Contents

1 Introduction

On a basic web page, we can construct and page element called a "canvas".

It needs a bit of CSS to see what we have done as it begins as an empty transparent rectangle on the page

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head> <meta charset="UTF-8"> <title>JavaScript</title>
4      <style>
5          body{ background-color: #aaaaaa; }
6          canvas{
7              background-color: #ffffff;
8              width:800px;
9              height:600px;}
10     </style></head>
11 <body>
12 <canvas id='myCanvas'></canvas>
13 </body>
14 </html>
```
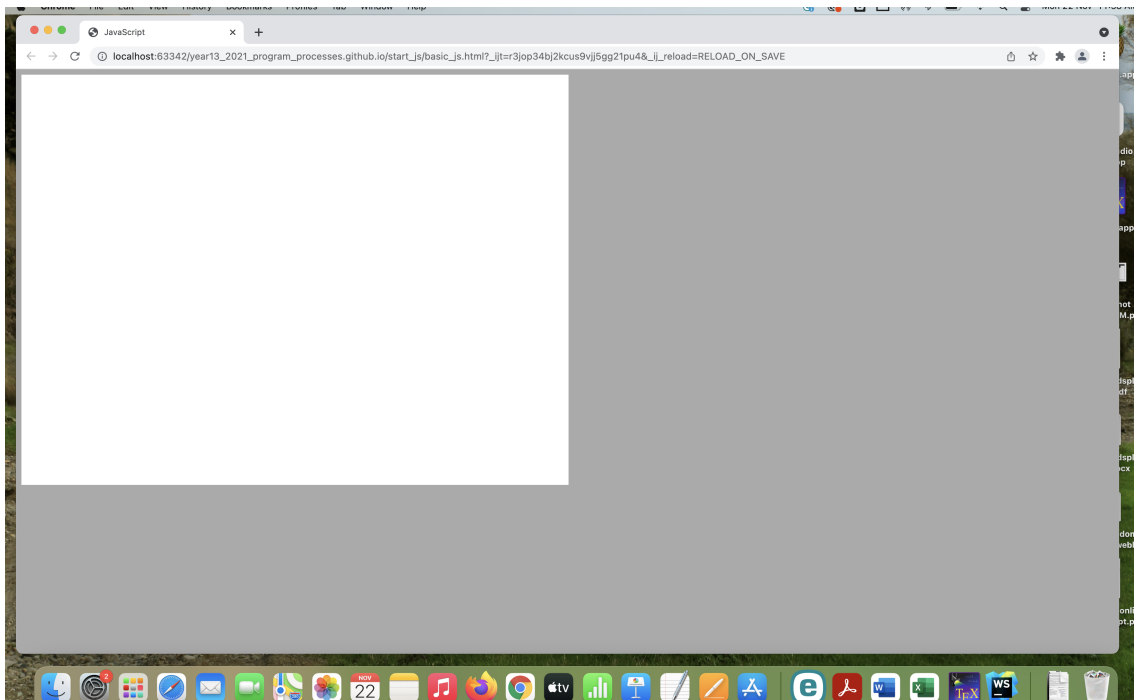
Listing 1: Basic Canvas

Figure 1: Canvas in browser

The canvas environment allows us to draw shapes and also run animations, so with a bit of work, we are able to create a small web applications such as games or other more interactively rich experiences. Our aim is to learn about this environment and to create a small drawing/painting program.

2 Getting started

To be able to do thing to the canvas we use JavaScript. This is a programming language and in general has many similarities to Python but has some differences in syntax.

JavaScript can be written in a web page within the $<$ script $>$ tag.

It can also be loaded from an external file (which we will do for most of the unit). We will have to learn quite a few things about JavaScript as we go through this unit, but regarding drawing shapes, it will always follow the same basic set of steps.

- begin a path (i.e the shape)

- define the path with its associated parameters (x,y , width, height, ..)

- define the fill, stroke and line width (do not need all)

- tell the canvas to fill and/or stroke the path

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head> <meta charset="UTF-8"> <title>JavaScript</title>
4      <style>
5          body{ background-color: #aaaaaa; }
6          canvas{
7              background-color: #ffffff;
8              width:800px;
9              height:600px;}
10     </style></head>
11 <body>
12 <canvas id='myCanvas'></canvas>
13 <script>
14     // get the canvas element using the id name
15     canvas = document.querySelector('#myCanvas');
16     // define a 2d context and associate it with the variable ctx
17     // all canvas commands will require the ctx.
18     let ctx = canvas.getContext('2d');
19     // specify height and width of canvas (which should be identical to the CSS
20     let width = 800;
21     let height = 600;
22     canvas.width = width;
23     canvas.height = height;
```

```
24
25    // draw some things
26    // rectangle
27    // start the path then define as a rectangle with parameters
28    ctx.beginPath();
29    ctx.rect(10,10,100,100);
30    // set the context
31    ctx.fillStyle='rgb(0,153,204)';
32    ctx.strokeStyle='rgb(0,0,0)';
33    ctx.lineWidth=10;
34    // actually fill and stroke
35    ctx.stroke();
36    ctx.fill();
37    // the following follow exactly the same pattern
38    // circle
39    ctx.beginPath();
40    ctx.arc(200,60, 50, 0, 2*Math.PI);
41    ctx.fillStyle='rgb(255,204,51)';
42    ctx.strokeStyle='rgb(51,51,255)';
43    ctx.lineWidth=10;
44    ctx.stroke();
45    ctx.fill();
46    // line
47    ctx.beginPath();
48    // set start point of the line
49    ctx.moveTo(0, 200);
50    // set next point of the line
51    ctx.lineTo(750,200);
52    ctx.strokeStyle="rgb(255,0,0)";
53    ctx.lineWidth=1;
54    ctx.stroke();
55
56    // rectangle with a gradient fill
57
58    ctx.beginPath()
59    ctx.rect(10,350, 200,200);
60    let my_gradient=ctx.createLinearGradient(10,350,10,550);
61    my_gradient.addColorStop(0,"rgb(255,102,102)");
62    my_gradient.addColorStop(0.5,"rgb(255,255,153)");
63    my_gradient.addColorStop(1,"rgb(0,153,204)");
64    ctx.fillStyle=my_gradient;
65    ctx.fill();
66    // note that the stroke picks up the previous context
67    ctx.stroke();
68
69    // quadratic curves (bezier)
70    ctx.strokeStyle="rgb(255,0,0)";
```

4

```
71    ctx.beginPath();
72    ctx.moveTo(300,400);
73    ctx.lineWidth=10;
74    ctx.quadraticCurveTo(500, 550, 700, 400);
75    ctx.lineCap = "round";
76    ctx.stroke();
77
78
79    // add text, set the context then fill the text
80    ctx.fillStyle="rgb(0,0,255)";
81    // shorthand css to set basic options
82    let myFont= "bold 30px monospace";
83    ctx.font=myFont;
84    ctx.fillText("Hello World", 300,50);
85
86    // images can be placed on the canvas but we need to know if they have loaded
87    // there other ways of dealing with this
88    let img = new Image();
89    img.onload = function(e){
90        let img_h = img.height;
91        let img_w= img.width
92        console.log(e);
93        ctx.drawImage(img, 500,220, img_w/4, img_h/4);
94    }
95    img.src= "red_kangaroo.jpeg"
96 </script>
97 </body>
98 </html>
```

Listing 2: Basic Shapes

This just gives a "general idea" and more complete references:

https://www.w3schools.com/html/html5_canvas.asp

https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

3   JavaScript in an external file

It will become quite cumbersome having the JavaScript in the same file as the HTML page.

So it is better to have it as a separate file that is linked in.

This may make some of the code easier to re-use, as well.

```
1 <canvas id='myCanvas'></canvas>
2 <script type="text/javascript" src="basic.js"></script>
```

Listing 3: Loading javascript

4 Improving the set up

It is also helpful to add to the code that connects to that initialises the canvas.

The code below can be discussed further, but it allows us the manage the size and basic styling of the canvas from within the JavaScript.

```javascript
canvas = document.querySelector('#myCanvas');
let ctx = canvas.getContext('2d');
// define width and height
let width = 1000;
let height = 500;
// define scale of 1. This may be changed later to improve resolution
let scale = 2;
// set the canvas width and height
canvas.width = width*scale;
canvas.height = height*scale;
// scale the canvas
ctx.scale(scale,scale);
// get the canvas element
// style it here so it will be consistent
let my_c = document.getElementById('myCanvas');
my_c.style.backgroundColor = "rgb(100,100,100)"
my_c.style.width = width+"px";
my_c.style.height = height+"px";
my_c.style.border = "6px solid rgba(200,200,200,0.5)";
my_c.style.display = "block";
my_c.style.margin = "auto";
document.body.style.backgroundColor = "rgb(190,190,190)";
```

Listing 4: Initial JS

5   Functions

We can immediately see that even drawing a small number of shapes starts tp build up code, so we want to look at ways of reducing this and having as much as possible available for "re-use". Let's look at designing a rectangle function and see what we can do with it.

Picture of design

```
1  /**
2   * Draw a rectangle
3   *
4   * @param {number} x corner x
5   * @param {number} y corner y
6   * @param {number} w width
7   * @param {number} h height
8   * @param {string} fillColour rgb string
9   * @param {string} strokeColour rgb string.
10  * @param {number} strokeWidth x coordinate of second point.
11  * @return {null}
12  */
13 function drawRect(x,y,w,h, fillColour, strokeColour, strokeWidth){
14     ctx.fillStyle = fillColour;
15     ctx.strokeStyle = strokeColour;
16     ctx.lineWidth = strokeWidth;
17     ctx.beginPath()
18     ctx.rect(x,y,w,h)
19     ctx.fill();
20     ctx.stroke();
```

```
21 }
22
23 drawRect(20,20,200,130, "rgb(232,109,135)", "rgb(236,198,76)", 2);
24 drawRect(20,170,200,130, "rgba(234,225,144,0.83)", "rgb(80,0,80)", 4);
25 drawRect(20,320,200,130, "rgba(239,89,217,0.83)", "rgb(46,169,239)", 4);
26 drawRect(240,20,700,430, "rgba(17,96,239,0.83)", "rgb(46,169,239)", 4);
27
28 //loop
29 for(let i = 0; i<26; i++){
30     drawRect(260+25*i,40,20,20, "rgba(170,193,238,0.83)", "rgb(46,169,239)", 1);
31 }
32 // double loop for a grid
33 for(let i=0; i<5 ; i++){
34     for(let j = 0; j<5; j++){
35         drawRect(260+25*i,100+25*j,20,20, "rgba(170,193,238,0.83)", "rgb
    (46,169,239)", 1);
36     }
37 }
```
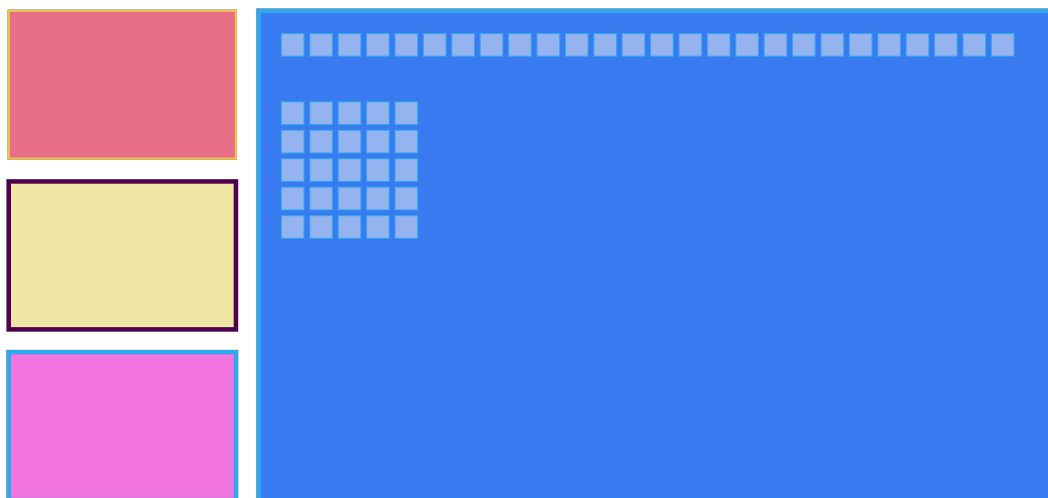
Listing 5: Python example

Figure 2: Canvas in browser

Design Functions for:

- Circle

- Line
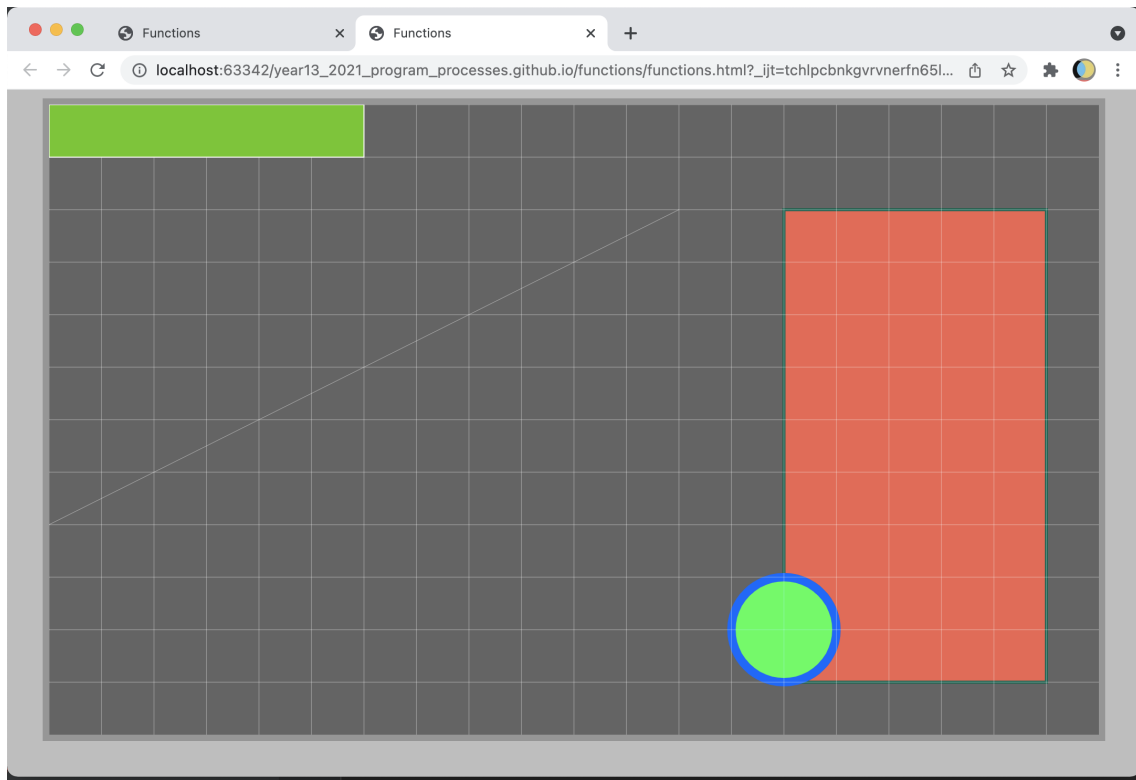
- Triangle

- Square

- A Grid

8

- Text Box

Figure 3: Canvas in browser

6 Making a grid

7 Update Context Function

We want to reduce code repetition as much as much as possible.

It might be an idea to have a small function to manage the fill and stroke commands.// This 'update context' function can sit near the top of the code and then be called from each of the shape drawing functions.

```javascript
function updateContext(fillColour, strokeColour, strokeWidth){
    ctx.fillStyle = fillColour;
    ctx.strokeStyle = strokeColour;
    ctx.lineWidth = strokeWidth;
    if(fillColour){
        ctx.fill();
    }
    if(strokeColour|| strokeWidth){
        ctx.stroke();
    }
}
/**
 * Draw a rectangle
 *
 * @param {number} x corner x
 * @param {number} y corner y
 * @param {number} w width
 * @param {number} h height
 * @param {string} fillColour rgb string
 * @param {string} strokeColour rgb string.
 * @param {number} strokeWidth
 * @return {null}
 */
function drawRectangle(x,y,w,h, fillColour, strokeColour, strokeWidth){
    ctx.beginPath();
    ctx.rect(x,y,w,h);
    updateContext(fillColour, strokeColour, strokeWidth)
}
```

Listing 6: Loading javascript

8 Colour management

Rather than writing the rgb strings all of the time, it might be helpful to start a little colour management data structure.

The structure below is actually a simple JavaScript object. But at this stage we can treat much like a Python dictionary.

It has a keyword and an associated value.

```
1  // two dimensional array of colours
2  const col= [
3  [ // opaque
4  // black (0)                grey (1)                white (2)
5  "rgba(0,0,0,1)" , "rgba(150,150,150,1)", "rgba(255,255,255,1)" ,
6  // pink  (3)          purple (4)        deep blue (5)
7  "rgb(243,92,155,1)", "rgb(153,19,206,1)", "rgb(16,16,162,1)",
8  // pale blue (6)            yellow  (7)         bright yellow (7)
9  "rgba(135,211,243,1)", "rgba(246,244,193,1)", "rgba(250,250,0,1)"
10 ],
11 [ // semi-transparent
12 // black (0)                grey (1)                white (2)
13 "rgba(0,0,0,0.5)" , "rgba(150,150,150,0.5)", "rgba(255,255,255,0.5)" ,
14 // pink  (3)          purple (4)        deep blue (5)
15 "rgb(243,92,155,0.5)", "rgb(153,19,206,0.5)", "rgb(16,16,162,0.5)",
16 // pale blue (6)            yellow  (7)         bright yellow (7)
17 "rgba(135,211,243,1,0.5)", "rgba(246,244,193,1,0.5)", "rgba(250,250,0,0.5)"
18 ]
19     ]
```

We can see its use in the function calls below

```
1   * Draw a rectangle
2   *
3   * @param {number} x corner x
4   * @param {number} y corner y
```

9   Organising an init file

We probably want to use the functions in various projects (subject to modifications).

We also have the set up code.

It might be a good idea to assemble this into a separate "initialisation" file (aka init.js). We could then have a separate file that does the "doing" of the program. We can load the files separately into the html document.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Functions Collection</title>
</head>
<body>
<canvas id='myCanvas'></canvas>
<script type="text/javascript" src="init.js"> </script>
<script type="text/javascript" src="interface.js"> </script>
</body>
</html>
```

Listing 7: Loading javascript

```javascript
drawRectangle(20,20,200,130, col[0][3], undefined, undefined);
drawRectangle(20,170,200,130, undefined, col[1][6], 9);
drawRectangle(20,320,200,130, col[0][3], col[0][6], 4);
drawRectangle(240,20,700,430,undefined,col[0][6], 4);

drawCircle(590, 235, 100, col[0][7], col[1][0], 10)

drawLine(240,450, 940,20, col[0][2], 1);
drawLine(240,20, 940,450, col[0][2], 1);
let x= 50
let space = 300
drawTextBox(x, 480, 260,"Button One", col[0][2],col[0][5]);
drawTextBox(x+space, 480, 260,"Button Two", col[0][2],col[0][5]);
drawTextBox(x+2*space, 480, 260,"Button Three", col[0][2],col[0][5]);
```

Listing 8: Loading javascript

```javascript
canvas = document.querySelector('#myCanvas');
let ctx = canvas.getContext('2d');
// define width and height
let width = 1000;
let height = 600;
// define scale of 1. This may be changed later to improve resolution
```

```
7  let scale = 2;
8  // set the canvas width and height
9  canvas.width = width*scale;
10 canvas.height = height*scale;
11 // scale the canvas
12 ctx.scale(scale,scale);
13 // get the canvas element
14 // style it here so it will be consistent
15 let my_c = document.getElementById('myCanvas');
16 my_c.style.backgroundColor = "rgb(100,100,100)"
17 my_c.style.width = width+"px";
18 my_c.style.height = height+"px";
19 my_c.style.border = "6px solid rgba(200,200,200,0.5)";
20 my_c.style.display = "block";
21 my_c.style.margin = "auto";
22 document.body.style.backgroundColor = "rgb(190,190,190)";
23
24
25 // two dimensional array of colours
26 const col= [
27     [ // opaque
28 // black (0)                grey (1)                white (2)
29        "rgba(0,0,0,1)" , "rgba(150,150,150,1)", "rgba(255,255,255,1)" ,
30 // pink (3)             purple (4)        deep blue (5)
31        "rgb(243,92,155,1)", "rgb(153,19,206,1)", "rgb(16,16,162,1)",
32 // pale blue (6)            yellow  (7)         bright yellow (7)
33        "rgba(135,211,243,1)", "rgba(246,244,193,1)", "rgba(250,250,0,1)"
34     ],
35     [ // semi-transparent
36 // black (0)                grey (1)                white (2)
37        "rgba(0,0,0,0.5)" , "rgba(150,150,150,0.5)", "rgba(255,255,255,0.5)" ,
38 // pink (3)             purple (4)        deep blue (5)
39        "rgb(243,92,155,0.5)", "rgb(153,19,206,0.5)", "rgb(16,16,162,0.5)",
40 // pale blue (6)            yellow  (7)         bright yellow (7)
41        "rgba(135,211,243,0.5)", "rgba(246,244,193,0.5)", "rgba(250,250,0,0.5)"
42     ]
43 ]
44 /**
45  * Fill and or Stroke the Current Path
46  *
47  * @param {string} fillColour rgb string
48  * @param {string} strokeColour rgb string.
49  * @param {number} strokeWidth
50  * @return {null}
51  */
52 function updateContext(fillColour, strokeColour, strokeWidth){
53     ctx.fillStyle = fillColour;
```

```javascript
        ctx.strokeStyle = strokeColour;
        ctx.lineWidth = strokeWidth;
        if(fillColour){
            ctx.fill();
        }
        if(strokeColour|| strokeWidth){
            ctx.stroke();
        }
}
/**
 * Draw a rectangle
 *
 * @param {number} x corner x
 * @param {number} y corner y
 * @param {number} w width
 * @param {number} h height
 * @param {string} fillColour rgb string
 * @param {string} strokeColour rgb string.
 * @param {number} strokeWidth
 * @return {null}
 */
function drawRectangle(x,y,w,h, fillColour, strokeColour, strokeWidth){
    ctx.beginPath();
    ctx.rect(x,y,w,h);
    updateContext(fillColour, strokeColour, strokeWidth)
}
/**
 * Draw a circle
 *
 * @param {number} x centre x
 * @param {number} y centre y
 * @param {number} r radius
 * @param {string} fillColour rgb string
 * @param {string} strokeColour rgb string.
 * @param {number} strokeWidth
 * @return {null}
 */
function drawCircle(x,y,r,  fillColour, strokeColour, strokeWidth){
    ctx.beginPath();
    ctx.arc(x,y,r,0,2*Math.PI);
    updateContext(fillColour, strokeColour, strokeWidth)
}
/**
 * Draw a line
 *
    * @param {number} x_1 start x
    * @param {number} y_1 start y
```

```
101      * @param {number} x_2 end x
102      * @param {number} y_2 end y
103      * @param {string} strokeColour rgb string.
104      * @param {number} strokeWidth
105      * @return {null}
106  */
107 function drawLine(x_1,y_1, x_2, y_2, strokeColour,strokeWidth){
108     ctx.beginPath();
109     ctx.moveTo(x_1,y_1);
110     ctx.lineTo(x_2,y_2);
111     ctx.lineCap = "round";
112     updateContext(undefined, strokeColour, strokeWidth);
113 }
114 /**
115  * Draw text
116      *
117      * @param {number} x top corner x
118      * @param {number} y top corner y
119      * @param {string} txt
120      * @param {string} fillColour rgb string.
121      * @param {string} font css shorthand font style
122      * @return {null}
123  */
124 function drawText(txt ,x,y,fillColour, font = "bold 30px monospace" ) {
125     ctx.font = font;
126     ctx.fillStyle = fillColour;
127     ctx.fillText(txt, x,y);
128 }
129 /**
130  * Draw text box
131      *
132      * @param {number} x top corner x
133      * @param {number} y top corner y
134      * @param {number} w width
135      * @param {string} txt
136      * @param {string} backColour rgb string.
137      * @param {string} fillColour rgb string.
138      * @param {string} font css shorthand font style
139      * @return {null}
140  */
141 function drawTextBox(x,y,w,txt,backColour, fillColour, font = "bold 30px
        monospace"){
142     let h = 50;
143     drawRectangle(x,y,w,h, backColour, undefined, undefined)
144
145     drawLine(x,y+h/2,x+w,y+h/2, col[0][4],1);
146     ctx.textAlign = "center";
```
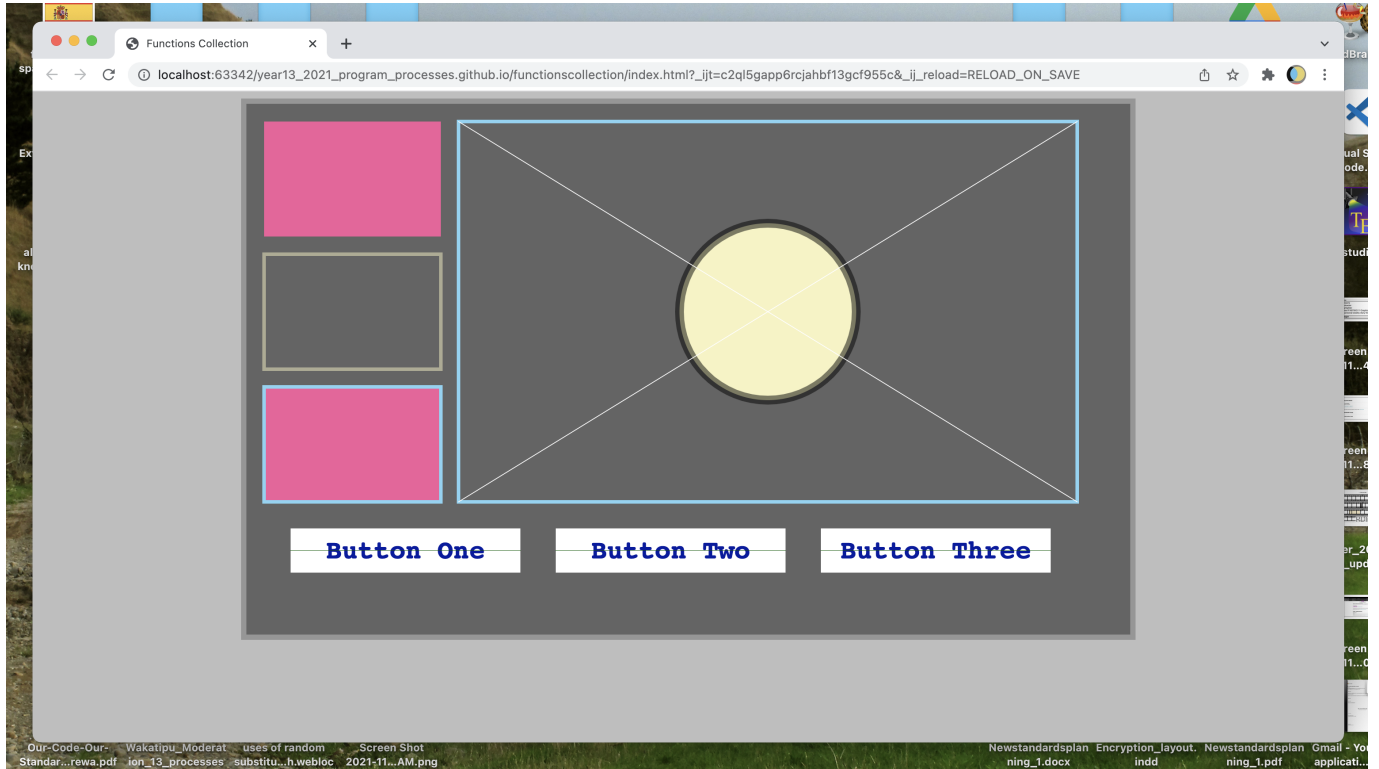
```
147    ctx.textBaseline = "middle";
148    drawText(txt, x+w/2,y+h/2, fillColour);
149 }
```

Listing 9: Loading javascript

10 Rotations

11 Objects

We are intending to introduce animation into our canvas environment.

To do so we will need to create objects.

At present, it will appear to serve no new purpose, so please be patient. Just about everything we make will be object based from now on.

Diagram

```
1  // init file with canvas set up
2  // colours
3  // functions
4  //...
```

Listing 10: Init file (not shown)

```
1  /**
2   * Filled Ball
3   * @param {number} x top corner of bounding box
4   * @param {number} y top corner of bounding box
5   * @param {string} fill fill colour
6   * @param {string} stroke stroke colour
7   * @param {number} strokeWidth  width of outline
8   */
9  class Ball{
10     constructor(x,y,r,fill, stroke, strokeWidth){
11         this.x = x;
12         this.y = y;
13         this.r=r;
14         this.fill=fill;
15         this.stroke = stroke;
16         this.strokeWidth = strokeWidth;
17     }
18     update(){
19         this.draw()
20     }
21     draw(){
22         this.drawCircle(this.x, this.y, this.r, this.fill, this.stroke, this.
    strokeWidth)
23     }
24 }
25 Ball.prototype.drawCircle = drawCircle
26 Ball.prototype.updateContext = updateContext
27
28 /**
29  * Filled Rectangle
30  * @param {number} x top corner of bounding box
```

19

```
31   * @param {number} y top corner of bounding box
32   * @param {number} w width
33   * @param {number} h height
34   * @param {string} fill fill colour
35   * @param {string} stroke stroke colour
36   * @param {number} strokeWidth  width of outline
37   */
38  class Rectangle{
39      constructor(x,y,w,h,fill, stroke, strokeWidth){
40          this.x = x;
41          this.y = y;
42          this.w=w;
43          this.h=h;
44          this.fill=fill;
45          this.stroke = stroke;
46          this.strokeWidth = strokeWidth;
47      }
48      update(){
49          this.draw()
50      }
51      draw(){
52          this.drawRectangle(this.x, this.y, this.w, this.h, this.fill, this.stroke
    , this.strokeWidth)
53      }
54  }
55  Rectangle.prototype.drawRectangle = drawRectangle
56  Rectangle.prototype.updateContext = updateContext
57  /**
58   * Filled TextBox
59   * @param {number} x top corner of bounding box
60   * @param {number} y top corner of bounding box
61   * @param {number} w width
62   * @param {string} txt text
63   * @param {string} fill fill colour
64   * @param {string} txtColour colour of text
65   */
66  class TextBox{
67      constructor(x,y,width,txt, fillColour, txtColour) {
68          this.x = x;
69          this.y = y;
70          this.w = width;
71          this.txt = txt;
72          this.fillColour = fillColour;
73          this.txtColour = txtColour;
74      }
75      update(txt){
76          this.txt = txt
```

```
77          this.draw()
78      }
79
80      draw(){
81          this.drawTextBox(this.x, this.y,this.w,this.txt, this.fillColour, this.
    txtColour)
82
83      }
84 }
85 TextBox.prototype.drawTextBox = drawTextBox;
86
87
88 /**
89  * Grid - square grid
90  * @param {number} w width of canvas
91  * @param {number} h height of canvas
92  * @param {number} intervalWidth height of canvas
93  * @param {string} strokeColour stroke colour
94  * @param {number} strokeWidth  width of outline
95  */
96 class Grid{
97      constructor(w,h,intervalWidth, strokeColour, strokeWidth){
98          this.w = w;
99          this.h = h;
100         this.intervalWidth=intervalWidth;
101         this.strokeColour = strokeColour;
102         this.strokeWidth = strokeWidth;
103     }
104     update(){
105         this.draw()
106     }
107     draw(){
108
109         for(let i = -this.w ; i <= this.w ; i+= this.intervalWidth){
110
111             this.drawLine(i,-this.h, i,this.h, this.strokeColour, this.
    strokeWidth);
112         }
113         for(let j = -this.h ; j <= this.h ; j+= this.intervalWidth){
114             this.drawLine(-this.w,j, this.w,j, this.strokeColour, this.
    strokeWidth);
115         }
116         this.drawCircle(0,0,20,undefined,col[0][4], 5);
117         this.drawLine(-this.w,0, this.w, 0, col[0][4], 4);
118         this.drawLine(0, -this.h, 0, this.h, col[0][4], 4);
119     }
120 }
```

```
121  Grid.prototype.drawCircle = drawCircle
122  Grid.prototype.drawLine = drawLine
123  Grid.prototype.updateContext = updateContext
```

Listing 11: objectSet file

```
1  <canvas id='myCanvas'></canvas>
2  <script type="text/javascript" src="init.js"> </script>
3  <script type="text/javascript" src="objectSet.js"> </script>
4  <script>
5
6
7
8      let G= new Grid(width, height, 100, col[0][2], 0.5);
9      let R = new Rectangle(-50,-25,100,50, col[0][6], undefined, undefined)
10
11     G.update();
12     ctx.save();
13     ctx.translate(500,300);
14     ctx.rotate(60*Math.PI/180);
15     //G.update();
16     R.update();
17     ctx.restore()
18
19     ballSet = []
20     let r = 20
21     let x = 100
22     let y = 400
23     let space = 5*r
24     for(let i= 0; i< col[0].length; i++){
25         let temp = new Ball(x+ space*i, y, r , col[0][i], col[0][5], 2)
26         ballSet.push(temp)
27
28     }
29     console.log(ballSet)
30     for(let j=0; j<ballSet.length; j++){
31         ballSet[j].update()
32     }
33
34     let T = new TextBox(100,100,150, "", col[0][1], col[0][2])
35     T.update("5")
```

Listing 12: Implementing (using HTML file)

12 Animation Frame

12.1 A bit of a code review - preparation

.

You do not need to have exactly this.

```
1  // init file with canvas set up
2  // colours
3  // functions
4  //...
5  canvas = document.querySelector('#myCanvas');
6  let ctx = canvas.getContext('2d');
7  // define width and height
8  let width = 1000;
9  let height = 600;
10 // define scale of 1. This may be changed later to improve resolution
11 let scale = 2;
12 // set the canvas width and height
13 canvas.width = width*scale;
14 canvas.height = height*scale;
15 // scale the canvas
16 ctx.scale(scale,scale);
17 // get the canvas element
18 // style it here so it will be consistent
19 let my_c = document.getElementById('myCanvas');
20 my_c.style.backgroundColor = "rgb(100,100,100)"
21 my_c.style.width = width+"px";
22 my_c.style.height = height+"px";
23 my_c.style.border = "6px solid rgba(200,200,200,0.5)";
24 my_c.style.display = "block";
25 my_c.style.margin = "auto";
26 document.body.style.backgroundColor = "rgb(190,190,190)";
27
28
29 // two dimensional array of colours
30 const col= [
31     [ // opaque
32 // black (0)              grey (1)              white (2)
33         "rgba(0,0,0,1)" , "rgba(150,150,150,1)", "rgba(255,255,255,1)" ,
34 // pink  (3)          purple (4)       deep blue (5)
35         "rgb(243,92,155,1)", "rgb(153,19,206,1)", "rgb(16,16,162,1)",
36 // pale blue (6)          yellow  (7)          bright yellow (7)
37         "rgba(135,211,243,1)", "rgba(246,244,193,1)", "rgba(250,250,0,1)"
38     ],
39     [ // semi-transparent
```

```
40 // black (0)                    grey (1)                    white (2)
41       "rgba(0,0,0,0.5)" , "rgba(150,150,150,0.5)", "rgba(255,255,255,0.5)" ,
42 // pink (3)              purple (4)         deep blue (5)
43       "rgb(243,92,155,0.5)", "rgb(153,19,206,0.5)", "rgb(16,16,162,0.5)",
44 // pale blue (6)              yellow  (7)         bright yellow (7)
45       "rgba(135,211,243,0.5)", "rgba(246,244,193,0.5)", "rgba(250,250,0,0.5)"
46     ]
47 ]
```

Listing 13: init

It is good to have a grid object and a textbox object.

Let's also start doc typing.

```
1  /**
2   * Grid - draws a square grid of given interval width
3   * across the whole canvas
4   * @param {number} w width of canvas
5   * @param {number} h height of canvas
6   * @param {number} intervalWidth distance each grid unit
7   * @param {string} strokeColour stroke colour
8   * @param {number} strokeWidth  width of outline
9   */
10 class Grid{
11    constructor(w,h,intervalWidth, strokeColour, strokeWidth){
12        this.w = w;
13        this.h = h;
14        this.intervalWidth=intervalWidth;
15        this.strokeColour = strokeColour;
16        this.strokeWidth = strokeWidth;
17    }
18    update(){
19        this.draw()
20    }
21    draw(){
22        // these loops also draw the grid outside (as is useful when analysing
    translations
23        // and rotations (so your can ignore the negatives and use 0 instead
24        // a loop for the vertical lines
25        for(let i = -this.w ; i <= this.w ; i+= this.intervalWidth){
26            this.drawLine(i,-this.h, i,this.h, this.strokeColour, this.
    strokeWidth);
27        }
28        // a loop for the horizontals
29        for(let j = -this.h ; j <= this.h ; j+= this.intervalWidth){
30            this.drawLine(-this.w,j, this.w,j, this.strokeColour, this.
    strokeWidth);
31        }
```

```
32        }
33        drawLine(x_1,y_1, x_2, y_2, strokeColour,strokeWidth){
34            ctx.beginPath();
35            ctx.moveTo(x_1,y_1);
36            ctx.lineTo(x_2,y_2);
37            ctx.lineCap = "round";
38            ctx.strokeStyle = strokeColour;
39            ctx.lineWidth = strokeWidth;
40            ctx .stroke()
41        }
42 }
43
44 /**
45  * A little textbox (text on coloured rectangle)
46  * @param {number} x top corner of bounding box
47  * @param {number} y top corner of bounding box
48  * @param {number} w width
49  * @param {string} txt text
50  * @param {string} fill fill colour
51  * @param {string} txtColour colour of text
52  */
53 class TextBox{
54     constructor(x,y,width, fillColour, txtColour) {
55            this.x = x;
56            this.y = y;
57            this.w = width;
58            // fixed height
59            this.h = 50;
60            // text managed through update
61            this.txt = "Placeholder";
62            console.log(this.txt)
63            this.fillColour = fillColour;
64            this.txtColour = txtColour;
65        }
66     // the text can be changed using the update function
67     update(txt ="Placeholder"){
68            this.txt = txt
69            this.draw()
70        }
71     draw(){
72            ctx.beginPath();
73            ctx.rect(this.x,this.y,this.w,this.h);
74            ctx.fillStyle= this.fillColour;
75            ctx.fill();
76            ctx.font = "20px monospace";
77            ctx.textAlign = "center";
78            ctx.textBaseline = "middle";
```

```
79        ctx.fillStyle = this.txtColour;
80        ctx.fillText(this.txt, this.x+this.w/2, this.y+this.h/2);
81    }
82 }
```

Listing 14: objects

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <meta http-equiv="X-UA-Compatible" content="ie=edge">
7    <title>Objects Start</title>
8  </head>
9  <body>
10 <canvas id='myCanvas'></canvas>
11 <script type="text/javascript" src="init.js"> </script>
12 <script type="text/javascript" src="objects.js"> </script>
13 <script>
14   let G = new Grid(width, height, 25, col[0][2], 0.3)
15   let T = new TextBox(800,550,200, col[0][4], col[0][2])
16   G.update();
17   // note that the text is set through the update function call
18   T.update("My Text");
19
20 </script>
21 </body>
22 </html>
```
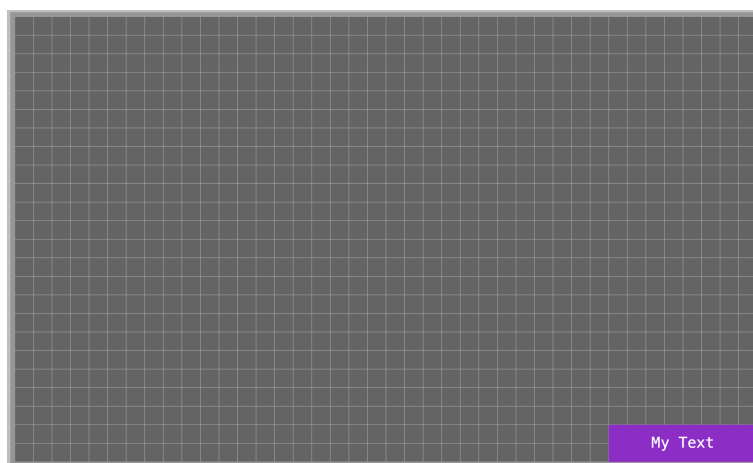
Listing 15: index (note text being set using update function call)

12.2 Introduce the animation frame

We can create a function, animate.

At the end of, and in, the function we have the 'window.requestAnimationFrame()', in the parameter you put the 'animate' function.

This means that, at the browser's discretion the function will be called over and over again. This should happen somewhere between 50 - 60 times persecond.

We don't give animate any parameter, but a parameter is sent (let's call it 't') and can be used.

In this case it is the number of milliseconds since the animation began.

Inside the animate function , the first thing we do is erase the canvas and then redraw what we want after that.

This will give us 'frame-by-frame' animation.

```
<script>
  let G = new Grid(width, height, 25, col[0][2], 0.3)
  let T = new TextBox(800,550,200, col[0][4], col[0][2])
  // create an animation function
  function animate(t){
    ctx.clearRect(0,0, width, height);
    G.update();
    let timer = Math.round(t)
    T.update(timer);
    // the call below is a request to the browser
    // the function is called again (about 50 times a second)
    window.requestAnimationFrame(animate)
  }
  // start off call to get it going
  animate()
</script>
```

Listing 16: index

13   Animation

13.1   Linear Interpolation

Consider the graph below.

The graph runs of a interval of T and moves between a maximum height of H and a minimum of 0.

We could use this to represent a, very simple, motion of a ball going up and down.

If we keep repeating the interval, the ball would go up and down indefinitely.

The lower case t represents the 'time ticks' and is like the x value.

The equations for this piecewise graph are given below and you should be able to work these out for yourself.

The graph has been drawn upside down, so it is like the canvas co-ordinate system
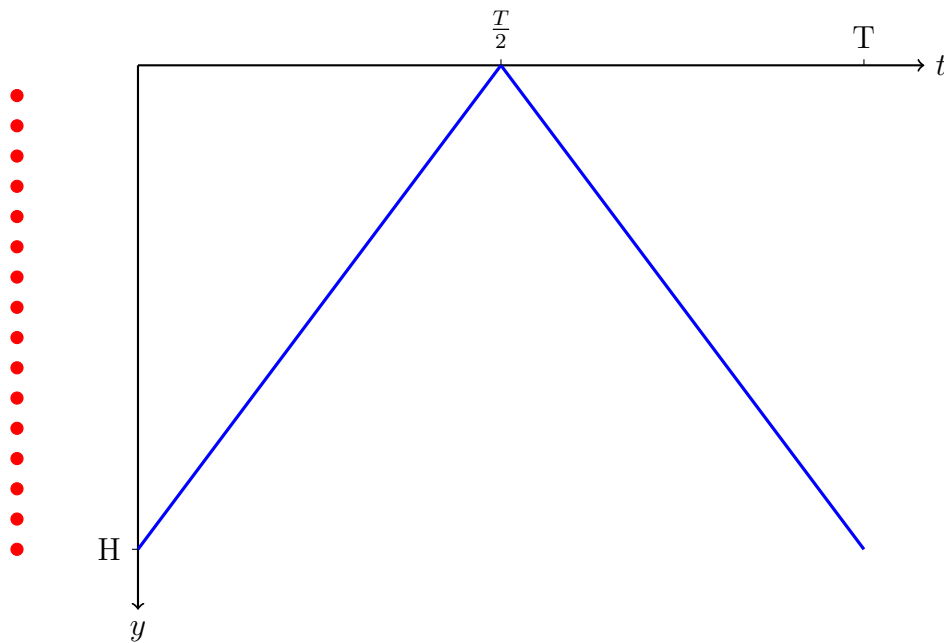
$$
\begin{cases}
y = \dfrac{-2Ht}{T} + H \quad , \quad 0 < t \leq \dfrac{T}{2} \\[3mm]
y = \dfrac{2Ht}{T} - H \quad , \quad \dfrac{T}{2} < x \leq T
\end{cases}
$$

We have an animation frame that runs at somewhere between 40 and 60 time ticks per second.

So we should be able to implement the functions given about and update the t value, for every tick of the animation frame.

Moving Ball Object

```
1  /**
2   * Ball that moves up down
3   * @param {number} x_b base x position
4   * @param {number} y_b base y position
5   * @param {number} radius radius
6   * @param {string} fillColour fill colour
7   * @param {number} T total Tick interval (50 ticks = about 1 second)
8   * @param {number} H total Height covered by up/down motion
9   */
10 class MovingBall{
11     constructor(x_b,y_b,r, fillcolour, T, H){
12         this.x_b = x_b;
13         this.y_b = y_b;
14         this.r = r;
15         this.fillColour = fillcolour;
16         // animation variables
17         this.t = 0;
18         this.T = T;
19         this.H = H;
20     }
21     update(){
22         // add one to the value of little t each time update is called
23         this.t +=1
24         this.draw()
25     }
26     draw(){
27         // get y value from the piecewise function
28         let y = this.linearinterpolate(this.t, this.T, this.H)
29         this.drawCircle(this.x_b,y+this.y_b, this.r)
30     }
31
32     linearinterpolate(t,T,H){
33         // takes parameter t , T, H
34         // we could hard code in this.T etc but is more fexible to have
   parameters
35         // make sure t is between 0 and T
36         t = t%T;  // modulus operator
37         // set y variable and use to get value from equations
38         let y;
39         if(t<T/2){
40             y = (-2*H*t)/(T) + H
41         }else{
42             y = (2*H*t)/(T) - H
43         }
44         return y
```

```
45      }
46      drawCircle(x,y,r){
47          ctx.beginPath()
48          ctx.arc(x, y, r, 0, 2*Math.PI)
49          ctx.fillStyle = this.fillColour
50          ctx.fill();
51      }
```

Listing 17: Moving Ball
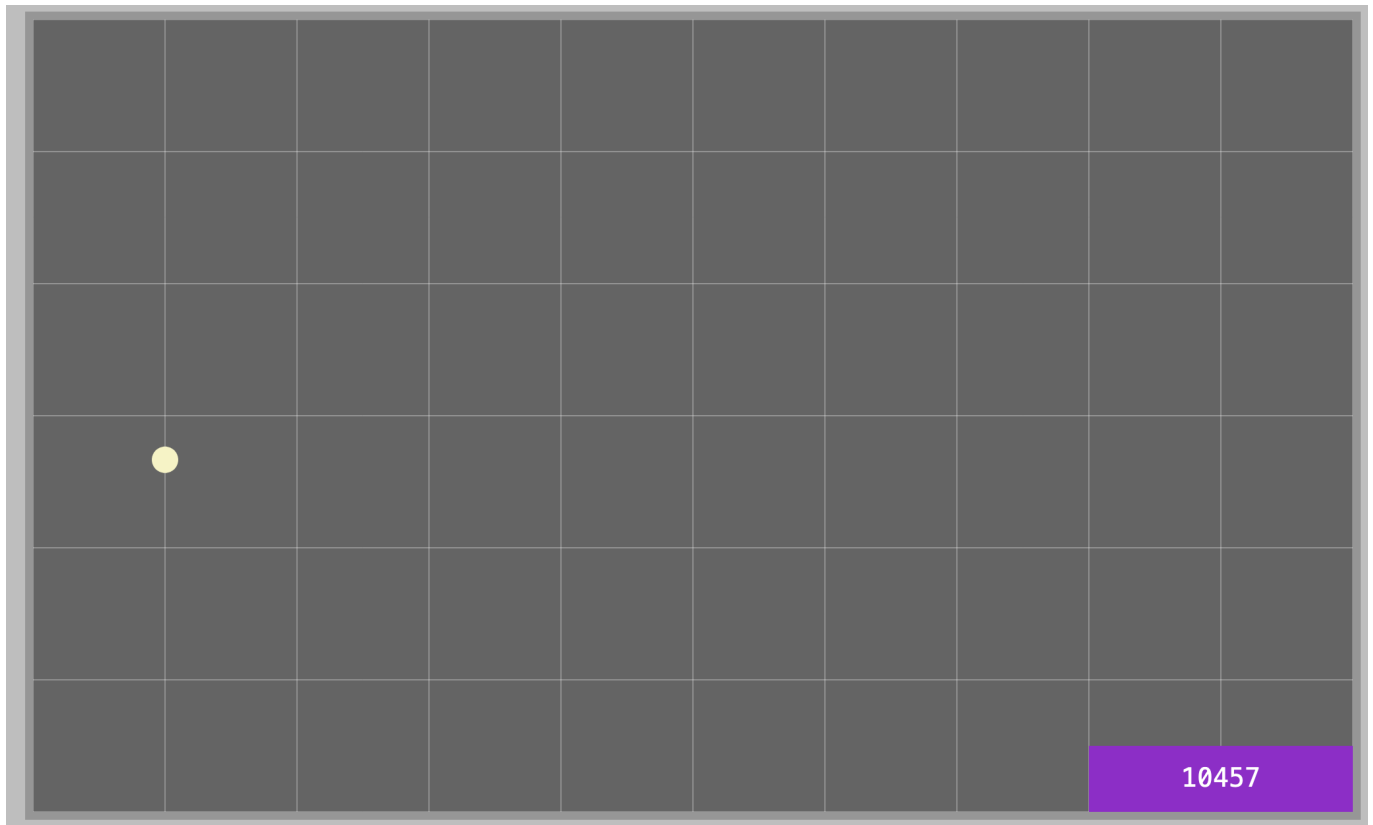
We can introduce left right motion.

```
*/
class MovingBallBoth{
    constructor(x_b,y_b,r, fillcolour, T, H, xIS){
        this.x_b = x_b;
        this.y_b = y_b;
        this.r = r;
        this.fillColour = fillcolour;
        // animation variables
        // set a random starting point (while be helpful when we have lots of
    moving balls)
        this.t = T*Math.random();
        this.T = T;
        this.H = H;
        // introduce an interval shift for the x interval
        // this will make the ball behave more naturalistically
        this.xIntervalShift = xIS
    }
    update(){
        // add one to the value of little t each time update is called
        this.t +=1
        this.draw()
    }
    draw(){
        // get y value from the piecewise function
        let y = this.linearinterpolate(this.t, this.T, this.H)
        // the interval is multiplied by the x interval shift
        let x = this.linearinterpolate(this.t, this.T*this.xIntervalShift, this.H
    )
        this.drawCircle(x+this.x_b,y+this.y_b, this.r)
```

Listing 18: Part of Bothways Moving Ball

32

12927
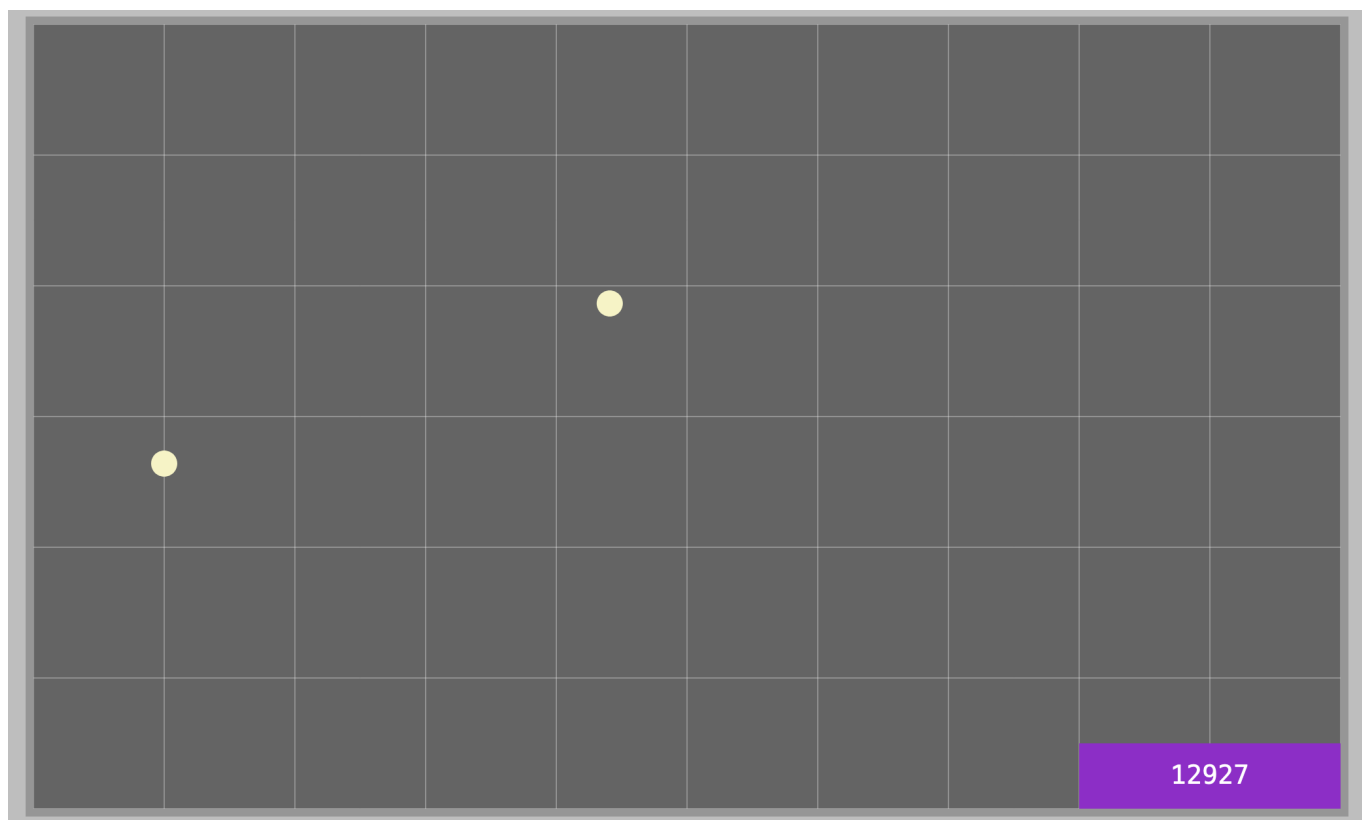
Introduce a whole group (or field) of moving balls.

```
1  /**
2   * Block of moving balls
3   * @param {number} x_b base x position
4   * @param {number} y_b base y position
5   * @param {number} radius radius
6   * @param {string} fillColour fill colour
7   * @param {number} T total Tick interval (50 ticks = about 1 second)
8   * @param {number} H total Height covered by up/down motion
9   */
10 class BallGroup{
11     constructor(x_b,y_b,r, fillColour, T, H){
12         this.x_b = x_b
13         this.y_b = y_b
14         this.H = H
15         this.fillColour = fillColour
16         // create a list that is going to hold a whole set of ball objects
17         this.BSet = []
18         //run a loop (in this case x50)
19         for(let i=0; i<50; i++){
20             // randomly set the amount of x interval shift
21             let xIS = 1+5*Math.random()
22             // randomly adjust the interval
23             let randT = T + 2*T*Math.random()
24             // create random red, green, blue
25             let red = 255*Math.random();
26             let green = 255*Math.random();
27             let blue = 255*Math.random();
28             // create random transparency
29             let alpha = Math.random();
30             // concatenate to make a rgb string
31             let randColour = "rgba("+ red + ","+ green + "," + blue + "," + alpha
    +")"
32             // randomly adjust the radius size
33             let radius = r*7*Math.random()+r
34             // create a moving ball using these values
35             let temp = new MovingBallBoth(x_b,y_b,radius, randColour, randT, H,
    xIS)
36             // push it into the BSet list
37             this.BSet.push(temp)
38         }
39     }
40     update(){
41         // this is an extra bit
42         // save the canvas
```

```
43        ctx.save()
44        // draw and fill a background rectangle
45        this.drawRect(this.x_b, this.y_b, this.H, this.H)
46        // the clip method will "clip out" anything outside the rectangle
47        ctx.clip()
48        // run a loop through the BSet
49        for(let i=0; i<this.BSet.length ; i++){
50            // call update on each moving ball
51            this.BSet[i].update()
52        }
53        // restore the context (this removes the clip)
54        ctx.restore()
```
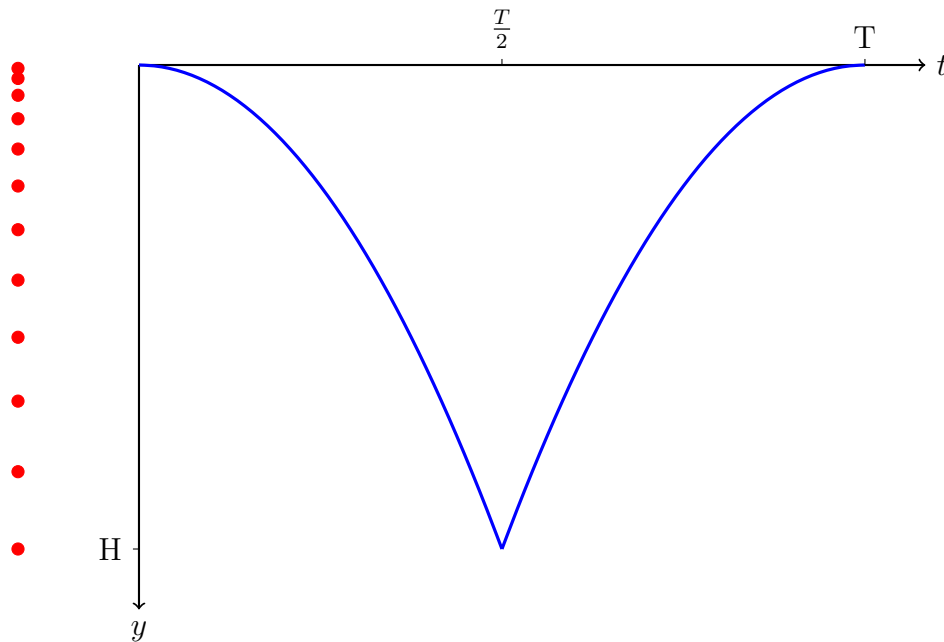
Listing 19: Field of moving balls
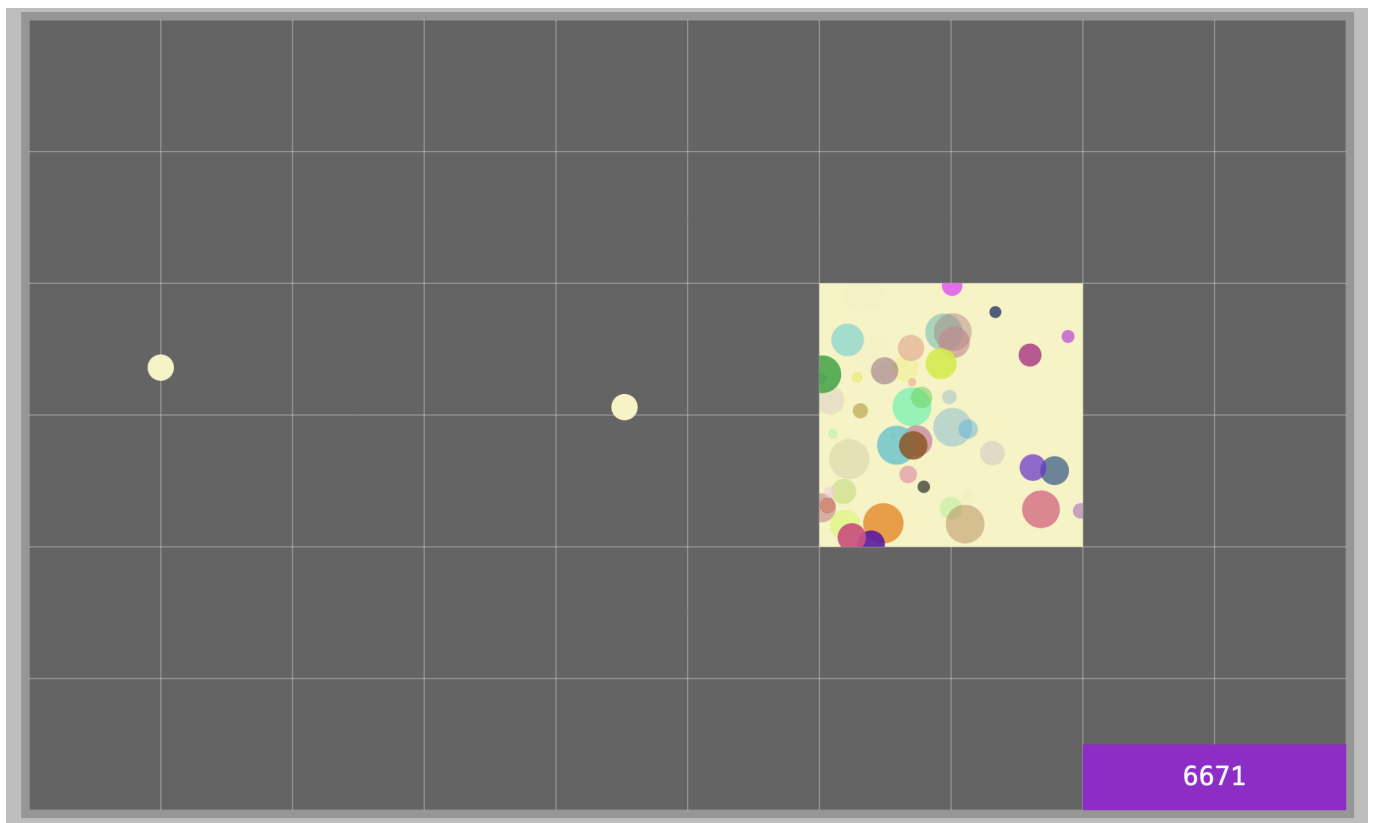
13.2   Quadratic Interpolation

We can follow the same idea using parabolas:

$$
\begin{cases}
y = \dfrac{4Ht^2}{T^2} & , \qquad 0 < t \le \dfrac{T}{2} \\[3em]
y = \dfrac{4H(t-T)^2}{T^2} & , \quad \dfrac{T}{2} < x \le T
\end{cases}
$$

In this case, we would have the up down motion as quadratic, and the the left right as linear.

Below is the code for a Quadratic Ball Class.

Notice how short it is. What's going on here?

```
class QuadraticBall extends MovingBallBoth{
    draw(){
        // get y value from the piecewise function
        let y = this.quadraticInterpolate(this.t, this.T, this.H)
        // the interval is multiplied by the x interval shift
        let x = this.linearinterpolate(this.t +this.T*this.xIntervalShift/2, this
    .T*this.xIntervalShift, this.H)
        this.drawCircle(x+this.x_b,y+this.y_b, this.r)
    }
    quadraticInterpolate(t,T,H){
        // takes parameter t , T, H
        // we could hard code in this.T etc but is more fexible to have
    parameters
        // make sure t is between 0 and T
        t = t%T;  // modulus operator
        // set y variable and use to get value from equations
```
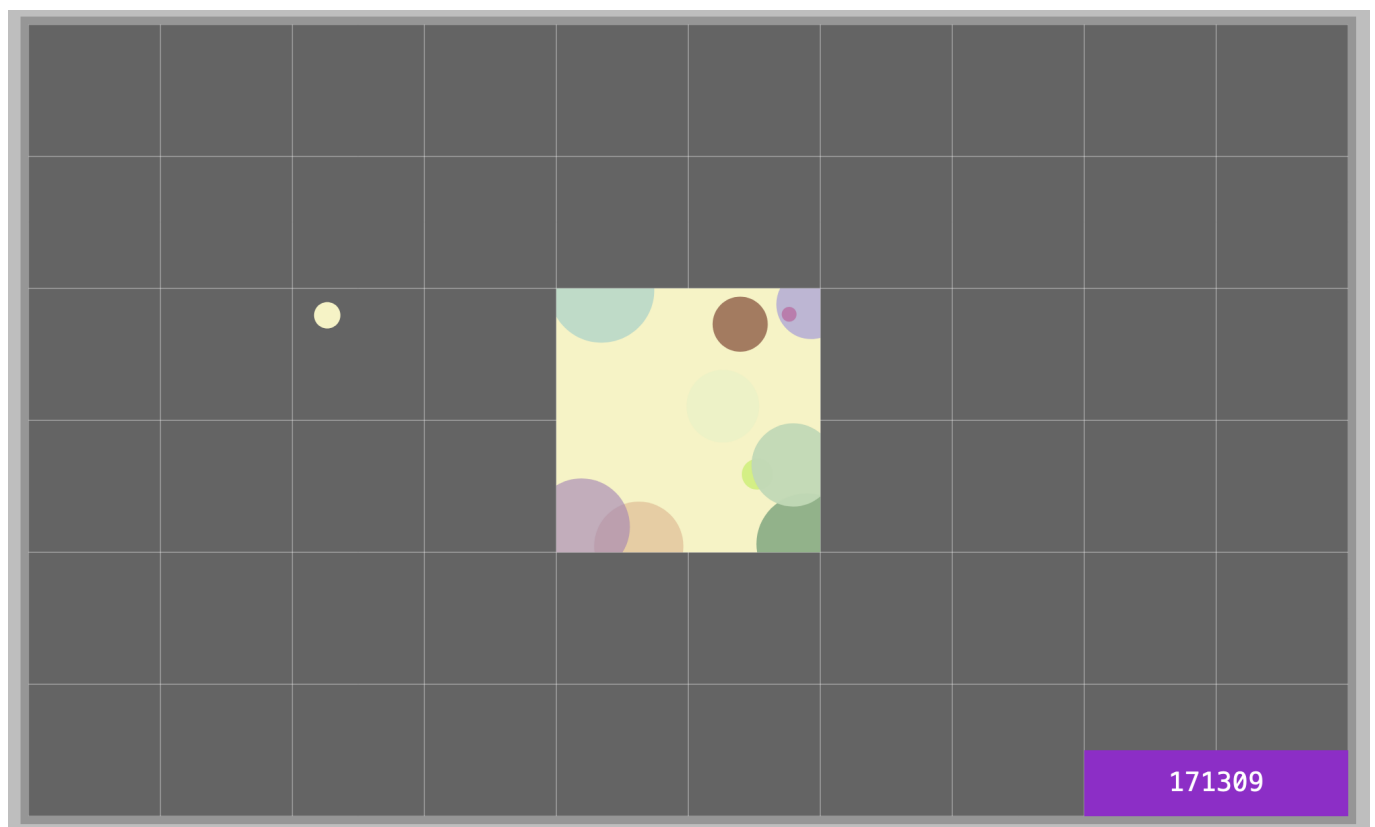
```
15        let y;
16        if(t<T/2){
17            y = ( 4*H*Math.pow(t,2) )/( Math.pow(T,2) )
18        }else{
19            y = ( 4*H*Math.pow(t-T,2) )/( Math.pow(T,2) )
20        }
21        return y
22    }
23 }
```

Listing 20: Quadratic Ball Class

A small rewrite has been done on the QuadraticBallGroup Class.

This will mean it can be extended in the next section.

```
1            let radius = r*10*Math.random()+r
2            // create a moving ball using these values
3            let temp = this.getObject(x_b,y_b,radius, randColour, randT, H, xIS)
4            // push it into the BSet list
5            this.BSet.push(temp)
6        }
7    }
8    getObject(x_b,y_b,radius, randColour, randT, H, xIS){
9        return  new QuadraticBall(x_b,y_b,radius, randColour, randT, H, xIS)
10    }
```
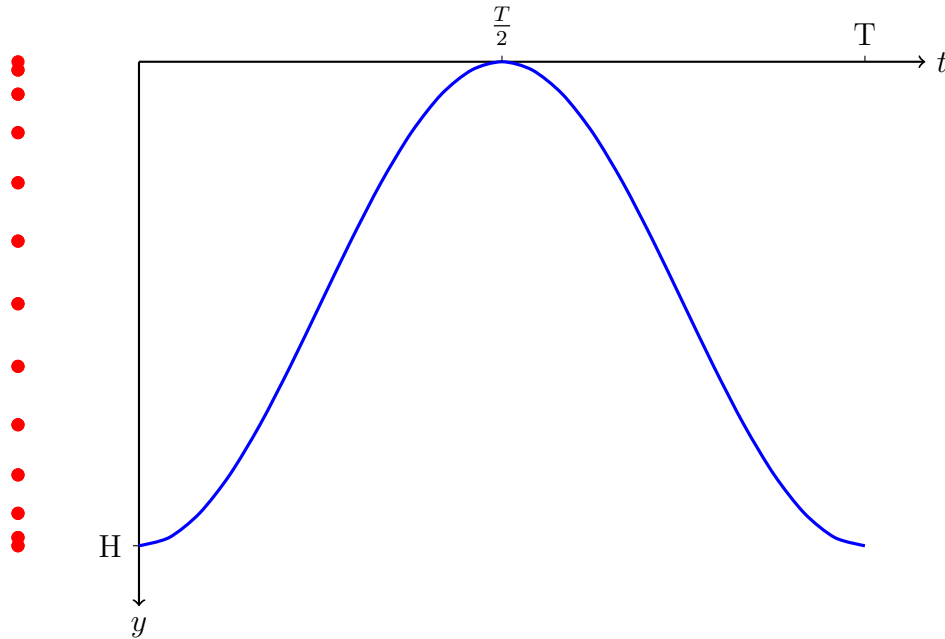
```
11    update(){
```

Listing 21: Quadratic Ball Class

13.3   Trigonometric Interpolation

We can follow the same idea using parabolas:

$$y = \frac{H}{2}\cos(\frac{2\pi}{T}t) + \frac{H}{2}$$

Complete code for both the TrigBall class and TrigBallGroup class (both are inheriting from the class referred to in the extension).

```
1  /**
2   * Ball that moves trigonometrically
3   * @param {number} x_b base x position
4   * @param {number} y_b base y position
5   * @param {number} radius radius
6   * @param {string} fillColour fill colour
7   * @param {number} T total Tick interval (50 ticks = about 1 second)
8   * @param {number} H total Height covered by up/down motion
9   * @param {number} xIS x interval shift (to make a different interval for left
      right movement
10  */
11 class TrigBall extends MovingBallBoth{
12     draw(){
13         // get y value from the piecewise function
14         let y = this.trigInterpolate(this.t, this.T, this.H)
15         // the interval is multiplied by the x interval shift
16         // see what happens with this commented out code
17         //let x = this.trigInterpolate(this.t + this.T/4, this.T, this.H)
18         let x = this.linearinterpolate(this.t +this.T*this.xIntervalShift/2, this
    .T*this.xIntervalShift, this.H)
```

```
19          this.drawCircle(x+this.x_b,y+this.y_b, this.r)
20      }
21      trigInterpolate(t,T,H){
22          let y = (H/2) * Math.cos( (2*Math.PI/T) *t) + H/2
23          return y
24      }
25 }
26 /**
27  *Group of trigonometrically moving balls
28  * @param {number} x_b base x position
29  * @param {number} y_b base y position
30  * @param {number} radius radius
31  * @param {string} fillColour fill colour
32  * @param {number} T total Tick interval (50 ticks = about 1 second)
33  * @param {number} H total Height covered by up/down motion
34  */
35 class TrigBallGroup extends QuadraticBallGroup{
36      getObject(x_b,y_b,radius, randColour, randT, H, xIS){
37          return new TrigBall(x_b,y_b,radius, randColour, randT, H, xIS)
38      }
39 }
```

Listing 22: Quadratic Ball Class
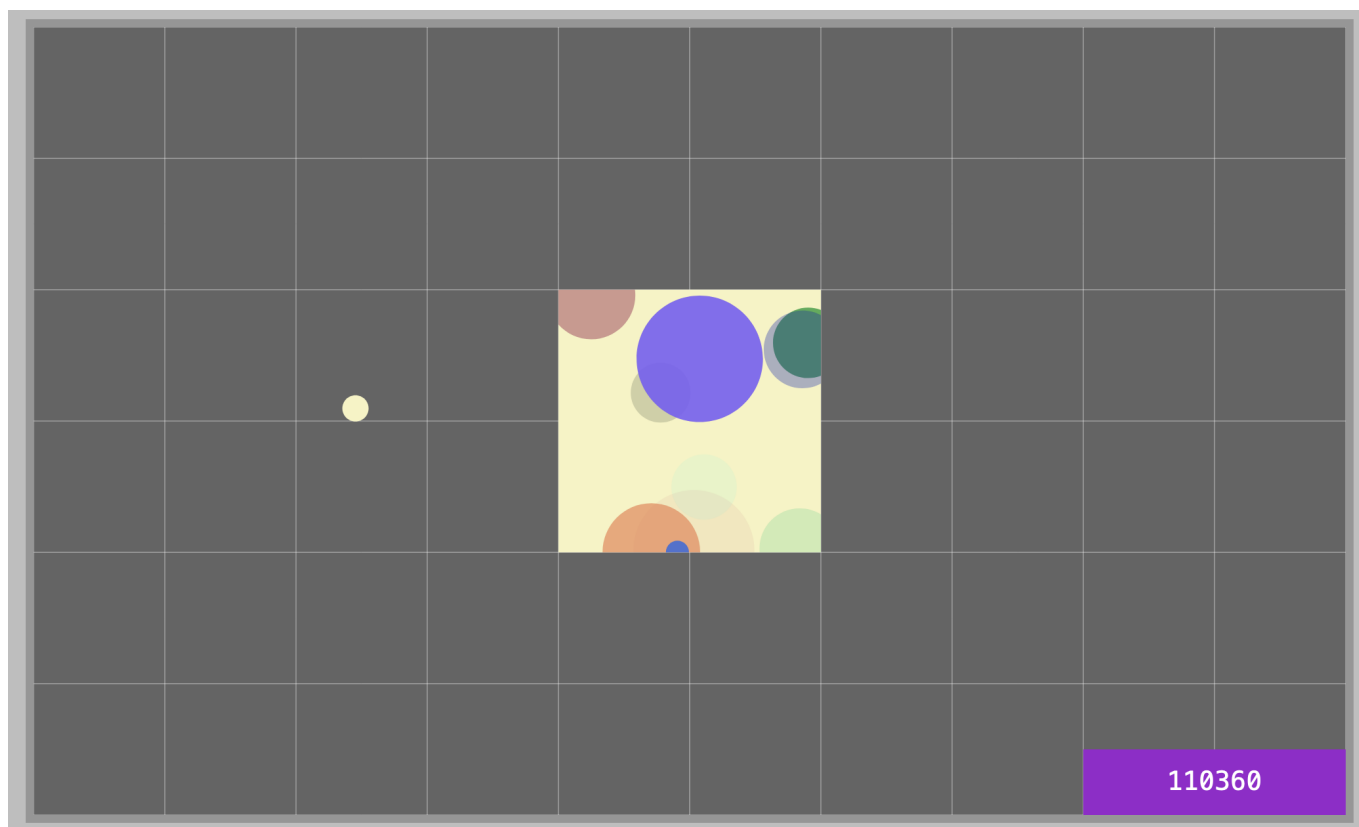
14 Event Handling

We want to able to monitor **events**. If you are researching this you are looking for 'DOM events javascript'.

The main events we are concerned with are **mouse events**.

These are when when the user clicks the mouse, presses the mouse down, moves the mouse.

There are also 'touch' events, but this adds another layer of complexity and testing, so this has been avoided for this course.

There are many other events as well (you can research).

We get these events by creating a **listener** that we attach to the canvas page element (node).

The listener then calls the function that is attached to it.

When the listener 'fires' it creates an event object , which contains a substantial amount of data.

To see it, put a console.log(e) in one of the mouse functions.

We mainly want to know:

- Is the mouse down?

- Is the mouse up?

- What is the current (x,y) position of the mouse (in terms of the default canvas coordinates)?

This creates a significant block of code, so it is best to create a single object to manage this, and we can then , via inheritance, integrate it into any object that needs it.

```
1  /**
2   * Captures mouse events
3   * Note that are no parameters for the constructor
4   * There is no update function - events are independent of the animation frame
5   */
6  class InteractiveObject{
7      constructor(){
8          // this listen for a mouse event - anywhere on the canvas
9          canvas.addEventListener('mousedown', this.mDown.bind(this));
10         canvas.addEventListener('mouseup', this.mUp.bind(this));
11         canvas.addEventListener('mousemove', this.mMove.bind(this));
12         canvas.addEventListener('mouseleave', this.mLeave.bind(this));
13         // variables to hold where the mouse was first clicked down
14         // we will need them later
15         this.xStart = 0
16         this.yStart = 0
17         // variables to hold the current mouse position
18         this.xMouse = 0;
19         this.yMouse = 0;
20         // it will also be helpful to know if the mouse is down
21         this.mouseIsDown = false;
```

```
22        }
23     mDown(e){
24         // update positions so this can be used in another object
25         this.xStart = e.offsetX;
26         this.yStart = e.offsetY;
27         // yes the mouse is down
28         this.mouseIsDown = true;
29         //once you have got the idea, comment out these (and remove later)
30         let output = "This mouse went DOWN at  x = " + e.offsetX + " and y = " +
    e.offsetY;
31         // console.log(output)
32     }
33     mUp(e){
34         // if the mouse is up, it can't be down :)
35         this.mouseIsDown = false;
36         //once you have got the idea, comment out these (and remove later)
37         let output = "This mouse went UP at x = " + e.offsetX + " and y = " + e.
    offsetY;
38         //console.log(output);
39
40     }
41     mMove(e){
42         // update positions so this can be used in another object
43         this.xMouse = e.offsetX;
44         this.yMouse = e.offsetY;
45         //console.log("moving")
46     }
47     mLeave(e){
48         // this might be a useful safety feature
49         // we could set mouseIsDown to false when the mouse leave the canvas
50         console.log("Mouse has left the canvas")
51     }
52 }
```

Listing 23: Interactive Object

15   Basic Draggable Point

To make use of it we can 'inherit' into an interactive ball object, which will now follow the mouse when it is down.

```
/**
 * Interactive Ball
 * Includes all functions from interactive object
 * @param {number} x ball centre x
 * @param {number} y ball centre y
 * @param {number} r radius of ball
 * @param {string} fill fill colour
 * @param {string} stroke stroke colour
 * @param {number} strokeWidth  width of outline
 */
class InteractiveBall extends InteractiveObject{
    // all the functions of InteractiveObject are part of InteractiveBall
    constructor(x,y,r,fill, stroke, strokeWidth){
        // super initialises the constructor of InteractiveObject
        super()
        this.x = x;
        this.y = y;
        this.r=r;
        this.fill=fill;
        this.stroke = stroke;
        this.strokeWidth = strokeWidth;
    }
    update(){
        this.draw();
        // check if mouse is down and update x, y coordinates of the ball
        // to be the same as the x,y mouse positions
        if(this.mouseIsDown){
            this.x = this.xMouse;
            this.y = this.yMouse;
        }
    }
    draw(){
        this.drawCircle(this.x, this.y, this.r, this.fill, this.stroke, this.
    strokeWidth)
    }
    drawCircle(x,y,r,f,s,l){
        ctx.beginPath()
        ctx.arc(x, y, r, 0, 2*Math.PI)
        ctx.fillStyle = f
        ctx.strokeStyle = s
        ctx.lineWidth = l
        ctx.fill();
```
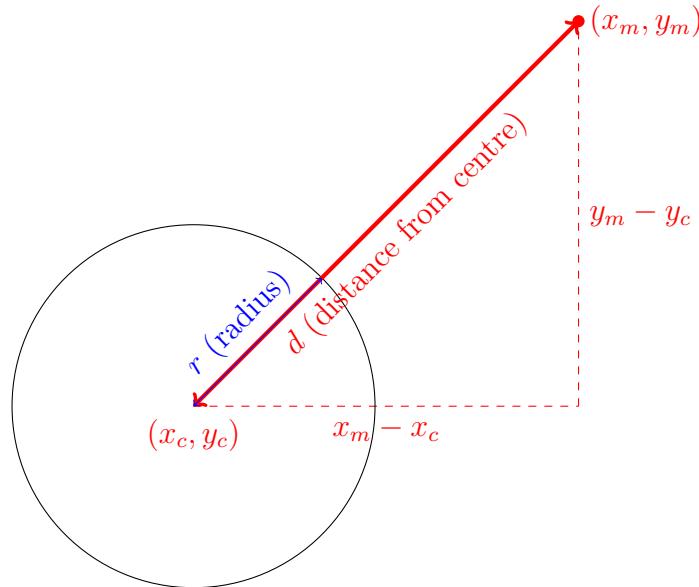
```
42        ctx.stroke()
43    }
44 }
```

Listing 24: Interactive Object

16   Better Draggable Point

We would like the ball to move only if the mouse is inside it.

This is called **hit testing**.

The distance from the centre of the circle to where the mouse is , is:

$$d = \sqrt{(x_m - x_c)^2 + (y_m - y_c)^2}$$

If $d < r$, then the mouse is inside the circle,

Otherwise it is not.

```
1  /**
2   * Interactive Ball Better
3   * Includes all functions from interactive object
4   * @param {number} x ball centre x
5   * @param {number} y ball centre y
6   * @param {number} r radius of ball
7   * @param {string} fill fill colour
8   * @param {string} stroke stroke colour
9   * @param {number} strokeWidth  width of outline
10  */
11 class InteractiveBallBetter extends InteractiveObject{
12     // all the functions of InteractiveObject are part of InteractiveBall
13     constructor(x,y,r,fill, stroke, strokeWidth){
14         // super initialises the constructor of InteractiveObject
15         super()
16         this.x = x;
17         this.y = y;
18         this.r=r;
```

```
19          this.fill=fill;
20          this.stroke = stroke;
21          this.strokeWidth = strokeWidth;
22          this.selected = false;
23          this.group = [];
24      }
25 // we are now overriding the mouse down function in the original interactive
      object
26 // because we need to add to it
27      mDown(e){
28          // call the super function so the variables update
29          super.mDown(e)
30          // introduce a get boundary call
31          // passing the x, y of the ball and the mouse position and the radius
32          if( this.getBoundary(this.x, this.y, this.xMouse, this.yMouse, this.r) ){
33              // if the mouse goes down and we are in the boundary
34              // then we can say the ball is selected
35              InteractiveBallBetter.taken = this;
36          }
37      }
38
39      update(){
40          this.draw();
41
42          // if selected is true all good for ball to follow mouse
43          if(InteractiveBallBetter.taken === this){
44              this.x = this.xMouse;
45              this.y = this.yMouse;
46          }
47          if(!this.mouseIsDown){
48              InteractiveBallBetter.taken="";
49          }
50      }
51      // boundary check function
52      getBoundary(x_c, y_c, x_m, y_m, r){
53          let d= Math.sqrt(Math.pow(x_m - x_c , 2) + Math.pow(y_m - y_c , 2) )
54          if(d<r){
55              return true
56          }else{
57              return false
58          }
59      }
60      draw(){
61          let fill = this.fill
62          if(this.selected){
63              fill = "rgb(255,255,255)"
64          }
```

```
65        this.drawCircle(this.x, this.y, this.r, fill, this.stroke, this.
      strokeWidth)
66      }
67      drawCircle(x,y,r,f,s,l){
68          ctx.beginPath()
69          ctx.arc(x, y, r, 0, 2*Math.PI)
70          ctx.fillStyle = f
71          ctx.strokeStyle = s
72          ctx.lineWidth = l
73          ctx.fill();
74          ctx.stroke()
75      }
76 }
77 // create a static variable this will hold the point that is selected
78 // only one point can ever be selected
79 InteractiveBallBetter.taken = ""
```

Listing 25: Interactive Object

Start up code:

```
1      let G = new Grid(width, height, 50, col[0][2], 0.3)
2      let T = new TextBox(800,550,200, col[0][4], col[0][2])
3      let I = new InteractiveObject();
4
5      let IBB_A = new InteractiveBallBetter(600,300,10, col[0][6], col[0][7], 5)
6      let IBB_B = new InteractiveBallBetter(700,300,10, col[0][6], col[0][7], 5)
7      let IBB_C = new InteractiveBallBetter(800,300,10, col[0][6], col[0][7], 5)
8      // create an animation function
9      function animate(t){
10          ctx.clearRect(0,0, width, height);
11          G.update();
12          let timer = Math.round(t)
13          T.update(timer);
14          IBB_A.update()
15          IBB_B.update()
16          IBB_C.update()
17
18          // the call below is a request to the browser
19          // the function is called again (about 50 times a second)
20          window.requestAnimationFrame(animate)
21      }
22      // start off call to get it going
23      animate()
24 </script>
25 </body>
26 </html>
```
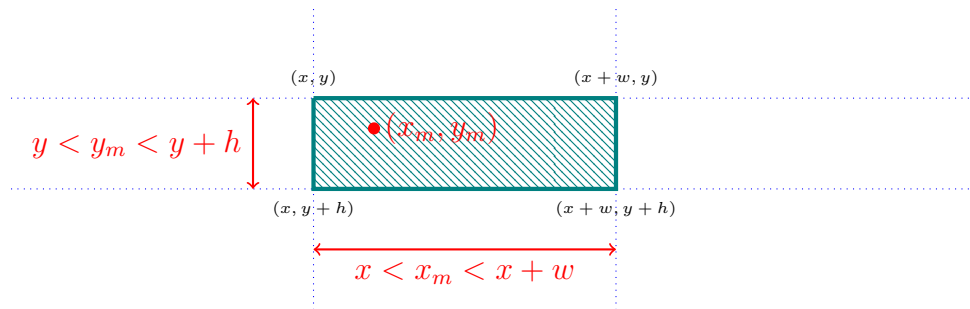
Listing 26: Interactive Object

17 Buttons

To create a button

- We need to register a user "click"

- We need to know that the user has clicked with the mouse inside the button

- We also need to give plenty of feedback (a hover colour and a colour change to show the button has been 'selected')

Hit testing for a rectangle:

For the point (x_m, y_m) to be inside the rectangle , the follow conditions must **all** be met.

$$x_m > x \ \ and \ \ x_m < x + w \ \ and \ \ y_m > y \ \ and \ \ y_m < y + h$$

```
 1  * Clickable Button
 2  * Includes all functions from interactive object
 3  * @param {number} x ball centre x
 4  * @param {number} y ball centre y
 5  * @param {number} w radius of ball
 6  * @param {number} h radius of ball
 7  * @param {string} fill fill colour
 8  * @param {string} over hover over  colour
 9  * @param {string} selected button has been clicked colour
10  * @param {string} stroke stroke colour
11  * @param {string} text  button text
12  * @param {string} text  button text colour
13  */
14 class InteractiveButton extends InteractiveObject{
15    constructor(x,y, w, h, fill, over, selected, stroke, text, textColour){
16        super()
17        this.x = x;
18        this.y = y;
19        this.w = w;
20        this.h = h;
21        this.fill = fill;
22        this.over=over;
```

```
23         this.selected = selected;
24         this.stroke=stroke;
25         this.text = text;
26         this.textColour = textColour
27         this.inBounds = false
28     }
29    update(){
30         this.inBounds = this.getBoundary(this.x, this.y, this.w, this.h, this.
   xMouse, this.yMouse)
31         let fill = this.fill
32         if(InteractiveButton.selected === this){
33             fill = this.selected
34         }else if(this.inBounds ){
35             fill = this.over
36         }
37         this.draw(this.x, this.y, this.w, this.h, fill, this.stroke, this.text,
   this.textColour )
38     }
39    mClick() {
40         // check mouse in bounds
41         if(this.inBounds){
42             InteractiveButton.selected = this;
43         }
44     }
45    getBoundary(x,y,w,h,x_m,y_m){
46         if(x_m > x && x_m < x + w && y_m > y && y_m < y +h){
47             return true
48         }else{
49             return false
50         }
51     }
52    draw(x,y, w,h,c,s, txt, txtCol){
53         ctx.beginPath()
54         ctx.rect(x,y,w,h);
55         ctx.lineWidth = 2;
56         ctx.strokeStyle = s;
57         ctx.fillStyle = c;
58         ctx.fill();
59         ctx.stroke();
60
61         let myFont= "bold 20px 'Trebuchet MS', Verdana, sans-serif  ";
62         ctx.textBaseline = 'middle';
63         ctx.textAlign = 'center';
64         ctx.font=myFont;
65
66         ctx.fillStyle = txtCol;
67         ctx.fillText(txt,   x+ w/2  ,y+h/2);
```
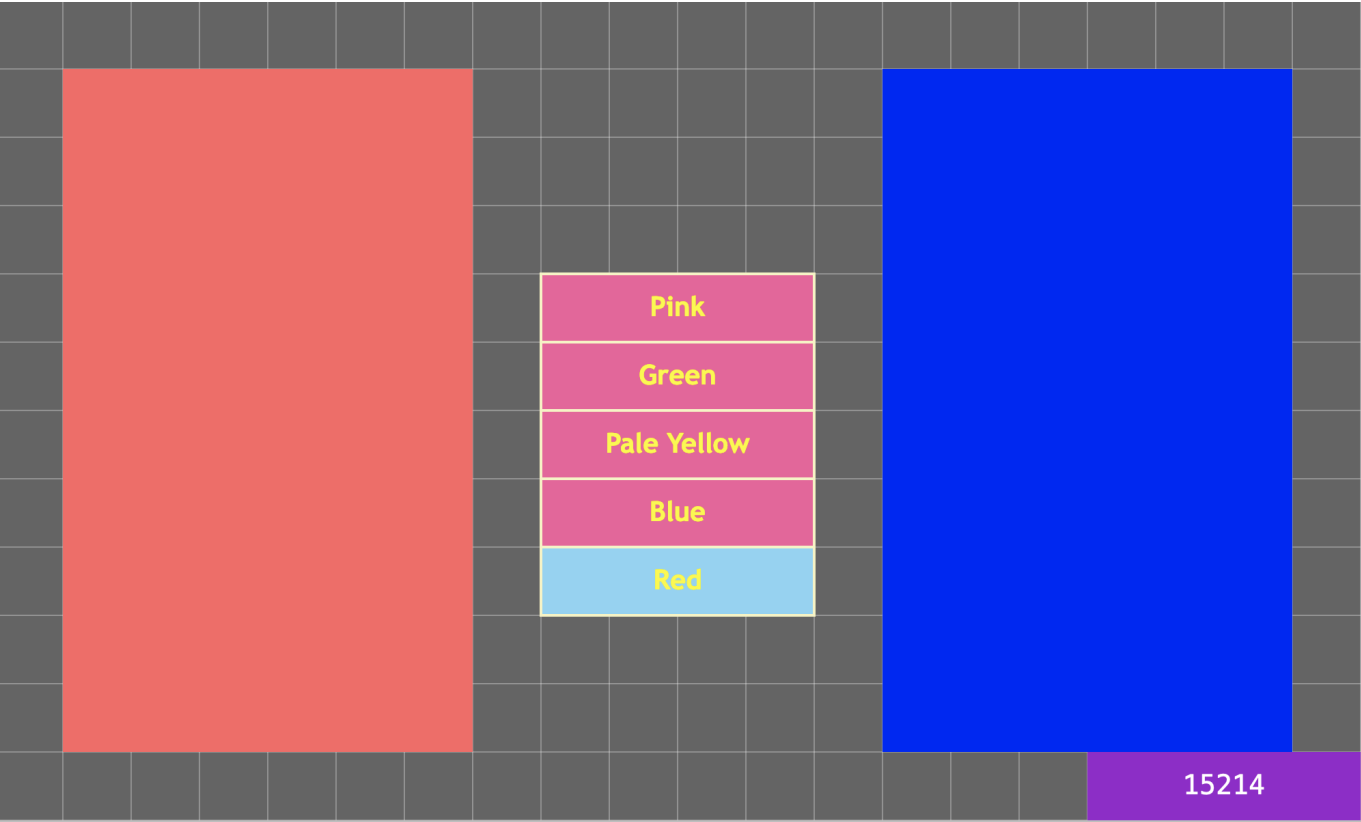
```
68        }
69 }
70 InteractiveButton.selected = null;
```

Listing 27: Button

18   Manager Object

50

| Pink |
| Green |
| Pale Yellow |
| Blue |
| Red |

15214

19 Code refactoring - prototyping

20 Old Stuff

Older Code example

```
1  console.log("point js file has been called");
2  class Point{
3  // class Point x,y,r, stroke, fill, over, canvas
4  constructor(x,y,r, stroke, fill, over){
5      //basic position, size and colours
6      this.x = x;
7      this.y = y;
8      this.r = r;
9      this.stroke = stroke;
10     this.fill = fill;
11     this.over = over;
12     //set true if mouse inside point circle
13     this.inBounds = false;
14     //cointinually registered mouse position
15     this.xMouse = 0;
16     this.yMouse = 0;
17     //listeners
18     canvas.addEventListener('mousedown', this.mDown.bind(this));
19     canvas.addEventListener('mousemove', this.mMove.bind(this));
20     canvas.addEventListener('mouseup', this.mUp.bind(this));
21  }
22  mDown(e){
23      // if the mouse is pressed (goes down) and the mouse is inside the point
       circle,
24      // set the this object as taken
25      if(this.inBounds){
26          Point.taken = this;
27      }
28  }
29  mMove(e){
30      // event registered every time the mouse moves
31      // object variables updated with current mouse position
32      this.xMouse = e.offsetX;
33      this.yMouse = e.offsetY;
34      //update boundary boolean
35      this.inBounds = this.boundsCheck(this.xMouse, this.yMouse, this.x, this.y,
       this.r);
36  }
37  mUp(e){
38      //when mouse goes up set taken point as nothing
39      //hence deselect this point
40      Point.taken = "";
41  }
```

```javascript
42 /**
43  * called from animation loop
44  */
45 update(){
46 // make x,y coordinates of the point the same as the mouse position
47 // if the point has been taken
48     if(Point.taken == this){
49         this.x=this.xMouse;
50         this.y=this.yMouse;
51     }
52     this.draw();
53 }
54 draw(){
55     // change fill state if mouse is over or the point is selected
56     if(this.inBounds || Point.taken == this){
57     ctx.fillStyle= this.over;
58     }else{
59         ctx.fillStyle= this.fill;
60     }
61     ctx.strokeStyle = this.stroke;
62     ctx.lineWidth = 2;
63     ctx.beginPath()
64     ctx.arc(this.x,this.y, this.r, 0, 2*Math.PI);
65     ctx.fill();
66     ctx.stroke();
67 }
68 /**
69  * Pythagoras distance check
70  * @param  x,y,positions of mouse and of point circle and radius of point circle
        (number)
71  * @return boolean
72  */
73 boundsCheck(x_1, y_1, x_2, y_2, r){
74         var d = Math.sqrt( Math.pow(x_2 - x_1, 2) + Math.pow(y_2 - y_1, 2) );
75         if(d<r){
76             return true;
77         }else{
78             return false;
79         }
80 }
81 /**
82  * Make x, y coordinates of point available outside of object
83  * @return number
84  */
85 getX(){
86     return this.x;
87 }
```

```
88 getY(){
89     return this.y;
90 }
91 }
92 // static variable available to all Point objects
93 // the same for all Point objects
94 // means only one Point can be selected and moveable
95 Point.taken="";
```

Listing 28: Python example