SERVIÇO PÚBLICO FEDERAL - MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA - UFV
CAMPUS FLORESTAL

Trabalho 1 - Projeto e análise de algoritmos

Bruno Vicentini Ribeiro - 5907
Erich Pinheiro Amaral - 5915
Fabricio Henrique Viana Albino - 5925
Pedro Paulo Paz do Nascimento - 5937
Vitor Mathias Andrade - 5901

Florestal - MG
2025

SUMÁRIO

INTRODUÇÃO

Este trabalho objetiva a implementação de um algoritmo baseado no paradigma *backtracking* — algoritmo que busca, por meio de força bruta, soluções para problemas computacionais —, que trace um percurso viável que leve a nave Expresso Interestelar de um ponto inicial ao denominado Planeta das Festividades, em um mapa de rotas interligadas.

Na situação, será necessária a administração racional da durabilidade da nave — que é reduzida a cada movimento realizado —, bem como da coleta de peças de reparo, que aumentam a durabilidade do Expresso Interestelar.

Sabendo disso, o algoritmo explora, de forma recursiva, as diferentes rotas possíveis, reiniciando o trajeto à medida que encontra situações adversas que impossibilitem a chegada ao seu destino final. Destaca-se, também, que o algoritmo comporta a alocação dinâmica de memória, o que permite que o programa seja executado com mapas de tamanhos variados.

Desse modo, o trabalho busca uma melhor compreensão de algoritmos baseados em *backtracking* e de custo computacional, o qual é visualizado por meio do modo de análise, responsável por contabilizar a quantidade máxima de chamadas recursivas e a profundidade máxima.

ORGANIZAÇÃO

Na figura 1, observa-se o panorama geral da organização do projeto, no qual estão elencados os arquivos necessários para o trabalho. Esses arquivos foram separados em pastas para garantir melhor organização e funcionamento do código.

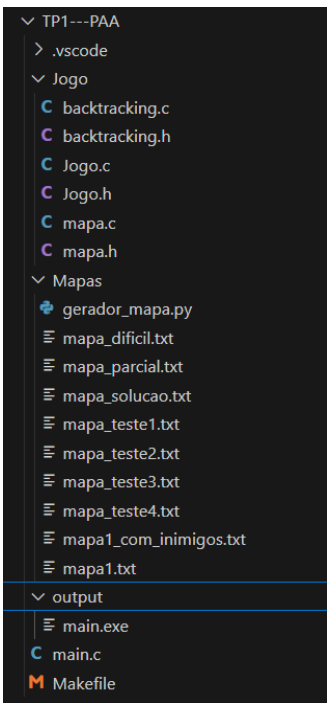

Figura 1: Repositório do projeto.

DESENVOLVIMENTO

- EXPLICAÇÃO DO ALGORITMO

O algoritmo de *backtracking* é uma técnica que procura soluções para um problema de forma recursiva, testando diversos caminhos até encontrar uma solução válida. Quando encontra um caminho que não leva ao destino desejado, ele retrocede e busca outras rotas. No contexto do trabalho, o algoritmo explora todas as rotas possíveis que a nave pode percorrer pelo mapa, respeitando as seguintes condições:

- Não é permitido ultrapassar os limites do mapa;
- Não é permitido passar por posições que tenham “.”;
- Cada setor da rota só pode visitado uma vez no mesmo caminho;
- O algoritmo também leva em consideração a durabilidade da nave, os inimigos, as peças de reparo coletadas e a condição de sucesso, que é chegar ao destino final.

A cada movimento realizado, o algoritmo verifica a durabilidade da nave e se o caminho é possível. Se o trajeto não levar ao destino final, o algoritmo retrocede um passo, restaura a durabilidade, os inimigos e as peças coletadas, e tenta fazer outro movimento. Esse processo é realizado até encontrar uma solução que leve ao destino final ou até encontrar todos os caminhos possíveis.

- IMPLEMENTAÇÃO DO ALGORITMO

A função responsável pelo *backtracking* é a `movimentar()`. Essa função contabiliza e controla a profundidade máxima atingida pelo algoritmo, além de armazenar o estado atual da nave (durabilidade, peças coletadas, peças restantes e status dos inimigos), permitindo que seja possível retroceder sem perder o estado anterior.

Ela verifica se a posição atual da nave possui algum inimigo (caso possua, a nave sofre um dano) e também confere a durabilidade na posição atual. Se existir um caminho possível até o destino final, esse caminho é armazenado.

A função marca a posição atual para evitar ciclos e utiliza as seguintes funções auxiliares para se movimentar: `movimentarDireita`, `movimentarEsquerda`, `movimentarCima` e `movimentarBaixo`. Após explorar todas as direções, o algoritmo desfaz as alterações realizadas, retornando ao estado anterior e garantindo que todos os caminhos sejam testados. Por fim, retorna o melhor caminho encontrado.

```c
int movimentar(Jogo* jogo, int linha, int coluna, int profundidade_atual) {
    jogo->dados->chamadas_recursivas++;
    if (profundidade_atual > jogo->dados->max_recursao) {
        jogo->dados->max_recursao = profundidade_atual;
    }

    Mapa* mapa = jogo->mapa_atual;
    Nave* nave = jogo->nave_atual;

    int durabilidade_anterior = nave->durabilidadeAtual;
    int pecas_coletadas_anterior = nave->pecasColetadas;
    int pecas_restantes_anterior = nave->pecasRestantes;

    int inimigo_index = -1;
    StatusInimigo status_anterior_inimigo;

    for (int i = 0; i < jogo->num_inimigos; i++) {
        if (jogo->inimigos[i].linha == linha && jogo->inimigos[i].coluna == coluna) {
            if (jogo->inimigos[i].status == ATIVO) {
                inimigo_index = i;
                status_anterior_inimigo = jogo->inimigos[i].status;
                nave->durabilidadeAtual -= jogo->inimigos[i].dano;
                jogo->inimigos[i].status = DERROTADO;
            }
            break;
        }
    }

    if (nave->durabilidadeAtual <= 0) {
        if (mapa->grid[linha][coluna] != 'F') {
            if (inimigo_index != -1) {
                jogo->inimigos[inimigo_index].status = status_anterior_inimigo;
            }
```

Figura 2: Função de movimentação.

- ESTRUTURA DE DADOS

A `struct Mapa` armazena uma matriz de caracteres que representa o mapa *(grid)*, uma matriz booleana com as posições já visitadas *(visitados)*, a altura e a largura do mapa, além da quantidade total de peças a serem coletadas.

```c
typedef struct Mapa{
    int altura, largura;
    int total_pecas;
    char** grid;
    bool** visitados;
} Mapa;
```

Figura 3: Estrutura “Mapa”.

A `struct Nave` armazena a durabilidade da nave, o valor de durabilidade perdido a cada movimento, a durabilidade adicionada a cada peça coletada, o número de peças coletadas e restantes.

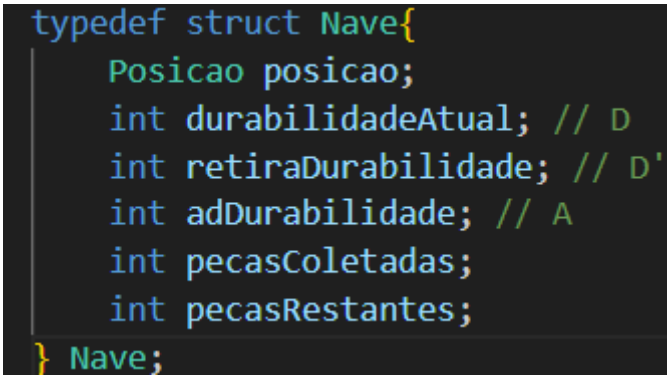

Figura 4: Estrutura “Nave”

A `struct Inimigo` armazena a posição no mapa em que o inimigo está localizado, o dano que ele causará na nave e o seu status (ativo ou derrotado).

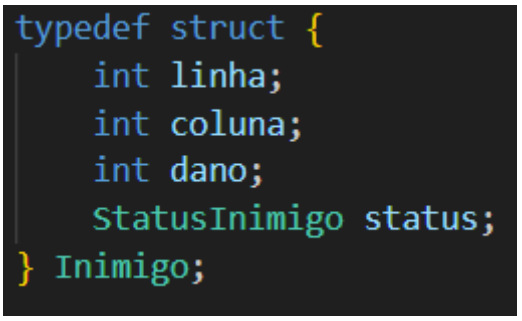

Figura 5: Estrutura “Inimigo”.

A `struct Dados` armazena a quantidade de vezes que a função `movimentar` foi chamada, além da maior profundidade atingida pelo algoritmo.

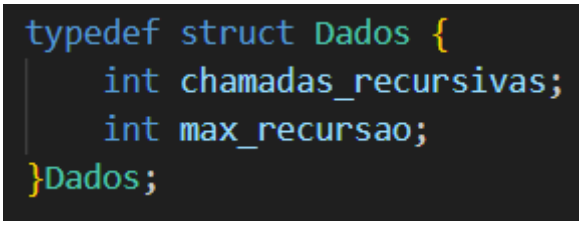

Figura 6: Estrutura “Dados”.

A `struct Jogo` armazena ponteiros para o mapa, a nave e os dados, além de um vetor de inimigos. Também contém matrizes para armazenar o percurso atual e o caminho final correto, bem como variáveis de controle (`tamanhoAtual`, `tamanhoFinal` e `num_inimigos`).

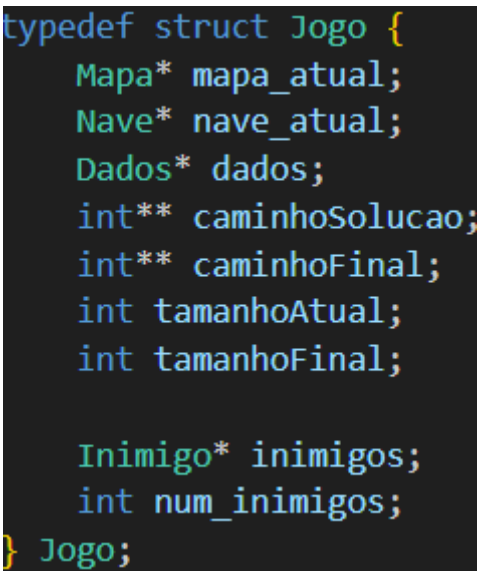

Figura 7: Estrutura “Jogo”.

- MODO DE ANÁLISE

O modo de análise tem como objetivo avaliar o desempenho do algoritmo de desempenho. Ele é ativado quando o usuário responde com “s” — sim — à seguinte pergunta:

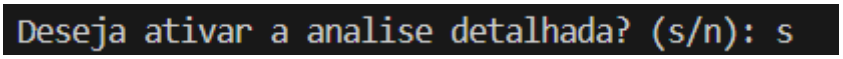

Figura 8: Pergunta de ativação do modo de análise.

Quando ativado, o modo de análise exibe ao usuário o número de chamadas recursivas e a profundidade máxima que foi atingida pelo algoritmo.

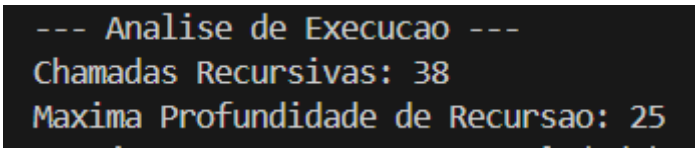

Figura 9: Resultado da análise.

Essas métricas permitem avaliar a complexidade do mapa e a eficiência do algoritmo.

- INTERFACE

A interface é baseada em entrada e saída no terminal, tornando a interação com o usuário simples e direta. Durante a execução do programa, o usuário deve informar o nome do arquivo de entrada, escolher se deseja ativar o modo de análise e, ao final, recebe os resultados da execução.

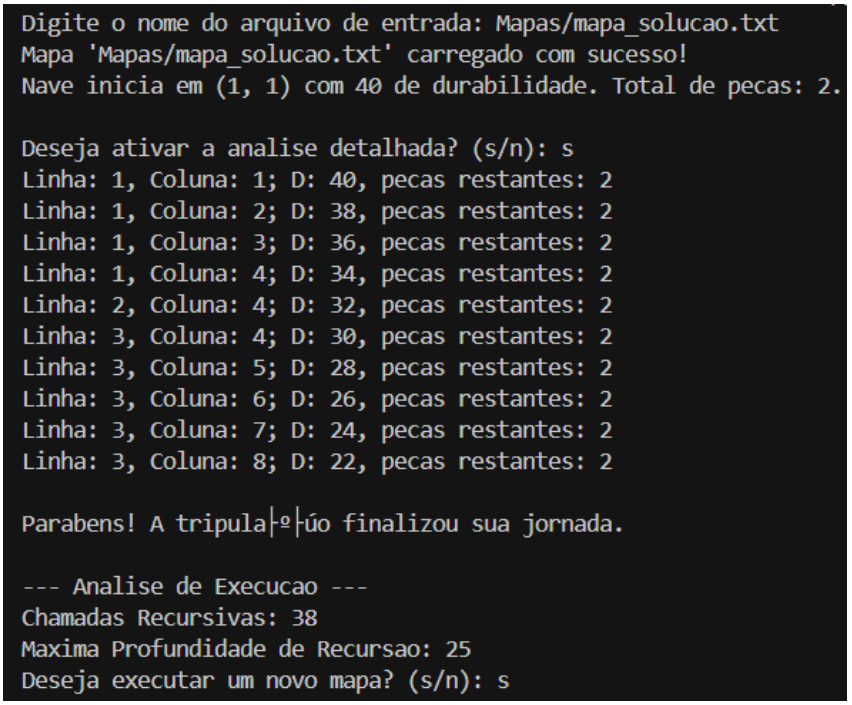

Figura 10: Interface do programa.

Essa abordagem permite uma visualização simples, porém completa, do comportamento do algoritmo.

- INIMIGOS

No programa, há inimigos posicionados em certos locais do mapa. Quando a nave entra em uma dessas posições, ela sofre dano, diminuindo sua durabilidade. Caso a nave possua durabilidade maior que zero após o ataque, o inimigo é marcado como “derrotado”, impedindo que cause dano novamente se a posição for revisitada. Se, após o dano sofrido pelo inimigo, a durabilidade da nave chegar a zero, o algoritmo retrocede e desfaz o movimento.

COMPILAÇÃO E EXECUÇÃO

Para facilitar a compilação e execução do programa, foi utilizado um arquivo *Makefile.* Para compilar, utiliza-se o seguinte comando:

```
None
make
```

Após isso, o programa é executado com:

```
None
app.exe
```

Em seguida, o usuário deve informar o arquivo de entrada. Como todos os mapas estão na pasta “Mapas”, o comando deve seguir o formato:

```
None
Mapas/nome_do_mapa.txt
```

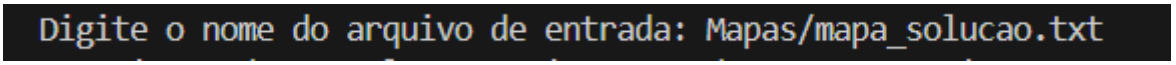

Figura 11: Seleção do arquivo de entrada.

RESULTADOS

Para avaliar o desempenho e o comportamento do algoritmo, foram testados diversos mapas de diferentes tamanhos, com e sem solução, e também inimigos. Além disso, quando o modo de análise é habilitado, o programa exibe, ao final, o número de chamadas recursivas e a profundidade máxima.

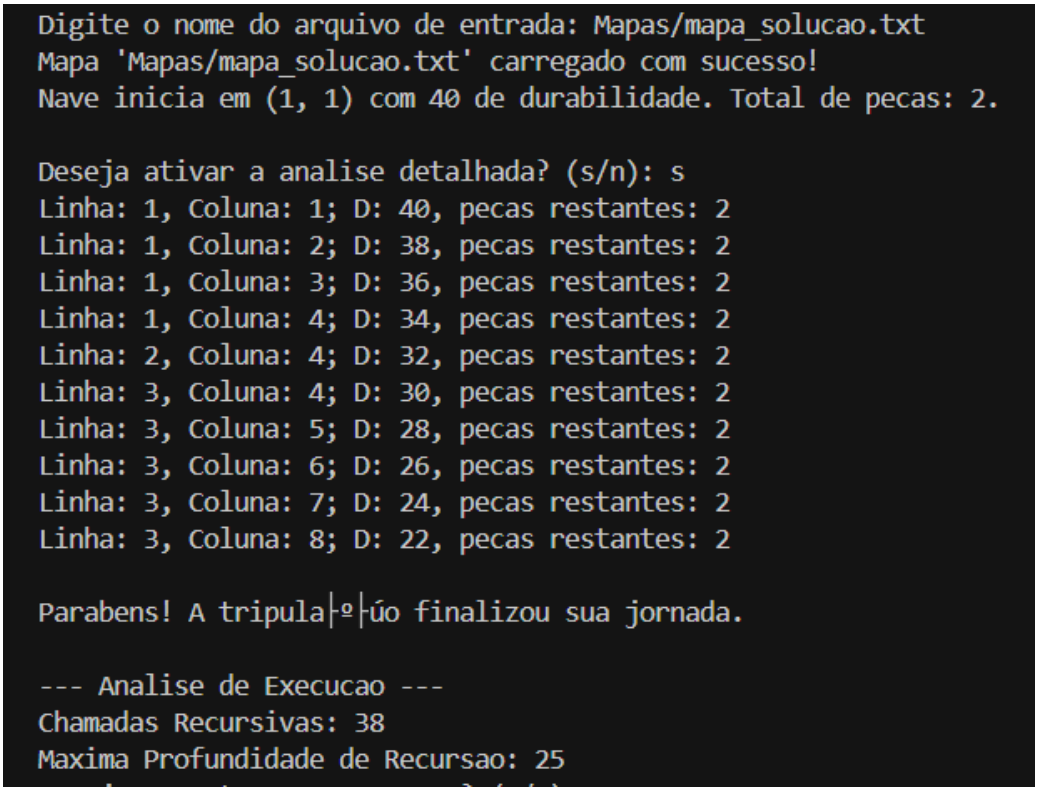

Figura 12: Mapa com solução.

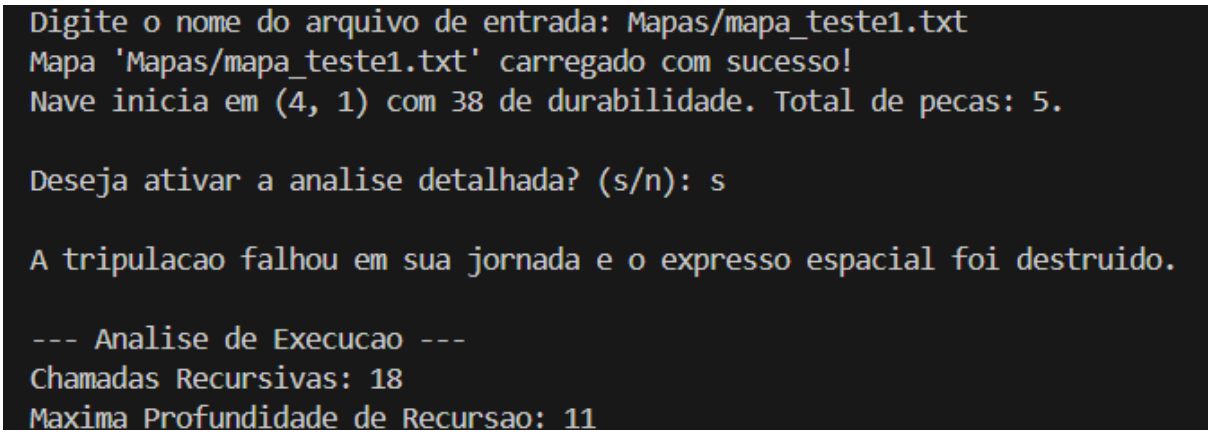

Figura 13: Mapa sem solução.

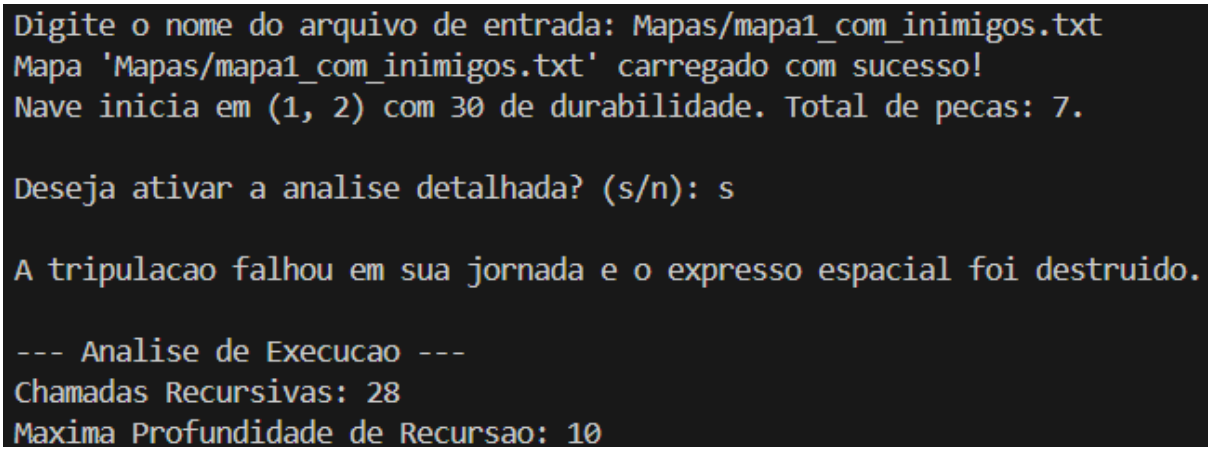

Figura 14: Mapa sem solução com inimigos.

CONCLUSÃO

A implementação do algoritmo de *backtracking* permitiu compreender, de forma prática, como essa técnica explora todas as possibilidades de solução em problemas complexos. No contexto do trabalho, o algoritmo demonstrou ser eficiente para encontrar a melhor rota até o destino final, respeitando restrições de durabilidade, inimigos, coletas de peças e limites do mapa.

Os resultados demonstraram que o *backtracking* sempre encontra uma solução, caso ela exista. Porém, observou-se que o número de chamadas recursivas cresce significativamente em mapas com muitos caminhos e ramificações, evidenciando sua limitação em problemas de grande complexidade.

A inclusão do modo de análise possibilitou melhor visualização do comportamento do algoritmo, enquanto os elementos adicionais — limites do mapa, peças e inimigos — aumentaram os desafios da busca por rotas possíveis.

Conclui-se, portanto, que o trabalho atingiu seus objetivos de aplicar e compreender o funcionamento do algoritmo de *backtracking*, reforçando a importância da escolha adequada de algoritmos para cada tipo de problema, a fim de uma melhor eficiência.

REFERÊNCIAS

- O QUE é um algoritmo Backtracking? *Stack Overflow em Português*, 2015. Disponível em: https://pt.stackoverflow.com/questions/103184/o-que-%C3%A9-um-algoritmo-backtracking. Acesso em: 11 de outubro de 2025.
- RAO, Pooja. *Backtracking in C*. StudyMite, [s. l.], 12 min read. Disponível em: https://www.studymite.com/blog/backtracking-in-c. Acesso em: 11 de outubro de 2025.