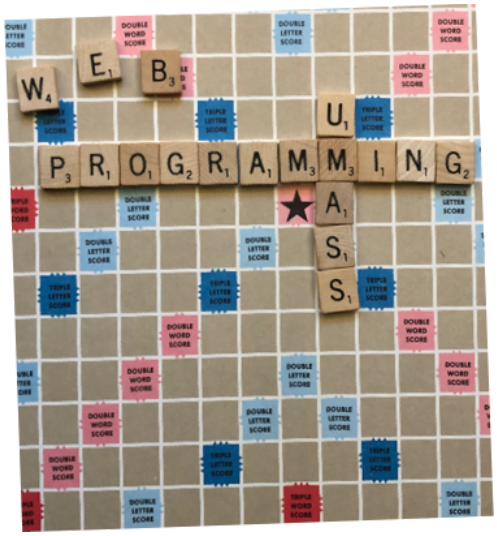
COMPSCI 326 - Web Programming
Homework 5 - Save Game & Refactor - *individual assignment*
due October 18, 2021, 11pm EDT
(GitHub classroom link: https://classroom.github.com/a/Epd7a7--)



This is the fifth part of a series of assignments around the game of Scrabble. We hope that it will be a fun experience in progressively learning all pieces of modern web development, so as to engineer a fully functional game. In this assignment, you add a new functionality to save the game state, create one more utility function, and do a little refactoring.

Please submit this assignment on GitHub Classroom. There will be an automated grader that will check the functionality of your submission. It will be helpful to come up with test cases, and we encourage you to share them amongst each other; this will make everyone's code better and is actually how Quality Assurance (QA) can work in practice. However, this is an individual assignment and you **cannot share code**; submissions will be run against plagiarism detection tools. Additionally, we will be spot checking the code for good coding practices. It is expected your code **does not** contain (1) extraneous variables/code, (2) missing semicolons, (3) missing curly braces, (4) use of double equals, (5) use of `let` when a `const` would suffice, (6) use of `var`. Furthermore, you should use whitespace consistently and to make the code legible. Now that you've learned how to use ESLint it should be easy to satisfy these requirements.

You will find a template with this homework's skeleton included in your repository when you create it. Please **do not** rename any of the existing files or change the directory structure. You are free to create more files and import them. However, you **cannot** use any external modules beyond those provided without prior permission.

1. Saving the state of the game

In this first part, you will make it possible for the game state to be saved and restored when the page is reloaded. More precisely, you should save the grid state in browser local storage, and restore it when the page loads. *Every time a word is placed on the grid* (using the controls from the last homework), *the version in storage should be updated*. When the page loads, you should *check if there is a version in storage*, and if there is, you should restore it. You should also *add a reset button* to your controls, which should clear the grid and delete the copy in local storage.

Since this part builds on your work from HW#4, we will not include any *empty* files in the template; you should just copy your previous work (or use the solution we will post during the week).

Note. We will learn more about local storage next week.

2. Checking if a word if valid

In this second part, you will write another utility function which will determine if a word is valid (i.e. it is in the dictionary). The word will be played by the user, and therefore can contain wildcards. The function should return `true` if the input matches any of the words in the dictionary, else `false`. You should place this function in `scrabbleUtils.js`.

```
/**
 * This function will check if a word is valid, that is if it matches any of the words in the
dictionary.
 * @param {string} word A string containing lowercase letters, with possible wildcards.
 * @returns {boolean} Returns whether the given word is a valid word.
 */
export function isValid(word) {
    // TODO
}
```

You should work on this part in Node, and use the following to import the dictionary.

```
import {readFileSync} from "fs";
const dictionary = JSON.parse(readFileSync("./dictionary.json"));
```

3. Refactoring

It is almost always the case in practical software development that code written originally will have to be repurposed and rearranged as the program evolves. In the first two assignments, you wrote a few methods that worked on a copy of the `availableTiles` parameter. It is likely that each of these functions first had code to copy the parameter. Since functions should be focused

units of code, in addition to the fact it is not good to duplicate logic, we would like you to *refactor* your code and create a local function (i.e. **not** exported) `copyAvailableTiles`. This function will simply take the `availableTiles` object and make a copy of it. You then will call this function in any and all methods that need to first copy `availableTiles`, replacing any existing copy code that may be present.

```
/**
 * This helper function will make a copy of a set of available tiles.
 * As you can see, this function is NOT exported. It is just a helper function for other
functions in this file.
 * @param {Object<string, number>} availableTiles A mapping of available tiles to their
amount.
 * @returns {Object<string, number>} A copy of the parameter.
 */
function copyAvailableTiles(availableTiles) {
    // TODO
}
```