Operating System Project Documentation

Members Names:

اسامه وسام اسامه جاسر	20210144
خلود صادق محمد صلاح	20210306
زياد ايمن الحسيني احمد	20210360
إسراء شعبان عبدالجواد سالم	20210152
زياد اسامه فتحي	20210357
زياد اشرف عبدالرحمن	20210358

Project Name:

Project 3: Rat in a Maze Problem

Project Description:

User enter the dimensions of the maze(N), A Maze is given as $N \times N$ 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.
n 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.
The Rat when find 2 ways (forward & down) will create 2 Threads the first thread will begin from the forward block and the other thread will begin from the down block and the Thread1 will wait until the other threads finished .
But If find1 way just The Thread1 will continue in this way .
When Thread arrived to the destination block the other will stop and game is end.

Team members role:

Esraa Shaaban -> Interface GUI (interface class) & multithreading (DepthFirst class) (
Ziad Ayman ->Maze draw GUI (mazee class & maze class)
Kholod Sadek, Osama Wesam -> Algorithm of the code (DFS Algorithm & maze class) & Document & Video
Ziad Ashraf, Ziad Osama -> Move the ball animation on the maze (main class) & Testing The code

Code Documentation:

1- Mazee Class:

This class is define the dead block and the initial state and destination state and the border of the game so:
Maze[1][1] = 0; //initial block
Maze[n][n] = 9; //end block
Maze[x][y] = 2; //visited block
Maze[x][y] = 1; //is dead block

```
public class Mazee {
    public static int [][] numOfMaze(int n){


  Random rand = new Random();

   int [][] maze = new int[n+2][n+2];

      for(int i = 0; i < maze.length ; i++) {
          for (int j = 0; j < maze.length ; j++) {
              maze[i][j] = rand.nextInt(bound: 2);
              if((i==1 && j==1)||
                      (i==1 && j==3) || (i==1 && j==5) ||
                      (i==1 && j==6) || (i==1 && j==7) ||
                      (i==3 && j==4) || (i==3 && j==5) ||
                      (i==4 && j==1) || (i==4 && j==2)||
                      (i==4 && j==5) || (i==5 && j==2) ||
                      (i==5 && j==4) || (i==5 && j==5) ||
                      (i==6 && j==0) || (i==8 && j==3)||
                      (i==9 && j==3) || (i==10 && j==3) ||
                      (i==2 && j==9) || (i==2 && j==10) ||
                      (i==3 && j==9) || (i==3 && j==10) ||
                      (i==4 && j==9) || (i==4 && j==10) ||
                      (i==8 && j==8) || (i==9 && j==7) ||
                       (i==9 && j==8) || (i==9 && j==9) ||
                       (i==11 && j==0) || (i==11 && j==5) ) {
                  maze[i][j]= 1;
              }
              else{
                  maze[i][j]=0;
              }

          }
       }
      for (int i = 0; i < n + 2; i++) {
          maze[i][0] = 1;     // Set left border to 1
          maze[i][n + 1] = 1; // Set right border to 1
       }

       for (int j = 0; j < n + 2; j++) {
          maze[0][j] = 1;     // Set top border to 1
          maze[n + 1][j] = 1; // Set bottom border to 1
       }
      maze [1][1]=0;
      maze [n][n]=9;

      return maze;
  }

}
```

2- Interface class:

In this class will implement our interface GUI of the game
We put the Rat icon and Button Enter to enter the size of the maze and there is label to write on the size.

3- DepthFirst class:

Here we Implement the Multithreading to move in the maze we have 3 cases If the Thread1 has 2 ways (forward & down) so will create 2 threads the Thread2 will continue in forward way and the other Thread3 will continue in down way and If the Thread1 has 1 way will continue in its way and if arrived to destination state all the Threads will stop and the game will finish.

```
public class DepthFrist implements Runnable{

    private int[][] maze;
    private int x;
    private int y;
    private List<Integer> path;
    private List<Integer> path1;

    public DepthFrist(int[][] maze, int x, int y, List<Integer> path) {
        this.maze = maze;
        this.x = x;
        this.y = y;
        this.path = path;
    }
    @Override
    public void run() {
         searchPath(maze, x, y, path);

       if((searchPath(maze, x , y + 1, path)) && (searchPath(maze, x + 1, y, path))){
                path.add(e: x);
                path.add(e: y);
                DepthFrist th1 = new DepthFrist(maze, x, y+1, path);
                Thread t1 = new Thread(task: th1);
                t1.start();
            try {
                t1.join();
            } catch (InterruptedException ex) {
            } catch (InterruptedException ex) {
                Logger.getLogger(name: DepthFrist.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
            }
               DepthFrist th2 = new DepthFrist(maze, x+1, y, path);
               Thread t2 = new Thread(task: th2);
               t2.start();
            try {
                t2.join();
            } catch (InterruptedException ex) {
                Logger.getLogger(name: DepthFrist.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
            }
           }
       else{
           searchPath(maze, x, y, path);
       }
        }
```

SearchPath() function
This function whose create the cases and check if the thread arrived to the destination state and check if the thread go to the block before or no and if the

bock doesn't visit the block make it visit and check on it the 3 cases that I explained in the above and make it visit in the end of the case.

```
public static boolean searchPath(int[][] maze, int x, int y, List<Integer> path ) {
    if (maze[y][x] == 9) {
        path.add(e: x);
        path.add(e: y);
        return true;
    }
    if (maze[y][x] == 0) {
        if((searchPath(maze, x , y + 1, path)) && (searchPath(maze, x + 1, y, path))){
            path.add(e: x);
            path.add(e: y);
            return true;
        }

        if (searchPath(maze, x , y+1 , path)) {
            path.add(e: x);
            path.add(e: y);
            return true;
        }
        if (searchPath(maze, x+1 , y, path)) {
            path.add(e: x);
            path.add(e: y);
            return true;
        }
         maze[y][x] =2;
    }
```

4- Maze Class:

Here will Implement the maze GUI and color the threads and the deadblocks Thread color is green and the deadblocks is black and incase 2 threads walked in the same block its color of this block is red .

```
public class Maze extends JFrame implements ActionListener {

    private int[][] maze;
    private String[][] pathc;
    private int pathIndex;
     private int threadX = 1; // Starting position of the thread
    private int threadY = 1;
    private Timer timer;

    public List<Integer> path = new ArrayList<>();
    public List<Integer> path1 = new ArrayList<>();
    public ArrayList<Boolean> visited = new ArrayList<>();

    public Maze(int n) {
        setTitle(title: "Maze");
        setSize(width: 480, height: 480);
        setLocationRelativeTo(c: null);
        setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);

        maze = Mazee.numOfMaze(n);
        pathc = new String[maze.length][maze[0].length];

        timer = new Timer(delay: 500, listener: this); // Timer for animation
        timer.start();

  ExecutorService executor = Executors.newFixedThreadPool(nThreads: 4);
```

```
        executor.submit(() -> {
            DepthFrist thl = new DepthFrist(maze, x: 1, y: 1, path);
                    Thread tl = new Thread(task: thl);
                    tl.start();
        });

        executor.shutdown();


    }
private int thread2X=1; // X-coordinate for the second paint
private int thread2Y=2; // Y-coordinate for the second paint
    @Override
    public void actionPerformed(ActionEvent e) {
        int[] dx = {1, 0, -1, 0}; // Possible movements in x-direction
        int[] dy = {0, 1, 0, -1}; // Possible movements in y-direction
        int[] dx1 = {0, 1, -1, 0}; // Possible movements in x-direction
        int[] dy1 = {1, 0, 0, -1}; // Possible movements in y-direction


    // Check the available neighboring cells
    for (int i = 0; i < 4; i++) {
        int nextX = threadX + dx[i];
        int nextY = threadY + dy[i];

        // Check if the neighboring cell is within the maze boundaries and is part of the available path
        if (nextX >= 0 && nextX < maze[0].length && nextY >= 0 && nextY < maze.length
                && maze[nextY][nextX] == 0 || maze[nextY][nextX] == 9) {
            threadX = nextX;
            threadY = nextY;
            break;
        }
    }
     // Check if the first paint has reached the end point
    if (threadX == maze.length - 2 && threadY == maze.length - 2) {
        timer.stop();
    }
    for (int i = 0; i < 4; i++) {
        int nextX = thread2X + dx1[i];
        int nextY = thread2Y + dy1[i];

        // Check if the neighboring cell is within the maze boundaries and is part of the available path
        if (nextX >= 0 && nextX < maze[0].length && nextY >= 0 && nextY < maze.length
                && maze[nextY][nextX] == 0 || maze[nextY][nextX] == 9) {
            thread2X = nextX;
            thread2Y = nextY;
            break;
        }
    }

        repaint(); // Update the display
    }
      @Override
      public void paint(Graphics g) {
          super.paint(g);

          g.translate(x: 50, y: 50);

          // Draw the maze
          for (int row = 0; row < maze.length; row++) {
              for (int col = 0; col < maze[0].length; col++) {
                  Color color;
                  switch (maze[row][col]) {
                      case 1:
                          color = Color.BLACK;
                          break;
                      case 9:
                          color = Color.ORANGE;
                          break;
                      case 2:
                          color = Color.RED;
                          break;
                      case 4:
                          color = Color.GREEN;
                          break;
                      default:
                          color = Color.WHITE;
                  }
                  g.setColor(c: color);
                  g.fillRect(30 * col, 30 * row, width: 30, height: 30);
```

```
                }
                g.setColor(c: color);
                g.fillRect(30 * col, 30 * row, width: 30, height: 30);
                g.setColor(c: Color.GRAY);
                g.drawRect(30 * col, 30 * row, width: 30, height: 30);
            }
        }

    // Draw the solved path
    for (int p = 0; p < path.size(); p += 2) {
        int pathX = path.get(index: p);
        int pathY = path.get(p + 1);
        g.setColor(c: Color.GREEN);
        g.fillRect(pathX * 30, pathY * 30, width: 30, height: 30);
         g.setColor(c: Color.blue);
         g.fillOval(threadX * 30, threadY * 30, width: 30, height: 30);
         g.setColor(c: Color.orange);
         g.fillOval(thread2X * 30, thread2Y * 30, width: 30, height: 30);
    }

    // Stop the animation when the thread reaches the end of the path
    if (thread2X == maze.length-2 && thread2Y == maze.length-2) {
        timer.stop();
    }
        }
```

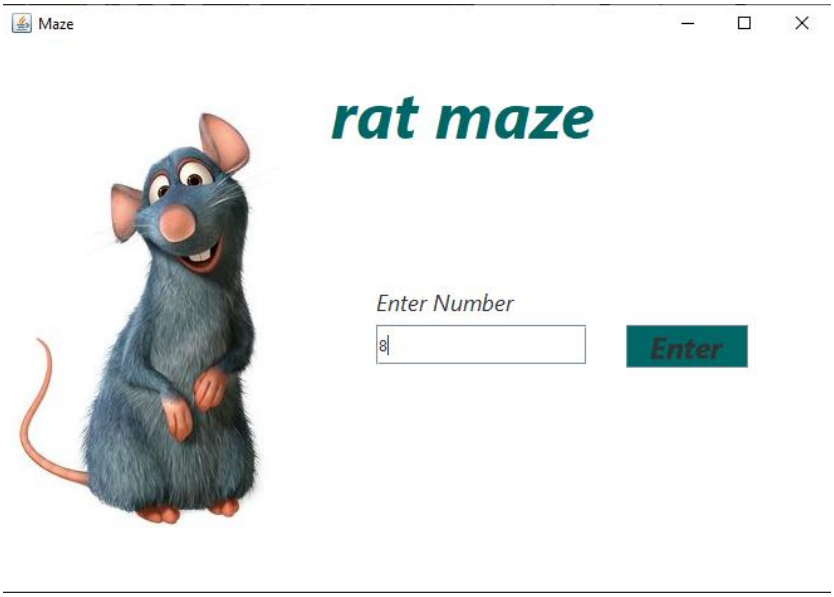
The Main:

```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
              InterFace numInterface = new InterFace();
              numInterface.setVisible(b: true);


            }
        });
    }
}
```

Here is Output :

First : Interface GUI:
To take the size of the maze



Second Solve Maze GUI:
Here is maze solve by threads and this dynamic balls go in the 2 correct path and will see which ball will arrive first so here is the thread who arrived first .