ALGORITHMS

Author: Abel Gallart Bonome

MINESWEEPER

En JavaFX Swing

Minesweeper is a single-player video game created by Curt Jhonson and Robert Donner in 1989. The objective of the game is to clear a minefield without detonating any. The game consists **of discovering all the locations of a gridded board in which there are mines**. The game begins with the entire board covered and to start playing we will have to click on any square, which will show us its content, which can be: a mine, in which case we will lose the game; a number, indicating the number of mines in the 8 adjacent boxes; or it may appear as a blank box, indicating that it has no mine in the boxes around it. When we click on a mineless box sometimes the surrounding boxes that do not have mines in their respective adjacent boxes will open until the open space is surrounded either by the limits of the map or by squares with mines in their surroundings, although this does not mean that there are boxes beyond those that have been shown without mines since only the boxes with adjacent mines will be shown. that have an unnumbered box in their nearby boxes. **Knowing the number of mines in the adjacent boxes of several positions at the**

same time you can deduce the location of the mines, which we must mark until we have indicated the number of mines initially hidden, once this is done if we have hit the boxes marked **we will have won the game**.

The game can be configured to play in **different difficulties** already pre-designed or you can manually customize the size of the board and the number of mines that we want to hide in it, thus adjusting the difficulty to the taste of each player. Regardless of the settings, **each game will be unique** as the mines are put on the board randomly each time.

What is the Minesweeper for?

The minesweeper was **one of the first games that were implemented in modern personal computers**, along with other games such as Solitaire or Hearts, all these **were not created seeking to entertain users**, on the contrary, they sought **to get people used to using the mouse**, making these games you need to **click on small squares with precision** as in the case of minesweeper or minesweeper or **press and drag** the cards as in the case of solitaire. These games over the years have continued to be part of Microsoft's operating systems natively, undergoing updates and changes in the interface until **Windows 8, which is the last version of Windows that included them**.

The classic minesweeper **uses a grid** of different dimensions, but over the years other versions have appeared that take the concept further by introducing **triangular boards or with other geometric shapes** in addition to new mechanics such as the possibility of playing **in three dimensions or that several mines may appear in the same square**.

British neurologists conducted a study between those who play and those who do not play. The results surprised the public. It turned out that people who play minesweeper learn motor skills much faster and their **brain absorbs** new information faster. Bonus: Improves players' perception of the small components at all.

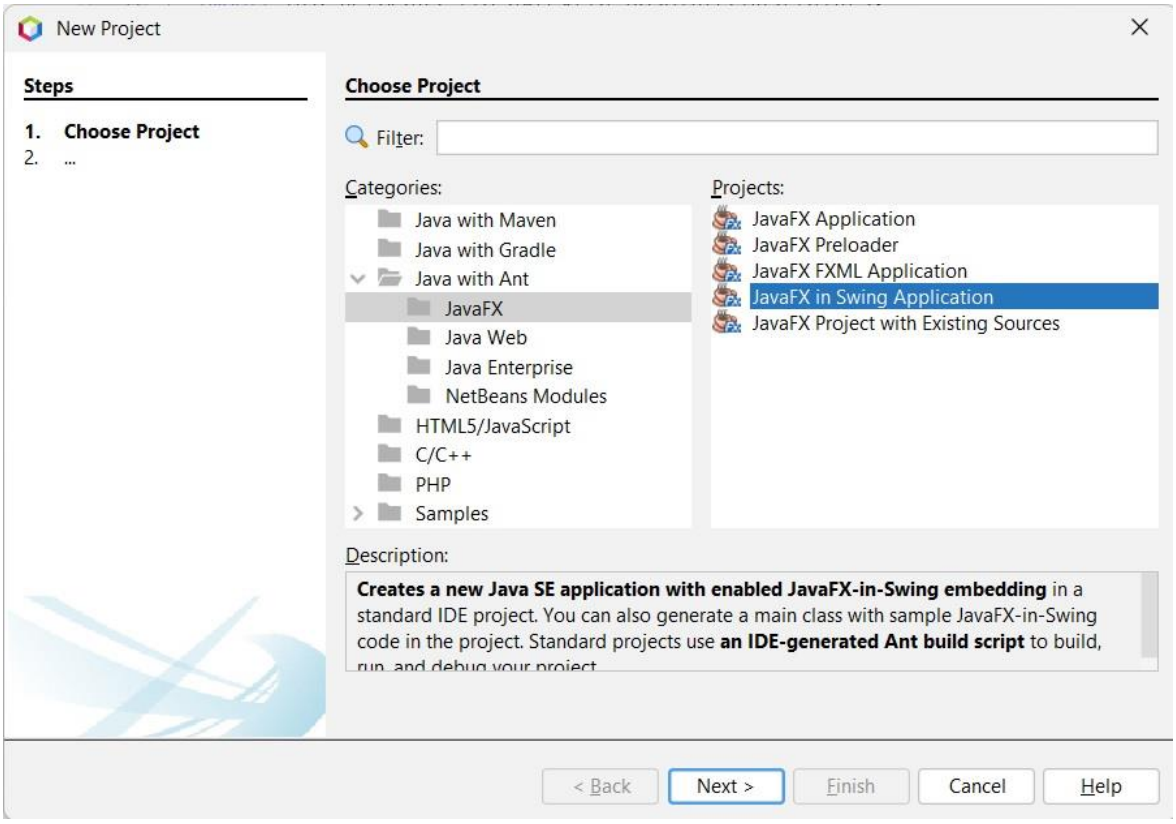
The online minesweeper series of games deserves attention. Despite the **militant name**, the online game Minesweeper is **quite peaceful**. The game is included in the set of free toys for all windows. A simple plot does not require a deep study of the rules! But the game requires attention and ingenuity on the part of the players. A false move will result in an explosion. A misstep towards explosion and loss. We'll have to start over. The minesweeper game is so attractive that it can easily captivate you for several hours.

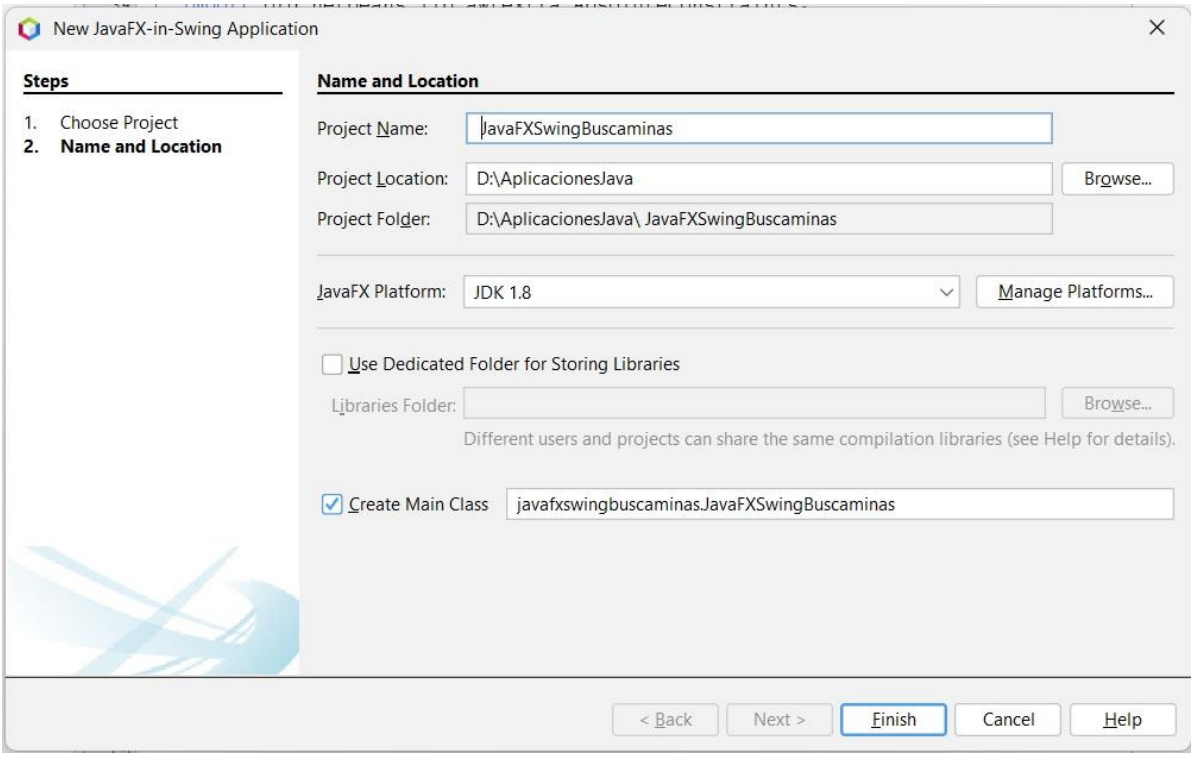
Keep in mind that you can lose from the first move. If you accidentally land in a mine behind hidden squares, you'll have to start over. Don't bother, but the hand itself directs the computer mouse to restart the process.

Minesweeper is suitable for **children and adults**. There is no doubt that this minigame has benefits, which are the following:

- Improves the brain;
- Develops thinking;
- Develops the ability to respond effectively;
- Develop motor skills in a child aged 3 to 9 years.
- It helps the brain stay in shape.

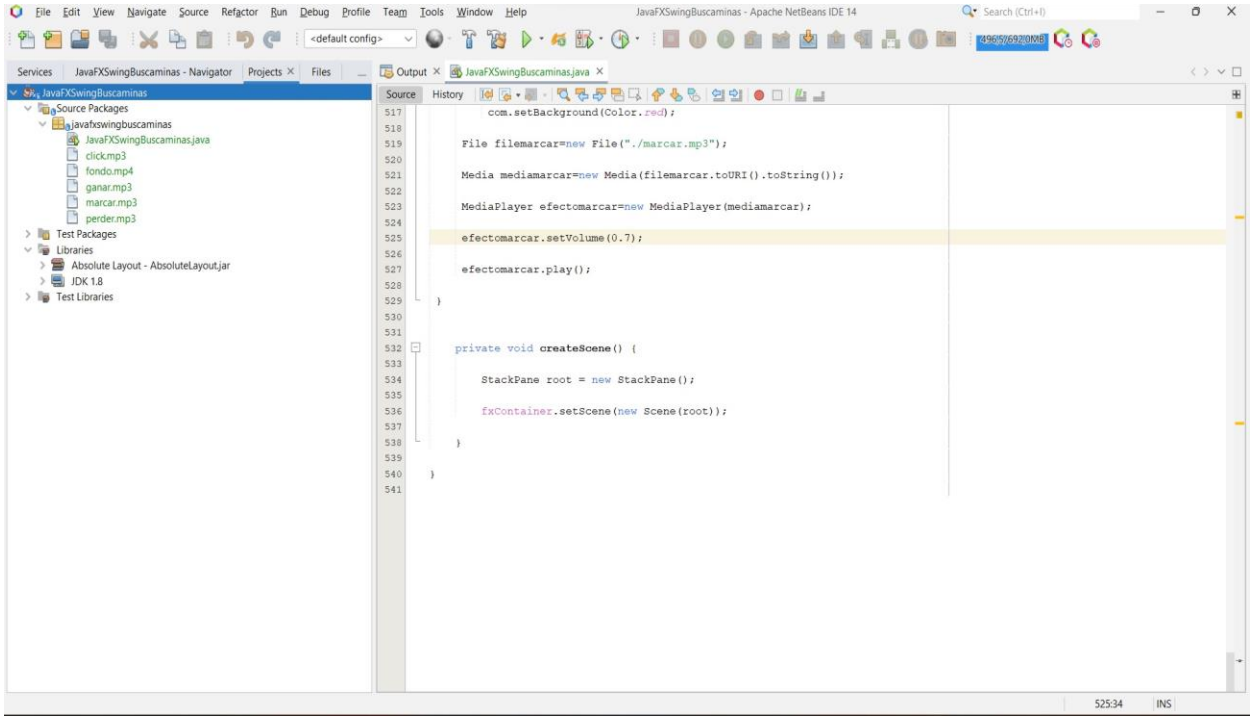
To get started we opened NetBeans and created our JavaFX Swing project.





And we add to our package the multimedia files that we will use as sound effects, specifically the sound files win.mp3, lose.mp3, click.mp3, mark.mp3, and background.mp4 a video that plays in the background of our game, to make the game more visual.

Source code:



And now if we go on to code our algorithm, we have to express in Java language the dynamics of the game.

/*

* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

* Click nbfs://nbhost/SystemFileSystem/Templates/javafx/FXSwingMain.java to edit this template

*/

package javafxswingbuscaminas;

We import the classes we will need

import java.awt.Color;

import java.awt.Component;

import java.awt.Dimension;

import java.awt.Font;

```
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.net.URISyntaxException;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Platform;
import javafx.embed.swing.JFXPanel;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javax.swing.BorderFactory;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import org.netbeans.lib.awtextra.AbsoluteConstraints;
import org.netbeans.lib.awtextra.AbsoluteLayout;

/**
*
* @author Abel Gallart Bonome
```

```
*/
public class JavaFXSwingBuscaminas extends JApplet {

  We declare the variables of the class

   int JFXPANEL_WIDTH_INT = 1000;

   int JFXPANEL_HEIGHT_INT = 1000;

   JFXPanel fxContainer;

   static int ancho=30;

   static int largo=30;

   static int pumps=30;

   static int points=0;

   static int level=0;

   Random ran;

   String [][]matrix;

   JButton b[][];

   JLabel l[][];

   static JFrame frame;
```

```
    static JApplet applet;

    private JLabel jLabel1;

    private JLabel jLabel2;

    private JLabel jLabel3;

    private JPanel jPanel1;

    private JPanel jPanel2;

    private JPanel jPanel3;
```

In our main method we create the window and display an instance of our class.

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

        SwingUtilities.invokeLater(() -> {

            try {

UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
            }

            catch (Exception e){}

            frame = new JFrame("Buscaminas");
```

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setUndecorated(true);

        frame.setOpacity((float) 1.0);

        applet = new JavaFXSwingBuscaminas();

        applet.init();

        frame.setContentPane(applet.getContentPane());

        frame.pack();

        frame.setLocationRelativeTo(null);

        frame.setVisible(true);

        applet.start();
    });
}
```

In the init() method, the variables and objects necessary for both graph and logic are initialized.

```
@Override
public void init() {

    jPanel2 = new JPanel();
```

```
jLabel1 = new JLabel();

jLabel2 = new JLabel();

jLabel3 = new JLabel();

jPanel3 = new JPanel();

jPanel2.setSize(new Dimension(720, 720));

jPanel2.setOpaque(false);

jPanel2.setLayout(new AbsoluteLayout());

jLabel1.setFont(new Font("Segoe UI", 3, 36)); // NOI18N

jLabel1.setForeground(new Color(255, 255, 255));

jLabel1.setText("Pumps"+bombs);

jLabel2.setFont(new Font("Segoe UI", 3, 36)); // NOI18N

jLabel2.setForeground(new Color(255, 255, 255));

jLabel2.setText("Level"+level);

jLabel3.setFont(new Font("Segoe UI", 3, 36)); // NOI18N

jLabel3.setForeground(new Color(255, 255, 255));
```

```
jLabel3.setText("Points"+points);

jPanel3.setBackground(new Color(255, 153, 0));

jPanel3.setBorder(BorderFactory.createTitledBorder(null, "Buscaminas",
javax.swing.border.TitledBorder.CENTER, javax.swing.border.TitledBorder.ABOVE_TOP, new
java.awt.Font("Segoe UI", 0, 12), new java.awt.Color(255, 255, 255))); // NOI18N

ran=new Random();

matris=new String[ancho][largo];

for(int x=0;x<ancho;x++)

   for(int y=0;y<largo;y++)

   {matris[x][y]="0";}

int c=0;
```

A Random ram object is created and a wide*long array is initialized with the text "0" and we take a c counter that we initialize to zero as well.

We repeat the following operations that are inside the "while" while the counter c is less than the number of pumps.

```
while(c<bombas){

   int x=(int)(ancho*ran.nextFloat());

   int y=(int)(long*ran.nextFloat());
```

We take a position x, y within the matrix[x][y] randomly y ....

```
        if (!matris[x][y].equals("B"))
        {  if in that position there is no pump "B" we place it matris[x][y]="B" and increase the pump counter by 1
            uterus[x][y]="B";
            c++;
        }

    }
```

So now we have a long*wide array that has a number of randomly arranged bombs within the array.

```
    for(int x=0;x<ancho;x++)

        for(int y=0;y<largo;y++)

        {

            c=0;

            try{if (matris[x-1][y].equals("B"))c++;} catch(Exception e){}

            try{if (matris[x-1][y-1].equals("B"))c++;} catch(Exception e){}

            try{ if (matris[x][y-1].equals("B"))c++;} catch(Exception e){}

            try{if (matris[x+1][y-1].equals("B"))c++;} catch(Exception e){}

            try{if (matris[x+1][y].equals("B"))c++;} catch(Exception e){}
```

```
try{if (matris[x+1][y+1].equals("B"))c++;} catch(Exception e){}

try{if (matris[x][y+1].equals("B"))c++;} catch(Exception e){}

try{if (matris[x-1][y+1].equals("B"))c++;} catch(Exception e){}
```

Time we go through the matrix and in each box we see if the 8 neighboring squares have pumps and we count, then we will have in a counter c the pumps that are in the neighboring boxes, this process is done for each of the boxes, the try & catch we use it to avoid overflows when we place ourselves in the squares of the edges of the matrix, in which case an "index out of..." exception is thrown that is captured and the program continues in the next box.

```
if (!matris[x][y].equals("B")) matris[x][y]=c+"";
```

And now in all the squares of the matrix if there is no pump the number of pumps that are in the neighboring squares is placed.

```
}
```

So we already have a string matrix of length * width with random pumps and in the rest of the boxes the number of pumps that are in the neighbors

```
b=new JButton[width][length];

l=new JLabel[width][length];
```

The next step is to create that matrix but with graphic elements, because we will create an array of JLabel [length][width] tags and another of JButton[length][width] buttons.

```
for(int x=0;x<ancho;x++)

  for(int y=0;y<largo;y++)
    {

      l[x][y]=new JLabel();
```

```
l[x][y].setForeground(Color.WHITE);

l[x][y].setHorizontalAlignment(JLabel.CENTER);

if (!matris[x][y].equals("0")) l[x][y].setText(" "+matris[x][y]);
```

We go through our matrix again and in the matrix of tags we write the contents of our matrix[lx][y] of String if it is not "0" .

```
b[x][y]=new JButton();

b[x][y].setBackground(Color.WHITE);
```

We place in our matrix of buttons a button in each position and declare a Mouse Event we specify that if a click event is made on the button we perform the following procedure.

```
b[x][y].addMouseListener(new MouseAdapter() {

  @Override
  public void mouseClicked(MouseEvent evt) {

    try {

      if(evt.getButton()==MouseEvent.BUTTON3) Button3(evt);
```

If the click event was done with the right mouse button perform Button3(evt) which describes the procedure to follow to check the box as a possible bomb

```
      if(evt.getButton()==MouseEvent.BUTTON1) Button1(evt);
```

If the click event has been done with the left mouse button perform Button1(evt) which describes the procedure to follow to find out if there is a pump. If so, we have lost but, we will discover all the empty squares to the edge of the mined areas.

```
    }
```

```
                catch (URISyntaxException ex) {

Logger.getLogger(JavaFXSwingBuscaminas.class.getName()).log(Level.SEVERE, null, ex);

                }

        }


            });

        jPanel2.add(b[x][y], new AbsoluteConstraints(x*length,y*width,29,29));

        jPanel2.add(l[x][y], new AbsoluteConstraints(x*length,y*width,29,29));
```

And finally we add to the panel "JPanel2" in the x position, and of the grid a label that where the number of neighboring pumps was written in each one and on top of it a button that temporarily hides the contents of the label until an event occurs shades the button

```
}
```

So far we have the area of the grid that represents the minefield

```
    jPanel1 = new javax.swing.JPanel();

    jPanel1.setSize(new
java.awt.Dimension(JFXPANEL_WIDTH_INT,JFXPANEL_WIDTH_INT));

    jPanel1.setLayout(new AbsoluteLayout());
```

```
        jPanel1.add(jLabel1, new AbsoluteConstraints(110, 10,270, 60));

        jPanel1.add(jLabel2, new AbsoluteConstraints(330, 10, 190, 60));

        jPanel1.add(jLabel3, new AbsoluteConstraints(540, 10, 240, 60));

        jPanel1.add(jPanel3, new AbsoluteConstraints(100, 10, 670, 60));

        jPanel1.add(jPanel2, new AbsoluteConstraints(50, 80, -1, -1));

        setLayout(new AbsoluteLayout());

        jPanel1.setOpaque(false);

        add(jPanel1,new AbsoluteConstraints(0,0, -1,-1));

        fxContainer = new JFXPanel();

        fxContainer.setPreferredSize(new
Dimension(JFXPANEL_WIDTH_INT,JFXPANEL_HEIGHT_INT));

        add(fxContainer,new AbsoluteConstraints(0,0,-1,-1));



        Platform.runLater(new Runnable() {

            @Override
            public void run() {
                try {
                    createScene();
```

```
            File filefondo=new File("./fondo.mp4");

            Media mediafondo=new Media(filefondo.toURI().toString());

            MediaPlayer fondo=new MediaPlayer(mediafondo);

            MediaView mv=new MediaView(fondo);

            fxContainer.setScene(new Scene(new Group(mv)));

            background.setCycleCount(MediaPlayer.INDEFINITE);

            background.setVolume(0.7);

            mv.setFitHeight(2000);

            mv.setFitWidth(2000);

            background.play();

        } catch (Exception ex) {}

    }

});
```

We finished configuring the rest of the graphical features of the interface

```
}
```

```
public void analysis(JButton bot){

    if (!bot.isVisible()) return ;

    int x=bot.getX()/30;

    int y=bot.getY()/30;

    bot.setVisible(false);

    if (matris[x][y].equals("0"))
    {

        points++;

        jLabel3.setText("Points"+points);

        try {analiza(b[x-1][y]);} catch(Exception e){}

        try {analiza(b[x-1][y-1]);} catch(Exception e){}

        try {analiza(b[x][y-1]);} catch(Exception e){}

        try {analiza(b[x+1][y-1]);} catch(Exception e){}

        try {analiza(b[x+1][y]);} catch(Exception e){}

        try {analiza(b[x+1][y+1]);} catch(Exception e){}

        try {analiza(b[x][y+1]);} catch(Exception e){}
```

```
        try { analiza(b[x-1][y+1]);} catch(Exception e){}

    }

}
```

This method is called when we left click on the button, here we discover the box, if there is a mine we lose, but the rest of the mineless squares are discovered in cascade to the edges of the mined areas

```
private void Button1(MouseEvent   evt) throws URISyntaxException {

    Component com=((Component) evt.getSource());
```

We get the button on which the click was given

```
    int x=com.getX()/30;

    int y=com.getY()/30;
```

We obtain the position x, y in the matrix from the coordinates of the button (x, y) in the graph panel, as the boxes have a size of 30*30, by dividing the coordinates by 30 we obtain the position in the array of String objects.

```
    if (matris[x][y].equals("B"))
    {
```

If in the string matrix we fear a pump in the obtained position x,y, then we have lost.

```
        File fileperder=new File("./perder.mp3");

        Media mediaperder=new Media(fileperder.toURI().toString());

        MediaPlayer effect=new MediaPlayer(mediaperder);

        effect.setVolume(0.7);

        effect.play();
```

Here an audio effect is played that indicates that we have lost

```
javax.swing.JOptionPane.showMessageDialog
    (null,"You've Lost","End game....",javax.swing.JOptionPane.OK_OPTION);
```

We launch a dialog window with a message that indicates to the user that he has lost we stop the applet and hide the window.

```
applet.stop();


frame.setVisible(false);
```

We refer our window to a new Window and leave without reference the previous one that will already be taken care of by the garbage collector of the system.

```
frame = new JFrame("Buscaminas");


frame.setUndecorated(true);


frame.setOpacity((float) 1.0);


frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


applet = new JavaFXSwingBuscaminas();


points=0;


level=0;


pumps=40;


applet.init();


frame.setContentPane(applet.getContentPane());


frame.pack();
```

```
            frame.setLocationRelativeTo(null);

            frame.setVisible(true);

            applet.start();
            Here we finish launching the new window in case we have lost
           }

      else{

        In the event that the box at the position (x , y) of the matrix is not a bomb We call the
method analyzes(b[x][y]) and pass it as an argument, the button of the matrix of buttons in
the position (x, y) as a result all the boxes without pumps will be discovered cascading to the
edge of the mined areas.
        analyzes(b[x][y]);

         points++;
         And we add 1 point and write it on the label
         jLabel3.setText("Points"+points);

         File fileclick=new File("./click.mp3");

         Media mediaclick=new Media(fileclick.toURI().toString());

         MediaPlayer efectoclick=new MediaPlayer(mediaclick);

         efectoclick.setVolume(0.7);

         efectoclick.play();
```

And then we play an audio that indicates that we have given a left mouse click. As you can see, these audio files are located inside the project folder.

```
int c=0;

for(int i=0;i<ancho;i++)

   for(int j=0;j<largo;j++)

      if ( b[i][j].isVisible())  c++;
```

Here we go through the matrix of buttons and count those that are visible, if they coincide with the number of bombs this means that we have discovered all the empty squares and we have won and we go to the next level.

```
if (c==pumps)

{
   File fileganar=new File("./ganar.mp3");

   Media mediaganar=new Media(fileganar.toURI().toString());

   MediaPlayer efectoganar=new MediaPlayer(mediaganar);

   efectoganar.setVolume(0.7);

   efectoganar.play();
```

Here it plays the audio that indicates that we have won.

```
   javax.swing.JOptionPane.showMessageDialog
        (null,"You Won","End of Game....",javax.swing.JOptionPane.OK_OPTION);
```

A dialog window is launched with a message to the user indicating that he has won then the applet is stopped and the window is hidden.

```
   applet.stop();
```

```
frame.setVisible(false);

frame = new JFrame("Buscaminas");
```

Reference the window to a new one leaving the old one for the garbage collector.

```
frame.setUndecorated(true);

frame.setOpacity((float) 1.0);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

applet = new JavaFXSwingBuscaminas();

pumps=(pumps+1>width*long/2)?pumps:pumps+1;

level++;

points+=200;
```

We launch a new window with 1 more pump than the previous one and update the variable pump to the limit of length *width/2, that is, half of the squares of the matrix, we go up one level and 200 points.

```
applet.init();

frame.setContentPane(applet.getContentPane());

frame.pack();

frame.setLocationRelativeTo(null);

frame.setVisible(true);

applet.start();
```

```
        }

      }
    }
```

This method describes the actions to take when a left-click event is triggered. These consist of marking and unchecking alternately in red and white.

```
private void Button3(MouseEvent  evt) {

   Component com=((Component) evt.getSource());
```

We get the button on which the left click was made.

```
   if (com.getBackground().equals(Color.red))
```

{ If it is red we will put it in white and return to the position of the program from where the method was called.

```
      com.setBackground(Color.WHITE);

 return;
      }

   if (com.getBackground().equals(Color.WHITE))
```

If it is white we will put it red and play an audio that indicates that we have marked a bomb.

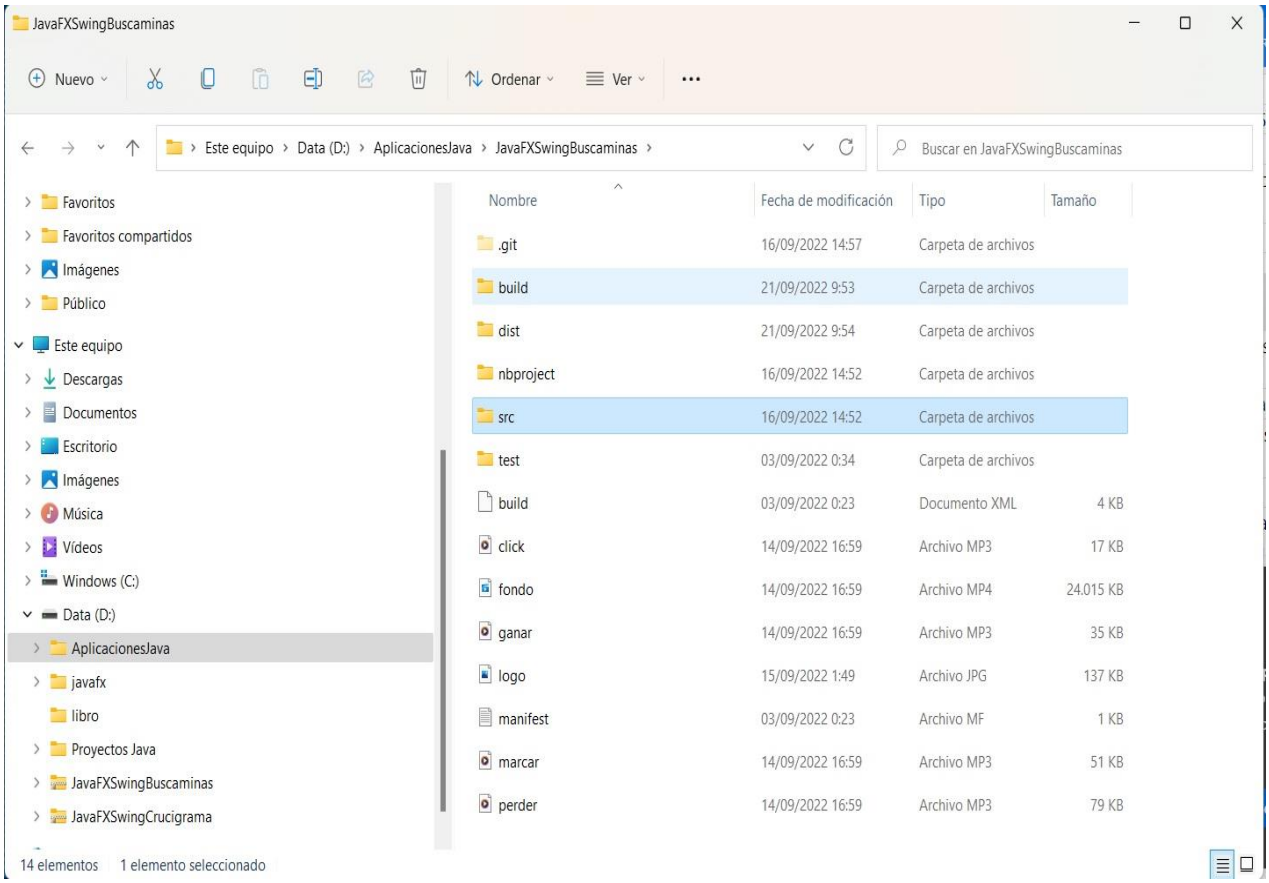
```
      com.setBackground(Color.red);

   File filemarcar=new File("./marcar.mp3");

   Media mediamarcar=new Media(filemarcar.toURI().toString());

   MediaPlayer efectomarcar=new MediaPlayer(mediamarcar);
```

```
    efectomarcar.setVolume(0.7);

    efectomarcar.play();

}
```

this method is called when you create the window to initialize a number of graphical features

```
   private void createScene() {

      StackPane root = new StackPane();

      fxContainer.setScene(new Scene(root));

   }

}
```

And here ends the code of the JavaFXSwingBuscaminas class.



Contents of the project folder.