Mansoura University

Faculty of Engineering

Computer Engineering and Control Systems Department

Artificial Intelligence

Maze solving competition

Team 3: Troll FTW

Robot: "حِناوي"


محمد خالد صلاح أبو الشريبني

عبدالرحمن محمد زكي حسن

فريد احمد فريد شرف احمد

محمد حسين محمد عطية شعيشع

هاجر محمد محمد أسعد الدعدع

سارة السيد عبدالحميد مصطفى القاضي

سهر حسام الدين متولي سعد النجار

حسن مجدي قاسم علي جبر

# Table of Contents

# INTRODUCTION

# 1. Introduction

A **maze** is a network of paths, contains an entrance and exit slots. The concept of Maze was invented in Egypt. From then, many mathematicians made various algorithm to solve the maze.

Now, Maze Solving Problem is included in robotics as it depends on “Decision Making Algorithm”. The robot will be in Unknown place in the maze and required to make decisions to solve a maze I.e. reach the exit point.

In this project, the main system design depends on Ultrasonic sensor to avoid wall

# 2. Ways to solve a maze

Various ways are used to solve a maze, listing some of them here ...

## 2.1. Wall Follower robot

Most mazes, however complex their design may appear, were essentially formed from one continuous wall with many connections to the exit.

If the wall surrounding the goal of a maze is connected to the entrance of the maze, the maze can always be solved by keeping one hand in contact with the wall, or as it called Wall following way.

Here it passes through two passes

- **First Pass:** The robot finds its way out from any point in the maze.
- **Second Pass:** Once the robot found a possible maze solution, it should optimize its solution finding the shortest path from start to finish.

After passes, construct a software with the combination of wall following and flood fill algorithms then implement the software in the hardware of maze solving robot.

## 2.2. Explorer

This method requires to use path finding technique, the robot can scan the maze into its memory and perform image processing against it, converting the pixels in the image into a data representation of the maze.

With the maze analyzed, an algorithm can run through it to determine a solution path through the maze. All of this can be done within seconds.

Once complete, the robot can simply walk through the maze in one try.

## 3.3. Search Algorithms

The way we used in this robot depends on search algorithms and PID control to move the robot through the maze.

A Python code consisting of BFS -Breadth First Search- or DFS –Depth First Search- is given the maze construction and analysis it finding the shortest path for the robot to solve it.

After finding the path in python code, directions are transmitted to the Arduino using Bluetooth module which will convert it to inspire the robot to solve the maze

# HARDWARE

# 3. Hardware Description

This section covers the important information about all hardware used in this project.

## 3.1. Arduino

Arduino is an open source hardware development board. Arduino hardware consists of an open hardware design with an Atmel AVR processor. Arduino programming language is used to program the processor.

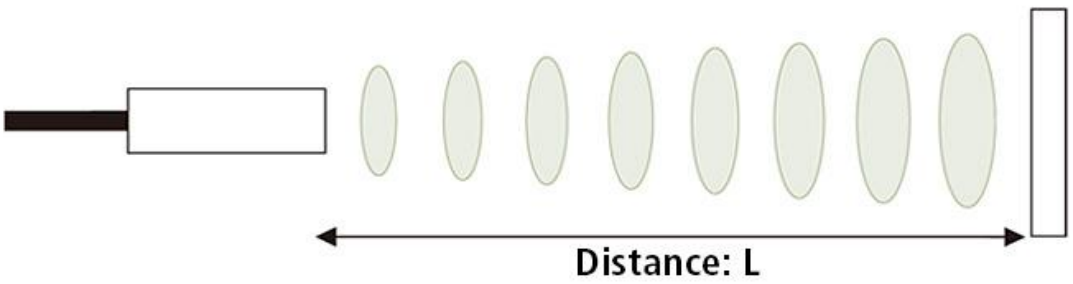
There are many types of Arduino board available in the market. But, in this project Arduino UNO is used.

Arduino UNO has 14 input/output pins. 6 digital pins can be used as PWM outputs. It has 6 analog input pins. It has a 16 MHz quartz crystal. It contains USB connection port, dc power port.

## 3.2. Ultrasonic Sensor (HC-SR04)

As the name indicates, ultrasonic sensors measure distance by using ultrasonic waves.

The sensor head emits an ultrasonic wave and receives the wave reflected back from the target. Ultrasonic Sensors measure the distance to the target by measuring the time between the emission and reception.



The distance can be calculated with the following formula:

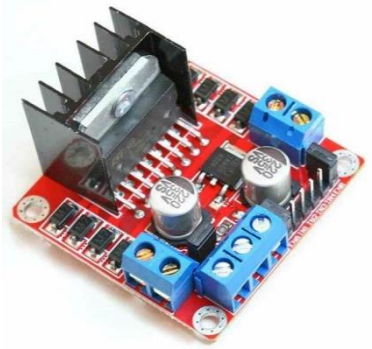
Distance L = 1/2 × T × C

where L is the distance, T is the time between the emission and reception, and C is the sonic speed. (The value is multiplied by 1/2 because T is the time for go-and-return distance.)

**Features:**

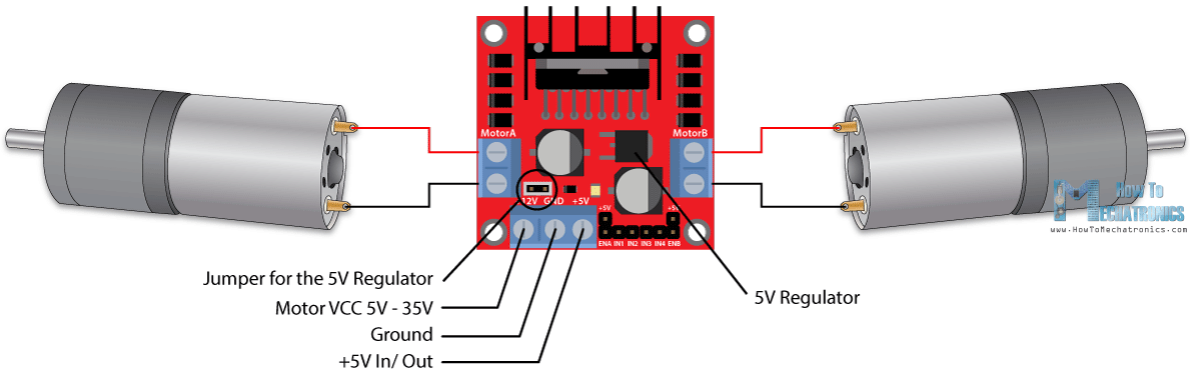
- Operating Voltage: 5V DC
- Operating Current: 15mA
- Measure Angle: 15°
- Ranging Distance: 2cm - 4m

## 3.3. Motor Driver L298N



The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

a closer look at the pinout of L298N module




The module has two screw terminal blocks for the motor A and B, and another screw terminal block for the Ground pin, the VCC for motor and a 5V pin which can either be an input or output.

**Features:**

- Logic voltage 5V
- Drive voltage 5V to 35V
- Logic Current 0mA-36mA
- Drive current 2A (MAX each bridge)
- Maximum Power 25W

## 3.4. DC Gear Motors

Two DC gear motors have been used to drive this robot. This gear motor is ideal for robotic cars.

With plastic construction and colored in bright yellow, the DC gear motor measures approx. 2.5-inch-long, 0.85-inch-wide and 0.7 inch thick.

The wheel can be mounted on either side and the gearmotor works well between 4V to 7V.

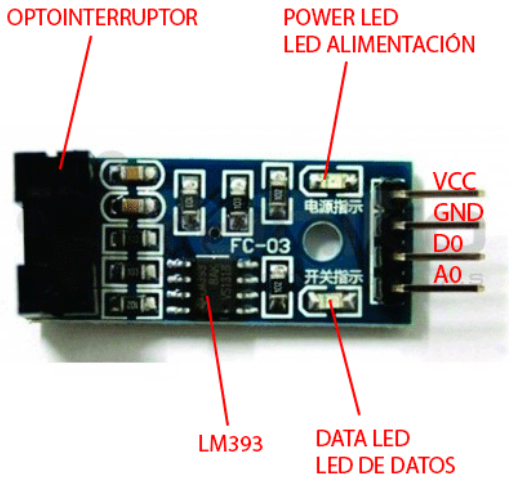
An Encoder Disc is attached so an encoder can be used later for speed and/or position control.

## 3.5. LM393 IR Sensor

This IR speed module sensor with the comparator LM393, we can calculate the speed of rotation of the wheels of our robot. If we place a ring gear that rotates attached to our wheel. It could also be used as an optical switch.

The basic operation of this sensor is as follows; If anything is passed between the sensor slot, it creates a digital pulse on the D0 pin. This pulse goes from 0V to 5V and is a digital TTL signal. Then with Arduino we can read this pulse.

Here are the different parts of the encoder:




## 3.6. Batteries

For the power source, 3 AA Batteries are used to supply the motor driver with the needed power. With series connection, batteries supply total 11.1 volt.

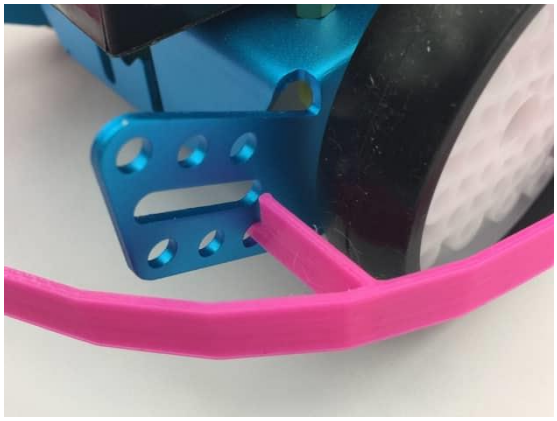
A power bank is used to supply the Arduino with the power.

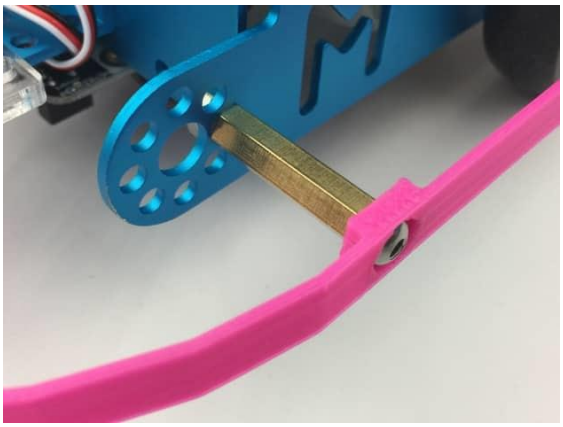
# MECHANICAL DESIGN

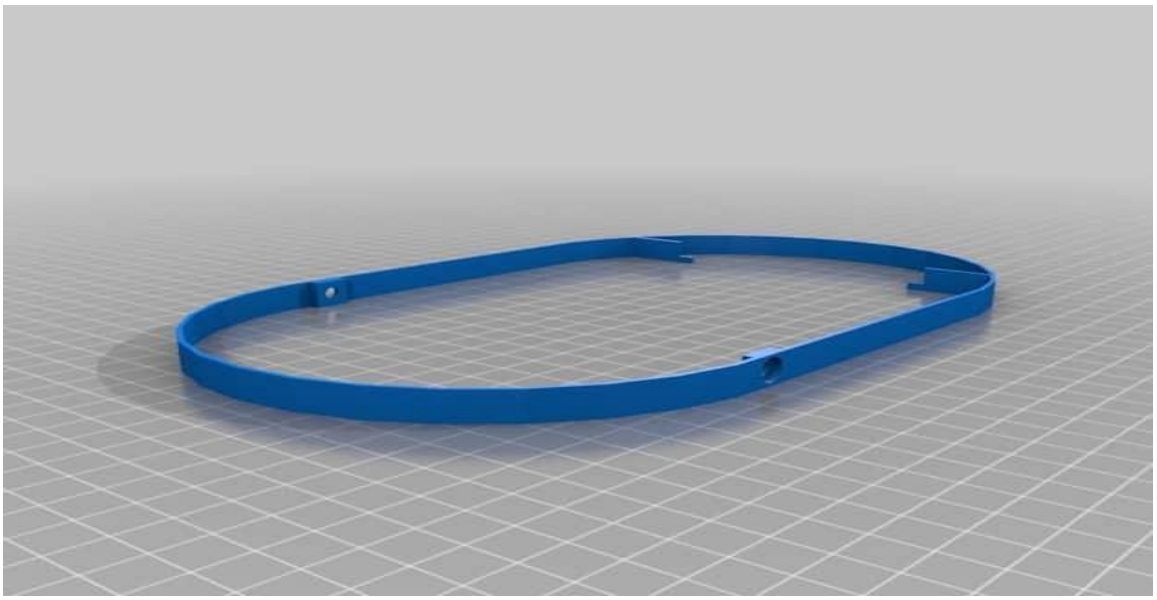
# 4. Robot design

The design was circular with rounded corners to make it easy moving between blocks and avoid hitting the wall.
It is made of acrylic plastic with many holes in each level which allowed us to fix all the components and gave a good air flow for the fixed components to get rid of the overheat problems.

Also the rounded corners was effective, a mechanical part was added to the robot to make it move smoothly, a capsule -Printed- made of plastic was mounted on the body of the lower level so it constraints the movement only in forward lines without hitting any wall so robot doesn't stop moving.







# ALGORITHMS

# 5. Python code

## 5.1. Stack

**Class** Stack:

An implementation for a data structure type contains a set of homogeneous elements, in which first entered element is the last one taken out.

```
#Class implementation for stack
#-----------------------------
class Stack:
    def __init__(self):
        self._L = []

    def Push(self,Element):
        self._L.append(Element)

    def pop(self):
        if (self.Count() > 0):
            return self._L.pop()
        else:
            return None
    def Count(self):
        return len(self._L)
    def isEmpty(self):
        return self.Count() == 0
```

**Functions:**

__init__ : A constructor taking one parameter –self- and constructing a list with dynamic length extending on adding items.

Push: Function takes two parameters, self and the element we need to add in the last index of the stack.

Pop: Removes the first element of the stack and return its value when called, else if the stack is empty ... it will return NULL.

Count: return the length of the stack, one based.

isEmpty: I Check if the stack is empty –elements = zero-? True: False

## 5.2. Queue

**Class** Queue:

An implementation for a data structure type contains a set of homogeneous elements, in which first entered element is the First one taken out.

```
#Class implementation for queue
#-----------------------------
class Queue:
    Lst = []
    def __init__(self):
        print(self.Lst)

    def enqueue(self, element):
        self.Lst.append(element)

    def dequeue(self):#remove first element and return it's value
        if(self.isEmpty() == False):
            x = self.Lst[0]
            self.Lst.pop(0)
            return x
        else:
            return None
    def count(self):
        return len(self.Lst)

    def isEmpty(self):
        return self.count() == 0
```

**Functions:**

__init__ : A constructor taking one parameter –self- and constructing a list with dynamic length extending on adding items.

enqueue: Function takes two parameters, self and the element we need to add in the last index of the queue.

dequeue: Removes the first element of the queue and return its value when called, else if the queue is empty ... it will return NULL.

Count: return the length of the queue, one based.

isEmpty: I Check if the queue is empty –elements = zero-? True: False

## 5.3. Maze node

**Class** MazeNode:

A class describes the state of each square in the maze. It takes the x and y co-ordinates and the state of each side of the borders, storing it in a Boolean variable to indicate which one is the way to exit.

```
# Maze Node class
#----------------
class MazeNode:
    def __init__ (self,x,y,isLeft,isRight,isTop,isBottom):
        self._x = x                             #X coordinates of the node
        self._y = y                             #y coordinates of the node
        self._isLeft = isLeft                   #is the way on the left?
        self._isRight = isRight                 #is the way on the Right?
        self._isTop = isTop                     #is the way on the Top?
        self._isBottom = isBottom               #is the way on the Bottom?
```

## 5.4. Maze

**Functions:**

- __init__: Constructor, takes three parameters including number of the squares in the x dimension and y dimension in the maze.
- _xSquares: Squares number in the x-dimension.
- _ySquares: Squares number in the y-dimension.
- Maze: Initialize a maze with the dimensions of x and y squares.
- startNode: The entrance of the maze.
- goalNode: The exit gate.

InitializeMaze:

Create an empty maze with the given dimensions for the maze in the constructor.

Using nested for loops, the outer loop counts the number of rows and on each loop, it initializes an empty row. The inner loop appends empty cells to the row by the numbers of the columns.

After finishing the inner loop, it adds the created empty row to the maze.

setStartNode: define the entrance point of the maze.

setGoalNode: define the last point in the maze.

## 5.5. Full python code

```
import serial
from time import sleep

class SerialTransfer(object):
    def __init__(self,port,baudRate=9600):
        self._port=port
        self._baudRate = baudRate
        # Establish the connection on a specific port with a specific baud
rate.
        self._serial = serial.Serial(self._port ,self._baudRate, timeout=5)
        self._serial.flushInput()
        self._serial.flushOutput()
        print("== SERIAL CONNECTION IS ESTABLISHED SUCCESSFULLY! ==")

    def __del__(self):
        self.close()

    def close(self):
        self._serial.close()

    def send(self, statement):
        if(self._serial.isOpen()):
            print("\n== SERIAL PORT IS OPENED! == send() ==")
            statement += '\r\n'
            encoded = statement.strip().encode()
            print ("Serial is sending:",encoded)
            self._serial.write(encoded)
            self._serial.flush()
            sleep(1)
        else:
            print("== SERIAL PORT IS NOT OPENED! ==")
```

```
    def recieve(self):
        if(self._serial.isOpen()):
            print("\n== SERIAL PORT IS OPENED! == recieve() ==")
            recievedData = self._serial.readline()
            stmt = recievedData.decode().strip()
            return stmt
        else:
            print("== SERIAL PORT IS NOT OPENED! ==")
            return None

#Stack Class
class Stack:
    def __init__ (self):
        self._L =[]
    def push(self, item):
        self._L.append(item)
    def pop(self):
        if (self.count() > 0):
            return self._L.pop()
        else:
            return None
    def count(self):
        return len(self._L)
    def isEmpty(self):
        return self.count()==0

#Queue Class
class Queue:
    def __init__ (self):
        self._L =[]
    def enqueue(self, item):
        self._L.append(item)
    def dequeue(self):
        if (self.count() > 0):
            return self._L.pop(0)
        else:
            return None
    def count(self):
        return len(self._L)
    def isEmpty(self):
        return self.count()==0

#Maze Node Class
class MazeNode:
    def __init__ (self, x, y, isLeft, isRight, isTop, isBottom):
        self._x = x
        self._y = y
```

```
self._isLeft = isLeft
        self._isRight = isRight
        self._isTop = isTop
        self._isBottom = isBottom

#Maze Class
import copy
class Maze:
    def __init__ (self, xSquares, ySquares):
        self._xSquares = xSquares
        self._ySquares = ySquares
        self._maze = []
        self._startNode = None
        self._goalNode = None
        self.initializeMaze()
    def initializeMaze(self):
        for i in range(self._ySquares):
            row =[]
            for j in range(self._xSquares):
                row.append(None)
            self._maze.append(row)
    def setStartNode(self, x, y):
        self._startNode = (x, y)
    def setGoalNode(self, x, y):
        self._goalNode =(x, y)
    def addMazeSquare(self, x, y, isLeft, isRight, isTop, isBottom):
        newMazeSquare = MazeNode(x, y, isLeft, isRight, isTop, isBottom)
        self._maze[y][x] =newMazeSquare
    def getMazeNodeByXY(self, x, y):
        return self._maze[y][x]
    def getMazeNodeByCoords(self, coords):
        return self._maze[coords[1]][coords[0]]
    def getMazeNodeChildren(self, x, y):
        children =[]
        if (x > 0 and not self._maze[y][x]._isLeft):
            children.append((self._maze[y][x - 1]._x, self._maze[y][x -
1]._y))
        if (x < self._xSquares - 1 and not self._maze[y][x]._isRight):
            children.append((self._maze[y][x + 1]._x, self._maze[y][x
+ 1]._y))
        if (y > 0 and not self._maze[y][x]._isTop):
            children.append((self._maze[y - 1][x]._x, self._maze[y -
1][x]._y))
        if (y < self._ySquares - 1 and not self._maze[y][x]._isBottom):
            children.append((self._maze[y +1][x]._x, self._maze[y + 1][x]._y))
        return children
```

```
    def depthFirstTraverse(self):
        if (self._startNode is None or self._goalNode is None):
            return None
        stack = Stack()
        stack.push([self._startNode])
        paths = []
        while(not stack.isEmpty()):
            currentPath = stack.pop()
            currentXY = currentPath[-1]
            if (self._goalNode == currentXY):
                paths.append(currentPath)
            currentNode = self.getMazeNodeByCoords(currentXY)
            currentChildren = self.getMazeNodeChildren(currentXY[0],
currentXY[1])[::-1]
            for child in currentChildren:
                if (child in currentPath):
                    continue
                stack.push(currentPath + [child])
        return paths
    def breadthFirstTraverse(self):
        if (self._startNode is None or self._goalNode is None):
            return None
        queue = Queue()
        queue.enqueue([self._startNode])
        paths = []
        while(not queue.isEmpty()):
            currentPath = queue.dequeue()
            currentXY = currentPath[-1]
            if (self._goalNode == currentXY):
                paths.append(currentPath)
            currentNode = self.getMazeNodeByCoords(currentXY)
            currentChildren = self.getMazeNodeChildren(currentXY[0],
currentXY[1])[::-1]
            for child in currentChildren:
                if (child in currentPath):
                    continue
                queue.enqueue(currentPath + [child])
        return paths
```

```
# Maze 2
game2 = Maze(8, 8)

#                 (x, y, isLeft, isRight, isTop, isBottm)
# First Rowt
t=True
f=False
game2.addMazeSquare(0, 0,t ,f ,t ,f )
game2.addMazeSquare(1, 0,f ,f ,t ,t )
game2.addMazeSquare(2, 0, f, f,t ,t )
game2.addMazeSquare(3, 0,f ,f ,t ,f )
game2.addMazeSquare(4, 0,f ,f ,t ,t )
game2.addMazeSquare(5, 0,f ,f ,t ,t )
game2.addMazeSquare(6, 0,f ,f ,t ,t )
game2.addMazeSquare(7, 0,f ,t ,t ,t )

#Second Row
game2.addMazeSquare(0, 1,t ,t ,f ,f )
game2.addMazeSquare(1, 1,t ,f ,t ,f )
game2.addMazeSquare(2, 1,f ,t ,t ,f )
game2.addMazeSquare(3, 1,t ,t ,f ,f )
game2.addMazeSquare(4, 1,t  ,f ,t ,f )
game2.addMazeSquare(5, 1,f  ,f ,t ,f )
game2.addMazeSquare(6, 1,f  ,f ,t ,t )
game2.addMazeSquare(7, 1, f ,f ,t ,f )


#Third Row
game2.addMazeSquare(0, 2,t  ,t ,f ,f )
game2.addMazeSquare(1, 2, t ,f ,f ,t )
game2.addMazeSquare(2, 2, f ,f , f,t )
game2.addMazeSquare(3, 2, f ,t ,f , f)
game2.addMazeSquare(4, 2,  t,t ,f ,t )
game2.addMazeSquare(5, 2, t ,f ,f,f )
game2.addMazeSquare(6, 2, f ,t ,t ,t )
game2.addMazeSquare(7, 2,  t, t,f ,f )


#Forth Row
game2.addMazeSquare(0, 3, t ,f ,f ,t )
game2.addMazeSquare(1, 3, f ,t , t, f)
game2.addMazeSquare(2, 3, t ,f ,t ,t )
game2.addMazeSquare(3, 3, f , f, f,t )
game2.addMazeSquare(4, 3, f ,f ,t , t)
game2.addMazeSquare(5, 3,  f,t,f , f)
game2.addMazeSquare(6, 3, t ,f , t,f )
game2.addMazeSquare(7, 3, f ,t ,f ,t )
```

```
#Fifth Row
game2.addMazeSquare(0, 4, t ,t ,f )
game2.addMazeSquare(1, 4, t ,f ,f ,f )
game2.addMazeSquare(2, 4, f , f,t ,t )
game2.addMazeSquare(3, 4, f ,f ,t ,t )
game2.addMazeSquare(4, 4,  f,f ,t ,f )
game2.addMazeSquare(5, 4,  f,t ,f ,t )
game2.addMazeSquare(6, 4,  t,f ,f ,f )
game2.addMazeSquare(7, 4,  f,t ,t ,f )


# Sixth Row
game2.addMazeSquare(0, 5, t ,f ,f ,f )
game2.addMazeSquare(1, 5, f ,t ,f , f)
game2.addMazeSquare(2, 5, t ,f ,t ,f )
game2.addMazeSquare(3, 5,  f,t ,t ,f )
game2.addMazeSquare(4, 5, t ,t ,f ,f )
game2.addMazeSquare(5, 5, t ,f ,t ,t )
game2.addMazeSquare(6, 5, f ,t ,f ,t )
game2.addMazeSquare(7, 5, t , t,f ,f )

# Seventh Row
game2.addMazeSquare(0, 6, t ,t ,f ,f )
game2.addMazeSquare(1, 6, t , t,f ,f )
game2.addMazeSquare(2, 6, t ,f ,f ,t )
game2.addMazeSquare(3, 6, f ,f ,f ,t )
game2.addMazeSquare(4, 6, f ,f,f ,f )
game2.addMazeSquare(5, 6, f ,f ,t ,t )
game2.addMazeSquare(6, 6, f ,f ,t ,t )
game2.addMazeSquare(7, 6,  f, t,f , t)


# Eighth Row
game2.addMazeSquare(0, 7, t ,t ,f ,f )
game2.addMazeSquare(1, 7, t ,f ,f ,t )
game2.addMazeSquare(2, 7, f ,f ,t ,t )
game2.addMazeSquare(3, 7,  f,t , t, t)
game2.addMazeSquare(4, 7,  t,f ,f ,t )
game2.addMazeSquare(5, 7, f ,f ,t ,t )
game2.addMazeSquare(6, 7, f , f,t ,t )
game2.addMazeSquare(7, 7, f ,t ,t ,t )


game2.setStartNode(4, 2)
game2.setGoalNode(7, 1)
print("\nMaze 2 8x8")
print("Using Depth First Search: ")
print(game2.depthFirstTraverse())
print("\nUsing Breadth First Search: ")
```

```
print(game2.breadthFirstTraverse())

paths=[]
MIN_PATH=""
S={'S':'f',
   'E':"l",
   'W':"r"}

N={'N':'f',
   'E':"r",
   'W':"l"}

E={'E':'f',
   'N':"l",
   'S':"r"}

W={'W':'f',
   'N':"r",
   'S':"l"}

CurrentDir=N
paths.extend(game2.depthFirstTraverse())
paths.extend(game2.breadthFirstTraverse())

if (len(paths)>0):
    minpath=paths[0]
    for path in paths:
        if(len(path)<len(minpath)):
            minpath=path
    minpathStr=""
    for coord in minpath:
        minpathStr+="("+str(coord[0])+"_"+str(coord[1])+")"
    print("MIN_PATH in char:",paths[0])
    for i in range(1, len(minpath)):

        if(minpath[i][0] > minpath[i-1][0]):       # X > X_current ==> E
            MIN_PATH +=CurrentDir['E']
            CurrentDir=E
        if(minpath[i][0] < minpath[i-1][0]):       # X < X_current ==> W
            MIN_PATH +=CurrentDir['W']
            CurrentDir=W
        if(minpath[i][1] > minpath[i-1][1]):       # Y > Y_current ==> S
            MIN_PATH +=CurrentDir['S']
            CurrentDir=S
        if(minpath[i][1] < minpath[i-1][1]):       # Y < Y_current ==> N
            MIN_PATH +=CurrentDir['N']
            CurrentDir=N
    MIN_PATH_new=""
```

```
   for i in range(len(MIN_PATH)):
        if((i<len(MIN_PATH)-
1)and((MIN_PATH[i]=='r')or(MIN_PATH[i]=='l'))and(MIN_PATH[i+1]=='f')):
            MIN_PATH_new+=MIN_PATH[i]+'x'
        else:
            MIN_PATH_new+=MIN_PATH[i]

    for coord in minpath:
        minpathStr+="("+str(coord[0])+"_"+str(coord[1])+")"
    print("min. length path:",minpathStr)
    print("MIN_PATH in char:",MIN_PATH_new)
```

Download

## 5.6. Arduino Code

Download

# 6. Robot pictures

High Quality photos here