Zagazig University
Faculty of Engineering
Computer and Systems Engineering Dept.

Maze Solving

Using BFS And DFS

This document is the report concerning the project done in the winter semester 2016 by our group. The group was composed of 4 students who studied Computer and Systems Engineering on the 4th year at Zagazig University, Faculty of Engineering.

Zagazig, Sunday, Feb 5, 2017.

Participants (alphabetically ordered):

Name	ID	Signature
Ahmed Mahmoud Ali	15	
Anas Ahmed Fouad	16	
Menna-Allah Ashraf	5	
Mustafa Kamel El-Sayed	27	

Abstract:

In this project we aimed to solve any Maze using Breads First Search or Depth First Search.

The Program is written in JAVA and the results is represented using GUI instead AND CL interface.

Both are shown when user attempts to solve the maze via the program.

5 button are used by the GUI, first button is “Solve DFS” used to solve using Depth First Search.
Second button is “Solve BFS” used to solve using Depth First Search.
Third button is “Clear” used to clear any traces of previous attempts to solve the maze.
Fourth button is “Exit” used to close the GUI interface and Halt the program.
Fifth button is “Generate Random Maze” used to clear the maze screen and generate a new maze that may or may not be solvable.

Keywords:

Artificial intelligence, Programming, Java, GUI, Command Line

Zagazig University
Faculty of Engineering
Computer and Systems Engineering Dept.

Maze Solving

Using BFS And DFS

Project Report
by

Ahmed Mahmoud Ali
Anas Ahmed Fouad
Mahmoud El-Sayed Hussein
Menna-Allah Ashraf
Mustafa Kamel El-Sayed
Sultan Ibrahim Ibrahim

Source Code

```
package maze;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.URL;
import java.util.LinkedList;
import java.util.Random;
import java.util.Stack;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class Maze extends JFrame {

    //block (black square)
    final static int X = 1;
    //free space   (white Square)
    final static int C = 0;
    //initial state
    final static int S = 2;
    //goal
    final static int E = 8;
    // the path
    final static int V = 9;

    //initial state   (i,j)
    final static int START_I = 1, START_J = 1;
    //goal  (i,j)
    final static int END_I = 2, END_J = 9;

    int[][] maze = new int[][]{ // array for the maze
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {1, 2, 0, 0, 0, 0, 0, 0, 0, 1},
        {1, 0, 0, 0, 1, 0, 1, 1, 0, 8},
        {1, 0, 1, 1, 1, 0, 1, 1, 0, 1},
        {1, 0, 0, 0, 0, 1, 1, 1, 0, 1},
        {1, 1, 1, 1, 0, 1, 1, 1, 0, 1},
        {1, 1, 1, 1, 0, 1, 0, 0, 0, 1},
        {1, 1, 0, 1, 0, 1, 1, 0, 0, 1},
        {1, 1, 0, 0, 0, 0, 0, 0, 0, 1},
        {1, 1, 1, 1, 1, 1, 0, 1, 1, 1}
    };

    // for random array
```

```
    int[][] arr;

    // Buttons For GUI
    JButton solveStack;
    JButton solveBFS;
    JButton clear;
    JButton exit;
    JButton genRandom;

    // JLabel and JTextField For GUI
    JLabel elapsedDfs;
    JTextField textDfs;
    JLabel elapsedBFS;
    JTextField textBFS;

    boolean repaint = false;

    // start time
    long startTime;
    //stop time
    long stopTime;
    //calculate the elapsed time
    long duration;
    //time for DFS
    double dfsTime;
    //time for BFS
    double bfsTime;

   // take copy of the original maze
   int[][] savedMaze = clone();


    public Maze() {

        setTitle("Maze");     //Title For JFrame
        setSize(960, 530);    // Size For JFrame  (width,height)


        URL urlIcon = getClass().getResource("flat-theme-action-maze-
icon.png");    // Path for image for The JFrame
        ImageIcon image = new ImageIcon(urlIcon);
        setIconImage(image.getImage());     // set The Image For JFrame

        setLocationRelativeTo(null);       // JFrame or The app Show in the
Middle of the screen
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);    // To close the
app when click on exit (X)
        setLayout(null);     // to set the position of the components  by
myself

        // initialize objects for Jlabel and JTextField
        elapsedDfs = new JLabel("Elapsed Time :");
        elapsedBFS = new JLabel("Elapsed Time :");
        textDfs = new JTextField();
        textBFS = new JTextField();

        // initialize objects for Buttons
        solveStack = new JButton("Solve DFS");
        solveBFS = new JButton("Solve BFS");
        clear = new JButton("Clear");
        exit = new JButton("Exit");
```

```
        genRandom = new JButton("Generate Random Maze");

        // Add The Buttons to JFrame
        add(solveStack);
        add(solveBFS);
        add(clear);
        add(elapsedDfs);
        add(textDfs);
        add(elapsedBFS);
        add(textBFS);
        add(exit);
        add(genRandom);

        // make JFrame visible
        setVisible(true);

        // set the positions of the components on the JFrame
(x,y,width,height)
        solveStack.setBounds(500, 50, 100, 40);
        solveBFS.setBounds(630, 50, 100, 40);
        clear.setBounds(760, 50, 100, 40);
        exit.setBounds(760, 115, 100, 40);
        elapsedDfs.setBounds(500, 100, 100, 40);
        genRandom.setBounds(500, 180, 170, 40);
        elapsedBFS.setBounds(630, 100, 100, 40);
        textDfs.setBounds(500, 130, 100, 25);
        textBFS.setBounds(630, 130, 100, 25);

        // what happen when click on Generate Random Maze Button
        genRandom.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                int x[][] = GenerateArray();  // generate random array and
store in x
                repaint = true;
                restore(x);   // make the array maze equal the array x  ||
to show the array x on screen
                repaint();   // repaint the maze on the JFrame
            }
        });

        // what happen when click on Exit Button
        exit.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);        //Close The App
            }
        });

        // what happen when click on Clear Button
        clear.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {

                if (arr == null) {
                    repaint = true;
                    restore(savedMaze);  // restore the maze to the
original
```

```
                    repaint();           // repaint the maze on the JFrame
                } else {
                    repaint = true;
                    restore(arr);     // to make the array arr equal to the
array maze || to show the array arr on the screen
                    repaint();        // repaint the maze on the JFrame
                }

                textBFS.setText("");       // clear the text in JTextField
                textDfs.setText("");       // clear the text in JTextField

            }
        });

        // what happen when click on Solve DFS Button
        solveStack.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                if (arr == null) {
                    restore(savedMaze);
                    repaint = false;
                    solveStack();
                    repaint();
                } else {

                    restore(arr);
                    repaint = false;
                    solveStack();
                    repaint();

                }

            }
        });

        // what happen when click on Solve BFS Button
        solveBFS.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {

                if (arr == null) {
                    restore(savedMaze);
                    repaint = false;
                    solveQueue();
                    repaint();
                } else {
                    restore(arr);
                    repaint = false;
                    solveQueue();
                    repaint();
                }

            }
        });

    }

    // size of the maze
    public int Size() {
```

```
        return maze.length;
    }

    // Print the Maze
    public void Print() {
        for (int i = 0; i < Size(); i++) {
            for (int J = 0; J < Size(); J++) {
                System.out.print(maze[i][J]);
                System.out.print(' ');
            }
            System.out.println();
        }
    }

    //return true if cell is within maze
    public boolean isInMaze(int i, int j) {

        if (i >= 0 && i < Size() && j >= 0 && j < Size()) {
            return true;
        } else {
            return false;
        }
    }

    public boolean isInMaze(MazePos pos) {
        return isInMaze(pos.i(), pos.j());
    }

    // to mark the node in the array with value 9
    public int mark(int i, int j, int value) {
        assert (isInMaze(i, j));  // it is used for test.if the condition
is false it will throw an error named AssertionError.
        int temp = maze[i][j];
        maze[i][j] = value;
        return temp;
    }

    public int mark(MazePos pos, int value) {
        return mark(pos.i(), pos.j(), value);
    }

    // return true if the node equal to 9
    public boolean isMarked(int i, int j) {
        assert (isInMaze(i, j));
        return (maze[i][j] == V);

    }

    public boolean isMarked(MazePos pos) {
        return isMarked(pos.i(), pos.j());
    }

    // return true if the node is equal to 0
    public boolean isClear(int i, int j) {
        assert (isInMaze(i, j));
        return (maze[i][j] != X && maze[i][j] != V);

    }

    public boolean isClear(MazePos pos) {
        return isClear(pos.i(), pos.j());
```

```
    }

    // to make sure if it is reach the goal
    public boolean isFinal(int i, int j) {

        return (i == Maze.END_I && j == Maze.END_J);
    }

    public boolean isFinal(MazePos pos) {
        return isFinal(pos.i(), pos.j());
    }

    // make Copy from the original maze
    public int[][] clone() {
        int[][] mazeCopy = new int[Size()][Size()];
        for (int i = 0; i < Size(); i++) {
            for (int j = 0; j < Size(); j++) {
                mazeCopy[i][j] = maze[i][j];
            }
        }
        return mazeCopy;
    }

    // make the coming array equal to the array maze
    // to restore the maze to the initial state
    public void restore(int[][] savedMazed) {
        for (int i = 0; i < Size(); i++) {
            for (int j = 0; j < Size(); j++) {
                maze[i][j] = savedMazed[i][j];
            }
        }

        maze[1][1] = 2;
        maze[2][9] = 8;
    }

    //generate random maze whith values 0 and 1
    public int[][] GenerateArray() {
        arr = new int[10][10];
        Random rnd = new Random();
        int min = 0;
        int high = 1;

        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                int n = rnd.nextInt((high - min) + 1) + min;
                arr[i][j] = n;

            }
        }

//        for(int i=0; i<10;i++){
//           for(int j=0;j<10;j++){
//               System.out.print(arr[i][j]);
//               System.out.print(' ');
//            }
//           System.out.println("");
//          }
        return arr;
    }
```

```
    //draw the maze on the JFrame
    @Override
    public void paint(Graphics g) {
        super.paint(g);
        g.translate(70, 70);      //move the maze to begin at 70 from x and
70 from y

        // draw the maze
        if (repaint == true) {
            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.darkGray;       // block
                            break;
                        case 8:
                            color = Color.RED;          // goal
                            break;
                        case 2:
                            color = Color.YELLOW;      //initial state
                            break;
                        //   case '.' : color=Color.ORANGE; break;
                        default:
                            color = Color.WHITE;        // white free space
0
                    }
                    g.setColor(color);
                    g.fillRect(40 * col, 40 * row, 40, 40);    // fill
rectangular with color
                    g.setColor(Color.BLUE);
                    g.drawRect(40 * col, 40 * row, 40, 40);    //draw
rectangular with color

                }
            }
        }

        if (repaint == false) {
            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.darkGray;
                            break;
                        case 8:
                            color = Color.RED;
                            break;
                        case 2:
                            color = Color.YELLOW;
                            break;
                        case 9:
                            color = Color.green;   // the path from the
initial state to the goal
                            break;
                        default:
                            color = Color.WHITE;
                    }
                    g.setColor(color);
                    g.fillRect(40 * col, 40 * row, 40, 40);
```

```
                g.setColor(Color.BLUE);
                g.drawRect(40 * col, 40 * row, 40, 40);

            }

        }

    }

}

public static void main(String[] args) {

    SwingUtilities.invokeLater(new Runnable() {

        @Override
        public void run() {
            Maze maze = new Maze();

        }
    });

}

public void solveStack() {
    // start of the time
    startTime = System.nanoTime();

    //stack of MazPos
    Stack<MazePos> stack = new Stack<MazePos>();

    //insert the start
    stack.push(new MazePos(START_I, START_I));

    MazePos crt;   //current node
    MazePos next;  //next node
    while (!stack.empty()) {

        //get current position
        crt = stack.pop();
        if (isFinal(crt)) {

            break;
        }

        //mark the current position
        mark(crt, V);

        //put its neighbours in the queue
        next = crt.north();    // go up
        if (isInMaze(next) && isClear(next)) {
            stack.push(next);
        }
        next = crt.east();    //go right from the current node
        if (isInMaze(next) && isClear(next)) {
            stack.push(next);
        }
        next = crt.west();    //go left from the current node
        if (isInMaze(next) && isClear(next)) {
            stack.push(next);
        }
```

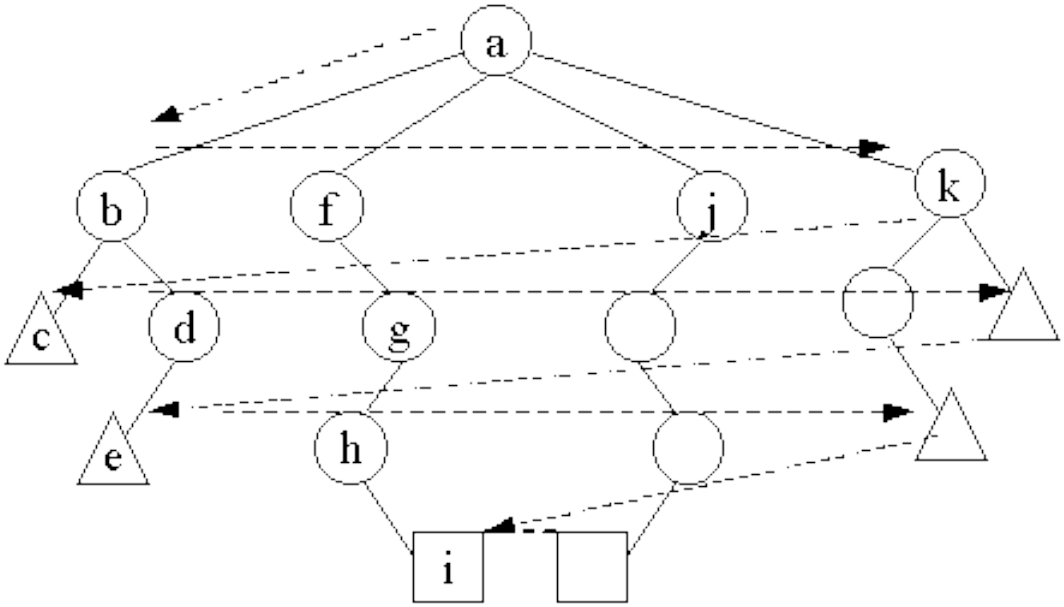
```
            next = crt.south();  // go down from the current node
            if (isInMaze(next) && isClear(next)) {
                stack.push(next);
            }
        }

        if (!stack.empty()) {
             stopTime = System.nanoTime();
            JOptionPane.showMessageDialog(rootPane, "You Got it");

        } else {
            JOptionPane.showMessageDialog(rootPane, "You Are stuck in the
maze");
        }

        System.out.println("\nFind Goal By DFS : ");
        Print();
        // stop time

        duration = stopTime - startTime;    //calculate the elapsed time

        dfsTime =  (double)duration / 1000000;   //convert to ms
        System.out.println(String.format("Time %1.3f ms", dfsTime));

        textDfs.setText(String.format("%1.3f ms", dfsTime));

    }

    public void solveQueue() {
        //start the timer
        startTime = System.nanoTime();

        //list from MazePos
        LinkedList<MazePos> list = new LinkedList<MazePos>();

        // add initial node to the list
        list.add(new MazePos(START_I, START_J));

        MazePos crt, next;
        while (!list.isEmpty()) {

            //get current position
            crt = list.removeFirst();

            // to be sure if it reach the goal
            if (isFinal(crt)) {
                break;
            }

            //mark the current position
            mark(crt, V);

            //put its neighbors in the queue
            next = crt.north();    //move up
            if (isInMaze(next) && isClear(next)) {
                list.add(next);
            }
            next = crt.east();    //move right
            if (isInMaze(next) && isClear(next)) {
                list.add(next);
            }
```

```
             next = crt.west();    //move left
            if (isInMaze(next) && isClear(next)) {
                list.add(next);
            }
            next = crt.south();   //move down
            if (isInMaze(next) && isClear(next)) {
                list.add(next);
            }

        }

        if (!list.isEmpty()) {
            stopTime = System.nanoTime();    //stop the timer
            JOptionPane.showMessageDialog(rootPane, "You Got it");
        } else {
            JOptionPane.showMessageDialog(rootPane, "You Are stuck in the
maze");
        }

        System.out.println("\nFind Goal By BFS : ");
        Print();

        duration = stopTime - startTime;      //calculate the elapsed time

        bfsTime = (double) duration / 1000000;     //convert to ms
        System.out.println(String.format("Time %1.3f ms", bfsTime));

        textBFS.setText(String.format("%1.3f ms", bfsTime));


    }

}
```

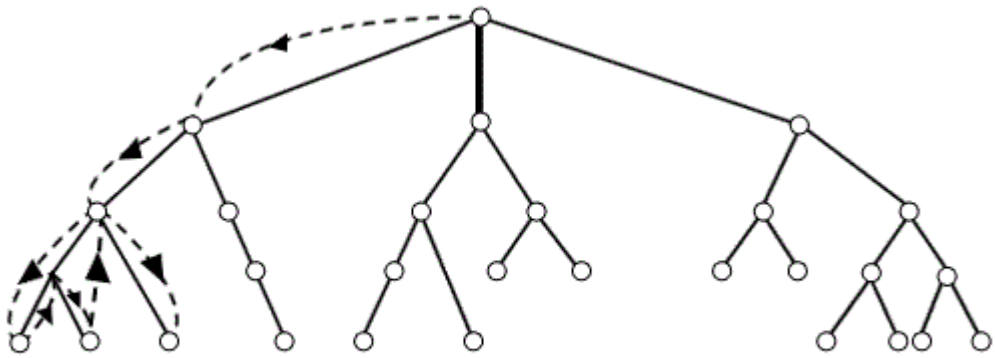
How does it work?

- The program consist of two main parts:
  - The GUI coding.
  - The BFS and DFS algorithm.
- The GUI consist of five buttons and 2 text boxes.
  - Buttons:
    1. *The first button is "Solve DFS" used to solve using Depth First Search.*
    2. *The second button is "Solve BFS" used to solve using Depth First Search.*
    3. *The third button is "Clear" used to clear any traces of previous attempts to solve the maze.*
    4. *The fourth button is "Exit" used to close the GUI interface and Halt the program.*
    5. *The fifth button is "Generate Random Maze" used to clear the maze screen and generate a new maze that may or may not be solvable.*
  - Text boxes:
    1. *The First is the Elapsed time of DFS.*
    2. *The Second is the Elapsed time of BFS.*

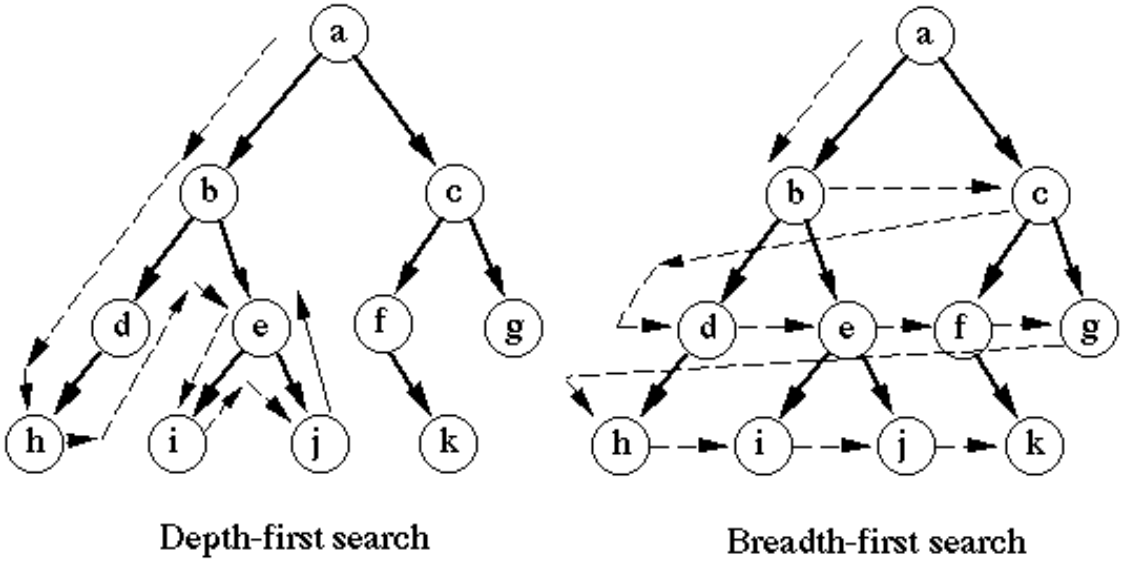
- The two algorithms are encapsulated in solveStack() and solveQueue().
- The BFS explore the nodes in a horizontal order



- The DFS on the other hand explore the nodes in a vertical order.

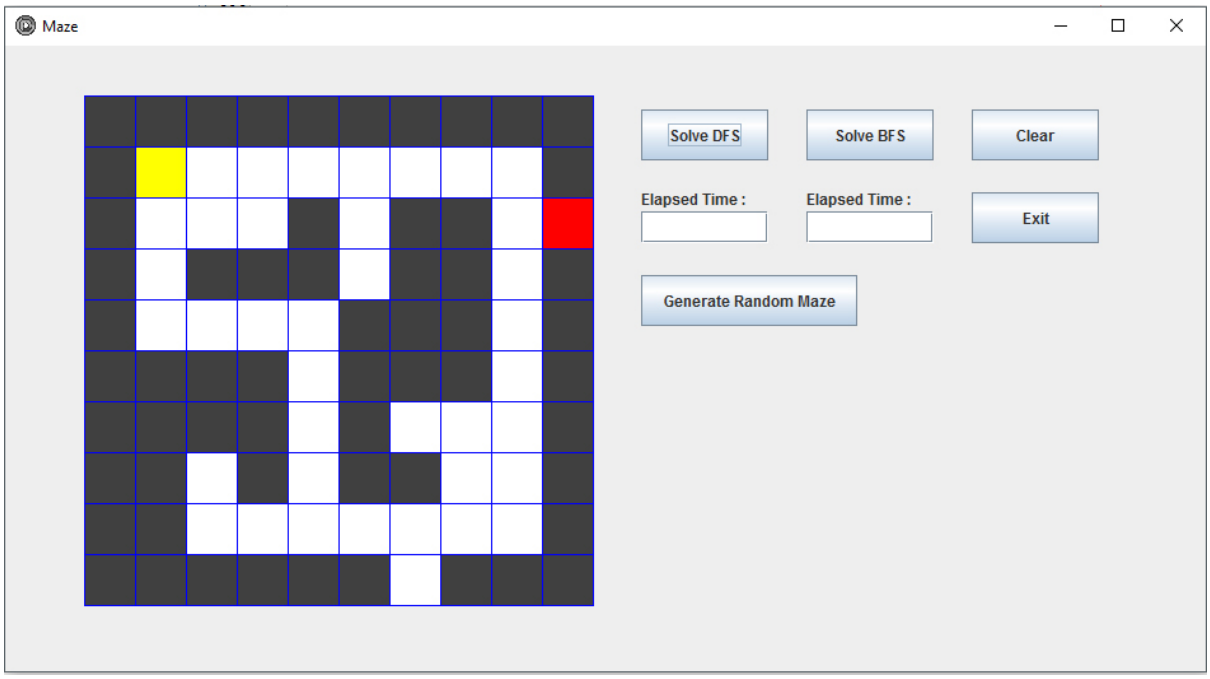


- For illustration purposes here is a visual comparison:

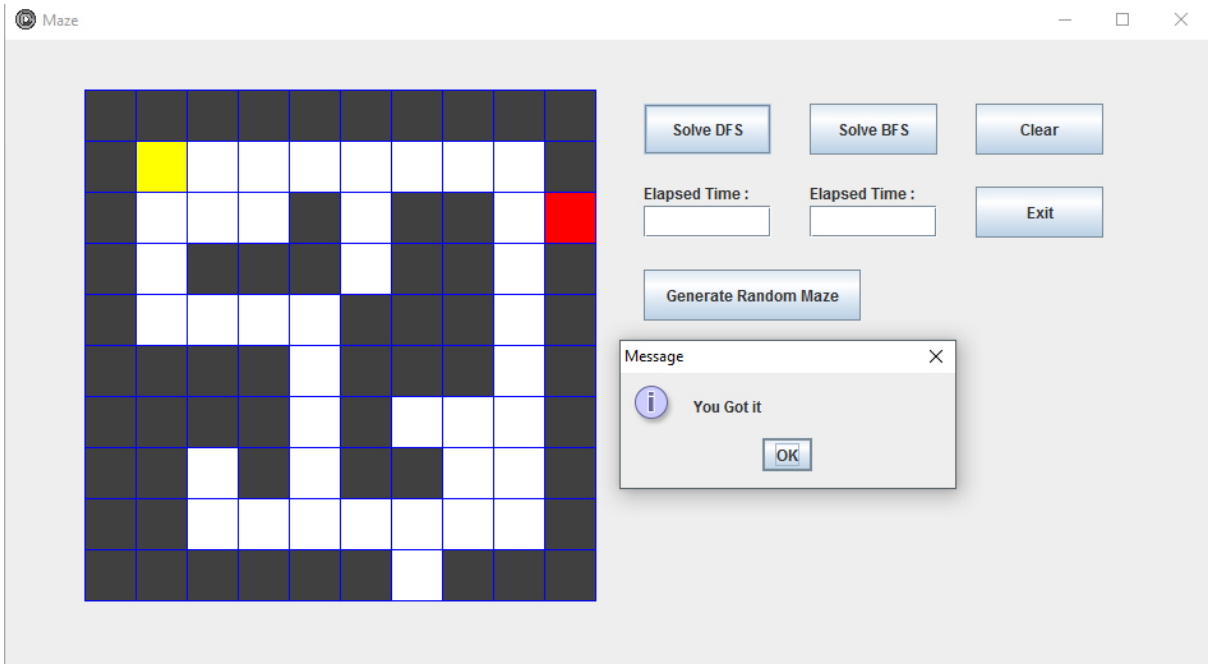


- The two methods solveQueue() and solveQueue() are java code representation of the two algorithms.
- The code is well documented so you can refer back to the comments to understand how everything fit together.
- As for the GUI is pretty straightforward.

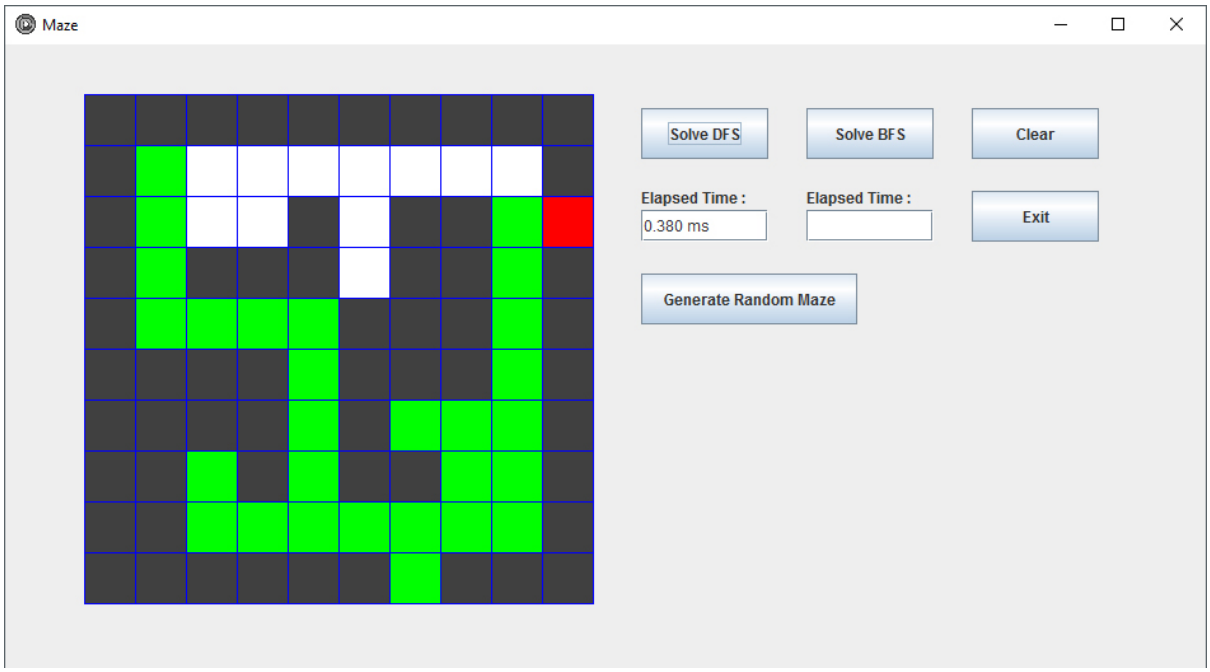
- Just take a look at the GUI interface below:



- As we mentioned before we have 5 buttons and 2 text boxes.
- We also mentioned every button and text box purpose.
- So if we pressed “Solve DFS” button:

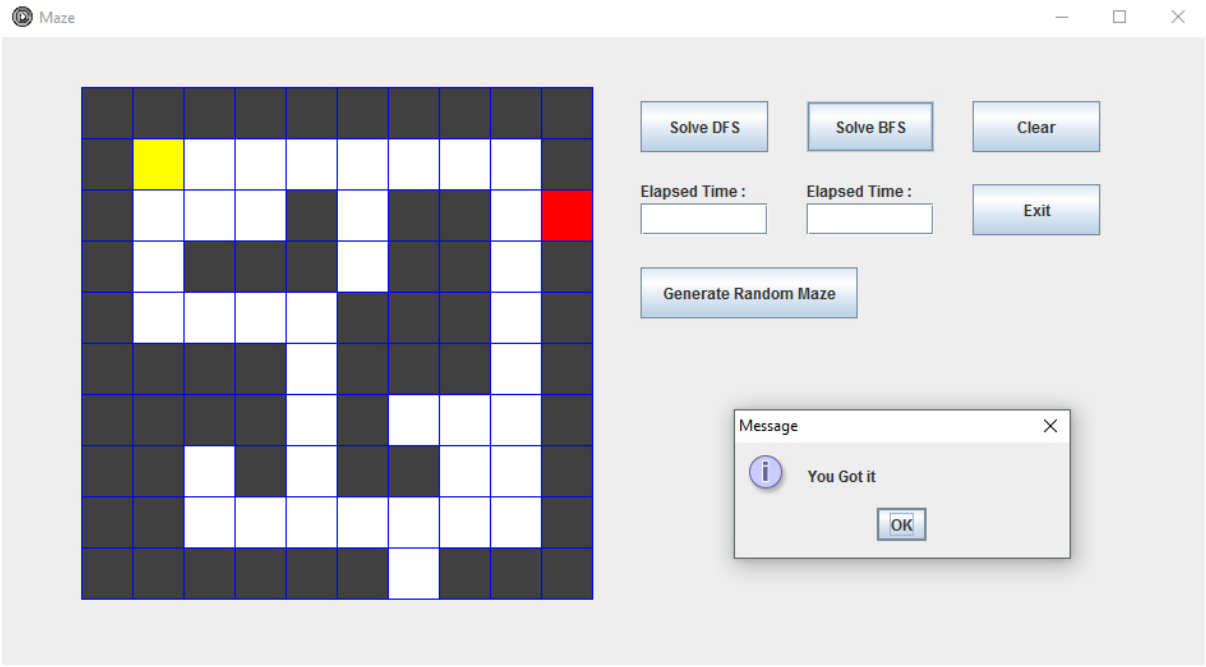


- You get a message “You Got it” which means a solution was found, the message is an inside joke between us.
- The green boxes indicate the nodes that the algorithm explored, white means “move is permitted” or “legal move”, black means “move is NOT permitted” or “illegal move” and red is the goal.

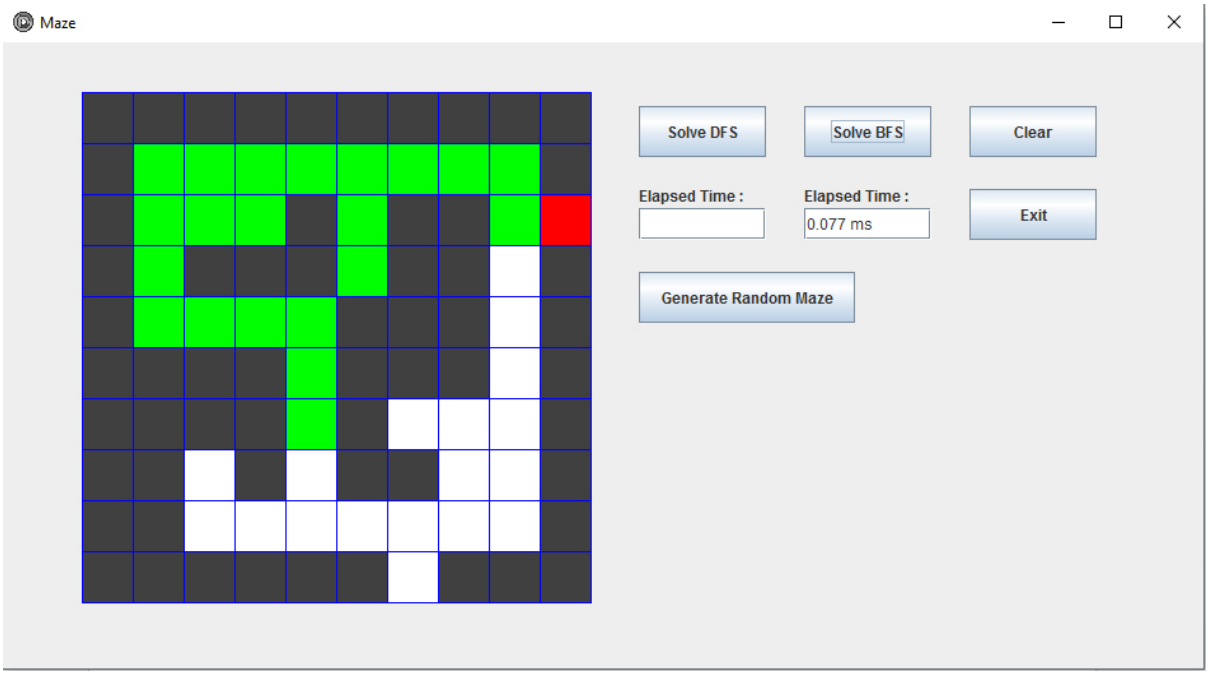


- As you can see the DFS elapsed time is 0.380 ms.

➢ Now if we pressed “Clear” then “Solve BFS” button:

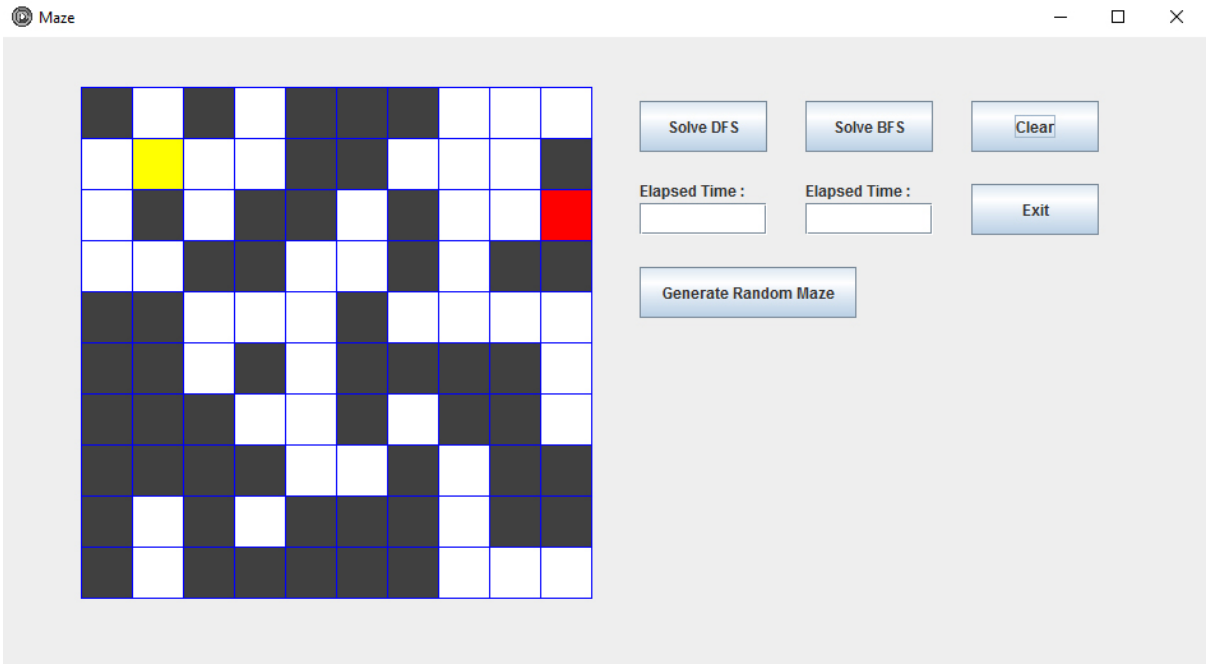


➢ You get a message “You Got it” which means a solution was found.



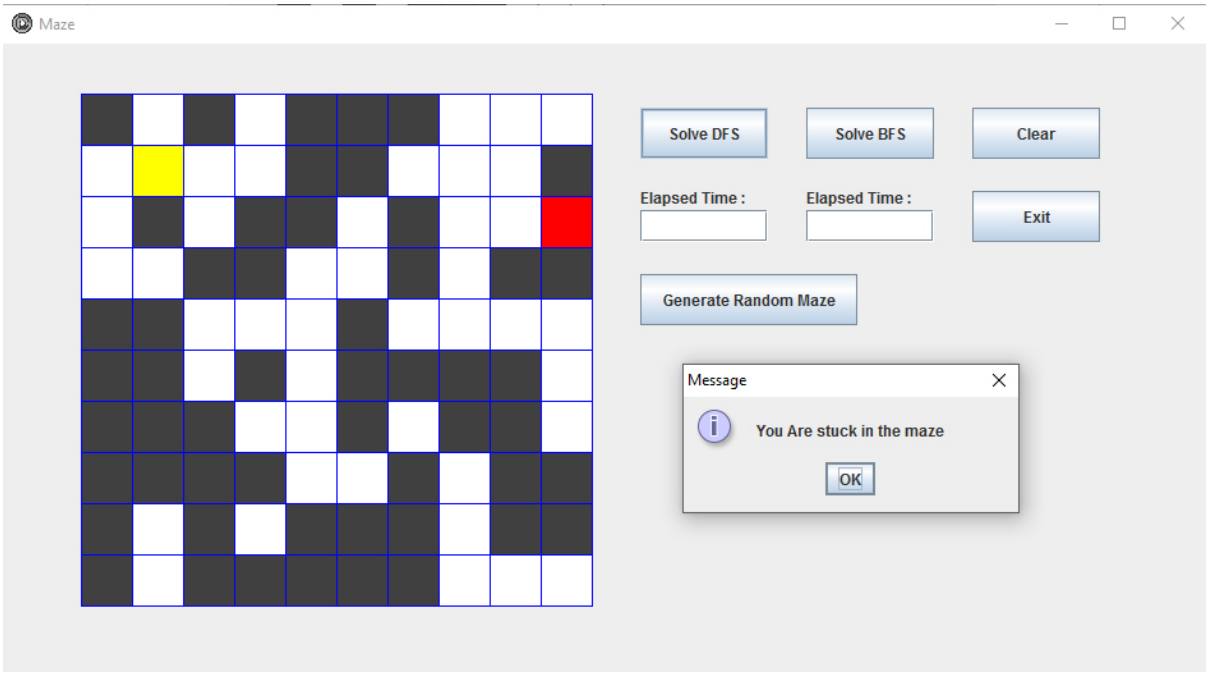
➢ And the Time elapsed is 0.077 ms.

➢ If we tried to “Generate Random Maze” :



➢ You can predict from the Maze layout that this maze has no possible solution.

➢ So if we tried to solve it using DFS or BFS we get this message.



- You get the message “You Are stuck in the maze” which indicate that the Algorithm didn’t find a solution for the maze.
- We deliberately wanted unsolvable mazes to happen to make sure the Algorithms can handle any possible layout for the maze.
- This sums up the program.
- We’re happy to answer any question you may have.