Minesweeper: Project Report

Jamel Saleh (20210815)

Omar Arabi (20200161)

Sammy Medawar (20210031)

Wally el Sayed (20210280)

Department of Computer Information Systems, Rafik Hariri University

COSC316: Design and Analysis of Algorithms

Dr. Roaa Soloh

November 25, 2022

Table of Content

Introduction……..1

Goal Behind Our Idea……..2

Game Screenshots…….……..3

Time Complexity Analysis and Comparison……...…7

Softwares and Languages Used…….……15

What We Would Implement If We Had More Time……...…….....15

References……...……..16

Introduction

What is Minesweeper?

Minesweeper is a logical video game that can improve deductive reasoning. Its origins are unknown; however, it's most well-known for its version on Windows XP. It's also known as the game everybody tried solving by pressing on buttons randomly. But in actuality it's a game that is 95% logic and 5% randomness.

What Functions Did We Use:

The main functions of a Minesweeper game includes:

- A function to dynamically create a grid system based on the rows and columns entered by the user
- A function to generate bombs randomly
- A function to count the number of bombs adjacent for each button - We implemented this using an edited version of Breadth-First Search
- A function to open all adjacent zero boxes upon clicking on a button that has 0 as the number of bombs adjacent to it - this was implemented fully using a recursive Depth-First Search.
- Functions to check if the user won or lost and reveal all buttons either way.

How is the Game Played?

- A grid of buttons are dynamically generated.

- Each button either has a number or a bomb inside it which can be revealed by left clicking.
- The number on a button indicates the number of bombs surrounding the button.
- If the user presses on a bomb, they lose.
- A user can use a flag as a warning that this button has a bomb inside it.
- If all the bombs have been flagged or if all non-bombs were opened, the player wins.

Goal Behind Our Idea

Remake Minesweeper from Scratch:

The goal of our project was to rely on our critical thinking and programming skills to remake a popular beloved game from scratch without help in making the core game. We remade multiple versions of the same game until we reached a version that utilized algorithms in a proper way. External help was only used to understand how Java Swing and Java AWT components work such as how to dynamically create a Grid Layout JFrame and was also used to understand Breadth-First Search and Depth-First Search in general. All other functions in the code were made using only our brains.

Create a Project We Can Put On Our Portfolios

Another goal of this project was to make it decent enough to put on our portfolios. While we've made significant progress until this point, the work is far from over. Even after this project report, we will continue working on the project by ourselves and implement features we couldn’t implement until today.

Game Screenshots

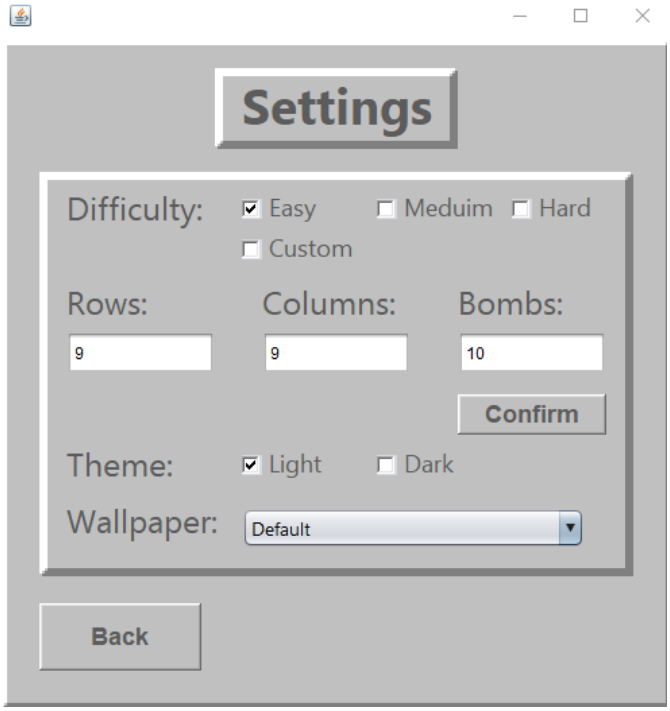

Figure 1: Settings Tab in Normal Theme

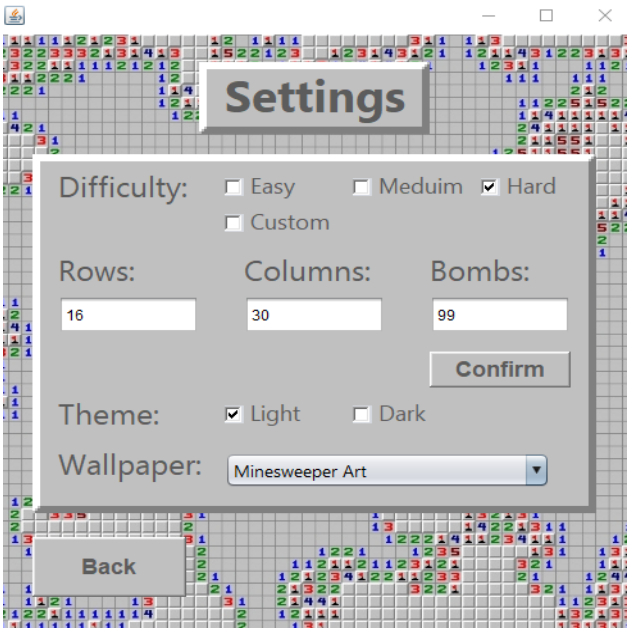

Figure 2: Settings Tab in Minesweeper Art Wallpaper

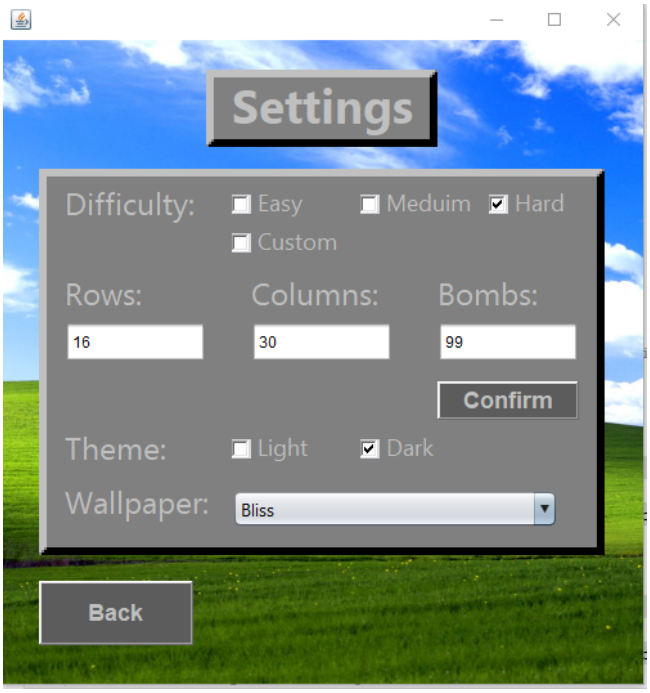

Figure 3: Settings Tab with Dark Mode and Bliss Wallpaper

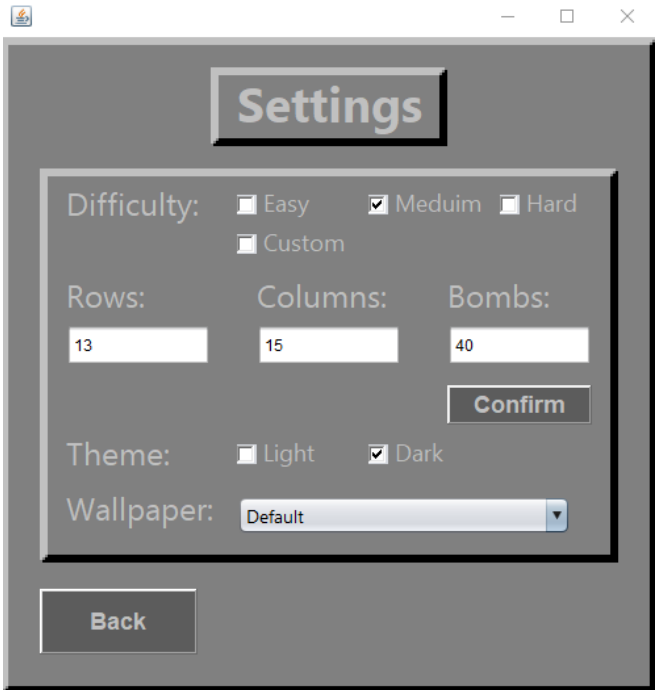

Figure 4: Settings Tab with Normal Dark Mode

Figure 5: Main Menu

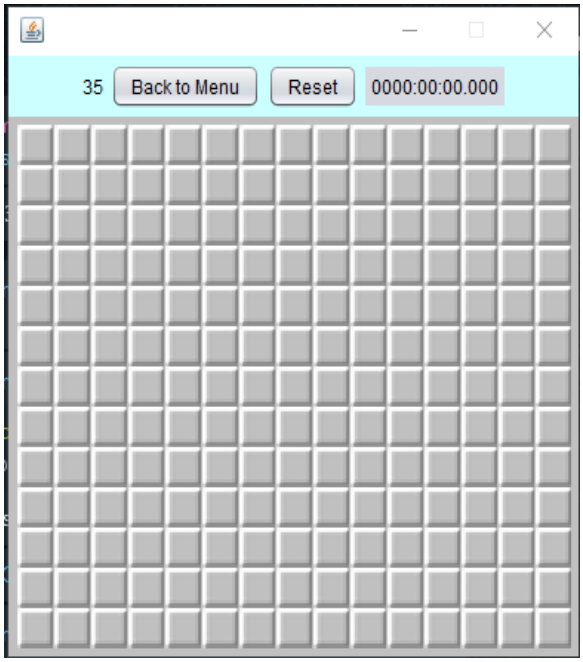

Figure 6: Normal Mode

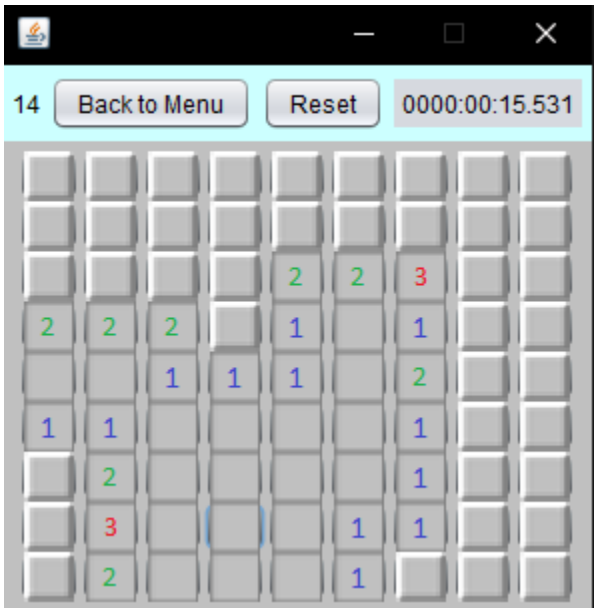

Figure 7: Demonstrating DFS in Easy Mode

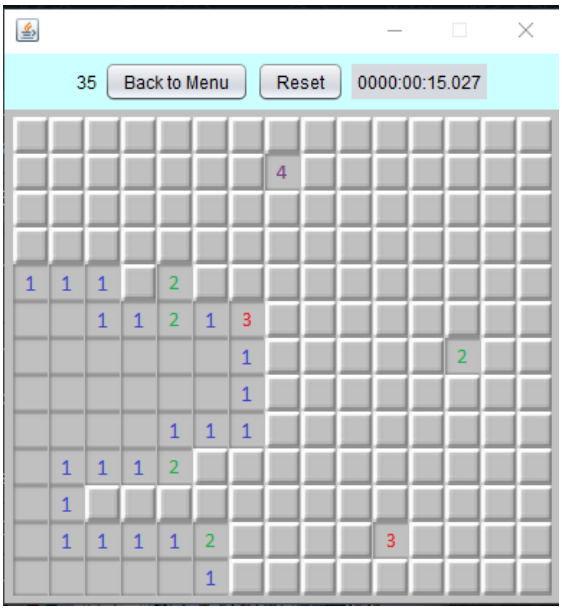

Figure 8: Demonstrating DFS in Normal Mode

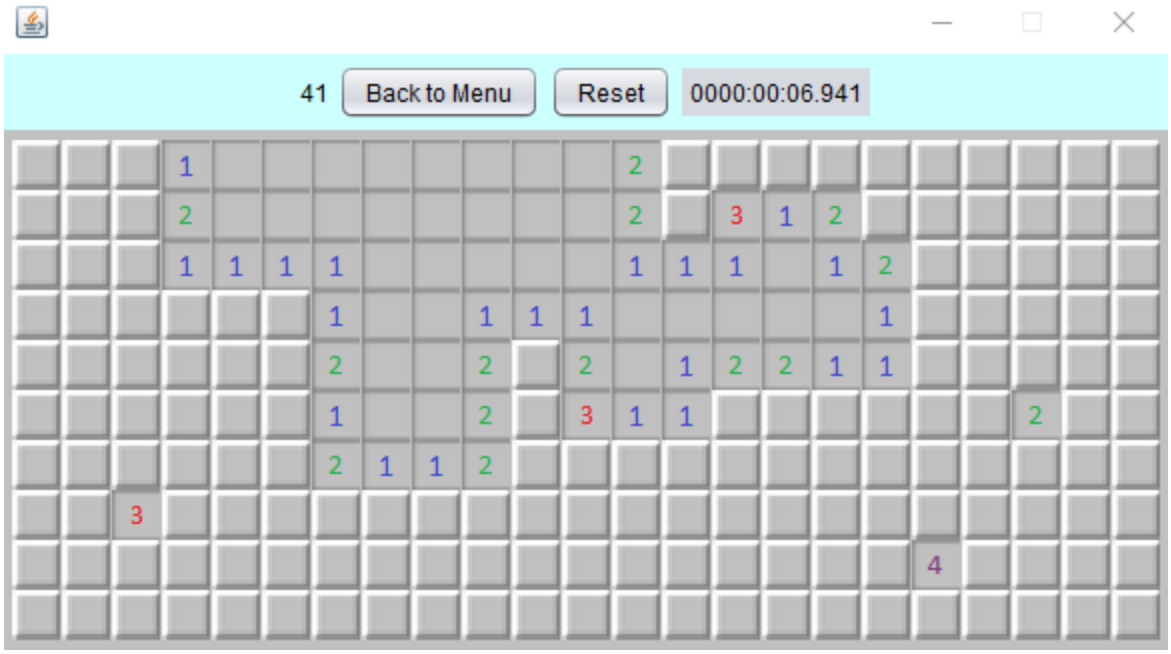

Figure 9: Custom Mode with DFS

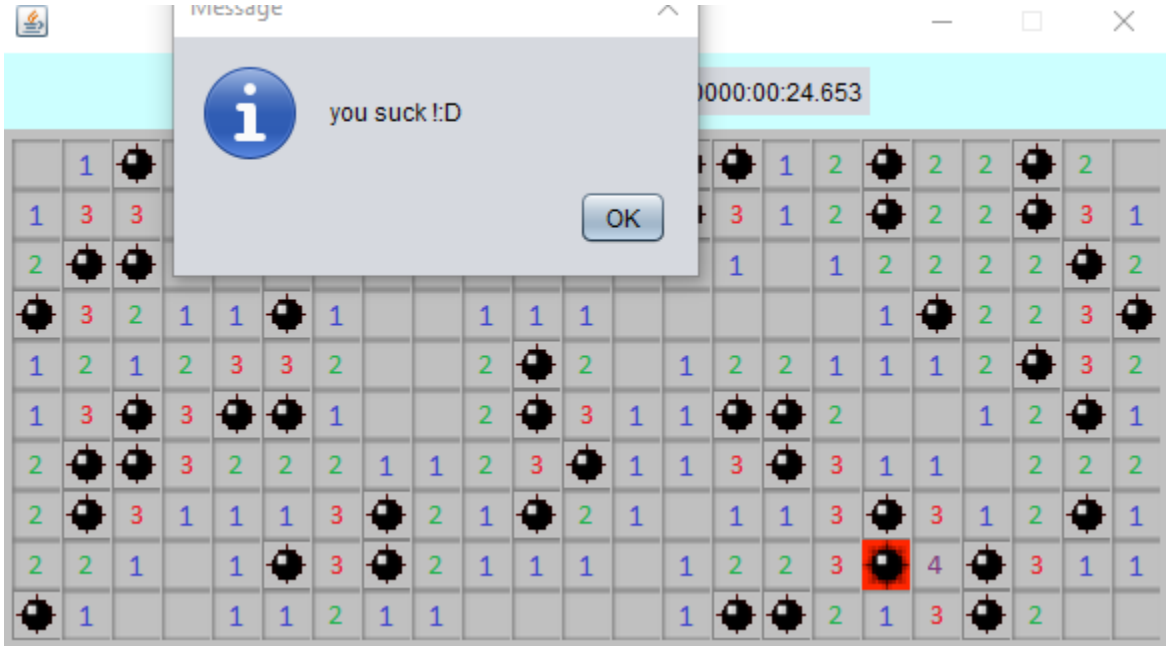

Figure 10: Game Over Screen (Dialog Box Moved From Center for Convenience)

Time Complexity and Analysis

Breadth First Search

```
public int bfs(CustomButtonV2 c){
    Queue<CustomButtonV2> perimeter = new LinkedList<>();
    ArrayList<CustomButtonV2> known = c.neighbors;
    int count =0, i =0, length = known.size();
    perimeter.add(known.get(i));
    while(!perimeter.isEmpty()){
        CustomButtonV2 from = perimeter.remove();
        if(from.isBomb)
            count++;
        if(i<length-1){
            perimeter.add(known.get(i+1));
            i++;
        }
    }
    return count;
}
```

Will loop only either 3,5, or 8 times since each button can only have 3,5, or 8 neighbors

Figure 11: Breadth First Search

We used Breadth First Search to basically find the number of bombs adjacent (directly connected) to each button. Prior to this function, we looped over every button in the game panel, and for each button we used the bfs function to check only the vertices DIRECTLY connected to said button which is saved through the c.neighbors array.

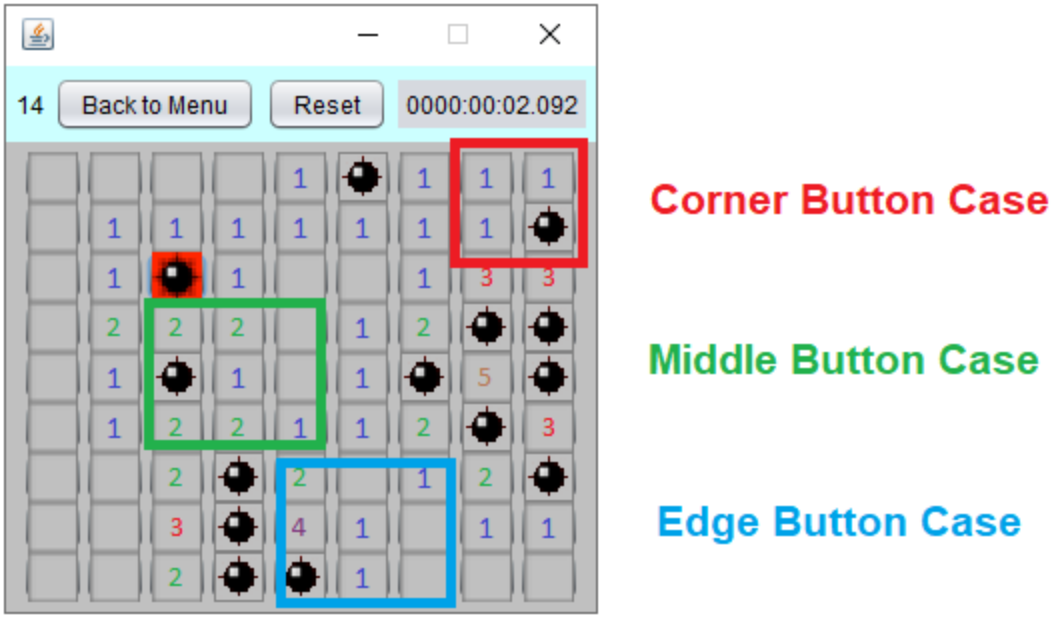

Figure 12: Cases of Number of Bombs Adjacent

The time complexity of BFS according to Geeks for Geeks is O(V+E) where V stands for Vertices and E stands for Edges.

In the case of a corner button (3 neighbors), the time complexity would be O(4 + 3) = O(7).

In the case of an edge button (5 neighbors), the time complexity would be O(6 + 5) = O(11).

In the case of a button in the middle (8 neighbors) the time complexity would be O(9 + 8) = O(17) .

Technically, however, the BFS function will loop over itself N times where N is the number of total buttons in the game. This is because the function prior to BFS utilizes BFS in a for each loop.

```
public void createAdjacents(){
    for(Component c: panelGame.getComponents()){
        if(c instanceof CustomButtonV2){
            CustomButtonV2 b = (CustomButtonV2) c;
            if(!b.isBomb){
                int badj = bfs(b);
                b.nbrBombsAdjacent = badj;
            }
        }
    }
}
```

Figure 13: createAdjacents Function

Hence why the time complexity is O(1) for bfs alone but O(n) for createAdjacents along with BFS.

Depth First Search

```
public void dfs(CustomButtonV2 c){
    if(c.nbrBombsAdjacent != 0 || c.isReveal)
        return;
    if(c.nbrBombsAdjacent == 0 && !c.isReveal){
        c.isReveal = true;
        if(!c.isBomb)
        c.setIcon(new javax.swing.ImageIcon(getClass().getResource("/minesweeper/resources/revealed.png")));
        for(CustomButtonV2 n: c.neighbors){
            if(!n.isBomb){
                n.isReveal=true;
                reveal(n);
                dfs(n);
            }
        }
    }
}
```

Figure 14: Depth First Search

We used a recursive DFS for when a user reveals a button that has zero adjacent bombs, that button has to reveal all its neighbors at the same time. Moreover, if that button has other buttons that also has zero adjacent bombs, then we should also make that button undergo DFS.

By marking all buttons we visited as isReveal = true, we make sure that each button is only looped over once. Hence the function's time complexity is also O(n) where n is the total number of buttons in the field. Note that the for each loop inside the function will only loop over itself 8 times at most and 3 times at least, so it is negligible.

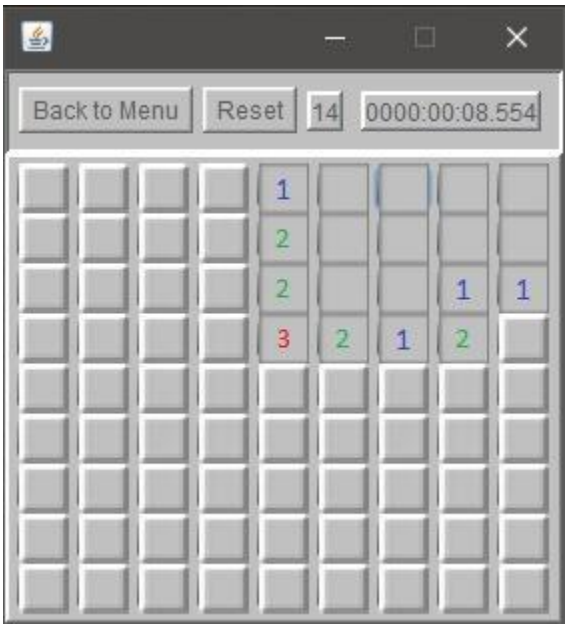

Figure 15: DFS Shown in Easy Mode

In figure 15, the user pressed on the button in the top right button and found that it was a zero. The BFS will then check the button to its left and open it, then it will check the button to its left again and open it, and then it will check the button on its left again and open it. It will then go back and do the process for the buttons on the bottom left and so on.

Comparison

In previous iterations, we made each function without using an algorithm. Each time, we used a double nested for each loop over every button in the grid. Basically each function was O(n^2). By using the DFS and BFS algorithms we were able to optimize the time complexity.

```
public void revealAll(CustomButtonV2 givenButton){
    for (Component c: panelGame.getComponents()){
        if(c instanceof CustomButtonV2){
            CustomButtonV2 b = (CustomButtonV2) c;
            if(b == givenButton){
            //no
            }
            if(b != givenButton){
            if(b.isBomb){
                b.setIcon(images[9]);
                isTimerActive=false;
            }
            else if(!b.isBomb){

                switch(b.nbrBombsAdjacent){
                    case 0:
                        if(b != givenButton)
                        b.setIcon(images[0]);
                        break;
                    case 1:
                        b.setIcon(images[1]);
                        break;
                    case 2:
                        b.setIcon(images[2]);
                        break;
                    case 3:
                        b.setIcon(images[3]);
                        break;
                    case 4:
                        b.setIcon(images[4]);
                        break;
                    case 5:
                        b.setIcon(images[5]);
                        break;
                    case 6:
                        b.setIcon(images[6]);
                        break;
                    case 7:
                        b.setIcon(images[7]);
                        break;
                    case 8:
                        b.setIcon(images[8]);
                        break;
                }
            }
            }
        }
    }


}
```

Figure 16: function that revealsall the buttons with a complexity of O(n)

```
public void reveal(CustomButtonV2 b){ //O(1)
    b.setBackground(new java.awt.Color( r:192, g:192, b:192));
    if(b.isBomb){
       revealAll( givenButton:b);
       isTimerActive = false;
       myTimer.timer.stop();
       b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/detonatedBomb.png")));
       JOptionPane.showMessageDialog( parentComponent: frame,  message:"you suck !:D");

    }
    if(!b.isBomb){
    switch(b.nbrBombsAdjacent){
        case 0:
            b.setIcon(images[0]);
            break;
        case 1:
            b.setIcon(images[1]);
            break;
        case 2:
            b.setIcon(images[2]);
            break;
        case 3:
            b.setIcon(images[3]);
            break;
        case 4:
            b.setIcon(images[4]);
            break;
        case 5:
            b.setIcon(images[5]);
            break;
        case 6:
            b.setIcon(images[6]);
            break;
        case 7:
            b.setIcon(images[7]);
            break;
        case 8:
            b.setIcon(images[8]);
            break;
    }
    }

}
```

Figure 17: a reveal function with a complexity of O(1)

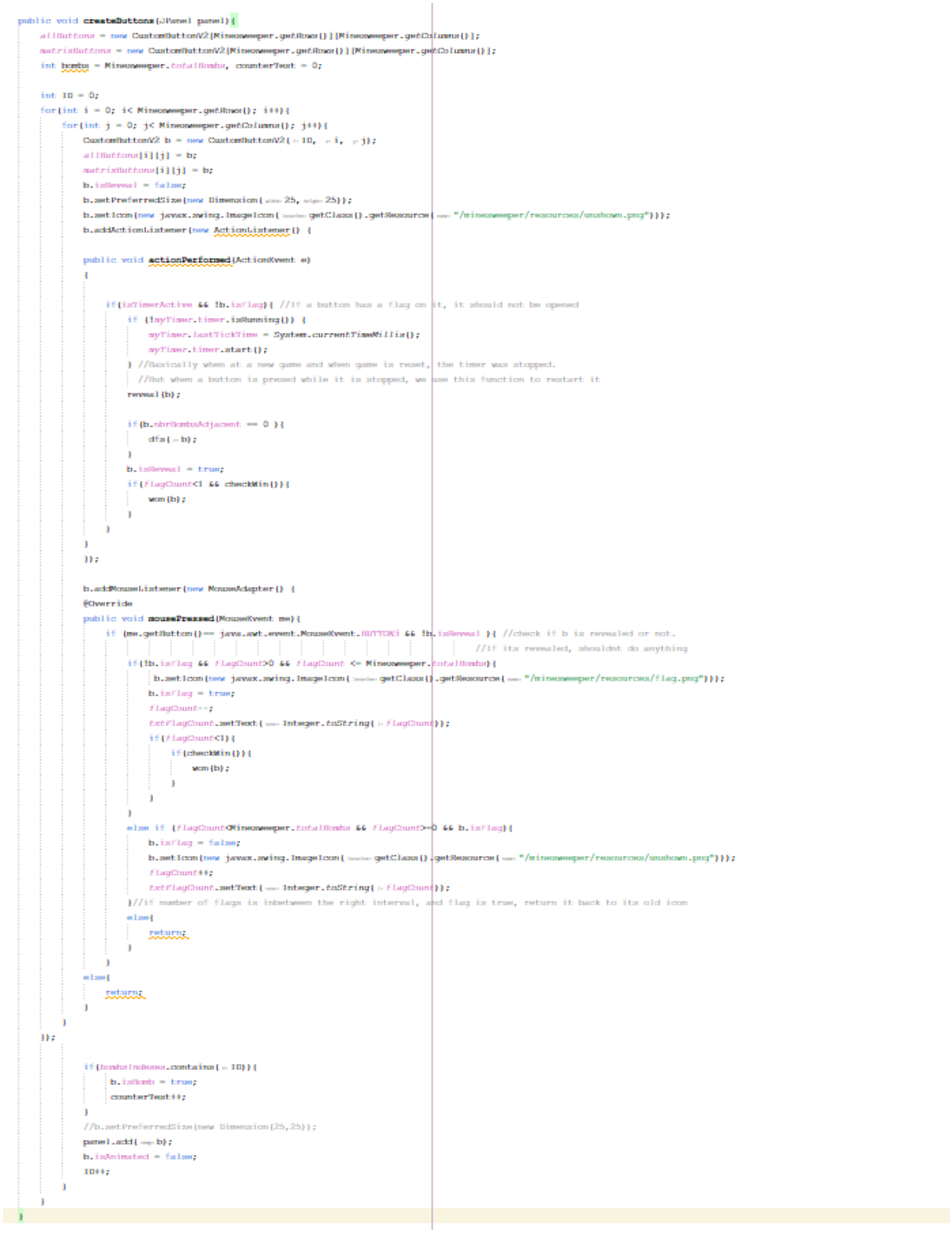

Figure 18: a create button function with a complexity of O(RxC) = O(n)

```java
public boolean checkWin(){
    int count = 0;
    int totalFlags = Minesweeper.totalBombs;
    for(Component c: panelGame.getComponents()){
        if (c instanceof CustomButtonV2){
            CustomButtonV2 b = (CustomButtonV2) c;
            if(b.isBomb && b.isFlag){
                count++;
            }
        }
    }
    if (count == totalFlags)
        return true;
    return false;
}
```

Figure 19: a checking function that has a complexity of O(n)

```java
public void Test(){
    for(Component c: panelGame.getComponents()){ //O(B) where it is equal to the total bombs which itsef is equal to O(0.17n)
        if(c instanceof CustomButtonV2){
            CustomButtonV2 b = (CustomButtonV2) c;
            if(b.isBomb == true)
              b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/detonatedBomb.png")));
            else{
                switch(b.nbrBombsAdjacent){
                    case 0:
                        b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/revealed.png")));
                        break;
                    case 1:
                        b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/1.png")));
                        break;
                    case 2:
                        b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/2.png")));
                        break;
                    case 3:
                        b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/3.png")));
                        break;
                    case 4:
                        b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/4.png")));
                        break;
                    case 5:
                        b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/5.png")));
                        break;
                    case 6:
                        b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/6.png")));
                        break;
                    case 7:
                        b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/7.png")));
                        break;
                    case 8:
                        b.setIcon(new javax.swing.ImageIcon( location:getClass().getResource( name:"/minesweeper/resources/8.png")));
                        break;
                }
            }

        }
    }
}
```

Figure 18: a function that has the time complexity of O(B) where it is equal to the total bombs Which itself is equal to O(0.17n)

```
public void createBombsIndexes(){
    Random ran = new Random();
    int low=1;
    int high=Minesweeper.getRows()*Minesweeper.getColumns();
    int ranValue;
    for (int i = 0 ; i< Minesweeper.totalBombs;i++){
        ranValue = ran.nextInt(high - low)+low;

        if(bombsIndexes.contains( o: ranValue)){
            i--;
            continue;
        }

    bombsIndexes.add( e: ranValue);
    }
}
```

Figure19: a function with the complexity of O(n)

```
public void createToolsPanel(){
    JPanel pane = panelTool;
    txtFlagCount = new JLabel( text: Integer.toString( i: Minesweeper.totalBombs));
    Button btnBackToMenu = new Button( label: "Back to Menu");
    Button btnReset = new Button( label: "Reset");
    myTimer = new StopWatchPanel();
    panelTool.setBorder( border: BorderFactory.createLoweredBevelBorder());
    txtFlagCount.setBorder( border: BorderFactory.createRaisedBevelBorder());
```

Figure 20: this function is the tool panel with a time complexity of O(1)

```
public void createGamePanel(){
    matrixButtons = new CustomButtonV2 [rows][columns];
    JPanel panel = panelGame;
    bombsIndexes = new ArrayList<>();
    panel.setBackground(new java.awt.Color( r:192,  g:192,  b:192));

    panelGame.setLayout(new java.awt.GridLayout( rows:Minesweeper.getRows(), cols:Minesweeper.getColumns())); //Creating a new grid layout and putting it inside the old layout
    createBombsIndexes();
    createButtons(panel);
    createNeighbors();
    createAdjacents();
    //Test();
    panel.setBorder( border:BorderFactory.createRaisedBevelBorder());
}
```

Figure 21: this function has a time complexity of O(4n) = O(n)

```
public void createMainPanel(){
    JPanel pane = panelMain;
    createToolsPanel();
    createGamePanel();
    panelGame.setBorder( border:javax.swing.BorderFactory.createLineBorder(new java.awt.Color( r:192,  g:192,  b:192),  thickness:5));
    this.setContentPane( contentPane:panelMain);
    this.setResizable( resizable:false);
    this.pack();
    this.setLocationRelativeTo( c:null);
    validate();
    revalidate();
}
```

Figure 22:a function with the time complexity O(n)

```
            public void actionPerformed(ActionEvent e)
            {

                int result = JOptionPane.showConfirmDialog( parentComponent:null, message:"Are you sure you want to exit? All progress will be lost",  title:"Warning",  optionType:JOptionPane.
                if(result == JOptionPane.YES_OPTION){
                    MainMenu s = new MainMenu();
                    frame.dispose();
                    s.setVisible( b:true);
                }else if (result == JOptionPane.NO_OPTION){
                    return;
                }
            }
    });

    btnReset.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e)
            {
                JPanel panel = panelGame;
                pane.remove( comp:myTimer);
                myTimer = new StopWatchPanel();
                pane.add( comp:myTimer);
                if (Minesweeper.isDarkMode)
                    myTimer.setBackground( bg:black128);
                else
                    myTimer.setBackground( bg:grey192);
                if (!myTimer.timer.isRunning()) {
                    myTimer.lastTickTime = System.currentTimeMillis();
                }
                panelGame.removeAll();
                panelGame.revalidate();
                panelGame.repaint();
                createGamePanel();
                isTimerActive = true;
                flagCount =  Minesweeper.totalBombs;
                txtFlagCount.setText( text:Integer.toString( i:flagCount));
            }
    });
```

Figure 23: a function with a complexity of O(n)

```
public boolean isMultiple(int base, int multiple){   // O(1)
    if(multiple%base == 0)
        return true;
    return false;
}
```

Figure 15: a function with the complexity of O(1)

```
public void createNeighbors(){ //O(n)
    int columns = Minesweeper.getColumns(), rows = Minesweeper.getRows();
    for(Component c: panelGame.getComponents()){
        ArrayList<CustomButtonV2> neighbors = new ArrayList<>();
        if(c instanceof CustomButtonV2){
            CustomButtonV2 b = (CustomButtonV2) c;
            int currentRow = b.x, currentCol = b.y;
```

Figure 16: function with a complexity of O(n)

Conclusion

Overall complexity for the initialization of the game screen is O(n). For other functions such as button presses and checkWins, they are also O(n). So we can comfortably say our overall complexity is O(n) as well.

Softwares and Languages Used

- Netbeans
- Java
- Java Swing
- Java AWT
- Photopea to edit pictures

What Would We Implement If We Had More Time?

- We'd create a second game mode where instead of placing flags to avoid bombs, you have to click on all the bombs to win
- We'd work more on the GUI and complete finish Dark Mode
- Add scroll view to the game so that we can increase the row and column limits
- Optimize the code
- Make so that it's impossible for user to die on first click
- Make a game mode that is 0% luck using a Minesweeper solver algorithm

References

User, stackoverflow. (1969, December 5). *How do I dynamically resize a JFrame based on its components (a jpanel)?* Stack Overflow. https://stackoverflow.com/questions/74249372/how-do-i-dynamically-resize-a-jframe-based-on-its-components-a-jpanel

Depth first search or DFS for a graph. GeeksforGeeks. (2022, September 1).https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

Breadth first search or BFS for a graph. GeeksforGeeks. (2022, November 18).https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/