HONORS PROJECT

Bee Sweeper

Student: Mehmet Sahin
Mentor: Yan Chen
Course: CSC 210
Section: Honors
Semester: Fall 2017
Date: 12/07/17

1. Introduction

Bee Sweeper is a puzzle game that has a grid full of rectangles. Each of these rectangles hides some information for a player to figure out the next steps. Each rectangle has a width and height of 30 pixels. The way to calculate rows and columns of this grid depends on the width of the Panel, Frame or Window. For this game, 600 X 600 has been chosen. So, 600/30 gets 20 rows and columns. Each of these rows and columns' location (a rectangle) contains a class called Cell. The Cell class hold its own location, width, height, bee, reveal, number of bees of the neighbors and flag. The way rectangles hold a class of Cell is shown in Figure 1.

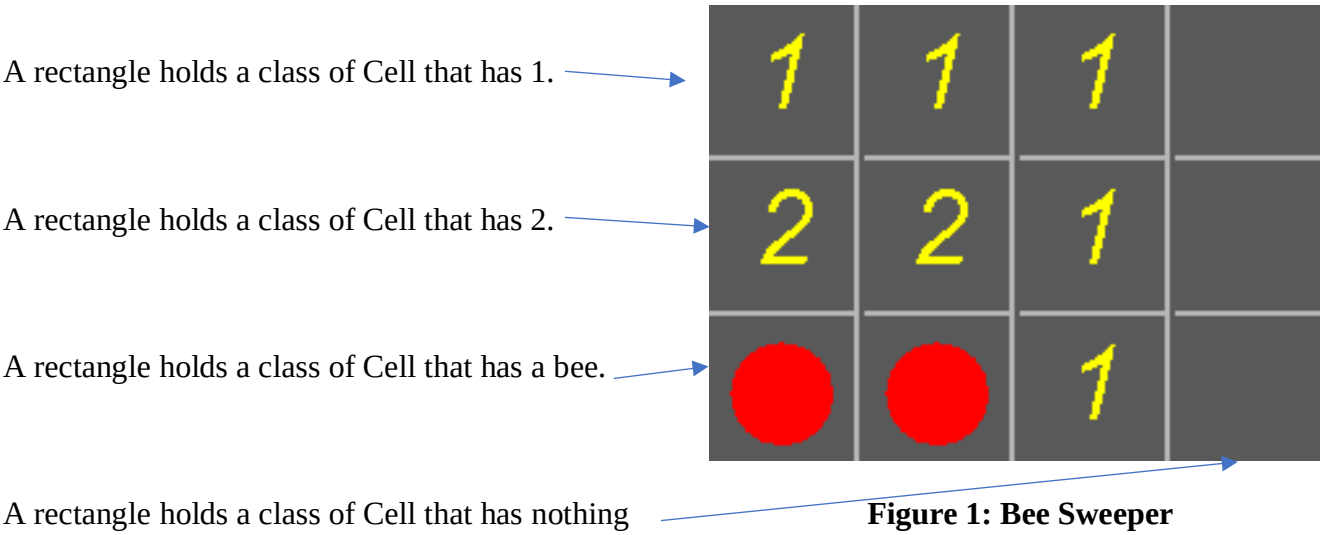


Figure 1: Bee Sweeper

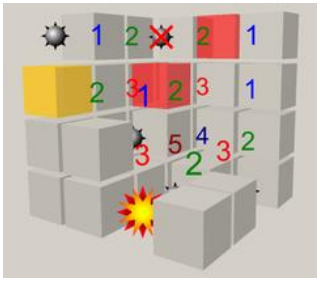
In this report, Section 2 discusses the history of Mine Sweeper; Section 3 introduces why this game; Section 4 presents algorithms for constructing Mine Sweeper's grid; Section 5 depicts a Java application developed to algorithmically generate a Mine Sweeper; Section 6 is the conclusion.

2. The history of Mine Sweeper

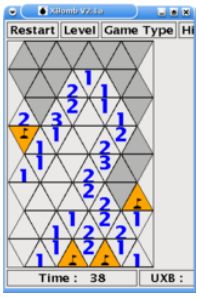
Minesweeper is a puzzle game that comes from the 1960s and 1970s. The earliest version of Minesweeper was Jerimac Ratliff's Cube. Even though both games do not share common aspects, it is believed that Jerimac Ratliff's Cube is the ancestor of Minesweeper.

The most used and known version of the Minesweeper was created by Robert Donner and Curtis Johnson, who worked for Windows. They developed and released Minesweeper first for Windows 3.1in 1990.

Since the 1960s, there has been a various existence of the game such as 3D, Hexagonal, Triangular and Rectangular version of it. And, my own version is called Bee Sweeper



3D



Triangular

Rectangular

3. Why This Game

Choosing this game for my final project was not something I planned. I did not even think about it because I thought that it would be really hard, creating something with drawings and windows. But, there are many things that inspired me to do this game for my final project. The first reason is that I got inspired by a Youtube channel. During those times, I wanted to learn JavaScript and I ended up finding this channel called, CodingTrain. The presenter was doing lots of challenges and I really like the way he was teaching. I thought it would be cool to do the same thing in Java and I start searching for it. I found useful information. The second reason is that I already knew 2D arrays from my class and that made things way easier for me. I liked that I could practice what I have just learned. The last reason is that I used to play this game a lot (seriously a lot!) when I was 12 years old. I considered myself the best player in this game. I cannot tell you how happy I was while creating this game and thinking that I could create my own version of it and play with it. Isn't that cool?

I can tell you that if you like something that you always wanted to create. Go for it! You would learn anything and everything for it. While doing that, you would gain amazing insight and information for coding.

4. How to Play

The main focus of the Bee Sweeper is that a player figures out where the bees are with the help of hidden instructions inside the cells/rectangles. Before we go further and explain how to play the game, it would be good to understand why I am calling this game Bee Sweeper instead of Minesweeper. I got this name from the Youtube video I used to watch while creating the game. I wanted to store a picture of a bee inside the cell. But, I never had enough time to do that. And I think that the circle hidden inside the cell do not look like bombs so I keep calling my version of the game as Bee Sweeper. I really like the name as well.

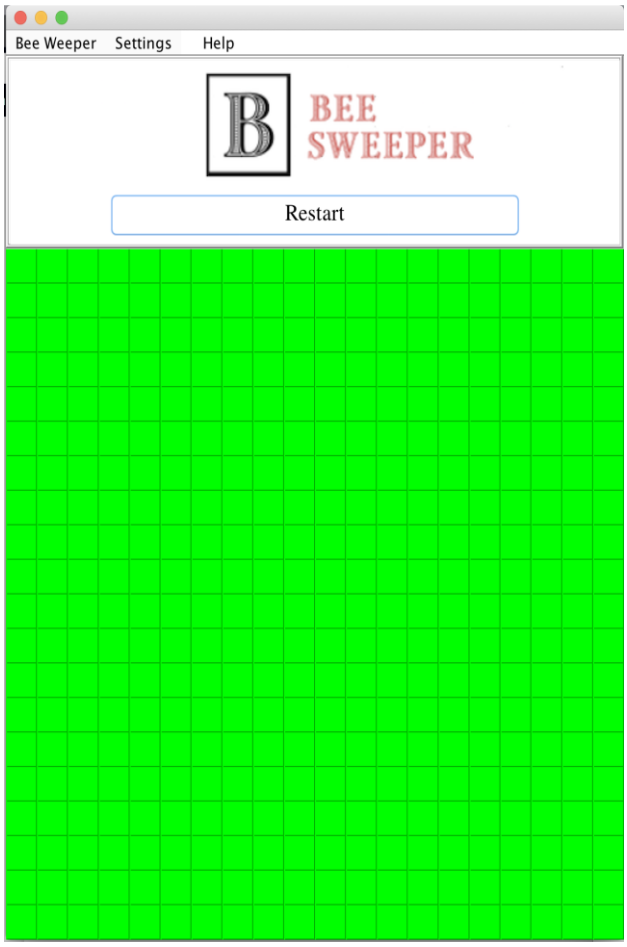


Figure 2

To win the game, a player needs to reveal all the cells that do not have bees. There are 400 cells inside the grid since the grid's width and height are 600 pixels which means there are 20 rows and columns.

Firstly, let's go to Settings of Bee Sweeper. The setting menu is on the top side of the game. You can check this in Figure 2. If you go on it, it will give you 3 radio buttons: (1) easy, (2) medium and (3) hard. Check Figure 3. Easy radio button is preselected which means there are

20 bees seeded randomly in the grid. If the player clicks on Medium radio button, the game will repaint and there will be 35 bees in the grid. If Hard is clicked, there will be 50 bees. With 3 different option, the user will have options to choose and also challenge herself.

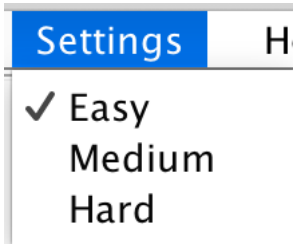


Figure 3

Now let's start playing the game. The player can click on the green rectangles with the mouse which will reveal what is inside a cell. Figure 4.

If the cell that user revealed does not have something in it, it will check its neighbors whether they have bees or not. If they do not have bees, it will force them to be revealed as well. If one of them has a bee, it will stop revealing. Why this is important is that it gives more option to the player.

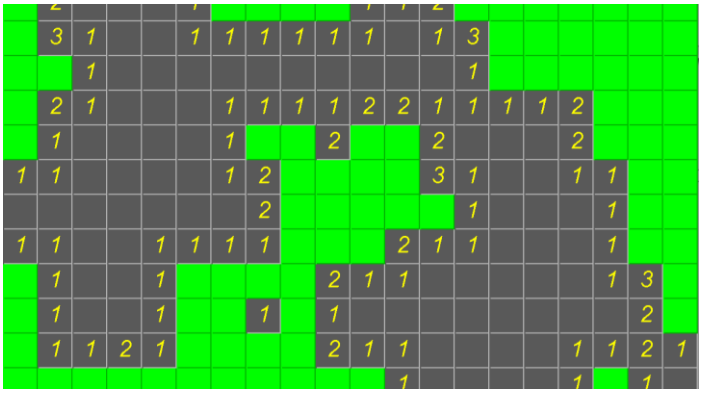
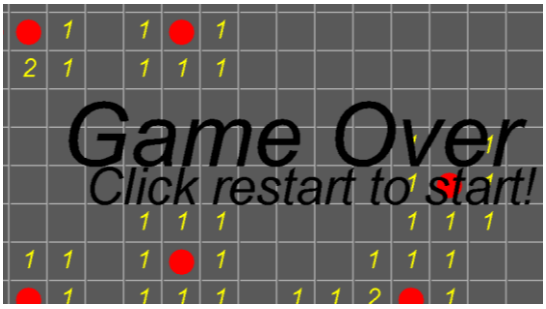


Figure 4

If a bee is hidden inside the cell, the player will lose. Check Figure 5. And, if the player is able to reveal all the cells that do not have bees, the player wins. Figure 6. The good thing about winning is that, it is more colorful.



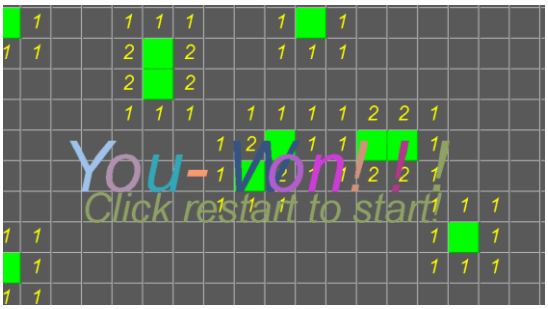


Figure 5 **Figure 6**

5. The structure of the Game

In this section, I will go over the UML of the classes and how these are related to each other. Lastly, we will look at what methods were the hardest ones and how I overcome them.

Firstly, let's check the UML. Figure 7, 8 and 9. As we see in the figures, there are three classes, Cells, BeeSweeper and PlayTheGame. I use Cells class in the BeeSweeper for graphics. The BeeSweeper draws the rectangles, circles, flags, and numbers based on what the Cells class fields and methods hold. Then, the BeeSweeper is passed to PlayTheGame class to create the JFrame and other settings aspects of the game. The BeeSweeper class itself is a JPanel that can be added to PlayTheGame class since it is a JFrame.

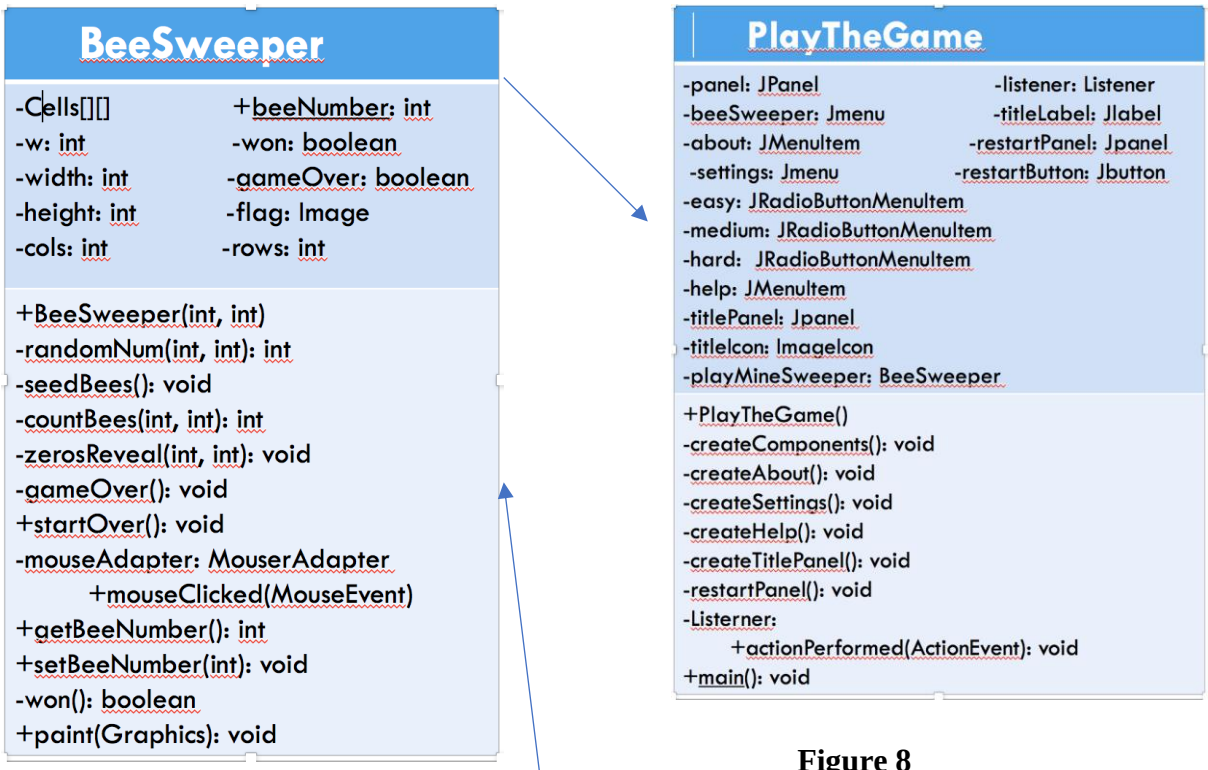


Figure 7

Figure 8

Figure 9

Lastly, the methods that I struggled the most and how I overcome. One of the most time-consuming parts of the game was to reveal all the cell that has nothing in it and how that can trigger its neighbors to reveal itself. As we see in Figure 10, when the player clicks a cell that has nothing in It, it will be revealed. It will not only be revealed, it will also check its neighbors whether they have bees or not. Why this is important is because it gives more information to the player to figure the next steps out. The name of this method is called zerosReveal() in class BeeSweeper (Figure 7). This method is a recursion method that calls itself many times until it reaches its mission which is to reveal all the zeros its neighboring. The method's parameters are the column and row of the cell and that checks the cell's neighbors by adding or deducting 1 from the original column and row. (you can check more about this method in Appendix 2). The other method that I had a hard time with was restart the game when the player changes the settings and when the player clicks the start button. This method is called startOver() in BeeSweeper class(Figure 7). What this method does is that, it sets all the bees, locations, cells and other parts of the game to 0 and then recreates them by calling a method called repaint().

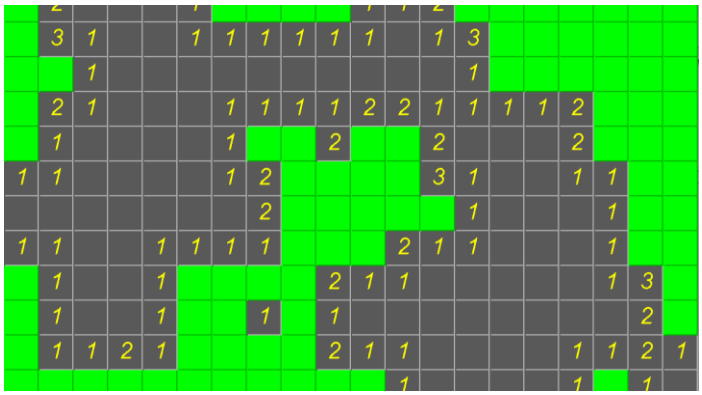


Figure 10

Sometimes you struggle with things but when you figure out how you can work it, it is a great feeling. By the time you figured it out, you already learned a lot of new and cool information about it. And, that sticks with you for a long time.

6. Conclusion

Even though I had no idea if I would be able to do this game for my final project, I can tell you that I had the opportunity to literally practice all my knowledge and more only in one project. Did I love it, yes. I love it not only because I learned something new but because I learned that by challenging yourself and doing what you love, you can be more creative which keeps you going for more challenging and interesting things. I can tell you that I was able to practice GUI, 2D arrays, Object Oriented Programming and debugger in IDEs in this project.

I will be playing my own version of Minesweeper which I call Bee Sweeper from now on instead of playing other people's version of the game.

Work Cited

History of Minesweeper, “www.minesweeperonline.net/history.php”, accessed: 12/07/2017.

Minesweeper (video game), “en.wikipedia.org/wiki/Minesweeper_(video_game)”, accessed: 12/07/2017.

Appendix 1: PlayTheGame.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.border.Border;

/**
 *
 * @author mehmetsahin
 */
public class PlayTheGame extends JFrame {

    private JPanel panel;
    private JPanel headPanel;

    private JMenuBar menubar;

    private JMenu beeSweeper;
    private JMenuItem about;

    private JMenu settings;
    private JRadioButtonMenuItem easy;
    private JRadioButtonMenuItem medium;
    private JRadioButtonMenuItem hard;

    private JMenuItem help;

    //title panel
    private JPanel titlePanel;
    private ImageIcon titleIcon;
    private JLabel titleLabel;

    //restartpanel
    private JPanel restartPanel;
    private JButton restartButton;

    private Listener listener = new Listener();

    private BeeSweeper playMineSweeper;

    public PlayTheGame() {
        panel = new JPanel();
        headPanel = new JPanel();

        menubar = new JMenuBar();

        playMineSweeper = new BeeSweeper(600, 600);
        playMineSweeper.setPreferredSize(new Dimension(600, 600));

        //setSize(600, 900);
        setBackground(Color.white);
        setSize(600, 800);
```

```
    setResizable(true);
    Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
    this.setLocation(dim.width / 2 - this.getSize().width / 2, dim.height / 4 - this.getSize().height / 4 - 200);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    createComponents();
    setJMenuBar(menubar);

    headPanel.setLayout(new BoxLayout(headPanel, BoxLayout.Y_AXIS));
    Border outer = BorderFactory.createEtchedBorder();
    Border inner = BorderFactory.createRaisedBevelBorder();
    Border compound = BorderFactory.createCompoundBorder(inner, outer);
    headPanel.setBorder(compound);
    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

    panel.add(headPanel);
    panel.add(playMineSweeper);
    add(panel);
    pack();
}

/**
 * Creates all the components of the frame
 */
private void createComponents() {
    createAbout();
    createSetting();
    createHelp();
    createTitlePanel();
    restartPanel();
}

/**
 * Creates the About Menu
 */
private void createAbout() {
    //create
    beeSweeper = new JMenu("Bee Weeper");
    about = new JMenuItem("About");

    about.addActionListener(listener);

    //add
    beeSweeper.add(about);
    menubar.add(beeSweeper);
}

/**
 * Creates the settings Menu with Radio Buttons
 */
private void createSetting() {
    //create
    settings = new JMenu("Settings");

    easy = new JRadioButtonMenuItem("Easy", true);
    medium = new JRadioButtonMenuItem("Medium");
    hard = new JRadioButtonMenuItem("Hard");
```

```
    //add group
    ButtonGroup bG = new ButtonGroup();
    bG.add(easy);
    bG.add(medium);
    bG.add(hard);

    //action
    easy.addActionListener(listener);
    medium.addActionListener(listener);
    hard.addActionListener(listener);

    //add
    settings.add(easy);
    settings.add(medium);
    settings.add(hard);
    menubar.add(settings);

}

/**
 * Creates the Help Menu
 */
private void createHelp() {
    //create
    help = new JMenuItem("Help");

    help.addActionListener(listener);

    //add
    menubar.add(help);
}

/**
 * Creates the title panel with the title image and label
 *
 */
private void createTitlePanel() {
    //create
    titlePanel = new JPanel();
    titlePanel.setBackground(Color.WHITE);
    titleIcon = new ImageIcon("bee2.png");
    titleLabel = new JLabel(titleIcon);

    //action
    //add
    titlePanel.add(titleLabel);
    headPanel.add(titlePanel);
}

/**
 * Creates the restart button sets the size of the button and the Font style
 */
private void restartPanel() {
    //create
    restartPanel = new JPanel();
    restartPanel.setBackground(Color.WHITE);
    restartButton = new JButton("Restart");
```

```
        restartButton.setFont(new Font("Serif", Font.PLAIN, 20));
        restartButton.setPreferredSize(new Dimension(400, 40));

        //action
        restartButton.addActionListener(listener);

        //add
        restartPanel.add(restartButton);
        headPanel.add(restartPanel);
    }

    private class Listener implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            if (easy.isSelected()) {
                playMineSweeper.setBeeNumber(20);
                playMineSweeper.startOver();
            } else if (medium.isSelected()) {
                playMineSweeper.setBeeNumber(35);
                playMineSweeper.startOver();
            } else if (hard.isSelected()) {
                playMineSweeper.setBeeNumber(50);
                playMineSweeper.startOver();
            } else {
                ;
            }
            if (e.getSource() == restartButton) {
                playMineSweeper.startOver();
            }
            if (e.getSource() == about) {
                JOptionPane.showMessageDialog(null, "BeeSweeper is a single-player puzzle video game. "
                        + "\nThe objective of the game is to clear a rectangular board containing "
                        + "\nhidden \"bees\" or bombs without detonating any of them, with help "
                        + "\nfrom clues about the number of neighboring mines in each field. "
                        + "\nThe game originates from the 1960s, and has been written for many computing "
                        + "\nplatforms in use today. It has many variations and offshoots. (took from Wikipedia)"
                        + "\n"
                        + "\nCreated BY @MEHMET SAHIN", "", JOptionPane.INFORMATION_MESSAGE);
            }
            if (e.getSource() == help) {
                JOptionPane.showMessageDialog(null, "Please check out Google for more information", "",
JOptionPane.INFORMATION_MESSAGE);
            }
        }

    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new PlayTheGame();
            }
        });
    }
}
```

Appendix 2: BeeSweeper.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Random;

public class BeeSweeper extends JPanel {

   private Cells[][] cells;

   private int w = 30;  // width and height of cell
   private int width; //width of the window
   private int height; //height of the window
   private int cols;
   private int rows;

   public static int beeNumber = 20; //number of bees in all cells

   private Image flag;

   private boolean gameOver = false;
   private boolean won = false;

   public BeeSweeper(int width, int height) {
      this.width = width;
      this.height = height;

      cols = this.width / w;
      rows = this.height / w;
      cells = new Cells[cols][rows];

      for (int i = 0; i < cols; i++) {
         for (int j = 0; j < rows; j++) {
            cells[i][j] = new Cells(i, j, w);
         }
      }

      seedBees();

      //sets how many bees around
      for (int i = 0; i < cols; i++) {
         for (int j = 0; j < rows; j++) {
            cells[i][j].setCountBees(countBees(i, j));
         }
      }

      setOpaque(true);
      addMouseListener(mouseAdapter);

   }
```

```
/**
 * Generates a random number between low and high
 *
 * @param low an integer of lowest number
 * @param high an integer of highest number
 * @return a random number between low and high
 */
private int randomNum(int low, int high) {
   Random random = new Random();

   return random.nextInt(high + 1 - low) + low;
}

/**
 * Seeds all the bees around the grid
 */
private void seedBees() {
   for (int n = 0; n < this.beeNumber; n++) {
      int c = randomNum(0, cols - 1);
      int r = randomNum(0, rows - 1);

      if (!cells[c][r].isBee()) {
         cells[c][r].setBee(true);
      }
   }
}

/**
 * Counts how many bees around a neighboor cell
 *
 * @param c is the colum index
 * @param r is the row index
 * @return total number of bees in a neighboorhood
 */
private int countBees(int c, int r) {
   int total = 0;

   for (int xoff = -1; xoff <= 1; xoff++) {
      for (int yoff = -1; yoff <= 1; yoff++) {
         int i = c + xoff;
         int j = r + yoff;
         if (i > -1 && i < cols && j > -1 && j < rows) {
            if (cells[i][j].isBee()) {
               total++;
            }
         }
      }
   }
   return total;
}

/**
 * Finds all the zeros around neighboors and reveals them to public
 *
 * @param c is the column index
 * @param r is the row index
 */
```

```
private void zerosReveal(int c, int r) {
   for (int xoff = -1; xoff <= 1; xoff++) {
      for (int yoff = -1; yoff <= 1; yoff++) {
         int i = c + xoff;
         int j = r + yoff;
         if (i > -1 && i < cols && j > -1 && j < rows) {
            if (!cells[i][j].isBee() && !cells[i][j].isRevealed()) {
               cells[i][j].setRevealed(true);
               if (cells[i][j].getCountBees() == 0) {
                  zerosReveal(i, j);
               }
            }
         }
      }
   }
}

/**
 * Once user steps over a bee, game is over by reaveling all cells
 */
private void gameOver() {
   for (int i = 0; i < cols; i++) {
      for (int j = 0; j < rows; j++) {
         cells[i][j].setRevealed(true);
      }
   }
}

/**
 * Starts the game over
 */
public void startOver() {
   for (int i = 0; i < cols; i++) {
      for (int j = 0; j < rows; j++) {
         cells[i][j].setRevealed(false);
         cells[i][j].setBee(false);
         cells[i][j].setCountBees(0);
         cells[i][j].setFlagged(false);
         won = false;
         gameOver = false;
      }
   }

   seedBees();

   //sets how many bees around by coling setCountBees method
   for (int i = 0; i < cols; i++) {
      for (int j = 0; j < rows; j++) {
         cells[i][j].setCountBees(countBees(i, j));
      }
   }
   repaint();
}

/**
 * For mouse
 */
private MouseAdapter mouseAdapter;
```

```
{
   mouseAdapter = new MouseAdapter() {
      @Override
      public void mouseClicked(MouseEvent me) {

         if (SwingUtilities.isLeftMouseButton(me)) {
            int x = me.getX();
            int y = me.getY();

            for (int i = 0; i < cols; i++) {
               for (int j = 0; j < rows; j++) {
                  if (cells[i][j].isLocation(x, y) && !cells[i][j].isRevealed() && !cells[i][j].isFlagged()) {
                     cells[i][j].setRevealed(true);
                     if (cells[i][j].isBee()) {
                        gameOver = true;
                        gameOver();
                     }

                     if (cells[i][j].getCountBees() == 0) {
                        zerosReveal(i, j); //recursion
                     }

                  }
               }
            }
         }
         //this is for the flagging and unflagging
         if (SwingUtilities.isRightMouseButton(me)) {
            int x = me.getX();
            int y = me.getY();
            for (int i = 0; i < cols; i++) {
               for (int j = 0; j < rows; j++) {
                  if (cells[i][j].isLocation(x, y) && !cells[i][j].isRevealed() && !cells[i][j].isFlagged()) {
                     cells[i][j].setFlagged(true);
                  } else if (cells[i][j].isLocation(x, y) && cells[i][j].isFlagged()) {
                     cells[i][j].setFlagged(false);
                  }
               }
            }
         }

         repaint();
      }
   };
}

/**
 * Gets the number of bees on the grid
 *
 * @return number of bees on the grid
 */
public int getBeeNumber() {
   return beeNumber;
}

/**
 * Sets the number of bees on the grid
```

```
 *
 * @param beeNumber an integer for bee number
 */
public void setBeeNumber(int beeNumber) {
   this.beeNumber = beeNumber;
}

/**
 * When all the bees are found, the user winds
 *
 * @return true if all the bees are revealed
 */
private boolean won() {
   for (int i = 0; i < cols; i++) {
      for (int j = 0; j < rows; j++) {
         if (!cells[i][j].isRevealed() && !cells[i][j].isBee()) {
            return false;
         }
      }
   }
   won = true;
   return true;
}

public void paint(Graphics g) {

   //flag icon
   ImageIcon i1 = new ImageIcon("flag.png");
   flag = i1.getImage();

   // draws the cells
   for (int i = 0; i < cols; i++) {
      for (int j = 0; j < rows; j++) {
         g.setColor(Color.green);
         g.fill3DRect(cells[i][j].getI() * w, cells[i][j].getJ() * w, w, w, true);
      }
   }

   for (int i = 0; i < cols; i++) {
      for (int j = 0; j < rows; j++) {
         if (cells[i][j].isRevealed() && !cells[i][j].isFlagged()) {
            if (cells[i][j].isBee()) { // if it is a bee
               g.setColor(Color.gray);
               g.fill3DRect(i * w, j * w, w, w, false);

               g.setColor(Color.red);
               g.fillOval(i * w + 5, j * w + 5, w - 10, w - 10);

               g.setColor(Color.BLACK);
               g.setFont(new Font("arial", Font.ITALIC, 70));
               g.drawString("Game Over", 100, height / 2);

               g.setFont(new Font("arial", Font.ITALIC, 40));
               g.drawString("Click restart to start!", 120, height / 2 + 30);

            } else { // if it is a number
               if (cells[i][j].getCountBees() == 0) {
                  g.setColor(Color.gray);
```

```
                    g.fill3DRect(i * w, j * w, w, w, false);
                } else {
                    g.setColor(Color.gray);
                    g.fill3DRect(i * w, j * w, w, w, false);

                    g.setColor(Color.yellow);
                    g.setFont(new Font("arial", Font.ITALIC, 20));
                    g.drawString(Integer.toString(cells[i][j].getCountBees()), i * w + 10, j * w + 20);
                }
            }
        } else {
            if (cells[i][j].isFlagged()) { //flag image
                g.drawImage(flag, i * w, j * w, w, w, null);
            }
        }
    }
}

/**
 * This print "You win" string when the user wins if the user has not
 * lost the game
 */
if (won() && !gameOver) {
    String won = "You-Won!!!";
    for (int i = 0; i < won.length(); i++) {
        int r = randomNum(50, 255);
        int ge = randomNum(50, 255);
        int y = randomNum(50, 255);
        g.setColor(new Color(r, ge, y));
        g.setFont(new Font("arial", Font.ITALIC, 70));
        g.drawString(String.valueOf(won.charAt(i)), 100 + (i * 40), height / 2);
        repaint();
    }

    g.setFont(new Font("arial", Font.ITALIC, 40));
    g.drawString("Click restart to start!", 120, height / 2 + 30);
}
}
}
```

Appendix 3: Cells.java

```
class Cells {

    private int w;  // width and height of cell

    private int i;  //i * w = x(location)  // index of column
    private int j;  //j * w = y(location)  // index of raw

    private boolean bee;  // a cirlcle inside an ractengle
    private int countBees;

    private boolean revealed = false;
```

```
private boolean flagged;

public Cells(int i, int j, int w) {

   this.i = i;
   this.j = j;
   this.w = w;

}

/**
 * Sets the cell as flagged
 *
 * @param flagged true or false
 */
public void setFlagged(boolean flagged) {
   this.flagged = flagged;
}

/**
 * Checks if the cell is flagged or not
 *
 * @return
 */
public boolean isFlagged() {
   return this.flagged;
}

/**
 * index of the cell
 *
 * @return index of the cell
 */
public int getI() {
   return i;
}

/**
 * index of the cell
 *
 * @return index of the cell
 */
public int getJ() {
   return j;
}

/**
 * Puts a bee in a cell
 *
 * @param bee a boolean for setting a bee
 */
public void setBee(boolean bee) {
   this.bee = bee;
}

/**
 * Checks if there is a bee inside a cell
```

```
     *
     * @return true or false based on if there is a bee inside the cell
     */
    public boolean isBee() {
        return bee;
    }

    /**
     * Check if a cell is been revealed
     *
     * @return true or false based on if there a cell is been revealed
     */
    public boolean isRevealed() {
        return revealed;
    }

    /**
     * Reveals or unreveals a cell
     *
     * @param revealed a boolean for revealing
     */
    public void setRevealed(boolean revealed) {
        this.revealed = revealed;
    }

    /**
     * Count of the bee around this cell
     *
     * @return how many bees around this cell
     */
    public int getCountBees() {
        return countBees;
    }

    /**
     * Sets the bee count around this cell
     *
     * @param countBees
     */
    public void setCountBees(int countBees) {
        this.countBees = countBees;
    }

    // for the mouse!!
    /**
     * Check if mouse clicked is inside this cell.
     *
     * @param x location of mouse x
     * @param y location of mouse y
     * @return true if mouse is inside this cell otherwise false
     */
    public boolean isLocation(int x, int y) {
        return (x > (this.i * w) && x < (this.i * w) + this.w && y > (this.j * w) && y < (this.j * w) + this.w);
    }
}
```