Programming assignment: Wordle

Introduction

The programming test is going to examine the following aspects:

- Understanding of abstract problem
- Decision making based on requirements
- Code quality and organization
- Documentation
- Source code repository practice

Language is not specified and any language can be used, but we prefer some languages over the others:

- C, C++, C#, Go, Java, Javascript, Python, Rust, Typescript

It is not required to use only 1 language.

Assignment Content

This assignment will consist of 4 tasks:

1. Normal wordle
2. Server/client wordle
3. Host cheating wordle
4. Multi-player wordle

Task 1 is expected to be done before tasks 2-4. Candidate is not required to finish all tasks.

There is no precedence on which of tasks 2-4 should be done first, but the solution is expected to be only 1 code base that implements the tasks. e.g. If task 1, 2 and task 4 are done, then the solution should be a server/client wordle that supports multiple-player.

Task 1: Normal wordle

Reference: https://www.nytimes.com/games/wordle/index.html

The game will select a 5-letter word (aka. **`answer`**) from a predefined list (configurable), all 5-letter words are expected to consist of English alphabet only and case-insensitive.

In the assignment, the scoring rule will be the same as the reference game:

- **Hit**: the letter is in the correct spot of **`answer`**.
- **Present**: the letter is in the **`answer`** but wrong spot.
- **Miss**: the letter is not in the **`answer`**.
- The exact logic should refer to the game of reference.

The solution is expected:

1. The wordle must have at least 2 configurations:
    - The maximum number of rounds before game over
    - The list of 5-letter words.
2. The player can be identified as win if they guess the answer within the max allowed rounds
3. The player can be identified as lose if they failed to guess the answer after the max allowed rounds

Task 2: Server/client wordle

Based on task 1, modify the solution to support the server / client model.

The scoring rule will be the same as task 1.

The solution is expected:

1. To have expectation of task 1
2. Client side will not know the answer before the client guessed correctly or game over.
3. Server side will have input validation.

Task 3: Host cheating wordle

Reference: https://absurdle.online/

Based on task 1, modify the solution to support the “host cheating” feature.

The game flow will be similar to task 1, but the host will not select the `answer` at the beginning of the game. Instead it will keep a list of candidates based on the received input.

The scoring rule will be the same as task 1. But there are new rules when comparing to other guess:

1. More **Hit** will have higher scores.
2. If the number of **Hit** is the same, more **Present** will have higher score.

The list of candidates after each round should meet the criteria:

- They should have lowest score in the finished round
- They should match the result of previous rounds.

The solution is expected:

1. To have expectation of task 1
2. The external observer cannot tell if the host is cheating based on the guesses.

There is an example at the end of the document.

Task 4: Multi-player wordle

Based on task 1, modify the solution to support multi-player feature.

There is no limitation on how a multi-play wordle should work. Here are some examples:

- Each of Player A and Player B provide a 5-letter word, and let others guess.
- 2 players guessing the same word, while being able to monitor their opponents’ progress.

The solution is expected:

1. To design the game play, and state the reason / trade-off of the considerations
2. To have expectations of task 1, except how to determine the player wins.
3. There should be interaction between players.
4. If the **answer** is not provided by players, it will be the same across players.
5. Have clear rules on how players win, lose or tie.

Bonus: Bells and whistles

Any enhancements / features that can be added in tasks 1-4.

Examples:
- Stores the high score, and displays before the game starts or after the game is over.
- Animation on the game board UI

The solution is expected:
1. A document that describes the bonus feature (not necessary to be implemented)
2. Benefits to the project, or how it improves the user experience
3. Ideas that implemented will have more points

Measurement Criteria

Understanding of abstract problem

- The difference between the solution and our expectation.
- How many good questions that are asked to clarify ambiguous requirements

Decision making based on requirements

- Trade off that made to reduce complexity / enhance maintainability / improve user experience
- Documented trade-off consideration

Code quality and organization

- The features that are implemented.
- The bells and whistles that are documented and implemented.
- Folder / file / code structure that help locating codes to related features / functionalities
- Naming conventions / coding practice / style that help to trace code / readability / refactoring

Documentation

- Comments on code that help to understanding / trace / refactor
- How to setup / test / run the project
- Decision / trade-off made during development

Source code repository practice

- How the repository is managed
- How good are the commits in the repo history

Example

Here is examples of task 3, in text base mode.
The program will print different letters based on user's input:

- '0' means **Hit** (letter is in the target word, and correct spot)
- '?' means **Present** (letter is in the target word, but not in correct spot)
- '_' means **Miss** (letter is not in the target word).

Example 1

The list of words are [`HELLO`, `WORLD`, `QUITE`, `FANCY`, `FRESH`, `PANIC`, `CRAZY`, `BUGGY`].

Input	Output	Explanation
`HELLO`	`_____`	The program found there are words that do **NOT** match the input (HELLO). Remaining candidates: [`FANCY`, `PANIC`, `CRAZY`, `BUGGY`]
`WORLD`	`_____`	The program found there are words that do not match in 2 rounds. Remaining candidates: [`FANCY`, `PANIC`, `BUGGY`]
`FRESH`	`_____`	The program found there are words that do not match after 3 rounds. Remaining candidates: [`PANIC`, `BUGGY`]
`CRAZY`	`?_?__`	The program found that PANIC has 2 **Present**(C & A), and BUGGY has 1 **Hit** (Y). As PANIC has less **Hit** than BUGGY, the finalized answer is PANIC
`QUITE`	`__?__`	Just like a normal wordle.
`FANCY`	`_OO?_`	Player failed to guess the word in 6 rounds.

Example 2

The list of words are [HELLO, WORLD, QUITE, FANCY, FRESH, PANIC, CRAZY, BUGGY, SCARE].

Input	Output	Explanation
BUGGY	_____	The program found there are words that do **NOT** match the input (BUGGY). Remaining candidates: [HELLO, WORLD, FRESH, PANIC,SCARE]
SCARE	___?_	The program found all remaining words has some score, [H**E**LLO, WO**R**LD, F**RES**H, P**ANIC**, **SCARE**], [HELLO, WORLD] has same score (1 Present), so the program will select 1 as the answer (as they cannot coexists)
WORLD	00000	The remaining word is WORLD, so the player can guess it and win.