# WORDS UNSCRAMBLE GAME

A project report submitted in partial fulfillment of the requirements

For the award of credits to

Python a Skill Enhancement Course of

**Bachelor of Technology**

In

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM)**

By

**RESHMA.KODAVALI**

**23BQ1A4281**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM)**

**VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY**

**(Approved by AICTE and permanently affiliated to JNTUK)**

**Accredited by NBA and NAAC with 'A' Grade**

**NAMBUR (V), PEDAKAKANI (M), GUNTUR-522 508**

**DECEMBER 2024**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM)**

**VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY: NAMBUR**

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA**

# CERTIFICATE

This is to certify that the project titled "**WORDS UNSCRAMBLE GAME**" is a bonafide record of work done by **Ms.Reshma Kodavali** under the guidance of **Mr. M. Pardha Saradhi**, **Associate Professor** in partial fulfillment of the requirement for the award of credits to **Python -** a skill enhancement course of Bachelor of Technology in Computer Science & Engineering – Artificial Intelligence & Machine Learning (CSM), JNTUK during the academic year 2024-25.

| | |
|---|---|
| **M. Pardha Saradhi** | **Prof. K. Suresh Babu** |
| **Course Instructor** | **Head of the Department** |

# DECLARATION

I, **Kodavali Reshma (23BQ1A4281),** hereby declare that the Project Report entitled "**WORDS UNSCRAMBLE GAME**" done by me under the guidance of **Mr.M. Pardha Saradhi, Associate Professor** is submitted in partial fulfillment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM).**

DATE : **SIGNATURE OF THE CANDIDATE**

PLACE : **KODAVALI.RESHMA**

# ACKNOWLEDGEMENT

We express our sincere thanks to the Chairman, **Vasireddy Venkatadri Institute of Technology, Sri Vasireddy Vidya Sagar** for providing us well equipped infrastructure and environment.

We thank **Dr. Y. Mallikarjuna Reddy**, Principal, Vasireddy Venkatadri Institute of Technology, Nambur, for providing us the resources for carrying out the project.

We express our sincere thanks to **Dr. K. Giribabu**, Dean of Academics for providing support and stimulating environment for developing the project.

Our sincere thanks to **Dr. K. Suresh Babu**, Head of the Department, Department of CSM, for his co-operation and guidance which helped us to make our project successful and complete in all aspects.

We also express our sincere thanks and are grateful to our guide **Mr. M. Pardha Saradhi**, Associate Professor, Department of CSM, for motivating us to make our project successful and fully complete. We are grateful for his precious guidance and suggestions.

We also place our floral gratitude to all other teaching staff and lab technicians for their constant support and advice throughout the project.

**KODAVALI RESHMA**

23BQ1A4281

# TABLE OF CONTENTS

# List of Figures

# LIST OF TABLES

# ABSTRACT

The project focuses on developing an interactive word unscramble game using Python and the pygame library. The game features a graphical user interface (GUI) that enhances the gameplay experience, providing an engaging and user-friendly platform.

**Word Scrambling and Selection:** The system randomly selects words from a predefined list, scrambles them, and presents them to the user as a challenge.

**User Interaction:** The user guesses the scrambled word, receiving real-time feed back on the correctness of their guesses.

**Visual Feedback and Scoring:** The game tracks the user's score and provides visual feedback on their performance. A countdown timer adds urgency to each round.

**Graphical User Interface:** The gameplay is rendered using pygame, featuring a clear and intuitive interface with turn-based mechanics where the system continuously presents scrambled words for the user to guess.

This single-player game offers an engaging experience where players test their vocabulary and problem-solving skills. The implementation leverages Python's libraries to create a smooth, interactive environment. Upon running the program, the game window displays the scrambled words, real-time feedback on user inputs, a timer, and the user's score.

# CHAPTER - 1

# INTRODUCTION

*Fig 1 : Scramble*

The Word Scramble Game is an educational and entertaining project that blends cognitive challenge with gameplay, aimed at improving vocabulary, problem-solving, and memory skills. This game is designed using Python, leveraging libraries like Pygame for the graphical interface and NLTK (Natural Language Toolkit) for a robust set of English words. The game revolves around players unscrambling jumbled letters to form valid words, with the game providing real-time feedback and scoring.

The game operates on a simple yet effective model, where players attempt to guess the original word from a scrambled version. It is suitable for both casual gamers who enjoy fun word s and individuals who wish to improve their linguistic abilities. The combination of a turn-based mechanic for both players and AI ensures a dynamic and competitive atmosphere.

By playing this game, players not only engage with fun challenges but also enhance their vocabulary and spelling skills. Whether for educational purposes or simply for entertainment, the Word Scramble Game serves both as a valuable learning tool and an enjoyable way to pass the time.

## 1.1 Aim of the Project

The **Word Scramble Game** is designed with the primary objective of combining fun and education to create a unique, interactive experience. This game serves not only as an entertainment tool but also as a means of improving vocabulary, cognitive function, and problem-solving skills. By leveraging **Python** and libraries like **Pygame** for the graphical interface and **NLTK** (Natural Language Toolkit) for word generation, the project aims to provide an engaging and visually appealing platform where players can test and improve their skills.

### 1.1.1 Creating an Engaging Game Environment

The Word Scramble Game aims to provide a visually appealing, user-friendly interface with sections for scrambled words, input, scoreboards, and hints. Designed for all ages, it features colorful graphics, simple gameplay, and a responsive design with smooth animations. The goal is to create an engaging experience that enhances vocabulary and problem-solving skills.

### 1.1.2. Promoting Vocabulary Enhancement

***Fig 2 : Enhancing Vocabulary***

The Word Scramble Game is designed to improve players' vocabulary through engaging gameplay and practical learning opportunities.

**Exposure to New Words**: Players are introduced to new words each time they unscramble a , gradually expanding their vocabulary through repeated exposure.

**Increasing Difficulty**: As players advance, the game presents longer and more complex words, ensuring they remain challenged and continue improving their skills.

**Active Learning**: By interacting with scrambled words, players practice spelling and word recognition, strengthening both written and verbal language abilities.

**Practical Benefits**: Each session leaves players with an enhanced vocabulary, translating into improved communication skills for everyday life.

### 1.1.3 Encouraging Problem-Solving and Cognitive Development

The Word Scramble Game promotes critical thinking, memory improvement, and problem-solving through engaging words.

**Pattern Recognition:** Players develop their ability to identify letter patterns and word structures, sharpening their recognition skills.

**Memory Enhancement:** Repeated exposure to word s strengthens memory recall and retention, improving the ability to recognize letter combinations and word structures.

**Logical Thinking:** Solving word s challenges players to logically determine how letters fit together, fostering analytical and decision-making skills.

**Broader Cognitive Benefits:** This cognitive workout not only enhances language skills but also boosts critical thinking and memory for use in daily life.

### 1.1.4 Providing Educational Support through Hints

The game offers hints to assist players when they struggle with a word, ensuring a balanced and enjoyable experience.

**Guided Learning:** Hints like revealing the first and last letters help players progress without frustration.

**Encouraging Persistence:** Hints make challenging s manageable, motivating players to continue learning and improving.

This feature ensures an enjoyable and supportive learning experience for all players.

### 1.1.5 Utilizing Python Libraries for Efficient Development

The game leverages Python's readability, flexibility, and extensive libraries to ensure efficient development.

**Pygame**: Pygame is crucial for the graphical interface, handling user input, displaying text, and providing visual feedback, ensuring smooth and responsive gameplay.

**NLTK**: The NLTK library offers a vast English word set, enabling integration of a wide range of words to increase both difficulty and educational value.

These libraries ensure the game is scalable, maintainable, and provides a solid foundation for future updates and improvements.

## Expected Outcomes

By the end of the project, the Word Scramble Game will achieve key outcomes:

**Educational Tool:** The game will be an engaging tool for learning new words and improving cognitive skills.

**Cognitive Development:** Players will enhance problem-solving, memory, and word association abilities.

**Entertainment and Learning:** Players will enjoy a competitive experience while expanding their vocabulary.

**Scalability:** The game's design will allow for future updates, including more words, difficulty levels, and new features.

### 1.2 Problem Statement

The Word Scramble Game aims to provide an engaging, educational platform for enhancing vocabulary and cognitive skills. It challenges players to recall word structures, spelling, and vocabulary by unscrambling letters, helping them continuously improve their language abilities. The game also promotes cognitive function by stimulating brain areas responsible for pattern recognition, logical reasoning, and memory. With turn-based gameplay, hints, and a scoring system, it introduces friendly competition that encourages players to continue playing and improving. The hint system provides support for players who may struggle, offering partial information to prevent frustration and foster learning.

**Input :** The inputs to the Word Scramble Game include data inputs and user interactions, both essential for smooth gameplay.

**Data Inputs:**

**Word List (NLTK Word Corpus):**

The NLTK library provides a vast collection of over 200,000 English words, ensuring a continuous supply of diverse words for the game. Each round, a word is randomly selected from this corpus, scrambled, and presented to the player as a challenge. The randomness of word selection ensures a unique experience in every round. **Hint Data**:

The hint system provides partial information to assist players without revealing the entire word. It reveals the first and last letters of the word, helping players start solving and triggering their memory of possible words. For example, for the word "scramble," the hint would show "S" as the first letter and "E" as the last, narrowing down the possibilities.

**User Interactions:**

**Player Guesses**:

Player input is vital to the game. The player types their guess for the scrambled word, and the system checks it against the correct word. If the guess is correct, points are awarded, and feedback is given. If incorrect, feedback is provided, allowing the player to keep guessing or move to the next round.

**Game Controls :**

The Backspace key allows players to delete the last character they entered in case of a mistake. The Hint Request feature lets players ask for help by revealing the first and last letter of the word, ensuring they can continue playing and learning even when stuck.

## Output :

The expected outputs from the Word Scramble Game can be broken down into several categories: gameplay feedback, game progression, and interface updates.

### 1.2.1 Gameplay Feedback:

**Correct/Incorrect Feedback**:

When the player guesses correctly, the game displays "Correct!" and awards points, providing immediate positive feedback. If the guess is incorrect, the game shows the correct word with a message like "Incorrect, the correct word was [word]," helping the player learn and continue.

**Hints**:
The hint system is triggered, revealing the first and last letters of the word to help narrow down guesses.
**Score Update**:
The player's score is updated after each guess, with correct guesses earning points. In multiplayer mode, both players' scores are visible, adding competition.
At the end of each round, the score is displayed, and the game moves to the next round or allows players to start a new round

**Game Progression:**
After each round, the word is scrambled again, and the round number and progress are displayed, maintaining engagement. The game continues until the player quits or a set number of rounds is completed, with the winner determined by the highest score. In multiplayer mode, turns alternate between players or between the player and computer, ensuring a competitive and dynamic experience..

**Interface Updates:**
The game's interface displays key information like the scrambled word, player guesses, feedback, score, and round number, updating in real-time to show progress. The visual design, powered by Pygame, includes updated fonts, colors, and layout for clarity. User inputs, such as guesses or hint requests, are immediately reflected on the screen, with visual feedback provided in real-time.

**Steps Involved:**
The development of the **Word Scramble Game** involves a series of methodical steps. Each step builds upon the previous to ensure a cohesive and functioning game.

**Step 1: Setup the Game Environment:**
First, initialize Pygame using pygame.init() to set up the game window and handle user input. Then, create the game window with pygame.display.set_mode(), and define fonts and colors for a consistent user interface**.**

**Step 2: Word List and Scrambling:**

Words are randomly selected from the NLTK Word Corpus using the nltk.corpus.words module, and random.choice() selects a word. The selected word is then scrambled using Python's random.sample() method, which rearranges the letters to create a scrambled version for the player to solve.

**Step 3: Game Mechanics:**

When the player types their guess, the system checks it against the original word. If the player requests a hint, the system reveals the first and last letter of the word. A score variable is maintained, updating after each correct guess and displayed on the screen.

**Step 4: Game Progression and Testing and Debugging**

Each round begins with a new scrambled word, and after the player guesses, the next word is selected. The game ends after a set number of rounds or when the player quits, displaying the final score and offering a restart option. Thorough testing ensures all game mechanics function correctly, and any identified bugs are fixed to ensure smooth gameplay**.**

## Input Files :

**Source of Word Data**

The word list in the game is sourced from the NLTK Word Corpus, a reliable open-source collection of English words. It is accessed via the nltk.corpus.words module and the words.words() function, providing tens of thousands of entries. The corpus includes a range of vocabulary from simple words like "cat" and "dog" to more complex ones like "exacerbate" and "juxtaposition," catering to different difficulty levels. The NLTK library must be installed and its datasets downloaded using nltk.download().

## 1.3 Scope of the Project

Words unscramble game using Pygame and NLTK. It displays a scrambled 5-letter word and asks the player to guess the correct word within a 60-second timer. The game provides hints, tracks the player's score, and updates the screen with the word, score, timer, and messages. Players input guesses using the keyboard, and feedback is given on correctness. After each round, the word and timer reset for the next round. The game is single-player and offers a fun, timed challenge to practice vocabulary.

### 1.3.1 Applications

**Vocabulary Learning**: Helps students and language learners improve spelling and word recognition.

**Brain Training**: Offers cognitive exercises for memory and problem-solving. **Casual Entertainment**: Provides a fun, quick challenge for casual players. **Classroom Activities**: Can be used in classrooms for educational games or group activities.

**Competitive Play**: Suitable for timed challenges and competitive word-solving. **Game Development**: A foundation for building more complex word-based games or apps.

### 1.3.2 Use cases

**Word Scramble Game Mechanics**:

The game selects a random word from a list of 5-letter words using the NLTK corpus. It scrambles the selected word by rearranging its letters to create a challenge. The game ensures that the scrambled word is not the same as the original word, using a recursive check.

**Timer and Turn-based System**: The game has a time limit of 60 seconds for each turn. A countdown timer is displayed, and the round resets when the timer runs out or when the player makes a guess. After each round, a new word is selected, and the scramble process starts again.

**Scoring System**: The player receives a point each time they correctly guess the word within the allotted time. The score is displayed on the screen, and after each guess, the game provides feedback on whether the player was correct or not.

**Hints**: Players are given a hint about the word by showing its first and last letters. This helps players who might need a little extra guidance in figuring out the word.

**User Input**: Players type their guesses using the keyboard. The game displays the typed input on the screen. If players make a mistake, they can use the backspace key to delete characters and correct their guesses

# CHAPTER – 2

# LITERATURE REVIEW

## 2.1 Existing Methods

The problem of unscrambling words and forming meaningful words has been explored in various contexts, such as educational games, spelling tests, and word puzzle solvers. Existing methods generally employ algorithms for text manipulation, pattern matching, and word validation. Below is a brief overview:

### 2.1.1 Brute Force Word Matching:

**Method**: Generate all possible permutations of the scrambled word and match them against a dictionary.

**Advantages**: Guaranteed to find all valid matches.

**Disadvantages**: Computationally expensive, especially for longer words with many permutations.

### 2.1.2 Sorting-Based Matching:

**Method**: Sort the letters of both the scrambled word and the dictionary words. Words with the same sorted form are anagrams.

**Advantages**: Efficient and avoids unnecessary permutations.

**Disadvantages**: Limited to detecting anagrams and assumes the dictionary is well-optimized.

### 2.1.3Hashing-Based Matching:

**Method**: Use a hash function (e.g., frequency of each letter) to map words to unique hashes. Match the scrambled word's hash with dictionary entries.

**Advantages**: Highly efficient for lookup.

**Disadvantages**: Requires preprocessing and storage space for hash tables.

### 2.1.4Machine Learning for Context-Based Solving:

**Method**: Use language models (e.g., GPT, BERT) to predict possible words from scrambled input based on linguistic context.

**Advantages**: Can provide contextually appropriate solutions and handle partial or ambiguous inputs.

**Disadvantages**: Overkill for simple word unscrambling and requires training data.

### 2.2 Python Modules and Libraries Used

#### 2.2.1 Pygame

Pygame is a cross-platform Python library designed for creating video games and multimedia applications. It provides functionality for rendering graphics, playing sounds, and handling user inputs like keyboard and mouse events. In this project, Pygame serves as the backbone for the graphical interface, enabling the creation of an interactive and visually appealing game.

***Fig 3 : Pygame***

**Key Methods and Attributes Used:**

1. **`pygame.init()`**: Initializes all imported Pygame modules.

```
pygame.init()
```

2. **`pygame.display.set_mode((WIDTH, HEIGHT))`**: Creates a game window with specified dimensions.

```
screen = pygame.display.set_mode((800, 500))
```

3. **`pygame.display.set_caption("WORDS UNSCRAMBLE GAME"):`** Sets the title of the game window.

```
pygame.display.set_caption("My Game")
```

4. **`pygame.font.Font(None, 36):`** Loads a font object for rendering text. The None parameter uses the default font, and 36 specifies the font size.

```
FONT = pygame.font.Font(None, 36)
```

5. **`pygame.KEYDOWN`**: Represents the event of a key press. Used in the event loop to detect user inputs.

```
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN:
        print("Key Pressed")
```

6. **`pygame.K_RETURN, pygame.K_BACKSPACE:`** Constants for detecting specific key presses like the Enter (RETURN) and Backspace keys.

```
if event.key == pygame.K_RETURN:
    print("Enter key pressed")
```

7. **`screen.fill(WHITE):`** Fills the screen with a specific color (RGB tuple).

```
screen.fill((255, 255, 255))  # White
```

8. **`pygame.font.Font.render(text, antialias, color):`** Renders text with the specified anti-aliasing (True/False) and color.

```
rendered_text = FONT.render("Hello", True, (255, 0, 0))  # Red
```

9. **`pygame.display.flip():`**Updates the entire screen with the latest rendered graphics.

```
pygame.display.flip()
```

10. **`pygame.time.Clock().tick(fps):`** Controls the frame rate of the game loop.

```
clock = pygame.time.Clock()
clock.tick(30)  # Limit to 30 frames per second
```

### 2.2.2 Natural Language Toolkit (NLTK)

The Natural Language Toolkit (NLTK) is a comprehensive library widely used for natural language processing tasks. It offers tools for text processing, tokenization, stemming, and accessing linguistic datasets. In this project, NLTK is employed to provide a word list through its nltk.corpus.words module.

***Fig 4 : NLTK***

**Key Methods and Attributes Used:**

**`1.nltk.download('words'):`** Downloads the words corpus, which contains a large collection of English words.

```
import nltk
nltk.download('words')
```

2. **`nltk.corpus.words.words()`:** Returns a list of all English words in the corpus.

```
from nltk.corpus import words
word_list = words.words()
print(len(word_list))  # Total words in the corpus
```

**3. Filtering the Words List :** Filters the words corpus to include only lowercase words of length 5.

```
word_list = [word.lower() for word in words.words() if len(word) == 5]
```

### 2.2.3 Random

The random module in Python is a lightweight library used to introduce randomness and unpredictability into programs. In this project, it plays a vital role in selecting and scrambling words, ensuring that each game round is unique.

**Key Methods Used:**

**`1.random.sample(population, k)`:** Returns a new list with k random elements from the input sequence population.

```
shuffled_word = ''.join(random.sample("hello", len("hello")))
print(shuffled_word)  # e.g., "ohell"
```

**`2. random.choice(sequence)`:** Returns a random element from a non-empty sequence.

```
word = random.choice(["apple", "banana", "cherry"])
print(word)  # e.g., "banana"
```

By using pygame for graphics and user interaction, random for introducing unpredictability, and nltk for handling linguistic data, the project successfully delivers a fun and educational experience. This integration demonstrates the power of Python in game development and highlights its ability to merge logic, creativity, and functionality. The game serves as a foundation for further enhancements and a practical example of effective programming practices.

***Table* – 1 Existing Methods In Code**

| Method Name | Purpose | Excitement |
|---|---|---|
| scramble_word(word) | Scrambles the input word to create a random variation. | Adds a challenge by ensuring the word is scrambled differently each time, preventing predictable outcomes. |
| get_random_word() | Selects a random word from a filtered list of 5-letter words. | Keeps the game fresh and varied by randomly selecting a new word for each round. |
| get_hint(word) | Provides a textual hint about the word (first and last letter). | Offers a subtle hint that helps the player while preserving the challenge, adding a layer of strategic thinking |
| game_loop() | The core game logic loop, handling events, game updates, and rendering. | Manages the flow of the game, including timers, scoring, word challenges, and rendering, creating an engaging and dynamic experience for the player. |

This project builds upon existing methods for word scramble games, using basic algorithms like random shuffling for scrambling, hint generation, and scoring systems to track performance. These methods provide the foundation for creating an interactive, turn-based game that is both easy to implement and engaging.

Key Python libraries used include Pygame for GUI management and user input handling, and NLTK for accessing a corpus of English words to generate valid game words. Together, these modules enable efficient game logic, text rendering, and randomization.

The project successfully leverages these tools to create a dynamic and engaging word scramble game, combining real-time interaction with features like scoring and hints, offering a fun and educational experience.

# CHAPTER-3
# PROJECT

## 3.1 Materials:

**Libraries**:

**Pygame**: Used for developing the graphical user interface (GUI), including screen display, text rendering, user input handling, and game loop control.

**NLTK (Natural Language Toolkit)**: The nltk.corpus.words module is used to retrieve a list of English words to generate random words for the game.

**Random Module**: Utilized to generate scrambled versions of words by randomly shuffling the letters.

## 3.2 Methods:

The following methods and algorithms were employed in the development of the Word Scramble Game:

**Initialization**:

1. Pygame is initialized with `pygame.init()`.
2. The screen dimensions are set to 600x400 pixels.
3. A font for text rendering is created using `pygame.font.Font()`.
4. The NLTK word list is loaded via `nltk.corpus.words.words()`.

**Word Scrambling Algorithm**:
The function `scramble_word(word)` takes a word as input and uses random.sample() to shuffle the letters of the word.The algorithm ensures that the word is not returned in its original order by recursively re-scrambling the word if needed.This guarantees that the game presents a true scramble of the word and avoids returning the same word as the scrambled word.

**Word Generation**:

The function get_random_word() randomly selects a word from the NLTK word list and returns it in lowercase for consistency.
This word is then passed to the scramble_word() function to generate the scrambled version.

**Hint Generation**:
The function get_hint(word) provides a hint for the player by revealing the first and last letters of the word. This helps the player make an educated guess.
The hint is updated every time the word changes, ensuring the player has a new hint for each round.

**Game Loop**:
The game is structured within a main loop (game_loop()) that runs continuously until the user decides to quit.The game loop manages player input, displays scrambled words, updates the score, and provides feedback on the player's guesses. The player's and computer's scores are updated based on correct guesses and turn-based challenges.

**Event Handling**:
The loop listens for pygame.QUIT events to close the game window.Player input is handled through key events. When the RETURN key is pressed, the player's guess is compared to the actual word. The BACKSPACE key allows the player to delete characters from their guess.The unicode attribute of the KEYDOWN event captures player input as characters, which are added to the guess.

**Score Management**:
The player and computer scores are stored in separate variables (player_score and computer_score).After each round, the scores are updated based on whether the player's guess is correct or not.

**Graphics Rendering**:
Various pieces of information are rendered on the screen using pygame.font.Font.render(). This includes the scrambled word, score, user's current guess, messages (correct/incorrect guesses), and hints.to guess the player's scrambled word.

### 3.2.1 Algorithms:

**Scrambling Algorithm**:

```
def scramble_word(word):
    scrambled = ''.join(random.sample(word, len(word)))
    return scrambled if scrambled != word else
scramble_word(word)
```

This uses the random.sample() function to generate a scrambled version of the input word by randomly rearranging its characters. If the scrambled word is the same as the original word, the function recursively calls itself to generate a new scramble.

**Hint Generation Algorithm**:

```
def get_hint(word):
        return f"Hint: Starts with '{word[0]}' and ends with
      '{word[-1]}'"
```

This function constructs a string that provides the first and last letter of the word as a hint for the player.

**Game Loop and Event Handling**: The main game loop repeatedly:

- Accepts player input and updates the game state.
- Renders the updated state on the screen.
- Handles scoring, alternating turns, and managing the transition between rounds.

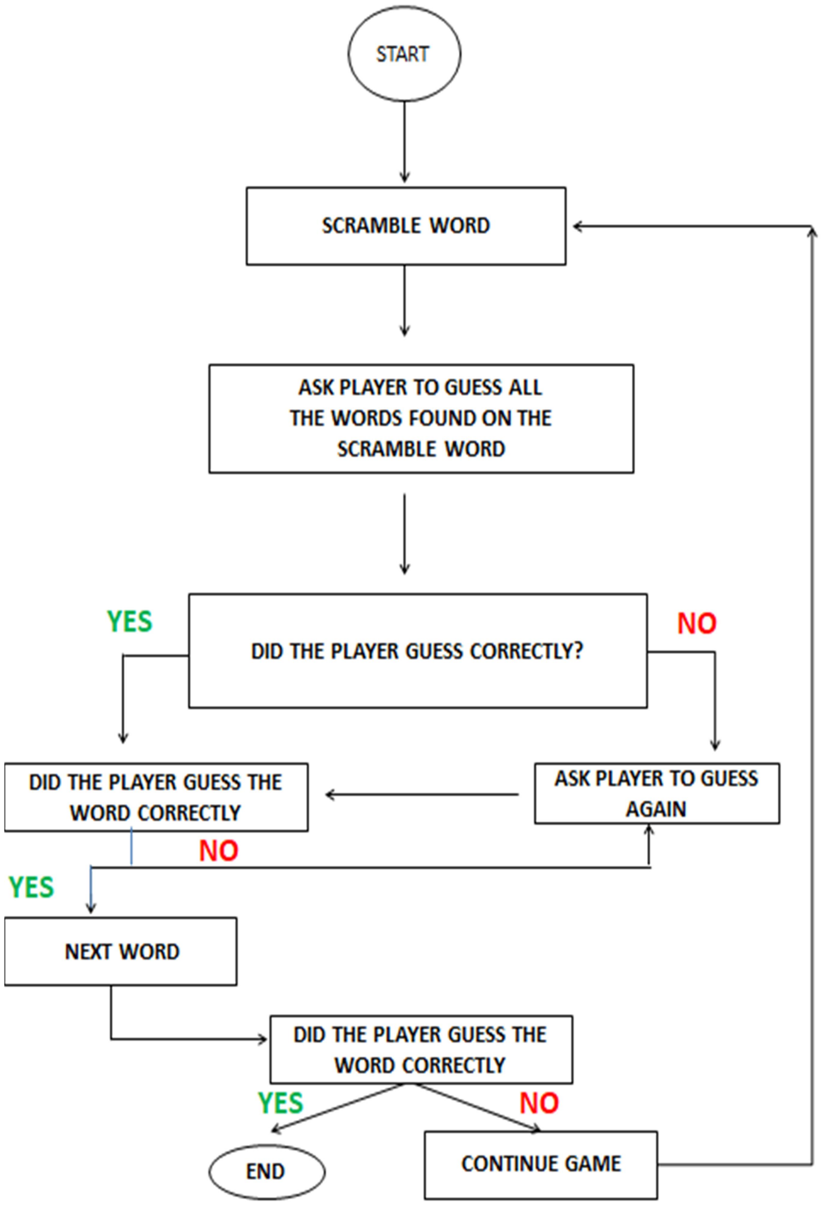

START
SCRAMBLE WORD
ASK PLAYER TO GUESS ALL THE WORDS FOUND ON THE SCRAMBLE WORD
YES
DID THE PLAYER GUESS CORRECTLY?
NO
DID THE PLAYER GUESS THE WORD CORRECTLY
ASK PLAYER TO GUESS AGAIN
NO
YES
NEXT WORD
DID THE PLAYER GUESS THE WORD CORRECTLY
YES
NO
END
CONTINUE GAME

***Fig 5 : Flowchart***

## 3.3 Project Code

```
import pygame

import random
import nltk
from nltk.corpus import words
# Initialize Pygame and NLTK
pygame.init()
nltk.download('words')
# Screen setup
WIDTH, HEIGHT = 800, 500
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption(" WORDS UNSCRAMBLE GAME")
# Fonts and colors
FONT = pygame.font.Font(None, 36)
WHITE = (255, 255, 255)
RED = (220, 20, 60)  # Crimson red for scrambled word
BLUE = (0, 0, 205)  # Medium blue for scores
GREEN = (34, 139, 34)  # Forest green for messages
ORANGE = (255, 140, 0)  # Dark orange for user input
PURPLE = (75, 0, 130)  # Indigo for hints
GOLD = (255, 215, 0)  # Gold for the timer
# Game settings
TIME_LIMIT = 60  # seconds per turn
word_list = [word.lower() for word in words.words() if len(word) == 5]  #
Filter words to length of 5
player_score = 0
# Utility functions
def scramble_word(word):
    scrambled = ''.join(random.sample(word, len(word)))
    return scrambled if scrambled != word else scramble_word(word)  # Ensure
it's actually scrambled
def get_random_word():
    return random.choice(word_list)
def get_hint(word):
    return f"Hint: Starts with '{word[0]}' and ends with '{word[-1]}'"
# Main game loop
def game_loop():
    global player_score
    clock = pygame.time.Clock()
    run = True
    # Game variables
    current_word = get_random_word()
    scrambled_word = scramble_word(current_word)
    user_input = ""
    message = "Guess the word!"
    hint_message = get_hint(current_word)
    timer = TIME_LIMIT  # Set timer for each round
    while run:
        screen.fill(WHITE)
        # Timer countdown
        timer -= 1 / 30  # Decrease by 1 second every 30 frames
        if timer <= 0:
            message = f"Time's up! The correct word was '{current_word}'."
            timer = TIME_LIMIT  # Reset timer
            current_word = get_random_word()
            scrambled_word = scramble_word(current_word)
            user_input = ""
            hint_message = get_hint(current_word)
        # Render game elements with colors
        word_text = FONT.render(f"Scrambled: {scrambled_word}", True, RED)
          score_text  =  FONT.render(f"Player  Score:  {player_score}",  True,
BLUE)
        input_text = FONT.render(f"Your Guess: {user_input}", True, ORANGE)
        message_text = FONT.render(message, True, GREEN)
        hint_text = FONT.render(hint_message, True, PURPLE)
        timer_text = FONT.render(f"Time left: {int(timer)}s", True, GOLD)
        # Blit to screen
        screen.blit(word_text, (50, 50))
        screen.blit(score_text, (50, 100))
        screen.blit(input_text, (50, 150))
        screen.blit(message_text, (50, 200))
        screen.blit(hint_text, (50, 250))
        screen.blit(timer_text, (600, 50))  # Display timer in the top-right
corner
        pygame.display.flip()
        # Event handling
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RETURN:  # Submit guess
                    if user_input.lower() == current_word:
                        message = "Correct! You guessed it."
                        player_score += 1
                    else:
                                 message  =  f"Wrong!  The  correct  word  was
{current_word}."
                    # Reset for next round
                    current_word = get_random_word()
                    scrambled_word = scramble_word(current_word)
                    user_input = ""
                    hint_message = get_hint(current_word)
                    timer = TIME_LIMIT  # Reset timer
                elif event.key == pygame.K_BACKSPACE:
                    user_input = user_input[:-1]
                else:
                    user_input += event.unicode
        clock.tick(30)
    pygame.quit()
# Run the game
game_loop()
```

## Explanation:

### 1. Importing Required Libraries :

**pygame**: A library for creating 2D games, responsible for rendering graphics, handling events, and controlling game loops.

**random**: Used to randomly select words and shuffle them.

**nltk**: A natural language processing toolkit, here used to access a list of English words from the NLTK words corpus.

### 2. Initializing Pygame and NLTK

**pygame.init():** Initializes all Pygame modules needed for game development (such as rendering, event handling).

**nltk.download('words'):** Downloads the words corpus from NLTK, which provides a large list of English words for use in the game.

## 3. Game Settings and Configuration

**Window Setup:** The game window is set to 800x500 pixels with the title "WORDS UNSCRAMBLE GAME".
**Fonts and Colors:** Font and color settings for text are defined (e.g., white for background, red for scrambled words).
**Game Constants:** A timer of 60 seconds is set for each round, and a list of 5-letter words is generated from the NLTK words corpus. The player's score is initialized to 0.

## 4. Utility Functions

**`scramble_word(word):`**Scrambles the given word by randomly rearranging its letters.If the scrambled word is identical to the original, the function is called recursively until the letters are shuffled into a new order.
**`get_random_word():`**Selects and returns a random word from the list of 5-letter words.
**`get_hint(word):`** Generates a hint for the player, showing the first and last letter of the word to help them guess.

## 5.Main Game Loop (game_loop):

**Global Variable**: The player_score is accessed to track the score.
**Clock**: The clock object is used to control the frame rate (30 frames per second).
**Game Loop**: The main loop (run = True) continues until the game window is closed.

### 5.1 Initial Game Setup

The game initializes the first round with a random word, scrambled version of the word, an empty input field, a message for the player, a hint, and sets the timer to 60 seconds.

### 5.2 Timer Countdown

The timer decreases with every frame (1 second every 30 frames).If the timer reaches 0, a message informs the player that time is up, and the game resets for the next round with a new word and a timer reset.

### 5.3 Rendering Game Elements

Various game elements are rendered:

1.Scrambled word (`word_text`).
2.Player score (`score_text`).
3.User's current guess (`input_text`).
4.Status message (`message_text`).
5.Hint (`hint_text`).
6.Timer countdown (`timer_text`).

All these elements are drawn on the screen using screen.blit(), and pygame.display.flip() updates the screen to show changes.

### 5.4 Handling Player Input

**Event Handling**: The program listens for events, such as:

**1.QUIT:** If the window is closed, the game ends (run = False).

**2.KEYDOWN**: If a key is pressed, the following actions occur:

**3.Enter**: Submits the player's guess. If correct, the score increases. Otherwise, the correct word is shown.

**4.Backspace**: Removes the last character from the player's input.

**5.Other keys**: Adds the pressed character to the player's guess.

## 6. Exiting the Game

**`pygame.quit()` :** After the game loop ends, pygame.quit() is called to clean up Pygame resources and exit the game.

**`game_loop()` :** The game_loop() is called to start the game.

# CHAPTER – 4
# RESULTS

## 4.1 Output

The project implements a simple word scramble game using Pygame, where players guess scrambled words. Here's an overview of the results based on the functionality and gameplay elements implemented:

**Game Structure and Flow**

The game structure defines how the game progresses from one stage to another. In this game, the flow is broken down into a series of steps that alternate between the player and the computer. Here's how each component works

**Game Output**

Upon running the game, the player is presented with a GUI window that displays the following key components:

**Scrambled Word**:

The system selects a random 5-letter word from a predefined list, scrambles it, and displays the scrambled version to the player. The goal is for the player to guess the original word within a 60-second time limit.

**Timer**:

A countdown timer is shown at the top-right corner of the screen. The timer starts at 60 seconds and decreases by 1 second for every 30 frames, which is equivalent to the game's frame rate of 30 frames per second.

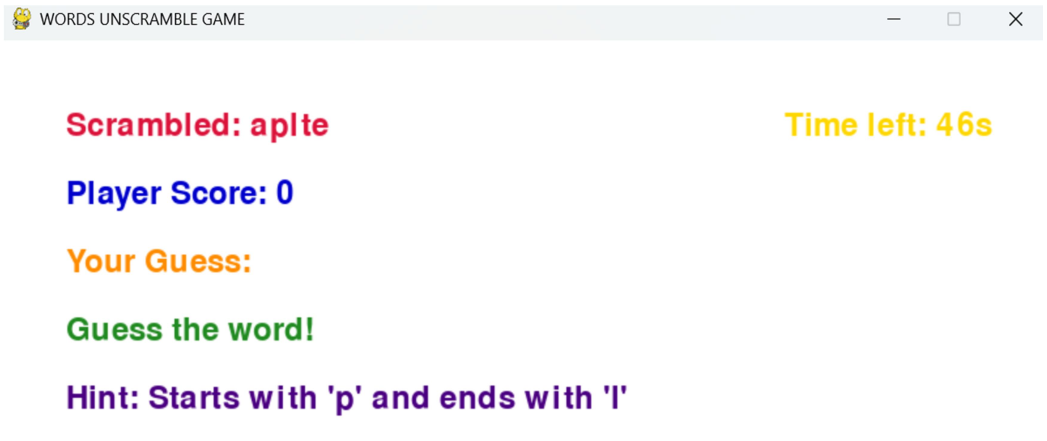
**Player Score**:

The player's score is displayed in blue at the top-left corner of the screen. The score increases by 1 each time the player guesses the word correctly. This score is a direct reflection of the player's success rate.

**Hint**::A hint is provided at the beginning of each round. The system shows the first and last letters of the word, giving the player additional assistance in unscrambling the word.

**User Input**:

The player can input their guesses using the keyboard. As they type, their guess is shown in real time in the input area. Feedback is immediately provided:


WORDS UNSCRAMBLE GAME

Scrambled: aplte

Time left: 46s

Player Score: 0

Your Guess:

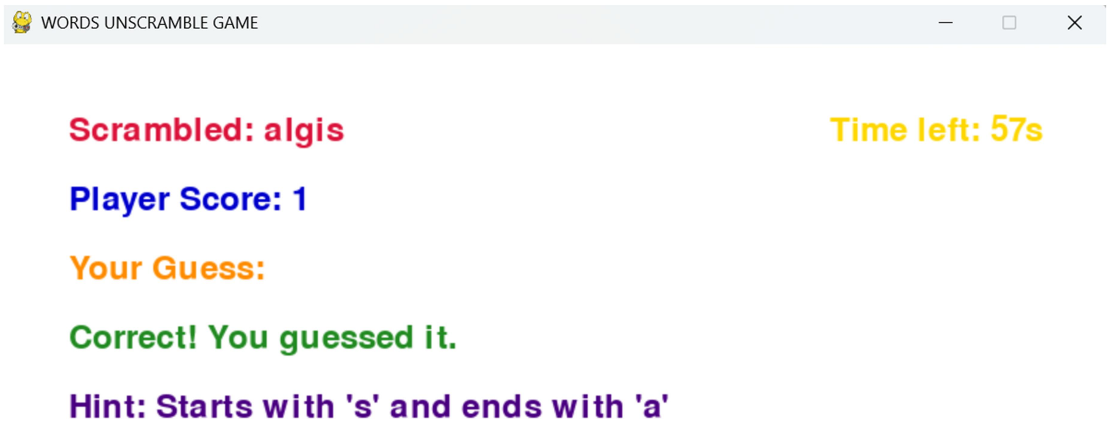
Guess the word!

Hint: Starts with 'p' and ends with 'l'

*Fig 6 : Scramble Word*

## Result

**Correct Guess**: The score increases, and the player sees a message confirming their correct guess.


WORDS UNSCRAMBLE GAME

Scrambled: algis

Time left: 57s

Player Score: 1

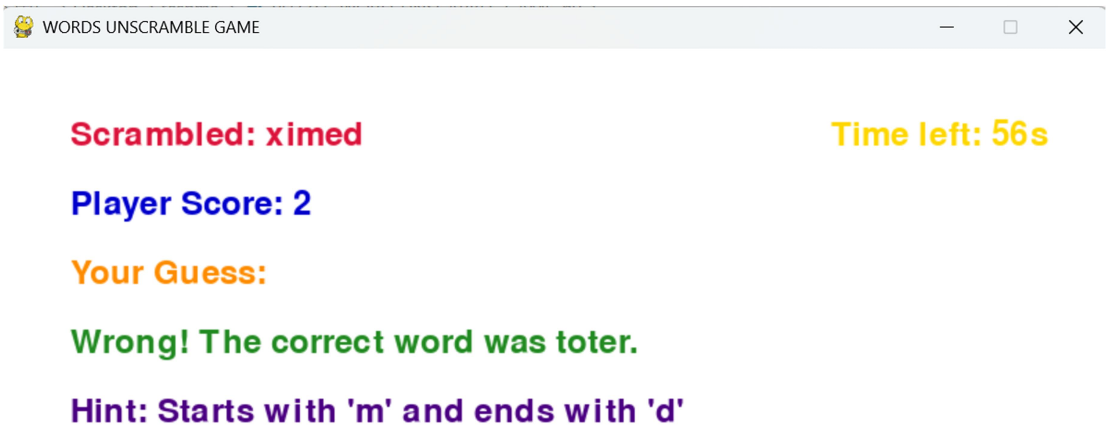
Your Guess:

Correct! You guessed it.

Hint: Starts with 's' and ends with 'a'

*Fig 7 : Output of the Correct Word*

**Wrong Guess**: The correct word is displayed, but the score remains unchanged.


WORDS UNSCRAMBLE GAME

```
Scrambled: ximed                                  Time left: 56s
Player Score: 2
Your Guess:
Wrong! The correct word was toter.
Hint: Starts with 'm' and ends with 'd'
```

***Fig 8 : Output of the wrong guess***

**Game Reset**:

After each round, whether the guess was correct or incorrect, the system generates a new scrambled word, resets the timer, and clears the user input, starting a fresh round.

## 4.2 Discussion

### Scoring System and Game Progression

The **scoring system** is central to keeping track of progress and rewarding players. Here's how you can expand this feature:

**Score Scaling**: The current scoring system gives a point for each correct guess. To make this more dynamic, you could:

**Game Length and Rounds**: The game currently doesn't have a fixed length, but introducing **round limits** or a "best-of" format would give players a sense of accomplishment when they win a set number of rounds or achieve a high score.

**Hints System :** The **hint system** provides players with valuable clues to make the game more fun and engaging. In the current version, the hint reveals the first and last letters of the word.

**Future Extensions**:Looking forward, here are a few ways to **extend the game**:

**Leaderboard**: Keeping track of the best scores can add a competitive element. This could be local (on the player's machine) or online with cloud storage to track scores across sessions.

**Mobile Version**: Adapting the game for **mobile devices** (iOS/Android) could make it accessible to a wider audience. The game would need to be optimized for smaller screens and touch inputs.

**Multilingual Support**: If you want to reach a global audience, you could add **multilingual support**, allowing players to choose their language. This could make the game more inclusive and help players learn new words in different languages.

Expanding the **game features**, enhancing the **UI**, and introducing more interactivity would significantly improve the overall gameplay experience. As the game grows, optimizing performance, adding sound and visual effects, and providing more challenging word lists can ensure the game remains engaging and replayable. The detailed expansion of these aspects can result in a polished, enjoyable game that attracts a broader audience.

## 4.3 Performance Analysis:

### 1. Efficiency:

The performance of the game is smooth on most systems. The use of the pygame library ensures that the rendering of text and graphical elements is efficient. The game runs at a stable frame rate of 30 frames per second (FPS), which provides a fluid user experience.

The timer is updated once per frame, and the game logic (checking if the word is guessed correctly, updating the score, resetting the round) is processed at the same rate. This ensures that the game remains responsive and performs well.

### 2. Resource Usage:

**CPU and Memory Consumption:** The game's CPU usage is minimal, as the main processes involve rendering text and handling basic input/output (keyboard events). The game’s memory consumption is also low, as the word list is stored in memory as a list of strings.

**Load Times:** The game loads quickly, as it does not require extensive computational resources. Since it only uses a few libraries (pygame and nltk), it has a low overhead and loads promptly.

# CHAPTER-5

# CONCLUSION

## 5.1 Summary

This interactive word unscramble game, developed using Python and the pygame library, offers a fun and engaging challenge for word puzzle enthusiasts. The game's design revolves around presenting players with scrambled words that they must unscramble and guess correctly within a time limit. The key features of the game include:

**Dynamic Word Selection and Scrambling:** The game pulls random 5-letter words from the nltk corpus, ensuring a diverse and rich vocabulary. The system scrambles the selected word and displays it, allowing for a fresh challenge every round. The random selection mechanism guarantees that no two rounds are alike, enhancing replayability.

**Real-Time Feedback and Scoring:** The user is provided with immediate feedback upon guessing the scrambled word. If the player guesses correctly, they are rewarded with a score increment and a congratulatory message. Incorrect guesses prompt the system to reveal the correct word, offering an opportunity to learn and go to the next round. This continuous feedback helps players track their progress and stay engaged.

**Time-Limited Gameplay:** Each round is constrained by a 60-second countdown timer, which adds a sense of urgency to the challenge. As the player races against time, the timer acts as a pressure mechanism, forcing quick thinking and promoting a competitive atmosphere. Once the timer runs out, the round ends, and the correct word is displayed, resetting the game for the next round.

**Hint System for Assistance:** To support players when they are stuck, the game provides hints by revealing the first and last letters of the word. This clue can help the player make an educated guess and enhances the problem-solving aspect of the game. The hints can be crucial in cases where the player is unsure, increasing the chances of success.

**Visually Appealing GUI:** The game uses pygame to create a smooth, visually rich environment. The game window features colorful elements, such as the scrambled word (in red), the player's score (in blue), the timer (in gold), and feedback messages (in green and orange). This careful use of color helps organize the information and makes the game more visually stimulating.

**Keyboard Input Handling:** The game allows players to interact with the system using standard keyboard controls. The player types their guess and presses the Enter key to submit it. The Backspace key enables the player to correct or adjust their input if needed. This intuitive event handling ensures smooth and uninterrupted gameplay.

## 5.2 Conclusion

This word unscramble game provides an immersive and enjoyable experience by blending logic, speed, and vocabulary skills. With its engaging gameplay mechanics, intuitive user interface, and real-time feedback, it presents a fun challenge for players of all ages.

The use of Python's pygame library provides a flexible and powerful tool for creating rich graphical user interfaces, and the integration of the nltk word list ensures that the game can continually offer new words, maintaining a sense of novelty and challenge. The real-time feedback system enhances player engagement, allowing players to feel a sense of accomplishment when they get the word right and motivating them to improve when they get it wrong.

The timer mechanic adds an exciting layer of urgency, prompting quick thinking and decision-making, while the hint system ensures that players are not left frustrated by difficult words. This balance of challenge and support is key to keeping players invested.

Looking ahead, there are numerous opportunities for expanding the game's features to elevate the experience. Additional features such as varying difficulty levels, multiplayer modes, and custom word packs could increase replay value. A leaderboard or ranking system could add a competitive edge, motivating players to improve their performance over time. Implementing a system that tracks progress and offers rewards for streaks or consistent performance would further enhance the game's long-term engagement.

Moreover, the game could evolve by adding more complex hints (such as scrambled word parts or synonyms), or by incorporating themes, levels, or daily challenges to give players something new to look forward to each day. A "challenge the computer" mode, where the player can provide their own scrambled words for the system to guess, would be a highly innovative addition, increasing interactivity and offering new ways to play.

In its current form, this word unscramble game offers a solid and enjoyable experience. However, its flexibility and modular nature make it highly adaptable, allowing for a vast array of new features and mechanics to be incorporated. This makes it not just a simple game but a platform with great potential for future expansion, making it suitable for players looking for casual fun or those seeking a more competitive challenge.

## 5.3 Future Scope

The word unscramble game is engaging but could be enhanced in several areas to make it more versatile and competitive.

Introducing multiple difficulty levels—easy, medium, and hard—would cater to players of all skill levels. Easy mode could feature shorter words, while hard mode offers longer, more complex words.

Adding multiplayer functionality would allow players to compete in real-time, either turn-based or simultaneous play. Online play could also enable global competition.

A challenge where players provide scrambled words for the computer to guess would add complexity. The AI could adjust its accuracy and difficulty as the player progresses.

Word customization and themes would increase replayability. Players could choose from categories like animals, food, or movies, and participate in daily or weekly challenges.

Leaderboards and achievement systems would introduce a competitive edge. Players could track scores globally or locally, earning achievements for milestones.

Advanced AI techniques, like machine learning, could improve the game's functionality. The AI could learn from previous rounds to make better guesses, enhancing difficulty.

Visual and audio enhancements, such as smoother animations, sound effects, and background music, would make the game more immersive.

Educational features, like word definitions or example sentences, would provide added value, especially for language learners.

Expanding the game to mobile platforms would reach a larger audience, with cross-platform synchronization allowing players to switch devices while retaining progress.

AI-powered feedback would help players improve by analyzing their performance, pointing out strengths and areas for growth.

In conclusion, these enhancements—multiplayer, difficulty levels, AI improvements, and educational features—would make the game more engaging and accessible, offering an evolving challenge for a broader audience.

## 5.4 References

### Books

4. **The Art of Game Design: A Book of Lenses**
   Schell, J. (2020). *The Art of Game Design: A Book of Lenses*. CRC Press.
5. **Fundamentals of Game Design**
   Adams, E. (2014). *Fundamentals of Game Design*. New Riders Publishing.
6. **The Design of Everyday Things**
   Norman, D. A. (2013). *The Design of Everyday Things*. Basic Books.
7. **Designing Games: A Guide to Engineering Experiences**
   Despain, W. (2013). *Designing Games: A Guide to Engineering Experiences*. O'Reilly Media.
8. **Gamification by Design**
   Zichermann, G., & Cunningham, C. (2011). *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. O'Reilly Media.
9. **What Video Games Have to Teach Us About Learning and Literacy**
   Gee, J. P. (2007). *What Video Games Have to Teach Us About Learning and Literacy*. Palgrave Macmillan.
10. **Beginning Game Development with Python and Pygame**
    McGugan, W. (2007). *Beginning Game Development with Python and Pygame: From Novice to Professional*. Apress.
11. **Automate the Boring Stuff with Python**
    Sweigart, A. (2015). *Automate the Boring Stuff with Python*. No Starch Press.
12. **Head First Python**
    Barry, P. (2016). *Head First Python*. O'Reilly Media.
13. **Problem Solving with Algorithms and Data Structures Using Python**
    Miller, B. N., & Ranum, D. L. (2011). *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle & Associates.

14. **Official Documentation and Online References**
15. **Pygame Official Documentation**
    Pygame team. (n.d.). Pygame documentation. Retrieved from https://www.pygame.org/docs/

16. **Python Random Module Documentation**
    Python Software Foundation. (n.d.). random — Generate pseudo-random numbers. Retrieved from https://docs.python.org/3/library/random.html
17. **Python PEP 8 - Style Guide for Python Code**
    Python Software Foundation. (n.d.). PEP 8 — Style Guide for Python Code. Retrieved from https://peps.python.org/pep-0008/
18. **Python String Manipulation**
    Python Software Foundation. (n.d.). Built-in types — Text sequence types. Retrieved from https://docs.python.org/3/library/stdtypes.html#textseq
19. **Interactive Graphics in Python**
    Python Software Foundation. (n.d.). Pygame and event-driven programming. Retrieved from https://www.python.org/doc/
20. **NLTK Official Documentation**
    Bird, S., Klein, E., & Loper, E. (2009). Natural Language Toolkit (NLTK) documentation. Retrieved from https://www.nltk.org/
21. **NLTK Wordlist Corpus**
    Bird, S., Klein, E., & Loper, E. (2009). NLTK corpus how-to. Retrieved from https://www.nltk.org/howto/corpus.html

## Tutorials and Articles

**Real Python Pygame Tutorial:** Real Python. (n.d.). *Introduction to Pygame: A primer*. Retrieved from **https://realpython.com/pygame-a-primer/**

# APPENDIX

```
import pygame
import random
import nltk

from nltk.corpus import words
# Initialize Pygame and NLTK
pygame.init()
nltk.download('words')
# Screen setup
WIDTH, HEIGHT = 800, 500
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption(" WORDS UNSCRAMBLE GAME")
# Fonts and colors
FONT = pygame.font.Font(None, 36)
WHITE = (255, 255, 255)
RED = (220, 20, 60)  # Crimson red for scrambled word
BLUE = (0, 0, 205)  # Medium blue for scores
GREEN = (34, 139, 34)  # Forest green for messages
ORANGE = (255, 140, 0)  # Dark orange for user input
PURPLE = (75, 0, 130)  # Indigo for hints
GOLD = (255, 215, 0)  # Gold for the timer
# Game settings
TIME_LIMIT = 60  # seconds per turn
word_list = [word.lower() for word in words.words() if len(word) == 5]  #
Filter words to length of 5
player_score = 0
# Utility functions
def scramble_word(word):
    scrambled = ''.join(random.sample(word, len(word)))
    return scrambled if scrambled != word else scramble_word(word)  # Ensure
it's actually scrambled
def get_random_word():
    return random.choice(word_list)
def get_hint(word):
    return f"Hint: Starts with '{word[0]}' and ends with '{word[-1]}'"
# Main game loop
def game_loop():
    global player_score
    clock = pygame.time.Clock()
    run = True
    # Game variables
    current_word = get_random_word()
    scrambled_word = scramble_word(current_word)
    user_input = ""
    message = "Guess the word!"
    hint_message = get_hint(current_word)
    timer = TIME_LIMIT  # Set timer for each round
    while run:
        screen.fill(WHITE)
        # Timer countdown
        timer -= 1 / 30  # Decrease by 1 second every 30 frames
        if timer <= 0:
            message = f"Time's up! The correct word was '{current_word}'."
            timer = TIME_LIMIT  # Reset timer
            current_word = get_random_word()
            scrambled_word = scramble_word(current_word)
            user_input = ""
            hint_message = get_hint(current_word)
        # Render game elements with colors
        word_text = FONT.render(f"Scrambled: {scrambled_word}", True, RED)
        score_text = FONT.render(f"Player Score: {player_score}", True,
BLUE)
        input_text = FONT.render(f"Your Guess: {user_input}", True, ORANGE)
        message_text = FONT.render(message, True, GREEN)
        hint_text = FONT.render(hint_message, True, PURPLE)
        timer_text = FONT.render(f"Time left: {int(timer)}s", True, GOLD)
        # Blit to screen
        screen.blit(word_text, (50, 50))
        screen.blit(score_text, (50, 100))
        screen.blit(input_text, (50, 150))
        screen.blit(message_text, (50, 200))
        screen.blit(hint_text, (50, 250))
        screen.blit(timer_text, (600, 50))  # Display timer in the top-right
corner
        pygame.display.flip()
        # Event handling
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RETURN:  # Submit guess
                    if user_input.lower() == current_word:
                        message = "Correct! You guessed it."
                        player_score += 1
                    else:
                        message = f"Wrong! The correct word was {current_word}."
                    # Reset for next round
                    current_word = get_random_word()
                    scrambled_word = scramble_word(current_word)
                    user_input = ""
                    hint_message = get_hint(current_word)
                    timer = TIME_LIMIT  # Reset timer
                elif event.key == pygame.K_BACKSPACE:
                    user_input = user_input[:-1]
                else:
                    user_input += event.unicode
        clock.tick(30)
    pygame.quit()
# Run the game
game_loop()
```