**HANGMAN GAME**

An

Object-Oriented Programming through Java Course Project Report

in partial fulfilment of the degree

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**by**

MD NABEEL AHMED (2103A51058)

K. NIKHIL (2103A51130)

G. VIGNESH (2103A51353)

**Submitted to**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

This is to certify that the **Object Oriented Programming Concepts through Java Course Project** entitled **“HANGMAN GAME"** is a record of bonafied work carried out by the students(s) **MD NABEEL AHMED, K. NIKHIL, G. VIGNESH** bearing Roll No(s) **2103A51058, 2103A51130, 2103A51353** during the academic year 2023-2024 in the partial fulfilment of the award of the degree of ***Bachelor of Technology in Computer Science & Engineering*** by the SR University, Waranagal.

**Lab In-charge Head of the Department**

**TABLE OF CONTENTS**

**Chapter No. Title Page No.**

Abstract 04

1 **.** Objective of the Project 05

2 . Definitions of the elements used in the project 06 - 07

3 . Design 08 - 11

4. Implementation 12 - 16

5. Result Screens 17 - 19

6. Conclusion 20

***ABSTRACT***

The "Hangman Game in Java" project is a classic word-guessing game developed using the Java programming language. The game is designed to entertain and challenge players by testing their vocabulary and word-guessing skills. It provides an interactive and user-friendly interface, making it enjoyable for players of all ages.

The project involves the use of various Java features and concepts, such as user input handling, data structures, and object-oriented programming. Players are presented with a secret word, represented by a series of blank spaces, and they must guess individual letters to reveal the hidden word within a limited number of attempts. A hanging man is used to represent the number of incorrect guesses made by the player.

**Key Features of the Hangman Game Project:**

1. **Random Word Selection:** The game randomly selects a word from a predefined list of words, ensuring a different challenge with each playthrough.
2. **User-Friendly Interface:** The game provides an intuitive and engaging graphical user interface (GUI) that allows players to input letters and track their progress.
3. **Letter Validation:** The program validates user input to ensure it is a single letter and not a previously guessed letter.
4. Incorrect Guess Tracking: The game keeps track of incorrect guesses and displays a hangman figure as a visual representation of the player's progress.
5. **Win/Lose Conditions:** The game checks for win and lose conditions, informing the player when they have successfully guessed the word or when they have exhausted their allowed attempts.
6. **Replayability:** Players can choose to play the game again and enjoy a new word, creating a dynamic and entertaining experience.
7. **Scalability:** The code is designed to be easily expandable for implementing additional features such as player names, scoring, and more.

This project not only provides an entertaining gaming experience but also serves as a valuable learning resource for Java programming beginners. It demonstrates how to create a simple yet engaging game with a graphical interface and incorporates fundamental Java concepts. Players can enhance their problem-solving skills while having fun with the Hangman Game in Java.

# **1. OBJECTIVE OF THE PROJECT**

The objective of the Hangman Game Project in Java with GUI is to design and implement an interactive and visually appealing hangman game application that provides an engaging user experience. The project aims to incorporate key programming concepts and features, including but not limited to:

1. **Graphical User Interface (GUI):**  Develop a user-friendly graphical interface using Java's Swing or JavaFX libraries to enhance the overall user experience. The GUI should include elements such as buttons, labels, and text fields to facilitate user interaction.
2. **Game Logic:** Implement the core game logic, including the selection of a random word, tracking of guessed letters, updating the display of the hidden word, and managing the state of the hangman figure. Ensure that the game adheres to the traditional rules of Hangman.
3. **Word Database:** Integrate a word database or list to provide a diverse set of words for the game. This database should allow for the selection of random words during each game session, ensuring variety and replayability.
4. **User Feedback:** Provide informative feedback to the user, including messages for correct and incorrect guesses, notifications for game status (win/lose), and updates on the progression of the hangman figure.
5. **Customization Options:** Allow users to customize aspects of the game, such as difficulty level (easy, medium, hard), theme, or the option to choose categories for words (e.g., animals, sports, technology).
6. **Error Handling:** Include robust error-handling mechanisms to gracefully manage unexpected user inputs or system errors, ensuring a stable and reliable gaming experience.
7. **Testing:** Conduct thorough testing of the application to identify and address any bugs, ensuring a high level of reliability and performance.

By achieving these objectives, the Hangman Game Project aims to deliver an enjoyable and well-crafted gaming experience with a polished graphical interface and robust functionality.

## **2. DEFINITIONS OF THE ELEMENTS USED IN THE PROJECT**

To implement a functional Hangman Game using Java programming with a GUI(Graphical User Interface) we require some modules to use. These modules enable us to use some already built-in functions and classes which makes it much easier to implement all the desired functionalities of the Game.

The modules required to implement a functional Hangman Game are:

### **Swing (javax.swing):**

Swing is a set of graphical user interface (GUI) components and tools included in the Java Standard Edition (Java SE) platform. It provides a rich set of libraries for creating and managing interactive graphical applications in Java. Swing allows developers to create cross-platform GUIs that can run on various operating systems without modification.

* **JFrame:** JFrame is the primary container for a Swing application. It represents the main window or frame of the application.
* **JPanel:** JPanel is a lightweight container used to organize and group components. It can be added to a JFrame to structure the layout of the GUI.
* **JButton:** JButton is a component that represents a clickable button. It is commonly used to trigger actions or events in response to user interaction.
* **JLabel:** JLabel is used to display text or images. It is a non-editable text component that provides a simple way to show information to the user.
* **LayoutManager:** Swing provides layout managers, such as FlowLayout, BorderLayout, and GridLayout, to define how components are arranged within a container.
* **Event Handling:** Swing supports event-driven programming, and it provides interfaces like ActionListener and classes like ActionEvent for handling user actions, such as button clicks.
* **SwingUtilities:** SwingUtilities is a utility class that provides various methods to assist in Swing programming. It includes methods for threading, invoking methods on the event dispatch thread, and other common tasks.
* **Custom Painting:** Swing allows custom painting through the paintComponent method, which can be overridden in components like JPanel to provide custom graphics.

### **Abstract Window Toolkit (java.awt):**

AWT (Abstract Window Toolkit) is a set of application programming interfaces (APIs) developed by Sun Microsystems (now owned by Oracle) as part of the Java Standard Edition (Java SE) platform. AWT provides a collection of classes and methods for creating and managing graphical user interfaces (GUIs) in Java applications. It is the original GUI toolkit for Java and is a part of the java.awt package.

* **Graphics and JPanel:** Graphics and JPanel from the java.awt package are used for custom drawing in the HangmanDrawingPanel class.
* **paintComponent:** The paintComponent method is overridden to draw the hangman figure based on the number of incorrect guesses.

### **ActionListener:** The **ActionListener** is an interface in the java.awt.event package that defines the method actionPerformed(ActionEvent e). It is used to handle action events, such as button clicks. In this project, the LetterButtonListener class implements ActionListener to respond to letter button clicks.

### **Graphics:** The **Graphics class**, part of the java.awt package, provides methods for drawing shapes and text. In this project, it is used in the paintComponent method of the HangmanDrawingPanel class to draw the hangman figure.

### **Random Class:** The **Random** class in Java is part of the java.util package. In this project, the Math.random() method is used to generate a random index to select a word from the wordsList.

### **SwingUtilities: SwingUtilities** is a class in Swing that provides utility methods for dealing with Swing components. In this project, it is used to invoke the JOptionPane on the event dispatch thread to display end-of-game messages.

### **StringBuilder: StringBuilder** is a class in Java that provides an efficient way to concatenate and manipulate strings. In this project, StringBuilder is used to construct and update the guessedWord and wrongGuesses strings.

Understanding and utilising these Swing and AWT components is crucial for developing Java applications with graphical user interfaces. The modular design of Swing allows developers to create interactive and visually appealing applications by combining different components to achieve the desired functionality.

**3. DESIGN**

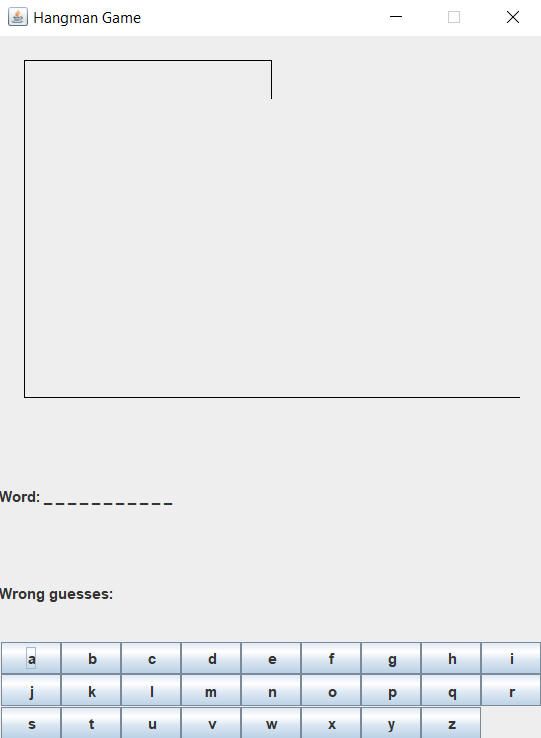
The design of a functional Hangman Game is broken down into the following elements which use the modules to make the game functional:

**1. User Interface (UI):**

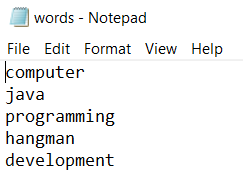
* The project uses the Java Swing library to create a graphical user interface (GUI) for the Hangman game.
* The main UI components include a `JFrame` for the game window, `JPanel` for organizing components, `JLabel` for displaying the word and wrong guesses, and `JButton` for letter selection.

**2. Game Initialization:**

* The game starts by creating an instance of the `HangmanGameGUI` class, which sets up the mainframe, initialises the game, and displays the UI.



**3. File-Based Word List (Updated):**

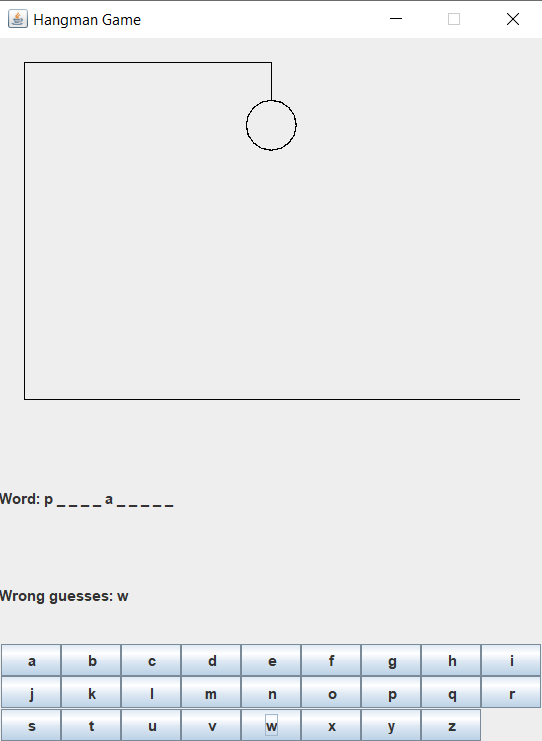
* The `WORDS` array is replaced with a `wordsList` read from a file (`words.txt`). This enhances the scalability and maintainability of the game by allowing easy addition or removal of words without modifying the code.

**4. Word Selection:**

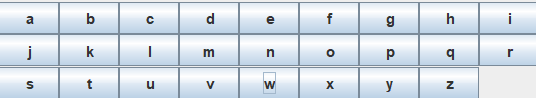
* The game randomly selects a word from a predefined array (`WORDS`) of possible words for the player to guess.



**5. Game Logic:**

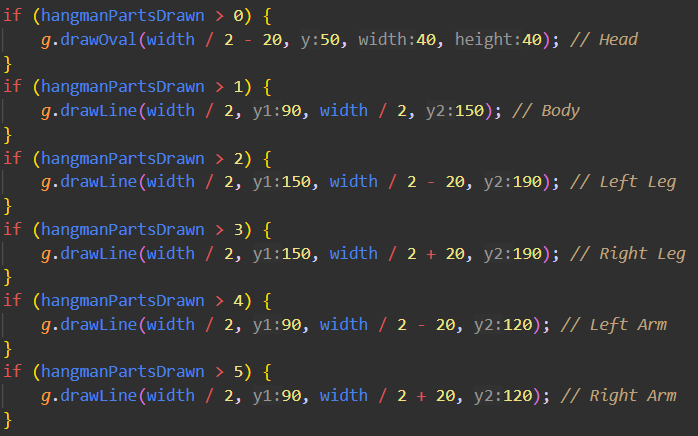
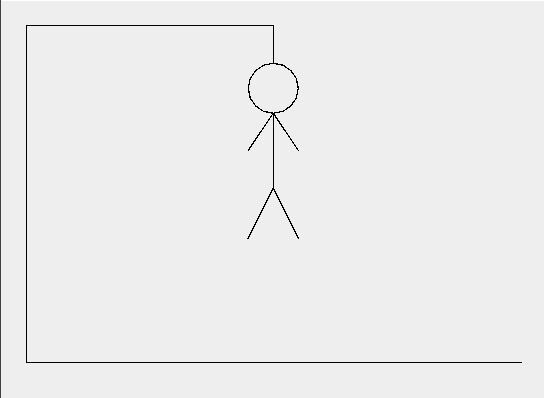
* The game logic is encapsulated in the `HangmanGameGUI` class.
* The core game loop allows the player to make letter guesses until they win or run out of attempts.
* The game tracks the current state of the guessed word, the number of tries left, and the incorrect guesses.

**5. User Interaction:**

* Letter buttons are provided for user input. Each button is associated with an `ActionListener` (`LetterButtonListener`) to handle letter selection.
* When a letter is guessed, the game updates the UI to reflect the changes, either revealing correct letters or updating the hangman figure for incorrect guesses.

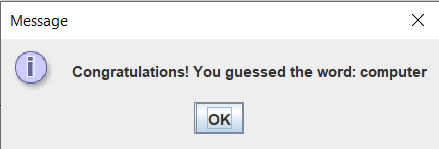
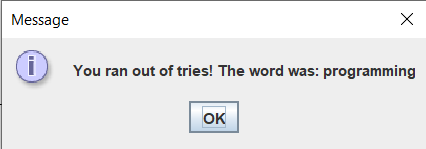
**6. Hangman Drawing Panel:**

* The `HangmanDrawingPanel` class extends `JPanel` and is responsible for drawing the hangman figure based on the number of incorrect guesses.
* The drawing includes lines representing the gallows, head, body, arms, and legs.



**7. Game Outcome Handling:**

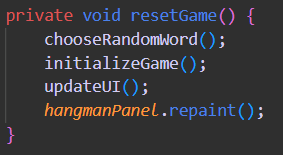
* The game checks for win or loss conditions after each letter guess.
* If the player successfully guesses the word, a congratulatory message is displayed, and the game is reset for a new round.
* If the player runs out of attempts, a message is displayed with the correct word, and the game is reset.



1. **Random Word Selection:**

* The `chooseRandomWord` method selects a random word from the predefined array of words.

1. **Game Reset:**

* The `resetGame` method is responsible for resetting the game state, selecting a new word, and updating the UI for a new round.

**10. Object-Oriented Approach:**

* The code follows an object-oriented approach, with classes like `HangmanGameGUI`, `LetterButtonListener`, and `HangmanDrawingPanel`.
* Encapsulation is used to hide the internal state of the game and provide well-defined interfaces for interaction.

The design of the Hangman game project is structured, leveraging the Swing library for GUI development and implementing game logic in an object-oriented manner. The user interface provides an intuitive way for players to interact with the game, and the game logic ensures a challenging and enjoyable gaming experience. The incorporation of a file-based word list allows for easy expansion of the game's vocabulary without code modification. Overall, the project design is modular, making it maintainable and extensible for future enhancements.

### **4. IMPLEMENTATION**

The Source code for a functional Hangman Game:

**import javax.swing.\*;**

**import java.awt.\*;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import java.io.BufferedReader;**

**import java.io.FileReader;**

**import java.io.IOException;**

**import java.util.ArrayList;**

**import java.util.List;**

**public class HangmanGameGUI {**

**private static final String WORDS\_FILE\_PATH = "D:\\EZTraining\\words.txt";**

**private List<String> wordsList;**

**private String wordToGuess;**

**private StringBuilder guessedWord;**

**private int triesLeft;**

**private StringBuilder wrongGuesses;**

**private JFrame frame;**

**private HangmanDrawingPanel hangmanPanel;**

**private JPanel textPanel;**

**private JLabel wordLabel;**

**private JLabel wrongGuessesLabel;**

**private JPanel buttonPanel;**

**private JButton[] letterButtons;**

**private int hangmanPartsDrawn = 0;**

**public HangmanGameGUI() {**

**frame = new JFrame("Hangman Game");**

**frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**frame.setSize(450, 600);**

**// Read words from the file**

**readWordsFromFile();**

**textPanel = new JPanel();**

**frame.add(textPanel, BorderLayout.SOUTH);**

**hangmanPanel = new HangmanDrawingPanel();**

**frame.add(hangmanPanel, BorderLayout.CENTER);**

**chooseRandomWord();**

**initializeGame();**

**createUI();**

**frame.setResizable(false);**

**frame.setVisible(true);**

**}**

**private void readWordsFromFile() {**

**wordsList = new ArrayList<>();**

**try (BufferedReader reader = new BufferedReader(new FileReader(WORDS\_FILE\_PATH))) {**

**String line;**

**while ((line = reader.readLine()) != null) {**

**wordsList.add(line.trim());**

**}**

**}**

**catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**private void chooseRandomWord() {**

**wordToGuess = wordsList.get((int) (Math.random() \* wordsList.size()));**

**}**

**private void initializeGame() {**

**guessedWord = new StringBuilder("\_".repeat(wordToGuess.length()));**

**triesLeft = 6;**

**wrongGuesses = new StringBuilder();**

**hangmanPartsDrawn = 0;**

**}**

**private void createUI() {**

**textPanel.setLayout(new GridLayout(3, 1));**

**wordLabel = new JLabel("Word: " + guessedWord.toString().replaceAll("", " ").trim());**

**textPanel.add(wordLabel);**

**wrongGuessesLabel = new JLabel("Wrong guesses: " + wrongGuesses.toString());**

**textPanel.add(wrongGuessesLabel);**

**buttonPanel = new JPanel(new GridLayout(3, 9));**

**letterButtons = new JButton[26];**

**for (char c = 'a'; c <= 'z'; c++) {**

**JButton button = new JButton(String.valueOf(c));**

**letterButtons[c - 'a'] = button;**

**button.addActionListener(new LetterButtonListener(c));**

**buttonPanel.add(button);**

**}**

**textPanel.add(buttonPanel);**

**}**

**private void updateUI() {**

**wordLabel.setText("Word: " + guessedWord.toString().replaceAll("", " ").trim());**

**wrongGuessesLabel.setText("Wrong guesses: " + wrongGuesses.toString());**

**}**

**private void resetGame() {**

**chooseRandomWord();**

**initializeGame();**

**updateUI();**

**hangmanPanel.repaint();**

**}**

**private class LetterButtonListener implements ActionListener {**

**private char letter;**

**public LetterButtonListener(char letter) {**

**this.letter = letter;**

**}**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**if (triesLeft > 0 && guessedWord.indexOf("\_") != -1) {**

**if (wordToGuess.contains(String.valueOf(letter))) {**

**for (int i = 0; i < wordToGuess.length(); i++) {**

**if (wordToGuess.charAt(i) == letter) {**

**guessedWord.setCharAt(i, letter);**

**}**

**}**

**} else {**

**triesLeft--;**

**wrongGuesses.append(letter).append(" ");**

**hangmanPartsDrawn++;**

**}**

**updateUI();**

**if (guessedWord.indexOf("\_") == -1) {**

**JOptionPane.showMessageDialog(frame, "Congratulations! You guessed the word: " + wordToGuess);**

**resetGame();**

**}**

**hangmanPanel.repaint();**

**}**

**}**

**}**

**private class HangmanDrawingPanel extends JPanel {**

**@Override**

**protected void paintComponent(Graphics g) {**

**super.paintComponent(g);**

**int width = getWidth();**

**int height = getHeight();**

**int bodyParts = 6;**

**g.setColor(Color.BLACK);**

**g.drawLine(20, height - 40, width - 20, height - 40);**

**g.drawLine(20, height - 40, 20, 20);**

**g.drawLine(20, 20, width / 2, 20);**

**g.drawLine(width / 2, 20, width / 2, 50);**

**if (hangmanPartsDrawn > 0) {**

**g.drawOval(width / 2 - 20, 50, 40, 40); // Head**

**}**

**if (hangmanPartsDrawn > 1) {**

**g.drawLine(width / 2, 90, width / 2, 150); // Body**

**}**

**if (hangmanPartsDrawn > 2) {**

**g.drawLine(width / 2, 150, width / 2 - 20, 190); // Left Leg**

**}**

**if (hangmanPartsDrawn > 3) {**

**g.drawLine(width / 2, 150, width / 2 + 20, 190); // Right Leg**

**}**

**if (hangmanPartsDrawn > 4) {**

**g.drawLine(width / 2, 90, width / 2 - 20, 120); // Left Arm**

**}**

**if (hangmanPartsDrawn > 5) {**

**g.drawLine(width / 2, 90, width / 2 + 20, 120); // Right Arm**

**}**

**if (hangmanPartsDrawn > bodyParts - 1) {**

**SwingUtilities.invokeLater(() -> {**

**JOptionPane.showMessageDialog(frame, "You ran out of tries! The word was: " + wordToGuess);**

**resetGame();**

**});**

**}**

**}**

**}**

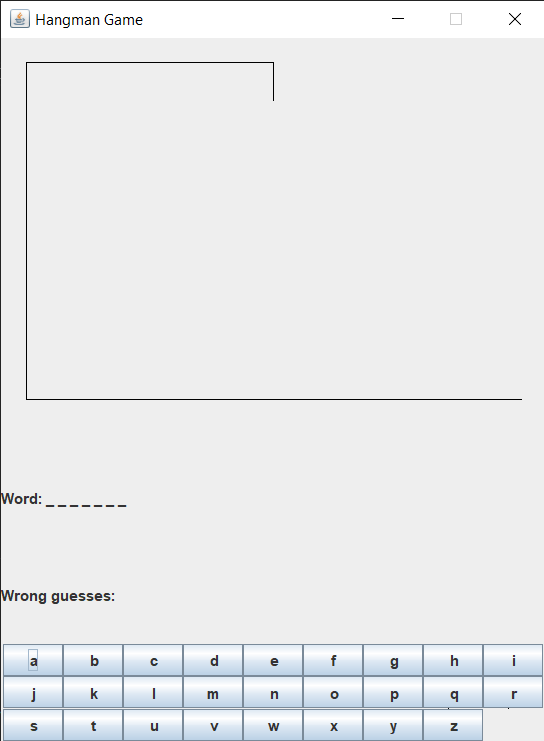
**public static void main(String[] args) {**

**SwingUtilities.invokeLater(() -> new HangmanGameGUI());**

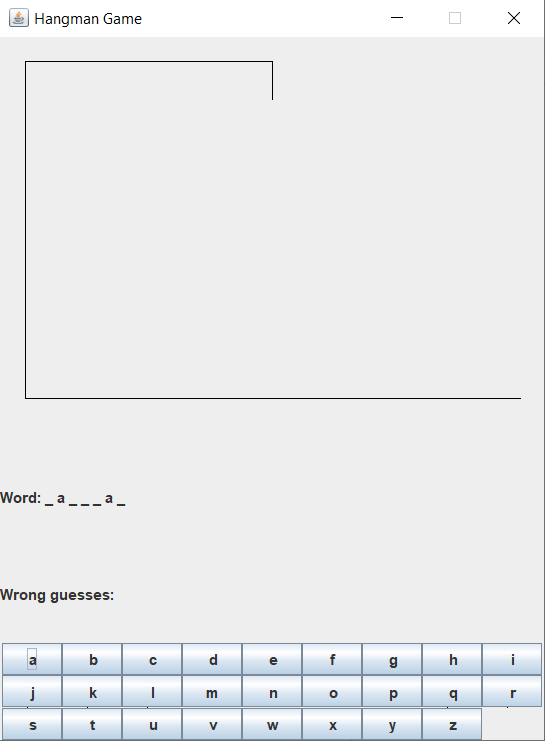
**}**

**}**

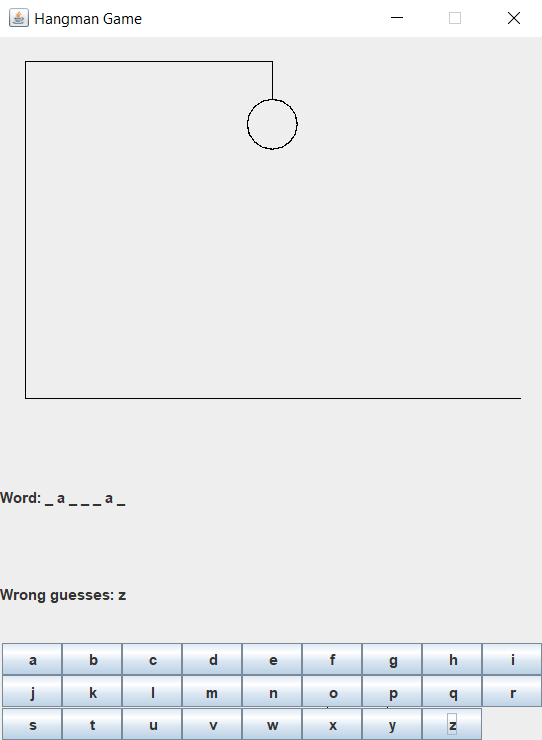
**5. RESULT SCREENS**

****

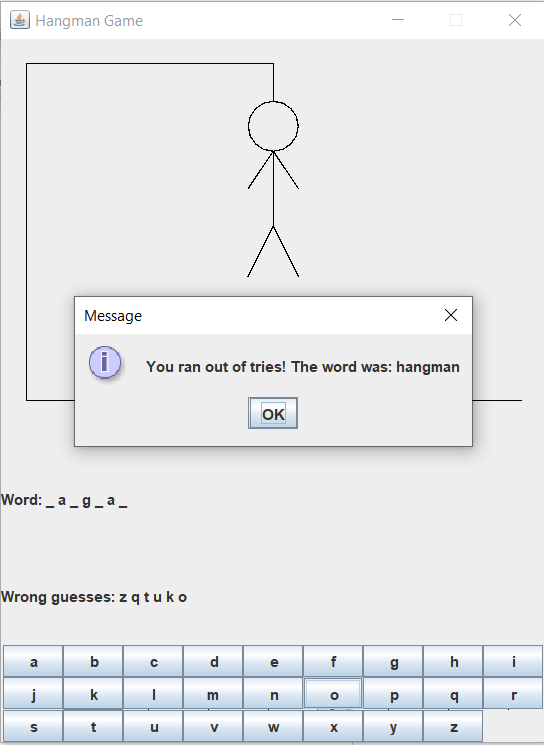
* GUI of Hangman Game at the beginning of the game:
* When player guesses the right letter:



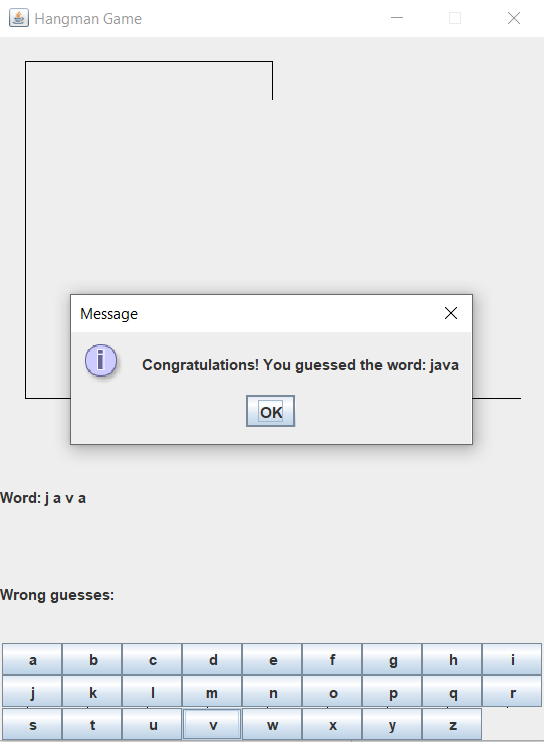
* When player guesses the wrong letter:

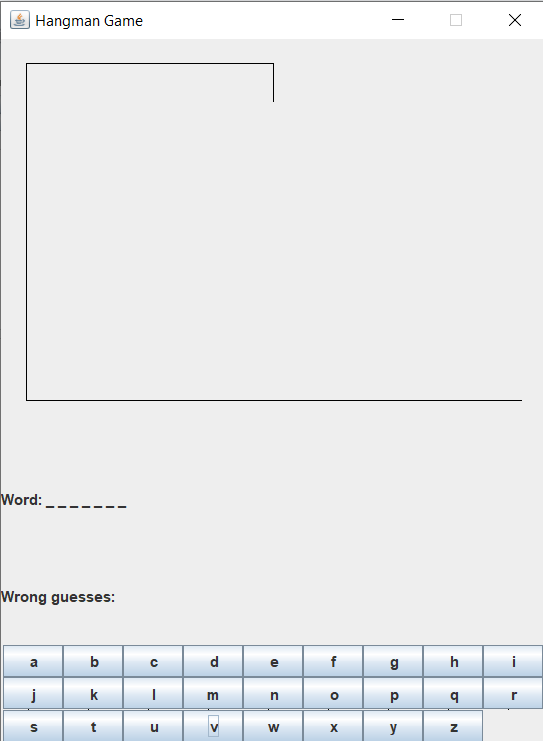


* When player runs out of tries and loses:



* When player guesses the word right:



* When player clicks on OK after winning or loosing:  
  

**6. CONCLUSION**

The Hangman game project demonstrates a well-designed and interactive implementation of the classic word-guessing game using Java and the Swing library. The following key points summarise the project's conclusion:

* + **Successful Implementation:** The project successfully implements a user-friendly Hangman game with a graphical user interface, providing an engaging gaming experience.
  + **Object-Oriented Design:** The project follows an object-oriented design, encapsulating game logic within classes, promoting code modularity, and facilitating maintainability.
  + **Effective Use of Java Swing:** The use of Java Swing components ensures a visually appealing and responsive user interface. The GUI design enhances the overall gaming experience.
  + **Dynamic Game Flow:** The game dynamically updates the UI based on user input, providing real-time feedback on letter guesses and illustrating the hangman figure as the game progresses.
  + **File-Based Word List:** The project includes a file-based word list, enhancing the flexibility to add, remove, or modify words without altering the source code. This feature makes the game easily adaptable to different word sets.
  + **Clear Win/Loss Handling:** The game effectively handles both winning and losing scenarios, displaying appropriate messages and resetting the game for continued play.

The Hangman game project offers various opportunities for future enhancements and expansions:

* + **User Authentication:** Implement user authentication to allow multiple players to have personalised game sessions and track individual scores.
  + **Multiplayer Support:** Extend the game to support multiplayer functionality, allowing users to compete against each other, either locally or over a network.
  + **Difficulty Levels:** Introduce different difficulty levels with varying word lengths or limited attempts to cater to players of different skill levels.
  + **Word Categories:** Categorize words and provide users with the option to choose specific themes or categories, adding variety to the gameplay.
  + **Score Tracking:** Implement a scoring system to track user performance, including metrics such as the number of games won, lost, and average guesses per game.
  + **Customizable Themes:** Allow users to customise the game theme, including background images, fonts, and colour schemes, to personalise their gaming environment.