
	<p>Operating Systems 2 (Fall 2021) Project Discussion</p>	
---	---	---

Operating systems 2 project documentation

Project description:

Our project is more like a game called “Rat in Maze”, Maze is given as $N \times N$ binary matrix of blocks where source block is the upper left most block and destination block is lower right most.

In maze matrix a rat start from source and must reach the destination , the rat can move only in two direction : forward and down .

The inputs are given as:

1- number of N , which N the number of rows and columns of the matrix , N must be 3 or more Example on the input:

3

6



8

Example on the output:

Matrix 3×3

Matrix 6×6

Matrix 8×8

	<p>Operating Systems 2 (Fall 2021) Project Discussion</p>	 <p>كلية الحاسبات والذكاء الاصطناعي Faculty of Computers & Artificial Intelligence</p>
---	---	--

What we have actually did:

It took us a while to figure out what to do in this project, it wasn't easy but we did it, we found a way of cooperating with one another as a team and we have done the project exactly as it was described.

We used a 2D arrays to specify the pieces and the grid (the square), we used object oriented programming and Java threads to implement this project.

Team members roles:

Algorithm: Fatma Salama, Noha Mostafa and Mayar Ahmad.

The matrix: Sohaila Mohamed and Fatma El Zharaa

Testing the code: Fatma sayed and Mayar Ahmad.

Documentation: Zeinab Amr.

Gui: Zeinab Amr.

Code documentation:

DFS Class

Here we define what we will need in our DFS (our graph, its size and the traversed array)

We also define what we will need when it comes to multithreading (like the MAX # of threads in parallel).

```
public class DFS_Class {  
    // Here we define what we will need in our DFS (our graph, its size and the traversed array)  
    // We also define what we will need when it comes to multithreading (like the MAX # of threads i  
  
    private char[][] graph;  
    private int N;  
    private boolean[][] traversed;  
    private int MAX_THREADS = 1024;  
    private boolean found = false;  
    private ArrayList<Pair> Answer;  
  
    public DFS_Class(char[][] indexes, int dimensions) {  
        graph = indexes;  
        this.N = dimensions;  
        this.traversed = new boolean[N][N];  
        ArrayList<Pair> Arr = new ArrayList<>();  
        DFS(0, 0, Arr);  
  
        // A loop is used here to make sure that we traversed every possible route before we move to
```

A loop is used here to make sure that we traversed every possible route before we move to the answer

```
// A loop is used here to make sure that we traversed every possible route before we move
while (Thread.activeCount() > 3) {

}

if (found) {
    for (Pair pair : Answer) {
        //System.out.print(Answer.get(i).First + " " + Answer.get(i).Second + "\n");
        graph[pair.First][pair.Second] = '2';
    }
    graph[0][0] = '2';
    getAnswer();
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            System.out.print(graph[i][j]);
        System.out.println("\n");
    }

} else {
    System.out.println("Not found!");
}

}
```

The class that gets threaded, it creates a thread for every path possible until one of the conditions are met

- Possibilities are: 1- a deadend is reached, we end the path /
- 2- we reached the maximum # of threads, we use normal DFS
- 3- We found a path, we inform the main class and safely end other threads

```
private class realDFS implements Runnable {  
  
    int x, y;  
    ArrayList<Pair> ans;  
  
    realDFS(int x, int y, ArrayList<Pair> ans) {  
        this.x = x;  
        this.y = y;  
        this.ans = ans;  
    }  
  
    public void run() {  
        try {  
            DFS(x, y, ans);  
        } catch (InterruptedException ex) {  
            Logger.getLogger(DFS_Class.class.getName()).log(Level.SEVERE, null, ex);  
        }  
    }  
}
```

Our init DFS method, invoked only once in most cases when MAXTHREAD's value is more than 2

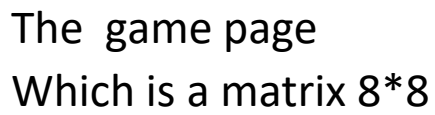
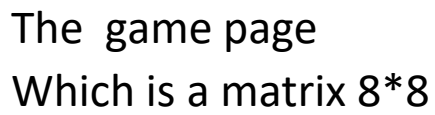
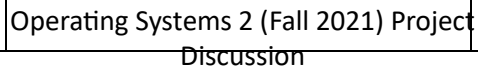
```
private void DFS(int x, int y, ArrayList<Pair> ans) {  
    if (found) {  
        return;  
    }  
    if (graph[x][y] == '2') {  
        Answer = ans;  
        found = true;  
        return;  
    }  
    if (!(x + 1 == N) && !(traversed[x + 1][y]) && !(graph[x + 1][y] == '0')) {  
  
        ArrayList<Pair> ans1 = new ArrayList<>(ans);  
        ans1.add(new Pair(x + 1, y));  
  
        traversed[x + 1][y] = true;  
        if (Thread.activeCount() < MAX_THREADS) {  
            realDFS gl = new realDFS(x + 1, y, ans1);  
            Thread t1 = new Thread(gl);  
  
            t1.start();  
  
        } else {  
            DFS(x + 1, y, ans1);  
        }  
    }  
}
```

Graphical user interface:

The first page:

Which ask you entre the N (Number of rows and columns)





Solution page:

