SEMI-FINAL PROJECT: GAME DEVELOPMENT USING JAVA LANGUAGE

GROUP NAME: Y8 (GROUP 4)

NAME OF MEMBERS:

AQUINO, CYRUS BRYLLE DE GUZMAN **(LEADER)**

CABIGON, ANDREA PARAS

DACANAY, REAN MARTIN JUGAL

LOMIBAO, VENICE GASPAR

PERALTA, JANRY MAYNIGO

PROJECT TITLE: 2048 GAME

GAME HISTORY:

2048 was developed by Gabriele Cirulli, a web developer from Italy, who released the game in March 2014 as an open-source project. Inspired by the earlier game Threes!, Cirulli aimed to create a simpler version that retained the core gameplay mechanics. Despite its simplicity, 2048 remains a significant milestone in the history of casual puzzle games, showcasing the power of minimalist design and engaging gameplay.

GAME DESCRIPTION:

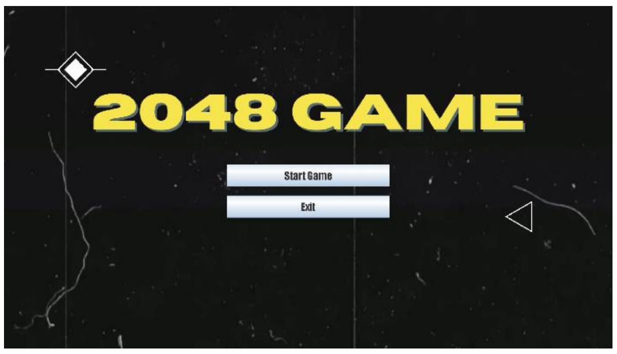
2048 is a single-player sliding tile puzzle game. The objective of the game is to combine tiles with the same number to create a tile with the sum of both tiles until 2048 tile is achieved. As the board fills up and strategic options dwindle, players must carefully plan their moves to avoid running out of space. With its simple yet challenging gameplay, 2048 offers hours of addictive puzzling fun, testing players' logic, spatial reasoning, and strategic thinking skills.

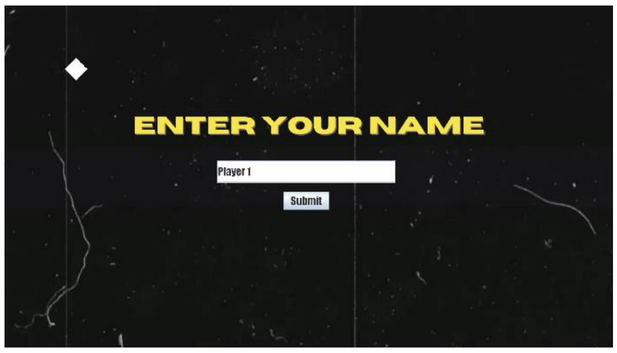
GAME MECHANICS:

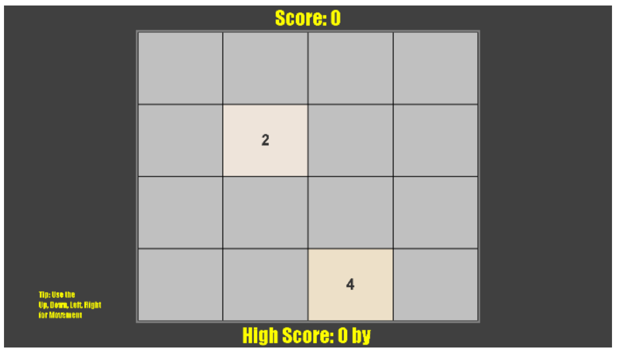
1. The game is played on a 4x4 grid.
2. Initially, the board has two tiles, each either a 2 or a 4.
3. Tiles can be moved in four directions: up, down, left, or right.

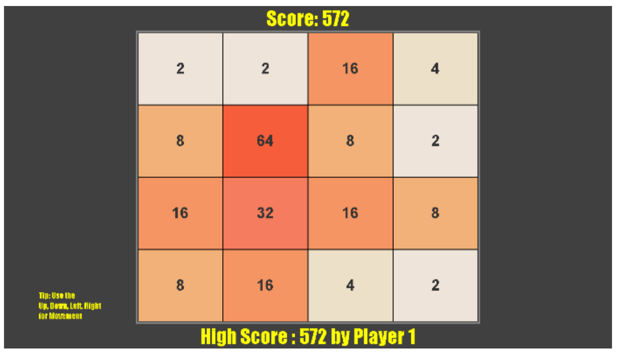
4. When a move is made, all tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid.
5. When two tiles with the same number collide while moving, they merge into a single tile with the sum of the two original numbers.
6. For example, if two tiles with the number 2 collide, they form a single tile with the number 4.
7. The resulting tile cannot merge with another tile again in the same move.
8. After each move, a new tile with a value of either 2 or 4 appears on an empty spot on the board.
9. The new tile's value and its position are determined randomly.
10. The player wins the game by creating a tile with the number 2048.
11. However, the game can continue beyond 2048 to achieve higher scores.
12. The game is over when there are no legal moves left, meaning no empty spaces on the board and no adjacent tiles with the same number that can be merged.

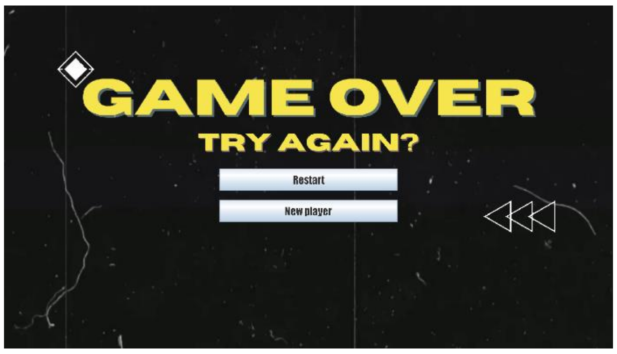
Screenshots of the Program:

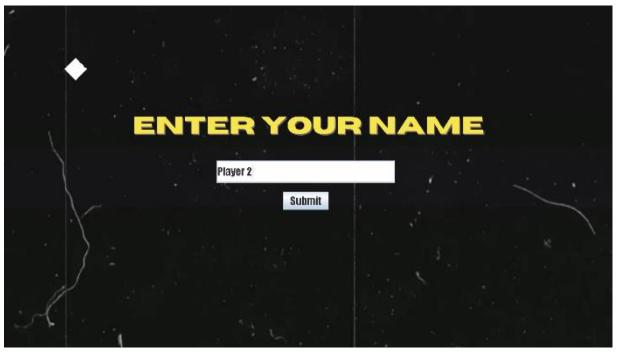


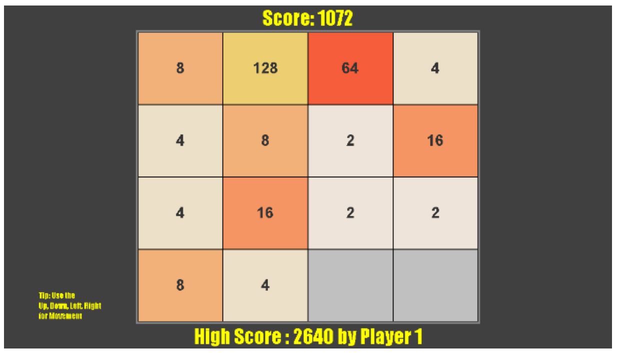


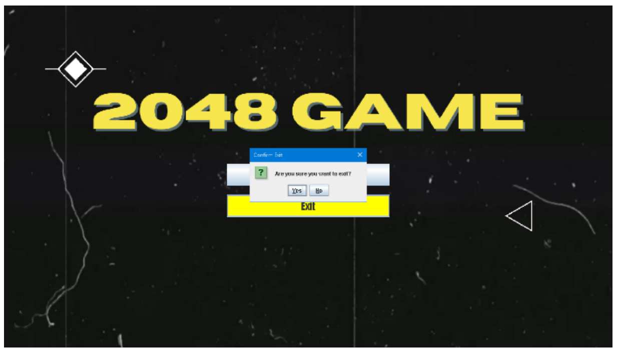



Score: 572
2 2 16 4
8 64 8 2
16 32 16 8
8 16 4 2
Tip: Use the Up, Down, Left, Right for Movement
High Score : 572 by Player 1









Source Code:

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.sound.sampled.*;

// Main class for the 2048 game
public class Game2048 {
    private static final int SIZE = 4; // Size of the grid (4x4)
    private int[][] grid; // Grid for the game
    private Random random; // Random number generator
    private int score; // Current score
    private String playerName; // Current player's name
    private String highScorePlayer; // Name of the player with the highest score
    private int highScore; // Highest score
    private JFrame frame; // Main window frame
    private JPanel gridPanel; // Panel for the grid
    private JPanel menuPanel; // Panel for the main menu
    private JLabel[][] gridLabels; // Labels for displaying the grid tiles
    private JLabel scoreLabel; // Label for displaying the score
    private JLabel highScoreLabel; // Label for displaying the high score
    private JLabel sideLabelLeft;
    private JLabel sideLabelRight;
    private Map<String, Integer> highScores; // Map for storing high scores
    private JPanel gameOverPanel;
    private Clip backgroundMusic; // Clip object for background music

    // Constructor to initialize the game
    public Game2048() {
        grid = new int[SIZE][SIZE];
        random = new Random();
        score = 0;
        highScore = 0;
        highScores = new HashMap<>();

        loadHighScores();

        // Set up the main window frame
        frame = new JFrame("2048 Game");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
        frame.setUndecorated(true); // Remove window decorations
        frame.setSize(600, 650);
        frame.setLayout(new BorderLayout());
        frame.setLocationRelativeTo(null);


        // Set up the grid panel
        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("Arial", Font.BOLD, 34));
                gridLabels[i][j].setOpaque(true);
                gridLabels[i][j].setBackground(Color.LIGHT_GRAY);
                gridPanel.add(gridLabels[i][j]);
            }
        }
        gridPanel.setBorder(new EmptyBorder(10, 10, 10, 10));
        gridPanel.setBorder(BorderFactory.createLineBorder(Color.gray, 5));

        // Set up the info panel for displaying the score and high score
        JPanel infoPanel = new JPanel(new GridLayout(2, 1));
        scoreLabel = new JLabel("Score: 0", JLabel.CENTER);
        scoreLabel.setFont(new Font("Impact", Font.PLAIN, 44));
        scoreLabel.setOpaque(true); // Make the JLabel transparent
        scoreLabel.setBackground(Color.DARK_GRAY); // Set background color to
black
        scoreLabel.setForeground(Color.YELLOW); // Set text color to white

        highScoreLabel = new JLabel("High Score: 0 by ", JLabel.CENTER);
        highScoreLabel.setFont(new Font("Impact", Font.PLAIN, 44));
        highScoreLabel.setOpaque(true); // Make the JLabel opaque
        highScoreLabel.setBackground(Color.DARK_GRAY); // Set background color to
black
        highScoreLabel.setForeground(Color.YELLOW); // Set text color to white

        sideLabelLeft = new JLabel("<html><div style='text-align: left;padding-
left: 60px; padding-right:
60px;'><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><b
r><br><br><br><br><br><br>Tip: Use the <br>Up, Down, Left, Right<br> for
Movement</html>", JLabel.CENTER);
        sideLabelLeft.setFont(new Font("Impact", Font.PLAIN, 18));
        sideLabelLeft.setOpaque(true); // Make the JLabel opaque
        sideLabelLeft.setBackground(Color.DARK_GRAY); // Set background color to
black
        sideLabelLeft.setForeground(Color.YELLOW); // Set text color to white

        sideLabelRight = new
JLabel("
 ", JLabel.CENTER);
        sideLabelRight.setFont(new Font("Impact", Font.PLAIN, 24));
        sideLabelRight.setOpaque(true); // Make the JLabel opaque
        sideLabelRight.setBackground(Color.DARK_GRAY); // Set background color to
black
        sideLabelRight.setForeground(Color.WHITE); // Set text color to white

        infoPanel.add(scoreLabel);
        infoPanel.add(highScoreLabel);
        infoPanel.add(sideLabelLeft);
        infoPanel.add(sideLabelRight);

        // Add a key listener to handle user input
        frame.addKeyListener(new KeyAdapter() {
            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()) {
                    showGameOverPanel();
                }
            }
        });
        frame.setFocusable(true);
        frame.requestFocus();
        frame.setVisible(true);
        frame.getContentPane().add(gridPanel, BorderLayout.CENTER);

        // Show the main menu when the game starts
        createMainMenu();
        showMainMenu();

        playBackgroundMusic("background_music.wav");
    }

    public class ImagePanel extends JPanel {
        private Image backgroundImage;

        // Constructor to set the background image
        public ImagePanel(String imagePath) {
            backgroundImage = new ImageIcon(imagePath).getImage();
            setLayout(new GridBagLayout()); // Use GridBagLayout for components
        }

        // Override the paintComponent method to draw the background image scaled
        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            g.drawImage(backgroundImage, 0, 0, getWidth(), getHeight(), this);
        }
    }

    private void playBackgroundMusic(String filePath) {
        try {
            // Load the audio file
            AudioInputStream audioInputStream =
AudioSystem.getAudioInputStream(new File(filePath).getAbsoluteFile());
            backgroundMusic = AudioSystem.getClip();
            backgroundMusic.open(audioInputStream);

            // Loop the background music indefinitely
            backgroundMusic.loop(Clip.LOOP_CONTINUOUSLY);

            // Start playing the background music
            backgroundMusic.start();
        } catch (UnsupportedAudioFileException | IOException |
LineUnavailableException e) {
            e.printStackTrace();
        }
    }

    // Show the main menu
    public void showMainMenu() {
        frame.getContentPane().removeAll();
        frame.getContentPane().add(menuPanel, BorderLayout.CENTER);
        frame.revalidate();
        frame.repaint();

    }

    // Initialize the grid with starting numbers
    public void initializeGrid() {
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                grid[i][j] = 0; // Set all grid cells to 0
            }
        }
        addNewNumber();
        addNewNumber();
    }

    // Add a new number (2 or 4) to a random empty cell in the grid
    public void addNewNumber() {
        int row, col;
        do {
            row = random.nextInt(SIZE);
            col = random.nextInt(SIZE);
        } while (grid[row][col] != 0); // Find an empty cell
        grid[row][col] = (random.nextInt(2) + 1) * 2; // Add a 2 or 4 to the cell
    }

    // Update the display of the grid
    public void updateGridLabels() {
        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) {
                    gridLabels[i][j].setText(String.valueOf(grid[i][j]));
                    gridLabels[i][j].setBackground(getTileColor(grid[i][j]));
                } else {
                    gridLabels[i][j].setText(String.valueOf(grid[i][j]));
                    gridLabels[i][j].setBackground(getTileColor(grid[i][j]));
                }
                gridLabels[i][j].setBorder(BorderFactory.createLineBorder(Color.B
LACK));
            }
        }
    }
```

```
// Get the color for a tile based on its value
    public Color getTileColor(int value) {
        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;
        }
    }

    // Move tiles up and combine if necessary
    public void moveUp() {
        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();
        }
    }

    // Move tiles down and combine if necessary
    public void moveDown() {
        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();
        }
    }

    // Move tiles left and combine if necessary
    public void moveLeft() {
        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();
        }
    }

    // Move tiles right and combine if necessary
    public void moveRight() {
        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();
        }
    }

    // Update the displayed score
    public void updateScore() {
        scoreLabel.setText("Score: " + score);
        if (score > highScore) {
            highScore = score;
            highScorePlayer = playerName;
            highScoreLabel.setText("High Score : " + highScore + " by " +
highScorePlayer);
        }
    }

    // Check if the game is over
    public boolean isGameOver() {
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                if (grid[i][j] == 0) {
                    return false; // There is at least one empty cell
                }
                if (i > 0 && grid[i][j] == grid[i - 1][j]) {
                    return false; // There is a mergeable cell above
                }
                if (i < SIZE - 1 && grid[i][j] == grid[i + 1][j]) {
                    return false; // There is a mergeable cell below
                }
                if (j > 0 && grid[i][j] == grid[i][j - 1]) {
                    return false; // There is a mergeable cell to the left
                }
                if (j < SIZE - 1 && grid[i][j] == grid[i][j + 1]) {
                    return false; // There is a mergeable cell to the right
                }
            }
        }
        return true; // No moves left, game over
    }

    // Show the game over message and save the high score
    public void createGameOverPanel() {
        gameOverPanel = new ImagePanel("gameover.png");
        gameOverPanel.setLayout(new GridBagLayout());
        gameOverPanel.setBackground(Color.LIGHT_GRAY);

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.anchor = GridBagConstraints.CENTER; // Center horizontally
        gbc.insets = new Insets(20, 0, 0, 0); // Add some top margin

        JLabel gameOverLabel = new JLabel(" ");
        gameOverLabel.setFont(new Font("Arial", Font.BOLD, 36));
        gbc.gridx = 0;
        gbc.gridy = 0;
        gameOverPanel.add(gameOverLabel, gbc);

        gbc.gridy++; // Move to the next row
        JButton restartButton = new JButton("Restart");
        restartButton.setFont(new Font("Impact", Font.PLAIN, 24));
        restartButton.setPreferredSize(new Dimension(400, 50)); // Adjust width
and height

        restartButton.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent e) {
                restartButton.setBackground(Color.YELLOW); // Change background
color when mouse enters
            }

            public void mouseExited(MouseEvent e) {
                restartButton.setBackground(UIManager.getColor("Button.background
")); // Restore default background color when mouse exits
            }
        });

        restartButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                startGame(); // Restart the game when the button is clicked
            }
        });
        gameOverPanel.add(restartButton, gbc);

        gbc.gridy++; // Move to the next row
        JButton newPlayerButton = new JButton("New player");
        newPlayerButton.setFont(new Font("Impact", Font.PLAIN, 24));
        newPlayerButton.setPreferredSize(new Dimension(400, 50)); // Adjust width
and height

        newPlayerButton.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent e) {
                newPlayerButton.setBackground(Color.YELLOW); // Change background
color when mouse enters
            }

            public void mouseExited(MouseEvent e) {
                newPlayerButton.setBackground(UIManager.getColor("Button.backgrou
nd")); // Restore default background color when mouse exits
            }
        });

        newPlayerButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showMainMenu(); // Restart the game when the button is clicked
            }
        });
        gameOverPanel.add(newPlayerButton, gbc);
    }

    // Method to show the game over panel
    public void showGameOverPanel() {
        frame.getContentPane().removeAll(); // Clear the content pane
        createGameOverPanel(); // Create the game over panel
        frame.getContentPane().add(gameOverPanel, BorderLayout.CENTER); // Add
the game over panel
        frame.revalidate(); // Revalidate the frame to reflect changes
        frame.repaint(); // Repaint the frame to reflect changes
    }

    // Create the main menu panel with start and exit buttons
    public void createMainMenu() {
        menuPanel = new ImagePanel("menu.png"); // Create an ImagePanel with the
background image
        menuPanel.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.insets = new Insets(10, 500, 10, 500);  // Add padding around
components

        JLabel titleLabel = new JLabel(" ", JLabel.CENTER);
        titleLabel.setFont(new Font("Arial", Font.PLAIN, 36));
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.gridwidth = 2;
        menuPanel.add(titleLabel, gbc);


        JButton startButton = new JButton("Start Game");
        startButton.setFont(new Font("Impact", Font.PLAIN, 24));
        startButton.setPreferredSize(new Dimension(200, 50));
        startButton.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent e) {
                startButton.setBackground(Color.yellow); // Change background
color when mouse enters
            }

            public void mouseExited(MouseEvent e) {
                startButton.setBackground(UIManager.getColor("Button.background")
); // Restore default background color when mouse exits
            }
        });
        startButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showPlayerNameInput(); // Call showPlayerNameInput directly
            }
        });
        gbc.gridx = 0;
        gbc.gridy = 1; // Change the gridy value for startButton
        gbc.gridwidth = 2; // Span both columns
        gbc.weightx = 0.5;
        gbc.anchor = GridBagConstraints.CENTER; // Center horizontally
        menuPanel.add(startButton, gbc);

        JButton exitButton = new JButton("Exit");
        exitButton.setFont(new Font("Impact", Font.PLAIN, 24));
        exitButton.setPreferredSize(new Dimension(200, 50));
        exitButton.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent e) {
                exitButton.setBackground(Color.yellow); // Change background
color when mouse enters
            }

            public void mouseExited(MouseEvent e) {
                exitButton.setBackground(UIManager.getColor("Button.background"))
; // Restore default background color when mouse exits
            }
        });
        exitButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int choice = JOptionPane.showConfirmDialog(null, "Are you sure
you want to exit?", "Confirm Exit", JOptionPane.YES_NO_OPTION);
                if (choice == JOptionPane.YES_OPTION) {
                    System.exit(0);
                }
            }
        });
        gbc.gridx = 0;
        gbc.gridy = 2; // Change the gridy value for exitButton
        gbc.gridwidth = 2; // Span both columns
        gbc.weightx = 0.5;
        gbc.anchor = GridBagConstraints.CENTER; // Center horizontally
        menuPanel.add(exitButton, gbc);
    }

    // Show the player name input panel
    public void showPlayerNameInput() {
        ImagePanel playerNamePanel  = new ImagePanel("entername.png");
        playerNamePanel.setLayout(new BoxLayout(playerNamePanel,
BoxLayout.Y_AXIS));
```

```
        playerNamePanel.setBorder(BorderFactory.createEmptyBorder(300, 100, 100,
100));

        JLabel promptLabel = new JLabel(" ");
        promptLabel.setFont(new Font("Arial", Font.BOLD, 24));
        promptLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        playerNamePanel.add(promptLabel);

        JTextField nameField = new JTextField();
        nameField.setFont(new Font("Impact", Font.PLAIN, 24));
        nameField.setMaximumSize(new Dimension(400, 50));
        playerNamePanel.add(Box.createVerticalStrut(20));
        playerNamePanel.add(nameField);

        JButton submitButton = new JButton("Submit");
        submitButton.setFont(new Font("Impact", Font.PLAIN, 24));
        submitButton.setAlignmentX(Component.CENTER_ALIGNMENT);
        submitButton.addMouseListener(new MouseAdapter() {
            public void mouseEntered(MouseEvent e) {
                submitButton.setBackground(Color.yellow); // Change background
color when mouse enters
            }

            public void mouseExited(MouseEvent e) {
                submitButton.setBackground(UIManager.getColor("Button.background"
)); // Restore default background color when mouse exits
            }
        });
        submitButton.addActionListener(e -> {
            playerName = nameField.getText();
            if (playerName.isEmpty()) {
                JOptionPane.showMessageDialog(frame, "Please enter your name.",
"Error", JOptionPane.ERROR_MESSAGE);
            } else {
                startGame();
            }
        });
        playerNamePanel.add(Box.createVerticalStrut(20));
        playerNamePanel.add(submitButton);

        frame.getContentPane().removeAll();
        frame.getContentPane().add(playerNamePanel, BorderLayout.CENTER);
        frame.revalidate();
        frame.repaint();
    }

    // Start the game
    public void startGame() {
        frame.getContentPane().removeAll(); // Clear the content pane
        showPlayerNameInput(); // Show the player name input panel
        frame.revalidate(); // Revalidate the frame to reflect changes
        frame.repaint(); // Repaint the frame to reflect changes
        score = 0; // Reset the score
        initializeGrid();
        updateGridLabels();
        updateScore();

        frame.getContentPane().removeAll();
        frame.getContentPane().add(gridPanel, BorderLayout.CENTER);
        frame.getContentPane().add(scoreLabel, BorderLayout.NORTH);
        frame.getContentPane().add(highScoreLabel, BorderLayout.SOUTH);
        frame.getContentPane().add(sideLabelLeft, BorderLayout.WEST);
        frame.getContentPane().add(sideLabelRight, BorderLayout.EAST);
        frame.revalidate();
        frame.repaint();
    }

    // Load high scores from a file
    private void loadHighScores() {
        try (BufferedReader reader = new BufferedReader(new
FileReader("highscores.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(":");
                if (parts.length == 3) {  // Expecting
"name:score:highScorePlayer"
                    String name = parts[0].trim();
                    int score = Integer.parseInt(parts[1].trim());
                    String player = parts[2].trim();
                    highScores.put(name, score);
                    if (score > highScore) {
                        highScore = score;
                        highScorePlayer = player;
                        highScoreLabel.setText("High Score: " + highScore + " by
" + highScorePlayer);
                    }
                }
            }
        } catch (IOException e) {
            // Handle file not found or other IO exceptions
        }
    }

    // Save high scores to file
    public void saveHighScores() {
        highScores.put(playerName, score);
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("highscores.dat"))) {
```

```
            oos.writeObject(highScores);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // Main method to start the game
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new Game2048();
            }
        });
    }
}
```

GDRIVE LINK OF PROGRAM RESOURCES:

https://drive.google.com/drive/folders/1PZQqjKjdAMlasPBpgkoTFrv2pOSUdQst?usp=sharing