게임 서버 프로그래밍

Term Project

2020182042 최준하

목차 table of contents

커맨드

방향키(→,←,↑,↓): 플레이어 이동

A: 플레이어 공격

T: 채팅

ESC: 프로그램 종료

구현 내용

플레이어

- 공격 (1초에 한번)
- 이동 (1초에 두번)
- 채팅
- 스프라이트

몬스터

- 몬스터 종류 (3가지)
- 부활 (30초)
- 이동(고정, 로밍)

시스템

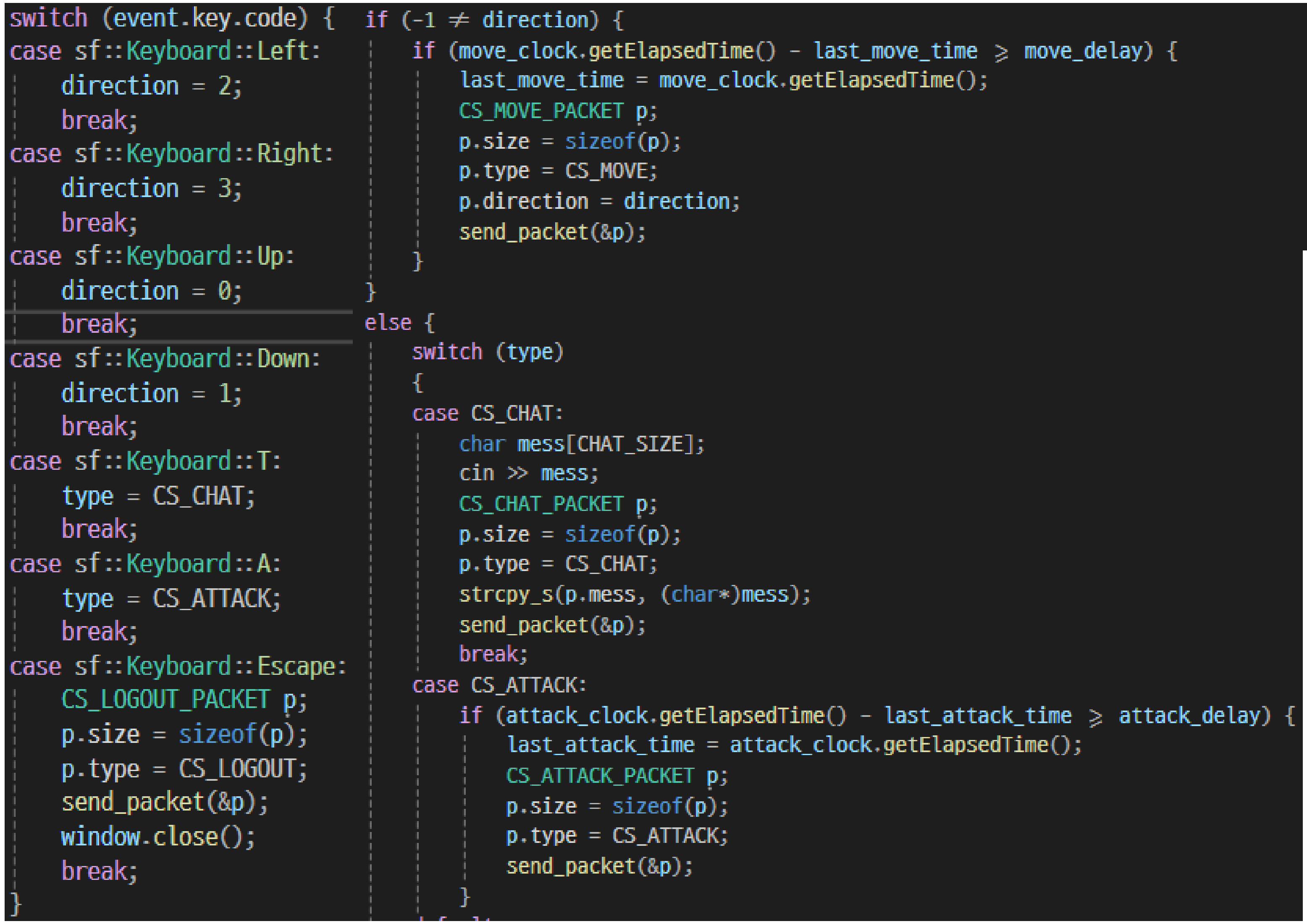
- 시스템 메시지 전송
  (사망, 레벨 업, 공격, 피격)
- 5초마다 DB 자동저장

장애물

- 랜덤 배치

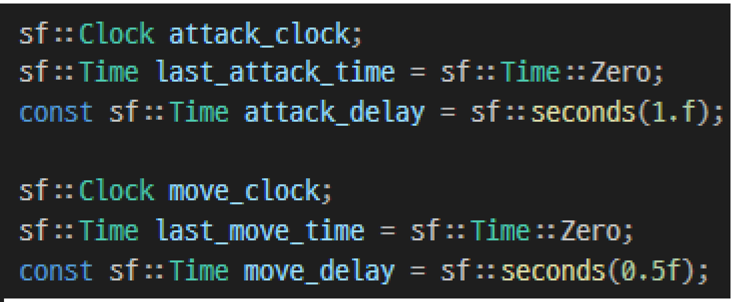
Part 2

플레이어



```
switch (event.key.code) {
case sf::Keyboard::Left:
    direction = 2;
    break;
case sf::Keyboard::Right:
    direction = 3;
    break;
case sf::Keyboard::Up:
    direction = 0;
    break;
case sf::Keyboard::Down:
    direction = 1;
    break;
case sf::Keyboard::T:
    type = CS_CHAT;
    break;
case sf::Keyboard::A:
    type = CS_ATTACK;
    break;
case sf::Keyboard::Escape:
    CS_LOGOUT_PACKET p;
    p.size = sizeof(p);
    p.type = CS_LOGOUT;
    send_packet(&p);
    window.close();
    break;
}
```

```
if (-1 ≠ direction) {
    if (move_clock.getElapsedTime() - last_move_time ≥ move_delay) {
        last_move_time = move_clock.getElapsedTime();
        CS_MOVE_PACKET p;
        p.size = sizeof(p);
        p.type = CS_MOVE;
        p.direction = direction;
        send_packet(&p);
    }
}
else {
    switch (type)
    {
    case CS_CHAT:
        char mess[CHAT_SIZE];
        cin >> mess;
        CS_CHAT_PACKET p;
        p.size = sizeof(p);
        p.type = CS_CHAT;
        strcpy_s(p.mess, (char*)mess);
        send_packet(&p);
        break;
    case CS_ATTACK:
        if (attack_clock.getElapsedTime() - last_attack_time ≥ attack_delay) {
            last_attack_time = attack_clock.getElapsedTime();
            CS_ATTACK_PACKET p;
            p.size = sizeof(p);
            p.type = CS_ATTACK;
            send_packet(&p);
        }
```



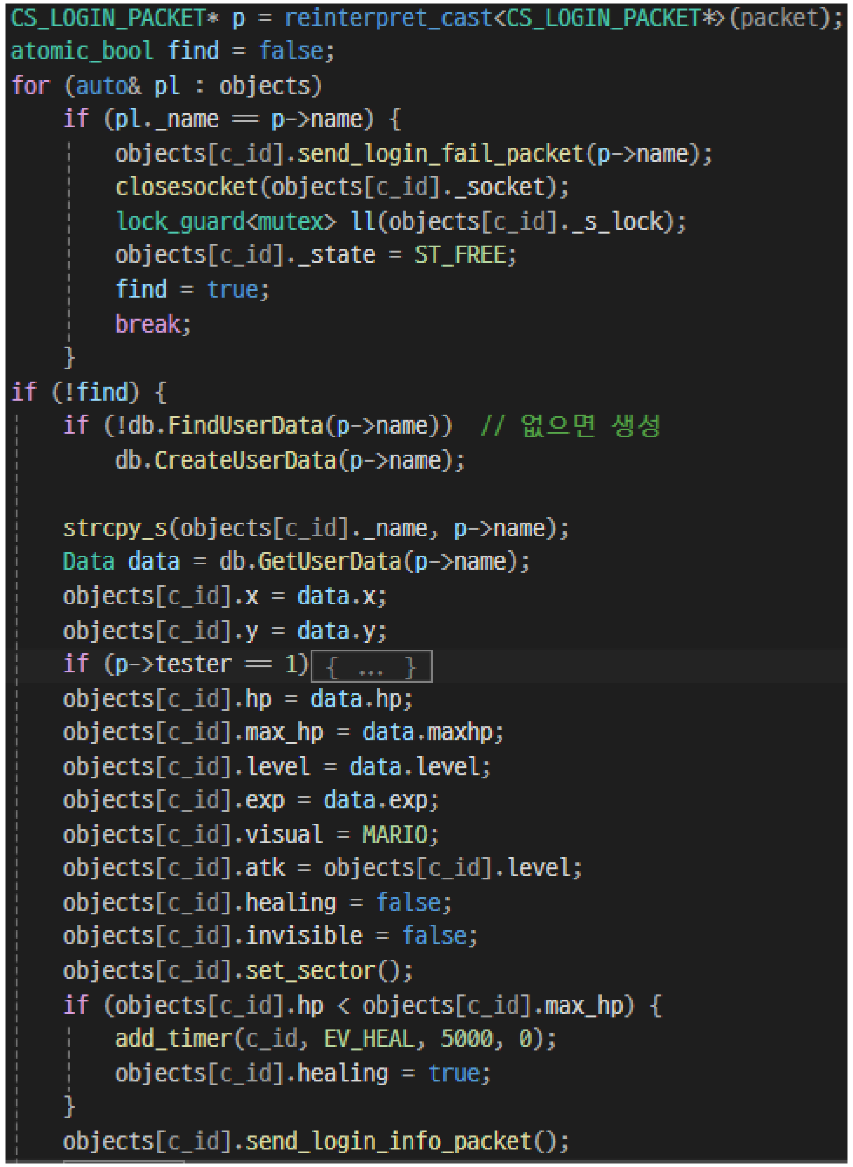
```
sf::Clock attack_clock;
sf::Time last_attack_time = sf::Time::Zero;
const sf::Time attack_delay = sf::seconds(1.f);

sf::Clock move_clock;
sf::Time last_move_time = sf::Time::Zero;
const sf::Time move_delay = sf::seconds(0.5f);
```

커맨드에 해당하는 패킷 전송

공격과 이동은 마지막으로 패킷을 보낸 시간을 측정하여 지연 시간 부여

Part 2

플레이어



```
CS_LOGIN_PACKET* p = reinterpret_cast<CS_LOGIN_PACKET*>(packet);
atomic_bool find = false;
for (auto& pl : objects)
    if (pl._name == p->name) {
        objects[c_id].send_login_fail_packet(p->name);
        closesocket(objects[c_id]._socket);
        lock_guard<mutex> ll(objects[c_id]._s_lock);
        objects[c_id]._state = ST_FREE;
        find = true;
        break;
    }
if (!find) {
    if (!db.FindUserData(p->name))  // 없으면 생성
        db.CreateUserData(p->name);

    strcpy_s(objects[c_id]._name, p->name);
    Data data = db.GetUserData(p->name);
    objects[c_id].x = data.x;
    objects[c_id].y = data.y;
    if (p->tester == 1) { ... }
    objects[c_id].hp = data.hp;
    objects[c_id].max_hp = data.maxhp;
    objects[c_id].level = data.level;
    objects[c_id].exp = data.exp;
    objects[c_id].visual = MARIO;
    objects[c_id].atk = objects[c_id].level;
    objects[c_id].healing = false;
    objects[c_id].invisible = false;
    objects[c_id].set_sector();
    if (objects[c_id].hp < objects[c_id].max_hp) {
        add_timer(c_id, EV_HEAL, 5000, 0);
        objects[c_id].healing = true;
    }
    objects[c_id].send_login_info_packet();
```

로그인 했을 때 이미 로그인 한 이름(id)
라면 login_fail_packet 전송

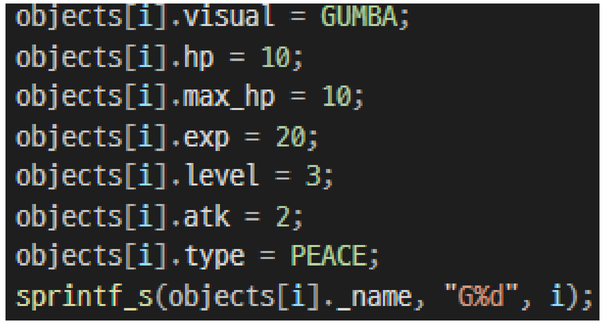
Database에 해당하는 id 가 없다면 새로 생성

있다면 해당 유저 데이터를 받아와서 설정한다.

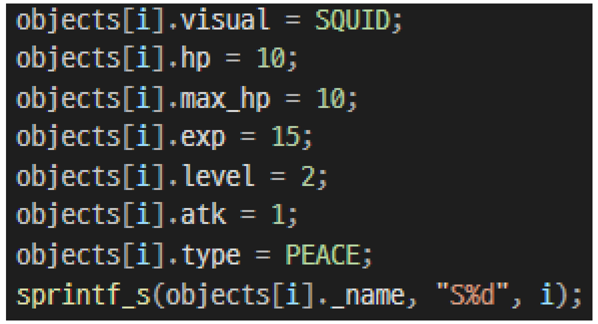
공격력 수치 atk = level
체력 수치 max_hp = 10 * level
경험치 통 수치 = 2^(level-1) * 100

Part 2

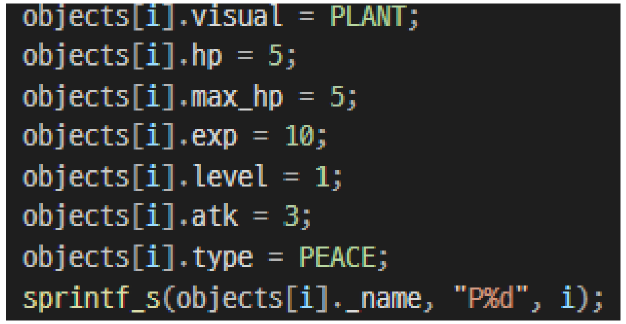
몬스터



```
objects[i].visual = GUMBA;
objects[i].hp = 10;
objects[i].max_hp = 10;
objects[i].exp = 20;
objects[i].level = 3;
objects[i].atk = 2;
objects[i].type = PEACE;
sprintf_s(objects[i]._name, "G%d", i);
```



```
objects[i].visual = SQUID;
objects[i].hp = 10;
objects[i].max_hp = 10;
objects[i].exp = 15;
objects[i].level = 2;
objects[i].atk = 1;
objects[i].type = PEACE;
sprintf_s(objects[i]._name, "S%d", i);
```



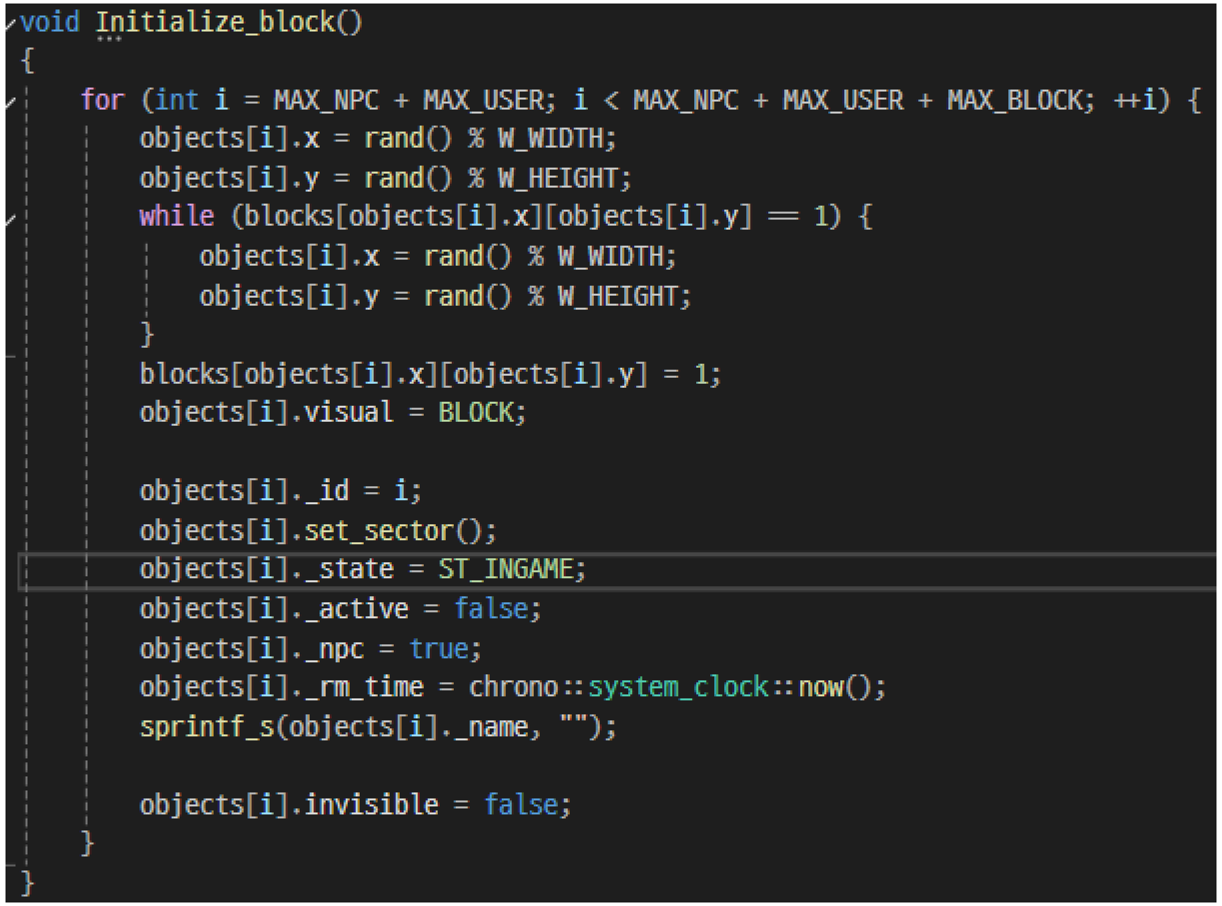
```
objects[i].visual = PLANT;
objects[i].hp = 5;
objects[i].max_hp = 5;
objects[i].exp = 10;
objects[i].level = 1;
objects[i].atk = 3;
objects[i].type = PEACE;
sprintf_s(objects[i]._name, "P%d", i);
```

로밍형

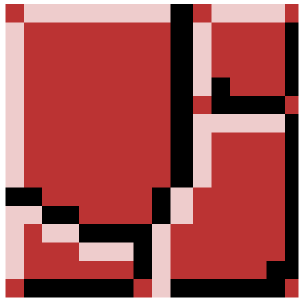
로밍형

고정형

장애물

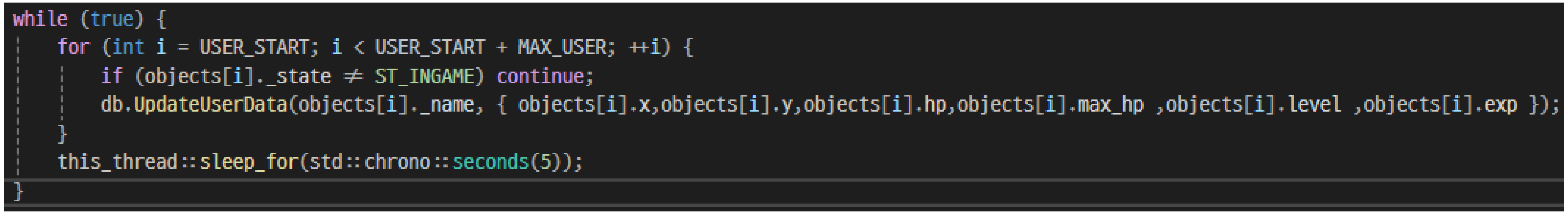


```
void Initialize_block()
{
    for (int i = MAX_NPC + MAX_USER; i < MAX_NPC + MAX_USER + MAX_BLOCK; ++i) {
        objects[i].x = rand() % W_WIDTH;
        objects[i].y = rand() % W_HEIGHT;
        while (blocks[objects[i].x][objects[i].y] == 1) {
            objects[i].x = rand() % W_WIDTH;
            objects[i].y = rand() % W_HEIGHT;
        }
        blocks[objects[i].x][objects[i].y] = 1;
        objects[i].visual = BLOCK;

        objects[i]._id = i;
        objects[i].set_sector();
        objects[i]._state = ST_INGAME;
        objects[i]._active = false;
        objects[i]._npc = true;
        objects[i]._rm_time = chrono::system_clock::now();
        sprintf_s(objects[i]._name, "");

        objects[i].invisible = false;
    }
}
```



장애물을 배치 시 중복 설치 배제

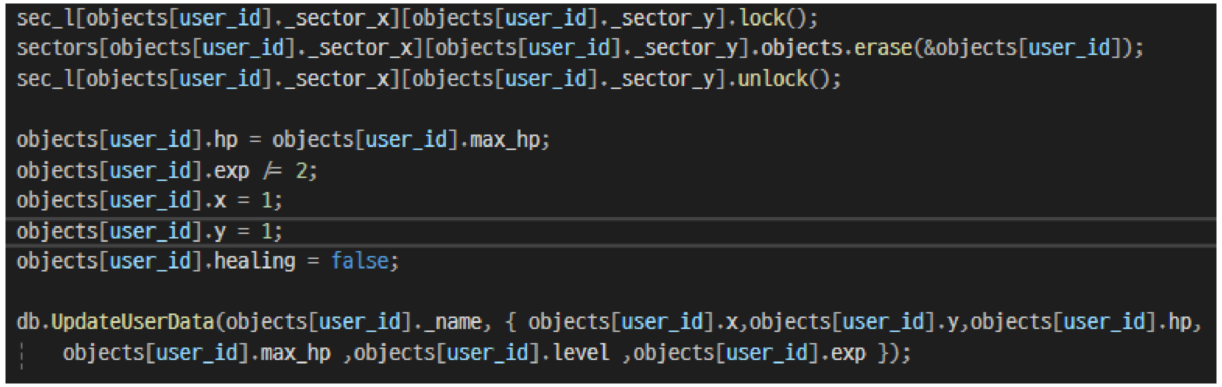
시스템



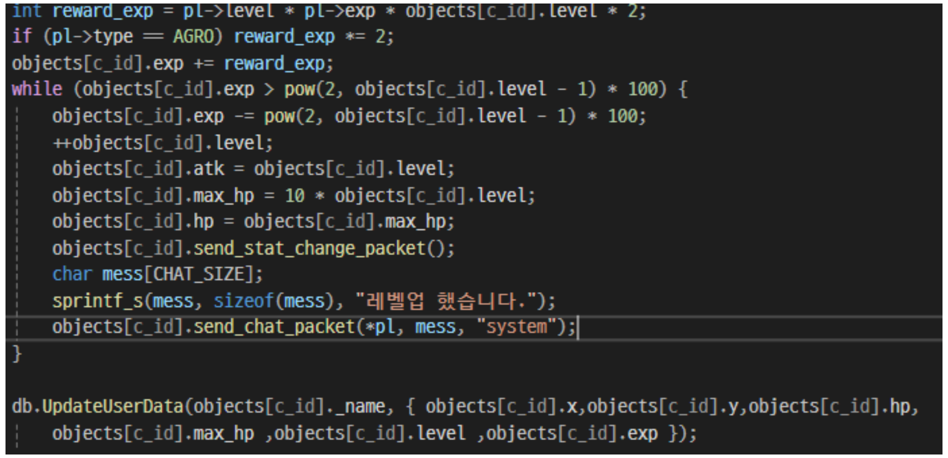
```cpp
while (true) {
    for (int i = USER_START; i < USER_START + MAX_USER; ++i) {
        if (objects[i]._state != ST_INGAME) continue;
        db.UpdateUserData(objects[i]._name, { objects[i].x,objects[i].y,objects[i].hp,objects[i].max_hp ,objects[i].level ,objects[i].exp });
    }
    this_thread::sleep_for(std::chrono::seconds(5));
}
```

DataBase 저장

1. 5초마다 자동저장
2. 플레이어 사망
3. 플레이어 경험치 획득



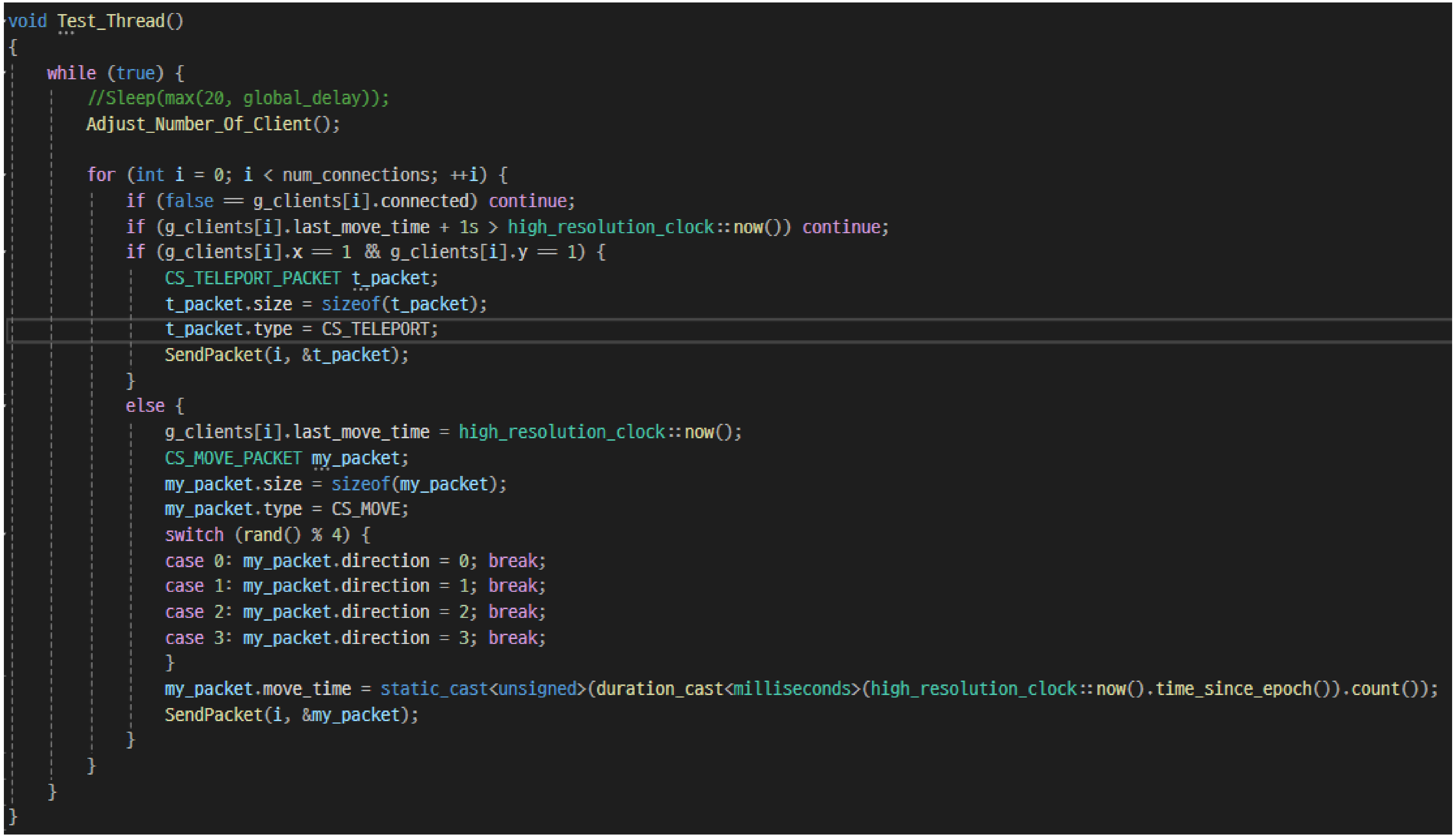
```cpp
sec_l[objects[user_id]._sector_x][objects[user_id]._sector_y].lock();
sectors[objects[user_id]._sector_x][objects[user_id]._sector_y].objects.erase(&objects[user_id]);
sec_l[objects[user_id]._sector_x][objects[user_id]._sector_y].unlock();

objects[user_id].hp = objects[user_id].max_hp;
objects[user_id].exp /= 2;
objects[user_id].x = 1;
objects[user_id].y = 1;
objects[user_id].healing = false;

db.UpdateUserData(objects[user_id]._name, { objects[user_id].x,objects[user_id].y,objects[user_id].hp,
    objects[user_id].max_hp ,objects[user_id].level ,objects[user_id].exp });
```



```cpp
int reward_exp = pl->level * pl->exp * objects[c_id].level * 2;
if (pl->type == AGRO) reward_exp *= 2;
objects[c_id].exp += reward_exp;
while (objects[c_id].exp > pow(2, objects[c_id].level - 1) * 100) {
    objects[c_id].exp -= pow(2, objects[c_id].level - 1) * 100;
    ++objects[c_id].level;
    objects[c_id].atk = objects[c_id].level;
    objects[c_id].max_hp = 10 * objects[c_id].level;
    objects[c_id].hp = objects[c_id].max_hp;
    objects[c_id].send_stat_change_packet();
    char mess[CHAT_SIZE];
    sprintf_s(mess, sizeof(mess), "레벨업 했습니다.");
    objects[c_id].send_chat_packet(*pl, mess, "system");
}

db.UpdateUserData(objects[c_id]._name, { objects[c_id].x,objects[c_id].y,objects[c_id].hp,
    objects[c_id].max_hp ,objects[c_id].level ,objects[c_id].exp });
```

Part 3

STRESS TEST



```cpp
void Test_Thread()
{
    while (true) {
        //Sleep(max(20, global_delay));
        Adjust_Number_Of_Client();

        for (int i = 0; i < num_connections; ++i) {
            if (false == g_clients[i].connected) continue;
            if (g_clients[i].last_move_time + 1s > high_resolution_clock::now()) continue;
            if (g_clients[i].x == 1 && g_clients[i].y == 1) {
                CS_TELEPORT_PACKET t_packet;
                t_packet.size = sizeof(t_packet);
                t_packet.type = CS_TELEPORT;
                SendPacket(i, &t_packet);
            }
            else {
                g_clients[i].last_move_time = high_resolution_clock::now();
                CS_MOVE_PACKET my_packet;
                my_packet.size = sizeof(my_packet);
                my_packet.type = CS_MOVE;
                switch (rand() % 4) {
                case 0: my_packet.direction = 0; break;
                case 1: my_packet.direction = 1; break;
                case 2: my_packet.direction = 2; break;
                case 3: my_packet.direction = 3; break;
                }
                my_packet.move_time = static_cast<unsigned>(duration_cast<milliseconds>(high_resolution_clock::now().time_since_epoch()).count());
                SendPacket(i, &my_packet);
            }
        }
    }
}
```