

# Whack-A-Mole for Linux | Plug & Play Edition

---

 Download & Try Now

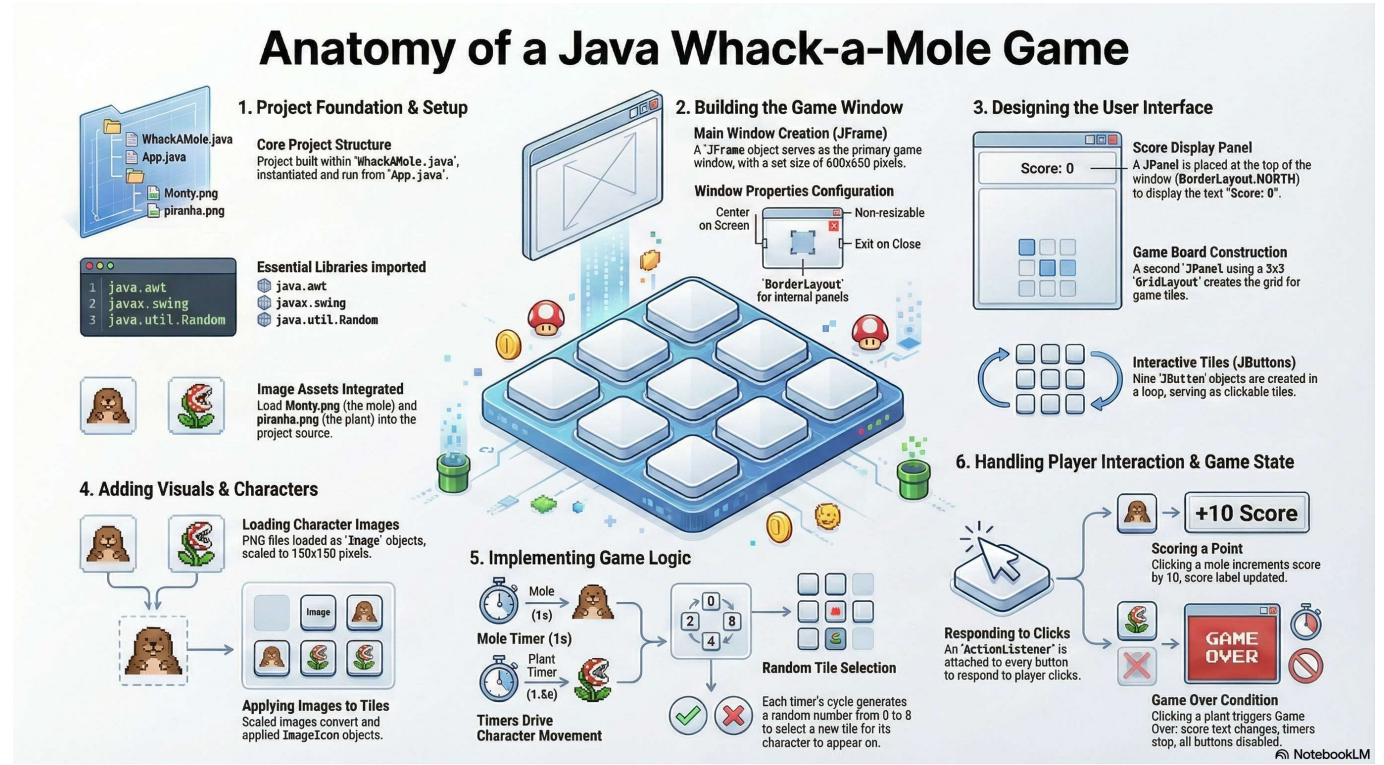
 whackamole | Plug & Play Edition


 Download [whackamole.deb](#)

Whack-A-Mole is a Java Swing desktop game that implements a 3x3 grid-based whack-a-mole mechanic with timed events, animated mole and Piranha Plant enemies, scoring, and game-over logic. Built using Java, Swing, AWT, javax.swing.Timer, and ImageIcon, it features enhancements like a welcome screen, reset/retry flow, high-score tracking, multiple hazards, and polished UI with custom AI-generated sprites/characters. The project demonstrates core concepts in GUI programming, event-driven design, image handling, and basic game logic, and is packaged as a .deb application for easy installation on Linux.

TechStack: Java, Swing, AWT, javax.swing.Timer, ImageIcon, VS Code

- Author: [Kintsugi-Programmer](#)



-  Download & Try Now
- [Table of Contents](#)
- [Introduction](#)
  - [Game Overview](#)
- [Project Setup](#)
  - [Development Environment](#)
  - [Project Creation Steps](#)
  - [Initial Project Structure](#)
  - [Download Required Files](#)
  - [Initial App.java Setup](#)
- [Game Window Setup](#)
  - [Window Specifications](#)
  - [Creating the Window Frame](#)
  - [Import Statements](#)
  - [Frame Properties](#)
  - [Important Note on Timing](#)
- [Text Panel and Score Display](#)
  - [Creating the Text Panel](#)
  - [Formatting the Text Label](#)
  - [Adding to Frame](#)
- [Game Board Panel Setup](#)
  - [Creating the Board Panel](#)
  - [Optional Styling](#)
- [Creating Buttons for Tiles](#)
  - [Button Array Setup](#)
  - [Creating and Adding Buttons](#)
- [Loading and Scaling Images](#)
  - [Image Files](#)
  - [Loading and Scaling Images](#)
  - [Explanation of Image Loading Process](#)
  - [Why Two-Step Process?](#)
  - [Removing Button Focus Rectangle](#)
- [Variables for Game State](#)
- [Moving the Mole](#)
  - [Mole Timer Setup](#)
  - [Why Use Intermediate Variable?](#)
- [Moving the Plant](#)
  - [Plant Timer Setup](#)
  - [Why Different Timer Values?](#)
- [Handling Tile Conflicts](#)
  - [The Conflict Problem](#)
  - [Solution: Tile Occupancy Check](#)
  - [How It Works](#)
- [Detecting Button Clicks](#)
  - [Adding Click Listeners to Tiles](#)
  - [Type Casting](#)

- Mole Click Behavior
- Plant Click Behavior
- Complete Game Flow
  - Game Sequence
  - Important Implementation Details
- Enhancements
  - Enhancement 1: Hole instead of null spaces
  - Enhancement 2: GameOver Mole Teasing Buttons
  - Enhancement 3: Second Piranha Plant
  - Enhancement 4: Reset Button
    - UI Changes
    - Logic Changes
    - Player Experience
  - Enhancement 5: High Score Tracking
  - Enhancement 6: Welcome Screen with Play Button
    - Purpose
    - Welcome Screen Design
    - Play Button Behavior
    - Technical Implementation
    - Player Experience
- Complete Code Structure
  - Class Variables (Instance Variables)
  - Constructor Method
- Key Concepts Learned
  - GUI Components
  - Layouts
  - Event Handling
  - Image Processing
  - Game Logic
- Final Codes
  - `src/App.java`
  - `src/WhackAMole.java`
- Development Notes
- Build + Export .deb Software Script
- Install and Run .deb Linux
  - Performance Tip
  - Testing Approach
  - Debugging
- Conclusion
- LICENSE

## Introduction

This Documentation teaches how to create a simple Whack-a-Mole game in Java with a graphical user interface (GUI). This is an excellent beginner project for those starting to learn Java and GUI development.

## Game Overview

- **Game Elements:** Uses images AI-Generated
- **Game Board:** Nine tiles arranged in a 3x3 grid
- **Main Character:** A mole that hops around the tiles
- **Obstacle:** A Piranha Plant that also moves around the board
- **Objective:** Click on tiles with the mole to earn points
- **Losing Condition:** Clicking on the Piranha Plant results in "Game Over"

## Project Setup

### Development Environment

- **IDE:** Visual Studio Code
- **Language:** Java
- **Prerequisites:** Java setup in Visual Studio Code (Documentation available in video description)

### Project Creation Steps

1. Open Visual Studio Code
  2. Use keyboard shortcut: **Ctrl + Shift + P**
  3. Select "Java: Create Java Project"
  4. Choose "No build tools"
  5. Create project on Desktop
  6. Name the project: "whack-a-mole"
  7. This creates a project folder with a **src** folder containing **App.java**
- java readme default

#### ## Getting Started

Welcome to the VS Code Java world. Here is a guideline to help you get started to write Java code in Visual Studio Code.

#### ## Folder Structure

The workspace contains two folders by default, where:

- ``src``: the folder to maintain sources
- ``lib``: the folder to maintain dependencies

Meanwhile, the compiled output files will be generated in the ``bin`` folder by default.

> If you want to customize the folder structure, open ``.vscode/settings.json`` and update the related settings there.

#### ## Dependency Management

The ``JAVA PROJECTS`` view allows you to manage your dependencies. More details can be found [\[here\]](https://github.com/microsoft/vscode-java-)(<https://github.com/microsoft/vscode-java->

```
dependency#manage-dependencies).
```

## Initial Project Structure

```
whack-a-mole/
├── src/
│   ├── App.java
│   ├── WhackAMole.java (create this file)
│   ├── piranha.PNG (download)
│   └── monty.PNG (download)
```

## Download Required Files

Before coding:

1. Visit GitHub link in video description
2. Download two image files:
  - **piranha.PNG** (Piranha Plant image)
  - **monty.PNG** (Mole image)
3. Drag both images into the **src** folder

## Initial App.java Setup

Modify the default **App.java** file:

```
public class App {
    public static void main(String[] args) throws Exception {
        new WhackAMole();
    }
}
```

- Remove the default print statement
- Create an instance of the **WhackAMole** class

```
public class WhackAMole {
}
```

- Collapse the App.java file as main code goes in WhackAMole.java

## Game Window Setup

- creating window
-

## Window Specifications

- **Width:** 600 pixels
- **Height:** 650 pixels
- **Note:** Game board is 600x600, but height is 650 to provide room for a score display text at the top

## Creating the Window Frame

```
// Create variables
int boardWidth = 600;
int boardHeight = 650;
JFrame frame = new JFrame("Whack-A-Mole");
```

## Import Statements

Add these import statements at the top of WhackAMole.java:

```
import java.awt.*; // GUI
import java.awt.event.*;
import java.util.Random; // Random Positions
import javax.swing.*; // GUI , Contain JFrame
```

Originally javax was intended to be for extensions, and sometimes things would be promoted out of javax into java.

## Frame Properties

Set frame properties in the constructor:

```
frame.setVisible(true);
frame.setSize(boardWidth, boardHeight);
frame.setLocationRelativeTo(null); // Opens window at center of screen
frame.setResizable(false);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // X button
terminates program
frame.setLayout(new BorderLayout());
```

this will create empty window blank, when run App.java



### Important Note on Timing

Place `frame.setVisible(true)` at the very end of the constructor after all components are added. This ensures all components load before the window becomes visible, preventing slow loading of individual buttons.

## Text Panel and Score Display

### Creating the Text Panel

```
// Create text label
JLabel textLabel = new JLabel();
```



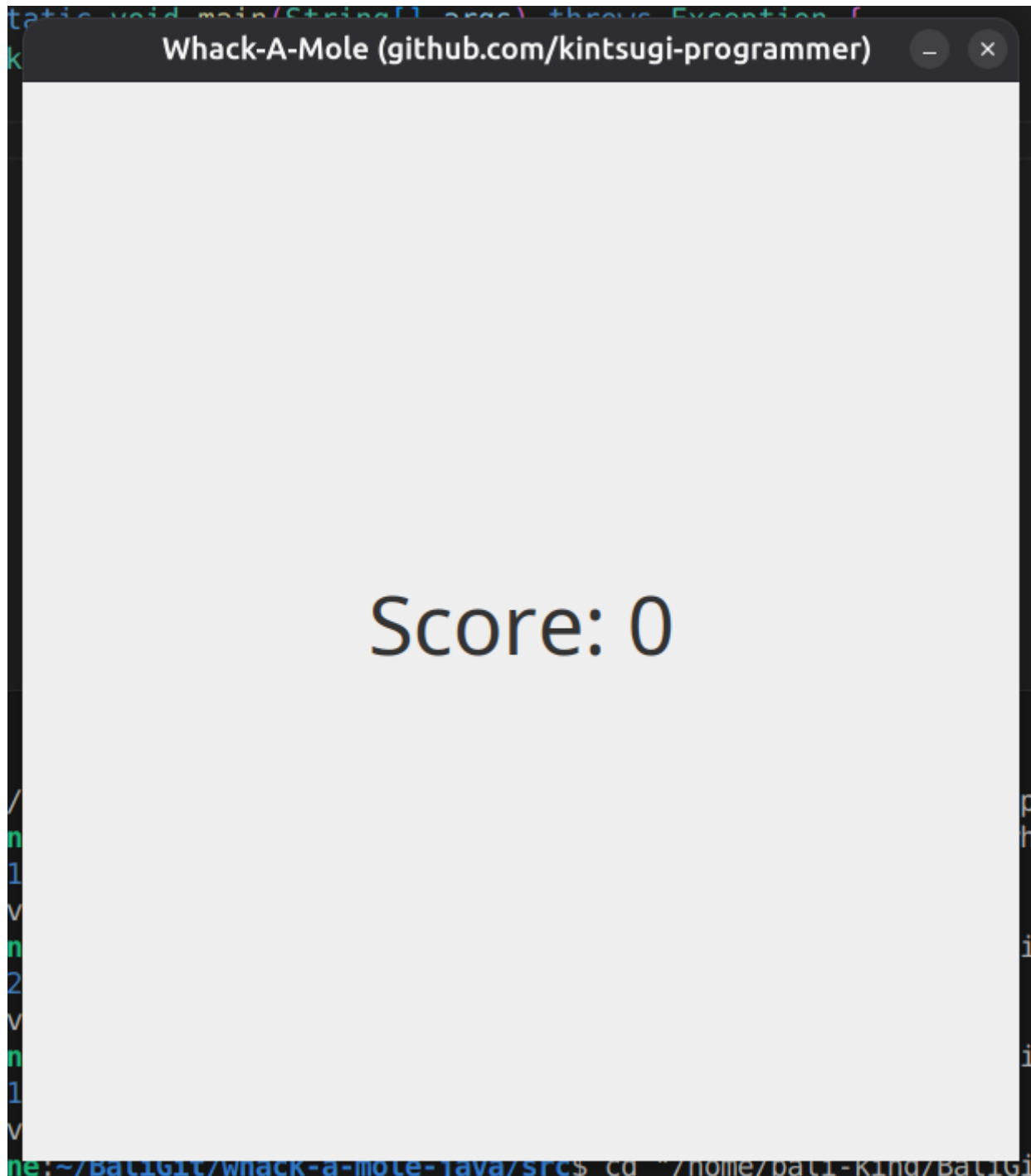
```
// Create text panel to hold label  
JPanel textPanel = new JPanel();
```

## Formatting the Text Label

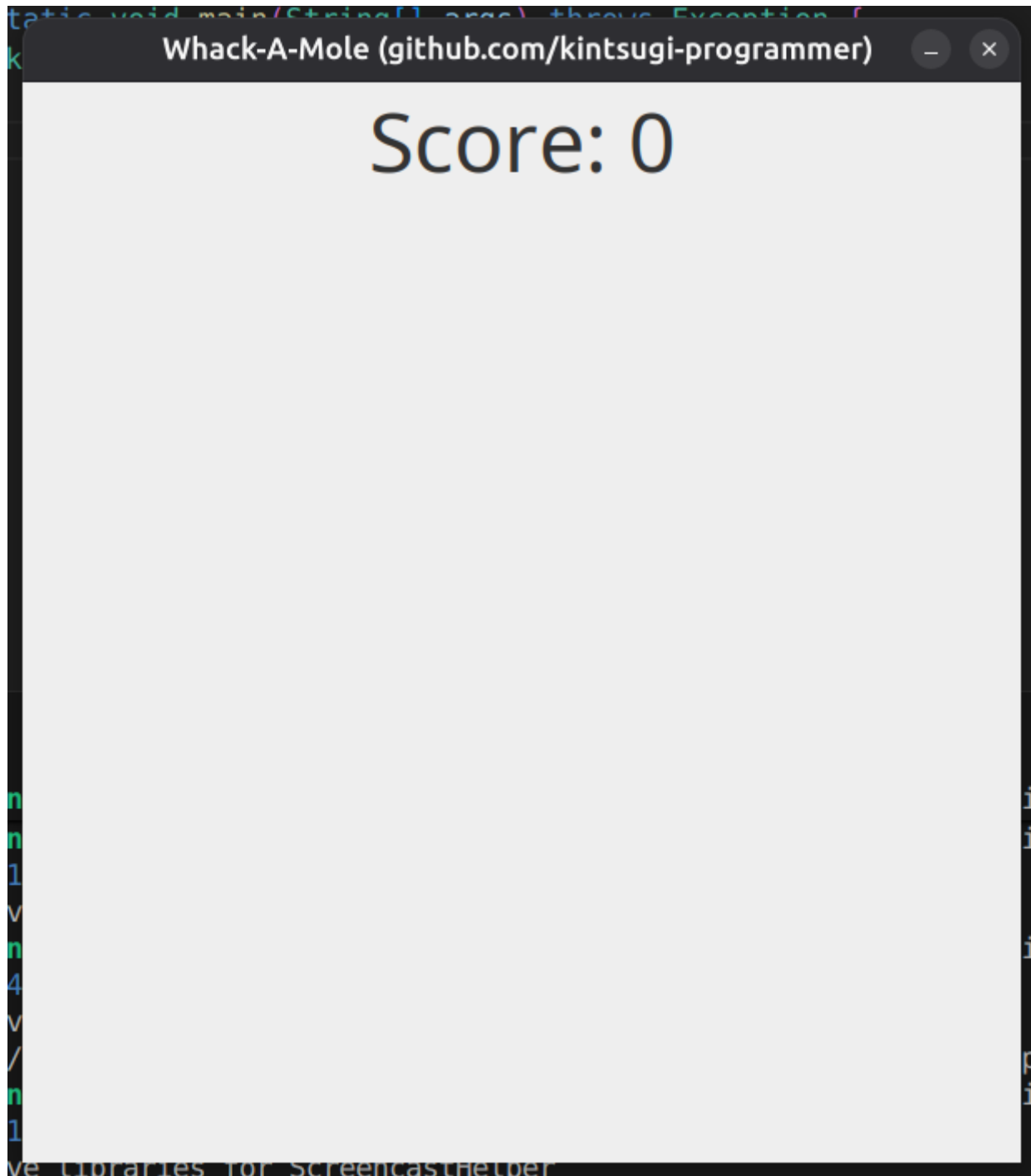
```
textLabel.setFont(new Font("Arial", Font.PLAIN, 50)); // 50px  
textLabel.setHorizontalAlignment(JLabel.CENTER); // Centers text  
horizontally  
textLabel.setText("Score: 0");  
textLabel.setOpaque(true); // Makes background visible
```

## Adding to Frame

```
textPanel.setLayout(new BorderLayout());  
textPanel.add(textLabel); // add textLabel in textPanel  
frame.add(textPanel); // add textPanel to frame
```



```
frame.add(textPanel, BorderLayout.NORTH); // BorderLayout.NORTH pushes
panel to top // add textPanel to frame
```



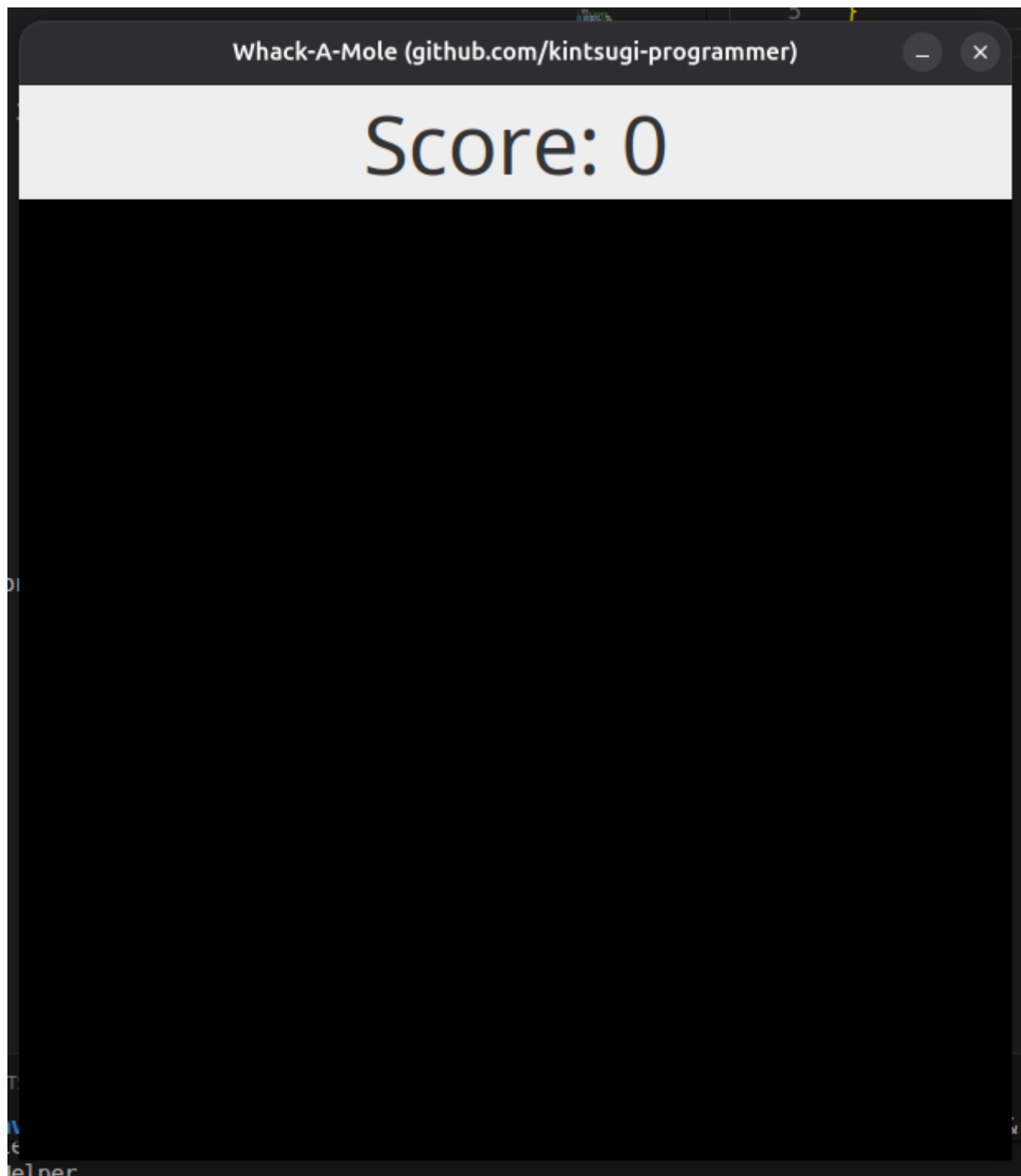
## Game Board Panel Setup

### Creating the Board Panel

```
JPanel boardPanel = new JPanel();  
boardPanel.setLayout(new GridLayout(3, 3)); // 3x3 grid of tiles  
frame.add(boardPanel);
```

### Optional Styling

```
boardPanel.setBackground(Color.BLACK); // Optional: makes background  
visible during development
```



## Creating Buttons for Tiles

### Button Array Setup

Create an array to track all nine buttons(outside constructor):

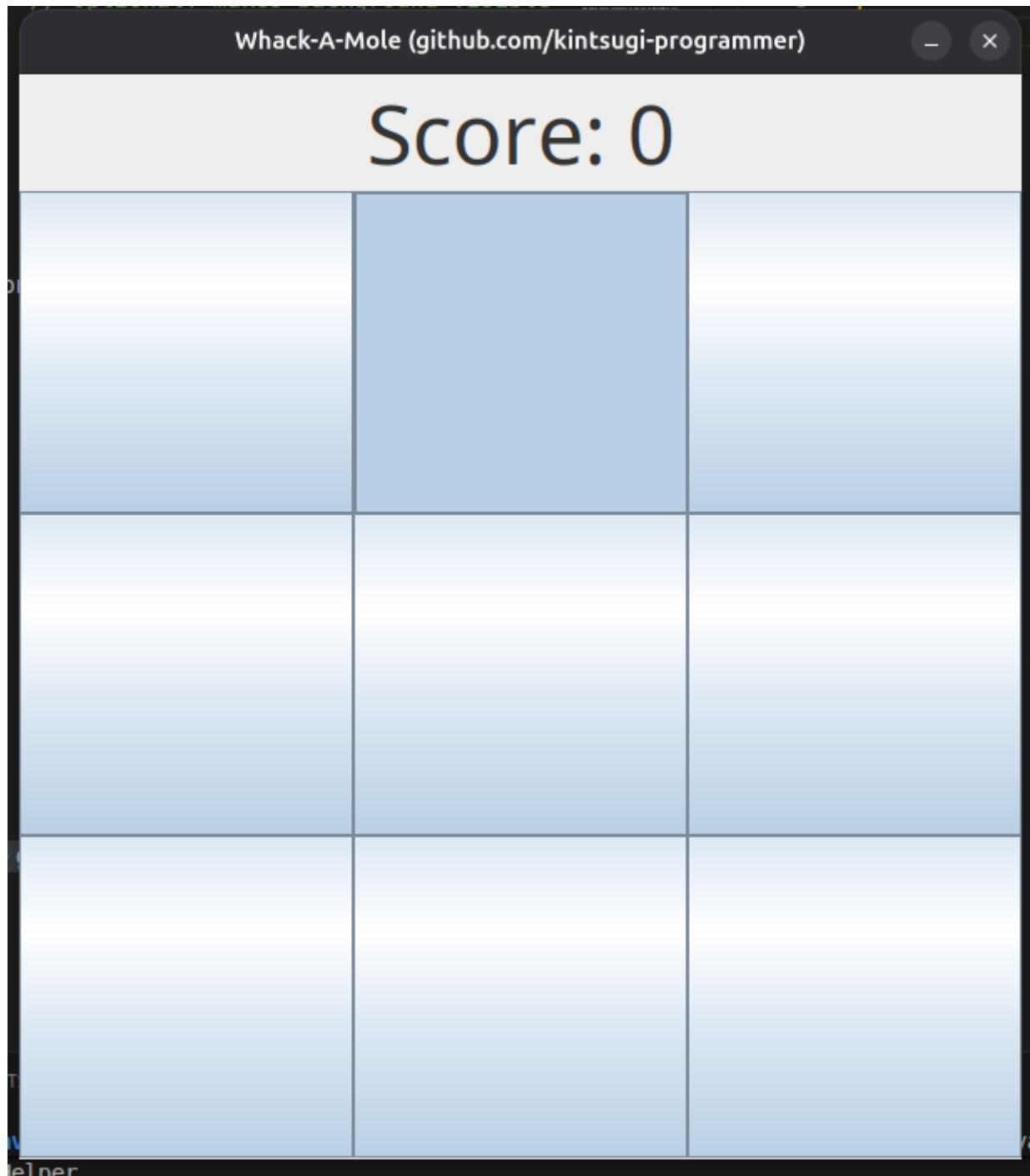
```
JButton[] board = new JButton[9];
```

### Creating and Adding Buttons

Use a for loop to create nine buttons:

```
for (int i = 0; i < 9; i++) {  
    JButton tile = new JButton();  
    board[i] = tile;  
    boardPanel.add(tile);  
}
```

Each button represents one tile on the game board; also clickable



## Loading and Scaling Images

Make ImageIcon outside Constructors

```
ImageIcon moleIcon;  
ImageIcon plantIcon;
```

### Inside Constructor

```
// Loading and Scaling Images  
moleIcon = new ImageIcon(getClass().getResource("./monty.png"));  
  
// constructing buttons/ tiles  
// JButton button1 = new JButton(); // one button  
for ( int i = 0; i<9; i++){ // using loop, to create tiles/buttons;  
instead of hardcode  
    JButton tile = new JButton();  
    board[i] = tile;  
    boardPanel.add(tile); // appending to boardPanel Grid one-by-one  
    tile.setIcon(moleIcon); // IMAGE INSIDE TILE  
} // Each button represents one tile on the game board.  
// also clickable
```



now as you can see, image is not setted ,but; image is not scaled...

## Image Files

The game uses two image files:

- `piranha.png` - Piranha Plant image
- `monty.png` - Mole image

put it in src for direct access ease

```
bali-king@war-machine:~/BaliGit/whack-a-mole-java/src$ tree
.
├── App.class
```

```
|— App.java
|— monty2.png
|— monty.png
|— piranha.png
|— WhackAMole.class
|— WhackAMole.java
```

1 directory, 7 files

bali-king@war-machine:~/BaliGit/whack-a-mole-java/src\$

## Loading and Scaling Images

The images must be scaled to fit the buttons (150x150 pixels). The process involves two steps:

```
// Load mole image
Image moleImage = new
ImageIcon(getClass().getResource("./monty.png")).getImage();
moleIcon = new ImageIcon(moleImage.getScaledInstance(150, 150,
Image.SCALE_SMOOTH));

// Load plant image
Image plantImage = new
ImageIcon(getClass().getResource("./piranha.png")).getImage();
plantIcon = new ImageIcon(plantImage.getScaledInstance(150, 150,
Image.SCALE_SMOOTH));
```

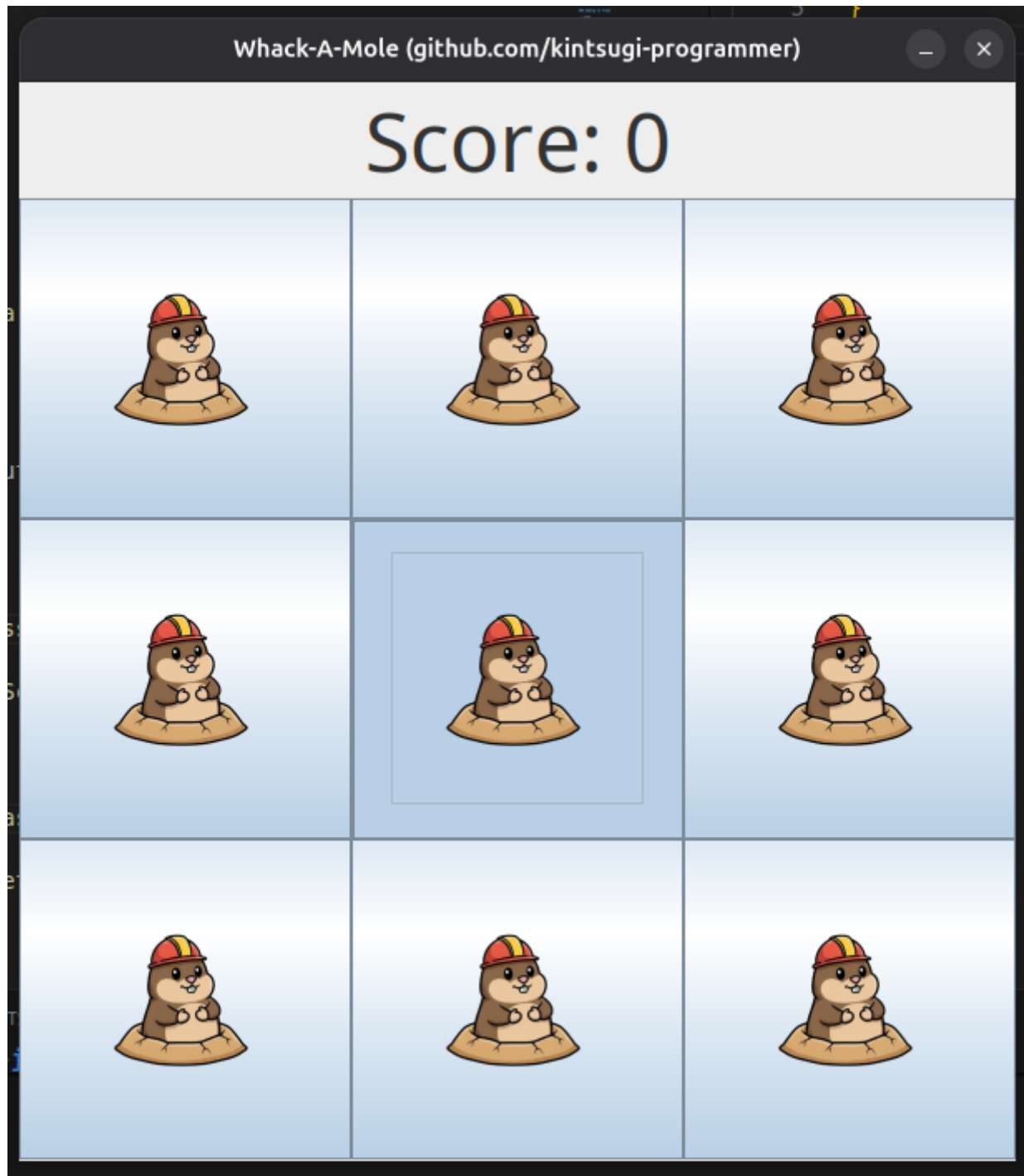
## Explanation of Image Loading Process

1. **getClass().getResource()** - Loads the image file from the source folder
2. **getImage()** - Extracts the actual image from the ImageIcon
3. **getScaledInstance(150, 150, Image.SCALE\_SMOOTH)** - Resizes image to 150x150 pixels with smooth scaling
4. **new ImageIcon()** - Converts scaled image back to ImageIcon for button display

## Why Two-Step Process?

- Cannot directly set ImageIcon from file path with scaling
- Must first get the image, then scale it, then create ImageIcon from scaled image

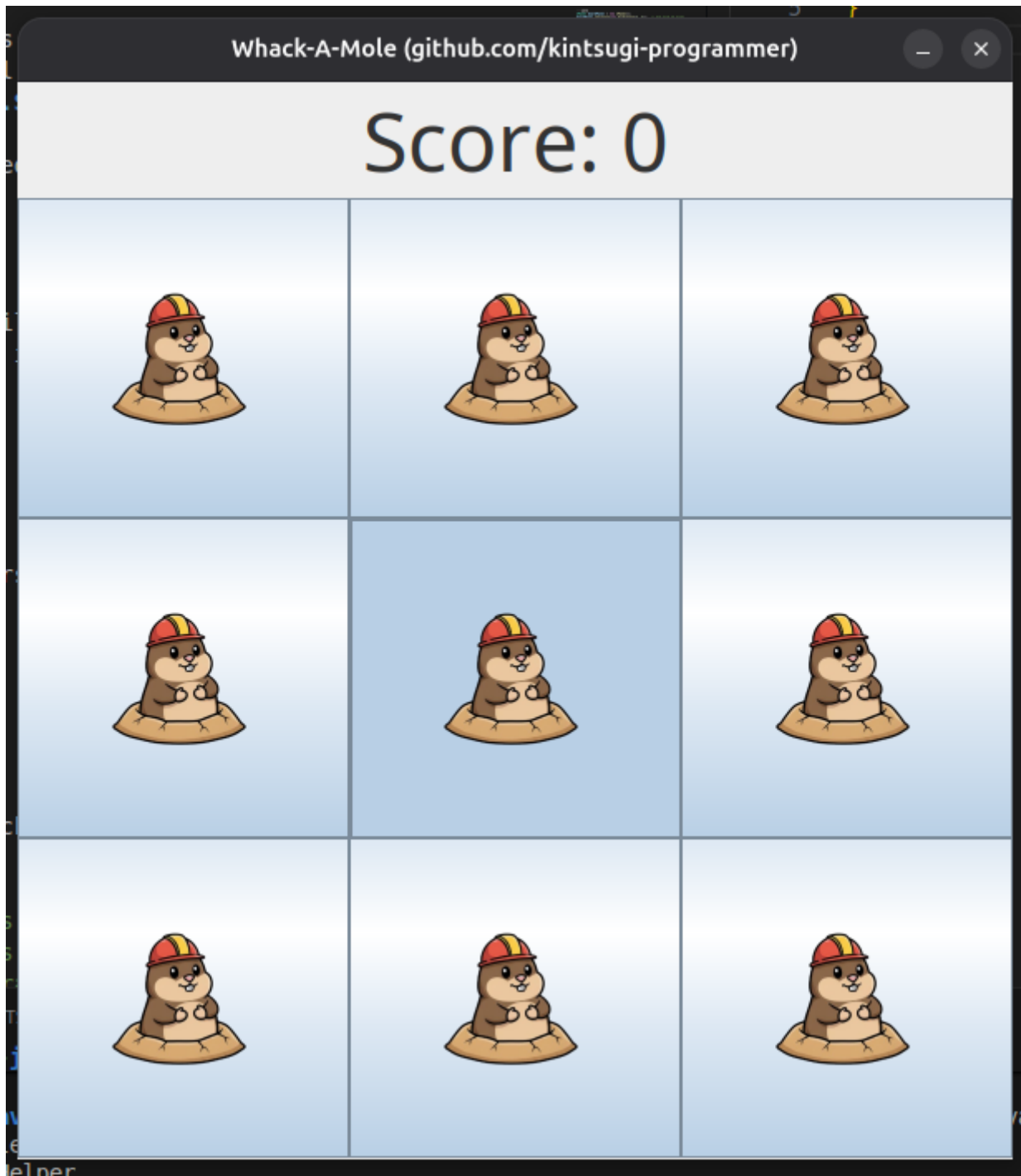




## Removing Button Focus Rectangle

Disable the focus rectangle that appears when buttons are clicked:

```
tile.setFocusable(false);
```



## Variables for Game State

Create these instance variables to track game state:

```
 JButton currentMoleTile;           // Tracks which button currently has the mole
 JButton currentPlantTile;         // Tracks which button currently has the
 plant
 Random random = new Random();     // For random tile selection
 Timer setMoleTimer;               // Timer to move mole to new tiles
 Timer setPlantTimer;              // Timer to move plant to new tiles
 int score = 0;                   // Tracks player score
```

## Moving the Mole

## Mole Timer Setup

Create a timer that moves the mole every 1000 milliseconds (1 second):

```
setMoleTimer = new Timer(1000, new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Remove mole from current tile
        if (currentMoleTile != null) {
            currentMoleTile.setIcon(null); // Remove image from button
            currentMoleTile = null;
        }

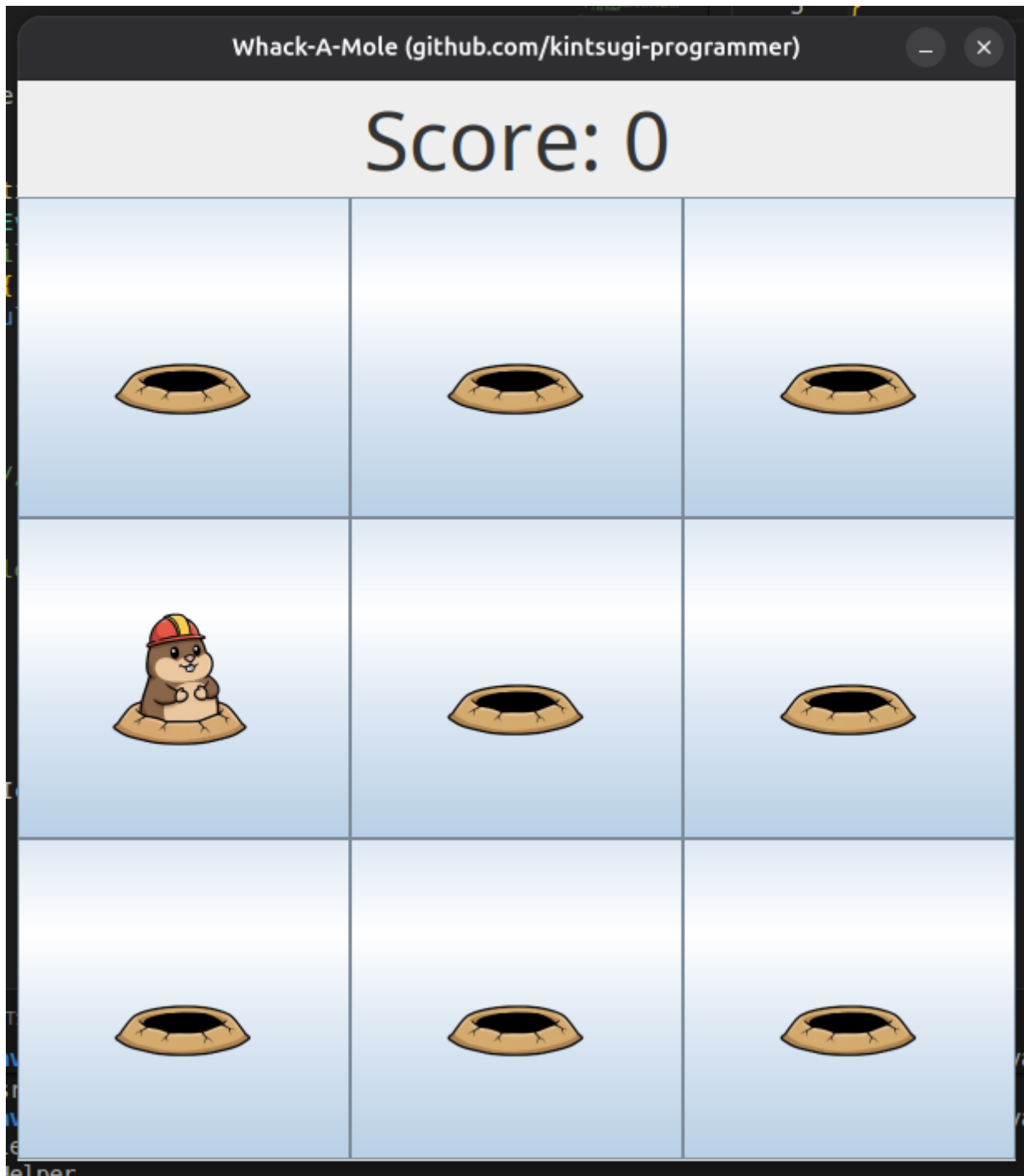
        // // normal way, but doesn't check if other object also have same
        // tile or not
        // currentMoleTile = board[random.nextInt(9)];
        // currentMoleTile.setIcon(moleIcon);

        // new way, makes objects of random number and check if plant
        // already is at tile
        // Select random tile
        int num = random.nextInt(9); // Random number 0-8
        JButton tile = board[num];

        // Check if plant is on this tile (avoid conflict)
        if (currentPlantTile == tile) {
            return; // Skip this turn
        }

        // Place mole on new tile
        currentMoleTile = tile;
        currentMoleTile.setIcon(moleIcon);
    }
});

setMoleTimer.start();
```



## Why Use Intermediate Variable?

The code uses:

```
JButton tile = board[num];
```

Instead of:

```
currentMoleTile = board[num];
```

This is because there's an edge case that must be covered (conflict checking with plant tile).

# Moving the Plant

## Plant Timer Setup

Create a timer that moves the plant every 1500 milliseconds (1.5 seconds):

```
setPlantTimer = new Timer(1500, new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Remove plant from current tile
        if (currentPlantTile != null) {
            currentPlantTile.setIcon(null); // Remove image from button
            currentPlantTile = null;
        }

        // Select random tile
        int num = random.nextInt(9); // Random number 0-8
        JButton tile = board[num];

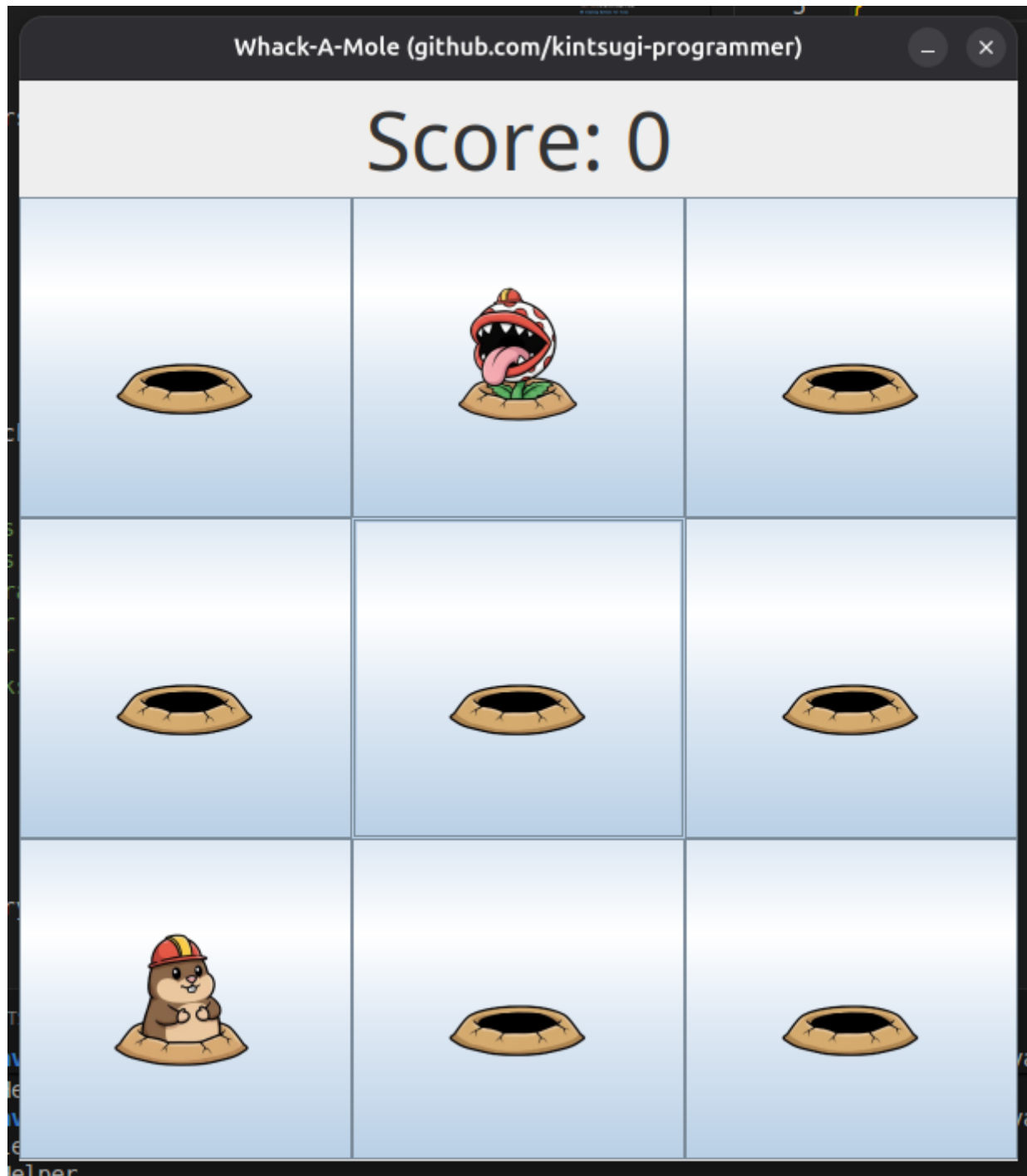
        // Check if mole is on this tile (avoid conflict)
        if (currentMoleTile == tile) {
            return; // Skip this turn
        }

        // Place plant on new tile
        currentPlantTile = tile;
        currentPlantTile.setIcon(plantIcon);
    }
});

setPlantTimer.start();
```

## Why Different Timer Values?

- Mole: 1000 milliseconds (moves every second)
- Plant: 1500 milliseconds (moves every 1.5 seconds)
- This creates variety in movement patterns



## Handling Tile Conflicts

### The Conflict Problem

Since there are only 9 tiles and two separate timers randomly placing mole and plant:

- High chance both timers select the same tile
- One icon overwrites the other
- Only one image displays on the shared tile
- Game logic becomes inconsistent

### Solution: Tile Occupancy Check

When placing mole or plant, check if the target tile is occupied:

**In Mole Timer:**

```
if (currentPlantTile == tile) {  
    return; // Skip this turn if plant is here  
}
```

**In Plant Timer:**

```
if (currentMoleTile == tile) {  
    return; // Skip this turn if mole is here  
}
```

**How It Works**

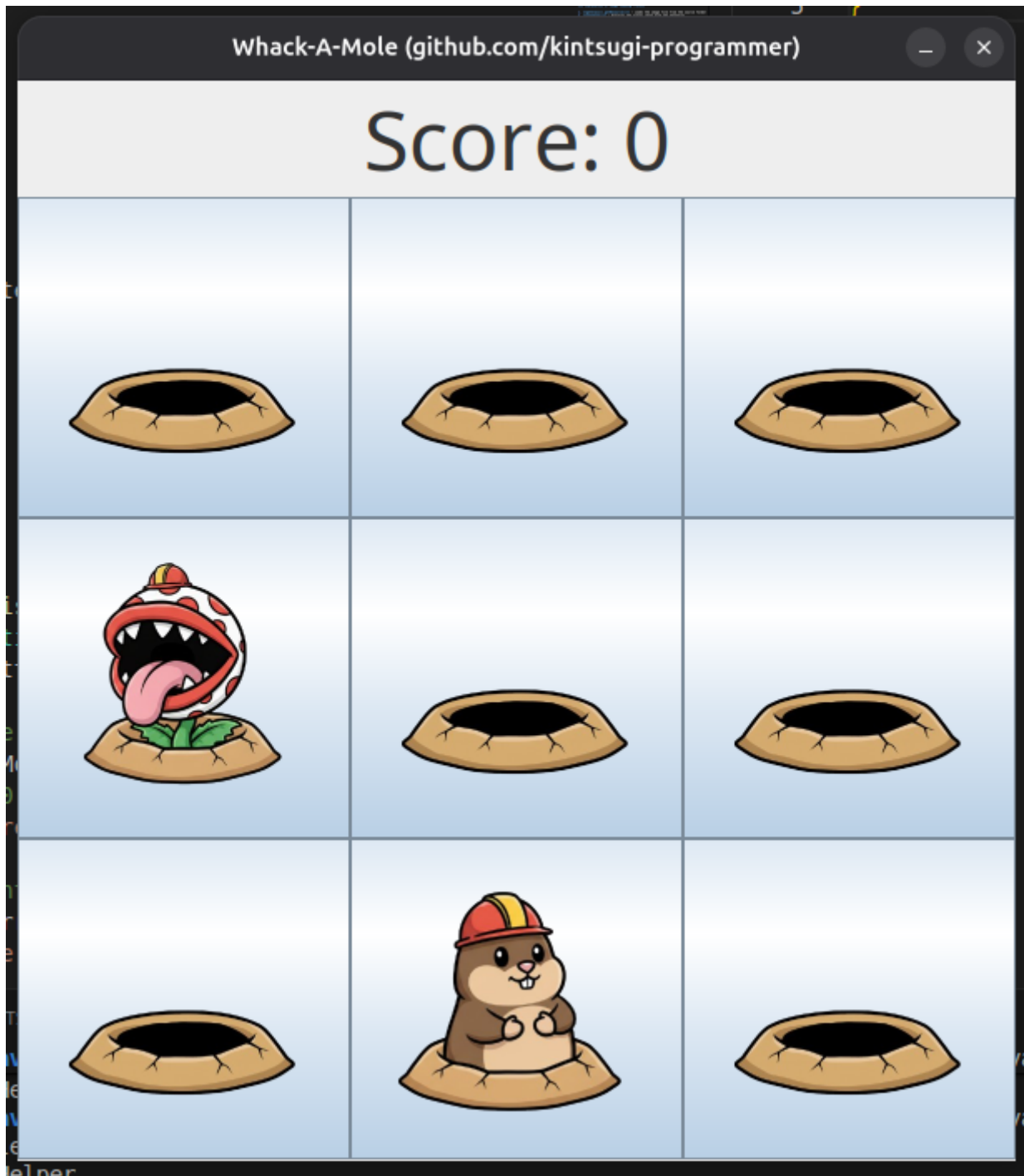
1. If selected tile already has mole or plant, return early
2. The current icon remains on that tile
3. No overwriting occurs
4. Game logic remains consistent







after resizing to make pleasing



## Detecting Button Clicks

### Adding Click Listeners to Tiles

In the for loop where buttons are created, add an action listener to each button:

```
for (int i = 0; i < 9; i++) {  
    JButton tile = new JButton();  
    board[i] = tile;  
    boardPanel.add(tile);  
  
    // Add click handler  
    tile.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent e) {
```

```

        JButton clickedTile = (JButton) e.getSource();// input user
        click & typecast that input to button

        // Check if clicked on mole
        if (clickedTile == currentMoleTile) {
            score += 10; // Add 10 points
            textLabel.setText("Score: " + Integer.toString(score));
        }
        // Check if clicked on plant
        else if (clickedTile == currentPlantTile) {
            textLabel.setText("Game Over");

            // Stop movement
            setMoleTimer.stop();
            setPlantTimer.stop();

            // Disable all buttons
            for (int i = 0; i < 9; i++) {
                board[i].setEnabled(false);
            }
        }
    }
});
}

```

## Type Casting

```
JButton clickedTile = (JButton) e.getSource();
```

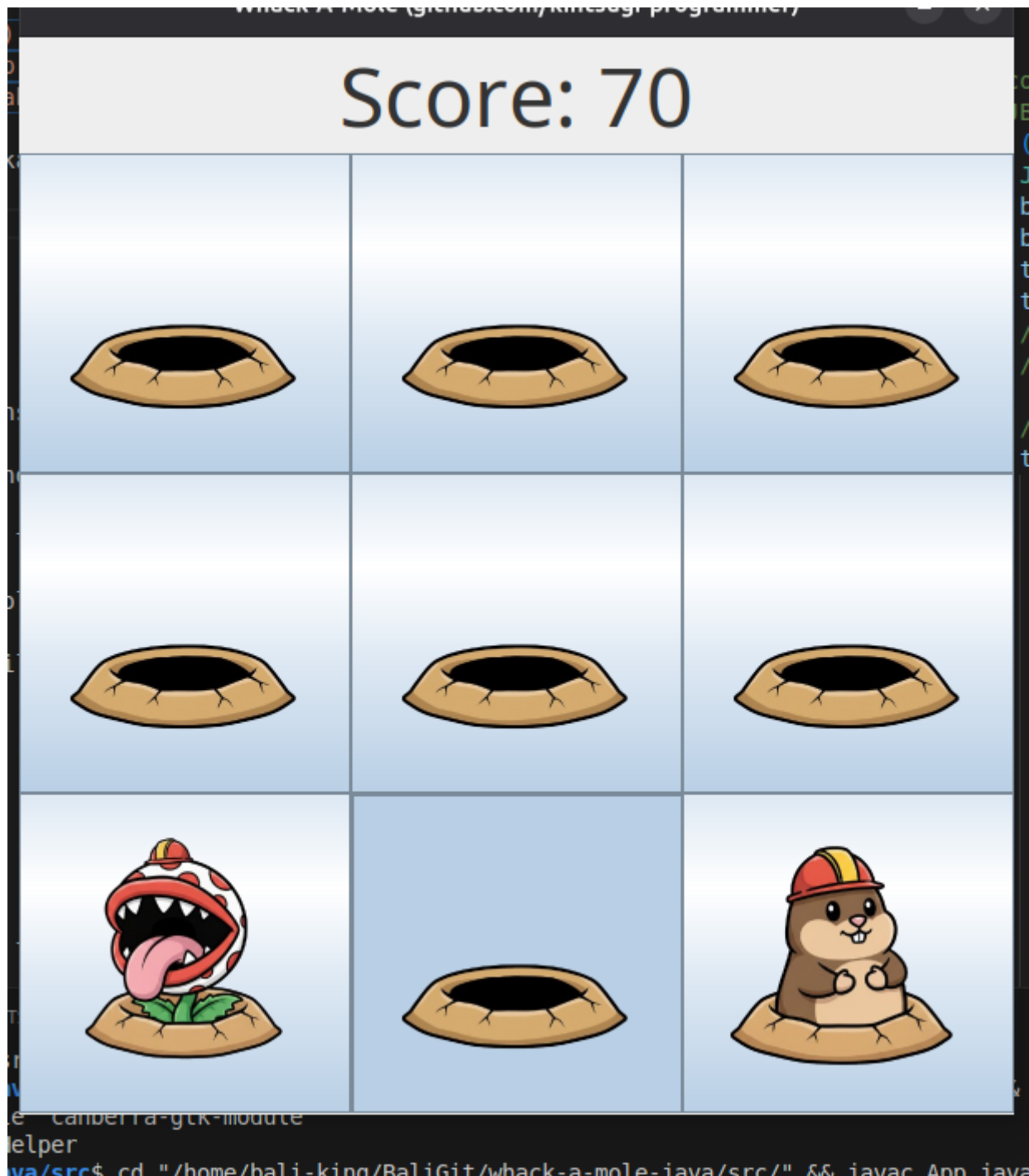
- `e.getSource()` returns an Object
- Must cast to JButton to access button-specific methods

## Mole Click Behavior

- Increment score by 10 points
- Update score text display
- Mole continues moving (no stopping)
- Player can continue playing

## Plant Click Behavior

- Display "Game Over" text
- Stop mole timer: `setMoleTimer.stop()`
- Stop plant timer: `setPlantTimer.stop()`
- Disable all buttons: `board[i].setEnabled(false)`
- Mole and plant stop moving
- Buttons become grayed out and unclickable



## Complete Game Flow

### Game Sequence

1. Window opens with 3x3 grid of buttons
2. Score displays at top as "Score: 0"
3. Mole randomly appears on one tile and moves every 1 second
4. Plant randomly appears on different tile and moves every 1.5 seconds
5. Player clicks on tiles containing mole to earn points (+10 each)
6. Player tries to avoid clicking on tile with plant
7. Clicking plant triggers "Game Over"
8. When game ends:
  - Timers stop
  - All buttons disabled

- Text changes to "Game Over"

## Important Implementation Details

- Mole and plant can never occupy same tile (prevented by checks)
- Score updates only when mole is clicked
- Game cannot be played after "Game Over" unless program is restarted
- Focus rectangles removed from buttons for cleaner appearance
- All components load before window becomes visible

## Enhancements

The game improved with these features:

### Enhancement 1: Hole instead of null spaces

```
ImageIcon holeIcon; // before constructor

// Inside constructor
Image holeImage = new
ImageIcon(getClass().getResource("./hole.png")).getImage();
holeIcon = new ImageIcon(holeImage.getScaledInstance(150, 150,
Image.SCALE_SMOOTH));
```

then replace icon tiles `null` with `holeIcon`

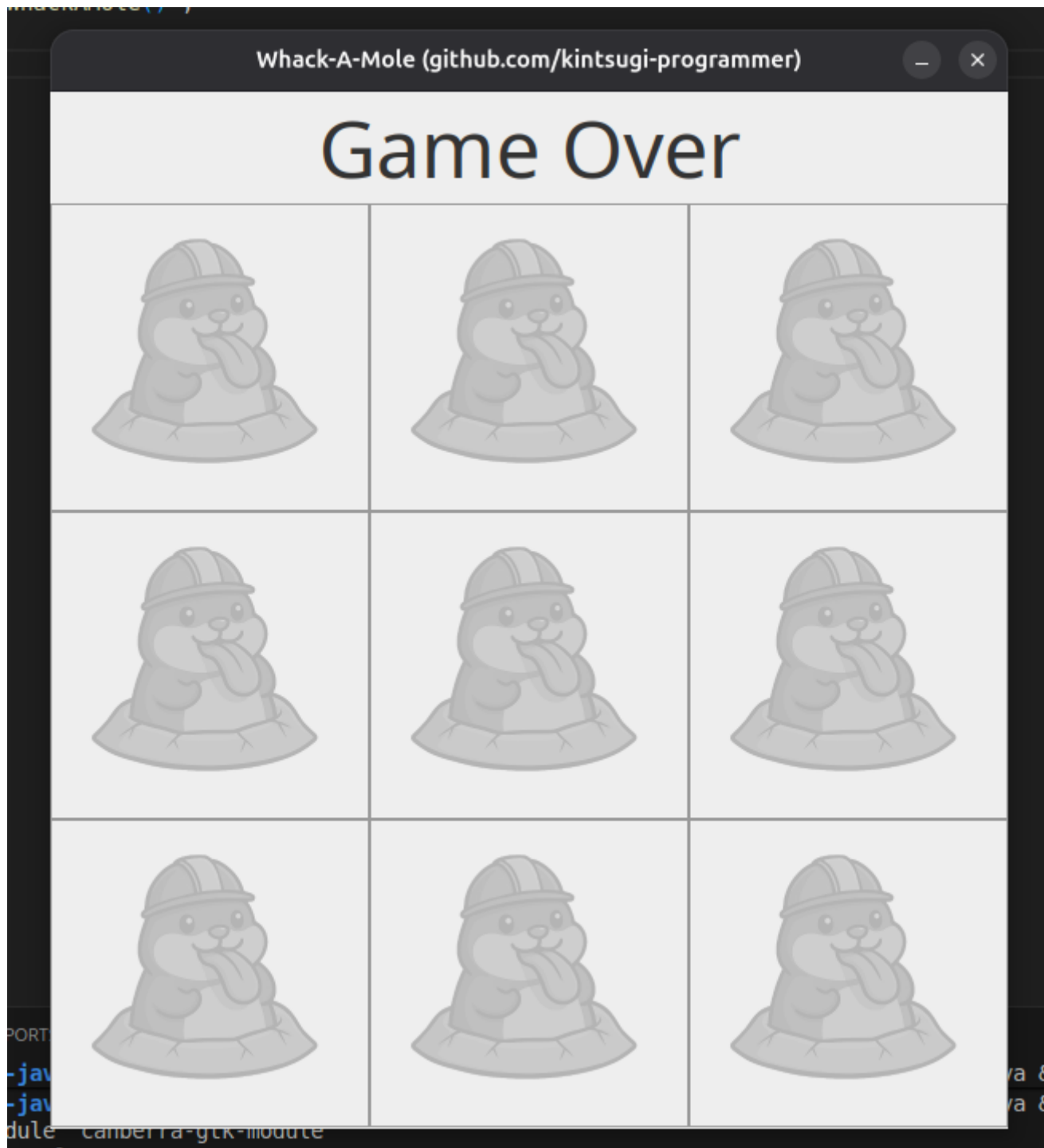
### Enhancement 2: GameOver Mole Teasing Buttons

```
ImageIcon molemockingIcon;
```

```
// inside constructor
Image molemockingImage = new
ImageIcon(getClass().getResource("./montymocking.png")).getImage();

molemockingIcon = new ImageIcon(molemockingImage.getScaledInstance(250,
250, Image.SCALE_SMOOTH));
```

```
// inside gameover logic
// Disable all buttons
for (int i = 0; i<9; i++){
    board[i].setIcon(molemockingIcon);
    board[i].setEnabled(false);
}
```



### Enhancement 3: Second Piranha Plant

This enhancement adds a **second independent Piranha Plant** to the game, increasing difficulty without changing the core mechanics.

#### Implementation details:

- A new variable `currentPlantTile2` tracks the second plant's position.
- A second timer `setPlantTimer2` controls the movement of the second plant.
- Each plant:
  - Moves on its own timer interval.
  - Cannot occupy the same tile as the mole.
  - Cannot occupy the same tile as the other plant.
- Clicking **either** plant immediately ends the game.

### Gameplay impact:

- Players must now track two hazards instead of one.
- Different timer intervals create unpredictable movement patterns.
- The difficulty increases naturally while keeping the implementation simple and readable.



### Enhancement 4: Reset Button

This enhancement adds a **Reset** button to the top bar, to the **right of the score**. The player can restart the game at any time without closing and reopening the program.

### UI Changes

- A `JButton` `resetButton` is added next to the score label.

- The top `textPanel` uses `BorderLayout`, with:
  - `textLabel` (Score: X) in `BorderLayout.CENTER`
  - `resetButton` in `BorderLayout.EAST`

This keeps the UI compact and visually aligned with the score.

## Logic Changes

A new method `resetGame()` is introduced, which:

1. **Stops all timers** (`setMoleTimer`, `setPlantTimer`, `setPlantTimer2`) to avoid movement during the reset.
2. **Resets the score** back to `0` and updates the text label to `"Score: 0"`.
3. **Clears game state**, by setting:
  - `currentMoleTile = null`
  - `currentPlantTile = null`
  - `currentPlantTile2 = null`
4. **Restores the board** by looping over all 9 buttons:
  - Setting their icon back to the `holeIcon`
  - Re-enabling them in case they were disabled after "Game Over".
5. **Restarts all timers** so the mole and plants begin moving again from a clean state.

```
// new, inside cons.
// constructing Reset Button
JButton resetButton = new JButton("Reset");
resetButton.setFont(new Font("Arial",Font.PLAIN,20));
resetButton.setFocusable(false);

resetButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e){
        resetGame();
    }
});
```

```
// tweak
// constructing textPanel to hold textLabel
// and making frame to hold textPanel
textPanel.setLayout(new BorderLayout());
textPanel.add(textLabel, BorderLayout.CENTER); // score at center
textPanel.add(resetButton, BorderLayout.EAST); // right side reset
button
frame.add(textPanel,BorderLayout.NORTH);
```



```
// new outside cons.
private void resetGame(){
    // stop timers safely check
    setMoleTimer.stop();
    setPlantTimer.stop();
    setPlantTimer2.stop();

    // reset score
    score = 0;
    textLabel.setText("Score: 0");

    // clear board, for new start
    for (int i=0; i<9; i++){
        board[i].setIcon(holeIcon);
        board[i].setEnabled(true);
    }

    // clear reference
    currentMoleTile = null;
    currentPlantTile = null;
    currentPlantTile2 = null;

    // restart timers
    setMoleTimer.start();
    setPlantTimer.start();
    setPlantTimer2.start();

}
```

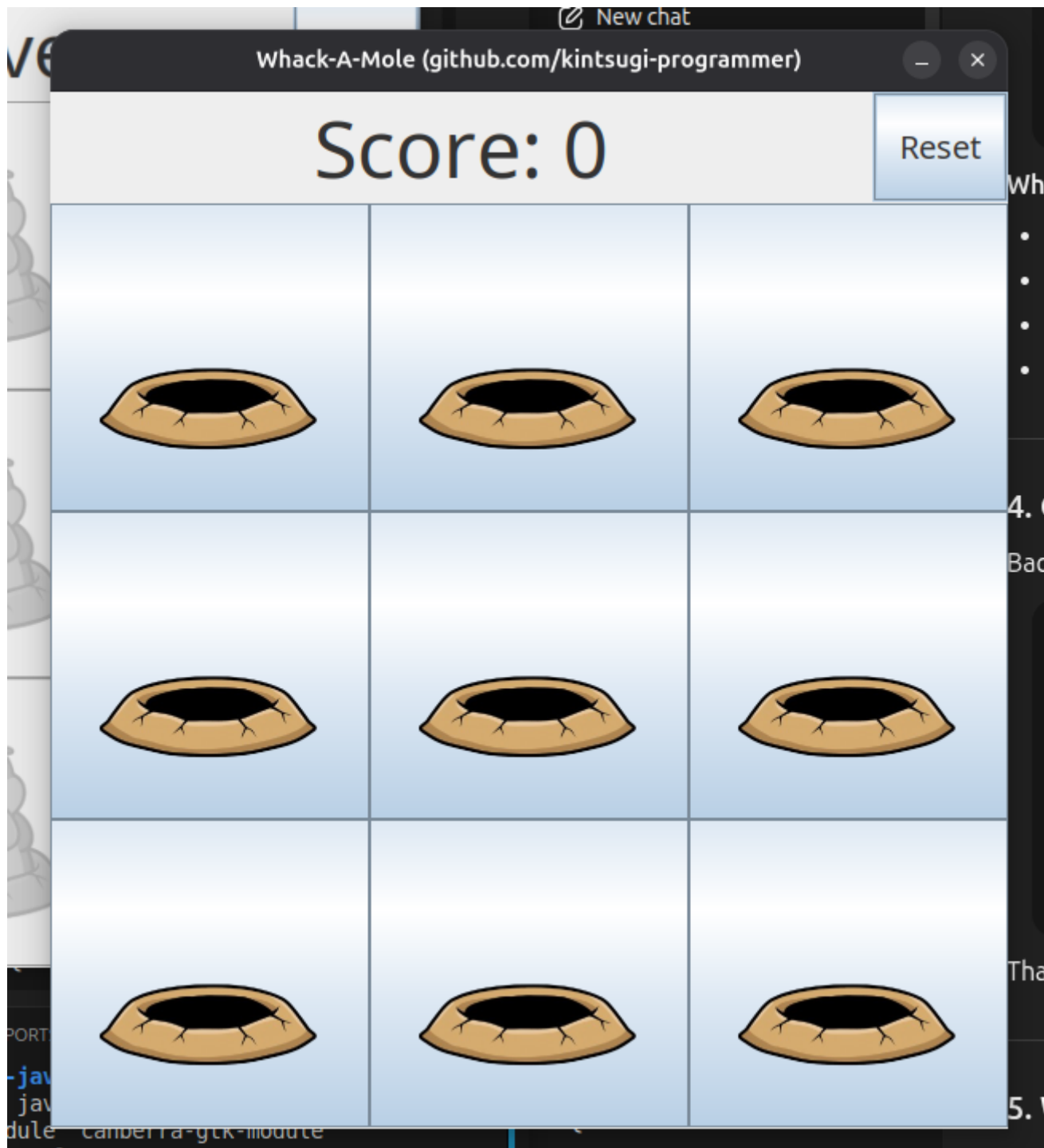
## Player Experience

- After a **Game Over**, instead of restarting the application, the player can simply click **Reset** to start a new round.
- The Reset button also works mid-game, instantly restarting the board and score.
- The mocking game-over state (all tiles showing the teasing mole) is cleared on reset, returning the board to fresh hole tiles.









### Enhancement 5: High Score Tracking

- Create variable to track all-time high score
- Display high score on screen
- Compare current score with high score
- Persist high score between games

```
// outside constructor  
int score = 0, highScore=0;
```

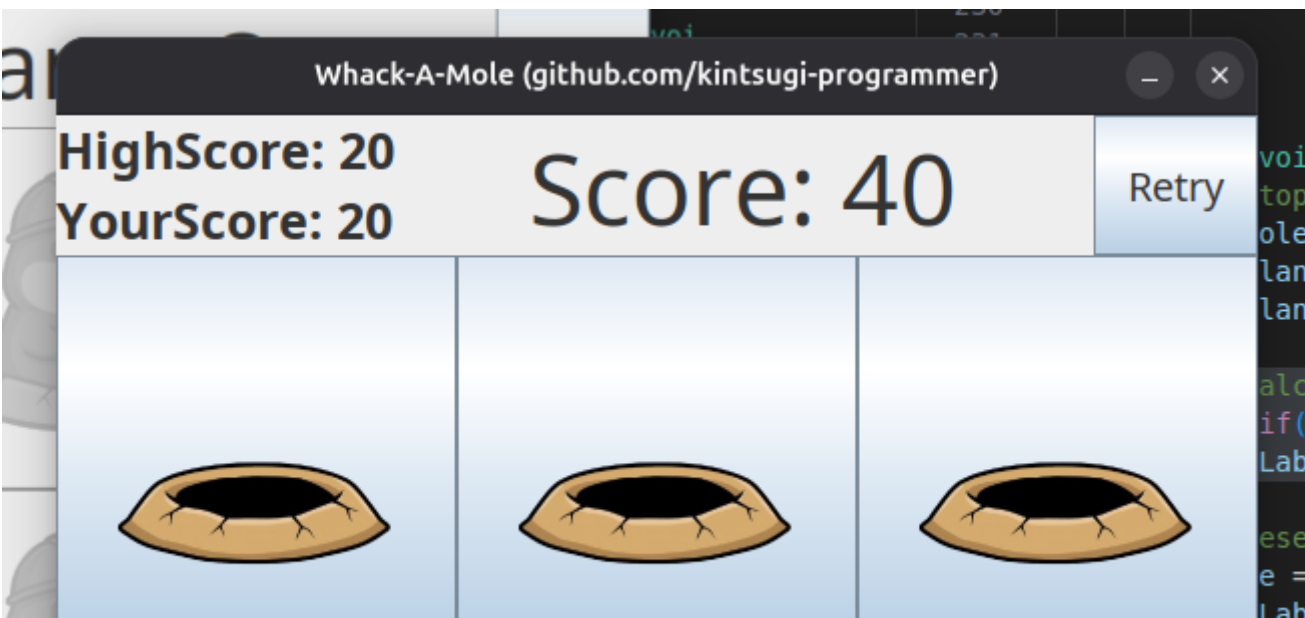
```
// inside constructor
    // constructing textLabel2
    textLabel2.setFont(new Font("Arial",Font.BOLD,25)); // Set Font to
Label, Arial Font , Plain 50px
    textLabel2.setHorizontalAlignment(JLabel.CENTER); // center text
auto
    textLabel2.setText("<html>"+ "HighScore:
"+Integer.toString(highScore)+"<br>YourScore: 0"+"</html>");
    textLabel2.setOpaque(true);
```

```
// inside constructor
    // constructing textPanel to hold textLabel
    // and making frame to hold textPanel
    textPanel.setLayout(new BorderLayout());
    textPanel.add(textLabel, BorderLayout.CENTER); // score at center
    textPanel.add(textLabel2, BorderLayout.WEST); // score at center
    textPanel.add(resetButton, BorderLayout.EAST); // right side reset
button
    frame.add(textPanel,BorderLayout.NORTH);
```

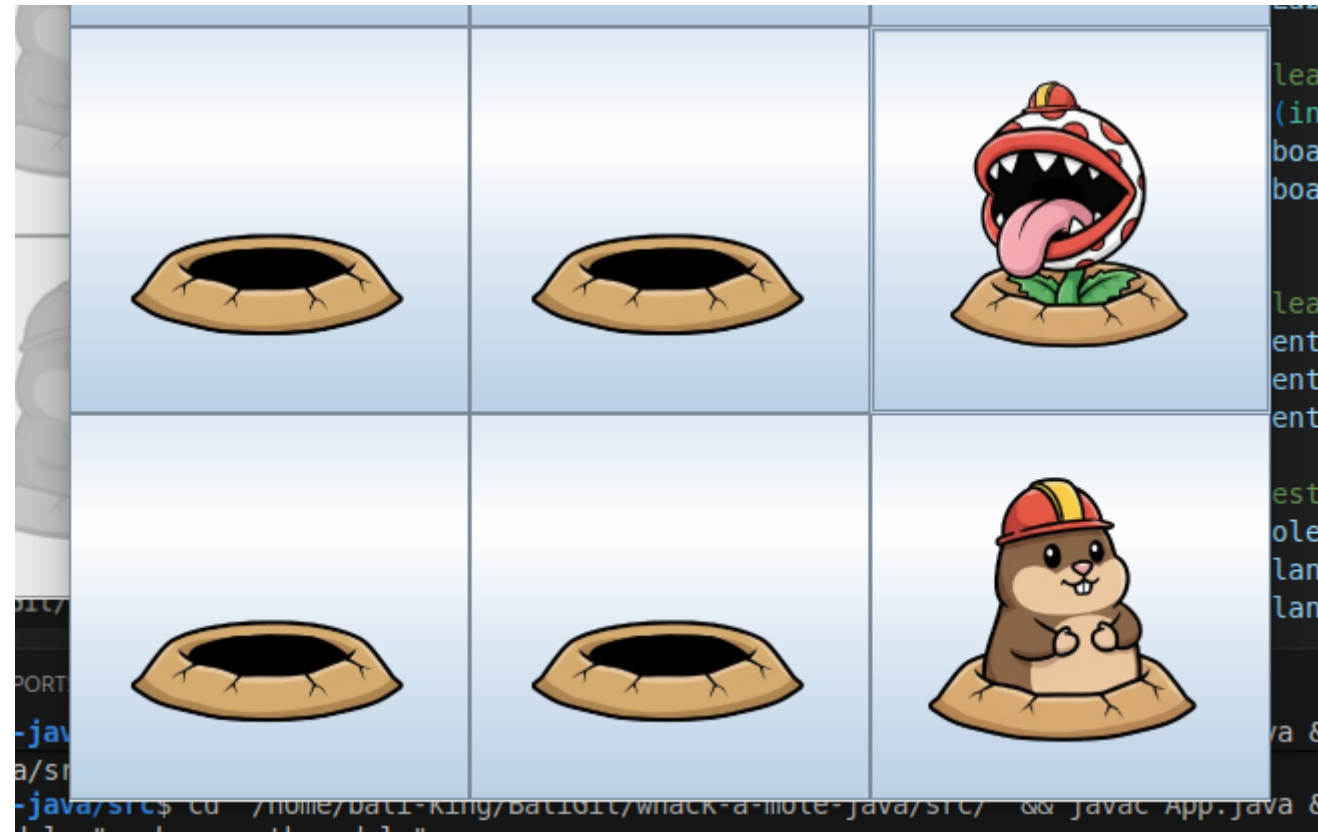
```
// at game over logic
                                // calculate and update highscore
                                if(highScore<score){highScore=score;}
                                textLabel2.setText("<html>"+ "HighScore:
"+Integer.toString(highScore)+"<br>YourScore: "+score+ "</html>");
```

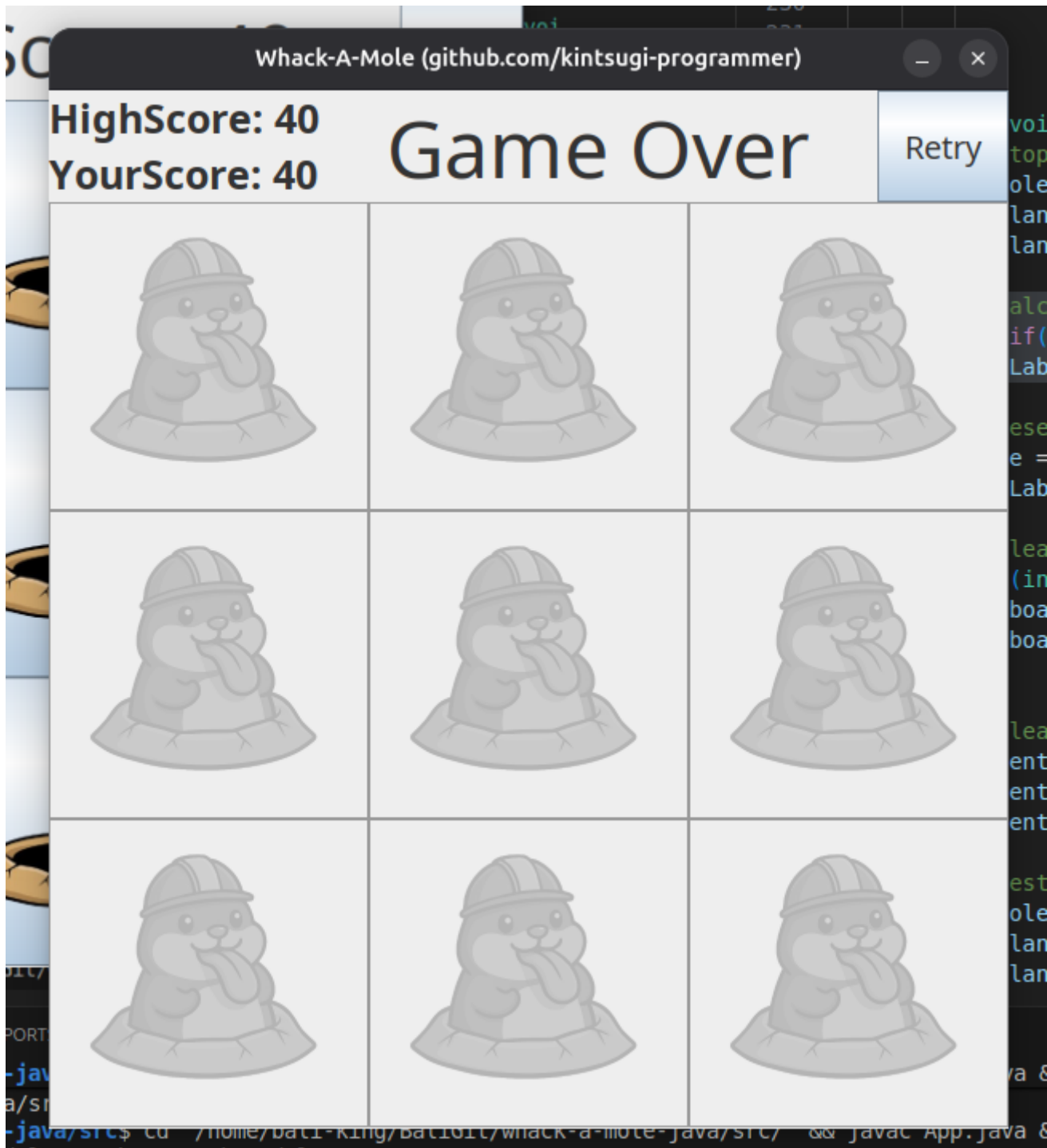
```
// at resetGame()
    // calculate and update highscore before reset score
    if(highScore<score){highScore=score;}
    textLabel2.setText("<html>"+ "HighScore:
"+Integer.toString(highScore)+"<br>YourScore: "+score+ "</html>");
```











### Enhancement 6: Welcome Screen with Play Button

This enhancement introduces a **dedicated welcome screen** that appears when the game is launched, providing a polished entry point before gameplay begins.

```
private void showWelcomeScreen() {  
    // true = modal; blocks until closed  
    JDialog dialog = new JDialog(frame, "Welcome to Whack-A-Mole!", true);  
    dialog.setSize(350, 400);  
    dialog.setLocationRelativeTo(frame);  
    dialog.setLayout(new BorderLayout());  
  
    // Image
```

```
        ImageIcon welcomeRaw = new
ImageIcon(getClass().getResource("./monty2.png"));
        Image welcomeImg = welcomeRaw.getImage().getScaledInstance(250, 250,
Image.SCALE_SMOOTH);
        JLabel imageLabel = new JLabel(new ImageIcon(welcomeImg));
        imageLabel.setHorizontalAlignment(SwingConstants.CENTER);
        // Message label
        JLabel welcomeLabel = new JLabel(
            "<html><center>Welcome to Whack-A-Mole!<br>" +
            "Click PLAY to start the game.<br>" +
            "Hit the mole, avoid the plants!</center></html>",
            SwingConstants.CENTER
        );
        welcomeLabel.setFont(new Font("Arial", Font.PLAIN, 16));
        dialog.add(imageLabel, BorderLayout.NORTH);
        dialog.add(welcomeLabel, BorderLayout.CENTER);

        // Play button
        JButton playButton = new JButton("Play");
        playButton.setFont(new Font("Arial", Font.BOLD, 18));
        playButton.setFocusable(false);

        playButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Start the game when user clicks Play
                startGame();
                dialog.dispose();
            }
        });

        // Optional: Quit button
        JButton quitButton = new JButton("Quit");
        quitButton.setFont(new Font("Arial", Font.PLAIN, 14));
        quitButton.setFocusable(false);
        quitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });

        JPanel buttonPanel = new JPanel();
        buttonPanel.add(playButton);
        buttonPanel.add(quitButton);
        dialog.add(buttonPanel, BorderLayout.SOUTH);

        dialog.setResizable(false);
        dialog.setVisible(true);
    }private void startGame() {
        setMoleTimer.start();
        setPlantTimer.start();
        setPlantTimer2.start();
    }
}
```

```
// at cons. end

        // commented because of welcome game logic
        // setMoleTimer.start();
        // setPlantTimer.start();
        // setPlantTimer2.start();

        // frame.setVisible(true); // visibility, only after loading
everything    frame.setVisible(true); // show main window first
              showWelcomeScreen(); // then show modal welcome + Play
button
```

## Purpose

Previously, the game started immediately upon launch. The welcome screen improves user experience by:

- Giving players a clear starting point.
- Explaining basic gameplay rules.
- Preventing the game from running before the player is ready.

## Welcome Screen Design

- A modal **welcome dialog** is displayed on top of the main game window.
- The screen includes:
  - A game title (“Whack-A-Mole”).
  - A centered **image** (mole or custom welcome artwork).
  - A short description explaining the objective.
  - A **Play** button to begin the game.

The dialog is modal, meaning the player must interact with it before accessing gameplay.

## Play Button Behavior

- Game timers (mole and Piranha Plants) are **not started automatically** when the application launches.
- When the **Play** button is clicked:
  1. The welcome screen is closed.
  2. The game state is reset (board cleared, score reset).
  3. All movement timers are started, beginning gameplay.

This ensures the game always starts in a clean, predictable state.

## Technical Implementation

- A `JDialog` is used for the welcome screen.
- Images are displayed using `JLabel` with a scaled `ImageIcon`.
- Timer initialization is decoupled from the constructor and triggered only after user interaction via a dedicated `startGame()` method.

## Player Experience

- Players are welcomed with a visually engaging introduction.
- Gameplay begins only after an explicit action, making the game feel more complete and intentional.
- The welcome screen enhances accessibility, clarity, and overall polish.

This enhancement transforms the project from a technical demo into a more complete, user-friendly desktop game.



# Complete Code Structure

## Class Variables (Instance Variables)

```
int boardWidth = 600;
int boardHeight = 650;
JFrame frame;
JPanel textPanel;
JLabel textLabel;
JPanel boardPanel;
JButton[] board;
ImageIcon moleIcon;
ImageIcon plantIcon;
JButton currentMoleTile;
JButton currentPlantTile;
Random random;
Timer setMoleTimer;
Timer setPlantTimer;
int score;
```

## Constructor Method

Contains all initialization code:

- Frame setup
- Component creation
- Image loading and scaling
- Button creation with event listeners
- Timer setup with movement logic
- Frame visibility set at end

## Key Concepts Learned

### GUI Components

- JFrame: Main window container
- JPanel: Layout containers (text panel, board panel)
- JButton: Interactive tiles
- JLabel: Text display

### Layouts

- BorderLayout: For overall window layout (North for text panel)
- GridLayout: For 3x3 button grid

### Event Handling

- ActionListener: Handles button clicks
- Timer: Handles repeating events (mole/plant movement)

- ActionEvent: Contains event information

## Image Processing

- Loading images from files using getResource()
- Scaling images to specific dimensions
- Converting images to ImageIcon for display
- Using ImageIcon objects on buttons

## Game Logic

- State tracking (current mole/plant position, score)
- Conflict prevention (checking tile occupancy)
- Game flow control (stopping timers, disabling buttons)
- User feedback (updating score display)

## Final Codes

src/App.java

```
public class App {  
    public static void main(String[] args) throws Exception {  
        WhackAMole whackAMole = new WhackAMole() ;  
    }  
}
```

src/WhackAMole.java

```
// Import Packages  
import java.awt.*;  
import java.awt.event.*;  
import java.util.Random;  
import java.util.random.*;  
import javax.swing.*;  
  
public class WhackAMole {  
    // Build Board  
    int boardWidth= 600;  
    int boardHeight= 650;  
    JFrame frame = new JFrame("Whack-A-Mole (github.com/kintsugi-programmer)");// frame  
  
    JLabel textLabel = new JLabel(); // text label  
    JPanel textPanel = new JPanel(); // text panel to hold text label  
    JLabel textLabel2 = new JLabel(); // text label  
    JPanel textPanel2 = new JPanel(); // text panel to hold text label  
    JButton[] board = new JButton[9]; // make button array for tracking all  
    9 buttons efficiently
```

```

    ImageIcon moleIcon;
    ImageIcon plantIcon;
    ImageIcon holeIcon;
    ImageIcon molemockingIcon;

    // game stats vars
    JButton currentMoleTile;
    JButton currentPlantTile,currentPlantTile2;
    Random random = new Random();
    Timer setMoleTimer;
    Timer setPlantTimer,setPlantTimer2;
    int score = 0,highScore=0;

    // Constructor
    WhackAMole(){

        // constructing frame
        // frame.setVisible(true);// visibility
        frame.setSize(boardWidth, boardHeight);// set size
        frame.setLocationRelativeTo(null);// make sure to open at center
        frame.setResizable(false);// no resize
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);// Close (X)
        Button of program,exit it
        frame.setLayout(new BorderLayout());

        // constructing textLabel
        textLabel.setFont(new Font("Arial",Font.PLAIN,50));// Set Font to
        Label, Arial Font , Plain 50px
        textLabel.setHorizontalAlignment(JLabel.CENTER);// center text auto
        textLabel.setText("Score: 0");// set text as score 0 default
        textLabel.setOpaque(true);

        // constructing textLabel2
        textLabel2.setFont(new Font("Arial",Font.BOLD,25));// Set Font to
        Label, Arial Font , Plain 50px
        textLabel2.setHorizontalAlignment(JLabel.CENTER);// center text
        auto
        textLabel2.setText("<html>"+ "HighScore:
"+Integer.toString(highScore)+"<br>YourScore: 0"+"</html>");
        textLabel2.setOpaque(true);

        // constructing Reset Button
        JButton resetButton = new JButton("Retry");
        resetButton.setFont(new Font("Arial",Font.PLAIN,20));
        resetButton.setFocusable(false);

        resetButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e){
                resetGame();
            }
        });

        // constructing textPanel to hold textLabel

```



```

        // and making frame to hold textPanel
        textPanel.setLayout(new BorderLayout());
        textPanel.add(textLabel, BorderLayout.CENTER); // score at center
        textPanel.add(textLabel2, BorderLayout.WEST); // score at center
        textPanel.add(resetButton, BorderLayout.EAST); // right side reset
button
        frame.add(textPanel, BorderLayout.NORTH);

        // constructing board panel
        JPanel boardPanel = new JPanel();
        boardPanel.setLayout(new GridLayout(3, 3)); // 3x3 Grid Tiles
        frame.add(boardPanel); // add boardpanel to frame
        // boardPanel.setBackground(Color.BLACK); // Optional: makes
background visible during development

        // Loading and Scaling Images
        // moleIcon = new ImageIcon(getClass().getResource("./monty.png"));
old
        // New Approach
        // Create Image Objects
        Image moleImage = new
        ImageIcon(getClass().getResource("./monty.png")).getImage();
        Image plantImage = new
        ImageIcon(getClass().getResource("./piranha.png")).getImage();
        Image holeImage = new
        ImageIcon(getClass().getResource("./hole.png")).getImage();
        Image molemockingImage = new
        ImageIcon(getClass().getResource("./montymocking.png")).getImage();
        // Create Icon of scaling that Image Object
        moleIcon = new ImageIcon(moleImage.getScaledInstance(250, 250,
        Image.SCALE_SMOOTH));
        plantIcon = new ImageIcon(plantImage.getScaledInstance(250, 250,
        Image.SCALE_SMOOTH));
        holeIcon = new ImageIcon(holeImage.getScaledInstance(250, 250,
        Image.SCALE_SMOOTH));
        molemockingIcon = new
        ImageIcon(molemockingImage.getScaledInstance(250, 250,
        Image.SCALE_SMOOTH));

        // constructing buttons/ tiles
        // JButton button1 = new JButton(); // one button
        for ( int i = 0; i < 9; i++) { // using loop, to create
tiles/buttons; instead of hardcode
        JButton tile = new JButton();
        board[i] = tile;
        boardPanel.add(tile); // appending to boardPanel Grid one-by-
one

        tile.setIcon(holeIcon); // debugging purposes
        tile.setFocusable(false);
        // also clickable
        // Each button represents one tile on the game board.

        // Click Handler Logic

```

```

        tile.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e){
                JButton clickedTile = (JButton) e.getSource(); // input
                user click & typecast that input to button

                // Score Logic, when Click Mole
                if(clickedTile==currentMoleTile){
                    score+=10; // Score Add 10pts
                    textLabel.setText("Score:
"+Integer.toString(score));
                }

                // GameOver Logic, when Click Plant
                else if (clickedTile==currentPlantTile ||
clickedTile==currentPlantTile2){
                    textLabel.setText("Game Over");

                    // calculate and update highscore
                    if(highScore<score){highScore=score;}
                    textLabel2.setText("<html>"+ "HighScore:
"+Integer.toString(highScore)+"<br>YourScore: "+score+ "</html>");

                    // Stop Movement of Objects
                    setMoleTimer.stop();
                    setPlantTimer.stop();
                    setPlantTimer2.stop();

                    // Disable all buttons
                    for (int i = 0; i<9; i++){
                        board[i].setIcon(molemockingIcon);
                        board[i].setEnabled(false);
                    }
                }
            }
        });
    }

    setMoleTimer = new Timer(750, new ActionListener() { // 1 sec, init
        ActionListener
        public void actionPerformed(ActionEvent e){
            // Remove mole from current tile
            if (currentMoleTile != null){ // if button is not null
                currentMoleTile.setIcon(holeIcon); // clear the hole,
                replace plant/mole with hole
                currentMoleTile = null;
            }

            // // old normal way, but doesn't check if other object
            also have same tile or not
            // currentMoleTile = board[random.nextInt(9)];
            // currentMoleTile.setIcon(moleIcon);

```

```

        // new way, makes objects of random number and check if
plant already is at tile
        // Select random tile
        int num = random.nextInt(9); // random 0 to 8
        JButton tile = board[num];

        // Check if plant is on this tile (avoid conflict)
        if (currentPlantTile == tile || currentPlantTile2 == tile)
{return;} // skip

        // Place mole on new tile
        currentMoleTile = tile;
        currentMoleTile.setIcon(moleIcon);

    }
});

    setPlantTimer = new Timer(1000, new ActionListener() { // 1.5 sec,
init ActionListner
        public void actionPerformed(ActionEvent e){
            // Remove mole from current tile
            if (currentPlantTile != null){ // if button is not null
                currentPlantTile.setIcon(holeIcon); // clear the hole,
replace plant/mole with hole
                currentPlantTile = null;
            }

            // // old normal way, but doesn't check if other object
also have same tile or not
            // currentPlantTile = board[random.nextInt(9)];
            // currentPlantTile.setIcon(moleIcon);

            // new way, makes objects of random number and check if
plant already is at tile
            // Select random tile
            int num = random.nextInt(9); // random 0 to 8
            JButton tile= board[num];

            // Check if mole is on this tile (avoid conflict)
            if (currentMoleTile == tile || currentPlantTile2 == tile )
{return;} // skip

            // Place plant on new tile
            currentPlantTile = tile;
            currentPlantTile.setIcon(plantIcon);
        }
    });

    setPlantTimer2 = new Timer(850, new ActionListener() { // 1.5 sec,
init ActionListner
        public void actionPerformed(ActionEvent e){
            // Remove mole from current tile
            if (currentPlantTile2 != null){ // if button is not null

```

```

        currentPlantTile2.setIcon(holeIcon); // clear the hole,
replace plant/mole with hole
        currentPlantTile2 = null;
    }

    // // old normal way, but doesn't check if other object
also have same tile or not
    // currentPlantTile2 = board[random.nextInt(9)];
    // currentPlantTile2.setIcon(moleIcon);

    // new way, makes objects of random number and check if
plant already is at tile
    // Select random tile
    int num = random.nextInt(9); // random 0 to 8
    JButton tile = board[num];

    // Check if mole is on this tile (avoid conflict)
    if (currentMoleTile == tile || currentPlantTile == tile )
{return;} // skip

    // Place plant on new tile
    currentPlantTile2 = tile;
    currentPlantTile2.setIcon(plantIcon);
}
});

    // commented because of welcome game logic
    // setMoleTimer.start();
    // setPlantTimer.start();
    // setPlantTimer2.start();

    // frame.setVisible(true); // visibility, only after loading
everything
    frame.setVisible(true); // show main window first
    showWelcomeScreen(); // then show modal welcome + Play
button

}

private void resetGame(){
    // stop timers safely check
    setMoleTimer.stop();
    setPlantTimer.stop();
    setPlantTimer2.stop();

    // calculate and update highscore before reset score
    if(highScore<score){highScore=score;}
    textLabel2.setText("<html>"+HighScore:
"+Integer.toString(highScore)+"<br>YourScore: "+score+ "</html>");

    // reset score
    score = 0;
    textLabel.setText("Score: 0");

```

```
// clear board, for new start
for (int i=0; i<9; i++){
    board[i].setIcon(holeIcon);
    board[i].setEnabled(true);
}

// clear reference
currentMoleTile = null;
currentPlantTile = null;
currentPlantTile2 = null;

// restart timers
setMoleTimer.start();
setPlantTimer.start();
setPlantTimer2.start();

}

private void showWelcomeScreen() {
    // true = modal; blocks until closed
    JDialog dialog = new JDialog(frame, "Welcome to Whack-A-Mole!", true);
    dialog.setSize(350, 400);
    dialog.setLocationRelativeTo(frame);
    dialog.setLayout(new BorderLayout());

    // Image
    ImageIcon welcomeRaw = new
    ImageIcon(getClass().getResource("./monty2.png"));
    Image welcomeImg = welcomeRaw.getImage().getScaledInstance(250, 250,
    Image.SCALE_SMOOTH);
    JLabel imageLabel = new JLabel(new ImageIcon(welcomeImg));
    imageLabel.setHorizontalAlignment(SwingConstants.CENTER);
    // Message label
    JLabel welcomeLabel = new JLabel(
        "<html><center>Welcome to Whack-A-Mole!<br>" +
        "Click PLAY to start the game.<br>" +
        "Hit the mole, avoid the plants!</center></html>",
        SwingConstants.CENTER
    );
    welcomeLabel.setFont(new Font("Arial", Font.PLAIN, 16));
    dialog.add(imageLabel, BorderLayout.NORTH);
    dialog.add(welcomeLabel, BorderLayout.CENTER);

    // Play button
    JButton playButton = new JButton("Play");
    playButton.setFont(new Font("Arial", Font.BOLD, 18));
    playButton.setFocusable(false);

    playButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            // Start the game when user clicks Play
            startGame();
        }
    });
}
```

```
        dialog.dispose();
    }
});

// Optional: Quit button
JButton quitButton = new JButton("Quit");
quitButton.setFont(new Font("Arial", Font.PLAIN, 14));
quitButton.setFocusable(false);
quitButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});

JPanel buttonPanel = new JPanel();
buttonPanel.add(playButton);
buttonPanel.add(quitButton);
dialog.add(buttonPanel, BorderLayout.SOUTH);

dialog.setResizable(false);
dialog.setVisible(true);
}private void startGame() {
    setMoleTimer.start();
    setPlantTimer.start();
    setPlantTimer2.start();
}

}
```

## Development Notes

### Build + Export .deb Software Script

```
rm -rf dist && mkdir dist && \
cp build/whackamole.jar dist/ && \
cp src/monty2.png dist/ && \
jpackage \
  --type deb \
  --name whackamole \
  --input dist \
  --main-jar whackamole.jar \
  --main-class App \
  --icon monty2.png \
  --app-version 1.0.0 \
  --vendor "Kintsugi-Programmer"
```

## Install and Run .deb Linux

```
sudo dpkg -i whackamole_1.0.0-1_amd64.deb # adjust name if slightly
different
whackamole
```

```
sudo dpkg -r whackamole # uninstall
```

### Performance Tip

Set `frame.setVisible(true)` at the very end of constructor to ensure all components load before window displays. This prevents UI components from appearing slowly one by one.

### Testing Approach

- Test each component as it's added
- Verify image scaling displays correctly
- Check that mole and plant move independently
- Verify no conflicts occur on same tile
- Test click detection for both mole and plant
- Confirm game ends properly on plant click

### Debugging

- Use `setBackground(Color.BLACK)` temporarily to visualize panel boundaries
- Watch console for any import errors
- Verify image file names match exactly (case-sensitive)
- Check that resource paths use correct file names with extensions

## Conclusion

This Documentation covers all fundamental aspects of creating a GUI game in Java:

- Window and component creation
- Image loading and manipulation
- Event-driven programming with listeners and timers
- Game state management
- User interaction and feedback

The resulting game is fully functional and provides a solid foundation for learning GUI programming concepts.

---

## LICENSE

MIT License

Copyright (c) 2025 Siddhant Bali

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

---

End-of-File

The [KintsugiStack](#) repository, authored by Kintsugi-Programmer, is less a comprehensive resource and more an Artifact of Continuous Research and Deep Inquiry into Computer Science and Software Engineering. It serves as a transparent ledger of the author's relentless pursuit of mastery, from the foundational algorithms to modern full-stack implementation.

Made with  [Kintsugi-Programmer](#)