2048 GAME USING JAVA

by

DIVAKAR K G - 71382202032

PROJECT BASED LEARNING

report submitted in partial fulfilment of

the requirements for the

Degree of Bachelor of Engineering in

Computer Science and Engineering

SECOND YEAR

Sri Ramakrishna Institute of Technology

Coimbatore – 641010

DECEMBER 2023

APPROVAL AND DECLARATION

This project report titled "**2048 Game using JAVA**" was prepared and submitted by **Divakar K G (71382202032)** and has been found satisfactory in terms of scope, quality and presentation as partial fulfillment of the requirement for the **Bachelor of Engineering (Computer Science and Engineering)** in Sri Ramakrishna Institute of Technology, Coimbatore (SRIT).

Checked and Approved by

Dr. S. GAYATHRI DEVI

Associate Professor

Project Supervisor

Department of Computer Science and Engineering

Sri Ramakrishna Institute of Technology

Coimbatore -10

December 2023

ABSTRACT

The JAVA 2048 game project is a software implementation of the popular puzzle game 2048 developed using the JAVA programming language. The game is designed to provide an interactive and engaging user experience, challenging players to combine numbered tiles strategically to reach the elusive 2048 tile.

The 2048 Game Project aims to implement the popular puzzle game "2048" using JAVA programming language. The game revolves around merging numbered tiles on a grid to reach the elusive 2048 tile. The objective is to strategically shift tiles across the board, combining identical numbers to form higher-valued tiles, all while tactically managing limited space.

This project focuses on utilizing JAVA's object-oriented principles to create an interactive and engaging gaming experience. It involves implementing a graphical user interface (GUI) using Java's Swing or JavaFX libraries to render the game grid, display tiles, and facilitate user interactions. Key features include a responsive interface, smooth animation for tile movements, and an intuitive control scheme allowing users to navigate the grid seamlessly.

The game will incorporate essential functionalities such as score tracking, win/loss detection, and adaptive difficulty levels to enhance the overall gaming experience. The underlying algorithms will manage tile movements, merging logic, and game state updates efficiently.

Additionally, the project will emphasize code modularity, readability, and documentation to ensure ease of understanding, maintenance, and potential future enhancements. Testing methodologies will be employed to validate the game's functionality, ensuring a bug-free and robust application.

TABLE OF CONTENTS

LIST OF FIGURES

.

LIST OF ABBREVIATIONS

GUI	Graphical User Interface
JavaFX	Java's special EFF-ECTS
JDK	Java Development Kit
UI	User Interface

CHAPTER 1

INTRODUCTION

The 2048 game project is a strategy-based, single-player puzzle game that involves sliding numbered tiles on a grid to reach a tile with the number 2048.

Key Features of the Java 2048 Game Project:

- **Grid-based Gameplay:** The game is played on a 4x4 grid where tiles with numbers are randomly placed. The player can move these tiles in four directions - up, down, left, and right - combining tiles with the same numbers when they collide.
- **Tile Movement and Combination:** The player can move tiles by pressing arrow keys or buttons. When two tiles with the same number collide due to a move, they merge into a single tile with the sum of their values.
- **Scoring System:** The game keeps track of the player's score, increasing every time two tiles are merged. The aim is to reach the 2048 tile with the highest possible score.
- **Game Over Conditions:** The game ends when the grid is full, and no more moves are possible, or when the player reaches the 2048 tile. At this point, the player can choose to restart the game and try to achieve a higher score.
- **User Interface:** The game will have a graphical user interface (GUI) displaying the grid, tiles with numbers, score, and game status. It will provide a user-friendly experience for interacting with the game.
- **Logic Implementation:** The project involves implementing the game's logic, including tile movement, merging, checking for game-over conditions, and handling user inputs.
- **Modularity and Extensibility:** The code will be designed to be modular, allowing for easy modification and potential future enhancements, such as adding features, improving the UI, or implementing additional functionalities.

Overall, the 2048 game project aims to provide an engaging and interactive implementation of the 2048 puzzle game using JAVA , offering players a challenging and enjoyable gaming experience.

CHAPTER 2

REQUIREMENTS SPECIFICATION

2.1 Software Requirements

Operating System	:	Windows 7 & above
Simulator Tool	:	Notepad or Notepad++
Programming Package	:	Java JDK-21

2.2 Hardware Requirements

Processor	:	Any Intel or AMD x 86-64 processor
RAM	:	Minimum 4 GB and Recommended 8 GB
Hard Disk	:	24 GB for all product installation and an HDD is strongly required
Input Device	:	Standard Keyboard and Mouse
Output Device	:	VGA and High-Resolution Monitor

CHAPTER 3

METHODOLOGY

3.1 Flow Chart of the Proposed System

For Determining Next Move:

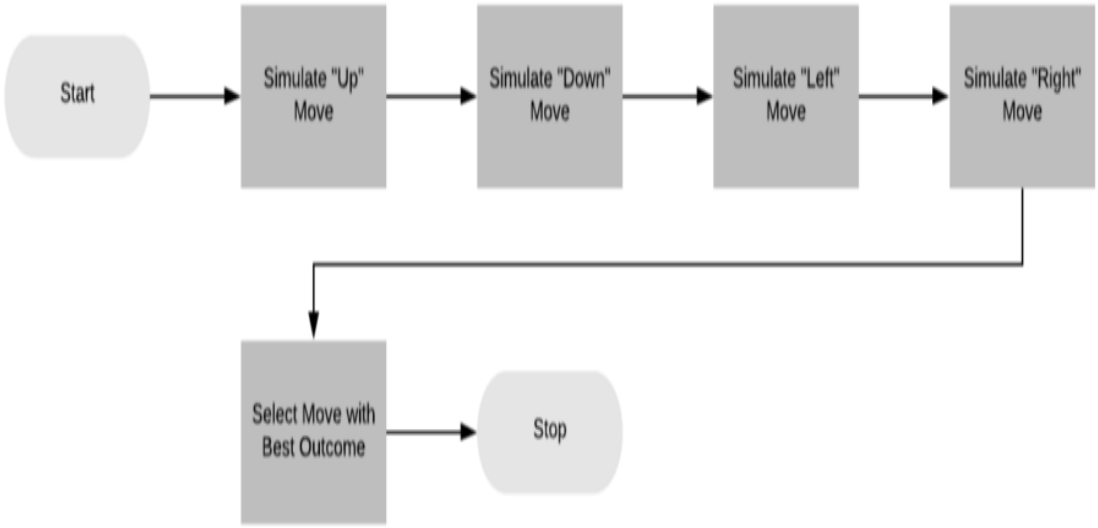

Fig 3.11 Flow Chart of Determining Next Move

The figure appears to be a flowchart specifically related to the 2048 game. The content seems to include instructions for simulating movements such as "Up," "Down," "Left," and "Right," as well as commands for selecting moves and stopping. The overall layout and content suggest a visual representation of the decision flow within the game.

For Generating Score:

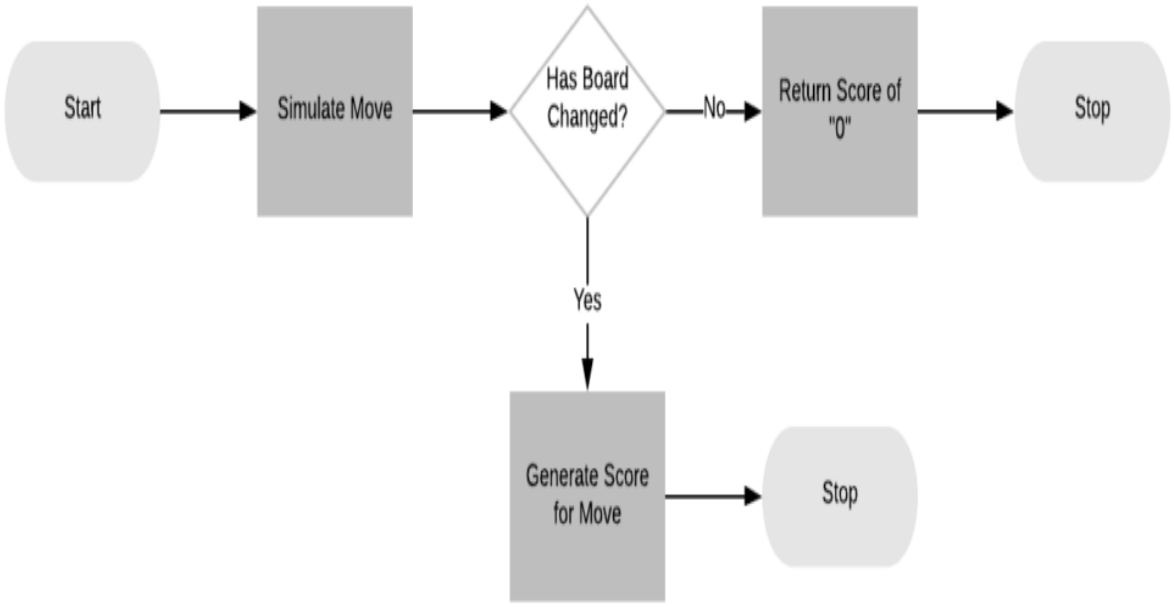

Fig 3.12 Flow Chart of Generating Score

The figure appears to be a flowchart specifically related to the 2048 game. It contains instructions related to scoring and simulating moves in a game and explains the process for generating a score.

3.2 Algorithm of the Proposed System

These are the steps to build a 2048 game in JAVA:

- Game structure and GUI setup
- Initializing the game
- Adding new tiles
- Grid Manipulation
- Updating the GUI
- Game over and Restart
- Update the score
- Main() method
- Test the game

1. Game Structure and GUI Setup

Import Statements: The import statements at the beginning of the code import necessary classes and packages from the Java AWT, Swing, and util libraries.

Class definition and declaring variables: Define a class called "Game2048" and declare all necessary variables and GUI components needed to manage the game state and display the game on the screen.

Constructor: The class constructor initializes the game state, sets up the GUI components, and starts the game. The GUI components are set up, including the frame, grid panel, and information panel. Key listeners are added to handle user input.

2. Initializing the game

The initializeGrid() method is responsible for initializing the game grid. This method iterates through each cell of the grid and initializes it to 0. It then calls the method addNewNumber() twice to add two initial numbers to the grid.

3. Adding new tiles

The 'addNewNumber()' method is responsible for adding a new tile (either 2 or 4) to a random empty cell in the grid. It uses the random number generator to select a random row and column and checks if the selected cell is empty. If the cell is empty, a new tile is added.

4. Grid Manipulation

The game allows the player to move the tiles in four directions: up, down, left, and right. The 'moveup()', 'moveDown()', 'moveLeft()', and 'moveRight()' methods handle the logic for moving the tiles in the respective directions. These methods use nested loops to iterate over the grid and update the positions of the tiles accordingly. If any tiles are merged during the movement, the score is updated.

5. Updating the GUI:

The updateGridLabels() method is used to update the visual representation of the grid in the GUI. It iterates over the grid and updates the JLabel components accordingly. The

background colour and text of each label are set based on the value of the corresponding grid cell. The getTileColor() method maps each tile value to a specific color.

6. Game over and Restart

The 'isGameOver()' method checks if the game is over by examining the grid or by checking whether the win condition is reached. If no empty cells are available, and there are no adjacent cells with the same value, the game is considered over.

The 'showGameOverMessage()' method displays a message dialog to the player, indicating whether they have won or lost the game. The player can choose to play again or exit the game.

The 'restartGame()' method resets the game state, including the score, win condition, grid, and GUI. It then initializes the game again.

7. Update the score

The 'updateScore()' method updates the score and high score labels in the GUI.

8. Main() method

The 'main()' method is the entry point of the program. It creates an instance of the Game2048 class, which starts the game by invoking the constructor.

9. Test the game

Verify that the game functions correctly by playing through various scenarios, including wins and losses.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Output of the Proposed System:

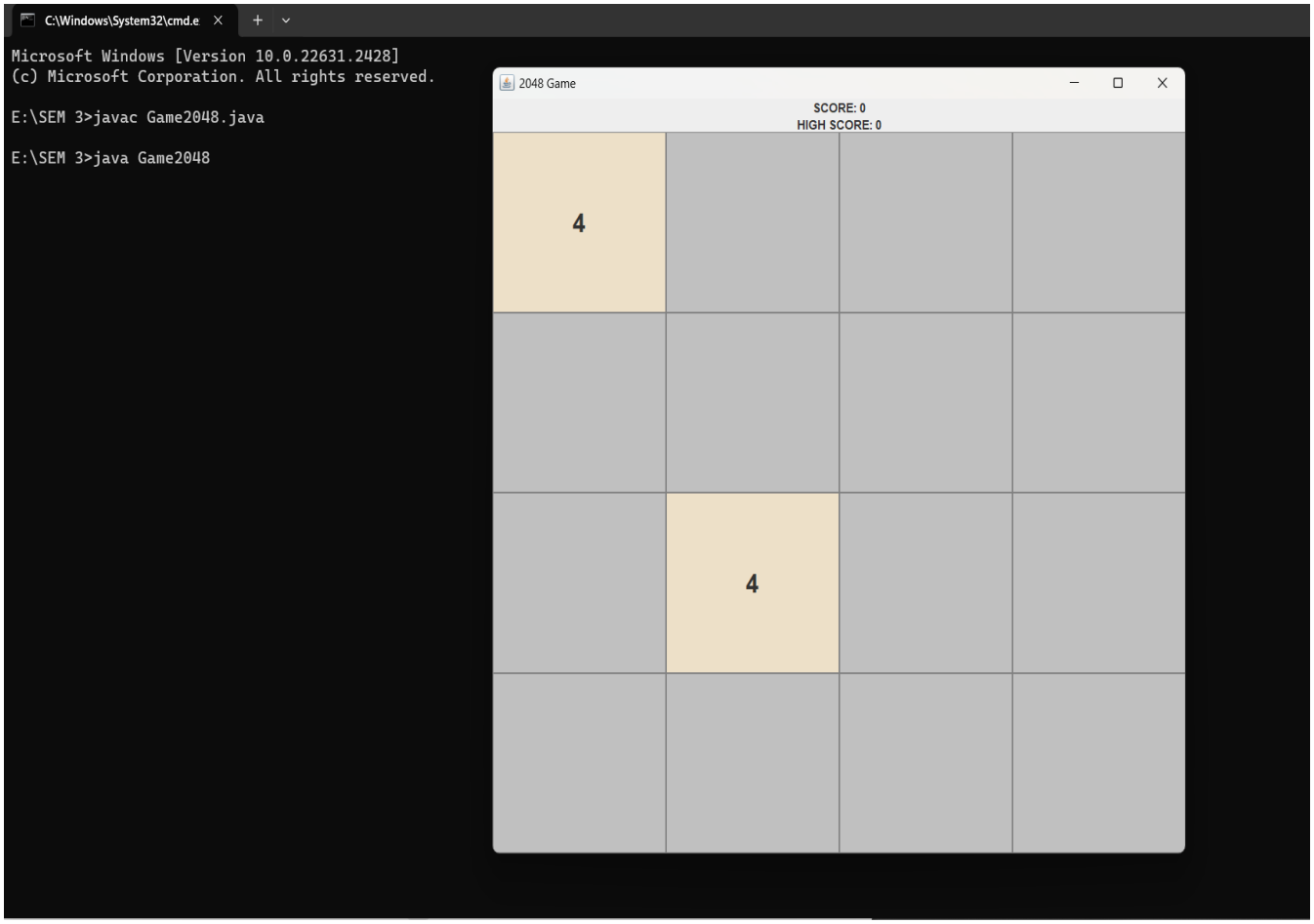

Fig 4.1 Output with executing command

The Figure 4.1 describes content seems to be a screenshot of a command line interface showing the execution of a Java file called "Game2048.java". The screenshot includes a command to compile the file with "javac", and then running the game with "java Game2048". It also displays the game's User Interface.

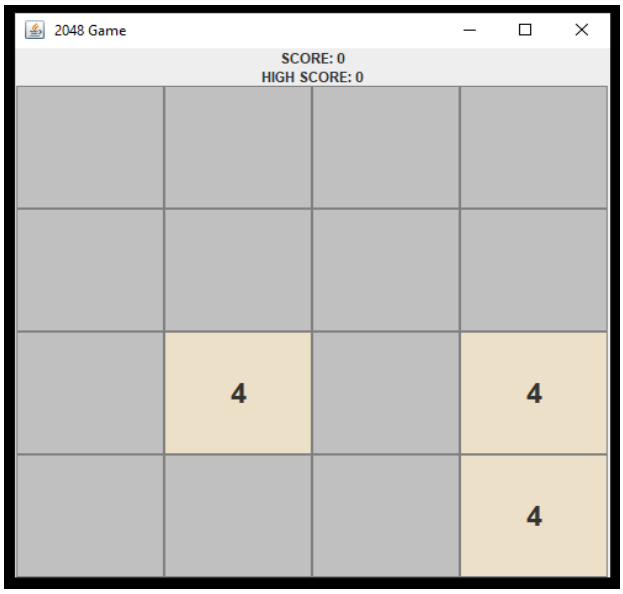

Fig 4.2 Output of starting condition of game

The Fig 4.2 describes is a screenshot of the starting condition of a 2048 game. The score and high score are both 0. The game grid consists of 4x4 squares.

Fig 4.3 Output of Game Over condition

The Fig 4.3 is a screenshot of a graphical user interface for a game called "2048." It shows the game board with numbers like 4, 2, 8, and 16, and displays the player's score and high score. The game is over, and it prompts the player to choose whether to play again.

CHAPTER 5

CONCLUSION

5.1 Summary

In essence, the 2048 game java project stands as a testament to the culmination of technical skills, problem-solving abilities, user-centric design, and a platform for continuous learning and growth in the realm of Java-based game development. This project serves not only as an accomplishment but also as a stepping stone for further explorations and innovations in game programming and software development.

REFERENCES

1. Herbert Schildt, “JAVA: The Complete Reference”, McGraw Hill Professional, 11th Edition, 2018.
2. Cay S. Hortmann, “Core Java Volume – I Fundamentals”, Pearson Education, 11th Edition, 2018.
3. Deitel and Deitel, “Java How to Program”, Pearson Education India, 10th Edition 2016.
4. https://www.baeldung.com/2048-java-solver
5. https://www.javatpoint.com/java-swing
6. https://copyprogramming.com/howto/simplified-version-of-2048-game
7. https://chat.openai.com/

APPENDIX

SOURCE CODE

/**The above program to demonstrate the 2048 math game it is full of 4x4 matrix addition of numbers 2,4,8,16,32,64,128,512,1024,2048. when 2048 arrived user won the game but it is not simple to play this as you think if you not play well all tiles are filled with number the game will terminated.*/

//Now, develop 2048 game using java

```
import java.awt.*;
import java.awt.event.*;
import java.util.Random;
import javax.swing.*;
public class Game2048 {
    // Size of the game grid
    private static final int SIZE = 4;
    //Instance variables
    private int[][] grid;
    private Random random;
    private int score;
    private int highScore;
    private JFrame frame;
    private JPanel gridPanel;
    private JLabel[][] gridLabels;
    private JLabel scoreLabel;
    private JLabel highScoreLabel;
    private boolean winConditionReached;
    public Game2048() {
        // Constructor code
        grid = new int[SIZE][SIZE];
        random = new Random();
        score = 0;
        highScore = 0;
        // Create the GUI components
        frame = new JFrame(" 2048 Game");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(750, 750);
        frame.setLayout(new BorderLayout());
        gridPanel = new JPanel(new GridLayout(SIZE, SIZE));
        gridLabels = new JLabel[SIZE][SIZE];
        for (int i = 0; i < SIZE; i++) {
```

```
        for (int j = 0; j < SIZE; j++) {
            gridLabels[i][j] = new JLabel("", JLabel.CENTER);
            gridLabels[i][j].setFont(new Font("SANS_SERIF", Font.BOLD, 24));
            gridLabels[i][j].setOpaque(true);
            gridLabels[i][j].setBackground(Color.LIGHT_GRAY);
            gridPanel.add(gridLabels[i][j]);
        }
    }
    // Add components to the frame
    frame.add(gridPanel, BorderLayout.CENTER);
    JPanel infoPanel = new JPanel(new GridLayout(2, 2));
    scoreLabel = new JLabel("SCORE: 0", JLabel.CENTER);
    highScoreLabel = new JLabel("HIGH SCORE: 0", JLabel.CENTER);
    infoPanel.add(scoreLabel);
    infoPanel.add(highScoreLabel);
    frame.add(infoPanel, BorderLayout.NORTH);
    frame.addKeyListener(new KeyAdapter() {
        // Handle key events
        public void keyPressed(KeyEvent e) {
            int keyCode = e.getKeyCode();
            if (keyCode == KeyEvent.VK_UP) {
                moveUp();
            } else if (keyCode == KeyEvent.VK_DOWN) {
                moveDown();
            } else if (keyCode == KeyEvent.VK_LEFT) {
                moveLeft();
            } else if (keyCode == KeyEvent.VK_RIGHT) {
                moveRight();
            }
            updateGridLabels();
            updateScore();
            if (isGameOver()) {
                showGameOverMessage();
            }
        }
    });
    // Set frame properties and initialize the grid
    frame.setFocusable(true);
    frame.requestFocus();
    frame.setVisible(true);
    initializeGrid();
    updateGridLabels();
}
public void initializeGrid() {
    // Grid initialization code
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
```

```
                grid[i][j] = 0;
            }
        }
        // Add two new numbers to random positions
        addNewNumber();
        addNewNumber();
    }
    public void addNewNumber() {
        // New number generation code
        int row, col;
        do {
            // Generate random row and column indices
            row = random.nextInt(SIZE);
            col = random.nextInt(SIZE);
        } while (grid[row][col] != 0);
        // Assign a new number (2 or 4) to the empty cell
        grid[row][col] = (random.nextInt(2) + 1) * 2;
    }
    public void updateGridLabels() {
        // Update the GUI grid labels with the current grid state
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                if (grid[i][j] == 0) {
                    gridLabels[i][j].setText("");
                    gridLabels[i][j].setBackground(Color.LIGHT_GRAY);
                } else if (grid[i][j] == 2048) {
                    winConditionReached = true;
                    gridLabels[i][j].setText(String.valueOf(grid[i][j]));
                    gridLabels[i][j].setBackground(getTileColor(grid[i][j]));
                } else {
                    // Set the text of each label to the corresponding grid cell value
                    gridLabels[i][j].setText(String.valueOf(grid[i][j]));
                    gridLabels[i][j].setBackground(getTileColor(grid[i][j]));
                }
                // Add grid lines
                gridLabels[i][j].setBorder(BorderFactory.createLineBorder(Color.GRAY));
            }
        }
    }
    public Color getTileColor(int value) {
        //Maps each tile value to a specific color
        switch (value) {
            case 2:
                return new Color(238, 228, 218);
            case 4:
                return new Color(237, 224, 200);
            case 8:
```

```
            return new Color(242, 177, 121);
        case 16:
            return new Color(245, 149, 99);
        case 32:
            return new Color(246, 124, 95);
        case 64:
            return new Color(246, 94, 59);
        case 128:
            return new Color(237, 207, 114);
        case 256:
            return new Color(237, 204, 97);
        case 512:
            return new Color(237, 200, 80);
        case 1024:
            return new Color(237, 197, 63);
        case 2048:
            return new Color(237, 194, 46);
        default:
            return Color.WHITE;
    }
  }
  public void moveUp() {
    //Moves and merges the tiles upwards
    int prevScore = score;
    boolean moved = false;
    for (int j = 0; j < SIZE; j++) {
        int mergeValue = -1;
        for (int i = 1; i < SIZE; i++) {
            if (grid[i][j] != 0) {
                int row = i;
                while (row > 0 && (grid[row - 1][j] == 0 || grid[row - 1][j] == grid[row][j]))
{
                    if (grid[row - 1][j] == grid[row][j] && mergeValue != row - 1) {
                        grid[row - 1][j] *= 2;
                        score += grid[row - 1][j];
                        grid[row][j] = 0;
                        mergeValue = row - 1;
                        moved = true;
                    } else if (grid[row - 1][j] == 0) {
                        grid[row - 1][j] = grid[row][j];
                        grid[row][j] = 0;
                        moved = true;
                    }
                    row--;
                }
            }
        }
```

```
        }
        if (moved) {
            addNewNumber();
            updateScore();
        }
    }
    public void moveDown() {
        //Moves and merges the tiles downwards
        int prevScore = score;
        boolean moved = false;
        for (int j = 0; j < SIZE; j++) {
            int mergeValue = -1;
            for (int i = SIZE - 2; i >= 0; i--) {
                if (grid[i][j] != 0) {
                    int row = i;
                    while (row < SIZE - 1 && (grid[row + 1][j] == 0 || grid[row + 1][j] ==
grid[row][j]))                {
                        if (grid[row + 1][j] == grid[row][j] && mergeValue != row + 1) {
                            grid[row + 1][j] *= 2;
                            score += grid[row + 1][j];
                            grid[row][j] = 0;
                            mergeValue = row + 1;
                            moved = true;
                        } else if (grid[row + 1][j] == 0) {
                            grid[row + 1][j] = grid[row][j];
                            grid[row][j] = 0;
                            moved = true;
                        }
                        row++;
                    }
                }
            }
        }
        if (moved) {
            addNewNumber();
            updateScore();
        }
    }
    public void moveLeft() {
        //Moves and merges the tiles towards left
        int prevScore = score;
        boolean moved = false;
        for (int i = 0; i < SIZE; i++) {
            int mergeValue = -1;
            for (int j = 1; j < SIZE; j++) {
                if (grid[i][j] != 0) {
                    int col = j;
```

```
            while (col > 0 && (grid[i][col - 1] == 0 || grid[i][col - 1] == grid[i][col])) {
              if (grid[i][col - 1] == grid[i][col] && mergeValue != col - 1) {
                grid[i][col - 1] *= 2;
                score += grid[i][col - 1];
                grid[i][col] = 0;
                mergeValue = col - 1;
                moved = true;
              } else if (grid[i][col - 1] == 0) {
                grid[i][col - 1] = grid[i][col];
                grid[i][col] = 0;
                moved = true;
              }
              col--;
            }
          }
        }
      }
      if (moved) {
        addNewNumber();
        updateScore();
      }
    }
    public void moveRight() {
      //Moves and merges the tiles towards right
      int prevScore = score;
      boolean moved = false;
      for (int i = 0; i < SIZE; i++) {
        int mergeValue = -1;
        for (int j = SIZE - 2; j >= 0; j--) {
          if (grid[i][j] != 0) {
            int col = j;
            while (col < SIZE - 1 && (grid[i][col + 1] == 0 || grid[i][col + 1] ==
grid[i][col])) {
              if (grid[i][col + 1] == grid[i][col] && mergeValue != col + 1) {
                grid[i][col + 1] *= 2;
                score += grid[i][col + 1];
                grid[i][col] = 0;
                mergeValue = col + 1;
                moved = true;
              } else if (grid[i][col + 1] == 0) {
                grid[i][col + 1] = grid[i][col];
                grid[i][col] = 0;
                moved = true;
              }
              col++;
            }
          }
```

```
            }
        }
        if (moved) {
            addNewNumber();
            updateScore();
        }
    }
    public boolean isGameOver() {
        //Game-over condition
        if(winConditionReached){
            return true;
        }
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                if (grid[i][j] == 0 ||
                        (i > 0 && grid[i][j] == grid[i - 1][j]) ||
                        (i < SIZE - 1 && grid[i][j] == grid[i + 1][j]) ||
                        (j > 0 && grid[i][j] == grid[i][j - 1]) ||
                        (j < SIZE - 1 && grid[i][j] == grid[i][j + 1])) {
                    return false;
                }
            }
        }
        return true;
    }
    public void showGameOverMessage() {
        //Displays game-over message
        String message;
        if (winConditionReached) {
            message = "Congratulations! You reached the 2048 tile!\nDo you want to
continue playing?";
        } else {
            message = "Game over! Do you want to play again?";
        }
        int choice = JOptionPane.showConfirmDialog(frame, message, "Game Over",
JOptionPane.YES_NO_OPTION);
        if (choice == JOptionPane.YES_OPTION) {
            restartGame();
        } else {
            System.exit(0);
        }
    }
    public void restartGame() {
        //Restarts the game
        score = 0;
        winConditionReached=false;
        updateScore();
```

```
        initializeGrid();
        updateGridLabels();
    }
    public void updateScore() {
        //Updates the score
        scoreLabel.setText("SCORE: " + score);
        if (score > highScore) {
            highScore = score;
            highScoreLabel.setText("HIGH SCORE: " + highScore);
        }
    }
    public static void main(String[] args) {
        //Main method
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new Game2048();
            }
        });
    }
}
```