Name:Aierken shalayiding

lab1 A*

Proper interpretation of input files:

For the program 3 different input files are needed:

1.Orginal image map file as .png

2.mmp.txt which is the file that contains all elevation values for image pixel

3.path.txt which is the file contains all the point that we have to visit

My Program handles the file by reading a file line by line and strip the line ending '\n' and split the data within the line and read the data to list:

For example :

```
with open(sys.argv[2]) as eleva_f:          #read from two file
    Input_eleva_matrix = [[float(value) for value in (line.strip()).split()] for line in eleva_f]
```

all the information is stored inside of the eleva_matrix from file 2, the same is applied for file 3.

Map generation :

The case that map is been generated as Width * Height of matrix and store object point inside of each position.

```
def Makemap(terrain_matrix, row, col, ele_value_set):  # build the map
    map = []
    for i in range(row):
        tmp = []
        for j in range(col):
            obj = Point(i, j)
            obj.ele_value = ele_value_set[j][i]     #2d list swap i and j
            type_key = tuple(terrain_matrix[j,i][:-1])      #2d list
            obj.terrain_type = terraintype_set[type_key]    #get type from hash table
            tmp.append(obj)
        map.append(tmp)
    return map
```

Make map is used to set the elevation value of the object and find proper terrain type from pixel values in the image.

A* algorithm :

In the A* algorithm 3 different kinds of scores have been used.

1: Hscore, is simply a manhattan distance from point A to B, since we have elevation I consider that as 3 point manhattan distance.By testing couple different point and path I found out that this gives me best result for most of the cases which make sense that from this equation we are getting 3 dimension distance from point A to point B.

```python
def Calc_Hscore(point_1, point_2):  # calculate H score
    x0, x1 = point_1.x, point_2.x
    y0, y1 = point_1.y, point_2.y
    z0, z1 = point_1.ele_value, point_2.ele_value
    return abs(x1-x0)+abs(y1-y0)+abs(z1-z0)  #consider ele as 3d distance
```

2.Gscore, is calculated by distance/speed(terrain type) where the pixel first moves (change of x or change of y) every pixel move as x count as 10.29m, every pixel move as y count as 7.55m, and then divide the speed from distance will give us the G score. For my cost function, I was trying to consider if we go from uphill to down our speed will increase, if we go from down to up our speed should decrease. if we go from higher elevation to lower our speed will increase by the difference of ele_value multiplied by 0.1, and if we go from a lower elevation to higher we should get a negative number to reduce our speed.

```python
def Calc_Gscore(point_1,point_2):
    ele_diff = (point_1.ele_value - point_2.ele_value)*0.1
    if point_1.y == point_2.y:
        distance = 10.29
    else :
        distance = 7.55
    point_2.distance = distance
    return distance / (point_2.terrain_type + ele_diff)
```

3.Fscore, Fscore is the final score that we use to decide whether or not to pick that point, it is simply calculated by Gscore+Hscore.

A* Main:

The function requires 3 inputs which are: start point, endpoint, and the map we build. Two buffers are used to record our status which is, visited_point which records all the points that we have been visited, and ready_to_visit which is the point that we want to explore. if we find that the start point is equal to the endpoint we have to trace back to the start point and record all the paths.

```python
#check if we reach the end or not
if start_point == end_point:
    path = []
    while start_point.camefrom: #while point is camefrom other point
        path.append(start_point)
        start_point = start_point.camefrom  #go to place where point is camefrom
    path.append(start_point)
    return path
```

else we have to generate 4 neighbor values up, down, right, left, and then check if any of the points are outside of the image. if it is remove the point from the four_point list. Calculate the point G,H score and do for all the points in the four_point list. Then find the best point in four

values and explore that point. After we have the path simply loop the list and write the path into the image. Save the image and print the distance on the terminal.

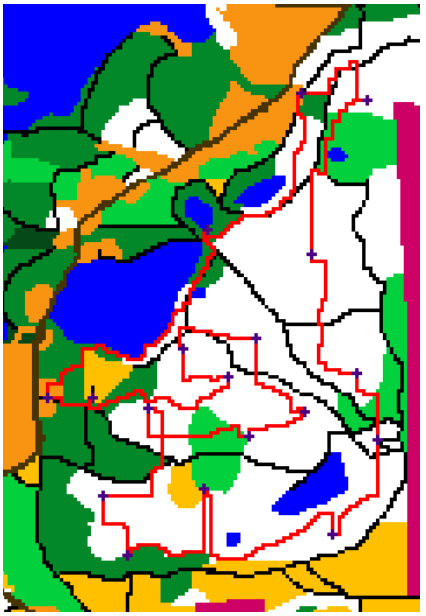
close pixel generator that generates 4 pixels value around the point we select

```python
def Closepixles(point, map, width, height):  # get four different neighbour
    four_value = []
    if point.y+1 < height:
        four_value.append(map[point.x][point.y+1])  # down

    if point.y-1 > 0:
        four_value.append(map[point.x][point.y-1])  # up

    if point.x+1 < width:
        four_value.append(map[point.x+1][point.y])  # right

    if point.x-1 > 0:
        four_value.append(map[point.x-1][point.y])  # left

    return four_value
```

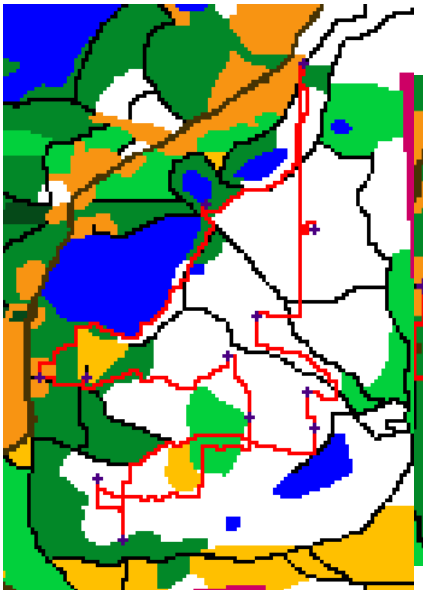
Test Case :

The Program is good on all the test cases provided, within the time requirement.

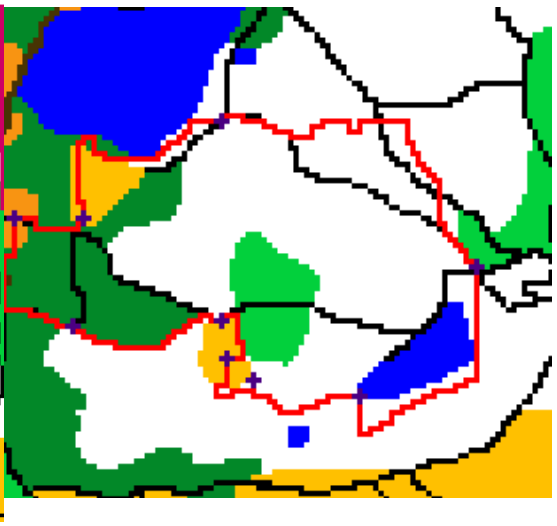
redout.png



brown.png



white.png



Other test case images are in the file, the purple color on the map is just a scaler version of our start and end points. Distance :

total distance is meter is 7568.36000000004 for red.txt

total distance is meter is 5846.1600000000235 for brown.txt

total distance is meter is: 3287.310000000005 for white.txt