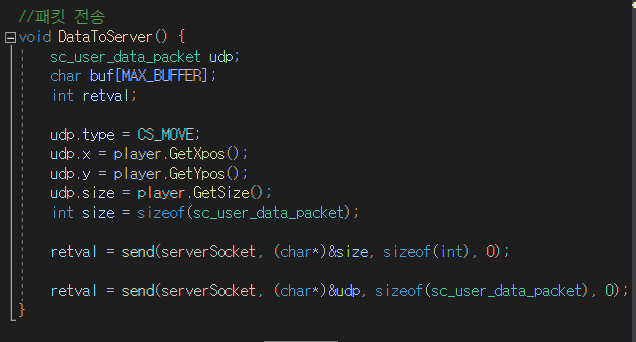
**Item Class : 아이템 클래스**



서버에서 아이템의 좌표를 받아오고 타입에 따라 아이템을 구별함. 화면에 노란색 삼각형으로 그림.

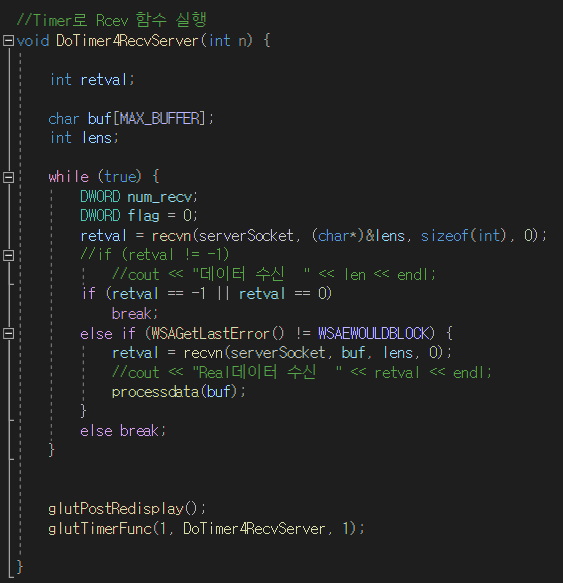
<먹이, 아이템, 함정들은 생성과 소멸을 반복하는 것이 아니라 특정 플레이어와 충돌할 경우 새로운 위치로 변경됨>

**함수**



**void SendDatatoServer()**

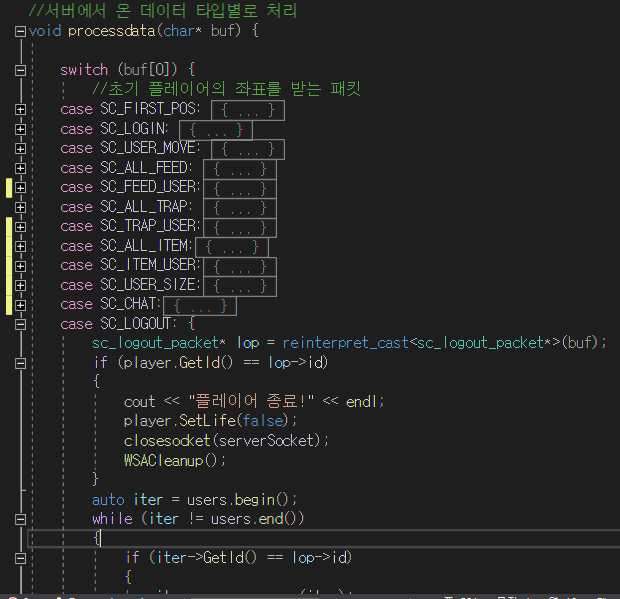
플레이어의 현재의 좌표와 크기를 서버로 전송



**void DoTimerForRecvServer(int n);**

타이머를 돌면서 일정시간마다 Recv 함수 호출

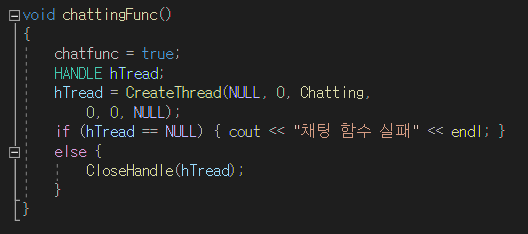
받을 데이터가 없다면 while 문을 빠져나옵니다.  
Recv된 데이터가 있다면 processdata를 호출시켜 해당 데이터를 처리함



**void ProcessData(char\* buf)**

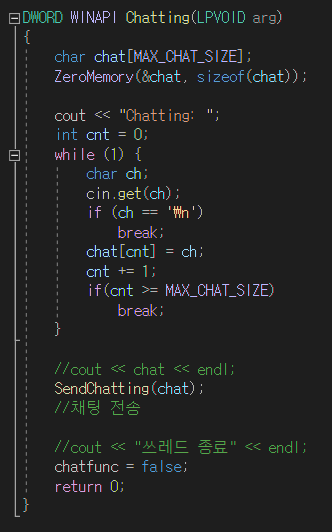
서버에서 넘어온 받은 데이터를 타입 별로 확인하여 해당 타입에 따라 패킷을 받고 데이터를 처리함

++추가구현 채팅기능



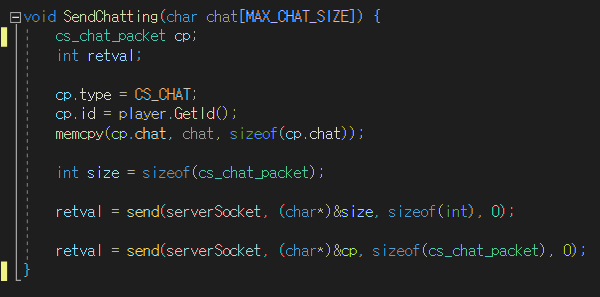
**void chattingFunc()**

클라이언트에서 채팅전송 키를 눌렀을 때 실행되며, 채팅전송을 위한 새로운 스레드를 생성함. 생성된 스레드는 Chatting 함수를 실행함.



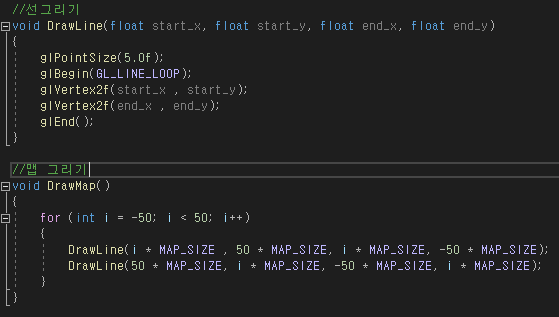
**DWORD WINAPI Chatting(LPVOID arg)**

클라이언트에서 채팅 내용을 입력받고 해당 채팅 내용을 SendChatting을 호출시켜 서버로 전송.



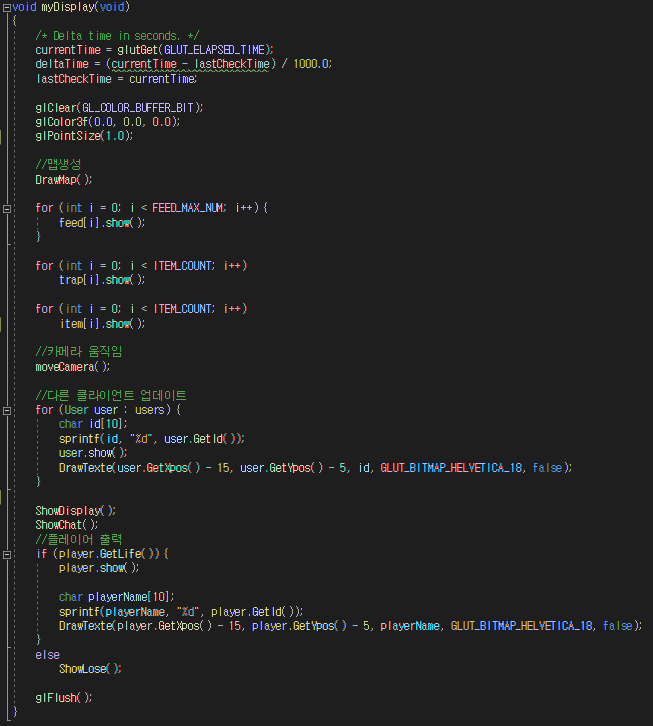
**void SendChatting(char chat[MAX\_CHAT\_SIZE])**

인자로 받은 채팅 문자열과 채팅을 보내는 클라이언트 id를 함께 패킷에 담아 서버로 전송.

**void DrawMap()** 

정의된 맵의 크기에 맞게 격자형태의 맵을 그려줌

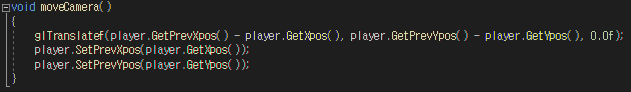
**void myDisplay(void)**



컴퓨터 성능에 따른 렌더링 성능 차이를 줄이기 위해 deltaTime을 구현

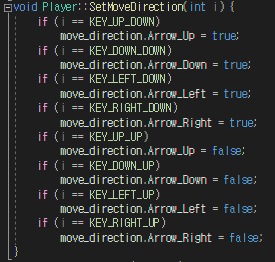
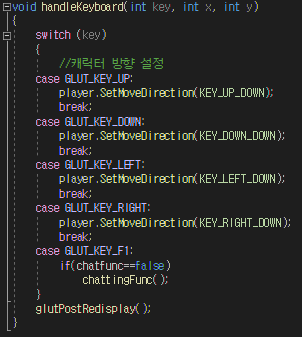
화면에 맵, 먹이, 아이템, 함정, 플레이어(본인), 다른 유저들을 그려주며, 좌측 상단에 랭크를 표시하고 좌측 하단에 채팅창을 보여줌.

**void moveCamera(int key)**



키를 입력 받을 때마다 카메라를 이동. 플레이어가 화면에 중심에 오도록 카메라 위치를 조정.

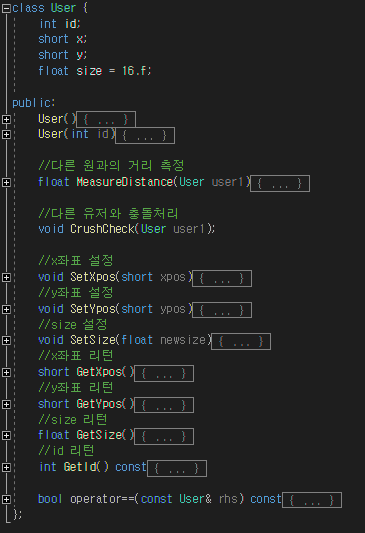
**void handleKeyboard(int key, int x, int y)**



키가 눌려 있는지를 판별하여 플레이어를 자연스럽게 이동시키고 플레이 화면에서 f1을 누를 경우 콘솔창에서 채팅을 입력 받음.

**-Server**

**User Class : 유저 클래스**



서버에 접속한 유저들의 객체를 생성하기 위한 클래스 유저의 ID를 받아 유저들을 구분하고 클라이언트에서 넘어온 x, y 좌표를 받아 수정한다. 서버 내에서 유저와 유저, 먹이, 장애물에 충돌할 경우 size 값을 수정.

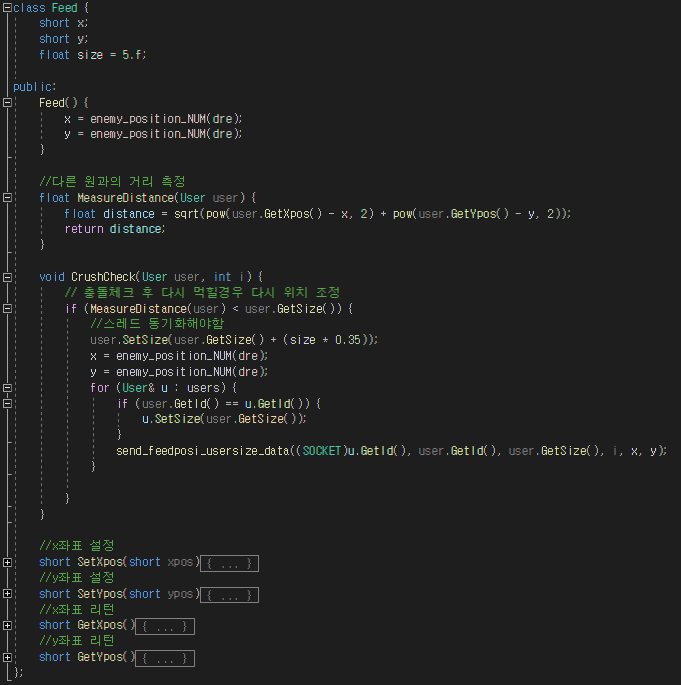
**float MeasureDistance(User user1)**

다른 유저 클라이언트 원의 중심과 현재 클라이언트 원의 중심 사이의 거리를 계산 후 리턴.

**void ChrushCheck(User user1)**

MeasuerDistance()함수를 사용하여 다른 클라이언트와의 거리를 계산하고 원의 충돌이 발생하는 순간 사이즈가 큰 플레이어가 작은 플레이어를 흡수하여 크기가 커지고 작은 원에 해당하는 클라이언트의 로그아웃과 사이즈가 변화한 플레이어의 정보를 알림.

**Feed Class : 먹이 클래스**



먹이들의 크기는 모두 같으며 초기 x, y 좌표는 맵 내에 랜덤하게 배치됨. 작은 원의 모양을 하고 색상은 반짝거리며 바뀐다. 플레이어가 먹이를 먹을 경우 해당 먹이는 사라지고 다른 곳에 랜덤하게 다시 배치됨.

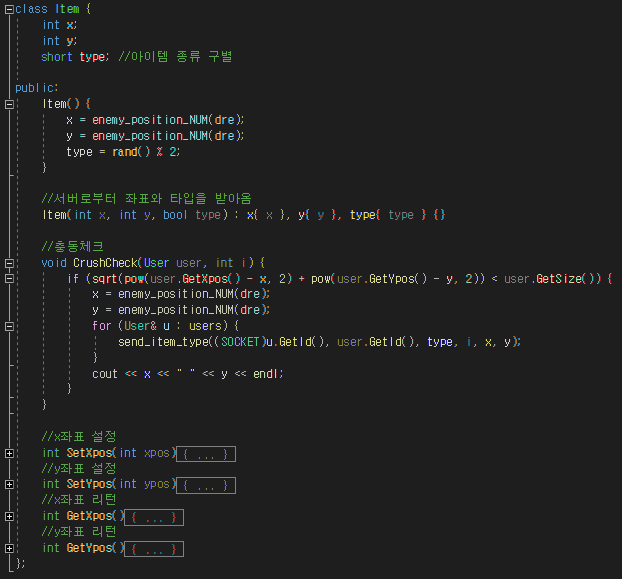
**void MeasuerDistance()**

먹이의 중심점과 유저의 중심점 사이의 거리를 리턴

**void CrushCheck()**

먹이의 중심점과 유저의 중심점 사이의 거리가 유저의 반지름 길이보다 짧아진다면 먹이를 먹은 것으로 판정되어 유저의 원 크기를 키워주고 먹이의 좌표를 랜덤한 위치에 재배치.

**Item Class : 아이템 클래스**

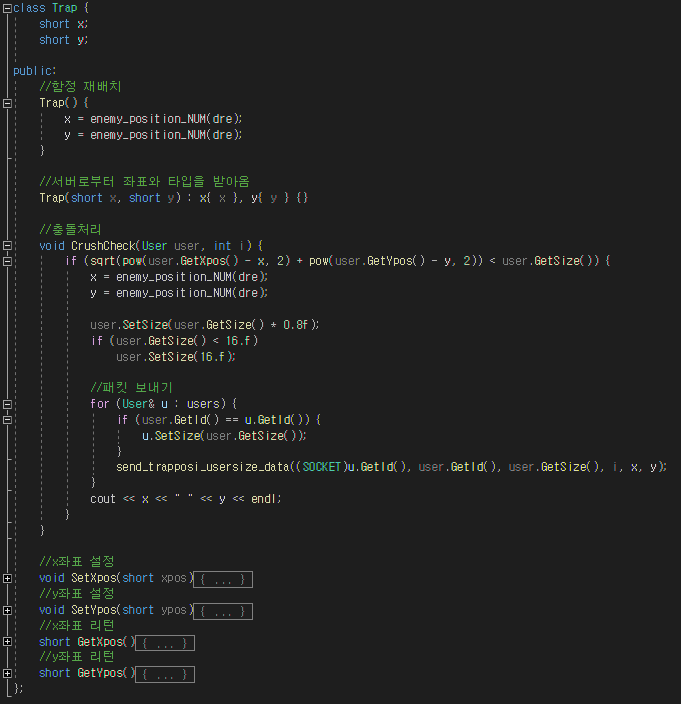


아이템은 2가지의 아이템이 존재. 하나는 일정시간 동안 속도를 높여주는 속도 향상 아이템, 다른 하나는 일정 시간동안 움직이지 못하게 만드는 스턴 아이템이 있음. 아이템의 타입은 랜덤하게 설정되고 세모의 형태로 배치됨. 플레이어들은 아이템 획득 시 랜덤한 효과를 얻게 됨. 아이템 또한 맵 내에 랜덤한 위치에 배치되고 플레이어가 획득한 경우 사라지고 다른 곳에 재배치됨.

**void CrushCheck()**

아이템의 중심점과 유저의 중심점 사이의 거리가 유저의 반지름 길이보다 짧으면 아이템를 먹은 것으로 판정되어 유저에게 아이템에 맞는 효과를 부여하고 아이템은 랜덤한 위치에 재배치.

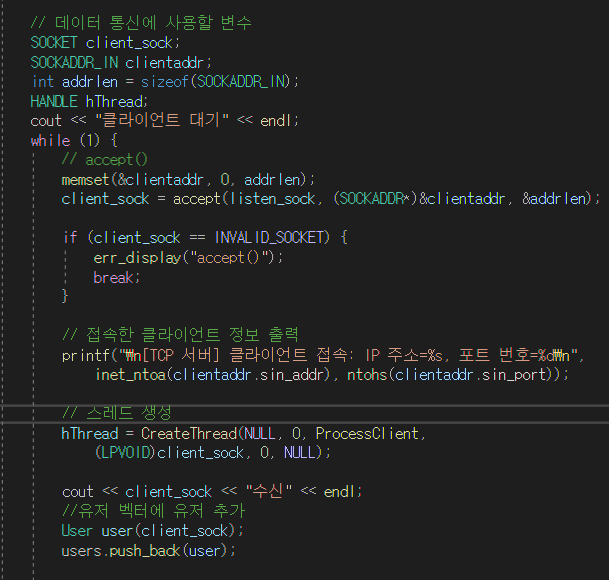
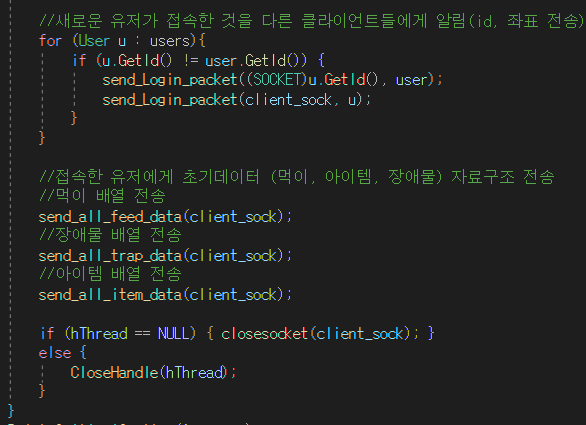
**Trap Class : 함정 클래스**



함정들은 검정색의 네모 모양의 형태로 맴 내에 랜덤한 위치에 배치되어 있다. 플레이어와 함정이 충돌할 경우 플레이어의 크기가 20% 줄어들게 되고 해당 함정은 사라지고 다른 랜덤한 위치에 재배치.

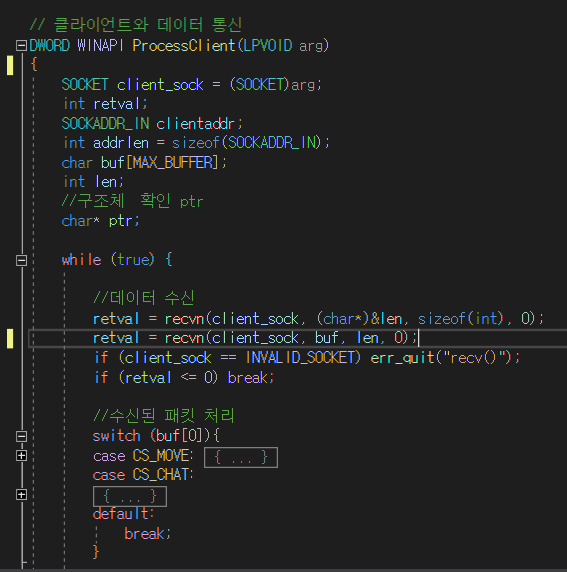
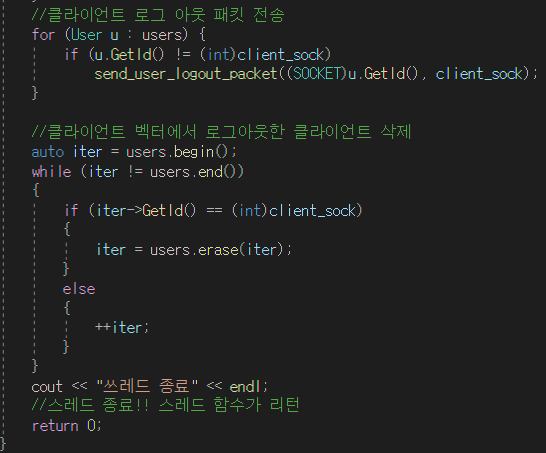
**void CrushCheck()**

함정의 중심점과 유저의 중심점 사이의 거리가 유저의 반지름 길이보다 짧으면 함정에 걸린 것으로 판정되어 유저의 원 크기를 3분에 1로 줄이고 함정은 랜덤한 위치에 재배치.

**int main()**

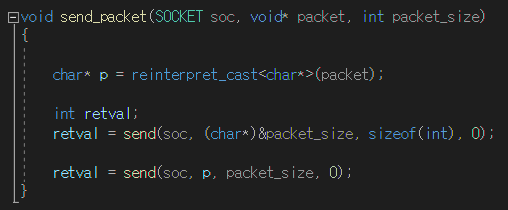
클라이언트의 접속을 기다리다 클라이언트가 접속하면 해당 클라이언트의 전용 스레드를 생성. 이후 유저 벡터에 해당 클라이언트를 추가하고 다른 클라이언트에게 해당 유저의 접속을 알림. 초기 좌표 값을 전송하고 먹이, 아이템, 함정의 초기 데이터를 전송. 전송이 완료되면 다시 새로운 클라이언트를 기다림.

**DWORD ProcessClient()**

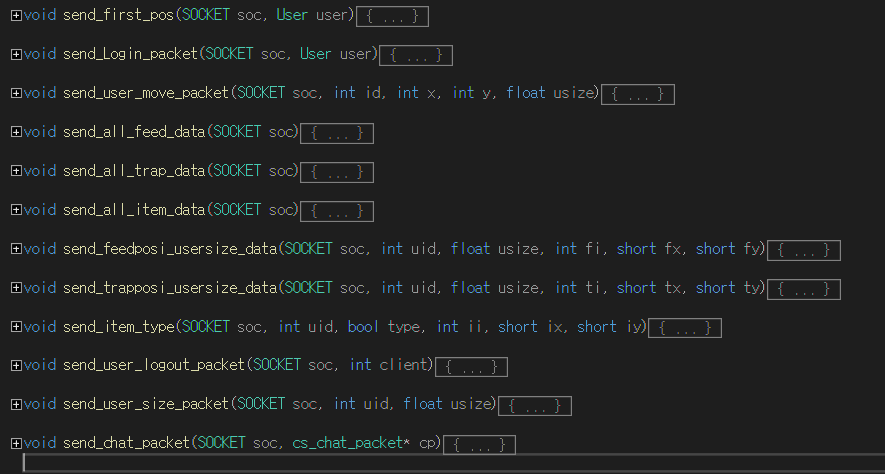
생성된 유저 스레드가 실행할 함수

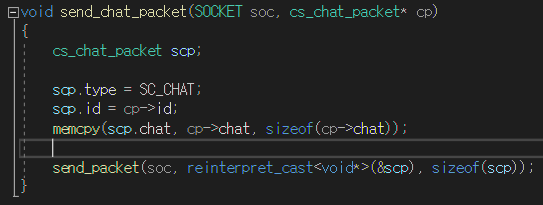
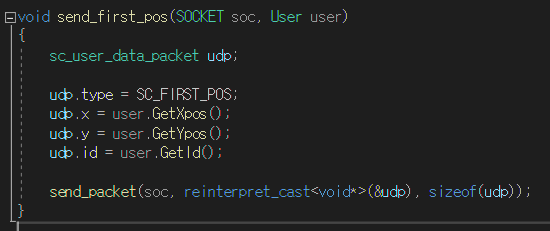
연결된 클라이언트에서 좌표를 받아 다른 클라이언트들에게 전송 하고 해당 클라이언트의 플레이어 객체와 다른 플레이어, 먹이, 아이템, 함정, 맵과의 충돌 처리 연산을 하고 바뀐 데이터가 있다면 바뀐 데이터를 모든 클라이언트에게 전송.



**void send\_packet(SOCKET soc, void\* packet, int packet\_size)**

각각의 전송 함수에서 보내는 패킷과 그 사이즈를 받아 해당 소켓에 패킷을 전송하는 역할을 한다. 전송을 요하는 각 함수에서 send 함수가 다수 호출되기 때문에 코드의 최적화를 하기 위해 만들어진 함수





**서버에서 클라이언트로 데이터를 전송하는 함수들**

보낼 상대의 소켓 값과 데이터를 인자로 받아 해당 구조체 패킷에 전송할 내용을 전달하고 해당 패킷의 타입을 정의한 뒤 전송

**임계영역**

각각의 프로세스를 나누어 만든 이유 -> 멀티 스레드 동기화를 위해

CRITICAL\_SECTION ac\_move;

CRITICAL\_SECTION ac\_user;

CRITICAL\_SECTION ac\_feed;

CRITICAL\_SECTION ac\_item;

CRITICAL\_SECTION ac\_trap;

**EnterCriticalSection**(&ac\_move)

클라이언트에서 받은 좌표를 해당 클라이언트를 찾아서 수정함

**LeaveCriticalSection**(&ac\_move)

**EnterCriticalSection**(&ac\_user)

클라이언트와 클라이언트의 충돌처리를 하며 데이터를 수정하고 수정된 데이터를 전송

**LeaveCriticalSection**(&ac\_user)

**EnterCriticalSection**(&ac\_feed)

먹이 배열과 클라이언트의 충돌처리로 인한 데이터 수정

전역변수 Feed feed[100]의 데이터 수정

**LeaveCriticalSection**(&ac\_feed)

**EnterCriticalSection**(&ac\_item)

아이템 배열과 클라이언트의 충돌처리로 인한 데이터 수정

전역변수 Item item[100]의 데이터 수정

**LeaveCriticalSection**(&ac\_item)

**EnterCriticalSection**(&ac\_trap)

함정 배열과 클라이언트의 충돌처리로 인한 데이터 수정

전역변수 Trap trap[100]의 데이터 수정

**LeaveCriticalSection**(&ac\_trap)

1. **개발환경**

Visual Studio 2019

**-Platform**

Windows PC 게임

**-Client**

Open\_Gl ->그래픽

STL ->자료구조

C++ ->기본 연산 및 출력

**-Server**

Windows Socket Programming ->윈도우 소켓

STL ->자료구조

C++ ->기본 연산 및 출력

**-Protocol**

TCP/IP

IPv4

1. **팀원 별 역할분담**

|  |  |  |  |
| --- | --- | --- | --- |
| **구분** | **내용** | **김동년** | **소준** |
| **기획** | **게임 구상** | **O** | **O** |
|  | **High\_Lv 구상** | **O** |  |
|  | **Low\_Lv 구상** | **O** | **O** |
|  | **기획서 작성** | **O** | **O** |
| **공용 프로토콜** | **Global variable** | **O** | **O** |
|  | **struct cs\_my\_packet** | **O** |  |
|  | **struct sc\_feed\_packet** | **O** |  |
|  | **struct sc\_item\_packet** | **O** |  |
|  | **struct sc\_trap\_packet** | **O** |  |
|  | **struct sc\_user\_packet** |  | **O** |
|  | **struct sc\_login\_packet** |  | **O** |
|  | **struct sc\_mysize\_packet** |  | **O** |
|  | **struct sc\_lose\_packet** |  | **O** |
| **클라이언트** | **Player class** | **O** |  |
|  | **User class** | **O** |  |
|  | **Feed class** | **O** |  |
|  | **Item class** | **O** |  |
|  | **Trap class** | **O** |  |
|  | **void SendDatatoServer()** | **O** |  |
|  | **void TimerForRecvData()** | **O** |  |
|  | **void ProcessData()** |  | **O** |
|  | **void DrawMap()** | **O** | **O** |
|  | **void myDisplay()** |  | **O** |
|  | **void moveCamera()** |  | **O** |
|  | **void handleKeyboard()** | **O** |  |
| **서버** | **User class** |  | **O** |
| **구분** | **내용** | **김동년** | **소준** |
|  | **Feed class** |  | **O** |
|  | **Item class** |  | **O** |
|  | **Trap class** |  | **O** |
|  | **HANDLE CreateThread()** | **O** |  |
|  | **DWORD UserThread()** | **O** |  |
|  | **void BroadCastToClient()** | **O** | **O** |
|  | **Void UserLogin()** |  | **O** |
|  | **void UserLogoutOrLose()** |  | **O** |
|  | **void ProcessUsers()** | **O** |  |
|  | **void ProcessFeed()** | **O** |  |
|  | **void ProcessItem()** |  | **O** |
|  | **void ProcessTrap()** |  | **O** |
|  | **void CrushCeck()** |  | **O** |
| **마무리** | **오류 및 버그 수정** | **O** | **O** |
|  | **Project Progress Report 작성** | **O** | **O** |

1. **개발일정**

**-Big flow**

**1. 클라이언트 프레임, 서버 프레임 제작**

**2. 클라이언트 플레이어 및 맵 생성 후 그리기 구현**

**3. 서버 멀티 클라이언트 접속 구현**

**4. 클라이언트 실시간 좌표 송신 및 서버 좌표 수신과 처리**

**5. 먹이 클래스 구현 후 자료구조 생성**

**6. 새로 접속한 클라이언트에 먹이 자료구조 및 기존 플레이어 정보 송신**

**7. 서버에서 플레이어와 먹이 충돌 처리 및 결과 브로드 캐스트**

**8. 아이템 클래스 및 자료구조 생성 및 처리**

**9. 함정 클래스 및 자료구조 생성 및 처리**

**-****Schedule Calendar**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **월** | **화** | **수** | **목** | **금** | **토** | **일** |
|  | 10/26 | 10/27 | 10/28 | 10/29 | 10/30 | 10/31 | 11/01 |
| **김동년** | 중간고사 | | | 기획 및 기획서 작성 | 기획서 작성 | 아르바이트 | 아르바이트 |
| **소준** | 개인 정비 | 개인 정비 |
|  | 11/02 | 11/03 | 11/04 | 11/05 | 11/06 | 11/07 | 11/08 |
| **김동년** | 기획서 작성 | 서버 프레임 제작 | 멀티 스레드 구현  클라이언트 플레이어 클래스 생성 | | 클라이언트 플레이어 좌표 수신 및 처리 | 아르바이트 | 아르바이트 |
| **소준** | 클라이언트 프레임 제작 | | 클라이언트 게임 구현 | 플레이어 좌표 송신 | 개인 정비 | 개인 정비 |
|  | 11/09 | 11/10 | 11/11 | 11/12 | 11/13 | 11/14 | 11/15 |
| **김동년** | 다중 클라이언트 접속 처리 | 클라이언트 로그인 브로드 캐스트 | 플레이어 좌표 브로드 캐스트 | 먹이 클래스 생성 및 클라이언트 송신 | 서버, 클라이언트 정상 동작 확인 및 오류 검사 | 아르바이트 | 아르바이트 |
| **소준** | 로그인 구조체 패킷 생성 | 클라이언트 상 로그인 패킷 수신 및 처리 | 클라이언트 상 좌표 패킷 수신 및 처리 | 클라이언트 내 먹이 객체 띄우기 | 개인 정비 | 개인 정비 |
|  | 11/16 | 11/17 | 11/18 | 11/19 | 11/20 | 11/21 | 11/22 |
| **김동년** | 새로운 클라이언트 접속 시 먹이 배열 전송 | 먹이 구조체 패킷 생성 및 전송 | 먹이를 먹은 클라이언트에 사이즈 업 패킷 전송 | 서버에서 로그아웃 및 패배한 클라이언트 삭제 | 로그아웃 한 플레이어 정보를 브로드 캐스트 및 수신 처리 | 아르바이트 | 아르바이트 |
| **소준** | 플레이어와 먹이 충돌 처리 | 먹이 좌표 패킷 수신 후 처리 | 클라이언트에서 사이즈 패킷 처리 | 플레이어 로그아웃 패킷 구현 | 클라이언트 Lose 화면 구현 | 개인 정비 | 개인 정비 |
|  | 11/23 | 11/24 | 11/25 | 11/26 | 11/27 | 11/28 | 11/29 |
| **김동년** | 클라이언트 상 아이템 클래스 구현 및 서버에서 전송 처리 | | 아이템 좌표 패킷 생성 후 전송 | 클라이언트 상에서 맵과 플레이어 충돌처리 구현 | 서버, 클라이언트 정상 동작 확인 및 오류 검사 | 아르바이트 | 아르바이트 |
| **소준** | 서버 상 아이템 클래스 생성 | 플레이어와 아이템 충돌처리 | 아이템 패킷 수신 처리 | 개인 정비 | 개인 정비 |
|  | 11/30 | 12/01 | 12/02 | 12/03 | 12/04 | 12/05 | 12/06 |
| **김동년** | 클라이언트 상 트랩 클래스 구현 | 트랩 좌표 패킷 생성 후 송신 | 플레이어의 사이즈 패킷 송신 | 추가 기능 구현 | 추가 기능 구현 및 오류 검사 | 아르바이트 | 아르바이트 |
| **소준** | 서버에서 트랩 클래스 구현 | 트랩 과 플레이어 충돌처리 | 트랩 충돌처리 결과 수신 처리 | 개인 정비 | 개인 정비 |
|  | 12/07 | 12/08 |  |  |  |  |  |
| **김동년** | Project Progress Report 작성 | 프로젝트 제출 및 시연 |  |  |  |  |  |
| **소준** |  |  |  |  |  |