

# **UNIVERSIDADE FEDERAL DE VIÇOSA**

PAA - PROJETO E ANÁLISE DE ALGORITMO

LEANDRO LÁZARO ARAÚJO VIEIRA - EF03513

MATEUS PINTO DA SILVA - EF03489

## **BILUMAZE.C**

Solucionador de Labirintos Alienígenas

## Sumário

<b>1. Introdução</b>	<b>3</b>
<b>2. O problema</b>	<b>4</b>
<b>3. Solução</b>	<b>5</b>
<b>4. Alguns detalhes da implementação</b>	<b>6</b>
<b>5. Exemplos de uso</b>	<b>9</b>
<b>6. Desafios encontrados</b>	<b>13</b>
<b>7. Diferenciais</b>	<b>14</b>
<b>8. Conclusão</b>	<b>17</b>

## **1. Introdução**

Este trabalho consiste na construção de um solucionador de labirintos alienígenas utilizando backtracking na linguagem C. Assim que finalizado o projeto, um usuário poderá adicionar ou gerar aleatoriamente labirintos e, por fim, solucioná-los.

## **2. O problema**

O problema consiste em encontrar o mais rápido possível uma saída para um labirinto gerado aleatoriamente ou inserido pelo usuário através de um arquivo txt. Além disso, outro desafio é desenvolver o algoritmo responsável por encontrar esse caminho utilizando a técnica de backtracking.

### **3. Solução**

Primeiramente, para armazenar os labirintos, foi criado o TAD maze que, além de conter o tamanho do labirinto, contém um parâmetro de ponteiro para ponteiro de um dado do tipo TAD cell que nada mais é do que a matriz dinamicamente alocada que representa o labirinto em si. O TAD maze também armazena o número de chaves que o estudante contém e a posição onde ele está.

Em segundo lugar, como já citado, foi criado o TAD cell que busca representar cada posição da matriz, ou seja, cada posição do labirinto. Esse TAD armazena um indicador de passagem utilizado posteriormente no algoritmo de backtracking e um identificador que define qual o conteúdo daquela célula (espaço vazio, parede, porta, estudante, chave ou saída).

Por último, o algoritmo utilizado para encontrar o caminho potencial de executar quatro movimentos possíveis em relação a uma célula (esquerda, cima, direita e baixo respectivamente). Enquanto ele não é impedido de executar um dos movimentos, chama a célula correspondente a direção do movimento executado recursivamente tentando nela os mesmo movimentos e, assim como fez anteriormente, não muda de direção enquanto não for necessário. Dessa forma, se houver saída para o labirinto, será encontrado um caminho até ela e não necessariamente será o menor caminho possível.

#### 4. Alguns detalhes da implementação

O primeiro detalhe de grande importância é o algoritmo de backtracking. O algoritmo começa pela célula onde está o estudante e partir dela, explora células ao seu redor avaliando se é possível navegar por elas, isto é, se essas células são espaços vazios, se contém chaves, se é uma saída, se é uma porta e nesse caso se o estudante tem alguma chave para abri-la, se é uma parede ou se é uma célula que o estudante já passou. Caso uma das três primeiras condições seja atendida, o algoritmo leva o estudante para essa célula e faz os mesmos testes. No caso de uma das condições de sucesso não ser atendida, o algoritmo tenta mover o estudante para outras direções. No entanto, se não for possível mover para nenhuma das direções, o algoritmo leva o estudante de volta para uma célula já percorrida onde é possível ainda efetuar algum movimento diferente dos que já foram efetuados nesta célula. Dessa forma, se houver alguma saída para o labirinto, o algoritmo irá encontrá-la assim que encontrar uma célula que se identifique com saída.

```
typedef struct cell{  
    typeCell _typeCell;  
    int pathUsed;  
}cell;
```

Figura 4.1 - TAD cell - representa uma célula do labirinto.

```
typedef struct maze{
    int sizeX;
    int sizeY;
    cell **_cell;
    int studentCoordinateX;
    int studentCoordinateY;
    int keysNumber;
}maze;
```

Figura 4.2 - TAD maze - representa um labirinto.

```
int mazeMoveStudent(maze *_maze, int backtrackingCoordinateY, int
backtrackingCoordinateX, int *sucessCoordinateY, int *sucessCoordinateX, int *movements,
stack ** exitRoute);
```

Figura 4.3 - Escopo da função de backtracking.

Além disso, para debugar o código foram criadas bandeiras chamadas DEBUG que executam uma versão diferente do código caso no momento de compilar o usuário a defina junto ao comando no console.

```

#ifdef DEBUG

    int mazeMoveStudent(maze *_maze, int backtrackingCoordinateY, int
        backtrackingCoordinateX, int *sucessCoordinateY, int *sucessCoordinateX, int
        *movements, int *recursiveCalls, stack ** exitRoute);

#endif

#ifndef DEBUG

    int mazeMoveStudent(maze *_maze, int backtrackingCoordinateY, int
        backtrackingCoordinateX, int *sucessCoordinateY, int *sucessCoordinateX, int
        *movements, stack ** exitRoute);

#endif

```

Figura 4.6 - Exemplo de implementação da bandeira DEBUG na função de backtracking. Toda vez que DEBUG é definida na compilação a primeira versão da função é compilada e a segunda ignorada.

Por último, outra parte do código que merece um pouco de atenção é a estrutura de dados stack. Ela nada mais é do que uma pilha para armazenar os movimentos necessários para se chegar na saída encontrados pelo algoritmo de backtracking. Para exibir esse caminho, basta desempilhar essa pilha.

```

typedef struct stack {

    int X;

    int Y;

    struct stack * next;

} stack;

```

Figura 4.4 - TAD stack.

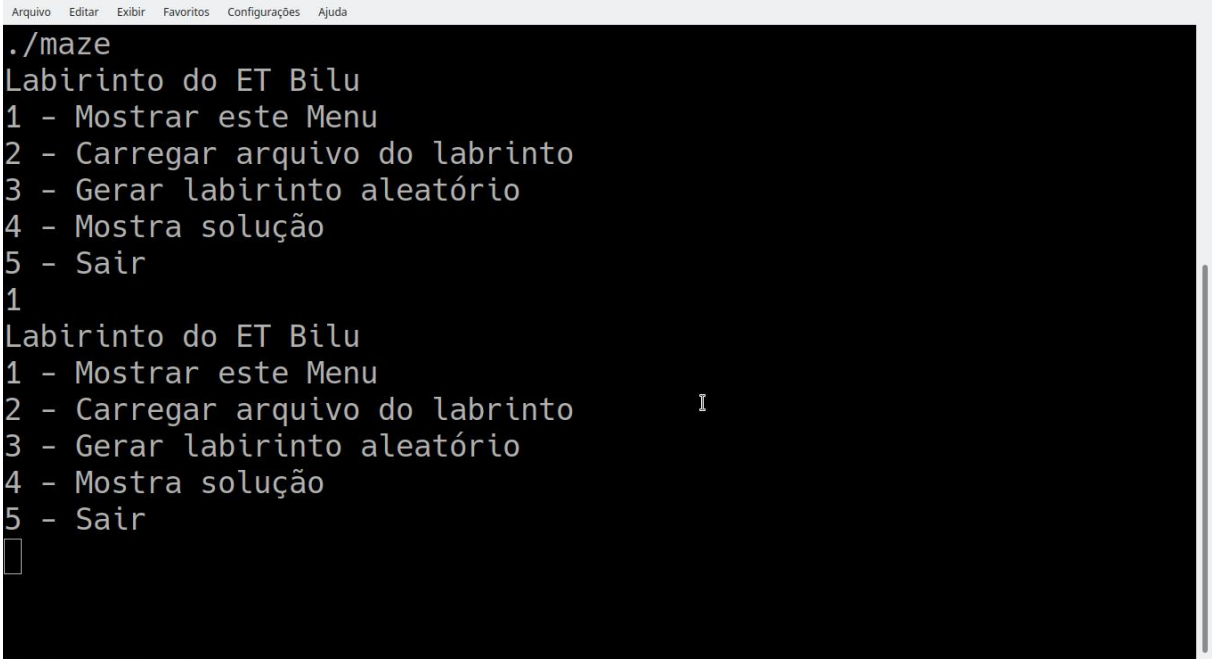


## 5. Exemplos de uso

Para utilizar o BILUMAZE.C, o solucionador de labirintos alienígenas, é preciso possuir o gcc (compilador) instalado em sua máquina. Após atender esse requisito, basta abrir o terminal, navegar até a pasta do projeto e digitar make (para compilar) ou make debug (caso queira compilar a versão de debug) e, em seguida, make run (para executar). Depois disso, basta selecionar uma das opções sugeridas assim como é mostrado nos exemplos.

### 5.1 Exemplo 1:

Opção selecionada: 1



```
Arquivo  Editar  Exibir  Favoritos  Configurações  Ajuda
./maze
Labirinto do ET Bilu
1 - Mostrar este Menu
2 - Carregar arquivo do labrinto
3 - Gerar labirinto aleatório
4 - Mostra solução
5 - Sair
1
Labirinto do ET Bilu
1 - Mostrar este Menu
2 - Carregar arquivo do labrinto
3 - Gerar labirinto aleatório
4 - Mostra solução
5 - Sair
█
```

TP1-Backtracking\_PAA : make

## 5.2 Exemplo 2:

Opção selecionada: 2

```
Arquivo  Editar  Exibir  Favoritos  Configurações  Ajuda
Labirinto do ET Bilu
1 - Mostrar este Menu
2 - Carregar arquivo do labrinto
3 - Gerar labirinto aleatório
4 - Mostra solução
5 - Sair
2
Digite o nome do arquivo txt:
maze.txt
Montando labirinto...
5555555555
2222322222
1111111111
1111111111
2222322222
1111111111
3222222111
1111112111
1222222111
1111111111
1111111111
1111011111
Labirinto montado com sucesso!
Digite outra opção ou digite 1 para mostrar o menu novamente

```

## 5.3 Exemplo 3:

Opção selecionada: 3

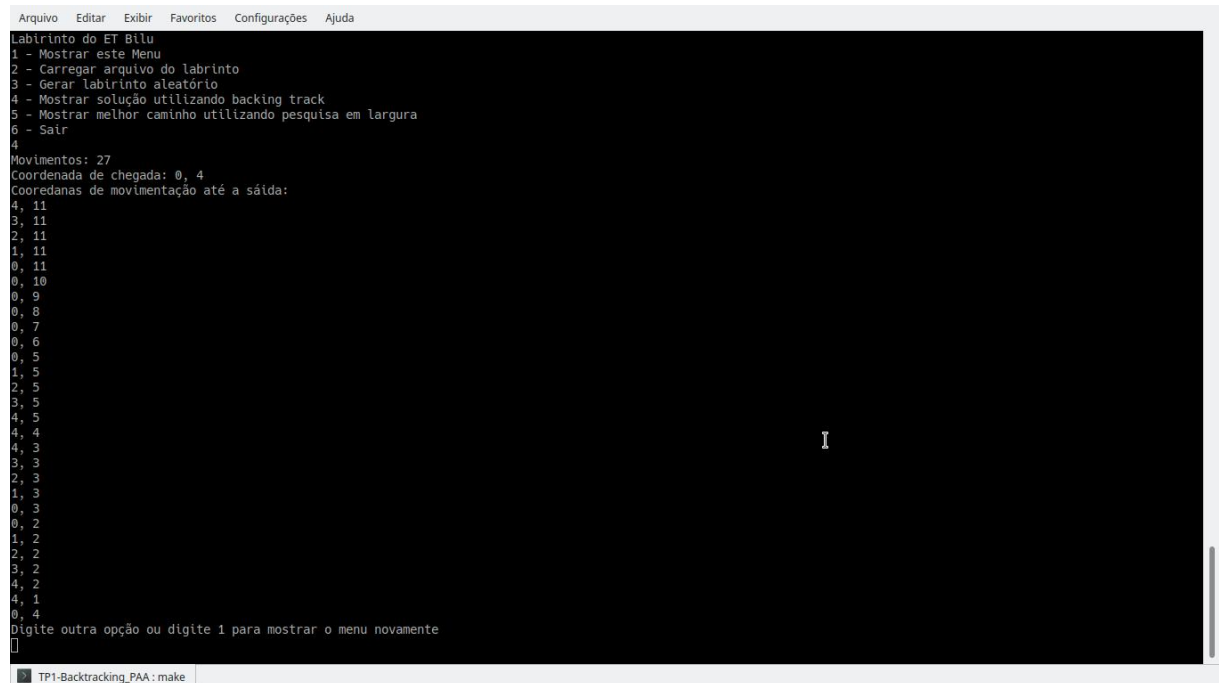
```
Arquivo  Editar  Exibir  Favoritos  Configurações  Ajuda
Labirinto do ET Bilu
1 - Mostrar este Menu
2 - Carregar arquivo do labrinto
3 - Gerar labirinto aleatório
4 - Mostra solução
5 - Sair
3
Escolha uma dificuldade:
0-Fácil:
1-Médio
2-Difícil
1
Montando labirinto...
5555555555
111122232
2111123222
1111131111
1222231111
1111121111
1121221111
1121221111
122222322
1211231111
1211221111
1311231111
121112222
1111121111
4111222321
122322232
1222222111
111110111
Labirinto montado com sucesso!
Digite outra opção ou digite 1 para mostrar o menu novamente

```

## 5.4 Exemplo 4:

Opção selecionada: 4

Observação: A solução encontrada equivale ao labirinto gerado selecionando a opção 2.

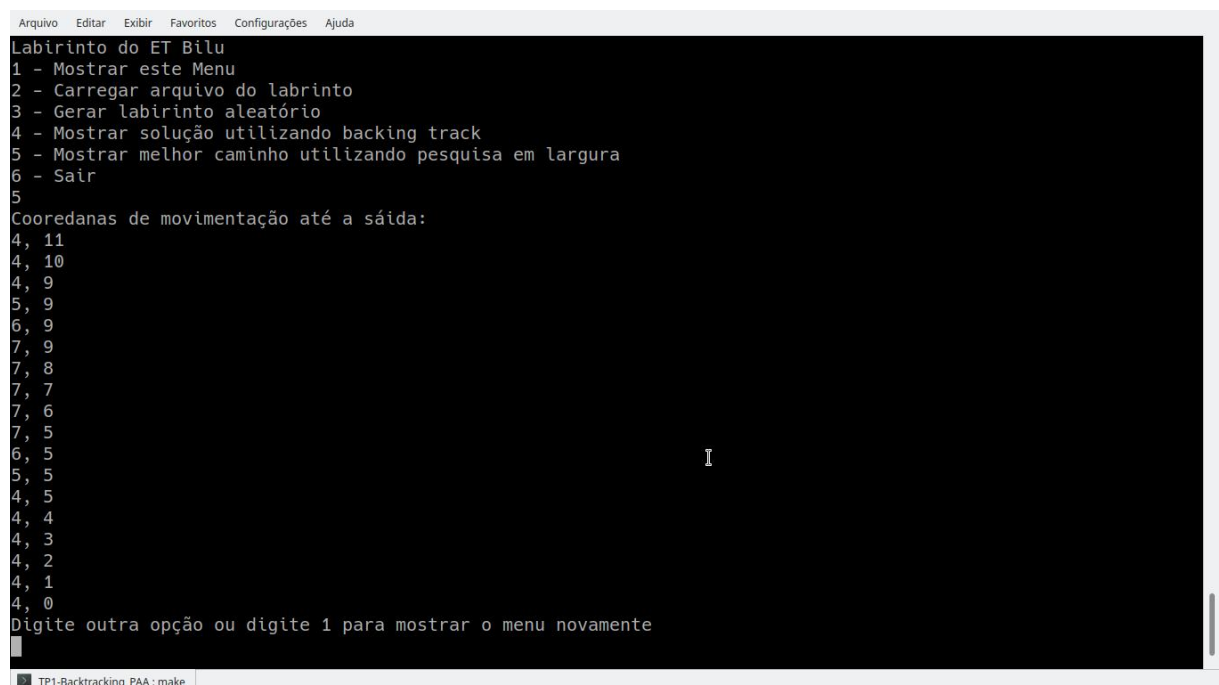


```
Arquivo  Editar  Exibir  Favoritos  Configurações  Ajuda
Labirinto do ET Bilu
1 - Mostrar este Menu
2 - Carregar arquivo do labirinto
3 - Gerar labirinto aleatório
4 - Mostrar solução utilizando backing track
5 - Mostrar melhor caminho utilizando pesquisa em largura
6 - Sair
4
Movimentos: 27
Coordenada de chegada: 0, 4
Coordenadas de movimentação até a saída:
4, 11
3, 11
2, 11
1, 11
0, 11
0, 10
0, 9
0, 8
0, 7
0, 6
0, 5
1, 5
2, 5
3, 5
4, 5
4, 4
4, 3
3, 3
2, 3
1, 3
0, 3
0, 2
1, 2
2, 2
3, 2
4, 2
4, 1
0, 4
0, 4
Digite outra opção ou digite 1 para mostrar o menu novamente
1
```

## 5.6 Exemplo 6:

Opção selecionada: 5

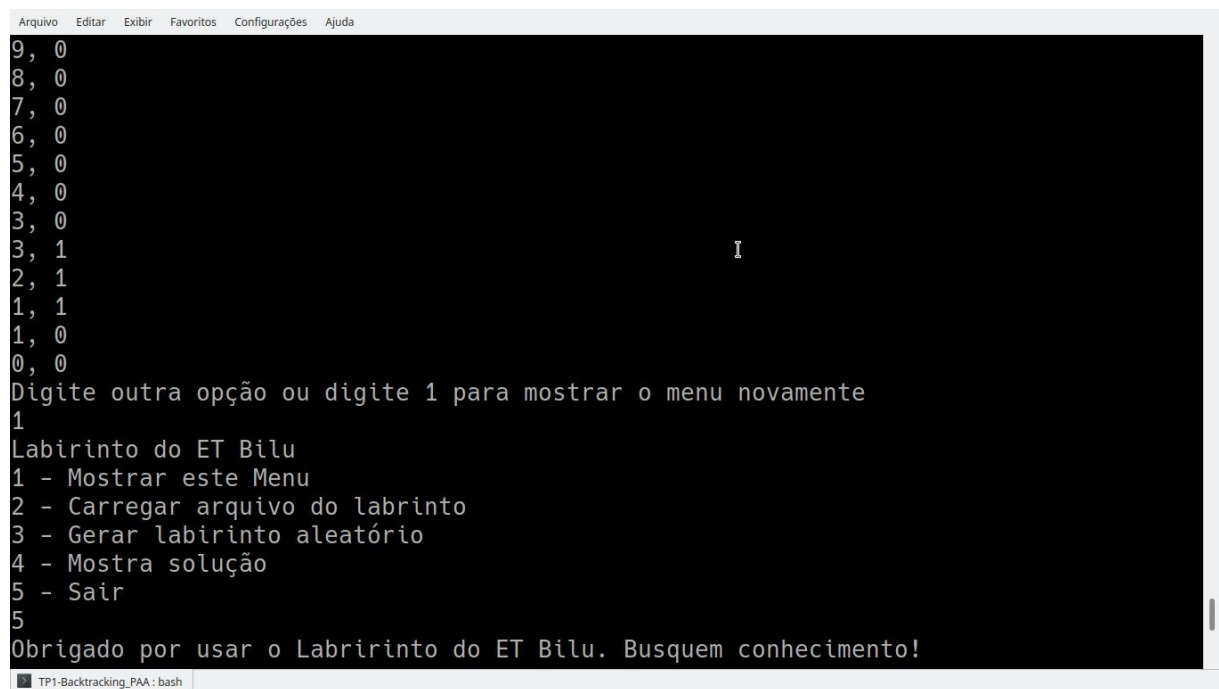
Observação: A solução encontrada equivale ao labirinto gerado selecionando a opção 2.



```
Arquivo  Editar  Exibir  Favoritos  Configurações  Ajuda
Labirinto do ET Bilu
1 - Mostrar este Menu
2 - Carregar arquivo do labirinto
3 - Gerar labirinto aleatório
4 - Mostrar solução utilizando backing track
5 - Mostrar melhor caminho utilizando pesquisa em largura
6 - Sair
5
Coordenadas de movimentação até a saída:
4, 11
4, 10
4, 9
5, 9
6, 9
7, 9
7, 8
7, 7
7, 6
7, 5
6, 5
5, 5
4, 5
4, 4
4, 3
4, 2
4, 1
4, 0
4, 0
Digite outra opção ou digite 1 para mostrar o menu novamente
1
```

## 5.6 Exemplo 6:

Opção selecionada: 5



```
Arquivo  Editar  Exibir  Favoritos  Configurações  Ajuda
9, 0
8, 0
7, 0
6, 0
5, 0
4, 0
3, 0
3, 1
2, 1
1, 1
1, 0
0, 0
Digite outra opção ou digite 1 para mostrar o menu novamente
1
Labirinto do ET Bilu
1 - Mostrar este Menu
2 - Carregar arquivo do labirinto
3 - Gerar labirinto aleatório
4 - Mostra solução
5 - Sair
5
Obrigado por usar o Labirinto do ET Bilu. Busquem conhecimento!
```

TP1-Backtracking\_PAA: bash

## **6. Desafios encontrados**

Embora que fazer o mínimo necessário para entregar esse trabalho tenha sido relativamente fácil, o algoritmo extra para encontrar o melhor caminho foi um grande desafio para nós e por pouco quase não conseguimos entregar uma solução dentro do prazo. No entanto, assimilando o problema com técnicas aprendidas na disciplina de grafos, conseguimos solucionar o problema de forma incrivelmente eficiente se comparado com as soluções tentadas anteriormente.

## 7. Diferenciais

O primeiro diferencial é o algoritmo gerador de labirintos aleatórios. Esse algoritmo cria labirintos com três dificuldades diferentes (fácil, médio e difícil). Para criar labirintos de dificuldade diferente, ele muda o domínio de sorteio de determinados elementos do labirinto, como número de paredes, número máximo de portas por parede, número de chaves com o estudante e tamanho do labirinto.

```
//Facil
if(dificult==0){

    sizeY=(rand() % (10 - 5 + 1)) + 5;

    sizeX=(rand() % (10 - 5 + 1)) + 5;

    wallQuantity=(rand() % (5 - 0 + 1)) + 0;

    keysQuantity=(rand() % (5 - 0 + 1)) + 0;

    limiteDoorPerWallQuantity=(rand() % (5 - 0 + 1)) + 0;

    amountkeysOnFloor=(rand() % (10 - 0 + 1)) + 0;

//Medio
}else if(dificult==1){

    sizeY=(rand() % (20 - 10 + 1)) + 10;

    sizeX=(rand() % (20 - 10 + 1)) + 10;

    wallQuantity=(rand() % (20 - 10 + 1)) + 10;

    keysQuantity=(rand() % (20 - 0 + 1)) + 0;

    limiteDoorPerWallQuantity=(rand() % (2 - 0 + 1)) + 0;

    amountkeysOnFloor=(rand() % (5 - 0 + 1)) + 0;

//Difícil
}else{

    sizeY=(rand() % (30 - 20 + 1)) + 20;
```

```

sizeX=(rand() % (300 -20 + 1)) + 20;

wallQuantity=(rand() % (15 - 10 + 1)) + 10;

keysQuantity=(rand() % (15 - 0 + 1)) + 0;

limiteDoorPerWallQuantity=(rand() % (2 - 0 + 1)) + 0;

amountkeysOnFloor=(rand() % (2 - 0 + 1)) + 0;
}

```

Figura 7.1 - Trecho de código da função **mazeInitRandomMaze** responsável por definir a dificuldade do labirinto a ser gerado baseado no parâmetro de entrada **difficulty** definido pelo usuário em tempo de execução.

O segundo grande diferencial foi a implementação de um eficiente algoritmo para encontrar o melhor caminho entre onde o estudante está e alguma saída. Para isso, utilizamos a ideia de busca em largura, já que, dessa forma o primeiro caminho encontrado até a saída necessariamente é o mais curto. Como auxílio foi utilizado o TAD row (uma fila) que em cada um dos nós armazena uma posição da matriz, a quantidade de chaves do estudante quando esteve nessa posição e um endereço para o nó anterior e posterior. Essa fila tem como responsabilidade armazenar vários caminhos (e no final da execução o menor deles) e saber qual célula da matriz já foi percorrida por cada caminho.

```

typedef struct row {

    int X;

    int Y;

    int keysQuantity;

    struct row * previous;

    struct row * next;

} row;

```

Figura 7.2 - TAD row.

Dentre os diferenciais ainda é necessário citar uma modificação feita por nós quanto a identificação das células do labirinto. Da forma que fizemos é possível colocar a saída não somente na primeira linha da matriz (ou no topo do labirinto) mas em qualquer lugar e ainda assim o algoritmo conseguirá encontrar uma saída (se for possível chegar até ela). Adotamos o seguinte padrão:

- 0- Estudante;
- 1- Espaço Vazio;
- 2- Parede;
- 3- Porta;
- 4- Chave;
- 5- Saída.

Por último e não menos importante, é necessário citar que embora os arquivos do projeto estejam no formato cpp e hpp (arquivos C++), o trabalho foi inteiramente codificado na linguagem C. Essa alteração ocorreu devido a ideia inicial de utilizar GTK para implementar uma interface gráfica, no entanto essa ideia foi abandonada no decorrer da produção do projeto.



## **8. Conclusão**

Este trabalho foi bem eficaz mostrando que o backtracking é bem relativo no que diz respeito a sua eficiência, pois para alguns casos apresenta desempenho satisfatório e outros não, já que, se colocada uma parede a mais em um labirinto que levava cerca de alguns milissegundos para ser resolvido, o tempo de execução passava de minutos ou até mesmo horas. Portanto, é uma técnica que tem suas aplicações, mas deve ser usada com cautela e sabedoria.