ARK Task Round Documentation

Intermediate Tasks

Task 3.1 Path planning and Computer Vision Puzzle

I. Introduction

This task is related to computer vision and path finding algorithms. The task had been divided into multiple smaller tasks. To perform the various tasks, I have used libraries such as OpenCV, NumPy, PIL, argparse and SciPy. To solve the maze, I have implemented the A* algorithm for which I have used libraries such as NumPy, OpenCV, time, pandas and heapq.

II. Problem Statement

In the first part of the task, you were given an image. You had to convert the image to grayscale and then convert each of the pixel to its corresponding ascii value. Example if a particular pixel has a grayscale value of 65 then the ascii character that would be displayed would be 'A'.

This is the first image.

The next part of the task was to find the first time the colon ':' appears in the text that you had gotten from the first part of the task. Then ignoring all the pixels before the ':' you had to rearrange the remaining part of the image to a 200x150 image.

This is the image you get after rearrangement.

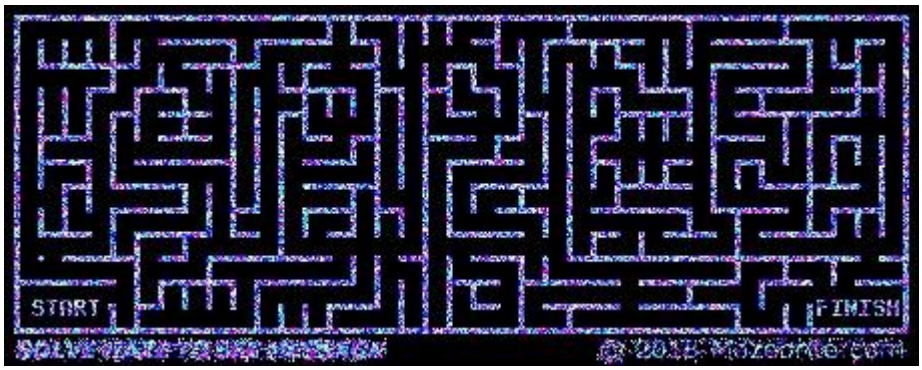
The image you get after rearranging is a part of a larger image. You had to find the x coordinate of the start point from where this rearranged image was taken from the larger image.

This is the larger image.

The x coordinate represents the colour of a monochrome maze hidden in an image with coloured noise. You had to find the maze.

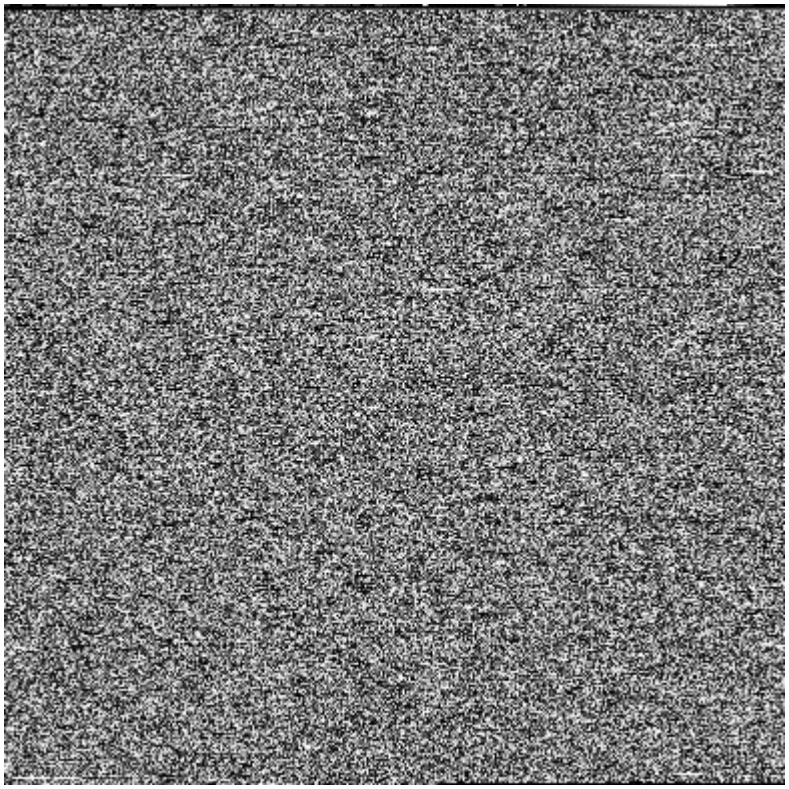
This is the image from which you had to find the monochrome image.

Then you had to solve the maze using any algorithm. Once the maze is solved you will see a word. This word is a password to a zip folder will see an image file.



This is the maze.

Lastly you had to convert the image into an audio file.



This is the image that you need to convert to an mp3 file.

III. Related Work

Most part of them problem involved image processing techniques. So, you had to convert an image to grayscale, display the pixel values of an image, crop images, use dilation, remove noise from an image and find the coordinates of a particular colour or part of an image. The other important (and difficult) part of this task was to solve the maze using any algorithm. I solved the maze using A* algorithm.

IV. Initial Attempts

In the first part of the code, I was not able to convert it to a text fille. So, I had manually started to convert the code to its ascii character. That did not help much. After which I tried to do some ascii art which again was not really helpful. Later on, the ‘chr’ function helped me to solve this part.

For the next part instead of rearranging I was just cropping the top lines thinking it really would not affect much but well it did not work out.

In the part where you had to remove the coloured noise, I had assumed that the green and red channels would be 0 and I ended up getting a black image sprinkle with a few random blue dots.

For solving the maze, the first time I got the output it was almost a straight line. The code had found way from those jigsaw pieces and those pixels and had somehow managed to arrive at the end point. After dilating the image, the output was a bit better.

For the last part of the task, I tried a bunch of things like converting it to bytes, characters but all I ended up getting was static noise and some random message.

V Final Approach

For the first part of the task, I have converted the image to a grayscale image. Then I have accessed the pixel values using img.getdata() and finally used chr() to convert the numbers to their respective ASCII codes.

```
from PIL import Image


img = Image.open('C:/Research Group/Level1.png').convert('L')
WIDTH, HEIGHT = img.size
print(WIDTH,HEIGHT)
data = list(img.getdata())
data = ([data[offset:offset+WIDTH] for offset in range(0, WIDTH*HEIGHT,
WIDTH)])


for i in range (WIDTH):
    for j in range (HEIGHT):
        print(chr(data[i][j]),end ="")
```

Then I have stored the message (that is the part before the colon) in a string variable and counted the total number of characters present in that string (which came out to be 1155).

```
string="Congrats on solving the first level of this task! You were able to
figure out the ASCII text from the image, but wait! Your journey is not yet
over. After the end of the text you will find a (200, 150, 3) coloured image.
This image is a part of the bigger image called 'zucky_elon.png'. Find the top
left coordinate (Image convention) from where this image was taken. The x
coordinate represents the colour of a monochrome maze hidden in an image with
coloured noise. Find the maze and solve the maze using any algothrim like dfs
but better. Try comparing them and seeing how they perform like A*, RRT, RRT*
for example. Once the maze is solved you will see a word. This word is a
password to a password protected zip file which contains a png. Note that the
password is case sensitive and all the aplhabets in the password will be
capital letters This is your treasure. To open the treasure you need to
convert the image in to an audio file in a simple way like you did for this
ASCII text. Once converted, open the .mp3 file and enjoy your treasure, you
deserved it! A part of the image 'zucky_elon.png' will begin immediately after
the colon,image-IV2:"

count = 0;

#Counts each character

for i in range(0, len(string)):

  count = count + 1;


#Displays the total number of characters present in the given string

print("Total number of characters in a string: " + str(count));
```

Next, I had taken a blank (black background) image of size 200x150. I ran a for loop to replace those black pixels with corresponding pixels from the image (the level 1 image) to get the rearranged image.

```
import cv2

import numpy as np


image=cv2.imread("C:/Research Group/Level1.png", cv2.IMREAD_COLOR)
```

```
img=np.zeros((200,150,3),dtype="uint8")
cv2.imshow("Black",img)
x = 0
y = 0
for i in range(94,177):

    img[y,x,0]=image[6,i,0]
    img[y,x,1]=image[6,i,1]
    img[y,x,2]=image[6,i,2]
    x=x+1



for j in range(7,176):
    for i in range(0,177):

        img[y,x,0]=image[j,i,0]
        img[y,x,1]=image[j,i,1]
        img[y,x,2]=image[j,i,2]
        x=x+1
        if(x>149):
          y=y+1
          x=0




cv2.imshow("New",img)



cv2.waitKey(0)
```

```
k = cv2.waitKey(0)
if k == ord('q'):
    cv2.destroyAllWindows()
```

To find the location of the rearranged image on the larger image I used the function 'cv2.matchTemplate' that used the method cv2.TM_SQDIFF_NORMED to find the location of the rearranged (smaller image) on the larger image. We then draw a rectangle on the larger image to find the x coordinates of the starting point of the smaller image. I have changed the colour of the starting coordinates to verify if the coordinates were correct or wrong (The x coordinate comes out to be 230).

```
import cv2
import numpy as np
method = cv2.TM_SQDIFF_NORMED



# Read the images from the file
small_image = cv2.imread('C:/Research Group/Saved Image.png')
large_image = cv2.imread('C:/Research Group/zucky_elon.png')


result = cv2.matchTemplate(small_image, large_image, method)


# We want the minimum squared difference
mn,_,mnLoc,_ = cv2.minMaxLoc(result)


# Draw the rectangle:
# Extract the coordinates of our best match
MPx,MPy = mnLoc
```

```
# Step 2: Get the size of the template. This is the same size as the match.
trows,tcols = small_image.shape[:2]


# Step 3: Draw the rectangle on large_image
cv2.rectangle(large_image, (MPx,MPy),(MPx+tcols,MPy+trows),(60,76,231),2)


# Display the original image with the rectangle around the match.
cv2.imshow('output',large_image)


start_point=[60,76,231]
# Get X and Y coordinates of all blue pixels
X,Y = np.where(np.all(large_image==start_point,axis=2))
print(X,Y)
large_image[459,230]=(255,0,0)
cv2.imshow("Change", large_image)
# The image is only displayed if we call this
cv2.waitKey(0)
```

Next, we apply a mask on our original image. We define the range of the mask to be from (230,0,0) to (230,255,255). We use cv2.bitwise_and which means that the parts of the image which are same in the image as well as mask would het highlighted whereas the rest of the image would be black. In this way we are able to separate the maze from the coloured noise. The mistake that I had been doing here was assuming it to be 0 in the green and red channels. So initially I had gotten a black image with sprinkled blue dots. On speaking with a senior, Agni Keyoor Purani I was able to understand my mistake and was finally able to obtain the maze.

```
# import the necessary packages
import numpy as np
```

```
import argparse
import cv2
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()


# load the image
image = cv2.imread("C:/Research Group/maze_lv3.png")
boundaries = [([230, 0, 0], [230, 255, 255])]
for (lower, upper) in boundaries:
    # create NumPy arrays from the boundaries
    lower = np.array(lower, dtype = "uint8")
    upper = np.array(upper, dtype = "uint8")
    # find the colors within the specified boundaries and apply
    # the mask
    mask = cv2.inRange(image, lower, upper)
    output = cv2.bitwise_and(image, image, mask = mask)
    # show the images
    cv2.imshow("images", output)
    cv2.imwrite('C:/Research Group/New.jpg',output)




cv2.waitKey(0)
```

Next, I converted the maze to a black and white image.

```
import cv2


originalImage = cv2.imread('C:/Research Group/Moon.jpg')

grayImage = cv2.cvtColor(originalImage, cv2.COLOR_BGR2GRAY)


(thresh, blackAndWhiteImage) = cv2.threshold(grayImage, 50,
255,cv2.THRESH_BINARY)

print("Hello")

cv2.imshow('Black white image', blackAndWhiteImage)

cv2.imshow('Original image',originalImage)

cv2.imshow('Gray image', grayImage)

cv2.imwrite('C:/Research Group/Sun.jpg',blackAndWhiteImage)


cv2.waitKey(0)

cv2.destroyAllWindows()
```

Initially I had used this image itself as an input for the A* algorithm. But the output I received was almost a straight line. This was due to the fact that the white was not that visible and sort of scattered. Hence, I used dilution to get a much clearer image.

```
import cv2

import numpy as np


# Reading the input image

img = cv2.imread("C:/Research Group/Moon.jpg")


# Taking a matrix of size 5 as the kernel

kernel = np.ones((3,3), np.uint8)
```

```
# The first parameter is the original image,
# kernel is the matrix with which image is
# convolved and third parameter is the number
# of iterations, which will determine how much
# you want to erode/dilate a given image.


img_dilation = cv2.dilate(img, kernel, iterations=1)
cv2.imshow('Input', img)
cv2.imshow('Dilation', img_dilation)
cv2.imwrite("C:/Research Group/Dilation_1.jpg",img_dilation)
cv2.waitKey(0)
```

After this I had to find the start and end coordinates. For this I used the function cv2.EVENT_LBUTTONDOWN and cv2.EVENT_RBUTTONDOWN which display the coordinates of the point on the image where I had pressed the key.

```
# importing the module
import cv2

# function to display the coordinates of
# of the points clicked on the image
def click_event(event, x, y, flags, params):

    # checking for left mouse clicks
    if event == cv2.EVENT_LBUTTONDOWN:

        # displaying the coordinates
        # on the Shell
        print(x, ' ', y)
```

```
            # displaying the coordinates
            # on the image window
            font = cv2.FONT_HERSHEY_SIMPLEX
            cv2.putText(img, str(x) + ',' +
                        str(y), (x,y), font,
                        1, (255, 0, 0), 2)
            cv2.imshow('image', img)

        # checking for right mouse clicks
        if event==cv2.EVENT_RBUTTONDOWN:

            # displaying the coordinates
            # on the Shell
            print(x, ' ', y)

            # displaying the coordinates
            # on the image window
            font = cv2.FONT_HERSHEY_SIMPLEX
            b = img[y, x, 0]
            g = img[y, x, 1]
            r = img[y, x, 2]
            cv2.putText(img, str(b) + ',' +
                        str(g) + ',' + str(r),
                        (x,y), font, 1,
                        (255, 255, 0), 2)
            cv2.imshow('image', img)

# driver function
if __name__=="__main__":

    # reading the image
```

```
img = cv2.imread('C:/Research Group/Moon.jpg', 1)

# displaying the image
cv2.imshow('image', img)

# setting mouse hadler for the image
# and calling the click_event() function
cv2.setMouseCallback('image', click_event)

# wait for a key to be pressed to exit
cv2.waitKey(0)

# close the window
cv2.destroyAllWindows()
```

Now after getting the maze and the start and end coordinates, I solve it using A* algorithm.

A* algorithm has 3 parameters.

1) g: It is the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell.

2) h: It is also known as the heuristic value; it is the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. One must make sure that there is never an over estimation of the cost.

3) f: It is the sum of g and h.

So, f = g + h

The algorithm makes its decisions by taking the f-value into account. The algorithm selects the smallest f-valued cell and moves to that cell. This process continues until the algorithm reaches its goal cell.

Terminology:

Node (also called State) — all potential positions or stops with a unique identification

Transition — the act of moving between states or nodes.

Starting Node — where to start searching

Goal Node — the target to stop searching

Search Space — a collection of nodes, like all board positions of a board game

Cost — numerical value (say distance, time, or financial expense) for the path from a node to another node.

g(n) —the exact cost of the path from the starting node to any node n

h(n) —the heuristic estimated cost from node n to the goal node

f(n) — lowest cost in the neighbouring node n

Each time A* enters a node, it calculates the cost, f(n) (n being the neighbouring node), to travel to all of the neighbouring nodes and then enters the node with the lowest value of f(n). These values are calculated using following formula.

$f(n) = g(n) + h(n)$

I have tried to implement the A* code using different heuristics like Manhattan distance, Euclidian distance, diagonal distance, H1(non – admissible) or Dijkstra (that is no heuristics)

I have used priority queue to decrease the time complexity of the code

I have implemented different heuristics by defining another variable which stores the choice of the user and calculates accordingly. The different choices also allow you to choose if you want to have diagonal distance and the cost type.

The cost function returns the cost as 1 if you move right or down and 1.41 if you move diagonally. And I have defined another cost function that accounts for extra movement.

```
import numpy as np
import cv2
import collections
import heapq
```

```
'''
Why use heap?
The heap implementation ensures time complexity is logarithmic. Thus push/pop operations are
proportional to  the base-2 Logarithm of number of elements.
Implementation is through a binary tree (Just like a sieve!)
Heap property: Value of node is always smaller than both of its children unlike a binary search tree.
'''
import queue
import time
import pandas as pd

'''
Why Priority Queue?
Because here we have to decide the priority. So what if we had a queue that adjusts the priority for us!
Earlier attempt was to use a list/collection of nodes and find out the minimum. But the time complexity increases in that case.
'''
#Appendix:
#Diagonal Distance=1
#Dijkstra=2
#Euclidean=3
#H1=4   ...A non-admissible Heuristic.
#Manhattan=5



#Appendix:
choice=3 # Heuristic_type
```

```
choice2=2 # Travel_selection
'''
Choice2=1 means diagonal movement allowed
Choice2=2 means diagonal movement not allowed
'''
choice3=1  # Cost_selection
'''
Choice3=1 means normal given cost
Choice3=2 valid for only Choice2=2
It is a cost method developed assuming that a bot takes some time to turn.
It is designed so that the bot will prefer to go straight (According to the
path along which it has entered the current node)
'''


heuristic=['Diagonal Distance','Dijkstra','Euclidean','H1','Manhattan']
#For documentation
write_path='a a astar.txt'
#Read Image Path
img_path='C:/Research Group/Dilation_1.jpg'
#img_path='sample_for_Astar.png'


#Write Image Path
img_write_path='AStar_'+str(choice)+'_'+heuristic[choice-1]+'_'+'Case-
'+str(choice2)
if(choice2==2 and choice3==2):
    img_write_path=img_write_path+'_2'
img_write_path=img_write_path+'.png'


#Written as variables so that making changes in code according to the need
become easy
```

```
wait=10000
#Define color parameters
ob_color = [255,255,255]
np_color =[0,0,0]
path_pointer=[255,255,0]
start=(139,60)
end=(141,420)


#Read Image
img = cv2.imread(img_path,cv2.IMREAD_COLOR)
#Calculate image size before search
h,w,c = img.shape


#Calculate Heuristic Func
def calcHeuristic(point1,point2=None,startpt=None,endpt=None):

    '''
 Calculates Heuristic Function according to the global choice selected.\n
 Calculates the Heuristic between points point1 and point2


 choice==1 Diagonal Distance\n
 returns max(abs(point1[0] - point2[0]),abs(point1[1] - point2[1]))\n
 choice==2 Dijkstra\n
 returns 0 (No Heuristic)\n
 Choice==3 Euclidean=3\n
 returns ((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)**0.5\n
```

```
 Can delete **0.5 according to need\n

 Choice==4 H1   ...A non-admissible Heuristic.\n

 H1 is a non-Admissible Heuristic.\n Based on distance between point and a
line.\nWorks well with no obstacles\n

 Choice==5 Manhattan\n

 returns abs(point1[0] - point2[0]) + abs(point1[1] - point2[1])\n

 @param startpt and endpt required for H1\n
    '''
    if(choice==1):
        return max(abs(point1[0] - point2[0]),abs(point1[1] - point2[1]))


    if(choice==2):
        return 0


    if(choice==3):
        return ((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)**0.5


    if(choice==4):
        a=start[1]-end[1]
        b=end[0]-start[0]
        c=start[0]*end[1]-start[1]*end[0]
        if(a*point1[0]+b*point1[1]+c==0):
            return -10
        return abs(a*point1[0]+b*point1[1]+c)


    if(choice==5):
        return abs(point1[0] - point2[0]) + abs(point1[1] - point2[1])
```

```
def calcCost(point1,point2):
    '''
    Calculates cost
    '''
    if(abs(point1[0]-point2[0])==1 and abs(point1[1]-point2[1])==0):
        return 1.0
    if(abs(point1[0]-point2[0])==1 and abs(point1[1]-point2[1])==1):
        return 1.41421356237      #define square root of 2
    if(abs(point1[0]-point2[0])==0 and abs(point1[1]-point2[1])==1):
        return 1.0
    if(abs(point1[0]-point2[0])==0 and abs(point1[1]-point2[1])==0):
        return 0.0
    else:
      return np.inf



def calcCost2(child,current,start,parent=None):
    '''
    Special cost function\n
    based on real runtime situation.\n
    Accounts time taken for turning by the bot\n

    '''
    point1=child.position
    point2=current.position
    start_point=start.position
    if not (point2[0]==start_point[0] and point2[1]==start_point[1]):
```

```
        parent_point=parent.position
        if(float(point2[0])==(point1[0]+parent_point[0])/2 and
float(point2[1])==(point1[1]+parent_point[1])/2 ):
            return 1
        if(parent_point[0]==point1[0] and parent_point[1]==point1[1]):
            return 3
        if(point1[0]==point2[0] and point1[1]==point2[1]):
            return 0
        else:
            return 2
    else:
     if(abs(point1[0]-point2[0])==1 and abs(point1[1]-point2[1])==0):
         return 1
     if(abs(point1[0]-point2[0])==0 and abs(point1[1]-point2[1])==1):
         return 1
     if(abs(point1[0]-point2[0])==0 and abs(point1[1]-point2[1])==0):
         return 0




#check for navigation path
def isinrange(img,position):
    '''
    Returns False if given node is not accessible
    \nElse returns True
    '''
    b=False
    x=position[0]
    y=position[1]
    ob_color = [255,255,255]
    if(x>=0 and x<img.shape[0] and y >=0 and y<img.shape[1]):
```

```
        b=True
        if(img[x,y,0] ==ob_color[0] and img[x,y,1] ==ob_color[1] and
img[x,y,2] ==ob_color[2]):
            b=False
    return b



# class for handling priority queues
class PriorityQueue :
    '''
    creates a Queue Class
    '''


    def __init__(self):
        '''
        Creates a queue
        '''
        self.Queue=[]


    def isempty(self):
        '''
        Checks if given Queue is empty.
        \nReturns True if Empty
        '''
        if not self.Queue:
            return 1
        else:
             return 0
```

```
    def put(self,index):
        '''
        Puts given element onto the queue
        \nUses heapq.heappush for faster aproach.
        '''
        heapq.heappush(self.Queue,index)


    def get(self):
        '''
        Returns the smallest-in-priority element using heapq.heappop
        \n\nComparison based on __lt__ and __gt__ of node class
        '''
        #print(self.Queue)
        return heapq.heappop(self.Queue)




class node:
    '''
    class for handling nodes\n
    @param index=input default=None\nstands for position\n
    @param is_in_list and is_current are check-points\n
    @param parent=input default=None\n
    @param f,g,h=cost\n
    @param h stands for Heuristic cost\n
    @param f stands for total cost
    '''
    #constructor
    def __init__(self,index=None,parent=None):
        # Initialise params
```

```
        self.position=index  #position/location
        self.is_in_list=False  # chkpt
        self.is_current=False  #chkpt
        self.parent=parent     #parent node of the node
                                  # __
        self.f=np.inf             #   |
        self.g=np.inf             #   | initialise cost
        self.h=np.inf             # __|
        self.isvisted=False

    def __lt__(self,other):
     '''
     overload operator < (chk parameter=cost)
     '''
     return self.f<other.f

    def __gt__(self,other):
     '''
     overload operator > (chk parameter=cost)
     '''
     return self.f>other.f




#get neighbourhood according to choice2
def get_nbd(current):
    '''
    returns list of neighbourhood positions
    \nchoice is made based on choice2
    '''
```

```
    l=[]


    if(choice2==1):
     for i in range(-1,2):
         for j in range(-1,2):
             position=(current.position[0]+i,current.position[1]+j)
             if (isinrange(img,position)==True):
                 l.append(position)


    if(choice2==2):
     for i in range(-1,2):
         position=(current.position[0]+i,current.position[1])
         if (isinrange(img,position)==True):
                l.append(position)
     for j in range(-1,2):
         if(j==0): #skip repetition
             continue
         position=(current.position[0],current.position[1]+j)
         if (isinrange(img,position)==True):
             l.append(position)

    return l


# create a matrix/dict of objects
node_matrix=np.empty((h,w),dtype=object)
#node_matrix=collections.defaultdict(node)
for i in range(h):
    for j in range(w):
        node_matrix[i,j]=node()
```

```
            node_matrix[i,j].position=(i,j)



#main function
def main_func(img,startpt,endpt):
    '''
    The main traversing function\n
    returns time taken for traversing
    '''
    #create lists
    visited=[]
    path=[]


    h,w,c=img.shape


    #create priority queue
    pt_list=PriorityQueue()

    #Initialise start node
    start=node(startpt)
    start.g=0
    start.f=start.h=calcHeuristic(start.position,endpt)
    start.isvisted=True
    node_matrix[startpt]=start
    pt_list.put(start)


    #Start Traversing
    beg=time.time()
```

```
    while(not pt_list.isempty()):

        #while the list is not empty obtain the node with smallest f

        current=pt_list.get()

        #Now that current node is not in the list change the corresponding
chkpts

        current.is_in_list=False
        current.is_current=True

        #Now to preserve the node use the matrix

        node_matrix[current.position]=current

        #break condition

        if(current.position==endpt):
            break

        #get the neighbourhood points
```

```
        nbd_list=get_nbd(current)

        # searching operation

        for pos in nbd_list:
            # earlier attempt was to use a list of objects. But finally
dealing with position.

            nbd=node_matrix[pos]

            #get the temp_cost

            if(choice2==2 and choice3==2):
                g_temp=current.g+calcCost2(nbd,current,start,current.parent)
            else:
                g_temp=current.g+calcCost(pos,current.position)

            # if this newly calculated cost< the stored cost-
            if(g_temp<nbd.g):

                #make note that it is visited

                nbd.isvisted=True

                #if the point is current and the new cost is less than its
stored cost-
```

```
                #make the current as it's parent and put the nbd node in the
pt_list.

                if nbd.is_current:
                    nbd.is_current=False
                    nbd.parent=current
                    nbd.g=g_temp
                    nbd.h=calcHeuristic(nbd.position,endpt)
                    nbd.f=nbd.g+nbd.h
                    nbd.is_in_list=True
                    pt_list.put(nbd)


                # if the nbd node is not current- then put the nbd node in the
pt_list (if it is not there) along with setting the chk_points.
                else:
                    nbd.parent=current
                    nbd.g=g_temp
                    nbd.h=calcHeuristic(nbd.position,endpt)
                    nbd.f=nbd.g+nbd.h
                    if(not nbd.is_in_list):
                        pt_list.put(nbd)
                    nbd.is_in_list=True
                    nbd.is_current=False  #Just for ensuring


        # to display progress  ************************
        #showPath(img,current,start=start,is_end=False)
        #cv2.waitKey(1)



    # finish traversing
```

```
    finish=time.time()
    print(round(finish-beg,3))

    # display final path
    visited,path,cost=showPath(img,current,True,start,visited,path)
    path.reverse()

    #end with saving the results in a txt document
    documentation(visited,path,finish-beg,cost,start,current,1)

    #documentation(visited,path,finish-beg,cost,start,current,0)

    return finish-beg


def showPath(img,current,is_end,start,visited_list=None,parent_list=None):
    '''
    displays current progress as image if is_end is disabled\n
    if is_end is enabled-\n
    returns visited list and parent list and calculates cost
    '''
    cost=0.0
    visited_color=[100,0,100]
    path_pointer=[0,255,0]

    # to avoid errors
    if(parent_list==None):
```

```
    parent_list=[]
if(visited_list==None):
    visited_list=[]


img2=np.copy(img)

# display/compile visited path



while(current.position!=start.position):

    #display image

    temp=current.position
    img2[temp[0],temp[1],0]=path_pointer[0]
    img2[temp[0],temp[1],1]=path_pointer[1]
    img2[temp[0],temp[1],2]=path_pointer[2]

    if(not is_end):
        current=current.parent
        continue

    else:
     parent_list.append(temp)
```

```
            temp=current.parent

            #calculate cost

            if(choice2==2 and choice3==2):
                cost=cost+calcCost2(current,temp,start,temp.parent)
            else:
                cost=cost+calcCost(current.position,temp.position)

            current=temp

        if(is_end):
         parent_list.append(start.position)

        #imshow and imwrite

        cv2.resize(img2,(1000,1000))
        cv2.namedWindow('path',cv2.WINDOW_NORMAL)
        cv2.imshow('path',img2)
        if(is_end):
            cv2.imwrite(img_write_path,img2)

        #return the lists and the cost
        return (visited_list,parent_list,cost)


# display option changed according to option
```

```
def display(val,option,w=None):
    '''
    write data according to option specified
    '''
    if(option==0):
        print(val)
    else:
        w.write(val)
        w.write('\n')



# Documentation according to option


def documentation(visited_list,parent_list,val,cost,start,end,option):
    '''
    document the data
    '''
    if(option==1):
     #open desired file to write result-
     w=open(write_path,'w')


     #write the data
    display(str(choice)+')'+heuristic[choice-1]+' case('+str(choice2)+') -
',option,w)
    if(choice2==2 and choice3==2):
        display('Using a different cost function',option,w)
    display('cost of traversing by distance='+str(round(cost,2)),option,w)
    display('cost stored at end point='+str(round(end.f,2)),option,w)
```

```
    display('No. of nodes in path='+str(len(parent_list)),option,w)
    display('No of nodes visited='+str(len(visited_list)),option,w)
    display('start point='+str(start.position),option,w)
    display('end point='+str(end.position),option,w)
    display('Nodes in path-',option,w)
    #display('Nodes visited-',option,w)
    #display(str(visited_list),option,w)
    display('time taken for traversing='+str(val),option,w)
    display(str(parent_list),option,w)


    if(option==1):
     #close file
     w.close()



t=main_func(img,start,end)
#print(t)
cv2.waitKey(0)
```

On solving the maze, you notice the word. The password was Apple.

For the last part of the code, I had used SciPy library to convert the image into an audio file. I had run a for loop to find the data of the image and store it in an array and then convert it to sound but I got static noise.

This was my first try...

```
import numpy as np
from scipy.io.wavfile import write
from PIL import Image
from numpy import asarray, uint16
import cv2
img = Image.open('C:/Users/admin/Downloads/treasure_mp3.png')# convert image to 8-bit grayscale
#img=cv2.imread("C:/Users/admin/Downloads/treasure_mp3.png")
WIDTH, HEIGHT = img.size
#data = [[0 for i in range(390)] for j in range(390)]
#WIDTH=390
#HEIGHT=390
print(WIDTH,HEIGHT)
#for i in range (WIDTH):
    #for j in range (WIDTH):
        #data[i][j]=img[i][j]


data = list(img.getdata())
data = ([data[offset:offset+WIDTH] for offset in range(0, WIDTH*HEIGHT, WIDTH)])
print(data)
arr = np.array(data)
scaled = np.int16(arr/np.max(np.abs(arr)) * 32767)
write('test_new.wav', 44100, scaled)
```

This is what I had tried next :

```
import numpy as np
import cv2
from scipy.io.wavfile import write
```

```
cv2.namedWindow('image',cv2.WINDOW_NORMAL)
img=cv2.imread('C:/Users/admin/Downloads/treasure_mp3.png',0)
n,m=img.shape
k=0
data=np.empty(n*m,np.int16)
for i in range(n):
    for j in range (m):
        data[k]=int(img[i][j])
        k=k+1;


#scaled=np.int16(data/np.max(np.abs(data))*32767)
write('test.wav', n*m, data)
cv2.imshow('image',img.astype(np.uint8))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

I had gotten several errors while implementing the algorithms. Firstly, when I was trying to convert to data to its respective character it was all getting printed in a new line making it very difficult to read. Using end="" solved the problem. Also, when I was rearranging the image, it showed an error that it was out of size. I was able to rectify this error by messing around a bit and trying to understand what was happening. Using debugging techniques, I was able to keep a track of variables thus making it simpler to rectify. Another problem that I encountered was during dilation. When I first applied the kernel, the image was completely white. On changing the values of the kernel and number of iterations I was able to obtain the maze. For the last part of the task, I am getting static noise and I really was not able to solve this art of the task.

VI. Results and Observations

I have used A* algorithm to solve the maze. I had tried to implement several different Heuristics techniques in it like using Manhattan distance, Euclidian distance, diagonal distance or Dijkstra (that is no heuristics). The drawback of A* algorithm is that it uses

heuristics so it might not always be the shortest path. Dijkstra's algorithm is more helpful if you want to find the shortest path whereas A* helps you to decrease the time that your algorithm takes to find the solution. RRT algorithms are faster than A* algorithms but the path that is shown is usually longer than A*. So RRT algorithm would give a longer path but would take lesser time whereas Dijkstra would take longer time but give a shorter path. I feel that A* was a good combination of both time and distance and hence used it solve the maze. Plus, I was more familiar with it.

Heuristic Choice	Diagonal Distance	Dijkstra	Euclidean Case	H1 (Non Admissible)	Manhattan
cost of traversing by distance	1702.0	1702.0	1702.0	1702.0	1702.0
cost stored at end point	1702.0	1702.0	1702.0	1692.0	1702.0
No. of nodes in path	1703	1703	1703	1703	1703
No of nodes visited	0	0	0	0	0
time taken for traversing	1.9102973937988281	2.1914866924285889	1.7515993118286133	1.5631198883 05664	1.8035826683044434

VII. Future Works

The future work that could be done is to improve on the maze, as in to decrease the noise further and to use sliders to be able to view the text given at the bottom of the maze. Also, I would also like to solve the maze using different algorithms like RRT and RRT* to compare the different results. The algorithms could be modified to account for different roads like for a rough path or a path that has a busier route. And yeah, I want to convert that image to an audio file as well.

Conclusion

The problem was about image processing and path finding. Variety of applications were included in this task like dilation, erosion, cropping of images, reading images and converting them to audio files, noise reduction etc...

Image processing can be used in several application such as computer vision, face detection, sending coded messages, remote sensing, feature extraction etc...

Path finding algorithms can be used to find routes which have least obstacles or shortest distance paths or paths that take the least time.

References

http://robotics.caltech.edu/wiki/images/e/e0/Astar.pdf

https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/astar.html

https://note.nkmk.me/en/python-numpy-image-processing/