Project Instructions

Open the project files in your favorite text editor and preview them in the browser. Also, it's a smart idea to test your project in multiple browsers!

Github

- Have a GitHub account and create a new repo for this project.
- Create a README.md file for your repo that explains what the project is and anything your fellow developers might need to know to use the project.

Download the project files

- An index.html file with the basic HTML markup.
- A css folder containing two CSS files with the basic CSS styles for the game.
- An images folder containing two image files that are used for the game's scoreboard.
- A js folder containing the following JavaScript files:
    - **app.js** to create a new instance of the Game class and add event listeners for the start button and onscreen keyboard buttons.
    - **Phrase.js** to create a Phrase class to handle the creation of phrases.
    - **Game.js** to create a Game class with methods for starting and ending the game, handling interactions, getting a random phrase, checking for a win, and removing a life from the scoreboard.
- An example_phrase_html.txt text file containing an example of the HTML for displaying a phrase.

Understand the rules of the game:

- The player's goal is to guess all the letters in a hidden, random phrase. At the beginning, the player only sees the number of letters and words in the phrase, represented by blank boxes on the screen.
- The player clicks an onscreen keyboard to guess letters in the phrase.
- The letter is disabled on the onscreen keyboard and a player can't select that letter again.
- If the selected letter is in the phrase at least once, the letter and its position in the phrase is highlighted on screen. All instances of the letter are made visible (so if there are 3 A's, all of the A's in the phrase appear at once).
- If the selected letter is not in the phrase, one of the player's hearts in the scoreboard is changed from a "live" heart to a "lost" heart.
- The player keeps choosing letters until they reveal all the letters in the phrase, or they make five incorrect guesses.

Create the *Phrase* class in the Phrase.js file.

- The class should include a constructor that receives a phrase parameter and initializes the following properties:

    - phrase: this is the actual phrase the Phrase object is representing. This property should be set to the phrase parameter, but converted to all lower case.
- The class should also have these methods:
    - **addPhraseToDisplay()**: this adds letter placeholders to the display when the game starts. Each letter is presented by an empty box, one li element for each letter. See the example_phrase_html.txt file for an example of what the rendered HTML for a phrase should look like when the game starts, including any id or class attributes needed. When the player correctly guesses a letter, the empty box is replaced with the matched letter (see the showMatchedLetter() method below). Make sure the phrase displayed on the screen uses the letter CSS class for letters and the space CSS class for spaces.
    - **checkLetter()**: checks to see if the letter selected by the player matches a letter in the phrase.
    - **showMatchedLetter()**: reveals the letter(s) on the board that matches the player's selection. To reveal the matching letter(s), select all of the letter DOM elements that have a CSS class name that matches the selected letter and replace each selected element's hide CSS class with the show CSS class.

Create the Game class in the Game.js file.

- The class should include a constructor that initializes the following properties:
    - missed: used to track the number of missed guesses by the player. The initial value is 0, since no guesses have been made at the start of the game.
    - phrases: an array of five Phrase objects to use with the game. A phrase should only include letters and spaces— no numbers, punctuation or other special characters.
    - activePhrase: This is the Phrase object that's currently in play. The initial value is null. Within the startGame() method, this property will be set to the Phrase object returned from a call to the getRandomPhrase() method.
- The class should also have these methods:
    - **startGame()**: hides the start screen overlay, calls the getRandomPhrase() method, and sets the activePhrase property with the chosen phrase. It also adds that phrase to the board by calling the addPhraseToDisplay() method on the active Phrase object.
    - **getRandomPhrase()**: this method randomly retrieves one of the phrases stored in the phrases array and returns it.
    - **handleInteraction()**: this method controls most of the game logic. It checks to see if the button clicked by the player matches a letter in the phrase, and then directs the game based on a correct or incorrect guess. This method should:
        - Disable the selected letter's onscreen keyboard button.
        - If the phrase does not include the guessed letter, add the wrong CSS class to the selected letter's keyboard button and call the removeLife() method.
        - If the phrase includes the guessed letter, add the chosen CSS class to the selected letter's keyboard button, call the showMatchedLetter() method on the phrase, and then call the checkForWin() method. If the player has won the game, also call the gameOver() method.

- **removeLife()**: this method removes a life from the scoreboard, by replacing one of the liveHeart.png images with a lostHeart.png image (found in the images folder) and increments the missed property. If the player has five missed guesses (i.e they're out of lives), then end the game by calling the gameOver() method.
- **checkForWin()**: this method checks to see if the player has revealed all of the letters in the active phrase.
- **gameOver()**: this method displays the original start screen overlay, and depending on the outcome of the game, updates the overlay h1 element with a friendly win or loss message, and replaces the overlay's start CSS class with either the win or lose CSS class.

Update the app.js file.

- Create a new instance of the Game class and add event listeners for the start button and onscreen keyboard buttons:
    - Add a click event listener to the "Start Game" button which creates a new Game object and starts the game by calling the startGame() method.
    - Add click event listeners to each of the onscreen keyboard buttons, so that clicking a button calls the handleInteraction() method on the Game object. Event delegation can also be used in order to avoid having to add an event listener to each individual keyboard button. Clicking the space between and around the onscreen keyboard buttons should not result in the handleInteraction() method being called.

Resetting the gameboard between games.

- After a game is completed, the gameboard needs to be reset so that clicking the "Start Game" button will successfully load a new game.
    - Remove all li elements from the Phrase ul element.
    - Enable all of the onscreen keyboard buttons and update each to use the key CSS class, and not use the chosen or wrong CSS classes.
    - Reset all of the heart images (i.e. the player's lives) in the scoreboard at the bottom of the gameboard to display the liveHeart.png image.

Add code comments.

Cross-Browser consistency

- Google Chrome has become the default development browser for most developers. With such a selection of browsers for users to choose from, it's a good idea to get in the habit of testing your projects in all modern browsers.

EXTRA CREDIT

Add keyboard functionality

- Let players use their physical computer keyboard to enter guesses. You'll need to use the keydown or keyup event.

Making the project your own

- The general layout should remain the same, but feel free to make the project your own by experimenting with things like color, background color, font, borders, shadows, transitions, animations, filters, etc.