Министерство науки и высшего образования Российской Федерации Пензенский государственный университет

Кафедра «Вычислительная техника»

Пенза 2024

Выполнил:

студент группы 21ВВП1

Тюрин В.И.

Принял:

Юрова О.В.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Программирование на языке Java»

на тему «Разработка многомодульного приложения на языке Java»

# Содержание

[Содержание 4](#_Toc168270314)

[Введение 6](#_Toc168270315)

[1. Постановка задачи 7](#_Toc168270316)

[2. Выбор решения 8](#_Toc168270317)

[3. Описание программы 9](#_Toc168270318)

[4. Описание способа реализации пользовательского интерфейса 12](#_Toc168270319)

[5. Описание результатов работы программы 13](#_Toc168270320)

[Заключение 16](#_Toc168270321)

[Список используемых источников 17](#_Toc168270322)

[Приложение A. Листинг программы 18](#_Toc168270323)

[Приложение A.1. Файл «Server.java» Server 18](#_Toc168270324)

[Приложение A.2. Файл «ClientHandler.java» Server 19](#_Toc168270325)

[Приложение A.3. Файл «User.java» Server 21](#_Toc168270326)

[Приложение A.4. Файл «UserDatabase.java» Server 22](#_Toc168270327)

[Приложение A.5. Файл «Game.java» Client 23](#_Toc168270328)

[Приложение A.6. Файл «GlobalConstance.java» Client 24](#_Toc168270329)

[Приложение A.7. Файл «Level.java» Client 25](#_Toc168270330)

[Приложение A.8. Файл «LevelService.java» Client 26](#_Toc168270331)

[Приложение A.9. Файл «RandGenerate.java» Client 27](#_Toc168270332)

[Приложение A.10. Файл «ServerTask.java» Client 28](#_Toc168270333)

[Приложение A.11. Файл «EventHandler.java» Client 29](#_Toc168270334)

[Приложение A.12. Файл «PlayerController.java» Client 30](#_Toc168270335)

[Приложение A.13. Файл «AddEnemy.java» Client 31](#_Toc168270336)

[Приложение A.14. Файл «EnemyTankMovement.java» Client 33](#_Toc168270337)

[Приложение A.15. Файл «GiveGift.java» Client 35](#_Toc168270338)

[Приложение A.16. Файл «GameData.java» Client 37](#_Toc168270339)

[Приложение A.17. Файл «SceneObject.java» Client 40](#_Toc168270340)

[Приложение A.18. Файл «GifType.java» Client 41](#_Toc168270341)

[Приложение A.19. Файл «OperatorGift.java» Client 42](#_Toc168270342)

[Приложение A.20. Файл «TankSide.java» Client 44](#_Toc168270343)

[Приложение A.21. Файл «TankType.java» Client 45](#_Toc168270344)

[Приложение A.22. Файл «Tank.java» Client 46](#_Toc168270345)

[Приложение A.23. Файл «WallType.java» Client 49](#_Toc168270346)

[Приложение A.24. Файл «Wall.java» Client 50](#_Toc168270347)

[Приложение A.25. Файл «Direction.java» Client 52](#_Toc168270348)

[Приложение A.26. Файл «Flag.java» Client 53](#_Toc168270349)

[Приложение A.27. Файл «Shot.java» Client 54](#_Toc168270350)

[Приложение A.28. Файл «Player.java» Client 57](#_Toc168270351)

[Приложение A.29. Файл «GameStatus.java» Client 58](#_Toc168270352)

[Приложение A.30. Файл «GameObjectHelper.java» Client 59](#_Toc168270353)

[Приложение A.31. Файл «SceneHelper.java» Client 61](#_Toc168270354)

[Приложение A.32. Файл «EndGameScene.java» Client 63](#_Toc168270355)

[Приложение A.33. Файл «StartMenuHelper.java» Client 64](#_Toc168270356)

[Приложение A.34. Файл «StartMenu.java» Client 66](#_Toc168270357)

[Приложение A.35. Файл «LoginPage.java» Client 67](#_Toc168270358)

[Приложение A.36. Файл «GameEnvironmentHelper.java» Client 69](#_Toc168270359)

[Приложение A.37. Файл «GamePage.java» Client 71](#_Toc168270360)

[Приложение A.38. Файл «module-info.java» Client 73](#_Toc168270361)

[Приложение A.39. Файл «pom.xml» Client 74](#_Toc168270362)

[Приложение B. UML - диаграммы 76](#_Toc168270363)

[Приложение B.1. Диаграммы классов 76](#_Toc168270364)

[Приложение B.2. Диаграмма развёртывания 78](#_Toc168270365)

[Приложение B.3. Диаграмма последовательности 79](#_Toc168270366)

[Приложение B.4. Диаграмма деятельности 80](#_Toc168270367)

[Приложение B.5. Диаграмма вариантов использования приложения 81](#_Toc168270368)

# Введение

В последние годы наблюдается рост интереса к многопользовательским играм, что приводит к увеличению числа игр с клиент-серверной архитектурой. Одним из таких проектов является многомодульная игра "Змейка", где пользователи могут соревноваться друг с другом в реальном времени. Эта игра не только предлагает увлекательный геймплей, но и требует эффективного и стабильного обмена данными между клиентами и сервером для синхронизации действий игроков.

Клиент-серверная архитектура представляет собой распределенную систему, в которой задачи и сетевые нагрузки разделяются между серверами (поставщиками услуг) и клиентами (потребителями). Взаимодействие между клиентами и серверами осуществляется через компьютерные сети и может включать как физические устройства, так и программные компоненты. Простой пример технологии клиент-сервер заключается в том, что пользователь отправляет запрос (например, подключение к игровому серверу), а сервер отвечает, предоставляя данные о текущем состоянии игры.

Цель данного курсового проекта - разработать многомодульное приложение с клиент-серверной архитектурой для игры "Змейка", включающее сервер и клиентов, взаимодействующих по сети. В проекте будут реализованы механизмы управления игровым процессом, синхронизации действий игроков, а также обеспечена поддержка сетевой безопасности и устойчивости к нагрузкам.

# Постановка задачи

В курсовой работе необходимо разработать многомодульную программу на языке Java – игровое приложение "Змейка".

Необходимо создать сервер и клиент. Клиент – игровое приложение, которое будет отправлять серверу запросы. Сервер должен отвечать на эти запросы.

Функции клиента:

* Интерфейс пользователя
* Взаимодействие с сервером

Функции сервера:

* Взаимодействие с программами-клиентами

Операционная система – Windows.

Язык программирования – Java.

Среда программирования: IntelliJ IDEA Community Edition.

# Выбор решения

Важной частью курсовой работы является передача сообщений от клиента к серверу. Для реализации этого используются TCP протокол. TCP расшифровывается как Transmission Control Protocol. По сравнению с UDP, сетевой протокол TCP более надежен, но медленнее и сложнее. Он часто используется в ситуациях, когда важна надёжность.

В отличие от UDP, протокол TCP устанавливает предварительную связь между двумя сторонами. Ввиду этого иногда данные не могут потеряться в пути, взамен получения более низкой скорости передачи пакетов.

Протокол TCP был выбран, потому что нам необходимо передавать данные пользователя для авторизации и его прогресс прохождения игры, эти данные не большого размера, поэтому скорость не так важна как надёжность. Для взаимодействия между компьютерами используются адреса и порты.

# Описание программы

Разработанная программа состоит основывается на вышеописанных протоколах, а также из нескольких модулей, которые реализовывались в следующем порядке:

1. SnakeGame.java – основной файл проекта игры.
2. NetworkClient.java,– вспомогательный модуль, для подключения к серверу.
3. NetworkServer.java – основный файл проекта сервера.
4. ServerTask.java – отвечает за отправку запросов на сервер и получение ответов.
5. ClientHandler.java, – отвечает за обработку взаимодействия с клиентами в многопользовательской игре.

При запуске программы загружаются изображения игровых объектов, создается графический интерфейс для игры и устанавливается соединение с сервером. Игровой интерфейс включает поле, на котором игрок управляет змейкой, собирая яблоки и избегая столкновений.

Когда пользователь вводит свое имя и начинает игру, программа создает окно с игровым полем и инициализирует сетевое соединение для взаимодействия с сервером. В процессе игры отслеживаются направления движения змейки и ее столкновения с границами поля или собственным телом.

Основные модули программы:

SnakeGame.java – главный файл игры:

Загрузчик изображений для игровых объектов (змейка, яблоки, фон).

Настройка графического интерфейса и инициализация сетевого подключения.

Логика игры: движение змейки, проверка на столкновения, сбор яблок и обновление очков.

Отправка очков на сервер по завершении уровня или игры.

NetworkClient.java – клиентская часть для взаимодействия с сервером:

Установка соединения с сервером по заданному адресу и порту.

Отправка запросов на сервер и получение ответов.

Для работы с сервером необходимо запустить серверную часть, которая состоит из нескольких модулей:

Server.java – основной файл сервера, который принимает подключения от клиентов и распределяет их на обработку.

ClientHandler.java – поток для обработки запросов от клиентов:

Обработка сообщений о новых очках игроков.

Отправка текущих результатов всем подключенным клиентам.

При подключении к серверу, клиент может отправить запрос на получение текущих очков или обновить свои очки по завершении уровня. Сервер обрабатывает эти запросы и обновляет данные, обеспечивая синхронизацию прогресса игроков.

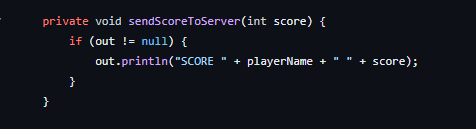
Таким образом, взаимодействие между клиентом и сервером обеспечивает динамическое обновление данных о состоянии игры и прогрессе игроков, что позволяет поддерживать многопользовательский режим и сетевую игру в реальном времени.

Рисунок 1 – Отправка сообщений от клиента серверу.

Отправка сообщений от сервера клиенту.

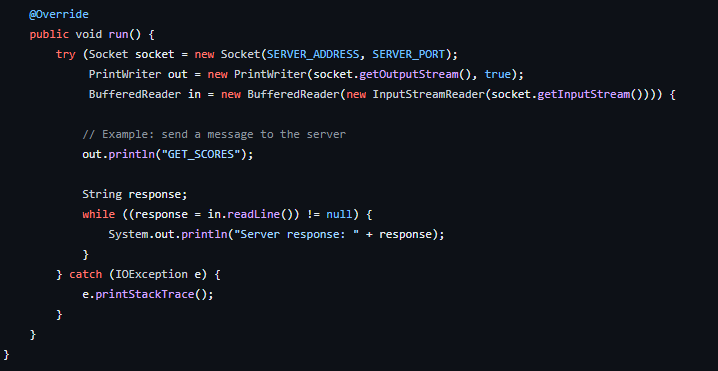


Рисунок 2 – Отправка сообщений от сервера клиенту.

Взаимодействие пользователя-клиента-сервера отображено в Приложение B.3 Диаграмма последовательности. Работа клиента и сервера показаны в Приложение B.2 Диаграмма развёртывания и в Приложение B.4 Диаграмма деятельности.

# Описание способа реализации пользовательского интерфейса

Пользовательский интерфейс присутствует только в клиентской части приложения и реализован с использованием библиотеки Swing. Приложение состоит из одного экрана, на котором происходит сам игровой процесс.

При запуске программы пользователь вводит свое имя в появившемся диалоговом окне. После ввода имени открывается основное игровое окно, где отображается игровое поле.

Основные элементы интерфейса:

Игровое поле: На экране отображается поле игры, состоящее из ячеек, по которому перемещается змейка и появляются яблоки.

Очки: В нижней части экрана отображается текущий счет игрока.

Управление игрой:

Движение змейки осуществляется с помощью клавиш: W (вверх), A (влево), S (вниз), D (вправо).

Перезапуск игры: Нажатие клавиши R перезапускает игру после её завершения.

Логика работы интерфейса:

Отрисовка объектов: Игра загружает изображения для змейки, яблок и фона, и обновляет их положение на экране по мере продвижения игры.

Обработка событий: Обработка нажатий клавиш для управления змейкой и перезапуска игры.

Игровые состояния: Обработка состояния игры (в игре, игра окончена) и соответствующее обновление интерфейса.

Сетевое взаимодействие: При завершении уровня или игры, текущий счет отправляется на сервер для обновления данных.

Игровой интерфейс обеспечивает визуальную обратную связь игроку, позволяя ему видеть текущее состояние игры, а также взаимодействовать с игрой с помощью клавиатуры. Для выхода из приложения необходимо закрыть окно, нажав на крестик в правом верхнем углу.

Возможности клиента отображены в Приложении B.5 Диаграмма вариантов использования приложения.

# Описание результатов работы программы

Ниже представлена работа клиент-серверного приложения.

Для начала необходимо запустить сервер.

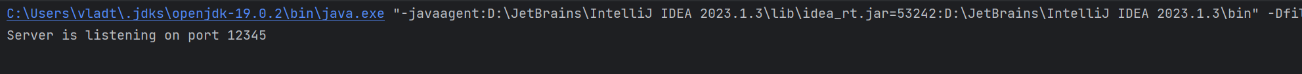


Рисунок 3 – Запуск сервера.

Далее запускаем клиент.

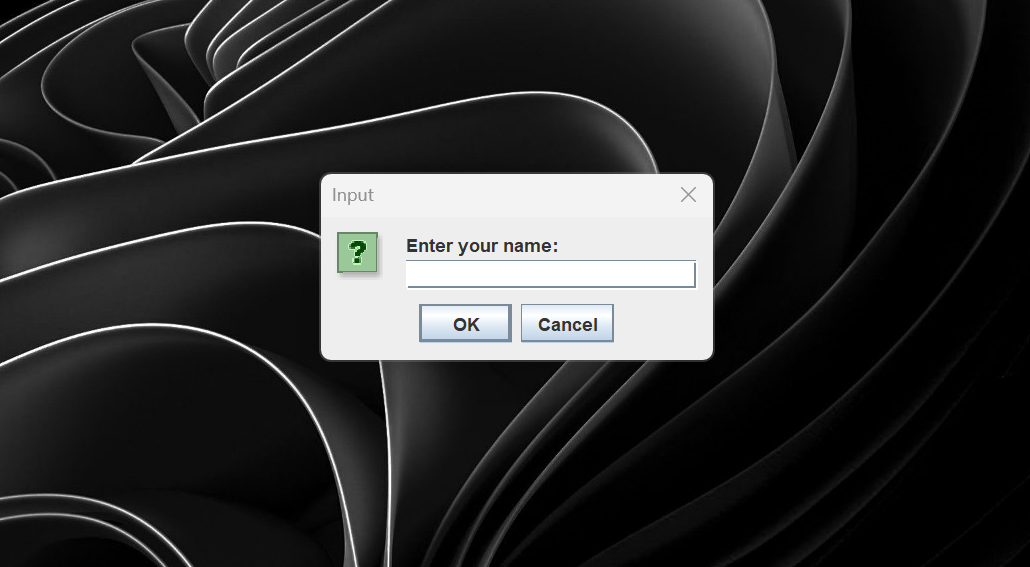


Рисунок 4 – Сцена авторизации.

Не верно ввели логин или пароль, срабатывает исключение.

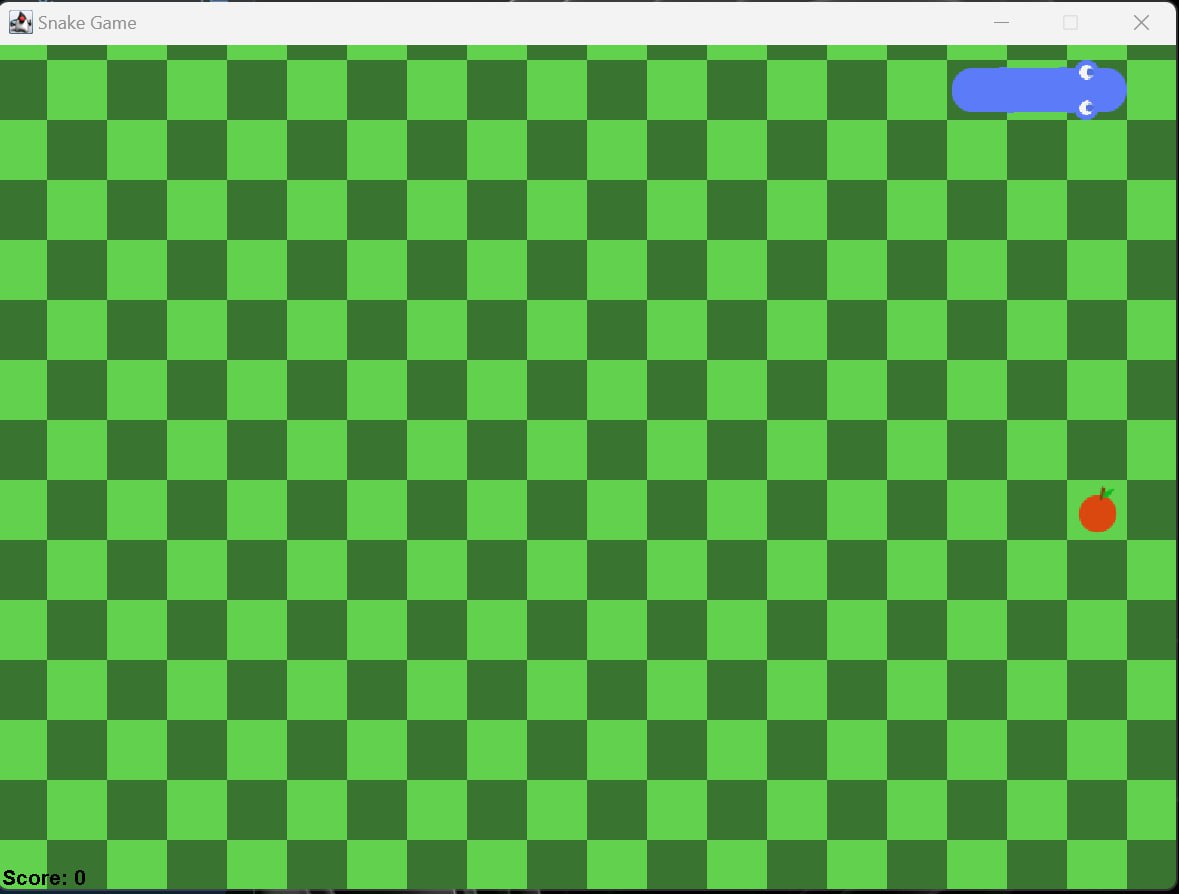


Рисунок 5 – Геймплей игры.

Вводим свое имя, запускается игра, если змейка врезается в стенку, то игра заканчивается, и пишется ваш счет. Если вы набираете 20 очков, собирая яблоки, то вы побеждаете и игра заканчивается.

# Заключение

В процессе выполнения данного курсового проекта были изучены и применены на практике принципы разработки многомодульного приложения на языке Java. Также изучены принципы разработки графического интерфейса и освоены следующие технологии:

Java Collections Framework: Использование коллекций для хранения и управления данными.

Механизм обработки исключений: Обработка ошибок и исключений для обеспечения надежной работы программы.

Java Stream API: Обработка данных с использованием потоков.

Java Multithreading: Реализация многопоточности для повышения производительности и отзывчивости приложения.

Сетевое взаимодействие: Обеспечение взаимодействия между клиентом и сервером по сети.

В результате выполнения данного курсового проекта была разработана система программ с клиент-серверной архитектурой, обеспечивающая взаимодействие между клиентами и сервером.

Программа написана на языке Java с использованием среды разработки IntelliJ IDEA Community Edition.

Разработанное приложение позволяет пользователям играть в многопользовательскую игру "Змейка" в режиме реального времени, обеспечивая увлекательный игровой процесс и надежную синхронизацию данных между клиентами и сервером.

# Список используемых источников

1. code-basics.com Сайт о программировании [Электронный ресурс]. Режим доступа: <https://code-basics.com/>, свободный (Дата обращения 22.05.2024).
2. javarush.com Сайт о программировании [Электронный ресурс]. Режим доступа: <https://javarush.com/>, свободный (Дата обращения 22.05.2024).
3. sentry.io Сайт о программировании [Электронный ресурс]. Режим доступа: <https://sentry.io/>, свободный (Дата обращения 22.05.2024).

# Приложение A. Листинг программы

## Приложение A.1. Файл «SnakeGame.java» Server

package com.snakegame;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

import java.util.List;

import java.util.concurrent.CopyOnWriteArrayList;

public class SnakeGame extends JFrame implements ActionListener {

private static final int WIDTH = 800;

private static final int HEIGHT = 600;

private static final int DOT\_SIZE = 40;

private static final int RAND\_POS\_X = WIDTH / DOT\_SIZE;

private static final int RAND\_POS\_Y = HEIGHT / DOT\_SIZE;

private static final int DELAY = 140;

private final List<Point> snake = new CopyOnWriteArrayList<>();

private Point apple;

private boolean leftDirection = false;

private boolean rightDirection = true;

private boolean upDirection = false;

private boolean downDirection = false;

private boolean inGame = true;

private Timer timer;

private int score = 0;

private String playerName;

private PrintWriter out;

private BufferedReader in;

// Images

private Image appleImage;

private Image headUp, headDown, headLeft, headRight;

private Image bodyHorizontal, bodyVertical;

private Image bodyTopLeft, bodyTopRight, bodyBottomLeft, bodyBottomRight;

private Image tailUp, tailDown, tailLeft, tailRight;

private Image backgroundImage;

public SnakeGame(String playerName) {

this.playerName = playerName;

loadImages();

initBoard();

initNetwork();

}

private void loadImages() {

try {

appleImage = new ImageIcon(getClass().getClassLoader().getResource("resources/apple.png")).getImage();

headUp = new ImageIcon(getClass().getClassLoader().getResource("resources/head\_up.png")).getImage();

headDown = new ImageIcon(getClass().getClassLoader().getResource("resources/head\_down.png")).getImage();

headLeft = new ImageIcon(getClass().getClassLoader().getResource("resources/head\_left.png")).getImage();

headRight = new ImageIcon(getClass().getClassLoader().getResource("resources/head\_right.png")).getImage();

bodyHorizontal = new ImageIcon(getClass().getClassLoader().getResource("resources/body\_horizontal.png")).getImage();

bodyVertical = new ImageIcon(getClass().getClassLoader().getResource("resources/body\_vertical.png")).getImage();

bodyTopLeft = new ImageIcon(getClass().getClassLoader().getResource("resources/body\_topleft.png")).getImage();

bodyTopRight = new ImageIcon(getClass().getClassLoader().getResource("resources/body\_topright.png")).getImage();

bodyBottomLeft = new ImageIcon(getClass().getClassLoader().getResource("resources/body\_bottomleft.png")).getImage();

bodyBottomRight = new ImageIcon(getClass().getClassLoader().getResource("resources/body\_bottomright.png")).getImage();

tailUp = new ImageIcon(getClass().getClassLoader().getResource("resources/tail\_up.png")).getImage();

tailDown = new ImageIcon(getClass().getClassLoader().getResource("resources/tail\_down.png")).getImage();

tailLeft = new ImageIcon(getClass().getClassLoader().getResource("resources/tail\_left.png")).getImage();

tailRight = new ImageIcon(getClass().getClassLoader().getResource("resources/tail\_right.png")).getImage();

backgroundImage = new ImageIcon(getClass().getClassLoader().getResource("resources/background.png")).getImage();

System.out.println("Images loaded successfully");

} catch (Exception e) {

System.out.println("Failed to load images");

e.printStackTrace();

}

}

private void initBoard() {

addKeyListener(new TAdapter());

setBackground(Color.white);

setFocusable(true);

setSize(WIDTH, HEIGHT);

setResizable(false);

setTitle("Snake Game");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

initGame();

repaint(); // Force repaint

}

private void initGame() {

snake.clear();

for (int z = 0; z < 3; z++) {

snake.add(new Point((50 / DOT\_SIZE) \* DOT\_SIZE - z \* DOT\_SIZE, (50 / DOT\_SIZE) \* DOT\_SIZE));

}

locateApple();

timer = new Timer(DELAY, this);

timer.start();

}

private void initNetwork() {

try {

Socket socket = new Socket("localhost", 12345);

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

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

new Thread(() -> {

String response;

try {

while ((response = in.readLine()) != null) {

System.out.println("Server: " + response);

}

} catch (IOException e) {

e.printStackTrace();

}

}).start();

} catch (IOException e) {

e.printStackTrace();

}

}

private void locateApple() {

int r = (int) (Math.random() \* RAND\_POS\_X);

int q = (int) (Math.random() \* RAND\_POS\_Y);

apple = new Point(r \* DOT\_SIZE, q \* DOT\_SIZE);

}

@Override

public void paint(Graphics g) {

super.paint(g);

if (backgroundImage != null) {

g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);

}

if (inGame) {

drawApple(g);

drawSnake(g);

drawScore(g);

} else {

gameOver(g); // Переместите вызов gameOver сюда

}

Toolkit.getDefaultToolkit().sync();

}

private void drawApple(Graphics g) {

g.drawImage(appleImage, apple.x, apple.y, this);

}

private void drawSnake(Graphics g) {

// Draw head

Point head = snake.get(0);

Image headImage = null;

if (leftDirection) headImage = headLeft;

else if (rightDirection) headImage = headRight;

else if (upDirection) headImage = headUp;

else if (downDirection) headImage = headDown;

g.drawImage(headImage, head.x, head.y, this);

// Draw body

for (int i = 1; i < snake.size() - 1; i++) {

Point p = snake.get(i);

Point next = snake.get(i + 1);

Point prev = snake.get(i - 1);

Image img = bodyHorizontal; // Default to horizontal body segment

if (prev.x == next.x) img = bodyVertical;

else if (prev.y == next.y) img = bodyHorizontal;

else if (prev.x < p.x && next.y < p.y || next.x < p.x && prev.y < p.y) img = bodyTopLeft;

else if (prev.x > p.x && next.y < p.y || next.x > p.x && prev.y < p.y) img = bodyTopRight;

else if (prev.x < p.x && next.y > p.y || next.x < p.x && prev.y > p.y) img = bodyBottomLeft;

else if (prev.x > p.x && next.y > p.y || next.x > p.x && prev.y > p.y) img = bodyBottomRight;

g.drawImage(img, p.x, p.y, this);

}

// Draw tail

Point tail = snake.get(snake.size() - 1);

Point tailPrev = snake.get(snake.size() - 2);

Image tailImage = null;

if (tailPrev.x < tail.x) tailImage = tailRight;

else if (tailPrev.x > tail.x) tailImage = tailLeft;

else if (tailPrev.y < tail.y) tailImage = tailDown;

else if (tailPrev.y > tail.y) tailImage = tailUp;

g.drawImage(tailImage, tail.x, tail.y, this);

}

private void drawScore(Graphics g) {

g.setColor(Color.black);

g.setFont(new Font("Helvetica", Font.BOLD, 14));

// Отображение счета внизу слева. Учитываем высоту строки, чтобы текст не находился слишком близко к краю окна.

g.drawString("Score: " + score, 10, HEIGHT - 10);

}

private void gameOver(Graphics g) {

String msg = score >= 20 ? "Congratulations! You scored 20!" : "Game Over";

String scoreMsg = "Final Score: " + score;

String replayMsg = "Press R to Restart";

Font font = new Font("Helvetica", Font.BOLD, 14);

g.setColor(Color.black);

g.setFont(font);

FontMetrics metr = getFontMetrics(font);

g.drawString(msg, (WIDTH - metr.stringWidth(msg)) / 2, HEIGHT / 2 - metr.getHeight());

g.drawString(scoreMsg, (WIDTH - metr.stringWidth(scoreMsg)) / 2, HEIGHT / 2);

g.drawString(replayMsg, (WIDTH - metr.stringWidth(replayMsg)) / 2, HEIGHT / 2 + metr.getHeight() + 10);

}

private void checkApple() {

Point head = snake.get(0);

if (head.equals(apple)) {

System.out.println("Apple eaten!");

snake.add(new Point(-1, -1)); // Добавление нового элемента к змейке

score += 2; // Увеличение счёта

locateApple(); // Перемещение яблока

if (score >= 20) {

inGame = false; // Завершить игру, если счёт достиг или превысил 50

sendScoreToServer(score); // Отправка счёта на сервер

}

}

}

private void move() {

for (int z = snake.size() - 1; z > 0; z--) {

snake.set(z, new Point(snake.get(z - 1)));

}

Point head = snake.get(0);

if (leftDirection) {

head.translate(-DOT\_SIZE, 0);

} else if (rightDirection) {

head.translate(DOT\_SIZE, 0);

} else if (upDirection) {

head.translate(0, -DOT\_SIZE);

} else if (downDirection) {

head.translate(0, DOT\_SIZE);

}

snake.set(0, head);

}

private void sendScoreToServer(int score) {

if (out != null) {

out.println("SCORE " + playerName + " " + score);

}

}

private void checkCollision() {

Point head = snake.get(0);

if (head.x >= WIDTH - 20 || head.x < 0 || head.y >= HEIGHT - 20 || head.y < 0) {

inGame = false;

sendScoreToServer(score);

}

for (int z = snake.size() - 1; z > 0; z--) {

if (snake.get(z).equals(head)) {

inGame = false;

sendScoreToServer(score);

}

}

if (!inGame) {

timer.stop();

repaint(); // Запрашивает перерисовку, gameOver будет вызван в методе paint

}

}

@Override

public void actionPerformed(ActionEvent e) {

if (inGame) {

checkApple();

checkCollision();

move();

}

repaint();

}

private class TAdapter extends KeyAdapter {

@Override

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

if ((key == KeyEvent.VK\_LEFT) && (!rightDirection)) {

leftDirection = true;

upDirection = false;

downDirection = false;

}

if ((key == KeyEvent.VK\_RIGHT) && (!leftDirection)) {

rightDirection = true;

upDirection = false;

downDirection = false;

}

if ((key == KeyEvent.VK\_UP) && (!downDirection)) {

upDirection = true;

rightDirection = false;

leftDirection = false;

downDirection = false;

}

if ((key == KeyEvent.VK\_DOWN) && (!upDirection)) {

downDirection = true;

rightDirection = false;

leftDirection = false;

}

if ((key == KeyEvent.VK\_R) && (!inGame)) {

inGame = true;

score = 0;

leftDirection = false;

rightDirection = true;

upDirection = false;

downDirection = false;

initGame();

}

}

}

public static void main(String[] args) {

String playerName = JOptionPane.showInputDialog("Enter your name:");

EventQueue.invokeLater(() -> {

JFrame ex = new SnakeGame(playerName);

ex.setVisible(true);

});

}

}

## Приложение A.2. Файл «NetworkClien.java» Server

package com.snakegame;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

public class NetworkClient implements Runnable {

private static final String SERVER\_ADDRESS = "localhost";

private static final int SERVER\_PORT = 12345;

@Override

public void run() {

try (Socket socket = new Socket(SERVER\_ADDRESS, SERVER\_PORT);

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

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

// Example: send a message to the server

out.println("GET\_SCORES");

String response;

while ((response = in.readLine()) != null) {

System.out.println("Server response: " + response);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

## Приложение A.3. Файл «NetwokrServer.java» Server

package com.snakegame.server;

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.concurrent.Executors;

import java.util.concurrent.ThreadPoolExecutor;

public class NetworkServer {

private static final int PORT = 12345;

public static void main(String[] args) {

ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.newCachedThreadPool();

try (ServerSocket serverSocket = new ServerSocket(PORT)) {

System.out.println("Server is listening on port " + PORT);

while (true) {

Socket socket = serverSocket.accept();

executor.execute(new ClientHandler(socket));

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

## Приложение A.4. Файл «ClientHandler.java» Server

package com.snakegame.server;

import java.io.\*;

import java.net.Socket;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

import java.util.stream.Collectors;

public class ClientHandler implements Runnable {

private static final Map<String, Integer> scores = new ConcurrentHashMap<>();

private final Socket socket;

public ClientHandler(Socket socket) {

this.socket = socket;

}

@Override

public void run() {

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

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

String clientMessage;

while ((clientMessage = in.readLine()) != null) {

System.out.println("Received: " + clientMessage);

if (clientMessage.startsWith("SCORE")) {

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

String playerName = parts[1];

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

scores.put(playerName, score);

out.println("Score received for " + playerName);

} else if (clientMessage.equals("GET\_SCORES")) {

String scoresList = scores.entrySet()

.stream()

.map(entry -> entry.getKey() + ": " + entry.getValue())

.collect(Collectors.joining(", "));

out.println("Scores: " + scoresList);

} else {

out.println("Unknown command");

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

# Приложение B. UML - диаграммы

## Приложение B.1. Диаграммы классов

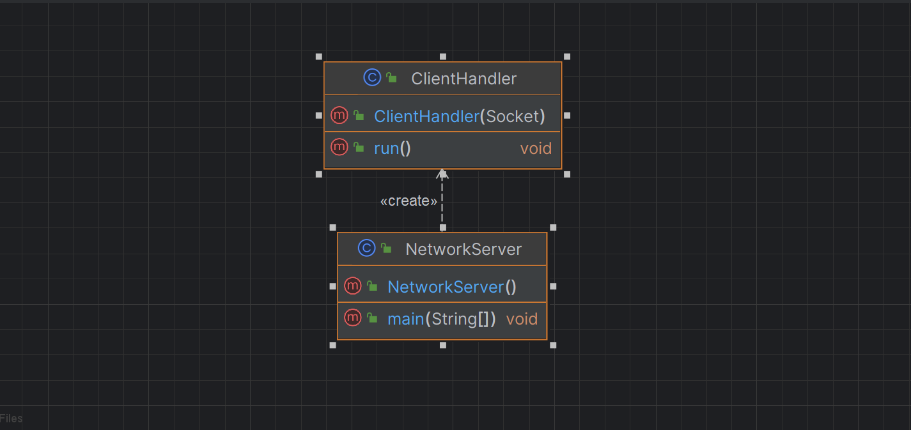


Рисунок 9 – UML - диаграмма классов сервера.

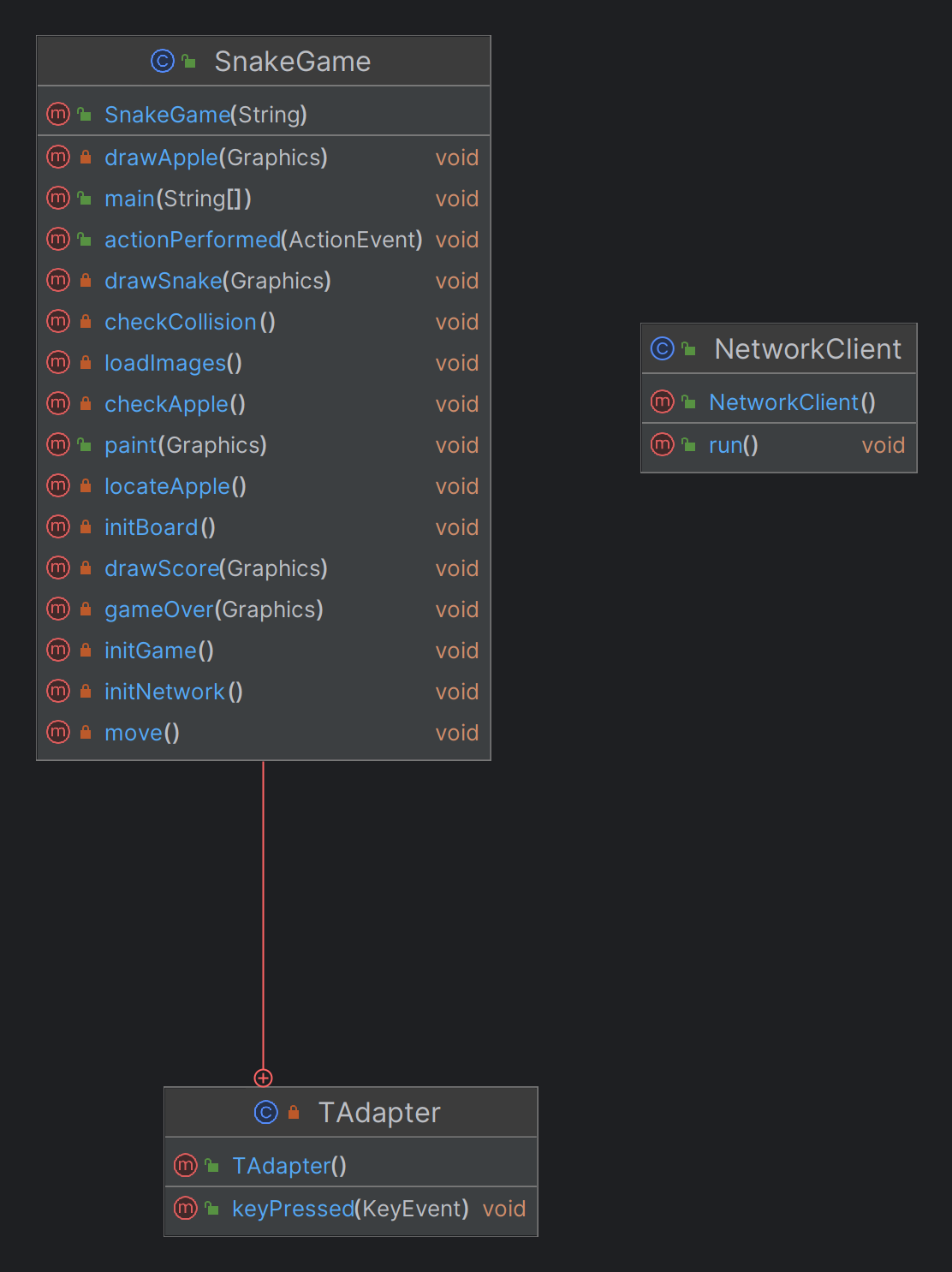


Рисунок 10 – UML - диаграмма классов клиента.

## Приложение B.2. Диаграмма развёртывания

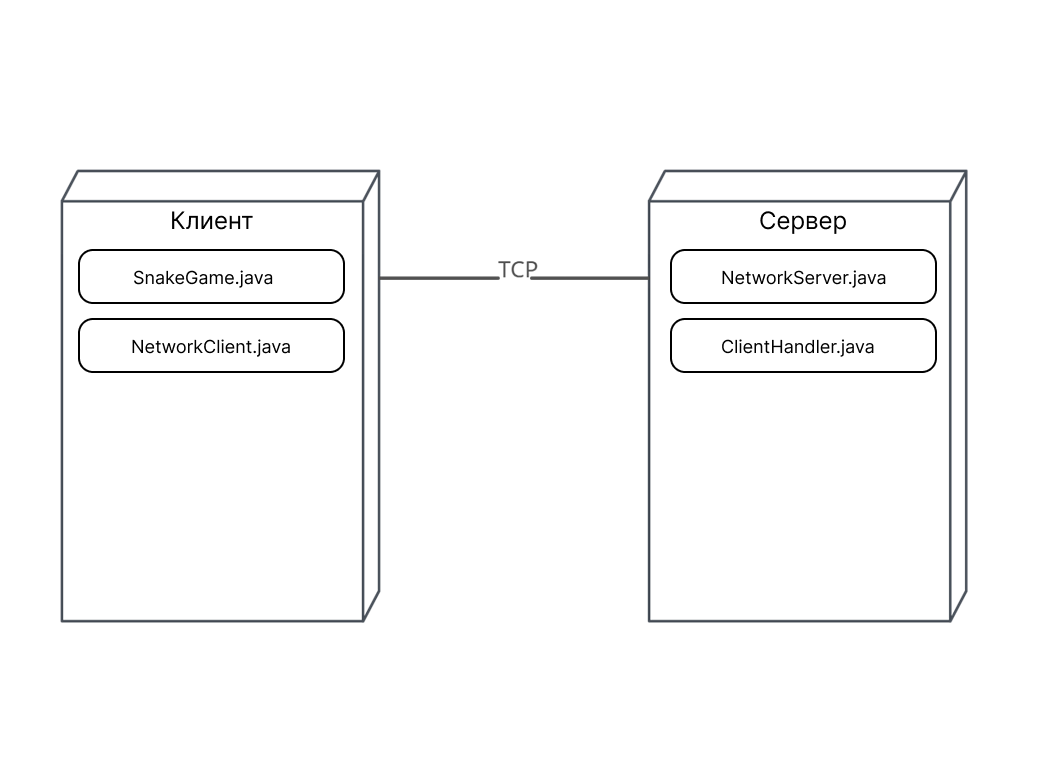


Рисунок 11 – UML - диаграмма развёртывания.

## Приложение B.3. Диаграмма последовательности

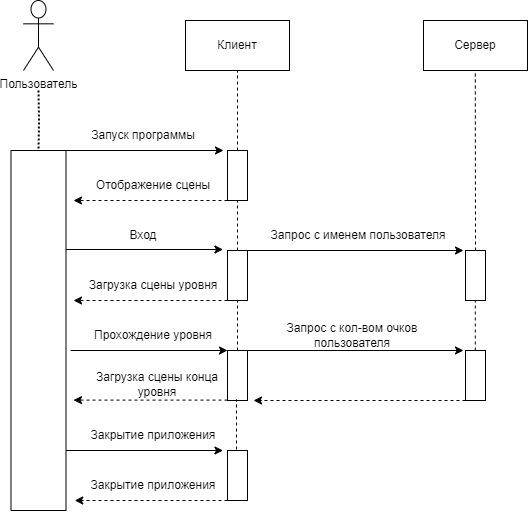


Рисунок 12 – UML - диаграмма последовательности.

## Приложение B.4. Диаграмма деятельности

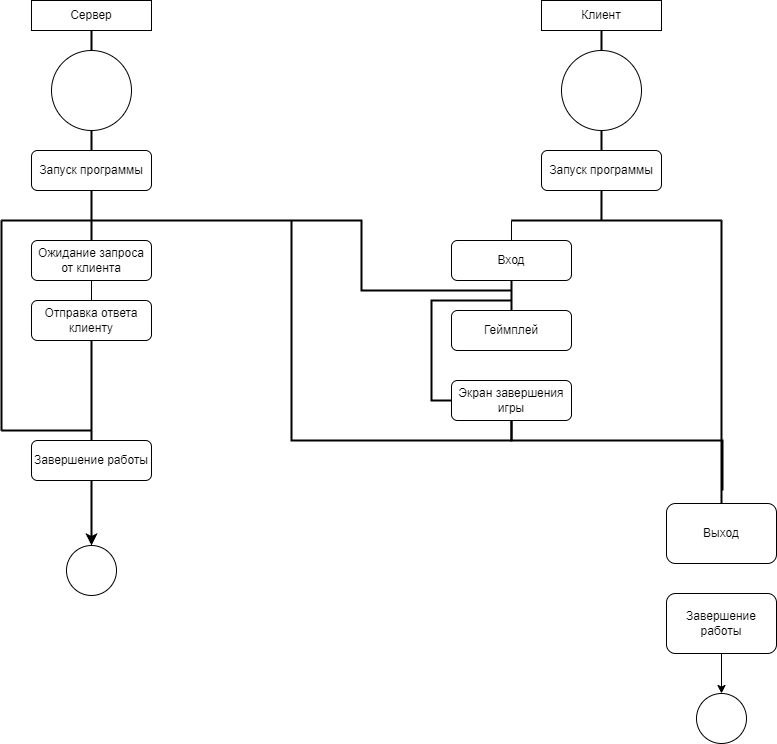


Рисунок 13 – UML - диаграмма деятельности.

## Приложение B.5. Диаграмма вариантов использования приложения

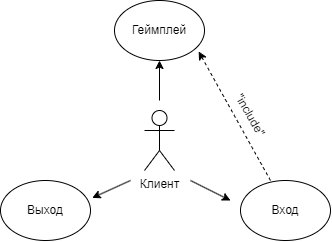


Рисунок 14 – UML – диаграмма вариантов использования приложения.