|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BỘ CÔNG THƯƠNG**  **TRƯỜNG ĐẠI HỌC ĐIỆN LỰC**  **KHOA CÔNG NGHỆ THÔNG TIN**  **C:\Users\VCIS\Desktop\logo-epu-inkythuatso-14-15-47-22.jpg**  **BÁO CÁO CHUYÊN ĐỀ HỌC PHẦN**  **TRÍ TUỆ NHÂN TẠO**  **THUẬT TOÁN TÌM ĐƯỜNG MỌI GÓC TIÊU ĐIỂM DỰA TRÊN A\* TRONG ĐỒ THỊ CÓ KHẢ NĂNG HIỂN THỊ**   |  |  | | --- | --- | | **Giảng viên hướng dẫn** | **: TS. VŨ VĂN ĐỊNH** | | **Sinh viên thực hiện** | **: PHẠM ĐĂNG KHUÊ 22810310270** | |  | **VÕ NGUYÊN BÁCH 22810310239** | |  | **NGUYỄN AN NINH 22810310288** | | **Ngành** | **: CÔNG NGHỆ THÔNG TIN** | | **Chuyên ngành** | **: CÔNG NGHỆ PHẦN MỀM** | | **Lớp** | **: D17CNPM4** | | **Khóa** | **: 2022 – 2027** |   ***Hà Nội, ngày … tháng ... năm 2025*** |

**PHIẾU CHẤM ĐIỂM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Họ và tên sinh viên** | **Nội dung thực hiện** | **Điểm** | **Chữ ký** |
| **1** | Phạm Đăng Khuê | * Mô phỏng thuật toán * Nghiên cứu hướng phát triển * Viết báo cáo |  |  |
| **2** | Võ Nguyên Bách | * Dịch bài báo * Nghiên cứu phương pháp và thuật toán |  |  |
| **3** | Nguyễn An Ninh | * Dịch bài báo * Chạy thuật toán và kiểm tra thực nghiệm |  |  |

|  |  |  |
| --- | --- | --- |
| **Họ và tên giảng viên chấm** | **Chữ ký** | **Ghi chú** |
| Giảng viên chấm 1: |  |  |
| Giảng viên chấm 2: |  |  |

LỜI CẢM ƠN

Trong bối cảnh cuộc cách mạng công nghiệp số đang diễn ra mạnh mẽ trên toàn thế giới, việc ứng dụng trí tuệ nhân tạo (AI) vào đời sống và sản xuất đang trở thành một xu thế tất yếu, đóng vai trò quan trọng trong việc định hình tương lai. Từ các ứng dụng như trợ lý ảo, tự động hóa quy trình sản xuất, đến các giải pháp thông minh hỗ trợ trong giáo dục và y tế, trí tuệ nhân tạo đã và đang mang lại những giá trị vượt trội, tối ưu hóa nguồn lực và nâng cao hiệu quả hoạt động trong nhiều lĩnh vực. Đây không chỉ là kết quả của sự phát triển công nghệ mà còn là động lực thúc đẩy sự đổi mới sáng tạo, mở ra nhiều cơ hội và thách thức mới trong ngành công nghệ thông tin.

Với tinh thần tiếp cận và ứng dụng công nghệ tiên tiến, chúng em đã thực hiện đồ án trí tuệ nhân tạo nhằm nghiên cứu và triển khai các thuật toán tìm kiếm thông minh, cụ thể là thuật toán "A\*" và "FA\*". Thông qua đồ án này, chúng em đã xây dựng thành công một trò chơi tìm đường đi ngắn nhất, không chỉ nhằm minh họa hiệu quả của các thuật toán trong việc tối ưu hóa không gian và thời gian mà còn để tạo cơ hội thực hành và nâng cao kỹ năng lập trình, phân tích, cũng như tư duy giải thuật. Đây là một bước đi quan trọng giúp chúng em tiếp cận sâu hơn với lĩnh vực trí tuệ nhân tạo và khẳng định khả năng ứng dụng thực tiễn của các kiến thức đã học.

Trong suốt quá trình thực hiện đồ án, chúng em nhận được sự hướng dẫn tận tình và hỗ trợ quý báu từ thầy Vũ Văn Định. Thầy không chỉ truyền đạt cho chúng em những kiến thức chuyên môn cần thiết mà còn động viên, chỉ dẫn chúng em cách tiếp cận vấn đề một cách khoa học và hiệu quả. Những lời khuyên và góp ý của thầy đã giúp chúng em hiểu rõ cách thức hoạt động của các thuật toán, đồng thời hoàn thiện bài báo cáo này một cách tốt nhất.

Chúng em hiểu rằng, mặc dù đã nỗ lực hết mình, nhưng với kinh nghiệm nghiên cứu và thực hành còn hạn chế, bài báo cáo không tránh khỏi những thiếu sót. Vì vậy, chúng em rất mong nhận được sự thông cảm và những góp ý quý báu từ thầy để chúng em có thể hoàn thiện đồ án một cách toàn diện hơn trong tương lai. Đây sẽ là nguồn động lực để chúng em tiếp tục nghiên cứu, học hỏi và phát triển trong lĩnh vực trí tuệ nhân tạo.

Cuối cùng, chúng em xin gửi lời cảm ơn chân thành đến thầy Vũ Văn Định và tất cả những người đã tạo điều kiện và hỗ trợ chúng em hoàn thành đồ án này. Những kiến thức, kinh nghiệm và sự chỉ dẫn mà chúng em nhận được sẽ là hành trang quý giá để chúng em tự tin bước tiếp trên con đường học tập và phát triển bản thân.

MỤC LỤC

Trang

[Mở đầu 1](#_Toc190970794)

[Chương 1: TỔNG QUAN VỀ ĐỀ TÀI 2](#_Toc190970795)

[1.1. Tổng quan đề tài 2](#_Toc190970796)

[1.2. Một số giải pháp phổ biến hiện tại 2](#_Toc190970797)

[1.3. Phương hướng nghiên cứu 4](#_Toc190970798)

[CHƯƠNG 2: CÁC PHƯƠNG PHÁP, THUẬT TOÁN TIỀN ĐỀ 5](#_Toc190970799)

[2.1. Đỉnh ứng cử viên (Candidate Vertices) 5](#_Toc190970800)

[2.1.1. Tổng quan và khái niệm về Đỉnh ứng cử viên (Candidate Vertices) 5](#_Toc190970801)

[2.1.2. Tiền xử lý (Preprocessing) 5](#_Toc190970802)

[2.1.3. Sinh Đỉnh Ứng Cử viên (Candidate Vertice) 7](#_Toc190970803)

[2.2. Kiểm tra tầm nhìn (VISIBILITY CHECK) 10](#_Toc190970804)

[Chương 3: FOCAL ANY-ANGLE A\* trên đồ thị tầm nhìn 12](#_Toc190970805)

[2.1. Thuật toán FA-A\* 12](#_Toc190970806)

[2.2. Tính tối ưu 13](#_Toc190970807)

[2.3. Phân tích độ phức tạp thuật toán 14](#_Toc190970808)

[2.4. Cấu trúc dữ liệu 15](#_Toc190970809)

[Chương 4: Kiểm tra số liệu 17](#_Toc190970810)

[4.1. Định tuyến mê cung 17](#_Toc190970811)

[4.2. Tìm đường đi có trọng số nhỏ nhất trên đồ thị trọng số 19](#_Toc190970812)

[4.3. Định tuyến mê cung với đường đi được gắn trọng số 20](#_Toc190970813)

[Chương 5: cài đặt thuật toán 22](#_Toc190970814)

[5.1. Công cụ sử dụng 22](#_Toc190970815)

[5.1.1. Ngôn ngữ lập trình python 22](#_Toc190970816)

[5.1.2. Các thư viện hỗ trợ trong ngôn ngữ lập trình python 23](#_Toc190970817)

[5.2. Sinh mê cung với 1.0.maze\_generator.py 24](#_Toc190970818)

[5.3. Trình diễn định tuyến mê cung với 1.1.FA\_A\_Star.py 25](#_Toc190970819)

[5.3.1. Cung cấp các thư viện thông qua import 25](#_Toc190970820)

[5.3.2. Phần khởi tạo và cấu hình chương trình 25](#_Toc190970821)

[5.3.3. Lớp Node (Đại diện cho một nút trong thuật toán) 25](#_Toc190970822)

[5.3.4. Hàm Heuristic (Tính toán khoảng cách heuristic) 25](#_Toc190970823)

[5.3.5. Hàm lấy các ô lân cận (Get Neighbors) 26](#_Toc190970824)

[5.3.6. Hàm tính toán góc quay giữa hai điểm (Calculate Angle) 26](#_Toc190970825)

[5.3.7. Thuật toán FA-A 26](#_Toc190970826)

[5.3.8. Hàm vẽ mê cung và quá trình tìm kiếm (Draw Maze) 27](#_Toc190970827)

[5.3.9. Hàm tải mê cung từ file (Load Maze from File) 28](#_Toc190970828)

[5.3.10. Quá trình hiển thị và lưu kết quả 28](#_Toc190970829)

[5.4. Sinh ma trận trọng số với 2.0.matrix\_generator.py 29](#_Toc190970830)

[5.5. Trình diễn tìm đường đi ngắn nhất với 2.1.FA\_A\_Star.py 29](#_Toc190970831)

[5.5.1 Cấu hình các thông số mặc định 29](#_Toc190970832)

[5.5.2 Hỗ trợ tìm kiếm 29](#_Toc190970833)

[5.5.3 Thuật toán FA-A\* trong tìm kiếm đường đi ngắn nhất 30](#_Toc190970834)

[5.5.4 Hiển thị nhằm trực quan hóa cho người xem 30](#_Toc190970835)

[5.5.5 Hàm main 31](#_Toc190970836)

[5.6. Sinh mê cung trọng số với 3.0.matrix\_maze\_generator.py 32](#_Toc190970837)

[5.7. Trình diễn định tuyến mê cung với 3.1.FA\_A\_Star.py 33](#_Toc190970838)

[5.7.1. Cấu hình các thông số mặc định 33](#_Toc190970839)

[5.7.2. Hỗ trợ tìm kiếm 33](#_Toc190970840)

[5.7.3. Thuật toán FA-A\* trong định tuyến mê cung có trọng số 33](#_Toc190970841)

[5.7.4. Hiển thị nhằm trực quan hóa cho người xem 34](#_Toc190970842)

[5.7.5. Hàm main 36](#_Toc190970843)

[5.8. Kết quả thu được 37](#_Toc190970844)

[5.8.1. Kết quả thu được từ 1.0.maze\_generator.py 37](#_Toc190970845)

[5.8.2. Kết quả thu được từ 2.0.matrix\_generator.py 38](#_Toc190970846)

[5.8.3. Kết quả thu được từ 3.0.matrix\_maze\_generator.py 39](#_Toc190970847)

[5.8.4. Kết quả thu được từ thuật toán A\* và FA\* trong định tuyến mê cung 39](#_Toc190970848)

[5.8.5. Kết quả thu được từ thuật toán A\* và FA\* trong tìm đường đi ngắn nhất 40](#_Toc190970849)

[5.8.6. Kết quả thu được từ thuật toán A\* và FA\* trong định tuyến mê cung có trọng số 40](#_Toc190970850)

[Kết luận 41](#_Toc190970851)

DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **STT** | **Từ viết tắt** | **Diễn giải** |
|  | FA-A\* | FOCAL ANY-ANGLE A\* |
|  | CV | CANDIDATE VERTICES |

**DANH MỤC BẢNG**

Bảng 1: Bảng thống kê thời gian trung bình 17

Bảng 2: Bảng thống kê số ô trung bình trên đường đi ngắn nhất 18

Bảng 3: Bảng so sánh kết quả trọng số trung bình của 2 thuật toán 20

Bảng 4: Bảng so sánh kết quả trọng số trung bình của 2 thuật toán 21

**Danh mục ảnh**

[Hinh 1.1. Đường đi ngắn nhất đồ thị (a) so với đường đi liên tục ngắn nhất (b) 3](#_Toc190971090)

[Hinh 1.2. Đường đi tìm thấy bởi Theta\* so với đường đi ngắn nhất thực tế 3](#_Toc190971091)

[Hinh 2.1. Đồ thị tầm nhìn của hai nút có hai vật cản và đường đi ngắn nhất 5](#_Toc190971092)

[Hinh 2.2. Di chuyển đường chéo giữa các vật cản 6](#_Toc190971093)

[Hinh 2.3. Một ví dụ về việc tìm các đỉnh ứng viên 7](#_Toc190971094)

[Hinh 2.4. Lưu đồ thuật toán của việc tìm các Đỉnh Ứng Cử Viên 9](#_Toc190971095)

[Hinh 2.5. Mô tả các vật cản nằm giữa nút bắt đầu và nút mục tiêu 10](#_Toc190971096)

[Hinh 2.6. Ví dụ kiểm tra khả năng hiển thị 10](#_Toc190971097)

[Hinh 3.1. Minh họa so sánh đánh giá nút giữa A\* trên lưới và FA-A\* 13](#_Toc190971098)

[Hinh 4.1. Bảng thống kê thời gian chạy trung bình của 2 thuật toán 18](#_Toc190971099)

[Hinh 4.2. Bảng thống kê số ô trên đường đi ngắn nhất trung bình 19](#_Toc190971100)

[Hinh 4.3. Bảng thống kê trọng số trung bình của 2 thuật toán 20](#_Toc190971101)

[Hinh 4.4. Bảng thống kê trọng số trung bình của 2 thuật toán 21](#_Toc190971102)

[Hinh 5.1. Kết quả thu được từ 1.0.maze\_generator.py 37](#_Toc190971103)

[Hinh 5.2. Kết quả thu được từ 2.0.matrix\_generator.py 38](#_Toc190971104)

[Hinh 5.3. Kết quả thu được từ 3.0.matrix\_maze\_generator.py 39](#_Toc190971105)

[Hinh 5.4. Kết quả thu được từ thuật toán A\* và FA\* trong định tuyến mê cung 39](#_Toc190971106)

[Hinh 5.5. Kết quả thu được từ thuật toán A\* và FA\* trong tìm đường đi ngắn nhất 40](#_Toc190971107)

[Hinh 5.6. Kết quả thu được từ thuật toán A\* và FA\* trong định tuyến mê cung trọng số 40](#_Toc190971108)

# Mở đầu

Trong nghiên cứu này, chúng tôi điều tra chủ đề tìm đường và giải mê cung. Một phiên bản thu gọn của đồ thị tầm nhìn (khả năng hiển thị) dựa trên các Đỉnh ứng viên được xây dựng, tiếp theo là một kỹ thuật kiểm tra khả năng hiển thị mới. Sự kết hợp như vậy cho phép chúng tôi nhanh chóng xác định các đỉnh hữu ích và do đó tìm ra đường dẫn tối ưu hiệu quả hơn. Thuật toán được đề xuất được chứng minh trên nhiều trường hợp tìm đường khác nhau. Hiệu suất của kỹ thuật mới trên đồ thị khả năng hiển thị được so sánh với A\* truyền thống trên Lưới, Theta\* và A\* trên Đồ thị tầm nhìn về độ dài đường đi được tìm thấy, số lượng nút được đánh giá cũng như thời gian tính toán. Đóng góp chính của thuật toán là cách tiếp cận mới kết hợp các ưu điểm của phương pháp dựa trên lưới và phương pháp dựa trên đồ thị khả năng hiển thị và do đó mang lại hiệu suất tổng thể tốt hơn.

# Chương 1: TỔNG QUAN VỀ ĐỀ TÀI

## 1.1. Tổng quan đề tài

Vấn đề tìm đường đi ngắn nhất thường gặp trong các lĩnh vực như game, robot, định vị GPS và tìm đường đi, v.v. Một nghiên cứu gần đây đã nghiên cứu phương pháp để có được đường dẫn ngắn nhất đến multiple hydraulic components trong additive manufacturing. Trong đó độ phức tạp của hệ thống kết nối ảnh hưởng trực tiếp đến trọng lượng, kích thước, và chi phí của hệ thống. Tìm đường trên địa hình đã biết thường được chia thành hai bước:

1) phân chia không gian định tuyến liên tục

2) tìm kiếm trên đồ thị để tìm đường đi có chi phí nhỏ nhất, đại diện cho tổng chiều dài của đường đi từ nút bắt đầu đến nút đích trong một đồ thị không trọng số.

Trong nghiên cứu này, để không làm mất tính tổng quát, lưới ô vuông được sử dụng để phân chia địa hình nhờ tính đơn giản và sự phổ biến của nó trong nhiều ứng dụng khác nhau.

Nhiều nghiên cứu đã được thực hiện về vấnđề này. Một trong những nghiên cứu sớm nhất là thuật toán Dijkstra sử dụng giá trị chi phí để tìm đến mục tiêu gần nhất.. Thuật toán A\* đã cải tiến thuật toán Dijkstra bằng cách thêm giá trị ước lượng heuristic từ nút hiện tại đến nút đích vào quá trình đánh giá. Nhờ tính đơn giản và đảm bảo tính tối ưu, thuật toán A\* được sử dụng rộng rãi để giải quyết các bài toán tìm đường vì nó ổn định về mặt toán học sẽ tìm được giải pháp tối ưu.

Tuy nhiên, đường đi ngắn nhất trong đồ thị dạng lưới không thích hợp với đường đi ngắn nhất trong không gian liên tục, nơi mà các đa thức dọc theo cạnh của lưới có thể được thay thế bằng các đường thẳng. Một ví dụ được minh họa trong Hình 1.1(a) cho thấy đường đi ngắn nhất trên đồ thị được tìm bởi thuật toán A\*. Vì thuật toán A\* chỉ có thể thực hiện các bước tiến ngang, dọc hoặc chéo (tức là các đường zigzag) đến các nút lân cận, hướng đi của đường đi bị giới hạn bởi các hướng tiến. Đường đi liên tục ngắn nhất được thể hiện trong Hình 1.1(b), nơi chiều dài đường đi rõ ràng ngắn hơn so với trong Hình 1.1(a).

## 1.2. Một số giải pháp phổ biến hiện tại

Một giải pháp phổ biến cho vấn đề này là áp dụng các kỹ thuật làm mịn hậu xử lý (post-processing smoothing techniques) Để tối ưu hóa đường đi ngắn nhất trên lưới với việc chi phí tính toán tăng lên. Các kỹ thuật hậu xử lý này thường tìm đường đi ngắn hơn ở mọi góc độ so với A\*, nhưng không đảm bảo tìm được đường đi ngắn nhất thực sự. Điều này là do chúng điều chỉnh trực tiếp các đường đi được tìm bởi A\* mà không biết liệu đường đi sau xử lý có tối ưu hay không. Các kỹ thuật hậu xử lý này thường không hiệu quả do thiếu thông tin về các đường đi sau xử lý.

A graph of a graph of a line

Description automatically generated with medium confidence

Hinh 1.1. Đường đi ngắn nhất đồ thị (a) so với đường đi liên tục ngắn nhất (b)

Nhằm khắc phục những hạn chế của A\*, thuật toán Field D\*, một phiên bản nâng cao của D\* Lite, đã được đề xuất, sử dụng nội suy tuyến tính chi phí đường đi của các đỉnh để tìm đường đi ở mọi góc độ. Tuy nhiên, Field D\* có thể dẫn đến việc tạo ra các đường đi có thay đổi hướng không cần thiết, cần phải áp dụng thêm bước làm mịn. Một số phương pháp khác, chẳng hạn như Theta\* và Lazy Theta\*, cũng đã được đề xuất. Các phương pháp này tích hợp quá trình làm mịn vào quá trình tìm kiếm của A\* để giảm bớt ràng buộc về hướng tìm đường (45 hoặc 90 độ).

Thuật toán Theta\* tìm các đường đi ngắn hơn trong thời gian ngắn hơn so với Field D\*. Tuy nhiên, tính tối ưu vẫn không được đảm bảo. Hình 1.2 minh họa đường đi tối ưu được tìm thấy bởi Theta\* và đường đi ngắn nhất .Giống như Theta, thuật toán Accelerated A là một thuật toán tìm đường với khả năng tối ưu ở mọi góc độ, tuy nhiên không có cơ sở lý thuyết vững chắc để chứng minh. Thuật toán này có thể bao gồm một số lượng lớn các nút trong danh sách đóng (Closed List), đồng nghĩa với việc cần phải xét nhiều nút trong các bài toán phức tạp

A graph with a line going up

Description automatically generated

Hinh 1.2. Đường đi tìm thấy bởi Theta\* so với đường đi ngắn nhất thực tế

Mặc dù trong các lĩnh vực ứng dụng như game và navigations, Trong khi hiệu quả tính toán được ưu tiên hơn thuộc tính đường đi ngắn nhất, điều này có thể trở thành một vấn đề quan trọng trong các tình huống thực tế. Ví dụ, các đường nối giữa các thành phần quyết định tổng kích thước hệ thống được tạo, từ đó ảnh hưởng đến kích thước, thời gian tạo và chi phí. Một phương pháp phổ biến để tìm đường ngắn nhất trên địa hình đã biết với các vật cản đa giác là A\* trên Đồ thị Tầm nhìn (Visibility Graphs). Tuy nhiên, A\* trên Đồ thị Tầm nhìn gặp phải vấn đề kém hiệu quả vì nó tìm kiếm dọc theo các cạnh của đồ thị tầm nhìn (vision graph), số lượng cạnh này tăng theo cấp số nhân cùng với kích thước của lưới.

Mặc dù quá trình này có thể được tối ưu bằng cách thực hiện kiểm tra tầm nhìn động hoặc sử dụng các đồ thị tầm nhìn (dynamic visibility check) giảm bớt, A\* trên Đồ thị Tầm nhìn vẫn được coi là chậm. Các thuật toán lập kế hoạch đường đi như Dijkstra liên tục và các biến thể của nó cũng như thuật toán Anya gần đây, cũng tìm ra đường đi ngắn nhất nhưng vẫn chưa được đánh giá kỹ lưỡng qua các thử nghiệm. Nói chung, để cân bằng giữa khoảng cách và tốc độ, một thuật toán tìm đường ở mọi góc độ nên được thiết kế để tìm đường đi ngắn hơn so với A\* và nhanh hơn so với A\* trên Đồ thị Tầm nhìn.

## 1.3. Phương hướng nghiên cứu

Trong bài báo này, một đồ thị tầm nhìn dựa trên các Đỉnh ứng cử viên (Candidate Vertices - CV) được phát triển, cùng với một kỹ thuật kiểm tra tầm nhìn sử dụng phương pháp phát tia (ray-casting). Phương pháp mới này giảm thiểu số lượng đánh giá cần thiết để tìm đường, so với các phương pháp sử dụng lưới hoặc đồ thị tầm nhìn (Visibility Graphs) mà không làm giảm tính tối ưu. Cấu trúc dữ liệu được gợi ý để đ giải quyết các vấn đề về hiệu suất trong việc áp dụng thuật toán A\* trên Đồ thị Tầm nhìn (Visibility Graphs). Hiệu suất của thuật toán mới, gọi là Focal Any-Angle A\* (FA-A\*), được so sánh với các phương pháp A\*, Theta\* và A\* trên Đồ thị Tầm nhìn thông qua các chỉ số như chiều dài đường đi, số lượng nút được xét và thời gian tính toán.

# CHƯƠNG 2: CÁC PHƯƠNG PHÁP, THUẬT TOÁN TIỀN ĐỀ

## 2.1. Đỉnh ứng cử viên (Candidate Vertices)

### 2.1.1. Tổng quan và khái niệm về Đỉnh ứng cử viên (Candidate Vertices)

Đồ thị Tầm nhìn (Visibility Graphs) của một bản đồ lưới bao gồm điểm khởi đầu, điểm kết thúc và tất cả các đỉnh nằm trên các vật cản. Lý do cần sử dụng đồ thị tầm nhìn (Visibility Graphs) là vì các đường đi ngắn nhất thực sự chỉ thay đổi hướng tại các đỉnh nằm trên các vật cản (nếu có). Tuy nhiên, các đỉnh mà đường đi ngắn nhất có thể đi qua chỉ là một tập con nhỏ của tất cả các đỉnh (Hình 2.1). Đây là những đỉnh mà chúng ta thực sự cần theo dõi.

Tại đây, một phương pháp để xác định các đỉnh mà đường đi tối ưu có thể đi qua. Tập hợp các đỉnh này được gọi là Đỉnh Ứng cử viên (Candidate Vertices - CV). Tất cả các Đỉnh Ứng cử viên cùng với nút bắt đầu, nút đích và các cạnh của chúng tạo thành đồ thị tầm nhìn cắt tỉa trọng tâm (focal pruned visibility graph). Phương pháp này được giải thích chi tiết dưới đây, bao gồm cả bước tiền xử lý một lần. Tập hợp CV được xây dựng động tại mỗi bước trong quá trình tìm kiếm.

A diagram of a diagram of a target

Description automatically generated

Hinh 2.1. Đồ thị tầm nhìn của hai nút có hai vật cản và đường đi ngắn nhất

### 2.1.2. Tiền xử lý (Preprocessing)

Đối với một bài toán có n vật cản, một ma trận đối xứng n x n được gọi là ma trận khoảng cách (proximity matrix) D được sử dụng để ghi lại khoảng cách từng cặp giữa các vật cản. Ví dụ, giá trị khoảng cách giữa vật cản thứ i và thứ j được xác định bởi khoảng cách giữa chúng nếu các vật cản là các hình vuông đơn vị:

****

Sau đó, các vật cản có giá trị khoảng cách nhỏ hơn √2 sẽ được gom nhóm (cluster) lại nếu cho phép di chuyển theo đường chéo giữa các vật cản. Nếu không, chúng ta gom nhóm các cặp có giá trị khoảng cách nhỏ hơn hoặc bằng √2 (Hình 2.2). Tại đây, phương pháp liên kết đơn (single linkage method) được áp dụng để gom nhóm các điểm vật cản.

A diagram of a square with a dotted line and a dotted line

Description automatically generated

Hinh 2.2. Di chuyển đường chéo giữa các vật cản

Tiếp theo, đối với mỗi cụm, thuật toán Quick Hull (Barber et al., 1995) được áp dụng để thu được các đỉnh trên vỏ bao lồi của mỗi cụm. Có thể xảy ra trường hợp khi các cụm không lồi được đặt gần nhau một cách chặt chẽ như răng cưa, con đường tối ưu có thể đi qua các đỉnh không nằm trên vỏ bao lồi của các cụm. Do đó, khuyến nghị nên xem xét tất cả các đỉnh ở các góc lồi của một cụm khi tỷ lệ vật cản lớn hoặc khi đang giải quyết các bài toán định tuyến mê cung. Tuy nhiên, đối với các bài toán tránh vật cản, các đỉnh trên vỏ bao lồi là đủ và do đó sẽ được sử dụng trong ba lần kiểm tra số học đầu tiên trong bài báo này. Các đỉnh ở các góc lồi sẽ được xem xét. Sau khi loại bỏ các đỉnh trên biên, các đỉnh còn lại sẽ là những đỉnh sẽ được sử dụng để xây dựng CV, và ta gọi nhóm các đỉnh này là V1. Một ví dụ về bài toán tìm đường sau khi xử lý trước có thể được thấy trong Hình 2.3(a), nơi có 36 đỉnh trong bộ các Đỉnh Ứng cử viên. Quá trình xử lý trước chỉ cần thực hiện một lần cho mỗi bản đồ dạng lưới bất kể vị trí của nút bắt đầu và nút đích. Mặt khác, các đồ thị tầm nhìn đầy đủ cần phải được xây dựng cho các thiết lập nút bắt đầu và nút đích khác nhau. Một ưu điểm của kỹ thuật xử lý trước này là, bất kể kích thước lưới nhỏ như thế nào hay có bao nhiêu vật cản được sử dụng để bao trùm hình dạng của một đặc điểm cản trở, nó luôn có thể được coi là một cụm thay vì một số lượng lớn các vật cản. Điều này giúp cải thiện hiệu suất tính toán của thuật toán khi bản đồ có càng nhiều chi tiết.

**Ưu điểm của kỹ thuật tiền xử lý này**

Bất kể kích thước lưới nhỏ đến đâu hoặc số lượng vật cản được sử dụng để mô tả chi tiết một đặc điểm chặn là bao nhiêu, chúng ta có thể được coi là một cụm duy nhất thay vì một số lượng lớn vật cản. Điều này tăng cường hiệu suất tính toán của thuật toán khi bản đồ ngày càng có nhiều chi tiết hơn.

### 2.1.3. Sinh Đỉnh Ứng Cử viên (Candidate Vertice)

A collage of images of different shapes

Description automatically generated

(a) Đỉnh trong V1 sau khi tiền xử lý

(b) Cụm vật cản và V2 (bên trong hộp)

(c) Đỉnh tương ứng với góc nhọn nhất

(d) Mở rộng tam giác

(e) Đỉnh trong V3

(f) Đỉnh Ứng Cử viên (được đánh dấu bằng vòng tròn chấm)

Hinh 2.3. Một ví dụ về việc tìm các đỉnh ứng viên

Sau khi tiền xử lý, các Đỉnh Ứng Cử viên (CV) sẽ được tạo ra liên tục trong suốt quá trình tìm kiếm. Trong mỗi bước, các cụm vật cản chặn đường thẳng giữa nút hiện tại và nút đích sẽ được xem xét đầu tiên. Nói cách khác, chúng ta sẽ kiểm tra các đỉnh trên bao lồi của các cụm vật cản chặn đường thẳng giữa nút hiện tại và nút đích.

Hình 2.3 minh họa các bước tìm các Đỉnh Ứng Cử viên trong FA-A\*. Như thể hiện trong Hình 2.3(b), Khu vực được đánh dấu chỉ ra các cụm vật cản và các đỉnh liên quan. Chúng ta gọi tập hợp các đỉnh này là V2, một tập con của V1. Kiểm tra tầm nhìn sẽ được thực hiện để xác định xem có cụm vật cản nào chặn đường thẳng giữa hai nút hay không. Kỹ thuật kiểm tra tầm nhìn sẽ được trình bày trong nội dung tiếp theo.

Tiếp theo, chúng ta tìm đỉnh trên mỗi bên của đường thẳng từ V2 tạo ra góc nhọn lớn nhất (α) với nút hiện tại và nút đích, như thể hiện trong Hình 2.3(c). Đỉnh trên mỗi bên tương ứng với góc nhọn lớn nhất được ký hiệu lần lượt là và . Đây là góc nhọn vì nếu một cụm vật cản trải dài từ đầu này sang đầu kia, sự chú ý của chúng ta vào mục tiêu sẽ bị hạn chế. Một cách ngầm định, các cụm gần nút hiện tại sẽ được ưu tiên hơn trong việc chứa  và .

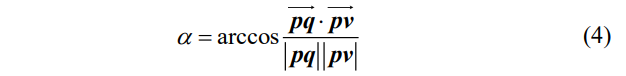
Giả sử nút hiện tại là p, nút đích là q, và đỉnh là v. Chúng ta sử dụng phương trình của đường thẳng để xác định điểm v nằm ở phía nào của đường thẳng.Đường thẳng được định nghĩa bởi p và q là,



Trong đó  , ,và 

Thay vị trí của đỉnh vào phương trình trên, ta có:

******

Vị trí của đỉnh v so với đường thẳng có thể được xác định bằng cách so sánh D với O. Sau đó, ta có thể tính góc α lần lượt cho các đỉnh ở mỗi bên, tức là:  


Chúng ta hiện có một tam giác được xác định bởi nút hiện tại,  và . Một trường hợp cực đoan là khi ,  đồng trục và do đó, không thể tạo thành một tam giác hay đa giác nào. Trong trường hợp đó, chỉ những đỉnh trong V1 nằm trên đoạn thẳng giữa  và  sẽ được xem xét. Thêm vào đó, nếu tất cả các đỉnh trong V2 đều nằm cùng một phía của đường thẳng,  và  sẽ tương ứng với các góc nhọn lớn nhất và nhỏ nhất.

Chúng ta cũng giới thiệu một kỹ thuật mở rộng tam giác bằng cách đẩy  và  ra khỏi trọng tâm o của tam giác.  và  đã được cập nhật có thể được tính toán như sau:

A black and white text

Description automatically generated with medium confidence

Trong đó,  là hệ số tỷ lệ. Giá trị mặc định của  là 1, tam giác vẫn giữ nguyên. Hình 2.3(d) cho thấy sự khác và giống nhau giữa ,  và ,  khi  là 2.

Các đỉnh trong V1 thuộc các cụm bên trong hoặc tiếp xúc với tam giác này được thu hẹp lại thành tập hợp các đỉnh V3 (Hình 2.3(e)). Kiểm tra xem một điểm có nằm trong đa giác hay không là một phép toán đơn giản. Cuối cùng, chúng tôi kiểm tra xem các đỉnh trong V3 có thể nhìn thấy từ nút hiện tại không; tất cả các đỉnh có thể nhìn thấy được gọi là Các Đỉnh Ứng Cử Viên (CV). Các CV, cùng với nút hiện tại, nút mục tiêu và các cạnh của chúng, tạo thành phiên bản rút gọn của đồ thị tầm nhìn (Hình 2.3(f)). Sau đó, các CV được sử dụng trong thuật toán A\* như các nút lân cận thay vì các nút liền kề với nút hiện tại.

A flowchart of a algorithm

Description automatically generated

Hinh 2.4. Lưu đồ thuật toán của việc tìm các Đỉnh Ứng Cử Viên

Một lưu đồ thuật toán được vẽ để minh họa quá trình xác định các Đỉnh Ứng Cử Viên một cách lặp đi lặp lại như một phần của thuật toán tìm đường trong Hình 2.4. Với các thủ tục đã được đề cập ở trên, chúng ta có thể giảm tải tính toán của A\* trên các Đồ Thị Tầm Nhìn thông qua việc kiểm tra tầm nhìn của các đỉnh trong V3 thay vì tất cả các đỉnh và chỉ tiến hành lan truyền đến các CV, trong khi vẫn duy trì nguyên tắc cơ bản của A\* trên Đồ Thị Tầm Nhìn

## 2.2. Kiểm tra tầm nhìn (VISIBILITY CHECK)

Trong Theta\*, một kỹ thuật kiểm tra tầm nhìn gọi là "line-of-sight" (tầm nhìn trực tiếp) được áp dụng, dựa trên nghiên cứu của. Ý tưởng của kiểm tra tầm nhìn trực tiếp rất đơn giản. Nó kiểm tra các ô vuông giữa hai nút dựa trên vị trí tương đối của chúng. Nếu ô vuông được kiểm tra là một vật cản, hai nút sẽ không thể thấy nhau. Kiểm tra tầm nhìn trực tiếp có thể được thực hiện một cách hiệu quả chỉ với các phép toán nguyên trên các lưới vuông. Tuy nhiên, kỹ thuật này lại khá đặc thù khi áp dụng vào các phân bố khác nhau hoặc các quy tắc tìm kiếm khác, vì các ô cần kiểm tra có thể thay đổi tùy theo từng trường hợp. Do đó, trong báo cáo này, chúng tôi đề xuất một thuật toán ray-casting để kiểm tra tầm nhìn giữa hai nút. Độ phức tạp tính toán của hai phương pháp này là (O(n)), nhưng phương pháp được sử dụng linh hoạt hơn và có thể đáp ứng được nhiều ứng dụng khác nhau một cách có hệ thống.

A crossword puzzle with different colored squares

Description automatically generated

Hinh 2.5. Mô tả các vật cản nằm giữa nút bắt đầu và nút mục tiêu

A diagram of a triangle with lines and a square

Description automatically generated

Hinh 2.6. Ví dụ kiểm tra khả năng hiển thị

Các bước kiểm tra khả năng nhìn thấy giữa hai nút khi có sự tồn tại của các vật cản được trình bày dưới đây:

1. ***Xem xét các vật cản xảy ra giữa các nút***

Quan sát Hình 2.5. Khi có một vật cản xảy ra giữa hai nút, các giá trị x, y của vật cản trong hệ tọa độ Descartes không được vượt quá các giá trị *x*, *y* tối đa hoặc tối thiểu của hai nút. Các vật cản được đánh dấu bằng dấu chéo trong Hình 2.5 cần được loại bỏ. Nếu hai nút có cùng giá trị *x* hoặc *y*, thì sự nhìn thấy có thể được xác định trực tiếp bằng cách kiểm tra các ô vuông giữa chúng.

1. ***Kiểm tra xem các vật cản có chặn tia giữa các nút hay không***

Chúng ta kiểm tra các vật cản bị giới hạn bởi các nút (Hình 2.5) lần lượt để xem liệu có vật cảnnào cắt đoạn thẳng giữa hai nút hay không. Lấy Hình 2.6 làm ví dụ. **p** và **q** là hai nút, và **Obs** là một vật cản đơn vị, trong đó các đỉnh có giá trị x, y nhỏ nhất và lớn nhất lần lượt là **vmin** và **vmax**.

Nếu

A black and white math equation

Description automatically generated with medium confidence

Và

A math equation with a square and square with a square and a square with a square and a square with a square and a square with a square and a square with a square and a square with

Description automatically generated

Sau đó, đoạn thẳng nối p và q đi qua vật cản(giao nhau). Trong các phương trình (6) và (7), ,là phép chiếu của trên trục x và trục y, *txmin*,tymin là phép chiếu của  trên trục x và trục y, và *txmax*,tymax là phép chiếu của trên trục x và trục y, như được chỉ ra trong Hình 2.6. Nếu chúng ta thay dấu ‘<’ thành ‘≤’ trong các phương trình (6) và (7), chỉ cần đi qua một đỉnh cũng sẽ được coi là giao nhau.

Khi một vật cản được tính toán là đã giao với đoạn thẳng giữa hai nút, chúng ta có thể kết thúc quá trình kiểm tra và kết luận rằng hai nút không thể nhìn thấy nhau

# Chương 3: FOCAL ANY-ANGLE A\* trên đồ thị tầm nhìn

## 2.1. Thuật toán FA-A\*

Chúng ta đề xuất phương pháp giảm kích thước đồ thị tầm nhìn và kiểm tra khả năng quan sát, gọi là FA-A\* (Focal Any-Angle A\*). Mã giả của FA-A\* như sau:

* **"n"** là nút; nstart, ntarget, ncurrent lần lượt là nút bắt đầu, nút mục tiêu và nút hiện tại.
* **g(n)**: độ dài đường đi ngắn nhất từ nút bắt đầu (**nstart**) đến nút **n** đã tìm thấy.
* **c(n1, n2)**: chi phí di chuyển từ **n1** đến **n2**, được tính bằng khoảng cách Euclid giữa **n1** và **n2**.
* **h(n)**: chi phí ước lượng từ nút **n** đến nút mục tiêu (**ntarget**). Trong nghiên cứu này, chi phí ước lượng được xấp xỉ bằng khoảng cách Euclid giữa hai nút.

Main()

open = PriorityQueue() // Open list to hold nodes to explore

g(nstart) = 0 // Cost to reach the start node is 0

parent(nstart) = nstart // The start node is its own parent initially

open.Insert(nstart, parent(nstart), g(nstart), h(nstart))

ncurrent = nstart

While open is not empty:

propagate = CV of ncurrent

if ncurrent is ntarget:

propagate

open.Insert(ntarget, ncurrent, g(ncurrent) + c(ncurrent, ntarget), 0)

return "Path found"

end if

open.Refresh(propagate, ncurrent)

open.Close(ncurrent) // Mark the current node as explored

ncurrent = the node in open with the smallest g + h

End while

return "Path not found"

End Main

open.Refresh(propagate, ncurrent)

for every neighbor n' of ncurrent:

propagate

g(n')new = g(ncurrent) + c(ncurrent, n') // Calculate tentative g value

if n' exists in open:

if g(n')new < g(n') in open:

open.Update(n', ncurrent, g(n')new, h(n'))

end if

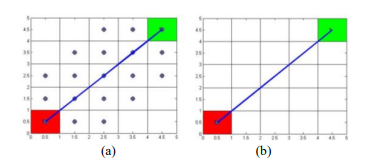
else:

open.Insert(n', ncurrent, g(n')new, h(n'))

end if

end for

End Refresh



Hinh 3.1. Minh họa so sánh đánh giá nút giữa A\* trên lưới và FA-A\*

Sự khác biệt chính giữa FA-A\* và A\* là FA-A\* chỉ lan truyền đến các Đỉnh ứng cử viên (CV) của nút hiện tại thay vì tất cả các nút lân cận (vui lòng xem dòng được tô xám trong mã giả). Xét một trường hợp đơn giản không có chướng ngại vật. A\* hoặc Theta\* cần đánh giá 17 nút trong quá trình tìm kiếm để tìm được nút mục tiêu (Hình 3.1(a)), trong khi FA-A\* có thể tìm được nút mục tiêu trực tiếp từ nút bắt đầu vì nút mục tiêu là một trong các Đỉnh ứng cử viên của nút bắt đầu (Hình 3.1(b)). Các nút đã được đánh giá được đánh dấu bằng chấm trong Hình 3.1. Lưu ý rằng trong mã giả, chúng ta chỉ duy trì một danh sách mở, trong khi trong A\* và Theta\*, danh sách đóng đồng thời tồn tại với danh sách mở. Sự thay đổi này được thực hiện bằng cách sử dụng thao tác open.Close để ngăn nút hiện tại trở thành nút hiện tại hoặc xuất hiện lại trong CV trong tương lai. Sự thay đổi này có thể giúp cải thiện hiệu suất của FA-A\* về mặt hiệu quả.

## 2.2. Tính tối ưu

Quá trình lựa chọn các Đỉnh ứng cử viên được đề xuất có thể được coi là một cách tiếp cận tham lam để giảm số lần kiểm tra khả năng nhìn thấy và đánh giá hàm so với các phương pháp sử dụng đồ thị tầm nhìn đầy đủ. Thuật toán được lấy cảm hứng từ các trường hợp khi hai nút được nối bằng dây cao su vì các kết nối dây cao su tối ưu luôn có khoảng cách ngắn nhất có thể và sự thay đổi hướng chủ yếu xảy ra tại hoặc giữa các đỉnh thuộc các cụm đóng vai trò lớn nhất trong việc cản trở hai nút. Vai trò của các chướng ngại vật trong việc cản trở hai nút là một khái niệm trừu tượng được định lượng trong nghiên cứu của chúng tôi là α. Sau đó, va và vb tương ứng với tầm với lớn nhất của các cụm cản trở ở mỗi bên. Bằng cách xác định V3, chúng tôi nhằm xác định tất cả các đỉnh mà đường đi tối ưu có thể đi qua.

Mặc dù thuật toán tìm được các đường đi tốt hơn A\* và Theta\*, tính tối ưu không phải lúc nào cũng được đảm bảo trong một số trường hợp rất rời rạc. Điều này sẽ được minh họa trong các bài kiểm tra số A và B. Tuy nhiên, phương pháp được đề xuất tìm các đường đi rất gần với đường đi ngắn nhất. Đó là lý do chúng tôi giới thiệu hệ số tỷ lệ w để mở rộng tam giác, tức là bao gồm nhiều đỉnh hơn trong V3 với chi phí hiệu suất tính toán kém hơn. Ban đầu, chúng tôi cố gắng giảm số đỉnh được xem xét so với đồ thị tầm nhìn đầy đủ. Việc tăng giá trị của w sẽ dần làm giảm hiệu ứng giảm số lượng này và cho phép thuật toán tìm đường đi ngắn nhất trong các tình huống cụ thể. Như kết quả kiểm tra số trong Mục V cho thấy, trong hầu hết các trường hợp, giá trị mặc định 1 cho w đủ tốt để thuật toán tìm đường đi ngắn nhất.

## 2.3. Phân tích độ phức tạp thuật toán

Phân tích độ phức tạp của FA-A\*. Các hoạt động của FA-A\* và độ phức tạp trong trường hợp xấu nhất được mô tả như sau:

1. **Tiền xử lý**: O(∣Obs∣log∣Obs∣), đây là độ phức tạp của thuật toán quick hull. Ở đây, ∣Obs∣ là số lượng chướng ngại vật.
2. **Kiểm tra cụm nào cản trở đường thẳng giữa nút bắt đầu và nút hiện tại**: O(n). O(n) là độ phức tạp của kiểm tra khả năng nhìn thấy, trong đó nnn là số chướng ngại vật giữa hai nút.
3. **Tính toán α**: O(∣V2∣). ∣V2∣ là kích thước của tập hợp V2
4. **Kiểm tra xem một đỉnh có nằm trong tam giác hay không**: O(3∣V1∣).O(3) là độ phức tạp của việc kiểm tra xem một điểm có nằm trong tam giác hay không.
5. **Kiểm tra xem các đỉnh trong V3 có nhìn thấy được từ nút hiện tại hay không**: O(n∣V3∣)

Tóm lại, độ phức tạp tính toán của FA-A\* là:



Với ExpanFA-A∗​ là số lượng lần mở rộng trong FA-A\*. Công thức này có thể được đơn giản hóa thành:



Thuật ngữ đầu tiên trong phương trình trên bị lược bỏ vì thuật ngữ thứ hai có tác động chi phối đến độ phức tạp khi kích thước bài toán tăng lên.

Trong khi đó, độ phức tạp của các thuật toán khác như sau:

* Độ phức tạp của A\*:

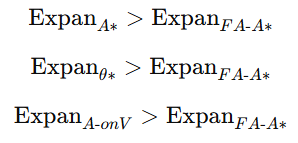


* Độ phức tạp của Theta\*:



* Độ phức tạp của A\* trên đồ thị tầm nhìn: 

Số lượng mở rộng của mỗi thuật toán có mối quan hệ thực nghiệm như sau:



Vì VVV là tập hợp của tất cả các đỉnh, ta có:



FA-A\* có thể được coi là một phiên bản nhẹ của A\* trên đồ thị tầm nhìn về mặt độ phức tạp tính toán, dựa trên các phương trình trên. Vì kích thước của V1V1V1, V2V2V2, và V3V3V3 tỷ lệ thuận với số lượng cụm, FA-A\* hiệu quả trong việc giải quyết các bài toán có số lượng cụm chướng ngại vật đáng kể hơn là các chướng ngại vật nhỏ lẻ.

Kết quả là, FA-A\* có lợi thế so với các phương pháp lan truyền đến các nút lân cận trực tiếp như A\* và Theta\* về mặt tính toán khi xử lý các bản đồ có số lượng cụm chướng ngại vật nhỏ. Trong khi đó, FA-A\* luôn hiệu quả hơn A\* trên đồ thị tầm nhìn. Tất cả những điều này khiến FA-A\* hữu ích trong việc giải quyết các vấn đề tránh chướng ngại vật hoặc định tuyến mê cung.

## 2.4. Cấu trúc dữ liệu

Trong nghiên cứu này, một cấu trúc dữ liệu gọn nhẹ được sử dụng với hai ma trận chính:

1. **Ma trận Vall**: Là ma trận có kích thước ∣V∣×4 trong đó ∣V∣ là tổng số đỉnh. Mỗi dòng của Vall đại diện cho một đỉnh trong tập hợp V, với:
   * Hai cột đầu tiên lưu trữ tọa độ của đỉnh trong hệ trục tọa độ.
   * Cột thứ ba chứa chỉ số chướng ngại vật mà đỉnh thuộc về.
   * Cột cuối cùng chứa chỉ số cụm mà chướng ngại vật thuộc về.
2. **Ma trận Vconvex**: Là ma trận có kích thước ∣V1∣×4 trong đó mỗi dòng tương ứng với một đỉnh trong tập V1

Ma trận Vall được sử dụng để xác định tập V1 trong giai đoạn tiền xử lý, và ma trận Vconvex được tạo ra dựa trên đó. Sau khi sử dụng Vconvex để xác định V2, Vall được dùng để kiểm tra xem một cụm có nằm trong hoặc tiếp tuyến với tam giác hay không. Cuối cùng, Vconvex được sử dụng để xác định tập V3

# Chương 4: Kiểm tra số liệu

## 4.1. Định tuyến mê cung

Trong bài kiểm tra này, 100 mê cung có kích thước 50\*50 được sinh ngẫu nhiên được sử dụng, với nút bắt đầu ở góc trên bên trái và nút mục tiêu ở góc dưới bên phải. Với tiêu chí đánh giá chủ yếu là thời gian và số lượng nút nằm trên đường đi ngắn nhất mà thuật toán tìm được.

Hệ số ước lược được sử dụng trong các thí nghiệm là khoảng cách Euclid từ nút hiện tại đến nút mục tiêu.

Tất cả các thuật toán được triển khai trên **Python 3.11.7** và chạy trên laptop có bộ xử lý **Intel(R) Core™ i5-4219M** tốc độ **2.60 GHz**.

Đầu tiên, các thuật toán được so sánh trên các mê cung 50x50. Các mê cung được khởi tạo với thuật toán **Randomized Prim's Algorithm** với đặc điểm:

* Tạo mê cung có đường đi liên thông: Không có vùng bị cô lập.
* Có xu hướng tạo nhiều nút giao (ngã tư), ít đường cụt hơn so với thuật toán "Recursive Backtracking".
* Dễ mở rộng: Có thể áp dụng cho mê cung có kích thước lớn

Dưới đây là bảng thống kê số liệu thông qua thuật toán

Bảng 1: Bảng thống kê thời gian trung bình

|  |  |  |  |
| --- | --- | --- | --- |
| **Map Info** | **Avg(Times)** | **Algorithm** | **Avg(Time)** |
| 50x50 | 10 | A\* | 1.81 |
| FA-A\* | 2.05 |
| 30 | A\* | 2.17 |
| FA-A\* | 2.48 |
| 50 | A\* | 2.15 |
| FA-A\* | 2.60 |
| 100 | A\* | 2.14 |
| FA-A\* | 2.61 |

Hinh 4.1. Bảng thống kê thời gian chạy trung bình của 2 thuật toán

Bảng 2: Bảng thống kê số ô trung bình trên đường đi ngắn nhất

|  |  |  |  |
| --- | --- | --- | --- |
| **Map Info** | **Avg(Times)** | **Algorithm** | **Avg(NoC)** |
| 50x50 | 10 | A\* | 113.90 |
| FA-A\* | 68.70 |
| 30 | A\* | 111.47 |
| FA-A\* | 70.70 |
| 50 | A\* | 111.80 |
| FA-A\* | 70.10 |
| 100 | A\* | 112.33 |
| FA-A\* | 71.01 |

Hinh 4.2. Bảng thống kê số ô trên đường đi ngắn nhất trung bình

Như … trên thì FA-A\* luôn tìm được đường đi ngắn nhất vượt trội hơn đường đi mà thuật toán A\* truyền thống có thể tìm được. Hơn nữa với mê cung kích thước 50x50 được tạo bằng thuật toán Randomized Prim's Algorithm thì số ô nằm trên đường đi ngắn nhất của FA-A\* ít hơn đáng kể, cụ thể là khoảng 35% đến 40% so với thuật toán A\*.

## 4.2. Tìm đường đi có trọng số nhỏ nhất trên đồ thị trọng số

Tiếp theo, tôi áp dụng thuật toán cho đồ thị trọng số 50\*50 với giá trị trọng số trong khoảng [1,100]. Nút bắt đầu được đặt ở góc trên bên trái và và nút mục tiêu được đặt ở góc dưới bên phải.

Tiêu chí so sánh giữa 2 thuật toán FA-A\* và A \* ở đây là giá trị trọng số đường đi. Kết quả thu được được lưu ở Bảng 3 và Hình 4.3 dưới đây

Bảng 3: Bảng so sánh kết quả trọng số trung bình của 2 thuật toán

|  |  |  |  |
| --- | --- | --- | --- |
| **Map Info** | **Times** | **Algorithm** | **Weight Average** |
| 50x50 | 10 | A\* | 2748.00 |
| FA-A\* | 1416.00 |
| 30 | A\* | 2722.73 |
| FA-A\* | 1404.27 |
| 50 | A\* | 2729.58 |
| FA-A\* | 1406.30 |
| 100 | A\* | 2747.46 |
| FA-A\* | 1408.64 |

Hinh 4.3. Bảng thống kê trọng số trung bình của 2 thuật toán

Thông qua thống kê, chúng ta có thể nhìn thấy 1 cách đơn giản rằng với đồ thị trọng số 50x50 thì tổng trọng số trên đường đi của thuật toán FA-A\* luôn vượt trội hơn so với thuật toán A\* với giá trị gần như là gấp đôi.

### 4.3. Định tuyến mê cung với đường đi được gắn trọng số

Ở hạng mục này, tôi đánh giá độ hiệu quả thuật toán thông qua mê cung với đường đi được gán thêm các giá trị trọng số để tìm đường ra trong mê cung có trọng số nhỏ nhất. Cũng với kích thước là 50x50 của mê cung thì tiêu chí chính để đánh giá trong hạng mục này

Rõ ràng, các phương pháp FA-A\* vượt trội hơn so với các phương pháp A\* truyền thống thông qua giá trị được hiển thị trên Bảng 4 và Hình 4.4 dưới đây.

Bảng 4: Bảng so sánh kết quả trọng số trung bình của 2 thuật toán

|  |  |  |  |
| --- | --- | --- | --- |
| **Map Info** | **Avg(Times)** | **Algorithm** | **Avg(Weight)** |
| 50x50 | 10 | A\* | 5138.50 |
| FA-A\* | 3306.40 |
| 30 | A\* | 5202.37 |
| FA-A\* | 3445.47 |
| 50 | A\* | 5188.84 |
| FA-A\* | 3468.14 |
| 100 | A\* | 5143.12 |
| FA-A\* | 3478.26 |

Hinh 4.4. Bảng thống kê trọng số trung bình của 2 thuật toán

# Chương 5: cài đặt thuật toán

## 5.1. Công cụ sử dụng

### 5.1.1. Ngôn ngữ lập trình python

Python là ngôn ngữ lập trình phù hợp để triển khai hai chương trình tạo mê cung và tìm đường trong mê cung, nhờ vào cú pháp đơn giản và dễ đọc. Các thuật toán phức tạp, như tạo mê cung hoặc FA-A\*, có thể được biểu diễn ngắn gọn và rõ ràng trong Python, giúp lập trình viên nhanh chóng xây dựng và kiểm tra chương trình. Python cung cấp các thư viện tiêu chuẩn mạnh mẽ như random, os, time, heapq, và math, giúp giảm bớt công sức khi lập trình các chức năng cơ bản.

Python cũng vượt trội nhờ khả năng hỗ trợ đồ họa và trực quan hóa. Thư viện Pillow giúp tạo và lưu hình ảnh mê cung một cách dễ dàng, trong khi Pygame cung cấp công cụ trực quan hóa quá trình tìm đường. Việc hiển thị đồ họa sinh động này không chỉ tăng cường khả năng kiểm tra thuật toán mà còn mang lại trải nghiệm sinh động cho người dùng.

Ngoài ra, Python quản lý bộ nhớ tự động, giảm nguy cơ lỗi liên quan đến phân bổ hoặc giải phóng bộ nhớ như trong C/C++. Với sự hỗ trợ từ thư viện NumPy, Python xử lý hiệu quả các thao tác dữ liệu mảng, phù hợp với cấu trúc mê cung dưới dạng ma trận. Cấu trúc dữ liệu linh hoạt của Python cũng giúp dễ dàng mở rộng kích thước mê cung mà không cần định trước kích thước cụ thể.

Python là lựa chọn tuyệt vời để phát triển nhanh các nguyên mẫu (prototyping) nhờ khả năng viết mã gọn gàng và đa nền tảng. Chương trình viết bằng Python có thể chạy trên Windows, macOS, hoặc Linux mà không cần thay đổi mã nguồn. Ngoài ra, Python dễ dàng tích hợp với các ngôn ngữ khác như C/C++ hoặc sử dụng công cụ tăng tốc hiệu năng (như cython hoặc numba) nếu cần tối ưu thời gian chạy.

Mặc dù các ngôn ngữ khác như C++, Java hay Matlab cũng có thể được sử dụng, chúng gặp nhiều hạn chế so với Python. C++ yêu cầu quản lý bộ nhớ thủ công và việc hiển thị đồ họa phức tạp hơn khi sử dụng các thư viện như SDL hoặc OpenGL. Java có cú pháp dài dòng và không cung cấp các công cụ gọn nhẹ như Pygame. Matlab tuy mạnh trong xử lý ma trận, nhưng không phù hợp để tạo đồ họa tương tác hoặc đa dụng như Python.

### 5.1.2. Các thư viện hỗ trợ trong ngôn ngữ lập trình python

Thư viện **random** được chọn trong chương trình tạo mê cung vì tính đơn giản và hiệu quả trong việc tạo ra các số ngẫu nhiên, điều này rất quan trọng trong việc xác định các điểm bắt đầu hoặc các vị trí cần xử lý trong mê cung. Việc tạo ra các số ngẫu nhiên giúp thuật toán thêm tường vào mê cung một cách ngẫu nhiên và xây dựng các cấu trúc mê cung phức tạp mà không cần phải xác định thủ công các điểm. Thư viện này rất nhẹ và dễ sử dụng, là một công cụ lý tưởng cho các tác vụ ngẫu nhiên trong thuật toán.

Thư viện **os** được sử dụng để quản lý các thư mục và tệp tin trong quá trình lưu trữ mê cung và các kết quả. Việc lưu trữ các hình ảnh mê cung hoặc các tệp văn bản đòi hỏi phải tạo và kiểm tra sự tồn tại của các thư mục. Thư viện này hỗ trợ thao tác với các hệ thống tệp một cách linh hoạt, giúp đảm bảo các thư mục được tạo đúng cách và tránh lỗi khi lưu dữ liệu đầu ra vào các tệp tin hoặc thư mục khác nhau trên các nền tảng hệ điều hành.

Thư viện **Pillow** (PIL) được chọn để tạo và lưu hình ảnh mê cung nhờ khả năng hỗ trợ nhiều định dạng ảnh, đặc biệt là PNG. Việc trực quan hóa mê cung dưới dạng hình ảnh giúp người dùng dễ dàng theo dõi và kiểm tra kết quả của thuật toán. **Pillow** cung cấp các công cụ mạnh mẽ để vẽ các hình chữ nhật (tường và đường đi) với màu sắc tùy chỉnh, giúp tạo ra hình ảnh mê cung một cách chính xác và dễ hiểu.

Thư viện **numpy** là công cụ lý tưởng cho việc xử lý ma trận trong chương trình giải mê cung. Mê cung được đại diện dưới dạng ma trận 2D, và **numpy** cho phép thao tác nhanh chóng và hiệu quả với các phần tử trong ma trận. Việc sử dụng **numpy** giúp tối ưu hóa các phép toán trên ma trận như tìm kiếm các tế bào hợp lệ, tính toán các giá trị liên quan đến gói thông tin trong thuật toán A\* và FA-A\*, giúp cải thiện hiệu suất xử lý khi làm việc với dữ liệu có kích thước lớn.

Thư viện **heapq** được sử dụng trong thuật toán A\* và FA-A\* để duy trì hàng đợi ưu tiên, giúp tìm kiếm và xử lý các nút có giá trị thấp nhất. Đây là một thành phần quan trọng trong cả hai thuật toán, vì hàng đợi ưu tiên giúp xác định lựa chọn tối ưu cho từng bước đi tiếp theo. **heapq** hỗ trợ thao tác thêm và truy xuất phần tử với độ phức tạp O(log n), giúp thuật toán chạy nhanh hơn và đảm bảo tính tối ưu trong quá trình tìm kiếm.

Thư viện **math** đóng vai trò quan trọng trong việc tính toán các phép toán hình học trong thuật toán FA-A\*. Cụ thể, **math** cung cấp các hàm cần thiết để tính toán khoảng cách Euclidean và góc giữa các điểm, là những yếu tố quan trọng trong việc xác định hướng di chuyển tối ưu. Với các phép toán này, thuật toán có thể tính toán chính xác chi phí di chuyển từ điểm này đến điểm khác và tìm ra con đường ngắn nhất.

Cuối cùng, thư viện **pygame** được lựa chọn để xây dựng giao diện đồ họa và trực quan hóa quá trình giải mê cung. **pygame** cung cấp các công cụ vẽ đồ họa mạnh mẽ, hỗ trợ việc vẽ các hình ảnh và cập nhật màn hình trong thời gian thực, giúp người dùng dễ dàng theo dõi tiến trình của thuật toán. Thư viện này cũng giúp tạo hiệu ứng trực quan, như làm nổi bật các điểm, các ô tường, hoặc đường đi, mang lại trải nghiệm trực quan sinh động khi theo dõi thuật toán A\* và FA-A\*.

## 5.2. Sinh mê cung với 1.0.maze\_generator.py

Chức năng chính của Module này là tạo ra một mê cung ngẫu nhiên với kích thước đã được chỉ định.

Sau đó, một mê cung được khởi tạo dưới dạng ma trận, với tất cả các ô là tường (được ký hiệu bằng số 1) và một ô ngẫu nhiên được chọn làm điểm bắt đầu và được gán giá trị 0, đại diện cho lối đi.

*maze = [[1 for \_ in range(width)] for \_ in range(height)]*

*start\_x,start\_y = random.randrange(1,width,2), random.randrange(1,height,2)*

*maze[start\_y][start\_x] = 0*

Tiếp đến, các bức tường ban đầu được khởi tạo. Mỗi tường được xác định bằng cách tạo ra các cặp tọa độ của các tường xung quanh ô bắt đầu và đưa vào danh sách walls

*walls = [(start\_x + dx, start\_y + dy, start\_x + 2 \* dx, start\_y + 2 \* dy)*

*for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]*

*if 0 <= start\_x + 2 \* dx < width and 0 <= start\_y + 2 \* dy < height]*

Tiếp theo, các tường xung quanh ô bắt đầu được đưa vào danh sách, và thuật toán phá tường sẽ được áp dụng để tạo ra các lối đi, đảm bảo rằng các ô lối đi được nối với nhau mà không bị chồng lấn. Quá trình tiếp tục cho đến khi không còn tường nào có thể phá vỡ, đảm bảo rằng tất cả các ô trong mê cung đã được kết nối với nhau bằng các lối đi

*while walls:*

*wall\_x, wall\_y, nx, ny = random.choice(walls)*

*walls.remove((wall\_x, wall\_y, nx, ny))*

*if maze[ny][nx] == 1:*

*maze[wall\_y][wall\_x] = 0*

*maze[ny][nx] = 0*

*for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:*

*next\_wall\_x, next\_wall\_y = nx + dx, ny + dy*

*if (0 <= nx + 2 \* dx < width and 0 <= ny + 2 \* dy < height*

*and maze[ny + dy][nx + dx] == 1):*

*walls.append((nx + dx, ny + dy, nx + 2 \* dx, ny + 2 \* dy))*

## 5.3. Trình diễn định tuyến mê cung với 1.1.FA\_A\_Star.py

### 5.3.1. Cung cấp các thư viện thông qua import

*import pygame*

*import numpy as np*

*import heapq*

*import time*

*from math import sqrt, atan2, degrees*

*import os*

### 5.3.2. Phần khởi tạo và cấu hình chương trình

Khởi tạo môi trường giao diện pygame. Cấu hình hằng số gồm xác định kích thước ô (cell) và kích thước cửa sổ hiển thị cũng như các màu sắc sử dụng trong giao diện đồ họa.

*pygame.init()*

*CELL\_SIZE = 12*

*WINDOW\_SIZE = None*

*WHITE = (255, 255, 255)*

*BLACK = (0 , 0 , 0 )*

*RED = (255, 0 , 0 )*

*GREEN = (0 , 255, 0 )*

*BLUE = (0 , 0 , 255)*

*YELLOW = (255, 255 , 0)*

### 5.3.3. Lớp Node (Đại diện cho một nút trong thuật toán)

Mỗi đối tượng của lớp Node đại diện cho một ô trong mê cung, bao gồm các thông tin như tọa độ (position), chi phí (g, h, f), và hướng di chuyển từ nút cha (direction).

*class Node:*

*def \_\_init\_\_(self, position, g, h, direction):*

*self.position = position # (x, y)*

*self.g = g # cost from start to this node*

*self.h = h # heuristic cost to target*

*self.f = g + h # total cost*

*self.direction = direction*

*def \_\_lt\_\_(self, other):*

*return self.f < other.f*

### 5.3.4. Hàm Heuristic (Tính toán khoảng cách heuristic)

Hàm này tính toán khoảng cách Euclid giữa hai điểm a và b, sử dụng cho thuật toán FA-A\*

def heuristic(a, b):

return sqrt((a[0] - b[0]) \*\* 2 + (a[1] - b[1]) \*\* 2)

### 5.3.5. Hàm lấy các ô lân cận (Get Neighbors)

Hàm này tìm các ô xung quanh một ô cho trước (cả 4 hướng chính và 4 hướng chéo), kiểm tra xem ô đó có phải là tường không và có nằm trong phạm vi của mê cung không.

*def get\_neighbors(position, maze):*

*neighbors = []*

*x, y = position*

*for dx,dy in [(-1,0),(1,0),(0,-1),(0,1),(-1,-1),(1,1),(-1,1),(1,-1)]:*

*nx, ny = x + dx, y + dy*

*if 0 <=nx< maze.shape[0] and 0 <=ny< maze.shape[1] and maze[nx][ny]==0:*

*neighbors.append((nx, ny))*

*return neighbors*

### 5.3.6. Hàm tính toán góc quay giữa hai điểm (Calculate Angle)

Hàm này tính toán góc giữa hai điểm bằng cách sử dụng hàm *atan2* và chuyển đổi kết quả từ radian sang độ

*def calculate\_angle(from\_node, to\_node):*

*dx, dy = to\_node[0] - from\_node[0], to\_node[1] - from\_node[1]*

*return degrees(atan2(dy, dx))*

### 5.3.7. Thuật toán FA-A

Hàm này triển khai thuật toán FA-A\*

*def focal\_any\_angle\_a\_star(maze, start, target, screen):*

*open\_set = []*

*start\_node = Node(start, 0, heuristic(start, target), None)*

*heapq.heappush(open\_set, start\_node)*

*came\_from = {}*

*g\_score = {start: 0}*

*f\_score = {start: heuristic(start, target)}*

*closed\_set = set()*

*path = []*

*while open\_set:*

*current = heapq.heappop(open\_set)*

*screen.fill(BLACK)*

*draw\_maze(maze, path, screen, open\_set, closed\_set, current.position)*

*pygame.display.flip()*

*time.sleep(0.0001)*

*if current.position == target:*

*while current.position in came\_from:*

*path.append(current.position)*

*current = came\_from[current.position]*

*path.append(start)*

*path.reverse()*

*screen.fill(BLACK)*

*draw\_maze(maze,path,screen,open\_set,closed\_set,current.position)*

*pygame.display.flip()*

*time.sleep(0.05)*

*return path*

*closed\_set.add(current.position)*

*neighbors = get\_neighbors(current.position, maze)*

*for neighbor in neighbors:*

*if neighbor in closed\_set:*

*continue*

*tentative\_g\_score=g\_score[current.position]+ heuristic(current.position, neighbor)*

*if neighbor not in g\_score or tentative\_g\_score < g\_score[neighbor]:*

*angle = calculate\_angle(current.position, neighbor)*

*came\_from[neighbor] = Node(current.position, g\_score[current.position], f\_score[current.position], angle)*

*g\_score[neighbor] = tentative\_g\_score*

*f\_score[neighbor] = tentative\_g\_score + heuristic(neighbor, target)*

*if not any(neighbor == node.position for node in open\_set):*

*heapq.heappush(open\_set, Node(neighbor, tentative\_g\_score, heuristic(neighbor, target), angle))*

*return None*

### 5.3.8. Hàm vẽ mê cung và quá trình tìm kiếm (Draw Maze)

Hàm này vẽ mê cung và các trạng thái của thuật toán như open\_set, closed\_set, current\_position, và đường đi (nếu có)

*def draw\_maze(maze, path, screen, open\_set, closed\_set, current\_position):*

*for x in range(maze.shape[0]):*

*for y in range(maze.shape[1]):*

*rect = pygame.Rect(y \* CELL\_SIZE, x \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)*

*if maze[x, y] == 1:*

*pygame.draw.rect(screen, RED, rect) # Obstacle (red)*

*else:*

*pygame.draw.rect(screen, WHITE, rect) # Free space (white)*

*pygame.draw.rect(screen, BLACK, rect, 1) # Draw grid lines*

*if current\_position:*

*x, y = current\_position*

*rect = pygame.Rect(y \* CELL\_SIZE, x \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)*

*pygame.draw.rect(screen, GREEN, rect) # Current node (blue)*

*for node in open\_set:*

*x, y = node.position*

*rect = pygame.Rect(y \* CELL\_SIZE, x \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)*

*pygame.draw.rect(screen, YELLOW, rect) # Open set (yellow)*

*for pos in closed\_set:*

*x, y = pos*

*rect = pygame.Rect(y \* CELL\_SIZE, x \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)*

*pygame.draw.rect(screen, BLACK, rect) # Closed set (black)*

*for pos in path:*

*x, y = pos*

*rect = pygame.Rect(y \* CELL\_SIZE, x \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE)*

*pygame.draw.rect(screen, BLUE, rect) # Path (green)*

### 5.3.9. Hàm tải mê cung từ file (Load Maze from File)

Hàm này đọc dữ liệu từ tệp văn bản và chuyển thành mảng numpy để xử lý

*def load\_maze\_from\_file(filename):*

*with open(filename, 'r') as file:*

*lines = file.readlines()*

*maze = np.array([[int(cell) for cell in line.strip().split()] for line in lines])*

*return maze*

### 5.3.10. Quá trình hiển thị và lưu kết quả

Sau khi thuật toán hoàn thành, hiển thị thời gian chạy và kết quả đường đi trên cửa sổ Pygame

*maze = load\_maze\_from\_file("maze.txt")*

*WINDOW\_SIZE = (maze.shape[1] \* CELL\_SIZE, maze.shape[0] \* CELL\_SIZE + 50)*

*# Starting and target positions*

*start = (0, 0) # Top-left corner*

*target = (maze.shape[0] - 1, maze.shape[1] - 1) # Bottom-right corner*

*screen = pygame.display.set\_mode(WINDOW\_SIZE)*

*pygame.display.set\_caption("FA-A\* Pathfinding Visualization")*

*start\_time = time.time()*

*path = focal\_any\_angle\_a\_star(maze, start, target, screen)*

*end\_time = time.time()*

*elapsed\_time = end\_time - start\_time*

*font = pygame.font.SysFont(None, 36)*

*text = font.render(f"Time: {elapsed\_time:.4f} seconds", True, WHITE)*

*screen.blit(text, (10, maze.shape[0] \* CELL\_SIZE + 10))*

*pygame.display.flip()*

*pygame.time.wait(2000)*

*pygame.quit()*

## 5.4. Sinh ma trận trọng số với 2.0.matrix\_generator.py

Chức năng chính của Module này là tạo ra một ma trận trọng số ngẫu nhiên từ 1 đến 100 với kích thước đã được chỉ định thông qua 1 hàm đơn giản.

def create\_grid (n, min\_weight=1, max\_weight=101):

return [[random.randint(min\_weight,max\_weight) for\_in range(n)] for\_in range(n)]

## 5.5. Trình diễn tìm đường đi ngắn nhất với 2.1.FA\_A\_Star.py

## 5.5.1 Cấu hình các thông số mặc định

Phần cấu hình các thông số mặc định của chương trình như kích thước cửa sổ, kích thước cell, số lượng cell và các màu sắc để trình diễn

n = 50

cell\_size = 13

width, height = n \* cell\_size, n \* cell\_size + 50

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

GRAY = (200, 200, 200)

GREEN = (0, 255, 0)

RED = (255, 0, 0)

BLUE = (0, 0, 255)

YELLOW = (255, 255, 0)

## 5.5.2 Hỗ trợ tìm kiếm

Gồm hàm Heuristic và Kiểm tra tầm nhìn (line of sight):

def heuristic(node, goal):

dx, dy = abs(node[0] - goal[0]), abs(node[1] - goal[1])

return math.sqrt(dx \*\* 2 + dy \*\* 2)

def line\_of\_sight(grid, start, end):

x0, y0 = start

x1, y1 = end

dx, dy = abs(x1 - x0), abs(y1 - y0)

sx, sy = (1 if x1 > x0 else -1), (1 if y1 > y0 else -1)

err = dx - dy

while (x0, y0) != (x1, y1):

if grid[x0][y0] == 0:

return False

e2 = 2 \* err

if e2 > -dy:

err -= dy

x0 += sx

if e2 < dx:

err += dx

y0 += sy

return True

Kiểm tra xem có thể di chuyển trực tiếp từ start đến end mà không bị vật cản hay không. Và dùng thuật toán **Bresenham's Line Algorithm** để kiểm tra từng ô trên đường đi. Nếu gặp ô có trọng số **0** (vật cản), trả về False.

## 5.5.3 Thuật toán FA-A\* trong tìm kiếm đường đi ngắn nhất

Hàm cốt lõi quan trọng nhất của chương trình thực hiện chức năng tìm kiếm đường đi ngắn nhất trong ma trận trọng số

def fa\_a\_star(grid, start, goal, screen):

size = len(grid)

open\_set = []

g\_score = {start: 0}

f\_score = {start: heuristic(start, goal)}

open\_set.append((f\_score[start], start))

came\_from = {}

closed\_set = set()

while open\_set:

open\_set.sort()

\_, current = open\_set.pop(0)

draw\_grid(screen, grid, open\_set, closed\_set, start, goal, path=None)

pygame.display.flip()

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1], g\_score[goal]

closed\_set.add(current)

x, y = current

neighbors = [

(x + dx, y + dy)

for dx, dy in [(-1,0),(1,0),(0,-1),(0,1),(-1,-1),(-1,1),(1,-1),(1,1)]

if 0 <= x + dx < size and 0 <= y + dy < size

]

for neighbor in neighbors:

if line\_of\_sight(grid, current, neighbor) and neighbor not in closed\_set:

nx, ny = neighbor

tentative\_g\_score = g\_score[current] + grid[nx][ny]

if neighbor not in g\_score or tentative\_g\_score < g\_score[neighbor]:

g\_score[neighbor] = tentative\_g\_score

f\_score[neighbor] = tentative\_g\_score + heuristic(neighbor, goal)

open\_set.append((f\_score[neighbor], neighbor))

came\_from[neighbor] = current

return None, None

## 5.5.4 Hiển thị nhằm trực quan hóa cho người xem

Dùng **pygame** để hiển thị ma trận và các trạng thái:

* **closed\_set**: Xám
* **open\_set**: Vàng
* **path**: Xanh
* **start**: Xanh lá
* **goal**: Đỏ

def draw\_grid(screen,grid,open\_set,closed\_set,start,goal,path,total\_weight=None):

screen.fill(WHITE)

font = pygame.font.Font(None, 14)

for i in range(n):

for j in range(n):

rect = pygame.Rect(j \* cell\_size, i \* cell\_size, cell\_size, cell\_size)

pygame.draw.rect(screen, BLACK, rect, 1)

if closed\_set:

for (x, y) in closed\_set:

pygame.draw.rect(screen, GRAY, pygame.Rect(y \* cell\_size, x \* cell\_size, cell\_size, cell\_size))

if open\_set:

for \_, (x, y) in open\_set:

pygame.draw.rect(screen, YELLOW, pygame.Rect(y \* cell\_size, x \* cell\_size, cell\_size, cell\_size))

if path:

for (x, y) in path:

pygame.draw.rect(screen,BLUE, pygame.Rect(y \* cell\_size, x \* cell\_size, cell\_size, cell\_size))

pygame.draw.rect(screen, BLACK, pygame.Rect(y \* cell\_size, x \* cell\_size, cell\_size, cell\_size), 1)

if start:

pygame.draw.rect(screen, GREEN, pygame.Rect(start[1] \* cell\_size, start[0] \* cell\_size, cell\_size, cell\_size))

if goal:

pygame.draw.rect(screen, RED, pygame.Rect(goal[1] \* cell\_size, goal[0] \* cell\_size, cell\_size, cell\_size))

for i in range(n):

for j in range(n):

text = font.render(str(grid[i][j]), True, BLACK)

screen.blit(text, text.get\_rect(center=(j \* cell\_size + cell\_size // 2, i \* cell\_size + cell\_size // 2)))

if total\_weight is not None:

text = font.render(f"Total Weight: {total\_weight}", True, BLACK)

screen.blit(text, text.get\_rect(center=(width // 2, height - 25)))

## 5.5.5 Hàm main

Hàm tổng hợp thực hiện các chức năng trên:

def main():

pygame.init()

screen = pygame.display.set\_mode((width, height))

pygame.display.set\_caption("FA-A\* Pathfinding")

grid = load\_matrix\_from\_file("matrix.txt")

start, goal = (0, 0), (n - 1, n - 1)

path, total\_weight = fa\_a\_star(grid, start, goal, screen)

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

if path:

draw\_grid(screen,grid,path=path,start=start,goal=goal, total\_weight=total\_weight)

else:

draw\_grid(screen, grid, start=start, goal=goal)

pygame.display.flip()

output\_dir = os.path.join("RESULT", "Weight")

os.makedirs(output\_dir, exist\_ok=True)

pygame.image.save(screen, os.path.join(output\_dir, "FA\_A\_Star.png"))

pygame.quit()

if \_\_name\_\_ == "\_\_main\_\_":

main()

## 5.6. Sinh mê cung trọng số với 3.0.matrix\_maze\_generator.py

Chức năng chính của này là tạo ra một mê cung ngẫu nhiên với kích thước đã được chỉ định kèm thêm là trọng số trong đường đi với thuật toán Randomized Prim's Algorithm thông qua 1 hàm cơ bản .

def create\_maze(width, height, min\_weight=1, max\_weight=99):

maze = np.full((height, width), 999, dtype=int)

start\_x, start\_y = 1, 1

maze[start\_y, start\_x] = random.randint(min\_weight, max\_weight)

frontier = [(start\_x + dx, start\_y + dy, start\_x, start\_y) for dx, dy in DIRECTIONS if 1 <= start\_x + dx < width - 1 and 1 <= start\_y + dy < height - 1]

while frontier:

rand\_index = random.randint(0, len(frontier) - 1)

x, y, px, py = frontier.pop(rand\_index)

if maze[y, x] == 999:

maze[y, x] = random.randint(min\_weight, max\_weight)

maze[(y + py) // 2, (x + px) // 2] = random.randint(min\_weight, max\_weight)

frontier.extend([(x + dx, y + dy, x, y) for dx, dy in DIRECTIONS if 1 <= x + dx < width - 1 and 1 <= y + dy < height - 1 and maze[y + dy, x + dx] == 999])

return maze

Hàm **create\_maze(width, height, min\_weight=1, max\_weight=99)** được thiết kế để tạo một mê cung ngẫu nhiên sử dụng thuật toán Randomized Prim's Algorithm, trong đó các ô đường đi được gán trọng số ngẫu nhiên.

Trước tiên, hàm khởi tạo một ma trận có kích thước **height x width**, trong đó tất cả các ô đều được gán giá trị **999**, tượng trưng cho tường của mê cung. Sau đó, điểm xuất phát tại tọa độ (1,1) được chọn và gán một trọng số ngẫu nhiên trong khoảng từ **min\_weight** đến **max\_weight**. Các ô lân cận của điểm bắt đầu cũng được thêm vào danh sách "mặt trận" – đây là các ô có thể được mở để tạo đường đi.

Tiếp theo, hàm liên tục chọn một ô ngẫu nhiên từ danh sách "mặt trận". Nếu ô đó vẫn chưa được mở (tức là vẫn mang giá trị **999**), nó sẽ được gán một trọng số ngẫu nhiên và đường đi sẽ được mở bằng cách cập nhật ô trung gian giữa ô mới và ô trước đó. Sau khi mở đường, các ô lân cận của ô mới lại được thêm vào danh sách "mặt trận", tiếp tục quá trình mở rộng mê cung.

Quá trình này tiếp diễn cho đến khi không còn ô nào trong danh sách "mặt trận", đảm bảo rằng toàn bộ mê cung được tạo ra với các đường đi kết nối hợp lý. Kết quả cuối cùng là một ma trận số nguyên, trong đó **999** đại diện cho tường, còn các số khác là trọng số của các ô đường đi.

## 5.7. Trình diễn định tuyến mê cung với 3.1.FA\_A\_Star.py

## 5.7.1. Cấu hình các thông số mặc định

Phần cấu hình các thông số mặc định của chương trình như kích thước cửa sổ, kích thước cell, số lượng cell và các màu sắc để trình diễn

n = 51

cell\_size = 13

width, height = n \* cell\_size, n \* cell\_size + 50

# Màu sắc

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

GRAY = (200, 200, 200)

GREEN = (0, 255, 0)

RED = (255, 0, 0)

BLUE = (0, 0, 255)

YELLOW = (255, 255, 0)

## 5.7.2. Hỗ trợ tìm kiếm

Hàm heuristic(node, goal) tính khoảng cách Euclidean giữa một ô hiện tại và ô đích, được dùng để đánh giá chi phí di chuyển trong thuật toán FA-A\*.

def heuristic(node, goal):

dx, dy = abs(node[0] - goal[0]), abs(node[1] - goal[1])

return math.sqrt(dx \*\* 2 + dy \*\* 2)

## 5.7.3. Thuật toán FA-A\* trong định tuyến mê cung có trọng số

def fa\_a\_star(grid, start, goal, screen):

size = len(grid)

open\_set = []

g\_score = {start: 0}

f\_score = {start: heuristic(start, goal)}

open\_set.append((f\_score[start], start))

came\_from = {}

closed\_set = set()

while open\_set:

open\_set.sort()

\_, current = open\_set.pop(0)

draw\_grid(screen, grid, open\_set, closed\_set, start, goal, path=None)

pygame.display.flip()

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1], g\_score[goal]

closed\_set.add(current)

x, y = current

neighbors = [

(x + dx, y + dy)

for dx, dy in [(-1,0),(1,0),(0,-1),(0,1),(-1,-1),(-1,1),(1,-1),(1,1)]

if 0 <= x + dx < size and 0 <= y + dy < size and grid[x + dx][y + dy] != 999

]

for neighbor in neighbors:

nx, ny = neighbor

tentative\_g\_score = g\_score[current] + grid[nx][ny]

if neighbor not in g\_score or tentative\_g\_score < g\_score[neighbor]:

g\_score[neighbor] = tentative\_g\_score

f\_score[neighbor] = tentative\_g\_score + heuristic(neighbor, goal)

open\_set.append((f\_score[neighbor], neighbor))

came\_from[neighbor] = current

return None, None

## 5.7.4. Hiển thị nhằm trực quan hóa cho người xem

Dùng pygame để hiển thị ma trận và các trạng thái:

• closed\_set: Xám

• open\_set: Vàng

• path: Xanh

• start: Xanh lá

• goal: Đỏ

def draw\_grid(screen,grid,open\_set,closed\_set,start,goal,path,total\_weight):

screen.fill(WHITE)

font = pygame.font.Font(None, 14)

for i in range(n):

for j in range(n):

rect = pygame.Rect(j \* cell\_size, i \* cell\_size, cell\_size, cell\_size)

if grid[i][j] == 999: # Ô chặn

pygame.draw.rect(screen, BLACK, rect)

pygame.draw.rect(screen, BLACK, rect, 1)

if closed\_set:

for (x, y) in closed\_set:

rect = pygame.Rect(y \* cell\_size, x \* cell\_size, cell\_size, cell\_size)

pygame.draw.rect(screen, GRAY, rect)

if open\_set:

for \_, (x, y) in open\_set:

rect = pygame.Rect(y \* cell\_size, x \* cell\_size, cell\_size, cell\_size)

pygame.draw.rect(screen, YELLOW, rect)

if path:

for (x, y) in path:

rect = pygame.Rect(y \* cell\_size, x \* cell\_size, cell\_size, cell\_size)

pygame.draw.rect(screen, BLUE, rect)

pygame.draw.rect(screen, BLACK, rect, 1)

if start:

rect = pygame.Rect(start[1] \* cell\_size, start[0] \* cell\_size, cell\_size, cell\_size)

pygame.draw.rect(screen, GREEN, rect)

pygame.draw.rect(screen, BLACK, rect, 1)

if goal:

rect = pygame.Rect(goal[1] \* cell\_size, goal[0] \* cell\_size, cell\_size, cell\_size)

pygame.draw.rect(screen, RED, rect)

pygame.draw.rect(screen, BLACK, rect, 1)

for i in range(n):

for j in range(n):

if grid[i][j] != 999:

rect = pygame.Rect(j \* cell\_size, i \* cell\_size, cell\_size, cell\_size)

weight = grid[i][j]

text = font.render(str(weight), True, BLACK)

text\_rect = text.get\_rect(center=rect.center)

screen.blit(text, text\_rect)

if total\_weight is not None:

text = font.render(f"Total Weight: {total\_weight}", True, BLACK)

text\_rect = text.get\_rect(center=(width // 2, height - 25))

screen.blit(text, text\_rect)

## 5.7.5. Hàm main

Hàm tổng hợp thực hiện các chức năng trên:

def main():

pygame.init()

screen = pygame.display.set\_mode((width, height))

pygame.display.set\_caption("FA-A\* Pathfinding")

running = True

grid = load\_matrix\_from\_file("matrix\_maze.txt")

start, goal = (0, 0), (n - 1, n - 1)

path, total\_weight = fa\_a\_star(grid, start, goal, screen)

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

if path:

draw\_grid(screen,grid,path=path,start=start,goal=goal, total\_weight=total\_weight)

else:

draw\_grid(screen, grid, start=start, goal=goal)

pygame.display.flip()

output\_dir = os.path.join("RESULT", "Weight\_Maze")

os.makedirs(output\_dir, exist\_ok=True)

output\_path = os.path.join(output\_dir, "FA\_A\_Star.png")

pygame.image.save(screen, output\_path)

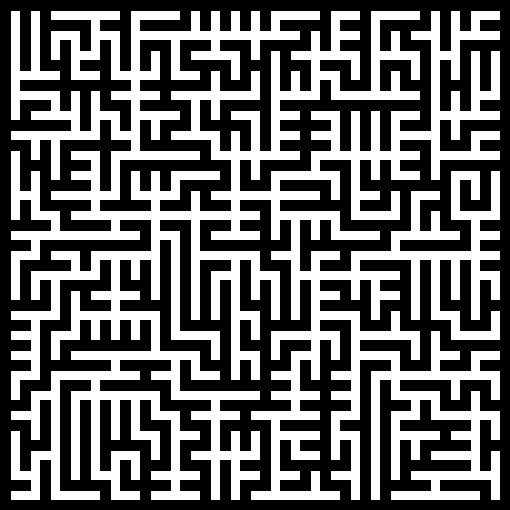
pygame.quit()

if \_\_name\_\_ == "\_\_main\_\_":

main()

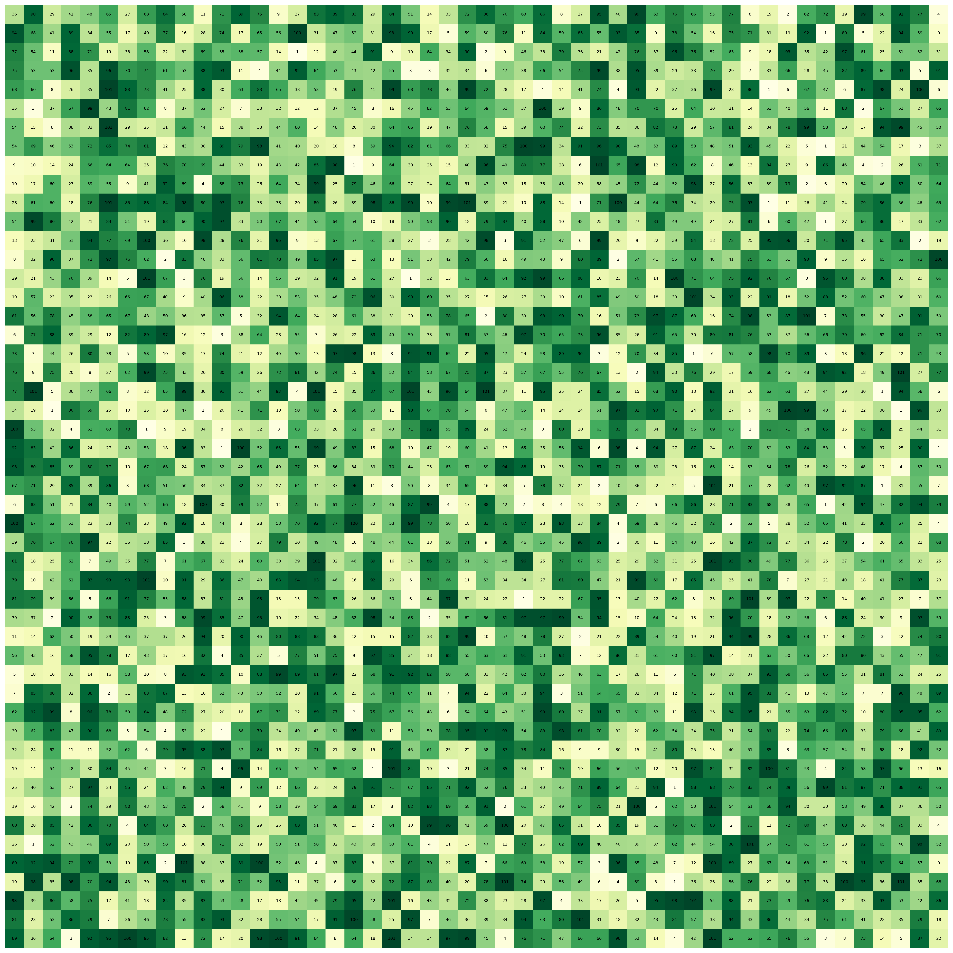
## 5.8. Kết quả thu được

### 5.8.1. Kết quả thu được từ 1.0.maze\_generator.py



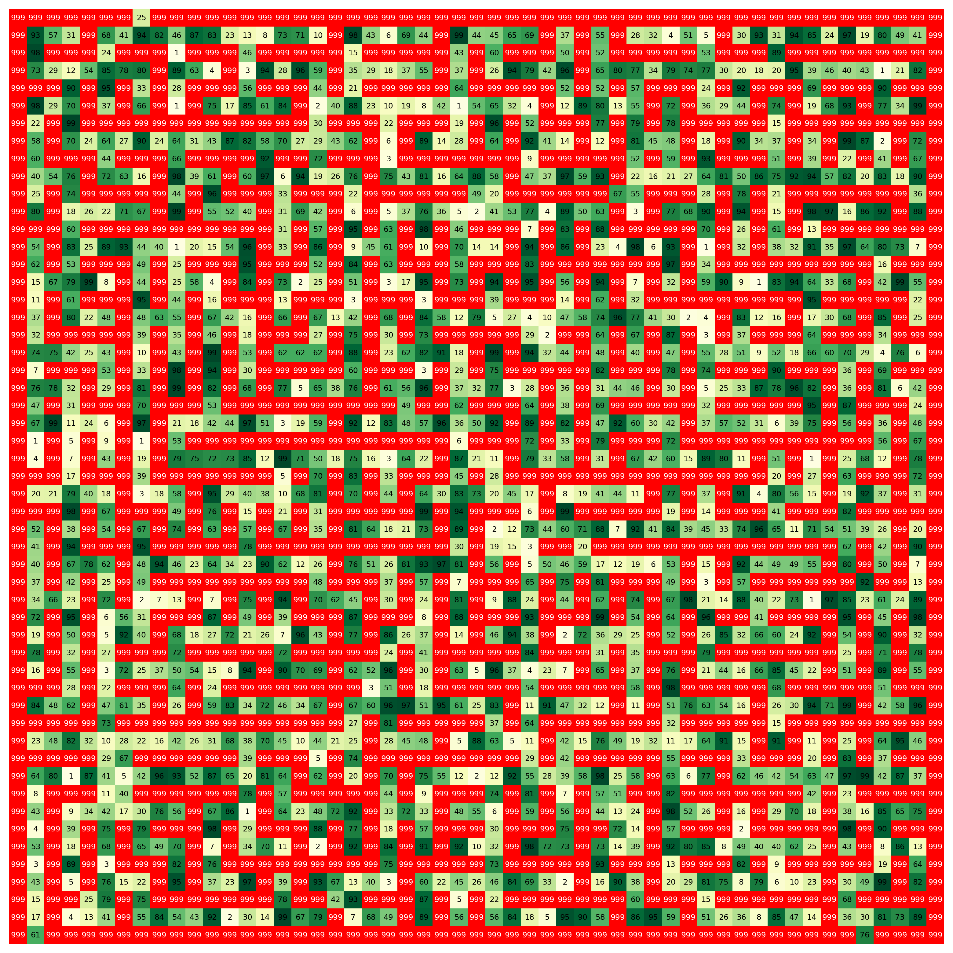
Hinh 5.1. Kết quả thu được từ 1.0.maze\_generator.py

### 5.8.2. Kết quả thu được từ 2.0.matrix\_generator.py



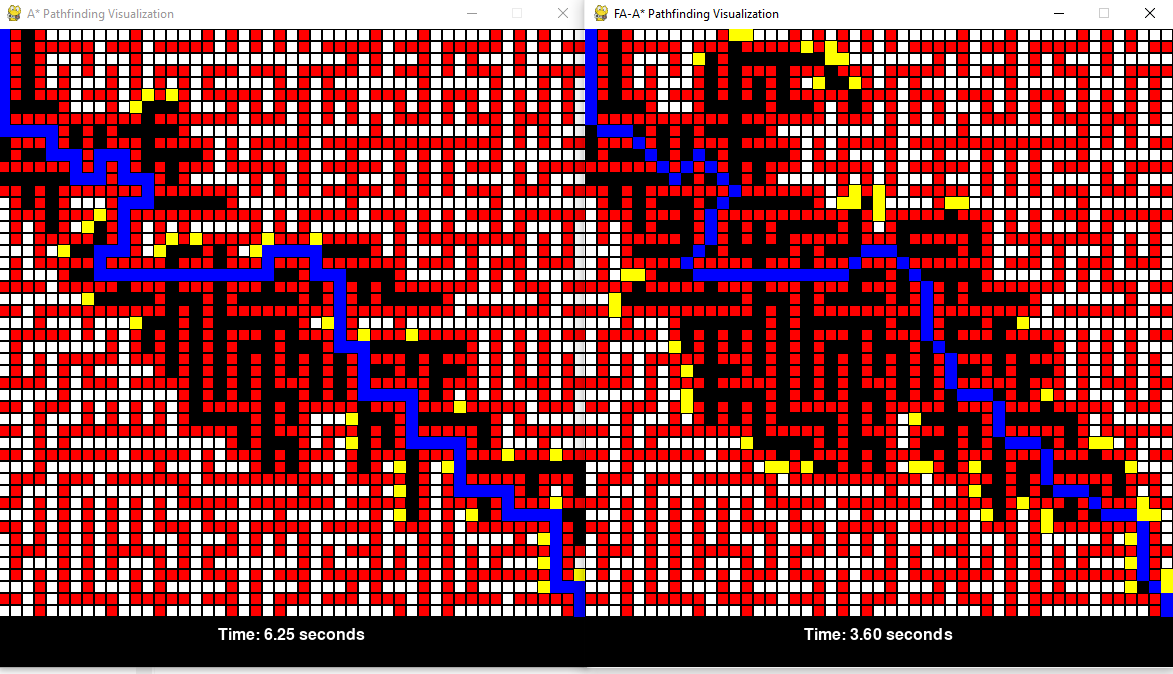
Hinh 5.2. Kết quả thu được từ 2.0.matrix\_generator.py

### 5.8.3. Kết quả thu được từ 3.0.matrix\_maze\_generator.py



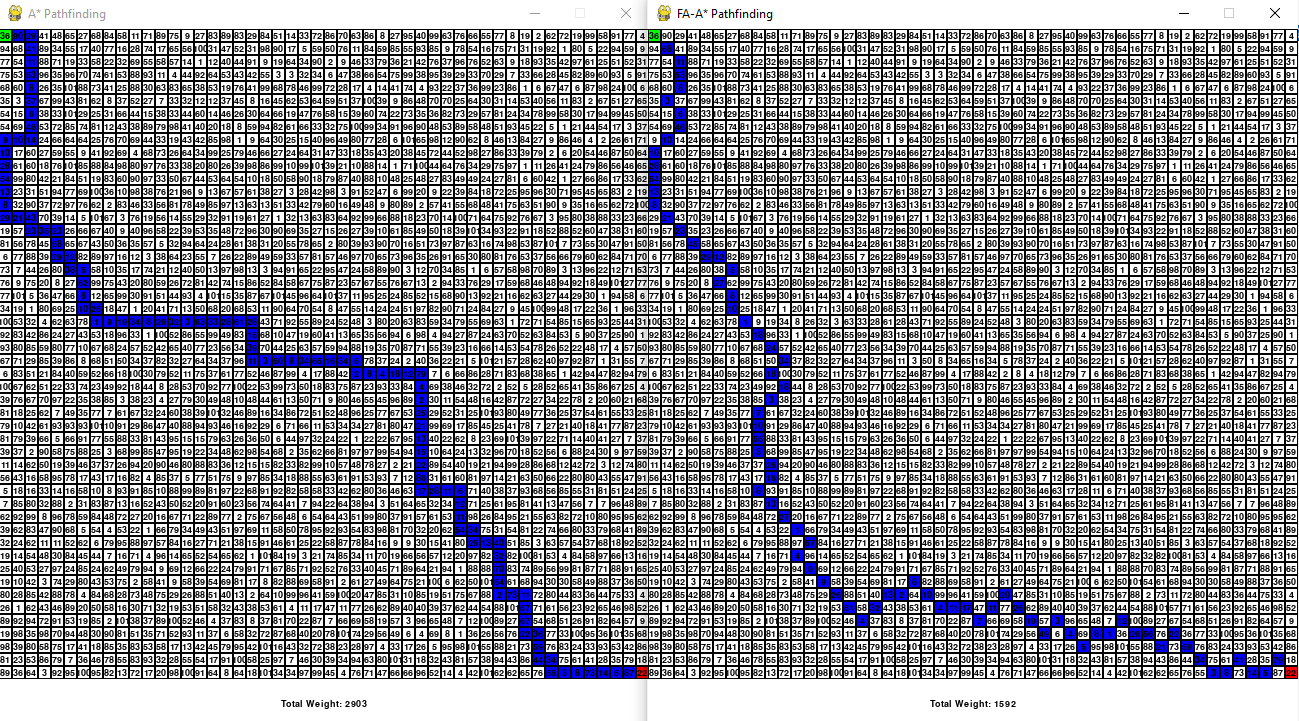
Hinh 5.3. Kết quả thu được từ 3.0.matrix\_maze\_generator.py

### 5.8.4. Kết quả thu được từ thuật toán A\* và FA\* trong định tuyến mê cung



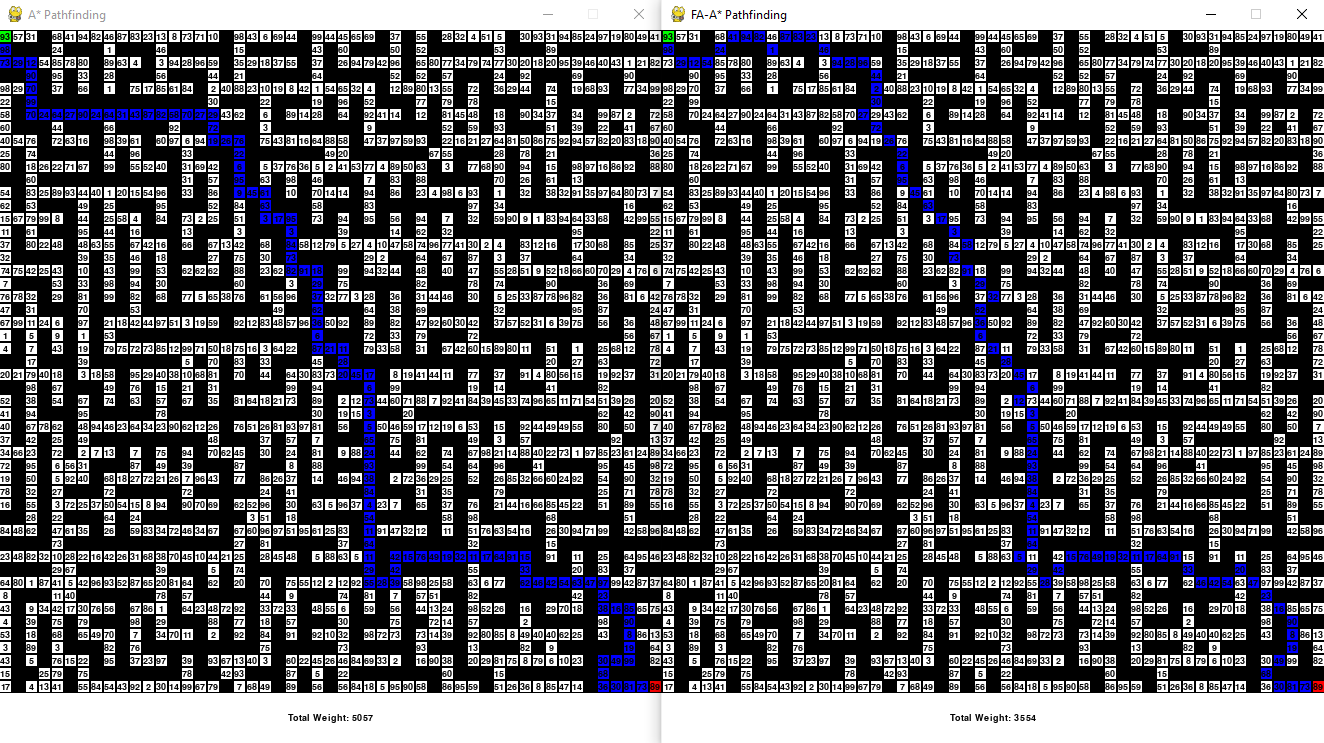
Hinh 5.4. Kết quả thu được từ thuật toán A\* và FA\* trong định tuyến mê cung

### 5.8.5. Kết quả thu được từ thuật toán A\* và FA\* trong tìm đường đi ngắn nhất



Hinh 5.5. Kết quả thu được từ thuật toán A\* và FA\* trong tìm đường đi ngắn nhất

### 5.8.6. Kết quả thu được từ thuật toán A\* và FA\* trong định tuyến mê cung có trọng số



Hinh 5.6. Kết quả thu được từ thuật toán A\* và FA\* trong định tuyến mê cung trọng số

# Kết luận

Trong nghiên cứu này, chúng tôi phát triển thuật toán A\* tiêu điểm góc bất kỳ dựa trên đồ thị tầm nhìn (FA-A\*). FA-A\* thực hiện việc tìm đường tập trung vào vị trí của mục tiêu và cắt bớt đồ thị tầm nhìn đầy đủ cho phù hợp. Thuật toán tăng cường hiệu suất tính toán của A\* trên đồ thị tầm nhìn như được thể hiện trong các thử nghiệm số. Trong khi đó, nó luôn tìm thấy những đường dẫn tốt hơn A\*. Hơn nữa, FA-A\* có ít đánh giá nút nhất cho tất cả các trường hợp thử nghiệm. Mặc dù trong bài viết này, tất cả chi phí đường đi được đánh giá là khoảng cách Euclide, nhưng đối với các vấn đề có chi phí đường đi phức tạp hơn, FA-A\* sẽ đạt được lợi thế đáng kể hơn về mặt thời gian tính toán. FA-A\* không chỉ có thể duy trì tính tối ưu của các phương pháp dựa trên đồ thị tầm nhìn mà còn có thể theo kịp các phương pháp dựa trên lưới về mặt tính toán. Do đó, phương pháp mới được phát triển có khả năng tìm đường tốt hơn A\* đồng thời mang lại hiệu quả tính toán tốt hơn A\* trên đồ thị tầm nhìn. Thuật toán có thể được sử dụng trong nhiều ứng dụng khác nhau, ví dụ: để tìm đường đi tối ưu trong môi trường tĩnh cho robot đi, giải quyết các vấn đề tránh chướng ngại vật cho phương tiện hoặc tìm kết nối tốt nhất trong các thành phần, ….

TÀI LIỆU THAM KHẢO

[1]. Pei Cao, Zhaoyan Fan, Robert X. Gao, and J. Tang, “A Focal Any-Angle Path-finding Algorithm Based on A\* on Visibility Graphs”, 2017

[2] K. Daniel, A. Nash, S. Koenig, and A. Felner, “Theta\*: Any-angle path planning on grids,” Journal of Artificial Intelligence Research, vol. 39, pp. 533-579, 2010.

[3] A. Nash, K. Daniel, S. Koenig and A. Felner, “Theta\*: any-angle path planning on grids,” in Proc. National Conference on Artificial Intelligence, vol. 22, Menlo Park, CA; Cambridge, MA, London, AAAI Press, MIT Press, 1999, 2007

PHỤ LỤC

Để tham khảo mã nguồn chi tiết của mô phỏng thuật toán FA-A\* cho bài toán vui lòng truy cập trang GitHub dưới đây:

* Focal Any-Angle A*\*:* <https://github.com/PeDrKi/Artificial_Intelligence_PDK>

Trang GitHub này chứa mã nguồn đầy đủ của thuật toán FA-A\* bao gồm các tệp Python để triển khai thuật toán, các ví dụ và các tài liệu hướng dẫn sử dụng.

Dưới đây là một phần mã nguồn của thuật toán FA-A\* trong bài toàn định tuyến mê cung:

*def focal\_any\_angle\_a\_star(maze, start, target, screen):*

*open\_set = []*

*start\_node = Node(start, 0, heuristic(start, target), None)*

*heapq.heappush(open\_set, start\_node)*

*came\_from = {}*

*g\_score = {start: 0}*

*f\_score = {start: heuristic(start, target)}*

*closed\_set = set()*

*path = []*

*while open\_set:*

*current = heapq.heappop(open\_set)*

*if current.position == target:*

*while current.position in came\_from:*

*path.append(current.position)*

*current = came\_from[current.position]*

*path.append(start)*

*path.reverse()*

*return path*

*closed\_set.add(current.position)*

*neighbors = get\_neighbors(current.position, maze)*

*for neighbor in neighbors:*

*if neighbor in closed\_set:*

*continue*

*...*

*return None*