**TRƯỜNG ĐẠI HỌC QUẢN LÝ VÀ CÔNG NGHỆ**

THÀNH PHỐ HỒ CHÍ MINH



**ĐỒ ÁN MÔN HỌC**

**GIỚI THIỆU VỀ TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI**: *Tìm đường thoát thông minh trong trò chơi mê cung*

**Giảng viên**: Nguyễn Quản Bá Hồng

**Họ và tên sinh viên**: Hấu Trung Thành

**Mã sinh viên**: 2302700234

**Mã học phần**: BIT115V1

**MỤC LỤC**

1. LỜI MỞ ĐẦU
2. Lý do chọn đề tài
3. Mục tiêu và cách giải quyết
4. TỔNG QUAN VỀ CÔNG NGHỆ
5. Ngôn ngữ và thư viện
6. Định dạng đầu vào - ảnh mê cung
7. THIẾT LẬP GIẢI PHÁP
8. Mô hình mê cung 2D dưới dạng lưới (grid)
9. Thuật toán A\* và heuristic Manhattan
10. Cấu trúc dữ liệu chính
11. CÀI ĐẶT CHI TIẾT
12. Tải ảnh và chuyển sang ma trận nhị phân (phần 1 của code)
13. Xác định Start và Goal (phần 2 của code)
14. Cài đặt hàm astar(phần 3)
    1. Hàm Heuristic
    2. Khởi tạo open head và g\_score
    3. Vòng lặp chính: pop head, kiểm tra goal, mở động đỉnh
    4. Reconstruct path từ came\_from
15. Nén đường dẫn thành các đoạn liên kết – hàm compress (phần 4)
16. Chuyển pixel thành ô và in kết quả
17. Vẽ đường đi, đánh dấu Start, Goal (Phần 6)
18. KẾT QUẢ THỬ NGHIỆM
19. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Phần I: LỜI MỞ ĐẦU

1. **Lý do chọn đề tài:**

Việc tìm đường đi trong mê cung là một bài toán cổ điển nhưng vẫn giữ được tính ứng dụng cao trong nhiều linh vực như robot tự hành, trò chơi, hệ thống hỗ trợ định vị trong những không gian phức tạp. Cá nhân tôi cũng hứng thú với các trò chơi giải mê cung: Việc tự động hóa quá trình tìm đường đi không chỉ là một thử thách cho tôi mà còn thỏa mãn đam mê giải đố, giúp tôi hiểu rõ hơn về các yếu tố đằng sau bước di chuyển, từ đó có thể tối ưu hơn trong việc trải nghiệm trò chơi giải đố của bản thân.

1. **Mục tiêu và cách giải quyết:**

**Mục tiêu:** Tìm đường đi ngắn nhất từ điểm bắt đầu đến điểm kết thúc của một mê cung 2D cho trước.

**Cách giải quyết:**

* **Tiền xử lý:** 
  + Dùng PIL chuyển ảnh mê cung sang ảnh xám.
  + Chuyển thành mảng NumPy và đặt điều kiện (arr > 128) để xét đâu là tường đâu là đường đi.
  + Tính kích thước pixel trên mỗi ô (pixel\_per\_cell) dựa theo độ phân giải của ảnh và số hàng, số cột theo yêu cầu.
* **Xác định Start, Goal:**
  + Quét hàng đầu (y=0) để tìm đường vào, được hiểu là điểm bắt đầu.
  + Quét hàng cuối (y=h-1) để tìm đường ra, được hiểu là điểm kết thúc.
  + Chuyển sang tọa độ (x, y) cho điểm bắt đầu và kết thúc để dễ quan sát.
* **Tìm đường đi bằng A\*:**
  + Khởi tạo open head (f, g, node) với f = g + h
  + Khởi tạo g\_score[start] = 0 và close = set().
  + Vòng lặp chính: pop node có f nhỏ nhất, mở rộng 4 hướng (lên, xuống, trái, phải), cập nhật g\_score, came\_from và đẩy vào open\_h.
  + Khi gặp Goal, dựng lại đường đi bằng bản đồ came\_from.
* **Nén đường đi & chuyển đổi:**
  + Hàm compress(path) gom các bước pixel liên tiếp cùng hướng thành đoạn (direction, pixel\_lenght).
  + Chuyển mỗi pixel\_length sang số ô (cel\_length) bằng công thức:

|  |  |
| --- | --- |
| px\_col = w/NUM\_COLS | px\_row = h/NUM\_ROWS |

Pixel\_per\_cell = (px\_col + px\_row) / 2

* **Trực quan hóa:** 
  + Quy mỗi cell thành điểm tâm trong ảnh gốc.
  + Dùng matplotlib vẽ nền mê cung gốc, đường đi màu đỏ mịn và marker Start/Goal.

Phần II: TỔNG QUAN VỀ CÔNG NGHỆ

1. **Ngôn ngữ và thư viện:**

Trong dự án này, em sử dụng Python làm ngôn ngữ lập trình chính bởi tính linh hoạt. Cụ thể các thư viện sau được áp dụng:

* **Pillow**: Thư viện mở rộng của PIL (Python Imaging Library) để đọc, ghi và chuyển đổi ảnh. Chức năng sử dụng: ***Image.open(...).convert(“L”)*** – Cho phép chuyển ảnh màu sang ảnh thang xám (Grayscale), bước tiên quyết để trích xuất ma trận nhị phân.
* **NumPy**: Thư viện tính toán số (numerical computing) với cấu trúc mảng N chiều (ndarray). Chúng tôi chuyển đổi ảnh thang xám thành mảng NumPy để so sánh giá trị pixel và tạo ma trận Boolean (***free = arr > 128***)
* **heapq**: Mô-đun chuẩn của Python hỗ trợ heap (priority queue). Thuật toán A\* sử dụng heap để lưu trữ và truy xuất nhanh nút có giá trị f=g+hf = g + hf=g+h nhỏ nhất.
* **Matplotlib:** Thư viện vẽ đồ thị và hình ảnh. Kết quả đường đi được trực quan hóa bằng lệnh ax.plot(...) và ax.scatter(...), đảm bảo đường nét mượt mà và marker rõ ràng.
* **pathlib**: Mô-đun quản lý đường dẫn tệp (file paths) theo kiểu hướng đối tượng, giúp xác định vị trí file mê cung một cách linh hoạt bất kể hệ điều hành.

Những thư viện trên đều là các gói nguồn mở, cập nhật thường xuyên và có cộng đồng hỗ trợ rộng lớn, giúp dễ dàng cài đặt qua pip và tích hợp nhanh vào dự án.

1. **Định dạng đầu vào - ảnh mê cung**

Mê cung đầu vào được lưu dưới dạng PNG (Portable Network Graphics), định dạng nén không mất mát (lossless) phù hợp cho việc xử lý pixel chính xác. Ảnh gốc có kích thước *w x h* pixel, trong đó mỗi ô mê cung tương đương với một vùng vuông kích thước:

*(w / NUM\_COLS)* x *(h / NUM\_ROWS)* pixel.

**Quy trình chuyển đổi ảnh bao gồm:**

* **Đọc file**: Sử dụng ***Image.open(path)*** để tải ảnh màu.
* **Chuyển sang thang xám**: Lệnh ***.convert("L")*** biến mỗi pixel thành mức xám 0–255.
* **Ngưỡng nhị phân (binary threshold)**: So sánh với giá trị 128 để phân loại ô “tường” (***pixel ≤ 128***) và ô “đường đi” (***pixel > 128***). Kết quả là ma trận Boolean free kích thước ***h*** x ***w***, trong đó ***True*** biểu thị ô thông.

Nhờ quy trình này, cấu trúc mê cung 2D dễ dàng được ánh xạ vào lưới (grid) ô vuông, làm cơ sở cho thuật toán A\* thực hiện tìm đường một cách chính xác và hiệu quả.

Phần III: Thiết kế giải pháp

1. **Mô hình mê cung 2D dưới dạng lưới (grid)**

***Ảnh mê cung*** được chia thành ma trận ô vuông kích thước NUM\_ROWS x NUM\_COLS (ở đây em dùng là 20x20).

Mỗi ***pixel*** trong ảnh quy về một phần tử mảng Boolean ***free[y, x]*** (***True*** nếu trống, ***False*** nếu là tường).

Ta tính trước ***pixel\_per\_cell*** để biết mỗi ô gồm bao nhiêu pixel:

A screen shot of a computer program

AI-generated content may be incorrect.

Khi tìm đường, mỗi bước di chuyển chỉ qua tâm các ô: từ ô ***(r, c)*** → một trong bốn ô lân cận ***(r±1, c)*** hoặc ***(r, c±1)***.

1. **Thuật toán A\* và heuristic Manhattan**

**A\*** (A‑star) là thuật toán tìm đường ngắn nhất trên đồ thị có trọng số không âm, tối ưu và hoàn chỉnh nếu heuristic thỏa tính **admissible** (không đánh giá quá cao).

**Heuristic sử dụng:**

h((y1, x1), (y2, x2)) = |y1 – y2| + |x1 – x2|

* Ước lượng khoảng các Manhattan trong lưới 4 (hướng)

**Hàm f(n) cho mỗi nút n:**

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

* Trong đó g(n) là chi phí đã đi từ Start đến n (ở đây mỗi bước có trọng số 1), h(n) là ước lượng đến Goal.

**Quy trình chính:**

1. Đưa Start vào ***open\_set*** (priority queue) với ***f = h(start)***, ***g = 0***.
2. Mỗi bước lấy nút có ***f*** nhỏ nhất.
3. Nếu là **Goal** → dừng và reconstruct đường đi.
4. Ngược lại, mở rộng 4 lân cận, cập nhật ***g\_score*** và đẩy vào ***open\_set*** nếu tốt hơn.
5. Chuyển nút vừa xét sang ***closed\_set*** để không xét lại.
6. **Cấu trúc dữ liệu chính**

**open\_set**: heap (qua heapq), lưu tuple (f, g, (y, x)) để luôn pop nút có f nhỏ nhất.

**g\_score**: dict mapping mỗi ô → chi phí g tốt nhất tìm được tới ô đó.

**came\_from**: dict mapping mỗi ô → ô cha (dùng để reconstruct đường đi khi đến Goal).

**closed\_set**: set chứa các ô đã xét, tránh lặp vô hạn.

A screen shot of a computer program

AI-generated content may be incorrect.

**Reconstruct đường đi** khi cur == goal:

A screen shot of a computer code

AI-generated content may be incorrect.

Phần thiết kế giải pháp trên đảm bảo thuật toán vận hành chính xác, hiệu quả và dễ dàng mở rộng (ví dụ có thể thêm chi phí khác nhau cho mỗi ô).

Phần IV: Cài đặt chi tiết

1. **Tải ảnh và chuyển sang ma trận nhị phân (Phân 1 của code)**

A screen shot of a computer program

AI-generated content may be incorrect.

**Image.open(…)**: đọc file ảnh gốc (màu hoặc đen‑trắng).

**.convert("L")**: chuyển về ảnh thang xám, mỗi pixel là giá trị 0–255.

**np.array(img)**: biến đối tượng ảnh thành mảng NumPy 2D (uint8).

**free = arr > 128**: ngưỡng 128 để phân biệt “ô thông” (True) và “ô tường” (False).

**h, w**: chiều cao (số hàng) và chiều rộng (số cột) của ảnh, tính bằng pixel.

Tính kích thước trung bình mỗi ô (cell) trong pixel:

* + px\_col: pixel trên mỗi cột.
  + px\_row: pixel trên mỗi hàng.
  + Lấy trung bình rồi làm tròn để dùng cho chuyển đổi pixel – ô vuông.

1. **Xác định Start và Goal (Phần 2 của code)**

A screen shot of a computer program

AI-generated content may be incorrect.

* Quét dòng đầu (y = 0) để tìm ô trống làm **Start**.
* Quét dòng cuối (y = h -1) để tìm ô trống làm **Goal**.
* Chọn cửa đầu tiên trong mỗi danh sách.
* Nếu không tìm thấy cửa nào trống, thì báo lỗi và kết thúc chương trình.

A screen shot of a computer code

AI-generated content may be incorrect.

* Chuyển kiểu tọa độ từ (row, col) thành (x, y) cho dễ tương tác.

1. **Cài đặt thuật toán A\*:**
2. Heuristic (Manhattan)

A black background with white text

AI-generated content may be incorrect.

Công thức: h(a,b) = |y₁ - y₂| + |x₁ - x₂|

* Ước lượng số bước cần đi trong lưới
* Đảm bảo khoảng cách tới đích là ngắn nhất, để A\* có thể tìm được đường đi ngắn nhất.

1. Khởi tạo open head và g\_score

A screen shot of a computer code

AI-generated content may be incorrect.

1. Vòng lặp chính: pop head, kiểm tra Goal, mở/đóng đỉnh

A screen shot of a computer screen

AI-generated content may be incorrect.

***Heapq.heappop***: Pop node có f nhỏ nhất.

***Closed***: Đảm bảo mỗi node chỉ xét đúng 1 lần.

***Mở rộng***: Duyệt sang các node mới theo 4 hướng (Trái, phải, lên, xuống), tính g mới ***ng = g + 1***.

***Cập nhật***: Nếu ng tốt hơn, lưu vào ***came\_from*** và đẩy lại heap với ***f = ng + h***.

Nếu như đã tìm được đích:

* + Ta sẽ gắn điểm hiện tại vào trong path
  + Truy ngược lại để tạo thành đường dẫn từ Goal tới Start
  + In ra đường dẫn ngược (Từ Start tới Goal)

1. **Nén đường dẫn t hành các đoạn – hàm *compress* (Phần 4 của code)**

A screen shot of a computer program

AI-generated content may be incorrect.

**Dir\_map**: Ánh vector thành tên hướng.

Mỗi lần di chuyển sang cùng 1 hướng thì tăng count (cnt), mỗi khi đổi hướng, in ra đoạn đã cộng rồi reset để tiếp tục tạo thành 1 đoạn đường đi mới.

1. **Chuyển pixel thành ô và in kết quả**

A screen shot of a computer code

AI-generated content may be incorrect.

Chuyển đổi độ dài pixel thành độ dài ô.

In ra tổng bước và chi tiết từng đoạn di chuyển để người xem dễ dàng theo dõi kết quả.

1. **Vẽ đường đi, đánh dấu Start, Goal (phần 6 của code)**

A screen shot of a computer program

AI-generated content may be incorrect.

6.1) Lọc trùng cell để không vẽ thừa.

6.2) Tính tọa độ tâm cho đường vẽ ra chính giữa.

6.3) Offset Start để marker “nhô” khỏi mê cung, em muốn hình vẽ sẽ kiểu, mình đi từ bên ngoài vào trong mê cung, cho nên em muốn phần đánh dấu Start ở phía bên ngoài mê cung.

6.4, 6.5) Vẽ nền mê cung, đường đỏ và marker Start/Goal.

Phần V: Kết quả thử nghiệm

A screenshot of a maze game

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Phần VI: Kết luận và hướng phát triển:

1. **Kết luận**

Giải pháp A\* với Heuristic Manhattan cho phép tìm đường ngắn nhất trong mê cung 2D một cách chính xác và nhanh chóng.

Toàn bộ quy trình từ tiền xử lý hình ảnh, xác định Start/Goal, xây dựng A\*, nén kết quả để trực quan hóa được triển khai 1 cách hoàn thiện.

1. **Thiếu sót và hướng phát triển:**

**Thiếu sót:**

* Phần xác định Start và Goal trong chương trình của em vẫn còn sử dụng cách mặc định, thiếu linh hoạt, giả sử như thay nguồn ảnh mê cung thành 1 mê cung khác với 2 cửa ra vào nằm ở 2 bên trái phải chứ không phải ở trên dưới như mê cung hiện tại, thì với cách xác định Start và Goal như hiện tại thì không thể nào tìm thấy được Start và Goal.
* Vẫn chưa thiết lập được các ô chi phí khác nhau, hiện tại thì trong chương trình của em, các bước đi trong mê cung đều tốn 1 mức chi phí là bằng nhau và cố định là 1.

**Hướng phát triển:**

* Cải thiện thêm về chương trình, nâng cấp phương thức xác định cửa ra và cửa vào của mê cung, như là sẽ xét toàn bộ tường ngoài của mê cung, sau đó tìm ra 2 ô trống, xác định cửa ra và cửa vào theo quy ước: Nếu như 2 cửa tìm được, 1 cửa ở trên và 1 cửa ở dưới thì mặc định cửa trên là cửa vào, và cửa dưới là cửa ra; nếu như 2 cửa tìm được ở 2 bên trái phải, cửa bên trái sẽ là cửa vào, cửa bên phải sẽ là cửa ra; còn nếu 2 cửa tìm được là cùng 1 bên, thì cửa nào tìm được trước sẽ là cửa vào.
* Cải thiện thêm về phần chi phí tiêu tốn các bước đi ở mỗi ô trong mê cung, sẽ có thêm những ô tốn nhiều chi phí hơn.