

**Оглавление**

[**Введение** 5](#_Toc136901540)

[**1.** **Постановка задачи** 7](#_Toc136901541)

[**2.** **Выбор решения** 8](#_Toc136901542)

[**3.** **Описание программы** 9](#_Toc136901543)

[**4.** **Описание способа реализации пользовательского интерфейса** 10](#_Toc136901544)

[**5.** **Описание результатов работы программы** 11](#_Toc136901545)

[**Заключение** 15](#_Toc136901546)

[**Литература** 16](#_Toc136901547)

[**Приложение А. Листинг программы** 17](#_Toc136901548)

[**Приложение А.1. Файл «Server.java» Server.** 17](#_Toc136901549)

[**Приложение А.2. Файл «ClientHandler.java» Server.** 18](#_Toc136901550)

[**Приложение А.3. Файл «Form.java» Client.** 20](#_Toc136901551)

[**Приложение А.4. Файл «GameFieldPanel.java» Client.** 22](#_Toc136901552)

[**Приложение А.5. Файл «ImageLoader.cs» Client.** 28](#_Toc136901553)

[**Приложение Б. UML-диаграммы** 30](#_Toc136901554)

[Приложение Б.1. UML – диаграмма деятельности 30](#_Toc136901555)

[Приложение Б.2. UML – диаграмма вариантов использования приложения. 31](#_Toc136901556)

[Приложение Б.3. UML – диаграмма классов. 32](#_Toc136901557)

[Приложение Б.4. UML – диаграмма последовательности. 33](#_Toc136901558)

[Приложение Б.5. UML – диаграмма развёртывания. 34](#_Toc136901559)

**Введение**

**Четыре в ряд** – усложнённая версия классического жанра компьютерных игр «три в ряд», где вместо последовательности из трёх и более элементов, необходима последовательность из четырёх и более.

**Три в ряд** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) match-three) — [жанр](https://ru.wikipedia.org/wiki/%D0%96%D0%B0%D0%BD%D1%80_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%BE%D0%B9_%D0%B8%D0%B3%D1%80%D1%8B) [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0). Игры этого жанра характеризуются тем, что их игровой мир состоит из таблицы или сетки элементов, а задачей игрока является манипулирование элементами таким образом, чтобы совпали заданные игрой шаблонные комбинации, и после выполнения условия собранные элементы исчезают. Характерным представителем этого жанра является выпущенная в 2001 году игра Bejeweled, основанная на игре 1994 года «Шарики», а сама история появления элементов жанра прослеживается до «Тетриса» и Chain Shot!  (англ.)[рус.](https://ru.wikipedia.org/w/index.php?title=Chain_Shot!&action=edit&redlink=1), изданных в 1985 году. Начиная с середины 2000-х годов представители жанра «три в ряд» чаще всего относятся к [казуальным играм](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%B7%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), которые распространяются через [цифровую дистрибуцию](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B4%D0%B8%D1%81%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%86%D0%B8%D1%8F).

Смысл игры незамысловат. Классический вариант подразумевает простое манипулирование элементами, перемещая их по клеткам в любую сторону, меняясь при этом местами с элементом, стоящим в той самой клетке. Единственным и основным условием перестановки является то, что после перестановки элементов один из них должен образовать комбинацию из трёх и более схожих элементов (рисунок, цвет, фигура). Комбинацией считается последовательность элементов, лежащая в таблице либо по вертикали, либо по горизонтали, но никак не по диагонали. После образования комбинации она удаляется с поля и образуются пустые ячейки, на места которых спускаются элементы, стоящие выше. Таким образом игра может быть бесконечной.

Со временем разработчики стали модернизировать механику своих игр, добавляя новые условия и препятствия. Начиная ограничением перестановок и заканчивая добавлением особенных элементов, которые нужно уничтожить специальным алгоритмом. Данный жанр и по сей день успешно развивается и имеет большой спрос среди игроков.

1. **Постановка задачи**

Необходимо разработать игру жанра «три в ряд» через взаимодействие клиента и сервера для записи актуальных результатов игроков и вывода общей таблицы.

Функции клиента:

* Обеспечение интерфейса с пользователем
* Взаимодействие с сервером по обмену данных

Функции сервера:

* Взаимодействие с программами-клиентами

Используется в разработке среда IntelliJ IDEA Community Edition 2022, язык программирования Java и операционная система Windows 10.

1. **Выбор решения**

Важной частью курсовой работы является передача сообщений от клиента к серверу. Для реализации этого используются TCP-сокеты. TCP гарантирует доставку пакетов, их очередность, автоматически разбивает данные на пакеты и контролирует их передачу, в отличии от UDP. TCP — это ориентированный на соединения протокол, предназначенный для обеспечения надежной передачи данных между процессами, выполняемыми или на одном и том же компьютере, или на разных компьютерах. Термин "ориентированный на соединения" означает, что два процесса или приложения прежде, чем обмениваться какими-либо данными должны установить TCP-соединение.

Протокол TCP был выбран так как нам необходимо гарантировать доставку данных. Для взаимодействия между компьютерами используются адреса и порты. Адрес представляет собой 32-битную структуру. Номер порта – целое число от 0 до 65535.

1. **Описание программы**

Разработанная программа основывается на вышеописанных протоколах и состоит из клиента и сервера.

Данная программа основана на соединении между сервером и клиентов через протокол TCP. При запуске клиентской программы игрок попадает непосредственно в главное меню, где изображены приветственные надписи, поле ввода и 2 кнопки. Поле ввода предназначено для указания своего имени (никнейма) для дальнейшего прикрепления и сохранения результатов. Первая кнопка в главном меню запускает главный фрейм непосредственно с нашей игрой, а вторая кнопка закрывает главное меню и, следовательно, игру. Попав на игровое поле, в первую очередь, мы увидим таблицу, состоящую из различных элементов, которые и представляют собой игру. Также сверху показано количество возможных перестановок местами, а снизу отображается игровой счёт пользователя. По истечении ходов игра выводит конечный результат и завершается.

Параллельно с этим запускаем сервер и создаем сокет. Устанавливаем локальную точку для прослушивания подключений. Запускаем прослушивание входящих подключений и ожидаем новых клиентов в бесконечном цикле. После того, как клиент был принят, он передается на обработку новой нить в функцию обработки клиентов. После того как игра была завершена, результат игрока отправляется на сервер, где хранится база данных всех игроков. Там он сравнивается со всеми результатами и вписывается список. После этого обратно посылается клиенту для отображения общей таблицы всех результатов. В случае если вдруг сервер не запущен, выведется в таблицу только последний результат. Чтобы закрыть игру можно нажать на крестик на нашем окне.

1. **Описание способа реализации пользовательского интерфейса**

Пользовательский интерфейс присутствует только в клиентской части приложения с использованием Java Swing. Реализован он путём нескольких всплывающих окон и главное фрейма. Первое всплывающее окно это главное меню с полем ввода и двумя кнопками, описанными в предыдущем пункте. Далее идёт главный фрейм. В его основе лежит компонент Jpanel и две надписи JLabel. На панели изображено наше игровое поле, а надписи это наш счёт и количество оставшихся ходов. По окончании игры всплывает ещё одно диалоговое окно с результатами игрока. После нажатия «Ок» на всплывающем диалоговом окне осуществляется подключение к серверу. В случае неудачи появится новое диалоговое окно с предупреждением о проблемах с подключением. В случае же успеха появится ещё одно всплывающее окно, в основе которого лежит JScrollPane c помещённой внутрь таблицей, где и отображаются все результаты.

1. **Описание результатов работы программы**

Ниже представлена работа клиент-серверного приложения.

Для начала необходимо запустить сервер:

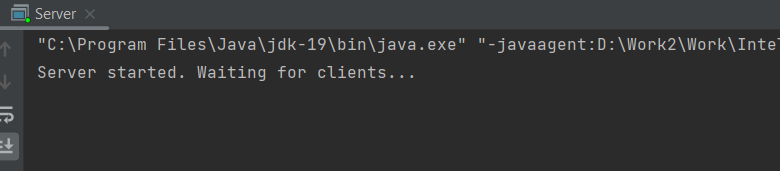
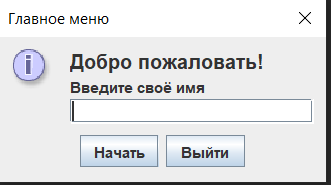


Рисунок 1. Окно запуска сервера

Далее запускаем клиент



**Рисунок 2.Окно клиентского приложения.**

Нажав на кнопку «Начать», мы получим следующее предупреждение

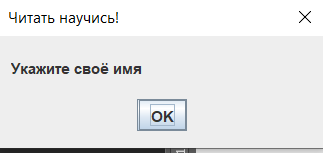


Рисунок 3 Окно предупреждения

Нажав «Ок» и вернувшись снова в главное меню, на этот раз вводим любое имя и нажимаем кнопку «Начать».

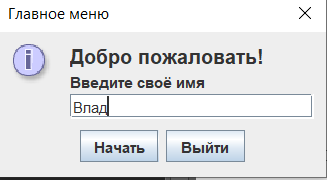


Рисунок 4 Образец корректного ввода

После всплывает новое окно и на этот раз с нашей игрой.



**Рисунок 5. Окно игры**

Собрав несколько комбинаций у нас заканчиваются жизни, и игра завершается.

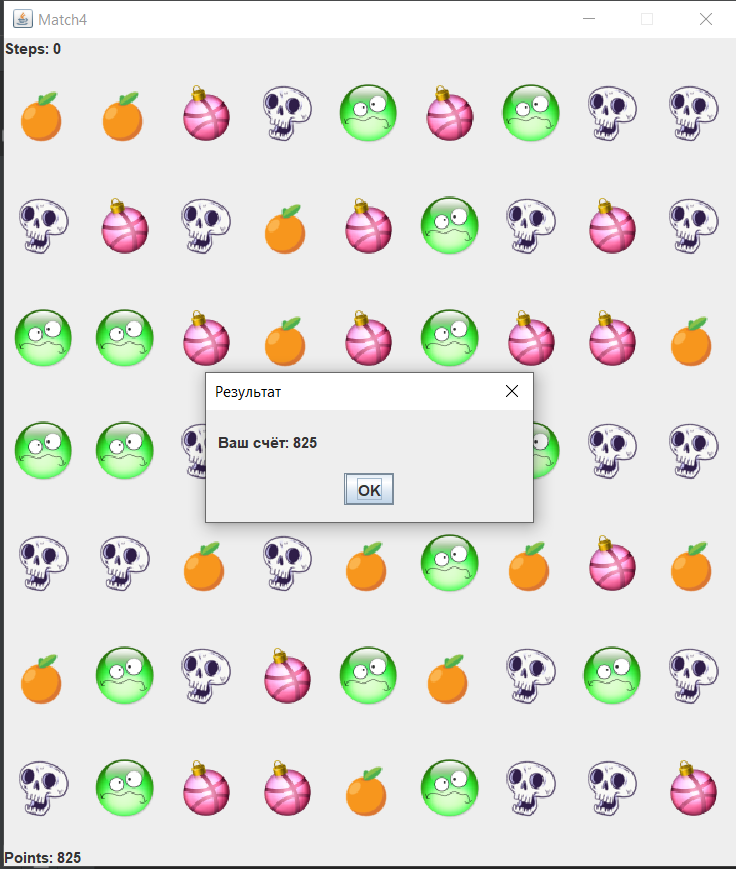


Рисунок 6 Конец игры

После нажатия на «Ок» происходит соединение с сервером

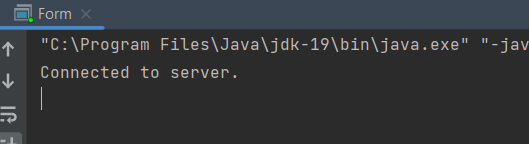
****

Рисунок 7 Консоль клиента

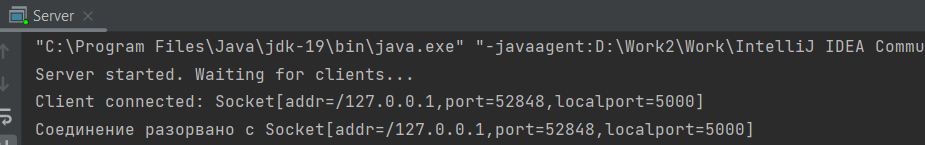


Рисунок 8 Консоль сервера

В результате чего, на экране появляется новое окно с таблицей наилучших результатов игроков

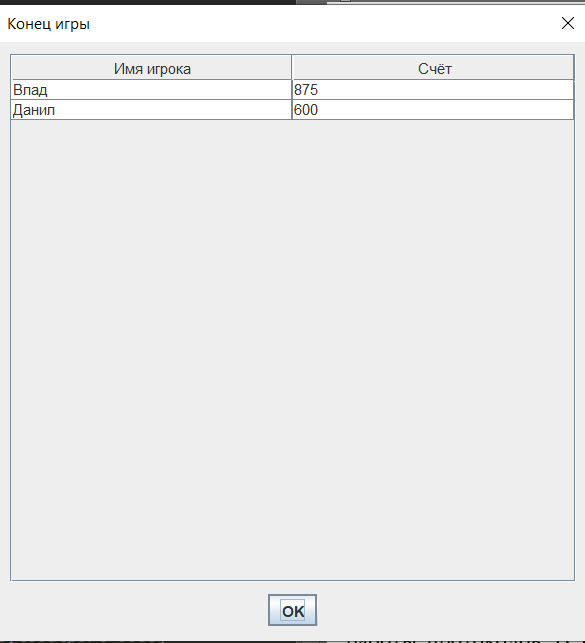


Рисунок 9 Таблица с результатами

**Заключение**

В ходе выполнения курсового проектирования были изучены принципы работы протокола TCP. Были получены навыки разработки приложения с пользовательским интерфейсом. Было разработано клиент-серверное приложение жанра «три в ряд», используя среду разработки IntelliJ IDEA Community Edition 2022 и язык программирования Java с использованием графической библиотеки Swing. Удалось научиться грамотно отлаживать программу и находить в ней слабые уязвимые места.

**Литература**

1. "Head First Java, Изучаем Java", Кэти Сьерра, Берт Бэйтс
2. "Java. Руководство для начинающих", Герберт Шилдт
3. "Java для чайников", Барри Бёрд
4. "Java. Полное руководство", Герберт Шилдт
5. Гурвиц Г. Разработка реального приложения в среде клиент-сервер ДВГУПС, 2005

**Приложение А. Листинг программы**

**Приложение А.1. Файл «Server.java» Server.**

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

public class Server {

public static void main(String[] args) {

try {

ServerSocket serverSocket = new ServerSocket(5000);

System.out.println("Server started. Waiting for clients...");

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected: " + clientSocket);

// Обработка подключенного клиента в отдельном потоке

ClientHandler clientHandler = new ClientHandler(clientSocket);

clientHandler.start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Приложение А.2. Файл «ClientHandler.java» Server.**

import java.io.\*;

import java.net.Socket;

import java.util.ArrayList;

public class ClientHandler extends Thread {

private Socket clientSocket;

private ObjectInputStream in;

private ObjectOutputStream out;

private ObjectInputStream readFile;

private ObjectOutputStream writeFile;

private ArrayList<Data> stroka = new ArrayList<>();

private ArrayList<Data> strokaTmp = new ArrayList<>();

public ClientHandler(Socket clientSocket) throws IOException {

this.clientSocket = clientSocket;

in = new ObjectInputStream(clientSocket.getInputStream());

out = new ObjectOutputStream(clientSocket.getOutputStream());

}

@Override

public void run() {

try {

boolean error = false;

strokaTmp = (ArrayList<Data>) in.readObject();

boolean replace = true;

try {

readFile = new ObjectInputStream(new BufferedInputStream(

new FileInputStream("A.ser")));

stroka = (ArrayList<Data>) readFile.readObject();

readFile.close();

} catch(IOException ex) {

System.out.println("Отсутствует необходимый файл. Будет создан новый");

error = true;

} catch (ClassNotFoundException e) {

System.out.println("Что-то пошло не так");

}

stroka.add(new Data(strokaTmp.get(0).getName(), strokaTmp.get(0).getPoints()));

if(!error) {

for(int j = 0; j < stroka.size() - 1; j++){

int i = stroka.size() - 1;

if(stroka.get(i).getName().equals(stroka.get(j).getName())){

if(stroka.get(i).getPoints() > stroka.get(j).getPoints()){

stroka.remove(j);

}

else {

stroka.remove(i);

}

}

}

Data temp;

while (replace) {

replace = false;

for (int i = stroka.size() - 1; i > 0; i--) {

if(stroka.get(i).getPoints() > stroka.get(i-1).getPoints()){

temp = stroka.get(i - 1);

stroka.set(i - 1, stroka.get(i));

stroka.set(i,temp);

replace = true;

}

}

}

}

writeFile = new ObjectOutputStream(new BufferedOutputStream(

new FileOutputStream("A.ser")));

writeFile.writeObject(stroka);

writeFile.close();

out.writeObject(stroka);

System.out.println("Соединение разорвано с " + clientSocket);

in.close();

out.close();

clientSocket.close();

} catch (IOException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

throw new RuntimeException(e);

}

}

}

**Приложение А.3. Файл «Form.java» Client.**

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.util.ArrayList;

public class Form extends JFrame {

private JPanel gameAndScore;

private JPanel startPanel;

private GameFieldPanel GameFieldPanel;

public static JTextField nameField;

public static JLabel status, steps;

public static int points , life;

private int choice;

public static ArrayList<Data> stroka = new ArrayList<>();

public Form() {

startPanel = new JPanel(new GridLayout(3, 1));

JLabel startLabel = new JLabel("Добро пожаловать!");

startLabel.setFont(new Font("Arial", Font.BOLD, 22));

startPanel.add(startLabel);

JLabel nameLabel = new JLabel("Введите своё имя");

startLabel.setFont(new Font("Arial", Font.BOLD, 16));

startPanel.add(nameLabel);

nameField = new JTextField(10);

nameField.setToolTipText("Поле для имени");

startPanel.add(nameField);

while (true) {

choice = JOptionPane.showOptionDialog(this, startPanel,

"Главное меню", JOptionPane.DEFAULT\_OPTION, JOptionPane.INFORMATION\_MESSAGE,

null, new String[]{"Начать", "Выйти"}, null);

if (nameField.getText().equals(""))

JOptionPane.showMessageDialog(this, "Укажите своё имя", "Читать научись!", JOptionPane.PLAIN\_MESSAGE);

else break;

}

startGame();

}

public Form( int life)

{

this.life = life;

}

public void startGame() {

points = 0;

life = 1;

if (choice == 0) {

//JOptionPane.showInputDialog();

setLayout(new GridLayout(1, 2));

status = new JLabel("Points: " + String.valueOf(points));

steps = new JLabel("Steps: " + String.valueOf(life));

gameAndScore = new JPanel();

gameAndScore.setLayout(new BorderLayout());

gameAndScore.setOpaque(false);// Сделать фон прозрачным

GameFieldPanel=new GameFieldPanel();

gameAndScore.add(GameFieldPanel);

gameAndScore.add(status, "South");

gameAndScore.add(steps, "North");

getContentPane().add(gameAndScore);

getContentPane().revalidate();

getContentPane().repaint();

} else {

System.exit(0);

}

}

public void endGame() {

JTable table = new JTable();

table.setDefaultEditor(Object.class, null);

DefaultTableModel MyModel = (DefaultTableModel) table.getModel();

MyModel.addColumn("Имя игрока");

MyModel.addColumn("Счёт");

JScrollPane scroll = new JScrollPane(table);

for (int i = 0; i < stroka.size(); i++) {

MyModel.addRow(new Object[]{stroka.get(i).getName(), stroka.get(i).getPoints()});

}

JOptionPane.showMessageDialog(this, scroll, "Конец игры", JOptionPane.PLAIN\_MESSAGE);

System.exit(0);

}

public static void main(String[] args) {

Form MainFrame = new Form();

// MainFrame.setContentPane(MainFrame.MainPanel);

MainFrame.setTitle("Match4");

MainFrame.setSize(600, 700);

MainFrame.setResizable(false);

MainFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

MainFrame.setVisible(true);

}

**Приложение А.4. Файл «GameFieldPanel.java» Client.**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.\*;

import java.net.\*;

import java.util.ArrayList;

import java.util.Random;

public class GameFieldPanel extends JPanel implements ActionListener {

protected int rowNum;

protected int columnNum;

public static int[][] board;

static final int EMPTY = 0;

static final int MAX = 4;

protected JButton[][] balls;

public static int selectedRow = -1;

public static int selectedCol = -1;

public static boolean pieceSelected = false;

private ImageLoader imageLoader;

GameFieldPanel(){

setLayout(new GridLayout(7, 9));

setPreferredSize(new Dimension(580, 470));

setOpaque(false);// Сделать фон прозрачным

reset();

}

public void reset() {

rowNum = 7;

columnNum = 9;

board = new int[rowNum][columnNum];

// Заполняем таблицу случайными шарами

for(int row = 0; row < rowNum; row++)

for(int col = 0; col < columnNum; col++)

{

board[row][col] = EMPTY;

}

start();

}

public void start(){

while(existsEmptyCell())

{

dropPieces();

eliminateMatches();

}

balls = new JButton[rowNum][columnNum];

imageLoader = new ImageLoader();

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

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

balls[i][j] = new JButton();

balls[i][j].setIcon(new ImageIcon(imageLoader.getImage(board[i][j])));

balls[i][j].addActionListener(this);

balls[i][j].setBorder(BorderFactory.createEmptyBorder());

balls[i][j].setContentAreaFilled(false);

add(balls[i][j]);

}

}

}

public boolean existsEmptyCell() {

boolean empty\_cell\_found = false;

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

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

if(board[i][j] == EMPTY) {

empty\_cell\_found = true;

}

}

}

return empty\_cell\_found;

}

public void dropPieces() {

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

int[] thiscol = new int[rowNum];

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

thiscol[i] = EMPTY;

}

int target\_index = rowNum - 1;

for(int i=rowNum-1; i>=0; i--) {

if(board[i][j] != EMPTY) {

thiscol[target\_index] = board[i][j];

target\_index--;

}

}

while(target\_index >= 0) {

thiscol[target\_index] = new Random().nextInt(MAX)+1;

target\_index--;

}

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

board[i][j] = thiscol[i];

}

}

}

public void eliminateMatches() {

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

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

// 5 in a row

if(0<j && j<columnNum-3) {

if(board[i][j-1] == board[i][j] &&

board[i][j+1] == board[i][j] &&

board[i][j+2] == board[i][j] &&

board[i][j+3] == board[i][j]) {

board[i][j] = EMPTY;

board[i][j-1] = EMPTY;

board[i][j+1] = EMPTY;

board[i][j+2] = EMPTY;

board[i][j+3] = EMPTY;

Form.points+=100;

}

}

// 5 in a col

if(0<i && i<rowNum-3) {

if(board[i-1][j] == board[i][j] &&

board[i+1][j] == board[i][j] &&

board[i+2][j] == board[i][j] &&

board[i+3][j] == board[i][j]) {

board[i][j] = EMPTY;

board[i-1][j] = EMPTY;

board[i+1][j] = EMPTY;

board[i+2][j] = EMPTY;

board[i+3][j] = EMPTY;

Form.points += 100;

}

}

// 4 in a row

if(0<j && j<columnNum-2) {

if(board[i][j-1] == board[i][j] &&

board[i][j+1] == board[i][j] &&

board[i][j+2] == board[i][j]) {

board[i][j] = EMPTY;

board[i][j-1] = EMPTY;

board[i][j+1] = EMPTY;

board[i][j+2] = EMPTY;

Form.points += 50;

}

}

// 4 in a col

if(0<i && i<rowNum-2) {

if(board[i-1][j] == board[i][j] &&

board[i+1][j] == board[i][j] &&

board[i+2][j] == board[i][j]) {

board[i][j] = EMPTY;

board[i-1][j] = EMPTY;

board[i+1][j] = EMPTY;

board[i+2][j] = EMPTY;

Form.points += 50;

}

}

// 3 in a row

if(0<j && j<columnNum-2) {

if(board[i][j-1] == board[i][j] &&

board[i][j+1] == board[i][j]) {

board[i][j] = EMPTY;

board[i][j-1] = EMPTY;

board[i][j+1] = EMPTY;

Form.points += 25;

}

}

// 3 in a col

if(0<i && i<rowNum-2) {

if(board[i-1][j] == board[i][j] &&

board[i+1][j] == board[i][j]) {

board[i][j] = EMPTY;

board[i-1][j] = EMPTY;

board[i+1][j] = EMPTY;

Form.points += 25;

}

}

}

}

}

public boolean isValidSwap(int row1,int col1,int row2,int col2) {

if(row1==row2 && Math.abs(col1-col2)==1) {

// ok

} else if (col1==col2 && Math.abs(row1-row2)==1) {

// ok

} else {

return false;

}

int tmp = board[row1][col1];

board[row1][col1] = board[row2][col2];

board[row2][col2] = tmp;

eliminateMatches();

if(existsEmptyCell()) {

do {

dropPieces();

eliminateMatches();

} while(existsEmptyCell());

return true;

} else {

tmp = board[row1][col1];

board[row1][col1] = board[row2][col2];

board[row2][col2] = tmp;

return false;

}

}

public void handleClick(int row,int col) {

if(!pieceSelected) {

pieceSelected = true;

selectedRow = row;

selectedCol = col;

balls[row][col].setIcon(new ImageIcon(imageLoader.getImage(board[row][col] + 4)));

return;

}

if(pieceSelected && selectedRow==row && selectedCol==col) {

pieceSelected = false;

selectedRow = -1;

selectedCol = -1;

balls[row][col].setIcon(new ImageIcon(imageLoader.getImage(board[row][col])));

return;

}

if(!(isValidSwap(row,col,selectedRow,selectedCol))) {

Form.status.setText("Invalid move");

return;

}

pieceSelected = false;

selectedRow = -1;

selectedCol = -1;

redrawBoard();

Form.life--;

Form.status.setText("Points: " + String.valueOf(Form.points));

Form.steps.setText("Steps: " + String.valueOf(Form.life));

if(Form.life == 0) {

Form.stroka.add(new Data(Form.nameField.getText(), Form.points));

JOptionPane.showMessageDialog(this, "Ваш счёт: " + Form.points, "Результат", JOptionPane.PLAIN\_MESSAGE);

connectToServer();

Form a = new Form( 3);

a.endGame();

}

}

public void redrawBoard() {

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

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

balls[i][j].setIcon(new ImageIcon(imageLoader.getImage(board[i][j])));

}

}

}

public void actionPerformed(ActionEvent e) {

//setStatus("Points: " + String.valueOf(points));

JButton button = (JButton) e.getSource();

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

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

if (button == balls[i][j]) {

handleClick(i, j);

break;

}

}

}

}

public void connectToServer() {

try {

Socket socket = new Socket("localhost", 5000);

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

ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

out.writeObject(Form.stroka);

Form.stroka = (ArrayList<Data>) in.readObject();

out.close();

in.close();

socket.close();

} catch (IOException e) {

JOptionPane.showMessageDialog(this, "Не удалось синхронизироваться с сервером", "Конец игры", JOptionPane.PLAIN\_MESSAGE);

} catch (ClassNotFoundException e) {

throw new RuntimeException(e);

}

}

}

**Приложение А.5. Файл «ImageLoader.cs» Client.**

import javax.imageio.ImageIO;

import javax.swing.\*;

import java.awt.\*;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

public class ImageLoader {

public Image loadImage(String fileName,int width,int height) {

Image img = null;

try {

File imageFile = null;

imageFile = new File(getClass().getClassLoader().getResource(fileName).getFile());

// Очистка изображения, если оно было до этого

if (img != null) {

img.flush();

}

// Загрузка изображения из файла в объект img

img = ImageIO.read(imageFile).getScaledInstance(width, height, Image.SCALE\_AREA\_AVERAGING);

} catch (IOException exOb) {

// Вывод в консоль сообщения об ошибке

System.out.println("Cannot load image file: " + fileName);

}

return img;

}

public ImageIcon loadIconImage(String fileName) {

ImageIcon img = null;

img = new ImageIcon(fileName);

return img;

}

public BufferedImage loadBuffImage(String fileName) {

BufferedImage img = null;

File imageFile;

// Получение файла из папки ресурсов

imageFile = new File(fileName);

// Загрузка изображения из файла в объект img

try {

img = ImageIO.read(imageFile);

} catch (IOException e) {

System.out.println("Cannot load image file: " + fileName);

}

return img;

}

private Image buffIMG;

public Image getImage(int id)

{

switch (id)

{

case 1:

buffIMG = loadBuffImage("src/Image/blackBallDark.png");

break;

case 2:

buffIMG = loadBuffImage("src/Image/greenBallDark.png");

break;

case 3:

buffIMG = loadBuffImage("src/Image/orangeBallDark.png");

break;

case 4:

buffIMG = loadBuffImage("src/Image/pinkBallDark.png");

break;

case 5:

buffIMG = loadBuffImage("src/Image/blackBallLight.png");

break;

case 6:

buffIMG = loadBuffImage("src/Image/greenBallLight.png");

break;

case 7:

buffIMG = loadBuffImage("src/Image/orangeBallLight.png");

break;

case 8:

buffIMG = loadBuffImage("src/Image/pinkBallLight.png");

break;

}

return buffIMG;

}

}

**Приложение Б. UML-диаграммы**

Приложение Б.1. UML – диаграмма деятельности



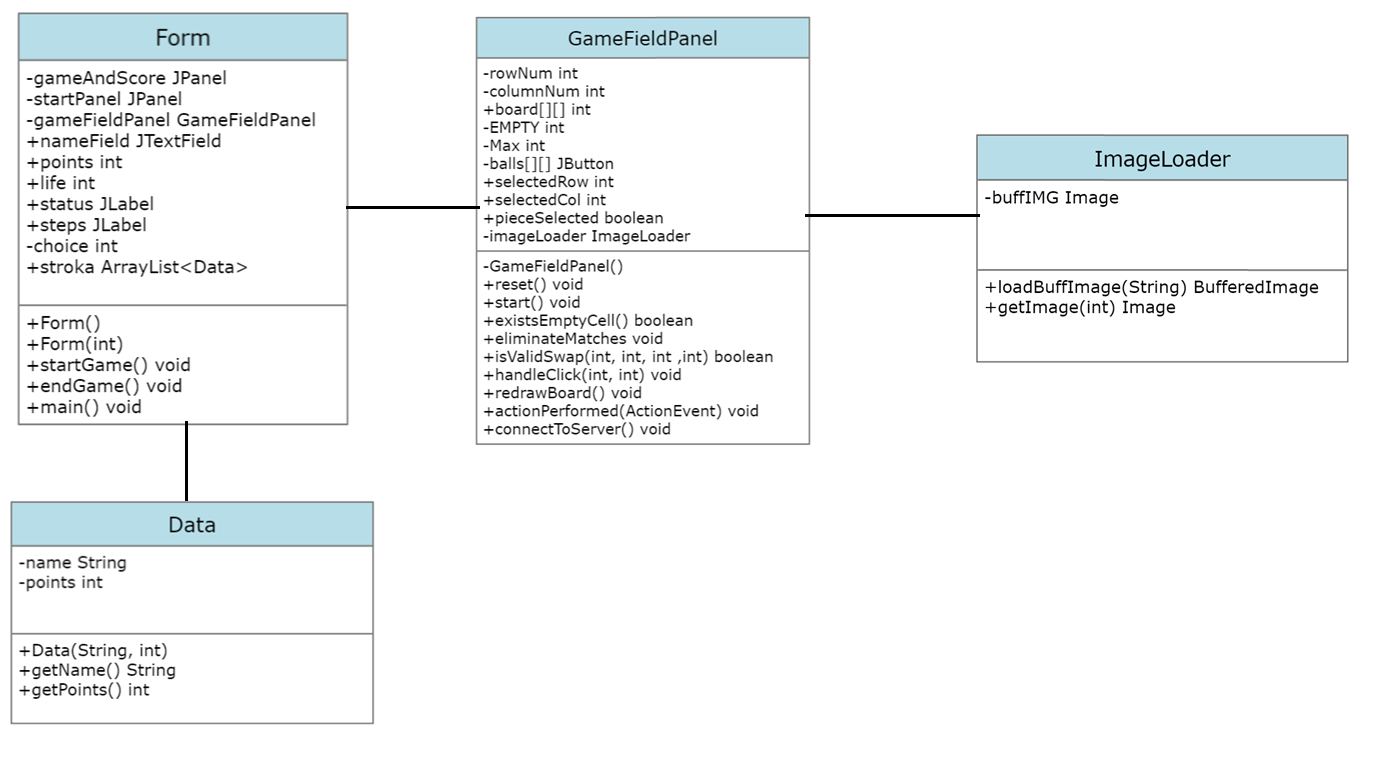
**Рисунок 10.UML – диаграмма деятельности**

Приложение Б.2. UML – диаграмма вариантов использования приложения.



**Рисунок 11. UML - диаграмма вариантов использования приложения.**

Приложение Б.3. UML – диаграмма классов.



**Рисунок 12.UML - диаграмма классов клиента**

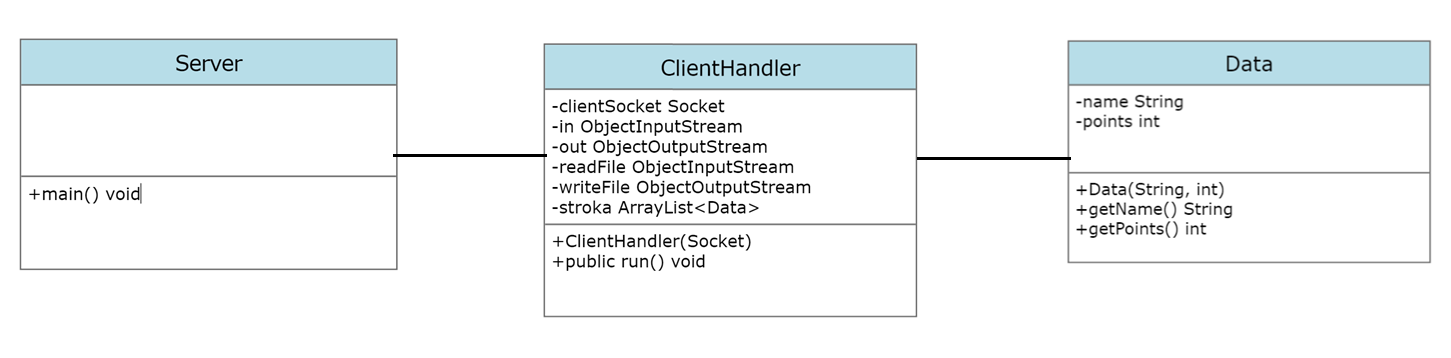


Рисунок 13.UML - диаграмма классов сервера

Приложение Б.4. UML – диаграмма последовательности.

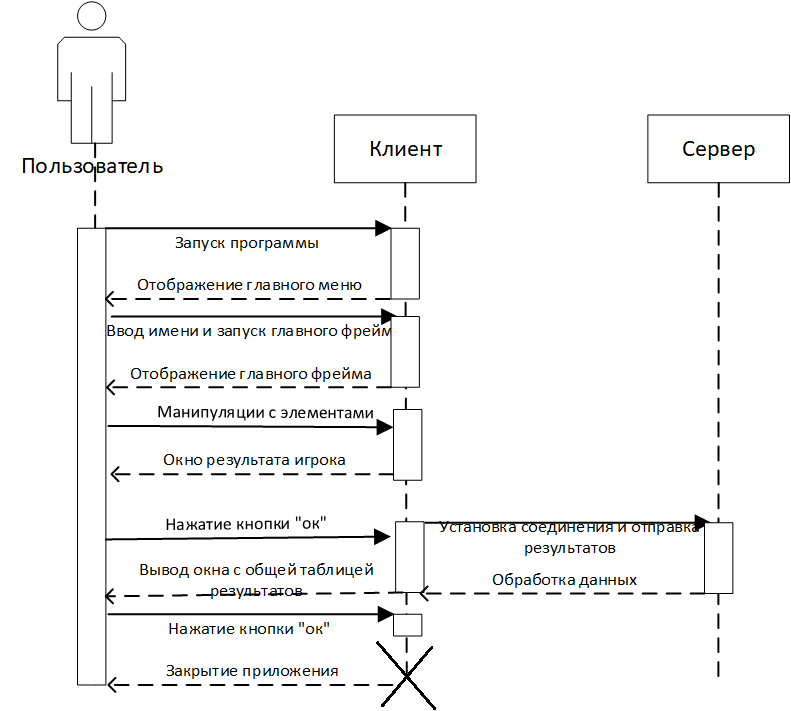


Рисунок 14 UML – диаграмма последовательности.

Приложение Б.5. UML – диаграмма развёртывания.



Рисунок 15 диаграмма развертывания.