Project Instructions

Project 6: Game Show App

Before you start:

1.) **As with the previous projects, you'll submit your finished working using GitHub. If you need a reminder on how to use GitHub and GitHub desktop to create a new repository check out this workshop: Share Your Projects WIth GitHub Desktop.**

2.) **Download the project files. We've supplied several files for you to use:**
- An index.html file with the basic HTML markup.
- A css folder containing styles.css, a file with the basic CSS you'll need.
- An images folder containing liveHeart.png and lostHeart.png
- You will need to create your own external JavaScript file.

3.) **Look at the project 6 study guide**:
https://drive.google.com/file/d/1tijAzRqxbcUZ8Ylj8onWKmJA4Fnmsf1V/view

Instructions:

1.) **In this project, you'll create a word guessing game. Players will click letters from an onscreen keyboard to try to guess a random phrase. The player's goal is to guess all the letters in the phrase. A player can keep choosing letters until they make five incorrect guesses. Letters guessed correctly will appear in the phrase. Letters guessed incorrectly will be counted at the bottom of the screen.**

2.) **Get the elements you'll need from your HTML**
- Get the element with the ID of qwerty and save it to a variable.
- Get the element with the ID of phrase and save it to a variable.
- Create a missed variable, initialized to 0, that you'll use later to keep track of the number of guesses the player has missed (remember, if the player guesses wrong 5 times, they lose the game)

3.) **Attach a event listener to the "Start Game" button to hide the start screen overlay.**

4.) **Create a phrases array that contains at least 5 different phrases as strings.**

- Make sure that each phrase contains only letters and spaces, so players won't need to guess punctuation or special characters.

5.) **Create a *getRandomPhraseAsArray* function.**
   - This function should randomly choose a phrase from the phrases array and split that phrase into a new array of characters. The function should then return the new character array.
   - Keep in mind that you'll need to write this function so that it is reusable-- meaning that it can take any given array of strings (with no special characters) and return an array of characters. To do that, you'll write the function so that it takes an array as a parameter.
   - To use the function, you'll pass in the phrases array as an argument when you call the function.

6.) **Set the game display.**
   - Create an addPhraseToDisplay function that loops through an array of characters. Inside the loop, for each character in the array, you'll create a list item, put the character inside of the list item, and append that list item to the #phrase ul in your HTML. If the character in the array is a letter and not a space, the function should add the class "letter" to the list item.

You'll need to write the addPhraseToDisplay function so that it can take any array of letters and add it to the display. To do that, the function will need to take an array as a parameter:

```
function addPhraseToDisplay(arr){
    // do stuff any arr that is passed in, and add to `#phrase ul`
}
```

To use the function, you'll get the value returned by the getRandomPhraseAsArray, save it to a variable, and pass it to addPhraseToDisplay as an argument:

```
const phraseArray = getRandomPhraseAsArray(phrases);
addPhrasetoDisplay(phraseArray);
```

7.) **Create a checkLetter function.**
   - The checkLetter function will be used inside of the event listener you'll write in the next step.
   - This function should have one parameter: the button the player has clicked when guessing a letter.
   - The checkLetter function should get all of the

- elements with a class of “letter” (remember that we added the letter class to all of the letters and none of the spaces when we made the game display). The function should loop over the letters and check if they match the letter in the button the player has chosen.
- If there’s a match, the function should add the “show” class to the list item containing that letter, store the matching letter inside of a variable, and return that letter.
- If a match wasn’t found, the function should return null.

8.) **Add an event listener to the keyboard.**
- Use event delegation to listen only to button events from the keyboard. When a player chooses a letter, add the “chosen” class to that button so the same letter can’t be chosen twice. Note that button elements have an attribute you can set called “disabled” that when set to true will not respond to user clicks. See the MDN documentation for more details.
- Pass the button to the checkLetter function, and store the letter returned inside of a variable called letterFound. At this point, you can open the index.html file, click any of the letters on the keyboard, and start to see the letters appear in the phrase.

9.) **Count the missed guesses in the game.**
- If the checkLetter function returns a null value, the player has guessed the wrong letter. In the keyboard event listener, after checkLetter is called, write a statement to check the value of the letterFound variable. If the value is null, remove one of the tries from the scoreboard. If you haven't created it yet, make sure you have a missed variable to store the state of the scoreboard (initialized to 0). When you remove a try from the scoreboard, make sure to increase the missed count by 1.

10.) **Create a checkWin function.**
- Each time the player guesses a letter, this function will check whether the game has been won or lost. At the very end of the keyboard event listener, you’ll run this function to check if the number of letters with class “show” is equal to the number of letters with class “letters”. If they’re equal, show the overlay screen with the “win” class and appropriate text. Otherwise, if the number of misses is equal to or greater than 5, show the overlay screen with the “lose” class and appropriate text.

If you're having trouble with this project, make sure you take a look at this great study guide:
https://drive.google.com/file/d/1tijAzRqxbcUZ8Ylj8onWKmJA4Fnmsf1V/view

EXTRA CREDIT

1.) Create CSS transitions for each letter in the phrase display as they are revealed.

2.) Add a button to the “success” and “failure” screens that reset the game. You’ll have to recreate the buttons in the keyboard, generate a new random phrase, and set the number of misses to zero.