TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC
TRÍ TUỆ NHÂN TẠO

ỨNG DỤNG MỘT SỐ THUẬT TOÁN ĐỂ MÔ PHỎNG ROBOT TÌM ĐƯỜNG TRONG MÊ CUNG

HỌC KỲ 2 – NĂM HỌC: 2024 - 2025

MÃ LỚP HỌC PHẦN: ARIN330585_06

Giảng viên hướng dẫn: Ts. Bùi Mạnh Quân

Nhóm sinh viên thực hiện:	Nguyễn Quân Bảo	MSSV: 23110181
	Nguyễn Đăng Tường	MSSV: 23110360
	Dương Minh Tâm	MSSV: 23110311

TP. HCM, tháng 05 năm 2025

BÌA LÓT

(Nội dung bìa lót giống như bìa chính)

DANH SÁCH NHÓM

HỌC KỲ II, NĂM HỌC 2024-2025

Tên đề tài: ỨNG DỤNG MỘT SỐ THUẬT TOÁN MÔ PHỎNG ROBOT TÌM ĐƯỜNG TRONG MÊ CUNG

STT	HỌ VÀ TÊN	Mã số sinh viên	Tỉ lệ % hoàn thành
01	Nguyễn Quân Bảo	23110181	100%
02	Nguyễn Đăng Tường	23110360	100%
03	Dương Minh Tâm	23110311	100%

Ghi chú:

- Tỷ lệ % = XX%: mức độ phần trăm của từng học sinh tham gia được đánh giá bởi nhóm trường và thống nhất giữa các thành viên trong nhóm

- Trưởng nhóm: Nguyễn Quân Bảo

NHẬN XÉT VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT

Tên viết tắt	Ý nghĩa đầy đủ
BFS	Breadth-First Search
DFS	Depth-First Search
A*	A Star Search
GUI	Graphical User Interface
UI	User Interface
FPS	Frames Per Second
CPU	Central Processing Unit
IDE	Integrated Development Environment
Tkinter	(Không viết tắt – tên thư viện Python)
PIL	Python Imaging Library
AI	Artificial Intelligence

LỜI NÓI ĐẦU

Nhóm em xin bày tỏ lời cảm ơn chân thành tới các thầy giáo trường Đại học Sư Phạm Kĩ Thuật đã dạy dỗ chúng em trong suốt quá trình học tập chương trình cao học tại trường.

Đặc biệt nhóm em xin bày tỏ lòng biết ơn sâu sắc tới thầy TS. Bùi Mạnh Quân, Trường Đại học Sư Phạm Kĩ Thuật đã quan tâm, định hướng và đưa ra những góp ý, gợi ý, chỉnh sửa quý báu cho chúng em trong quá trình làm đồ án học phần. .

Cuối cùng, nhóm em xin cảm ơn quý thầy cô trường Đại học Sư Phạm Kĩ Thuật đã tạo điều kiện thuận lợi, cung cấp môi trường học tập và các tài liệu cần thiết để nhóm em có thể thực hiện đồ án này.

Nhóm em xin chân thành cảm ơn!

CHƯƠNG 1 - GIỚI THIỆU

1.1. Lý do chọn đề tài

Trong thời đại công nghệ 4.0 đang cực kì hiện đại ngày nay thì trí tuệ nhân tạo (AI) ngày càng khẳng định vai trò then chốt trong nhiều lĩnh vực của đời sống và sản xuất. Từ các hệ thống tự động hóa phức tạp trong công nghiệp, đến những ứng dụng thông minh phục vụ đời sống hàng ngày, khả năng tự động tìm kiếm và lập kế hoạch hành động hiệu quả là một yếu tố cốt lõi. Thuật toán tìm kiếm đường đi, một nhánh quan trọng của AI, đóng vai trò như bộ não điều khiển, giúp các hệ thống thông minh này "suy nghĩ" và vạch ra lộ trình tối ưu để đạt được mục tiêu.

Giải mã mê cung là một bài toán kinh điển trong khoa học máy tính, có ứng dụng rộng rãi từ trò chơi, robot tự hành, đến các hệ thống định tuyến. Việc sử dụng các thuật toán tìm đường tiên tiến như A*, Breadth-First Search (BFS), Depth-First Search (DFS) và Dijkstra's algorithm có thể mang lại nhiều lợi ích thiết thực, bao gồm tối ưu hóa đường đi, tiết kiệm thời gian và nâng cao hiệu suất giải quyết vấn đề.

Chính vì lẽ đó, việc nghiên cứu đề tài "Thuật toán tìm kiếm đường đi và ứng dụng trong trò giải mã mê cung" trở nên vô cùng thiết thực và mang tính thời sự. Đề tài này không chỉ cung cấp cái nhìn sâu sắc vào nền tảng lý thuyết của một lĩnh vực quan trọng trong AI, mà còn mở ra cơ hội khám phá những ứng dụng cụ thể và đầy thú vị trong lĩnh vực trò chơi điện tử. Việc giải mã mê cung, một bài toán cổ điển nhưng vẫn đầy thách thức, là một minh chứng sinh động cho sức mạnh và tính linh hoạt của các thuật toán tìm kiếm đường đi, hứa hẹn mang lại những kết quả nghiên cứu giá trị cả về mặt lý thuyết lẫn ứng dụng thực tiễn.

1.2. Mục tiêu đề tài

Mục tiêu chung: nghiên cứu, phân tích các thuật toán tìm kiếm đường đi cơ bản và nâng cao, đồng thời ứng dụng chúng để giải quyết bài toán tìm đường trong môi trường mê cung.

Mục tiêu cụ thể: tìm hiểu và hệ thống hóa cơ sở lý thuyết về bài toán tìm kiếm đường đi, bao gồm các khái niệm về không gian trạng thái, hành động, chi phí, và các thuật toán phổ biến như A*, Breadth-First Search (BFS), Depth-First Search (DFS) và Dijkstra's algorithm.

Nghiên cứu sâu về thuật toán A*, đặc biệt tập trung vào vai trò của hàm heuristic và các phương pháp đo khoảng cách khác nhau (Euclidean, Manhattan, Chebyshev, Octile, Angle Euclidean) trong việc ảnh hưởng đến hiệu suất tìm kiếm. Xây dựng một mô hình môi trường mê cung trên máy tính, cho phép người dùng tương tác và thiết lập các cấu hình mê cung khác nhau (kích thước, vị trí chướng ngại vật, điểm bắt đầu, điểm kết thúc). Triển khai và cài đặt các thuật toán tìm kiếm đường đi đã nghiên cứu (ít nhất A*, và có thể so sánh với Breadth-First Search (BFS), Depth-First Search (DFS) và Dijkstra's algorithm) để giải bài toán tìm đường trong môi trường mê cung đã xây dựng.

1.3. Phạm vi đề tài

Không gian: tập trung vào việc nghiên cứu và ứng dụng các thuật toán tìm kiếm đường đi trong một môi trường cụ thể là mê cung hai chiều tĩnh (static 2D maze)

Thuật toán trọng tâm: tập trung chủ yếu vào thuật toán A* Search và việc ứng dụng các hàm heuristic khác nhau (Euclidean, Manhattan, Chebyshev, Octile, và có thể xem xét Angle Euclidean). Ngoài ra, đề tài có thể so sánh hiệu suất của A* với các thuật toán cơ bản khác như Breadth-First Search (BFS), Depth-First Search (DFS) và Dijkstra's algorithm.

1.4 Đối tượng nghiên cứu

Các đối tượng chính được nghiên cứu trong đề tài

+ Các thuật toán tìm kiếm đường đi : tập trung vào các thuật toán cơ bản và nâng cao được sử dụng để giải quyết bài toán tìm đường trong không gian trạng thái, đặc biệt là các thuật toán như Breadth-First Search (BFS), Depth-First Search (DFS), Dijkstra's Algorithm, Greedy Best-First Search, và trọng tâm là thuật toán A* Search

+ Hàm heuristic: Nghiên cứu sâu về vai trò, đặc điểm và ảnh hưởng của các hàm heuristic khác nhau (Euclidean, Manhattan, Chebyshev, Octile, Angle Euclidean) đến hiệu suất và kết quả tìm kiếm của thuật toán A*

+ Bài toán tối ưu giải mã mê cung: Yếu tố đầu vào (vị trí bắt đầu, chướng ngại vật), đầu ra (tuyến đường, chi phí ước tính)

1.5. Phương pháp nghiên cứu

Thu thập thông tin: khảo sát tải liệu học tập của Giáo Viên: Ts. Mạnh Bùi Quân về các thuật toán A*, Breadth-First Search (BFS), Depth-First Search (DFS) và Dijkstra's algorithm.

Xây dựng mô hình và cài đặt thuật toán: sử dụng ngôn ngữ lập trình Python và các thư viện phù hợp (ví dụ như Matplotlib để trực quan hóa) để xây dựng mô hình môi trường mê cung và cài đặt các thuật toán tìm kiếm đường đi đã nghiên cứu.

Thực nghiệm và thu thập dữ liệu: tiến hành thực nghiệm bằng cách chạy các thuật toán đã cài đặt trên các mẫu mê cung thu thập được. Trong quá trình thực nghiệm, các dữ liệu về hiệu suất của thuật toán sẽ được thu thập một cách có hệ thống.

Phương pháp định lượng: đánh giá hiệu suất thuật toán qua các chỉ số (thời gian chạy, chi phí đường đi)

Trực quan hóa kết quả: sử dụng các công cụ trực quan hóa (Matplotlib) để biểu diễn quá trình tìm kiếm đường đi và kết quả đạt được, giúp người đọc dễ dàng hình dung và đánh giá hiệu quả của các thuật toán.

1.6. Bố cục đề tài

Chương 1: Giới thiệu

Chương 2: Cơ sở lý thuyết về bài toán tìm đường đi

Chương 3: Mô tả chi tiết thiết kế hệ thống, từ quy trình tiếp nhận yêu cầu giao hàng, xử lý dữ liệu đầu vào, đến cách thực áp dụng thuật toán để tối ưu lộ trình. Đồng thời, so sánh hiệu quả giữa các thuật toán khác

Chương 4: Đưa ra kết luận về kết quả đạ được, hạn chế của đề tài và hướng phát triển trong tương lai, chẳng hạn như mở rộng bài toán cho đa điểm hoặc tích hợp học máy để dự đoán tắc đường

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Các module sử dụng

2.1.1. matplotlib.pyplot (thường được import là plt)

Đây là một module con của thư viện Matplotlib, một thư viện vẽ đồ thị 2D mạnh mẽ trong Python. pyplot cung cấp một giao diện giống như MATLAB, cho phép người dùng dễ dàng tạo ra các biểu đồ, đồ thị, histogram, scatter plot, và nhiều loại hình trực quan hóa dữ liệu khác.

matplotlib.pyplot xây dựng trên ý tưởng về một "figure" (hình vẽ) chứa một hoặc nhiều "axes" (hệ trục tọa độ). Mỗi axes có thể chứa các "artist" (các đối tượng đồ họa) như đường thẳng, điểm, văn bản, đa giác, v.v.

2.1.2. matplotlib.colors

Module này cung cấp các công cụ để làm việc với màu sắc trong Matplotlib. Nó bao gồm các cách định nghĩa màu (ví dụ: RGB, hex), các colormap (bảng màu) định sẵn, và các lớp để tạo colormap tùy chỉnh hoặc chuyển đổi giữa các hệ màu.

Trong trực quan hóa dữ liệu, màu sắc đóng vai trò quan trọng trong việc truyền tải thông tin. matplotlib.colors cho phép bạn ánh xạ các giá trị dữ liệu đến các màu sắc khác nhau thông qua colormap.

2.1.3. matplotlib.widgets

Module này cung cấp các widget tương tác có thể được nhúng vào figure của Matplotlib. Các widget như nút bấm (Button), thanh trượt (Slider), nút radio (RadioButtons), hộp kiểm (CheckButtons), v.v., cho phép người dùng tương tác với đồ thị, điều chỉnh tham số, hoặc kích hoạt các hành động.

2.1.4. numpy (thường được import là np)

Đây là thư viện nền tảng cho tính toán số trong Python. Nó cung cấp hỗ trợ cho các mảng đa chiều (ndarray), các đối tượng ma trận, và một bộ sưu tập lớn các hàm toán học bậc cao để thực hiện các phép toán trên các mảng này một cách hiệu quả.

numpy dựa trên khái niệm về mảng (array) như một cấu trúc dữ liệu đồng nhất, cho phép lưu trữ và xử lý số lượng lớn dữ liệu số một cách nhanh chóng.

2.1.5. heapq

Đây là một module trong thư viện chuẩn của Python, cung cấp việc triển khai thuật toán heap queue (còn được gọi là priority queue). Heap queue là một cấu trúc dữ liệu cây đặc biệt, nơi phần tử nhỏ nhất luôn nằm ở gốc. Nó hỗ trợ các thao tác cơ bản như thêm phần tử (heappush) và lấy phần tử nhỏ nhất (heappop) một cách hiệu quả (thường là độ phức tạp thời gian O(logn)).

2.1.6. Một số module khác

math: Đây là một module trong thư viện chuẩn của Python, cung cấp các hàm toán học cơ bản như căn bậc hai (sqrt), lũy thừa, các hàm lượng giác, v.v.

random: Đây là một module trong thư viện chuẩn của Python, cung cấp các hàm để tạo số ngẫu nhiên.

2.1.7. Công cụ lập trình Visual Studio Code

Visual Studio Code là một công cụ mạnh mẽ có thể giúp nhóm em:

+ Viết mã hiệu quả và chính xác hơn nhờ các tính năng soạn thảo thông minh.

+ Dễ dàng tìm và sửa lỗi bằng trình gỡ lỗi tích hợp.

+ Trực quan hóa kết quả của thuật toán thông qua tích hợp với Matplotlib.

+ Quản lý mã nguồn và phiên bản một cách hiệu quả với tích hợp Git.

+ Mở rộng khả năng của trình soạn thảo thông qua vô số extensions hữu ích.

Tóm lại, đồ án "Thuật toán tìm kiếm đường đi và ứng dụng trong trò giải mã mê cung" đã tận dụng sức mạnh của các module Python như matplotlib cho việc trực quan hóa, numpy cho xử lý dữ liệu lưới hiệu quả, heapq cho việc triển khai các thuật toán tìm kiếm ưu tiên, cùng với các công cụ hỗ trợ khác như math và random. Đặc biệt, công cụ lập trình Visual Studio Code đã đóng vai trò quan trọng trong việc cung cấp một môi trường phát triển hiệu quả, hỗ trợ viết mã, gỡ lỗi và quản lý dự án một cách tối ưu. Sự kết hợp giữa các thư viện mạnh mẽ và một công cụ phát triển hiện đại đã tạo điều kiện thuận lợi cho việc nghiên cứu, triển khai và thử nghiệm các thuật toán tìm kiếm đường đi, từ đó xây dựng thành công ứng dụng giải mã mê cung này.

2.2. Bài toán tìm đường đi

Trong lĩnh vực trí tuệ nhân tạo và robotics, bài toán tìm đường đi (pathfinding) là một trong những vấn đề cơ bản và quan trọng, liên quan đến việc xác định một chuỗi các hành động hoặc một lộ trình tối ưu (hoặc khả thi) giữa điểm bắt đầu và điểm đích trong một môi trường cụ thể. Bài toán này thường đi kèm với các ràng buộc như tránh chướng ngại vật, tối ưu hóa thời gian di chuyển hoặc giảm thiểu chi phí. Một ví dụ điển hình là việc robot di chuyển trong mê cung, nơi mà nó cần tìm đường từ vị trí ban đầu đến đích một cách hiệu quả nhất.

Không gian trạng thái (State Space) trong bài toán này có thể được mô hình hóa bằng các ô lưới của mê cung. Mỗi ô đại diện cho một trạng thái mà robot có thể ở tại một thời điểm. Tập hợp tất cả các ô mà robot có thể di chuyển qua chính là không gian trạng thái. Ví dụ, trong một mê cung kích thước 4x4, không gian trạng thái bao gồm 16 ô (nếu không có ô nào bị chặn). Nếu có các chướng ngại vật, một số trạng thái sẽ không khả dụng, làm thu hẹp không gian tìm kiếm.

Hành động (Actions) mà robot có thể thực hiện thường là các bước di chuyển đơn giản như lên, xuống, trái, phải (trong môi trường 4 hướng) hoặc thêm các hướng chéo (trong môi trường 8 hướng). Từ một ô hiện tại, robot có thể chọn một hành động hợp lệ để chuyển sang trạng thái mới, miễn là ô đó không bị chặn hoặc vượt ra khỏi biên của mê cung.

Chi phí (Cost) liên quan đến mỗi hành động thường được xác định dựa trên độ dài đường đi. Trong trường hợp đơn giản, mỗi lần di chuyển sang ô kế cận có chi phí là 1. Tuy nhiên, trong một số bài toán phức tạp hơn, chi phí có thể thay đổi tùy theo địa hình (ví dụ: di chuyển qua vùng bùn lầy có chi phí cao hơn đường bằng phẳng).

Mục tiêu (Goal) của bài toán là đưa robot đến một trạng thái cụ thể, thường là ô đích trong mê cung. Quá trình tìm kiếm sẽ kết thúc khi robot đến được vị trí này, và đường đi tối ưu được đánh giá dựa trên tổng chi phí thấp nhất (hoặc thỏa mãn một số tiêu chí khác như thời gian ngắn nhất).

Như vậy, bài toán tìm đường đi không chỉ đơn thuần là xác định một lộ trình từ điểm A đến điểm B, mà còn đòi hỏi sự cân nhắc về hiệu suất, ràng buộc môi trường và phương pháp tối ưu hóa. Các thuật toán như Dijkstra, A*, hay thuật toán tìm kiếm theo chiều rộng (BFS) thường được áp dụng để giải quyết vấn đề này một cách hiệu quả.

2.3. Mô hình hóa môi trường thực hiện

2.3.1. Tính rời rạc hóa không gian

Trong thực tế, môi trường thường là liên tục (ví dụ: không gian vật lý, bản đồ địa lý). Tuy nhiên, để thuận tiện cho việc tính toán và lập trình, chúng ta thường rời rạc hóa không gian bằng cách chia nó thành một lưới ô vuông (grid). Lý do sử dụng mô hình lưới là vì sự đơn giản: thay vì làm việc với tọa độ liên tục (số thực) thì ta chỉ cần xử lý các ô vuông rời rạc (số nguyên), dễ dàng biểu diễn và xử lý: cấu trúc lưới phù hợp với cách máy tính lưu trữ dữ liệu (ma trận, mảng 2D), phù hợp với nhiều thuật toán tìm kiếm như BFS, DFS, A*, Dijkstra,...

2.3.2. Biểu diễn trạng thái của các ô

Mỗi ô trong lưới có thể có một trong các trạng thái sau:

Ô trống: Robot có thể di chuyển vào ô này. (mang giá trị 0).

Ô chướng ngại vật: Robot không thể di chuyển vào ô này (mang giá trị 1).

Ô bắt đầu: Vị trí xuất phát của robot (mang icon lá cờ).

Ô mục tiêu: Vị trí đích mà robot cần đến (mang icon tấm bia)

2.3.3 Thuật toán tạo mê cung

a) Recursive Backtracking (Quay lui đệ quy)

+ Nguyên lý: Recursive Backtracking là một thuật toán dựa trên tìm kiếm theo chiều sâu (DFS), tạo mê cung bằng cách "khắc" các lối đi từ một ô ban đầu. Thuật toán chọn ngẫu nhiên một ô lân cận chưa được thăm, phá bỏ bức tường giữa hai ô, và tiếp tục cho đến khi tất cả các ô được thăm.

+ Quy trình:

- Chọn một ô bắt đầu và đánh dấu là đã thăm.

- Chọn ngẫu nhiên một ô lân cận chưa thăm, phá tường giữa chúng.

- Đệ quy tiếp tục từ ô mới, quay lui khi không còn ô lân cận nào chưa thăm.

- Kết thúc khi tất cả ô được thăm.

+ Ưu điểm: Tạo mê cung hoàn hảo (chỉ có một đường duy nhất giữa hai điểm bất kỳ), đơn giản để triển khai.

+ Nhược điểm: Có thể tạo các đường đi dài, thiên về cấu trúc dạng "đường hầm".

+ Ứng dụng: Phù hợp với mê cung cần độ phức tạp cao, như trong các bài toán tìm đường thử nghiệm.

b) Prim

+ Nguyên lý: Thuật toán Prim tạo mê cung dựa trên ý tưởng của thuật toán Prim trong lý thuyết đồ thị, xây dựng cây bao trùm tối thiểu (MST). Nó bắt đầu từ một ô và dần mở rộng bằng cách thêm các ô lân cận với chi phí ngẫu nhiên.

+ Quy trình:

- Chọn một ô bắt đầu, thêm vào tập hợp ô đã thăm.

- Thêm tất cả các ô lân cận vào hàng đợi ưu tiên với chi phí ngẫu nhiên.

- Lấy ô có chi phí thấp nhất từ hàng đợi, phá tường để kết nối với ô đã thăm.

- Lặp lại cho đến khi tất cả ô được thăm.

+ Ưu điểm: Tạo mê cung với nhiều nhánh ngắn, ít đường dài hơn Recursive Backtracking.

+ Nhược điểm: Có thể yêu cầu nhiều bộ nhớ hơn cho hàng đợi ưu tiên.

+ Ứng dụng: Thích hợp cho mê cung cần phân bố đồng đều hơn về độ dài đường đi.

c) Kruskal

+ Nguyên lý: Thuật toán Kruskal cũng dựa trên cây bao trùm tối thiểu, nhưng sử dụng cách tiếp cận hợp nhất các tập hợp (Union-Find). Nó xem xét tất cả các bức tường trong mê cung và phá chúng ngẫu nhiên nếu không tạo vòng lặp.

+ Quy trình:

- Khởi tạo mỗi ô là một tập hợp riêng biệt.

- Tạo danh sách tất cả các bức tường, sắp xếp ngẫu nhiên.

- Xử lý từng bức tường: nếu hai ô nó nối thuộc hai tập hợp khác nhau, phá tường và hợp nhất tập hợp.

- Kết thúc khi tất cả ô thuộc cùng một tập hợp.

+ Ưu điểm: Tạo mê cung với cấu trúc ngẫu nhiên, ít thiên hướng về đường dài.

+ Nhược điểm: Cần cấu trúc dữ liệu Union-Find, phức tạp hơn một chút để triển khai.

+ Ứng dụng: Phù hợp khi cần mê cung với tính ngẫu nhiên cao.

d) Eller

+ Nguyên lý: Thuật toán Eller tạo mê cung từng hàng một, đảm bảo mê cung hoàn hảo mà không cần lưu trữ toàn bộ lưới trong bộ nhớ. Nó sử dụng các tập hợp để theo dõi kết nối giữa các ô.

+ Quy trình:

- Khởi tạo hàng đầu tiên, mỗi ô là một tập hợp riêng.

- Ngẫu nhiên phá các bức tường ngang giữa các ô lân cận nếu chúng thuộc tập hợp khác nhau.

- Ngẫu nhiên phá ít nhất một bức tường dọc để kết nối với hàng tiếp theo.

- Lặp lại cho mỗi hàng, hợp nhất các tập hợp khi cần, đến hàng cuối cùng thì đảm bảo tất cả ô được kết nối.

+ Ưu điểm: Hiệu quả về bộ nhớ, chỉ lưu trữ một hàng tại một thời điểm.

Nhược điểm: Phức tạp hơn trong việc đảm bảo tính ngẫu nhiên và kết nối.

Ứng dụng: Lý tưởng cho các hệ thống tài nguyên hạn chế hoặc mê cung lớn.

2.3.4. Các biến thể trong mê cung

a) EcoBot Navigator

+ Nguyên lý: EcoBot Navigator là một phương pháp tìm đường được thiết kế cho các robot trong EcoBot Challenge, tập trung vào điều hướng mê cung với các chướng ngại vật chủ đề nông nghiệp. Thuật toán này thường dựa trên các chiến lược tìm kiếm đơn giản như Wall-Following (theo tường) hoặc A*, nhưng được tùy chỉnh để xử lý các yếu tố môi trường như câu hỏi kiến thức hoặc điều kiện địa hình.

+ Ưu điểm: Linh hoạt, phù hợp với môi trường mê cung có yếu tố tương tác.

+ Nhược điểm: Phụ thuộc vào cảm biến và lập trình cụ thể của robot.

b) Mud Maze

+ Nguyên lý: Mud Maze là một biến thể tìm đường trong môi trường số hoặc thực tế, nơi địa hình (như bùn hoặc nước) ảnh hưởng đến khả năng di chuyển. Thuật toán thường dựa trên

Breadth-First Search (BFS) hoặc Dijkstra, với trọng số được gán cho các ô có địa hình khó (như bùn làm chậm di chuyển).

+ **Ưu điểm**: Xử lý tốt các mê cung với địa hình phức tạp.

+ **Nhược điểm**: Tăng độ phức tạp tính toán khi có nhiều loại địa hình.

Blind Maze

+ **Nguyên lý**: Blind Maze yêu cầu điều hướng mà không có thông tin đầy đủ về cấu trúc mê cung, tương tự các bài toán tìm đường trong điều kiện hạn chế thông tin. Thuật toán thường dựa trên Random Walk hoặc Pledge Algorithm, nơi robot di chuyển ngẫu nhiên hoặc theo quy tắc cố định cho đến khi tìm được đích.

+ **Ưu điểm**: Đơn giản, không cần bản đồ toàn cục.

+ **Nhược điểm**: Hiệu quả thấp, có thể mất nhiều thời gian để tìm đích.

2.4. Các thuật toán tìm đường

Trong lĩnh vực trí tuệ nhân tạo và lý thuyết đồ thị, có rất nhiều thuật toán tìm kiếm đường đi đóng vai trò quan trọng trong việc giải quyết các bài toán tối ưu hóa. Đồ án của nhóm em lựa chọn ba thuật toán phổ biến để thực hiện, đó là A*, Breadth-First Search (BFS), Depth-First Search (DFS) và Dijkstra's algorithm, các thuật toán đều có những ưu, nhược điểm riêng, phù hợp với các tình huống khác nhau. Các phân tích sau đây sẽ cho thấy cơ chế hoạt động, hiệu quả, và ứng dụng cơ bảncủa từng thuật toán.

2.4.1. Thuật toán A* (A-Star Algorithm)

Cơ chế hoạt động: A* là sự kết hợp giữa tìm kiếm theo chi phí (UCS) và tìm kiếm heuristic (Greedy). Nó sử dụng hàm đánh giá: $f(n)=g(n)+h(n)$

$g(n)$: Chi phí thực tế từ điểm xuất phát đến nút hiện tại.

$h(n)$: Hàm heuristic ước lượng chi phí từ nút hiện tại đến đích.

Ưu điểm: A*là luôn tìm được đường đi ngắn nhất nếu heuristic chấp nhận được (không ước lượng quá chi phí thực) và nhờ heuristic, A* giảm đáng kể số lượng nút cần duyệt so với BFS và UCS.

Nhược điểm: nếu heuristic kém chất lượng, hiệu suất của chương trình sẽ giảm và lưu trữ nhiều trạng thái trong hàng đợi ưu tiên dẫn đến tốn kém bộ nhớ.

2.4.2. Thuật toán BFS (Breadth-First Search)

Cơ chế hoạt động: BFS hoạt động trên nguyên tắc tìm kiếm theo chiều rộng, nghĩa là nó sẽ duyệt tất cả các đỉnh ở cùng một mức độ sâu trước khi chuyển sang mức tiếp theo. Thuật toán này sử dụng cấu trúc dữ liệu hàng đợi (queue) để đảm bảo thứ tự duyệt các đỉnh.

Ưu điểm : luôn tìm được đường đi ngắn nhất (theo số cạnh) đảm bảo tính tối ưu của chương trình và không phụ thuộc vào heuristic, đơn giản vì chỉ sử dụng queue.

Nhược điểm: so với A* thì BFS kém hiệu quả với đồ thị lớn do phải duyệt nhiều nút không liên quan tới đích và không xử lý được đồ thị có trọng số khác nhau giữa các cạnh.

2.4.3. Thuật toán DFS (Depth-First Search)

Cơ chế hoạt động: DFS tập trung vào việc khám phá theo chiều sâu. DFS sẽ đi theo một nhánh đến tận cùng trước khi quay lui để khám phá các nhánh khác. Thuật toán này có thể cài đặt bằng đệ quy hoặc sử dụng ngăn xếp (stack).

Ưu điểm: tiết kiệm bộ nhớ do chỉ cần lưu trữ đường đi hiện tại, đồng thời phù hợp cho các bài toán cần duyệt toàn bộ không gian trạng thái.

Nhược điểm: không đảm bảo tìm được đường đi ngắn nhất và có thể rơi vào vòng lặp vô hạn nếu không được kiểm soát.

2.4.4. Thuật toán Dijkstra

Cơ chế hoạt động: Dijkstra là một thuật toán tìm đường đi ngắn nhất cổ điển dành cho đồ thị có trọng số không âm. Khác với BFS, Dijkstra sử dụng hàng đợi ưu tiên (priority queue) để luôn chọn ra đỉnh có chi phí tích lũy nhỏ nhất tại mỗi bước.

Ưu điểm: đảm bảo tìm được đường đi tối ưu nhờ nguyên lý "lựa chọn tham lam" (greedy choice property). Ưu điểm chính của Dijkstra là hiệu quả cao với đồ thị có trọng số.

Nhược điểm: chậm hơn A* trong nhiều trường hợp, tốn nhiều bộ nhớ vì phải lưu trữ nhiều nút không cần thiết do không dùng heuristic và không xử lý được trọng số âm.

2.5. Các hàm heuristic tính khoảng cách sử dụng

2.5.1. Euclidean

Khoảng cách Euclidean hay còn gọi là khoảng cách đường thẳng hoặc đường chim bay, là một phương pháp đo lường trực quan và đơn giản, tính toán khoảng cách ngắn nhất giữa hai điểm trong không gian. Công thức toán học của nó dựa trên định lý Pythagoras. Ưu điểm nổi bật của khoảng cách Euclidean là tính trực quan, phản ánh đúng khoảng cách vật lý, và tính đơn giản trong công thức.

Công thức: $\text{Khoảng cách} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

2.5.2. Manhattan

Khoảng cách Manhattan, ngược lại, đo lường khoảng cách giữa hai điểm bằng tổng các giá trị tuyệt đối của hiệu tọa độ của chúng. Phương pháp này đặc biệt phù hợp với các môi trường có cấu trúc lưới, nơi các đối tượng chỉ có thể di chuyển theo hướng ngang hoặc dọc. Ưu điểm lớn nhất của khoảng cách Manhattan là khả năng phản ánh chính xác các bước di chuyển trên lưới 4 hướng và hiệu suất tính toán cao do chỉ sử dụng phép cộng và trị tuyệt đối.

Công thức: $\text{Khoảng cách} = |x_2 - x_1| + |y_2 - y_1|$

2.5.3. Chebyshev

Khoảng cách Chebyshev đo lường khoảng cách giữa hai điểm bằng giá trị lớn nhất của hiệu tuyệt đối của tọa độ của chúng. Phương pháp này đặc biệt hữu ích trong các không gian cho phép di chuyển theo 8 hướng (ngang, dọc và chéo), với giả định rằng chi phí cho mỗi bước di chuyển là như nhau. Ưu điểm của khoảng cách Chebyshev là khả năng phản ánh chính xác số bước di chuyển tối thiểu trong môi trường 8 hướng và tốc độ tính toán tương đối nhanh.

Công thức: $\text{Khoảng cách} = \max(|x_2 - x_1|, |y_2 - y_1|)$

2.5.4. Octicle

Là một biến thể của khoảng cách, kết hợp Manhattan và Chebyshev, thường dùng cho di chuyển 8 hướng và tính toán phức tạp hơn một chút, nhưng chính xác hơn cho lưới 8 hướng.

2.5.5. Tie-breaking

Tie-breaking không phải là một thuật toán đo khoảng cách mà là một tập hợp các kỹ thuật được sử dụng trong các thuật toán tìm kiếm dựa trên chi phí như A*. Khi có nhiều đường đi hoặc nút có cùng giá trị hàm đánh giá, tie-breaking giúp thuật toán đưa ra quyết định nhất quán về nút nào sẽ được khám phá tiếp theo.

2.5.6. Angle Euclidean

Khoảng cách Angle Euclidean là một biến thể ít phổ biến hơn của khoảng cách Euclidean, cố gắng tích hợp thêm yếu tố góc giữa các điểm vào phép đo khoảng cách. Mặc dù có tiềm năng trong các ứng dụng đặc thù, chẳng hạn như điều hướng robot với các ràng buộc về hướng di chuyển, việc tính toán góc thường phức tạp hơn và việc đảm bảo tính chấp nhận được của heuristic trở nên khó khăn hơn.

CHƯƠNG 3. PHÂN TÍCH HỆ THỐNG VÀ XÂY DỰNG SẢN PHẨM

Chương này trình bày một cách toàn diện quá trình phân tích, thiết kế, triển khai và xây dựng hệ thống mô phỏng robot tìm đường sử dụng các thuật toán đã học. Hệ thống gồm ba thành phần cốt lõi:

1. Lõi thuật toán (logic): gồm các thuật toán tìm đường và sinh mê cung.

2. Các giao diện mô phỏng độc lập: mỗi loại giao diện phục vụ mục tiêu riêng (mô phỏng cơ bản, mô phỏng địa hình tiêu hao, mê cung biến đổi...).

3. Bộ giao diện so sánh thuật toán: cho phép đánh giá định lượng hai thuật toán bất kỳ trong môi trường giống nhau.

Hệ thống được thiết kế và phát triển bằng Python, sử dụng CustomTkinter để tạo giao diện người dùng thân thiện, trực quan và dễ thao tác. Bên cạnh đó, các thuật toán được tổ chức dưới dạng mô-đun, giúp dễ bảo trì, mở rộng và áp dụng vào nhiều biến thể môi trường khác nhau.

3.1. Phân tích yêu cầu và chức năng hệ thống

3.1.1. Mục tiêu hệ thống

- Cung cấp công cụ mô phỏng robot tìm đường bằng nhiều thuật toán, phục vụ học tập và nghiên cứu.

- Hỗ trợ quan sát trực quan quá trình hoạt động của thuật toán qua animation từng bước.

- Cho phép so sánh hiệu suất các thuật toán về:

+ Số lượng ô đã duyệt.

+Độ dài đường đi tìm được

+ Thời gian thực thi.

- Triển khai các môi trường mô phỏng thực tế như:

+ Địa hình tiêu hao nhiên liệu (EcoBot).

+ Mê cung biến đổi (Mud Maze).

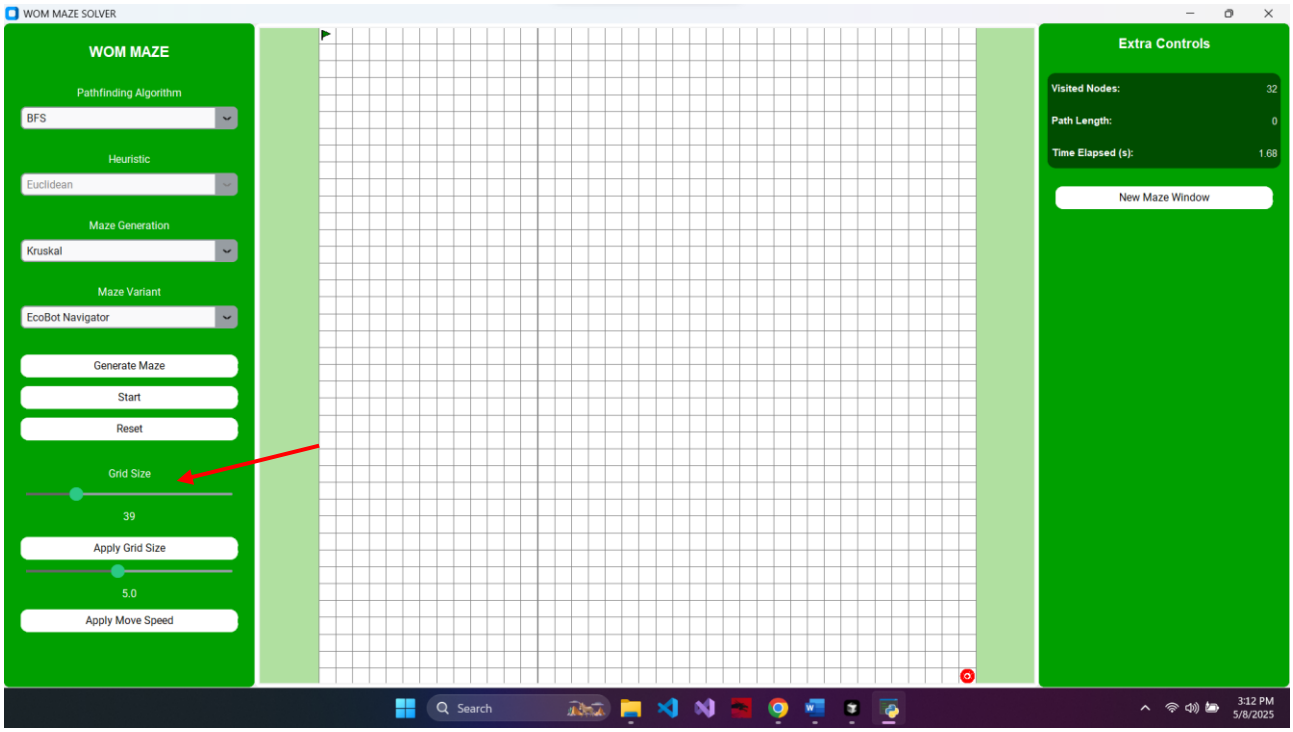
- Hỗ trợ tùy biến cao: sinh mê cung, chọn heuristic, điều chỉnh tốc độ mô phỏng, kích thước lưới...

3.1.2. Yêu cầu chức năng

a) Chức năng khởi tạo mê cung

Cho phép sinh mê cung tự động theo các thuật toán: Recursive Backtracking, Prim, Kruskal, Eller

Chọn kích thước lưới từ 20 đến 100



Tùy chọn điểm bắt đầu, kết thúc, vị trí trạm nhiên liệu (EcoBot) hoặc vị trí bùn (Mud Maze).

b) Chức năng chọn và mô phỏng thuật toán

Thuật toán hỗ trợ: BFS, DFS, Dijkstra, A* với nhiều kiểu heuristic như Manhattan, Euclidean, Chebyshev, Octile...

Animation từng bước với tốc độ điều chỉnh được (delay từ 10 đến 100ms).

Ghi nhận và hiển thị thông tin: số ô duyệt, độ dài đường đi, thời gian, tổng chi phí (với địa hình).

c) Chức năng so sánh hai thuật toán

Giao diện chia đôi canvas, cho phép chạy song song 2 thuật toán trên cùng mê cung.

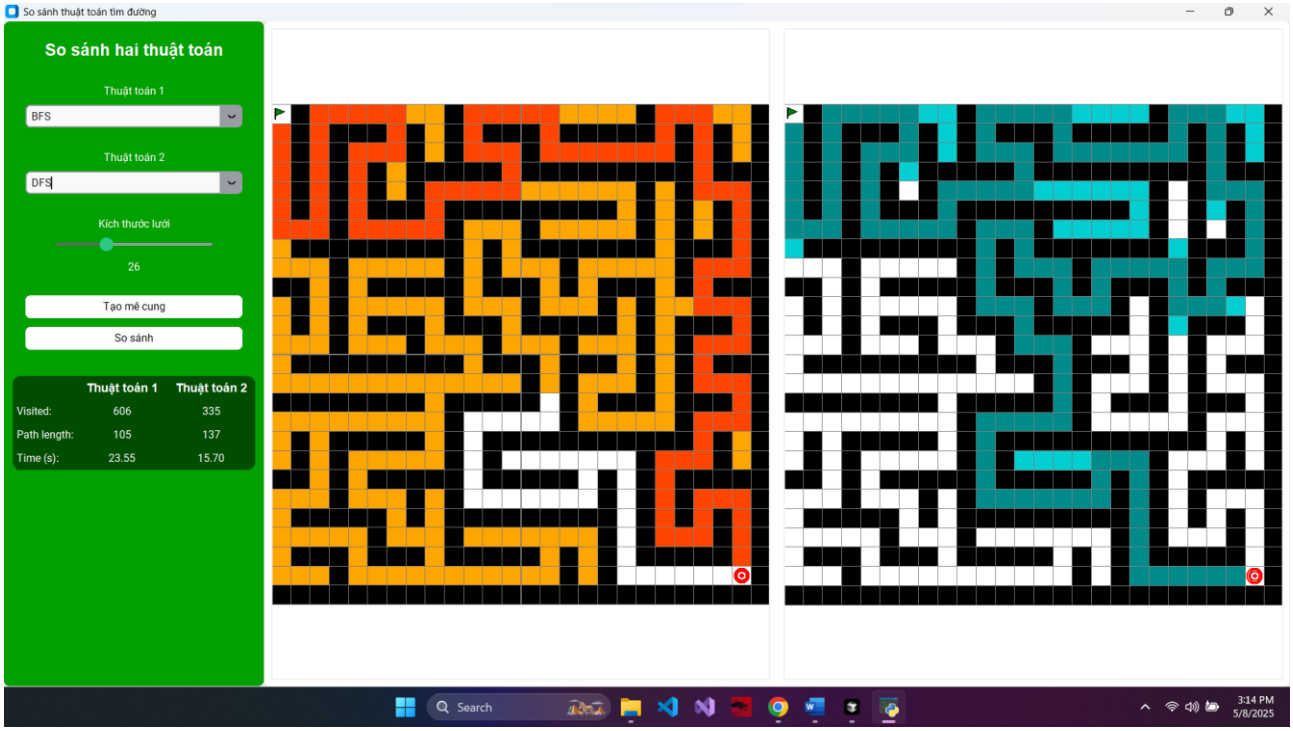
Mỗi thuật toán sử dụng màu sắc khác nhau (cam, vàng, xanh dương, tím) để biểu diễn visited/path

Có bảng thống kê tự động cập nhật kết quả cho từng thuật toán:

+ Visited cells

+ Path length

+ Execution time (s)



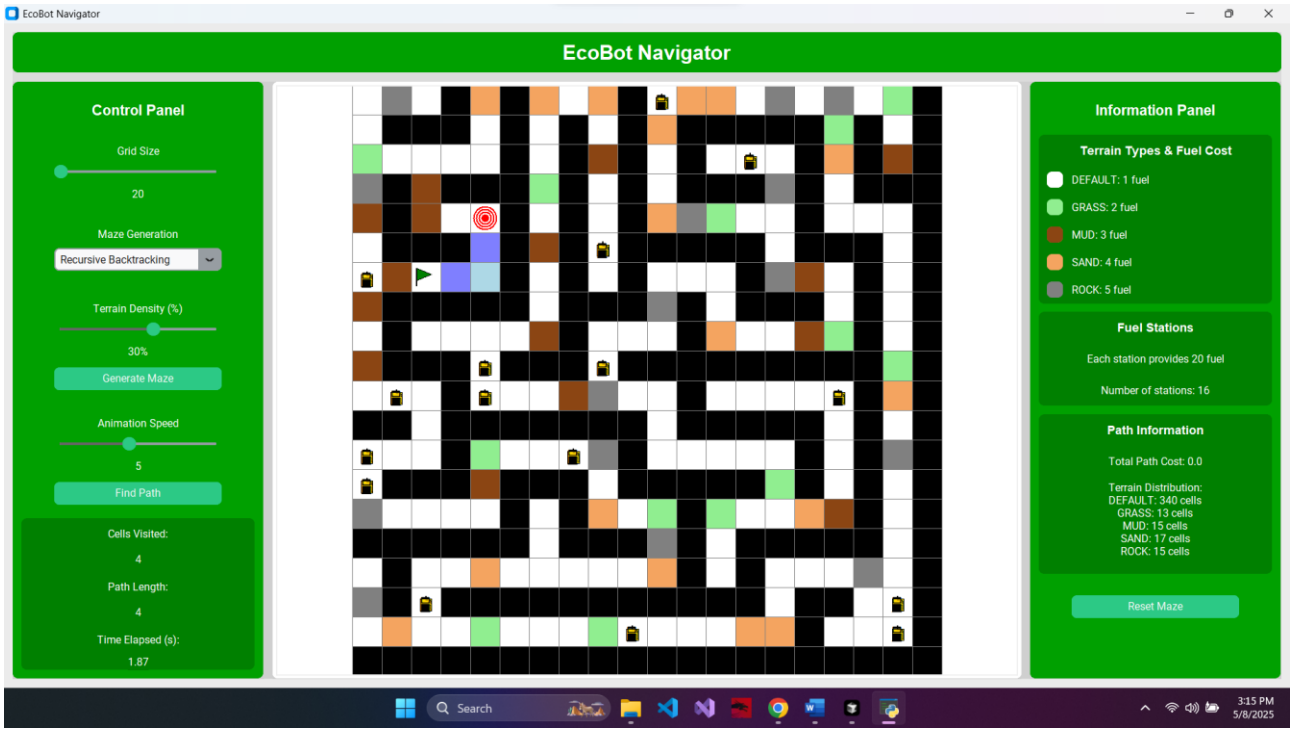
d) Chức năng mô phỏng đặc biệt

- EcoBot Navigator:

+ Mỗi ô địa hình tiêu tốn lượng nhiên liệu khác nhau (đồng bằng, đồi, sỏi, đá...).

+ Mục tiêu là tối ưu hóa đường đi sao cho tổng chi phí nhiên liệu thấp nhất.

+ Có thêm tính năng tiếp nhiên liệu và hiển thị phân bố địa hình



- Mud Maze:

+ Tường mê cung có thể thay đổi ngẫu nhiên trong quá trình tìm đường.

+ Buộc thuật toán tái định tuyến, rất hữu ích để kiểm tra khả năng thích ứng (reactive path planning)

e) Giao diện và tương tác người dùng

Tương tác chuột để đặt điểm bắt đầu/kết thúc/tường.

Nút chức năng rõ ràng, layout logic theo thói quen sử dụng phần mềm trực quan.

Mỗi thay đổi về lưới đều có hiệu ứng thị giác trực tiếp (animation + đổi màu)

3.2. Thiết kế và tổ chức hệ thống

3.2.1. Cấu trúc mã nguồn

Hệ thống chia thành 2 khối rõ ràng:

A. Logic (thuật toán):

File: WOM_MAZE_LOGIC.py

Gồm các module:

generate_maze() – sinh mê cung.

bfs(), dfs(), dijkstra(), astar() – các thuật toán tìm đường.

add_loops(), add_targeted_loops() – tạo thêm đường rẽ để tăng độ phức tạp.

get_neighbors(), get_neighbors_cost(), get_heuristic() – các hàm tiện ích.

B. Giao diện:

WOM_MAZE-UI.py: mô phỏng cơ bản.

WOM_MAZE_COMPARE_UI.py: giao diện so sánh.

WOM_MAZE_ECOBOT_UI.py: mô phỏng môi trường địa hình.

WOM_MAZE_MUD_UI.py: mê cung biến đổi theo thời gian.

3.2.2. Luồng hoạt động chung

Người dùng chọn thuật toán, kích thước, điểm start/end.

Hệ thống sinh mê cung theo thuật toán lựa chọn

Thuật toán được khởi chạy theo dạng generator → từng bước được cập nhật trên canvas.

Khi kết thúc, các chỉ số thống kê được cập nhật và in ra.

3.2.3. Mô hình hóa hệ thống

Use Case Diagram: có 1 tác nhân duy nhất (User) tương tác với 5 chức năng chính.

Activity Diagram: mô tả tuần tự quy trình từ khi sinh mê cung → chọn thuật toán → chạy → hiển thị kết quả.

Class Diagram: phân lớp rõ ràng giữa MazeLogic, MazeApp, CanvasGrid, CompareApp, EcoBotApp.

3.3. Chi tiết thuật toán triển khai

Thuật toán	Đặc điểm	Ưu điểm	Nhược điểm
BFS	Không trọng số	Tìm đường ngắn nhất	Chạy chậm trên lưới lớn
DFS	Không trọng số	Đơn giản, nhanh	Không tối ưu
Dijkstra	Có trọng số	Luôn tìm đường tốt nhất	Tốn tài nguyên (heap)
A*	Có trọng số + heuristic	Nhanh và chính xác	Phụ thuộc chất lượng heuristic

Heuristic được hỗ trợ:

+ Manhattan (nhanh, chính xác cho grid)

+ Euclidean (thực tế hơn)

+ Chebyshev, Octile, Tie-breaking (phức tạp hơn cho nghiên cứu)

3.4. Giao diện và trải nghiệm người dùng

Canvas đồ họa: hiển thị toàn bộ mê cung với kích thước tự động điều chỉnh theo cửa sổ.

Animation trực quan: các bước duyệt màu cam, đường đi màu xanh.

Hiệu ứng icon: điểm bắt đầu có biểu tượng cờ; điểm đích có vòng tròn đỏ đồng tâm

Bảng điều khiển bên phải: chọn thuật toán, kích thước, tốc độ, reset, sinh mê cung.Thông tin động: số ô duyệt, độ dài đường đi, thời gian chạy, phân bố địa hình (EcoBot), tổng nhiên liệu tiêu hao.

3.5. Đánh giá và khả năng mở rộng

3.5.1. Đánh giá

Hệ thống có thể chạy mượt trên lưới 40×40 với thời gian tìm đường dưới 1s (A*, Dijkstra).

Tính trực quan cao, dễ sử dụng với sinh viên, giảng viên và người mới học AI.

Tương thích Windows, Linux, không yêu cầu cài đặt thư viện ngoài ngoại trừ customtkinter.

3.5.2. Mở rộng tương lai

Tích hợp robot thực tế: truyền tín hiệu từ mô hình sang robot Arduino hoặc Raspberry Pi.

Môi trường 3D: chuyển sang PyOpenGL hoặc Unity cho mô phỏng không gian 3 chiều.

Thuật toán nâng cao: Ant Colony, Genetic, Q-learning...

Lưu trạng thái và xuất video: cho phép quay lại toàn bộ mô phỏng để nghiên cứu.

CHƯƠNG 4. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1. Kết luận

Trong quá trình thực hiện đề tài "ứng dụng một số thuật toán mô phỏng robot tìm đường trong mê cung", nhóm em đã đặt ra mục tiêu nghiên cứu và cài đặt một số thuật toán tìm kiếm cổ điển trong trí tuệ nhân tạo như: A*, BFS, DFS, Dijkstra, đồng thời xây dựng hệ thống mô phỏng trực quan quá trình tìm đường của các thuật toán này. Kết quả đạt được so với mục tiêu đã đề ra như sau:

+ Đã hoàn thành việc tìm hiểu, phân tích và lập trình các thuật toán nêu trên.

+ Phân tích ưu nhược điểm, độ phức tạp về thời gian và không gian của từng thuật toán trong bối cảnh bài toán tìm đường trong mê cung.

+ Nghiên cứu các khái niệm liên quan như không gian trạng thái, hàm heuristic (đối với A*), và biểu diễn mê cung trên máy tính.

+ Đã thực hiện trên nhiều loại mê cung khác nhau, thu thập và phân tích dữ liệu về hiệu suất sau đó so sánh và rút ra kết luận về ưu điểm và nhược điểm giữa các thuật toán tìm đường nêu trên.

+ So với các sản phẩm tương tự, hệ thống của nhóm em tuy còn đơn giản về mặt giao diện nhưng có điểm nổi bật là tích hợp nhiều thuật toán khác nhau trong cùng một hệ thống, giúp dễ dàng đánh giá và học tập.

Trong suốt quá trình thực hiện, nhóm em đã:

+ Làm được: Tự nghiên cứu tài liệu tiếng Anh chuyên ngành, vận dụng được kiến thức đã học để xây dựng mô phỏng và giải quyết bài toán thực tế.

+ Chưa làm được: Chưa triển khai mô hình trên robot vật lý; chưa có giao diện người dùng (UI) thân thiện.

+ Đóng góp nổi bật: Tổng hợp các thuật toán tìm kiếm phổ biến, thể hiện bằng mô phỏng trực quan và dễ so sánh.

+ Bài học rút ra: Cần lập kế hoạch kỹ lưỡng hơn trong việc phân bổ thời gian; nên thường xuyên kiểm thử từng phần nhỏ của hệ thống trước khi tích hợp toàn bộ.

4.2. Hướng phát triển

Dự án hiện tại đã đạt được các mục tiêu quan trọng: triển khai các thuật toán tìm đường như A*, BFS, DFS, Dijkstra với giao diện trực quan, dễ sử dụng. Đặc biệt, với sự đa dạng về môi trường mê cung và biến thể khác nhau đã đem tới nhiều góc nhìn khác nhau cho người dùng, từ đó giúp nhận biết được từng ưu điểm và nhược điểm riêng biệt trong các môi trường khác nhau. Tuy nhiên, để tiếp tục phát triển và mở rộng ứng dụng, dự án có thể cải tiến theo các hướng sau:

Mở rộng danh sách thuật toán: Hiện tại, dự án đã hỗ trợ nhiều thuật toán tìm kiếm phổ biến, nhưng sẽ còn thú vị hơn nếu bổ sung những thuật toán nâng cao như Greedy Best-First Search, UCS,...

Ngoài ra, có thể mở rộng sang các thuật toán dành cho môi trường động, chẳng hạn như D Lite*, giúp xử lý các mê cung có tường thay đổi theo thời gian thực, tạo điều kiện thử nghiệm trong những bài toán phức tạp hơn.

Nếu dự án muốn mở rộng hơn nữa, việc tích hợp mê cung 3D hoặc đồ thị không phải dạng lưới (graph-based) sẽ là bước tiến lớn, phù hợp để mô phỏng các tình huống như điều hướng robot hoặc lập kế hoạch đường đi trong không gian thực.

Tối ưu giao diện và trải nghiệm người dùng bằng một giao diện mượt mà, dễ thao tác sẽ giúp người dùng tiếp cận và thử nghiệm thuật toán tốt hơn. Một số cải tiến có thể thực hiện như các công cụ chỉnh sửa nâng cao, chẳng hạn như kéo thả để đặt nhiều tường hoặc đồng xu cùng lúc thay vì thao tác từng ô một, lưu/tải cấu hình mê cung dưới dạng tệp, giúp người dùng chia sẻ và tiếp tục thử nghiệm dễ dàng và phát triển phiên bản web bằng HTML5 và JavaScript, giúp ứng dụng có thể chạy trực tiếp trên trình duyệt mà không cần cài đặt Pygame. Bên cạnh đó, còn có thể tích hợp chế độ hướng dẫn (tutorial mode) giúp người dùng hiểu từng thuật toán thông qua các bước tương tác trực tiếp.

Nâng cao khả năng phân tích và đo lường hiệu suất để hiểu rõ sức mạnh của từng thuật toán. Dự án có thể bổ sung chức năng so sánh chi tiết hiệu suất, theo các yếu tố như thời gian chạy, số nút đã thăm, bộ nhớ sử dụng, giúp người dùng có góc nhìn toàn diện hơn. Ngoài ra, có thể tích hợp biểu đồ hiệu suất sẽ giúp trực quan hóa kết quả, từ đó giúp đưa ra những đánh giá sâu sắc hơn về ưu và nhược điểm của từng thuật toán.

Ứng dụng vào thực tế, nghiên cứu khả năng ứng dụng các thuật toán này trong các bài toán thực tế khác như điều hướng robot trong kho hàng, tìm đường đi cho xe tự lái, hoặc giải các bài toán quy hoạch đường đi trong game.

Những cải tiến trên giúp mở rộng phạm vi ứng dụng và nâng cao giá trị giáo dục, nghiên cứu và công nghiệp, đồng thời duy trì mục tiêu trực quan hóa thuật toán và học thuật trong lĩnh vực tìm đường đi.

TÀI LIỆU THAM KHẢO

[1] Russell, S., & Norvig, P. (2020). Artificial intelligence: A modern approach (4th ed.). Pearson.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.

[3] Benjamin Baka. (2017). Python Data Structures and Algorithms. https://edu.anarcho-copy.org/Programming%20Languages/Python/Python%20Data%20Structures%20and%20Algorithms.pdf

[4] Koenig, S., & Likhachev, M. (2002). D* Lite. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 15, pp. 476–483).

[5] Passino, K. M., & Antsaklis, P. J. (1990). A system and control theoretic perspective on artificial intelligence planning systems. IEEE Transactions on Systems, Man, and Cybernetics, 20(2), 296–310. https://doi.org/10.1109/21.52547

[6] GeeksforGeeks. (2023). A* search algorithm. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/a-search-algorithm/

[7] Bellman, R. (1958). On a routing problem. Quarterly of Applied Mathematics, 16(1), 87–90. https://doi.org/10.1090/qam/102435

[8] Graphviz. (n.d.). Graphviz - Graph visualization software. Graphviz. Retrieved from https://graphviz.org/

[9] VNOI Wiki. (n.d.). VNOI Wiki. Retrieved from VNOI Wiki | VNOI Wiki

[10] Bùi Minh Quân. (n.d.). Các slide bài giảng.