Report

Hang the Man

The Hangman Game

Team Members

Mahi J Doshi	PES2UG25AM151
Manisha Balaji	PES2UG25CS293
Lisha Chowdary	PES2UG25AM144
Nidhi Begane Sreesha	PES2UG25CS332

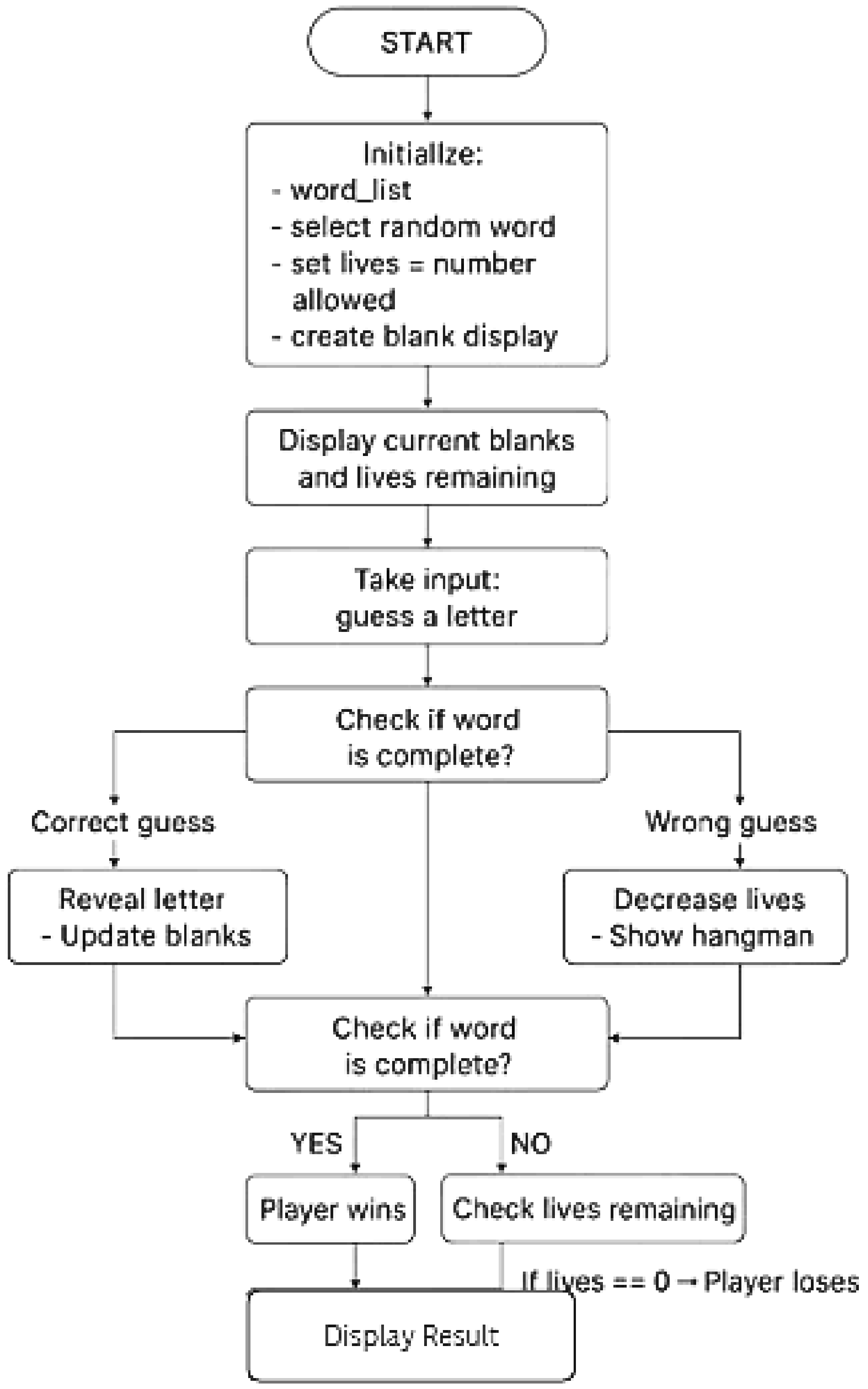
Table of Contents

Problem Statement

The objective of this project is to design and implement a fully functional Hangman game in Python by dividing the program into four independent modules.

- The game should allow users to guess a hidden word by selecting letters, while keeping track of wrong guesses and remaining chances.
- The game should support multiple word categories (such as fruits, animals, and countries), and each category should allow difficulty levels (easy, medium, hard).
- If the user selects a random category, the game should automatically choose one and display it.

Block Diagram



START
Initiallze:
- word_list
- select random word
- set lives = number allowed
- create blank display
Display current blanks and lives remaining
Take input: guess a letter
Check if word is complete?
Correct guess
Wrong guess
Reveal letter - Update blanks
Decrease lives - Show hangman
Check if word is complete?
YES
NO
Player wins
Check lives remaining
If lives == 0 → Player loses
Display Result

Approach Used

The game is divided into four separate modules, each handling a specific role:

logic.py

Contains the HangmanGame class responsible for all game rules:

- Validating guesses
- Tracking chances
- Revealing letters
- Determining win/loss
- Managing game state

wordlist.py

Manages all categories and difficulty settings and selects a random word.

ui.py

Builds the Tkinter graphical user interface:

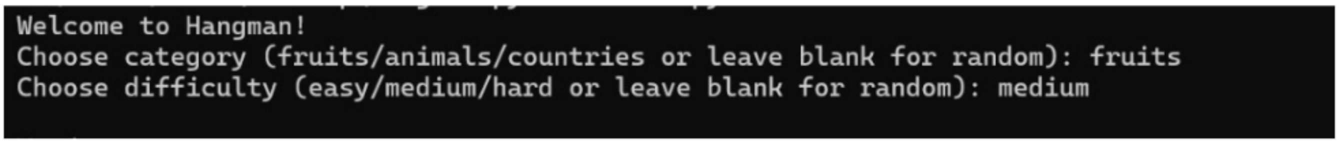
- Screens for welcome, category selection, and game play
- Clickable letter buttons
- Hangman ASCII drawing
- Messages and animations
- Game restart logic

main.py

Runs a console version that interacts only with the logic and wordlist modules.

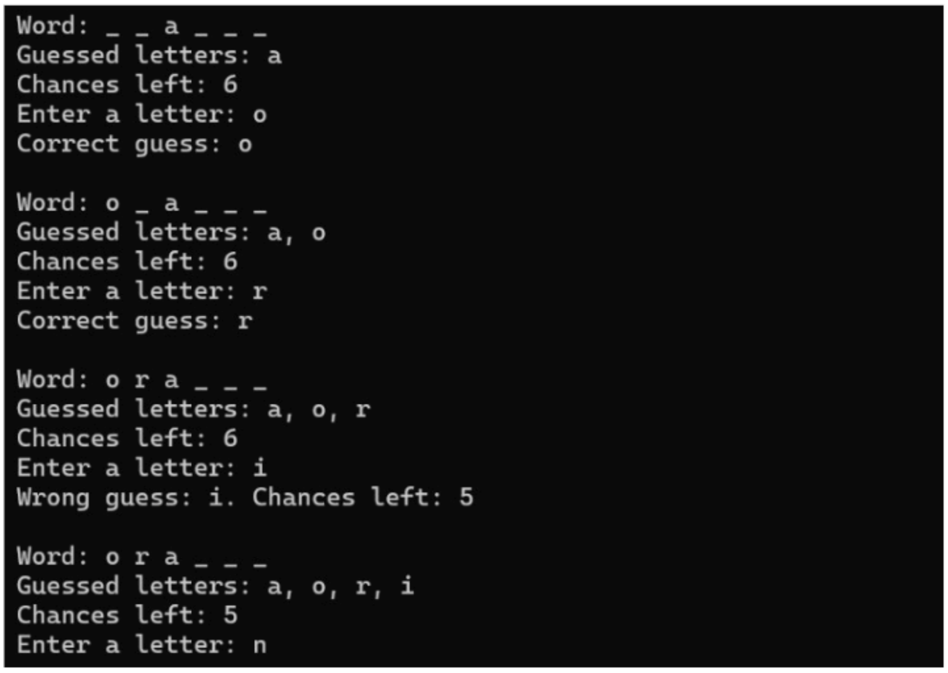
Sample Input/Output

Input:

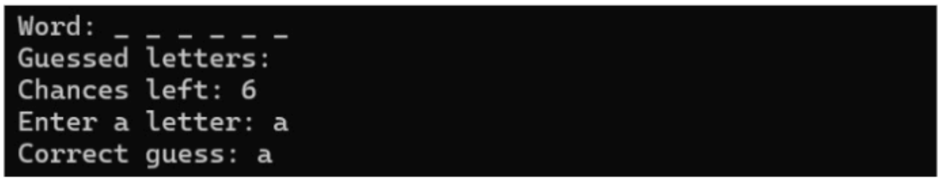


```
Welcome to Hangman!
Choose category (fruits/animals/countries or leave blank for random): fruits
Choose difficulty (easy/medium/hard or leave blank for random): medium
```

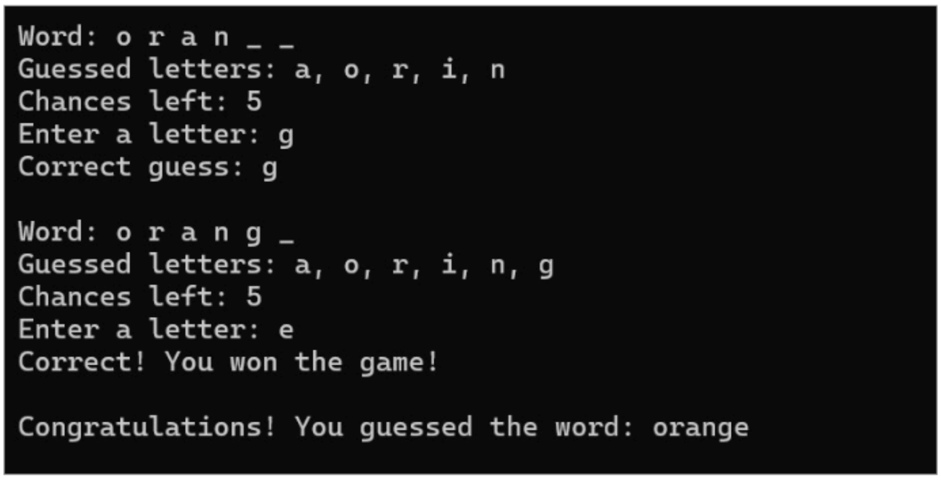
Output:



```
Word: _ _ a _ _ _
Guessed letters: a
Chances left: 6
Enter a letter: o
Correct guess: o

Word: o _ a _ _ _
Guessed letters: a, o
Chances left: 6
Enter a letter: r
Correct guess: r

Word: o r a _ _ _
Guessed letters: a, o, r
Chances left: 6
Enter a letter: i
Wrong guess: i. Chances left: 5

Word: o r a _ _ _
Guessed letters: a, o, r, i
Chances left: 5
Enter a letter: n
```



```
Word: _ _ _ _ _ _
Guessed letters:
Chances left: 6
Enter a letter: a
Correct guess: a
```



```
Word: o r a n _ _
Guessed letters: a, o, r, i, n
Chances left: 5
Enter a letter: g
Correct guess: g

Word: o r a n g _
Guessed letters: a, o, r, i, n, g
Chances left: 5
Enter a letter: e
Correct! You won the game!

Congratulations! You guessed the word: orange
```

Challenges Faced

Integrating Multiple Modules
Managing four separate files (logic, UI, wordlist, main) required careful structuring to avoid circular imports and maintain clean communication between modules.

Connecting UI With Game Logic
Updating frames, button states, and game visuals after each guess needed multiple adjustments.

Category & Difficulty Handling
Implementing the “random” option required secretly selecting a category, applying difficulty filters, and syncing everything with the UI so the game remained consistent.

UI Updates
Refreshing hangman ASCII art, buttons, guessed letters, and messages caused layout issues initially. Ensuring smooth, bug-free updates after every guess took time.

Preventing Duplicate Guesses
Disabling letter buttons, changing their color, and blocking repeated guesses required extra logic — especially when resetting the game.

Bug-Free Game Reset
“Play Again” had to properly reset all game states without leftover data. This required reorganizing the reset flow.

Uploading to GitHub
Learning Git basics—initializing, committing, pushing, fixing path/remote errors—was a challenge at first.