VERKFRÆÐILEGAR BESTUNARAÐFERÐIR

Day 1

T-423-ENOP

Arnar Gylfi Haraldsson
arnarh23@ru.is

Hafþór Árni Hermannsson
hafthorh20@ru.is

Ragnheiður Gná Gústafsdóttir
ragnheidurg@ru.is

April 30, 2024

Exercise 1

Overview

The people.m MATLAB function manages a database of names and ages stored in a .mat file, supporting operations to reset, list, insert, and remove records.

Functionality

- Reset: people(db_file, 'reset') initializes an empty database.
- List: people(db_file, 'list') displays all entries or indicates if the database is empty or the file is missing.
- Insert: people(db_file, 'insert', 'Name', Age) adds new entries, checking for valid data format and duplication.
- Remove: people(db_file, 'remove', 'Name', Age) deletes specified entries if they exist.

Testing

The function was validated using test_database.m, demonstrating its capability to handle database initialization, data manipulation, and error handling effectively. Tests included adding and removing entries, checking for non-existent data, and handling invalid inputs.

```
>> people('data.mat', 'reset');
Database initialized
>> people('data.mat', 'list');
Database is empty.
>> people('data.mat', 'insert', 'John', 25, 'Anna', 17, 'Max',50);
>> people('data.mat', 'list');
 ----------------------
    Database contents

John, 25
Anna, 17
Max, 50
  ---------------------
>> people('data.mat', 'remove', 'John', 25);
>> people('data.mat', 'list');
 ----------------------
    Database contents

Anna, 17
Max, 50
  ---------------------
>> people('data.mat', 'remove', 'Jack', 25);
No entry found for Jack, 25 to remove.
>> people('data.mat', 'insert', 'Anna', 17);
Person Anna, age 17 already exists in the database. No insertion made.
>> people('data.mat', 'insert', 'Lisa', -10);
Invalid input for person: name must be a string and age must be a non-negative number. No insertion made for Lisa, -10.
```

Figure 1: Database test

Exercise 2

Overview

A function was implemented that draws a polygon defined by its vertices and rotates it with respect to the center of the coordinate system.

Functionality

We decided to let the vertices of the polygon, the number of rotations and theta, the degree of which each rotation rotates in deg, is to be determined by the user.
Firstly, we plot the original polygon (blue lines) as well as the center (red dot). The polygon plot is stored in an handle because we have to be able to delete the old structure later on when we are ready to plot the rotated structure.
This following part is iterated as many times as the user defined the number of rotations. The new vertices are determined by the following equation:

$$newVertices = (oldvertices - center) * R + center \tag{1}$$

where R is the rotation matrix, defined below where theta has been changed from deg to rad:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2}$$

Now we delete the last polygon plot and make a new one with the new vertices. When all iterations are done we close the plot. Figure 2 show the position of the polygon shortly after start and figure 3 shows the last position when the polygon is rotated 10 times, 5 degrees each time.

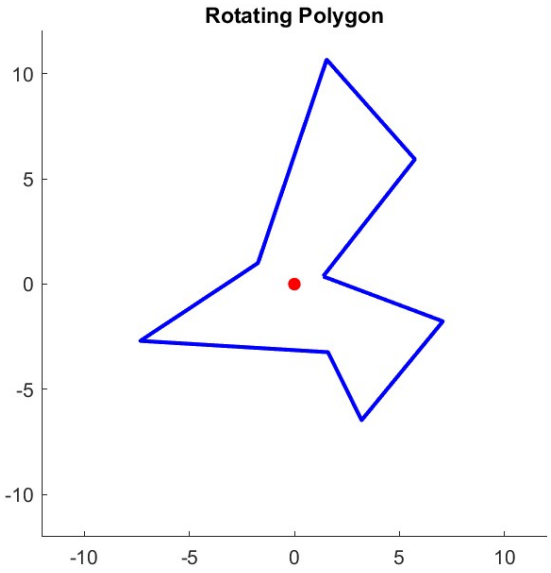

Figure 2: Position shortly after start

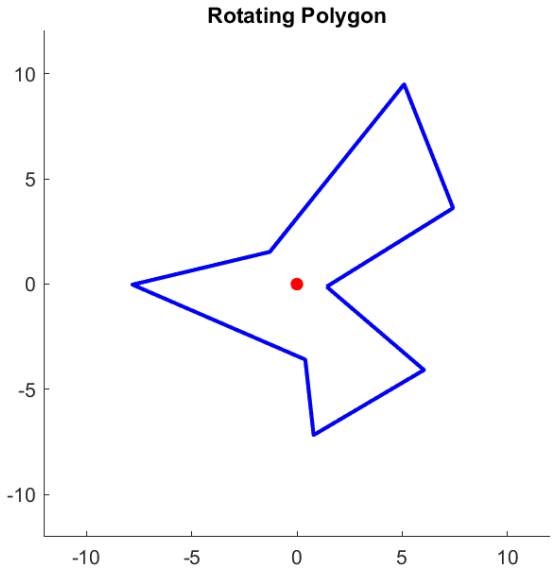

Figure 3: Final position after 10 iterations

Exercise 3

Overview

Exercise 3 focuses on the problem of finding a path from the bottom to the top of a $N \times N$ grid while avoiding obstacles. This is a classic pathfinding problem, typically addressed using algorithms like Breadth-First Search (BFS), which is used here. The task also involves visualizing the grid, obstacles, and the identified path.

Functionality

The implemented MATLAB code PathFinder(N, P) generates a grid where P represents the probability of each cell being an obstacle. The algorithm searches for the shortest path from the top edge to the bottom edge of the grid, considering only horizontal or vertical movements. If a path is found, it is displayed; otherwise, the program outputs that no path is available. The algorithm first initializes the grid where all the values are 0. Then it assigns a value of -1 of all the randomly selected obstacles and marks all the non obstacle values of the top row to be 1. Then it checks all the neighbours of these start points and if they are not an obstacle it gives it a value of one more, 2. Then the same process is repeated for these neighbours, who all have the value 2 now, and their non-obstacle neighbours given the number 3, and so on and so forth until we have reached the bottom. If we mark a member of the bottom row we know we have reached the bottom and therefore found a path, a the end point is stores and a boolean value is set to true.
Then we backtrack trough the path always finding a neighbor with the value one less than the point in question and mark down each point in the path. Everything is then plotted and if a path is found it is plotted with a green line, the obstacles are marked with a red circle.

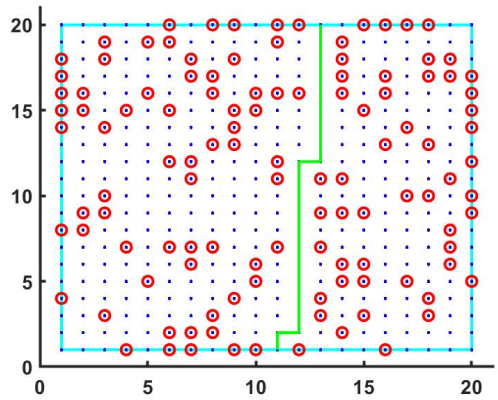

Figure 4: Shortest path found with N = 20, obstacle density = 0.3

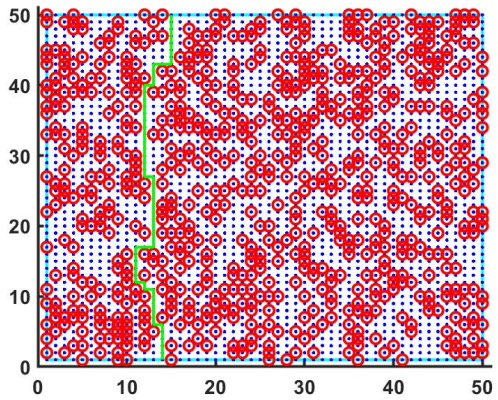

Figure 5: Shortest path found with N = 50, obstacle density = 0.3

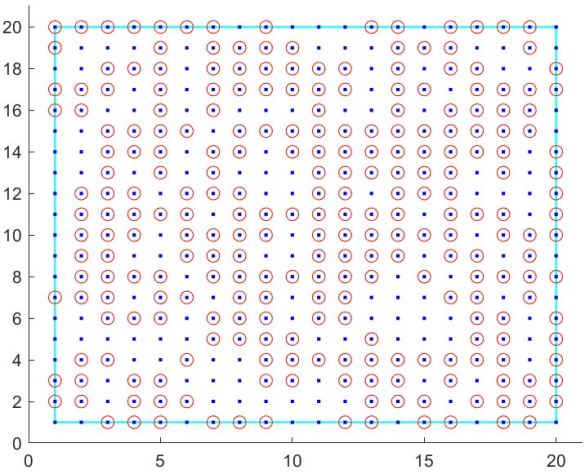

Figure 6: No path found with N = 20, obstacle density = 0.6

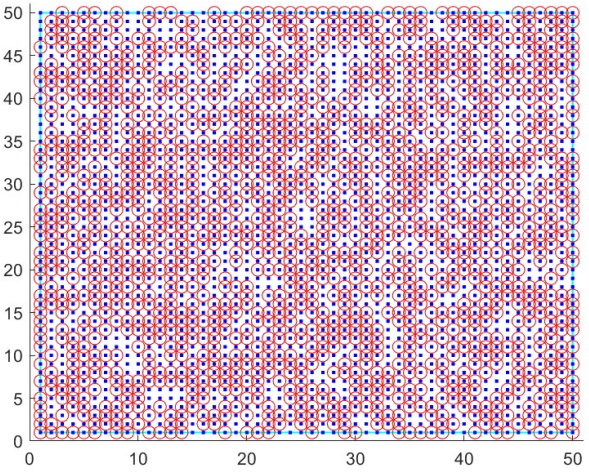

Figure 7: No path found with N = 50, obstacle density = 0.6

Testing

We did a 100 simulations at each obstacle density value from 0.0 to 1.0 with two different size matrices, N = 30 and N = 100, and plotted up the success rate of paths found. As can be seen in figures 8 and 9 the success rate start high and then dramatically plunges around the value P = 0.4 and converges towards zero from then on.

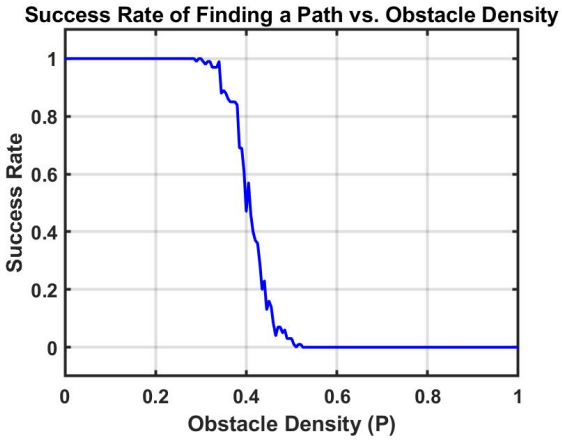

Figure 8: 100 simulations at each obstacle density value with N = 30

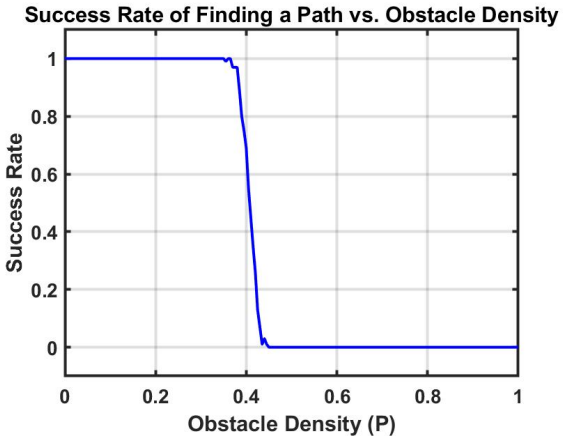

Figure 9: 100 simulations at each obstacle density value with N = 100