ConnorRoss_T1A3

Wordle Application

Features of Wordle

Wordle is comprised of 4 main components. Determining if the guess has the correct letters in the correct spots (Coloured Green) or correct letter in the incorrect spot (Coloured Yellow) and if it is not in the word at all (Coloured Red). The final component is guessing the word in the set amount of attempts.

```
colored_output = ""
for i, (f, g) in enumerate(zip(feedback, guess)):
    if f == 'G':
        colored_output += Fore.GREEN + g + " "
    elif f == 'Y':
        colored_output += Fore.YELLOW + g + " "
    else:
        colored_output += Fore.RED + g + " "
```

```
max_attempts = 6 if mode == 'regular' else 5
attempts_left = max_attempts
```

The code block on the left dictates the colour output based on the uses guess, utilising the PyPi Package, Colorama. The code on the right is how many attempts the user will be given based on their difficulty option.

Additional Features - Word Script

I wanted to make this game slightly more difficult than the standard Wordle. Wordle is comprised of 2315 words. My version of the game has over 34,000 words for combined between 5 and 7 letter words for the regular and hard difficulties. To do this I used the NLTK Library Package and created a script which would pull every 5 letter word and store it in a text document.

```
import string
from nltk.corpus import words

UPPERCASE = list(string.ascii_uppercase)

nltk_word_list = words.words()
filter_words= [word for word in nltk_word_list if len(word) in [5, 7]]

with open("word_list.txt", "w") as file:
    for word in filter_words:
        file.write(f" {word}\n")
```

Additional Features - Word Script cont.

Using the list that was generated, I could pull these words every time a new game was started and taking one at random.

```
def play_wordle():
    main()
    while True:
        mode = choose_game_difficulty()
        word_list = word_list_based_on_mode(mode)
        secret_word = random.choice(word_list)
        max_attempts = 6 if mode == 'regular' else 5
        attempts_left = max_attempts
        game_won = False  # Initialize game_won flag
```

This was then stored under “secret_word” which will be used against the user inputted guess.

Additional Features - Difficulty

The standard Wordle only offers one game per day and it is always a 5 letter word. I wanted to add an optional challenge of a harder mode that is 7 letter words in 5 guesses. To design this I needed to produce the option to the user and have the game changed accordingly.

```
def choose_game_difficulty():
        while True:
        # Added .lower() to make sure that if the user entered in captials only or mixed it would still be accepted.
            choice = input("Would you like to play (regular) or (hard) mode?").strip().lower()
            if choice in ["regular", "hard"]:
                return choice
            else:
                print("Invalid choice. Select either 'regular' or 'hard'.")
```

Once the user chooses an option the game will begin with their selected difficulty setting.

Additional Features - Replayability

The original version of Wordle by The New York Times Company is made to be a daily puzzle. I wanted to introduce a “play again” feature so that the user can play more games and change difficulty if they wish.

By adding the game_won variable I was able to determine if the loop for the game will continue or reset. If the game is won, the player is presented an option to play again. If the player loses by running out of attempts they will also be given the same choice.

```
            if feedback == ['G'] * len(secret_word):
                print(Fore.GREEN + f"Congratulations! You've guessed the word: {secret_word}")
                game_won = True
                break

        if not game_won and attempts_left == 0:
            print(Fore.RED + f"You've run out of attempts. The secret word was: {secret_word}")

        play_again = input("Do you wish to play again? (yes/no): ").lower()
        if play_again != "yes":
            print("Thanks for playing!")
            break  # Exit the main loop to end the game
```

Testing

Testing this application took place over several different functions. The user guess input and outcome for the word was the first test I tackled. Firstly, the colour output needed to be resembled correctly AND if there were duplicate letters in the word, it needed to show their positioning correctly. For example, if the user guessed “apple” but the secret word was “plate” then only one p should be yellow to inform the user that there is one of that letter in the word. In earlier stages, this was not the outcome. To get around this, I made the default that every word was red. The game then passes to check for greens first and then yellows. This checks for any duplicates in the passes.

```
    for i, g in enumerate(guess):
        if g == secret_word[i]:
            feedback.append('G')
        elif g in secret_word:
            feedback.append('Y')
        else: feedback.append('R')
```

```
colored_output = ""
for i, (f, g) in enumerate(zip(feedback, guess)):
    if f == 'G':
        colored_output += Fore.GREEN + g + " "
    elif f == 'Y':
        colored_output += Fore.YELLOW + g + " "
    else:
        colored_output += Fore.RED + g + " "
```

Testing

Another factor of testing was when the user was guessing a word. In early testing I found that the user could put a space in their word and it would attempt to register it. As the word is not in the list it would not matter. However, I wanted to ensure the user was repeated feedback into what the word requires.

```
while attempts_left > 0:
    attempt_number = max_attempts - attempts_left + 1
    guess = input(f"Attempt {attempt_number}: Guess a {len(secret_word)}-letter word: ").lower()

    if len(guess) != len(secret_word) or ' ' in guess:
        print(f"Please enter a {len(secret_word)}-letter word.")
        continue
```

The user was also able to enter capitals and the word would not register. To resolve this, any time a word is input it is followed by a .lower() method. The word imported from the list follows the same rule so that it remains consistent no matter what word is chosen and no manual adjustment is required.

Walkthrough

```
connor@DESKTOP-LTD0L3F:~/ConnorRoss_T1A3$ bash wordle_script.sh
```

As written in the README.md file, if the user uses this command all dependencies will be downloaded and the game will automatically start.

I added an ASCII art to help bolster the user interface of the terminal and add a bit of variety from just looking at text.

Walkthrough

```
Would you like to play (regular) or (hard) mode?regular
Attempt 1: Guess a 5-letter word: crate
c r a t e
```

Choosing the regular mode, the user is given 6 attempts to guess the 5 letter word. Using this guess, the t is in the correct spot and the e is in the word but not in that spot.

Once the user guesses the correct word, they receive a message:

```
Congratulations! You've guessed the word: jetty
Do you wish to play again? (yes/no):
```

If they select yes, the app will choose a new word and start from the beginning where they can change difficulty if they wish. If the user selects no, they will receive a message that says “Thanks For Playing”

Reflection

This was a great learning opportunity. I was expanding on my knowledge gained thus far and was happy with the outcome of the application.

My favourite part is being able to select a different difficulty. I am an avid Wordle player so being able to play a harder mode is very enjoyable.

If I could change one thing, it would be the word list. As previously mentioned, the script in making_words.py, there is 34,372 words in total for the 5 and 7 letter words combined. The standard version of wordle has just over 2000 and allows plurals. E.g. “ducks”. The NLTK Library is very extensive and I would change this back down to be more fitting of other words which people likely recognise.

Reflection

Another 2 features I was hoping to add were a custom mode, were the user could input a word for their friend to guess. This would have been an easy addition by creating a third mode and then changing the secret word to take an input rather than pulling from the text file. Due to time constraints I was not able to do this.

Another feature would have been the coloured boxes you see at the end of a wordle telling you how fast you got it. See an example below. This could be done using ASCII art for the box and utilising colorama's package. By writing a function for it to be created on each attempt and then displaying all attempts at the end of the game the user would be able to see the boxes.

"Score Boxes -Wordle Example"

Thanks For Watching!