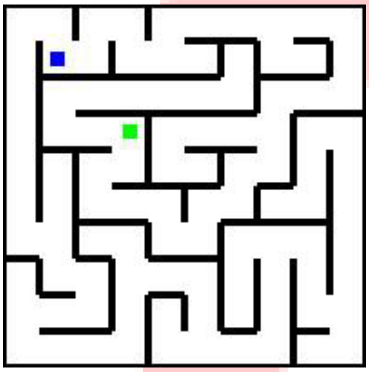
Task-1 – Practice

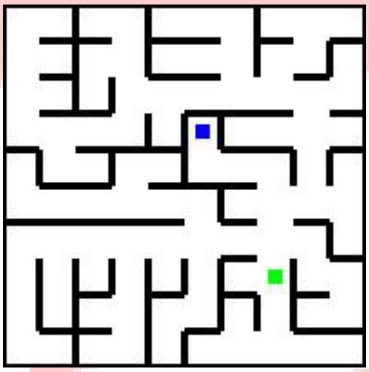
Section – 3

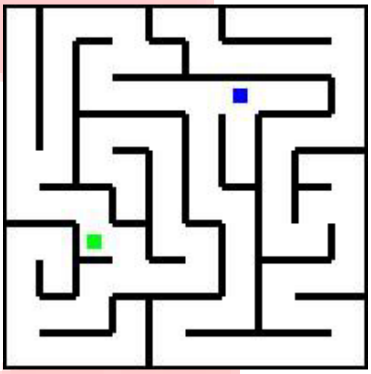
Robot path planning usually consists of challenges where a robot has to visit multiple locations in order to carry out tasks at those locations and then return to base. In such missions the need arises to optimize the path traversed between these locations.

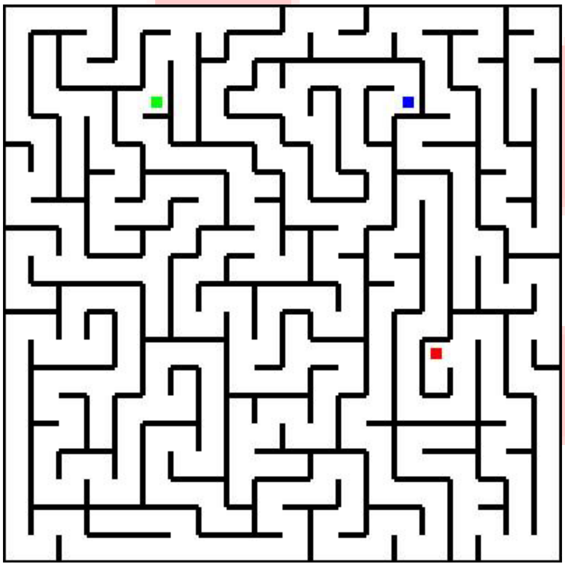
In this section we are representing various locations to be traversed as various coloured **markers** that are scattered randomly around the mazes.

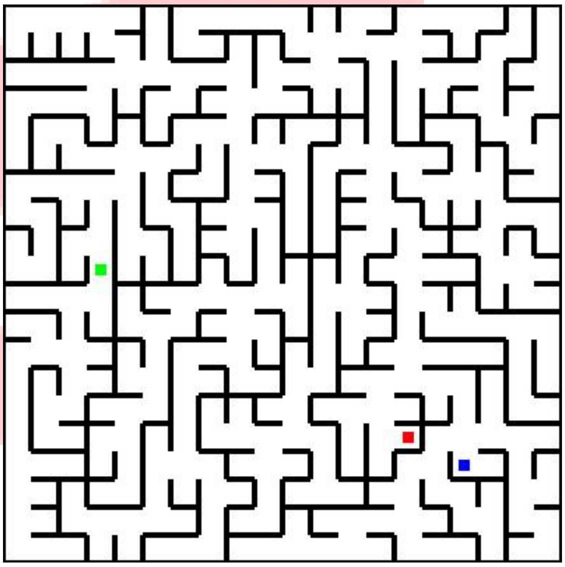
We can have maze images such as these:

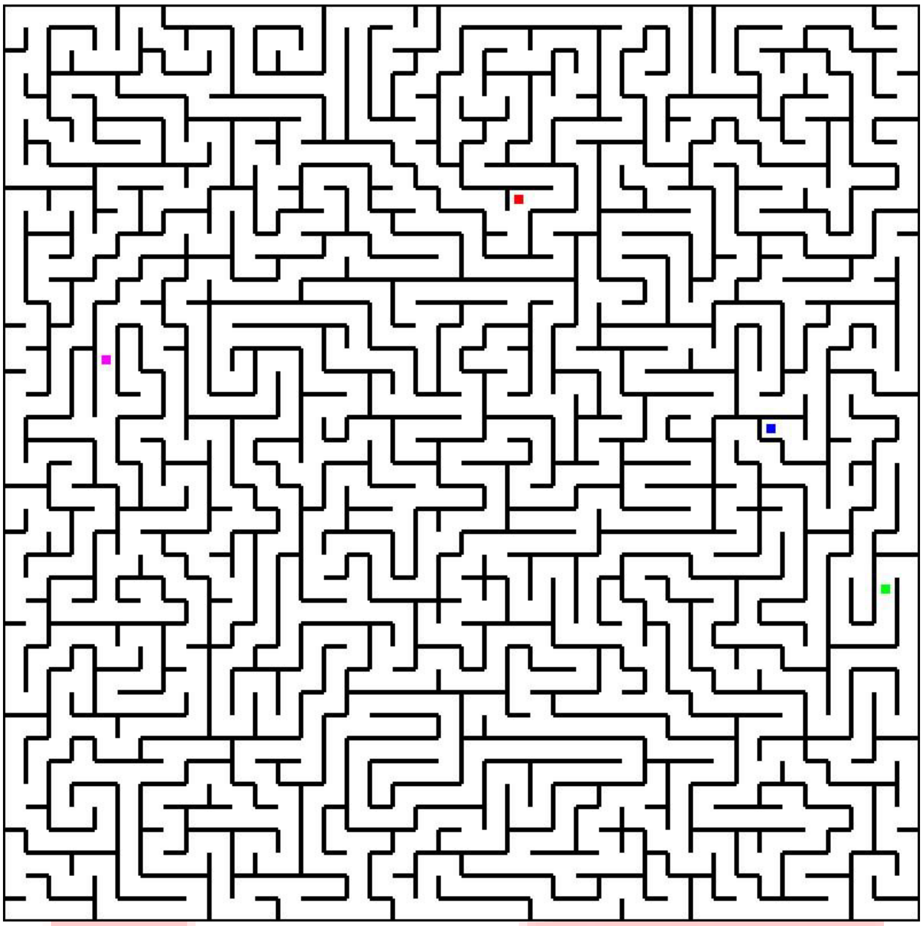












There can be upto 4 **coloured markers** in each maze image and all of them will be a distinct colour (**Blue**, **Green**, **Red** **or** **Pink**).
The **start coordinates** for maze will be the bottom left corner **(0, breadth – 1)** of the maze.
The **end coordinates** for maze will be the top right corner **(length - 1, 0)** of the maze.

The aim of this section is to find the optimum path between initial and final points while traversing to each coloured marker in between.

1. Open the **Task1_Practice folder**.
2. In the **Section-3 folder**, open the ***section3.py*** file.
3. Follow the instructions to modify functions in the ***section3.py*** file as given below:

MAIN

```
## Modify the filepath in this section to test your solution for
## different maze images.
if __name__ == "__main__":
    filePath = "maze00.jpg" ## Insert 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

```
## This is the main() function for the code, you are not
## allowed to change any statements in this part of the code.
## You are only allowed to change the arguments supplied in the
## findMarkers(), findOptimumPath() and colourPath() functions.

def main(filePath, flag = 0):
    imgHSV = readImageHSV(filePath)
    listOfMarkers = findMarkers(  )
    test  = str(listOfMarkers)
    imgBinary = readImageBinary(filePath)
    initial_point = ((len(imgBinary)/20)-1,0)
    final_point = (0, (len(imgBinary[0])/20) - 1)
    pathArray = findOptimumPath(     )
    print pathArray
    img = colourPath(imgBinary, pathArray)
    if __name__ == "__main__":
        return img
    else:
        if flag == 0:
            return pathArray
        elif flag == 1:
            return test + "\n"
        else:
            return img
```

This is the main() function which is being called in the previous section. The functions called in this main function() will be explained in detail. You are expected to understand what this snippet of code does, on your own. You can refer online resources to learn the python syntax such as given here: https://www.tutorialspoint.com/python/

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

The following section will explain the functions used in the ***section3.py*** script in detail:

FUNCTIONS

```
## Reads image in HSV format. Accepts filepath as input argument ##
and returns the HSV equivalent of the image.
def readImageHSV(filePath):
    ################      Add your Code here    #################


    ############################################################
    return hsvImg

## Reads image in binary format. Accepts filepath as input
## argument and returns the binary equivalent of the image.
def readImageBinary(filePath):
    ################        Add your Code here    ###############


    ############################################################
    return binaryImage
```

These functions receive the file path of the image and return the HSV and binary equivalent of the image respectively. Please refer to the image processing resources on the portal for information on how to implement this.

FUNCTIONS

```
## The findNeighbours function takes a maze image and row and
## column coordinates of a cell as input arguments and returns a ##
stack consisting of all the neighbours of the cell as
## output.
## Note :- Neighbour refers to all the adjacent cells one can
## traverse to from that cell provided only horizontal and vertical ##
traversal is allowed.
def findNeighbours(img,row,column):
    neighbours = []
    ################     Add your Code here     #################


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

## colourCell basically highlights the given cell by painting it
## with the given colourVal. Care should be taken that the
## function doesn't paint over the black walls and only paints ## the
empty spaces. This function returns the image with the ## painted cell.
You can change the colourCell() functions used ## in the previous
sections to suit your requirements.

def colourCell(   ):   ## Add required arguments here.
    ################     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

## Finds shortest path between two coordinates in the maze.
## Returns a set of coordinates from initial point to final
## point.
def findPath(): ## You can pass your own arguments in this space.
    ################        Add your Code here   ###############


    #############################################################
    return shortest
```

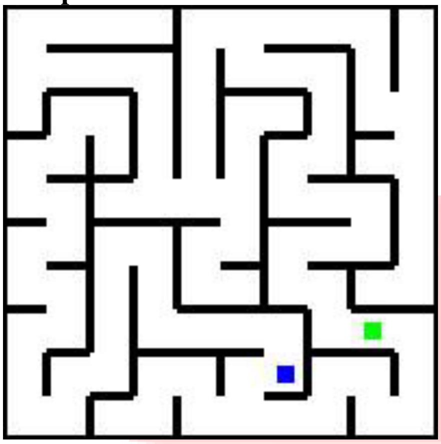
You can copy the functions definitions you used in section 2 here in these functions.

FUNCTIONS

```
## The findMarkers() function returns a list of coloured markers ## in
form of a python dictionary.
## For example if a blue marker is present at (3,6) and red
## marker is present at (1,5) then the dictionary is returned as ## :-
##          list_of_markers = { 'Blue':(3,6), 'Red':(1,5)}

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


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

The findMarkers() functions takes input arguments and detects and returns the coordinates of all the markers along with their colour.

Suppose we pass this image as test input:



Then the function will return the coordinates of blue and green markers as following python dictionary:

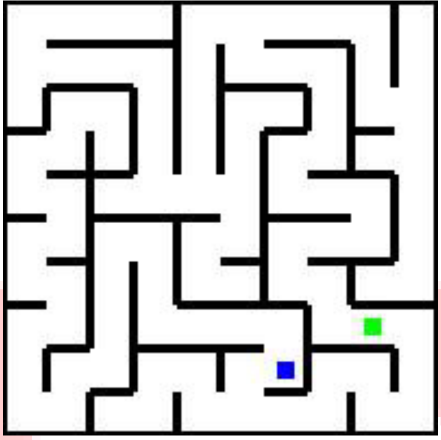
{ Blue : (8, 6) , Green : (7,8) }

FUNCTIONS

```
## The findOptimumPath() function returns a python list which
## consists of all paths that need to be traversed in order to ##
start from the bottom left corner of the maze, collect all ## the
markers by traversing to them and then traverse to the ## top right
corner of the maze.

def findOptimumPath(list_of_markers,     ):     ## You can pass your
own arguments in this space.
    path_array = []
    ################        Add your Code here    ###############


    ############################################################
    return path_array
```

The findOptimumPath() function returns a collection of paths in the variable *path_array* that start from the start coordinates and traverse to each coloured marker in the most optimum way possible and then return to the end coordinates.

Suppose we pass this image as test input:



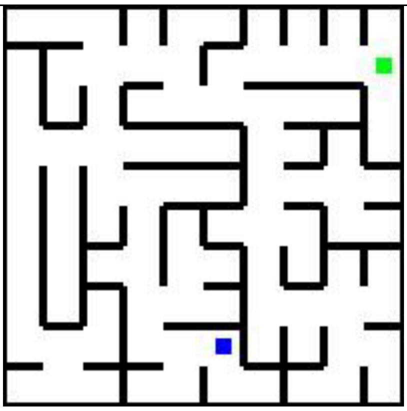
We specify the starting and ending coordinates as (9,0) and (0,9) respectively.
The output returned by this function will be :-
[[(9, 0), (9, 1), (8, 1), (8, 2), (7, 2), (6, 2), (5, 2), (5, 3), (6, 3), (7, 3), (7, 4), (7, 5), (7, 6), (8, 6)], [(8, 6), (8, 5), (9, 5), (9, 6), (9, 7), (8, 7), (8, 8), (9, 8), (9, 9), (8, 9), (7, 9), (7, 8)], [(7, 8), (7, 7), (6, 7), (6, 6), (5, 6), (5, 7), (5, 8), (4, 8), (4, 7), (4, 6), (3, 6), (3, 7), (2, 7), (1, 7), (1, 6), (1, 5), (0, 5), (0, 6), (0, 7), (0, 8), (1, 8), (2, 8), (2, 9), (1, 9), (0, 9)]]
The path highlighted in violet represents the path from start coordinates to blue marker. The path highlighted in orange represents path between blue and green markers. The sky blue highlighted path represents the path between green marker and end coordinates.

Note – The colours have been used just to show the different paths in the path array and are not part of the main code.

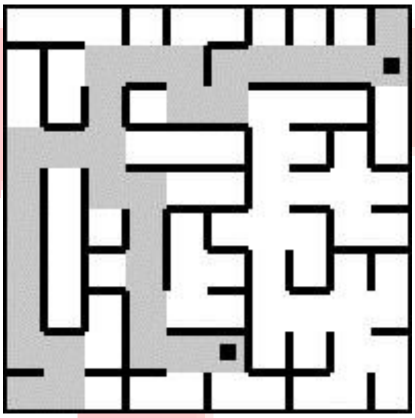
FUNCTIONS

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

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


    ##############################################################

    return img
```

The colour path function will highlight or paint the whole set of cell coordinates which need to be traversed.
Suppose we pass this image as test input:



The output for this image :-



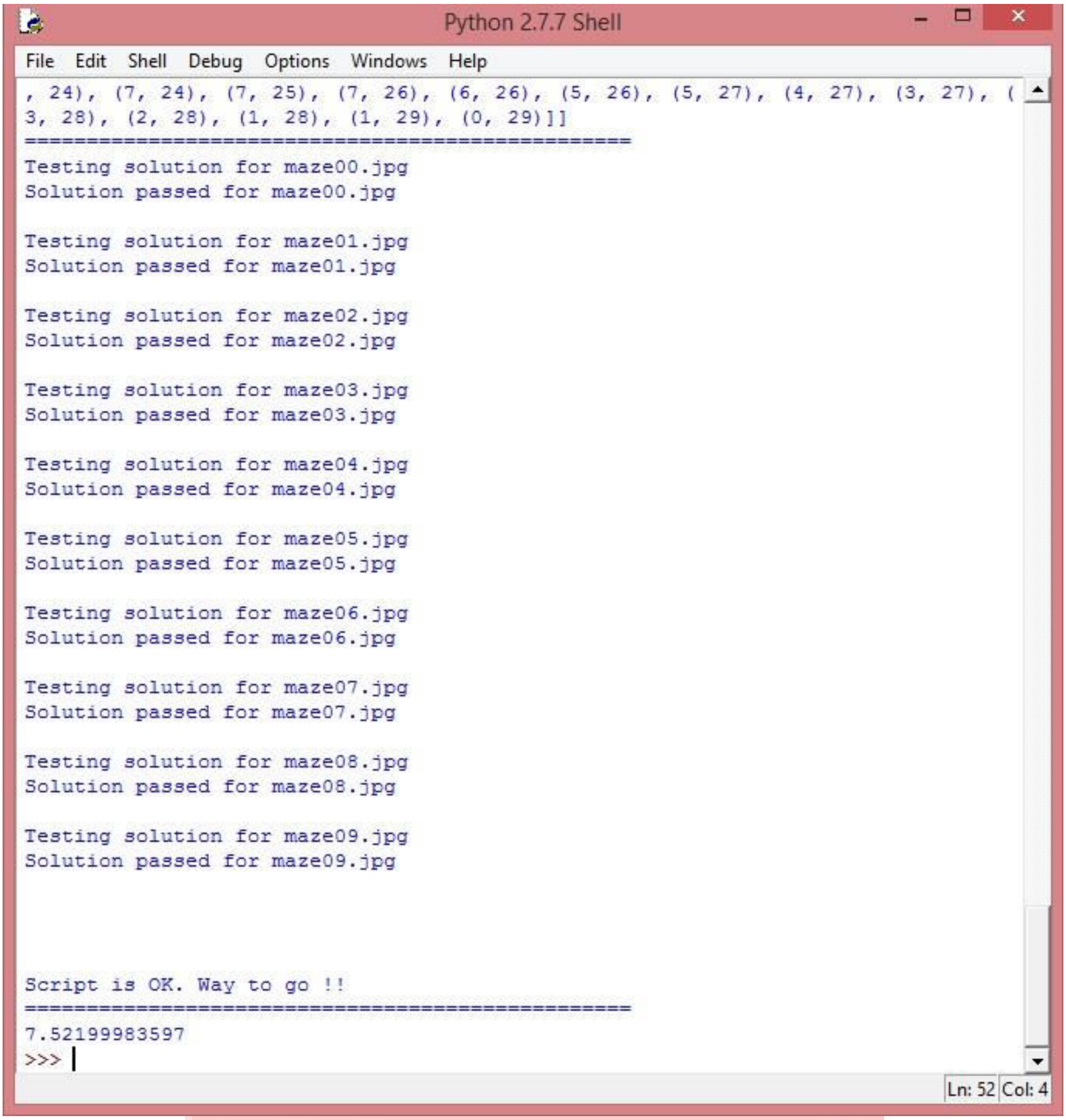
You are allowed to add additional utility functions in the space given in the ***section3.py*** file. The functions must be properly explained with comments for every step.

Testing the Solution

In the **Section-3 folder,** in addition to maze test images and ***section3.py*** there are three more files, namely the ***section3.txt*** file, ***hash.txt file*** and the ***TestSuite_3.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 ***section3.py*** file, open the ***TestSuite_3.py*** file and run it in the python shell. The output of **TestSuite_3** script should resemble the following screenshot:



```
, 24), (7, 24), (7, 25), (7, 26), (6, 26), (5, 26), (5, 27), (4, 27), (3, 27), (
3, 28), (2, 28), (1, 28), (1, 29), (0, 29)]]
=================================================
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 !!
=================================================
7.52199983597
>>> 
```

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

Congratulations. You are done with Task 1. Cheers !!