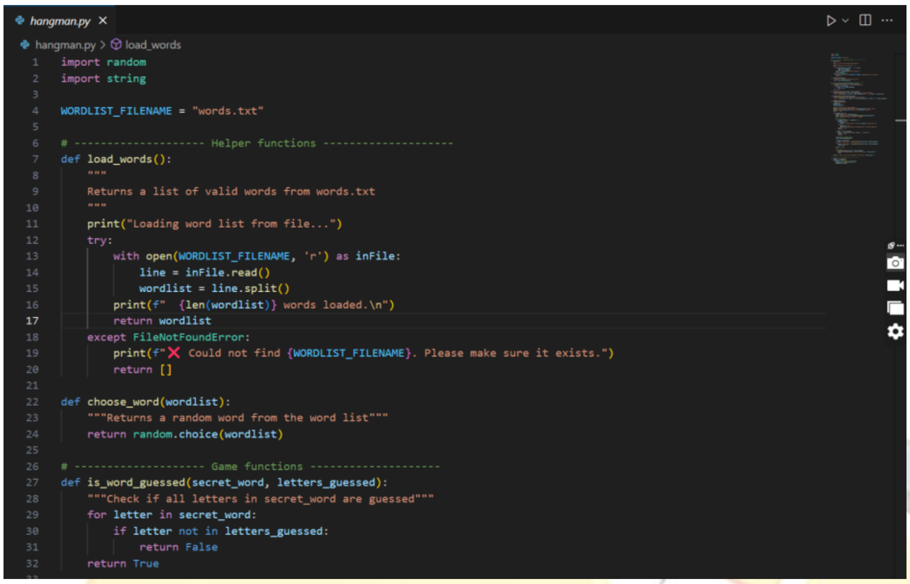
Name:	**Aleena Zainab**
Intern ID:	**TN/IN02/PY/005**
Task no:	**Hang Man Game**

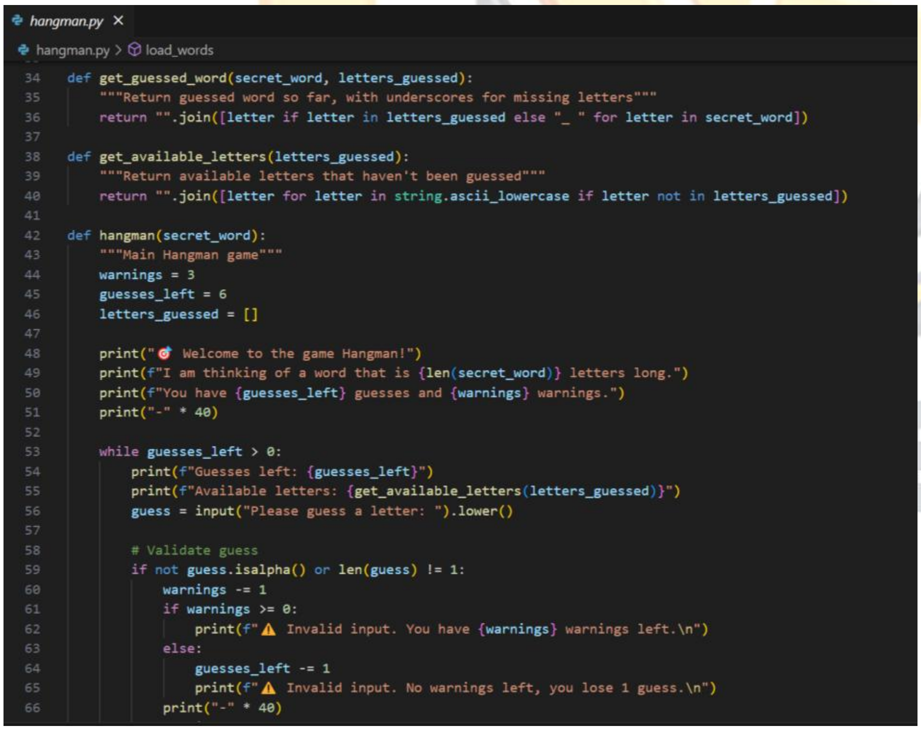
Project: HangMan Game

Code:



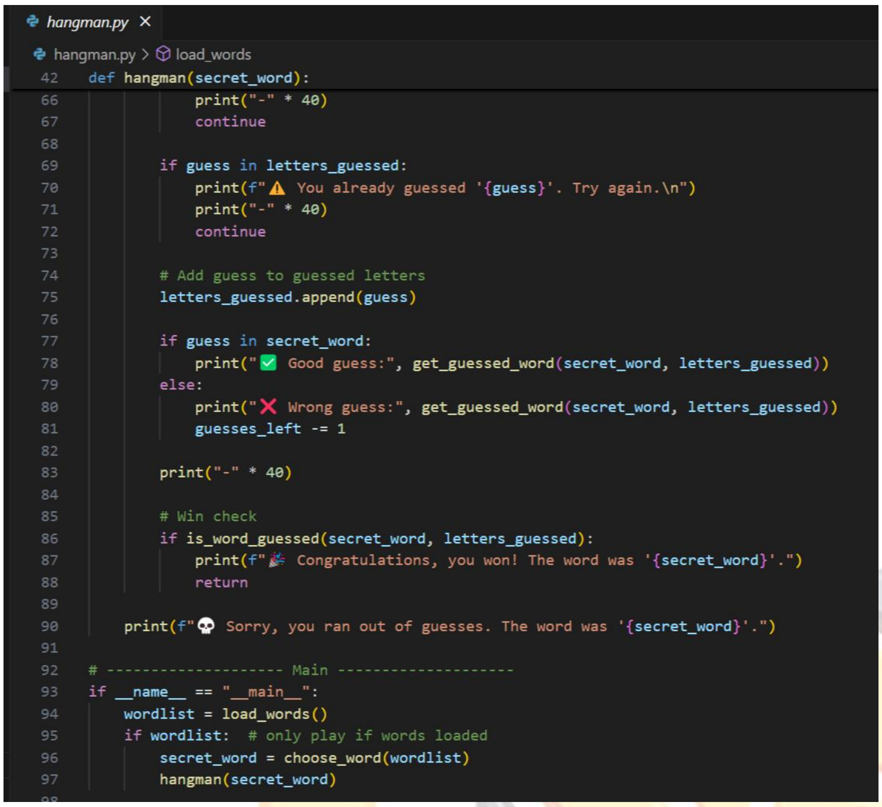
hangman.py > load_words

```python
import random
import string

WORDLIST_FILENAME = "words.txt"

# ------------------- Helper functions -------------------
def load_words():
    """
    Returns a list of valid words from words.txt
    """
    print("Loading word list from file...")
    try:
        with open(WORDLIST_FILENAME, 'r') as inFile:
            line = inFile.read()
            wordlist = line.split()
        print(f"  {len(wordlist)} words loaded.\n")
        return wordlist
    except FileNotFoundError:
        print(f"❌ Could not find {WORDLIST_FILENAME}. Please make sure it exists.")
        return []

def choose_word(wordlist):
    """Returns a random word from the word list"""
    return random.choice(wordlist)

# ------------------- Game functions -------------------
def is_word_guessed(secret_word, letters_guessed):
    """Check if all letters in secret_word are guessed"""
    for letter in secret_word:
        if letter not in letters_guessed:
            return False
    return True
```



hangman.py > load_words

```python
def get_guessed_word(secret_word, letters_guessed):
    """Return guessed word so far, with underscores for missing letters"""
    return "".join([letter if letter in letters_guessed else "_ " for letter in secret_word])

def get_available_letters(letters_guessed):
    """Return available letters that haven't been guessed"""
    return "".join([letter for letter in string.ascii_lowercase if letter not in letters_guessed])

def hangman(secret_word):
    """Main Hangman game"""
    warnings = 3
    guesses_left = 6
    letters_guessed = []

    print("🎯 Welcome to the game Hangman!")
    print(f"I am thinking of a word that is {len(secret_word)} letters long.")
    print(f"You have {guesses_left} guesses and {warnings} warnings.")
    print("-" * 40)

    while guesses_left > 0:
        print(f"Guesses left: {guesses_left}")
        print(f"Available letters: {get_available_letters(letters_guessed)}")
        guess = input("Please guess a letter: ").lower()

        # Validate guess
        if not guess.isalpha() or len(guess) != 1:
            warnings -= 1
            if warnings >= 0:
                print(f"⚠ Invalid input. You have {warnings} warnings left.\n")
            else:
                guesses_left -= 1
                print(f"⚠ Invalid input. No warnings left, you lose 1 guess.\n")
            print("-" * 40)
```

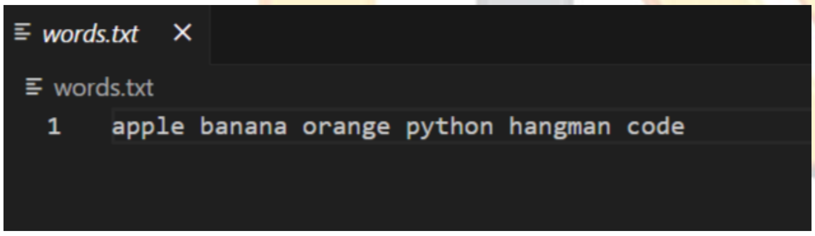


hangman.py

hangman.py > load_words

```python
def hangman(secret_word):
            print("-" * 40)
            continue

        if guess in letters_guessed:
            print(f"⚠️ You already guessed '{guess}'. Try again.\n")
            print("-" * 40)
            continue

        # Add guess to guessed letters
        letters_guessed.append(guess)

        if guess in secret_word:
            print("✅ Good guess:", get_guessed_word(secret_word, letters_guessed))
        else:
            print("❌ Wrong guess:", get_guessed_word(secret_word, letters_guessed))
            guesses_left -= 1

        print("-" * 40)

        # Win check
        if is_word_guessed(secret_word, letters_guessed):
            print(f"🎉 Congratulations, you won! The word was '{secret_word}'.")
            return

    print(f"💀 Sorry, you ran out of guesses. The word was '{secret_word}'.")

# -------------------- Main --------------------
if __name__ == "__main__":
    wordlist = load_words()
    if wordlist:  # only play if words loaded
        secret_word = choose_word(wordlist)
        hangman(secret_word)
```

Output.txt:

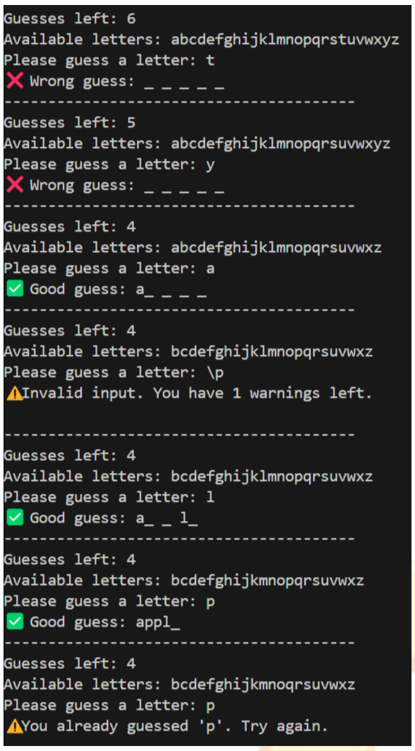


words.txt

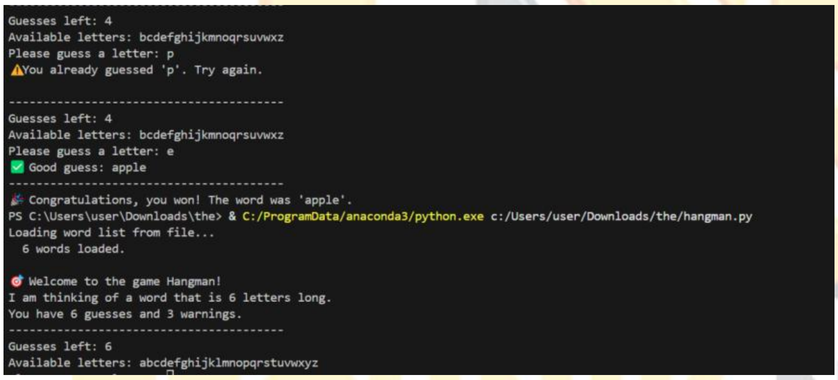
words.txt

```
apple banana orange python hangman code
```

Output:

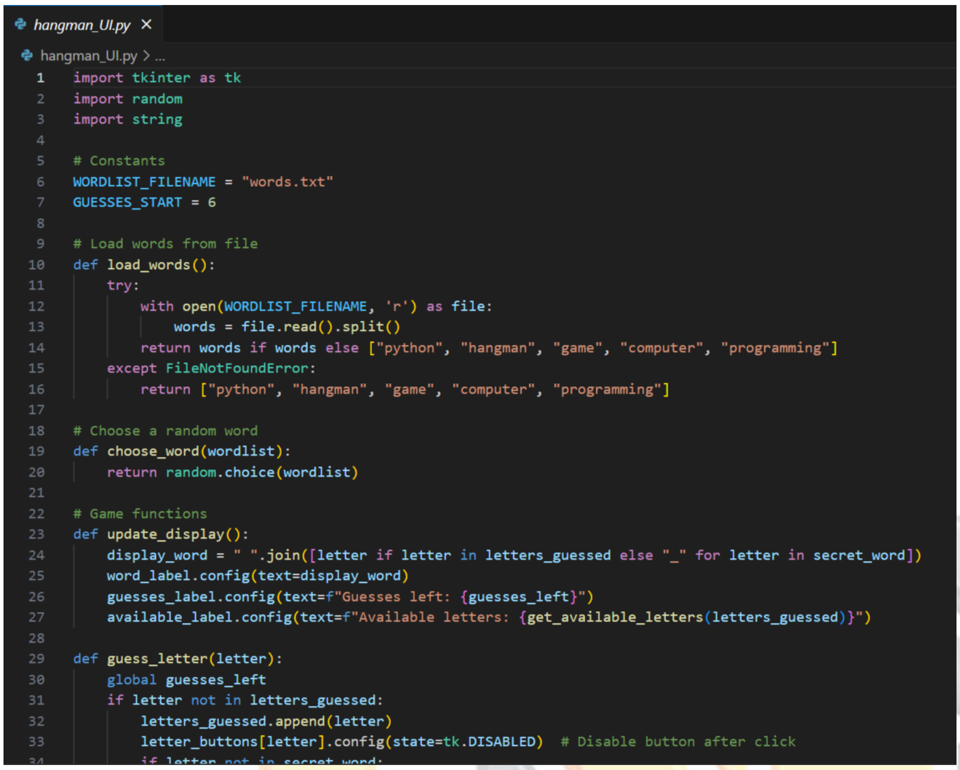


```
Guesses left: 6
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: t
❌ Wrong guess: _ _ _ _ _
----------------------------------------
Guesses left: 5
Available letters: abcdefghijklmnopqrsuvwxyz
Please guess a letter: y
❌ Wrong guess: _ _ _ _ _
----------------------------------------
Guesses left: 4
Available letters: abcdefghijklmnopqrsuvwxz
Please guess a letter: a
✅ Good guess: a_ _ _ _
----------------------------------------
Guesses left: 4
Available letters: bcdefghijklmnopqrsuvwxz
Please guess a letter: \p
⚠️Invalid input. You have 1 warnings left.

----------------------------------------
Guesses left: 4
Available letters: bcdefghijklmnopqrsuvwxz
Please guess a letter: l
✅ Good guess: a_ _ l_
----------------------------------------
Guesses left: 4
Available letters: bcdefghijkmnopqrsuvwxz
Please guess a letter: p
✅ Good guess: appl_
----------------------------------------
Guesses left: 4
Available letters: bcdefghijkmnoqrsuvwxz
Please guess a letter: p
⚠️You already guessed 'p'. Try again.
```



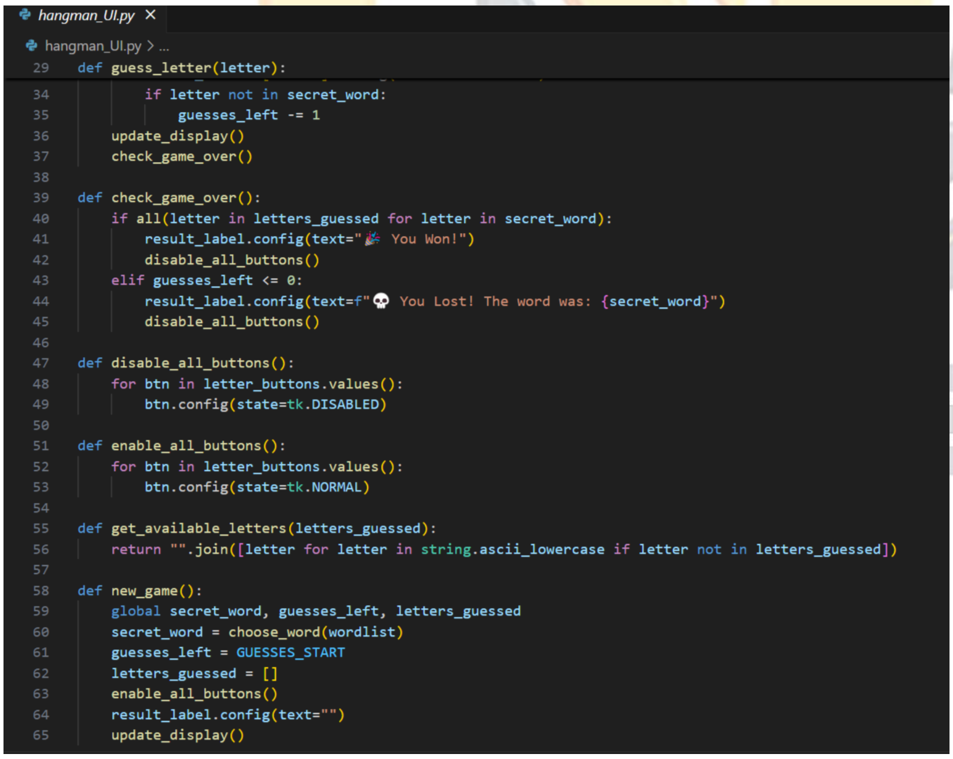
```
Guesses left: 4
Available letters: bcdefghijkmnoqrsuvwxz
Please guess a letter: p
⚠️You already guessed 'p'. Try again.

----------------------------------------
Guesses left: 4
Available letters: bcdefghijkmnoqrsuvwxz
Please guess a letter: e
✅ Good guess: apple
----------------------------------------
🎉 Congratulations, you won! The word was 'apple'.
PS C:\Users\user\Downloads\the> & C:/ProgramData/anaconda3/python.exe c:/Users/user/Downloads/the/hangman.py
Loading word list from file...
  6 words loaded.

🎯 Welcome to the game Hangman!
I am thinking of a word that is 6 letters long.
You have 6 guesses and 3 warnings.
----------------------------------------
Guesses left: 6
Available letters: abcdefghijklmnopqrstuvwxyz
```

I used GUI to make it look like a Game:



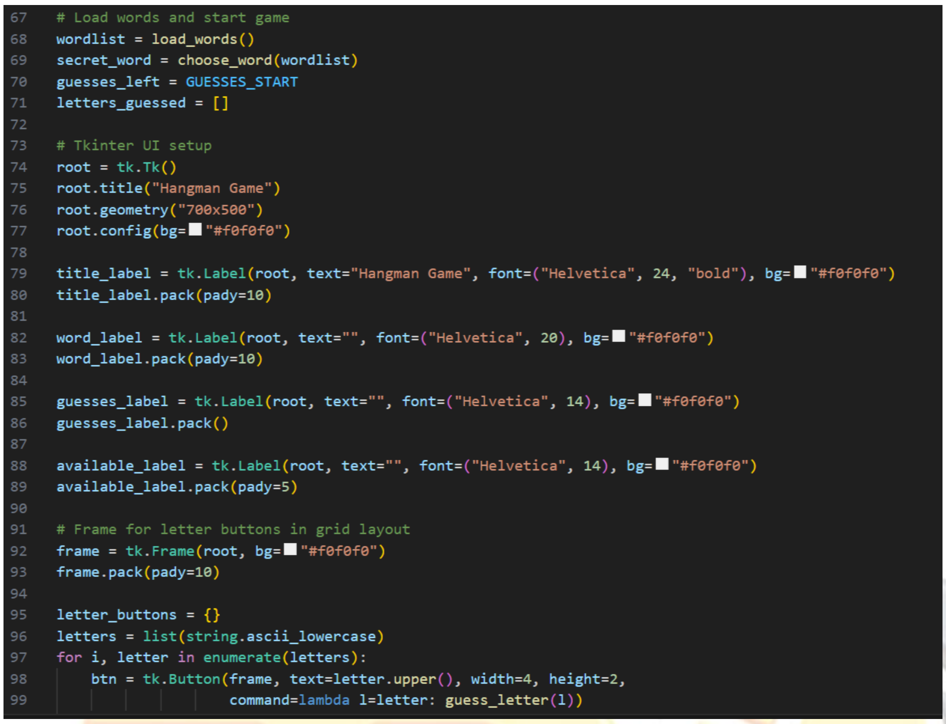
hangman_UI.py

```python
import tkinter as tk
import random
import string

# Constants
WORDLIST_FILENAME = "words.txt"
GUESSES_START = 6

# Load words from file
def load_words():
    try:
        with open(WORDLIST_FILENAME, 'r') as file:
            words = file.read().split()
        return words if words else ["python", "hangman", "game", "computer", "programming"]
    except FileNotFoundError:
        return ["python", "hangman", "game", "computer", "programming"]

# Choose a random word
def choose_word(wordlist):
    return random.choice(wordlist)

# Game functions
def update_display():
    display_word = " ".join([letter if letter in letters_guessed else "_" for letter in secret_word])
    word_label.config(text=display_word)
    guesses_label.config(text=f"Guesses left: {guesses_left}")
    available_label.config(text=f"Available letters: {get_available_letters(letters_guessed)}")

def guess_letter(letter):
    global guesses_left
    if letter not in letters_guessed:
        letters_guessed.append(letter)
        letter_buttons[letter].config(state=tk.DISABLED)  # Disable button after click
        if letter not in secret_word:
```

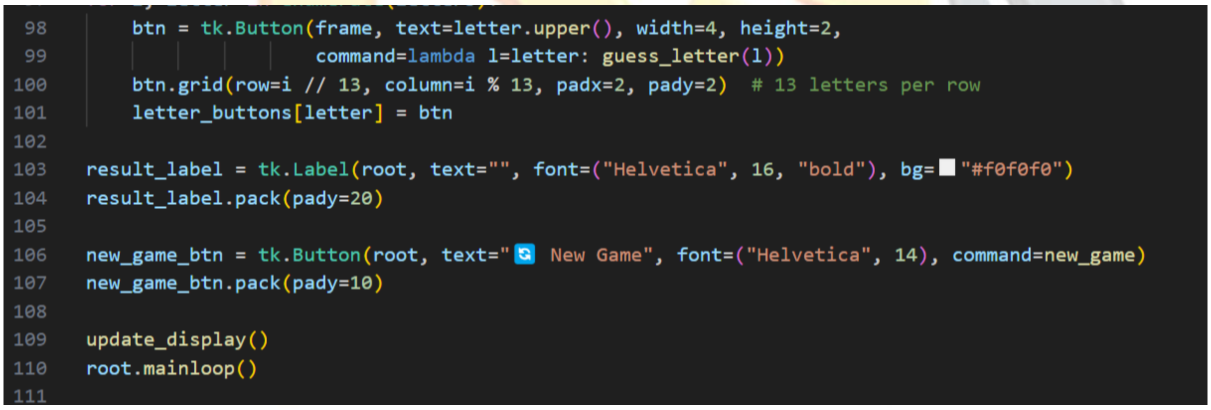


hangman_UI.py

```python
def guess_letter(letter):
        if letter not in secret_word:
            guesses_left -= 1
    update_display()
    check_game_over()

def check_game_over():
    if all(letter in letters_guessed for letter in secret_word):
        result_label.config(text="🎉 You Won!")
        disable_all_buttons()
    elif guesses_left <= 0:
        result_label.config(text=f"💀 You Lost! The word was: {secret_word}")
        disable_all_buttons()

def disable_all_buttons():
    for btn in letter_buttons.values():
        btn.config(state=tk.DISABLED)

def enable_all_buttons():
    for btn in letter_buttons.values():
        btn.config(state=tk.NORMAL)

def get_available_letters(letters_guessed):
    return "".join([letter for letter in string.ascii_lowercase if letter not in letters_guessed])

def new_game():
    global secret_word, guesses_left, letters_guessed
    secret_word = choose_word(wordlist)
    guesses_left = GUESSES_START
    letters_guessed = []
    enable_all_buttons()
    result_label.config(text="")
    update_display()
```



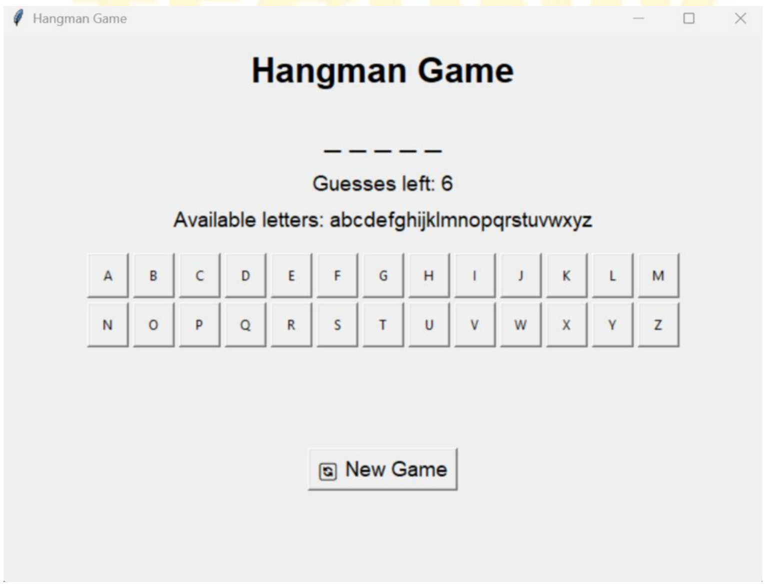
```python
# Load words and start game
wordlist = load_words()
secret_word = choose_word(wordlist)
guesses_left = GUESSES_START
letters_guessed = []

# Tkinter UI setup
root = tk.Tk()
root.title("Hangman Game")
root.geometry("700x500")
root.config(bg="#f0f0f0")

title_label = tk.Label(root, text="Hangman Game", font=("Helvetica", 24, "bold"), bg="#f0f0f0")
title_label.pack(pady=10)

word_label = tk.Label(root, text="", font=("Helvetica", 20), bg="#f0f0f0")
word_label.pack(pady=10)

guesses_label = tk.Label(root, text="", font=("Helvetica", 14), bg="#f0f0f0")
guesses_label.pack()

available_label = tk.Label(root, text="", font=("Helvetica", 14), bg="#f0f0f0")
available_label.pack(pady=5)

# Frame for letter buttons in grid layout
frame = tk.Frame(root, bg="#f0f0f0")
frame.pack(pady=10)

letter_buttons = {}
letters = list(string.ascii_lowercase)
for i, letter in enumerate(letters):
    btn = tk.Button(frame, text=letter.upper(), width=4, height=2,
                    command=lambda l=letter: guess_letter(l))
```



```python
    btn = tk.Button(frame, text=letter.upper(), width=4, height=2,
                    command=lambda l=letter: guess_letter(l))
```

```python
    btn.grid(row=i // 13, column=i % 13, padx=2, pady=2)  # 13 letters per row
    letter_buttons[letter] = btn

result_label = tk.Label(root, text="", font=("Helvetica", 16, "bold"), bg="#f0f0f0")
result_label.pack(pady=20)

new_game_btn = tk.Button(root, text="🔄 New Game", font=("Helvetica", 14), command=new_game)
new_game_btn.pack(pady=10)

update_display()
root.mainloop()
```

Output:



```
Hangman Game

_ _ _ _ _
Guesses left: 6
Available letters: abcdefghijklmnopqrstuvwxyz
```

Task 1: Loading Words

Code Section: `load_words()`

- Reads words from `words.txt` into a list.
- Example words: `apple, banana, orange, python, hangman, code`.
- Prints the total number of words loaded.

Purpose: Prepares the word bank for the game.
Challenge: Initially, if `words.txt` was not in the correct directory, the game showed:

e.g. `Could not find words.txt. Please make sure it exists.`

Solution: Place `words.txt` in the same folder as `hangman.py`.

Task 2: Choosing the Secret Word

Code Section: `choose_word(wordlist)`

- Selects a random word from the loaded word list.
- Example: If the list contains `[apple, python, code]`, it may pick `"python"`.

Purpose: Ensures randomness in each new game.

Task 3: Core Game Functions

1. **`is_word_guessed()`** → Checks if all letters of the secret word have been guessed.
2. **`get_guessed_word()`** → Shows progress, e.g., guessing `"a"` in `"apple"` gives:
3. `a_ _ _ _`
4. **`get_available_letters()`** → Tracks unused letters from the alphabet.

Purpose: These functions handle game logic for guesses and display.

Task 4: Main Game Loop (`hangman()`)

- Player starts with **6 guesses** and **3 warnings**.
- Each round shows:
  - Remaining guesses
  - Available letters
  - Current progress on the secret word
- Player enters a letter guess, which updates the game state.
- Win condition: all letters guessed.

- Lose condition: guesses drop to zero.

Sample Output (simulation):

```
🎯 Welcome to the game Hangman!
I am thinking of a word that is 6 letters long.
You have 6 guesses and 3 warnings.
----------------------------------------
Guesses left: 6
Available letters: abcdefghijklmnopqrstuvwxyz
Please guess a letter: a
✅ Good guess: a_ _ _ _
```

Challenges Faced

1. **File Not Found Error:** Game couldn't find `words.txt` at first.
   - **Solution:** Ensured the file is in the same directory as the script.
2. **Input Validation:** If user enters a number or repeated letter, the game gives a warning and reduces guesses if warnings run out.
   - **Solution:** Used `isalpha()` and tracking guessed letters.

Conclusion

This project successfully implements a **text-based Hangman game** with input validation, warnings, and random word selection. The main challenge was handling file paths and input validation, which were solved through proper file placement and condition checks.