UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ (UTFPR)
CURSO SISTEMAS DE INFORMAÇÃO

GABRIELA B. TAQUEGAMI
LUCAS AMARAL
THIAGO K. TOYOMOTO

PATHFINDING COM FLOOD-FILL E LEE'S ALGORITHM

CURITIBA

2021

GABRIELA B. TAQUEGAMI
LUCAS AMARAL
THIAGO K. TOYOMOTO

PATHFINDING COM FLOOD-FILL E LEE'S ALGORITHM

Relatório Final da disciplina Estrutura de Dados I, do curso de Sistemas de Informação, apresentado ao professor que ministra a mesma na Universidade Tecnológica Federal do Paraná.

Orientador: Prof. PhD. Rodrigo Minetto

CURITIBA

2021

1 PATHFINDING COM FLOOD-FILL E LEE'S ALGORITHM

1.1 DESCRIÇÃO

Este trabalho tem como objetivo descobrir o caminho entre um ponto inicial e um ponto final em uma matriz *mxn*. Para tal, utilizou-se o algoritmo Flood-fill e parte do Lee's algorithm.

O Flood-fill é um algoritmo usado para determinar uma área limitada conectada a um determinado nó em uma matriz multidimensional, o seu objetivo é atingir todos os possíveis elementos, com uma regra pré-definida, que tenham o mesmo estado/valor. Para isso, ele tem a possibilidade de utilizar-se de várias estruturas de dados para chegar no mesmo fim, no caso deste trabalho prático optou-se pela utilização de uma fila para a sua implementação.

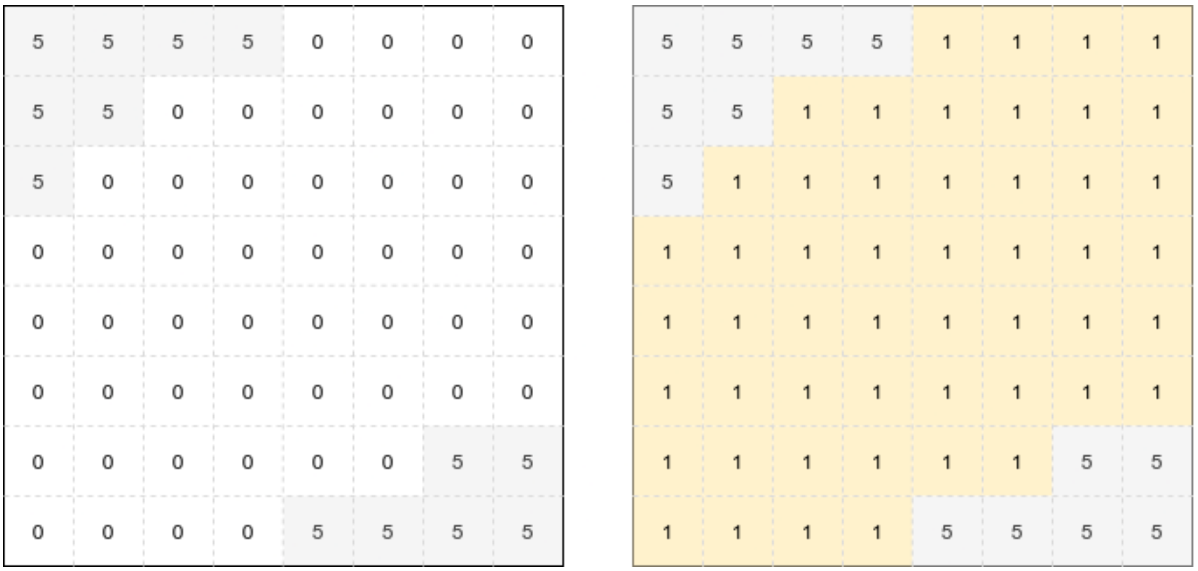

Figura 1: Representação do comportamento do Flood-fill

O Lee's algorithm é um algoritmo utilizado para descobrir o menor caminho em um labirinto, para isso ele aplica o método Flood-fill para preenchê-lo e quando chega na posição final, faz o caminho inverso, verificando o caminho através dos números na matriz.

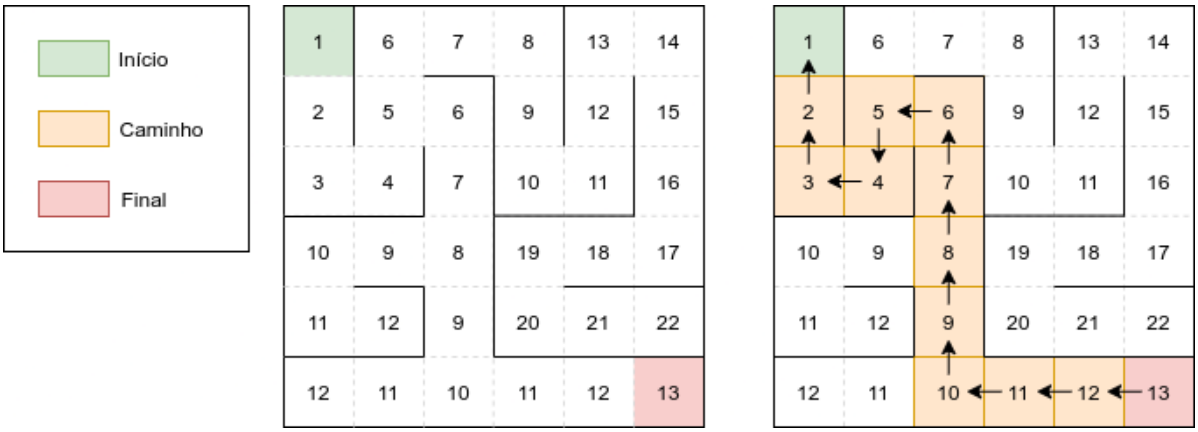

Figura 2: Representação do comportamento do Lee's algorithm

1.2 CONCEITOS DA DISCIPLINA ABORDADOS

No desenvolvimento da solução foram abordados os conceitos de Fila e de Lista Encadeada, utilizada para montar a Fila.

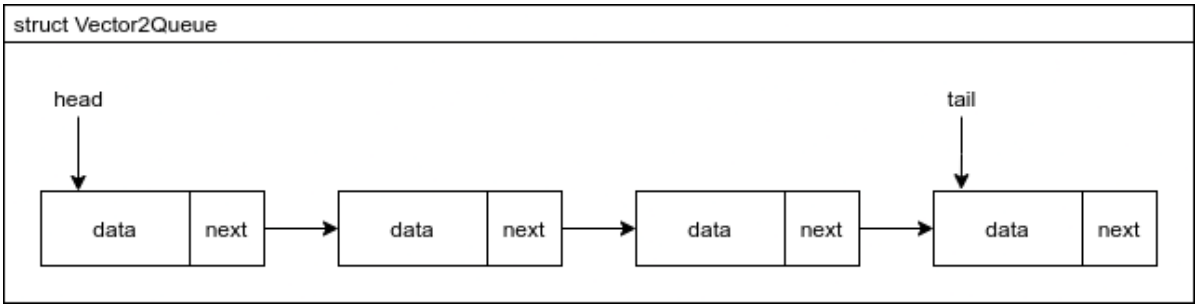

Figura 3: Representação da Fila utilizando Lista Encadeada

1.3 IMPLEMENTAÇÃO

A execução da função *flood_fill(Field *, Positions *)* começa em um determinado ponto de partida (x, y) e atribuí 1 a este ponto, após a atribuição, verifica as quatro posições vizinhas: Norte $(x, y-1)$, Sul $(x, y+1)$, Leste $(x+1, y)$ e Oeste $(x-1, y)$, preenchendo com o seu valor +1 caso essa posição vizinha já não estiver preenchida com outro valor $(= 0)$. Em seguida passa para algum outro ponto vizinho e repete até que todos os pontos da matriz estejam preenchidos. Nas seguintes imagens é possível ver como o algoritmo se comporta iniciando nas respectivas posições: $(1,1)$, $(3,4)$ e $(2,2)$.

Figura 4: Representação do comportamento do algoritmo Flood-Fill

Após o preenchimento da matriz, é chamado a função *set_path(Field *, Positions *)*, que primeiramente começa na posição final e verifica os vizinhos que estão com o valor - 1, caso tenha somente um vizinho que respeite as condições, a nova posição que ele irá considerar será ela mesma, porém, caso tenha mais de um, há uma verificação para ver qual é a posição mais próxima do ponto inicial, e assim, se repetindo até encontrar o ponto inicial.

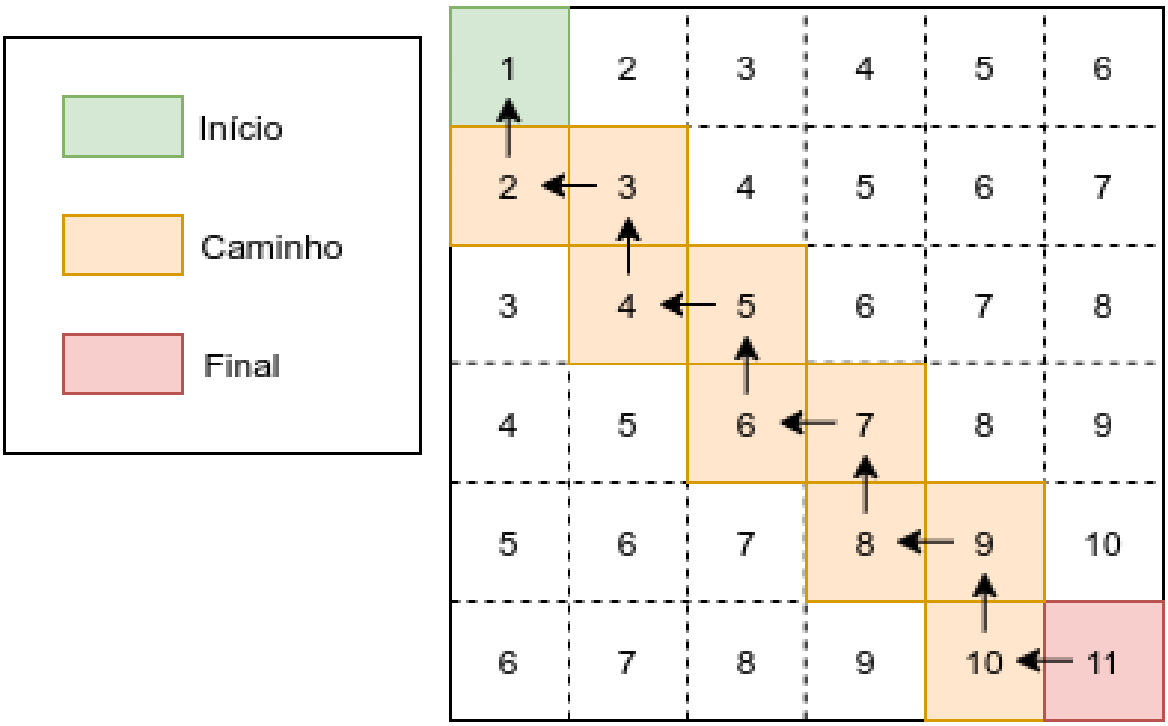

Figura 5: Representação do comportamento da função *set_path(Field *, Positions *)*

A execução das duas funções citadas acima ocorrem na primeira iteração e posteriormente quando ocorre alguma modificação na matriz, como: adicionar ou retirar uma parede, mudar a posição inicial e mudar a posição final.

A interação do usuário é toda feita através do teclado, utilizando as teclas A (mudar a posição inicial para a posição selecionada caso não seja a posição final ou não seja uma parede), S (mudar a posição final para a posição selecionada caso não seja a posição inicial ou não seja uma parede), D (adicionar ou remover uma parede na posição selecionada), ARROW_UP, ARROW_LEFT, ARROW_DOWN, ARROW_RIGHT (selecionar a próxima posição) e Q (sair).

1.4 TESTES

Para testar o funcionamento do programa, iniciou-se com posições padrões determinadas no código, como mostrado na Figura 6. Em seguida, alterou-se a posição inicial, identificada com um ”S” verde, e a posição final, identificada com um ”T” vermelho, Figura 7.

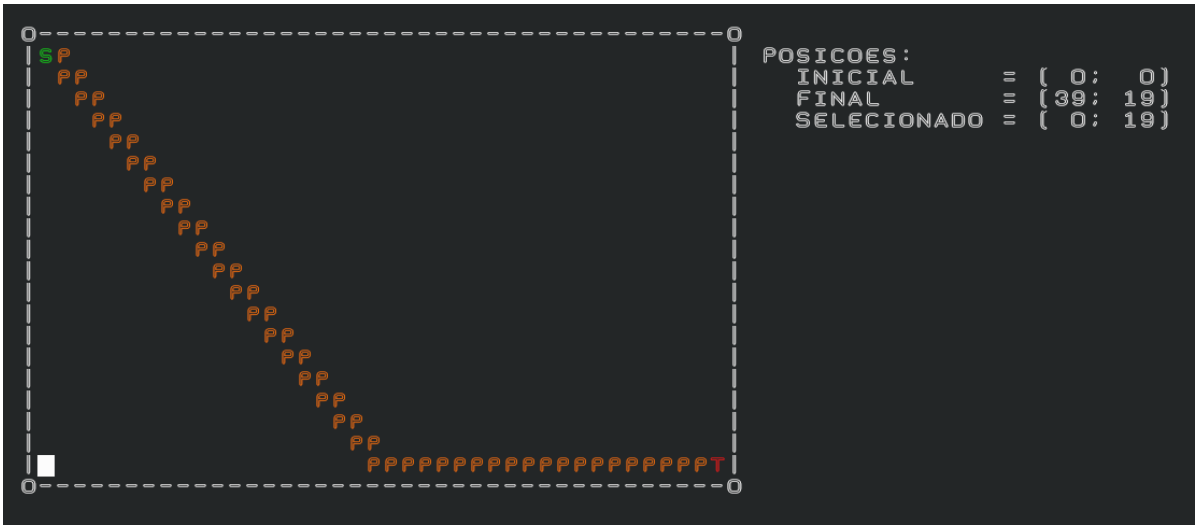

Figura 6: Demonstração do Funcionamento do Programa sem obstáculos

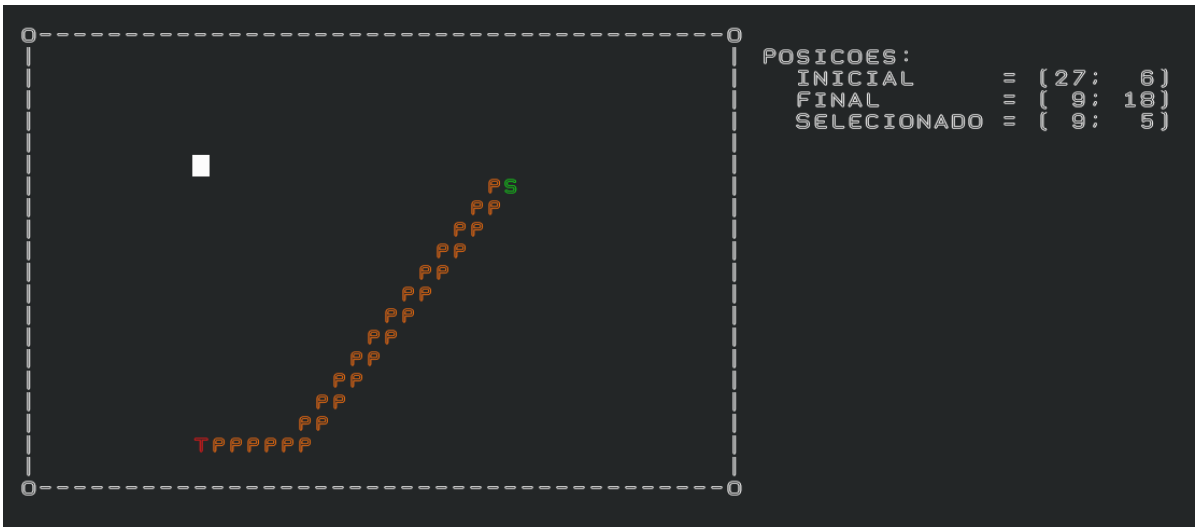

Figura 7: Demonstração do Funcionamento do Programa sem obstáculos e trocando as posições inicial e final

Para observar o comportamento do programa com obstáculos, inseriu-se paredes, identificadas com ”W” azul, como pode ser notado nas figuras 8 e 9, o caminho mostrado desvia de maneira eficaz dos obstáculos.

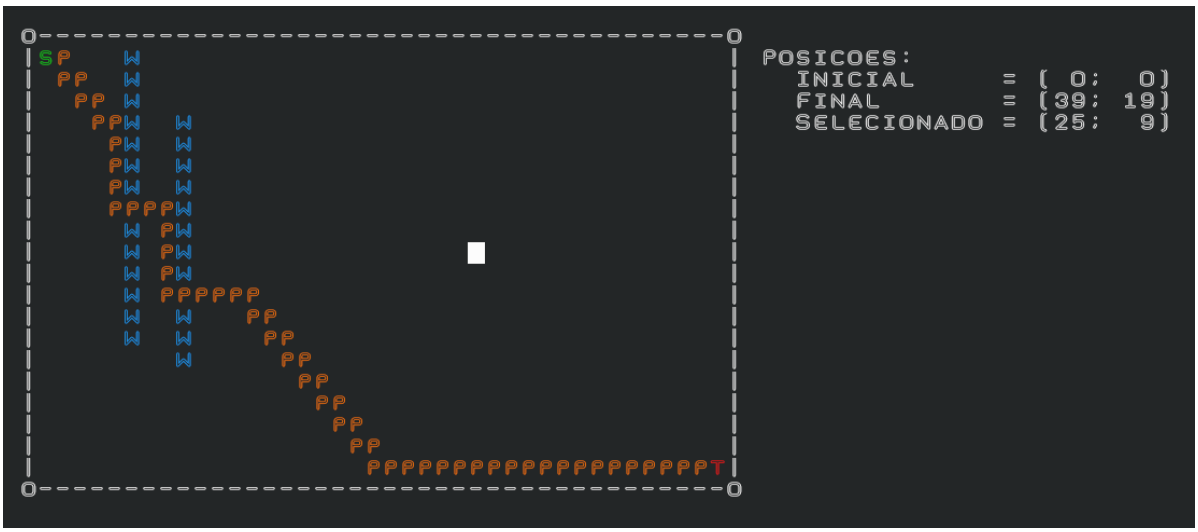

Figura 8: Demonstração do Funcionamento do Programa com obstáculos

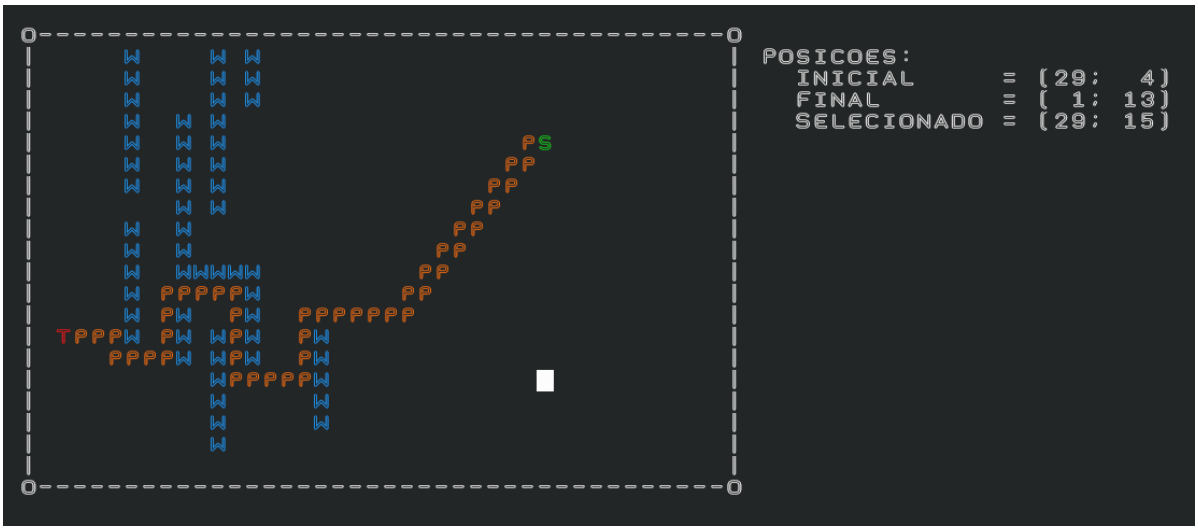

Figura 9: Demonstração do Funcionamento do Programa com mais obstáculos

Testou-se o caso de isolamento da posição inicial, Figura 10, e posição final, Figura 11. Como nenhum caminho foi encontrado, podemos afirmar que o programa não gera falsos caminhos para estes casos.

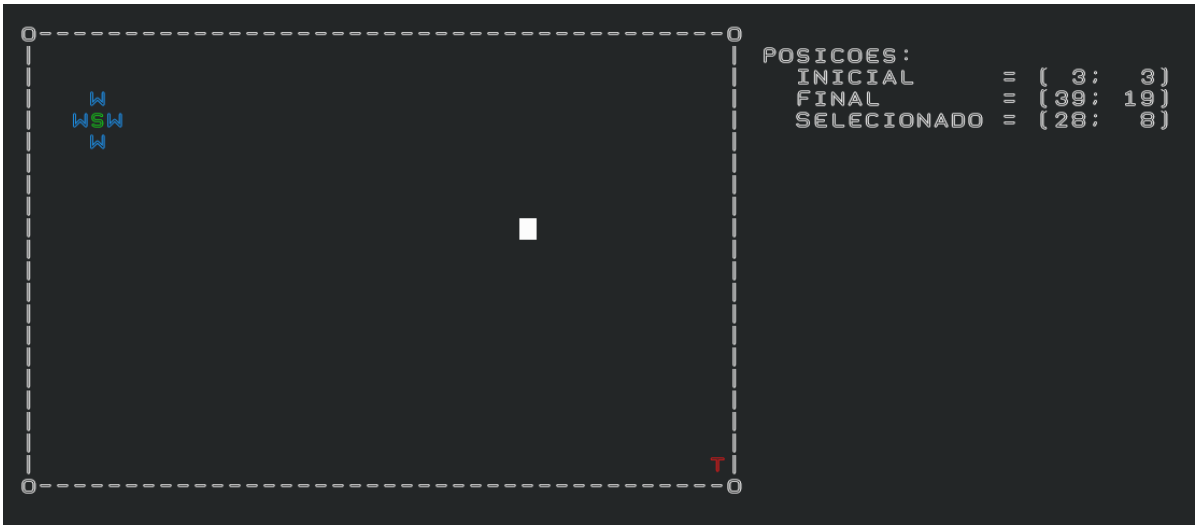

Figura 10: Demonstração do Funcionamento do Programa com obstáculos bloqueando a posição inicial

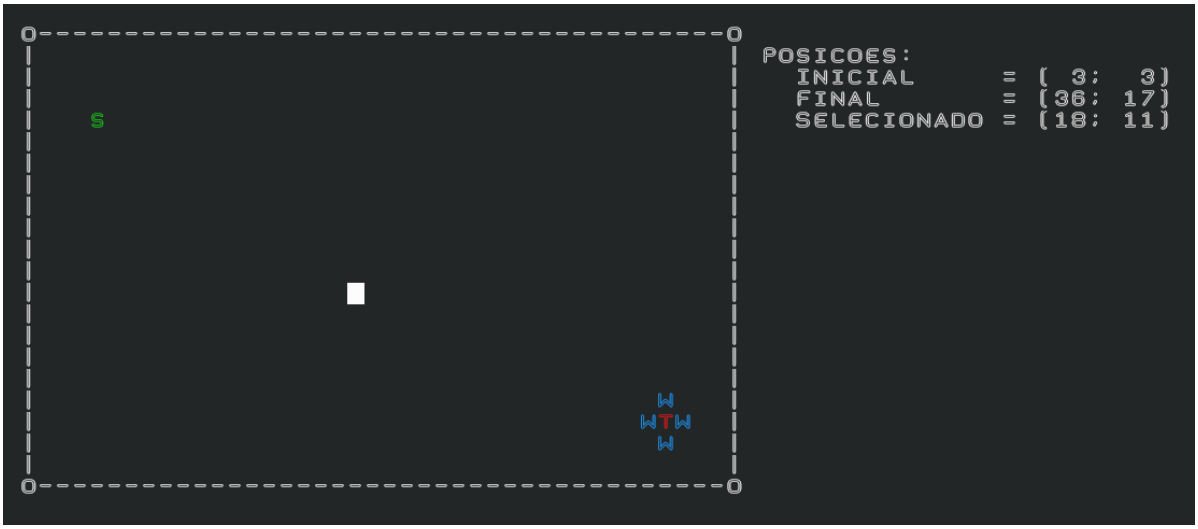

Figura 11: Demonstração do Funcionamento do Programa com obstáculos bloqueando a posição final

A fim de facilitar a visualização dos testes, adicionou-se um vídeo aos arquivos do projeto.