Artificial Intelligence: `Lab Assignment-4`

Group Members-
Abhishek Katara
Ujwal tewari
Nikhil Thota
Group-7

February 1, 2018

1 Question-1

Lemma Any Shortest Path between start and goal point among a set S of disjoint polynomial obstacle is a polygonal path whose inner vertices are vertices of set S.

Create a Visiblity Graph
(1) Input a set of vertices whose edges don't intersect.
(2) Join the all vertices ng to ni.
(3) Check if Line Intersects of not.

1.1 Pseudo Code-

```
Q = Priority Queue (consists of all nodes)
V = List of visited nodes
E(n1,n2) = Edge between n1 and n2
h(n) = Heuristic length of node from root node.
g(n) cost function of path.
f(n) = Total path function

astar {
While( Q is not Empty)
pick best node N from Q for f(N_best) <= f(n) for all N belongs to Q
Transfer N_best to Visited Nodes after reaching
if(N_best == GOAL)
 EXIT

Choose N_best from Q which are not visited

if(N_x not belongs to Q)
 add N_x to Q
```

```
else if (g(N_best) + c(N_best, N_x) < g(N_x))
    Update the Graph
}
```

1.2 Code and Implementation

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
class Node:
    def __init__(self,value,point):
        self.value = value
        self.point = point
        self.parent = None
        self.H = 0
        self.G = 0
    def move_cost(self,other):
        return 0 if self.value == '.' else 1

def children(point,grid):
    x,y = point.point
    links = [grid[d[0]][d[1]] for d in [(x-1, y),(x,y - 1),(x,y + 1),(x+1,y)]]
    return [link for link in links if link.value != '%']
def manhattan(point,point2):
    return abs(point.point[0] - point2.point[0]) + abs(point.point[1]-point2.po
def aStar(start, goal, grid):
    #The open and closed sets
    openset = set()
    closedset = set()
    #Current point is the starting point
    current = start
    #Add the starting point to the open set
    openset.add(current)
    #While the open set is not empty
    while openset:
        #Find the item in the open set with the lowest G + H score
        current = min(openset, key=lambda o:o.G + o.H)
        #If it is the item we want, retrace the path and return it
        if current == goal:
            path = []
            while current.parent:
                path.append(current)
                current = current.parent
            path.append(current)
            return path[::-1]
        #Remove the item from the open set
        openset.remove(current)
        #Add it to the closed set
        closedset.add(current)
        #Loop through the node's children/siblings
        for node in children(current,grid):
            #If it is already in the closed set, skip it
```

```
            if node in closedset:
                continue
            #Otherwise if it is already in the open set
            if node in openset:
                #Check if we beat the G score
                new_g = current.G + current.move_cost(node)
                if node.G > new_g:
                    #If so, update the node to have a new parent
                    node.G = new_g
                    node.parent = current
            else:
                #If it isn't in the open set, calculate the G and H score for t
                node.G = current.G + current.move_cost(node)
                node.H = manhattan(node, goal)
                #Set the parent to our current item
                node.parent = current
                #Add it to the set
                openset.add(node)
    #Throw an exception if there is no path
    raise ValueError('No Path Found')
def next_move(pacman,food,grid):
    #Convert all the points to instances of Node
    for x in xrange(len(grid)):
        for y in xrange(len(grid[x])):
            grid[x][y] = Node(grid[x][y],(x,y))
    #Get the path
    path = aStar(grid[pacman[0]][pacman[1]],grid[food[0]][food[1]],grid)
    #Output the path
    print len(path) - 1
    for node in path:
        x, y = node.point
        print x, y
pacman_x, pacman_y = [ int(i) for i in raw_input().strip().split() ]
food_x, food_y = [ int(i) for i in raw_input().strip().split() ]
x,y = [ int(i) for i in raw_input().strip().split() ]

grid = []
for i in xrange(0, x):
    grid.append(list(raw_input().strip()))

next_move((pacman_x, pacman_y),(food_x, food_y), grid)
```

1.3 Output

(359).png

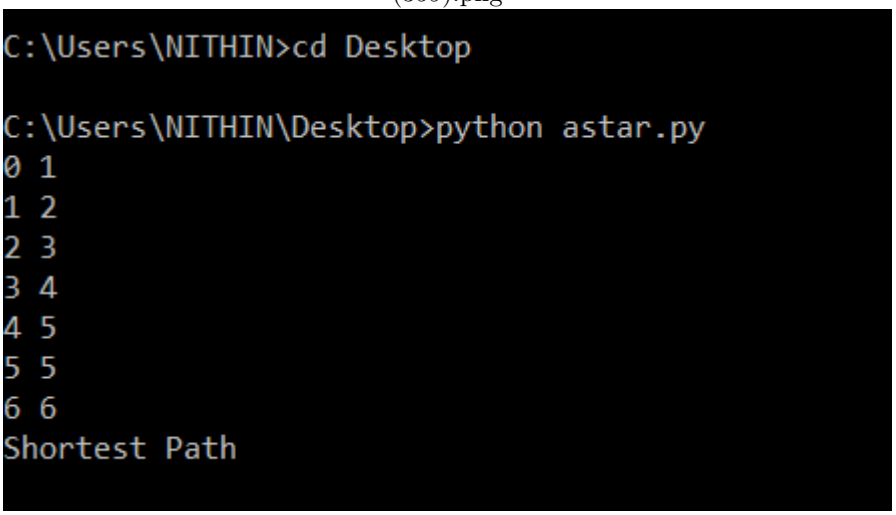

Figure 1: Output of Astar Search