포트폴리오

서버 프로그래머 지원

목차

The Toys

목적 : 언리얼과 C++ 서버 연동

개발 기간 : 2023년 12월 ~ 2024년 7월

사용 도구 : C++(서버), Unreal 5(클라이언트)

개발 인원 : 3명 (서버 1명, 클라이언트 1명, 모델러/기획자 1명)

담당 업무 : IOCP 로비 서버 구현,

Boost ASIO 게임 서버 구현,

로비 서버에서 매칭 시스템 구현,

비 정상적인 클라이언트 조작 방지 구현

The Toys

The Toys
서버 요약

게임 실행시 로비 서버와 연결을 함.

회원가입 후, 아이디와 비밀번호를 사용해 로그인을 하면 로비로 이동.

로비에서 도망자, 술래 중 역할과 캐릭터 선택 후 확인을 누르면 매칭이 잡힘.

매칭이 잡히면 게임 서버에 연결.

술래가 이기거나 도망자가 이겨서 게임이 끝날 시. 결과화면이 뜨고. 로비로 이동을 누르면 게임서버와 연결을 끊고 로비로 이동.

The Toys
IOCP 로비 서버

GameServer와 Clients에게
각각 다른 포트로 접속 하게 한 뒤.
각각 다른 Key값을 주어.
이 후에는 key값으로 구분.

```
case OP_ACCEPT: {
    if (static_cast<int>(key) == 9000) {
        lsession.socket_ = l_c_socket;
        lsession.id_ = 100'000;
        lsession.prev_remain_ = 0;
        CreateIoCompletionPort(reinterpret_cast<HANDLE>(l_c_socket), h_iocp, 100'000, 0);
        lsession.do_recv();
        l_c_socket = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
        ZeroMemory(&l_a_over.over, sizeof(l_a_over.over));
        int addr_size = sizeof(SOCKADDR_IN);
        AcceptEx(l_s_socket, l_c_socket, l_a_over.send_buf, 0, addr_size + 16, addr_size + 16, 0, &l_a_over.over);
    }
    else {
        int client_id = get_new_client_id();
        if (client_id != -1) {
            {
                lock_guard<mutex> ll(clients[client_id].s_lock_);
                clients[client_id].state_ = ST_ALLOC;
            }
            clients[client_id].id_ = client_id;
            clients[client_id].prev_remain_ = 0;
            clients[client_id].socket_ = g_c_socket;
            CreateIoCompletionPort(reinterpret_cast<HANDLE>(g_c_socket),
                h_iocp, client_id, 0);
            clients[client_id].do_recv();
            g_c_socket = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
        }
        else {
            cout << "Max user exceeded.₩n";
        }
        ZeroMemory(&g_a_over.over, sizeof(g_a_over.over));
        int addr_size = sizeof(SOCKADDR_IN);
        AcceptEx(g_s_socket, g_c_socket, g_a_over.send_buf, 0, addr_size + 16, addr_size + 16, 0, &g_a_over.over);
    }
    break;
}
```

The Toys
IOCP 로비 서버

술래 1명, 도망자 4명이 모이면
플레이어들을 빼 와서
게임을 시작할 준비를 함.

```
bool allPlayersReady = false;
if (ChaserQueue.unsafe_size() >= MAX_CHASER_NUM) {
    if (RunnerQueue.unsafe_size() >= MAX_RUNNER_NUM) {
        allPlayersReady = true;
    }
}

if (allPlayersReady) {
    int chaser;
    if (!ChaserQueue.try_pop(chaser))
        break;
    if (clients[chaser].state_ == ST_FREE)
        break;
    int runners[MAX_RUNNER_NUM];
    for (int i = 0; i < MAX_RUNNER_NUM; ++i) {
        runners[i] = -1;
        if (!RunnerQueue.try_pop(runners[i])) {
            for (int rn : runners) {
                if (rn == -1) {
                    break;
                }
                RunnerQueue.push(rn);
            }
            ChaserQueue.push(chaser);
            return;
        }
        if (clients[runners[i]].state_ == ST_FREE) {
            i--;
            continue;
        }
    }
```

The Toys
IOCP 로비 서버

매칭이 잡히면
관리하는 방이 가장 적은
게임서버의 쓰레드에게
방 생성 요청을 보낸 후
각각의 클라이언트에게
접속해야 할 게임서버의
주소 및 포트번호를 전송

```
while (true) {
    int thread_index = 0;
    int min_thread_contention = 0x7FFFFFFF;
    for (int i = 0; i < GameServerThreadContention.size(); ++i) {
        if (GameServerThreadContention[i] < min_thread_contention) {
            min_thread_contention = GameServerThreadContention[i];
            thread_index = i;
        }
    }
    if (thread_index >= 0 && thread_index < GameServerThreadContention.size()) {
        if (GameServerThreadContention[thread_index].compare_exchange_weak(min_thread_contention, min_thread_contention + 1)) {
            int room_num = GameServerThreadRoomCount[thread_index]++;
            lsession.SendCreateRoomPacket(chaser, runners);
            if (chaser >= 0 && chaser <MAX_USER) {
                clients[chaser].SendGameStartPacket(GameServerPortNums[thread_index]);
                for (int rn : runners) {
                    clients[rn].SendGameStartPacket(GameServerPortNums[thread_index]);
                }
                break;
            }
        }
    }
}
```

The Toys
IOCP 게임 서버

게임서버가 로비 서버에 연결(Connect) 시도.
로비서버는 항상 Accept 만 하도록 함.

```cpp
class cClient : public std::enable_shared_from_this<cClient> {
public:
    cClient(boost::asio::io_context& io_context, const std::string& host, const std::string& port)
        : io_context_(io_context), socket_(io_context) {
        tcp::resolver resolver(io_context_);
        auto endpoints = resolver.resolve(host, port);
        do_connect(endpoints);
    }

    void do_write(unsigned char* packet, std::size_t length)
    {
        socket_.async_write_some(boost::asio::buffer(packet, length), [this, packet, length](boost::system::error_code ec, std::size_t bytes_transferred) {
            if (!ec)
            {
                if (length != bytes_transferred) {
                    cout << "Incomplete Send occured on Session To Lobby Server. This Session should be closed.₩n";
                }
                delete packet;
            }
            });
    }
    void send_packet(void* packet)
    {
        int packet_size = reinterpret_cast<unsigned char*>(packet)[0];
        unsigned char* buff = new unsigned char[packet_size];
        memcpy(buff, packet, packet_size);
        do_write(buff, packet_size);
    }
private:
    void do_connect(const tcp::resolver::results_type& endpoints) {
        boost::asio::async_connect(socket_, endpoints,
            [this](boost::system::error_code ec, tcp::endpoint) {
                if (!ec) {
                    GAME_SERVER_OPENED_PACKET p;
                    p.size = sizeof(GAME_SERVER_OPENED_PACKET);
                    p.type = GAME_SERVER_OPENED;
                    memcpy(p.address, "127.0.0.1", sizeof("127.0.0.1"));
                    p.portNum = 9001;
                    send_packet(&p);
                    do_read();
                }
            });
    }
```

The Toys
IOCP 게임 서버

```cpp
class cServer
{
private:
    tcp::acceptor   acceptor;
    void do_Accept()
    {
        acceptor.async_accept([this](boost::system::error_code ec, tcp::socket socket) {
            if (!ec)
            {
                int p_id = Get_New_ClientID();
                clients[p_id] = std::make_shared<cSession>(std::move(socket), p_id);
                clients[p_id]->ingame_ = true;
                clients[p_id]->Start();
                do_Accept();
            }
            else {
                cout << "ERROR : " << ec.what() << endl;
            }
            });
    }

public:
    cServer(boost::asio::io_context& io_service, int port, int thread_num) : acceptor(io_service, tcp::endpoint(tcp::v4(), port))
    {
        do_Accept();
    }
};
```

매칭이 잡힌
클라이언트들이 로비서버에서 준 주소로
게임서버에 접속을 하면
게임서버에서는 해당 Room 플레이어들이
모두 접속할때까지 대기하였다가.
모든 플레이어들이 접속을 하면
게임을 시작 함.

```cpp
int my_count = WaitingQueue[p->GroupNum]++;
if (my_count == MAX_ROOM_PLAYER-1)
    isGroupReady = true;
WaitingMap[p->GroupNum][my_count] = c_id;

if (isGroupReady) {
    WaitingQueue.erase(p->GroupNum);
    IngameMapData igmd;
    int player_count = 1;
    for (int id : WaitingMap[p->GroupNum]) {
        if (clients[id]->charactor_num_ >= 6) {
            igmd.player_ids_[0] = id;
        }
        else {
            igmd.player_ids_[player_count++] = id;
        }
    }
```

Academy RPG

목적 : 소규모 MMORPG 제작

개발 기간 : 2024년 6월 14일 ~ 2024년 6월 18일

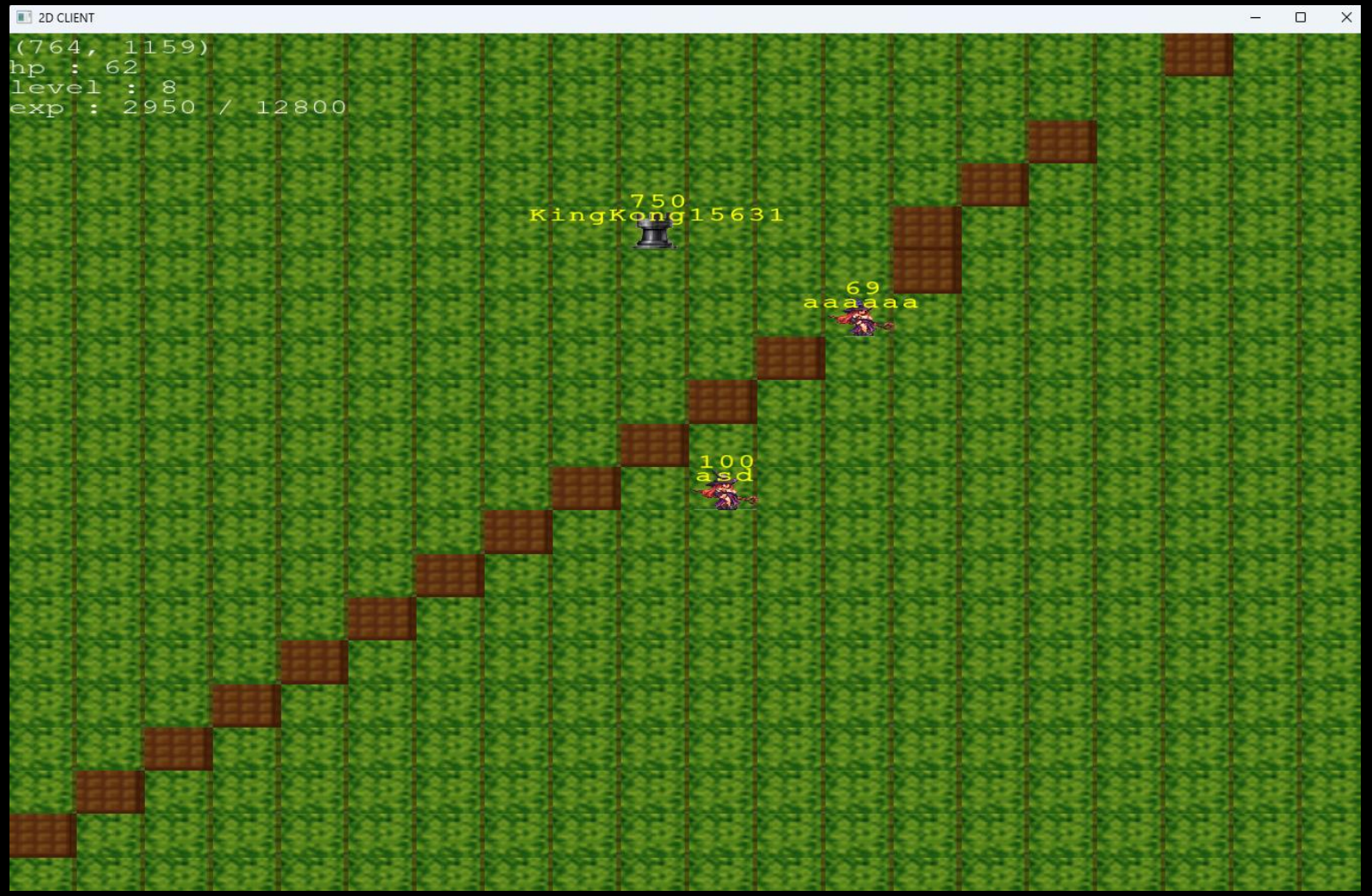
사용 도구 : C++(서버, 클라이언트)

개발 인원 : 1명

담당 업무 : MultiThread와 IOCP를 사용한 소규모 MMORPG 서버 제작.

간단한 클라이언트 제작.

Academy RPG


2D CLIENT
(764, 1159)
hp : 62
level : 8
exp : 2950 / 12800
750
KingKong15631
69
aaaaaa
100
asd

Academy RPG

몬스터 20만 마리 배치

몬스터 5종류 구현

보스 몬스터 구현

몬스터별 공격범위를 다르게 구현

몬스터 레벨에 따른 공격력, 체력 구현

고정몬스터 + 로밍몬스터 구현

몬스터 사망시 30초 후 부활하도록 구현

플레이어 3종류 구현

플레이어 3종류
Warrior : 시선방향 1칸 공격
Mage : 시선방향 3칸 공격
Prist : 주위 5칸 공격 + 주위 5칸 플레이어 힐

경험치 시스템, 레벨업, 성장 공격력, 방어력 구현

Academy RPG

로그인시.
나와 내 주변 8방향 섹터 확인 후
섹터 안에 있는 플레이어에 대하여
NPC면 깨우고 Player면 그 플레이어에게
나를 추가하도록 알림

```
SL.lock();
for (auto& pl : Sector) {
    if (clients[pl.first].in_use_ == false) continue;
    if (clients[pl.first].id_ == c_id) continue;
    if (false == can_see(c_id, pl.first))
        continue;
    if (is_pc(pl.first)) clients[pl.first].send_add_object_packet(c_id);
    else WakeUpNPC(pl.first, c_id);
    clients[c_id].send_add_object_packet(pl.first);
}
SL.unlock();
```

Academy RPG

이동 시.
내 섹터 다시 설정 및, 내 주변 섹터에 있는 플레이어들에 대하여 viewList 다시설정후. 내 시야 범위 +1 안에 있는 플레이어들 에게만 move 패킷 전송

```
SL.lock();
for (auto& cl : Sector) {
    if (clients[cl.first].in_use_ == false) continue;
    if (clients[cl.first].state_ != ST_INGAME) continue;
    if (clients[cl.first].id_ == c_id) continue;
    if (can_see(c_id, clients[cl.first].id_))
        near_list.insert(clients[cl.first].id_);
}
SL.unlock();
clients[c_id].send_move_packet(c_id);

for (auto& pl : near_list) {
    auto& cpl = clients[pl];
    if (is_pc(pl)) {
        cpl.vl_.lock();
        if (clients[pl].view_list_.count(c_id)) {
            cpl.vl_.unlock();
            clients[pl].send_move_packet(c_id);
        }
        else {
            cpl.vl_.unlock();
            clients[pl].send_add_object_packet(c_id);
        }
    }
    else WakeUpNPC(pl, c_id);

    if (old_vlist.count(pl) == 0)
        clients[c_id].send_add_object_packet(pl);
}

for (auto& pl : old_vlist) {
    if (0 == near_list.count(pl)) {
        clients[c_id].send_remove_player_packet(pl);
        if (is_pc(pl))
            clients[pl].send_remove_player_packet(c_id);
    }
}
break;
```

Academy RPG

내가 공격했을때 적이 죽었으면, 30초 뒤에 부활하도록 Timer event에 넣어줌.
그 후 경험치 획득을 함.
일정 경험치 이상 쌓이면 레벨업 후,
필요 경험치량 2배 증가

```
if (clients[pl].hp_ <= 0) {
    clients[pl].in_use_ = false;
    TIMER_EVENT ev{ pl, chrono::system_clock::now() + 30s, EV_RESURRECTION, 0 };
    timer_queue.push(ev);
    clients[c_id].exp_ += clients[pl].level_* 50;
    while (true) {
        if (clients[c_id].exp_ >= clients[c_id].max_exp_) {
            clients[c_id].exp_ -= clients[c_id].max_exp_;
            clients[c_id].max_exp_ *= 2;
            clients[c_id].level_ += 1;
            clients[c_id].update_status();
            clients[c_id].send_stat_change_packet(c_id, clients[c_id].max_hp_,
                clients[c_id].hp_, clients[c_id].level_, clients[c_id].exp_);
        }
        else {
            clients[c_id].send_stat_change_packet(c_id, clients[c_id].max_hp_,
                clients[c_id].hp_, clients[c_id].level_, clients[c_id].exp_);
            break;
        }
    }

}
```