BÁO CÁO ĐỒ ÁN 1

CHỦ ĐỀ: TÌM KIẾM HEURISTIC VỚI A*

MÔN HỌC
CƠ SỞ TRÍ TUỆ NHÂN TẠO

[DATE]

[COMPANY NAME]

[Company address]

Mục lục

1 Thông tin nhóm

STT	MSSV	Tên Thành Viên	Email
1	1612348	Lý Vĩnh Lợi	vinhloiit1327@gmail.com
2	1612756	Nguyễn Hữu Trường	ngoctruong9x.inc@gmail.com

2 Danh sách công việc được giao

STT	**Người thực hiện**	**Nội dung công việc**	**Mức độ hoàn thành**
1	Nguyễn Hữu Trường	Cài đặt thuật toán A*	100%
		Cài đặt thuật toán ARA	100%
		Xây dựng hàm heuristic	100%
		Chứng minh hàm heuristic là chấp nhận được	100%
		Test chương trình	100%
2	Lý Vĩnh Lợi	Thiết kế kiến trúc ứng dụng	100%
		Cài đặt giao diện người dùng	100%
		Parse tham số từ cửa sổ dòng lệnh	100%
		Xây dựng bộ test	100%
		Test chương trình	100%

3 Những vấn đề chưa thực hiện được

Không có

4 Các bộ test đã chạy

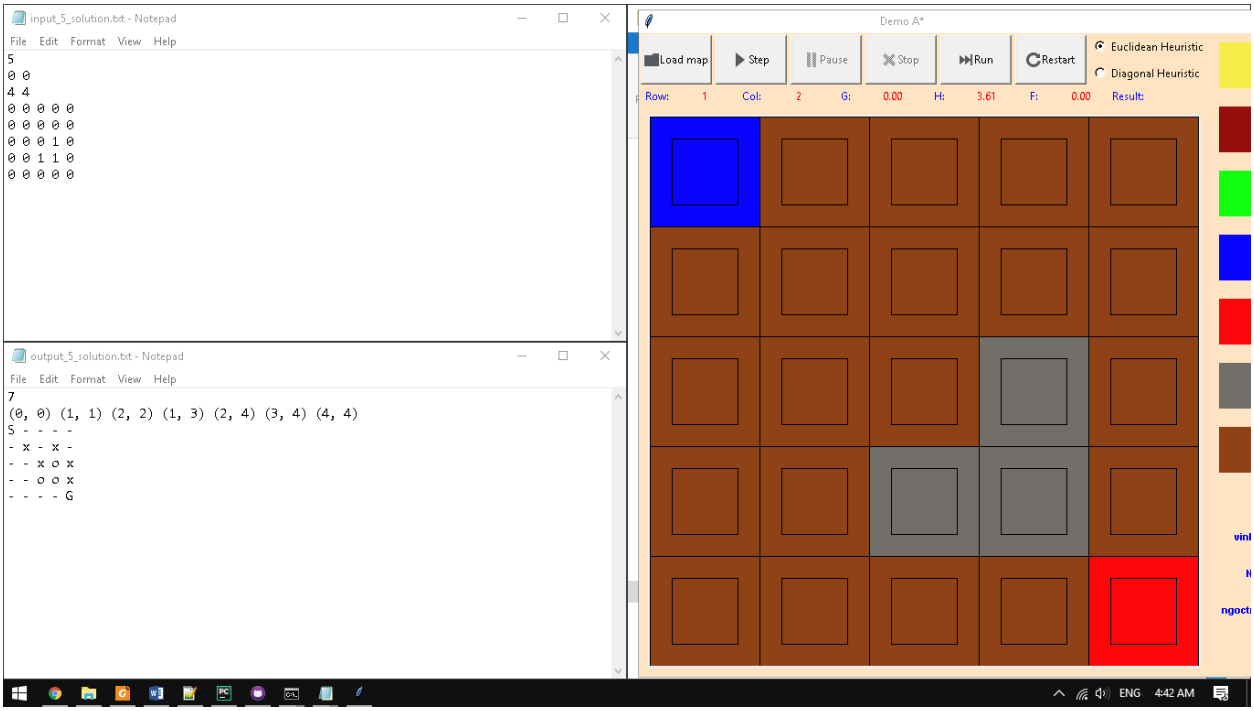

Figure 1 Bộ test chạy với kích thước 5x5, có lời giải

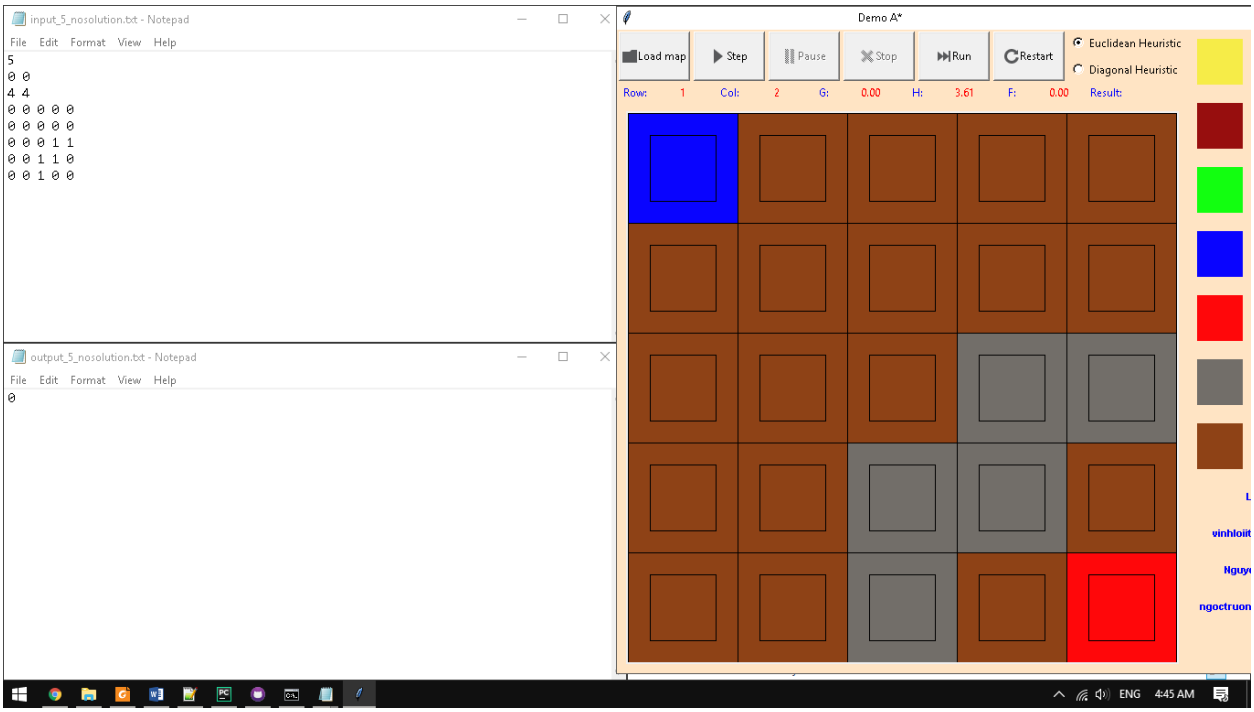

Figure 2 Bộ test chạy với kích thước 5x5, không có lời giải

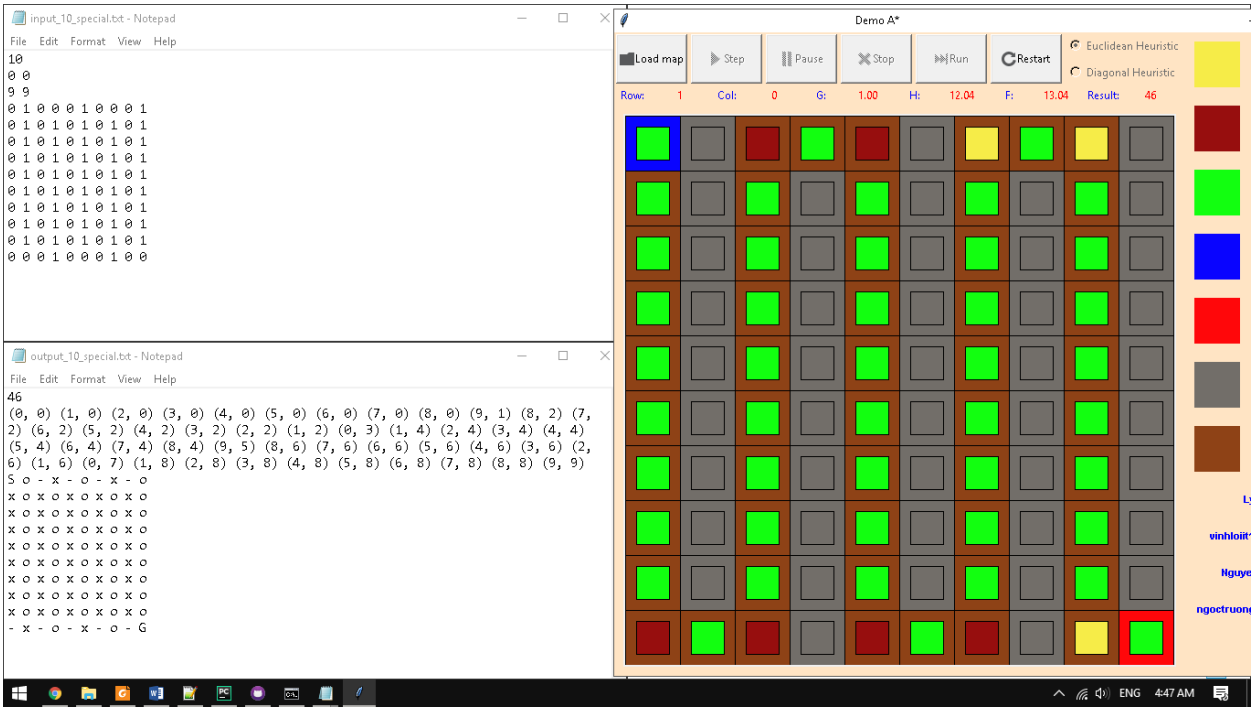

Figure 3 Bộ test chạy với kích thước 10x10, có lời giải

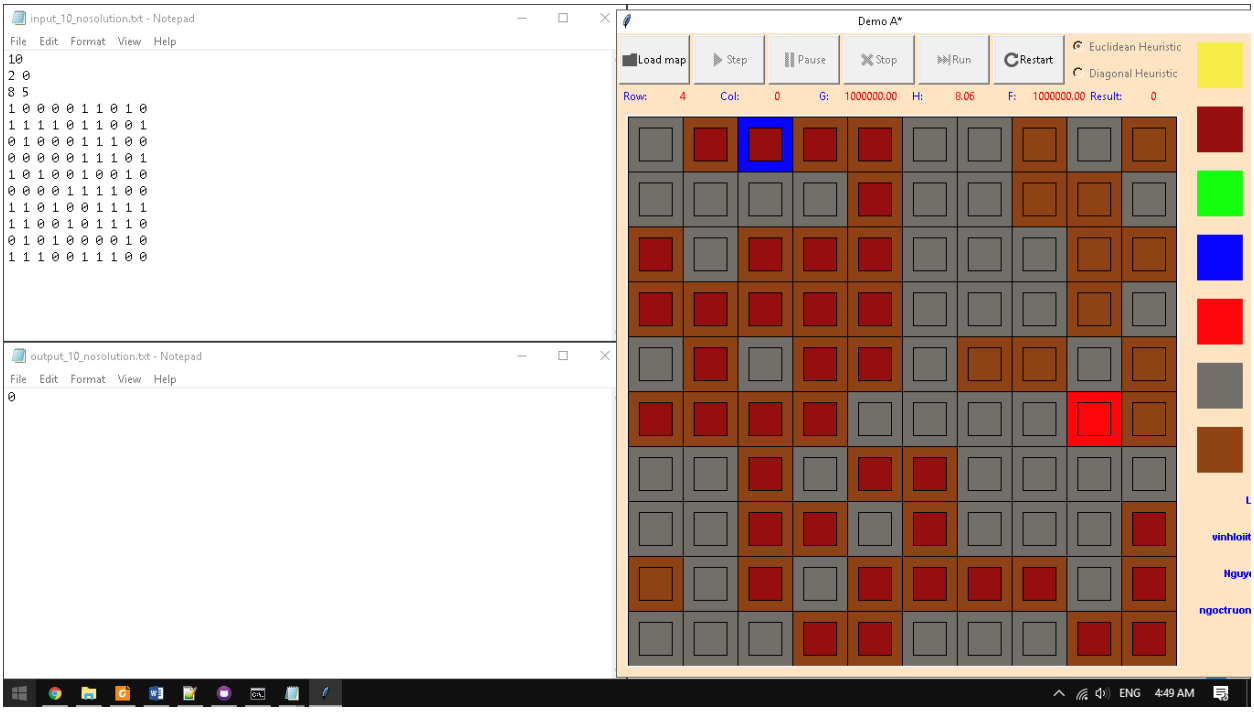

Figure 4 Bộ test chạy với kích thước 10x10, không có lời giải

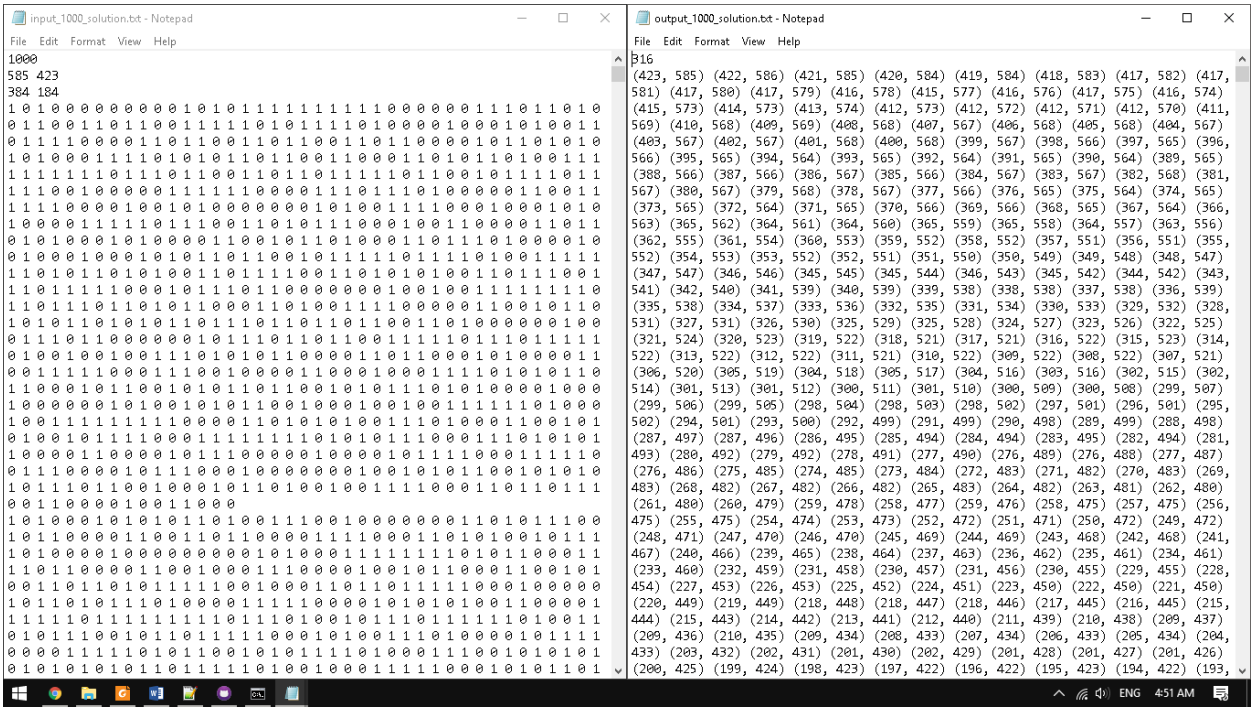

Figure 5 Bộ test chạy với kích thước 1000x1000, có lời giải

5 Hàm heuristic nhóm đề xuất và chứng minh

5.1 Hàm heuristic

Với n là một node trên bản đồ, $goal$ là node đích, ta có:

$$H(n) = \max(|n.x - goal.x|, |n.y - goal.y|)$$

5.2 Chứng minh

Gọi:

- $H^*(n)$ là chi phí tối ưu từ n đến $goal$
- $H(n)$ là chiều dài đường đi ngắn nhất từ n di chuyển đến $goal$

Nhận xét:

- Từ n có thể đi *tối đa* 8 nút xung quanh

Xét 2 trường hợp:

- Từ n đến $goal$ không có bất cứ vật cản nào trên đường đi, ta có $H(n) = H^*(n)$
- Trong các trường hợp khác, tức là có vật cản trên đường đi thì $H(n) < H^*(n)$

Vậy $H(n)$ chấp nhận được với mọi n

6 Sơ đồ biểu diễn hệ thống phần mềm

6.1 Sơ đồ khối tổng quát phần mềm

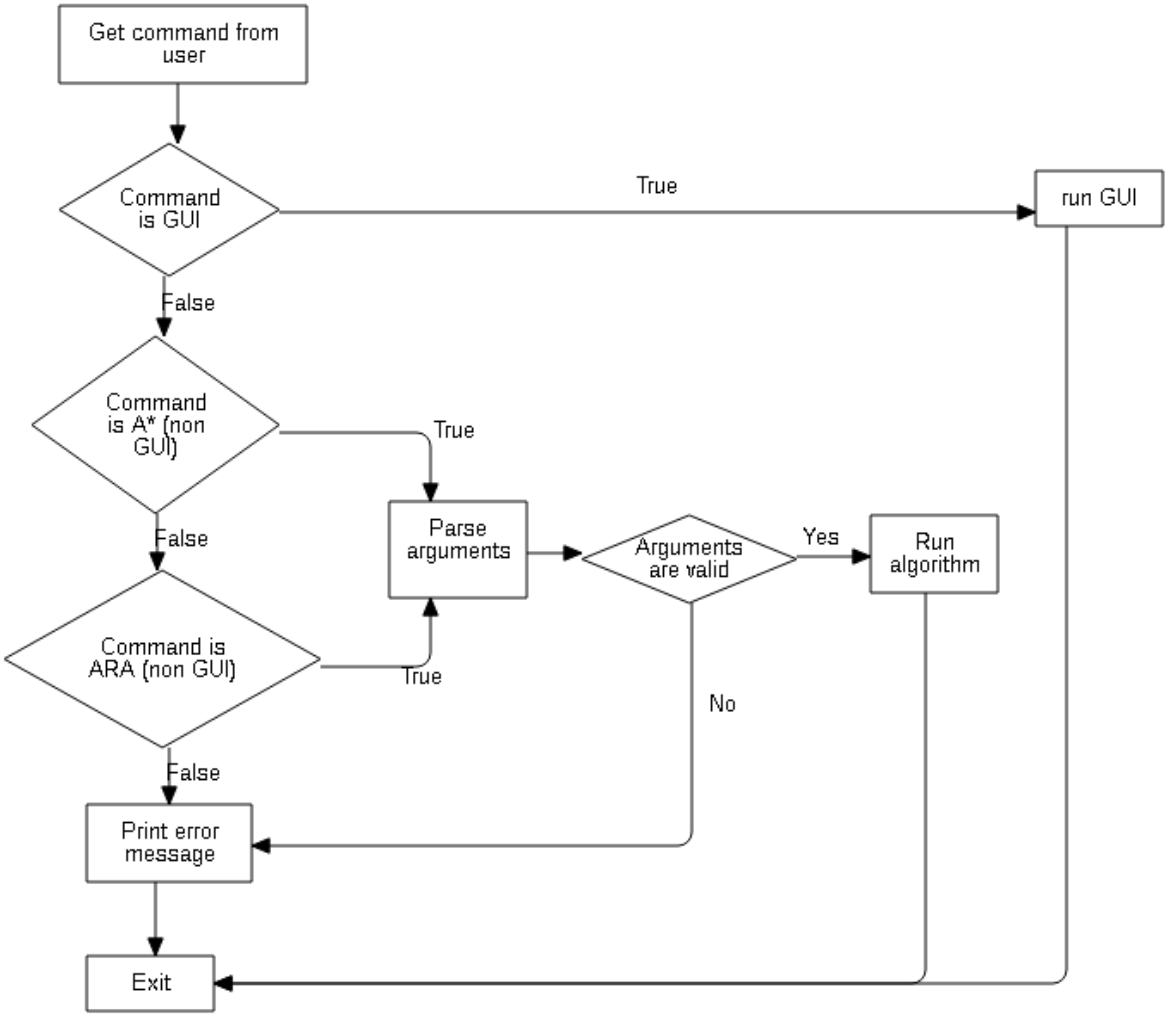

Figure 6 Trình tự thực hiện chương trình

6.2 Giao diện

Figure 7 Giao diện người dùng đồ họa của ứng dụng

7 Cấu trúc dữ liệu đã được cài đặt

Thuật toán được cài đặt theo hướng đối tượng, tất cả các thông tin về các lớp được lưu trữ và xử lý bên trong lớp đó, chỉ cung cấp các interface cho các lớp bên ngoài giao tiếp. Bên cạnh các lớp xử lý logic – là những lớp chứa các thông tin cơ bản trong thuật toán – còn có các lớp giao diện để hỗ trợ trong xử lý giao diện và hiển thị Node lên giao diện đồ họa người dùng

Các lớp nói chung khi cài đặt đều phải tuân thủ theo nguyên tắc của hướng đối tượng: “**Tell, don’t ask**”. Tuy nhiên sẽ có một số ngoại lệ khi cân nhắc về tốc độ xử lý

Ngoài ra, trong quá trình cài đặt còn sử dụng một số mẫu thiết kế và tính đa hình để giảm thời gian cài đặt và thuận tiện mở rộng sau này

7.1 Sơ đồ lớp tổng quát

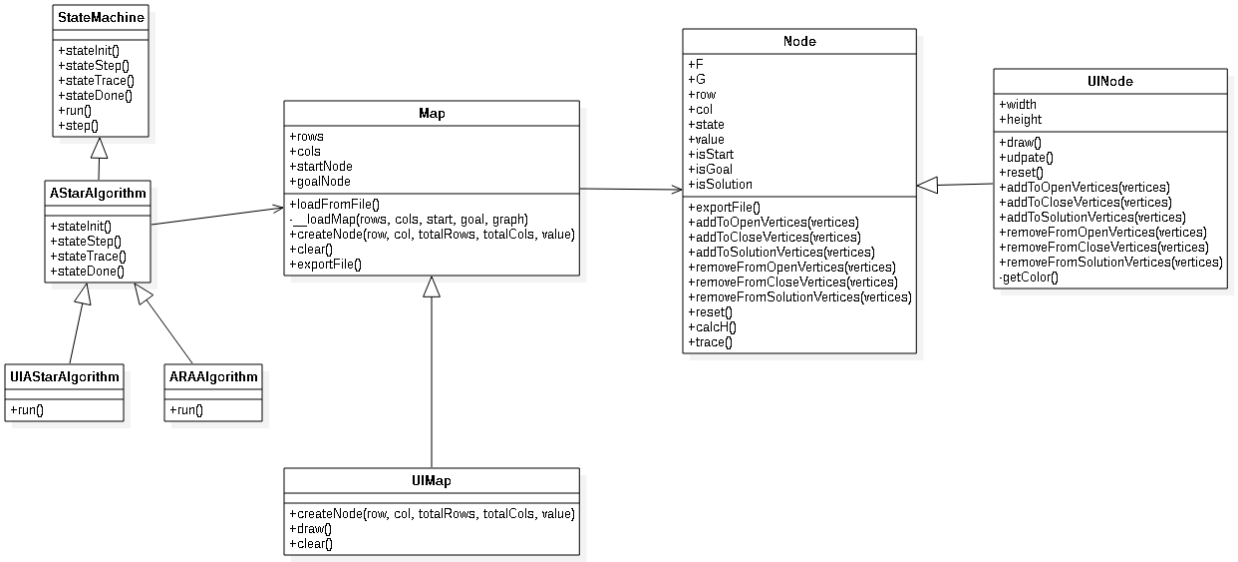

Figure 8 Sơ đồ lớp tổng quát

7.2 Class StateMachine

Đây là lớp cơ bản của mọi thuật toán (cụ thể ở đây là thuật toán A* và ARA)

Vì thuật toán có 2 tính chất quan trọng mà máy trạng thái (state machine) có thể đáp ứng được đó là:

- Tính xác định: sau khi thực hiện một thao tác, thì hoặc là thuật toán kết thúc, hoặc là có đúng một thao tác xác định để thực hiện tiếp theo [1]
- Tính hữu hạn (tính dừng): thuật toán phải kết thúc sau một số hữu hạn lần thực hiện các thao tác [1]

A*, ARA là các thuật toán, do đó ta có thể cài đặt thuật toán như là một máy trạng thái

7.3 Class AStarAlgorithm

A* là một thuật toán được cài đặt như một máy trạng thái, do đó lớp này kế thừa từ lớp StateMachine. Bên cạnh đó, nó override lại các phương thức (tương ứng với các trạng thái) của class StateMachine. Mỗi phương thức thực hiện các thao tác của thuật toán được trình bày trong phần 8.1

7.4 Class ARAAlgorithm

ARA là một thuật toán biến thể từ A*. Về mặt cài đặt tương tự như A* nên ARAAlgorithm kế thừa từ A*, nó override lại các phương thức nhất định

7.5 Class UIAStarAlgorithm

Lớp UIAStarAlgorithm là lớp giao diện của lớp AStarAlgorithm. Nó override lại các phương thức để vừa cập nhật giá trị logic, vừa cập nhật giao diện thông qua đa hình.

7.6 Class Node

Lớp Node là lớp cơ bản nhất, nó chứa tất cả thông tin về một Node, bao gồm:

- Vị trí: tọa độ dòng, cột
- Các giá trị F, G được tính toán và lưu lại
- Trạng thái: đóng, mở, hay không đóng không mở

- Các cờ (flag) xác định node là node bắt đầu hay node kết thúc
- Giá trị: là 0 nếu là một đỉnh có thể đi qua, hoặc 1 nếu là một đỉnh có chướng ngại vật

7.7 Class UINode

Để hỗ trợ việc hiển thị Node lên màn hình ta cần một lớp có khả năng *biết* được các thay đổi của Node và tương ứng mỗi thay đổi, ta sẽ cập nhật lên giao diện tương ứng. Do đó class UINode kế thừa từ class Node và override lại các phương thức thêm, xóa khỏi các tập đóng, tập mở, tập kết quả. Khi override, các phương thức này gọi lại phương thức cha – tức cập nhật về mặt logic – đồng thời cập nhật trạng thái trên giao diện

7.8 Class Map

Lớp Map là lớp đại diện cho một bản đồ, nó chứa các thông tin về bản đồ như:

- Kích thước: Số lượng dòng cột
- Một danh sách các Node
- Node bắt đầu và node kết thúc. Ở đây về nguyên tắc hướng đối tượng thì không nên lưu lại 2 giá trị này (vì bản thân mỗi node đã có thông tin đó). Tuy nhiên do số lần truy xuất đến 2 node này nhiều mà mỗi lần truy xuất phải duyệt toàn bộ map khiến tốc độ bị giảm đi không đáng có

7.9 Class UIMap

Tương tự như UINode, UIMap là một class thể hiện về mặt giao diện cho class Map

8 Mô tả thuật toán chính đã được thực hiện và những thay đổi

8.1 Thuật toán A*

8.1.1 Mã giả

h(node) is a heuristic function

AstarAlgorithm

```
1 Put node_start in the OPEN list with F(node_start) = h(node_start)
2 while OpenSet not empty
3         current_node = node( F_node min in OpenSet)
4         if current_node is node_goal
                  we have found the solution; break
5         add current_node to CloseSet
6         remove current_node from OpenSet
7                 for each neighb_node of current_node
8                         if neighb_node not in CloseSet and isn't obstacle
9                                 add neighb_node to OpenSet
10                                update F and G(neighb_node)
```

8.1.2 Giải thích

Mục tiêu của hàm AStarAlgorithm là tìm ra đường đi tối ưu từ start đến goal (tối ưu theo hàm heuristic h)

- Dòng 1: Thêm node_start vào tập Open đồng thời tính giá trị F của nó
- Các dòng tiếp theo được đặt trong vòng lặp while để tìm ra đường đi ngắn nhất
- Dòng 3: tìm node có F min trong tập Open rồi đặt nó là current_node
- Dòng 4: Nếu current_node là goal, tức là đã tìm đến điểm đích, kết thúc thuật toán

- Dòng 5: cập nhật tập Open và Close
- Các dòng sau đó: Lặp tất cả các node "hàng xóm" của current_node để tìm xem node nào khả thi để đi tiếp

8.2 Ý tưởng cải tiến:

Ta nhận thấy rằng, với mỗi vòng lặp while đều có tìm node có Fmax trong OpenSet. Chúng ta có thể cải tiến bằng cách lưu OpenSet dưới dạng heap

Node_F_min_in_OpenSet=OpenSet[0]

=>Giảm đáng kể thời gian tìm kiếm

9 Thuật toán cải tiến

9.1 Đặt vấn đề

Với hàm heuristic $H(n)$ "chấp nhận được" khiến cho quá trình tìm kiếm trở nên lâu hơn do phải mở 1 số lượng lớn node. Với thời gian bị giới hạn, thì thuật toán A* khó có thể đưa ra được 1 lời giải trong khoảng thời gian đó. Do vậy bằng cách "nới lỏng" hàm $H(n)$, tức là $H(n) \geq H^*(n)$, chúng ta có thể tìm ra đường đi 1 cách nhanh nhất, tất nhiên nó không đảm bảo là tối ưu.

Đề xuất: $F(n) = G(n) + \varepsilon * H(n)$ $với$ $\varepsilon > 1$

9.2 Mã giả

InconSet lưu trữ những node thuộc CloseSet mà có G thay đổi nhằm mục đích sẽ xét lại trong những lần sau chạy A*

```
1 AstarAlgorithm
2           current_node=node which has maxF in OpenSet
3 while F[goal]>F[current_node]
4           add current_node to CloseSet
5           remove current_node from OpenSet
6                       for each neighb_node of current_node
7                          if neighb_node isn't obstacle and G[neighb_node]>G[current_node]+ 1
8                                       update F and G(neighb_node)
9                                       if neighb_node not in CloseSet
10                                                  if neighb_node not in OpenSet
11                                                              add current_node to OpenSet
12                          else add neighb_node to InconSet
13      current_node=node which has maxF in OpenSet
```

Figure 9 Mã giả thuật toán A cải tiến*

```
ARA suppose e=2, e is weighted heuristic
1 add start_node to  OpenSet with F
2           AstarAlgorithm(e)
3 while e>1
4           e=e-0.05
5           OpenSet=OpenSet+InconSet
6                       clear InconSet
7                          update F value of node in OpenSet with new e
                              G[neighb_node]>G[current_node]+ 1
8                                   update F and G(neighb_node)
9                                   clear CloseSet
10                      AstarAlgorithm(e)
```

Figure 10 Mã giả thuật toán ARA

9.2.1 Giải thích code:

- Thay vì điều kiện lặp của vòng lặp while số 3 (phần trước) là OpenSet not empty thì ở phần này chúng ta so sánh giá trị F. Điều kiện này tương đương với đã duyệt hết mọi node trong OpenSet (đối với trường hợp không có đường đi) và đã tìm đến đích (trong trường hợp có lời giải).
- Hàm ARA thực chất là gọi lại AstarAlgorithm(e) nhiều lần với e giảm dần nhằm tìm kiếm kết quả tốt hơn (nếu có).
- Với mỗi lần gọi AstarAlgorithm(e), ta cần thực hiện:
  - Gộp InconSet vào OpenSet. InconSet ở đây là lưu những node nằm trong CloseSet mà có giá trị G thay đổi (Dòng 8 ARA)
  - Clear InconSet
  - Update giá trị F của node trong OpenSet với e mới
  - Clear CloseSet

9.2.2 *e* có giá trị bao nhiêu là hợp lí?

Với bài toán này, giữa chọn $e = 2$ hoặc $e = 10$ (hoặc nhiều hơn) không khác nhau nhiều, vì chi phí để di chuyển giữa 2 node gần nhau đều bằng 1, việc $e = 2$ nó đã làm cho F lớn hơn rất nhiều. Sự khác biệt rõ ràng nhất là khi e tiến về 1, ví dụ: 1.25, 1.05, ...
Thậm chí với $e < 1$ thì đường đi tìm được còn tốt hơn nhiều so với $e = 1$. Theo ý kiến chủ quan, thứ nhất là do chi phí giữa các node kề nhau đều bằng nhau, thứ hai là do e nhỏ sẽ làm cho hàm heuristic bị "thắt chặt" làm cho kết quả tốt hơn rất nhiều.
Mặc dù vậy ở đây nhóm vẫn chọn chọn $e = 2$ với $\Delta e = 0.05$ và $e > 1$ để đảm bảo logic thuật toán.

⇨ Với những sự thay đổi trên so với thuật toán A*, ARA tỏ ra cực kì hiệu quả khi thời gian bị giới hạn.
Chú ý: Ta vẫn có thể cải tiến để AstarAlgorithm(e) chạy nhanh hơn đó là sử dụng cấu trúc heap để lưu OpenSet

10 Tài liệu tham khảo

[1] "Bài toán và thuật toán," [Online]. Available: https://sites.google.com/site/nthanhbanvl/tin-hoc-lop-10/chuong-i/bai-4-bai-toan-va-thuat-toan. [Accessed 28 Obtober 2018].

[2] G. G. S. T. Maxim Likhachev, "ARA*: Anytime A* with Provable Bounds on Sub-Optimality," Advances in Neural Information Processing Systems 16 (NIPS), MIT Press, Cambridge, 2004.

[3] "Using PyInstaller," 28 October 2018. [Online]. Available: https://pyinstaller.readthedocs.io/en/v3.4/usage.html.

[4] "Tkinterbook," 28 October 2018. [Online]. Available: http://effbot.org/tkinterbook/.