Games Development in JavaScript

Derek O'Reilly, Dundalk Instute of Technology, Ireland

Abstract

The course prepare participants to create games and animations in JavaScript. In the materials students can learn how to create a browser game, what is an animation and how to create one. The course uses code examples that can be reused by students and on that base the participants can create new games.

The original version of this material can be found on project web page: `http://geniusgamedev.eu/teaching.php`.

Type: E-learning module, online-training, MOOC

Domain: Software development

Requirements: Basic knowledge in JavaScript programming

Target group: Computer Science students

License

Disclaimer

HTML Canvas

The HTML <canvas> provides a rectangular area on a webpage on which graphics can be drawn. Canvas is used with javascript to create animated browser games.

Every canvas needs to have an id assigned to it. This will be used in the javascript code to access the canvas.

```
<canvas id="gameCanvas"></canvas>
```

Use the javascript getElementById() function to access the canvas id within the javascript code.

```
<script>
let canvas = document.getElementById("gameCanvas");
</script>
```

The physical width and height of the canvas element are set in the stylesheet. These values define how much physical space the canvas occupies on the webpage.

```
<style>
#gameCanvas
{
  outline:1px solid darkgrey;
  width:500px;
  height:500px;
}
</style>
```

The logical width and height of the canvas are defined using javascript. These are the values that are will be used in the game. If the canvas size does not match the style size, then the canvas drawing will be scaled. It is usually a good idea to ensure that the physical and logical sizes match. The physical size of the canvas can be got using the clientWidth and clientHeight properties of the javascript canvas. In these notes, the logical width and height will always be set to match the physical width and height, as shown below.

```
<script>
let canvas = document.getElementById("gameCanvas");
canvas.width = canvas.clientWidth;
canvas.height = canvas.clientHeight;
</script>
```

In order to draw onto the canvas, we need to assign a graphics context to it. This is done in javascript code using the getContext("2d") function. The code is implemented in the Game object constructor.

```
let ctx = canvas.getContext("2d");
```

Example of a minimal canvas, which draws a red square (Run Example)

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <style>
            #gameCanvas
            {
                /* the canvas styling usually does not change */
                outline:1px solid darkgrey;
                width:500px;
                height:500px;
            }
        </style>
    </head>

    <body>
        <canvas id = "gameCanvas"></canvas>

        <script>
            let canvas = document.getElementById("gameCanvas");
            canvas.width = canvas.clientWidth;
            canvas.height = canvas.clientHeight;
            let ctx = canvas.getContext("2d");

            // draw something on the canvas
            ctx.fillStyle = "red";
            ctx.fillRect(100, 100, 300, 300);
        </script>
    </body>
</html>
```

Exercises

Write code to draw four squares on the canvas, as shown here.

Text

We can draw text or outlined (stroke) text on a canvas using the code below:

```
ctx.fillStyle = "red";
ctx.font = "100px Times Roman";
ctx.fillText("Some Text", 10, 150);

ctx.strokeStyle = "blue";
ctx.font = "50px Arial";
ctx.lineWidth = 3;
ctx.strokeText("Some Text Outline", 10, 350);
```

Example of text on a canvas (Run Example)

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <link rel="shortcut icon" type="image/png" href="images/genius_icon.png"/>
        <style>
            #gameCanvas
            {
                /* the canvas styling usually does not change */
                outline:1px solid darkgrey;
                width:500px;
                height:500px;
            }
        </style>
    </head>

    <body>
        <canvas id = "gameCanvas"></canvas>

        <script>
            let canvas = document.getElementById("gameCanvas");
            canvas.width = canvas.clientWidth;
            canvas.height = canvas.clientHeight;
            let ctx = canvas.getContext("2d");

            // draw text on a canvas
            ctx.fillStyle = "red";
            ctx.font = "100px Times Roman";
            ctx.fillText("Some Text", 10, 150);
```

```
            ctx.strokeStyle = "blue";
            ctx.font = "50px Arial";
            ctx.lineWidth = 3;
            ctx.strokeText("Some Text Outline", 10, 350);
        </script>
    </body>
</html>
```

Exercises

Write code to dislay a message with an outline, as shown here.

Items will be draw on the canvas in the order that they are listed in the code. Any drawing outside of a canvas's logical area will be clipped.

Example of clipped text on a canvas (Run Example)

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <link rel="shortcut icon" type="image/png" href="images/genius_icon.png"/>
        <style>
            #gameCanvas
            {
                /* the canvas styling usually does not change */
                outline:1px solid darkgrey;
                width:500px;
                height:500px;
            }
        </style>
    </head>

    <body>
        <canvas id = "gameCanvas"></canvas>

        <script>
            let canvas = document.getElementById("gameCanvas");
            canvas.width = canvas.clientWidth;
            canvas.height = canvas.clientHeight;
            let ctx = canvas.getContext("2d");

            // draw some text on the canvas
            ctx.fillStyle = "red";
            ctx.font = "120px Times Roman";
            ctx.fillText("Clipped Text", 10, 150);
        </script>
    </body>
</html>
```

Images

We can draw an image onto a canvas, as shown below:

```
let img = new Image();
img.src = "images/dkit01.png";


window.onload = function ()
{
    ctx.drawImage(img, 100, 100, 300, 300);
}
```

This will draw the image at position, 100, 100 with a width and height of 300. An image will only be drawn on a canvas if the image has been loaded into the webpage. For this reason, we should always wrap the canvas code inside a "window.onload()" function. The image must be declared outside of the window.onload() function. That way, the window.onload() function will only be called after the image has loaded. This is shown in the example below.

Example of a canvas image (Run Example)

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <style>
            #gameCanvas
            {
                /* the canvas styling usually does not change */
                outline:1px solid darkgrey;
                width:500px;
                height:500px;
            }
        </style>

        <script>
            let img = new Image(); // note that the offscreen image must be declared OUTSIDE of the window.onload() func
            img.src = "images/city.png";

            window.onload = function ()
            {
                let canvas = document.getElementById("gameCanvas");
                canvas.width = canvas.clientWidth;
                canvas.height = canvas.clientHeight;
                let ctx = canvas.getContext("2d");

                // draw an image on the canvas
                ctx.drawImage(img, 100, 100, 300, 300);
            };
        </script>
    </head>

    <body>
        <canvas id = "gameCanvas"></canvas>
    </body>
</html>
```

Exercises

Write code to draw a centred image and four corner images on the canvas, as shown here.

Rotations

Rotations are made around the origin. In order to rotate around any other point, we translate that point to the origin, perform the rotation and translate back to the point's original position again, as shown below.

```
ctx.translate(canvas.width / 2, canvas.height / 2);
ctx.rotate(radiants);
ctx.translate(-canvas.width / 2, -canvas.height / 2);
```

Rotations are performed in radiants. We can convert from user friendly degrees to radiants using the formula below:

```
radiants = degrees * Math.PI / 180;
```

This can be achieved using the function below:

```
/* Convert from degrees to radians */
Math.radians = function (degrees)
{
    return degrees * Math.PI / 180;
};
```

By default, all drawing on the canvas after the rotation will be affected by the rotation. Normally, we do not want to do this.
Use the save() and restore() functions to limit the scope of a rotation, as shown below:

```
ctx.save();
ctx.translate(canvas.width / 2, canvas.height / 2);
ctx.rotate(Math.radians(90));
ctx.translate(-canvas.width / 2, -canvas.height / 2);

// anything drawn here will be rotated

ctx.restore();
```

Example showing text rotation (Run Example)

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <style>
            #gameCanvas
            {
                /* the canvas styling usually does not change */
```

```
                outline:1px solid darkgrey;
                width:500px;
                height:500px;
            }
        </style>

        <script>
            /* Convert from degrees to radians */
            Math.radians = function (degrees)
            {
                return degrees * Math.PI / 180;
            };
        </script>
    </head>

    <body>
        <canvas id = "gameCanvas"></canvas>

        <script>
            let canvas = document.getElementById("gameCanvas");
            canvas.width = canvas.clientWidth;
            canvas.height = canvas.clientHeight;
            let ctx = canvas.getContext("2d");
            ctx.fillStyle = "black";
            ctx.font = "50px Times Roman";

            ctx.fillText("Normal Text", 100, 50);

            ctx.save();
            ctx.translate(canvas.width / 2, canvas.height / 2);
            ctx.rotate(Math.radians(45));
            ctx.translate(-canvas.width / 2, -canvas.height / 2);

            ctx.fillText("Diagonal Text", 100, 250);
            ctx.restore();
            ctx.fillText("Normal Again", 100, 450);
        </script>
    </body>
</html>
```

Code Explained

Everything inside the save()/restore() functions will be included in the rotaiton.

Exercises

Write code to rotate an image by 90 degrees, as shown here. Remember that you need to ensure that the image is loaded before you attempt to draw it.

Offscreen Canvas

An offscreen canvas is a canvas that is not visible on the screen. It can be used to construct the contents of a canvas prior to displaying the contents.

An offscreen canvas is just a canvas element. To create a canvas element we use the code below:

```
var offscreenCanvas = document.createElement('canvas');
```

As with any other canvas, we can associate a 2d graphic context with an offscreen canvas, as shown in the code below

```
var offscreenCanvasCtx = offscreenCanvas.getContext('2d');
```

We need to ensure that the offscreen canvas is the exact same size as the on-screen canvas that it is being associated with, as shown below:

```
offscreenCanvas.width = canvas.width;
offscreenCanvas.height = canvas.height;
```

To draw an offscreen canvas onto another canvas, pass the offscreen canvas name as the element to be drawn, as shown below:

```
ctx.drawImage(offscreenCanvas,0, 0, width, height);  // draw the offscreen canvas and not the graphic context
```

Example of an offscreen canvas (Run Example)

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <style>
            #gameCanvas
            {
                /* the canvas styling usually does not change */
                outline:1px solid darkgrey;
                width:500px;
                height:500px;
            }
        </style>
    </head>

    <body>
        <canvas id = 'gameCanvas'></canvas>

        <script>
            let cityImage = new Image();
            cityImage.src = "images/city.png";

            let beachImage = new Image();
            beachImage.src = "images/beach.png";

            window.onload = function ()
            {
                let canvas = document.getElementById("gameCanvas");
                canvas.width = canvas.clientWidth;
                canvas.height = canvas.clientHeight;
                let ctx = canvas.getContext("2d");

                offscreenCanvas = document.createElement('canvas');
                offscreenCanvasCtx = offscreenCanvas.getContext('2d');
                offscreenCanvas.width = canvas.width;
                offscreenCanvas.height = canvas.height;

                // draw onto the offscreen canvas
                offscreenCanvasCtx.drawImage(beachImage, 0, 0, canvas.width, canvas.height);

                // draw an image directly onto the screen's canvas
                ctx.drawImage(cityImage, 0, 0, canvas.width, canvas.height);
```

```
                // draw the offscreen buffer onto the screen's canvas
                ctx.drawImage(offscreenCanvas, 0, 0, 200, 200);
            };
        </script>
    </body>
</html>
```

Code Explained

```
                offscreenCanvas = document.createElement('canvas');
                offscreenCanvasCtx = offscreenCanvas.getContext('2d');
                offscreenCanvas.width = canvas.width;
                offscreenCanvas.height = canvas.height;
```

The code above creates an offscreen canvas and a graphics context up the offscreen canvas. It does not have to be the same size as the HTML canvas, but it usually makes sense to match the size.

```
                // draw onto the offscreen canvas
                offscreenCanvasCtx.drawImage(beachImage, 0, 0, CANVAS_WIDTH, CANVAS_HEIGHT);
```

Drawing on an offscreen canvas is done in exactly the same way as for a HTML canvas.

```
                // draw the offscreen buffer onto the screen's canvas
                ctx.drawImage(offscreenCanvas, 0, 0, 200, 200);
```

Any canvas can be drawn onto another canvas using the drawImage() funciton. In this example, the offscreen canvas is drawn onto the HTML canvas.

Pixels

It is possible to read and write to individual pixels on a canvas.

The getImageData() function allows us to read rectangular areas from a canvas.

```
imageData = ctx.getImageData(x, y, width, height);
```

Example of reading and writting pixels on a canvas (Run Example).

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <style>
            #gameCanvas
            {
                /* the canvas styling usually does not change */
                outline:1px solid darkgrey;
                width:500px;
                height:500px;
            }
        </style>
    </head>

    <body>
        <canvas id = "gameCanvas"></canvas>

        <script>
            let img = new Image(); // note that the offscreen image must be declared OUTSIDE of the window.onload() func
            img.src = "images/city.png";

            window.onload = function ()
            {
                let canvas = document.getElementById("gameCanvas");
                canvas.width = canvas.clientWidth;
                canvas.height = canvas.clientHeight;
                let ctx = canvas.getContext("2d");

                // draw an image on the canvas
                ctx.drawImage(img, 0, 0, canvas.width, canvas.height);

                // get the pixels from the canvas
                // NOTE: getImageData() will only work if the image in drawImage is
                // on the same server as the webpage

                let imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
                let data = imageData.data;

                const RED = 0;
                const GREEN = 1;
                const BLUE = 2;
                const ALPHA = 3;

                for (var i = 0; i < data.length; i += 4)
                {
                    data[i + RED] = 255 - data[i + 0];
                    data[i + GREEN] = 255 - data[i + 1];
                    data[i + BLUE] = 255 - data[i + 2];
                    data[i + ALPHA] = 255;
                }

                ctx.putImageData(imageData, 0, 0);
            };
        </script>
    </body>
</html>
```

Code Explained

```
                let imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
                let data = imageData.data;
```

The above two lines of code above copy the selected pixels into a one-dimensional array.

```
const RED = 0;
const GREEN = 1;
const BLUE = 2;
const ALPHA = 3;

for (var i = 0; i < data.length; i += 4)
{
    data[i + RED] = 255 - data[i + 0];
    data[i + GREEN] = 255 - data[i + 1];
    data[i + BLUE] = 255 - data[i + 2];
    data[i + ALPHA] = 255;
}
```

The for-loop above steps through each of the selected pixels. Each pixel has a red, green, blue and alpha value, each of which is in the range 0-255.

```
ctx.putImageData(imageData, 0, 0);
```

The putImageData() function allows us to write the selected pixels on the canvas.

Exercises

Write code to place coloured text on a grayscale image, as shown here.

Read/Write Part Of A Canvas

The getImageData() and putImageData() functions allow us to read/write onto any rectangular area of a canvas, as shown here.

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <style>
            #gameCanvas
            {
                /* the canvas styling usually does not change */
                outline:1px solid darkgrey;
                width:500px;
                height:500px;
            }
        </style>
    </head>

    <body>
        <canvas id = "gameCanvas"></canvas>

        <script>
            let image = new Image();
            image.src = "images/city.png";

            window.onload = function ()
            {
                let canvas = document.getElementById("gameCanvas");
                canvas.width = canvas.clientWidth;
                canvas.height = canvas.clientHeight;
                let ctx = canvas.getContext("2d");

                ctx.drawImage(image, 0, 0, canvas.width, canvas.height);

                // get the pixels from the canvas
                // NOTE: getImageData() will only work if the image in drawImage is
                // on the same server as the webpage
                let imageData = ctx.getImageData(canvas.width / 3, 0, canvas.width / 3, canvas.height); // x,y position
                let data = imageData.data;
```

```
                const RED = 0;
                const GREEN = 1;
                const BLUE = 2;
                const ALPHA = 3;

                // Manipulate the pixel data
                for (var i = 0; i < data.length; i += 4)
                {
                    imageData.data[i + RED] = 255 - imageData.data[i + RED];
                    imageData.data[i + GREEN] = 255 - imageData.data[i + GREEN];
                    imageData.data[i + BLUE] = 255 - imageData.data[i + BLUE];
                    imageData.data[i + ALPHA] = 255;
                }

                ctx.putImageData(imageData, canvas.width / 3, 0);
            };
        </script>
    </body>
</html>
```

Code Explained

```
imageData = ctx.getImageData(canvas.width / 3, 0, canvas.width / 3, canvas.height); // x,y position and regtangle width
```

The code above shows that we only manipulate the middle one-third of the canvas.

Exercises

Write code to display a greyscaled image with a coloured border, as shown here.

Accessing One Pixel

We can read/write any single canvas pixel by selecting a rectangle of size 1 pixel. The code below reads a single pixel.

```
var imageData = someOffscreenCanvas.getImageData(x, y, 1, 1);
var data = imageData.data;

if (data[ALPHA] !== 0)
{
    // This pixel is not transparent
}
```

We can use an offscreen canvas to test the mouse pointer against the transparent and non-transparent parts of an image. This is especially useful for accurate collision detection. We shall look at this in more detail later.

Canvas Timers

Timers can be used to call a javascript function at set intervals. There are two ways to call a timer function:

setTimeout(functionName, milliseconds)
This will cause a function to be called once. The function will be called after waiting a specified number of milliseconds.

setInterval(functionName, milliseconds)
This will cause a function to be called repeatedly at specified time intervals.

A timer that has been set by setInterval() can be stopped by calling:

clearInterval(timerVariable)
This will cause timerVariable to stop calling its associated timer function.

In order to use clearInterval(), a timer variable needs to be associated with a call to setInterval. This is done below:

```
var myTimer = setInterval(myFunction, 100);
clearInterval(myTimer);
```

Example using setInterval (Run Example)

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <link rel="shortcut icon" type="image/png" href="images/genius_icon.png"/>
        <style>
            #gameCanvas
            {
                /* the canvas styling usually does not change */
                outline:1px solid darkgrey;
                width:500px;
                height:500px;
            }
        </style>
    </head>

    <body>
        <canvas id = "gameCanvas"></canvas>

        <script>
            let mapImage = new Image();
            mapImage.src = "images/map.png";

            let carImage = new Image();
            carImage.src = "images/car.png";

            window.onload = function ()
            {
                let canvas = document.getElementById("gameCanvas");
                canvas.width = canvas.clientWidth;
                canvas.height = canvas.clientHeight;
                let ctx = canvas.getContext("2d");

                let x = 0;

                animationInterval = setInterval(renderCanvas, 25); // call renderCanvas function every 25 milliseconds

                function renderCanvas()
                {
                    ctx.drawImage(mapImage, 0, 0, canvas.width, canvas.height); // clear any previous drawing

                    if (x > (canvas.width + 1))
                    {
                        x = 0;
                    }
                    else
                    {
                        ctx.drawImage(carImage, x, 240, 40, 20);
                        x++;
                    }
                }
            };
```

```
        </script>
    </body>
</html>
```

RequestAnimationFrame

In the above example, the rendering code is tied to the moving image (gameObject) code. This solution is fine when there is only one gameObject. It is not practical when there are many gameObjects. Ideally, we should seperate the rendering code from the gameObject code.

In order to ensure smooth animation display, the rendering code should be called as often as possible. Javascript has an additional timer, called requestAnimationFrame(). This timer will run as fast as the device allows. Faster devices will call this timer more frequently that slower devices when running the same program. Using requestAnimationFrame will always result in the best game rendering. The example below separates the gameObject update code from the rendering code.

Example of separate update and render code (Run Example)

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <link rel="shortcut icon" type="image/png" href="images/genius_icon.png"/>
        <style>
            #gameCanvas
            {
                /* the canvas styling usually does not change */
                outline:1px solid darkgrey;
                width:500px;
                height:500px;
            }
        </style>

    </head>

    <body>
        <canvas id = "gameCanvas"></canvas>

        <script>
            let imgImage = new Image();
            imgImage.src = "images/map.png";

            let carImage = new Image();
            carImage.src = "images/car.png";

            window.onload = function ()
            {
                let canvas = document.getElementById("gameCanvas");
                canvas.width = canvas.clientWidth;
                canvas.height = canvas.clientHeight;
                let ctx = canvas.getContext("2d");

                let x = 0;

                animationInterval = setInterval(updateCarState, 25); // call renderCanvas function every 25 milliseconds

                renderCanvas(); // first call to renderCanvas

                function updateCarState()
                {
                    if (x > (canvas.width + 1))
                    {
                        x = 0;
                    }
                    else
                    {
                        x++;
                    }
                }

                function renderCanvas()
                {
                    ctx.drawImage(imgImage, 0, 0, canvas.width, canvas.height); // clear any previous drawing
                    ctx.drawImage(carImage, x, 240, 40, 20);

                    requestAnimationFrame(renderCanvas); // call renderCanvas again as soon as possible
                }
```

```
            };
        </script>
    </body>
</html>
```

Framework

Many canvas games consist of multiple gameObjects moving around and interacting on a canvas. We shall develop a simple object oriented framework that we can use to develop gameObject-based canvas games. The framework consists of the basic html code, various javascript classes and global variables. The minimum requirement to use this framework is to include the four framework read-only files below and two game-specific files:

Framework read-only files:

- index.js
- game.css
- CanvasGame.js
- GameObject.js

Game-specific files:

- example_game.html
- example_game.js

Index.Html

index.html

```
/*****************************************************************************/
/* This file is the same for every game.                                     */
/* DO NOT EDIT THIS FILE                                                     */
/*****************************************************************************/


/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.       */
/* This file holds the global variables that will be used in all games.    */
/* This file always calls the playGame function().                         */
/* It also holds game specific code, which will be different for each game */

/************** Declare data and functions that are needed for all games ************/

/* Always create a canvas and a ctx */
let canvas = null;
let ctx = null;

/* Always create an array that holds the default game gameObjects */
let gameObjects = [];

/*********** END OF Declare data and functions that are needed for all games *********/

/* Wait for all game assets, such as audio and images to load before starting the game */
/* The code below will work for both websites and Cordova mobile apps                  */
window.addEventListener("load", onAllAssetsLoaded);           // needed for websites
document.addEventListener("deviceready", onAllAssetsLoaded);  // needed for Cordova mobile apps

document.write("<div id='loadingMessage'>Loading...</div>");
function onAllAssetsLoaded()
{
    /* hide the webpage loading message */
    document.getElementById('loadingMessage').style.visibility = "hidden";

    /* Initialise the canvas and associated variables */
    /* This code never changes                        */
    canvas = document.getElementById("gameCanvas");
    ctx = canvas.getContext("2d");
    canvas.width = canvas.clientWidth;
    canvas.height = canvas.clientHeight;

    playGame(); // Each game will include its own .js file, which will hold the game's palyGame() function
}


/* global functions */

/* Convert from degrees to radians */
Math.radians = function (degrees)
{
    return degrees * Math.PI / 180;
};
```

Code Explained

Code Explained

This file sets the global variables that are used in all games. In particular, it sets "canvas" and "ctx" to be global. As these are used in both the CanvasGame and GameObject class, making them global improves performance.
This gameObjects[] array holds the game's gameObjects. This ties into the render() method code within CanvasCode. Having all of the gameObjects in one array makes it easy for us to step thgough them all, ensuring that all gameObjects are continuously rendered during gameplay. Using the gameObjects[] array makes writing rendering code much simpler for the developer.

```
/************** Declare data and functions that are needed for all games ************/

/* Always create a canvas and a ctx */
let canvas = null;
let ctx = null;

/* Always create an array that holds the default game gameObjects */
let gameObjects = [];

/*********** END OF Declare data and functions that are needed for all games *********/
```

This code displays a "Loading..." message while the game assets, such as images, are loading. Once all of the webpage assets have loaded, the onAllAssetsLoaded() function hides the load message and calls the playGame() function.

```
/* Wait for all game assets, such as audio and images to load before starting the game */
/* The code below will work for both websites and Cordova mobile apps                  */
window.addEventListener("load", onAllAssetsLoaded);           // needed for websites
document.addEventListener("deviceready", onAllAssetsLoaded);  // needed for Cordova mobile apps

document.write("<div id='loadingMessage'>Loading...</div>");
function onAllAssetsLoaded()
{
    /* hide the webpage loading message */
    document.getElementById('loadingMessage').style.visibility = "hidden";

    ...
```

The global variables canvas and ctx can only be initialised once the game's webpage has been loaded loaded.

```
    /* Initialise the canvas and associated variables */
    /* This code never changes                        */
    canvas = document.getElementById("gameCanvas");
    ctx = canvas.getContext("2d");
    canvas.width = canvas.clientWidth;
    canvas.height = canvas.clientHeight;
```

The function playGame() plays the game. The code for playGame() will be contained in the .js game file that was included in the game html file (in this example example_game.js and example_game.html).

```
    playGame(); // Each game will include its own .js file, which will hold the game's palyGame() function
```

Any global functions should be added to the index.js file. For these notes, the only global function is Math.radians().

```
/* global functions */

/* Convert from degrees to radians */
Math.radians = function (degrees)
{
    return degrees * Math.PI / 180;
};
```

Game Stylesheet

The game stylesheet is always the same. If game-specific css code is needed, then create another css file to hold it.

game.css

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */
/* style class for game's container, canvas and loading message      */
```

```
#gameContainer
{
    /* style the gameContainer if the game includes non-canvas elements */
}

#gameCanvas
{
    /* the canvas styling usually does not change */
    outline:1px solid darkgrey;
    width:500px;
    height:500px;
}

#loadingMessage
{
    /* the loading message styling usually does not change */
    position:absolute;
    top:100px;
    left:100px;
    z-index:100;
    font-size:50px;
}
```

Code Explained

The gameContainer and gameCanvas hold the game.

The loadingMessage is the message that appears while the game assets are loading.

CanvasGame.Js

This class holds the game loop, renders the gameObjects and looks after collision detection.

```
/*****************************************************************************************************/
/* This file is the same for every game.                                                             */
/* DO NOT EDIT THIS FILE.                                                                            */
/*                                                                                                   */
/* If you need to modify the methods, then you should create a sub-class that extends from this class. */
/*                                                                                                   */
/*****************************************************************************************************/


/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.                                 *
/* The CanvasGame class is responsible for rendering all of the gameObjects and other game graphics on the canvas.
/* If you want to implement collision detection in your game, then you MUST overwrite the collisionDetection() method. *

class CanvasGame
{
    constructor()
    {
        /* render the game on the canvas */
        this.playGameLoop();
    }

    start()
    {
        for (let i = 0; i < gameObjects.length; i++)
        {
            gameObjects[i].start();
        }
    }

    playGameLoop()
    {
        this.collisionDetection();
        this.render();

        /* recursively call playGameLoop() */
        requestAnimationFrame(this.playGameLoop.bind(this));
    }
```

```
    render()
    {
        /* clear previous rendering from the canvas */
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        /* render game gameObjects on the canvas */
        for (let i = 0; i < gameObjects.length; i++)
        {
            /* Use an empty gameObject to ensure that there are no empty entries in the gameObjects[] array */
            /* This is needed because an empty entry in the gameObjects[] array will cause the program to freeze */
            if(gameObjects[i] === undefined)
            {
                gameObjects[i] = new GameObject();
            }

            if (gameObjects[i].isDisplayed())
            {
                gameObjects[i].render();
            }
        }
    }

    collisionDetection()
    {
        /* If you need to implement collision detection in your game, then you can overwrite this method in your sub-cla
        /* If you do not need to implement collision detection, then you do not need to overwrite this method.
    }
}
```

Code Explained

```
constructor()
    {
        /* render the game on the canvas */
        this.playGameLoop();
    }
```

The start() function is called inside the game's main js file (in this webpage, it is the file example_game.js).
It starts the various gameObjects' timers.

```
    start()
    {
        for (let i = 0; i < this.gameObjects.length; i++)
        {
            this.gameObjects[i].start();
        }

    }
```

The playGameLoop() continuously checks the gameObjects for collisions and renders the gameObjects.

```
 playGameLoop()
    {
        this.collisionDetection();
        this.render();

        /* recursively call playGameLoop() */
        requestAnimationFrame(this.playGameLoop.bind(this));
    }
```

The render() function renders all of the gameObjects that are included in the game.

Because of the way that entries are added to the gameObjects[] array, it is possible that an entry in the array will be empty. When this occurs, we need to place an empty placeholder gameObject in the array, as shown in red.

Each gameObject contains an isDisplayed() method, which will return true whenever the gameObject's start() method is called and false whenever the gameObject's stopAndHide() method is called. In this way, we can show or hide gameObjects on the canvas. We test against the isDisplayed() method to display only those gameObjects that are visible, as shown in blue.

```
    render()
    {
        /* clear previous rendering from the canvas */
        ctx.clearRect(0, 0, canvas.width, canvas.height);

        /* render game gameObjects on the canvas */
```

```
        for (let i = 0; i < gameObjects.length; i++)
        {
            /* Use an empty gameObject to ensure that there are no empty entries in the gameObjects[] array */
            /* This is needed because an empty entry in the gameObjects[] array will cause the program to freeze */
            if(gameObjects[i] === undefined)
            {
                gameObjects[i] = new GameObject();
            }

            if (gameObjects[i].isDisplayed())
            {
                gameObjects[i].render();
            }
        }
    }
```

The collisionDetection() function will be specific to each game. If a game does not involve gameObject collision detection, then leave this funciton empty.
If your game has collision detection, then you should create your own sub-class that extends from this class. Inside your new sub-class you should override this function.

```
    collisionDetection()
    {
        /* If you need to implement collision detection in your game, then you can overwrite this method in your sub-cla
        /* If you do not need to implement collision detection, then you do not need to overwrite this method.
    }
```

GameObject.Js

GameObject.js is the GameObject super-class. All of the gameObjects in a game must extend from GameObject.

```
/*****************************************************************************************************/
/* This file is the same for every game.                                                             */
/* DO NOT EDIT THIS FILE.                                                                            */
/*                                                                                                   */
/* If you need to modify the methods, then you should create a sub-class that extends from this class. */
/*                                                                                                   */
/*****************************************************************************************************/


/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.
/* This is the superclass that gameObjects extend from.
/* It holds the data and functionality that is common to all gameObjects.
/* If you want to implement state change in your game, then you MUST overwrite the updateState() method in your sub-clas
/* You MUST overwrite the code inside this if-statement with your own render() code in your sub-class that extends GameO
/* This class will usually not change.

class GameObject
{
    constructor(updateStateMilliseconds, delay = 0)
    {
        /* These are ALWAYS needed */
        this.gameObjectInterval = null; /* set to null when not running */
        this.gameObjectIsDisplayed = false;
        this.updateStateMilliseconds = updateStateMilliseconds; /* change to suit the gameObject state update in millise
        this.delay = delay; /* delay the start of the updateState() method */
    }

    start()
    {
        if ((this.updateStateMilliseconds !== null) && (this.gameObjectInterval === null))
        {
            setTimeout(startUpdateStateInterval.bind(this), this.delay);
        }
        else if (this.updateStateMilliseconds === null)
        {
            this.gameObjectIsDisplayed = true; // by default, gameObjects that have no updateState() interval should be
        }

        function startUpdateStateInterval() // this function is only ever called from inside the start() method
        {
            this.gameObjectInterval = setInterval(this.updateState.bind(this), this.updateStateMilliseconds);
            this.gameObjectIsDisplayed = true;
```

```
        }
    }

    stop()
    {
        if (this.gameObjectInterval !== null)
        {
            clearInterval(this.gameObjectInterval);
            this.gameObjectInterval = null; /* set to null when not running */
        }
        this.gameObjectIsDisplayed = true;
    }

    stopAndHide()
    {
        this.stop();
        this.gameObjectIsDisplayed = false;
    }

    isDisplayed()
    {
        return (this.gameObjectIsDisplayed);
    }

    updateState()
    {
        /* If you need to change state data in your game, then you can overwrite this method in your sub-class. */
        /* If you do not need to change data state, then you do not need to overwrite this method.              */
    }

    render()
    {
        /* If your gameObject renders, then you overwrite this method with your own render() code */
    }
}
```

You will always need to create one or more of your own sub-classes that extend from the GameObject class.

Code Explained

The constructor is always given a updateStateMilliseconds value. This is the length of the interval between each call to the class's update() method.
A delay (in milliseconds) can also be provided. The first call of update will wait for the delay has passed before executing.
The code in this method never changes for different gameObjects.

```
    constructor(updateStateMilliseconds, delay = 0)
    {
        /* These are ALWAYS needed */
        this.gameObjectInterval = null; /* set to null when not running */
        this.gameObjectIsDisplayed = false;
        this.updateStateMilliseconds = updateStateMilliseconds; /* change to suit the gameObject state update in millise
        this.delay = delay; /* delay the start of the updateState() method */
    }
```

The start() method waits until the delay has passed before calling the startUpdateStateInterval() function. The start() method will not call startUpdateStateInterval() if the gameObject is already started.

```
    start()
    {
        if ((this.updateStateMilliseconds !== null) && (this.gameObjectInterval === null))
        {
            setTimeout(startUpdateStateInterval.bind(this), this.delay);
        }
        else if (this.updateStateMilliseconds === null)
        {
            this.gameObjectIsDisplayed = true; // by default, gameObjects that have no updateState() interval should be
        }

        function startUpdateStateInterval() // this function is only ever called from inside the start() method
        {
            this.gameObjectInterval = setInterval(this.updateState.bind(this), this.updateStateMilliseconds);
            this.gameObjectIsDisplayed = true;
        }
    }
```

Both stop() and stopAndHide() stop the gameObject's timer. If you stop() a gameObject, it still remains visible. If you stopAndHide() a gameObject, it is not rendered.
The isDisplayed() method returns true if the gameObject should be rendered.
The code in all three of these methods never change for different gameObjects.

```
    stop()
    {
        if (this.gameObjectInterval !== null)
        {
            clearInterval(this.gameObjectInterval);
            this.gameObjectInterval = null; /* set to null when not running */
        }
        this.gameObjectIsDisplayed = true;
    }

    stopAndHide()
    {
        this.stop();
        this.gameObjectIsDisplayed = false;
    }

    isDisplayed()
    {
        return (this.gameObjectIsDisplayed);
    }
```

The code for updateState() and render() will be determined by the game.
You should always create your own sub-class that extends from the GameObject class. If your gameObject can change state, then you need to override the updateState() method. If your gameObject does not change state, then do nothing.

Your gameObject will usually render. If your gameObject renders, then so you need to override the render() method.

```
    updateState()
    {
        /* If you need to change state data in your game, then you can overwrite this method in your sub-class. */
        /* If you do not need to change data state, then you do not need to overwrite this method.              */
    }

    render()
    {
        /* If your gameObject renders, then you overwrite this method with your own render() code */
    }
```

Game HTML File

This is the HTML that holds the canvas. In this example, assume that the filename is "example_game.html". The template for the HTML is shown below:

example_game.html

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Genius worked example</title>
        <link rel="shortcut icon" type="image/png" href="images/genius_icon.png"/>

        <!-- Always include the game stylesheet, the Game class, the GameObject class and index.js -->
        <!-- These four must be included in all games. This code never changes.                    -->
        <link href="css/game.css" rel="stylesheet" type="text/css"/>
        <script src="js/CanvasGame.js" type="text/javascript"></script>
        <script src="js/GameObject.js" type="text/javascript"></script>
        <script src="js/index.js" type="text/javascript"></script>

        <!-- ************************************************************************** -->
        <!-- ************** THE CODE BELOW CAN BE DIFFERENT FOR EACH GAME ************** -->

        <!-- Always include the game javascript that matches the html file name -->
        <script src="js/example_game.js" type="text/javascript"></script>

        <!-- Include any classes that extend from GameObject that are used in this game -->

        <!-- ********************** END OF GAME SPECIFIC CODE ********************** -->
        <!-- ************************************************************************** -->
    </head>

    <body>
```

```
        <div id="gameContainer"> <!-- having a container will allow us to have a game that includes elements that are ou
            <canvas id="gameCanvas" tabindex="1"></canvas>

            <!-- ************************************************************************** -->
            <!-- *************** THE CODE BELOW CAN BE DIFFERENT FOR EACH GAME *************** -->


            <!-- ************************* END OF GAME SPECIFIC CODE ************************ -->
            <!-- ************************************************************************** -->

        </div>
    </body>
</html>
```

Code Explained

It is useful to note that most of the code never changes between games. All non-highlighted code in the listing above will be the same for every game.

We need to always include the four game framework read-only files. The workings of these four files was described at the start of this webpage.

```
        <!-- Always include the game stylesheet, the Game class, the GameObject class and index.js -->
        <!-- These four must be included in all games. This code never changes.                 -->
        <link href="css/game.css" rel="stylesheet" type="text/css"/>
        <script src="js/CanvasGame.js" type="text/javascript"></script>
        <script src="js/GameObject.js" type="text/javascript"></script>
        <script src="js/index.js" type="text/javascript"></script>
```

We always have to include a main game javascript file and at least one class that extends from GameObject.
Although it is not strictly necessary, it makes sense that the filename example_game.js matches the name of the HTML file (in this case example_game.HTML).

```
<!-- Always include the game javascript that matches the html file name -->
<script src="js/example_game.js" type="text/javascript"></script>
```

In this example, MygameObject is a class that extends from GameObject (which is included in GameObject.js). There will always be at least one type of gameObject in every game. Otherwise, it is pointless to use this framework. There can be more than one gameObject sub-class in a game. In this case, each gameObject sub-class will have its own .js file included in the above code.

```
<!-- Include any classes that extend from GameObject that are used in this game -->
<script src="js/MygameObject.js" type="text/javascript"></script>
```

The canvas must always have the id "gameCanvas", as it is used in other parts of the code to connect the HTML webpage to the javascript game.
The canvas is wrapped in a div. This will alllow us to develop games that include HTML elements outside of the canvas. Later on, we shall see that these other HTML elements are able to communicate with the canvas. The div must always be called "gameContainer", as it is used in other parts of the code to connect the HTML webpage to the javascript game.

```
<body>
    <div id="gameContainer"> <!-- having a container will allow us to have a game that includes elements that are outsid
        <canvas id = "gameCanvas" tabindex="1"></canvas>
    </div>
</body>
```

Game .Js File

example_game.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.            */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                              */
/* It also holds game specific code, which will be different for each game      */
```

```
/******************** Declare game specific global data and functions *****************/
/* images must be declared as global, so that they will load before the game starts  */

/****************** END OF Declare game specific data and functions *****************/




/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* Specifically, this function will:                                */
    /* 1. initialise the canvas and associated variables                */
    /* 2. create the various game gameObjects,                  */
    /* 3. store the gameObjects in an array                     */
    /* 4. create a new Game to display the gameObjects          */
    /* 5. start the Game                                                */



    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new MyGame();

    /* Always play the game */
    game.start();


    /* If they are needed, then include any game-specific mouse and keyboard listners */
}
```

Code Explained

Each game can have its own global variables and functions. These will be declared here:

```
/******************** Declare game specific global data and functions *****************/
/* images must be declared as global, so that they will load before the game starts  */



/****************** END OF Declare game specific data and functions *****************/
```

Every gameObject-based game must have an array of gameObjects. It is declared as shown below. The gameObjects[] array is accessed by the game's render() and collisionDetection() methods in CanvasGame class.

```
/* Always create an array that holds the default game gameObjects */
let gameObjects = [];
```

Every game will have a playGame() function. This is the beginning point of the game. This function is called from the game's HTML file above. Most of the playGame() function code below is the same for every game. Only the highlighted code below will change between games.

```
/********** END OF Declare data and functions that are needed for all games *********/
```

```
/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */


    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();
}
```

Static Images

Quite often, there will be some static images in a game. If we want to display static images, then we can use a gameObject that does not override the GameObject class's update() method. An example is shown here. The code for this example is shown below.

StaticImage.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class StaticImage extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image, x, y, width, height)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;
        this.width = width;
        this.height = height;
        this.x = x;
        this.y = y;
    }

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

Code Explained

The render() method displays the image at the position and with the width and height that were passed into the constructor. As there is no state change, we do not need to override the update() method.

Along with the StaticImage class, we need to make some changes to the static_image.html and static_image.js files. All other files remain unchanged.

The majority of the code in static_image.html and static_image.js remain unchanged from previous examples. The changes are highlighted in the code below.

four_corner_images.html

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <link rel="shortcut icon" type="image/png" href="images/genius_icon.png"/>

        <!-- Always include the game stylesheet, the Game class, the GameObject class and index.js -->
        <!-- These four must be included in all games. This code never changes.                  -->
        <link href="css/game.css" rel="stylesheet" type="text/css"/>
        <script src="js/CanvasGame.js" type="text/javascript"></script>
        <script src="js/GameObject.js" type="text/javascript"></script>
        <script src="js/index.js" type="text/javascript"></script>

        <!-- ************************************************************************** -->
        <!-- ************** THE CODE BELOW CAN BE DIFFERENT FOR EACH GAME ************** -->

        <!-- Always include the game javascript that matches the html file name -->
        <script src="js/four_corner_images.js" type="text/javascript"></script>

        <!-- Include any classes that extend from GameObject that are used in this game -->
        <script src="js/StaticImage.js" type="text/javascript"></script>

        <!-- ************************ END OF GAME SPECIFIC CODE ************************ -->
        <!-- ************************************************************************** -->
    </head>

    <body>
        <div id="gameContainer"> <!-- having a container will allow us to have a game that includes elements that are ou
            <canvas id="gameCanvas" tabindex="1"></canvas>
```

```
            <!-- ********************************************************************************** -->
            <!-- ************** THE CODE BELOW CAN BE DIFFERENT FOR EACH GAME ************** -->


            <!-- ************************ END OF GAME SPECIFIC CODE ************************ -->
            <!-- ********************************************************************************** -->
        </div>
    </body>
</html>
```

four_corner_images.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.             */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */




/******************* Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */

let cityImage = new Image();
cityImage.src = "images/city.png";

let beachImage = new Image();
beachImage.src = "images/beach.png";

let mountainImage = new Image();
mountainImage.src = "images/mountain.png";

let riverImage = new Image();
riverImage.src = "images/river.png";

/****************** END OF Declare game specific data and functions ****************/



/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */

    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[0] = new StaticImage(cityImage, 0, 0, canvas.width / 2, canvas.height / 2);
    gameObjects[1] = new StaticImage(beachImage, canvas.width / 2, 0, canvas.width / 2, canvas.height / 2);
    gameObjects[2] = new StaticImage(mountainImage, 0, canvas.height / 2, canvas.width / 2, canvas.height / 2);
    gameObjects[3] = new StaticImage(riverImage, canvas.width / 2, canvas.height / 2, canvas.width / 2, canvas.height /

    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();


    /* If they are needed, then include any game-specific mouse and keyboard listners */
}
```

Exercises

Write code to draw four coloured blocks on the canvas, as shown here.

Write code to draw a block that fades out from white to red, as shown here.

Write code to draw a set of randomly coloured blocks, as shown here.

Write code to make a grayscale circle gradient, as shown here.

Write code to make a grayscale circle gradient, as shown here.

Static Text

Quite often, there will be some static text in a game. If we want to display static images, then we can use a gameObject that does not override the GameObject class's update() method. An example is shown here. The code for this example is shown below.

StaticText.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

const STATIC_TEXT_CENTRE = -1;

class StaticText extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(text, x, y, font, fontSize, colour)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.text = text;
        this.x = x;
        this.y = y;
        this.font = font;
        this.fontSize = fontSize;
        this.colour = colour;

        ctx.font = this.fontSize + "px " + this.font;
        this.width = ctx.measureText(this.text).width;
        if (this.x === STATIC_TEXT_CENTRE)
        {
            this.x = (canvas.width - this.width) / 2;
        }
    }

    render()
    {
        ctx.fillStyle = this.colour;
        ctx.fillText(this.text, this.x, this.y);
    }
}
```

Code Explained

STATIC_TEXT_CENTRE is used as a flag that will set the text to be horrizontally centred.

```
const STATIC_TEXT_CENTRE = -1;
```

In the constructor() method, we measure the width of the text using the highlighted code below:

```
constructor(text, x, y, font, fontSize, colour)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.text = text;
        this.x = x;
```

```
        this.y = y;
        this.font = font;
        this.fontSize = fontSize;
        this.colour = colour;

        ctx.font = this.fontSize + "px " + this.font;
        this.width = ctx.measureText(this.text).width;
        if (this.x === STATIC_TEXT_CENTRE)
        {
            this.x = (canvas.width - this.width) / 2;
        }
    }
```

If "this.x" has been set to be the value STATIC_TEXT_CENTRE, then we centre the text, as shown in the highlighted code below.

```
constructor(text, x, y, font, fontSize, colour)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.text = text;
        this.x = x;
        this.y = y;
        this.font = font;
        this.fontSize = fontSize;
        this.colour = colour;

        ctx.font = this.fontSize + "px " + this.font;
        this.width = ctx.measureText(this.text).width;
        if (this.x === STATIC_TEXT_CENTRE)
        {
            this.x = (canvas.width - this.width) / 2;
        }
    }
```

Pulsating Images

A pulsating image is similar to a static image in that it will have a position, a width and a height. Unlike a static image, a pulsating image will continuously update its position, as shown here. The code for this example is shown below.

PulsatingImage.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class PulsatingImage extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image, x, y, originalWidth, originalHeight, numberOfSteps, stepSize, intervalTime)
    {
        super(intervalTime); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;
        this.x = x;
        this.y = y;
        this.width = originalWidth;
        this.height = originalHeight;
        this.numberOfSteps = numberOfSteps;

        this.stepSize = stepSize; // can be positive or negative. Negative decreases the original image size
        if (this.stepSize > 0) // image is pulsating between original size to a larger size
        {
            this.incrementing = true;
        }
        else // image is pulsating between original size to a larger size
        {
            this.incrementing = false;
        }

        // the image pulsates about its centre
        this.centreX = x + (this.width / 2);
        this.centreY = y + (this.height / 2);

        this.currentStep = 0;
    }

    updateState()
    {
        if (this.stepSize > 0) // image pulses into a bigger image
        {
            if (this.incrementing) // incrementing to increase size of original image
            {
                this.currentStep++;
                if (this.currentStep === this.numberOfSteps)
                {
                    this.incrementing = false;
                }
            }
            else // drecrementing to return to original image size
            {
                this.currentStep--;
                if (this.currentStep === 0)
                {
                    this.incrementing = true;
                }
            }
        }
        else // image pulses into a smaller image
        {
            if (!this.incrementing) // drecrementing to decrease size of original image
            {
                this.currentStep++;
                if (this.currentStep === this.numberOfSteps)
                {
                    this.incrementing = true;
                }
            }
            else // drecrementing to return to original image size
            {
                this.currentStep--;
                if (this.currentStep === 0)
                {
                    this.incrementing = false;
```

```
                }
            }
        }
    }

    render()
    {
        ctx.drawImage(this.image, this.x - (this.currentStep * this.stepSize) / 2, this.y - (this.currentStep * this.ste
    }
}
```

Code Explained

The intervalTime is passed to the Visual superclass (as shown below in green). This determines the frequency that updateState() method is called. Smaller values of intervalTime will result in a faster pulsating effect.
The image pulsates about its centre. The initialisation of this is shown below in blue.
The stepSize (as shown below in red) is the number of pixels to increment/decrement the width and height of the displayed image on each call of updateState(). The image can pulsate in a positive or negative direction. A positive value for stepSize will cause the image to pulsate between the original and a larger image. A negavite value for stepSize will cause the image to pulsate between the original and a smaller image.

```
    constructor(image, x, y, originalWidth, originalHeight, numberOfSteps, stepSize, intervalTime, centreX = null, centr
    {
        super(intervalTime); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;
        this.x = x;
        this.y = y;
        this.width = originalWidth;
        this.height = originalHeight;
        this.numberOfSteps = numberOfSteps;
                this.stepSize = stepSize; // can be positive or negative. Negative decreases the original image size
        if (this.stepSize > 0) // image is pulsating between original size to a larger size
        {
            this.incrementing = true;
        }
        else // image is pulsating between original size to a larger size
        {
            this.incrementing = false;
        }

        // the image pulsates about its centre
        this.centreX = x + (this.width / 2);
        this.centreY = y + (this.height / 2);

        this.currentStep = 0;
    }
```

The updateState() method deals with the cases of the image incrementing (i.e. pulsating between the original and a larger image) or decrementing (i.e. pulsating between the original and a smaller image). The two sets of code are very similar. The incrementing code is shown below in red and the decrementing code is shown below in blue.

```
    updateState()
    {
        if (this.stepSize > 0) // image pulses into a bigger image
        {
            if (this.incrementing) // incrementing to increase size of original image
            {
                this.currentStep++;
                if (this.currentStep === this.numberOfSteps)
                {
                    this.incrementing = false;
                }
            }
            else // drecrementing to return to original image size
            {
                this.currentStep--;
                if (this.currentStep === 0)
                {
                    this.incrementing = true;
                }
            }
        }
        else // image pulses into a smaller image
        {
            if (!this.incrementing) // drecrementing to decrease size of original image
            {
                this.currentStep++;
                if (this.currentStep === this.numberOfSteps)
                {
```

```
                this.incrementing = true;
            }
        }
        else // drecrementing to return to original image size
        {
            this.currentStep--;
            if (this.currentStep === 0)
            {
                this.incrementing = false;
            }
        }
    }
}
```

The render() method draws the image on the canvas. It is straight forward and does not need explaining.

```
    render()
    {
        ctx.drawImage(this.image, this.x - (this.currentStep * this.stepSize) / 2, this.y - (this.currentStep * this.ste
    }
```

pulsating_image.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.             */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */




/******************** Declare game specific global data and functions *****************/
/* images must be declared as global, so that they will load before the game starts  */
let beachImage = new Image();
beachImage.src = "images/beach.png";
/****************** END OF Declare game specific data and functions *****************/







/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[0] = new PulsatingImage(beachImage, 0, 0, canvas.width, canvas.height, 10, 2, 100);

    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();


    /* If they are needed, then include any game-specific mouse and keyboard listners */
}
```

Code Explained

We add a PulsatingImage object to the gameObjects[], as shown below.

```
    gameObjects[0] = new PulsatingImage(beachImage, 0, 0, canvas.width, canvas.height, 10, 2, 100);
```

Exercises

Write a class to cause an image to rotate about a given centre point, as shown here. If no centre point is given, then the image should rotate about its own centre. The user should be able to specify a positive or negative rotation stepSize.

Buttons

Buttons are very similar to static text and static images. If we want to display a button, then we can use a gameObject that does not override the GameObject class's update() method. The Button class can contain text and/or an image.

Button.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

const BUTTON_CENTRE = -1;
const TEXT_WIDTH = -2;
const TEXT_HEIGHT = -3;

class Button extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(x, y, width, height, text, backgroundImage = null, fontSize = 50, font = "Times Roman", textColour = "Bl
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.text = text;
        this.backgroundImage = backgroundImage;

        this.isHovering = false;

        /* set default text values */
        this.font = font;
        this.fontSize = fontSize;
        this.textColour = textColour;
        this.backgroundColour = backgroundColour;
        this.strokeStyle = borderColour;
        this.borderWidth = borderWidth;

        /* set the size and postition of the button */
        if (this.width === TEXT_WIDTH)
        {
            ctx.font = this.fontSize + "px " + this.font;
            this.width = ctx.measureText(this.text).width * 1.1; // make the box slightly wider than the text
        }

        if (this.height === TEXT_HEIGHT)
        {
            this.height = this.fontSize * 1.1;
        }

        if (this.x === BUTTON_CENTRE)
        {
            this.x = (canvas.width - this.width) / 2;
        }
    }

    render()
    {
        /* the ImageButton's border */
        ctx.beginPath();
        ctx.strokeStyle = this.strokeStyle;
        ctx.lineWidth = this.borderWidth;
        ctx.moveTo(this.x, this.y);
        ctx.lineTo(this.x + this.width, this.y);
        ctx.lineTo(this.x + this.width, this.y + this.height);
        ctx.lineTo(this.x, this.y + this.height);
        ctx.lineTo(this.x, this.y);
        ctx.stroke();
        ctx.fillStyle = this.backgroundColour;
        ctx.fillRect(this.x, this.y, this.width, this.height);
        ctx.closePath();

        /* the button's image */
        if (this.backgroundImage !== null)
        {
            ctx.drawImage(this.backgroundImage, this.x, this.y, this.width, this.height);
        }
```

```
        /* the button's text */
        ctx.fillStyle = this.textColour;
        ctx.font = this.fontSize + "px " + this.font; // need to set the font each time, as it might have been changed e
        ctx.fillText(this.text, this.x + this.width * 0.05, this.y + this.height * 0.75);

        /* brighten the image if the mouse is hovering over it */
        if (this.isHovering)
        {
            let imageData = ctx.getImageData(this.x, this.y, this.width, this.height);
            let data = imageData.data;

            /* Manipulate the pixel data */
            for (let i = 0; i < data.length; i += 4)
            {
                data[i + 0] = data[i + 0] + 30;
                data[i + 1] = data[i + 1] + 30;
                data[i + 2] = data[i + 2] + 30;
            }

            ctx.putImageData(imageData, this.x, this.y);
        }
    }

    pointIsInsideBoundingRectangle(pointX, pointY)
    {
        if (!this.gameObjectIsDisplayed)
        {
            this.isHovering = false;
            return false;
        }
        if ((pointX > this.x) && (pointY > this.y))
        {
            if (pointX > this.x)
            {
                if ((pointX - this.x) > this.width)
                {
                    this.isHovering = false;
                    return false; // to the right of this gameObject
                }
            }

            if ((pointY - this.y) > this.height)
            {
                if (pointY > this.height)
                {
                    this.isHovering = false;
                    return false; // below this gameObject
                }
            }
        }
        else // above or to the left of this gameObject
        {
            this.isHovering = false;
            return false;
        }
        this.isHovering = true;
        return true; // inside this gameObject
    }
}
```

Code Explained

BUTTON_CENTRE is used as a flag that will set the button to be horrizontally centred.
const TEXT_WIDTH is used as a flag to indicate that the Button should be the width of the text that is on the Button.
const TEXT_HEIGHT is used as a flag to indicate that the Button should be the height of the text that is on the Button.

The render() method will draw the button's border, then its image and then its text. If the flag "this.isHovering" is set to true, then the render() method will brighten the button's image and text.

The "this.isHovering" flag matches the return value of pointIsInsideBoundingRectangle().
The pointIsInsideBoundingRectangle() method is used to detect that the mouse is hovering on the button.
The mousedown and mousemove event listners are used in the main game code to interact with the Button object, as shown in the example below.

We detect mouse over and mouse pressing of the button in the main js file, as shown in the example below.

Example hovering over and clicking a Button gameObject (Run Example).

demo_button.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.             */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */




/******************* Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */
const TEXT_BUTTON = 0;
const SMALL_TEXT_BUTTON = 1;
const IMAGE_BUTTON = 2;
const TEXT_AND_IMAGE_BUTTON = 3;
const MESSAGE = 4;

let cityImage = new Image;
cityImage.src = "images/city.png";
/****************** END OF Declare game specific data and functions ****************/



/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[TEXT_BUTTON] = new Button(BUTTON_CENTRE, 10, TEXT_WIDTH, TEXT_HEIGHT, "Text Button!");
    gameObjects[SMALL_TEXT_BUTTON] = new Button(BUTTON_CENTRE, 100, TEXT_WIDTH, TEXT_HEIGHT, "Small Text!", false, null,
    gameObjects[IMAGE_BUTTON] = new Button(BUTTON_CENTRE, 150, 200, 150, "", false, cityImage);
    gameObjects[TEXT_AND_IMAGE_BUTTON] = new Button(100, 325, 300, 100, "Centred Text on Image!", true, cityImage, 20, "

    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();

    /* If they are needed, then include any game-specific mouse and keyboard listners */
    document.getElementById("gameCanvas").addEventListener("mousedown", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            if (gameObjects[TEXT_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[MESSAGE] = new StaticText("Text button was pressed", STATIC_TEXT_CENTRE, 490, "Times Roman",
                gameObjects[MESSAGE].start();
            }
            else if (gameObjects[SMALL_TEXT_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[MESSAGE] = new StaticText("Small text button was pressed", STATIC_TEXT_CENTRE, 490, "Times R
                gameObjects[MESSAGE].start();
            }
            else if (gameObjects[IMAGE_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[MESSAGE] = new StaticText("Image button was pressed", STATIC_TEXT_CENTRE, 490, "Times Roman"
                gameObjects[MESSAGE].start();
            }
            else if (gameObjects[TEXT_AND_IMAGE_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
```

```
                gameObjects[MESSAGE] = new StaticText("Text and image button was pressed", STATIC_TEXT_CENTRE, 490, "Tim
                gameObjects[MESSAGE].start();
            }
        }
    });

    document.getElementById("gameCanvas").addEventListener("mousemove", function (e)
    {
        if (e.which === 0) // no button selected
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            gameObjects[TEXT_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY);
            gameObjects[SMALL_TEXT_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY);
            gameObjects[IMAGE_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY);
            gameObjects[TEXT_AND_IMAGE_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY);
        }
    });
}
```

Code Explained

If the left mouse button is clicked and the mouse is inside a Button object, then we fire some event, as highlighted in red.

```
    document.getElementById("gameCanvas").addEventListener("mousedown", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            if (gameObjects[TEXT_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[MESSAGE] = new StaticText("Text button was pressed", STATIC_TEXT_CENTRE, 490, "Times Roman",
                gameObjects[MESSAGE].start();
            }
            else if (gameObjects[SMALL_TEXT_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[MESSAGE] = new StaticText("Small text button was pressed", STATIC_TEXT_CENTRE, 490, "Times R
                gameObjects[MESSAGE].start();
            }
            else if (gameObjects[IMAGE_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[MESSAGE] = new StaticText("Image button was pressed", STATIC_TEXT_CENTRE, 490, "Times Roman"
                gameObjects[MESSAGE].start();
            }
            else if (gameObjects[TEXT_AND_IMAGE_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[MESSAGE] = new StaticText("Text and image button was pressed", STATIC_TEXT_CENTRE, 490, "Tim
                gameObjects[MESSAGE].start();
            }
        }
    });
```

If the mouse hovers over a Button object and no mouse button has been pressed, then the Button object is highlighted by changing its colour. The colour change happens inside the Button object. Whenever the Button object returns true, it sets its "this.isHovering" flag to true, which in turn causes the Button's render() method to change the Button object's colour. This is highlighted in red in the code below.

```
  document.getElementById("gameCanvas").addEventListener("mousemove", function (e)
    {
        if (e.which === 0) // no button selected
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;
            gameObjects[TEXT_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY);
            gameObjects[SMALL_TEXT_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY);
            gameObjects[IMAGE_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY);
            gameObjects[TEXT_AND_IMAGE_BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY);
        }
    });
```

External Inputs

It is possible to have interaction between the game canvas and external HTML elements, as shown here.

StaticImageWithInputs.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class StaticImageWithInputs extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image, x, y, width, height)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;
        this.width = width;
        this.height = height;
        this.x = x;
        this.y = y;
    }

    render()
    {
        ctx.drawImage(this.image, this.x, this.y, this.width, this.height);
    }

    setX(newX)
    {
        this.x = newX;
    }

    setY(newY)
    {
        this.y = newY;
    }

    setWidth(newWidth)
    {
        this.width = newWidth;
    }

    setHeight(newHeight)
    {
        this.height = newHeight;
    }
}
```

Code Explained

The class has four setter methods, one for each of x, y, width and height, as highlighted in the code above.

```
    setX(newX)
    {
        this.x = newX;
    }

    setY(newY)
    {
        this.y = newY;
    }

    setWidth(newWidth)
    {
        this.width = newWidth;
    }

    setHeight(newHeight)
    {
        this.height = newHeight;
    }
```

static_image_with_inputs.html

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <link rel="shortcut icon" type="image/png" href="images/genius_icon.png"/>

        <!-- Always include the game stylesheet, the Game class, the GameObject class and index.js -->
        <!-- These four must be included in all games. This code never changes.                 -->
        <link href="css/game.css" rel="stylesheet" type="text/css"/>
        <script src="js/CanvasGame.js" type="text/javascript"></script>
        <script src="js/GameObject.js" type="text/javascript"></script>
        <script src="js/index.js" type="text/javascript"></script>

        <!-- ************************************************************************** -->
        <!-- ************** THE CODE BELOW CAN BE DIFFERENT FOR EACH GAME ************** -->

        <!-- Always include the game javascript that matches the html file name -->
        <script src="js/static_image_with_inputs.js" type="text/javascript"></script>

        <!-- Include any classes that extend from GameObject that are used in this game -->
        <script src="js/StaticImageWithInputs.js" type="text/javascript"></script>

        <!-- ************************ END OF GAME SPECIFIC CODE ************************ -->
        <!-- ************************************************************************** -->
    </head>

    <body>
        <div id="gameContainer"> <!-- having a container will allow us to have a game that includes elements that are ou
            <canvas id="gameCanvas" tabindex="1"></canvas>

            <!-- ************************************************************************** -->
            <!-- ************** THE CODE BELOW CAN BE DIFFERENT FOR EACH GAME ************** -->

            <br>
            <label for="x">X: </label><input type="range" min="0" max="490" value="125" id="x"><br>
            <label for="y">Y: </label><input type="range" min="0" max="490" value="125" id="y"><br>
            <label for="width">Width: </label><input type="range" min="10" max="500" value="250" id="width"><br>
            <label for="height">Height: </label><input type="range" min="10" max="500" value="250" id="height">

            <!-- ************************ END OF GAME SPECIFIC CODE ************************ -->
            <!-- ************************************************************************** -->

        </div>
    </body>
</html>
```

Code Explained

The various html input elements that accept the user input are all included inside the "gameContainer" div. This way, we can keep the game code separate to any other code that a developer might include on a webpage.

```
            <!-- ************************************************************************** -->
            <!-- ************** THE CODE BELOW CAN BE DIFFERENT FOR EACH GAME ************** -->

            <br>
            <label for="x">X: </label><input type="range" min="0" max="490" value="125" id="x"><br>
            <label for="y">Y: </label><input type="range" min="0" max="490" value="125" id="y"><br>
            <label for="width">Width: </label><input type="range" min="10" max="500" value="250" id="width"><br>
            <label for="height">Height: </label><input type="range" min="10" max="500" value="250" id="height">

            <!-- ************************ END OF GAME SPECIFIC CODE ************************ -->
            <!-- ************************************************************************** -->
```

Note that the "onchange" functionality for the html input elements **is not included** in the HTML file. Instead, it will be included in the game's static_image_with_inputs.js file.

static_image_with_inputs.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.            */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                              */
/* It also holds game specific code, which will be different for each game      */
```

```
/******************** Declare game specific global data and functions *****************/
/* images must be declared as global, so that they will load before the game starts  */
let cityImage = new Image();
cityImage.src = "images/city.png";

/****************** END OF Declare game specific data and functions ****************/




/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[0] = new StaticImageWithInputs(cityImage, 0, 0, 250, 250);

    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();


    /* If they are needed, then include any game-specific mouse and keyboard listners */
    /* set the initial x,y, width and height of the image */
    gameObjects[0].setX(document.getElementById("x").value);
    gameObjects[0].setY(document.getElementById("y").value);
    gameObjects[0].setWidth(document.getElementById("width").value);
    gameObjects[0].setHeight(document.getElementById("height").value);

    document.getElementById("x").addEventListener("change", function ()
    {
        gameObjects[0].setX(document.getElementById("x").value);
    });

    document.getElementById("y").addEventListener("change", function ()
    {
        gameObjects[0].setY(document.getElementById("y").value);
    });

    document.getElementById("width").addEventListener("change", function ()
    {
        gameObjects[0].setWidth(document.getElementById("width").value);
    });

    document.getElementById("height").addEventListener("change", function ()
    {
        gameObjects[0].setHeight(document.getElementById("height").value);
    });
}
```

Code Explained

The image's original position is set using the data from the four HTML input elements.

```
    gameObjects[0].setX(document.getElementById("x").value);
    gameObjects[0].setY(document.getElementById("y").value);
    gameObjects[0].setWidth(document.getElementById("width").value);
    gameObjects[0].setHeight(document.getElementById("height").value);
```

The "onchange" functionality for each of the four input elements is implemented in this file, as shown below.

```
document.getElementById("x").addEventListener("change", function ()
{
    gameObjects[0].setX(document.getElementById("x").value);
});

document.getElementById("y").addEventListener("change", function ()
{
    gameObjects[0].setY(document.getElementById("y").value);
});

document.getElementById("width").addEventListener("change", function ()
{
    gameObjects[0].setWidth(document.getElementById("width").value);
});

document.getElementById("height").addEventListener("change", function ()
{
    gameObjects[0].setHeight(document.getElementById("height").value);
});
```

Exercises

Write code to allow the user to adjust the size of multiple blocks, as shown here.

Write code to generate a piece of block art, as shown here. Note that the user should be able to specify how many blocks they would like to have.

Write code to adjust the brightness of an image, as shown here.

Write code to adjust the brightness of the red, green and blue components of an image, as shown here.

Moving Images

A moving image is similar to a static image in that it will have a position, a width and a height. Unlike a static image, a moving image will continuously update its position, as shown here. The code for this example is shown below.

Car.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class Car extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image, y, width, height, updateStateMilliseconds)
    {
        super(updateStateMilliseconds); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;
        this.width = width;
        this.height = height;
        this.x = 0;
        this.y = y;
    }

    updateState()
    {
        this.x++;
        if (this.x > canvas.width)
        {
            this.x = -this.width;
        }
    }

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

Code Explained

The updateState() method increments the x coordinate of the car until it moves off the right-side of the canvas. It then reappears at "-this.width" on the original side of the screen. Without this, the car would sudenly reappear at x coordinate 0, which does not look smooth.

```
    updateState()
    {
        this.x++;
        if (this.x > canvas.width)
        {
            this.x = -this.width;
        }
    }
```

Along with the Car class, we need to make some changes to the moving_image.html and moving_image.js files. All other files remain unchanged.

The majority of the code in moving_image.html and static_image.js remain unchanged from previous examples. The changes are highlighted in the code below.

moving_image.html

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <link rel="shortcut icon" type="image/png" href="images/genius_icon.png"/>

        <!-- Always include the game stylesheet, the Game class, the GameObject class and index.js -->
        <!-- These four must be included in all games. This code never changes.                    -->
        <link href="css/game.css" rel="stylesheet" type="text/css"/>
```

```
        <script src="js/CanvasGame.js" type="text/javascript"></script>
        <script src="js/GameObject.js" type="text/javascript"></script>
        <script src="js/index.js" type="text/javascript"></script>

        <!-- ************************************************************************** -->
        <!-- ************** THE CODE BELOW CAN BE DIFFERENT FOR EACH GAME ************** -->

        <!-- Always include the game javascript that matches the html file name -->
        <script src="js/moving_image.js" type="text/javascript"></script>

        <!-- Include any classes that extend from GameObject that are used in this game -->
        <script src="js/StaticImage.js" type="text/javascript"></script>
        <script src="js/Car.js" type="text/javascript"></script>

        <!-- ************************ END OF GAME SPECIFIC CODE ************************ -->
        <!-- ************************************************************************** -->
    </head>

    <body>
        <div id="gameContainer"> <!-- having a container will allow us to have a game that includes elements that are ou
            <canvas id="gameCanvas" tabindex="1"></canvas>

            <!-- ************************************************************************** -->
            <!-- ************** THE CODE BELOW CAN BE DIFFERENT FOR EACH GAME ************** -->


            <!-- ************************ END OF GAME SPECIFIC CODE ************************ -->
            <!-- ************************************************************************** -->

        </div>
    </body>
</html>
```

In this case, we have two different GameObject sub-classes: StaticImage and Car. Therefore, we need to include a link to both files

```
        <!-- Include any classes that extend from GameObject that are used in this game -->
        <script src="js/StaticImage.js" type="text/javascript"></script>
        <script src="js/Car.js" type="text/javascript"></script>
```

moving_image.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.            */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                              */
/* It also holds game specific code, which will be different for each game      */




/******************** Declare game specific global data and functions *****************/
/* images must be declared as global, so that they will load before the game starts  */
let mapImage = new Image();
mapImage.src = "images/map.png";

let carImage = new Image();
carImage.src = "images/car.png";

const MAP = 0;
const CAR = 1;

/****************** END OF Declare game specific data and functions *****************/







/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
```

```
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[MAP] = new StaticImage(mapImage, 0, 0, 500, 500);
    gameObjects[CAR] = new Car(carImage, 235, 40, 25, 25);

    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();


    /* If they are needed, then include any game-specific mouse and keyboard listners */
}
```

Code Explained

If there is more than one type of gameObject, it can be useful to assign constants to the various gameObjects. The red highlighted code above shows how to do this.

Exercises

Write code to allow a user to stop and restart a car image. The user should also be able to adjust the speed that the car travels, as shown here.

Moving Text

Text can be scrolled, scaled, rotated or animated in various other ways.

Example of scrolling text (Run Example)

ScrollingText.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class ScrollingText extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(text, y, endX, fontSize, textColour, delay)
    {
        super(5, delay); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.x = canvas.width;
        this.y = y;
        this.endX = endX;
        this.text = text;
        this.fontSize = fontSize;
        this.textColour = textColour;
    }

    updateState()
    {
        this.x--;
        if (this.x <= this.endX)
        {
            this.stop();
        }
    }

    render()
    {
        ctx.fillStyle = this.textColour;
        ctx.font = this.fontSize + "px Times Roman";
        ctx.fillText(this.text, this.x, this.y);
    }
}
```

Code Explained

The updateState() method scrolls the text until it reaches it endX location.

```
updateState()
    {
        this.x--;
        if (this.x <= this.endX)
        {
            this.stop();
        }
    }
```

scrolling_text.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.            */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                              */
/* It also holds game specific code, which will be different for each game      */




/****************** Declare game specific global data and functions ***************/
```

```
/* images must be declared as global, so that they will load before the game starts  */

/****************** END OF Declare game specific data and functions ****************/




/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[0] = new ScrollingText("Welcome", 70, 135, 60, "red", 0);
    gameObjects[1] = new ScrollingText("to", 300, 135, 300, "green", 2300);
    gameObjects[2] = new ScrollingText("Europe", 450, 20, 160, "blue", 4300);

    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();


    /* If they are needed, then include any game-specific mouse and keyboard listners */
}
```

Code Explained

Each text message is a seperate object, as declared below:

```
    gameObjects[0] = new ScrollingText("Welcome", 70, 135, 60, "red", 0);
    gameObjects[1] = new ScrollingText("to", 300, 135, 300, "green", 2300);
    gameObjects[2] = new ScrollingText("Europe", 450, 20, 160, "blue", 4300);
```

Scrolling Background Image

Assumeing that an image has been stitched so that its two ends match, scrolling can be achieved by displaying the same image twice beside each other. The image scrolls to the left until the first of the two images has been totally scrolled out, leaving only the second image being displayed on the canvas. At this point, we reset the scrolling position of the first image back to zero and start the scrolling code again.

Example of a scrolling background image (Run Example).

scrolling_background_image.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.            */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                              */
/* It also holds game specific code, which will be different for each game      */




/******************** Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */
let scrollingBackgroundImage = new Image();
scrollingBackgroundImage.src = "images/scrolling_background.png";
/****************** END OF Declare game specific data and functions ****************/






/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                  */
    /* 2. store the game gameObjects in an array                    */
    /* 3. create a new Game to display the game gameObjects         */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[0] = new ScrollingBackgroundImage(scrollingBackgroundImage, 20);

    // make the canvas wider for this example
    document.getElementById("gameCanvas").style.width = "1000px";
    /* END OF game specific code. */

    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();


    /* If they are needed, then include any game-specific mouse and keyboard listners */
}
```

ScrollingBackgroundImage.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class ScrollingBackgroundImage extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image, updateStateMilliseconds)
```

```
    {
        super(updateStateMilliseconds); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;

        this.x = 0;
    }

    updateState()
    {
        this.x--;
        if (this.x <= -canvas.width)
        {
            this.x = 0;
        }
    }

    render()
    {
        ctx.drawImage(this.image, this.x, 0, canvas.width, canvas.height);
        ctx.drawImage(this.image, this.x + canvas.width, 0, canvas.width, canvas.height);
    }
}
```

Code Explained

In updateState(), x is continuously decremented until the first image is fully off the canvas. At this point, x is reset to 0, as shown below.

```
    updateState()
    {
        this.x--;
        if (this.x <= -canvas.width)
        {
            this.x = 0;
        }
    }
```

The same image is drawn twice on the canvas.

```
    render()
    {
        ctx.drawImage(this.image, this.x, 0, canvas.width, canvas.height);
        ctx.drawImage(this.image, this.x + canvas.width, 0, canvas.width, canvas.height);
    }
```

Scrolling Parallax Images

Parallax is a 3D effect that is achieved using scrolling 2D images. Instead of using just one background image, a scene's background can be comprised of several overlaid images. Parallax is achieved if the foreground image scrolls fastest and each image that represents a depth that is further away from the viewer scrolls progressively slower, as shown in the example below.

Example of a parallax background (Run Example).

scrolling_parallax_images.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.            */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                              */
/* It also holds game specific code, which will be different for each game      */




/******************* Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */
let skyImage = new Image();
skyImage.src = "images/scrolling_background_sky.png";
let middleImage = new Image();
middleImage.src = "images/scrolling_background_middle.png";
let foregroundImage = new Image();
foregroundImage.src = "images/scrolling_background_foreground.png";
/****************** END OF Declare game specific data and functions ****************/







/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[0] = new ScrollingBackgroundImage(skyImage, 30);
    gameObjects[1] = new ScrollingBackgroundImage(middleImage, 40);
    gameObjects[2] = new ScrollingBackgroundImage(foregroundImage, 50);

    // make the canvas wider for this example
    document.getElementById("gameCanvas").style.width = "1000px";

    /* END OF game specific code. */

    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();


    /* If they are needed, then include any game-specific mouse and keyboard listners */
}
```

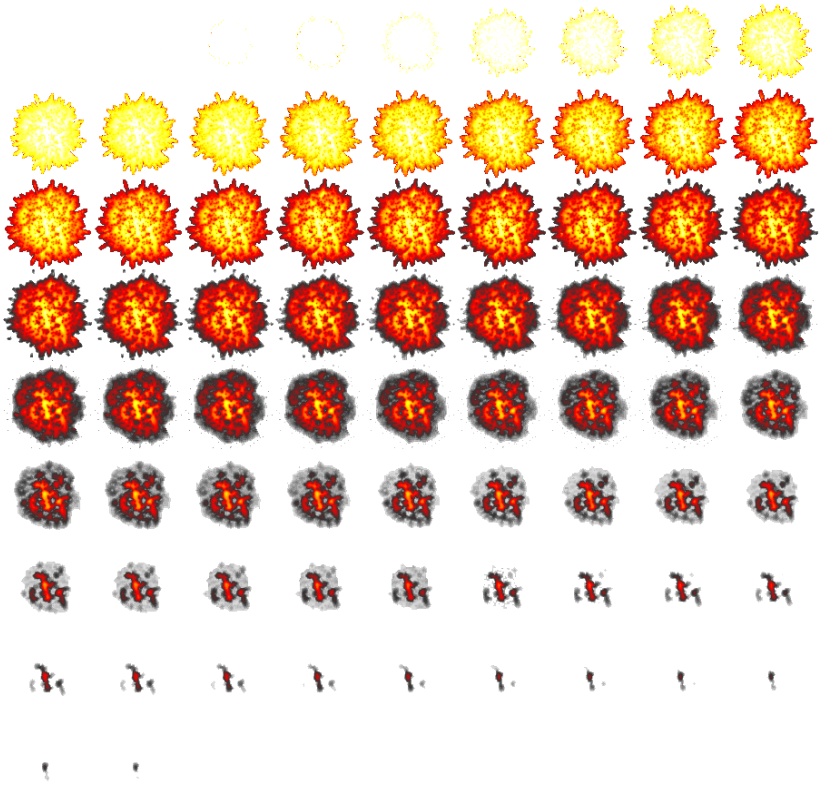
Code Explained

In this example, we have three overlaid images, each moving at different speeds

```
    gameObjects[0] = new ScrollingBackgroundImage(skyImage, 30);
    gameObjects[1] = new ScrollingBackgroundImage(middleImage, 40);
    gameObjects[2] = new ScrollingBackgroundImage(foregroundImage, 50);
```

Animated Images

Animated images (gameObject images) are images that contain multiple smaller images, which can be displayed in sequence to produce an animation. An example of an animated image is shown below:



Example of an exploding Animation (Run Example)

Explosion

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */
/* The Explosion GameObject is an animated gameObject of an explosion    */

class Explosion extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(explosionImage, explosionSound, centreX, centreY, size, delay = 0)
    {
        super(40, delay); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.explosionImage = explosionImage;
        this.explosionSound = explosionSound;
        this.centreX = centreX;
        this.centreY = centreY;
        this.size = size;
        this.delay = delay;
        this.NUMBER_OF_SPRITES = 74; // the number of gameObjects in the gameObject image
        this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE = 9; // the number of columns in the gameObject image
        this.NUMBER_OF_ROWS_IN_SPRITE_IMAGE = 9; // the number of rows in the gameObject image
        this.START_ROW = 0;
        this.START_COLUMN = 0;

        this.currentgameObject = 0; // the current gameObject to be displayed from the gameObject image
        this.row = this.START_ROW; // current row in gameObject image
        this.column = this.START_COLUMN; // current column in gameObject image

        this.SPRITE_WIDTH = (this.explosionImage.width / this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE);
        this.SPRITE_HEIGHT = (this.explosionImage.height / this.NUMBER_OF_ROWS_IN_SPRITE_IMAGE);
```

```
        this.isFirstCallOfUpdateState = true; // used to synchronise explosion sound with start of animation
    }

    updateState()
    {
        if (this.isFirstCallOfUpdateState)
        {
            this.explosionSound.currentTime = 0;
            this.explosionSound.play();
            this.isFirstCallOfUpdateState = false;
        }

        if (this.currentgameObject === this.NUMBER_OF_SPRITES)
        {
            this.stopAndHide();
        }
        this.currentgameObject++;

        this.column++;
        if (this.column >= this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE)
        {
            this.column = 0;
            this.row++;
        }
    }

    render()
    {
        ctx.drawImage(this.explosionImage, this.column * this.SPRITE_WIDTH, this.row * this.SPRITE_WIDTH, this.SPRITE_WI
    }
}
```

Code Explained

In order to animate a gameObject, we need to be able step through each sub-image in turn. We initialise the gameObject image based on the number of columns and rows that it contains, as shown in the code below:

```
        this.NUMBER_OF_SPRITES = 74; // the number of gameObjects in the gameObject image
        this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE = 9; // the number of columns in the gameObject image
        this.NUMBER_OF_ROWS_IN_SPRITE_IMAGE = 9; // the number of rows in the gameObject image
        this.START_ROW = 0;
        this.START_COLUMN = 0;

        this.currentgameObject = 0; // the current gameObject to be displayed from the gameObject image
        this.row = this.START_ROW; // current row in gameObject image
        this.column = this.START_COLUMN; // current column in gameObject image

        this.SPRITE_WIDTH = (this.explosionImage.width / this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE);
        this.SPRITE_HEIGHT = (this.explosionImage.height / this.NUMBER_OF_ROWS_IN_SPRITE_IMAGE);
```

We step through the various sub-images in turn, as shown in red in the code below. After all of the sub-images have been displayed, we kill the Explosion object, so that it no longer displays on the canvas.

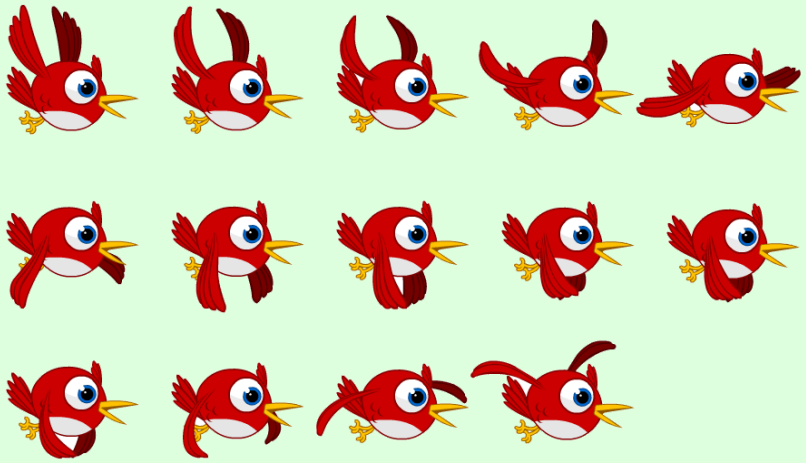
```
    updateState()
    {
        if (this.isFirstCallOfUpdateState)
        {
            this.explosionSound.currentTime = 0;
            this.explosionSound.play();
            this.isFirstCallOfUpdateState = false;
        }

        if (this.currentgameObject === this.NUMBER_OF_SPRITES)
        {
            this.stopAndHide();
        }
        this.currentgameObject++;

        this.column++;
        if (this.column >= this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE)
        {
            this.column = 0;
            this.row++;
        }
    }
```

We can synchronise an audio clip to play with the animation, as shown in blue in the code below. The flag this.isFirstCallOfUpdateState is set as soon as the animation starts to play.

```
    updateState()
    {
        if (this.isFirstCallOfUpdateState)
        {
            this.explosionSound.currentTime = 0;
            this.explosionSound.play();
            this.isFirstCallOfUpdateState = false;
        }

        if (this.currentgameObject === this.NUMBER_OF_SPRITES)
        {
            this.stopAndHide();
        }
        this.currentgameObject++;

        this.column++;
        if (this.column >= this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE)
        {
            this.column = 0;
            this.row++;
        }
    }
```

Exercises

Write code to show three animated birds flying across a scrolling background, as shown here. The bird image is:



Mouse Events

It is possible to detect mouse and keyboard events for a canvas (or any other HTML element). There are seven mouse events:

click
The event occurs when the mouse is clicked on the canvas.

dblclick
The event occurs when the mouse is double-clicked on the canvas.

mousemove
The event occurs when the mouse is moved while it is over the canvas.

mouseover
The event occurs when the mouse is moved onto the canvas.

mouseout
The event occurs when the mouse is moved out of the canvas.

mousedown
The event occurs when any mouse button is pressed over the canvas.

mouseup
The event occurs when any mouse button is released over the canvas.

We need to insert two pieces of code in order to use a mouse event.

We need to associate the event function with the canvas. This is done by adding an event listener to the canvas.

```
canvas.addEventListener('click', function(e){});
```

The variable, 'e' is provided by the system. It provides access to the x and y location of the screen pixel that was clicked. To get the canvas pixel location, we need to subtract the canvas top left corner x and y value from the screen x and y. The example below gets the correct canvas x and y location.

Example showing how to get the canvas x and y position (Run Example).
Note that this example does not need to use the framework classes, as nothing is being drawn on the canvas!

```
<!-- Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. -->

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>GENIUS worked example</title>
        <link rel="shortcut icon" type="image/png" href="images/genius_icon.png"/>
        <link href="css/game.css" rel="stylesheet" type="text/css"/>

        <style>
            #gameCanvas
            {
                margin-top:100px;
                margin-left: 100px;
            }
        </style>

        <script>
            let mouseX = 0;
            let mouseY = 0;

            window.onload = function ()
            {
                let canvas = document.getElementById("gameCanvas");
                canvas.width = canvas.clientWidth;
                canvas.height = canvas.clientHeight;
                let ctx = canvas.getContext("2d");

                canvas.addEventListener('click', function (e)
                {
                    if (e.which === 1)
                    {
                        var canvasBoundingRectangle = canvas.getBoundingClientRect();
                        mouseX = e.clientX - canvasBoundingRectangle.left;
                        mouseY = e.clientY - canvasBoundingRectangle.top;

                        alert("x:" + mouseX + "    y:" + mouseY);
                    }
                });
            };
        </script>
    </head>
```

```
    <body>
        <canvas id = "gameCanvas" tabindex="1"></canvas>
        <p>Click the mouse on the canvas to get the mouse x,y position within the canvas.<br>
            The coordinates will still work after the browser has been resized. Try resizing the browser to test this.<b
            Note that this canvas is 500 by 500 pixels.</p>
    </body>
</html>
```

Code Explained

The canvas is moved away from the document top-left corner for this example, so as to show that our code can detect the mouse coordinates **inside** the canvas.

```
        <style>
            #gameCanvas
            {
                margin-top:100px;
                margin-left: 100px;
            }
        </style>
```

A mouse click handler is added to the canvas.

```
                canvas.addEventListener('click', function (e)
                {
                    if (e.which === 1)
                    {
                        var canvasBoundingRectangle = canvas.getBoundingClientRect();
                        mouseX = e.clientX - canvasBoundingRectangle.left;
                        mouseY = e.clientY - canvasBoundingRectangle.top;

                        alert("x:" + mouseX + "    y:" + mouseY);
                    }
                });
```

If the left mouse button has been clicked, then e.which will hold the value 1.

```
            canvas.addEventListener('click', function (e)
            {
                if (e.which === 1)
                {

                }
            }
```

The mouse click returns the screen x and y values rather than the canvas values. The highlighted code below converts from screen to canvas coordinates.

```
            canvas.addEventListener('click', function (e)
            {
                if (e.which === 1)
                {
                    var canvasBoundingRectangle = canvas.getBoundingClientRect();
                    mouseX = e.clientX - canvasBoundingRectangle.left;
                    mouseY = e.clientY - canvasBoundingRectangle.top;

                    alert("x:" + mouseX + "    y:" + mouseY);
                }
            }
```

In the example below, an image is drawn at the location of a mouse click.

Example of a mouse click event class (Run Example)

event_mouse_click.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.            */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                              */
```

```
/* It also holds game specific code, which will be different for each game       */



/******************** Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */


/****************** END OF Declare game specific data and functions ****************/




/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[0] = new EventMouseClick();

    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();

    /* add event listners for input changes */
    document.addEventListener("click", function (e)
    {
        let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
        let mouseX = e.clientX - canvasBoundingRectangle.left;
        let mouseY = e.clientY - canvasBoundingRectangle.top;

        gameObjects[0].setX(mouseX);
        gameObjects[0].setY(mouseY);
    });
}
```

Code Explained

In the file "**event_mouse_click.js**", whenever the user clicks the mouse, we convert it from screen coordinates (shown in red). We then pass the canvas coordinates to the the gameObject (shown in blue).

```
    /* add event listners for input changes */
    document.addEventListener("click", function (e)
    {
        let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
        let mouseX = e.clientX - canvasBoundingRectangle.left;
        let mouseY = e.clientY - canvasBoundingRectangle.top;

        gameObjects[0].setX(mouseX);
        gameObjects[0].setY(mouseY);
    });
```

In the file "EventMouseClick.js", we use two setter methods to set the position of the gameObject whenever the user clicks the mouse. In the code below, **this.x** and **this.y** are the top-left corner of the gameObject.

EventMouseClick.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class EventMouseClick extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor()
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.width = 100;
        this.height = 100;
        this.x = 100;
        this.y = 100;

    }

    render()
    {
        ctx.fillStyle = 'black';
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }

    setX(newCentreX)
    {
        this.x = newCentreX - (this.width / 2);
    }

    setY(newCentreY)
    {
        this.y = newCentreY - (this.height / 2);
    }
}
```

Code Explained

The black rectangle has its centre set to be the new x and y positions, as shown below.

```
    setX(newCentreX)
    {
        this.x = newCentreX - (this.width / 2);
    }

    setY(newCentreY)
    {
        this.y = newCentreY - (this.height / 2);
    }
```

Exercises

Write code to draw an image in a new location whenever the user clicks on the canvas, as shown in this link.

Dragging An Image

Example that detects if the mouse is inside an image and takes the x and y offsets into account when dragging an image (Run Example)

drag_image.js

```
&/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.            */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */
```

```
/****************** Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */
let beachImage = new Image();
beachImage.src = "images/beach.png";
/****************** END OF Declare game specific data and functions ****************/




/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[0] = new DragImage(beachImage);

    /* END OF game specific code. */

    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();

    /* add event listners for input changes */
    document.addEventListener("mousedown", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            if (gameObjects[0].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[0].setOffsetX(mouseX);
                gameObjects[0].setOffsetY(mouseY);
            }
        }
    });

    document.addEventListener("mousemove", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            if (gameObjects[0].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[0].setX(mouseX);
                gameObjects[0].setY(mouseY);
            }
        }
    });
}
```

Code Explained

The mousedown event handler function is used to initialise the offset of the mouse from the top-left corner of the image. This is needed to ensure that the dragging of the image is smooth. Without this, the image top-left corner would jump to the current mouse position when we start to drag an image.

The mousemove event handler function is used to drag the image.

If we need to detect if the mouse is inside a GameObject, we need to write a method within the GameObject class to deal with this. We can call the method, as shown below:

```
if (gameObjects[0].pointIsInsideBoundingRectangle(mouseX, mouseY))
```

DragImage.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class DragImage extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;
        this.width = 100;
        this.height = 100;
        this.x = 100;
        this.y = 100;
        this.offsetX = 0;
        this.offsetY = 0;
    }

    render()
    {
        ctx.drawImage(this.image, this.x, this.y, this.width, this.height);
        ctx.strokeStyle = 'black';
        ctx.strokeRect(this.x - 1, this.y - 1, this.width + 2, this.height + 2);
    }

    setX(newMouseX)
    {
        this.x = newMouseX - this.offsetX;
    }

    setY(newMouseY)
    {
        this.y = newMouseY - this.offsetY;
    }

    setOffsetX(newMouseX)
    {
        this.offsetX = newMouseX - this.x;
    }

    setOffsetY(newMouseY)
    {
        this.offsetY = newMouseY - this.y;
    }

    pointIsInsideBoundingRectangle(pointX, pointY)
    {
        if ((pointX > this.x) && (pointY > this.y))
        {
            if (pointX > this.x)
            {
                if ((pointX - this.x) > this.width)
                {
                    return false; // to the right of this gameObject
                }
            }

            if (pointY > this.y)
            {
                if ((pointY - this.y) > this.height)
                {
                    return false; // below this gameObject
                }
            }
        }
```

```
        else // above or to the left of this gameObject
        {
            return false;
        }
        return true; // inside this gameObject
    }
}
```

Code Explained

Before we can drag an image, we need to be able to detect:

1. if the mouse is inside the image
2. the mouse offset position relative to the top-left corner of the image

The code below will detect if the location (x, y) is positioned inside an image.

DragImage.js

```
pointIsInsideBoundingRectangle(pointX, pointY)
    {
        if ((pointX > this.x) && (pointY > this.y))
        {
            if (pointX > this.x)
            {
                if ((pointX - this.x) > this.width)
                {
                    return false; // to the right of this gameObject
                }
            }

            if (pointY > this.y)
            {
                if ((pointY - this.y) > this.height)
                {
                    return false; // below this gameObject
                }
            }
        }
        else // above or to the left of this gameObject
        {
            return false;
        }
        return true; // inside this gameObject
    }
```

The offset needs to be calculated when the mouse is pressed down on an image. The code below will calculate the offsetX and offsetY of the mouse within an image.

drag_image.js

```
    document.addEventListener("mousedown", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            if (gameObjects[0].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[0].setOffsetX(mouseX);
                gameObjects[0].setOffsetY(mouseY);
            }
        }
    });
```

Whenever an image changes position, the offset needs to be taken into account when calculating the new image top-left x and y positions. This is done by subtracting the offset values that were calculated in the mousedownHandler(e) code above.

DragImage.js

```
    setOffsetX(newMouseX)
    {
        this.offsetX = newMouseX - this.x;
    }

    setOffsetY(newMouseY)
    {
        this.offsetY = newMouseY - this.y;
    }
```

Exercises

Write code to make a drawing tool that is similar to the one shown at this link. Hint: Use two gameObjects, one each for the image and the scribble and use an offscreen canvas for the Scribble

Mouse Wheel Event

We can associate a function with the mouse wheel using the code below. Unfortunately, Firefox deals with the mousewheel differently to other browsers, so we must add a separate handler for it.

```
// IE, Chrome, Safari, Opera
document.getElementById('gameCanvas').addEventListener("mousewheel", mouseWheelHandler, false);

// Firefox
document.getElementById('gameCanvas').addEventListener("DOMMouseScroll", mouseWheelHandler, false);
```

The function is provided with a system variable, e. This variable e.wheelDelta contains information relating to the mouse wheel. This value increments/decrements in units of 120. Therefore, we need to devide it by 120 to get a unit increment/decrement value.

```
function mouseWheelHandler(e)
{
  unitChange = e.wheelDelta / 120;  // unitChange will be equal to either +1 or -1

  // code to use the unitChange value is placed below
}
```

Example using the mouse wheel to scale an image (Run Example). In this example, the image will only scale if the mouse is over the image.

scale_image.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.            */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                              */
/* It also holds game specific code, which will be different for each game      */




/******************* Declare game specific global data and functions *****************/
/* images must be declared as global, so that they will load before the game starts  */
let beachImage = new Image();
beachImage.src = "images/beach.png";
/****************** END OF Declare game specific data and functions *****************/







/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                  */
    /* 2. store the game gameObjects in an array                    */
    /* 3. create a new Game to display the game gameObjects         */
```

```
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[0] = new ScaleImage(beachImage);

    /* END OF game specific code. */

    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();

    /* add event listners for input changes */
    // IE, Chrome, Safari, Opera
    document.getElementById('gameCanvas').addEventListener("mousewheel", mouseWheelHandler, false);
    // Firefox
    document.getElementById('gameCanvas').addEventListener("DOMMouseScroll", mouseWheelHandler, false);

    function mouseWheelHandler(e)
    {
        let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
        let mouseX = e.clientX - canvasBoundingRectangle.left;
        let mouseY = e.clientY - canvasBoundingRectangle.top;

        if (gameObjects[0].pointIsInsideBoundingRectangle(mouseX, mouseY))
        {
            unitChange = e.wheelDelta / 120;
            gameObjects[0].changeWidthAndHeight(unitChange);
        }
    }
}
```

ScaleImage.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class ScaleImage extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */
        /* These variables depend on the object */
        this.image = image;
        this.x = 200;
        this.y = 200;
        this.width = 100;
        this.height = 100;
    }

    render()
    {
        ctx.drawImage(this.image, this.x, this.y, this.width, this.height);
        ctx.strokeStyle = 'black';
        ctx.strokeRect(this.x - 1, this.y - 1, this.width + 2, this.height + 2);
    }

    changeWidthAndHeight(sizeChange)  // note that stepSize will be negative for scaling down
    {
        this.width += sizeChange;
        this.height += sizeChange;

        /* Scaling is about the centre of the image, so adust the x and y position to match new size */
        this.x -= sizeChange / 2;
        this.y -= sizeChange / 2;
    }

    pointIsInsideBoundingRectangle(pointX, pointY)
    {
        if ((pointX > this.x) && (pointY > this.y))
        {
            if (pointX > this.x)
            {
                if ((pointX - this.x) > this.width)
                {
                    return false; // to the right of this gameObject
                }
```

```
            }

            if (pointY > this.y)
            {
                if ((pointY - this.y) > this.height)
                {
                    return false; // below this gameObject
                }
            }
        }
        else // above or to the left of this gameObject
        {
            return false;
        }
        return true; // inside this gameObject
    }
}
```

Code Explained

The image has its width and height changed by the amount sizeChange. The image's x and y coordinates are adjusted to ensure that the scaling happens about the centre of the image.

```
    changeWidthAndHeight(sizeChange)  // note that stepSize will be negative for scaling down
    {
        this.width += sizeChange;
        this.height += sizeChange;

        /* Scaling is about the centre of the image, so adust the x and y position to match new size */
        this.x -= sizeChange / 2;
        this.y -= sizeChange / 2;
    }
```

The pointIsInsideBoundingRectangle() method ensures that scaling only occurs if the mouse if inside the image when the user attempts to scale.

```
    pointIsInsideBoundingRectangle(pointX, pointY)
    {
        ...
    }
```

Exercises

Write code to move, drag and scale an image, as shown in this link.

Keyboard Events

There are three keyboard events:

onkeydown
The event occurs when as soon as any key has been pressed.

onkeypress
The event occurs when any key is being pressed.

onkeyup
The event occurs when a key is being released.

As with the mouse event, we need to insert two pieces of code in order to use a keyboard event: a function containing the action and an event listener. Unlike the mouse event, the canvas cannot detect keyboard events. Therefore, we tie the keyboard event to the html document.

```
document.addEventListener("keydown", function(e){});
```

When using keyboard events, we need to be able to detect which key has been pressed. The code below will test if the left arrow has been pressed.

```
document.addEventListener("keydown", function (e)
{
    if (e.keyCode === 37)  // left arrow key has been pressed
    {
        // do something
    }
}
```

In the example above, the variable, 'e' is provided by the system. It provides access to the keyCode of the key that has been pressed. The complete set of keyCodes can be found at this link.

The example below allows the user to move an image around the canvas using the four arrow keys.

Example of a keyboard event (Run example)

key_down.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.            */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                              */
/* It also holds game specific code, which will be different for each game      */




/******************* Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */
let beachImage = new Image();
beachImage.src = "images/beach.png";
/****************** END OF Declare game specific data and functions ****************/




/************** Declare data and functions that are needed for all games ************/

/* Always create a canvas and a ctx */
canvas = document.getElementById("gameCanvas");
ctx = canvas.getContext("2d");
canvasWidth = canvas.clientWidth;
canvasHeight = canvas.clientHeight;

/* Always create an array that holds the default game gameObjects */
let gameObjects = [];

/*********** END OF Declare data and functions that are needed for all games *********/
```

```
/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[0] = new KeyDown(beachImage);

    /* END OF game specific code. */

    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();

    /* add event listners for input changes */
    document.addEventListener("keydown", function (e)
    {
        var stepSize = 10;

        if (e.keyCode === 37)  // left
        {
            gameObjects[0].changeX(-stepSize);
        }
        else if (e.keyCode === 38) // up
        {
            gameObjects[0].changeY(-stepSize);
        }
        else if (e.keyCode === 39) // right
        {
            gameObjects[0].changeX(stepSize);
        }
        else if (e.keyCode === 40) // down
        {
            gameObjects[0].changeY(stepSize);
        }
    });
}
```

Code Explained

```
    document.addEventListener("keydown", function (e)
    {
        var stepSize = 10;

        if (e.keyCode === 37)  // left
        {
            gameObjects[0].changeX(-stepSize);
        }
        else if (e.keyCode === 38) // up
        {
            gameObjects[0].changeY(-stepSize);
        }
        else if (e.keyCode === 39) // right
        {
            gameObjects[0].changeX(stepSize);
        }
        else if (e.keyCode === 40) // down
        {
            gameObjects[0].changeY(stepSize);
        }
    });
```

The keydown event checks against the four arrow keys. It changes the x or y position of the image depending on which key was pressed.

KeyDown.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class KeyDown extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;
        this.width = 100;
        this.height = 100;
        this.x = 100;
        this.y = 100;
    }

    render()
    {
        ctx.drawImage(this.image, this.x, this.y, this.width, this.height);
        ctx.strokeStyle = 'black';
        ctx.strokeRect(this.x - 1, this.y - 1, this.width + 2, this.height + 2);
    }

    changeX(changeSize)
    {
        this.x += changeSize;
    }

    changeY(changeSize)
    {
        this.y += changeSize;
    }
}
```

Code Explained

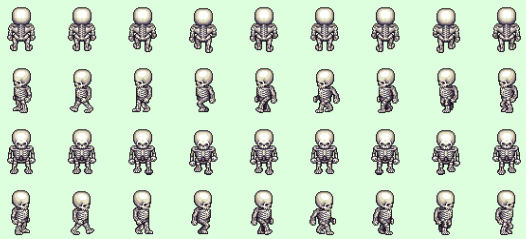
```
    changeX(changeSize)
    {
        this.x += changeSize;
    }

    changeY(changeSize)
    {
        this.y += changeSize;
    }
```

The gameObject can change x or y.

Exercises

Amend the above code to stop the image moving off the canvas.

Write code to animate a gameObject character walking, as shown here. The gameObject image is below:

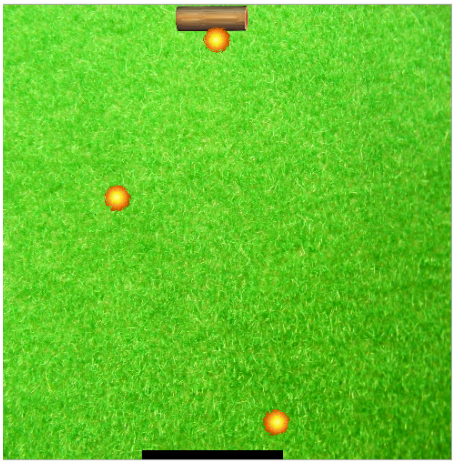


Case Study... Fireball Game

The aim of this game is to use the fireballs to destroy the log without letting any falling fireball hit the bat.

This game shows:

- collision detection using a bounding rectangle
- keyboard input
- gameObject rotations
- how to programme gameObjects outside of the gameObjects[] array



Play game

Download .zip file

fireball_game.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.             */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */




/******************** Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */
let backgroundImage = new Image();
backgroundImage.src = "images/grass.png";

let logImage = new Image();
logImage.src = "images/log.png";

let fireballImage = new Image();
fireballImage.src = "images/fireball.png";

const BACKGROUND = 0;
const WIN_LOSE_MESSAGE = 1;

/* Instead of using gameObject[], we can declare our own gameObject variables */
let bat = null; // we cannot initialise gameObjects yet, as they might require images that have not yet loaded
let target = null;

let fireballs = [];
let numberOfBulletsFired = 0; // no bullets fired yet
/****************** END OF Declare game specific data and functions ****************/







/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
```

```
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* Specifically, this function will:                                */
    /* 1. initialise the canvas and associated variables                */
    /* 2. create the various game gameObjects,                  */
    /* 3. store the gameObjects in an array                     */
    /* 4. create a new Game to display the gameObjects          */
    /* 5. start the Game                                                */



    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[BACKGROUND] = new StaticImage(backgroundImage, 0, 0, canvas.width, canvas.height);
    bat = new Bat(0, canvas.height - 10, 125);
    target = new Target(logImage, 100, 0, 100);
    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new FireballCanvasGame();

    /* Always play the game */
    game.start();


    /* If they are needed, then include any game-specific mouse and keyboard listners */
    document.addEventListener("keydown", function (e)
    {
        var stepSize = 10;

        if (e.keyCode === 37)  // left
        {
            bat.changeX(-stepSize);
        }
        else if (e.keyCode === 39) // right
        {
            bat.changeX(stepSize);
        }
        else if (e.keyCode === 32) // space bar
        {
            fireballs[numberOfBulletsFired] = new Fireball(fireballImage, bat.getCentreX());
            fireballs[numberOfBulletsFired].start();
            numberOfBulletsFired++;
            bat.setWidth(bat.getWidth() + 10);
        }
    });
}
```

Code Explained

The fireballs are not contained in the gameObjects[] array. Instead, they are stored in a fireballs[] array.

```
let fireballs = [];
```

A new fireball is added each time the user hits the space bar. The fireball fires from the centre of the bat.
In this game, the bat gets bigger each time a fireball is fired.

```
        else if (e.keyCode === 32) // space bar
        {
            fireballs[numberOfBulletsFired] = new Fireball(fireballImage, bat.getCentreX());
            fireballs[numberOfBulletsFired].start();
            numberOfBulletsFired++;
            bat.setWidth(bat.getWidth() + 10);
        }
```

FireballCanvasGame.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.                                  *
/* The CanvasGame class is responsible for rendering all of the gameObjects and other game graphics on the canvas.
/* If you want to implement collision detection in your game, then you MUST overwrite the collisionDetection() method. *
/* This class will usually not change.                                                                *
```

```
class FireballCanvasGame extends CanvasGame
{
    constructor()
    {
        super();
    }

    collisionDetection()
    {
        for (let i = 0; i < numberOfBulletsFired; i++)
        {
            if (target.pointIsInsideBoundingRectangle(fireballs[i].getCentreX(), fireballs[i].getCentreY()))
            {
                target.setWidth(target.getWidth() - 10);
                target.setX(Math.random() * (canvas.width - target.getWidth()));

                if (target.getWidth() < target.getMinimumSize())
                {
                    /* Player has won */
                    for (let i = 0; i < fireballs.length; i++) /* stop all gameObjects from animating */
                    {
                        fireballs[i].stop();
                    }
                    gameObjects[WIN_LOSE_MESSAGE] = new StaticText("Win!", 150, 270, "Times Roman", 100, "black");
                    gameObjects[WIN_LOSE_MESSAGE].start(); /* render win message */
                }
            }
            else if (bat.pointIsInsideBoundingRectangle(fireballs[i].getCentreX(), fireballs[i].getCentreY()))
            {
                /* Player has lost */
                for (let i = 0; i < fireballs.length; i++) /* stop all gameObjects from animating */
                {
                    fireballs[i].stop();
                }
                gameObjects[WIN_LOSE_MESSAGE] = new StaticText("LOSE!", 100, 270, "Times Roman", 100, "red");
                gameObjects[WIN_LOSE_MESSAGE].start(); /* render lose message */
            }
        }
    }

    render()
    {
        super.render();

        bat.render();
        target.render();
        for (let i = 0; i < fireballs.length; i++)
        {
            fireballs[i].render();
        }
    }
}
```

Code Explained

The collision detection checks each of the fireballs in the fireballs[] array to see if it collides with the target.

```
    collisionDetection()
    {
        if (this.gameState === PLAYING)
        {
            for (let i = 0; i < numberOfBulletsFired; i++)
            {
                if (target.pointIsInsideBoundingRectangle(fireballs[i].getCentreX(), fireballs[i].getCentreY()))
                {
                    ...
```

The fireballs are checked against the bounding rectangle of the target. If there is a collision between a fireball and the target, then the target reduces in size and moves.
If the target becomes smaller that target.getMinimumSize(), then the player wins.

```
                if (target.pointIsInsideBoundingRectangle(fireballs[i].getCentreX(), fireballs[i].getCentreY()))
                {
                    target.setWidth(target.getWidth() - 10);
                    target.setX(Math.random() * (500 - target.getWidth()));

                    if (target.getWidth() < target.getMinimumSize())
                    {
```

```
                    this.gameState = WON;
                }
            }
```

The fireballs are checked against the bat's bounding rectangle. If a fireball hits the bat, then the player loses.

```
            else if (bat.pointIsInsideBoundingRectangle(fireballs[i].getCentreX(), fireballs[i].getCentreY()))
            {

                this.gameState = LOST;
            }
```

The game needs to render the background gameObjects. This is done by calling its parent class's super.render() method.
The bat, target and fireballs rendering methods are called directly, as shown below.

```
    render()
    {
        if (this.gameState === PLAYING)
        {
            super.render();

            bat.render();
            target.render();
            for(let i=0;i < fireballs.length;i++)
            {
                fireballs[i].render();
            }
        }
        ...
```

Fireball.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class Fireball extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image, centreX)
    {
        super(5); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;
        this.width = 30;
        this.height = 30;
        this.centreX = centreX;
        this.centreY = canvas.height - this.height - 1;
        this.stepSize = -1;
        this.rotation = 360;
    }

    updateState()
    {
        this.rotation -= 3;
        if (this.rotation < 1)
        {
            this.rotation = 360;
        }

        if (this.stepSize < 0)
        {
            this.centreY--;
            if (this.centreY < 0)
            {
                this.stepSize = 1;
            }
        }
        else // this.stepSize >= 0
        {
            this.centreY++;
            if (this.centreY > canvas.height)
            {
                this.stepSize = -1;
            }
        }
    }
```

```
    render()
    {
        ctx.save();
        ctx.translate(this.centreX, this.centreY);
        ctx.rotate(Math.radians(this.rotation));
        ctx.translate(-this.centreX, -this.centreY);

        ctx.drawImage(this.image, this.centreX - this.width / 2, this.centreY - this.width / 2, this.width, this.height)
        ctx.restore();
    }

    getCentreX()
    {
        return this.centreX;
    }

    getCentreY()
    {
        return this.centreY;
    }
}
```

Code Explained

The fireballs rotate when they are moving. Firstly, the rotation amount is set inside the updateState() method. The -3 means that the fireball will rotate -3 degrees each time updateState() is called.

```
    updateState()
    {
        this.rotation -= 3;
        if (this.rotation < 1)
        {
            this.rotation = 360;
        }
        ...
```

Secondly, the canvas is rotated by the rotation amount when the fireball is being drawn on the canvas.

```
    render()
    {
        ctx.save();
        ctx.translate(this.centreX, this.centreY);
        ctx.rotate(Math.radians(this.rotation));
        ctx.translate(-this.centreX, -this.centreY);

        ctx.drawImage(this.image, this.centreX - this.width / 2, this.centreY - this.width / 2, this.width, this.height)
        ctx.restore();
    }
```

The fireballs change direction when they hit the top or bottom of the canvas.

```
        if (this.stepSize < 0)
        {
            this.centreY--;
            if (this.centreY < 0)
            {
                this.stepSize = 1;
            }
        }
        else // this.stepSize >= 0
        {
            this.centreY++;
            if (this.centreY > canvas.height)
            {
                this.stepSize = -1;
            }
        }
```

Bat.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class Bat extends GameObject
```

```
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(x, y, width)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = 10;
    }

    render()
    {
        ctx.fillStyle = 'black';
        ctx.fillRect(this.x, this.y, this.width, this.height);
    }

    changeX(changeAmount)
    {
        this.x += changeAmount;

        /* Ensure that only half of the bat can be off the screen                              */
        /* This ensures that the bat can still fire at a log that is on the edge of the screen, */
        /* while at the same time the bat cannot hide fully from oncoming fireballs.           */
        if(this.x > canvas.width - (this.width / 2))
        {
            this.x = canvas.width - (this.width / 2);
        }
        else if(this.x < -(this.width / 2))
        {
            this.x = -(this.width / 2);
        }
    }

    getWidth()
    {
        return this.width;
    }

    setWidth(newWidth)
    {
        this.width = newWidth;
    }

    getCentreX()
    {
        return this.x + this.width / 2;
    }

    pointIsInsideBoundingRectangle(pointX, pointY)
    {
        if ((pointX > this.x) && (pointY > this.y))
        {
            if (pointX > this.x)
            {
                if ((pointX - this.x) > this.width)
                {
                    return false; // to the right of this gameObject
                }
            }

            if (pointY > this.y)
            {
                if ((pointY - this.y) > this.height)
                {
                    return false; // below this gameObject
                }
            }
        }
        else // above or to the left of this gameObject
        {
            return false;
        }
        return true; // inside this gameObject
    }
}
```

Code Explained

The changeX() method ensures that the bat can never move fully off the canvas. As the fireballs are fired from the centre of the bat, it must be possible to move the centre of the bat to either "0" or "canvas.width". The changeX() method implements this.

```
    changeX(changeAmount)
    {
        this.x += changeAmount;

        /* Ensure that only half of the bat can be off the screen                              */
        /* This ensures that the bat can still fire at a log that is on the edge of the screen, */
        /* while at the same time the bat cannot hide fully from oncoming fireballs.           */
        if(this.x > canvas.width - (this.width / 2))
        {
            this.x = canvas.width - (this.width / 2);
        }
        else if(this.x < -(this.width / 2))
        {
            this.x = -(this.width / 2);
        }
    }
```

Target.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class Target extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image, x, y, width)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = 30;

        this.minimumSize = 20;
    }

    render()
    {
        ctx.drawImage(this.image, this.x, this.y, this.width, this.height);
    }

    getX()
    {
        return this.x;
    }

    getY()
    {
        return this.y;
    }

    getWidth()
    {
        return this.width;
    }

    setX(newX)
    {
        this.x = newX;
    }

    setY(newY)
    {
        this.y = newY;
    }

    setWidth(newWidth)
    {
        this.width = newWidth;
    }
```

```
    getMinimumSize()
    {
        return this.minimumSize;
    }

    pointIsInsideBoundingRectangle(pointX, pointY)
    {
        if ((pointX > this.x) && (pointY > this.y))
        {
            if (pointX > this.x)
            {
                if ((pointX - this.x) > this.width)
                {
                    return false; // to the right of this gameObject
                }
            }

            if (pointY > this.y)
            {
                if ((pointY - this.y) > this.height)
                {
                    return false; // below this gameObject
                }
            }
        }
        else // above or to the left of this gameObject
        {
            return false;
        }
        return true; // inside this gameObject
    }
}
```

Code Explained

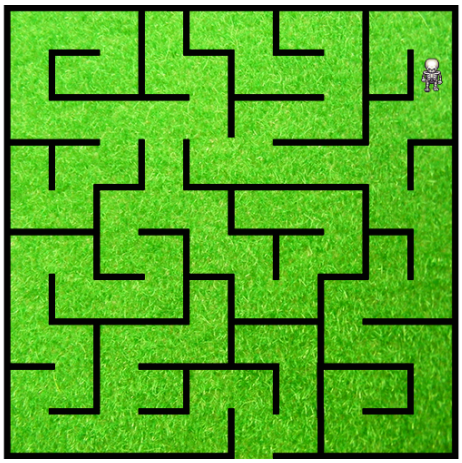
The code for Target.js is straightforward and needs no explaining.

Case Study... Maze Game

The aim of this game is to use the arrow keys to walk the skeleton character to the exit of the maze.

This game shows:

- collision detection using an offscreen canvas
- keyboard input
- an animated gameObject image



Play game

Download .zip file

maze_game.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.             */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */




/******************** Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */

let skeletonImage = new Image();
skeletonImage.src = "images/skeleton.png";

let background = new Image;
background.src = "images/maze_background.png";

let mazeGrid = new Image;
mazeGrid.src = "images/maze_grid.png";


/* Direction that the skeleton is walking */
/* Note that this matches the row in the gameObject image for the given direction */
const UP = 0;
const LEFT = 1;
const DOWN = 2;
const RIGHT = 3;
const STOPPED = 4;

/* The various gameObjects */
/* These are the positions that each gameObject is held in the gameObjects[] array */
const BACKGROUND = 0;
const MAZE = 1;
const SKELETON = 2;
const WIN_MESSAGE = 3;
/****************** END OF Declare game specific data and functions ****************/
```

```
/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */


    gameObjects[BACKGROUND] = new StaticImage(background, 0, 0, canvas.width, canvas.height);
    gameObjects[MAZE] = new StaticImage(mazeGrid, 0, 0, canvas.width, canvas.height);
    gameObjects[SKELETON] = new MazeSkeleton(skeletonImage, canvas.width - 70, 80);


    /* END OF game specific code. */


    /* Always create a game that uses the gameObject array */
    let game = new MazeSkeletonCanvasGame(mazeGrid);

    /* Always play the game */
    game.start();

    /* If they are needed, then include any game-specific mouse and keyboard listners */
    document.addEventListener('keydown', function (e)
    {
        if (e.keyCode === 37)  // left
        {
            gameObjects[SKELETON].setDirection(LEFT);
        }
        else if (e.keyCode === 38) // up
        {
            gameObjects[SKELETON].setDirection(UP);
        }
        else if (e.keyCode === 39) // right
        {
            gameObjects[SKELETON].setDirection(RIGHT);
        }
        else if (e.keyCode === 40) // down
        {
            gameObjects[SKELETON].setDirection(DOWN);
        }
    });
}
```

Code Explained

The game will display a background image, a grid and a skeleton

```
    gameObjects[BACKGROUND] = new StaticImage(background, 0, 0, canvas.width, canvas.height);
    gameObjects[MAZE] = new StaticImage(mazeGrid, 0, 0, canvas.width, canvas.height);
    gameObjects[SKELETON] = new MazeSkeleton(skeletonImage, canvas.width - 70, 80);
```

Because we have collision detection, we need to have our own CanvasGame class.

```
    let game = new MazeSkeletonCanvasGame(mazeGrid);
```

The keyboard is used to control the direction that the skeleton travels. The skeleton will continue to travel in the same direction until another arrow key is hit.

```
    document.addEventListener('keydown', function (e)
    {
        if (e.keyCode === 37)  // left
        {
            gameObjects[SKELETON].setDirection(LEFT);
        }
        else if (e.keyCode === 38) // up
        {
```

```
            gameObjects[SKELETON].setDirection(UP);
        }
        else if (e.keyCode === 39) // right
        {
            gameObjects[SKELETON].setDirection(RIGHT);
        }
        else if (e.keyCode === 40) // down
        {
            gameObjects[SKELETON].setDirection(DOWN);
        }
    });
```

MazeCanvasGame.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.       */
/* A CanvasGame that implements collision detection.                       */
/* The game allows the user to walk a skeleton around a maze.              */
/* If the skeleton is guided to the maze exit, then a win message appears. */

class MazeSkeletonCanvasGame extends CanvasGame
{
    constructor(mazeGridImage)
    {
        super();

        /* this.mazeCtx will be used for collision detection */
        let mazeOffscreenCanvas = document.createElement('canvas');
        this.mazeCtx = mazeOffscreenCanvas.getContext('2d');
        mazeOffscreenCanvas.width = canvas.width;
        mazeOffscreenCanvas.height = canvas.height;
        this.mazeCtx.drawImage(mazeGridImage, 0, 0, canvas.width, canvas.height);
    }

    collisionDetection()
    {
        if (!this.mazeCtx)
        {
            return;
        }
        if (gameObjects[SKELETON].getDirection() === UP)
        {
            let imageData = this.mazeCtx.getImageData(gameObjects[SKELETON].getCentreX(), gameObjects[SKELETON].getCentr
            if (imageData.data[3] !== 0)
            {
                gameObjects[SKELETON].setDirection(DOWN);
            }
        }
        else if (gameObjects[SKELETON].getDirection() === LEFT)
        {
            let imageData = this.mazeCtx.getImageData(gameObjects[SKELETON].getCentreX() - 15, gameObjects[SKELETON].get
            if (imageData.data[3] !== 0)
            {
                gameObjects[SKELETON].setDirection(RIGHT);
            }
        }
        else if (gameObjects[SKELETON].getDirection() === DOWN)
        {
            let imageData = this.mazeCtx.getImageData(gameObjects[SKELETON].getCentreX(), gameObjects[SKELETON].getCentr
            if (imageData.data[3] !== 0)
            {
                gameObjects[SKELETON].setDirection(UP);
            }

            if (gameObjects[SKELETON].getCentreY() > canvas.height)
            {
                /* Player has won */
                for (let i = 0; i < gameObjects.length; i++) /* stop all gameObjects from animating */
                {
                    gameObjects[i].stop();
                }
                gameObjects[WIN_MESSAGE] = new StaticText("Well Done!", 20, 280, "Times Roman", 100, "red");
                gameObjects[WIN_MESSAGE].start(); /* render win message */
            }
        }
        else if (gameObjects[SKELETON].getDirection() === RIGHT)
        {
            let imageData = this.mazeCtx.getImageData(gameObjects[SKELETON].getCentreX(), gameObjects[SKELETON].getCentr
            if (imageData.data[3] !== 0)
            {
                gameObjects[SKELETON].setDirection(LEFT);
```

```
            }
        }
    }
}
```

Code Explained

Set up an offscreen canvas to hold the grid image that is used for collision detection.

```
        /* this.mazeCtx will be used for collision detection */
        let mazeOffscreenCanvas = document.createElement('canvas');
        this.mazeCtx = mazeOffscreenCanvas.getContext('2d');
        mazeOffscreenCanvas.width = canvas.width;
        mazeOffscreenCanvas.height = canvas.height;
        this.mazeCtx.drawImage(mazeGridImage, 0, 0, canvas.width, canvas.height);
```

Do collision detection for the four mid-side points. For example, if we are detecting a collision as the skeleton moves up the screen, then we need to use the skeleton's (centreX, centreY - 20).
The reason why the '20' is a different number for UP, DOWN, LEFT and RIGHT is to allow for the differences of the gameObject size when it is displayed moving in different directions.

Whenever the skeleton hits the maze grid, if changes direction.

```
        if (gameObjects[SKELETON].getDirection() === UP)
        {
            let imageData = this.mazeCtx.getImageData(gameObjects[SKELETON].getCentreX(), gameObjects[SKELETON].getCentr
            if (imageData.data[3] !== 0)
            {
                gameObjects[SKELETON].setDirection(DOWN);
            }
        }
```

Stop all gameObjects from animating and display the win message.

```
/* Player has won */
                for (let i = 0; i < gameObjects.length; i++) /* stop all gameObjects from animating */
                {
                    gameObjects[i].stop();
                }
                gameObjects[WIN_MESSAGE] = new StaticText("Well Done!", 20, 280, "Times Roman", 100, "red");
                gameObjects[WIN_MESSAGE].start(); /* render win message */
```

MazeSkeleton.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class MazeSkeleton extends Skeleton
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(skeletonImage, centreX, centreY)
    {
        super(skeletonImage, centreX, centreY); /* as this class extends from GameObject, you must always call super() *

        /* These variables depend on the object */
        this.WIDTH_OF_SKELETON_ON_CANVAS = 50; /* the width and height that the skeleton will take up on the canvas */
        this.HEIGHT_OF_SKELETON_ON_CANVAS = 50;

        this.centreX = centreX; /* set the start position of the skeleton in the maze */
        this.centreY = centreY;

        this.SKELETON_SPEED = 1; /* set the skeleton's speed */
    }
}
```

Case Study... Monster Game

The aim of this game is to use the arrow keys to direct the monster to the bullseye.

This game shows:

- collision detection using an offscreen canvas
- keyboard input
- a shaking canvas screen effect

Play game

Download .zip file

monster_game.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.             */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */




/******************** Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */

let obstaclesImage = new Image();
obstaclesImage.src = "images/obstacles.png";

let monsterImage = new Image();
monsterImage.src = "images/monster.png";

let bullseyeImage = new Image();
bullseyeImage.src = "images/bullseye.png";

let starsImage = new Image();
starsImage.src = "images/stars.png";

/* Direction that the skeleton is walking */
/* Note that this matches the row in the gameObject image for the given direction */
const UP = 0;
const LEFT = 1;
const DOWN = 2;
const RIGHT = 3;
const STOPPED = 4;
const START = 5;

/* The various gameObjects */
/* These are the positions that each gameObject is held in the gameObjects[] array */
const BACKGROUND = 0;
const OBSTACLES = 1;
const BULLSEYE = 2;
const MONSTER = 3;
const WIN_MESSAGE = 4;
```

```
/****************** END OF Declare game specific data and functions ****************/




/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */

    gameObjects[BACKGROUND] = new StaticImage(starsImage, 0, 0, canvas.width, canvas.height);
    gameObjects[OBSTACLES] = new StaticImage(obstaclesImage, 0, 0, canvas.width, canvas.height);
    gameObjects[BULLSEYE] = new Bullseye(bullseyeImage, canvas.width - 50, canvas.height - 50, 50, 50);
    gameObjects[MONSTER] = new Monster(monsterImage);

    /* END OF game specific code. */

    /* Always create a game that uses the gameObject array */
    let game = new MonsterCanvasGame(obstaclesImage);

    /* Always play the game */
    game.start();

    /* If they are needed, then include any game-specific mouse and keyboard listners */
    document.addEventListener('keydown', function (e)
    {
        if (e.keyCode === 37)  // left
        {
            gameObjects[MONSTER].setDirection(LEFT);
        }
        else if (e.keyCode === 38) // up
        {
            gameObjects[MONSTER].setDirection(UP);
        }
        else if (e.keyCode === 39) // right
        {
            gameObjects[MONSTER].setDirection(RIGHT);
        }
        else if (e.keyCode === 40) // down
        {
            gameObjects[MONSTER].setDirection(DOWN);
        }
        else if (e.keyCode === 32) // space
        {
            gameObjects[MONSTER].setDirection(START);
        }
    });

    document.addEventListener('keyup', function (e)
    {
        gameObjects[MONSTER].setDirection(STOPPED);
    });
}
```

Code Explained

The game displays a monster, a background, a bullseye and obstacles.

```
    gameObjects[BACKGROUND] = new StaticImage(starsImage, 0, 0, canvas.width, canvas.height);
    gameObjects[OBSTACLES] = new StaticImage(obstaclesImage, 0, 0, canvas.width, canvas.height);
    gameObjects[BULLSEYE] = new Bullseye(bullseyeImage, canvas.width - 50, canvas.height - 50, 50, 50);
    gameObjects[MONSTER] = new Monster(monsterImage);
```

The arrow keys are used to control the direction that the monster moves.

```
document.addEventListener('keydown', function (e)
    {
        if (e.keyCode === 37)  // left
        {
            gameObjects[MONSTER].setDirection(LEFT);
        }
        else if (e.keyCode === 38) // up
        {
            gameObjects[MONSTER].setDirection(UP);
        }
        else if (e.keyCode === 39) // right
        {
            gameObjects[MONSTER].setDirection(RIGHT);
        }
        else if (e.keyCode === 40) // down
        {
            gameObjects[MONSTER].setDirection(DOWN);
        }
        else if (e.keyCode === 32) // space
        {
            gameObjects[MONSTER].setDirection(START);
        }
    });
```

The monster stops moving when the arrow keys are released.

```
    document.addEventListener('keyup', function (e)
    {
        gameObjects[MONSTER].setDirection(STOPPED);
    });
```

MonsterCanvasGame.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.       */
/* A CanvasGame that implements collision detection.                       */
/* The game allows the user to walk a skeleton around a maze.              */
/* If the skeleton is guided to the maze exit, then a win message appears. */


class MonsterCanvasGame extends CanvasGame
{
    constructor(obstaclesImage)
    {
        super();

        /* this.monsterObstalesCtx will be used for collision detection */
        let monsterObstalesOffscreenCanvas = document.createElement('canvas');
        monsterObstalesOffscreenCanvas.width = canvas.width;
        monsterObstalesOffscreenCanvas.height = canvas.height;
        this.monsterObstalesCtx = monsterObstalesOffscreenCanvas.getContext('2d');
        this.monsterObstalesCtx.drawImage(obstaclesImage, 0, 0, canvas.width, canvas.height);

        this.screenShakeInterval = null;
        this.screenIsRotatingToTheLeft = false;
        this.NUMBER_OF_SCREEN_SHAKES_INTERATIONS = 10;
        this.numberOfScreenShakes = 0;
    }

    collisionDetection()
    {
        if (!this.monsterObstalesCtx)
        {
            return;
        }

        let imageData = this.monsterObstalesCtx.getImageData(gameObjects[MONSTER].getX(), gameObjects[MONSTER].getY(), 1
        let dataTop = imageData.data;
        imageData = this.monsterObstalesCtx.getImageData(gameObjects[MONSTER].getX() + gameObjects[MONSTER].getWidth() *
        let dataRight = imageData.data;
        imageData = this.monsterObstalesCtx.getImageData(gameObjects[MONSTER].getX(), gameObjects[MONSTER].getY() + game
        let dataBottom = imageData.data;
        imageData = this.monsterObstalesCtx.getImageData(gameObjects[MONSTER].getX() + gameObjects[MONSTER].getWidth() *
        let dataLeft = imageData.data;
        if ((dataTop[3] !== 0) || (dataRight[3] !== 0) || (dataBottom[3] !== 0) || (dataLeft[3] !== 0))
        {
            if (gameObjects[MONSTER].getDirection() === UP)
            {
                gameObjects[MONSTER].setDirection(DOWN);
                gameObjects[MONSTER].setY(gameObjects[MONSTER].getY() + 5);
```

```
            }
            else if (gameObjects[MONSTER].getDirection() === DOWN)
            {
                gameObjects[MONSTER].setDirection(UP);
                gameObjects[MONSTER].setY(gameObjects[MONSTER].getY() - 5);
            }
            else if (gameObjects[MONSTER].getDirection() === LEFT)
            {
                gameObjects[MONSTER].setDirection(RIGHT);
            }
            else if (gameObjects[MONSTER].getDirection() === RIGHT)
            {
                gameObjects[MONSTER].setDirection(LEFT);
            }

            if (this.screenShakeInterval === null)
            {
                this.screenShakeInterval = setInterval(this.shakeScreen.bind(this), 10);
            }

        }
        else if (gameObjects[BULLSEYE].pointIsInsideBullseyeRectangle(gameObjects[MONSTER].getX() + gameObjects[MONSTER]
        {
            /* Player has won */
            for (let i = 0; i < gameObjects.length; i++) /* stop all gameObjects from animating */
            {
                gameObjects[i].stop();
            }
            gameObjects[WIN_MESSAGE] = new StaticText("Well Done!", 20, 280, "Times Roman", 100, "red");
            gameObjects[WIN_MESSAGE].start(); /* render win message */
        }
    }

    render()
    {
        ctx.save();
        if (this.screenShakeInterval !== null) // hit an obstacle
        {
            if (this.screenIsRotatingToTheLeft)
            {
                ctx.translate(canvas.width / 2, canvas.height / 2);
                ctx.rotate(Math.radians(1));
                ctx.translate(-canvas.width / 2, -canvas.height / 2);
            }
            else
            {
                ctx.translate(canvas.width / 2, canvas.height / 2);
                ctx.rotate(Math.radians(-1));
                ctx.translate(-canvas.width / 2, -canvas.height / 2);
            }
        }

        super.render();
        ctx.restore();
    }

    shakeScreen()
    {
        if (this.screenIsRotatingToTheLeft)
        {
            this.screenIsRotatingToTheLeft = false;
        }
        else // screen is rotating to the right
        {
            this.screenIsRotatingToTheLeft = true;
        }
        this.numberOfScreenShakes++;
        if (this.numberOfScreenShakes >= this.NUMBER_OF_SCREEN_SHAKES_INTERATIONS)
        {
            this.numberOfScreenShakes = 0;
            clearInterval(this.screenShakeInterval);
            this.screenShakeInterval = null;
        }
    }
}
```

Code Explained

Whenever the skeleton hits the maze grid, if changes direction.

```
            /* Player has won */
            for (let i = 0; i < gameObjects.length; i++) /* stop all gameObjects from animating */
```

```
            {
                gameObjects[i].stop();
            }
            gameObjects[WIN_MESSAGE] = new StaticText("Well Done!", 20, 280, "Times Roman", 100, "red");
            gameObjects[WIN_MESSAGE].start(); /* render win message */
```

Monster.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class Monster extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(monsterImage)
    {
        super(5); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.startX = 20;
        this.startY = 20;
        this.x = this.startX;
        this.y = this.startY;

        this.monsterImage = monsterImage;
        this.width = 40;
        this.height = 40;
        this.setDirection(STOPPED);
    }

    updateState()
    {
        if (this.direction === UP)
        {
            this.y--;
        }
        else if (this.direction === LEFT)
        {
            this.x--;
        }
        else if (this.direction === DOWN)
        {
            this.y++;
        }
        else if (this.direction === RIGHT)
        {
            this.x++;
        }
    }

    render()
    {
        ctx.drawImage(this.monsterImage, this.x, this.y, this.width, this.height);
    }

    setDirection(newDirection)
    {
        if (this.direction !== START)
        {
            this.direction = newDirection;
        }
        else // spacebar hit, so set monster back to start
        {
            this.x = this.startX;
            this.y = this.startY;
            this.direction = STOPPED;
        }
    }

    getDirection()
    {
        return(this.direction);
    }

    getX()
    {
        return this.x;
    }
```

```
    getY()
    {
        return this.y;
    }

    setX(newX)
    {
        this.x = newX;
    }

    setY(newY)
    {
        this.y = newY;
    }
    getWidth()
    {
        return this.width;
    }

    getHeight()
    {
        return this.height;
    }
}
```

Code Explained

Bullseye.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class Bullseye extends StaticImage
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(image, x, y, width, height)
    {
        super(image, x, y, width, height); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.image = image;
        this.width = width;
        this.height = height;
        this.x = x;
        this.y = y;

        this.bullseyeSize = 10; // the granularity of the bullseye target.
    }

    getX()
    {
        return this.x;
    }

    getY()
    {
        return this.y;
    }

    getWidth()
    {
        return this.width;
    }

    getHeight()
    {
        return this.height;
    }

    pointIsInsideBullseyeRectangle(pointX, pointY)
    {
        /* The bullseye is set to have a width and height of bullseySize */
        /* The bulleseye is set from the centre of the bullseye image    */
```

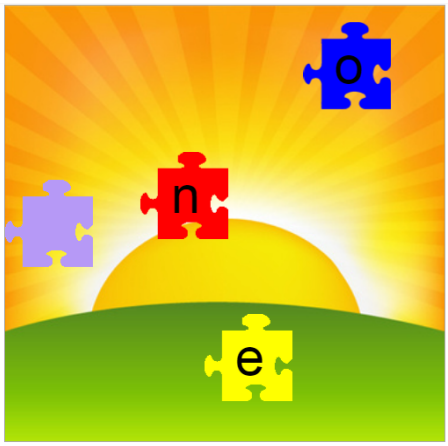
```
        let bullseyeX = this.x + ((this.width - this.bullseyeSize) / 2);
        let bullseyeY = this.y + ((this.height - this.bullseyeSize) / 2);

        if ((pointX > bullseyeX) && (pointY > bullseyeY))
        {
            if (pointX > bullseyeX)
            {
                if ((pointX - bullseyeX) > this.bullseyeSize)
                {
                    return false; // to the right of this gameObject
                }
            }

            if (pointY > bullseyeY)
            {
                if ((pointY - bullseyeY) > this.bullseyeSize)
                {
                    return false; // below this gameObject
                }
            }
        }
        else // above or to the left of this gameObject
        {
            return false;
        }
        return true; // inside this gameObject
    }
}
```

Case Study... Jigsaw Game

The aim of this game is to construct the correct word from the letter pieces.

This game shows:

- override the game's render() method
- override the game's collisionDetection() method
- transparancy test using an offscreen canvas
- use of Button class



Play game

Download .zip file

jigsaw_game.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.             */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */




/******************** Declare game specific global data and functions *****************/
/* images must be declared as global, so that they will load before the game starts  */
let background = new Image();
background.src = "images/sunshine.png";

let jigsawPieceImage = new Image();
jigsawPieceImage.src = "images/jigsaw_piece.png";

const BACKGROUND = 0;
const ANCHOR_PIECE = 1;
const BUTTON = 2;
const MESSAGE = 3;

const JIGSAW_Y = 200;  // the y-position of the jigsaw on the canvas
const JIGSAW_PIECE_SIZE = 100; // width and height of each jigsaw piece

/****************** END OF Declare game specific data and functions *****************/







/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
```

```
    /* This function will:                                          */
    /* 1. create the various game game gameObjects                 */
    /* 2. store the game gameObjects in an array                   */
    /* 3. create a new Game to display the game gameObjects        */
    /* 4. start the Game                                            */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */
    gameObjects[BACKGROUND] = new StaticImage(background, 0, 0, canvas.width, canvas.height);
    gameObjects[ANCHOR_PIECE] = new StaticImage(jigsawPieceImage, 0, JIGSAW_Y, JIGSAW_PIECE_SIZE, JIGSAW_PIECE_SIZE);
    gameObjects[BUTTON] = new Button(BUTTON_CENTRE, 400, TEXT_WIDTH, TEXT_HEIGHT, "Continue");
    gameObjects[MESSAGE] = new StaticText("Well Done!", STATIC_TEXT_CENTRE, 450, "Times Roman", 100, "black");

    let selectedPiece = 0; // default to any piece initially being selected

    /* END OF game specific code. */

    /* Always create a game that uses the gameObject array */
    let game = new JigsawCanvasGame(["one", "two"], jigsawPieceImage, JIGSAW_PIECE_SIZE, JIGSAW_Y);

    /* Always play the game */
    game.start();

    /* Hide the "Continue" button and win message until they are needed */
    gameObjects[BUTTON].stopAndHide();
    gameObjects[MESSAGE].stopAndHide();

    /* If they are needed, then include any game-specific mouse and keyboard listners */
    document.getElementById("gameCanvas").addEventListener("mousedown", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            if (gameObjects[BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[BUTTON].stopAndHide();
                game.createNewWordJigsaw();
            }
            for (let i = pieces.length - 1; i >= 0; i--)
            {
                if (pieces[i].pointIsInsideBoundingRectangle(mouseX, mouseY))
                {
                    pieces[i].setOffsetX(mouseX);
                    pieces[i].setOffsetY(mouseY);
                    selectedPiece = i;
                    break;
                }
            }
        }
    });

    document.getElementById("gameCanvas").addEventListener("mousemove", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            for (let i = 0; i < pieces.length; i++)
            {
                if (selectedPiece === i)
                {
                    if (pieces[i].pointIsInsideBoundingRectangle(mouseX, mouseY))
                    {
                        pieces[i].setX(mouseX);
                        pieces[i].setY(mouseY);
                        break;
                    }
                }
            }
        }
        else if (e.which === 0) // no button selected
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;
            gameObjects[BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY);
        }
    });
}
```

Code Explained

There are six different jigsaw colours. Each colour has its own image file.

```
let jigsawPiece = new Image();
redPiece.src = "images/jigsaw_piece.png";
```

The ANCHOR_PIECE is the jigsaw piece that that is placed at the start position of where the jigsaw word is placed.

```
const BACKGROUND = 0;
const ANCHOR_PIECE = 1;
const BUTTON = 2;
const MESSAGE = 3;
```

JIGSAW_Y is the vertical position where the jigsaw word will be placed.
JIGSAW_PIECE_SIZE is the width and height of each jigsaw piece.

```
const JIGSAW_Y = 200;  // the y-position of the jigsaw on the canvas
const JIGSAW_PIECE_SIZE = 100; // width and height of each jigsaw piece
```

JigsawCanvasGame takes in an array of words. The game will allow the user to complete one word jigsaw at a time. Once all of the jigsaw have been completed, the player wins the game.

```
    let game = new JigsawCanvasGame(["one", "two"], , jigsawPieceImage, JIGSAW_PIECE_SIZE, JIGSAW_Y);
```

After each word is completed, the user will be presented with a "Continue" BUTTON. When the user clicks the "Continue" BUTTON, the next jigsaw word will be created, as highlighted in red.
Whenever the user selects a jigsaw piece, then selectedPiece is set to hold the number of that jigsaw piece, as shown in blue.

```
/* If they are needed, then include any game-specific mouse and keyboard listners */
    document.getElementById("gameCanvas").addEventListener("mousedown", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            if (gameObjects[BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY))
            {
                gameObjects[BUTTON].stopAndHide();
                game.createNewWordJigsaw();
            }
            for (let i = pieces.length - 1; i >= 0; i--)
            {
                if (pieces[i].pointIsInsideBoundingRectangle(mouseX, mouseY))
                {
                    pieces[i].setOffsetX(mouseX);
                    pieces[i].setOffsetY(mouseY);
                    selectedPiece = i;
                    break;
                }
            }
        }
    });
```

As the mouse is moved, the selected jigsaw piece's x and y positions are updated, as shown in red.
If no mouse button is selected (e.which === 0) and the mouse hovers over the "Continue" BUTTON, then the button will change colour. The changing of its colour takes place inside the Button object. However, we need to call gameObjects[BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY) to fire the colour change event.

```
 document.getElementById("gameCanvas").addEventListener("mousemove", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            for (let i = 0; i < pieces.length; i++)
            {
                if (selectedPiece === i)
                {
                    if (pieces[i].pointIsInsideBoundingRectangle(mouseX, mouseY))
                    {
```

```
                        pieces[i].setX(mouseX);
                        pieces[i].setY(mouseY);
                        break;
                    }
                }
            }
        }
        else if (e.which === 0) // no button selected
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;
            gameObjects[BUTTON].pointIsInsideBoundingRectangle(mouseX, mouseY);
        }
    });
```

JigsawCanvasGame.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.       */
/* A CanvasGame that implements collision detection.                       */


let pieces = [];       /* These two variables are used in both this class and in the JigsawPiece class */
let currentPiece = 0;  /* Therefore, they need to be global to both classes.                           */

class JigsawCanvasGame extends CanvasGame
{
    constructor(wordList, jigsawPieceImage, pieceSize, wordY)
    {
        super();

        this.wordList = wordList;
        this.jigsawPieceImage = jigsawPieceImage;
        this.pieceSize = pieceSize;
        this.wordY = wordY;

        this.currentWord = 0;
        this.createNewWordJigsaw();
    }

    createNewWordJigsaw()
    {
        currentPiece = 0;

        let colours = [[255, 0, 0], [0, 255, 0], [0, 0, 255], [255, 0, 255], [255,255,0], [255, 100, 0]];
        let currentColour = -1;  // the fixed jigsaw start piece is not contained in the colours array
        let newColour = null; // newColour is randomly assigned below

        for (let letter = 0; letter < this.wordList[this.currentWord].length; letter++)
        {
            do
            {
                newColour = Math.floor(Math.random() * colours.length);
            } while (newColour === currentColour); // make sure that pieces beside each other are different colours
            currentColour = newColour;
            pieces[letter] = new JigsawPiece(this.jigsawPieceImage, colours[newColour], this.wordList[this.currentWord][
        }
        this.currentWord++;
    }

    render()
    {
        super.render();
        for (let i = 0; i < pieces.length; i++)
        {
            pieces[i].render();
        }
    }

    collisionDetection()
    {
        if ((pieces.length === 0) || (this.wordList.length === 0))
        {
            return;
        }

        if (currentPiece === pieces.length)
        {
            if (this.currentWord < this.wordList.length)
            {
                gameObjects[BUTTON].start(); /* render "Continue" message */
            }
```

```
            else
            {
                for (let i = 0; i < pieces.length; i++) /* stop all gameObjects from animating */
                {
                    pieces[i].stop();
                }

                gameObjects[MESSAGE].start(); /* render "Well Done!" message */
            }
        }
    }
}
```

Code Explained

The pieces[] array holds the various letter jigsaw pieces that make up the current word.
currentPiece is the piece that is currently able to be anchored to build up the jigsaw word. The player can move any piece, but only the currentPiece can be clicked into place in the jigsaw word. Once a piece has been clicked into place, it can no longer be moved.
selectedPiece is the currently selected piece.

```
let pieces = [];
let currentPiece = 0;
```

The wordList holds the array of words in this game.
This word will be placed vertically on the canvas at postion 'wordY'.
The game will build one word at a time. The word being built at any time is 'this.currentWord'.

```
    constructor(wordList, jigsawPieceImage, pieceSize, wordY)
    {
        super();

        this.wordList = wordList;
        this.jigsawPieceImage = jigsawPieceImage;
        this.pieceSize = pieceSize;
        this.wordY = wordY;

        this.currentWord = 0;
        this.createNewWordJigsaw();
    }
```

The createNewWordJigsaw() method creates the pieces that are needed for the currentWord in the jigsaw's list of words. This method uses a for loop to fill the pieces[] array with the individual jigsaw pieces that contain each of the letters in a jigsaw word.
A jigsaw piece can be any of the colours in the colours[] array.
The do...while loop is used to assign a colour to each jigsaw piece. The do...while loop is used to ensure that no two piece beside each other have the same colour.

```
    createNewWordJigsaw()
    {
        currentPiece = 0;

        let colours = [[255, 0, 0], [0, 255, 0], [0, 0, 255], [255, 0, 255], [255,255,0], [255, 100, 0]];
        let currentColour = -1;  // the fixed jigsaw start piece is not contained in the colours array
        let newColour = null; // newColour is randomly assigned below

        for (let letter = 0; letter < this.wordList[this.currentWord].length; letter++)
        {
            do
            {
                newColour = Math.floor(Math.random() * colours.length);
            } while (newColour === currentColour); // make sure that pieces beside each other are different colours
            currentColour = newColour;
            pieces[letter] = new JigsawPiece(this.jigsawPieceImage, colours[newColour], this.wordList[this.currentWord][
        }
        this.currentWord++;
    }
```

The jigsaw pieces are not contained in the gameObjects[] array. Instead, they are contained in an array called pieces[]. Therefore, we must override the render() method, so that the jigsaw pieces (in the pieces[] array) can be drawn.

```
    render()
    {
        super.render();
        for (let i = 0; i < pieces.length; i++)
        {
            pieces[i].render();
        }
    }
```

The collisionDetection() method is used to detect the end of each round and the end of the game.

```
    collisionDetection()
    {
        if ((pieces.length === 0) || (this.wordList.length === 0))
        {
            return;
        }

        if (currentPiece === pieces.length)
        {
            if (this.currentWord < this.wordList.length)
            {
                gameObjects[BUTTON].start(); /* render "Continue" message */
            }
            else
            {
                for (let i = 0; i < pieces.length; i++) /* stop all gameObjects from animating */
                {
                    pieces[i].stop();
                }

                gameObjects[MESSAGE].start(); /* render "Well Done!" message */
            }
        }
    }
```

JigsawPiece.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class JigsawPiece extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(jigsawPieceImage, colour, letter, id, size, granulatity, startX, startY, finalX, finalY)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.letter = letter;
        this.id = id;
        this.width = size;
        this.height = size;
        this.x = startX;
        this.y = startY;
        this.offsetX = 0;
        this.offsetY = 0;

        this.finalX = finalX; // this is the position where the jigsaw piece needs to end up at
        this.finalY = finalY;
        this.granulatity = granulatity; // the +/- resolution of the accuracy of where the piece needs to end up
        this.isLocked = false; // set to true when the piece is at its final place


        this.jigsawCanvas = document.createElement('canvas');
        this.jigsawCanvasCtx = this.jigsawCanvas.getContext("2d");
        this.jigsawCanvas.width = jigsawPieceImage.width;
        this.jigsawCanvas.height = jigsawPieceImage.height;
        this.jigsawCanvasCtx.drawImage(jigsawPieceImage, 0, 0, jigsawPieceImage.width, jigsawPieceImage.height); /* As a

        let imageData = this.jigsawCanvasCtx.getImageData(0, 0, this.jigsawCanvas.width, this.jigsawCanvas.height);
        let data = imageData.data;
        if (data[i + 3] !== 0)
        {
            // Manipulate the pixel data
            for (var i = 0; i < data.length; i += 4)
            {
                data[i + 0] = colour[0];
                data[i + 1] = colour[1];
                data[i + 2] = colour[2];
            }
        }
        this.jigsawCanvasCtx.putImageData(imageData, 0, 0);

        this.jigsawCanvasCtx.strokeStyle = "black";
        this.jigsawCanvasCtx.font = this.height * 0.6 + "px Arial";  // scale the font to match the size of the jigsaw p
        this.jigsawCanvasCtx.fillText(this.letter, this.height * 0.35, this.height * 0.70);   // position the letter in
    }
```

```
render()
{
    ctx.drawImage(this.jigsawCanvas, this.x, this.y, this.width, this.height);
}

isPieceAtFinalPosition()
{
    if (this.id !== currentPiece)
    {
        return false;
    }
    if (this.isLocked)
    {
        return false;
    }
    if ((this.x > this.finalX - this.granulatity) &&
            (this.x < this.finalX + this.granulatity) &&
            (this.y > this.finalY - this.granulatity) &&
            (this.y < this.finalY + this.granulatity))
    {

        this.x = this.finalX;
        this.y = this.finalY;
        this.isLocked = true;

        currentPiece++; // allow the next jigsaw piece to be locked

        return true;
    }
    return false;
}

setX(newMouseX)
{
    if (this.isLocked)
    {
        return;
    }
    if (!this.isPieceAtFinalPosition())
    {
        this.x = newMouseX - this.offsetX;
    }
}

setY(newMouseY)
{
    if (this.isLocked)
    {
        return;
    }
    this.y = newMouseY - this.offsetY;
}

setOffsetX(newMouseX)
{
    if (this.isLocked)
    {
        return;
    }
    this.offsetX = newMouseX - this.x;
}

setOffsetY(newMouseY)
{
    if (this.isLocked)
    {
        return;
    }
    this.offsetY = newMouseY - this.y;
}

pointIsInsideBoundingRectangle(pointX, pointY)
{
    if (this.isLocked)
    {
        return;
    }
    if ((pointX > this.x) && (pointY > this.y))
    {
        if (pointX > this.x)
        {
            if ((pointX - this.x) > this.width)
            {
                return false; // to the right of this gameObject
```

```
                }
            }

            if (pointY > this.y)
            {
                if ((pointY - this.y) > this.height)
                {
                    return false; // below this gameObject
                }
            }
        }
        else // above or to the left of this gameObject
        {
            return false;
        }

        // passed basic bounding test
        // now test for the transparent part of the jigsaz piece
        let imageData = this.jigsawCanvasCtx.getImageData(pointX - this.x, pointY - this.y, 1, 1);
        let data = imageData.data;

        // Check the pixel data for transparancy
        if (data[3] === 0)
        {
            return false;
        }

        // mouse is on top of jigsaw piece
        return true;
    }
}
```

Code Explained

An offscreen canvas is used to hold the jigsaw piece. The jigsaw piece colour is determined by the colour that is passed into the constructor. The offscreen buffer jigsaw piece is filled with 'colour'. The jigsaw piece will also used for tansparancy testing when the user tries to click the mouse on the jigsaw piece.

Each jigsaw piece has various data, such as width and startX associated with it. The id of the piece (this.id) is its position within the word array. The id is used to give the order that the letters need to be put together in order to create the jigsaw word.

The granularity of the jigsaw piece (this.granularity) allows for the piece to be detected as hitting the target pixel when it is close to the target. A higher granularity will mean that the piece is more easily placed at its target. The granularity is the number of pixels around the final x,y position of the jigsaw piece that the user needs to place the jigsaw piece.

Once a jigsaw piece has been locked in place, the this.isLocked flag is set. This will be used to stop the jigsaw piece from being moved again.

```
    constructor(jigsawPieceImage, colour, letter, id, size, granulatity, startX, startY, finalX, finalY)
    {
        super(null); /* as this class extends from GameObject, you must always call super() */

        /* These variables depend on the object */
        this.letter = letter;
        this.id = id;
        this.width = size;
        this.height = size;
        this.x = startX;
        this.y = startY;
        this.offsetX = 0;
        this.offsetY = 0;

        this.finalX = finalX; // this is the position where the jigsaw piece needs to end up at
        this.finalY = finalY;
        this.granulatity = granulatity; // the +/- resolution of the accuracy of where the piece needs to end up
        this.isLocked = false; // set to true when the piece is at its final place


        this.jigsawCanvas = document.createElement('canvas');
        this.jigsawCanvasCtx = this.jigsawCanvas.getContext("2d");
        this.jigsawCanvas.width = jigsawPieceImage.width;
        this.jigsawCanvas.height = jigsawPieceImage.height;
        this.jigsawCanvasCtx.drawImage(jigsawPieceImage, 0, 0, jigsawPieceImage.width, jigsawPieceImage.height); /* As a

        let imageData = this.jigsawCanvasCtx.getImageData(0, 0, this.jigsawCanvas.width, this.jigsawCanvas.height);
        let data = imageData.data;
        if (data[i + 3] !== 0)
        {
            // Manipulate the pixel data
            for (var i = 0; i < data.length; i += 4)
            {
                data[i + 0] = colour[0];
                data[i + 1] = colour[1];
                data[i + 2] = colour[2];
```

```
            }
        }
        this.jigsawCanvasCtx.putImageData(imageData, 0, 0);

        this.jigsawCanvasCtx.strokeStyle = "black";
        this.jigsawCanvasCtx.font = this.height * 0.6 + "px Arial";  // scale the font to match the size of the jigsaw p
        this.jigsawCanvasCtx.fillText(this.letter, this.height * 0.35, this.height * 0.70);   // position the letter in
    }
```

The isPieceAtFinalPosition() method test if the current x and y are within the granulatity of the final x and y positions, as shown in red.

If the jigsaw piece is within the granularity, then the jigsaw piece is set to the final x and y position, the isLocked flag is set and the currentPiece is incremented to the next jigsaw piece.

```
    isPieceAtFinalPosition()
    {
        if (this.id !== currentPiece)
        {
            return false;
        }
        if (this.isLocked)
        {
            return false;
        }
        if ((this.x > this.finalX - this.granulatity) &&
                (this.x < this.finalX + this.granulatity) &&
                (this.y > this.finalY - this.granulatity) &&
                (this.y < this.finalY + this.granulatity))
        {

            this.x = this.finalX;
            this.y = this.finalY;
            this.isLocked = true;

            currentPiece++; // allow the next jigsaw piece to be locked

            return true;
        }
        return false;
    }
```

The pointIsInsideBoundingRectangle() method does the same rectangular bounding test that we have seen in previous examples. If this test is passed, then a second test is done to check if the mouse is on a transparent pixel within the jigsaw piece. The transparency test code is highlighted in red.

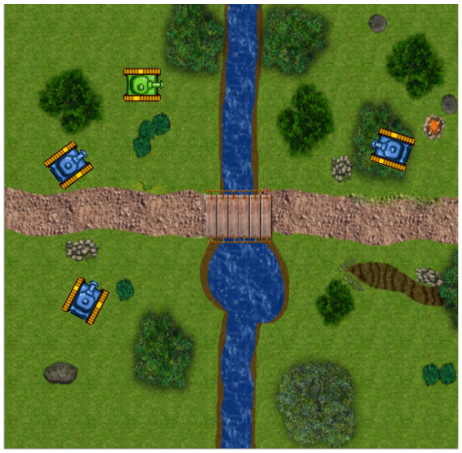
```
    pointIsInsideBoundingRectangle(pointX, pointY)
    {
        if (this.isLocked)
        {
            return;
        }
        if ((pointX > this.x) && (pointY > this.y))
        {
            if (pointX > this.x)
            {
                if ((pointX - this.x) > this.width)
                {
                    return false; // to the right of this gameObject
                }
            }

            if (pointY > this.y)
            {
                if ((pointY - this.y) > this.height)
                {
                    return false; // below this gameObject
                }
            }
        }
        else // above or to the left of this gameObject
        {
            return false;
        }

        // passed basic bounding test
        // now test for the transparent part of the jigsaz piece
        let imageData = this.jigsawCanvasCtx.getImageData(pointX - this.x, pointY - this.y, 1, 1);
        let data = imageData.data;

        // Check the pixel data for transparancy
        if (data[3] === 0)
        {
            return false;
```

```
        }

        // mouse is on top of jigsaw piece
        return true;
    }
```

Case Study... Tank Game

A simple game consists of:

- Getting user input and updating the game state. Input is got via the keyboard.
- Rotation of objects on the canvas
- Collision detection between multiple moving objects
- Collision detection of rotated bounding rectangles



The aim is to destroy the blue enemy tanks.
Use the arrow keys to steer the green tank.
Use the space bar to fire a shell.

Play game

Download .zip file

tank_game.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.             */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */



/******************** Declare game specific global data and functions *****************/
/* images must be declared as global, so that they will load before the game starts  */
let fieldImage = new Image();
fieldImage.src = "images/field.png";
let riverImage = new Image();
riverImage.src = "images/tankFieldRiver.png";
let tankImage = new Image();
tankImage.src = "images/tank.png";
let explosionImage = new Image();
explosionImage.src = "images/explosion.png";

let tankMovingSound = new Audio();
tankMovingSound.src = 'audio/tankMoving.mp3';
let fireShellSound = new Audio();
fireShellSound.src = 'audio/tankFire.mp3';
let shellExplosionSound = new Audio();
shellExplosionSound.src = 'audio/shellExplosionSound.mp3';

const BACKGROUND = 0;
const TANK = 1;
const FIRE_SHELL = 2;  // animation showing initial firing of shell
const SHELL = 3; // a shell that is fired from the tank
const EXPLOSION = 4; // the explosion that results from the firing of the shell
const WIN_MESSAGE = 5;

let enemyTanks = [];
const numberOfEnemyTanks = 3;
/****************** END OF Declare game specific data and functions ****************/
```

```
/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */
    gameObjects[BACKGROUND] = new StaticImage(fieldImage, 0, 0, canvas.width, canvas.height);
    gameObjects[TANK] = new PlayerTank(tankImage, riverImage, tankMovingSound, 100, 100, 90);
    gameObjects[TANK].startMoving();

    // create the enemy tanks
    for (let i = 0; i < numberOfEnemyTanks; i++)
    {
        do
        {
            // make sure that the enemy tanks do not randomly spawn in the river
            enemyTanks[i] = new EnemyTank(tankImage, riverImage, tankMovingSound, Math.random() * (canvas.width - 25) +
        }while (enemyTanks[i].collidedWithRiver())
        enemyTanks[i].start();
        enemyTanks[i].startMoving();
    }

    /* END OF game specific code. */

    /* Always create a game that uses the gameObject array */
    let game = new TankCanvasGame();

    /* Always play the game */
    game.start();


    /* If they are needed, then include any game-specific mouse and keyboard listners */
    document.addEventListener("keydown", function (e)
    {
        var ANGLE_STEP_SIZE = 10;

        if (e.keyCode === 37)  // left
        {
            gameObjects[TANK].setDirection(gameObjects[TANK].getDirection() - ANGLE_STEP_SIZE);
        }
        else if (e.keyCode === 39) // right
        {
            gameObjects[TANK].setDirection(gameObjects[TANK].getDirection() + ANGLE_STEP_SIZE);
        }
        else if (e.keyCode === 32) // space
        {
            gameObjects[FIRE_SHELL] = new FireShellAnimation(tankImage, fireShellSound, gameObjects[TANK].getX(), gameOb
            gameObjects[FIRE_SHELL].start();

            gameObjects[SHELL] = new Shell(explosionImage, shellExplosionSound, gameObjects[TANK].getX(), gameObjects[TA
            gameObjects[SHELL].start();
        }
    });
}
```

Code Explained

The game consists of various gameObjects. In particular, when a shell is fired, three different things happen. Firstly, a small explosion occurs at the top of the tank's gun barrel. Secondly, the shell moves along its path. Thirdly, as it moves, it performs collision with the enemy tanks. If it collides with an enamy tank, then a large explosion occurs at the point of collision. If the shell does not collide with any enemy tanks within its range, then a large explosion occurs when the shell reaches its range.
The small explosion is contained in FIRE_SHELL, the shell movement is contained in SHELL and the large explosion is contained in EXPLOSION.

```
const BACKGROUND = 0;
const TANK = 1;
const FIRE_SHELL = 2;  // animation showing initial firing of shell
const SHELL = 3; // a shell that is fired from the tank
```

```
const EXPLOSION = 4; // the explosion that results from the firing of the shell
const WIN_MESSAGE = 5;
```

The enemyTanks are held in their own array, called enemyTanks[]. In this game, there are 3 enemy tanks.
As the enemy tanks are not being held in gameObjects[], the game code will be responsible for their rendering.

```
let enemyTanks = [];
const numberOfEnemyTanks = 3;
```

When they are created, the enemy tanks are placed in random positions on the canvas. When creating the enemy tanks, we use a do...while loop to ensure that the tanks do get spawned inside the river.

```
    // create the enemy tanks
    for (let i = 0; i < numberOfEnemyTanks; i++)
    {
        do
        {
            // make sure that the enemy tanks do not randomly spawn in the river
            enemyTanks[i] = new EnemyTank(tankImage, riverImage, tankMovingSound, Math.random() * (canvas.width - 25) +
        }while (enemyTanks[i].collidedWithRiver())
        enemyTanks[i].start();
        enemyTanks[i].startMoving();
    }
```

The left and right arrow keys are used to turn the player tank
The space bar is used to fire a shell. When a shell is fired, a small explosion occurs at the top of the tank's gun barrel, as shown in red below.
The shell is fired in the direction that the tank is facing, as shown in blue below.

```
 document.addEventListener("keydown", function (e)
    {
        var ANGLE_STEP_SIZE = 10;

        if (e.keyCode === 37)  // left
        {
            gameObjects[TANK].setDirection(gameObjects[TANK].getDirection() - ANGLE_STEP_SIZE);
        }
        else if (e.keyCode === 39) // right
        {
            gameObjects[TANK].setDirection(gameObjects[TANK].getDirection() + ANGLE_STEP_SIZE);
        }
        else if (e.keyCode === 32) // space
        {
            gameObjects[FIRE_SHELL] = new FireShellAnimation(tankImage, fireShellSound, gameObjects[TANK].getX(), gameOb
            gameObjects[FIRE_SHELL].start();

            gameObjects[SHELL] = new Shell(explosionImage, shellExplosionSound, gameObjects[TANK].getX(), gameObjects[TA
            gameObjects[SHELL].start();
        }
    });
```

TankCanvasGame.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.       */
/* A CanvasGame that implements collision detection.                       */


class TankCanvasGame extends CanvasGame
{
    constructor()
    {
        super();

        this.numberOfEnemytanksDestroyed = 0;
    }

    collisionDetection()
    {
        /* Collision detection for the player tank bumping into an enemy tank */
        for (let i = 0; i < enemyTanks.length; i++)
        {
            if ((enemyTanks[i].pointIsInsideTank(gameObjects[TANK].getFrontLeftCornerX(), gameObjects[TANK].getFrontLeft
                    (enemyTanks[i].pointIsInsideTank(gameObjects[TANK].getFrontRightCornerX(), gameObjects[TANK].getFron
            {
                enemyTanks[i].reverse(5);
                enemyTanks[i].setDirection(enemyTanks[i].getDirection() + 10); // turn away from the river, so that the
            }
        }
```

```
        /* Collision detection for the enemy tanks bumping into each other */
        for (let i = 0; i < enemyTanks.length; i++)
        {
            /* check if enemy tank bumps into player tank */
            if ((gameObjects[TANK].pointIsInsideTank(enemyTanks[i].getFrontLeftCornerX(), enemyTanks[i].getFrontLeftCorn
                 (gameObjects[TANK].pointIsInsideTank(enemyTanks[i].getFrontRightCornerX(), enemyTanks[i].getFrontRig
            {
                enemyTanks[i].reverse(5);
                enemyTanks[i].setDirection(enemyTanks[i].getDirection() + 10); // turn away from the river, so that the
            }

            /* check if enemy tank bumps into another enemy tank */
            for (let j = 0; j < enemyTanks.length; j++)
            {
                if (i !== j)
                {
                    if ((enemyTanks[i].pointIsInsideTank(enemyTanks[j].getFrontLeftCornerX(), enemyTanks[j].getFrontLeft
                           (enemyTanks[i].pointIsInsideTank(enemyTanks[j].getFrontRightCornerX(), enemyTanks[j].getFron
                    {
                        enemyTanks[i].reverse(3);
                        enemyTanks[i].setDirection(enemyTanks[i].getDirection() + 10); // turn away from the river, so t
                    }
                }
            }
        }


        /* Collision detection of the player tank with the river */
        if (gameObjects[TANK].collidedWithRiver())
        {
            gameObjects[TANK].reverse();
        }

        /* Collision detection for enemy tanks with the river */
        for (let i = 0; i < enemyTanks.length; i++)
        {
            if (enemyTanks[i].collidedWithRiver())
            {
                enemyTanks[i].reverse();
                enemyTanks[i].setDirection(enemyTanks[i].getDirection() + 10); // turn away from the river, so that the
            }
        }

        /* Collision detection for a shell that is firing */
        if (gameObjects[SHELL] === undefined)
        {
            return;
        }
        for (let i = 0; i < enemyTanks.length; i++)
        {
            if (gameObjects[SHELL].isFiring())
            {
                if ((enemyTanks[i].isDisplayed()) && (enemyTanks[i].pointIsInsideTank(gameObjects[SHELL].getX(), gameObj
                {
                    enemyTanks[i].stopAndHide();
                    gameObjects[EXPLOSION] = new Explosion(explosionImage, shellExplosionSound, enemyTanks[i].getX(), en
                    gameObjects[EXPLOSION].start();
                    gameObjects[SHELL].stopAndHide();

                    this.numberOfEnemytanksDestroyed++;
                    if (this.numberOfEnemytanksDestroyed === numberOfEnemyTanks)
                    {
                        /* Player has won
                        /* Have a two second delay to show the last enemy tank blowing up beofore displaying the 'Game O
                        setInterval(function ()
                        {
                            for (let j = 0; j < gameObjects.length; j++) /* stop all gameObjects from animating */
                            {
                                gameObjects[j].stopAndHide();
                            }
                            gameObjects[TANK].stopMoving(); // turn off tank moving sound
                            gameObjects[BACKGROUND].start();
                            gameObjects[WIN_MESSAGE] = new StaticText("Game Over!", 5, 270, "Times Roman", 100, "red");
                            gameObjects[WIN_MESSAGE].start(); /* render win message */
                        }, 2000);
                    }
                }
            }
        }
    }

    render()
    {
        super.render();
        for (let i = 0; i < enemyTanks.length; i++)
```

```
        {
            if (enemyTanks[i].isDisplayed())
            {
                enemyTanks[i].render();
            }
        }
    }
}
```

Code Explained

Various collision detection is done in this game.
If the player's tank bumps into an enemy tank, then the enemy tank will reverse by 5 units and turn 10 degrees clockwise. The enemy tank will then continue forward on its new path.
Note, that collision detection is done by checking if either the front-left or front-right corners of the player tank are inside the bounding rectangle of the enemy tank.

```
        /* Collision detection for the player tank bumping into an enemy tank */
        for (let i = 0; i < enemyTanks.length; i++)
        {
            if ((enemyTanks[i].pointIsInsideTank(gameObjects[TANK].getFrontLeftCornerX(), gameObjects[TANK].getFrontLeft
                    (enemyTanks[i].pointIsInsideTank(gameObjects[TANK].getFrontRightCornerX(), gameObjects[TANK].getFron
            {
                enemyTanks[i].reverse(5);
                enemyTanks[i].setDirection(enemyTanks[i].getDirection() + 10); // turn away from the river, so that the
            }
        }
```

Each enemy tank needs to be tested for collision against the player tank all other enemy tanks. This is done in the same way as the player tank was tested for collision above, were the front-left and front-right corners of the enemy tank are tested againsts the bounding rectangles of the player tank and the other enemy tanks.

```
        /* Collision detection for the enemy tanks bumping into each other */
        for (let i = 0; i < enemyTanks.length; i++)
        {
            /* check if enemy tank bumps into player tank */
            if ((gameObjects[TANK].pointIsInsideTank(enemyTanks[i].getFrontLeftCornerX(), enemyTanks[i].getFrontLeftCorn
                    (gameObjects[TANK].pointIsInsideTank(enemyTanks[i].getFrontRightCornerX(), enemyTanks[i].getFrontRig
            {
                enemyTanks[i].reverse(5);
                enemyTanks[i].setDirection(enemyTanks[i].getDirection() + 10); // turn away from the river, so that the
            }

            /* check if enemy tank bumps into another enemy tank */
            for (let j = 0; j < enemyTanks.length; j++)
            {
                if (i !== j)
                {
                    if ((enemyTanks[i].pointIsInsideTank(enemyTanks[j].getFrontLeftCornerX(), enemyTanks[j].getFrontLeft
                            (enemyTanks[i].pointIsInsideTank(enemyTanks[j].getFrontRightCornerX(), enemyTanks[j].getFron
                    {
                        enemyTanks[i].reverse(3);
                        enemyTanks[i].setDirection(enemyTanks[i].getDirection() + 10); // turn away from the river, so t
                    }
                }
            }
        }
```

If the player's tank collides with the river, then it is reversed back out of the river.

```
/* Collision detection of the player tank with the river */
        if (gameObjects[TANK].collidedWithRiver())
        {
            gameObjects[TANK].reverse();
        }
```

If an enemy tank collides with the river, it is reversed back out of the river and it changes direction by 10 degrees in a clockwise direction. In this way the tank will eventually move away from the river.

```
        /* Collision detection for enemy tanks with the river */
        for (let i = 0; i < enemyTanks.length; i++)
        {
            if (enemyTanks[i].collidedWithRiver())
            {
                enemyTanks[i].reverse();
                enemyTanks[i].setDirection(enemyTanks[i].getDirection() + 10); // turn away from the river, so that the
            }
```

```
        }
```

Collision detection of a shell can only happen if there is a shell. The test 'gameObjects[SHELL] === undefined' is needed to stop a code crash when there is no shell firing. The test 'gameObjects[SHELL].isFiring()' is used to detect that a shell is firing.
A shell can only strike a tank that is being displayed. The test '(enemyTanks[i].isDisplayed())' is used to detect this.
A shell will strike a tank if it is insdie the tank's bounding rectangle. The test '(enemyTanks[i].pointIsInsideTank(gameObjects[SHELL].getX(), gameObjects[SHELL].getY()))' is used to detect this.
If a shell hits an enemy tank, then:

- that tank is no longer displayed on the canvas.
- an explosion occurs at the point where the shell hit the tank
- if all enemy tanks have been destroyed, then the game ends

When the game ends, all of the gameObjects except the BACKGROUND are hidden. The two-second interval allows the gameObject effect of the last enemy tank blowing up to be displayed prior to the game ending.

```
        /* Collision detection for a shell that is firing */
        if (gameObjects[SHELL] === undefined)
        {
            return;
        }
        for (let i = 0; i < enemyTanks.length; i++)
        {
            if (gameObjects[SHELL].isFiring())
            {
                if ((enemyTanks[i].isDisplayed()) && (enemyTanks[i].pointIsInsideTank(gameObjects[SHELL].getX(), gameObj
                {
                    enemyTanks[i].stopAndHide();
                    gameObjects[EXPLOSION] = new Explosion(explosionImage, shellExplosionSound, enemyTanks[i].getX(), en
                    gameObjects[EXPLOSION].start();
                    gameObjects[SHELL].stopAndHide();

                    this.numberOfEnemytanksDestroyed++;
                    if (this.numberOfEnemytanksDestroyed === numberOfEnemyTanks)
                    {
                        /* Player has won
                        /* Have a two second delay to show the last enemy tank blowing up beofore displaying the 'Game O
                        setInterval(function ()
                        {
                            for (let j = 0; j < gameObjects.length; j++) /* stop all gameObjects from animating */
                            {
                                gameObjects[j].stopAndHide();
                            }
                            gameObjects[TANK].stopMoving(); // turn off tank moving sound
                            gameObjects[BACKGROUND].start();
                            gameObjects[WIN_MESSAGE] = new StaticText("Game Over!", 5, 270, "Times Roman", 100, "red");
                            gameObjects[WIN_MESSAGE].start(); /* render win message */
                        }, 2000);
                    }
                }
            }
        }
```

Tank.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class Tank extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(tankImage, riverImage, tankMovingSound, speed, centreX, centreY, direction, startRow, startColumn)
    {
        super(200 - speed); /* as this class extends from GameObject, you must always call super() */

        this.tankImage = tankImage;
        this.tankMovingSound = tankMovingSound;

        /* this.offscreenObstaclesCtx will be used for collision detection with the river */
        this.offscreenObstacles = document.createElement('canvas');
```

```
        this.offscreenObstaclesCtx = this.offscreenObstacles.getContext('2d');
        this.offscreenObstacles.width = canvas.width;
        this.offscreenObstacles.height = canvas.height;
        this.offscreenObstaclesCtx.drawImage(riverImage, 0, 0, canvas.width, canvas.height);

        this.centreX = centreX;
        this.centreY = centreY;
        this.size = 50;  // the width and height of the tank
        this.halfSize = this.size / 2;

        this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE = 8; // the number of columns in the sprite image
        this.NUMBER_OF_ROWS_IN_SPRITE_IMAGE = 4; // the number of rows in the sprite image
        this.NUMBER_OF_SPRITES = 8; // the number of sprites in the sprite image

        this.START_ROW = startRow;
        this.START_COLUMN = startColumn;

        this.currentSprite = 0; // the current sprite to be displayed from the sprite image
        this.row = this.START_ROW; // current row in sprite image
        this.column = this.START_COLUMN; // current column in sprite image
        this.SPRITE_INCREMENT = -1; // sub-images in the sprite image are ordered from bottom to top, right to left

        this.SPRITE_WIDTH = (this.tankImage.width / this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE);
        this.SPRITE_HEIGHT = (this.tankImage.height / this.NUMBER_OF_ROWS_IN_SPRITE_IMAGE);

        this.STEP_SIZE = 2; // the number of pixels to move forward
        this.setDirection(direction);

        this.isMoving = false; // the tank is initially stopped
    }

    updateState()
    {
        if (!this.isMoving)
        {
            return;
        }
        this.currentSprite++;
        this.column += this.SPRITE_INCREMENT;
        if (this.currentSprite >= this.NUMBER_OF_SPRITES)
        {
            this.currentSprite = 0;
            this.row = this.START_ROW;
            this.column = this.START_COLUMN;
        }

        if (this.column < 0)
        {
            this.column = this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE - 1;
            this.row--;
        }

        this.centreX += this.stepSizeX;
        this.centreY += this.stepSizeY;

        /* if the tank goes off the canvas, then make it reappear at the opposite side of the canvas */
        if ((this.centreX - this.halfSize) > canvas.width)
        {
            this.centreX = -this.halfSize;
        }
        else if ((this.centreY - this.halfSize) > canvas.height)
        {
            this.centreY = -this.halfSize;
        }
        else if ((this.centreX + this.halfSize) < 0)
        {
            this.centreX = canvas.width + this.halfSize;
        }
        else if ((this.centreY + this.halfSize) < 0)
        {
            this.centreY = canvas.height + this.halfSize;
        }
    }

    render()
    {
        ctx.save();
        ctx.translate(this.centreX, this.centreY);
        ctx.rotate(Math.radians(this.direction));
        ctx.translate(-this.centreX, -this.centreY);

        ctx.drawImage(this.tankImage, this.column * this.SPRITE_WIDTH, this.row * this.SPRITE_HEIGHT, this.SPRITE_WIDTH,
        ctx.restore();
    }
```

```
pointIsInsideTank(x, y)
{
    /* transform the shell into the enemy tank's coordinate system */
    let transformedX = x - this.centreX;
    let transformedY = y - this.centreY;
    x = transformedX * Math.cos(Math.radians((this.direction))) - transformedY * Math.sin(Math.radians(this.directio
    y = transformedX * Math.sin(Math.radians((this.direction))) + transformedY * Math.cos(Math.radians(this.directio
    x += this.centreX;
    y += this.centreY;

    let imageTopLeftX = this.centreX - parseInt(this.size / 2);
    let imageTopLeftY = this.centreY - parseInt(this.size / 2);
    if ((x > imageTopLeftX) && (y > imageTopLeftY))
    {
        if (x > imageTopLeftX)
        {
            if ((x - imageTopLeftX) > this.size)
            {
                return false; // to the right of the tank image
            }
        }

        if (y > imageTopLeftY)
        {
            if ((y - imageTopLeftY) > this.size)
            {
                return false; // below the tank image
            }
        }
    }
    else // above or to the left of the tank image
    {
        return false;
    }
    return true; // inside tank image
}

collidedWithRiver()
{
    /* test the front-left corner and the front-right corner of the tank for collision with the river */
    /* we only need to test the front of the tank, as the tank can only move forward
    if ((this.pointCollisionWithRiver(this.getFrontLeftCornerX(), this.getFrontLeftCornerY())) ||
            (this.pointCollisionWithRiver(this.getFrontRightCornerX(), this.getFrontRightCornerY())))
    {
        return true;
    }
    return false;
}

pointCollisionWithRiver(x, y)
{

    let transformedX = x - this.centreX;
    let transformedY = y - this.centreY;
    x = transformedX * Math.cos(Math.radians((this.direction))) - transformedY * Math.sin(Math.radians(this.directio
    y = transformedX * Math.sin(Math.radians((this.direction))) + transformedY * Math.cos(Math.radians(this.directio
    x += this.centreX;
    y += this.centreY;

    let imageData = this.offscreenObstaclesCtx.getImageData(x, y, 1, 1);
    let data = imageData.data;
    if (data[3] !== 0)
    {
        return true;
    }
    return false;
}

positionRandomly()
{
    this.centreX = Math.random() * (canvas.width - (this.size * 2)) + this.size;
    this.centreY = Math.random() * (canvas.height - (this.size * 2)) + this.size;
    this.setDirection(Math.random() * 360);

    // reset the tank sprite
    this.currentSprite = 0;
    this.row = this.START_ROW;
    this.column = this.START_COLUMN;
}

startMoving()
{
    this.isMoving = true;
    this.tankMovingSound.currentTime = 0;
    this.tankMovingSound.play();
```

```
    /* ensure that the sound loops continuously */
    this.tankMovingSound.addEventListener('ended', function ()
    {
        this.currentTime = 0;
        this.play();
    });
}

stopMoving()
{
    this.isMoving = false;
    this.tankMovingSound.pause();
}

tankIsMoving()
{
    return this.isMoving;
}

reverse(numberOfReverseSteps = 1)
{
    // move in reverse direction
    for (let i = 0; i < numberOfReverseSteps; i++)
    {
        this.setX(this.getX() - this.getStepSizeX());
        this.setY(this.getY() - this.getStepSizeY());
    }
}

getDirection()
{
    return this.direction;
}

setDirection(newDirection)
{
    this.direction = newDirection;

    this.stepSizeX = this.STEP_SIZE * Math.sin(Math.radians(this.direction));
    this.stepSizeY = -this.STEP_SIZE * Math.cos(Math.radians(this.direction));
}

getX()
{
    return this.centreX;
}

getY()
{
    return this.centreY;
}

setX(x)
{
    this.centreX = x;
}

setY(y)
{
    this.centreY = y;
}

getStepSizeX()
{
    return this.stepSizeX;
}

getStepSizeY()
{
    return this.stepSizeY;
}

getSize()
{
    return this.size;
}

getFrontLeftCornerX()
{
    return this.centreX - this.getSize() / 2.8;
}

getFrontLeftCornerY()
{
```

```
        return this.centreY - this.getSize() / 2.8;
    }

    getFrontRightCornerX()
    {
        return this.centreX + this.getSize() / 2.8;
    }

    getFrontRightCornerY()
    {
        return this.centreY - this.getSize() / 2.8;
    }

    getFrontCentreX()
    {
        return this.centreX;
    }

    getFrontCentreY()
    {
        return this.centreY - this.getSize() / 2.8;
    }
}
```

Code Explained

Inside the constructor() method, an offscreen canvas is created to hold an image of the river. This will be used for collision detection between the tank and the river.

```
        /* this.offscreenObstaclesCtx will be used for collision detection with the river */
        this.offscreenObstacles = document.createElement('canvas');
        this.offscreenObstaclesCtx = this.offscreenObstacles.getContext('2d');
        this.offscreenObstacles.width = canvas.width;
        this.offscreenObstacles.height = canvas.height;
        this.offscreenObstaclesCtx.drawImage(riverImage, 0, 0, canvas.width, canvas.height);
```

The direction that the tank is moving in is given in degrees. The variable 'this.STEP_SIZE' determines how many pixels the tank will move forward on each call to updateState(). Higher values will cause the tank to move faster.

```
        this.STEP_SIZE = 2; // the number of pixels to move forward
        this.setDirection(direction);
```

The tank code includes methods for making the tank move and stop. The tank is initially set to be stopped.

```
        this.isMoving = false; // the tank is initially stopped
```

The updateState() method does three things:

- checks if the tank is stopped or moving. If the tank is stopped, then it does not need to update its state. This is shown in green in the code below.
- move through the sprite image, to animate the tank movement. This is shown in blue in the code below.
- move the tank forward. If the tank moves off the edge of the canvas, then it is made to reappear on the opposite side of the canvas. This is shown in red in the code below.

```
updateState()
    {
        if (!this.isMoving)
        {
            return;
        }

        this.currentSprite++;
        this.column += this.SPRITE_INCREMENT;
        if (this.currentSprite >= this.NUMBER_OF_SPRITES)
        {
            this.currentSprite = 0;
            this.row = this.START_ROW;
            this.column = this.START_COLUMN;
        }

        if (this.column < 0)
        {
            this.column = this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE - 1;
            this.row--;
        }
```

```
        this.centreX += this.stepSizeX;
        this.centreY += this.stepSizeY;

        /* if the tank goes off the canvas, then make it reappear at the opposite side of the canvas */
        if ((this.centreX - this.halfSize) > canvas.width)
        {
            this.centreX = -this.halfSize;
        }
        else if ((this.centreY - this.halfSize) > canvas.height)
        {
            this.centreY = -this.halfSize;
        }
        else if ((this.centreX + this.halfSize) < 0)
        {
            this.centreX = canvas.width + this.halfSize;
        }
        else if ((this.centreY + this.halfSize) < 0)
        {
            this.centreY = canvas.height + this.halfSize;
        }
    }
```

When rendering the tank, we must rotate the canvas so that the tank is made to point in its current direction.

```
    render()
    {
        ctx.save();
        ctx.translate(this.centreX, this.centreY);
        ctx.rotate(Math.radians(this.direction));
        ctx.translate(-this.centreX, -this.centreY);

        ctx.drawImage(this.tankImage, this.column * this.SPRITE_WIDTH, this.row * this.SPRITE_HEIGHT, this.SPRITE_WIDTH,
        ctx.restore();
    }
```

We need to account for the fact that the tank is displayed in a rotated position on the canvas. The best way to achieve this is to rotate the tank and the bullet by an amount that will bring the tank back to a position where it is not rotated. The rotation of the point is done by rotating the point by the minus angle of 'this.direction'. The rotation is about the tank's centre point. This is shown in red in the code below.
The rest of the code is a standard test of a point against a rectangle.

```
pointIsInsideTank(x, y)
    {
        /* transform the shell into the enemy tank's coordinate system */
        let transformedX = x - this.centreX;
        let transformedY = y - this.centreY;
        x = transformedX * Math.cos(Math.radians((this.direction))) - transformedY * Math.sin(Math.radians(this.directio
        y = transformedX * Math.sin(Math.radians((this.direction))) + transformedY * Math.cos(Math.radians(this.directio
        x += this.centreX;
        y += this.centreY;

        let imageTopLeftX = this.centreX - parseInt(this.size / 2);
        let imageTopLeftY = this.centreY - parseInt(this.size / 2);
        if ((x > imageTopLeftX) && (y > imageTopLeftY))
        {
            if (x > imageTopLeftX)
            {
                if ((x - imageTopLeftX) > this.size)
                {
                    return false; // to the right of the tank image
                }
            }

            if (y > imageTopLeftY)
            {
                if ((y - imageTopLeftY) > this.size)
                {
                    return false; // below the tank image
                }
            }
        }
        else // above or to the left of the tank image
        {
            return false;
        }
        return true; // inside tank image
    }
```

The The collidedWithRiver() method tests the front-left and front-right corners of the tank against the offscreen canvas that contains the river image.
The pointCollisionWithRiver() method is used for the collision detection of one point against the offscreen canvas containing the river image.

```
    collidedWithRiver()
    {
        /* test the front-left corner and the front-right corner of the tank for collision with the river */
        /* we only need to test the front of the tank, as the tank can only move forward
        if ((this.pointCollisionWithRiver(this.getFrontLeftCornerX(), this.getFrontLeftCornerY())) ||
                (this.pointCollisionWithRiver(this.getFrontRightCornerX(), this.getFrontRightCornerY())))
        {
            return true;
        }
        return false;
    }

    pointCollisionWithRiver(x, y)
    {

        let transformedX = x - this.centreX;
        let transformedY = y - this.centreY;
        x = transformedX * Math.cos(Math.radians((this.direction))) - transformedY * Math.sin(Math.radians(this.directio
        y = transformedX * Math.sin(Math.radians((this.direction))) + transformedY * Math.cos(Math.radians(this.directio
        x += this.centreX;
        y += this.centreY;

        let imageData = this.offscreenObstaclesCtx.getImageData(x, y, 1, 1);
        let data = imageData.data;
        if (data[3] !== 0)
        {
            return true;
        }
        return false;
    }
```

The positionRandomly() method places the tank ramdomly on the canvas. The calculation in red ensures that the entire tank is displayed on the canvas.

```
    positionRandomly()
    {
        this.centreX = Math.random() * (canvas.width - (this.size * 2)) + this.size;
        this.centreY = Math.random() * (canvas.height - (this.size * 2)) + this.size;
        this.setDirection(Math.random() * 360);

        // reset the tank sprite
        this.currentSprite = 0;
        this.row = this.START_ROW;
        this.column = this.START_COLUMN;
    }
```

The rest of the Tank class methods are straight forward.

PlayerTank.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class PlayerTank extends Tank
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(tankImage, riverImage, tankMovingSound, centreX, centreY, direction)
    {
        super(tankImage, riverImage, tankMovingSound, 100, centreX, centreY, direction, 1, 0); /* as this class extends
    }
}
```

EnemyTank.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class EnemyTank extends Tank
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(tankImage, riverImage, tankMovingSound, x, y, direction)
    {
        super(tankImage, riverImage, tankMovingSound, 70, x, y, direction, 2, 0);
    }
}
```

Code Explained

Code Explained

The player tank and the enemy tank use a different set of sprites, so that they are two different colours. This is done using the last two parameters of the Tank constructor, as shown in red in the code above.

Shell.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class Shell extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(explosionImage, shellExplosionSound, x, y, direction)
    {
        super(5); /* as this class extends from GameObject, you must always call super() */

        this.explosionImage = explosionImage;
        this.shellExplosionSound = shellExplosionSound;
        this.x = x;
        this.y = y;
        this.explosionTargetX = x;
        this.explosionTargetY = y;
        this.direction = direction;

        /* define the maximum range of the shell */
        /* the shell will explode here if it has not hit a target beforehand */
        this.shellRange = 200;
        this.distanceShellTravelled = gameObjects[TANK].getSize() / 2;  // the shell starts from the front of the tank's
    }

    updateState()
    {
        if (this.distanceShellTravelled < this.shellRange)
        {
            this.distanceShellTravelled += this.stepSize;
            this.explosionTargetX = this.x + (this.distanceShellTravelled * Math.sin(Math.radians(this.direction)));
            this.explosionTargetY = this.y - (this.distanceShellTravelled * Math.cos(Math.radians(this.direction)));
        }
        else
        {
            this.stopAndHide();
            gameObjects[EXPLOSION] = new Explosion(this.explosionImage, this.shellExplosionSound, this.explosionTargetX,
            gameObjects[EXPLOSION].start();
        }
    }

    getX()
    {
        return this.explosionTargetX;
    }

    getY()
    {
        return this.explosionTargetY;
    }

    getRange()
    {
        return this.shellRange;
    }

    isFiring()
    {
        return this.gameObjectIsDisplayed;
    }
}
```

Code Explained

In the constructor() method, we need to declare the range of the shell. The shell starts from the top of the barrel of the tank's gun.

```
        /* define the maximum range of the shell */
        /* the shell will explode here if it has not hit a target beforehand */
        this.shellRange = 200;
```

```
        this.distanceShellTravelled = gameObjects[TANK].getSize() / 2;  // the shell starts from the front of the tank's
```

The shell will travel until it either collides with an enemy tank or reaches its range. The collision with an enemy tank is dealt with inside the TankCanvasGame collisionDetection() method. Therefore, the updateState() method below only needs to test if the shell has reached its maximum range. If the shell has not reached its maximum range, then move it to its next position. If the shell has reached it maximum range, then hide the shell and show an explosion.

```
    updateState()
    {
        if (this.distanceShellTravelled < this.shellRange)
        {
            this.distanceShellTravelled += this.stepSize;
            this.explosionTargetX = this.x + (this.distanceShellTravelled * Math.sin(Math.radians(this.direction)));
            this.explosionTargetY = this.y - (this.distanceShellTravelled * Math.cos(Math.radians(this.direction)));
        }
        else
        {
            this.stopAndHide();
            gameObjects[EXPLOSION] = new Explosion(this.explosionImage, this.shellExplosionSound, this.explosionTargetX,
            gameObjects[EXPLOSION].start();
        }
    }
```

The rest of the Shell code is straight-forward.

FireShellAnimation.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class FireShellAnimation extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(tankImage, fireShellSound, centreX, centreY, direction)
    {
        super(50); /* as this class extends from GameObject, you must always call super() */

        this.tankImage = tankImage;

        this.centreX = centreX;
        this.centreY = centreY;
        this.direction = direction;

        this.NUMBER_OF_SPRITES = 3; // the number of sprites in the sprite image
        this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE = 8; // the number of columns in the sprite image
        this.NUMBER_OF_ROWS_IN_SPRITE_IMAGE = 4; // the number of columns in the sprite image
        //    this.NUMBER_OF_SPRITES = 3; // the number of sprites in the sprite image
        this.START_ROW = 2;
        this.START_COLUMN = 1;

        this.currentSprite = 0; // the current sprite to be displayed from the sprite image
        this.row = this.START_ROW; // current row in sprite image
        this.column = this.START_COLUMN; // current column in sprite image
        this.spriteIncrement = 1;
        this.SPRITE_WIDTH;
        this.SPRITE_HEIGHT;
        this.size = 50;

        fireShellSound.currentTime = 0;
        fireShellSound.play();

        this.SPRITE_WIDTH = (tankImage.width / this.NUMBER_OF_COLUMNS_IN_SPRITE_IMAGE);
        this.SPRITE_HEIGHT = (tankImage.height / this.NUMBER_OF_ROWS_IN_SPRITE_IMAGE);
    }

    updateState()
    {
        this.currentSprite++;
        if (this.currentSprite >= this.NUMBER_OF_SPRITES)
        {
            this.stopAndHide();
        }

        this.column += this.spriteIncrement;
    }

    render()
    {
```

```
        ctx.save();
        ctx.translate(this.centreX, this.centreY);
        ctx.rotate(Math.radians(this.direction));
        ctx.translate(-this.centreX, -this.centreY);

        ctx.drawImage(this.tankImage, this.column * this.SPRITE_WIDTH, this.row * this.SPRITE_HEIGHT, this.SPRITE_WIDTH,
        ctx.restore();
    }
}
```

Code Explained

The FireShellAnimation uses standard sprite animation code.

Case Study... Multiple Choice Quiz Game

The aim of this game is to answer the multiple-choice questions.

This game shows:

- use of a json file to hold game data
- creation of customised Button sub-class

Play game

Download .zip file

The game data is held in a json file, as shown below. Each answer can contain text and/or an image.

multiple_choice_quiz.json.js

```
{
    "questions":
            [
                {
                    "question": "How many countries are there in the EU?",
                    "correct": 4,
                    "answers":
                            [
                                {
                                    "text": "17",
                                    "image": ""
                                },
                                {
                                    "text": "18",
                                    "image": ""
                                },
                                {
                                    "text": "27",
                                    "image": ""
                                },
                                {
                                    "text": "28",
                                    "image": ""
                                }
                            ]
                },
                {
                    "question": "Click on the map that highlights Poland",
                    "correct": 2,
                    "answers":
                            [
                                {
                                    "text": "",
                                    "image": "images/map_czech_republic.png"
                                },
                                {
```

```
                                "text": "",
                                "image": "images/map_poland.png"
                            },
                            {
                                "text": "",
                                "image": "images/map_belarus.png"
                            },
                            {
                                "text": "",
                                "image": "images/map_ukraine.png"
                            }

                        ]
                },
                {
                    "question": "Click on the Finnish flag",
                    "correct": 3,
                    "answers":
                            [
                            {
                                "text": "",
                                "image": "images/flag_denmark.png"
                            },
                            {
                                "text": "",
                                "image": "images/flag_sweden.png"
                            },
                            {
                                "text": "",
                                "image": "images/flag_finland.png"
                            },
                            {
                                "text": "",
                                "image": "images/flag_norway.png"
                            }
                        ]
                }
            ]
}
```

multiple_choice_quiz_game.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.             */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */




/******************** Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */


/* The various gameObjects */
/* These are the positions that each gameObject is held in the gameObjects[] array */
const QUESTION = 0;
const WIN_MESSAGE = 1;
/******************** END OF Declare game specific data and functions ****************/







/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                   */
    /* 2. store the game gameObjects in an array                     */
    /* 3. create a new Game to display the game gameObjects          */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */
```

```
    // make the canvas wider for this example

    /* END OF game specific code. */

    /* Always create a game that uses the gameObject array */
    let game = null;

    /* The game must be loaded from the json file before it is played */
    fetch('json/multiple_choice_quiz.json').then(function (response)
    {
        return response.json();
    }).then(function (jsonData)
    {
        game = new MultipleChoiceQuizCanvasGame(jsonData);

        /* Always play the game */
        game.start();
    });



    /* If they are needed, then include any game-specific mouse and keyboard listners */
    document.getElementById("gameCanvas").addEventListener("mousedown", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            game.mousedown(mouseX, mouseY);
        }
    });

    document.addEventListener("mousemove", function (e)
    {
        if (e.which === 0) // no button selected
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            game.mousemove(mouseX, mouseY);
        }
    });
}
```

Code Explained

An AJAX fetch is done on the json file. The MultipleChoiceQuizCanvasGame game object cannot be created and the game cannot start until the json data has been fetched, as the jsonData is needed to create the MultipleChoiceQuizCanvasGame (as shown in red).

```
    /* The game must be loaded from the json file before it is played */
    fetch('json/multiple_choice_quiz.json').then(function (response)
    {
        return response.json();
    }).then(function (jsonData)
    {
        game = new MultipleChoiceQuizCanvasGame(jsonData);

        /* Always play the game */
        game.start();
    });
```

The mousedown and mousemove events are passed to the game for handling, as shown in red below.

```
/* If they are needed, then include any game-specific mouse and keyboard listners */
    document.getElementById("gameCanvas").addEventListener("mousedown", function (e)
    {
        if (e.which === 1)  // left mouse button
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            game.mousedown(mouseX, mouseY);
        }
    });
```

```
    document.addEventListener("mousemove", function (e)
    {
        if (e.which === 0) // no button selected
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            game.mousemove(mouseX, mouseY);
        }
    });
```

MultipleChoiceQuizCanvasGame.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.       */
/* A CanvasGame that implements collision detection.                       */

let buttons = [];

class MultipleChoiceQuizCanvasGame extends CanvasGame
{
    constructor(jsonData)
    {
        super();
        this.jsonData = jsonData;

        /* If the current question has images on the buttons, they will be stored here. */
        this.currentImage = [new Image(), new Image(), new Image(), new Image()];

        this.score = 0;
        this.currentQuestion = 0;

        /* Load the first question */
        this.loadQuestion();
    }

    render()
    {
        super.render();
        for (let i = 0; i < buttons.length; i++)
        {
            if (buttons[i].isDisplayed())
            {
                buttons[i].render();
            }
        }
    }

    mousedown(x, y)
    {
        for (let i = 0; i < buttons.length; i++)
        {
            if (buttons[i].isDisplayed())
            {
                if (buttons[i].pointIsInsideBoundingRectangle(x, y))
                {
                    if (buttons[i].isThisTheCorrectAnswer())
                    {
                        this.score++;
                    }

                    this.currentQuestion++;

                    /* Check to see if all of the questions have been completed */
                    if (this.gameIsOver())
                    {
                        for (let i = 0; i < gameObjects.length; i++) /* stop all gameObjects from animating */
                        {
                            gameObjects[i].stopAndHide();
                        }
                        for (let i = 0; i < buttons.length; i++) /* stop all buttons from animating */
                        {
                            buttons[i].stopAndHide();
                        }

                        gameObjects[WIN_MESSAGE] = new StaticText("Your score is: " + this.score, 15, 260, "Times Roman"
                        gameObjects[WIN_MESSAGE].start(); /* render win message */
                    }
                    else /* Load the next question */
                    {
```

```
                        this.loadQuestion();
                    }

                    return; /* There is no need to test any other items in the for-loop */
                }
            }
        }
    }

    mousemove(x, y)
    {
        for (let i = 0; i < buttons.length; i++)
        {
            if (buttons[i].isDisplayed())
            {
                buttons[i].pointIsInsideBoundingRectangle(x, y); /* Used to highlight the button that the mouse is curre
            }
        }
    }

    gameIsOver()
    {
        /* Test for end-of-game */
        return (this.currentQuestion === this.jsonData.questions.length);
    }

    loadQuestion()
    {
        /* Load the question from the json */
        gameObjects[QUESTION] = new StaticText(this.jsonData.questions[this.currentQuestion].question, STATIC_TEXT_CENTR
        gameObjects[QUESTION].start();

        /* Load the buttons from the json */
        for (let i = 0; i < this.jsonData.questions[this.currentQuestion].answers.length; i++)
        {
            /* Identify the button with the correct answer */
            let isCorrect = false;
            if (this.jsonData.questions[this.currentQuestion].correct === i + 1)
            {
                isCorrect = true;
            }

            /* Buttons might or might not have an image. We need to deal with both cases */
            if (this.jsonData.questions[this.currentQuestion].answers[i].image !== "")
            {
                this.currentImage[i].src = this.jsonData.questions[this.currentQuestion].answers[i].image;

                buttons[i] = new MultipleChoiceQuizButton(isCorrect, (canvas.width * 0.5) * (i % 2), (canvas.height * 0.
            }
            else
            {
                buttons[i] = new MultipleChoiceQuizButton(isCorrect, (canvas.width * 0.5) * (i % 2), (canvas.height * 0.
            }

            /* Display the button */
            buttons[i].start();
        }
    }
}
```

Code Explained

the buttons[] array will hold the four buttons for the current question.

```
let buttons = [];
```

The json file that hold the game data can only contain strings. However, the Button class takes an image object rather than a string filename. Therefore, we need to store the images that are contained in the json strings into Image objects prior to creating Button objects. The array 'this.currentImage' will be used to hold the four buttons' images.

```
constructor(jsonData)
    {
        super();
        this.jsonData = jsonData;

        /* If the current question has images on the buttons, they will be stored here. */
        this.currentImage = [new Image(), new Image(), new Image(), new Image()];

        this.score = 0;
        this.currentQuestion = 0;
```

```
        /* Load the first question */
        this.loadQuestion();
    }
```

This game extend from the class CanvasGame. The render() method will call super.render() to render the gameObjects[] that are contained in the game.
The game-specific button objects are then rendered.

```
    render()
    {
        super.render();

        for (let i = 0; i < buttons.length; i++)
        {
            if (buttons[i].isDisplayed())
            {
                buttons[i].render();
            }
        }
    }
```

The mousedown() method steps through each button and checks if it has been clicked.
If the button contains the correct answer, then the score is incremented (shown in green).
The current question is incremented (shown in blue).
A test is then done to see if the game is over. If the game is over, then the score is displayed. Otherwise, the next question is loaded (shown in red).

```
mousedown(x, y)
    {
        for (let i = 0; i < buttons.length; i++)
        {
            if (buttons[i].isDisplayed())
            {
                if (buttons[i].pointIsInsideBoundingRectangle(x, y))
                {
                    if (buttons[i].isThisTheCorrectAnswer())
                    {
                        this.score++;
                    }

                    this.currentQuestion++;

                    /* Check to see if all of the questions have been completed */
                    if (this.gameIsOver())
                    {
                        for (let i = 0; i < gameObjects.length; i++) /* stop all gameObjects from animating */
                        {
                            gameObjects[i].stopAndHide();
                        }
                        for (let i = 0; i < buttons.length; i++) /* stop all buttons from animating */
                        {
                            buttons[i].stopAndHide();
                        }

                        gameObjects[WIN_MESSAGE] = new StaticText("Your score is: " + this.score, 15, 260, "Times Roman"
                        gameObjects[WIN_MESSAGE].start(); /* render win message */
                    }
                    else /* Load the next question */
                    {
                        this.loadQuestion();
                    }

                    return; /* There is no need to test any other items in the for-loop */
                }
            }
        }
    }
```

The mousemove highlights the button that the mouse is currently hovering over

```
    mousemove(x, y)
    {
        for (let i = 0; i < buttons.length; i++)
        {
            if (buttons[i].isDisplayed())
            {
                buttons[i].pointIsInsideBoundingRectangle(x, y); /* Used to highlight the button that the mouse is curre
            }
        }
    }
```

MultipleChoiceQuizButton.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */


class MultipleChoiceQuizButton extends Button
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(isCorrectAnswer, x, y, width, height, text, backgroundImage)
    {
        super(x, y, width, height, text, true, backgroundImage, 20, "Times Roman", "Black", "#ee6", "#999", 5);

        /* These variables depend on the object */
        this.isCorrectAnswer = isCorrectAnswer;
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.text = text;
        this.backgroundImage = backgroundImage;
    }

    isThisTheCorrectAnswer()
    {
        return this.isCorrectAnswer;
    }
}
```

Code Explained

The game is over if all of the questions from the json file have been completed.

```
    gameIsOver()
    {
        /* Test for end-of-game */
        return (this.currentQuestion === this.jsonData.questions.length);
    }
```

The loadQuestion() method displays the 'this.currentQuestion' from jsonData. Each question consists of a question and four answer buttons.
The question is placed in gameObjects[QUESTION] (shown in blue)
The four answers are placed in four buttons. When creating the buttons, we need to account for the fact that an answer might or might not contain an image (as shown in green). If a button does contain an image, then the json file will hold the image's name. The MultipleChoiceQuizButton class requires an image rather than an image filename. Therefore, we need to create an Image() object to hold the image, so that we can create a new MultipleChoiceQuizButton object. This is shown in red.

```
    loadQuestion()
    {
        /* Load the question from the json */
        gameObjects[QUESTION] = new StaticText(this.jsonData.questions[this.currentQuestion].question, STATIC_TEXT_CENTR
        gameObjects[QUESTION].start();

        /* Load the buttons from the json */
        for (let i = 0; i < this.jsonData.questions[this.currentQuestion].answers.length; i++)
        {
            /* Identify the button with the correct answer */
            let isCorrect = false;
            if (this.jsonData.questions[this.currentQuestion].correct === i + 1)
            {
                isCorrect = true;
            }

            /* Buttons might or might not have an image. We need to deal with both cases */
            if (this.jsonData.questions[this.currentQuestion].answers[i].image !== "")
            {
                this.currentImage[i].src = this.jsonData.questions[this.currentQuestion].answers[i].image;

                buttons[i] = new MultipleChoiceQuizButton(isCorrect, (canvas.width * 0.5) * (i % 2), (canvas.height * 0.
            }
            else
            {
                buttons[i] = new MultipleChoiceQuizButton(isCorrect, (canvas.width * 0.5) * (i % 2), (canvas.height * 0.
```

```
            }

            /* Display the button */
            buttons[i].start();
        }
    }
```

Case Study... Teddy

The aim of this game is to use the mouse to move Teddy's limbs.

This game shows:

- a complex animated gameObject
- multiple canvas rotation
- recursion
- mouse input

Drag on the hotspots to move Teddy.

Play game

Download .zip file

teddy_game.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland.             */
/* There should always be a javaScript file with the same name as the html file. */
/* This file always holds the playGame function().                               */
/* It also holds game specific code, which will be different for each game       */



/******************* Declare game specific global data and functions ****************/
/* images must be declared as global, so that they will load before the game starts  */
let backgroundImage = new Image();
backgroundImage.src = "images/sunshine.png";

let torsoImage = new Image();
torsoImage.src = "images/torso.png";

let headImage = new Image();
headImage.src = "images/head.png";

let leftArmUpperImage = new Image();
leftArmUpperImage.src = "images/left_arm_upper.png";

let leftArmLowerImage = new Image();
leftArmLowerImage.src = "images/left_arm_lower.png";

let leftArmHandImage = new Image();
leftArmHandImage.src = "images/left_arm_hand.png";

let rightArmUpperImage = new Image();
rightArmUpperImage.src = "images/right_arm_upper.png";

let rightArmLowerImage = new Image();
rightArmLowerImage.src = "images/right_arm_lower.png";

let rightArmHandImage = new Image();
rightArmHandImage.src = "images/right_arm_hand.png";

let leftLegUpperImage = new Image();
leftLegUpperImage.src = "images/left_leg_upper.png";
```

```
let leftLegLowerImage = new Image();
leftLegLowerImage.src = "images/left_leg_lower.png";

let leftLegFootImage = new Image();
leftLegFootImage.src = "images/left_leg_foot.png";

let rightLegUpperImage = new Image();
rightLegUpperImage.src = "images/right_leg_upper.png";

let rightLegLowerImage = new Image();
rightLegLowerImage.src = "images/right_leg_lower.png";

let rightLegFootImage = new Image();
rightLegFootImage.src = "images/right_leg_foot.png";

let oldMouseX = 0; // oldMouseX and oldMouseY hold the previous mouse position. This is needed to calculate the
let oldMouseY = 0; // direction that the mouse is moving in, so that we can rotate around a body part's centre of rotati

const BACKGROUND = 0;
const TEDDY1 = 1;
const TEDDY2 = 2;

let FIRST_TEDDY = TEDDY1;
let LAST_TEDDY = TEDDY2;
/****************** END OF Declare game specific data and functions ****************/




/* Always have a playGame() function                                    */
/* However, the content of this function will be different for each game */
function playGame()
{
    /* We need to initialise the game objects outside of the Game class */
    /* This function does this initialisation.                          */
    /* This function will:                                              */
    /* 1. create the various game game gameObjects                  */
    /* 2. store the game gameObjects in an array                    */
    /* 3. create a new Game to display the game gameObjects         */
    /* 4. start the Game                                                */


    /* Create the various gameObjects for this game. */
    /* This is game specific code. It will be different for each game, as each game will have it own gameObjects */
    gameObjects[BACKGROUND] = new StaticImage(backgroundImage, 0, 0, canvas.width, canvas.height);
    gameObjects[TEDDY1] = new Teddy(160, 250, 0.7, torsoImage, headImage, leftArmUpperImage, leftArmLowerImage, leftArmH
    gameObjects[TEDDY2] = new Teddy(400, 320, 0.4, torsoImage, headImage, leftArmUpperImage, leftArmLowerImage, leftArmH

    /* show hotspots on Teddy */
    /* Teddy hotspots can be draged. This will allow us to use the mouse to move the Teddy body parts */
    for (let teddy = FIRST_TEDDY; teddy <= LAST_TEDDY; teddy++)
    {
        if (gameObjects[teddy] !== undefined)
        {
            gameObjects[teddy].setHotSpotDrawState(true);
        }
    }
    /* END OF game specific code. */

    /* Always create a game that uses the gameObject array */
    let game = new CanvasGame();

    /* Always play the game */
    game.start();

    /* If they are needed, then include any game-specific mouse and keyboard listners */
    let NO_BODYPART_SELECTED = -1;
    let selectedTeddy = TEDDY1;
    let selectedBodyPart = NO_BODYPART_SELECTED;
    document.getElementById('gameCanvas').addEventListener('mousedown', function (e)
    {
        if (e.which === 1)  // left mouse button pressed
        {
            //  mouseIsDown = true;
        }
        let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
        let mouseX = e.clientX - canvasBoundingRectangle.left;
        let mouseY = e.clientY - canvasBoundingRectangle.top;

        document.body.style.cursor = "move";
        for (let i = 0; i < gameObjects[TEDDY1].getNumberOfBodyParts(); i++)
```

```
        {
            for (let teddy = FIRST_TEDDY; teddy <= LAST_TEDDY; teddy++)
            {
                if (gameObjects[teddy] !== undefined)
                {
                    if (gameObjects[teddy].getBodyPart(i).isHotSpot(mouseX, mouseY) === true)
                    {
                        selectedTeddy = teddy;
                        selectedBodyPart = i;
                    }
                }
            }
        }
        oldMouseX = mouseX;
        oldMouseY = mouseY;
    });

    let ROTATION_STEP_SIZE = 1;
    document.getElementById('gameCanvas').addEventListener('mousemove', function (e)
    {
        // only process 'mousemove' when a body part has been selected
        if (selectedBodyPart !== NO_BODYPART_SELECTED)
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            // rotate the selected bodyPart
            gameObjects[selectedTeddy].getBodyPart(selectedBodyPart).changeRotationValue(mouseX, mouseY, ROTATION_STEP_S

           // update the x and y location
           oldMouseX = mouseX;
           oldMouseY = mouseY;
        }
    });

    document.addEventListener('mouseup', function (e)
    {
        selectedBodyPart = NO_BODYPART_SELECTED;
        document.body.style.cursor = "auto";
    });
}
```

Code Explained

Teddy is made up by joining images of his 14 body parts together.

```
let backgroundImage = new Image();
backgroundImage.src = "images/sunshine.png";

let torsoImage = new Image();
torsoImage.src = "images/torso.png";

let headImage = new Image();
headImage.src = "images/head.png";

let leftArmUpperImage = new Image();
leftArmUpperImage.src = "images/left_arm_upper.png";

let leftArmLowerImage = new Image();
leftArmLowerImage.src = "images/left_arm_lower.png";

let leftArmHandImage = new Image();
leftArmHandImage.src = "images/left_arm_hand.png";

let rightArmUpperImage = new Image();
rightArmUpperImage.src = "images/right_arm_upper.png";

let rightArmLowerImage = new Image();
rightArmLowerImage.src = "images/right_arm_lower.png";

let rightArmHandImage = new Image();
rightArmHandImage.src = "images/right_arm_hand.png";

let leftLegUpperImage = new Image();
leftLegUpperImage.src = "images/left_leg_upper.png";

let leftLegLowerImage = new Image();
leftLegLowerImage.src = "images/left_leg_lower.png";

let leftLegFootImage = new Image();
leftLegFootImage.src = "images/left_leg_foot.png";
```

```
let rightLegUpperImage = new Image();
rightLegUpperImage.src = "images/right_leg_upper.png";

let rightLegLowerImage = new Image();
rightLegLowerImage.src = "images/right_leg_lower.png";

let rightLegFootImage = new Image();
rightLegFootImage.src = "images/right_leg_foot.png";
```

We need to keep track of the last mouse, so that we can detect the direction that the mouse is moving. This is needed when we drag on a hotspot. It allows us to know what way the hotspot should rotate.

```
let oldMouseX = 0; // oldMouseX and oldMouseY hold the previous mouse position. This is needed to calculate the
let oldMouseY = 0; // direction that the mouse is moving in, so that we can rotate around a body part's centre of rotati
```

There are three gameObjects in this game.
FIRST_TEDDY and LAST_TEDDY are used to identify the position of the first and last Teddy gameObjects. Keeping track of the first and last Teddy objects will allow us to add more Teddy objects to the game, if we wish.

```
const BACKGROUND = 0;
const TEDDY1 = 1;
const TEDDY2 = 2;

let FIRST_TEDDY = TEDDY1;
let LAST_TEDDY = TEDDY2;
```

Below is an example where FIRST_TEDDY and LAST_TEDDY are used.
Here, we step through each Teddy object and enable it to show hotspots.

```
    /* show hotspots on Teddy */
    /* Teddy hotspots can be draged. This will allow us to use the mouse to move the Teddy body parts */
    for (let teddy = FIRST_TEDDY; teddy <= LAST_TEDDY; teddy++)
    {
        if (gameObjects[teddy] !== undefined)
        {
            gameObjects[teddy].setHotSpotDrawState(true);
        }
    }
```

When the mouse is pressed down (inside the 'mousedown' event handler), we check all of the hotspots on each Teddy. If we find that the mouse was clicked on a hotspot, then we set selectedTeddy and selectedBodyPart to identify the Teddy and BodyPart.
When the mouse is moved (inside the 'mousemove' event handler), we check if a bodyPart was selected. If yes, then we rotate that bodyPart. The selected bodyPart is rotated (by calling its changeRotationValue() method in the code below). The rotation is based on the direction that the mouse is moving. The oldMouseX, oldMouseY, mouseX and mouseY positions will be used to calculate the direction of that the mouse is moving.

```
    /* If they are needed, then include any game-specific mouse and keyboard listners */
    let NO_BODYPART_SELECTED = -1;
    let selectedTeddy = TEDDY1;
    let selectedBodyPart = NO_BODYPART_SELECTED;
    document.getElementById('gameCanvas').addEventListener('mousedown', function (e)
    {
        if (e.which === 1)  // left mouse button pressed
        {
            //  mouseIsDown = true;
        }
        let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
        let mouseX = e.clientX - canvasBoundingRectangle.left;
        let mouseY = e.clientY - canvasBoundingRectangle.top;

        document.body.style.cursor = "move";
        for (let i = 0; i < gameObjects[TEDDY1].getNumberOfBodyParts(); i++)
        {
            for (let teddy = FIRST_TEDDY; teddy <= LAST_TEDDY; teddy++)
            {
                if (gameObjects[teddy] !== undefined)
                {
                    if (gameObjects[teddy].getBodyPart(i).isHotSpot(mouseX, mouseY) === true)
                    {
                        selectedTeddy = teddy;
                        selectedBodyPart = i;
                    }
                }
            }
        }
        oldMouseX = mouseX;
        oldMouseY = mouseY;
    });

    let ROTATION_STEP_SIZE = 1;
```

```
    document.getElementById('gameCanvas').addEventListener('mousemove', function (e)
    {
        // only process 'mousemove' when a body part has been selected
        if (selectedBodyPart !== NO_BODYPART_SELECTED)
        {
            let canvasBoundingRectangle = document.getElementById("gameCanvas").getBoundingClientRect();
            let mouseX = e.clientX - canvasBoundingRectangle.left;
            let mouseY = e.clientY - canvasBoundingRectangle.top;

            // rotate the selected bodyPart
            gameObjects[selectedTeddy].getBodyPart(selectedBodyPart).changeRotationValue(mouseX, mouseY, ROTATION_STEP_S

            // update the x and y location
            oldMouseX = mouseX;
            oldMouseY = mouseY;
        }
    });

    document.addEventListener('mouseup', function (e)
    {
        selectedBodyPart = NO_BODYPART_SELECTED;
        document.body.style.cursor = "auto";
    });
}
```

Teddy.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */

class Teddy extends GameObject
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(torsoCentreX, torsoCentreY, scale, torsoImage, headImage, leftArmUpperImage, leftArmLowerImage, leftArmH
    {
        super(delay); /* as this class extends from GameObject, you must always call super() */

        this.scale = scale; // the scale is needed to convert the physical image data into the required size on the canv

        /* These variables depend on the object */
        // assign locations in the body array for each body part
        this.HEAD = 0;
        this.TORSO = 1;
        this.LEFT_ARM_UPPER = 2;
        this.LEFT_ARM_LOWER = 3;
        this.LEFT_ARM_HAND = 4;
        this.RIGHT_ARM_UPPER = 5;
        this.RIGHT_ARM_LOWER = 6;
        this.RIGHT_ARM_HAND = 7;
        this.LEFT_LEG_UPPER = 8;
        this.LEFT_LEG_LOWER = 9;
        this.LEFT_LEG_FOOT = 10;
        this.RIGHT_LEG_UPPER = 11;
        this.RIGHT_LEG_LOWER = 12;
        this.RIGHT_LEG_FOOT = 13;
        this.NUMBER_OF_BODY_PARTS = 14;

        this.bodyPart = [];

        //  TeddyBodyPart(name, scale, sourceImage, width, height, thetha, centreOfRotationX, centreOfRotationY, hotSpot
        this.bodyPart[this.TORSO] = new TeddyBodyPart(scale, torsoImage, 200, 200, 0.0, 100, 150, 100, 75, 75);
        this.bodyPart[this.HEAD] = new TeddyBodyPart(scale, headImage, 100, 100, 0.0, 50, 80, 50, 30, 40);

        this.bodyPart[this.LEFT_ARM_UPPER] = new TeddyBodyPart(scale, leftArmUpperImage, 100, 50, Math.PI / 3, 0, 25, 70
        this.bodyPart[this.LEFT_ARM_LOWER] = new TeddyBodyPart(scale, leftArmLowerImage, 100, 50, Math.PI / 3, 0, 25, 70
        this.bodyPart[this.LEFT_ARM_HAND] = new TeddyBodyPart(scale, leftArmHandImage, 50, 50, Math.PI / 3, 5, 35, 35, 2

        this.bodyPart[this.RIGHT_ARM_UPPER] = new TeddyBodyPart(scale, rightArmUpperImage, 100, 50, Math.PI * 2.7, 0, 25
        this.bodyPart[this.RIGHT_ARM_LOWER] = new TeddyBodyPart(scale, rightArmLowerImage, 100, 50, Math.PI * 2.7, 0, 25
        this.bodyPart[this.RIGHT_ARM_HAND] = new TeddyBodyPart(scale, rightArmHandImage, 50, 50, Math.PI * 2.7, 5, 15, 3

        this.bodyPart[this.LEFT_LEG_UPPER] = new TeddyBodyPart(scale, leftLegUpperImage, 100, 100, 0.0, 50, 10, 60, 50,
        this.bodyPart[this.LEFT_LEG_LOWER] = new TeddyBodyPart(scale, leftLegLowerImage, 50, 100, 0.0, 25, 10, 25, 50, 4
        this.bodyPart[this.LEFT_LEG_FOOT] = new TeddyBodyPart(scale, leftLegFootImage, 60, 50, 0.0, 30, 5, 30, 55, 30);

        this.bodyPart[this.RIGHT_LEG_UPPER] = new TeddyBodyPart(scale, rightLegUpperImage, 100, 100, 0.0, 50, 10, 40, 50
        this.bodyPart[this.RIGHT_LEG_LOWER] = new TeddyBodyPart(scale, rightLegLowerImage, 50, 100, 0.0, 25, 10, 25, 50,
        this.bodyPart[this.RIGHT_LEG_FOOT] = new TeddyBodyPart(scale, rightLegFootImage, 60, 50, 0.0, 30, 5, 30, 55, 30)

        // set the dependencies
        // torso is dependent on the canvas.
```

```
        // place the torso in the centre of the canvas
        this.bodyPart[this.TORSO].setInitialParentXandY(torsoCentreX, torsoCentreY);

        // all other body parts are dependent on the position of their parent
        // head
        this.bodyPart[this.TORSO].setChild(this.bodyPart[this.HEAD], 100, 10);

        // left arm
        this.bodyPart[this.TORSO].setChild(this.bodyPart[this.LEFT_ARM_UPPER], 180, 50);
        this.bodyPart[this.LEFT_ARM_UPPER].setChild(this.bodyPart[this.LEFT_ARM_LOWER], 80, 25);
        this.bodyPart[this.LEFT_ARM_LOWER].setChild(this.bodyPart[this.LEFT_ARM_HAND], 100, 25);

        // right arm
        this.bodyPart[this.TORSO].setChild(this.bodyPart[this.RIGHT_ARM_UPPER], 20, 50);
        this.bodyPart[this.RIGHT_ARM_UPPER].setChild(this.bodyPart[this.RIGHT_ARM_LOWER], 80, 25);
        this.bodyPart[this.RIGHT_ARM_LOWER].setChild(this.bodyPart[this.RIGHT_ARM_HAND], 100, 25);

        // left leg
        this.bodyPart[this.TORSO].setChild(this.bodyPart[this.LEFT_LEG_UPPER], 135, 190);
        this.bodyPart[this.LEFT_LEG_UPPER].setChild(this.bodyPart[this.LEFT_LEG_LOWER], 60, 90);
        this.bodyPart[this.LEFT_LEG_LOWER].setChild(this.bodyPart[this.LEFT_LEG_FOOT], 25, 70);

        // right leg
        this.bodyPart[this.TORSO].setChild(this.bodyPart[this.RIGHT_LEG_UPPER], 70, 190);
        this.bodyPart[this.RIGHT_LEG_UPPER].setChild(this.bodyPart[this.RIGHT_LEG_LOWER], 40, 90);
        this.bodyPart[this.RIGHT_LEG_LOWER].setChild(this.bodyPart[this.RIGHT_LEG_FOOT], 25, 70);
    }

    render()
    {
        // place all of the bodyParts in the correct position
        this.bodyPart[this.TORSO].setChildrenPositions();

        // draw all of the body parts on the main game canvas
        for (let i = this.bodyPart.length - 1; i >= 0; i--)
        {
            this.bodyPart[i].render();
        }
    }

    getBodyPart(i)
    {
        return this.bodyPart[i];
    }

    setHotSpotDrawState(state)
    {
        for (let i = 0; i < this.NUMBER_OF_BODY_PARTS; i++)
        {
            this.bodyPart[i].setDrawHotSpot(state);
        }
    }

    getNumberOfBodyParts()
    {
        return this.NUMBER_OF_BODY_PARTS;
    }
}
```

Code Explained

The parameter 'scale' is used to set the size of a Teddy. It is sized with a sclae that is relative to the canvas width and height.
Each of the bodyParts of the Teddy are given an identifier (as shown in green below).
The bodyPart[] array is used to hold the bodyParts (as shown in red below).
The child dependencies of each bodyParts are set up using the BodyPart's setChild() method (as shown in blue below). The linking of parent and child bodyParts allows us to rotate child bodyParts whenever a parent bodyPart has been rotated.

```
    constructor(torsoCentreX, torsoCentreY, scale, torsoImage, headImage, leftArmUpperImage, leftArmLowerImage, leftArmH
    {
        super(delay); /* as this class extends from GameObject, you must always call super() */

        this.scale = scale; // the scale is needed to convert the physical image data into the required size on the canv

        /* These variables depend on the object */
        // assign locations in the body array for each body part
        this.HEAD = 0;
        this.TORSO = 1;
        this.LEFT_ARM_UPPER = 2;
        this.LEFT_ARM_LOWER = 3;
        this.LEFT_ARM_HAND = 4;
        this.RIGHT_ARM_UPPER = 5;
        this.RIGHT_ARM_LOWER = 6;
```

```
        this.RIGHT_ARM_HAND = 7;
        this.LEFT_LEG_UPPER = 8;
        this.LEFT_LEG_LOWER = 9;
        this.LEFT_LEG_FOOT = 10;
        this.RIGHT_LEG_UPPER = 11;
        this.RIGHT_LEG_LOWER = 12;
        this.RIGHT_LEG_FOOT = 13;
        this.NUMBER_OF_BODY_PARTS = 14;

        this.bodyPart = [];

        //  TeddyBodyPart(name, scale, sourceImage, width, height, thetha, centreOfRotationX, centreOfRotationY, hotSpot
        this.bodyPart[this.TORSO] = new TeddyBodyPart(scale, torsoImage, 200, 200, 0.0, 100, 150, 100, 75, 75);
        this.bodyPart[this.HEAD] = new TeddyBodyPart(scale, headImage, 100, 100, 0.0, 50, 80, 50, 30, 40);

        this.bodyPart[this.LEFT_ARM_UPPER] = new TeddyBodyPart(scale, leftArmUpperImage, 100, 50, Math.PI / 3, 0, 25, 70
        this.bodyPart[this.LEFT_ARM_LOWER] = new TeddyBodyPart(scale, leftArmLowerImage, 100, 50, Math.PI / 3, 0, 25, 70
        this.bodyPart[this.LEFT_ARM_HAND] = new TeddyBodyPart(scale, leftArmHandImage, 50, 50, Math.PI / 3, 5, 35, 35, 2

        this.bodyPart[this.RIGHT_ARM_UPPER] = new TeddyBodyPart(scale, rightArmUpperImage, 100, 50, Math.PI * 2.7, 0, 25
        this.bodyPart[this.RIGHT_ARM_LOWER] = new TeddyBodyPart(scale, rightArmLowerImage, 100, 50, Math.PI * 2.7, 0, 25
        this.bodyPart[this.RIGHT_ARM_HAND] = new TeddyBodyPart(scale, rightArmHandImage, 50, 50, Math.PI * 2.7, 5, 15, 3

        this.bodyPart[this.LEFT_LEG_UPPER] = new TeddyBodyPart(scale, leftLegUpperImage, 100, 100, 0.0, 50, 10, 60, 50,
        this.bodyPart[this.LEFT_LEG_LOWER] = new TeddyBodyPart(scale, leftLegLowerImage, 50, 100, 0.0, 25, 10, 25, 50, 4
        this.bodyPart[this.LEFT_LEG_FOOT] = new TeddyBodyPart(scale, leftLegFootImage, 60, 50, 0.0, 30, 5, 30, 55, 30);

        this.bodyPart[this.RIGHT_LEG_UPPER] = new TeddyBodyPart(scale, rightLegUpperImage, 100, 100, 0.0, 50, 10, 40, 50
        this.bodyPart[this.RIGHT_LEG_LOWER] = new TeddyBodyPart(scale, rightLegLowerImage, 50, 100, 0.0, 25, 10, 25, 50,
        this.bodyPart[this.RIGHT_LEG_FOOT] = new TeddyBodyPart(scale, rightLegFootImage, 60, 50, 0.0, 30, 5, 30, 55, 30)

        // set the dependencies
        // torso is dependent on the canvas.
        // place the torso in the centre of the canvas
        this.bodyPart[this.TORSO].setInitialParentXandY(torsoCentreX, torsoCentreY);

        // all other body parts are dependent on the position of their parent
        // head
        this.bodyPart[this.TORSO].setChild(this.bodyPart[this.HEAD], 100, 10);

        // left arm
        this.bodyPart[this.TORSO].setChild(this.bodyPart[this.LEFT_ARM_UPPER], 180, 50);
        this.bodyPart[this.LEFT_ARM_UPPER].setChild(this.bodyPart[this.LEFT_ARM_LOWER], 80, 25);
        this.bodyPart[this.LEFT_ARM_LOWER].setChild(this.bodyPart[this.LEFT_ARM_HAND], 100, 25);

        // right arm
        this.bodyPart[this.TORSO].setChild(this.bodyPart[this.RIGHT_ARM_UPPER], 20, 50);
        this.bodyPart[this.RIGHT_ARM_UPPER].setChild(this.bodyPart[this.RIGHT_ARM_LOWER], 80, 25);
        this.bodyPart[this.RIGHT_ARM_LOWER].setChild(this.bodyPart[this.RIGHT_ARM_HAND], 100, 25);

        // left leg
        this.bodyPart[this.TORSO].setChild(this.bodyPart[this.LEFT_LEG_UPPER], 135, 190);
        this.bodyPart[this.LEFT_LEG_UPPER].setChild(this.bodyPart[this.LEFT_LEG_LOWER], 60, 90);
        this.bodyPart[this.LEFT_LEG_LOWER].setChild(this.bodyPart[this.LEFT_LEG_FOOT], 25, 70);

        // right leg
        this.bodyPart[this.TORSO].setChild(this.bodyPart[this.RIGHT_LEG_UPPER], 70, 190);
        this.bodyPart[this.RIGHT_LEG_UPPER].setChild(this.bodyPart[this.RIGHT_LEG_LOWER], 40, 90);
        this.bodyPart[this.RIGHT_LEG_LOWER].setChild(this.bodyPart[this.RIGHT_LEG_FOOT], 25, 70);
    }
```

The render() method steps through the bodyParts[] array and renders each of the Teddy bodyParts.

```
    render()
    {
        // place all of the bodyParts in the correct position
        this.bodyPart[this.TORSO].setChildrenPositions();

        // draw all of the body parts on the main game canvas
        for (let i = this.bodyPart.length - 1; i >= 0; i--)
        {
            this.bodyPart[i].render();
        }
    }
```

TeddyBodyPart.js

```
/* Author: Derek O Reilly, Dundalk Institute of Technology, Ireland. */


class TeddyBodyPart
```

```
{
    /* Each gameObject MUST have a constructor() and a render() method.        */
    /* If the object animates, then it must also have an updateState() method. */

    constructor(scale, sourceImage, width, height, radiants, centreOfRotationX, centreOfRotationY, hotSpotX, hotSpotY, h
    {
        // There needs to be one bodyPart for each seperate part of the body
        // For example, the left arm consists of 3 bodyParts: LEFT_ARM_UPPER, LEFT_ARM_LOWER and LEFT_HAND

        /* These variables depend on the object */
        this.scale = scale; // a scale of 1.0 means this whole Teddy is the size of the canvas.
        this.bodyPartImageFile = sourceImage; // the image file that contains this bodyPart

        this.width = width * scale;        // width and height of the bodyPart in canvas coordinates
        this.height = height * scale;
        this.radiants = radiants;      // the angle of rotation (in radiants)
        this.centreOfRotationX = centreOfRotationX * scale;  // the point around which rotations occur
        this.centreOfRotationY = centreOfRotationY * scale;

        this.hotSpotX = hotSpotX * scale;             // hot spot for identifying where the user can drag the mouse to
        this.hotSpotY = hotSpotY * scale;             // rotate this bodyPart
        this.hotSpotRadius = hotSpotRadius * scale;   // size of the hotspot

        this.children = [];
        this.numberOfChildren = 0;      // used to keep track of the number of children in the this.children array above

        this.hotSpotIsDrawn = false;  // by default, do not draw the hotspot on the screen

        /* this.offscreenBodyPartCanvasCtx will be used to hold the rotated bodyPart */
        /* When a bodyPart rotates, it will also cause its children to rotate */
        this.offscreenBodyPartCanvas = document.createElement('canvas');
        this.offscreenBodyPartCanvasCtx = this.offscreenBodyPartCanvas.getContext('2d');
        this.offscreenBodyPartCanvas.width = canvas.width;
        this.offscreenBodyPartCanvas.height = canvas.height;
    }

    render()
    {
        // draw rotated bodyPart onto its offscreen canvas
        this.offscreenBodyPartCanvasCtx.clearRect(0, 0, canvas.width, canvas.height);
        this.rotateOffscreenCanvas(this.radiants, this.parentX, this.parentY);
        this.offscreenBodyPartCanvasCtx.drawImage(this.bodyPartImageFile, this.parentX - this.centreOfRotationX, this.pa
        this.rotateOffscreenCanvas(-this.radiants, this.parentX, this.parentY);

        // if it is enabled, then draw the the hotspot on the offscreen canvas
        if (this.hotSpotIsDrawn === true)
        {
            this.drawHotSpotOnOffscreenCanvas();
        }

        // draw the offscreen canvas onto the main canvas
        ctx.drawImage(this.offscreenBodyPartCanvas, 0, 0, canvas.width, canvas.height);
    }

    setChildrenPositions()
    {
        // This method adjusts the postion of this bodyPart's children to take account of this bodyPart's postion.
        // To achieve the adjustment of position, this method sets the 'this.parentX', 'this.parentY' and 'this.radiant'

        // update children to account for the rotation of this bodyPart
        for (var i = 0; i < this.numberOfChildren; i++)
        {
            if (this.children[i] !== undefined)
            {
                // rotate each child to account for the rotation of this bodyPart
                this.children[i].updateRotatedParentXandY(this.parentX, this.parentY, this.radiants, this.centreOfRotati

                // recursively rotate each child bodyPart about its own centre of rotation
                this.children[i].setChildrenPositions();
            }
        }
    }

    updateRotatedParentXandY(centreOfRotationX, centreOfRotationY, radiants, offsetX, offsetY)
    {
        // rotate this bodyPart to account for rotations of this bodyPart's parent
        this.parentX = this.originalParentX;
        this.parentY = this.originalParentY;
        this.parentX -= offsetX;
        this.parentY -= offsetY;

        let newX = this.parentX * Math.cos(radiants) - this.parentY * Math.sin(radiants);
        this.parentY = this.parentX * Math.sin(radiants) + this.parentY * Math.cos(radiants);
        this.parentX = newX;
```

```
        this.parentX += centreOfRotationX;
        this.parentY += centreOfRotationY;
    }

    rotateOffscreenCanvas(radiants, centreX, centreY)
    {
        // this is a helper method for the method 'rotate()'
        // rotate the offscreen canvas of this bodyPart around this the given centre of rotation
        this.offscreenBodyPartCanvasCtx.translate(centreX, centreY);
        this.offscreenBodyPartCanvasCtx.rotate(radiants);
        this.offscreenBodyPartCanvasCtx.translate(-centreX, -centreY);
    }

    changeRotationValue(mouseX, mouseY, degrees)
    {
        // depending on the position of the mouse relative to the position of this.parentX and this.parentY, add or subt
        if (this.isRotatingClockwise(this.parentX, this.parentY, mouseX, mouseY) === true)
        {
            this.radiants += Math.radians(degrees);
        }
        else
        {
            this.radiants -= Math.radians(degrees);
        }
    }

    isRotatingClockwise(centerX, centerY, x, y)
    {
        // return true if the mouse is dragging in a clockwise direction, else return false
        if (Math.abs(x - oldMouseX) >= Math.abs(y - oldMouseY))
        {
            // mouse is being moved primarily along the x-axis
            if ((x < oldMouseX))  // mouse is moving to the left
            {
                if (y < centerY)     // mouse is above the image
                {
                    return false;
                }
                else   // mouse is below the image
                {
                    return true;
                }
            }
            else  // mouse is moving to the right
            {
                if (y < centerY)   // mouse is to the left of the image
                {
                    return true;
                }
                else  // mouse is to the right of the image
                {
                    return false;
                }
            }
        }
        else // mouse is being moved primarily along the y-axis
        {
            // mouse is being moved primarily along the y-axis
            if (y < oldMouseY)  // mouse is moving to the left
            {
                if (x < centerX)     // mouse is above the image
                {
                    return true;
                }
                else   // mouse is below the image
                {
                    return false;
                }
            }
            else  // mouse is moving to the right
            {
                if (x < centerX)   // mouse is to the left of the image
                {
                    return false;
                }
                else  // mouse is to the right of the image
                {
                    return true;
                }
            }
        }
    }

    isHotSpot(canvasX, canvasY)
    {
```

```
    // return true if the canvasX, canvasY is in this bodyPart's hotspot
    var hotSpotX = this.parentX - this.centreOfRotationX + this.hotSpotX;
    var hotSpotY = this.parentY - this.centreOfRotationY + this.hotSpotY;

    // rotate the hotspot around the parentX and parentY
    hotSpotX -= this.parentX;
    hotSpotY -= this.parentY;

    let newX = hotSpotX * Math.cos(this.radiants) - hotSpotY * Math.sin(this.radiants);
    hotSpotY = hotSpotX * Math.sin(this.radiants) + hotSpotY * Math.cos(this.radiants);
    hotSpotX = newX;

    hotSpotX += this.parentX;
    hotSpotY += this.parentY;

    if ((canvasX > hotSpotX - this.hotSpotRadius) && (canvasX < hotSpotX + this.hotSpotRadius) &&
            (canvasY > hotSpotY - this.hotSpotRadius) && (canvasY < hotSpotY + this.hotSpotRadius))
    {
        return true;
    }
    else
    {
        return false;
    }
}

drawHotSpotOnOffscreenCanvas()
{
    // draw a bodyPart's hotspot on the offscreenBodyPartCanvas
    if (this.hotSpotIsDrawn === false)
    {
        return;
    }

    var hotSpotX = this.parentX - this.centreOfRotationX + this.hotSpotX;
    var hotSpotY = this.parentY - this.centreOfRotationY + this.hotSpotY;

    // rotate the hotspot around the parentX and parentY
    hotSpotX -= this.parentX;
    hotSpotY -= this.parentY;

    let newX = hotSpotX * Math.cos(this.radiants) - hotSpotY * Math.sin(this.radiants);
    hotSpotY = hotSpotX * Math.sin(this.radiants) + hotSpotY * Math.cos(this.radiants);
    hotSpotX = newX;

    hotSpotX += this.parentX;
    hotSpotY += this.parentY;

    // draw the hotspot
    this.offscreenBodyPartCanvasCtx.globalAlpha = 0.15;
    this.offscreenBodyPartCanvasCtx.beginPath();
    this.offscreenBodyPartCanvasCtx.fillStyle = "blue";
    this.offscreenBodyPartCanvasCtx.arc(hotSpotX, hotSpotY, this.hotSpotRadius, 0, Math.PI * 2);
    this.offscreenBodyPartCanvasCtx.fill();
    this.offscreenBodyPartCanvasCtx.closePath();
    this.offscreenBodyPartCanvasCtx.globalAlpha = 1;
}

setChild(child, parentX, parentY)
{
    // set a direct child of this bodyPart
    // For example, LEFT_ARM_UPPER has one direct child, which is LEFT_ARM_LOWER
    this.children[this.numberOfChildren] = child;
    child.setInitialParentXandY(parentX * this.scale, parentY * this.scale);
    this.numberOfChildren++;
}

setInitialParentXandY(newParentX, newParentY)
{
    // this method ties the parent's coordinates to the child's centreOfRotation coordinates
    // The parent's coordinates are the x,y values of the joint on the parent where
    // the child's rotation x,y coordinates will attach to
    this.parentX = newParentX;
    this.parentY = newParentY;
    this.originalParentX = this.parentX;
    this.originalParentY = this.parentY;
}

setDrawHotSpot(state)
{
    // Not strictly necessary, but might be useful as a guide to showing younger users where to drag
    this.hotSpotIsDrawn = state;
}

getDegrees()
```

```
    {
        // return the rotation angle as degrees
        return this.radiants * 180 / Math.PI;
    }

    setRadiants(newDegrees)
    {
        // set the rotation angle as radiants
        // note that the input parameter, newDegrees, is given in degrees
        this.radiants = Math.radians(newDegrees);
    }

    setDrawHotSpot(state)
    {
        this.hotSpotIsDrawn = state;
    }
}
```

Code Explained

Teddy is made up of 14 BodyParts, which are linked together in a parent-child structure. Each BodyPart can have zero or more child BodyParts. Parents and children are linked at the child's point 'this.centreOfRotationX,this.centreOfRotationX' (as shown in red below). The parent's connecting point is provided an a parameter input to the method setInitialParentXandY(), which we shall look at further down in these notes.
Hotspots can be used to rotate BodyParts. Hopspots will be highlighted if the 'this.hotSpotIsDrawn' flag is set to true (as shown in blue below).
Each BodyPart has an offscreen canvas (as shown in green below). Rotations of a BodyPart will be implemented by rotating the offscreen canvas. The offscreen canvas is the same size as the main game canvas. This means that any rotations that occur on the offscreen canvas will be reflected in the final canvas when the offscreen canvas is drawn onto the main canvas.

```
    constructor(scale, sourceImage, width, height, radiants, centreOfRotationX, centreOfRotationY, hotSpotX, hotSpotY, h
    {
        // There needs to be one bodyPart for each seperate part of the body
        // For example, the left arm consists of 3 bodyParts: LEFT_ARM_UPPER, LEFT_ARM_LOWER and LEFT_HAND

        /* These variables depend on the object */
        this.scale = scale; // a scale of 1.0 means this whole Teddy is the size of the canvas.
        this.bodyPartImageFile = sourceImage; // the image file that contains this bodyPart

        this.width = width * scale;        // width and height of the bodyPart in canvas coordinates
        this.height = height * scale;
        this.radiants = radiants;      // the angle of rotation (in radiants)
        this.centreOfRotationX = centreOfRotationX * scale;  // the point around which rotations occur
        this.centreOfRotationY = centreOfRotationY * scale;

        this.hotSpotX = hotSpotX * scale;             // hot spot for identifying where the user can drag the mouse to
        this.hotSpotY = hotSpotY * scale;             // rotate this bodyPart
        this.hotSpotRadius = hotSpotRadius * scale;   // size of the hotspot

        this.children = [];
        this.numberOfChildren = 0;      // used to keep track of the number of children in the this.children array above

        this.hotSpotIsDrawn = false;  // by default, do not draw the hotspot on the screen

        /* this.offscreenBodyPartCanvasCtx will be used to hold the rotated bodyPart */
        /* When a bodyPart rotates, it will also cause its children to rotate */
        this.offscreenBodyPartCanvas = document.createElement('canvas');
        this.offscreenBodyPartCanvasCtx = this.offscreenBodyPartCanvas.getContext('2d');
        this.offscreenBodyPartCanvas.width = canvas.width;
        this.offscreenBodyPartCanvas.height = canvas.height;
    }
```

The render() method draws the rotated offscreen canvas on the main game canvas. It also draws the hotspot, if it is enabled.

```
    render()
    {
        // draw rotated bodyPart onto its offscreen canvas
        this.offscreenBodyPartCanvasCtx.clearRect(0, 0, canvas.width, canvas.height);
        this.rotateOffscreenCanvas(this.radiants, this.parentX, this.parentY);
        this.offscreenBodyPartCanvasCtx.drawImage(this.bodyPartImageFile, this.parentX - this.centreOfRotationX, this.pa
        this.rotateOffscreenCanvas(-this.radiants, this.parentX, this.parentY);

        // if it is enabled, then draw the the hotspot on the offscreen canvas
        if (this.hotSpotIsDrawn === true)
        {
            this.drawHotSpotOnOffscreenCanvas();
        }

        // draw the offscreen canvas onto the main canvas
        ctx.drawImage(this.offscreenBodyPartCanvas, 0, 0, canvas.width, canvas.height);
    }
```

The setChildrenPositions() method recursively adjusts all of this BodyPart's children to match this BodyPart's current position (as shown in red in the code below).

```
    setChildrenPositions()
    {
        // This method adjusts the postion of this bodyPart's children to take account of this bodyPart's postion.
        // To achieve the adjustment of position, this method sets the 'this.parentX', 'this.parentY' and 'this.radiant'

        // update children to account for the rotation of this bodyPart
        for (var i = 0; i < this.numberOfChildren; i++)
        {
            if (this.children[i] !== undefined)
            {
                // rotate each child to account for the rotation of this bodyPart
                this.children[i].updateRotatedParentXandY(this.parentX, this.parentY, this.radiants, this.centreOfRotati

                // recursively rotate each child bodyPart about its own centre of rotation
                this.children[i].setChildrenPositions();
            }
        }
    }
```

The rotateOffscreenCanvas() method rotates the offscreen canvas.

```
    rotateOffscreenCanvas(radiants, centreX, centreY)
    {
        // this is a helper method for the method 'rotate()'
        // rotate the offscreen canvas of this bodyPart around this the given centre of rotation
        this.offscreenBodyPartCanvasCtx.translate(centreX, centreY);
        this.offscreenBodyPartCanvasCtx.rotate(radiants);
        this.offscreenBodyPartCanvasCtx.translate(-centreX, -centreY);
    }
```

The changeRotationValue() method adds/subtracts 'degrees' based on the current mouse position on the canvas. The code is straightforward and does not need explaining.

```
    changeRotationValue(mouseX, mouseY, degrees)
    {
        // depending on the position of the mouse relative to the position of this.parentX and this.parentY, add or subt
        if (this.isRotatingClockwise(this.parentX, this.parentY, mouseX, mouseY) === true)
        {
            this.radiants += Math.radians(degrees);
        }
        else
        {
            this.radiants -= Math.radians(degrees);
        }
    }
```

The isRotatingClockwise() method is used to determine the direction that the mouse is being moved on the canvas. The code is straightforward and does not need explaining.

```
    isRotatingClockwise(centerX, centerY, x, y)
    {
        // return true if the mouse is dragging in a clockwise direction, else return false
        if (Math.abs(x - oldMouseX) >= Math.abs(y - oldMouseY))
        {
            // mouse is being moved primarily along the x-axis
            if ((x < oldMouseX))  // mouse is moving to the left
            {
                if (y < centerY)     // mouse is above the image
                {
                    return false;
                }
                else   // mouse is below the image
                {
                    return true;
                }
            }
            else  // mouse is moving to the right
            {
                if (y < centerY)   // mouse is to the left of the image
                {
                    return true;
                }
                else  // mouse is to the right of the image
                {
                    return false;
                }
            }
```

```
        }
        else // mouse is being moved primarily along the y-axis
        {
            // mouse is being moved primarily along the y-axis
            if (y < oldMouseY)  // mouse is moving to the left
            {
                if (x < centerX)     // mouse is above the image
                {
                    return true;
                }
                else   // mouse is below the image
                {
                    return false;
                }
            }
            else  // mouse is moving to the right
            {
                if (x < centerX)   // mouse is to the left of the image
                {
                    return false;
                }
                else  // mouse is to the right of the image
                {
                    return true;
                }
            }
        }
    }
```

The isHotSpot() method determines if an canvasX, canvasY coordinate is indide the hotspot area of a bodyPart. The location on the canvas must be rotated to account for the rotation of the offscreen canvas of this bodyPart.

```
    isHotSpot(canvasX, canvasY)
    {
        // return true if the canvasX, canvasY is in this bodyPart's hotspot
        var hotSpotX = this.parentX - this.centreOfRotationX + this.hotSpotX;
        var hotSpotY = this.parentY - this.centreOfRotationY + this.hotSpotY;

        // rotate the hotspot around the parentX and parentY
        hotSpotX -= this.parentX;
        hotSpotY -= this.parentY;

        let newX = hotSpotX * Math.cos(this.radiants) - hotSpotY * Math.sin(this.radiants);
        hotSpotY = hotSpotX * Math.sin(this.radiants) + hotSpotY * Math.cos(this.radiants);
        hotSpotX = newX;

        hotSpotX += this.parentX;
        hotSpotY += this.parentY;

        if ((canvasX > hotSpotX - this.hotSpotRadius) && (canvasX < hotSpotX + this.hotSpotRadius) &&
                (canvasY > hotSpotY - this.hotSpotRadius) && (canvasY < hotSpotY + this.hotSpotRadius))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

The setChild() method sets a bodyPart as being a child of this bodyPart. This is needed to allow us to adjust child bodyParts whenever we rotate their parent bodyPart.
The setInitialParentXandY() method connects a child back to its parent. It is called once for each child (as shown in red below)

```
    setChild(child, parentX, parentY)
    {
        // set a direct child of this bodyPart
        // For example, LEFT_ARM_UPPER has one direct child, which is LEFT_ARM_LOWER
        this.children[this.numberOfChildren] = child;
        child.setInitialParentXandY(parentX * this.scale, parentY * this.scale);
        this.numberOfChildren++;
    }

    setInitialParentXandY(newParentX, newParentY)
    {
        // this method ties the parent's coordinates to the child's centreOfRotation coordinates
        // The parent's coordinates are the x,y values of the joint on the parent where
        // the child's rotation x,y coordinates will attach to
        this.parentX = newParentX;
        this.parentY = newParentY;
        this.originalParentX = this.parentX;
```

```
        this.originalParentY = this.parentY;
    }
```

The other methods are straightforward and do not need explaining.

```
    setDrawHotSpot(state)
    {
        // Not strictly necessary, but might be useful as a guide to showing younger users where to drag
        this.hotSpotIsDrawn = state;
    }

    getDegrees()
    {
        // return the rotation angle as degrees
        return this.radiants * 180 / Math.PI;
    }

    setRadiants(newDegrees)
    {
        // set the rotation angle as radiants
        // note that the input parameter, newDegrees, is given in degrees
        this.radiants = Math.radians(newDegrees);
    }

    setDrawHotSpot(state)
    {
        this.hotSpotIsDrawn = state;
    }
```

Exercises

Extend Teddy to make a DancingTeddy, as shown here. Hint: The animation is created by randomly moving some of the DancingTeddy bodyParts[] inside the DancingTeddy updateState() method.