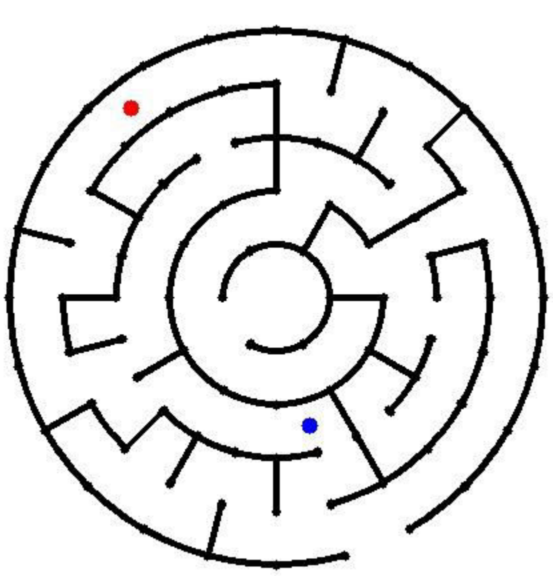
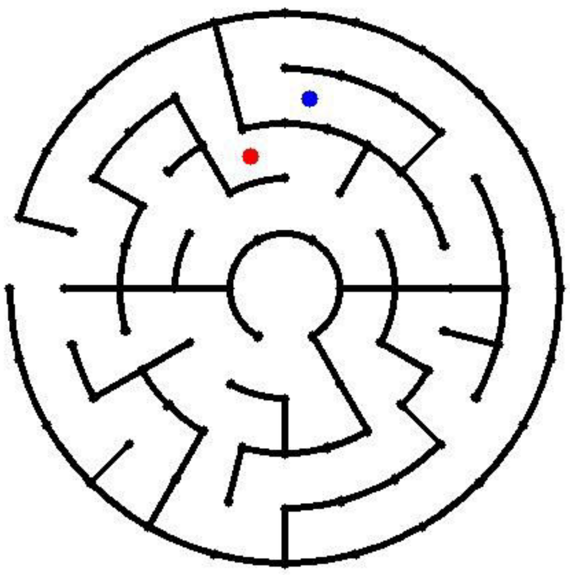
Task-2 – Practice

Section – 2

In this section we are representing locations to be traversed as coloured **markers** that are scattered randomly around the mazes.

We can have maze images such as these:

There will be two markers that will be placed in each maze image and they will distinct colours (**Blue**, **Red**).
Start and **Finish** have been defined in the Task1_Section_1_Description.pdf file.

The aim of this section is to find a path from **Start** to **Finish** such that the following criteria are satisfied:

- Only one of the markers needs to be acquired by the algorithm.
- The marker that will be acquired depends on which sum total path will be shorter.
- If the path traversed for acquiring Red marker is equal to path traversed for acquiring Blue marker then the path with the shorter distance of the respective marker from **Start** is chosen.

Do the following:

1. Open the Task2_Practice Folder.
2. In the Section-2 folder open the section2.py folder
3. Follow the instructions to modify functions in the section2.py file as given below:

MAIN

```
if __name__ == '__main__':
    filePath = 'maze00.jpg'## Specify the filepath of image here
    img = main(filePath)
    cv2.imshow('canvas', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

This section of code calls the main() function with a filepath which you can specify. You can specify filepath as "maze00.jpg" to "maze09.jpg". Apart from the filepath please do not change any other code in this section.

MAIN CODE

```
def main(filePath, flag = 0):
    img = readImageHSV(filePath)
    imgBinary = readImageBinary(filePath)
    if len(img) == 440:
        size = 1
    else:
        size = 2
    listofMarkers = findMarkers(    )
    path = findOptimumPath(    )
    img = colourPath(     )
    print path
    print listofMarkers
    if __name__ == "__main__":
        return img
    else:
        if flag == 0:
            return path
        elif flag == 1:
            return str(listofMarkers) + "\n"
        else:
            return img
```

This is the main() function which is being called in the previous sections. The functions called in the main function will be explained in detail. You are expected to understand what this snippet of code does, on your own.

You need to provide arguments in the function calls for findMarkers(), findOptimumPath() and colourPath() functions. Other than that you are not allowed to change this section of code.

FUNCTIONS

```
def findNeighbours(img, level, cellnum, size):
    neighbours = []
    ####################    Add your Code Here ########################



    ####################################################################
    return neighbours

def colourCell(img, level, cellnum, size, colourVal):
    #####################   Add your Code Here   ######################
    ####################################################################
    return img

##  Function that accepts some arguments from user and returns the graph
##  of the maze image.
def buildGraph(  ):   ## You can pass your own arguments in this space.
    graph = {}
    #####################     Add your Code Here     ###################


    ####################################################################
    return graph

def findPath(    ):   ## You can pass your own arguments in this space.
    ######################   Add your Code Here    #####################


    ####################################################################
    return shortest

def findMarkers(    ): ## You can pass your own arguments in this space.
    list_of_markers = {}
    ########################   Add your Code Here ######################


    ####################################################################
    return list_of_markers
```

These functions have already been explained previously in Task 1. These functions need to be modified in order for them to work for theta mazes. Hence you need to write the theta maze equivalent of each of these functions.

FUNCTIONS

```
##  Function accepts some arguments and returns the Start coordinates of
##  the maze.
def findStartPoint(  ):## You can pass your own arguments in this space.
    ########################  Add your Code Here   #####################


    ####################################################################
    return start
```

This function returns the coordinates of Start for each maze.

FUNCTIONS

```
def findOptimumPath(   ):## You can pass your own arguments in this space.
    #######################   Add your Code Here  ########################


    ######################################################################
    return pathArray
def colourPath(     ):    ## You can pass your own arguments in this space.
    #######################    Add your Code Here    #######################


    ######################################################################
    return img
```

The findOptimalPath function returns a python list which consists of all paths that need to be traversed in order to start from the START cell, traverse to any one of the markers (either Blue or Red) and then traverse to FINISH. The length of path should be shortest (most optimal solution).

The colourPath() function highlights the whole path that needs to be traversed in the maze image and returns the final image.

UTILITY FUNCTIONS

```
##  Returns sine of an angle.
def sine(angle):
    return math.sin(math.radians(angle))

##  Returns cosine of an angle
def cosine(angle):
    return math.cos(math.radians(angle))

##  Reads an image from the specified filepath and converts it to
##  Grayscale. Then applies binary thresholding to the image.

def readImageBinary(filePath):
    mazeImg = cv2.imread(filePath)
    grayImg = cv2.cvtColor(mazeImg, cv2.COLOR_BGR2GRAY)
    ret,binaryImage = cv2.threshold(grayImg,127,255,cv2.THRESH_BINARY)
    return binaryImage

## Reads image in HSV format. Accepts filepath as input argument and
## returns the HSV equivalent of the image.
def readImageHSV(filePath):
    mazeImg = cv2.imread(filePath)
    hsvImg = cv2.cvtColor(mazeImg, cv2.COLOR_BGR2HSV)
    return hsvImg
```

These are utility functions that have been predefined for your ease.

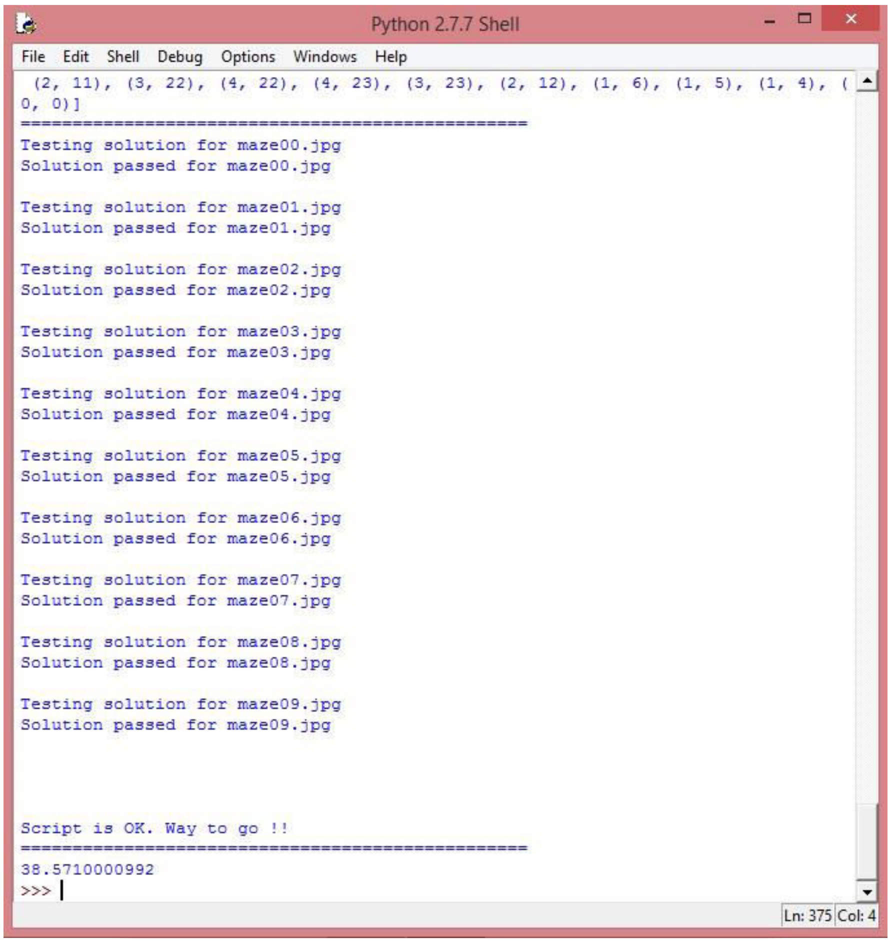
You are allowed to add utility functions to the code as per your requirement as long they are called inside the functions definitions of functions you are supposed to write code for.

Testing the Solution

In the **Section-2 folder,** in addition to maze test images and ***section2.py*** there are three more files, namely the ***section2.txt*** file, ***hash.txt file*** and the ***TestSuite_2.py.***

You are **not allowed** to make any changes to these three files. Anybody found to have tampered with these files will be disqualified.

After you are done modifying the code in the ***section2.py*** file, open the ***TestSuite_2.py*** file and run it in the python shell. The output of **TestSuite_2** script should resemble the following screenshot:



```
 (2, 11), (3, 22), (4, 22), (4, 23), (3, 23), (2, 12), (1, 6), (1, 5), (1, 4), (
0, 0)]
==================================================
Testing solution for maze00.jpg
Solution passed for maze00.jpg

Testing solution for maze01.jpg
Solution passed for maze01.jpg

Testing solution for maze02.jpg
Solution passed for maze02.jpg

Testing solution for maze03.jpg
Solution passed for maze03.jpg

Testing solution for maze04.jpg
Solution passed for maze04.jpg

Testing solution for maze05.jpg
Solution passed for maze05.jpg

Testing solution for maze06.jpg
Solution passed for maze06.jpg

Testing solution for maze07.jpg
Solution passed for maze07.jpg

Testing solution for maze08.jpg
Solution passed for maze08.jpg

Testing solution for maze09.jpg
Solution passed for maze09.jpg



Script is OK. Way to go !!
==================================================
38.5710000992
>>> 
```

If it runs successfully without any errors then your solution is correct and it passed all 10 test cases provided to you.

Please note that files submitted by you will be run through similar test cases.

You are done with Task 2.

Way to go!!