МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева (Самарский университет)»

Институт информатики и кибернетики

Кафедра программных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ**  
по дисциплине «Объектная распределенная обработка»

по теме «Распределенное клиент-серверное приложение «Игра «Крестики-нолики»

Выполнил:

обучающийся группы №6415-020302D Д.А. Борисов

Проверил:

руководитель работы,

доцент кафедры ПС, к.т.н., доцент О.А. Гордеева

Самара 2024

ЗАДАНИЕ

Написать распределённое клиент-серверное приложение, используя технологию сокетных соединений. Приложение реализует игру «Крестики-нолики» со следующими правилами:

игра будет предоставлять возможность играть как человеку, так и компьютеру;

игра начнется с хода игрока-человека, после чего ход переходит к компьютеру или другому игроку в зависимости от выбранного режима игры;

размер игрового поля можно задавать, поле должно быть квадратным;

игроки по очереди занимают ячейку поля своим знаком с целью выстроить длинную последовательность (горизонтальную, вертикальную или диагональную) и помешать сделать то же оппоненту;

если на игровом поле заполнены все клетки и победитель не определен, игра заканчивается ничьей;

при наличии победителя или ничьей выводится соответствующее сообщение, после чего предлагается начать новую партию с теми же или новыми параметрами или завершить игру.

РЕФЕРАТ

Пояснительная записка 29 с, 10 рисунков, 1 таблица, 10 источников,  
1 приложение.

КРЕСТИКИ-НОЛИКИ, ИГРА, КЛЕТКА, ИГРОВОЕ ПОЛЕ, ПОБЕДА, НИЧЬЯ, ИГРОК, КОМПЬЮТЕР, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, СОКЕТ.

Во время курсовой работы разработаны алгоритмы и соответствующая им программа игры «Крестики-нолики». Визуализация и игровой процесс происходят на клиентской части, а логика игры – на серверной части.

Программа написана на языке Java с помощью среды разработки Visual Studio Code и функционирует под управлением операционной системы Windows 7 и выше.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc167695421)

[1 Описание и формализация предметной области 6](#_Toc167695422)

[1.1 Описание игры «Крестики-нолики» 6](#_Toc167695423)

[1.2 Постановка задачи 7](#_Toc167695424)

[2 Проектирование системы 9](#_Toc167695425)

[2.1 Описание используемых технологий 9](#_Toc167695426)

[2.2 Структурная схема системы 11](#_Toc167695427)

[2.3 Диаграмма компонентов 12](#_Toc167695428)

[2.4 Диаграмма вариантов использования 13](#_Toc167695429)

[2.5 Диаграмма деятельности 14](#_Toc167695430)

[2.6 Диаграмма последовательности 15](#_Toc167695431)

[3 Реализация системы 17](#_Toc167695432)

[3.1 Разработка и описание интерфейса пользователя 17](#_Toc167695433)

[ЗАКЛЮЧЕНИЕ 19](#_Toc167695434)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc167695435)

[ПРИЛОЖЕНИЕ А Листинг программы 21](#_Toc167695436)

ВВЕДЕНИЕ

Крестики-нолики — логическая игра между двумя противниками на квадратном поле 3 на 3 клетки или большего размера (вплоть до «бесконечного поля»). Один из игроков играет «крестиками», второй — «ноликами».

Игроки по очереди ставят на свободные клетки поля знаки (один всегда крестики, другой всегда нолики). Первый, выстроивший в ряд свои фигуры по вертикали, горизонтали или большой диагонали, выигрывает. Если игроки заполнили все ячейки и оказалось, что ни в одной вертикали, горизонтали или большой диагонали нет одинаковых знаков, партия считается закончившейся в ничью. Первый ход делает игрок, ставящий крестики.

Обычно по завершении партии выигравшая сторона зачёркивает чертой свои знаки (нолики или крестики), составляющие сплошной ряд [1].

Во время курсовой работы необходимо написать распределённое клиент-серверное приложение «Игра «Крестики-нолики», с помощью которого игрок сможет соревноваться в крестики-нолики с компьютером или с другом.

При проектировании системы будут использоваться методология ООАП (Object-Oriented Analysis/Design), в основу которой положена объектно-ориентированная методология представления предметной области в виде объектов, являющихся экземплярами соответствующих классов, и язык моделирования UML (Unified Modeling Language), который является стандартным инструментом для разработки «чертежей» программного обеспечения [2].

Описание и формализация предметной области

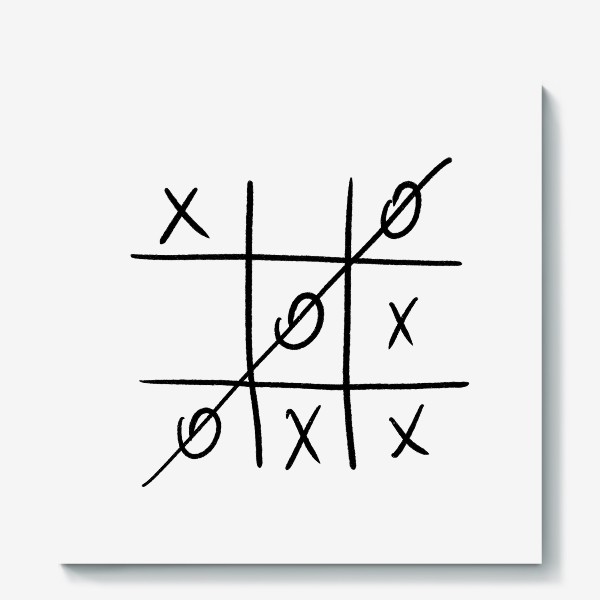
Предметная область – множество всех предметов, свойства и отношения которых рассматриваются в научной теории. Предметная область не только определяет множество объектов, рассматриваемых в пределах одного рассуждения или в научной теории, но также их свойства, отношения, функциональные связи [3].

Описание игры «Крестики-нолики»

Крестики-нолики — это простая настольная игра для двух игроков. Игра ведется на квадратном поле, обычно размером 3x3. Один игрок использует знак "крестик", другой — "нолик". Игроки по очереди ставят свои знаки в свободные клетки поля, стремясь разместить три своих знак подряд — по горизонтали, вертикали или диагонали. Первый, кому это удается, выигрывает партию. Если все клетки заполнены, а ни один из игроков не выстроил три символа подряд, игра заканчивается ничьей.

Несмотря на свою простоту, крестики-нолики предоставляют широкий простор для стратегического мышления. Опытные игроки могут предвидеть ходы соперника и блокировать их попытки выиграть, что часто приводит к ничьей. Кроме того, она может быть реализована в виде компьютерных программ, что позволяет использовать её для изучения основ программирования и искусственного интеллекта.

На рисунке 1 представлен пример завершенной партии.

  
Рисунок 1 – Пример завершенной партии в игре «Крестики-нолики»

Постановка задачи

Во время курсовой работы написать распределённое клиент-серверное приложение «Игра «Крестики-нолики», с помощью которого пользователь может играть как против компьютера, так и против другого игрока.

Правила игры:

игра будет предоставлять возможность играть как человеку, так и компьютеру;

игра начнется с хода игрока-человека, после чего ход переходит к компьютеру или другому игроку в зависимости от выбранного режима игры;

размер игрового поля можно задавать, поле должно быть квадратным;

игроки по очереди занимают ячейку поля своим знаком с целью выстроить длинную последовательность (горизонтальную, вертикальную или диагональную) и помешать сделать то же оппоненту;

если на игровом поле заполнены все клетки и победитель не определен, игра заканчивается ничьей;

при наличии победителя или ничьей выводится соответствующее сообщение, после чего предлагается начать новую партию с теми же или новыми параметрами или завершить игру.

Входные параметры:

выбор режима игры (игрок против игрока, игрок против компьютера);

выбор размера игрового поля;

выбор клетки для установки символа.

Выходные параметры:

вывод на экран игрового поля;

вывод сообщений о текущем состоянии игры (ход игрока, победа, ничья);

вывод на экран ошибок или предупреждений в случае некорректного ввода или других проблем во время игры.

Ограничения:

* количество режимов игры – 2 (человек и компьютер, человек и человек);
* размер игрового поля – 3 (3х3, 4х4, 5х5);
* максимальное количество партий в одной игре - неограниченно.

Функции сервера:

создание игрового поля;

проверка победы;

проверка ничьей;

сброс доски;

Функции клиента:

запуск игры;

визуализация игрового процесса.

Технология распределенной разработки – сокетные соединения.

Проектирование системы

Проектирование программного обеспечения – это процесс определения архитектуры, компонентов, интерфейсов, других характеристик системы и конечного состава программного продукта.

Описание используемых технологий

Для реализации программного приложения в курсовой работе используются сокетные соединения.

Сокет — это программная (логическая) конечная точка, устанавливающая двунаправленную коммуникацию между сервером и одной или несколькими клиентскими программами. Прикладное программное обеспечение определяет сокет так, чтобы он использовал порты на основном компьютере для его реализации. Это позволяет программистам комфортно работать с низкоуровневыми деталями сетевых коммуникаций, такими как порты, маршрутизация и т. д., внутри прикладного кода.

Всего существует 2 типа сокетов: TCP и UDP. В данной работе используется TCP-сокет.

TCP-сокет устанавливает связь между клиентом и сервером в несколько этапов. На рисунке 2 представлена блок-схема коммуникации TCP-сокета.

Socket()  –  на сервере создается конечная точка для коммуникации.

Bind()  –  сокету присваивается уникальный номер и для него резервируется уникальная комбинации IP-адреса и порта.

Listen()  –  после создания сокета сервер ожидает подключения клиента.

Accept()  –  сервер получает запрос на подключение от клиентского сокета.

Connect()  –  клиент и сервер соединены друг с другом.

Send()/Recieve()  –  обмен данными между клиентом и сервером.

Close()  –  после обмена данными сервер и клиент разрывают соединение [4].

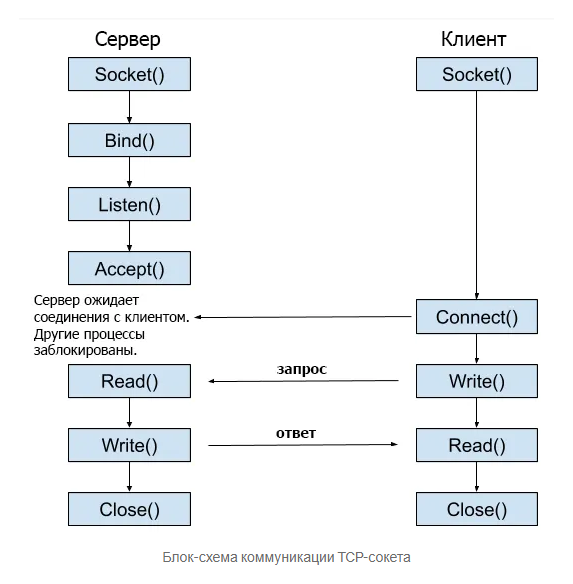


Рисунок 2 – Блок-схема коммуникации TCP-сокета

Для отрисовки пользовательского интерфейса приложения используется Java FX.

JavaFX представляет собой инструментарий для создания кроссплатформенных графических приложений на платформе Java.

JavaFX предоставляет большие возможности по сравнению с рядом других подобных платформ, в частности, по сравнению со Swing. Это и большой набор элементов управления, и возможности по работе с мультимедиа, двухмерной и трехмерной графикой, декларативный способ описания интерфейса с помощью языка разметки FXML, возможность стилизации интерфейса с помощью CSS и многое другое. [5].

Структурная схема системы

Приложение будет разработано в соответствии со структурным подходом к проектированию, заключающемся в декомпозиции системы на функциональные подсистемы, подсистемы – на подфункции, подфункции разбиваются на задачи и так далее. Структурное проектирование – это метод определения подсистем, компонентов и способов их соединения, задающий ограничения, при которых система должна функционировать. При этом система сохраняет целостность и взаимосвязь всех компонентов, которые иерархически упорядочиваются [6].

На рисунке 3 приведена структурная схема разрабатываемой системы, разделяется клиентскую и серверную часть. Взаимодействие между ними осуществляется через сокет.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описаниеРисунок 3 – Структурная схема системы

В состав клиентской части входит:

подсистема взаимодействия с сервером, которая осуществляет установку соединения с сервером, формирование и отправку запросов;

подсистема визуализации, которая отображает игровое поле и результаты действий пользователя на экран;

подсистема ввода-вывода, отвечающая за ввод информации пользователем и вывод на экран пользователя;

подсистема вывода результата игры, отвечающая за вывод информации о состоянии текущего хода игрока;

подсистема настроек игры, которая позволяет пользователю выбрать размер игрового поля и режим игры.

подсистема выполнения хода.

В состав серверной части входит:

подсистема взаимодействия с клиентом, которая осуществляет приём данных с клиента и передачу их на обработку;

подсистема логики игры, которая отвечает за соответствие хода игрока правилам, а также за сам игровой процесс.

Диаграмма компонентов

Диаграмма компонентов предоставляет возможность описать структуру программы, определить компоненты, их взаимосвязи и зависимости. Она является инструментом для перехода от абстрактного логического представления системы к ее физической реализации.

Основные элементы диаграммы включают в себя компоненты, интерфейсы и связи между ними. Иногда на диаграмме также могут быть отображены ключевые классы, входящие в состав компонентов.

Под компонентом в данном контексте понимается физическая часть системы, которая должна иметь определенный набор интерфейсов для взаимодействия с другими компонентами [7].

На рисунках 4 представлена диаграмма компонентов системы.

Изображение выглядит как текст, диаграмма, линия, План

Автоматически созданное описание  
Рисунок 4 – Диаграмма компонентов

В таблице 1 приведено описание диаграммы.

1. Таблица 1 – Описание компонентов диаграммы

|  |  |
| --- | --- |
| Название | Назначение |
| SettingsController | Стартовый экран игры, на котором происходят настройки игровой сессии |
| GameScreenController | Основной экран игры, в котором происходит игровой процесс |
| GameController | Компонент, отвечающий за выполнение хода |
| App | Класс, предназначенный для визуализации и отправки запросов от игрока серверу |
| Server | Класс, отвечающий за получение запросов от клиента |
| Game | Компонент, отвечающий за логику игры |
| Socket | Класс, предназначенный для взаимодействия сервера и клиента |

Диаграмма вариантов использования

Диаграмма вариантов использования представляет собой наиболее общую концептуальную модель сложной системы, которая является исходной для построения всех остальных диаграмм. На ней изображаются отношения между актерами и вариантами использования [8].

На рисунке 5 изображена диаграмма вариантов использования.

Изображение выглядит как текст, диаграмма, круг, Шрифт

Автоматически созданное описание   
Рисунок 5 – Диаграмма вариантов использования

Перед тем, как начать игру, пользователю необходимо настроить параметры игровой сессии. Для этого ему нужно выбрать размер игрового поля и режим игры

Диаграмма деятельности

Диаграмма деятельности − это графическое представление процессов и задач, выполняемых в рамках определенного проекта или деятельности. Эта диаграмма используется для анализа и оптимизации бизнес-процессов, планирования работы, принятия решений и управления проектами. Диаграмма деятельности состоит из набора символов и линий, каждый из которых имеет свое значение и назначение [9].

На рисунке 6 изображена диаграмма деятельности.

Изображение выглядит как текст, диаграмма, чек, зарисовка

Автоматически созданное описаниеРисунок 6 – Диаграмма деятельности

Диаграмма последовательности

Диаграмма последовательности − UML-диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие акторов (действующих лиц) информационной системы в рамках прецедента [10].

На рисунке 7 изображена диаграмма последовательности.

Изображение выглядит как текст, чек, снимок экрана, Параллельный

Автоматически созданное описаниеРисунок 7 – Диаграмма последовательности

Реализация системы

Разработка и описание интерфейса пользователя

На рисунке 8 изображена стартовая экранная форма приложения. Данная форма открывается при запуске приложения. Здесь пользователь может выбрать размер игрового поля и режим игры.

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Автоматически созданное описание

Рисунок 8 – Стартовая экранная форма

После выбора параметров игры, игрок переходит на экран игры. Здесь пользователь выбирает клетку, куда поставить крестик, далее сразу отображается ход компьютера или другого игрока (в зависимости от выбранного режима игры). Экранная форма игрового поля представлена на рисунке 9.

Изображение выглядит как снимок экрана, Прямоугольник, прямоугольный, текст

Автоматически созданное описание

Рисунок 9 – Экранная форма игры

В случае победы или ничьей на экран выводится форма с соответствующим сообщением и предложением начать новую игру со старыми или новыми параметрами. Форма представлена на рисунке 10.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 10 – Экранная форма с сообщением о завершении игры

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсовой работы было разработано распределённое клиент-серверное приложение «Игра «Крестики-нолики» с использованием сокетных соединений.

Сначала был проведен анализ предметной области, рассмотрены правила игры «Крестики-нолики». Затем на основе проведенного анализа была сформулирована постановка задачи.

В следующем разделе были рассмотрены технологии, с помощью которых была проведена разработка приложения. Также была построена структурная схема системы, описана диаграмма компонентов и разработан информационно-логический проект по методологии UML, состоящий из диаграмм вариантов использования, деятельности, последовательности.

В последнем разделе были описаны экранные формы системы в соответствии с диаграммой деятельности.

Разработанная система может использоваться в рамках развлекательной деятельности для игры в «Крестики-нолики».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Крестики-нолики — Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Крестики-нолики (дата обращения: 08.05.2024).
2. Буч Г., Рамбо Д., Якобсон А. Язык UML. Руководство пользователя. Изд. 2-е. М.: ДМК Пресс, 2006. 546 с.
3. Предметная область [Электронный ресурс] // Википедия: [сайт]. URL: https://ru.wikipedia.org/wiki/Предметная\_область (дата обращения: 08.05.2024).
4. Основы программирования TCP-сокетов на Java [Электронный ресурс]. URL: https://medium.com/nuances-of-programming/основы-программирования-tcp-сокетов-на-java-594b6519d494 (дата обращения: 08.05.2024).
5. Введение в Java FX [Электронный ресурс] // Metanit: [сайт]. URL: https://metanit.com/java/javafx/1.1.php (дата обращения: 08.05.2024).
6. Структурный подход к проектированию ИС [Электронный ресурс]. URL: https://studfile.net/preview/2704746/page:7/ (дата обращения: 08.05.2024).
7. Диаграмма объектов предметной области [Электронный ресурс] URL: https://studfile.net/preview/2202888/page:4/ (дата обращения: 08.05.2024).
8. Диаграмма состояний (UML) – Википедия [Электронный ресурс] // Википедия: [сайт]. URL: https://ru.wikipedia.org/wiki/Диаграмма\_ состояний\_(UML) (дата обращения: 08.05.2024).
9. Диаграмма деятельности UML [Электронный ресурс]. URL: https://itonboard.ru/analysis/664 (дата обращения: 08.05.2024).
10. Диаграмма последовательности (UML) – Википедия [Электронный ресурс] // Википедия: [сайт]. URL: https://ru.wikipedia.org/wiki/ Диаграмма\_последовательности (дата обращения: 08.05.2024).

ПРИЛОЖЕНИЕ А  
Листинг программы

Server.java

package server;

import java.io.*\**;

import java.net.ServerSocket;

import java.net.Socket;

public class Server {

    private static final int PORT = 2715;

    private static int boardSize = 3;

    private static char[][] board;

    private static char currentPlayer = 'X';

    public static void main(String[] args) {

        try {

            ServerSocket serverSocket = new ServerSocket(PORT);

            System.out.println("Server started on port " + PORT + ". Waiting for clients...");

            Socket clientSocket = serverSocket.accept();

            System.out.println("Client connected.");

            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            while (true) {

                String[] message = in.readLine().split(" ");

                if (message[0].equals("SIZE")) {

                    boardSize = Integer.parseInt(message[1]);

                    board = new char[boardSize][boardSize];

                    currentPlayer = 'X';

                    initializeBoard();

                    out.println("SIZE " + boardSize);

                } else if (message[0].equals("MOVE")) {

                    if (board == null) {

                        board = new char[boardSize][boardSize];

                        initializeBoard();

                    }

                    int row = Integer.parseInt(message[2]);

                    int col = Integer.parseInt(message[3]);

                    if (isValidMove(row, col)) {

                        board[row][col] = currentPlayer;

                        out.println("MOVE " + currentPlayer + " " + row + " " + col);

                        if (isWinner(currentPlayer)) {

                            out.println("WIN " + currentPlayer);

                            resetBoard();

                        } else if (isBoardFull()) {

                            out.println("DRAW");

                            resetBoard();

                        } else {

                            currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';

                        }

                    } else {

                        out.println("INVALID\_MOVE");

                    }

                }

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    private static void initializeBoard() {

        for (int i = 0; i < boardSize; i++) {

            for (int j = 0; j < boardSize; j++) {

                board[i][j] = '\0';

            }

        }

    }

    private static boolean isValidMove(int row, int col) {

        return row >= 0 && row < boardSize && col >= 0 && col < boardSize && board[row][col] == '\0';

    }

    private static boolean isBoardFull() {

        for (int i = 0; i < boardSize; i++) {

            for (int j = 0; j < boardSize; j++) {

                if (board[i][j] == '\0') {

                    return false;

                }

            }

        }

        return true;

    }

    private static boolean isWinner(char player) {

        for (int i = 0; i < boardSize; i++) {

            if (checkLine(player, board[i])) return true;

            char[] col = new char[boardSize];

            for (int j = 0; j < boardSize; j++) {

                col[j] = board[j][i];

            }

            if (checkLine(player, col)) return true;

        }

        char[] mainDiagonal = new char[boardSize];

        char[] antiDiagonal = new char[boardSize];

        for (int i = 0; i < boardSize; i++) {

            mainDiagonal[i] = board[i][i];

            antiDiagonal[i] = board[i][boardSize - 1 - i];

        }

        if (checkLine(player, mainDiagonal)) return true;

        return checkLine(player, antiDiagonal);

    }

    private static boolean checkLine(char player, char[] line) {

        if (player == '\0') return false;

        for (char c : line) {

            if (c != player) return false;

        }

        return true;

    }

    private static void resetBoard() {

        board = new char[boardSize][boardSize];

        currentPlayer = 'X';

        initializeBoard();

    }

}

App.java

package tictactoe\_test;

import javafx.application.Application;

import javafx.application.Platform;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.*\**;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import java.io.*\**;

import java.net.Socket;

import java.util.Random;

import javafx.animation.PauseTransition;

import javafx.util.Duration;

public class App extends Application {

    private char currentPlayer = 'X';

    private Button[][] buttons;

    private Socket socket;

    private BufferedReader in;

    private PrintWriter out;

    private int boardSize = 3;

    private boolean playerVsComputer = false;

    private char[][] board;

    private static boolean isGameFinished = false;

    private Stage gameStage;

    private int port = 2715;

    private boolean isFirstGame = true;

    @Override

    public void start(Stage primaryStage) {

        showSettingsDialog();

    }

    private void showSettingsDialog() {

        Stage settingsStage = new Stage();

        settingsStage.setTitle("Tic-tac-toe: New game");

        VBox vbox = new VBox(10);

        vbox.setAlignment(Pos.CENTER);

        Label sizeLabel = new Label("Choose board size:");

        ComboBox<String> sizeComboBox = new ComboBox<>();

        sizeComboBox.getItems().addAll("3x3", "4x4", "5x5");

        sizeComboBox.setValue("3x3");

        Label modeLabel = new Label("Choose game mode:");

        ComboBox<String> modeComboBox = new ComboBox<>();

        modeComboBox.getItems().addAll("Player vs Player", "Player vs Computer");

        modeComboBox.setValue("Player vs Player");

        Button startButton = new Button("Start Game");

        startButton.setOnAction(e -> {

            String size = sizeComboBox.getValue();

            switch (size) {

                case "4x4":

                    boardSize = 4;

                    break;

                case "5x5":

                    boardSize = 5;

                    break;

                default:

                    boardSize = 3;

            }

            playerVsComputer = modeComboBox.getValue().equals("Player vs Computer");

            settingsStage.close();

            initializeGame();

        });

        Button closeButton = new Button("Exit");

        closeButton.setOnAction(e -> {

            settingsStage.close();

        });

        vbox.getChildren().addAll(sizeLabel, sizeComboBox, modeLabel, modeComboBox, startButton, closeButton);

        Scene scene = new Scene(vbox, 300, 200);

        settingsStage.setScene(scene);

        settingsStage.show();

    }

    private void initializeGame() {

        if (!isFirstGame) {

            out.println("SIZE " + boardSize);

        }

        gameStage = new Stage();

        GridPane pane = new GridPane();

        buttons = new Button[boardSize][boardSize];

        board = new char[boardSize][boardSize];

        for (int i = 0; i < boardSize; i++) {

            for (int j = 0; j < boardSize; j++) {

                Button button = new Button();

                button.setPrefSize(100, 100);

                final int row = i;

                final int col = j;

                button.setOnAction(e -> handleButtonClick(button, row, col));

                buttons[i][j] = button;

                pane.add(button, j, i);

            }

        }

        gameStage.setOnCloseRequest(e -> {

            try {

                if (socket != null && !socket.isClosed()) {

                    socket.close();

                }

            } catch (IOException ex) {

                ex.printStackTrace();

            }

        });

        Scene scene = new Scene(pane, 100 \* boardSize, 100 \* boardSize);

        gameStage.setScene(scene);

        gameStage.setTitle("Tic-tac-toe");

        gameStage.show();

        if (isFirstGame) {

            System.out.println("Starting server");

            startServer();

        }

    }

    private void handleButtonClick(Button button, int row, int col) {

        if (button.getText().isEmpty() && (currentPlayer == 'X' || !playerVsComputer)) {

            button.setText(String.valueOf(currentPlayer));

            board[row][col] = currentPlayer;

            out.println("MOVE " + currentPlayer + " " + row + " " + col);

            if (isPlayerWon(currentPlayer, row, col) || isGameFinished) {

                return;

            }

            currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';

        }

        if (playerVsComputer && currentPlayer == 'O' && !isBoardFull()) {

            computerMove();

        }

    }

    private void computerMove() {

        Random rand = new Random();

        int row, col;

        do {

            row = rand.nextInt(boardSize);

            col = rand.nextInt(boardSize);

        } while (board[row][col] != '\0');

        out.println("MOVE O " + row + " " + col);

        buttons[row][col].setText("O");

        board[row][col] = 'O';

        currentPlayer = 'X';

    }

    private void startServer() {

        new Thread(() -> {

            try {

                socket = new Socket("localhost", port);

                System.out.println("Connected to server.");

                in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

                out = new PrintWriter(socket.getOutputStream(), true);

                out.println("SIZE " + boardSize);

                while (true) {

                    String response = in.readLine();

                    handleServerResponse(response);

                }

            } catch (IOException e) {

                e.printStackTrace();

            }

        }).start();

    }

    private void handleServerResponse(String response) {

        System.out.println(response);

        if (response.startsWith("MOVE")) {

            Platform.runLater(() -> {

                String[] parts = response.split(" ");

                char player = parts[1].charAt(0);

                int row = Integer.parseInt(parts[2]);

                int col = Integer.parseInt(parts[3]);

                buttons[row][col].setText(String.valueOf(player));

                board[row][col] = player;

*// currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';*

            });

        } else if (response.startsWith("WIN") || response.startsWith("DRAW")) {

            isGameFinished = true;

            Platform.runLater(() -> showEndGameDialog(response));

        }

    }

    private void showEndGameDialog(String message) {

        Alert alert = new Alert(Alert.AlertType.INFORMATION);

        alert.setTitle("Game over!");

        if (message.contains("DRAW")) {

            alert.setHeaderText("It's a draw!");

        } else {

            String[] winMessage = message.split(" ");

            alert.setHeaderText("Player \"" + winMessage[1] + "\" wins!");

        }

        ButtonType replayButton = new ButtonType("Replay");

        ButtonType newGameButton = new ButtonType("New Game");

        ButtonType closeButton = new ButtonType("Exit", ButtonBar.ButtonData.CANCEL\_CLOSE);

        alert.getButtonTypes().setAll(replayButton, newGameButton, closeButton);

        alert.showAndWait().ifPresent(type -> {

            if (type == replayButton) {

                resetGame();

                System.out.println("Game restarted");

            } else if (type == newGameButton) {

                resetGame();

                showSettingsDialog();

                if (gameStage != null) {

                    gameStage.close();

                }

                System.out.println("New game");

            } else if (type == closeButton) {

                if (gameStage != null) {

                    gameStage.close();

                }

            }

        });

    }

    private void resetGame() {

        isFirstGame = false;

        isGameFinished = false;

        currentPlayer = 'X';

        board = new char[boardSize][boardSize];

        Platform.runLater(() -> {

            for (int i = 0; i < boardSize; i++) {

                for (int j = 0; j < boardSize; j++) {

                    buttons[i][j].setText("");

                }

            }

        });

        System.out.println("Player vs Computer: " + playerVsComputer);

        System.out.println("Is first game: " + isFirstGame);

        out.println("SIZE " + boardSize);

    }

    private boolean isBoardFull() {

        for (int i = 0; i < boardSize; i++) {

            for (int j = 0; j < boardSize; j++) {

                if (board[i][j] == '\0') {

                    return false;

                }

            }

        }

        isGameFinished = true;

        return true;

    }

    private boolean isPlayerWon(char player, int row, int col) {

*// Проверка строки*

        for (int i = 0; i < boardSize; i++) {

            if (board[row][i] != player) {

                break;

            }

            if (i == boardSize - 1) {

                return true;

            }

        }

*// Проверка столбца*

        for (int i = 0; i < boardSize; i++) {

            if (board[i][col] != player) {

                break;

            }

            if (i == boardSize - 1) {

                return true;

            }

        }

*// Проверка главной диагонали*

        if (row == col) {

            for (int i = 0; i < boardSize; i++) {

                if (board[i][i] != player) {

                    break;

                }

                if (i == boardSize - 1) {

                    return true;

                }

            }

        }

*// Проверка побочной диагонали*

        if (row + col == boardSize - 1) {

            for (int i = 0; i < boardSize; i++) {

                if (board[i][(boardSize - 1) - i] != player) {

                    break;

                }

                if (i == boardSize - 1) {

                    return true;

                }

            }

        }

        return false;

    }

    public static void main(String[] args) {

        launch();

    }

}