

PROJETO:

JOGOS VORAZES

GRUPO 8 POR:

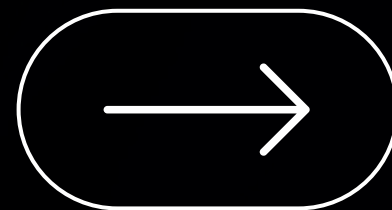
ANA CLARA, MIKAELE E WINNE

SOBRE O JOGO


O jogo consiste em uma simulação onde um tributo está preso em um labirinto e precisa escapar enquanto monstros se movimentam e invadem diferentes partes do labirinto. O objetivo do tributo é encontrar uma saída, alcançando uma das bordas do labirinto, sem ser capturado pelos monstros ou bloqueado por obstáculos, como paredes ou áreas invadidas.

ESTRUTURAS USADAS

Foram usadas as seguintes estruturas para organizar as informações e controlar o fluxo de ações:




Macros



```
1 //definindo as macros, true como 1 e false como 0.
2 #define true 1
3 #define false 0
4
5
6 // definindo a macro do tipo int para booleano, só pra
  ser mais fácil
7 typedef int booleano;
```

Labirinto



```
1 typedef struct{
2     int altura;
3     int largura;
4     char **matriz;
5     int tributoX, tributoY;
6 }Labirinto;
```

Fila

```
1 typedef struct auxNo{
2     int x, y;
3     struct auxNo* prox;
4 }No;
5
6 typedef struct{
7     No *inicio;
8     No *fim;
9 }Fila;
```

Direção



```
1  typedef struct {  
2      int x, y;  
3      char dir;  
4  } Direcao;  
5  
6  extern Direcao movimentos[4];
```



```
1  extern Direcao movimentos[4];  
    //usei o extern para essa chamada global não ir para os outros arquivos!  
2  //Além disso, fiz um vetor da struct, para armazenar 4 posições!.
```

Main

```
1  int main() {
2      int n, m;
3      scanf("%d %d", &n, &m);
4
5      Labirinto labirinto;
6      labirinto.altura = n;
7      labirinto.largura = m;
8      labirinto.matriz = (char **)malloc(n * sizeof(char *));
9      for (int i = 0; i < n; i++) {
10         labirinto.matriz[i] = (char *)malloc(m * sizeof(char));
11     }
12
13     for (int i = 0; i < n; i++) {
14         for (int j = 0; j < m; j++) {
15             char c;
16             scanf(" %c", &c);
17             labirinto.matriz[i][j] = c;
18             //Definição da posição inicial do Tributo.
19             if (c == 'A') {
20                 labirinto.tributoX = i;
21                 labirinto.tributoY = j;
22             }
23         }
24     }
```


Main

```
1 // Verifica se já é possível escapar de imediato
2 /* Essa função verifica se o tributo está em uma das bordas, se ele estiver,
3 já é possível ele escapar e imprime 0. Portanto, não é preciso fazer a BFS ou propagar
4 porque ele já pode escapar.
5 */
6 if (labirinto.tributoX == 0 || labirinto.tributoY == 0 ||
7     labirinto.tributoX == n - 1 || labirinto.tributoY == m - 1) {
8     printf("YES\n0\n");
9     return 0;
10 }
11
12 // A propagação dos monstros do tributo e a BFS começa aqui
13 buscaEmLargura(&labirinto);
14
15 // Liberação da memória.
16 for (int i = 0; i < n; i++) {
17     free(labirinto.matriz[i]);
18 }
19 free(labirinto.matriz);
20
21 return 0;
```



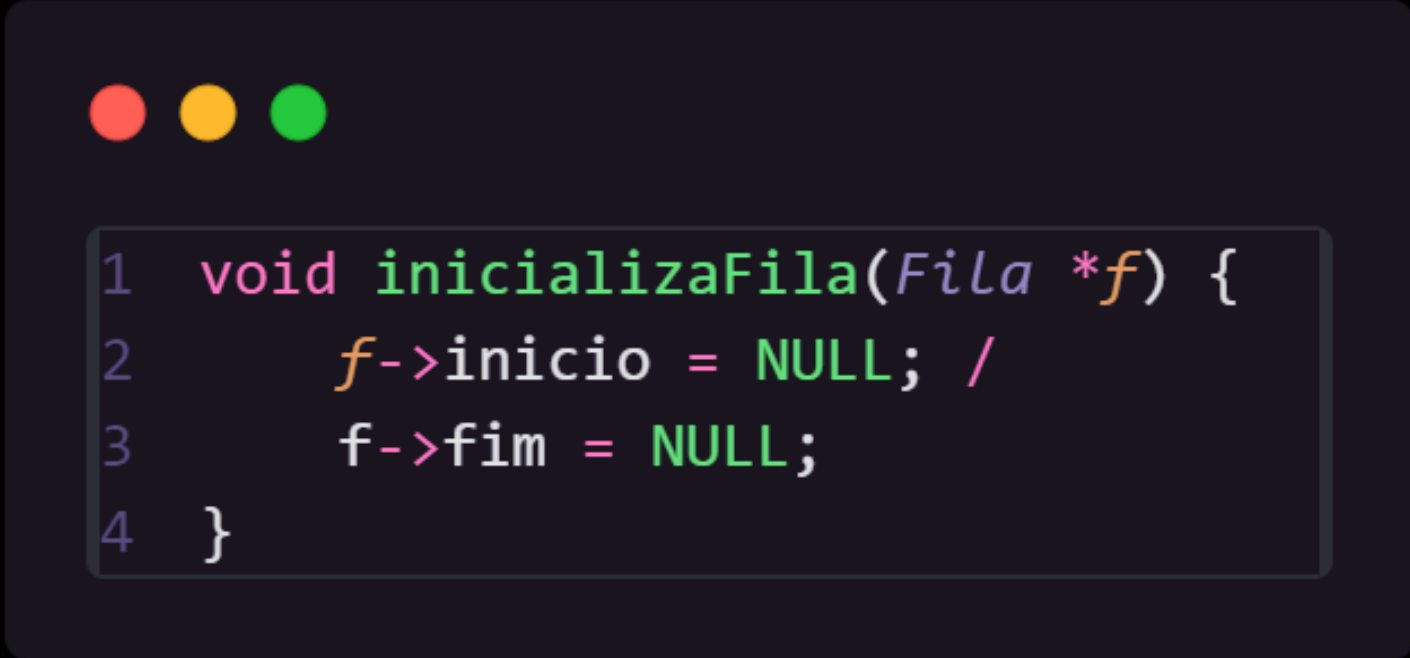
```
1 Direcao movimentos[4] = {{-1, 0, 'U'}, {1, 0, 'D'}, {0, -1, 'L'}, {0, 1, 'R'}};  
2
```

Representa as direções possíveis para se mover no labirinto. Cada direção é representada por um par de coordenadas (x, y) e um caractere:

- $\{-1, 0, 'U'\}$: Move para cima ('U' para "Up").
- $\{1, 0, 'D'\}$: Move para baixo ('D' para "Down").
- $\{0, -1, 'L'\}$: Move para a esquerda ('L' para "Left").
- $\{0, 1, 'R'\}$: Move para a direita ('R' para "Right").

FUNÇÕES GERAIS:


- Função para inicializar a Fila



```
1 void inicializaFila(Fila *f) {  
2     f->inicio = NULL; //  
3     f->fim = NULL;  
4 }
```

FUNÇÕES GERAIS:

- Função para verificar se a fila está vazia



```
1  booleano filaVazia(Fila *f) {  
2      return (f->inicio == NULL);  
3  }
```

FUNÇÕES GERAIS:

- Função para Inserir

```
1 void insereFila(Fila *f, int x, int y) {
2     No *novo = (No *)malloc(sizeof(No));
3     novo->x = x;
4     novo->y = y;
5     novo->prox = NULL;
6
7     if (f->fim == NULL) {
8         f->inicio = novo;
9         f->fim = novo;
10    } else {
11        f->fim->prox = novo;
12        f->fim = novo;
13    }
14 }
```

FUNÇÕES GERAIS:

- Função para Excluir

```
1  No *excluiFila(Fila *f) {
2      if (filaVazia(f)) return NULL;
3
4      No *atual = f->inicio;
5      f->inicio = atual->prox;
6
7      if (f->inicio == NULL) {
8          f->fim = NULL;
9      }
10     return atual;
11 }
```

FUNÇÕES GERAIS:

- Essa função verifica as posições válidas.

```
1  booleano posicaoValida(Labirinto *labirinto, int x, int y) {
2      if (x < 0 || y < 0 || x >= labirinto->altura || y >=
    labirinto->largura) return false;
3      if (labirinto->matriz[x][y] == '#') return false;
4
5      if (labirinto->matriz[x][y] == 'M') return false;
6
7      if (labirinto->matriz[x][y] == 'V') return false;
8      if (labirinto->matriz[x][y] == 'M') return false;
9
10     return true;
```

FUNÇÕES GERAIS:

- Verifica se a posição está em uma borda do labirinto.

```
1  booleano isEscape(Labirinto *labirinto, int x, int y) {  
2      if ((x == 0 || y == 0 || x == labirinto->altura - 1  
3          || y == labirinto->largura - 1)) {  
4          return true;  
5      }  
6      return false;  
7  }
```


Busca Em Largura

- Função de Busca que implementa o algoritmo BFS para explorar o labirinto e tentar encontrar uma rota de escape para o tributo.

```
1  booleano buscaEmLargura(Labirinto *labirinto) {
2      Fila fila;
3      inicializaFila(&fila);
4
5      // Matriz de direções para armazenar de onde o tributo veio
6      char **direcoes = (char **)malloc(labirinto->altura * sizeof(char *));
7      for (int i = 0; i < labirinto->altura; i++) {
8          direcoes[i] = (char *)malloc(labirinto->largura * sizeof(char));
9          for (int j = 0; j < labirinto->largura; j++) {
10             direcoes[i][j] = '\\0'; // Inicializa as direções como vazias
11         }
12     }
13
14     // Insere todos os monstros na fila primeiro
15     for (int i = 0; i < labirinto->altura; i++) {
16         for (int j = 0; j < labirinto->largura; j++) {
17             if (labirinto->matriz[i][j] == 'M') {
18                 insereFila(&fila, i, j); // Adiciona monstros na fila
19             }
20         }
21     }
```

Busca Em Largura

- Função de Busca que implementa o algoritmo BFS para explorar o labirinto e tentar encontrar uma rota de escape para o tributo.

```
1 // Depois, insere o tributo
2 insereFila(&fila, labirinto->tributoX, labirinto->tributoY);
3 labirinto->matriz[labirinto->tributoX][labirinto->tributoY] = 'V';
4 // Marca como visitado pelo tributo
5
6 while (!filaVazia(&fila)) {
7     No *atual = excluiFila(&fila);
8     int x = atual->x;
9     int y = atual->y;
10    free(atual);
11
12    // Verifica se é um monstro
13    if (labirinto->matriz[x][y] == 'M') {
14        // Movimentação dos monstros
15        for (int i = 0; i < 4; i++) {
16            int novoX = x + movimentos[i].x;
17            int novoY = y + movimentos[i].y;
18
19            if (posicaoValida(labirinto, novoX, novoY)) {
20                labirinto->matriz[novoX][novoY] = 'M';
21            }
22        }
23    }
24    // Marca como invadido por monstros
25    insereFila(&fila, novoX, novoY);
26
27    // Caso seja o tributo
28    else if (labirinto->matriz[x][y] == 'V') {
29        // Movimentação do tributo
30        for (int i = 0; i < 4; i++) {
31            int novoX = x + movimentos[i].x;
32            int novoY = y + movimentos[i].y;
```


Busca Em Largura

- Função de Busca que implementa o algoritmo BFS para explorar o labirinto e tentar encontrar uma rota de escape para o tributo.

```
1 // Imprime o comprimento do caminho e o caminho
2         printf("%d\n", passos);
3         for (int j = passos - 1; j >= 0; j--) {
4             printf("%c", caminho[j]);
5         }
6         printf("\n");
7         return true;
8     }
9 }
10 }
11 }
12 }
13
14 // Se não for possível escapar
15 printf("NO\n");
16 return false;
17 }
18
```

OBRIGADA