ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

BÁO CÁO ĐỒ ÁN
CÁC THUẬT TOÁN TÌM KIẾM

Môn học: Cơ sở trí tuệ nhân tạo

Giáo viên hướng dẫn:	Lê Hoài Bắc Nguyễn Duy Khánh
Sinh viên thực hiện:	Bùi Quốc Trung - 20120023 Dũ Quốc Huy - 20120101 Nguyễn Kông Đại - 20120448

Thành phố Hồ Chí Minh, tháng 10 năm 2022

Mục Lục

1 Đánh giá mức độ hoàn thành 2

2 Mô tả thuật toán 2
2.1 Thuật toán DFS (Depth First Search) 2
2.2 Thuật toán BFS (Breadth First Search) 3
2.3 Thuật toán UCS (Uniform-Cost Search) 3
2.4 Thuật toán GBFS (Greedy Best First Search) 4
2.5 Thuật toán A* 4
2.6 Thuật toán đề xuất cho bản đồ có điểm thưởng 5
2.7 Thuật toán đề xuất cho bản đồ có điểm đón 5

3 Kết quả 5
3.1 Bản đồ không có điểm thưởng 5
3.1.1 Bản đồ 1: 16 x 36 5
3.1.2 Bản đồ 2: 17 x 40 8
3.1.3 Bản đồ 3: 11 x 22 10
3.1.4 Bản đồ 4: 16 x 36 11
3.1.5 Bản đồ 5: 20 x 40 13
3.2 Bản đồ có điểm thưởng 15
3.2.1 Bản đồ 2 điểm thưởng: 11 x 22 15
3.2.2 Bản đồ 5 điểm thưởng: 16 x 36 15
3.2.3 Bản đồ 10 điểm thưởng: 20 x 40 16
3.2.4 Đề xuất cải tiến 17
3.3 Bản đồ có điểm đón 17
3.3.1 Bản đồ 5 điểm đón: 16 x 36 17
3.3.2 Bản đồ 10 điểm đón: 20 x 40 18
3.3.3 Bản đồ 25 điểm đón: 20 x 48 19

4 Mô tả file run.sh 20

Tài liệu tham khảo 21

1. Đánh giá mức độ hoàn thành

Mức yêu cầu	Mức độ hoàn thành	Điểm tương ứng
Bản đồ không có điểm thưởng (mức 1a): thiết kế đủ 5 bản đồ và hoàn thành 3 thuật toán tìm kiếm không có thông tin	100%	5 điểm
Bản đồ không có điểm thưởng (mức 1b): hoàn thành tiếp 2 thuật toán tìm kiếm có thông tin	100%	2 điểm
Bản đồ có điểm thưởng (mức 2a): đề xuất thuật toán và các hàm heuristic phù hợp để giải quyết, thiết kế đủ 3 bản đồ và mô phỏng cách thuật toán chạy trên các bản đồ	100%	1 điểm
Bản đồ có điểm thưởng (mức 2b): cài đặt thành công thuật toán	100%	1 điểm
Bản đồ có điểm đón (mức 3): đề xuất thuật toán và cài đặt thành công trường hợp có các điểm đón với tối thiểu 3 bản đồ	100%	1 điểm
Điểm cộng: Kịch bản nâng cấp	0%	0 điểm
Điểm cộng: Video minh họa	100%	0.5 điểm
Tổng kết		10.5 điểm

2. Mô tả thuật toán

2.1. Thuật toán DFS (Depth First Search)

DFS là một thuật toán tìm kiếm không có thông tin. Là thuật toán tìm kiếm theo chiều sâu, tức là DFS sẽ bắt đầu từ điểm gốc, và tìm kiếm phát triển xa nhất với mỗi nhánh con. Thuật toán được cài đặt theo cấu trúc dữ liệu ngăn xếp (LIFO-Last In First Out):

1. Đưa điểm Start vào ngăn xếp.

2. Lấy phần tử trên cùng của ngăn xếp ra để duyệt: Ở vị trí hiện tại (ban đầu là Start), ta duyệt hết 4 vị trí xung quanh nó để xem điểm nào thể đi tới được mà chưa được duyệt và đưa vào ngăn xếp.
3. Trong quá trình đưa vào ngăn xếp, ta cần gán điểm đó là đã duyệt, lưu lại vị trí điểm đó và điểm đi được với nó (hỗ trợ việc vẽ hình và truy xuất đường đi trở lại). Nếu điểm đó là đích thì dừng quá trình duyệt, nếu không thì quay trở lại **Bước 2** để tiếp tục duyệt.
4. Quá trình duyệt sẽ dừng lại khi ngăn xếp trống.

2.2. Thuật toán BFS (Breadth First Search)

BFS là một thuật toán duyệt đồ thị theo chiều rộng (tìm kiếm không có thông tin). Là thuật toán tìm kiếm theo chiều rộng, bắt đầu từ điểm gốc, BFS sẽ lan dần đều ra xung quanh và nếu không có điều kiện dừng lại thì BFS sẽ duyệt hết bản đồ rồi mới dừng lại. Thuật toán BFS sẽ trả ra đường đi ngắn nhất có thể trong trường hợp đồ thị không có trọng số hoặc tất cả các trọng số bằng nhau.
Thuật toán được cài đặt theo cấu trúc dữ liệu hàng đợi (First in first out) như sau:

1. Đưa điểm Start vào hàng đợi.
2. Lấy phần tử đầu tiên của hàng đợi ra duyệt: duyệt tất cả các đỉnh kề với điểm được duyệt, tiếp đó kiểm tra xem đỉnh nào chưa được duyệt thì đưa chúng lần lượt vào hàng đợi.
3. Trong quá trình đưa vào hàng đợi, ta cần đánh giá điểm đó là đã duyệt, lưu lại vị trí điểm đó và điểm đi đến nó (điểm đang dùng để duyệt) (hỗ trợ việc vẽ hình và truy xuất đường đi trở lại). Quay trở lại **Bước 2** để tiếp tục duyệt.
4. Quá trình duyệt sẽ dừng lại khi hàng đợi trống.

2.3. Thuật toán UCS (Uniform-Cost Search)

UCS là thuật toán tìm kiếm không có thông tin. UCS là một cách duyệt cây dùng cho việc duyệt hay tìm kiếm một cây, cấu trúc cây, hoặc đồ thị có trọng lượng (chi phí). Việc tìm kiếm bắt đầu tại nút gốc. Việc tìm kiếm tiếp tục bằng cách duyệt các nút tiếp theo với trọng lượng hay chi phí thấp nhất tính từ nút gốc. Các nút được duyệt tiếp tục cho đến khi đến được nút đích cần đến. Đối với bản đồ chi phí đều, UCS tương đương với BFS. Nhìn chung UCS là thuật toán dijkstra. Thuật toán được cài đặt với cấu trúc dữ liệu là hàng đợi ưu tiên:

1. Đưa điểm Start vào hàng đợi
2. Lấy ra phần tử trên cùng của hàng đợi (có chi phí nhỏ nhất) để duyệt, nếu đó là điểm End thì dừng.
3. Tại điểm vừa lấy ra (ban đầu là Start), ta duyệt hết 4 vị trí xung quanh nó để xem điểm nào có thể đi tới được mà chưa được duyệt. Xét xem đường đi đến điểm đó có thấp nhất so với đường đi cũ không nếu có ta sẽ thêm nó vào hàng đợi để tiếp tục xét.
4. Trong quá trình đưa vào ngăn xếp, ta cần gán điểm đó là đã duyệt, lưu lại vị trí điểm đó và điểm đi được với nó (hỗ trợ việc vẽ hình và truy xuất đường đi trở lại). Quay trở lại **Bước 2** để tiếp tục duyệt.
5. Quá trình duyệt sẽ dừng lại khi ngăn xếp trống hoặc đã tìm được tới điểm End.

2.4. Thuật toán GBFS (Greedy Best First Search)

GBFS là một thuật toán tìm kiếm có thông tin. Bằng việc sử dụng một hàm **heuristic h(u)** để xác định được điểm nên đi tới tiếp theo. Thuật toán được nhóm cài đặt với việc sử dụng cấu trúc dữ liệu ngăn xếp (LIFO-Last In First Out):

1. Định nghĩa hàm **heuristic** phù hợp nhất
2. Đưa điểm Start vào ngăn xếp
3. Lấy ra phần tử trên cùng của ngăn xếp để duyệt, nếu đó là điểm End thì dừng.
4. Tại điểm vừa lấy ra (ban đầu là Start), ta duyệt hết 4 vị trí xung quanh nó để xem điểm nào có thể đi tới được mà chưa được duyệt. Sử dụng hàm **heuristic** để ước lượng khoảng cách các điểm đó đến đích và đưa vào ngăn xếp theo thứ tự từ khoảng cách xa nhất đến khoảng cách gần nhất (Điểm trên cùng ngăn xếp sẽ có ước lượng khoảng cách ngắn nhất).
5. Trong quá trình đưa vào ngăn xếp, ta cần gán điểm đó là đã duyệt, lưu lại vị trí điểm đó và điểm đi được với nó (hỗ trợ việc vẽ hình và truy xuất đường đi trở lại). Nếu điểm đó là đích thì dừng quá trình duyệt, nếu không thì quay trở lại **Bước 2** để tiếp tục duyệt.
6. Quá trình duyệt sẽ dừng lại khi ngăn xếp trống.

2.5. Thuật toán A*

A* là một thuật toán tìm kiếm có thông tin, thuật toán này yêu cầu định nghĩa một hàm heuristic **h(n)** phù hợp nhất và dựa vào đó cộng thêm với trọng số giữa các đỉnh trong đồ thị **g(n)** để xác định đường đi tối ưu nhất đến một đỉnh từ một đỉnh bất kỳ. Thuật toán sử dụng cấu trúc dữ liệu hàng đợi ưu tiên với giá trị ưu tiên là $\mathbf{f(n) = g(n) + h(n)}$ để cài đặt.

1. Định nghĩa hàm heuristic **h(n)** phù hợp nhất
2. Khởi tạo hàng đợi ưu tiên rỗng và đưa điểm Start vào hàng đợi
3. Lấy phần tử đầu tiên của hàng đợi ưu tiên ra duyệt, nếu nó là điểm End thì đánh dấu đã đến và dừng
4. Xét điểm vừa lấy ra, duyệt các điểm kề với nó, so sánh và cập nhật lại **g(n)**(nếu giá trị mới nhỏ hơn giá trị cũ), sau đó tính toán **f(n)** và đưa giá trị này cũng như điểm đang duyệt đến vào hàng đợi ưu tiên, sau đó lưu lại đường đi đến nó (điểm vừa được lấy từ hàng đợi ra duyệt)
5. Sau khi duyệt xong các điểm kề với điểm đang xét, đánh dấu là điểm này thuộc đường đi và quay lại **Bước 2**
6. Quá trình này sẽ dừng lại khi đã duyệt hết các giá trị trong hàng đợi ưu tiên hoặc đã đi tới điểm End ở bước 2.

2.6. Thuật toán đề xuất cho bản đồ có điểm thưởng

Thuật toán mà nhóm lựa chọn để sử dụng cho bản đồ có điểm thưởng là duyệt tham lam nhưng sẽ kèm theo điều kiện. Cụ thể như sau:

1. Ta sẽ sắp xếp lại các giá trị trong danh sách điểm thưởng theo thứ tự khoảng cách với điểm bắt đầu tăng dần
2. Đưa điểm Start vào vòng lặp, tính giá trị **point** là khoảng cách từ điểm Start đến điểm End
3. Tính toán giá trị **f(n) = g(n) + h(n) + b** cho tất cả các điểm thưởng, trong đó g(n) là khoảng cách từ điểm bắt đầu đến điểm thưởng, h(n) là khoảng cách từ điểm thưởng đến điểm End, b là giá trị của điểm thưởng tại vị trí điểm thưởng đó
4. Lấy giá trị **f(n)** nhỏ nhất ra ngoài và so sánh với point, nếu lớn hơn thì áp dụng thuật toán **A*** tìm đường đi từ Start đến End và kết thúc, ngược lại thì gán point = f(n)
5. Dùng thuật toán **A*** tìm đường đi từ Start đến điểm thưởng đang giữ giá trị **f(n)** nhỏ nhất và gán **Start = vị trí điểm thưởng đó** và quay lại **Bước 2**
6. Quá trình này sẽ dừng lại khi đã đi được đến End hoặc trả về **None** nếu không tìm được đường đến đó.

2.7. Thuật toán đề xuất cho bản đồ có điểm đón

Ta sẽ duyệt tham lam trên danh sách điểm đón. Đầu tiên ta xét điểm gần với điểm start nhất để đi dựa vào hàm Heurictic. Cứ thế từ điểm đón đang xét ta sẽ tìm điểm đón tiếp theo chưa được xét và có heurictis từ điểm đang xét đến điểm đón đó thấp nhất để ta đi tới. Thuật toán để tìm đường đi giữa 2 điểm này ta sẽ chọn A_star. và cuối cùng là ta sẽ đi từ điểm đón cuối cùng về điểm END. Ở đây thì, nhóm em cho phép là có thể đi qua lại những điểm đã đi qua. Và mỗi lần đi giã 2 điểm ta sẽ lưu lại đường đi để có đường đi cuối cùng.

Hạn chế ở đây là nếu như các điểm đón gần start sau đó các điểm đón tiếp theo đi về phía end nhưng có một số điểm đón nó ở xa start và end thì thuật toán của chúng ta khi gần đến End thì phải quay lại đón các điểm đó và quay về nên phải chịu chi phí lớn.

Giải pháp cải tiến đề ra ở đây là: đầu tiên ta cần tính khoảng cách giữa các điểm đón với nhau và 2 điểm start and end. Để tính khoảng cách này ta dùng thuật BFS từ start và loang hết bản đồ. Sau khi tình khoảng cách ta sẽ tiến hành tìm đường đi từ điểm đầu qua các điểm đón và đến đích qua thuật toán dijkstra (UCS) thông qua các khoảng cách mà ta đã tính ở trên. => Độ phức tạp của thuật toán này: O(m * n * (số điểm đón)^2 / 2)

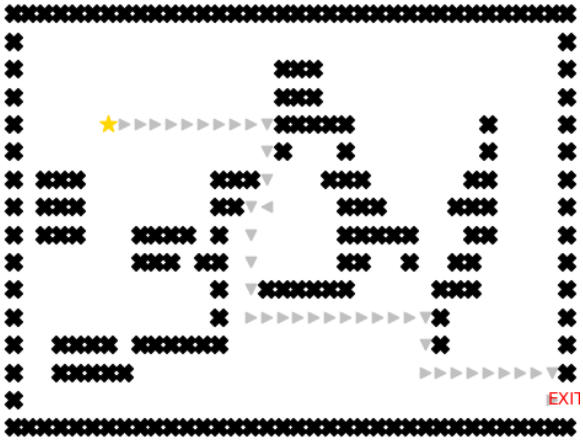
3. Kết quả

3.1. Bản đồ không có điểm thưởng

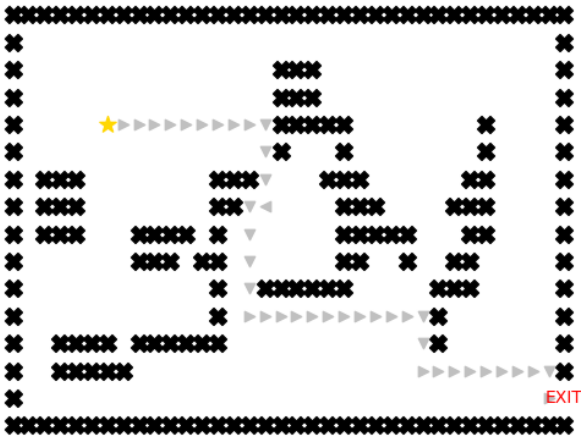
3.1.1. Bản đồ 1: 16 x 36

Depth-First Search

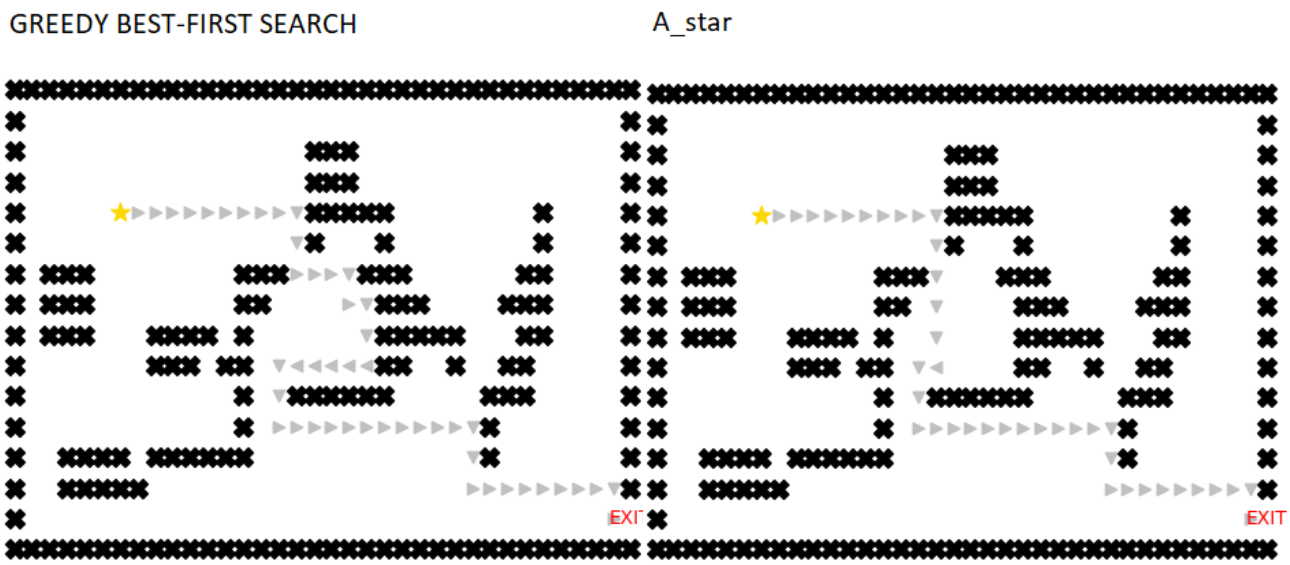
Breadth First Search



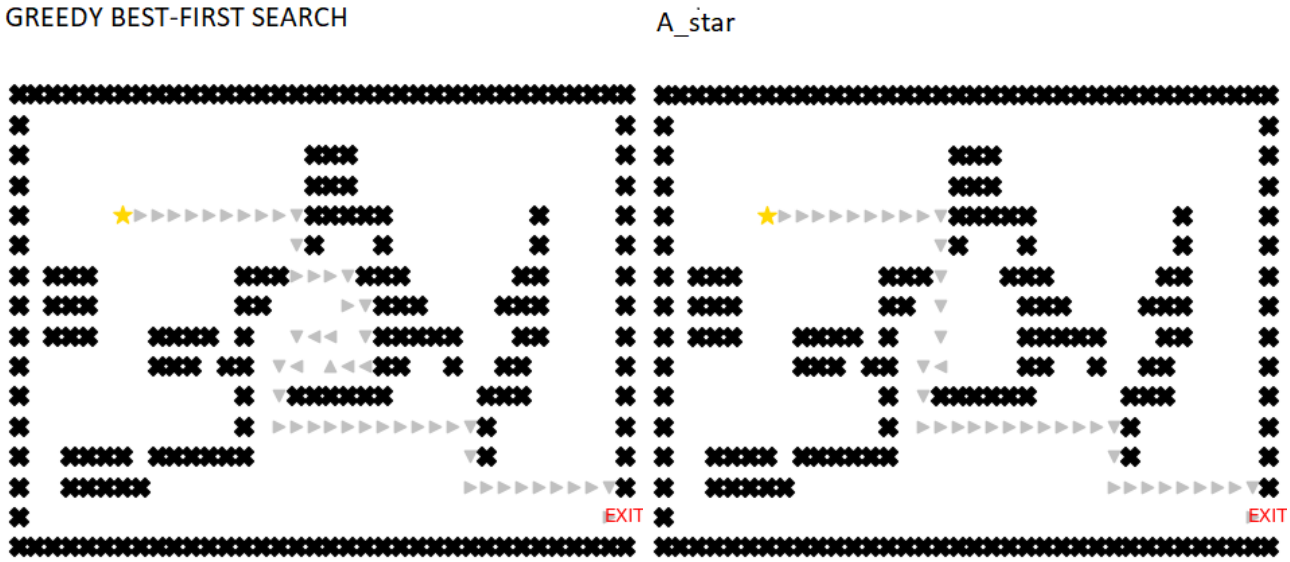
Uniform Cost Search



HEURISTIC: Manhattan



HEURISTIC: Euclidean



Phân tích:
Uninformed Search: DFS, BFS, UCS
Vì duyệt trên đồ thị có có chi phí đều nên BFS và UCS tương đương nhau.

- Kết quả: Cả DFS, BFS, UCS đều tìm được đường đi đến điểm ra (nếu đồ thị có đường đi). Tuy nhiên, đường đi của UCS, BFS thì tối ưu hơn vì tìm được đường đi ngắn nhất đến lối thoát, trong khi DFS đi sai hướng.
- Thời gian: trong đồ thi này DFS sẽ chạy nhanh hơn BFS, UCS do DFS chỉ cần đi theo đúng đường về còn BFS, UCS nó phải loang hết các điểm để tím được đường đến đích tối ưu nhất.

Informed Search: GBFS, A*

- Kết quả: ở bản đồ này cả 2 thuật toán đều tìm được đường đi để thoát ra khỏi mê cung. Tuy nhiên ta thấy việc tìm đường đi của A* tối ưu hơn GBFS. Vì GBFS nó ưu tiên lại

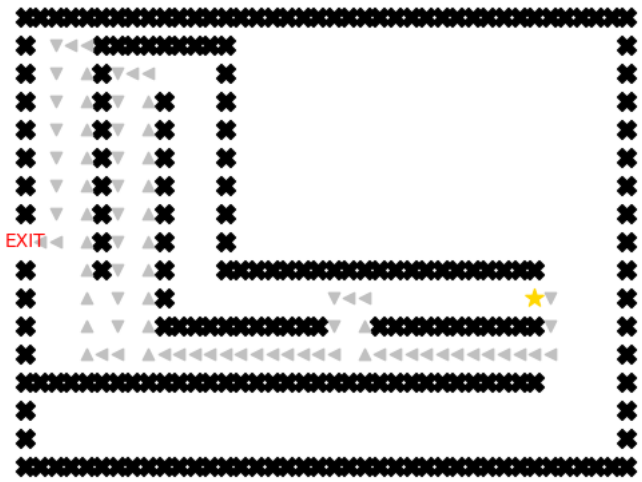
gần đích (h(n)) nên đoạn giữa nó đi qua phải rồi quay lại phía trái. Trong khi đó, A* nó dựa vào đường đi (g(n)) và heuristic (h(n)) nên tìm được đường đi hợp lý (đường đi ngắn nhất)

- So sánh các hàm heuristic: Ta sẽ dựa vào thuật toán GBFS để so sánh hàm heuristic vì GBFS chỉ sử dụng heuristic để tìm đường đi nên sẽ cho ta kết quả so sánh tường minh hơn. Ta thấy được trong bản đồ này hàm heuristic Manhattan sẽ cho ra kết quả hợp lý hơn hàm heuristic Euclidean. Vì trong bản đồ này ta chỉ di chuyển theo 4 hướng, trong khí đó Manhattan là tính khoảng cách dựa trên ngang dọc không tính chéo còn Euclidean là tính khoảng cách dựa theo tọa độ. Nên Manhattan sẽ đưa ra cho chúng ta dự đoán hợp lý hơn. Còn về thuật toán A* thì cả 2 hàm heuristic đều cho kết quả tối ưu vì A* còn dựa vào chi phí đường đi.

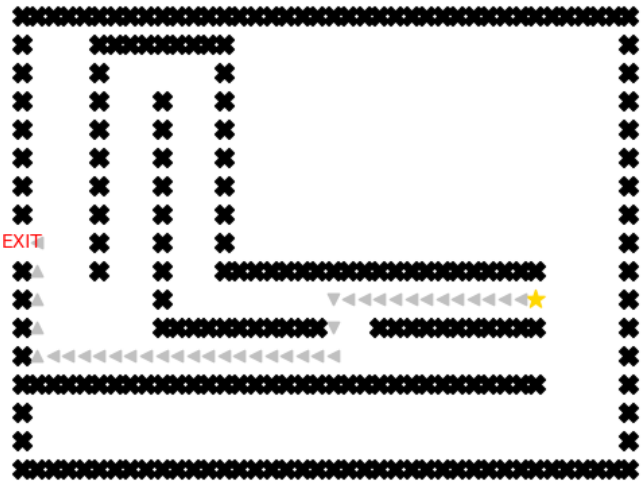
Tóm lại: Các thuật toán BFS, UCS, A* cho ra được đường đi tối ưu nhất. Nhưng A* sẽ có thời gian chạy ngắn hơn cũng như việc duyệt các đỉnh sẽ ít hơn. Vì BFS, UCS phải duyệt hết các hướng để xét, trong khi đó A* có hàm heuristic để xác định hướng đi tốt hơn. Và Heuristic tối ưu cho bản đồ này là Manhattan.

3.1.2. Bản đồ 2: 17 x 40

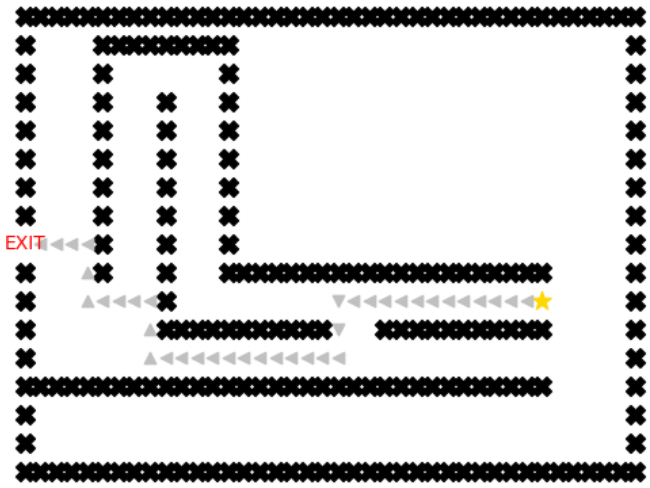
Depth-First Search



Breadth First Search

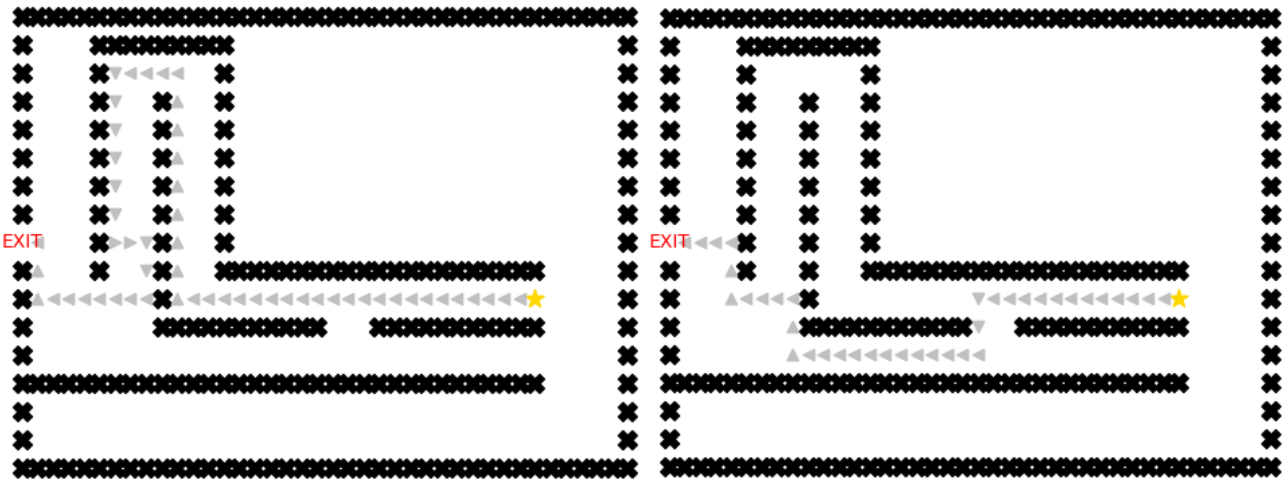


Uniform Cost Search



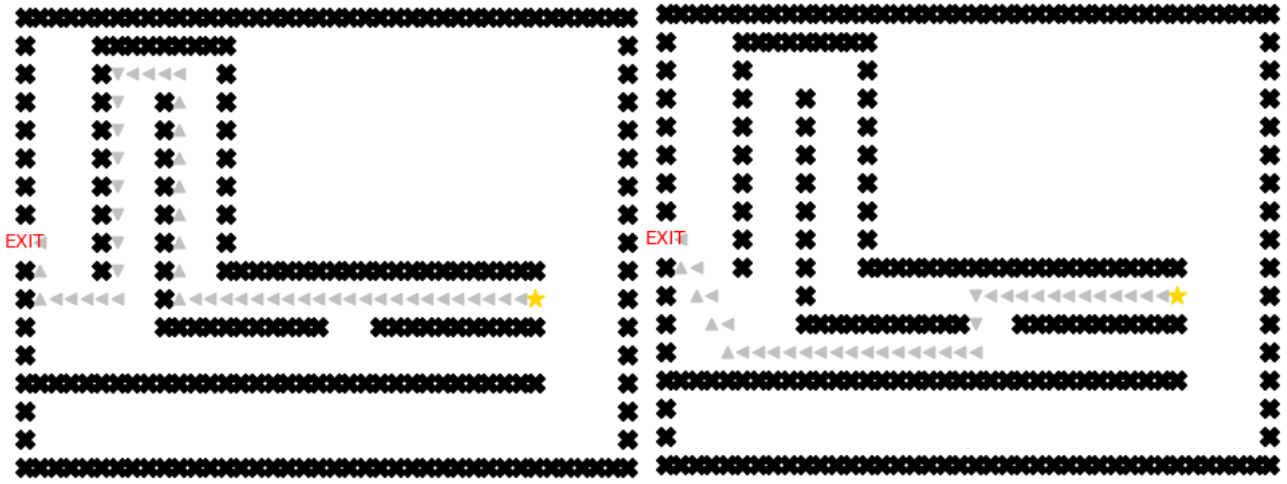
HEURISTIC: Manhattan

GREEDY BEST-FIRST SEARCH A_star



HEURISTIC: Euclidean

GREEDY BEST-FIRST SEARCH A_star



Phân tích:
Uninformed Search: DFS, BFS, UCS

- Kết quả: Cả DFS, BFS, UCS đều tìm được đường đi đến điểm ra (nếu đồ thị có đường đi). Tuy nhiên, đường đi của UCS, BFS thì tối ưu hơn DFS.
- Thời gian: Các thuật toán DFS, BFS, UCS với bản đồ này thì có thời gian chạy sấp sĩ nhau. Vì tất cả thuật toán này đều duyệt gần hết bản đồ.

Informed Search: GBFS, A*

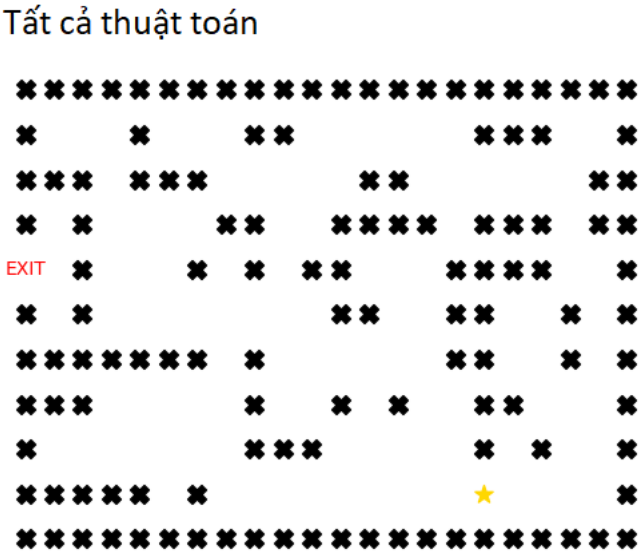
- Kết quả: ở bản đồ này cả 2 thuật toán đều tìm được đường đi để thoát ra khỏi mê cung. Tuy nhiên ta thấy việc tìm đường đi của A* tối ưu hơn GBFS. Vì GBFS nó ưu tiên lại gần đích (h(n)) nên nó chọn đường phía trên để đi vì thế nó phải đi lên rồi mới về đích.

Trong khi đó, A* nó dựa vào đường đi (g(n)) và heuristic (h(n)) nên nó đi theo hướng bên dưới. Tuy nhiên ở đây vì đường đường đi đến đích đặc biệt nên (h(n)) giúp cho GBFS duyệt ít điểm hơn A*. => GBFS chạy nhanh hơn

- Đầu tiên ta, với thuật toán A* ta thấy 2 hàm Heuristic đã đưa ra 2 đường đi khác nhau để đến đích nhưng đều tối ưu (chi phí = 40). Với thuật toán GBFS, với bản đồ này ta thấy được hàm Heuristic Euclidean có đường đi tối ưu hơn hàm Heuristic Manhattan. Hơn nữa nó còn duyệt ít điểm hơn.

Tóm lại: Tất của các thuật toán đều tìm ra được đường đi đến đích. Các thuật toán BFS, UCS, A* cho ra được đường đi tối ưu nhất. Nhưng A* vẫn chạy nhanh hơn vì có hàm dự đoán để định hướng thay vì phải đi hết các hướng như BFS. Và việc dùng hàm Heuristic Manhattan sẽ tối ưu trong đa số trường hợp của bản đồ di chuyển 4 hướng. Nhưng khai phải là tối ưu nhất, bản đồ trên là một minh chứng.

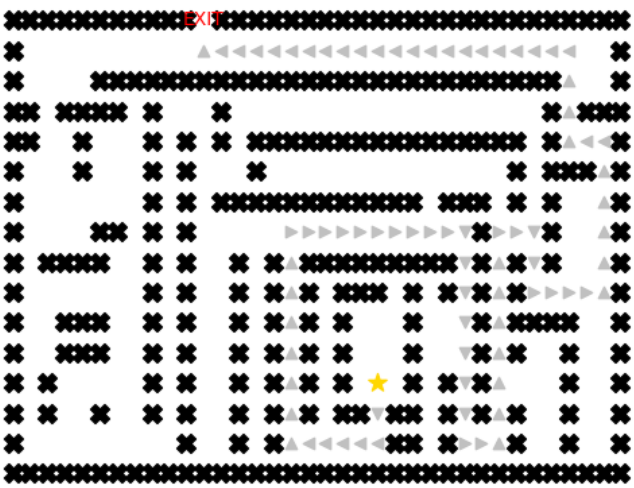
3.1.3. Bản đồ 3: 11 x 22



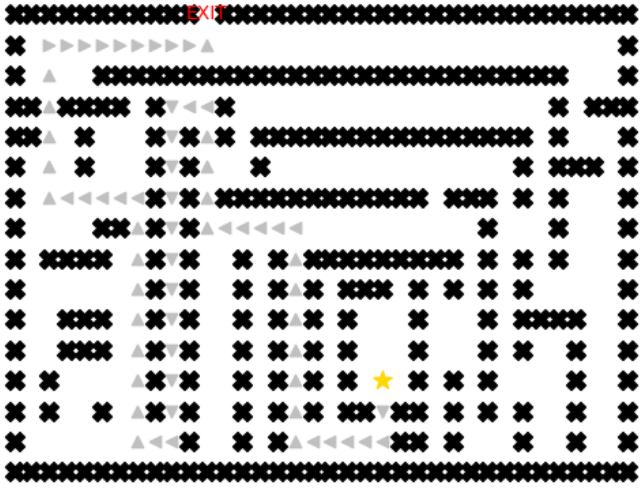
Phân tích:
Dây là bản đồ đặc biệt, bản đồ không thể thoát khỏi. Tất cả thuật toán chúng ta đã duyệt hết các điểm có thể đi từ điểm bắt đầu. Vì thế, với bản đồ này chúng ta sẽ nhận được kết quả là No (không đó đường thoát khỏi bản đồ), và thời gian thực hiện hay số điểm duyệt đến là như nhau.

3.1.4. Bản đồ 4: 16 x 36

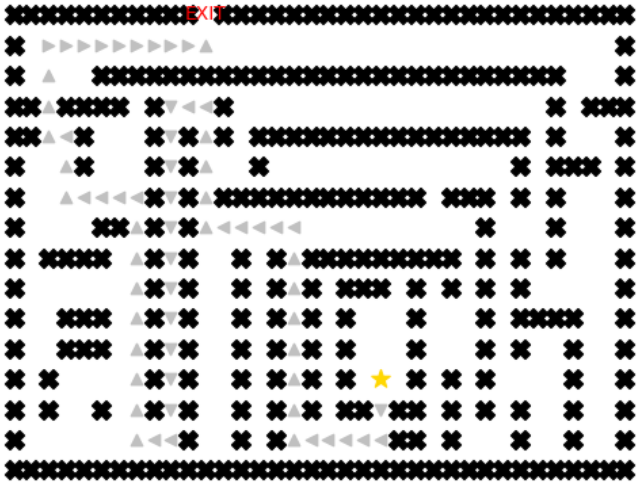
Depth-First Search



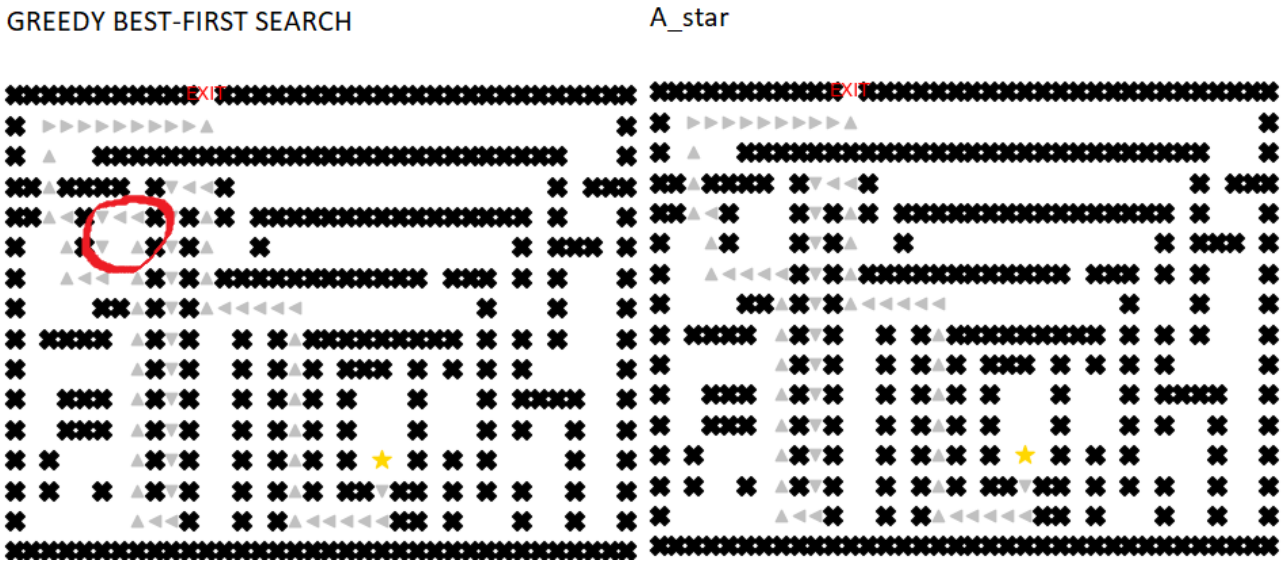
Breadth First Search



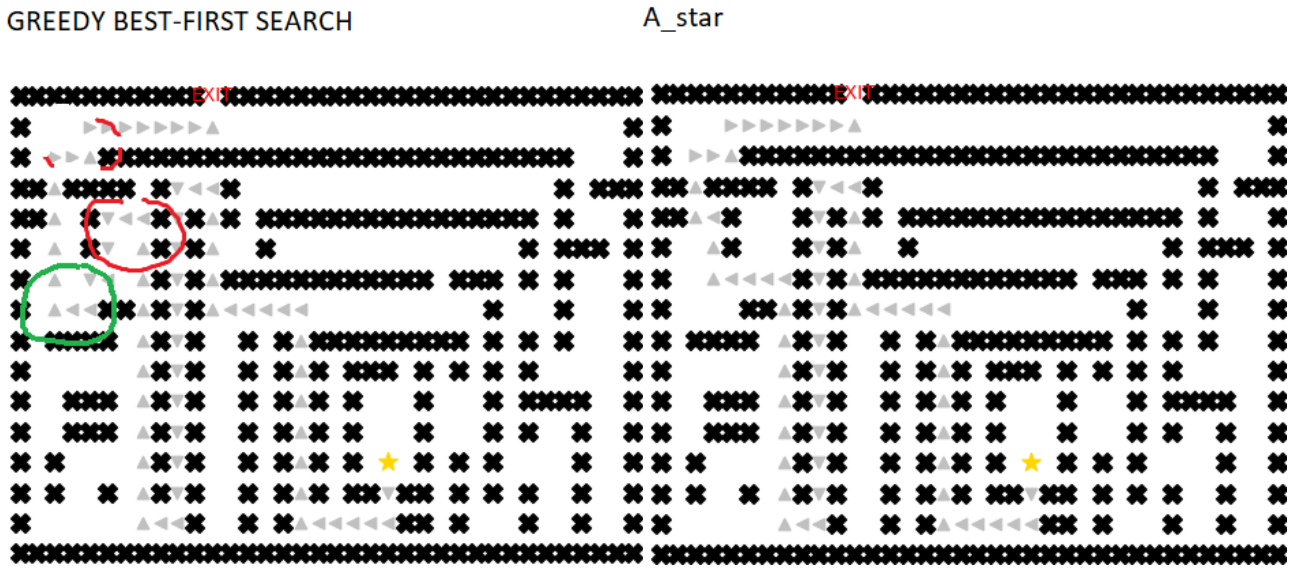
Uniform Cost Search



HEURISTIC: Manhattan



HEURISTIC: Euclidean



Phân tích:
Uninformed Search: DFS, BFS, UCS

- Kết quả: Cả DFS, BFS, UCS đều tìm được đường đi đến điểm ra (nếu đồ thị có đường đi). Tuy nhiên, đường đi của UCS, BFS thì tối ưu hơn DFS.
- Thời gian: trong đồ thi này DFS sẽ chạy nhanh hơn BFS, UCS do DFS chỉ cần đi theo đúng đường về còn BFS, UCS nó phải loang hết các điểm để tím được đường đến đích tối ưu nhất.

Informed Search: GBFS, A*

- Kết quả: ở bản đồ này cả 2 thuật toán đều tìm được đường đi để thoát ra khỏi mê cung. Tuy nhiên ta thấy việc tìm đường đi của A* tối ưu hơn GBFS. Trong điểm khoang tròn

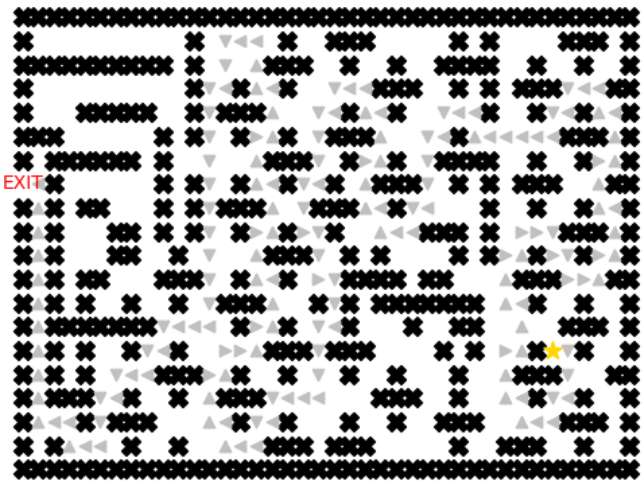
màu đỏ, ta thấy được GBFS sẽ đi lên phía trên vì nó dự đoán là sẽ thoát được mê cung gần nhất, tuy nhiên lên đến đó không có đường đi nên phải quay xuống. Trong khi đó, A* nó dựa vào đường đi (g(n)) và heuristic (h(n)) nên tìm được đường đi hợp lý (đường đi ngắn nhất)

- Như phân tích ở bản đồ 1 ở đây ta nhìn vào điểm được khoang xanh trong hình thì hàm heuristic Euclidean sẽ di chuyển không tối ưu bằng hàm heuristic Manhattan. Còn về thuật toán A* thì cả 2 hàm heuristic đều cho kết quả tối ưu vì A* còn dựa vào chi phí đường đi.

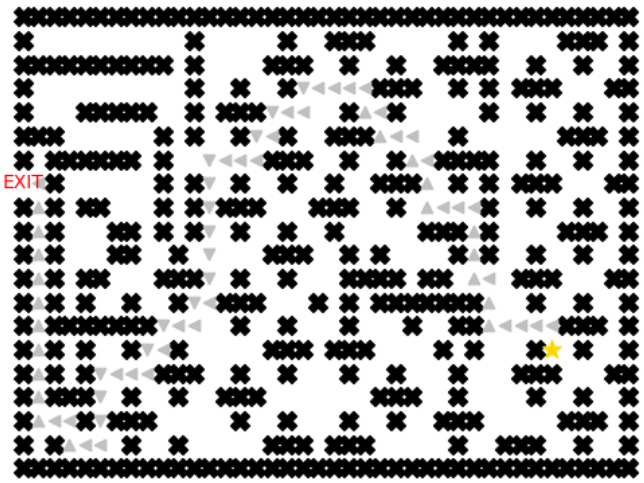
Tóm lại: Các thuật toán BFS, UCS, A* cho ra được đường đi tối ưu nhất. Nhưng A* vẫn chạy nhanh hơn vì có hàm dự đoán để định hướng thay vì phải đi hết các hướng như BFS. Ngoài ra có thể thấy các thuật toán của chúng ta đều có thể tìm ra được điểm ra. Ngoài ra, hàm GBFS ngay chổ khoang đỏ để so sánh với A* nếu ta mở thêm 1 cổng để đi thẳng lên điểm ra thì nó cũng sẽ giúp ta có đường đi tối ưu.

3.1.5. Bản đồ 5: 20 x 40

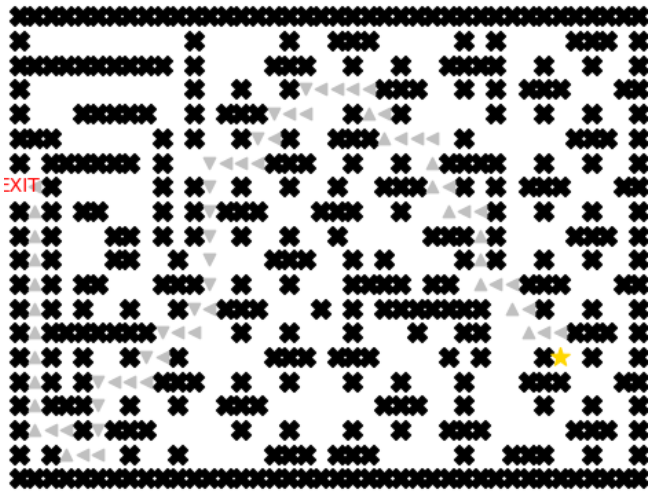
Depth First Search



Breadth First Search

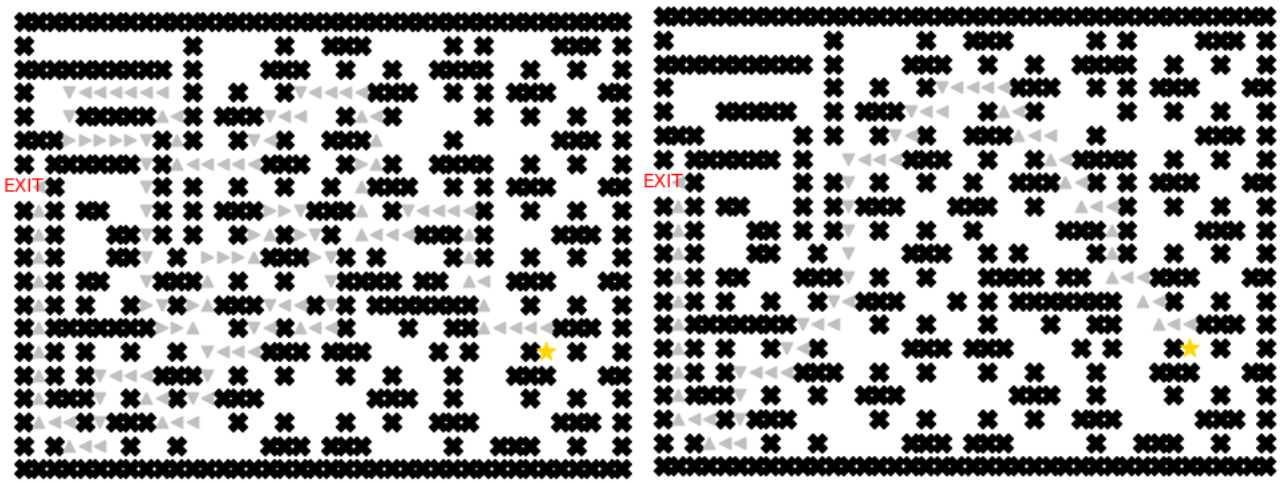


Uniform-Cost Search



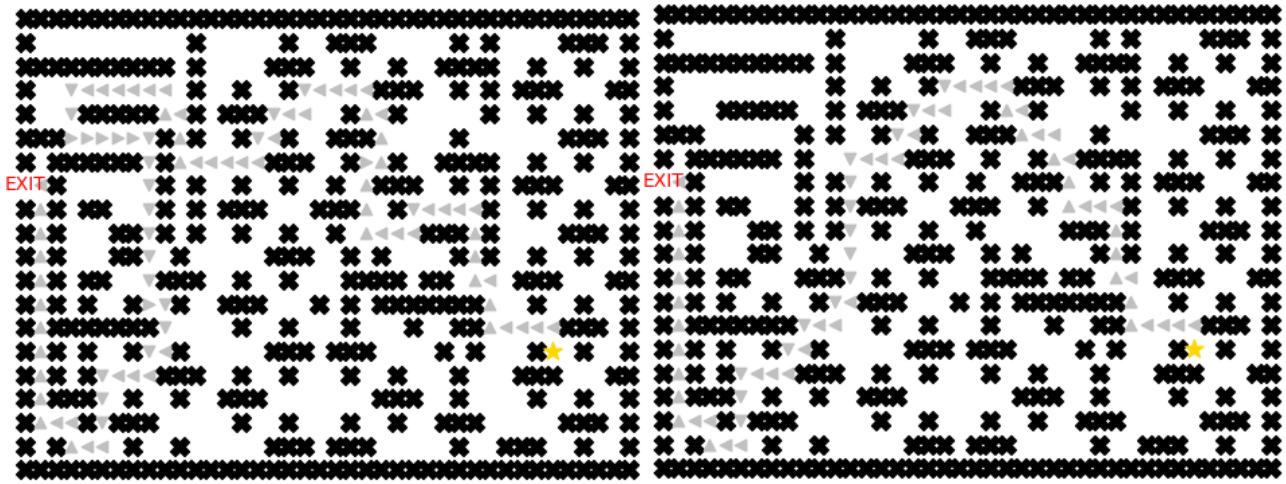
HEURISTIC: Manhattan

GREEDY BEST-FIRST SEARCH A_star



HEURISTIC: Euclidean

GREEDY BEST-FIRST SEARCH A_star



Phân tích:
Uninformed Search: DFS, BFS, UCS

- Kết quả: Cả DFS, BFS, UCS đều tìm được đường đi đến điểm ra (nếu đồ thị có đường đi). Tuy nhiên, đường đi của UCS, BFS thì tối ưu hơn DFS.
- Thời gian: trong đồ thi này DFS sẽ chạy nhanh hơn BFS, UCS do DFS chỉ cần đi theo đúng đường về còn BFS, UCS nó phải loang hết các điểm để tím được đường đến đích tối ưu nhất.

Informed Search: GBFS, A*

- Kết quả: ở bản đồ này cả 2 thuật toán đều tìm được đường đi để thoát ra khỏi mê cung. Tuy nhiên ta thấy việc tìm đường đi của A* tối ưu hơn GBFS. Vì GBFS nó ưu tiên lại

gần đích dự trên (h(n)) nên trong GBFS ta thấy nó đi qua bên trái gần điểm thoát nên nó phải quay trở xuống và đến đích. Trong khi đó, A* nó dựa vào đường đi (g(n)) và heuristic (h(n)) nên tìm được đường đi hợp lý (đường đi ngắn nhất)

- Với thuật toán A*, 2 hàm heuristic của chúng ta đều cho ra đường đi ngắn nhất. Trong khi đó với thuật toán GBFS (thuật toán chỉ dựa vào hheuristic) thì hàm Heuristic Eulidean cho ra đường đi tối ưu hơn hẳn Manhattan.

Tóm lại: Các thuật toán của chúng ta đều tìm ra đường đi để thoát ra khỏi mê cung. Trong đó, BFS, UCS và A* cho ra đường đi tối ưu nhất (chi phí ít nhất), A* có thời gian chạy nhanh hơn cũng như duyệt ít đỉnh hơn BFS và UCS.

3.2. Bản đồ có điểm thưởng

3.2.1. Bản đồ 2 điểm thưởng: 11 x 22

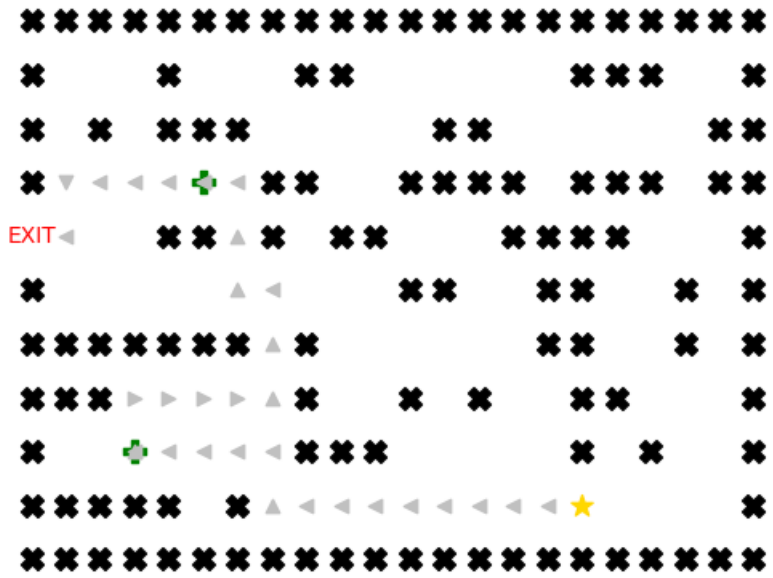


Figure 1: Bản đồ 2 điểm thưởng

Note: giá trị các điểm thưởng là: [(3, 5), -7] và [(8, 3), -8]
Nhận xét: Đối với bản đồ này, thuật toán chạy khá chính xác, chi phí tìm ra là bé hơn so với đường đi ngắn nhất từ Start đến End. Có thể đánh giá đây là đường đi tốt nhất có thể tìm được.

3.2.2. Bản đồ 5 điểm thưởng: 16 x 36

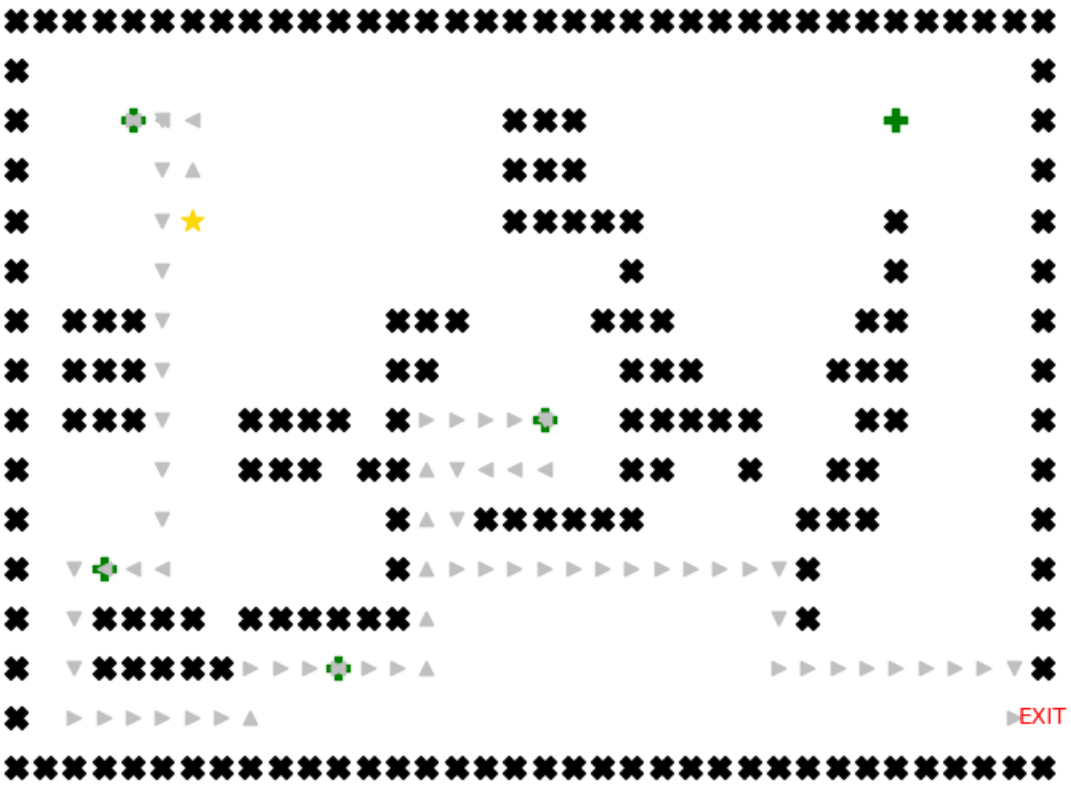


Figure 2: Bản đồ 5 điểm thưởng

Note: giá trị các điểm thưởng là: [(2, 4), -40], [(8, 18), -15], [(2, 30), -1], [(13, 11), -25] và [(11, 3), -1]
Nhận xét: Đối với bản đồ này, thuật toán chạy khá chính xác, chi phí tìm ra là bé hơn so với đường đi ngắn nhất từ Start đến End. Có thể đánh giá đây là đường đi tốt nhất có thể tìm được. Tuy nhiên, đường đi có đi qua điểm thưởng tại vị trí (11, 3) nhưng giá trị của điểm thưởng tại vị trí này không được tính vào chi phí đường đi, nghĩa là nó thuật toán chỉ đi qua điểm này như các điểm bình thường khác (tại vì giá trị của điểm thưởng tại điểm này nhỏ nên trong quá trình tính toán của thuật toán đã bị loại ra).

3.2.3. Bản đồ 10 điểm thưởng: 20 x 40

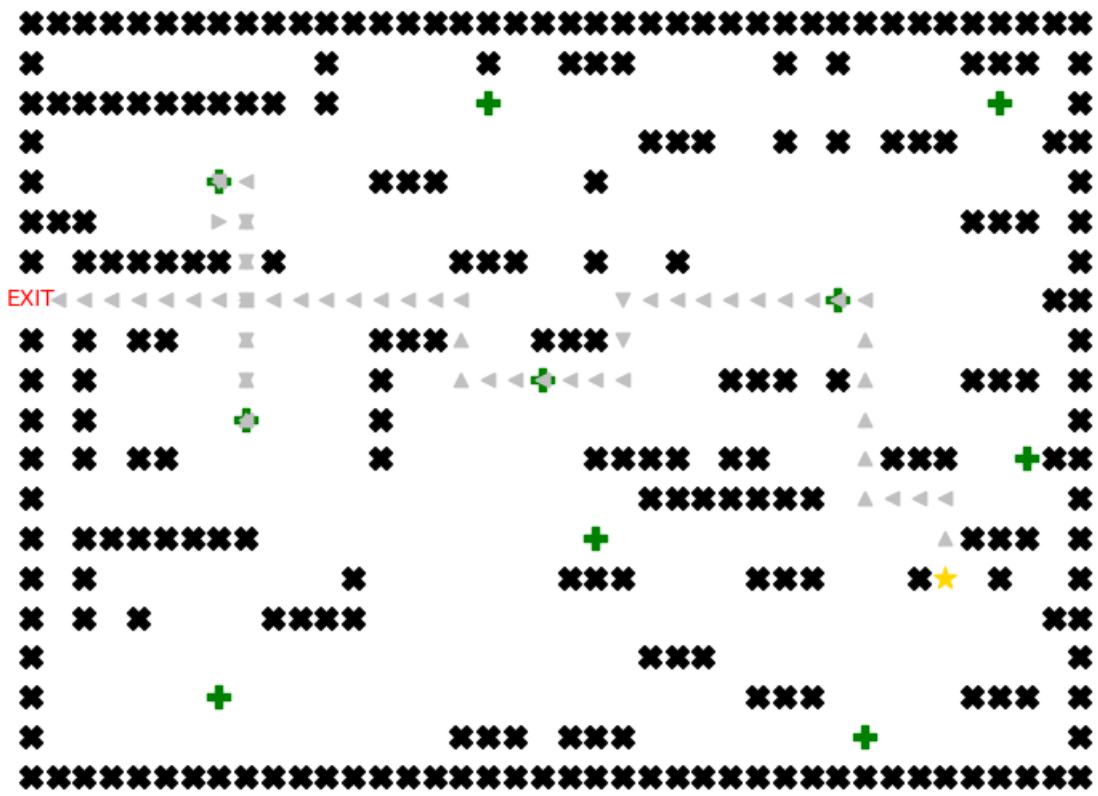


Figure 3: Bản đồ 10 điểm thưởng

Note: giá trị các điểm thưởng là: [(4, 7), -10], [(2, 17), -12], [(10, 8), -13], [(17, 7), -15], [(2, 36), -17], [(7, 30), -20], [(18, 31), -11], [(9, 19), -12], [(13, 21), -15] và [(11, 37), -17]
Nhận xét: Đối với bản đồ này, thuật toán chạy khá chính xác, chi phí tìm ra là bé hơn so với đường đi ngắn nhất từ Start đến End. Tuy nhiên, đây không phải là đường đi tốt nhất, tại vì nhìn sơ qua ta có thể thấy có thể lên điểm thưởng ở vị trí (11, 37) trước thì chi phí sẽ được giảm đi thêm, nhưng do thuật toán không phải tối ưu nhất nên đường đi không phải là tốt nhất.

3.2.4. Đề xuất cải tiến

Dùng thuật toán BFS lan ra tất cả các điểm thưởng của bản đồ cũng như điểm Start và điểm End, tính toán chi phí giữa tất cả các điểm đó (tính cả việc trừ đi giá trị của điểm thưởng tại các điểm thưởng), sau đó sắp xếp chúng lại và dùng giải thuật quay lui để tìm ra đường đi tối ưu nhất.

3.3. Bản đồ có điểm đón

3.3.1. Bản đồ 5 điểm đón: 16 x 36

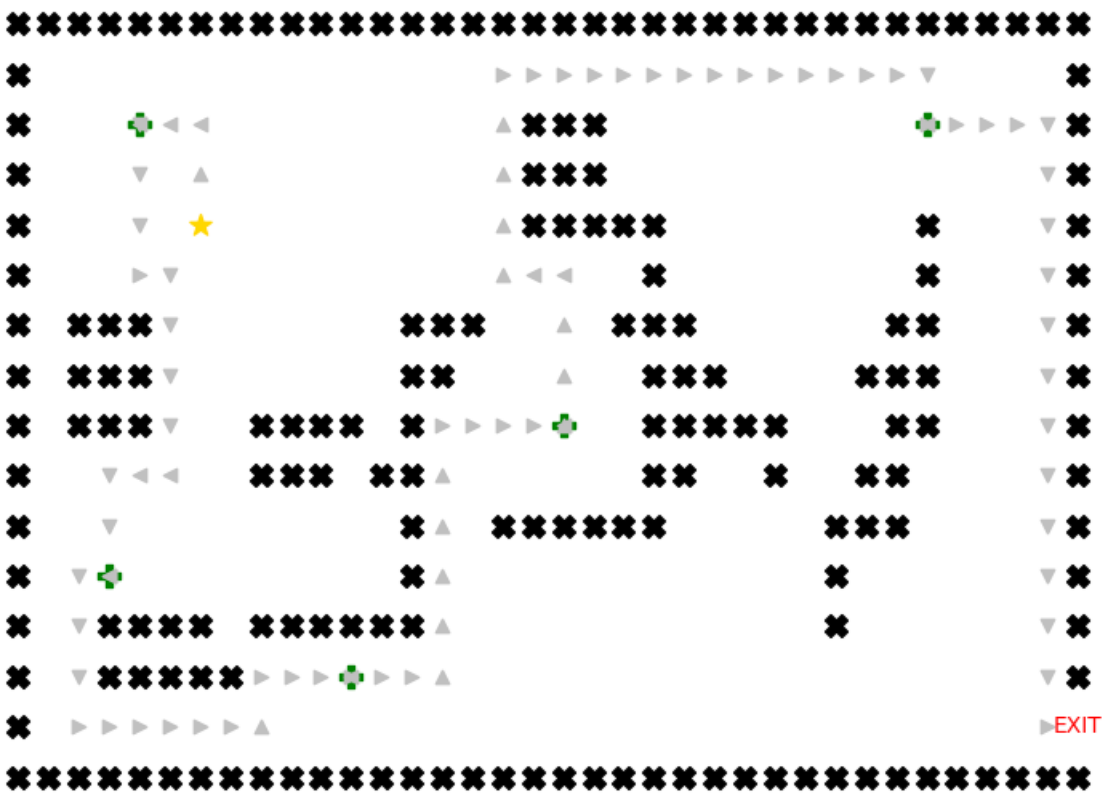


Figure 4: Bản đồ 5 điểm đón

Note: vì bản đồ có điểm đều có các giá trị tại đó là 0 nên nhóm em không để tọa độ các điểm ở đây, có thể đối chiếu xuống bản đồ bên dưới để nhận xét.
Nhận xét: Đối với bản đồ này, thuật toán chạy khá chính xác, chi phí tìm ra có thể nói là tối ưu nhất khi đi từ Start đến End nhưng vẫn đảm bảo đi qua hết tất cả các điểm đón. Bởi vì bản đồ này các điểm đón nằm cách khá đều nhau từ điểm Start đến điểm End nên điểm mạnh của thuật toán được phát huy tối đa.

3.3.2. Bản đồ 10 điểm đón: 20 x 40

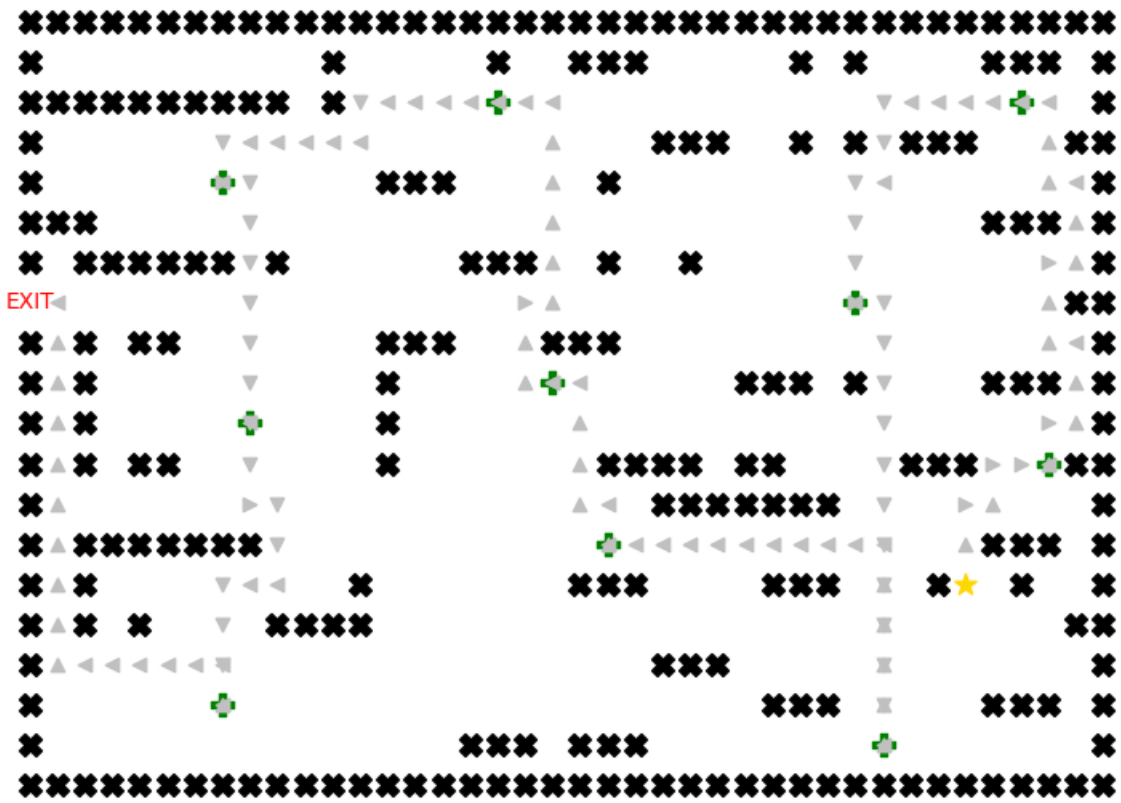


Figure 5: Bản đồ 10 điểm đón

Nhận xét: Đối với bản đồ này, thuật toán chạy khá chính xác, chi phí tiêu tốn có thể nói là ít nhất có thể.

3.3.3. Bản đồ 25 điểm đón: 20 x 48

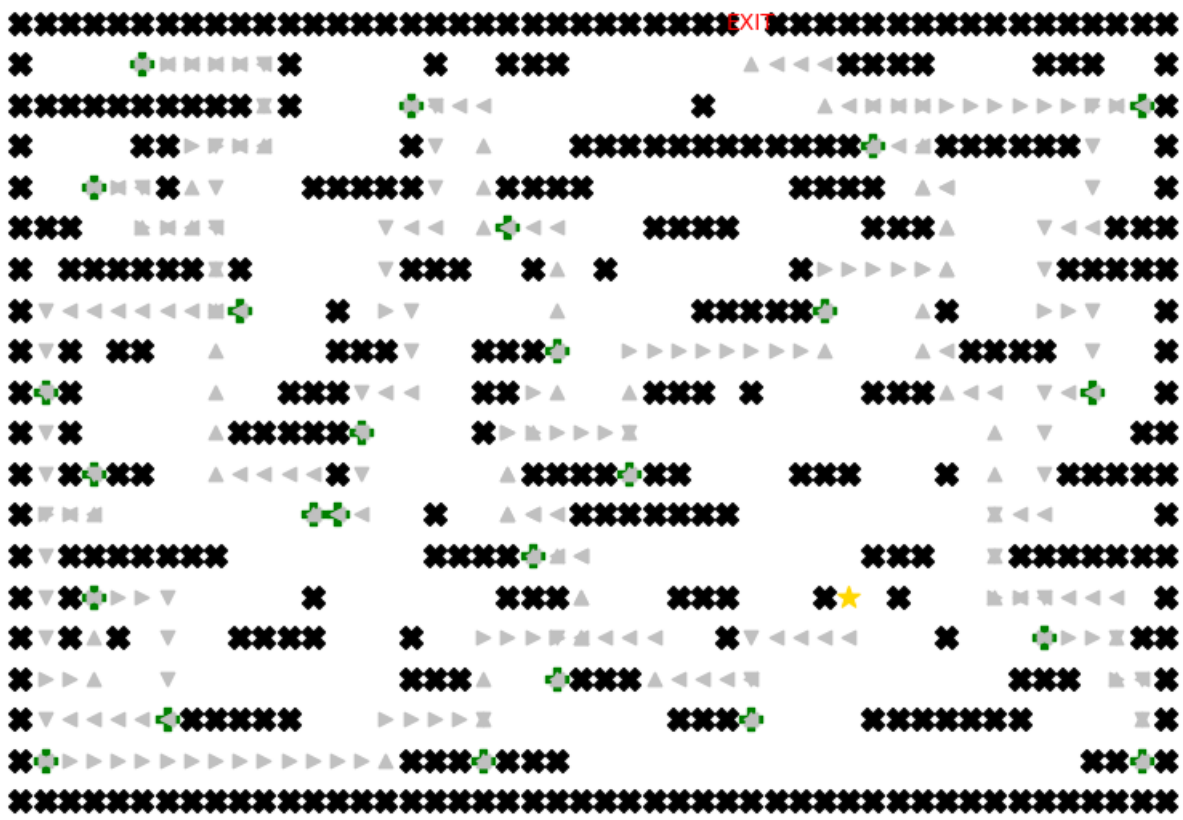


Figure 6: Bản đồ 25 điểm đón

Nhận xét: Đối với bản đồ này, bởi vì các điểm thưởng nằm rải rác khắp bản đồ với số lượng lớn nên thuật toán áp dụng vào đã không được tối ưu, nhưng vẫn đảm bảo đi được hết tất cả các điểm đón có trên bản đồ.

4. Mô tả file run.sh

File run.sh sẽ chứa thông tin của các thành viên trong nhóm và câu lệnh gọi tới file main.py nằm trong thư mục source để chạy toàn bộ chương trình.
Khi được gọi chạy, main,py sẽ thực hiện việc đọc từng thư mục trong thư mục input (level_1, level_2, level_3, advance) và đọc các tập tin input<x>.txt bên trong đó.

Tài liệu tham khảo

[1] Slies bài giảng của thầy Lê Hoài Bắc và hướng dẫn thực hiện đồ án của thầy Nguyễn Duy Khánh

[2] BFS (Breadth-first search): https://vnoi.info/wiki/algo/graph-theory/breadth-first-search.md

[3] A* Algorithm Concepts and Implementation: https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm

[4] vidmaker 2.3.0: https://pypi.org/project/vidmaker/