INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO ESPÍRITO SANTO
IFES - CAMPUS SERRA

INTELIGÊNCIA ARTIFICIAL

KELVIN LEHRBACK
20171BSI0448

A*(A-ESTRELA)

SERRA
2022

Sumário

1 - Explicação teórica do algoritmo

Dado um mapa, ponto inicial e ponto final, o algoritmo A* (A-estrela) busca o caminho de um grafo de um vértice inicial até um vértice final, somando custos e decidindo o melhor caminho (menor custo) até o vértice final.

1.1 - Os parâmetros F, G e H

o A* tem 3 parâmetros:

G é a distância entre o nó atual e o nó inicial., ou seja: o custo de mover o nó inicial para o nó atual. Basicamente, é a soma de todos os nós que foram visitados desde que saíram do primeiro nó..

H é a heurística - distância estimada entre o nó atual e o nó final. Para este algoritmo utilizaremos como heurística a distância de manhattan, cujo fórmula é:
$|x1 – x2| + |y1 – y2|$.

F é o custo total do nó (G + H).

2 - Problema proposto

Implementar o algoritmo A* para encontrar o melhor caminho dentro de um grid (arquivo .txt com 0 e 1, sendo 0 = caminho livre e 1 = caminho bloqueado). O agente poderá se mover em quatro direções em busca do melhor caminho (cima, baixo, direita e esquerda), ou seja, movimentos em 90 graus. A entrada consiste em: o caminho até o arquivo do grid, posição inicial e posição final (objetivo).

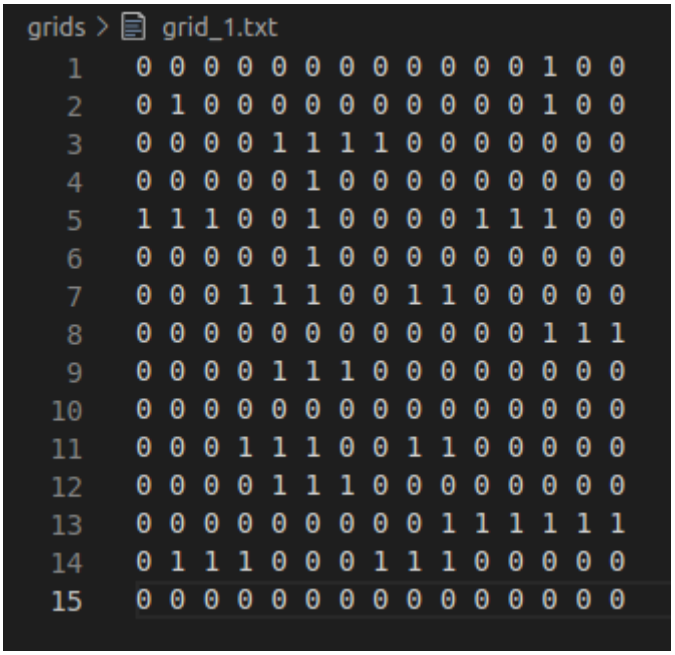
3 - Implementação

Linguagem utilizada: Python 3.
Sistema Operacional utilizado: Ubuntu 18.04
IDE: Vs Code.

O código está muito bem comentado, mas de qualquer forma, a seguir está a explicação dos principais trechos.

3.1 O mapa

O mapa (grid / labirinto) consiste em um arquivo txt contendo ‘0’ e ‘1’, sendo 0 caminho livre, ou seja, por onde o agente poderá se locomover e 1 caminho bloqueado, ou seja, não poderá utilizar essa coordenada para se locomover.



Para ler o arquivo, foi criada a função ***read_grid,*** que recebe como argumento o caminho do arquivo até o grid e retorna uma matriz de 0 e 1.

```
#Recebe o caminho do arquivo do grid
#Retorna uma matriz com base nos valores do grid
def read_grid(path_file):
      grid = []
      with open(path_file) as file:
            #Lendo todas as linhas do arquivo de uma vez
            lines = file.readlines()
            for line in lines:
                  #Removendo caracteres especiais(\n) e separando
por espaco em branco
                  line = line.strip().split(' ')
                  #Convertendo os valores da linha para inteiro
                  for i in range(len(line)):
                        line[i] = int(line[i])
                  grid.append(line)

      return grid
```

3.2 Heurística

Após ler o grid do arquivo e transformá-lo em uma matriz, é necessário calcular a heurística, ou seja, a distância de cada coordenada até o nó de destino. Para isso, foi utilizado a distância de Manhattan.

As funções responsáveis por retornar a matriz com a heurística são ***calc_heuristic***, que recebe como argumento o grid e o destino e retorna uma matriz com os valores de cada coordenada e ***get_distance*** que recebe como argumento as coordenadas que se deseja saber o valor e o destino, e devolve a distância calculada (valor de H).

```
#Baseado na distancia de mannhatan
#Formula: |x1 - x2| + |y1 - y2|
#Ou para N dimensoes: SUM(i=1~~N)|pi - qi|
def get_distance(p, q):
      distance = 0
    #Funcao ZIP retorna uma lista de tuplas
      for p_i, q_i in zip(p, q):
            distance += abs(p_i - q_i)

      return distance

#Recebe o grid o objetivo
#Calcula a distancia de todos os pontos ate o objetivo
#Retorna a matriz com as heuristicas
def calc_heuristic(grid, end):
      heuristic = []
      for i in range(len(grid)):
            j = 0
            new_line = []
            for j in range(len(grid[i])):
                  new_value = get_distance([i,j], end)
                  new_line.append(new_value)
            heuristic.append(new_line)

      return heuristic
```

A seguir, vamos ver como fica a matriz de heurística para o destino (13,12). Ou seja, o valor de cada coordenada em relação ao destino que será somado posteriormente com o valor do nó anterior em relação ao ponto inicial.

```
Caso 1: (0,0), 13,12)
[25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 14, 15]
[24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 13, 14]
[23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 12, 13]
[22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 11, 12]
[21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 10, 11]
[20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 9, 10]
[19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 8, 9]
[18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 7, 8]
[17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 6, 7]
[16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 5, 6]
[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 4, 5]
[14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 3, 4]
[13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 2, 3]
[12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 1, 2]
[13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 2, 3]
```

3.3 Nó

Para manipular as coordenadas do grid e salvar quem é o pai de cada Nó, foi criada uma classe ***Node,*** cuja inicialização é feita informando o nó pai e a coordenada do nó a ser criado. Na classe contém métodos auxiliares para comparação de classes, utilizando as funções auxiliares do python __eq__, __lt__ e __gt__, que verificam se um dado valor de uma classe é igual, menor ou maior (neste caso, a posição e o custo total do nó, respectivamente).

```python
class Node:
    #Recebe como argumento o no parente e a posicao
    #O no parente eh necessario para "voltar" ao inicio
    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position

        #Valores iniciais de G, H e F,
        #sendo:
            # f = Custo total do noh
            # g = Distancia do noh atual ate o noh inicial
            # h = distancia do noh atual ate o noh final
        self.g = 0
        self.h = 0
        self.f = 0

       #Os metodos dessa classe (Noh) foram criados conforme
```

```
tutorial:

#https://www.tutorialspoint.com/How-to-implement-Python-lt-gt-cust
om-overloaded-operators
    #Para fins de comparacao com o mesmo tipo de classe

    #Para verificar se os nohs tem a mesma posicao
    def __eq__(self, other):
        return self.position == other.position

    #Para verificar se os valores sao menores ou maiores, com base
no custo total de cada noh
    def __lt__(self, other):
      return self.f < other.f

    def __gt__(self, other):
      return self.f > other.
```

3.4 A*(A-Estrela)

Tendo o grid de caminho, a heurística, ponto inicial e destino, hora de executar o A*. Vale lembrar que usei a biblioteca de ***heapq,*** disponível no python, auxiliando na busca do nó de menor valor, criando uma pilha com cada nó (coordenada) verificado.
Eis o passo a passo do algoritmo.
1 - Adicione o nó(coordenada(inicial à lista aberta
2 - Repita:

A) Procure o nó de custo F mais baixo na lista aberta. Neste momento, este nó é o atual.

B) Mude para a lista fechada.

C) Para cada um dos quatro nós adjacentes(cima, baixo, direita e esquerda), faça:

a) Se não for possível ir para este nó ou se ele estiver na lista fechada, ignore-o. Caso contrário, faça:
b) Se não estiver na lista aberta, adicione-o à lista aberta. Faça o nó atual ser o pai deste quadrado e registre os custos de F, G e H do quadrado.
c) Se já estiver na lista aberta, verifique se este caminho para aquele nó é o melhor, usando G como custo de medida. Um

custo G menor significa que este é um caminho melhor. Nesse caso, altere o pai do nó para o nó atual e recalcule as pontuações de G e F do nó.

D) A condição de parada do algoritmo é:
   a) Quando é adicionado o nó alvo (destino) à lista fechada, caso o caminho for encontrado, ou
   b) Falha ao encontrar o nó de destino e a lista aberta está vazia. Neste caso, não há caminho.

3 - Salve o caminho. Vá regredindo os nós a partir do nó de destino(último encontrado), passando por cada nó pai e salvando a coordenada de cada um deles até o início. Este é o melhor caminho encontrado pelo algoritmo (se houver caminho).

A saída consiste nas coordenadas para realizar o melhor caminho ou 'None', caso o caminho não for encontrado / não seja possível.

```
#Para fins de prioridade do Noh, utilizei o heapq, cujo tutorial
vi em:
import                                                          heapq
##https://www.geeksforgeeks.org/heap-queue-or-heapq-in-python/

#importando a classe de No
from node import Node

#Retornar o caminho correto dado um noh
#Sempre verificando qual era o seu parente
def return_path(current_node):
    path = []
    current = current_node
    while current is not None:
        path.append(current.position)
        current = current.parent

    #Retorna o caminho reverso (sort.reverse)
    return path[::-1]

#Recebe como argumento:
    #Grid de caminho
    #Grid da heuristica de cada posicao no grid de caminhos
    #Posicao inicial e final
```

```
def a_estrela(maze, heuristic, start, end):
    #Cria o no inicial e final
    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0

    #Iniciando as listas abertas e fechadas
    open_list = []
    closed_list = []

    #Iniciando a pilha e colocando o no inicial
    heapq.heapify(open_list)
    heapq.heappush(open_list, start_node)

    # matriz que direciona os possiveis caminhos do ponto atual
    move = ((0, -1), #Esquerda
            (0, 1),  #Direita
            (-1, 0), #Cima
            (1, 0))  #Baixo

    #Enquanto tiver nos que nao foram verificados, procure!
    while len(open_list) > 0:
        #Captura o menor noh para fins de verificacao
        current_node = heapq.heappop(open_list)
        closed_list.append(current_node)

        #Verifica se chegou ao fim
        if current_node == end_node:
            return return_path(current_node)

          #Caso contrario, cria uma lista para armazenar as novas
posicoes
        children = []

          #Para cada posicao que o Noh pode se mover (Cima, baixo,
direita cima)
        for new_position in move:

              #Calcula qual sera a nova posicao do noh com base no
movimento
                    node_position  =  (current_node.position[0]  +
```

```
new_position[0], current_node.position[1] + new_position[1])

            #Verifica se esta dentro dos limites do grid
                    if  node_position[0]  >  (len(maze)  -  1)  or
node_position[0] < 0 or node_position[1] > (len(maze[len(maze)-1])
-1) or node_position[1] < 0:
                continue

            #Verifica se nao eh um obstaculo
            if maze[node_position[0]][node_position[1]] != 0:
                continue

            #Cria um novo noh, com base nessa nova posicao
            new_node = Node(current_node, node_position)

            #Salva essa posicao na lista de nos validos
            children.append(new_node)

        #Verifico a lista de nohs filhos
        for child in children:
            #Se o noh filho tiver caminho para ir
              if len([closed_child for closed_child in closed_list
if closed_child == child]) > 0:
                continue

            #Calculo os valores de 'g', 'h' e 'f'
            child.g = current_node.g + 1

            #Valor de 'h' com base na distancia de manhattan
                                                    child.h     =
heuristic[child.position[0]][child.position[1]]
            child.f = child.g + child.h

            #se o noh filho ja estiver na lista aberta,
                 if  len([open_node  for  open_node  in  open_list  if
child.position == open_node.position and child.g > open_node.g]) >
0:
                continue

            #Coloca essa lista de nohs no topo da fila
            heapq.heappush(open_list, child)
```

```
    #Nao encontrou nem um caminho
    return None
```

3.5 Saída do código

Além da lista de coordenadas, foi criada também uma função para gerar visualmente (no terminal / CMD) o caminho percorrido pelo agente, caso a função A* retorne um caminho. A função responsável por isso é a ***print_path,*** que recebe como argumento o grid, o caminho encontrado pelo algoritmo A*, posição inicial e posição final. Depois de popular a matriz "visual" com as coordenadas do ponto inicial, final, obstáculos, caminho livre e caminho feito pelo agente, é chamada a função ***print_grid***, que nada mais faz do que mostrar um print formatado de uma matriz.

@ = Nó inicial
x = Nó final
# = Caminho bloqueado
' ' = Caminho que o agente poderia realizar / explorar
* = Caminho realizado pelo agente / trajeto

```
#Mostra na tela de forma limpa uma matriz
def print_grid(grid):
     for line in grid:
          print(line)
     print("\n")


#Dado o grid, uma lista de coordenadas (caminho),
#posicao inicial e posicao final:
#Mostra na tela de forma "bonita" o caminho realizado ->
#Sendo  @ == posicao inicial
#       x == posinal final
#      ' '== caminho livre
#       # == obstaculo
#       * == trajeto
def print_path(grid, path, start, end):
    grid_path = []

    #Criando o grid_path vazio
    for i in range(len(grid)):
        line = []
        for j in range(len(grid[i])):
            if(grid[i][j] == 0):
```

```
                #Caminho livre
                line.append(' ')
            else:
                #Obstaculo
                line.append('#')
        grid_path.append(line)

    # '@' = Inicio
    # 'x' = Fim
    grid_path[start[0]][start[1]] = '@'
    grid_path[end[0]][end[1]] = 'x'

    #Apagando as primeiras e ultimas posicoes do path,
    #visto que ja foram computadas no grid_path
    del path[0]
    del path[len(path) -1]
    for coord in path:
        grid_path[coord[0]][coord[1]] = '*'


    print_grid(grid_path)
```

4 - Resultados

Após a execução, é informado na tela do terminal / cmd as coordenadas com o melhor caminho (ou None, caso não encontre) bem como o print do grid do trajeto feito pelo agente e o tempo de execução do algoritmo do A* para cada caso.

Lembrando que o primeiro print é o grid do arquivo .txt.

```
Labirinto:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1]
[0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

4.1 Primeiro Caso: (0,0) → (13, 12)

```
Caso 1: (0,0), 13,12)
Tempo total: 0.02723 segundos
Caminho encontrado:
[(0, 0), (1, 0), (2, 0), (2, 1), (3, 1), (3, 2), (3, 3), (4, 3), (5, 3), (5, 2), (6, 2), (7, 2), (8, 2), (9, 2), (10, 2), (11, 2), (11, 3),
 (12, 3), (12, 4), (13, 4), (14, 4), (14, 5), (14, 6), (14, 7), (14, 8), (14, 9), (14, 10), (14, 11), (14, 12), (13, 12)]


Caminho desenhado:
['@', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ']
['*', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ']
['*', '*', ' ', ' ', '#', '#', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', '*', '*', '*', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
['#', '#', '#', '*', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', '#', '#', ' ', ' ', ' ']
[' ', ' ', ' ', '*', '*', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', '#', '#', '#', ' ', ' ', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '#', '#', '#']
[' ', ' ', ' ', '*', ' ', '#', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', '#', '#', '#', ' ', ' ', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', '*', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', '*', '*', ' ', ' ', ' ', '#', '#', '#', '#', '#', '#', '#', '#']
[' ', '#', '#', '#', '*', ' ', ' ', ' ', '#', '#', '#', ' ', ' ', 'x', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', ' ', ' ', ' ']
```

4.1 Segundo Caso (volta): (13, 12) → (0, 0)

```
Caso 2: (13,12), (0,0)
Tempo total: 0.00355 segundos
Caminho encontrado:
[(13, 12), (13, 11), (14, 11), (14, 10), (14, 9), (14, 8), (14, 7), (14, 6), (14, 5), (13, 5), (12, 5), (12, 4), (12, 3), (12, 2), (11, 2),
 (10, 2), (9, 2), (8, 2), (7, 2), (6, 2), (5, 2), (5, 3), (4, 3), (3, 3), (2, 3), (2, 2), (1, 2), (0, 2), (0, 1), (0, 0)]


Caminho desenhado:
['x', '*', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ']
[' ', '#', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ']
[' ', ' ', '*', '*', '#', '#', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
['#', '#', '#', '*', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', '#', '#', ' ', ' ', ' ']
[' ', ' ', ' ', '*', '*', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', '#', '#', '#', ' ', ' ', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '#', '#', '#']
[' ', ' ', ' ', '*', ' ', '#', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', '#', '#', '#', ' ', ' ', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', ' ', '#', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', '*', '*', '*', ' ', ' ', ' ', '#', '#', '#', '#', '#', '#', '#']
[' ', '#', '#', '#', ' ', '*', ' ', '#', '#', '#', ' ', '*', '@', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', '*', '*', '*', '*', '*', '*', '*', ' ', ' ', ' ', ' ']
```

4.3 Terceiro Caso (Definido pelo usuário):

Neste caso, escolhi a coordenada inicial (14,0) e a coordenada final (0,14).

```
Input do Usuario:  (14, 0) (0, 14)
Tempo total: 0.00727 segundos
Caminho encontrado:
[(14, 0), (13, 0), (12, 0), (12, 1), (12, 2), (11, 2), (10, 2), (9, 2), (9, 3), (8, 3), (7, 3), (7, 4), (7, 5), (7, 6), (6,
6), (5, 6), (5, 7), (4, 7), (3, 7), (3, 8), (3, 9), (3, 10), (3, 11), (3, 12), (2, 12), (2, 13), (2, 14), (1, 14), (0, 14)]


Caminho desenhado:
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', 'x']
[' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', '*']
[' ', ' ', ' ', ' ', ' ', '#', '#', '#', '#', ' ', ' ', ' ', ' ', '*', '*', '*', '*']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', '*', '*', '*', '*', '*', ' ', ' ']
['#', '#', '#', ' ', ' ', ' ', '#', ' ', ' ', '*', ' ', ' ', ' ', '#', '#', '#', ' ']
[' ', ' ', ' ', ' ', ' ', ' ', '#', '*', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', '#', '#', '#', '*', ' ', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', ' ', '*', '*', '*', '*', ' ', ' ', ' ', ' ', ' ', ' ', '#', '#', '#']
[' ', ' ', ' ', ' ', '*', '#', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', '#', '#', '#', ' ', ' ', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ']
[' ', ' ', ' ', '*', ' ', '#', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
['*', '*', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '#', '#', '#', '#', '#', '#']
['*', '#', '#', '#', ' ', ' ', ' ', ' ', '#', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ']
['@', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
```

5 - Referências

- Artigos disponibilizados pelo professor no AVA.
- Agradecimentos especiais à Betina Carol Zanchin, autora do artigo "*ANÁLISE DO ALGORITMO A* (A ESTRELA) NO PLANEJAMENTO DE ROTAS DE VEÍCULOS AUTÔNOMOS*" e pessoa que me tirou algumas dúvidas do código.
- Explicação de heapq: *https://www.geeksforgeeks.org/heap-queue-or-heapq-in-python/*
- Explicação dos métodos de classe: https://www.tutorialspoint.com/How-to-implement-Python-lt-gt-custom-overloaded-operators
- Explicação em vídeo do A*: https://youtu.be/o5_mqZKhTvw