

Contents

1	Introduction	2
2	Getting started	3
3	JavaScript in an external file	5
4	Improving the set up	6
5	Functions	7
6	Making a grid	10
7	Update Context Function	11
8	Colour management	12
9	Organising an init file	13
10	Rotations	18
11	Objects	19
12	Animation Frame	23
13	Animation	28
13.1	Linear Interpolation	28
13.2	Quadratic Interpolation	35
13.3	Trigonometric Interpolation	38
14	Event Handling	40
15	Basic Draggable Point	40
16	Better Draggable Point	40
17	Buttons	42

1 Introduction

On a basic web page, we can construct a 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

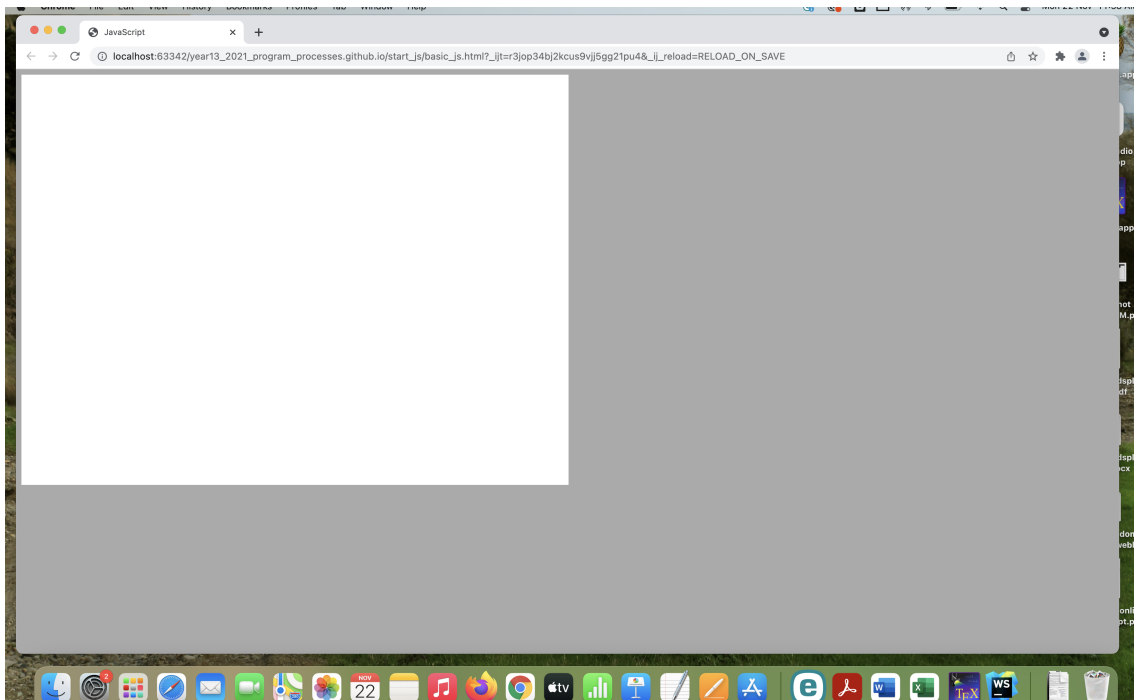


Figure 1: Canvas in browser

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)";

```

```

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 to manage the size and basic styling of the canvas from within the JavaScript.

```
1 canvas = document.querySelector('#myCanvas');
2 let ctx = canvas.getContext('2d');
3 // define width and height
4 let width = 1000;
5 let height = 500;
6 // 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)";
```

Listing 4: Initial JS

5 Functions

We can immediately see that even drawing a small number of shapes starts to 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

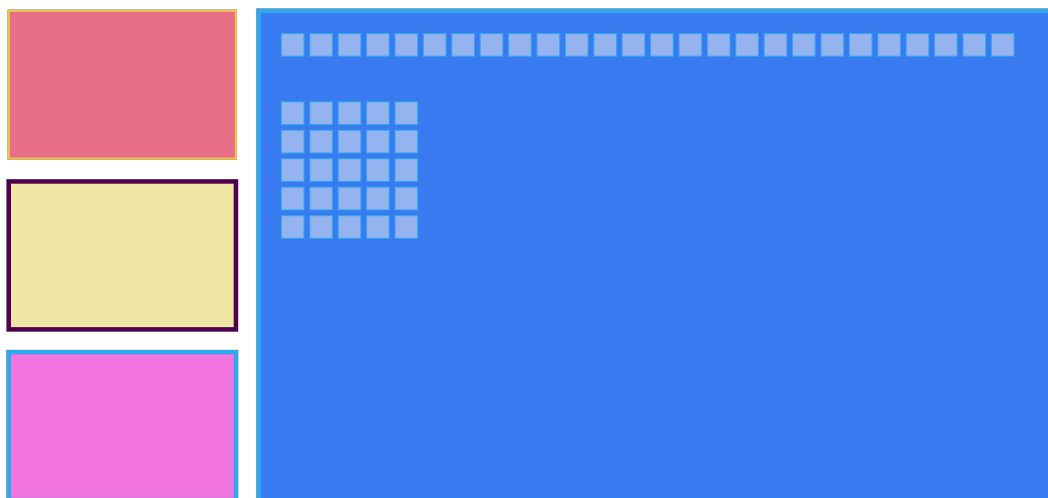


Figure 2: Canvas in browser

Design Functions for:

- Circle
- Line
- Triangle
- Square
- A Grid

- Text Box

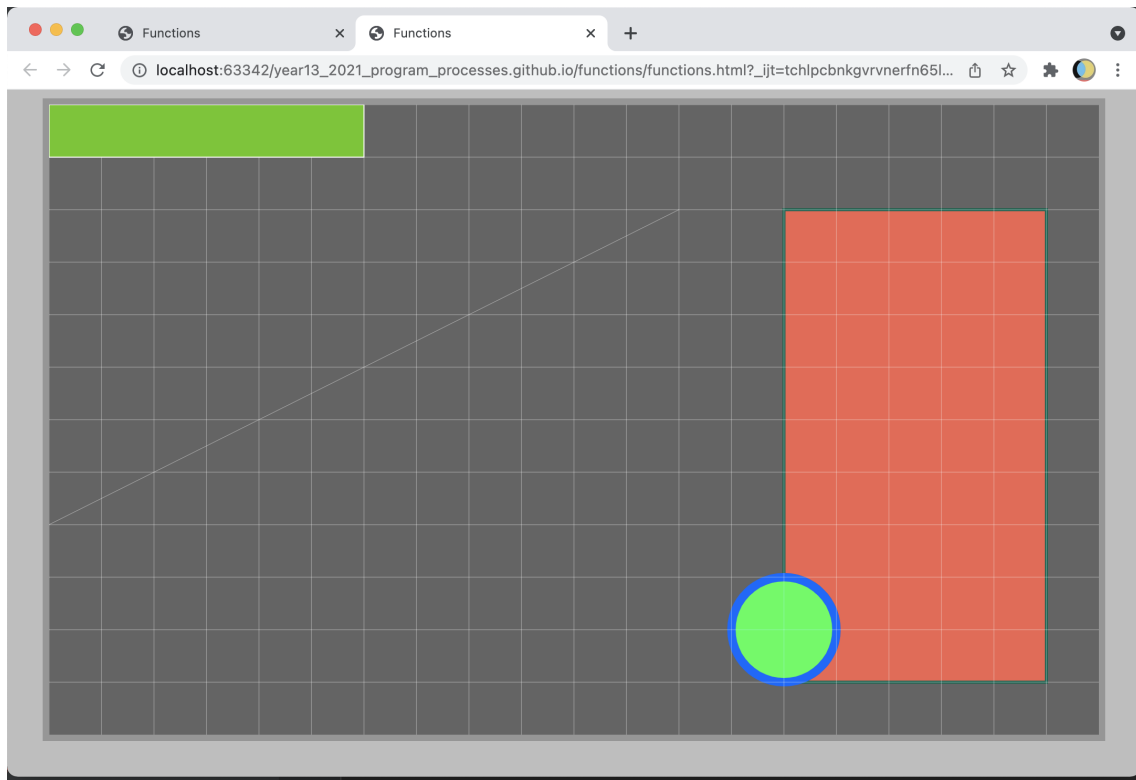


Figure 3: Canvas in browser

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.

```
1 function updateContext(fillColour, strokeColour, strokeWidth){
2     ctx.fillStyle = fillColour;
3     ctx.strokeStyle = strokeColour;
4     ctx.lineWidth = strokeWidth;
5     if(fillColour){
6         ctx.fill();
7     }
8     if(strokeColour || strokeWidth){
9         ctx.stroke();
10    }
11 }
12 /**
13  * Draw a rectangle
14  *
15  * @param {number} x corner x
16  * @param {number} y corner y
17  * @param {number} w width
18  * @param {number} h height
19  * @param {string} fillColour rgb string
20  * @param {string} strokeColour rgb string.
21  * @param {number} strokeWidth
22  * @return {null}
23  */
24 function drawRectangle(x,y,w,h, fillColour, strokeColour, strokeWidth){
25     ctx.beginPath();
26     ctx.rect(x,y,w,h);
27     updateContext(fillColour, strokeColour, strokeWidth)
28 }
```

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.

```
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>Functions Collection</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="interface.js"> </script>
13 </body>
14 </html>
```

Listing 7: Loading javascript

```
1 drawRectangle(20,20,200,130, col[0][3], undefined, undefined);
2 drawRectangle(20,170,200,130, undefined, col[1][6], 9);
3 drawRectangle(20,320,200,130, col[0][3], col[0][6], 4);
4 drawRectangle(240,20,700,430,undefined,col[0][6], 4);
5
6 drawCircle(590, 235, 100, col[0][7], col[1][0], 10)
7
8 drawLine(240,450, 940,20, col[0][2], 1);
9 drawLine(240,20, 940,450, col[0][2], 1);
10 let x= 50
11 let space = 300
12 drawTextBox(x, 480, 260,"Button One", col[0][2],col[0][5]);
13 drawTextBox(x+space, 480, 260,"Button Two", col[0][2],col[0][5]);
14 drawTextBox(x+2*space, 480, 260,"Button Three", col[0][2],col[0][5]);
```

Listing 8: Loading javascript

```
1 canvas = document.querySelector('#myCanvas');
2 let ctx = canvas.getContext('2d');
3 // define width and height
4 let width = 1000;
5 let height = 600;
6 // 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;

```

```

54     ctx.strokeStyle = strokeColour;
55     ctx.lineWidth = strokeWidth;
56     if(fillColour){
57         ctx.fill();
58     }
59     if(strokeColour|| strokeWidth){
60         ctx.stroke();
61     }
62 }
63 /**
64  * Draw a rectangle
65  *
66  * @param {number} x corner x
67  * @param {number} y corner y
68  * @param {number} w width
69  * @param {number} h height
70  * @param {string} fillColour rgb string
71  * @param {string} strokeColour rgb string.
72  * @param {number} strokeWidth
73  * @return {null}
74  */
75 function drawRectangle(x,y,w,h, fillColour, strokeColour, strokeWidth){
76     ctx.beginPath();
77     ctx.rect(x,y,w,h);
78     updateContext(fillColour, strokeColour, strokeWidth)
79 }
80 /**
81  * Draw a circle
82  *
83  * @param {number} x centre x
84  * @param {number} y centre y
85  * @param {number} r radius
86  * @param {string} fillColour rgb string
87  * @param {string} strokeColour rgb string.
88  * @param {number} strokeWidth
89  * @return {null}
90  */
91 function drawCircle(x,y,r, fillColour, strokeColour, strokeWidth){
92     ctx.beginPath();
93     ctx.arc(x,y,r,0,2*Math.PI);
94     updateContext(fillColour, strokeColour, strokeWidth)
95 }
96 /**
97  * Draw a line
98  *
99     * @param {number} x_1 start x
100    * @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";

```

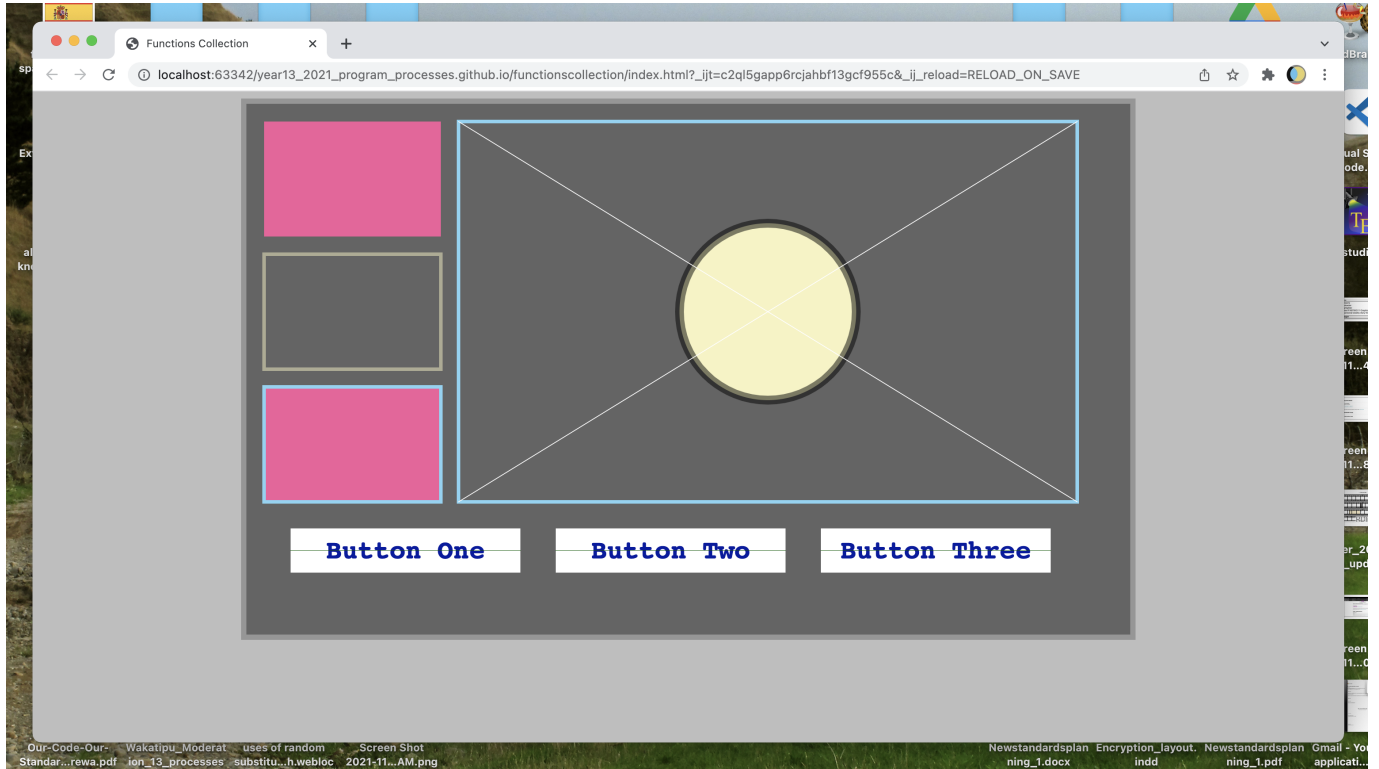


```

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
```

```

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
53         , this.strokeWidth)
54     }
55 }
56 Rectangle.prototype.drawRectangle = drawRectangle
57 Rectangle.prototype.updateContext = updateContext
58 /**
59 * Filled TextBox
60 * @param {number} x top corner of bounding box
61 * @param {number} y top corner of bounding box
62 * @param {number} w width
63 * @param {string} txt text
64 * @param {string} fill fill colour
65 * @param {string} txtColour colour of text
66 */
67 class TextBox{
68     constructor(x,y,width,txt, fillColour, txtColour) {
69         this.x = x;
70         this.y = y;
71         this.w = width;
72         this.txt = txt;
73         this.fillColour = fillColour;
74         this.txtColour = txtColour;
75     }
76     update(txt){
77         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

Set up code

```
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

```

1 /**
2  * Grid - square grid
3  * @param {number} w width of canvas
4  * @param {number} h height of canvas
5  * @param {number} intervalWidth distance each grid unit
6  * @param {string} strokeColour stroke colour
7  * @param {number} strokeWidth width of outline
8  */
9 class Grid{
10     constructor(w,h,intervalWidth, strokeColour, strokeWidth){
11         this.w = w;
12         this.h = h;
13         this.intervalWidth=intervalWidth;
14         this.strokeColour = strokeColour;
15         this.strokeWidth = strokeWidth;
16     }
17     update(){
18         this.draw()
19     }
20     draw(){
21         for(let i = -this.w ; i <= this.w ; i+= this.intervalWidth){
22             this.drawLine(i,-this.h, i,this.h, this.strokeColour, this.
strokeWidth);
23         }
24
25         for(let j = -this.h ; j <= this.h ; j+= this.intervalWidth){
26             this.drawLine(-this.w,j, this.w,j, this.strokeColour, this.
strokeWidth);
27         }
28
29     }
30
31     drawLine(x_1,y_1, x_2, y_2, strokeColour,strokeWidth){
32         ctx.beginPath();
33         ctx.moveTo(x_1,y_1);
34         ctx.lineTo(x_2,y_2);
35         ctx.lineCap = "round";
36         ctx.strokeStyle = strokeColour;
37         ctx.lineWidth = strokeWidth;
38         ctx .stroke()
39     }

```



```

40 }
41
42 /**
43  * Filled TextBox
44  * @param {number} x top corner of bounding box
45  * @param {number} y top corner of bounding box
46  * @param {number} w width
47  * @param {string} txt text
48  * @param {string} fill fill colour
49  * @param {string} txtColour colour of text
50  */
51 class TextBox{
52     constructor(x,y,width, fillColour, txtColour) {
53         this.x = x;
54         this.y = y;
55         this.w = width;
56         // fixed height
57         this.h = 50;
58         // text managed through update
59         this.txt = "Placeholder";
60         console.log(this.txt)
61         this.fillColour = fillColour;
62         this.txtColour = txtColour;
63     }
64     update(txt ="Placeholder"){
65         this.txt = txt
66         this.draw()
67     }
68
69     draw(){
70         ctx.beginPath();
71         ctx.rect(this.x,this.y,this.w,this.h);
72         ctx.fillStyle= this.fillColour;
73         ctx.fill();
74         ctx.font = "20px monospace";
75         ctx.textAlign = "center";
76         ctx.textBaseline = "middle";
77         ctx.fillStyle = this.txtColour;
78         ctx.fillText(this.txt, this.x+this.w/2, this.y+this.h/2);
79     }
80 }

```

Listing 14: objects

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">

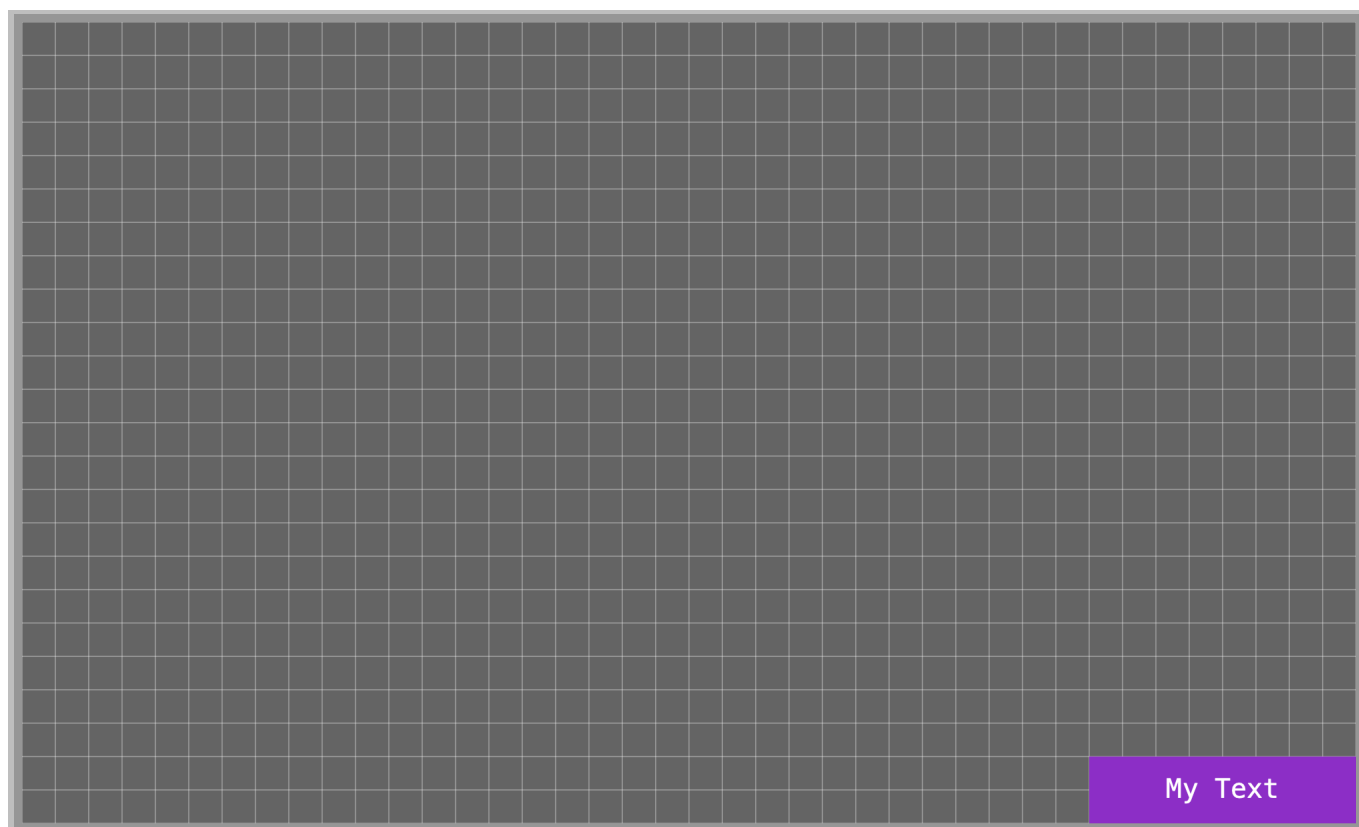
```

```

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



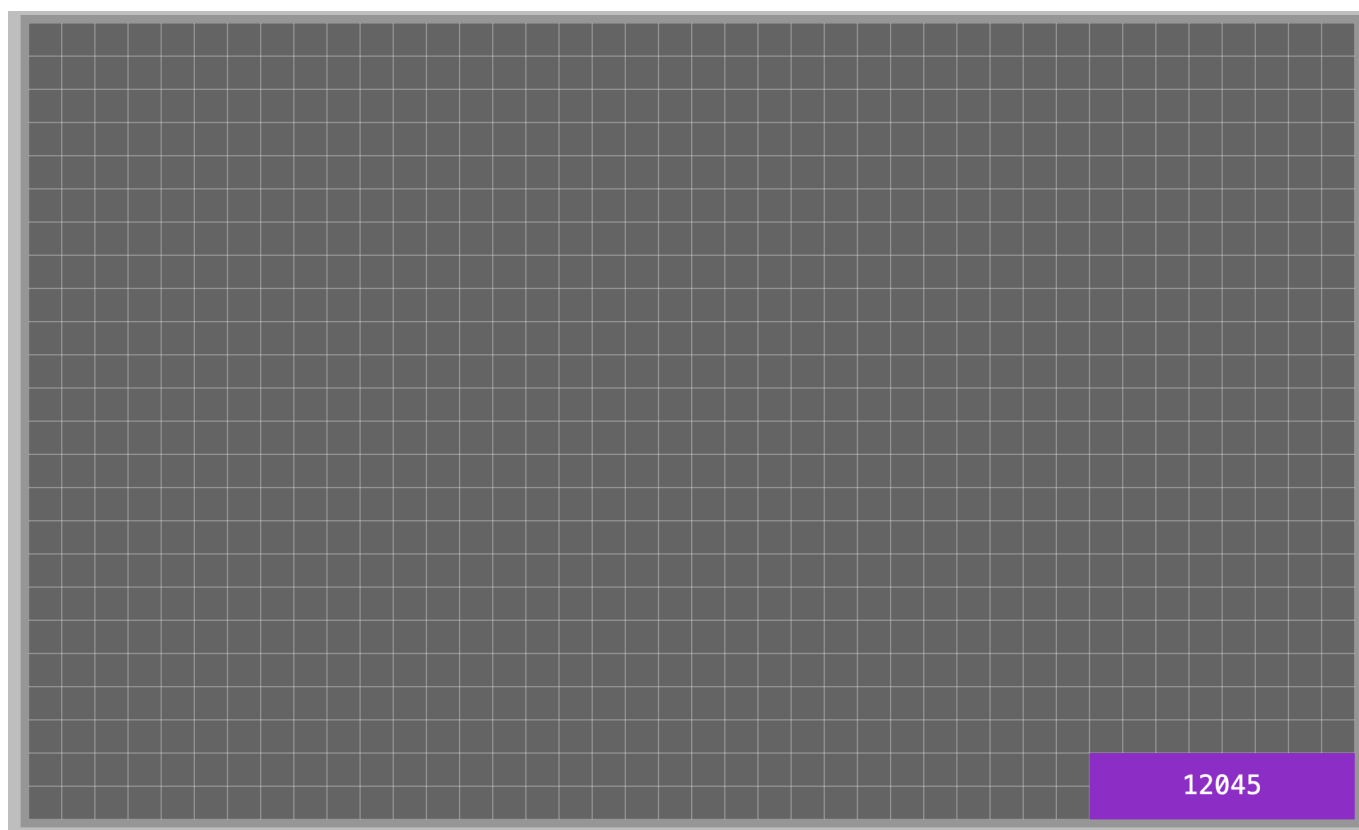
```

1 <script>
2   let G = new Grid(width, height, 25, col[0][2], 0.3)
3   let T = new TextBox(800,550,200, col[0][4], col[0][2])
4   // create an animation function
5   function animate(t){

```

```
6   ctx.clearRect(0,0, width, height);
7   G.update();
8   let timer = Math.round(t)
9   T.update(timer);
10  // the call below is a request to the browser
11  // the function is called again (about 50 times a second)
12  window.requestAnimationFrame(animate)
13  }
14  // start off call to get it going
15  animate()
16 </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.

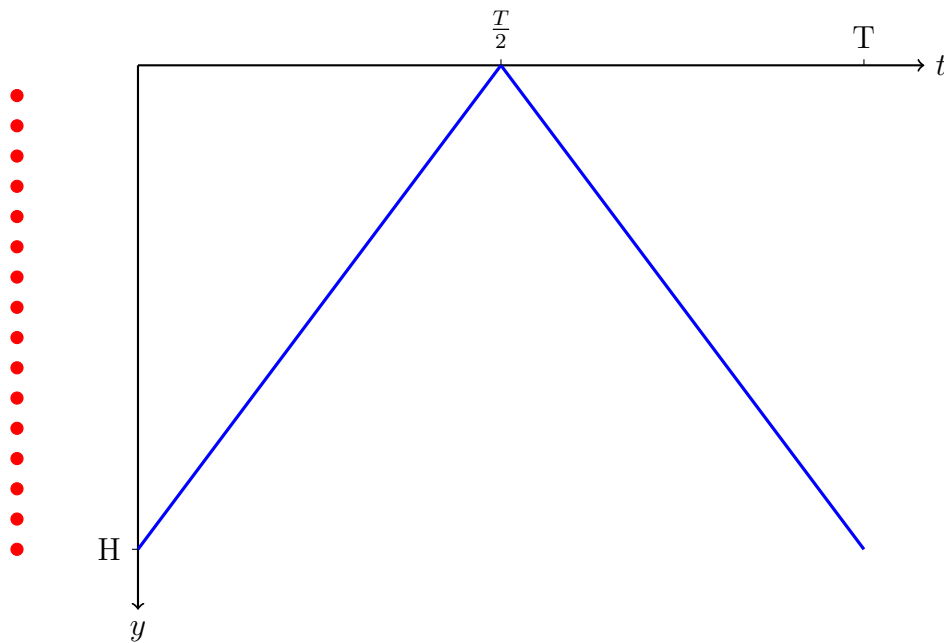
We could use this to represent a, very simple, motion of a ball going up and down.

If we keep repeating the interval, the ball would go up and don 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



$$\begin{cases} y = \frac{-2Ht}{T} + H & , \quad 0 < t \leq \frac{T}{2} \\ y = \frac{2Ht}{T} - H & , \quad \frac{T}{2} < x \leq T \end{cases}$$

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

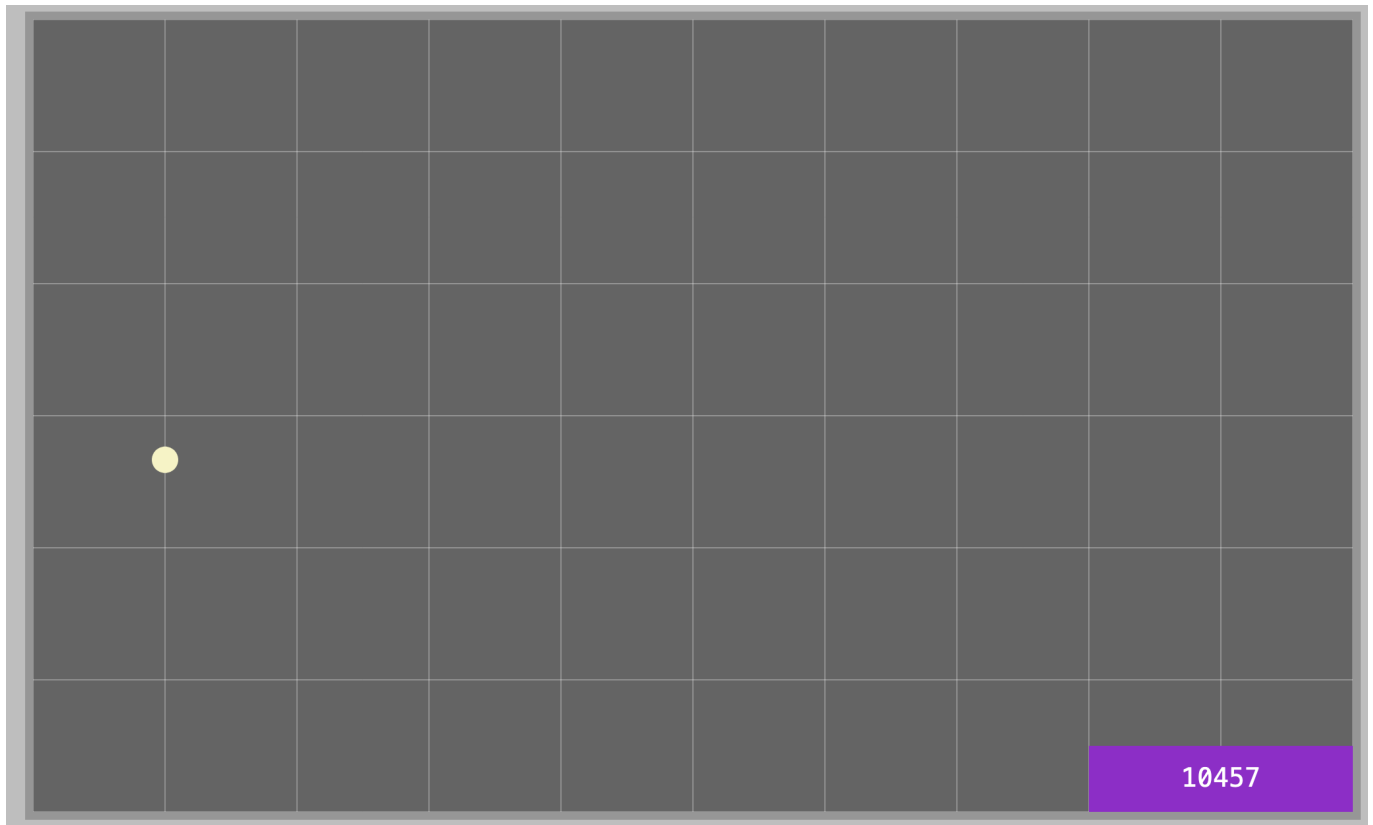
So we should be able to implement the functions given about and update the t value, for every tick of the animation frame.

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 flexible to have
35         parameters
36         // make sure t is between 0 and T
37         t = t%T; // modulus operator
38         // set y variable and use to get value from equations
39         let y;
40         if(t<T/2){
41             y = (-2*H*t)/(T) + H
42         }else{
43             y = (2*H*t)/(T) - H
44         }
45         return y
46     }
47 }
```

```
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     }
```

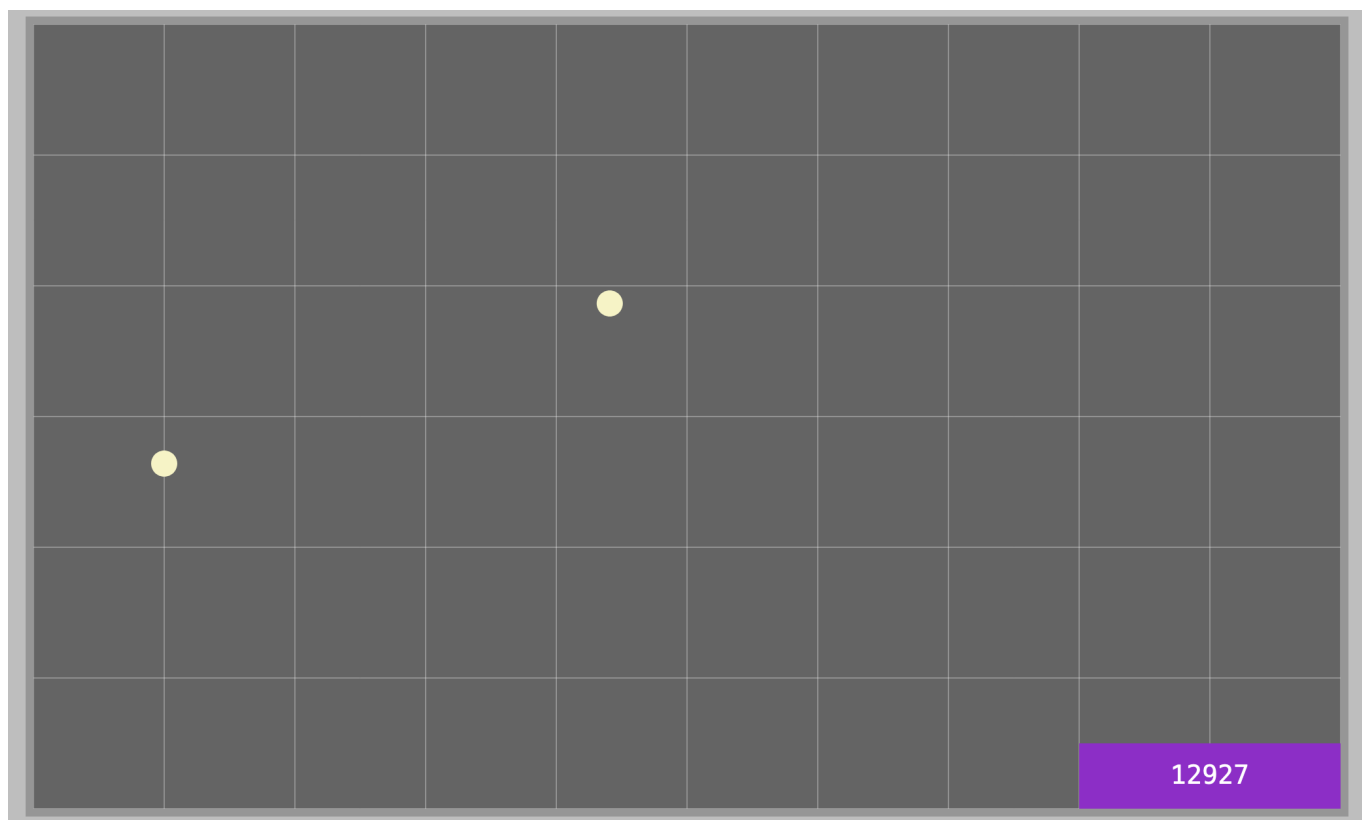
Listing 17: Moving Ball



We can introduce left right motion.

```
1  */
2  class MovingBallBoth{
3      constructor(x_b,y_b,r, fillcolour, T, H, xIS){
4          this.x_b = x_b;
5          this.y_b = y_b;
6          this.r = r;
7          this.fillColour = fillcolour;
8          // animation variables
9          // set a random starting point (while be helpful when we have lots of
moving balls)
10         this.t = T*Math.random();
11         this.T = T;
12         this.H = H;
13         // introduce an interval shift for the x interval
14         // this will make the ball behave more naturalistically
15         this.xIntervalShift = xIS
16     }
17     update(){
18         // add one to the value of little t each time update is called
19         this.t +=1
20         this.draw()
21     }
22     draw(){
23         // get y value from the piecewise function
24         let y = this.linearinterpolate(this.t, this.T, this.H)
25         // the interval is multiplied by the x interval shift
26         let x = this.linearinterpolate(this.t, this.T*this.xIntervalShift, this.H
27     )
        this.drawCircle(x+this.x_b,y+this.y_b, this.r)
```

Listing 18: Part of Bothways Moving Ball



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

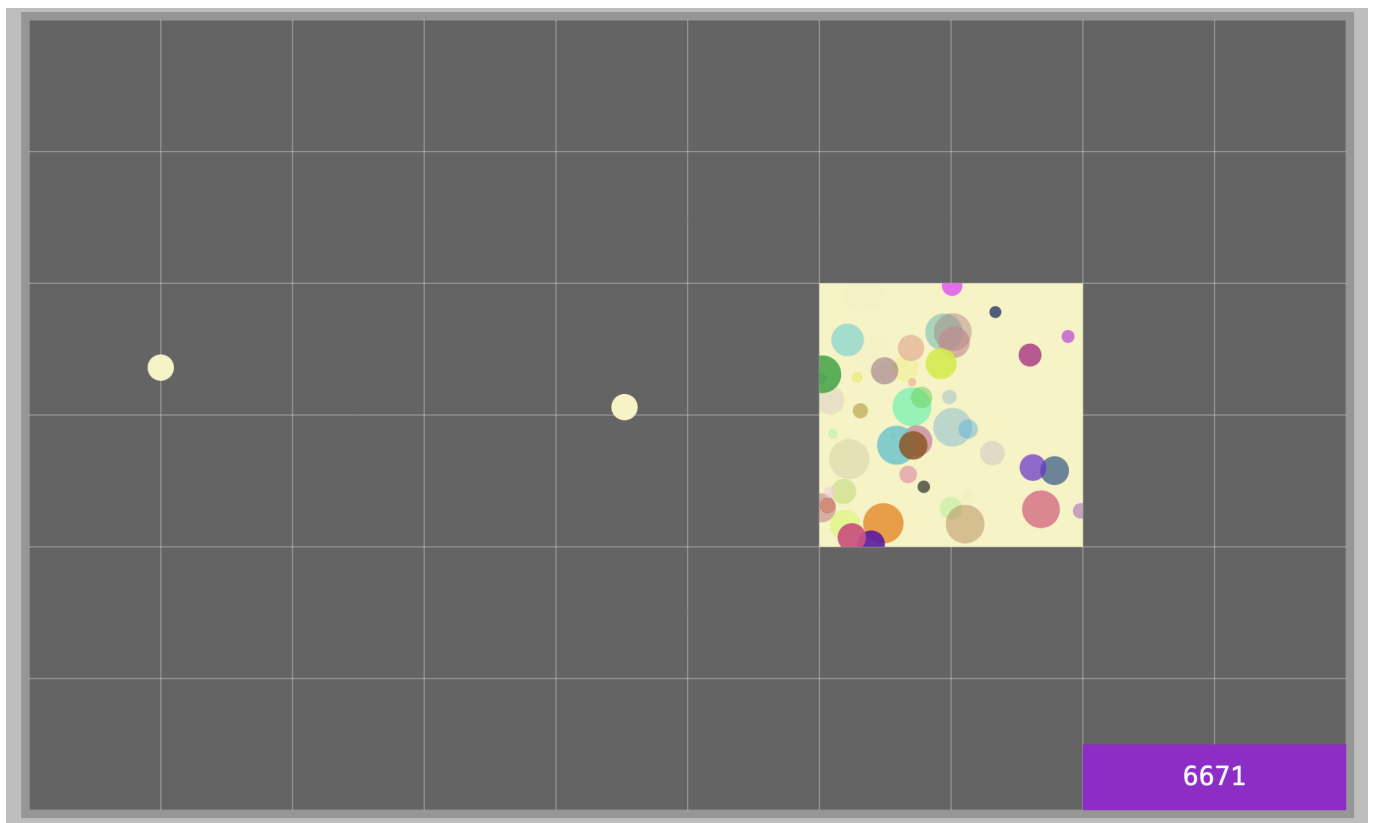
```
1  /**
2  * Ball that moves up down
3  * @param {number} x_b base x position
4  * @param {number} y_b base y position
5  * @param {number} radius radius
6  * @param {string} fillColour fill colour
7  * @param {number} T total Tick interval (50 ticks = about 1 second)
8  * @param {number} H total Height covered by up/down motion
9  */
10 class 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             +")"
33             // randomly adjust the radius size
34             let radius = r*7*Math.random()+r
35             // create a moving ball using these values
36             let temp = new MovingBallBoth(x_b,y_b,radius, randColour, randT, H,
37             xIS)
38             // push it into the BSet list
39             this.BSet.push(temp)
40         }
41     }
42     update(){
43         // this is an extra bit
44         // 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()

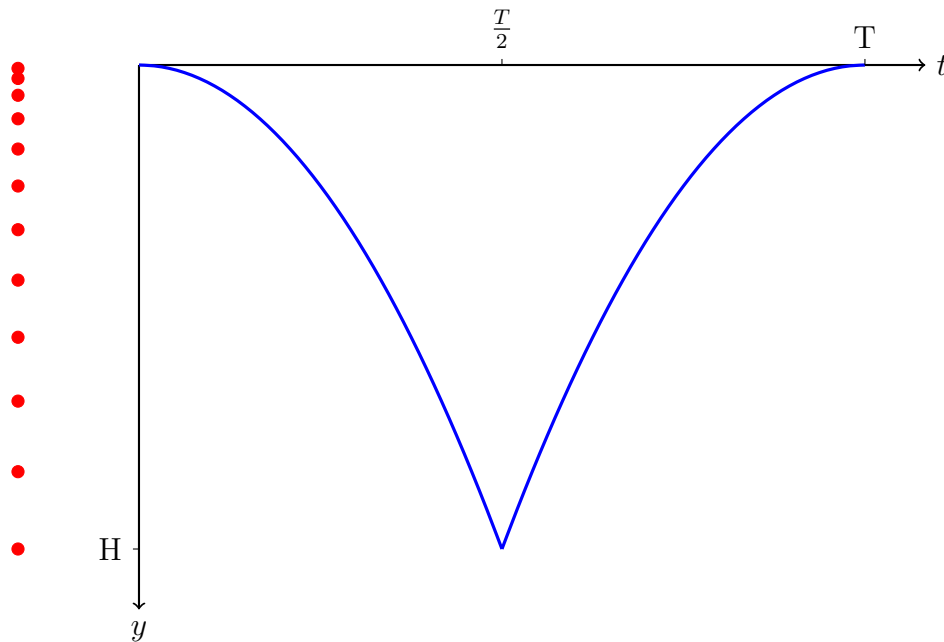
```

Listing 19: Field of moving balls



13.2 Quadratic Interpolation

We can follow the same idea using parabolas:



$$\begin{cases} y = \frac{4Ht^2}{T^2} & , \quad 0 < t \leq \frac{T}{2} \\ y = \frac{4H(t-T)^2}{T^2} & , \quad \frac{T}{2} < t \leq T \end{cases}$$

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?

```

1 class QuadraticBall extends MovingBallBoth{
2     draw(){
3         // get y value from the piecewise function
4         let y = this.quadraticInterpolate(this.t, this.T, this.H)
5         // the interval is multiplied by the x interval shift
6         let x = this.linearinterpolate(this.t + this.T*this.xIntervalShift/2, this
7         .T*this.xIntervalShift, this.H)
8         this.drawCircle(x+this.x_b,y+this.y_b, this.r)
9     }
10    quadraticInterpolate(t,T,H){
11        // takes parameter t , T, H
12        // we could hard code in this.T etc but is more flexible to have
13        parameters
14        // make sure t is between 0 and T
15        t = t%T; // modulus operator
16        // set y variable and use to get value from equations

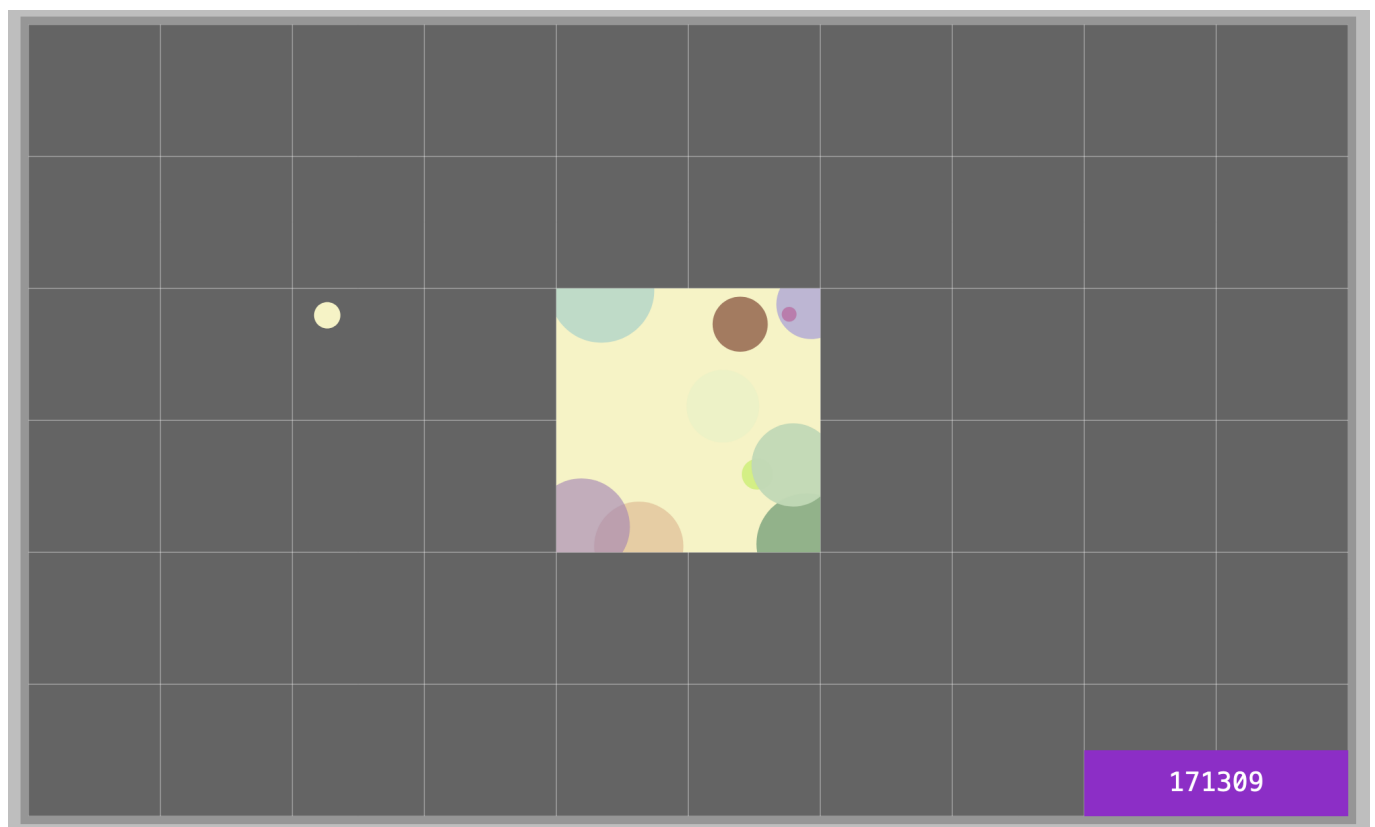
```

```

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 }

```

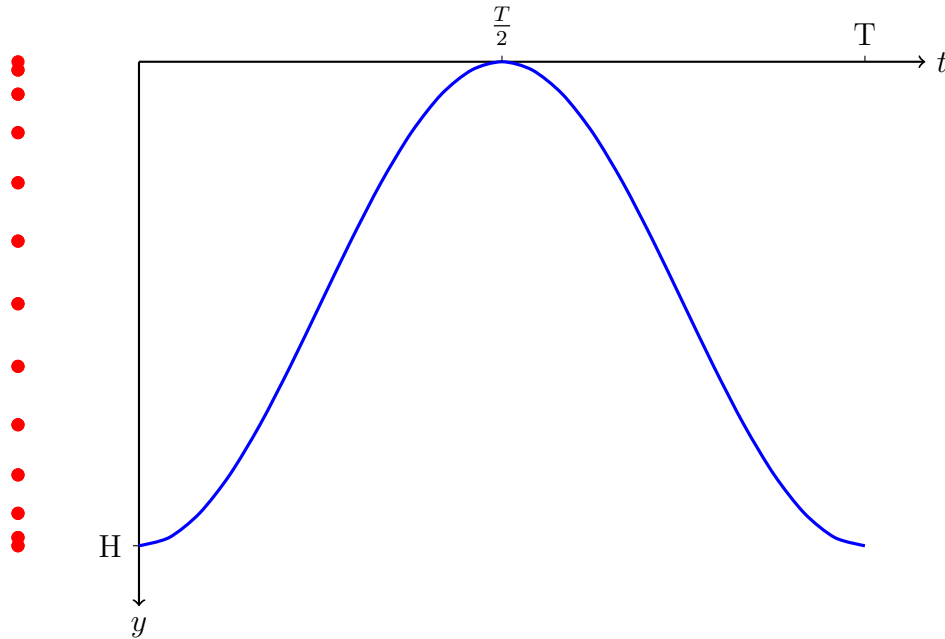
11

```
update() {
```

Listing 21: Quadratic Ball Class

13.3 Trigonometric Interpolation

We can follow the same idea using parabolas:



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

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
10   *   right movement
11  */
12  class TrigBall extends MovingBallBoth{
13      draw(){
14          // get y value from the piecewise function
15          let y = this.trigInterpolate(this.t, this.T, this.H)
16          // the interval is multiplied by the x interval shift
17          // see what happens with this commented out code
18          //let x = this.trigInterpolate(this.t + this.T/4, this.T, this.H)
19          let x = this.linearInterpolate(this.t + this.T*this.xIntervalShift/2, this
20          .T*this.xIntervalShift, this.H)

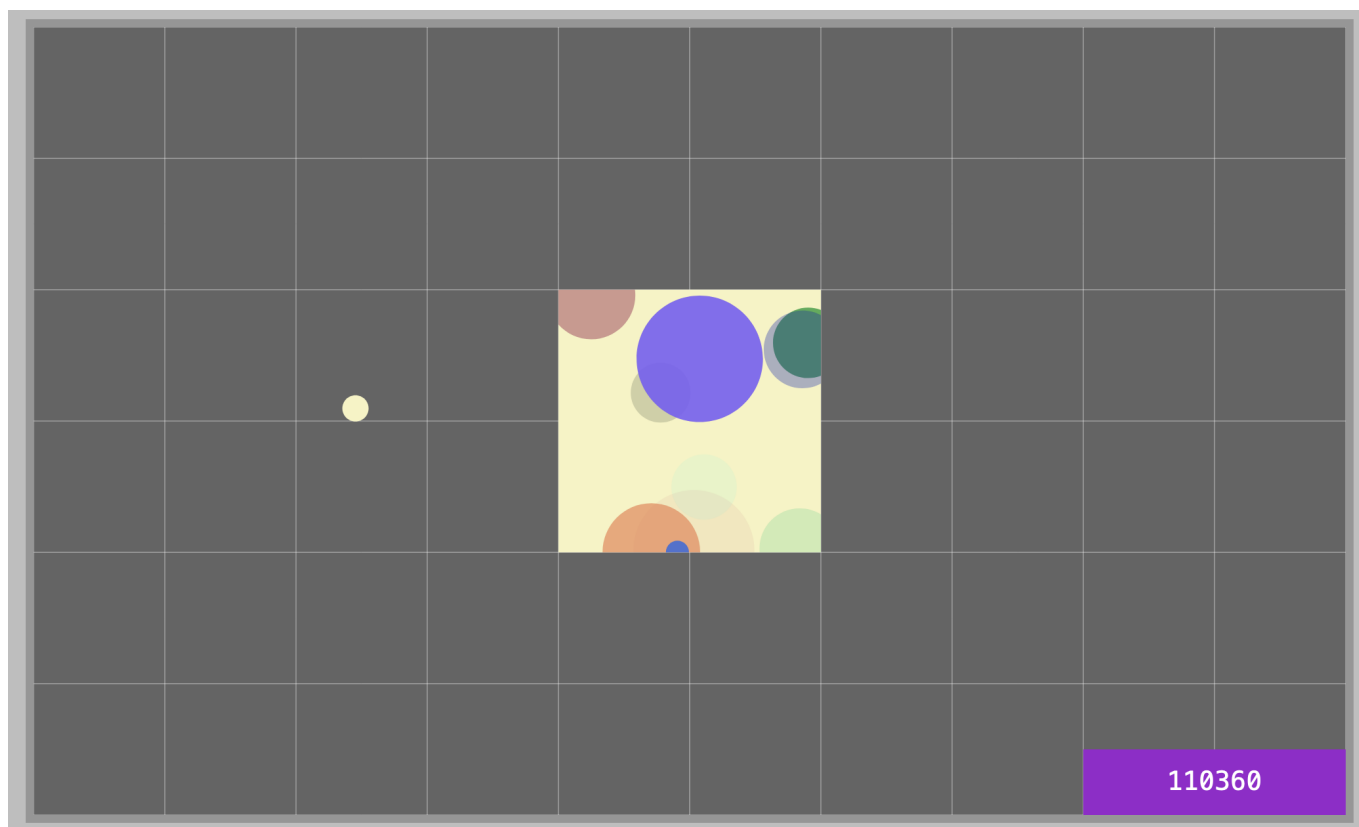
```

```

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

15 Basic Draggable Point

16 Better Draggable Point

```
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    //continually 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 }
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
71  *         (number)
72  * @return boolean
73  */
74 boundsCheck(x_1, y_1, x_2, y_2, r){
75     var d = Math.sqrt( Math.pow(x_2 - x_1, 2) + Math.pow(y_2 - y_1, 2) );
76     if(d<r){
77         return true;
78     }else{
79         return false;
80     }
81 }
82 /**
83  * Make x, y coordinates of point available outside of object
84  * @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 23: Python example

17 Buttons