HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1

_____ o0o _____

BÀI TẬP LỚN NHẬP MÔN TRÍ TUỆ NHÂN TẠO

Tên đề tài: Sử dụng thuật toán tìm kiếm A* để tìm ra đường đi ngắn nhất từ điểm xuất phát đến điểm đích trong một lưới không gian 2 chiều có vật cản.

LỚP : N10
Số thứ tự nhóm: 02

Trần Mạnh Dương	**MSSV: B22DCAT069**
Nguyễn Đức Trung	**MSSV: B22DCCN871**
Nguyễn Văn Đạt	**MSSV: B22DCCN199**
Nguyễn Tuấn Quỳnh	**MSSV: B22DCKH101**
Nguyễn Đình Dũng	**MSSV: B22DCCN131**

Giảng viên hướng dẫn: Ths. Hoài Thư

HÀ NỘI, 05/2023

LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn ThS. Hoài Thư – giảng viên hướng dẫn đã tận tình chỉ bảo, hỗ trợ chúng em trong suốt quá trình thực hiện bài tập lớn môn Nhập môn Trí tuệ nhân tạo.

Trong quá trình nghiên cứu và hoàn thiện đề tài, chúng em đã học hỏi được nhiều kiến thức quý báu liên quan đến lĩnh vực trí tuệ nhân tạo nói chung và thuật toán cây quyết định nói riêng. Đây là nền tảng quan trọng giúp chúng em hiểu rõ hơn về ứng dụng của các mô hình học máy trong thực tế.

Bên cạnh đó, chúng em cũng xin cảm ơn Học viện Công nghệ Bưu chính Viễn thông, cùng các thầy cô trong khoa Công nghệ Thông tin 1 đã tạo điều kiện và cung cấp môi trường học tập thuận lợi để chúng em hoàn thành bài báo cáo này.

Tuy đã có nhiều cố gắng, nhưng bài làm không tránh khỏi những thiếu sót. Chúng em rất mong nhận được những góp ý quý báu từ thầy cô để bài báo cáo được hoàn thiện hơn.

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT

Thuật ngữ	**Tiếng Anh**	**Tiếng Việt**
A*	A* Algorithm	Thuật toán tìm kiếm sử dụng hàm heuristic để tìm đường đi ngắn nhất trong đồ thị hoặc lưới.
DFS	Depth-First Search	Thuật toán tìm kiếm theo chiều sâu trong cây hoặc đồ thị, đi sâu theo nhánh trước khi quay lại.
AI	Artificial Intelligence	Trí tuệ nhân tạo – mô phỏng hành vi và quá trình tư duy của con người trên máy tính.
GPS	Global Positioning System	Hệ thống định vị toàn cầu, dùng để xác định vị trí địa lý dựa trên tín hiệu vệ tinh.
VLSI Design	Very Large Scale Integration Design	Thiết kế tích hợp vi mạch với hàng triệu linh kiện trên một chip, ứng dụng thuật toán tìm đường để tối ưu bố trí.
2D	Two-Dimensional	Không gian hai chiều – được mô hình hóa bằng lưới gồm các hàng và cột, thường dùng trong mô phỏng.
f(n)	Total Cost Function	Tổng chi phí tại một nút: $f(n) = g(n) + h(n)$, gồm chi phí thực tế và chi phí ước lượng.
g(n)	Actual Cost	Chi phí thực tế từ điểm bắt đầu đến nút n.
h(n)	Heuristic Function	Hàm heuristic ước lượng chi phí từ nút hiện tại đến đích.
Grid	Grid	Lưới ô vuông 2D – không gian tìm kiếm trong bài toán.
Spot	Grid Cell / Spot	Một ô trong lưới, đại diện cho một vị trí có thể đi qua hoặc là vật cản.
PriorityQueue	Priority Queue	Hàng đợi ưu tiên – cấu trúc dữ liệu dùng trong thuật toán để chọn nút có $f(n)$ nhỏ nhất.

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong các bài toán thực tế như tìm đường đi, lập lịch, robot di chuyển trong môi trường có vật cản,... việc tìm kiếm đường đi ngắn nhất và tối ưu luôn là một yêu cầu quan trọng. Một trong những hướng tiếp cận hiệu quả là sử dụng các thuật toán tìm kiếm có sử dụng đánh giá chi phí, trong đó thuật toán A* (A-star) được đánh giá là một trong những thuật toán mạnh và phổ biến nhất.

Thuật toán A* được ứng dụng rộng rãi trong nhiều lĩnh vực như trí tuệ nhân tạo, game, hệ thống định vị, robot tự hành, ... Bài toán tìm đường đi ngắn nhất trong một không gian 2 chiều có vật cản là một bài toán thực tiễn, vừa mang tính ứng dụng cao, vừa thể hiện rõ năng lực triển khai và phân tích thuật toán tìm kiếm.

1.2 Mục tiêu và phạm vi đề tài

Đề tài hướng tới việc:

- Tìm hiểu và trình bày nguyên lý hoạt động của thuật toán A*.
- Xây dựng phần mềm mô phỏng bài toán tìm đường trong lưới 2 chiều có vật cản, có khả năng trực quan hóa quá trình tìm kiếm.
- Cho phép người dùng dễ dàng tùy chỉnh điểm bắt đầu, điểm kết thúc và cấu hình vật cản.
- Đánh giá kết quả tìm đường của thuật toán A* trên các tình huống khác nhau.

Phạm vi của đề tài dừng lại ở không gian 2 chiều, lưới hình chữ nhật, vật cản được xác định trước và không thay đổi trong quá trình tìm kiếm.

Phần mềm mô phỏng sẽ bao gồm các chức năng chính như: khởi tạo lưới và vật cản, lựa chọn điểm bắt đầu và kết thúc, thực thi thuật toán A*, hiển thị trực quan đường đi ngắn nhất tìm được. Phần mềm hướng tới sự đơn giản, dễ sử dụng và có thể mở rộng cho các nghiên cứu nâng cao trong tương lai.

1.3 Định hướng giải pháp

Giải pháp được nhóm lựa chọn bao gồm các bước chính:

- Phân tích nguyên lý của thuật toán A* và các thành phần như hàm chi phí tổng $f(n) = g(n) + h(n)$.
- Xây dựng mô hình lưới 2 chiều, định nghĩa điểm bắt đầu, điểm kết thúc, và các vật cản.
- Cài đặt thuật toán A* bằng ngôn ngữ lập trình phù hợp ở đây là Python

- Mô phỏng trực quan quá trình tìm đường để kiểm chứng độ chính xác và hiệu quả.

Thuật toán A* được kỳ vọng mang lại kết quả chính xác và nhanh chóng trong việc tìm ra đường đi tối ưu, đồng thời dễ mở rộng hoặc tích hợp với các hệ thống lớn hơn trong tương lai.

1.4 Bố cục bài tập lớn

Chương 1: Giới thiệu Trình bày tổng quan đề tài, mục tiêu và định hướng giải pháp dựa trên thuật toán A*.

Chương 2: Cơ sở lý thuyết Trình bày nguyên lý hoạt động và ứng dụng của thuật toán A* trong bài toán tìm đường.

Chương 3: Thực nghiệm và kết quả Mô tả cấu trúc phần mềm, giao diện và cách người dùng tương tác để mô phỏng tìm đường.

Chương 4: Kết luận Trình bày kết quả thực nghiệm, đánh giá hiệu quả thuật toán và đề xuất cải tiến trong tương lai.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Tổng quan

Chương này trình bày về bài toán tìm đường trong không gian và sự cần thiết của nó trong các ứng dụng thực tế. Thuật toán A* được giới thiệu chi tiết về nguyên lý, cách hoạt động và ưu nhược điểm. Cuối chương là ví dụ minh họa và các ứng dụng tiêu biểu của thuật toán trong đời sống và công nghệ.

2.2 Giới thiệu về bài toán tìm đường trong không gian

- Bài toán tìm đường trong không gian là một vấn đề phổ biến trong nhiều lĩnh vực như đồ họa máy tính, robot tự động, trí tuệ nhân tạo (AI) và hệ thống điều hướng. Mục tiêu của bài toán là tìm ra một lộ trình tối ưu (hoặc hợp lý) từ một điểm xuất phát đến một điểm đích trong một không gian, với điều kiện có thể có những rào cản hoặc các yếu tố ảnh hưởng đến quá trình di chuyển.
- Không gian tìm kiếm có thể được mô hình hóa dưới dạng một lưới (grid), đồ thị (graph), hoặc bất kỳ cấu trúc không gian nào khác, trong đó các điểm (hoặc đỉnh) đại diện cho các vị trí và các cạnh nối các điểm này với nhau. Trong một số trường hợp, không gian có thể có nhiều chiều và các rào cản có thể có hình dạng phức tạp.
- Vấn đề tìm đường có thể được chia thành các loại khác nhau, ví dụ như:
  - Tìm đường trong không gian hai chiều: Đây là dạng bài toán phổ biến nhất, chẳng hạn như tìm đường đi trong một thành phố, nơi mỗi điểm trên lưới đại diện cho một khu vực hoặc ngã tư.
  - Tìm đường trong không gian ba chiều: Sử dụng trong các ứng dụng như điều hướng robot hoặc tàu vũ trụ, nơi có sự di chuyển trong không gian 3D.
  - Tìm đường với rào cản: Trong nhiều bài toán, không gian không phải lúc nào cũng trống rỗng mà có các vật thể cản trở, tạo thành các rào cản mà robot hoặc đối tượng phải tránh.
- Với sự phát triển của các thuật toán tìm kiếm, bài toán này đã được giải quyết hiệu quả trong các tình huống thực tế, từ việc điều hướng trong các trò chơi điện tử đến việc lập kế hoạch hành trình cho robot.

2.3 Thuật toán A*

2.3.1 Nguyên lý hoạt động

- Thuật toán A* là một thuật toán tìm kiếm có thông tin, được sử dụng để tìm lộ

trình tối ưu từ điểm xuất phát đến điểm đích trong một không gian tìm kiếm. Nguyên lý hoạt động của thuật toán A* kết hợp giữa tìm kiếm theo chi phí thấp nhất (Dijkstra) và tìm kiếm theo chiều sâu (DFS), cùng với việc sử dụng một hàm đánh giá để lựa chọn các bước tiếp theo trong quá trình tìm kiếm.

- A* sử dụng hai hàm chính để tính toán chi phí:
  - $g(n)$ là giá thành đường đi từ nút xuất phát đến nút n.
  - $h(n)$ là giá thành ước lượng từ nút n đến đích.
- Tổng chi phí $f(n)$ tại mỗi nút n được tính theo công thức:

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

  trong đó:
  - $g(n)$ là giá thành đường đi từ nút xuất phát đến nút n.
  - $h(n)$ là giá thành ước lượng từ nút n đến đích.
- Thuật toán A* yêu cầu hàm $h(n)$ là hàm chấp nhận được (admissible) theo định nghĩa sau:
  - Hàm $h(n)$ được gọi là **chấp nhận được** nếu thỏa mãn:

$$\forall n : h(n) \leq h^*(n)$$

    trong đó:
    - $h^*(n)$ là khoảng cách thực tế từ n tới đích
    - $h(n)$ là khoảng cách ước lượng từ n tới đích
  - **Ví dụ:** Giả sử với các nút n như sau:

Nút n	$h(n)$	$h^*(n)$
A	30	40
B	38	55

Bảng 2.1: Kết quả so sánh giữa $h(n)$ và $h^*(n)$

    Vì $h(n) \leq h^*(n)$ với mọi n, nên $h(n)$ là một hàm chấp nhận được.
- Mục tiêu của A* là tìm đường đi có tổng chi phí $f(n)$ nhỏ nhất, tức là hàm $f(n)$ giúp A* quyết định điểm tiếp theo để mở rộng tìm kiếm dựa trên chi phí thực tế và chi phí ước lượng.
- Thuật toán A* mở rộng các điểm theo thứ tự các giá trị $f(n)$ từ nhỏ đến lớn, và

chọn điểm có giá trị $f(n)$ thấp nhất để tiếp tục. Thuật toán dừng lại khi điểm đích được tìm thấy hoặc không còn điểm nào để kiểm tra.

2.3.2 Thuật toán A*

- Dưới đây là mô tả chi tiết các bước thực hiện của thuật toán A*:

Thuật toán: A*(Q, S, G, P, c, h)
- **Đầu vào:** bài toán tìm kiếm, hàm heuristic h
- **Đầu ra:** đường đi ngắn nhất từ nút xuất phát đến nút đích
- **Khởi tạo:** tập các nút biên (nút mở) $O \leftarrow S$

While (O không rỗng) **do**
1. Lấy nút n có $f(n)$ nhỏ nhất ra khỏi O
2. Nếu $n \in G$ thì **return** (đường đi tới n)
3. Với mọi nút $m \in P(n)$ thực hiện:
   i. $g(m) = g(n) + c(m, n)$
   ii. $f(m) = g(m) + h(m)$
   iii. Thêm m vào O cùng giá trị $f(m)$

Return: không tìm được đường đi

Bảng 2.2: Thuật toán A*

2.3.3 Ưu điểm và nhược điểm của thuật toán A*

- **Ưu điểm:**
  - **Tìm kiếm tối ưu (optimal):** Nếu hàm heuristic $h(n)$ là chấp nhận được và không vượt quá chi phí thực tế đến đích, thuật toán A* sẽ đảm bảo tìm ra đường đi ngắn nhất.
  - **Tìm kiếm đầy đủ (complete):** Thuật toán sẽ tìm được lời giải nếu tồn tại, miễn là không gian tìm kiếm hữu hạn.
  - **Hiệu quả cao trong nhiều trường hợp thực tế:** Khi hàm heuristic được thiết kế tốt, A* có thể tìm ra lời giải nhanh hơn nhiều so với các thuật toán không sử dụng thông tin.
  - **Dễ điều chỉnh:** Có thể điều chỉnh độ chính xác và tốc độ bằng cách thay đổi hàm heuristic.
- **Nhược điểm:**
  - **Tiêu tốn bộ nhớ:** A* yêu cầu lưu trữ tất cả các nút đã mở và đã xét, dẫn đến tiêu tốn bộ nhớ rất lớn trong không gian tìm kiếm lớn.
  - **Phụ thuộc vào chất lượng heuristic:** Nếu hàm $h(n)$ không chính xác hoặc không phù hợp, hiệu quả của thuật toán có thể giảm đáng kể, thậm

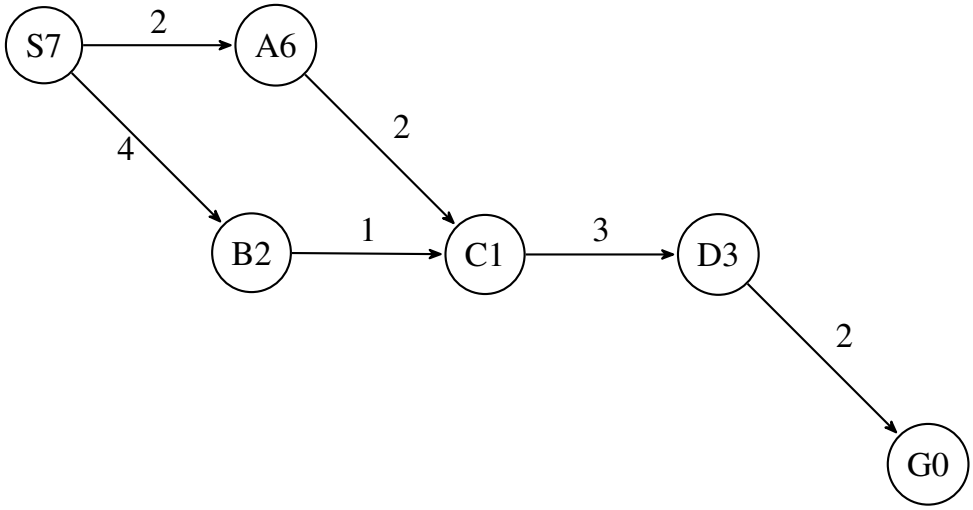
chí không tối ưu.

– **Không phù hợp với các hệ thống thời gian thực nếu không có tối ưu hóa thêm:** Vì có thể cần nhiều bước để tính toán trước khi đưa ra quyết định.

2.4 Ứng dụng thuật toán A* trong bài toán tìm đường

Giả sử chúng ta có một đồ thị như sau, trong đó:

- Nút bắt đầu là S, nút đích là G
- Các cạnh mang trọng số là chi phí thực tế $c(n, m)$ giữa hai nút
- Số nằm trong mỗi đỉnh là giá trị heuristic $h(n)$ ước lượng khoảng cách còn lại đến đích G



Hình 2.1: Đồ thị minh họa cho bài toán tìm đường

Mục tiêu: Tìm đường đi từ S đến G bằng thuật toán A*.

Bảng theo dõi quá trình tìm kiếm A*:

Bước	Nút được duyệt	Tập biên O
KT	$\emptyset$	S(7)
1	S	$A_S(8)$, $B_S(6)$
2	B	$A_S(8)$, $C_B(6)$
3	C	$A_S(8)$, $D_C(11)$
4	A	$D_C(11)$, $C_A(5)$
5	C	$D_C(10)$
6	D	G(9)
7	G	Đích

Bảng 2.3: Các bước duyệt nút theo thuật toán A*

Đường đi được chọn: S $\rightarrow$ A $\rightarrow$ C $\rightarrow$ D $\rightarrow$ G

Tổng chi phí: $g(G) = 9$

Cách xử lý khi gặp nút lặp trong A*:

- Nếu nút lặp đã nằm trong danh sách các nút đã được duyệt (đã đóng), thì thêm lại vào danh sách chờ nếu có giá trị tốt hơn
- Nếu nút lặp đang nằm trong danh sách chờ (tập mở), thì:
  - So sánh chi phí mới g_{new} với chi phí cũ g_{old} đã lưu.
  - Nếu $g_{new} < g_{old}$ thì cập nhật lại đường đi và giá trị chi phí mới.

Một số ứng dụng thực tế:

- **Trí tuệ nhân tạo trong trò chơi (Game AI):** A* được sử dụng để điều khiển hành vi của các nhân vật trong trò chơi, chẳng hạn như tìm đường đi từ vị trí hiện tại đến mục tiêu trong một bản đồ có chướng ngại vật.
- **Hệ thống định vị và dẫn đường (Navigation systems):** Trong các hệ thống GPS và ứng dụng bản đồ (Google Maps, Apple Maps), thuật toán A* có thể được dùng để tính toán lộ trình tối ưu giữa hai vị trí, có xét đến chi phí (quãng đường, thời gian, giao thông,...).
- **Robot di động và lập kế hoạch chuyển động (Motion Planning):** A* là thuật toán cơ bản trong việc lập kế hoạch đường đi cho robot trong môi trường có chướng ngại vật, chẳng hạn như robot hút bụi, robot vận chuyển trong kho (warehouse robots), hoặc drone bay trong không gian 3D.
- **Mạng máy tính và truyền thông:** A* có thể được sử dụng trong việc định tuyến gói tin qua mạng sao cho chi phí truyền tải thấp nhất.
- **Thiết kế mạch điện tử (VLSI Design):** Trong việc thiết kế mạch tích hợp, A* được ứng dụng để tìm các đường đi ngắn nhất cho dây dẫn trong mạch với độ chính xác và tối ưu cao.

2.5 Kết chương

Qua phân tích chi tiết và ví dụ minh họa, ta có thể thấy rằng thuật toán A* là một công cụ rất mạnh mẽ và linh hoạt trong việc giải quyết các bài toán tìm đường trong không gian có ràng buộc. Nhờ sự kết hợp giữa chi phí thực tế và chi phí ước lượng, A* không những đảm bảo tìm kiếm tối ưu mà còn hoạt động hiệu quả trong nhiều trường hợp thực tế như robot tự hành, hệ thống điều hướng, và trí tuệ nhân tạo trong trò chơi.

Tuy nhiên, hiệu quả của A* phụ thuộc đáng kể vào hàm heuristic được lựa chọn.

Do đó, việc thiết kế một hàm heuristic tốt, phù hợp với đặc thù không gian tìm kiếm, là yếu tố then chốt để phát huy tối đa sức mạnh của thuật toán.

Trong các chương tiếp theo, chúng tôi sẽ triển khai mô phỏng thuật toán A* bằng ngôn ngữ lập trình Python, kết hợp thư viện `pygame` để trực quan hóa hoạt động tìm đường, qua đó đánh giá hiệu quả và khả năng ứng dụng thực tế của giải pháp này.

CHƯƠNG 3. THỰC NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

3.1 Tổng quan chương

Chương này trình bày quá trình thiết kế, triển khai và đánh giá một ứng dụng mô phỏng thuật toán tìm đường A* sử dụng thư viện `pygame` trong Python. Nội dung gồm mô tả hệ thống, cấu trúc dữ liệu, thuật toán áp dụng, quy trình thực nghiệm và phân tích kết quả đạt được.

3.2 Thiết kế và triển khai hệ thống

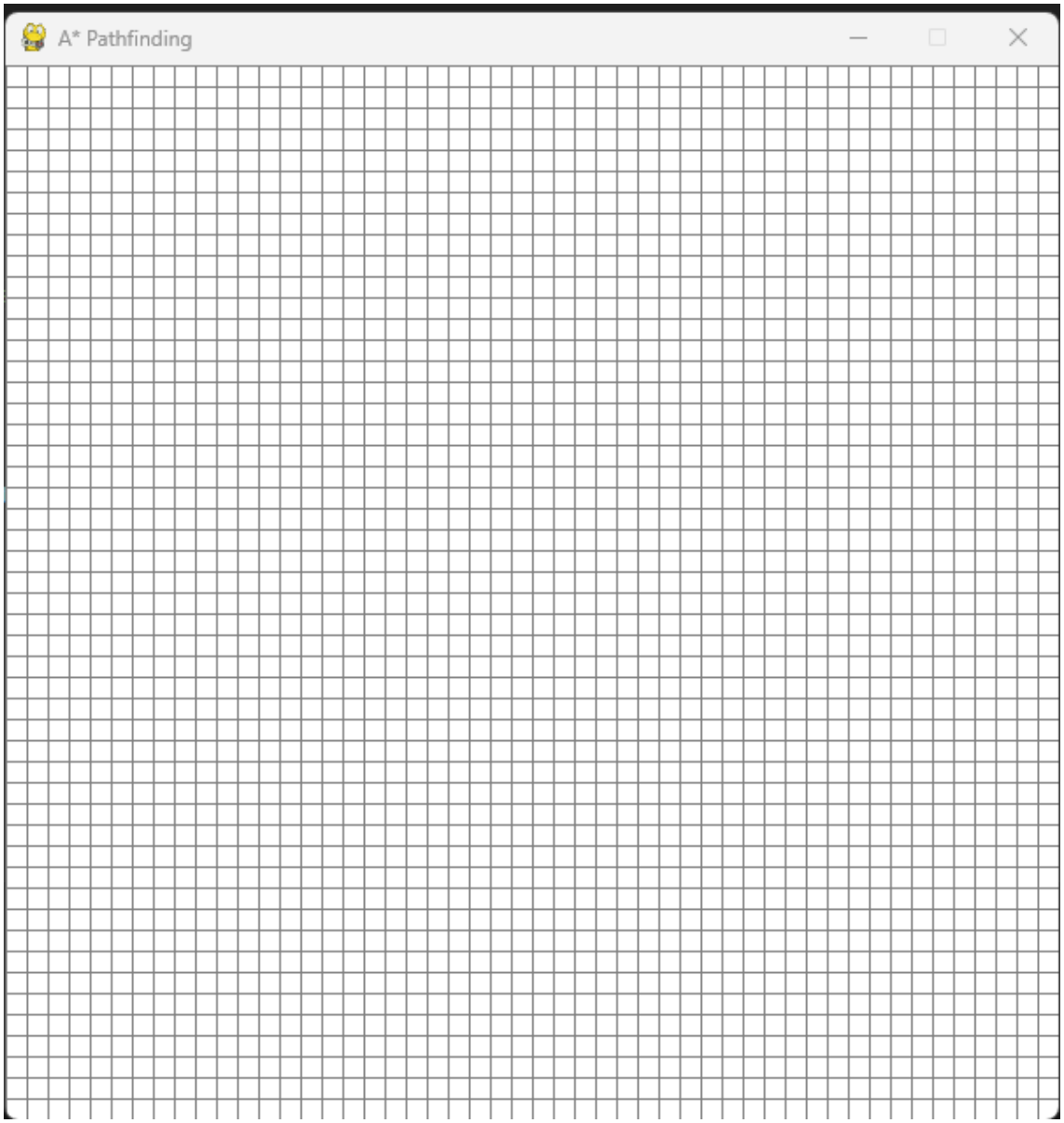
3.2.1 Thiết kế chức năng

Ứng dụng mô phỏng có các chức năng chính như sau:

- **Tạo điểm và chướng ngại vật:** Người dùng sử dụng chuột trái để đặt điểm bắt đầu, điểm kết thúc và các ô chướng ngại trên lưới. Nhấn chuột phải để xóa các ô đã chọn.
- **Khởi chạy thuật toán bằng phím `Space`:** Sau khi thiết lập xong, nhấn `Space` để bắt đầu thuật toán A*. Quá trình tìm đường sẽ được thực hiện nếu tồn tại đường đi hợp lệ.
- **Xóa và thiết lập lại bằng phím `C`:** Nhấn `C` để xóa toàn bộ lưới và khởi động lại từ đầu.
- **Hiển thị trực quan:** Quá trình duyệt và đường đi được hiển thị bằng màu sắc khác nhau, giúp dễ dàng theo dõi thuật toán hoạt động.

3.2.2 Thiết kế giao diện người dùng

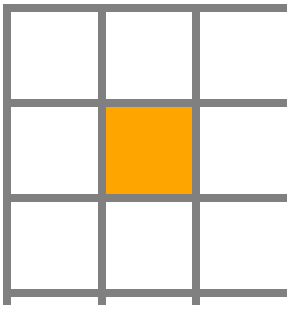
Giao diện người dùng được thiết kế đơn giản, trực quan nhằm giúp người dùng dễ dàng tương tác và theo dõi quá trình hoạt động của thuật toán tìm đường. Thành phần chính của giao diện là một lưới vuông có kích thước 50×50 ô, mô phỏng không gian **2D** mà thuật toán sẽ hoạt động trên đó.



Hình 3.1: Mô tả hình ảnh sơ đồ lưới 50x50

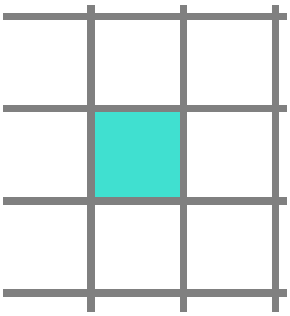
Mỗi ô trong lưới đại diện cho một điểm trên bản đồ và có thể mang một trong các trạng thái sau đây:

- **Trắng** – *Ô trống*: Trạng thái mặc định, có thể đi qua được.
- **Cam** – *Điểm bắt đầu*: Ô nơi thuật toán bắt đầu tìm đường.



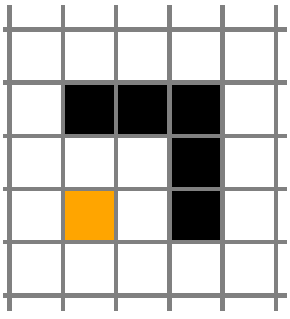
Hình 3.2: Điểm bắt đầu

- **Xanh dương** – *Điểm kết thúc*: Ô đích mà thuật toán cần tìm tới.



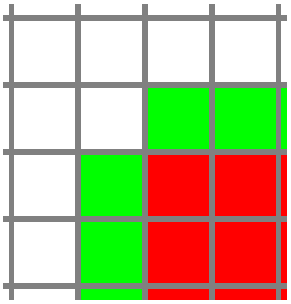
Hình 3.3: Điểm kết thúc

- **Đen** – *Chướng ngại vật*: Ô không thể đi qua, buộc thuật toán tìm đường vòng.



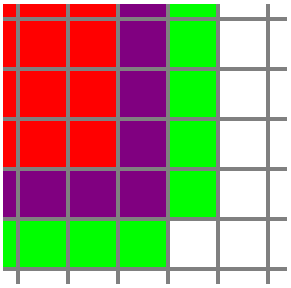
Hình 3.4: Chướng ngại vật

- **Xanh lá** – *Ô đang xét* & **Đỏ** – *Ô đã xét*: Màu xanh lá cho biết các ô đang được xem xét, còn màu đỏ là những ô đã duyệt xong.



Hình 3.5: nút đang xét và nút đã xét

- **Tím** – *Đường đi*: Kết quả cuối cùng, biểu thị đường đi ngắn nhất tìm được.



Hình 3.6: Đường đi cuối cùng

Việc sử dụng màu sắc khác nhau giúp người dùng dễ dàng phân biệt trạng thái của từng ô trên lưới, đồng thời trực quan hóa rõ ràng quá trình hoạt động của thuật toán A*. Nhờ đó, không chỉ kết quả đầu ra mà cả quá trình tìm đường cũng trở nên dễ quan sát và dễ hiểu hơn.

3.2.3 Công cụ và thư viện sử dụng

- **Ngôn ngữ lập trình:** Python 3.13.2
- **Thư viện đồ họa:** `pygame` 2.6.1 – hỗ trợ hiển thị và xử lý sự kiện.
- **Cấu trúc dữ liệu hàng đợi ưu tiên:** `queue.PriorityQueue` – quản lý tập mở trong thuật toán A*.

3.3 Thuật toán và dữ liệu sử dụng

3.3.1 Mô tả thuật toán A*

Thuật toán A* là một giải pháp tìm đường đi ngắn nhất trên đồ thị có trọng số, sử dụng hàm ước lượng (heuristic) để định hướng tìm kiếm. Trong ứng dụng này, hàm heuristic được sử dụng là khoảng cách Manhattan:

$$h(n) = |x_1 - x_2| + |y_1 - y_2|$$

Code thuật toán A*:

```python
def a_star(draw, grid, start, end):
    count = 0
    open_set = PriorityQueue()       # Hang doi uu tien theo f-score
    open_set.put((0, count, start))
    came_from = {}       # Luu vet duong di
    g_score = {spot: float("inf") for row in grid for spot in row}
    g_score[start] = 0
    f_score = {spot: float("inf") for row in grid for spot in row}
    f_score[start] = h(start.get_pos(), end.get_pos())
    open_set_hash = {start}      # Tap de kiem tra nhanh
    while not open_set.empty():
        current = open_set.get()[2]      # Lay phan tu co f-score nho nhat
        open_set_hash.remove(current)
        if current == end:
            # Da tim duoc duong di
            reconstruct_path(came_from, end, draw)
            end.make_end()
            start.make_start()
            return True
        for neighbor in current.neighbors:
            temp_g_score = g_score[current] + 1
            if temp_g_score < g_score[neighbor]:
                # Cap nhat duong di tot hon
                came_from[neighbor] = current
                g_score[neighbor] = temp_g_score
                f_score[neighbor] = temp_g_score + h(neighbor.get_pos(), end.get_pos())
                if neighbor not in open_set_hash:
                    count += 1
                    open_set.put((f_score[neighbor], count, neighbor))
                    open_set_hash.add(neighbor)
                    neighbor.make_open()
        draw()
        if current != start:
            current.make_closed()
    return False
```

Bảng 3.1: Thuật toán A* để tìm đường đi ngắn nhất

3.3.2 Cấu trúc dữ liệu

- **Đối tượng `Spot`:** Mỗi ô trong lưới được biểu diễn bằng một đối tượng thuộc lớp `Spot`. Đây là lớp trung tâm của hệ thống, dùng để lưu trữ trạng thái của từng ô và hỗ trợ vẽ cũng như tính toán đường đi. Một số thuộc tính chính của

lớp này:

- `row`, `col`: vị trí của ô trên lưới (hàng, cột).
- `x`, `y`: tọa độ pixel trên cửa sổ Pygame, tính từ kích thước mỗi ô.
- `color`: màu sắc hiện tại của ô, đại diện cho trạng thái (trắng, đen, cam, xanh, đỏ...).
- `neighbors`: danh sách các ô hàng xóm (không bị chắn) để phục vụ tìm đường.
- `width`, `total_rows`: dùng để tính kích thước và cập nhật vị trí trên màn hình.

```
class Spot:
    # Khoi tao mot o trong grid
    def __init__(self, row, col, width,
       total_rows):
        self.row = row
        self.col = col
        self.x = row * width
        self.y = col * width
        self.color = WHITE
        self.neighbors = []
        self.width = width
        self.total_rows = total_rows
```

Bảng 3.2: Định nghĩa lớp `Spot` đại diện cho một ô trong lưới

- **Lưới (Grid)** Lưới bản đồ được tổ chức dưới dạng một danh sách hai chiều `grid`, trong đó mỗi phần tử là một đối tượng `Spot`. Lưới có kích thước mặc định 50×50 ô, được tạo bởi hàm:

```
# Tao grid 2D gom cac Spot
def make_grid(rows, width):
    grid = []
    gap = width // rows
    for i in range(rows):
        grid.append([Spot(i, j, gap, rows)
           for j in range(rows)])
    return grid
```

Bảng 3.3: Tạo lưới 2D từ các đối tượng Spot

- **Tọa độ và xử lý chuột** Tương tác chuột được xử lý thông qua hàm `get_clicked_pos()` để chuyển đổi tọa độ pixel của con trỏ sang chỉ số ô trong lưới.

```
1         # Chuyen toa do chuot ve chi so row/col
2         def get_clicked_pos(pos, rows, width):
3             gap = width // rows
4             y, x = pos
5             return y // gap, x // gap
```

Bảng 3.4: Chuyển tọa độ chuột về chỉ số dòng/cột

- **Vẽ giao diện** Mỗi ô trong lưới được vẽ bằng phương thức `draw(win)` của đối tượng `Spot`, sử dụng hàm `pygame.draw.rect()`.

```
1         # Ve o len man hinh
2         def draw(self, win):
3             pygame.draw.rect(win, self.color, (self.x
                 , self.y, self.width, self.width))
```

Bảng 3.5: Phương thức `draw` để vẽ một ô Spot lên màn hình bằng `pygame`

3.4 Thực nghiệm và kết quả

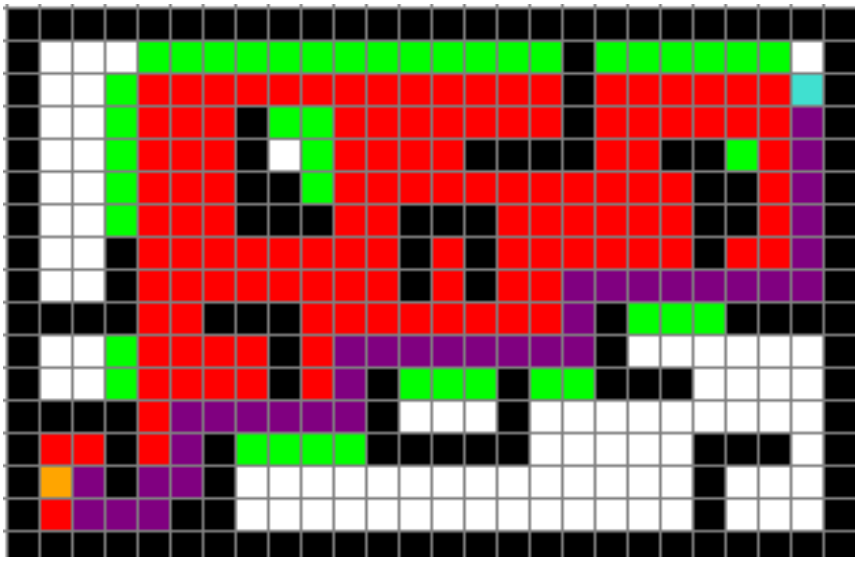
3.4.1 Quy trình thực nghiệm

Các bước thực nghiệm được thực hiện như sau:

- Khởi tạo lưới kích thước 50×50.
- Đặt điểm bắt đầu và kết thúc bằng chuột.
- Vẽ các chướng ngại vật tùy ý.
- Nhấn phím `Space` để khởi chạy thuật toán A*.
- Quan sát trực quan quá trình tìm kiếm và ghi nhận kết quả.

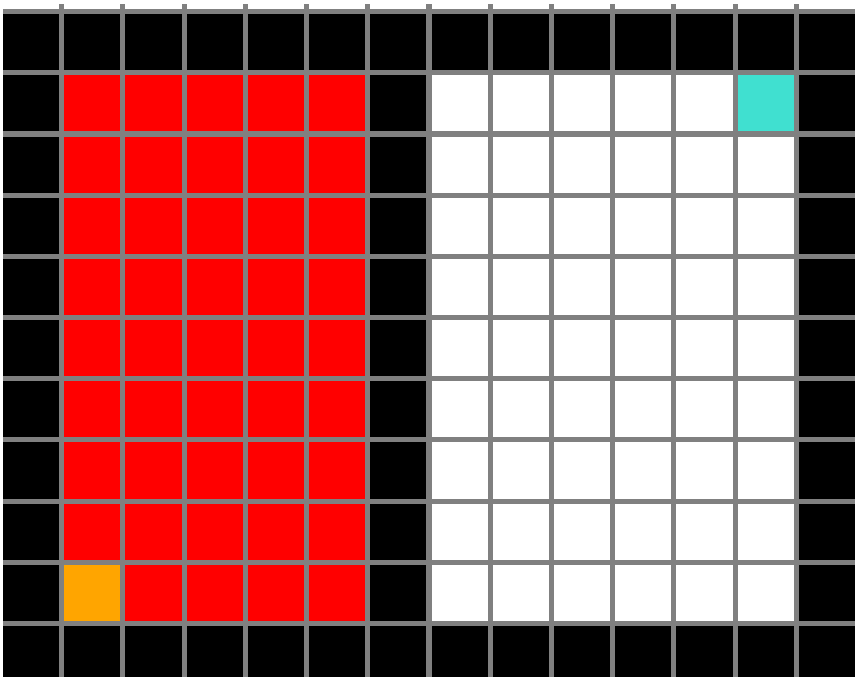
3.4.2 Kết quả đạt được

- Ứng dụng đã mô phỏng thành công quá trình hoạt động của thuật toán A*, với khả năng hiển thị trực quan các bước duyệt ô trong quá trình tìm đường.
- **Trường hợp tồn tại đường đi**: Thuật toán tìm ra đường đi ngắn nhất từ điểm xuất phát đến điểm đích. Các ô được duyệt được tô màu riêng biệt, và đường đi tối ưu được đánh dấu nổi bật, giúp người dùng dễ dàng quan sát và hiểu rõ quá trình hoạt động của thuật toán.



Hình 3.7: Tìm được đường đi

- **Trường hợp không tồn tại đường đi**: Khi không có đường hợp lệ do bị chặn bởi chướng ngại vật, thuật toán tự động dừng và không hiển thị đường đi. Giao diện vẫn thể hiện rõ các ô đã được duyệt, giúp người dùng nhận biết nguyên nhân không tìm được đường.



Hình 3.8: Không tìm được đường đi được đường đi

- Tốc độ hiển thị và khả năng tương tác của ứng dụng ổn định, đảm bảo trải nghiệm mượt mà, không xảy ra giật/lỗi khi người dùng tương tác với bản đồ hoặc thay đổi dữ liệu đầu vào.

3.5 Kết chương

Chương này đã mô tả toàn diện quá trình xây dựng hệ thống mô phỏng thuật toán A* từ thiết kế giao diện, triển khai thuật toán, đến thực nghiệm và đánh giá. Qua thực nghiệm, hệ thống cho thấy khả năng hoạt động hiệu quả, trực quan và dễ sử dụng. Những đánh giá và hạn chế nêu trên là cơ sở để cải tiến và mở rộng hệ thống trong các nghiên cứu tiếp theo.

CHƯƠNG 4. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1 Kết luận

- Trong quá trình thực hiện bài tập lớn, nhóm đã xây dựng thành công phần mềm mô phỏng thuật toán A*, giúp tìm đường đi ngắn nhất trên lưới ô vuông với giao diện trực quan, dễ sử dụng.
- **Giải pháp:** Phần mềm cho phép người dùng thiết lập bản đồ, chọn điểm bắt đầu, kết thúc và chướng ngại vật. Thuật toán A* được hiện thực hiệu quả với heuristic Manhattan, hiển thị trực tiếp quá trình duyệt và đường đi kết quả.
- **Kết quả đạt được:** Hệ thống hoạt động ổn định, phản hồi nhanh, xử lý đúng cả khi có và không có đường đi. Có thể áp dụng cho các tình huống như robot di chuyển, AI game, ...
- **Đánh giá và bài học:** Trong quá trình phát triển, nhóm đã rèn luyện được kỹ năng phân tích bài toán, lập trình giao diện, kiểm thử giải thuật. Những bài học đáng giá là cần chia nhỏ bài toán, trực quan hóa giúp hiểu sâu hơn, và kết hợp lý thuyết với ứng dụng thực tế.
- **Hạn chế:** Phần mềm hiện chưa hỗ trợ lựa chọn kích thước lưới, hiển thị đồ dài đường đi, chưa đo được bộ nhớ sử dụng và thời gian tìm kiếm.

4.2 Hướng phát triển

- **Hoàn thiện sản phẩm:** Bổ sung lựa chọn kích thước lưới (5x5,10x10,25x25,...), hiển thị số liệu như số ô duyệt, bộ nhớ sử dụng, hỗ trợ bản đồ lớn và lưu/tải bản đồ.
- **Mở rộng ứng dụng:** Đọc bản đồ từ dữ liệu thực, tích hợp cho mô phỏng robot hoặc AI trong game, so sánh A* với các thuật toán khác (Dijkstra, BFS...).

PHÂN CÔNG NHIỆM VỤ

Dưới đây là bảng phân công nhiệm vụ của các thành viên trong quá trình thực hiện bài tập lớn:

STT	Họ và tên	Nhiệm vụ
1	Trần Mạnh Dương	Tìm hiểu nội dung đề tài, nghiên cứu lý thuyết liên quan và thực hiện viết báo cáo bài tập lớn.
2	Nguyễn Đức Trung	Thiết kế slide và thực hiện thuyết trình bài tập lớn.
3	Nguyễn Văn Đạt	Thực hiện lập trình và xây dựng phần mềm mô phỏng (code demo) cho bài tập lớn.
4	Nguyễn Tuấn Quỳnh	Thực hiện lập trình và xây dựng phần mềm mô phỏng (code demo) cho bài tập lớn.
5	Nguyễn Đình Dũng	Thực hiện lập trình và xây dựng phần mềm mô phỏng (code demo) cho bài tập lớn.

Bên cạnh các nhiệm vụ chính nêu trên, tất cả các thành viên đều cùng nhau tham gia thảo luận, đóng góp ý kiến và hoàn thiện nội dung báo cáo để đảm bảo chất lượng chung của bài tập lớn.

TÀI LIỆU THAM KHẢO

R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of A*," *Journal of the ACM*, vol. 32, no. 3, pp. 505–536, 1985.

E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed., Pearson, 2010.

S. Koenig and M. Likhachev, "D* Lite," in *Proceedings of the AAAI Conference on Artificial Intelligence*, Edmonton, Canada, 2002, pp. 476–483.

A. Knott, "A data-driven methodology for motivating a set of coherence relations," Ph.D. dissertation, The University of Edinburgh, UK, 1996.

Python Software Foundation, *Python 3.13.2 Documentation*. [Online]. Available: `https://docs.python.org/3.13/` (visited on 05/10/2025).

pygame Community, *pygame Documentation*. [Online]. Available: `https://www.pygame.org/docs/` (visited on 05/10/2025).