Unit 4 - OOP Game Show App

Create a browser-based, word guessing game: "Phrase Hunter." You'll use JavaScript and OOP (Object-Oriented Programming) to select a random, hidden phrase. A player tries to guess the phrase by selecting individual letters from an onscreen keyboard. Can they guess the phrase before they run out of attempts?

Theory

- Object-Oriented JavaScript Course - https://teamtreehouse.com/library/objectoriented-javascript-2
- Practice Object-Oriented JavaScript - https://teamtreehouse.com/library/practice-objectoriented-javascript
- Focus your mind, improve your learning - https://teamtreehouse.com/library/focus-your-mind-improve-your-learning-3
- Callback Functions in JavaScript Course - https://teamtreehouse.com/library/callback-functions-in-javascript
- JavaScript Array Iteration Methods Course - https://teamtreehouse.com/library/javascript-array-iteration-methods
- Learn more about HTML/CSS - https://teamtreehouse.com/library/learn-more-about-css-html
- Object-Oriented Practice Project - https://teamtreehouse.com/library/objectoriented-practice-project

In this unit, you'll build an interactive game app. This project is both fun and a great way to show off your JavaScript skills -- you'll learn a lot more about JavaScript and working with the DOM, or Document Object Model. You'll also start using object-oriented programming (OOP) principles just like professional JavaScript developers.

Project (16h) - OOP Game Show App

Project zip:

oop_game-v2.zip

oop_game-v2.zip

In this project, you'll create a browser-based, word guessing game: "Phrase Hunter." You'll use JavaScript and OOP (Object-Oriented Programming) to select a random, hidden phrase, which a player tries to guess, by clicking letters on an onscreen keyboard.

Using JavaScript, you'll create two JavaScript classes with specific properties and methods. You'll create a Game class for managing the game, and a Phrase class to help with creating an array of Phrase objects.

Your code will choose a random phrase, split the phrase into letters, and put those letters onto the gameboard.

Each time the player guesses a letter, the program compares the letter the player has chosen with the random phrase. If the letter is in the phrase, the gameboard displays the chosen letters on the screen.

A player continues to select letters until they guess the phrase (and win), or make five incorrect guesses (and lose).

If the player completes the phrase before they run out of guesses, a winning screen appears. If the player guesses incorrectly five times, a losing screen appears.

A player can guess a letter only once. After they've guessed a letter, your programming will need to disable that letter on the onscreen keyboard.

After building this project, you'll have another great portfolio piece to show off your skills, a fun app that you can share with your friends and family, and good understanding of the principles of Object-Oriented Programming.

Before you start

To prepare for this project you'll need to make sure you complete and understand these steps.

GitHub

- Create a new repo for this project.
- Create a `README.md` file for your repo that explains what the project is and anything your user or fellow developers might need to know to use the project.

Download the project files. We've supplied the following files for you to use:

- 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.

Project Instructions

To complete this project, follow the instructions below:

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 good code comments

Check for cross-browser consistency

How you will be graded

	Needs Work	Meets Expectations	Exceeds Expectations
Phrase Class	• No constructor or constructor doesn't properly initialize a `phrase`property • Is missing any of the following methods: `addPhraseToDisplay()`, `checkLetter()`, `showMatchedLetter()`	• Includes constructor that receives a `phrase` parameter and initializes a `phrase` property set to the phrase • Includes `addPhraseToDisplay()`method which adds the phrase to the gameboard • Includes `checkLetter()` method which checks if a letter is in the phrase • Includes `showMatchedLetter()`method which reveals the letter(s) on the board that matches the player's selection	• N/A
Game Class Constructor	• No constructor or constructor doesn't properly initialize the `missed`, `phrases`, or `activePhrase`properties • Phrases added to the game include numbers, punctuation or special characters	• Includes a constructor that initializes a `missed` property set to `0`, a `phrases` property set to an array of five Phrase objects, and an `activePhrase` property set to `null`initially • Phrases added to the game only include letters and spaces	• N/A

Game Class Methods	• Is missing any of the following methods: `startGame()`, `getRandomPhrase()`, `handleInteraction()`, `checkForWin()`,`removeLife()`,gameOver()	• Includes `startGame()` method that hides the start screen overlay, sets the `activePhrase` property to a random phrase, and calls the `addPhraseToDisplay()` method on the active phrase • Includes `getRandomPhrase()`method that randomly retrieves one phrase from the `phrases` array • Includes `handleInteraction()`method that: ◦ Disables the selected letter's onscreen keyboard button ◦ If the phrase does **not** include the guessed letter, the `wrong` CSS class is added to the selected letter's keyboard button and the `removeLife()` method is called ◦ If the phrase includes the guessed letter, the `chosen` CSS class is added to the selected letter's keyboard button, the `showMatchedLetter()` method is called on the phrase, and the `checkForWin()` method is called. If the player has won the game, the `gameOver()` method is called • Includes `checkForWin()` method that checks if the player has revealed all of the letters in the active phrase • Includes a `removeLife()` method that removes a life from the scoreboard (one of the `liveHeart.png` images is replaced with a `lostHeart.png` image), increments the `missed` property, and if the player has lost the game calls the `gameOver()` method • Includes `gameOver()` method that displays a final "win" or "loss" message by showing the original start screen overlay styled with either the `win` or `lose` CSS class	• N/A
app.js	• Clicking the "Start Game" button doesn't create a new `Game` object or doesn't start a new game • Clicking an onscreen keyboard button doesn't result in a call to the `handleInteraction()` method or the method is called for the incorrect keyboard button • Clicking the spaces between and around the onscreen keyboard buttons results in the `handleInteraction()` method being called	• Clicking the "Start Game" button creates a new `Game` object and starts the game • Clicking an onscreen keyboard button results in a call to the `handleInteraction()` method for the clicked keyboard button • Clicking the spaces between and around the onscreen keyboard buttons does not result in the `handleInteraction()` method being called	• Event listener has been added for the `keypress` event so that pressing a physical keyboard button results in the `handleInteraction()`method being called for the associated onscreen keyboard button
Resetting the Gameboard	• After a game is completed, the gameboard isn't reset so that clicking the "Start Game" button fails to load a new game	• After a game is completed, the gameboard is reset so that clicking the "Start Game" button loads a new game	• N/A

HTML and CSS	• Provided HTML or CSS is not used	• Provided HTML and CSS is used	• App styles have been personalized and changes have been noted in the `README.md` file and the project submission notes

Extra Credit

To get an "exceeds" rating, you can expand on the project in the following ways:

Add keyboard functionality

- Let players use their physical computer keyboard to enter guesses. You'll need to use the `keypress` 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.
- Detail your style changes in your `README.md` file **and** in your submission notes