**MAZE SOLVER**

# (JAVA PROJECT)

# INTRO

# *GROUP MEMBERS:*

PRITHVIRAJ PATIL -501850

ARNAV PANDEY -501847

PRANAV PATIL -501849

THIS PROJECT IS ABOUT CREATING AND SOLVING THE MAZE.BFS ALGORITHM IS USED FOR SEARCHING WHERE BFS MEANS BREADTH FIRST SEARCH.BREADTH FIRST SEARCH IS ONE OF THE GRAPH TRAVERSAL METHOD. THIS GIVES THE **SHORTEST PATH** FROM ENTRY TO THE EXIT BY BACKTRACKING THE ORIGINS OF THE NODES.IF PATH DOESNOT EXIST IT WILL SHOW MESSEGE “PATH NOT FOUND”.

TO MAKE THIS PROJECT, THE FOLLOWING PROGRAMMING TOOLS ARE USED:

* ECLIPSE IDE

# DESIGN

FRAME 1 – IT IS ABOUT TAKING INPUT OF ROWS AND COLUMNS FROM USER .IT ALSO PROVIDES

MANUAL FOR USER TO UNDERSTAND PURPOSE OF EACH BUTTON.

FRAME 2 –IT PROVIDES USER A EMPTY GROUND WITH NO WALLS NO GATES AT INITIAL

BY USING DIFFERENT BUTTONS DIFFERENT ACTIONS LIKE GIVING GATES AND WALLS OR

REMOVING WALLS AND GATES CAN BE DONE.AFTER MAKING A MAZE USER CAN GET

SHORTEST PATH FROM ENTRY GATE TO EXIT GATE.

* Packages like Swing,util,awt are used in this project.For paint purpose class Graphics is used from awt package .For action listeners for all the buttons ActionListener interface from awt.event package is used.
* Basically it works on the principle of 2D array where grass denotes 0(zero) in the matrix.Walls denote 1(one) in the matrix . entry gate denotes 3(three) in the matrix and exit gate denotes 4(four) in the matrix.
* BFS always moves horizontally and visit nodes. BFS algorithm will serch for the exit node from start node and when we get the exit node it backtracks the whole path by keeping 2 at that positions in the matrix. The path is denoted by stones on the screen.

# CODE

## FRAME 1

*/\*THIS FRAME IS FOR TAKING INPUT OF ROWS AND COLUMNS FROM USER AND CONTAINS MANUAL FOR SOME INSTRUCTIONS\*/*

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.\*;

class Example2 extends JFrame {

JLabel l3,background;

JTextField tf1,tf2,l1,l2;

JButton b1,b2;

JPanel p1;

public Example2() {

setTitle("MAZE CReaTOR");

setSize(500,450);

setVisible(true);

setResizable(false);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

void setComponent(){

p1=new JPanel();

l1=new JTextField("Enter No. of rows:");

tf1=new JTextField();

l2=new JTextField("Enter No. of columns:");

tf2=new JTextField();

l3=new JLabel("Click Button to make grid");

b2=new JButton("MANUAL");

b1=new JButton("MAKE");

ImageIcon img=new ImageIcon("C:\\Users\\sanpa\\Desktop\\mazesolver\\image5.jpg");

background=new JLabel("",img,JLabel.CENTER);

background.setBounds(0,0,490,440);

p1.setLayout(null);

p1.setBounds(0, 0, 500, 315);

l1.setBounds(90,40,150,20);

tf1.setBounds(280,40,50,20);

l2.setBounds(90,120,150,20);

tf2.setBounds(280,120,50,20);

l3.setBounds(125,160,170,15);

b1.setBounds(160,180,80,30);

b2.setBounds(150, 250, 100, 30);

p1.add(l1);

p1.add(l2);

p1.add(tf1);

p1.add(tf2);

p1.add(l3);

p1.add(b1);

p1.add(b2);

b1.addActionListener(new Action());

b2.addActionListener(new Action());

add(p1);

p1.add(background);

}

public static void main(String[] args) {

Example2 jf=new Example2();

jf.setComponent();

}

class Action implements ActionListener{

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

int nr,nc;

if(e.getSource()==b1) {

nr=Integer.parseInt(tf1.getText());

nc=Integer.parseInt(tf2.getText());

if(nr<=15 && nc<=25) {

Example3 ob=new Example3(nr,nc);

ob.setComponents();

}

else {

JOptionPane.showMessageDialog(new JFrame(),"Rows greater than 15 OR\n columns greater than 25 are not allowed ","Dialog",JOptionPane.ERROR\_MESSAGE);

}

}

if(e.getSource()==b2) {

new Manual();

}

}

}

}

class Manual extends JFrame{

JTextArea a;

public Manual() {

setVisible(true);

setSize(300,400);

setResizable(false);

setTitle("MANUAL");

setComponents();

}

public void setComponents() {

a=new JTextArea();

a.setBounds(5, 5, 195, 395);

a.setText("\n No. of Rows should less Than 16\n No. of columns should less than 26\n\n BUTTONS :\n\n ENTRY :- add entry gate\n\n Exit :-Add Exit Gate\n\n WALL :- Add Walls\n\n RE.PATH :- Remove Existing Path\n\n REMOVE :-Removing Unwanted Walls Doors\n\n SOLVE :-Make Path from Entry To Exit\n\n RANDOM :-Add Walls Randomly");

add(a);

a.setEditable(false);

}

}

# FRAME 2

/\*THIS FRAME IS ABOUT THE MAZE CREATION AND FINDING PATH AS PER THE USERS INPUT.IT ALSO CONSISTS OF RE PATH ,REMOVE,RANDOM FUCTIONS\*/

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.event.\*;

import java.util.\*;

import javax.swing.\*;

class Example3 extends JFrame implements MouseMotionListener,MouseListener{

public int space=1,x=-100,y=-100;

int flag=0;

byte flag2=1,flag3=1,flag4=0,flag5=1,flag6=1;

static int rows,cols;

JButton b1,b2,b3,b4,b5,b6,b7;

JLabel l1,l2,background;

Image image;

public int arr[][];

public Example3(int nr,int nc) {

rows=nr;

cols=nc;

setTitle("M A Z E");

setVisible(true);

setSize(1024,740);

setResizable(false);

arr=new int[rows][cols];

}

public void setComponents() {

addMouseMotionListener(this);

addMouseListener(this);

setLayout(null);

b1=new JButton("SOLVE");

b1.setBounds(800,650,80,25);

b1.addActionListener(new Graph());

b2=new JButton("RANDOM");

b2.setBounds(650,650,100,25);

b2.addActionListener(new Graph());

b3=new JButton("REMOVE");

b3.setBounds(520,650,100,25);

b3.addActionListener(new Graph());

b4=new JButton("ENTRY");

b4.setBounds(20,650,100,25);

b4.addActionListener(new Graph());

b5=new JButton("EXIT");

b5.setBounds(140,650,100,25);

b5.addActionListener(new Graph());

b6=new JButton("WALL");

b6.setBounds(270,650,100,25);

b6.addActionListener(new Graph());

b7=new JButton("RE.PATH");

b7.setBounds(390,650,100,25);

b7.addActionListener(new Graph());

}

class Graph implements ActionListener {

int v,precount=1;

int[] pre;

private LinkedList<Integer> adj[];

/\*Constructor of Graph Class\*/

public Graph() {

v=rows\*cols;

adj=new LinkedList[v];

pre=new int[v];

for(int i=0;i<v;i++) {

pre[i]=-1;

}

for(int i=0;i<v;i++) {

adj[i]=new LinkedList<>();

}

}

/\*function to add connection between two nodes\*/

void addEdge(int l,int w) {

try{adj[l].add(w);}catch(Exception e){}

}

/\*Bfs search function \*/

void Bfs(int s,int key) {

boolean visited[]=new boolean[v];

LinkedList<Integer> queue=new LinkedList<Integer>();

visited[s]=true;

queue.add(s);

while(queue.size()!=0) {

s=queue.poll();

Iterator<Integer> i=adj[s].listIterator();

while(i.hasNext()) {

int n=i.next();

if(!visited[n]) {

visited[n]=true;

queue.add(n);

pre[n]=s;

if(n==key) {

flag++;

return;

}

}

}

}

if(flag==0) {

JOptionPane.showMessageDialog(new JFrame(),"PATH NOT FOUND","Dialog",JOptionPane.ERROR\_MESSAGE);

}

}

/\*backtracking the possible path\*/

void backtrack(int start,int end){

int cont;

for(int i=end;i!=-1;i=pre[i]) {

cont=(rows-1)\*(cols-1);

for(int k1=rows-1;k1>=0;k1--) {

for(int k2=cols-1;k2>=0;k2--) {

if(k1+k2+cont==start) {

arr[k1][k2]=3;

}

if(k1+k2+cont==end) {

arr[k1][k2]=4;

}

else if(i==(k1+k2+cont)) {

arr[k1][k2]=2;

}

}

cont=cont-(cols-1);

}

}

cont=0;

for(int k1=0;k1<rows;k1++) {

for(int k2=0;k2<cols;k2++) {

if(k1+k2+cont==start) {

arr[k1][k2]=3;

}

if(k1+k2+cont==end) {

arr[k1][k2]=4;

}

}

cont=cont+(cols-1);

}

repaint();

}

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

if(e.getSource()==b1) {

int i,j,constant=0,key=-1,strt=-1,flag=0;

Graph graph=new Graph();

for(int m=0;m<rows;m++) {

for(int n=0;n<cols;n++) {

if(arr[m][n]==3) {

strt=m+n+flag;

arr[m][n]=0;

}

if(arr[m][n]==4){

key=m+n+flag;

arr[m][n]=0;

}

}

flag=flag+(cols-1);

}

/\*connecting blocks which don't have walls\*/

for(i=0;i<rows;i++) {

for(j=0;j<cols;j++){

if(arr[i][j]==0) {

if(j!=cols-1)

if(arr[i][j+1]==0)

graph.addEdge(constant,constant+1);

if(j!=0)

if(arr[i][j-1]==0){

graph.addEdge(constant,constant-1);

}

if(i!=0)

if(arr[i-1][j]==0) {

graph.addEdge(constant,constant-cols);

}

if(i!=rows-1)

if(arr[i+1][j]==0) {

graph.addEdge(constant,constant+cols);

}

}

constant++;

}

}

if(key!=-1) {

graph.Bfs(strt,key);

graph.backtrack(strt,key);

}

else {

JOptionPane.showMessageDialog(new JFrame(),"PATH NOT FOUND","Dialog",JOptionPane.ERROR\_MESSAGE);

}

}

else if(e.getSource()==b2){

Random r=new Random();

for(int i=0;i<rows;i++) {

for(int j=0;j<cols;j++) {

if(arr[i][j]!=3&&arr[i][j]!=4) {

if(r.nextInt(rows\*cols)<(rows\*cols)/4) {

arr[i][j]=1;

}

}

}

}

repaint();

}

else if(e.getSource()==b3){

flag5=0;

flag2=flag3=flag6=1;

}

else if(e.getSource()==b4){

flag2=0;

flag5=flag3=flag6=1;

}

else if(e.getSource()==b5) {

flag3=0;

flag2=flag5=flag6=1;

}

else if(e.getSource()==b6){

flag6=0;

flag2=flag3=flag5=1;

}

else if(e.getSource()==b7) {

for(int i=0;i<rows;i++) {

for(int j=0;j<cols;j++) {

if(arr[i][j]==2) {

arr[i][j]=0;

}

}

}

flag=0;

repaint();

}

}

}

public void paint(Graphics g) {

if(flag4==0) {

g.setColor(Color.DARK\_GRAY);

g.fillRect(0, 0, 1024,740);

add(b1);

add(b2);

add(b3);

add(b4);

add(b5);

add(b6);

add(b7);

flag4++;

}

for(int j=0;j<rows;j++) {

for(int i=0;i<cols;i++) {

if(arr[j][i]==2) {

ImageIcon img=new ImageIcon("C:\\Users\\sanpa\\Desktop\\mazesolver\\road1.jpg");

image=img.getImage();

g.drawImage(image,space+i\*40+10,space+j\*40+40,null);

}

else if(arr[j][i]==1) {

ImageIcon img=new ImageIcon("C:\\Users\\sanpa\\Desktop\\mazesolver\\wall.jpg");

image=img.getImage();

g.drawImage(image,space+i\*40+10,space+j\*40+40,null);

}

else if(arr[j][i]==3) {

ImageIcon img=new ImageIcon("C:\\Users\\sanpa\\Desktop\\mazesolver\\door.jpg");

image=img.getImage();

g.drawImage(image,space+i\*40+10,space+j\*40+40,null);

}

else if(arr[j][i]==4) {

ImageIcon img=new ImageIcon("C:\\Users\\sanpa\\Desktop\\mazesolver\\door.jpg");

image=img.getImage();

g.drawImage(image,space+i\*40+10,space+j\*40+40,null);

}

else

{

ImageIcon img=new ImageIcon("C:\\Users\\sanpa\\Desktop\\mazesolver\\grass.jpg");

image=img.getImage();

g.drawImage(image,space+i\*40+10,space+j\*40+40,null);

}

}

}

}

@Override

public void mouseMoved(MouseEvent e) {

// TODO Auto-generated method stub

x=e.getX();

y=e.getY();

}

@Override

public void mouseClicked(MouseEvent e) {

// TODO Auto-generated method stub

if(flag6==0)

{if(inboxX()!=-1&&inboxY()!=-1)

{arr[inboxX()][inboxY()]=1;

repaint();

}

}

if(flag3==0) {

if(inboxX()!=-1&&inboxY()!=-1)

{arr[inboxX()][inboxY()]=4;

flag3++;

repaint();

}

}

if(flag2==0)

{

if(inboxX()!=-1&&inboxY()!=-1)

{arr[inboxX()][inboxY()]=3;

flag2++;

repaint();

}

}

if(flag5==0) {

if(inboxX()!=-1&&inboxY()!=-1) {

arr[inboxX()][inboxY()]=0;

repaint();

}

}

}

/\*returns row number for click\*/

public int inboxX() {

// TODO Auto-generated method stub

for(int j=0;j<rows;j++) {

for(int i=0;i<cols;i++) {

if(x>=space+i\*40+10 && x<space+i\*40+40-2\*space+10 && y>=space+j\*40+40 && y<space+j\*40+40+40-2\*space) {

return j;

}

}

}

return -1;

}

/\*returns column number for click\*/

public int inboxY() {

// TODO Auto-generated method stub

for(int j=0;j<rows;j++) {

for(int i=0;i<cols;i++) {

if(x>=space+i\*40+10 && x<space+i\*40+40-2\*space+10 && y>=space+j\*40+40 && y<space+j\*40+40+40-2\*space) {

return i;

}

}

}

return -1;

}

@Override

public void mousePressed(MouseEvent e) {

// TODO Auto-generated method stub

}

@Override

public void mouseReleased(MouseEvent e) {

// TODO Auto-generated method stub

}

@Override

public void mouseEntered(MouseEvent e) {

// TODO Auto-generated method stub

}

@Override

public void mouseExited(MouseEvent e) {

// TODO Auto-generated method stub

}

@Override

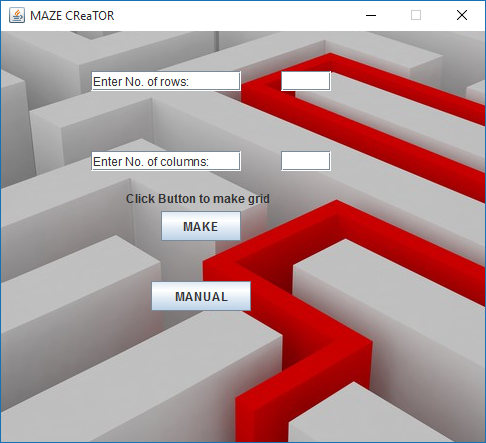
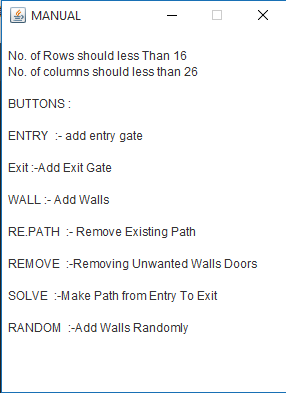
public void mouseDragged(MouseEvent e) {

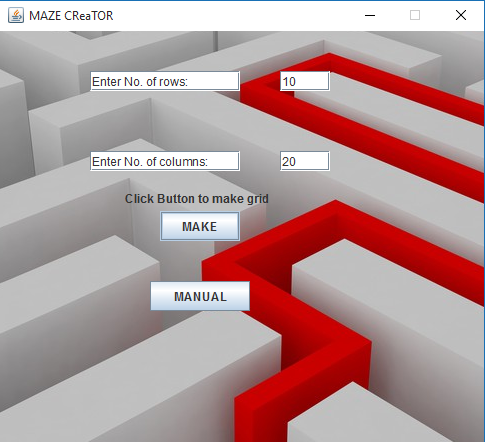
// TODO Auto-generated method stub

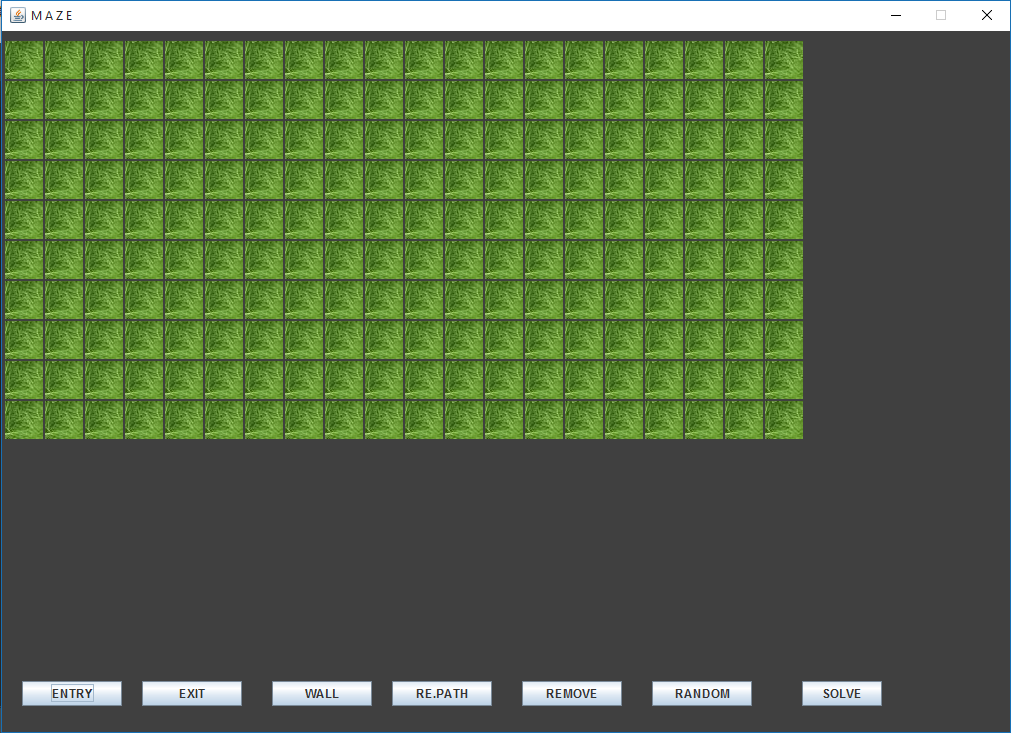
}

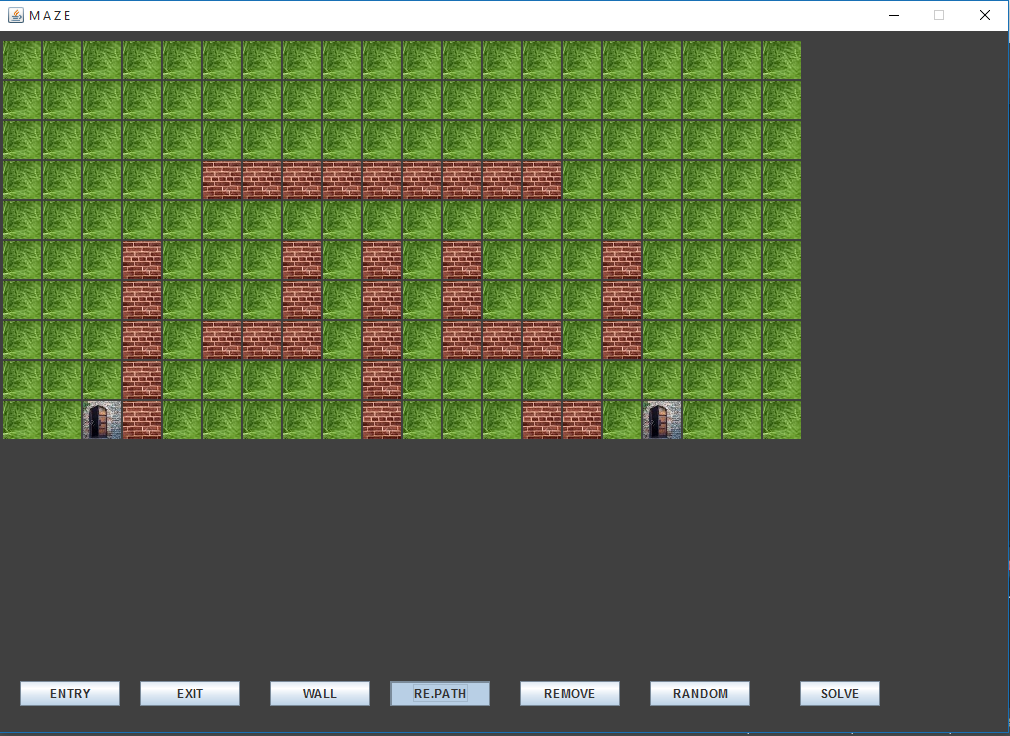
}

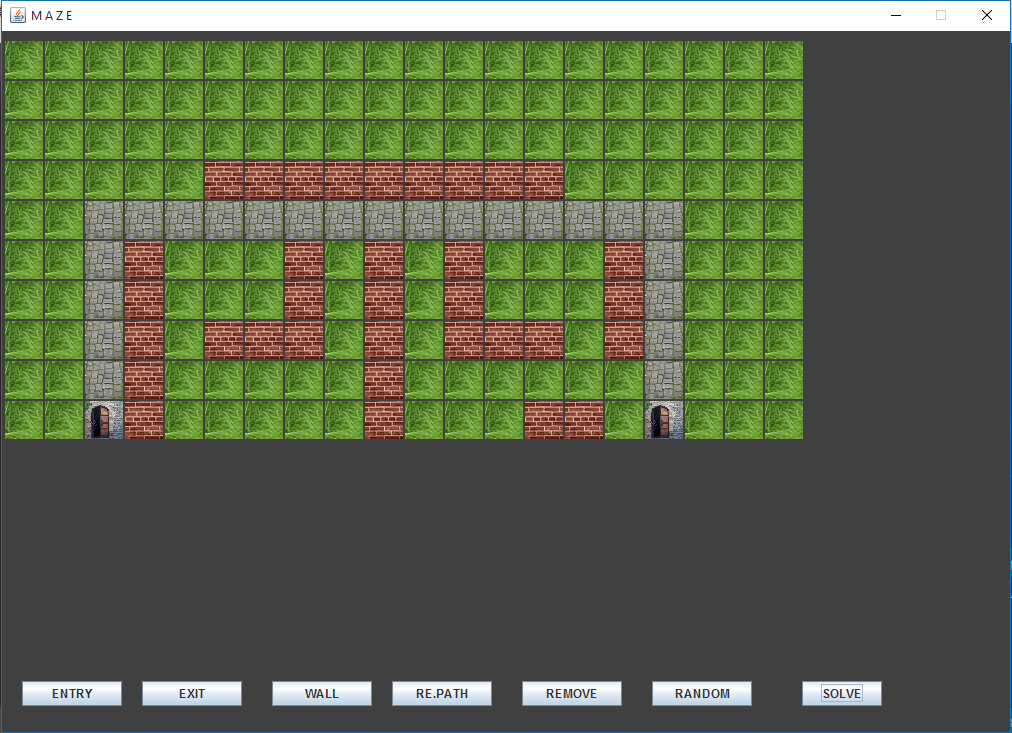
# OUTPUT

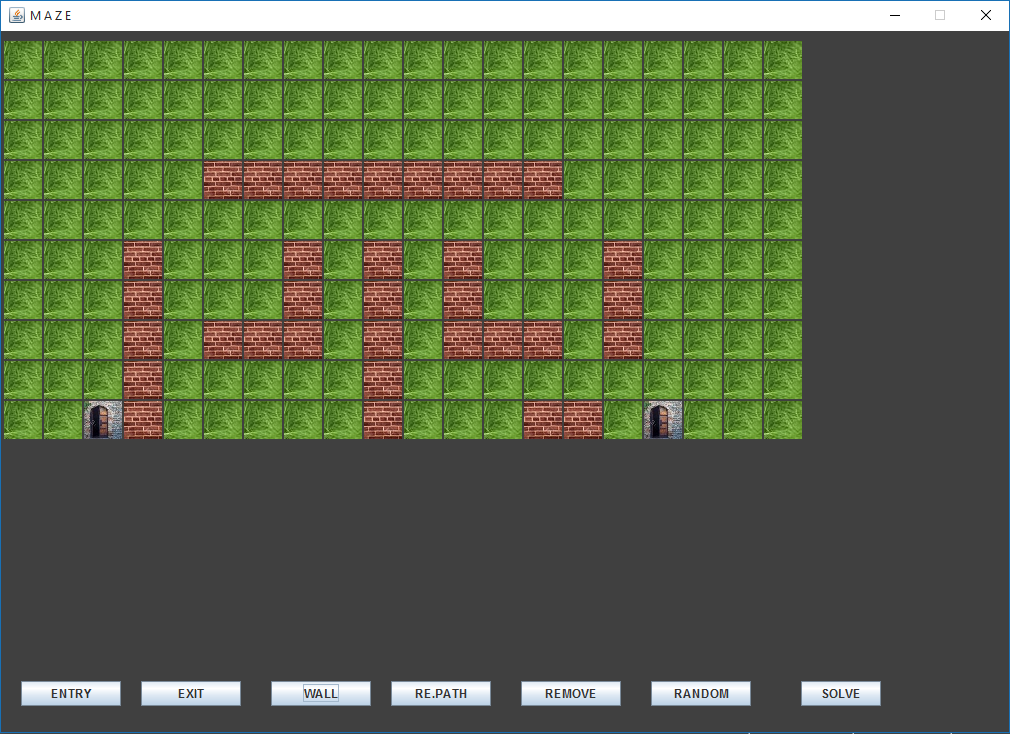


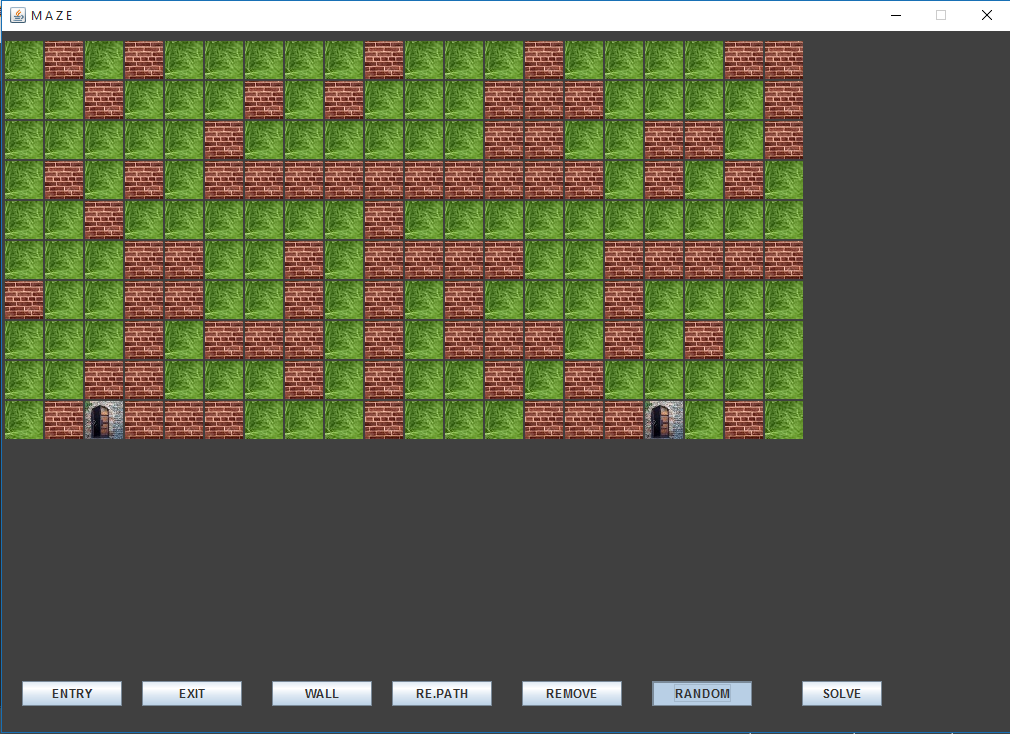


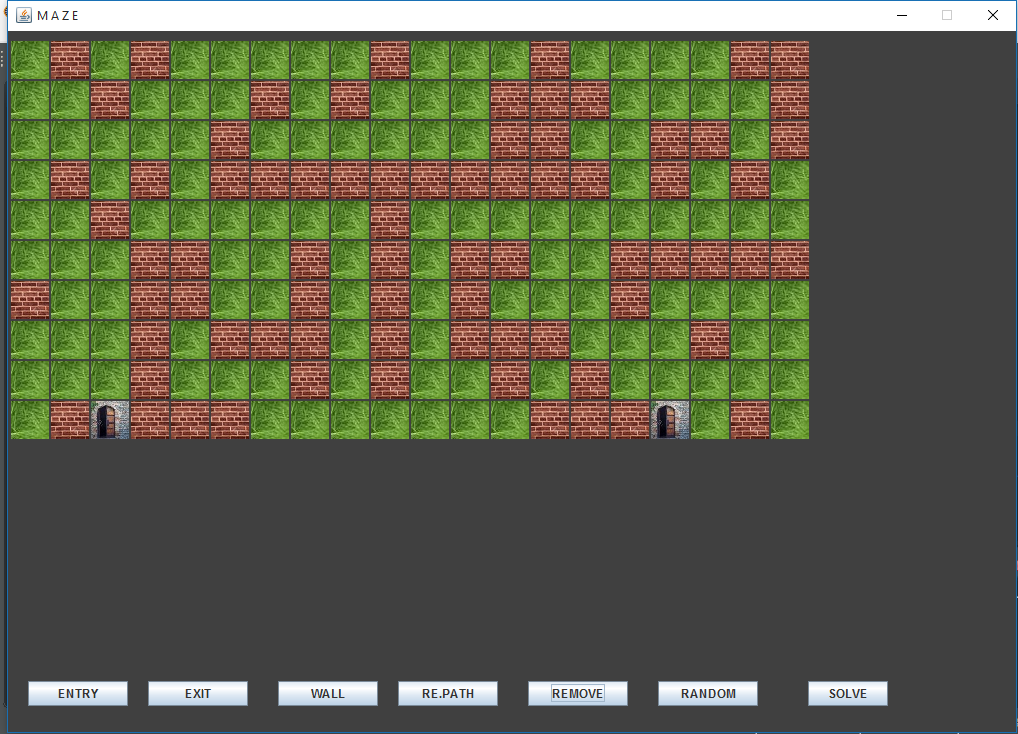


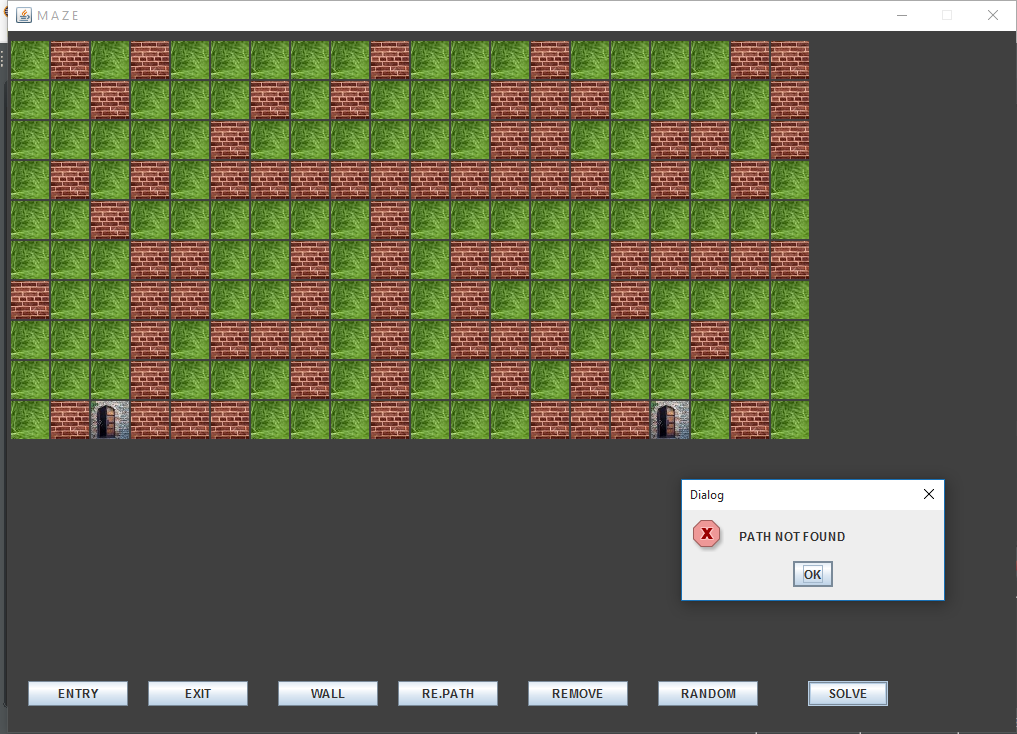


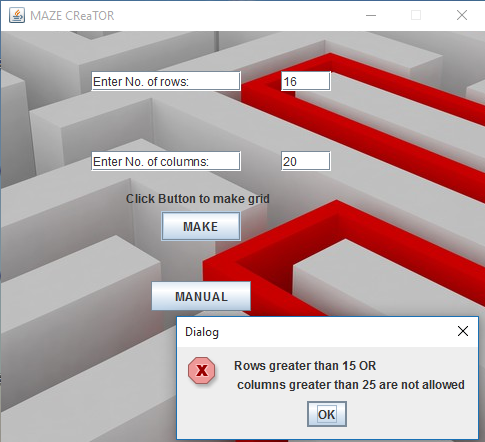












# REFERENCES

* javaTpoint <https://www.javatpoint.com/java-swing>
* Geeks for Geek <https://www.geeksforgeeks.org/creating-frames-using-swings-java/>
* Stack Overflow [https://stackoverflow.com](https://stackoverflow.com/)
* Tutorials point <https://www.tutorialspoint.com/awt/awt_graphics_class.htm>l