```
// import fs module
var fs = require('fs');

// Initialize a 2D array representing a maze from maze.txt. '####' are obstacles
and '????' are open spaces.
function initializeMaze(maze) {
    var mazeString = fs.readFileSync('maze.txt', 'utf8');
    var mazeArray = mazeString.split('\n');
    for (var i = 0; i < mazeArray.length; i++) {
        maze[i] = mazeArray[i].replace(/\r/g, '').split(' ');
    }
    return maze;
}

// Write a function that prints the maze as a string to the console and is evenly
spaced
function printMaze(maze) {
    var mazeString = '';
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            // print 0's as 0.00
            if (maze[i][j] == 0) {
                maze[i][j] = '0.00';
            }
            if (maze[i][j].toString().length > 4) {
                mazeString += maze[i][j] + ' ';
            }
            else mazeString += maze[i][j] + '  ';
        }
        mazeString += '\n';
    }
    console.log(mazeString);
}

// overload print maze to take in a string to print before maze printing
function printMaze(maze, string) {
    var mazeString = string + '\n';
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            // print 0's as 0.00
            if (maze[i][j] == 0) {
                maze[i][j] = '0.00';
            }
            if (maze[i][j].toString().length > 4) {
                mazeString += maze[i][j] + ' ';
            }
            else mazeString += maze[i][j] + '  ';
        }
        mazeString += '\n';
    }
    console.log(mazeString);
}

// Initialize all '????' parts of the maze to be 100 / # of '????' squares
function initializeProbabilities(maze) {
    var totalSpaces = 0;
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            if (maze[i][j] == '????') {
```

```
                totalSpaces++;
            }
        }
    }
    var probability = 100 / totalSpaces;
    // limit probability to 2 decimal places
    probability = Math.round(probability * 100) / 100;
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            if (maze[i][j] == '????') {
                maze[i][j] = probability;
            }
        }
    }
    return maze;
}

// Add in Filtering Algorithm
function calculateProbability(maze, e, x, y) {
    // e is the surrounding evidence (i.e. [1, 0, 1, 0])
    // e[0] is west, e[1] is north, e[2] is east, e[3] is south
    // 1 is an obstacle, or an out of bounds index. 0 is an open space.
    // Correctly identifying an obstacle = 0.9, correctly identifying an open space
= 0.95
    // Incorrectly identifying an obstacle = 0.05, incorrectly identifying an open
space = 0.1
    var correctObstacle = 0.9;
    var correctOpenSpace = 0.95;
    var incorrectObstacle = 0.05;
    var incorrectOpenSpace = 0.1;
    // Probability = P(e[0] | x-1) * P(e[1] | x+1) * P(e[2] | y-1) * P(e[3] | y+1)
    var probability = 1;
    for (var i = 0; i < e.length; i++) {
        if (e[i] == 1) {
            if (i == 0) {
                // west
                if (y > 0 && maze[x][y-1] == '####') {
                    probability *= correctObstacle;
                } else {
                    probability *= incorrectObstacle;
                }
            } else if (i == 1) {
                // north
                if (x > 0 && maze[x-1][y] == '####') {
                    probability *= correctObstacle;
                } else {
                    probability *= incorrectObstacle;
                }
            } else if (i == 2) {
                // east
                if (y < maze.length - 1 && maze[x][y+1] == '####') {
                    probability *= correctObstacle;
                } else {
                    probability *= incorrectObstacle;
                }
            } else if (i == 3) {
                // south
                if (x < maze[0].length - 1 && maze[x+1][y] == '####') {
                    probability *= correctObstacle;
```

```
                } else {
                    probability *= incorrectObstacle;
                }
            }
        } else {
            if (i == 0) {
                // west
                if (y > 0 && maze[x][y-1] != '####') {
                    probability *= correctOpenSpace;
                } else {
                    probability *= incorrectOpenSpace;
                }
            } else if (i == 1) {
                // north
                if (x > 0 && maze[x-1][y] != '####') {
                    probability *= correctOpenSpace;
                } else {
                    probability *= incorrectOpenSpace;
                }
            } else if (i == 2) {
                // east
                if (y < maze.length - 1 && maze[x][y+1] != '####') {
                    probability *= correctOpenSpace;
                } else {
                    probability *= incorrectOpenSpace;
                }
            } else if (i == 3) {
                // south
                if (x < maze[0].length - 1 && maze[x+1][y] != '####') {
                    probability *= correctOpenSpace;
                } else {
                    probability *= incorrectOpenSpace;
                }
            }
        }
    }

    return probability;
}

function filtering(maze, e) {
    // e is the surrounding evidence (i.e. [1, 0, 1, 0])
    // var probabilityObject = {x: i, y: j, probability: 0};
    var probabilities = [];
    var probabilitySum = 0;
    // Loop through maze and sense at every open spot/ calculate that probability
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            if (maze[i][j] != '####') {
                let tempProbOb = {x: i, y: j, probability: maze[i][j] *=
calculateProbability(maze, e, i, j)};
                probabilities.push(tempProbOb);
                probabilitySum += tempProbOb.probability;
            }
        }
    }
    // Normalize the probabilities by dividing by the sum of all probabilities
    for (var i = 0; i < probabilities.length; i++) {
        probabilities[i].probability /= probabilitySum;
```

```
    }
    // Update the maze with the new probabilities
    for (var i = 0; i < probabilities.length; i++) {
        // convert probability to percentages
        probabilities[i].probability *= 100;
        // round probability to 2 decimal places
        probabilities[i].probability = Math.round(probabilities[i].probability *
100) / 100;
        maze[probabilities[i].x][probabilities[i].y] =
probabilities[i].probability;
    }
    return maze;
}

function calculateDrift(maze, d, x, y) {
    // d is the direction to move (i.e. W, N, E)
    // There is a 0.75 chance of moving in the correct direction(straight), a 0.15
chance he blows left
    // and a 0.15 chance he blows right. If he blows left or right into an obstacle
he stays in the same spot.
    // If he blows left or right into an open space he moves into that space.
    var relativeStraight = 0.75;
    var leftDirection = 0.15;
    var rightDirection = 0.1;

    // Outcomes will be an array of objects with the following properties:
    // x, y, probability
    // x and y are the coordinates of the outcome, probability is the probability
of that outcome
    var outcomes = [];

    // If the direction is west
    if (d == 'W') {
        if (y>0 && maze[x][y-1] != '####') {
            // If the spot to the left is an open space
            // Add the spot to the left to the outcomes array
            outcomes.push({x: x, y: y-1, probability: maze[x]
[y]*relativeStraight});
        }
        else {
            // Add the current spot to the outcomes array
            outcomes.push({x: x, y: y, probability: maze[x][y]*relativeStraight});
        }
        // Calculate relative drift direction probabilities
        // Calculate left drift direction from robot facing west
        if (x>0 && maze[x+1][y] != '####') {
            outcomes.push({x: x+1, y: y, probability: maze[x][y]*leftDirection});
        }
        else {
            outcomes.push({x: x, y: y, probability: maze[x][y]*leftDirection});
        }
        // Calculate right drift direction from robot facing west
        if (x>0 && maze[x-1][y] != '####') {
            outcomes.push({x: x-1, y: y, probability: maze[x][y]*rightDirection});
        }
        else {
            outcomes.push({x: x, y: y, probability: maze[x][y]*rightDirection});
        }
    }
```

```
    // If the direction is north
    else if (d == 'N') {
        if (x>0 && maze[x-1][y] != '####') {
            // If the spot to the left is an open space
            // Add the spot to the left to the outcomes array
            outcomes.push({x: x-1, y: y, probability: maze[x]
[y]*relativeStraight});
        }
        else {
            // Add the current spot to the outcomes array
            outcomes.push({x: x, y: y, probability: maze[x][y]*relativeStraight});
        }
        // Calculate relative drift direction probabilities
        // Calculate left drift direction from robot facing north
        if (y>0 && maze[x][y-1] != '####') {
            outcomes.push({x: x, y: y-1, probability: maze[x][y]*leftDirection});
        }
        else {
            outcomes.push({x: x, y: y, probability: maze[x][y]*leftDirection});
        }
        // Calculate right drift direction from robot facing north
        if (y<maze[0].length-1 && maze[x][y+1] != '####') {
            outcomes.push({x: x, y: y+1, probability: maze[x][y]*rightDirection});
        }
        else {
            outcomes.push({x: x, y: y, probability: maze[x][y]*rightDirection});
        }
    }
    // If the direction is east
    else if (d == 'E') {
        if (y<maze[0].length-1 && maze[x][y+1] != '####') {
            // If the spot to the left is an open space
            // Add the spot to the left to the outcomes array
            outcomes.push({x: x, y: y+1, probability: maze[x]
[y]*relativeStraight});
        }
        else {
            // Add the current spot to the outcomes array
            outcomes.push({x: x, y: y, probability: maze[x][y]*relativeStraight});
        }
        // Calculate relative drift direction probabilities
        // Calculate left drift direction from robot facing east
        if (x>0 && maze[x-1][y] != '####') {
            outcomes.push({x: x-1, y: y, probability: maze[x][y]*leftDirection});
        }
        else {
            outcomes.push({x: x, y: y, probability: maze[x][y]*leftDirection});
        }
        // Calculate right drift direction from robot facing east
        if (x<maze.length-1 && maze[x+1][y] != '####') {
            outcomes.push({x: x+1, y: y, probability: maze[x][y]*rightDirection});
        }
    }
    return outcomes;
}

// Add in Prediction Algorithm
function move(maze, d) {
    var outcomes = [];
```

```
    // Loop through maze
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            // If the current spot is not an obstacle
            if (maze[i][j] != '####') {
                // Append the entries of the returned array from calculateDrift to
outcomes
                let temp = calculateDrift(maze, d, i, j);
                outcomes.push.apply(outcomes, temp);
            }
        }
    }

    // Loop through the maze and everytime you find an open space
    // Add all of the outcome probabilities with the same i,j coordinates
    // And initialize that to maze at i,j
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            if (maze[i][j] != '####') {
                var sum = 0;
                for (var k = 0; k < outcomes.length; k++) {
                    if (outcomes[k].x == i && outcomes[k].y == j) {
                        sum += outcomes[k].probability;
                    }
                }
                // round sum to 2 decimal places
                sum = Math.round(sum*100)/100;
                maze[i][j] = sum;
            }
        }
    }
    return maze;
}


maze = [];
maze = initializeMaze(maze);
maze = initializeProbabilities(maze);
/* 1. Sensing: [1, 0, 1, 0]
2. Moving: N
3. Sensing: [0, 0, 0, 0]
4. Moving: E
5. Sensing: [0, 1, 0, 1]
6. Moving: E
7. Sensing: [0, 0, 1, 1]
8. Moving: N
9. Sensing: [1, 0, 1, 0] */
printMaze(maze, "Initial Probabilities");
maze = filtering(maze, [1, 0, 1, 0]);
printMaze(maze, `Filtering: [1, 0, 1, 0]`);
maze = move(maze, 'N');
printMaze(maze, `Prediction after action: N`);
maze = filtering(maze, [0, 0, 0, 0]);
printMaze(maze, `Filtering: [0, 0, 0, 0]`);
maze = move(maze, 'E');
printMaze(maze, `Prediction after action: E`);
maze = filtering(maze, [0, 1, 0, 1]);
printMaze(maze, `Filtering: [0, 1, 0, 1]`);
```

```
maze = move(maze, 'E');
printMaze(maze, `Prediction after action: E`);
maze = filtering(maze, [0, 0, 1, 1]);
printMaze(maze, `Filtering: [0, 0, 1, 1]`);
maze = move(maze, 'N');
printMaze(maze, `Prediction after action: N`);
maze = filtering(maze, [1, 0, 1, 0]);
printMaze(maze, `Filtering: [1, 0, 1, 0]`);
console.log("Programmed By Yours Truly, Samuel Hale");
//console.log the current time
console.log("Current Time: " + new Date().toLocaleTimeString());
```