# JAVA PROJECT

COMPUTER SCIENCE & ENGINEERING

(Artificial Intelligence & Machine Learning)


Submitted by –

24WH1A6650 - Ms. TABASSUM BEGUM
24WH1A6653 - Ms. SANJANA SIRIMALLA
24WH1A6660 - Ms. KRITIKA ENDURI
24WH1A6662 - Ms. YASHASWINI NOOLE
24WH1A6613 - Ms. JAHNAVI YELLURI


**Under the esteemed guidance of**

**Dr. B Lakshmi Praveena**

**Professor**



BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Accredited by NAAC with A Grade

Bachupally, Hyderabad – 500090

# BVRIT HYDERABAD

# COLLEGE OF ENGINEERING FOR WOMEN

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Accredited by NAAC with A Grade
Bachupally, Hyderabad – 500090

**Department of**

Computer Science & Engineering
(Artificial Intelligence & Machine Learning)


## CERTIFICATE

This is to certify that the Java Project is a Bonafide work carried out by 24WH1A6650 Ms. Tabassum Begum, 24WH1A6653 Ms. Sanjana Sirimalla, 24WH1A6660 Ms. Kritika Enduri, 24WH1A6662 Ms. Yashaswini Noole, 24WH1A6613 Ms. Jahnavi Yelluri in partial fulfilment for the award of B.Tech degree in Computer Science & Engineering (AI & ML), BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad, affiliated to Jawaharlal Nehru Technological University Hyderabad, under my guidance and supervision. The results embodied in the project work have not been submitted to any other.

Dr. B Lakshmi Praveena
Professor
Dept of CSE (AI & ML)

# INDEX

# ABSTRACT:

The Memory Game – Flipping Tiles is a Java-based desktop application developed using the Swing framework.
The main objective of this project is to create an interactive tile-matching game where players flip tiles to find matching pairs within a grid layout. The game features a graphical user interface with clickable tiles that reveal hidden images or symbols when selected, challenging players to remember tile positions and make successful matches.
The application demonstrates core concepts of Java GUI development, event handling, game logic implementation, and dynamic component state management. This project is simple, lightweight, and ideal for understanding how interactive memory-based games work at a fundamental level.

# INTRODUCTION:

Memory and pattern recognition games have been popular tools for cognitive development and entertainment across all age groups.
Although advanced memory games may include features like multiplayer modes, databases for score tracking, and web-based interfaces, this project focuses on the core gameplay mechanics and graphical user interface aspects of a tile-matching game.
The goal of the Memory Game – Flipping Tiles is to create an interactive grid-based window where players can click on tiles to reveal hidden images or symbols, challenge their memory by finding matching pairs, and complete the game by matching all tiles successfully.
The application uses Java Swing, which provides robust components for building graphical user interfaces and handling user interactions.
This project helps beginners understand:

- GUI development and design principles
- Layout management in Java (GridLayout for tile arrangement)
- Event-driven programming using ActionListeners and MouseListeners
- Working with Swing components like JButton, JPanel, JLabel, and ImageIcon
- Game state management and logic implementation

The simplicity of this project makes it ideal for learning the foundations of Java UI programming while creating an engaging and interactive gaming experience.

# LITERATURE REVIEW:

Java Swing has long served as a widely adopted toolkit for creating desktop-based graphical user interfaces. Prior research and educational material on Java GUI development emphasize several foundational concepts that directly support the implementation of interactive games such as the Memory Tile Game

**Swing Framework**

Swing, a major part of the Java Foundation Classes (JFC), provides a robust set of platform-independent UI components suitable for building event-driven games. The framework is known for:

- Its lightweight component architecture
- Customizable widgets such as JButton, JPanel, and JFrame
- Support for double buffering, enabling smoother animations and transitions

In educational contexts, Swing is frequently recommended for beginner and intermediate developers due to its accessibility and flexibility.
For this project, Swing components such as JButton (tiles), JPanel (game grid), and JFrame (main game window) form the backbone of the interface.

**Event Handling in Swing**

Event-driven programming is a central paradigm in GUI development.

Key elements include:

- The ActionListener interface for responding to tile flips
- Timer events (javax.swing.Timer) for implementing delayed flips when tiles do not match

In the Memory Tile Game, event handling ensures interactive behavior such as:

- Flipping tiles when clicked
- Temporarily revealing mismatched tiles
- Detecting game-complete conditions

**Use of Layout Managers**

For the Memory Game:

- A GridLayout arranges the tile buttons into a uniform grid
- BorderLayout used in the main window to position score panels or menus

These techniques support maintainability and clarity in UI design, as emphasized in Java GUI development guidelines.

**Game Logic**

In this project, classes like Tile and GameBoard hold the underlying game logic (Model), Swing components display the game state (View), and event-handling code acts as the Controller. This separation enhances maintainability and allows multiple developers to work independently.

# TECHNOLOGY STACK:

## Programming Language

- **Java (JDK 8+)**
  Used to develop and run the application.

## Libraries and Frameworks

- **Java Swing**
  Used to build the GUI components like buttons, text areas, and combo boxes.
- **Java Utilities**
  Used to access and use data structures needed by the model.

## Tools Used

- **IDE:** VS Code (Windows)
- **JVM:** To execute the compiled Java program

## Concepts Used

- Object-Oriented Programming
- Event Handling
- GUI Development
- Layout Management

# IMPLEMENTATION(CODE):

**Main.java:**

```
public class Main {
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                GameWindow window = new GameWindow();
                window.setVisible(true);
            }
        });
    }
}
```

**GameWindow.java:**

```
import javax.swing.*;
import java.awt.*;

public class GameWindow extends JFrame {
    CardLayout cardLayout;
    JPanel mainPanel;
    public static final String MAIN_MENU = "MainMenu";
    public static final String GAME = "Game";

    public GameWindow() {
        super("Memory Game");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(640, 720);
        setLocationRelativeTo(null);

        cardLayout = new CardLayout();
        mainPanel = new JPanel(cardLayout);

        MainMenuPanel menu = new MainMenuPanel(this);
        GamePanel game = new GamePanel(this);

        mainPanel.add(menu, MAIN_MENU);
        mainPanel.add(game, GAME);

        setContentPane(mainPanel);
        showScreen(MAIN_MENU);
    }
```

```
    public void showScreen(String screenName) {
        cardLayout.show(mainPanel, screenName);
    }
}
```

**MainMenuPanel.java:**

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MainMenuPanel extends JPanel {
    private GameWindow parent;

    public MainMenuPanel(GameWindow parent) {
        this.parent = parent;
        setLayout(new GridBagLayout());
        setBackground(new Color(255, 192, 203)); // Pink background

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(20, 20, 20, 20);
        gbc.gridx = 0;
        gbc.gridy = 0;

        // Title
        JLabel title = new JLabel("MEMORY FLIP GAME");
        title.setFont(new Font("Arial", Font.BOLD, 40));
        title.setForeground(new Color(75, 0, 130)); // Indigo color
        add(title, gbc);

        // Start Game Button
        gbc.gridy++;
        JButton startBtn = new JButton("Start Game");
        startBtn.setFont(new Font("Arial", Font.PLAIN, 18));
        startBtn.setBackground(new Color(255, 105, 180)); // Hot pink
        startBtn.setForeground(Color.BLACK);
        startBtn.setFocusPainted(false);
        startBtn.setPreferredSize(new Dimension(200, 50));
        startBtn.addActionListener(e -> parent.showScreen(GameWindow.GAME));
        add(startBtn, gbc);

        // Instructions Button
        gbc.gridy++;
        JButton instructionsBtn = new JButton("Instructions");
```

```
        instructionsBtn.setFont(new Font("Arial", Font.PLAIN, 18));
        instructionsBtn.setBackground(new Color(255, 255, 0)); // Yellow
        instructionsBtn.setForeground(Color.BLACK);
        instructionsBtn.setFocusPainted(false);
        instructionsBtn.setPreferredSize(new Dimension(200, 50));
        instructionsBtn.addActionListener(e -> showInstructions());
        add(instructionsBtn, gbc);

        // Exit Button
        gbc.gridy++;
        JButton exitBtn = new JButton("Exit");
        exitBtn.setFont(new Font("Arial", Font.PLAIN, 18));
        exitBtn.setBackground(new Color(255, 99, 71)); // Tomato/Red
        exitBtn.setForeground(Color.WHITE);
        exitBtn.setFocusPainted(false);
        exitBtn.setPreferredSize(new Dimension(200, 50));
        exitBtn.addActionListener(e -> {
            Window w = SwingUtilities.getWindowAncestor(MainMenuPanel.this);
            if (w != null) w.dispose();
        });
        add(exitBtn, gbc);
    }

    private void showInstructions() {
        JOptionPane.showMessageDialog(this,
            "Memory Match Game Instructions:\n\n" +
            "1. Click on tiles to reveal numbers\n" +
            "2. Try to find matching pairs\n" +
            "3. Match all pairs to win\n" +
            "4. The fewer moves, the better!",
            "Instructions", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

**GamePanel.java:**

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

public class GamePanel extends JPanel {
```

```
    private GameWindow parent;
    private JPanel gridPanel;
    private JLabel movesLabel;

    private TileButton firstSelected = null;
    private TileButton secondSelected = null;

    private int moves = 0;
    private int pairsFound = 0;

    private int rows = 4, cols = 4;
    private TileButton[][] grid;

    private Timer flipBackTimer;

    public GamePanel(GameWindow parent) {
        this.parent = parent;
        setLayout(new BorderLayout());

        // Top panel
        JPanel top = new JPanel(new BorderLayout());
        JLabel title = new JLabel("Memory Match", SwingConstants.CENTER);
        title.setFont(new Font("Arial", Font.BOLD, 28));
        top.add(title, BorderLayout.CENTER);

        movesLabel = new JLabel("Moves: 0", SwingConstants.CENTER);
        movesLabel.setFont(new Font("Arial", Font.PLAIN, 18));
        top.add(movesLabel, BorderLayout.SOUTH);

        add(top, BorderLayout.NORTH);

        // Grid
        gridPanel = new JPanel(new GridLayout(rows, cols, 8, 8));
        gridPanel.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
        add(gridPanel, BorderLayout.CENTER);

        // Bottom buttons
        JPanel bottom = new JPanel(new FlowLayout());
        JButton backBtn = new JButton("Back to Menu");
        backBtn.addActionListener(e -> {
            reset();
            parent.showScreen(GameWindow.MAIN_MENU);
        });
        bottom.add(backBtn);
```

```
        JButton resetBtn = new JButton("Reset");
        resetBtn.addActionListener(e -> reset());
        bottom.add(resetBtn);

        add(bottom, BorderLayout.SOUTH);

        initializeBoard();

        flipBackTimer = new Timer(800, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                flipBackTimer.stop();
                if (firstSelected != null) firstSelected.hideValue();
                if (secondSelected != null) secondSelected.hideValue();
                firstSelected = secondSelected = null;
            }
        });
        flipBackTimer.setRepeats(false);
    }

    private void initializeBoard() {
        gridPanel.removeAll();
        grid = new TileButton[rows][cols];

        ArrayList<Integer> values = new ArrayList<>();
        for (int i = 1; i <= (rows * cols / 2); i++) {
            values.add(i);
            values.add(i);
        }
        Collections.shuffle(values);
        Iterator<Integer> it = values.iterator();

        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                TileButton tb = new TileButton(r, c);
                tb.setValue(it.next());
                tb.addActionListener(e -> onTileClick(tb));
                grid[r][c] = tb;
                gridPanel.add(tb);
            }
        }

        moves = 0;
        pairsFound = 0;
```

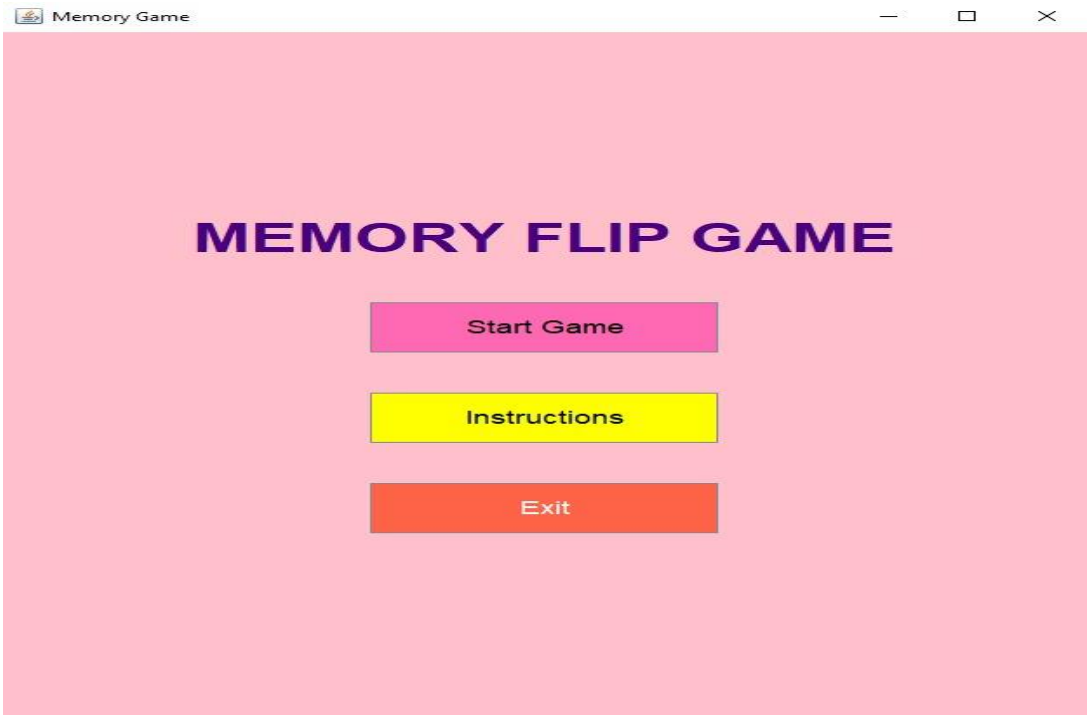
```
        movesLabel.setText("Moves: 0");
        revalidate();
        repaint();
    }

    private void onTileClick(TileButton b) {
        if (b.isRevealed() || b.isMatched()) return;
        if (flipBackTimer.isRunning()) return;

        if (firstSelected == null) {
            firstSelected = b;
            b.showValue();
        }
        else if (secondSelected == null && b != firstSelected) {
            secondSelected = b;
            b.showValue();

            moves++;
            movesLabel.setText("Moves: " + moves);

            if (firstSelected.getValue() == secondSelected.getValue()) {
                firstSelected.setMatched(true);
                secondSelected.setMatched(true);
                firstSelected = secondSelected = null;
                pairsFound++;
                checkGameOver();
            } else {
                flipBackTimer.restart();
            }
        }
    }

    private void checkGameOver() {
        if (pairsFound >= (rows * cols / 2)) {
            JOptionPane.showMessageDialog(this, "You won! Total moves: " +
moves);
        }
    }

    public void reset() {
        initializeBoard();
    }
}
```

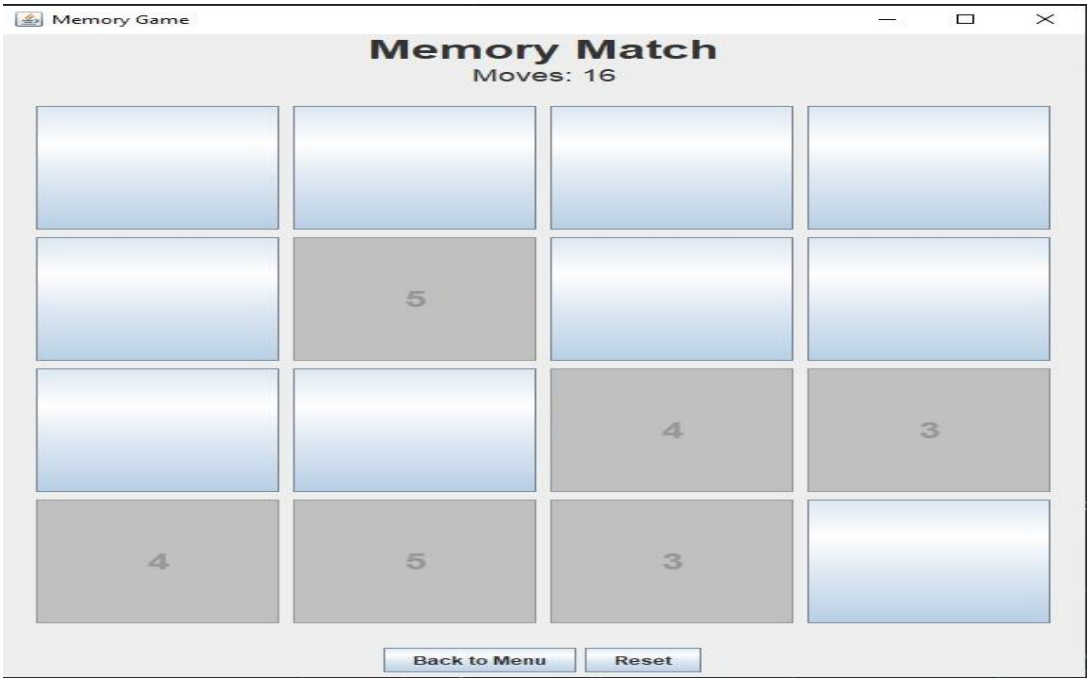
**TileButton.java:**

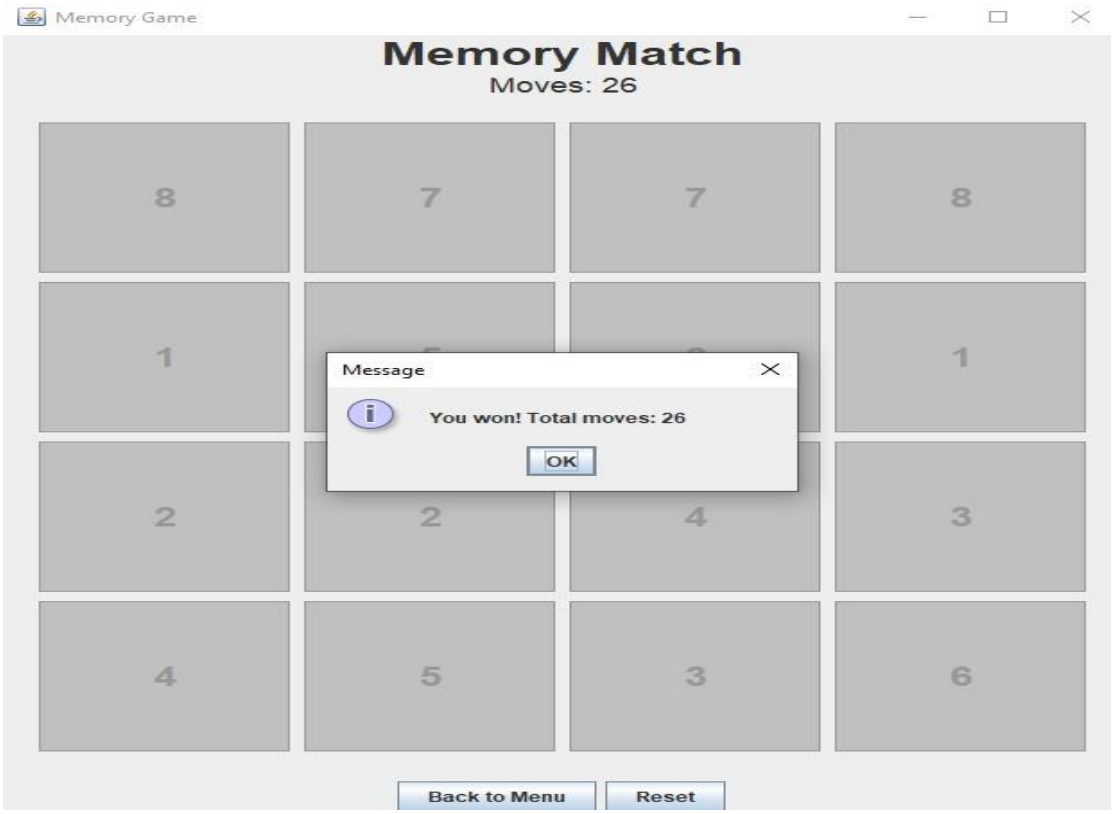
```
import javax.swing.*;
import java.awt.*;

public class TileButton extends JButton {
    private int row, col;
    private int value;
    private boolean revealed = false;
    private boolean matched = false;

    public TileButton(int r, int c) {
        super("");
        this.row = r;
        this.col = c;
        setFont(new Font("Arial", Font.BOLD, 24));
        hideValue();
    }

    public void setValue(int v) { this.value = v; }
    public int getValue() { return value; }
    public int getRow() { return row; }
    public int getCol() { return col; }

    public void showValue() {
        revealed = true;
        setText(Integer.toString(value));
        setEnabled(false);
    }

    public void hideValue() {
        if (!matched) {
            revealed = false;
            setText("");
            setEnabled(true);
        }
    }

    public void setMatched(boolean m) {
        matched = m;
        if (m) {
            setBackground(Color.LIGHT_GRAY);
            setEnabled(false);
        }
    }
```

```
    public boolean isMatched() { return matched; }
    public boolean isRevealed() { return revealed; }
}
```

# Output:




Memory Game
Memory Match
Moves: 26
8
7
7
8
1
1
2
2
4
3
4
5
3
6
Message
You won! Total moves: 26
OK
Back to Menu
Reset

# CONCLUSION:

The Memory Game - Flipping Tiles successfully demonstrates the fundamental concepts of Java GUI development using Swing. Players can engage in an interactive tile-matching experience by clicking on tiles to reveal hidden content and testing their memory skills to find all matching pairs. Although it does not use a database for score persistence or advanced features, the project serves as a strong foundation for building more complex gaming applications in the future. This project enhances understanding of event handling, game logic implementation, dynamic UI state management, and user interaction models in Java

# REFERENCES:

1. Oracle Java Tutorials - Creating Graphical User Interfaces
2. Oracle Java Documentation - Swing
3. Deitel & Deitel, **Java: How to Program**
4. Tutorials Point - Java Swing Basics
5. GeeksforGeeks - Java UI Programm