| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка  ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  **Кафедра програмних систем і технологій**                Дисципліна  **« Кросплатформне програмування,»**      **Лабораторна робота № 3**  **на тему:**  **"Гра Життя"** | | | |
| --- | --- | --- | --- |
| **Виконав:** | Безруков Андрій Миколайович | **Перевірив**: | Васильєв Олексій Миколайович |
| Група | ІПЗ-33 | Дата перевірки |  |
| Форма навчання | денна | Оцінка |  |
| Спеціальність | 121 |
| 2024 | | | |

## **1. Постановка задачі**

## Необхідно створити Java-застосунок, що реалізує клітинний автомат «Гра Життя» (Conway’s Game of Life):

## «Всесвіт» представлений двовимірною сіткою клітин, кожна клітина може бути живою або мертвою.

## Кожна клітина має вісім сусідів (по горизонталі, вертикалі та діагоналях).

## Початкове розміщення живих клітин задає перше покоління.

## Кожне наступне покоління обчислюється за правилами:

## Жива клітина з менш ніж двома живими сусідами гине (недостатня популяція).

## Жива клітина з двома або трьома живими сусідами продовжує жити.

## Жива клітина з більше ніж трьома живими сусідами гине (перенаселення).

## Мертва клітина з рівно трьома живими сусідами оживає (репродукція).

## Застосунок повинен мати два режими:

## Графічний інтерфейс (GUI) з можливістю:

## кліком миші переключати стан клітин;

## керувати симуляцією кнопками Start, Stop, Clear, Load from File;

## візуалізувати покоління в реальному часі з таймером.

## Консольна версія: введення координат клітин вручну або з файлу та відображення поколінь у терміналі.

## Показати роботу програми в ОС Windows та Linux.

## 

## **2. Опис реалізації та програмний код**

## У реалізації використано javax.swing для GUI та клас Timer для анімації. Консольну версію реалізовано окремим методом.

## package lab3;

## 

## import javax.swing.\*;

## import java.awt.\*;

## import java.awt.event.\*;

## import java.io.BufferedReader;

## import java.io.FileReader;

## import java.io.IOException;

## import java.util.Scanner;

## 

## public class GameOfLife extends JFrame {

## private static final int GRID\_HEIGHT = 7;

## private static final int GRID\_LENGTH = 8;

## private static final int CELL\_SIZE = 30;

## private boolean[][] grid = new boolean[GRID\_HEIGHT+1][GRID\_LENGTH+1];

## private Timer timer;

## private JPanel gridPanel;

## 

## public GameOfLife() {

## setTitle("Conway's Game of Life");

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

## setLayout(new BorderLayout());

## 

## // Grid panel

## gridPanel = new JPanel() {

## @Override

## protected void paintComponent(Graphics g) {

## super.paintComponent(g);

## drawGrid(g);

## }

## };

## gridPanel.setPreferredSize(new Dimension(GRID\_LENGTH \* CELL\_SIZE, GRID\_HEIGHT \* CELL\_SIZE));

## add(gridPanel, BorderLayout.CENTER);

## 

## // Control panel

## JPanel controlPanel = new JPanel();

## JButton startButton = new JButton("Start");

## JButton stopButton = new JButton("Stop");

## JButton clearButton = new JButton("Clear");

## JButton loadButton = new JButton("Load from File");

## 

## controlPanel.add(startButton);

## controlPanel.add(stopButton);

## controlPanel.add(clearButton);

## controlPanel.add(loadButton);

## add(controlPanel, BorderLayout.SOUTH);

## 

## // Add mouse listener to toggle cells

## gridPanel.addMouseListener(new MouseAdapter() {

## @Override

## public void mouseClicked(MouseEvent e) {

## int x = e.getX() / CELL\_SIZE + 1;

## int y = e.getY() / CELL\_SIZE + 1;

## 

## if (x >= 1 && x < GRID\_LENGTH && y >= 1 && y < GRID\_HEIGHT) {

## grid[y][x] = !grid[y][x];

## gridPanel.repaint();

## }

## }

## });

## 

## // Timer for animation

## timer = new Timer(200, new ActionListener() {

## @Override

## public void actionPerformed(ActionEvent e) {

## determineState();

## gridPanel.repaint();

## }

## });

## 

## // Button actions

## startButton.addActionListener(e -> timer.start());

## stopButton.addActionListener(e -> timer.stop());

## clearButton.addActionListener(e -> {

## clearGrid();

## gridPanel.repaint();

## });

## loadButton.addActionListener(e -> loadFromFile());

## 

## // Display rules when starting

## showRules();

## 

## pack();

## setLocationRelativeTo(null);

## setVisible(true);

## }

## 

## private void showRules() {

## JOptionPane.showMessageDialog(this,

## "THE GAME OF LIFE - Implementation in Java\n\n" +

## "Also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970.\n\n" +

## "Rules:\n" +

## "The universe of the Game of Life is a two-dimensional grid of square cells,\n" +

## "each of which is in one of two possible states, alive or dead.\n" +

## "Every cell interacts with its eight neighbors, which are the cells that are horizontally, vertically, or diagonally adjacent.\n\n" +

## "At each step in time, the following transitions occur:\n" +

## "1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.\n" +

## "2. Any live cell with two or three live neighbors lives on to the next generation.\n" +

## "3. Any live cell with more than three live neighbors dies, as if by over-population.\n" +

## "4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.\n\n" +

## "Click on cells to toggle them alive/dead, then press Start to begin the simulation.",

## "Game Rules",

## JOptionPane.INFORMATION\_MESSAGE);

## }

## 

## private void drawGrid(Graphics g) {

## // Draw cells

## for (int y = 1; y < GRID\_HEIGHT; y++) {

## for (int x = 1; x < GRID\_LENGTH; x++) {

## if (grid[y][x]) {

## g.setColor(Color.GREEN);

## g.fillRect((x-1) \* CELL\_SIZE, (y-1) \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

## }

## g.setColor(Color.GRAY);

## g.drawRect((x-1) \* CELL\_SIZE, (y-1) \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE);

## }

## }

## }

## 

## private void determineState() {

## boolean[][] gridCopy = new boolean[GRID\_HEIGHT+1][GRID\_LENGTH+1];

## copyGrid(grid, gridCopy);

## 

## for (int y = 1; y < GRID\_HEIGHT; y++) {

## for (int x = 1; x < GRID\_LENGTH; x++) {

## int alive = 0;

## 

## // Check all 8 neighbors

## for (int dy = -1; dy <= 1; dy++) {

## for (int dx = -1; dx <= 1; dx++) {

## if (!(dx == 0 && dy == 0)) {

## if (gridCopy[y+dy][x+dx]) {

## alive++;

## }

## }

## }

## }

## 

## // Apply rules

## if (gridCopy[y][x]) {

## // Live cell

## if (alive < 2 || alive > 3) {

## grid[y][x] = false; // Dies

## }

## } else {

## // Dead cell

## if (alive == 3) {

## grid[y][x] = true; // Becomes alive

## }

## }

## }

## }

## }

## 

## private void copyGrid(boolean[][] source, boolean[][] destination) {

## for (int y = 0; y < GRID\_HEIGHT+1; y++) {

## for (int x = 0; x < GRID\_LENGTH+1; x++) {

## destination[y][x] = source[y][x];

## }

## }

## }

## 

## private void clearGrid() {

## for (int y = 0; y < GRID\_HEIGHT+1; y++) {

## for (int x = 0; x < GRID\_LENGTH+1; x++) {

## grid[y][x] = false;

## }

## }

## }

## 

## private void loadFromFile() {

## JFileChooser fileChooser = new JFileChooser();

## int result = fileChooser.showOpenDialog(this);

## 

## if (result == JFileChooser.APPROVE\_OPTION) {

## clearGrid();

## try (BufferedReader reader = new BufferedReader(new FileReader(fileChooser.getSelectedFile()))) {

## String line;

## while ((line = reader.readLine()) != null) {

## String[] coords = line.split(" ");

## if (coords.length >= 2) {

## int x = Integer.parseInt(coords[0]);

## int y = Integer.parseInt(coords[1]);

## 

## if (x >= 1 && x < GRID\_LENGTH && y >= 1 && y < GRID\_HEIGHT) {

## grid[y][x] = true;

## }

## }

## }

## gridPanel.repaint();

## } catch (IOException e) {

## JOptionPane.showMessageDialog(this,

## "Error reading file: " + e.getMessage(),

## "File Error",

## JOptionPane.ERROR\_MESSAGE);

## }

## }

## }

## 

## // Command-line version (similar to original C++ code)

## public static void runConsoleVersion() {

## Scanner scanner = new Scanner(System.in);

## boolean[][] grid = new boolean[GRID\_HEIGHT+1][GRID\_LENGTH+1];

## 

## // Display introduction

## System.out.println("\u001B[32m"); // Green text

## System.out.println(" THE GAME OF LIFE - Implementation in Java");

## System.out.println();

## System.out.println("Also known simply as life,");

## System.out.println("is a cellular automaton devised by the British mathematician John Horton Conway in 1970.");

## System.out.println();

## System.out.println("Rules");

## System.out.println("The universe of the Game of life is an infinite two-dimensional orthogonal grid of square cells,");

## System.out.println("each of which is in one of two possible states, life or dead. Every");

## System.out.println("cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent.");

## System.out.println("At each step in time, the following transitions occur:");

## System.out.println("1. Any live cell with fewer than two live neighbours dies, as if caused by under-population.");

## System.out.println("2. Any live cell with two or three live neighbours lives on to the next generation.");

## System.out.println("3. Any live cell with more than three live neighbours dies, as if by over-population.");

## System.out.println("4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.");

## System.out.println();

## System.out.println("O - living cell");

## System.out.println(". - dead cell");

## System.out.println();

## 

## System.out.print("Enter the number of cells, or 'r' to read cells from file: ");

## String input = scanner.next();

## 

## if (input.equals("r")) {

## while (true) {

## System.out.println("Enter name of file to read from:");

## String filename = scanner.next();

## 

## try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {

## String line;

## while ((line = reader.readLine()) != null) {

## String[] coords = line.split(" ");

## if (coords.length >= 2) {

## int x = Integer.parseInt(coords[0]);

## int y = Integer.parseInt(coords[1]);

## grid[y][x] = true;

## }

## }

## break;

## } catch (IOException e) {

## System.out.println("No such file, try again.");

## }

## }

## } else {

## int numCells = Integer.parseInt(input);

## for (int i = 0; i < numCells; i++) {

## System.out.print("Enter the coordinates of cell " + (i+1) + " : ");

## int x = scanner.nextInt();

## int y = scanner.nextInt();

## grid[y][x] = true;

## printGrid(grid);

## }

## }

## 

## System.out.println("Grid setup is done. Start the game? (y/n)");

## printGrid(grid);

## String start = scanner.next();

## 

## if (start.equals("y") || start.equals("Y")) {

## try {

## while (true) {

## printGrid(grid);

## determineStateConsole(grid);

## Thread.sleep(200);

## clearConsoleScreen();

## }

## } catch (InterruptedException e) {

## System.out.println("Game interrupted.");

## }

## } else {

## System.out.println("\u001B[0m"); // Reset color

## clearConsoleScreen();

## }

## 

## scanner.close();

## }

## 

## private static void printGrid(boolean[][] grid) {

## for (int y = 1; y < GRID\_HEIGHT; y++) {

## for (int x = 1; x < GRID\_LENGTH; x++) {

## System.out.print(grid[y][x] ? " O " : " . ");

## if (x == GRID\_LENGTH - 1) {

## System.out.println();

## }

## }

## }

## }

## 

## private static void determineStateConsole(boolean[][] grid) {

## boolean[][] gridCopy = new boolean[GRID\_HEIGHT+1][GRID\_LENGTH+1];

## 

## // Copy grid

## for (int y = 0; y < GRID\_HEIGHT+1; y++) {

## for (int x = 0; x < GRID\_LENGTH+1; x++) {

## gridCopy[y][x] = grid[y][x];

## }

## }

## 

## for (int y = 1; y < GRID\_HEIGHT; y++) {

## for (int x = 1; x < GRID\_LENGTH; x++) {

## int alive = 0;

## 

## for (int dy = -1; dy <= 1; dy++) {

## for (int dx = -1; dx <= 1; dx++) {

## if (!(dx == 0 && dy == 0)) {

## if (gridCopy[y+dy][x+dx]) {

## alive++;

## }

## }

## }

## }

## 

## if (alive < 2) {

## grid[y][x] = false;

## } else if (alive == 3) {

## grid[y][x] = true;

## } else if (alive > 3) {

## grid[y][x] = false;

## }

## }

## }

## }

## 

## private static void clearConsoleScreen() {

## String os = System.getProperty("os.name").toLowerCase();

## 

## try {

## if (os.contains("win")) {

## new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();

## } else {

## System.out.print("\033[H\033[2J");

## System.out.flush();

## }

## } catch (Exception e) {

## // Fallback if clearing fails

## for (int i = 0; i < 50; i++) {

## System.out.println();

## }

## }

## }

## 

## public static void main(String[] args) {

## if (args.length > 0 && args[0].equals("--console")) {

## runConsoleVersion();

## } else {

## SwingUtilities.invokeLater(() -> new GameOfLife());

## }

## }

## }

## 

## 

## **Пояснення коду:**

## **GUI:** побудовано на Swing з JFrame та кастомним JPanel для сітки.

## **Таймер:** автоматичне оновлення поколінь кожні 200 мс.

## **Обчислення покоління:** копіювання поточного стану та застосування правил Конвея.

## **Завантаження:** читання координат із файла й оновлення сітки.

## **Консольна версія:** підтримує введення вручну та завантаження з файла, відображення символами у терміналі.

## 

## **3. Результати тестування**

## Застосунок перевірено на:

## **Windows 10** (OpenJDK 17) — обертання поколінь працює коректно; файл з координатами завантажується без помилок.

## **Ubuntu 22.04 LTS** (OpenJDK 17) — GUI та консольна версія функціонують стабільно; очищення консолі працює через ANSI-коди.

## У GUI-режимі створено декілька фігур ("глайдер", "блоки"), симуляція відповідає очікуванням правил.

## 

## **4. Висновки**

## Розроблено крос‑платформний застосунок Game of Life на Java з двома режимами: GUI та консольним. Програма:

## демонструє всі чотири правила Конвея;

## підтримує інтерактивне редагування та завантаження початкового покоління;

## забезпечує плавну анімацію в GUI та коректне відображення в консолі.

## Тестування показало стабільну роботу в Windows та Linux. Подальші вдосконалення: розширити розмір сітки, додати налаштування швидкості та збереження/відновлення конфігурацій.

## 