Word Wizard

The intention behind this exercise is to hand-hold learners who have no/little coding experience and build a simple web game, inspired by the phenomenally popular **Wordle** (https://www.powerlanguage.co.uk/wordle/). While we will not be making Wordle, we will make a simplified edition that has limited features to give a taste of web application development to the learners.

Pre-class: This should be an interactive session. Please walk through these steps to ensure learners can follow in the workshop

1. Make sure they have Node.js installed. You can ask them to open the terminal/command prompt and type **node -v** and report back on the version they see.
2. Make sure they have VS Code installed.
3. Make sure they have downloaded and extracted the project archive from : https://bit.ly/01-camp-alpha

Overview

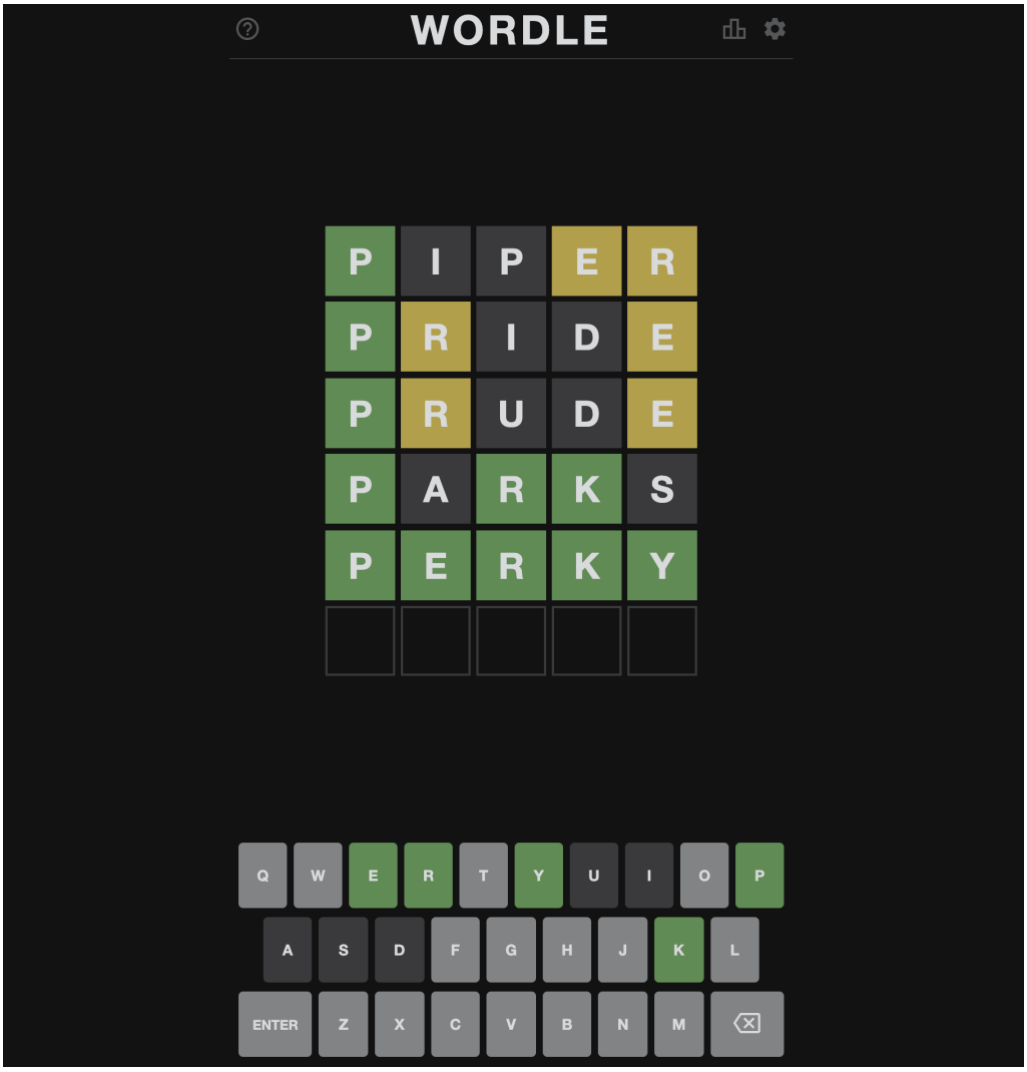
Wordle is a web-based game that has taken the world by storm. Every day a new word stands to be guessed over 6 attempts during which characters in the correct position are shown with green tiles, characters that are in the word but not in the correct position are shown with orange tiles and characters which do not exist in the word are shown in grey. Only word in

the game's dictionary is permitted.

Our Version: Try out the completed game and you can show it to the class as well by going to https://living-rifle.surge.sh

Approach

1. Begin by explaining what the actual Wordle game (https://www.powerlanguage.co.uk/wordle/) is all about - "Every day you have a new word to guess in 6 or less attempts. When you type in a guess, each character appears in one of three colours. If its green, then the character is in the right position in the word of the day, if the character is orange, then the character exists in the word but is not in the right position. Finally, if the character is grey, then the character doesn't exist in the word at all. Once you get a couple of greens and oranges, you must play smartly to avoid the grey ones to guess the correct word. The words you type must also be in the game's dictionary so that arbitrary words aren't accepted and spoil chances."



2. In terms of pseudocode, here is how the core of the game might look like:

```
// When the game is loaded, let's fetch a random word from the dictionary of words
SET word as a new randomly chosen word from the dictionary

// We want to give 6 chances to the user. This variable will keep track of the available chances.
SET totalChances to 6

// We want to get into action whenever the user presses & releases a key in an input box
WHEN the user presses and releases a button on the keyboard after typing in the input box
    SET guess to the word the user typed in the input box.

    // We should only proceed if the user pressed 'Enter' and the word typed is a 5 letter word and if we have chances left
    IF 'Enter' was pressed AND guess is a 5-letter word AND totalChances is greater than 0 THEN

            // If the user's typed word (guess) is the same as the one picked by the game, then the game is won!
            IF guess is the same as the word THEN
                CALL gameWon so the game is ended, and the user is congratulated
                SET input box as empty
                STOP
            ENDIF

            // Otherwise, let's process the word typed by the user. First up, it should be valid as per the dictionary
            IF the guess is in the game's dictionary, THEN
                // We'll loop across the guess word and match every character with the chosen word
                FOR every character in word
                    // If the character is in the same position as in the word, then show it in green
                    IF the character in guess is equal to the character in word THEN
                        DISPLAY character as a green tile
                    // If the character is in NOT in the same position but appears in the word, then
                    // show it in orange
                    ELSE IF the character in guess exists in the word
                        DISPLAY character as an orange tile
                    // If the character isn't present anywhere in the word, then show it in grey
                    ELSE
                        DISPLAY character as a grey tile
                    ENDIF
                ENDFOR

                // At every attempt by the player, subtract 1 from the total available chances
                SET totalChances as totalChances minus 1
            ENDIF

            // Let's empty out our input box at every attempt so the user can type in a new word
            SET input box as empty

        ENDIF

        // If the player runs out of available chances, then end the game by displaying 'Better luck next time'
        IF totalChances is less than or equal to 0 THEN
            CALL gameOver so the game is ended, and we show 'Better luck next time' to user
            STOP
        ENDIF
ENDWHEN
```

3. In terms of implementation, you're provided with two folders:
    a. **end** - This folder contains the result of the project. Install all dependencies using **npm install**, then run **npm run dev** to bring up the development server (hot-reloading). The project uses the Parcel bundler for simplicity. You can run **npm run build** to produce a distro in the *dist* folder.

    b. **start** - This is what the learners and you should use in the class as you build the app. Just setup dependencies and bring up the development server and start writing in the **src/js/app.js** file.

4. The **start** app is partially built:
    a. The stylesheet is pre-built and should be shown to give a quick glimpse into what it looks like. It resides in the **src/css/style.css** file.

    b. The HTML file **src/index.html** needs to be populated with the HTML structure inside the body. Remove the H1 element and build as per the **end/src/index.html** document. Talk about how you're creating elements which are then given the look and feel by the CSS in the stylesheet.

    c. There are three JavaScript modules that are pre-built:
        i. **dictionary.js** - This exports an Array of about 1566 permissible words
        ii. **words.js** - This exports two functions:
            1. **getWord()** : When called, it returns a random word from the dictionary
            2. **isInDictionary(string)** : When called with a word, it validates the presence of the word in the dictionary with a Boolean result
        iii. **tiles.js** - This exports three functions:
            1. **gameOver()** : When called, it displays a message which says 'Better luck next time' and disables the input field.
            2. **gameWon(string)** : When called, it displays the message 'Great Work! The word was: *chosenWord*' and disables the input field.
            3. **addTile(string, color)** : When called with a character as the first argument and one of three strings 'green', 'orange', 'grey' as the color choices, adds a tile on the game board with the desired color and the character.

    d. You will be working in the **index.html** and the **app.js** file. Handhold the learners as they write their first lines in HTML and JavaScript. The above three modules should be explained without getting into complexities.

    e. The algorithm used in this game is a naïve version which does not consider repetitive characters. For instance, if the chosen word was 'ports' and your guess was 'peers', the character 'e' would be highlighted as orange twice even though it only exists once. This is because we're doing positional and presence-based tests and not considering character counts for brevity and

simplicity. To prevent such anomalies, the dictionary has been toned down such that it does not contain any word where a character appears more than once.

5. **BONUS**: As a bonus step, get learners to host their completed and working apps on **surge.sh** as follows:
    a. Install surge globally by **npm install --global surge**
    b. Once they've built the game and is working offline, they should run **npm run build**
    c. The above step will produce the final package to be uploaded in the **dist** folder.
    d. From the terminal, go into the **dist** folder, then type **surge** to start the tool.
    e. The user may be asked to register once on the terminal. Once done, press enter to confirm the current folder as **dist**
    f. Then **surge** will recommend a randomly chosen sub-domain name. Press 'enter' and it'll upload and deploy the website in a few seconds. Then open the address shown in a browser and your site is online for free.