# Ramo Estudantil IEEE - UEL

FELIPE GUILHEN LUKASZCZUK
(felipe.guilhen@uel.br)
JOÃO PEDRO ALMEIDA AZEVEDO
(joao.pedro.almeida@uel.br)
PEDRO TOMONAGA SCHETTINI
(pedro.schettini@uel.br)
SUZANY YUKO SHIRAISCHI TSURU
(suzany.shiraischi@uel.br)

**RELATÓRIO FINAL:**

GERADOR DE CAÇA-PALAVRAS

Londrina
2023

Contato do Ramo: sb.uel@ieee.org
Institute of Electrical and Electronics Engineers – IEEE
Universidade Estadual de Londrina - UEL • Paraná - Brasil

# Ramo Estudantil IEEE - UEL

FELIPE GUILHEN LUKASZCZUK
JOÃO PEDRO ALMEIDA AZEVEDO
PEDRO TOMONAGA SCHETTINI
SUZANY YUKO SHIRAISCHI TSURU

## RELATÓRIO FINAL:

GERADOR DE CAÇA-PALAVRAS

Relatório apresentado ao Ramo Estudantil IEEE da Universidade Estadual de Londrina.

**Diretor de Projetos:** Nathan Andreani Netzel
**Gestores de Projetos**: Daniel Tresse Dourado, Levi Monteiro dos Santos

Londrina
2023

Contato do Ramo: sb.uel@ieee.org
Institute of Electrical and Electronics Engineers – IEEE
Universidade Estadual de Londrina - UEL • Paraná - Brasil

LUKASZCZUK, Felipe Guilhen. AZEVEDO, João Pedro Almeida. SCHETTINI, Pedro Tomonaga. TSURU, Suzany Yuko Shiraischi. **Relatório Final**: GERADOR DE CAÇA-PALAVRAS. 2023. XX folhas. Relatório apresentado ao Ramo Estudantil IEEE da Universidade Estadual de Londrina, Londrina, 2023.

## RESUMO

O projeto "Gerador de Caça-Palavras" foi desenvolvido em Linguagem C de programação e sua principal missão reside em desenvolver uma ferramenta que seja lúdica, mas também enriqueça o processo de ensino. Bem como colocar em prática alguns conceitos apreendidos em sala de aula e o estudo e aplicabilidade de novas noções de programação em Linguagem C. Este projeto oferece uma oportunidade valiosa de aprimoramento dos estudos, além de ser um projeto que promove a criatividade e o trabalho em equipe dos discentes.

**Palavras-chave:** Linguagem C. Caça-palavras. Programação.

## OBJETIVOS

A criação do projeto “Gerador de Caça-Palavras” teve como objetivo o desenvolvimento de um programa funcional, além de ter sido uma forma de aprimorar habilidades e adquirir novos conhecimentos de programação em Linguagem C. O programa deve preencher uma matriz de certo tamanho fornecido pelo usuário, com palavras escolhidas também pelo usuário, com letras aleatórias do alfabeto, e então, montar um tabuleiro de caça-palavras.

SUMÁRIO

# Sumário

## 1. INTRODUÇÃO

O projeto "Gerador de Caça-Palavras" representa uma iniciativa desafiadora dentro do âmbito acadêmico do ramo estudantil - IEEE. Este projeto foi desenvolvido com o objetivo de criar uma ferramenta eficiente, funcional e lúdica para a criação de caça-palavras personalizados.

Uma característica fundamental deste projeto é a utilização de funções estruturantes que facilitam a criação, verificação e manipulação do caça-palavras. Estas funções foram cuidadosamente projetadas e implementadas para oferecer uma experiência de usuário intuitiva e eficaz, tornando a compreensão de um possível processo para criação de um caça-palavras uma tarefa acessível a estudantes que estejam interessados no estudo da programação em Linguagem C.

## 2. METODOLOGIA

### 2.1 FUNÇÃO: POSIÇÃO DAS PALAVRAS

Após perguntar ao usuário o tamanho do tabuleiro, quantas e quais palavras serão inseridas na matriz final, e salvar essas palavras em um vetor, a algoritmo começa a inserir as palavras na matriz. Então, o algoritmo escolhe, aleatoriamente, por meio da função rand( ), uma coordenada (x,y) para começar a inserir a palavra fornecida, e uma direção para as letras seguintes serem inseridas.

Entretanto, e se a palavra começar a ser inserida em uma posição já ocupada pelas anteriores? O código garante, por meio de uma função, uma verificação na main e uma matriz auxiliar (matrizAux), que nenhuma palavra será inserida por cima da outra, ou sequer passará pelo limite do tabuleiro. Depois, ainda é preciso preencher o restante da matriz com letras aleatórias maiúsculas, de A a Z. Com a matriz resultante completa, a implementação pede para que o tabuleiro seja salvo em um arquivo .txt, que é o que a última das 5 funções principais faz.

Essa função é responsável por escolher as coordenadas da primeira letra da palavra a ser inserida, assim como a direção onde as letras continuarão, dentre 7 direções: horizontal, anda para a direita; vertical, anda para cima; diagonal 1, para direita e para cima; diagonal 2, para direita e para baixo; diagonal 3, anda para e esquerda e para cima; diagonal 4, anda para a esquerda e para baixo; de trás para frente, anda para a esquerda. Então, para cada direção, o algoritmo sorteia um número, que é comparado dentro de uma estrutura switch case. Para cada direção, as variáveis passoX e passoY recebem uma combinação de valores diferentes, a fim de subirem ou descerem uma linha ou coluna na matriz, para inserir a próxima letra.

Então, é chamada a função Verificacao (explicada no tópico 1.2), que se validada, chama a função InserirPalavra (explicada no tópico 1.3). É importante ressaltar que, a repetição while só é repetida enquanto o número de tentativas for

menor que a área da matriz, o que garante que se o algoritmo percorrer toda a matriz e não encontrar uma posição válida, não há onde encaixar essa palavra.

### 2.1.1 Função: Verificação

Esta é provavelmente a função que mais tomou tempo do grupo para resolver, pois já havíamos terminado o código quando percebemos que algumas palavras se sobressaiam perante as outras, sempre as que haviam sido inseridas por último. Então, precisaríamos de um método para garantir que isso não aconteceria, o que comprometeria o resultado.

Por ser uma função booleana, ou seja, que retorna simplesmente a falso ou verdadeiro, ela faz uma verificação pequena, mas importante. Ao percorrer cada letra da palavra a ser inserida, é verificado se a letra a ser colocada na matriz passou do limite do tabuleiro, seja nas linhas ou nas colunas. E, o mais importante, garante que não há uma palavra já posicionada anteriormente, pois se tiver, essa função retorna a falso e por meio do restante da função PosicaoDasPalavras, outra coordenada e direção são escolhidas. Se a verificação for bem-sucedida, a função retorna a verdadeiro e a palavra é inserida por meio da função InserirPalavra.

### 2.1.2 Função: Inserir palavras

Essa função fica encarregada de, após a verificação for correta e não tiver nenhuma outra palavra no caminho da direção a ser inserida, colocar a palavra de acordo com as coordenadas e a direção escolhida randomicamente pelo próprio algoritmo. Ela também a marca as posições onde já reside uma letra de uma palavra inserida anteriormente na matriz auxiliar, que participará da próxima verificação. A

matriz principal agora ganha mais uma palavra, e a matriz auxiliar fica preenchida com 1 nas posições preenchidas, e 0s nas posições vazias.

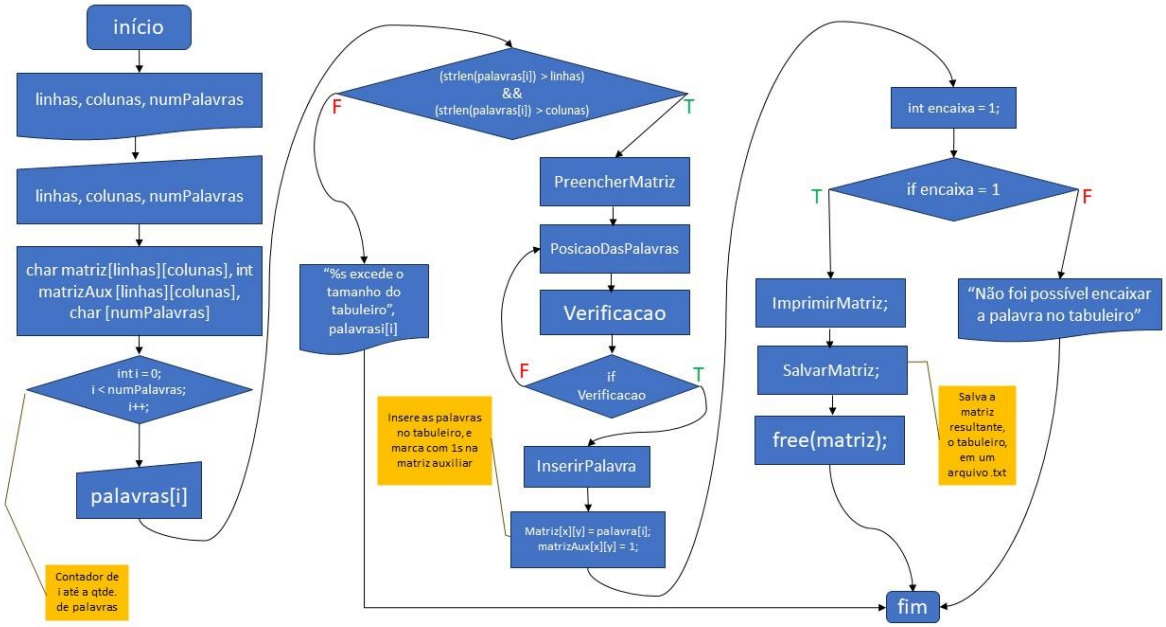
### 2.2 FUNÇÃO: PREENCHER

Essa função faz com que o tabuleiro seja preenchido por letras maiúsculas de A a Z, a fim de se misturarem com as palavras inseridas pelo usuário. Depois de preencher com letras aleatórias, o algoritmo insere as palavras que o usuário forneceu.

#### 2.2.1 Função: Salvar matriz

Essa função fica responsável para criar e abrir um arquivo, a fim de escrever cada elemento da matriz resultante dentro deste arquivo .txt.

**Figura 1** – Fluxograma



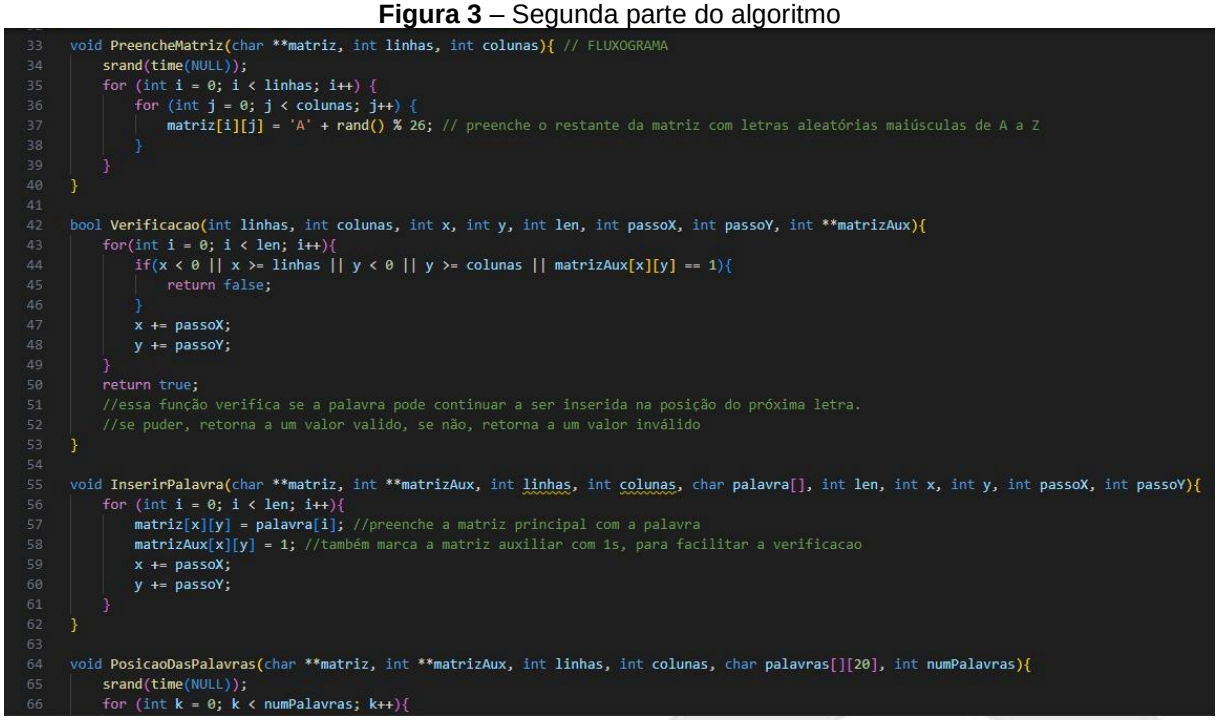

**Fonte:** Os autores

**Figura 2** – Primeira parte do algoritmo

```c
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <stdbool.h>

void ImprimirMatriz(char **matriz, int linhas, int colunas){
    for (int i = 0; i < linhas; i++){
        for (int j = 0; j < colunas; j++){
            printf("%c ", matriz[i][j]);
        }
        printf("\n");
    }
}

void SalvarMatriz(char **matriz, int linhas, int colunas){ //salva a matriz resultante, o tabuleiro, em um arquivo txt
    FILE *arquivo;
    arquivo = fopen("TABULEIRO.txt", "w");
    if (arquivo == NULL) {
        printf("Nao foi possivel criar o arquivo\n");
        return ;
    }
    for (int i = 0; i < linhas; i++){
        for (int j = 0; j < colunas; j++){
            fprintf(arquivo, "%c ", matriz[i][j]);
        }
        fprintf(arquivo, "\n");
    }
    fclose(arquivo);
    printf("Salvo em 'TABULEIRO.txt'.\n");
}

void PreencheMatriz(char **matriz, int linhas, int colunas){ // FLUXOGRAMA
    srand(time(NULL));
```

**Fonte**: Os autores

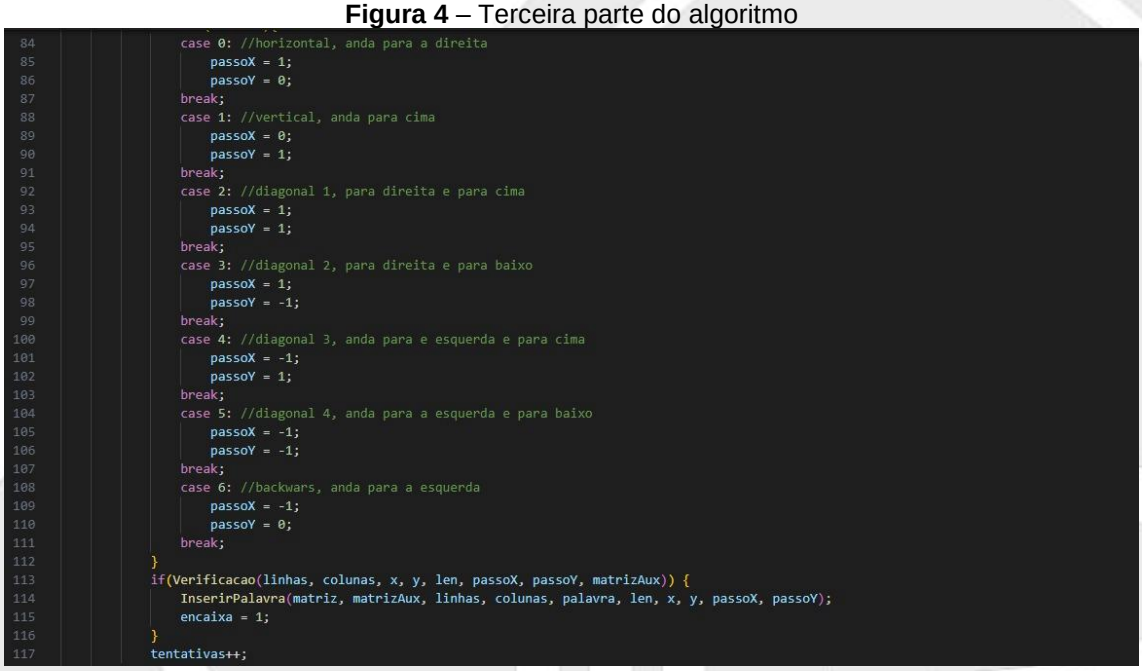
**Figura 3** – Segunda parte do algoritmo



```
33  void PreencheMatriz(char **matriz, int linhas, int colunas){ // FLUXOGRAMA
34      srand(time(NULL));
35      for (int i = 0; i < linhas; i++) {
36          for (int j = 0; j < colunas; j++) {
37              matriz[i][j] = 'A' + rand() % 26; // preenche o restante da matriz com letras aleatórias maiúsculas de A a Z
38          }
39      }
40  }
41
42  bool Verificacao(int linhas, int colunas, int x, int y, int len, int passoX, int passoY, int **matrizAux){
43      for(int i = 0; i < len; i++){
44          if(x < 0 || x >= linhas || y < 0 || y >= colunas || matrizAux[x][y] == 1){
45              return false;
46          }
47          x += passoX;
48          y += passoY;
49      }
50      return true;
51      //essa função verifica se a palavra pode continuar a ser inserida na posição do próxima letra.
52      //se puder, retorna a um valor valido, se não, retorna a um valor inválido
53  }
54
55  void InserirPalavra(char **matriz, int **matrizAux, int linhas, int colunas, char palavra[], int len, int x, int y, int passoX, int passoY){
56      for (int i = 0; i < len; i++){
57          matriz[x][y] = palavra[i]; //preenche a matriz principal com a palavra
58          matrizAux[x][y] = 1; //também marca a matriz auxiliar com 1s, para facilitar a verificacao
59          x += passoX;
60          y += passoY;
61      }
62  }
63
64  void PosicaoDasPalavras(char **matriz, int **matrizAux, int linhas, int colunas, char palavras[][20], int numPalavras){
65      srand(time(NULL));
66      for (int k = 0; k < numPalavras; k++){
```

**Fonte**: Os autores

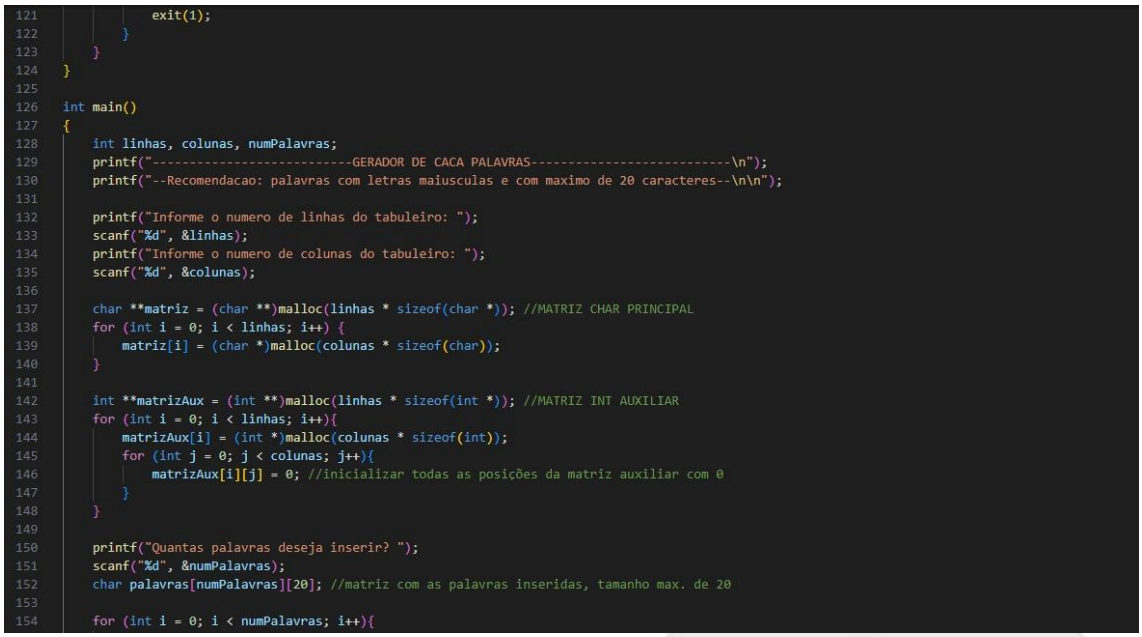
**Figura 4** – Terceira parte do algoritmo



```
84                  case 0: //horizontal, anda para a direita
85                      passoX = 1;
86                      passoY = 0;
87                  break;
88                  case 1: //vertical, anda para cima
89                      passoX = 0;
90                      passoY = 1;
91                  break;
92                  case 2: //diagonal 1, para direita e para cima
93                      passoX = 1;
94                      passoY = 1;
95                  break;
96                  case 3: //diagonal 2, para direita e para baixo
97                      passoX = 1;
98                      passoY = -1;
99                  break;
100                 case 4: //diagonal 3, anda para e esquerda e para cima
101                     passoX = -1;
102                     passoY = 1;
103                 break;
104                 case 5: //diagonal 4, anda para a esquerda e para baixo
105                     passoX = -1;
106                     passoY = -1;
107                 break;
108                 case 6: //backwars, anda para a esquerda
109                     passoX = -1;
110                     passoY = 0;
111                 break;
112             }
113             if(Verificacao(linhas, colunas, x, y, len, passoX, passoY, matrizAux)) {
114                 InserirPalavra(matriz, matrizAux, linhas, colunas, palavra, len, x, y, passoX, passoY);
115                 encaixa = 1;
116             }
117             tentativas++;
```

**Fonte**: Os autores

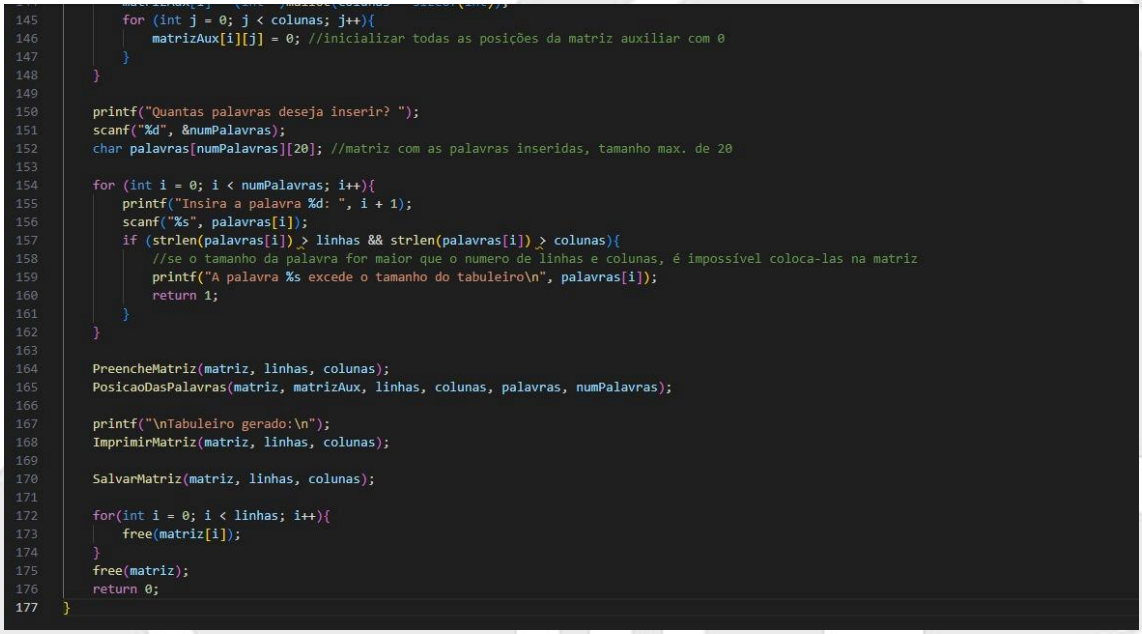
**Figura 5** – Quarta parte do algoritmo



```
            exit(1);
        }
    }
}

int main()
{
    int linhas, colunas, numPalavras;
    printf("--------------------------GERADOR DE CACA PALAVRAS-------------------------\n");
    printf("--Recomendacao: palavras com letras maiusculas e com maximo de 20 caracteres--\n\n");

    printf("Informe o numero de linhas do tabuleiro: ");
    scanf("%d", &linhas);
    printf("Informe o numero de colunas do tabuleiro: ");
    scanf("%d", &colunas);

    char **matriz = (char **)malloc(linhas * sizeof(char *)); //MATRIZ CHAR PRINCIPAL
    for (int i = 0; i < linhas; i++) {
        matriz[i] = (char *)malloc(colunas * sizeof(char));
    }

    int **matrizAux = (int **)malloc(linhas * sizeof(int *)); //MATRIZ INT AUXILIAR
    for (int i = 0; i < linhas; i++){
        matrizAux[i] = (int *)malloc(colunas * sizeof(int));
        for (int j = 0; j < colunas; j++){
            matrizAux[i][j] = 0; //inicializar todas as posições da matriz auxiliar com 0
        }
    }

    printf("Quantas palavras deseja inserir? ");
    scanf("%d", &numPalavras);
    char palavras[numPalavras][20]; //matriz com as palavras inseridas, tamanho max. de 20

    for (int i = 0; i < numPalavras; i++){
```

**Fonte**: Os autores

**Figura 6** – Quinta parte do algoritmo



```
        for (int j = 0; j < colunas; j++){
            matrizAux[i][j] = 0; //inicializar todas as posições da matriz auxiliar com 0
        }
    }

    printf("Quantas palavras deseja inserir? ");
    scanf("%d", &numPalavras);
    char palavras[numPalavras][20]; //matriz com as palavras inseridas, tamanho max. de 20

    for (int i = 0; i < numPalavras; i++){
        printf("Insira a palavra %d: ", i + 1);
        scanf("%s", palavras[i]);
        if (strlen(palavras[i]) > linhas && strlen(palavras[i]) > colunas){
            //se o tamanho da palavra for maior que o numero de linhas e colunas, é impossível coloca-las na matriz
            printf("A palavra %s excede o tamanho do tabuleiro\n", palavras[i]);
            return 1;
        }
    }

    PreencheMatriz(matriz, linhas, colunas);
    PosicaoDasPalavras(matriz, matrizAux, linhas, colunas, palavras, numPalavras);

    printf("\nTabuleiro gerado:\n");
    ImprimirMatriz(matriz, linhas, colunas);

    SalvarMatriz(matriz, linhas, colunas);

    for(int i = 0; i < linhas; i++){
        free(matriz[i]);
    }
    free(matriz);
    return 0;
}
```

**Fonte**: Os autores

## 3. PROBLEMAS ENCONTRADOS

Durante o processo de desenvolvimento do projeto, uma das questões cruciais que surgiram foi a sobreposição de palavras dentro do caça-palavras. Isso significa que, em alguns casos, as palavras escolhidas para serem inseridas na matriz se cruzavam ou ocupavam a mesma posição, prejudicando a legibilidade do quebra-cabeças. Esse problema exigiu uma análise minuciosa e ajustes na lógica do algoritmo para garantir que cada palavra fosse inserida de forma independente, sem conflitos.

A fim de tornar o código mais compreensível e facilitar a manutenção, realizamos uma alteração significativa na função de verificação. Inicialmente, essa função retornava um valor inteiro, mas percebemos que essa abordagem poderia ser simplificada, retornando a um valor falso ou verdadeiro, ou seja, utilizando uma função booleana.

Outro desafio encontrado durante o desenvolvimento foi relacionado à aleatoriedade na escolha das direções para a inserção das palavras na matriz. Para melhorar a experiência do usuário e garantir que as palavras fossem inseridas de maneira mais compreensível, foi implementado a escolha randômica das direções.

Ao iniciar o projeto, contávamos com um determinado número de membros na equipe de desenvolvimento, ao longo do processo de desenvolvimento, ocorreram mudanças na composição da equipe. A redução na equipe alterou a dinâmica de trabalho, bem como exigiu uma redistribuição de responsabilidades e um reajuste nas metas de desenvolvimento. Essa adaptação foi essencial para garantir que o projeto continuasse progredindo de maneira funcional e que os objetivos fossem alcançados dentro do cronograma estabelecido.

## 4. RESULTADOS E DISCUSSÕES

Durante a fase de desenvolvimento do projeto, identificamos que o algoritmo estava sobrepondo palavras no caça-palavras e para resolver esse problema criamos uma matriz auxiliar que registrasse as posições ocupadas na matriz principal. Essa matriz auxiliar seria preenchida com valores 0s e 1s, onde 0 indicaria uma posição livre e 1 indicaria uma posição ocupada. Dessa forma, antes de inserir uma palavra na matriz principal, o algoritmo consultaria a matriz auxiliar para garantir que não houvesse sobreposição de letras.

Para aprimorar a clareza e a compreensão do código, decidimos realizar uma mudança na função de verificação. Inicialmente, essa função retornava um valor inteiro, o que tornava a interpretação dos resultados um pouco mais complexa para iniciantes na programação em C e na própria lógica de programação. Para facilitar a leitura do código por todos os membros da equipe, alteramos a função para retornar um valor booleano, ou seja, verdadeiro ou falso. Essa mudança simplificou consideravelmente o processo de verificação e tornou as condições de verificação mais explicativas.

Na função responsável por inserir palavras aleatoriamente no caça-palavras, enfrentamos o desafio das escolhas randômicas das direções. No contexto desse projeto, era importante que as direções de inserção fossem aleatórias para criar caça-palavras diversificados. A solução encontrada envolveu a geração de números aleatórios de 0 a 6, onde cada número representava uma direção específica. Cada direção foi associada a um valor numérico, permitindo que o algoritmo escolhesse a direção aleatoriamente, tornando o processo de criação do caça-palavras mais dinâmico e variado.

Em resumo, ao longo do desenvolvimento do projeto, enfrentamos desafios que exigiram criatividade e adaptação. A solução para cada problema demonstrou a habilidade do grupo em trabalhar em equipe para que o projeto gerador de caça-palavras fosse desenvolvido dentro do prazo estabelecido. Esses ajustes e inovações

foram cruciais para garantir a qualidade do projeto e sua utilidade tanto para fins educacionais.

## 5. CONCLUSÕES

Em conclusão, ao longo do desenvolvimento do projeto de geração de caça-palavras, enfrentamos desafios significativos que exigiram soluções flexíveis e ajustes. Cada um desses desafios foi superado com êxito, resultando em melhorias significativas na funcionalidade do código.

A identificação e resolução da sobreposição de palavras por meio da matriz auxiliar contribuiu para garantir clareza. Além disso, a mudança na função de verificação para retornar valores booleanos simplificou a leitura e a manutenção do código, tornando-o mais acessível a todos os membros da equipe. Por fim, a resolução das escolhas randômicas das direções proporcionou caça-palavras mais dinâmicos.

Esses ajustes e inovações não apenas aprimoraram o código e a funcionalidade da ferramenta, mas também demonstraram a capacidade da equipe de superar os desafios e cooperação entre os integrantes do grupo.

## REFERÊNCIAS BIBLIOGRÁFICAS

**Matrizes e Arrays multidimensionais em C**. Disponível em: https://www.youtube.com/watch?v=8d1bH8bkj5Q

SOFFNER, Renato. **Algoritmos e Programação em linguagem C** / Renato Soffner. - 1. ed. - São Paulo: Saraiva, 2013.