client.hpp

```
#pragma once

#include <SFML/Network.hpp>
#include <boost/multiprecision/cpp_int.hpp>
#include <thread>
#include <mutex>

typedef boost::multiprecision::cpp_int bigint;

using boost::multiprecision::powm;

//bool tryLogIn(const string& username, const string& password, string&
error);

bool sendUDP(sf::UdpSocket& socket, const string& message, string&
error);
bool recvUDP(sf::UdpSocket& socket, void*& buffer, int& buffer_size);

void printBytes(const unsigned char* pBytes, const uint32_t nBytes);
void printBytes(void* pBytes, const uint32_t nBytes);


//Encryption
string encryptAES(const std::string& plaintext, unsigned char* key,
string& error);
string decryptAES(const string& ciphertext, unsigned char* key, string&
error);

//string bigintToHexString(const bigint& number);
void bigintToBytes(bigint key, unsigned char* buffer);

class Client
{
private:
    sf::TcpSocket tcp_socket;
    sf::UdpSocket udp_socket;
    sf::IpAddress udp_sender_address, udp_listener_address;
    string ip;
    unsigned short udp_sender_port, udp_listener_port;

    bigint p, g, secret;
    unsigned char key_bytes[16];


public:
    int player_id;
    string username;

    struct PlayerInfo
    {
        enum Flag
        {
            moving = 1,
```

```
            forward = 2,
            gun_shot = 4,
            quit = 8,
            got_shot = 16,
            dead = 32
        };
        int player_id;
        float dist2wall;
        float pos_x, pos_y, rot_x, rot_y;
        int flags;
        int score;
        char username[16];
    };


    Client();


    bool connectToServer(string& error);

    bool tryLogIn(const string& username, const string& password,
    string& error);
    bool trySignUp(const string& username, const string& password,
    string& error);

    bool sendEncryptedTCP(const string& msg, string& error);
    bool recvEncryptedTCP(void*& buffer, int& bufferSize, string&
    error);
    static bool recvTCP(sf::TcpSocket& socket, void*& buffer, int&
    buffer_size);
    static bool sendTCP(sf::TcpSocket& socket, const string& message,
    string& error);

    bool sendEncryptedUDP(void* buffer, int size, string& error);
    bool recvEncryptedUDP(void*& buffer, int& bufferSize, string&
    error);
    bool sendUDP(const string& message, string& error);
    bool recvUDP(void*& buffer, int& buffer_size);
    bool connected = false;
};
```

headers.hpp

```
#ifndef HEADERS_HPP
#define HEADERS_HPP


#include <SFML/Graphics.hpp>
#include <SFML/Network.hpp>


#include <vector>
#include <iostream>
#include <cmath>


typedef sf::Vector2f v2f;
typedef sf::Vector2i v2i;
using std::cout;
using std::vector;
using std::string;

const double PI = 3.14159265358979323846;
const double HALF_PI = 3.14159265358979323846/2;
const float TO_DEGREES = 57.29577955f;

const int WIDTH = 1280, HEIGHT = 720;


#endif // !HEADERS.HPP
```

main.cpp

```
#include "headers.hpp"
#include "tools.hpp"
#include "player.hpp"
#include "map.hpp"
#include "object.hpp"
#include "client.hpp"
#include "toaster.hpp"

#include <chrono>

void loginPage(sf::RenderWindow& window, Player& player, Toaster&
toaster);
void mainLoop(sf::RenderWindow& window, Player& player, Toaster&
toaster);



int main()
{

    //cout << "Start.\n";

    //sf::UdpSocket udp1, udp2;
    //if (udp1.bind(sf::Socket::AnyPort) != sf::Socket::Done)
    //    cout << "Problem Binding 1\n";

    //if (udp2.bind(sf::Socket::AnyPort) != sf::Socket::Done)
    //    cout << "Problem Binding 2\n";


    //cout << udp2.getLocalPort() << "\n";

    //string message = "Penis";

    //udp1.send(message.c_str(), message.size(), "87.71.155.68",
    21568);

    //void* buffer = malloc(128);
    //size_t buffer_size = 128;
    //size_t received;
    //sf::IpAddress address("87.71.155.68");
    //unsigned short port = 21568;
    //udp2.receive(buffer, buffer_size, received, address, port);

    //cout << "End.\n";
    //std::cin.get();

    //return 0;

    //Window
    sf::RenderWindow window(sf::VideoMode(WIDTH, HEIGHT), "Program",
    sf::Style::Close, sf::ContextSettings(24, 8, 8));
```

```
    window.setFramerateLimit(60);


    Toaster toaster;

    Player player(40, 21, window, toaster);

    loginPage(window, player, toaster);


    // Game loop
    mainLoop(window, player, toaster);

    return 0;
}


void loginPage(sf::RenderWindow& window, Player& player, Toaster&
toaster)
{
    // background image
    sf::Texture login_tex, signup_tex;
    login_tex.loadFromFile("sprites/loginpage.jpg");
    signup_tex.loadFromFile("sprites/signuppage.jpg");

    sf::Sprite bg_sprite(login_tex);

    // font
    sf::Font input_font;
    if (!input_font.loadFromFile("Fonts/Roboto-Regular.ttf"))
    {
        std::cerr << "Error Loading File.\n";
        return;
    }

    bool logging_in = true;
    bool enter_pressed = false; // SHOULD BE
    FALSEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE

    // text box and text

    TextBox username(v2f(194, 249), v2f(461, 70), "", input_font);
    TextBox password(v2f(194, 355), v2f(461, 70), "", input_font);

    password.hidden = true;

    TextBox* text_boxes[3] = { nullptr, &username, &password };


    int box_focused = 1;

    sf::Clock clock;
```

main.cpp

```
v2f enter_position(193, 469), enter_size(462, 61);
v2f switch_position(530, 185), switch_size(130, 30);

while (window.isOpen())
{
    float dt = clock.restart().asSeconds();

    string typed_text = "";
    int backspace_counter = 0;


    sf::Event event;
    while (window.pollEvent(event))
        if (event.type == sf::Event::Closed)
            window.close();
        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
            window.close();
        else if (event.type == sf::Event::TextEntered) {
            //cout << event.text.unicode << "\n";
            // actual typing
            if (event.text.unicode > 32 && event.text.unicode <
            127) {
                typed_text += event.text.unicode;
            }

            // backspaces
            if (event.text.unicode == '\b')
                backspace_counter++;
            if (event.text.unicode == 127) // ctrl backspace
                backspace_counter = -100;

            // tab
            if (event.text.unicode == '\t' && box_focused)
            {
                box_focused = (box_focused + 1) % 3;
                if (box_focused == 0)
                    box_focused = 1;

                text_boxes[box_focused]->turnOnCursor();
            }

            //enter
            if (event.text.unicode == '\r')
                enter_pressed = true;

        }
        else if (event.type == sf::Event::MouseButtonPressed) {
            // Check if mouse click is within the text box
            v2i mousePos = sf::Mouse::getPosition(window);

            box_focused = 0;
            for (int i = 1; i < 3; i++)
            {
```

```
            if (text_boxes[i]->inBox(mousePos))
            {
                box_focused = i;
                text_boxes[i]->turnOnCursor();
            }

        }


        if (inBounds(enter_position, enter_size, mousePos))
            enter_pressed = true;

        if (inBounds(switch_position, switch_size, mousePos))
        {
            logging_in ^= true;
            if (logging_in) bg_sprite.setTexture(login_tex);
            else bg_sprite.setTexture(signup_tex);
            text_boxes[1]->clearText();
            text_boxes[2]->clearText();
            box_focused = 1;
        }
    }


if (enter_pressed)
{
    string error;
    if (logging_in)
    {
        if (player.client.tryLogIn(username.getString(),
        password.getString(), error))
        {
            toaster.toast("Connection Successful!");
            return;
        }
        toaster.toast(error);

    }

    else // signing up
    {
        if (player.client.trySignUp(username.getString(),
        password.getString(), error))
        {
            toaster.toast("Signup Successful!");
            logging_in = true;
            bg_sprite.setTexture(login_tex);
            text_boxes[1]->clearText();
            text_boxes[2]->clearText();
            box_focused = 1;
        }
        else
            toaster.toast(error);
```

```
            }
        }

        enter_pressed = false;


        window.clear(sf::Color::Red);

        window.draw(bg_sprite);


        //box highlight
        if (box_focused)
        {
            if (typed_text.size())
                text_boxes[box_focused]->addText(typed_text);
            if (backspace_counter)
                text_boxes[box_focused]->backspace(backspace_counter);
        }

        for (int i = 1; i < 3; i++)
            text_boxes[i]->draw(window, i == box_focused);

        toaster.drawToasts(window, dt);


        window.display();

    }
}

void mainLoop(sf::RenderWindow& window, Player& player, Toaster&
toaster)
{
    v2i screen_center(WIDTH / 2, HEIGHT / 2);


    int frame_count = 0;
    sf::Clock clock;



    Player::HitInfo* hits = new Player::HitInfo[WIDTH];

    std::thread udpThread(&Player::listenToServer, &player);

    player.setFocus(true);

    while (window.isOpen())
    {
        float dt = clock.restart().asSeconds();

        sf::Event event;
```

main.cpp

```
while (window.pollEvent(event))
    if (event.type == sf::Event::Closed)
        player.quitGame();
    else if (player.window_focused && event.type ==
    sf::Event::MouseButtonPressed)
        player.shootGun(event.mouseButton.button ==
        sf::Mouse::Left);
    else if (event.type == sf::Event::LostFocus)
        player.setFocus(false);
    else if (event.type == sf::Event::GainedFocus)
        player.setFocus(true);
    else if (player.window_focused && event.type ==
    sf::Event::MouseMoved)
    {
        v2i current_pos = sf::Mouse::getPosition(window);

        player.rotateHead(current_pos.x - screen_center.x,
            current_pos.y - screen_center.y, dt);

        sf::Mouse::setPosition(screen_center, window);
    }
    else if (event.type == sf::Event::KeyReleased)
    {
        if (event.key.code == sf::Keyboard::Space)
            player.respawn();
    }


//if (frame_count % 100 == 0)
//    cout << (1 / dt) << "\n";


//cout << "Frame: " << frame_count << '\n';

player.updateServer();


player.handleKeys(dt);


// Graphics
window.clear(sf::Color::Red);

player.map.drawSky(); // Sky

player.map.drawGround();


player.shootRays(hits); // populate hits[]

// World

{
```

```
            auto start = std::chrono::high_resolution_clock::now();
            std::lock_guard<std::mutex> lock(player.mtx);
            auto end = std::chrono::high_resolution_clock::now();

            player.drawWorld(hits, dt);

            std::chrono::duration<double> elapsed = end - start;

            //std::cout << "Elapsed time: " << elapsed.count() * 1000
            << " ms" << std::endl;
        }


        if (player.debug_mode)
        {
            player.rotateHead(1, 0, 0.3);
        }
        //player.debug();

        player.drawGun(dt); // Gun

        player.drawCrosshair(dt); // Crosshair

        player.drawDeathScreen(dt);


        toaster.drawToasts(window, dt);
        toaster.drawLeaderboard(window, player.leaderboard, dt);

        window.display(); // Render to screen

        frame_count++;
    }



    delete[] hits;
}
```

map.hpp

```
#pragma once

class Map
{
private:
■int* data;
■sf::RenderWindow& window;

■sf::Texture sky_tex;
■sf::Sprite sky_sprite;

public:
■int width, height;
■v2i position;

■int cell_size = 4;

■float floor_level = 354;

■float sky_offset = 0;
■float sky_scale = 2.5f;
■float sky_width = 1833;
■float sky_sensitivity = -1000;

■sf::Color sky_color, ground_color;

■Map(int dist_from_side, sf::RenderWindow& window);


■int getCell(int x, int y);
■void drawMap();
■void drawPoint(float x, float y);

■void drawGround();
■void drawSky();
■void shiftSky(float offset);
■void darkenScreen();
};
```

object.hpp

```
#pragma once
#include "headers.hpp"
#include "client.hpp"

class Object
{

public:
    sf::Sprite sprite;

    v2f tex_size;
    float scale_by, shrink_by;

    v2f position;
    float rotation_x = 0;
    int direction_index; // 0, 1, 2, 3, 4, 5, 6, 7

    bool moving = false, started_moving = false, forward;
    int animation_index = 0;
    float animation_timer = 0;

    float gun_timer = -1;

    float getting_shot_timer = -1;

    bool dead = false;
    float dying_timer = -1;

    int player_id = -1;
    string username = "MISSING USERNAME";


    Object();
    Object(float x, float y, const sf::Texture& tex);
    float distFrom(const v2f& pos);

    void loadPlayerInfo(Client::PlayerInfo player_info);

    void animate(float dt);
    void shootGun();
    void gotShot();
    void gotKilled();

    static sf::IntRect getTextureRect(float rotation, float frame);
};
```

player.hpp

```
#pragma once

#include "headers.hpp"
#include "map.hpp"
#include "object.hpp"
#include "client.hpp"
#include "toaster.hpp"
#include <SFML/Audio.hpp>


class Player
{
private:
    // game logic
    sf::RenderWindow& window;
    Toaster& toaster;
    bool has_quit;
    bool dead = false;

    sf::Texture* wall_texs;
    sf::Sprite wall_sprite;

    sf::Texture enemy_tex;
    vector<Object> objects;
    vector<Object*> sorted_objects;



    // gun animation
    int gun_animation_frame;
    float gun_animation_timer, gun_movement_stopwatch;
    float* gun_animation_duration;
    sf::Sprite gun_sprite;
    sf::Texture* gun_texs;
    v2f gun_position;
    float gun_offset_y = 0;
    v2f gun_offset;
    float max_hand_range = 40;
    float hand_move_range;

    string verbs[9] = { "zoinked", "zooked", "styled on",
        "slaughtered", "kassifed on", "nuked",
        "eradicated", "decimated", "pulverized" };

    // hit direction and reticle indicator
    sf::Texture indicator_texture, reticle_texture;
    sf::Sprite hit_indicator_sprite, reticle_sprite;
    vector<float> hit_direction_timers;
    vector<float> hit_direction_angles;
    float reticle_timer = -1;



    //gun shot
```

```
    bool gun_shot;
    sf::Texture damage_overlay_tex;
    sf::Sprite damage_overlay_sprite;
    float current_damage_opacity;
    float max_damage_opacity = 130;
    string killer_name;

    // movement
    v2f position;
    float speed = 2.0f;
    bool running = false, crouching = false;
    bool moving = false, moving_forward;
    float run_multiplier = 1.75f, crouch_multiplier = 0.5f;

    // orientation
    float rotation_x = -3.169f;
    float rotation_y = -0.56f;
    float mouse_sensitivity = 0.05f;
    float fov_y = 0.7f;
    float fov_x = 1.22173f; // 70 degrees

    // map
    float body_radius = 0.4f;

    //sound
    sf::SoundBuffer gunshot_buffer, gunclick_buffer;
    sf::Sound gun_sound, click_sound;

    //font
    sf::Font nametag_font, bold_font, deathscreen_font;

    //debug
    float debug_float = 0;

    int received_events_size = 0;
    char received_events[128];

    int score = 0;

public:
    Client client;
    std::mutex mtx;
    Map map;
    bool window_focused;

    // leaderboard
    vector<Toaster::LeaderboardEntry> leaderboard;

    bool debug_mode = false;

    struct HitInfo {
        float distance;
        bool on_x_axis;
```

```
    float texture_x;

    float perceived_distance;
};



Player(int x, int y, sf::RenderWindow& window, Toaster& toaster);

void setFocus(bool focus);
void handleKeys(float dt);
void rotateHead(int delta_x, int delta_y, float dt);
void move(float angle_offset, float dt);

void shootRays(HitInfo*& hits);
void drawWorld(HitInfo*& hits, float dt);
void drawColumn(int x, const Player::HitInfo& hit_info);
Player::HitInfo shootRay(float angle_offset);

void drawObject(Object& object, float dt);
void drawGun(float dt);
void drawCrosshair(float dt);
void drawDeathScreen(float dt);

void shootGun(bool left_click);
void getShot(int shooter_id);

void loadSFX();
void loadTextures();

void quitGame();

//server
void updateServer();
void listenToServer();
void handleEvents(char* events, int event_count);
Client::PlayerInfo getPlayerInfo();

Object* getObject(int id);
Object* getAnyObject();
void handle_shooting_victim(int victim_id, int shooter_id);
void handle_killing(int killer_id, int victim_id);
void getKilled(const string& killer_name);
void respawn();

void addToLeaderboard(int player_id, int score, const string&
username);
void updateLeaderboard(int player_id);


string getUsername(int id);

//
```

```
    void debug();

};
```

toaster.hpp

```
#pragma once
#include "headers.hpp"
#include "tools.hpp"

class Toaster
{
private:
    vector<string> toasts;
    vector<float> toast_slides;
    vector<float> toast_timers;

    v2f first_toast_position;
    float goal_y = 0;
    v2f toast_size = { 250, 80 };

    sf::Texture toast_tex;
    sf::Sprite toast_sprite;

    sf::Font font;
    sf::Text text;

    v2f text_position = { 36, 40 };

    float lifetime = 6; // in seconds

    // leaderboard

    sf::Texture notch_tex, board_tex;
    sf::Sprite notch_sprite, board_sprite;


    v2f leaderboard_position = { 20, 10 };
    float leaderboard_scale = 0.8f;

    sf::Text leaderboard_text;
    v2f leaderboard_name_offset = { 9, 6 };
    v2f leaderboard_score_offset = { 185, 6 };

public:
    struct LeaderboardEntry
    {
        int player_id;
        string username;
        int score;

        float position_y;

        LeaderboardEntry(int player_id, int score, const string&
        username);
    };

    Toaster();
    void drawToasts(sf::RenderWindow& window, float dt);
```

toaster.hpp

```
    void drawLeaderboard(sf::RenderWindow& window,
    vector<LeaderboardEntry>& leaderboard, float dt);
    void toast(const string& text);
};
```

tools.hpp

```
#pragma once
#include "headers.hpp"
#include <sstream>

v2i min(const v2i& first, const v2i& second);

v2i max(const v2i& first, const v2i& second);

float mag(const v2f& vec);

v2f norm(const v2f& vec);

float lerp(float a, float b, float t);

sf::Color lerp(sf::Color c1, sf::Color c2, float t);

bool inBounds(const v2f& box_pos, const v2f& box_size, const v2i& pos);

vector<string> split(const string& str);

float angleBetweenVectors(const v2f& v1, const v2f& v2);


struct TextBox
{
    v2f position, size;

    string text_string;
    sf::Text text;
    v2f text_offset = { 20, 16 };

    sf::RectangleShape shadow;

    sf::Clock cursor_timer;

    sf::RectangleShape cursor;
    bool cursor_visible = true;

    bool hidden = false;

    TextBox(const v2f& pos, const v2f& size, const string& str, const
    sf::Font& font);


    void addText(const string& added_text);
    void backspace(int backspace_counter);
    void draw(sf::RenderWindow& window, bool is_focused);
    void clearText();

    string getString();
    bool inBox(const v2i& pos);

    void turnOnCursor();
```

```
};
```