ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

ROBOT TÌM ĐƯỜNG

CƠ SỞ TRÍ TUỆ NHÂN TẠO

• **Nhóm thực hiện:**

1712702 – Nguyễn Hà Quang
1712227 – Lâm Thành Lộc
1712441 – Trần Đình Tôn Hiếu

❖ **Khóa 2017 - 2021**

MỤC LỤC

PHẦN 1: THÔNG TIN NHÓM

1. Phân công công việc

Thành viên	MSSV	Nhiệm vụ
Trần Đình Tôn Hiếu	1712441	Đọc dữ liệu từ file input, cài đặt giao diện
Nguyễn Hà Quang	1712702	Cài đặt thuật toán
Lâm Thành Lộc	1712227	Tối ưu thuật toán, chạy test, viết báo cáo

2. Mức độ hoàn thành mỗi mức yêu cầu

Mức	Tỉ trọng	Công việc	Hoàn thành
1	40%	Cài đặt thành công thuật toán BFS và chạy thử	100%
2	30%	Cài đặt thêm 3 thuật toán tìm đường: DFS, Greedy, A-star. Chạy thử và so sánh các thuật toán	100%
3	20%	Đặt thêm các điểm đón và cài đặt thuật toán để đến tất cả các điểm này trước khi đến đích.	100%
4	10%	(Không thực hiện)	0%

PHẦN 2: TỔNG QUAN CÁCH CÀI ĐẶT

1. Môi trường lập trình

Ngôn ngữ lập trình sử dụng: Python 3

Hệ điều hành: Unix (project không chạy được trên Windows)

2. Các thư viện sử dụng

Terminal: đồ họa cơ bản trên Terminal

Bresenham: nối 2 điểm trong hệ trục tọa độ

Time: sleep chương trình

Permutations: tạo hoán vị của một dãy số

```
from blessings import Terminal
from bresenham import bresenham
import numpy as np
import time
from itertools import permutations
```

3. Hướng dẫn chạy chương trình

- Vào thư mục Source, mở Terminal
- Nhập lệnh: python main.py (hoặc: python3 main.py)
- Cài đặt thêm các thư viện còn thiếu (pip install <tên thư viện>)
- Nhập Selection từ 1 đến 5. Trong đó:
  + 1-4 tương ứng với 4 thuật toán tìm đường đã được cài đặt (1. DFS | 2. Greedy | 3. BFS | 4. A-star)
  + 5 là lựa chọn tương ứng với bài toán đi qua các điểm đón trước khi về đích (dùng A-star và brute-force)

4. Quy ước riêng

- Mỗi nước đi chỉ đi được tối đa 4 hướng (không đi chéo)
- Chi phí mỗi hướng đi là như nhau -> đồ thị không trọng số

PHẦN 3: CHI TIẾT THUẬT TOÁN

1. Mức 1

Thuật toán được sử dụng: Breadth-first search (BFS)

Mô tả thuật toán: việc tìm kiếm chỉ bao gồm 2 thao tác

a. Cho trước một đỉnh của đồ thị

b. Thêm các đỉnh kề với đỉnh vừa cho vào danh sách có thể hướng tới tiếp theo.

Tính đầy đủ: thuật toán bảo đảm tìm thấy lời giải nếu có

Tính tối ưu: vì đồ thị không có trọng số nên thuật toán bảo đảm tìm thấy lời giải tối ưu, tức là luôn tìm được đường đi có chi phí nhỏ nhất (nếu có)

Độ phức tạp về thời gian: không quá lớn

Độ phức tạp về không gian: đối với thuật toán BFS, các nút được phát sinh đều được lưu trong bộ nhớ, do đó độ phức tạp về mặt không gian là rất lớn -> không đủ bộ nhớ để tìm đường đi ngắn nhất đối với các bản đồ có kích thước quá lớn.

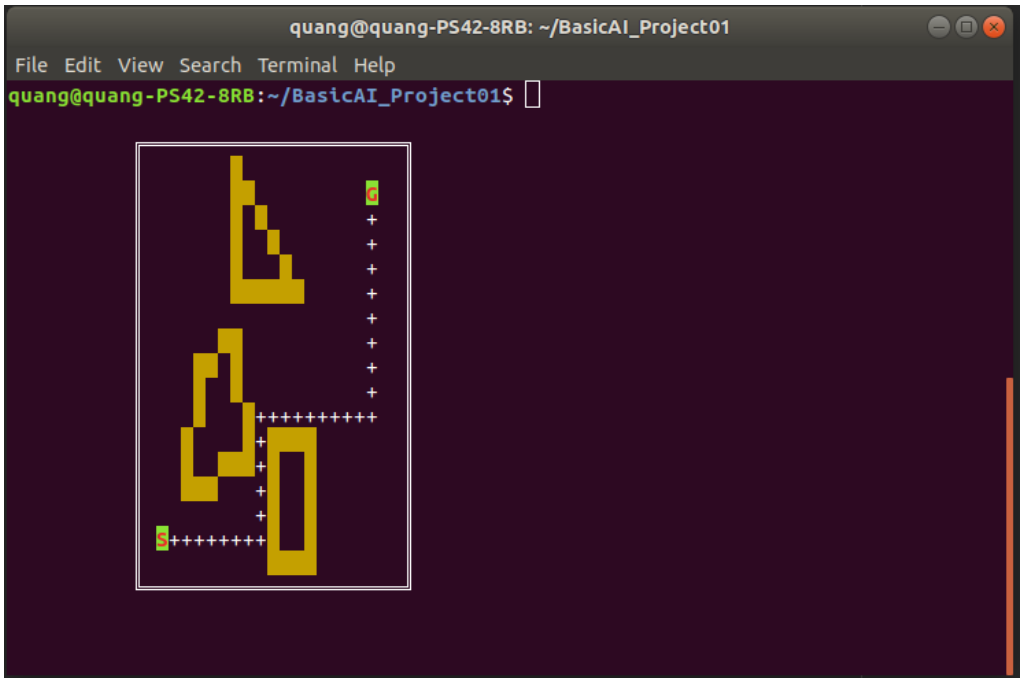
- **Cách cài đặt:**

```
def bfs():
    flag = np.zeros((height + 1, width + 1))
    flag[start.y][start.x] = 1
    tracking = [[Point(-1, -1) for x in range(width + 1)] for y in range(height + 1)]

    queue = [start]
    found = False
    while len(queue) != 0 and not found:
        open_node = queue.pop(0)
        # open nodes and add to queue
        for i in range(4):
            xx = open_node.x + dx[i]
            yy = open_node.y + dy[i]
            # return false if out the bound
            if xx < 1 or yy < 1 or xx > width - 1 or yy > height - 1:
                continue
            # return false if checked or crash the shape
            if flag[yy][xx] == 1 or my_map[yy][xx] == 1:
                continue

            queue.append(Point(xx, yy))
            tracking[yy][xx] = Point(open_node.x, open_node.y)
            flag[yy][xx] = 1
            if xx == end.x and yy == end.y:
                found = True
                break

    if tracking[end.y][end.x] == Point(-1, -1):
        return False
```

- **Chạy thử:**

Test 1

Kết quả: tìm được đường đi ngắn nhất

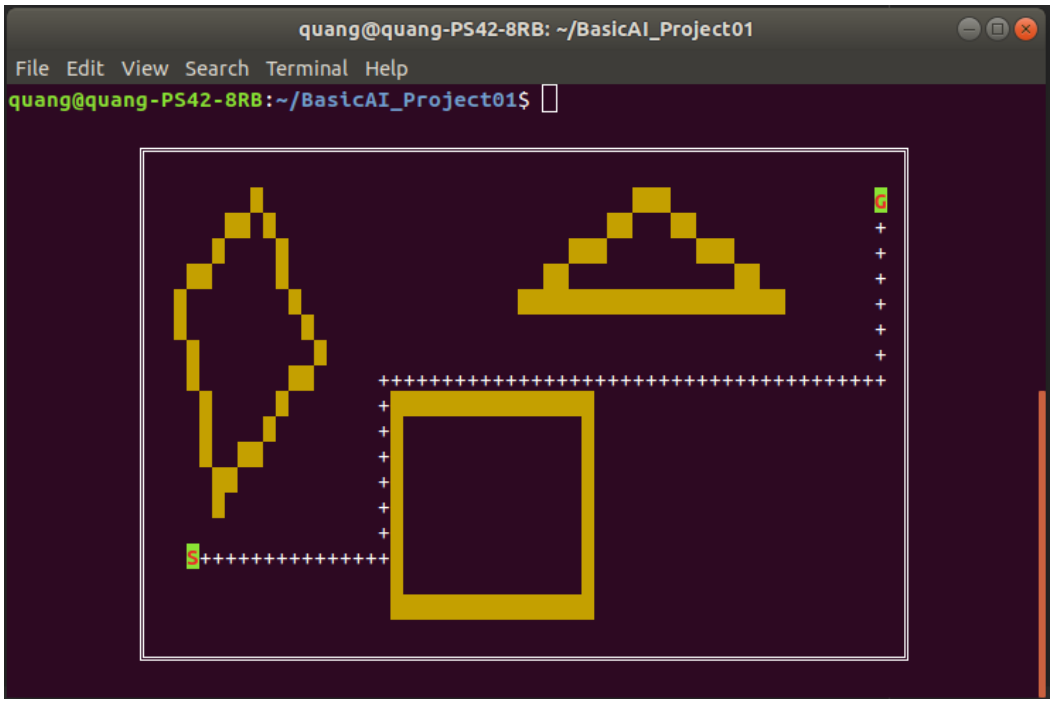


Test 2

Kết quả: không tìm được đường đi

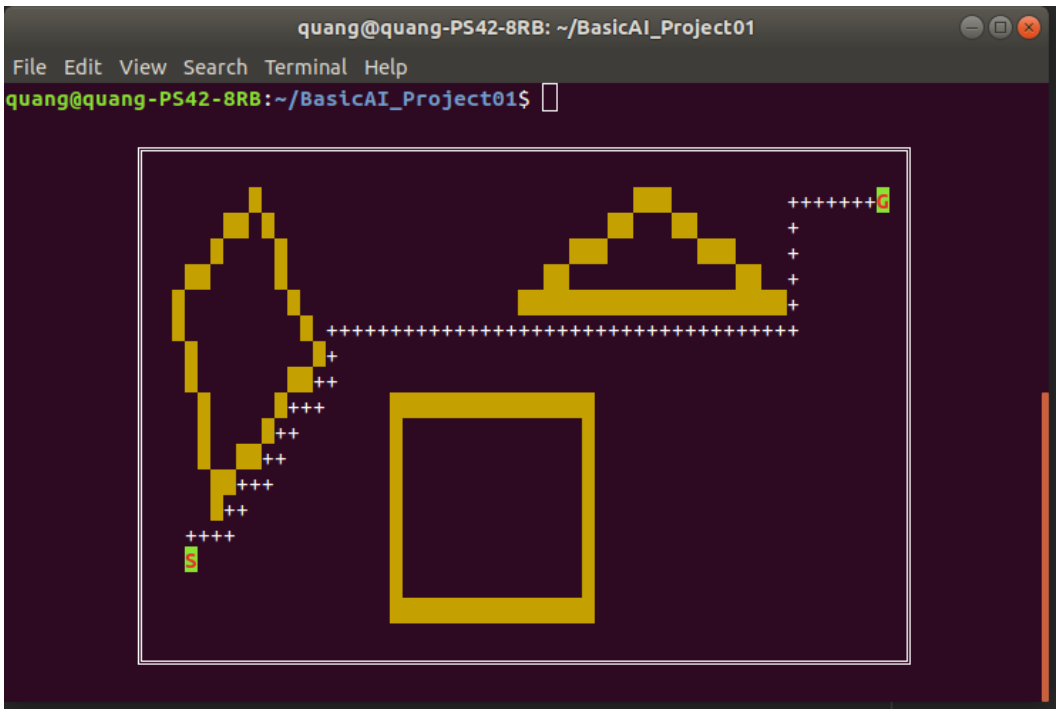
Test 3

Kết quả: tìm được đường đi ngắn nhất



Test 3: thay đổi độ ưu tiên của các hướng đi

Kết quả: vẫn tìm được đường đi ngắn nhất nhưng khác với đường đi ở trên



2. Mức 2

Cài thêm 3 thuật toán

2.1. Depth-first search (DFS)

Mô tả thuật toán: quá trình tìm kiếm được phát triển tới đỉnh con đầu tiên của nút đang tìm kiếm cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó giải thuật quay lui về đỉnh vừa mới tìm kiếm ở bước trước.

Tính đầy đủ: thuật toán bảo đảm tìm thấy lời giải nếu có

Tính tối ưu: trong đa số các trường hợp, DFS không đưa ra được lời giải tối ưu (chỉ tìm được đường đi nhưng không phải đường đi ngắn nhất)

Độ phức tạp về thời gian: nhanh hơn BFS

Độ phức tạp về không gian: thuật toán DFS sử dụng bộ nhớ khá tối ưu, nó chỉ lưu trữ những nút trên nhánh đường đi đang được xét đến, cùng với những nút lân cận chưa được mở rộng trên đường đi.

- **Cách cài đặt:**

```
def dfs_recursive(flag, x, y):
    flag[y][x] = 1
    routing.append(Point(x, y))

    if x == end.x and y == end.y:
        return True

    for i in range(4):
        xx = x + dx[i]
        yy = y + dy[i]
        # return false if out the bound
        if xx < 1 or yy < 1 or xx > width - 1 or yy > height - 1:
            continue
        # return false if checked or crash the shape
        if flag[yy][xx] == 1 or my_map[yy][xx] == 1:
            continue

        if dfs_recursive(flag, xx, yy):
            return True

    # flag[y][x] = 0
    del routing[-1]
    return False
```

- **Chạy thử:**

Test 1

Kết quả: tìm được đường đi (không phải ngắn nhất)

Test 1: thay đổi độ ưu tiên của các hướng đi

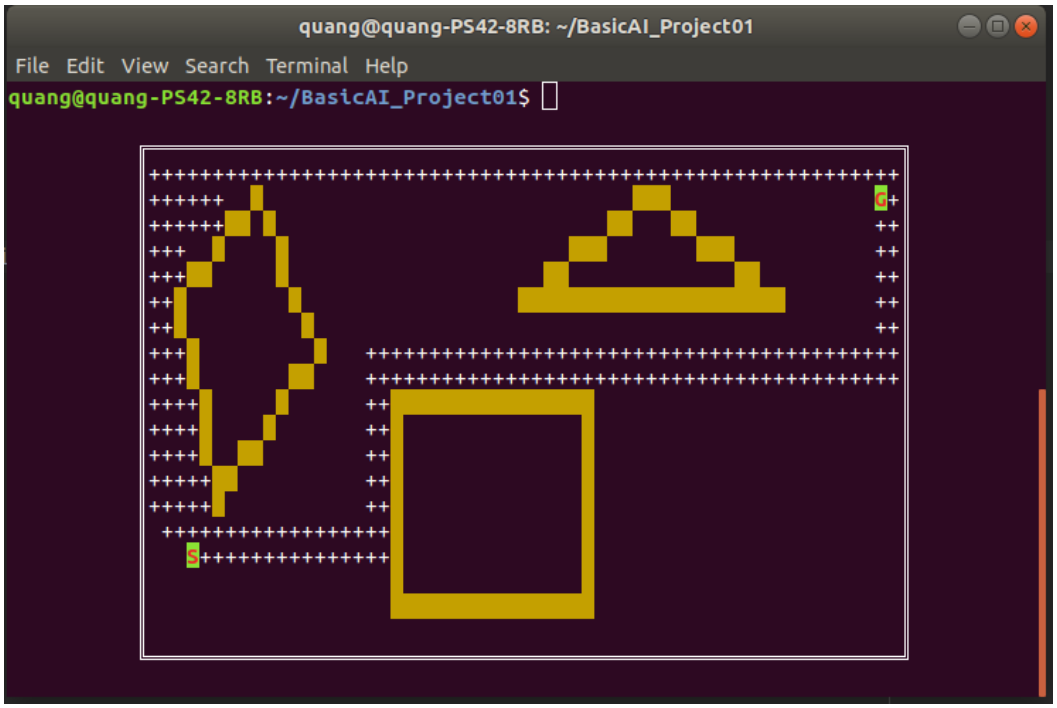
Kết quả: tìm được đường đi nhưng thậm chí kém tối ưu hơn đường đi ở trên

Test 2

Kết quả: không tìm được đường đi

Test 3

Kết quả: tìm được đường đi (không phải ngắn nhất)



2.2. Greedy Best-first search (GBFS)

Mô tả thuật toán: tìm kiếm tối ưu kiểu tham lam mở rộng theo chiều sâu nút gần đích nhất với hi vọng cách làm này sẽ dẫn đến lời giải một cách nhanh nhất. Thuật toán này đánh giá mức độ "gần đích" của một nút n (với đích là s) dựa theo hàm heuristic: h(n) = manhattan(n, s)

Trong đó manhattan(n, s) là khoảng cách manhattan từ nút n đến nút s trong ma trận bản đồ.

Tính đầy đủ: thuật toán bảo đảm tìm thấy lời giải nếu có

Tính tối ưu: trong đa số các trường hợp, GBFS đưa ra được lời giải gần tối ưu (là đường đi không phải ngắn nhất nhưng có thể chấp nhận được). Tuy nhiên, vì cách cài đặt tương tự như DFS (chỉ đi theo một con đường để lần tới đích, gặp ngõ cụt thì mới quay lại đi theo hướng khác) nên vẫn có những trường hợp GBFS trả về đường đi kém tối ưu.

Độ phức tạp về thời gian: nhanh hơn DFS, BFS. Đặc biệt, với hàm heuristic đủ tốt, độ phức tạp về thời gian có thể giảm đi đáng kể.

Độ phức tạp về không gian: tương tự DFS

- **Cách cài đặt:**

```
def greedy_recursive(flag, x, y):
    flag[y][x] = 1
    routing.append(Point(x, y))

    if x == end.x and y == end.y:
        return True

    heu = []
    for i in range(4):
        heu.append(Heuristic(i, -1))
        xx = x + dx[i]
        yy = y + dy[i]
        # return false if out the bound
        if xx < 1 or yy < 1 or xx > width - 1 or yy > height - 1:
            continue
        # return false if checked or crash the shape
        if flag[yy][xx] == 1 or my_map[yy][xx] == 1:
            continue
        heu[i].heuristic = manhattan(xx, yy, end.x, end.y)
```

```
    heu = [x for x in heu if x.heuristic != -1]
    heu.sort()

    while len(heu) != 0:
        xx = x + dx[heu[0].index]
        yy = y + dy[heu[0].index]
        if greedy_recursive(flag, xx, yy):
            return True
        heu.pop(0)

    del routing[-1]
    return False
```

- **Chạy thử:**

Test 1

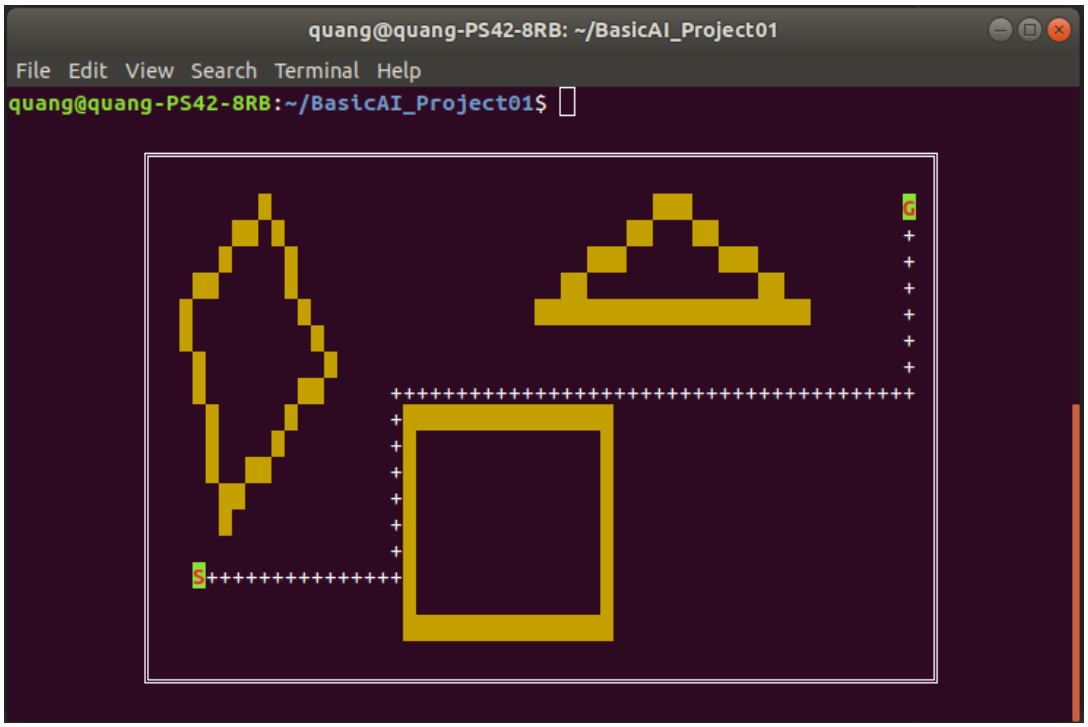
Kết quả: tìm được đường đi ngắn nhất

Test 2

Kết quả: không tìm được đường đi

Test 3

Kết quả: tìm được đường đi ngắn nhất



Test 3: thay đổi độ ưu tiên của các hướng đi

Kết quả: tìm được đường đi nhưng không phải đường đi ngắn nhất

2.3. A-star (A*)

Mô tả thuật toán: thuật toán này đánh giá một nút dựa trên chi phí đi từ nút gốc đến nút đó – g(n), cộng với chi phí heuristic từ nút đó đến đích – h(n)

Trong đó: h(n) = manhattan(n, s)

Tính đầy đủ và tối ưu: với hàm heuristic đã chọn, thuật toán cho ta kết quả đầy đủ và tối ưu

Độ phức tạp về thời gian: là thuật toán cho kết quả nhanh nhất trong 4 thuật toán được cài đặt

Độ phức tạp về không gian: vì được cài đặt tương tự BFS nên A* cần bộ nhớ lớn để lưu tất cả các node đã được mở

- **Cách cài đặt:**

```
def a_star():
    flag = np.zeros((height + 1, width + 1))
    flag[start.y][start.x] = 1
    tracking = [[Point(-1, -1) for x in range(width + 1)] for y in range(height + 1)]

    g = 0
    h = manhattan(start.x, start.y, end.x, end.y)
    queue = [Node_AS(start, g, h)]
    found = False
    while len(queue) != 0 and not found:
        open_node = queue.pop(0)
        for i in range(4):
            xx = open_node.point.x + dx[i]
            yy = open_node.point.y + dy[i]
            # return false if out the bound
            if xx < 1 or yy < 1 or xx > width - 1 or yy > height - 1:
                continue
            # return false if checked or crash the shape
            if flag[yy][xx] == 1 or my_map[yy][xx] == 1:
                continue

            # calculate current cost and heuristic
            g = open_node.g + 1
            h = manhattan(xx, yy, end.x, end.y)
            new_node = Node_AS(Point(xx, yy), g, h)
            queue.append(new_node)
            queue.sort()

            tracking[yy][xx] = Point(open_node.point.x, open_node.point.y)
            flag[yy][xx] = 1

            # return true if open end point
            if xx == end.x and yy == end.y:
                found = True
                break

    if tracking[end.y][end.x] == Point(-1, -1):
        return False

    # routing by using 'tracking'
    temp = end
    while temp != start:
        routing.append(temp)
        temp = tracking[temp.y][temp.x]

    routing.append(start)
    routing.reverse()
    return True
```

- **Chạy thử:**

Test 1

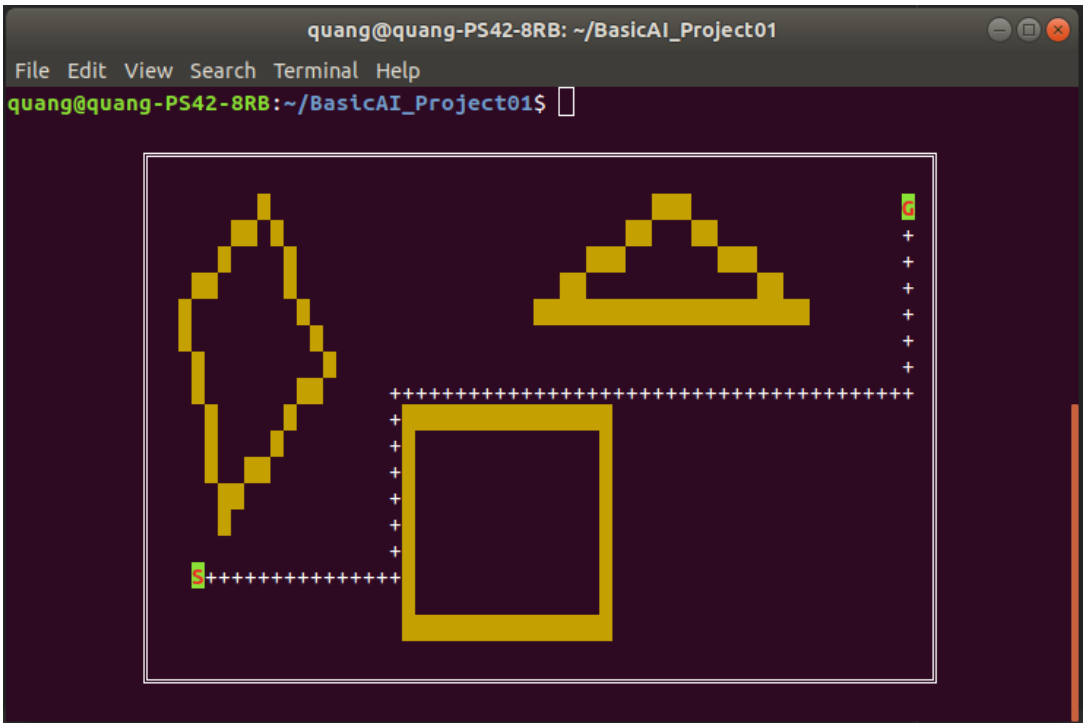
Kết quả: tìm được đường đi ngắn nhất

Test 2

Kết quả: không tìm được đường đi

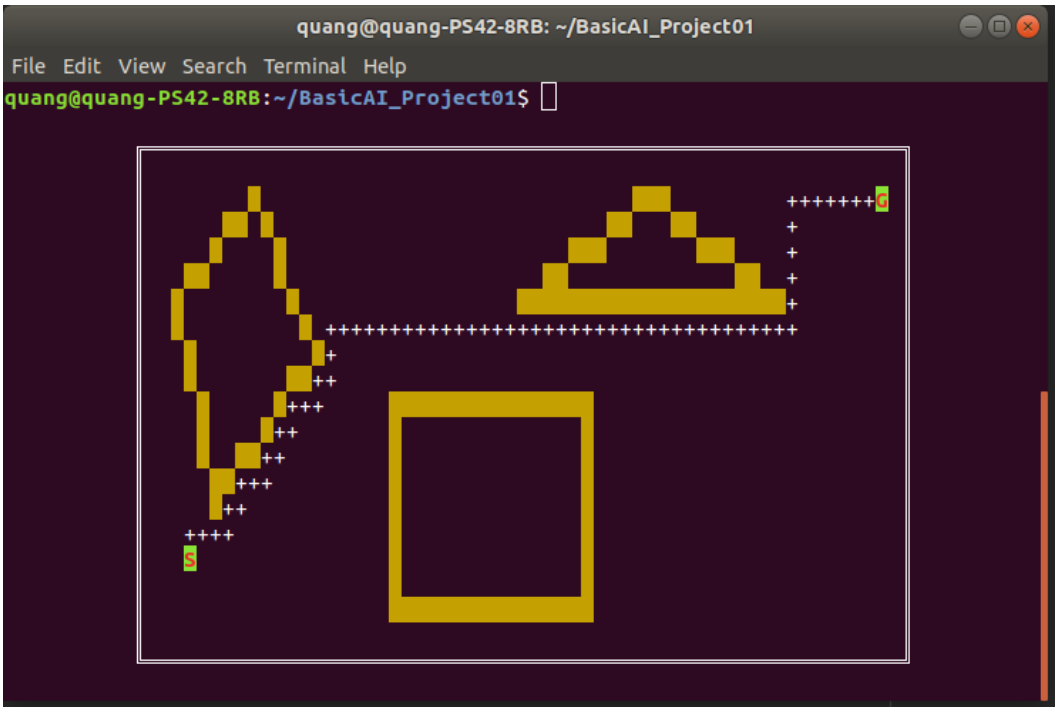
Test 3

Kết quả: tìm được đường đi ngắn nhất



Test 3: thay đổi độ ưu tiên của các hướng đi

Kết quả: vẫn tìm được đường đi ngắn nhất nhưng khác với đường đi ở trên



3. Mức 3

Yêu cầu: trên bản đồ xuất hiện thêm một số điểm đón, tìm đường đi ngắn nhất xuất phát từ S, đi qua tất cả các điểm đón và đến G.
Phương pháp sử dụng: Brute-force search và A-star.
Mô tả thuật toán: sử dụng thư viện Permutations, ta tạo một list toàn bộ các hoán vị của danh sách điểm đón. Thử lần lượt từng cách đi và chọn cách đi có chi phí nhỏ nhất. Để tìm đường đi giữa 2 điểm đón (hoặc giữa điểm đầu/đích với điểm đón), ta gọi lại hàm a_star() đã được cài đặt ở trên.
Tính đầy đủ và tối ưu: thuật toán đảm bảo tìm thấy đường đi ngắn nhất (nếu có).
Độ phức tạp về thời gian: vì sử dụng thuật toán Brute-force nên chương trình sẽ có độ phức tạp về thời gian rất lớn (cần tính chi phí tất cả các đường đi qua 2 điểm đón bất kì).
Độ phức tạp về không gian: tương tự như A-star.

- **Cách cài đặt:**

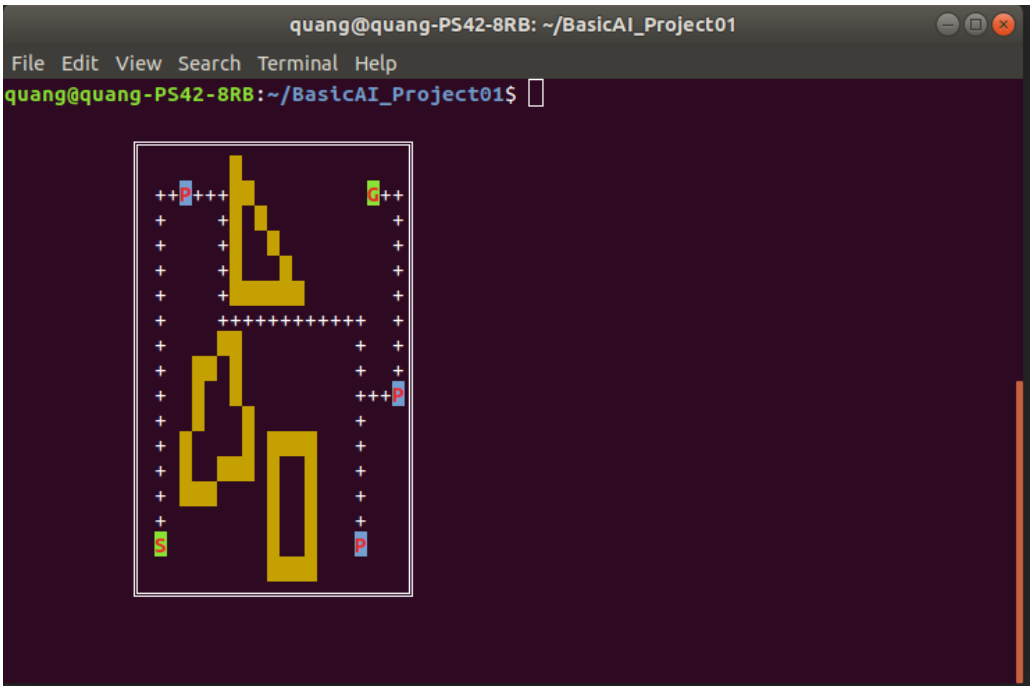
```
# generate all possible route (brute force)
perm = list(permutations(range(len(point_stop))))

min_routing = []
min_cost = width * height
save_start = start
save_end = end
for i in range(len(perm)):
    big_routing = []

    start = save_start
    end = point_stop[perm[i][0]]
    routing = []
    if not a_star():  # find route from 2 point using a-star
        print("Routing not found!")
        sys.exit()
    big_routing.append(routing)

    length = len(perm[i])
```

```
for j in range(length - 1):
    start = point_stop[perm[i][j]]
    end = point_stop[perm[i][j + 1]]
    routing = []
    if not a_star():  # find route from 2 point using a-star
        print("Routing not found!")
        sys.exit()
    big_routing.append(routing)

start = point_stop[perm[i][length - 1]]
end = save_end
routing = []
if not a_star():  # find route from 2 point using a-star
    print("Routing not found!")
    sys.exit()
big_routing.append(routing)

cost = 0  # calculate total cost
for c in big_routing:
    cost = cost + len(c)

if cost < min_cost:
    min_cost = cost
    min_routing = big_routing
```

- **Chạy thử:**

Test 1

Kết quả: tìm được đường đi ngắn nhất

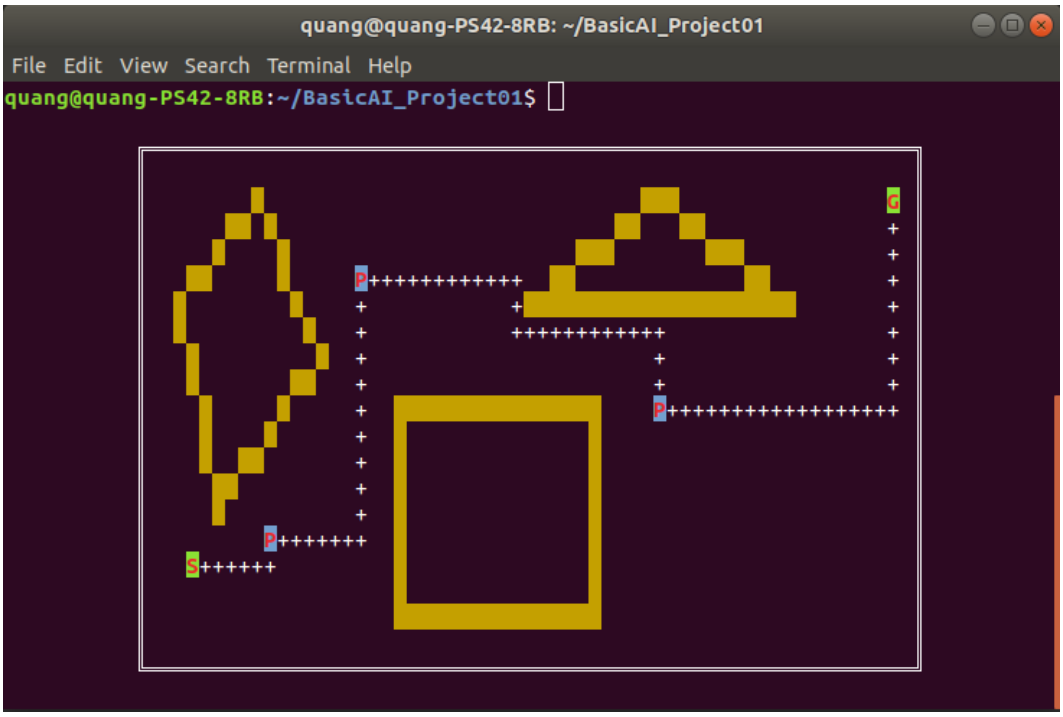


Test 2

Kết quả: không tìm được đường đi

Test 3

Kết quả: tìm được đường đi ngắn nhất



--- THE END ---