**Московский государственный технический**
**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»
Отчет по Домашнему заданию
Игра «Крестики-нолики»

Выполнил:
студент группы ИУ5-35Б
Рябов М.А.

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Москва, 2025 г.

## Постановка задачи

Разработать игру «Крестики-нолики» на языке программирования Java. Представить графический интерфейс игры в виде текста в терминале. Реализовать функцию игры на двоих и игры с компьютером.

## Код программы

## Main.java

```
package tictactoe;

public class Main {
    public static void main(String args[]) {
        Game game = new Game();
        game.start();
    }
}
```

## Game.java

```
package tictactoe;

import java.util.Scanner;

public class Game {

    private final Board board;
    private final Player[] players;

    private final int maxPlayerCount = 2;

    private int currentPlayerIndex;

    public Game() {
        this.board = new Board();
        this.players = new Player[maxPlayerCount];
        this.currentPlayerIndex = 0;

        initializePlayers();
    }

    private void initializePlayers() {
        Scanner scanner = new Scanner(System.in);

        int playerCount;
        char[] playerSymbols = {'X', 'O'};

        while (true) {
            try {
                System.out.println("Введите количество игроков (1 или 2): ");
```

```
                playerCount = Integer.parseInt(scanner.nextLine());

                if (playerCount == 1 || playerCount == 2) {
                    break;
                } else {
                    System.out.println("Пожайлуйста, введите целое число от 1 до
2 включительно");
                }
            } catch (NumberFormatException e) {
                System.out.println("Пожайлуйста, введите целое число.");
            }
        }

        for (int i = 0; i < playerCount; i++) {
            System.out.println("Игрок " + (i + 1) + ", введите свое имя: ");
            String playerName = scanner.nextLine();

            if (playerName.isEmpty()) {
                playerName = "Игрок " + (i + 1);
            }

            players[i] = new Player(playerName, playerSymbols[i]);
        }

        if (playerCount == 1) {
            players[maxPlayerCount - 1] = new CPU("CPU",
playerSymbols[maxPlayerCount - 1]);
        }
    }

    public void start() {
        System.out.println("\n=== НАЧАЛО ИГРЫ ===");
        board.print();

        while (true) {
            Player currentPlayer = players[currentPlayerIndex];

            // Ход текущего игрока
            if (!currentPlayer.makeMove(board)) {
                continue;
            }

            // Печатаем обновленное поле
            board.print();

            // Проверяем условия окончания игры
            if (board.checkWinner() != ' ') {
                System.out.println("Поздравляем! " + currentPlayer.getName() + "
победил!");
                break;
            }
```

```
            if (board.isFull()) {
                System.out.println("Ничья! Поле полностью заполнено.");
                break;
            }

            // Передаем ход следующему игроку
            currentPlayerIndex = (currentPlayerIndex + 1) % 2;
        }
    }
}
```

## Board.java

```
package tictactoe;

import java.util.ArrayList;
import java.util.List;

public class Board {

    private final char[][] grid;
    private static final int SIZE = 3;

    public Board() {
        grid = new char[SIZE][SIZE];
        initializeBoard();
    }

    public Board(Board other) {
        this.grid = new char[SIZE][SIZE];
        for (int i = 0; i < SIZE; i++) {
            System.arraycopy(other.grid[i], 0, this.grid[i], 0, SIZE);
        }
    }

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

    public void print() {
        System.out.println("\n  1   2   3");

        for (int i = 0; i < SIZE; i++) {
            System.out.print((i + 1) + " ");
```

```
            for (int j = 0; j < SIZE; j++) {
                System.out.print(grid[i][j]);
                if (j < SIZE - 1) {
                    System.out.print(" | ");
                }
            }

            System.out.println();

            if (i < SIZE - 1) {
                System.out.println("  ---------");
            }
        }
    }

    public List<int[]> getAvailableMoves() {
        List<int[]> moves = new ArrayList<>();
            for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (grid[i][j] == ' ') {
                    moves.add(new int[]{i, j});
                }
            }
        }
        return moves;
    }

    public boolean makeMove(int row, int col, char symbol) {
        if (row < 0 || row >= SIZE || col < 0 || col >= SIZE) {
            return false;
        }

        if (grid[row][col] != ' ') {
            return false;
        }

        grid[row][col] = symbol;
        return true;
    }

    public char checkWinner() {
        // Проверка строк
        for (int i = 0; i < SIZE; i++) {
            if (grid[i][0] != ' ' && grid[i][0] == grid[i][1] && grid[i][1] ==
grid[i][2]) {
                return grid[i][0];
            }
        }

        // Проверка столбцов
        for (int j = 0; j < SIZE; j++) {
```

```
            if (grid[0][j] != ' ' && grid[0][j] == grid[1][j] && grid[1][j] ==
grid[2][j]) {
                return grid[0][j];
            }
        }

        // Проверка диагоналей
        if (grid[0][0] != ' ' && grid[0][0] == grid[1][1] && grid[1][1] ==
grid[2][2]) {
            return grid[0][0];
        }

        if (grid[0][2] != ' ' && grid[0][2] == grid[1][1] && grid[1][1] ==
grid[2][0]) {
            return grid[0][2];
        }

        return ' ';
    }

    public boolean isFull() {
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                if (grid[i][j] == ' ') {
                    return false;
                }
            }
        }

        return true;
    }
}
```

## Player.java

```
package tictactoe;

import java.util.Scanner;

public class Player {
    private final String name;
    private final char symbol;

    public Player(String name, char symbol) {
        this.name = name;
        this.symbol = symbol;
    }

    public String getName() {
        return name;
```

```
    }

    public char getSymbol() {
        return symbol;
    }

    public boolean makeMove(Board board) {
        try {
            Scanner scanner = new Scanner(System.in);

            System.out.println("\n" + this.name + " (" + this.symbol + "), ваш
ход.");
            System.out.print("Введите номер строки (1-3): ");

            int row = Integer.parseInt(scanner.nextLine()) - 1;

            System.out.print("Введите номер столбца (1-3): ");

            int col = Integer.parseInt(scanner.nextLine()) - 1;

            if (board.makeMove(row, col, this.symbol)) {
                return true;
            } else {
                System.out.println("Эта ячейка уже занята или координаты неверны.
Попробуйте снова.");
            }
        } catch (NumberFormatException e) {
            System.out.println("Пожалуйста, введите числа от 1 до 3.");
        }

        return false;
    }
}
```

## CPU.java

```
package tictactoe;

import java.util.List;
import java.util.Random;

public class CPU extends Player {
    // Вероятность того, что компьютер ошибется (в процентах)
    private final int maxErrorProbability = 25;

    public CPU(String name, char symbol) {
        super(name, symbol);
    }

    private int minimax(Board board, int depth, boolean isMaximizing) {
```

```
        if (board.checkWinner() == 'O') {
            return 10 - depth;
        }
        if (board.checkWinner() == 'X') {
            return depth - 10;
        }
        if (board.getAvailableMoves().isEmpty()) {
            return 0; // Ничья
        }

        if (isMaximizing) {
            // Максимизируем счет игрока (O)
            int bestScore = Integer.MIN_VALUE;
            for (int[] move : board.getAvailableMoves()) {
                Board newBoard = new Board(board);
                newBoard.makeMove(move[0], move[1], 'O');
                int score = minimax(newBoard, depth + 1, false);
                bestScore = Math.max(bestScore, score);
            }
            return bestScore;
        } else {
            // Минимизируем счет игрока (X)
            int bestScore = Integer.MAX_VALUE;
            for (int[] move : board.getAvailableMoves()) {
                Board newBoard = new Board(board);
                newBoard.makeMove(move[0], move[1], 'X');
                int score = minimax(newBoard, depth + 1, true);
                bestScore = Math.min(bestScore, score);
            }
            return bestScore;
        }
    }

    private int[] findBestMove(Board board) {
        int bestScore = Integer.MIN_VALUE;
        int[] bestMove = new int[]{-1, -1};

        // Пробуем все ходы
        for (int[] move : board.getAvailableMoves()) {
            Board newBoard = new Board(board);
            newBoard.makeMove(move[0], move[1], super.getSymbol());

            // Считает итоговый счет для каждого хода
            int score = minimax(newBoard, 0, false);

            // Ищем максимальный счет
            if (score > bestScore) {
                bestScore = score;
                bestMove = move;
            }
        }
```

```
        return bestMove;
    }

    public int[] findRandomMove(Board board) {
        Random random = new Random();

        List<int[]> availableMoves = board.getAvailableMoves();

        int randomMoveIndex = random.nextInt(availableMoves.size());

        return availableMoves.get(randomMoveIndex);
    }

    @Override
    public boolean makeMove(Board board) {
        Random random = new Random();
        int errorProbability = random.nextInt(100);

        int[] move = (errorProbability < maxErrorProbability)
            ? findRandomMove(board)
            : findBestMove(board);

        board.makeMove(move[0], move[1], super.getSymbol());
        return true;
    }
}
```

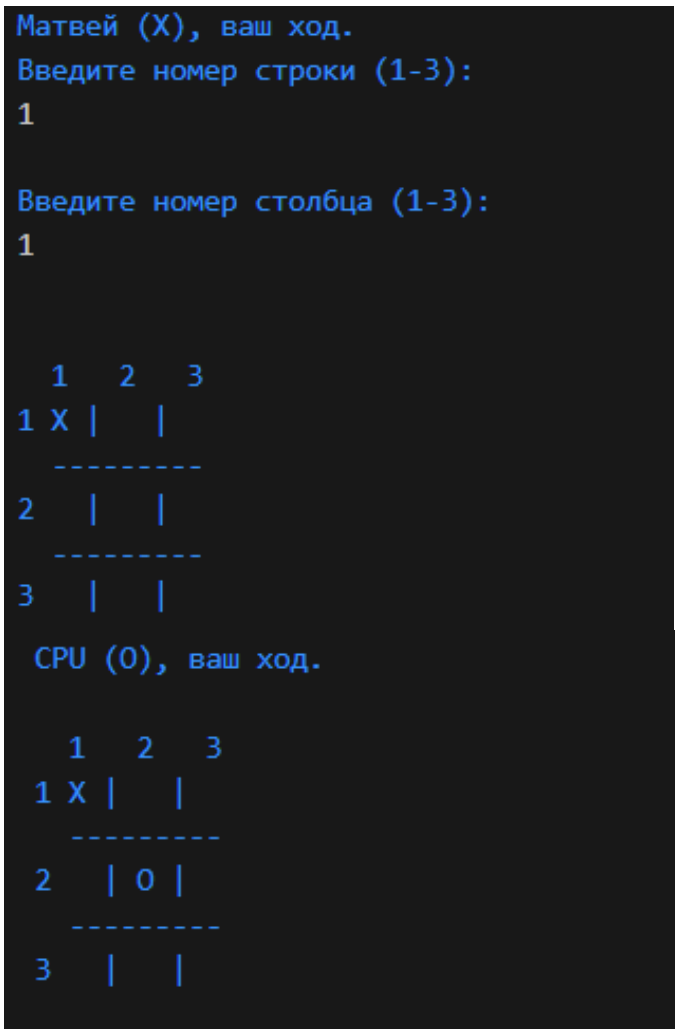
## Выполнение программы

```
Введите количество игроков (1 или 2):
1

Игрок 1, введите свое имя:
Матвей


=== НАЧАЛО ИГРЫ ===

  1   2   3
1   |   |
  ---------
2   |   |
  ---------
3   |   |
```


Матвей (X), ваш ход.
Введите номер строки (1-3):
1
Введите номер столбца (1-3):
1
1 2 3
1 X | |
---
2 | |
---
3 | |
CPU (O), ваш ход.
1 2 3
1 X | |
---
2 | O |
---
3 | |