

Teddywaddy Code Club

Activity 6a

Coding with three.js



three.js Coding

three.js is a fantastic platform for coding any 3D application from gaming to engineering displays. Getting started is easy, but any advanced application will require substantial understanding of 3D coordinate systems.

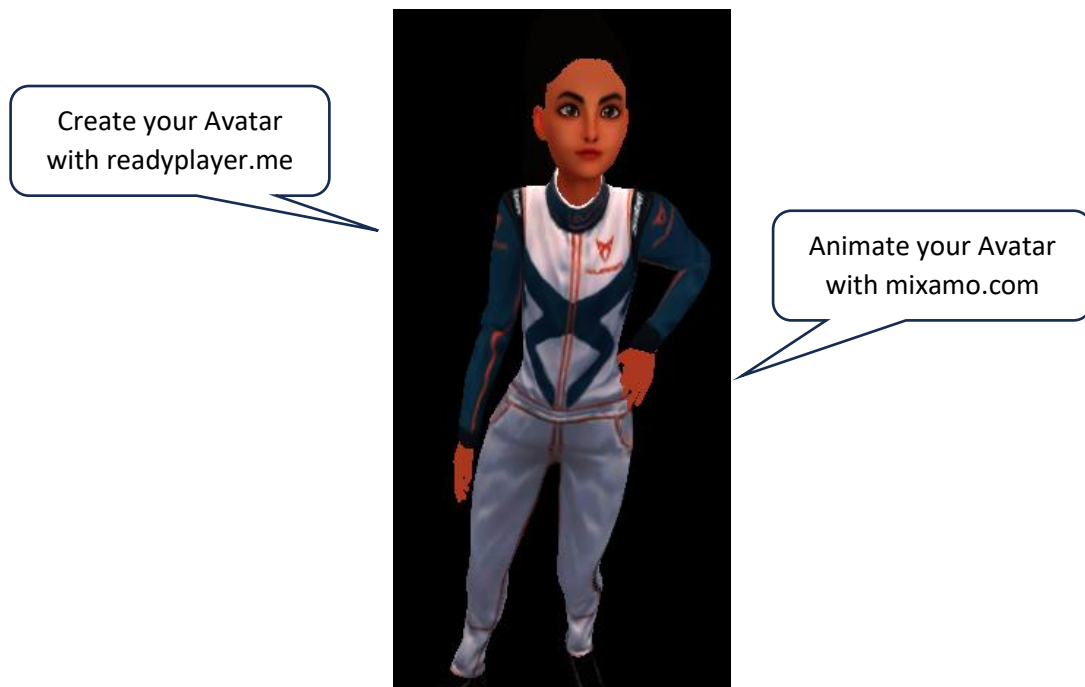
Gaming will also require additional knowledge of animation and character data structures. Many online sites offer free and low-cost models to download. Also, it is possible to create characters and Avatars by visiting sites such as <https://readyplayer.me/> and adding animations with <https://www.mixamo.com/>.

The good news is that there is a lot of online support and many examples to begin simply and work towards more complex displays.

All three.js coding is done in JavaScript - three.js is a library that extends what can be done with JavaScript.

Just like HTML and JavaScript there are various boilerplate starting points for coding in three.js.

One of the main considerations is how the three.js library will be accessed. The following example is one of the simplest ways to get started.



All the three.js activities will be easier to understand with some JavaScript background. However, they can be completed and will assist in learning JavaScript along with the three.js code.

Important: Refer to <https://threejs.org/> for tutorials, examples and documentation. Reading and interpreting the documentation is a **critical skill**. Even if you download the files, try to refer to the documentation about things like Scenes, Geometry, Lights and so on to start understanding how three.js works.

three.js Boilerplate

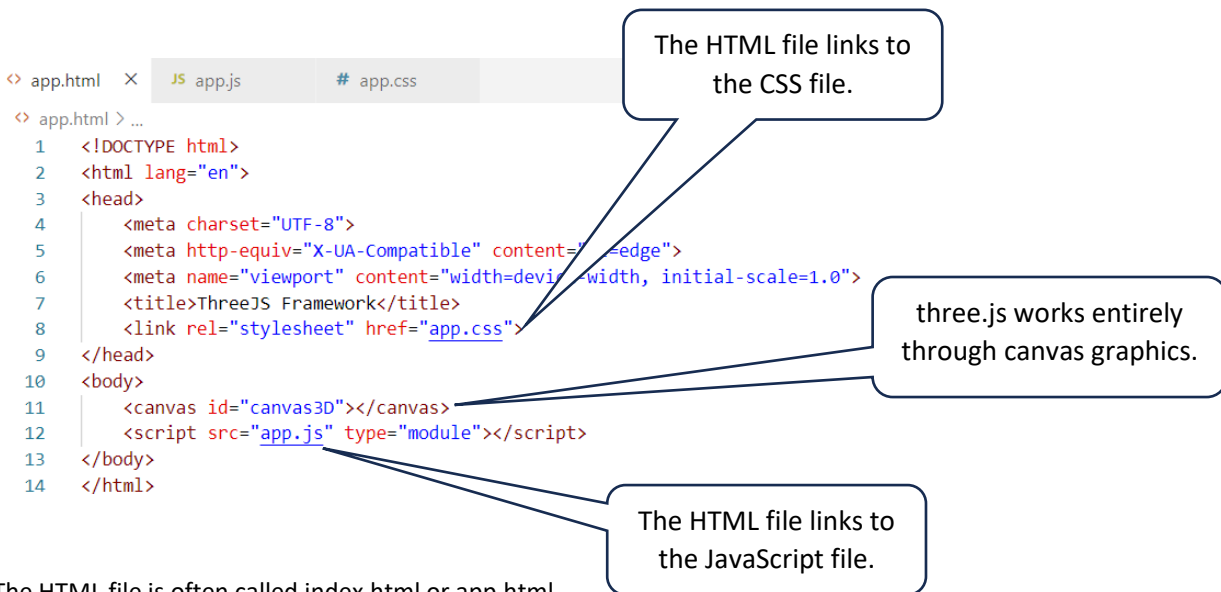
Just like most web pages, a minimum of three files are needed.

HTML file

CSS file

JavaScript file

The minimum HTML file



```
<> app.html X JS app.js # app.css
<> app.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="ie=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>ThreeJS Framework</title>
8   <link rel="stylesheet" href="app.css">
9 </head>
10 <body>
11   <canvas id="canvas3D"></canvas>
12   <script src="app.js" type="module"></script>
13 </body>
14 </html>
```


The HTML file links to the CSS file.

three.js works entirely through canvas graphics.

The HTML file links to the JavaScript file.

The HTML file is often called index.html or app.html.

The minimum CSS file



```
<> app.html JS app.js # app.css X
# app.css > ...
1 html{
2   margin: 0;
3   height: 100%;
4 }
5
6 body {
7   margin: 0;
8   height: 100%;
9   background-color: black;
10  overflow: hidden;
11 }
12 #canvas {
13   width: 100%;
14   height: 100%;
15   display: block;
16 }
```

The main purpose of the CSS file is to create a full window canvas. It is possible to have HTML elements appearing as well as, or over the canvas graphics.

The minimum three.js file (one of many possibilities)

```
index.html JS app.js X # app.css
JS app.js > app
1 // ThreeJS framework
2
3 import * as THREE from 'https://cdn.skypack.dev/three@0.136.0';
4
5 var basicScene;
6 var camera;
7 var renderer;
8
9 function app() {
10
11 // Create the Scene object
12
13 basicScene = new THREE.Scene();
14 basicScene.background = new THREE.Color('black');
15
16 // Create a camera object
17
18 camera = new THREE.PerspectiveCamera(75,window.innerWidth/window.innerHeight,0.1,100)
19
20 camera.position.x = 0;
21 camera.position.y = 0;
22 camera.position.z = 10;
23
24 var light = new THREE.AmbientLight(0x404040,5);
25
26 basicScene.add(light);
27
28 // Create a 3D scene
29
30 // Setup a renderer
31
32
33 const canvas = document.getElementById("canvas3D");
34 renderer = new THREE.WebGLRenderer({canvas});
35 renderer.setSize(window.innerWidth,window.innerHeight);
36
37 // Set up the render loop
38
39 function render() {
40
41 // Any scene updates go here
42
43
44
45 // Render
46
47 renderer.render(basicScene, camera);
48
49 window.requestAnimationFrame(render);
50
51 }
52
53 // Start the animation process
54
55 window.requestAnimationFrame(render);
56
57
58
59 // Start the whole three.js application
60
61 app();
62
```

Bringing the three.js library into this code

A Scene is required and has many properties that can be set, like the background colour

A Camera is required and what the camera sees is what is displayed on the window. The camera can be placed anywhere in 3D space.

A Light is required otherwise the scene will be dark (black).

These two lines connect the three.js library to the canvas. Anything the camera sees is displayed on the canvas – this is called rendering.

This is the line that renders the scene on the canvas.

Tells the browser what function to call to perform any scene updates before the window is refreshed.
More explanation on this later.

Invoke the app() function.

The above minimum three.js file doesn't display anything because there is nothing in the *Create a 3D scene* section.

First three.js app – threejs01

Create a **new folder** and then open that folder in VS Code. It is important that a new folder is used for each project to keep all the files organised such that filenames, such as index.html or app.js, can be reused.

Download or create the HTML, CSS and JavaScript files as already shown in this activity. Look for the threejs01 set of files.

Download the files at <http://www.teddywaddy.com.au/resources.html#ThreeJS>

Make sure the HTML file refers to the app.css file in the <link> tag and the app.js file in the <script> tag.

Open the HTML file in Live Server. A black window should appear – because as yet there are no actual visible models. You should also check the Console for any error messages.

Add some geometry to the scene.

```
// Create a 3D scene

// Create a cube, then add it to the scene

var geometry = new THREE.BoxGeometry(1, 1, 1);
var material = new THREE.MeshLambertMaterial({color: 0x7CFC00}); // lawngreen
var cube = new THREE.Mesh(geometry, material);

cube.position.y = 2;
cube.position.x = 2;

scene.add(mesh);
```

Add mouse control

To enable some capability to alter the view of the scene (using the mouse), make the following changes.

```
// ThreeJS framework
import * as THREE from 'https://cdn.skypack.dev/three@0.136.0';
import { OrbitControls } from 'https://cdn.skypack.dev/three@0.136.0/examples/jsm/controls/OrbitControls.js';
```

Add the orbit controls code straight after the renderer set up.

```
// Setup a renderer
const canvas = document.getElementById("canvas3D");
renderer = new THREE.WebGLRenderer({canvas});
renderer.setSize(window.innerWidth,window.innerHeight);

// Orbit controls

const control = new OrbitControls(camera, canvas);
control.target.set(0,0,0);
control.update();
```

Add this line straight after the import of the three.js library.

Add these lines

The mouse should now be able to rotate and zoom in and out.

Extensions

1. Try changing the background to a different colour.

```
basicScene.background = new THREE.Color('black');
```

2. Ambient light is everywhere, so models don't have light and dark sides. Try changing to a point light. And try changing the parameters and position of the point light.

```
var light = new THREE.AmbientLight(0x404040,5);
```

Replace this line

```
var light = new THREE.PointLight(0xffffff, 1, 0);  
light.position.set(0,0,2);
```

With a point light you need to set a position

0xffffff means white light
1 is the intensity of the light
0 is how far away the light works and 0 means infinite

3. Try changing the colour, size and position of the cube. One common issue is that the camera can end up inside a model or some geometry, which can mean a black scene (no light).

```
// Create a cube, then add it to the scene
```

```
var geometry = new THREE.BoxGeometry(1, 1, 1);  
var material = new THREE.MeshLambertMaterial({color: 0x7CFC00}); // lawngreen  
var cube = new THREE.Mesh(geometry, material);
```

Change the cube size.

```
cube.position.y = 2;  
cube.position.x = 2;
```

Change the cube position.
Add a z position value as well.

4. There are many geometries in three.js. Refer to the documentation (<https://threejs.org/>) and try adding a sphere, cylinder or plane geometry. Practice positioning the items to start understanding the 3D space.