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

Дисципліна
« Кросплатформне програмування,»

Лабораторна робота № 7
на тему:
"Гра п’ять в ряд"

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 гри «п’ять в ряд», у якій двоє гравців по черзі ставлять хрестики (X) та нулі (O) у клітинки поля розміром 10×10. Перемагає той, хто першим виконає лінію з п’яти своїх символів у горизонталь, вертикаль або діагональ.

Додатково необхідно реалізувати алгоритм вибору ходу комп’ютером та обґрунтувати його оптимальність чи доцільність.

2. Опис реалізації та програмний код

- Параметри гри: поле 10×10, п’ять у ряд для перемоги, користувач ходить першим (X), комп’ютер ходить другим (O).
- Інтерфейс реалізовано на базі Swing: кнопки для кожної клітинки, статусна панель із повідомленням ходу та результату.
- Логіка перевірки перемоги: метод `checkWin` з підрахунком послідовних символів у чотирьох напрямках.

Розділ "Програмний код"

```
package lab7;



import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

import java.util.Random;
```

```
public class lab7 extends JFrame {
    private static final int BOARD_SIZE = 10;
    private static final int CELL_SIZE = 60;
    private static final int IN_A_ROW_TO_WIN = 5;

    private char[][] board = new char[BOARD_SIZE][BOARD_SIZE];
    private boolean playerTurn = true; // true: player, false: computer
    private boolean gameOver = false;
    private JButton[][] buttons = new JButton[BOARD_SIZE][BOARD_SIZE];
    private JLabel statusLabel;
    private final Random random = new Random();

    public lab7() {
        setTitle("Five in a Row");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(BOARD_SIZE * CELL_SIZE + 16, BOARD_SIZE * CELL_SIZE +
100);
        setLayout(new BorderLayout());

        initializeBoard();
        createBoardPanel();

        statusLabel = new JLabel("Your turn (X)", JLabel.CENTER);
        statusLabel.setFont(new Font("Arial", Font.BOLD, 16));
```

```
        add(statusLabel, BorderLayout.SOUTH);


        setLocationRelativeTo(null);
        setVisible(true);
    }


    private void initializeBoard() {
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                board[i][j] = ' ';
            }
        }
    }


    private void createBoardPanel() {
        JPanel boardPanel = new JPanel(new GridLayout(BOARD_SIZE,
BOARD_SIZE));


        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                buttons[i][j] = new JButton();
                buttons[i][j].setFont(new Font("Arial", Font.BOLD, 24));
                buttons[i][j].setFocusPainted(false);
                final int row = i;
                final int col = j;
```

```
                buttons[i][j].addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent e) {
                        if (playerTurn && !gameOver && board[row][col]
== ' ') {
                            makeMove(row, col, 'X');

                            if (!gameOver) {
                                playerTurn = false;
                                statusLabel.setText("Computer's turn
(O)");

                                // Add small delay for computer's move
                                Timer timer = new Timer(500, new
ActionListener() {
                                    @Override
                                    public void
actionPerformed(ActionEvent e) {
                                        computerMove();
                                        playerTurn = true;
                                        if (!gameOver) {
                                            statusLabel.setText("Your
turn (X)");
                                        }
                                    }
                                });
```

```
                                timer.setRepeats(false);
                                timer.start();
                            }
                        }
                    }
                });

                boardPanel.add(buttons[i][j]);
            }
        }

        add(boardPanel, BorderLayout.CENTER);
    }

    private void makeMove(int row, int col, char symbol) {
        board[row][col] = symbol;
        buttons[row][col].setText(String.valueOf(symbol));

        if (checkWin(row, col, symbol)) {
            gameOver = true;
            String winner = (symbol == 'X') ? "You win!" : "Computer
wins!";
            statusLabel.setText(winner);
            highlightWinningLine(row, col, symbol);
        } else if (isBoardFull()) {
```

```
            gameOver = true;
            statusLabel.setText("Game over! It's a draw!");
        }
    }

    private boolean isBoardFull() {
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                if (board[i][j] == ' ') {
                    return false;
                }
            }
        }
        return true;
    }

    private boolean checkWin(int row, int col, char symbol) {
        // Check horizontal
        if (countConsecutive(row, col, 0, 1, symbol) +
countConsecutive(row, col, 0, -1, symbol) - 1 >= IN_A_ROW_TO_WIN) {
            return true;
        }

        // Check vertical
        if (countConsecutive(row, col, 1, 0, symbol) +
countConsecutive(row, col, -1, 0, symbol) - 1 >= IN_A_ROW_TO_WIN) {
```

```
            return true;
        }

        // Check diagonal (top-left to bottom-right)
        if (countConsecutive(row, col, 1, 1, symbol) +
countConsecutive(row, col, -1, -1, symbol) - 1 >= IN_A_ROW_TO_WIN) {
            return true;
        }

        // Check diagonal (top-right to bottom-left)
        if (countConsecutive(row, col, 1, -1, symbol) +
countConsecutive(row, col, -1, 1, symbol) - 1 >= IN_A_ROW_TO_WIN) {
            return true;
        }

        return false;
    }

    private int countConsecutive(int row, int col, int dRow, int dCol,
char symbol) {
        int count = 0;

        while (row >= 0 && row < BOARD_SIZE && col >= 0 && col <
BOARD_SIZE && board[row][col] == symbol) {
            count++;
            row += dRow;
            col += dCol;
```

```
        }


        return count;
    }


    private void highlightWinningLine(int row, int col, char symbol) {
        // Check horizontal
        int horizontalCount = countConsecutive(row, col, 0, 1, symbol) +
countConsecutive(row, col, 0, -1, symbol) - 1;
        if (horizontalCount >= IN_A_ROW_TO_WIN) {
            highlightCells(row, col, 0, 1, symbol);
            highlightCells(row, col, 0, -1, symbol);
            return;
        }


        // Check vertical
        int verticalCount = countConsecutive(row, col, 1, 0, symbol) +
countConsecutive(row, col, -1, 0, symbol) - 1;
        if (verticalCount >= IN_A_ROW_TO_WIN) {
            highlightCells(row, col, 1, 0, symbol);
            highlightCells(row, col, -1, 0, symbol);
            return;
        }


        // Check diagonal (top-left to bottom-right)
```

```
        int diagonalCount1 = countConsecutive(row, col, 1, 1, symbol) +
countConsecutive(row, col, -1, -1, symbol) - 1;

        if (diagonalCount1 >= IN_A_ROW_TO_WIN) {

            highlightCells(row, col, 1, 1, symbol);

            highlightCells(row, col, -1, -1, symbol);

            return;

        }



        // Check diagonal (top-right to bottom-left)

        int diagonalCount2 = countConsecutive(row, col, 1, -1, symbol) +
countConsecutive(row, col, -1, 1, symbol) - 1;

        if (diagonalCount2 >= IN_A_ROW_TO_WIN) {

            highlightCells(row, col, 1, -1, symbol);

            highlightCells(row, col, -1, 1, symbol);

        }

    }



    private void highlightCells(int startRow, int startCol, int dRow,
int dCol, char symbol) {

        int row = startRow;

        int col = startCol;



        while (row >= 0 && row < BOARD_SIZE && col >= 0 && col <
BOARD_SIZE && board[row][col] == symbol) {

            buttons[row][col].setBackground(Color.GREEN);

            row += dRow;

            col += dCol;
```

```
        }

        // Also highlight the starting cell
        buttons[startRow][startCol].setBackground(Color.GREEN);
    }

    private void computerMove() {
        if (gameOver) return;

        // 1. Try to win - look for 4 in a row
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                if (board[i][j] == ' ') {
                    board[i][j] = 'O';  // Try placing O
                    if (checkWin(i, j, 'O')) {
                        board[i][j] = ' ';  // Reset for further check
                        makeMove(i, j, 'O');
                        return;
                    }
                    board[i][j] = ' ';  // Reset
                }
            }
        }

        // 2. Block player's winning move
```

```
        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                if (board[i][j] == ' ') {
                    board[i][j] = 'X';  // Try placing X (simulate
player's move)
                    if (checkWin(i, j, 'X')) {
                        board[i][j] = ' ';  // Reset for further check
                        makeMove(i, j, 'O');  // Block with O
                        return;
                    }
                    board[i][j] = ' ';  // Reset
                }
            }
        }

        // 3. Look for potential threats and opportunities (3 in a row)
        Point bestMove = findBestMove();
        if (bestMove != null) {
            makeMove(bestMove.x, bestMove.y, 'O');
            return;
        }

        // 4. If first move, try to place near center
        if (isFirstMove('O')) {
            int centerRegionStart = BOARD_SIZE / 2 - 1;
```

```
            int centerRegionEnd = BOARD_SIZE / 2 + 1;


            for (int attempts = 0; attempts < 10; attempts++) {
                int i = random.nextInt(centerRegionEnd -
centerRegionStart + 1) + centerRegionStart;
                int j = random.nextInt(centerRegionEnd -
centerRegionStart + 1) + centerRegionStart;


                if (board[i][j] == ' ') {
                    makeMove(i, j, 'O');
                    return;
                }
            }
        }


        // 5. Random move as last resort
        while (true) {
            int i = random.nextInt(BOARD_SIZE);
            int j = random.nextInt(BOARD_SIZE);


            if (board[i][j] == ' ') {
                makeMove(i, j, 'O');
                return;
            }
        }
    }
```

```
private boolean isFirstMove(char symbol) {
    int count = 0;
    for (int i = 0; i < BOARD_SIZE; i++) {
        for (int j = 0; j < BOARD_SIZE; j++) {
            if (board[i][j] == symbol) {
                count++;
            }
        }
    }
    return count == 0;
}

private Point findBestMove() {
    int[][] scoreBoard = new int[BOARD_SIZE][BOARD_SIZE];

    // Evaluate each empty cell
    for (int i = 0; i < BOARD_SIZE; i++) {
        for (int j = 0; j < BOARD_SIZE; j++) {
            if (board[i][j] == ' ') {
                // Evaluate for both computer (O) and player (X)
                int computerScore = evaluatePosition(i, j, 'O');
                int playerScore = evaluatePosition(i, j, 'X');
```

```
                    // Prioritize computer's offense but also consider
defense
                    scoreBoard[i][j] = computerScore * 2 + playerScore;
                }
            }
        }

        // Find cell with highest score
        int maxScore = -1;
        Point bestMove = null;

        for (int i = 0; i < BOARD_SIZE; i++) {
            for (int j = 0; j < BOARD_SIZE; j++) {
                if (board[i][j] == ' ' && scoreBoard[i][j] > maxScore) {
                    maxScore = scoreBoard[i][j];
                    bestMove = new Point(i, j);
                }
            }
        }

        return bestMove;
    }

    private int evaluatePosition(int row, int col, char symbol) {
        int score = 0;
```

```
        // Try placing symbol temporarily
        board[row][col] = symbol;

        // Check in all 8 directions
        int[][] directions = {
            {0, 1}, {1, 0}, {1, 1}, {1, -1},  // Right, Down, Down-Right, Down-Left
            {0, -1}, {-1, 0}, {-1, -1}, {-1, 1}  // Left, Up, Up-Left, Up-Right
        };

        for (int d = 0; d < 8; d += 2) {  // We check directions in pairs (opposite directions)
            int[] dir1 = directions[d];
            int[] dir2 = directions[d + 1];

            int count = 1;  // Count the cell itself
            int openEnds = 0;

            // Count consecutive symbols in first direction
            int r = row + dir1[0];
            int c = col + dir1[1];
            while (r >= 0 && r < BOARD_SIZE && c >= 0 && c < BOARD_SIZE && board[r][c] == symbol) {
                count++;
                r += dir1[0];
```

```
                c += dir1[1];
            }

            // Check if first end is open
            if (r >= 0 && r < BOARD_SIZE && c >= 0 && c < BOARD_SIZE &&
board[r][c] == ' ') {
                openEnds++;
            }

            // Count consecutive symbols in second direction
            r = row + dir2[0];
            c = col + dir2[1];
            while (r >= 0 && r < BOARD_SIZE && c >= 0 && c < BOARD_SIZE
&& board[r][c] == symbol) {
                count++;
                r += dir2[0];
                c += dir2[1];
            }

            // Check if second end is open
            if (r >= 0 && r < BOARD_SIZE && c >= 0 && c < BOARD_SIZE &&
board[r][c] == ' ') {
                openEnds++;
            }

            // Score based on consecutive symbols and open ends
```

```
            if (count >= 5) {
                score += 10000;  // Winning move
            } else if (count == 4 && openEnds > 0) {
                score += (openEnds == 2) ? 2000 : 500;  // Four with
open end(s)
            } else if (count == 3 && openEnds > 0) {
                score += (openEnds == 2) ? 200 : 50;  // Three with open
end(s)
            } else if (count == 2 && openEnds > 0) {
                score += (openEnds == 2) ? 10 : 5;  // Two with open
end(s)
            }
        }

        // Reset the cell
        board[row][col] = ' ';

        return score;
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new lab7();
            }
```

```
        });

    }

}
```

3. Опис алгоритму ходів комп’ютера

Алгоритм комп’ютера складається з кількох етапів:

1. **Переможний хід**: перебір всіх порожніх клітинок, спроба поставити O, перевірка на перемогу за допомогою `checkWin`. Якщо можливо виграти негайно — зробити хід.
2. **Блокування суперника**: аналогічний перебір із установкою X для перевірки, чи може гравець виграти наступним ходом; якщо так, поставити O у цю клітинку.
3. **Оцінка позицій**: для кожної порожньої клітинки обчислюється потужність ходу як `score = 2*computerScore + playerScore`, де `evaluatePosition` аналізує довжини послідовностей та відкриті кінці у всіх напрямках.
4. **Центральна стратегія**: якщо це перший хід комп’ютера, вибір випадкової клітинки в центральній зоні.
5. **Випадковий хід**: якщо попередні стратегії не визначили хід.

Доцільність алгоритму: балансує між атакою (пошук переможного ходу), захистом (блокування), стратегічною оцінкою та позиціонуванням, що робить гру проти комп’ютера складною, але відносно простою у реалізації.

4. Результати тестування

Програма протестована на:

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

Перевірено сценарії:

- перемоги користувача та комп’ютера;
- нічийні ігри на заповненому полі;
- ефективність блокування та пошуку виграшного ходу комп’ютером;
- коректність підсвічування лінії перемоги.

5. Висновки

Розроблено крос- платформний GUI- застосунок гри «п’ять в ряд» з реалізацією інтелектуального алгоритму для комп’ютера.

Програма забезпечує:

- коректне виявлення перемоги за будь- яких напрямів;
- стратегічні і оборонні ходи комп’ютера;
- стабільну роботу на Windows та Arch Linux.

Перспективи: розширити поле, налаштувати рівні складності алгоритму, зберігати та аналізувати статистику ігор.