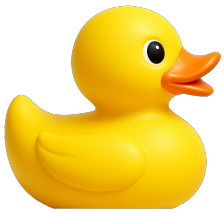
Duck Wordle

Creado por: Víctor Moreno
GitHub: @drexiing



---

1. Acerca de

Duck Wordle es un juego creado en Python 3 que consiste en el clásico juego de Wordle en que debes adivinar una palabra de 5 letras, pero la diferencia es que todas las palabras son relacionadas a **patos**. Este proyecto ha sido desarrollado en un tramo de 3 días, desde el lunes 5 hasta el miércoles 8 de mayo.

2. Estructura del Repositorio

El repositorio ha sido organizado de la siguiente manera:

```
DuckWordle/
|-- assets/
    |-- data/
        duck_words.txt
        es_words.txt
    |-- images/
        check.png
        duck.png
        erase.png
    |-- sounds/
        quack.mp3
        win.mp3
|-- core/
    |-- classes.py
    |-- utils.py
|-- docs/
    |-- assets/
        |-- duck.png
    |-- config/
        |-- commands.tex
        |-- usepackages.tex
    |-- sections/
        |-- about.tex
        |-- algorithm.tex
        |-- front.tex
        |-- license.tex
```

```
        |-- structure.tex
    |-- main.tex
|-- .gitignore
|-- LICENSE
|-- README-ES.md
|-- README.md
|-- index.py
|-- requirements.txt
```

2.1. Descripción de Carpetas y Archivos

- **assets/**: Archivos multimedia y de texto.
- **core/**: Código fuente principal del proyecto.
- **docs/**: Documentación del proyecto (incluye este archivo LaTeX).
- **.gitignore**: Archivo que ignora cache, variables de entorno, etc.
- **README-ES.md**: Archivo de introducción al proyecto en español.
- **README.md**: Archivo de introducción al proyecto en inglés.
- **index.py**: Archivo principal de ejecución del proyecto.
- **requirements.txt**: Dependencias del entorno.

3. Explicación de los Algoritmos

Los algoritmos de este programa han sido desarrollados completamente desde cero por mí. De todos ellos, uno de los que más desafíos me presentó —y que considero especialmente destacable— fue el siguiente:

```
def color\_match(self, guess: str):
    """
    Checks which characters in the user's word match the hidden word,
    and changes the colors of the boxes.

    Parameters:
        self (`self`): The instance of DuckWordle class.
        guess (`str`): The guessed word.
    """
    char\_list = list(self.hidden\_word)
    edited\_word = ""

    for c1, c2 in zip(self.hidden\_word, guess):
        if c1 == c2:
            edited\_word += c1
        else:
            edited\_word += "#"

    for i, (c1, c2) in enumerate(zip(self.hidden\_word, guess)):
        self.boxes = Label(
            self.boxes\_frame, width=4, height=2, bg=self.gray,
            text=c2.upper(),
```

```
            font=font.Font(family="Arial", size=15, weight="bold"),
            fg="white", highlightthickness=2, highlightbackground=self.gray
        )
        self.boxes.grid(column=i, row=self.row, padx=3, pady=3)

        # Right position (green)
        if c1 == c2:
            self.boxes["bg"] = self.green
            self.boxes["highlightbackground"] = self.green
            continue

        # Not in hidden word (gray)
        if (c2 not in self.hidden\_word) or
        (char\_list.count(c2) <= edited\_word.count(c2)):
            continue

        # Bad position in hidden word (yellow)
        self.boxes["bg"] = self.yellow
        self.boxes["highlightbackground"] = self.yellow
        char\_list.remove(c2)
```

Este código fue especialmente complejo de implementar al principio, debido a la dificultad de evitar falsos positivos al asignar colores a las letras. Sin embargo, tras varios intentos, logré una versión funcional y precisa.

El método `color_match` tiene como objetivo analizar la palabra ingresada por el usuario y compararla con la palabra oculta del juego. A partir de esa comparación, se modifica el color de las casillas gráficas para indicar el grado de acierto del intento.

Primero, se transforma la palabra oculta (`self.hidden_word`) en una lista de caracteres (`char_list`) para poder manipularla con mayor flexibilidad. Luego, se inicializa una cadena llamada `edited_word`, que almacenará el resultado parcial del análisis.

En el primer bucle `for`, se comparan letra por letra ambas palabras usando la función `zip()`. Si el carácter de la palabra oculta coincide con el carácter correspondiente de la palabra ingresada, se añade esa letra a `edited_word`; de lo contrario, se añade el símbolo `#`. Esto permite registrar únicamente los aciertos exactos.

A continuación, se ejecuta un segundo bucle, también con `zip()`, pero esta vez usando `enumerate()` para obtener el índice de cada letra. En cada iteración, se crea un nuevo objeto `Label`, que representa visualmente la letra correspondiente. Este objeto se configura con un tamaño específico, fuente, color de texto y de fondo, y se posiciona dentro de la interfaz mediante el método `grid`.
Luego, se aplica la lógica de color:

- Si la letra coincide exactamente en posición y contenido (`c1 == c2`), se colorea la casilla de **verde**, indicando un acierto total.
- Si la letra no aparece en la palabra oculta, o ya ha sido contabilizada como coincidencia en una posición previa (controlado mediante la condición `char_list.count(c2) <= edited_word.count(c2)`), se mantiene en **gris**, indicando un error o repetición innecesaria.
- Si la letra sí está en la palabra oculta pero en una posición incorrecta, se marca con **amarillo**. Para evitar que una misma letra se cuente varias veces, se elimina de la lista `char_list` mediante `char_list.remove(c2)`.

De este modo, logré desarrollar un algoritmo funcional para mi juego de Wordle.

4. Licencia

Este proyecto está bajo la licencia MIT. Ver archivo `LICENSE` para más detalles.