게임 서버 프로그래밍

Term Project

게임 설명서

2017180012

게임공학과

박주용

1. 프로토콜

* 기존 텀 프로젝트 용 프로토콜에서 cs\_packet\_login 패킷에다가 is\_dummy 추가. (사람인지 더미 클라이언트인지를 구별해서 수많은 더미 클라이언트가 DB에 쓸데없이 저장되는 것을 막음.)
* sc\_packet\_status\_change 패킷에서 x, y를 추가하여 플레이어가 텔레포트 할 때, 죽을 때 플레이어의 위치 좌표를 바꿔주는 역할을 함.
* NPC\_PEACE(0)와 NPC\_AGRO(0)를 define함.
* sc\_packet\_put\_object 패킷에 npc\_type을 추가하여 서버에서 7(Peace) : 3(Agro) 비율로 몬스터의 타입을 결정하고 클라이언트에 패킷을 넘겨줘서 타입에 따라 텍스처를 결정한다.

#pragma once

#define NPC\_PEACE 0

#define NPC\_AGRO 1

const short SERVER\_PORT = 4000;

const int WORLD\_HEIGHT = 2000;

const int WORLD\_WIDTH = 2000;

const int MAX\_NAME\_SIZE = 20;

const int MAX\_CHAT\_SIZE = 100;

const int MAX\_USER = 10000;

const int MAX\_NPC = 20000;

constexpr int NPC\_ID\_START = MAX\_USER;

constexpr int NPC\_ID\_END = MAX\_USER + MAX\_NPC - 1;

const char CS\_PACKET\_LOGIN = 1;

const char CS\_PACKET\_MOVE = 2;

const char CS\_PACKET\_ATTACK = 3;

const char CS\_PACKET\_CHAT = 4;

const char CS\_PACKET\_TELEPORT = 5;

const char SC\_PACKET\_LOGIN\_OK = 1;

const char SC\_PACKET\_MOVE = 2;

const char SC\_PACKET\_PUT\_OBJECT = 3;

const char SC\_PACKET\_REMOVE\_OBJECT = 4;

const char SC\_PACKET\_CHAT = 5;

const char SC\_PACKET\_LOGIN\_FAIL = 6;

const char SC\_PACKET\_STATUS\_CHANGE = 7;

#pragma pack (push, 1)

struct cs\_packet\_login {

unsigned char size;

char type;

char name[MAX\_NAME\_SIZE];

//+추가

int is\_dummy;

};

struct cs\_packet\_move {

unsigned char size;

char type;

char direction; // 0 : up, 1: down, 2:left, 3:right

int move\_time;

};

struct cs\_packet\_attack {

unsigned char size;

char type;

};

struct cs\_packet\_chat {

unsigned char size;

char type;

char message[MAX\_CHAT\_SIZE];

};

struct cs\_packet\_teleport {

// 서버에서 장애물이 없는 랜덤 좌표로 텔레포트 시킨다.

// 더미 클라이언트에서 동접 테스트용으로 사용.

unsigned char size;

char type;

};

struct sc\_packet\_login\_ok {

unsigned char size;

char type;

int id;

short x, y;

short level;

short hp, maxhp;

int exp;

};

struct sc\_packet\_move {

unsigned char size;

char type;

int id;

short x, y;

int move\_time;

};

struct sc\_packet\_put\_object {

unsigned char size;

char type;

int id;

short x, y;

char object\_type;

char name[MAX\_NAME\_SIZE];

//+추가

char npc\_type;

};

struct sc\_packet\_remove\_object {

unsigned char size;

char type;

int id;

};

struct sc\_packet\_chat {

unsigned char size;

char type;

int id;

char message[MAX\_CHAT\_SIZE];

int chat\_type;

};

struct sc\_packet\_login\_fail {

unsigned char size;

char type;

int reason; // 0: 중복 ID, 1:사용자 Full

};

struct sc\_packet\_status\_change {

unsigned char size;

char type;

short level;

short hp, maxhp;

int exp;

//+추가

short x, y;

};

#pragma pack(pop)

1. 자료구조
2. DB 구조체를 만들어서 플레이어의 id, level, x, y, username을 관리함. Sql 데이터들을 여기에 저장한다.

struct DB

{

int id;

short level;

int x, y;

char userName[20];

};

1. DB 구조체를 다시 vector로 관리한다. 구조체의 내용들을 emplace\_back하여 데이터들을 저장 해놓고 클라이언트로부터 오는 패킷이나 sql 함수 인자와 비교할 때 등에 쓰인다.

vector<DB> g\_db;

1. 로그인 한 플레이어들의 유저 네임을 저장하기 위한 vector. 로그인에 성공한 아이디들을 이곳에 저장 해놓고 뒤에 로그인하는 아이디들과 비교하기 위해 쓰인다. 아이디가 같으면 아이디 중복으로 로그인 실패, 아이디가 같으면 로그인이 성공한다.

vector<char\*>login\_players;

1. 어그로 몬스터의 시야 반경 안에 들었을 때 해당 플레이어의 id를 저장한다. 해당 플레이어에 대한 타이머 이벤트(EVENT\_NPC\_AGRO\_CHASE)를 작동시켜 NPC가 플레이어를 쫓아가게 한다.

unordered\_set<int> agro\_list;

1. 타이머 관련 구조체와 concurrent\_priority\_queue 등. 초당 move나 attack 등 어떠한 이벤트를 진행하기 위해 사용된다. 타이머 큐에 이벤트를 push 하는 것은 push\_timer\_event라는 함수로 따로 만들었다.

enum EVENT\_TYPE { EVENT\_NPC\_MOVE, EVENT\_NPC\_AGRO\_CHASE, EVENT\_NPC\_AGRO\_ATTACK};

struct timer\_event {

int obj\_id;

int target\_id;

chrono::system\_clock::time\_point start\_time;

EVENT\_TYPE ev;

constexpr bool operator < (const timer\_event& \_Left) const

{

return (start\_time > \_Left.start\_time);

}

};

concurrency::concurrent\_priority\_queue <timer\_event> timer\_queue;

mutex t\_lock;

cf)

void push\_timer\_event(int obj\_id, int target\_id, int sec, EVENT\_TYPE ev\_type)

{

timer\_event ev = { obj\_id, target\_id, chrono::system\_clock::now() + chrono::seconds(sec), ev\_type };

t\_lock.lock();

timer\_queue.push(ev);

t\_lock.unlock();

}

1. 게임 흐름
2. 로그인

* 클라이언트가 서버에 접속할 때 서버는 로그인 패킷 중 is\_dummy를 통해 해당 클라이언트가 사람인지 더미 클라이언트인지 파악한다.
* 더미 클라이언트이면 곧바로 login success, 사람이면 클라이언트에서 보낸 name 패킷과 DB에 저장된 username을 비교한다.
* 비교 결과 같으면 다시 로그인 중인 플레이어와 username을 비교한다. 같으면 아이디 중복이라는 reason으로 로그인 실패, 다르면 로그인 성공이다.
* name 패킷과 DB의 username이 다르다면 새로운 플레이어 정보를 DB에 insert하고 로그인을 성공 처리한다.

1. 인 게임

* 게임에 들어가면 기존 유저의 경우에는 마지막에 있었던 x, y 좌표와 저장된 level 값으로 시작한다.
* 신규 유저의 경우 집에서(10, 10 위치) 1레벨로 시작하고 포탈 안에서 t키를 누르면 랜덤한 좌표로 순간이동 한다.
* 파란(PEACE) 몬스터와 만났을 때 스페이스 바를 누르면 몬스터를 잡을 수 있고 레벨이 1레벨 증가한다. 파란 몬스터는 플레이어를 공격하지 않는다.
* 주황(AGRO) 몬스터 시야 안에 들면 몬스터가 플레이어를 쫓아오고 플레이어의 근처까지 오게 되면 플레이어를 공격한다. 플레이어의 레벨이 1인 상태에서 주황 몬스터에게 공격당하면 플레이어는 죽게 되고 다시 초기위치(집)로 이동한다. 여기서 더미 클라이언트의 경우에도 집으로 이동하면 집에 더미 클라이언트들이 바글바글 쌓이므로 더미 클라이언트는 다른 랜덤한 위치로 순간 이동하게 하였다.

1. 알고리즘
2. 몬스터->플레이어 Chase & Attack 알고리즘

* 두 점 간의 거리공식을 이용하여 몬스터와 플레이어의 x거리, y거리를 각각 구해준다. x거리가 y거리보다 클 때, x거리가 y거리보다 작을 때, x거리와 y거리가 같을 때를 각각 비교하여 몬스터가 플레이어 왼쪽에 있으면 오른쪽으로 한 칸, 오른쪽에 있으면 왼쪽으로 한 칸, 위에 있으면 밑으로 한 칸, 밑에 있으면 위로 한 칸 이런 식으로 구현하여 타이머 이벤트(EVENT\_NPC\_AGRO\_CHASE)가 발생할 때마다 몬스터가 플레이어에게 다가가도록 한다. 만약 몬스터와 플레이어 간의 x거리와 y거리가 모두 1 이내일 시, EVENT\_NPC\_AGRO\_ATTACK 타이머 이벤트를 push해준다. 그렇게 되면 플레이어가 주황 몬스터에게 잡혔을 때 매초마다 공격을 당하게 된다.

//어그로 몬스터를 만났을 시 이 함수를 불러준다.

void NpcChasingUpdate(int npc\_id, int player\_id)

{

int p\_x = clients[player\_id].x;

int p\_y = clients[player\_id].y;

int n\_x = clients[npc\_id].x;

int n\_y = clients[npc\_id].y;

int distance\_x = abs(p\_x - n\_x);

int distance\_y = abs(p\_y - n\_y);

//1초마다 한칸 씩 플레이어 쪽으로 간다.

if (distance\_x > distance\_y)

{

if (n\_x > p\_x)

clients[npc\_id].x--;

else if (n\_x < p\_x)

clients[npc\_id].x++;

}

else if (distance\_x < distance\_y)

{

if (n\_y > p\_y)

clients[npc\_id].y--;

else if (n\_y < p\_y)

clients[npc\_id].y++;

}

else if (distance\_x == distance\_y)

{

if (n\_x > p\_x)

{

clients[npc\_id].x--;

clients[npc\_id].y--;

}

else if (n\_x < p\_x)

{

clients[npc\_id].x++;

clients[npc\_id].y++;

}

}

//agro\_npc랑 만남

if (distance\_x <= 1 && distance\_y <= 1)

{

clients[npc\_id].x = n\_x;

clients[npc\_id].y = n\_y;

cout << "잡았다!" << endl;

//TODO

push\_timer\_event(npc\_id, player\_id, 1, EVENT\_NPC\_AGRO\_ATTACK);

}

}

1. 맵 생성 알고리즘

* 맵 크기(2000x2000)만큼의 규모 내에서 각 타일의 좌상, 우상, 우하, 좌하에 대한 정점과 텍스처의 좌표 값을 정해준다.

for (unsigned int i = 0; i < mapSize.x; ++i)

{

for (unsigned int j = 0; j < mapSize.y; ++j)

{

int tileNumber = tiles.data()[i + j \* mapSize.x];

Vertex\* quad = &vertices[(i + j \* mapSize.x) \* 4]; //vertex 4개 당 quad 하나

float tileX = (float)tileSize.x;

float tileY = (float)tileSize.y;

int texU = tileNumber % (tileSet->getSize().x / tileSize.x);

int texV = tileNumber / (tileSet->getSize().x / tileSize.x);

quad[0].position = Vector2f(i \* tileX, j \* tileY); //좌상

quad[1].position = Vector2f((i + 1) \* tileX, j \* tileY); //우상

quad[2].position = Vector2f((i + 1) \* tileX, (j + 1) \* tileY); //우하

quad[3].position = Vector2f(i \* tileX, (j + 1) \* tileY); //좌하

//텍스처에 대한 텍스코드를 정해준다.

quad[0].texCoords = Vector2f(texU \* tileX, texV \* tileY); //좌상

quad[1].texCoords = Vector2f((texU + 1) \* tileX, texV \* tileY); //우상

quad[2].texCoords = Vector2f((texU + 1) \* tileX, (texV + 1) \* tileY); //우하

quad[3].texCoords = Vector2f(texU \* tileX, (texV + 1) \* tileY); //좌하

}

}

1. 로그인 알고리즘

* 위 간단한 설명에 대한 자세한 코드

switch (packet\_type) {

case CS\_PACKET\_LOGIN: {

cs\_packet\_login\* packet = reinterpret\_cast<cs\_packet\_login\*>(p);

bool insert\_once = false;

bool skip = false;

if (packet->is\_dummy == 0) //더미 클라가 아니면(사람이면)

{

for (auto& d : g\_db)

{

//만약 클라가 입력한 아이디와 db에 저장된 아이디가 같다면

if (strcmp(packet->name, d.userName) == 0)

{

if (login\_players.size() == 0)

{

//login\_players.emplace\_back(packet->name);

cl.login\_success = true;

cl.x = d.x;

cl.y = d.y;

cl.level = d.level;

skip = true;

break;

}

else

{

for (auto& lp : login\_players) //로그인 한 플레이어들의 아이디와 비교 후

{

if (strcmp(lp, packet->name) == 0) //같으면

{

//이미 동일한 아이디로 로그인 한 유저가 있음.

cout << "이미 동일한 아이디로 로그인 한 유저가 있음." << endl;

//로그인 실패(아이디 중복)

send\_login\_fail\_packet(client\_id, 0);

Disconnect(client\_id);

skip = true;

break;

}

else //다르면 로그인 성공

{

cl.login\_success = true;

//저장된 플레이어의 좌표값

cl.x = d.x;

cl.y = d.y;

cl.level = d.level;

skip = true;

break;

}

}

}

}

//다 돌았는데 같은 아이디가 없다.

}

//새로운 플레이어 정보insert

if (insert\_once == false && skip != true)

{

WCHAR\* temp;

int nChars = MultiByteToWideChar(CP\_ACP, 0, packet->name, -1, NULL, 0);

temp = new WCHAR[nChars];

MultiByteToWideChar(CP\_ACP, 0, packet->name, -1, (LPWSTR)temp, nChars);

int new\_id = g\_db.back().id + 1;

sql\_new\_player\_insert(new\_id, temp, 1, 10, 10);

insert\_once = true;

cl.login\_success = true;

cl.x = 10;

cl.y = 10;

cl.level = 1;

}

}

else { //더미 클라이면 그냥 로그인success

cl.login\_success = true;

cl.is\_dummy = true;

}

strcpy\_s(cl.name, packet->name);

if (cl.login\_success)

{

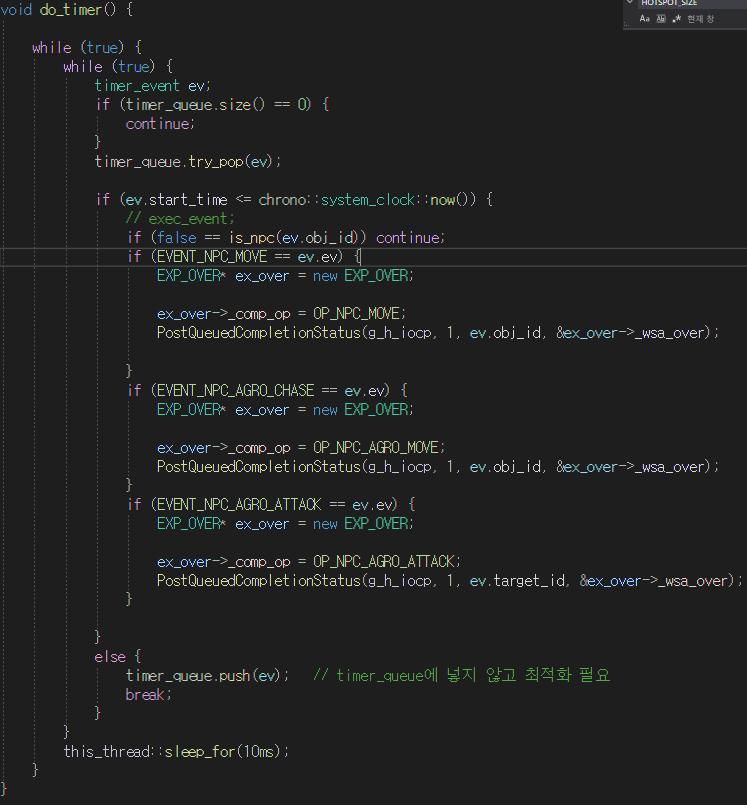
login\_players.emplace\_back(cl.name); //로그인 성공하면 login\_players 벡터에 집어넣기

send\_login\_ok\_packet(client\_id);

…

1. 타이머 큐 관련 타이머 스레드

* 밑에 캡쳐 사진 첨부.
* NPC 관련 처리해야 할 각각의 AI(move, agro\_move(chase), agro\_attack)에 대하여 worker\_thread로 일을 떠넘기고 그 일을 worker\_thread에서 처리한다.
* +추가설명) 몬스터 npc의 1초에 한 번씩 배회하는 이동에 대한 처리 같은 경우에는 최적화를 위해서 is\_active 변수를 이용하여 플레이어가 npc를 깨울 경우에만 active 상태로 만들었고 플레이어가 시야에서 사라질 경우 active를 false 처리하여 비활성화 하였다.

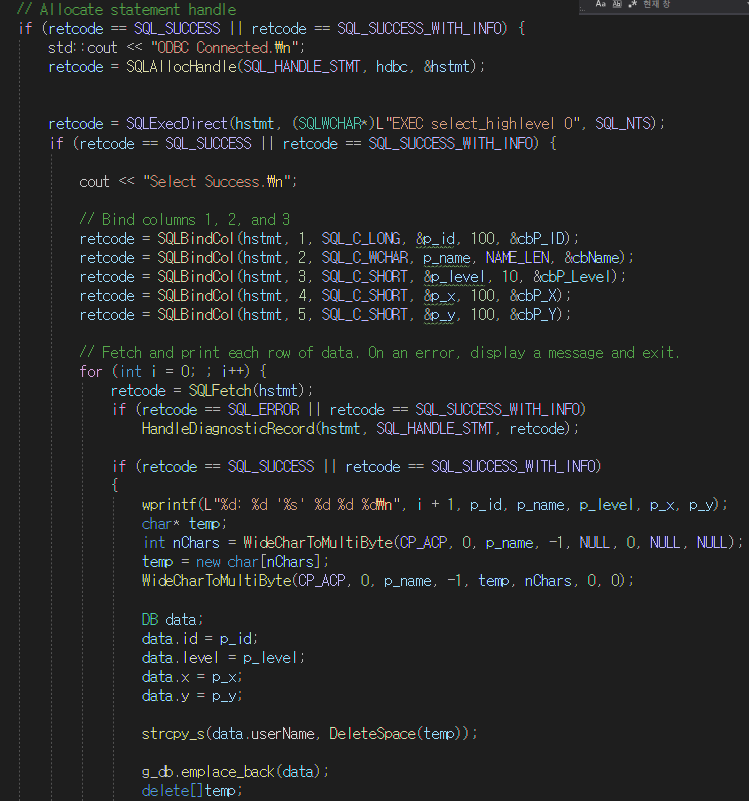


1. SQL DB 쿼리

* void sql\_player\_data\_read(): DB에 저장된 데이터(id, username, id, x, y) 들을 가져오는 함수

텍스트이(가) 표시된 사진

자동 생성된 설명



* void sql\_player\_data\_save(wchar\_t\* name, int level, int x, int y): 플레이어가 종료 시 플레이어 정보를 update한 상태로 저장. (단, release 모드로 실행 시 저장이 안되는 문제가 있음.)

SQLWCHAR query[1000];

wsprintf(query, L"UPDATE player\_data SET player\_level = %d, player\_x = %d, player\_y = %d WHERE player\_name = '%s'", level, x, y, name);

* void sql\_player\_level\_realtime\_update(wchar\_t\* name, int level): 몬스터를 잡을 때, 몬스터에게 공격당했을 때 레벨 실시간 업데이트

wsprintf(query, L"UPDATE player\_data SET player\_level = %d WHERE player\_name = '%s'", level, name);

* void sql\_new\_player\_insert(int id, wchar\_t\* name, int level, int x, int y): DB에 존재하지 않는 아이디를 입력했을 때 DB에 새로운 플레이어 정보를 넣고 로그인 성공시킴.

wsprintf(query, L"INSERT INTO player\_data (player\_id, player\_name, player\_level, player\_x, player\_y) VALUES (%d, '%s', %d, %d, %d)",id, name, level, x, y);

1. 텀 프로젝트 종료 후 깨달은 문제점.
2. Release 모드 실행 시 클라이언트를 종료했을 때, 플레이어 정보가 저장되지 않는다. (Debug 모드에서는 정상 작동)
3. 지금까지 npc의 숫자가 20만 마리가 아닌 2만에 놓고 stress test를 하고 있었다. 20만 마리로 바꾸니 서버가 버티질 못한다.
4. 2번에 대한 문제를 생각해보니 agro 몬스터와 관련 있는 듯하다. 플레이어가 다가가서 npc를 깨우는 방식으로 했어야 하는데, npc가 스스로 플레이어를 발견해서 해당 플레이어의 id를 agro\_list에 넣고 그 플레이어를 쫓아가는 방식으로 구현했다.
5. 발표를 너무 간략하게 한 듯하다. 게임설명서 내용처럼 내가 어떤 식으로 구현했는지 코드 스크린샷과 더불어 설명할 걸 후회가 된다.