Introduction

MineSweeper, created in the Java programming language, is a game where the player reveals or clicks on cells in a grid while avoiding hidden bombs. The game includes different difficulty levels and a timer to challenge speed.

The program includes various features such as difficulty management, a timer, and score handling.

Reasons for Choosing the Game

I chose to develop CampoFiorito for several reasons:

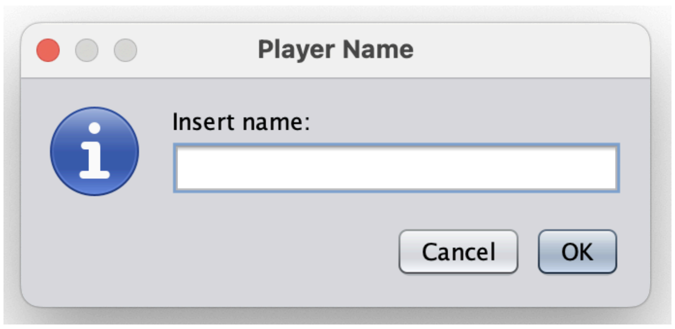
- **Complexity Management:** Implementing different difficulty levels helped me understand how to balance game complexity. I had to dynamically manage the number of bombs and winning conditions, improving my ability to write flexible and modular code.

- **Practical Application:** I saw this project as an opportunity to apply theoretical programming concepts in a concrete context. This allowed me to consolidate my knowledge and see how data structures, algorithms, and GUI management work in practice.

In **conclusion**, I chose to develop CampoFiorito for the complexity, to improve my programming skills, and to create a game that offers both fun and educational value.

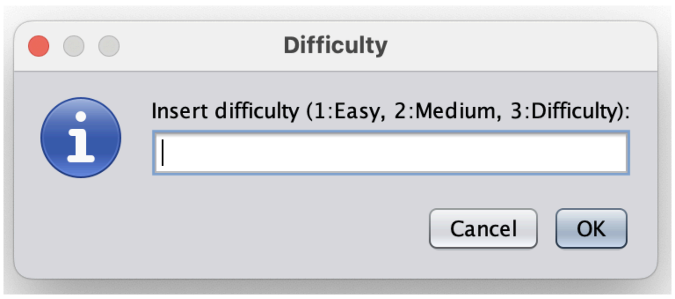
How to Play

1. Requirements – Install NetBeans, JDK (Java Developer Kit), and WinRAR.
2. Unzip the file named MineSweeper.zip.
3. Open NetBeans and click on "Open Project."
4. Select the project called Gioco and run it.
5. Enter your name and then choose the difficulty level



Player Name

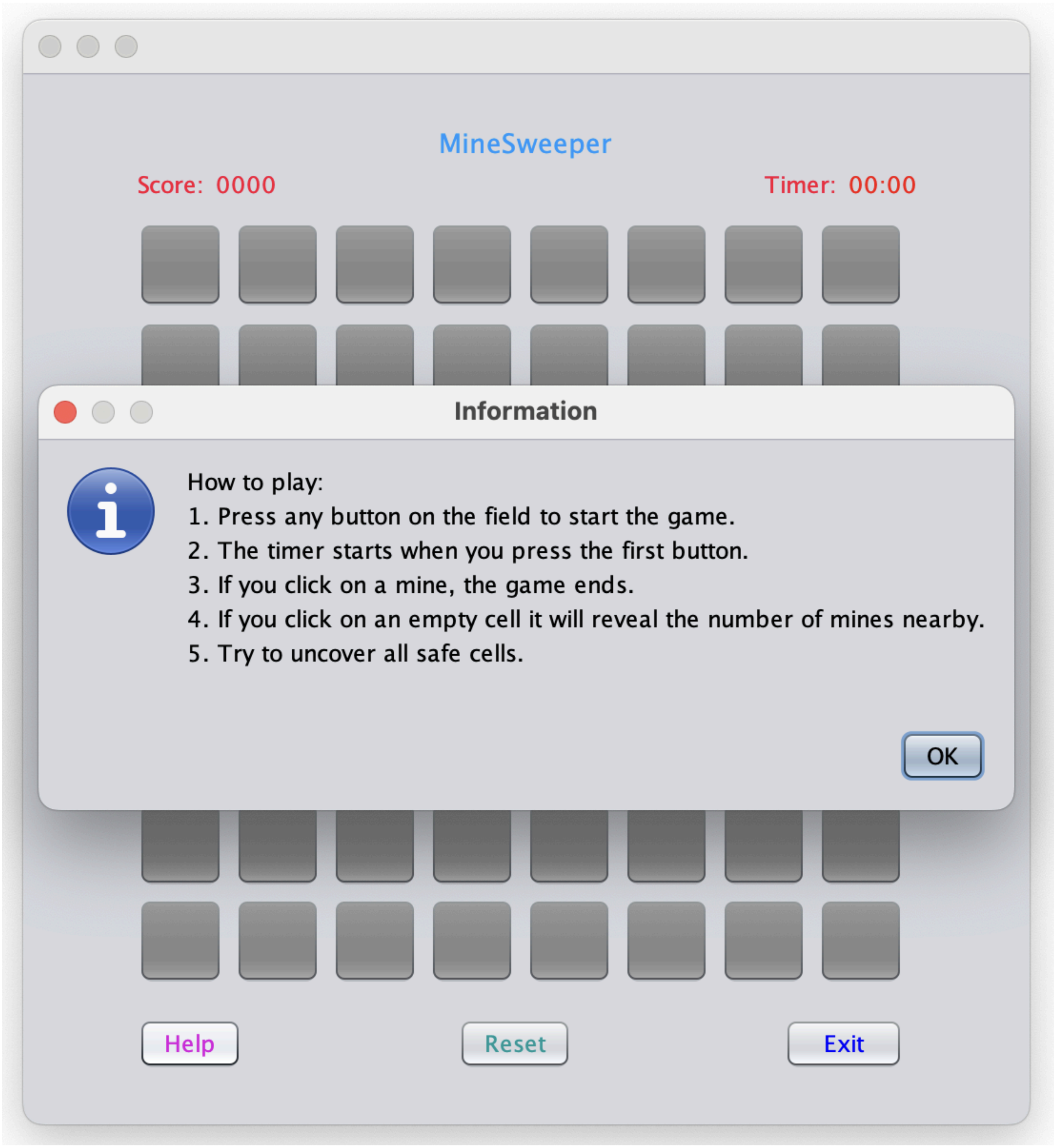
Insert name:



Difficulty

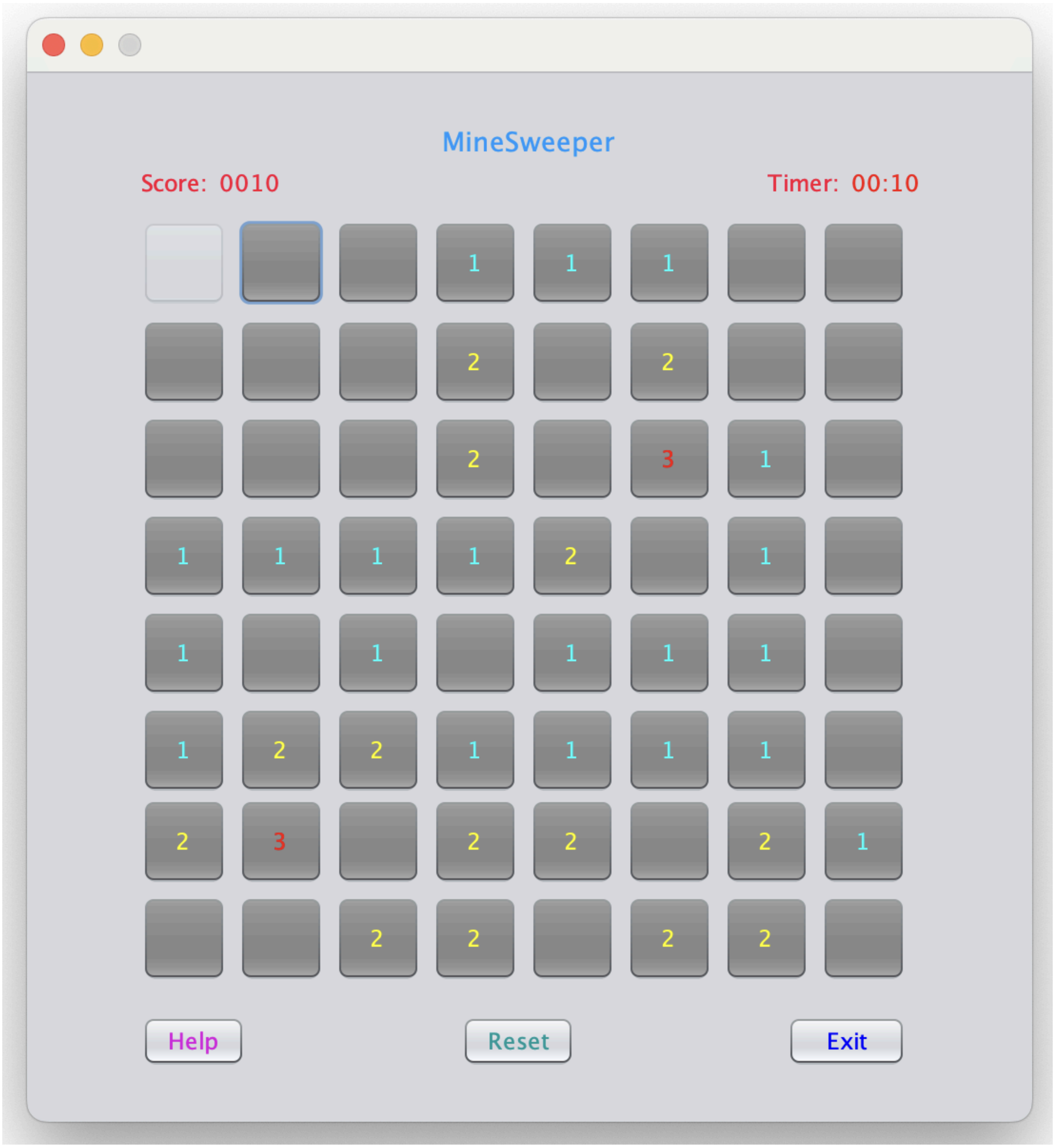
Insert difficulty (1:Easy, 2:Medium, 3:Difficulty):

6. Click the “Help” button if you want some instructions about the game.



How to play:
1. Press any button on the field to start the game.
2. The timer starts when you press the first button.
3. If you click on a mine, the game ends.
4. If you click on an empty cell it will reveal the number of mines nearby.
5. Try to uncover all safe cells.

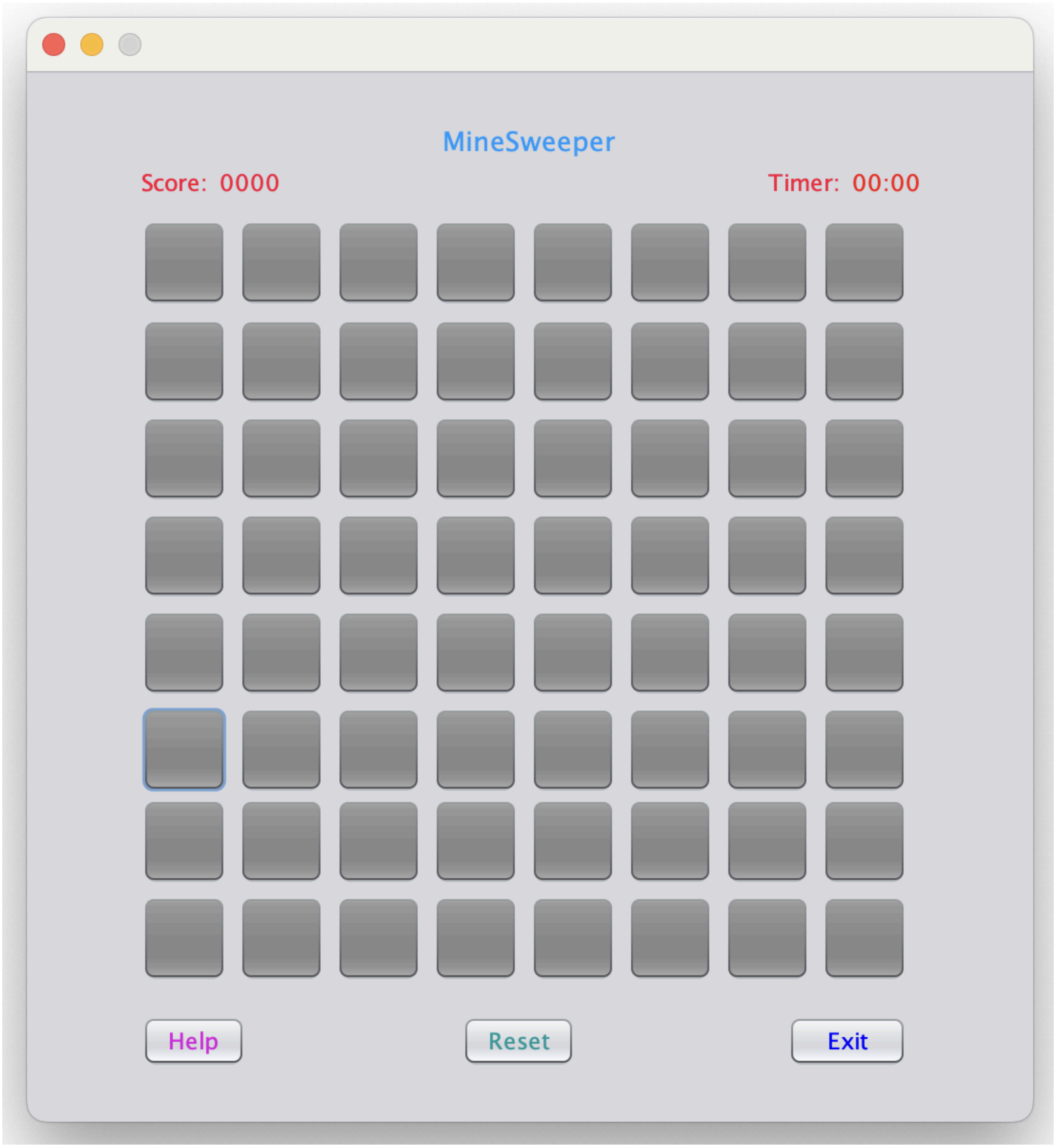
7. Click any button inside the grid to start the game.

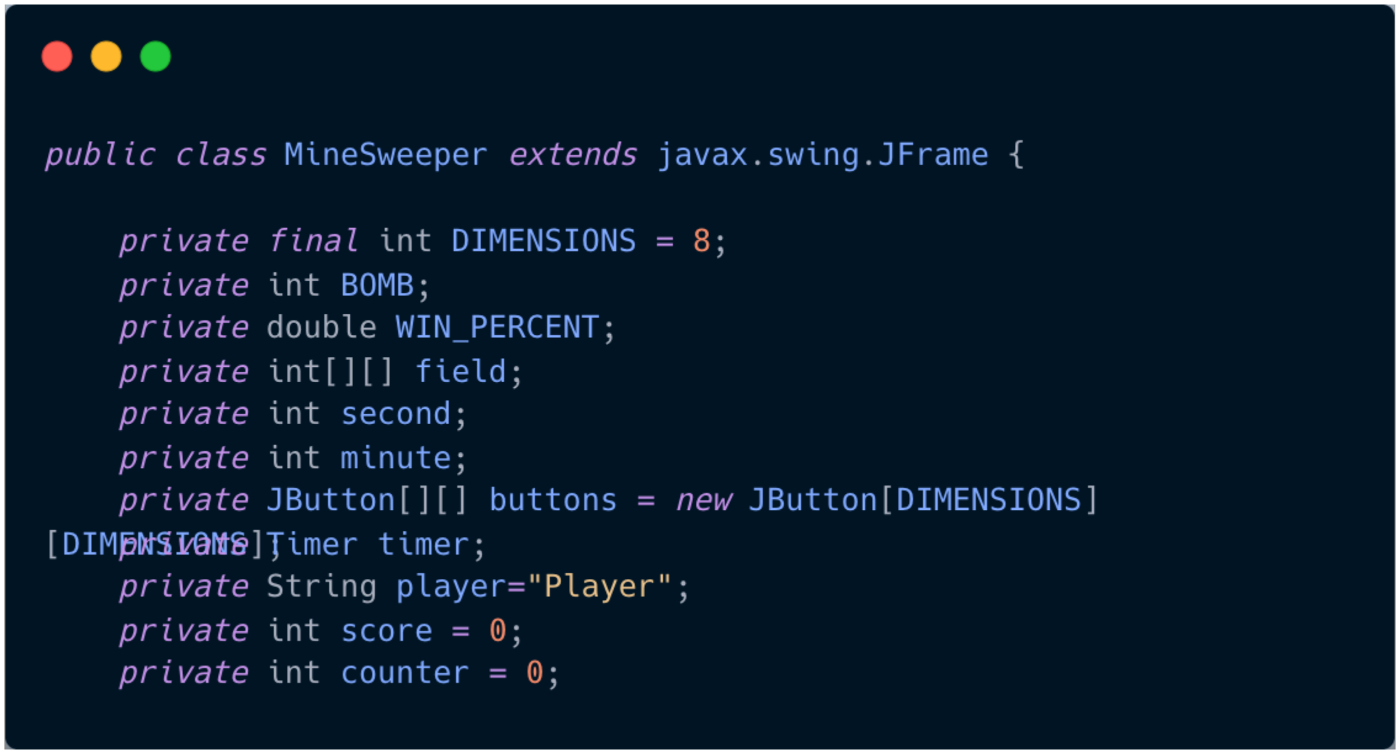

MineSweeper
Score: 0010
Timer: 00:10
1 1 1
2 2
2 3 1
1 1 1 1 2 1
1 1 1 1 1
1 2 2 1 1 1 1
2 3 2 2 2 1
2 2 2 2
Help
Reset
Exit

Game Configuration

Class: MineSweeper

This is the main class that configures and manages the game. It extends `javax.swing.JFrame` to create a graphical window.


MineSweeper
Score: 0000
Timer: 00:00
Help
Reset
Exit



```java
public class MineSweeper extends javax.swing.JFrame {

    private final int DIMENSIONS = 8;
    private int BOMB;
    private double WIN_PERCENT;
    private int[][] field;
    private int second;
    private int minute;
    private JButton[][] buttons = new JButton[DIMENSIONS]
[DIMENSIONS];
    private Timer timer;
    private String player="Player";
    private int score = 0;
    private int counter = 0;
```

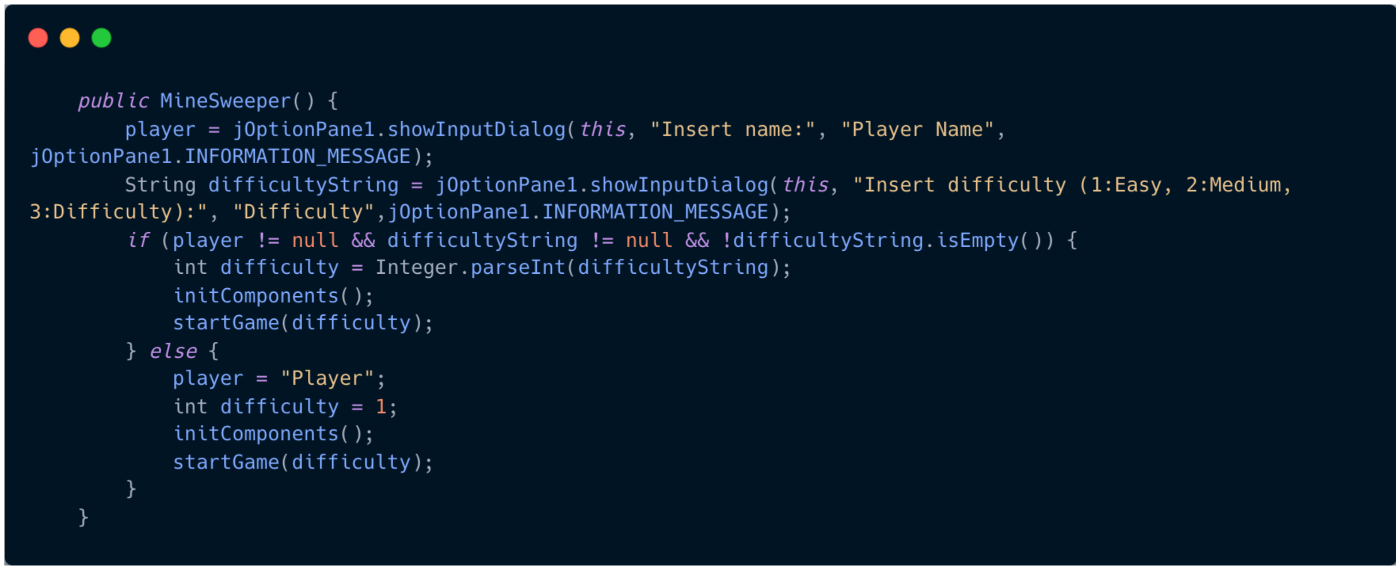
Description of Attributes

These are the main variables used in the game, including the field size, number of bombs, timer, and score:

- `DIMENSION`: Grid size (constant, set to 8).
- `BOMB`: Number of bombs in the grid, determined by the difficulty.
- `WIN_PERCENT`: Winning percentage, determined by the difficulty.
- `field`: 2D array representing the game field.
- `second`: Seconds counter for the game timer.
- `minute`: Minutes counter for the game timer.
- `buttons`: 2D array of `JButton` representing the grid buttons.
- `timer`: Game timer.
- `player`: Player's name.
- `score`: Player's score.
- `counter`: Counter for revealed cells.

Constructor

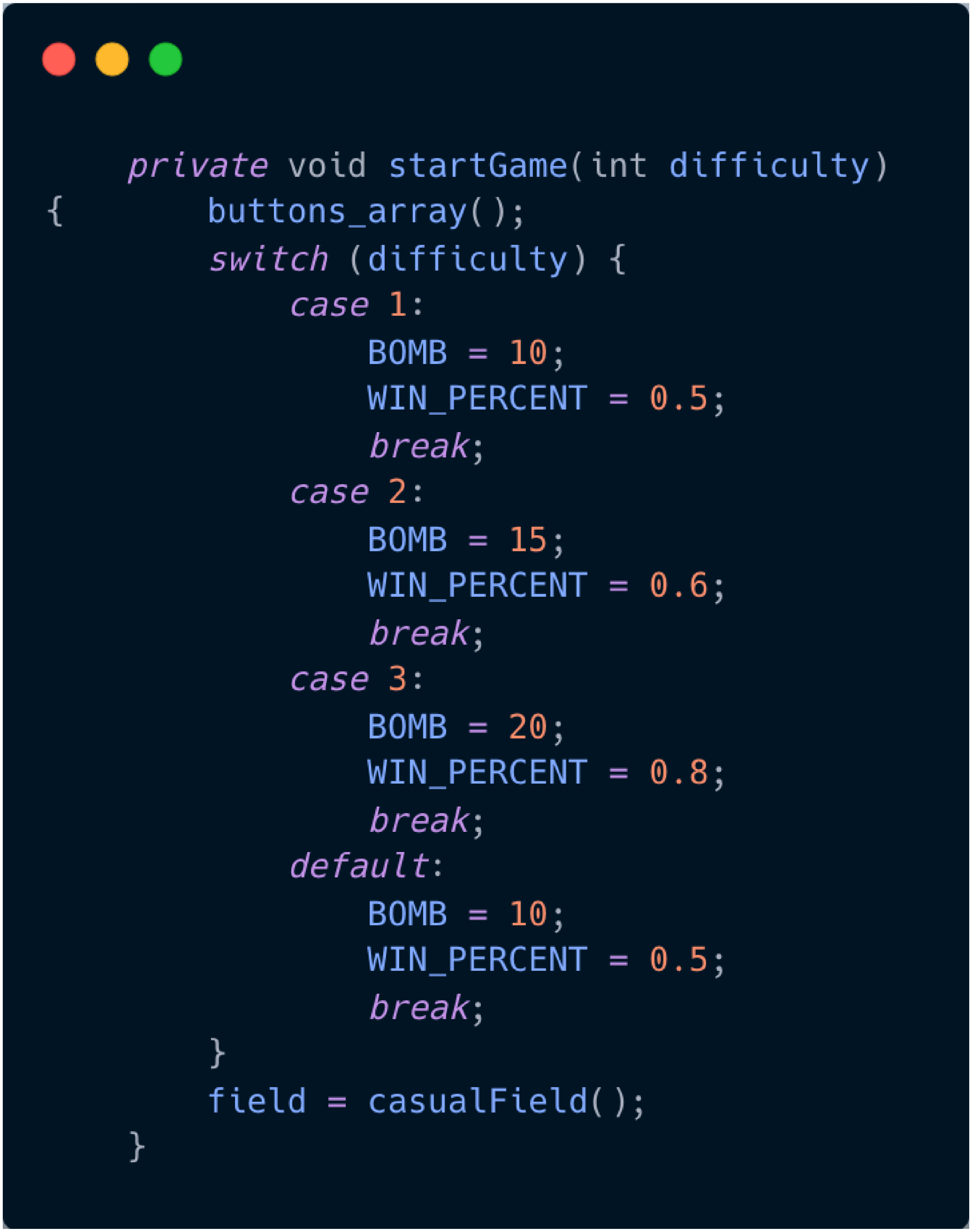
The constructor initializes the game by setting up the GUI, asking for the player's name and difficulty level, and starting the game.



```java
    public MineSweeper() {
        player = jOptionPane1.showInputDialog(this, "Insert name:", "Player Name",
jOptionPane1.INFORMATION_MESSAGE);
        String difficultyString = jOptionPane1.showInputDialog(this, "Insert difficulty (1:Easy, 2:Medium,
3:Difficulty):", "Difficulty",jOptionPane1.INFORMATION_MESSAGE);
        if (player != null && difficultyString != null && !difficultyString.isEmpty()) {
            int difficulty = Integer.parseInt(difficultyString);
            initComponents();
            startGame(difficulty);
        } else {
            player = "Player";
            int difficulty = 1;
            initComponents();
            startGame(difficulty);
        }
    }
```

Metodi

`startGame()`



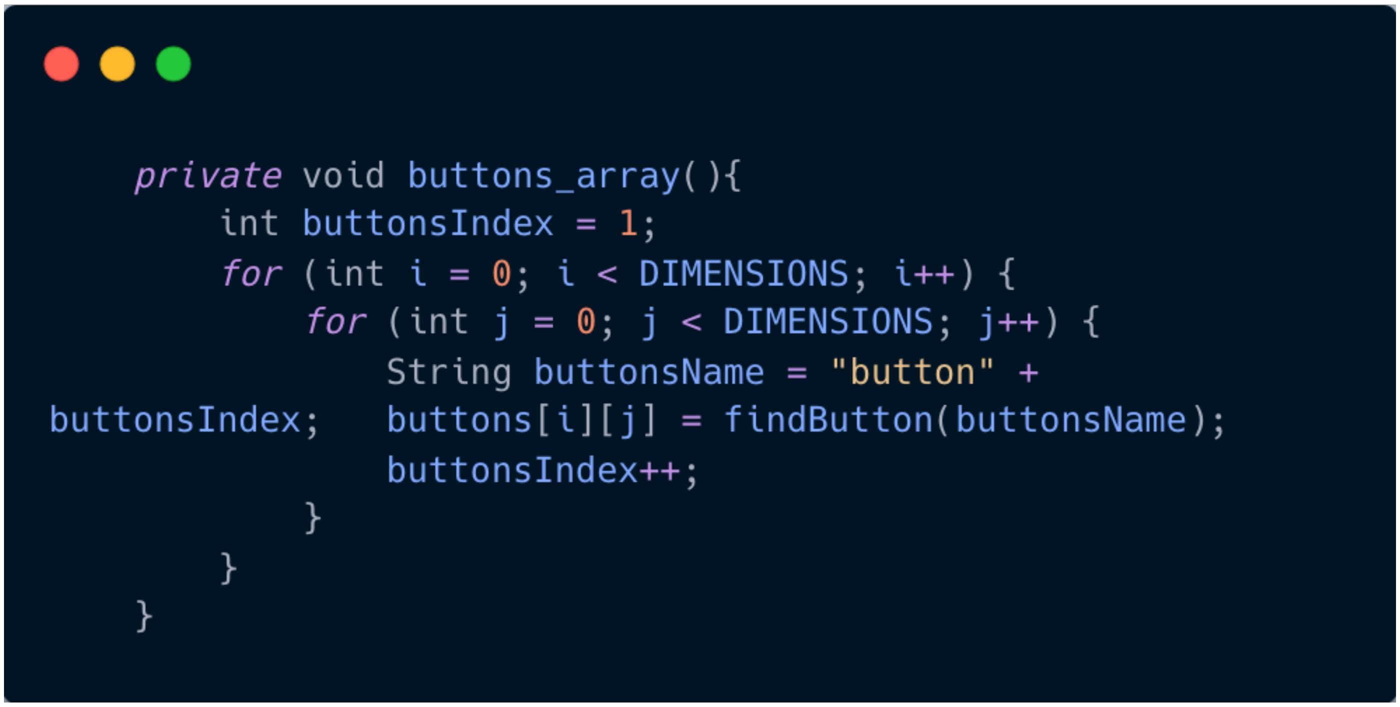
```
    private void startGame(int difficulty)
{       buttons_array();
        switch (difficulty) {
            case 1:
                BOMB = 10;
                WIN_PERCENT = 0.5;
                break;
            case 2:
                BOMB = 15;
                WIN_PERCENT = 0.6;
                break;
            case 3:
                BOMB = 20;
                WIN_PERCENT = 0.8;
                break;
            default:
                BOMB = 10;
                WIN_PERCENT = 0.5;
                break;
        }
        field = casualField();
    }
```

Sets the number of bombs and winning percentage based on the chosen difficulty, then generates the game field.

- **Parameters:**
  - `difficulty`: Level of difficulty(1: Easy, 2: Medium, 3: Difficult).

`buttons_array()`

Initializes the array of `buttons`, linking each grid button to the corresponding GUI component.

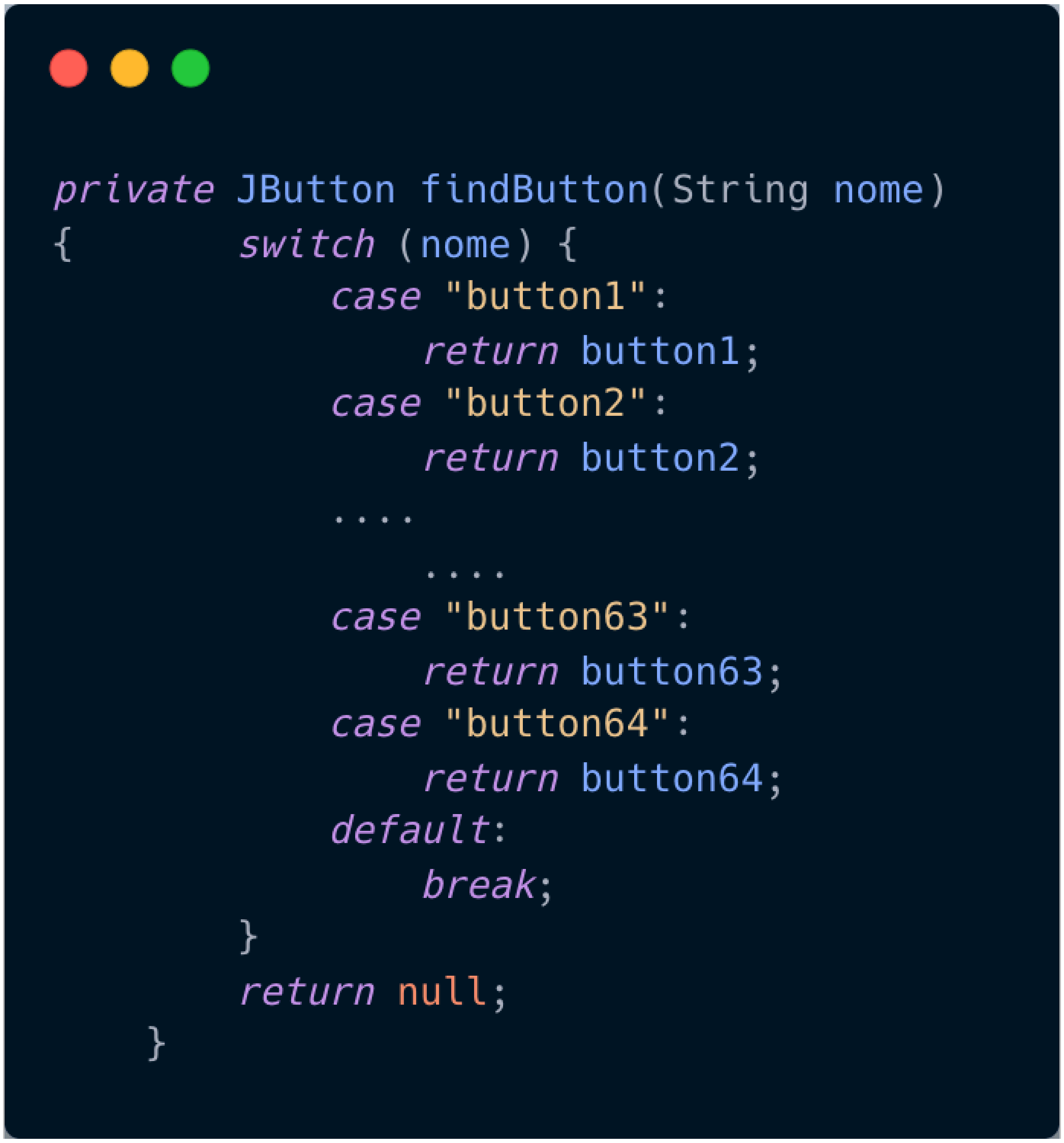


```
    private void buttons_array(){
        int buttonsIndex = 1;
        for (int i = 0; i < DIMENSIONS; i++) {
            for (int j = 0; j < DIMENSIONS; j++) {
                String buttonsName = "button" +
buttonsIndex;   buttons[i][j] = findButton(buttonsName);
                buttonsIndex++;
            }
        }
    }
```

`findButton()`

Returns the button corresponding to the name passed as a parameter.

- **Parameters:**
    - `name`: Name of button.
- **Returns:**
    - `JButton`: Corresponding button object.

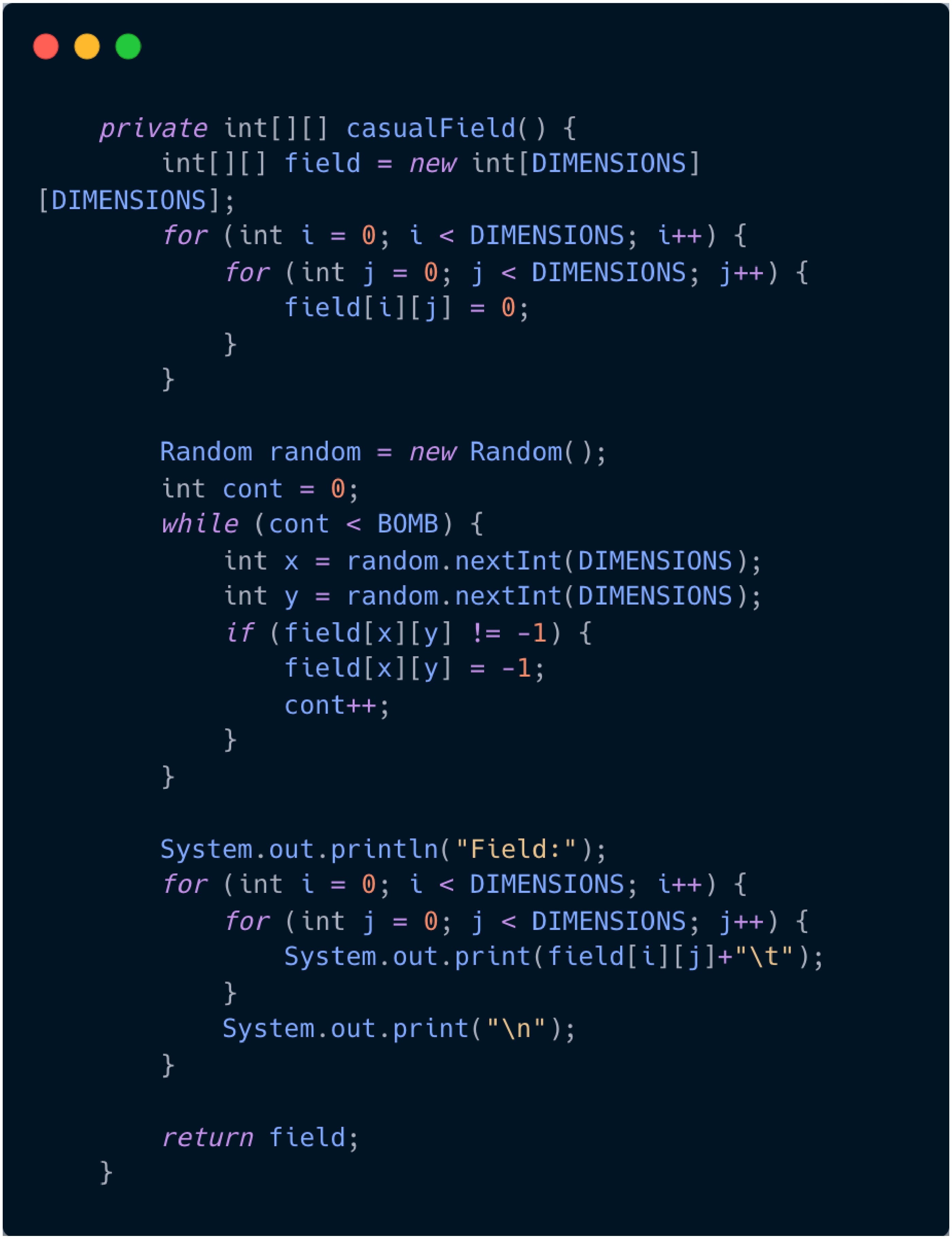


```
private JButton findButton(String nome)
{       switch (nome) {
            case "button1":
                return button1;
            case "button2":
                return button2;
            ....
                ....
            case "button63":
                return button63;
            case "button64":
                return button64;
            default:
                break;
        }
        return null;
    }
```

`casualField()`

Randomly generates the game field by placing bombs.

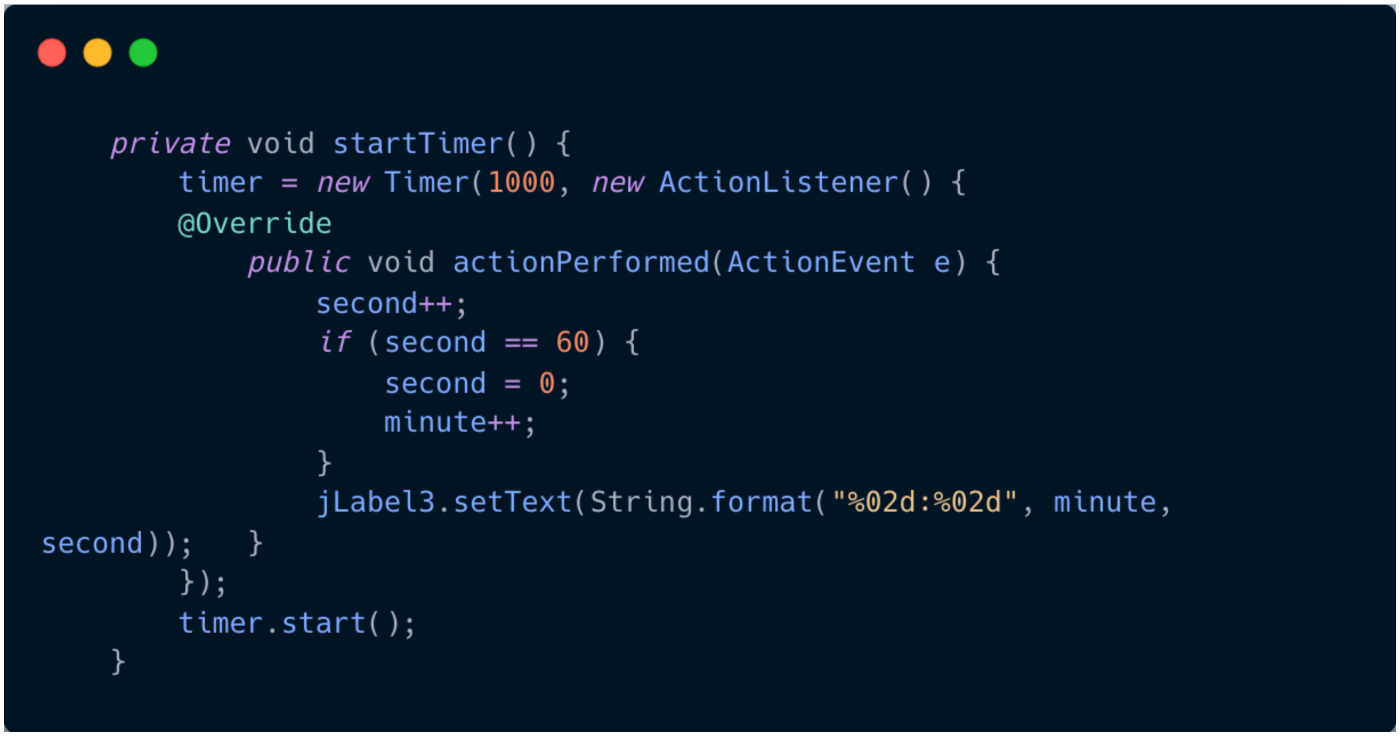
- **Returns:**
    - `int[][]`: Array 2D which represents the game field.



```
    private int[][] casualField() {
        int[][] field = new int[DIMENSIONS]
[DIMENSIONS];
        for (int i = 0; i < DIMENSIONS; i++) {
            for (int j = 0; j < DIMENSIONS; j++) {
                field[i][j] = 0;
            }
        }

        Random random = new Random();
        int cont = 0;
        while (cont < BOMB) {
            int x = random.nextInt(DIMENSIONS);
            int y = random.nextInt(DIMENSIONS);
            if (field[x][y] != -1) {
                field[x][y] = -1;
                cont++;
            }
        }

        System.out.println("Field:");
        for (int i = 0; i < DIMENSIONS; i++) {
            for (int j = 0; j < DIMENSIONS; j++) {
                System.out.print(field[i][j]+"\t");
            }
            System.out.print("\n");
        }

        return field;
    }
```

`startTimer()`

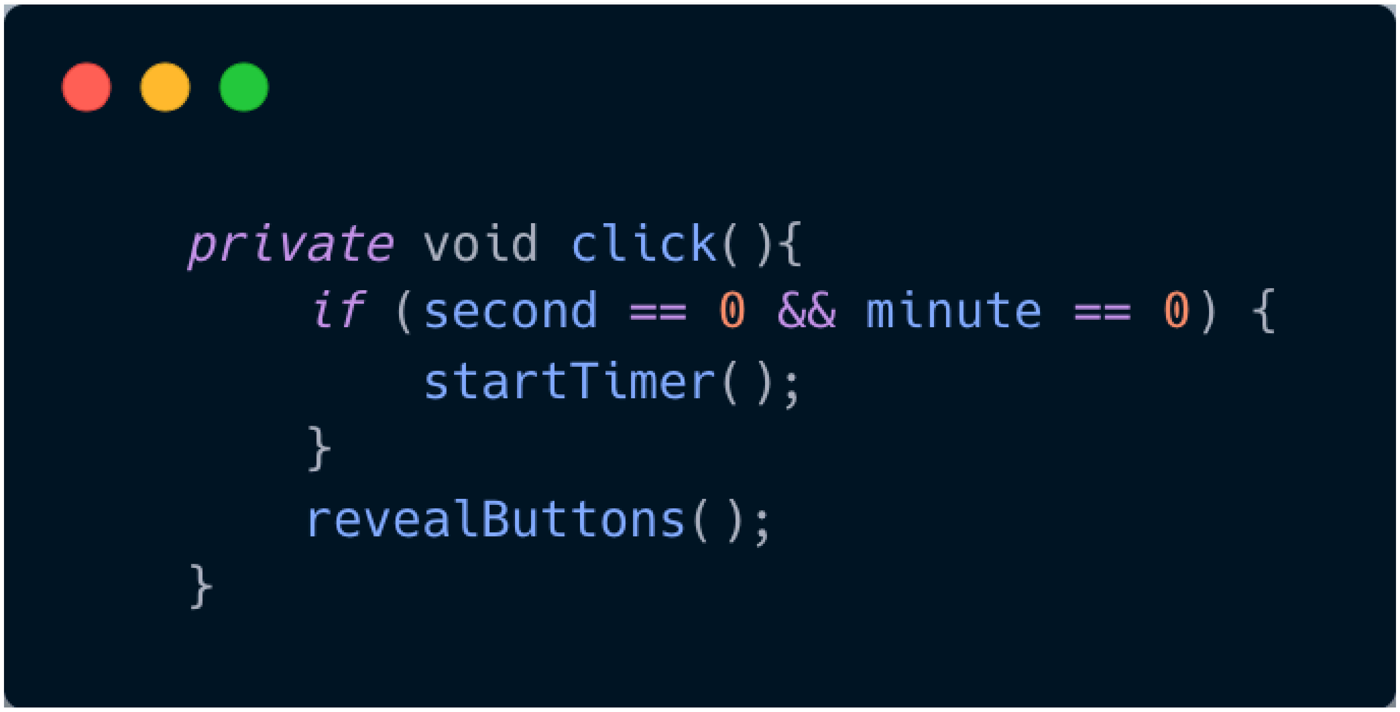
Starts a timer that updates elapsed seconds and minutes.



```java
    private void startTimer() {
        timer = new Timer(1000, new ActionListener() {
        @Override
            public void actionPerformed(ActionEvent e) {
                second++;
                if (second == 60) {
                    second = 0;
                    minute++;
                }
                jLabel3.setText(String.format("%02d:%02d", minute,
second));   }
        });
        timer.start();
    }
```

`click()`

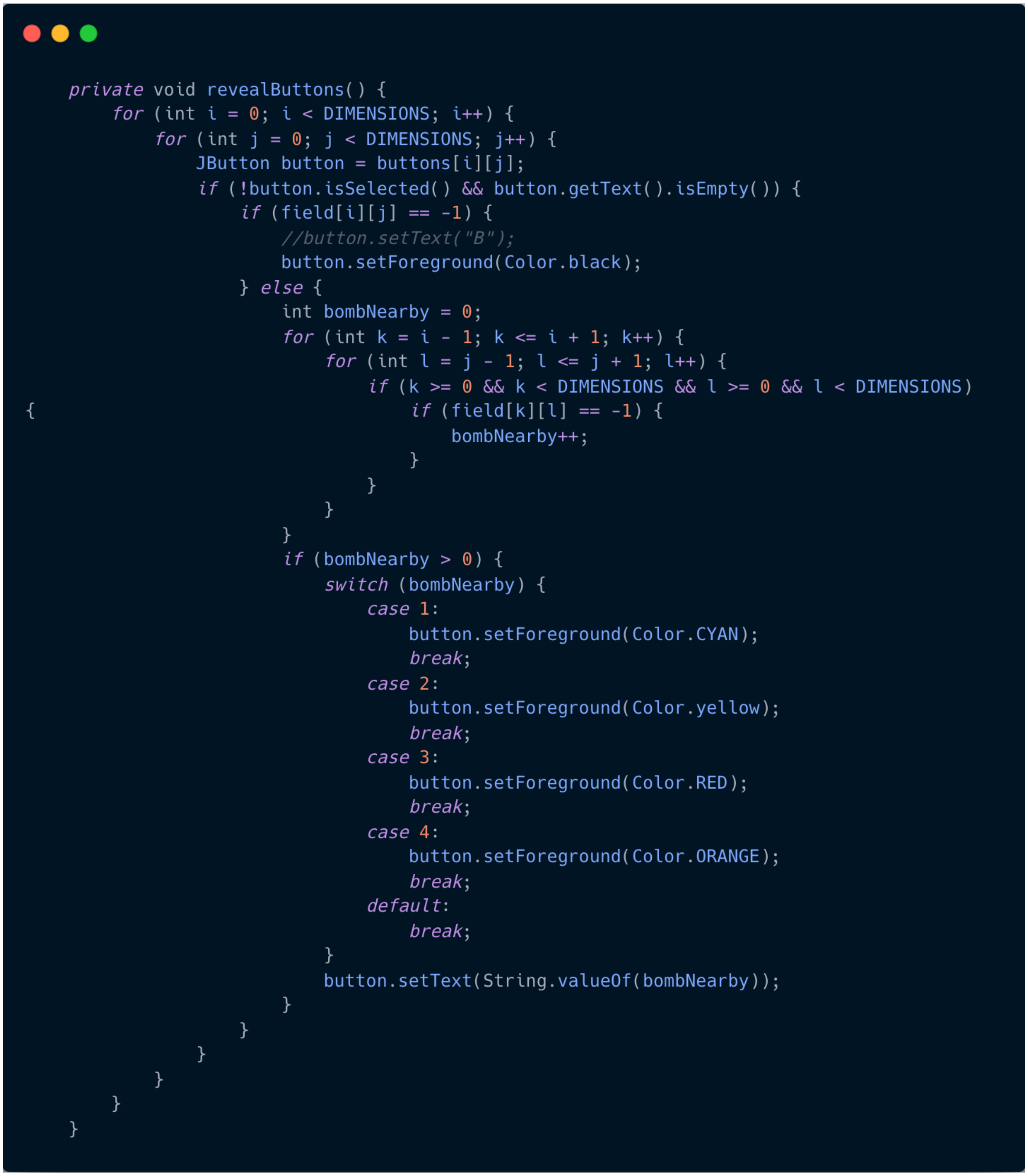
Handles a button click, starting the timer if it hasn't started yet, and revealing buttons.



```
private void click(){
    if (second == 0 && minute == 0) {
        startTimer();
    }
    revealButtons();
}
```

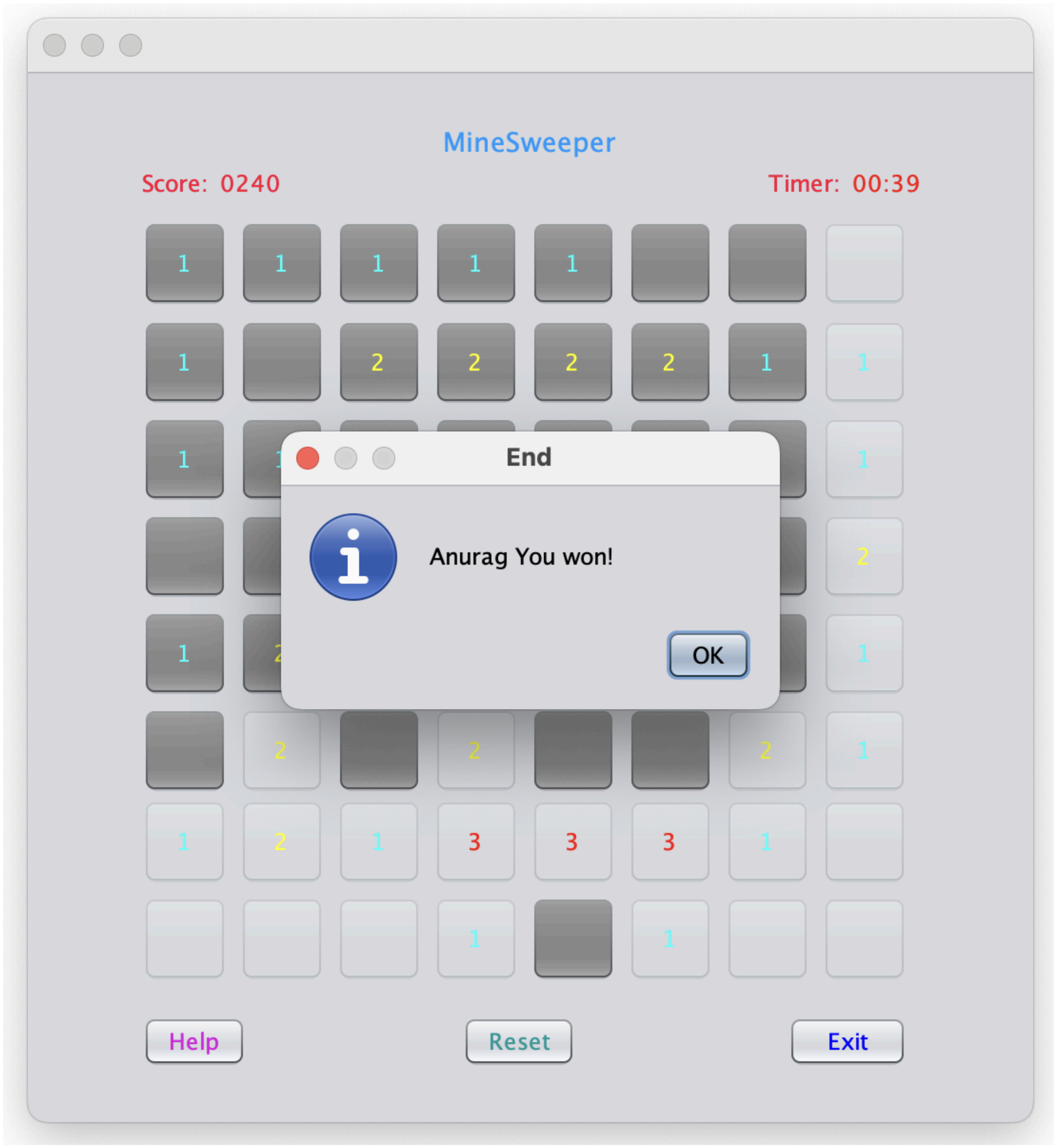
`revealButtons()`

Reveals button content, showing bombs or the number of nearby bombs



```java
    private void revealButtons() {
        for (int i = 0; i < DIMENSIONS; i++) {
            for (int j = 0; j < DIMENSIONS; j++) {
                JButton button = buttons[i][j];
                if (!button.isSelected() && button.getText().isEmpty()) {
                    if (field[i][j] == -1) {
                        //button.setText("B");
                        button.setForeground(Color.black);
                    } else {
                        int bombNearby = 0;
                        for (int k = i - 1; k <= i + 1; k++) {
                            for (int l = j - 1; l <= j + 1; l++) {
                                if (k >= 0 && k < DIMENSIONS && l >= 0 && l < DIMENSIONS)
{                                   if (field[k][l] == -1) {
                                        bombNearby++;
                                    }
                                }
                            }
                        }
                        if (bombNearby > 0) {
                            switch (bombNearby) {
                                case 1:
                                    button.setForeground(Color.CYAN);
                                    break;
                                case 2:
                                    button.setForeground(Color.yellow);
                                    break;
                                case 3:
                                    button.setForeground(Color.RED);
                                    break;
                                case 4:
                                    button.setForeground(Color.ORANGE);
                                    break;
                                default:
                                    break;
                            }
                            button.setText(String.valueOf(bombNearby));
                        }
                    }
                }
            }
        }
    }
```

```
controller(int row, int column)
```

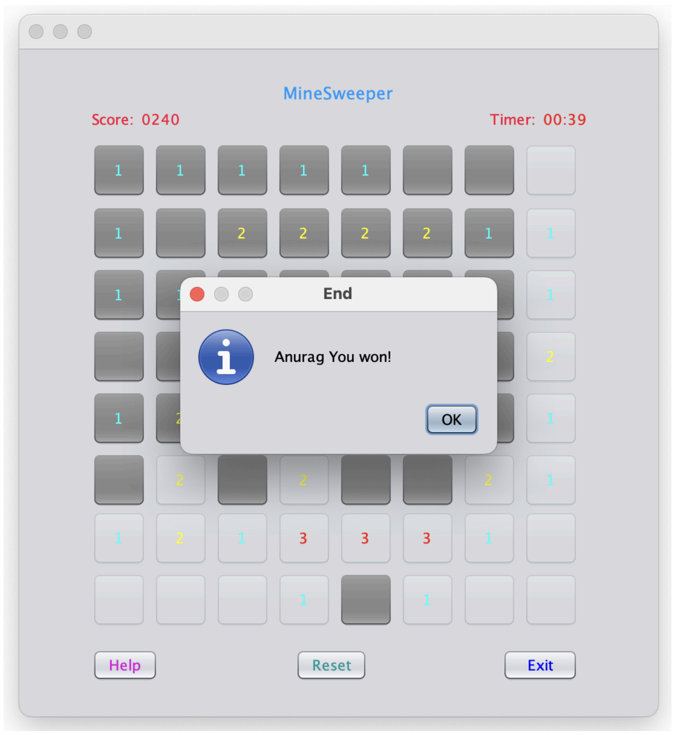


Score: 0240

Timer: 00:39

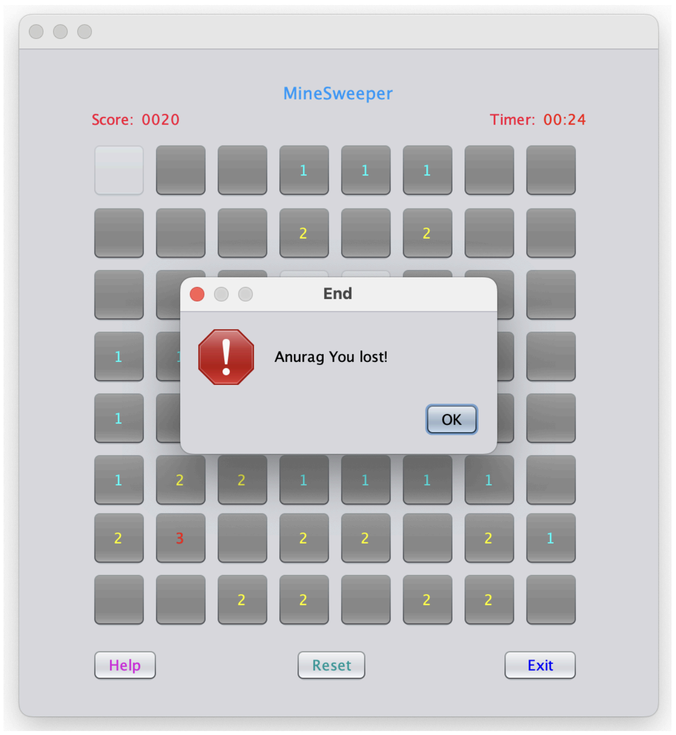
End

Anurag You won!



End

Anurag You won!

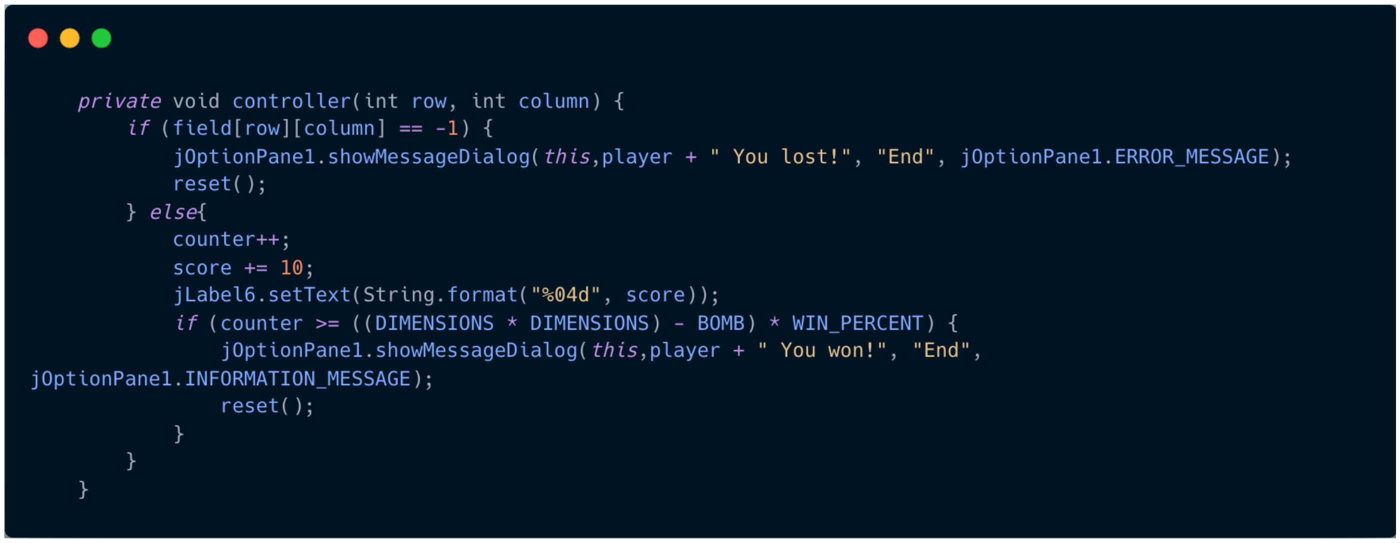


End

Anurag You lost!

Checks if the player clicked a bomb or has won the game, updating the score and showing end-game messages.

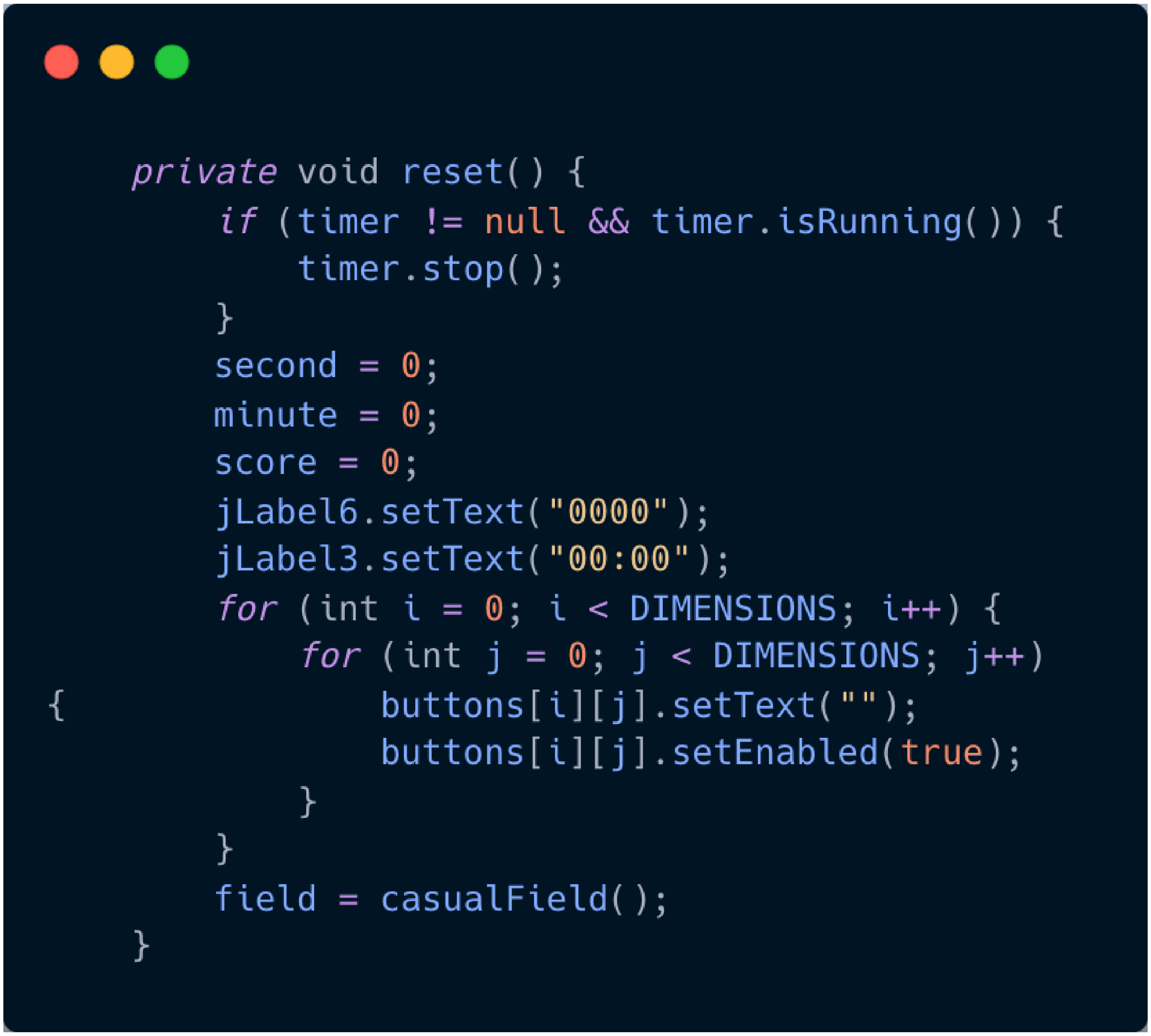
- **Parameters:**
  - `row`: Row of cell.
  - `column:` Column of cell.



```
    private void controller(int row, int column) {
        if (field[row][column] == -1) {
            jOptionPane1.showMessageDialog(this,player + " You lost!", "End", jOptionPane1.ERROR_MESSAGE);
            reset();
        } else{
            counter++;
            score += 10;
            jLabel6.setText(String.format("%04d", score));
            if (counter >= ((DIMENSIONS * DIMENSIONS) - BOMB) * WIN_PERCENT) {
                jOptionPane1.showMessageDialog(this,player + " You won!", "End",
jOptionPane1.INFORMATION_MESSAGE);
                reset();
            }
        }
    }
```

`reset():`

Resets the game, stopping the timer and resetting the field and scores.



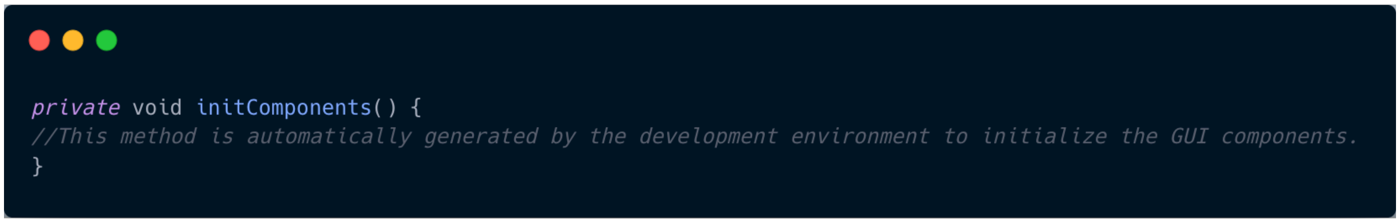
```
    private void reset() {
        if (timer != null && timer.isRunning()) {
            timer.stop();
        }
        second = 0;
        minute = 0;
        score = 0;
        jLabel6.setText("0000");
        jLabel3.setText("00:00");
        for (int i = 0; i < DIMENSIONS; i++) {
            for (int j = 0; j < DIMENSIONS; j++)
{                buttons[i][j].setText("");
                buttons[i][j].setEnabled(true);
            }
        }
        field = casualField();
    }
```

GUI Components

The user interface is managed using Swing components:

- `javax.swing.JOptionPane`: For dialog boxes.
- `javax.swing.JLabel`: To display the timer and score.
- `javax.swing.JButton`: For grid cells and control buttons (help, exit, reset).

GUI Initialization



```java
private void initComponents() {
//This method is automatically generated by the development environment to initialize the GUI components.
}
```

Game Logic

Bomb Placement:

Bombs are randomly placed on the grid using the Random class. The number of bombs depends on the selected difficulty

Cell Revealing:

When a cell is clicked, it reveals the number of adjacent bombs or a bomb if present. The text color changes based on the number of adjacent bombs.

Win/Loss Conditions:

- **Lost**: If a bomb is clicked.
- **Win:** If the player reveals enough non-bomb cells as determined by the winning percentage.

Timer:

The timer increments every second, updating the display. It starts when the first cell is clicked.

User Interaction

Starting a Game:

- The player is asked to enter their name and choose a difficulty level.
- The grid is initialized, and the game starts with the first click.

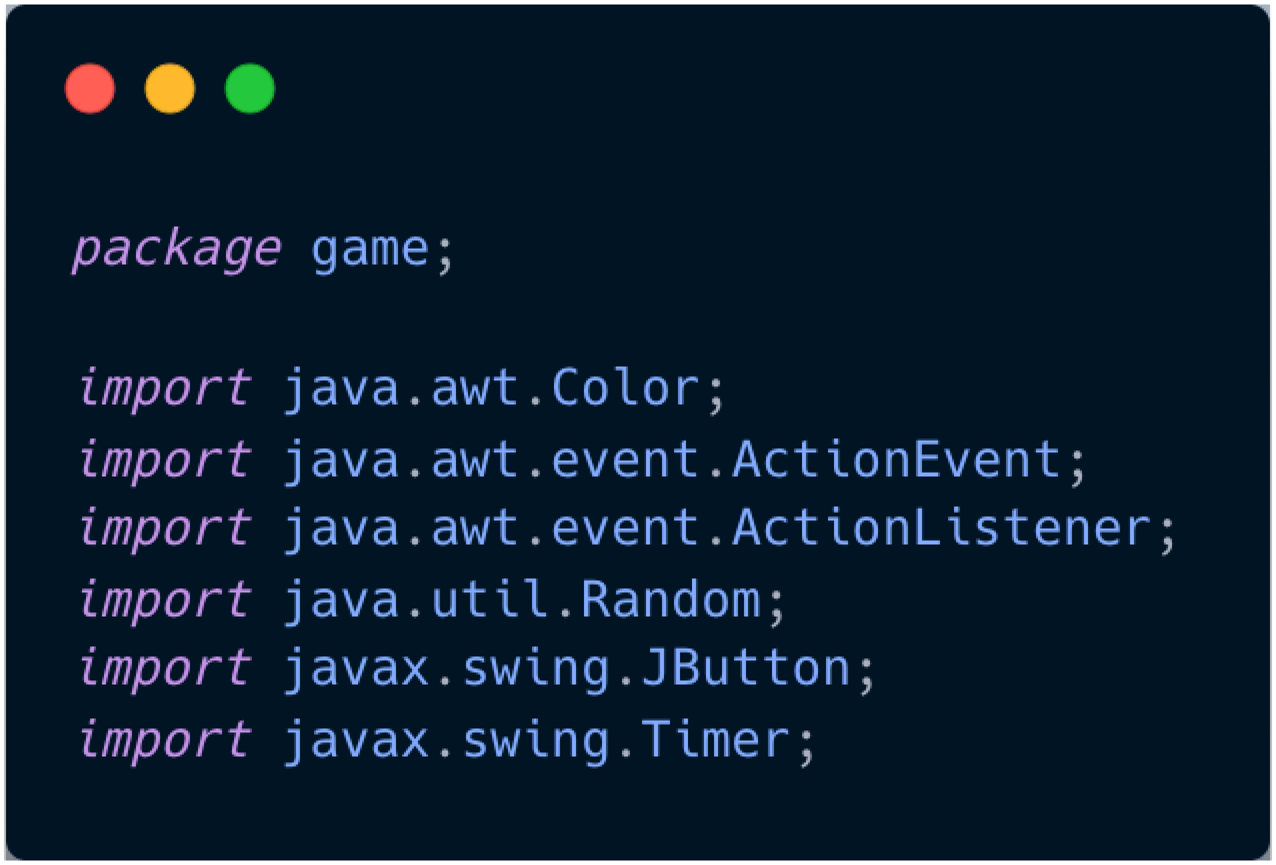
In-Game Actions:

- Clicking a cell reveals its content.
- The “Help” button shows game instructions.
- The “Exit” button closes the game.
- The “Reset” button restarts the game.

Code Structure

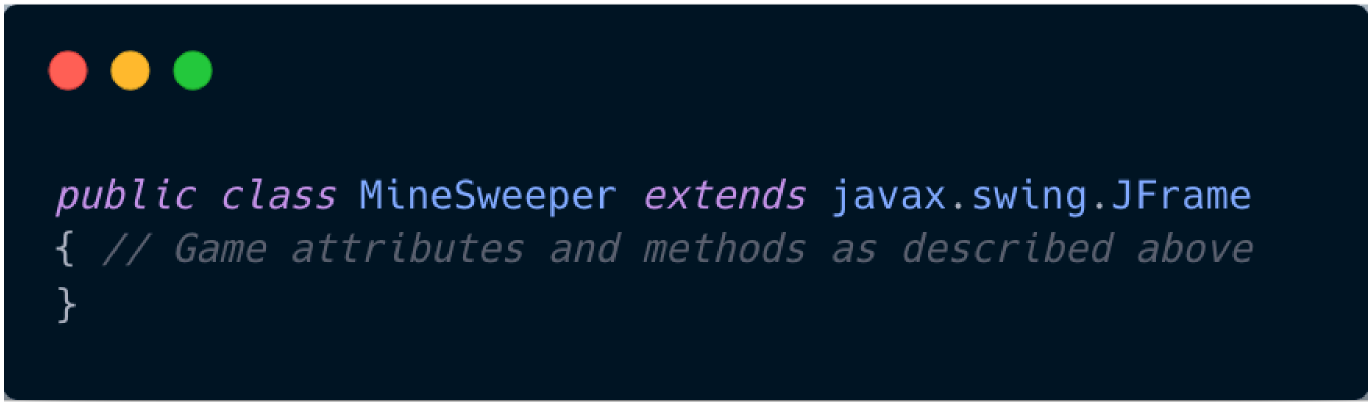
Package and Imports:

Contains all game logic and GUI handling.



```java
package game;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;
import javax.swing.JButton;
import javax.swing.Timer;
```

Main Class



```java
public class MineSweeper extends javax.swing.JFrame
{ // Game attributes and methods as described above
}
```

Conclusion

This documentation provides information about the game **MineSweeper**, its setup, logic, and user interaction. The game uses a grid of buttons to simulate a minefield, with logic for managing difficulty, scoring, and the timer. It describes the key components and methods used in the game, offering a comprehensive guide to reading and understanding the code.