МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
**Кафедра програмних систем і технологій**

Дисципліна
**« Кросплатформне програмування,»**

**Лабораторна робота № 4**
**на тему:**
**"Гра Криниця-Ножиці-Папір"**

| | | | |
|---|---|---|---|
| **Виконав:** | Безруков Андрій Миколайович | **Перевірив**: | Васильєв Олексій Миколайович |
| Група | ІПЗ-33 | Дата перевірки | |
| Форма навчання | денна | Оцінка | |
| Спеціальність | 121 | | |
| 2024 | | | |

# 1. Постановка задачі

Розробити Java- застосунок «Криниця- Ножиці- Папір», у якому користувач грає проти комп'ютера:

1. На початку користувач визначає кількість ігрових сеансів.
2. Перед грою вибирається режим поведінки комп'ютера:
    1. випадкові ходи;
    2. з урахуванням ходів користувача поточного сеансу;
    3. з урахуванням історичних ходів (читання з файлу).
3. Після кожного сеансу результат (перемога/поразка/нічиї) і ходи користувача записуються у файл `game_stats.txt`.
4. Після завершення сеансів користувачу пропонується переглянути загальну статистику.

# 2. Опис реалізації та програмний код

У програмі реалізовано наступні ключові компоненти:

```
package lab4;

import java.io.*;
import java.util.*;

public class lab4 {
   private static final String[] CHOICES = {"WELL", "SCISSORS",
"PAPER"};
   private static final String STATS_FILE = "game_stats.txt";
   private static Scanner scanner = new Scanner(System.in);
   private static Random random = new Random();
   private static List<Integer> userMoves = new ArrayList<>();

   public static void main(String[] args) {
       System.out.println("Welcome to the Well-Scissors-Paper game!");

       // Get number of sessions
       System.out.print("Enter the number of sessions to play: ");
       int sessions = getValidIntInput();
```

```
        // Choose computer mode
        System.out.println("\nChoose computer mode:");
        System.out.println("1. Random moves");
        System.out.println("2. Consider user moves during current
session");
        System.out.println("3. Consider user moves from previous
sessions");
        System.out.print("Your choice (1-3): ");
        int mode = getValidModeInput();

        // Game stats
        int userWins = 0;
        int computerWins = 0;
        int draws = 0;

        // Load historical data if mode 3
        Map<Integer, Integer> historicalMoves = new HashMap<>();
        if (mode == 3) {
            loadHistoricalMoves(historicalMoves);
        }

        // Play sessions
        for (int i = 0; i < sessions; i++) {
            System.out.println("\n--- Session " + (i+1) + " ---");
            int result = playSession(mode, historicalMoves);

            if (result > 0) {
                userWins++;
                System.out.println("You win this session!");
            } else if (result < 0) {
                computerWins++;
                System.out.println("Computer wins this session!");
            } else {
                draws++;
                System.out.println("This session is a draw!");
            }
        }

        // Save results to file
        saveResults(sessions, userWins, computerWins, draws, userMoves);

        // Show final stats
        System.out.println("\n--- Game Results ---");
```

```
        System.out.println("Sessions played: " + sessions);
        System.out.println("Your wins: " + userWins);
        System.out.println("Computer wins: " + computerWins);
        System.out.println("Draws: " + draws);

        // Ask if user wants to see statistics
        showStatisticsOption();
    }

    private static int playSession(int mode, Map<Integer, Integer>
historicalMoves) {
        int sessionResult = 0; // 0 for draw, 1 for user win, -1 for
computer win

        while (true) {
            // Get user move
            System.out.println("\nMake your move:");
            System.out.println("1. Well");
            System.out.println("2. Scissors");
            System.out.println("3. Paper");
            System.out.print("Your choice (1-3): ");
            int userMove = getValidMoveInput() - 1; // 0-based index
            userMoves.add(userMove);

            // Get computer move based on mode
            int computerMove = getComputerMove(mode, historicalMoves);

            System.out.println("You chose: " + CHOICES[userMove]);
            System.out.println("Computer chose: " +
CHOICES[computerMove]);

            // Determine winner
            int roundResult = determineWinner(userMove, computerMove);

            if (roundResult == 0) {
                System.out.println("It's a tie! Play again.");
            } else {
                sessionResult = roundResult;
                break;
            }
        }

        return sessionResult;
```

```
    }

    private static int getComputerMove(int mode, Map<Integer, Integer>
historicalMoves) {
        switch (mode) {
            case 1: // Random mode
                return random.nextInt(3);

            case 2: // Consider current session
                if (userMoves.isEmpty()) {
                    return random.nextInt(3);
                } else {
                    // Simple strategy: counter the most frequent move
                    int[] counts = new int[3];
                    for (int move : userMoves) {
                        counts[move]++;
                    }

                    int mostFrequentMove = 0;
                    for (int i = 1; i < 3; i++) {
                        if (counts[i] > counts[mostFrequentMove]) {
                            mostFrequentMove = i;
                        }
                    }

                    // Return counter move (the move that beats the
predicted move)
                    return (mostFrequentMove + 1) % 3;
                }

            case 3: // Consider historical data
                if (historicalMoves.isEmpty()) {
                    return random.nextInt(3);
                } else {
                    // Find most frequent historical move
                    int mostFrequentMove = 0;
                    int maxCount = 0;

                    for (Map.Entry<Integer, Integer> entry :
historicalMoves.entrySet()) {
                        if (entry.getValue() > maxCount) {
                            maxCount = entry.getValue();
                            mostFrequentMove = entry.getKey();
```

```
                        }
                    }

                    // Return counter move
                    return (mostFrequentMove + 1) % 3;
                }

            default:
                return random.nextInt(3);
        }
    }

    private static int determineWinner(int userMove, int computerMove) {
        // 0: Well, 1: Scissors, 2: Paper
        // Well beats Scissors, Scissors beats Paper, Paper beats Well

        if (userMove == computerMove) {
            return 0; // Draw
        }

        if ((userMove == 0 && computerMove == 1) || // Well beats
Scissors
            (userMove == 1 && computerMove == 2) || // Scissors beats
Paper
            (userMove == 2 && computerMove == 0)) { // Paper beats Well
            return 1; // User wins
        } else {
            return -1; // Computer wins
        }
    }

    private static void loadHistoricalMoves(Map<Integer, Integer>
historicalMoves) {
        try (BufferedReader reader = new BufferedReader(new
FileReader(STATS_FILE))) {
            String line;
            boolean readingMoves = false;

            while ((line = reader.readLine()) != null) {
                if (line.startsWith("User moves:")) {
                    readingMoves = true;
                    continue;
                }
```

```
                if (readingMoves && !line.isEmpty() &&
Character.isDigit(line.charAt(0))) {
                    int move = Integer.parseInt(line);
                    historicalMoves.put(move,
historicalMoves.getOrDefault(move, 0) + 1);
                }
            }

            System.out.println("Loaded historical data: " +
historicalMoves.size() + " moves");
        } catch (IOException e) {
            System.out.println("No historical data found or error
reading file.");
        }
    }

    private static void saveResults(int sessions, int userWins, int
computerWins, int draws, List<Integer> userMoves) {
        try (PrintWriter writer = new PrintWriter(new
FileWriter(STATS_FILE, true))) {
            // Write session info
            writer.println("--- Game Session ---");
            writer.println("Date: " + new Date());
            writer.println("Sessions played: " + sessions);
            writer.println("User wins: " + userWins);
            writer.println("Computer wins: " + computerWins);
            writer.println("Draws: " + draws);

            // Write user moves for future analysis
            writer.println("User moves:");
            for (int move : userMoves) {
                writer.println(move);
            }
            writer.println(); // Empty line between sessions

            System.out.println("\nGame results saved to " + STATS_FILE);
        } catch (IOException e) {
            System.out.println("Error saving game results: " +
e.getMessage());
        }
    }
```

```
    private static void showStatisticsOption() {
        System.out.print("\nDo you want to see game statistics? (y/n):
");
        String response = scanner.nextLine().trim().toLowerCase();

        if (response.equals("y") || response.equals("yes")) {
            showStatistics();
        }
    }

    private static void showStatistics() {
        try (BufferedReader reader = new BufferedReader(new
FileReader(STATS_FILE))) {
            System.out.println("\n----- Game Statistics -----");

            int totalGames = 0;
            int totalUserWins = 0;
            int totalComputerWins = 0;
            int totalDraws = 0;

            String line;
            while ((line = reader.readLine()) != null) {
                if (line.startsWith("Sessions played:")) {
                    totalGames += Integer.parseInt(line.split(": ")[1]);
                } else if (line.startsWith("User wins:")) {
                    totalUserWins += Integer.parseInt(line.split(":
")[1]);
                } else if (line.startsWith("Computer wins:")) {
                    totalComputerWins += Integer.parseInt(line.split(":
")[1]);
                } else if (line.startsWith("Draws:")) {
                    totalDraws += Integer.parseInt(line.split(": ")[1]);
                }
            }

            System.out.println("Total games played: " + totalGames);
            System.out.println("Total user wins: " + totalUserWins + "
(" +
                               calculatePercentage(totalUserWins,
totalGames) + "%)");
            System.out.println("Total computer wins: " +
totalComputerWins + " (" +
```

```
                            calculatePercentage(totalComputerWins,
totalGames) + "%)");
            System.out.println("Total draws: " + totalDraws + " (" +
                            calculatePercentage(totalDraws,
totalGames) + "%)");

        } catch (IOException e) {
            System.out.println("No statistics available or error reading
file.");
        }
    }

    private static double calculatePercentage(int value, int total) {
        if (total == 0) return 0.0;
        return Math.round((double) value / total * 100 * 10) / 10.0;
    }

    private static int getValidIntInput() {
        while (true) {
            try {
                int value = Integer.parseInt(scanner.nextLine().trim());
                if (value > 0) {
                    return value;
                } else {
                    System.out.print("Please enter a positive number:
");
                }
            } catch (NumberFormatException e) {
                System.out.print("Invalid input. Please enter a number:
");
            }
        }
    }

    private static int getValidModeInput() {
        while (true) {
            try {
                int value = Integer.parseInt(scanner.nextLine().trim());
                if (value >= 1 && value <= 3) {
                    return value;
                } else {
                    System.out.print("Please enter a number between 1
and 3: ");
```

```
                }
            } catch (NumberFormatException e) {
                System.out.print("Invalid input. Please enter a number:
");
            }
        }
    }

    private static int getValidMoveInput() {
        while (true) {
            try {
                int value = Integer.parseInt(scanner.nextLine().trim());
                if (value >= 1 && value <= 3) {
                    return value;
                } else {
                    System.out.print("Please enter a number between 1
and 3: ");
                }
            } catch (NumberFormatException e) {
                System.out.print("Invalid input. Please enter a number:
");
            }
        }
    }
}
private static int playSession(int mode, Map<Integer, Integer>
historicalMoves) {
        int sessionResult = 0; // 0 for draw, 1 for user win, -1 for
computer win

        while (true) {
            // Get user move
            System.out.println("\nMake your move:");
            System.out.println("1. Well");
            System.out.println("2. Scissors");
            System.out.println("3. Paper");
            System.out.print("Your choice (1-3): ");
            int userMove = getValidMoveInput() - 1; // 0-based index
            userMoves.add(userMove);

            // Get computer move based on mode
            int computerMove = getComputerMove(mode, historicalMoves);
```

```
            System.out.println("You chose: " + CHOICES[userMove]);
            System.out.println("Computer chose: " +
CHOICES[computerMove]);

            // Determine winner
            int roundResult = determineWinner(userMove, computerMove);

            if (roundResult == 0) {
                System.out.println("It's a tie! Play again.");
            } else {
                sessionResult = roundResult;
                break;
            }
        }

        return sessionResult;
    }

    private static int getComputerMove(int mode, Map<Integer, Integer>
historicalMoves) {
        switch (mode) {
            case 1: // Random mode
                return random.nextInt(3);

            case 2: // Consider current session
                if (userMoves.isEmpty()) {
                    return random.nextInt(3);
                } else {
                    // Simple strategy: counter the most frequent move
                    int[] counts = new int[3];
                    for (int move : userMoves) {
                        counts[move]++;
```

**Пояснення основних методів:**

- `promptSessions()` і `promptMode()` забезпечують коректний ввід кількості сеансів та режиму.
- `playSession(...)` реалізує гру до визначення результату одного сеансу з повторами у разі нічиєї.

- `getComputerMove(...)` враховує заданий режим комп’ютера: випадковий вибір, аналіз поточної чи історичної статистики ходів.
- `saveResults(...)` додає інформацію про кожний сеанс до файлу `game_stats.txt` із датою, результатами та списком ходів користувача.
- `showStatisticsOption()` та `showStatistics()` дають змогу переглянути загальні показники (загальна кількість ігор, % перемог/поразок/нічиїх).

# 3. Результати тестування

Програма протестована у середовищах:

- **Windows 10** (
- **Arch Linux**

**Приклад роботи:**

Enter the number of sessions to play: 5
Choose computer mode:
1. Random moves
2. Current-session analysis
3. Historical analysis
Your choice (1-3): 2

--- Session 1 ---
Make your move:
1. Well
2. Scissors
3. Paper
Your choice (1-3): 1
You chose: WELL
Computer chose: SCISSORS
You win this session!
...

--- Game Results ---
Sessions played: 5
Your wins: 3
Computer wins: 1
Draws: 1

Do you want to see game statistics? (y/n): y
Total games played: 5
Total user wins: 3 (60.0%)
Total computer wins: 1 (20.0%)
Total draws: 1 (20.0%)

# 4. Висновки

Розроблено крос- платформний консольний застосунок на Java, що реалізує гру "Криниця- Ножиці- Папір" із трьома режимами інтелекту комп’ютера.
Програма:

- коректно обробляє повторні ходи при нічиї;
- зберігає повну історію сеансів у файл та аналізує її;
- відображає статистику відсотків перемог/поразок/нічиїх.

Тестування показало стабільну роботу на Windows та Arch Linux. Подальші покращення: розширити інтерфейс до GUI, додати графічне представлення статистики та зберігання даних у базі замість текстових файлів.