**MỤC LỤC**

[**Lời mở đầu** 2](#_Toc132262577)

[**CHƯƠNG 1: TỔNG QUAN** 4](#_Toc132262578)

[**1.1 Giới thiệu chung** 4](#_Toc132262579)

[**1.2 Phát biểu bài toán** 4](#_Toc132262580)

[**1.3 Mục đích và phạm vi đề tài** 5](#_Toc132262581)

[**CHƯƠNG 2: THUẬT TOÁN A\* ỨNG DỤNG CHO BÀI TOÁN TÌM ĐƯỜNG ĐI TRONG MỘT LƯỚI 2 CHIỀU CÓ VẬT CẢN** 6](#_Toc132262582)

[**2.1 Giới thiệu về thuật toán A\*** 6](#_Toc132262583)

[**2.2 Thuật toán A\* ứng dụng cho bài toán tìm đường đi trong một lưới 2 chiều có vật cản** 7](#_Toc132262584)

[**CHƯƠNG 3: THỰC NGHIỆM VÀ KẾT QUẢ** 10](#_Toc132262585)

[**3.1 Mô tả dữ liệu** 10](#_Toc132262586)

[**3.2 Kết quả thực nghiệm** 10](#_Toc132262587)

[**CHƯƠNG 4: KẾT LUẬN** 14](#_Toc132262588)

[**4.1 Kết luận** 14](#_Toc132262589)

[**4.2 Hướng phát triển của đồ án trong tương lai** 14](#_Toc132262590)

[**Tài liệu tham khảo** 15](#_Toc132262591)

**DANH MỤC HÌNH VẼ**

**Hình 1.1** Tìm đường đi trong một lưới 2 chiều có vật cản

**Hình 2.1** Lớp Point

**Hình 2.2** Lớp Node

**Hình 2.3** Khoảng cách Manhattan

**Hình 2.4** Hàm heuristic tính bằng khoảng cách Manhattan

**Hình 2.5** Khởi tạo

**Hình 2.6** Thuật toán

**Hình 3.1** Giao diện chính

**Hình 3.2** Bản đồ 10 x 10

**Hình 3.3** Bản đồ 20 x 20

**Hình 3.4** Bản đồ 40 x 40

**Hình 3.5** Bản đồ 80 x 80

**Hình 3.6** Bản đồ 160 x 160

# **Lời mở đầu**

***Lý do chọn đề tài***

Thuật toán A\* là một thuật toán phổ biến và hiệu quả được sử dụng rộng rãi trong các hệ thống rô-bốt, trò chơi và điều hướng để tìm đường đi hiệu quả nhất từ điểm này đến điểm khác. Thuật toán này đặc biệt hữu ích khi có chướng ngại vật cần phải tránh, khiến nó trở thành công cụ thiết yếu để tìm đường trong môi trường phức tạp. Bằng cách nghiên cứu chủ đề này, ta sẽ hiểu sâu hơn về cách hoạt động của thuật toán A\* và cách áp dụng thuật toán này cho các tình huống trong thế giới thực. Ngoài ra, ta cũng sẽ phát triển các kỹ năng lập trình và giải quyết vấn đề có thể áp dụng cho nhiều dự án về khoa học và kỹ thuật máy tính.

***Các vấn đề cần thực hiện***

* Cách để ứng dụng thuật toán A\* vào bài toán tìm đường đi ngắn nhất từ điểm xuất phát đến điểm đích trong một lưới không gian 2 chiều có vật cản.
* Xây dựng một ứng dụng nhỏ để trực quan hóa thuật toán A\*.

***Phương pháp thực hiện***

* Sử dụng ngôn ngữ Java để xây dựng các cấu trúc dữ liệu cần thiết đồng thời sử dụng các cấu trúc dữ liệu có sẵn, cài đặt thuật toán dựa trên các cấu trúc dữ liệu.
* Triển khai một giao diện sử dụng thư viện Swing để trực quan hóa thuật toán.

***Công cụ sử dụng***

* IntelliJ IDEA.

Báo cáo sẽ tập trung trình bày một số nội dung chính như sau:

**Chương 1: Tổng quan về tìm đường đi trong một lưới 2 chiều có vật cản**

Nội dung chương 1 sẽ khái quát vấn đề, phát biểu bài toán tìm đường đi trong một lưới 2 chiều có vật cản, khảo sát một số thuật toán tìm đường đi và trình bày về phạm vi, mục đích của báo cáo.

**Chương 2: Thuật toán A\* ứng dụng cho bài toán tìm đường đi trong một lưới 2 chiều có vật cản**

Nội dung của chương 2 sẽ giới thiệu kiến thức cơ bản về trí tuệ nhân tạo và thuật toán A\* cũng như các bước sử dụng thuật toán A\* để giải bài toán tìm đường đi trong một lưới 2 chiều có vật cản.

**Chương 3: Thực nghiệm và kết quả:**

Nội dung của chương 3 mô tả dữ liệu sử dụng và trình bày các kết quả của quá trình thực nghiệm.

**Chương 4: Tổng kết:**

Nội dung của chương 4 sẽ tổng kết bài toán, tóm tắt những kết quả đã đạt được và còn chưa đạt được. Từ đó đề xuất mục tiêu hướng tới cũng như hướng nghiên cứu, phát triển tiếp theo.

# **CHƯƠNG 1: TỔNG QUAN**

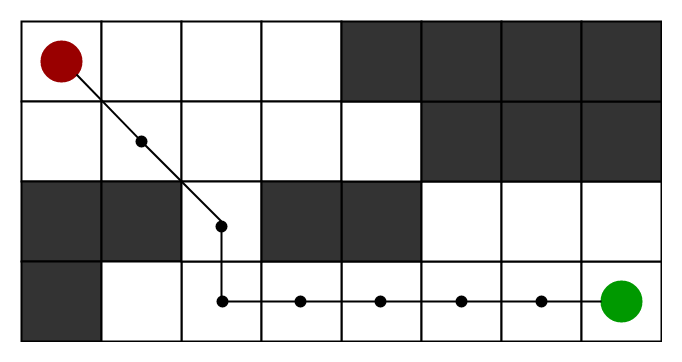
## **1.1 Giới thiệu chung**

Bài toán tìm đường là một bài toán cổ điển trong khoa học máy tính liên quan đến việc tìm đường đi ngắn nhất hoặc hiệu quả nhất giữa hai điểm trong đồ thị hoặc mạng. Bài toán này có nhiều ứng dụng trong thế giới thực như tìm tuyến đường tốt nhất cho phương tiện, người đi bộ hoặc lái xe hoặc thậm chí là rô-bốt điều hướng trong một môi trường. Bài toán tìm đường được giải quyết bằng cách sử dụng các thuật toán khác nhau được thiết kế riêng cho tìm đường chẳng hạn như thuật toán Dijkstra, thuật toán A\*, tìm kiếm theo chiều rộng (BFS), tìm kiếm theo chiều sâu (DFS), ... . Các thuật toán này sử dụng các cách tiếp cận khác nhau để tìm kiếm đường dẫn tối ưu, một số thuật toán sử dụng phương pháp tìm kiếm mù (không có thông tin) trong khi một số khác lại sử dụng phương pháp tìm kiếm có thông tin để nâng cao hiệu quả.

Tuy nhiên, trên thực tế do sự phức tạp của mạng lưới đường trong thế giới thực, các thuật toán tìm kiếm truyền thống như Tìm kiếm theo chiều rộng (BFS) hoặc Tìm kiếm theo chiều sâu (DFS) có thể không đủ hiệu quả để xử lý dữ liệu quy mô lớn. Do đó, thuật toán A\*, biến thể của thuật toán Dijkstra, đã được áp dụng rộng rãi để giải bài toán đường đi ngắn nhất.

## **1.2 Phát biểu bài toán**

Bài toán tìm đường đi trong một lưới 2 chiều có vật cản là một bài toán thuộc lớp bài toán tìm đường đi trong không gian đa chiều. Nó liên quan đến việc tìm một đường đi từ một điểm xuất phát đến một điểm đích trên một lưới hình chữ nhật có kích thước n x m và chứa các ô vuông có thể trống hoặc chứa vật cản. Mục tiêu là tìm đường đi từ điểm xuất phát đến điểm đích sao cho đường đi đó có chi phí thấp nhất hoặc đường đi ngắn nhất



**Hình 1.1** Tìm đường đi trong một lưới 2 chiều có vật cản

Có nhiều thuật toán được sử dụng để giải quyết bài toán tìm đường đi trong một lưới 2 chiều có vật cản. Một số thuật toán phổ biến bao gồm:

1. **Dijkstra's Algorithm:** Thuật toán này sử dụng phương pháp duyệt đồ thị để tìm đường đi ngắn nhất giữa hai điểm. Thuật toán Dijkstra's được sử dụng khi không có thông tin về chi phí hoặc khoảng cách giữa các nút.
2. **A\* Algorithm:** A\* kết hợp giữa thuật toán tìm kiếm đường đi và hàm heuristic để tìm đường đi nhanh nhất.
3. **Breadth-First Search (BFS):** BFS là thuật toán tìm kiếm đường đi đơn giản nhất trong một lưới 2 chiều có vật cản. Thuật toán này sử dụng phương pháp duyệt tầng để tìm đường đi từ điểm bắt đầu đến điểm kết thúc.
4. **Depth-First Search (DFS):** DFS cũng là một thuật toán tìm kiếm đường đi đơn giản, tuy nhiên, nó không đảm bảo tìm ra đường đi ngắn nhất. Thuật toán này sử dụng phương pháp duyệt sâu để tìm đường đi.
5. **Greedy search algorithm:** Thuật toán tìm kiếm tham lam luôn chọn hướng đi có lợi nhất, dù cho nó có dẫn đến kết quả tối ưu hay không. Trong mỗi bước của thuật toán, nó chọn hướng đi dựa trên tiêu chí lựa chọn tối ưu nhất, thường là khoảng cách nhỏ nhất, chi phí thấp nhất hoặc lợi ích cao nhất.

## **1.3 Mục đích và phạm vi đề tài**

***Mục đích:***

* Tìm hiểu thuật toán A\* và phạm vi ứng dụng của nó trong bài toán tìm đường đi trong một lưới 2 chiều có vật cản.
* Xây dựng một chương trình có khả năng tìm đường đi trong một lưới 2 chiều có vật cản theo thuật toán A\*.

***Phạm vi:*** Chương trình thực hiện trong một bản đồ (lưới 2 chiều) với các quy mô

* Rất nhỏ: 10 x 10.
* Nhỏ: 20 x 20.
* Vừa: 40 x 40.
* Lớn: 80 x 80.
* Rất lớn: 160 x 160.

# **CHƯƠNG 2: THUẬT TOÁN A\* ỨNG DỤNG CHO BÀI TOÁN TÌM ĐƯỜNG ĐI TRONG MỘT LƯỚI 2 CHIỀU CÓ VẬT CẢN**

## **2.1 Giới thiệu về thuật toán A\***

Thuật toán A\* là một chiến lược tìm kiếm có thông tin (Informed search) hay còn được gọi là tìm kiếm heuristic sử dụng thêm thông tin từ bài toán để định hướng tìm kiếm, cụ thể là lựa chọn thứ tự mỏ rộng nút theo hướng mau dẫn tới đích hơn.

Nguyên tắc chung của tìm kiếm có thông tin là sử dụng một hàm f (n) để đánh giá độ “tốt” tiềm năng của nút n, từ đó chọn nút n có hàm f tốt nhất để mở rộng trước. Thông thường, độ tốt được đo bằng giá thành đường đi tới đích, do vậy nút có hàm f (n) nhỏ được ưu tiên mở rộng trước.

Trên thực tế, việc xây dựng hàm f (n) phản ánh chính xác độ tốt của nút thường không thực hiện được, thay vào đó ta chỉ có thể ước lượng hàm f (n) dựa vào thông tin có được từ bài toán. Hàm f (n) thường chứa một thành phần là hàm heuristic h(n), là hàm ước lượng khoảng cách từ nút n tới đích.

Thuật toán A\* sử dụng hàm đánh giá f(n) với hai thành phần, thành phần thứ nhất là đường đi từ nút đang xét tới nút xuất phát, thành phần thứ hai là khoảng cách ước lượng tới đích.

**Phương pháp:** Thuật toán A\* sẽ sử dụng hàm 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 đường đi từ nút n đến nút đích; h(n) là hàm heuristic.

Thuật toán A\* yêu cầu hàm h(n) là hàm chấp nhận được (admissible) theo định nghĩa sau.

**Định nghĩa:** Hàm h(n) được gọi là chấp nhận được nếu h(n) không lớn hơn độ dài đường đi thực ngắn nhất từ n tới nút đích.

**Định lý:** Thuật toán A\* sẽ cho kết quả tối ưu nếu ℎ(𝑛) là hàm chấp nhận được

Thuật toán A\* được thể hiện như sau:

**Thuật toán:** A\*(Q, S, G, P, c, h) (𝑄: không gian trạng thái, 𝑆: trạng thái bắt đầu, 𝐺: đích, 𝑃: hành động, c: giá, h: heuristic)

* 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 ← S

While(O không rỗng) do

1. Lấy nút n có f(n) nhỏ nhất ra khỏi O
2. Nếu n thuộc G, return(đường đi tới n)
3. Với mọi m ∈ P(n)
4. g(m) = g(n) + c(m, n)
5. f(m) = g(m) + h(m)
6. Thêm m vào O cùng giá trị f(m)

Return: không tìm được đường đi

## **2.2 Thuật toán A\* ứng dụng cho bài toán tìm đường đi trong một lưới 2 chiều có vật cản**

**Xây dụng các cấu trúc dữ liệu cần thiết:**

**Lớp Point**: Mô phỏng tọa độ một điểm trên bản đồ (lưới 2 chiều)

public class Point {  
 int x;  
 int y;  
}

**Hình 2.1** Lớp Point

**Lớp Node**: Mô phỏng một nút với các thuộc tính: tọa độ, nút cha, giá trị g(n), giá trị f(n).

public class Node {  
 Point point;  
 Node parent;  
 int g;  
 int f;  
}

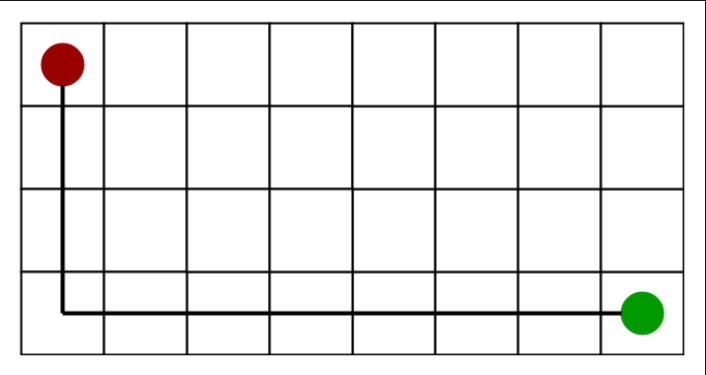
**Hình 2.2** Lớp Node

Đồng thời ta cũng sẽ sử dụng đến **HashSet** (tập hợp + bảng băm) và **PriorityQueue** (hàng đợi ưu tiên) của Java.

**Hàm heuristic:**

Vì bài toán của ta chỉ cho phép đi theo 4 hướng là: lên, trái, phải, xuống nên để tính hàm heuristic ta sẽ sử dụng khoảng cách Manhattan.

**Khoảng cách Manhattan**: Hiệu giá trị tuyệt đối của tọa độ x hiện tại và tọa độ x đích + Hiệu giá trị tuyệt đối của tọa độ y hiện tại và tọa độ y đích



**Hình 2.3** Khoảng cách Manhattan

private int heuristic(Point *point*) {  
 int x = *point*.x, y = *point*.y;  
 int dx = Math.**abs**(x-goal.x);  
 int dy = Math.**abs**(y-goal.y);  
 return dx+dy;  
}

**Hình 2.4** Hàm heuristic tính bằng khoảng cách Manhattan

**Thuật toán:**

*Khởi tạo:*

* Tạo tập đóng (HashSet).
* Tạo tập mở (PriorityQueue).
* Thêm nút đầu tiên (nút bắt đầu) vào tập mở.

// tap dong  
 HashSet<Node> closedList = new HashSet<>();  
 // tap mo  
 PriorityQueue<Node> openList = new PriorityQueue<>();  
  
 Node first = new Node(start, null, 0, heuristic(start));  
 openList.add(first);

**Hình 2.5** Khởi tạo

*Lặp:*

* Lấy ra nút đầu tiên trong tập mở
* Nếu đây là đích, in ra đường đi, dừng thuật toán
* Kiểm tra nút đã có trong tập đóng chưa:
  + Nếu có, kiểm tra xem nếu giá trị f nhỏ hơn thì cập nhật lại nút
  + Nếu chưa, thêm vào tập đóng
* Thăm 4 hàng xóm của nút, với mỗi hàng xóm:
  + Kiểm tra nút đã có trong tập mở chưa:
    - Nếu có, kiểm tra xem nếu giá trị f nhỏ hơn thì cập nhật lại nút
    - Nếu chưa, thêm vào tập mở
  + Kiểm tra nút đã có trong tập đóng chưa:
    - Nếu có, kiểm tra xem nếu giá trị f nhỏ hơn thì cập nhật lại nút
    - Nếu chưa, thêm vào tập đóng
  + Nếu không thuộc cả 2 trường hợp trên: thêm nút vào tập mở
* while(!openList.isEmpty()) {  
   Node node = openList.poll();  
     
   // neu day la dich  
   if(node.point.equals(goal)) {  
   // so buoc de den dich  
   System.out.println(node.f);  
     
   // in ra duong di  
   while(node != null) {  
   System.out.println(node);  
   node = node.parent;  
   }  
   return;  
   }  
   // neu da co trong tap dong  
   if(closedList.contains(node)) {  
   if(node.f < previousF) {  
   closedList.remove(node);  
   closedList.add(node);  
   }  
   }  
   else {  
   closedList.add(node);  
   }  
    
   for(int i = 0; i < 4; i++) {  
   int newX = node.point.x + **moveX**[i];  
   int newY = node.point.y + **moveY**[i];  
   Point newPoint = new Point(newX, newY);   
   // kiem tra nuoc di hop le va o nay co bi chan khong  
   if(isValid(newPoint) && !isBlocked(newPoint)) {  
   Node neighbour = new Node(newPoint, node, node.g + 1, node.g + 1 + heuristic(newPoint));  
     
   // neu da co trong tap mo  
   if(openList.contains(neighbour)) {  
   if(neighbour.f < previousF) {  
   openList.remove(neighbour);  
   openList.add(neighbour);  
   }  
    
   }  
   // neu da co trong tap dong  
   else if(closedList.contains(neighbour)) {  
   if(neighbour.f < previousF) {  
   closedList.remove(neighbour);  
   closedList.add(neighbour);  
   }  
   }  
   // neu khong thi them vao tap mo  
   else {  
   openList.add(neighbour);  
   }  
   }  
   }  
  }

**Hình 2.6** Thuật toán

# **CHƯƠNG 3: THỰC NGHIỆM VÀ KẾT QUẢ**

## **3.1 Mô tả dữ liệu**

Chương trình thực hiện dựa trên dữ liệu từ người dùng nhập vào, cụ thể hơn là thông qua giao diện chương trình. Người dùng có thể tùy chọn điểm bắt đầu, điểm kết thúc, các ô vật cản.



**Hình 3.1** Giao diện chính

## **3.2 Kết quả thực nghiệm**

Dưới đây là các kết quả khi chạy chương trình với các quy mô bản đồ khác nhau

A picture containing timeline

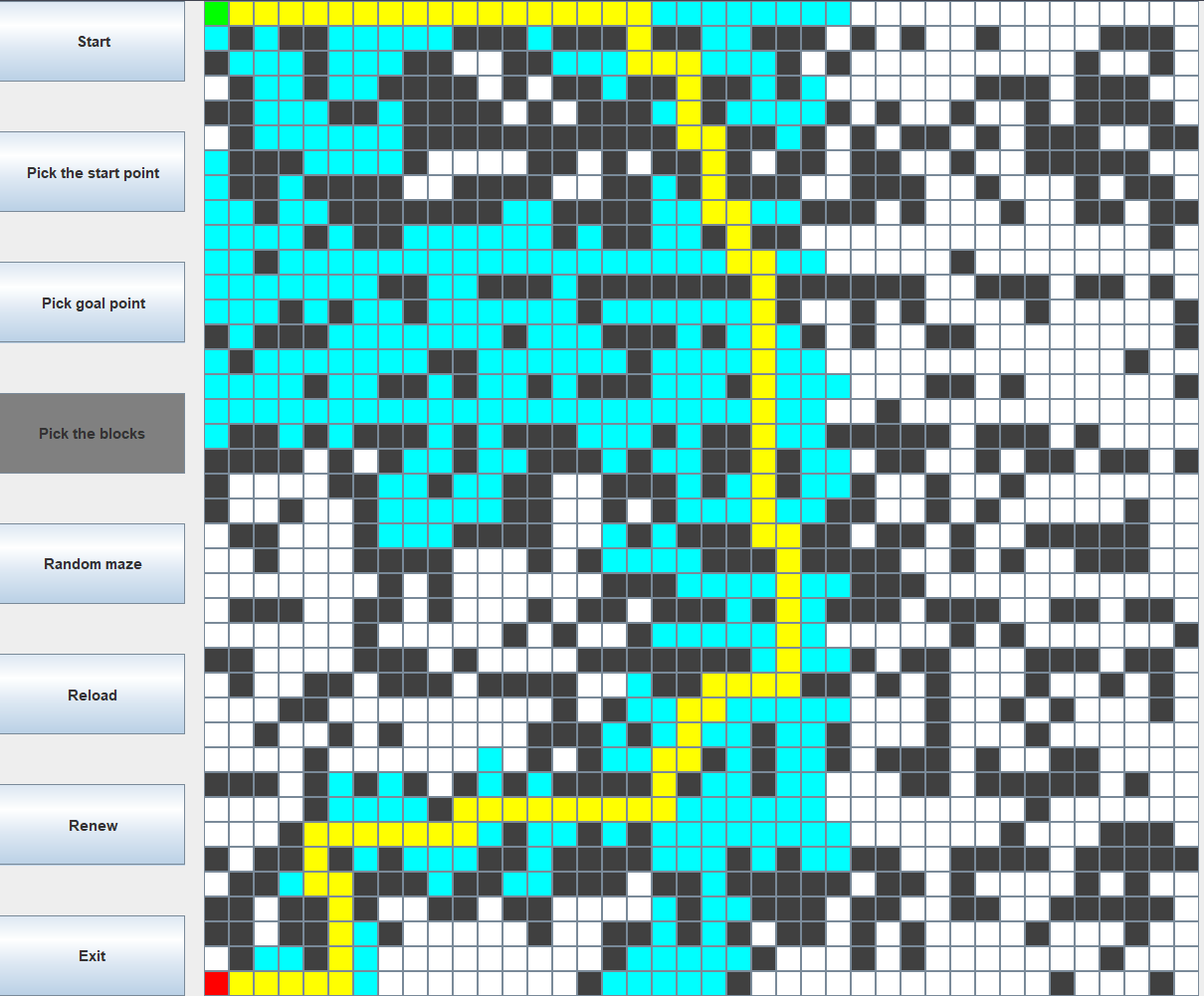
Description automatically generated

**Hình 3.2** Bản đồ 10 x 10

A screenshot of a computer

Description automatically generated with medium confidence

**Hình 3.3** Bản đồ 20 x 20



**Hình 3.4** Bản đồ 40 x 40

Qr code

Description automatically generated

**Hình 3.5** Bản đồ 80 x 80

Diagram

Description automatically generated

**Hình 3.6** Bản đồ 160 x 160

# **CHƯƠNG 4: KẾT LUẬN**

## **4.1 Kết luận**

Kết quả nghiên cứu và ứng dụng thuật toán A\* vào bài toán tìm đường đi ngắn nhất từ điểm xuất phát đến điểm đích trong một lưới không gian 2 chiều có vật cản đã đạt được hai yêu cầu chính của đề bài đặt ra: Tìm đường đi từ điểm bắt đầu đến điểm kết thúc và số bước di chyển là ngắn nhất (do ta dùng khoảng cách Manhattan => hàm heuristic là hàm chấp nhận được)

Kết quả của nghiên cứu cho thấy tốc độ tìm kiếm đường đi ngắn nhất của thuật toán A\* so với các thuật toán khác như Tìm kiếm theo chiều rộng (BFS), Thuật toán Dijkstra, … là tốt hơn. Tuy nhiên mặc dù kết quả đạt được đã chính xác, với cách cài đặt thuật toán như đã trình bày, thuật toán chỉ khả thi khi chạy trong bản đồ tối đa là 1000 x 1000. Như vậy có thể thấy cách cài đặt này không có nhiều tính thực tiễn, chỉ phù hợp với một chương trình nhỏ dùng để demo thuật toán.

## **4.2 Hướng phát triển của đồ án trong tương lai**

Đề tài được lấy cảm hứng từ video: AI Learns to Escape (Deep Reinforcement Learning), link: <https://www.youtube.com/watch?v=v3UBlEJDXR0>. Tuy nhiên do chưa học về Unity cũng như Deep Reinforcement Learning nên đề tài mới chỉ dừng ở mức ứng dụng thuật toán A\* để tìm đường đi trong lưới 2 chiều. Trong tương lai, nếu như có thời gian, nhóm sẽ cố gắng phát triển đề tài theo hướng như trong video: Đó là xây dựng một AI có khả năng tự tìm đường đi + giải đố để tự thoát khỏi màn chơi.

# **Tài liệu tham khảo**

|  |  |
| --- | --- |
| [1] | Từ Minh Phương. Giáo trình Nhập môn trí tuệ nhân tạo. Nhà xuất bản  Thông tin và truyền thông, 2016. |
| [2] | Geeksforgeeks.org, A\* Search Algorithm, https://www.geeksforgeeks.org /a-search-algorithm, truy cập tháng 3.2023. |