

**Jawaharlal Nehru Technological University Hyderabad,
Telangana – 500085**



Industrial Oriented Mini Project

On

“FLIP FLOP WORD GAME”

**Project report Submitted in partial fulfilment of the requirements for
the award of degree of**

Bachelor of Technology In

COMPUTER SCIENCE AND ENGINEERING

By

AKHILA THAMMENENWAR (22P71A0502)

ANJALI TALLOJU (22P71A0503)

PRANATHI SAI KALAVAGUNTA (22P71A0531)

NEHA GOUTE (22P71A0559)

Under the Esteemed Guidance of

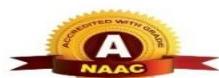
Mrs. Ch.Hyma

Head Of Department

Department of CSE



SVIT



**SWAMI VIVEKANANDA INSTITUTE OF TECHNOLOGY
Under the Management of Mahbub College (MHSS)
Mahbub College Campus, Patny Centre, S.D.Road, Secunderabad - 500003, T.S.**

SWAMI VIVEKANANDA INSTITUTE OF TECHNOLOGY

(Affiliated to JNTU-H)

2022-2026



SWAMI VIVEKANANDA INSTITUTE OF TECHNOLOGY
(An Autonomous College, approved by AICTE, Affiliated to JNTUH)
Near Patny Centre, Secunderabad, Hyderabad – 500003

CERTIFICATE

This is to certify that the Industrial Oriented Mini Project report entitled “FLIP FLOP WORD GAME” is being submitted AKHILA THAMMENENWAR(22P71A0502), PRANATHI SAI KALAVAGUNTA(22P71A0531), ANJALI TALLOJU(22P71A0503), NEHA GOUTE(22P71A0559) in partial fulfilment for the award of Degree of BACHELOR OF TECHNOLOGY in CSE to the Jawaharlal Nehru Technological University is a record of Bonafide work carried out by him/her under my guidance and supervision.

Date:

Internal Guide

HOD-CSE

External Examiner

Principal

ACKNOWLEDGEMENT

First of all, we thank our project guide and HOD Mrs. Hyma Chodapalli of the Department CSE for giving us this opportunity in developing this project. This project has really helped us in enhancing our skills of programming, the perspective with which we should view projects and above all, our presentation and interpersonal skills. Last but not the least, we also thank the professors of our department for lending their helping hand whenever it was necessary.

AKHILA THAMMENENWAR	(22P71A0502)
ANJALI TALLOJU	(22P71A0503)
PRANATHI SAI KALAVAGUNTA	(22P71A0531)
NEHA GOUTE	(22P71A0559)

DECLARATION

We hereby declare that the work which is being presented in this **Industrial Oriented Mini Project** entitled, “**FLIP FLOP WORD GAME**” submitted to JNTU-H, in the partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in CSE, is an authentic record of my own work carried out from January 2025 to June 2025 under the supervision of **Mrs. Ch.Hyma HOD of CSE Dept., SVIT, Mahbub Campus.**

The matter embodied in this project report has not been submitted by me for the award of any other degree.

Place: SECUNDERABAD

Date:

AKHILA THAMMENENWAR	(22P71A0502)
ANJALI TALLOJU	(22P71A0503)
PRANATHI SAI KALAVAGUNTA	(22P71A0531)
NEHA GOUTE	(22P71A0559)

ABSTRACT

Hand gestures are a powerful communication medium for Human Computer Interaction (HCI).

Despite being widely used, a number of input devices for computer interaction, such as the keyboard, mouse, joystick, and touch screen, are inaccessible to people with impairments. Moreover, in a running car, it is difficult for passengers or the driver to operate using touch-technology. A sound controller that uses hand gestures can be useful in these circumstances.

In this paper we are developing a volume controller in which we are using hand gestures as the input to control the system, OpenCV module is basically used in this implementation to control the gestures. This system basically uses the web camera to record or capture the images /videos and accordingly on the basis of the input, the volume of the system is controlled by this application. The

main function is to increase and decrease the volume of the system. The project is implemented using Python, OpenCV.

We can use our hand gestures to control the basic operation of a computer like increasing and decreasing volume. Therefore, people will not have to learn machine-like skills which are a burden most of the time. This type of hand gesture systems provides a natural and innovative modern way of nonverbal communication. These systems have a wide area of application in human computer interaction.

The purpose of this project is to discuss a volume control using hand gesture recognition system based on detection of hand gestures. In this the system is consist of a high resolution camera to recognize the gesture taken as input by the user. The main goal of hand gesture recognition is to create a system which can identify the human hand gestures and use same input as the information for controlling the device and by using real time gesture recognition specific user can control a computer by using hand gesture in front of a system video camera linked to a computer. In this project we are developing a hand gesture volume controller system with the help of OpenCV, Python. In this system can be controlled by hand gesture without making use of the keyboard and mouse.

LIST OF FIGURES			
S.NO	Figure No.	Title of Figure	Page No
1	4.1	Python	22
2	4.2	Flet Framework	23
3	7.3	Responsive Diagram Screenshots	32 - 33
4	8.2	Output Screens of Android	37 - 39

LIST OF ABBREVIATIONS		
S.NO	ACRONYM	EXPANSION
1	APK	Android Package
2	CMD	Command Prompt
3	venv	Virtual Environment (python)
4	UI	User Interface
5	SDK	Software Development Kit

INDEX

CONTENTS	Page No
1. INTRODUCTION	9 - 16
1.1 Project Introduction	9
1.2 Motivation	10
1.3 Problem Statement	10
1.4 Scope	11
1.5 Project Overview	11
1.6 Objectives	12
1.7 Data Set	12
1.8 Existing System	13
1.9 Disadvantage of the Existing System	13
1.10 Proposed System	14
1.11 Advantage of the Proposed System	15
2. LITERATURE SURVEY	16

2.1 Related Work	16
3. SYSTEM ANALYSIS	11-13
3.1 Functional Requirements	17
3.2 Performance Requirements	17
3.3 Software Requirements	19
3.4 Hardware Requirements	19
3.5 Feasibility study (Technical/Economical/Operational)	19-20
4. TECHNOLOGY STACK	21-23
4.1 Python	20-21
4.2 Flet Framework	21-22
4.3 Platform	22
5. ABOUT THE GAME / APP	
5.1 Game Explanation	23
5.2 Gameplay Logic and Mechanics	23-24
6. LIBRARIES AND MODULES USED	25-26
6.1 Flet	25
6.2 Random / Timer / State Management	26
6.3 Others (e.g., OS, JSON)	26
7. IMPLEMENTATION	27 - 33
7.1 Key Modules and Logic	27
7.2 Sample Code Snippets	28-31
7.3 Responsive Layout and Controls	31-33
8. OUTPUT AND RESULTS	34-38
8.1 Game Flow	34
8.2 Results (Accuracy) / Output Screens	35-38
9. APK GENERATION	39-40
10. TESTING	41-43
11. CONCLUSION	43
12. FUTURE SCOPE	44-45

1. INTRODUCTION

1.1 Project Introduction

The Flip-Flop Word Game is a dynamic and engaging word-based puzzle game developed using Python and the Flet framework. It is designed as an Android-compatible application that offers an interactive platform for users to enhance their vocabulary, memory, and problem-solving skills. The core gameplay involves presenting users with a word for a brief duration, after which the word is scrambled, and the player is required to reconstruct the original word using the shuffled letters. The game incorporates multiple difficulty levels—easy, medium, and hard—to adapt to the player's growing proficiency and to provide a progressively challenging experience.

This project leverages the Flet framework, which allows developers to build user interfaces in Python and render them as Flutter apps, making it possible to deploy cross-platform mobile applications without writing native Android code. By combining Flet's UI capabilities with Python's logical structure, the game demonstrates how educational and cognitive enhancement tools can be efficiently developed and packaged into an Android APK. The project not only serves as a fun and useful learning tool but also exemplifies modern approaches to mobile app development using emerging frameworks.

1.2 Motivation

In today's fast-paced digital world, where distractions are ever-present and attention spans are continuously shrinking—especially among younger generations—there is a growing need for applications and games that not only entertain but also serve a meaningful educational or cognitive purpose. Modern educational tools must evolve to match the habits and expectations of today's users. This project was motivated by the desire to create an interactive and intelligent game that could subtly train the brain while providing enjoyment. Unlike traditional passive forms of learning, games encourage active engagement, repeated practice, and self-paced improvement.

The Flip-Flop Word Game specifically addresses the need for tools that enhance short-term memory, pattern recognition, and concentration—skills that are increasingly critical in both academic and real-world contexts. The idea originated from observing how quickly users forget simple lists or sequences when distracted, which is a core challenge in human cognition. By converting this challenge into a gamified experience, the game makes the process of memory retention fun and competitive.

1.3 Problem Statement

Traditional word games often lack dynamic interaction and memory challenges that adapt to player performance. Many existing games are either too simplistic or too complex, failing to strike a balance that appeals to casual players and those seeking cognitive development. The problem lies in the absence of an engaging game that focuses on word memorization, disruption, and recall under pressure. Thus, there is a need for a word game that can enhance memory skills while maintaining user interest through levels of increasing difficulty and a responsive UI.

1.4 Scope

The scope of the Flip-Flop Word Game covers the complete design, development, testing, and deployment of an Android-based cognitive training game focused on improving users' short-term

memory and attention through interactive word challenges. Developed using technologies such as Kotlin or Java with Android Studio, or cross-platform frameworks like Flutter, the application is targeted at mobile users and is optimized for a smooth and responsive experience on Android smartphones and tablets.

The core gameplay revolves around three key interactive phases: Memorization, Disruption (Scrambling), and Recall. In the Memorization Phase, users are shown a list of words (typically 3–10, depending on difficulty) for a fixed amount of time. The Disruption Phase introduces a brief distraction by hiding or scrambling the words, simulating real-world memory challenges. In the Recall Phase, users are prompted to enter the words in the correct sequence using touch inputs such as typing, drag-and-drop, or button selections. The app evaluates the responses in real time and assigns scores based on accuracy and speed.

1.5 Project Overview

- This project consists of a user interface where the player first sees a list of words to memorize. After a short time, the words are scrambled or hidden, and the player is asked to reconstruct the original list in the correct order. The game features multiple difficulty levels, each varying the number of words, memorization time, and complexity of the scramble. The application maintains scores based on the accuracy of user input, encourages replayability, and can store high scores. The game uses Java Swing (or other technologies) for GUI implementation and basic data structures for game logic.

1.6 Objectives

- Enhance short-term memory and attention span through gameplay.

- Develop an engaging and interactive user interface.
- Implement dynamic levels of difficulty to suit a wide range of users.
- Incorporate a timer-based challenge to build pressure-based cognitive skills.
- Record scores to allow players to track their performance.
- Provide a foundation that can be further developed into a more complex educational game.

1.7 Data Set(s)

The dataset used in the Flip-Flop Word Game consists of a predefined list of English words categorized by difficulty level (easy, medium, hard). These words are stored internally in arrays or files and are randomly selected during gameplay. In future expansions, the dataset can be sourced from external dictionaries or user input to make the game more dynamic and personalized. The size and nature of the dataset allow for rapid access and manipulation, which is critical for maintaining performance during game transitions.

1.8 Existing Systems

Existing word-based games, such as Scrabble, crosswords, word search puzzles, and spelling quizzes, primarily focus on testing vocabulary, word formation, and spelling abilities. These games are effective for enhancing language and lexical skills but are not specifically designed to train memory recall, especially short-term memory involving word sequences or visual placement. Most of these games operate in a static environment with fixed levels and no real-time changes in difficulty. They also lack active time-bound challenges that push the player to recall information under pressure.

While they are engaging for casual play, they do not emphasize the mental flexibility or sequential memory required to remember and reorder elements accurately after disruption. Additionally, these systems usually provide passive gameplay and do not evaluate the user's memory or adapt based on their performance, limiting cognitive stimulation.

1.9 Disadvantage of the Existing System

- Primarily vocabulary-based: Focus on spelling and language skills rather than memory training.
- No visual memory component: Do not challenge users to remember word positions or sequences.
- Absence of dynamic difficulty: Levels and challenges do not adapt to the player's memory performance or improvement.
- Lack of real-time recall: Most word games do not use time-based recall phases that test short-term memory under pressure.
- Limited cognitive challenge: Without timed, adaptive, or memory-focused gameplay, traditional systems fail to provide the mental stimulation required for improving memory skills.

1.10 Proposed System

The proposed Flip-Flop Word Game system introduces a fresh approach to cognitive training by focusing on sequential memory recall through a time-structured word challenge. The game is divided into three interactive phases:

- Display Phase – A word is shown clearly for 50 seconds, allowing the player to memorize its correct letter order.
- Scramble Phase – The letters of the word are randomly shuffled and displayed in individual numbered blocks for 1 minute, challenging the player to remember the new positions while mentally retaining the original sequence.
- Recall Phase – The shuffled blocks are hidden, and the player is prompted to reconstruct the original word by entering the correct sequence based on the numbered block positions shown earlier.

This game is developed using Python and the Flet framework, which allows for rapid development of responsive UI with a native app feel. Flet's ability to build cross-platform applications, especially for Android, makes it ideal for deploying lightweight, interactive games without requiring native Android code. The game uses touch-optimized UI elements such as buttons and input fields, all designed using Flet's component-driven architecture.

The system is modular and supports easy extension to features like scoring, difficulty levels, timers, and user feedback. It provides an engaging and educational mobile experience while maintaining simplicity in both design and deployment.

1.11 Advantages of Proposed System

- The Flip-Flop Word Game offers several advantages over traditional vocabulary-based games. Its unique structure focuses on visual memory, sequence retention, and timed recall—key cognitive skills that are often neglected in standard word games. By introducing distinct phases (display, scramble, recall), the game provides a more dynamic and mentally stimulating experience.

Built using Python and Flet, the system benefits from rapid development cycles, a clean UI design, and ease of deployment on Android devices. Unlike heavier mobile frameworks, Flet offers

lightweight app performance and compatibility with both touchscreen gestures and keyboard input, making it accessible to a broad audience.

Additionally, the game can scale in difficulty and feature set without reworking the core logic. It supports offline use, does not require a large app footprint, and promotes cognitive training through an enjoyable, gamified experience. This makes it well-suited for students, educators, and casual players seeking both fun and mental engagement.

2. LITERATURE SURVEY

2.1 Related Work

Several popular word and memory games, such as Scrabble, Wordle, and Memory Match, are designed to test vocabulary skills or general memory. However, most of these games focus either on language proficiency or basic recall and do not integrate time-based visual memory challenges involving scrambling and positional recognition. The Flip-Flop Word Game fills this gap by uniquely emphasizing the player's ability to remember scrambled visual patterns and their positions within a limited time, thereby combining both cognitive and visual memory under timed conditions for a more comprehensive brain-training experience.

By requiring players to quickly memorize the original word, adapt to its scrambled layout, and then recall the correct sequence based on position, the game stimulates short-term memory, attention to detail, and sequential processing. Unlike traditional games that rely mainly on vocabulary knowledge, this approach targets the brain's capacity to process and reorganize visual information under pressure, which is crucial for real-life skills such as problem-solving, multitasking, and learning.

Furthermore, the timed aspect introduces an element of urgency that enhances cognitive challenge and keeps users motivated to improve their memory speed and accuracy. This dynamic, multi-phase gameplay not only makes the Flip-Flop Word Game engaging and fun but also serves as a practical tool for mental fitness, making it suitable for players of all ages who wish to sharpen their cognitive abilities through interactive play.

3. SYSTEM ANALYSIS

3.1 Functional Requirements

- **Word Dataset Creation:** A dataset of words categorized by difficulty levels must be created to serve as the basis for gameplay. This dataset will be used for selecting words in different game rounds.
- **Display Phase:** The system must show the original word for a fixed duration to allow the player to memorize it.
- **Scramble Phase:** The system must shuffle the letters of the displayed word and present them in numbered blocks for a set time.
- **Recall Phase:** The system must prompt the player to input the original word's letter sequence by selecting blocks based on the scramble.
- **Input Handling:** The system should accept touch input (buttons) for letter selection and support keyboard input where applicable.
- **Scoring and Feedback:** The system should calculate scores based on accuracy and speed, providing immediate feedback after each round.
- **User Interface Navigation:** The system must allow smooth navigation between menus, gameplay screens, and high-score records.
- **Offline Functionality:** The game should work without requiring an active internet connection.
- **Error Handling:** The system should manage invalid inputs gracefully without crashing.

3.2 Performance Requirements

- **Responsiveness:** The app should respond instantly (within 100 ms) to user inputs to maintain a fluid gaming experience.
- **Smooth Transitions:** Animations and phase transitions must run at a minimum of 30 frames per second.

- **Loading Time:** Game screens and word data should load within 2 seconds.
- **Resource Efficiency:** The app must use minimal CPU and memory to conserve device battery life.
- **Robustness:** The system should perform reliably across various Android devices and screen sizes without crashing or lagging.
- **Scalability:** The system should maintain performance when new words or difficulty levels are added.

3.3 Software Requirements

- **Programming Language:** Python (for game logic and backend processes).
- **UI Framework:** Flet (for building the user interface and handling touch input).
- **Development Environment:** Visual Studio Code or PyCharm (for code editing and debugging).
- **Packaging Tools:** Tools supporting Python app packaging for Android (e.g., Briefcase, BeeWare) or Flet deployment utilities.
- **Version Control:** Git and GitHub for source code management.

3.4 Hardware Requirements

- Desktop or Laptop with at least 8 GB RAM and a modern multi-core processor.
- Android smartphone or emulator (Android 6.0 or later) for testing.

3.5 Feasibility study (Technical/Economical/Operational)

Technical Feasibility:

- The game's logic and UI can be efficiently developed using Python and Flet, enabling cross-platform deployment with native-like performance.
- Existing Python packaging tools make Android deployment feasible without deep native coding knowledge.
- The project scope is moderate, focusing on relatively simple UI and gameplay mechanics.
- Offline operation and modular design enhance maintainability and future feature extension.

Economic Feasibility:

- The project uses free and open-source software tools, minimizing software costs.
- Development primarily requires developer time and standard hardware, reducing overall expenses.
- The game targets a wide Android user base, providing potential for monetization or educational licensing.
- No additional costly infrastructure or licenses are needed, making the project cost-effective.

Operational Feasibility:

- The game is designed for intuitive use, requiring minimal training or technical expertise for users.
- Offline mode allows use in low-connectivity environments, increasing accessibility.
- Regular updates and maintenance can be managed using common software deployment tools.
- The development team's existing Python and UI skills align well with project requirements.
- Privacy concerns are minimal as no sensitive data is collected or transmitted.

4. TECHNOLOGY STACK

A **Technology Stack** refers to the combination of programming languages, frameworks, tools, and platforms used to develop and deploy a software application. It encompasses all the technologies that work together to build the frontend (user interface), backend (logic and data processing), and infrastructure (hardware and operating system) of a project. Selecting the right technology stack is crucial for ensuring efficient development, performance, scalability, and maintainability of the application.

4.1 PROGRAMMING LANGUAGE - PYTHON

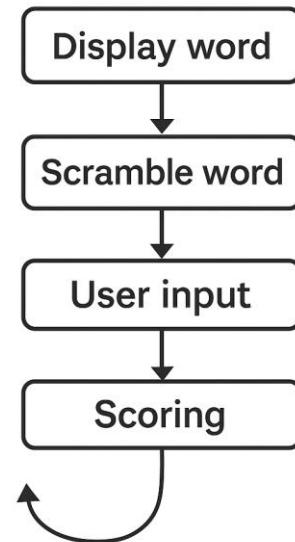
Python is a versatile, high-level, interpreted programming language widely used for software development, data analysis, automation, artificial intelligence, and more. Known for its clean and readable syntax, Python emphasizes code readability and simplicity, making it an excellent choice for both beginners and experienced developers.

Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. This flexibility allows developers to write clean, modular code that is easy to maintain and extend — a critical advantage when developing interactive applications like the Flip-Flop Word Game.

Some key features of Python that make it ideal for game development include:

- **Rich Standard Library:** Python comes with an extensive library of modules and tools, reducing the need to write code from scratch for common tasks such as file handling, networking, and data manipulation.
- **Large Community and Ecosystem:** With a vast and active user base, Python offers countless third-party libraries and frameworks that speed up development.
- **Cross-Platform Compatibility:** Python programs run on various operating systems, including Windows, Linux, and macOS, facilitating development and testing across platforms.

- **Ease of Integration:** Python can be easily integrated with other languages and technologies, allowing hybrid solutions where performance-critical parts can be written in languages like C++.
- **Rapid Prototyping:** The interpreted nature of Python allows for quick testing and iteration of game features, which accelerates the development cycle.



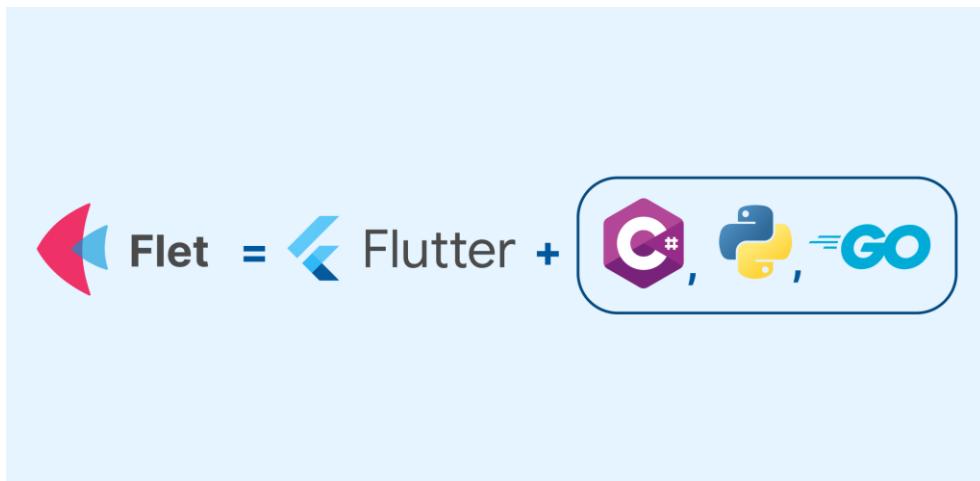
4.2 Flet Framework

Flet is an open-source framework that allows you to **build beautiful, interactive, and real-time web, desktop, and mobile applications using Python**.

In essence, Flet bridges the gap between Python development and modern, cross-platform UI technologies. It lets Python developers create rich user interfaces without needing to learn JavaScript, HTML/CSS, or specific mobile/desktop UI frameworks like React Native, Swift/Kotlin, or WPF/Qt.

- **You write Python code:** You use Flet's Python API to define your application's UI components (buttons, text fields, layouts, etc.) and its logic.
- **Flet translates to Flutter:** Flet acts as a "compiler" or "bridge." It takes your Python code and translates it into instructions that a Flutter engine can understand.
- **Flutter renders native UI:** The Flutter engine then takes these instructions and renders high-performance, beautiful UI components that look and feel native on the target platform (web browser, Windows/macOS/Linux desktop, Android, iOS).

- **Real-time updates:** Flet maintains a real-time connection between your Python backend and the Flutter frontend, allowing for instant UI updates and interactivity as your Python code modifies the application state.



4.3 Platform

The Flip-Flop Word Game targets the Android platform, one of the world's most widely used mobile operating systems. Android's extensive device ecosystem and large user base make it an ideal platform for distributing the game to a broad audience. Using Python and Flet, the game can be packaged and deployed as a native-like Android application, ensuring smooth performance and access to essential device features such as touch input and local storage. Android's support for offline apps aligns with the project's goal to provide an uninterrupted, engaging experience without requiring internet connectivity.

5. ABOUT THE GAME

5.1 Game Explanation

The Flip-Flop Word Game is a memory and word-unscrambling challenge designed to test a player's ability to recall sequential information. The core objective is to correctly reconstruct an

original word by identifying the numerical sequence of its scrambled letter blocks. The game progresses through distinct phases, requiring players to memorize the original word, then the scrambled word with associated block numbers, and finally to input the numerical sequence that restores the word to its original order. Points are awarded for correct answers and deducted for incorrect ones, encouraging precision and quick thinking.

5.2 Gameplay logics and Mechanics

The game unfolds in rounds, each consisting of three main states, managed by the `GameState` enum:

1. Showing Original Word (`GameState.SHOWING_ORIGINAL`):

- a. A random word is selected based on the chosen difficulty.
- b. This word is displayed to the player for a set amount of time (e.g., 20-40 seconds, depending on difficulty).
- c. The player's task is to memorize this original word.

2. Showing Scrambled Word (`GameState.SHOWING_SCRAMBLED`):

- a. After the original word timer expires, the game transitions to this state.
- b. The original word's letters are scrambled, and each letter block is assigned a unique number indicating its original position. For example, if "CAT" is scrambled to "ACT," 'A' might be block 2, 'C' block 3, and 'T' block 1.
- c. This scrambled word with its numbered blocks is displayed for a longer duration (e.g., 25-45 seconds, depending on difficulty).
- d. Players must memorize the position numbers associated with each letter.

3. Waiting for Answer (`GameState.WAITING_FOR_ANSWER`):

- a. Once the scrambled word timer finishes, the game moves to this state.
- b. The player is prompted to enter the sequence of block numbers that would unscramble the word back to its original form. For "CAT" scrambled to "ACT" where A=2, C=3, T=1, the correct answer would be "321" (C from block 3, A from block 2, T from block 1).

- c. A text input field is provided for the player to enter their numerical sequence.
- d. The "Submit" button processes the player's input.

4. Showing Result (`GameState.SHOWING_RESULT`):

- a. Immediately after the player submits an answer, or if time runs out on the answer phase, the game shows the result.
- b. It indicates whether the answer was "Correct!" or "Wrong!".
- c. The original word, the correct sequence, and the player's submitted answer are displayed for review.
- d. After a short delay (3 seconds), a new round automatically begins.

6. LIBRARIES AND MODULES USED

6.1 FLET

flet (aliased as ft): This is the foundational framework that enables the entire application to be built using Python, providing cross-platform UI capabilities.

UI Components: Flet supplies all the visual elements (widgets) that constitute the game's interface. Examples include `ft.Text` for displaying score, timer, and game messages, `ft.ElevatedButton` for difficulty selection and submitting answers, `ft.TextField` for user input, `ft.Container` and `ft.Column/ft.Row` for layout and organization of elements on the page. `ft.Icon` is used for visual feedback on correct/wrong answers.

Styling and Theming: Flet provides extensive styling options. `ft.Colors` is used for setting background colors, text colors, and border colors to create a visually appealing theme. `ft.ButtonStyle`, `ft.TextStyle`, `ft.FontWeight`, `ft.TextAlign` are used to customize the appearance of buttons and text for better user experience and readability. `ft.ThemeMode.LIGHT` sets the overall theme of the application.

Page Management: `ft.Page` represents the application window or screen. Its properties (`page.title`, `page.vertical_alignment`, `page.window_width`, `page.padding`, etc.) control the overall layout and appearance of the app window. Methods like `page.add()`, `page.clear()`, and `page.update()` are used to dynamically manage and refresh the UI content based on game state changes.

Event Handling: Flet enables interaction through event handlers, such as `on_click` on buttons, which trigger Python functions (e.g., `start_game`, `check_answer_click`, `show_menu`) in response to user actions.

Application Entry Point: `ft.app(target=main)` is the entry point that initializes and runs the Flet application, launching the main function.

- **Reduction of Data Annotation Efforts:**
- **Semi-Supervised and Self-Supervised Learning:** AI techniques like semisupervised and selfsupervised learning reduce the dependency on labeled data for training hand recognition

6.2 Random / Time / State Management

random: This standard Python module is critical for the "Flip-Flop" aspect of the game.

Word Selection: `random.choice(word_list)` is used to select a random word from the `easy_words`, `medium_words`, or `hard_words` lists for each new round of the game.

Word Scrambling: `random.shuffle(scrambled_chars)` is employed to randomize the order of the letters of the `current_word`, creating the scrambled word blocks that the player needs to unscramble.

threading: This module facilitates concurrent execution, preventing the UI from becoming unresponsive during time-sensitive game phases.

Background Timer: A `threading.Thread` is used to run the `timer_worker` function in a separate thread. This allows the game's countdown timer to operate independently in the background, updating the `timer_text` and managing game state transitions without blocking the main UI thread.

time: This module provides time-related functions used in conjunction with threading for the game's timing mechanisms.

Timer Delay: `time.sleep(1)` is specifically used within the `timer_worker` thread to pause execution for one second, creating the precise one-second intervals for the countdown timer.

enum (specifically Enum): This built-in module helps in defining symbolic names for sets of constant values, improving code readability and maintainability.

Difficulty Levels: The Difficulty enum (EASY, MEDIUM, HARD) categorizes the game's challenge settings, each associated with different word lists and timer durations.

Game States: The GameState enum (SHOWING_ORIGINAL, SHOWING_SCRAMBLED, WAITING_FOR_ANSWER, SHOWING_RESULT) defines the distinct phases a game round progresses through, allowing the game logic to clearly track and transition between different UI and interaction modes.

6.3 Others

typing (specifically List, Tuple): This module provides standard type hints, which are used for documentation and static analysis purposes.

Code Clarity: Type hints like List[str] (for lists of strings like word lists) and Tuple[str, int] (for the scrambled word blocks, which are character-number pairs) are used throughout the FlipFlopGame class and function signatures. They improve code readability by indicating the expected data types for variables, function arguments, and return values, making the codebase easier to understand, debug, and extend.

7. IMPLEMENTATION

7.1 Key Modules and Logic

The Flip-Flop Word Game's implementation is structured around a central FlipFlopGame class that encapsulates the game's state and core logic, driven by Flet's UI framework.

- **FlipFlopGame Class:**
 - Manages game-wide data, including word lists for different difficulties (easy_words, medium_words, hard_words).
 - Tracks the current_word, scrambled_word, score, time_left, and correct_answer.
 - Crucially, it manages the game_state (using the GameState enum) and difficulty (using the Difficulty enum) to control the flow and rules of the current round.
 - **Methods:**
 - `__init__()`: Initializes all game variables.
 - `setup_difficulty()`: Configures original_word_time and scrambled_word_time based on the selected Difficulty.
 - `get_word_list_for_difficulty()`: Returns the appropriate word list based on the current difficulty.

- `start_new_round()`: Selects a new random word, generates its scrambled version with sequential block numbers, and calculates the `correct_answer` sequence. It also resets the game state to `SHOWING_ORIGINAL`.
 - `check_answer()`: Compares the user's input with the `correct_answer`, updates the score, and sets `is_answer_correct` and `game_state` to `SHOWING_RESULT`.
- **Flet main(page: ft.Page) Function:**
 - This is the entry point for the Flet application.
 - Initializes the `ft.Page` properties, such as `page.title`, `page.theme_mode`, `page.window_width`, and `page.window_height`.
 - Instantiates the `FlipFlopGame` class.
 - Defines all UI components (e.g., `score_text`, `timer_text`, `answer_input`) and their initial properties.
 - Contains the core logic for rendering different game screens (`build_original_word_display`, `build_scrambled_word_display`, `build_answer_input`, `build_result_display`) based on the `game_state`.
 - Manages the display of the main menu (`show_menu`) and transitions to the game screen (`start_game`).
 - Handles user interactions through event handlers (`on_click`) linked to buttons and input fields.
- **Timer and Threading Logic:**
 - The `timer_worker()` function runs in a separate `threading.Thread`.
 - It decrements `game.time_left` every second using `time.sleep(1)` and updates the `timer_text`.
 - When `game.time_left` reaches zero, `timer_worker` is responsible for automatically transitioning the `game_state` (e.g., `SHOWING_ORIGINAL` to `SHOWING_SCRAMBLED`, or `SHOWING_SCRAMBLED` to `WAITING_FOR_ANSWER`, or `SHOWING_RESULT` to a new round) and updating the UI accordingly.
 - `start_timer()` and `stop_timer()` functions control the lifecycle of this background timer thread.

7.2 Sample Code Snippet

1. FlipFlopGame Class Initialization and Difficulty Setup: This snippet shows the game's initial state variables and how difficulty settings are applied.

Python

```

class FlipFlopGame:
    def __init__(self):
        self.easy_words = [
            "CAT", "DOG", "RUN", "JUMP", "PLAY", "RED"
        ]
        self.medium_words = [
            "APPLE", "TABLE", "CHAIR", "HOUSE", "PHONE"
        ]
        self.hard_words = [
            "COMPUTER", "ELEPHANT", "MOUNTAIN", "UNIVERSE"
        ]

        self.current_word = ""
        self.scrambled_word = [] # Stores (character, block_number) tuples
        self.score = 0
        self.time_left = 0
        self.game_state = GameState.SHOWING_ORIGINAL
        self.difficulty = Difficulty.EASY
        self.original_word_time = 20
        self.scrambled_word_time = 25
        self.correct_answer = "" # The sequence of numbers, e.g., "321"
        self.user_answer = ""
        self.is_answer_correct = False
        self.timer_running = False
        self.timer_thread = None

    def setup_difficulty(self, difficulty: Difficulty):
        self.difficulty = difficulty
        if difficulty == Difficulty.EASY:
            self.original_word_time = 20
            self.scrambled_word_time = 25
        elif difficulty == Difficulty.MEDIUM:
            self.original_word_time = 30
            self.scrambled_word_time = 35
        else: # HARD
            self.original_word_time = 40
            self.scrambled_word_time = 45

```

2. Scrambling Word and Generating Correct Answer Sequence: This crucial part of `start_new_round` demonstrates how the word is scrambled and the numerical sequence for the correct answer is generated.

Python

```

def start_new_round(self):
    word_list = self.get_word_list_for_difficulty()
    self.current_word = random.choice(word_list)

    original_chars = list(self.current_word)

```

```

scrambled_chars = original_chars.copy()
random.shuffle(scrambled_chars) # Shuffle characters for display

self.scrambled_word = []
for i, char in enumerate(scrambled_chars):
    # Assign sequential block numbers to scrambled chars
    self.scrambled_word.append((char, i + 1))

answer_sequence = []
# Logic to find the block numbers to reconstruct the original word
for original_char in self.current_word:
    for i, (scrambled_char, block_num) in
enumerate(self.scrambled_word):
        if scrambled_char == original_char and block_num not in
answer_sequence:
            answer_sequence.append(block_num)
            break

self.correct_answer = ''.join(map(str, answer_sequence))
# ... (rest of the method)

```

3. Timer Worker Thread: This snippet shows the background thread that handles the game's countdown timer.

Python

```

def timer_worker():
    while game.timer_running:
        time.sleep(1) # Pause for 1 second
        if not game.timer_running:
            break

        if game.time_left > 0:
            game.time_left -= 1
            timer_text.value = f"Time: {game.time_left}"
            page.update() # Update UI from background thread
        else:
            # Logic to transition game states when timer runs out
            if game.game_state == GameState.SHOWING_ORIGINAL:
                game.game_state = GameState.SHOWING_SCRAMBLED
                game.time_left = game.scrambled_word_time
                update_game_content()
            # ... (other state transitions)

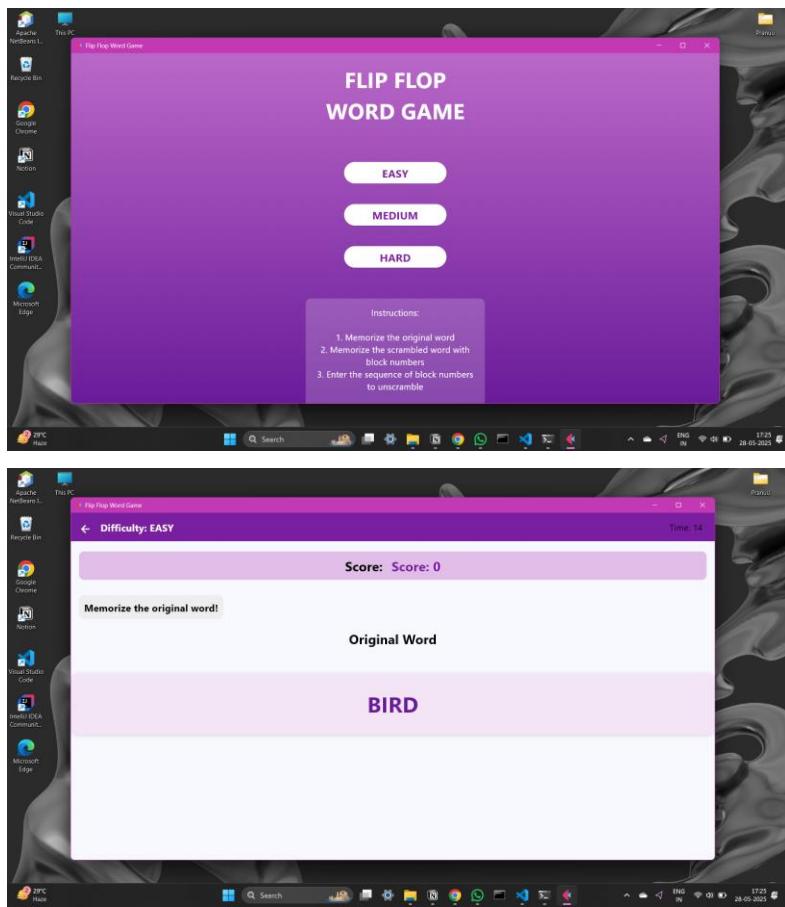
```

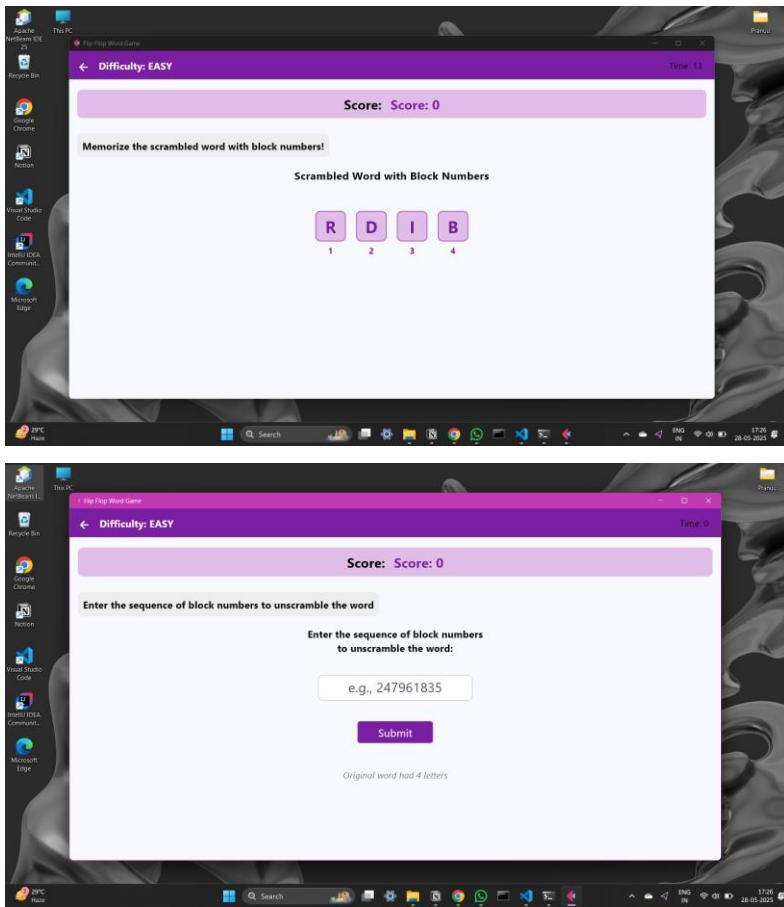
4. Button Styling for Main Menu: This demonstrates how `ft.ButtonStyle` is used to create visually appealing buttons.

Python

```
ft.ElevatedButton(
    "EASY",
    on_click=lambda _: start_game(Difficulty.EASY),
    style=ft.ButtonStyle(
        bgcolor=ft.Colors.WHITE,
        color=ft.Colors.PURPLE_700,
        padding=ft.padding.symmetric(horizontal=50,
vertical=15),
        text_style=ft.TextStyle(size=20,
weight=ft.FontWeight.BOLD),
        shape=ft.RoundedRectangleBorder(radius=30)
    ),
    width=200
),
```

7.3 Responsive layout and Controls





The application is designed with a mobile-first approach, using Flet's layout controls to provide a consistent and responsive experience within its defined boundaries.

Fixed Window Dimensions: The application sets a fixed page.window_width = 400 and page.window_height = 700. While this means the window size itself is not dynamically resizable by the user, the internal layout is structured to fit well within these dimensions and adapt logically if the application were to scale to slightly different sizes or orientations on various devices.

Vertical and Horizontal Alignment: page.vertical_alignment = ft.MainAxisAlignment.CENTER and page.horizontal_alignment = ft.CrossAxisAlignment.CENTER ensure that the primary content of each screen is centered, providing a balanced visual appearance.

Flexible Containers and Columns/Rows:

ft.Column and ft.Row are extensively used to organize widgets vertically and horizontally. They often use expand=True to allow content to take up available space, and alignment properties (ft.MainAxisAlignment.CENTER, ft.CrossAxisAlignment.CENTER) to position child widgets within them.

ft.Container widgets are used as flexible wrappers for content, allowing precise control over padding, margin, bgcolor, border_radius, and shadow effects for visual separation and styling. They often use expand=True to fill available space, contributing to the responsive nature.

Dynamic UI Content: The main_content container's content property is dynamically updated based on the game.game_state (e.g., build_original_word_display(), build_scrambled_word_display()), ensuring that only relevant UI elements are shown at each phase of the game.

Custom Letter Blocks: The create_letter_block function dynamically generates ft.Container widgets for each letter, including nested ft.Column and ft.Text for the character and its block number. These blocks are then arranged in ft.Rows which are themselves placed in a ft.Column (build_scrambled_word_display), allowing for flexible arrangement of blocks, potentially wrapping to new rows if many letters are present.

Interactive Controls Design:

Buttons: ft.ElevatedButtons are styled with ft.ButtonStyle to have consistent background colors, text colors, padding, and rounded shapes, providing clear visual cues for clickable elements. The text_style property ensures consistent font size and weight.

Text Input: The answer_input (an ft.TextField) is centered, has a clear hint, and uses keyboard_type=ft.KeyboardType.NUMBER to facilitate easy input of numerical sequences on mobile devices.

8. OUTPUT AND RESULTS

8.1 Game Flow

The game's flow is driven by state changes managed by the GameState enum, which dictates what content is displayed to the user.

Game Flow Description: The game initiates at the main menu, where the player selects a difficulty. This choice triggers the start_game function, which prepares the first round. Each round then systematically cycles through the following phases:

1. **Showing Original Word (GameState.SHOWING_ORIGINAL):** The player is presented with the correct, unscrambled word for a set duration, primarily for memorization.
2. **Showing Scrambled Word (GameState.SHOWING_SCRAMBLED):** The word's letters are rearranged, and each character is displayed with a number indicating its original sequential position. The player must now memorize the association between characters and their original block numbers.
3. **Waiting for Answer (GameState.WAITING_FOR_ANSWER):** The numerical block numbers are hidden, and the player is prompted to enter the sequence of numbers that would reassemble the word into its original form. An interactive text input field and a "Submit" button are provided.
4. **Showing Result (GameState.SHOWING_RESULT):** After submission or timer expiration, the game immediately displays feedback ("Correct!" or "Wrong!"). It also shows the original word, the correct numerical sequence, and the player's submitted answer for comparison. After a brief pause, the game automatically transitions to a new round.

Throughout these phases, the score and a countdown timer are consistently displayed at the top of the screen.

8.2 Results & Output Screen

Scoring and Accuracy Feedback: The game features a clear and immediate scoring system designed to motivate players and provide real-time performance feedback:

- **Correct Answer:** If the player successfully enters the correct numerical sequence, **+20 points** are added to their current score.

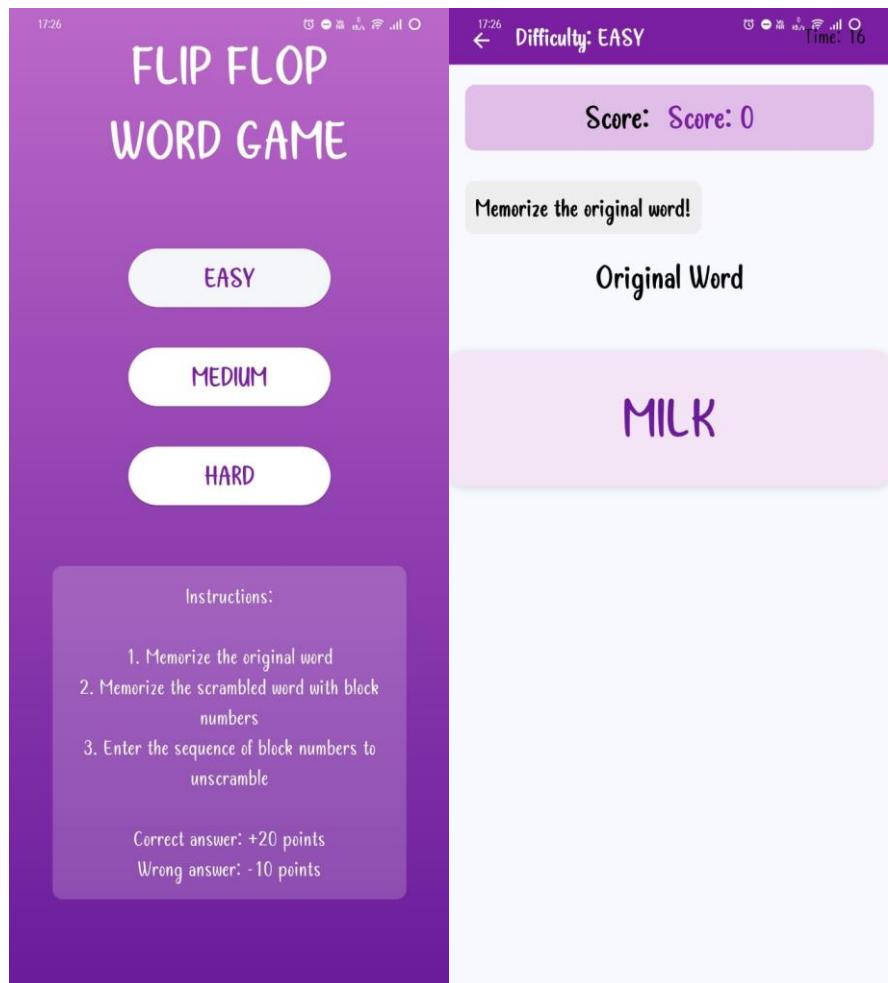
- **Incorrect Answer:** If the player's input is incorrect or if the time runs out before submission, **-10 points** are deducted from their score.
- **Minimum Score:** The player's score is prevented from dropping below 0, ensuring a non-negative score always.

The current score is prominently displayed at the top of the game screen and is updated immediately after each answer. The "Result" screen explicitly states whether the answer was correct or wrong and visually highlights the accuracy with an icon and corresponding color.

Output Screens (Visual Examples): The following figures illustrate the various screens that the user interacts with or observes during gameplay.

Screenshots:

1. Main Menu Screen
2. Game Play - "Showing Original Word" State
3. Game Play - "Showing Scrambled Word" State
4. Game Play - "Waiting for Answer" State
5. Game Play - "Result" State



17:26 ← Difficulty: EASY 17:27 ← Difficulty: EASY

Time: 24

Time: 0

Score: Score: 0

Score: Score: 0

Memorize the scrambled word with block numbers!

Enter the sequence of block numbers to unscramble the word

Scrambled Word with Block Numbers

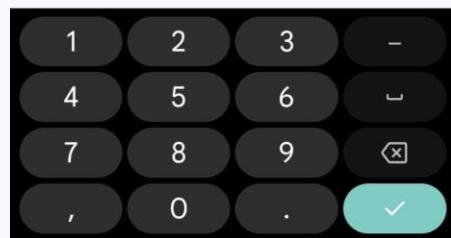
Enter the sequence of block numbers to unscramble the word:

K I L M
1 2 3 4

4231

Submit

Original word had 4 letters



17:27 ← Difficulty: EASY

Time: 2

Score: Score: 20

Correct! +20 points



Correct!

Original Word:

MILK

Correct Sequence:

4231

Your Answer:

4231

9. APK GENERATION

Flutter SDK Installation:

1. Flet relies on Flutter to compile your Python code into native mobile applications.
2. **Installation:** Download and install the Flutter SDK according to the official Flutter documentation for your operating system.
3. **PATH Configuration:** Ensure that the Flutter `bin` directory is added to your system's PATH environment variable so that the `flutter` command is accessible from your terminal.
4. **Verification:** Run `flutter doctor` in your terminal. All crucial checks (Flutter, Android toolchain, Android Studio, connected device) should show a green checkmark or indicate resolvable issues. Address any reported problems (e.g., accepting Android licenses with `flutter doctor --android-licenses`).
5. **Android SDK and Java Development Kit (JDK):**
6. The Android SDK (which includes the Android SDK Command-line Tools and Build-Tools) and a Java Development Kit (JDK) are fundamental for building Android applications.
7. **Installation:** Flet's build process often attempts to install these automatically if not found. However, a robust setup often involves installing Android Studio, which bundles the necessary SDK components and a JDK.
8. **ANDROID_HOME Environment Variable:** Set the `ANDROID_HOME` environment variable to point to your Android SDK installation directory.

Building the APK

9. Once your environment is correctly configured and verified by `flutter doctor`, navigate to the root directory of your Flet project in your terminal (e.g., `C:\Users\pranu\OneDrive\Desktop\Pranuu\ClgProjects\project3\flip-flop`).

10. Activate Virtual Environment (Optional but Recommended):

11. Bash
12. # On Windows
`.\venv\Scripts\activate`

```
# On macOS/Linux  
source venv/bin/activate
```

13. **Install Flet Build Dependencies:** Ensure you have the necessary Flet build tools installed within your virtual environment:

14. Bash

15. `pip install flet[build]`

16. **Execute the Build Command:** Run the following command to generate the release APK:

```
flet build apk
```

17. This command will perform several operations:

- It packages your Python application and its dependencies.
- It generates a temporary Flutter project.
- It invokes the Flutter build process to compile the native Android application.
- The generated APK will be a "fat" APK, supporting common Android architectures (`arm64-v8a` and `armeabi-v7a`).

18. **Output Location:** The final `.apk` file will typically be found in a path similar to:

```
your_project_directory/build/app/outputs/flutter-apk/app-release.apk
```

10. TESTING

Manual testing on an Android device is crucial to ensure the Flip-Flop Word Game functions correctly, performs smoothly, and provides a positive user experience in a real-world mobile environment. This involves a systematic approach to verify all features and aspects of the application.

Test Objectives:

- **Functionality:** Verify that all game mechanics, logic, and scoring work as expected.
- **Usability:** Assess the ease of use, intuitiveness of the UI, and overall user flow.
- **Performance:** Check for smooth transitions, responsiveness, and absence of significant lag or crashes.
- **Compatibility:** Ensure the app runs correctly across different Android versions and device screen sizes/densities.
- **Stability:** Identify any potential bugs, crashes, or unexpected behaviors under various scenarios.

Test Scenarios (Examples):

- **Installation & Launch:**
 - Successfully install the APK on a physical Android device (or emulator).
 - Launch the app and verify the main menu loads correctly.
- **Main Menu Navigation:**
 - Tap each difficulty button (EASY, MEDIUM, HARD) and verify the game starts with the correct difficulty settings.
 - Verify the "Back to Menu" button functions correctly during gameplay.
- **Game Flow - Easy Difficulty:**
 - Start an Easy game.
 - Verify the original word is displayed for the correct duration (20s).
 - Verify the scrambled word with blocks is displayed for the correct duration (25s).
 - Input a correct numerical sequence and verify +20 points are awarded.
 - Input an incorrect numerical sequence and verify -10 points are deducted (score doesn't go below 0).

- Allow the timer to run out in the "Waiting for Answer" phase and verify -10 points deduction.
 - Verify the result screen accurately shows "Correct!"/"Wrong!", original word, correct sequence, and user's answer.
 - Confirm the game automatically transitions to a new round after the result display.
- **Game Flow - Medium/Hard Difficulty:**
 - Repeat above scenarios for Medium and Hard difficulties, verifying correct word lengths and timer durations (30s/35s for Medium, 40s/45s for Hard).
- **Score Management:**
 - Verify the score updates correctly after each answer.
 - Confirm the score never drops below zero.
- **UI Responsiveness and Interaction:**
 - Verify all buttons are clickable and respond as expected.
 - Check that the `ft.TextField` accepts numeric input correctly and the virtual keyboard type is numeric.
 - Observe if UI elements are well-aligned and visible on different device orientations (if rotation is enabled) or screen sizes (if tested on multiple devices/emulators).
- **Background Operation:**
 - Minimize the app during gameplay and reopen it; verify game state resumes correctly or restarts appropriately.
- **Error Handling (Implicit):**
 - Attempt to input non-numeric characters (if the keyboard allows, though `keyboard_type=ft.KeyboardType.NUMBER` should prevent this on most devices).
 - Rapidly tap buttons to test for UI unresponsiveness or crashes.

11. CONCLUSION

In conclusion, the project aimed to develop an engaging and intuitive Flip-Flop Word Game leveraging the Flet framework to achieve seamless cross-platform deployment. The primary objective was to demonstrate the feasibility of building interactive, native-like applications for the Android platform predominantly using Python, thereby streamlining the development process.

The successful implementation of the game's core mechanics and logic—including randomized word selection, letter scrambling with positional numbering, and a time-based answer validation system—underscores the robustness of the chosen approach. The game's structure, driven by GameState and Difficulty enums, ensures clear progression and adaptable challenges for the user. Furthermore, the strategic use of Python's random, threading, and time modules effectively manages dynamic content generation and ensures a non-blocking, responsive user experience.

The user interface (UI) design prioritizes clarity and ease of interaction, adhering to mobile design principles. Through Flet's versatile layout (ft.Column, ft.Row, ft.Container) and customizable controls (ft.ElevatedButton, ft.TextField), the application provides a visually appealing and functional interface that guides the user through the game's various phases. The "Output & Results" section clearly illustrates these transitions, showcasing the real-time feedback from the scoring system and the visual presentation of each game state.

Ultimately, the successful APK generation for the Android platform highlights Flet's capacity to transform Python code into deployable mobile applications. This project serves as a compelling example of how Flet empowers Python developers to expand their reach into native application development, simplifying a historically complex process and proving Python's viability in building modern, interactive, and portable software.

12. FUTURE SCOPE

The Flip-Flop Word Game, while functional and engaging in its current state, presents several avenues for future enhancement and expansion. These potential developments could significantly enrich the user experience, broaden its appeal, and increase its complexity.

1. Expanded Word Database and Themed Packs:

- a. **Larger Vocabulary:** Integrate a much larger and more diverse dictionary to reduce word repetition and increase replayability.
- b. **Themed Word Packs:** Introduce word categories (e.g., animals, countries, food, science terms) that players can choose from, adding variety and catering to different interests. This could be a free or in-app purchase feature.

2. Enhanced Difficulty and Game Modes:

- a. **Custom Difficulty:** Allow users to customize parameters like word length, memorization time, and scrambled word display time.
- b. **New Game Modes:**
 - i. **Untimed Mode:** For casual play, focusing purely on memory and reconstruction without time pressure.
 - ii. **Challenge Mode:** Introduce a limited number of lives or incorrect attempts per round.
 - iii. **Progressive Difficulty:** The game automatically adjusts difficulty based on player performance.
 - iv. **Multi-word Sequences:** Instead of single words, present short phrases or multiple words to unscramble.

3. User Progression and Achievements:

- a. **High Score Tracking:** Implement a local high score system to record top scores for different difficulty levels.
- b. **Player Profiles:** Allow users to create profiles to track their progress over time.
- c. **Achievements/Badges:** Award virtual badges for completing milestones (e.g., "First Correct Answer," "100 Points," "Master of Hard Difficulty").

4. Improved User Interface (UI) and Experience (UX):

- a. **Visual Feedback:** More dynamic animations and visual cues for correct/incorrect answers, timer expiration, and state transitions.
- b. **Customization:** Allow players to choose different themes, color schemes, or font styles.
- c. **Sound Effects and Background Music:** Add subtle sound effects for button presses, correct/incorrect answers, and a calming background music track.

- d. **Onboarding Tutorial:** A simple, interactive tutorial for first-time users to quickly grasp the gameplay mechanics.

5. Cross-Platform Expansion:

- a. **iOS Deployment:** Extend the Flet build process to target iOS, making the game available on Apple's mobile ecosystem.
- b. **Desktop/Web Enhancements:** Optimize the UI and controls further for desktop environments (e.g., keyboard shortcuts for input) or web browsers (e.g., URL routing for different game states).

6. Persistence and Cloud Integration:

- a. **Local Storage:** Implement local storage to save game settings, high scores, and user progress, even if the app is closed.
- b. **Cloud Saves (Optional):** For multi-device play, integrate with a cloud service (e.g., Firebase, SQLite for local, or a simple Python backend) to synchronize user data and high scores across devices.

7. Multiplayer Feature:

- a. **Local Multiplayer:** Allow two players to compete on the same device by taking turns.
- b. **Online Multiplayer:** Implement real-time online multiplayer where players compete head-to-head or asynchronously to unscramble words. This would require a backend server.

13. BIBLIOGRAPHY

Flet Framework:

- [Flet: Build multi-platform apps in Python powered by Flutter](#)
- [Introduction - Flet Documentation](#)
- [Controls reference - Flet Documentation](#)
- [Packaging app for Android - Flet Documentation](#)

Python Programming Language:

- [Welcome to Python.org](#)

- [Our Documentation | Python.org](#)

Flutter SDK (Underlying for Flet Mobile):

- [Install - Flutter Documentation](#)
- [Docs | Flutter Documentation](#)
- [Flutter - Dart API docs](#)

Android Platform:

- [Develop for Android](#)
- [Android Platform Guide - Apache Cordova](#)