Assignment One

SOFE 3720/ CSCI 4610: Introduction to Artificial Intelligence/ Artificial Intelligence

Winter 2019

Dr. Sukhwant Kaur Sagar

Submission Deadline: Friday Feb. 15, 2019 11:59 PM

Name	Student Number
Yin Zhou	100314426
Kavinaa Uthayachandran	100566729
Mohtasim Siddiqui	100635463

Introduction

This assignment is to demonstrate an understanding of finding the shortest path using A* search algorithm. As outlined in the assignment, we have selected an area in Oshawa wherein which the algorithm has been applied to a set of nodes, determining the optimal route for walking between the two nodes.

A* search algorithm utilizes the function **f(n) = g(n) + h(n)**, where

- **g(n)** is the path cost from the start node
- **h(n)** is the heuristic estimating the cost of the cheapest path from the start node
- **n** is the previous node

When determining which path to take, the value of **f** should be the lowest amongst all of the nodes that are in consideration. This is repeatedly done through every traverse, with respect to all nodes in the range, when determining an optimal path.

Cost and heuristic function

In terms of determining the path with the lowest cost, all possible nodes around the source node would be considered in which the node that has the lowest value in distance would be included in the path. Frontier of the tree needs to be expanded in the direction of the goal node.

Therefore, the heuristic function will be the path directly from the starting node to the goal node. In other words, heuristic function tells us how close we are to the goal node and can be implemented as follows. Instead of expanding the frontier of all the adjacent nodes, priority is given to the nodes with the closest distance to the goal node and this makes the search more efficient.

$$h(x,y) = \sqrt{(x2-x1)^2+(y2-y1)^2}$$

```
def hur(self, node, dest):
        #closest distance between two node is the straight line distance between two node.
return node_dist(node,dest)
```

```
def node_dist(n1, n2):
   # to find distance between two point
   dx = (n2.pos[0] - n1.pos[0]) * MPERLON
   dy = (n2.pos[1] - n1.pos[1]) * MPERLAT
return math.sqrt(dx * dx + dy * dy)
```

Path Cost: g(nlogn)

The Path cost function , *g(nlogn)* assigns cost to each of the adjacent nodes based on three variables. Suppose that *n* is set of values *x,y,z* representing *latitude, longitude* and *altitude* respectively. Then the path cost function takes in successor node based on the heuristic search and assigns weight to each node based on these variables. It starts by calculating distance between starting node and goal node based on latitude and longitude then it multiplies this by a factor of altitude. Every point on the map has an altitude value and the cost function simply subtracts these values. If the altitude is positive i.e increase in height then it multiplies the cost by a factor of 2. On the other hand, if altitude is decreasing it multiplies the cost value by a factor of 1.5. if altitude is the same then it multiplies cost by a factor of 1. The above equation can be summarized as follows.

$$g(x,y,z) = h(n') + (x' + y')z',\ z' = 2 : z > 0,\ z' = 1.5 : z < 0,\ z' = 1 : z = 0$$

This implementation is also known as *A** where a shortest path is found if the heuristic used and the cost to reach the destination node is never overestimated. In other words, the total sum is the assigned costs to the successor node and the current node is then incremented to the next node based on this cost value.

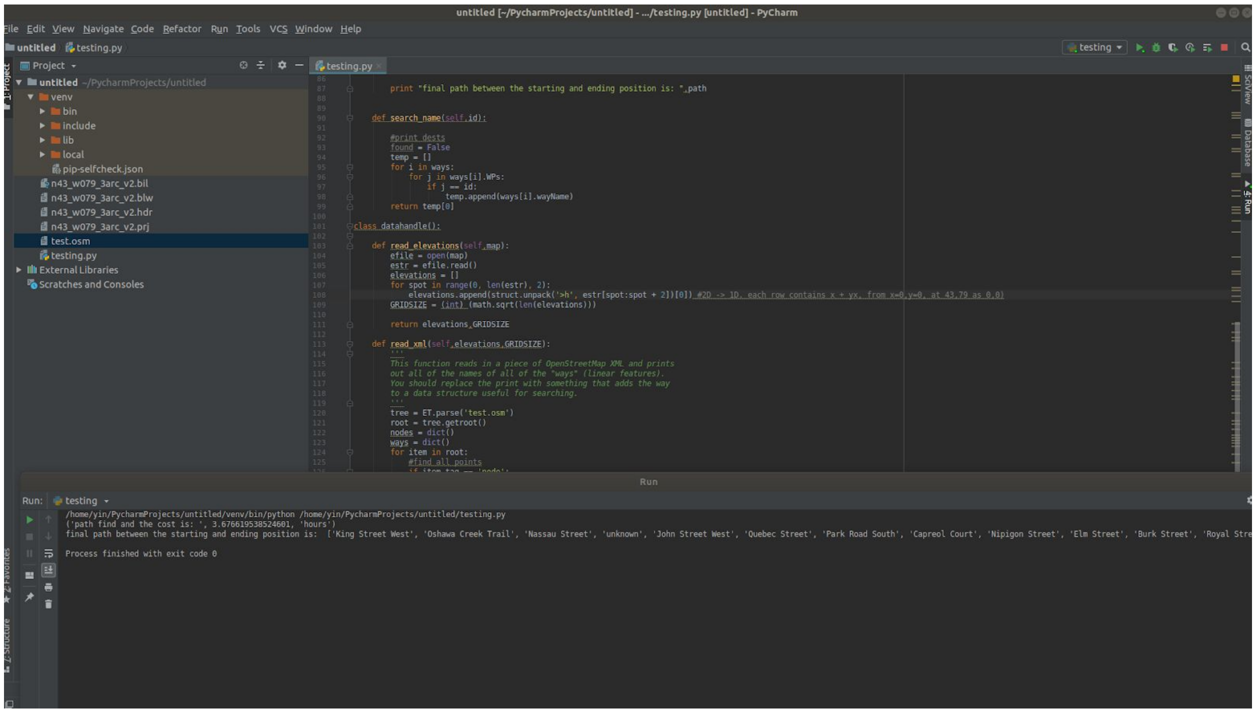
Implementation of this can be seen as follows

```
class Cost():
   def __init__(self,begin,end):
      self.begin = begin
      self.end = end
```

```
        self.cost = node_dist(begin,end)
        #uphill to downhill
        if (begin.elev - end.elev) > 1:
            self.cost *= 1.5
        elif (begin.elev - end.elev) < 1:
            self.cost *= 2
        else:
self.cost
```

```
class pathFinding():

    def __init__(self,nodes,ways):
        self.nodes = nodes
        self.path = ways

    def heur(self,start, fixend):
        #cloeset distacen between two node is the stright line dintance between two node.
        return node_dist(start,fixend)

    def astar(self, starts, dests):
        open = {}
        cost = {}
        frontier = PriorityQueue()
        init_cost = self.heur(nodes[starts], nodes[dests])
        frontier.put(starts,init_cost)
        open[starts] = None
        cost[starts] = 0

        while not frontier.empty():
            current = frontier.get()

            if current == dests:
                print ("path find and the cost is: ", cost[dests]*60/100000, "hours")
                return self.display_path (open,dests)

            for way in nodes[current].ways:
                cost_update = cost[current] + way.cost

                #check if it is in the open list
                #check if new cost is better
                if way.end.id not in open or cost_update < cost[way.end.id]:
                    nway_id = way.end.id
                    cost[nway_id] = cost_update
                    f = cost_update + self.heur(nodes[nway_id], nodes[dests])
                    frontier.put(nway_id,f)
                    open[way.end.id] = (current,way.wayz)

          # for i in current.ways

    def display_path(self,open,dests):
        path = []
        for i in open:
            name = self.search_name(i)

            if name not in path:
                path.append(name)

        print "final path between the starting and ending position is: ",path


    def search_name(self,id):

        #print dests
        found = False
        temp = []
        for i in ways:
            for j in ways[i].WPs:
                if j == id:
                    temp.append(ways[i].wayName)
return temp[0]
```

Outputs



```python
        print "final path between the starting and ending position is: ",path


    def search_name(self,id):

        #print dests
        found = False
        temp = []
        for i in ways:
            for j in ways[i].WPs:
                if j == id:
                    temp.append(ways[i].wayName)
        return temp[0]

class datahandle():

    def read_elevations(self,map):
        efile = open(map)
        estr = efile.read()
        elevations = []
        for spot in range(0, len(estr), 2):
            elevations.append(struct.unpack('>h', estr[spot:spot + 2])[0]) #2D -> 1D, each row contains x + yx, from x=0,y=0, at 43,79 as 0,0)
        GRIDSIZE = (int) (math.sqrt(len(elevations)))

        return elevations,GRIDSIZE

    def read_xml(self,elevations,GRIDSIZE):
        '''
        This function reads in a piece of OpenStreetMap XML and prints
        out all of the names of all of the "ways" (linear features).
        You should replace the print with something that adds the way
        to a data structure useful for searching.
        '''
        tree = ET.parse('test.osm')
        root = tree.getroot()
        nodes = dict()
        ways = dict()
        for item in root:
            #find all points
```

```
/home/yin/PycharmProjects/untitled/venv/bin/python /home/yin/PycharmProjects/untitled/testing.py
('path find and the cost is: ', 3.676619538524601, 'hours')
final path between the starting and ending position is:  ['King Street West', 'Oshawa Creek Trail', 'Nassau Street', 'unknown', 'John Street West', 'Quebec Street', 'Park Road South', 'Capreol Court', 'Nipigon Street', 'Elm Street', 'Burk Street', 'Royal Stre

Process finished with exit code 0
```

Github Link

https://github.com/m00cha7/aiFinal/

Work Distribution

Name	Tasks

Yin Zhou	**Research** **Coding Implementation,** *PathCost(), A*(), Heuristic* **Report** *Output & Results*
Kavinaa Uthayachandran	**Report** *Research* *Coding, Refactoring*
Mohtasim Siddiqui	**Research,** **Coding Implementation,** *XML Parser, Nodes(), Ways()* **Report** *Cost and Heuristic Function*