Task 2

TIC-TAC-TOE AI

Code:

```
import java.util.Scanner;

public class TicTacToeAI {

    // Constants for the game
    private static final char EMPTY = ' ';
    private static final char PLAYER_X = 'X';
    private static final char PLAYER_O = 'O';
    private static final int SIZE = 3;

    private static char[][] board = new char[SIZE][SIZE];

    public static void main(String[] args) {
        initializeBoard();
        Scanner scanner = new Scanner(System.in);
        boolean gameEnded = false;

        while (!gameEnded) {
            printBoard();
            playerMove(scanner);
            if (checkWin(PLAYER_X)) {
                printBoard();
                System.out.println("Player X wins!");
            gameEnded = true;
        } else if (isBoardFull()) {
            printBoard();
            System.out.println("The game is a draw.");
            gameEnded = true;
        } else {
            aiMove();
            if (checkWin(PLAYER_O)) {
                printBoard();
                System.out.println("Player O wins!");
                gameEnded = true;
            } else if (isBoardFull()) {
                printBoard();
                System.out.println("The game is a draw.");
                gameEnded = true;
            }
        }
    }
    scanner.close();
}

// Initialize the game board
private static void initializeBoard() {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            board[i][j] = EMPTY;
        }
        }
    }

    // Print the game board
    private static void printBoard() {
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                System.out.print(board[i][j]);
                if (j < SIZE - 1) System.out.print("|");
            }
            System.out.println();
            if (i < SIZE - 1) System.out.println("-----");
        }
    }

    // Handle player's move
    private static void playerMove(Scanner scanner) {
        int row, col;
        while (true) {
            System.out.print("Enter your move (row and column): ");
            row = scanner.nextInt() - 1;
            col = scanner.nextInt() - 1;
            if (row >= 0 && row < SIZE && col >= 0 && col < SIZE &&
board[row][col] == EMPTY) {
                board[row][col] = PLAYER_X;
                break;
            } else {
                System.out.println("This move is not valid. Try again.");
            }
        }
    }

    // Check if the board is full
    private static boolean isBoardFull() {
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                if (board[i][j] == EMPTY) {
                    return false;
                }
            }
        }
        return true;
    }

    // Check if a player has won
    private static boolean checkWin(char player) {
        // Check rows, columns, and diagonals
        for (int i = 0; i < SIZE; i++) {
            if ((board[i][0] == player && board[i][1] == player && board[i][2] ==
player) ||
                (board[0][i] == player && board[1][i] == player && board[2][i] ==
player)) {
                return true;
            }
        }
        if ((board[0][0] == player && board[1][1] == player && board[2][2] ==
player) ||
            (board[0][2] == player && board[1][1] == player && board[2][0] ==
player)) {
            return true;
        }
        return false;
    }

    // AI's move using Minimax algorithm
    private static void aiMove() {
        int[] bestMove = minimax(board, PLAYER_O);
        board[bestMove[0]][bestMove[1]] = PLAYER_O;
    }

    // Minimax algorithm
    private static int[] minimax(char[][] board, char player) {
        char opponent = (player == PLAYER_O) ? PLAYER_X : PLAYER_O;
        int bestScore = (player == PLAYER_O) ? Integer.MIN_VALUE :
Integer.MAX_VALUE;
        int[] bestMove = new int[]{-1, -1};

        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                if (board[i][j] == EMPTY) {
                    board[i][j] = player;
                    int score = minimaxScore(board, player);
                    board[i][j] = EMPTY;
                if (player == PLAYER_O) {
                    if (score > bestScore) {
                        bestScore = score;
                        bestMove[0] = i;
                        bestMove[1] = j;
                    }
                } else {
                    if (score < bestScore) {
                        bestScore = score;
                        bestMove[0] = i;
                        bestMove[1] = j;
                    }
                }
            }
        }
    }
    return bestMove;
}

private static int minimaxScore(char[][] board, char player) {
    if (checkWin(PLAYER_O)) return 10;
    if (checkWin(PLAYER_X)) return -10;
    if (isBoardFull()) return 0;


    char opponent = (player == PLAYER_O) ? PLAYER_X : PLAYER_O;
    int bestScore = (player == PLAYER_O) ? Integer.MIN_VALUE :
Integer.MAX_VALUE;
```

```
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                if (board[i][j] == EMPTY) {
                    board[i][j] = player;
                    int score = minimaxScore(board, opponent);
                    board[i][j] = EMPTY;

                    if (player == PLAYER_O) {
                        bestScore = Math.max(score, bestScore);
                    } else {
                        bestScore = Math.min(score, bestScore);
                    }
                }
            }
        }
        return bestScore;
    }
}
```