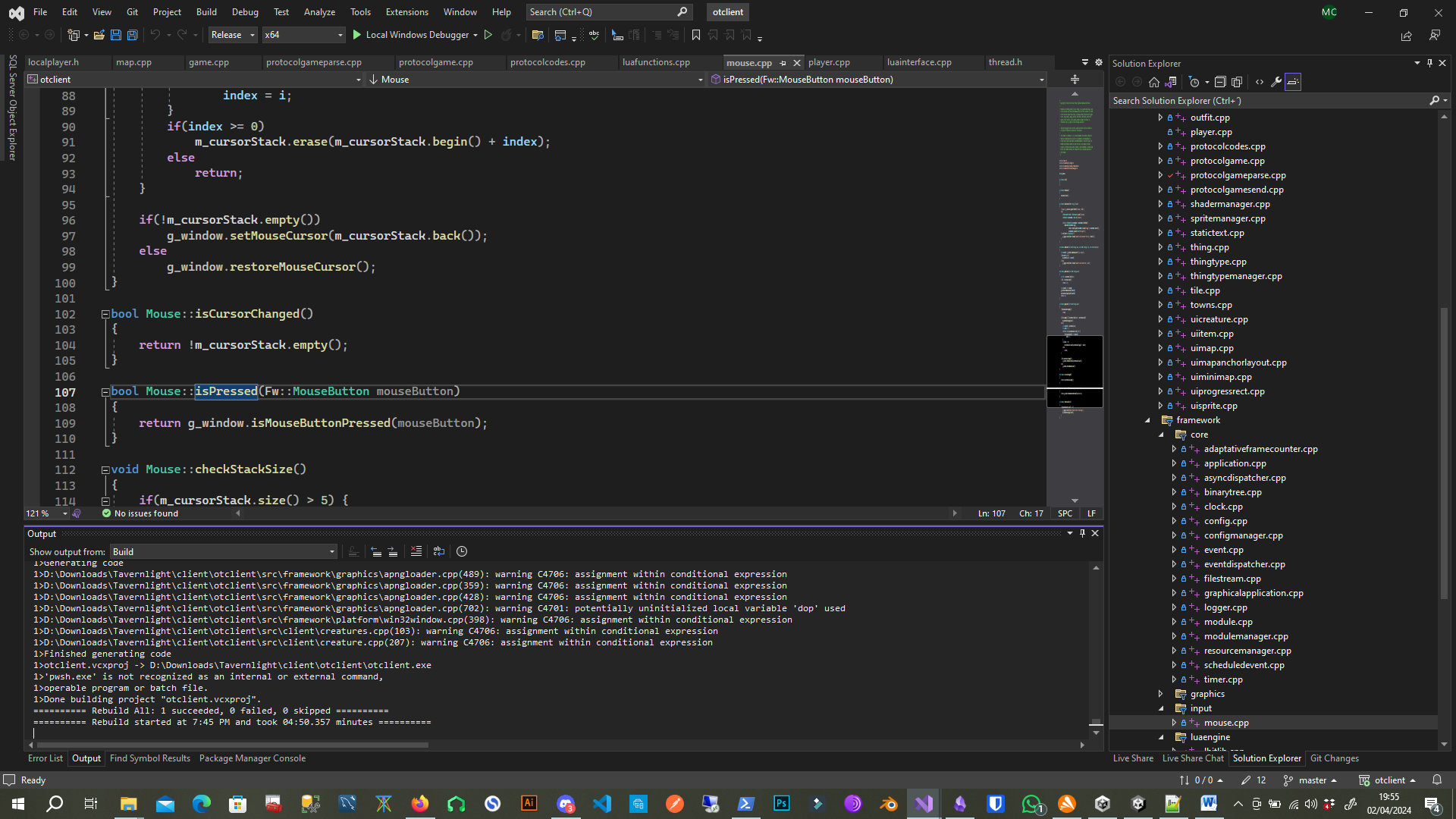
I was able to compile the client and server, but the main branch version of the server wasn’t compatible with the protocol version of the client.

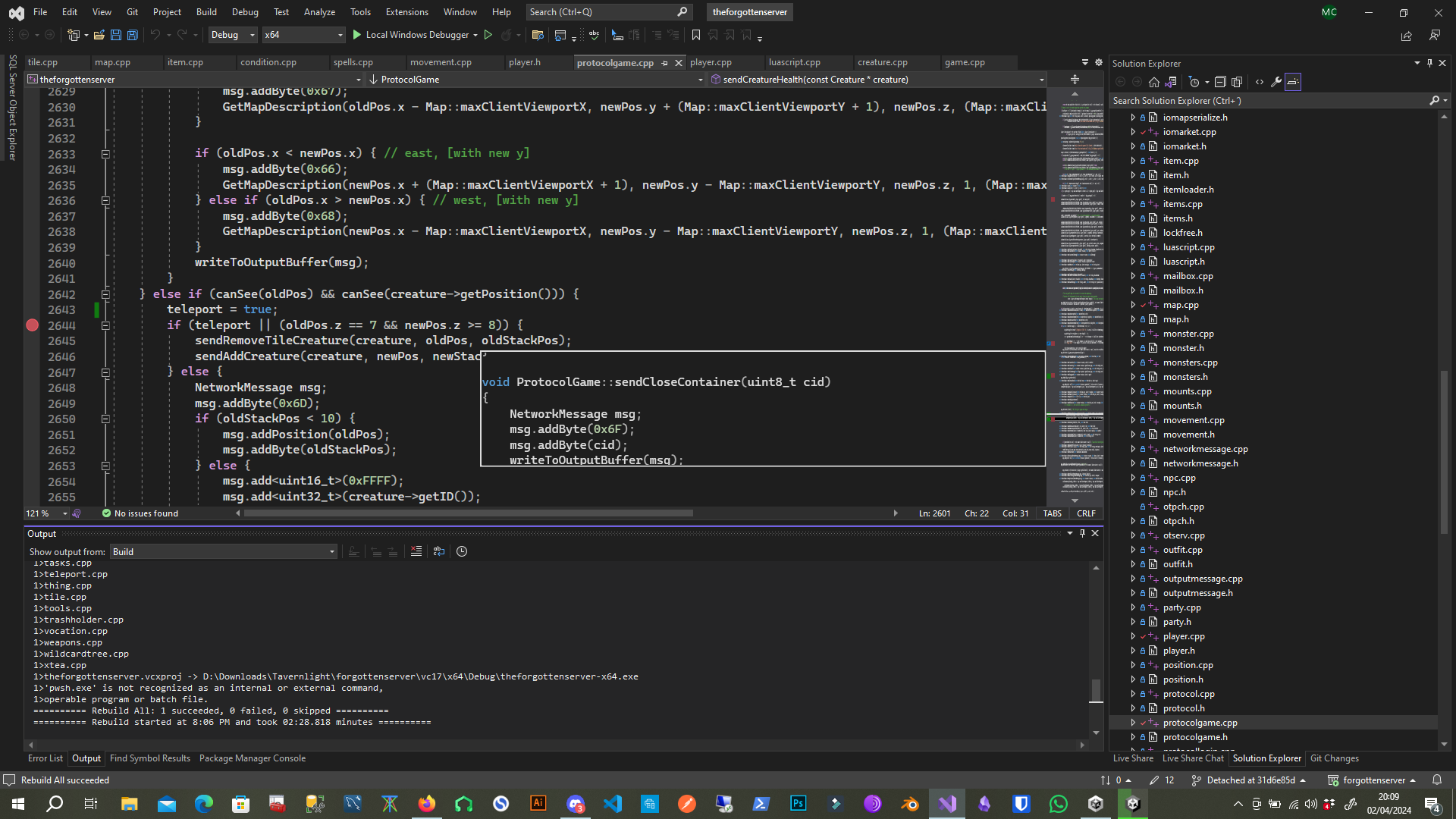


I got the tag v1.4.2, which was compatible with the same client protocol (1098), but I had to fix several bugs to make it work. After that, I didn’t have much more time since I work full-time.

**Client compilation**



**The Forgotten Server tag V1.4.2**



I can provide a video of the client server working if needed.

**CLIENT SIDE CODE**

***gameinterface.lua***

function processMouseAction(menuPosition, mouseButton, autoWalkPos, lookThing, useThing, creatureThing, attackCreature)

  local keyboardModifiers = g\_keyboard.getModifiers()

  if not modules.client\_options.getOption('classicControl') then

    if keyboardModifiers == KeyboardNoModifier and mouseButton == MouseRightButton then

      createThingMenu(menuPosition, lookThing, useThing, creatureThing)

      return true

    elseif lookThing and keyboardModifiers == KeyboardShiftModifier and (mouseButton == MouseLeftButton or mouseButton == MouseRightButton) then

      g\_game.look(lookThing)

      return true

    elseif useThing and keyboardModifiers == KeyboardCtrlModifier and (mouseButton == MouseLeftButton or mouseButton == MouseRightButton) then

      if useThing:isContainer() then

        if useThing:getParentContainer() then

          g\_game.open(useThing, useThing:getParentContainer())

        else

          g\_game.open(useThing)

        end

        return true

      elseif useThing:isMultiUse() then

        startUseWith(useThing)

        return true

      else

        g\_game.use(useThing)

        return true

      end

      return true

    elseif attackCreature and g\_keyboard.isAltPressed() and (mouseButton == MouseLeftButton or mouseButton == MouseRightButton) then

      g\_game.attack(attackCreature)

      return true

    elseif creatureThing and creatureThing:getPosition().z == autoWalkPos.z and g\_keyboard.isAltPressed() and (mouseButton == MouseLeftButton or mouseButton == MouseRightButton) then

      g\_game.attack(creatureThing)

      return true

    end

  -- classic control

  else

    if useThing and keyboardModifiers == KeyboardNoModifier and mouseButton == MouseRightButton and not g\_mouse.isPressed(MouseLeftButton) then

      local player = g\_game.getLocalPlayer()

      if attackCreature and attackCreature ~= player then

        g\_game.attack(attackCreature)

        return true

      elseif creatureThing and creatureThing ~= player and creatureThing:getPosition().z == autoWalkPos.z then

        g\_game.attack(creatureThing)

        return true

      elseif useThing:isContainer() then

        if useThing:getParentContainer() then

          g\_game.open(useThing, useThing:getParentContainer())

          return true

        else

          g\_game.open(useThing)

          return true

        end

      elseif useThing:isMultiUse() then

        startUseWith(useThing)

        return true

      else

        g\_game.use(useThing)

        return true

      end

      return true

    elseif lookThing and keyboardModifiers == KeyboardShiftModifier and (mouseButton == MouseLeftButton or mouseButton == MouseRightButton) then

      g\_game.look(lookThing)

      return true

    elseif lookThing and ((g\_mouse.isPressed(MouseLeftButton) and mouseButton == MouseRightButton) or (g\_mouse.isPressed(MouseRightButton) and mouseButton == MouseLeftButton)) then

      g\_game.look(lookThing)

      return true

    elseif useThing and keyboardModifiers == KeyboardCtrlModifier and (mouseButton == MouseLeftButton or mouseButton == MouseRightButton) then

      createThingMenu(menuPosition, lookThing, useThing, creatureThing)

      return true

    elseif attackCreature and g\_keyboard.isAltPressed() and (mouseButton == MouseLeftButton or mouseButton == MouseRightButton) then

      g\_game.attack(attackCreature)

      return true

    elseif creatureThing and creatureThing:getPosition().z == autoWalkPos.z and g\_keyboard.isAltPressed() and (mouseButton == MouseLeftButton or mouseButton == MouseRightButton) then

      g\_game.attack(creatureThing)

      return true

    end

  end

  local player = g\_game.getLocalPlayer()

  player:stopAutoWalk()

  if autoWalkPos and keyboardModifiers == KeyboardNoModifier and mouseButton == MouseLeftButton then

    -- player:autoWalk(autoWalkPos)

**g\_game.teleport(autoWalkPos)**

    return true

  end

  return false

end

I commented the autoWalk and created a new function called teleport.

-- player:autoWalk(autoWalkPos)

**g\_game.teleport(autoWalkPos)**

This will call the teleport function I created in the game.cpp and was registered among other lua functions in luafunctions.cpp

***game.cpp***

void Game::teleport(const Position& pos)

{

//since this method is called every frame only sends my custom teleport packet when the player moves to a valid position

if(m\_localPlayer->getPosition() != pos && pos.isValid())

m\_protocolGame->sendTeleporth(pos);

}

This function will send the position where the player clicked to teleport the avatar.

***protocolgame.cpp***

void ProtocolGame::sendTeleporth(const Position& pos)

{

OutputMessagePtr msg(new OutputMessage);

msg->addU8(Proto::ClientTeleport);

addPosition(msg, pos);

send(msg);

}

This function will send the position to the server.

msg->addU8(Proto::ClientTeleport);

The line above adds the opcode to the message.

addPosition(msg, pos);

This method to add a position to a msg already existed I only used it instead of reinventing the wheel.

***protocolcodes.h***

// NOTE: add any custom opcodes in this range

// 51 - 99

ClientTeleport = 99,

Created my custom opcode with the decimal 99 but in the server side it’s the hexadecimal number 0x63

**SERVER SIDE CODE**

***protocolgame.cpp***

void ProtocolGame::parsePacket(NetworkMessage& msg)

{

if (!acceptPackets || g\_game.getGameState() == GAME\_STATE\_SHUTDOWN || msg.getLength() == 0) {

return;

}

uint8\_t recvbyte = msg.getByte();

if (!player) {

if (recvbyte == 0x0F) {

disconnect();

}

return;

}

//a dead player can not performs actions

if (player->isRemoved() || player->getHealth() <= 0) {

if (recvbyte == 0x0F) {

disconnect();

return;

}

if (recvbyte != 0x14) {

return;

}

}

switch (recvbyte) {

case 0x14: g\_dispatcher.addTask(createTask(std::bind(&ProtocolGame::logout, getThis(), true, false))); break;

case 0x1D: addGameTask(&Game::playerReceivePingBack, player->getID()); break;

case 0x1E: addGameTask(&Game::playerReceivePing, player->getID()); break;

case 0x32: parseExtendedOpcode(msg); break; //otclient extended opcode

case 0x63: parseTeleport(msg); break;//My custom code

case 0x64: parseAutoWalk(msg); break;

case 0x65: addGameTask(&Game::playerMove, player->getID(), DIRECTION\_NORTH); break;

case 0x66: addGameTask(&Game::playerMove, player->getID(), DIRECTION\_EAST); break;

case 0x67: addGameTask(&Game::playerMove, player->getID(), DIRECTION\_SOUTH); break;

case 0x68: addGameTask(&Game::playerMove, player->getID(), DIRECTION\_WEST); break;

case 0x69: addGameTask(&Game::playerStopAutoWalk, player->getID()); break;

case 0x6A: addGameTask(&Game::playerMove, player->getID(), DIRECTION\_NORTHEAST); break;

case 0x6B: addGameTask(&Game::playerMove, player->getID(), DIRECTION\_SOUTHEAST); break;

case 0x6C: addGameTask(&Game::playerMove, player->getID(), DIRECTION\_SOUTHWEST); break;

case 0x6D: addGameTask(&Game::playerMove, player->getID(), DIRECTION\_NORTHWEST); break;

case 0x6F: addGameTaskTimed(DISPATCHER\_TASK\_EXPIRATION, &Game::playerTurn, player->getID(), DIRECTION\_NORTH); break;

case 0x70: addGameTaskTimed(DISPATCHER\_TASK\_EXPIRATION, &Game::playerTurn, player->getID(), DIRECTION\_EAST); break;

case 0x71: addGameTaskTimed(DISPATCHER\_TASK\_EXPIRATION, &Game::playerTurn, player->getID(), DIRECTION\_SOUTH); break;

case 0x72: addGameTaskTimed(DISPATCHER\_TASK\_EXPIRATION, &Game::playerTurn, player->getID(), DIRECTION\_WEST); break;

case 0x77: parseEquipObject(msg); break;

case 0x78: parseThrow(msg); break;

case 0x79: parseLookInShop(msg); break;

case 0x7A: parsePlayerPurchase(msg); break;

case 0x7B: parsePlayerSale(msg); break;

case 0x7C: addGameTask(&Game::playerCloseShop, player->getID()); break;

case 0x7D: parseRequestTrade(msg); break;

case 0x7E: parseLookInTrade(msg); break;

case 0x7F: addGameTask(&Game::playerAcceptTrade, player->getID()); break;

case 0x80: addGameTask(&Game::playerCloseTrade, player->getID()); break;

case 0x82: parseUseItem(msg); break;

case 0x83: parseUseItemEx(msg); break;

case 0x84: parseUseWithCreature(msg); break;

case 0x85: parseRotateItem(msg); break;

case 0x87: parseCloseContainer(msg); break;

case 0x88: parseUpArrowContainer(msg); break;

case 0x89: parseTextWindow(msg); break;

case 0x8A: parseHouseWindow(msg); break;

case 0x8B: parseWrapItem(msg); break;

case 0x8C: parseLookAt(msg); break;

case 0x8D: parseLookInBattleList(msg); break;

case 0x8E: /\* join aggression \*/ break;

case 0x96: parseSay(msg); break;

case 0x97: addGameTask(&Game::playerRequestChannels, player->getID()); break;

case 0x98: parseOpenChannel(msg); break;

case 0x99: parseCloseChannel(msg); break;

case 0x9A: parseOpenPrivateChannel(msg); break;

case 0x9E: addGameTask(&Game::playerCloseNpcChannel, player->getID()); break;

case 0xA0: parseFightModes(msg); break;

case 0xA1: parseAttack(msg); break;

case 0xA2: parseFollow(msg); break;

case 0xA3: parseInviteToParty(msg); break;

case 0xA4: parseJoinParty(msg); break;

case 0xA5: parseRevokePartyInvite(msg); break;

case 0xA6: parsePassPartyLeadership(msg); break;

case 0xA7: addGameTask(&Game::playerLeaveParty, player->getID()); break;

case 0xA8: parseEnableSharedPartyExperience(msg); break;

case 0xAA: addGameTask(&Game::playerCreatePrivateChannel, player->getID()); break;

case 0xAB: parseChannelInvite(msg); break;

case 0xAC: parseChannelExclude(msg); break;

case 0xBE: addGameTask(&Game::playerCancelAttackAndFollow, player->getID()); break;

case 0xC9: /\* update tile \*/ break;

case 0xCA: parseUpdateContainer(msg); break;

case 0xCB: parseBrowseField(msg); break;

case 0xCC: parseSeekInContainer(msg); break;

case 0xD2: addGameTask(&Game::playerRequestOutfit, player->getID()); break;

case 0xD3: parseSetOutfit(msg); break;

case 0xD4: parseToggleMount(msg); break;

case 0xDC: parseAddVip(msg); break;

case 0xDD: parseRemoveVip(msg); break;

case 0xDE: parseEditVip(msg); break;

case 0xE6: parseBugReport(msg); break;

case 0xE7: /\* thank you \*/ break;

case 0xE8: parseDebugAssert(msg); break;

case 0xF0: addGameTaskTimed(DISPATCHER\_TASK\_EXPIRATION, &Game::playerShowQuestLog, player->getID()); break;

case 0xF1: parseQuestLine(msg); break;

case 0xF2: parseRuleViolationReport(msg); break;

case 0xF3: /\* get object info \*/ break;

case 0xF4: parseMarketLeave(); break;

case 0xF5: parseMarketBrowse(msg); break;

case 0xF6: parseMarketCreateOffer(msg); break;

case 0xF7: parseMarketCancelOffer(msg); break;

case 0xF8: parseMarketAcceptOffer(msg); break;

case 0xF9: parseModalWindowAnswer(msg); break;

default:

// std::cout << "Player: " << player->getName() << " sent an unknown packet header: 0x" << std::hex << static\_cast<uint16\_t>(recvbyte) << std::dec << "!" << std::endl;

break;

}

if (msg.isOverrun()) {

disconnect();

}

}

I added the following line to ParsePacket

case 0x63: parseTeleport(msg); break;//My custom code

In this line the server will parse the teleport location sent by the client, 0x63 is the decimal 99.

void ProtocolGame::parseTeleport(NetworkMessage& msg)

{

Position destination = msg.getPosition();

addGameTask(&Game::teleport, player->getID(), destination, false, 0);

}

parseTeleport will get the position from the message and then calls a task with the teleport parameters to process the teleport server side since it’s an authoritative server.

***game.cpp***

//My custom code

void Game::teleport(uint32\_t playerId, const Position& newPos, bool pushMove, uint32\_t flags)

{

Player\* thing = getPlayerByID(playerId);

if (!thing) {

return;

}

if (newPos == thing->getPosition()) {

return;

}

else if (thing->isRemoved()) {

return;

}

Tile\* toTile = map.getTile(newPos);

if (!toTile) {

return;

}

std::vector<Position> points = pointsBetween(thing->getPosition(), newPos);

//This code didn't work to create the speed effect

for (const auto& point : points) {

map.instantiateCreature(thing, toTile, false);

}

//change player direction

Direction dir = getDirectionTo(thing->getPosition(), newPos);

if ((dir & DIRECTION\_DIAGONAL\_MASK) == 0) {

internalCreatureTurn(thing, dir);

}

if (Creature\* creature = thing->getCreature()) {

ReturnValue ret = toTile->queryAdd(0, \*creature, 1, FLAG\_NOLIMIT);

if (ret != RETURNVALUE\_NOERROR) {

return;

}

map.moveCreature(\*creature, \*toTile, !pushMove);

return;

}

}

The following code gets makes some validations to ensure that all parameters are fine.

Player\* thing = getPlayerByID(playerId);

if (!thing) {

return;

}

if (newPos == thing->getPosition()) {

return;

}

else if (thing->isRemoved()) {

return;

}

Tile\* toTile = map.getTile(newPos);

if (!toTile) {

return;

}

* Evaluates if the player exists
* If the new position is the same as the player position, if it is there is no need to teleport
* Verifies if the player was already removed
* Verifies if the player’s tile exists

The method pointsBetween was created to create the speed effect and get a position interpolation between the current player’s position and the teleport location and instantiate the player’s avatar in the interpolated positions starting in the origin and ending in the destination, adding more transparency to the sprites near the origin. The idea was when the player arrived to the teleport destination point to fire an event and start destroying the sprite near the origin point all the way to the destination. The best way to accomplish this effect is probably to use a shader but I never coded a shader before and I didn’t have the necessary time to learn it. The not efficient code I made to create the effect didn’t work either.

std::vector<Position> points = pointsBetween(thing->getPosition(), newPos);

//My custom code - get several coordinates between 2 points

std::vector<Position> Game::pointsBetween(const Position& a, const Position& b) {

std::vector<Position> points;

// Calculate the distance between points

uint16\_t dx = b.x - a.x;

uint16\_t dy = b.y - a.y;

double distance = std::sqrt(dx \* dx + dy \* dy);

// Normalize the distance vector to have unit length

double unitDx = dx / distance;

double unitDy = dy / distance;

// Iterate from point A to point B with a step of 1 tile distance

for (double t = 0; t <= distance; t += 1.0) {

uint16\_t x = std::round(a.x + t \* unitDx);

uint16\_t y = std::round(a.y + t \* unitDy);

points.push\_back({ x, y, a.z }); // Use the z coordinate from point A

}

return points;

}

The following code would instantiate the player sprite in the interpolated points.

//This code didn't work to create the speed effect

for (const auto& point : points) {

map.instantiateCreature(thing, toTile, false);

}

***map.cpp***

//My custom code - disn't work to create the speed effect

void Map::instantiateCreature(Creature\* creature, Tile\* newTile, bool forceTeleport/\* = false\*/)

{

Tile\* oldTile = creature->getTile();

Position oldPos = oldTile->getPosition();

Position newPos = newTile->getPosition();

QTreeLeafNode\* new\_leaf = getQTNode(newPos.x, newPos.y);

if (oldPos.y > newPos.y) {

creature->setDirection(DIRECTION\_NORTH);

}

else if (oldPos.y < newPos.y) {

creature->setDirection(DIRECTION\_SOUTH);

}

if (oldPos.x < newPos.x) {

creature->setDirection(DIRECTION\_EAST);

}

else if (oldPos.x > newPos.x) {

creature->setDirection(DIRECTION\_WEST);

}

// Switch the node ownership

new\_leaf->addCreature(creature);

//add the creature

newTile->addThing(creature);

newTile->postAddNotification(creature, oldTile, 0);

}

The code above was supposed to ask the client to create a player graphical clone in each point (this didn’t work either)

//change player direction

Direction dir = getDirectionTo(thing->getPosition(), newPos);

if ((dir & DIRECTION\_DIAGONAL\_MASK) == 0) {

internalCreatureTurn(thing, dir);

}

The code above changes the player direction to face the point where it will be teleported.

if (Creature\* creature = thing->getCreature()) {

ReturnValue ret = toTile->queryAdd(0, \*creature, 1, FLAG\_NOLIMIT);

if (ret != RETURNVALUE\_NOERROR) {

return;

}

map.moveCreature(\*creature, \*toTile, !pushMove);

return;

}

I used moveCreature to teleport the player since it was an already existent feature but that only could be used if the player had a teleport object in the inventory.

In the end I was able to turn the player and teleport it to the mouse clicked position but I was unable to create de speed effect on the client.