VIETNAM NATIONAL UNIVERSITY HCMC

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

Wordle

Topic: Game

Student:
Nguyen Thanh Dat
(24127020)

Professors:
M.S CS Huynh Lam Hai Dang
M.S CS Nguyen Thanh Tinh

Course:
Computational Thinking

October 23, 2025

Contents

1 Introduction

1.1 Project Overview

This report details the implementation of a Wordle game clone. This application was developed entirely in Python, utilizing the **Tkinter** library to build the graphical user interface (GUI).

The project is structured into two primary components:

- **Backend (‘back_end.py‘):** The ‘Wordle‘ class manages the core game logic, including selecting the secret word, processing each guess, and determining the win/loss state.
- **Frontend (‘front_end.py‘):** The ‘WordleApp‘ class is responsible for rendering the UI, handling all user input (from both the physical and virtual keyboards), and executing visual animations.

1.2 Requirements and Objectives

The main objective was to create a feature-complete and interactive Wordle clone. Based on the provided code, the key requirements met include:

- **Graphical User Interface (GUI):** The game features a full, intuitive graphical interface instead of running in a terminal.
- **Unlimited Version:** Users can immediately start a new game after one ends via the Play Again button.
- **User Feedback:** The application provides animations, such as the tile-flipping reveal and the row-shaking animation for invalid inputs, to enhance the user experience.

1.3 Game Rules

The game rules strictly follow the original Wordle implementation, as defined in the ‘Wordle‘ class:

- The player must guess a secret **5-letter word**.
- The player has a total of **6 attempts** (guesses).
- After each guess, every tile receives colored feedback:
  - **Correct (Green):** The letter is correct and in the correct position.
  - **Present (Yellow):** The letter is in the word but in the wrong position.
  - **Absent (Gray):** The letter is not in the word.
- The game ends when the player correctly guesses the word (a win) or runs out of the 6 attempts (a loss).

2 Game Logic (back_end.py)

The core mechanics of the game are encapsulated within the `Wordle` class, defined in `back_end.py`. This class is solely responsible for managing the game's state and logic, independent of the user interface.

2.1 Game Initialization

When a `Wordle` object is instantiated, it is initialized with a `secret` word.

- `self.secret`: Stores the secret word in lowercase.
- `self.guesses`: An empty list that will store the feedback for each guess.
- `self.game_over` and `self.win`: Boolean flags initialized to `False` to track the game's state.

2.2 Guess Processing and Feedback

The `check_guess` method contains the most critical logic in the project. It processes a user's guess and generates the color-coded feedback. This method correctly handles duplicate letters by using a two-pass system.

1. **First Pass (Correct):** The code iterates through the guess. If a letter is in the exact same position as in the secret word, it is marked as **'correct'**. To prevent this letter from being counted again, its position in a temporary copy of the secret word (`temp_secret`) is set to `None`.
2. **Second Pass (Present/Absent):** The code iterates a second time. For any letter not already marked 'correct', it checks if that letter exists anywhere in the `temp_secret`.
   - If it exists, the letter is marked as **'present'**, and its first occurrence in `temp_secret` is set to `None` to handle duplicates correctly.
   - If it does not exist in `temp_secret`, it is marked as **'absent'**.

Finally, the method updates the `self.game_over` and `self.win` flags based on whether the guess was correct or if the player has used all 6 attempts.

```python
def check_guess(self, guess: str):
    guess = guess.lower()
    feedback = []
    temp_secret = list(self.secret)

    # First pass: Check for 'correct'
    for i in range(len(guess)):
        if (guess[i] == temp_secret[i]):
            feedback.append({'letter': guess[i].upper(), 'status': 'correct'})
            temp_secret[i] = None
        else:
            feedback.append({'letter': guess[i].upper(), 'status': 'pending'})

    # Second pass: Check for 'present' and 'absent'
    for i in range(len(guess)):
        if feedback[i]['status'] == 'pending':
            if feedback[i]['letter'].lower() in temp_secret:
                feedback[i]['status'] = 'present'
                temp_secret[temp_secret.index(feedback[i]['letter'].
lower())] = None
            else:
                feedback[i]['status'] = 'absent'

    self.guesses.append(feedback)

    # Update game state
    if all(f['status'] == 'correct' for f in feedback):
        self.game_over = True
        self.win = True
    elif len(self.guesses) >= 6:
        self.game_over = True
        self.win = False

    return feedback
```

Listing 1: The two-pass feedback logic from back_end.py.

2.3 Frontend Interaction (front_end.py)

The frontend triggers this logic from the `submit_guess_event` method.

- It first validates that the guess is 5 letters long. If not, it shows a notification ("Your word must have 5 letters!") and triggers the `shake_grid` animation.
- If valid, it calls `self.game.check_guess(self.current_guess)` to get the feedback.
- This feedback is then passed to the `animate_flip_row` method to display the results visually.

3 Game UI and Interaction (front_end.py)

The `WordleApp` class, which inherits from `tk.Tk`, builds and manages the entire user interface and user experience.

3.1 Window and Widget Setup

Upon initialization (`__init__`), the application:

- Defines a `self.colors` dictionary to manage the application's theme.
- Sets the window title and a fixed geometry ("800x950").
- Attempts to load and display a background image (`bg.jpg`). If the image is not found, it defaults to a solid white background.

- Binds the `"<Key>"` event to the `handle_key_press` method, allowing it to capture all physical keyboard input.
- Calls `self.create_widgets()` to build the UI elements and `self.new_game()` to initialize the first game.

The `create_widgets` method is responsible for building the static UI elements. It creates a 6x5 grid of `tk.Label` widgets, storing them in `self.tiles` for future access. It also creates the virtual keyboard and notification labels.

3.2 Event Handling

User interaction is managed through several key methods:

- `handle_key_press(event)`: This is the central handler for the physical keyboard. It intercepts the `event.keysym` and routes the action to the appropriate method:
  - `'Return'` calls `self.submit_guess_event()`.
  - `'Delete'` or `'BackSpace'` calls `self.delete_letter()`.
  - Alphabetical keys call `self.add_letter(key)`.
- `add_letter(letter)` & `delete_letter()`: These methods manage the `self.current_guess` string and update the text on the `tk.Label` tiles in the current active row. They include checks to prevent input when the game is over, an animation is playing, or the row is full/empty.
- `create_virtual_keyboard()`: This method builds the on-screen keyboard. It dynamically creates `tk.Button` widgets for each key. Crucially, it assigns a `lambda` function to each button's `command` argument to call the correct handler (`add_letter`, `delete_letter`, or `submit_guess_event`).

```python
def handle_key_press(self, event):
    """Handles physical keyboard press events."""
    if self.is_animating: return
    key = event.keysym

    if key == 'Return':
        self.submit_guess_event()
    elif key == 'Delete' or key == "BackSpace":
        self.delete_letter()
    elif len(key) == 1 and 'a' <= key.lower() <= 'z':
        self.add_letter(key)
```

Listing 2: Snippet of the physical keyboard event handler.

3.3 Visual Feedback and Animations

To create a polished user experience, several animations are implemented using Tkinter's `after` method.

3.3.1 Shake Animation

If a user submits an invalid guess (e.g., not 5 letters), the `shake_grid` method is called. It uses a recursive function `do_shake` that slightly changes the `padx` padding of the `shake_container` frame in rapid succession, creating a horizontal shaking effect.

```python
def shake_grid(self, shakes=8, offset=5, delay=50):
    def do_shake(count):
        if count <= 0:
            self.shake_container.pack_configure(padx=0)
            return
        current_offset = offset if count % 2 == 0 else 0
        self.shake_container.pack_configure(padx=(current_offset, 0))
        self.after(delay, lambda: do_shake(count - 1))
    do_shake(shakes)
```

Listing 3: Shake animation logic from front_end.py.

3.3.2 Flip Animation

The most complex animation is `animate_flip_row`. When a valid guess is submitted:

1. The `self.is_animating` flag is set to `True` to block further input.
2. The method iterates through the 5 tiles in the current row.
3. For each tile, it schedules a call to `self.flip_tile` using `self.after()`, with each call delayed by 200ms more than the last. This creates the sequential flipping effect.
4. The `flip_tile` method itself is a simple two-step animation: it first turns the tile's background to the border color (simulating the start of the flip), then 100ms later, it reveals the true feedback color and letter.
5. Finally, a call to `check_game_over_state` is scheduled to run after the entire row animation is complete.

```python
def animate_flip_row(self, row_index, feedback):
    self.is_animating = True

    for col_index, tile_feedback in enumerate(feedback):
        tile = self.tiles[row_index][col_index]
        self.after(col_index * 200,
            lambda t=tile, fb=tile_feedback: self.flip_tile(t, fb))

    total_animation_time = len(feedback) * 200 + 200
    self.after(total_animation_time, self.check_game_over_state)

def flip_tile(self, tile, feedback):
    tile.config(text="", bg=self.colors['border'])

    def reveal():
        tile.config(text=feedback['letter'],
                    bg=self.colors[feedback['status']],
                    fg=self.colors['btn_text'])

    self.after(100, reveal)
```

Listing 4: Sequential flip animation logic.

3.4 Game State Management

- `check_game_over_state()`: This method is called after the flip animation. It clears the `self.is_animating` flag, updates the keyboard colors using `update_keyboard_colors`, and checks the `self.game.game_over` flag. If the game is over, it displays the win/loss message and shows the Play Again button.
- `new_game()`: This method resets the application for a new game. It creates a new `Wordle` instance with a new secret word, resets all tiles in the grid, hides the "Play Again" button, and resets all colors on the virtual keyboard.

```python
def check_game_over_state(self):
    self.is_animating = False
    self.update_keyboard_colors(self.game.guesses[-1])

    if self.game.game_over:
        if self.game.win:
            self.show_notification("You won!",
                duration=5000, color=self.colors['correct'])
        else:
            msg = f"Secret word: {self.game.secret.upper()}"
            self.show_notification(msg, duration=5000)

        self.after(1000, self.play_again_button.pack)

def new_game(self):
    secret = secretWord("output_words.txt")
    self.game = Wordle(secret)

    self.current_row = 0
    self.current_col = 0
    self.current_guess = ""

    if hasattr(self, 'tiles'):
        self.update_grid()
        self.hide_notification()
        if hasattr(self, 'play_again_button'):
                self.play_again_button.pack_forget()
        if hasattr(self, 'keyboard_frame'):
                self.reset_keyboard_colors()
```

Listing 5: Game state and new game handlers.

4 Conclusion

This project successfully fulfilled the objective of creating a fully functional Wordle game clone. By leveraging Python's **Tkinter** library, the application provides a complete graphical user interface, moving beyond a simple command-line program.

The project was successfully divided into two main components: a robust backend (`back_end.py`) that accurately manages game state and feedback logic, and an interactive frontend (`front_end.py`) that handles user input and visual rendering.

Key features were successfully implemented, including the core 6-try, 5-letter guess mechanic, an interactive virtual keyboard that updates its colors, and the required "unlimited" play mode via a "Play Again" button.

Furthermore, additional polish was added through visual animations like the sequential tile-flip reveal and the row-shake animation for invalid inputs. These features significantly improved the user experience, making the application feel responsive and complete. This project served as an excellent practical exercise in Python, GUI development with Tkinter, and event-driven programming.

5 References

References

[1] The New York Times. (n.d.). *Wordle*. Retrieved October 23, 2025, from https://www.nytimes.com/games/wordle/index.html

[2] tabatkins. (n.d.). *wordle-list*. GitHub. Retrieved October 23, 2025, from https://github.com/tabatkins/wordle-list

[3] Pillow. (n.d.). *Pillow (PIL Fork) Documentation*. Retrieved October 23, 2025, from https://pillow.readthedocs.io/en/stable/

6 Demo Video

A video demonstration of the completed Wordle game is available online. This video showcases the core gameplay, user input, visual animations (tile-flipping and shake), and the "Play Again" functionality.

Demo Link: Click here to watch the project demo

Or copy this URL into your browser: `https://youtu.be/-uHt0qac-s8`