ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

****

**BÁO CÁO ĐỒ ÁN 01**

**CÁC THUẬT TOÁN TÌM KIẾM**

**Môn học: Cơ sở trí tuệ nhân tạo**

**Giảng viên hướng dẫn**

**Nguyễn Duy Khánh**

**Nguyễn Ngọc Băng Tâm**

Thành phố Hồ Chí Minh - 2022

**MỤC LỤC**

[**THÔNG TIN THÀNH VIÊN NHÓM** 4](#_Toc117411271)

[**I.** **Tự đánh giá** 5](#_Toc117411272)

[**II.** **Mô tả bài toán** 6](#_Toc117411273)

[**1.** **Trạng thái bắt đầu của tác nhân** 6](#_Toc117411274)

[**2.** **Các hành động của tác nhân** 6](#_Toc117411275)

[**3.** **Các bước di chuyển** 6](#_Toc117411276)

[**4.** **Kết thúc** 6](#_Toc117411277)

[**III.** **Thuật toán** 7](#_Toc117411278)

[**1.** **Thuật toán tìm kiếm DFS (tìm kiếm không có thông tin)** 7](#_Toc117411279)

[**2.** **Thuật toán tìm kiếm BFS (tìm kiếm không có thông tin)** 8](#_Toc117411280)

[**3.** **Thuật toán tìm kiếm UCS (tìm kiếm không có thông tin)** 9](#_Toc117411281)

[**4.** **Thuật toán tìm kiếm A\* (tìm kiếm có thông tin)** 10](#_Toc117411282)

[**5.** **Thuật toán tìm kiếm tham lam Greedy Best First Search (tìm kiếm có thông tin)** 11](#_Toc117411283)

[**6.** **Lựa chọn heuristic cho các chiến lược tìm kiếm có thông tin** 12](#_Toc117411284)

[*a)* *Lựa chọn heuristic là khoảng cách Mathanttan đến vị trí exit* 12](#_Toc117411285)

[*b)* *Lựa chọn heuristic là khoảng cách euclid đến vị trí exit* 13](#_Toc117411286)

[**IV.** **Chiến lược tìm kiếm không điểm thưởng trên các loại bản đồ khác nhau** 14](#_Toc117411287)

[**1.** **Bản đồ maze\_map1.txt – bản đồ thông thường** 14](#_Toc117411288)

[**2.** **Bản đồ maze\_map2.txt – bản đồ có nhiều bức tường song song, tạo thành 2 hướng về đích** 17](#_Toc117411289)

[**3.** **Bản đồ maze\_map3.txt – bản đồ có 1 đường đi ngắn, 1 đường đi dài** 20](#_Toc117411290)

[**4.** **Bản đồ maze\_map4.txt – bản đồ có nhiều vật chắn, ngõ cụt** 23](#_Toc117411291)

[**5.** **Bản đồ maze\_map5.txt – bản đồ lớn có nhiều vật chắn nhỏ, thưa** 26](#_Toc117411292)

[**6.** **Kết luận** 28](#_Toc117411293)

[**V.** **Chiến lược tìm kiếm trên các bản đồ có điểm thưởng khác nhau để đạt được chi phí thấp nhất** 30](#_Toc117411294)

[**1.** **Chiến lược tìm kiếm** 30](#_Toc117411295)

[**2.** **Chạy thuật toán với các bản đồ có 2, 5, 10 điểm thưởng** 32](#_Toc117411296)

[**VI.** **Chiến lược tìm kiếm trên bản đồ có các cổng dịch chuyển** 34](#_Toc117411297)

[**1.** **Chiến lược** 34](#_Toc117411298)

**2. Chạy thuật toán với các bản đồ có cổng dịch chuyển với thuật toán A\***…...35

[**VII.** **Mô tả chiến lược tìm kiếm của các thuật toán với Video cụ thể** 37](#_Toc117411299)

[**1.** **Bản đồ không điểm thưởng maze\_map1.txt** 37](#_Toc117411300)

[**2.** **Bản đồ không điểm thưởng maze\_map2.txt** 40](#_Toc117411301)

[**3.** **Bản đồ không điểm thưởng maze\_map3.txt** 43](#_Toc117411302)

[**4.** **Bản đồ không có điểm thưởng maze\_map4.txt** 46](#_Toc117411303)

[**5.** **Bản đồ không có điểm thưởng maze\_map5.txt** 4](#_Toc117411303)9

[**6.** **Bản đồ có 2 điểm thưởng maze\_map1.txt** 50](#_Toc117411304)

[**7.** **Bản đồ có 5 điểm thưởng maze\_map2.txt** 51](#_Toc117411305)

[**8.** **Bản đồ có 10 điểm thưởng maze\_map3.txt** 51](#_Toc117411306)

[**9.** **Bản đồ có 1 cổng dịch chuyển maze\_map1.txt** 52](#_Toc117411307)

[**10.** **Bản đồ có 2 cổng dịch chuyển maze\_map2.txt** 52](#_Toc117411308)

[**11.** **Bản đồ có 3 cổng dịch chuyển maze\_map3.txt** 53](#_Toc117411309)

[**VIII.** **Mô tả các câu lệnh trong file run.sh** 54](#_Toc117411310)

**IX. Thuật toán chạy trên Google Colab** 55

[**TÀI LIỆU THAM KHẢO** 5](#_Toc117411311)6

# **THÔNG TIN THÀNH VIÊN NHÓM**

|  |  |  |
| --- | --- | --- |
| MSSV | Họ Tên | Công việc |
| 20120165 | Hồng Nhất Phương | 1, 2, 3, 4, 9 |
| 20120176 | Nguyễn Ngọc Thúy Quỳnh | 1, 5, 6, 9 |
| 20120201 | Phạm Gia Thông | 7, 8, 9, 10 |

|  |  |
| --- | --- |
| Danh sách công việc | |
| 1 | Mô tả thuật toán |
| 2 | Thiết kế bản đồ |
| 3 | Viết các hàm phụ trợ |
| 4 | Thiết kế thuật toán tìm kiếm trên bản đồ có điểm thưởng |
| 5 | Thiết kế thuật toán tìm kiếm trên bản đồ không điểm thưởng |
| 6 | Thiết kế thuật toán trên bản đồ có các cổng dịch chuyển |
| 7 | Viết báo cáo |
| 8 | Tìm hiểu thuật toán |
| 9 | Kiểm thử |
| 10 | Thiết kế bản chạy trên các hệ điều hành thuộc họ Unix |

1. **Tự đánh giá**

|  |  |  |
| --- | --- | --- |
| STT | Tên yêu cầu | Mức độ hoàn thành |
| 1 | Thiết kế 5 bản đồ không điểm thưởng và hoàn thành 3 thuật toán tìm kiếm không có thông tin | Hoàn thành |
| 2 | Thiết kế 5 bản đồ không điểm thưởng và hoàn thành 2 thuật toán tìm kiếm có thông tin | Hoàn thành |
| 3 | Đề xuất thuật toán, hàm heuristic phù hợp và thiết kế 3 bản đồ tìm kiếm có điểm thưởng | Hoàn thành |
| 4 | Cài đặt thành công thuật toán tìm kiếm trên bản đồ có điểm thưởng | Hoàn thành |
| 5 | Kịch bản nâng cấp có các cổng dịch chuyển, tìm lời giải tối ưu và thiết kế 3 bản đồ | Hoàn thành |
| 6 | Output ra video minh họa thay vì chỉ có ảnh thông thường | Hoàn thành |

1. **Mô tả bài toán**
2. **Trạng thái bắt đầu của tác nhân**

Trạng thái bắt đầu tìm kiếm đường đi để thoát khỏi mê cung là tọa độ vị trí start (kí hiệu ‘\*’) của tác nhân trong mê cung

1. **Các hành động của tác nhân**

Với 1 điểm bất kỳ trong mê cung (không phải là tường (có giá trị ‘x’)), các hành động có thể thực hiện là di chuyển phải, trái, lên, xuống 1 đơn vị trong mê cung nếu vị trí tiếp theo không phải là tường.

1. **Các bước di chuyển**

Sau khi chọn 1 trong các hành động có thể thực hiện (hướng di chuyển đến vị trí kế tiếp), vị trí của tác nhân hiện tại là vị trí ô nằm ở hướng di chuyển tương ứng.

Không gian trạng thái là các vị trí của tác nhân trong mê cung và cạnh là các hành động lên, xuống, trái, phải.

Một đường đi trong không gian trạng thái là một chuỗi các vị trí của tác nhân được kết nối bằng chuỗi các hành động di chuyển.

1. **Kết thúc**

Bài toán kết thúc khi tác nhân đang ở trạng thái đích, nghĩa là tác nhân đến được vị trí EXIT để thoát khỏi mê cung.

1. **Thuật toán**

Trước tiên, ta cần tìm hiểu về các chiến lược tìm kiếm, có 5 thành phần để biểu diễn một bài toán tìm kiếm:

+ Không gian trạng thái: là tập hợp tất cả các trạng thái có thể có của một bài toán.

+ Trạng thái ban đầu: nơi bắt đầu công việc tìm kiếm.

+ Hàm trạng thái con: phát sinh trạng thái kế tiếp từ một trạng thái cụ thể. Hàm này thông thường dùng để mô tả những biến đổi hay hành động có thể xảy ra cho một trạng thái nào đó.

+ Một trạng thái đích (kết thúc): trạng thái này đều thuộc không gian trạng thái.

+ Hàm chi phí đường đi: Dùng để tính chi phí của bài toán.

Cách giải một bài toán tìm kiếm:

+ Tìm ra được 5 thành phần biểu diễn bài toán tìm kiếm

+ Nhờ vào không gian trạng thái, ta bắt đầu thực hiện tìm kiếm.

+ Ta xây dựng cây tìm kiếm từ trạng thái ban đầu và hàm trạng thái con.

+ Xem xét nút hiện tại có phải là đích cần tìm hay không. Nếu không thì ta xét các trạng thái đích bằng cách tìm đến các trạng thái tiếp theo qua đó tạo ra một trạng thái mới. Sau đó từ trạng thái đã chọn và xét tiếp.

+ Tiếp tục công đoạn kiểm tra cho đến khi tìm được đích hoặc không thể mở rộng thì dừng tìm kiếm.

1. **Thuật toán tìm kiếm DFS (tìm kiếm không có thông tin)**

Ý tưởng chung của thuật toán:

DFS luôn mở rộng nút nằm sâu nhất trong nhánh hiện tại đang xét trên cây tìm kiếm (nút lá). Với những nút đã được mở rộng sẽ loại khỏi nhánh tìm kiếm, khi đó công việc tìm kiếm sẽ quay lại nút gần nhất có các nút con chưa được xét đến.

Các bước thực hiện đối với bài toán tìm đường mê cung:

* Sử dụng 2 stack: **open** dùng để lưu các vị trí có thể đi tới và **close** dùng để đánh dấu các vị trí đã thăm, thêm vị trí bắt đầu vào ngăn xếp là điểm xuất phát của tác nhân.
* Thực hiện lặp, với mỗi vòng lặp, lấy 1 đỉnh từ trong **open**, nếu không phải là đích đến thì đưa vào **close**. Sau đó tìm các vị trí đi tiếp được trong 4 hướng Đông, Nam, Bắc, Tây (ESNW), nếu có đường đi tới nhưng đã nằm trong **close** thì không làm gì cả, ngược lại thì lưu vào **open**. Nếu vị trí đang xét không có đường đi tiếp thì tiếp tục lấy điểm tiếp theo trong **open** để xét. Thực hiện lặp lại các bước trên cho đến khi **open** rỗng hoặc đã tìm được đích.
* Từ vị trí đích, dùng dictionary **path** như 1 con trỏ quay lui lưu vết đi từ đích về nguồn. Kết quả của thuật toán trả về list **res** chứa các giá trị (value) của **path** được đảo ngược để có con đường từ nguồn đến đích theo DFS.

1. **Thuật toán tìm kiếm BFS (tìm kiếm không có thông tin)**

Ý tưởng chung của thuật toán:

Từ một đỉnh ban đầu, ta tìm các đỉnh kề rồi duyệt qua các đỉnh này. Sau đó tiếp tục tìm đến các đỉnh kề của đỉnh vừa xét rồi duyệt tiếp đến khi qua tất cả các đỉnh có thể đi. Trong quá trình đi đến đỉnh kề, tiến hành lưu lại đỉnh cha của đỉnh kề để khi đi ngược lại từ đỉnh kết thúc đến đỉnh kề, ta sẽ tìm được đường đi ngắn nhất.

Ta cũng có các bước thực hiện như thuật toán DFS:

* Sử dụng 2 queue: **open** dùng để lưu các vị trí có thể đi tới và **close** dùng để đánh dấu các vị trí đã thăm, thêm vị trí bắt đầu vào hàng đợi là điểm xuất phát của tác nhân.
* Thực hiện lặp, với mỗi vòng lặp, ta dequeue 1 điểm từ **open**, nếu chưa phải là đích thì đưa vào close. Sau đó tìm tất cả các vị trí có thể đi tiếp được trong 4 hướng ESNW, nếu có đường đi tới nhưng nằm trong **close** thì không làm gì cả, ngược lại thì enqueue vào **open**. Nếu điểm đang xét không có đường đi tiếp thì tiếp tục enqueue điểm tiếp theo trong **open** để xét. Thực hiện lặp lại các bước trên cho đến khi **open** rỗng hoặc đã tìm được đích.
* Từ vị trí đích, dùng dictionary **path** như 1 con trỏ quay lui lưu vết đi từ đích về nguồn. Kết quả của thuật toán trả về list **res** chứa các value của **path** được đảo ngược để có con đường từ nguồn đến đích theo BFS.

1. **Thuật toán tìm kiếm UCS (tìm kiếm không có thông tin)**

Ý tưởng chung của thuật toán:

UCS là một cách duyệt dùng cho việc duyệt hay tìm kiếm trên cây có trọng số. Thuật toán sẽ phát triển các nút chưa xét có chi phí thấp nhất – các nút được xét theo thứ tự chi phí tăng dần. Khi đồ thi có chi phí ở mỗi bước là như nhau thì thuật toán trở thành phương pháp tìm kiếm theo chiều rộng.

Các bước thực hiện trong bài toán tìm đường mê cung:

* Sử dụng 2 queue: **open** dùng để lưu các vị trí có thể đi tới và chi phí g, **close** dùng để đánh dấu các vị trí đã thăm. **Open** là một **priority queue**.
* Thực hiện lặp, với mỗi vòng lặp, ta dequeue 1 điểm từ **open**, nếu điểm đó không là đích thì đưa vào **close**. Sau đó tìm tất cả vị trí có thể đi tiếp được theo 4 hướng ESNW, nếu có đường đi tới nhưng đã nằm trong close thì không làm gì cả, ngược lại thì thêm vị trí và chi phí (g + 1) vào **open**. Nếu điểm đang xét không có đường đi thì tiếp tục xét điểm tiếp theo trong **open**. Lặp lại các bước trên đến khi **open** rỗng hoặc đã tìm được đích.
* Từ vị trí đích, dùng dictionary **path** như 1 con trỏ quay lui lưu vết đi từ đích về nguồn. Kết quả của thuật toán trả về list **res** chứa các value của path được đảo ngược để có con đường từ nguồn đến đích theo UCS.

1. **Thuật toán tìm kiếm A\* (tìm kiếm có thông tin)**

Ý tưởng chung của thuật toán:

Thuật toán đánh giá một nút dựa trên chi phí đi từ nút gốc đến nút đó 𝑔(𝑛) cộng chi phí từ nút đó đến đích thông qua hàm heuristic ℎ(𝑛) , 𝑓(𝑛) = 𝑔(𝑛) + ℎ(𝑛) .

Do giá trị 𝑔(𝑛) cho biết chi phí đường đi từ nút gốc đến nút n và ℎ(𝑛) là ước lượng chi phí đường đi ngắn nhất từ nút n đến đích nên ta có 𝑓(𝑛) là ước lượng chi phí của lời giải tốt nhất qua n. Giá trị của f càng nhỏ, độ ưu tiên càng cao. Nhờ việc sử dụng hàm ℎ(𝑛) , thuật toán A\* cho phép tìm đường đi nhanh hơn, tránh các đường đi có chi phí lớn.

Các bước thực hiện trong bài toán tìm đường mê cung:

Gọi g(n) là chi phí đi từ start đến điểm đang xét và h(n) là hàm heuristic ước lượng khoảng cách từ điểm đang xét đến đích. Dùng 2 dictionary để lưu giá trị g(n), h(n) của từng điểm, ban đầu g(n) và h(n) có giá trị vô cực (đối với start thì g(n) = 0 và h(n) là heuristic từ start đến goal)

Gọi **open** là hàng đợi ưu tiên lưu 3 giá trị f(n), h(n) và toạ độ của điểm đang xét

Dùng vòng lặp chạy đến khi hàng đợi rỗng hoặc tìm được đích:

* Lấy toạ độ điểm có chi phí f(n) nhỏ nhất, nếu f(n) bằng nhau thì sẽ lấy điểm có h(n) nhỏ nhất (gần với đích nhất)
* Xét tất cả vị trí có thể đi tiếp trong 4 hướng ESNW rồi tính giá trị g(n) và f(n) của chúng, nếu chi phí hiện tại nhỏ hơn chi phí ban đầu thì cập nhật chúng và đưa vào **open**.

Dùng dictionary **path** như 1 con trỏ quay lui lưu vết đi từ đích về nguồn. Kết quả của thuật toán trả về list **res** chứa các value của **path** được đảo ngược để có con đường từ nguồn đến đích theo Astar.

1. **Thuật toán tìm kiếm tham lam Greedy Best First Search (tìm kiếm có thông tin)**

Ý tưởng chung của thuật toán:

Thuật toán đánh giá một nút dựa trên chi phí hàm ước lượng heuristic h(n) từ đỉnh n.

Việc tìm kiếm bắt đầu tại nút gốc và tiếp tục bằng cách duyệt các nút tiếp theo có giá trị của hàm đánh giá heuristic h(n) là thấp nhất so với các nút còn lại nằm trong hàng đợi.

Các bước thực hiện đối với bài toán tìm đường mê cung:

* Tính giá trị heuristic cho tất cả điểm không phải tường trên mê cung, h(n) là hàm heuristic ước lượng khoảng cách từ điểm đang xét đến đích và dùng dictionary để lưu giá rị h(n) của từng điểm.
* Sử dụng **open** là hàng đợi ưu tiên lưu 2 giá trị h(n) và tọa độ của điểm đang xét, **close** dùng để đánh dấu các vị trí đã thăm (giá trị heuristic càng nhỏ, độ ưu tiên càng cao), thêm vị trí bắt đầu vào hàng đợi là điểm xuất phát của tác nhân.
* Thực hiện lặp, với mỗi vòng lặp, ta dequeue 1 điểm từ **open**, nếu điểm đó không là đích thì đưa vào **close**. Sau đó tìm tất cả vị trí có thể đi tiếp được theo 4 hướng đông nam bắc tây, nếu có đường đi tới nằm trong close thì continue, ngược lại thì thêm vị trí và h(n) vào open. Nếu điểm đang xét không có đường đi thì tiếp tục xét điểm tiếp theo trong open. Lặp lại các bước trên đến khi open rỗng hoặc đã tìm được đích.
* Từ vị trí đích, dùng dictionary **path** như 1 con trỏ quay lui lưu vết đi từ đích về nguồn. Kết quả của thuật toán trả về list **res** chứa các values của **path** được đảo ngược để có con đường từ nguồn đến đích theo Greedy Best First Search.

1. **Lựa chọn heuristic cho các chiến lược tìm kiếm có thông tin**
2. *Lựa chọn heuristic là khoảng cách Mathanttan đến vị trí exit*

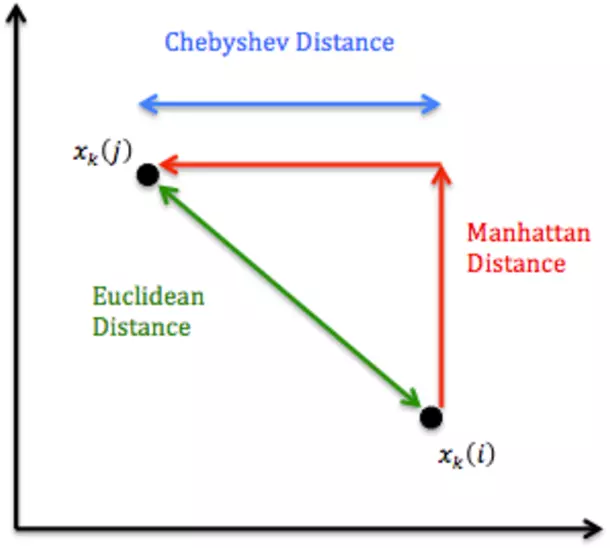
Thực tế trong mê cung, ta hoàn toàn không có cách nào đi theo đường thẳng đến đích vì không gian mê cung của chúng ta là rời rạc. Vì vậy, ta có thể ước lượng chi phí đến đích chính xác hơn thông qua chi phí ít nhất cần có để đến đích. Đó là khoảng cách Manhattan.

Heuristic này không ước lượng quá cao chi phí đến đích và luôn nhỏ hơn hoặc bằng chi phí đến đích (đường đi theo Manhattan luôn đi qua số điểm là ít nhất do chỉ đi về 1 hướng theo trục Ox, và Oy). Suy ra đây là Heuristic hợp lý.

Gọi n là 1 điểm trên mê cung, n’ là 1 succesor của n.

Gọi d(xki, xkj) là khoảng cách Manhattan từ a đến b.

Ta có: 𝑑(𝑛, 𝑒𝑥𝑖𝑡) ≤ 𝑑(𝑛, *n*’) + 𝑑(𝑛’, 𝑒𝑥𝑖𝑡) ⇒ ℎ(𝑛) ≤ 𝑐(𝑛, 𝑛’) + ℎ(𝑛’)  
Vậy đây là 1 heuristic nhất quán.



1. *Lựa chọn heuristic là khoảng cách euclide đến vị trí exit*

Ta có thể hướng tác nhân đến đích bằng cách lấy hàm heuristic là khoảng cách Euclid từ điểm đang xét đến đích

Heuristic này không ước lượng quá cao chi phí đến đích và luôn nhỏ hơn chi phí đến đích (đường thẳng là đường đi ngắn nhất giữa 2 điểm). Suy ra đây là Heuristic hợp lý.

Gọi n là 1 điểm trên mê cung, n’ là 1 succesor của n.

Gọi 𝑑(a, b) là khoảng cách Euclid từ a đến b.

Ta có: 𝑑(𝑛, 𝑒𝑥𝑖𝑡) ≤ 𝑑(𝑛, 𝑛’) + 𝑑(𝑛’, 𝑒𝑥𝑖𝑡) ⇒ ℎ(𝑛) ≤ 𝑐(𝑛, 𝑛’) + ℎ(𝑛’)

Vậy đây là 1 heuristic nhất quán.

1. **Chiến lược tìm kiếm không điểm thưởng trên các loại bản đồ khác nhau**
2. **Bản đồ maze\_map1.txt – bản đồ thông thường**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chiến lược** | **Chi phí** | **Đường đi** | **Ghi chú** |
| **BFS** | 55 |  | BFS cho ta được kết quả đi qua ít bước trong bản đồ, trong trường hợp chi phí di chuyển là như nhau, BFS cũng cho ra đường đi ngắn nhất, tuy nhiên, thời gian thực hiện của BFS lâu hơn so với các thuật toán khác (không đáng kể). |
| **DFS** | 287 |  | DFS cho ra một đường đi xấu và chi phí cao, gần như tác nhân đi qua hầu hết mọi vị trí sâu nhất trong bản đồ do mở các nút theo chiều sâu. |
| **UCS** | 55 |  | Đường đi của UCS tương tự như đường đi mà thuật toán BFS cho ra nhưng UCS xét chi phí để đến vị trí tiếp theo nên tốt hơn và có thời gian thực hiện nhanh hơn BFS. |
| **Greedy BFS Euclidean** | 63 |  | Thuật toán tìm kiếm tham lam sẽ cố gắng tìm đường đi đến đích theo heuristic. Trong trường hợp này, khoảng cách Euclide làm cho tác nhân đi theo zigzac ở đoạn đường gần về đích |
| **Greedy BFS Manhattan** | 101 |  | Thuật toán vì sử dụng heuristic Manhattan để đánh giá khoảng cách do đó đường đi có xu hướng tiến lên trên dẫn đến việc đụng vào tường chắn và tìm cách quay lại. Trong trường hợp này, heuristic là Manhattan cho ra đường đi tốn kém hơn so với Euclide. |
| **A\* Euclidean** | 55 |  | Khi quan tâm thêm số bước đi của đường đi, thuật toán A\* cùng với heuristic là khoảng cách Euclide, đã hướng về đường đi ngắn nhất. Cũng giống với GBFS, đường đi vẫn có đoạn zigzac. |
| **A\* Manhattan** | 55 |  | A\* với heuristic là khoảng cách Manhattan cũng cho ra một trong những đường đi tốt nhất và khử zigzac. |

1. **Bản đồ maze\_map2.txt – bản đồ có nhiều bức tường song song, tạo thành 2 hướng về đích**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chiến lược** | **Chi phí** | **Đường đi** | **Ghi chú** |
| **BFS** | 57 |  | BFS cho ra là đường đi ngắn với chi phí thấp mà thuật toán tìm kiếm được. |
| **DFS** | 245 |  | Với bản đồ này, DFS vẫn duyệt gần như hết bản đồ, kết quả tìm kiếm là đooạn đường có chi phí rất cao. |
| **UCS** | 57 |  | UCS cho ra kết quả gần như tương tự đường đi mà thuật toán BFS cho ra nhưng tối ưu hơn do UCS xét chi phí để đi đến từng vị trí tiếp theo nên có thời gian thực hiện nhanh hơn BFS vì phải mở hết tất cả các nút trên cùng một tầng. |
| **Greedy BFS Euclidean** | 83 |  | Trong trường hợp này, thuật toán tham lam vừa tìm được đường đi có chi phí thấp, vừa chạy với thời gian ngắn nhờ chiến lược tìm kiếm hợp lý. |
| **Greedy BFS Manhattan** | 119 |  | Cho ra đường đi gần giống GBFS với heuristic là Euclidean cho ra. Tuy nhiên do sử dụng heuristic là khoảng cách Manhattan nên có đoạn đi men theo tường chắn song song |
| **A\* Euclidean** | 57 |  | Thuật toán A\* với hàm heuristic dùng chiến lược tìm kiếm Euclidean cho ra đường đi có chi phí tốt nhất. |
| **A\* Manhattan** | 57 |  | Đường đi có chi phí tương tự khi sử dụng khoảng cách Euclide, tuy nhiên do sử dụng khoảng cách Manhattan nên đường đi có xu hướng tiến về phía trên, tiếp cận với độ cao nơi có đích đến |

1. **Bản đồ maze\_map3.txt – bản đồ có 1 đường đi ngắn, 1 đường đi dài**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chiến lược** | **Chi phí** | **Đường đi** | **Ghi chú** |
| **BFS** | 40 |  | Với bản đồ này, vị trí cửa thoát nằm gần và ngang với vị trí bắt đầu của tác nhân, tạo điều kiện thuận lợi cho BFS tìm ra một đường đi ngắn nhất cùng một chi phí hợp lý. |
| **DFS** | 224 |  | Ở bản đồ này, chiến lược tìm kiếm theo chiều sâu của DFS đã gây ra nhiều bất lợi trong việc tìm kiếm ra đường đi đến đích. DFS gần như phủ kín mọi vị trí mà nó mở ra được, điều này dẫn đến một đường đi với chi phí cực kì cao và phức tạp nhất. Qua bản đồ này, ta cũng thấy được độ mở nút chằng chịt theo chiều sâu của thuật toán DFS để tìm ra được đường đi đến đích. |
| **UCS** | 40 |  | Tương tự với chiến lược tìm kiếm đường đi như BFS, UCS cũng cho ta một đường đi tối ưu và có chi phí thấp nhất để đến được đích. |
| **Greedy BFS Euclidean** | 40 |  | Với bản đồ có khoảng cách vị trí đích gần với vị trí bắt đầu, không quan tâm đến số bước đi, tìm kiếm tham lam có hàm heuristic Euclide tối ưu về khoảng cách cho ta một đường đi ngắn nhất đến đích. |
| **Greedy BFS Manhattan** | 40 |  | Tương tự với heuristic Euclide, heuristic dùng chiến lược Manhanttan vẫn cho ta một đường đi ngắn nhất bởi thuật toán tìm kiếm tham lam. |
| **A\* Euclidean** | 40 |  | Khi cần quan tâm thêm số bước đi và dùng heuristic Euclid có độ tối ưu về khoảng cách trên bản đồ này, A\* cho ta một đường đi đến đích ngắn nhất và nhanh nhất. |
| **A\* Manhattan** | 40 |  | Thay đổi hàm heuristic có chiến lược tìm kiếm là Manhattan, A\* quan tâm thêm số bước đi vẫn cho ta một đường đi ngắn nhất. |

1. **Bản đồ maze\_map4.txt – bản đồ có nhiều vật chắn, ngõ cụt**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chiến lược** | **Chi phí** | **Đường đi** | **Ghi chú** |
| **BFS** | 71 |  | Thuật toán BFS trong trường hợp này sớm kết thúc và cho ta đường đi ngắn. Tuy nhiên, BFS không có tính tối ưu do phải mở hết các nút để tìm kiếm đường đi nên chi phí của đường đi tìm ra không phải là thấp nhất tại thời điểm đang xét. |
| **DFS** | 243 |  | Đường đi của DFS phụ thuộc rất lớn vào thứ tự duyệt vị trí lân cận của một điểm. Trong trường hợp điểm thoát mê cung ở vị trí bất lợi so với thứ tự duyệt, đường đi sẽ có chi phí cao. |
| **UCS** | 71 |  | Đường đi của UCS tương tự như đường đi mà thuật toán BFS nhưng UCS tối ưu và có thời gian thực hiện nhanh hơn vì số lượng nút mở ít hơn BFS. |
| **Greedy BFS Euclidean** | 71 |  | Với tìm kiếm tham lam, ta không quan tâm đến số bước đã đi, vì vậy thuật toán sẽ cố gắng tìm đường đi đến đích theo heuristic. Trong trường hợp này, khoảng cách Euclid đã hướng đường đi theo hướng đường đi tối ưu. |
| **Greedy**  **BFS Manhattan** | 71 |  | Sử dụng Heuristic tốt hơn khiến cho thuật toán tham lam không cần quan tâm đến số bước đi vẫn tìm ra được đường đi ngắn nhất. |
| **A\***  **Euclidean** | 71 |  | Cũng với heuristic là khoảng cách Euclide, khi quan tâm thêm số bước đi của đường đi, thuật toán A\* đã hướng đường về đường đi ngắn nhất. |
| **A\* Manhattan** | 71 |  | Với hàm heuristic là Manhattan, A\* quan tâm thêm số bước đi, cũng cho ra một trong những đường đi tốt nhất. |

1. **Bản đồ maze\_map5.txt – bản đồ lớn có nhiều vật chắn nhỏ, thưa**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chiến lược** | **Chi phí** | **Đường đi** | **Ghi chú** |
| **BFS** | 92 |  | BFS cho đường đi ngắn, tránh thooát các vật chắn thưa, men theo tường để về đích. |
| **DFS** | 414 |  | Chi phí đường đi DFS tìm kiếm vẫn rất cao và gần như lấp đầy tất cả đường đi hẹp. |
| **UCS** | 92 |  | UCS cũng cho ra một đường đi tốt, tránh các ngõ cụt mà tác nhân có thể đi vào, với một chi phí đường đi thấp. |
| **Greedy BFS Euclidean** | 134 |  | Đường đi cố gắng tìm tới những vị trí gần đích, do đó có nhiều khúc phải quay lại. Thuật toán đi theo heuristic là khoảng cách Euclide, do đó có nhiều khúc zigzag, đi chéo về hướng đích. |
| **Greedy BFS Manhattan** | 122 |  | Sử dụng Heuristic tốt hơn, ít đoạn zigzac, tuy nhiên, trong trường hợp này, chi phí thực hiện vẫn con cao do vẫn còn nhiều đoạn quay lại. |
| **A\* Euclidean** | 92 |  | Với A\*, do thuật toán còn quan tâm đến số bước đi, những đoạn quay lại đã bị hạn chế do những chúng tốn nhiều chi phí thực hiện. |
| **A\* Manhattan** | 92 |  | Cho đường đi có chi phí thấp nhất và ít khử đoạn zigzac ở vài bước đi cuối. |

1. **Kết luận**

Chiến lược tìm kiếm ở các thuật toán trên các loại bản đồ không có điểm thưởng nhìn chung đều có thời gian thực hiện thuật toán nhanh và cho ra hình ảnh của đường đi tìm đường gần như tương đồng nhau (trong khoảng 0 – 1.5s).

Tất cả thuật toán tìm kiếm đường đi đều sẽ cho ra đường đi tới đích nếu tồn tại ít nhất 1 đường đi. Vì trong trường hợp xấu nhất, dictionary **path** của tất cả thuật toán sẽ duyệt qua toàn bộ vị trí có thể đi đến trong mê cung.

Thuật toán DFS cho ta lời giải không tốt trong hầu hết trường hợp, có thể tăng hiệu xuất bằng cách thực hiện sắp xếp thứ tự duyệt hợp lý với vị trí điểm ra. Tuy nhiên, trong các trường hợp đường đi đầu tiên đến vị trí điểm ra có chi phí lớn DFS vẫn không thể cải thiện nhiều.

Thuật toán BFS luôn cho ta đường đi có chi phí thấp, tuy nhiên, thời gian duyệt BFS có thể rất lâu.

Thuật toán tham lam tỏ ra khá hữu ích trong các trường hợp đường đi tới đích ít cạnh chắn, những trường hợp này thời gian thực hiện thuật toán khá nhanh và tốn ít chi phí bộ nhớ hơn so với A\* (không cần lưu trữ thêm số bước đi). Tuy nhiên trong thực tế, xác suất xuất hiện những mê cung này là rất thấp.

Thuật toán A\* có thể xem là tối ưu nhất vì trong một trường hợp ngẫu nhiên, A\* vừa đảm bảo đi ít bước, vừa đảm bảo đi đường đi có khả năng về đích nhất.

DFS và BFS có số lượng nút mở nhiều nhất, A\* có số lượng nút mở tuy nhiều nhưng đoạn đường mở nút vẫn gần như tương tự với kết quả thuật toán.

Heuristic là khoảng cách Manhattan tỏ ra vượt trội hơn hẳn khoảng cách Euclide ở đa số bản đồ kiểm thử.

1. **Chiến lược tìm kiếm trên các bản đồ có điểm thưởng khác nhau để đạt được chi phí thấp nhất**
2. **Chiến lược tìm kiếm**

Ý tưởng: mục tiêu của tác nhân là tham lam ăn hết tất cả n điểm thưởng có trên bản đồ. Thứ tự ăn điểm thưởng sẽ căn cứ vào ước lượng (heuristic) khoảng cách từ start đến từng điểm thưởng, tác nhân sẽ ăn điểm thưởng gần mình nhất, đến điểm thưởng gần nhì,... cho đến khi tìm được lối ra khỏi mê cung.

**Trường hợp 1**: tác nhân ăn hết điểm thưởng và ra khỏi mê cung

Tác nhân tìm được đường đi đến và ăn mọi điểm thưởng, cuối cùng tìm được đường đi ra khỏi mê cung. Khi đó, chi phí sẽ là chi phí đường đi cộng với tổng điểm thưởng.

**Trường hợp 2**: tác nhân không ăn hết điểm thưởng

**Trường hợp 2.1**: Giả sử tác nhân đã ăn được i điểm thưởng và không thể tìm đường đến điểm thưởng i + 1, khi đó điểm thưởng i + 1 sẽ được bỏ qua và tác nhân sẽ tìm đường đi đến điểm thưởng thứ i + 2. Tương tự với những điểm thưởng khác cho đến khi về đích.

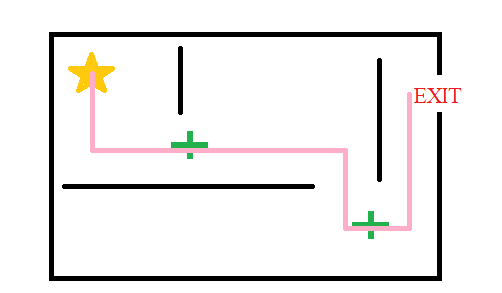
**Trường hợp 2.2**: Giả sử tác nhân đã ăn được i điểm thưởng và không thể tìm được đi đến các điểm thưởng khác kể cả đích đến, khi đó điểm thưởng ấy sẽ được loại bỏ hoàn toàn khỏi danh sách điểm thưởng, tác nhân sẽ tìm đường đi sao cho có thể ăn hết n - 1 điểm thưởng trên bản đồ. Tương tự với các điểm thưởng khác cho đến khi về đích.

Lựa chọn thuật toán tìm kiếm A\* là chiến lược tìm kiếm đường đi trên các bản đồ có điểm thưởng vì A\* đảm bảo được đường đi ngắn nhất mà vẫn giúp tác nhân ăn được các điểm thưởng trên đường đi.

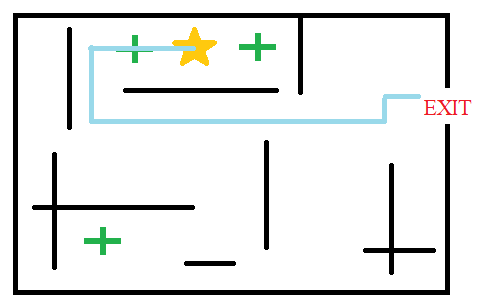
Thuật toán sẽ có điểm dừng, tuy nhiên, với trường hợp có 10 điểm thưởng, thuật toán sẽ chạy với thời gian khá lâu.

Tính chính xác của chi phí phụ thuộc vào tính chính xác của thuật toán A\*.

Minh họa ý tưởng thuật toán với bản đồ đơn giản:

* Tác nhân có thể ăn hết điểm thưởng

Khi số lượng điểm thưởng ít và nằm rải rác, tác nhân có thể tìm ra đường đi ngắn mà vẫn ăn được hết điểm thưởng và về đích

* Tác nhân bỏ một vài điểm thưởng

Khi số lượng điểm thưởng tăng lên, tác nhân lựa chọn bỏ một vài điểm thưởng nằm ở ngõ cụt hoặc phải đi đè lên con đường cũ. Trong ảnh minh họa, tác nhân chỉ chọn ăn một điểm thưởng rồi quay đầu về đích.

1. **Chạy thuật toán với các bản đồ có 2, 5, 10 điểm thưởng**

|  |  |  |  |
| --- | --- | --- | --- |
| **Bản đồ** | **Chi phí** | **Đường đi** | **Ghi chú** |
| **2điểm thưởng**  **6 2 -10**  **10 33 -5** | Fst:  74  Scd:  59 |  | Do số điểm thưởng ít, tác nhân tham lam ăn hết tất cả điểm thưởng có trên bản đồ và đạt được chi phí thấp nhất khi đến đích. |
| **3điểm thưởng**  **10 20 -10**  **8 1 -30**  **3 32 -5** | Fst: 66  Scd:  21 |  | Tác nhân không còn tìm đường đi ngang để đến đích ngắn nhất và nhanh nhất nữa mà ưu tiên tìm vị trí các điểm thưởng để ăn rồi mới tìm đường đến đích. Ở đây, số điểm thưởng vẫn ít và nằm rải rác trên bản đồ, do đó tác nhân vẫn dễ dàng tính toán và tìm được đích đến. |
| **10điểm thưởng**  **1 6 -20**  **2 36 -15**  **5 70 -10**  **7 22 -22**  **8 24 -22**  **9 35 -6**  **9 38 -8**  **12 10 -5**  **13 26 -3**  **15 59 -1** | Fst: 168  Scd:  79 |  | Với bản đồ có số điểm thưởng nhiều và chỉ có thể đi qua các vị trí duy nhất một lần, tác nhân buộc phải loại bỏ 2 điểm thưởng vì chúng dẫn vào ngõ cụt hoặc bước lại vào những vị trí đã đi qua. Nếu tham lam ăn hết tất cả điểm thưởng trên bản đồ, tác nhân sẽ không tìm ra được đường đi đến đích nữa. |

1. **Chiến lược tìm kiếm trên bản đồ có các cổng dịch chuyển**
2. **Chiến lược**

Nếu tác nhân đến điểm dịch chuyển, thay vì chỉ có các bước di chuyển là trên, dưới, trái, phải, ta thêm 1 trường hợp di chuyển nữa là cổng dịch chuyển ở đầu bên kia.

Thuật toán phải thêm 1 hàm kiểm tra điểm hiện tại có phải điểm dịch chuyển không, làm giảm hiệu xuất thực hiện của các thuật toán gốc.

* g(n) là chi phí từ start đến điểm đang xét
* h(n) là hàm heuristic ước lượng khoảng cách từ điểm đang xét đến đích.
* Dùng dictionary lưu giá trị h(n) và g(n) của từng điểm.
* Sử dụng **open** là hàng đợi ưu tiên lưu 3 giá trị f(n), h(n) và tọa độ của điểm đang xét.
* Sử dụng list **inTeleport** lưu các điểm là cổng dịch chuyển vào. List **outTeleport** là cổng dịch chuyển ra. List **openTele** chứa các điểm cổng dịch chuyển vào đã mở. Dictionary teleportations chứa cặp cổng dịch chuyển (từ điểm (x, y) chỉ có thể dịch chuyển đến điểm (x', y')).
* Dequeue 1 điểm từ open. Sau đó tìm tất cả vị trí có thể đi tiếp được theo 4 hướng đông nam bắc tây.
* **Trường hợp 1**: Nếu điểm đang xét là một điểm bình thường thì tính g + 1, và tính f.
* **Trường hợp 2**: Ngoài **childPoint**, có thêm 1 trường hợp là toạ độ cổng dịch chuyển ra. So sánh heuristic giữa **childPoint** và cổng dịch chuyển ra. Nếu như khoảng cách từ cổng dịch chuyển ra đến đích gần hơn thì **childPoint** lúc này là toạ độ cổng dịch chuyển ra (dịch chuyển thành công). Nếu không thì giữ nguyên **childPoint** (không dịch chuyển). Ở trường hợp dịch chuyển thành công thì g giữ nguyên. Tính f.
* Nếu chi phí hiện tại nhỏ hơn chi phí ban đầu thì cập nhật vào open. Đối với **childPoint** là cổng dịch chuyển ra, thêm **currPoint** vào **openTele**.
* Xét lặp lại các bước trên đến khi **open** rỗng hoặc đã tìm được đích.
* Dùng dictionary **path** như 1 con trỏ quay lui lưu vết đi từ đích về nguồn. Kết quả của thuật toán trả về list **res** chứa các values của **path** được đảo ngược để có con đường từ nguồn đến đích.

Định dạng file input với bản đồ có cổng dịch chuyển:

Dòng đầu là số lượng cặp cổng dịch chuyển n có trong bản đồ, n dòng tiếp theo, mỗi dòng gồm 4 số nguyên x, y, x', y' là tọa độ của các cổng dịch chuyển trong bản đồ. (x, y) là cổng dịch chuyển vào, (x', y') là cổng dịch chuyển ra. Trong các file input, maze\_map.txt chứa hình vẽ bản đồ, kí hiệu **i** là vị trí cổng dịch chuyển vào, kí hiệu **o** vị trí cổng dịch chuyển ra.

Các dòng tiếp theo mô tả bản đồ của trò chơi.

***Ghi chú***: màu xanh nhạt (dodgerblue) là cổng dịch chuyển vào, màu xanh đậm hơn (blue) là cổng dịch chuyển ra.

1. **Chạy thuật toán với các bản đồ có cổng dịch chuyển với thuật toán Astar**

|  |  |  |  |
| --- | --- | --- | --- |
| **Bản đồ** | **Chi phí** | **Đường đi** | **Ghi chú** |
| **1 cổng dịch chuyển**  **11 16 2 31** | 30 |  | Tác nhân chọn đường đi có đi ngang qua cổng dịch chuyển để giảm bớt chi phí và rút ngắn được thời gian đến đích. |
| **2 cổng dịch chuyển**  **13 26 2 30**  **13 23 9 4** | 49 |  | Tác nhân lựa chọn cổng vào thứ hai (13, 29) để đến điểm cổng ra (9, 4) gần với đích nhất. Nếu chỉ có một cổng vào (13, 26) thì cũng không lựa chọn đi vào đó vì cổng ra tạo nên đường đi về đích xa hơn. |
| **3 cổng dịch chuyển**  **7 40 9 68**  **9 64 3 51**  **12 7 7 33** | 65 |  | Tác nhân ưu tiên đi vào cổng thứ ba (tọa độ vào (12, 7); tọa độ ra (7, 33)) vì chúng gần vị trí xuất phát và giúp tác nhân đi đến giữa bản đồ. Tác nhân lựa chọn đi vào cổng thứ nhất (tọa độ vào (7, 40); tọa độ ra (9, 69)) và bỏ cổng còn lại vì so với việc đi tiếp một bước bình thường, cổng dịch chuyển lại dẫn tác nhân đến vị trí xa đích hơn. |

1. **Mô tả chiến lược tìm kiếm của các thuật toán với Video cụ thể**
2. **Bản đồ không điểm thưởng maze\_map1.txt**

|  |  |
| --- | --- |
| **Chiến lược** | **Đường đi được mô tả** |
| **BFS** |  |
| **DFS** |  |
| **UCS** |  |
| **Greedy BFS Euclidean** |  |
| **Greedy BFS Manhattan** |  |
| **A\* Euclidean** |  |
| **A\* Manhattan** |  |

1. **Bản đồ không điểm thưởng maze\_map2.txt**

|  |  |
| --- | --- |
| **Chiến lược** | **Đường đi được mô tả** |
| **BFS** |  |
| **DFS** |  |
| **UCS** |  |
| **Greedy BFS Euclidean** |  |
| **Greedy BFS Manhattan** |  |
| **A\* Euclidean** |  |
| **A\* Manhattan** |  |

1. **Bản đồ không điểm thưởng maze\_map3.txt**

|  |  |
| --- | --- |
| **Chiến lược** | **Đường đi được mô tả** |
| **BFS** |  |
| **DFS** |  |
| **UCS** |  |
| **Greedy BFS Euclidean** |  |
| **Greedy BFS Manhattan** |  |
| **A\* Euclidean** |  |
| **A\* Manhattan** |  |

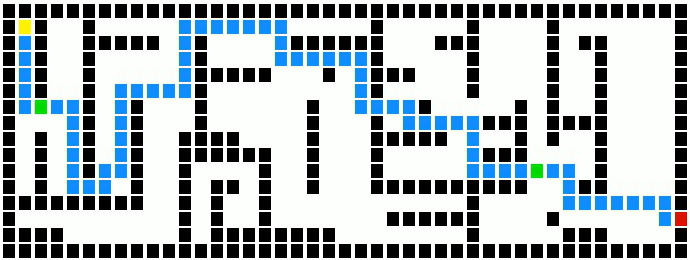
1. **Bản đồ không có điểm thưởng maze\_map4.txt**

|  |  |
| --- | --- |
| **Chiến lược** | **Đường đi được mô tả** |
| **BFS** |  |
| **DFS** |  |
| **UCS** |  |
| **Greedy BFS Euclidean** |  |
| **Greedy BFS Manhattan** |  |
| **A\* Euclidean** |  |
| **A\* Manhattan** |  |

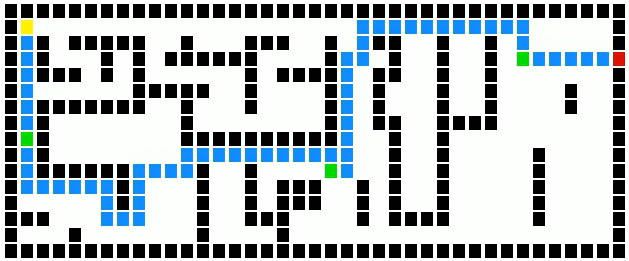
1. **Bản đồ không có điểm thưởng maze\_map5.txt**

|  |  |
| --- | --- |
| **Chiến lược** | **Đường đi được mô tả** |
| **BFS** |  |
| **DFS** |  |
| **UCS** |  |
| **Greedy BFS Euclidean** |  |
| **Greedy BFS Manhattan** |  |
| **A\* Euclidean** |  |
| **A\* Manhattan** |  |

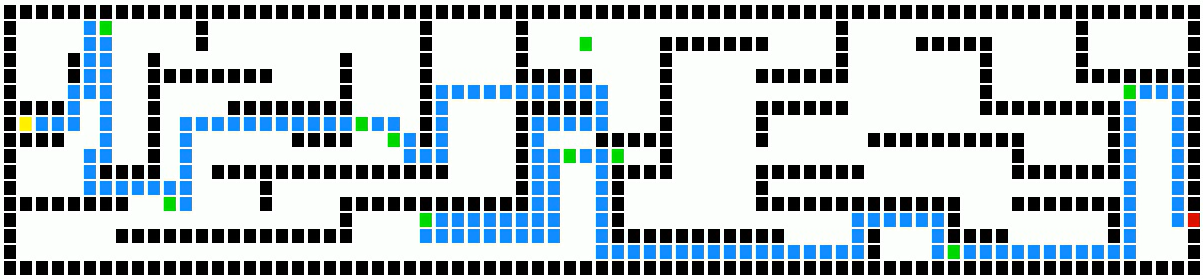
1. **Bản đồ có 2 điểm thưởng maze\_map1.txt**



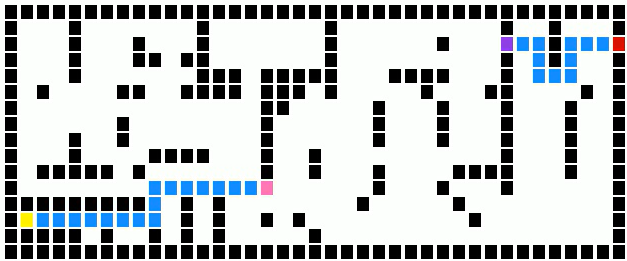
1. **Bản đồ có 5 điểm thưởng maze\_map2.txt**

****

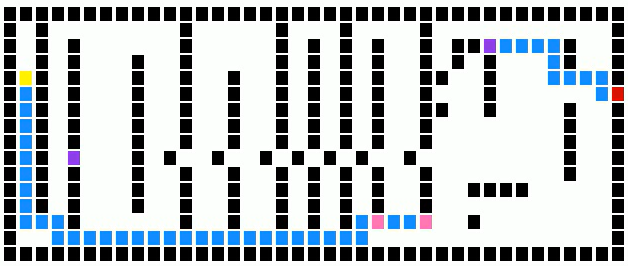
1. **Bản đồ có 10 điểm thưởng maze\_map3.txt**



1. **Bản đồ có 1 cổng dịch chuyển maze\_map1.txt**

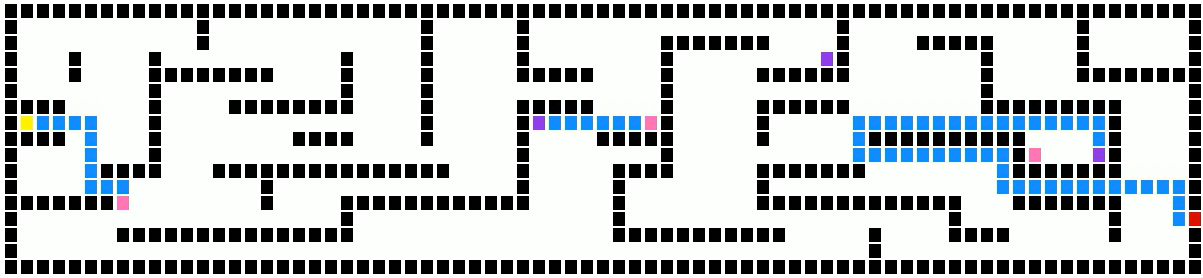
Ghi chú: màu hồng là cổng vào, màu tím là cổng ra

1. **Bản đồ có 2 cổng dịch chuyển maze\_map2.txt**

Ghi chú: màu hồng là cổng vào, màu tím là cổng ra

1. **Bản đồ có 3 cổng dịch chuyển maze\_map3.txt**

Ghi chú: màu hồng là cổng vào, màu tím là cổng ra



1. **Mô tả các câu lệnh trong file run.sh**

Nội dung trong file run.sh:

cd source

chmod +x ./main.py

python main.py

Mô tả:

**cd source**

Thay đổi thư mục hiện tại thành thư mục source

**chmod +x ./main.py**

Thêm quyền thực thi cho file main.py

**python main.py**

Chạy file main.py

Trong file main.py, nhóm đã tích hợp đầy đủ các câu lệnh để chạy toàn bộ thuật toán mà nhóm thiết kế, do đó chỉ cần chạy file main.py là đủ.

1. **Thuật toán khi chạy trên Google Colab**

Thuật toán được nhóm kiểm thử và chạy trên Google Colab:

<https://colab.research.google.com/drive/1WzH6gN_yUsbfQ5B7_9j8tcAMlbUEszbL?usp=sharing>

# **TÀI LIỆU THAM KHẢO**

**Danh mục tài liệu tham khảo:**

[1] Sách Cơ sở trí tuệ nhân tạo – Lê Hoài Bắc, Tô Hoài Việt*.*

[2]<https://datagy.io/manhattan-distance-python/>.

[3]<https://pythonprogramming.net/euclidean-distance-machine-learning-tutorial/>.

[4]<https://medium.com/@enriqueav/how-to-create-video-animations-using-python-and-opencv-881b18e41397>.

[5]<https://youtube.com/playlist?list=PLWF9TXck7O_zsqnufs62t26_LJnLo4VRA>

[6]<https://codelearn.io/sharing/lap-trinh-game-co-ban-voi-pygame>