TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

❖❖❖❖❖❖❖❖❖❖❖❖

BÁO CÁO

ĐỒ ÁN 1

Lớp: Cơ sở trí tuệ nhân tạo 19_21

DANH SÁCH THÀNH VIÊN

Họ và tên	**MSSV**
Nguyễn Minh Long	19120568
Phạm Văn Nam	19120597

NỘI DUNG BÁO CÁO

I. MÃ NGUỒN

1. Thư viện

```
import os
import matplotlib.pyplot as plt
```

- Sử dụng các thư viện này nhằm hỗ trợ việc visualize mê cung

2. Các hàm hỗ trợ nhập xuất thông tin

a. read_file (tái sử dụng hàm tham khảo của file hướng dẫn)

```
def read_file(file_name):
      . . .
   return bonus_points, matrix
```

- Input:

- file_name (string): tên file input

- Output:

- bonus_points (list): danh sách các điểm thưởng (trả về list rỗng nếu n_bonus_points = 0)
- matrix (list 2 chiều): ma trận 2 chiều chứa các kí tự:
  - ‘x’: tường
  - ‘ ’: đường đi
  - ‘S’: điểm bắt đầu

b. visualize_maze (tái sử dụng hàm tham khảo của file hướng dẫn, có bổ sung)

```
def visualize_maze(matrix, bonus, start, end, route=None, trav=None):
    . . .
    #2. Drawing the map
    add_row = 0
    ax=plt.figure(dpi=100).add_subplot(111)

    plt.text(0,-1,'Bonus_point: ' +
         str(len(bonus)),fontweight='bold',color='red',
         horizontalalignment='left',
         verticalalignment='center')
    add_row += 1
    if route:
      plt.text(0,-2,
     'Cost: ' + str(len(route)-1) + '   Traversed: ' + str(len(trav)-1),
          fontweight='bold',color='red',
```

```
        horizontalalignment='left',
        verticalalignment='center')
    add_row += 1

for i in ['top','bottom','right','left']:
    ax.spines[i].set_visible(False)

plt.scatter([i[1] for i in walls],[-i[0] - add_row for i in walls],
            marker='X',s=100,color='black')

plt.scatter([i[1] for i in bonus],[-i[0] - add_row for i in bonus],
            marker='P',s=100,color='green')

plt.scatter(start[1],-start[0] - add_row,marker='*',
            s=100,color='gold')

if route:
    for i in range(len(route)-2):
        plt.scatter(route[i+1][1],-route[i+1][0] - add_row,
                    marker=direction[i],color='silver')

plt.text(end[1],-end[0] - add_row,'EXIT',color='red',
     horizontalalignment='center',
     verticalalignment='center')
. . .
```

- Input:

- matrix (list 2 chiều): ma trận đọc từ file input
- bonus (list): danh sách các điểm thưởng
- start, end (tuple dạng '(x,y)'): toạ độ các điểm bắt đầu và kết thúc của mê cung
- route (list các tuple): danh sách đường đi (nếu có)
- trav (list các tuple): danh sách các điểm đã duyệt

- Output:

- Tạo cửa sổ Figure từ mathplotlib bao gồm:
  - Mê cung
  - Các điểm bắt đầu, kết thúc và điểm thưởng (nếu có)
  - Các thông tin: **Bonus_point** (số điểm thưởng), **Cost** (chi phí đường đi) và **Traversed** (số lượng điểm đã duyệt)

- Xuất ra python terminal thông tin các điểm bắt đầu, kết thúc và điểm thưởng

c. find_start_end (tái sử dụng hàm tham khảo của file hướng dẫn)

```
def find_start_end(matrix):
      . . .
    return start, end
```

- Input:

- matrix (list 2 chiều): ma trận đọc từ file input

- Output:

- start, end (tuple): toạ độ điểm bắt đầu, kết thúc

3. Hàm xây dựng cấu trúc mê cung và các hàm hỗ trợ gắn/lấy thông tin từ mê cung

a. maze_trans

```
def maze_trans(matrix):
  maze = []
  for i in range(len(matrix)):
    temp = []
    for j in range(len(matrix[0])):
      state = matrix[i][j]
      path_cost = 0
      heu = 0
      prev = None
      succs = [(i-1,j),(i,j-1),(i+1,j),(i,j+1)]
      a={'state':state,'path_cost':path_cost,'heu':heu,'prev':prev,'succs':succs}
      temp.append(a)
    maze.append(temp)
  return maze
```

- Input:

- matrix (list 2 chiều): ma trận đọc từ file input

- Output:

- maze (list 2 chiều): với mỗi phần tử maze[i][j] là một dict dạng {‘state’: , ‘path_cost’: , ‘heu’: , ‘prev’: , ‘succs’: }. Trong đó:
  - state (string): ứng với ký hiệu tại vị trí [i][j] trên matrix – ‘x’, ‘ ’, ‘S’

- path_cost (int/float): chi phí đường đi khi điểm [i][j] được duyệt bởi previous của nó
- heu (int/float): ước tính khoảng cách tới điểm kết thúc
- prev (tuple): toạ độ của điểm previous duyệt đến nó
- succs là tập chứa các điểm lân cận theo thứ tự **'trên', 'trái', 'dưới', 'phải'**

- Mô tả: duyệt đến từng vị trí [i][j] trong matrix, tạo một dict chứa các dữ liệu theo cấu trúc nhóm đặt ra sau đó đẩy vào list là dòng tương ứng và sau đó đẩy vào maze

b. Nhóm các hàm get thông tin từ maze

```
def get_state(maze,A):
  return maze[A[0]][A[1]]['state']

def get_succs(maze,A):
  return maze[A[0]][A[1]]['succs']

def get_path_cost(maze,A):
  return maze[A[0]][A[1]]['path_cost']

def get_prev(maze,A):
  return maze[A[0]][A[1]]['prev']

def get_heu(maze,A):
  return maze[A[0]][A[1]]['heu']
```

- Input:

- maze (list 2 chiều): cấu trúc mê cung
- A (tuple): toạ độ điểm

- Output:

- Trả về các thuộc tính tương ứng trong dict

c. Nhóm các hàm set thông tin vào maze

i. set_path_cost

```
def set_path_cost(maze,A,cost):
  prev = maze[A[0]][A[1]]['prev']
  if prev != None:
    maze[A[0]][A[1]]['path_cost'] += get_path_cost(maze,prev) + cost
```

- Input:

- maze (list 2 chiều): cấu trúc mê cung

- A (tuple): toạ độ điểm
- cost (int/float): chi phí kề của A và điểm prev (previous) của A

- Output: None

- Mô tả: kiểm tra vị trí nếu 'prev' của điểm A khác None thì đặt path_cost của A = path_cost của prev của A + chi phí kề

4. Các hàm heuristic và hàm gắn heuristic vào mê cung

a. Heuristic Euclid

```
def heuristic_euclid(A,end):
  h = ((A[0]-end[0])**2 + (A[1]-end[1])**2)**0.5
  return round(h,3)
```

- Input:

- A (tuple): toạ độ điểm
- end (tuple): toạ độ điểm kết thúc của mê cung

- Output:

- giá trị heuristic làm tròn đến chữ số thập phân thứ 3

- Mô tả: tính heuristic dựa trên khoảng cách euclid giữa 2 điểm A – B: $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$

b. Heuristic 2 Times Euclid

```
def heuristic_2_times_euclid(A,end):
  h = (A[0]-end[0])**2 + (A[1]-end[1])**2
  return h
```

- Input:

- A (tuple): toạ độ điểm
- end (tuple): toạ độ điểm kết thúc của mê cung

- Output:

- giá trị heuristic

- Mô tả: tính heuristic dựa trên khoảng cách Euclid^2 giữa 2 điểm A – B: $(x_A - x_B)^2 + (y_A - y_B)^2$

c. Heuristic Manhattan

```
def heruristic_manhattan(A,end):
  return abs(A[0]-end[0]) + abs(A[1]-end[1])
```

- Input:

- A (tuple): chứa toạ độ điểm
- end (tuple): chứa toạ độ điểm kết thúc của mê cung

- Output:

- giá trị heuristic

- Mô tả: tính heuristic dựa trên tổng số bước đi theo 2 cạnh hình chữ nhật giữa 2 điểm A – B: $|x_A - x_B| + |y_A - y_B|$

d. set_heuristic

```
def set_heuristic(maze,heuristic,end):
  for i in range(len(maze)):
    for j in range(len(maze[0])):
      maze[i][j]['heu'] = heuristic((i,j),end)
```

- Input:

- maze (list 2 chiều): cấu trúc mê cung
- heuristic: con trỏ trỏ tới hàm tương ứng gắn giá trị heuristic cho điểm A đến điểm Exit
- end (tuple): toạ độ điểm kết thúc của mê cung

- Output: None

- Mô tả: duyệt đến từng vị trí [i][j] trong mê cung và gắn heuristic bằng hàm heuristic tương ứng

5. Các thuật toán

a. Breadth First Search

```
def BFS(maze,start,end):
  V = [[start]]
  visited = [start]
  k = 0
  while end not in V[k] and len(V[k]) != 0:
    V_next = []
    for s1 in V[k]:
      if get_state(maze,s1) != 'x':
        for s2 in get_succs(maze, s1):
          if get_state(maze,s2) != 'x':
            if s2 not in visited:
              visited.append(s2)
              set_prev(maze,s2,s1)
              V_next.append(s2)
    V.append(V_next)
    k += 1
```

```
  if len(V[k]) == 0:
    return None, None
  else:
    route = [end]
    s = end
    for i in range(k):
      route.append(get_prev(maze,s))
      s = get_prev(maze,s)
    route.reverse()
  return route, visited
```

- Input:
- maze(list 2 chiều): cấu trúc mê cung
- start : tọa độ điểm của điểm bắt đầu trong mê cung
- end: tọa độ điểm kết thúc trong mê cung
- Output:
- route: danh sách thứ di chuyển từ điểm bắt đầu đến điểm kết thúc mê cung
- visited: danh sách các điểm đã duyệt trong quá trình tìm dường đi
- Mô tả thuật toán:
- V là tập điểm khởi đầu
- Visited là tập các điểm đã mở
- Khi điểm kết thúc vẫn chưa nằm trong V thì: Xét điểm đầu tiên trong V, thêm các điểm có thể đến xung quang điểm đang xét (theo thứ tự **'trên', 'trái', 'dưới', 'phải'**), nếu điểm đó chưa trong visited thì thêm điểm đó vào visited và V_next và thêm tính chất quay lui của điểm đó với điểm quay lui là điểm đang xét trong V. Thêm các điểm trong V_next vào V. Xét điểm tiếp theo trong V.
- Nếu không tìm được end hoặc tập V_next rỗng thì sẽ return None.
- Nếu tìm được lối ra thì truy từ end ngược về start theo tính chất quay lui đã thêm vào điểm ở trên rồi nghịch đảo các điểm sau đó return danh sách các điểm từ start đến end

b. Depth First Search:

```
def DFS_Al(maze,start,end,visited):
  m = len(maze)
  n = len(maze[0])
  visited.append(start)
  check_Around = 0
```

```
  check = True
  if(start != end):
    for s1 in get_succs(maze,start):
      if get_state(maze,s1) == 'x' or s1 in visited:
        check_Around += 1
      else:
        set_prev(maze,s1,start)
        temp,trav = DFS_Al(maze,s1,end,visited)
        if(temp == -1):
          check_Around += 1
  if(check_Around == 4):
    check = False
  if(check == True):
    return 0, visited
  else:
    return -1, visited
def DFS(maze,start,end):
  visited = []
  check,visited = DFS_Al(maze,start,end,visited)
  if check != -1:
    route = [end]
    s = end
    while s != None:
      route.append(get_prev(maze,s))
      s = get_prev(maze,s)
    route.pop(-1)
    route.reverse()
    return route,visited
  else: return None,visited
```

- Input:
- maze(list 2 chiều): cấu trúc mê cung
- start : tọa độ điểm của điểm bắt đầu trong mê cung
- end: tọa độ điểm kế thúc trong mê cung
- Output:
- route: danh sách thứ di chuyển từ điểm bắt đầu đến điểm kết thúc mê cung
- visited: danh sách các điểm đã duyệt trong quá trình tìm dường đi
- Mô tả thuật toán:
- Xét từ điểm khởi đầu, xét các điểm xung quanh có thể đến được từ điểm đang xét (theo thứ tự **'trên', 'trái', 'dưới', 'phải'**), nếu đi được điểm đó thì ta lại bắt đầu xét từ điểm đó.Xét cho đến khi tìm

được điểm end. Nếu xung quanh điểm xét không đi được điểm nào nữa thì lùi lại điểm xét trước đó.

- Nếu đã duyệt tất cả điểm mà không còn đường đi thì trả về -1 đến khi trả về điểm xuất phát thì trả về route = None

c. Greedy Best First Search:

```
def GBFS(maze,start,end):
  closed = []
  fringe = [start]
  while len(fringe) > 0:
    temp = [get_heu(maze,i) for i in fringe]
    index = temp.index(min(temp))
    if get_state(maze,fringe[index]) == 'x':
      fringe.remove(fringe[index])
      continue
    next = fringe.pop(index)
    if next == end:
      closed.append(next)
      break
    if next not in closed:
      closed.append(next)
    for i in get_succs(maze,next):
      if i not in fringe and i not in closed and get_state(maze,i) != 'x':
        fringe.append(i)
        set_prev(maze,i,next)
  if len(fringe) == 0 and end not in closed:
    return None, None
  else:
    route = [end]
    s = end
    while s != None:
      route.append(get_prev(maze,s))
      s = get_prev(maze,s)
    route.pop(-1)
    route.reverse()
  return route, closed
```

- Input:
- maze(list 2 chiều): cấu trúc mê cung
- start : tọa độ điểm của điểm bắt đầu trong mê cung
- end: tọa độ điểm kế thúc trong mê cung

- Output:

- route: danh sách thứ di chuyển từ điểm bắt đầu đến điểm kết thúc mê cung
- visited: danh sách các điểm đã duyệt trong quá trình tìm dường đi

- Mô tả thuật toán:

- Khởi tạo closed (list chứa tuple các điểm đã duyệt) rỗng và fringe (list chứa tuple các điểm biên) có chứa toạ độ điểm bắt đầu
- Xét các điểm trong fringe và pop khỏi fringe điểm có giá trị heuristic nhỏ nhất. Nếu điểm đó chưa có trong closed thì thêm vào closed và tìm những điểm xung quanh điểm đó, gắn toạ độ quay lui và thêm vào fringe, ngược lại bỏ qua và xét tiếp trong fringe. Xét đến khi điểm tiếp theo là điểm end.
- Nếu chưa tìm được end nhưng fringe rỗng thì route = None

d. A_Star:

```
def A_star(maze,start,end):
closed = []
  fringe = [start]
  while len(fringe) > 0:
    temp = [get_path_cost(maze,i) + get_heu(maze,i) for i in fringe]
    index = temp.index(min(temp))
    if get_state(maze,fringe[index]) == 'x':
      fringe.remove(fringe[index])
      continue
    next = fringe.pop(index)
    if next == end:
      closed.append(next)
      break
    if next not in closed:
      closed.append(next)
    for i in get_succs(maze,next):
      if i not in fringe and i not in closed and get_state(maze,i) != 'x':
        fringe.append(i)
        set_prev(maze,i,next)
        set_path_cost(maze,i,1)
  if len(fringe) == 0 and end not in closed:
    return None, None
  else:
    route = [end]
    s = end
    while s != None:
      route.append(get_prev(maze,s))
```

```
      s = get_prev(maze,s)
    route.pop(-1)
    route.reverse()
  return route, closed
```

- Input:
- maze(list 2 chiều): cấu trúc mê cung
- start : tọa độ điểm của điểm bắt đầu trong mê cung
- end: tọa độ điểm kế thúc trong mê cung
- Output:
- Danh sách các điểm cần đi để thoát khỏi mê cung.
- Mô tả thuật toán:
- Khởi tạo closed (list chứa tuple các điểm đã duyệt) rỗng và fringe (list chứa tuple các điểm biên) có chứa toạ độ điểm bắt đầu
- Xét các điểm trong fringe và pop khỏi fringe điểm có giá trị ước tính đường đi (path_cost + heuristic) là nhỏ nhất. Nếu điểm đó chưa có trong closed thì thêm vào closed và tìm những điểm xung quanh điểm đó, gắn toạ độ quay lui, đặt giá trị đường đi và thêm vào fringe, ngược lại bỏ qua và xét tiếp trong fringe. Xét đến khi điểm tiếp theo là điểm end.
- Nếu chưa tìm được end nhưng fringe rỗng thì route = None

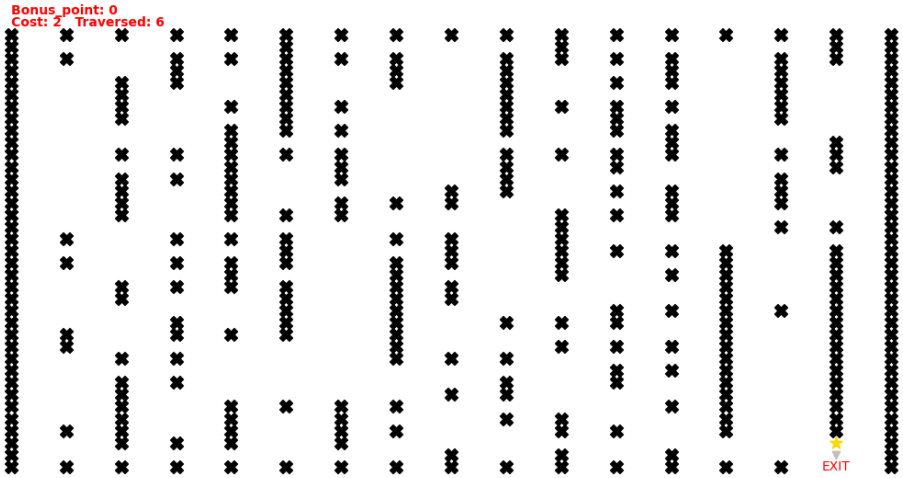
II. THỰC HIỆN

- Cách thức thực hiện: với mỗi thuật toán và mỗi bản độ, thực hiện 2 lần chạy với mỗi lần là một điểm Start khác nhau nhưng cùng điểm Exit. Đối với thuật toán tìm kiếm có thông tin thì thực hiện riêng biệt 3 heuristic => Mỗi bản đồ sẽ thực hiện 16 lần:

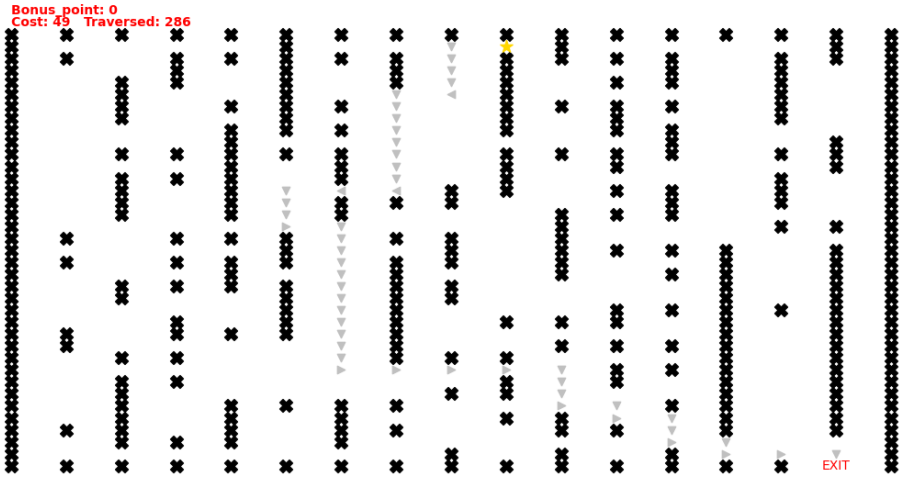
- DFS: 2 lần
- BFS: 2 lần
- Greedy BFS: 6 lần (2 lần mỗi heuristic)
- A*: 6 lần (2 lần mỗi heuristic

1. Map 1

a. BFS:

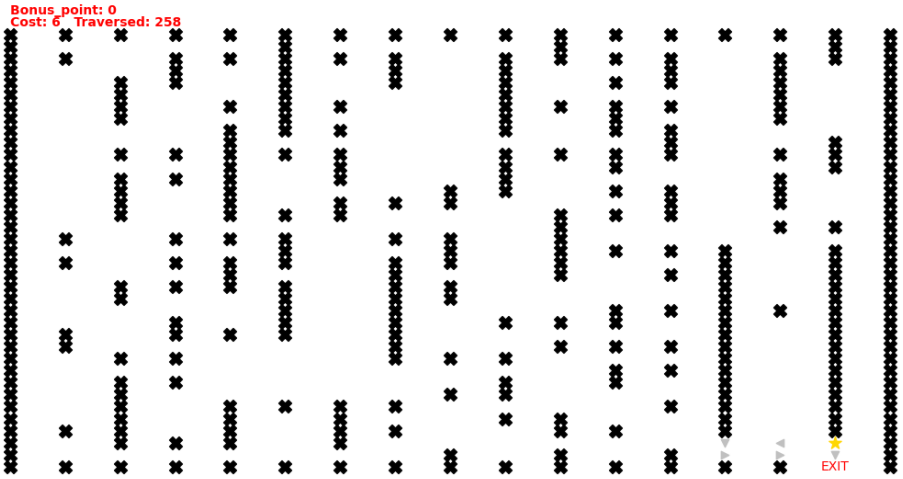


Start(15,35) - End(15,36)

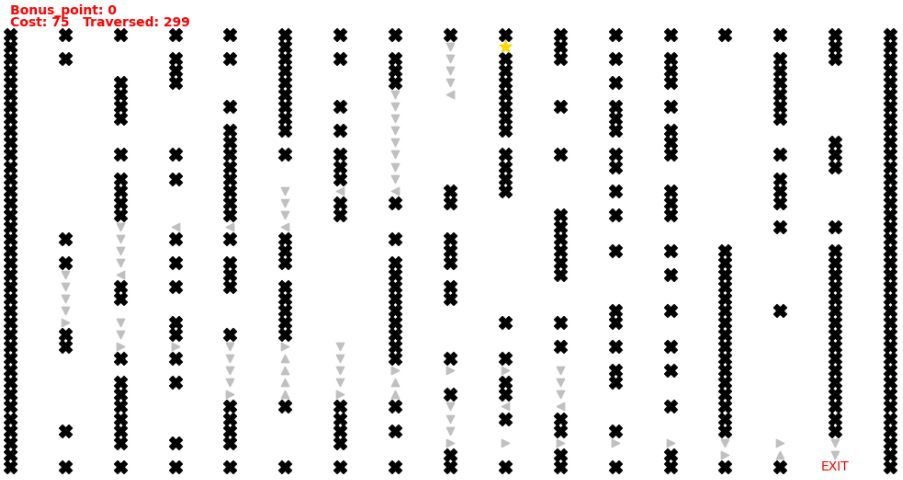


Start(1,9) - End(15,36)

b. DFS:



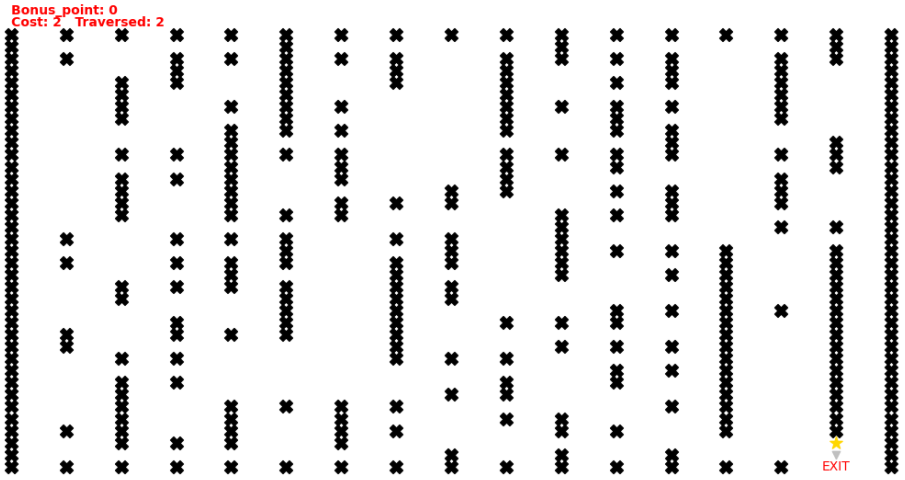
Start(15,35) - End(15,36)



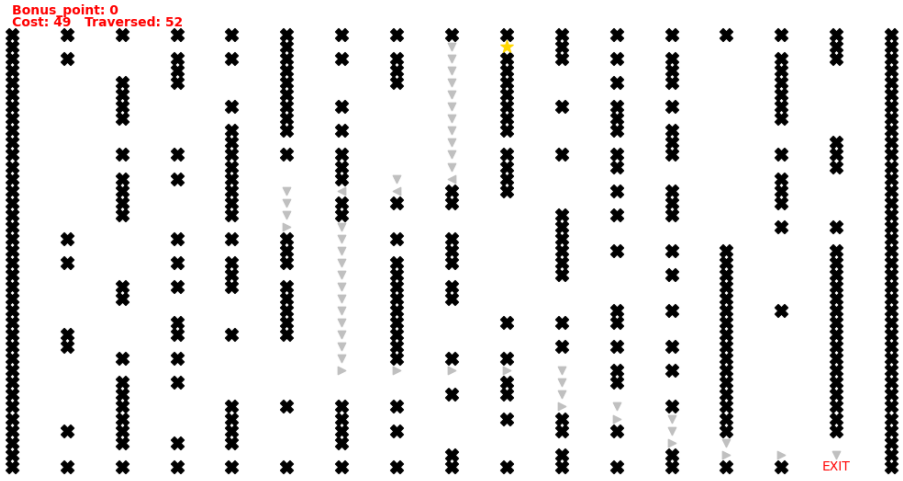
Start(1,9) - End(15,36)

c. GBFS:

- Heuristic Euclid:

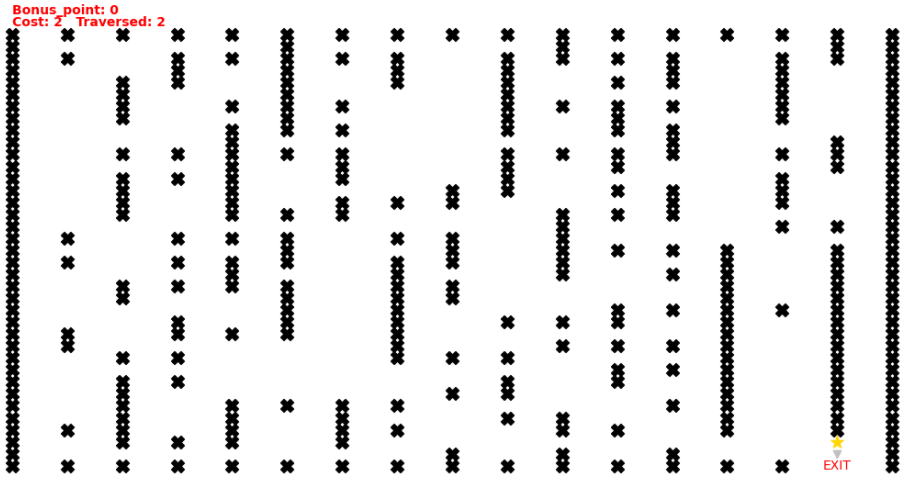


Start(15,35) - End(15,36)

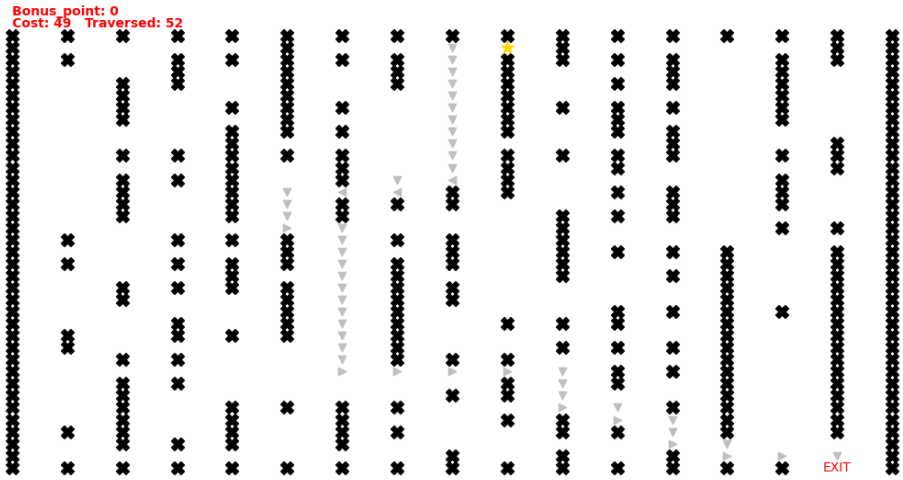


Start(1,9) - End(15,36)

- Heuristic 2 Times Euclid:

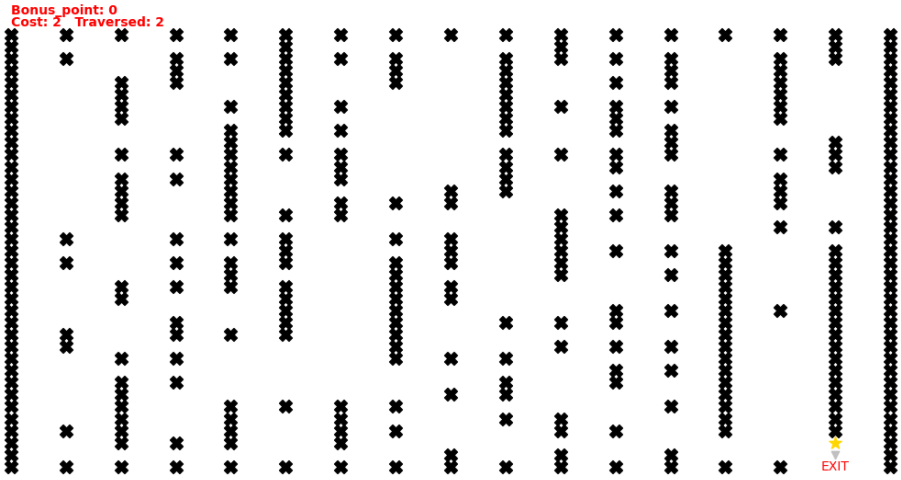


Start(15,35) - End(15,36)

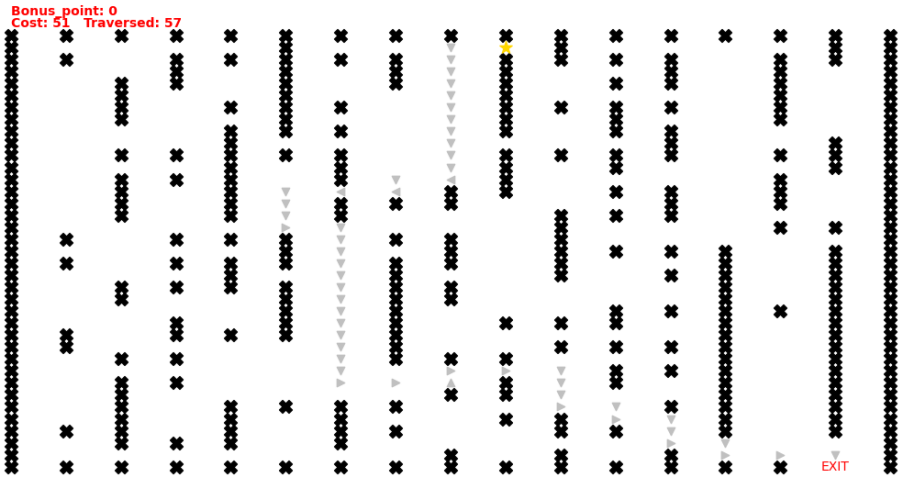


Start(1,9) - End(15,36

- Heuristic Manhattan:



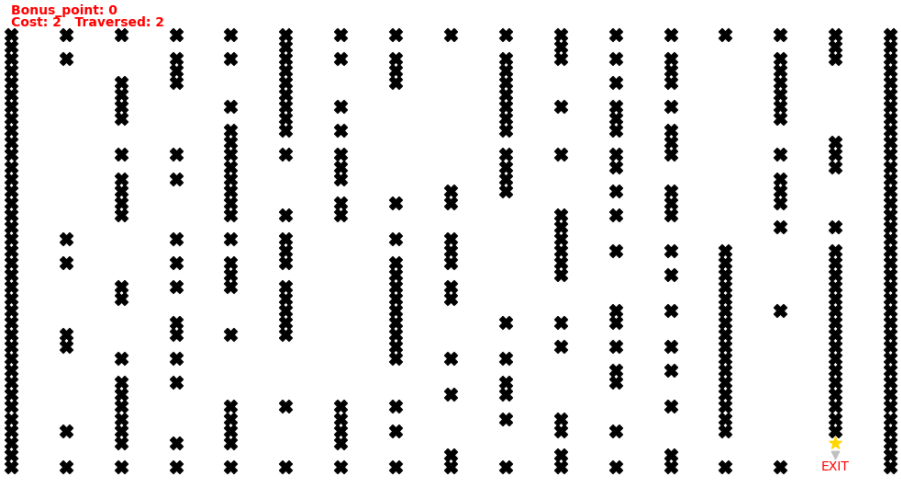
Start(15,35) - End(15,36)



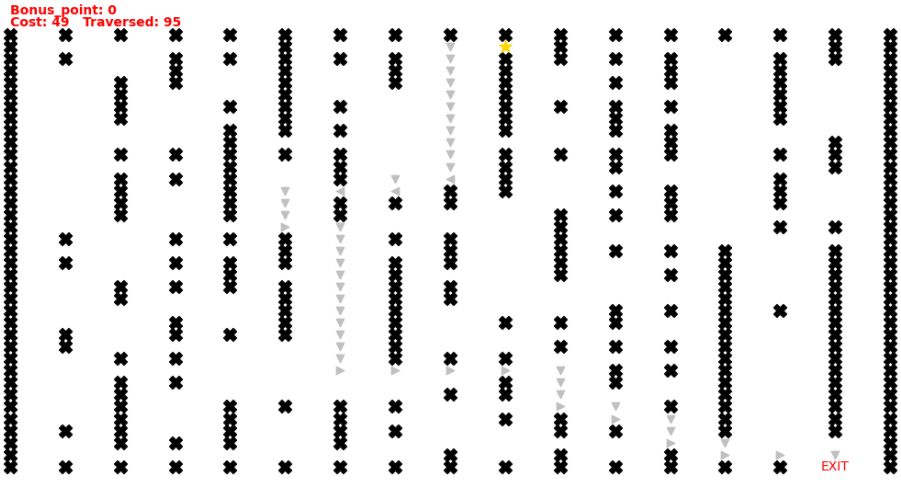
Start(1,9) - End(15,36)

d. A Star:

- Heuristic Manhattan:

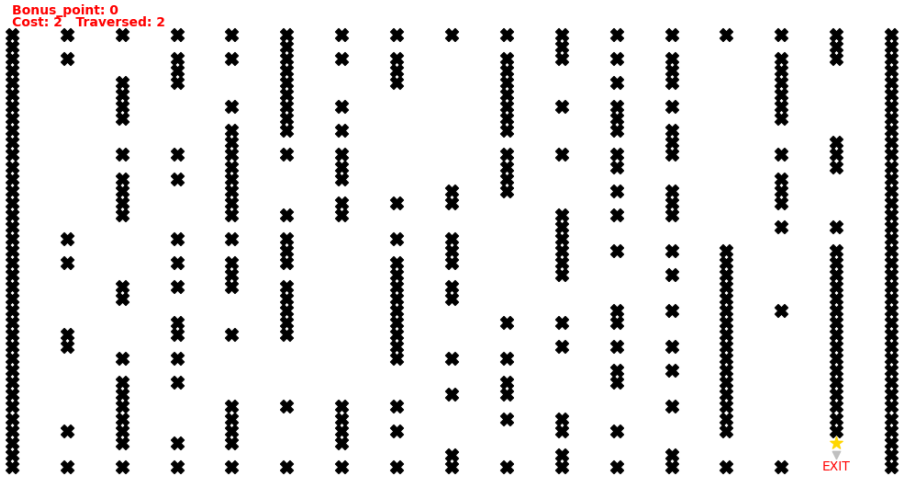


Start(15,35) - End(15,36)

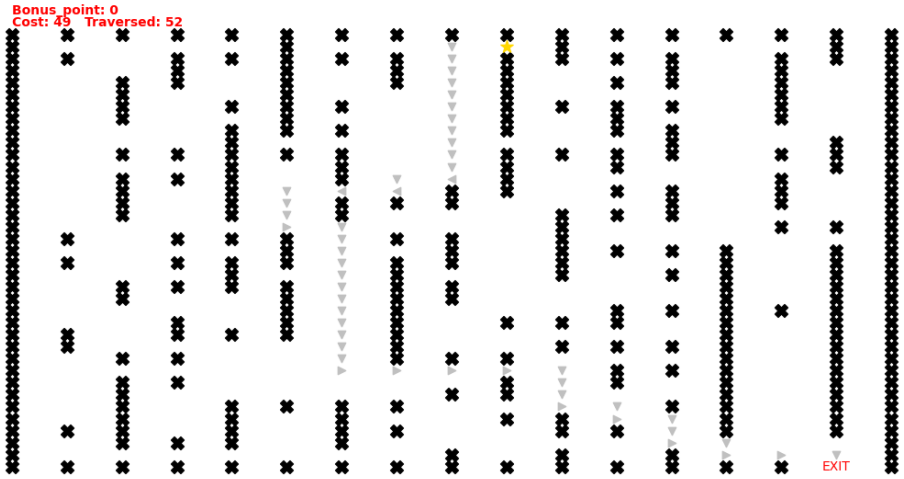


Start(1,9) - End(15,36)

- Heuristic 2 Times Euclid:

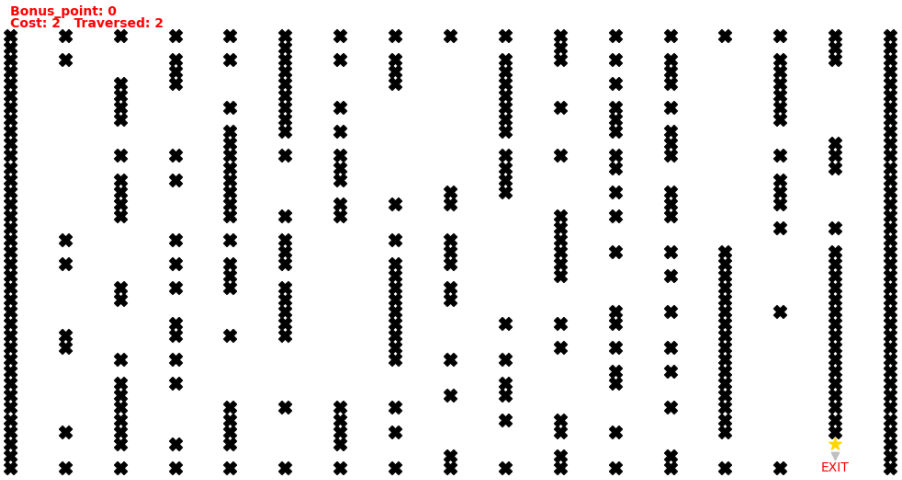


Start(15,35) - End(15,36)

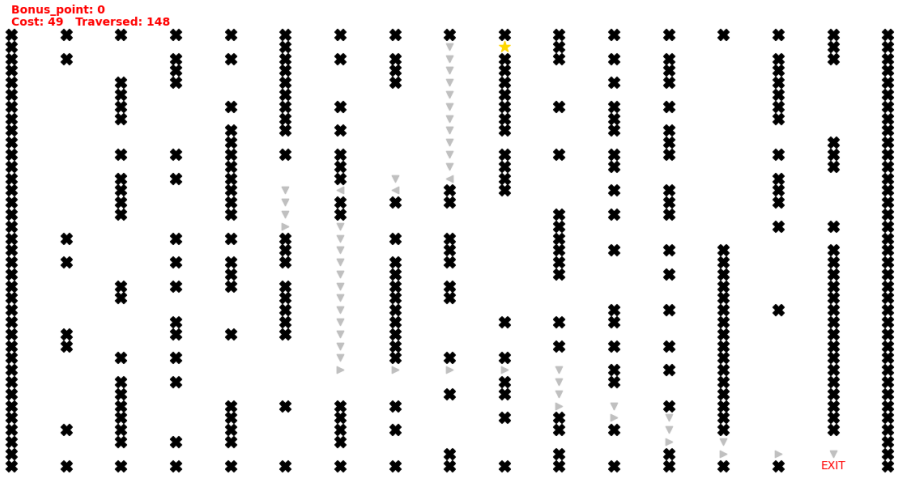


Start(1,9) - End(15,36)

- Heuristic Euclid:



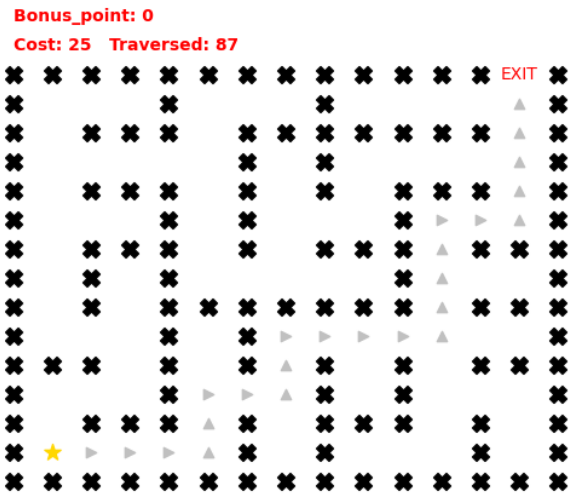
Start(15,35) - End(15,36)



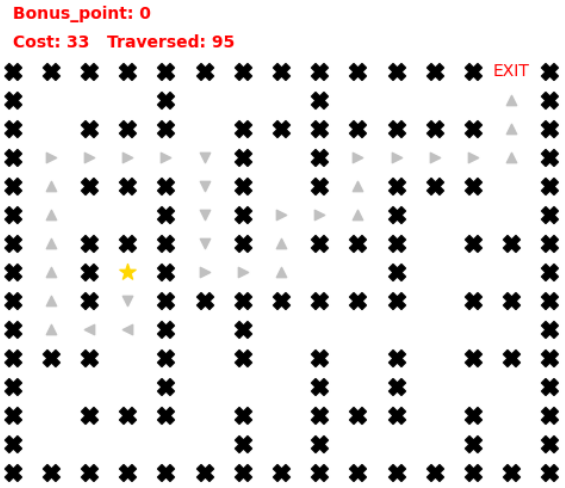
Start(1,9) - End(15,36)

2. Map 2

a. BFS:

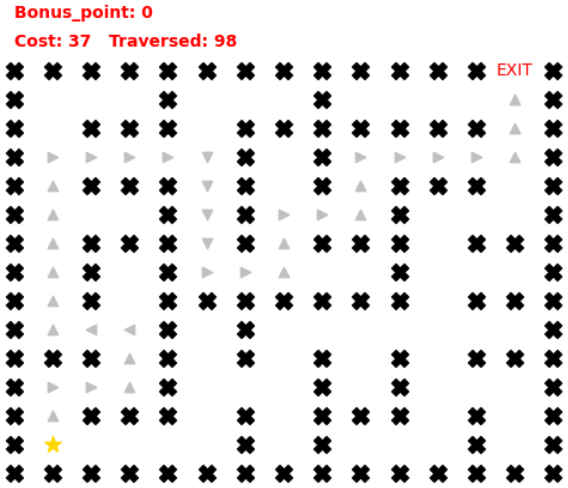


Start(13,1) - End(0,13)

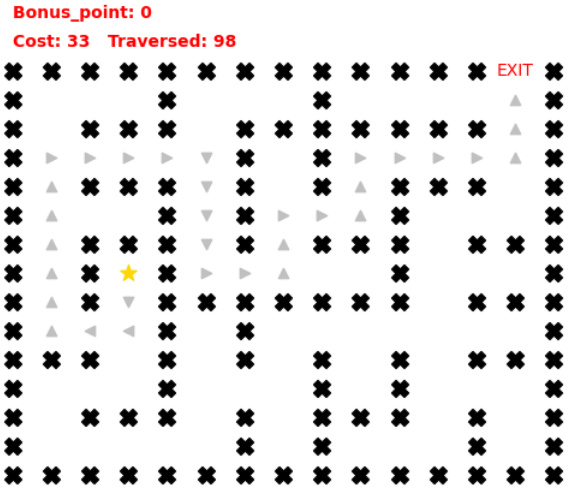


Start(7,3) - End(0,13)

b. DFS:



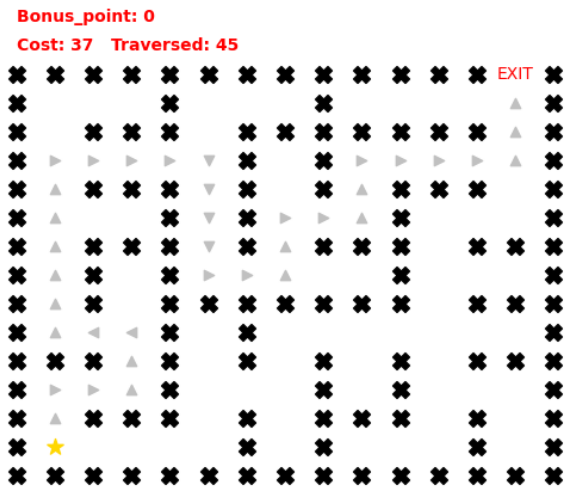
Start(13,1) - End(0,13)



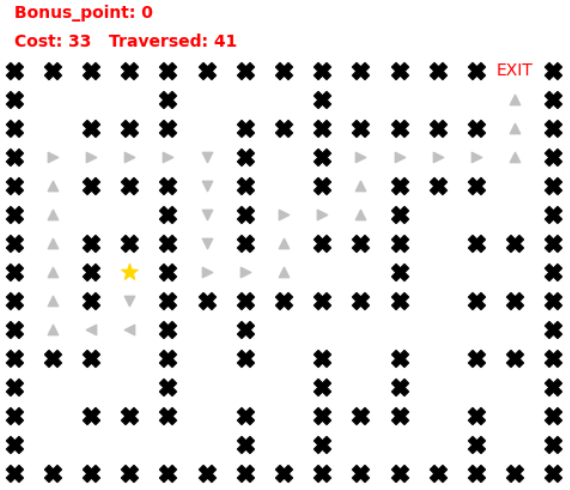
Start(7,3) - End(0,13)

c. GBFS:

- Heuristic Euclid:

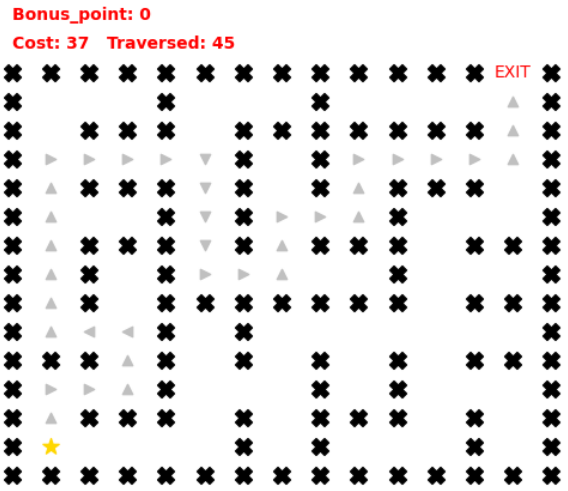


Start(13,1) - End(0,13)

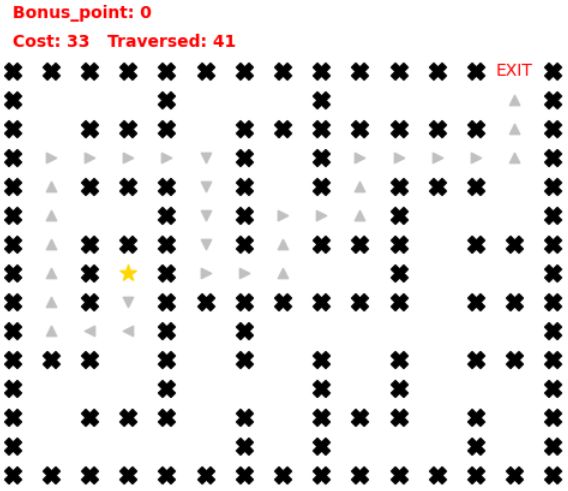


Start(7,3) - End(0,13)

- Heuristic 2 Times Euclid:

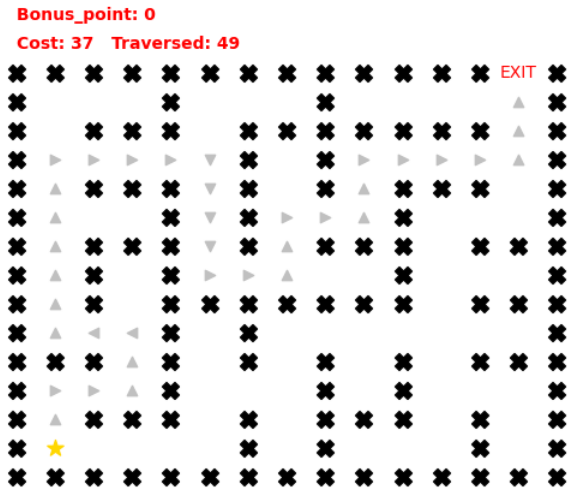


Start(13,1) - End(0,13)

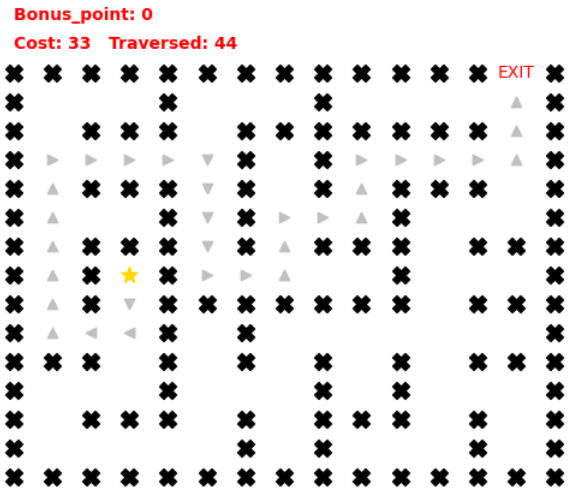


Start(7,3) - End(0,13)

- Heuristic Manhattan:



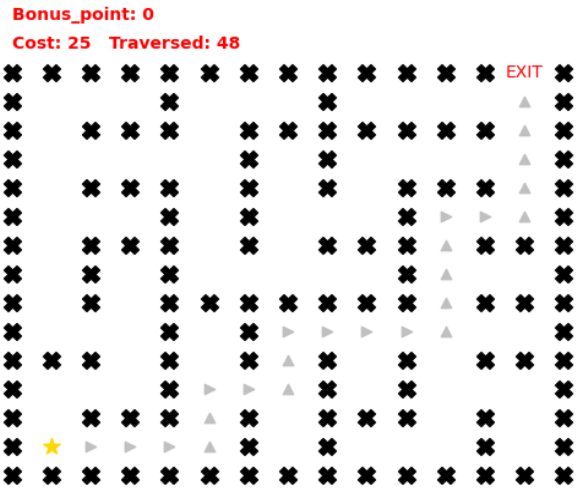
Start(13,1) - End(0,13)



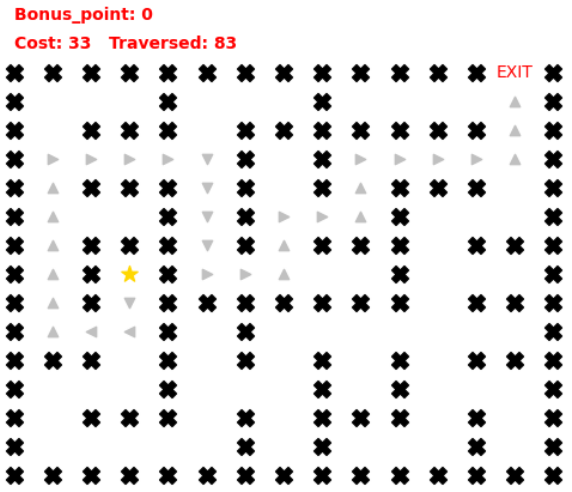
Start(7,3) - End(0,13)

d. A Star:

- Heuristic Euclid:

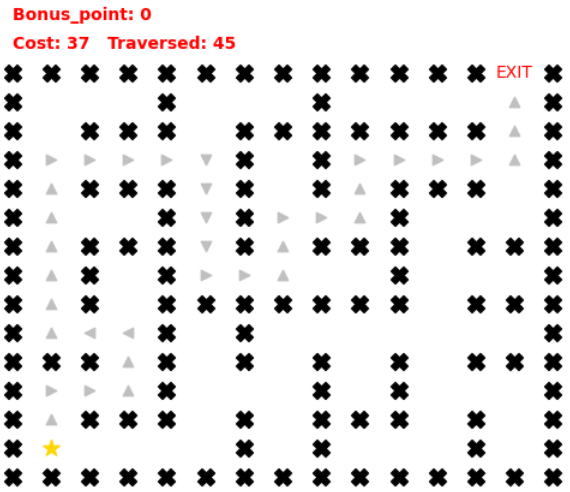


Start(13,1) - End(0,13)

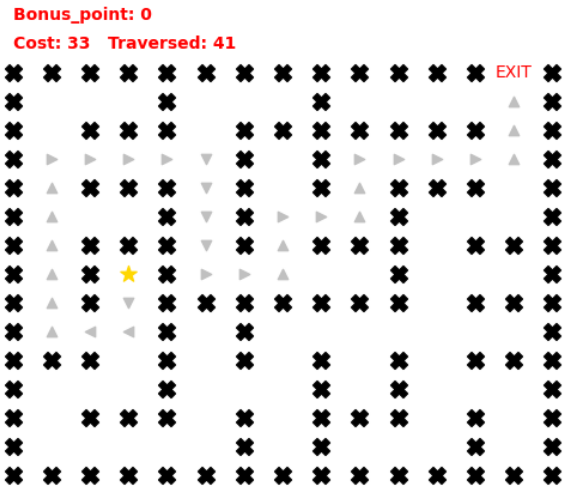


Start(7,3) - End(0,13)

- Heuristic 2 Times Euclid:

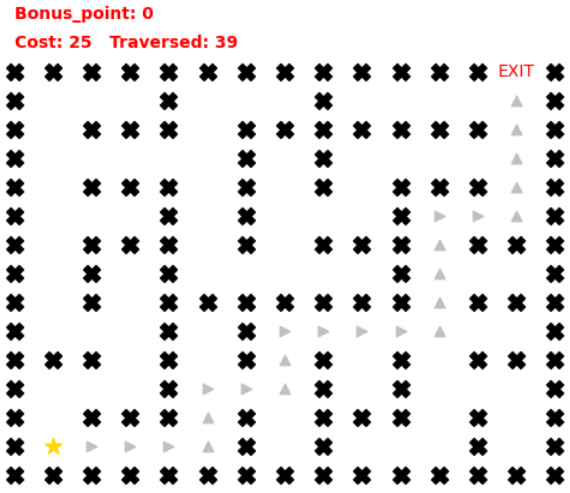


Start(13,1) - End(0,13)

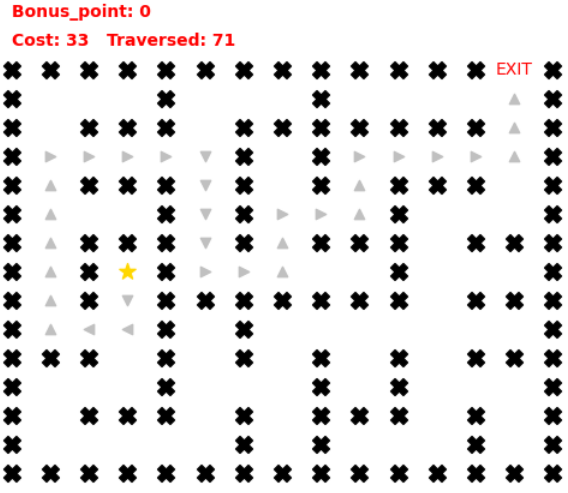


Start(7,3) - End(0,13)

- Heuristic Manhattan:



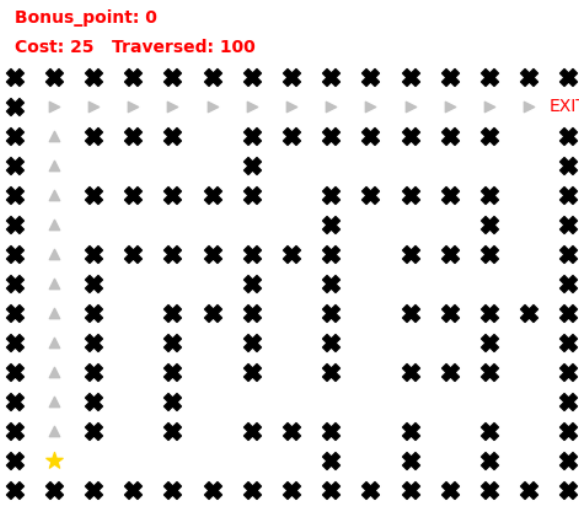
Start(13,1) - End(0,13)



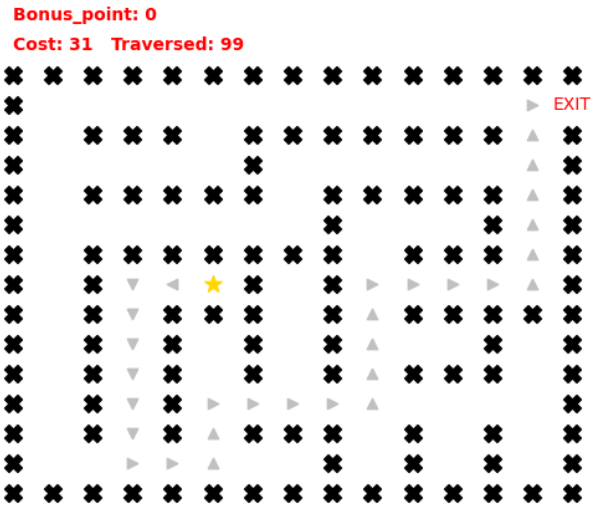
Start(7,3) - End(0,13)

3. Map 3

a. BFS:

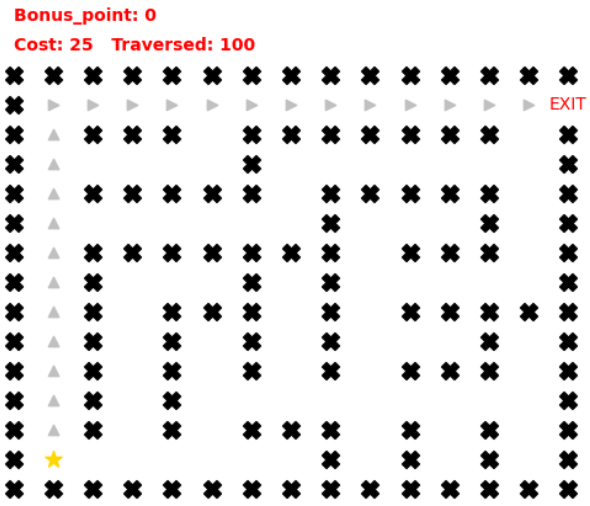


Start(13,1) - End(1,14)

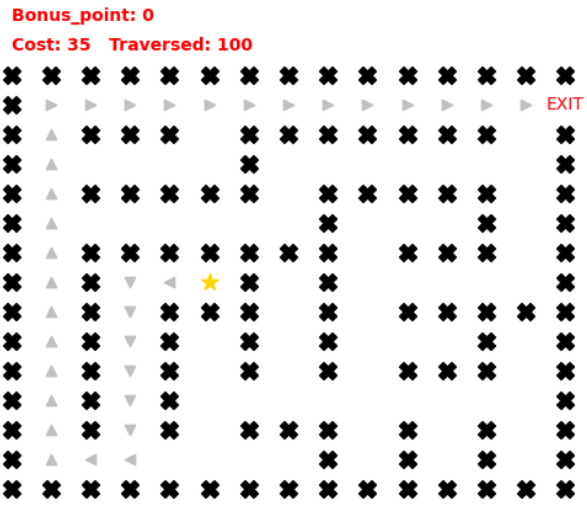


Start(7,5) - End(1,14)

b. DFS:



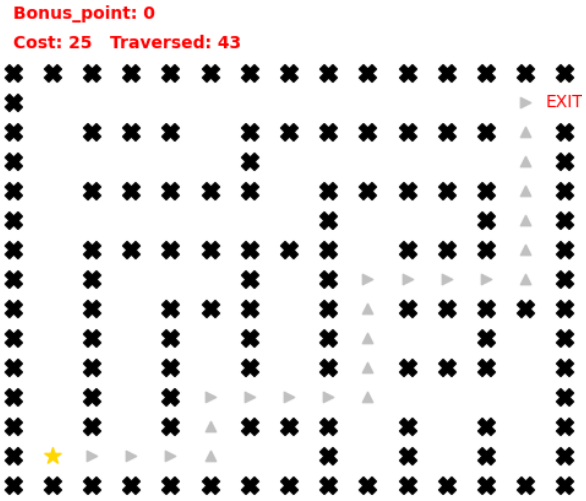
Start(13,1) - End(1,14)



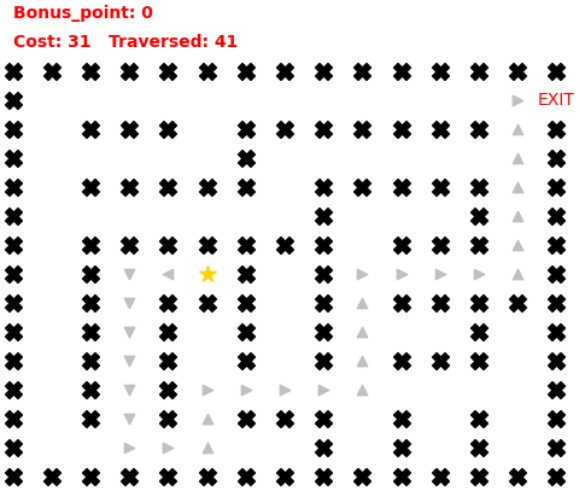
Start(7,5) - End(1,14)

c. GBFS:

- Heuristic Euclid:

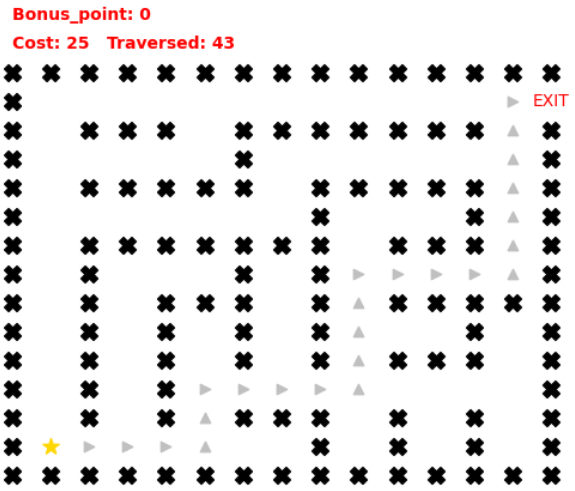


Start(13,1) - End(1,14)

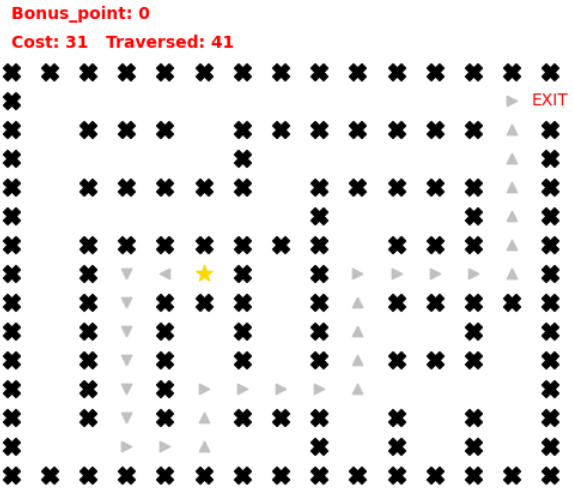


Start(7,5) - End(1,14)

- Heuristic 2 Times Euclid:

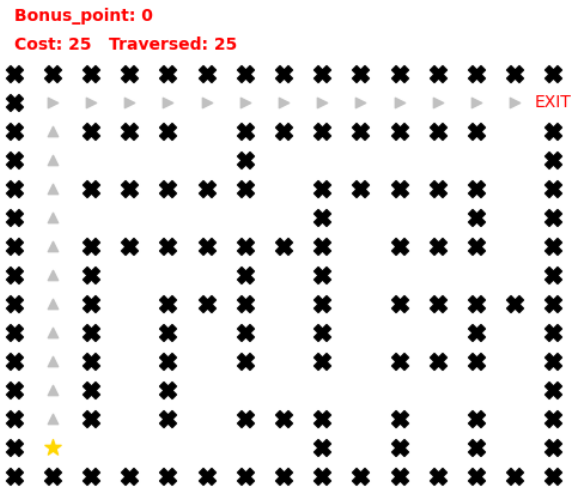


Start(13,1) - End(1,14)

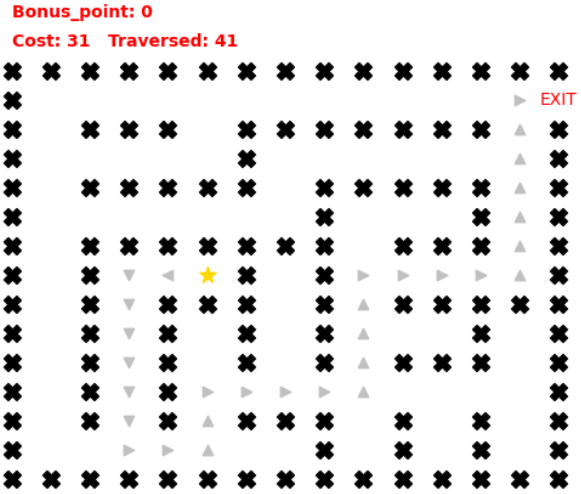


Start(7,5) - End(1,14)

- Heuristic Manhattan:



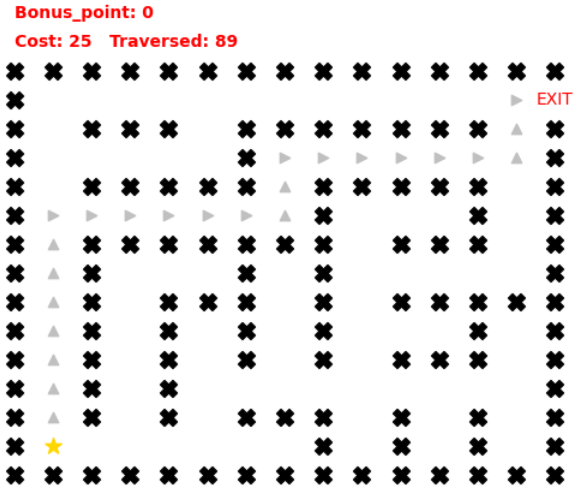
Start(13,1) - End(1,14)



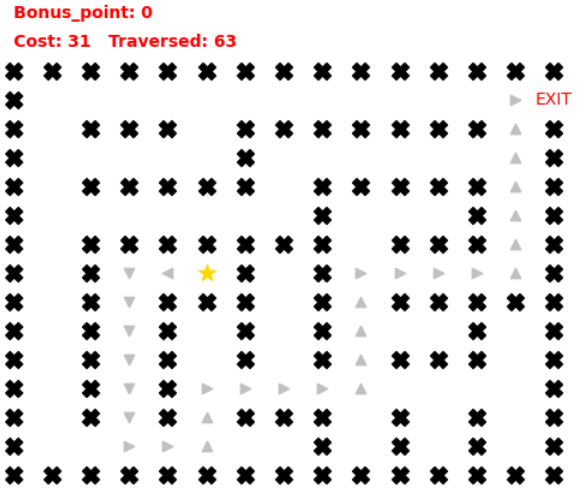
Start(7,5) - End(1,14)

d. A Star:

- Heuristic Euclid:

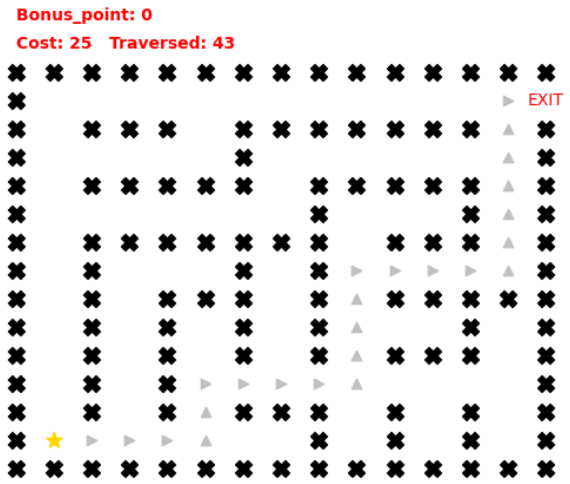


Start(15,1) - End(1,14)

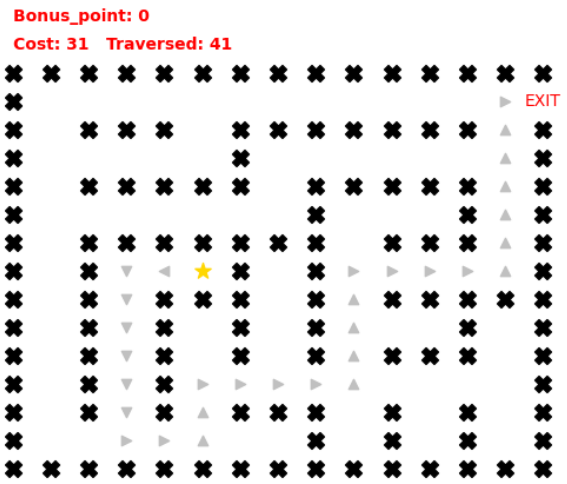


Start(7,5) - End(1,14)

- Heuristic 2 Times Euclid:

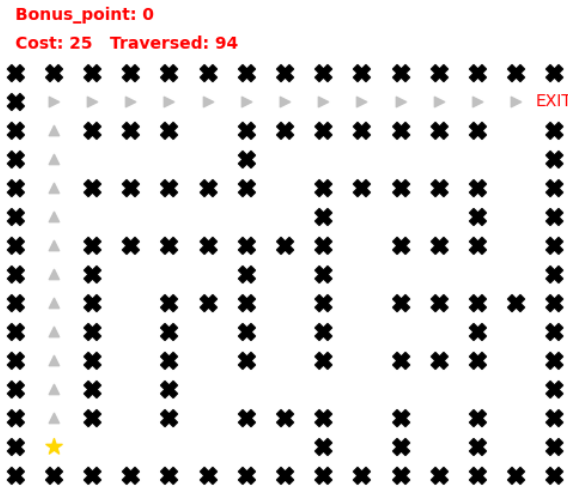


Start(13,1) - End(1,14)

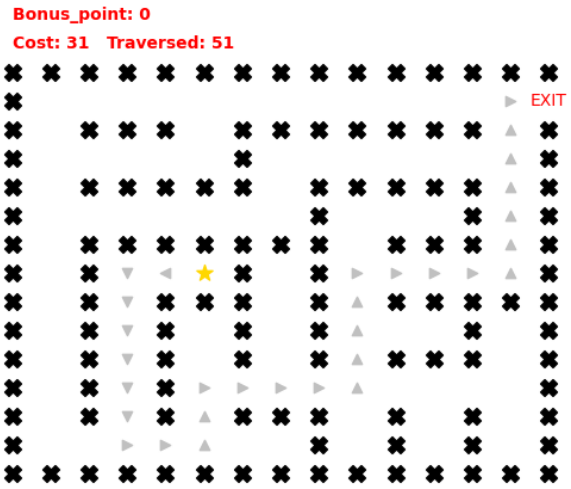


Start(7,5) - End(1,14)

- Heuristic Manhattan:



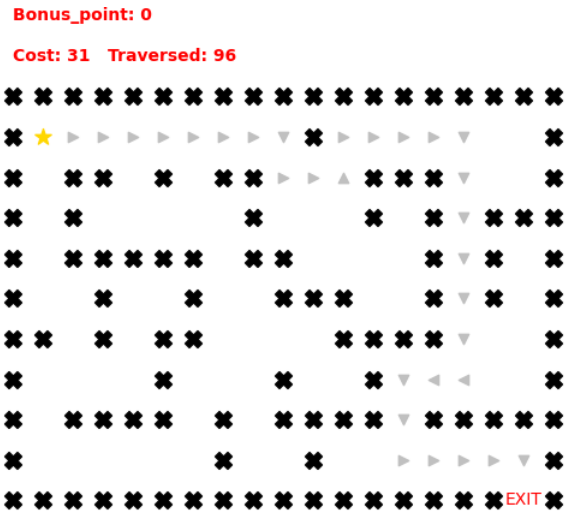
Start(13,1) - End(1,14)



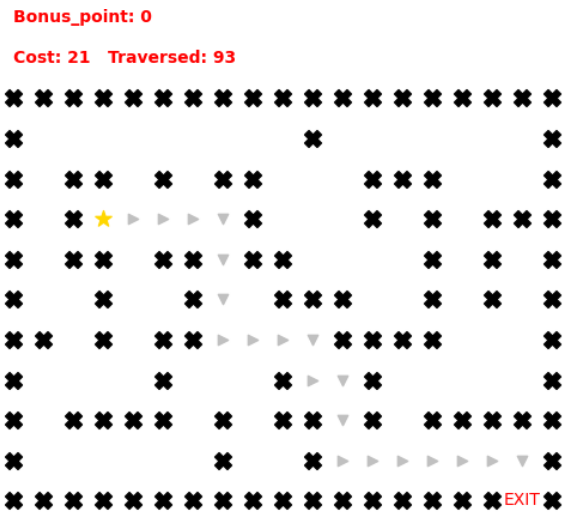
Start(7,5) - End(1,14)

4. Map 4

a. BFS:

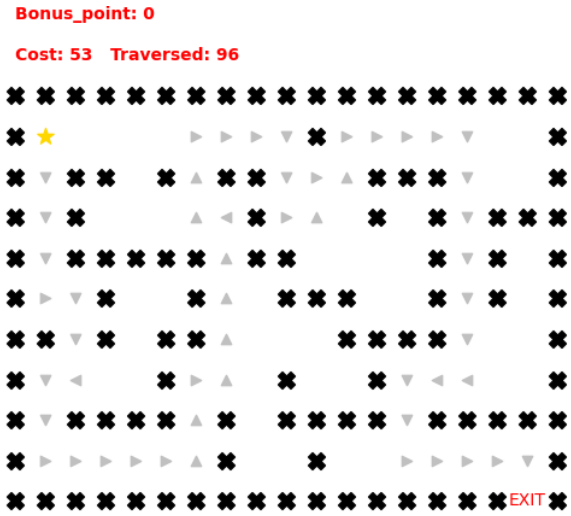


Start(1,1) - End(10,17)



Start(3,3) - End(10,17)

b. DFS:

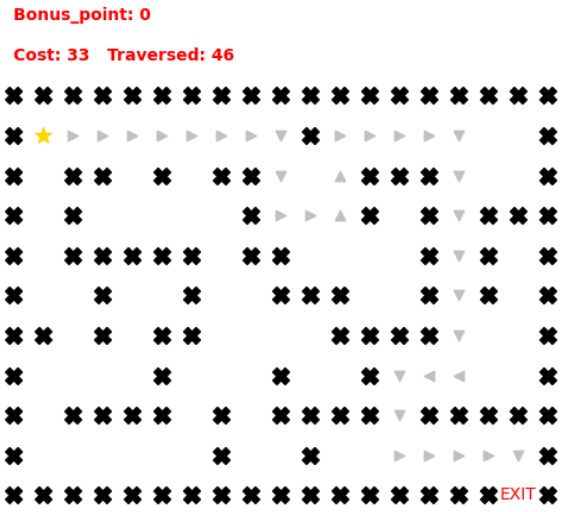


Start(1,1) - End(10,17)

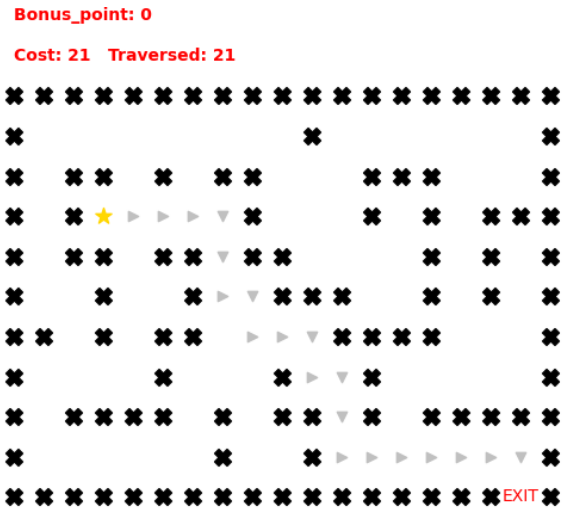
Start(3,3) - End(10,17)

c. GBFS:

- Heuristic Euclid:

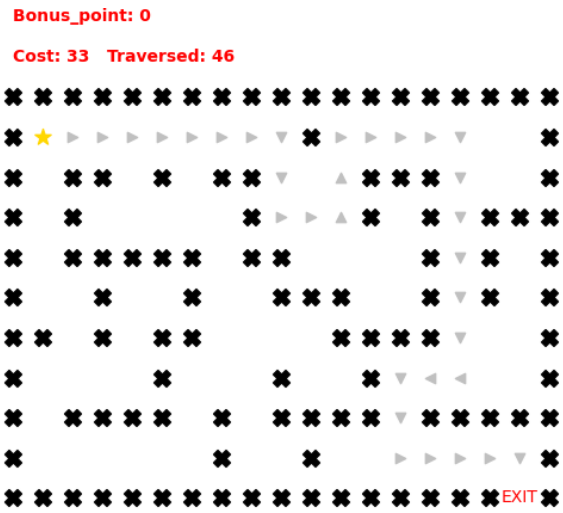


Start(1,1) - End(10,17)

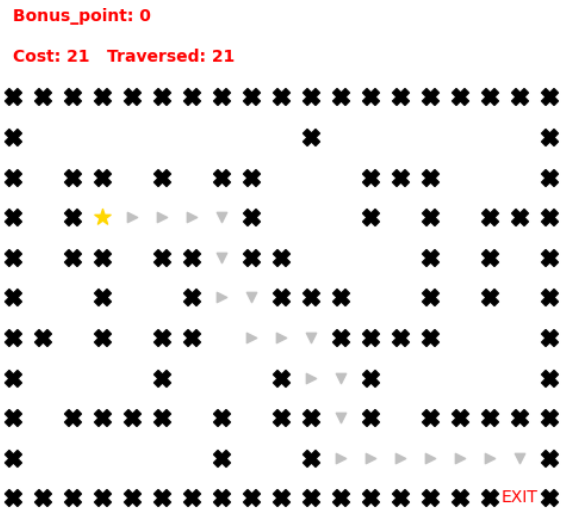


Start(3,3) - End(10,17)

- Heuristic 2 Times Euclid:

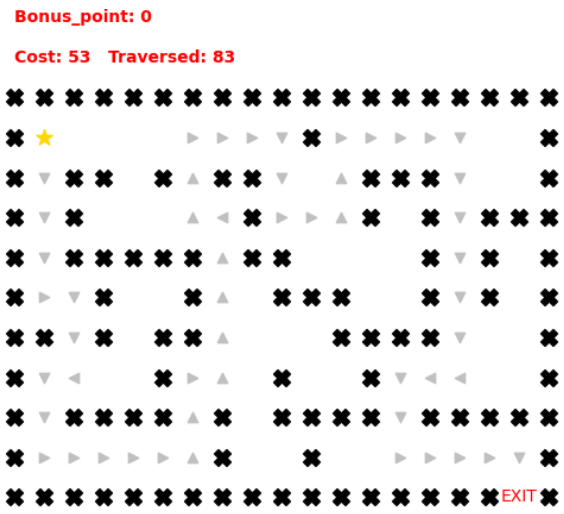


Start(1,1) - End(10,17)

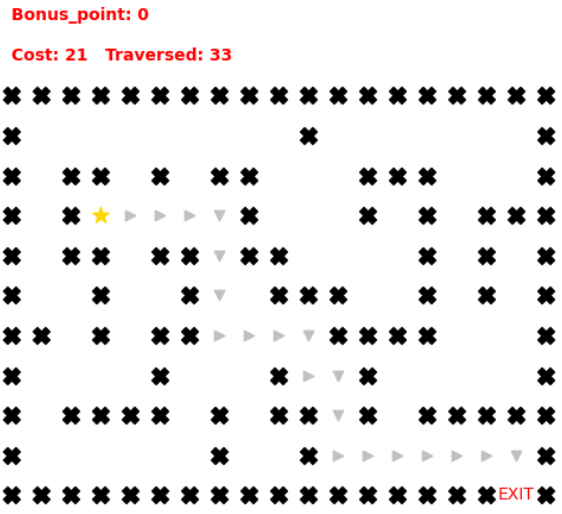


Start(3,3) - End(10,17)

- Heuristic Manhattan:



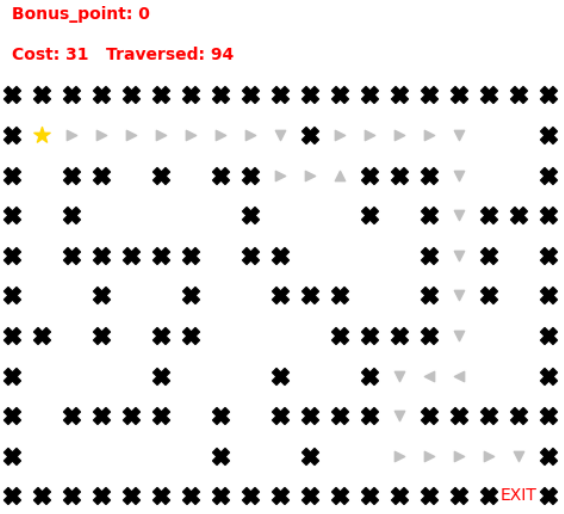
Start(1,1) - End(10,17)



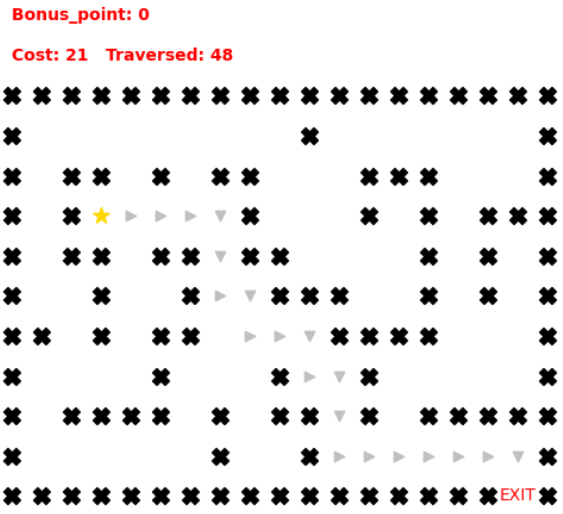
Start(3,3) - End(10,17)

d. A Star:

- Heuristic Euclid:

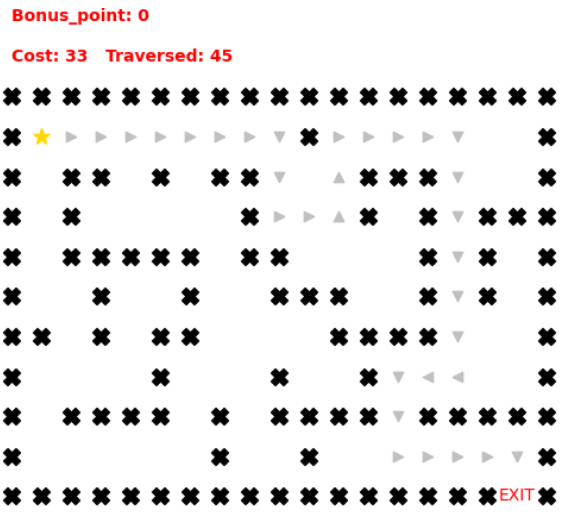


Start(1,1) - End(10,17)

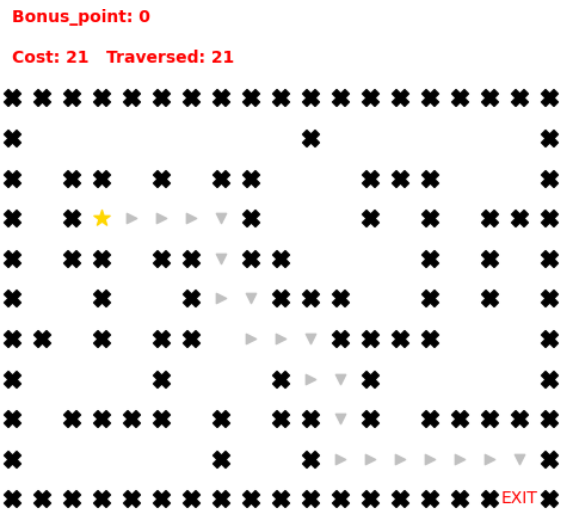


Start(3,3) - End(10,17)

- Heuristic 2 Times Euclid:

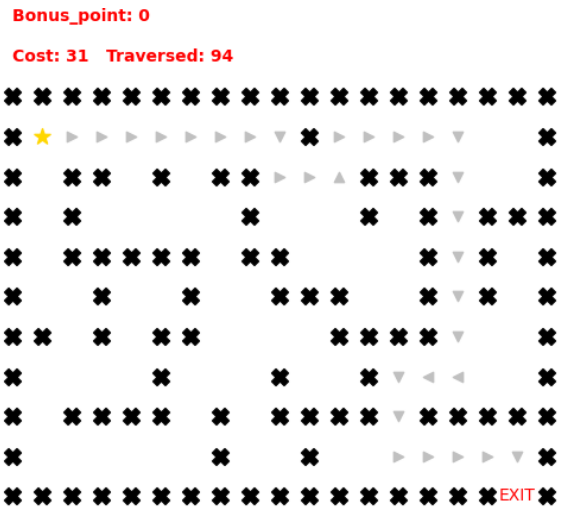


Start(1,1) - End(10,17)

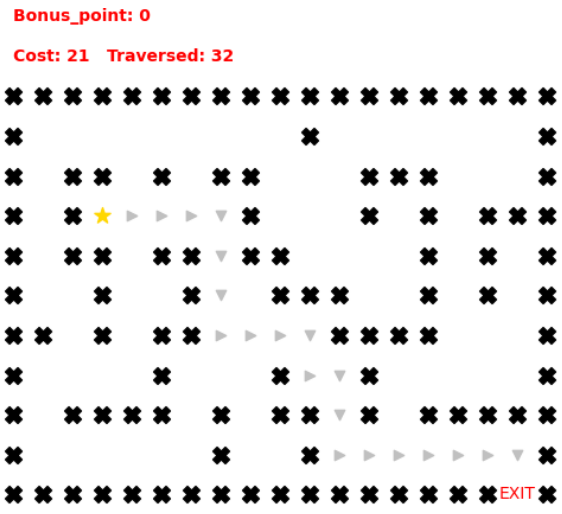


Start(3,3) - End(10,17)

- Heuristic Manhattan:



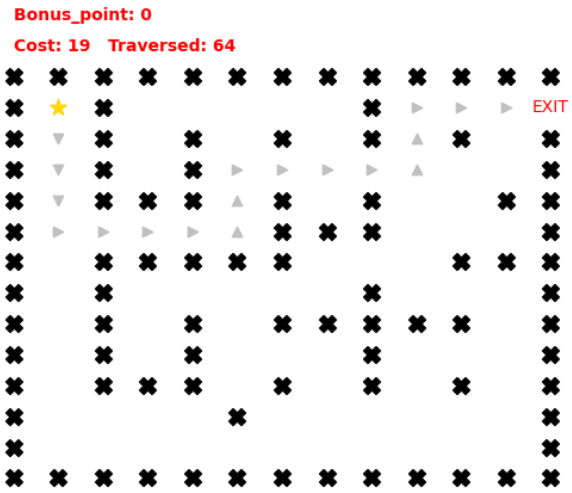
Start(1,1) - End(10,17)



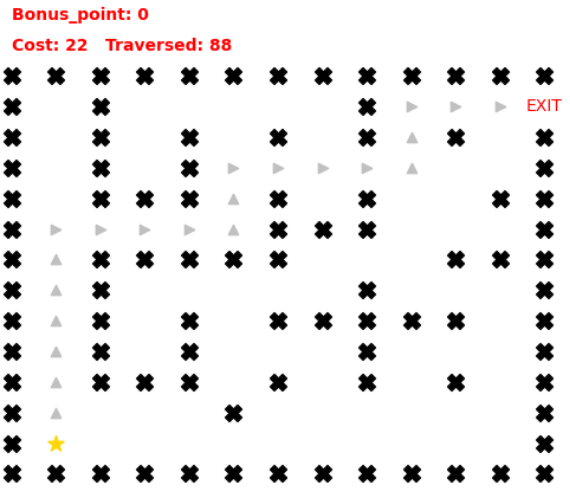
Start(3,3) - End(10,17)

5. Map 5

a. BFS:

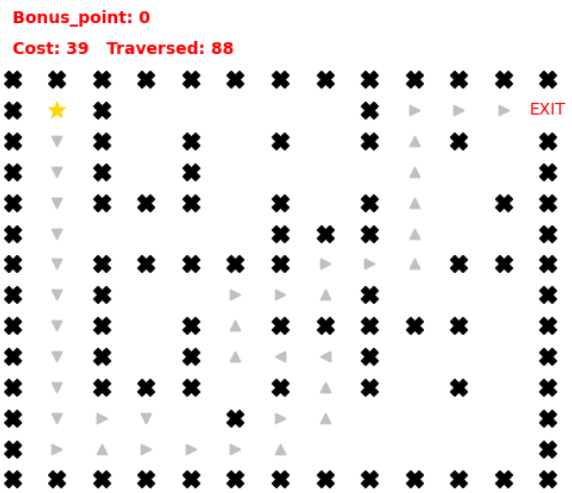


Start(1,1) - End(1,12)

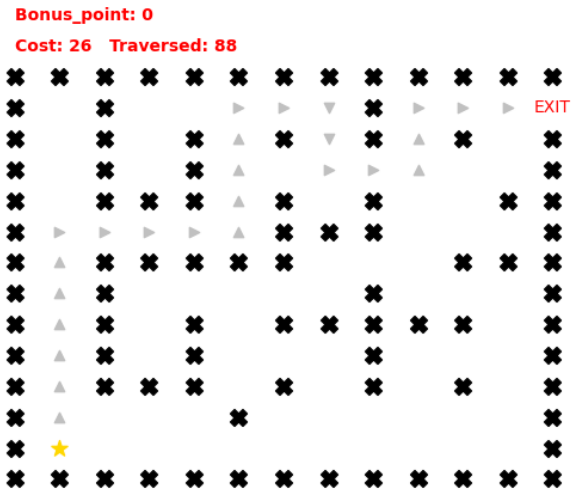


Start(12,1) - End(1,12)

b. DFS:



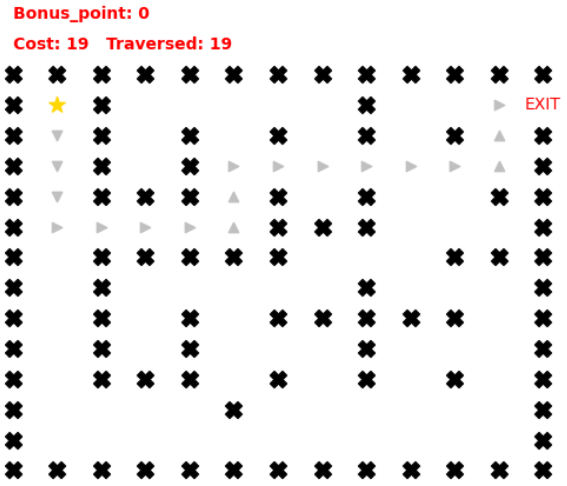
Start(1,1) - End(1,12)



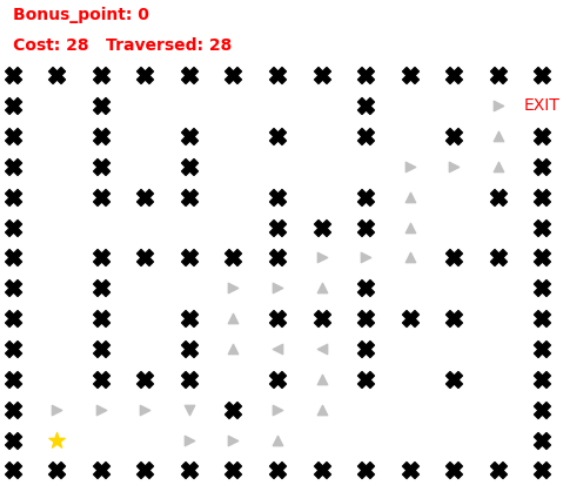
Start(12,1) - End(1,12)

c. GBFS:

- Heuristic Euclid:

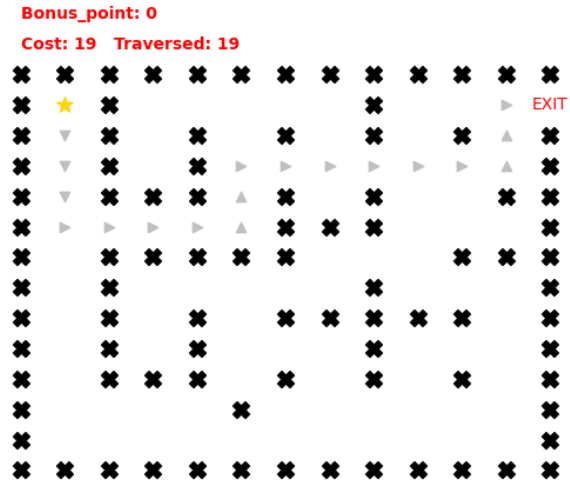


Start(1,1) - End(1,12)

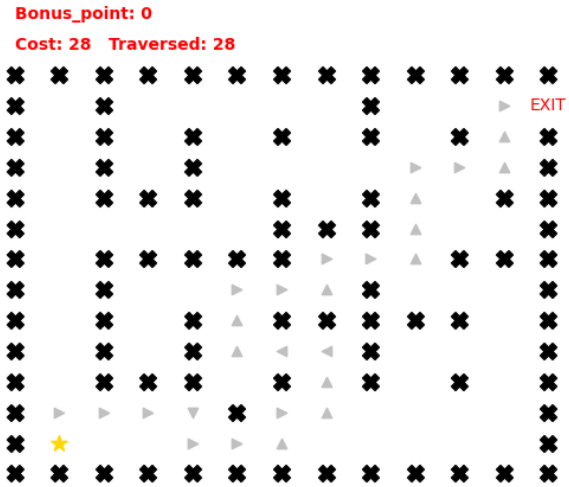


Start(12,1) - End(1,12)

- Heuristic 2 Times Euclid:

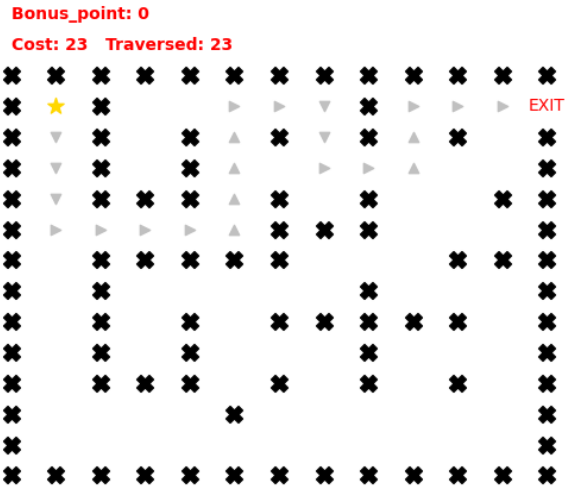


Start(1,1) - End(1,12)

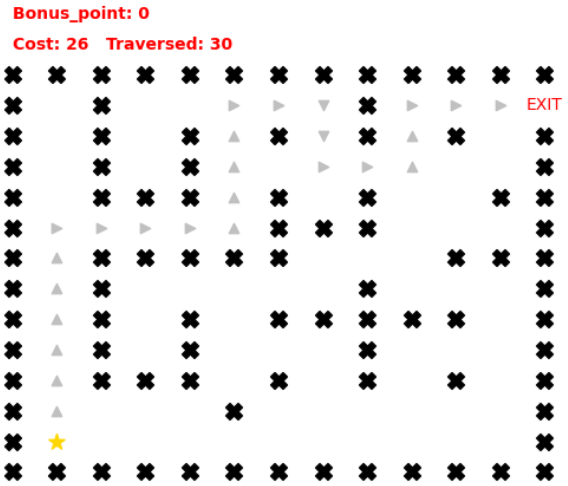


Start(12,1) - End(1,12)

- Heuristic Manhattan:



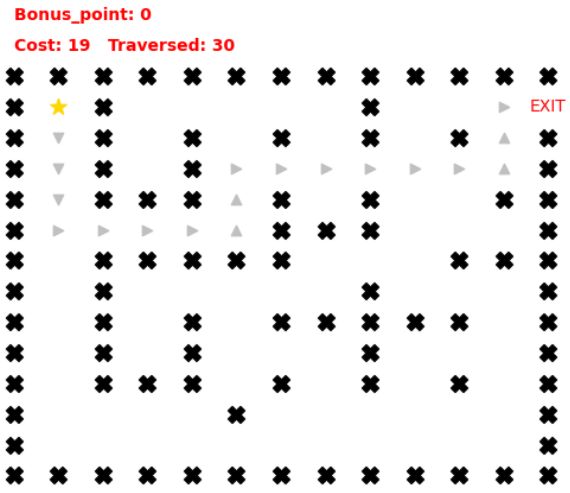
Start(1,1) - End(1,12)



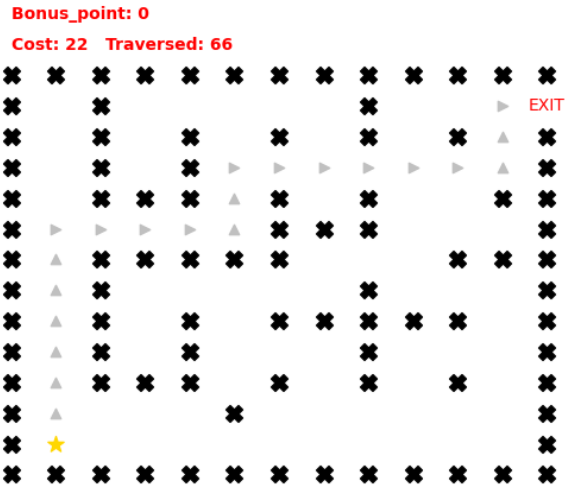
Start(12,1) - End(1,12)

d. A Star:

- Heuristic Euclid:

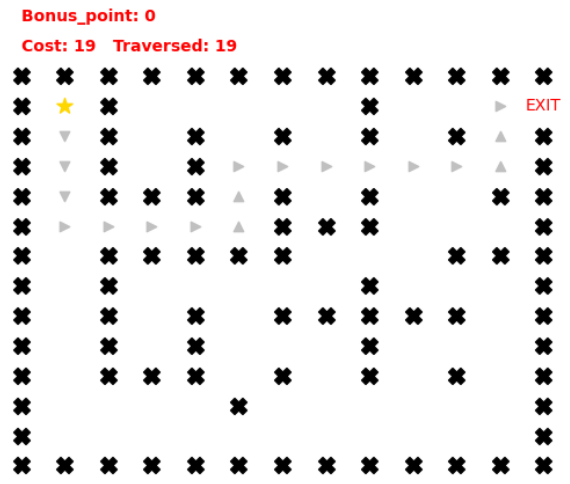


Start(1,1) - End(1,12)

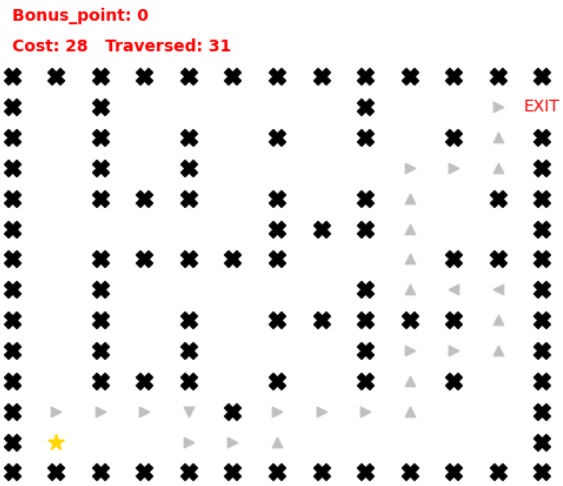


Start(12,1) - End(1,12)

- Heuristic 2 Times Euclid:

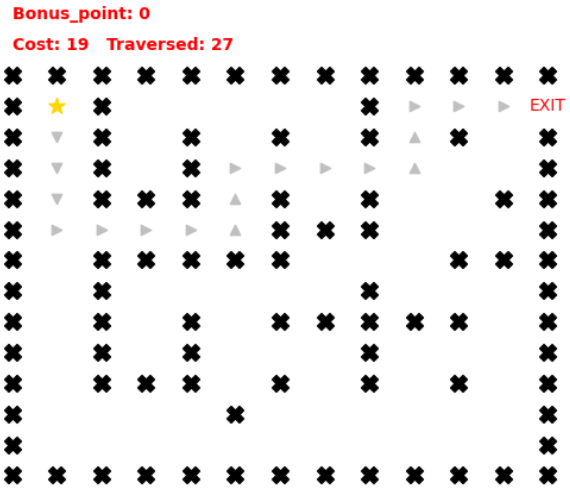


Start(1,1) - End(1,12)

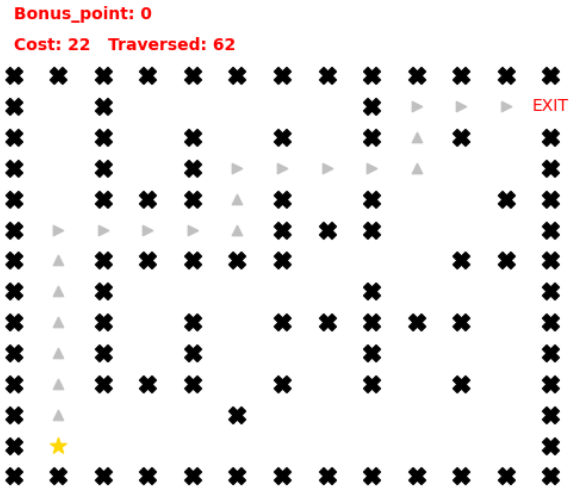


Start(12,1) - End(1,12)

- Heuristic Manhattan:



Start(1,1) - End(1,12)



Start(12,1) - End(1,12)

III. KẾT QUẢ & ĐÁNH GIÁ

1. Map 1

a. Start = (15,35)

- Kết quả:

- Breadth First Search: cost = 2, traversed = 6
- Depth First Search: cost = 6, traversed = 258
- Greedy Best First Search:
  - Heuristic Euclid: cost = 2, traversed = 2
  - Heuristic Euclid^2: cost = 2, traversed = 2
  - Heuristic Manhattan: cost = 2, traversed = 2
- A*:
  - Heuristic Euclid: cost = 2, traversed = 2
  - Heuristic Euclid^2: cost = 2, traversed = 2
  - Heuristic Manhattan: cost = 2, traversed = 2

Thuật toán		Đầy đủ	Tối ưu	Giải thích
BFS		Có	Có	BFS luôn tối ưu vì mê cung có chi phí di chuyển qua mỗi bước là bằng nhau
DFS			Không	Phụ thuộc vào thứ tự duyệt điểm tiếp theo của thuật toán (ở đây là ‘trên’, ‘trái’, ‘dưới’, ‘phải’) và vị trí đặt điểm bắt đầu
GBFS	H_euclid		Có	
	H_euclid^2		Có	
	H_manhattan		Có	
A*	H_euclid		Có	
	H_euclid^2		Có	
	H_manhattan		Có	

b. Start = (1,9)

- Kết quả:

- Breadth First Search: cost = 49, traversed = 286
- Depth First Search: cost = 75, traversed = 299
- Greedy Best First Search:
  - Heuristic Euclid: cost = 49, traversed = 52
  - Heuristic Euclid^2: cost = 49, traversed = 52
  - Heuristic Manhattan: cost = 51, traversed = 57
- A*:
  - Heuristic Euclid: cost = 49, traversed = 148
  - Heuristic Euclid^2: cost = 49, traversed = 52
  - Heuristic Manhattan: cost = 49, traversed = 95

Thuật toán		Đầy đủ	Tối ưu	Giải thích
BFS		Có	Có	Chi phí di chuyển qua mỗi bước là bằng nhau
DFS			Không	Phụ thuộc vào thứ tự duyệt và vị trí điểm bắt đầu
GBFS	H_euclid		Có	
	H_euclid^2		Có	
	H_manhattan		Không	Vì tồn tại một số điểm có số bước đi theo cạnh HCN bằng nhau nên việc lựa chọn giá trị heuristic nhỏ nhất phụ thuộc vào thứ tự duyệt trong list biên fringe
A*	H_euclid		Có	
	H_euclid^2		Có	
	H_manhattan		Có	

2. Map 2

a. Start = (13,1)

- Kết quả:

- Breadth First Search: cost = 25, traversed = 87
- Depth First Search: cost = 37, traversed = 98
- Greedy Best First Search:
  - Heuristic Euclid: cost = 37, traversed = 45
  - Heuristic Euclid^2: cost = 37, traversed = 45
  - Heuristic Manhattan: cost = 37, traversed = 49
- A*:
  - Heuristic Euclid: cost = 25, traversed = 48
  - Heuristic Euclid^2: cost = 37, traversed = 45
  - Heuristic Manhattan: cost = 25, traversed = 39

Thuật toán		Đầy đủ	Tối ưu	Giải thích
BFS		Có	Có	Chi phí di chuyển qua mỗi bước là bằng nhau
DFS			Không	Phụ thuộc vào thứ tự duyệt và vị trí điểm bắt đầu
GBFS	H_euclid		Không	Vì chỉ xét heuristic nên càng kiếm được h(n) bé thì càng duyệt
	H_euclid^2			
	H_manhattan			
A*	H_euclid		Có	Vì trong fringe còn những điểm có path_cost + heuristic bé
	H_euclid^2		Không	Vì độ lớn heuristic >> chi phí đường đi nên A* ~ GBFS
	H_manhattan		Có	//Như euclid

b. Start = (7,3)

- Kết quả:

- Breadth First Search: cost = 33, traversed = 95

- Depth First Search: cost = 33, traversed = 98
- Greedy Best First Search:
  - Heuristic Euclid: cost = 33, traversed = 41
  - Heuristic Euclid^2: cost = 33, traversed = 41
  - Heuristic Manhattan: cost = 33, traversed = 44
- A*:
  - Heuristic Euclid: cost = 33, traversed = 83
  - Heuristic Euclid^2: cost = 33, traversed = 41
  - Heuristic Manhattan: cost = 33, traversed = 71

Thuật toán		Đầy đủ	Tối ưu	Giải thích
BFS		Có	Có	Chi phí di chuyển qua mỗi bước là bằng nhau
DFS			Có	Rơi vào trường hợp tối ưu của DFS
GBFS	H_euclid		Có	
	H_euclid^2			
	H_manhattan			
A*	H_euclid		Có	
	H_euclid^2			
	H_manhattan			

3. Map 3

a. <u>Start = (13,1)</u>

- Kết quả:

- Breadth First Search: cost = 25, traversed = 100
- Depth First Search: cost = 25, traversed = 100
- Greedy Best First Search:
  - Heuristic Euclid: cost = 25, traversed = 43
  - Heuristic Euclid^2: cost = 25, traversed = 43
  - Heuristic Manhattan: cost = 25, traversed = 25
- A*:
  - Heuristic Euclid: cost = 25, traversed = 89

- Heuristic Euclid^2: cost = 25, traversed = 43
- Heuristic Manhattan: cost = 25, traversed = 94

Thuật toán		Đầy đủ	Tối ưu	Giải thích
BFS		Có	Có	Chi phí di chuyển qua mỗi bước là bằng nhau
DFS			Có	Rơi vào trường hợp tối ưu của DFS
GBFS	H_euclid		Có	
	H_euclid^2			
	H_manhattan			
A*	H_euclid		Có	
	H_euclid^2			
	H_manhattan			

b. <u>Start = (7,5)</u>

- Kết quả:

- Breadth First Search: cost = 31, traversed = 99
- Depth First Search: cost = 35, traversed = 100
- Greedy Best First Search:
  - Heuristic Euclid: cost = 31, traversed = 41
  - Heuristic Euclid^2: cost = 31, traversed = 41
  - Heuristic Manhattan: cost = 31, traversed = 41
- A*:
  - Heuristic Euclid: cost = 31, traversed = 63
  - Heuristic Euclid^2: cost = 31, traversed = 41
  - Heuristic Manhattan: cost = 31, traversed = 51

Thuật toán	Đầy đủ	Tối ưu	Giải thích
BFS	Có	Có	Chi phí di chuyển qua mỗi bước là bằng nhau

DFS			Không	Phụ thuộc vào thứ tự duyệt và vị trí điểm bắt đầu
GBFS	H_euclid		Có	
	H_euclid^2			
	H_manhattan			
A*	H_euclid		Có	
	H_euclid^2			
	H_manhattan			

4. Map 4

a. Start = (1,1)

- Kết quả:

- Breadth First Search: cost = 31, traversed = 96
- Depth First Search: cost = 53, traversed = 96
- Greedy Best First Search:
  - Heuristic Euclid: cost = 33, traversed = 46
  - Heuristic $Euclid^2$: cost = 33, traversed = 46
  - Heuristic Manhattan: cost = 53, traversed = 83
- A*:
  - Heuristic Euclid: cost = 31, traversed = 94
  - Heuristic $Euclid^2$: cost = 33, traversed = 45
  - Heuristic Manhattan: cost = 31, traversed = 94

Thuật toán		Đầy đủ	Tối ưu	Giải thích
BFS		Có	Có	Chi phí di chuyển qua mỗi bước là bằng nhau
DFS			Không	Phụ thuộc vào thứ tự duyệt và vị trí điểm bắt đầu
GBFS	H_euclid		Không	Vì chỉ xét heuristic nên càng kiếm được h(n) bé thì càng duyệt
	H_euclid^2			
	H_manhattan			

A*	H_euclid		Có	Vì trong fringe còn những điểm có path_cost + heuristic bé
	H_euclid^2		Không	Vì độ lớn heuristic >> chi phí đường đi nên A* ~ GBFS
	H_manhattan		Có	//Như euclid

b. Start = (3,3)

- Kết quả:

- Breadth First Search: cost = 21, traversed = 93
- Depth First Search: cost = 59, traversed = 98
- Greedy Best First Search:
  - Heuristic Euclid: cost = 21, traversed = 21
  - Heuristic $Euclid^2$: cost = 21, traversed = 21
  - Heuristic Manhattan: cost = 21, traversed = 83
- A*:
  - Heuristic Euclid: cost = 21, traversed = 48
  - Heuristic $Euclid^2$: cost = 21, traversed = 21
  - Heuristic Manhattan: cost = 21, traversed = 32

Thuật toán		Đầy đủ	Tối ưu	Giải thích
BFS		Có	Có	Chi phí di chuyển qua mỗi bước là bằng nhau
DFS			Không	Phụ thuộc vào thứ tự duyệt và vị trí điểm bắt đầu
GBFS	H_euclid		Có	
	H_euclid^2			
	H_manhattan			
A*	H_euclid		Có	
	H_euclid^2			
	H_manhattan			

5. Map 5

a. Start = (1,1)

- Kết quả:

- Breadth First Search: cost = 19, traversed = 64
- Depth First Search: cost = 39, traversed = 88
- Greedy Best First Search:
  - Heuristic Euclid: cost = 19, traversed = 19
  - Heuristic Euclid^2: cost = 19, traversed = 19
  - Heuristic Manhattan: cost = 23, traversed = 23
- A*:
  - Heuristic Euclid: cost = 19, traversed = 30
  - Heuristic Euclid^2: cost = 19, traversed = 19
  - Heuristic Manhattan: cost = 19, traversed = 27

Thuật toán		Đầy đủ	Tối ưu	Giải thích
BFS		Có	Có	Chi phí di chuyển qua mỗi bước là bằng nhau
DFS			Không	Phụ thuộc vào thứ tự duyệt và vị trí điểm bắt đầu
GBFS	H_euclid		Có	
	H_euclid^2		Có	
	H_manhattan		Không	Phụ thuộc vào thứ tự duyệt trong list biên fringe
A*	H_euclid		Có	
	H_euclid^2			
	H_manhattan			

b. Start = (12,1)

- Kết quả:

- Breadth First Search: cost = 22, traversed = 88
- Depth First Search: cost = 26, traversed = 88
- Greedy Best First Search:

- Heuristic Euclid: cost = 28, traversed = 28
- Heuristic Euclid^2: cost = 28, traversed = 28
- Heuristic Manhattan: cost = 26, traversed = 83

- A*:
  - Heuristic Euclid: cost = 22, traversed = 66
  - Heuristic Euclid^2: cost = 28, traversed = 31
  - Heuristic Manhattan: cost = 22, traversed = 62

Thuật toán		Đầy đủ	Tối ưu	Giải thích
BFS		Có	Có	Chi phí di chuyển qua mỗi bước là bằng nhau
DFS			Không	Phụ thuộc vào thứ tự duyệt và vị trí điểm bắt đầu
GBFS	H_euclid		Không	Vì chỉ xét heuristic nên càng kiếm được h(n) bé thì càng duyệt
	H_euclid^2			
	H_manhattan			
A*	H_euclid		Có	
	H_euclid^2		Không	Vì độ lớn heuristic >> chi phí đường đi nên A* ~ GBFS
	H_manhattan		Có	

6. Đề xuất heuristic giải quyết bản đồ điểm thưởng

- Ý tưởng: cải tiến hàm tìm đường đi ngắn nhất, thêm vào đó hàm xét đường đi khả thi từ điểm đang được duyệt tới các điểm thưởng trong danh sách. Đường đi khả thi là đường từ điểm đang xét → điểm thưởng → điểm đang xét < độ lớn giá trị điểm thưởng

- Thực hiện (ý tưởng):

a. Xây dựng attribute ‘prev’ của mê cung thành một list các tuple với:

- prev[0] là vị trí quay lui của đường đi chính và đường đi ‘tới’ điểm thưởng
- prev[i] (i>=1) là vị trí quay lui của đường đi ‘từ’ điểm thưởng trở về đường đi chính

- Trong đó i tương ứng với thứ tự (i – 1) của điểm thưởng trong danh sách điểm thưởng (initial list prev chứa len(bonus_points) giá trị là None)

b. Xây dựng hàm possible_bonus(bonus_point, cur_point) với:

- Input:
  - bonus_point (tuple): vị trí điểm thưởng cần xét
  - cur_point (tuple): vị trí điểm đang duyệt hiện tại trên đường đi
- Output:
  - True nếu khả thi và False nếu không khả thi
- Mô tả: Thuật toán thực hiện tìm đường đi từ điểm cur_point đến bonus_point, nếu 2 lần chiều dài đường đi < abs(giá trị điểm thưởng) thì set các prev tương ứng và trả về True, ngược lại set các prev đã set khi tìm đường đi về None và hàm trả về False

c. Xây dựng hàm tìm đường đi có qua bonus sử dụng lại toàn bộ code:

- Với mỗi điểm cần duyệt tiếp theo (next) sau khi tìm ước tính chi phí nhỏ nhất, ta thực hiện hàm possible_bonus trong vòng lặp với bonus_point lần lượt được lấy từ list bonus_points.
- Trong vòng lặp tìm các điểm prev từ điểm end để trả về đường đi, ta duyệt từ prev[-1] → prev[0]

IV. TÀI LIỆU THAM KHẢO

Slide bài giảng Tìm Kiếm, Tìm Kiếm Heuristic – giáo viên Lê Hoài Bắc

Slide bài giảng 2_Search, 4_Informed_Search – giáo viên Nguyễn Ngọc Đức