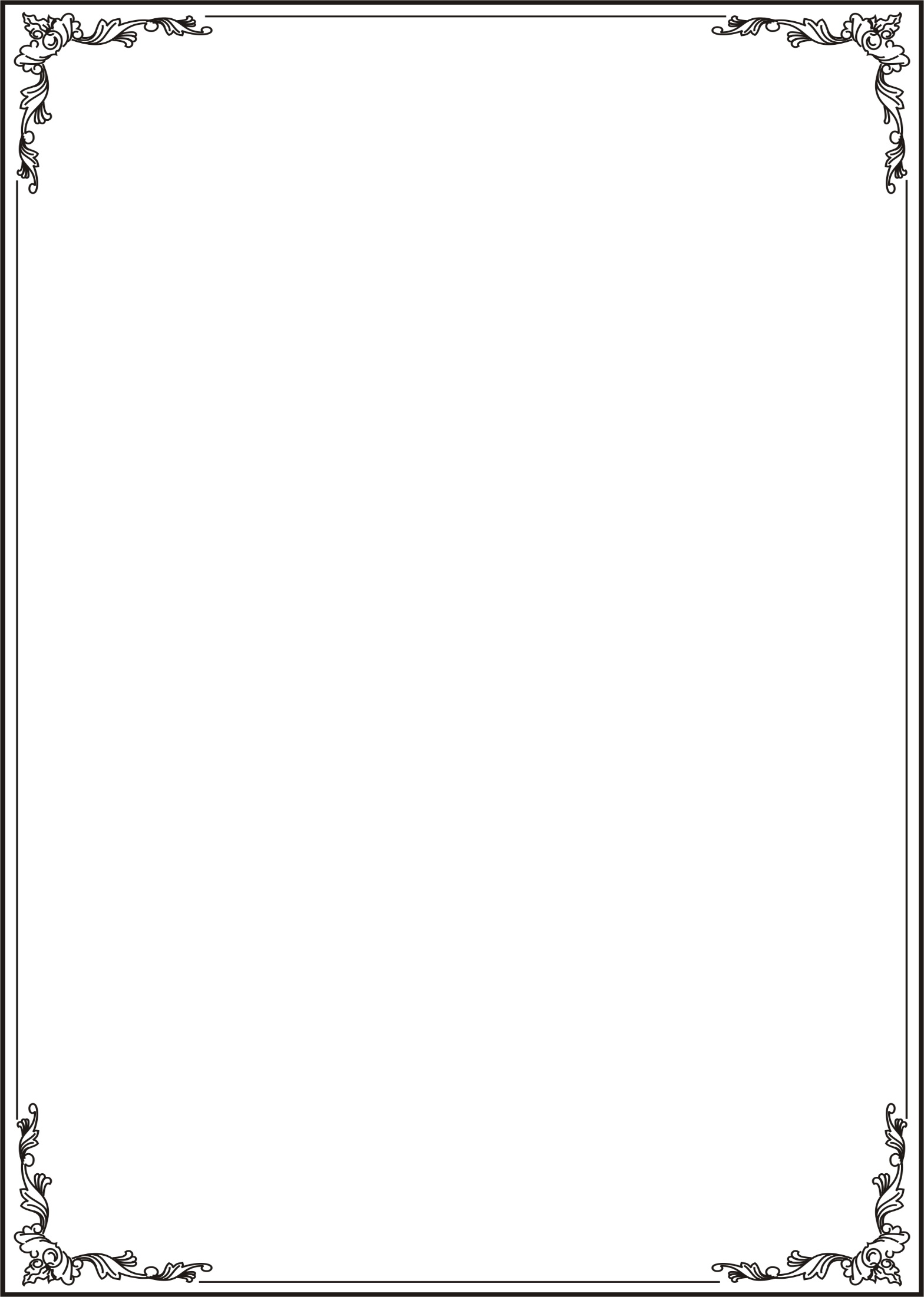
****

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO CUỐI KỲ**

**MÔN HỌC: TRÍ TUỆ NHÂN TẠO**

**BÁO CÁO**

**Mã lớp học phần: ARIN330585\_05CLC**

**Học kỳ 1 – Năm học 2025 – 2026**

**Giảng viên hướng dẫn: ThS. Phan Thị Huyền Trang**

**Danh sách sinh viên thực hiện:**

|  |  |
| --- | --- |
| **MSSV** | **Họ tên** |
| 23110101 | Đặng Gia Huy |

***Thành phố Hồ Chí Minh, tháng 10 năm 2025***

*Nhận xét của giảng viên*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*.................................................................................................................................*

*TP. Hồ Chí Minh, ngày … tháng… năm 2025*

*Giảng viên ký tên*

MỤC LỤC

PHẦN MỞ ĐẦU 1

**1. Lý do chọn đề tài** 1

**2. Mục đích nghiên cứu** 1

**3. Đối tượng và phạm vi nghiên cứu** 2

**4. Phương pháp nghiên cứu** 2

PHẦN NỘI DUNG 4

**CHƯƠNG I: CƠ SỞ LÝ THUYẾT VỀ CÁC THUẬT TOÁN TÌM KIẾM** 4

**1. Khái quát về Bài toán Tìm kiếm trong Trí tuệ Nhân tạo** 4

**1.1. Khái niệm cơ bản** 4

**1.2. Phân loại Thuật toán Tìm kiếm** 4

**2. Nhóm Thuật toán Tìm kiếm Mù (Uninformed Search Algorithms)** 5

**2.1. Tìm kiếm theo Chiều rộng (Breadth-First Search - BFS)** 5

**2.2. Tìm kiếm theo Chiều sâu (Depth-First Search - DFS)** 7

**2.3. Tìm kiếm theo Chi phí Đồng nhất (Uniform Cost Search - UCS)** 8

**2.4. Tìm kiếm Sâu dần Lặp (Iterative Deepening Search - IDS)** 9

**3. Nhóm Thuật toán Tìm kiếm có Thông tin (Informed Search Algorithms)** 10

**3.1. Hàm Heuristic và Khoảng cách Manhattan** 10

**3.2. Thuật toán Tìm kiếm A\* (A-star Search)** 10

**3.3. Tìm kiếm Tham lam Tốt nhất (Greedy Best-First Search)** 11

**3.4. Tìm kiếm Theo Dải (Beam Search)** 12

**CHƯƠNG II: PHÂN TÍCH VÀ THIẾT KẾ CHI TIẾT HỆ THỐNG** 13

**1. Giới thiệu Kiến trúc Hệ thống và Triết lý Thiết kế** 13

**1.1. Kiến trúc Module hóa** 13

**1.2. Triết lý Thiết kế: Tách biệt các Mối quan tâm (Separation of Concerns)** 14

**2.** **Phân tích và xây dựng thuật toán:** 14

**2.1. Nền tảng của các Thuật toán: Các Hàm Phụ trợ (Helpers)** 14

**2.1.1. Hàm neighbors: Tìm các Bước đi Hợp lệ** 14

**2.1.2. Hàm reconstruct\_path: Lần ngược Dấu vết** 15

**2.2. Xây dựng các thuật toán:** 16

**2.2.1. Thuật toán Tìm kiếm theo Chiều rộng (Breadth-First Search - BFS)** 16

**2.2.2. Phân tích Thuật toán Tìm kiếm theo Chiều sâu (DFS)** 17

**2.2.3. Phân tích Thuật toán Tìm kiếm Chi phí Thống nhất (UCS)** 18

**2.2.4. Phân tích Thuật toán Tham lam (Greedy Best-First Search)** 19

**2.2.5. Phân tích Thuật toán A (A-Star)** 19

**2.2.6. Phân tích Thuật toán Tìm kiếm (Beam Search)** 21

**2.2.7. Phân tích Thuật toán Tìm kiếm Sâu dần (IDS)** 21

**3. Xây dựng giao diện và chức năng:** 23

**3.1. Giao diện chính của chương trình** 23

**3.1.1.Bố cục và khởi tạo chương trình** 23

**3.1.2. Phân tích Bảng Điều khiển (Control Panel)** 23

**3.2. Giao diện mô phỏng bản đồ và chức năng:** 33

**3.2.1 Giao diện Ban đầu và sau khi Reset Map** 33

**3.2.2 Giao diện sau khi Chọn chức năng Random Map** 35

**4.Trực quan hóa thuật toán** 38

**4.1. Duyệt ô và chạy thuật toán:** 38

**4.2. Chức năng Chạy tất cả và So sánh** 41

**4.3. Chức năng di chuyển Robot** 43

**4.4. Bảng so sánh** 45

PHẦN KẾT LUẬN 48

TÀI LIỆU THAM KHẢO 50

# **PHẦN MỞ ĐẦU**

## **1. Lý do chọn đề tài**

Trong bối cảnh công nghiệp hóa và tự động hóa mạnh mẽ của thế kỷ 21, các hệ thống tự hành và robot di động (Autonomous Mobile Robots - AMR) đã trở thành công nghệ cốt lõi trong các lĩnh vực như logistics, giao hàng và thám hiểm. Yếu tố quyết định sự thành công của các hệ thống này chính là khả năng **tìm kiếm đường đi tối ưu (pathfinding)** một cách nhanh chóng, chính xác, và tiết kiệm chi phí. Bài toán này không chỉ đơn thuần là tìm đường đi ngắn nhất mà còn là **bài toán tối ưu hóa đa mục tiêu** trong môi trường có chi phí di chuyển không đồng nhất.

Nghiên cứu này ra đời nhằm giải quyết sự thiếu hụt các công cụ trực quan hóa chuyên sâu, cho phép người học và người nghiên cứu dễ dàng so sánh hiệu suất của các thuật toán tìm kiếm kinh điển trong một kịch bản thực tế, mô phỏng như ứng dụng **"Robot Giao Hàng"**. Cụ thể, môi trường được mô hình hóa là một lưới lớn **25x35 ô** với ba loại địa hình rõ rệt: **Biển** (chi phí thấp, C=1), **Bão tố** (chi phí cao, C=[3,5]) và **Tường** (chi phí vô hạn, C=999). Việc trực quan hóa quá trình khám phá nút và so sánh định lượng các chỉ số như **Thời gian thực thi**, **Số ô đã duyệt**, **Độ dài đường đi** và **Tổng chi phí** là cần thiết để xác định thuật toán nào phù hợp nhất với yêu cầu tối ưu hóa chi phí trong thực tiễn. Do đó, việc xây dựng một nền tảng thực nghiệm linh hoạt và trực quan là lý do cấp thiết để thực hiện đề tài này.

## **2. Mục đích nghiên cứu**

Mục đích nghiên cứu của đề tài là thiết kế, triển khai và đánh giá toàn diện một hệ thống mô phỏng thuật toán tìm đường trên nền tảng Python/Pygame.

Thứ nhất, về mặt **kỹ thuật và triển khai**, mục đích là hiện thực hóa và tích hợp thành công một bộ sưu tập đa dạng các thuật toán tìm kiếm, bao gồm nhóm **Tìm kiếm Mù** (BFS, DFS, UCS, IDS) và nhóm **Tìm kiếm Có thông tin** (A\*, Greedy Best-First Search, Beam Search). Việc này yêu cầu sử dụng các cấu trúc dữ liệu tối ưu như **Hàng đợi (deque)** cho BFS, **Ngăn xếp (list)** cho DFS và **Hàng đợi Ưu tiên (heapq)** cho các thuật toán chi phí và heuristic (UCS, A\*, Greedy).

Thứ hai, về mặt **mô hình hóa**, mục đích là áp dụng hàm heuristic **Manhattan distance** để định hướng cho các thuật toán Informed Search (A\*, Greedy, Beam), đảm bảo chúng có thể ước tính chi phí còn lại đến đích một cách hiệu quả.

Thứ ba, về mặt **phân tích định lượng**, mục đích là cung cấp bảng so sánh chi tiết các thông số hiệu suất cho tất cả các thuật toán trên cùng một kịch bản bản đồ tùy chỉnh hoặc ngẫu nhiên. Kết quả phân tích sẽ làm nổi bật sự đánh đổi giữa tính **hoàn chỉnh, tính tối ưu** và **tốc độ tìm kiếm** của từng phương pháp.

## **3. Đối tượng và phạm vi nghiên cứu**

Đối tượng chính của nghiên cứu là các **cơ chế thuật toán tìm kiếm** được triển khai trong ứng dụng. Bộ thuật toán được nghiên cứu bao gồm 7 phương pháp tìm kiếm cốt lõi: **BFS, DFS, UCS, IDS** (Iterative Deepening Search), **A\*** (A-star Search), **Greedy** (Best-First Search) và **Beam Search**.

Phạm vi nghiên cứu được giới hạn chặt chẽ trong môi trường mô phỏng:

* **Phạm vi Môi trường:** Bản đồ lưới hai chiều cố định kích thước **25 hàng 35 cột**.
* **Phạm vi Di chuyển:** Chỉ cho phép di chuyển theo **bốn hướng** cơ bản (lên, xuống, trái, phải).
* **Phạm vi Chi phí:** Mô hình hóa chi phí di chuyển theo trọng số địa hình rời rạc (C=1 cho Biển, C=[3,5] cho Bão tố, C=999 cho Tường), tập trung vào các bài toán tìm đường có chi phí tối ưu.
* **Phạm vi Công cụ:** Nghiên cứu và phát triển được thực hiện hoàn toàn bằng ngôn ngữ **Python** và thư viện đồ họa **Pygame** để đảm bảo tính trực quan và khả năng tùy biến cao.

## **4. Phương pháp nghiên cứu**

Nghiên cứu được tiến hành dựa trên sự kết hợp hài hòa giữa phương pháp nghiên cứu lý thuyết và phương pháp thực nghiệm khoa học.

Đầu tiên, **Phương pháp Nghiên cứu Lý thuyết** được áp dụng để phân tích chuyên sâu về lý thuyết đồ thị và cơ chế hoạt động, tính tối ưu và độ phức tạp về thời gian/không gian của từng thuật toán trong các tài liệu tham khảo chính thống. Đặc biệt, nghiên cứu kỹ lưỡng về vai trò của hàm **Manhattan Heuristic** trong việc cải thiện hiệu suất của A\* và Greedy Search.

Thứ hai, **Phương pháp Mô hình hóa và Lập trình** được sử dụng để chuyển hóa lý thuyết thành ứng dụng. Giai đoạn này bao gồm việc thiết lập cấu trúc dữ liệu cho lưới (mảng 25x35 cho tile và weight) và lập trình các hàm tìm kiếm. Việc sử dụng các cấu trúc dữ liệu Python chuyên biệt (deque, heapq) được ưu tiên để đảm bảo hiệu suất tính toán tối đa cho quá trình duyệt nút.

Cuối cùng, **Phương pháp Thực nghiệm và Phân tích Dữ liệu** được áp dụng. Bằng cách sử dụng chức năng "Chạy tất cả & So sánh", nghiên cứu tiến hành thu thập dữ liệu định lượng (thời gian, số nút duyệt, chi phí) từ các thử nghiệm trên nhiều kịch bản bản đồ khác nhau (tùy chỉnh và ngẫu nhiên). Dữ liệu này sau đó được sử dụng để xây dựng bảng so sánh chi tiết, làm cơ sở cho việc đánh giá và đưa ra kết luận về thuật toán có hiệu quả tối ưu nhất trong bài toán Robot Giao Hàng đa chi phí.

# **PHẦN NỘI DUNG**

## **CHƯƠNG I: CƠ SỞ LÝ THUYẾT VỀ CÁC THUẬT TOÁN TÌM KIẾM**

### **1. Khái quát về Bài toán Tìm kiếm trong Trí tuệ Nhân tạo**

Bài toán tìm kiếm trong Trí tuệ Nhân tạo được định nghĩa là một quy trình tìm kiếm một chuỗi các hành động (hay đường đi) chuyển tác nhân từ trạng thái ban đầu (S) đến một trạng thái mục tiêu (G) trong một không gian trạng thái đã xác định.

#### **1.1. Khái niệm cơ bản**

* **Không gian Trạng thái (State Space):** Là tập hợp tất cả các cấu hình khả thi mà tác nhân (robot) có thể đạt tới. Trong dự án này, không gian trạng thái là lưới 25x35 ô.
* **Trạng thái Ban đầu (Start State):** Vị trí xuất phát của robot, được người dùng chỉ định (biến start trong mã nguồn).
* **Hành động (Actions) và Hàm Kế thừa (Successor Function):** Là các hành động di chuyển cơ bản (Lên, Xuống, Trái, Phải). Hàm kế thừa sẽ trả về tập hợp các ô lân cận hợp lệ (biến neighbors(pos)), loại trừ các ô Tường.
* **Kiểm tra Mục tiêu (Goal Test):** Hàm kiểm tra xem trạng thái hiện tại có trùng với trạng thái đích (G) hay không (biến goal).
* **Chi phí Đường đi (Path Cost):** Tổng trọng số của tất cả các ô trên đường đi, phản ánh sự tiêu hao năng lượng và thời gian di chuyển. Đây là tiêu chí cần được tối ưu hóa.

#### **1.2. Phân loại Thuật toán Tìm kiếm**

Các thuật toán tìm kiếm được chia thành hai nhóm lớn dựa trên cách chúng sử dụng thông tin về đích đến:

* **Tìm kiếm không có thông tin (Uninformed Search):** Còn được gọi là tìm kiếm mù, là các thuật toán không sử dụng bất kỳ thông tin nào về khoảng cách hoặc chi phí ước tính đến đích. Chúng hoạt động theo kiểu thử nghiệm, nghĩa là chúng thử nghiệm mọi phần của không gian tìm kiếm một cách mù quáng.
* **Tìm kiếm có Thông tin (Informed Search):** **chiến lược tìm kiếm có thông tin sử dụng kiến ​​thức bổ sung vượt ra ngoài những gì chúng ta cung cấp trong định nghĩa vấn đề.** Kiến thức bổ sung này có được thông qua một hàm gọi là phương pháp tìm kiếm tự nhiên (heuristic). **Hàm này nhận một trạng thái đầu vào và ước tính mức độ gần với mục tiêu**. **Sử dụng phương pháp tìm kiếm tự nhiên, một chiến lược tìm kiếm có thể phân biệt giữa các trạng thái không phải mục tiêu và tập trung vào những trạng thái có vẻ hứa hẹn hơn.** Đó là lý do tại sao các kỹ thuật tìm kiếm có thông tin có thể tìm thấy mục tiêu nhanh hơn một thuật toán không có thông tin, với điều kiện hàm phương pháp tìm kiếm tự nhiên được xác định rõ ràng. [1]

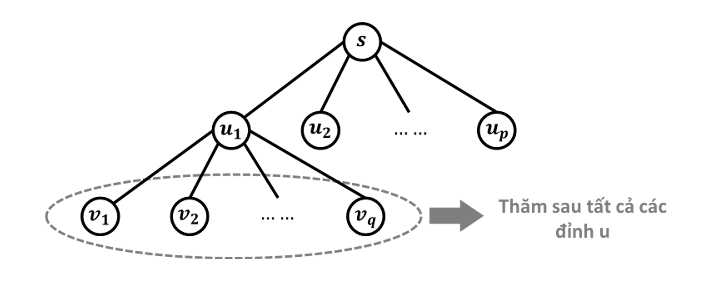
### **2. Nhóm Thuật toán Tìm kiếm Mù (Uninformed Search Algorithms)**

Nhóm thuật toán này (BFS, DFS, UCS, IDS) rất quan trọng vì chúng cung cấp nền tảng cơ bản và đảm bảo tính hoàn chỉnh (completeness) trong các không gian trạng thái hữu hạn.

#### **2.1. Tìm kiếm theo Chiều rộng (Breadth-First Search - BFS)**

BFS hoạt động bằng cách mở rộng tất cả các nút tại một độ sâu (d) trước khi chuyển sang các nút ở độ sâu (d+1).

* **Cơ chế:** BFS sử dụng cấu trúc dữ liệu **Hàng đợi (Queue)** theo nguyên tắc **FIFO (First-In, First-Out)** để lưu trữ các nút biên (q = deque([start]) trong mã nguồn). Nút được đưa vào trước sẽ được mở rộng trước.



**Cách thực hiện :**

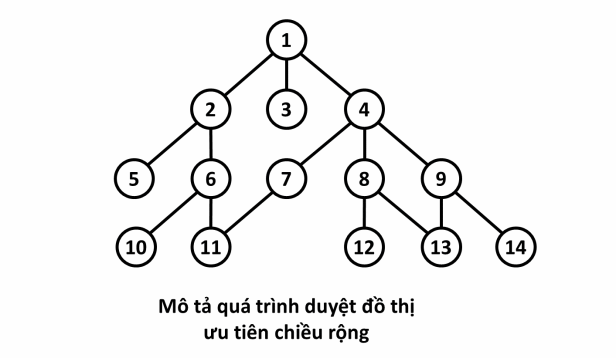
**Bước 1:** Khởi tạo

* Các đỉnh đều ở trạng thái chưa được đánh dấu. Ngoại trừ đỉnh nguồn  đã được đánh dấu.
* Một hàng đợi ban đầu chỉ chứa 1 phần tử là s.

**Bước 2:** Lặp lại các bước sau cho đến khi hàng đợi rỗng:

* Lấy đỉnh u ra khỏi hàng đợi.
* Xét tất cả những đỉnh v kề với u mà chưa được đánh dấu, với mỗi đỉnh v đó:
  + Đánh dấu v đã thăm.
  + Lưu lại vết đường đi từ u  đến v .
  + Đẩy v  vào trong hàng đợi (đỉnh v  sẽ chờ được duyệt tại những bước sau).

**Bước 3:** Truy vết tìm đường đi.

[2]

* **Tính chất:**
  + **Hoàn chỉnh:** Luôn tìm thấy giải pháp nếu có.
  + **Tối ưu:** BFS là tối ưu **về độ dài đường đi** nếu chi phí di chuyển giữa các nút là đồng nhất (hoặc bằng nhau). Trong bài toán Robot Giao Hàng, BFS có thể bị lỗi tối ưu về *chi phí* nếu nó chọn đường đi ngắn hơn nhưng lại đi qua nhiều ô "Bão tố" hơn so với đường đi dài hơn.
  + **Nhược điểm** chính là chi phí bộ nhớ rất lớn, đặc biệt đối với các không gian tìm kiếm rộng, vì nó phải lưu trữ tất cả các nút tại mức độ sâu hiện tại để thăm trong tương lai.
  + **Ví dụ:** Trên lưới 25x35 của dự án, khi Robot di chuyển trên địa hình nước (chi phí c=1 ), BFS sẽ tìm kiếm giống như một làn sóng hình vuông lan tỏa. Nếu đích đến nằm cách Start 10 bước, BFS sẽ tìm ra đường đi 10 bước này đầu tiên, bất kể đường đi đó có đi qua một ô Bão (chi phí 3-5 ) nào hay không, vì nó chỉ quan tâm đến số lượng ô đã đi.

#### **2.2. Tìm kiếm theo Chiều sâu (Depth-First Search - DFS)**

**Depth-First Search (DFS)** áp dụng nguyên tắc tìm kiếm dọc theo một nhánh cụ thể càng sâu càng tốt trước khi thực hiện **quay lui (backtrack).**

* **Cơ chế:** DFS sử dụng cấu trúc dữ liệu **Ngăn xếp (Stack)** theo nguyên tắc **LIFO (Last-In, First-Out)** (stack = [start] trong mã nguồn). Điều này giúp nó khám phá sâu một nhánh hoàn chỉnh trước khi xem xét các nhánh khác.

**Cách thực hiện :**

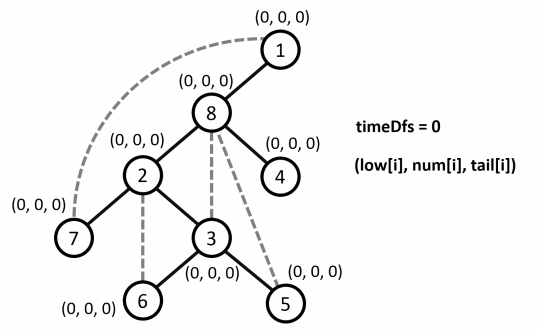
Đầu tiên, ta sẽ bắt đầu duyệt **DFS** từ đỉnh gốc. Khi duyệt tới đỉnh **u**, ta sẽ cập nhật thời điểm duyệt tới. Lúc này ta có **low[u] = num[u] =** thứ tự duyệt DFS. Ta sẽ duyệt tất cả các đỉnh con **v** trong gốc **u**.

Trong **trường hợp 1**, nếu đỉnh **v** chưa được thăm thì sau khi hoàn thành **DFS** của **v**, ta sẽ cập nhật lại giá trị của **low[u]** theo công thức **low[u] = min(low[u], low[v])**.

Trong **trường hợp 2**, nếu đỉnh **v** đã được thăm, thì ta sẽ cập nhật lại giá trị cho **low[u]** theo công thức **low[u] = min(low[u], num[v])**. Ở trường hợp này, ta không thể cập nhật **low[u] = min(low[u], low[v])**, vì khi ta thăm đến đỉnh **u** mà đỉnh **v** đã được thăm thì cạnh (**u**, **v**) là một cạnh nét đứt. Do đó, khi đi từ **u** tới **v** ta đã sử dụng một cạnh nét đứt nên không thể tiếp tục di chuyển nữa (theo định nghĩa của mảng **low[]**). Vì vậy, ta chỉ cập nhật **low[u] = min(low[u], num[v])**.

**Chú ý:** Nếu **v** là **cha trực tiếp** của **u** thì ta **bỏ qua, không xét đến**.

Khi đã duyệt xong đỉnh **u** và các nút trong cây con **DFS** gốc **u**, ta sẽ tiến hành cập nhật giá trị **tail[u] =** thời gian duyệt DFS hiện tại.

[3]

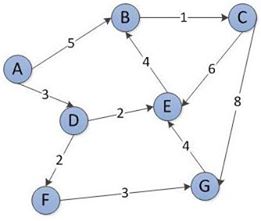
* **Tính chất:**
  + **Hoàn chỉnh:** Không hoàn chỉnh trong không gian trạng thái vô hạn hoặc khi bị kẹt trong vòng lặp. Tuy nhiên, nó là hoàn chỉnh trong đồ thị hữu hạn (như lưới 25 x 35) nếu có cơ chế chống lặp (biến visited và parent trong mã nguồn).
  + **Tối ưu:** DFS có **chi phí bộ nhớ cực kỳ thấp** so với BFS (chỉ cần lưu trữ đường đi hiện tại) và rất nhanh nếu đích nằm sâu trong một nhánh sớm được thăm dò
  + **Nhược điểm:** DFS **không đảm bảo tính tối ưu về độ dài** hay chi phí; nó có thể tìm thấy một đường đi rất dài và không hiệu quả nếu đích ở ngay bên cạnh Start nhưng lại nằm trên một nhánh chưa được khám phá.
  + **Ví dụ:** Trong lưới của Robot, DFS sẽ chọn một hướng (ví dụ: lên trên) và đi thẳng cho đến khi gặp Tường (Wall) hoặc biên. Sau đó, nó quay lại và chọn hướng chưa đi khác (ví dụ: sang phải), lặp lại quá trình này. Đường đi tìm thấy có thể là một vòng xoắn ốc dài gấp đôi đường đi ngắn nhất.

#### **2.3. Tìm kiếm theo Chi phí Đồng nhất (Uniform Cost Search - UCS)**

UCS là phiên bản tổng quát của BFS, được thiết kế để tìm kiếm đường đi có tổng chi phí tích lũy thấp nhất.

* **Cơ chế:** UCS sử dụng cấu trúc dữ liệu **Hàng đợi Ưu tiên (Priority Queue)**, được triển khai bằng thư viện heapq trong Python (pq = [(0, start)]). Thứ tự ưu tiên mở rộng nút u được xác định bởi chi phí thực tế từ điểm bắt đầu đến u, ký hiệu là g(u). Nút có g(u) nhỏ nhất sẽ được mở rộng trước.

**Cách thực hiện:**

 [4]

* **Tính chất:**
  + **Hoàn chỉnh:** Hoàn chỉnh nếu chi phí hành động không âm.
  + **Tối ưu:** Là thuật toán tìm kiếm tối ưu, đảm bảo tìm ra đường đi có **tổng chi phí thấp nhất** trong môi trường đa chi phí (Sea, Storm) của dự án. Điều này khiến UCS trở thành một thuật toán so sánh tiêu chuẩn cho các thuật toán tối ưu khác như A\*.
  + **Nhược điểm:** Hạn chế là nó phải duyệt qua rất nhiều nút để đảm bảo rằng không có tuyến đường rẻ hơn bị bỏ sót.
  + **Ví dụ:** Giả sử có hai tuyến đường đến Goal. Tuyến A dài 10 bước, chỉ qua ô Nước (g(n)=10). Tuyến B dài 5 bước, nhưng 4 bước qua ô Bão (chi phí 3 ), tổng g(n)=13 . UCS sẽ tìm ra Tuyến A (chi phí g=10 ) trước, mặc dù nó dài hơn về số bước.

#### **2.4. Tìm kiếm Sâu dần Lặp (Iterative Deepening Search - IDS)**

**Iterative Deepening Search (IDS)** là một siêu thuật toán (meta-algorithm) kết hợp tính **tối ưu về độ dài** của BFS với tính **tiết kiệm bộ nhớ** của DFS.

* **Cơ chế:** IDS thực hiện một chuỗi các lần chạy **DFS giới hạn độ sâu**, bắt đầu với giới hạn độ sâu L=1 , sau đó L=2 ,L=3 , và cứ thế tiếp tục cho đến khi tìm thấy đích. Khi tìm thấy đích tại độ sâu d , thuật toán sẽ dừng lại.
* **Tính chất:**
  + **Hoàn chỉnh:** Hoàn chỉnh.
  + **Tối ưu:** Tối ưu nếu chi phí hành động là đồng nhất (giống BFS).
  + **Bộ nhớ:** Yêu cầu bộ nhớ thấp như DFS (O(bd)), nhưng lại đảm bảo tìm được lời giải tối ưu về độ dài như BFS. Đây là một sự đánh đổi giữa hiệu quả bộ nhớ và tính toán.
  + **Ví dụ:** Nếu đích nằm ở độ sâu d=5 , IDS sẽ thực hiện 5 lần DFS giới hạn độ sâu từ L=1 đến L=5 . Lần chạy sẽ tìm thấy đích, và đường đi đó sẽ là đường đi ngắn nhất về số bước, tương đương với kết quả của BFS nhưng chỉ tốn bộ nhớ O(bd)

### **3. Nhóm Thuật toán Tìm kiếm có Thông tin (Informed Search Algorithms)**

Nhóm này sử dụng một hàm ước lượng – **Hàm Heuristic (h(n))** – để đánh giá độ gần của nút hiện tại với đích đến, giúp dẫn đường tìm kiếm hiệu quả hơn nhiều so với nhóm uninformed search.

#### **3.1. Hàm Heuristic và Khoảng cách Manhattan**

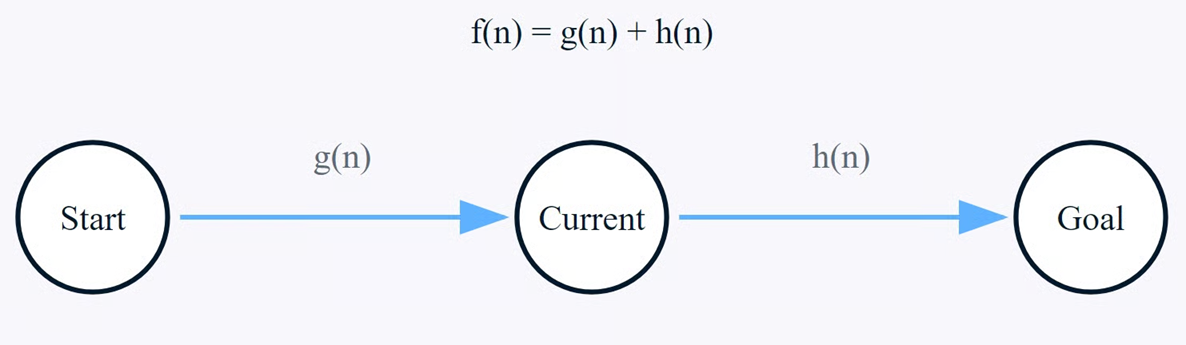
Trong các thuật toán tìm kiếm có thông tin, một hàm ước tính h(n) được sử dụng để đánh giá chi phí từ nút hiện tại n đến đích G .

* Manhattan Distance (Khoảng cách Manhattan): Đây là hàm heuristic được sử dụng trong mã nguồn (manhattan(a,b)). Nó được tính bằng tổng khoảng cách tuyệt đối theo phương ngang và phương dọc giữa hai ô:

Trong bài toán lưới di chuyển 4 hướng (Lên, Xuống, Trái, Phải) với chi phí di chuyển ≥1, khoảng cách Manhattan là một Heuristic Hợp lệ (Admissible). Tức là, nó không bao giờ đánh giá quá cao chi phí thực tế để đến đích, đảm bảo tính tối ưu của thuật toán A\*.

#### **3.2. Thuật toán Tìm kiếm A\* (A-star Search)**

A\* là thuật toán tìm kiếm có thông tin phổ biến nhất, nổi tiếng về tính tối ưu và hiệu quả.



* Cơ chế: A\* đánh giá mỗi nút n bằng một hàm chi phí tổng:

Trong đó:

* + : Chi phí thực tế từ điểm bắt đầu đến nút (đã được tính toán, giống như UCS).
  + : Chi phí ước tính (heuristic) từ đến đích (sử dụng Manhattan Distance).

A\* cũng sử dụng Hàng đợi ưu tiên (heapq) và ưu tiên mở rộng nút có f(n) nhỏ nhất.

* **Tính chất:**
  + **Tối ưu:** A\* là tối ưu **nếu** hàm là *hợp lệ* (admissible) (luôn ước tính chi phí nhỏ hơn hoặc bằng chi phí thực tế). Trong dự án, do sử dụng Manhattan Distance, A\* đảm bảo tìm ra đường đi tối ưu về chi phí.
  + **Hoàn chỉnh:** Hoàn chỉnh.
  + **Ví dụ:** A\* sẽ xem xét tuyến Bão (chi phí 15) và tuyến Nước (chi phí 8) cùng một lúc. Khi Robot di chuyển, A\* sẽ ưu tiên mở rộng các nút trên tuyến Nước trước vì tổng của chúng thấp hơn, và cuối cùng sẽ tìm ra đường đi chi phí 8.

#### **3.3. Tìm kiếm Tham lam Tốt nhất (Greedy Best-First Search)**

Greedy Search chỉ tập trung vào việc giảm thiểu chi phí ước tính đến đích, bỏ qua chi phí thực tế đã đi.

* **Cơ chế:** Thuật toán chỉ sử dụng hàm h(n) để đánh giá và ưu tiên mở rộng nút có giá trị h(n) nhỏ nhất. Mục tiêu là luôn tiến gần đến đích nhất có thể theo ước tính heuristic (heapq ưu tiên manhattan(v, goal)).
* **Tính chất:**
  + **Nhanh chóng:** Thường nhanh hơn A\* và UCS vì nó có sự dẫn hướng mạnh mẽ.
  + **Không Tối ưu:** Dễ mắc kẹt trong các con đường "tham lam" dẫn đến chi phí cao. Ví dụ, nó có thể chọn đi qua một ô Bão tố chỉ vì nó giúp nó đến gần đích hơn về mặt khoảng cách, thay vì chọn một con đường vòng dài hơn nhưng chỉ toàn ô Biển.
  + **Nhược điểm:** Việc bỏ qua chi phí thực tế đã đi () là nhược điểm lớn nhất.
  + **Ví dụ:** Robot có thể chọn đi qua một chuỗi 5 ô Bão (chi phí mỗi ô) chỉ vì chuỗi này có h(n) thấp hơn một chuỗi 8 ô Nước (chi phí mỗi ô) ở thời điểm ban đầu. Kết quả là, tổng chi phí của đường đi Greedy (tức là 5x3=15 ) lại cao hơn hẳn đường đi khác (tức là 8x1). Greedy Search chỉ tối ưu về tốc độ mà không tối ưu về chi phí.

#### **3.4. Tìm kiếm Theo Dải (Beam Search)**

Beam Search là một biến thể của Greedy Search, được sử dụng để giảm đáng kể yêu cầu về bộ nhớ và thời gian tính toán.

* **Cơ chế:** Thay vì giữ tất cả các nút lân cận tiềm năng trong biên (frontier), Beam Search chỉ giữ lại một số lượng nút tốt nhất cố định, được gọi là **Chiều rộng Dải (Beam Width - B)** (biến beam\_width=8 trong mã nguồn). Các nút được giữ lại là những nút có giá trị heuristic h(n) tốt nhất.
* **Tính chất:**
  + **Hiệu quả về Bộ nhớ:** Rất hiệu quả về bộ nhớ.
  + **Không Hoàn chỉnh và Không Tối ưu:** Do giới hạn kích thước dải B, thuật toán có thể bỏ qua một đường đi tối ưu (hoặc thậm chí là đường đi duy nhất) nếu nó không nằm trong B nút tốt nhất. Beam Search là một kỹ thuật đánh đổi giữa hiệu suất và chất lượng lời giải.
  + **Ví dụ:** Nếu , ở mỗi bước, Robot chỉ giữ lại 3 ô tiếp theo có vẻ gần đích nhất, bất kể có 10 ô khả thi khác. Điều này giúp giảm bộ nhớ, nhưng có thể bỏ lỡ một con đường đi tuyệt vời nằm ở ô thứ 4.

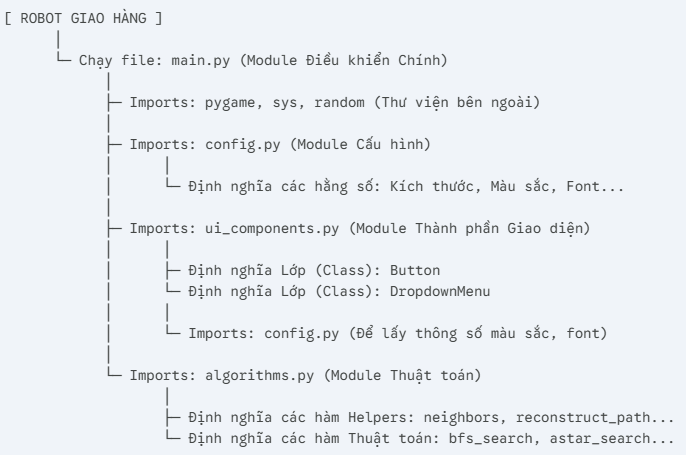
## **CHƯƠNG II: PHÂN TÍCH VÀ THIẾT KẾ CHI TIẾT HỆ THỐNG**

### **1. Giới thiệu Kiến trúc Hệ thống và Triết lý Thiết kế**

#### **1.1. Kiến trúc Module hóa**

Để quản lý độ phức tạp và thúc đẩy khả năng bảo trì, hệ thống được thiết kế theo kiến trúc module hóa. Nguyên tắc cốt lõi được áp dụng là **Tách biệt các Mối quan tâm (Separation of Concerns)**, trong đó mỗi thành phần của hệ thống chỉ chịu trách nhiệm cho một chức năng duy nhất.

Kiến trúc phần mềm được phân rã thành bốn module chính:



* **Module Cấu hình (config.py):** Đóng vai trò là nguồn chân lý duy nhất cho các giá trị tĩnh và hằng số của hệ thống.
* **Module Thành phần Giao diện (ui\_components.py):** Áp dụng lập trình hướng đối tượng để tạo ra các thành phần UI có thể tái sử dụng.
* **Module Thuật toán (algorithms.py):** Chứa logic "não bộ" của ứng dụng, hoàn toàn độc lập với phần giao diện đồ họa.
* **Module Điều khiển Chính (main.py):** Đóng vai trò là người điều phối, kết nối sự kiện người dùng, trạng thái ứng dụng và quá trình hiển thị.

#### **1.2. Triết lý Thiết kế: Tách biệt các Mối quan tâm (Separation of Concerns)**

* **Logic Thuật toán** hoàn toàn tách biệt với **Giao diện Đồ họa**. Các hàm trong algorithms.py có thể hoạt động mà không cần đến Pygame.
* **Thành phần Giao diện** không chứa logic nghiệp vụ phức tạp. Lớp Button chỉ biết cách tự vẽ và kiểm tra va chạm, nó không biết thuật toán A\* là gì.
* **Logic Điều khiển Chính** không trực tiếp triển khai các thuật toán phức tạp hay vẽ các thành phần chi tiết, mà nó ủy quyền các nhiệm vụ đó cho các module chuyên biệt.

Cách tiếp cận này giúp mã nguồn trở nên sạch sẽ, dễ kiểm thử, và cực kỳ linh hoạt để nâng cấp hoặc thay đổi trong tương lai.

### **2.** **Phân tích và xây dựng thuật toán:**

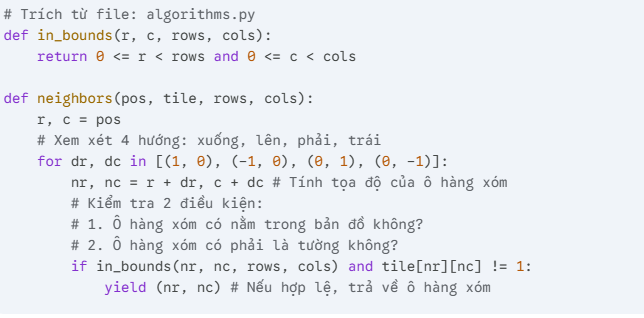
Đây là phần cốt lõi của chương trình, nơi logic tìm đường được triển khai. Toàn bộ phần này được thiết kế để hoạt động độc lập với giao diện, chỉ xử lý dữ liệu đầu vào và trả về kết quả.

#### **2.1. Nền tảng của các Thuật toán: Các Hàm Phụ trợ (Helpers)**

Trước khi đi vào từng thuật toán cụ thể, chúng ta cần phân tích các "công cụ" cơ bản được định nghĩa trong file algorithms.py mà tất cả các thuật toán đều sử dụng.

##### **2.1.1. Hàm neighbors: Tìm các Bước đi Hợp lệ**

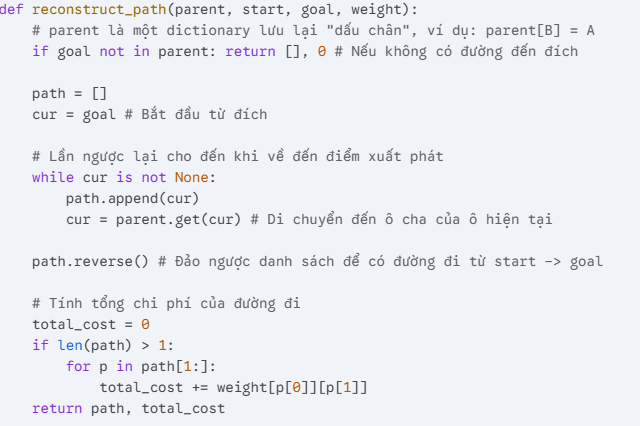
**Chức năng**: Từ một ô bất kỳ, hàm này có nhiệm vụ tìm ra tất cả các ô xung quanh mà robot có thể di chuyển tới.



**Giải thích chi tiết cách hoạt động:** Hàm này nhận đầu vào là tọa độ của một đỉnh pos. Nó thực hiện duyệt qua bốn hướng cơ bản (lên, xuống, trái, phải) để xác định tọa độ các đỉnh kề. Đối với mỗi đỉnh kề, hàm tiến hành hai bước kiểm tra: (1) tính hợp lệ về không gian (đỉnh có nằm trong giới hạn của lưới không) thông qua hàm in\_bounds, và (2) tính hợp lệ về địa hình (đỉnh không phải là chướng ngại vật, tức tile[nr][nc] != 1). Các đỉnh thỏa mãn cả hai điều kiện sẽ được trả về thông qua cơ chế yield, biến hàm này thành một generator để tối ưu hóa việc sử dụng bộ nhớ.

##### **2.1.2. Hàm reconstruct\_path: Lần ngược Dấu vết**

**Chức năng:** Sau khi thuật toán đã tìm thấy ô đích, hàm này có nhiệm vụ xây dựng lại toàn bộ con đường đi từ đầu đến cuối.

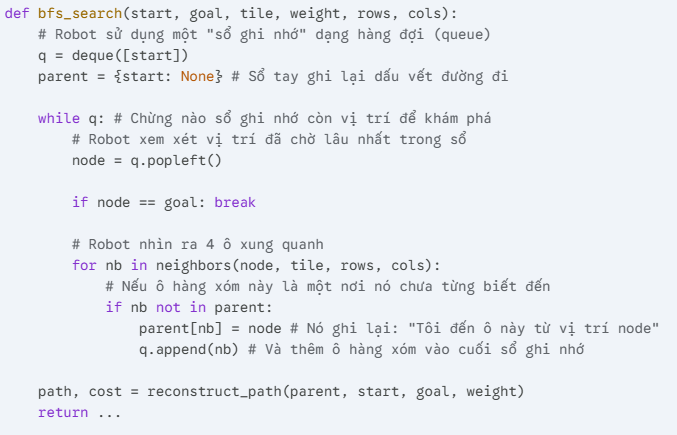
**Giải thích chi tiết cách hoạt động:** Các thuật toán tìm kiếm trong quá trình hoạt động sẽ duy trì một cấu trúc dữ liệu parent (thường là một dictionary) để lưu lại vết của đường đi (predecessor of each node). Hàm này thực hiện một quy trình truy vết ngược, bắt đầu từ đỉnh goal, lần theo các đỉnh cha đã được lưu trong parent cho đến khi gặp đỉnh start. Kết quả là một danh sách các đỉnh theo thứ tự ngược, sau đó được đảo lại để có được đường đi hoàn chỉnh. Cuối cùng, hàm tính toán tổng trọng số (chi phí) của đường đi dựa trên ma trận weight.

#### **2.2. Xây dựng các thuật toán:**

##### **2.2.1. Thuật toán Tìm kiếm theo Chiều rộng (Breadth-First Search - BFS)**

**Chiến lược của Robot (Nguyên lý):** Robot BFS là một nhà thám hiểm cực kỳ cẩn thận và có hệ thống. Nó hoạt động theo phương châm "khám phá hết những gì ở gần trước khi đi xa hơn". Nó sẽ kiểm tra tất cả các ô ngay sát cạnh nó, sau đó mới đến các ô xa hơn một bước, rồi hai bước, và cứ thế tiếp tục.

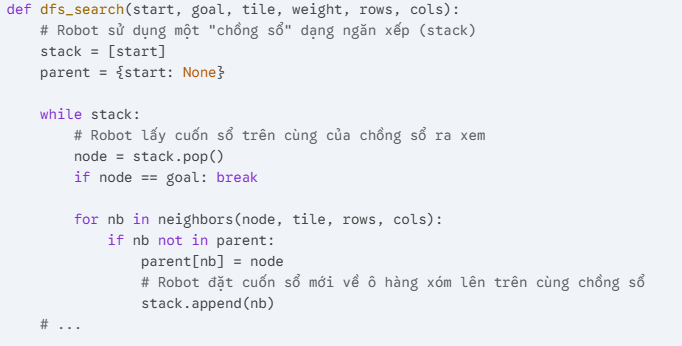
**Mục tiêu trong Hệ thống:** Mục tiêu của Robot BFS là tìm ra con đường đến khách hàng với **số lần di chuyển (số ô) ít nhất có thể**. Nó hoàn toàn không quan tâm đến "nhiên liệu" (chi phí). Việc đi qua một ô "Đường đi" (chi phí 1) hay một ô "Bão" (chi phí 5) đối với nó đều chỉ được tính là một bước đi.



**Mô tả hoạt động trong Hệ thống:** Robot bắt đầu tại điểm start và ghi nó vào "sổ ghi nhớ" (q). Sau đó, nó bắt đầu một chu trình làm việc không ngừng. Trong mỗi chu trình, nó lấy ra vị trí đã được ghi vào sổ sớm nhất để khám phá. Nó nhìn ra 4 hướng xung quanh, và với mỗi ô hàng xóm hợp lệ (không phải Tường) mà nó chưa từng biết đến, nó sẽ ghi lại dấu vết và thêm ô đó vào cuối sổ. Vì sổ ghi nhớ hoạt động theo kiểu "vào trước, ra trước", robot sẽ khám phá hết tất cả các ô cách nó 1 bước, rồi mới đến tất cả các ô cách 2 bước. Khi nhìn vào giao diện, ta sẽ thấy vùng màu xanh nhạt (ô đã duyệt) lan tỏa ra từ robot một cách đồng đều như sóng nước. Kết quả là, con đường nó tìm thấy chắc chắn là con đường có ít ô nhất.

##### **2.2.2. Phân tích Thuật toán Tìm kiếm theo Chiều sâu (DFS)**

* **Chiến lược của Robot (Nguyên lý):** Robot DFS là một nhà thám hiểm liều lĩnh và có phần bốc đồng. Nó chọn một hướng và đi một mạch cho đến khi nào gặp ngõ cụt (Tường) thì thôi. Chỉ khi không thể đi tiếp, nó mới chịu quay lại một bước để thử một ngã rẽ khác.
* **Mục tiêu trong Hệ thống:** Mục tiêu của Robot DFS là tìm ra **một con đường bất kỳ** đến khách hàng càng nhanh càng tốt, không hề quan tâm đến việc đường có dài hay tốn nhiên liệu hay không.

**Mô tả hoạt động trong Hệ thống:** Robot DFS sử dụng một "chồng sổ" (stack). Nó bắt đầu bằng cách đặt vị trí start lên chồng sổ. Trong mỗi chu trình, nó lấy cuốn sổ trên cùng ra xem. Sau khi tìm thấy một ô hàng xóm hợp lệ và chưa biết, nó sẽ ngay lập tức ghi lại thông tin và đặt một cuốn sổ mới về ô hàng xóm đó **lên trên cùng** của chồng sổ. Điều này khiến cho ở chu trình tiếp theo, nó lại tiếp tục làm việc với cuốn sổ vừa mới đặt lên. Hành vi này buộc robot phải liên tục đi sâu vào một con đường duy nhất. Khi quan sát trên giao diện, ta sẽ thấy vùng màu xanh nhạt phát triển thành một đường dài, ngoằn ngoèo. Con đường mà Robot DFS tìm ra thường rất kỳ lạ, không hiệu quả, nhưng nó lại sử dụng rất ít bộ nhớ vì nó không cần phải nhớ nhiều ngã rẽ cùng một lúc.

##### **2.2.3. Phân tích Thuật toán Tìm kiếm Chi phí Thống nhất (UCS)**

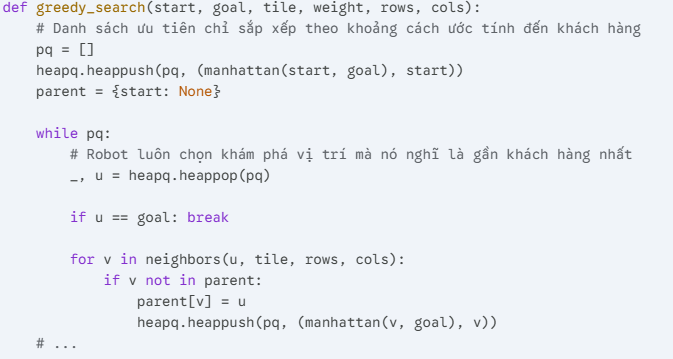
* **Chiến lược của Robot (Nguyên lý):** Robot UCS là một người giao hàng cực kỳ tiết kiệm. Ưu tiên hàng đầu của nó là hoàn thành chuyến đi với **tổng lượng "nhiên liệu" (chi phí) tiêu tốn ít nhất**.
* **Mục tiêu trong Hệ thống:** Tìm ra con đường có tổng trọng số từ các ô (weight) là nhỏ nhất. Nó hiểu rằng đi qua ô "Bão" sẽ tốn nhiều nhiên liệu hơn đi qua ô "Đường đi".



**Mô tả hoạt động trong Hệ thống:** Robot UCS sử dụng một "danh sách ưu tiên" (pq), nơi các vị trí được sắp xếp dựa trên tổng nhiên liệu đã tiêu thụ để đến đó (g). Nó bắt đầu với chi phí 0 tại start. Trong mỗi chu trình, nó luôn chọn khám phá vị trí u mà nó có thể đến được với lượng nhiên liệu ít nhất. Khi xem xét một ô hàng xóm v, nó sẽ tính toán tổng nhiên liệu mới (newg) nếu đi qua u. Nếu con đường mới này tiết kiệm nhiên liệu hơn con đường cũ đã biết để đến v, nó sẽ cập nhật lại thông tin và đưa v vào danh sách ưu tiên. Bằng cách luôn mở rộng từ vị trí "rẻ" nhất, robot sẽ lan tỏa vùng tìm kiếm của mình ra các khu vực có chi phí thấp trước. Nếu có một con đường ngắn nhưng phải đi qua nhiều ô "Bão", và một con đường dài hơn nhưng chỉ đi qua "Đường đi", Robot UCS chắc chắn sẽ chọn con đường dài hơn để tiết kiệm nhiên liệu.

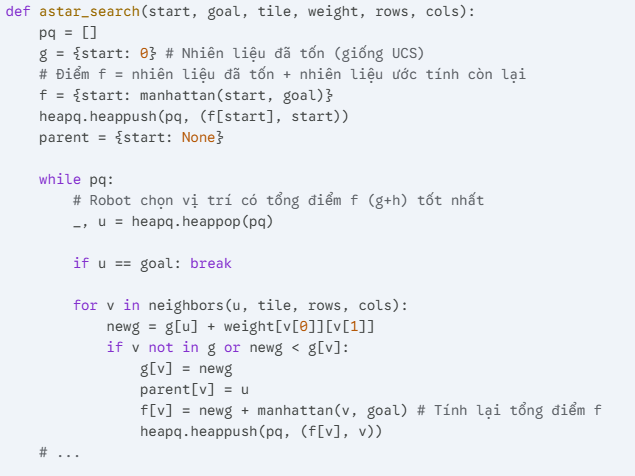
##### **2.2.4. Phân tích Thuật toán Tham lam (Greedy Best-First Search)**

* **Chiến lược của Robot (Nguyên lý):** Robot Greedy là một robot thiếu kiên nhẫn, lúc nào cũng chỉ "nhìn" về phía khách hàng. Ở mỗi ngã rẽ, nó sẽ luôn chọn con đường nào mà nó "nghĩ" là sẽ đưa nó đến gần khách hàng nhất, bất chấp mọi thứ khác.
* **Mục tiêu trong Hệ thống:** Tìm một đường đi đến khách hàng thật nhanh, dựa hoàn toàn vào ước tính khoảng cách. Nó không quan tâm đến nhiên liệu đã tốn và cũng không đảm bảo đường đi là tốt nhất.

**Mô tả hoạt động trong Hệ thống:** Robot Greedy cũng dùng danh sách ưu tiên, nhưng tiêu chí sắp xếp duy nhất là khoảng cách Manhattan (manhattan(v, goal)), tức là khoảng cách ước tính đến khách hàng. Nó không hề có biến g để ghi nhớ chi phí đã đi. Trong mỗi chu trình, nó luôn chọn khám phá ô u có khoảng cách ước tính tới đích ngắn nhất. Hành vi "tham lam" này giúp nó lao rất nhanh về phía khách hàng trên các bản đồ thoáng. Tuy nhiên, nó rất dễ bị "lừa". Nếu có một bức tường hình chữ U, robot có thể đi vào đáy của chữ U vì nơi đó gần khách hàng nhất về mặt hình học, nhưng thực tế đó lại là một ngõ cụt.

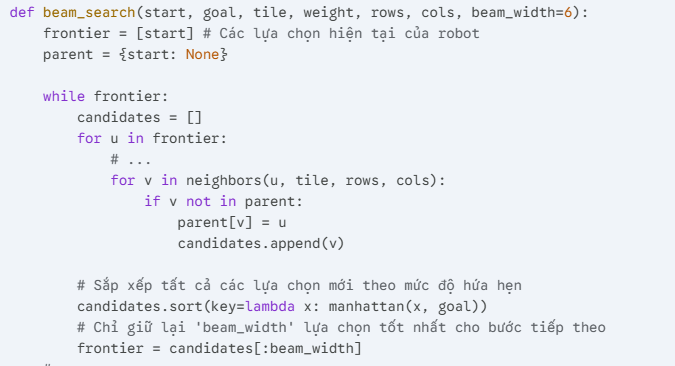
##### **2.2.5. Phân tích Thuật toán A (A-Star)**

* **Chiến lược của Robot (Nguyên lý):** Robot A\* là con robot thông minh và cân bằng nhất. Nó là sự kết hợp hoàn hảo giữa Robot UCS (tiết kiệm) và Robot Greedy (nhìn xa). Nó vừa tính toán lượng nhiên liệu đã tiêu tốn, vừa ước tính quãng đường còn lại để đưa ra quyết định tốt nhất.
* **Mục tiêu trong Hệ thống:** Tìm ra con đường có **tổng chi phí nhiên liệu thấp nhất** (giống UCS) nhưng với tốc độ và hiệu quả tìm kiếm cao hơn nhiều (nhờ khả năng định hướng của Greedy).

**Mô tả hoạt động trong Hệ thống:** Robot A\* sử dụng một danh sách ưu tiên được sắp xếp theo điểm f, với f = g + h. g là tổng chi phí nhiên liệu thực tế để đến một ô (giống hệt UCS), và h là chi phí nhiên liệu ước tính để từ ô đó đến khách hàng (giống Greedy). Trong mỗi chu trình, robot sẽ chọn khám phá ô có tổng điểm f thấp nhất. Điều này có nghĩa là nó ưu tiên những con đường vừa ngắn (chi phí g thấp) lại vừa đi đúng hướng (chi phí h thấp). Sự kết hợp thông minh này giúp nó tránh được việc khám phá các nhánh rẽ vô ích ở xa, tập trung vùng tìm kiếm của mình thành một hình elip hướng về phía đích. Kết quả là A\* vừa đảm bảo tìm ra con đường tối ưu về chi phí, vừa có tốc độ tìm kiếm vượt trội so với UCS.

##### **2.2.6. Phân tích Thuật toán Tìm kiếm (Beam Search)**

* **Chiến lược của Robot (Nguyên lý):** Robot Beam Search là một robot hoạt động dưới áp lực, với khả năng ghi nhớ hạn chế. Ở mỗi bước, nó chỉ có thể để mắt đến một vài lựa chọn tốt nhất và phải bỏ qua tất cả các khả năng khác.
* **Mục tiêu trong Hệ thống:** Tìm ra một con đường "khá tốt" một cách nhanh chóng và sử dụng rất ít bộ nhớ, chấp nhận rủi ro có thể bỏ lỡ con đường tốt nhất.



**Mô tả hoạt động trong Hệ thống:** Robot Beam Search bắt đầu với một danh sách các lựa chọn (frontier) chỉ có điểm start. Trong mỗi chu trình, nó xem xét tất cả các bước đi tiếp theo có thể từ các lựa chọn hiện tại và gom chúng vào một danh sách lớn candidates. Sau đó, nó sắp xếp danh sách này dựa trên mức độ hứa hẹn (khoảng cách đến khách hàng). Đây là bước quyết định: robot sẽ "cắt tỉa" danh sách này, chỉ giữ lại một số lượng nhỏ các lựa chọn tốt nhất (bằng beam\_width, ví dụ 6 lựa chọn) và vứt bỏ tất cả các lựa chọn còn lại. frontier mới này sẽ được dùng cho chu trình tiếp theo. Vì robot có thể đã vứt bỏ một lựa chọn dẫn đến con đường tối ưu, nó không đảm bảo tìm ra kết quả tốt nhất, nhưng bù lại, nó cực kỳ nhanh và không bao giờ bị quá tải bộ nhớ.

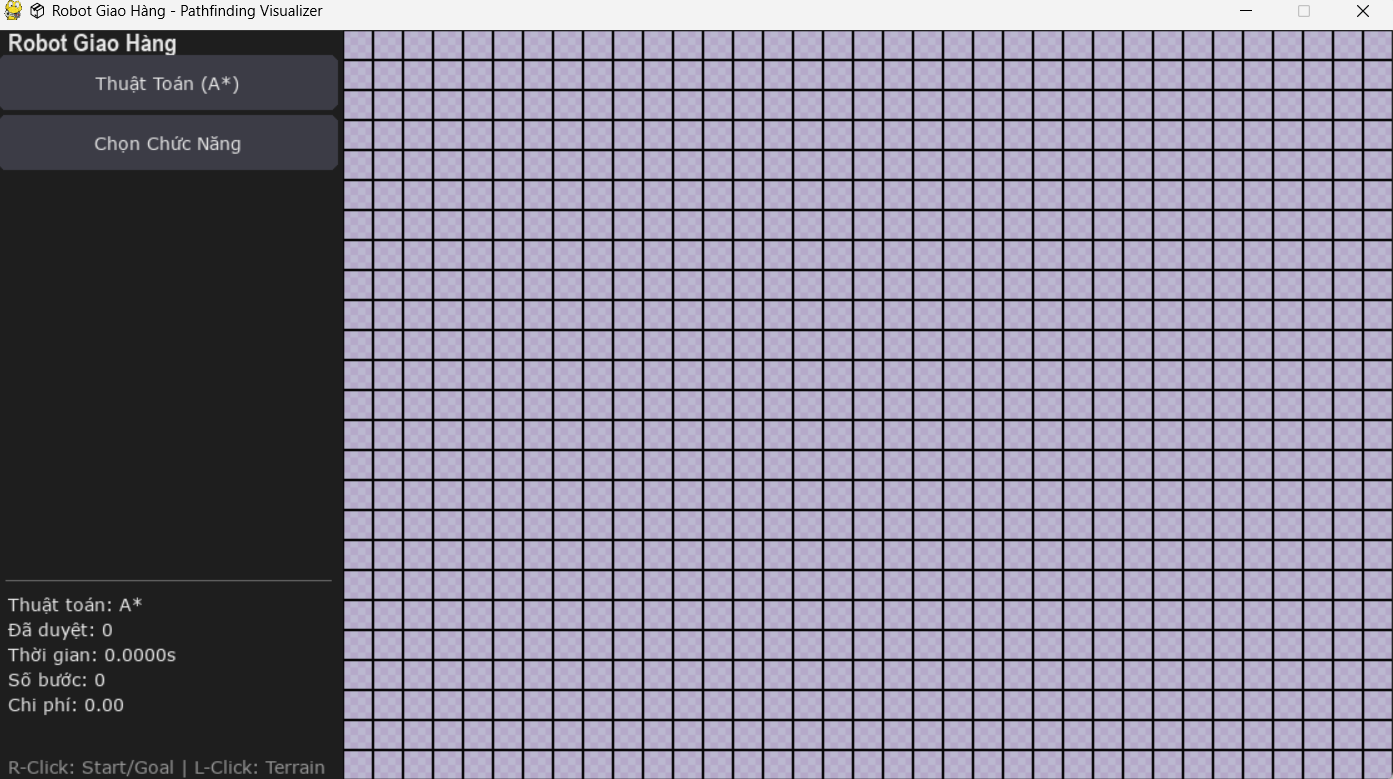
##### **2.2.7. Phân tích Thuật toán Tìm kiếm Sâu dần (IDS)**

* **Chiến lược của Robot (Nguyên lý):** Robot IDS là một robot rất kiên trì và có phương pháp. Nó kết hợp sự hệ thống của Robot BFS và sự tiết kiệm bộ nhớ của Robot DFS. Nó tự hỏi: "Mình có thể đến chỗ khách hàng trong 1 bước không?". Nếu không, nó sẽ quên hết và bắt đầu lại: "Mình có thể đến trong 2 bước không?".
* **Mục tiêu trong Hệ thống:** Tìm ra con đường có **số bước đi ít nhất** (giống BFS) mà không cần phải lưu trữ một hàng đợi khổng lồ.

**Mô tả hoạt động trong Hệ thống:** Robot IDS thực hiện một loạt các cuộc tìm kiếm giới hạn. Vòng lặp for depth\_limit điều khiển giới hạn số bước. Bên trong mỗi lần lặp, nó thực hiện một cuộc tìm kiếm giống hệt DFS bằng cách sử dụng stack, nhưng có thêm một điều kiện if current\_depth < depth\_limit để đảm bảo nó không đi quá giới hạn cho phép. Nếu không tìm thấy khách hàng trong giới hạn hiện tại, nó sẽ tăng giới hạn lên 1 và bắt đầu lại từ đầu. Mặc dù việc tìm kiếm lại từ đầu có vẻ lãng phí, nhưng nó đảm bảo rằng khi robot tìm thấy khách hàng, đó chắc chắn là con đường có số bước đi ít nhất có thể. Phiên bản được tối ưu trong chương trình còn có một danh sách visited\_for\_animation để đảm bảo hiệu ứng duyệt ô trên màn hình vẫn diễn ra mượt mà, không bị lặp lại.

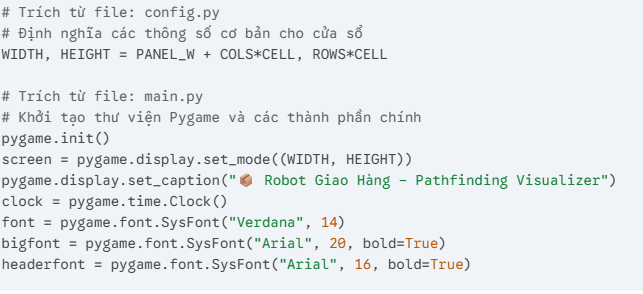
### **3. Xây dựng giao diện và chức năng:**

#### **3.1. Giao diện chính của chương trình**

****

##### **3.1.1.Bố cục và khởi tạo chương trình**

* **Chức năng:** Tạo ra cửa sổ chính của chương trình và các đối tượng quản lý cơ bản và font chữ.

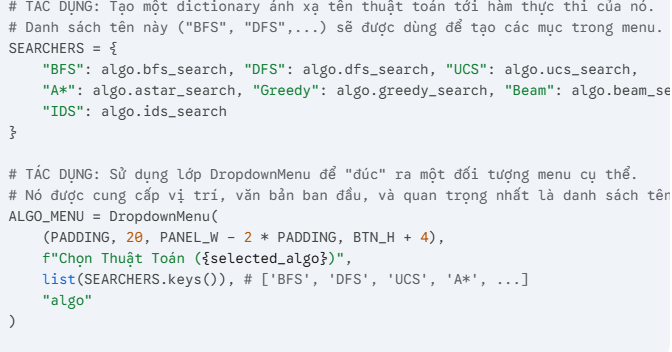


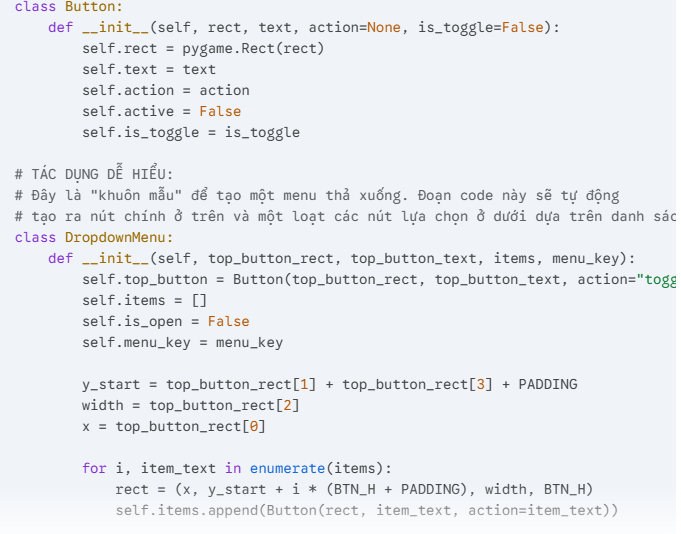
##### **3.1.2. Phân tích Bảng Điều khiển (Control Panel)**

**Phân tích Chức năng: Menu Lựa chọn Thuật toán:**Đây là thành phần tương tác cho phép chọn thuật toán sử dụng để robot tìm đường. Chức năng này được tạo nên từ sự phối hợp của các đoạn mã trong cả 3 file: main.py, ui\_components.py, và config.py

**1. Khởi tạo và Xây dựng Menu (Initialization and Construction)**

**Tác dụng:** Các đoạn code này **tạo ra một đối tượng menu hoàn chỉnh** trong bộ nhớ khi chương trình khởi động. Nó định nghĩa các lựa chọn có trong menu và cấu trúc của menu.

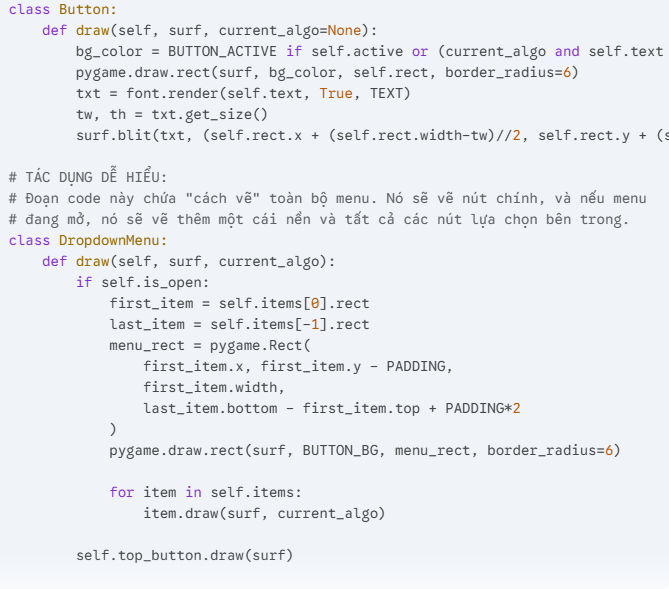




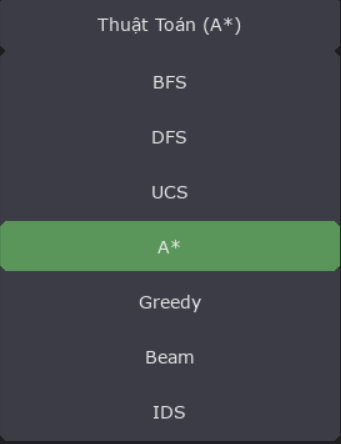
**2. Hiển thị Menu trên Giao diện (Displaying the Menu)**

**Tác dụng:** Các đoạn code này **vẽ menu lên màn hình** trong mỗi khung hình, phản ánh đúng trạng thái hiện tại của nó (đang đóng hay đang mở, và thuật toán nào đang được chọn).



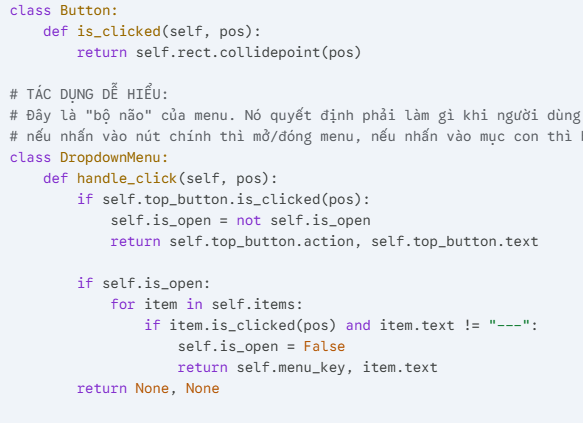
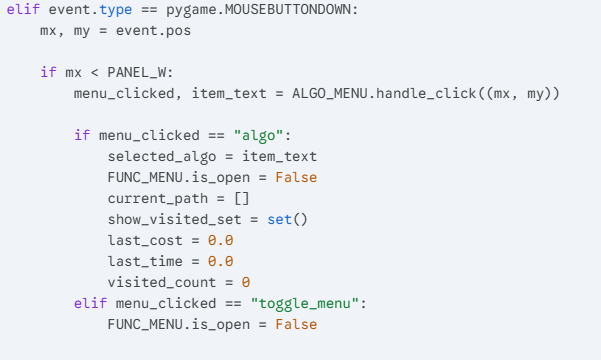


***Hình ảnh mô tả***

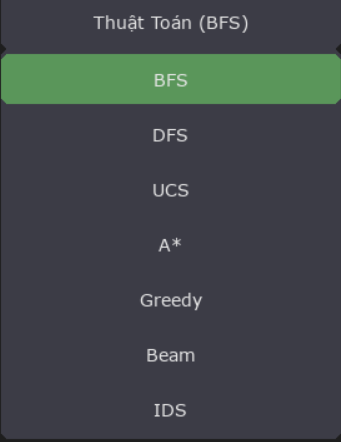
****

**3. Xử lý Tương tác của Người dùng (Handling User Interaction)**

**Tác dụng:** Các đoạn code này **phát hiện và xử lý hành động nhấp chuột** của người dùng. Nó xác định xem người dùng nhấp vào thuật toán nào, sau đó cập nhật trạng thái của chương trình cho phù hợp.



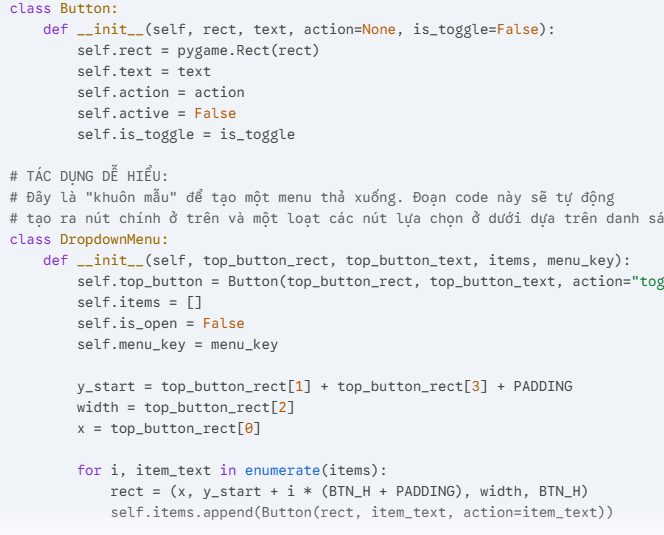
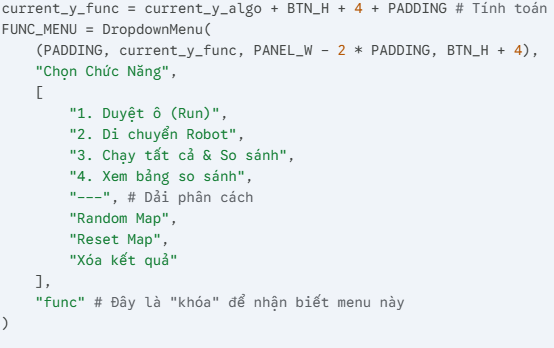
***Hình ảnh minh họa khi nhấp chọn thuật toán BFS***



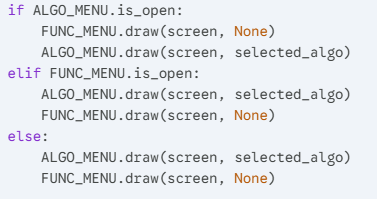
**Phân tích Chức năng: Menu Lựa chọn Chức năng**

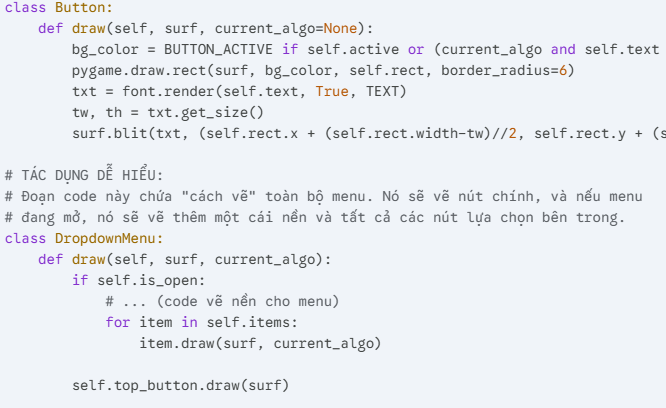
**1. Khởi tạo và Xây dựng Menu (Initialization and Construction)**

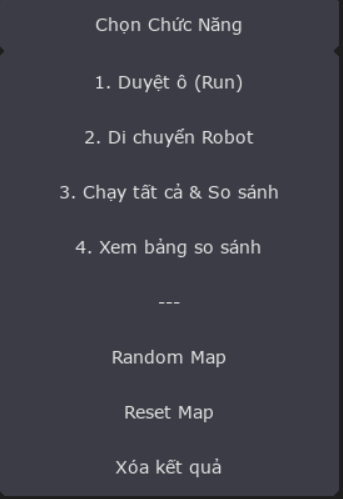
**Tác dụng:** Các đoạn code này **tạo ra một đối tượng menu hoàn chỉnh** trong bộ nhớ khi chương trình khởi động, định nghĩa tất cả các hành động mà người dùng có thể thực hiện.

**2. Hiển thị Menu trên Giao diện (Displaying the Menu)**

**Tác dụng:** Các đoạn code này **vẽ menu lên màn hình** trong mỗi khung hình. Logic vẽ này đảm bảo rằng menu đang mở sẽ luôn được hiển thị ở lớp trên cùng.

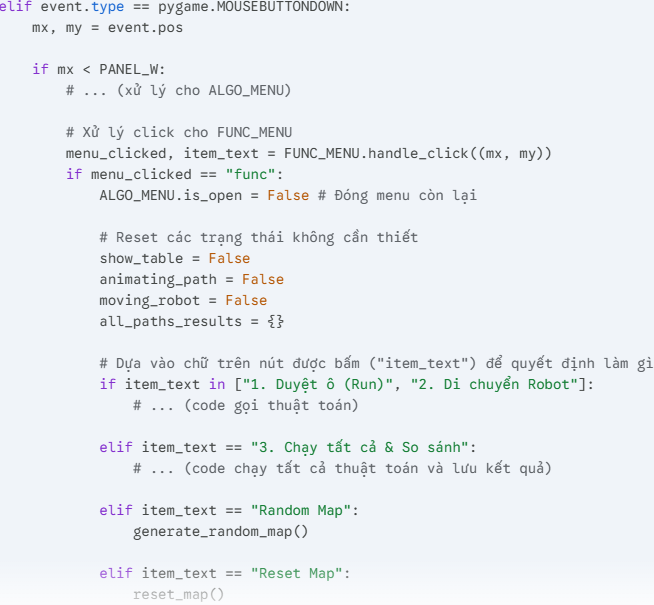


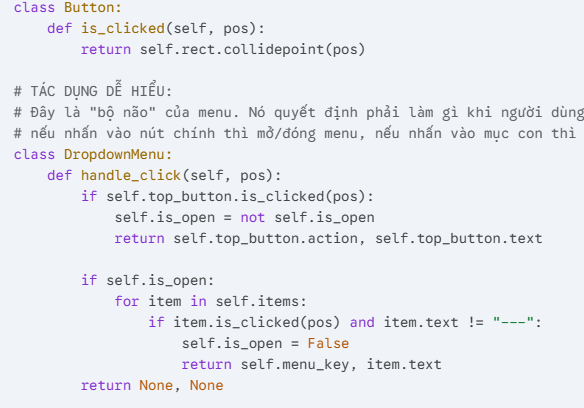




**3. Xử lý Tương tác của Người dùng (Handling User Interaction)**

**Tác dụng:** Các đoạn code này có tác dụng **phát hiện và xử lý hành động nhấp chuột** của người dùng. Nó xác định xem người dùng nhấp vào đâu, sau đó thực thi hành động tương ứng (chạy thuật toán, tạo bản đồ, v.v.).

****

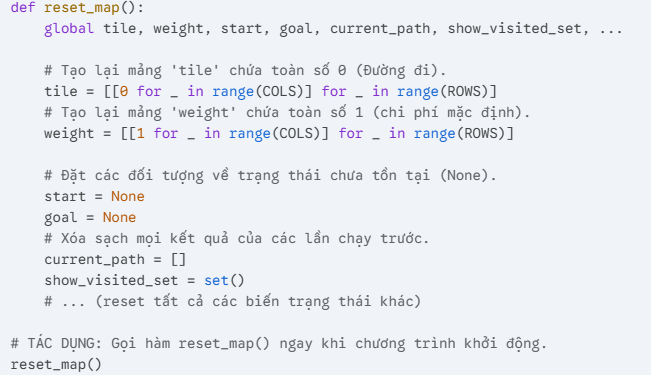


#### **3.2. Giao diện mô phỏng bản đồ và chức năng:**

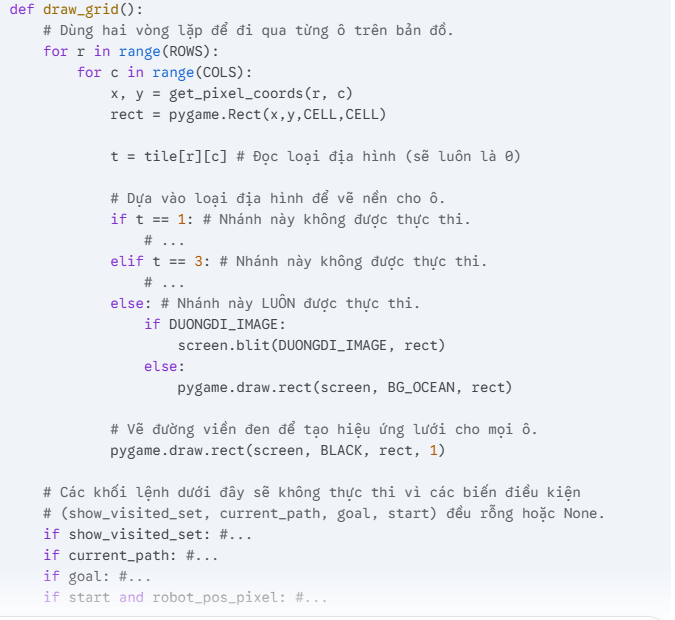
##### **3.2.1 Giao diện Ban đầu và sau khi Reset Map**

Đây là trạng thái của khu vực mô phỏng ngay khi người dùng khởi động chương trình hoặc sau khi chọn chức năng "Reset Map": một lưới ô vuông trống, sẵn sàng để tương tác.

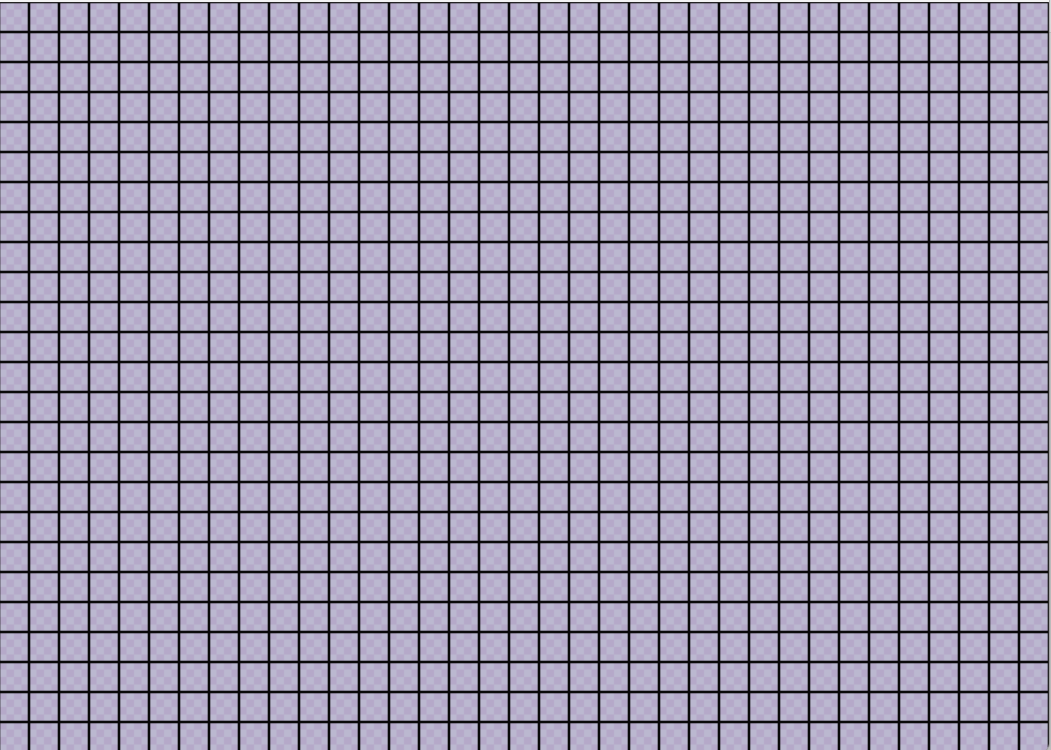
**a. Khởi tạo Trạng thái Dữ liệu (Tạo dữ liệu cho bản đồ trống)**

**Giải thích hoạt động:** Khi chương trình khởi động, hàm reset\_map() được gọi. Hàm này thực hiện việc "dọn dẹp" bằng cách tạo mới hoàn toàn các mảng dữ liệu. Nó tạo ra một mảng hai chiều tile chứa đầy các số 0 (mỗi số 0 đại diện cho một ô "Đường đi") và một mảng weight chứa đầy số 1. Đồng thời, các biến quan trọng như start, goal, và current\_path đều được đặt về trạng thái rỗng hoặc None. Kết quả là trong bộ nhớ máy tính đã có một "bản đồ dữ liệu" trống hoàn toàn.

**b. Logic Vẽ Lưới (Đọc dữ liệu và vẽ hình ảnh)**

****

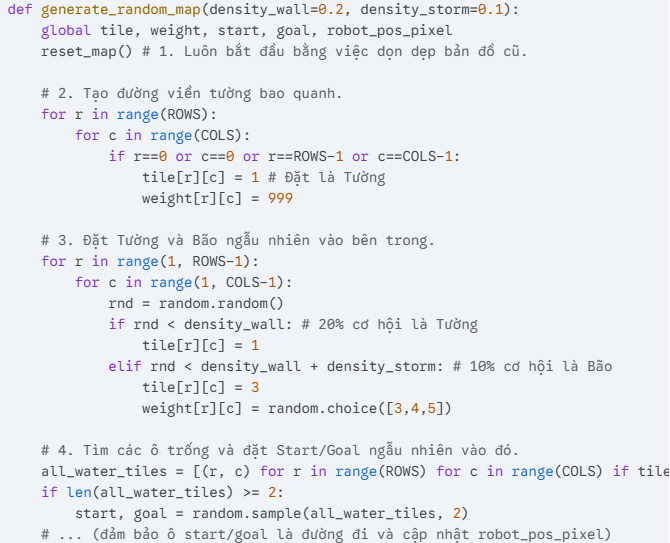
**Giải thích hoạt động:** Hàm draw\_grid() được gọi 60 lần mỗi giây. Nó duyệt qua từng ô của "bản đồ dữ liệu". Vì reset\_map() đã đảm bảo mảng tile chỉ chứa số 0, nên khi draw\_grid() đọc giá trị t = tile[r][c], t sẽ luôn là 0. Do đó, khối lệnh if/elif/else sẽ luôn thực thi nhánh else, vẽ hình ảnh/màu sắc của "Đường đi" cho mọi ô. Các khối lệnh vẽ Tường, Bão, Robot, Khách hàng, hay kết quả thuật toán đều bị bỏ qua vì dữ liệu tương ứng của chúng chưa tồn tại. Cuối cùng, một đường viền đen được vẽ cho mỗi ô, tạo nên hình ảnh một lưới trống hoàn chỉnh



##### **3.2.2 Giao diện sau khi Chọn chức năng Random Map**

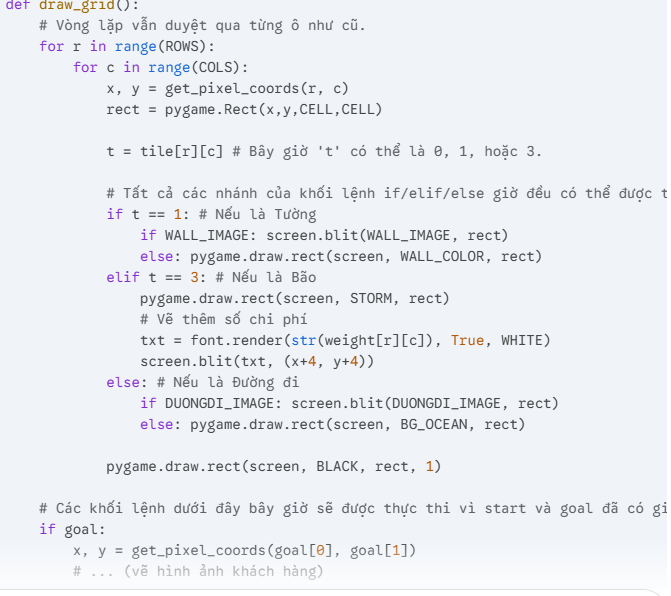
**Thay đổi hoàn toàn dữ liệu của bản đồ** và sau đó **vẽ lại hình ảnh mới** dựa trên dữ liệu đã thay đổi đó. Đầu tiên, hàm generate\_random\_map sẽ điền các giá trị ngẫu nhiên (Tường, Bão, Start, Goal) vào các mảng dữ liệu. Ngay sau đó, hàm draw\_grid (vẫn là hàm vẽ cũ) sẽ đọc dữ liệu mới này. Vì dữ liệu đã khác, hàm draw\_grid giờ đây sẽ vẽ ra một hình ảnh hoàn toàn khác, có cả Tường, Bão, Robot và Khách hàng.

* 1. **Logic Thay đổi Dữ liệu (Tạo dữ liệu cho bản đồ ngẫu nhiên)**

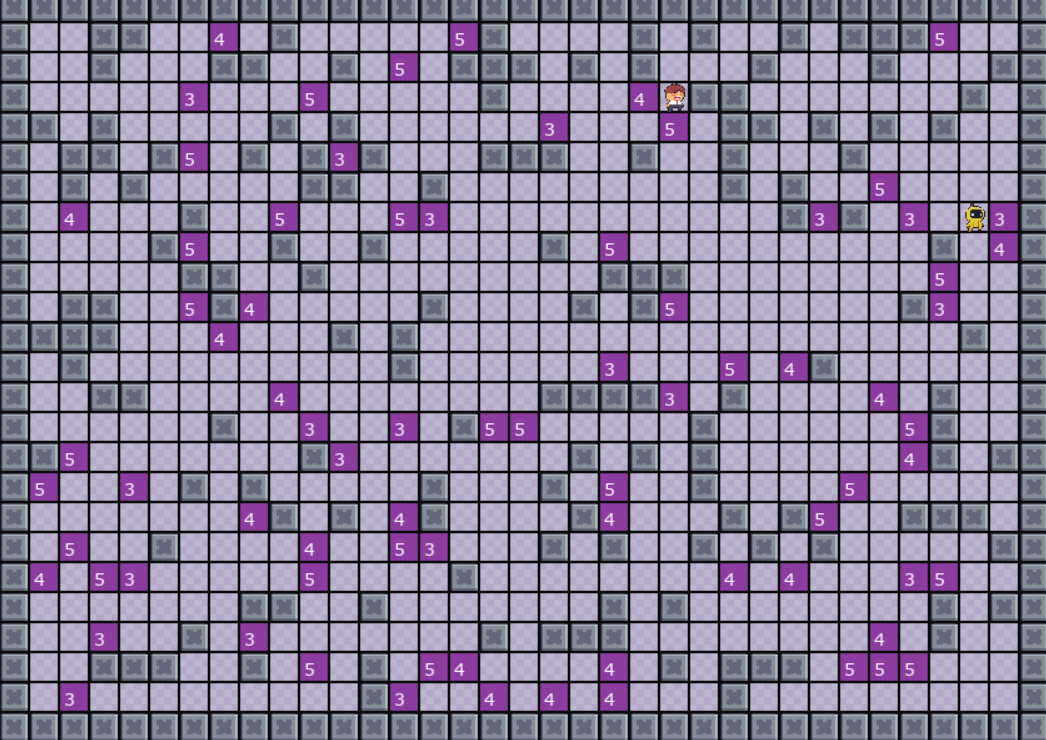
****

**Giải thích hoạt động:** Khi được gọi, hàm này trước tiên sẽ xóa sạch bản đồ bằng reset\_map(). Sau đó, nó thực hiện một loạt các thay đổi trên "bản đồ dữ liệu": nó điền số 1 (Tường) vào các cạnh của mảng tile; lấp đầy phần bên trong bằng các số 1 (Tường) và 3 (Bão) một cách ngẫu nhiên; cuối cùng, nó tìm tất cả các ô còn lại có giá trị 0 (Đường đi), chọn ngẫu nhiên hai ô trong số đó và gán tọa độ của chúng cho các biến start và goal. Kết quả là các mảng dữ liệu tile, weight và các biến start, goal giờ đây chứa đầy thông tin của một bản đồ phức tạp.

* 1. **Logic Vẽ Lưới (Đọc dữ liệu MỚI và vẽ hình ảnh)**

****

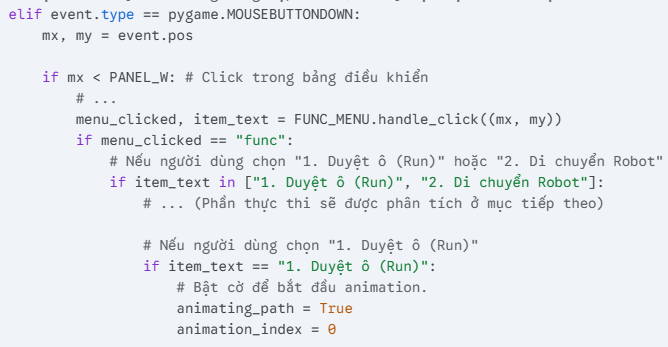
**Giải thích hoạt động:** Ở khung hình ngay sau khi generate\_random\_map() chạy xong, hàm draw\_grid() được gọi như bình thường. Tuy nhiên, lần này khi nó đọc giá trị t = tile[r][c], t không còn luôn là 0 nữa. Nó có thể là 1 hoặc 3. Do đó, các nhánh if t == 1: và elif t == 3: giờ đây sẽ được thực thi, khiến cho các hình ảnh/màu sắc của Tường và Bão được vẽ ra. Tương tự, vì các biến start và goal không còn là None, các khối lệnh if goal: và if start: ở cuối hàm cũng sẽ được thực thi, vẽ hình ảnh của robot và khách hàng lên màn hình. Kết quả cuối cùng là một bản đồ ngẫu nhiên, phức tạp được hiển thị đầy đủ.



### **4.Trực quan hóa thuật toán**

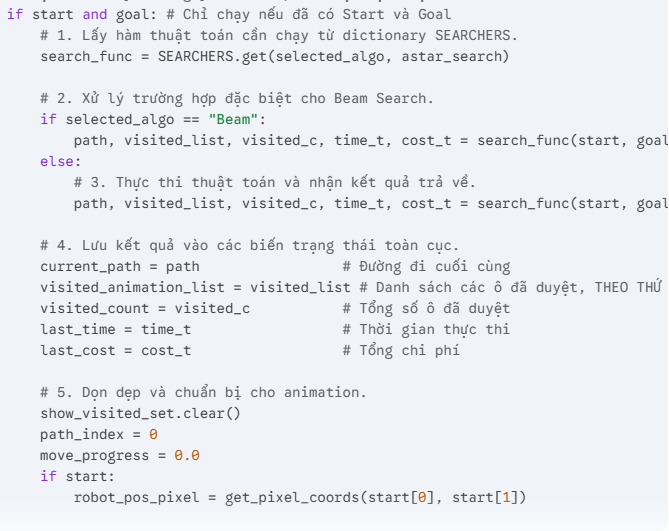
#### **4.1. Duyệt ô và chạy thuật toán:**

**Giai đoạn 1: Kích hoạt Chức năng**

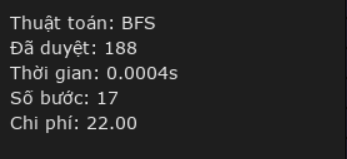
****

**Giải thích hoạt động:** Khi người dùng nhấp vào "1. Duyệt ô (Run)", main.py sẽ nhận diện item\_text tương ứng. Sau khi gọi thuật toán, nó sẽ đặt biến cờ animating\_path thành True và reset animation\_index về 0. Biến animating\_path chính là "công tắc" để khởi động quá trình vẽ hiệu ứng duyệt ô ở các khung hình tiếp theo.

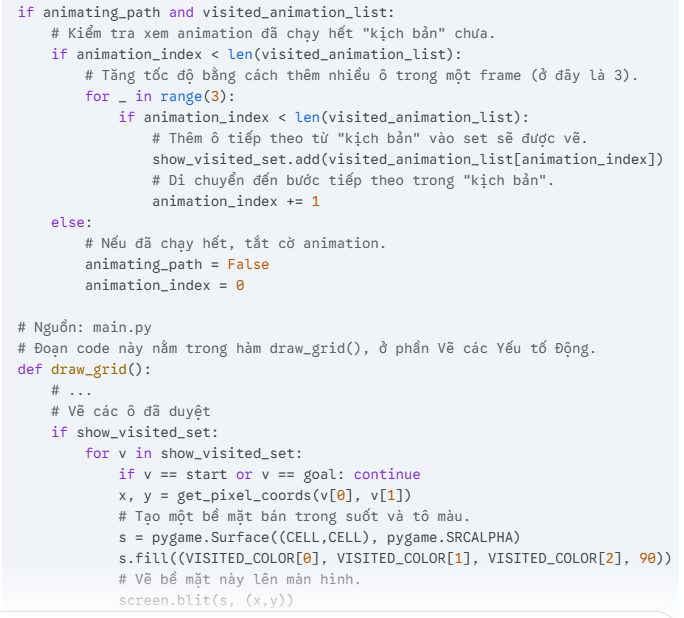
**Giai đoạn 2: Thực thi Thuật toán và Lưu trữ Kết quả**

Đoạn code này có tác dụng **gọi hàm thuật toán tương ứng**, nhận về kết quả, và **lưu trữ các kết quả này** vào các biến trạng thái toàn cục để chuẩn bị cho việc hiển thị và animation. **Giải thích hoạt động:**

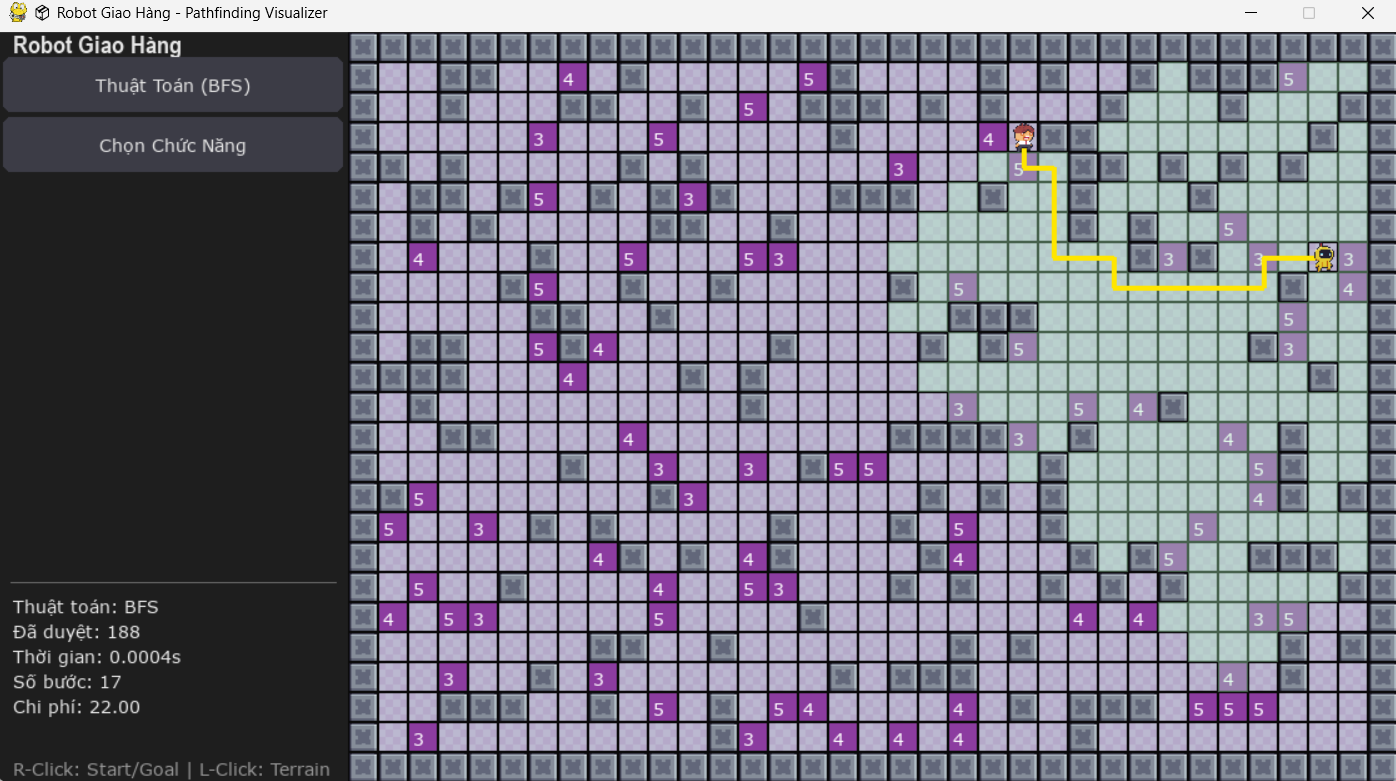
1. Chương trình dùng SEARCHERS.get(selected\_algo, ...) để lấy ra đúng hàm thuật toán mà người dùng đã chọn (ví dụ: bfs\_search).
2. Nó gọi hàm thuật toán này, truyền vào start và goal.
3. Hàm thuật toán sẽ chạy và trả về một tuple chứa 5 giá trị: đường đi, danh sách các ô đã duyệt theo thứ tự, tổng số ô đã duyệt, thời gian, và chi phí.
4. main.py sẽ "giải nén" tuple này và lưu từng giá trị vào các biến tương ứng. Quan trọng nhất là visited\_animation\_list, đây là "kịch bản" cho hiệu ứng animation sẽ diễn ra.
5. Cuối cùng, nó dọn dẹp show\_visited\_set (set dùng để vẽ) để chuẩn bị cho animation mới.



**Giai đoạn 3: Hiển thị Animation Duyệt ô**

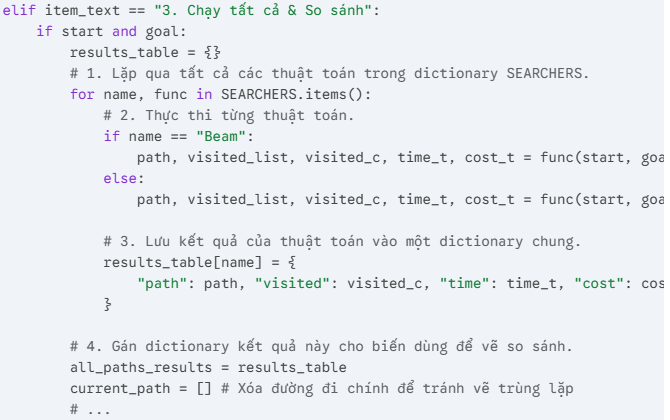
**Giải thích hoạt động:**

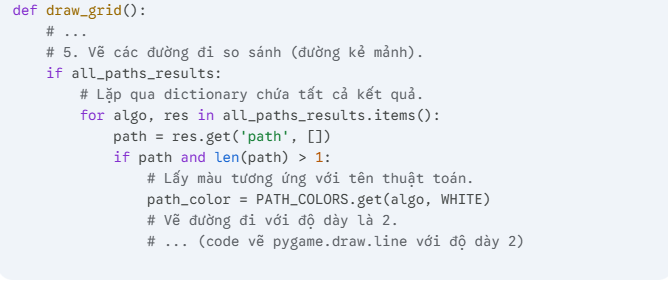
1. **Cập nhật Trạng thái:** Trong mỗi khung hình, nếu animating\_path là True, chương trình sẽ lấy 3 ô tiếp theo từ danh sách visited\_animation\_list (kịch bản) và thêm chúng vào show\_visited\_set. animation\_index được tăng lên để ghi nhớ vị trí hiện tại trong kịch bản.
2. **Vẽ Giao diện:** Ngay sau đó, hàm draw\_grid() được gọi. Nó sẽ duyệt qua tất cả các ô có trong show\_visited\_set và vẽ một lớp màu xanh nhạt, bán trong suốt lên chúng.
3. **Tạo hiệu ứng:** Vì quá trình (1) và (2) được lặp lại 60 lần mỗi giây, người dùng sẽ thấy các ô màu xanh nhạt xuất hiện tuần tự trên màn hình, tạo thành một hiệu ứng lan tỏa mượt mà, trực quan hóa chính xác thứ tự mà thuật toán đã khám phá các ô.



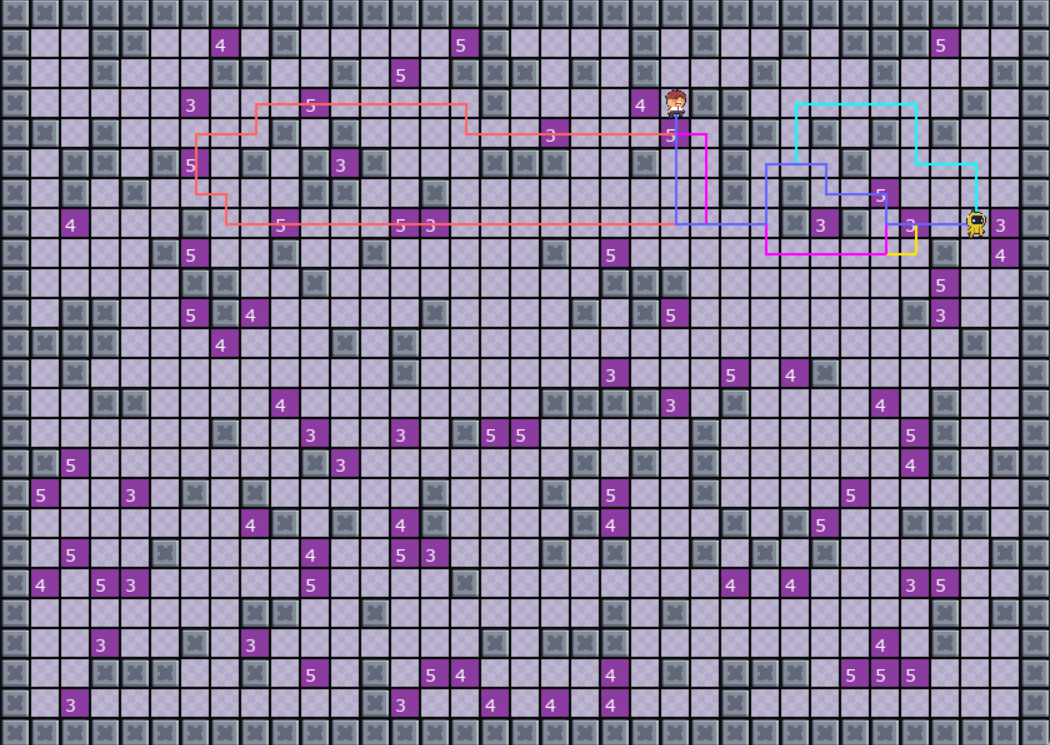
#### **4.2. Chức năng Chạy tất cả và So sánh**

Lặp **qua tất cả các thuật toán đã biết**, thực thi từng cái một, **lưu trữ tất cả các kết quả** vào một cấu trúc dữ liệu chung, và sau đó **vẽ tất cả các đường đi** này lên màn hình với màu sắc khác nhau.



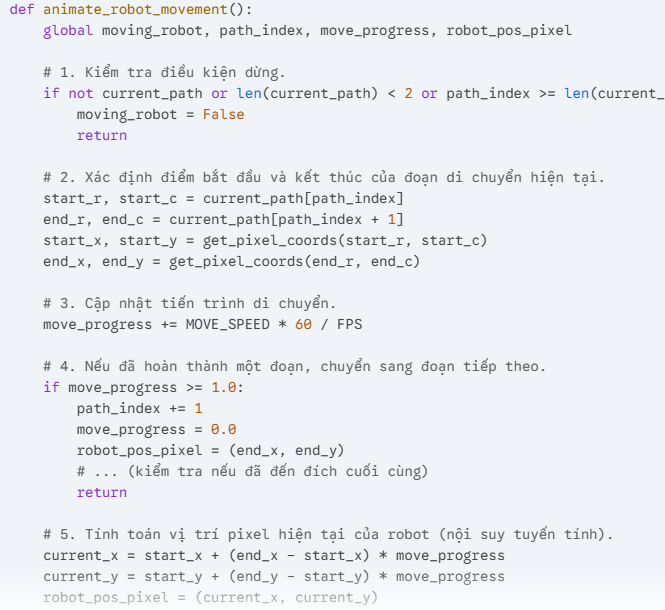
**Giải thích hoạt động:**

1. Khi người dùng chọn chức năng, chương trình lặp qua SEARCHERS.items(), lấy ra cả tên (name) và hàm (func) của từng thuật toán.
2. Nó thực thi mỗi hàm func.
3. Kết quả của mỗi thuật toán được lưu vào results\_table với name làm khóa.
4. Sau khi tất cả các thuật toán đã chạy xong, biến all\_paths\_results sẽ chứa kết quả của tất cả chúng.
5. Trong hàm draw\_grid(), đoạn code tương ứng sẽ kiểm tra if all\_paths\_results:. Nếu biến này có dữ liệu, nó sẽ lặp qua từng kết quả, lấy ra đường đi (path) và tên thuật toán (algo), sau đó vẽ mỗi đường đi với màu sắc tương ứng và độ dày mỏng hơn.



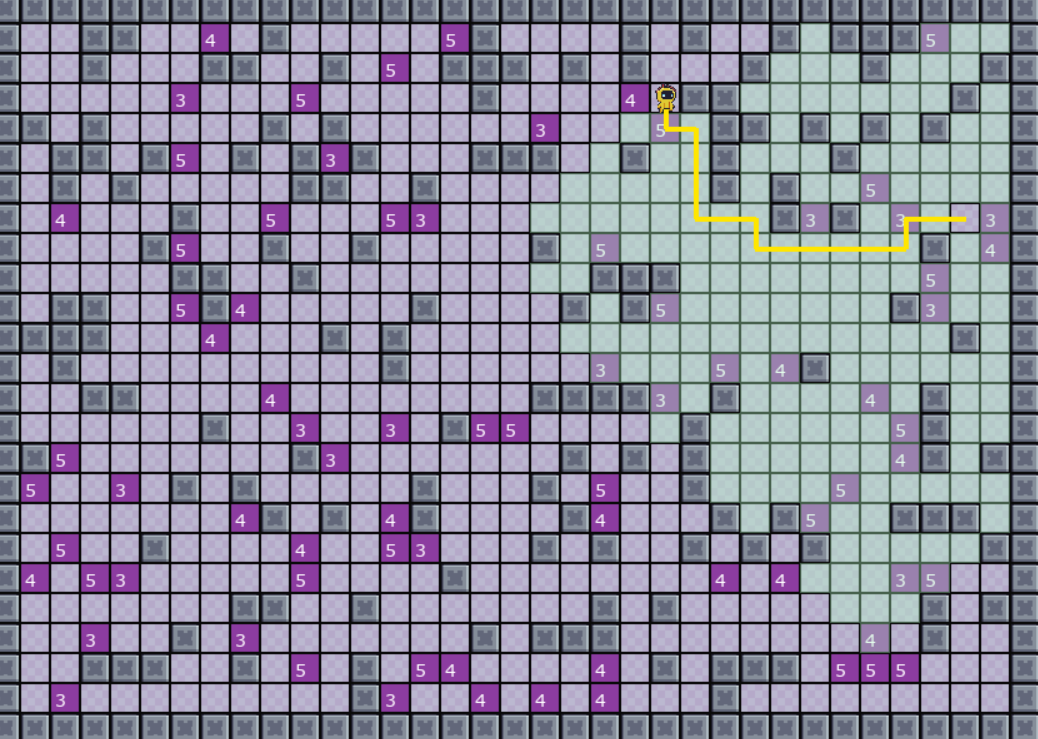
#### **4.3. Chức năng di chuyển Robot**

**Tính toán và cập nhật vị trí pixel của robot** trong mỗi khung hình để tạo ra di chuyển mượt mà từ ô này sang ô khác, thay vì chỉ "nhảy" tức thời.



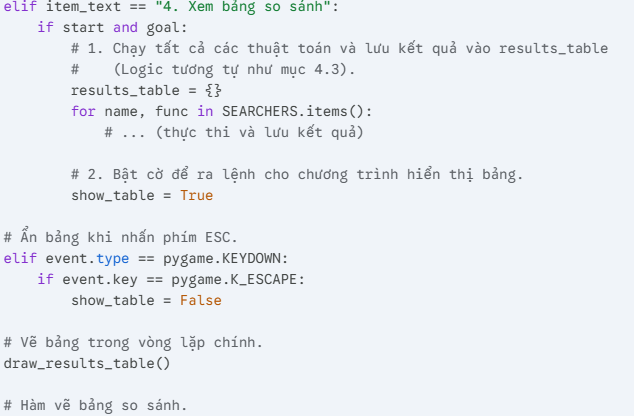
**Giải thích hoạt động:**

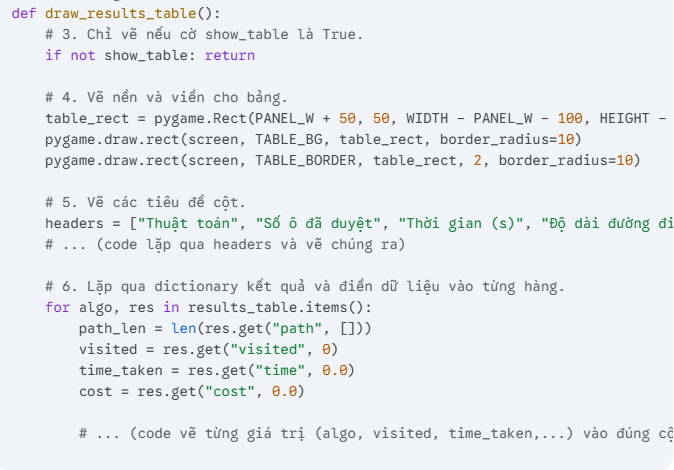
* **Kích hoạt:** Khi người dùng chọn chức năng, cờ moving\_robot được bật.
* **Logic Nội suy:** Hàm animate\_robot\_movement được gọi liên tục. Nó xác định hai ô liên tiếp trong đường đi (path\_index và path\_index + 1).
* Biến move\_progress (từ 0.0 đến 1.0) đại diện cho tiến trình di chuyển giữa hai ô này. Nó được tăng lên một chút trong mỗi khung hình.
* Công thức current\_x = start\_x + (end\_x - start\_x) \* move\_progress là phép nội suy tuyến tính, dùng để tính toán vị trí pixel chính xác của robot tại một điểm nằm giữa start\_x và end\_x, dựa trên move\_progress.
* Khi move\_progress đạt 1.0, điều đó có nghĩa là robot đã đi hết một đoạn. path\_index sẽ được tăng lên để bắt đầu di chuyển sang đoạn tiếp theo.
* **Hiển thị:** Hàm draw\_grid sẽ luôn đọc giá trị mới nhất của robot\_pos\_pixel để vẽ robot. Vì robot\_pos\_pixel thay đổi một chút trong mỗi khung hình, người dùng sẽ thấy robot di chuyển một cách mượt mà.



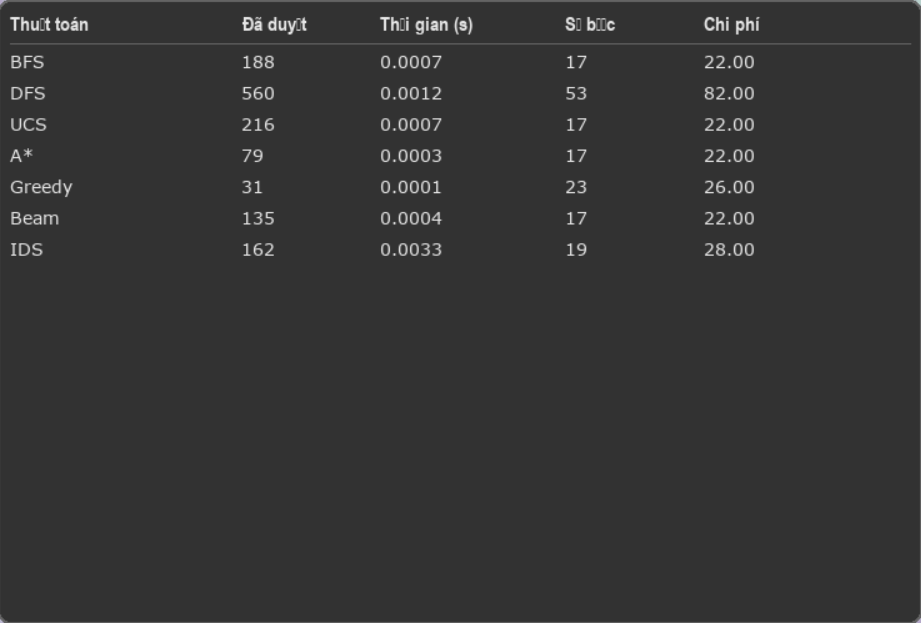
#### **4.4. Bảng so sánh**

**Chạy tất cả các thuật toán** (nếu chưa có kết quả), **bật cờ hiển thị bảng**, và sau đó **vẽ một bảng thống kê** lên màn hình, điền các dữ liệu về hiệu suất của mỗi thuật toán vào bảng đó.



**Giải thích hoạt động:**

1. Khi người dùng chọn chức năng, chương trình sẽ chạy tất cả các thuật toán và lưu kết quả vào results\_table (tương tự chức năng 4.3).
2. Quan trọng nhất, nó đặt show\_table = True.
3. Hàm draw\_results\_table() được gọi trong mỗi khung hình. Nó sẽ kiểm tra cờ này.
4. Nếu show\_table là True, nó sẽ vẽ nền và viền cho bảng.
5. Sau đó, nó vẽ các tiêu đề cột ("Thuật toán", "Số ô đã duyệt", v.v.).
6. Cuối cùng, nó lặp qua results\_table. Với mỗi thuật toán, nó lấy ra các giá trị hiệu suất (visited, time, cost...) và vẽ chúng thành một hàng trong bảng.
7. Nếu người dùng nhấn ESC, cờ show\_table sẽ được đặt lại thành False, và ở khung hình tiếp theo, hàm draw\_results\_table sẽ không vẽ gì cả, làm cho bảng biến mất.



# **PHẦN KẾT LUẬN**

Sau quá trình xây dựng và phát triển, chương trình **"Robot Giao Hàng - Pathfinding Visualizer"** đã được hoàn thiện. Có thể nói, đây không chỉ là một bài tập lập trình, mà đã trở thành một công cụ hoàn chỉnh và thực sự hữu ích. Mục tiêu ban đầu của báo cáo là tạo ra một môi trường trực quan để tìm hiểu và so sánh các thuật toán tìm đường, và sản phẩm cuối cùng đã đáp ứng trọn vẹn, thậm chí vượt qua mong đợi đó.

Điểm giá trị nhất mà chương trình này mang lại chính là khả năng biến nó thành một **phòng thí nghiệm ảo**. Thay vì phải đọc những lý thuyết khô khan, người dùng có thể "thả" các thuật toán vào một môi trường đầy thử thách và quan sát cách chúng "suy nghĩ". Bạn có thể tận mắt chứng kiến cách BFS "ngây thơ" lan toả khắp bản đồ, trong khi A\* "thông minh" tập trung đi thẳng đến mục tiêu. Đặc biệt, thông qua chức năng so sánh, sự khác biệt giữa các chiến lược tìm đường trở nên cực kỳ rõ nét. Khi đặt ra một bài toán có các vùng "Bão", ta sẽ thấy ngay sự đối lập: một bên là các thuật toán chỉ quan tâm đến "đường đi ngắn nhất" (ít ô nhất), và một bên là các thuật toán khôn ngoan hơn, tìm ra "đường đi tối ưu nhất" (tốn ít nhiên liệu nhất). Bảng so sánh với những con số cụ thể về thời gian, chi phí, số ô đã duyệt càng làm cho những quan sát trực quan này trở nên thuyết phục.

Tất nhiên, không có sản phẩm nào là hoàn hảo, và chương trình này cũng vậy. Cách tổ chức code khoa học ngay từ đầu đã mở ra rất nhiều hướng phát triển hay ho trong tương lai nếu có thêm thời gian. Chúng ta hoàn toàn có thể bổ sung các thuật toán "xịn" hơn như Jump Point Search để tăng tốc độ tìm kiếm, hay D\* Lite để robot có thể tự tìm đường mới khi có vật cản đột ngột xuất hiện. Về phần tương tác, việc cho phép người dùng lưu lại những bản đồ thú vị, thêm nhiều loại địa hình mới, hay thậm chí là mô phỏng một con robot có kích thước lớn hơn sẽ làm cho các bài toán trở nên thực tế và hấp dẫn hơn. Một chế độ chạy "từng bước" (step-by-step) cũng sẽ là một công cụ tuyệt vời cho việc giảng dạy và học tập.

Nhìn chung, báo cáo này đã trình bày quá trình xây dựng thành công một công cụ mạnh mẽ, không chỉ giải quyết được bài toán tìm đường mà còn mang lại giá trị cao trong việc giáo dục và phân tích thuật toán. Nó là một nền tảng vững chắc, một sản phẩm hoàn thiện và sẵn sàng cho những cải tiến, nâng cấp thú vị trong tương lai.

# **TÀI LIỆU THAM KHẢO**

[1] [Milos Simic](https://www.baeldung.com/cs/author/milossimic) (2024). “The Informed vs. Uninformed Search Algorithms”. Truy cập ngày 13/10/2025 từ <https://www.baeldung.com/cs/informed-vs-uninformed-search>

[2] Nguyễn Châu Khanh (2023). “Thuật toán duyệt đồ thị ưu tiên chiều rộng”. Truy cập ngày 13/10/2025 từ <https://wiki.vnoi.info/algo/graph-theory/breadth-first-search.md>

[3] Nguyễn Châu Khanh et al. (2024). “Cây DFS (Depth-First Search Tree) và ứng dụng”. Truy cập ngày 13/10/2025 từ <https://wiki.vnoi.info/algo/graph-theory/Depth-First-Search-Tree.md>

[4] Thơ Trần (2019). “Các thuật toán cơ bản trong AI - Phân biệt Best First Search và Uniform Cost Search (UCS)”. Truy cập ngày 13/10/2025 từ <https://viblo.asia/p/cac-thuat-toan-co-ban-trong-ai-phan-biet-best-first-search-va-uniform-cost-search-ucs-Eb85omLWZ2G>