*RAT-IN-MAZE-Problem*

|  |  |
| --- | --- |
| **Name** | **ID** |
| احمد اشرف محمد حسين | 202000014 |
| احمد هشام محمد | 202000093 |
| اسامة سيد مجاهد امين | 202000107 |
| نادين عيد رمضان محمود | 202000977 |
| عمر محسن احمد عبدالغنى | 202000606 |

**Problem 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.

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

* You should enter the dimensions of the maze, then a grid is generated.
* You should use 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.

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.

**Maze description:**

* The maze’s width is equal to maze’s high.
* The maze’s start point is always at the top right corner I.e., (0,0).
* The maze’s end point is always at the down left corner I.e., (N-1, N-1).

**How the algorithm works:**

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

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.A picture containing text, crossword puzzle

Description automatically generated

0

1

2

3

4

5

6

7

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.

We have one special case:

if the rat decides to take a specific path when there is more than one path to take and this path turned to be a dead end, so the rat will take several steps backwards to the nearest step that has more than one path and record that this direction is blocked and take the other path.

**Division of labor :**

Ahmed Ashraf ----------------------------------- Maze

Ahmed Hisham ----------------------------------- Gred

Nadien Eid ----------------------------------- Rat run forward

Omar Mohsen ----------------------------------- Rat run down

Osama Sayed ----------------------------------- GUI& Documentation

A close-up of a flower

Description automatically generated with medium confidence