MMORPG PROTO TYPE

2016182024 윤선규

네트워크

[패킷 재조립]

```
for (auto need_bytes = *reinterpret_cast<packet_size_t*>(pck_start);
    need_bytes <= remain_bytes && 0 != remain_bytes;)
{
    ProcessPacket(id, pck_start);

    pck_start += need_bytes;
    remain_bytes -= need_bytes;
    need_bytes = *reinterpret_cast<packet_size_t*>(pck_start);
}

client.prerecv_size = static_cast<packet_size_t>(remain_bytes);
memmove(client.recv_over.buf.data(), pck_start, remain_bytes);
```

기존의 플랫버퍼 방식은 매 recv 호출마다 memmove 함수를 호출하여야 했다. 이를 보완하기 위해 recv 용 링버퍼를 만들었다.

```
recvbuf.MoveRear(transfered);

auto pck_start = recvbuf.Begin();
auto remain_bytes = recvbuf.Size();

for (auto need_bytes = *reinterpret_cast<packet_size_t*>(pck_start);
    need_bytes <= remain_bytes && 0 != remain_bytes;)
{
    // 링버퍼경계에 걸친 패킷
    if (recvbuf.CheckOverflowOnRead(need_bytes))
    {
        // 미완성 패킷
        if (recvbuf.Size() < need_bytes)
            continue;

        thread_local static std::byte temp_packet[MAX_PACKET_SIZE];
        auto packetsize1 = recvbuf.FilledEdgespace();
        memcpy(temp_packet, pck_start, packetsize1);
        auto packetsize2 = need_bytes - packetsize1;
        memcpy(temp_packet + packetsize1, &recvbuf, packetsize2);
        pck_start = temp_packet;
    }

    ProcessPacket(Id_, pck_start);
    recvbuf.MoveFront(need_bytes);

    pck_start = recvbuf.Begin();
    remain_bytes = recvbuf.Size();
    need_bytes = *reinterpret_cast<packet_size_t*>(pck_start);
}
```

Recv 링버퍼를 사용하여 메모리 복사를 줄일 수 있었다.
다만 매 패킷조립 루프 마다 분기체크가 하나 추가되었다.
성능 테스트 결과 둘의 차이점은 미미하여 어느 것이 더 좋은지 알 수 없었으나, 표절논란을 피하기 위해 사용했다.

링버퍼의 구현은 아래와 같다.

```
// 멤버함수로는 데이터복사등의 조작이 불가함. 데이터는 내비두고 캐럿만 변화시킴.
template<size_t N, class Contanier = std::array<std::byte, N>>
class RecvRingBuffer : protected Contanier
{
    using value_type = Contanier::value_type;
public:
    value_type* Begin() { return &Contanier::operator[](BeginIdx_); }
    value_type* End() { return &Contanier::operator[](EndIdx_); }
    size_t Size() const { return Size_; }
    size_t Capacity() const { return Contanier::size(); }
    bool Full() const { return Capacity() == Size(); }
    size_t FilledEdgespace() const { return Capacity() - BeginIdx_; }
    size_t EmptyEdgespace() const { return Capacity() - EndIdx_; }
    size_t BytesToRecv() const { ... }
    // write
    void MoveRear(size_t bytes) { ... }
    // read
    void MoveFront(size_t bytes) { ... }

    bool CheckOverflowOnRead(size_t bytes) const
    {
        bool overflowed = FilledEdgespace() < bytes;
        return overflowed;
    }
private:
    size_t BeginIdx_{}, EndIdx_{}, Size_{};
};
```

패킷을 받을 때 마다 시작 idx 와 endidx 를 조작하여 링버퍼의 개념을 추상적으로 나타내고 recv 호출시 비어있는 공간만큼 받고, 패킷이 경계에 걸쳐 있을 경우에만 다른 버퍼로 메모리를 복사하여 ProcessPacket 에 넘겨준다.

[프로토콜]

AddObj 프로토콜이 없다.

클라이언트의 캐릭터 자료구조를 Unorderd_map 으로 만들어서,

기존에 없던 캐릭터의 정보가 하나라도 들어오면, 그때 오브젝트가 생성된다.

주로 set_position 에 의해 생성되지만, 다른 set 패킷으로 생성되기도 한다.

나머지 정보들은 필요할 경우 request 패킷을 id 와 함께 보내고 set 패킷을 받는다. 이 과정을 게터 함수 내에서 처리하도록 하였다.

```
template<class T> constexpr T NeedRequest();
template<> constexpr int NeedRequest() { return -1; }
template<> constexpr float NeedRequest() { return -1; }
template<> constexpr string NeedRequest() { return {}; }

template<class T> constexpr T WaitForRequestAnswer();
template<> constexpr int WaitForRequestAnswer() { return 0; }
template<> constexpr float WaitForRequestAnswer() { return 0; }
template<> constexpr string WaitForRequestAnswer() { return { " " }; }

#define REPLICATE(var) const auto& Get##var()                    \
{                                                                \
    if (var##_ == NeedRequest<decltype(var##_)>())               \
    {                                                            \
        var##_ = WaitForRequestAnswer<decltype(var##_)>();       \
        cs_request_##var request;                                \
        request.id = GetId();                                    \
        Networker::Get().DoSend(&request);                       \
    }                                                            \
    return var##_;                                               \
```

```
class Character : public DynamicObj
{
    REPLICATE(Exp);
    REPLICATE(Money);
    REPLICATE(Name);
    REPLICATE(AttackPoint);
    REPLICATE(ArmorPoint);
    REPLICATE(AdditionalHp);
    REPLICATE(MovemetSpeed);
```

필요한 값이 서버에서 전달되지 않았을 경우, Get 함수를 사용하면 서버에 알아서 요청이 가므로, 클라이언트 개발시에 신경을 덜 써도 된다.

각 패킷을 만들 때, 타입과 사이즈가 자동으로 초기화 되도록 만들어, 생산성을 높였다.

```
template<class T>
struct packet_base
{
    packet_size_t size = sizeof(T);
    PACKET_TYPE packet_type = +PACKET_TYPE::_from_string_nocase(typeid(T).name() + 7);
};
#define PACKET(name) struct name : packet_base<name>
```

전체 패킷은 아래와 같다

```
/* Client 2 Server */

, Cs_none = 10
, Cs_login
, Cs_signup
, Cs_input
, Cs_input_timestamp
, Cs_chat
, Cs_request_name
, Cs_request_hp
, Cs_request_money
, Cs_request_exp
, Cs_request_level
, Cs_use_skill
, Cs_use_item
, Cs_invite_to_party
, Cs_leave_party
, Cs_accept_party_invite
, Cs_request_AttackPoint
, Cs_request_ArmorPoint
, Cs_request_AdditionalHp
, Cs_request_MovemetSpeed
```

```
/* Server 2 Client */

, Sc_none = 100
, Sc_login_result
, Sc_ready
, Sc_set_position
, Sc_set_position_timestamp
, Sc_remove_obj
, Sc_set_name
, Sc_set_hp
, Sc_set_money
, Sc_set_exp
, Sc_set_level
, Sc_chat
, Sc_signup_result
, Sc_use_skill
, Sc_set_iteminstance_position
, Sc_remove_iteminstance
, Sc_sum_item
, Sc_equip_item
, Sc_insert_partycrew
, Sc_erase_partycrew
, Sc_join_party_request
, Sc_set_attack_point
, Sc_set_armor_point
, Sc_set_additional_hp
, Sc_set_movement_speed
```

이벤트

이벤트효과를 담은 람다객체와 이벤트 실행시간을 가지고 있는 Event 객체를 우선순위큐에 담아 꺼낸다.

```
struct Event
{
    Event(function<void()> f, milliseconds ms)
        : EventFunc{ f }, ActionTime{ clk::now() + ms }{}
    function<void()> EventFunc;
    clk::time_point ActionTime;
};

constexpr bool operator<(const Event& l, const Event& r) { return l.ActionTime < r.ActionTime; }
constexpr bool operator>(const Event& l, const Event& r) { return l.ActionTime > r.ActionTime; }

class EventManager
{
    SINGLE_TON(EventManager) = default;
public:
    void AddEvent(const Event& e) { EventQueue_.push(e); }
    void ProcessEventQueueLoop();
private:
    concurrent_priority_queue<Event, greater<Event>> EventQueue_{ 5000000 };
};
```

꺼내어진 이벤트객체에서 함수객체를 복사하여 expovelapped 를 생성, 이를 iocp 처리 로직으로 보내 처리한다.

```
struct EventExpOverlapped : ExpOverlappedBasic
{
    EventExpOverlapped(function<void()> e) : Ex
    function<void()> EventFunc;
};
```

```
void Server::OnEventTimerComplete(ExpOverlapped* exover)
{
    auto e = reinterpret_cast<EventExpOverlapped*>(exover);
    e->EventFunc();
    delete exover;
}
```

사용법은 아래와 같다.

Boost::Asio 처럼 람다객체를 통해 코드가 깔끔해지는 것을 노렸다.

```
EventManager::Get().AddEvent(
    { [id = p->GetId(), L = Scripts_[eScriptType::Upd
    {
        lock_guard lck{ mutex };
        lua_getglobal(L, "EventPlayerEnterSight");
        lua_pushnumber(L, id);
        lua_pcall(L, 1, 0, 0);
    }, 0s });
```

```
EventManager::Get().AddEvent({ [this]() { Update(); }, 500ms });
```

데이터 베이스

이벤트매니저 클래스와 비슷한 구조로 만들었다.

쿼리요청객체는 쿼리문, select 된 정보를 받는 Targets, Targets 를 인자로 받는 종료함수를 가지고 있다.

```
struct QueryRequest
{
    wstring Query;
    function<void(vector<any>)> Func;
    shared_ptr<vector<any>> Targets;
};
```

이벤트 처리와 같은 방식으로, 쿼리문을 실행하고 select 된 정보들과 종료함수를 iocp 처리 로직에 전달한다.

```
struct DataBaseExpOverlapped : ExpOverlappedBasic
{
    DataBaseExpOverlapped(function<void(vector<any>)> f, vector<any>& R)
        : ExpOverlappedBasic{ COMP_OP::OP_DB_EVENT }, Func{ f } { Results = move(R); }
    vector<any> Results;
    function<void(vector<any>)> Func;
};
```

```cpp
void Server::OnDataBaseQueryComplete(ExpOverlapped* exover)
{
    auto e = reinterpret_cast<DataBaseExpOverlapped*>(exover);
    e->Func(e->Results);
    delete exover;
}
```

사용법은 아래와 같다. 역시 이벤트객체와 비슷하다.

```cpp
QueryRequest q;
q.Query = L"EXEC SelectItemDataById "s + to_wstring(dbId);
q.Targets = make_shared<vector<any>>(); q.Targets->reserve(3);
q.Targets->emplace_back(make_any<wstring>());    // name
q.Targets->emplace_back(make_any<SQLINTEGER>()); // num
q.Targets->emplace_back(make_any<BOOL>()); // used
q.Func = [id](const vector<any>& t)
{
    auto itemName = any_cast<wstring>(t[0]);
    auto itemNum = any_cast<SQLINTEGER>(t[1]);
    auto itemUsed = any_cast<BOOL>(t[2]);
    { ... }
};
DataBase::Get().AddQueryRequest(q);
```

셀렉트문으로 어떤 타입의 값이 오더라도 상관없이 어느정도 일관되게 사용할 수 있는 코드를 짜기 위해서, vector<any>를 처리함수의 인자로 받도록 만들었다.

월드

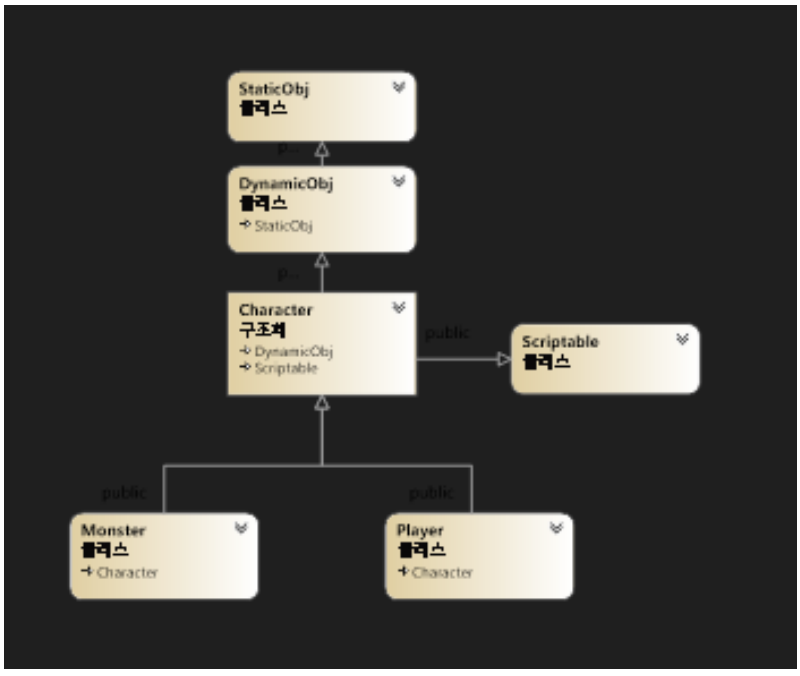
```cpp
class World
{
    SINGLE_TON(World) = default;
public:
    void ChangeSector(StaticObj* obj, Position newSector);
    Sector& GetSector(Position sector) { return Sectors_[sector.y][sector.x]; }
    array<Sector*, 4> GetNearSectors4(Position pos);
    array<Sector*, 4> GetNearSectors4(Position pos, Position sector);
    array<Sector*, 9> GetNearSectors9(Position pos, Position sector);
public:

private:
    Sector Sectors_[SECTOR_NUM][SECTOR_NUM];
};
```

```cpp
class Sector
{
    friend class World;
public:
    void Update();
public:
    void EraseObjFromSector(StaticObj* obj);
    void InsertObjSector(StaticObj* obj);
    GET_REF(Players);
    GET_REF(Monsters);
    GET_REF(Obstacles);
    GET_REF(Npcs);
    GET_REF_UNSAFE(ItemInstances);
public:
    shared_mutex PlayerLock;
    shared_mutex MonsterLock;
    shared_mutex ItemInstanceLock;
private:
    vector<DynamicObj*> Obstacles_;                 // 고정데이터..
    concurrent_unordered_set<ItemInstance> ItemInstances_;
    concurrent_unordered_set<Monster*> Monsters_;
    concurrent_unordered_set<DynamicObj*> Npcs_;
    concurrent_unordered_set<Player*> Players_;
};
```

월드맵을 섹터로 구분하고, 각 섹터에 오브젝트리스트를 저장, 상호작용이 필요할 경우 주위 4 개 섹터 내에서만 상호작용 검사를 하도록 하여 성능을 개선했다.
섹터를 변경할 때, read write 락을 써서 데이터 레이스를 없앴다.

캐릭터

```
class CharacterManager
{
    SINGLE_TON(CharacterManager);
public:
    bool Move(ID Id_, eMoveOper oper);
    bool Move(ID Id_, Position to);
    bool MoveForce(ID Id_, Position to);
    bool InitialMove(ID Id_, Position to);
    Position GetPosition(ID Id_) { return Characters_[Id_]->GetPos(); }
    void SetPosition(ID Id_, Position pos) { Characters_[Id_]->SetPos(pos); }
    void Enable(ID Id_) { Characters_[Id_]->Enable(); }
    void Disable(ID Id_) { Characters_[Id_]->Disable(); }
    void InitFromDataBase(ID id, DbCharacterID DbId);
    void ActivateSkill(ID id, eSkill skill);
    GET_REF_UNSAFE(Characters);
protected:
private:
    // PLAYER       [ 0, MAX_PLAYER )
    // MONSTER  [ MAX_PLAYER, MAX_PLAYER + MAX_MONSTER )
    // NPC      [ MAX_PLAYER + MAX_MONSTER, MAX_CHARACTER )
    array<unique_ptr<Character>, MAX_CHARACTER> Characters_;
    array<unique_ptr<DynamicObj>, MAX_OBSTACLE> Obstacles_;
};
```



몬스터 스크립트는 상태기계로 구현, 몬스터는 Movement / Aggressive 타입을 가지고 있음. 고정 몬스터 / 로밍몬스터 그리고 선공몬스터 / 평화몬스터 의 구분으로 총 4 개의 조합이 있고, 각 타입별로 스크립트를 만들었음. Movement 스크립트에서 Move 함수 구현을 결정, Aggressive 스크립트에서 시야내 플레이어 등장 이벤트 구현과 데미지계수를 달리 하였음

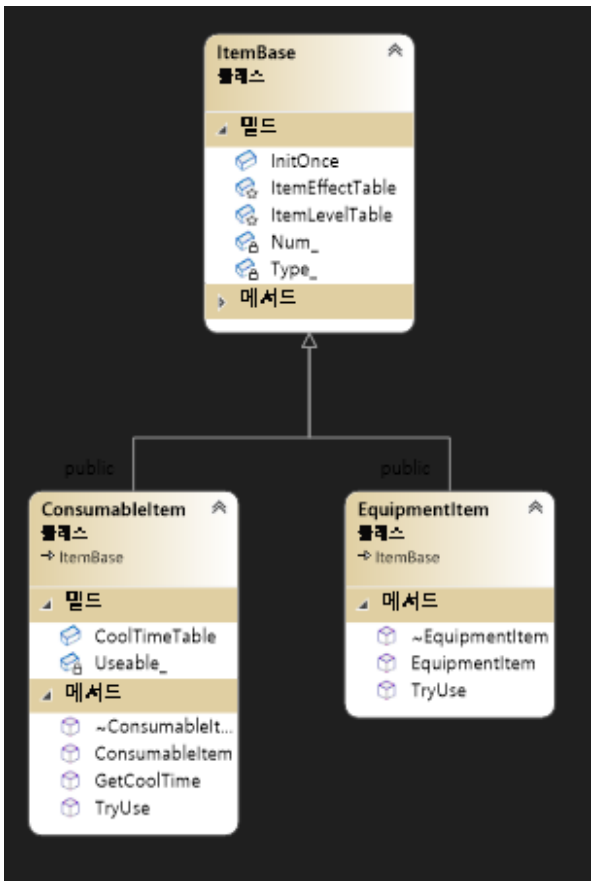
```
void Monster::CompileScript()
{
    lock_guard lck{ ScriptLock[eScriptType::UpdateAI] };
    lua_State* aiScript;
    aiScript = luaL_newstate();
    luaL_openlibs(aiScript);
    {
        string str; str.reserve(2500);
        ifstream basicGlobalDeclaration{ "LuaScript/Monster/monsterBasicDeclaration.lua" , ios::binary };
        ifstream movementRomaing{ "LuaScript/Monster/monsterMovementRoaming.lua" , ios::binary };
        ifstream movementFixed{ "LuaScript/Monster/monsterMovementFixed.lua" , ios::binary };
        ifstream aggressionAgro{ "LuaScript/Monster/monsterAggressionAgro.lua" , ios::binary };
        ifstream aggressionPeace{ "LuaScript/Monster/monsterAggressionPeace.lua" , ios::binary };

        str += string{ (std::istreambuf_iterator<char>(basicGlobalDeclaration)), std::istreambuf_iterator<char>() };

        switch (MovementType_) { ... }

        switch (AggressionType_) { ... }

        luaL_loadstring(aiScript, str.c_str());
        lua_pcall(aiScript, 0, 0, 0);
    }
    { ... }
    { ... }
    Scripts_[eScriptType::UpdateAI] = aiScript;
}
```

```
myId = nil
targetId = nil
state = nil


function SetObjectId(id)
    myId = id
end

function Update()
    state()
end

function Idle()
    Move()
end

function Battle()
    Attack()
end

state = Idle
```

아이템

타입과 누적개수를 멤버로 가지는 아이템 베이스.

```
class ItemBase
{
    inline static unordered_map<eItemType, function<void(ID)>> ItemEffectTable;
    inline static unordered_map<eItemType, int> ItemLevelTable;
    eItemType Type_{ eItemType::hpPotion };
    mutable atomic_int Num_{};
};
```



이를 소비아이템과 장비아이템으로 파생하였음.

TryUse 의 구현에서 소비아이템은 누적개수가 줄고, 장비아이템은 줄지 않는 것이 큰 차이점.

각 아이템 타입에 해당하는 효과테이블을 만들었음. TryUse 함수에서는 이 테이블에서 람다객체를 가져와 실행함.

```
ItemEffectTable[eItemType::hpPotion] = [](ID agent)
{
    auto& characters = CharacterManager::Get().GetCharacters();
    characters[agent]->HpIncrease(agent, 150);
};

ItemEffectTable[eItemType::expPotion] = [](ID agent)
{
    auto& characters = CharacterManager::Get().GetCharacters();
    if (auto p = dynamic_cast<Player*>(characters[agent].get()))
    {
        p->ExpSum(agent, 200);
    }
};

{
    auto b = static_cast<int>(eItemType::_EquipmentItemStartLine);
    auto e = static_cast<int>(eItemType::_EquipmentItemEndLine);
    for (auto i = b; i != e; i++) {
        auto type = static_cast<eItemType>(i);
        ItemEffectTable[type] = [type](ID agent)
        {
            auto& characters = CharacterManager::Get().GetCharacters();
            if (auto p = dynamic_cast<Player*>(characters[agent].get()))
            {
                p->Equip(type);
            }
        };
    }
}
```

Player->Equip(type) 함수에서는 타입을 토대로 어느 파트에 장비될 아이템인지 구분하여 각 파트에 매칭되는 스탯을 아이템레벨을 기반으로 조정함

4 개의 장착부위가 있음.

Head 방어력

Body 최대체력

Shoes 이동속도

Weapon 공격력

위와 같은 스탯에 관여함

```cpp
bool Player::Equip(eItemType item)
{
    auto itemLevel = ItemBase::GetItemLevel(item);

    switch (EquimentState_.Equip(item))
    {
    case eEquimentablePart::_Unable:
    { ... }
    CASE eEquimentablePart::head :
    {
        ArmorPoint_ = itemLevel * 2;

        sc_set_armor_point set;
        set.id = Id_;
        set.armorPoint = ArmorPoint_;
        Server::Get().GetClients()[Id_].DoSend(&set);
    }
    CASE eEquimentablePart::body :
    {
        AdditionalHp_ = itemLevel * 100;

        sc_set_additional_hp set;
        set.id = Id_;
        set.additionalHp = AdditionalHp_;

        auto party = PartyManager::Get().GetParty(PartyId_);
        if (party) { ... }
        else Server::Get().GetClients()[Id_].DoSend(&set);
```

열 이름	데이터 형식	Null 허용
ItemName	nchar(20)	☐
OwnnerID	int	☐
ItemCount	int	☐
Used	bit	☐

아이템은 데이터 베이스에 위와 같이 기록됨.

사용되었는지 여부는 장비아이템의 장착여부기록임

파티

파티원은 최대 4 명까지.

```
class Party
{
    friend class PartyManager;
public:
    Party() { for (auto& pc : PartyCrews_) pc = -1; }
    GET_REF(PartyCrews);
protected:
    array<atomic<ID>, MAX_PARTY_CREW> PartyCrews_;
    atomic_bool Empty{ true };
private:
};

class PartyManager
{
    SINGLE_TON(PartyManager) = default;
public:
    ID CreateParty(ID characterId);
    const Party* GetParty(ID partyId);
    bool JoinParty(ID partyId, ID characterId);
    bool ExitParty(ID partyId, ID characterId);
private:
    array<Party, MAX_PARTY> Partys_;
};
```

플레이어는 PartyIdx 를 멤버로 가짐.

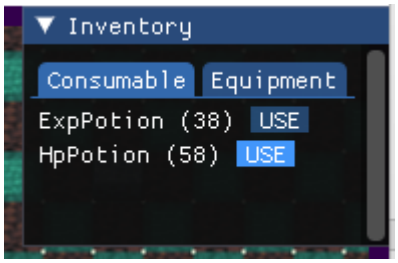
체력 변화, 레벨 변화 시에 파티원들에게 전송

파티사냥시 절반으로 줄어든 사냥 경험치를 공유함

시야 밖으로 벗어나더라도 파티원의 정보는 계속 업데이트됨

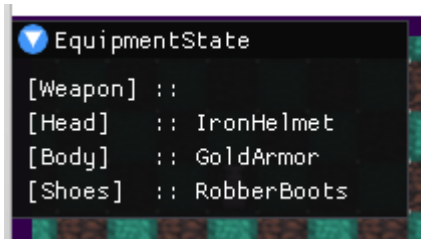
클라이언트


EquipmentState
[Weapon] ::
[Head] :: IronHelmet
[Body] :: GoldArmor
[Shoes] :: RobberBoots
▼ Status
AttackPoint :: 0
ArmorPoint :: 2
HP / MAXHP :: 1090/1100
MovementSpeed :: 2
▼ Inventory
IronHelmet (13) EQUIP
StoneSword (1) EQUIP
WingShoes (1) EQUIP
GoldArmor (3) EQUIP
IronSword (4) EQUIP
smash!
grr..
grr..
ID :: test
HP :: 1090/1100
LEVEL :: 6 EXP :: 396/3200
Money :: 9290
[19:30:29] test :: hix
[19:30:42] test :: p
ENTER



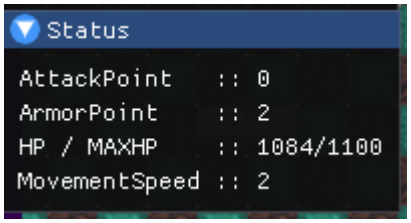
우상단 아이템창 (I 키)

소비템과 장비템 탭이 구분되어있음, 버튼을 눌러 사용하거나 장착가능.



좌상단 장비템창 (E 키)

현재 장착된 장비아이템을 보여줌

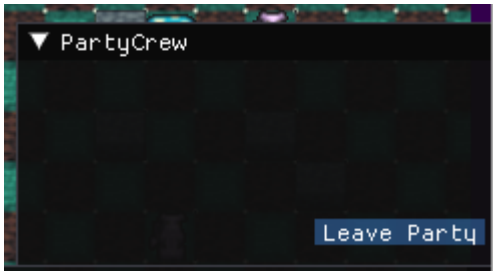


좌상단 스텟창 (S 키)

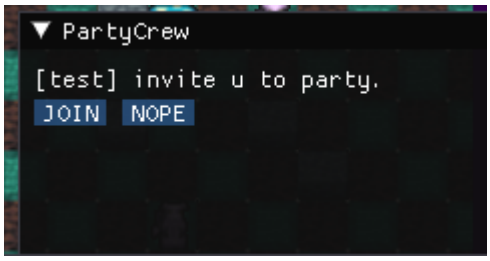
각 장비부위에 해당하는 스탯을 보여줌

하단 정보창, 채팅창

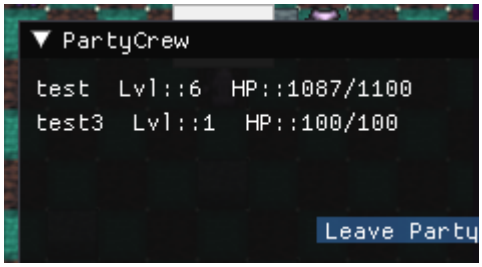
엔터를 누르면 채팅을 칠 수 있음. 시야 내 플레이어들에게 전달됨



파티가 없는 상태



파티에 초대받은 상태



파티에 참여된 상태

좌하단 파티창 (P 키)

파티초대 메시지가 오거나, 파티원의 정보를 보여줌. 파티 떠나기 버튼이 있음

플레이어 선택

몬스터 선택

캐릭터 상태창 (마우스 우클릭으로 캐릭터 선택)

선택된 캐릭터(플레이어, 몬스터, 장애물)의 정보를 보여줌

플레이어는 파티초대 버튼이 있음

기본공격 (A 키)

4 방향 공격 쿨타임 1 초

스킬공격 (X 키)

마름모꼴 범위공격 쿨타임 5 초

버프스킬 (C 키)

기본공격속도 증가 지속 8 초 쿨타임 15 초

아이템줍기 (Z 키)

[몬스터]

Agro / Roamer

Agro / Unmoveable

Peace / Roamer

Peace / Unmoveable

[플레이어 및 장애물]

플레이어

다른플레이어

장애물

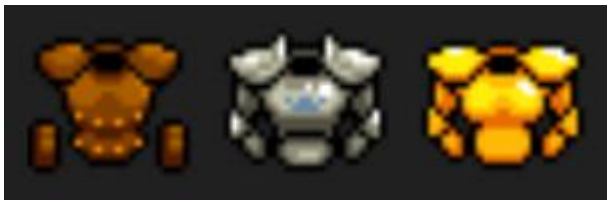
[소비아이템]

체력회복 경험치획득

[장비아이템]

머리

몸통



신발

무기

[플레이 방법]

1. 클라이언트 실행
2. 서버 IP 입력
3. L 또는 S 입력으로 로그인 혹은 회원가입
4. 게임을 플레이 (목적 없음)