3.1.4 Create the Project Files and Structure

Even though we won't be writing HTML in this lesson, browsers are designed to open HTML files, so we'll create one with the sole purpose of loading our JavaScript code.

In the command line, use the `touch` command to create an HTML file in the root of our project directory:

```
touch index.html
```

Next we'll create an `assets` directory to contain our app's JavaScript file. This will entail creating an `assets` subdirectory, a `js` subdirectory in `assets` (much like how we created a `css` directory for our style sheets), then creating a `game.js` JavaScript file in the `js` folder.

Sounds like a lot of steps, but luckily we can chain these commands together:

```
mkdir assets; cd assets; mkdir js; cd js; touch game.js; cd ../../;
```

Every command we just used is one we've seen and used before, but never in this fashion, so let's recap.

We were able to execute all the commands by writing them together on the same line, separated by semicolons, `;`. The semicolon is used to delimit each command. The computer can simply execute each command as if they were on separate lines.

NERD NOTE

A semicolon in programming is often referred to as a "terminator," meaning that it is used to inform the computer reading and running the code where a command ends so the next command can begin.

Now that we've created all of our files, let's open our project in VS Code. Remember, there are two main ways to do this:

- In VS Code, go to File > Open Folder. Then use the file navigator to locate the `robot-gladiators` folder, select it, and click Select Folder. This will open the entire project in VS Code.

- In the command line, make sure you're in the root of your project's directory and type the following command to automatically open the entire project:

```
code .
```

PAUSE

What does the period `.` in above the command above mean?

A single period `.` tells the computer we're referring to the current active directory. The `code .` command loads every file and folder in the current directory into VS Code.

Hide Answer

After VS Code has everything loaded, open the `index.html` file in the editor, then copy and paste the following code into it:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Robot Gladiators!</title>
  </head>
  <body>
    <script src="./assets/js/game.js"></script>
  </body>
</html>
```

Save `index.html` and let's take a look at the code we pasted. It looks like the boilerplate for a blank webpage, except there's something in the `<body>` that we've never seen before:

```
<script src="./assets/js/game.js"></script>
```

The `<script>` element is used to incorporate JavaScript code—in our case, the `game.js` file we created earlier—into an HTML file.

Notice the `src` attribute that's being used here. When we use `<script src="path/to/file.js"></script>`, we are instructing the browser to go

look for a JavaScript file at that path's location and incorporate it into the HTML file.

Why do you think we put this towards the bottom of the HTML file? When a browser sees a `<script>` element in the HTML file, it immediately runs the JavaScript in the file, preventing the browser from moving onto the next HTML tag until it's finished. This could delay the HTML content from making it to the page. Even worse, if the JavaScript file contains an error, the browser could just stop loading the page altogether. This means the page would indefinitely stay blank, leaving the user confused and frustrated.

Because of this, it is best practice to load the CSS first; that way, the HTML can load with the styles in place as it displays on the page. The `<script>` element is added after all the important HTML content, right above the closing `</body>` HTML tag. This way, even if the JavaScript file takes a little long to load or breaks, there is still a styled HTML page for the user to see.

Make Sure the JavaScript Loads

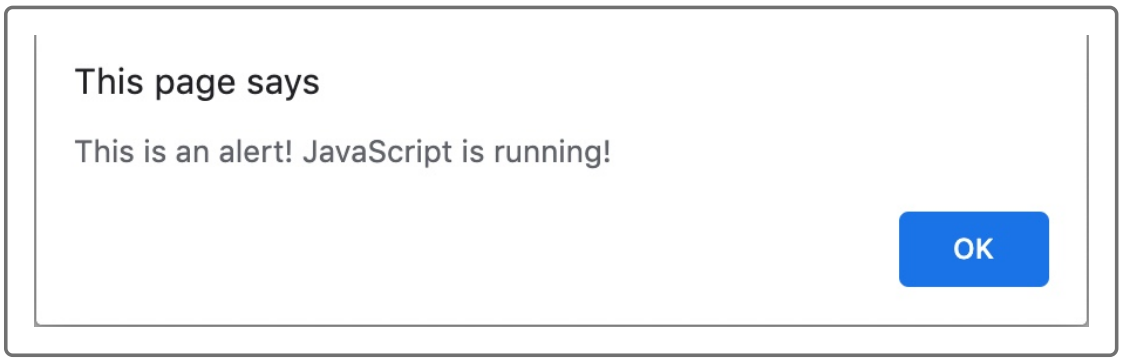
We're almost ready to start creating our game, but first we need to make sure the HTML file is reading the JavaScript file correctly.

In the `game.js` file, add the following line of code:

```
window.alert("This is an alert! JavaScript is running!");
```

Now save the file and open the HTML page in the browser. The first and only thing you should see is a small window that says "This is an alert! JavaScript is running!"

It should look something like this:



Notice that you can do nothing in this browser tab until you acknowledge the dialog window and click the button. This is called an **alert**.

Hey, what do you know? You just wrote your first line of JavaScript!

CONNECT THE DOTS

Consider what the word `window` might refer to in that piece of code. Also think about the `alert()` syntax and which programming concept it falls under. Does it store data or perform an action?

Because we just created our initial project structure, let's commit the repository's code to GitHub. Go through your usual Git add, commit, and push workflow now.