|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q2**  **TRƯỜNG KHOA HỌC MÁY TÍNH - ĐẠI HỌC DUY TÂN**  **KHOA CÔNG NGHỆ THÔNG TIN**  -----🙞🙜🕮🙞🙜-----    **BÁO CÁO ĐỒ ÁN MÔN HỌC**  **CƠ SỞ DỮ LIỆU NÂNG CAO**  **Tên đề tài**  **"Ứng Dụng Thuật Toán A\* Trong Tìm Kiếm Đường Đi An Toàn Cho Robot Di Chuyển Trong Môi Trường Động"**   |  |  | | --- | --- | | **GVHD:**  **Lớp:** | **TS. Lê Quốc Hải**  **CS417E** | | **Sinh viên thực hiện:** | 1. **Nguyễn Văn Tài** 2. **Nguyễn Hữu Hoàng** |   *Đà Nẵng, 12/2023* |

**Mục lục:**

Mục lục

**MỞ ĐẦU**

Trong thời đại ngày nay, trí tuệ nhân tạo đã trở thành một lĩnh vực ngày càng quan trọng, với ứng dụng rộng rãi từ tự động hóa, robot học và cả trong lĩnh vực giải quyết vấn đề.

Trong báo cáo này, chúng ta sẽ đàm phán về một chủ đề cụ thể trong trí tuệ nhân tạo - đó là ứng dụng của thuật toán A\* trong việc tìm kiếm đường đi trên cây trạng thái. Chúng tôi sẽ bắt đầu bằng việc trình bày bài toán cụ thể mà chúng tôi đã chọn để áp dụng thuật toán A\* và giải quyết. Sau đó, chúng tôi sẽ cung cấp một cái nhìn sâu sắc vào lý thuyết đằng sau thuật toán A\* và cách nó có thể được áp dụng để giải quyết bài toán cụ thể này.

Mục tiêu chính của báo cáo là không chỉ trình bày lý thuyết và cách áp dụng thuật toán, mà còn thực hiện một ứng dụng thực tế của nó. Chúng tôi sẽ trình bày sản phẩm cuối cùng của chúng tôi, bao gồm mã nguồn và giao diện đồ họa, cùng với mô phỏng các kết quả và ứng dụng của thuật toán A\* trong bối cảnh cụ thể mà chúng tôi đã nghiên cứu.

Chúng ta cùng khám phá và hiểu rõ hơn về cách thuật toán A\* có thể đóng góp vào việc giải quyết các thách thức đầy thú vị trong lĩnh vực trí tuệ nhân tạo.

**Nội dung:**

**I. Cơ sở lý thuyết:**

Bài toán cần giải quyết:

Chúng ta đối mặt với bài toán tìm đường đi an toàn cho robot trong một môi trường động. Môi trường này có chứa các vật cản xuất hiện và biến động, và chúng ta cần một giải pháp thông minh để đảm bảo robot di chuyển hiệu quả.

Thuật toán:

Để giải quyết bài toán này, nhóm đã nghiên cứu và chọn sử dụng thuật toán A\*. Thuật toán này được chọn vì khả năng tìm kiếm đường đi hiệu quả và có thể xử lý môi trường động.

**II. Kết quả nghiên cứu**

Nhóm đã thành công ứng dụng thuật toán A\* vào bài toán tìm đường đi cho robot trong môi trường động.

Mô tả cụ thể về cách thuật toán A\* được tích hợp để đảm bảo robot di chuyển an toàn, tránh vật cản và biến động.

Trình bày sản phẩm: phần code, giao diện, sử dụng.

Phần code:

import pygame

from queue import PriorityQueue

# Khởi tạo môi trường

WIDTH = 900

HEIGHT = 500

WIN = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Áp dụng Thuật toán A\*")

# Định nghĩa màu sắc

WHITE = (0, 0, 0)

BLACK = (255, 255, 255)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

BLUE = (0, 0, 255)

YELLOW = (255, 255, 0)

class Node:

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

def get\_pos(self):

return self.row, self.col

def is\_closed(self):

return self.color == RED

def is\_open(self):

return self.color == GREEN

def is\_barrier(self):

return self.color == BLACK

def is\_start(self):

return self.color == BLUE

def is\_end(self):

return self.color == YELLOW

def reset(self):

self.color = WHITE

def make\_start(self):

self.color = BLUE

def make\_closed(self):

self.color = RED

def make\_open(self):

self.color = GREEN

def make\_barrier(self):

self.color = BLACK

def make\_end(self):

self.color = YELLOW

def make\_path(self):

self.color = BLUE

def draw(self, win):

pygame.draw.rect(win, self.color, (self.x, self.y, self.width, self.width))

def update\_neighbors(self, grid):

self.neighbors = []

if self.row < self.total\_rows - 1 and not grid[self.row + 1][self.col].is\_barrier():

self.neighbors.append(grid[self.row + 1][self.col])

if self.row > 0 and not grid[self.row - 1][self.col].is\_barrier():

self.neighbors.append(grid[self.row - 1][self.col])

if self.col < self.total\_rows - 1 and not grid[self.row][self.col + 1].is\_barrier():

self.neighbors.append(grid[self.row][self.col + 1])

if self.col > 0 and not grid[self.row][self.col - 1].is\_barrier():

self.neighbors.append(grid[self.row][self.col - 1])

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

return False

# Hàm heuristic (ước lượng chi phí)

def h(p1, p2):

x1, y1 = p1

x2, y2 = p2

return abs(x1 - x2) + abs(y1 - y2)

# Thuật toán A\*

def astar(draw, grid, start, end):

count = 0

open\_set = PriorityQueue()

open\_set.put((0, count, start))

came\_from = {}

g\_score = {node: float("inf") for row in grid for node in row}

g\_score[start] = 0

f\_score = {node: float("inf") for row in grid for node in row}

print("Start in astar:", start)

print("End in astar:", end)

f\_score[start] = h(start.get\_pos(), end.get\_pos())

open\_set\_hash = {start}

while not open\_set.empty():

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

current = open\_set.get()[2]

open\_set\_hash.remove(current)

if current == end:

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]:

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

# Vẽ đường đi

def reconstruct\_path(came\_from, current, draw):

while current in came\_from:

current = came\_from[current]

current.make\_path()

draw()

# Vẽ lưới

def make\_grid(rows, width):

grid = []

gap = width // rows

for i in range(rows):

grid.append([])

for j in range(rows):

node = Node(i, j, gap, rows)

grid[i].append(node)

return grid

# Vẽ lưới

def draw\_grid(win, rows, width):

gap = width // rows

for i in range(rows):

pygame.draw.line(win, WHITE, (0, i \* gap), (width, i \* gap))

for j in range(rows):

pygame.draw.line(win, WHITE, (j \* gap, 0), (j \* gap, width))

# Vẽ tất cả

def draw(win, grid, rows, width):

win.fill(WHITE)

for row in grid:

for node in row:

node.draw(win)

draw\_grid(win, rows, width)

pygame.display.update()

# Chia lưới thành ô

def get\_clicked\_pos(pos, rows, width):

gap = width // rows

y, x = pos

row = y // gap

col = x // gap

return row, col

# Hàm chính

def main(win, width):

ROWS = 20 # Điều chỉnh số lượng ô trong mỗi hàng và cột

grid = make\_grid(ROWS, width)

start = None

end = None

run = True

started = False

while run:

draw(win, grid, ROWS, width)

for event in pygame.event.get():

if event.type == pygame.QUIT:

run = False

if pygame.mouse.get\_pressed()[0]:

pos = pygame.mouse.get\_pos()

row, col = get\_clicked\_pos(pos, ROWS, width)

node = grid[row][col]

if not start and node != end:

start = node

start.make\_start()

elif not end and node != start:

end = node

end.make\_end()

elif node != end and node != start:

node.make\_barrier()

elif pygame.mouse.get\_pressed()[2]:

pos = pygame.mouse.get\_pos()

row, col = get\_clicked\_pos(pos, ROWS, width)

node = grid[row][col]

node.reset()

if node == start:

start = None

elif node == end:

end = None

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE and not started:

for row in grid:

for node in row:

node.update\_neighbors(grid)

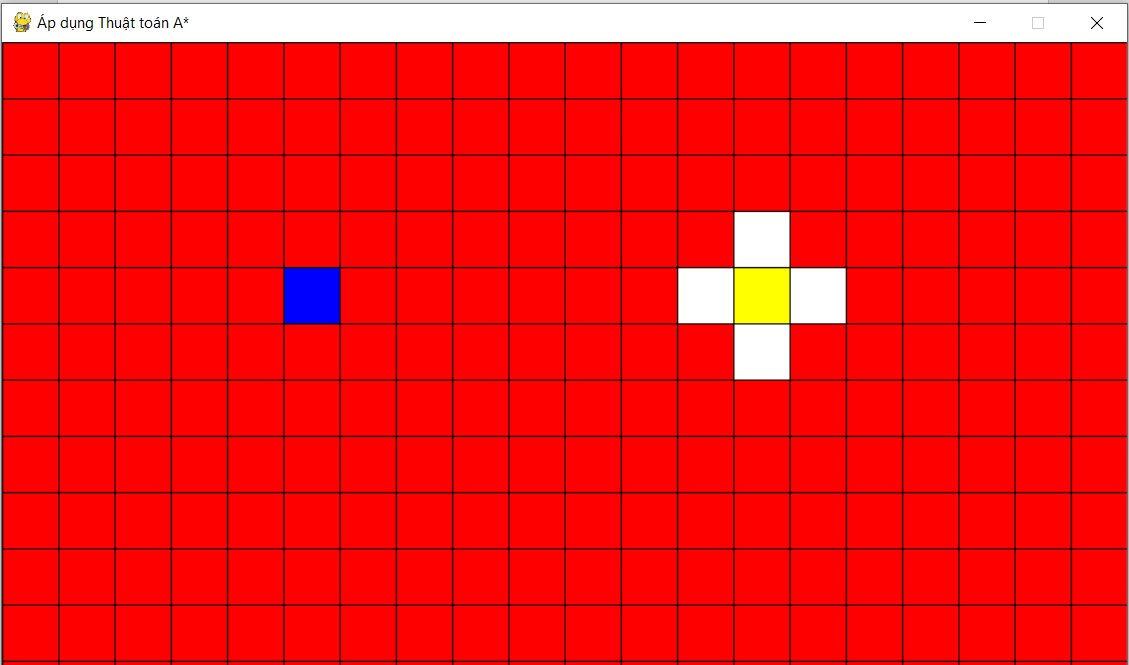
astar(lambda: draw(win, grid, ROWS, width), grid, start, end)

pygame.quit()

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

main(WIN, WIDTH)

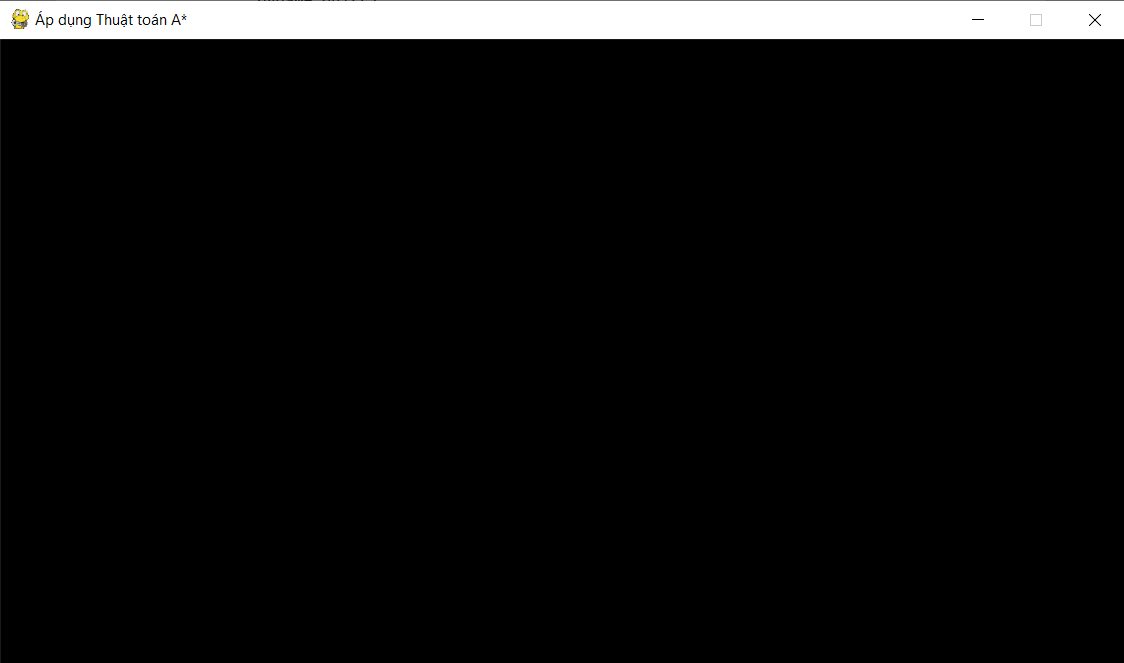
Phần giao diện:



Cách sử dụng:

**1. Khởi Tạo Môi Trường:**

- Mở ứng dụng, cửa sổ hiển thị giao diện của "**Áp dụng Thuật toán A\*".**



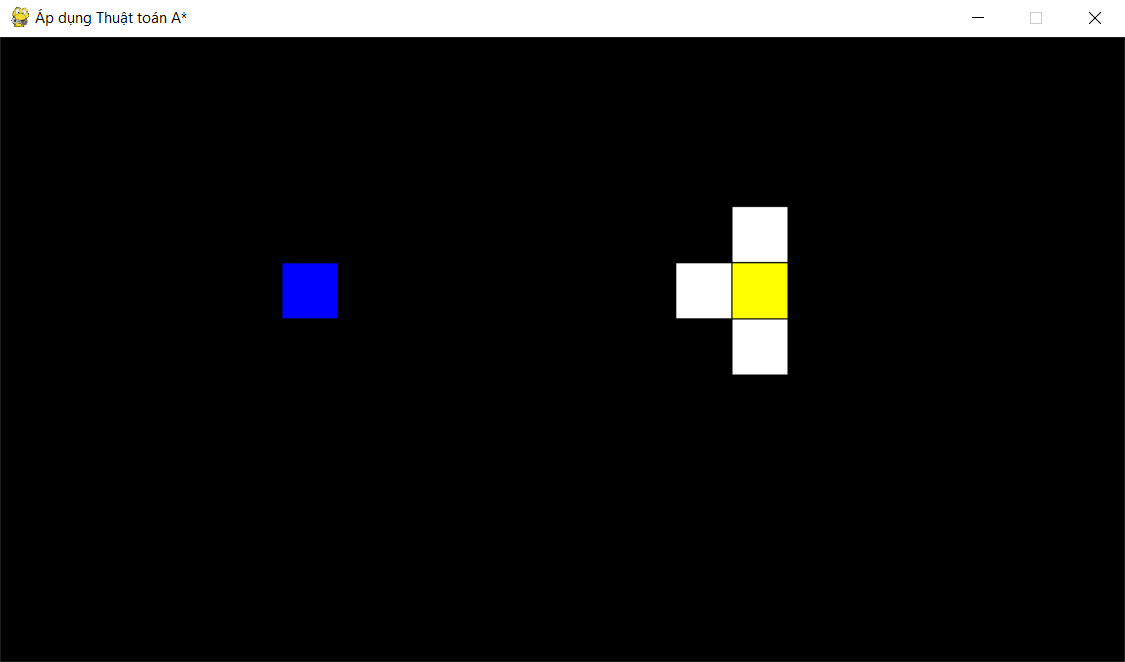
**2. Tạo Bản Đồ:**

- Bạn có thể tạo bản đồ bằng cách sử dụng chuột:

- Nhấp trái để đặt điểm bắt đầu (màu xanh da trời).

- Nhấp phải để đặt điểm kết thúc (màu vàng).

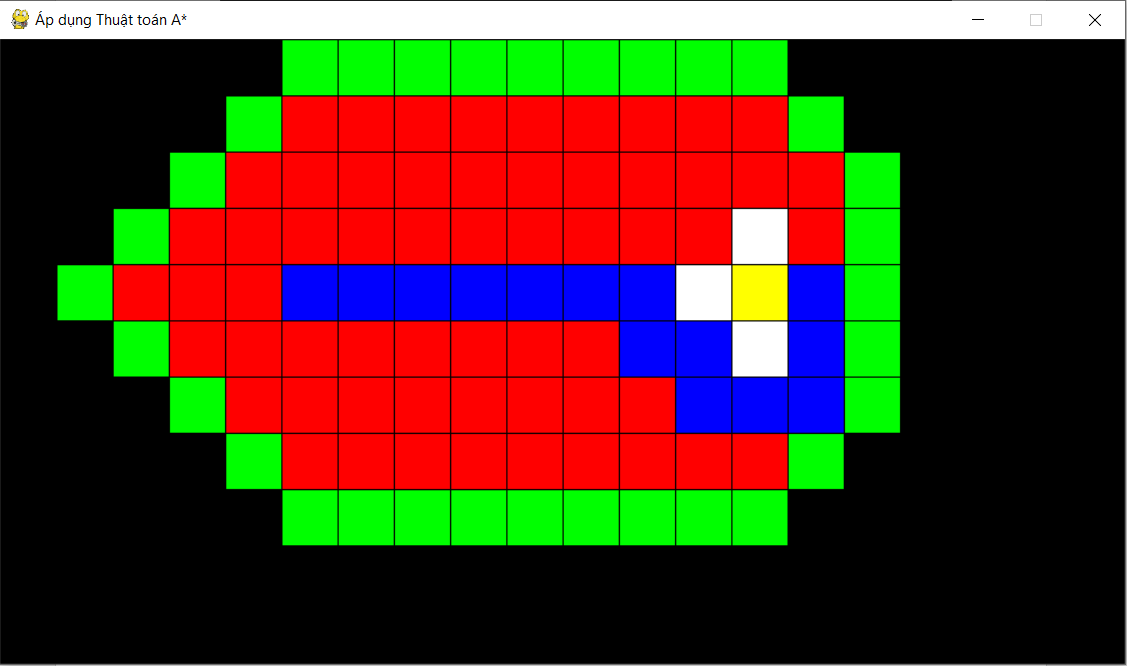
- Kéo chuột để tạo các rào cản (màu trắng).



**3. Khởi Chạy Thuật Toán:**

- Nhấn phím `SPACE` để bắt đầu chạy thuật toán A\* từ điểm bắt đầu đến điểm kết thúc.

- Thuật toán sẽ tìm kiếm đường đi ngắn nhất và hiển thị nó trên màn hình (màu xanh da trời).

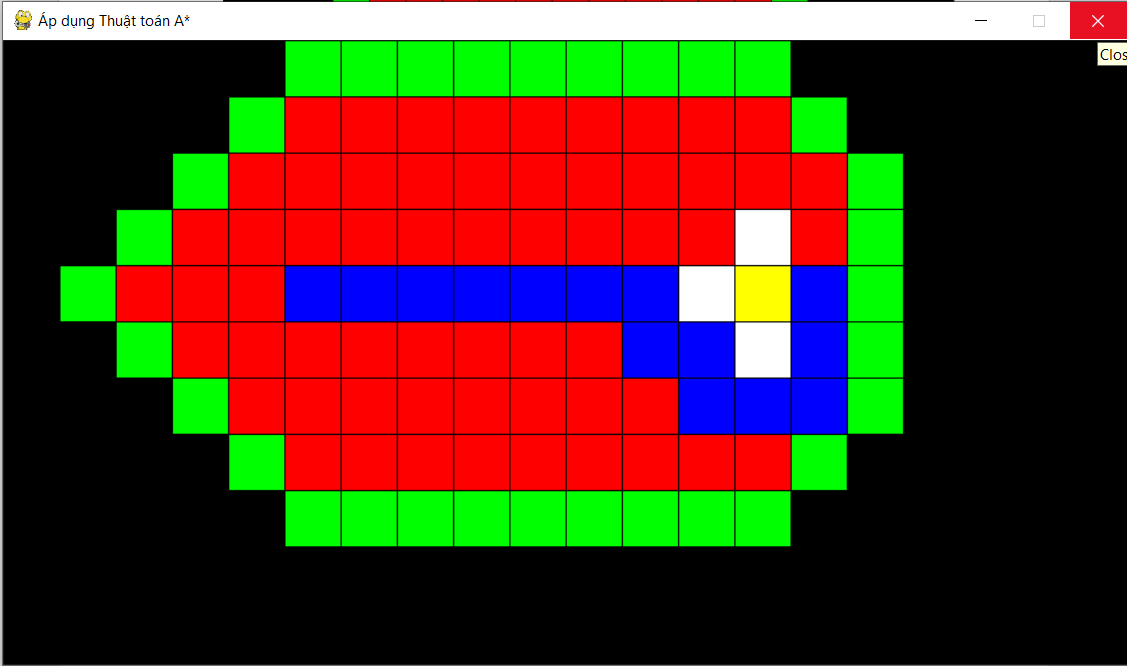


**4. Thay Đổi Bản Đồ:**

- Bạn có thể tiếp tục thay đổi bản đồ trong khi thuật toán đang chạy.

**5. Kết Thúc Chương Trình:**

- Nhấn nút đóng cửa sổ để thoát khỏi ứng dụng.



**Ghi Chú:**

- Các ô màu đen đại diện cho các rào cản.

- Các ô màu xanh lá cây là đường đi tìm được bởi thuật toán A\*.

- Điểm bắt đầu là ô màu xanh lá cây.

- Điểm kết thúc là ô màu vàng.

- Bạn có thể thêm, di chuyển hoặc xóa rào cản trong khi thuật toán đang chạy.

**III. Kết luận: Kết luận về những nội dung nhóm đã trình bày**

Nhóm đã mô tả chi tiết về quá trình nghiên cứu và triển khai thuật toán A\* để giải quyết bài toán tìm đường đi trong môi trường động.

Tổng kết ưu điểm của thuật toán trong việc đảm bảo hiệu suất và an toàn cho robot.

Đánh giá chi tiết về sản phẩm, bao gồm cả mã nguồn và giao diện, và làm rõ cách sử dụng nó trong môi trường thực tế.

Nêu rõ những hạn chế của nghiên cứu và đề xuất hướng phát triển cho tương lai, có thể bao gồm cải thiện hiệu suất thuật toán hoặc tích hợp các yếu tố môi trường phức tạp.