Introduction to Multimedia

Assignment 1, 2043 words

Candidate Number: 218721

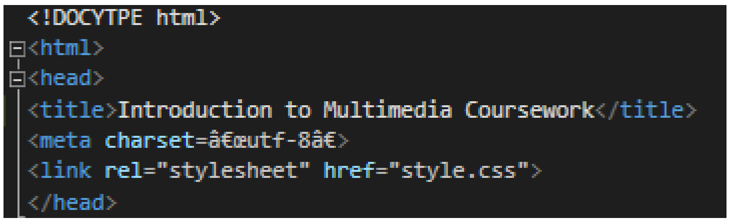
Contents

Developing the website

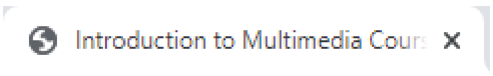
When developing my website, I kept in mind the key functionality outlined in the coursework specification.

Titles

When starting off creating the home page, I immediately started tackling the first criteria, which was to get a title on the home page. I wasn't entirely sure about the meaning of this criteria, so I decided to tackle it in multiple ways to be certain. The first interpretation I had of this criterion was to give the home page a title using <title> tags in the header, which is incredibly simple. This simply changes the text displayed on the tab in the browser:

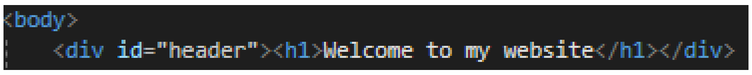


```
<!DOCTYPE html>
<html>
<head>
<title>Introduction to Multimedia Coursework</title>
<meta charset=â€œutf-8â€>
<link rel="stylesheet" href="style.css">
</head>
```

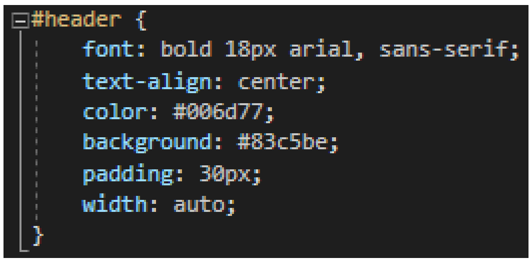


Introduction to Multimedia Cour

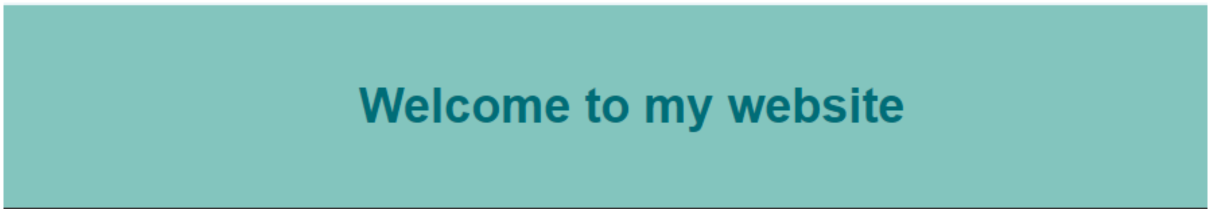
The other way I interpreted this was to give the home page and each subsequent page a literal title at the top of the screen. This was achieved by simply sandwiching an <h1> tag between a custom div that used CSS to create a consistent title at the top of the screen:



```
<body>
    <div id="header"><h1>Welcome to my website</h1></div>
```



```
#header {
    font: bold 18px arial, sans-serif;
    text-align: center;
    color: #006d77;
    background: #83c5be;
    padding: 30px;
    width: auto;
}
```

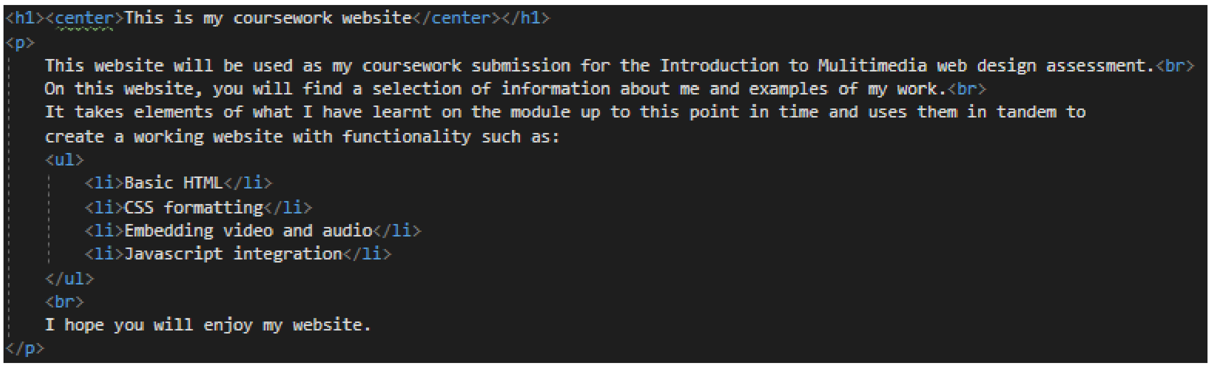


Welcome to my website

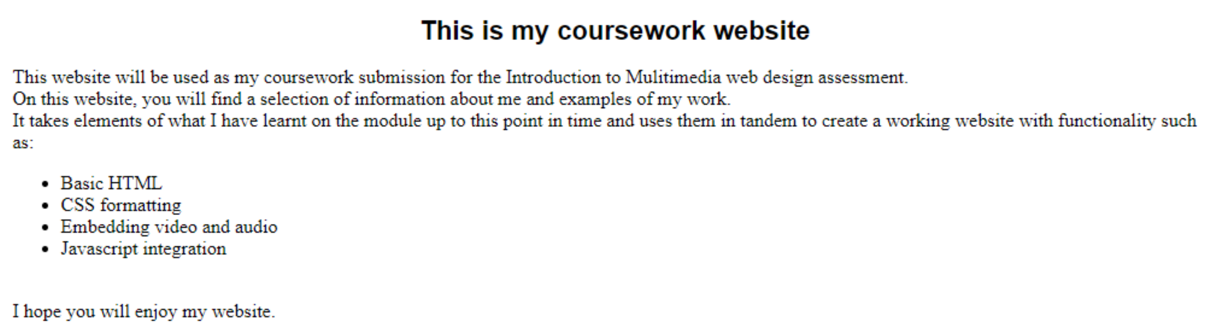
Both these methods are easily reusable, so I have applied both styles of title to every page on my website.

Text

Next, I decided to add some text. This was incredibly simple, using an <h1> tag for the heading, followed by a <p> tag with the desired text for the main bulk of the text. I also made use of the <br> tag and the list tags to format the text so it would be more readable:



```
<h1><center>This is my coursework website</center></h1>
<p>
    This website will be used as my coursework submission for the Introduction to Mulitimedia web design assessment.<br>
    On this website, you will find a selection of information about me and examples of my work.<br>
    It takes elements of what I have learnt on the module up to this point in time and uses them in tandem to
    create a working website with functionality such as:
    <ul>
        <li>Basic HTML</li>
        <li>CSS formatting</li>
        <li>Embedding video and audio</li>
        <li>Javascript integration</li>
    </ul>
    <br>
    I hope you will enjoy my website.
</p>
```



This is my coursework website

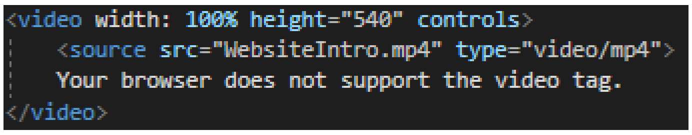
This website will be used as my coursework submission for the Introduction to Mulitimedia web design assessment.
On this website, you will find a selection of information about me and examples of my work.
It takes elements of what I have learnt on the module up to this point in time and uses them in tandem to create a working website with functionality such as:

- Basic HTML
- CSS formatting
- Embedding video and audio
- Javascript integration

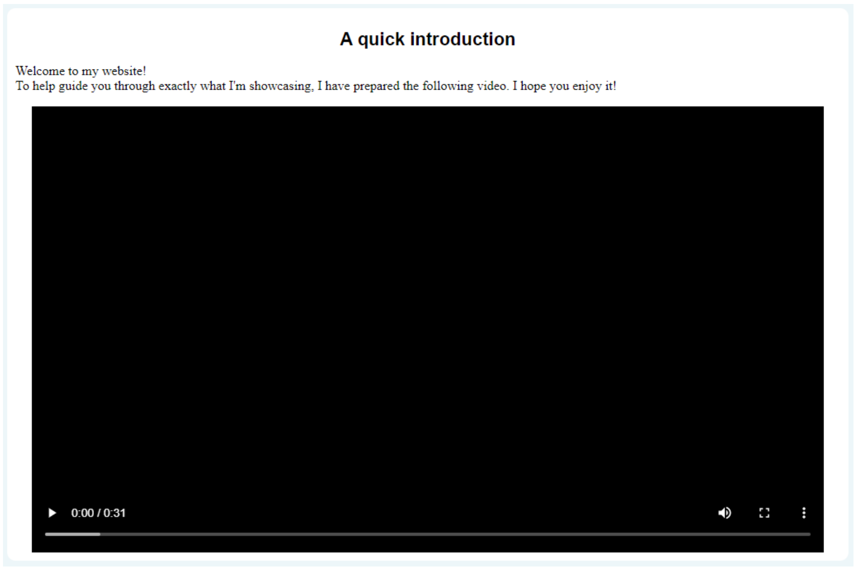
I hope you will enjoy my website.

Video

When it came to the video greeting page, I made use of the <video> tag to embed the video that I had uploaded to the directory into the webpage. Thankfully, HTML 5's video tag comes with inbuilt controls, so implementing functionality such as play, pause, mute, Fullscreen, etc was as simple as appending *controls* to the end of the video tag. I also included a message that will only be displayed in the browser if it doesn't support HTML 5's video tag.



```
<video width: 100% height="540" controls>
    <source src="WebsiteIntro.mp4" type="video/mp4">
    Your browser does not support the video tag.
</video>
```



A quick introduction

Welcome to my website!
To help guide you through exactly what I'm showcasing, I have prepared the following video. I hope you enjoy it!

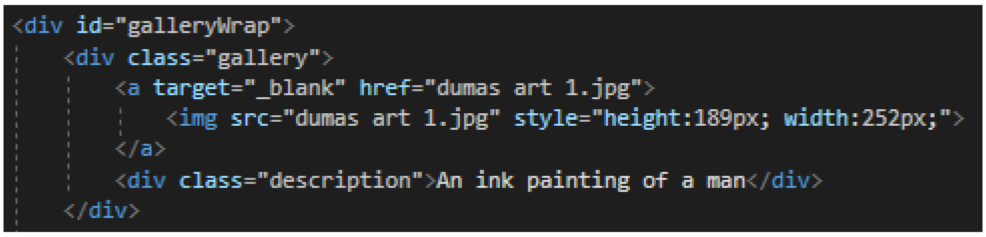
I decided to encode my videos in AVC (Advanced Video Coding aka MPEG-4 Part 10) as it is widely supported by all the popular browser choices, such as

- Google Chrome
- Firefox
- Internet Explorer
- Microsoft Edge
- Opera
- Etc.

Encoding the videos in this format meant that I did not have to create separate videos to be shown depending on the user's browser, thus saving a lot of time and effort. It is the most common format for video distribution and is used by 91% of industry developers as of September 2019.

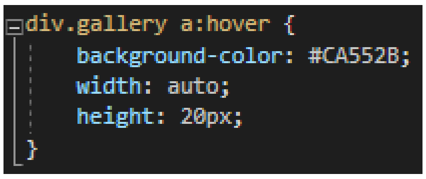
Image Gallery

On my hobbies page, I wanted to showcase three of my hobbies, being Art, Game Development and Video Editing. I started with art as it was going to be the easiest to implement, consisting mainly of an image tag with a link to lead to the full-size image when clicked. I wanted each picture to have more complex functionality, however, so I adapted some code from a W3Schools example [1].



```
<div id="galleryWrap">
    <div class="gallery">
        <a target="_blank" href="dumas art 1.jpg">
            <img src="dumas art 1.jpg" style="height:189px; width:252px;">
        </a>
        <div class="description">An ink painting of a man</div>
    </div>
```

Combining custom html classes with corresponding CSS, I created a gallery class to store each image and its corresponding elements. Each image had an <a> tag attached, so that I could easily edit its behaviour when selected. The CSS I wrote creates an orange underline at the bottom of the image whenever the mouse hovers over the image:

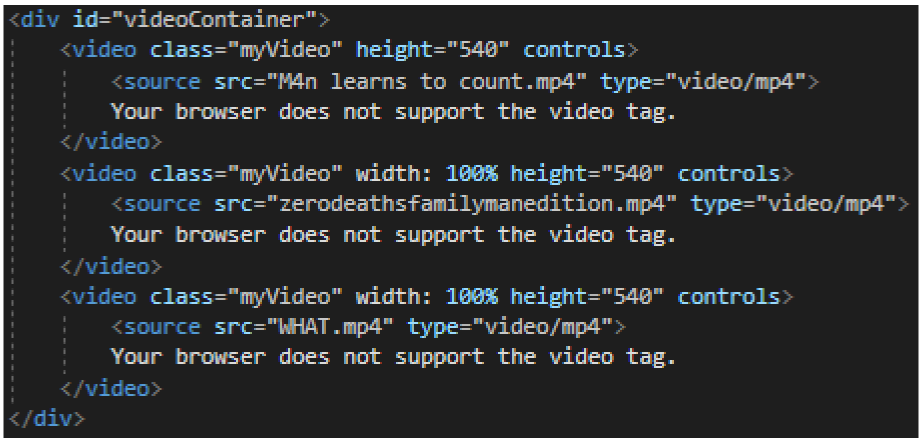


```
div.gallery a:hover {
    background-color: #CA552B;
    width: auto;
    height: 20px;
}
```

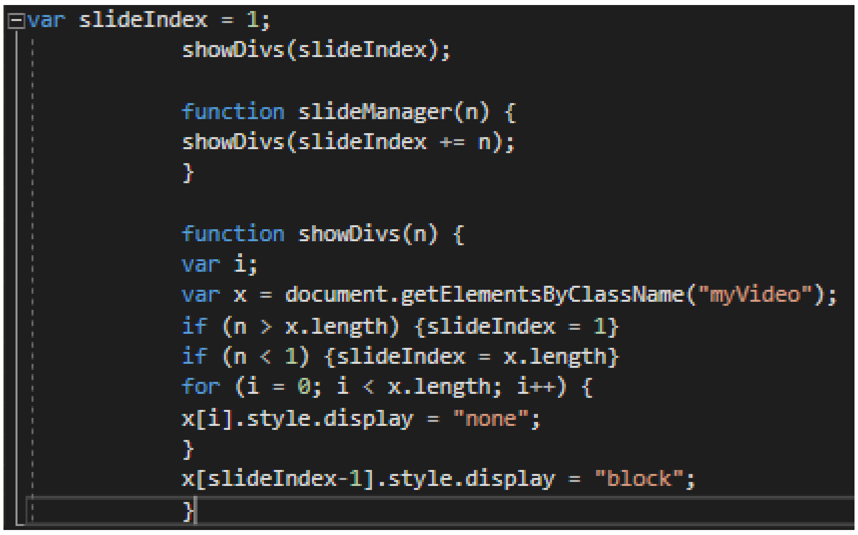
Each image has a preview, a link-through image and a description. Due to having each image contained within its own div, this allowed me to create visually pleasing, uniform image components. Finally, I created another div "galleryWrap", which contains all the other gallery divs, allowing for easy positioning of all the gallery divs as a collective.

Video Slideshow

Next, I wanted to work on implementing videos into my webpage. I initially did this by creating a video tag for each video and showing them all at the same time on the page, however, I thought this looked unprofessional, so I decided to implement a slideshow feature. Once again, this code is adapted from a W3Schools example [2] , with a few adjustments and changes:



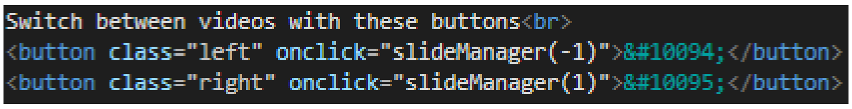
```html
<div id="videoContainer">
    <video class="myVideo" height="540" controls>
        <source src="M4n learns to count.mp4" type="video/mp4">
        Your browser does not support the video tag.
    </video>
    <video class="myVideo" width: 100% height="540" controls>
        <source src="zerodeathsfamilymanedition.mp4" type="video/mp4">
        Your browser does not support the video tag.
    </video>
    <video class="myVideo" width: 100% height="540" controls>
        <source src="WHAT.mp4" type="video/mp4">
        Your browser does not support the video tag.
    </video>
</div>
```



```javascript
var slideIndex = 1;
            showDivs(slideIndex);

            function slideManager(n) {
            showDivs(slideIndex += n);
            }

            function showDivs(n) {
            var i;
            var x = document.getElementsByClassName("myVideo");
            if (n > x.length) {slideIndex = 1}
            if (n < 1) {slideIndex = x.length}
            for (i = 0; i < x.length; i++) {
            x[i].style.display = "none";
            }
            x[slideIndex-1].style.display = "block";
            }
```

Since these videos are also in MPEG-4 Part 10 video formatting, they are supported by all major browsers, so no extra code was needed for different video types. Each video is contained within a video container class, where the current video is displayed. When the left and right arrow buttons are clicked, the variable i gets incremented or decremented accordingly. The corresponding video is then picked out of an array of every video on the page with the tag “myVideo” and the video in position i is shown.

The controls for the container are simply two buttons that run the slideManager function on click with a value of either 1 or -1 to shift the index.

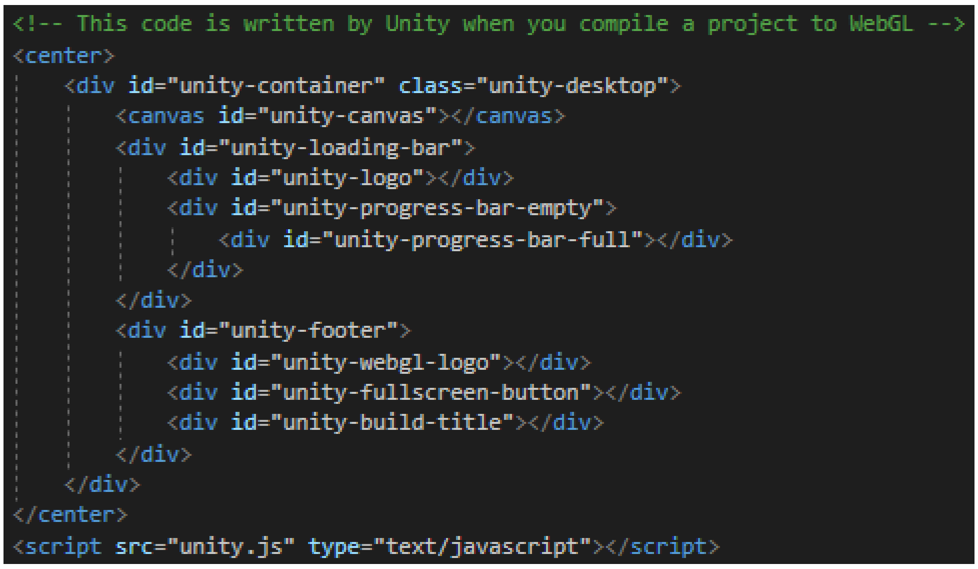


```html
Switch between videos with these buttons<br>
<button class="left" onclick="slideManager(-1)">&#10094;</button>
<button class="right" onclick="slideManager(1)">&#10095;</button>
```

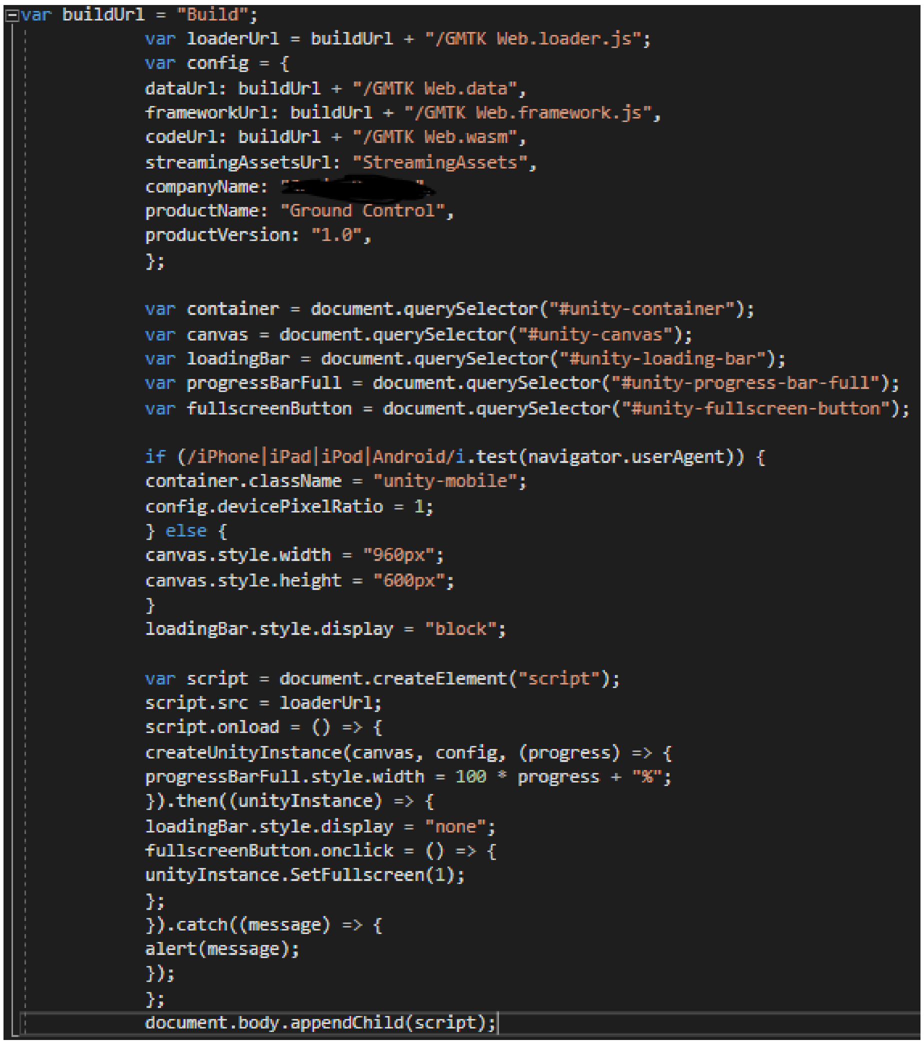
Embedding a game with Unity

The last thing I wanted to implement on my hobbies page was an example of my game development with Unity. I should make it abundantly clear that **I did not create the game on the website within the coursework period,** I made this game back in the summer as part of a 48-hour game jam. I did, however, have to adjust my game’s code to get it to compile after switching it to build for Unity’s WebGL package [3]. Otherwise, implementing my game onto my website was quick and easy.

When you build a unity project to WebGL, it creates a folder that contains an html webpage, the graphical assets for the web player and a few JavaScript codes to load the game onto the webpage. To implement this into my webpage, I simply took the relevant html code and placed it into my webpage's code, put the JavaScript code into its own file for efficiency (Unity creates the JavaScript as inline code into the html document) and put the web build folder into my Sussex directory.



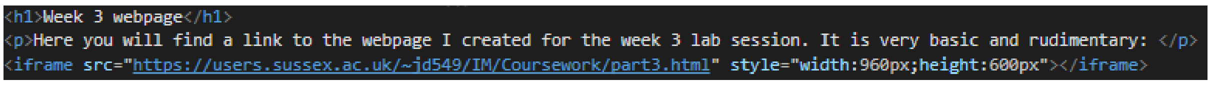
```html
<!-- This code is written by Unity when you compile a project to WebGL -->
<center>
    <div id="unity-container" class="unity-desktop">
        <canvas id="unity-canvas"></canvas>
        <div id="unity-loading-bar">
            <div id="unity-logo"></div>
            <div id="unity-progress-bar-empty">
                <div id="unity-progress-bar-full"></div>
            </div>
        </div>
        <div id="unity-footer">
            <div id="unity-webgl-logo"></div>
            <div id="unity-fullscreen-button"></div>
            <div id="unity-build-title"></div>
        </div>
    </div>
</center>
<script src="unity.js" type="text/javascript"></script>
```



```javascript
var buildUrl = "Build";
            var loaderUrl = buildUrl + "/GMTK Web.loader.js";
            var config = {
            dataUrl: buildUrl + "/GMTK Web.data",
            frameworkUrl: buildUrl + "/GMTK Web.framework.js",
            codeUrl: buildUrl + "/GMTK Web.wasm",
            streamingAssetsUrl: "StreamingAssets",
            companyName: [redacted]
            productName: "Ground Control",
            productVersion: "1.0",
            };

            var container = document.querySelector("#unity-container");
            var canvas = document.querySelector("#unity-canvas");
            var loadingBar = document.querySelector("#unity-loading-bar");
            var progressBarFull = document.querySelector("#unity-progress-bar-full");
            var fullscreenButton = document.querySelector("#unity-fullscreen-button");

            if (/iPhone|iPad|iPod|Android/i.test(navigator.userAgent)) {
            container.className = "unity-mobile";
            config.devicePixelRatio = 1;
            } else {
            canvas.style.width = "960px";
            canvas.style.height = "600px";
            }
            loadingBar.style.display = "block";

            var script = document.createElement("script");
            script.src = loaderUrl;
            script.onload = () => {
            createUnityInstance(canvas, config, (progress) => {
            progressBarFull.style.width = 100 * progress + "%";
            }).then((unityInstance) => {
            loadingBar.style.display = "none";
            fullscreenButton.onclick = () => {
            unityInstance.SetFullscreen(1);
            };
            }).catch((message) => {
            alert(message);
            });
            };
            document.body.appendChild(script);
```

Embedding another webpage with <iframe>

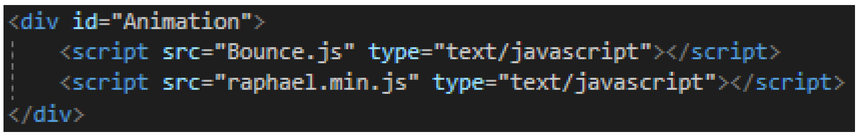
On the coursework page, I decided that I wanted more than just a link to my webpage that I created in the week 3 lab session, so I looked into other ways to display a webpage within a webpage. Eventually I stumbled across html's <iframe> tag, which allows you to display another webpage within a specified window:



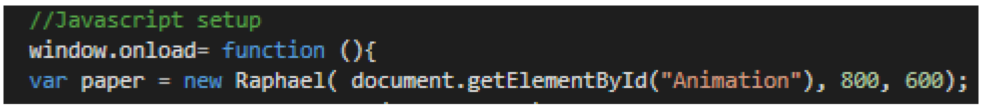
```
<h1>Week 3 webpage</h1>
<p>Here you will find a link to the webpage I created for the week 3 lab session. It is very basic and rudimentary: </p>
<iframe src="https://users.sussex.ac.uk/~jd549/IM/Coursework/part3.html" style="width:960px;height:600px"></iframe>
```

Embedding JavaScript animations

When including my animation from week 4, created a div container for the animation in the same way that I used for indexing the videos, so that I could go into the animation's JavaScript code and change the x and y co-ordinates for the paper component to be wherever the animation container was on the page, otherwise it would default to the top-left corner and be tricky to readjust if I moved components around the page:



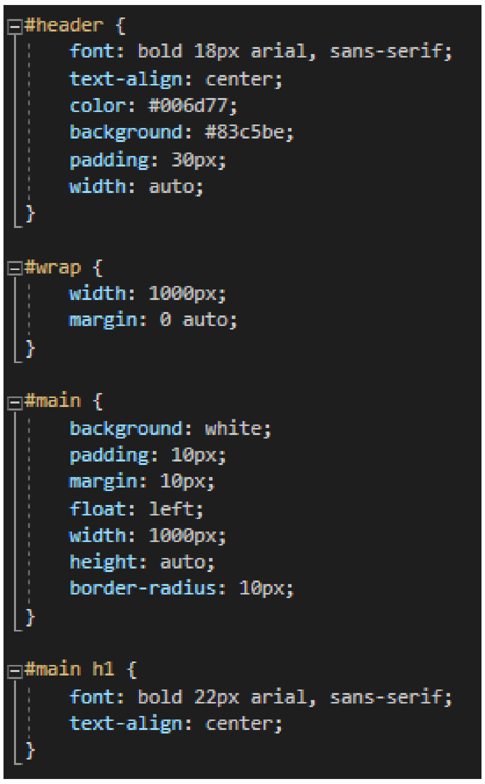
```
<div id="Animation">
    <script src="Bounce.js" type="text/javascript"></script>
    <script src="raphael.min.js" type="text/javascript"></script>
</div>
```



```
//Javascript setup
window.onload= function (){
var paper = new Raphael( document.getElementById("Animation"), 800, 600);
```

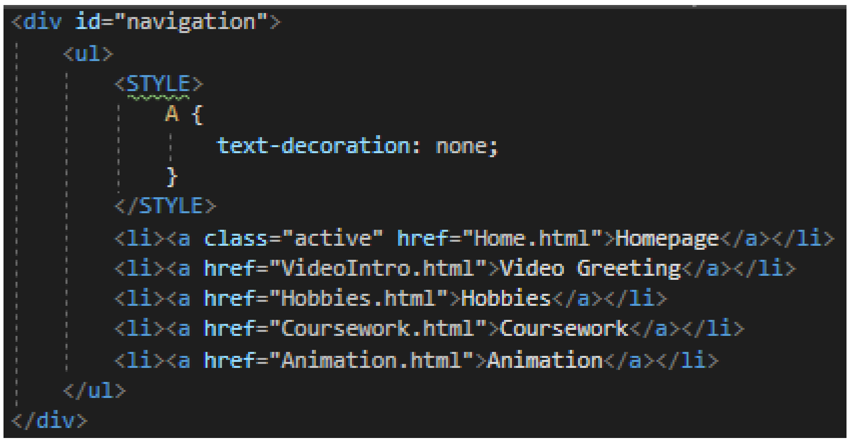
Additional functionality

When creating the webpage, I made sure that it would look the same in every browser by using CSS. I created a wrap for the main contents with automatic padding, so that it would stay in the centre of the screen, regardless of the resolution. To test this, you can take your browser out of Fullscreen and readjust the width.

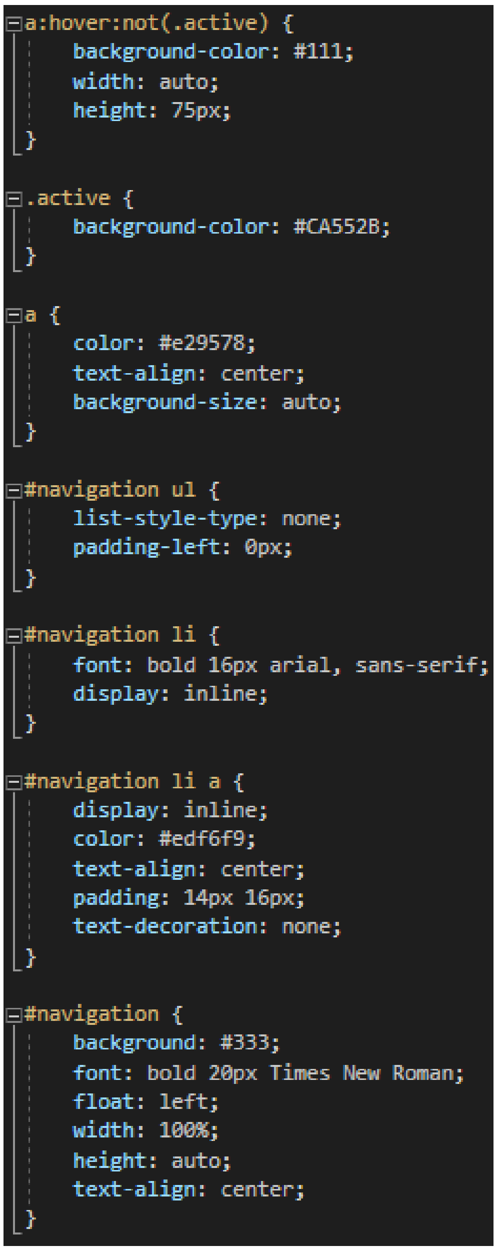


```
#header {
    font: bold 18px arial, sans-serif;
    text-align: center;
    color: #006d77;
    background: #83c5be;
    padding: 30px;
    width: auto;
}

#wrap {
    width: 1000px;
    margin: 0 auto;
}

#main {
    background: white;
    padding: 10px;
    margin: 10px;
    float: left;
    width: 1000px;
    height: auto;
    border-radius: 10px;
}

#main h1 {
    font: bold 22px arial, sans-serif;
    text-align: center;
}
```

I also created a navigation bar to help traverse the page, which is simply a list with CSS formatting. I made the current webpage be highlighted, which overrules the change of colour if you hover your mouse over the link.



```
<div id="navigation">
    <ul>
        <STYLE>
            A {
                text-decoration: none;
            }
        </STYLE>
        <li><a class="active" href="Home.html">Homepage</a></li>
        <li><a href="VideoIntro.html">Video Greeting</a></li>
        <li><a href="Hobbies.html">Hobbies</a></li>
        <li><a href="Coursework.html">Coursework</a></li>
        <li><a href="Animation.html">Animation</a></li>
    </ul>
</div>
```

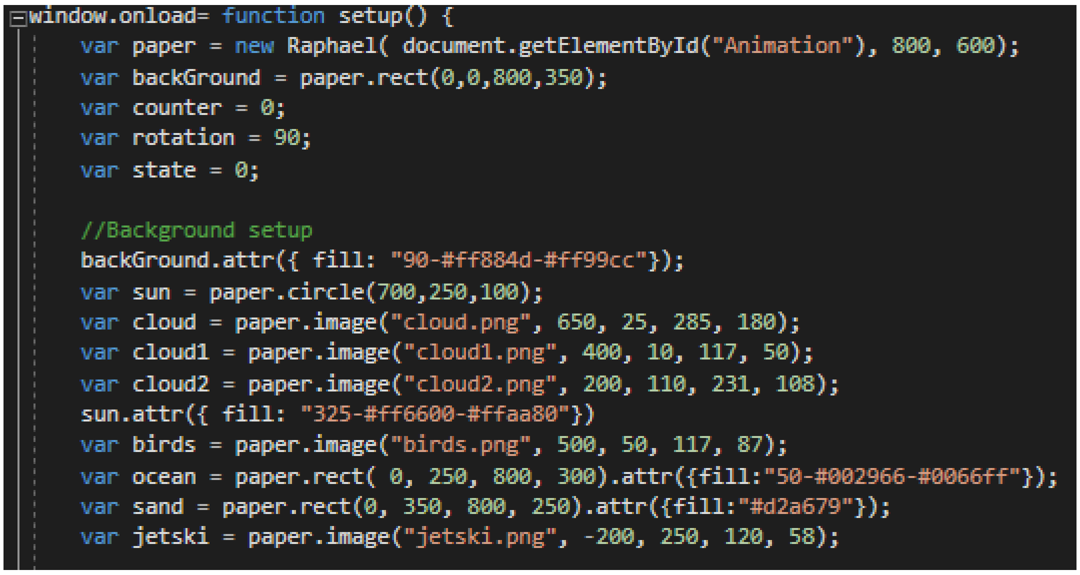


```
a:hover:not(.active) {
    background-color: #111;
    width: auto;
    height: 75px;
}

.active {
    background-color: #CA552B;
}

a {
    color: #e29578;
    text-align: center;
    background-size: auto;
}

#navigation ul {
    list-style-type: none;
    padding-left: 0px;
}

#navigation li {
    font: bold 16px arial, sans-serif;
    display: inline;
}

#navigation li a {
    display: inline;
    color: #edf6f9;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
}

#navigation {
    background: #333;
    font: bold 20px Times New Roman;
    float: left;
    width: 100%;
    height: auto;
    text-align: center;
}
```

Creating the JavaScript animation

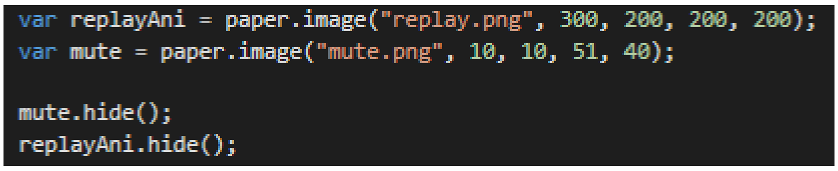
Scene setup

When creating the JavaScript animation, I used the Bounce.js code that I created in week 4 as a guide. With this, I created a Raphael [4] paper object with the same dimensions of 800x600 pixels. Having learnt from implementing the Bounce.js code into the coursework page, I used the same document.getElementById solution to position the animation where I wanted on the page. I also created some variables that allowed for tracking in my animation, such as the counter variable. For certain objects, I used attribute tags to fill the object in, usually with a gradient colour.



```
window.onload= function setup() {
    var paper = new Raphael( document.getElementById("Animation"), 800, 600);
    var backGround = paper.rect(0,0,800,350);
    var counter = 0;
    var rotation = 90;
    var state = 0;

    //Background setup
    backGround.attr({ fill: "90-#ff884d-#ff99cc"});
    var sun = paper.circle(700,250,100);
    var cloud = paper.image("cloud.png", 650, 25, 285, 180);
    var cloud1 = paper.image("cloud1.png", 400, 10, 117, 50);
    var cloud2 = paper.image("cloud2.png", 200, 110, 231, 108);
    sun.attr({ fill: "325-#ff6600-#ffaa80"})
    var birds = paper.image("birds.png", 500, 50, 117, 87);
    var ocean = paper.rect( 0, 250, 800, 300).attr({fill:"50-#002966-#0066ff"});
    var sand = paper.rect(0, 350, 800, 250).attr({fill:"#d2a679"});
    var jetski = paper.image("jetski.png", -200, 250, 120, 58);
```

I used similar functions to instantiate the rest of the scene, with some element being moved off screen so that they could move across at a later point and others being hidden with the object.hide() function, ready to be shown at the right time with the object.show() function.

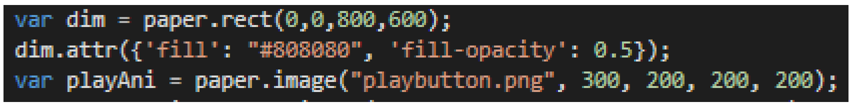


```
var replayAni = paper.image("replay.png", 300, 200, 200, 200);
var mute = paper.image("mute.png", 10, 10, 51, 40);

mute.hide();
replayAni.hide();
```

Implementing interactivity via a Play button

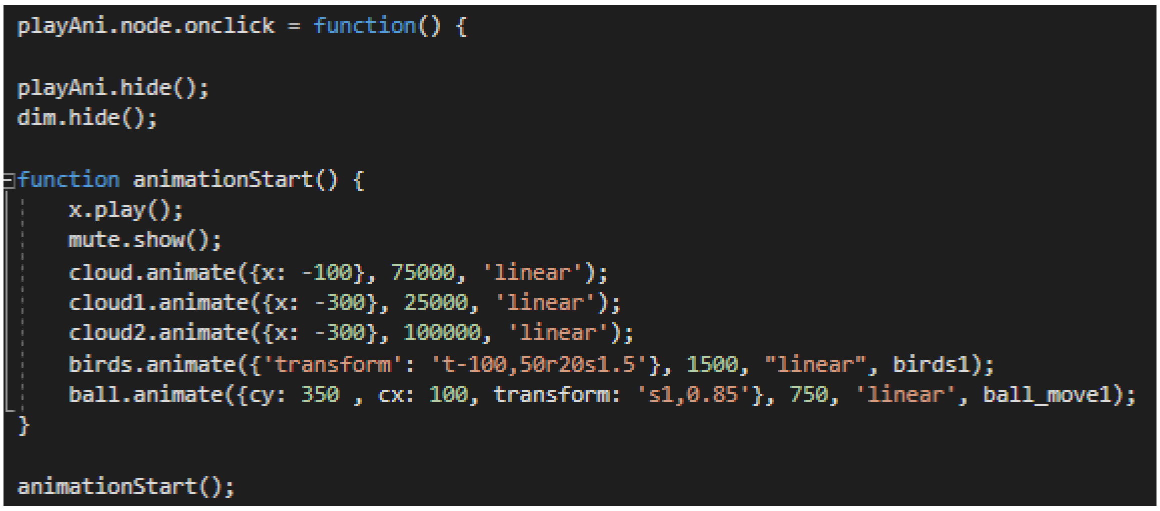
When I initially made my animation, it would just load and play as soon as the window was loaded on the browser. I decided I wanted to give the user some agency over the animation with the use of a play button as I found it a little bit annoying when the animation would start without me being able to read the rest of the page first.

To do this, I created a Raphael image object and positioned it in the centre of the screen, as well as a transparent grey overlay so the user would realise that the animation wasn't active.



```
var dim = paper.rect(0,0,800,600);
dim.attr({'fill': "#808080", 'fill-opacity': 0.5});
var playAni = paper.image("playbutton.png", 300, 200, 200, 200);
```

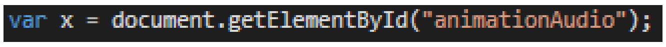
To give the button functionality, I added a node.onclick function to the playAni object which would hide both the playAni and dim objects and run a function that would trigger the animation.



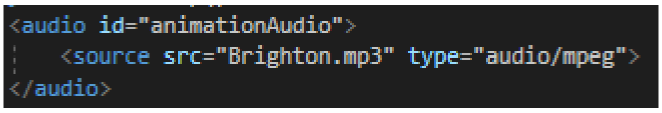
```
playAni.node.onclick = function() {

playAni.hide();
dim.hide();

function animationStart() {
    x.play();
    mute.show();
    cloud.animate({x: -100}, 75000, 'linear');
    cloud1.animate({x: -300}, 25000, 'linear');
    cloud2.animate({x: -300}, 100000, 'linear');
    birds.animate({'transform': 't-100,50r20s1.5'}, 1500, "linear", birds1);
    ball.animate({cy: 350 , cx: 100, transform: 's1,0.85'}, 750, 'linear', ball_move1);
}

animationStart();
```

Implementing sound

You may have noticed the x.play(); line in my animationStart function, this is used to play an audio file on the function. I created a variable x which I assigned to get an object in the html with the tag "animationAudio". Since I had set up an audio source with the same tag in my animation.html document, this refers to this audio file embedded within the webpage.



```
var x = document.getElementById("animationAudio");
```

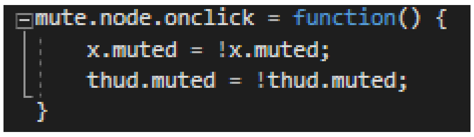


```
<audio id="animationAudio">
    <source src="Brighton.mp3" type="audio/mpeg">
</audio>
```

I did the same later in the animation with an audio file that I used for a sound effect when the ball gets squashed by a beach hut, set up in the same way.

Controlling sound

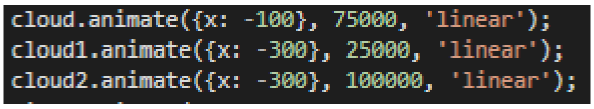
The next logical step for this was to allow the user to control the sound. I decided to go with a mute button, since it was considerably simpler to set up compared to an audio slider. Like with the play button, I made this a clickable object that executed a function with the onclick function. Since JavaScript has an in-built muted variable which returns a Boolean value, the code for this is to simply set the .muted variable to the opposite of its current state:



```
mute.node.onclick = function() {
    x.muted = !x.muted;
    thud.muted = !thud.muted;
}
```

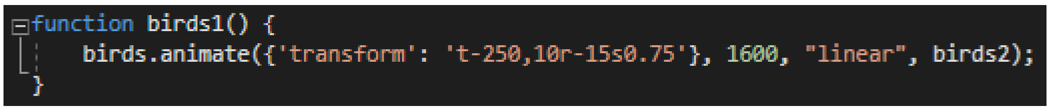
Creating movement

When moving the animation objects across the screen, I used either one of two methods. The first was to use the animate function with an end position. I used this method primarily for objects that had linear movements, where they would move in a straight line across one axis:



```
cloud.animate({x: -100}, 75000, 'linear');
cloud1.animate({x: -300}, 25000, 'linear');
cloud2.animate({x: -300}, 100000, 'linear');
```

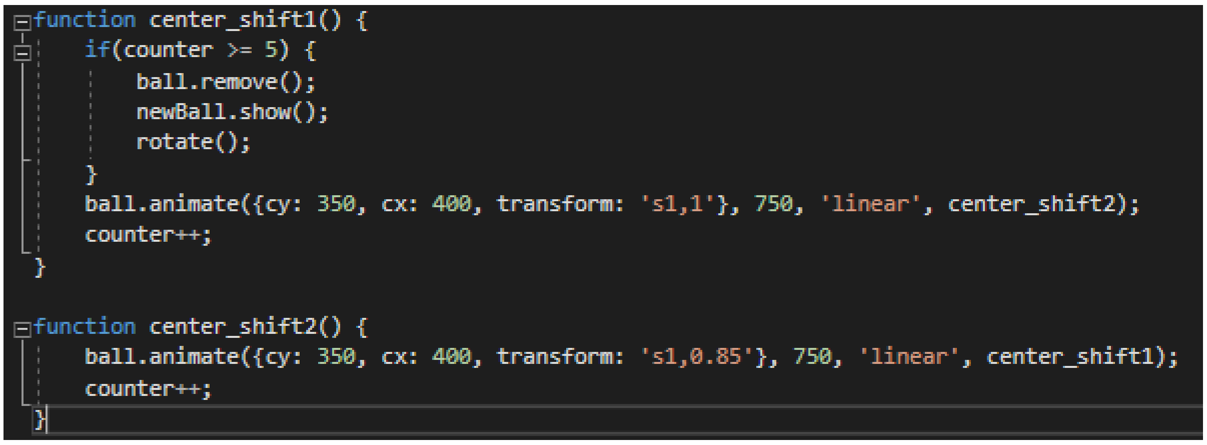
The second method was to use the animate function in conjunction with the transform function. This allowed for more complex changes in the end object, including rotation and scaling. This took a (somewhat difficult to understand) string as instructions for how to manipulate a vector object, with the end of the animate function remaining the same, taking the duration, easing method and follow-up function.



```
function birds1() {
    birds.animate({'transform': 't-250,10r-15s0.75'}, 1600, "linear", birds2);
}
```

Changing objects

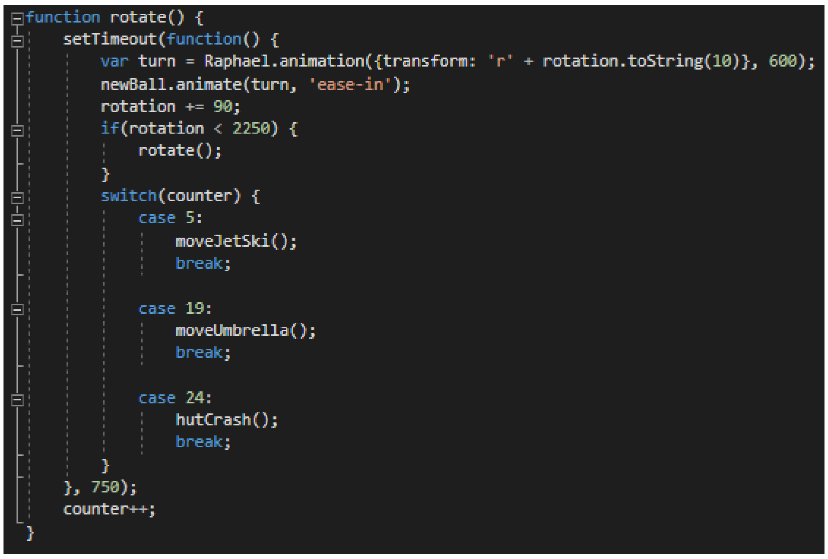
In my animation, I wanted to use each of the three main vector objects on the screen to showcase different vector transformations. With the circle object, I wanted to showcase scaling of objects. I decided to scale it over a single axis, giving the illusion of the ball bobbing along. I achieved this with alternating functions that changed the y-axis scaling:



```
function center_shift1() {
    if(counter >= 5) {
        ball.remove();
        newBall.show();
        rotate();
    }
    ball.animate({cy: 350, cx: 400, transform: 's1,1'}, 750, 'linear', center_shift2);
    counter++;
}

function center_shift2() {
    ball.animate({cy: 350, cx: 400, transform: 's1,0.85'}, 750, 'linear', center_shift1);
    counter++;
}
```

I made use of a counter variable to perform the animation a certain number of times until the object is obscured by the beach hut, at which point the new object will be shown and its function will be run.

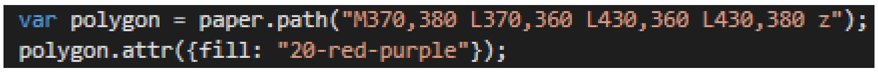
With the square object, I wanted to showcase the transform function's rotate component. When initially creating this code, I found that running it would execute every rotation up to the limit within a single animate function. To bypass this, I had to utilise JavaScript's setTimeout function and have the function call itself, reiterating over the loop after a set interval.



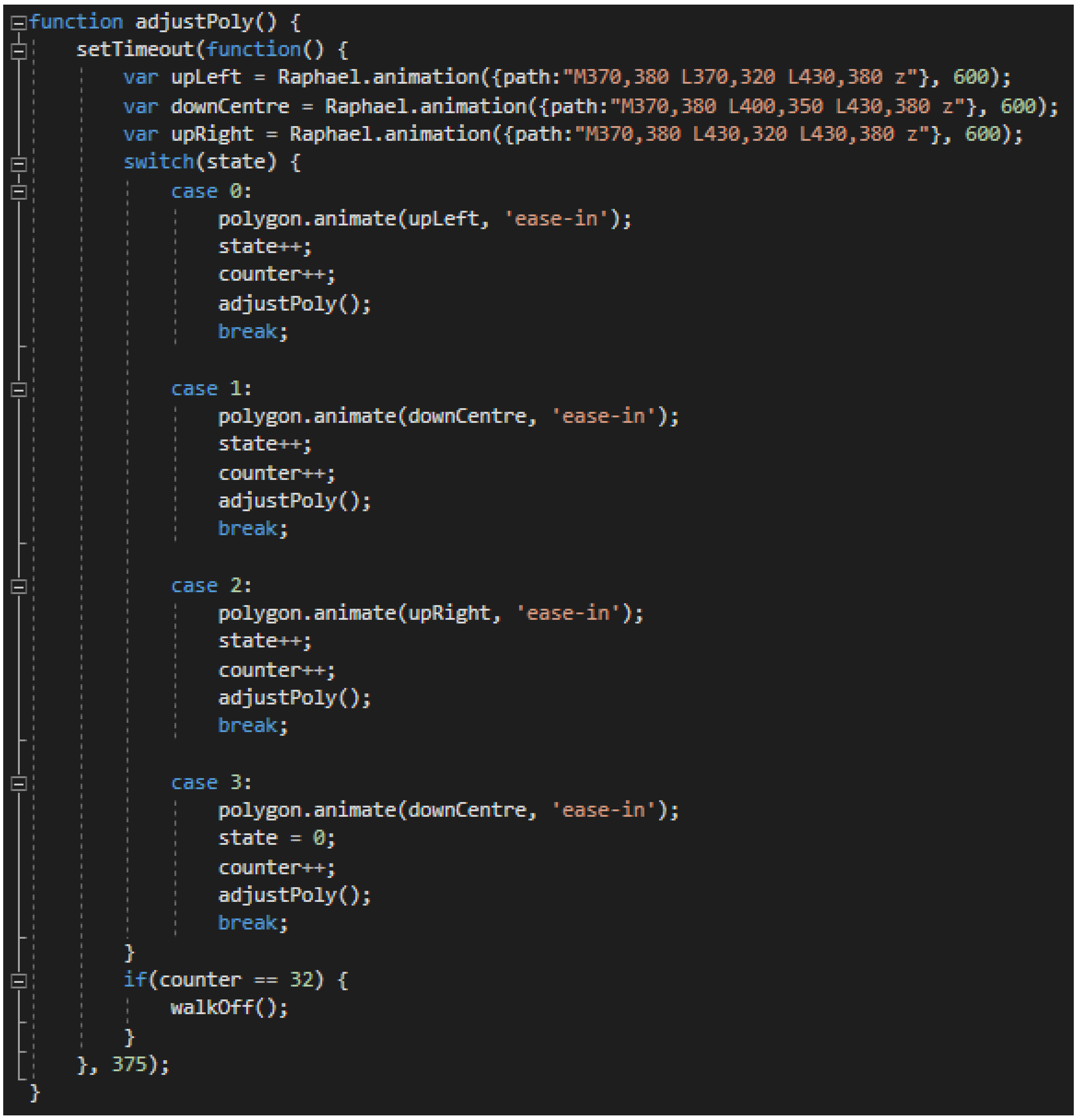
```
function rotate() {
    setTimeout(function() {
        var turn = Raphael.animation({transform: 'r' + rotation.toString(10)}, 600);
        newBall.animate(turn, 'ease-in');
        rotation += 90;
        if(rotation < 2250) {
            rotate();
        }
        switch(counter) {
            case 5:
                moveJetSki();
                break;

            case 19:
                moveUmbrella();
                break;

            case 24:
                hutCrash();
                break;
        }
    }, 750);
    counter++;
}
```

I also used a counter variable to move different objects at different times with a switch statement, allowing an exit from the loop.

Finally, I wanted to showcase some dynamic shape changing. To do this, I created a custom vector object using the path function. With the path function, you have to give information on every vertex for the object, which allows you to create complex shapes. This allowed me to create a squashed square in reaction to being crushed by the beach hut, which then morphed into a triangle and looped through changing the upper vertex. To switch through these animations and loop, I once again used a switch statement to manage it, with a counter to break out of the function and end the animation.



```
var polygon = paper.path("M370,380 L370,360 L430,360 L430,380 z");
polygon.attr({fill: "20-red-purple"});
```



```
function adjustPoly() {
    setTimeout(function() {
        var upLeft = Raphael.animation({path:"M370,380 L370,320 L430,380 z"}, 600);
        var downCentre = Raphael.animation({path:"M370,380 L400,350 L430,380 z"}, 600);
        var upRight = Raphael.animation({path:"M370,380 L430,320 L430,380 z"}, 600);
        switch(state) {
            case 0:
                polygon.animate(upLeft, 'ease-in');
                state++;
                counter++;
                adjustPoly();
                break;

            case 1:
                polygon.animate(downCentre, 'ease-in');
                state++;
                counter++;
                adjustPoly();
                break;

            case 2:
                polygon.animate(upRight, 'ease-in');
                state++;
                counter++;
                adjustPoly();
                break;

            case 3:
                polygon.animate(downCentre, 'ease-in');
                state = 0;
                counter++;
                adjustPoly();
                break;
        }
        if(counter == 32) {
            walkOff();
        }
    }, 375);
}
```

References

[1] – W3schools.com. 2020. *CSS Image Gallery*. [online] Available at: <https://www.w3schools.com/css/css_image_gallery.asp> [Accessed 3 November 2020].

[2] - W3schools.com. 2020. *How To Create A Slideshow*. [online] Available at: <https://www.w3schools.com/howto/howto_js_slideshow.asp> [Accessed 12 November 2020].

[3] - Technologies, U., 2020. *Unity - Manual: Getting Started With Webgl Development*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/Manual/webgl-gettingstarted.html> [Accessed 15 November 2020].

[4] - Baranovskiy, D., 2020. *Raphaël—Javascript Library*. [online] Dmitrybaranovskiy.github.io. Available at: <https://dmitrybaranovskiy.github.io/raphael/> [Accessed throughout].