KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
227 Nguyễn Văn Cừ, Phường 4, Quận 5, TP.HCM
Điện Thoại: (08) 38.354.266 - Fax:(08) 38.350.096

BÁO CÁO ĐỒ ÁN 1
CƠ SỞ TRÍ TUỆ NHÂN TẠO

Ứng dụng các giải thuật tìm kiếm

Robot tìm đường

GVLT : **Lê Hoài Bắc**

GVTH : **Nguyễn Ngọc Đức**

Lê Ngọc Thành

Nguyễn Ngọc Thảo

Lớp : **CQ2017/21**

Nhóm thực hiện :

Tên	MSSV
Võ Văn Quân	1712698
Châu Thiên Thanh	1712153
Nguyễn Gia Thuận	1712174

MỤC LỤC

1. THÔNG TIN NHÓM VÀ PHÂN CÔNG 3
1.1. Thông tin nhóm 3
1.2. Phân công 3
2. TÓM TẮT BÀI TOÁN : 3
3. ĐỒ THỊ VÀ THIẾT KẾ THUẬT TOÁN: 4
3.1. Vẽ đồ thị minh họa : 4
3.2. Thiết kế thuật toán : 4
i. Thuật toán Breadth first search: 4
ii. Uniform cost search/ Greedy best first search/ A*: 4
iii. Tìm đường đi ngắn nhất khi có thêm những điểm đón 6
4. MÔ TẢ CHẠY CHƯƠNG TRÌNH : CHẠY BẰNG JUPITER NOTEBOOK 6
4.1. Cấu trúc chương trình: 6
4.2. File input 7
5. TEST CASE 7
5.1. Các bộ test case cho level 1 và 2: 7
5.2. Level 1 : Breadth first search 8
5.3. Level 2 : USC/greedy/A* 9
5.4. Level 3 : Điểm đón 11
6. TỔNG KẾT : 12
6.1. Nhận xét về các thuật toán tìm kiếm đã sử dụng 12
6.2. Đánh giá hoàn thành đồ án 12

1. Thông tin nhóm và phân công

1.1. Thông tin nhóm

MSSV	Họ và tên	Email	SĐT
1712153	Châu Thiên Thanh	ctthanh1999@gmail.com	0973435058
1712174	Nguyễn Gia Thuận	ngthuan.fitus@gmail.com	0948480767
1712698	Võ Văn Quân	quan.vovan0209@gmail.com	0932442811

1.2. Phân công

Họ và tên	Công việc	Kết quả
Võ Văn Quân	Code thuật toán uniform cost search, A*. Làm báo cáo	100%
Châu Thiên Thanh	Code thuật toán Breadth first search. Tổng hợp code và chạy chương trình	100%
Nguyễn Gia Thuận	Code thuật toán Best first search. Đọc file input, đồ họa minh họa kết quả chương trình	100%
Các thành viên	Đưa ra bộ test và chỉnh sửa chương trình tối ưu nhất. Hoàn thành chỉnh sửa bổ sung báo cáo	100%

2. Tóm tắt bài toán :

Trên bản đồ phẳng xOy, sử dụng các thuật toán tìm kiếm đã được học để tìm đường đi từ đỉnh S(xs,ys) tới đỉnh G(xg,yg) và không đi qua các chướng ngại vật là đa giác lồi.

Các thuật toán tìm kiếm sử dụng trong đồ án :

- Level 1 : Breadth-First Search.
- Level 2 : Greedy Best-First Search (Manhattan), Uniform Cost Search, A*(Manhattan)
- Level 3 : Dùng A*

Start Point : (u,v)

End Point : (s,t)

Các bước di chuyển của robot (theo 4 hướng, không cho đi chéo)

	(x+1,y)	
(x,y-1)	(x,y)	(x,y+1)
	(x-1,y+1)	

3. Đồ thị và thiết kế thuật toán:

3.1. Vẽ đồ thị minh họa :

Bản đồ được vẽ bởi thư viện matplotlib trong python thông qua mảng lưu thông tin tìm kiếm :

3.2. Thiết kế thuật toán :

i. Thuật toán Breadth first search:

Mô tả thuật toán:

- Từ trạng thái ban đầu, ta xây dựng tập hợp S bao gồm các trạng thái kế tiếp (mà từ trạng thái ban đầu có thể biến đổi thành). Sau đó, ứng với mỗi trạng thái Tk trong tập S, ta xây dựng tập Sk bao gồm các trạng thái kế tiếp của Tk rồi lần lượt bổ sung các Sk vào S. Quá trình này cứ lặp lại cho đến lúc S có chứa trạng thái kết thúc hoặc S không thay đổi sau khi đã bổ sung tất cả Sk.

Mã giả:

```
def BreadthFirstSearch(A, startPoint, endPoint)
    Khởi tạo các mảng:  visit (đánh dấu những tọa độ đã đi qua)
                        trace (truy vết, lưu đường đi)
                        cost (lưu chi phí đã đi từ đỉnh xuất phát
                    tới điểm đang xét)
    u, v = startPoint ;
    s, t = endPoint
    trace[u][v] <= (0,0), cost[u][v] <= 0  #Khởi tạo ban đầu cho
  các mảng
    Thêm (u,v) vào hàng đợi
  While hàng đợi chưa rỗng loop
    Lấy từ hạng đợi 1 điểm x,y
     if x = s và y = t thì
        In ra đường đi từ start -> end
        kết thúc
    Thăm các đỉnh lân cận của x,y nếu đỉnh chưa xét và được
  phép đi thì lưu chi phí của đỉnh đó, thêm vào hàng đợi và đánh
  dấu đỉnh đó đã được thăm
    Trả về ma trận A và chi phí đường đi ngắn nhất
```

ii. Uniform cost search/ Greedy best first search/ A*:

Mô tả thuật toán:

Các thuật toán này đều thông qua việc tra xét những trạng thái có khả năng nhất tính từ vị trí hiện tại (đánh giá chi phí) để tìm đường đi từ đỉnh bắt đầu tới đích sao cho chi phí thấp nhất. Mỗi thuật toán có cách tính chi phí riêng

Cụ thể:

- **UCS:** Chi phí được tính từ nút gốc. Cụ thể trong bài toán là số bước đi tính từ vị trí hiện tại tới điểm đang xét. Thuật toán sẽ rõ ràng hơn trong trường hợp đường đi có trọng số

- **Greedy best first search:** Tìm kiếm tối ưu hóa theo chiều rộng với việc ăn tham theo lựa chọn tốt nhất với chi phí được tính bằng việc ước lượng khoảng cách từ điểm kết thúc tới điểm đang xét. Ở đây ta sử dụng khoảng cách Manhattan để tính

- **A*** : Thuật toán này cũng là một ví dụ của tìm kiếm theo lựa chọn tốt nhất. Chi phí được tính thông qua một đánh giá Heuristic và chi phí bước đi tới thời điểm hiện tại (Thông tin về quá khứ và tương lai).

Mã giả:

```
def Search(A, startPoint, endPoint)
    Khởi tạo các mảng:  visit (đánh dấu những tọa độ đã đi qua)
                        trace (truy vết, lưu đường đi)
                        cost (lưu chi phí đã đi từ đỉnh xuất phát tới điểm đang xét)
                        h (hàm heuristic, sử dụng trong thuật toán A*, Manhattan, sử dụng trong GreedyBFS)
    u, v = startPoint ; s, t = endPoint
    trace[u][v] <= (0,0), cost[u][v] <= 0  #Khởi tạo ban đầu cho các mảng
    Thêm (u,v) vào hàng đợi
    While hàng đợi chưa rỗng loop
        Lấy từ hạng đợi 1 điểm x,y có chi phí thấp nhất (chi phí này được tính theo từng thuật toán)
        if x = s và y = t thì
            In ra đường đi từ start -> end
```

kết thúc

Thăm các đỉnh lân cận của x,y nếu đỉnh chưa xét và được phép đi thì lưu chi phí của đỉnh đó, thêm vào hàng đợi và đánh dấu đỉnh đó đã được thăm

Trả về ma trận A và chi phí đường đi ngắn nhất

iii. **Tìm đường đi ngắn nhất khi có thêm những điểm đón**

- ***Mô tả***: Duyệt qua tất cả các hoán vị thứ tự các điểm đón và tìm ra hoán vị có tổng chi phí khi đi qua các điểm đón theo thứ tự đó là nhỏ nhất.
- ***Mã giả***:

+ Khởi tạo mảng permutations là tập các hoán vị các điểm đón
+ For X in permutations:

cost <- inf (Khởi tạo giá trị ban đầu)

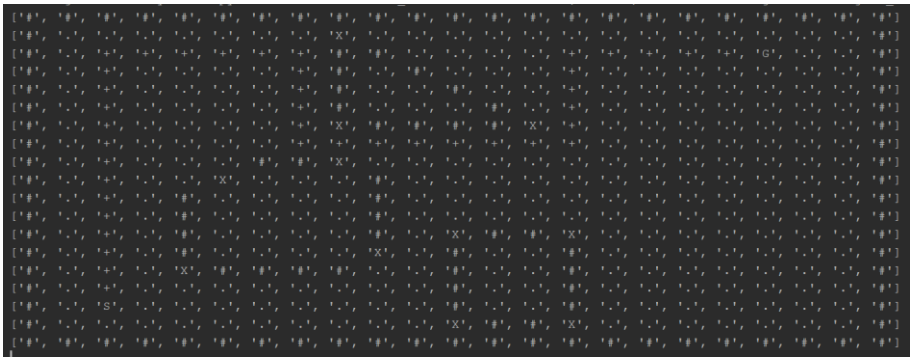
cost <- Chi phí khi đi qua các điểm đón theo thứ tự X (dùng 1 trong 4 thuật toán ở trên để tìm chi phí giữa 2 điểm đón trong X)

Lưu lại giá trị cost nào là nhỏ nhất sau khi duyệt hết các hoán vị, đồng thời lưu lại thứ tự X tương ứng với chi phí đó

+ Xuất kết quả tương ứng với đường đi theo X được trả về

4. Mô tả chạy chương trình : chạy bằng Jupiter notebook

4.1. **Cấu trúc chương trình:** Các thuật toán được chia làm các file riêng, trả về mảng lưu vết đường đi từ S->G với kí hiệu (# . +)



Một số tên biến và chức năng dùng trong chương trình:

- **A** : ma trận. Được khởi tạo từ input và dùng thao tác trong toàn bộ chương trình
- **trace**: Mảng lưu đường đi trong quá trình duyệt
- **visit**: Mảng đánh dấu các điểm đã/chưa duyệt
- **cost**: Mảng lưu chi phí đường đi ngắn nhất (số bước đi từ điểm start - end)
- **Xmax, Ymax**: Kích thước ma trận

Ngoài ra còn một số biến được dùng để lưu các giá trị trong quá trình tính toán trong suốt chương trình

4.2. File input

- File input1_**c**.txt (với **c** là số thứ tự test case) có cấu trúc như sau : (dùng để chạy level 1,2)
  - Dòng 1 : kích thước [w,h] của bản đồ.
  - Dòng 2 : 2 cặp tọa độ Start(u,v) và goal(s,t)
  - Dòng 3 : số lượng chướng ngại vật (đa giác) - n
  - N dòng tiếp theo : dòng thứ i chưa thông tin đa giác thứ i : cứ 2 cặp số là tọa độ 1 đỉnh.
- File input2_**c**.txt (với **c** là số thứ tự test case) có cấu trúc tương tự input1, nhưng dòng 2 : có thêm các cặp điểm được gọi là điểm đón (tức là robot sẽ qua các điểm đó trước khi tới điểm Goal)
  Lưu ý : các đỉnh được nhập theo 1 chiều nhất định, tránh trường hợp demo minh họa đồ thi gặp đa giác lõm.

Ví dụ :

Input1.txt :

22,18
2,2,19,16
3
4,4,5,9,8,10,9,5
8,12,8,17,13,12
11,1,11,6,14,6,14,1

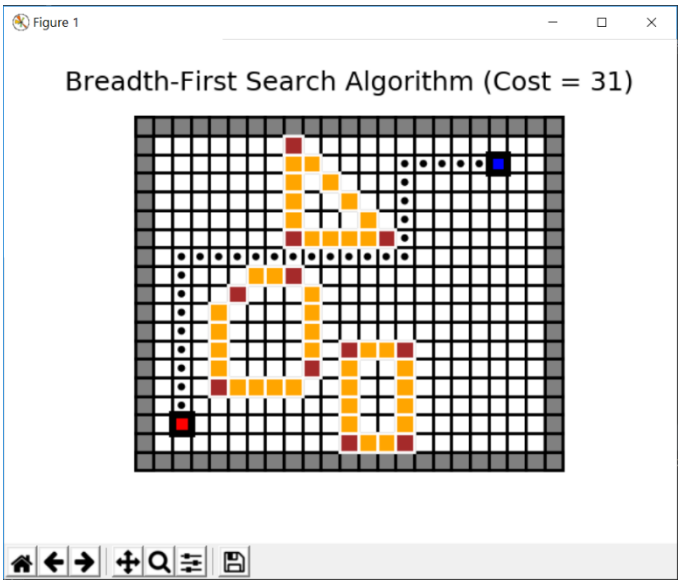
5. Test case

5.1. Các bộ test case cho level 1 và 2:

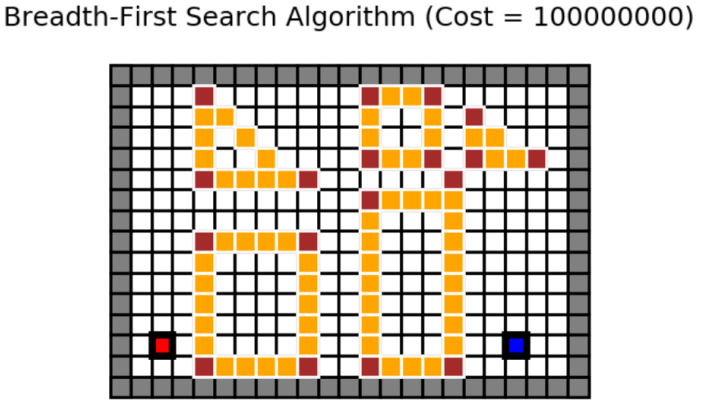
Testcase 1: input1_1.txt	Testcase 2: input1_2.txt	Testcase 3 : input1_3.txt	Test case 4: input1_4.txt
22,18 2,2,19,16 3 4,4,5,9,8,10,9,5 8,12,8,17,13,12 11,1,11,6,14,6,14,1	22,15 2,2,19,2 5 4,10,9,10,4,14 4,1,9,1,9,7,4,7 12,1,16,1,16,10,12,9 17,11,20,11,17,13 12,11,15,11,15,14,12,14	22,15 2,13,19,2 4 4,10,4,14,9,10 3,5,4,7,8,6,9,2 12,1,12,9,16,10,20,6,16,1 17,11,17,13,20,11	22,15 2,13,19,2 4 4,10,4,14,9,10 3,5,4,9,8,6,9,2 12,1,11,11,16,10,20,6,16,1 17,11,17,13,20,11

5.2. Level 1 : Breadth first search

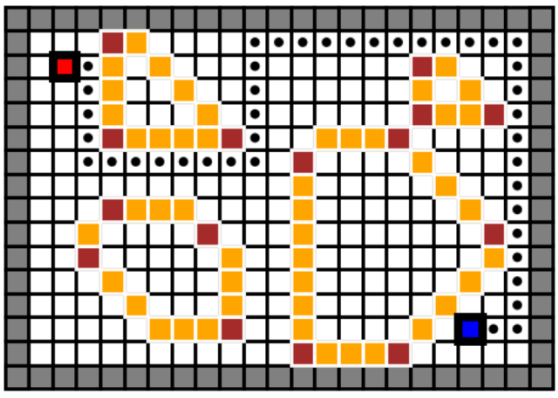
#testcase 1 Chi phí đường đi : 31



#testcase 2 : Không có đường đi => chỉ vẽ bản đồ và default chi phí = 100000000

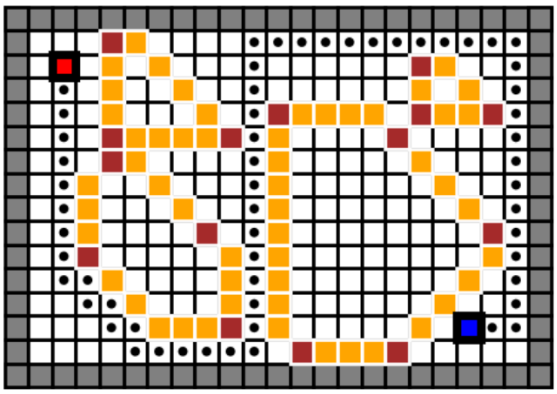


#testcase 3:



#testcase 4:

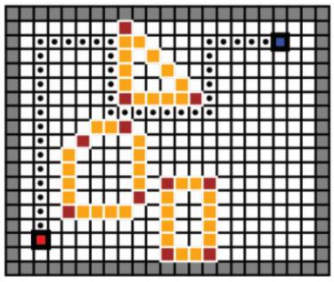
Breadth-First Search Algorithm (Cost = 58)



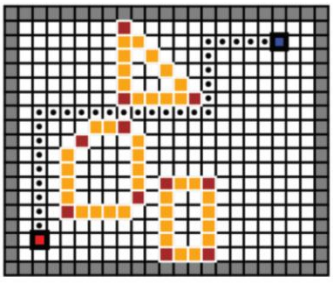
5.3. Level 2 : USC/greedy/A*

***#testcase 1*:**

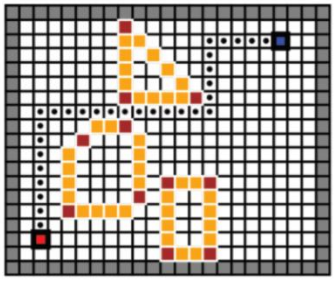
Greedy Best-First Search Algorithm (Cost = 41)



A* Search Algorithm (Cost = 31)



Uniform cost Search Algorithm (Cost = 31)

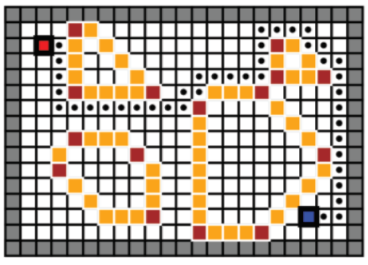


Đánh giá : đường đi của Greedy có chi phí cao hơn với A* và USC

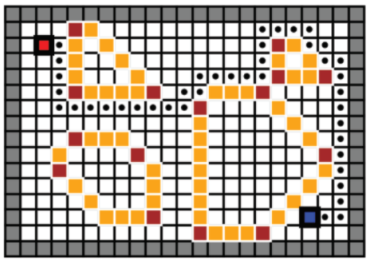
#testcase2 : không tìm thấy đường đi => tương tự lv1

#testcase3 :

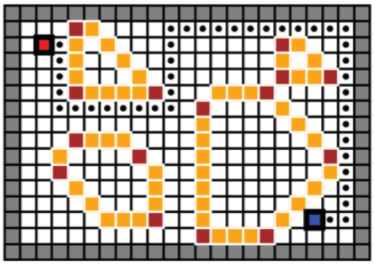
Greedy Best-First Search Algorithm (Cost = 42)



A* Search Algorithm (Cost = 42)



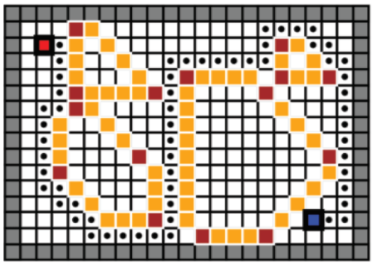
Uniform cost Search Algorithm (Cost = 42)



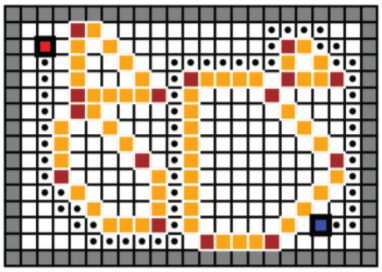
Đánh giá : mặc dù thuật toán tìm kiếm mù (USC) cho kết quả đường đi khác với 2 thuật toán còn lại nhưng chi phí đường đi là như nhau (bằng 42)

#testcase4 :

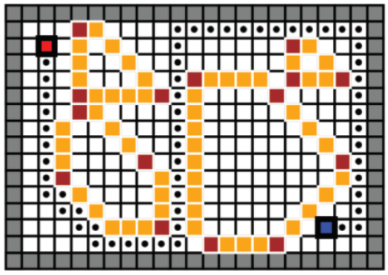
Greedy Best-First Search Algorithm (Cost = 60)



A* Search Algorithm (Cost = 58)



Uniform cost Search Algorithm (Cost = 58)



Đánh giá : một lần nữa Greedy lại cho ra chi phí đường đi cao hơn so với 2 thuật toán còn lại

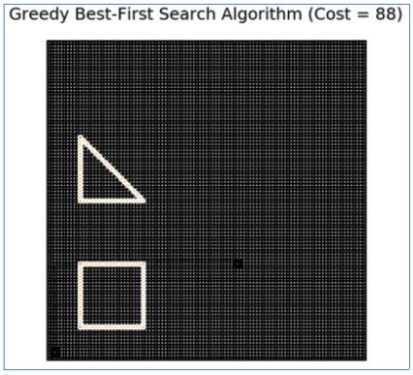
#testcase 5 : bộ input lớn, kích thướt bản đồ lớn nên không dùng breath-first search

Input1_5.txt :

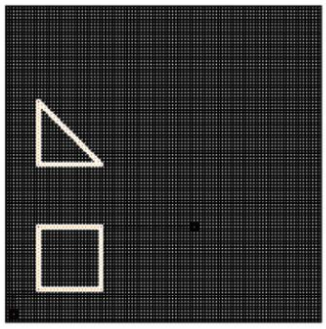
100,100
2,2,60,30
2
10,10,10,30,30,30,30,10
10,50,30,50,10,70

⇨ Kết quả : Chúng ta chú ý đến chi phí của từng thuật toán do vấn đề đồ họa chưa tính toán với kích thước dữ liệu lớn.

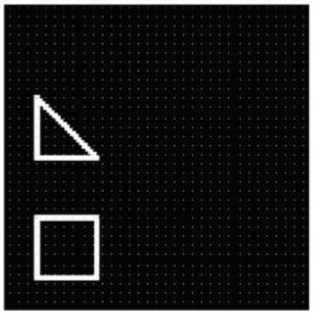
Greedy Best-First Search Algorithm (Cost = 88)



A* Search Algorithm (Cost = 86)



Uniform cost Search Algorithm (Cost = 86)



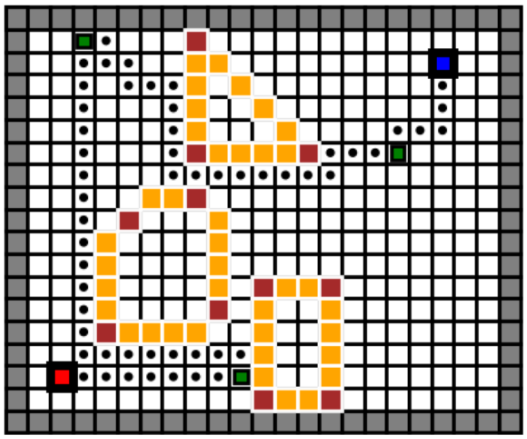
Đánh giá : Greedy tiếp tục cho chi phí đường đi lớn hơn 2 thuật toán còn lại. Ta xét thuật toán A* và UCS có cùng chi phí, nhưng lúc này UCS dường như không khác gì BFS bình thường (loang ra nhiều đỉnh để xét). Nên A* cho thời gian chạy nhanh hơn rất nhiều

5.4. Level 3 : Điểm đón

Test case 1: input2_1.txt	Test case 2: input2_2.txt	Test case 3
22,18 2,2,19,16,10,2,17,12,3,17 3 4,4,5,9,8,10,9,5 8,12,8,17,13,12 11,1,11,6,14,6,14,1	22,15 2,13,19,2,2,2,10,2,8,13,19,10,6,8 4 4,10,4,14,9,10 3,5,4,7,8,6,9,2 12,1,12,9,16,10,20,6,16,1 17,11,17,13,20,11	22,15 2,2,19,2,2,13,10,2,8,14 5 4,10,9,10,4,14 4,1,9,1,9,7,4,7 12,1,16,1,16,10,12,9 17,11,20,11,17,13 12,11,15,11,15,14,12,14

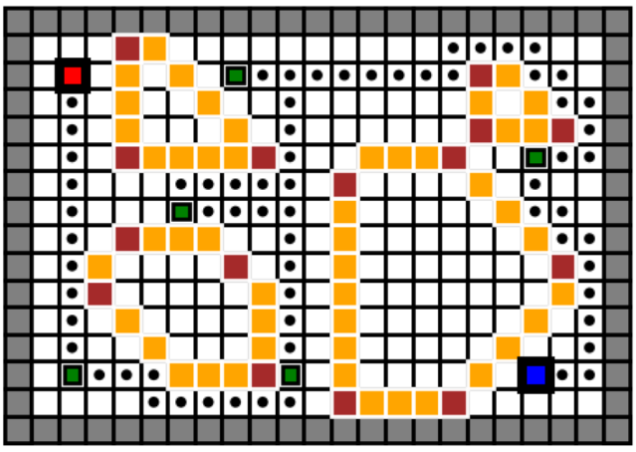
#testcase1

Algorithm used: A* (Cost = 57)

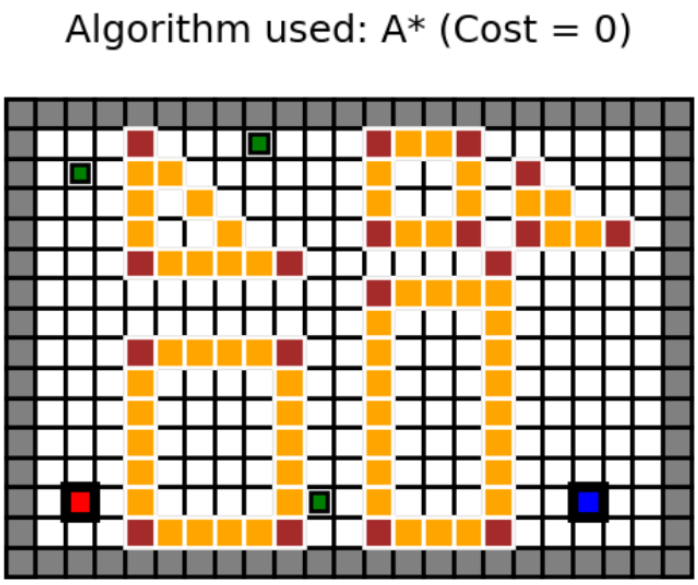


#testcase2 :

Algorithm used: A* (Cost = 74)



#testcase 3 : không tìm được đường đi



6. Tổng kết :

6.1. Nhận xét về các thuật toán tìm kiếm đã sử dụng

- **Thuật toán Breadth first search** : cho độ chính xác cao, nhưng thời gian chạy lâu.
- **Thuật toán Greedy BFS (manhattan)** : thời gian chạy nhanh, nhưng độ tối ưu về đường đi chưa cao.
- **Thuật toán Uniform cost search** : thời gian chạy nhanh đối với input nhỏ, độ chính xác cao.
- **Thuật toán A*** : thuật toán tối ưu về thời gian và độ chính xác. (thời gian có thể chậm hơn Greedy)

6.2. Đánh giá hoàn thành đồ án

Level	Mức độ hoàn thành
1	100%
2	100%
3	100%