Câu 3

1. Mô tả bài toán:

Cho môi trường A được viết dưới dạng ma trận MxN, điểm bắt đầu A(x\_0,y\_0) và điểm kết thúc B(x\_n,y\_n). Tìm đường đi ngắn nhất từ A đến B.

Mô tả trạng thái: Dictionary có các phần tử chứa:

* Index: tọa độ ô sau khi di chuyển
* Value: tọa độ trước di chuyển

Trạng thái khởi đầu: [((x\_0,y\_0),None)]

Trạng thái kết thúc: [((x\_n,y\_n),(x\_n-1,y\_n-1)),…,((x\_0,y\_0),None)]]

Ma trận A có 2 loại phần tử, TILE\_EMPTY(0) và TILE\_WALL(inf)

Luật di chuyển: từ ô (x,y), có thể di chuyển về các ô:

|  |  |  |
| --- | --- | --- |
| x-1,y-1 | x-1,y | x-1,y+1 |
| x,y-1 | x,y | x,y+1 |
| x+1,y-1 | x+1,y | x+1,y+1 |

Điều kiện di chuyển là ô nằm trong ma trận và là có giá trị TILE\_EMPTY. Khi di chuyển cập nhập trạng thái bằng cách thêm (tọa độ sau di chuyển,tọa độ trước di chuyển) vào dictionary.

Hàm giá g(x): chi phi di chuyển từ ô bắt đầu đến ô hiện tại tính bằng số ô đã đi qua

Hàm mục tiêu h(x): lấy khoảng cách (đường chim bay) từ đích đến điểm hiện tại, tức

1. Hiện thực bài toán:

* Class Maze: chứa, sinh môi trường và gọi danh sách láng giềng
* Class App: sử dụng thư viện pygame ngoài trực quan hóa bài toán còn chứa hàm A\_star thực hiện giải thuật

1. Quá trình giải:
   1. Sinh môi trường
   2. Chọn điểm bắt đầu, điểm kết thúc
   3. Tạo hàng đợi ưu tiên, đưa điểm bắt đầu vào với chỉ số ưu tiên là 0 (cao nhất)
   4. Tạo thư viện lưu vết, đưa vào (điểm bắt đầu,None)
   5. Tạo thư viện lưu giá g(x), đưa vào (điểm bắt đầu,0)
   6. Nếu hàng đợi còn phần tử:

* Lấy phần tử ra làm phần tử hiện tại
* Nếu phần tử hiện là đích, truy vết ngược lại về điểm bắt đầu qua thư viện, trả về danh sách, kết thúc thuật toán
* Lấy mỗi láng giềng của ô hiện tại, nếu chưa xuất hiện trong hàng đợi:
  + Tính và lưu g(x) của láng giềng vào thư viện giá
  + Tính và lưu độ ưu tiên h(x) + g(x) của láng giềng vào hàng đợi
  + Lưu vết (láng giềng, điểm hiện tại)
  1. Nếu đã hết hàng đợi mà vẫn không đến được đích thức không có đường đi, trả về danh sách rỗng