

# Terrain-Navigation

---

Shivansh Rao

Manish Aggarwal

Prabhakar Nawale

Deepank Grover

Department Of Electronics & Communication Engineering  
Delhi Technological University

## Abstract

Terrain navigation is one of the most important challenges in robotic technology. One of the many prerequisites for a robot rover is the ability to navigate a pre-explored terrain. These tasks involve complex challenges like image processing and path planning. Navigating through a pre-explored or unexplored area is very much like navigating through a maze of connected passages. This work will be an exercise in understanding these concepts.



Under the guidance of Prof. Madhusudan Singh  
Delhi Technological University

# CONTENTS

1. Introduction	2
2. Task -1	2-9
3. Task -2	10-13
4. Discussions	14
5. Future - Work	14
6. Bibliography	14

## 1. Introduction

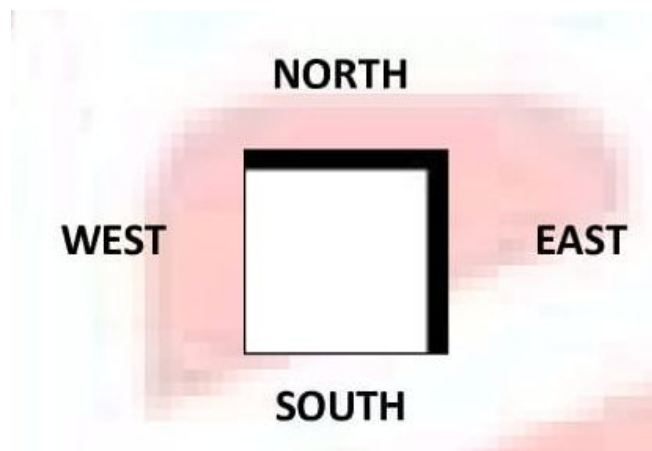
**Navigation** is a field of study that focuses on the process of monitoring and controlling the movement of a craft or vehicle from one place to another. The field of navigation includes four general categories: land navigation, marine navigation, aeronautic navigation, and space navigation.

It is also the term of the are used for the specialized knowledge used by navigators to perform navigation tasks. All navigational techniques involve locating the navigator's position compared to known locations or patterns. Navigation is a major problem when aiming for a totally autonomous system. Navigating through a pre-explored or unexplored area is very much like navigating through a maze of connected passages. In this project work we would be working on both square as well as circular mazes.

## 2. Task -1

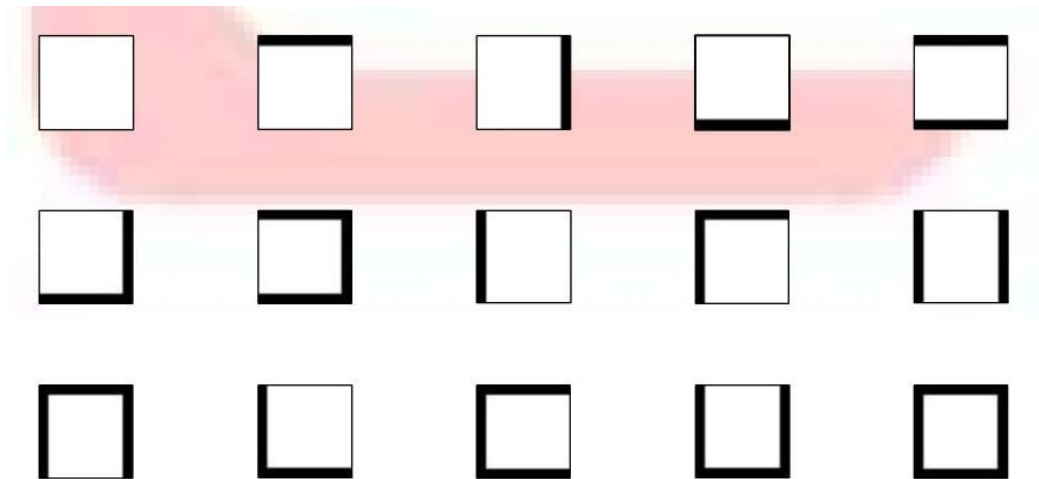
### Section -1

Let us look at the basic building element of a maze, the Cell.

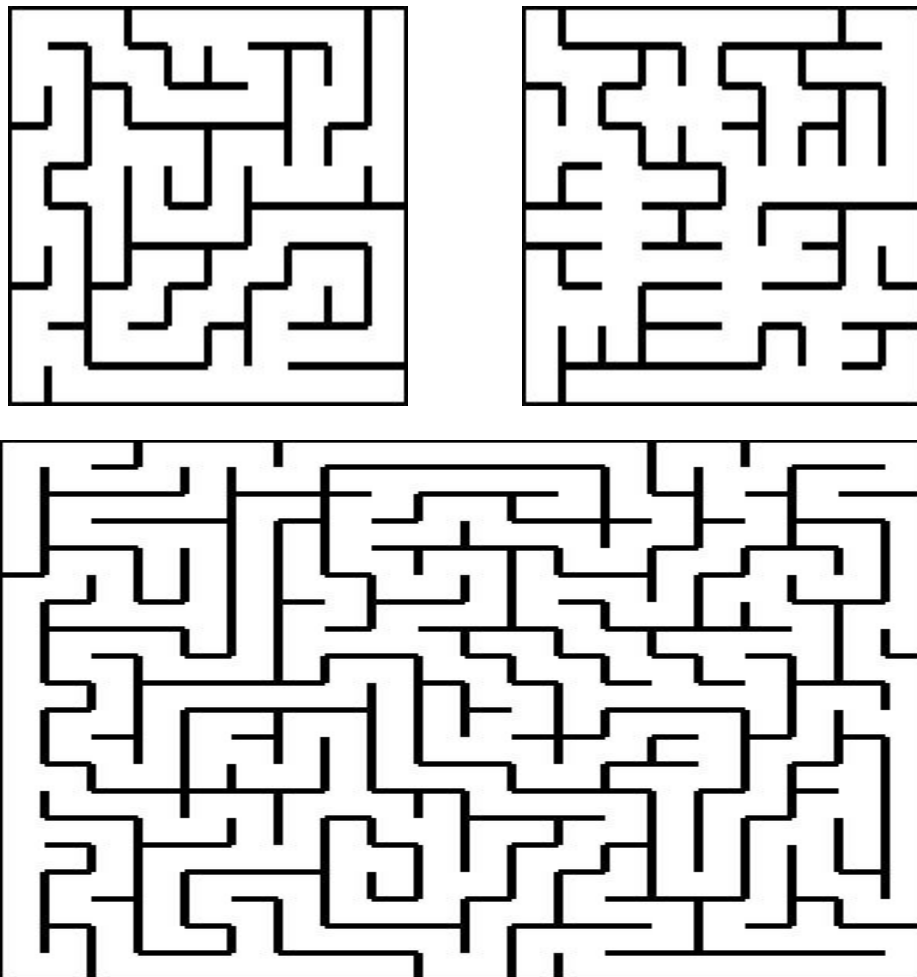


As shown in the figure above, the dark bands indicate presence of walls. The above cell has open passages towards WEST and SOUTH while the NORTH and EAST passages are closed by walls.

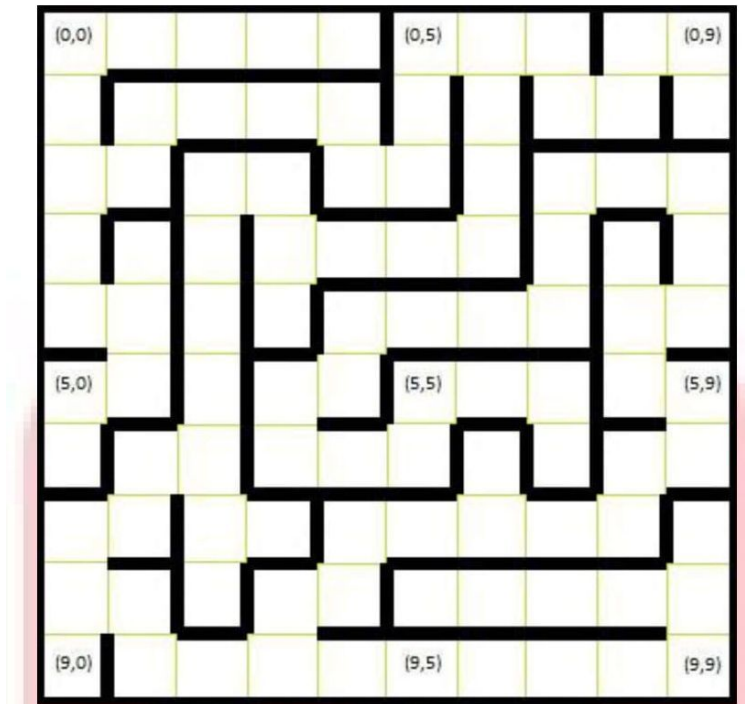
There are 15 more cells as shown below having the different possible wall configurations.



By combining cells having different wall configuration using special we can create mazes like these:



Cells are numbered in a maze using a coordinate system as given below:

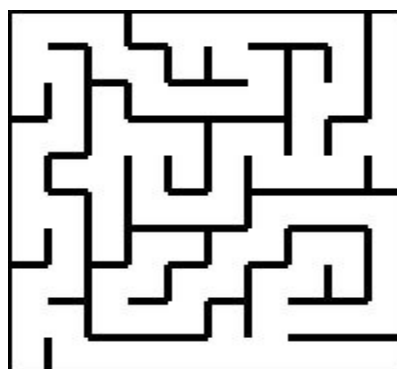


Given such a maze, the problem is to navigate from a START location in the maze to an END location, where the coordinates of START and END have been given.

The first step to solve such a problem is to examine cells in the maze image as well as analyze how these cells connect with each other.

We start by analyzing a maze and finding the neighbors of each of the specified cells. We create a findNeighbours function which accepts a binary image and the row and column coordinates of a particular cell as input and returns the neighbouring cells which are traversable from that particular cell.

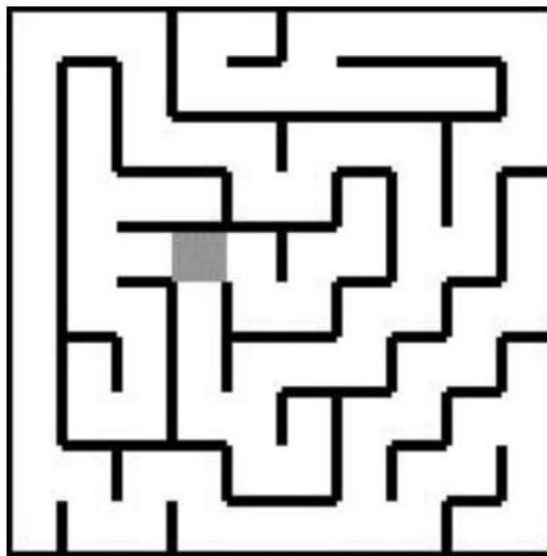
For example suppose we provide the following image as input to the function:



Assuming that we pass the other arguments as `findNeighbours(img, 0,0)` then this function will return `[(0,1) , (1,0)]` as output. If we use the same image and pass other arguments as `findNeighbours(img, 4,4)` which means the current cell coordinates are `(4,4)` then the function will return the value `[(4,3), (5,4)]` as output.

We then create a function which will highlight or 'paint' the cell whose coordinates have been specified by passing the appropriate row and column arguments. Suppose we use the following maze image:

Let us say `colourCell` is the name of the function for above mentioned purpose. Now if we pass the arguments as `colourCell(img, 4,3,150)`. The function will return the following output image

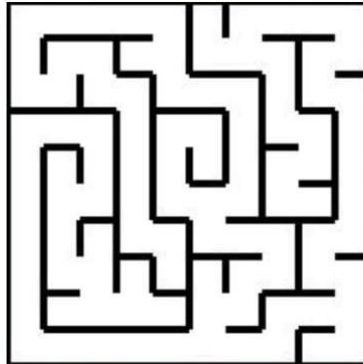


The last argument `colourVal` will be used to specify the intensity of colour we use to highlight the cell.

## EXPECTED OUTPUT OF PROGRAM:

### EXPECTED OUTPUT OF PROGRAM

For the *section1.py* script, if we provide the filepath of the following image in the space indicated:

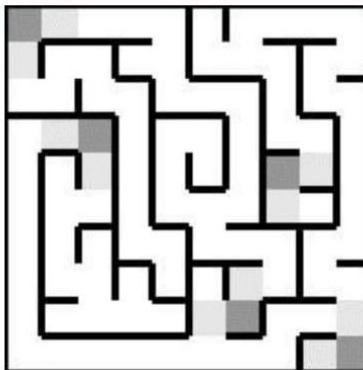


The expected output for the program will be:

```
>>>
```

```
[(0, 1), (1, 0)]  
[(8, 9), (9, 8)]  
[(4, 2), (3, 1)]  
[(4, 8), (5, 7)]  
[(7, 6), (8, 5)]
```

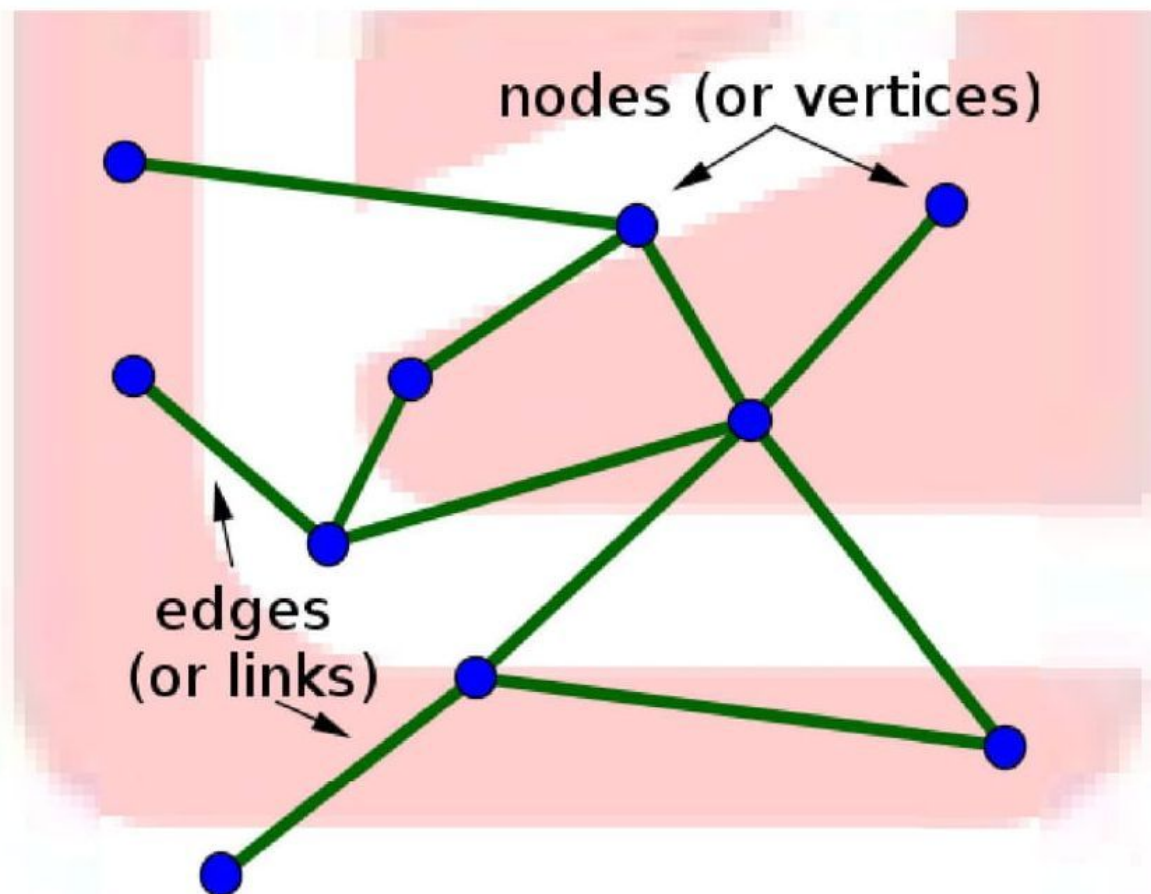
Output image will be:



## Section -2

Mazes in general are represented as undirected graphs so as to depict all traversable neighbors. In this section, you are supposed to find out how to represent a full maze in form of an undirected graph in python.

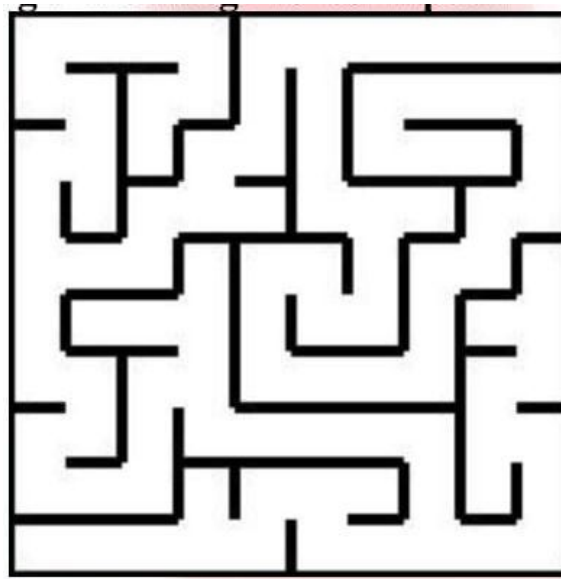
An undirected graph is a set of objects (called vertices or nodes) that are connected together, where all the edges are bidirectional. An undirected graph is sometimes called an undirected network. In contrast, a graph where the edges point towards specified directions is called a directed graph. When drawing an undirected graph, the edges are typically drawn as lines between pairs of nodes, as illustrated in the following figure:





In this part we need to create a function which builds the graph. Suppose it is `buildGraph( )` function which accepts some input arguments and returns the maze in the form of an undirected graph.

Suppose we are using the following maze image as test input :

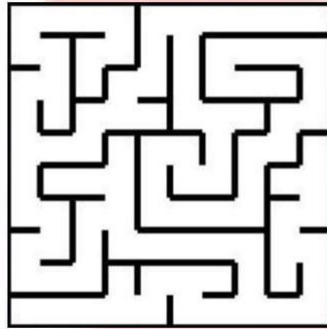


The output graph for this function will be similar to this :-

$\{ (0,0) : [ (1,0) , (0,1) ] , (0,1) : [ (0,0) , (0,2) ] , (0,2) : [ (0,1) , (0,3) ] , (0,3) : [ (0,2) , (1,3) ] , (0,4) : [ (1,4) , (0,5) ] \dots\dots\dots \}$

### EXPECTED OUTPUT OF PROGRAM

For the *section2.py* script, if we provide the filepath of the following image in the space indicated:

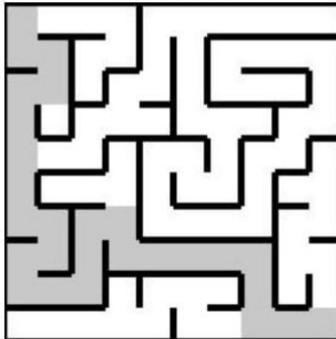


The output for this program should be:

```
>>>
```

```
[(0, 0), (1, 0), (1, 1), (2, 1),  
(2, 0), (3, 0), (4, 0), (5, 0),  
(6, 0), (6, 1), (7, 1), (7, 0),  
(8, 0), (8, 1), (8, 2), (7, 2),  
(6, 2), (6, 3), (7, 3), (7, 4),  
(7, 5), (7, 6), (7, 7), (8, 7),  
(9, 7), (9, 8), (9, 9)]
```

The output image should be:



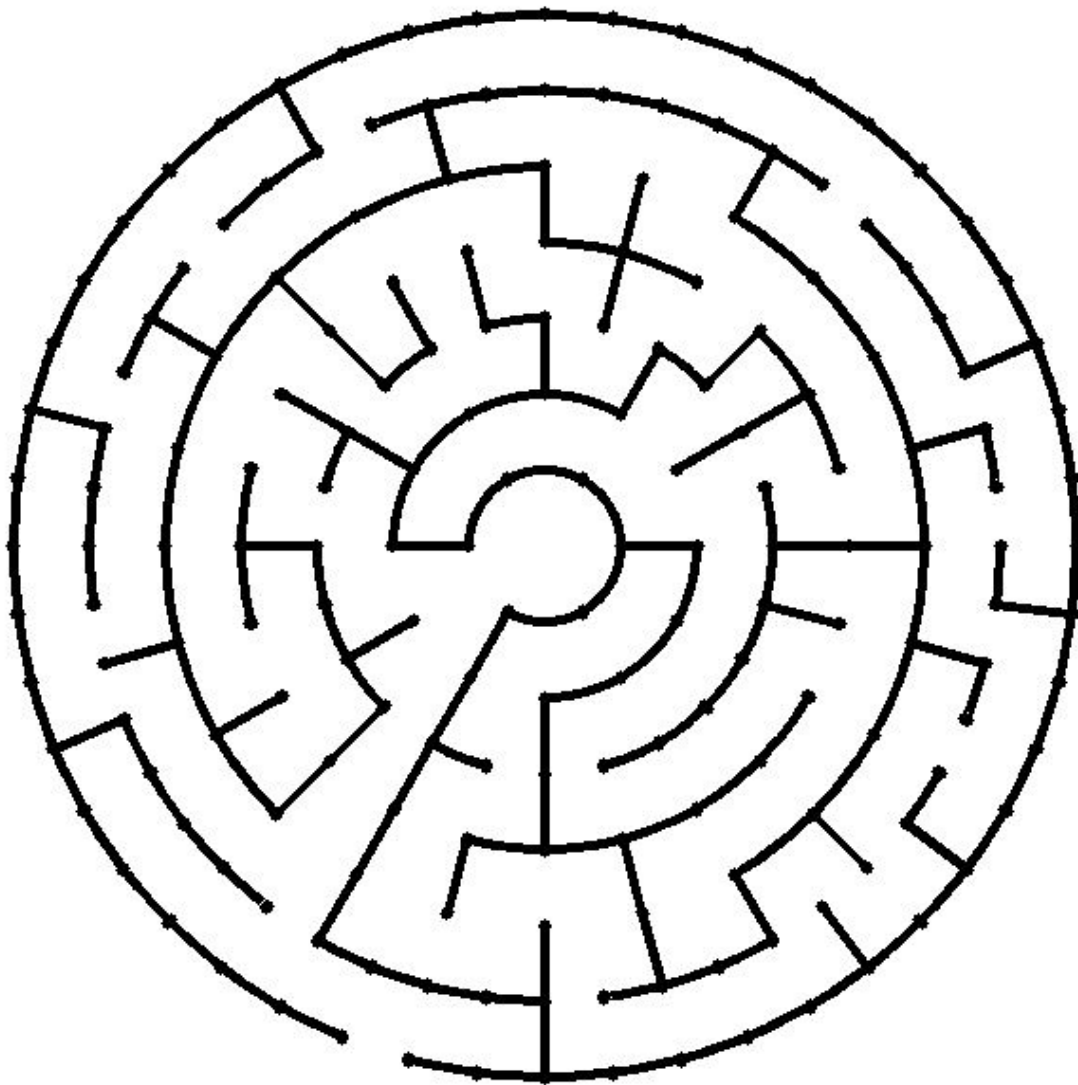
## Task -2 :

### Section -1 :

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

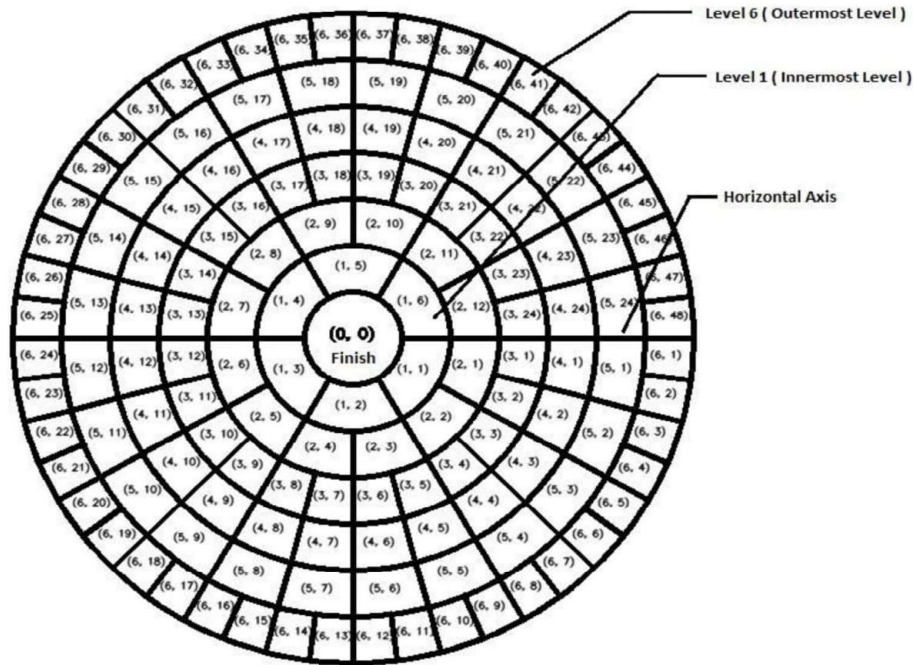
Theta mazes are composed of concentric circles of passages with the following properties: Centre can be either Start or Finish. If Start is in the centre, Finish is on the Outer Edge and vice versa. Cells typically have 4-5 possible passage connections.

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 Figure below.



## Creating a Theta Maze

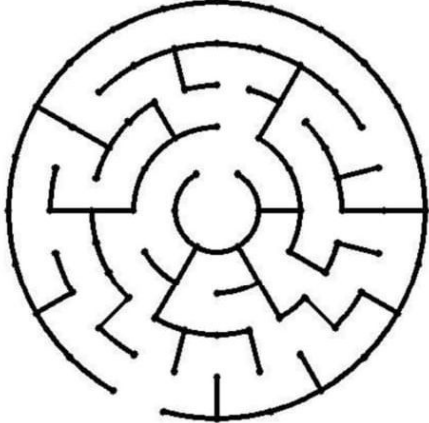
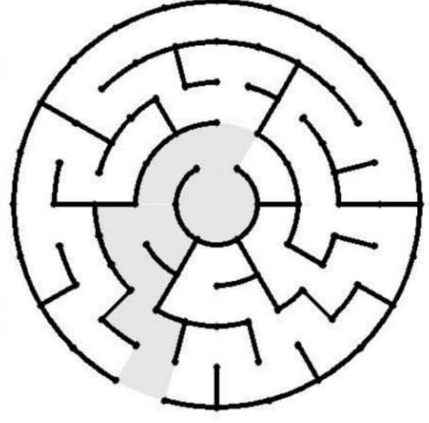
In this section, we will learn about how a Theta maze is constructed. We start with a Theta Grid. An example is shown in Figure below.



Considering the above Figure,

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  $360^\circ/6 = 60^\circ$ . Each cell's coordinates are represented using the following convention: (level, cell\_number)

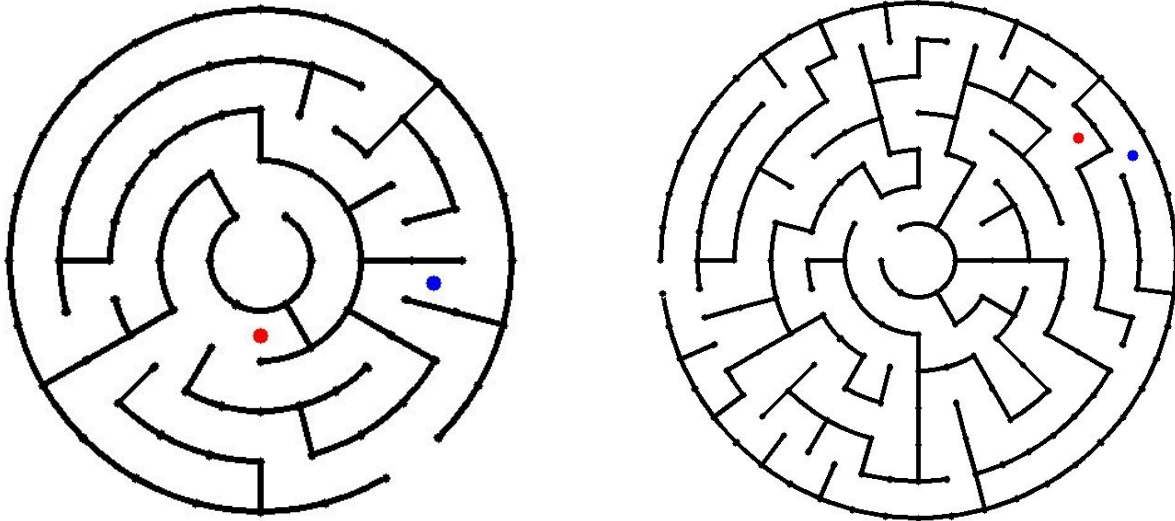
After following the similar procedure as that in Task -1 , the expected output of the program shall be as follows.

OUTPUT OF PROGRAM
<p>If we provide the filepath for the following image in the space indicated:</p> 
<p>The output for this program should be:</p> <pre>&gt;&gt;&gt; [(4, 8), (3, 8), (3, 9), (2, 5), (2, 6), (1, 3), (1, 4), (1, 5), (0, 0)]</pre>
<p>The output image should be:</p> 

## 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). 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.

Again a similar approach until now will be followed here to get the desired results.

**Discussions :**

As we have seen it is quite easy for solving navigation of the mazes be it either square shaped or theta shaped. It can be concluded that square mazes are quite simpler in comparison to theta shaped mazes. Since navigation is an important challenge, navigating through a pre-explored or unexplored area is very much like navigating through a maze of connected passage.

**Future Work :**

The future work for this project shall be applying this software implementation on hardware and test it for real-time application for autonomous systems.

**Bibliography :**

1. Reference material supplied by IIT-BOMBAY.