JavaScript Assignment

Overview

In this assignment, you'll create one of two possible computer games: Word Guess or Psychic. These apps will run in the browser, and feature dynamically updated HTML and CSS powered by your JavaScript code.

Commits

Having an active and healthy commit history on GitHub is important for your future job search. It is also extremely important for making sure your work is saved in your repository. If something breaks, committing often ensures you are able to go back to a working version of your code.

* Committing often is a signal to employers that you are actively working on your code and learning.

  * We use the mantra “commit early and often.” This means that when you write code that works, add it and commit it!

  * Numerous commits allow you to see how your app is progressing and give you a point to revert to if anything goes wrong.

* Be clear and descriptive in your commit messaging.

  * When writing a commit message, avoid vague messages like "fixed." Be descriptive so that you and anyone else looking at your repository knows what happened with each commit.

* We would like you to have well over 200 commits by graduation, so commit early and often!

Submission on BCS

* Please submit both the deployed Github.io link to your homework AND the link to the Github Repository!

Before You Begin

1. Create a new GitHub repo called `Word Guess Game` or `Psychic-Game`, in accordance with the assignment you choose to complete. Then, clone it to your computer.

2. Inside your local git repository, create an `index.html`.

3. While still in your local git repo, create a directory called `assets`.

4. `cd` your way into the `assets` folder, then make three additional folders: `javascript`, `css` and `images`.

   * In the `javascript` folder, make a file called `game.js`. Use the `src` attribute of the `script` tag to link to this file, rather than embedding the code directly in your HTML document.
   * In the `css` folder, make a file called `style.css`.
   * Also in the `css` folder, make a file called `reset.css`. Paste into it the code from the Meyerweb reset stylesheet. If you opt to use Bootstrap instead of writing your own CSS, skip this step, and simply include a link to Bootstrap via CDN.
   * In the `images` folder, save whatever images you plan on using.

```
├── assets
│   ├── css
│   │   └── style.css
│   ├── images
│   └── javascript
│       └── game.js
└── index.html
```

5. Push the above changes to GitHub.

6. Choose whichever game you'd like to build. Making the Psychic game will prove less challenging than coding Word Guess. However, as the challenge of the Word Guess exercise provides a more comprehensive review of this unit's material, we suggest attempting that assignment first.

7. Note: There's no shame if you'd prefer submitting Psychic—it's still a proper challenge.

8. Push your selected game to Github Pages.

Option One: Psychic Game (Basic)

1. [Watch the demo](https://youtu.be/qTc45Lox97g).

2. You're going to make a game just like the one in the video. Essentially, the app randomly picks a letter, and the user has to guess which letter the app chose. Put the following text on your page:

3. Guess what letter I'm thinking of

4. Wins: (# of times the user has guessed the letter correctly)

5. Losses: (# of times the user has failed to guess the letter correctly after exhausting all guesses)

6. Guesses Left: (# of guesses left. This will update)

7. Your Guesses So Far: (the specific letters that the user typed. Display these until the user either wins or loses.)

8. When the player wins, increase the Wins counter and start the game over again (without refreshing the page).

9. When the player loses, increase the Losses counter and restart the game without a page refresh (just like when the user wins).

Option Two: Word Guess Game (Challenge - Recommended)

1. [Watch the demo](https://youtu.be/W-lJcC4tYFI).

2. Choose a theme for your game! In the demo, we picked an 80s theme: 80s questions, 80s sound and an 80s aesthetic. You can choose any subject for your theme, though, so be creative!

3. Use key events to listen for the letters that your players will type.

4. Display the following on the page:

5. Press any key to get started!

6. Wins: (# of times user guessed the word correctly).

   * If the word is `madonna`, display it like this when the game starts: `_ _ _ _ _ _ _ _`.

   * As the user guesses the correct letters, reveal them: `m a d o _ _ a`.

7. Number of Guesses Remaining: (# of guesses remaining for the user).

8. Letters Already Guessed: (Letters the user has guessed, displayed like `L Z Y H`).

9. After the user wins/loses the game should automatically choose another word and make the user play it.

Word Guess Game Bonuses

1. Play a sound or song when the user guesses their word correctly, like in our demo.
2. Write some stylish CSS rules to make a design that fits your game's theme.
3. **HARD MODE:** Organize your game code as an object, except for the key events to get the letter guessed. This will be a challenge if you haven't coded with JavaScript before, but we encourage anyone already familiar with the language to try this out.
4. Save your whole game and its properties in an object.
5. Save any of your game's functions as methods, and call them underneath your object declaration using event listeners.
6. Don't forget to place your global variables and functions above your object.
   * Remember: global variables, then objects, then calls.
7. Definitely talk with a TA or your instructor if you get tripped up during this challenge.

- - -

A Few Tips

1. **IMPORTANT:** Whichever assignment you choose, code your game one piece at a time! Code all of your apps one piece at a time. _*Always code one piece at a time!*_
2. Pseudocode your program and break the app down into tiny, manageable fragments. This will make the coding process much less frustrating and a veritable Mach number faster. Otherwise, you'll be chipping away at a giant chunk of abstraction for way too many hours.

   * The ability to solve a large problem by treating it as a set of smaller ones is the hallmark of a strong programmer. Best start adapting this into your development routine now, to better prepare for your more complex future projects.
   * Remember:
     1. Split the whole program into many distinct, pseudocoded problems.
     2. Focus on one of the smaller problems and solve it.
     3. Only when you solve one problem should you then move onto your next problem.

3. When you encounter bugs (and we all do), `console.log` will become your best friend. Regularly check your console to make sure your app is spitting out the right values.

   * As a more advanced—but more powerful—alternative, feel free to experiment with the [Chrome DevTools Debugger](https://developers.google.com/web/tools/chrome-devtools/).

4. Try your best to deliver a 'working/playable game' by the end of the deadline. If you're not making progress with Word Guess, switch gears to the Psychic game. Contact your TA/Instructor if you're not making progress after 2 hours. We're here to help!

5. Substance over style! Submitting a working game matters more that making a broken app that at least looks pretty. We're focusing on game mechanics, not just on the look and feel of your app.

6. That said, coding a functional app that also looks pretty would be impressive.

7. Always commit your work and back it up with GitHub pushes. You don't want to lose hours of your work because you didn't push it to GitHub every half hour or so.

   * **Commit often**.

8. Turn in anything you have! Even if you don't finish, we still want to see what you were able to accomplish in the time we gave you. This will help us know what concepts we could help you with, as well as what topics we should focus on in the coming lectures.

Reminder: Submission on BCS

* Please submit both the deployed Github.io link to your homework AND the link to the Github Repository!

- - -

Minimum Requirements

Attempt to complete homework assignment as described in instructions. If unable to complete certain portions, please pseudocode these portions to describe what remains to be completed. Adding a README.md as well as adding this homework to your portfolio are required as well and more information can be found below.

- - -

Create a README.md

Add a `README.md` to your repository describing the project. Here are some resources for creating your `README.md`. Here are some resources to help you along the way:

* [About READMEs](https://help.github.com/articles/about-readmes/)

* [Mastering Markdown](https://guides.github.com/features/mastering-markdown/)

- - -

Add To Your Portfolio

After completing the homework please add the piece to your portfolio. Make sure to add a link to your updated portfolio in the comments section of your homework so the TAs can easily ensure you completed this step when they are grading the assignment. To receive an 'A' on any assignment, you must link to it from your portfolio.

- - -

One More Thing

If you have any questions about this project or the material we have covered, please post them in the community channels in slack so that your fellow developers can help you! If you're still having trouble, you can come to office hours for assistance from your instructor and TAs.

Good Luck!