МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

Шахматы

Курсовая работа
по дисциплине «Программирование»
студента 1 курса группы ПИ-б-о-201
Ильясова Эмира Шухратовича

направления подготовки 09.03.04 «Программная инженерия»

Научный руководитель _____
(оценка)

старший преподаватель кафедры
компьютерной инженерии и моделирования _____ Чабанов В.В.
(подпись, дата)

Симферополь, 2021

РЕФЕРАТ

Шахматы. – Симферополь: ФТИ КФУ им. В. И. Вернадского, 2021. – 41 с., 17 ил., 7 ист.

Объект исследования данной работы разработка игра в шахматы, сервер игры.

Цель работы – создание рабочего игрового проекта с реализацией клиент-сервер системы. Сервер реализуется с использованием языка Python, а клиент с использованием языка C++.

Было рассмотрено многочисленное количество вариантов реализации клиент-серверных систем, в ходе которых было принято решение использовать сокеты для взаимодействия клиента с сервером. Это позволило обеспечить обмен данными в режиме достаточно приближенному к реальному времени.

ПРОГРАММИРОВАНИЕ, C++, SFML, PYTHON, СОКЕТЫ, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ

Оглавление

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ПО	Программное обеспечение
IDE	Интегрированная среда разработки
TCP	Transmission Control Protocol
IP	Internet Protocol
SFML	Simple and Fast Multimedia Library

ВВЕДЕНИЕ

Данная работа основана на разработке игры в шахматы. Ее актуальность заключается в том, что, играя в шахматы, люди оттачивают стратегическое и тактическое мастерство, так же шахматы представляют собой великолепную модель для обучения детей внутреннему плану умственных действий, что позволит им принимать в жизни ответственные и логичные решения. Для реализации данного проекта были выбраны такие языки программирования как C++ и Python.

При планировании данного проекта была поставлена следующая цель – Программная реализация сетевой игры «Шахматы».

Для достижения цели в данной работе были поставлены следующие задачи:

1. изучить основные правила игры в шахматы;
2. изучить стандартную библиотеку SFML;
3. изучить библиотеку для работы с базами данных SQLite;
4. изучить работу с сокетами;
5. реализация программного кода;
6. отладка и тестирование.

Объектом исследования является игра в шахматы.

Предметом исследования является реализация клиент-серверной игры в шахматы.

Методы исследования:

- моделирование – построение и изучение моделей с целью приобретения новых знаний, улучшение характеристик объектов исследований или управление ими;
- сравнение – сравнение предметов между собой на выявление сходства и различия, преимуществ и недостатков.

ГЛАВА 1
ПОСТАНОВКА ЗАДАЧИ

1.1 Цель проекта

По итогу разработки планируется создать игру в шахматы, в которую можно будет играть двум пользователям по сети. Данная игра будет отличаться от классических шахмат тем, что будет возможность до начала партии поменять пешки на другие фигуры, если у игрока будут хватать на это очков.

1.2 Существующие аналоги

- NetChess – программа предназначена для игры в шахматы по локальной сети или на одном компьютере с оппонентом;
- Kasparov Chessmate – шахматная программа, которая позволяет играть не только по сети с другими игроками, но и с ботами разных уровней сложности;
- GambitChess – классические шахматы с возможностью играть в формате 2D либо 3D.

1.3 Основные отличия от аналогов

- Наличие сервера для ведения статистики игроков;
- Возможность получения очков за победу;
- Возможность использования полученных очков для смены пешек на другие фигуры до начала партии.

1.4 Техническое задание

Программа должна соответствовать следующему техническому заданию:

1. Главное меню программы должна иметь несколько кнопок для:
   1.1. Запуска партии на одном устройстве;
   1.2. Для создания игровой комнаты по сети;
   1.3. Для подключения к игровой комнате по сети;

1.4. Для просмотра статистики игроков.

2. Программа должна корректно интерпретировать и соблюдать следующие основные правила игры.

Шахматная партия играется между двумя партнерами, которые поочередно перемещают фигуры на квадратной доске, названной "шахматной". Играющий белыми начинает партию. Игрок получает право хода, когда партнер сделал ход.

Цель каждого игрока – атаковать короля партнера таким образом, чтобы партнер не имел никаких возможных ходов, которые позволяют избежать "взятия" короля на следующем ходу. Об игроке, который достиг этой цели, говорят, что участник поставил мат королю партнера и выиграл партию. Партнер, королю которого был поставлен мат, проиграл партию.

Если позиция такова, что никто из партнеров не может поставить мат, партия заканчивается вничью.

Шахматная доска состоит из 64 равных квадратов (8x8), поочередно светлых и темных. Доска располагается между игроками так, чтобы ближайшее угловое поле справа от игрока было светлым.

В начале партии один игрок имеет 16 светлых фигур ("белые"); другой - 16 темных фигур ("черные"): Белый король, Белый ферзь, Две белые ладьи, Два белых слона, Два белых коня, Восемь белых пешек, Черный король, Черный ферзь, Две черные ладьи, Два черных слона, Два черных коня, Восемь черных пешек.

Восемь вертикальных рядов квадратов называются "вертикалями". Восемь горизонтальных рядов квадратов называются "горизонталями". Прямые линии квадратов одного и того же цвета, касающихся углами, называются "диагоналями".

Ни одна из фигур не может быть перемещена на поле, занятое фигурой того же цвета. Если фигура переходит на поле, занимаемое фигурой партнера, последняя считается взятой и убирается с шахматной доски как часть того же самого хода. О фигуре говорят, что она атакует фигуру партнера, если эта фигура может произвести взятие на этом поле, согласно пунктам.

Слон может ходить на любое поле по диагоналям, на которых стоит.

Ладья может ходить на любое поле по вертикали или горизонтали, на которых стоит.

Ферзь ходит на любое поле по вертикали, горизонтали или диагонали, на которых стоит.

Когда делаются эти ходы, ферзь, ладья или слон не могут перемещаться через поле, занятое другой фигурой.

Конь может ходить на одно из ближайших полей от того, на котором стоит, но не на той же самой вертикали, горизонтали или диагонали.

Пешка может ходить вперед на свободное поле, расположенное непосредственно перед ней на той же самой вертикали, также с исходной позиции пешка может продвинуться на два поля по той же самой вертикали, если оба эти поля не заняты, либо пешка ходит на поле, занимаемое фигурой партнера, которая расположена по диагонали на смежной вертикали, одновременно забирая эту фигуру.

Когда пешка достигает самой дальней горизонтали от своей исходной позиции, то должна быть заменена на ферзя, ладью, слона или коня "своего" цвета, что является частью того же хода. Выбор игрока не ограничивается фигурами, которые были уже сняты с доски. Эта замена пешки на другую фигуру называется "превращением", и действие новой фигуры начинается сразу.

Король может ходить на любое примыкающее поле, которое не атаковано одной или более фигурами партнера. Считается, что фигуры партнера атакуют поле, даже в том случае, когда фигуры не могут ходить.

Рокировка – перемещение короля и одной из ладей того же цвета по крайней горизонтали. Это считается одним ходом короля и выполняется следующим образом: король перемещается с исходного поля на два поля по направлению к ладье, затем ладья переставляется через короля на последнее поле, которое только что король пересек.

Рокировка становится невозможной:

- если король уже ходил;

- если ладья уже ходила.

Рокировка временно невозможна:

- если поле, на котором стоит король, или поле, которое король должен пересечь, или поле, которое король должен занять, атаковано одной из фигур партнера;
- если между королем и ладьей, с которой должна быть произведена рокировка, находится какая-то фигура, мешающая рокировке.

Считается, что король находится "под шахом", если атакован хотя бы одной фигурой партнера, даже если такие фигуры сами не могут перемещаться. Объявление шаха не обязательно.

Ни одна из фигур не может сделать ход, который ставит или оставляет своего короля под шахом.

ГЛАВА 2
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

2.1 Анализ инструментальных средств

Самым большим вложением является SFML – Simple and Fast Multimedia Library. SFML – это библиотека для представления мультимедийных данных. SFML обеспечивает простой интерфейс для разработки игр и прочих мультимедийных приложений. Состоит она из пяти модулей:

- system – управление временем и потоками, он является обязательным, так как все модули зависят от него;
- window – управление окнами и взаимодействием с пользователем;
- graphics – делает простым отображение графических примитивов и изображений;
- audio – предоставляет интерфейс для управления звуком;
- network – для сетевых приложений.

Благодаря модулю network появилась возможность осуществления передачи данных по порту и прослушивания порта по протоколу TCP: Всё это было крайне важно при создании интерфейса программы, где нужно графически принимать ввод от пользователя для регистрации, входа и создания игровой комнаты, так же для подключения к серверу. Из аналогов были также рассмотрены такие библиотеки как SDL и Allegro. Allegro – это кроссплатформенная библиотека, в основном предназначенная для видеоигр и мультимедийных программ. Она выполняет общие, низкоуровневые задачи, такие как создание окон, прием пользовательского ввода, загрузка данных, рисование изображений, воспроизведение звуков. SDL – это свободная кроссплатформенная мультимедийная библиотека, реализующая единый программный интерфейс к графической подсистеме, звуковым устройствам и средствам ввода для широкого спектра платформ.

Не смотря на все плюсы библиотеки SFML, невозможно не заметить его минусы. К примеру, в данной библиотеке отсутствует простое создание

текстового поля для получения данных пользователя, такой обработчик придется реализовывать самостоятельно.

Еще были выбраны ряд других инструментов, таких как:

- Среда Microsoft Visual Studio 2019 – была выбрана в пользу знакомого интерфейса и возможности программирования на языке C++ и подключения библиотеки SFML;
- Интегрированная среда разработки JetBrains PyCharm была выбрана из-за ее приспособленности к разработке программ на языке Python и удобства подключения дополнительных библиотек;
- C++ и Python – как изучаемые языки программирования;
- Библиотека socket – сетевое взаимодействие со стороны сервера
- Библbотека sqllite3 для работы с базами данных на языке SQL.

2.2 Описание алгоритмов

2.2.1 Алгоритм работы окон

В данном проекте присутствуют понятия окон и перехода между ними. В оконной логике приложения есть 4 основных блока:

1. Блок управления – основной цикл, в котором последовательно строятся основные окна приложения в зависимости от готовности перейти на следующий блок;
2. Блок авторизации/регистрации – Первый оконный блок приложения, в котором можно зарегистрироваться и войти в свой аккаунт. Если вход возможен, то осуществляется переход на следующий блок;
3. Блок меню игры – Второй блок, содержащий в себе кнопки:
   3.1. «Одиночная» – Означает одиночную игру и одобряет переход на следующий блок игрового поля без клиент-серверных взаимодействий;
   3.2. «Сервер» – регистрирует имя комнаты на сервере и опрашивает сервер о наличии в комнате второго игрока, при наличии присоединённого к комнате второго игрока осуществляется переход к блоку игрового поля с клиент-серверным взаимодействием со вторым игроком;

3.3. «Клиент» – присоединяется к существующей комнате, при успешном добавлении второго игрока (самого «клиента») в созданную первым игроком («сервером») комнату, осуществляется переход к блоку игрового поля с клиент-серверным взаимодействием с первым игроком;

3.4. «Статистика» – получение от сервера списка лучшей десятки игроков.

4. Блок игрового поля – в зависимости от режима последовательность ходов будет осуществляться по-разному. При одиночной игре, игрок сам будет ходить по очереди в одном окне. При клиент серверном взаимодействии последовательность ходов будет осуществляться по очереди, ходящий игрок в конце хода отправляет актуальную доску другому игроку, а другой игрок в свое время периодически слушает порт и ожидает прихода команд от ходящего игрока (ожидания завершения им хода и приема актуального игрового поля). После закрытия блока игрового поля, осуществляется переход на блок меню игры. Выйти из игры можно на блоках авторизации/регистрации и меню игры.

Данные блоки взаимодействует с сервером. их взаимодействие показано на рисунке 2.1 в виде UML диаграммы.

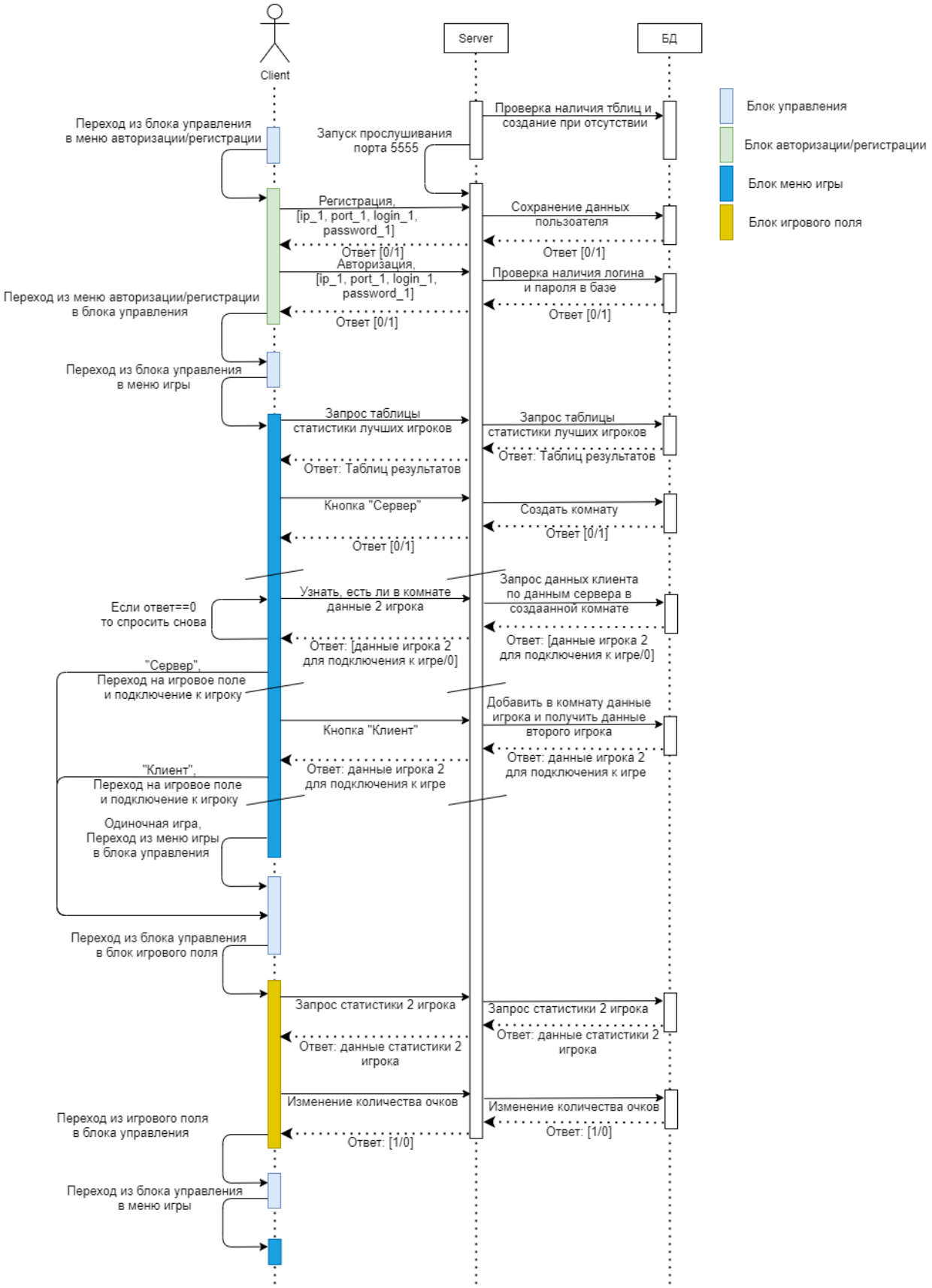

Рисунок 2.1. Схема взаимодействия клиент-сервер.

При запуске программы первое окно, которое видит пользователь – это окно входа и регистрации. Пользователь вводит данные логин и пароль, клиент

проверяет все ли поля заполнены корректно, после этого отправляет запрос на сервер для проверки существования учетной записи данного пользователя (рисунок 2.2).

```
int check_authorisation(string s_ip, string s_p, string data) {
    int accept = 0;
    std::vector<sf::TcpSocket*> clients;
    sf::TcpSocket socket;
    sf::TcpListener listener;
    listener.setBlocking(false);
    sf::SocketSelector selector;
    const std::string address = s_ip;
    const int port = atoi(s_p.c_str());
    if (socket.connect(address, port) == sf::Socket::Done) {
        std::cout << "Connected to the server" << std::endl;
    }
    selector.add(socket);
    sf::Packet packet;
    packet << data;
    socket.send(packet);
    packet.clear();

    char data1[2048];
    sf::Packet packet2;
    sf::Clock clock, pause;
    float ftime = clock.getElapsedTime().asSeconds();
    clock.restart();

    std::size_t received;
    socket.receive(data1, sizeof(data1), received);
    string ans = get_str_in_char(data1);

    accept = string(data1)[0] - '0';

    if (accept == 1)
        return 1;

    return 0;
}
```

Рисунок 2.2. Проверки авторизации пользователя.

После этого сервер проверяет данные в базе данных и возвращает результат (рисунок 2.3). Если такая учетная запись есть в базе данных пользователь попадет

в игровое меню, а если ее не существует, тогда пользователю придаться зарегистрироваться.

```
def run(connection, addr_user):
    try:
        data = get_data(connection)
        data=data.split('|')
        print(addr_user, data)
        if data[1]=='login':
            answer=str(Login(data[2],data[3]))
        elif data[1]=='reg':
            cou=find('users.db',"USERS",data[2])
            if cou==None:
                add_data_user('users.db',"USERS",[data[2],data[3], 0, 0])
                answer="1"
            else:
                answer="0"
```

Рисунок 2.3. Получение данных на сервере и их обработка.

2.2.2 Алгоритм проверки ходов фигур

В зависимости от того, кто ходит, белые или черные функция “Chess_move” (рисунок 2.4) обозначает направление движения и проверяет правильность хода фигур. В самом начале пешки могут по желанию игрока переместиться на две клетки, поэтому создается матрица, которая является своеобразным списком, для проверки сходила ли пешка на две клетки. После того как игрок хватает мышью фигуру и пытается ее перетащить на другую клетку, функция “Chess_move” получает данные об этой фигуре и о ее перемещении, после этого он запускает, ту или иную проверку хода.

```
void Chess_move(struct STATUS& status, int board[8][8]) {


    int Figure = status.fig;
    POS Old_p = status.p_old,
        New_p = status.p_new;

    status.ok = 0;
    status.attak = 0;
    status.p_attak.x = -1;
    status.p_attak.y = -1;

    int end = 0;
    if (Figure == figs.Pawn && end == 0) {
        int curent_vec = (status.whose_move == WHITE) ? 6 : 1;
        if (Old_p.y == curent_vec) {
            if (New_p.y == Old_p.y + status.whose_move * 1 &&
                New_p.x == Old_p.x &&
                board[Old_p.y + status.whose_move * 1][Old_p.x] == 0) {
                status.ok = 1;
                status.attak = 0;
            }
            if (New_p.y == Old_p.y + status.whose_move * 2 &&
                New_p.x == Old_p.x &&
                board[Old_p.y + status.whose_move * 1][Old_p.x] == 0 &&
                board[Old_p.y + status.whose_move * 2][Old_p.x] == 0) {
                PawnsJump[status.whose_move][Old_p.x] = 1;
                status.ok = 1;
                status.attak = 0;
            }
        }
        else if (New_p.y == Old_p.y + status.whose_move * 1 &&
            New_p.x == Old_p.x &&
            board[Old_p.y + status.whose_move * 1][Old_p.x] == 0) {
            status.ok = 1:
```

Рисунок 2.4. Функция для проверки ходов.

2.3 Описание структур данных

2.3.1 Клиент

Самой главной структурой данных выступает двумерный массив, хранящий в себе игровое поле с фигурами в виде чисел (рисунок 2.5). К данному массиву обращаются большинство функций в программе. При каждом ходе пользователя, числа меняются местами в массиве, после чего в соответствии с ним вырисовывается шахматная доска. В процессе игры, клиенты образуют между

собой клиент-серверное взаимодействие, это выражено тем что в моменты ожидания хода игрок слушает приходящие на порт команды и проверяет что команды принадлежат противоположному игроку по коду игровой сессии (перед переключении из меню в игровой режим, каждый из игроков получает информацию о противоположном игроке и ключ игровой комнаты, в которой они играют). После начала игры данный двумерный массив, как игровое поле постоянно передается между клиентами. Схема взаимодействия двух клиентов по сети показана на рисунке2.6.

```
int board[LENGTH][LENGTH] =
{ 2, 3, 4, 5, 6, 4, 3, 2,
  1, 1, 1, 1, 1, 1, 1, 1,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
 -1,-1,-1,-1,-1,-1,-1,-1,
 -2,-3,-4,-5,-6,-4,-3,-2,
};
```

Рисунок 2.5. Главная структура данных.

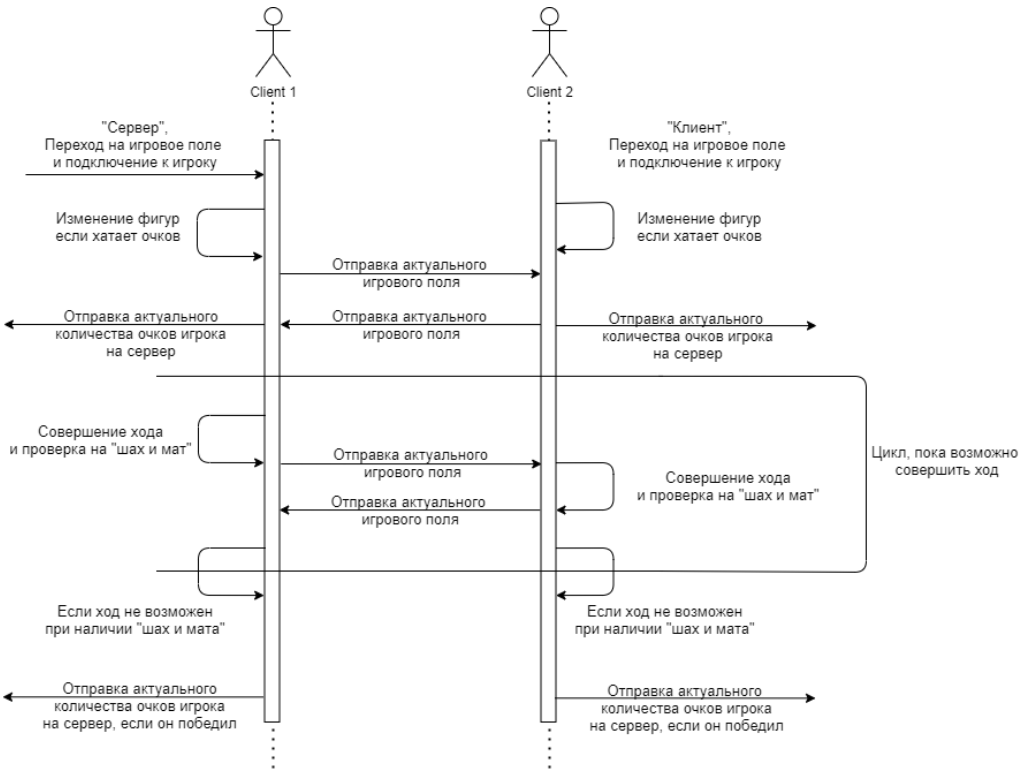

Рисунок 2.6. Схема взаимодействия клиент-клиент

В клиенте так же есть несколько других структур данных (рисунок 2.7) таких как:

- Структура координат позиций;
- Структура имен фигур;
- Структура контроля статуса хода.

```
struct POS {
    int x = -1, y = -1;
}p_cur, t_transform;

struct FIG {
    int Pawn = PAWN,
        Rook = ROOK,
        Knight = KNIGHT,
        Bishop = BITSHOP,
        Queen = QUEEN,
        King = KING;
}figs;

struct STATUS {
    POS p_old,
        p_new,
        p_attak;
    int ok,
        attak,
        whose_move,
        fig,
        figNum;

}status;
```

Рисунок 2.7. Структуры данных клиента.

2.3.2 Сервер

Для хранения информации об аккаунтах пользователей при запуске сервера автоматически создаются таблицы «users» и «rooms» базы данных (рисунок 2.8). В таблице «users» хранится информация о логине, пароле, количестве побед и очков. А таблица «rooms» хранит информацию о созданных игроками комнатах.

```
name_db=['users.db','rooms.db']


create_users="""
    CREATE TABLE USERS (
        id INTEGER NOT NULL ,
        name TEXT NOT NULL PRIMARY KEY,
        password TEXT,
        points INTEGER,
        wins INTEGER);"""
create_rooms="""
    CREATE TABLE ROOMS (
        id INTEGER NOT NULL ,
        name TEXT NOT NULL PRIMARY KEY,
        user1 TEXT,
        user1_ready INTEGER,
        user2 TEXT,
        user2_ready INTEGER,
        freedoom INTEGER,
        password_room TEXT);
"""
def init_db(name_db,cmd):
    con = sql3.connect(name_db)
    try:
        with con:
            con.execute(cmd)
    except:
        pass


init_db(name_db[0],create_users)
init_db(name_db[1],create_rooms)
```

Рисунок 2.8. Создание таблиц базы данных.

2.4 Описание основных модулей

После запуска игры пользователь попадает в окно авторизации (рисунок 2.9). После входа открывается главное меню (рисунок 2.10). Из главного меню пользователь может:

- перейти в одиночную игру нажав на кнопку «По очереди» (рисунок 2.11);
- создать игровую комнату нажав на кнопку «Сервер» и введя название комнаты;

- подключиться к игровой комнате, введя название комнаты;
- посмотреть статистику топ 10 игроков (рисунок 2.12).

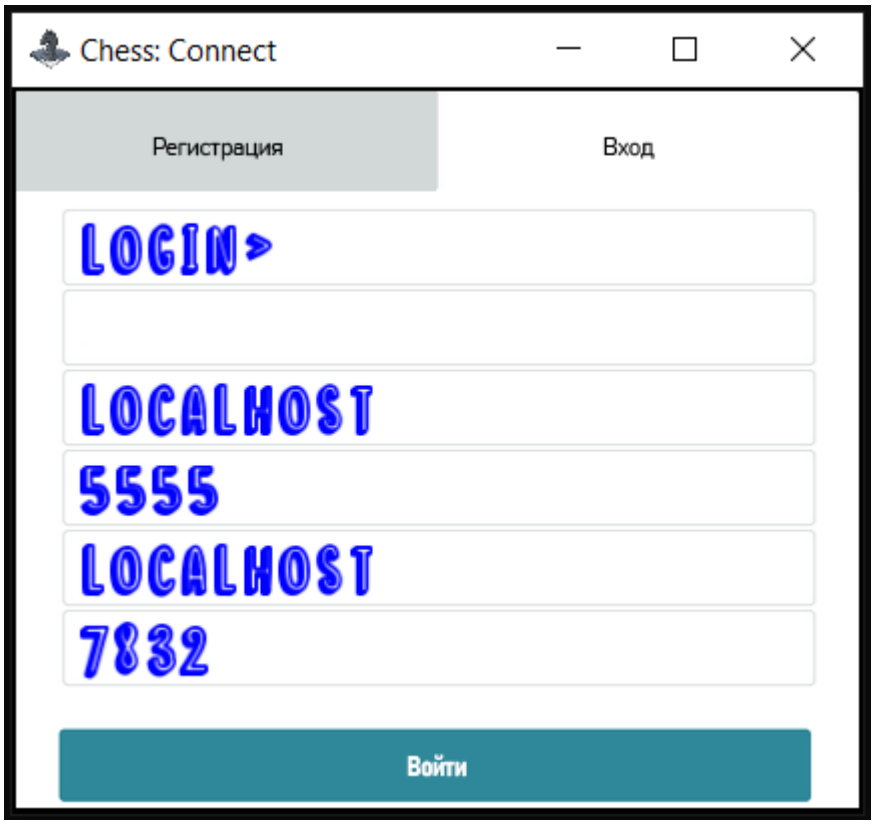

Рисунок 2.9. Окно авторизации.

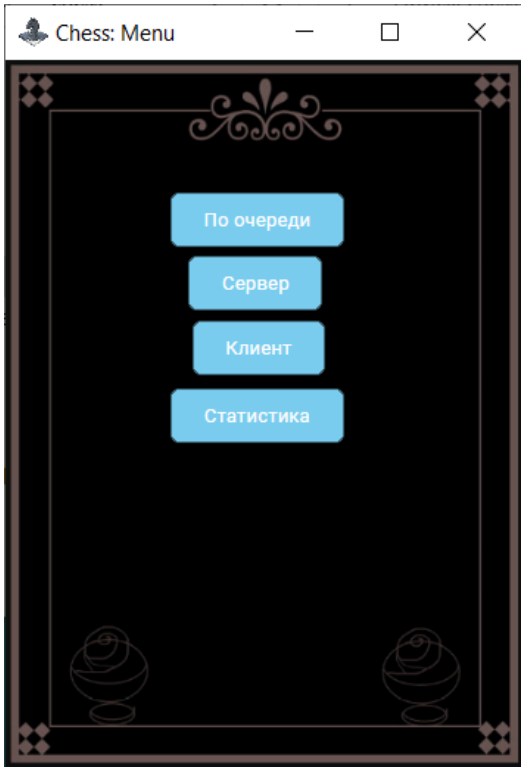

Рисунок 2.10. Главное меню игры

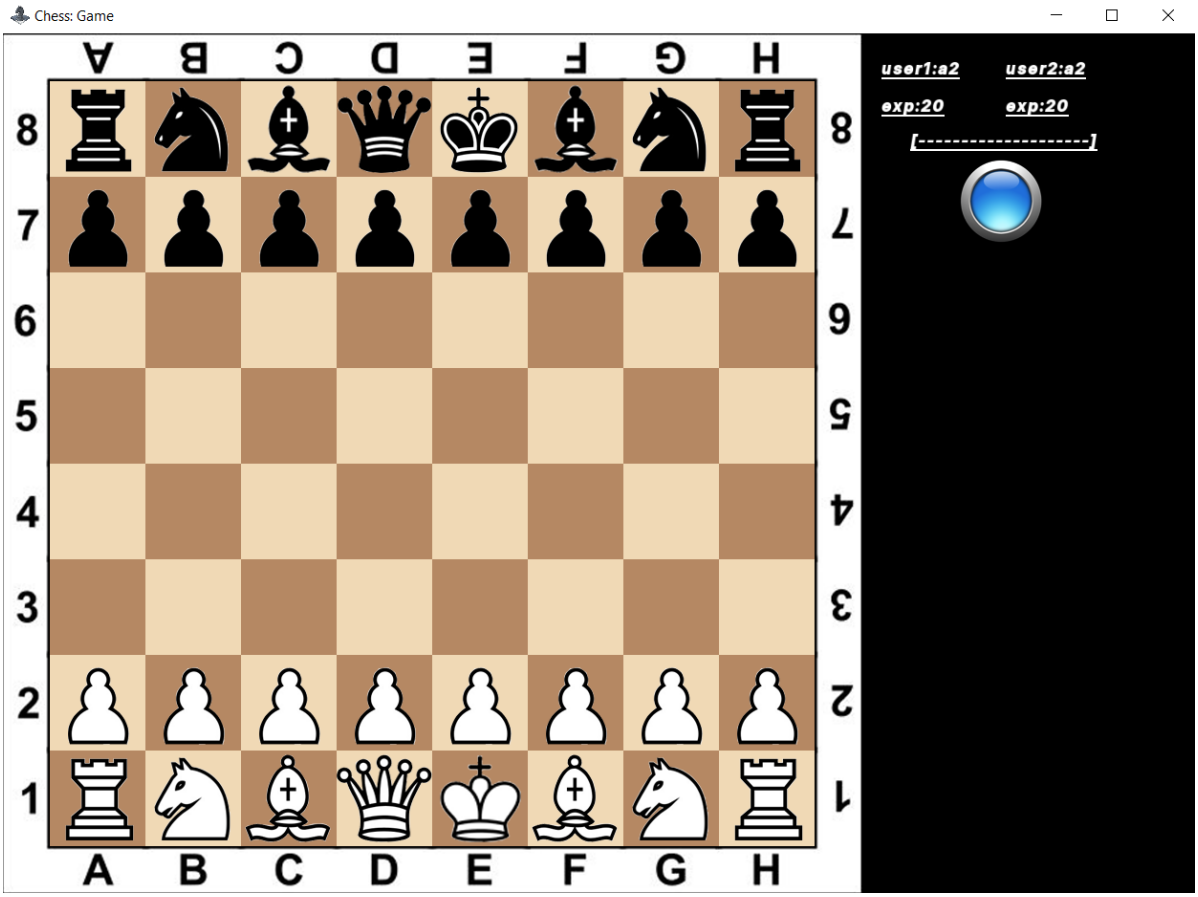

Рисунок 2.11. Одиночная игра.

Рисунок 2.12. Статистика игроков.

ГЛАВА 3
ТЕСТИРОВАНИЕ ПРОГРАММЫ

3.1 Тестирование исходного кода

Тесты для проверки исходного кода проекта были предоставлены, однако само тестирование проведено не было.

3.2 Тестирование интерфейса пользователя и юзабилити

Проект делится на два больших модуля: сервер и клиент. Все тесты на проверку данных модулей прошли успешно. Всего было пять тестов. Первый тест был направлен на работоспособность сервера (рисунок 3.1). Второй тест был направлен на проверку работоспособности клиента (рисунок 3.2). Третий тест проверял возможность подключения к серверу и авторизации (рисунок 3.3). Четвертый тест проверял работоспособность игрового меню: запуск одиночной игры, создание игровой комнаты, подключение к игровой комнате и просмотр статистики (рисунок 3.4). Последний тест был направлен на проверку работоспособности игры (рисунок 3.5): смены пешек на другую фигуру до начала партии, перемещение фигур по шахматным правилам и т.д.

Тест 1 - Запуск и закрытие сервера

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Запуск сервера	Программа запускается без падений и ошибок. В папке, где находиться программа создается два файла данных.	Соответствует ожидаемому результату	Пройден.	
2	Нажатие на кнопку «Выход»	Программа завершает свою работу	Программа выполняет завершение работы	Пройден.	

Рисунок 3.1. Тест сервера.

Тест 2 – Запуск и закрытие клиента

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Запуск клиента	После запуска клиента открываются окна для ввода логина и пароля или регистрации (на выбор).	Окна для ввода логина и пароля открываются, соответствует ожидаемому результату.	Пройден	Предварительно запустить сервер. Логин и пароль не могут быть на кириллице.
2	Нажатие на кнопку закрытия окна	Окно клиента закрывается.	Соответствует ожидаемому результату.	Пройден	

Рисунок 3.2. Тест клиента.

Тест 3 – Проверка возможности подключения к серверу

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	В окне вход ввести данные и нажать на кнопку “Войти”	После нажатия кнопки “Войти” появится окно меню. Пользователь видит четыре кнопки: “По очереди”, “Сервер”, “Клиент”, “Статистика”.	При входе в систему, открывается меню с ожидаемым результатом. (с предварительно запущенным клиентом)	Пройден	Предварительно запустить клиент. Используется клиент №1
2	Запуск второго клиента. Зарегистрировать второго пользователя и войти.	Вторая копия программы также запускается корректно, пользователь видит на втором окне все то же самое, что и на первом.	Соответствует ожидаемому результату.	Пройден	Используется клиент №2

Рисунок 3.3. Тест подключения к серверу при авторизации

Тест 4 – Проверка возможностей игрового меню

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	Запуск игры на одном клиенте, нажатие на кнопку «По очереди»	Будет отображено игровое поле в первом клиенте, будет доступна возможность ходить фигурами черного и белого цвета.	Отображается игровое поле в Клиенте №1, появляются возможности управления фигурами.	Пройден.	Используется клиент №1
2	Нажать на кнопку закрыть	Откроется окно меню	Открывается окно меню.	Пройден.	
3	Нажать на кнопку «Сервер», ввести название комнаты, и снова нажать на кнопку «Сервер»	Появиться либо сообщение «No», если комната с таким названием уже существует, либо сообщение «Ok», если комната успешно создана и ждет подключения второго игрока	При вводе необходимых данных отображается ожидаемый результат.	Пройден.	Используется клиент №1
4	Нажать на кнопку «Клиент», ввести название комнаты, и снова нажать на кнопку «Клиент»	На первом и втором клиентах появятся окна с игровыми полями	Соответствует ожидаемому результату.	Пройден.	Используется клиент №2
5	Нажать на кнопку закрыть на обоих клиентах	Откроется игровое меню	Открывается игровое меню.	Пройден.	

Рисунок 3.4. Тест для проверки работоспособности игрового меню

Тест 5 – Проверка работоспособности игры

Шаг /Step No.	Действие (операция) /Process (Actions)	Ожидаемый результат /Expected Results	Результат /Actual Results	Пройден /не пройден/ не доступен*	Комментарии /Notes
1	До начла партии, нажать левой кнопкой мыши на белую пешку и выбрать другую фигуру	Пешка должна поменяться на выбранную фигуру (если хватит очков для смены фигуры)	В результате замены фигуры (успешно меняется на выбранную)	Пройден.	Должно быть как минимум 15 очков у игрока
2	Для начала партии на обоих клиентах нажать на кнопку готовности в правом верхнем углу	Появиться возможность ходить белыми фигурами, а после хода белых черными	Ожидаемый функционал реализуется.	Пройден.	
3	Зажать левой кнопкой фигуру белого цвета и попробовать перетащить на другую ячейку	Фигура перейдет на другую ячейку, если переход допускается правилами игры в шахматы	Осуществляется переход на другую ячейку. (при условии правил шахмат)	Пройден.	
4	Попробовать сыграть партию в шахматы и проверить правильность ходов каждой фигуры	Фигуры должны перемещаться в соответствии с шахматными правилами, после завершения партии появиться сообщение об выигрыше у победителя и сообщение об окончании у проигравшего	Фигуры перемещаются только по им заданным правилам, после завершения партии выводится сообщение.	Пройден	
5	Закрыть окна с сообщениями	Возврат в главное меню	Возвращение в главное меню.	Пройден.	

Рисунок 3.5. Тест для проверки работоспособности игры

ГЛАВА 4
ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА

4.1 Перспективы технического развития

Во время реализации данного проекта были реализованы практически все задачи поставленные в начале работы. Однако из-за недостатка опыта разработки клиент-серверных приложений игра получилась очень простой, так же внешний вид игры не соответствует начальным ожиданиям. Библиотека SFML оказалась не лучшим решением для создания интерфейса игры. Ее возможности очень малы и для создания современного дизайна их точно не хватает.

Изначально не планировалось добавлять возможность одиночной игры, но это было сделано с целю в дальнейшем разработать ботов, которые смогут играть с пользователями. Добавление ботов в игру вызовут больший интерес к данному проекту, ведь не всегда можно найти соперника, а играть с самим собой не так уж и интересно.

В дальнейшем планируется не только добавления ботов, но и внешний интерфейс программы будет переписан с использованием веб-технологий.

4.2 Перспективы монетизации

На данном этапе монетизация не планируется, но после доработок игры есть возможность подключения контекстной рекламы.

ЗАКЛЮЧЕНИЕ

В результате работы над данным проектом был получен опыт в разработке проектов в среде разработки Microsoft Visual Studio 2019 и PyCharm, распределении времени и выставления приоритетов в процессе разработки. Так же был получен опыт в поиске и выборе библиотек, позволяющих реализовать функционал, нужный приложению. Был получен опыт работы со сторонними, ранее не изученными библиотеками, а именно SFML, socket и sqlite. Освоен принцип реализации работы клиент-серверных структур.

В процессе разработки проекта была достигнуты поставленные задачи, что привело к достижению поставленной цели, а именно программная реализация игры «шахматы», с добавлением возможности заменять пешки на другие фигуры при достаточном количестве очков для замены, а также реализацией многопользовательского режима (режима игры для двоих игроков).

Проект, находится на ранних этапах разработки, и все еще нуждается в доработке, интерфейс программы очень простой, в некоторых местах все еще можно найти грубоватые заготовки или не полностью продуманные функции, однако весь функционал в программе рабочий и отвечает целям, поставленным в начале разработки.

ЛИТЕРАТУРА

1. ГОСТ 19.002-80 Схемы алгоритмов и программ. Правила выполнения [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

2. ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначение условные графические [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

3. Оформление выпускной квалификационной работы на соискание квалификационного уровня «Магистр» («Бакалавр»): методические рекомендации. / сост. Бержанский В.Н., Дзедолик И.В., Полулях С.Н. – Симферополь: КФУ им. В.И.Вернадского, 2017. – 31 с.

4. Книга рецептов сетевого программирования Python [электронный ресурс] / Прейдибэн Катхирейвелу - режим доступа: http://onreader.mdl.ru/PythonNetworkProgrammingCookbook.2nd/content/index.html;

5. Руководство конфигурации проекта для работы с библиотекой SFML в среде Microsoft Visual Studio «SFML and Visual Studio» [Электронный ресурс] - режим доступа: https://www.sfml-dev.org/tutorials/2.5/start-vc.php

6. Уроки по графической библиотеке SFML [электронный ресурс] - режим доступа: https://ravesli.com/uroki-po-sfml/;

7. Документация по языку C++ [электронный ресурс] – режим доступа: https://ru.cppreference.com/w/, дата обращения: 10.04.2021;

ПРИЛОЖЕНИЕ 1

КОД ОСНОВНЫХ МОДУЛЕЙ ПРОЕКТА

Файл «main.cpp»

```
#include "include.h"

using namespace sf;
using std::string;
using std::to_string;

string get_str_in_char(char data[1024]) {
    string ans;
    for (int i = 0; i < 1024; i++)
        ans.push_back(data[i]);
    return ans;
}

string new_cmd(string s_ip, string s_p, string data) {
    int accept = 0;

    std::vector<sf::TcpSocket*> clients;
    sf::TcpSocket socket;

    sf::TcpListener listener;
    listener.setBlocking(false);


    sf::SocketSelector selector;

    const std::string address = s_ip;
    const int port = atoi(s_p.c_str());

    if (socket.connect(address, port) == sf::Socket::Done) {
        std::cout << "Connected to the server" << std::endl;
    }
    selector.add(socket);

    sf::Packet packet;
    packet << data;
    socket.send(packet);
    packet.clear();

    char data1[2048];
    sf::Packet packet2;


    sf::Clock clock, pause;
    float ftime = clock.getElapsedTime().asSeconds();
    clock.restart();
```

```
    std::size_t received;
    socket.receive(data1, sizeof(data1), received);
    string ans = get_str_in_char(data1);




    return ans;
}
int check_authorisation(string s_ip, string s_p, string data) {
    int accept = 0;

    std::vector<sf::TcpSocket*> clients;
    sf::TcpSocket socket;

    sf::TcpListener listener;
    listener.setBlocking(false);


    sf::SocketSelector selector;

    const std::string address = s_ip;
    const int port = atoi(s_p.c_str());

    if (socket.connect(address, port) == sf::Socket::Done) {
        std::cout << "Connected to the server" << std::endl;
    }
    selector.add(socket);

    sf::Packet packet;
    packet << data;
    socket.send(packet);
    packet.clear();

    char data1[2048];
    sf::Packet packet2;


    sf::Clock clock, pause;
    float ftime = clock.getElapsedTime().asSeconds();
    clock.restart();


    std::size_t received;
    socket.receive(data1, sizeof(data1), received);
    string ans = get_str_in_char(data1);

    accept = string(data1)[0] - '0';

    if (accept == 1)
        return 1;
```

```
    return 0;
}

std::vector<string> split(const string& str, const string& delim){
    std::vector<string> tokens;
    size_t prev = 0, pos = 0;
    do{
        pos = str.find(delim, prev);
        if (pos == string::npos) pos = str.length();
        string token = str.substr(prev, pos - prev);
        if (!token.empty()) tokens.push_back(token);
        prev = pos + delim.length();
    } while (pos < str.length() && prev < str.length());
    return tokens;
}

string repl(string data, string sub) {
    int i = 0;
    while (data[i] != NULL) {
        if (data[i] == sub) {
            data.erase(i, 1);
            if ((i - 1) >= 0) i--;
        }
        else i++;
    }
    return data;
}

string listen_cmd(string s_ip, string s_p, float hm) {
    sf::TcpListener listener;


    const std::string Address = s_ip;
    const int Port = atoi(s_p.c_str());
    listener.listen(Port);
    sf::TcpSocket client;
    listener.accept(client);


    char data[2048];
    std::size_t received;

    sf::Clock clock_listen;
    clock_listen.restart();
    string ans = "0";
    if (client.receive(data, sizeof(data), received) != sf::Socket::Done) {

        return ans;
    }
    else {
```

```
        data[received] = '\0';
        ans = std::string(data, data + received);

        const char out[] = "-1-";
        client.send(out, sizeof(out));
    }
    client.disconnect();
    listener.close();


    return ans;
}

void win_end(RenderWindow& window1,string Statistic) {

    sf::Image WindowIcon;
    if (!WindowIcon.loadFromFile("files/icon.png"))
        std::cout << "cant load texture from file " << std::endl;
    window1.setIcon(128, 128, WindowIcon.getPixelsPtr());
    window1.setMouseCursorVisible(true);

    //-----------

    sf::Font font;
    sf::Font& refFont = font;
    if (!font.loadFromFile("files/normal.otf")) {
        std::cout << "cant load font" << std::endl;
    }

    sf::Text TextBox_Statistic(Statistic, font, 36);
    TextBox_Statistic.setFillColor(sf::Color::Black);
    TextBox_Statistic.setStyle(sf::Text::Bold | sf::Text::Underlined);
    TextBox_Statistic.setPosition(50, 50);


    while (window1.isOpen()) {
        window1.clear(Color(200, 200, 200));
        sf::Event event;
        while (window1.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window1.close();
            if (event.type == sf::Event::KeyPressed)
                if (event.key.code == sf::Keyboard::Escape)
                    window1.close();
        }
        TextBox_Statistic.setString(Statistic);
        window1.draw(TextBox_Statistic);
        window1.display();
        window1.clear();
    }
}
```

```
int main() {
    setlocale(LC_ALL, "");

    struct Auth ath;
    ath.f.conn = false;
    ath.f.menu = false;
    ath.f.exit = false;
    std::string Statistic;
    while (ath.f.exit == false) {
        if (ath.f.conn == false) {
            RenderWindow window_connect(VideoMode(425, 363), "Chess: Connect");
            window_connect.setFramerateLimit(30);
            Auth at;
            at = Menu_Connect(window_connect, ath);
            ath = at;
        }
        else if (ath.f.menu == false && ath.f.exit == false) {
            RenderWindow window_menu(VideoMode(350, 480), "Chess: Menu");
            window_menu.setFramerateLimit(30);
            Auth at;
            at = Menu(window_menu, ath);
            ath = at;
        }
        else if (ath.f.exit == false) {
            RenderWindow window_game(VideoMode(1250, 900), "Chess: Game");
            window_game.setFramerateLimit(30);
            Auth at;
            if (ath.name_server == "Single")
                at = Game_single(window_game, ath);
            else
                at = Game(window_game, ath);
            ath = at;
            ath.f.menu = false;
        }

        if ((ath.who_win == -1 || ath.who_win == 1)) {
            if (ath.who_win == ath.whomai) {
                string answer = new_cmd(ath.ip, ath.port, "0000000|set_win|" + ath.L
ogin + "|" + ath.Password + "|" + ath.Login);
                std::vector<string> ans = split(answer, "-");
                sf::RenderWindow window1(sf::VideoMode(260, 130), "Chess: end game")
;
                Statistic = "Win";
                win_end(window1, Statistic);
            }
            else {
                sf::RenderWindow window1(sf::VideoMode(260, 130), "Chess: end game")
;
                Statistic = "End Game";
                win_end(window1, Statistic);
```

```
            }
            ath.who_win = -3;
        }

    }

    return 0;
}
```

Файл «server.py»

```
import socket
from threading import Thread

import sqlite3 as sql3
import random
import string
import sys


name_db=['users.db','rooms.db']


create_users="""
    CREATE TABLE USERS (
        id INTEGER NOT NULL ,
        name TEXT NOT NULL PRIMARY KEY,
        password TEXT,
        points INTEGER,
        wins INTEGER);"""
create_rooms="""
    CREATE TABLE ROOMS (
        id INTEGER NOT NULL ,
        name TEXT NOT NULL PRIMARY KEY,
        user1 TEXT,
        user1_ready INTEGER,
        user2 TEXT,
        user2_ready INTEGER,
        freedoom INTEGER,
        password_room TEXT);
"""
def init_db(name_db,cmd):
    con = sql3.connect(name_db)
    try:
        with con:
            con.execute(cmd)
    except:
        pass


init_db(name_db[0],create_users)
```

```
init_db(name_db[1],create_rooms)


def sql_cmd_select_all(name_db,name_table):
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        data = cur.execute("SELECT * FROM "+name_table)
        #data = cur.fetchone()
        for row in data:
            print(row)

def get_count(name_db,name_table):
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        try:
            cur.execute("SELECT COUNT(*) FROM "+name_table)
            data=cur.fetchone()
            return data[0]
        except:
            return 0

def add_data_user(name_db,name_table, data):
    i = get_count(name_db,name_table)
    data.insert(0,i)
    print(i)
    print(data)
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        cur.execute("INSERT INTO USERS VALUES(?,?, ?, ?, ?);", tuple(data))
        con.commit()
def add_data_room(name_db,name_table, data):
    i = get_count(name_db,name_table)
    data.insert(0,i)
    print(i)
    print(data)
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        cur.execute("INSERT INTO ROOMS VALUES(?,?,?,?,?,?,?,?,?);", tuple(data))
        con.commit()

def del_line(name_db,name_table, name):
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        cur.execute("DELETE FROM "+name_table+" WHERE name='"+name+"';")
        con.commit()
```

```
def find(name_db,name_table, name):
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        cur.execute("SELECT * FROM "+name_table+" WHERE name='"+name+"';")
        data = cur.fetchone()
        print("find=",data)
        return(data)


def Login(login, password):
    data=find('users.db',"USERS",login)
    if (data[1]==login and data[2]==password):
        print("Login: Ok")
        return "1"
    else:
        print("Login: No")
        return "0"

def get_all_stat(name_db):
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        cur.execute("SELECT name, points, wins FROM USERS ORDER BY points DESC LIMIT
 10")
        data = cur.fetchall()

        for row in data:
            print(row)
        return(data)

def create_room(name_db,Name_room, USER):
    nameroom=Name_room
    user1=USER
    user2=""
    freedoom=1
    passwd=""
    for i in range(5):
        passwd+=str(random.randint(100,999))
    data=[nameroom, user1,"0", user2,"0", freedoom, passwd]
    try:
        add_data_room(name_db,"ROOMS",data)
    except:
        return "0"
    ans=get_line_room(name_db,Name_room)
    if ans!=None or ans!=0 or ans!="":
        return "1"
    else:
        return "0"

def get_line_room(name_db,name_room):
```

```
    try:
        data=find(name_db,"ROOMS",name_room)
        print("line room=",data)
    except:
        print("er")
    return data

def find_room(name_db,Name_room, USER):
    ans=get_line_room(name_db,Name_room)
    print(ans[6])
    if ans!=None and ans!=0 and ans!="" and ans[6]==1:
        try:
            update_user2_freedom(name_db, [USER,0, Name_room])
            return "1"
        except:
            return "0"
    else:
        return "0"

def update_user2_freedom(name_db, data):
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        cur.execute('''UPDATE ROOMS SET user2=?, freedoom=? WHERE name=?''',(data[0]
,data[1],data[2]))
        con.commit()

def check_freedoom(name_db, data):
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        cur.execute("SELECT freedoom FROM ROOMS WHERE name='"+data+"';")
        data = cur.fetchone()
        print("check_freedoom=",data[0])
        return(str(data[0]))


def get_user(name_db, data,n):
    con = sql3.connect(name_db)
    if n=='1':
        with con:
            cur = con.cursor()
            cur.execute("SELECT user1 FROM ROOMS WHERE name='"+data+"';")
            data = cur.fetchone()
            print("get_user1=",data)
            return(data[0])
    elif n=='2':
        with con:
            cur = con.cursor()
            cur.execute("SELECT user2 FROM ROOMS WHERE name='"+data+"';")
            data = cur.fetchone()
```

```
            print("get_user2=",data)
            return(data[0])


def update_win_point(name_db, data):
    con = sql3.connect(name_db)
    with con:
        try:
            cur = con.cursor()
            cur.execute('''UPDATE USERS SET points=?, wins=? WHERE name=?''',(data[0
],data[1],data[2]))
            con.commit()
            return "1"
        except:
            return "0"

def get_data1(name_db, name_table,param,name):
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        cur.execute("SELECT "+param+" FROM "+name_table+" WHERE name='"+name+"';")
        data = cur.fetchone()
        return data

def statistic_u(name_db, name):
    con = sql3.connect(name_db)
    with con:
        cur = con.cursor()
        cur.execute("SELECT points, wins FROM USERS WHERE name='"+name+"';")
        data = cur.fetchall()
        #print("find=",data)
        return(data[0])

def set_win(name_db, data):
    ans=get_data1(name_db,"USERS","points",data[2])
    print("ans=",ans[0])
    data[0]+=int(ans[0])
    ans=get_data1(name_db,"USERS","wins",data[2])
    print("ans=",ans[0])
    data[1]+=int(ans[0])
    print("data=",data)
    return update_win_point(name_db, data)
def set_exp(name_db, data):
    con = sql3.connect(name_db)
    with con:
        try:
            cur = con.cursor()
            cur.execute('''UPDATE USERS SET points=? WHERE name=?''',(data[0],data[1
]))
            con.commit()
            return "1"
```

```
        except:
            return "0"

def set_ready(name_db, data):
    con = sql3.connect(name_db)
    with con:
        try:
            cur = con.cursor()
            cur.execute('''UPDATE ROOMS SET user1_ready=?, user2_ready=? WHERE name=
?''',(data[0],data[1],data[2]))
            con.commit()
            return "1"
        except:
            return "0"

def test():
    con = sql3.connect("users.db")
    with con:
        cur = con.cursor()
        cur.execute("SELECT name FROM USERS;")
        data=[]
        d="0"
        while(d!=None):
            d = cur.fetchone()
            if d!=None:
                d=d[0]
                data.append(d)
        print("check_freedoom=",data)
        for usr in data:
            set_exp("users.db",[50,usr])
        print("USERS")
        sql_cmd_select_all("users.db","USERS")

#-------------------------------------
#SERVER
#-------------------------------------

def send_data(connection, data):
    connection.send(data.encode('UTF-8'))


def get_data(connection):
    return connection.recv(1024).decode().strip()


def run(connection, addr_user):
    try:
        """
        while True:
            data = conn.recv(1024)
            print(data.decode('utf-8'))
```

```
        if not data:
            break
            """
    data = get_data(connection)
    data=data.split('|')
    print(addr_user, data)
    if data[1]=='login':
        answer=str(Login(data[2],data[3]))
    elif data[1]=='reg':
        cou=find('users.db',"USERS",data[2])
        if cou==None:
            add_data_user('users.db',"USERS",[data[2],data[3], 0, 0])
            answer="1"
        else:
            answer="0"
    else:
        answer=str(Login(data[2],data[3]))
        if answer=="1":
            if data[1]=="statistic":
                cou=find('users.db',"USERS",data[2])
                cou=list(cou)
                answer="-"+str(cou[3])+"-"+str(cou[4])+"-"
            if data[1]=="statistic_all":
                cou=get_all_stat("users.db")
                cou=list(cou)
                answer="-"+str(cou)+"-"
            if data[1]=="create_room":
                cou=create_room("rooms.db",data[4],data[5])
                answer="-"+str(cou)+"-"
            if data[1]=="find_room":
                cou=find_room("rooms.db",data[4],data[5])
                answer="-"+str(cou)+"-"
            if data[1]=="check_freedoom":
                cou=check_freedoom("rooms.db",data[4])
                answer="-"+str(cou)+"-"
            if data[1]=="get_user":
                cou=get_user("rooms.db",data[4],data[5])
                answer="-"+str(cou)+"-"
            if data[1]=="statistic_u":
                cou=statistic_u("users.db",data[4])
                cou="-"+str(cou[0])+"-"+str(cou[1])+"-"
                answer=cou
            if data[1]=="room_pass":
                ans=get_line_room("rooms.db",data[4])
                cou="-0-"
                us1=ans[2].split('=')[0]
                us2=ans[4].split('=')[0]
                print("us=",us1,"|",us2,"|",data[2])
                if str(data[2])==str(us1) or str(data[2])==str(us2):
                    print("us=",us1,"|",us2)
                    cou="-"+str(ans[7])+"-"
```

```
                    answer=cou
                if data[1]=="set_win":
                    cou=set_win("users.db",[5,1,data[4]])
                    answer="-"+str(cou)+"-"
                if data[1]=="get_ready":
                    ans="2"
                    line=get_line_room("rooms.db",data[4])
                    if data[5]=="-1":
                        if line[3]==1 and line[5]==0:
                            ans="1"
                        else:
                            ans="0"
                    elif data[5]=="1":
                        if line[3]==0 and line[5]==1:
                            ans="1"
                        else:
                            ans="0"
                    answer="-"+str(ans)+"-"
                if data[1]=="set_ready":
                    ans="2"
                    if data[5]=="-1":
                        ans=set_ready("rooms.db",["0","1",data[4]])
                    elif data[5]=="1":
                        ans=set_ready("rooms.db",["1","0",data[4]])
                    answer="-"+str(ans)+"-"
                if data[1]=="set_exp":
                    cou=set_exp("users.db",[data[5],data[4]])
                    answer="-"+str(cou)+"-"
            else:
                answer="0"

        print("answer=", answer)
        print("USERS")
        sql_cmd_select_all("users.db","USERS")
        print("ROOMS")
        sql_cmd_select_all("rooms.db","ROOMS")
        send_data(connection, answer)
    except Exception as e:
        print(e)
    finally:
        connection.close()


if __name__ == '__main__':
    if len(sys.argv)>2:
        HOST=sys.argv[1]
        PORT=int(sys.argv[2])
    else:
        HOST='localhost'
        PORT=5555
    print("Server started successfully")
```

```
list_ip=[]
sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
sock.bind((HOST, PORT))
sock.listen(50)

while True:
    connection, addr_user = sock.accept()
    list_ip.append([addr_user, connection])
    print('Connected by', addr_user)
    try:
        thread = Thread(
            target=run,
            args=(connection, addr_user),
            daemon=True)
        thread.start()
    except (Exception, KeyboardInterrupt) as e:
        print(e)
        connection.close()
        break
```