

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN



MÔN HỌC: CSC14003 – CƠ SỞ TRÍ TUỆ NHÂN TẠO

BÁO CÁO ĐỒ ÁN:

ĐỒ ÁN 1 – ROBOT TÌM ĐƯỜNG

Lớp: 22CLC04

Học sinh:

- Trần Thị Thiên Kim 22127225
- Phạm Nguyên Hải Long 22127248
- Võ Trung Tín 22127417
- Nguyễn Gia Phúc 22127482

Giáo viên hướng dẫn:

- Lê Nguyễn Nhựt Trường

TP. Hồ Chí Minh, ngày 21 tháng 3 năm 2024

MỤC LỤC

PHẦN 1: BÁO CÁO MỨC ĐỘ HOÀN THÀNH	4
PHẦN 2: PHÂN CÔNG CÔNG VIỆC.....	4
PHẦN 3: TỔ CHỨC DỮ LIỆU VÀ GIẢI THUẬT	5
<i>I. Phân tích bài toán.....</i>	<i>5</i>
<i>II. Tổ chức dữ liệu</i>	<i>5</i>
1. Dữ liệu đầu vào:	5
2. Dữ liệu đầu ra:	5
3. Đồ họa	5
4. Thuật toán giải quyết bài toán.....	5
5. Những giới hạn.....	6
<i>III. Mô tả thuật toán</i>	<i>6</i>
1. Mức 1: Cài đặt thuật toán để tìm đường đi từ S tới G	6
2. Mức 2: Cài đặt ít nhất 3 thuật toán và cho thấy sự khác nhau khi chạy thử 3 thuật toán.....	10
3. Mức 3: Trên bản đồ sẽ xuất hiện thêm một số điểm khác gọi là điểm đón. Cài đặt thuật toán tìm ra cách để tổng đường đi là nhỏ nhất.....	18
4. Mức 4: Các hình đa giác có thể di động được với tốc độ h tọa độ/s. Cách thức di động có thể ở mức đơn giản nhất là tới lui một khoảng nhỏ để đảm bảo không đè lên đa giác	20
5. Mức 5: Thể hiện mô hình trên không gian 3 chiều (3D).	21
<i>IV. Cách chạy chương trình</i>	<i>24</i>
NGUỒN THAM KHẢO	26

Lời cảm ơn

Chúng em xin gửi lời cảm ơn sâu sắc và chân thành đến Thầy Lê Nguyễn Nhựt Trường đã tận tâm và nhất quán hỗ trợ chúng em trong suốt quá trình thực hiện đồ án này. Sự chỉ dẫn của Thầy không chỉ giúp chúng em hoàn thành dự án mà còn mang lại cho chúng em những trải nghiệm về những kiến thức mới.

Thầy là nguồn động viên lớn giúp chúng em vượt qua những thách thức trong quá trình nghiên cứu, phân tích và triển khai dự án. Sự kiên nhẫn và tận tâm của Thầy đã giúp chúng em hiểu sâu hơn về lĩnh vực chúng em đang nghiên cứu và phát triển kỹ năng làm việc nhóm.

Chúng em cũng muốn bày tỏ lòng biết ơn đặc biệt đến những góp ý chi tiết và quan trọng của Thầy, giúp chúng em điều chỉnh và hoàn thiện dự án một cách chặt chẽ hơn. Điều này không chỉ làm tăng chất lượng của đồ án mà còn là nguồn động viên quan trọng cho sự phát triển cá nhân của chúng em.

Cuối cùng, chúng em muốn bày tỏ lòng biết ơn chân thành đến sự cam kết và lòng nhiệt thành của Thầy trong việc chia sẻ kiến thức và kinh nghiệm giảng dạy. Điều này làm cho quá trình học tập của chúng em trở nên ý nghĩa và phong phú hơn.

PHẦN 1: BÁO CÁO MỨC ĐỘ HOÀN THÀNH

Chức năng	Mức độ hoàn thành
Mức 1	100 %
Mức 2	100 %
Mức 3	100 %
Mức 4	100 %
Mức 5	100 %

PHẦN 2: PHÂN CÔNG CÔNG VIỆC

Bảng phân công công việc			
STT	Họ và tên	Nội dung được phân công	Mức độ hoàn thành
1	Nguyễn Gia Phúc	1. Thực hiện đồ họa 2D cho mức 1, 2, 3 2. Thực hiện đồ họa 3D cho mức 5 3. Thiết kế file input.txt cho mức 1, 2	100%
2	Trần Thị Thiên Kim	1. Lập trình thuật toán GBFS 2. Thiết kế file input.txt cho mức 3 3. Thiết kế file báo cáo đồ án	100%
3	Phạm Nguyên Hải Long	1. Lập trình thuật toán A* 2. Thực hiện mức 3 với thuật toán A* 3. Thực hiện mức 4 objects di chuyển với thuật toán A*	100%
4	Võ Trung Tín	1. Lập trình thuật toán Dijkstra 2. Thiết kế file main.py giao diện người dùng trên terminal 3. Hỗ trợ làm mức 4	100%

PHẦN 3: TỔ CHỨC DỮ LIỆU VÀ GIẢI THUẬT

I. Phân tích bài toán

Căn cứ vào những dữ liệu và yêu cầu của bài toán là tìm đường đi ngắn nhất dựa vào 2 điểm là điểm bắt đầu $S(x,y)$ và điểm kết thúc $G(x,y)$ cho trước, đồng thời có các chướng ngại vật là các hình đa giác lồi sao cho các đa giác này không được đặt chồng lên nhau. Ta sẽ đi tìm đường đi ngắn nhất từ điểm bắt đầu đến điểm kết thúc sao cho không có đường đi cắt xuyên qua các đa giác.

II. Tổ chức dữ liệu

1. Dữ liệu đầu vào:

- Dòng đầu là giới hạn của không gian, được mô tả lần lượt bởi kích thước ngang, kích thước dọc và kích thước cao (nếu có).
- Dòng thứ hai lần lượt là tọa độ điểm bắt đầu, tọa độ điểm kết thúc và tập hợp các điểm đón (nếu có).
- Dòng thứ ba là số lượng các đa giác có trong không gian.
- Các dòng tiếp theo, mỗi dòng chứa một đa giác theo quy tắc:
 - Đa giác là tập hợp các điểm kề nhau theo chiều kim đồng hồ. Điểm cuối cùng sẽ được hiểu ngầm là sẽ được nối đến điểm đầu tiên để tạo thành một đa giác lồi hợp lệ.
 - Mỗi số trong dữ liệu cách nhau bởi dấu phẩy.

2. Dữ liệu đầu ra:

- Đồ họa biểu diễn đa giác và đường đi.
- Chi phí

3. Đồ họa

- Sử dụng thư viện plotly để tạo đồ thị 3D, vẽ các hình khối đại diện cho các vật cản trong bản đồ 3D
- Sử dụng thư viện matplotlib để vẽ hình 2D
- Sử dụng thư viện imageio để tạo file .gif mô tả quá trình chạy thuật toán mức 4

4. Thuật toán giải quyết bài toán

- Thuật toán tìm kiếm tham lam (Greedy Best First Search)
- Thuật toán Dijkstra
- Thuật toán A* Search

5. Những giới hạn

- Các hình đa giác vẽ tương đối khi di chuyển từ 1 đỉnh đến đỉnh khác, sao cho đó là đường di chuyển ngắn nhất giữa hai đỉnh kề nhau. Không cho đan với đa giác khác khi hai đa giác tiếp xúc nhau.
- Để hạn chế việc đi xuyên qua đa giác, ứng dụng các giải pháp sau:
 - Thuật toán cho phép đi chéo
 - Thuật toán giúp phát hiện đường đi cắt vào các đa giác:
 - *Thuật toán GBFS*: Dùng thuật toán Ray Casting để xác định điểm đang xét có nằm trong đa giác hay không. Nó duyệt qua tất cả các cạnh của đa giác và xác định xem điểm có nằm trong đa giác bằng cách kiểm tra số lần giao nhau giữa đường thẳng song song với trục x và các cạnh đa giác.
 - *Thuật toán Dijkstra*: Kiểm tra về vị trí tiếp theo (x,y) trên ma trận đồ thị. Nếu điểm này không nằm ngoài giới hạn ma trận $0 \leq x < \text{cols}$ and $0 \leq y < \text{rows}$ và không nằm trong object (được đại diện bởi giá trị **1** trong ma trận `self.graph`), nó sẽ được thêm vào danh sách các điểm lân cận. Do đó, nếu cố gắng đi vào các đa giác, điểm đó sẽ không được thêm vào các danh sách, giúp phát hiện khi đi vào các đa giác lỗi.
 - *Thuật toán A**: Trong hàm `is_unblocked`, kiểm tra xem một điểm trên ma trận có phải là điểm không bị chặn hay không. Nếu giá trị của ô trong ma trận tương ứng với điểm đó là **2, 3, 4** hoặc **0**, tức là điểm đó không bị chặn, thuật toán sẽ tiếp tục xem xét điểm đó. Sau khi kiểm tra điều kiện về điểm hợp lệ và không bị chặn, thuật toán sẽ đi tìm đường đi từ điểm xuất phát đến điểm đích trên ma trận. Trong quá trình này, các ô trên ma trận sẽ được duyệt và kiểm tra tính hợp lệ và không bị chặn.

III. Mô tả thuật toán

1. Mức 1: Cài đặt thuật toán để tìm đường đi từ S tới G

❖ Thuật toán Greedy Best First Search

i. Giới thiệu thuật toán

Thuật toán Greedy Best First Search là một thuật toán tìm kiếm trong không gian trạng thái, nó sử dụng chiến lược tìm kiếm tốt nhất đầu tiên (Best-First) và là một biến thể của thuật toán tìm kiếm đầu tiên tốt nhất (Best-First Search).

Trong thuật toán GBFS, mỗi bước di chuyển, nó chọn trạng thái tiếp theo dựa trên một hàm ước lượng chi phí từ trạng thái hiện tại đến trạng thái đích (heuristic function).

GBFS ưu tiên mở rộng các nút mà được ước lượng là gần đích nhất, mà không quan tâm đến chi phí tính từ trạng thái ban đầu.

ii. Ứng dụng của thuật toán trong bài toán tìm đường đi

Trong bài toán tìm đường đi từ điểm S đến điểm G trên một bản đồ chứa các chướng ngại vật là các đa giác lồi, GBFS có thể được áp dụng để tìm kiếm đường đi ngắn nhất. Trong quá trình tìm kiếm, GBFS sẽ ưu tiên các đường đi mà có khả năng tiếp cận gần với điểm đích G nhất, dựa trên hàm heuristic, mà không cắt xuyên qua các đa giác lồi.

iii. Quá trình chạy thử

Trong quá trình chạy thử, chúng ta sẽ sử dụng GBFS để tìm đường đi từ một điểm bắt đầu S đến một điểm đích G trên một bản đồ chứa các đa giác lồi.

Bước 1: Khởi tạo GBFS với điểm bắt đầu S và điểm đích G.

Bước 2: Áp dụng hàm *eulidean* để tính khoảng cách giữa hai điểm trên mặt phẳng. Khoảng cách Euclid giữa hai điểm được tính bằng căn bậc hai của tổng bình phương của hiệu của các tọa độ x và y của hai điểm.

Bước 3: Lặp lại quá trình tìm kiếm cho đến khi tìm thấy đường đi từ S đến G hoặc không còn trạng thái nào để kiểm tra.

Bước 4: Kiểm tra xem đường đi có hợp lệ không, tức là không cắt xuyên qua các đa giác lồi. Sử dụng thuật toán Ray Casting:

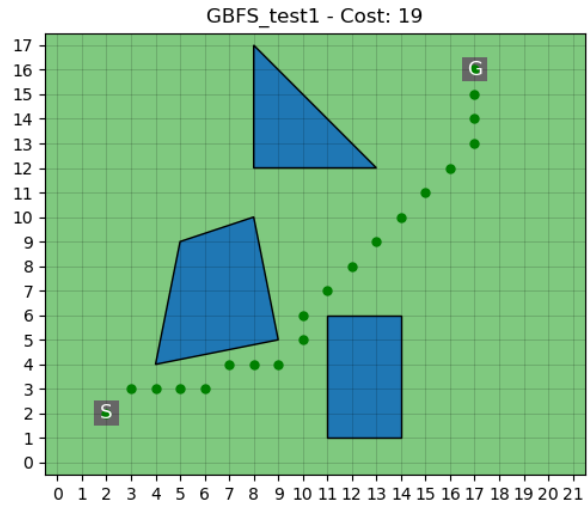
- Vẽ một đoạn thẳng từ điểm cần kiểm tra tới một điểm vô cùng xa bên ngoài đa giác và đếm số lần đoạn thẳng này giao với cạnh của đa giác.
- Nếu số lần giao là số lẻ, điểm đó nằm bên trong đa giác, ngược lại nếu số lần giao là số chẵn, điểm đó nằm ngoài đa giác.

```
for i in range(len(polygon)):  
    xi, yi = polygon[i][:2]  
    xj, yj = polygon[j][:2]  
    if (yi < y and yj >= y) or (yj < y and yi >= y):  
        if (xi + ((y - yi) / (yj - yi)) * (xj - xi) < x:  
            oddNodes = not oddNodes  
    j = i
```

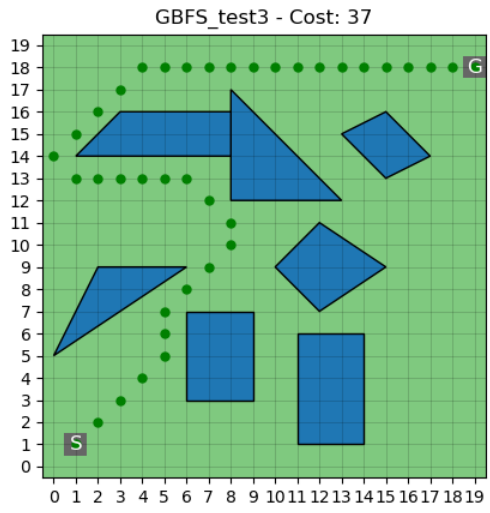
Bước 5: Trong quá trình tìm kiếm, nếu gặp một điểm mới không nằm trong các vùng cấm (được xác định bởi các đa giác), nó được thêm vào hàng đợi ưu tiên. Nếu đường đi là hợp lệ, in ra đường đi và kết thúc quá trình tìm kiếm. Nếu không, tiếp tục tìm kiếm.

iv. *Chạy thử*

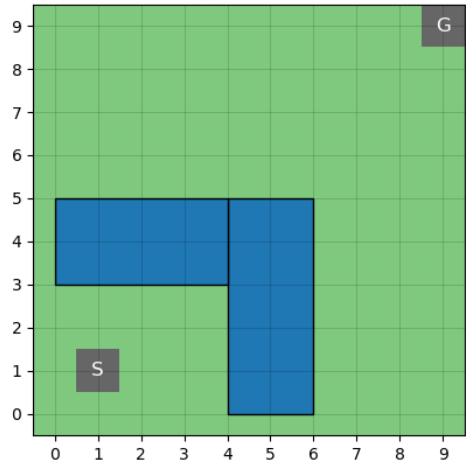
Trường hợp đồ thị xuất hiện các đa giác cơ bản:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 22,18 2 2,2,17,16 3 3 4 4,4,5,9,8,10,9,5 5 8,12,8,17,13,12 6 11,1,11,6,14,6,14,1 </pre>	 <p>The plot shows a 2D grid with x-axis from 0 to 21 and y-axis from 0 to 17. The start point 'S' is at (2, 2) and the goal point 'G' is at (17, 16). The path consists of green dots: (2,2), (3,2), (4,2), (5,2), (6,2), (7,2), (8,2), (9,2), (10,2), (10,3), (11,3), (11,4), (12,4), (12,5), (13,5), (14,5), (15,5), (16,5), (16,6), (16,7), (16,8), (16,9), (16,10), (16,11), (16,12), (16,13), (16,14), (16,15), (16,16). Obstacles are blue polygons: a triangle with vertices (8,12), (12,12), (8,16); a quadrilateral with vertices (4,4), (10,4), (10,9), (4,9); and a rectangle with vertices (11,1), (14,1), (14,6), (11,6).</p>

Trường hợp đồ thị xuất hiện nhiều đa giác phức tạp:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 20,20 2 1,1,19,18 3 7 4 2,9,6,9,0,5 5 8,17,13,12,8,12 6 11,1,11,6,14,6,14,1 7 10,9,12,11,15,9,12,7 8 1,14,3,16,8,16,8,14 9 13,15,15,16,17,14,15,13,14,14 10 6,3,6,7,9,7,9,3 </pre>	 <p>GBFS_test3 - Cost: 37</p>

Trường hợp không có đường đi:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 10,10 2 1,1,9,9 3 2 4 4,0,4,5,6,5,6,0 5 4,3,0,3,0,5,4,5 </pre>	 <p>GBFS_test5 - Cost: 0</p>

v. *Ưu và nhược điểm*

Ưu điểm:

- GBFS có thể hoạt động nhanh chóng trong các bài toán có không gian trạng thái lớn với một số lượng lớn các trạng thái có thể.
- Nó tiết kiệm bộ nhớ so với các thuật toán tìm kiếm thông tin đầy đủ như A* khi không lưu trữ thông tin chi phí.

Nhược điểm:

- Không đảm bảo tìm ra giải pháp tối ưu.
- Có thể bị mắc kẹt trong vòng lặp vô hạn nếu không có giới hạn trong việc lựa chọn nút tiếp theo.
- Rất nhạy cảm với lựa chọn hàm heuristic, có thể dẫn đến các lựa chọn không tối ưu.

vi. *Nhận xét*

Thuật toán Greedy Best-First Search là một lựa chọn hiệu quả để giải quyết bài toán tìm đường đi từ điểm S đến điểm G trên một bản đồ không cắt vào các đa giác lồi. Tuy nhiên, cần lưu ý rằng thuật toán GBFS có thể không đảm bảo tìm ra đường đi tối ưu nhất, nhưng nó có thể được cải thiện thông qua việc sử dụng hàm heuristic phù hợp.

2. **Mức 2: Cài đặt ít nhất 3 thuật toán và cho thấy sự khác nhau khi chạy thử 3 thuật toán**

❖ **Thuật toán Dijkstra**

i. *Giới thiệu thuật toán*

Thuật toán Dijkstra là một thuật toán trong lĩnh vực tối ưu hóa đường đi trong đồ thị. Thuật toán được sử dụng để tìm đường đi ngắn nhất giữa một đỉnh xuất phát và tất cả các đỉnh còn lại trong đồ thị có trọng số không âm.

Cốt lõi của thuật toán là duyệt qua các đỉnh của đồ thị, cập nhật các đường đi tốt nhất từ đỉnh xuất phát đến các đỉnh khác thông qua việc lựa chọn các cạnh có trọng số nhỏ nhất. Thuật toán duyệt qua các đỉnh một cách có hệ thống và không duyệt lại các đỉnh đã được xử lý trước đó.

ii. *Ứng dụng của thuật toán trong bài toán tìm đường đi*

Thuật toán Dijkstra được áp dụng để tìm đường đi ngắn nhất từ điểm bắt đầu đến điểm kết thúc trong đồ thị. Trong quá trình này, thuật toán sẽ tránh các cạnh mà nếu đi cắt xuyên vào các đa giác. Các cạnh của đồ thị được xác định bằng cách tìm các đoạn đường từ một đỉnh đến các đỉnh khác mà không cắt xuyên vào đa giác. Cách này có thể được

thực hiện bằng cách sử dụng thuật toán hoặc kỹ thuật phát hiện va chạm. Điều này đảm bảo rằng con đường tìm được là một đường đi hợp lệ mà không cắt xuyên vào các vùng bị cấm.

iii. Quá trình chạy thử

Bước 1: Khởi tạo Dijkstra với điểm bắt đầu `S` và điểm đích `G`.

Bước 2: Khởi tạo ma trận `dist` chứa khoảng cách từ điểm `start` tới các điểm khác

Bước 3: Khởi tạo một hàng đợi ưu tiên (priority queue) để lựa chọn nút kế tiếp có khoảng cách nhỏ nhất từ `start`. Duyệt qua các nút lân cận của nút hiện tại, cập nhật khoảng cách và thêm vào hàng đợi ưu tiên.

```
def get_neighbors(self, node):
    rows, cols = self.graph.shape
    neighbors = []
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1),
                  (-1, 1), (1, -1), (-1, -1)]
    for dx, dy in directions:
        x, y = node[1] + dx, node[0] + dy
        if 0 <= x < cols and 0 <= y < rows and
self.graph[y][x] != 1:
            neighbors.append((y, x))

    return neighbors
```

Bước 4: Lặp lại quá trình tìm kiếm cho đến khi hàng đợi ưu tiên trống.

Bước 5: Sau khi tìm thấy đường đi từ `S` đến `G`, kiểm tra xem đường đi có cắt xuyên vào các đa giác lỗi không. Điều này có thể được thực hiện bằng cách kiểm tra từng cạnh trên đường đi.

```
if 0 <= x < cols and 0 <= y < rows and self.graph[y][x] != 1:
```

Bước 6: Nếu đường đi là hợp lệ, in ra đường đi và kết thúc quá trình tìm kiếm. Ngược lại tiếp tục tìm kiếm bằng cách chọn đỉnh tiếp theo thỏa mãn yêu cầu bài toán.

iv. Chạy thử

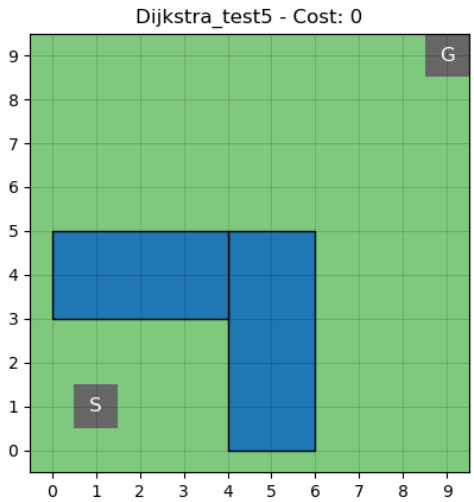
Trường hợp đồ thị xuất hiện các đa giác cơ bản:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 22,18 2 2,2,17,16 3 3 4 4,4,5,9,8,10,9,5 5 8,12,8,17,13,12 6 11,1,11,6,14,6,14,1 </pre>	

Trường hợp đồ thị xuất hiện nhiều đa giác phức tạp:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 20,20 2 1,1,19,18 3 7 4 2,9,6,9,0,5 5 8,17,13,12,8,12 6 11,1,11,6,14,6,14,1 7 10,9,12,11,15,9,12,7 8 1,14,3,16,8,16,8,14 9 13,15,15,16,17,14,15,13,14,14 10 6,3,6,7,9,7,9,3 </pre>	

Trường hợp không có đường đi:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 10,10 2 1,1,9,9 3 2 4 4,0,4,5,6,5,6,0 5 4,3,0,3,0,5,4,5 </pre>	

v. Ưu và nhược điểm

Ưu điểm:

- Dijkstra đảm bảo tìm ra đường đi ngắn nhất từ một điểm xuất phát đến tất cả các điểm còn lại trên đồ thị, trong trường hợp các trọng số trên cạnh không âm.
- Thuật toán Dijkstra có cấu trúc đơn giản và dễ hiểu, không yêu cầu kiến thức toán học phức tạp để triển khai.

Nhược điểm:

- Dijkstra không thể xử lý đồ thị có trọng số âm, vì nó dựa vào việc chọn nút có khoảng cách nhỏ nhất từ điểm xuất phát, và trọng số âm có thể gây ra vòng lặp vô hạn.
- Dijkstra yêu cầu lưu trữ ma trận khoảng cách và hàng đợi ưu tiên, vì vậy nó có thể tiêu tốn nhiều bộ nhớ đối với đồ thị lớn.
- Nếu đồ thị có chu trình âm, thuật toán không thể cho kết quả chính xác, và có thể gây ra lỗi vòng lặp vô hạn.
- Dijkstra chỉ tìm đường đi ngắn nhất, không linh hoạt trong việc tìm kiếm các loại đường đi khác như đường đi ngắn nhất qua một số điểm hoặc đường đi có trọng số nhỏ nhất.

vi. *Nhận xét*

Thuật toán Dijkstra đảm bảo tìm ra đường đi ngắn nhất từ điểm bắt đầu đến điểm kết thúc. Tuy nhiên, thuật toán không thể xử lý được các đồ thị có trọng số âm mà không cần điều chỉnh. Khi có trọng số âm, nó có thể dẫn đến việc tìm ra đường đi không tối ưu hoặc gây ra các vấn đề khác.

❖ **Thuật toán A***

i. *Giới thiệu thuật toán*

Thuật toán A* là một thuật toán tìm kiếm trong đồ thị. Thuật toán này tìm một đường đi từ nút khởi đầu tới một nút đích cho trước, sử dụng hàm đánh giá heuristic để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó. Thuật toán A* là một ví dụ của tìm kiếm theo lựa chọn tốt nhất (Best-First Search).

ii. *Ứng dụng của thuật toán trong bài toán tìm đường đi*

Thuật toán A* được áp dụng trong bài toán tìm đường đi ngắn nhất từ điểm bắt đầu đến điểm kết thúc trong một môi trường không gian 2D với các vật cản là các đa giác lồi. Các ràng buộc như tránh vật cản, tránh va chạm và tối ưu hóa thời gian hoặc chi phí có thể được tính toán và tích hợp vào hàm heuristic để tạo ra đường đi an toàn và hiệu quả.

iii. *Quá trình chạy thử*

Bước 1: Khởi tạo một đối tượng AStart với các thông số đầu vào là ma trận, điểm bắt đầu và điểm kết thúc.

Bước 2: Trong quá trình tìm kiếm, thuật toán sẽ kiểm tra tính hợp lệ và không chặn của ô trong ma trận bản đồ. Hai hàm `is_valid` & `is_unblocked` kiểm tra lần lượt xem một ô có nằm trong phạm vi của ma trận không và kiểm tra xem ô có bị chặn bởi các vật cản hay không.

Bước 3: Thuật toán tiếp tục tìm kiếm và mở rộng các ô tiếp theo cho đến khi tìm thấy đường đi từ điểm bắt đầu đến điểm kết thúc hoặc không còn ô nào để kiểm tra. Hai hàm kiểm tra xem một ô có đến đích và tính toán giá trị heuristic `is_destination` & `calculate_h_value`, tính toán giá trị h thường sử dụng công thức khoảng cách Euclid.

Bước 4: Nếu đường đi tìm thấy là hợp lệ, in ra đường đi. Ngược lại, tiếp tục tìm kiếm các ô thỏa mãn yêu cầu bài toán đưa ra hoặc đến khi không còn ô nào để kiểm tra.

iv. Chạy thử

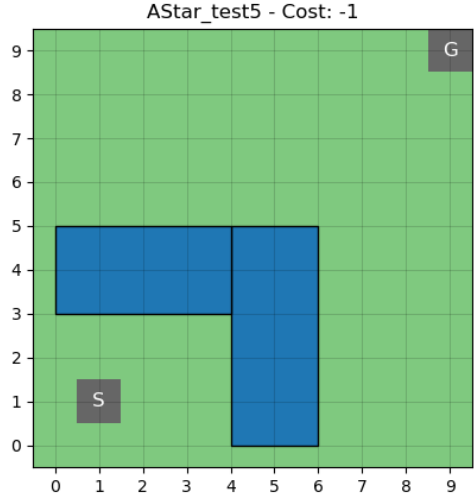
Trường hợp đồ thị xuất hiện các đa giác cơ bản:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 22,18 2 2,2,17,16 3 3 4 4,4,5,9,8,10,9,5 5 8,12,8,17,13,12 6 11,1,11,6,14,6,14,1 </pre>	<p>The figure shows a 2D grid map with a green background. The x-axis ranges from 0 to 21, and the y-axis ranges from 0 to 17. There are several blue obstacles: a triangle at (8,12) to (12,12) to (8,16), a quadrilateral at (4,4) to (9,4) to (10,9) to (4,9), and a rectangle at (11,1) to (14,1) to (14,6) to (11,6). A path of green dots starts at 'S' (2,3) and ends at 'G' (17,16). The path goes from S to (3,3), (4,3), (5,3), (6,3), (7,4), (8,4), (9,4), (10,5), (11,6), (12,7), (13,8), (14,9), (15,10), (16,11), (17,12), (17,13), (17,14), (17,15), and finally to G at (17,16). The title is 'AStar_test1 - Cost: 18'.</p>

Trường hợp đồ thị xuất hiện các đa giác phức tạp:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 20,20 2 1,1,19,18 3 7 4 2,9,6,9,0,5 5 8,17,13,12,8,12 6 11,1,11,6,14,6,14,1 7 10,9,12,11,15,9,12,7 8 1,14,3,16,8,16,8,14 9 13,15,15,16,17,14,15,13,14,14 10 6,3,6,7,9,7,9,3 </pre>	<p>The figure shows a 2D grid map with a green background. The x-axis ranges from 0 to 19, and the y-axis ranges from 0 to 19. There are several blue obstacles: a triangle at (1,5) to (4,5) to (1,9), a rectangle at (6,4) to (9,4) to (9,7) to (6,7), a rectangle at (11,1) to (14,1) to (14,6) to (11,6), a triangle at (12,12) to (17,12) to (12,16), a rectangle at (1,14) to (9,14) to (9,16) to (1,16), a diamond at (10,9) to (13,9) to (13,12) to (10,12), and a diamond at (14,14) to (17,14) to (17,16) to (14,16). A path of green dots starts at 'S' (1,1) and ends at 'G' (19,18). The path goes from S to (2,2), (3,3), (4,4), (5,3), (6,2), (7,2), (8,2), (9,2), (10,1), (11,0), (12,0), (13,0), (14,0), (15,1), (16,2), (17,3), (18,4), (18,5), (18,6), (18,7), (18,8), (18,9), (18,10), (18,11), (18,12), (18,13), (18,14), (18,15), (18,16), (18,17), and finally to G at (19,18). The title is 'AStar_test3 - Cost: 30'.</p>

Trường hợp không có đường đi:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 10,10 2 1,1,9,9 3 2 4 4,0,4,5,6,5,6,0 5 4,3,0,3,0,5,4,5 </pre>	

v. Ưu và nhược điểm

Ưu điểm:

- Thuật toán cho kết quả tối ưu nhanh chóng trên đa số bài toán tìm đường đi trên bản đồ lưới.
- Khi sử dụng hàm heuristic hợp lý, A* có thể tìm ra đường đi tối ưu từ điểm xuất phát đến điểm đích.
- A* thường tiêu tốn ít bộ nhớ hơn do chỉ cần lưu trữ các ô được mở rộng và các chi phí tương ứng.

Nhược điểm:

- Trong một số trường hợp, A* có thể không đảm bảo tìm ra đường đi tối ưu toàn cục nếu hàm heuristic không đủ chính xác.
- A* không phải lúc nào cũng hiệu quả trên các đồ thị với các trọng số âm hoặc các bài toán đòi hỏi sự phức tạp trong việc xử lý các ràng buộc và hạn chế.
- Hiệu suất của A* phụ thuộc rất nhiều vào chất lượng của hàm heuristic được sử dụng. Nếu hàm heuristic không chính xác, thuật toán có thể không tìm ra đường đi tối ưu.

vi. *Nhận xét*

Thuật toán A* thường cho kết quả tốt trong việc tìm đường đi ngắn nhất giữa hai điểm trong đồ thị, đặc biệt là trong bài toán tìm đường đi trên lưới 2D. Thuật toán sử dụng một hàm heuristic để ước lượng chi phí còn lại từ một nút đến đích, giúp thuật toán ưu tiên các nút có xác suất cao nhất để đưa đến đích trước tiên. Độ phức tạp thời gian của A* phụ thuộc vào cả kích thước của đồ thị và chất lượng của hàm heuristic, Tuy nhiên, trong thực tế, A* thường chạy hiệu quả trên các bài toán thực tế với đồ thị lớn.

❖ *So sánh thuật toán*

Đặc điểm	GBFS	Dijkstra	A*
<i>Ý tưởng chính</i>	Tìm kiếm theo chiều sâu, ưu tiên các nút gần điểm đích nhất dựa trên hàm heuristic	Tìm kiếm theo chiều rộng, tính toán đường đi ngắn nhất từ điểm bắt đầu đến tất cả các điểm còn lại trong đồ thị	Kết hợp giữa GBFS và Dijkstra, sử dụng hàm heuristic để ước lượng chi phí còn lại từ nút hiện tại đến điểm đích
<i>Sử dụng heuristic</i>	Có	Không	Có
<i>Sử dụng chi phí đường đi thực tế</i>	Không	Có	Có
<i>Đảm bảo tìm đường đi ngắn nhất</i>	Không	Có	Có
<i>Độ phức tạp</i>	Tùy thuộc vào hàm heuristic, có thể nhanh nếu có hàm heuristic tốt	Cao, tìm đường đi ngắn nhất đến tất cả các điểm	Tùy thuộc vào hàm heuristic, thường nhanh nếu có hàm heuristic tốt
<i>Khả năng tối ưu hóa</i>	Không	Có	Có
<i>Yêu cầu về hàm heuristic</i>	Cần hàm heuristic tốt	Không yêu cầu hàm heuristic	Cần hàm heuristic tốt

⇒ Thuật toán GBFS có thể không tìm ra đường đi ngắn nhất và có thể bị mắc kẹt trong vòng lặp vô hạn nếu không có giới hạn.

- ⇒ Thuật toán Dijkstra đảm bảo tìm ra đường đi ngắn nhất từ điểm bắt đầu đến tất cả các điểm khác trong đồ thị nếu có, nhưng có thể tốn nhiều thời gian và tài nguyên tính toán nếu đồ thị lớn.
- ⇒ Thuật toán A* sử dụng chi phí thực tế từ điểm bắt đầu đến nút hiện tại (g) và ước lượng chi phí từ nút hiện tại đến điểm đích (h), điều này giúp thuật toán tập trung vào việc mở các nút có giá trị f nhỏ nhất, tức là các nút có tiềm năng là đường đi tốt nhất.

3. Mức 3: Trên bản đồ sẽ xuất hiện thêm một số điểm khác gọi là điểm đón. Cài đặt thuật toán tìm ra cách để tổng đường đi là nhỏ nhất

❖ Thuật toán A* search với các điểm đón

- i. *Thuật toán áp dụng*
 - Thuật toán A*
- ii. *Quá trình chạy thử*

Bước 1: Khởi tạo Astar với ma trận đồ thị, điểm bắt đầu, điểm đích và các điểm đón.

Bước 2: Sử dụng thuật toán A* để tìm kiếm chi phí đi từ mỗi ô trong ma trận tới các ô khác. Giá trị cost tìm được sẽ lưu vào mảng 2D.

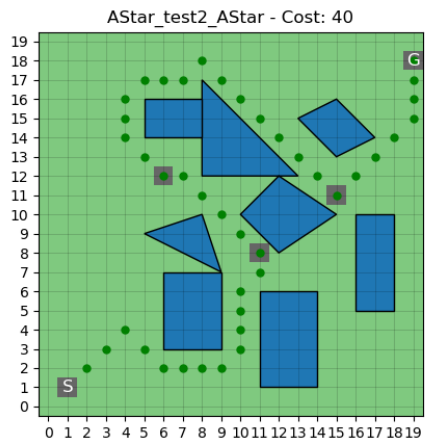
Bước 3: Để tìm đường đi ngắn nhất, ta sẽ lập ra các hoán vị đỉnh. Và xét từng trường hợp hoán vị, tìm xem hoán vị nào sẽ cho ra chi phí thấp nhất.

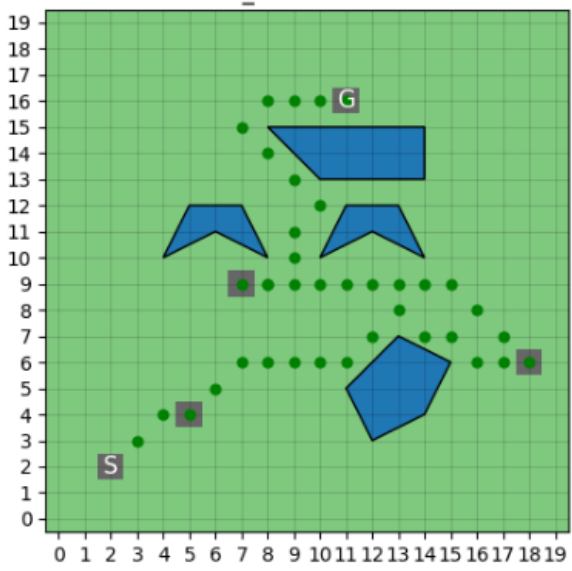
Bước 4: Tính toán chi phí các đường đi, được lập ra từ hoán vị đỉnh, từ kết quả ma trận 2 chiều vừa lập được, và tìm ra đường đi.

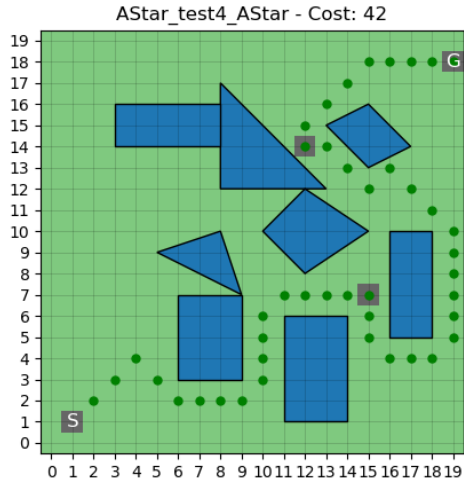
Bước 5: Sử dụng thuật toán A* để tìm ra đường đi cụ thể giữa các điểm trong danh sách đường đi vừa tìm được ở mức 4 và vẽ lên màn hình.

- iii. *Chạy thử*

Trường hợp đồ thị xuất hiện các đa giác cơ bản:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 20,20 2 1,1,19,18,11,8,6,12,15,11 3 8 4 5,9,8,10,9,7 5 8,17,13,12,8,12 6 11,1,11,6,14,6,14,1 7 10,10,12,12,15,10,12,8 8 5,14,5,16,8,16,8,14 9 13,15,15,16,17,14,15,13,14,14 10 6,3,6,7,9,7,9,3 11 16,5,16,10,18,10,18,5 </pre>	 <p>AStar_test2_AStar - Cost: 40</p>

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 20,20 2 2,2,11,16,5,4,18,6,7,9 3 4 4 10,10,12,11,14,10,13,12,11,12 5 12,3,14,4,15,6,13,7,11,5 6 4,10,6,11,8,10,7,12,5,12 7 10,13,8,15,14,15,14,13 </pre>	 <p>AStar_test3 - Cost: 37</p>

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 20,20 2 1,1,19,18,15,7,12,14 3 8 4 5,9,8,10,9,7 5 8,17,13,12,8,12 6 11,1,11,6,14,6,14,1 7 10,10,12,12,15,10,12,8 8 3,14,3,16,8,16,8,14 9 13,15,15,16,17,14,15,13,14,14 10 6,3,6,7,9,7,9,3 11 16,5,16,10,18,10,18,5 </pre>	

iv. Nhận xét

Cách giải mức 3 dựa theo nguyên lý Travelling salesman để tìm ra được đường đi tối ưu nhất khi thuật toán phải đi đón những điểm đón trên map.

Ngoài ra để có thể tạo ra được những hoán vị, chương trình mức 3 sử dụng thư viện itertools để hỗ trợ tạo ra các hoán vị nhanh hơn và ít phức tạp hơn.

4. Mức 4: Các hình đa giác có thể di động được với tốc độ h tọa độ/s. Cách thức di động có thể ở mức đơn giản nhất là tối lui một khoảng nhỏ để đảm bảo không đè lên đa giác

i. Thuật toán áp dụng

- Thuật toán A*

ii. Hướng giải quyết

- Vẽ các hình di chuyển

- Đưa ra hướng di chuyển cho mỗi objects. Ở đây chương trình sẽ cho các objects đi theo chiều kim đồng hồ.
- Lưu hướng di chuyển vào mảng directions
- Cập nhật các tọa độ cho hình liên tục cho vòng lặp while.
- Sau đó liên tục vẽ ma trận với objects tọa độ mới.

- *Chạy thuật toán với các hình khối di chuyển*

- Chương trình khi đã biết được hướng di chuyển của objects sẽ dựa vào đường đi mà objects di chuyển tạo ra một khu vực mà agent sẽ không được đi vào.
- Khi đó khi chạy thuật toán agent sẽ không đưa ra đường đi vi phạm khu vực mà các objects di chuyển tới.

iii. *Quá trình chạy thử*

Bước 1: Khởi tạo Astar với ma trận đồ thị, điểm bắt đầu, điểm đích và các điểm đón.

Bước 2: Cho dx, dy chạy trong danh sách directions và liên tục cập nhật tọa độ cho các objects và vẽ liên tục trên một file .png để tạo ra effect di chuyển

Bước 3: Lưu vết những ô di chuyển của các objects và đánh dấu bằng số 1

Bước 4: Sau đó chạy thuật toán A* trên ma trận mới lập ra từ việc đánh số 1 trên những ô di chuyển của objects

Bước 5: Lưu đường đi tìm được từ việc chạy thuật toán A* vào một danh sách

Bước 6: Vẽ từng ô đường đi được lưu trong danh sách ở mức 5

Bước 7: Lưu các hình render được từ chương trình vào một mảng trống và sử dụng thư viện imageio để tạo ra file gif mô phỏng quá trình chạy thuật toán

iv. *Nhận xét*

Khi objects di chuyển để không làm ảnh hưởng tới quá trình chạy thuật toán, ta có thể đánh dấu hướng di chuyển của các objects và tạo ra những vùng cấm từ đó agents khi chạy thuật toán sẽ không vi phạm ô di chuyển của objects.

5. **Mức 5: Thể hiện mô hình trên không gian 3 chiều (3D).**

❖ **Cấu trúc file input**

File input được thiết kế giống như file input của không gian 2D nhưng được thêm một vài thông tin bổ sung để thể hiện mô hình 3D:

- Dòng đầu tiên được thêm một số chỉ chiều cao của không gian.
- Các đỉnh trong đa giác vật cản được thể hiện theo tọa độ 3D (x,y,z).

Ví dụ:

```
1      22,22,22
2      2,2,17,16
3      3
4      4,4,0,5,9,0,10,10,0,7,3,0,5,10,9
5      8,12,0,8,17,0,13,12,0,10,15,3
6      11,1,0,11,6,0,14,6,0,14,1,10,8,2,5
```

❖ Tổ chức các class dữ liệu phục vụ cho không gian 3D

Các class dữ liệu của không gian 3D được xây dựng bằng cách kế thừa các class dữ liệu của không gian 2D: Map_3D kế thừa Map, MapInfo_3D kế thừa MapInfo, MapGenerator_3D kế thừa MapGenerator.

MapInfo_3D được bổ sung một biến lưu chiều cao không gian và một danh sách các vật cản được biểu diễn theo tọa độ 3D. Khi đọc từ file input vào, các thông tin về tọa độ các đỉnh của vật cản sẽ được lưu trong danh sách vật cản 3D rồi sau đó được chuyển đổi về không gian 2D để phục vụ cho việc tìm đường đi.

Trong class MapGenerator_3D, hàm import_file() được viết lại nhằm override hàm ở lớp cha, phục vụ riêng cho việc đọc thông tin ở không gian 3D.

Một số hàm được định nghĩa lại nhằm override các hàm ở class cha để thực hiện công việc của riêng không gian 3D.

vii. Thuật toán

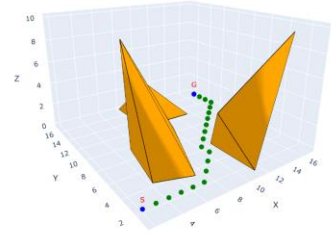
Về cơ bản, thuật toán tìm đường đi trên không gian 3D tương tự như trên không gian 2D. Nói cách khác, agent chỉ tìm đường đi ở mặt dưới của không gian 3D (mặt Oxy), không tìm đường đi lên trên.

Ta thực hiện quy trình tìm đường đi giống như trong không gian 2D với input là mặt Oxy. Sau đó, khi đã có đường đi, ta thực hiện vẽ không gian 3D và vẽ đường đi lên không gian đó với class vẽ đồ thị của không gian 3D (Displayer_3D). Class Displayer_3D sẽ dùng các thông tin bổ sung về không gian 3D (chiều cao của không gian, các đỉnh của đa giác) để vẽ và xuất ra kết quả dưới dạng file HTML.

Lưu ý: chương trình có thể dùng cả 3 thuật toán tìm đường đi ở trên để tìm đường đi cho không gian 3D.

viii. Chạy thử

Trường hợp không gian có các hình khối cơ bản và không có điểm đón:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 22,22,22 2 2,2,17,16 3 3 4 4,4,0,5,9,0,10,10,0,7,3,0,5,10,9 5 8,12,0,8,17,0,13,12,0,10,15,3 6 11,1,0,11,6,0,14,6,0,14,1,10,8,2,5 </pre>	<p>Dijkstra_test1 - Cost: 19</p> 

Trường hợp không gian có nhiều hình khối và có các điểm đón:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 20,20,20 2 1,1,19,18,11,8,6,12,15,11 3 8 4 5,9,0,8,10,0,9,7,0,7,8,5 5 8,17,0,13,12,0,8,12,0,10,13,7 6 11,1,0,11,4,0,14,6,0,14,1,0,12,3,10 7 10,10,0,12,12,0,15,10,0,12,8,0,11,9,10 8 5,14,0,5,16,0,8,16,0,8,14,0,6,15,3 9 13,15,0,15,16,0,17,14,0,15,13,0,14,14,0,14,2 10 6,3,0,6,7,0,9,7,0,9,3,0,7,4,10 11 16,5,0,16,10,0,18,10,0,18,5,0,16,5,9 </pre>	<p>AStar_test2 - Cost: 40</p> 

Trường hợp không gian có các hình khối cơ bản và không tìm được đường đi:

Dữ liệu đầu vào:	Thuật toán chạy ra kết quả đường đi:
<pre> 1 10,10,10 2 1,1,9,9 3 2 4 4,0,0,4,5,0,6,5,0,6,0,0,4,5,5 5 4,3,0,0,3,0,0,5,0,4,5,0,4,5,5 </pre>	<p>Dijkstra_test3 - Cost: 0</p> 

IV. Cách chạy chương trình

Prompt:

```
python main.py -filepath (đường dẫn) -level (nhập mức) -search (thuật toán)
```

- -filepath [-f]: Tên tệp đầu vào, đường dẫn tương đối hoặc đường dẫn tuyệt đối cần xử lý.
- -level [-l]: Chỉ định loại mức theo đồ án yêu cầu để chạy (1-5).
- -search [-s]: Tùy chọn giải thuật tìm kiếm, lựa chọn giải thuật tìm kiếm tương ứng:
 - Dijkstra: "dijkstra"
 - A* search: "astar"
 - Greedy Best First Search: "gbfs"

Note:

- Mặc định mức 3 chỉ có thuật toán A* search chạy được
- Thông số trong file phải phù hợp với các mức, chẳng hạn như có điểm đón hay không, map 3D hay 2D.

Ví dụ 1:

Prompt: `python main.py -f input.txt -l 3`

Trong ví dụ này:

- Chương trình sẽ chạy trên tệp `input.txt`.
- Bản đồ được sử dụng là mức 3.
- Mặc định sử dụng thuật toán tìm kiếm A*.

Ví dụ 2:

Prompt: `python main.py -f input.txt -l 3 -s dijkstra`

Trong ví dụ này:

- Chương trình sẽ chạy trên tệp `input.txt`.
- Bản đồ được sử dụng là mức 3.
- In ra thông báo lỗi rằng mức 3 chỉ chạy A*.

Ví dụ 3:

Prompt: `python main.py -filepath input.txt -level 5 -search gbfs`

Trong ví dụ này:

- Chương trình sẽ chạy trên tệp `input.txt`.
- Bản đồ được sử dụng là mức 5(3D).
- Sử dụng thuật toán tìm kiếm GBFS.

Demo video:

[HCMUS - AI - Robot Path Finding \(youtube.com\)](#)

NGUỒN THAM KHẢO

1. https://github.com/heraclex12/robot_path
2. [A* Search Algorithm - GeeksforGeeks](#)
3. <http://www.philliplemons.com/posts/ray-casting-algorithm>
4. [Python Itertools - GeeksforGeeks](#)
5. [Using Matplotlib for Animations - GeeksforGeeks](#)
6. [Traveling Salesman Problem \(TSP\) Implementation - GeeksforGeeks](#)