## The game

The game presents the user with a Rack, which contains seven letters, and Board, which has seven spaces. The letters on the Rack are chosen so that they can be used to compose at least one five letter word. The user has to create a word on the board with five or more letters in it.

The user can move letters between the Rack and the Board by clicking on them. When a letter is clicked on the Rack, it moves to the first available space on the Board, and vice versa. There is a “Go” button which validates the word on the Board, and a feedback indicator which displays a green tick or red cross as appropriate.

## Steps

1. Project setup
   1. Create an HTML page, a JS file and a stylesheet to form a single project
   2. Add a reference to jQuery
   3. Add a document ready function and make sure it is called
2. Initial markup
   1. Using bootstrap, add a row with seven bordered containers to represent the board
   2. Add a similar row to represent the rack
   3. Add the letters BELRAST to the rack
   4. Add a “Go” button
   5. Add a container for feedback
   6. Add a “Test” button. This won’t be used in the game but will be useful for testing features in isolation
3. Letter movement
   1. Write a function which takes a jQuery element for a container on the rack, and moves the contained letter to the first empty container on the board. Test this by moving the A and the B from the rack to the board when the Test button is clicked
   2. Write a function which takes a jQuery element for a container on the board, and moves the contained letter to the first empty container on the rack. Extend your test to move the A back from the board to the rack; verify visually that the rack contains ELRSTA.
   3. Combine these two functions into a single function which takes an additional parameter which is true if the letter is to be moved from the rack to the board and false if the letter is to be moved from the board to the rack
   4. Modify the function so that it degrades gracefully (ie, doesn’t fail, just does nothing) if there is no letter in the given container
   5. Add a click handler so that this function will be called when the user clicks on a container in the rack
   6. Add a click handler so that this function will be called when the user clicks on a container in the board
4. Degenerate functions
   1. Remove the letters in your markup so that the rack begins empty
   2. Add a function getInitialRack which returns an array of letters containing BELRAST
   3. Add a function initialiseRack which calls getInitialRack and puts the letters it returns into the containers on the rack
   4. Add a function initialiseBoard which clears the rack and the board and calls initialiseRack
   5. Call this in your document ready function

## Step 5 – Preparing to use templates

We’re going to use a library called JSRender which lets us create HTML from templates. A template is a block of HTML with placeholders that we can replace with data. We can also mark bits of the template as conditional (so we can render some bits of HTML in some cases and other bits in other cases), and we can put loops in the template (so, eg, produce one row of a table for each element in an array). For now, though, we’ll keep things simple.

What we are going to do is create a template which draws a tile for a given letter. Then, we are going to use this template to draw tiles where, at the moment, we just set the text of a container to a given letter.

The overall steps we will follow are:

1. Create some markup that shows the letter A in a way that looks a bit like a scrabble tile
2. Create a template from this markup and use it to show just the letter A
3. Modify the template so it can show any letter
4. Write a function to render the template into a container on the board or rack
5. Call this function instead of using .text() to display letters in the game

Here are the first two steps in more detail:

1. In the first container (ie div) on the rack, create another div containing the letter A, with the class tile.
2. Add a “return” as the first line of initialiseBoard – this will prevent it doing anything so we can play with things for a bit
3. Add to your css to make the letter A look like a scrabble tile – eg, set an ivory background colour, increase the font size, add rounded borders 🕿
4. Add a reference to JSRender to your project (ie, add a script tag to your html). The url is https://www.jsviews.com/download/jsviews.min.js
5. Add a section to your HTML as follows:

<script type="text/jsrender" id="tile-template">

</script>

1. Put the markup for your tile inside this section
2. Delete all the code from your click handler for the test button
3. Add the following code to that click handler:

let cont = $(".letter")[0];

let template = $.templates("#tile-template");

template.link(cont);

1. Remove the markup from the first container on the rack, so that all the containers on the rack are empty
2. Run and click the test button. A tile with an A should appear in the first space on the rack🕿

## Step 6 – Data in templates

The template we’ve got so far will only display the letter A, which isn’t much use. So, we’re going to set things up so we can pass jsRender an object with a “letter” property, and render this letter with our template

1. In the template (ie, the markup we put in the script tag in the previous step), replace the letter A with {{:letter}}.
2. In the test button click handler, replace the template.link call with

template.link(cont, {letter:”B”}

1. Clicking the test button should now show a tile with the letter B in it in the first container on the rack🕿

At this point, we’ve established how we can use jsRender to create a tile for any letter, so the next thing we need to do is create tiles in our game where we are currently just writing letters.

1. Create a function, renderTile, which takes two parameters, a container and a letter, and renders the tile template to the container to show the letter. You can base the code for this function on the code in the test button click handler🕿
2. Change the code in the test click handler to call this function so that it renders a C in the first space on the rack instead of rendering a B. 🕿
3. There are two places in the code where we set the text of a container on the board or the rack. Change these so that you don’t use the text function, but instead call renderTile. Note that you can’t do something like e.renderTile(letter) because renderTile isn’t part of a jQuery element; you will have to do something more like renderTile(e, letter) 🕿
4. Remove the “return” we added to initialiseBoard, and check everything works🕿

## Step 7 – Degenerate functions for answer validation

We’ve now got templates working, so we’re going to move on to using Ajax. We’ll use this for two things – generating the rack and checking the user’s answer. The second of these is a bit easier, so we will tackle it first. As usual, we will start with some degenerate functions so the overall flow is right.

1. Add a function showFeedback which takes a parameter indicating whether the word on the board is or isn’t valid and puts the text “right” or “wrong” in the feedback container as appropriate.
2. Add a function validateWord which takes a single parameter, “word”, and always returns false
3. Add a function getWordOnBoard which returns “ABLEST” 🕿
4. Add a click handler for the Go button. In this function, first call getWordOnBoard, and then pass the word it returns to validateWord. Then, pass whatever validateWord returns to showFeedback. To do all this, it’s probably easiest to use two local variables called “word” and “isRight”. 🕿
5. Verify that clicking Go now places “wrong” in the feedback container
6. Change validate word to return true. Verify that clicking Go now places “right” in the feedback container🕿

## Step 8 - Fleshing out answer validation using Ajax

Change validateWord so it looks like this:

async function validateWord(word) {

let url = "http://tom.mo2.dinksurveys.net/api/1.0/isword?word=" + word;

let result = await $.ajax(url);

return result.isWord;

}

You can see that there are two new keywords in this function, async and await. We can talk a bit more about what these do, but because we’ve introduced them, we need to use them in the click handler for the Go button. Basically what they do is tell the browser it can go off and do something else while it’s waiting for the server. My changes to the Go handler look like this:

$(".button-done").on("click", async function () {

let word = getWordOnBoard();

let isRight = await validateWord(word);

showFeedback(isRight);

});

The key thing is that we have the word “async” in front of “function” and “await” in front of validateWord. 🕿

1. Verify that clicking Go still places “right” in the feedback container
2. Change getWordOnBoard so that it returns a string made up of the letters on the board. You can use jQuery each to do this. The trick here is to declare a variable and set it to an empty string, then add one letter to the string each time the function you pass to “each” executes. Test this by calling it from the Test button click handler and logging or alerting the return value (you can remove the other code in this handler) 🕿
3. Test the overall process by playing the game, creating the words ABLER, ABLEST, BLEST and STEBL on the board. The first three should show “right” and the last “wrong” (you will need to change the initial letters so that they have an E in them rather than an F).

## Step 9 – Initialising the Rack at random

We’ve now got our game working fairly well, but we’re using the same rack of letters every time. We can use an Ajax call to get a random five letter word for our rack, and then we need to add two random letters to this, and scramble the result.

1. Add a function getFiveLetterWord which returns “ABLER” (we’ll replace this with an Ajax call later)
2. Modify getInitialRack to call getFiveLetterWord, create an array from the word it returns, and add the letters S and T to it. You can use split to turn a string into an array. 🕿
3. Add a function randomLetter as follows:

function randomLetter() {

return String.fromCharCode(Math.floor(Math.random() \* 26) + 65);

}

1. Modify getInitialRack again so that instead of adding S and T every time, it adds two letters generated by calling randomLetter (you’ll need to call it twice, once for each letter) 🕿
2. Modify getInitialRack again so that it randomises the order of the letters in the array before returning it. You should refer back to Scrabble 1 to remind yourself how to do this🕿

## Step 10 - Choosing random words with Ajax

You can get a five letter word from the server using <http://tom.mo2.dinksurveys.net/api/1.0/getWord?letters=5>. This will give you an object that looks like this:

{

"letters": 5,

"word": "PLUSH"

}

Have a look at what you get if you visit the “isWord” url:

<http://tom.mo2.dinksurveys.net/api/1.0/isWord?word=ablest>

You can see that the object you get back has an “isWord” property, and we refer to this in validateWord.

The next task, then, is to modify getFiveLetterWord so that it makes an Ajax call to get a word from the server. Look at validateWord to see how we make an Ajax call. You will need to change the url, and extract “word” rather than “isWord” from the object you get back. You will also need to add the async and await keywords. 🕿

Because we are using async and await in getFiveLetterWord, we also need to use them on all the functions that call it, and all the functions that call them, and so on. This means you will need to modify getInitialRack, initialiseRack, initialiseBoard, and your document ready function. The rule is that you need to declare these as async functions, and use await wherever you call them. 🕿