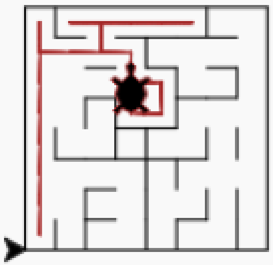
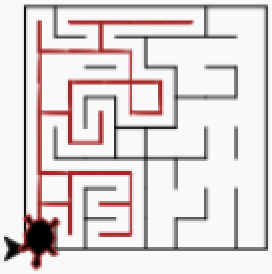
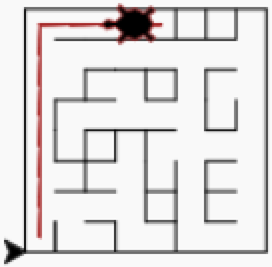
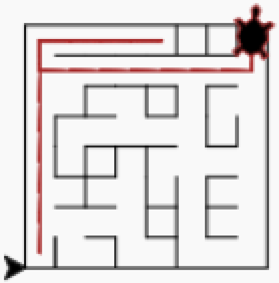
Development of Path Planning Simulation Software for Mobile Robots









Team Based Online Project (TOP) @ Comtel
Dec 2020 to Jan 2021

Team Number 2

Sawant Anand
U.I.E.T, Punjab Univ.
Chandigarh
sawantanand310@gmail.com

Soumya Ghosh
NIT Durgapur
Durgapur
soumya0341@gmail.com

Bharat Dadwaria
JNU Delhi
New Delhi
bkd0385@gmail.com

Rajeevlochana G Chittawadigi
Amrita Vishwa Vidyapeetham
Bengaluru
rg_chittawadigi@blr.amrita.edu

Contents

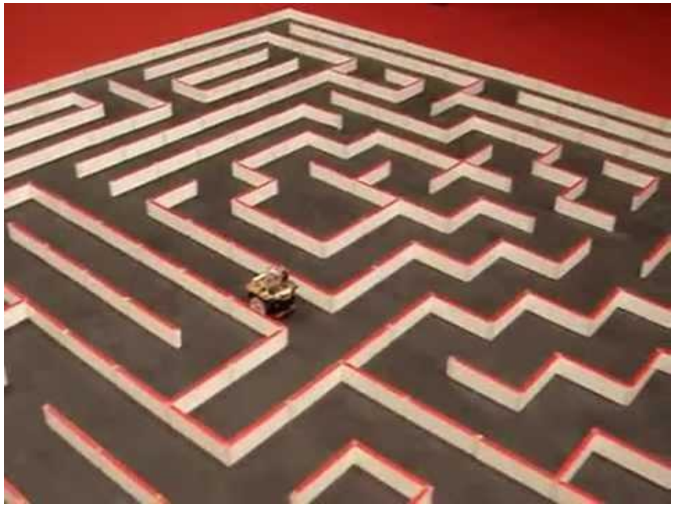
- Introduction
- Objectives
- Flowchart
- Literature Review-Mazes Algorithms
- Logic of Maze Generation
- Logic of Maze Solution
- Video of Demonstration
- Social Implication
- Conclusion
- References

Introduction

Mobile robots are used for various applications such as:

- Movement of material
- Scanning of rooms or regions, etc.

Before developing a physical prototype of a mobile robot, **it is important to simulate the motion of the robot for any given application**.



Micromouse Competition at IIT Bombay (TechFest)

Objectives

- **Develop a simulation environment** that can be used to perform **path planning of wheeled mobile robots**, assuming no slippage for simpler implementation.

- **Develop an algorithm for random maze generation** using turtlebot and python coding

- **Develop an algorithm to traverse the generated maze** using turtlebot and python coding using left hand wall following algorithm.

Literature Review-Mazes Algorithms

- Types of Solver based on view:
  - There are two types of solver based on view:
    - The random mouse, wall follower, Pledge, and Trémaux's algorithms are **designed to be used inside the maze by a traveller with no prior knowledge of the maze**
    - The dead-end filling and shortest path algorithms are designed to be used by a **person or computer program that can see the whole maze at once**.

Popular Algorithms:

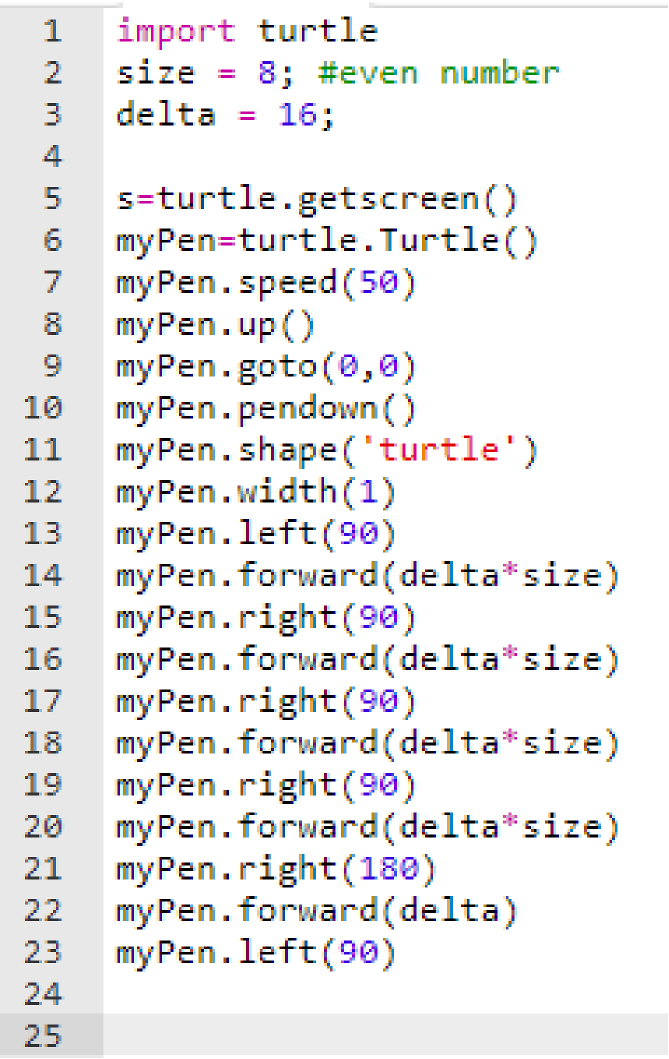
- Wall Follower
- Pledge Algorithm
- Trémaux's Algorithm
- Dead-end filling
- Recursive Algorithm
- Maze-Routing Algorithm
- Shortest path algorithm
- Backtracking Algorithm

Our Implementation

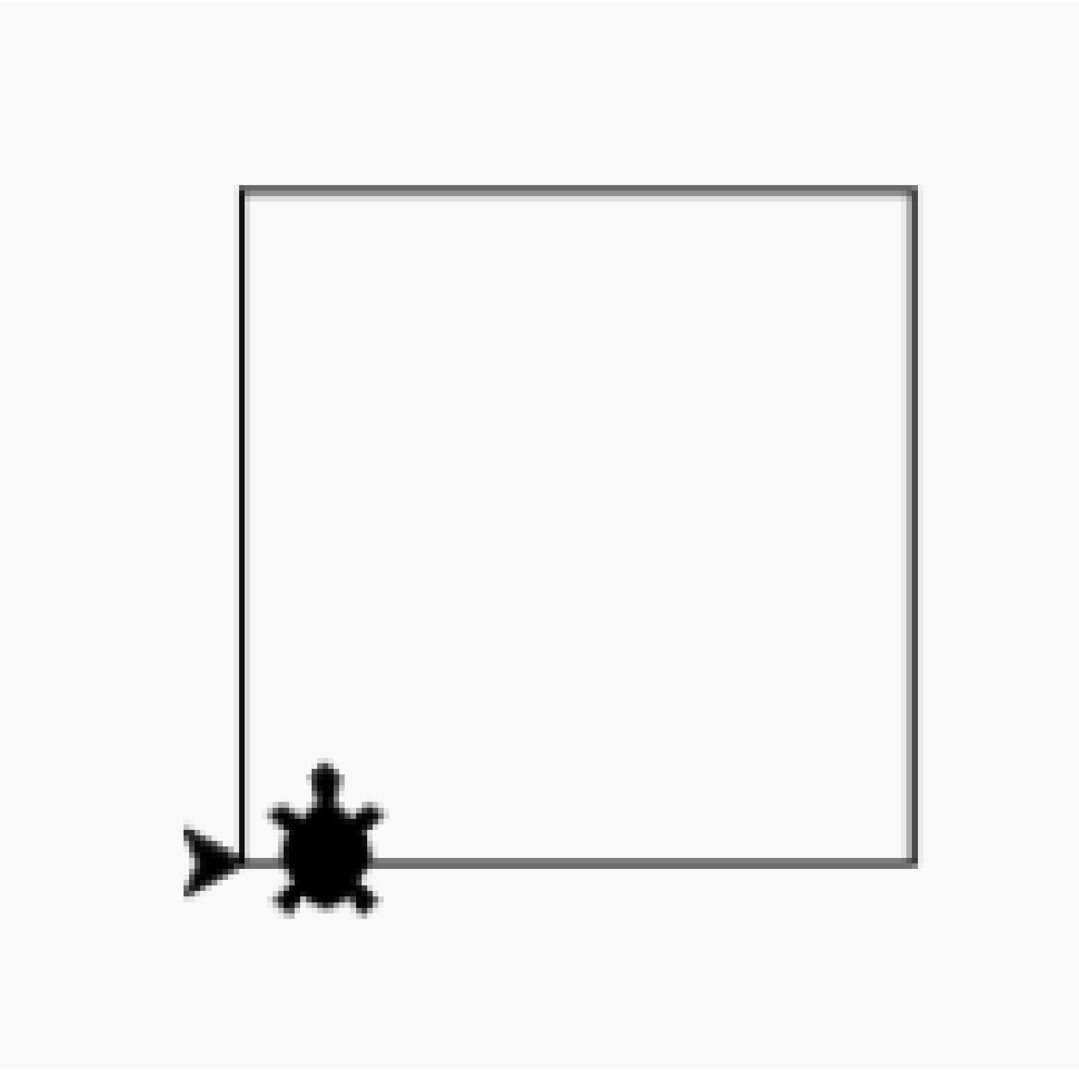
- Python Programming Language
- https://trinket.io/python/
  - Online Compiler

- Turtle programming
  - Easy to draw lines and animate motion

Sample Program

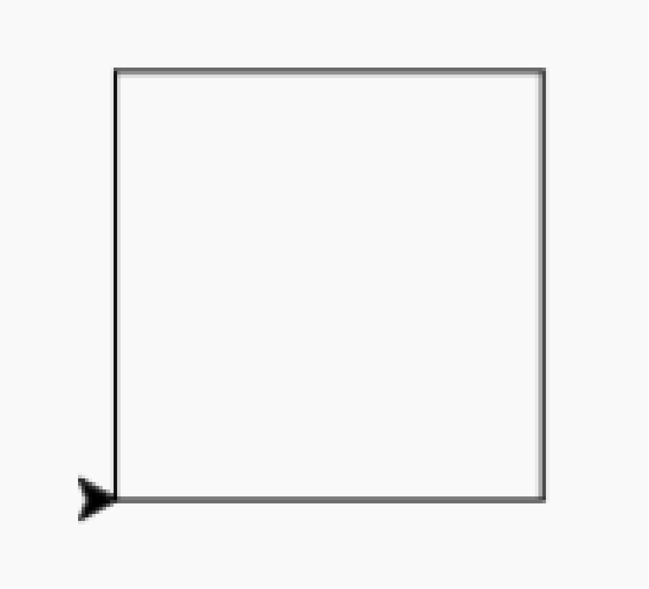


```python
import turtle
size = 8; #even number
delta = 16;

s=turtle.getscreen()
myPen=turtle.Turtle()
myPen.speed(50)
myPen.up()
myPen.goto(0,0)
myPen.pendown()
myPen.shape('turtle')
myPen.width(1)
myPen.left(90)
myPen.forward(delta*size)
myPen.right(90)
myPen.forward(delta*size)
myPen.right(90)
myPen.forward(delta*size)
myPen.right(90)
myPen.forward(delta*size)
myPen.right(180)
myPen.forward(delta)
myPen.left(90)
```



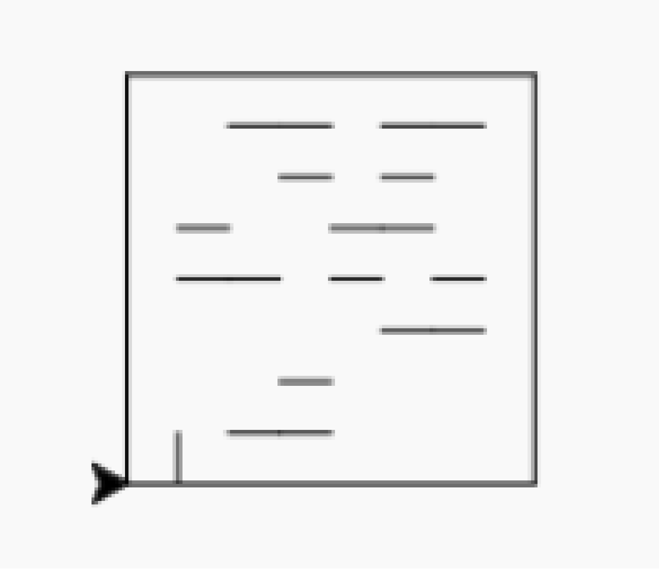
Maze Generation Program

- The maze is generated in 3 Distinct Steps:
- **First, given the size of the maze, walls of the maze using solid lines is created.**
- Second, using left-&-right motion, horizontal inner walls of the maze are generated, using a random function to generate a cell wall(0) or space(1)
- **Third, using up-&-down motion, vertical inner walls of the maze are generated, using a random function to generate a cell wall(0) or space(1)**
- Data of the walls are stored in two 2-D matrices in the binary form, where 1=Space and 0=Wall
- Disadvantages: This Algorithm may create a solvable or unsolvable maze



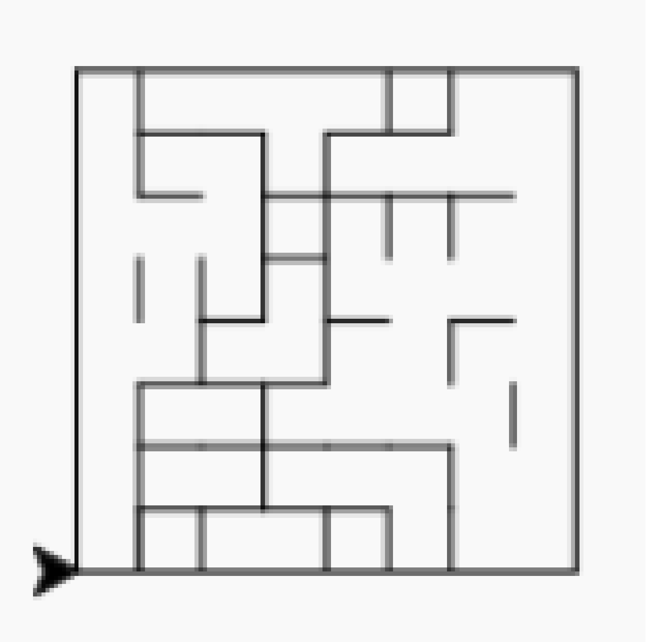
Maze Generation Program

- The maze is generated in 3 Distinct Steps:
- First, given the size of the maze, walls of the maze using solid lines is created.
- **Second, using left-&-right motion, horizontal inner walls of the maze are generated, using a random function to generate a cell wall(0) or space(1)**
- Third, using up-&-down motion, vertical inner walls of the maze are generated, using a random function to generate a cell wall(0) or space(1)
- Data of the walls are stored in two 2-D matrices in the binary form, where 1=Space and 0=Wall
- Disadvantages: This Algorithm may create a solvable or unsolvable maze



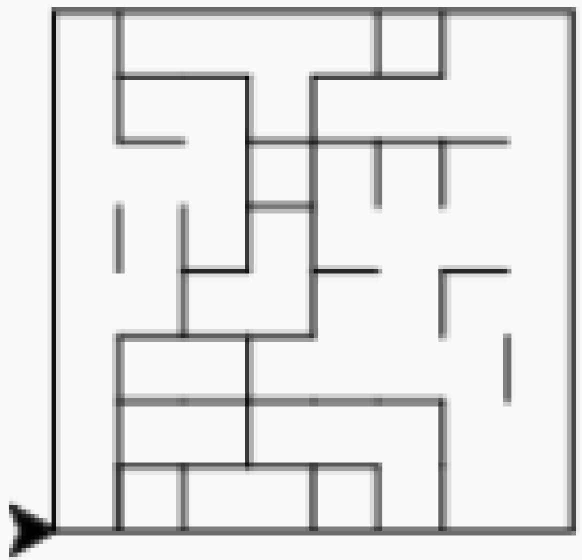
Maze Generation Program

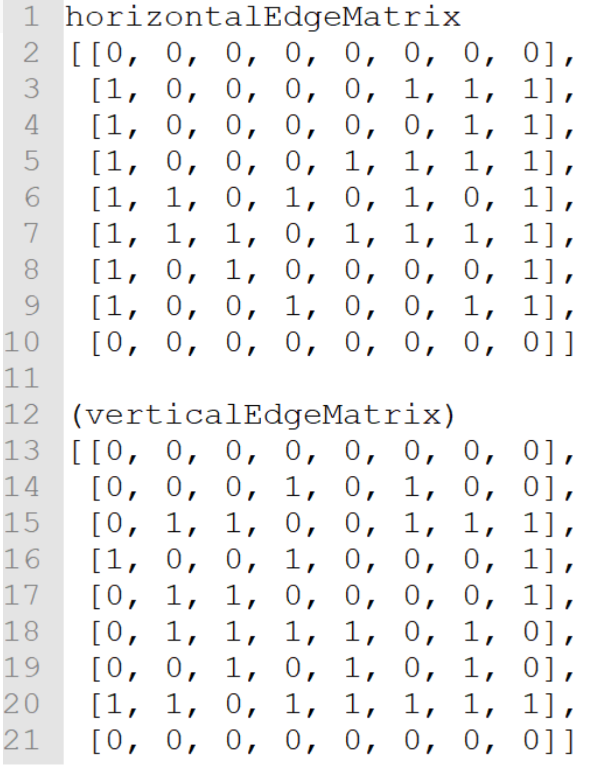
- The maze is generated in 3 Distinct Steps:
- First, given the size of the maze, walls of the maze using solid lines is created.
- Second, using left-&-right motion, horizontal inner walls of the maze are generated, using a random function to generate a cell wall(0) or space(1)
- **Third, using up-&-down motion, vertical inner walls of the maze are generated, using a random function to generate a cell wall(0) or space(1)**
- Data of the walls are stored in two 2-D matrices in the binary form, where 1=Space and 0=Wall
- Disadvantages: This Algorithm may create a solvable or unsolvable maze



Maze Generation Program

- The maze is generated in 3 Distinct Steps:
- First, given the size of the maze, walls of the maze using solid lines is created.
- Second, using left-&-right motion, horizontal inner walls of the maze are generated, using a random function to generate a cell wall(0) or space(1)
- Third, using up-&-down motion, vertical inner walls of the maze are generated, using a random function to generate a cell wall(0) or space(1)
- **Data of the walls are stored in two 2-D matrices in the binary form, where 1=Space and 0=Wall**
- Disadvantages: This Algorithm may create a solvable or unsolvable maze

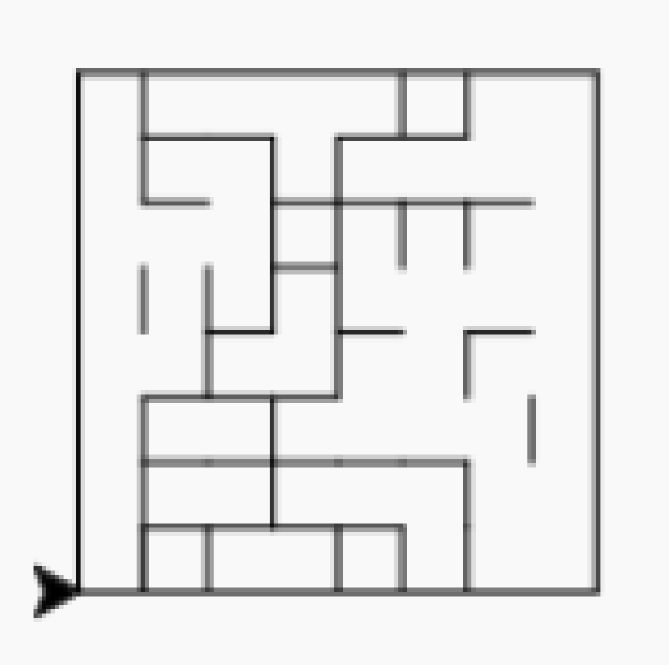


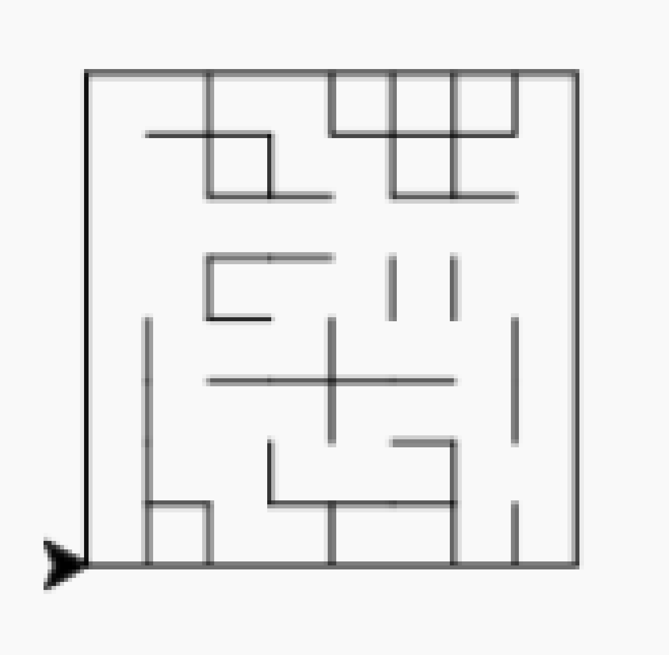


```
horizontalEdgeMatrix
[[0, 0, 0, 0, 0, 0, 0, 0],
 [1, 0, 0, 0, 0, 1, 1, 1],
 [1, 0, 0, 0, 0, 0, 1, 1],
 [1, 0, 0, 0, 1, 1, 1, 1],
 [1, 1, 0, 1, 0, 1, 0, 1],
 [1, 1, 1, 0, 1, 1, 1, 1],
 [1, 0, 1, 0, 0, 0, 0, 1],
 [1, 0, 0, 1, 0, 0, 1, 1],
 [0, 0, 0, 0, 0, 0, 0, 0]]

(verticalEdgeMatrix)
[[0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 1, 0, 0],
 [0, 1, 1, 0, 0, 1, 1, 1],
 [1, 0, 0, 1, 0, 0, 0, 1],
 [0, 1, 1, 0, 0, 0, 0, 1],
 [0, 1, 1, 1, 1, 0, 1, 0],
 [0, 0, 1, 0, 1, 0, 1, 0],
 [1, 1, 0, 1, 1, 1, 1, 1],
 [0, 0, 0, 0, 0, 0, 0, 0]]
```

Maze Generation Program

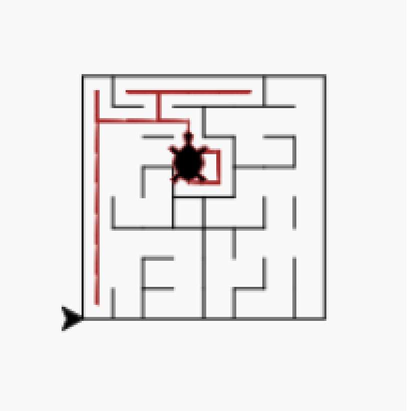
- The maze is generated in 3 Distinct Steps:
- First, given the size of the maze, walls of the maze using solid lines is created.
- Second, using left-&-right motion, horizontal inner walls of the maze are generated, using a random function to generate a cell wall(0) or space(1)
- Third, using up-&-down motion, vertical inner walls of the maze are generated, using a random function to generate a cell wall(0) or space(1)
- Data of the walls are stored in two 2-D matrices in the binary form, where 1=Space and 0=Wall
- **Disadvantages: This Algorithm may create a solvable or unsolvable maze**

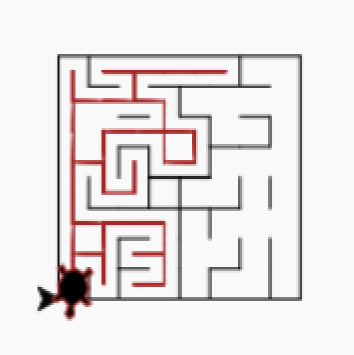


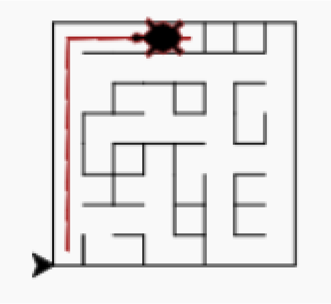


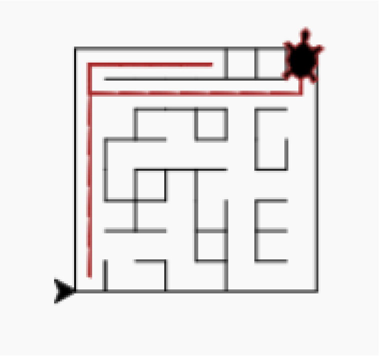
Maze Solving Logic: Left Wall Follower

- Here, the bot **starts from bottom-left corner** and tries to **reach top-right corner**.
- The maze is viewed as a grid with 4 walls/spaces depending on the orientation of the bot : left, right, front, back.
- The bot checks the data of the current cell-walls and makes the a decision based on the following conditions:
    - It will try to keep a wall on left.
    - If in a cell, there is no left wall, it will turn left and move 1 cell forward
    - Else-If it has front & left wall only, then it will turn right and move 1 cell forward
    - Else-If, it has front, left & right wall, then it will turn around (180°) and move 1 cell forward
    - Else,(only left wall) then it will move 1 cell forward
- Either it will reach the destination cell, or rotate the maze in a loop.

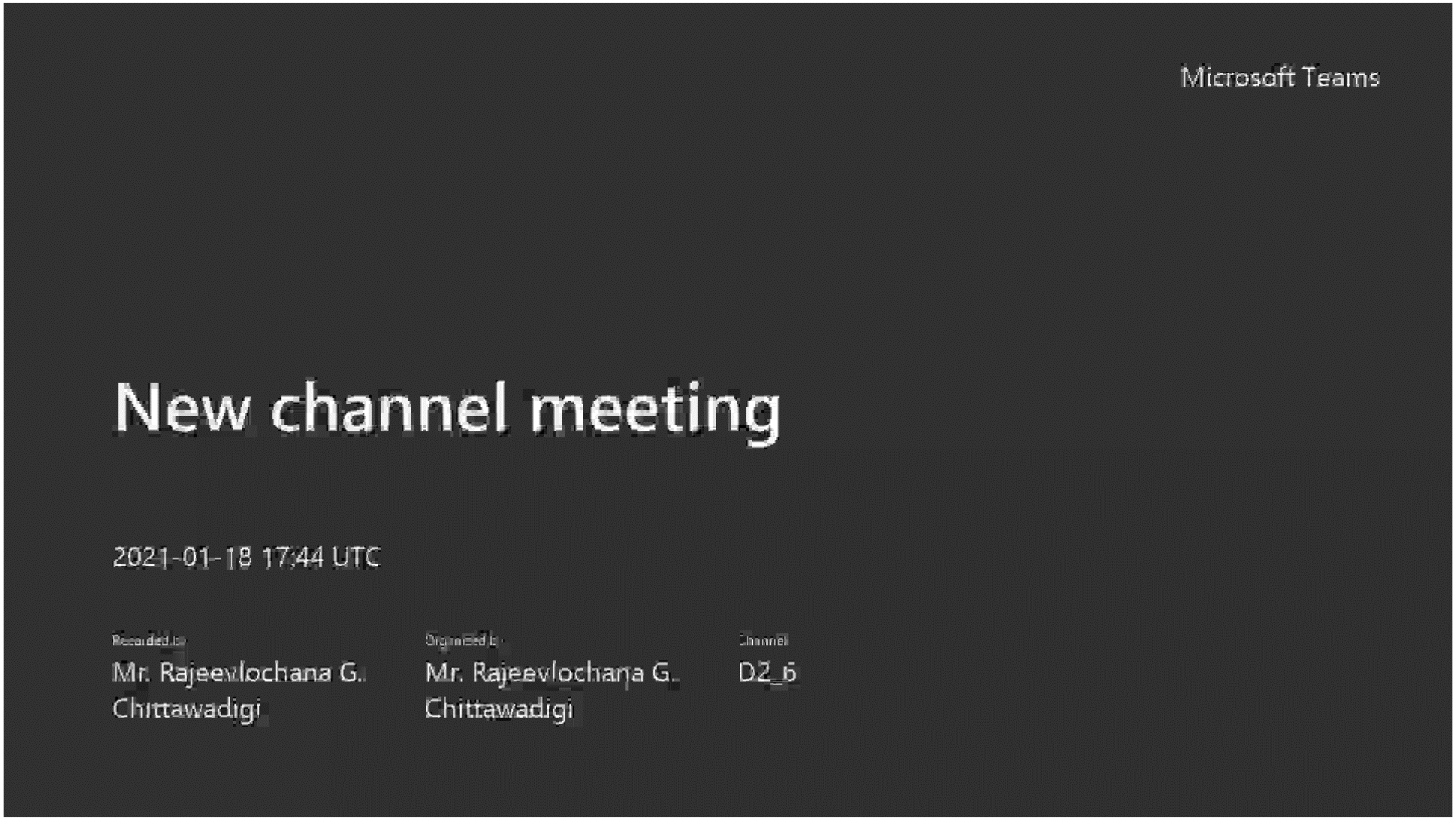








Video of Demonstration



Microsoft Teams

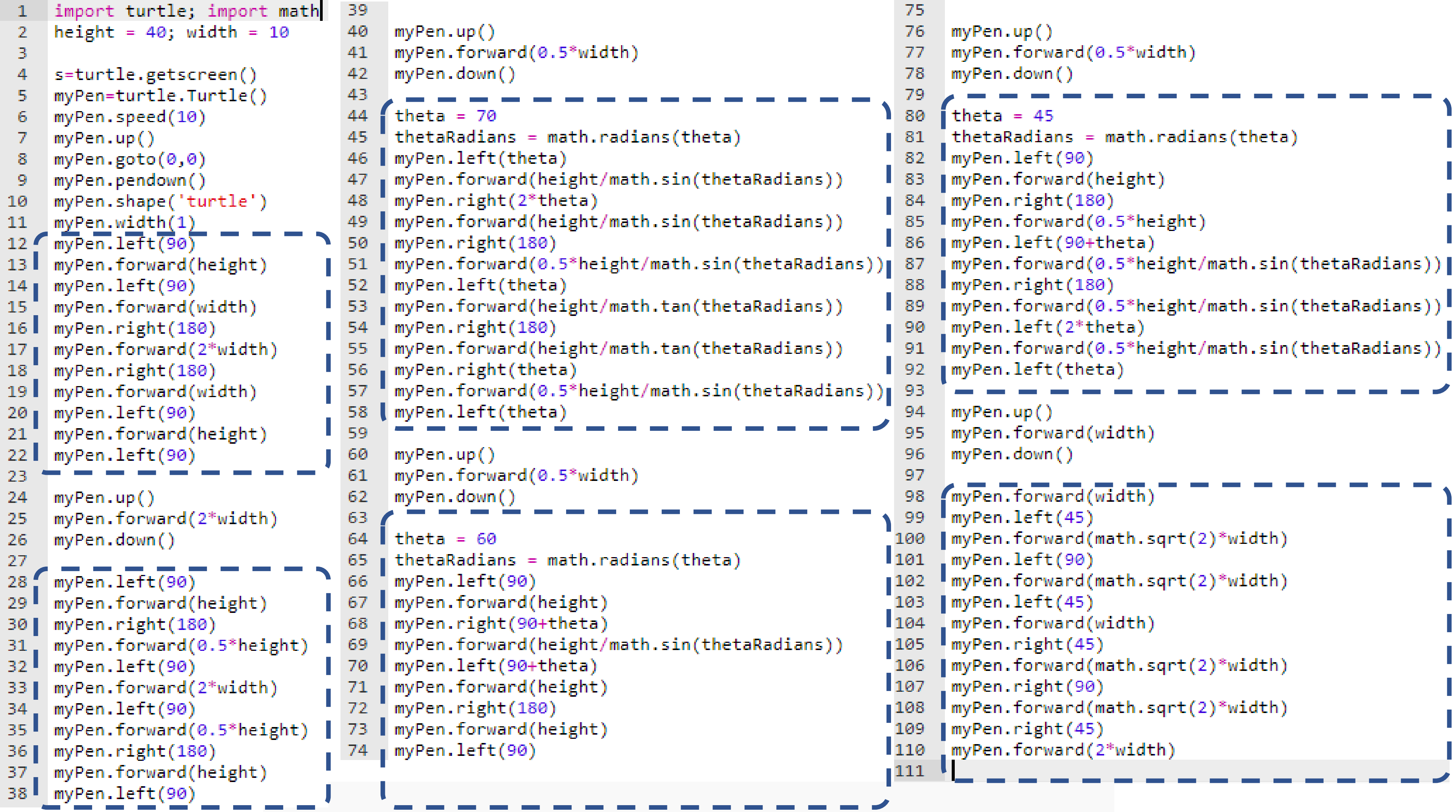
New channel meeting

2021-01-18 17:44 UTC

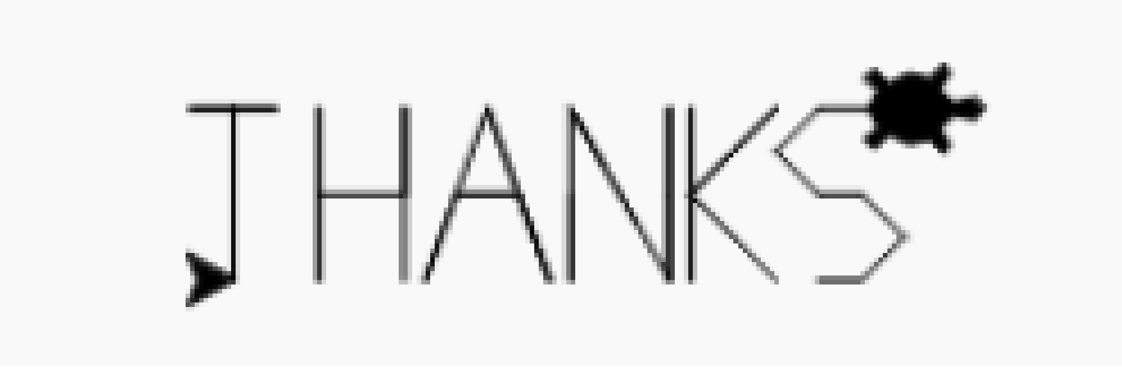
Recorded by
Mr. Rajeevlochana G. Chittawadigi

Organized by
Mr. Rajeevlochana G. Chittawadigi

Channel
D2_6



```
import turtle; import math
height = 40; width = 10

s=turtle.getscreen()
myPen=turtle.Turtle()
myPen.speed(10)
myPen.up()
myPen.goto(0,0)
myPen.pendown()
myPen.shape('turtle')
myPen.width(1)
myPen.left(90)
myPen.forward(height)
myPen.left(90)
myPen.forward(width)
myPen.right(180)
myPen.forward(2*width)
myPen.right(180)
myPen.forward(width)
myPen.left(90)
myPen.forward(height)
myPen.left(90)

myPen.up()
myPen.forward(2*width)
myPen.down()

myPen.left(90)
myPen.forward(height)
myPen.right(180)
myPen.forward(0.5*height)
myPen.left(90)
myPen.forward(2*width)
myPen.left(90)
myPen.forward(0.5*height)
myPen.right(180)
myPen.forward(height)
myPen.left(90)

myPen.up()
myPen.forward(0.5*width)
myPen.down()

theta = 70
thetaRadians = math.radians(theta)
myPen.left(theta)
myPen.forward(height/math.sin(thetaRadians))
myPen.right(2*theta)
myPen.forward(height/math.sin(thetaRadians))
myPen.right(180)
myPen.forward(0.5*height/math.sin(thetaRadians))
myPen.left(theta)
myPen.forward(height/math.tan(thetaRadians))
myPen.right(180)
myPen.forward(height/math.tan(thetaRadians))
myPen.right(theta)
myPen.forward(0.5*height/math.sin(thetaRadians))
myPen.left(theta)

myPen.up()
myPen.forward(0.5*width)
myPen.down()

theta = 60
thetaRadians = math.radians(theta)
myPen.left(90)
myPen.forward(height)
myPen.right(90+theta)
myPen.forward(height/math.sin(thetaRadians))
myPen.left(90+theta)
myPen.forward(height)
myPen.right(180)
myPen.forward(height)
myPen.left(90)

myPen.up()
myPen.forward(0.5*width)
myPen.down()

theta = 45
thetaRadians = math.radians(theta)
myPen.left(90)
myPen.forward(height)
myPen.right(180)
myPen.forward(0.5*height)
myPen.left(90+theta)
myPen.forward(0.5*height/math.sin(thetaRadians))
myPen.right(180)
myPen.forward(0.5*height/math.sin(thetaRadians))
myPen.left(2*theta)
myPen.forward(0.5*height/math.sin(thetaRadians))
myPen.left(theta)

myPen.up()
myPen.forward(width)
myPen.down()

myPen.forward(width)
myPen.left(45)
myPen.forward(math.sqrt(2)*width)
myPen.left(90)
myPen.forward(math.sqrt(2)*width)
myPen.left(45)
myPen.forward(width)
myPen.right(45)
myPen.forward(math.sqrt(2)*width)
myPen.right(90)
myPen.forward(math.sqrt(2)*width)
myPen.right(45)
myPen.forward(2*width)
```



Reference

- Books and Websites on Python programming