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: Ứng dụng thuật toán A* cho trò chơi ghép tranh

LỚP : N09
Số thứ tự nhóm: 09

Vũ Xuân Tùng	**MSSV: D21DCCN778**
Phan Văn Tú	**MSSV: D21DCCN754**
Đỗ Thành Công	**MSSV: D21DCCN022**
Vũ Anh Tuấn	**MSSV: B21DCCN763**

Giảng viên hướng dẫn: Ths. Hoài Thư

HÀ NỘI, 05/2023

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin cảm ơn cô Vũ Hoài Thư đã tận tình hướng dẫn, chỉnh sửa và góp ý để chúng em có thể hoàn thành đề tài này. Cô đã giúp chúng em tích lũy rất nhiều kiến thức và có cái nhìn xâu sắc hơn về bộ môn Nhập môn trí tuệ nhân tạo. Do kiến thức còn hạn hẹp nên trong quá trình thực hiện đề tài không thể tránh khỏi những sai sót. Vì vậy chúng em rất mong nhận được sự góp ý và đánh giá của cô để có thể hoàn thiện đề tài hơn.

Kính chúc cô sức khỏe, hạnh phúc và thành công trên con đường giảng dạy!

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

Viết tắt	Tên tiếng Anh	Tên tiếng Việt
API	Application Programming Interface	Giao diện lập trình ứng dụng
HTML	HyperText Markup Language	Ngôn ngữ đánh dấu siêu văn bản
CNTT		Công nghệ thông tin
SV		Sinh viên
BFS	Breadth-first-Search	Tìm kiếm theo chiều rộng
ID	Inversion Distance	
MD	Manhattan Distance	
WD	Walking Distance	

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Trong cuộc sống hàng ngày, việc tối ưu hóa đường đi là một vấn đề quan trọng đối với nhiều người. Từ việc di chuyển trong thành phố đông đúc đến quản lý dự án trong công việc, việc tìm ra đường đi ngắn nhất không chỉ giúp tiết kiệm thời gian và năng lượng mà còn mang lại nhiều lợi ích khác.

Trong lĩnh vực trí tuệ nhân tạo, bài toán này thường được mô hình hoá dưới dạng một vấn đề tìm đường đi ngắn nhất trong không gian trạng thái. Thuật toán A* được sử dụng phổ biến để giải quyết việc đó. Việc áp dụng thuật toán A* vào bài toán ghép tranh là một ví dụ cụ thể về việc tối ưu hóa đường đi. Trong trò chơi này, người chơi cần di chuyển các ô tranh để đưa chúng về vị trí đúng. Việc sử dụng thuật toán A* giúp tìm ra đường đi ngắn nhất từ trạng thái ban đầu đến trạng thái mục tiêu một cách hiệu quả và nhanh chóng.

Ngoài ra, việc giải quyết bài toán tối ưu hóa đường đi cũng có thể được áp dụng vào nhiều lĩnh vực khác nhau. Trong logistics và vận tải, việc tối ưu hóa tuyến đường và lộ trình giao hàng giúp giảm thiểu thời gian và giá thành. Trong robotics và trí tuệ nhân tạo, thuật toán tìm đường đi ngắn nhất cũng được sử dụng để điều khiển robot di chuyển trong môi trường phức tạp

1.2 Mục tiêu và phạm vi đề tài

Trong lĩnh vực trí tuệ nhân tạo và tối ưu hóa đường đi, nhiều nghiên cứu đã được tiến hành để giải quyết bài toán tìm đường đi ngắn nhất trong các mạng lưới và không gian trạng thái. Các phương pháp truyền thống như thuật toán Dijkstra và Bellman-Ford đã được áp dụng rộng rãi trong thực tế, nhưng chúng thường gặp phải nhược điểm như yêu cầu nhiều tài nguyên tính toán và thời gian tính toán lớn đối với các đồ thị lớn.

Trong thời gian gần đây, thuật toán A* đã trở thành một lựa chọn phổ biến và hiệu quả cho bài toán này. A* kết hợp giữa giá thành thực tế từ trạng thái ban đầu đến trạng thái hiện tại và ước lượng giá thành từ trạng thái hiện tại đến trạng thái mục tiêu, giúp tìm ra đường đi ngắn nhất một cách hiệu quả với thời gian tính toán ít hơn.

Mục tiêu của đề tài là nghiên cứu và phát triển một phần mềm sử dụng thuật toán A* để giải quyết bài toán ghép tranh một cách hiệu quả và đáng tin cậy. Phần mềm sẽ được thiết kế để xử lý các trường hợp đặc biệt và cải thiện hiệu suất của thuật toán A* trong việc tìm kiếm đường đi ngắn nhất. Điều này có thể đạt được

thông qua việc tối ưu hóa hàm ước lượng và xử lý các trường hợp đặc biệt trong quá trình tìm kiếm.

1.3 Định hướng giải pháp

1.3.1 Định hướng

Trong nghiên cứu này, chúng em áp dụng thuật toán A* để giải quyết bài toán game ghép tranh. Thuật toán A* là một thuật toán tìm kiếm thông minh, sử dụng kỹ thuật tìm kiếm theo lựa chọn tốt nhất để xác định đường đi tối ưu từ trạng thái ban đầu đến trạng thái mục tiêu. Đặc điểm nổi bật của A* là khả năng kết hợp giữa giá thành thực tế đã di chuyển từ trạng thái hiện tại đến trạng thái mục tiêu và một hàm heuristic để ước lượng giá thành còn lại.

1.3.2 Mô tả giải pháp

Thuật toán A* được áp dụng vào bài toán ghép tranh để tìm kiếm đường đi ngắn nhất từ trạng thái ban đầu đến trạng thái mục tiêu. Quá trình giải quyết bài toán được mô tả như sau: Đầu tiên chúng em xác định trạng thái ban đầu là trạng thái của bức tranh khi bắt đầu ghép và trạng thái kết thúc là trạng thái mà bức tranh đã hoàn thành. Sau đó chúng em xác định hàm giá thành (g(n)): Hàm giá thành trong trường hợp này có thể được đo lường bằng cách tính toán sự khác biệt giữa mảnh ghép ở trạng thái ban đầu so với trạng thái hiện tại. Mục tiêu là tìm cách di chuyển các mảnh ghép sao cho tổng giá thành là nhỏ nhất. Để tối ưu hóa hiệu suất của thuật toán, chúng em lựa chọn một hàm heuristic (h(n)) phù hợp như khoảng cách Manhattan. Tiếp theo, chúng em áp dụng thuật toán A* để tìm kiếm đường đi ngắn nhất từ trạng thái ban đầu đến trạng thái kết thúc. Trong quá trình này, chúng em duyệt qua các trạng thái có thể đạt được từ trạng thái hiện tại và chọn ra đường đi có tổng giá thành nhỏ nhất. Trong quá trình thực hiện thuật toán, có thể cần thực hiện các biện pháp tối ưu để giảm thiểu thời gian tính toán và tối ưu hóa chi phí. Đồng thời, kiểm tra điều kiện dừng để đảm bảo rằng thuật toán dừng lại khi đã tìm ra lời giải tối ưu hoặc đã đạt đến điều kiện kết thúc.

1.4 Bố cục bài tập lớn

Phần còn lại của báo cáo bài tập lớn này được tổ chức như sau.

Chương 2, cơ sở lý thuyết chúng em sẽ giới thiệu bài toán ghép tranh, thuật toán A* và các hàm heuristic.

Chương 3, chúng em sẽ trình bày về giao diện website của trò chơi và so sánh sự tối ưu giữa cách hàm heuristic.

Chương 4, kết luận và hướng phát triển, chúng em đánh giá lại quá trình và đề xuất các hướng phát triển

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Giới thiệu bài toán ghép tranh

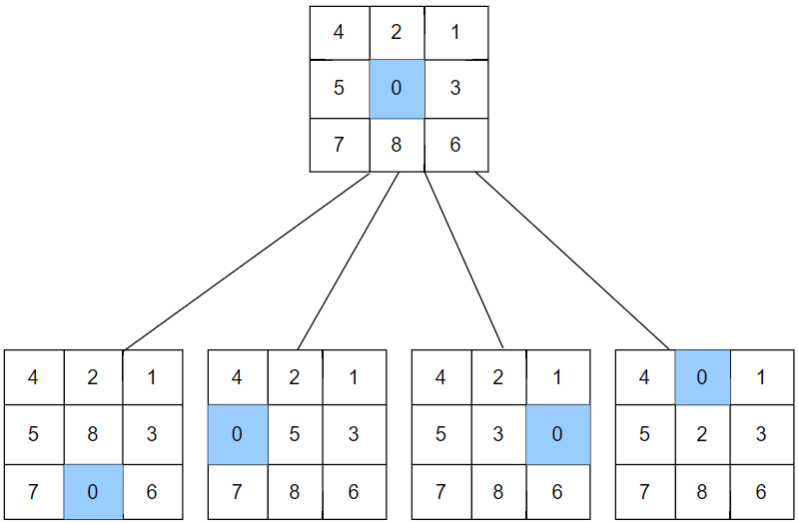
Bài toán Ghép Tranh, còn được biết đến với tên gọi 8-puzzle, 12-puzzle, 15-puzzle, là một bài toán logic cổ điển và phổ biến trong lĩnh vực trí tuệ nhân tạo và trò chơi logic. Bài toán xuất phát từ thế kỷ 19 và đã thu hút sự quan tâm của nhiều nhà nghiên cứu trong các thập kỷ sau này.

Trong Bài toán Ghép Tranh, Bảng gồm 1 ô trống và n - 1 ô chứa các số trong phạm vi [1, n - 1] (Với n là tích của số hàng và số cột của bảng tranh). Mục tiêu của bài toán là di chuyển các ô số để sắp xếp chúng theo một trình tự nhất định từ trái sang phải và từ trên xuống dưới, để tạo thành một trạng thái mục tiêu nhất định.

Bài toán Ghép Tranh có thể được giải quyết bằng các phương pháp tìm kiếm như thuật toán tham lam, IDA, A*,... tận dụng các hàm heuristic để ước lượng giá thành từ trạng thái hiện tại đến trạng thái đích. Thuật toán A* là một trong những phương pháp phổ biến nhất được sử dụng để giải quyết bài toán này, với khả năng tìm kiếm đường đi tối ưu trong thời gian hợp lý.

Bài toán Ghép Tranh không chỉ là một trò chơi giải trí thú vị mà còn có nhiều ứng dụng thực tiễn trong lĩnh vực trí tuệ nhân tạo và robot học. Nó được sử dụng để nghiên cứu các phương pháp tìm kiếm thông minh, hàm heuristic, và tối ưu hóa thuật toán.

Ví dụ về các bước có thể di chuyển của ô trống



Hình 2.1: Các bước di chuyển của ô trống

Như hình trên vậy khi ta click vào ô số 8 thì nó sẽ di chuyển lên. Còn click vào

ô số 5 thì nó sẽ sang phải. Click vào ô số 3 thì nó sẽ sang trái và click vào ô số 2 thì nó sẽ đi xuống.

2.2 Thuật toán A*

2.2.1 Giới thiệu

Thuật toán A* được phát triển vào năm 1986 bởi Peter Hart, Nils Nilson và Bertram Raphael.Thuật toán này là một phương pháp tìm kiếm đường đi trong đồ thị. Sử dụng một hàm đánh giá heuristic, A* ước lượng tuyến đường tốt nhất thông qua mỗi nút và duyệt các nút theo thứ tự ước lượng heuristic.

Trong bài toán tìm đường đi, A* xây dựng các tuyến đường từ điểm xuất phát tới khi tìm thấy đường đi tới điểm đích. Tuy nhiên, cũng như tất cả các thuật toán tìm kiếm có thông tin nó chỉ xây dựng các tuyến đường có vẻ dần về đích. Để đánh giá tuyến đường nào có khả năng dẫn tới đích, A* sử dụng hàm đánh giá heuristic về khoảng cách từ điểm hiện tại tới đích, thường là khoảng cách đường chim bay(là cách đo khoảng cách giữa hai điểm trên đồ thị theo đường thẳng, mà không cần đi theo đường đi cụ thể trên bề mặt của đồ thị.

Bên cạnh đó điểm khác biệt của A* đối với tìm kiếm theo lựa chọn tốt nhất(Best-First Search) là nó còn tính đến khoảng cách đã đi qua. Điều đó làm cho A* đầy đủ và tối ưu, nghĩa là A* sẽ luôn tìm thấy đường đi ngắn nhất nếu tồn tại một đường đi như thế.

Tuy nhiên, trong một môi trường dạng mê cung, cách duy nhất để đến đích có thể là trước hết phải đi về phía xa đích và cuối cùng mới quay trở lại. Trong trường hợp đó, việc thử các nút theo thứ tự "gần đích hơn thì được thử trước" có thể gây tốn thời gian.

2.2.2 Mô tả thuật toán

Thuật toán A* lưu giữ một tập các lời giải chưa hoàn chỉnh, nghĩa là các đường đi qua đồ thị, bắt đầu từ nút xuất phát. Thuật toán A* lưu trữ các tập các lời giải trong một hàng đợi ưu tiên, thứ tự ưu tiên được quyết định bởi hàm:

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

Trong đó:

- g(n): là giá thành của đường đi cho đến thời điểm hiện tại, nghĩa là tổng trọng số của các cạnh đã đi qua.
- h(n): là hàm đánh giá heuristic về giá thành nhỏ nhất để đến đích từ điểm n
- f(n): được sử dụng để đánh giá tổng chi phí của một đường đi từ điểm bắt đầu

đến đích, qua một điểm trung gian. f(n) có giá trị càng thấp thì độ ưu tiên của n càng cao

Thuật toán A*

A*(Q, S, G, P, c, h)

- Q: không gian trạng thái, S: trạng thái bắt đầu, G: trạng thái đích
- P: hành động
- g(m) là giá trị thực tế đã đi từ trạng thái bắt đầu S đến trạng thái m.
- g(n) là giá trị thực tế đã đi từ trạng thái bắt đầu S đến trạng thái n (trạng thái hiện tại).
- c: hàm tính giá thành
- h: hàm heuristic

Đầu vào: Bài toán tìm kiếm, hàm heuristic h
Đầu ra: Trạng thái đích
Khởi tạo: O $\leftarrow$ S (O: danh sách các nút mở)

while (O $\neq$ Ø) do:

(a) Lấy nút n có $f(n)$ là nhỏ nhất khỏi O

(b) if $n \in G$, return (đường đi tới n)

(c) Với mọi $m \in P(n)$:

i. $g(m) = g(n) + c(n, m)$

ii. $f(m) = g(m) + h(m)$

iii. thêm m vào O cùng với giá trị $f(m)$

return: Không tìm được đường đi.

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

Hàm heuristic **chấp nhận được** khi: với mọi nút n thì $h(n) \leq h^*(n)$, với $h^*(n)$ là giá thành thực để đi từ n tới đích.

Ví dụ: hàm heuristic đo khoảng cách đường chim bay là chấp nhận được.

2.3 Ứng dụng thuật toán A* vào bài toán ghép tranh

2.3.1 Biểu diễn bài toán ghép tranh như bài toán tìm đường

- Mỗi trạng thái là một cấu hình cụ thể của các mảnh ghép trên bảng. Khi di chuyển ô trống, vị trí các mảnh ghép sẽ thay đổi tao ra một trạng thái mới.

- Thực hiện đánh số các mảnh ghép theo thứ tự. Ví dụ đối với bảng 3x3

Hình 2.2: Các ô tranh được đánh số theo thứ tự

- Đối với ô trống, chọn số 0 để đại diện. Ta có thể biểu diễn trạng thái trên của bài toán trong một mảng một chiều khi đọc từ trái sang phải và từ trên xuống dưới [1, 2 , 3 , 4 , 5 , 6 , 7, 8, 0].

2.3.2 Xây dựng hàm heuristic

Hàm heuristic là hàm ước lượng chi phí cần thiết để đi được đến trạng thái đích. Đối với bài toán ghép tranh có thể sử dụng các heuristic sau:

- **Số lượng mảnh ghép sai vị trí (Misplaced - Hamming Distance):** Đây có lẽ là heuristic đơn giản nhất, dựa trên việc đếm số vị trí đứng không đúng so với trạng thái đích. Hiệu quả của heuristic này là tệ và không được khuyên dùng do không sử dụng thêm bất kì thông tin nào về khoảng cách hay số ô cần đi để đưa trạng thái về đích.

  Ví dụ với trạng thái sau:

Hình 2.3: Misplaced - Hamming Distance

  Biểu diễn trạng thái này thành dạng mảng một chiều [1, 2 , 3 , 4 , 5 , 6 , 0, 7, 8]. So với trạng thái đích vị trí ô 7 và 8 đang sai do đó ta có h(n) = 2

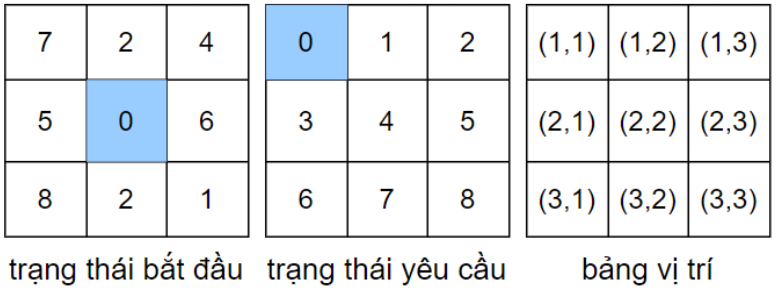
- **Khoảng cách Manhattan giữa các mảnh ghép (Manhattan Distance):**

  Heuristic này cộng khoảng cách manhattan của tất cả các ô từ vị trí hiện tại đến vị trí đích. Ước tính khoảng cách bằng cách sử dụng độ lệch theo chiều ngang và dọc trên đồ thị.

$$m_d = |x_a - x_b| + |y_a - y_b|$$

Với x_a và y_a lần lượt là vị trí cột và hàng của ô ở vị trí hiện tại, x_b và y_b lần lượt là vị trí cột và hàng của ô ở vị trí đích.

Ví dụ:



Hình 2.4: Manhattan Distance

khoảng cách Manhattan của 1 là md1 = |3-1| + |3-2| = 3

khoảng cách Manhattan của 2 là md1 = |1-1| + |2-3| = 1

khoảng cách Manhattan của 3 là md1 = |3-2| + |2-1| = 2

....

khoảng cách Manhattan của 8 là md1 = |3-3| + |3-1| = 2

Vậy h(n) = 3 + 1 + 2 + . . . + 2 = 18

- **Khoảng cách Inversion (Inversion Distance):**

  Một inversion xảy ra khi có một số lớn hơn xuất hiện trước một ô chứa số nhỏ hơn.

  Ví dụ ở trạng thái đích không có một inversion nào cả. Nhưng với [1, 2, 3, 4, 8, 6, 7, 5, 0], thì ta có 8 lớn hơn 3 số đằng sau nó (không xét số 0 vì 0 là ô trống), 6 lớn hơn 5, 7 lớn hơn 5 từ đó inversion trong trường hợp này là 5.

  Inversion chỉ thay đổi khi ta di chuyển ô trống lên hoặc xuống, còn không thay đổi khi ta dịch trái hoặc phải (điều này có thể dễ dàng nhìn ra). Với bài toán 3x3, số inversion có thể +2, -2 hoặc không thay đổi khi ta di chuyển ô trống lên hoặc xuống. Vậy nếu ta coi rằng số lần di chuyển lên di chuyển xuống cần thiết để giải bài toán là số lần di chuyển lên xuống cần thiết để inversion về 0 (trạng thái đích có inversion bằng 0). Từ đó ta có thể tính được số bước di chuyển lên xuống cần thiết bằng (inversion / 2). Từ cách tính số bước di chuyển lên xuống cần thiết để giải bài toán, ta có thể áp dụng tương tự

để tìm số bước di chuyển trái phải cần thiết để giải bài toán bằng cách đánh số bài toán theo chiều từ trên xuống dưới từ trái sang phải và thực hiện xét duyệt bài toán theo chiều đó.

Sau khi tính được số bước cần thiết di chuyển lên xuống và trái phải để giải bài toán. Ta có:

$$h(n) = vertical + horizontal$$

Với vertical là số bước tối thiểu để di chuyển lên xuống

horizontal là số bước tối thiểu để di chuyển trái phải

- **Kết hợp giữa Manhattan và Linear Conflict:**

Khi có hai ô t_j và t_k, ở cùng dòng và vị trí đúng của 2 ô đó cũng nằm trên dòng đó, t_j nằm bên phải t_k và vị trí đúng của t_j nằm bên trái vị trí đúng của t_k thì hai ô đó được gọi là một trường hợp của Linear Conflict

Ví dụ:

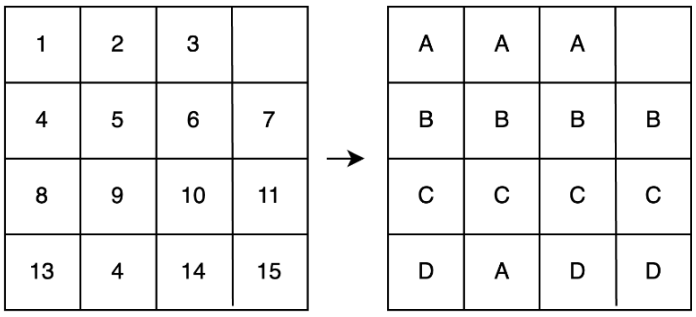
Hình 2.5: Manhattan và Linear Conflict:

Dễ dàng nhận thấy ô số 2 và 3 ở đây chính là một ví dụ của Linear Conflict. Nếu chỉ sử dụng Manhattan ở đây thì h(n) = 2, nhưng để đổi vị trí của 2 ô này không thể nào chỉ cần đến 2 bước, ít nhất phải có thêm 2 bước nữa. Từ đó mỗi khi có một trường hợp của Linear Conflict thì ta thêm vào h(n) += 2.

- **Walking Distance:**

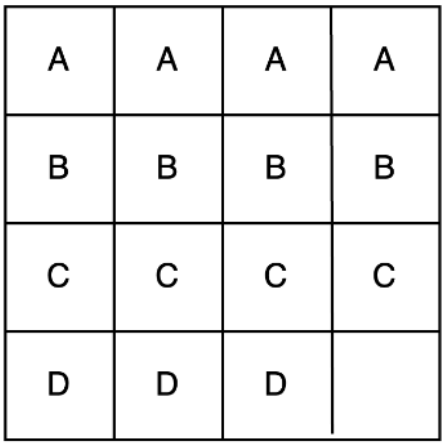
Đối với MD và ID cả hai đều hoạt động bằng cách sử dụng các thông tin như khoảng cách để đưa một ô về vị trí đích của nó hay là số bước di chuyển lên xuống trái phải ít nhất để giải được bài toán.

Vấn đề của MD là nó không xét đến sự tương tác giữa các ô khác thứ mà ID sử dụng, còn ID thì lại không quan tâm đến vị trí đúng của các ô. Nhưng ta có thể kết hợp hai ưu điểm của hai heuristics trên để tạo ra một heuristic tốt hơn.



Hình 2.6: Chuyển từ bảng số bình thường sang bảng dùng để xét

Giả sử ta đổi các ô cùng một dòng ở vị trí đúng của nó thành cùng một giá trị, ví dụ ở trường hợp dòng là hàng, ta coi mỗi hàng là một hộp, ta có thể lấy một giá trị từ hàng này chuyển sang hàng kế cận nếu hàng đó chứa một ô trống. Ví dụ như trường hợp trên, bất cứ giá trị nào ở hàng 2 từ trên xuống, đều có thể được chuyển qua ô trống phía trên.



Hình 2.7: Bảng dùng để xét ở trạng thái đích

Từ giả thuyết trên, sau khi trả hết vị trí của các ô về đúng vị trí theo hàng của nó, và đếm số bước ta sẽ tìm được số bước di chuyển lên xuống ít nhất cần để đưa tất cả các ô về vị trí đúng theo hàng của nó.

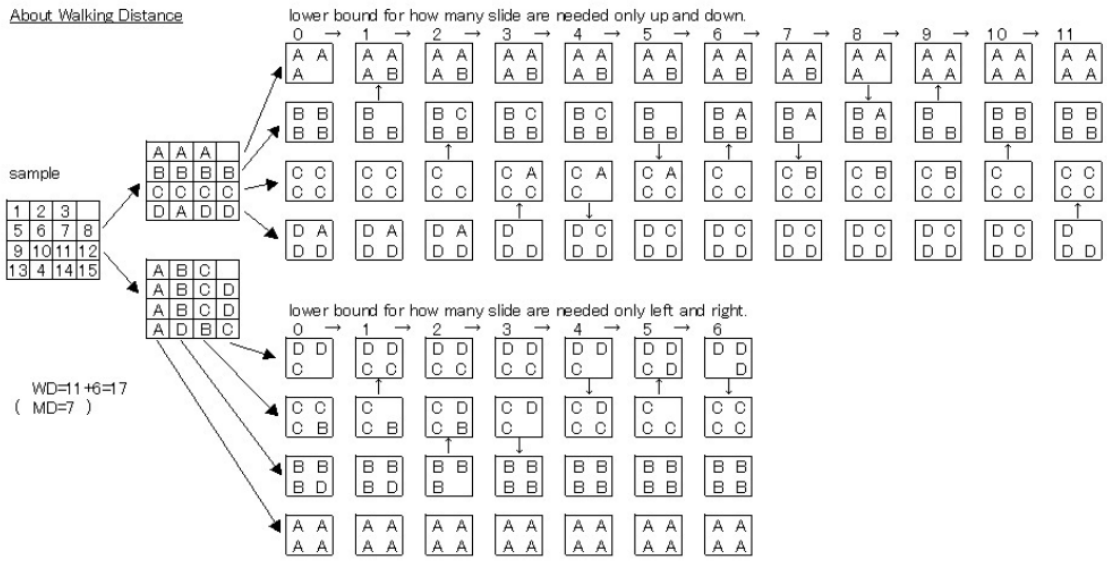
Tương tự với cột, ta cũng có thể tìm ra được số bước di chuyển trái phải ít nhất để đưa các ô về vị trí đúng theo cột của nó.

Từ đó ta có:

$$h(n) = vertical + horizontal$$

verical là số bước di chuyển lên xuống ít nhất cần để đưa tất cả các ô về vị trí đúng theo hàng của nó.

horizontal là số bước di chuyển trái phải ít nhất để đưa các ô về vị trí đúng theo cột của nó.



Hình 2.8: Minh hoạ

Để tìm được Walking Distance nhanh chóng, chúng em chạy BFS từ trạng thái đích sau khi đổi các ô cùng về cùng một giá trị, để tìm ra tất cả các trạng thái có thể xảy ra khi xét bài toán theo hàng và cột.

2.3.3 Chi tiết thuật toán

1. Khởi tạo:
   - Đặt trạng thái ban đầu vào Open list (danh sách mở).
   - Thiết lập g(n) = 0 cho trạng thái ban đầu (chi phí từ gốc đến trạng thái ban đầu).
2. Duyệt và mở rộng các trạng thái:

   Lặp lại cho đến khi Open list trống hoặc tìm thấy trạng thái đích:
   - Chọn trạng thái n từ Open list có giá trị f(n) nhỏ nhất.
   - Nếu n là trạng thái đích, dừng lại và trả về lời giải.
   - Di chuyển trạng thái n từ Open list sang Closed list (danh sách đóng).
   - Mở rộng n bằng cách tạo các trạng thái kề (các cấu hình mảnh ghép sau khi di chuyển ô trống về các hướng).
   - Với mỗi trạng thái kề m: Nếu m không có trong Closed list:

- Tính g(m), h(m), và f(m).(với bài toán ghép tranh c(n,m) = 1 vì ô tranh chỉ di chuyển 1 bước)
- Nếu m chưa có trong Open list, thêm m vào Open list.
- Nếu m đã có trong Open list, cập nhật nếu g(m) mới nhỏ hơn g(m) hiện tại.

3. Quản lý Open list và Closed list:
   - Sử dụng cấu trúc dữ liệu ưu tiên (priority queue) để quản lý Open list.
   - Closed list để lưu trữ các trạng thái đã được mở rộng, tránh việc lặp lại trạng thái.

2.3.4 Ví dụ minh họa

Ta có hình ảnh sau:

Hình 2.9: Ảnh minh họa

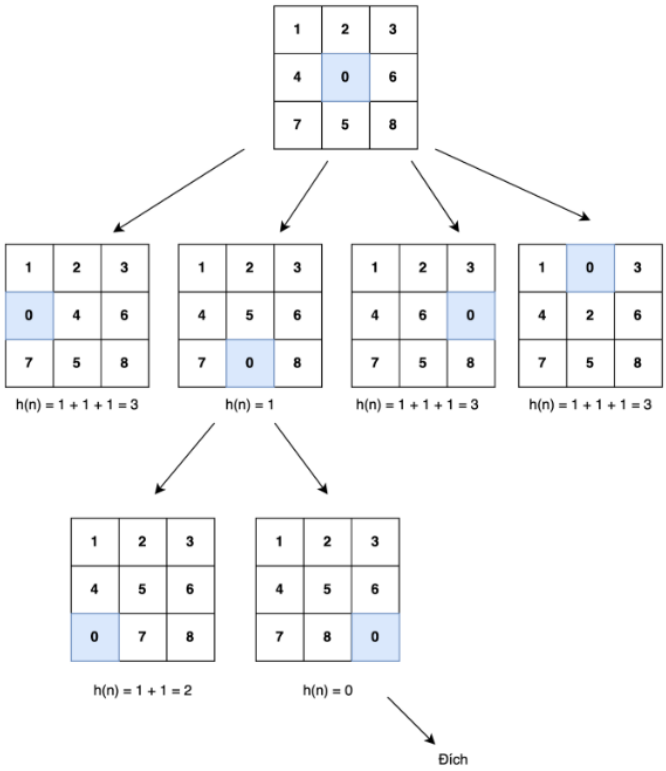
Xét hình ảnh bên trên, sau khi chia ô theo bảng 3x3 và đánh số ta được trạng thái đích như sau:

Hình 2.10: Chia ảnh minh họa theo bảng 3x3 ô

Ví dụ sử dụng thuật toán A* với Manhattan Distance để giải quyết bài toán sau:

Hình 2.11: Ảnh sau khi shuffle

Mô tả các bước giải bài toán:



Hình 2.12: Các bước giải bài toán dùng Heuristic Manhattan

CHƯƠNG 3. THỰC NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

1. Giới thiệu

Trong phần này, chúng em sẽ trình bày về việc triển khai và xây dựng phần mềm ứng dụng A* cho bài toán ghép tranh. Chúng em chọn sử dụng ReactJS và TypeScript để xây dựng một ứng dụng Web, cung cấp một giao diện thân thiện để tương tác với thuật toán.

2. Cấu trúc ứng dụng

Ứng dụng bao gồm 3 phần chính bao gồm:

- Giao diện tương tác
- Thuật toán A*
- Heuristics

Để ba phần này có thể tương tác với nhau một cách trơn tru, chúng cần có một ngôn ngữ chung và ở đây chính là mảng một chiều thể hiện cho trạng thái hiện tại của trò chơi ghép tranh. Với một bài toán ghép tranh dạng 3x3, ta cần mảng một chiều có độ dài là 9, tương tự 3x4 sẽ là 12, 4x4 là 16. Mảng sẽ chứa các số từ 0 tăng dần. Ví dụ với dạng 3x3, mảng sẽ chưa các số từ 0 -> 8. Trò chơi ghép tranh cần có một ô để trống, bọn em chọn số 0 đại diện cho ô trống đó.

3. Giao diện tương tác

Để giao diện tương tác được trực quan và rõ ràng, chúng em chia thành hai phần bao gồm:

- Bàn ghép tranh
- Bảng điều khiển

Đối với bàn ghép tranh, chúng em sẽ thiết kế có thể chọn ảnh hay tải ảnh mà mình muốn sử dụng, thay vì cung cấp một số ảnh mẫu. Chúng em sử dụng canvas (HTML element) để vẽ các ô hình, đúng với vị trí của nó.

Đối với bảng điều khiển chúng em đã quyết định thêm các lựa chọn về tỉ lệ trò chơi, các nút sử dụng để tự động đảo hình, reset hình về trạng thái ban đầu hay xoá hình hiện tại. Tiếp theo chúng em thiết kế thêm một menu để có thể chọn các heuristic muốn sử dụng. Và không thể thiếu là nút bắt đầu tìm đường. Sau khi tìm xong sẽ hiện lên các thông tin cần thiết cho việc so sánh các thuật toán heuristics.

4. Thuật toán A*

Việc cài đặt thuật toán A* sẽ giống như phần phân tích đã nêu. Ngoài ra cần phải xây dựng thêm các phương thức con như hàm sinh các trạng thái tiếp theo của một trạng thái, hàm kiểm tra trạng thái đích.

Để người dùng có thể thay đổi các heuristics nhanh chóng, thuật toán A* sẽ có các đầu vào sau:

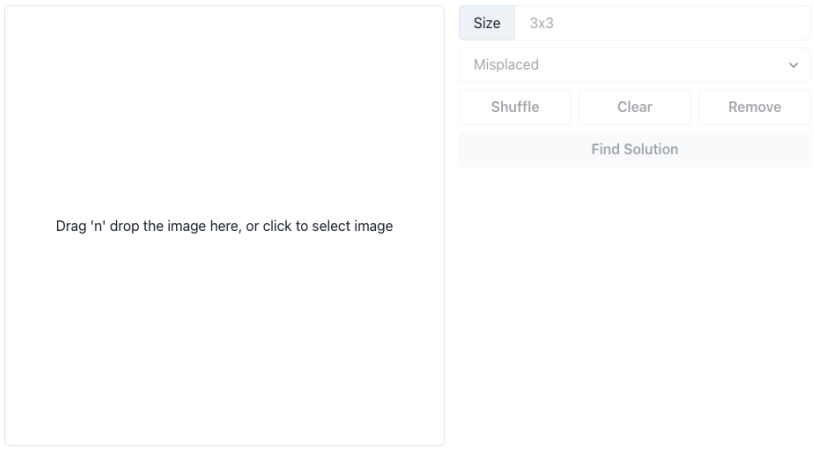
startState (trạng thái bắt đầu), heuristic (hàm đánh giá), rows (số hàng), columns (số cột). Và đầu ra của thuật toán sẽ là một danh sách các trạng thái từ trạng thái bắt đầu đến trạng thái đích. Thuật toán cần duy trì bốn tập hợp bao gồm:

- Visited: Tập chứa các trạng thái đã kiểm tra.
- Queue: Tập chứa các trạng thái chứa tiếp theo sẽ được xét và được sắp xếp theo giá thành.
- Parents: Tập chứa các trạng thái cha của một nút nào đó đã được xét.
- Costs: Tập chứa giá thành để đi đến một trạng thái nào đó từ trạng thái ban đầu.

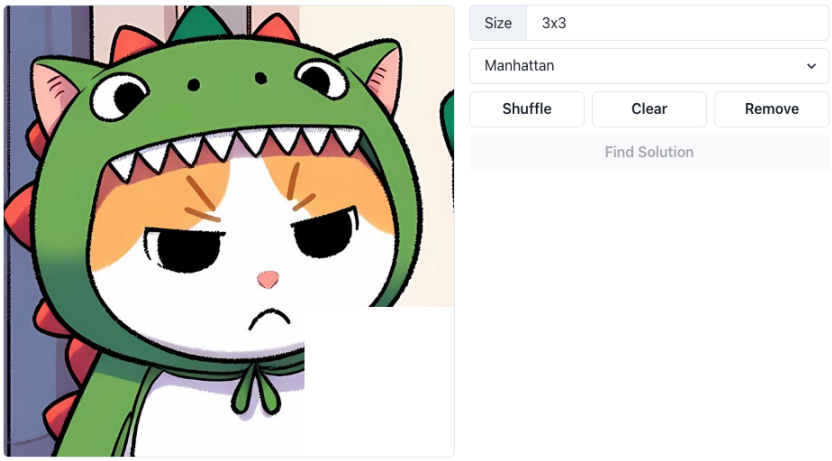
Với hàm sinh các trạng thái tiếp theo của trạng thái n với đầu vào là trạng thái n, thực hiện kiểm tra xem vị trí ô trống ở đâu và thực hiện tính toán ra các trạng thái tiếp theo phù hợp. Lấy ví dụ trạng thái hiện tại là [1, 2, 3, 4, 5, 6, 7, 0 ,8] đối với bàn 3x3. Ta có thể tính toán được là vị trí của ô trống đang ở hàng 3 và cột 2. Vậy các trạng thái phù hợp chỉ có thể là [1, 2, 3, 4, 5, 6, 0, 7, 8] , [1, 2, 3, 4, 5, 6, 7, 8, 0] , [1, 2, 3, 4, 0, 6, 7, 5, 8].

5. Kết quả

Ta có giao diện Website như sau: Bên trái là nơi để thêm ảnh, bên phải là các chức năng của Website



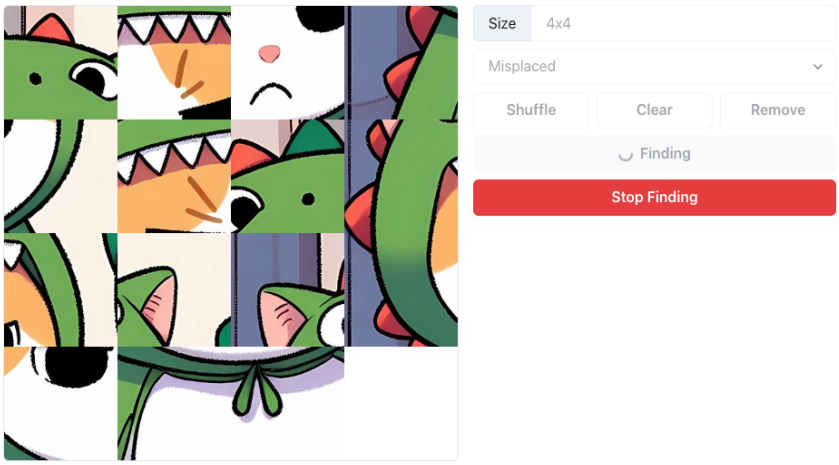
Hình 3.1: Giao diện chính của Website bao gồm khung chơi, và bảng điều khiển



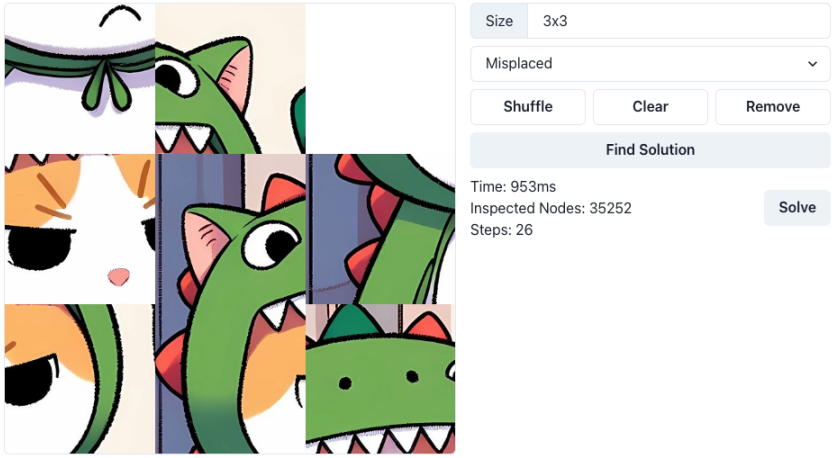
Hình 3.2: Sau khi chọn ảnh thì các chức năng bên bảng điều khiển sẽ được kích hoạt

Người dùng có thể tương tác chọn tỉ lệ mình muốn, heuristic muốn sử dụng. Các nút chức năng bao gồm:

- Shuffle: Tự động trộn hình
- Clear: Reset hình về trạng thái ban đầu hay trạng thái đích
- Remove: Xoá hình đang chọn
- Find Solution: Bắt đầu tìm lời giải

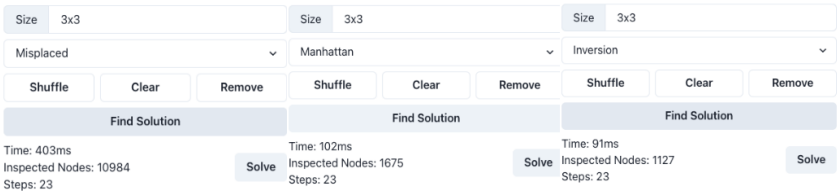


Hình 3.3: Giao diện khi đã shuffle ảnh và bấm nút Find Solution



Hình 3.4: Thông tin hiển thị sau khi tìm xong, nút Solve để bắt đầu giải

6. So sánh



Hình 3.5: So sánh 3 hàm Heuristics

Có thể thấy rõ với một hình 3x3, tốc độ giải và nút đã kiểm tra của 3 heuristic là khác nhau một cách rõ rệt.

- Misplaced:
  - Tốc độ: 403ms

    * Nút đã kiểm: 10984

- Manhattan:
    * Tốc độ: 102ms
    * Nút đã kiểm : 1675
- Inversion:
    * Tốc độ: 91ms
    * Nút đã kiểm: 1127

Misplaced như dự đoán trước tốc độ của heuristic này hoàn toàn không tốt và có thể nói với một bài 4x4, heuristic gần như là không giải được hoặc không thoả được về mặt thời gian và bộ nhớ. Với 3 heuristic còn lại về tốc độ không chênh lệnh quá nhiều, nhưng đối với số nút đã kiểm thì Inversion ít hơn đáng kể so với 2 hàm còn lại. Dưới đây là 4 bảng thống kê thời gian chạy của 4 heuristic này để làm rõ hơn.

th1: 4,7,5,6,0,2,14,13,3,1,9,10,15,11,12,8 th2: 0,1,6,3,10,2,12,11,15,8,5,13,14,9,4,7 th3: 7,4,8,11,1,10,14,2,12,0,6,13,3,9,5,15 th4: 8,1,14,15,2,0,3,12,4,6,13,11,5,9,10,7 th5: 13,11,2,6,3,4,12,8,7,1,0,5,15,14,9,10

Manhattan Distance

Trường hợp	Thời gian	Số nút đã kiểm
1	3837.400ms	407869
2	5862.300ms	599164
3	14084.500ms	1363631
4	619.300ms	71490
5	6271.700ms	762875

Bảng 3.1: Bảng thông số Manhattan Distance.

Inversion Distance

Trường hợp	Thời gian	Số nút đã kiểm
1	2044.200ms	146103
2	2024.400ms	176212
3	4431.100ms	293209
4	1145.500ms	84162
5	3143.600ms	241940

Bảng 3.2: Bảng thông số Inversion Distance.

Manhattan + Linear Conflict

Trường hợp	Thời gian	Số nút đã kiểm
1	3534.900ms	363407
2	3582.200ms	375410
3	16381.400ms	1516052
4	380.900ms	41089
5	5419.800ms	595122

Bảng 3.3: Bảng thông số Manhattan + Linear Conflict.

Walking distance

Trường hợp	Thời gian	Số nút đã kiểm
1	633.300ms	25146
2	4814.500ms	303073
3	798.600ms	32479
4	2874.900ms	118111
5	10012.800ms	558320

Bảng 3.4: Bảng thông số Walking Distance.

Do thời gian chạy phụ thuộc vào tương đối nhiều yếu tố nên để so sánh 3 heuristic trên chúng em chọn so sánh dựa vào số nút đã kiểm. Từ đó tính ra hàm Heuristic tối ưu nhất là Manhattan + Linear còlict

Tổng số nút:

Manhattan: 190261+218,802+327131+420186+3992454 = 7110834

Inversion: 208446+1173516+549498+579118+2593386=7103964

Manhattan + Linear Conflict: 30338+171245+35650+71391+318577=627201

Walking Distance: 25146+303073+32479+118111+558320=1037129

Dựa trên tổng số nút đã kiểm tra:

- Manhattan + Linear Conflict là heuristic tối ưu nhất với tổng số nút đã kiểm tra là 627201.
- Walking Distance là heuristic tối ưu tiếp theo với tổng số nút đã kiểm tra là 1037129.
- Inversion Distance là heuristic tiếp theo với tổng số nút đã kiểm tra là 7103964.
- Manhattan Distance là heuristic ít tối ưu nhất với tổng số nút đã kiểm tra là 7110834.

Ta có:

Manhattan + Linear Conflict so với Manhattan

Phần trăm chênh lệch = $\left(\frac{\text{Chênh lệch của tổng 2 nút}}{\text{Tổng số nút của Manhattan}}\right) \times 100\%$

= $\left(\frac{6483633}{7110834}\right) \times 100\%$

= 9.1%

Vậy Manhattan + Linear Conflict nhanh hơn Manhattan khoảng 9.1%

Manhattan + Linear Conflict so với Walking Distance

Phần trăm chênh lệch = $\left(\frac{\text{Chênh lệch của tổng 2 nút}}{\text{Tổng số nút của Walking Distance}}\right) \times 100\%$

= $\left(\frac{409928}{1037129}\right) \times 100\%$

= 39.53%

Vậy Manhattan + Linear Conflict nhanh hơn Walking Distance khoảng 39.53%

Manhattan + Linear Conflict so với Inversion Distance

Phần trăm chênh lệch = $\left(\frac{\text{Chênh lệch của tổng 2 nút}}{\text{Tổng số nút của Inversion Distance}}\right) \times 100\%$

= $\left(\frac{6476763}{7103964}\right) \times 100\%$

= 9.12%

Vậy Manhattan + Linear Conflict nhanh hơn Inversion Distance khoảng 9.12%

Inversion Distance so với Manhattan Distance

Phần trăm chênh lệch = $\left(\frac{\text{Chênh lệch của tổng 2 nút}}{\text{Tổng số nút của Manhattan Distance}}\right) \times 100\%$

= $\left(\frac{6870}{7110834}\right) \times 100\%$

= 0.09%

Vậy Inversion Distance nhanh hơn Manhattan Distance khoảng 0.09%

Walking Distance so với Inversion Distance

Phần trăm chênh lệch = $\left(\frac{\text{Chênh lệch của tổng 2 nút}}{\text{Tổng số nút của Inversion Distance}}\right) \times 100\%$

= $\left(\frac{6066835}{7103964}\right) \times 100\%$

= 85.4%

Vậy Walking Distance nhanh hơn Inversion Distance khoảng 85.4%

Walking Distance so với Manhattan Distance

Phần trăm chênh lệch = $\left(\frac{\text{Chênh lệch của tổng 2 nút}}{\text{Tổng số nút của Manhattan Distance}}\right) \times 100\%$

= $\left(\frac{6073705}{7110834}\right) \times 100\%$

= 85.41%

Vậy Walking Distance nhanh hơn Manhattan Distance khoảng 85.41%

Vì Mahattan chỉ dựa vào khoảng cách đến vị trí chính xác. Còn Inversion chỉ dựa vào số lần lên xuống, sang trái phỉa ít nhất để giải bài toán nên tùy vào những trường khác nhau mức tối ưu của Manhattan và Inversion sẽ khác nhau.

CHƯƠNG 4. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1 Kết luận

Qua quá trình học môn trí tuệ nhân tạo và việc thực hiện đề tài này, chúng em đã hiểu thêm hơn về việc ứng dụng trí tuệ nhân tạo trong việc giải quyết các vấn đề trong thực tế. Trò chơi ghép tranh là một trò chơi ứng dụng rất tốt thuật toán A*. Tuy nhiên do hạn chế về mặt kiến thức nên nhóm em chưa thể tối ưu được thuật toán, viết nhiều hàm heuristic mới và hay như mong đợi, cũng như trong quá trình thực hiện không thể tránh khỏi sai sót. Chúng em rất mong nhận được sự góp ý của cô để có thể hoàn thiện thuật toán A* và những hàm heuristic còn lại một cách tốt hơn.

Trong quá trình phát triển phần ứng dụng web, để tiện lợi cho việc kiểm tra, chúng em đã viết một thuật toán tự động trộn các mảnh ghép của trò chơi ghép tranh. Tuy nhiên, không hiểu vì lý do gì, nhiều lần sau khi trộn xong, chúng em lại không thể giải được bài toán 3x3, mặc dù đã sử dụng thuật toán Manhattan Distance. Ban đầu, chúng em nghĩ rằng bài toán quá khó hoặc do mã nguồn của chúng em có lỗi. Sau khi tìm hiểu về heuristic Inversion Distance, chúng em mới nhận ra rằng hàm trộn của mình đã viết sai. Đối với bài toán 3x3, số inversion ở trạng thái đích luôn là 0. Việc di chuyển các mảnh ghép chỉ thay đổi số inversion theo một lượng chẵn (+2, -2 hoặc không thay đổi). Do đó, để bài toán 3x3 có thể giải được, số inversion phải luôn là số chẵn. Sau khi sửa lại hàm trộn, chúng em nhận thấy rằng hầu hết các bài toán 3x3 đều có thể được giải nhanh chóng bằng thuật toán Manhattan Distance. Điều này giúp chúng em hiểu sâu hơn về bài toán ghép tranh, chẳng hạn như đối với bài toán 3x3 giải được thì chỉ tồn tại 9!/2 trạng thái có thể giải.

Trong quá trình triển khai heuristic Inversion Distance, chúng em gặp khá nhiều khó khăn. Đầu tiên là vấn đề tài liệu tham khảo. Tài liệu heuristic trên được giới thiệu nhưng phần giải thích lại không rõ ràng, cộng thêm việc vốn ngoại ngữ của chúng em còn hạn chế khiến cho việc hiểu rõ vấn đề trở nên khó khăn. Sau một thời gian nghiên cứu và tham khảo mã nguồn mẫu của một số người trên mạng nhưng không thu được kết quả khả quan, chúng em đã quyết định ngồi lại với nhau để thảo luận kỹ lưỡng về vấn đề này và cuối cùng đã tìm ra hướng giải quyết phù hợp.

4.2 Hướng phát triển

Cải thiện thuật toán để giảm thời gian chạy, thêm một số hàm heuristic như Walking Distance và Pattern Databases.

Nâng cấp Website thành ứng dụng chứa thông tin để mọi người có thể học cách thuật toán A* và cách Heuristic hoạt động. Cũng như thêm chức năng để người dùng có thể import các hàm Heuristic của bản thân vào để giải trò chơi ghép tranh.

TÀI LIỆU THAM KHẢO

1) https://drive.google.com/file/d/1ku3j6otJpxKqW2HtYt2CpAH3ft7ruf1t/view?usp=drive_link (Slide nhập môn trí tuệ nhận tạo chương I)

2) https://michael.kim/blog/puzzle

3) https://cse.sc.edu/~mgv/csce580sp15/gradPres/HansonMayerYung1992.pdf