ĐẠI HỌC KINH TẾ TP. HỒ CHÍ MINH

TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ

| KHOA CÔNG NGHỆ THÔNG TIN KINH DOANH

UEH
UNIVERSITY

BÁO CÁO CUỐI KỲ

Tối Ưu Bài Toán Robot Hút Bụi Bằng Thuật Toán A*

Môn học: LẬP TRÌNH PHÂN TÍCH DỮ LIỆU

Mã lớp học phần: 24C1INF50907001

Giảng viên hướng dẫn: TS. Nguyễn An Tế

Nhóm sinh viên thực hiện:

- Nguyễn Thành Vinh - 31221025662
- Trầm Thái Tú - 31221022394
- Trần Vọng Triển - 31221021725
- Nguyễn Vân Phi Yến - 31221021785

Hồ Chí Minh, ngày 23 tháng 11 năm 2024

MỤC LỤC

DANH MỤC HÌNH ẢNH

CHƯƠNG 1. TỔNG QUAN

1.1. Giới Thiệu Về Thuật Toán A*

Thuật toán A* được mô tả lần đầu tiên trong công trình nghiên cứu của ba nhà khoa học Peter Hart, Nils Nilsson và Bertram Raphael vào năm 1968. Ban đầu, họ đề xuất một thuật toán đơn giản có tên gọi là thuật toán A. Qua quá trình nghiên cứu và phát triển, họ đã chứng minh rằng khi kết hợp với một hàm đánh giá heuristic phù hợp, thuật toán này có thể đạt được hiệu quả tối ưu. Từ đó, thuật toán được đổi tên thành A* như chúng ta biết đến ngày nay.

Thuật toán tìm kiếm A * là một thuật toán tìm kiếm đường đi được sử dụng rộng rãi, được thiết kế để xác định đường đi ngắn nhất giữa điểm bắt đầu và điểm đích trong đồ thị có trọng số. A* kết hợp ưu điểm của tìm kiếm theo chi phí thấp nhất (như Dijkstra) và tìm kiếm theo chiều sâu (depth-first search), giúp tìm ra đường đi ngắn nhất một cách hiệu quả. Không giống như một số thuật toán khác, A* sử dụng hàm đánh giá heuristic đặc biệt, cho phép xếp loại và lựa chọn các điểm trên đồ thị. Hàm heuristic trong A* dựa trên ước lượng khoảng cách ngắn nhất từ một điểm đến đích, ví dụ như khoảng cách đường chim bay, tính toán đường giao thông…

Điểm khác biệt của A* so với các thuật toán tìm kiếm lựa chọn tốt nhất là nó tính cả khoảng cách đã đi qua, giúp thuật toán đầy đủ và tối ưu, luôn tìm được đường ngắn nhất nếu tồn tại.

1.2. Mô tả thuật toán

A * xem xét cả khoảng cách từ điểm bắt đầu đến nút hiện tại g(x), cũng như ước lượng khoảng cách từ nút hiện tại đến nút đích h(x). Bằng cách kết hợp hai giá trị này, A* tính điểm tổng f(x) cho mỗi nút, với mục tiêu chọn nút có giá trị f(x) thấp nhất làm bước tiếp theo trong quy trình tìm đường dẫn. Quá trình này tiếp tục lặp đi lặp lại cho đến khi đạt được nút đích.

Giả sử n là một trạng thái nhất định (có đường đi từ trạng thái ban đầu đến n). Ta xác định hàm đánh giá như sau:

$$f(n) = g(n) + h(n)$$

- $g(n)$: Khoảng cách từ điểm bắt đầu đến điểm hiện tại n.

- $h(n)$: Khoảng cách ước lượng từ điểm hiện tại n đến điểm đích.
- $f(n)$: Tổng khoảng cách ước lượng của đường đi qua nút hiện tại n đến đích.

Hàm heuristic $h(n)$ được xem là chấp nhận được nếu với mọi nút n, điều kiện sau được thỏa mãn:

$$0 \leq h(n) \leq h^*(n)$$

Trong đó, $h^*(n)$ là chi phí thực tế thấp nhất từ nút n đến đích.

1.3. Cài đặt thuật toán A*

OPEN (FRINGE): Là tập chứa các trạng thái đã được sinh ra nhưng chưa xét đến. OPEN là một hàng đợi ưu tiên, trong đó phần tử có độ ưu tiên cao nhất (tức là trạng thái có giá trị f nhỏ nhất) được xét đầu tiên.

CLOSE: Là tập chứa các trạng thái đã được xét đến. CLOSE được lưu lại để tránh việc xử lý trùng lặp các trạng thái đã xét, giúp tiết kiệm thời gian và tài nguyên.

Khi xét một trạng thái n_i trong OPEN, ngoài việc lưu các giá trị cơ bản như:

- g: Chi phí từ trạng thái ban đầu đến trạng thái hiện tại n_i
- h: Chi phí ước lượng từ n_i đến đích.
- f: Tổng chi phí f=g +h, phản ánh thứ tự ưu tiên của trạng thái.

Thuật toán A* còn lưu thêm:

- Trạng thái cha của n_i: Lưu trạng thái dẫn đến n_i, giúp truy vết lại đường đi từ đích về trạng thái ban đầu.
- Danh sách các trạng thái tiếp theo của n_i: Danh sách lưu các trạng thái kế tiếp n_k mà n_i có thể dẫn đến, với chi phí từ trạng thái ban đầu đến n_k thông qua n_i là thấp nhất. Danh sách này có thể được tính dựa trên thông tin của các trạng thái cha đã lưu.

Thuật toán A*

function Astar(n_0, n_{goal})

1. Khởi tạo:

- Tập OPEN chứa n_0
- Khởi tạo: g(n_0) = h(n_0) = f(n_0) = 0

- Khởi tạo CLOSE là tập rỗng

2. Lặp các bước sau cho đến khi thỏa điều kiện dừng:

2.1. Kiểm tra tập OPEN:

- Nếu OPEN rỗng: kết thúc thuật toán, bài toán vô nghiệm

2.2. Chọn trạng thái tiếp theo:

- Chọn n_i từ OPEN sao cho f(n_i) là nhỏ nhất

- Loại n_i ra khỏi OPEN

- Thêm n_i vào CLOSE

2.3. Kiểm tra điều kiện đích:

- Nếu n_i chính là n_{goal}: kết thúc thuật toán, trả về n_i là lời giải

2.4. Sinh các trạng thái kế tiếp:

- Nếu n_i không phải đích:

+ Tạo danh sách tất cả trạng thái kế tiếp n_k của n_i

+ Với mỗi trạng thái kế tiếp n_k:

- Tính g(n_k) = g(n_i) + cost(n_i , n_k)
- Tính h(n_k) theo hàm ước lượng
- Tính f(n_k) = g(n_k) + h(n_k)
- Đặt Cha(n_k) = n_i
- Nếu n_k chưa có trong cả OPEN và CLOSE: thêm n_k vào OPEN

1.4. Một Vài Ứng Dụng Của Thuật Toán A*

Thuật toán A* được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau của công nghệ và đời sống. Trong ngành game, A* đóng vai trò quan trọng trong việc tìm đường đi cho nhân vật, được sử dụng trong các tựa game nổi tiếng như Warcraft, Starcraft để điều khiển nhân vật tránh chướng ngại vật, hay trong Pokemon Go để tối ưu đường đi cho nhân vật ảo. Không chỉ dừng lại ở game, thuật toán này còn được ứng dụng mạnh mẽ trong lĩnh vực robotics, đặc biệt là trong việc điều khiển robot tự hành và xe tự lái, giúp chúng có thể tìm đường đi tối ưu và tránh va chạm.

Trong GPS và bản đồ số, A* là nền tảng cho các ứng dụng như Google Maps, giúp người dùng tìm được đường đi ngắn nhất giữa hai địa điểm. Các ứng dụng gọi xe như Uber/Grab cũng sử dụng thuật toán này để tối ưu hóa lộ trình cho tài xế. Đặc biệt trong ngành logistics, A* góp phần quan trọng trong việc lập kế hoạch giao hàng hiệu quả.

Trong lĩnh vực trí tuệ nhân tạo và machine learning, A* được sử dụng để giải quyết các bài toán như 8 puzzle hay tìm lời giải cho khối Rubik. Thuật toán cũng được áp dụng trong quy hoạch đường bay cho drone. Về mặt công nghiệp, A* đóng vai trò quan trọng trong việc tối ưu hóa đường đi trong mạch điện tử, lập kế hoạch sản xuất tự động và điều phối giao thông thông minh.

Trong đời sống hàng ngày, A* có nhiều ứng dụng thiết thực như tìm đường đi trong trung tâm thương mại, quy hoạch mạng lưới cáp quang hay thiết kế hệ thống thoát hiểm trong tòa nhà. Sự thành công của thuật toán A* đến từ những ưu điểm vượt trội: luôn tìm ra đường đi tối ưu (nếu tồn tại), hiệu quả hơn các thuật toán tìm kiếm mù như BFS, DFS, và đặc biệt là tính linh hoạt trong việc điều chỉnh hàm ước lượng h(n) để phù hợp với từng bài toán cụ thể.

CHƯƠNG 2. ỨNG DỤNG THUẬT TOÁN A* VÀO BÀI TOÁN ROBOT HÚT BỤI

2.1. Định Nghĩa Và Phân Tích Bài Toán Robot Hút Bụi

2.1.1. Cấu Trúc Ma Trận Và Các Trạng Thái Của Ô

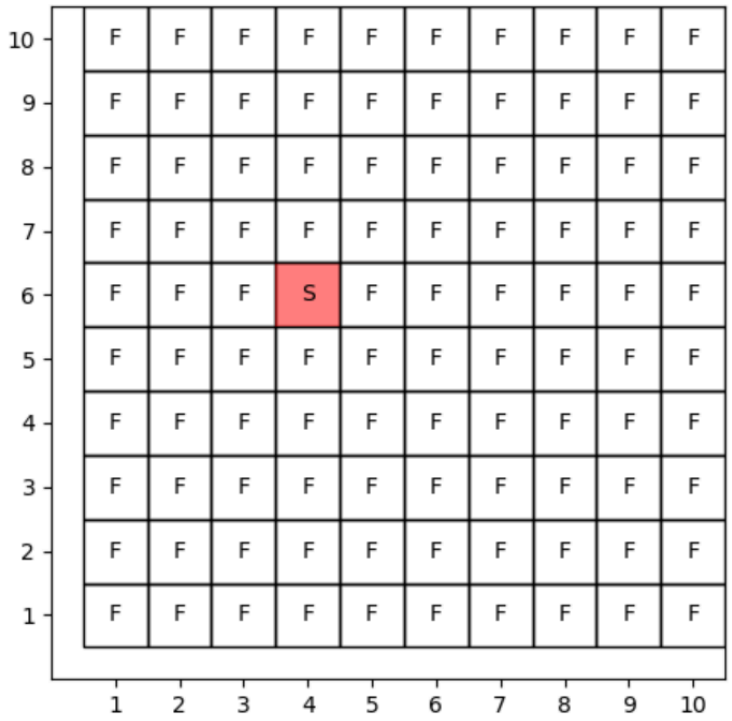
Cấu trúc ma trận: Vùng diện tích hình chữ nhật được biểu diễn dưới dạng ma trận $A_{m\times n}$Trong đó:

- m: Số hàng
- n: Số cột
- Góc tọa độ $(1, 1)$ nằm ở góc dưới bên trái của ma trận

Các trạng thái của ô: Mỗi ô trong ma trận có thể mang một trong ba trạng thái:

- *Free (F)*: Ô trống, không có bụi
- *Dirty (D):* Ô bẩn, có bụi cần được làm sạch
- *Clean (C):* Ô đã được làm sạch

Ví dụ: Một ma trận $A_{10\times 10}$ (10 hàng, 10 cột) có thể được biểu diễn như sau:



Hình 2.1.1. Mô phỏng ma trận $A_{10\times 10}$ (10 hàng, 10 cột) với vị trí bắt đầu của robot là (4,6)

2.1.2. Các Hành Động Của Robot

Mỗi lần, robot hút bụi sẽ chỉ được thực hiện một trong hai loại hành động sau:

a) Di chuyển:

Robot có thể di chuyển từ vị trí bắt đầu đến bất kỳ ô nào trong vùng 3x3 bao quanh nó. Điều này nghĩa là robot có thể di chuyển đến 8 ô liền kề, bao gồm các hướng:

- Trên (North) - Dưới (South) - Trái (West) - Phải (East)
- Chéo (North-East, North-West, South-East, South-West)

Ví dụ: Trong ma trận *Hình 2.1*, robot đang ở vị trí $(4, 6)$, các ô mà robot có thể di chuyển đến (nếu hợp lệ) bao gồm:

$$(3, 5);\ (4, 5);\ (5, 5);\ (3, 6);\ (5, 6);\ (3, 7);\ (4, 7);\ (5, 7)$$

b) Hút bụi (Suck):

Sau khi di chuyển đến vị trí các ô dơ, robot thực hiện thao tác hút bụi để làm sạch ô mà nó đang đứng. Nếu ô đó đang ở trạng thái *Dirty (D)*, sau hành động này, trạng thái của ô sẽ chuyển thành *Clean (C)*.

2.1.3. Chi Phí Của Các Hành Động Và Yếu Tố Ảnh Hưởng Đến Chi Phí

Trong bài toán robot hút bụi, chi phí được chia thành hai loại chính: *Chi phí di chuyển* và *Chi phí hút bụi*. Các chi phí này được tính toán và ảnh hưởng bởi các hành động của robot như sau:

- *Chi phí di chuyển:* là chi phí tiêu tốn khi robot di chuyển (Move) từ ô hiện tại đến một ô khác trong phạm vi 8 ô liền kề.
- *Chi phí hút bụi:* là chi phí tiêu tốn khi robot thực hiện hành động hút bụi (Suck) để làm sạch ô Dirty mà nó đang đứng.

Quy tắc cập nhật chi phí: Các chi phí trong bài toán được cập nhật theo các quy tắc sau:

a) Khi robot thực hiện hành động di chuyển:

- Chi phí di chuyển **tăng thêm 1 đơn vị**.
- Chi phí hút bụi tại tất cả các ô Dirty chưa được làm sạch **tăng thêm 1 đơn vị**.

b) Khi robot thực hiện hành động hút bụi:

- Chi phí hút bụi tại ô hiện tại được tính và cộng vào tổng chi phí.
- Chi phí hút bụi tại các ô Dirty khác **không thay đổi** sau hành động này.

Tổng chi phí của toàn bộ quá trình hoạt động của robot được tính bằng:

$$Tổng\ chi\ phí\ =\ Tổng\ chi\ phí\ di\ chuyển\ +\ Tổng\ chi\ phí\ hút\ bụi$$

2.2. Cấu Trúc Chương Trình

2.2.1. Lớp 'Position'

Lớp 'Position' được thiết kế để biểu diễn một điểm trong không gian 2D với hai tọa độ x và y. *Lớp 'Position'* giúp quản lý và so sánh các vị trí tọa độ trong không gian, đồng thời giúp tích hợp tốt với các cấu trúc dữ liệu như *set* hoặc *dict*.

```
class Position:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __hash__(self):
        return hash((self.x, self.y))
```

2.2.2. Lớp 'Cell'

Lớp 'Cell' trong thuật toán A* đóng vai trò quan trọng trong việc đại diện một ô trong quá trình tìm kiếm đường đi. Mỗi đối tượng *Cell* chứa thông tin về vị trí của ô trong không gian, chi phí đã đi qua *(g)*, chi phí ước lượng còn lại để đến đích *(h)*, thông tin về trạng thái và *"parent" (ô cha)* để phục hồi đường đi.

Lớp 'Cell' bao gồm các thành phần chính sau:

- *Thông tin vị trí (position):* Mỗi đối tượng *Cell* lưu trữ thông tin về tọa độ của ô trong không gian 2D.
- *Chi phí từ điểm bắt đầu đến vị trí hiện tại (g) và Chi phí ước lượng đến điểm đích (h - heuristic)*
- *Tính toán chi phí tổng (f):* Phương thức tính toán chi phí tổng dựa trên công thức sau:

$$f = g + h$$

```
class Cell:
    def __init__(self, position, parent, status, g, h):
        self.position = position
        self.parent = parent
        self.status = status
        self.g = g  # Chi phí từ điểm bắt đầu đến vị trí hiện tại
        self.h = h  # Chi phí ước lượng đến điểm đích (heuristic)
```

```
    def f(self):
        return self.g + self.h

    # Thêm phương thức __lt__ để hỗ trợ so sánh giữa các Cell
    def __lt__(self, other):
        return self.f() < other.f()  # So sánh theo tổng chi phí f
```

2.2.3. Lớp 'RobotVacuumCleaner'

a) Hàm 'cluster_dirty_cells'

Hàm 'cluster_dirty_cells' thực hiện phân nhóm các ô bẩn thành các cụm dựa trên khoảng cách Euclid và thuật toán DBSCAN (Density-Based Spatial Clustering of Appllications with Noise) để phát hiện các cụm của các ô bẩn, với khả năng phát hiện các cụm có mật độ cao.

Hàm 'cluster_dirty_cells" có ba tham số chính:

```
def cluster_dirty_cells(self,epsilon=1.5, min_samples=1, grid_new = None):
```

- *epsilon:* Khoảng cách tối đa giữa các điểm trong cùng một cụm. Nếu khoảng cách giữa hai ô bẩn nhỏ hơn hoặc bằng giá trị *epsilon* thì sẽ được gom thành một cụm.
- *min_samples:* Số lượng tối thiểu các ô bẩn để gom thành một cụm. Nếu một nhóm điểm có số lượng ô bẩn ít hơn *min_samples*, thì sẽ không được coi là một cụm hợp lệ.
- *grid_new:* Lưới chứa thông tin về trạng thái các ô (Ví dụ: 'Dirty' hoặc 'Clean'), được duyệt qua để xác định các ô bẩn cần được gom cụm.

b) Hàm 'heuristic'

Hàm 'heuristic' áp dụng khoảng cách Euclid để tính toán chi phí ước lượng giữa hai điểm. Khoảng cách Euclid giữa hai điểm (x_1, y_1) và (x_2, y_2) được tính bằng công thức sau:

$$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Trong đó:

- x_1, y_1 là tọa độ của điểm đầu tiên (vị trí ban đầu)
- x_2, y_2 là tọa độ của điểm thứ hai (vị trí đích)

```
def heuristic(self, pos1, pos2):
        return math.sqrt((pos1.x - pos2.x) ** 2 + (pos1.y - pos2.y) ** 2)
```

c) *Hàm 'a_star_search'*

Hàm 'a_star_search' triển khai thuật toán A* nhằm xác định đường đi tối ưu từ một điểm xuất phát (start_position) đến một điểm đích (goal_position). A* sử dụng tham số heuristic để ước lượng chi phí di chuyển còn lại đến đích và kết hợp với chi phí đã đi qua (g) để tìm đường đi ngắn nhất.

Khởi tạo hàm *'a_star_search'* với các thông số đầu vào bao gồm: *start_position* và *goal_position*. Tiếp theo, hàm sẽ tiến hành khởi tạo các thông số:

- *open_set:* Hàng đợi ưu tiên lưu trữ các ô đang chờ được dọn dẹp
- *came_from:* Dictionary lưu trữ thông tin về parent của mỗi ô
- *start_cell:* Tạo một đối tượng Cell đại diện cho ô bắt đầu

```
    def a_star_search(self, start_position, goal_position):
        open_set = []
        came_from = {}
        start_cell = Cell(start_position, None, 'start', 0, 0)
```

Sau khi khởi tạo, hàm sẽ thêm ô bắt đầu vào hàng đợi ưu tiên *open_set.*

```
        heapq.heappush(open_set, start_cell)
```

Kế đến, hàm sẽ bước vào vòng lặp chính để xác định tìm kiếm đường đi.

2.3. Quy Trình Hoạt Động Của Thuật Toán

Bước 1:

- Phân các ô bẩn vào các cụm bằng thuật toán phân cụm DBSCAN dựa trên khoảng cách Euclid và epsilon xác định dựa trên Elbow tự động.

Bước 2:

- Tính phương trình alpha * số lượng ô bẩn - beta * khoảng cách từ vị trí hiện tại của robot đến từng cụm.
- Sau đó sắp xếp giảm dần các cụm theo giá trị vừa tính được.
- Chọn cụm đầu tiên trong danh sách sau khi sắp xếp là cụm ưu tiên được làm sạch trước.

Bước 3: Dùng thuật toán A* để di chuyển robot đến vị trí ô bẩn gần robot nhất trong cụm vừa chọn.

Bước 4: Làm sạch lần lượt tất cả các ô bẩn trong cụm bằng thuật toán A*. Sau đó xóa cụm các ô bẩn này khỏi danh sách ưu tiên các cụm ô bẩn.

Quay lại bước 2 cho đến khi không còn cụm ô bẩn nào.

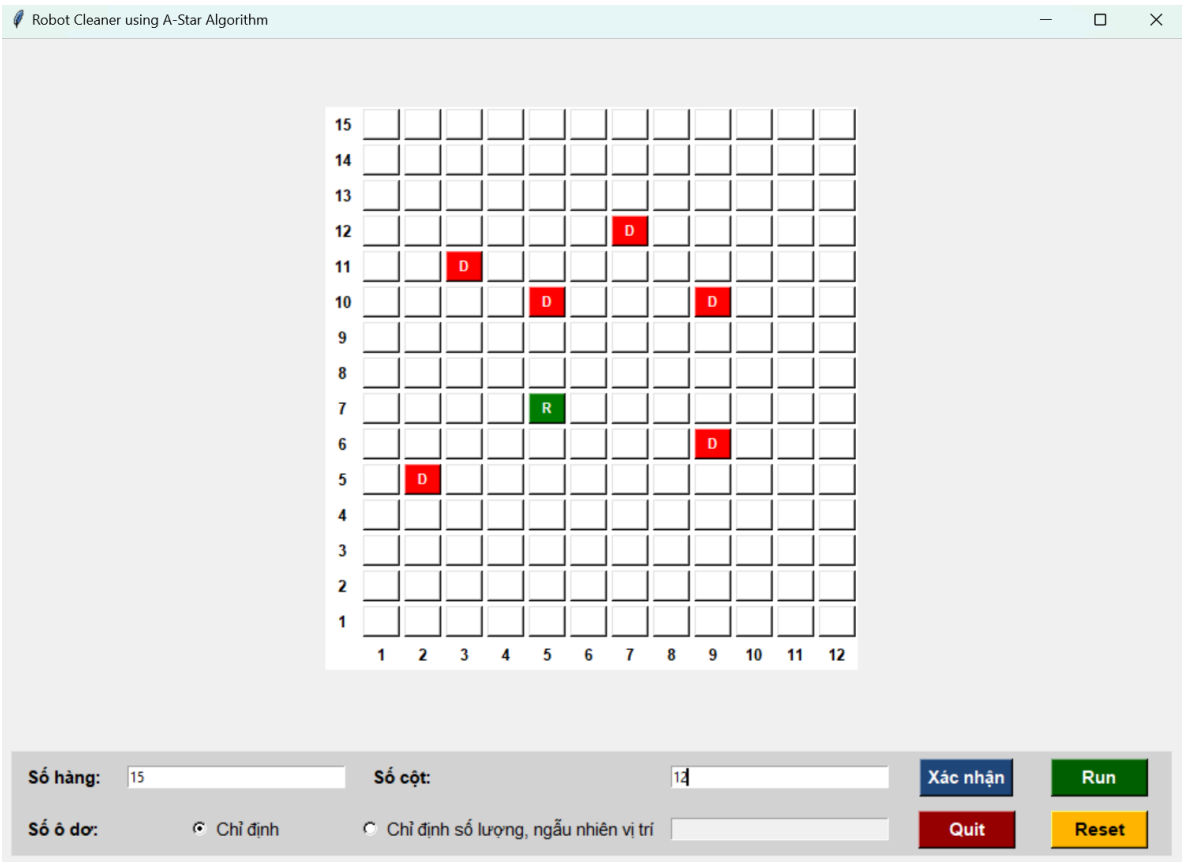
Giải thích thuật toán A* cho quá trình robot làm sạch ô bẩn trong bài:

- Thuật toán A* giúp robot tìm đường đi ngắn nhất từ vị trí hiện tại đến ô bẩn được chọn (ô bẩn được chọn theo tiêu chí gần với robot nhất trong cụm đã tìm).
- **Khởi tạo**:
    - Thuật toán bắt đầu bằng cách tạo một ô bắt đầu (start_cell) và đưa nó vào open_set (hàng đợi ưu tiên). Ô này có chi phí g = 0 và heuristic h được tính từ điểm bắt đầu đến đích. Hàm heuristic được ước lượng dựa trên khoảng cách Euclid.
- **Vòng lặp chính**:
    - **Lấy ô có giá trị f thấp nhất**:
        - Thuật toán tiếp tục khám phá ô có giá trị f = g + h thấp nhất trong open_set (dùng hàm heapq.heappop() để lấy phần tử có giá trị nhỏ nhất). Ô này sẽ được gọi là current_cell.
    - **Kiểm tra nếu đã đến đích**:
        - Nếu ô hiện tại (current_cell) là ô đích (goal_position), thuật toán sẽ bắt đầu phục hồi đường đi từ đích về bắt đầu bằng cách lần lượt truy ngược từ came_from.
        - Đường đi sẽ được lưu trong danh sách path, và kết quả cuối cùng sẽ là đường đi từ bắt đầu đến đích.
- **Khám phá các ô lân cận**:
    - Nếu không phải là ô đích, thuật toán sẽ kiểm tra các ô lân cận của ô hiện tại (gọi hàm get_neighbors() để lấy các vị trí lân cận).
    - Với mỗi ô lân cận (neighbor_pos), thuật toán sẽ:
        - **Tạo đối tượng ô lân cận (neighbor)**: Dựa trên vị trí lân cận, thuật toán tạo ra một đối tượng Cell mới. Ô này có giá trị g=∞ (không có giá trị g ban đầu) và hàm heuristic hhh tính từ vị trí lân cận đến đích.
        - **Tránh ô đã được dọn sạch**: Nếu ô đó là ô "clean", thuật toán sẽ bỏ qua ô này để không khám phá lại các ô đã sạch.

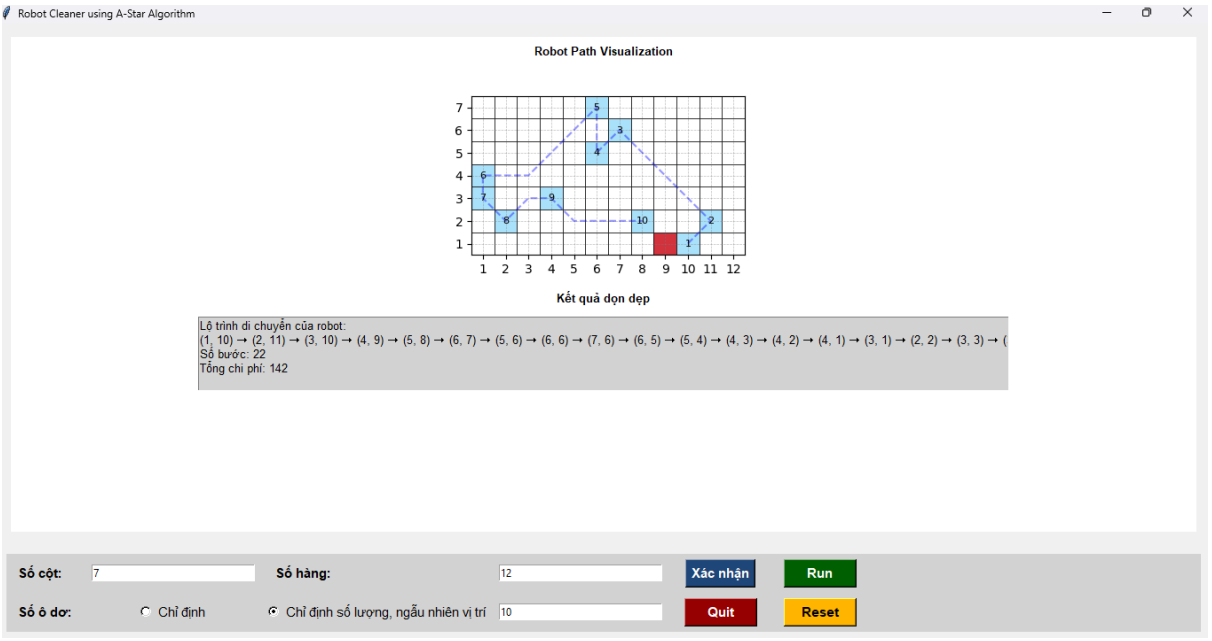
        - **Tính toán chi phí di chuyển tạm thời** (tentative_g_score): Đây là chi phí từ điểm bắt đầu đến ô lân cận. Trong trường hợp này, chi phí di chuyển mỗi bước là 1 (đơn vị di chuyển).
    - **Cập nhật nếu tìm thấy con đường ngắn hơn**:
        - Nếu chi phí di chuyển tạm thời (tentative g) từ ô hiện tại đến ô lân cận là nhỏ hơn giá trị g đã lưu trữ trong ô lân cận (ban đầu là vô cùng), thuật toán sẽ:
            - Cập nhật thông tin trong came_from để lưu lại ô hiện tại là "tiền thân" của ô lân cận.
            - Cập nhật giá trị g của ô lân cận.
            - Thêm ô lân cận vào open_set để tiếp tục khám phá sau.
- **Kết thúc**:
    - Nếu tất cả các ô lân cận đã được khám phá và không tìm được đường đi đến đích (trong trường hợp không có đường đi khả thi), thuật toán sẽ trả về None.

2.4. Kết Quả Thực Nghiệm

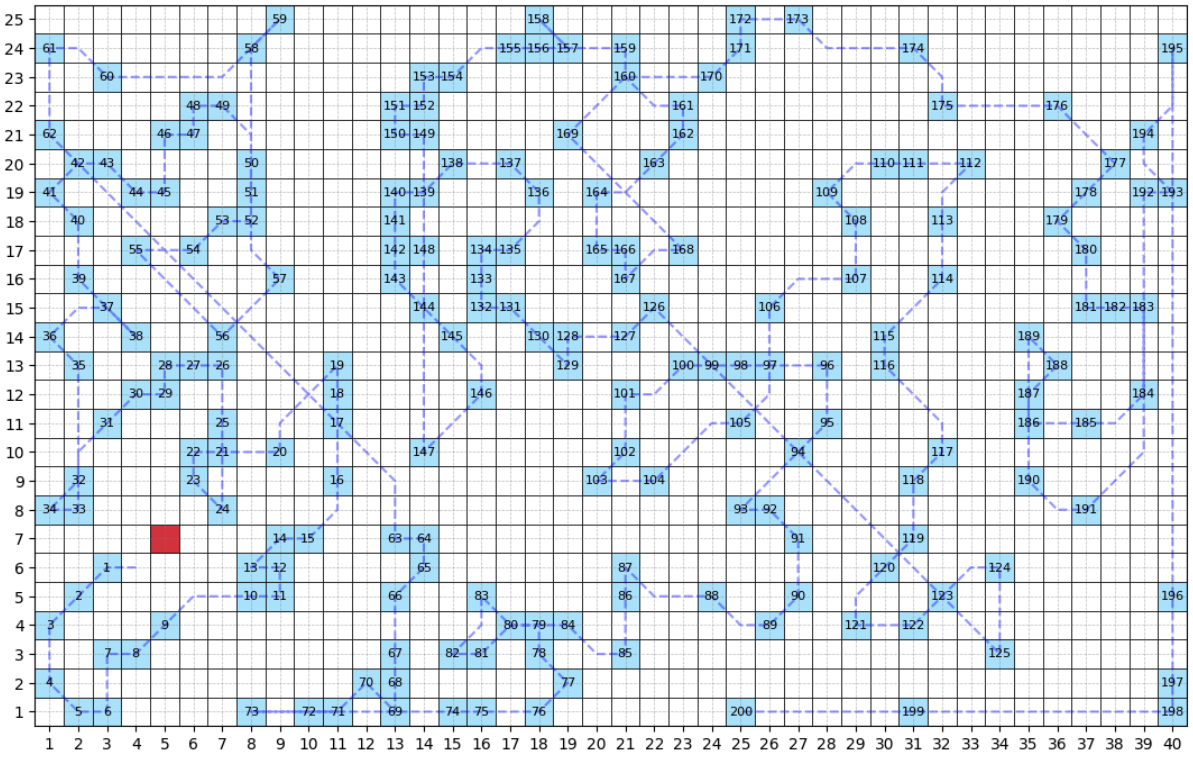
Chương trình đã đạt được mục tiêu trong việc tìm kiếm đường đi tối ưu với chi phí tốt để dọn sạch tất cả các ô bẩn trên bản đồ. Thuật toán A* kết hợp với phân cụm đã được áp dụng hiệu quả, giúp robot xác định các cụm ô bẩn và di chuyển một cách hợp lý. Chương trình cho phép người dùng nhập các thông số khởi tạo, đồng thời cung cấp giao diện hiển thị trực quan, giúp người dùng dễ dàng theo dõi quá trình làm sạch. Các kết quả thử nghiệm cho thấy hiệu quả tốt trong việc tối ưu hóa chi phí di chuyển và thời gian dọn dẹp, đáp ứng đầy đủ các yêu cầu đề ra.



Hình 2.4.1 Minh họa giao diện chương trình cho phép người dùng chỉ định vị trí xuất phát và vị trí ô bẩn.



Hình 2.4.2 Minh họa giao diện kết quả chương trình với ma trận 7x12, số ô dơ là 10, tổng chi phí là 142



Hình 2.4.3 Minh họa trực quan kết quả chương trình với ma trận 25x40, số ô dơ là 200 và tổng chi phí là 36574.

CHƯƠNG 3. TỔNG KẾT ĐỀ TÀI

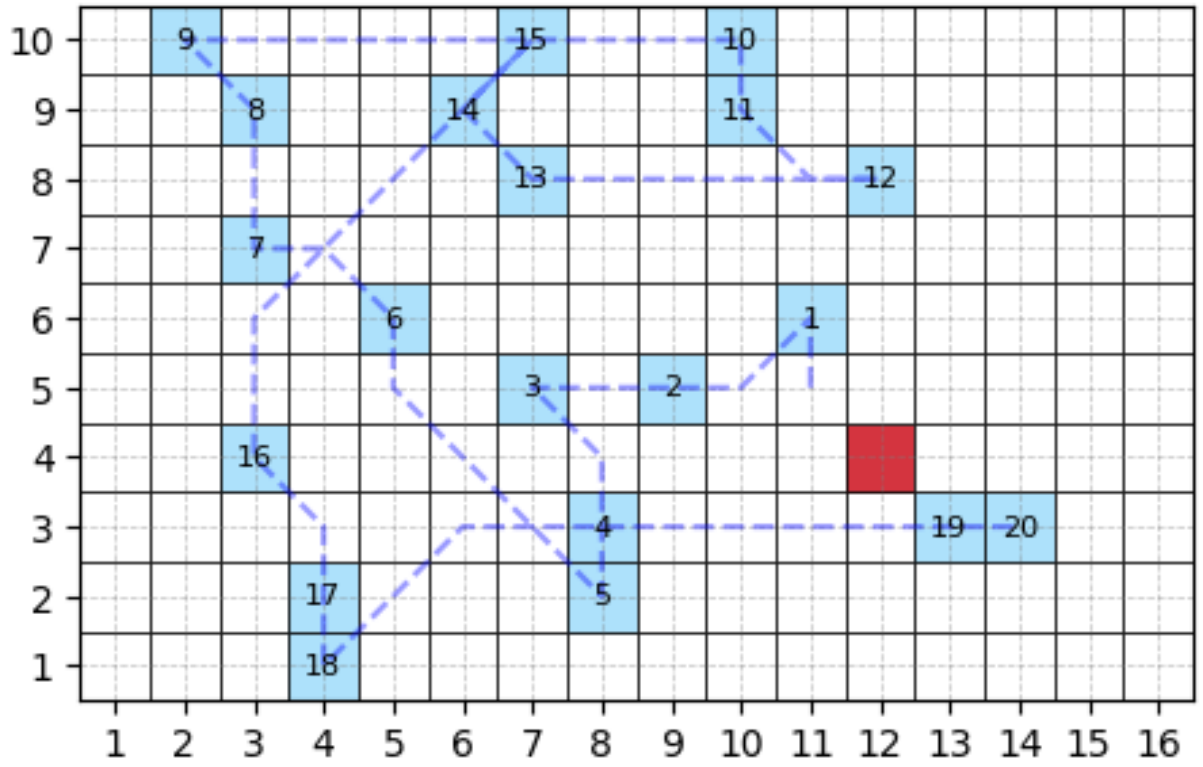
3.1. Những Hạn Chế

Trong chương trình này, thuật toán A* tập trung vào việc tìm đường ngắn nhất từ vị trí hiện tại đến một ô bẩn đích mà không xem xét toàn bộ chi phí tổng thể của việc dọn sạch tất cả các ô bẩn. Vì vậy, ý tưởng gom cụm các ô bẩn trước khi thực hiện tìm đường đi dựa trên khoảng cách Euclid và thuật toán DBSCAN được ra đời với mong muốn tối ưu hóa chiến lược dọn dẹp sao cho robot có thể dọn được nhiều ô bẩn trong giai đoạn đầu của quá trình di chuyển, giúp giảm thiểu chi phí dọn dẹp trong suốt hành trình.

Dù bước phân cụm này giúp xác định các ô bẩn có sự phân bố gần nhau, thuận tiện cho việc dọn nhiều ô bẩn liên tục với chi phí di chuyển ít, tuy nhiên, việc phân cụm còn một số yếu điểm sau:

- *Khoảng cách eps và phương pháp Elbow:* Việc tính toán giá trị eps dựa trên khoảng cáchh Euclid và phương pháp Elbow có thể không mang lại kết quả tốt trong một số trường hợp. Phương pháp Elbow không phải lúc nào cũng tìm ra một điểm gấp khúc rõ ràng. Nếu các ô bẩn không phân bố đồng đều, hoặc nếu có các vùng có mật độ ô bẩn rất cao hoặc rất thấp, việc xác định điểm gấp khúc có thể không đảm bảo hiệu quả.
- *Phân cụm lại sau mỗi lần dọn:* Sau khi robot dọn một cụm, chương trình phải phân lại các ô bẩn còn lại thành các cụm mới. Quá trình phân cụm lại này có thể làm tăng chi phí vô ích nếu các cụm ô bẩn không có biến động.
- *Sắp xếp thứ tự các cụm để dọn:* Tiêu chí chọn cụm để dọn trong chương trình là một hàm ước lượng giữa kích thước của cụm và khoảng cách từ robot đến cụm. Việc chọn cụm dựa trên hàm này không đảm bảo độ tối ưu và chưa cân nhắc đến các yếu tố khác như hình dạng cụm, chi phí di chuyển giữa các cụm và sự phân bố ô bẩn trong cụm.
- *Dọn các ô trong cụm:* Trong quá trình di chuyển và dọn dẹp các ô trong cụm, robot sẽ chọn các ô bẩn theo khoảng cách, đồng thời, ô bẩn được dọn cuối cùng trong cụm sẽ là điểm xuất phát để đi tới cụm tiếp theo, cách chọn này không đảm bảo tối ưu hóa chiến lược dọn dẹp. Đồng thời, thuật toán quy định robot cần dọn ô bẩn theo từng cụm, vì vậy, trong lộ trình di

chuyển, robot sẽ bỏ qua các ô bẩn tiềm năng thuộc cụm khác, điều này sẽ gây lãng phí chi phí.



Hình 3.1.1 Minh họa một trường hợp robot bỏ qua ô tiềm năng - ô 15 (10;7) trong quá trình di chuyển từ vị trí ô 9 (10;2) đến ô 10 (10;10) (cùng cụm).

3.2. Hướng Phát Triển

Để cải thiện chương trình và giảm thiểu các hạn chế trên, chương trình có thể được cải thiện theo các phương hướng sau:

- *Tối ưu hóa việc chia cụm:* Sử dụng một hàm đánh giá hiệu quả hơn cho các cụm, kết hợp các yếu tố với trọng số phù hợp hơn. Có thể áp dụng các thuật toán tối ưu hóa để xác định thứ tự làm sạch các cụm sao cho chi phí tổng thể thấp nhất.
- *Tái phân cụm tốt hơn:* Thay vì phân cụm lại mỗi lần sau khi dọn một cụm, có thể tích hợp thêm một cơ chế tái phân cụm ít tốn chi phí hơn, như sử dụng chiến lược phân cụm theo từng giai đoạn.

- *Tối ưu quá trình dọn cụm:* Thay vì chỉ tìm ô bẩn gần nhất, có thể áp dụng các chiến lược dọn dẹp hiệu quả hơn, ví dụ như giải bài toán TSP (Traveling Salesman Problem) để xác định thứ tự dọn các ô trong mỗi cụm sao cho chi phí dọn dẹp tổng thể là thấp nhất.
- *Dọn dẹp các ô tiềm năng:* trong quá trình di chuyển đến vị trí một ô bẩn đích, robot cần thêm cơ chế để chủ động dọn dẹp các ô tiềm năng trên đường đi để lãng phí chi phí quay lại ô đó vào lần sau.
- *Thêm các chướng ngại vật trên đường đi:* giúp cho bài toán trở nên thực tế hơn và tạo ra cơ hội để cải thiện thuật toán.

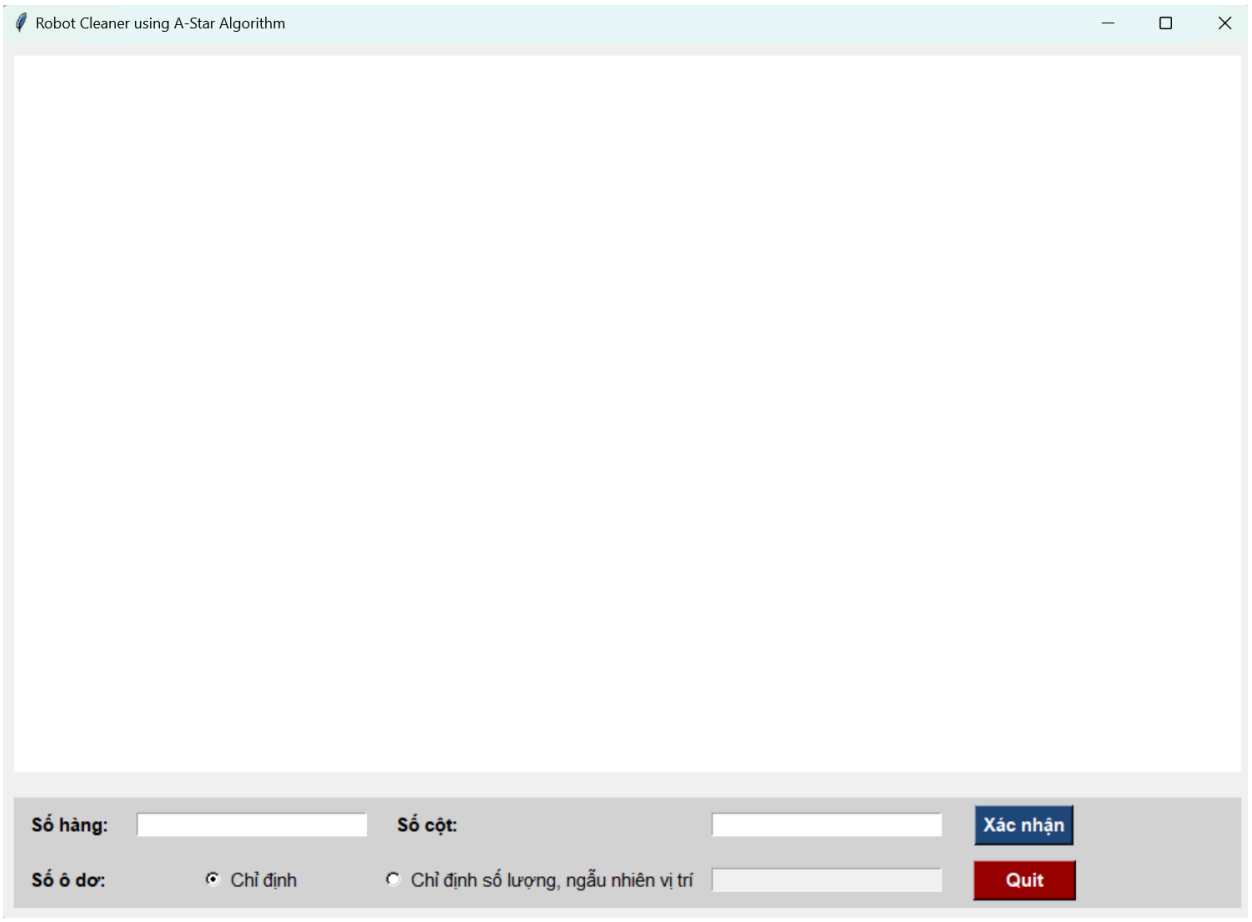
Với các cải tiến trên, thuật toán sẽ giảm được chi phí tổng thể và đạt được hiệu quả cao hơn với chi phí ít hơn.

PHỤ LỤC

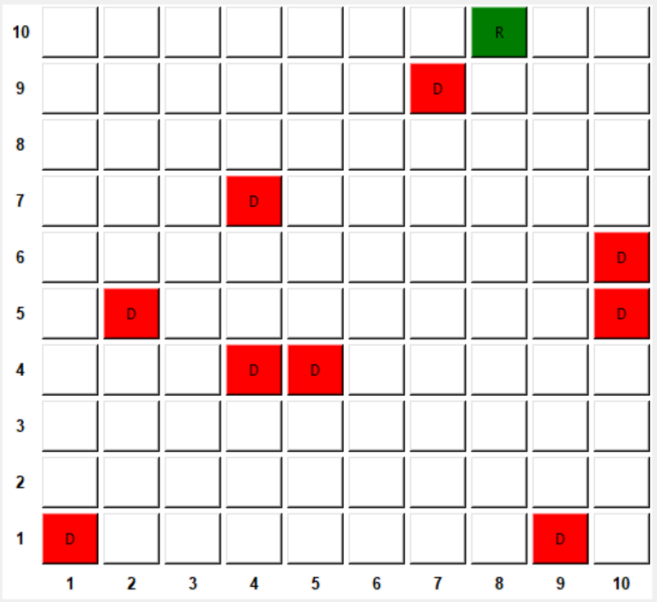
1. Phụ lục 1: Giao Diện Chương Trình trên hệ điều hành Window

Giao diện người dùng của thuật toán được xây dựng trên thư viện GUI Tkinter của Python. Các thành phần chính trong giao diện bao gồm:

- *Khung nhập liệu (Input Frame):* Giao diện cho phép người dùng nhập các giá trị kích thước của ma trận. Người dùng nhập số hàng và số cột vào các ô nhập liệu, và nhấn nút "Xác nhận" để xác nhận các giá trị nhập vào.
    - Số hàng
    - Số cột
    - Số ô dơ:
        - Chỉ định: Người dùng click nút này thì sẽ tự chỉ định vị trí và số lượng của các ô dơ và vị trí bắt đầu của robot.
        - Chỉ định số lượng, ngẫu nhiên vị trí: Khi click nút này, người dùng cần nhập số lượng các ô dơ mong muốn, chương trình sẽ tự động random vị trí các ô dơ theo số lượng đã chỉ định.
- *Khung hiển thị ma trận (Matrix Display Frame):* Sau khi người dùng xác nhận, một ma trận với các ô được tạo ra. Người dùng sẽ nhấp vào mỗi ô để chọn trạng thái cho mỗi ô trong ma trận:
    - Click - Vị trí bắt đầu của Robot (R - Màu xanh)
    - Double Click - Ô dơ (D - Màu đỏ)
    - Triple Click - Sạch (Màu trắng)
- *Khu vực hiển thị kết quả (Result Frame):* Sau khi thuật toán A* hoàn thành việc tìm đường đi và dọn dẹp, giao diện sẽ hiển thị kết quả bao gồm bản đồ đường đi của robot (Robot Path Visualization) và các thông số dọn dẹp (Tổng chi phí & Lộ trình di chuyển). Bản đồ sẽ được hiển thị dưới dạng hình ảnh, sử dụng thư viện matplotlib.
- *Các nút điều khiển:* Giao diện cung cấp các nút điều khiển như "Xác nhận", "Apply" để áp dụng các thay đổi. Bên cạnh đó, để làm mới giao diện, xóa toàn bộ thông tin nhập vào và bắt đầu lại từ đầu, người dùng có thể nhấn nút "Reset".


Robot Cleaner using A-Star Algorithm
Số hàng:
Số cột:
Xác nhận
Số ô dơ:
Chỉ định
Chỉ định số lượng, ngẫu nhiên vị trí
Quit

Robot Cleaner using A-Star Algorithm

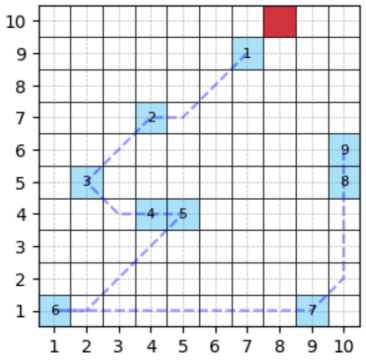


Số hàng: 10 **Số cột:** 10 **Xác nhận** **Run**

Số ô dơ: ○ Chỉ định ◉ Chỉ định số lượng, ngẫu nhiên vị trí 9 **Quit** **Reset**

Robot Cleaner using A-Star Algorithm

Robot Path Visualization



Kết quả dọn dẹp

Lộ trình di chuyển của robot:
(9, 7) → (8, 6) → (7, 5) → (7, 4) → (6, 3) → (5, 2) → (4, 3) → (4, 4) → (4, 5) → (3, 4) → (2, 3) → (1, 2) → (1, 1) → (1, 2) → (1, 3) → (1, 4) → (1, 5) → (1, 6
Số bước: 26
Tổng chi phí: 148

Số hàng: 10 **Số cột:** 10 **Xác nhận** **Run**

Số ô dơ: ○ Chỉ định ◉ Chỉ định số lượng, ngẫu nhiên vị trí 9 **Quit** **Reset**

2. Phụ lục 2: Phân công công việc

Nhiệm vụ	Thành viên	Mức độ hoàn thành
- Xây dựng file: thuật toán gom cụm và xử lý - Phụ trách nội dung chương 2	Nguyễn Thành Vinh (Nhóm trưởng)	100%
- Phụ trách nội dung chương 1	Trầm Thái Tú	100%
- Phụ trách nội dung chương 2, 3	Trần Vọng Triển	100%
- Xây dựng file: biểu diễn và giao diện - Phụ trách nội dung chương 2 - Làm Slide	Nguyễn Vân Phi Yến	100%

TÀI LIỆU THAM KHẢO

Belwariar, R. (2018, September 7). A* Search Algorithm - GeeksforGeeks. GeeksforGeeks. https://www.geeksforgeeks.org/a-search-algorithm

VietJack. (2024, October). Tổng hợp bài giảng môn Nhập môn trí tuệ nhân tạo_Thầy Nguyễn Nhật Quang| Bài giảng môn Nhập môn trí tuệ nhân tạo| Trường Đại học Bách Khoa Hà Nội | PDF. Docx.com.vn.

https://docx.com.vn/tai-lieu/tong-hop-bai-giang-mon-nhap-mon-tri-tue-nhan-tao-thay-nguyen-nhat-quan-125757

Wikipedia Contributors. (2019, March 10). A* search algorithm. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/A