Project 4: OOP Game Show App

Sections of this Guide:

- **How to approach this project** includes detailed guidance to help you think about how to organize your code, project and files.
- **How to succeed at this project** lists the grading requirements for the project, with hints, links to course videos to refresh your memory and helpful resources.

How to Approach This Project

There are a number of ways to approach and complete this project. Below, one approach is detailed in steps, but you do not have to follow this approach to the letter. You are free to explore your own solution, as long as it meets the requirements laid out in the "How to succeed at this project" section below.

Helpful hint:

The two classes used in this project (Game and Phrase) interact with one another. That means when you build this project you'll be moving between the app.js, Game.js and the Phrase.js files instead of building out each file independently. This means you might have to build a little bit more than you're used to before you can test something. In the instructions you'll find notes that indicate good break points to try and test out your code.

Part I

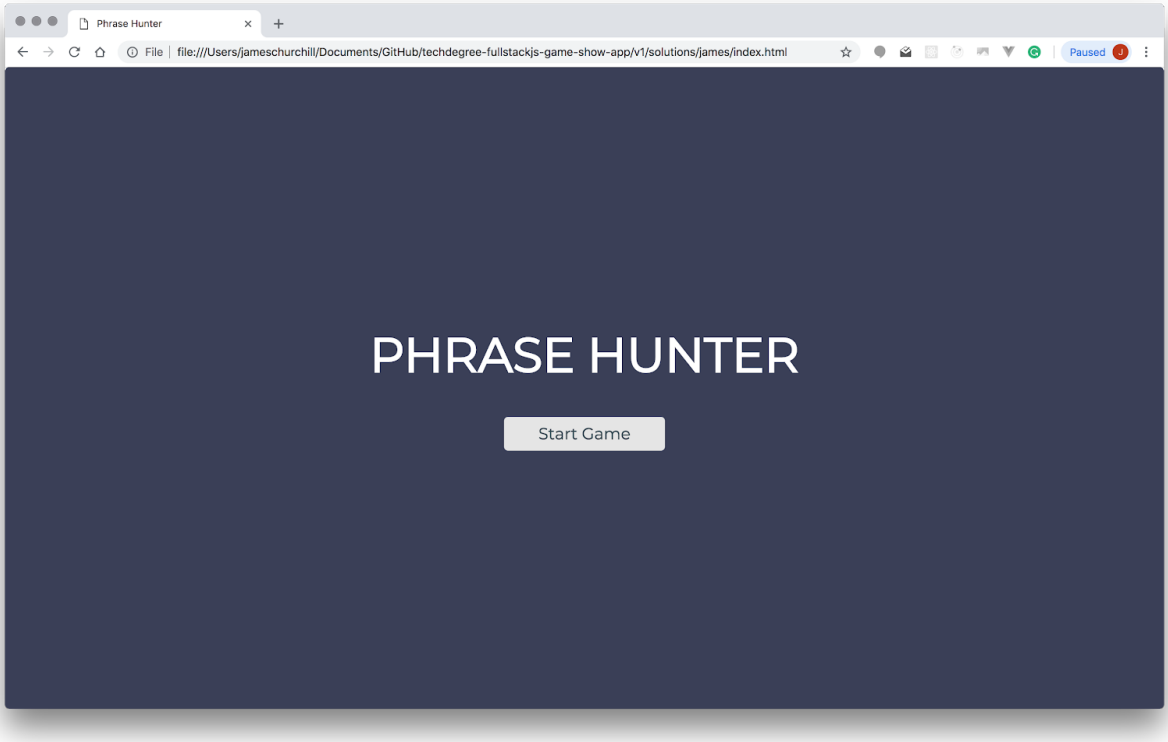
Step 1

Download the project files. In the project files folder, you'll see a folder called 'js' that includes the following 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 methods for starting and ending the game, handling interactions, getting a random phrase, checking for a win, and removing a life from the scoreboard.

Test Your Code!

Feel free to load the provided index.html file into your favorite browser. You haven't added any JavaScript code to the game yet, so you won't be able to interact with anything, but the page should look like this:



Step 2

Declare your classes.

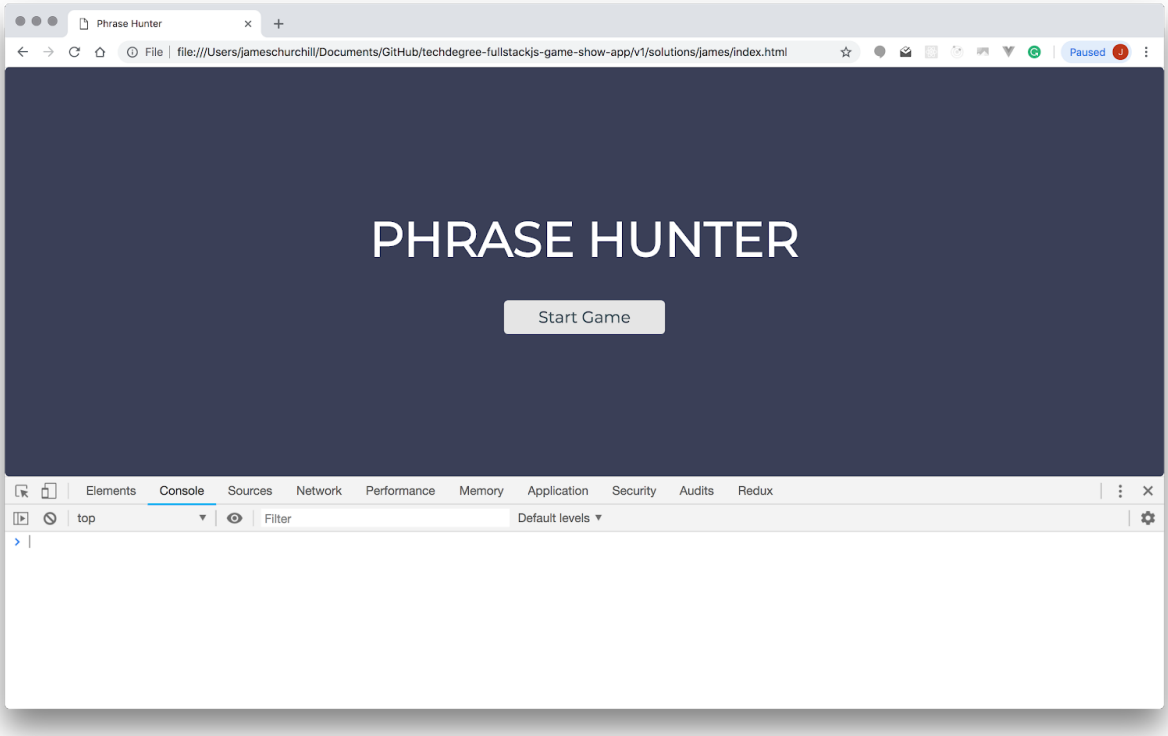
- Inside the Game.js file, declare the Game class.
- Inside the Phrase.js file, declare the Phrase class.

Test Your Code!

Now that you've defined your Phrase and Game classes, you can add temporary code to the app.js file to make sure that you can create instances of each class. Open up the app.js file and add the following code:

```
const phrase = new Phrase();
const game = new Game();
```

You haven't added any properties or methods to your classes yet, so reloading the index.html page in your browser won't produce anything noticeable:



But at least you know that you can successfully create instances of your Phrase and Game classes! Once you've verified that your code is working properly, you can comment out (or remove) the testing code that you just added to the app.js file.

Step 3

Create a constructor method inside each class.

- The Game class constructor method doesn't receive any parameters. The Game class has 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 Phrase objects to use with the game. For now, initialize the property to an empty array. In the next step you'll work on initializing this property with an array of Phrase objects.
    - **activePhrase**: This is the Phrase object that's currently in play. The initial value is `null`.
- The Phrase class constructor method should receive one parameter: `phrase`. The Phrase class has the following property:

- **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.

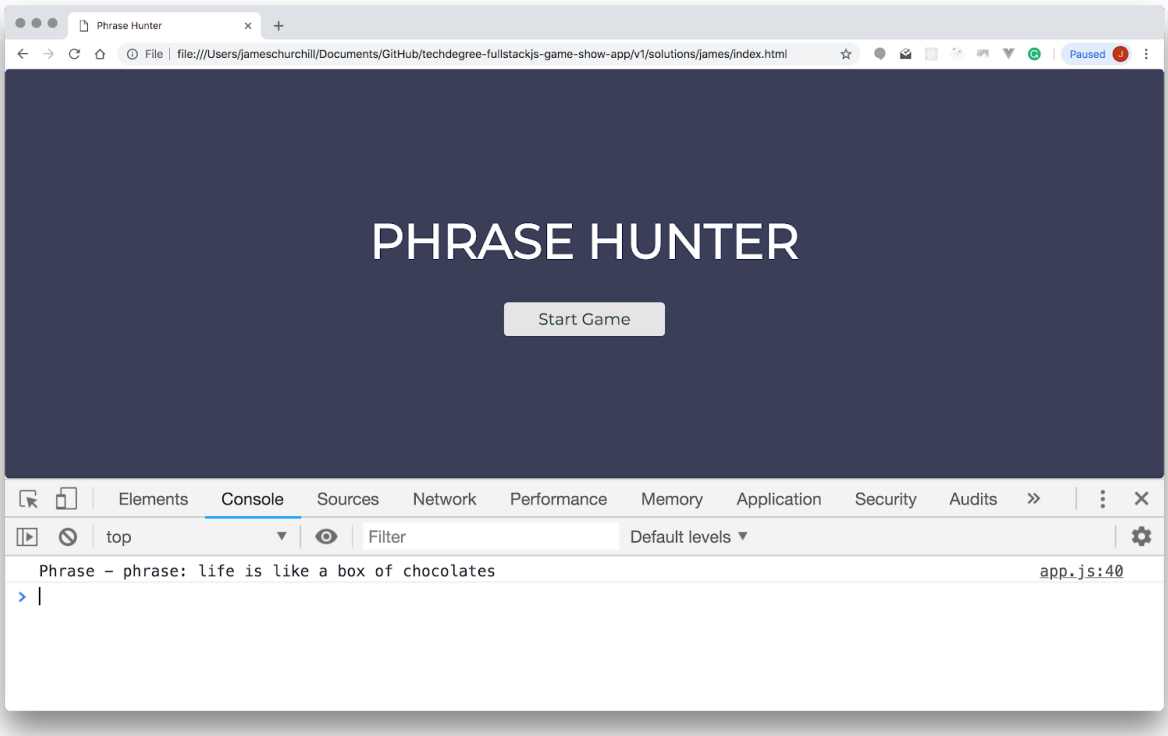
Test Your Code!

Now that you've added a constructor to your Phrase class, you can add temporary code to the app.js file to make sure that everything works as expected. Open up the app.js file and add the following code:

```
const phrase = new Phrase('Life is like a box of chocolates');

console.log(`Phrase - phrase: ${phrase.phrase}`);
```

Load the index.html page into your browser and view the developer tools console. Do you see the following output (notice that the phrase text is displayed in all lowercase letters)?



Once you've verified that your code is working properly, you can comment out (or remove) the testing code that you just added to the app.js file.

Step 4

Time to create your phrases that the game will randomly choose from when showing a new phrase to the player! There are multiple ways to create your phrases and add them to the Game class's `phrases` property:

- Option #1: Inside the Game class, create a method called `createPhrases()`, that creates and returns an array of 5 new Phrase objects, and then set the `phrases` property to call that method.
- Option #2: Simply add 5 new Phrase objects directly in the empty array that was originally set as the value of the `phrases` property.

When creating a Phrase object, don't forget to pass in the actual string phrase that the Phrase object is representing. *A string phrase should only include letters and spaces — no numbers, punctuation or other special characters.*

Here's the documentation for the `createPhrases()` method: (Helpful tip: You can copy the text in these boxes and paste it directly into the appropriate JS file)

```
/**
 * Creates phrases for use in game
 * @return {array} An array of phrases that could be used in the game
 */
createPhrases() {};
```

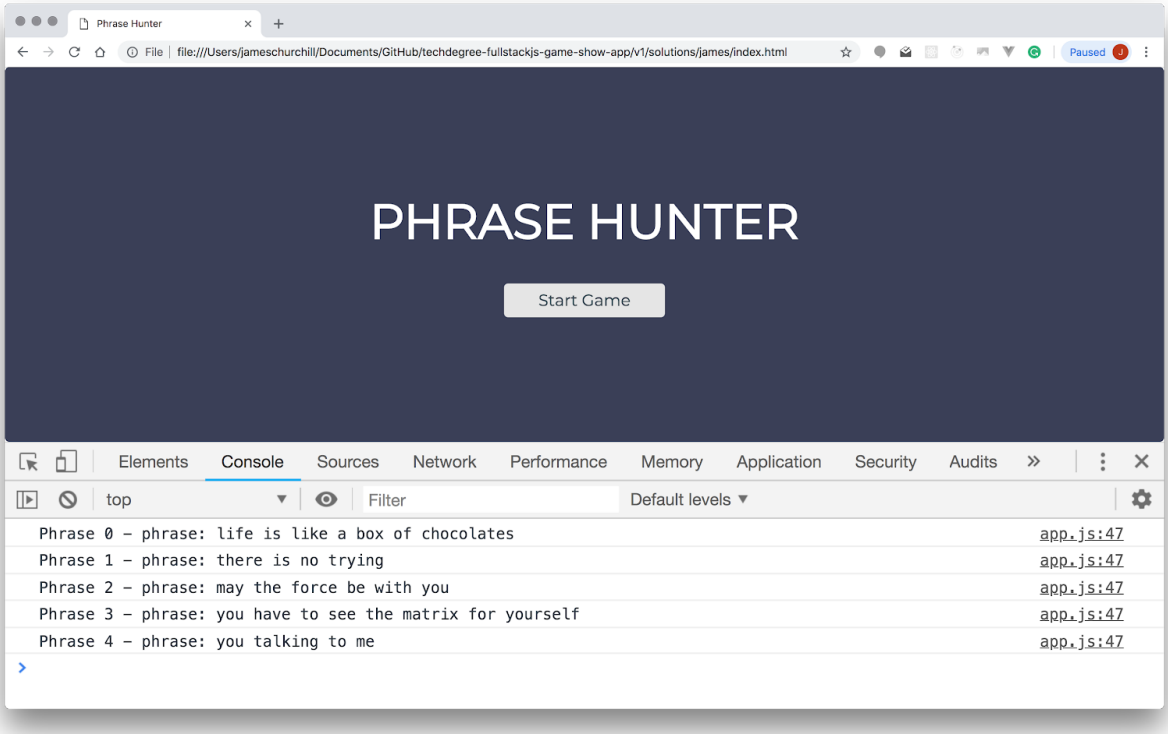
Test Your Code!

Now that you've added code to initialize and populate the Game class `phrases` property, let's add more temporary code to the app.js file to ensure that everything is working properly. Open up the app.js file and add the following code:

```
const game = new Game();

game.phrases.forEach((phrase, index) => {
  console.log(`Phrase ${index} - phrase: ${phrase.phrase}`);
});
```

Load the index.html page into your browser and view the developer tools console. Do you see something similar to the following output (keeping in mind that your phrases are probably different)?



Once you've verified that your code is working properly, you can comment out (or remove) the testing code that you just added to the app.js file.

Step 5

Let's write the `getRandomPhrase()` method mentioned in the last step. This method goes inside the Game class in the Game.js file. This method should select and then return a random phrase from the array of phrases stored in the Game class's `phrases` property.

```
/**
 * Selects random phrase from phrases property
 * @return {Object} Phrase object chosen to be used
 */
getRandomPhrase() {};
```

Test Your Code!

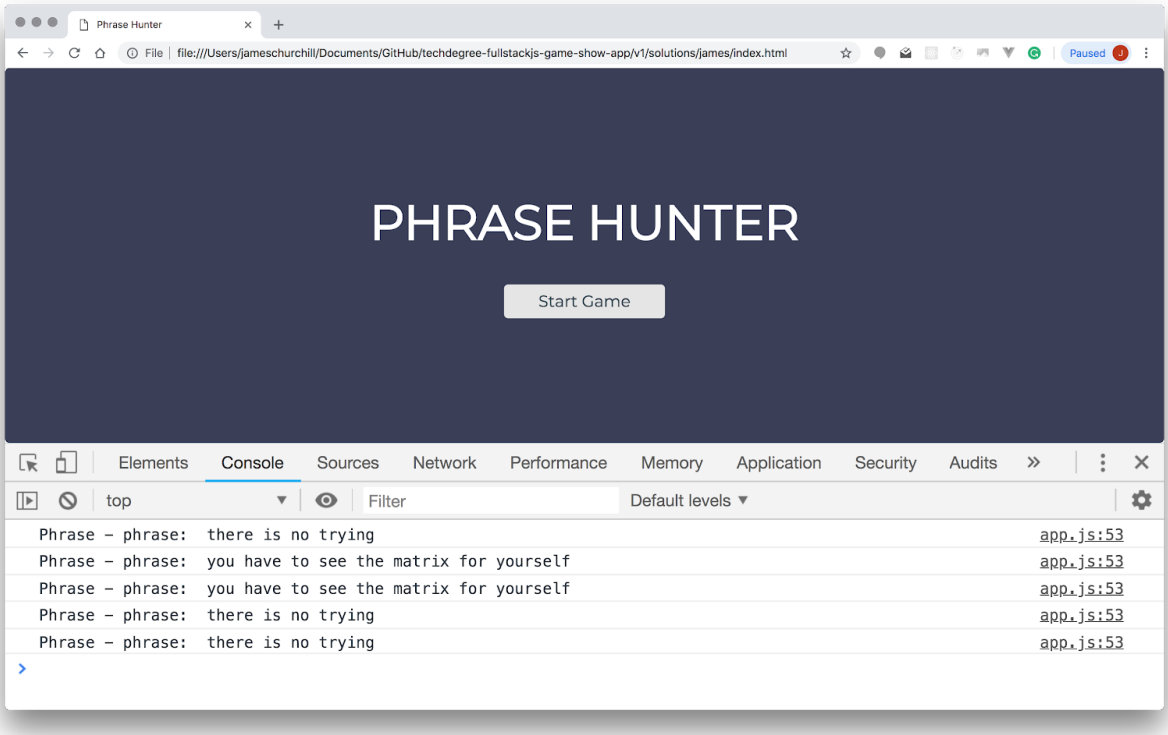
Now that you've added the code for the `getRandomPhrase()` method, let's add more temporary code to the app.js file to ensure that everything is working properly. Open up the app.js file and add the following code:

```
const logPhrase = (phrase) => {
  console.log(`Phrase - phrase: `, phrase.phrase);
};

const game = new Game();

logPhrase(game.getRandomPhrase());
logPhrase(game.getRandomPhrase());
logPhrase(game.getRandomPhrase());
logPhrase(game.getRandomPhrase());
logPhrase(game.getRandomPhrase());
```

Load the index.html page into your browser and view the developer tools console. Do you see something similar to the following output (keeping in mind again that your phrases are probably different)? Notice that instead of seeing a complete list of the phrases array, we're seeing a random phrase printed to the console.



Once you've verified that your code is working properly, you can comment out (or remove) the testing code that you just added to the app.js file.

Step 6

Switch gears for a moment and head to the Phrase class inside the Phrase.js file. Inside the Phrase class, create a method called `addPhraseToDisplay()`.

This method adds letter placeholders to the display when the game starts. Each letter is presented by an empty box, one list item for each letter. See the example_phrase_html.txt file for an example of what the render 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 space.

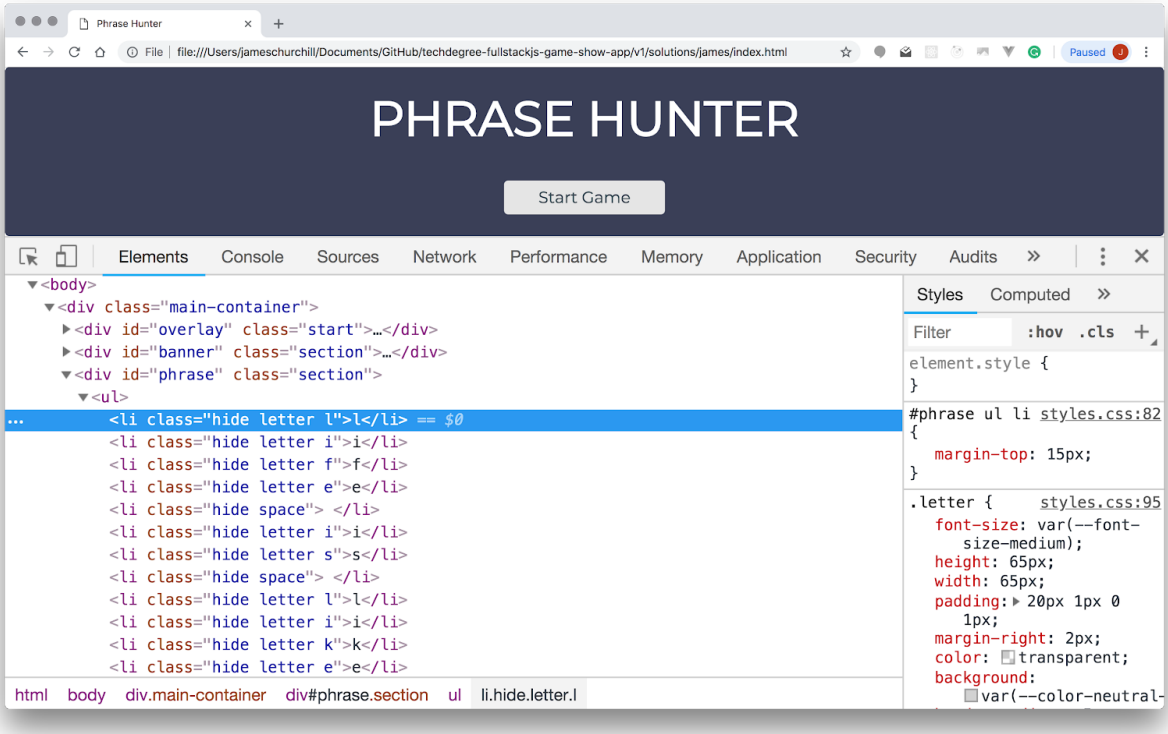
```
/**
 * Display phrase on game board
 */
addPhraseToDisplay() {};
```

Test Your Code!

Now that you've added the code for the `addPhraseToDisplay()` method, let's add more temporary code to the app.js file to ensure that everything is working properly. Open up the app.js file and add the following code:

```
const game = new Game();
game.getRandomPhrase().addPhraseToDisplay();
```

Load the index.html page into your browser and open the developer tools. If you're using Chrome, view the "Elements" tab, or if you're using Firefox, view the "Inspector" tab. Expand the `<body>` element, the `<div class="main-container">` element, and then the `<div id="phrase" class="section">` element, and you should see a list of `<li>` elements, one for each letter in your phrase. Verify that each element's CSS classes and inner content look correct. There should also be an `<li>` element for each space in your phrase, though with slightly different CSS classes.



You'll notice that even though the necessary elements have been added, you can't visually see the phrase on the page. That's because the start screen overlay is obscuring the underlying phrase elements. In just a bit, you'll write the code to hide the start screen overlay when a game is started.

Once you've verified that your code is working properly, you can comment out (or remove) the testing code that you just added to the app.js file.

Step 7

Head back to the Game.js file. Now that we can select a random phrase and display the placeholders for our selected phrase on the screen, let's create a `startGame()` method in the Game class that will put this all in motion for us.

The `startGame()` method hides the start screen overlay (the `div` element with an `id` of `overlay`), calls the `getRandomPhrase()` method to select a Phrase object from the Game object's array of phrases, and then adds the phrase to the gameboard by calling the `addPhraseToDisplay()` method (which is a method on the Phrase class) on the selected Phrase object. The selected phrase should be stored in the Game's `activePhrase` property, so it can be easily accessed throughout the game.

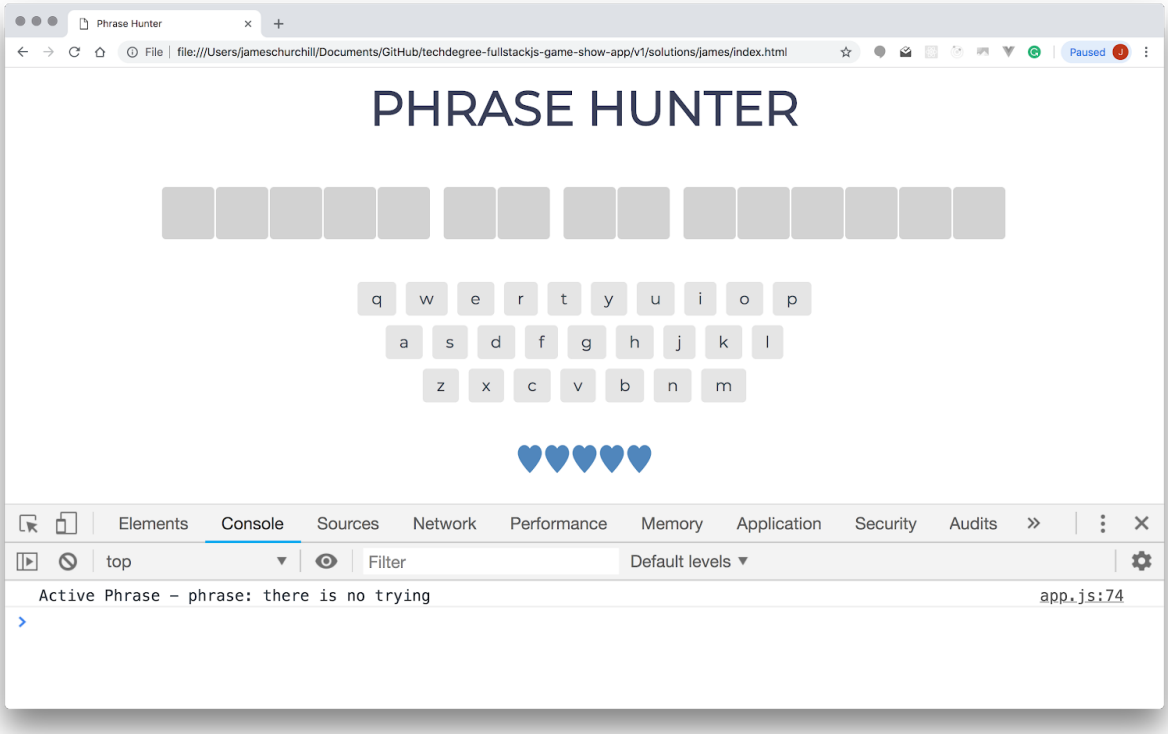
```
/**
 * Begins game by selecting a random phrase and displaying it to user
 */
startGame() {};
```

Test Your Code!

Now that you've added the code for the `startGame()` method, let's add more temporary code to the app.js file to ensure that everything is working properly. Open up the app.js file and add the following code:

```
const game = new Game();
game.startGame();
console.log(`Active Phrase - phrase: ${game.activePhrase.phrase}`);
```

Load the index.html page into your browser and view the developer tools console. Do you see the active phrase printed in the console (keeping in mind again that your phrases are probably different)? Also, has the start screen overlay been hidden so you can now see the elements for your active phrase on the page (presented by gray boxes)?



Once you've verified that your code is working properly, you can comment out (or remove) the testing code that you just added to the app.js file.

Step 8

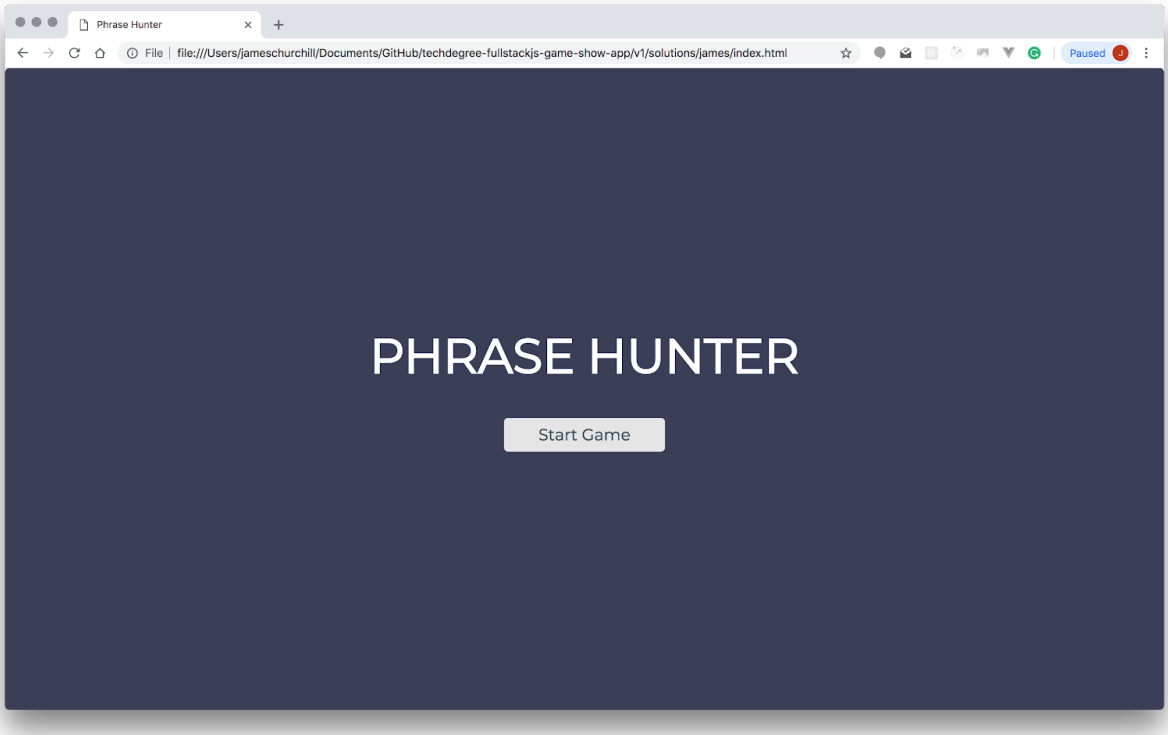
Now that you've built the basics, head over to the app.js file. This is where you'll create an event listener for the "Start Game" button that the user sees when they load your Phrase Hunter game.

- Inside the app.js file, declare a new variable called `game` that's not set to anything.
- Then, add a "click" event listener to the HTML `<button>` element with an `id` of `btn__reset`. Inside the callback function for this click event listener, use your `game` variable to instantiate a new Game object. Call the `startGame()` method on this new Game object.

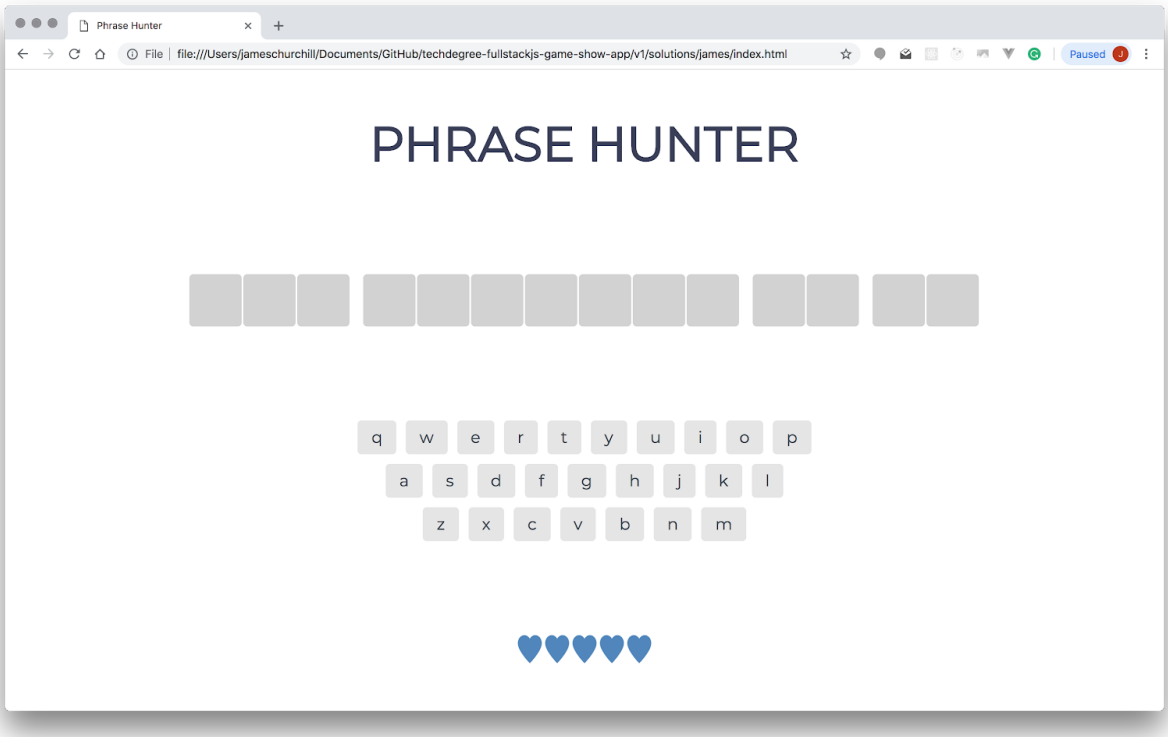
Test Your Code!

Now that you have the "click" event listener wired up to the "Start Game" button, you can simply load the index.html page into your browser and use your app's UI (user interface) to test your code.

Here's what the app should initially look like (before the "Start Game" is clicked):



And then after the "Start Game" button has been clicked:



Great work so far! You've made a lot of progress towards completing this project.

Part II

Step 9

Now it's time to start adding some user interaction to the game. When a user clicks on one of the onscreen keyboard buttons, several things need to happen:

- The clicked/chosen letter must be captured.
- The clicked letter must be checked against the phrase for a match.
- If there's a match, the letter must be displayed on screen instead of the placeholder.
- If there's no match, the game must remove a life from the scoreboard.
- The game should check if the player has won the game by revealing all of the letters in the phrase or if the game is lost because the player is out of lives.
- If the game is won or lost, a message should be displayed on screen.

The logic and branching to handle all of those steps can be included in one method in the Game class called `handleInteraction()`. Several supporting methods to handle the above individual

actions should be written inside both the Game and Phrase classes to keep the `handleInteraction()` method neat:

- Phrase class methods
    - `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.
- Game class methods
    - `checkForWin()`: This method checks to see if the player has revealed all of the letters in the active phrase.
    - `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.
    - `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.

Phrase class methods:

```
/**
 * Checks if passed letter is in phrase
 * @param (string) letter - Letter to check
 */
checkLetter(letter) {};
```

```
/**
 * Displays passed letter on screen after a match is found
 * @param (string) letter - Letter to display
 */
showMatchedLetter(letter) {};
```

Game class methods:

```
/**
 * Checks for winning move
 * @return {boolean} True if game has been won, false if game wasn't
won
 */
checkForWin() {};
```

```
/**
 * Increases the value of the missed property
 * Removes a life from the scoreboard
 * Checks if player has remaining lives and ends game if player is out
 */
removeLife() {};
```

```
/**
 * Displays game over message
 * @param {boolean} gameWon - Whether or not the user won the game
 */
gameOver(gameWon) {};
```

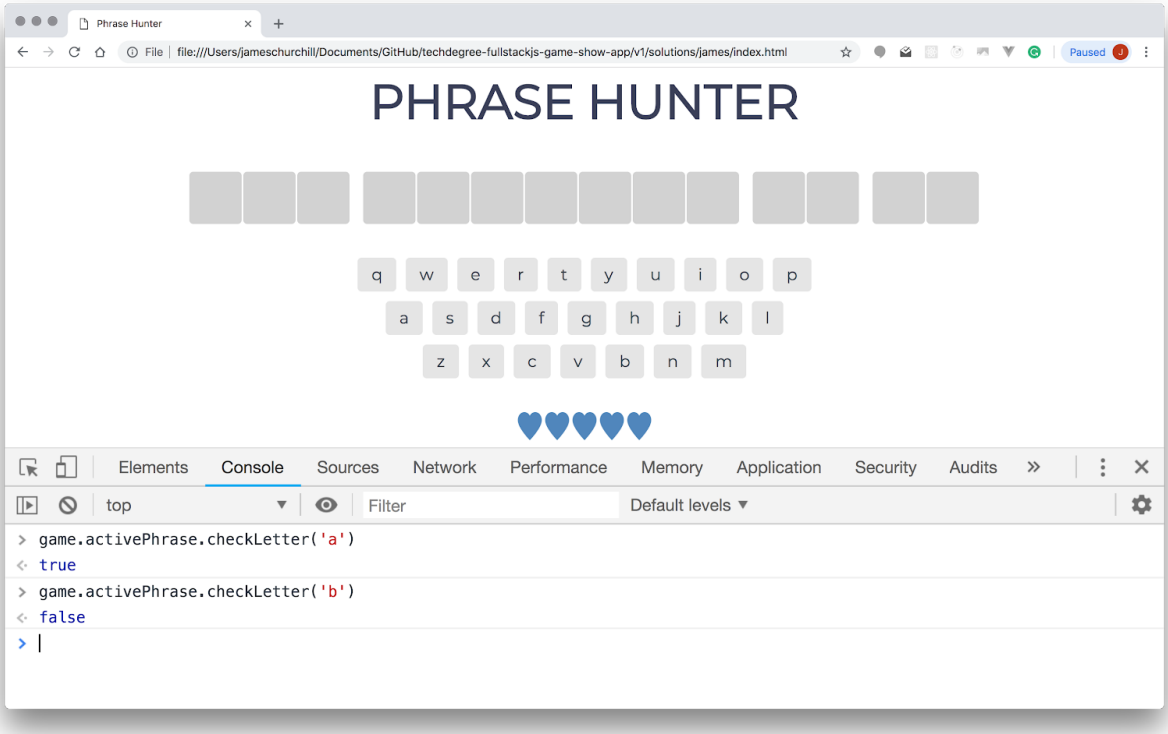
Test Your Code!

Now that you've added the code for the above supporting methods, load the index.html page into your browser, click the "Start Game" button to start a game, and view the developer tools console.

Let's start with testing the new Phrase class methods. To test the `checkLetter()` method, enter the following line of code directly into the console:

```
game.activePhrase.checkLetter('a')
```

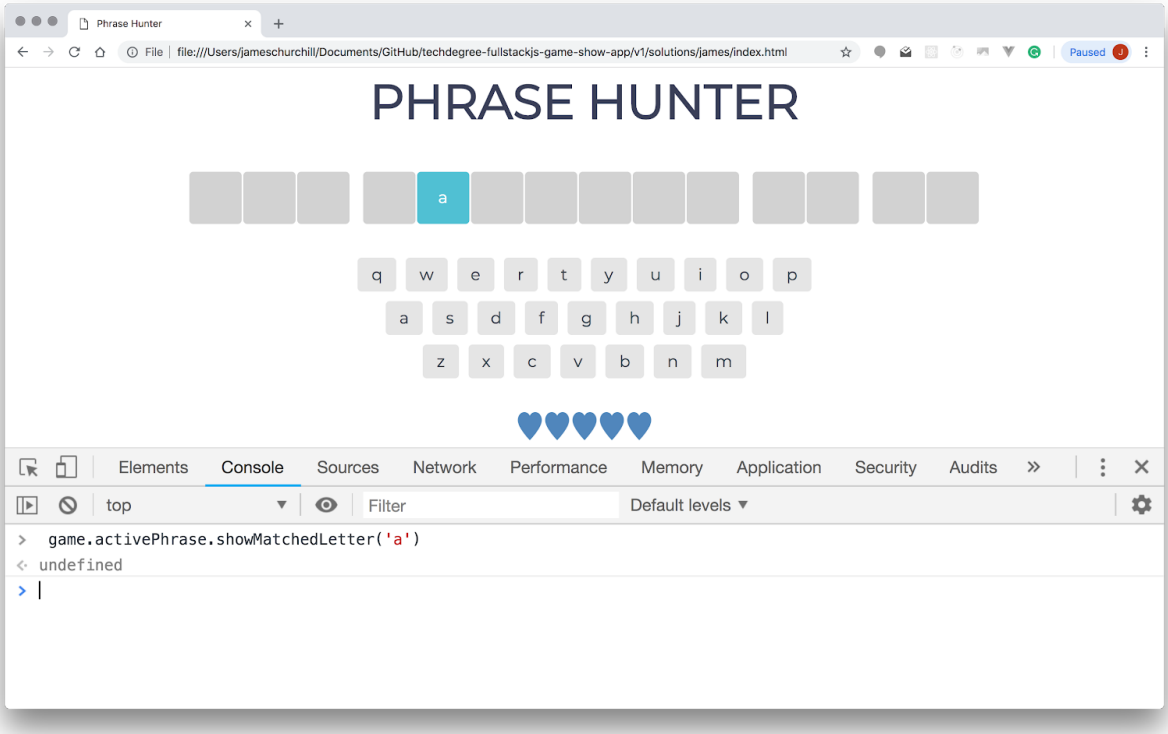
If the letter "a" is in your active phrase, you should see the value `true` printed to the console, otherwise you should see the value `false` printed. Here's what my console looked like when I called the `checkLetter()` method for both the letters "a" and "b":



To test the Phrase class's `showMatchedLetter()` method, enter the following line of code directly into the console, passing in a letter that you know is your active phrase:

```
game.activePhrase.showMatchedLetter('a')
```

After calling the `showMatchedLetter()` method, you should see the passed letter shown in the phrase. Here's what my page looked like after I called the `showMatchedLetter()` method for the letter "a":

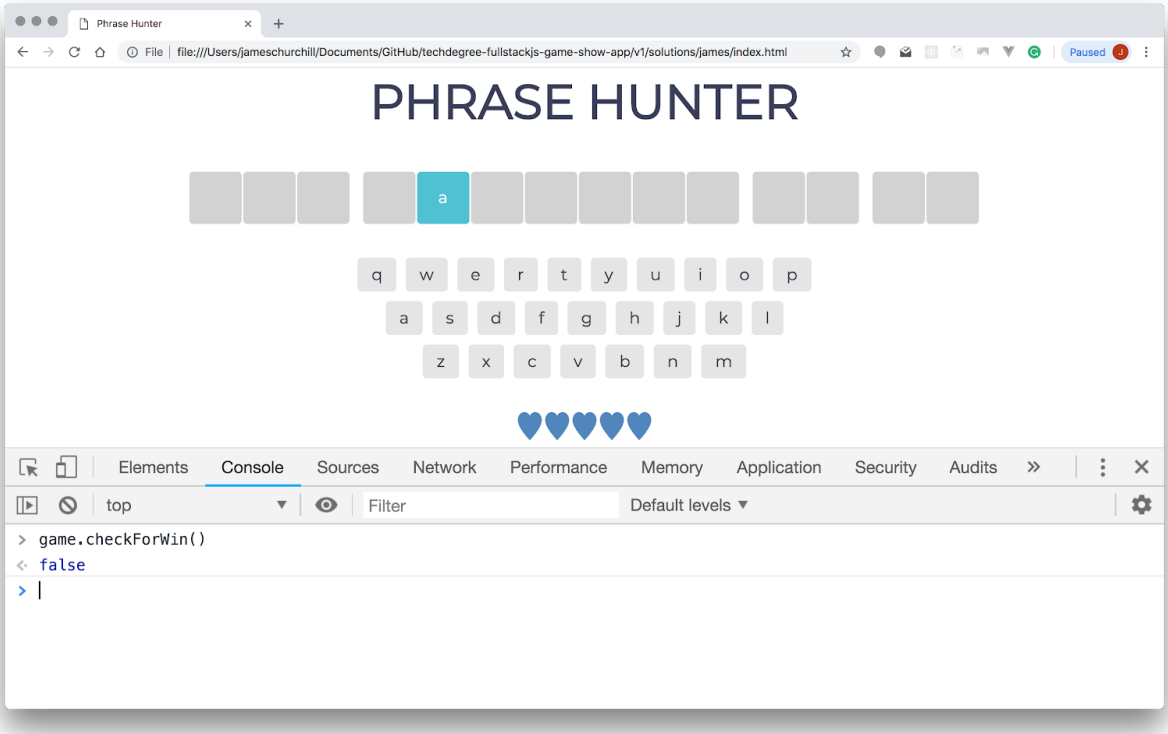


Note: The value `undefined` is printed to the console because the `showMatchedLetter()` method doesn't return a value.

Now let's test the new Game class methods. To test the `checkForWin()` method, enter the following line of code directly into the console:

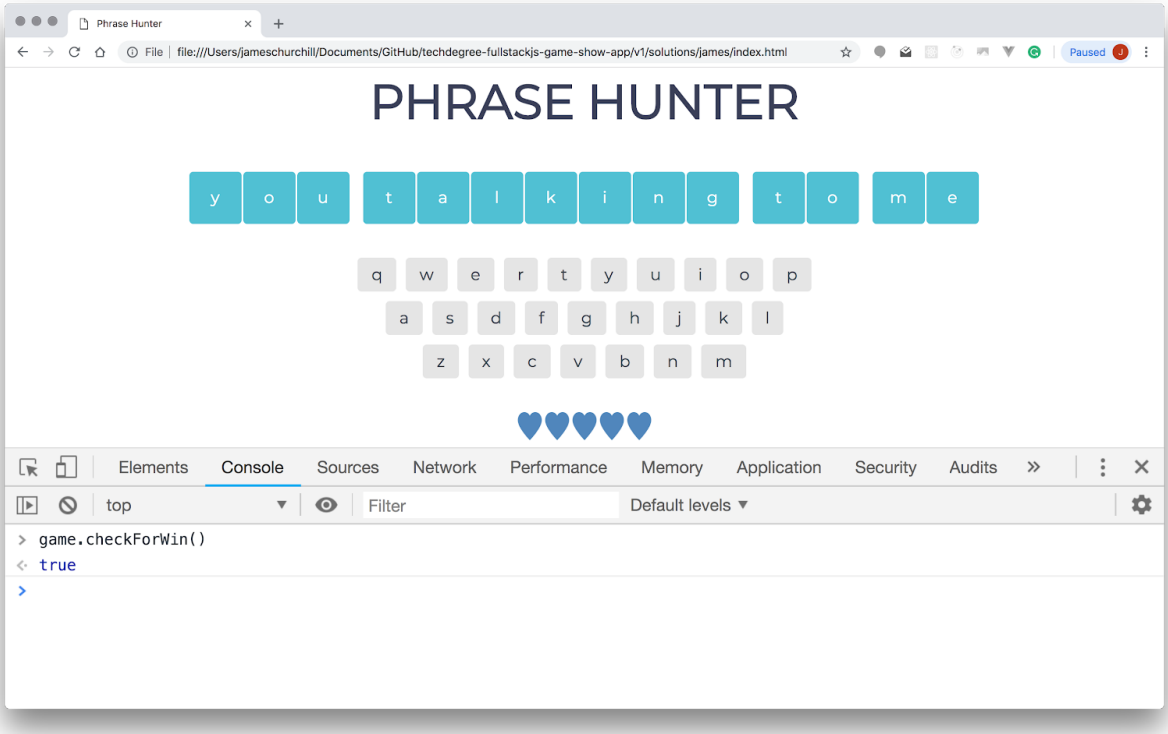
```
game.checkForWin()
```

Assuming that you haven't revealed all of the letters in the active phrase (if you have, you can reload the page to start a new game), you should see the following output in the console:



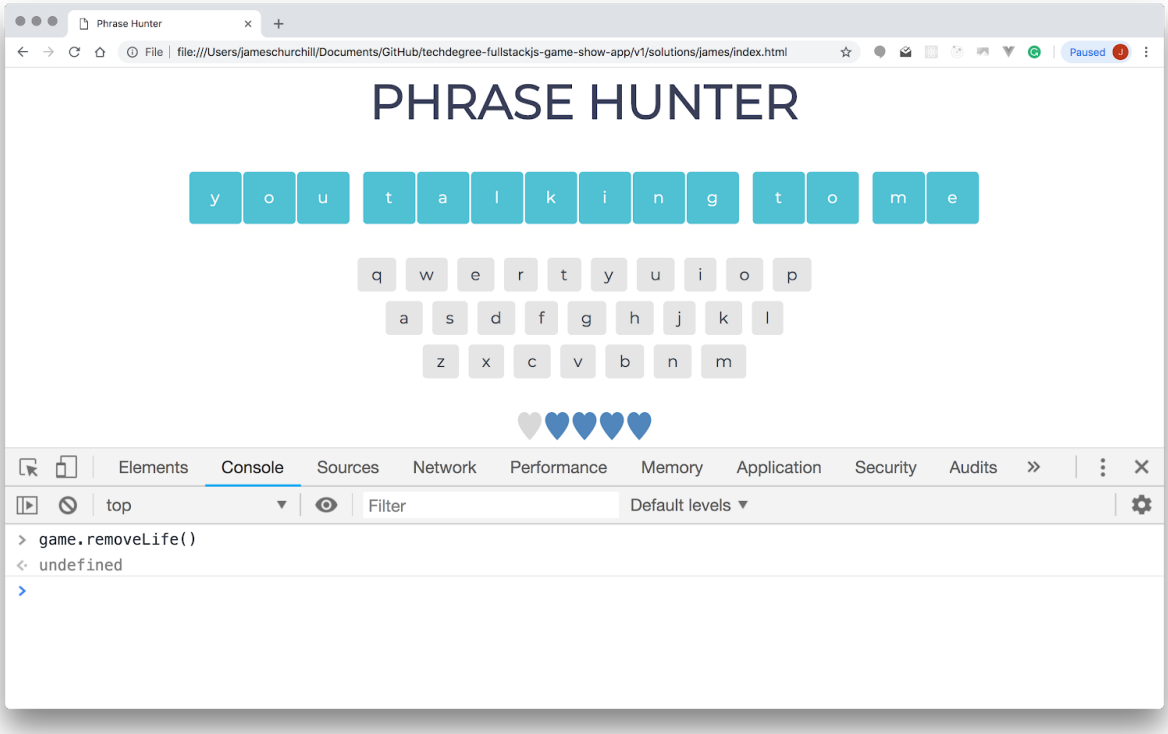
I haven't revealed all of the letters in the active phrase, so the `checkForWin()` method returned the value `false`, which is what we'd expect. Next, I called the `showMatchedLetter()` method once for each unique letter that was in the active phrase, which revealed all of the letters. Then I called the `checkForWin()` method again:

Tip: In the console, you can press the UP ARROW key to retrieve the prior line of code that was executed. Then you can use the LEFT and RIGHT ARROW keys to edit the code. This will keep you from having to type each line of code one-by-one!

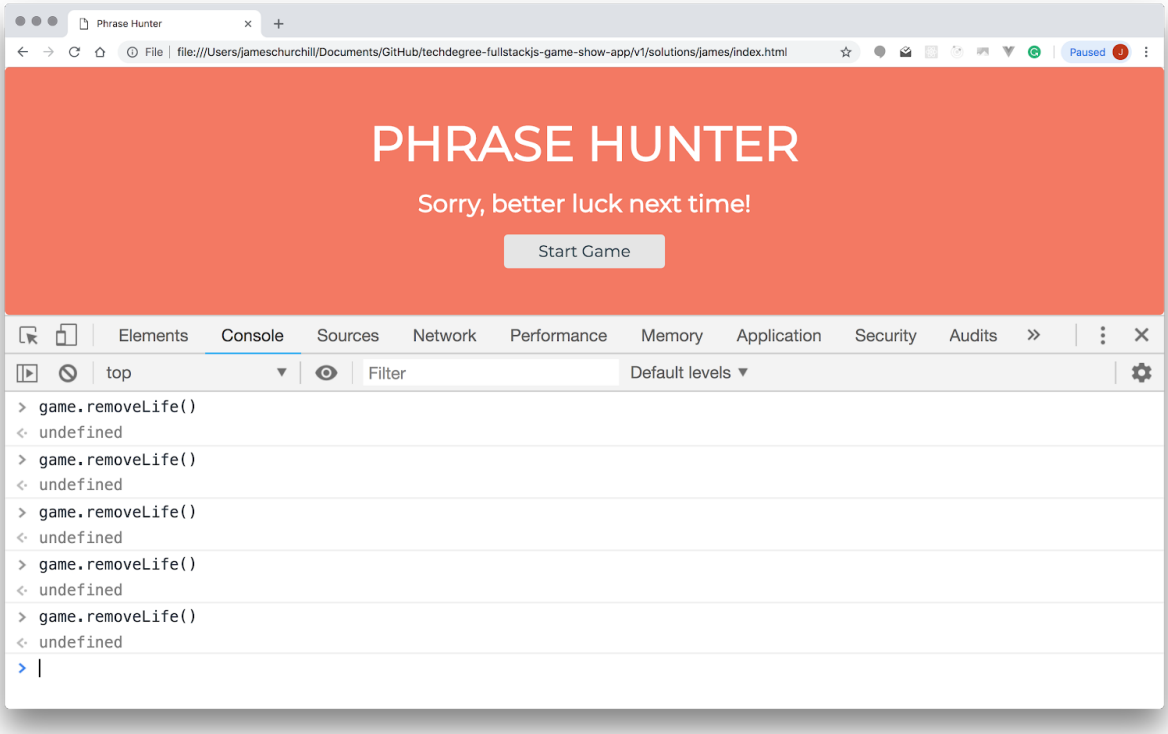


With all of the letters revealed in the active phrase, the `checkForWin()` method returned the value `true`.

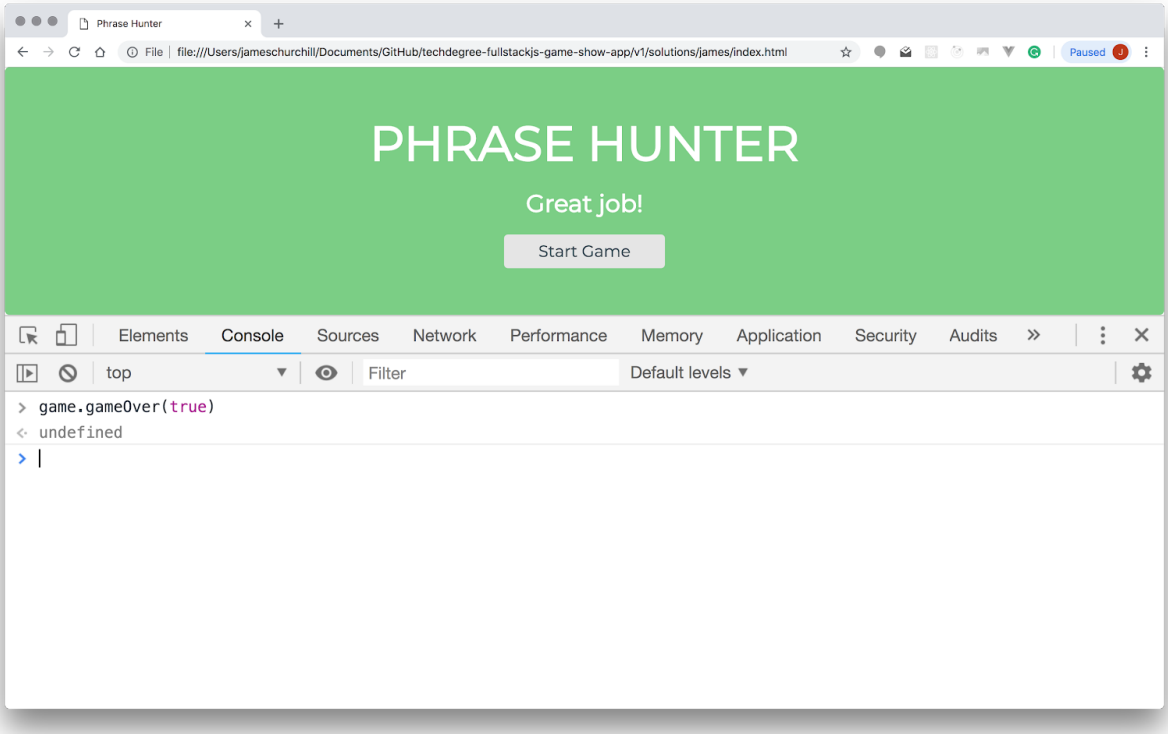
To test the `removeLife()` method, I simply called the method in the console to test that it properly updated a heart image in the scoreboard (indicating that a life was "lost"):



Then I called the `removeLife()` method four more times to test that the game would end and display the "lost" message:



And finally, I called the `gameOver()` method passing `true` to test that the game would end and display the "won" message:



That was a lot of testing! Testing each individual method takes extra time, but now we can proceed with implementing the `handleInteraction()` method with confidence knowing that all of our supporting methods are working as expected.

Step 10

It's finally time update your app so you can use the UI to guess a letter in the active phrase! In the app.js file, add a "click" event listener to each of the onscreen keyboard buttons or use event delegation and add a single event listenerthat listens for a click on any of the onscreen keyboard buttons. If you use event delegation, make sure that clicking the space between and around the onscreen keyboard buttons does not result in the `handleInteraction()` method being called.

In the callback function of your event listener(s), call the `handleInteraction()` method on the Game object. You'll find it helpful to add a (mostly) empty `handleInteraction()` method to the Game class before doing this step, with the intent to build it out in the next step.

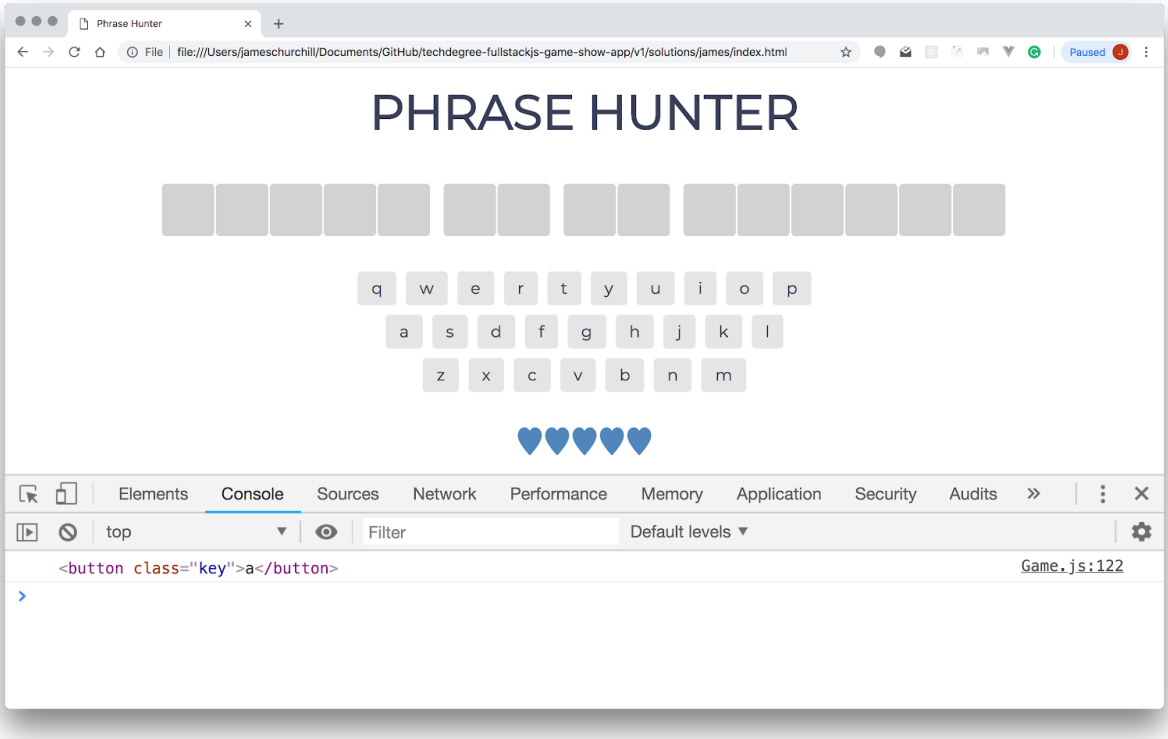
To aid your testing efforts, temporarily add a call to the `console.log()` method, passing in the `handleInteraction()` method's `button` parameter (i.e. the onscreen keyboard button that was clicked):

```
/**
 * Handles onscreen keyboard button clicks
 * @param (HTMLButtonElement) button - The clicked button element
 */
handleInteraction(button) {
  console.log(button);
};
```

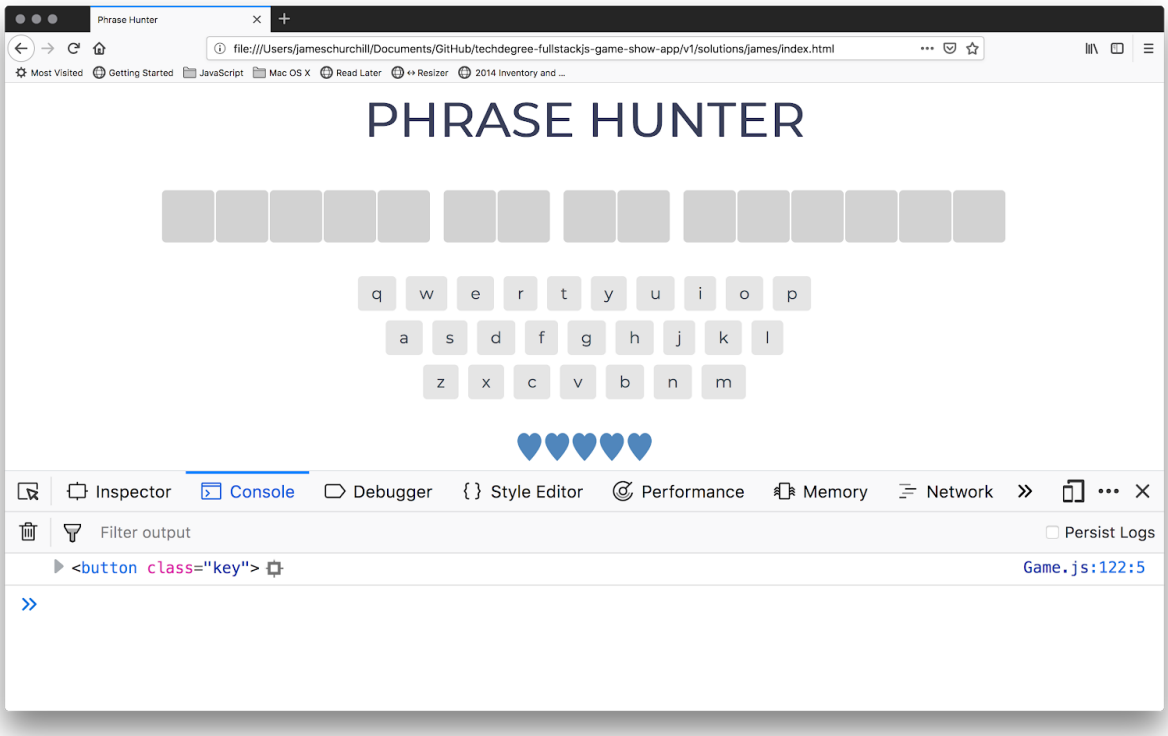
Test Your Code!

Now that you've added the code for your onscreen keyboard button event listener(s) and stubbed out the Game class's `handleInteraction() method, let's test your changes! Load the index.html page into your browser, click the "Start Game" button to start a game, and view the developer tools console.

Then click a button on the onscreen keyboard. Do you see the following output after clicking the letter "a" in the onscreen keyboard?



If you're using Firefox (instead of Chrome), it'll look something like this:



If you used event delegation, also test that clicking the space between and around the onscreen keyboard buttons does not result in the `handleInteraction()` method being called (i.e. you shouldn't see the button element printed to the console).

Step 11

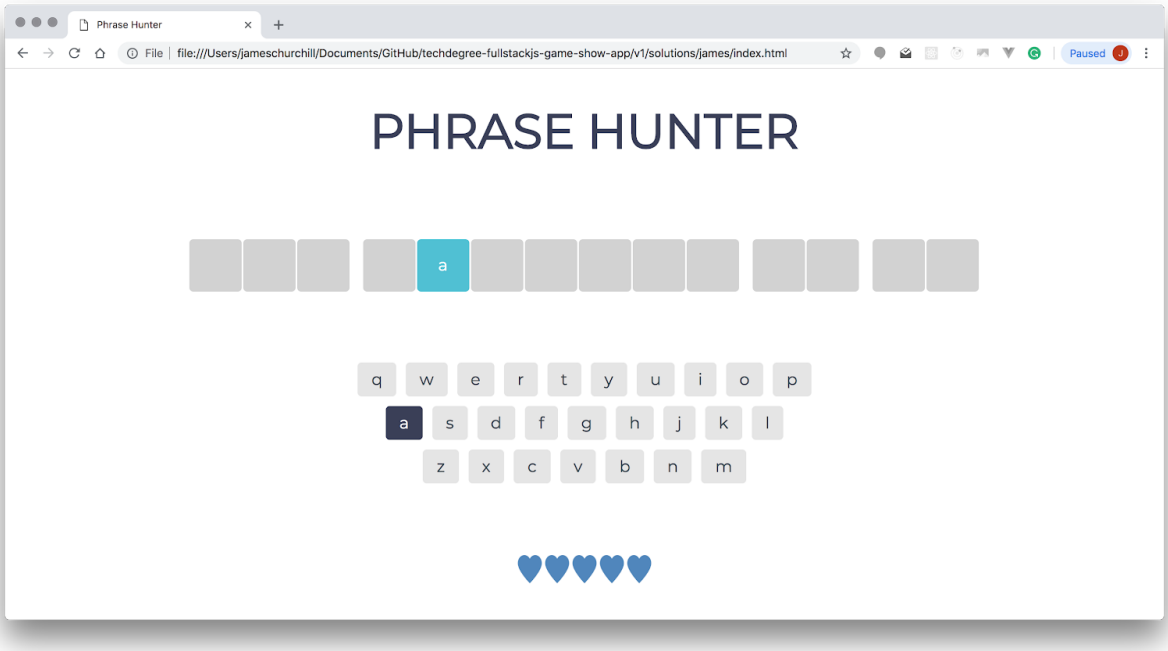
Build out the `handleInteraction()` method in the Game class making use of the support methods that you created in step 9. This method controls most of the game logic. It checks to see if the onscreen keyboard button clicked by the player matches a letter in the phrase, and then directs the game based 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.

Test Your Code!

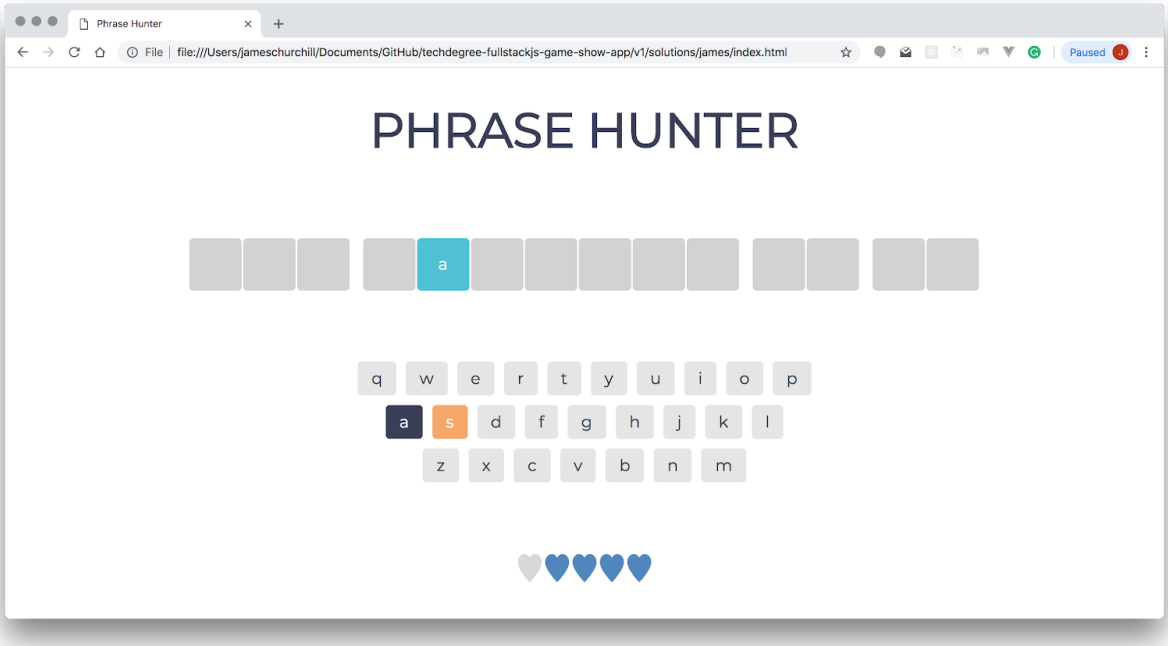
At this point in the development of your app, you can continue to use the UI to test your changes. Load the index.html page into your browser and click the "Start Game" button to start a game.

Let's start with the "happy" path by clicking an onscreen keyboard button for a letter that you know is contained within your active phrase (my active phrase was "you talking to me" so I clicked the letter "a"):



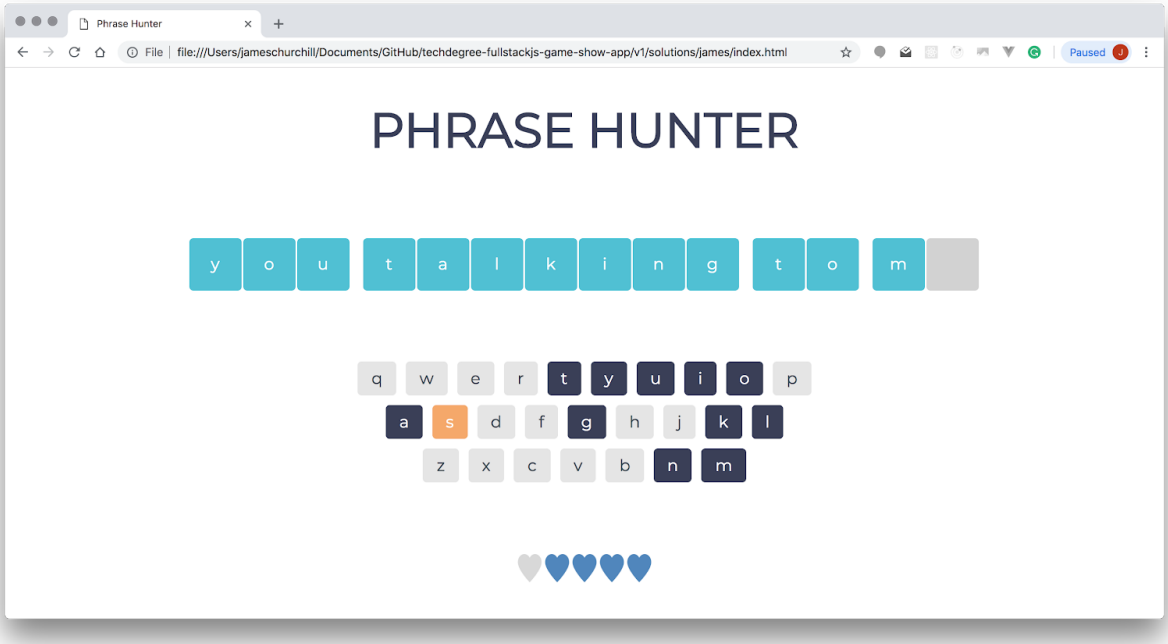
After clicking a correct letter in the onscreen keyboard, were all instances of the letter revealed in the phrase? Did the letter button change to a dark gray color? Was the letter button disabled, preventing you from clicking a second time?

Now click an onscreen keyboard button for a letter that you know isn't contained within your active phrase (I clicked the letter "s"):

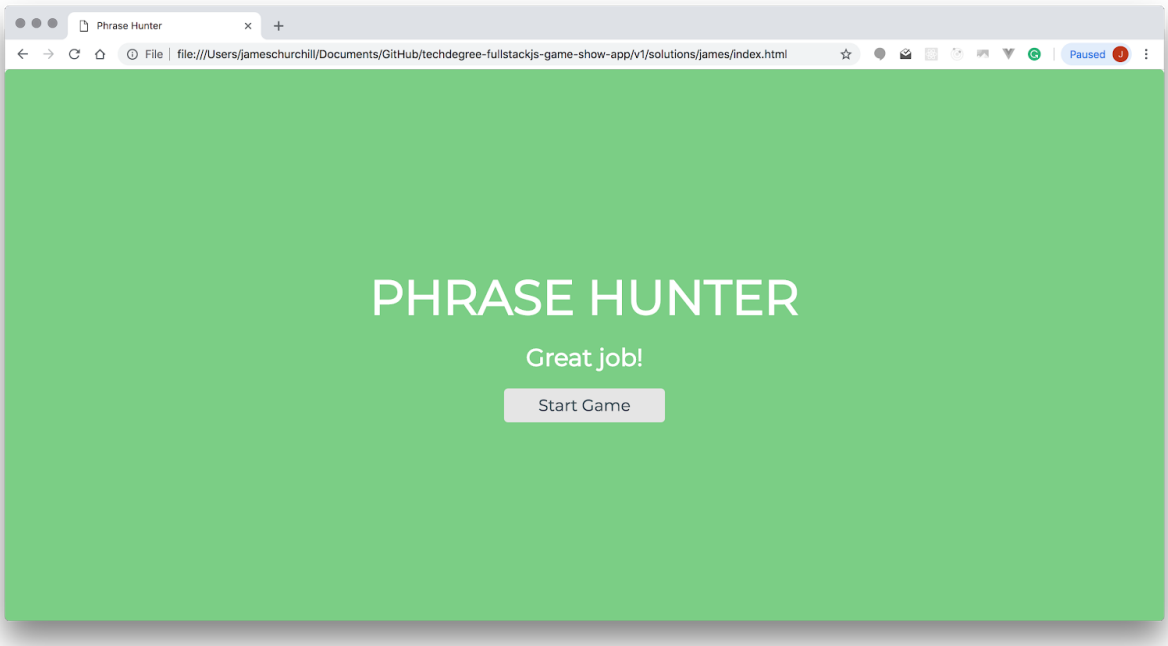


After clicking an incorrect letter in the onscreen keyboard, did the letter button change to an orange color? Was the letter button disabled, preventing you from clicking a second time? Was a life removed from the scoreboard?

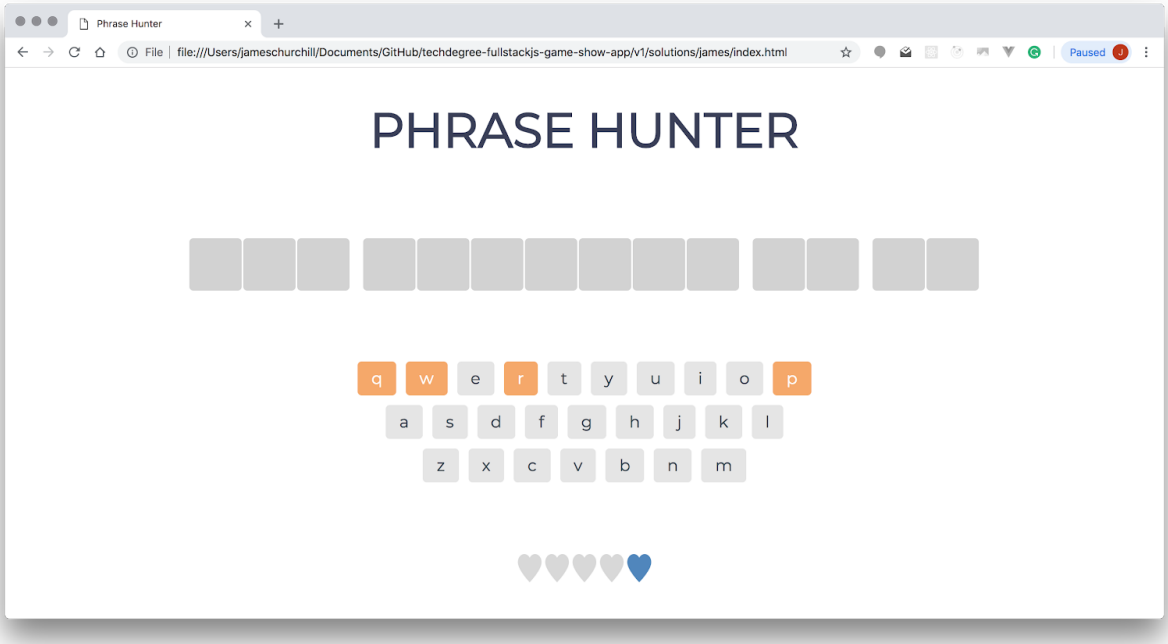
Now let's test to see if you can win the game, by guessing all of the letters in your active phrase. Here's my game just before I guessed the last letter in my phrase:



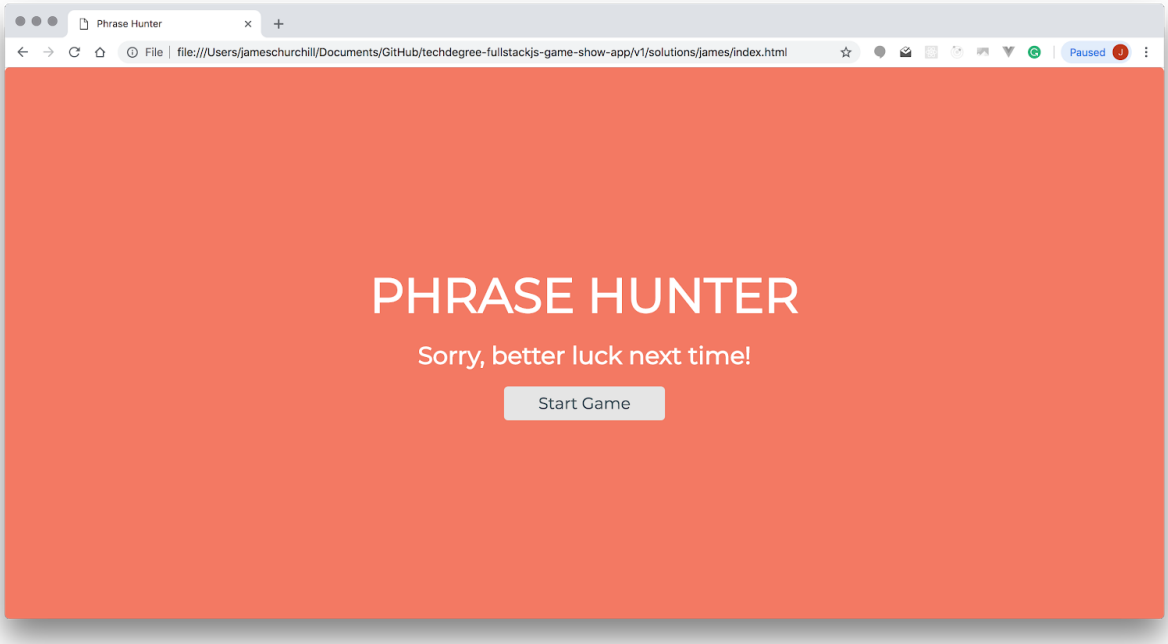
And here's my game after I clicked the letter "e":



And lastly, reload the page, so that you can test if you can lose the game, by incorrectly guessing five letters. Here's my game after four incorrect guesses:



And here's my game after I made another incorrect guess:



Way to go! You're almost done with developing your app. Just one more step to go.

Step 12

Update your app to reset 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.

Test Your Code!

After you've completed all of the necessary changes, just complete a game (either by guessing all of the letters in your active phrase or by incorrectly guessing five times) and then click the "Start Game" button on the win/loss screen overlay to check if you can successfully start a new game.

- Was the new phrase added to the gameboard correctly (i.e. all of the letters from the previous phrase were removed before the new phrase's letters were added?)
- Was the onscreen keyboard reset so that each letter button appears with a light gray color and is enabled (so that it can be clicked)?
- Was the scoreboard reset back to five lives?

Awesome job on completing your game!

How to succeed at this project

Here are the things you need to do pass this project. Make sure you complete them ***before*** you turn in your project.

Phrase Class

- ❏ **Includes constructor that receives a `phrase` parameter and initializes a `phrase` property set to the phrase**
  - ❏ *Related video:* **Writing Your First Class**
  - ❏ *Related video:* **Practice Classes in JavaScript**
  - ❏ *Related video:* **Adding Properties Inside the Constructor Method**
- ❏ **Includes `addPhraseToDisplay()` method which displays the phrase on 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**

- ❏ *Related video:* Adding Methods to our Class
- ❏ *Related text:* Adding Methods Solution
- ❏ *Related video:* Practice JavaScript Loops
- ❏ *Related video:* Practice If and Else Statements in JavaScript
- ❏ *Related video:* Changing Element Attributes
- ❏ *Related video:* Practice Selecting DOM Elements
- ❏ *Related video:* Practice Traversing the DOM
- ❏ *Related video:* Practice Manipulating the DOM

Game Class Constructor

- ❏ 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
  - ❏ *Related video:* Instantiating an Object
  - ❏ *Related video:* Practice Basic Arrays in JavaScript

Game Class Methods

- ❏ 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:
  - ❏ 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
  - ❏ *Related video:* Create a Random Number

app.js

- ❏ 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
    - ❏ *Related video:* Listening for Events with addEventListner()
    - ❏ *Related video:* The Event Object - Event Delegation
    - ❏ *Related video:* Practice Basic JavaScript Functions

Resetting the Gameboard

- ❏ After a game is completed, the gameboard is reset so that clicking the "Start Game" button loads a new game

HTML and CSS

- ❏ Provided HTML and CSS is used
    - ❏ *Related video:* Styling Elements (DOM API)