|  |  |  |  |
| --- | --- | --- | --- |
|  | Team Member ID | Team member name (in Arabic) | Grade |
| 1 | احمد رضا شعبان عبد الوهاب | 20210050 |  |
| 2 | يوسف محمد نادي حسن | 20211104 |  |
| 3 | محمد عبدالرزاق فؤاد | 20210807 |  |
| 4 | محمد أيمن محمد قنديل | 20210751 |  |
| 5 | محمد ناصر رفعت على | 20210841 |  |
| 6 | محمد عوض الله سري احمد | 20210822 |  |
| 7 | محمد مصطفي محمود عبدالمجيد | 20210837 |  |

# Project description:

A Maze is given as 𝑁 × 𝑁 binary matrix of blocks where source block is the upper left most block i.e., maze [0][0] and destination block is lower rightmost block i.e., maze[N-1][N-1]. A rat starts from source and must reach the destination. The rat can move only in two directions: forward and down. In the maze matrix, 0 means the block is a dead end and 1 means the block can be used in the path from source to destination.

We Use Multi-threading to solve this problem. We design a multithreaded JAVA program with the following features: -

● entering the dimensions of the maze, then a grid is generated.

● using the grid to specify dead blocks on runtime

**Solution:**

Since the rat can only move in two directions: forward and down, our first thinking was to make to parallel threads work at the same time. One of the two thread is to move forward, and one is to move down, so the forward thread will be put to sleep when the rat is facing a forward dead end and waits a random time until down thread works, so as forward thread, down thread will be put to sleep and waits a random time until forward thread works. The two threads access the same Boolean variable (deadEnd), if the forward thread faces a dead end and down thread faces another dead end so the down thread will access the down thread set the variable (deadEnd) to true and then the program is terminated.

**Critical section and race condition:**

The two (forward and down) threads access the same arrayList of a class (grid class), this class is made only to save our rat’s steps, So the rat can remember what path he took at the first time if it faced a dead end and take a step backward and try another path. So, we faced race condition problem as this arrayList is our critical section.

**Race condition solved:**

The two threads being accessing the same arrayList and the two threads can add and update it at self is a race condition problem as only one thread should access and alter it, otherwise, a problem will occur, and the data will not be correct.

This problem is solved using mutex lock on the critical section, Mutex lock ensures that only one thread can access the critical section which in our case the arrayList. I.e., Mutex Lock ensures mutual execution.

**A screenshot of a crossword puzzle

Description automatically generated**Also, we need this arrayList to be recorded in the main memory so every alteration to the list is done and every thread can see it.

**Initial state:**

**A screenshot of a game

Description automatically generated**

**Ending state:**

# Solving steps:

First, we can’t predict how the algorithm, because we don’t know what thread will be executed first if there are more than one free path is available for the rat to take.

Steps

1. We suppose the forward thread works first, So the rat will record that the current step is (0,1)

2. We suppose the down thread works, so

it will access the current step in the arrayList and increment X and see if the next step is block or not, in this case the rat will record that (1,1) is blocked and sleep.

3. We suppose the forward thread works, so it will access the current step in the arrayList and increment Y and see if the next step is block or not, in this case the rat will record that the next step is the current step, and the current step is (0,2) and sleep.

4. We suppose the down thread works, so it will access the current step in the arrayList and increment X and see if the next step is block or not, in this case the rat will record that the next step is the current step, and the current step is (1,2) and sleep.

5. We suppose the forward thread works, so it will access the current step in the arrayList and increment Y and see if the next step is block or not, in this case the rat will record that the next step is the current step, and the current step is (1,3) and sleep.

6. We suppose the down thread works, so it will access the current step in the arrayList and increment X and see if the next step is block or not, in this case the rat will record that the next step is the current step, and the current step is (2,2) and sleep.

And so on until the rat reaches its destination.

# -Team members roles:

**Documentation:** Ahmed Reda and Youssef Nady.

**Video:** Mohamed Mostafa.

**Video-editing:** Mohamed Abdulrazak.

**Maze:** Mohamed Qandil, Mohamed Mostafa.

**GUI:** Mohamed Abdulrazak and Mohamed Nasser.

**Testing the code:** Mohamed Nasser and Mohamed Mostafa.

**Run forward:** Youssef Nady and Mohamed Qandil.

**Run down:** Mohamed Awad and Ahmed Reda.

# Code documentation:

**initializeGUI()** :

-WE SHOW TO USER THE GUI OF OUR ALG AS OUR CELL 100 PIXEL WITH NxN MATRIX AND SUBMIT BUTTON.

**A screenshot of a computer code

Description automatically generated**

A screenshot of a computer code

Description automatically generateddrawMaze(Graphics g):

-WE PUT SOME COLORS

TO CELLS AS WHTE INDICATES

TO ACCESSABLE CELL, BLUE

INDICATES TO VISITED CELL

AND BLACK INDICATES TO DEAD CELL.

toggleCell(int row, int col) :

-WE TOGGLE OUR CELL FROM 0 TO 1 OR OTHERWISE AS USER CHANGES IT TO REPAINT IT.

A close up of text

Description automatically generated

exploreDirection(int r, int c, int[][] maze):

A close up of text

Description automatically generated-WE SHOW WHICH DIRECTIO THREAD MOVES AND RECURSION TO OUR MAIN FUNCTION.

class RatThread:

-WE CREATED THIS CLASS AS IT INCLUDES MAIN FUNCTIONS OF OUR ALG TO CREATE THREADS AND MOVES IT TO GOAL.

startMazeSolver ():

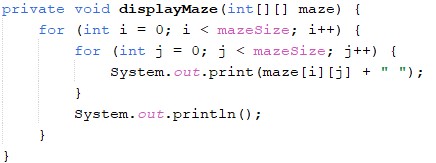
-WE HANDLE SOME ERRORS IN ALG AS IF USER DIDN’T TOGGLE CELL TO DEAD CELL OR TOGGLE INITIAL CELL OR CREATE OBJ TO RATTHREAD TO BEGIN ALG.

A computer screen shot of a computer code

Description automatically generated

displayMaze(int[][] maze)

-SHOW THREAD DIR TO GOAL IN TERMINAL.



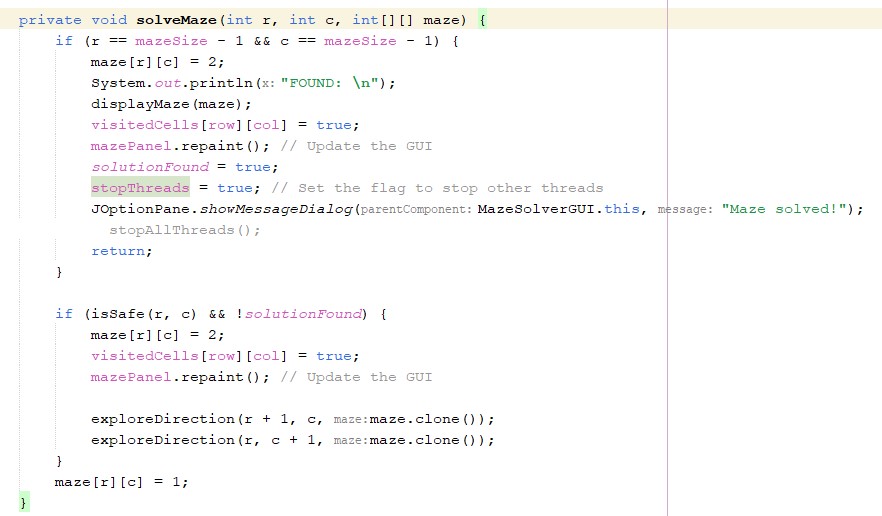
public static void main(String args[]) {}

-WE TAKE N AS DIMENSIONS FROM USER TO CREATE MAZE AND PASS IT PARAMETER TO OUR CLASS.

A screenshot of a computer code

Description automatically generated

private void solveMaze(int r, int c, int[][] maze)

-THIS FUNCTION THAT CREATES THREAD, EXPLORE DIRECTIONS AND SHOW THREAD DIRECTION.