Task-2 – Practice

Section - 1

In the last task, you learned about mazes and ways to solve mazes. In this task you will learn about a new kind of maze, called a **Theta maze**.

Theta mazes are composed of concentric circles of passages with the following properties:

- Centre can be either **Start** or **Finish.**
- If Start is in the centre, Finish is on the Outer Edge and vice versa.
- Cells typically have 4-5 possible passage connections.

The aim of this task is to find a solution path for a given Theta maze from Start to Finish. The Theta mazes can be of two different sizes as illustrated in Figures 1 and 2.

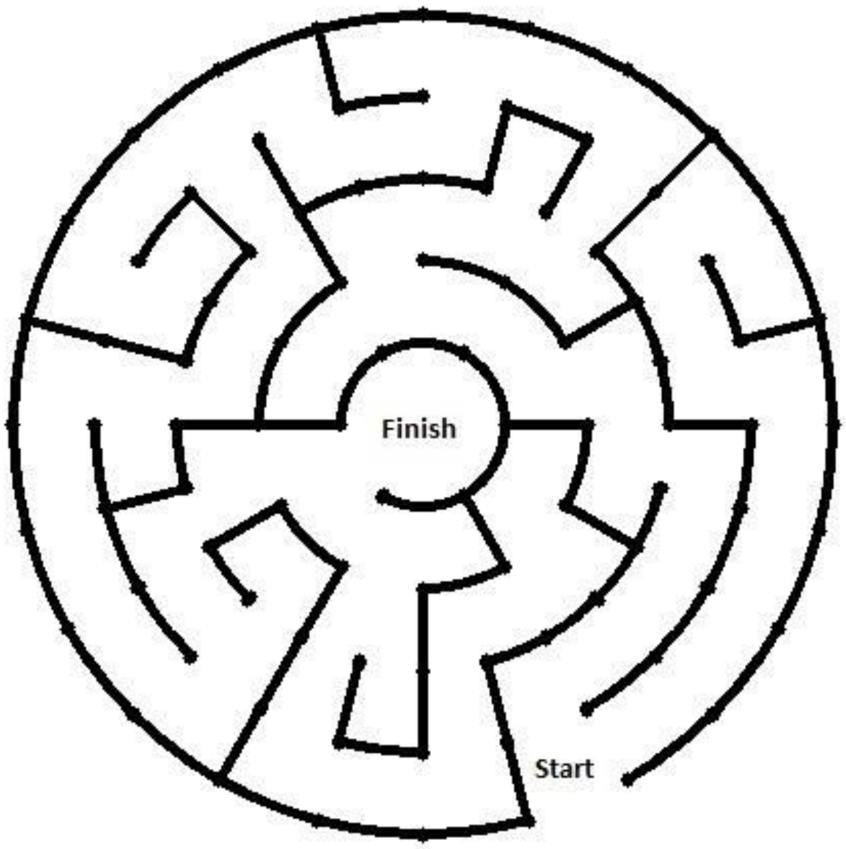


Figure 1: Maze 1 - Having 4 concentric circular passages

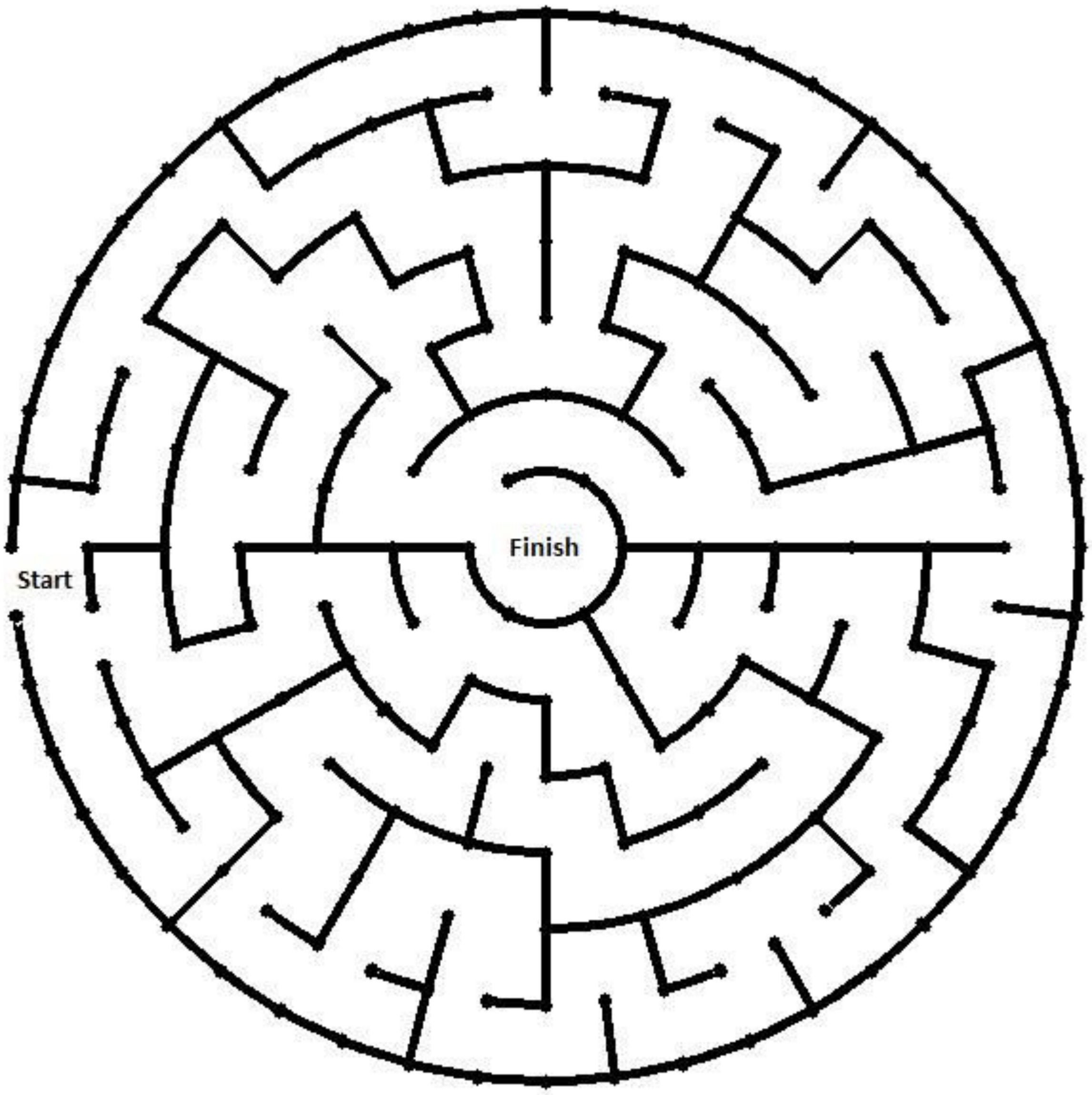


Figure 2: Maze 2 – Having 6 concentric circular passages.

Creating a Theta Maze

In this section, we will learn about how a Theta maze is constructed. We start with a **Theta Grid.** An example is shown in Figure 3.

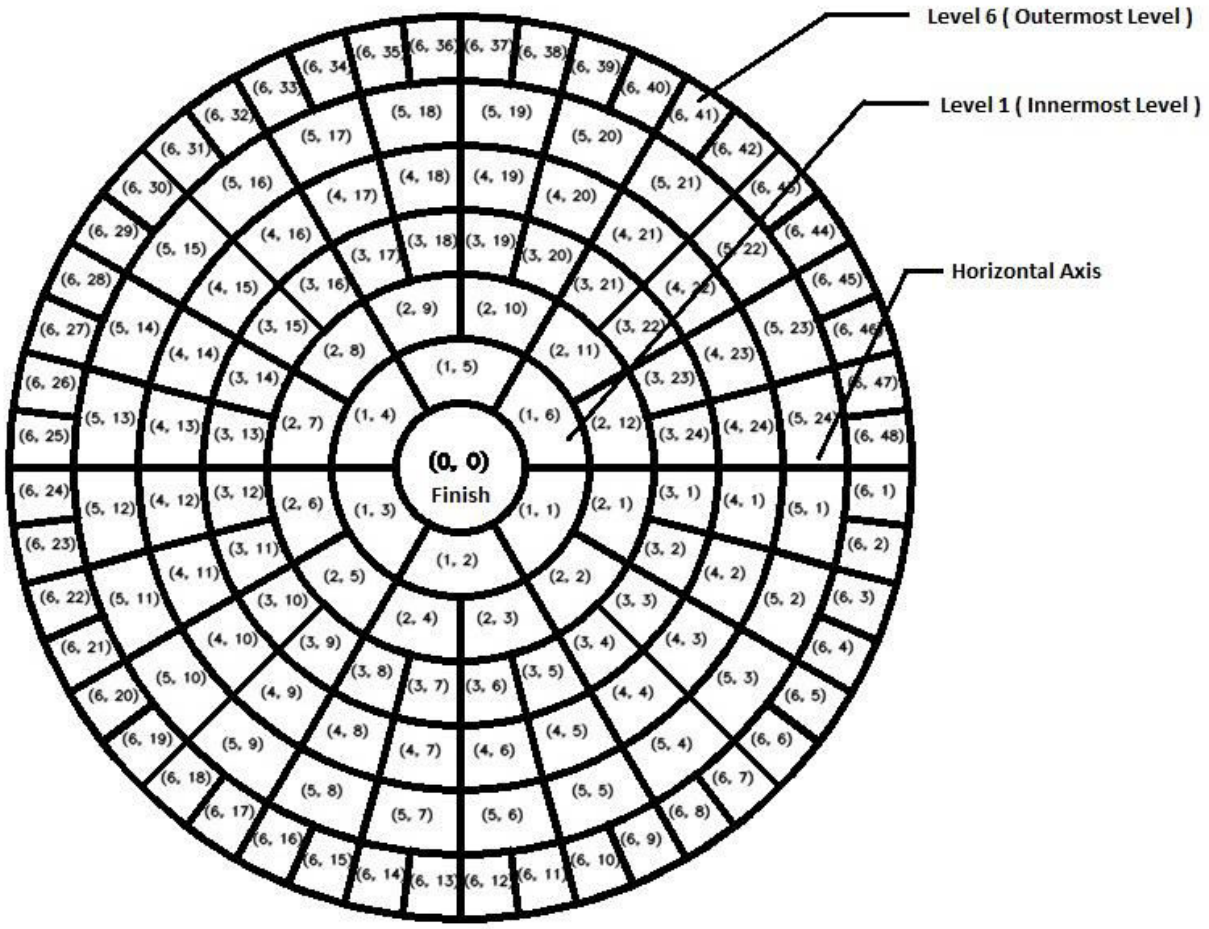


Figure 3: Theta Grid

Considering Figure 3,

- The innermost circular cell is called the **Finish.** It has coordinates of (0, 0).
- Each of the concentric circular passages is called a Level and they are numbered from inner circle to outer circle.
- The innermost level is labeled as Level 1 and the outermost level as Level 6.
- The cells are numbered using the following convention: Cells in the same level are numbered clockwise from below the **Horizontal Axis** on the right as shown in Figure 3.
- The number of cells in each level changes as we go from the innermost level to the outermost level. As you can see that the innermost level only has 6 cells while the outermost level has 48 cells.
- Number of cells in a particular level remains the same. For example, number of cells in level 4 will always be 24 regardless of the size of the Theta grid.
- Width of each level is **40 units.** Hence distance of each cell from centre of maze can be calculated.
- Since the number of cells in each level is finite, we can calculate the sector of angle swept by each cell. For example, in Level 1, there are 6 cells, hence each cell sweeps an angle of 360°/6 = 60°.
- Each cell's coordinates are represented using the following convention: (level, cell_number)

A Theta grid can be converted into a Theta maze by using special algorithms to carve out passages through the grid. An example maze carved out of Figure 3 is shown in Figure 4.

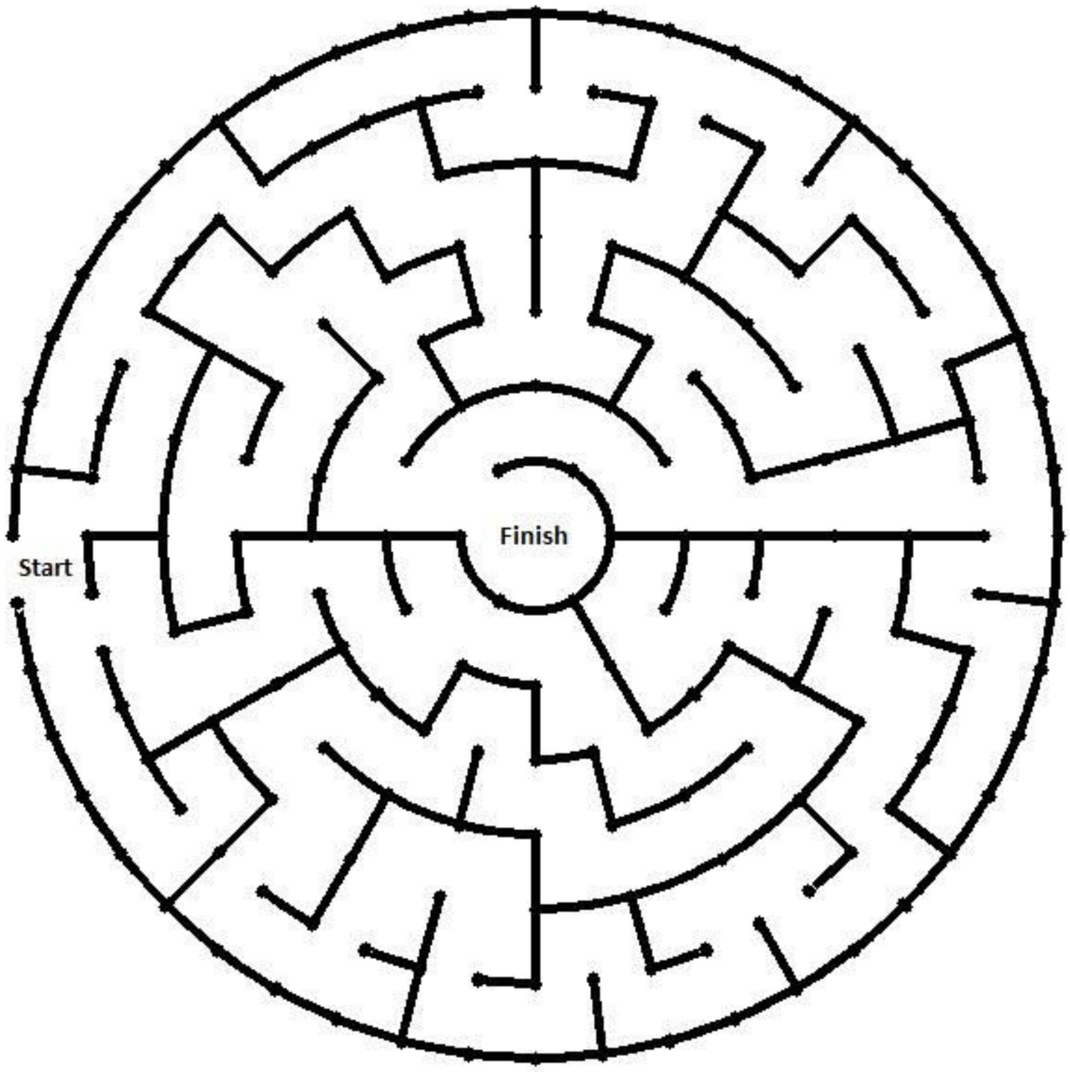


Figure 4: A Theta Maze

The cell with the opening towards the outside of the maze is called **Start.** The aim of this task is to find a path from **Start** to **Finish.**

Do the following:

1. Open the Task2_Practice Folder.
2. In the Section-1 folder open the section1.py folder
3. Follow the instructions to modify functions in the section1.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 FUNCTION

```
def main(filePath, flag = 0):
    img = readImage(filePath)    ## Read image with specified filepath
    if len(img) == 440: ## Smaller maze image has dimensions 440x440
        size = 1
    else:
        size = 2
    maze_graph = buildGraph(    )
    start = findStartPoint(    )
    shortestPath = findPath(    )
    print shortestPath
    string = str(shortestPath) + "\n"
    for i in shortestPath:     ## Loop to paint the solution path.
        img = colourCell(img, i[0], i[1], size, 230)
    if __name__ == '__main__':  ## Return value for main() function.
        return img
    else:
        if flag == 0:
            return string
        else:
            return graph
```

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 buildGraph(), findStartPoint() and findPath() functions. Other than that you are not allowed to change this section of code.

The following section will explain the functions used in the ***section1.py*** script in detail:

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
```

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.

UTILITY FUNCTIONS

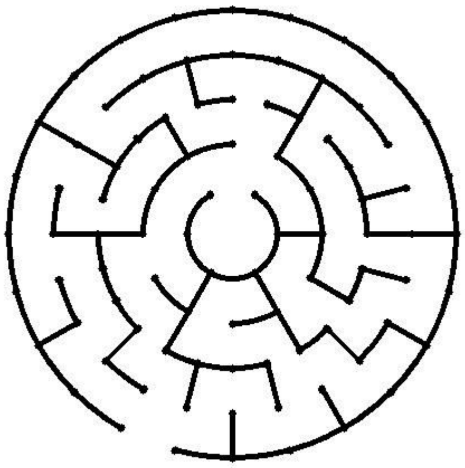
```
##  Returns sine of an angle.
def sine(angle):
    return math.sin(math.radians(angle))

##  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.
```

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

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

OUTPUT OF PROGRAM

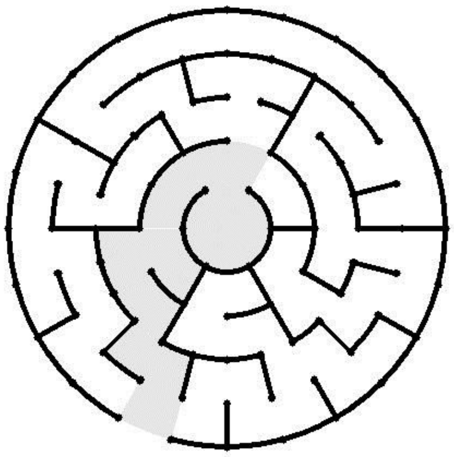
If we provide the filepath for the following image in the space indicated:



The output for this program should be:

```
>>> [(4, 8), (3, 8), (3, 9), (2, 5), (2, 6), (1, 3), (1, 4), (1, 5), (0,
0)]
```

The output image should be:

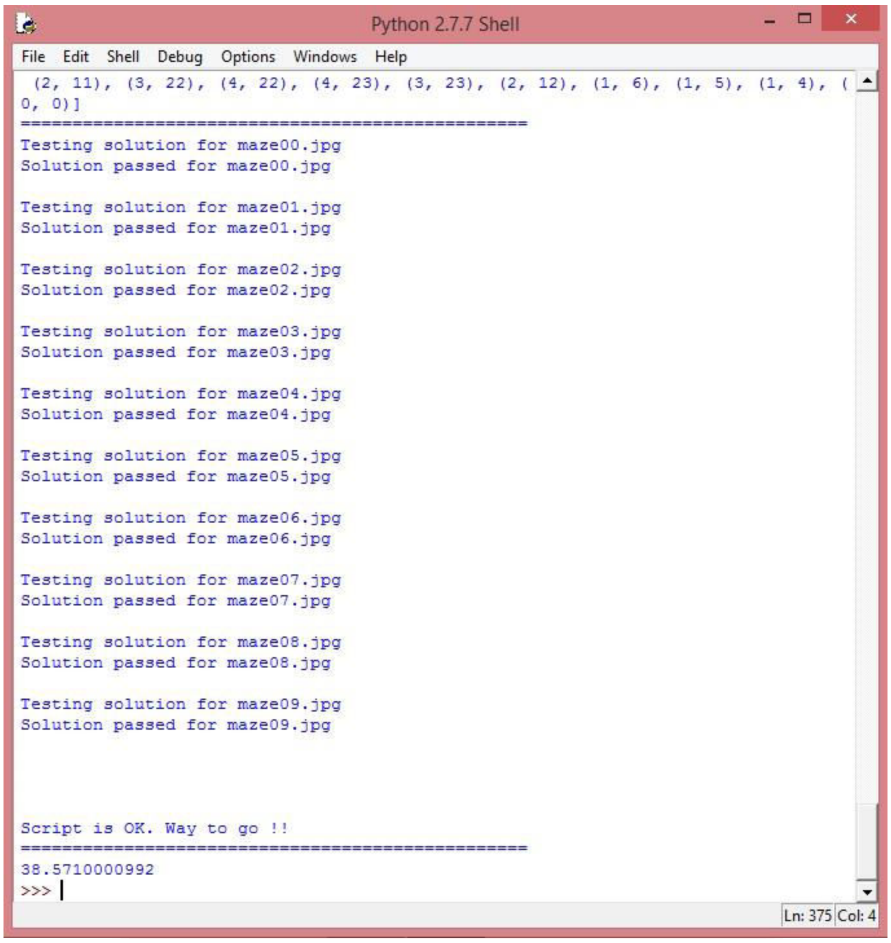


Testing the Solution

In the **Section-1 folder,** in addition to maze test images and ***section1.py*** there are three more files, namely the ***section1.txt*** file, ***hash.txt file*** and the ***TestSuite_1.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 ***section1.py*** file, open the ***TestSuite_1.py*** file and run it in the python shell. The output of **TestSuite_1** 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.

Now you can proceed to Section-2 of Task 2.

Way to go!!