PyWordle

Alison Gale

Background

This project is inspired by the popular word-guessing game, Wordle. In the game, users have six guesses to identify a hidden five-letter word. With each guess the user gets feedback on whether each letter isn't in the hidden word, is in the word in the correct location, or is in the word in an incorrect location. This game inspired a series of spin-off games with different themes and a swath of solvers to minimize the number of guesses needed to identify the hidden word.

Requirements

For this package, there were two sets of users I considered. The first category of users were ones wanting to create themed versions of the game who would want a game engine to drive it. The second category of users were ones building solvers or other algorithms that would need to interact with the game state. For reasons discussed further in the design section, I chose to focus on the first category of users.

Given that I wanted to build a game engine that would allow for creating themed versions of Wordle, the main requirement was implementing a mechanism to enforce the rules of the game. This includes enforcing the number of guesses, whether guesses are valid, and enforcing "hard mode" constraints where information from previous guesses must be used in subsequent guesses. Outside of that, I wanted to provide the ability to provide a custom set of solutions to choose from and the ability to pick a solution for a given game. I also wanted to provide good validation of inputs and data to ensure that the game is played correctly.

Design

This package utilizes two modules, Wordle and Game. The Wordle module represents a themed version of the original game, while the Game module represents a specific instance of the game where a user is making guesses towards the hidden solution. The diagram below highlights the methods in each module:

Wordle	**Game**
__init__(solutions) start_game(solution, hard_mode) __repr__()	__init__(solution, hard_mode) guess(word) is_valid(word) get_status() __str__()

	__repr__()

When designing this package, I initially planned to make a general purpose module that could serve the use cases of building a game engine and building a solver. As I worked through the method signatures and the state needed to represent this logic I found that the methods used by the solver had almost no overlap in functionality with the game engine methods. For example, when building a solver you are interested in which words are eliminated based on information from previous guesses, but this isn't useful for a game engine and would require a completely different data structure to encode some subtle pieces of information.

I focused on abstracting out parts of the logic that would allow the package to be extended to support additional use cases. Currently the library only supports five letter words, but instead of hardcoding that length in different places, I used a variable to represent that constant so it could be easily changed in the future if varying length words were to be supported.

Within the Game module, I utilized common libraries like enum and defaultdict to represent the state of previous guesses in order to enforce hard mode constraints. When hard mode isn't enabled, the process of validating guesses is very straightforward in checking whether a word is a valid word in the English language. But when hard mode is enabled or when printing out the string representation of a game board, there is a complex set of constraints that must be checked.

Within the design, I relied heavily on dunder methods like __str__ and __eq__ to allow for a more Pythonic style. So the pretty-print version of the game board is implemented in a __str__ method. And when I needed to properly deduplicate custom objects in a set, I used the __eq__ method to allow for proper comparisons between elements. I considered having the game return an object representing the state for the user to handle on their own, but I found that there is a lot of complex logic in how to print out the board based on the previous game state that would be useful to incorporate into the game engine. This allowed the module to be deeper as well because it doesn't change the public API, but does make for a much richer functionality that is provided by the public API.

Usage

This library allows you to easily create a themed instance of the Wordle module by providing a set of possible solutions. From this instance you can create individual games. By default it has hard mode disabled and selects a random solution. From there the user can make guesses towards the solution and print out the current state of the game. Here is an example of a basic interaction with the package:

```
from pywordle import Wordle

wordle = Wordle(WORD_LIST)
```

```
game = wordle.create_game()
game.guess("SPILL")
print(str(game))
```

Comparison

I will compare my library to the wordle-python library (https://pypi.org/project/wordle-python/) that is available on PyPI. The main difference between our libraries is that the wordle-python library handles all the I/O with the user while my library allows whoever uses the library to control receiving user input and passing it to the game. Here is an example of how that library is used:

```
import wordle

wordle.Wordle(word = wordle.random_answer(), real_words = True).run()
```

This difference allows the wordle-python library to have a single module as part of the public API because you just call a run method to start the game. I could have consolidated all of my logic into a single module that would take in a word list and settings, but the downside of this would be that for each game you create you would have to revalidate the list of solutions. My design allows that validation to be done once which improves performance for repeated games.

The wordle-python library is a highly specialized module that only lets you play with a specific set of words. It is deep because the public API is tiny, but it is not very extensible or flexible. This means that it is very useful for someone who wants to play that specific game on the command line, but if someone wanted to make a Flask app that allowed someone to play the game on a website they wouldn't be able to use the library.