Project 4: OOP Game Show App Study Guide

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 play.php, Game.php and the Phrase.php 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 breakpoints to try and test out your code.

Part I

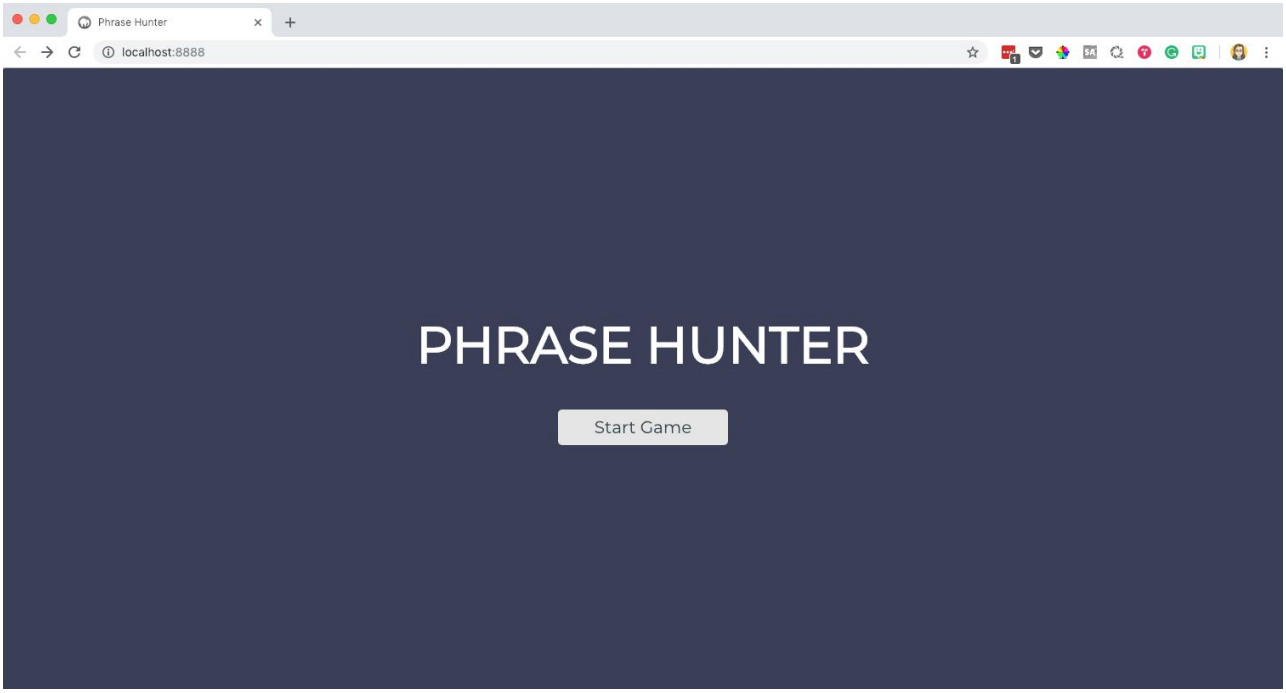
Step 1

Download the project files. In the project files folder, you'll see a folder called 'js' that includes the following files:

- HTML, CSS and images for your game
- example_html folder with HTML for keyboard, phrase and score
- play.php to store game progress, instantiate your objects and make your method calls
- inc/Phrase.php to create a Phrase class to manage individual phrases.
- inc/Game.php to create a Game class to manage the functioning of the game: it has methods for showing the game, handling interactions, and checking for game over.

Test Your Code!

Setup MAMP to point to this project directory. You haven't added any PHP code to the game yet, so you won't be able to interact with anything, but you should be able to go to your localhost (e.g. http://localhost:8888/) and see a page like this:



You should also be able to click the start page button and be directed to play.php which should show a blank page.

Step 2

Declare your classes.

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

Instantiate your classes.

- Inside the play.php file, include your 2 class files
- Create instances of each class

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

Test Your Code!

Now that you've defined your Phrase and Game classes and instantiated objects, you can add temporary code to the play.php file to make sure that you are actually creating instances of each class. Open up the play.php file and add the following code:

```
var_dump($phrase);
var_dump($game);
```

Navigate to play.php (http://localhost:8888/play.php) and you should see the following objects.

```
object(Phrase)#1 (0) { } object(Game)#2 (0) { }
```

Now you know that you can successfully create instances of your Phrase and Game classes! We'll be using these var_dumps to test the next step, so keep them there for now.

Step 2

Create the properties and constructors for your classes

- Inside the Phrase class
  - Add the following **private** properties:
    - **$currentPhrase**: An string containing the current phrase to be used in the game.
    - **$selected**: An array of letters the user has guessed. Initialize the property to an empty array.
  - Add the constructor method which should receive two OPTIONAL parameters: the current phrase as a string, and the selected letters as an array.
    - If the parameters are **not empty**, assign those values to the class properties
    - If the **phrase** parameters is not set, it will be set to a random phrase. For now, set the **$currentPhrase** property to use a string "dream big".
- Inside the Game class
  - Add the following **private** properties:
    - **$phrase**: a Phrase object, which will be set by the constructor.
    - **$lives**: Used to set how many wrong guesses a player has before Game Over. This should be set to 5.
  - Add the constructor method which should receive one REQUIRED parameters, a Phrase object. Use this object to set the $phrase property.
- Inside the play.php file, pass the $phrase object when instantiating the Game object.

Test Your Code!

Refresh your play.php page and you should see the properties of your objects now:

```
object(Phrase)#1 (3) {
  ["currentPhrase":"Phrase":private]=>
  NULL
  ["selected":"Phrase":private]=>
  array(0) {
  }
  ["phrase"]=>
  string(9) "dream big"
}
object(Game)#2 (2) {
  ["phrase":"Game":private]=>
  object(Phrase)#1 (3) {
    ["currentPhrase":"Phrase":private]=>
    NULL
    ["selected":"Phrase":private]=>
    array(0) {
    }
    ["phrase"]=>
    string(9) "dream big"
  }
  ["lives":"Game":private]=>
  int(5)
}
```

You should see TWO Phrase objects, because your Game object should also include the Phrase object. You can test both passing a phrase string and NOT passing a string into your Phrase object. You can also test both passing an array of selected letters and not an array into your Phrase object. Once you've verified that your code is working properly, you can comment out (or remove) the var_dumps that you used for testing in the play.php file.

Step 4

Let's start generating our game board. Head to the Phrase class inside Phrase.php. 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 character. 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. This method should **return** a string containing the HTML to display the Phrase boxes.

You'll need to loop through each character in the Phrase. To work with the individual characters of a string, you'll want to split (str_split) the characters into an array. You'll also want to make sure all letters are lowercase (strtolower).

```
//Split string into lowercase characters
$characters = str_split(strtolower($this->currentPhrase));
```

Make sure a space character has a class="space", while a letter character has class="hide letter". For now, all other characters should have the class="hide". *We'll worry about showing letters later.*

Test Your Code!

To make sure the code will display properly, add the following code to your play.php file:

```
echo $phrase->addPhraseToDisplay();
```

Refresh your play.php page. Because we haven't yet included our CSS, boxes should look like this:

```
    • d
    • r
    • e
    • a
    • m
    •
    • b
    • i
    • g
```

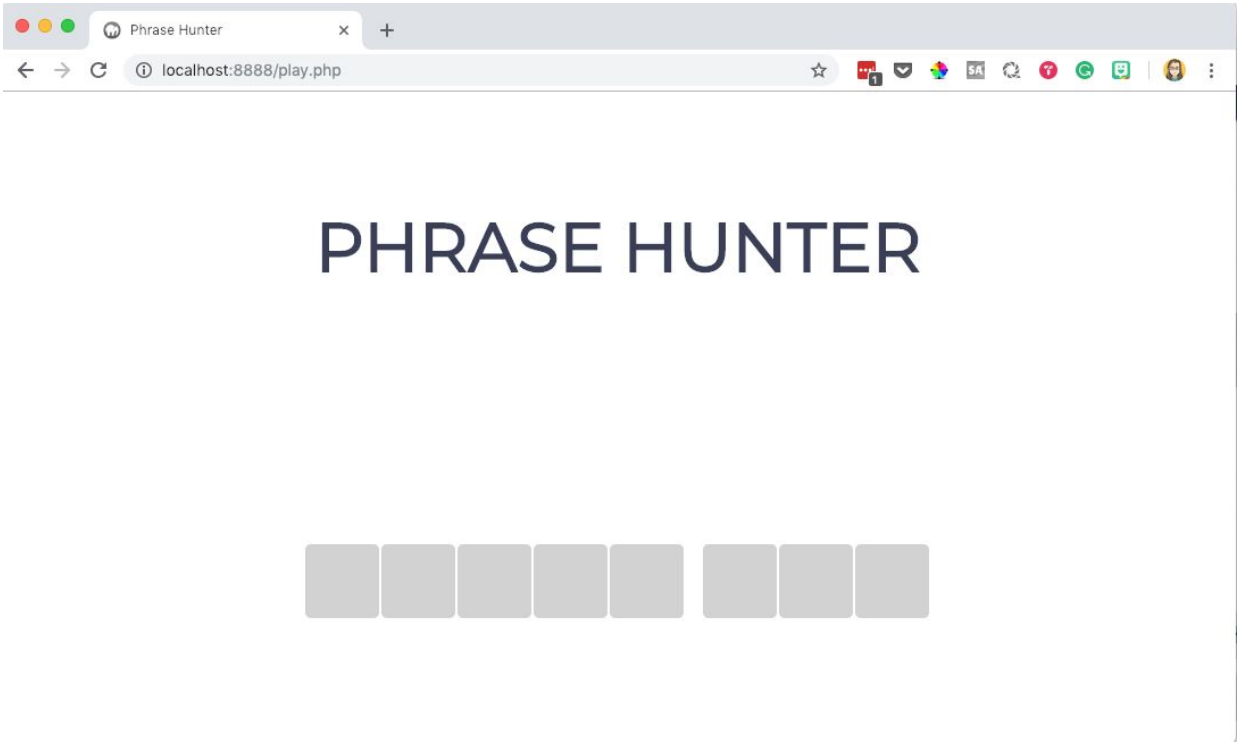
Step 5

Let's add the HTML and CSS for our page.

- Copy the HTML from index.html
- Within your play.php file, paste the HTML BELOW the PHP code block currently there.
- Remove all the HTML within the "main-container" div EXCEPT the h2 header. *There should be one opening div tag and one closing div tag.*
- Move the code to display your Phrase block after your h2 header.

Test Your Code!

Refresh your play.php page. Now your page should have the header and letter boxes like this:



Step 6

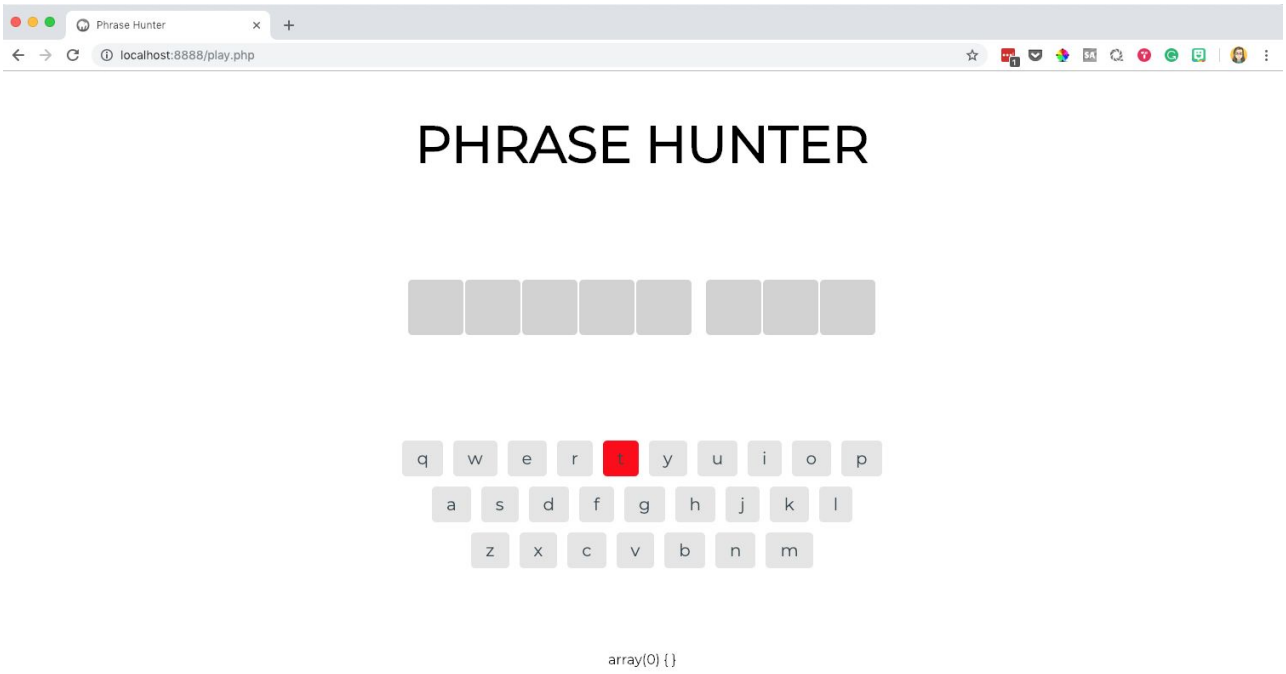
Head back to Game.php. Now we want to show our keyboard. Create a `displayKeyboard` function that builds the HTML of a form with each letter being a submit button. You can find a sample of the button HTML in example_html/keyboard.txt. Start by pasting this sample code into your `displayKeyboard` method. Make sure you surround this code with a form that POSTS to the play.php page. **Return** the HTML.

Test Your Code!

To make sure the code will display properly, add the following code in play.php after the phrase:

```
echo $game->displayKeyboard();
var_dump($_POST);
```

Refresh your play.php page. You should now see the keyboard below the letter blocks of the phrase. When you click on one of the keys, it should refresh the page and show the



For now the $_POST variable should either show an empty array or the start variable

```
array(0) { }
// or
array(1) { ["start"]=> string(10) "Start Game" }
```

Test if your form by choosing a letter. Your $_POST variable should then contain:

```
array(1) { ["key"]=> string(1) "u" }
```

Need help? Visit the **unit-04** Slack channel

Step 7

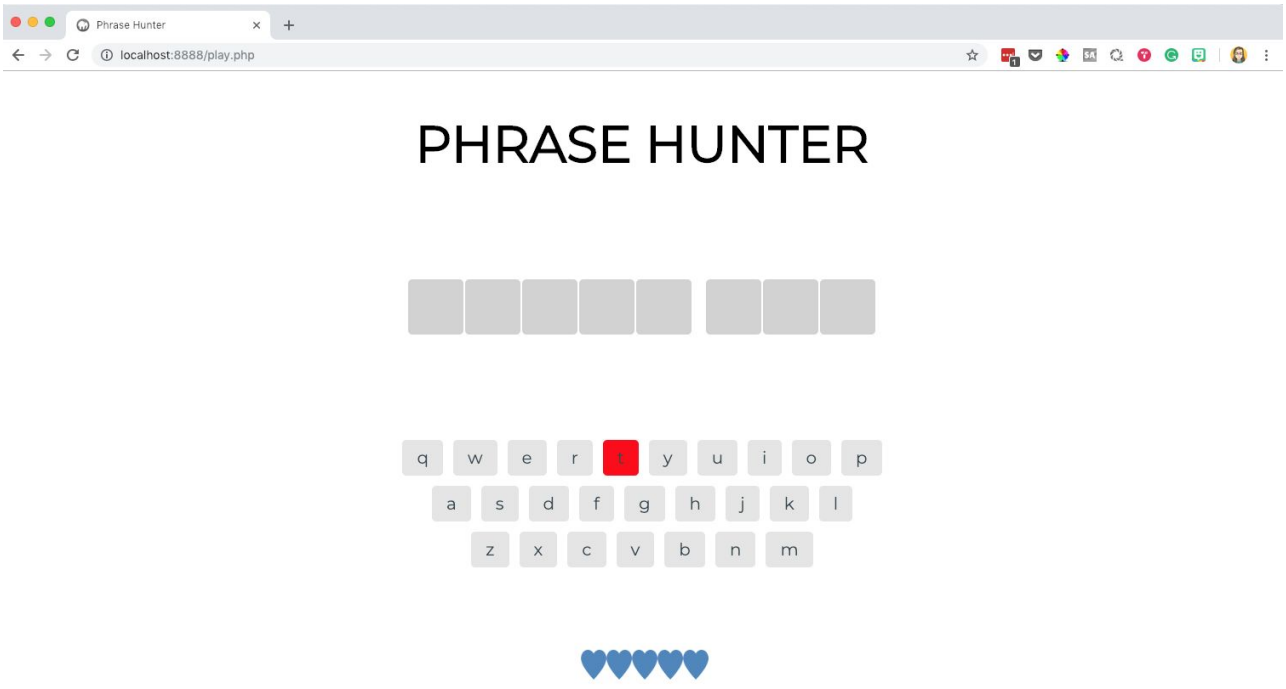
The final item we need to display is the score. Back in Game.php, add the `displayScore()` method. Once again, you can find an HTML sample ub example_html/scoreboard.txt. We will want to use a `for` loop to loop from 1 to the number of lives stored in the `lives` property of the game class. A little later we are going to display a `liveHeart.png` or a `loseHeart.png` depending upon how many lives have been lost. For now, just make them all `liveHeart.png`.

Test Your Code!

To make sure the code will display properly, add the following code in play.php after the keyboard:

```
echo $game->displayScore();
```

In the browser, refresh the play.php page. *If you want to refresh WITHOUT POST, place your cursor at the end of the address bar and press the enter key.*



You should now see the phrase boxes, keyboard and score. All our elements are set up and we're ready to add the functionality to our game.

Part II

Now it's time to start adding the actual game play with user interaction.

When a user clicks on the on screen keyboard, several things need to happen:

- The clicked/chosen letter must be captured and stored in a session, along with the current phrase
- 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 player.
- The game should check if a player has won/completed the phrase or if the game is over because too many lives were lost
- If the game is won or over, a message should be displayed on screen
- Once the game is over, there should be a button to restart the game

When the game is started/restarted

- There should be a new random phrase used for the current phrase
- The chosen letters should be cleared
- The phrase, keyboard and score should all be cleared to the starting views

To handle this functionality, we'll be updating some of the current methods and adding more, so let's get started!

Step 8

Let's start by storing our sessions. At the very top of our play.php file, we need to start our session. If we have POSTed a key click, then we should add that to a SESSION array named `selected`. Then our Phrase object should accept SESSION variables for the `phrase` and `selected` letters. We already have the interaction set up to handle if these passed variables are empty.

Test Your Code!

To make sure the session is working properly, add the following code in play.php.

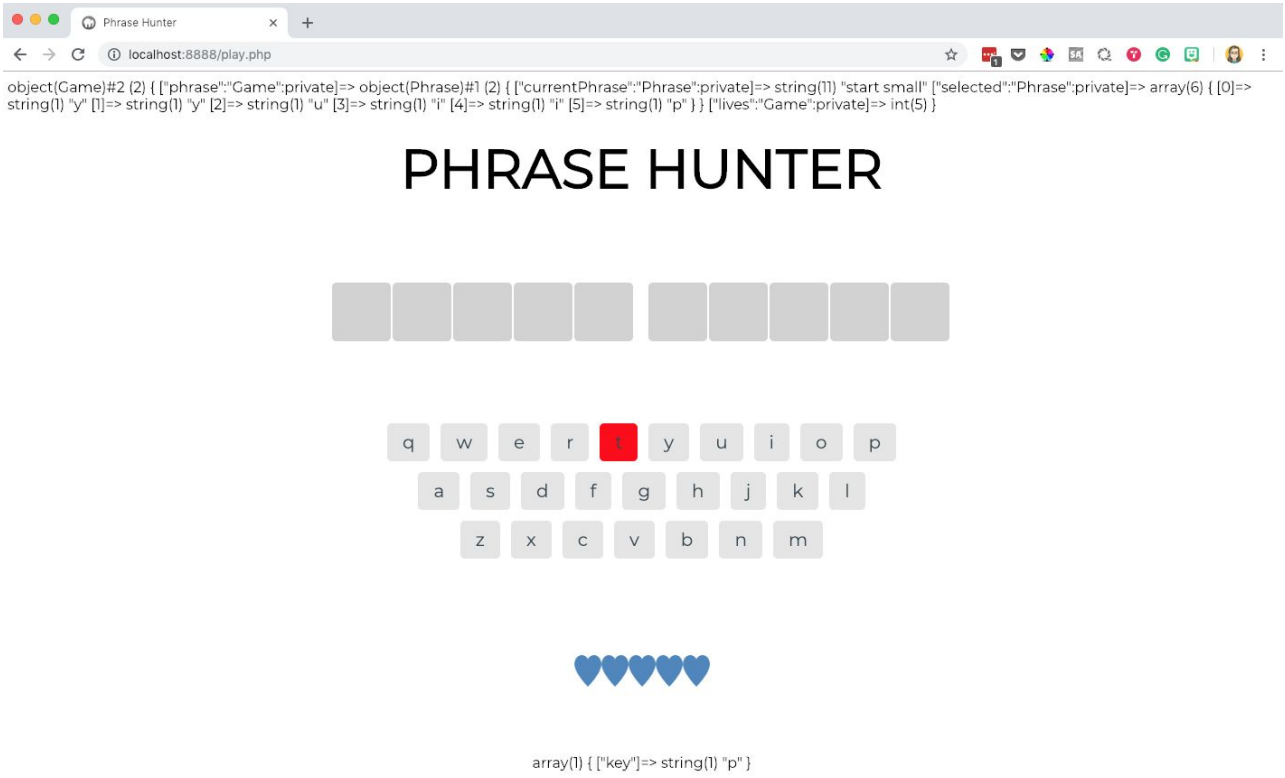
```
//after setting the session
var_dump($_SESSION);

//before creating the new Phrase object, set the phrase session
$_SESSION['phrase'] = 'start small';

//after creating both objects
```

```
var_dump($game);
```

In your browser, choose some letter. Your browser should show something like this:



Notice that your game board itself is not affected yet, and you can choose a letter more than once.

Step 9

The first thing we'll need to do is be able to check a specific letter against the letters in the current phrase. In the Phrase class we'll add our `checkLetter` method that accepts a single letter as it's parameter. We want to create an array of each unique letter in our `currentPhrase` property. We'll expand on what we did in the `addPhraseToDisplay` method.

```
//array of unique lowercase letters only in the currentPhrase
array_unique(str_split(str_replace(
    ' ',
    '',
    strtolower($this->currentPhrase)
)));
```

Now we will be able to use this method to check if a letter is within the current phrase.

Test Your Code!

In your play.php file, after creating your objects, add the following code to test the new method:

```
var_dump($phrase->checkLetter('a'));
```

In your browser, refresh your play.php page. You should now see either `bool(true)` or `bool(false)` based on whether the letter you passed to the method has been selected or not. You can test this by changing the letter you pass.

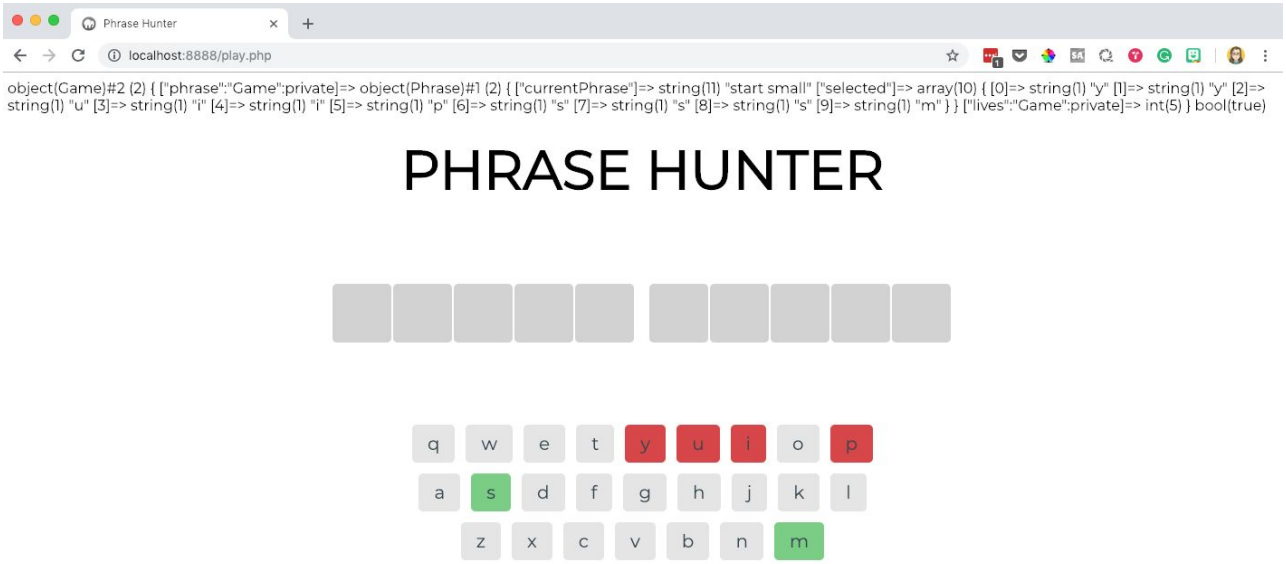
Step 10

Now we're ready to update our keyboard. This will require us to do the same thing for each key on the keyboard. In the Game class, create a new method for handling each letter key. The method should accept a single parameter which will be one of the letters for the keyboard.

A. The first thing we'll want to check is if the letter has been selected. We can use the `selected` array property from the internal phrase object. If the letter is **NOT** in the `selected` array, then we can return the letter without any added styling. You can copy a line from the keyboard and replace the letter.
B. If we pass this first condition, we can use the `checkLetter` method from the phrase to see if the letter is within the phrase. If so, add the class `correct`, and if not, add the class `incorrect`. In either case, the letter should be disabled
C. Now we can use this new method for each letter of our keyboard.

Test Your Code!

In your browser, refresh the play.php page. Your keyboard should now highlight the correctly guessed letters in green, and the incorrectly guessed letters in red. It should also prevent us from choosing any of those letter again.

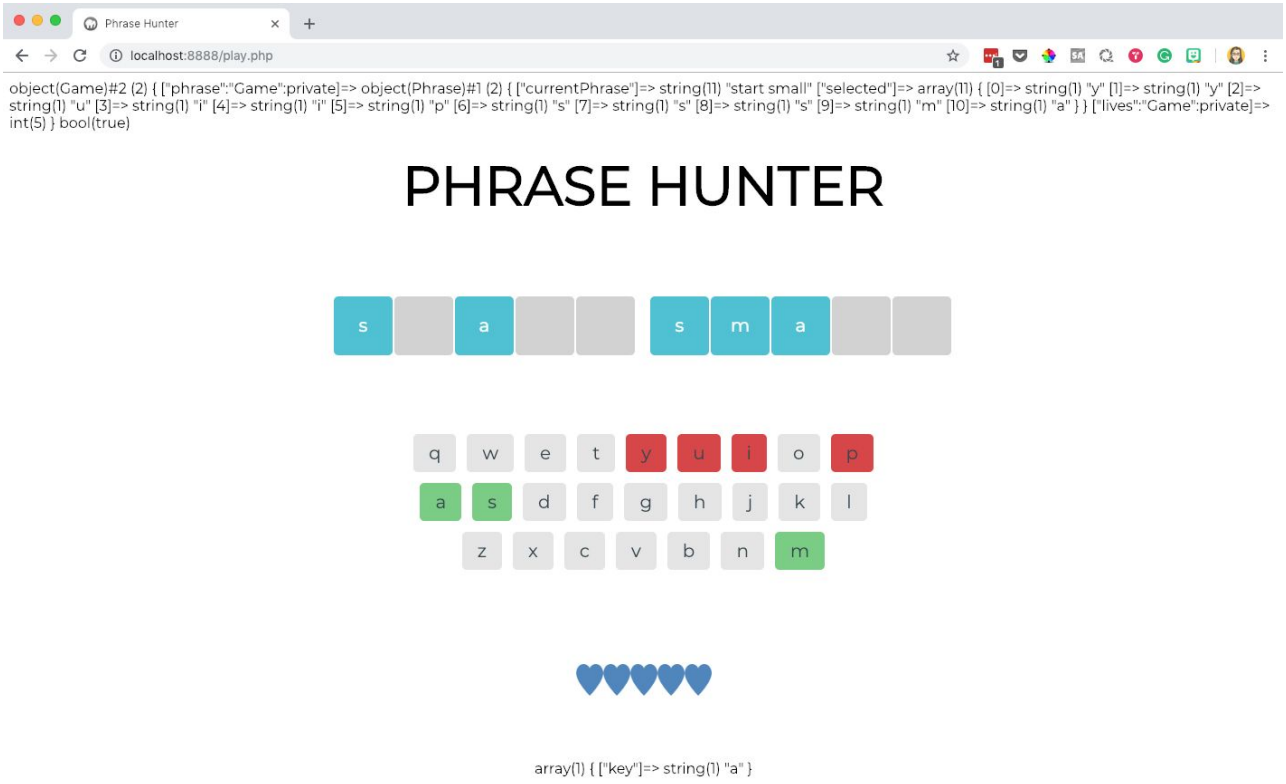


Step 11

Let's update our Phrase blocks next to show any correctly selected letters. In the Phrase class, we will update the `addPhraseToDisplay` method. Instead of the class "hide", We add a conditional to check if the letter has been selected. If selected, use the class "show" or else use the class "hide".

Test Your Code!

In your browser, refresh the play.php page. The correctly selected letters should be revealed in the phrase.



Step 12

Before we're ready to remove lives, let's reset our sessions and phrase. When we originally started the game, we submitted the POST variable `start`. Let's use this variable at the top of our play.php file to check if we are starting the game and need to **unset** our 'selected' and 'phrase' sessions. Using session_destroy will require a refresh of the page before the sessions are actually gone.

Test Your Code!

In your browser, go back to index.html and click "Start Game". The `selected` letters should be cleared now, but we will alway be using the same phrase since we are setting that manually. Let's go in and add our random phrase selection.

Step 13

In the Phrase class, we're going to add an array of phrases as a property of the Phrase class.

```
public $phrases = [
        'Boldness be my friend',
        'Leave no stone unturned',
        'Broken crayons still color',
        'The adventure begins',
```

Need help? Visit the **unit-04** Slack channel

```
                                    'Dream without fear',
        'Love without limits',
    ];
```

Now in the constructor, where we are setting the `currentPhrase` to 'dream big', instead we want to set the `currentPhrase` to one of our random phrases.

In the play.php file, we need to move the line where we set the 'phrase' SESSION below instantiating the Phrase object. Then we can set the 'phrase' SESSION to the `currentPhrase` instead of 'start small'.

Test Your Code!

In the play.php file, after instantiating your objects, echo $phrase->currentPhrase. In your browser, refresh the play.php page. Each time you POST with the start variable set, it should update the currentPhrase to a random phrase from the array.

Step 14

Now we're ready to calculate how many wrong letters have been chosen. We can calculate this number by comparing the selected letters with the UNIQUE letters from the phrase. We found the unique letters from the phrase in our `checkLetter` method in the Phrase class, see **Step 9**.

1. Move the unique letter array out to its own method named `getLetterArray`.
2. We can then use this new method in the `checkLetter` method and to calculate how many wrong letters have been chosen.
3. We know will need to calculate the wrong answers for both the score and also when we `checkForLose`. Let's create a new method to calculate the `numberLost`.
4. We can use the **array_diff** function to compare the `selected` letter array with the array returned from the `getLetterArray` method. It will returns the values in `selected` that are NOT present in any of the `getLetterArray`. We can then **count** the items in this resulting array and **return** that count.

Test Your Code!

Let's add some var_dumps to the `numberLost` method so we can make sure we have the correct results. Dump the following:

1. The results of the new `getLetterArray` method.
2. The results of the array_diff function.

In play.php file, after instantiating the Phrase object, echo $phrase->numberLost(). In your browser, go back to the index.html page and "Start Game". When you first start the game, your

first var_dump of `getLetterArray` results should be all the unique letters of the `currentPhrase`. If the `currentPhrase` equals "Broken crayons still color" var_dump will be:

```
array(13) {
    [0]=> string(1) "t" [1]=> string(1) "h" [2]=> string(1) "e"
[3]=> string(1) "a" [4]=> string(1) "d" [5]=> string(1) "v" [7]=>
string(1) "n" [9]=> string(1) "u" [10]=> string(1) "r" [12]=>
string(1) "b" [14]=> string(1) "g" [15]=> string(1) "i" [17]=>
string(1) "s"
}
```

The array_diff comparing the `selected` letters with the `getLetterArray` results will be an empty array, so the count equals 0. When you choose a letter that is NOT in the phrase, your array_diff will now contain that incorrect letter. Any incorrect letter should be added to that array, but any correct letter will not.

Step 15

Now we can update our `displayScore` method in the Game class. We'll use the `numberLost` method from the phrase object determine how many 'lostHeart' to display.

Test Your Code!

In your browser, choose letters and make sure the score matches up with the `numberLost`.

Step 16

Which brings us to the `checkForLose` method on the Game class. Using `numberLost` and the lives property, return true or false.

Test Your Code!

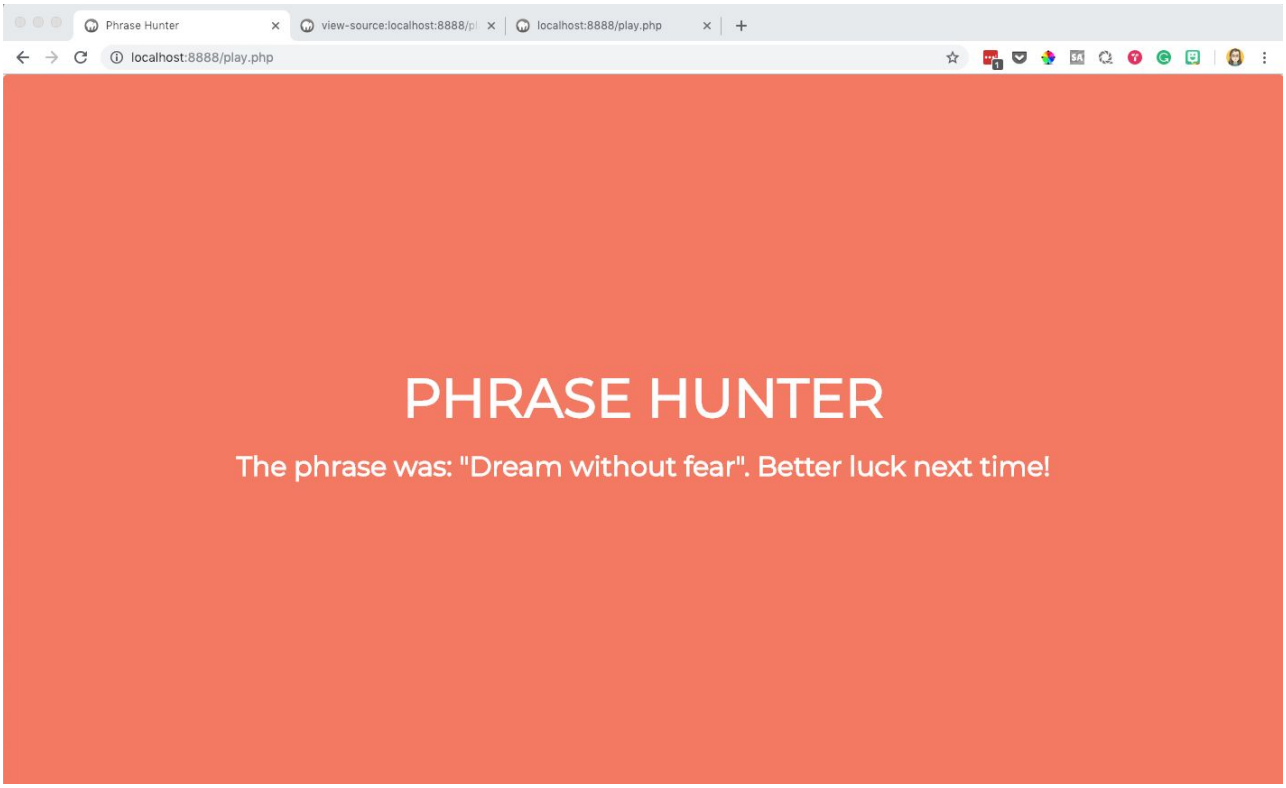
On the play.php page, after instantiating the objects, var_dump($game->checkForLose()); In your browser, continue selecting wrong letters until you have 5 or more wrong answers and the `checkForLose` returns true.

Step 17

Before we move on to `checkForWin`, let's add the `gameOver` method in the Game class, to display the `lose` overlay with message if `checkForLose` is true. See the example_html/game_over.txt for your `gameOver` messages. Return the overlay, or return false.

Test Your Code!

Within the "main-container" of page.php, call the `gameOver` method. In your browser, make sure you choose at least 5 wrong letter to make sure the overlay displays something like the following:



Step 18

Next we `checkForWin`. We calculate the correct answer much like we did the incorrect answer. Instead of returning values NOT present in another array, we want to return the values present in BOTH arrays. For this we can use the **array_intersect** function with the `selected` letters and those returned from the `getLetterArray` method. We can then compare the **count** of the array_intersect with the **count** of the `getLetterArray` to determine if we have matched all the letters and should return true or false.

Test Your Code!

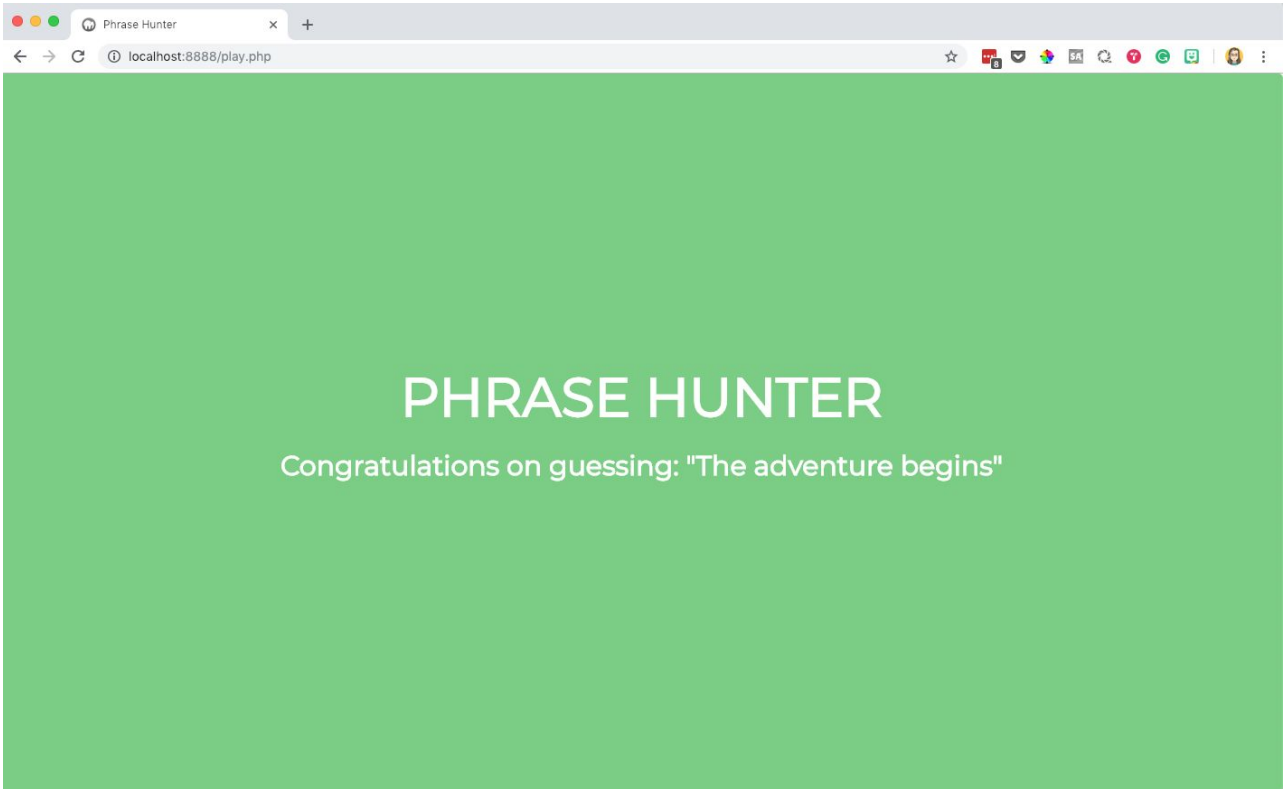
In your `checkForWin` method, var_dump the correct answers from the array_intersect.
In the page.php file, var_dump($game->checkForWin());
In your browser, go to index.html and "Start Game". This time choose the correct letters in the phrase. Watch the array_intersect increase and once all the letters have been guessed, the var_dump of `checkForWin` should equal true.

Step 19

Go to your `gameOver` method in the Game class, and add `checkForWin` to **return** the `win` overlay with message when the game has been won. See the example_html/game_over.txt for your `gameOver` messages.

Test Your Code!

In your browser, once you select all the correct letters, your page should look like this:



Step 20

The final step it to add the ability to restart the game. You can copy the form from the index.html page and add it to both the "win" and the "lose" overlays.

Final Testing

Clean up your test code by removing or commenting out any extra displayed variables (var_dump or echo). After you've completed all of the necessary steps, try to "win" a game by guessing all of the letters in the current phrase. Then try to "lose" the game by incorrectly guessing letters five times. Click the "Start Game" button on the win/loss screen overlay to check if you can successfully start a new game.

Awesome job on completing your game!

Suggestions for getting "unstuck".

1) Make sure you have all errors turned on so you can best diagnose any issues. Check the Basic Error Handling course for more information.
2) Check the "**How To Succeed At This Project**" section below. For every instruction step, you'll find a link to a specific resource to help you.
3) Reread the original project instructions and refer back to the material in the unit, checking for anything you may have missed or forgotten.
4) Do a good Google search. Even professional developers use Google a dozen or more times a day. Example search: "PHP multidimensional array".
5) Walk away from the computer for a minute. Often, just walking away from the screen for a few minutes can trigger a breakthrough.
6) Reach out on Slack. Briefly describe what's not working, where the problem is, and what you've tried so far, then post a friendly question on Slack. **Make sure you include your code, either as a code snippet or a link to your code online.**