

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA: CÔNG NGHỆ THÔNG TIN



BÁO CÁO
Project 1- Robot Tìm Đường

LỚP : CƠ SỞ TRÍ TUỆ NHÂN TẠO - CQ2017/21

GVLT : Lê Hoài Bắc

GVHDTH: Lê Ngọc Thành - Nguyễn Ngọc Thảo - Nguyễn Ngọc Đức

SINH VIÊN THỰC HIỆN:

1712787 - Nguyễn Văn Thìn

1712842 - Huỳnh Lương Phương Trúc

1712920 - Nguyễn Minh Vũ

Tp. Hồ Chí Minh, ngày 25 tháng 10 năm 2019

MỤC LỤC

1	Tổng quan & Milestone	3
	Thông tin nhóm	3
	Milestone	3
2	Thực hiện yêu cầu	4
2.1	Thuật toán sử dụng	4
	Các thuật toán được sử dụng	4
	Cách duyệt đồ thị	4
	Thuật giải	5
	Queue - Priority_Queue	6
	Các thuật toán được cài đặt theo từng đối tượng Class (OOP)	7
2.2	Chương trình & File Input.....	8
	Mức độ thực hiện chương trình	8
	File Input	9
	Cách thức chạy chương trình	9
2.3	Kết quả chạy chương trình	10
	Mức 1	10
	Mức 2	10
	Mức 3	Error! Bookmark not defined.3
	Mức 4	Error! Bookmark not defined.5
2.4	References	Error! Bookmark not defined.7

1

Tổng quan & Milestone

Thông tin nhóm

MSSV	Họ tên	Email
1712787	Nguyễn Văn Thìn	vanthin7111999@gmail.com
1712842	Huỳnh Lương Phương Trúc	huynhtruc0309@gmail.com
1712920	Nguyễn Minh Vũ	vumnguyenn@gmail.com

Milestone

Cột mốc	Công việc dự kiến	Ước lượng (hour)	Sản phẩm
29/09/2019 Tìm hiểu yêu cầu đề bài	Tìm hiểu kiến thức các thuật toán search	8	Thuật toán BFS, DFS, UCS, Greedy, A*
	Tìm hiểu thư viện đồ họa, các thư viện cần thiết khác trong Python	4	pygame, bresenham, messagebox, numpy, heapd, math Tk, time, sys, matplotlib, shapely
	Đưa ra các bộ test cho từng thuật toán	2	8 bộ test
18/10/2019 Thực hiện lập trình chương trình chạy	Thực hiện đọc file từ bộ test đưa vào	1	File.py main.py
	Thực hiện code và cài đặt thuật toán DFS, BFS	8	BFS.py DFS.py
	Code hàm đồ họa GUI cho chương trình, thực hiện mô phỏng trên giao diện đồ họa	8	GUI.py Giao diện đồ họa
	Thực hiện code và cài đặt thuật toán (A*, Greedy Best First Search, Uniform Cost Search)	16	Greedy.py Astar.py UCS.py PriorityQueue.py Heuristic.py

	Thực hiện code phần tìm đường đi ngắn nhất với các điểm đón Sử dụng thuật toán Astar để cài đặt	8	Astar_NPoint.py File_Npoint.py GUI_NPoint.py
	Thực hiện code phần tìm đường đi với trường hợp các hình chuyển động	12	moving_obstacles.py
	Kiểm thử, sửa lỗi và thực hiện yêu cầu nâng cao	5	Hoàn thiện Solution
24/10/2019 Hoàn tất báo cáo	Hoàn tất báo cáo	2	Báo cáo đồ án
25/10/2019 Nộp đồ án	Nộp đồ án về Moodle	0.5	Toàn bộ đồ án được nộp về Moodle

2

Thực hiện yêu cầu đề bài

2.1 Chi tiết thuật toán sử dụng

Các thuật toán được sử dụng

Có tất cả 5 thuật toán được dùng để thử nghiệm, bao gồm:

- **Breadth First Search** (Tìm kiếm theo chiều rộng)
- **Depth First Search** (Tìm kiếm theo chiều sâu)
- **Uniform Cost Search** (Tìm kiếm chi phí đồng nhất)
- **Greedy Best-First Search** (Tìm kiếm tối ưu kiểu tham lam)
- **A*** (Heuristic: Euclidean & Manhattan distances).

Cách duyệt đồ thị

Thực hiện duyệt qua 8 ô lân cận tại điểm đang xét rồi lựa chọn điểm đi tiếp theo

```
neighbors = [(-1, -1), (-1, 0), (-1, 1), (0, 1), (1, 1), (1, 0), (1, -1), (0, -1)]
```

$(x+1, y-1)$ sqrt(2)	$(x+1, y)$ (1)	$(x+1, y+1)$ sqrt(2)
$(x, y-1)$ (1)	(x, y)	$(x, y+1)$ (1)
$(x-1, y-1)$ sqrt(2)	$(x-1, y)$ (1)	$(x-1, y+1)$ sqrt(2)

Nhóm lấy chi phí đường đi thẳng là **1**, đường đi chéo lấy xấp xỉ **sqrt(2)=1.5**

Thuật giải

Thuật toán BFS, DFS

- Open: là tập hợp các đỉnh chờ được xét ở bước tiếp theo theo hàng đợi (hàng đợi: dãy các phần tử mà khi thêm phần tử vào sẽ thêm vào cuối dãy, còn khi lấy phần tử ra sẽ lấy ở phần tử đứng đầu dãy).
- Close: là tập hợp các đỉnh đã xét, đã duyệt qua.
- s: là đỉnh xuất phát, đỉnh gốc ban đầu trong quá trình tìm kiếm.
- g: đỉnh đích cần tìm.
- p: đỉnh đang xét, đang duyệt.

BFS	DFS
<ul style="list-style-type: none"> • Bước 1: Tập Open chứa đỉnh gốc s chờ được xét. • Bước 2: Kiểm tra tập Open có rỗng không. <ul style="list-style-type: none"> ◦ Nếu tập Open không rỗng, lấy một đỉnh ra khỏi tập Open làm đỉnh đang xét p. Nếu p là đỉnh g cần tìm, kết thúc tìm kiếm. ◦ Nếu tập Open rỗng, tiến đến bước 4. • Bước 3: Đưa đỉnh p vào tập Close, sau đó xác định các đỉnh kề với đỉnh p vừa xét. Nếu các đỉnh kề không thuộc tập Close, đưa chúng vào cuối tập Open. Quay lại bước 2. • Bước 4: Kết luận không tìm ra đỉnh đích cần tìm. 	<ul style="list-style-type: none"> • Bước 1: Tập Open chứa đỉnh gốc s chờ được xét. • Bước 2: Kiểm tra tập Open có rỗng không. <ul style="list-style-type: none"> ◦ Nếu tập Open không rỗng, lấy một đỉnh ra khỏi tập Open làm đỉnh đang xét p. Nếu p là đỉnh g cần tìm, kết thúc tìm kiếm. ◦ Nếu tập Open rỗng, tiến đến bước 4. • Bước 3: Đưa đỉnh p vào tập Close, sau đó xác định các đỉnh kề với đỉnh p vừa xét. Nếu các đỉnh kề không thuộc tập Close, đưa chúng vào đầu tập Open. Quay lại bước 2. • Bước 4: Kết luận không tìm ra đỉnh đích cần tìm.

Thuật toán UCS, A*, Greedy

Do cả ba loại thuật toán **UCS**, **A***, **Greedy** được sử dụng đều dùng chung một loại cấu trúc (chỉ khác nhau ở giá trị ưu tiên của đỉnh) nên ở đây sẽ chỉ trình bày mã thuật giải chung của cả 3 loại:

1. Gọi Open: tập các trạng thái đã được sinh ra nhưng chưa được xét đến.
2. Close: tập các trạng thái đã được xét đến.
3. $\text{Cost}(p, q)$: là khoảng cách giữa p, q .
4. $g(p)$: khoảng cách từ trạng thái đầu đến trạng thái hiện tại p .
5. $h(p)$: giá trị được lượng giá từ trạng thái hiện tại đến trạng thái đích.
6. $f(p) = g(p) + h(p)$.
 1. Bước 1:
 - Open: = {s}
 - Close: = {}
 2. Bước 2: while (Open != {})
 1. Chọn trạng thái (đỉnh) tốt nhất p trong Open (xóa p khỏi Open).
 2. Nếu p là trạng thái kết thúc thì thoát.
 3. Chuyển p qua Close và tạo ra các trạng thái kế tiếp q sau p .
 1. Nếu q đã có trong Open
 - Nếu $g(q) > g(p) + \text{Cost}(p, q)$
 - $g(q) = g(p) + \text{Cost}(p, q)$
 - $f(q) = g(q) + h(q)$
 - $\text{prev}(q) = p$ // đỉnh cha của q là p
 2. Nếu q chưa có trong Open
 - $g(q) = g(p) + \text{cost}(p, q)$
 - $f(q) = g(q) + h(q)$
 - $\text{prev}(q) = p$
 - Thêm q vào Open
 3. Nếu q có trong Close
 - Nếu $g(q) > g(p) + \text{Cost}(p, q)$
 - Bỏ q khỏi Close
 - Thêm q vào Open
3. Bước 3: Không tìm được.

Lưu ý: UCS: không sử dụng heuristic Greedy : $g(p) = 0$ nên $f(q) = h(q)$ A* : $f(q) = g(q) + h(q)$

Hàm Heuristic: ước lượng chi phí đường đi ngắn nhất từ 1 điểm n tới đích

Eclidean distance (dạng khoảng cách đường chim bay)

def heuristic(a, b):

return math.sqrt((b[0] - a[0])**2 + (b[1] - a[1])**2)

Manhattan distance

def heuristic(a, b):

return abs(b[0] - a[0]) + abs(b[1] - a[1])

Queue & Priority_Queue

Priority_queue: hàng đợi ưu tiên được sử dụng trong thuật toán A*, Greedy

Do trong Python không có sẵn thư viện **Priority_queue** nên nhóm đã thực hiện xây dựng một class **Priority_queue** (trên nền tảng Heap) (**PriorityQueue.py**):

Queue: hàng đợi được sử dụng trong thuật toán BFS, DFS

Trong Python có sẵn thư viện **SimpleQueue**

```
1 import heapq
2
3 class PriorityQueue:
4     def __init__(self):
5         self.elements = []
6
7     def empty(self):
8         return len(self.elements) == 0
9
10    def put(self, item, priority):
11        heapq.heappush(self.elements, (priority, item))
12
13    def get(self):
14        return heapq.heappop(self.elements)
15
16    def update(self, item, priority):
17        for index, (p, i) in enumerate(self.elements):
18            if i == item:
19                if p <= priority:
20                    break
21                del self.elements[index]
22                self.elements.append((priority, item))
23                heapq.heapify(self.elements)
24                break
25        else:
26            self.put(item, priority)
27
```

Các thuật toán được cài đặt theo từng đối tượng Class (OOP)

Ví dụ đối với thuật toán BFS

```
class BFS:
    #hàm khai báo các thành phần của class
    def __init__(self, input_name):

    #hàm lấy thông tin từ map bao gồm kích thước map, điểm start, điểm goal
    def getMapInformation(self):

    #hàm khởi tạo class
```

```

def getStart(self):

#hàm trả về kết quả đường đi ngắn nhất
def trackingPath(self):

#hàm kiểm tra 8 điểm lân cận có nằm trong map hay không và có đi vào phía trong vật hay không
def isValid(self, neighbor):

#hàm cài đặt thuật toán
def BFS(self, gui):

#hàm chạy thuật toán với testcase
def runBFS(self, gui, input_name):

```

Các thuật toán còn lại cũng sử dụng lại cấu trúc class trên.

Ở mức 4: chương trình được viết demo cho duy nhất thuật toán A*, không sử dụng OOP mà sử dụng các biến global

2.2

Chương trình & File Input

Mức độ hoàn thành bài tập:

Mức	Mô tả	Độ hoàn thành
1 (40%)	Cài đặt thành công 1 thuật toán để tìm đường đi từ S tới	100%
2 (30%)	Cài đặt ít nhất 3 thuật toán khác nhau (ví dụ tìm kiếm mù, tham lam, heuristic, ...). Báo cáo nhận xét sự khác nhau khi chạy thử 3 thuật toán.	100%
3 (30%)	Trên bản đồ sẽ xuất hiện thêm một số điểm khác được gọi là điểm đón. Tìm đường đi ngắn nhất qua các điểm đón rồi đến đích	100%
4 (điểm cộng 10%)	Các hình đa giác có thể di động được với tốc độ h tọa độ/s	90% (chưa xử lý hết trường hợp biên, thuật toán chưa được tối ưu)
5 (điểm cộng 10%)	Thể hiện mô hình trên không gian 3 chiều	0%

File Input

File Input (được lưu dưới dạng thuần văn bản) có cấu trúc như sau:

- Dòng 1: Kích thước [w, h] của bản đồ.
- Dòng 2: 2 cặp tọa độ (sX, sY), (gX, gY) là điểm start & goal
Lưu ý: đối với bài toán có thêm điểm đón thì sẽ thêm các điểm đón vào sau 2 cặp tọa độ này
- Dòng 3: Số lượng chướng ngại vật (đa giác) **n**
- n dòng tiếp theo: Dòng thứ **i**: Chứa thông tin đa giác thứ **i**: Cứ 2 cặp số là tọa độ của một đỉnh.
Lưu ý: Các đỉnh được nhập theo 1 chiều nhất định (ngược/cùng chiều). Chương trình sẽ vẽ liên tục các cạnh nên nếu nhập lộn xộn sẽ ra đa giác lõm.

Ví dụ file Input **input1.txt**: test trường hợp 3 đa giác

```
22,18
3,16,20,8
3
4,4,5,9,8,10,9,5
8,12,8,17,13,12
11,1,11,6,14,6,14,1
```

input2.txt – test trường hợp không tìm thấy đường đi

input3.txt – test trường hợp map lớn hơn và số đa giác là 5

input4.txt – test trường hợp map lớn hơn và số đa giác là 3

input5.txt – test trường hợp có 3 điểm đón, 3 đa giác

input6.txt – test trường hợp có 3 điểm đón, 5 đa giác

input7.txt – test trường hợp không tìm thấy đường đi (có 3 điểm đón, 3 đa giác)

extreme.txt – test cực khó cho phần vật cản di chuyển

Cách thức chạy chương trình

Các ví dụ sử dụng Command Line để chạy chương trình

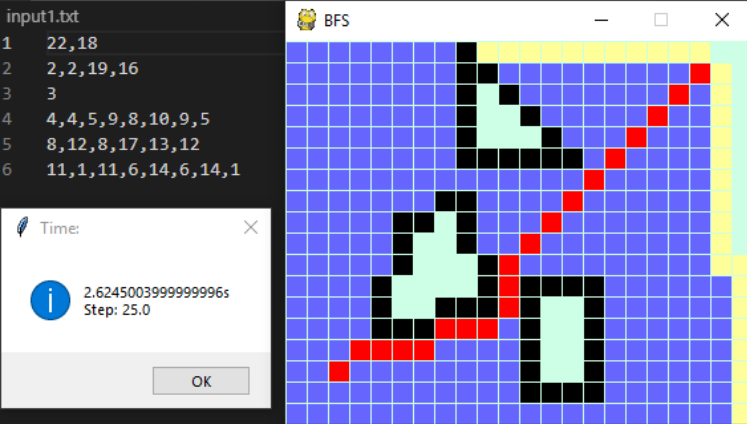
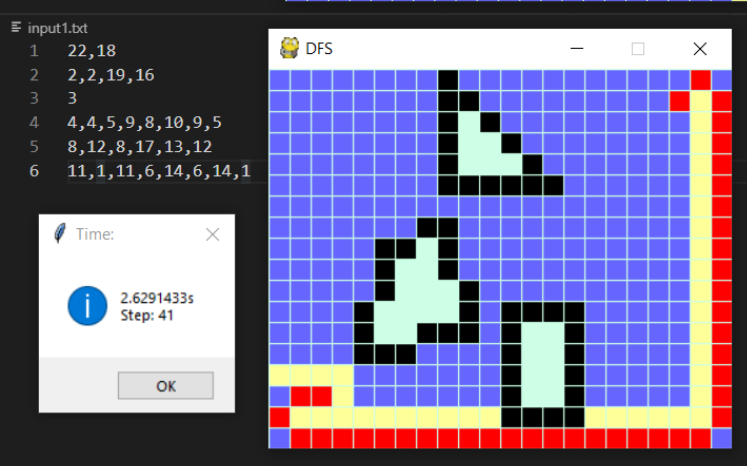
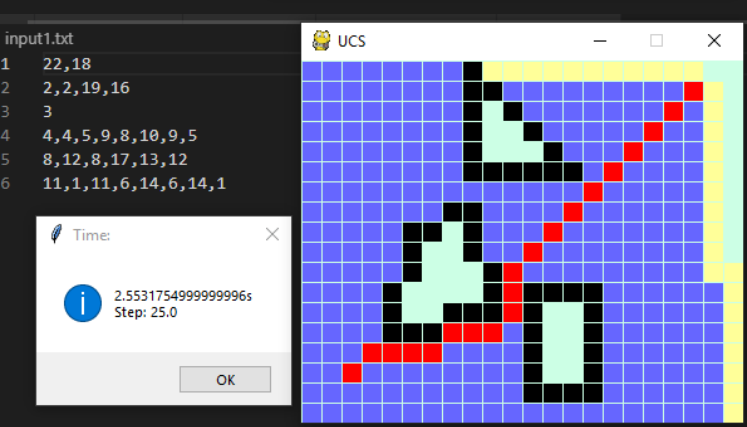
```
>python main.py BFS input1.txt
>python main.py DFS input1.txt
>python main.py UCS input1.txt
>python main.py AStar input1.txt
>python main.py Greedy input1.txt
>python main.py AStarNPoint input5.txt (với input5.txt là file có nhiều điểm đón)
>python moving_obstacles.py input1.txt 2 (số 2: vật cản di chuyển 2 tọa độ trong 0.1s)
```

2.3

Kết quả chạy chương trình

Mức 1 và 2: Áp dụng thuật toán **BFS, DFS, UCS, Greedy Best First Search, A***

MAP 1: input1.txt

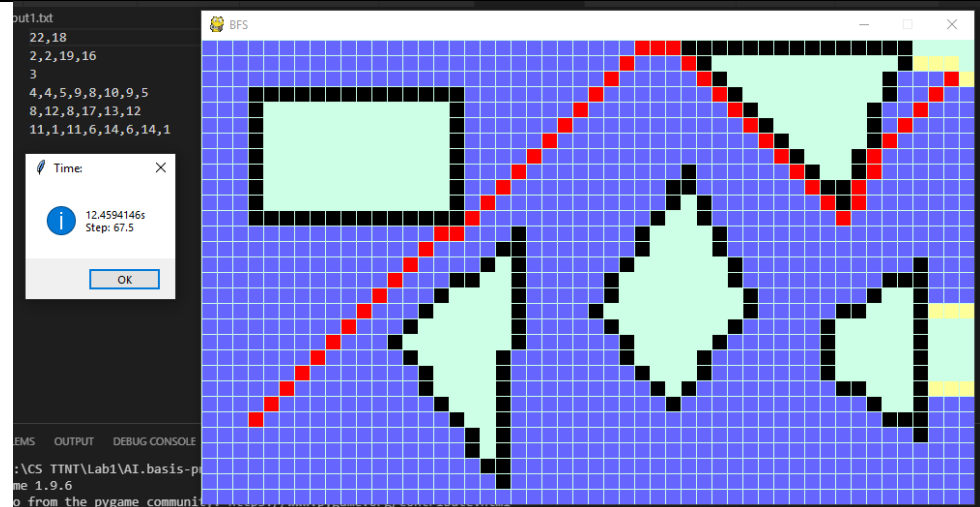
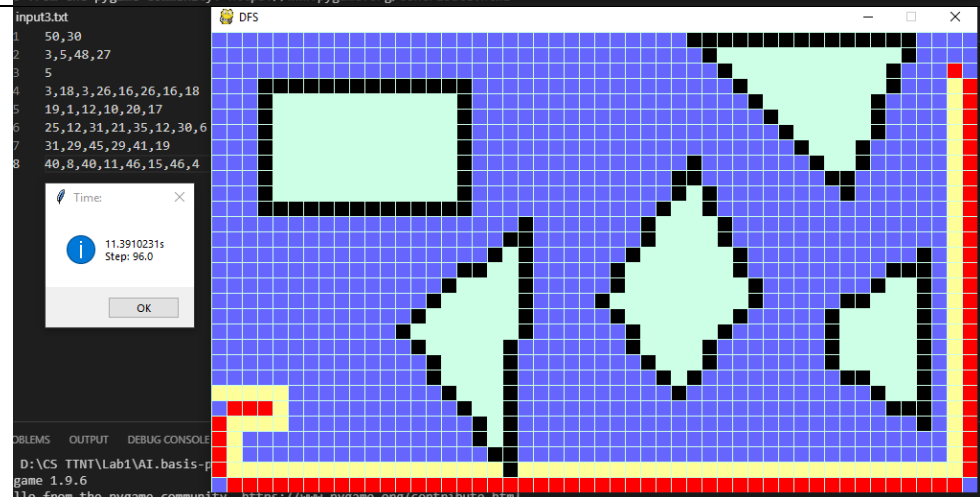
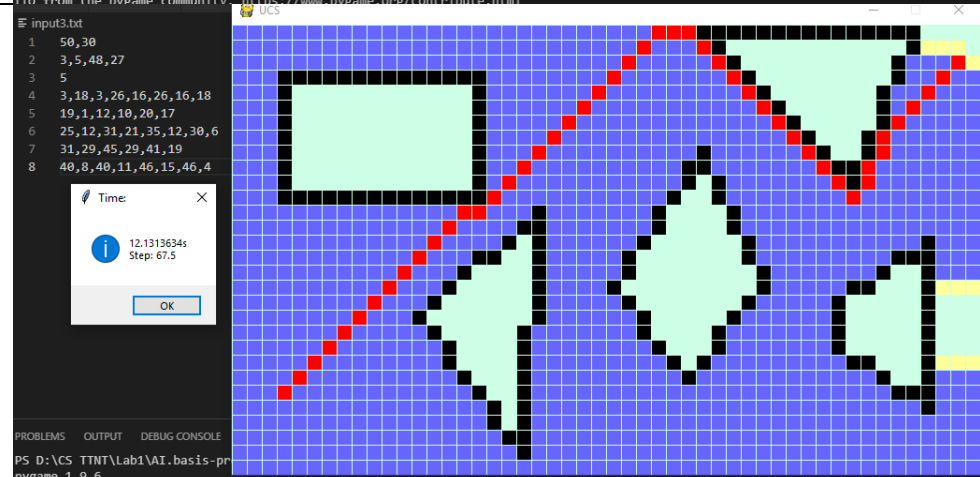
	<p>BFS</p> <p><i>BFS cho lời giải tốt hơn DFS do nó mở rộng mọi hướng để đến đích. Thời gian chênh lệch không đáng kể so với DFS</i></p>
	<p>DFS</p> <p><i>DFS cho lời giải không tốt bằng BFS do nó chỉ tập trung mở rộng theo một hướng. Thời gian chênh lệch không đáng kể so với BFS</i></p>
	<p>UCS</p> <p><i>Cách hoạt động tương tự BFS nhưng ưu tiên mở theo hướng có chi phí ít nhất tính từ nguồn. Thời gian chênh lệch không đáng kể</i></p>

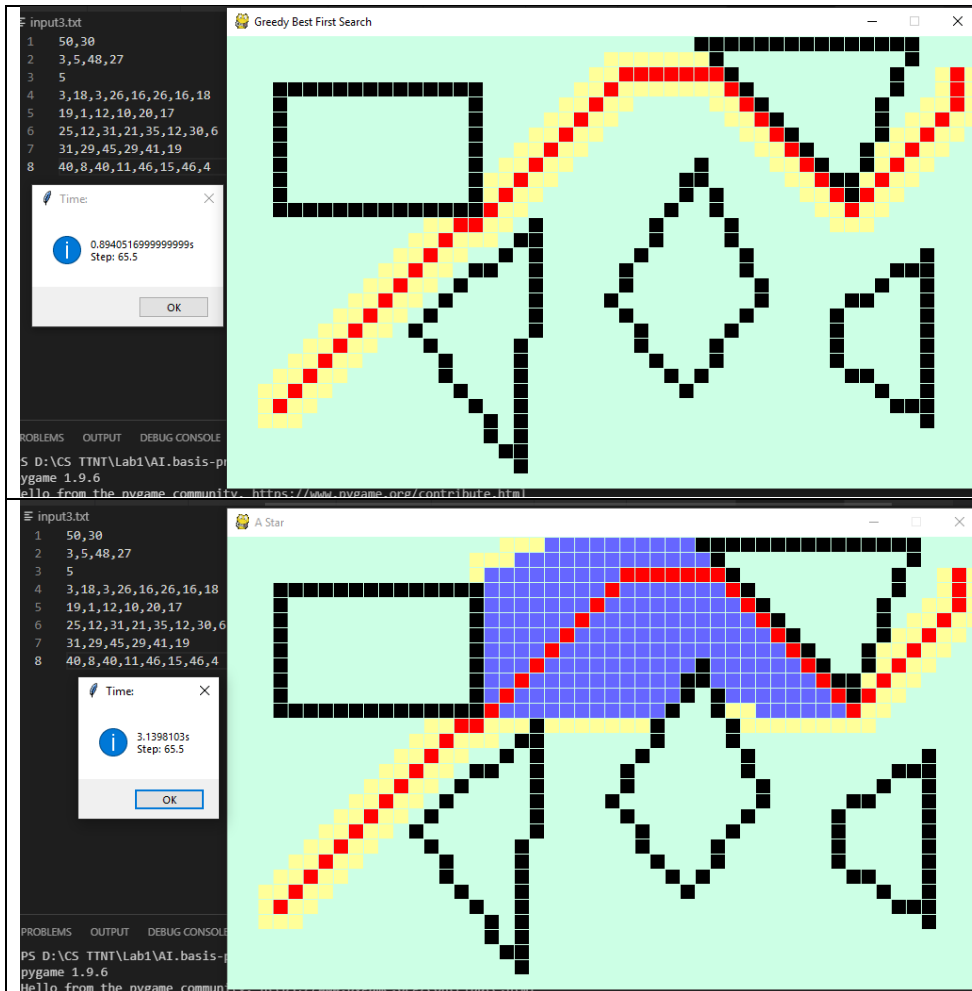
	<p>Greedy</p> <p>Chạy lâu nhất trong trường hợp này. Do Greedy chỉ quan tâm điểm gần đích nhất. Nên khi vừa đến điểm giữa của 2 hình, nó tiếp tục đi lên trên do gần với đích hơn. Dẫn đến việc đi lẩn quẩn bên phải tam giác.</p>
	<p>A*</p> <p>Chạy nhanh nhất trong các thuật toán. Trong map này, việc tìm kiếm có thông tin đã khiến A* và Greedy quyết định chọn hướng có khoảng cách chim bay gần đích nhất, nhưng không may là đường đi thực tế không được tốt như vậy.</p>

MAP 2: input2.txt

Tất cả các thuật toán đều phải quét hết bản đồ không tìm thấy đường đi.

MAP 3: input3.txt

	<p>BFS</p> <p>Do map lớn nên việc mở rộng mọi hướng làm BFS tốn nhiều thời gian để đến kết quả. Thời gian chênh lệch không đáng kể so với DFS</p>
	<p>DFS</p> <p>Do cách chọn hướng không tốt nên việc mở rộng theo một hướng của DFS cho ra kết quả không tốt. Nhưng thời gian chênh lệch không đáng kể so với BFS</p>
	<p>UCS</p> <p>Vẫn cho ra kết quả tương tự về thời gian và đường đi do UCS cũng thuộc vào loại tìm kiếm mù như BFS và DFS</p>



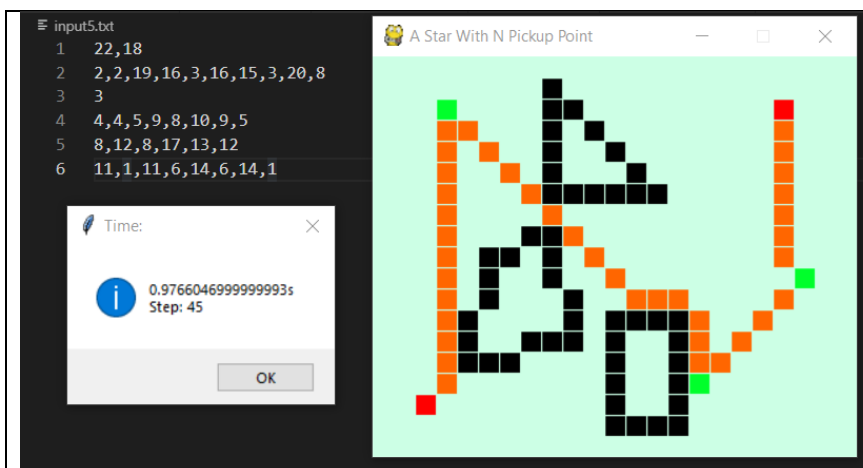
Greedy

Cho kết quả nhanh nhất và tốt nhất. Greedy lúc này thể hiện tốt được điểm mạnh khi không phải tốn nhiều thời gian để mở rộng vùng

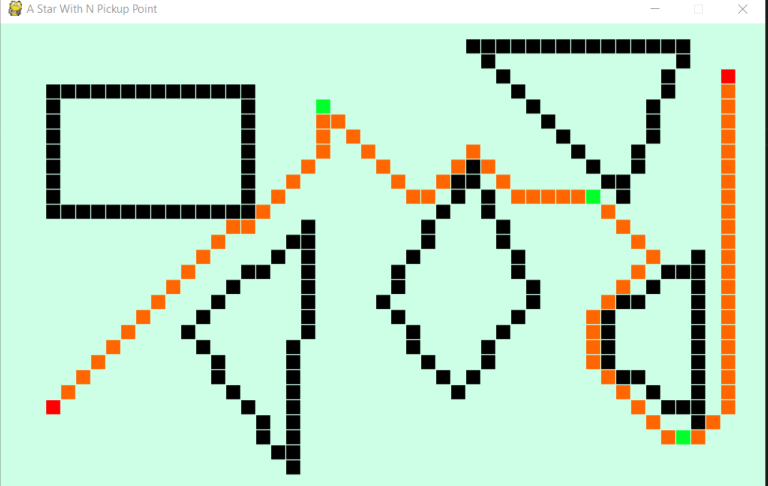
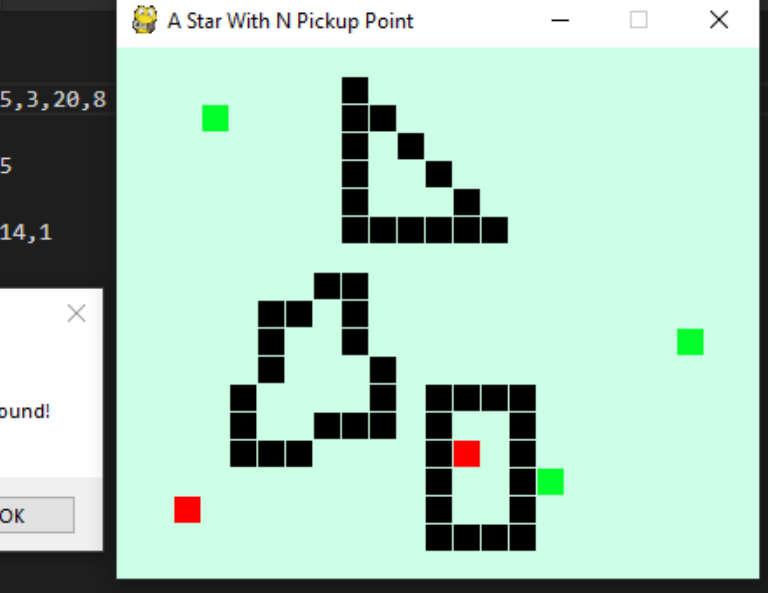
A*

Chạy tương đối nhanh và cũng tìm ra đường đi tốt nhất.

Mức 3: Áp dụng thuật toán A* khi bản đồ có thêm các điểm đón



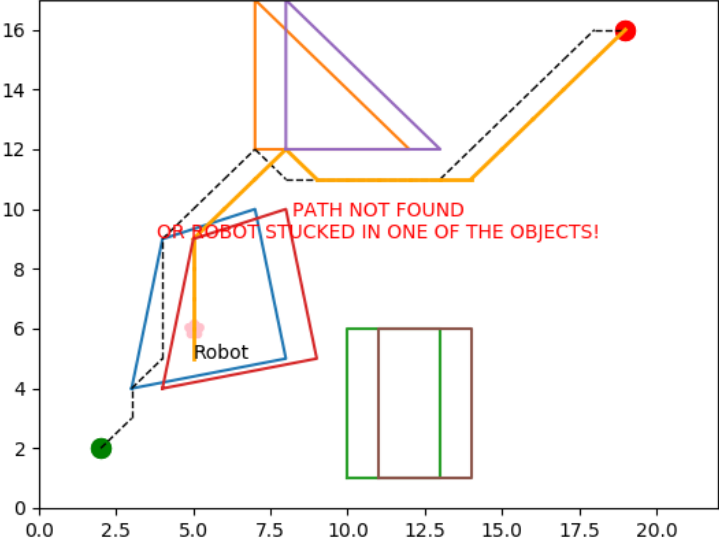
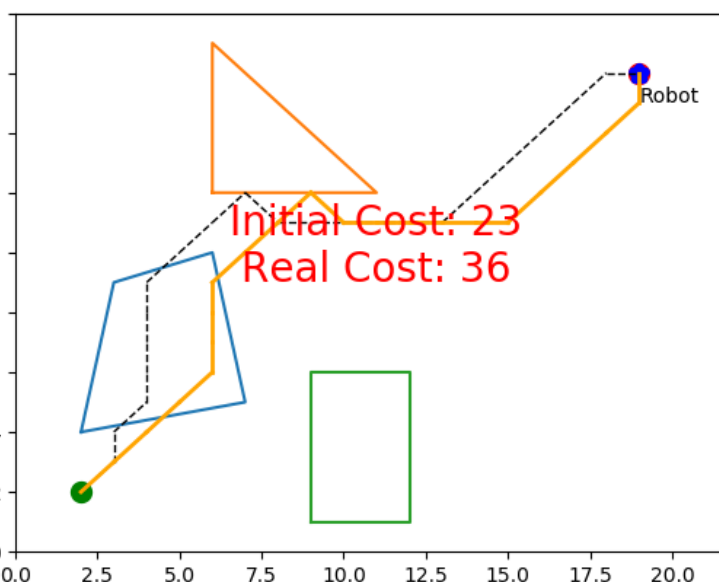
MAP 1: input5.txt

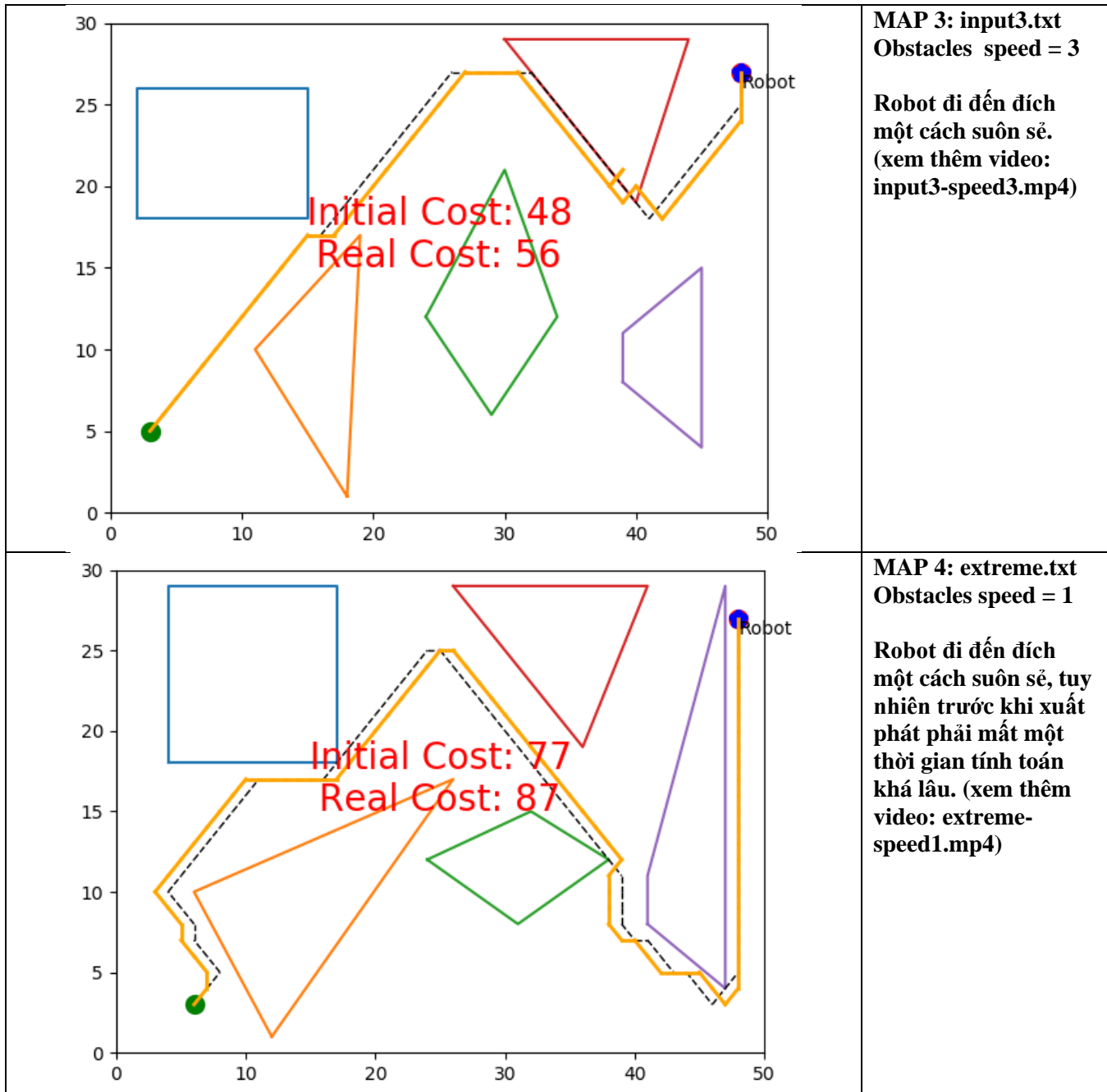
<pre> input6.txt 1 50,30 2 3,5,48,27,21,25,39,19,45,3 3 5 4 3,18,3,26,16,26,16,18 5 19,1,12,10,20,17 6 25,12,31,21,35,12,30,6 7 31,29,45,29,41,19 8 40,8,40,11,46,15,46,4 </pre>		<p>MAP 2: input6.txt</p>
<pre> input7.txt 1 22,18 2 2,2,12,4,3,16,15,3,20,8 3 3 4 4,4,5,9,8,10,9,5 5 8,12,8,17,13,12 6 11,1,11,6,14,6,14,1 </pre>		<p>MAP 3: input7.txt Không có đường đi</p>

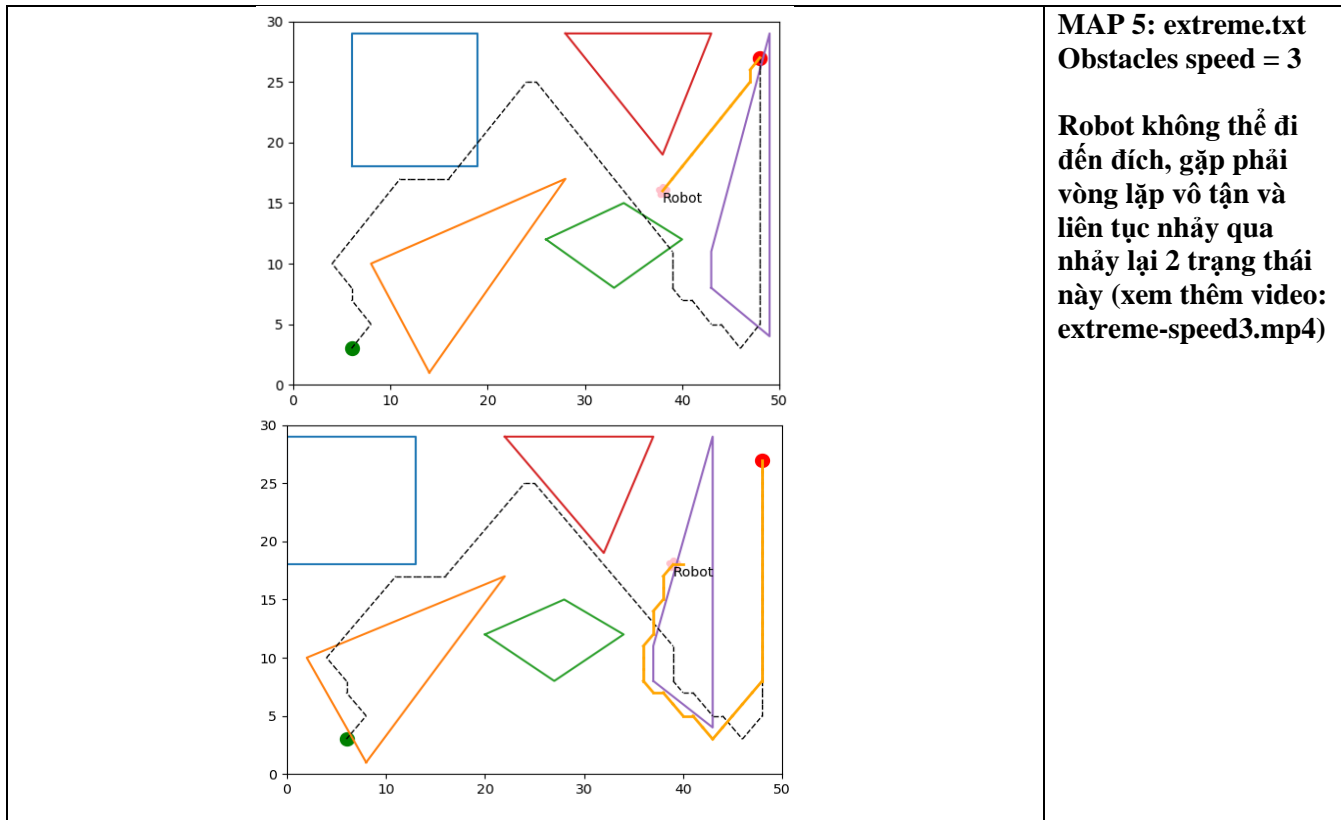
Mức 4: Các hình đa giác có thể di động được với tốc độ h tọa độ/s

(Link demo minh họa: https://www.youtube.com/playlist?list=PLT3P1FiVK_NHSFudx_eMfHA2l7pS1pare)

Để phù hợp cho việc minh họa, tốc độ được tăng lên thành h tọa độ/0.1s

	<p>MAP 1: input1.txt Obstacles speed = 1</p> <p>Robot bị mắc kẹt trong vật do không có cơ chế thay đổi tốc độ. (xem thêm video: input1-speed1.mp4)</p>
	<p>MAP 2: input1.txt Obstacles speed = 2</p> <p>Robot đi đến đích một cách suôn sẻ. Do trong trường hợp này, nó không bị “nuốt” bởi vật cản. (xem thêm video: input1-speed2.mp4)</p>





2.4

References

- Sách Cơ sở trí tuệ nhân tạo – Lê Hoài Bắc – Tô Hoài Việt
- Thuật toán BFS: <https://www.redblobgames.com/pathfinding/a-star/introduction.html#breadth-first-search>
- Thuật toán Greedy, UCS: <https://viblo.asia/p/cac-thuat-toan-co-ban-trong-ai-phan-biet-best-first-search-va-uniform-cost-search-ucs-Eb85omLWZ2G>
- “Dealing with moving obstacles”:
<http://theory.stanford.edu/~amitp/GameProgramming/MovingObstacles.html>
- Tham khảo module GUI.py và cách tổ chức thuật toán: <https://github.com/tuandoan998/astar>