Project One - Navigating Fixed World

Team Members: Gabriel De Sa and Arwa Jafferji

Robot Design:

For the design, we have a two motor robot with an omnidirectional wheel at the front for balance. We placed the EV3 block on top of the motors with some balance pieces. We decided that this was the best way for the robot to remain stable and not tip over during movement. The robot is similar to the reference robot on the lego website. To find the distance the robot will travel in one rotation, we took the diameter of the wheel. With the diameter we then used the circumference equation to find its circumference. This calculation for the larger wheel came out to be 17.28cm. We then knew that rotating the wheel 360º

Important Components:

To navigate through the course, we wanted to generate the path for the robot to follow. We decided to use a breadth-first search algorithm, avoiding the obstacles, to get from the start to the goal. To generate the world matrix, we take in the obstacles, start position, and the goal to build a matrix where the obstacles are represented with the number ‘1’ and the open positions are represented with the number ‘0’. To start, the path will be calculated using the bfs algorithm and then implemented using the moveToGoal function. Because we are using the tile length in respect to the robot’s size (we measured from the middle of the wheel to the front of the robot), it becomes easier for us to calculate the distance that we need to travel. Once the path is calculated, moveToGoal uses the angles and rotations calculated along with the moveToTarget function to go to each specified next coordinate.

Source Code:

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor, GyroSensor)
from pybricks.parameters import Port, Stop, Direction, Button, Color
from pybricks.tools import wait, StopWatch, DataLog
from pybricks.robotics import DriveBase
from pybricks.media.ev3dev import SoundFile, ImageFile
from collections import deque
import math



# This program requires LEGO EV3 MicroPython v2.0 or higher.
# Click "Open user guide" on the EV3 extension tab for more information.
```

```python
# Global Variables
# ROW = 12
# COL = 10
# tileLength = .305 # length of a single tile
xDim = 3.66
yDim = 3.05
tileLength = .11 # Length of the robot
ROW = round(xDim / tileLength)
COL = round(yDim / tileLength)
obstacleSize = .305
wheelCirc = (5.5*3.14159)/100 # meters

class node:
   def __init__(self, pt, distance: int, path: list):
       # coordinate
       self.pt = pt
       # distance from start
       self.distance = distance
       self.path = path


# check if the coordinate is valid
def checkValidity(row, col):
   return(row >= 0) and (row < ROW) and (col >= 0) and (col < COL)


# bfs
def bfs(matrix, startNode, goal):
   # getting four neighbors of any cell
   # for row and col
   # rowsNum = [-1, 0, 0, 1, -1, -1, 1, 1]
   # colsNum = [0, -1, 1, 0, -1, 1, -1, 1]
   rowsNum = [-1, 0, 0, 1]
   colsNum = [0, -1, 1, 0]

   if matrix[startNode[0]][startNode[1]] == 1 or matrix[goal[0]][goal[1]] == 1:
       return -1

   visited = [[False for i in range(COL)] for j in range(ROW)]

   queue = deque()  # queue = []
```

```python
    sourceDist = node(startNode, 0, [startNode])
    queue.append(sourceDist)

    while queue:
        current = queue.popleft()
        # done if we are at current
        pt = current.pt
        path = current.path
        if pt[0] == goal[0] and pt[1] == goal[1]:
            return path

        for i in range(len(rowsNum)):
            row = pt[0] + rowsNum[i]
            col = pt[1] + colsNum[i]
            if(checkValidity(row, col) and matrix[row][col] != 1 and not
visited[row][col]):
                visited[row][col] = True
                neighbor = (row, col)
                neighborPath = path.copy()
                neighborPath.append(neighbor)
                neighborCell = node(neighbor, current.distance + 1, neighborPath)
                queue.append(neighborCell)
    return -1

def generateWorld(obstaclesInput, start, goal):
    world = [[0] * COL for i in range(ROW)]
    startX = round(start[0]/tileLength) if round(start[0]/tileLength) < ROW else ROW-1
    startY = round(start[1]/tileLength) if round(start[1]/tileLength) < COL else COL-1
    goalX = round(goal[0]/tileLength) if round(goal[0]/tileLength) < ROW else ROW-1
    goalY = round(goal[1]/tileLength) if round(goal[1]/tileLength) < COL else COL-1
    world[startX][startY] = 2
    world[goalX][goalY] = 3
    divisions = round(.305/tileLength)
    for obstacle in obstaclesInput:
        x = round(obstacle[0]/tileLength)
        y = round(obstacle[1]/tileLength)
        if(divisions > 1):
            x_ = x - 1
            y_ = y - 1
            for i in range(divisions+2):
                xI = x_ + i
                if xI < 0:
```

```python
                    xI = 0
                elif xI >= ROW:
                    xI = ROW - 1
                for k in range(divisions+2):
                    yI = y_ + k
                    if yI < 0:
                        yI = 0
                    elif yI >= COL:
                        yI = COL-1
                    world[xI][yI] = 1
        else:
            world[x][y] = 1
    return world


def goToTarget(right, left, target, velocity = 360, wheelCirc = (5.5*3.14159)/100):
    resetAngles(right, left)
    desired_angle = (target/wheelCirc)*360
    right.run_angle(velocity, desired_angle, Stop.HOLD, False)
    left.run_angle(velocity, desired_angle)

def resetAngles(right, left, angle = 0):
    right.reset_angle(angle)
    left.reset_angle(angle)

def rotate(right, left, angle=420, velocity = 150):
    resetAngles(right, left)
    right.run_target(velocity, angle, Stop.HOLD, False)
    left.run_target(velocity, -angle)

def getAngles(path):
    dx = path[1][0] - path[0][0]
    dy = path[1][1] - path[0][1]
    delta = math.degrees(math.atan2(dy, dx))  # Finds angle between squares
    rotations = [delta]
    for i in range(2, len(path)):
        dx = path[i][0] - path[i-1][0]
        dy = path[i][1] - path[i-1][1]
        delta = math.degrees(math.atan2(dy, dx))  # Finds angle between squares
        rotations.append(delta)
    return rotations
```

```
def moveToGoal(rotations, right, left):
   rotate(right, left, rotations[0]*2)
   goToTarget(right, left, tileLength)
   for i in range(1, len(rotations)):
       angle, rotation = 0, rotations[i]
       if rotation > rotations[i-1]:
           angle = (rotation-rotations[i-1]) * 2
       elif rotation == rotations[i-1]:
           angle = 0
       elif (rotations[i-1] > 0) & (rotation == 0):
           angle = (-rotations[i-1]) * 2
       else:
           angle = rotation * 2
       rotate(right, left, angle)
       goToTarget(right, left, tileLength)

def addPath(world, path):
   for cord in path:
       world[cord[0]][cord[1]] = 4

def printMatrix(matrix):
   for row in matrix:
       print(row)

# Create your objects here.
def main():
   ev3 = EV3Brick()
   rightMotor = Motor(Port.A, Direction.COUNTERCLOCKWISE)
   leftMotor = Motor(Port.B, Direction.COUNTERCLOCKWISE)
   # Variable Changes
   start = (.305, .61)
   goal = (3.66, 1.83)
   obstacles = [(0.915, 0), (0.915, 0.305), (0.915, 0.61), (0.915, 0.915), (0.915,
1.22), (0.915, 1.525), (0.915, 1.833), (1.83, 1.22), (1.83, 1.525), ( 1.83, 1.83),
(1.83, 2.135), ( 1.83, 2.44), (1.83, 2.745), (1.83, 3.05), (2.745, 1.525), (2.745,
1.83), (3.355, 0.915), (3.355, 1.22), (3.66, 0.915), (3.66, 1.22)]
   world = generateWorld(obstacles, start, goal)
   # Finding path and moving robot.
   ev3.speaker.beep()
   startX = round(start[0]/tileLength) if round(start[0]/tileLength) < ROW else ROW-1
   startY = round(start[1]/tileLength) if round(start[1]/tileLength) < COL else COL-1
   goalX = round(goal[0]/tileLength) if round(goal[0]/tileLength) < ROW else ROW-1
```

```
    goalY = round(goal[1]/tileLength) if round(goal[1]/tileLength) < COL else COL-1
    path = bfs(world, (startX, startY), (goalX, goalY))
    addPath(world, path)
    printMatrix(world)
    rotations = getAngles(path)
    moveToGoal(rotations, rightMotor, leftMotor)

main()
```