ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHI MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO LAB 01:
TIỀN XỬ LÝ DỮ LIỆU

BỘ MÔN: CƠ SỞ TRÍ TUỆ NHÂN TẠO

19120533 - Ninh Duy Huy

19120557 - Trần Tuấn Kiệt

Phân công các thành viên trong nhóm :

Thành Viên	code	Công việc
19120533 - Ninh Duy Huy	DFS, A*	Tính chi phí, thời gian, sửa lỗi đánh máy trong báo cáo, xử lý lặp vô tận trong BFS.
19120557 - Trần Tuấn Kiệt	BFS, GBFS	Tổ chức file, cấu trúc class, hàm phụ trợ, độ phức tạp về ko gian, xử lí lặp vô tận trong DFS, GBFS.

- Mỗi thành viên tìm 2 hàm heuristic, sau đó cùng nhau lựa chọn những hàm heuristic phù hợp cho map có bonus points (ở đây ta chọn 3 hàm)
- Mỗi thành viên tự vẽ 4 map ko có điểm thưởng & 2 map có điểm thưởng, sau đó chọn ra những map phù hợp trong 12 map này.
- Báo cáo: mỗi người tự viết phần báo cáo tương ứng với phần mình làm, chú thích vào trong source code bằng tiếng anh để người còn lại đọc dễ hiểu.

Một số quy ước trước khi thiết kế:

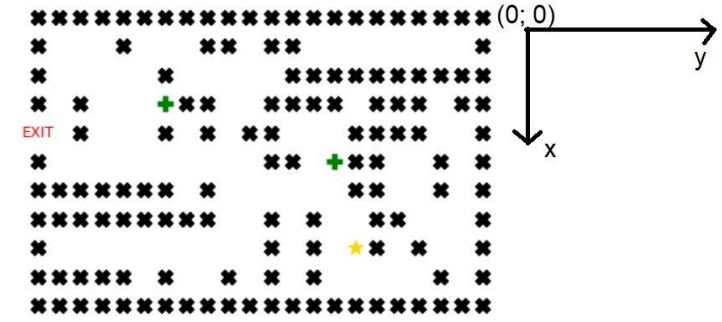
Con trỏ quay lui:

Lấy ý tưởng trong thần thoại Hy Lạp, người anh hùng Theseus đã vào trong mê cung, hạ quái vật nửa người, nửa bò Minotaur và quay trở ra bằng sợi chỉ đã để lại trước đó. Trong việc xử lý cũng vậy, đi vào mê cung mà không có "sợi chỉ" sẽ khiến lúc ra cũng khó khăn như lúc vào vậy, vì ta không thể nhớ hết mọi ngóc ngách.

Chính vì thế mà con trỏ quay lui sẽ lưu lại vết của con trỏ cha, lần mò từng con trỏ này sẽ giúp ta quay trở lại điểm xuất phát. Và đây cũng chính là lời giải của mê cung đó

Hệ tọa độ:

Tham khảo cách mà thầy trợ giảng hướng dẫn, nên ta sẽ coi hệ tọa độ sau dùng để định vị một mê cung (hay còn gọi là ma trận):




```
Starting point (x, y) = (8, 15)
Ending point (x, y) = (4, 0)
Bonus point at position (x, y) = (3, 6) with point -3
Bonus point at position (x, y) = (5, 14) with point -1
```

Hướng bản đồ: North
West East
South

Các mê cung sẽ sử dụng trong bài này:

(hình ảnh tất cả các mê cung kèm tọa độ start, end, bonus_point)

Xử lý lặp vô tận:

Trong tất cả các thuật toán tìm kiếm ở bên dưới, nếu không có phương pháp xử lí thích hợp, thì sẽ dẫn đến lặp vô tận (giống như khi ta lái xe vào bùng binh, cứ chạy mãi xung quanh cái bùng binh đó), ta đề xuất các cách xử lý như sau:

- Khi mở rộng một node, node mới sẽ không trùng với node cha trước đó.

- Mở rộng ý trên, node mới sau khi mở rộng sẽ không trùng với tất cả các node xét từ gốc đến node mới đó.
- Vẫn xảy ra lặp vô tận với cách xử lý trên. Node mới sẽ thêm vào một danh sách lưu trữ, những node mới xuất hiện sau đó sẽ tính là hợp lệ nếu không trùng với bất cứ node nào trong danh sách lưu trữ. Khắc phục được tình trạng tránh mở node mới khi mà node đó đã tồn tại trên nhánh khác của cây.

Thuật toán tìm kiếm không có thông tin

❖ Thuật toán tìm kiếm DFS (Depth-first search)

Ý tưởng: Mô phỏng lại hành vi của con người, khi cho một người vào mê cung, anh ta cứ chọn một đường mà đi, nếu là ngõ cụt, anh ấy quay đầu lại và chọn hướng khác.
Tức là: luôn luôn mở rộng nút nằm sâu nhất trong nhánh đang xét, tức là mở rộng theo một hướng duy nhất. Nếu không thể mở rộng được nữa, ta quay lại node cha để mở rộng tiếp, và cứ như thế cho đến khi tìm được lời giải.

Giả sử mỗi node sau khi được mở rộng có 4 tối đa node con (North, South, East, West), ở độ sâu d (depth), thì chi phí bộ nhớ tối đa sẽ là 4*d + 1 cần phải chứa khi chạy đệ quy. Trong đó, ta cộng 1 do tính thêm node gốc.

Mã giả: (không bao gồm cách xử lý lặp vô tận)

```
Function DFS( ):
        stack(node_gốc)
        Return Đệ_quy(stack)

Function Đệ_quy(stack):
        if là_rỗng(queue)
                then return Không_tìm_được_lời_giải
        node ← Lấy_phần_tử_cuối(queue)
        if node = Đích
                then return node
        for mỗi_node trong Mở_rộng(node):
                Thêm_vào_cuối(queue, mỗi_node)
                kết_quả ← Đệ_quy(stack)
                if kết_quả != False:
                        return kết_quả
        return False
```

Dự đoán nhược điểm có thể xảy ra:

Nếu lời giải nằm sát node gốc, mà ta đi theo hướng khác, thì ta cứ 'nhắm mắt' mà đi sâu xuống dưới, mất rất nhiều thời gian, để rồi khi không tìm thấy, ta phải lặn lội từ dưới lên lại node gốc để đi theo hướng khác (giống như đi lại từ đầu vậy!) → lệ thuộc hoàn toàn hướng (node) mà ta chọn, với cách chọn khác nhau dẫn đến thời gian giải khác nhau

Dự đoán ưu điểm có thể xảy ra:

Đối với bài toán có nhiều lời giải (chẳng hạn như mê cung mà ta đang làm), DFS có thể nhanh hơn so với BFS về mặt thời gian, lý do ở đây là vì DFS chỉ mở rộng một số lượng 'nhỏ' các node. Trong khi BFS phải duyệt qua hết tất cả đường đi ở độ sâu d để rồi mới duyệt đến độ sâu d+1

❖ Thuật toán tìm kiếm theo chiều rộng BFS (Breadth-first search)

Ý tưởng: Node gốc sẽ được mở rộng đầu tiên, các node con có được từ node gốc này sẽ tiếp tục mở rộng, tiếp theo các node này lần lượt được mở rộng, cứ như thế và tiếp tục.

Gọi là tìm kiếm theo 'chiều rộng' vì tất cả các node ở độ sâu d sẽ được mở rộng trước các node ở độ sâu d+1. Hay nói cách khác là phủ toàn bộ các nhánh, trước khi chiều cao của cây tăng thêm.
Cách xử lý lặp vô tận như đã nêu ở trên.
Giả sử mỗi node sau khi được mở rộng có 4 tối đa node con (North, South, East, West), ta có bộ nhớ cần lưu trữ tối đa cho độ sâu d là 4^d.
Nếu so sánh với DFS, thì BFS cần nhiều bộ nhớ hơn về mặt lý thuyết.
Mã giả: (không bao gồm cách xử lý lặp vô tận)

```
function BFS( ):
        queue(Node_gốc)
        loop:
                if là_rỗng(queue)
                        then return Không_tìm_được_lời_giải
                node ← Lấy_phần_tử_đầu(queue)
                if node = Đích
                        then return node
                for mỗi_node trong Mở_rộng(node):
                        Thêm_vào_cuối(queue, mỗi_node)
```

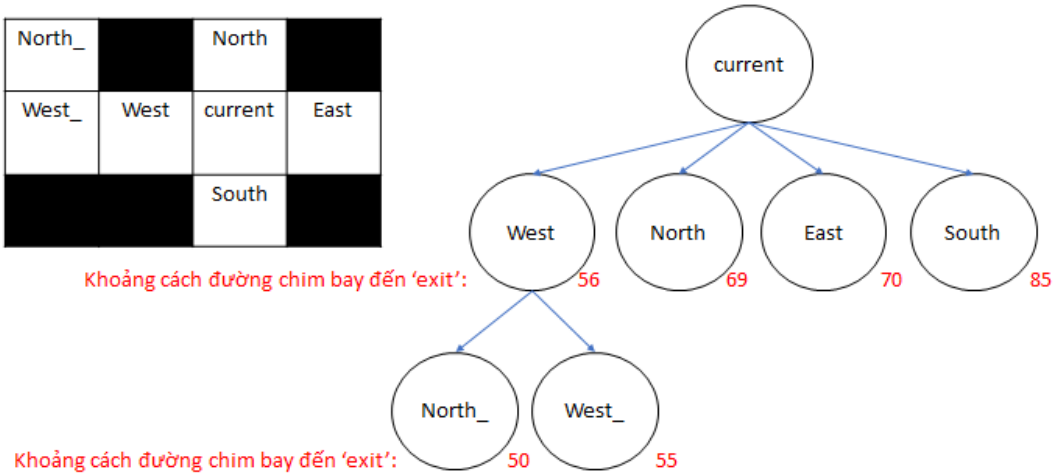
Dự đoán nhược điểm có thể xảy ra:
Thời gian chạy lâu do phải duyệt hết từng node trong cùng một độ sâu.
Yêu cầu không gian bộ nhớ lớn nếu giải mê cung lớn.
Dự đoán ưu điểm có thể xảy ra:
Có thể tìm thấy đường đi ngắn nhất

Thuật toán tìm kiếm có thông tin

❖ Thuật toán tìm kiếm tham lam (Greedy Best-first search):

Bản đồ không có điểm thưởng:
Ý tưởng: Ưu tiên mở rộng nút gần exit nhất. Sử dụng ý tưởng của DFS, Greedy Best-first search mở rộng hơn bằng cách xếp các node mới theo thứ tự độ ưu tiên từ trái sang. Node trái nhất sẽ được mở theo chiều sâu.
Hàm heuristic ở đây ta chọn là f(n) = h(n), trong đó h(n) là khoảng cách đường chim bay tính từ vị trí đang xét đến exit. Nên chi phí thực sự sẽ lớn hơn h(n). Từ đó cách sắp xếp các node mới sau khi mở rộng như đã nói ở trên, sẽ sắp xếp theo khoảng cách tăng dần từ node đó đến exit.

Hình ảnh minh họa:



Tại sao lại phải sắp xếp các node: Vì các node này sẽ được đưa vào stack theo thứ tự. Ta cần phải định nghĩa thứ tự này đúng như hàm heuristic đã đề ra.
Trong phần này, với mỗi heuristic khác nhau sẽ dẫn đến cách sắp xếp khác nhau.
Bản đồ có điểm thưởng:
(Mỗi khi ăn một điểm thưởng, ta loại bỏ điểm thưởng đó)
Hàm heuristic_1[(1)]:
Ý tưởng: Tưởng tượng ta có một chú chuột tham lam được thả vào mê cung, chú sẽ xem thức ăn (bonus_points) và exit là như nhau. Chú bắt đầu đánh hơi và đi đến điểm gần nhất. Nếu trên đường đi, chú chuột này thấy điểm nào gần hơn so với điểm trước đó, chú chuột sẽ tập trung vào đi đến điểm mới đó.
Cách thực hiện:

- Tính khoảng cách từ bonus_points, end đến current (current là node hiện tại)
- Chọn khoảng cách nhỏ nhất làm mục tiêu hướng đến. Khi mở rộng, ta ưu tiên chọn đi sâu vào nhánh để hướng gần đến mục tiêu này. Nếu có nhiều giá trị khoảng cách bằng nhau, ưu tiên chọn khoảng cách đường chim bay đến exit là ngắn nhất.
- Trong quá trình di chuyển, ta lặp lại 2 bước trên cho tới khi tới đích

Trường hợp xấu:
Các bonus_points sẽ kéo chú chuột này ra xa exit. Tưởng tượng như exit nằm ở hướng bắc, nhưng bonus_points nằm ở Nam và gần hơn, ta cứ mãi đi về hướng Nam làm cho khoảng cách đường chim bay đến exit dài ra.
Trường hợp tốt:
Mỗi khi tìm cách ăn bonus_points, ta lại gần exit hơn.

Nếu ta gần exit hơn so với bonus_points, thì node_mở_rộng sẽ di chuyển theo GBFS

Hàm heuristic_2[(2)]:

Ý tưởng: Tưởng tượng bonus_points và exit là như nhau và là những nam châm. Những nam châm này sẽ hút ta đến. Giả sử phía sau lưng ta có nhiều nam châm, phía trước mặt chỉ có một, thì ta sẽ đi về hướng sau lưng.
Ở đây, ta xét trên các node_mở_rộng, không xét node current. Ta ưu tiên chọn node_mở_rộng có "tổng giá trị khoảng cách đến các nam châm" là *nhỏ nhất* (total_distance is lowest)
Cách thực hiện:

- Tính total_distance cho từng node_mở_rộng
- Chọn total_distance nhỏ nhất. Nếu có nhiều node có cùng giá trị total_distance thì chọn node có khoảng cách đường chim bay đến exit là ngắn nhất.
- Trong quá trình di chuyển, ta lặp lại 2 bước trên cho tới khi tới đích.

Hàm heuristic_3$^{(3)}$:
Ý tưởng: Ta sẽ di chuyển từ start đến exit như bình thường. Trên đường di chuyển, nếu thấy khoảng cách đường chim bay từ current đến bonus_point "*nhỏ hơn"* giá trị của bonus_point đó, thì ta sẽ chọn node_mở_rộng gần nhất với bonus_point đó, mục tiêu là ăn điểm.
Còn nếu lớn hơn hay nói cách khác là không đáng để ăn thì ta tiếp tục đi đến exit như cũ.

Cách thực hiện:

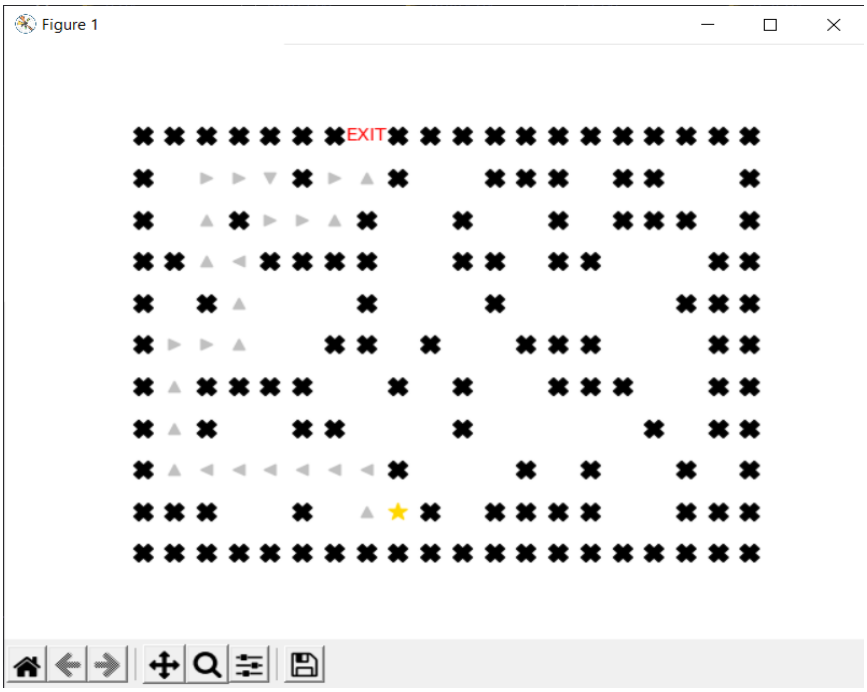
- Chọn những bonus_points mà khoảng cách đường chim bay từ current đến điểm này nhỏ hơn giá trị mà nó đang giữ.
- Lưu những bonus_points này vào một danh sách: "subset_of_bp" (danh sách này là tập con (subset) của danh sách bonus_points ban đầu).
- Nếu subset_of_bp là rỗng, ta dùng GBFS như là không có bonus_points, để di chuyển đến node tiếp theo và lặp lại 2 bước trên
- Nếu subset_of_bp không rỗng, ta ưu tiên chọn bonus_points gần nhất và đặt mục tiêu di chuyển đến nó như là **heuristic_1**
- Thuật toán dừng lại khi đến được đích

Vận dụng và so sánh:

Bản đồ không có điểm thưởng:

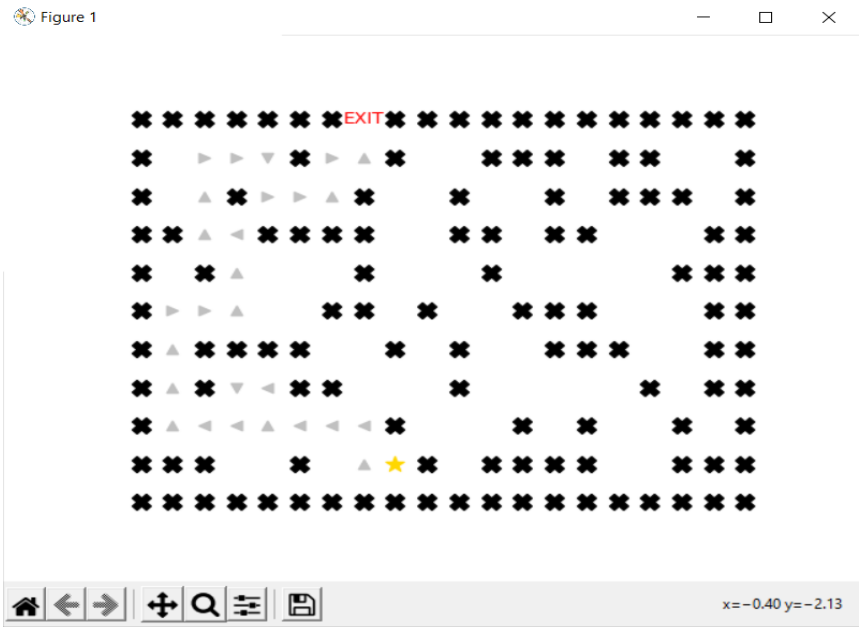
Bản đồ 1 (map1):

- *breath first search*
  - Tìm kiếm theo chiều rộng, **và chọn thứ tự ưu tiên trên, dưới, trái ,phải**
  - Tìm kiếm theo chiều sau tìm ra đường đi có số bước biến đổi ít nhất và cũng là đường đi ngắn nhất trong bài toán này vì chi phí cho một lần biến đổi là như nhau
  - step = 74 (có trong chảy chương trình và thể hiện số node đã duyệt để tiềm thấy lối ra)
  - Đường đi ngắn nhất
  - T = 0.001s (thời gian thực hiện)



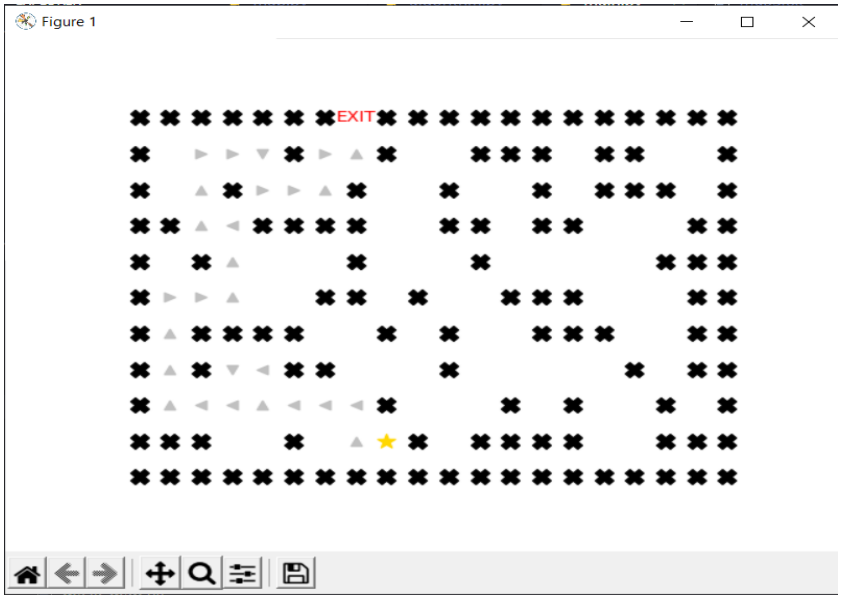
❖ *Depth first search*

- Tiềm kiếm theo chiều sau, **và chọn thứ tự ưu tiên trên, dưới, trái ,phải**
- Step = 71
- Đường đi khá tốt do lối ra nằm trên điểm suất phát (độ ưu tiên thứ nhất)
- T = 0.00103s



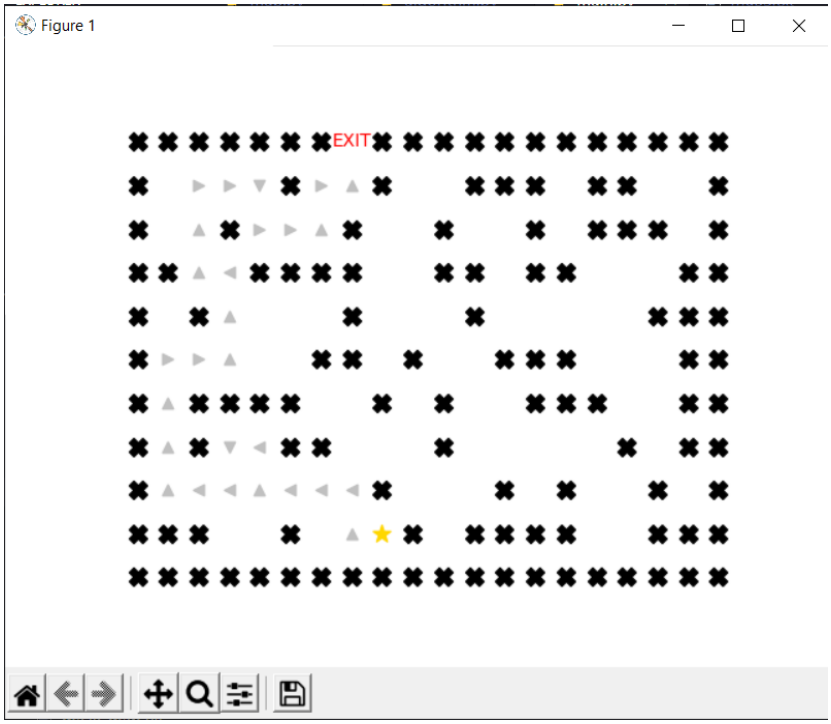
❖ *Greedy best first search*

- Thuật toán tìm kiếm tham lam, bài toán tiềm kiếm heuristic là khoảng cách đường chim bay từ node đang xử lý tới lối ra
- step = 67
- Đường đi khá tốt
- T = 0.002s



❖ *A* search*

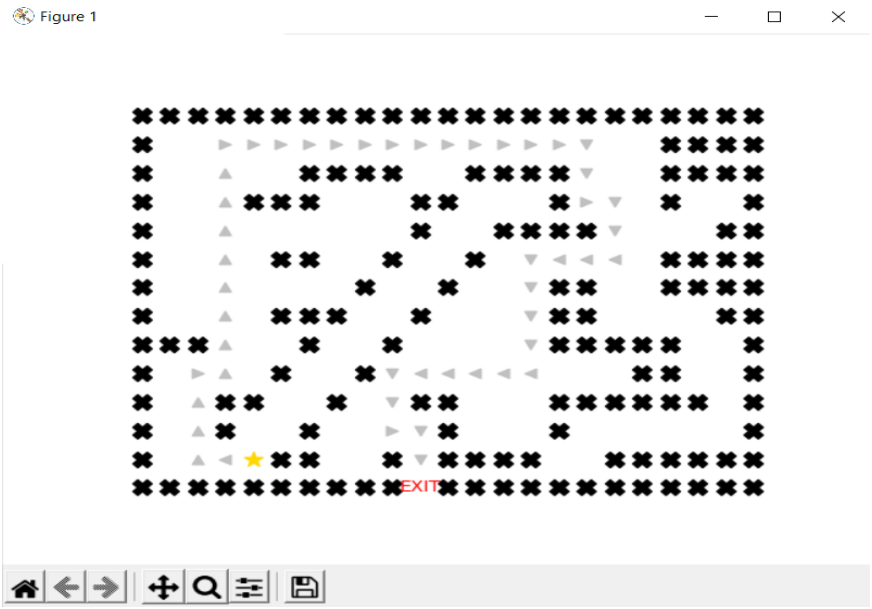
- Thuật toán heuristic là khoảng cách đường chim bay cộng với chi phi đi giữa các node (trong bài toán này thì A* search sẽ có kết qua gióng với greedy best first search do chi phi đi giữa các node bằng nhau)
- Step = 67
- T = 0.00103s



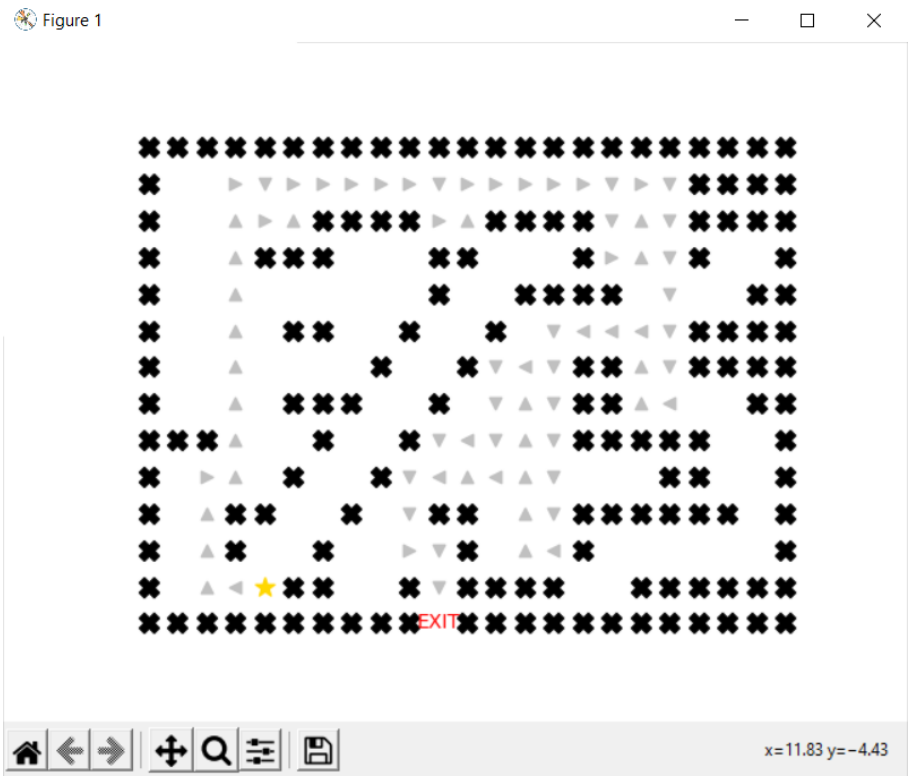
=> Đánh giá về bản đồ 1 nhìn chung kết quả khá gióng nhau và đường đi tốt nhất là BFS.

Bản đồ 2 (map2):

- *breath first search*
  - step = 165
  - Đường đi tốt nhất
  - Chi phi đường đi cao
  - T = 0.00901s



- *Depth first search*
  - Step = 116
  - Đường không tốt do đích ở phí dưới điểm xuất phát.
  - Chi phí trung bình.
  - T = 0.00404s



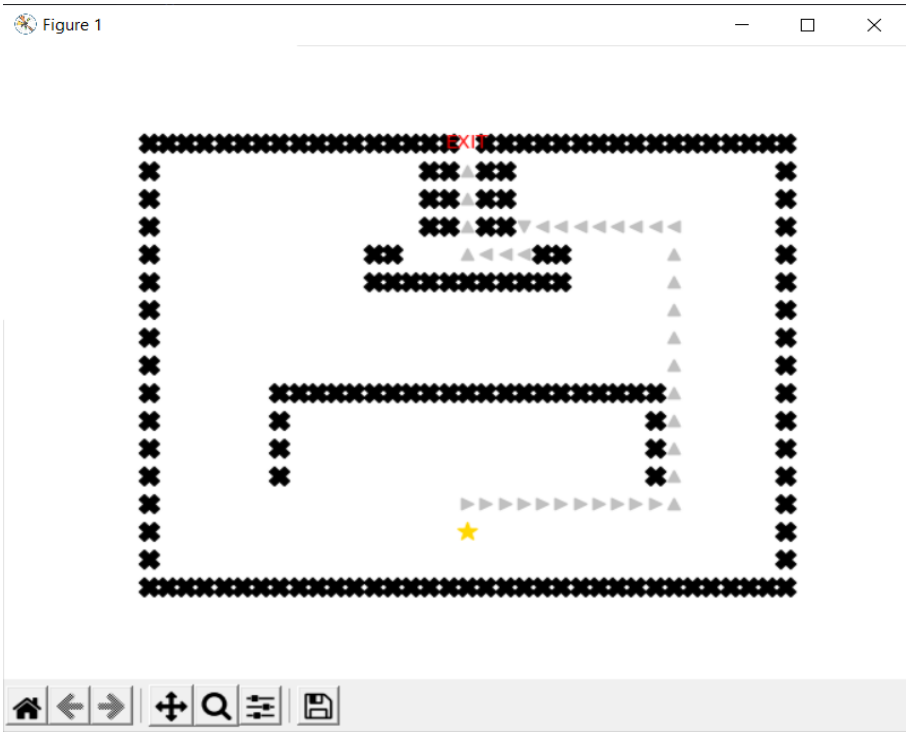
- ❖ *Greedy best first search và A* search*
  - Step = 140
  - Đường đi không tốt
  - Chi phí cao
  - T = 0.00804(GBFS) và T = 0.00799 (A*)

=> Đánh giá về bản đồ 2 nhìn đường đi của BFS, GBFS và A* không tốt, đường đi tốt nhất BFS

Bản đồ 3 (map3):

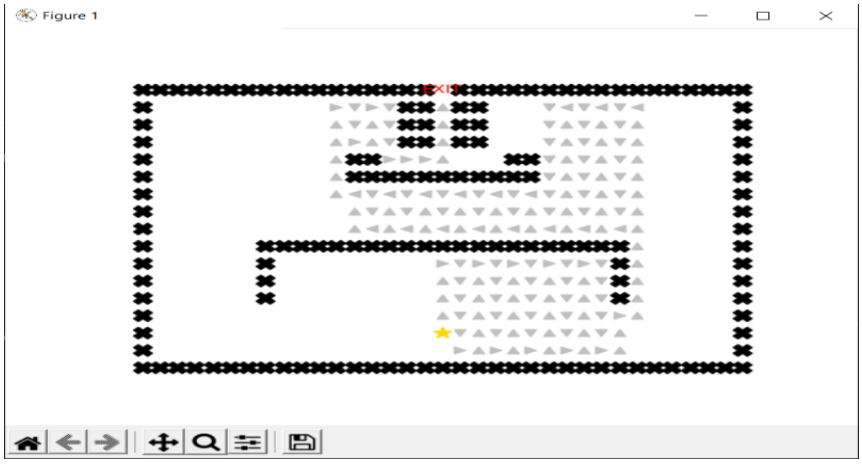
- ❖ ***breath first search***

- step = 442
- Đường đi tương đối tốt nhưng về mặt chi phí thì rất nhiều
- Chi phí cao
- T = 0.0858s



❖ *Depth first search*

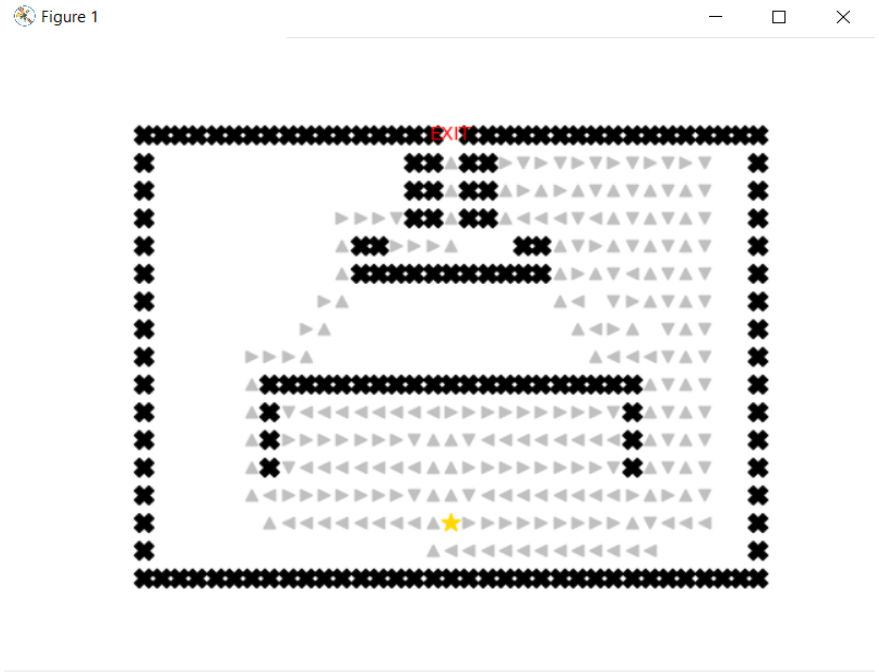
- *Step = 305*
- *Đường đi không tốt*
- Chi phí thấp
- T = 0.03796s



❖ *Greedy best first search và A* search*

- Step = 455
- Đường đi không tốt

- Chi phí cao
- T = 0.08982s (GBFS) và T = 0.08584s (A*)



=> Đánh giá bản đồ 3 DFS, GBFS và A* cho đường đi không tốt nhưng DFS lại có chi phí thấp hơn, BFS cho đường đi tốt nhất

Bản đồ 4 (map4):

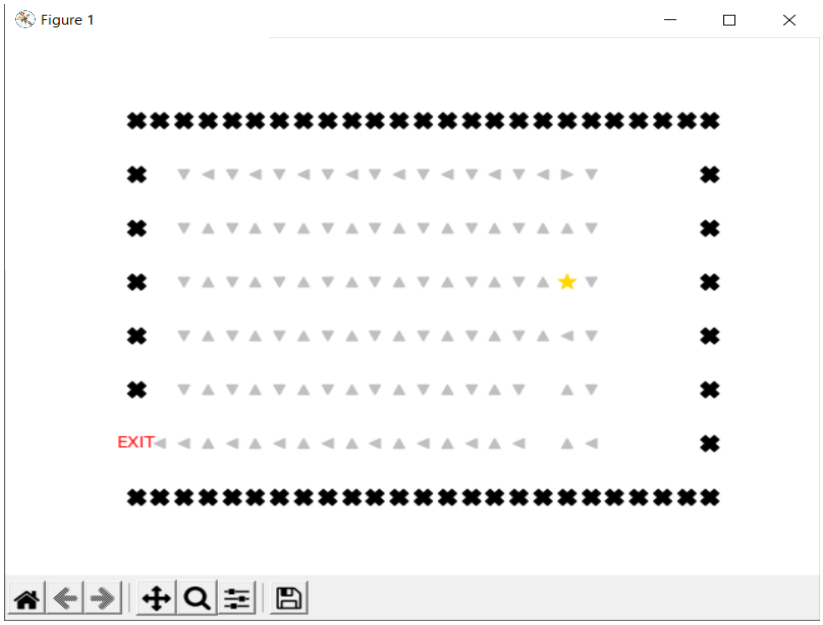
❖ *Breath first search*

- Step = 139
- Đường đi tốt
- Chi phí thấp
- T = 0.00901s

❖ *Depth first search*

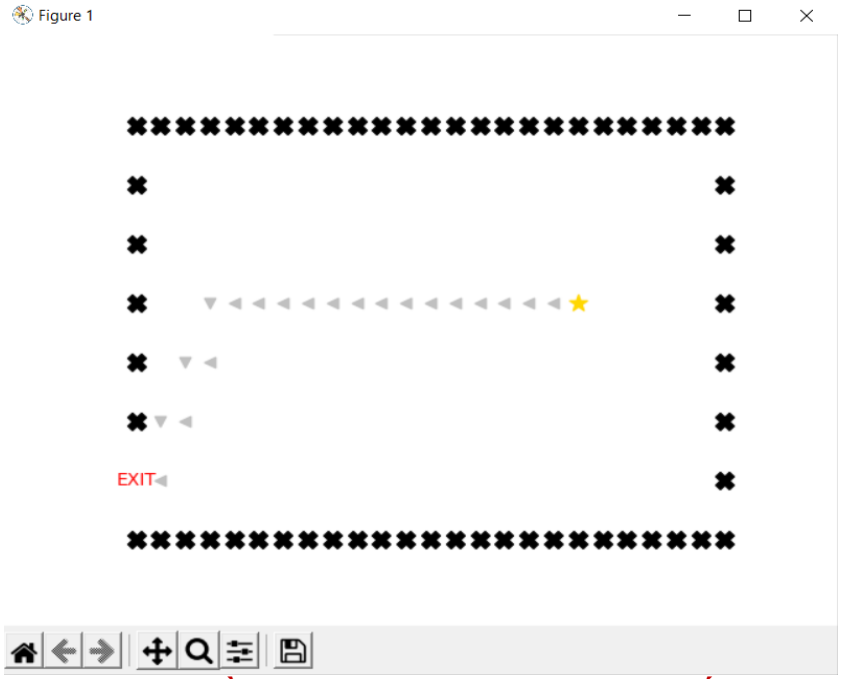
- Step = 154
- Đường đi xấu nhất đối với DFS
- Chi phí cao

- T = 0.01097s



❖ *Greedy best first search và A* search*

- Step = 22
- Đường đi tốt
- Chi phí thấp
- T ≈ 0 (đối với cả 2 trường hợp)

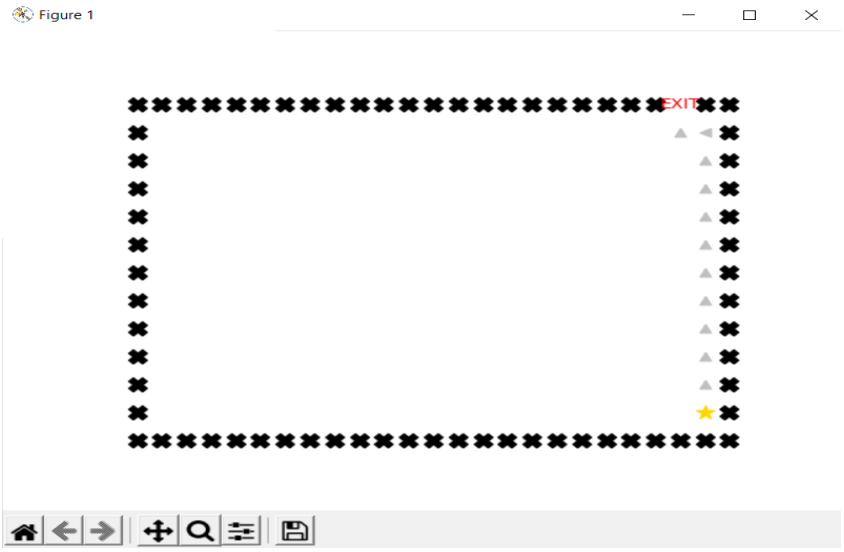


=> Đánh giá bản đồ 4 DFS rơi vào trường hợp sáu nhất đường đi sau và chí phí cao, BFS, GBFS và A* có đường đi tốt nhưng về mặt chi phí thì BFS có chi phí rất cao

Bản đồ 5 (map5):

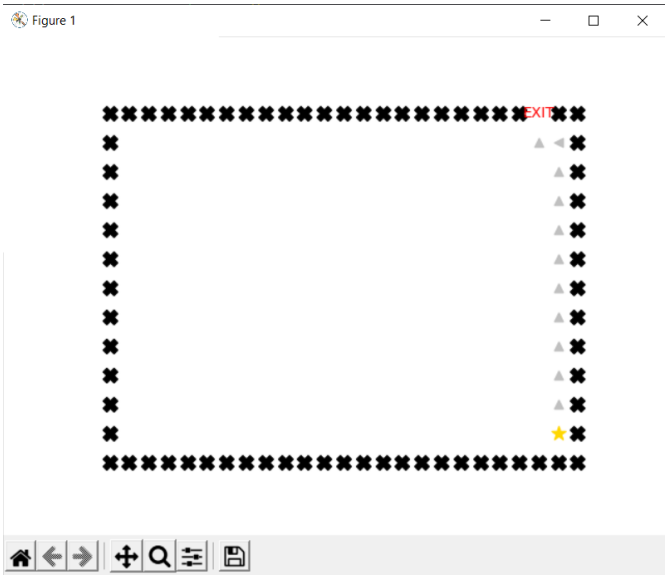
❖ *Breath first search*

- Step = 89
- Đường đi tốt
- Chi phí cao
- T = 0.00303



❖ *Depth Frist search*

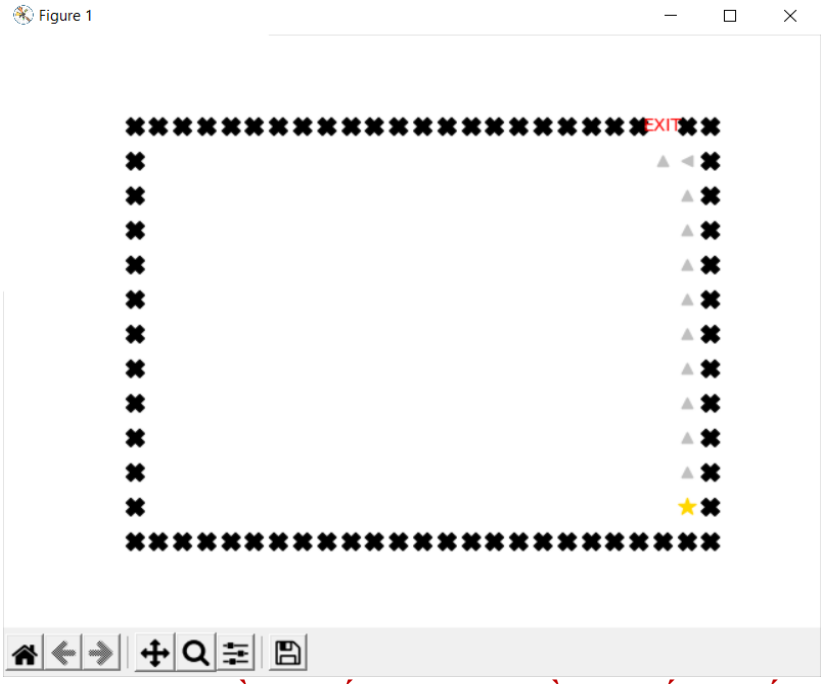
- Step = 13
- Đường đi này có step ngắn nhất do lối ra nằm ngay phía trên điểm xuất phát nên kết quả tốt nhất
- Chi phí thấp
- T ≈ 0



❖ *Greedy best firs search and A* search*

- Step = 13
- Đường đi tốt
- Chi phí thấp

- T ≈ 0



=> Đánh giá bản đồ 5 cả bốn thuật toán đều cho kết quả tốt nhưng ở BFS có chi phí cao còn DFS(trường hợp tối ưu nhất), GBFS, A* có chi phí thấp

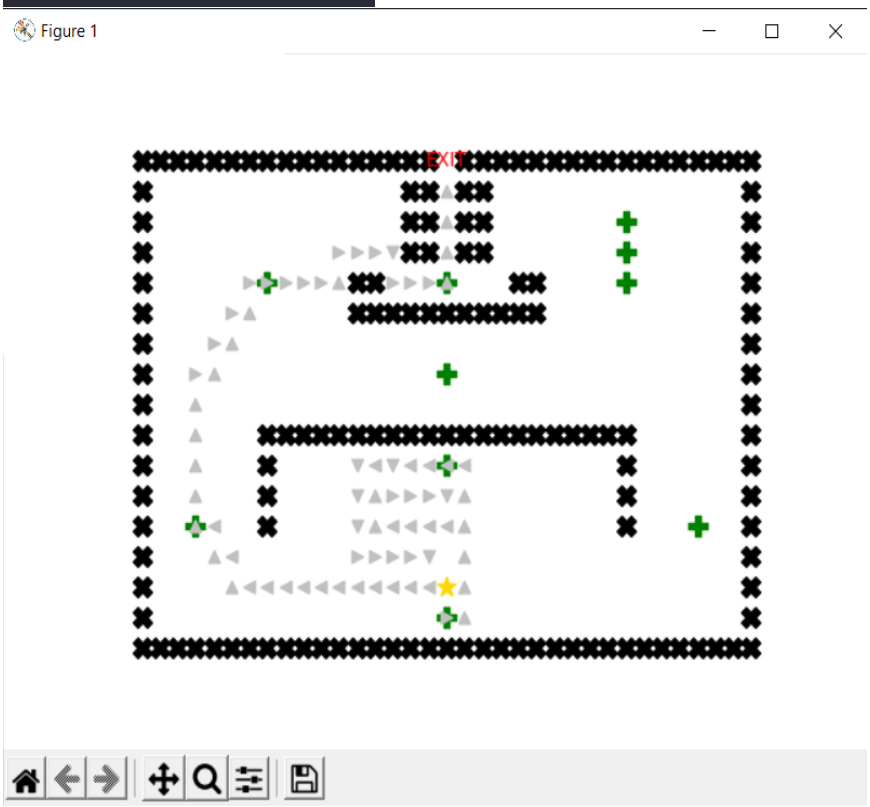
- ***Tổng kết trong bài toán*: BFS cho đường đi tốt nhất nhưng lại thường có chi phí cao, DFS kết quả bài toán thường phụ thuộc vào vị trí suất phát và lối ra không ổn định như BFS, GBFS và A* sẽ tìm ra đường đi tối ưu nếu vị trí xuất phát tới lối ra ít bị chặn bởi tường (bị chặn tại những vị trí gần lối ra)**

Bản đồ có điểm thưởng (sử dụng Greedy best first search):

Bản đồ 6 (map6):

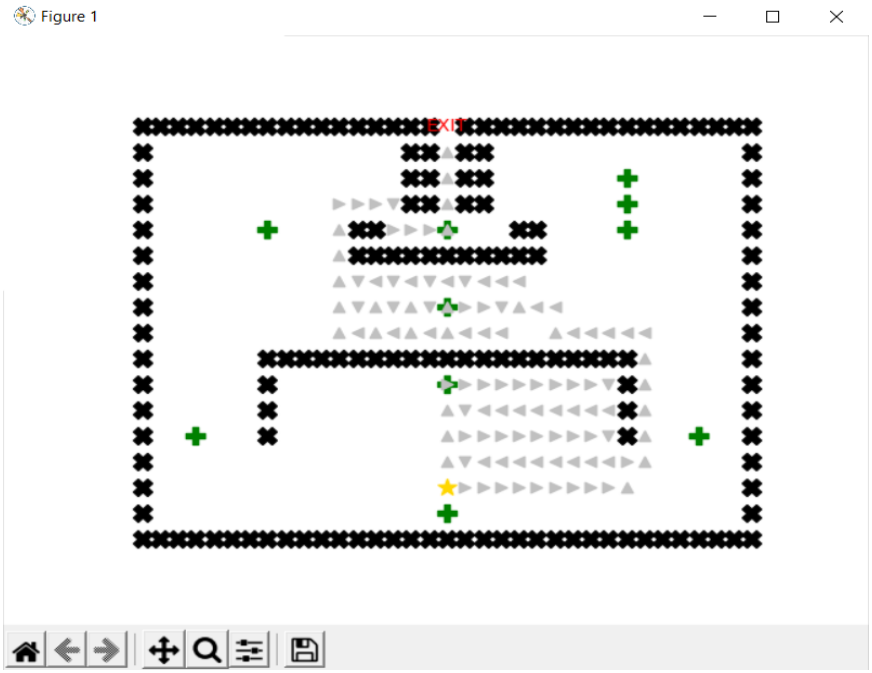
- *Heuristic 1:*
  - Vì có không được đi lại điểm đã đi qua và có thứ tự ưu tiên khác nhau nên đường đi có hướng qua trái(

```
Points collected:
(15, 17), value: 3
(10, 17), value: 1
(12, 3), value: 1
(4, 7), value: 7
(4, 17), value: 3
Total:  15
```



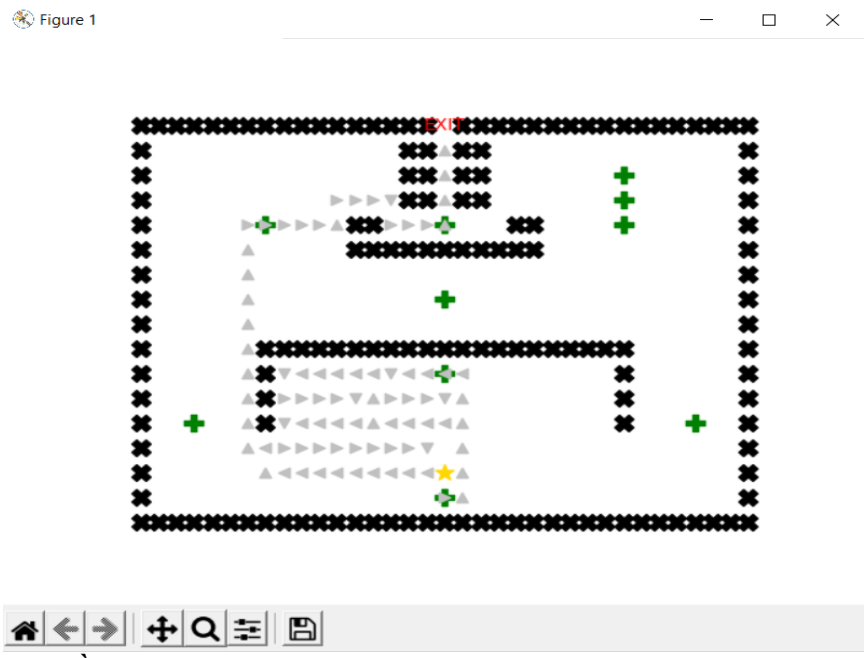
- Heuristic 2:
  - Có xu hướng đi về phía có nhiều điểm thưởng nên đường đi có xu hướng lệch sang bên trái

```
Points collected:
(10, 17), value: 1
(7, 17), value: 5
(4, 17), value: 3
Total:  9
```



- *Heuristic 3:*
  - Tiềm kiếm đường đi như GBFSs nhưng trên đường đi kiếm thấy có điểm thưởng có lợi cho đường đi thì sẽ lấy

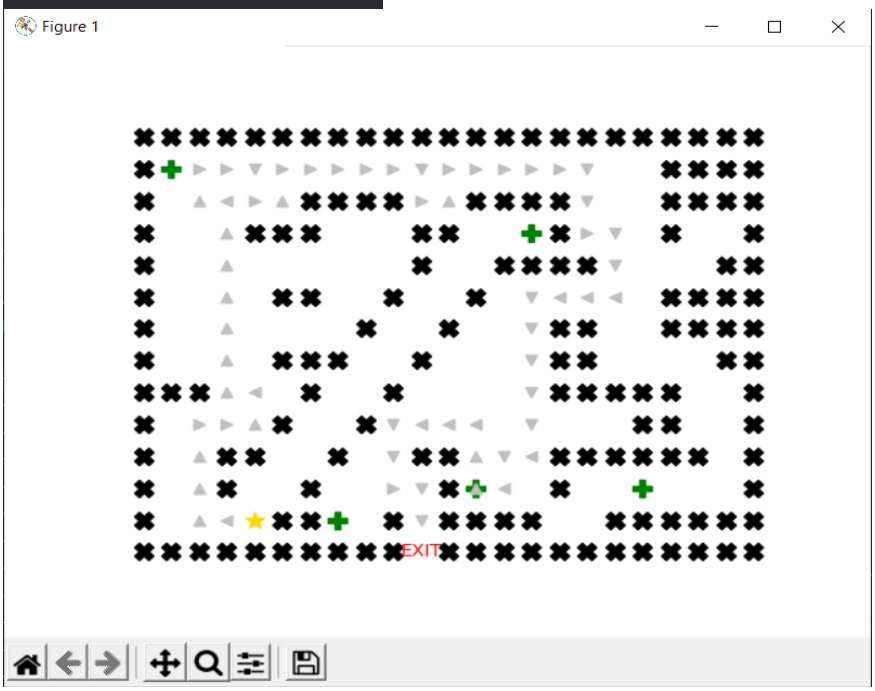
```
Points collected:
(15, 17), value: 3
(10, 17), value: 1
(4, 7), value: 7
(4, 17), value: 3
Total:  14
```



Bản đồ 7 (map7):

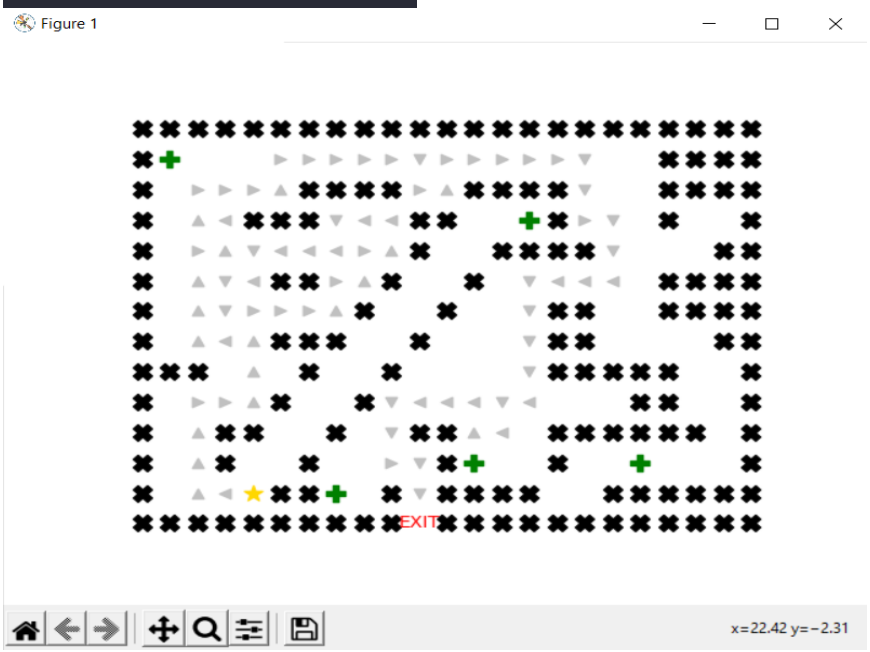
- *Heuristic 1:*

```
Points collected:
(11, 12), value: 3
Total:  3
```



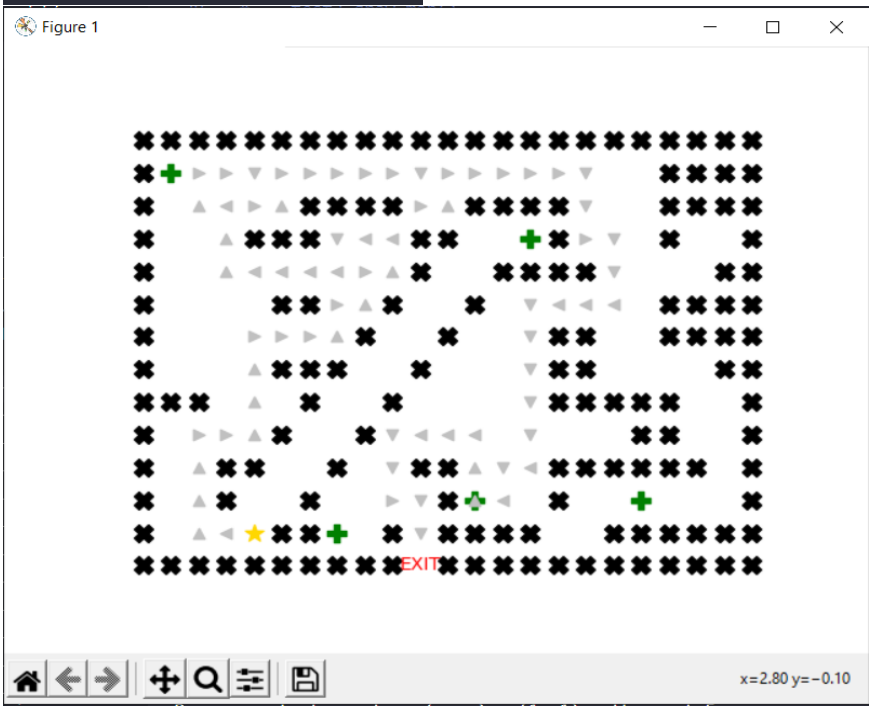
- *Heuristic 2:*

```
Points collected:
Total:  0
```



❖ *Heuristic 3:*

```
Points collected:
(11, 12), value: 3
Total:  3
```



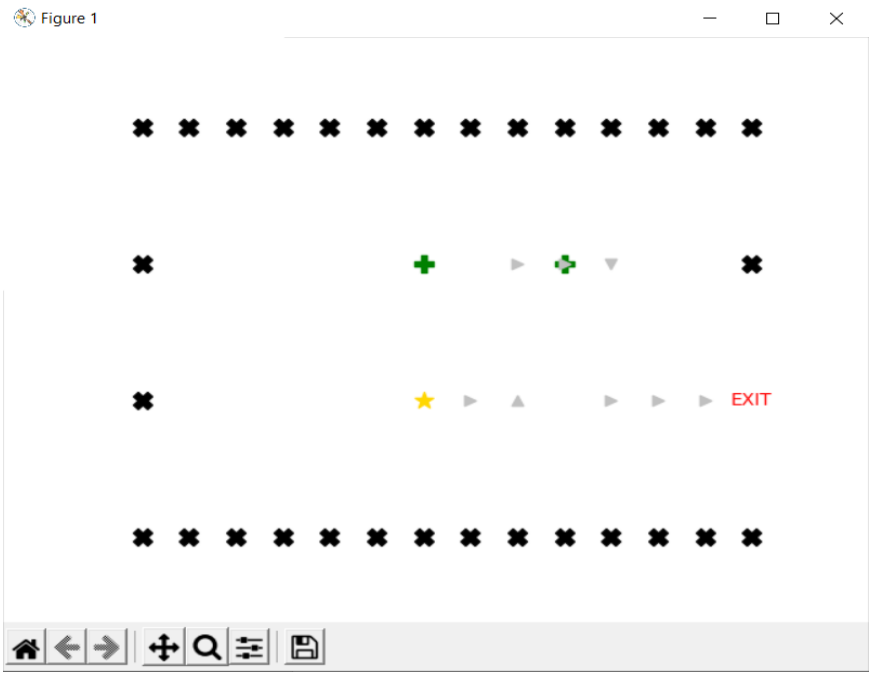
Bản đồ 8 (map8):

- *Heuristic 1:*

```
Points collected:
(1, 6), value: 15
(1, 9), value: 9
Total:  24
```

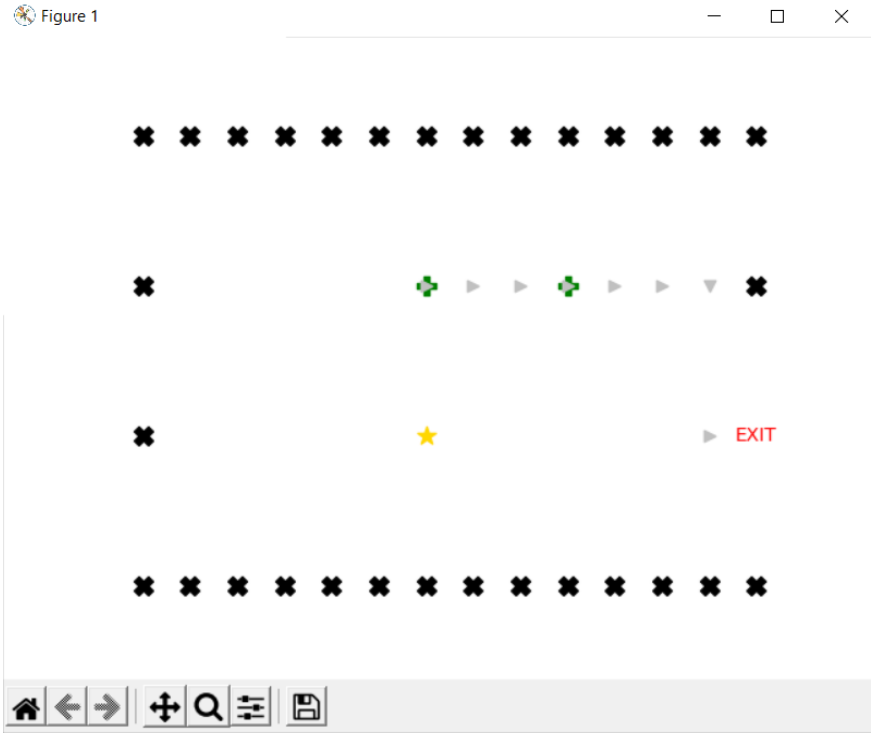
- *Heuristic 2:*

```
Points collected:
(1, 9), value: 9
Total:  9
```



- *Heuristic 3:*

```
Points collected:
(1, 6), value: 15
(1, 9), value: 9
Total:  24
```



- ***Tổng kết bài toán*** : Heuristic 1 tối ưu khi các vị trí điểm thưởng đi qua làm cho đường đi hướng về phía lối ra và không tốt khi vị trí điểm thưởng làm đường đi càn ngày càng đi xa lối ra , Heuristic 2 tối ưu khi số điểm thượng nằm lệch về 1 phía và không tối ưu nếu điểm thưởng rời rạc mạnh, Heuristic 3 tối ưu khi các điểm thưởng nằm gần đường đi đến lối ra và không tối ưu khi điểm thưởng nằm xa đường đi đến lối ra.

Tài liệu tham khảo:

Sách Cơ sở trí tuệ nhân tạo (Tác giả: Lê Hoài Bắc, Tô Hoài Việt, xuất bản 2014): bảng 1-2 trang 34. Trang 36 dòng 5: cách tính số nút tối đa của DFSMột số hàm xử lý hình vẽ mê cung của thầy Nguyễn Khánh Toàn, Lê Minh Nhật (link: https://colab.research.google.com/drive/1owoPjKdszirH1SSeBviJsYhDJqlS5Ndq?usp=sharing)