MAPA – Material de Avaliação Prática da Aprendizagem

Acadêmico: Robson Cruz Santos	**R.A.: 22117001-5**
Curso: Engenharia de Software	
Disciplina: Programação de Sistemas I	
Valor da atividade: 3,5	**Prazo: 24/11/2024**

CRIAÇÃO DO JOGO DA VELHA EM JAVA

O jogo Tic-tac-toe, conhecido como "jogo da velha" no Brasil, é um jogo clássico para dois jogadores. O tabuleiro é uma grade de 3x3 e os jogadores alternam turnos para marcar os espaços com seus símbolos, convencionalmente representados pelos caracteres "X" e "O". O objetivo de cada participante é ser o primeiro a alinhar três símbolos consecutivos na horizontal, vertical ou diagonal. Se todos os nove espaços são preenchidos sem que ninguém alinhe três, o jogo termina em empate. Simples, mas sempre divertido e estratégico

OBJETIVO

Desenvolver uma aplicação Java que implemente o jogo da velha (Tic-Tac-Toe) utilizando apenas a linha de comando. O objetivo é reforçar o entendimento sobre estruturas de dados, controle de fluxo, e manipulação de matrizes, além de praticar a interação com o usuário via console.

REQUISITOS FUNCIONAIS

1. O jogo deve permitir dois jogadores humanos, que se revezam inserindo os símbolos 'X' e 'O' em um tabuleiro 3x3.
2. Deve haver verificação automática de vitória, empate ou jogadas inválidas.
3. O jogo deve exibir o tabuleiro atualizado após cada jogada e solicitar a próxima entrada do jogador.

REQUISITOS TÉCNICOS

1. Utilizar uma matriz 3x3 para representar o tabuleiro do jogo.
2. Implementar toda a lógica do jogo na linha de comando, incluindo a exibição do tabuleiro e a coleta de entradas dos jogadores.
3. Incluir tratamento de exceções para evitar erros em tempo de execução, como inserção em células já ocupadas ou entradas inválidas.

4. Ao final do jogo, exibir uma mensagem indicando o vencedor ou declarando empate, e oferecer a opção de iniciar uma nova partida.

METODOLOGIA

O jogo foi desenvolvido com a linguagem de programação Java, versão JDK 21 (ORACLE. 2024); utilizou-se o ambiente de desenvolvimento integrado (Integrated Development Environment – IDE) Visual Studio Code, versão 1.84.2 (MICROSOFT, 2024a); e sistema operacional Windows 10 Pro de 64 bits, versão 22H2 (MICROSOFT, 2024b).

RESULTADOS

O código possui a classe denominada *TicTacToe*, a qual possui os seguintes métodos:

1. Método *main*: O método principal, utilizado para inicialização do jogo quando compilado. Este método chama os métodos *startGame* e *restartGame*, conforme o comportamento do jogo e da decisão dos participantes, além de mostrar a logomarca do jogo (ascii.co.uk, (2024), assim como instruções aos jogadores, conforme demonstrado através da *Figura 1*.

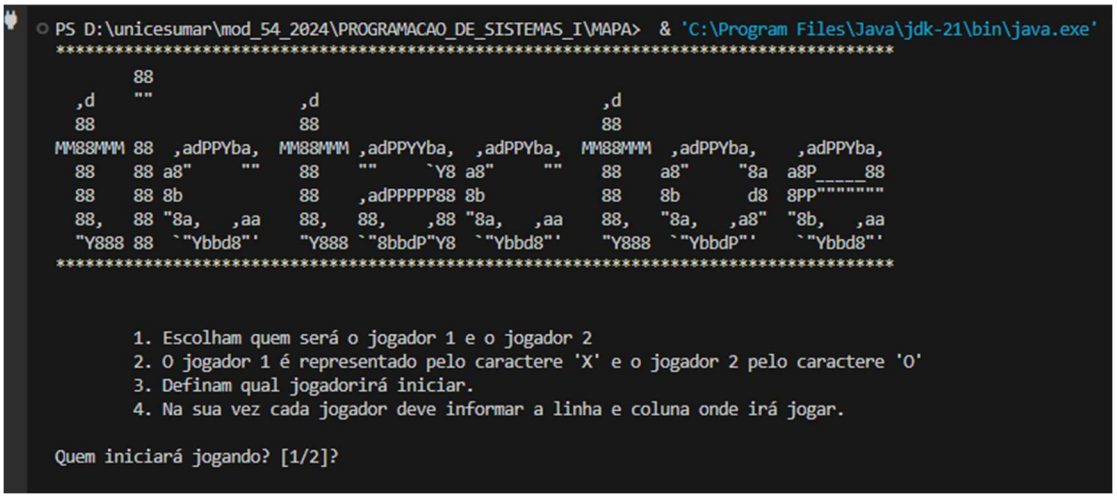

Figura 1. Saída do método main no terminal do Visual Studio Code.

Fonte: o autor.

2. Método *startGame:* Este método é utilizado para iniciar o jogo, recebendo como parâmetro *Scanner scanner*. Dentro deste método são chamados os seguintes métodos:
3. Método *mostrarMatrizPadrao*: utilizado para mostrar uma matriz no formato 3x3 preenchida com o caractere "-", a matriz é utilizada para representação gráfica do tabuleiro do jogo Tic-Tac-Toe. Dentro de *mostrarMatrizPadrao*, é chamado o método mostrarMatriz;
4. Método *inserirJogada:* utilizado para inserir as coordenadas dos jogadores na matriz 3x3, onde em cada turno, um jogador escolhe uma linha e coluna e, se a célula estiver vazia, o símbolo escolhido ('x' ou 'o') é colocado nessa célula.;
5. Método *checkWin*: usado para conferir se a cada jogada há um vencedor através da avaliação de linhas, colunas e diagonais da matriz 3x3;
6. Método *restartGame:* Após uma vitória ou empate, se os jogadores escolhem jogar novamente, o jogo reinicia sem fechar o programa;
7. Método *checkArrays:* método auxiliar do método *checkWin,*o que permite verificar se em uma linha, coluna ou as diagonais do tabuleiro possuem somente um determinado caractere.

A *Figura 2*, mostra a execução dos métodos *startGame* e *mostrarMatrizPadrao*.

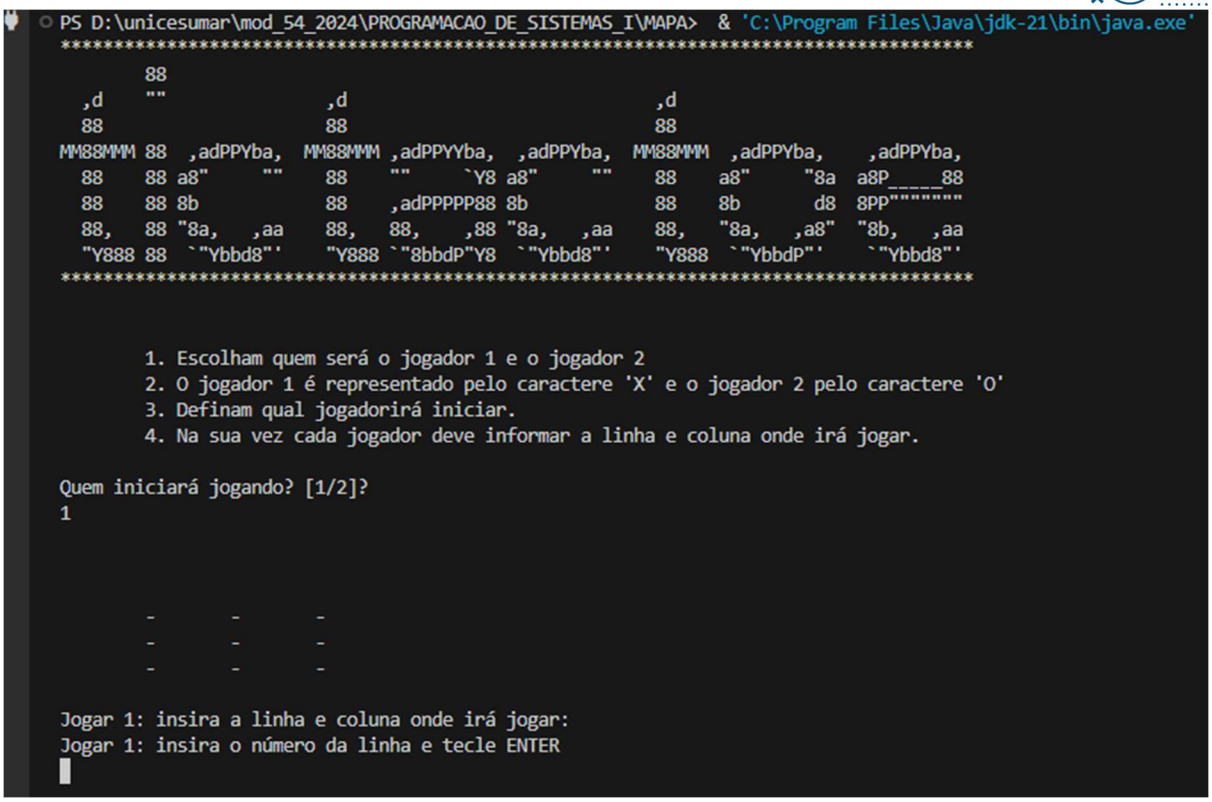

Figura 2. Execução do método startGame.

Fonte: o autor.

A *Figura 3*, demonstra atendimento aos requisitos funcionais do jogo, relacionados ao revezamento entre os participantes do jogo e ao preenchimento dos símbolos 'X' e 'O' no tabuleiro 3x3.

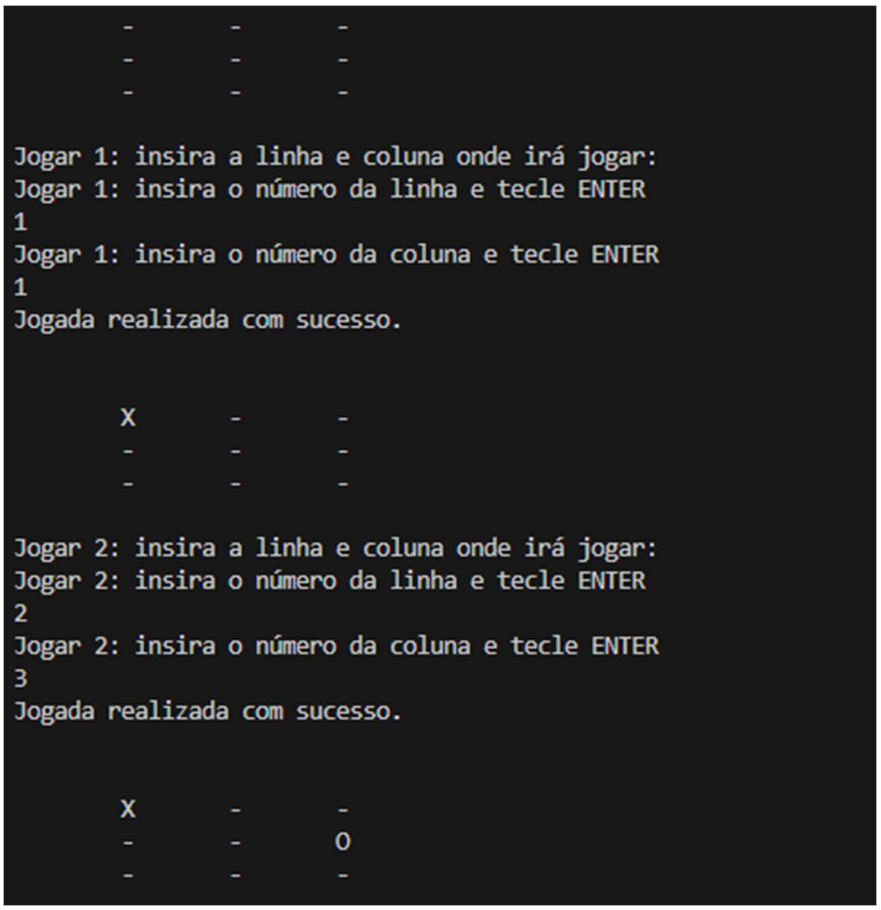

Figura 3. Revezamento entre jogadores durante uma partida.
Fonte: O autor.

Através da *Figura 4*, é demonstrado atendimento ao requisito funcional de verificação automática de vitória.

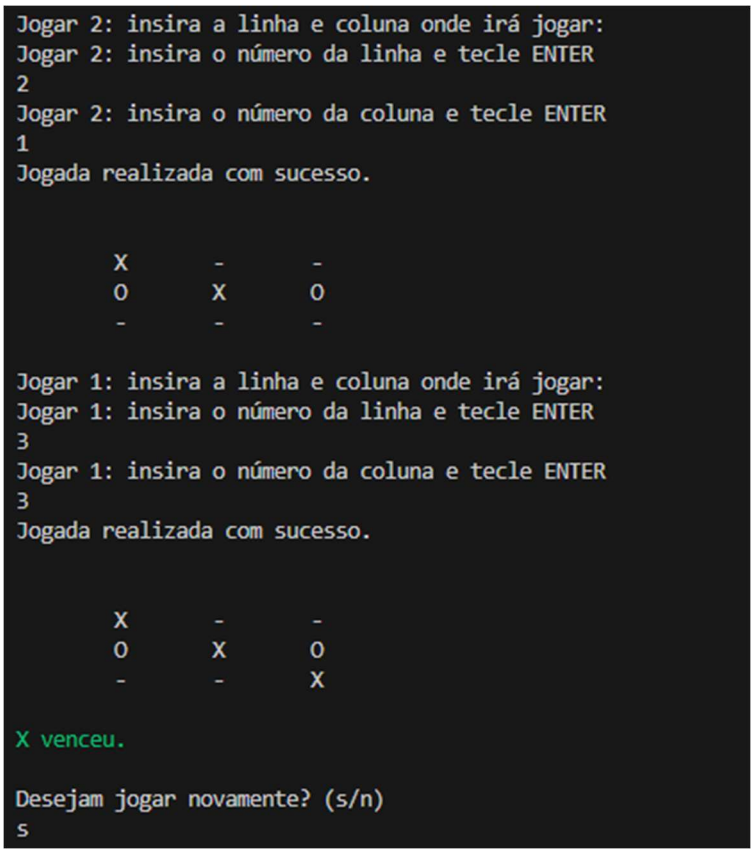

Figura 4. Verificação automática de vitória durante uma partida.
Fonte: O autor.

O código permite atendimento ao requisito funcional de verificação automática de jogada inválida, quando um jogador tenta inserir coordenadas inválidas no tabuleiro, por exemplo coordenadas com algum índice fora dos limites de linhas ou colunas da matriz 3x3, bem como tentar inserir um caractere em posição já preenchida em jogadas anteriores.

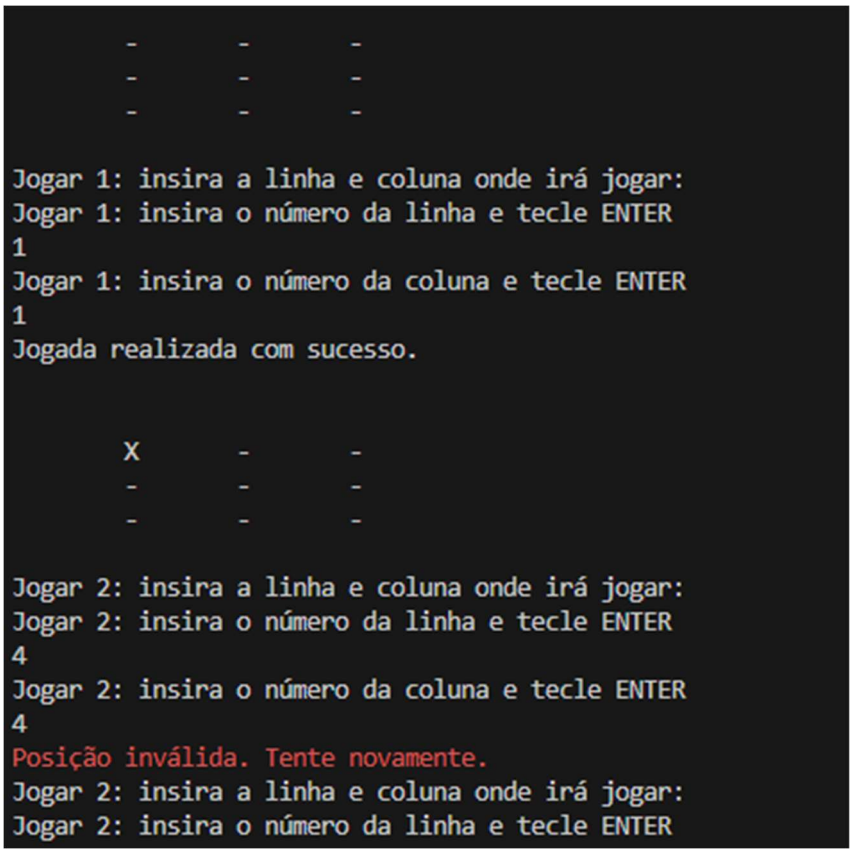

Figura 5. Verificação automática de tentativa de jogada inválida durante uma partida.

Fonte: O autor.

Após o jogo terminar, no caso de vitória ou empate, o jogo pergunta se os jogadores desejam reiniciar a partida, o que tratado pelo método *restartGame*. Se os jogadores escolhem jogar novamente, o jogo reinicia sem fechar o programa.

Por fim, através do *Quadro 1*, apresenta-se o código completo do programa escrito em linguagem de programação Java.

Quadro 1. Código em linguagem Java utilizado para desenvolver o jogo Tic-Tac-Toe.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;
```

```
/**
 * MAPA - ESOFT - PROGRAMAÇÃO DE SISTEMAS I - 54_2024
 * CRIAÇÃO DO JOGO DA VELHA EM JAVA
 */
public class TicTacToe {
    // Definição de constantes para manipular cores na saída do terminal
    private static final String ANSI_RED = "\u001B[31m";
    private static final String ANSI_RESET = "\u001B[0m";
    private static final String ANSI_GREEN = "\033[32m";
    private static final String ANSI_YELLOW = "\033[93m";

    // Variável global para controlar o estado do jogo
    private static boolean stopGame = false;

    /**
     * @param args
     */
    public static void main(String[] args) {
        // Leitura de arquivo como o nome do jogo
        String pathFile = "./ticTacToe.txt";
        try(BufferedReader reader = new BufferedReader(new FileReader(pathFile)))
{
            String fileLine;
            // Lê o arquivo linha por linha
            while ((fileLine = reader.readLine()) != null) {
                // Imprime cada linha no console
                System.out.println(fileLine);
            }
        } catch(IOException e) {
            e.printStackTrace();
        }
```

```
    Scanner scanner = new Scanner(System.in);
    boolean playAgain = true;

    while (playAgain) {
        startGame(scanner);

        // Após o jogo terminar, perguntar se querem jogar novamente
        System.out.println("Desejam jogar novamente? (s/n)");
        char resposta = scanner.next().charAt(0);
        if (resposta == 'n' || resposta == 'N') {
            playAgain = false;
            System.out.println("Obrigado por jogar!\n");
        } else {
            restartGame();
        }
    }
}

// Método para iniciar o jogo
/**
 * @param scanner
 */
public static void startGame(Scanner scanner) {
    // Definição de variáveis
    int currentPlayer;
    int x, y;
    int counter = 0;
    char[][] matrizJogo = new char[3][3];

    System.out.println("\n\n\t1. Escolham quem será o jogador 1 e o jogador 2");
```

```
        System.out.println("\t2. O jogador 1 é representado pelo caractere 'X' e o jogador 2 pelo caractere 'O'");
        System.out.println("\t3. Definam qual jogadorirá iniciar.");
        System.out.println("\t4. Na sua vez cada jogador deve informar a linha e coluna onde irá jogar.\n");
        System.out.println("Quem iniciará jogando? [1/2]?");

        currentPlayer = scanner.nextInt();

        mostrarMatrizPadrao(matrizJogo);

        while (counter < 9 && !stopGame) {
            if (currentPlayer == 1) {
                System.out.println("Jogar 1: insira a linha e coluna onde irá jogar:");
                System.out.println("Jogar 1: insira o número da linha e tecle ENTER");
                x = scanner.nextInt() - 1;
                System.out.println("Jogar 1: insira o número da coluna e tecle ENTER");
                y = scanner.nextInt() - 1;
                if (x >= 0 && x < 3 && y >= 0 && y < 3 && matrizJogo[x][y] == '-') {
                    inserirJogada(matrizJogo, 'X', x, y);
                    mostrarMatriz(matrizJogo);
                    counter ++;
                    checkWin(matrizJogo);
                    currentPlayer = 2; // Alterna para o jogador 2
                } else {
                    System.out.println(ANSI_RED + "Posição inválida. Tente novamente." + ANSI_RESET);
                }

            } else {
                System.out.println("Jogar 2: insira a linha e coluna onde irá jogar:");
                System.out.println("Jogar 2: insira o número da linha e tecle ENTER");
```

```
            x = scanner.nextInt() - 1;
            System.out.println("Jogar 2: insira o número da coluna e tecle ENTER");
            y = scanner.nextInt() - 1;
            if (x >= 0 && x < 3 && y >= 0 && y < 3 && matrizJogo[x][y] == '-') {
                inserirJogada(matrizJogo, 'O', x, y);
                mostrarMatriz(matrizJogo);
                counter ++;
                checkWin(matrizJogo);
                currentPlayer = 1; // Alterna para o jogador 1
            } else {
                System.out.println(ANSI_RED + "Posição inválida. Tente novamente."
+ ANSI_RESET);
            }
        }
    }

    if (!stopGame) {
        System.out.println(ANSI_YELLOW + "O jogo empatou!" + ANSI_RESET);
    }
}

// Método para reiniciar o jogo
public static void restartGame() {
    stopGame = false; // Reseta o status do jogo
}

// Método para mostrar a matriz do jogo vazia
/**
 * @param matriz
 */
public static void mostrarMatrizPadrao(char matriz[][]) {
    for (int linha = 0; linha < matriz.length; linha++) {
```

```
            for (int coluna = 0; coluna < matriz[linha].length; coluna++) {
                matriz[linha][coluna] = '-';
            }
        }
        System.out.println();

        mostrarMatriz(matriz);
    }

    // Método para inserir dados na matriz
    /**
     * @param matriz
     * @param caractere
     * @param linha
     * @param coluna
     */
    public static void inserirJogada(char matriz[][], char caractere, int linha, int coluna)
{
        if (matriz[linha][coluna] == 'O' | matriz[linha][coluna] == 'X') {
            System.out.println(ANSI_RED + "Jogada irregular, tente uma posição ainda
não ocupada." + ANSI_RESET);
        } else {
            matriz[linha][coluna] = caractere;
            System.out.println("Jogada realizada com sucesso.");
        }
    }

    // Método para mostrar a matriz atualizada
    /**
     * @param matriz
     */
    public static void mostrarMatriz(char matriz[][]) {
```

```
        System.out.println();
        System.out.println();

        for (int linha = 0; linha < matriz.length; linha++) {
            for (int coluna = 0; coluna < matriz[linha].length; coluna++) {
                System.out.print("\t" + matriz[linha][coluna] + "  ");
            }
             System.out.println(); // Pula para a próxima linha após imprimir uma linha completa
        }
        System.out.println();
    }

    // Método para checar Vitória
    /**
     * @param matriz
     */
    public static void checkWin(char matriz[][]) {

        // Definindo as possíveis combinações de vitória
        char[] diagonal1 = {matriz[0][0], matriz[1][1], matriz[2][2]};
        char[] diagonal2 = {matriz[2][0], matriz[1][1], matriz[0][2]};
        char[] column1 = {matriz[0][0], matriz[1][0], matriz[2][0]};
        char[] column2 = {matriz[0][1], matriz[1][1], matriz[2][1]};
        char[] column3 = {matriz[0][2], matriz[1][2], matriz[2][2]};
        char[] row1 = {matriz[0][0], matriz[0][1], matriz[0][2]};
        char[] row2 = {matriz[1][0], matriz[1][1], matriz[1][2]};
        char[] row3 = {matriz[2][0], matriz[2][1], matriz[2][2]};

        // Criando a matriz com todas as combinações de vitória
        char[][] winMatrix = {
            diagonal1, diagonal2,
```

```
            column1, column2, column3,
            row1, row2, row3
        };

        // Verificando se algum dos arrays de vitória tem todos os elementos iguais
        for (int i = 0; i < winMatrix.length; i++) {
            if (checkArrays(winMatrix[i])) {
                    System.out.println(ANSI_GREEN + winMatrix[i][0] + " venceu.\n" +
ANSI_RESET);
                stopGame = true;
                return; // Interrompe a verificação após encontrar um vencedor
            }
        }
    }

    // Método auxiliar para verificar se todos os elementos do array são iguais
    /**
     * @param array
     * @return
     */
    public static boolean checkArrays(char[] array) {
        if (array.length == 0 || array[0] == '-') { // Exclui células vazias (representadas
por '-')
            return false;
        }
        char firstElement = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i] != firstElement) {
                return false;
            }
        }
        return true;
```

```
    }
}
```

Fonte: O autor.

CONCLUSÃO

Os requisitos funcionais e técnicos estabelecidos no Material de Aprendizagem Prática e Avaliação (MAPA) foram atendidos, conforme mostrado no *Quadro 1* e demostrado através das *Figuras* enumeradas de 1 a 5.

REFERÊNCIAS BIBLIOGRÁFICAS

ascii.co.uk. Disponível em: <https://ascii.co.uk/> Acesso em: 11 out 2024.

MICROSOFT Corporation (2024a). Visual Studio Code. Disponível em: https://code.visualstudio.com/ Acesso em: 11 out. 2024.

MICROSOFT CORPORATION (2024b). Windows 10 Pro, versão 22H2. Disponível em: https://www.microsoft.com Acesso em: 11 out. 2024.

ORACLE CORPORATION. Java SE Development Kit 21. Disponível em: https://www.oracle.com/java/technologies/javase/jdk21-archive-downloads.html Acesso em: 11 out. 2024.