Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №6\_2**

**Дисциплина: Платформо-независимое программирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.С. Агаджанян

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В. И. Шиян

**Тема**: Синхронизация

**Задание**: реактивная змейка

**Описание работы программы:**

1. Программа запускается методом main(), который создает экземпляр класса SnakeGame и делает окно видимым. SwingUtilities.invokeLater() обеспечивает запуск графического интерфейса в отдельном потоке событий, что необходимо для корректной работы Swing.
2. При создании объекта SnakeGame инициализируется графический интерфейс, добавляются компоненты (панель для игры, метка для счета), регистрируется обработчик событий клавиатуры и вызывается метод startGame().
3. Метод startGame() инициализирует состояние игры: создает змейку в центре поля, генерирует первую еду, устанавливает начальное направление движения, устанавливает начальную скорость и запускает таймер gameTimer.
4. Ключевым моментом является запуск отдельного потока speedThread для управления скоростью игры. Этот поток выполняет следующие действия:
   * Ждет 5 секунд (Thread.sleep(5000)).
   * Синхронизирует доступ к общим переменным currentSpeed и isRunning с использованием блока synchronized (this).
   * Если игра запущена (isRunning) и текущая скорость больше максимальной скорости (currentSpeed > MAX\_SPEED), то скорость уменьшается на величину SPEED\_INCREMENT.
   * Таймер gameTimer останавливается, создается новый таймер с новой скоростью и запускается снова. Это обеспечивает динамическое изменение скорости игры.
   * Поток повторяет эти действия до тех пор, пока игра запущена.
5. Таймер gameTimer периодически вызывает метод move(), который перемещает змейку в соответствии с текущим направлением.
6. Метод move() также вызывает методы checkCollision() и checkFood() для проверки столкновений и поедания еды.
7. Обработчик событий клавиатуры реагирует на нажатия клавиш управления направлением движения змейки (UP, DOWN, LEFT, RIGHT) и на клавишу SPACE (пауза/возобновление игры).
8. Если змейка сталкивается со стеной или с самой собой, вызывается метод gameOver(), который завершает игру и отображает сообщение о проигрыше.

**Особенности реализации и синхронизация:**

* **Многопоточность:** Использование отдельного потока для управления скоростью позволяет избежать блокировки основного потока отрисовки и обработки пользовательского ввода. Это обеспечивает плавный и отзывчивый игровой процесс, даже когда скорость игры динамически изменяется.
* **Синхронизация:** Блок synchronized (this) используется для обеспечения безопасного доступа к общим переменным currentSpeed и isRunning между основным потоком игры и потоком управления скоростью. Без синхронизации возможно возникновение гонок данных, что может привести к непредсказуемому поведению программы (например, к некорректному изменению скорости или к завершению игры). Синхронизация гарантирует, что только один поток может изменять эти переменные в любой момент времени.
* **Обработка исключений:** В потоке управления скоростью обрабатывается исключение InterruptedException, которое может быть выброшено при вызове метода Thread.sleep(). Обработка исключения позволяет потоку корректно завершиться при прерывании. Thread.currentThread().interrupt() восстанавливает флаг прерывания потока, что позволяет другим частям программы узнать о прерывании.

Листинг:

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.util.LinkedList;

import java.util.Random;

public class SnakeGame extends JFrame {

    private static final int WIDTH = 600;

    private static final int HEIGHT = 600;

    private static final int CELL\_SIZE = 20;

    private static final int INITIAL\_SPEED = 200;

    private static final int SPEED\_INCREMENT = 10;

    private static final int MAX\_SPEED = 50;

    private JPanel gamePanel;

    private JLabel scoreLabel;

    private int score = 0;

    private LinkedList<Point> snake;

    private Point food;

    private Direction direction = Direction.RIGHT;

    private boolean isRunning = false;

    private Timer gameTimer;

    private int currentSpeed = INITIAL\_SPEED;

    private enum Direction {

        UP, DOWN, LEFT, RIGHT

    }

    public SnakeGame() {

        setTitle("Реактивный Змей");

        setSize(WIDTH, HEIGHT + 40); // +40 for the title bar and borders

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLocationRelativeTo(null);

        setResizable(false);

        gamePanel = new GamePanel();

        scoreLabel = new JLabel("Счет: 0");

        scoreLabel.setHorizontalAlignment(SwingConstants.CENTER);

        add(gamePanel, BorderLayout.CENTER);

        add(scoreLabel, BorderLayout.NORTH);

        addKeyListener(new KeyAdapter() {

            @Override

            public void keyPressed(KeyEvent e) {

                switch (e.getKeyCode()) {

                    case KeyEvent.VK\_UP:

                        if (direction != Direction.DOWN) {

                            direction = Direction.UP;

                        }

                        break;

                    case KeyEvent.VK\_DOWN:

                        if (direction != Direction.UP) {

                            direction = Direction.DOWN;

                        }

                        break;

                    case KeyEvent.VK\_LEFT:

                        if (direction != Direction.RIGHT) {

                            direction = Direction.LEFT;

                        }

                        break;

                    case KeyEvent.VK\_RIGHT:

                        if (direction != Direction.LEFT) {

                            direction = Direction.RIGHT;

                        }

                        break;

                    case KeyEvent.VK\_SPACE: // Пауза/Старт

                        togglePause();

                        break;

                }

            } // Закрывающая скобка, которой не было в предыдущей версии.

        });

        setFocusable(true);

        requestFocusInWindow();

        startGame();

    }

    private void startGame() {

        snake = new LinkedList<>();

        snake.add(new Point(WIDTH / 2 / CELL\_SIZE, HEIGHT / 2 / CELL\_SIZE));

        food = createFood();

        direction = Direction.RIGHT;

        isRunning = true;

        score = 0;

        scoreLabel.setText("Счет: 0");

        currentSpeed = INITIAL\_SPEED;

        createGameTimer(currentSpeed);

        gameTimer.start();

        Thread speedThread = new Thread(() -> {

            while (isRunning) {

                try {

                    Thread.sleep(5000);

                } catch (InterruptedException e) {

                    Thread.currentThread().interrupt();

                    return;

                }

                synchronized (this) {

                    if (currentSpeed > MAX\_SPEED && isRunning) {

                        currentSpeed -= SPEED\_INCREMENT;

                        gameTimer.stop();

                        createGameTimer(currentSpeed);

                        gameTimer.start();

                        System.out.println("Скорость увеличена: " + currentSpeed);

                    }

                }

            }

        });

        speedThread.start();

    }

    private void createGameTimer(int speed) {

        gameTimer = new Timer(speed, new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e) {

                if (isRunning) {

                    move();

                    checkCollision();

                    checkFood();

                    gamePanel.repaint();

                }

            }

        });

    }

    private void togglePause() {

        if (isRunning) {

            isRunning = false;

            gameTimer.stop();

        } else {

            isRunning = true;

            gameTimer.start();

        }

        requestFocusInWindow();

    }

    private void move() {

        Point head = snake.getFirst();

        Point newHead = null;

        switch (direction) {

            case UP:

                newHead = new Point(head.x, head.y - 1);

                break;

            case DOWN:

                newHead = new Point(head.x, head.y + 1);

                break;

            case LEFT:

                newHead = new Point(head.x - 1, head.y);

                break;

            case RIGHT:

                newHead = new Point(head.x + 1, head.y);

                break;

        }

        snake.addFirst(newHead);

        if (!newHead.equals(food)) {

            snake.removeLast();

        } else {

            food = createFood();

            score += 10;

            scoreLabel.setText("Счет: " + score);

        }

    }

    private void checkCollision() {

        Point head = snake.getFirst();

        // Check for wall collisions

        if (head.x < 0 || head.x >= WIDTH / CELL\_SIZE || head.y < 0 || head.y >= HEIGHT / CELL\_SIZE) {

            gameOver();

            return;

        }

        // Check for self-collision

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

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

                gameOver();

                return;

            }

        }

    }

    private void checkFood() {

        Point head = snake.getFirst();

        if (head.equals(food)) {

            food = createFood();

            score += 10;

            scoreLabel.setText("Счет: " + score);

        }

    }

   private Point createFood() {

        Random random = new Random();

        int maxX = WIDTH / CELL\_SIZE;

        int maxY = HEIGHT / CELL\_SIZE;

        Point newFood;

        do {

            newFood = new Point(random.nextInt(maxX), random.nextInt(maxY));

        } while (snake.contains(newFood));

        return newFood;

    }

    private void gameOver() {

        isRunning = false;

        gameTimer.stop();

        JOptionPane.showMessageDialog(this, "Игра окончена! Ваш счет: " + score);

        startGame();

    }

    class GamePanel extends JPanel {

        @Override

        protected void paintComponent(Graphics g) {

            super.paintComponent(g);

            draw(g);

        }

        private void draw(Graphics g) {

            // Fill background

            g.setColor(Color.BLACK);

            g.fillRect(0, 0, WIDTH, HEIGHT);

            // Draw snake

            g.setColor(Color.GREEN);

            for (Point point : snake) {

                g.fillRect(point.x \* CELL\_SIZE, point.y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

            }

            // Draw food

            g.setColor(Color.RED);

            g.fillRect(food.x \* CELL\_SIZE, food.y \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

        }

    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(() -> {

            SnakeGame game = new SnakeGame();

            game.setVisible(true);

        });

    }

}

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Контент, сгенерированный ИИ, может содержать ошибки.

**Вывод:**

В ходе выполнения лабораторной работы была разработана игра “Змейка” с использованием многопоточности для управления скоростью игры. При этом была обеспечена синхронизация доступа к общим ресурсам с помощью блока synchronized, что позволило избежать гонок данных и обеспечить корректную работу игры. Получены навыки работы с графическим интерфейсом Swing, обработки событий клавиатуры, использования таймеров и многопоточности для решения задач, требующих параллельного выполнения. Данная работа позволила закрепить знания о принципах синхронизации потоков и их практическом применении.