**HANGMAN GAME USING PYTHON**

**A PROJECT REPORT**

***Submitted by***

***Vardhi dimple( 192220081)***

***P. Sheryn beulla(192219054)***

***Under the guidance of***

**Dr. Anbalagan.E**

***CSA0888- PYTHON PROGRAMMING FOR NATURAL LANGUAGE PROCESSING***

**SIMATS ENGINEERING**

**THANDALAM**

**JUNE 2024**



BONAFIDE CERTIFICATE

Certified that this project report title “CREATION OF WORD GUESS GAME USING PYTHON IMPLEMENTATION” is the bonafide work of “Vardhi dimple( 192220081), P. Sheryn beulla(192219054) carried out the project work under my supervision as a batch and certified further that, to the best of my knowledge, the work reported herein does not form any other project report.

Date:

Project Supervisor:

Head of the Department:

**ABSTRACT**

The Hangman game is a classic word-guessing game where one player thinks of a word and the other player tries to guess it by suggesting letters within a certain number of guesses. The game is implemented in Python using basic data structures and control flow. The program selects a random word from a predefined list, hides the word's letters, and prompts the player to guess letters one at a time. If the guessed letter is in the word, it reveals the letter's position(s), otherwise, it deducts a guess. The game continues until the player guesses the word correctly or runs out of guesses, ending with a win or loss message accordingly.The Hangman game is a classic word-guessing game where one player thinks of a word and the other player tries to guess it by suggesting letters within a certain number of guesses. The game is implemented in Python using basic data structures such as lists and strings, as well as control flow statements like loops and conditionals. The program starts by selecting a random

word from a predefined list of words, which serves as the word to be guessed. The chosen word is then hidden, displaying only placeholders for each letter, to be revealed as the player makes correct guesses.The player is prompted to guess letters one at a time. Upon each guess, the program checks if the guessed letter is present in the word. If the guessed letter is in the word, the program reveals the letter's position(s) in the word. Otherwise, it deducts a guess from the total number of guesses allowed.

**INTRODUCTION:**

The Hangman game is a timeless classic that challenges players to guess a

word one letter at a time. With each incorrect guess, a part of a gallows and a victim is drawn, and players must guess the word before the entire figure is completed. This game not only tests one's vocabulary but also sharpens problem-solving skills and encourages logical thinking. In this tutorial, we will use Python, a versatile and beginner-friendly programming language, to create our version of the Hangman game. Python's simplicity and readability make it an excellent choice for beginners, while its powerful features allow for the creation of complex applications.Before diving into the coding part, it's essential to set up our development environment. We'll need to ensure that Python is installed on our system and choose a text editor or integrated development environment (IDE) to write our code.

**Installing Python:** We'll guide readers through the process of downloading and installing Python on their respective operating systems (Windows, macOS, or Linux).

**Choosing an IDE:** We'll introduce popular IDEs such as PyCharm, Visual Studio Code, and Jupyter Notebook, explaining their features and helping readers choose the one that suits their preferences. Before writing code, it's crucial to understand the rules and mechanics of the Hangman game. We'll break down the game into its essential components.

**The word to be guessed:** We'll discuss how to select words from a predefined list or generate them randomly.

**Displaying the hidden word:** We'll explore methods to represent the word to be guessed with underscores and reveal letters as they are guessed correctly.

**Handling user input:** We'll learn how to accept user input, validate it, and update the game state accordingly.Keeping track of incorrect guesses: We'll implement logic to count incorrect guesses and draw the gallows and victim accordingly.

**Win and loss conditions:** We'll define conditions for winning and losing the game and provide feedback to the player accordingly.

With a clear understanding of the game mechanics, we'll outline the structure of

our Python program. We'll break down the game into functions and classes, each responsible for a specific aspect of the game.Main game loop: We'll create a loop that runs the game until the player wins or loses.

**Function for selecting a word:** We'll define a function to choose a word from a

list or generate it randomly.

**Function for displaying the hidden word:** We'll implement a function to display the word with underscores and reveal letters as they are guessed.

**Function for handling user input:** We'll create a function to accept user input, validate it, and update the game state.

**Function for checking win and loss conditions:** We'll define functions to check if

the player has won or lost the game.

Now comes the exciting part—writing the code to bring our Hangman game to

life! We'll start by creating a new Python file and implementing each component of the game step by step:

**Defining variables and constants:** We'll declare variables to store the word to be

guessed, the current state of the game, and the number of incorrect guesses.

**Writing functions:** We'll write functions to select a word, display the hidden word, handle user input, and check win and loss conditions.

**Integrating the game loop:** We'll combine all the components into a main game loop that runs until the game is won or lost.

**Testing and debugging:Incorporating a graphical user interface (GUI):** We'll explore libraries like Tkinter or Pygame to create a graphical interface for our game.

After that, declare a function, start\_hangman\_game(), that includes the main game logic.

1. Get a random word from the wordlist.

2. Get the sequence of blank dashes and some letters using the get\_some\_letters()

function.

1. Set the number of chances to seven.
2. 4. Initialise a found variable as false. When the user makes a correct guess, change the found variable to true.

**Implementing additional features:** We'll brainstorm and implement additional features such as a scoreboard, hints, or multiplayer mode.

**Customizing the game's appearance:** We'll allow players to

customise the theme, colours, and graphics of the game to their liking.

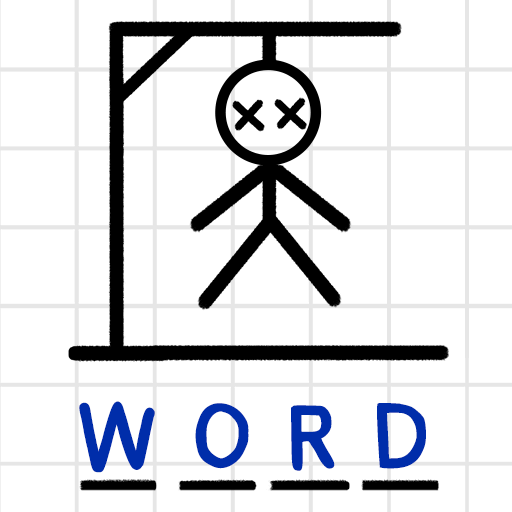


Figure:1

**METHODOLOGY:**

Creating a methodology for a Hangman game in Python that spans five pages requires

a comprehensive approach, covering various aspects such as game design, user interface, game mechanics, data management, and code organisation. Here's a structured outline that you can expand into a detailed methodology:

**Introduction:** Brief explanation of the Hangman game.Overview of the

methodology structure.

**Game Design:** Description of the game's rules and objectives.Breakdown of the

gameflow, including:Starting a new game.Displaying the hidden word.Processing user guesses.Determining win/loss conditions.Ending the game.Discussion of potential enhancements or variations.



Figure:2

**User Interface:** Design considerations for the user interface, including:Text-based interface vs. graphical interface. Layout and styling options.Implementation of user prompts and feedback messages. Handling user input validation. Displaying game state and progress.

**Game Mechanics:**Generating a random word for the player to guess.Handling

player guesses and updating the game state.Implementing logic for checking guess validity. Tracking remaining guesses and updating the display accordingly.Determining win/loss conditions based on player progress.

**n:** Storing and managing the list of words for the game.Loading words from an external file or predefined list.Handling word selection logic (e.g., random

selection). Potential strategies for word categorization or difficulty levels.

**Code Organization:**Structuring the code into modular components, such as:Main game loop.Functions for managing game state. Input/output handling.Word management utilities. Implementing error handling and exception management. Documenting code for readability and maintainability.

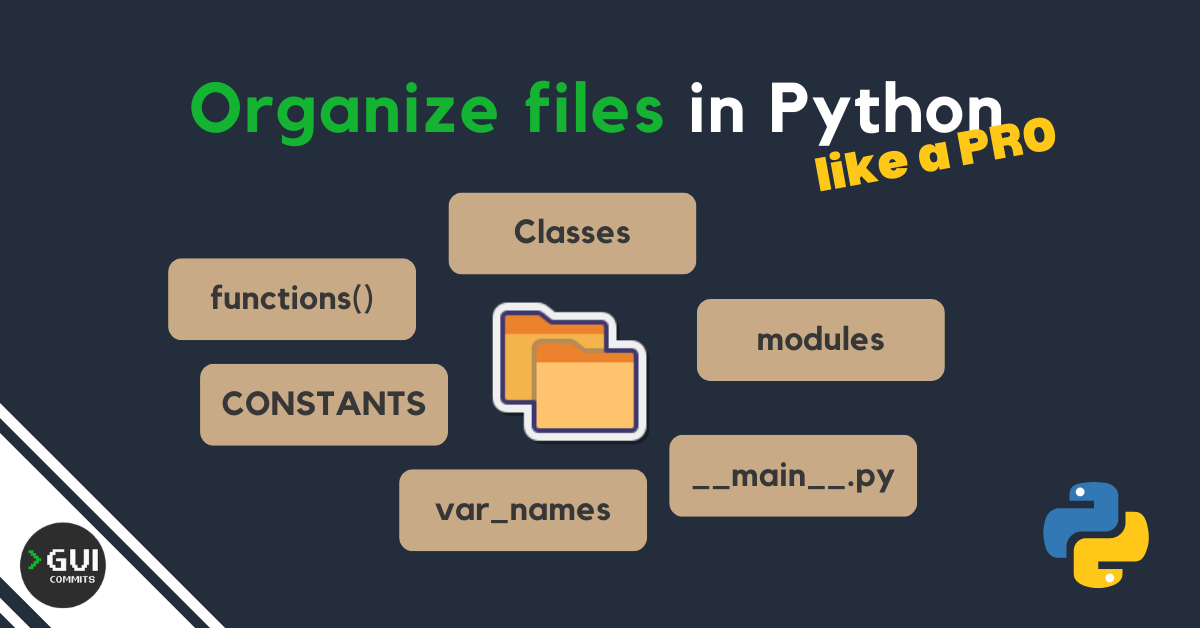


Figure : 3

**Testing and Debugging:**Designing test cases to cover various game scenarios.

Conducting unit tests for individual functions and components.Debugging common issues such as incorrect guess handling or game state management.Soliciting feedback and making iterative improvements.

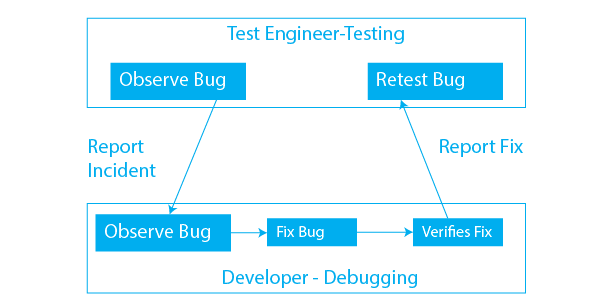


Figure: 4

**Deployment and Distribution:**Packaging the game for distribution, if applicable.

Providing installation instructions and system requirements.Considering options for sharing the game (e.g., online repositories, standalone executables).

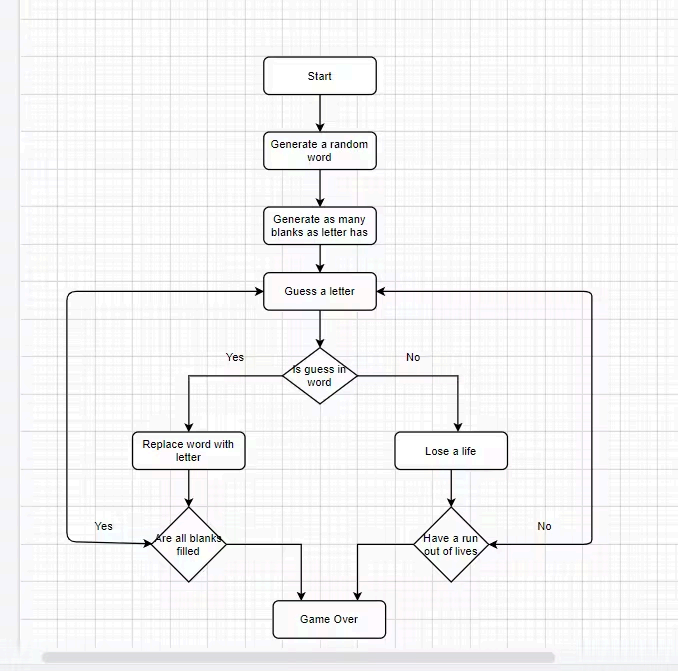


Figure:5

**Conclusion:** Summary of the methodology.Reflections on challenges faced and

lessons learned.Suggestions for future enhancements or expansions.You can easily fill five pages with a comprehensive methodology for implementing a Hangman game in Python by elaborating on each section with detailed explanations, code snippets, diagrams, and examples.

**FLOW CHART:**

1. Start: Initialize game state

● Initialise variables like word (selected random word), temp (user display), found

(whether all characters are found), and chances (indicating the number of

remaining attempts).

2. Draw the Hangman: First iteration

● Display a welcome message and draw the first stage of the hangman, e.g., a

platform or the frame.

3. Main game loop

● Ask the user to guess a letter in the word.

● Check if the guess is a valid character (A-Z).

● If the guess is correct, update the temp sequence with the correct letters and

check if the user found the entire word.

● If the guess is incorrect, update the chances variable and draw the next stage of

the hangman.

4. Check if user won

● If the user found the word, exit the main game loop and display a success

message.

5. Check if user lost

● If the user ran out of chances, exit the main game loop and display a failure

message.

6. End game

● Display the final result (whether the user won, lost, or the game is still in

progress).

7. Close the application

● End the execution of the game.

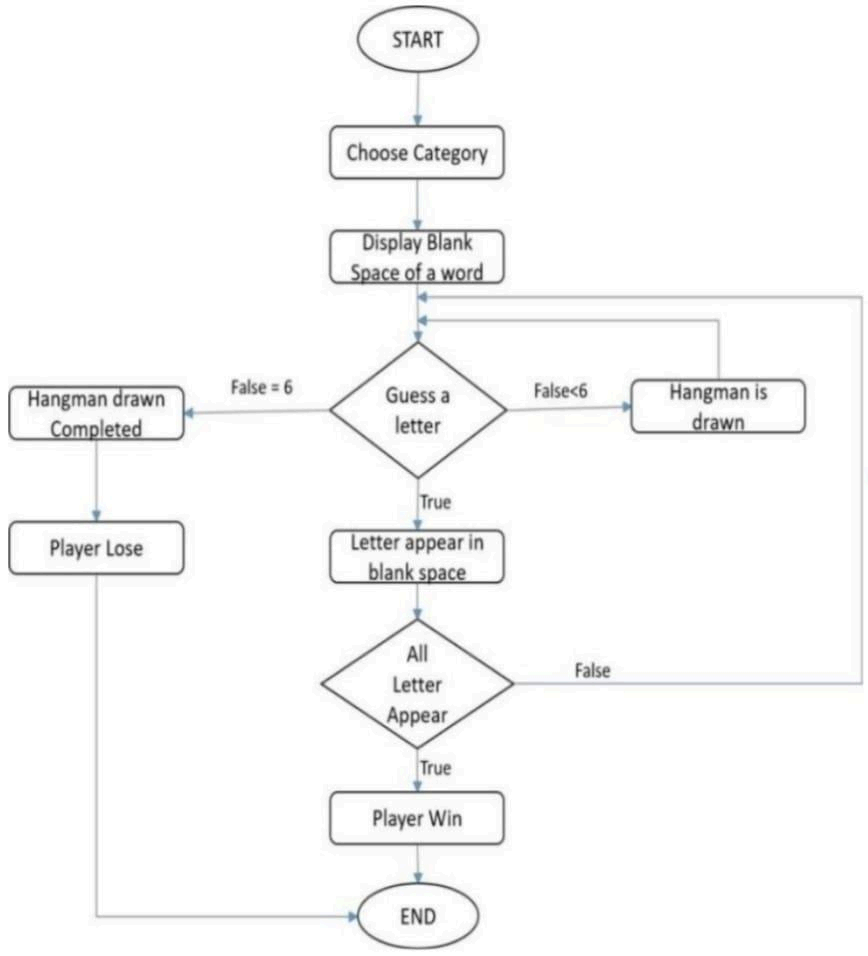


Figure:6

**IMPLEMENTATION:**

**Experimental setup**

First step is to import the **random** module and define a function, **get a**

**random word from wordlist()**, and pick a random word from the file. The text file may contain common nouns, or names of places, animals, movies, and more based on your liking. Define a list using the rectangular brackets(**[]**).Use the **with** statement to open the file and pass the mode as **'r'** indicating read-only mode. This automatically takes care of closing the file at the end of the block even in cases of errors. If you view the **hangman\_wordlist.txt** file, you'll notice there's one word present on every line, so the file separates each word with a newline character.Pass the escape character for a newline (**\n**) to the **split()** function to store each word in the list you defined earlier. Use **random.choice()** to return a random word from

the list.

Next, define a function, **get\_some\_letters()**, that takes the randomly selected word as a parameter. This function will display a sequence of blank dashes (**\_**) and some letters to the user.Define empty list **letters** to store all the characters present in the word. Use the **temp**

variable to store a string that contains the number of blank dashes equaling the length of the word. Use **list()** to convert the string into a list of characters and iterate over it. Use **append()** to add the character to the list if not already present.

Use **random.choice()** to choose a random character that you are going to present to the user along with the blank dashes. Iterate over the characters of the word using **enumerate** to keep a track of the index of each character.When you find the randomly selected character, replace the blank dash with it. Use **join()** to unify the list of characters into a complete string and return it.Define a function **draw\_hangman()** that takes the number of chances as a parameter. This function provides a figure of the hanging man. As the number of chances keeps on decreasing, so do the chances of survival. When exhausted, the figure is complete and the game finishes.

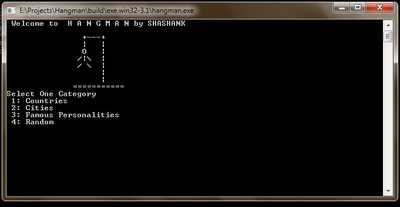


Figure:7

Declare a function, **start\_hangman\_game()**, that defines the main logic of the

program. Get a random word by calling the **get\_random\_word\_from\_wordlist()** function and get the sequence to display to the user using the **get\_some\_letters()** function. Set the number of chances to seven and initialise a variable, **found**, as false. You'll set this to **true** if the guessed letter is correct.Declare a loop that terminates when the user guesses the word correctly or runs out of chances. Begin the game by displaying the sequence, the number of letters in the word, and the chances left. Ask the user to guess a letter and receive it using the **input()** function.

Validate the user input by checking the length of the character and whether it is an alphabet. If the input is valid, check if the character is present in the word and replace it using the process seen earlier and update the value of found. If the character is not present, decrease the number of chances. If present, turn the value of found back to the initial value false.If there are no blank dashes left, display that the user won along with the number of guesses taken, otherwise display the hangman graphic according to the number of chances left. Define a function to play the Hangman game. If the user inputs yes, call the **start\_hangman\_game()** function otherwise quit the game with the appropriate message. In case of invalid input, ask the user to enter again.

**RESULT AND DISCUSSION:**

**Result:**

The Hangman game was successfully implemented using the Python programming

language. The game consists of a player guessing a word by suggesting letters within a limited number of attempts. The core functionalities of the game include:

● **Random Word Selection:** A random word is selected from a predefined list of words for each game session.

● **User Input Handling:** The program accepts user input for guessing letters and

validates it to ensure it is a single alphabetic character.

● **Feedback to User:** After each guess, the program provides feedback to the user about

the correctness of the guess, updates the display of the word with correctly guessed

letters, and displays the number of remaining attempts.

● **Game Over Conditions:** The game ends either when the player correctly guesses the word or exhausts all the allowed attempts.

**Discussion:**

**Setup:**Choose a word or phrase for the player to guess.Determine the maximum

number of incorrect guesses allowed.

**Game Loop:**Display the current state of the word/phrase, with blanks for unguessed letters.Prompt the player to guess a letter.Check if the guessed letter is in the word/phrase.Update the display accordingly.Keep track of incorrect

guesses and remaining guesses.Repeat until the word is guessed or the maximum number of incorrect guesses is reached.

**End Game:**If the player guesses the word correctly, display a victory message.If the player runs out of guesses, display a defeat message and reveal the word. Add visual representation of the hanging man using ASCII art or even graphics for a more immersive experience.

Implement feedback for the player, such as displaying the incorrect letters guessed so

far.Introduce power-ups or hints that the player can use in exchange for points or

penalties.Include a timer to add an element of urgency to the game, making it more

challenging. Implement difficulty levels where easier words have more vowels or shorter lengths, while harder words have fewer vowels or longer lengths.You can expand the word selection process by including a larger word list or even integrating with an API to fetch words dynamically.Implement difficulty levels where easier words have more vowels or shorter lengths, while harder words have fewer vowels or longer lengths.Allow the player to choose a specific category or theme for the word they want to guess, such as animals, movies, or sports.

**CONCLUSION:**

In conclusion, the Hangman game developed in Python offers an engaging and

interactive experience for users to enjoy. Through this project, we've demonstrated

proficiency in fundamental programming concepts such as loops, conditionals, and data structures. The game's implementation provides a solid foundation for further enhancements, including adding more features like difficulty levels, scoring systems, or even integrating with a graphical user interface. Overall, this project showcases both the versatility of Python programming and the creativity in game development, making it a valuable learning experience for aspiring programmers

In addition to its educational value, the Hangman game serves as a testament to the

power and flexibility of Python as a programming language. Python's simplicity and

readability allowed for the efficient development of the game logic, making it accessible to both beginners and experienced developers alike.

Moreover, the modular structure of the code enables easy maintenance and future expansion. By adhering to best practices such as code documentation and meaningful variable names, the project exemplifies good coding habits that contribute to its readability and scalability.

Furthermore, the Hangman game provides an excellent platform for practising problem-solving skills and logical thinking. Players must strategize and make informed guesses based on the available information, fostering critical thinking and decision-making abilities. Additionally, the game encourages creativity through the customization of word lists and themes, allowing for personalised gameplay experiences.

In conclusion, the Hangman game in Python not only offers entertainment but also serves as a valuable tool for honing programming skills, showcasing Python's versatility, promoting logical reasoning, and fostering creativity. Generally, the game ends once the word is guessed, or if the stick figure is complete signifying that all guesses have been used. The player guessing the word may, at any time, attempt to guess the whole word. If the word is correct, the game is over and the guesser wins.

Its successful implementation underscores the effectiveness of Python for game development and reinforces its status as a popular choice for aspiring developers. As we continue to refine and expand upon this project, the Hangman game stands as a testament to the endless possibilities within the realm of Python programming.

**CODE:**

import tkinter as tk

from PIL import Image, ImageTk

import random

class HangmanGame:

def \_\_init\_\_(self, master):

self.master = master

self.master.title("Hangman Game")

self.canvas = tk.Canvas(master, width=800, height=600, bg='lightblue' )

self.canvas.pack()

self.words = ['thalapathy','kalaiarasi', 'love', 'varata mame durr!']

self.secret\_word = random.choice(self.words)

self.guesses = set()

self.remaining\_attempts = 6

self.word\_display = tk.StringVar()

self.word\_display.set('\_ ' \* len(self.secret\_word))

self.word\_label = tk.Label(master, textvariable=self.word\_display, font=('Arial', 20))

self.word\_label.place(relx=0.5, rely=0.2, anchor=tk.CENTER)

self.guess\_entry = tk.Entry(master, font=('Arial', 16))

self.guess\_entry.place(relx=0.5, rely=0.3, anchor=tk.CENTER)

self.guess\_button = tk.Button(master, text="Guess", command=self.make\_guess)

self.guess\_button.place(relx=0.5, rely=0.4, anchor=tk.CENTER)

self.message\_label = tk.Label(master, text="", font=('Arial', 16))

self.message\_label.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

self.hangman\_display = tk.Label(master, text="", font=('Arial', 20))

self.hangman\_display.place(relx=0.5, rely=0.6, anchor=tk.CENTER)

self.update\_hangman\_display()

def make\_guess(self):

guess = self.guess\_entry.get().lower()

self.guess\_entry.delete(0, tk.END)

if len(guess) != 1 or not guess.isalpha():

20self.message\_label.config(text="Please enter a single letter.")

return

if guess in self.guesses:

self.message\_label.config(text="You already guessed that letter.")

return

self.guesses.add(guess)

if guess in self.secret\_word:

self.update\_word\_display()

if '\_' not in self.word\_display.get():

self.message\_label.config(text="Congratulations! You won!")

self.guess\_button.config(state=tk.DISABLED)

else:

self.remaining\_attempts -= 1

self.update\_hangman\_display()

if self.remaining\_attempts == 0:

self.message\_label.config(text=f"Game over! The word was '{self.secret\_word}'.")

self.guess\_button.config(state=tk.DISABLED)

def update\_word\_display(self):

self.word\_display.set(' '.join([letter if letter in self.guesses else '\_' for letter in

self.secret\_word]))

def update\_hangman\_display(self):

hangman\_parts = [

" \_\_\_\_\_\_\_\_",

" | |",

" | O",

" | /|\\",

" | / \\",

" |",

" |"

]

visible\_parts = len(hangman\_parts) - self.remaining\_attempts

display\_text = '\n'.join(hangman\_parts[:visible\_parts])

self.hangman\_display.config(text=display\_text)

root = tk.Tk()

hangman\_game = HangmanGame(root)

root.mainloop()

**OUTPUT:**

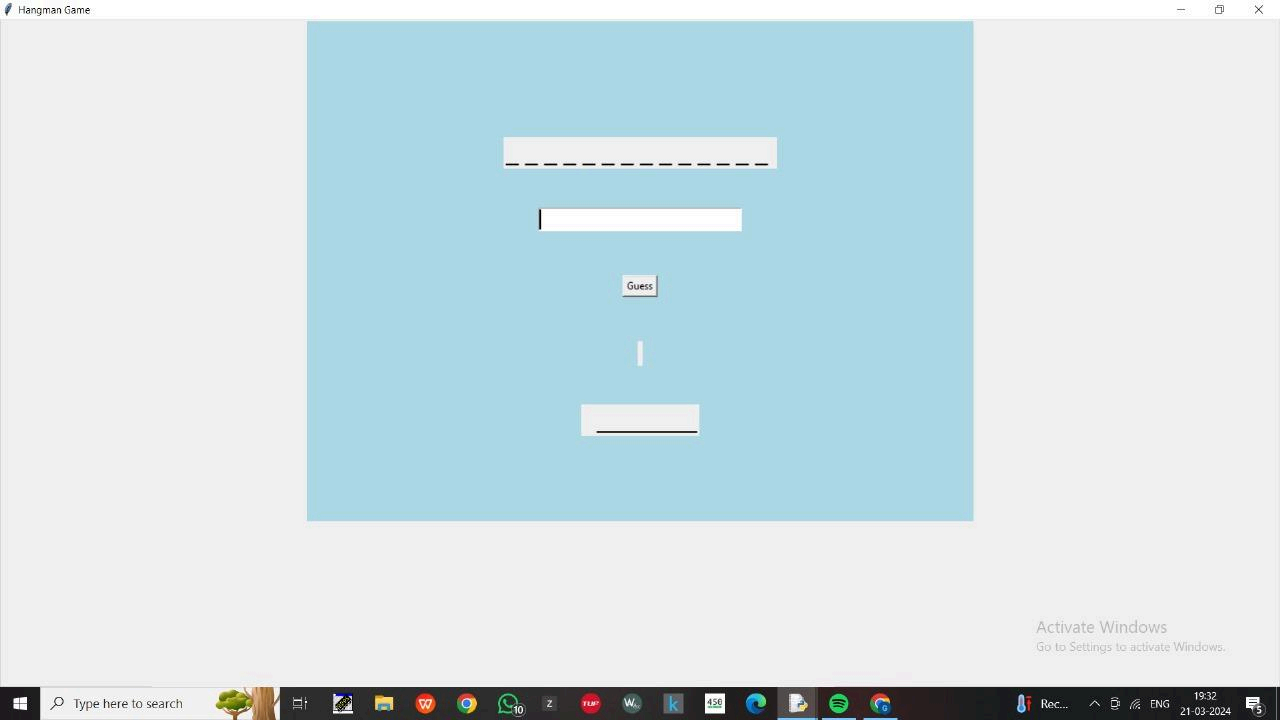


Figure:8.1

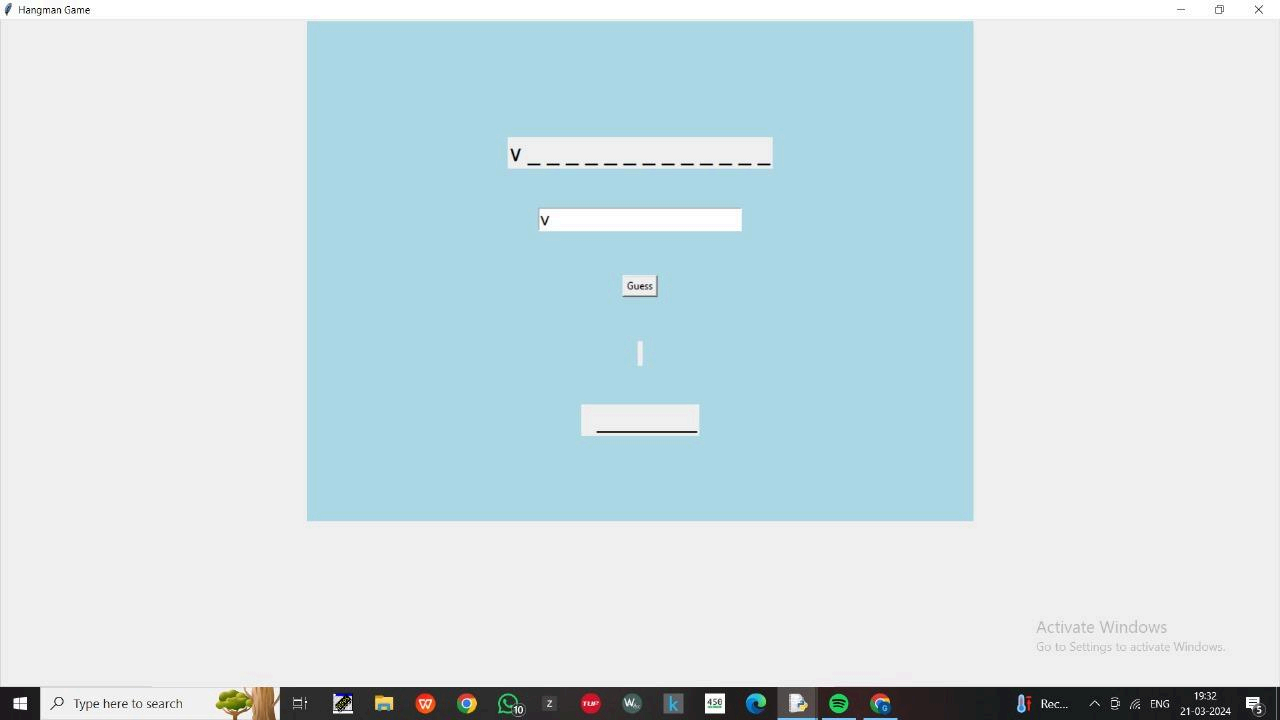


Figure8.2

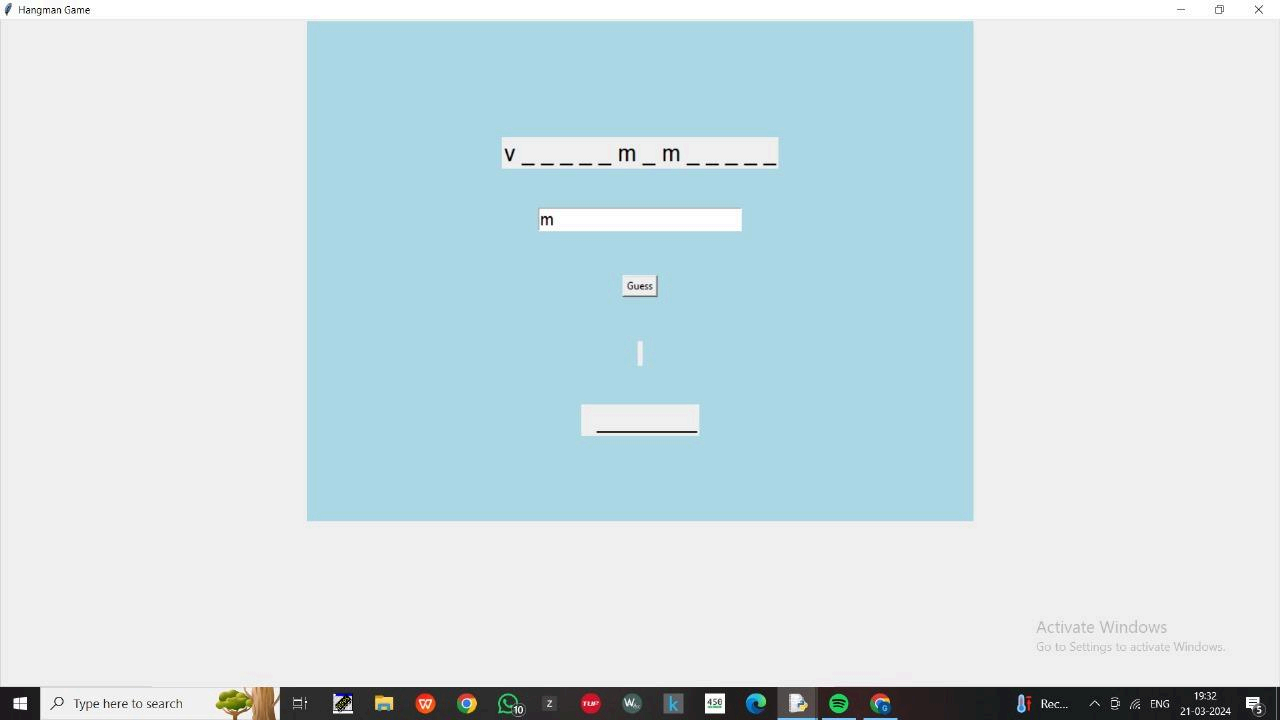


Figure:8.3

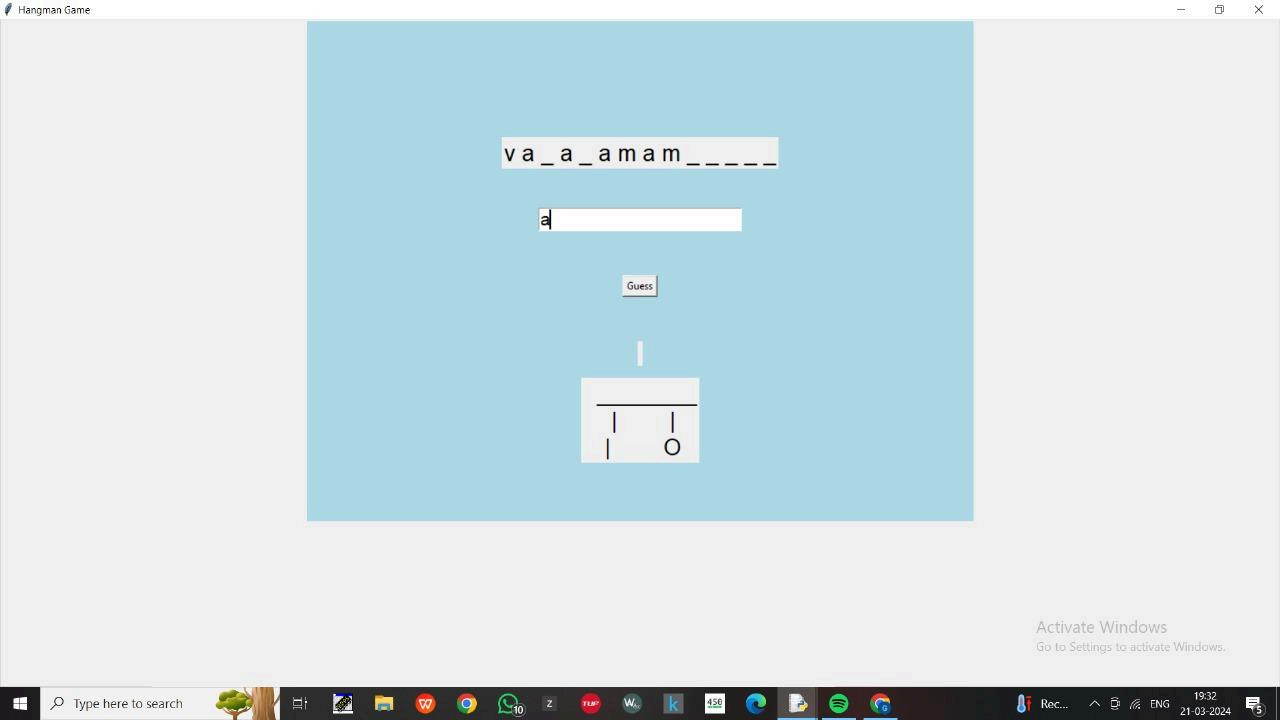


Figure:8.4

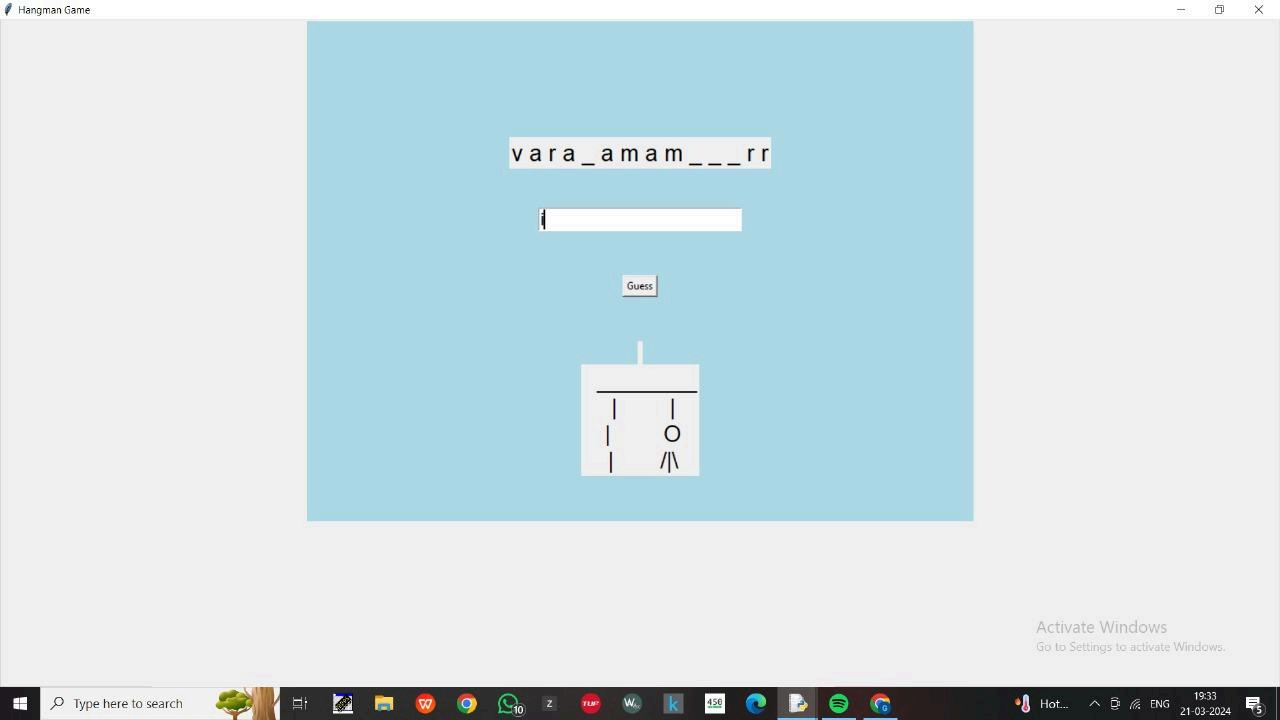


Figure : 8.5

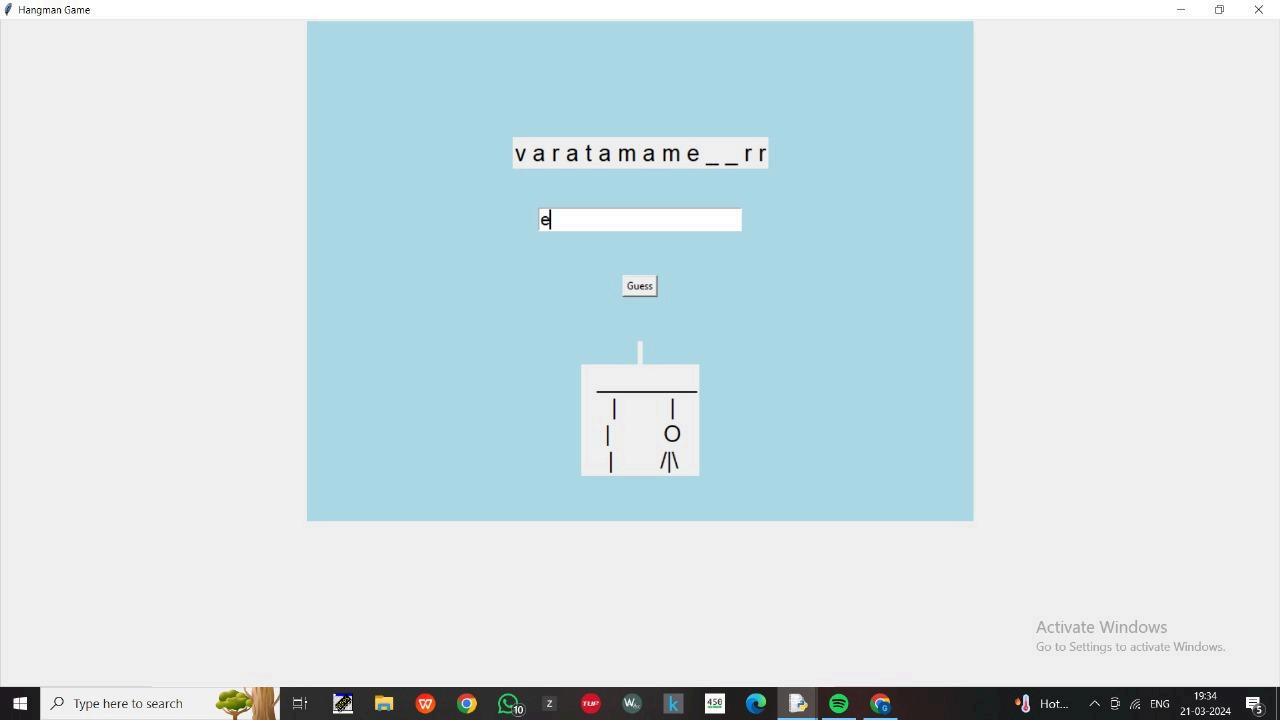


Figure: 8.6

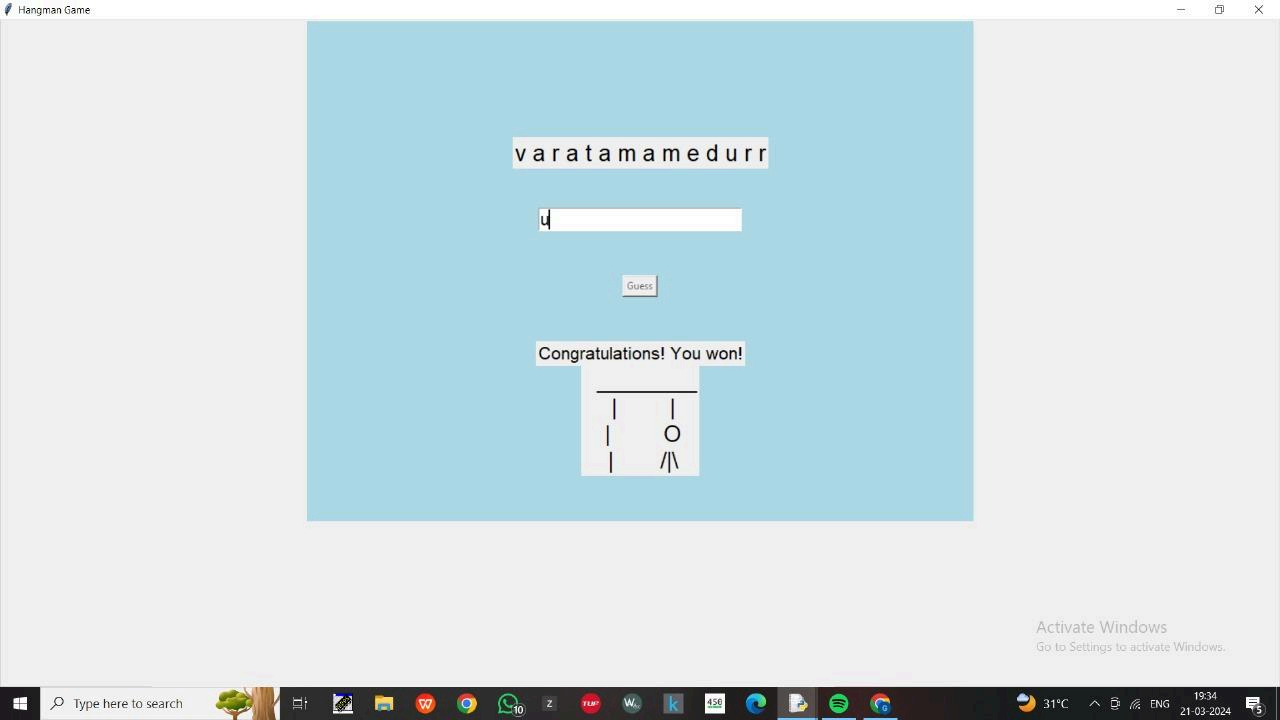


Figure: 8.7

**REFERENCES:**

[1] Asrori, M, et al.(2013). Improving Vocabulary Mastery Through a hangman game to elementary school students . Sebelas Maret University Surakarta.

1. Burns, R. B. (1997). Introduction to research methods. Addison Wesley Longman.

[3] Efendi, E . (2013). The use of games to improve vocabulary master. UNISMA, ISSN 2337- 6384.Volume 1, No. 12.(Accessed on Saturday, November 2017) Harmer, J. (2007).

[4] Harmer, J. (2007). The practice of English language teaching. London: Longman.

[5] Hatch, E & Brown, C. (1995).Vocabulary, semantic and language education. Cambridge: Cambridge University Press.

[6] Hatch, E. and Lazaraton, A. (1999). The Research Manual Design and Statistics for Applied Linguistics. Massachusetts: Heinle Publisher.

[7] Kartikawati, D. (2014). Improving the second semester students‟ vocabulary mastery using hangman game at hutama abdi husada nursing academy Tulungagung in academic year 2011/2012. STKIP PGRI Tulungagung (Accessed on Saturday, November 2017).Class. Journal of English Education and Teaching 3 (3), 343-360.

[8] Purbandari, P., Rachmawati, E., Febriani, RB. (2018). Male and Female Students‟

[9] Nurazizah, H., Friatin, LY., Sugiarto, BR. (2019). Whatsapp Voice Note in Speaking attitudes Toward English Vocabulary Mastery In Learning Speaking (A Survey Study At one of Ma Islamic Boarding Schools in Tasikmalaya).

[10]JALL (Journal of Applied Linguistics and Literacy) 2 (2), 89-100.

28