12 Animation Frame

12.1 A bit of a code review - preparation

.

You do not need to have exactly this.

```
// init file with canvas set up
// colours
// functions
//...
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
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)";


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

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

Listing 13: init

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

Let's also start doc typing.

```
/**
 * Grid - draws a square grid of given interval width
 * across the whole canvas
 * @param {number} w width of canvas
 * @param {number} h height of canvas
 * @param {number} intervalWidth distance each grid unit
 * @param {string} strokeColour stroke colour
 * @param {number} strokeWidth  width of outline
 */
class Grid{
    constructor(w,h,intervalWidth, strokeColour, strokeWidth){
        this.w = w;
        this.h = h;
        this.intervalWidth=intervalWidth;
        this.strokeColour = strokeColour;
        this.strokeWidth = strokeWidth;
    }
    update(){
        this.draw()
    }
    draw(){
        // these loops also draw the grid outside (as is useful when analysing translations
        // and rotations (so your can ignore the negatives and use 0 instead
        // a loop for the vertical lines
        for(let i = -this.w ; i <= this.w ; i+= this.intervalWidth){
            this.drawLine(i,-this.h, i,this.h, this.strokeColour, this.strokeWidth);
        }
        // a loop for the horizontals
        for(let j = -this.h ; j <= this.h ; j+= this.intervalWidth){
            this.drawLine(-this.w,j, this.w,j, this.strokeColour, this.strokeWidth);
        }
```

```
    }
    drawLine(x_1,y_1, x_2, y_2, strokeColour,strokeWidth){
        ctx.beginPath();
        ctx.moveTo(x_1,y_1);
        ctx.lineTo(x_2,y_2);
        ctx.lineCap = "round";
        ctx.strokeStyle = strokeColour;
        ctx.lineWidth = strokeWidth;
        ctx .stroke()
    }
}

/**
 * A little textbox (text on coloured rectangle)
 * @param {number} x top corner of bounding box
 * @param {number} y top corner of bounding box
 * @param {number} w width
 * @param {string} txt text
 * @param {string} fill fill colour
 * @param {string} txtColour colour of text
 */
class TextBox{
    constructor(x,y,width, fillColour, txtColour) {
        this.x = x;
        this.y = y;
        this.w = width;
        // fixed height
        this.h = 50;
        // text managed through update
        this.txt = "Placeholder";
        console.log(this.txt)
        this.fillColour = fillColour;
        this.txtColour = txtColour;
    }
    // the text can be changed using the update function
    update(txt ="Placeholder"){
        this.txt = txt
        this.draw()
    }
    draw(){
        ctx.beginPath();
        ctx.rect(this.x,this.y,this.w,this.h);
        ctx.fillStyle= this.fillColour;
        ctx.fill();
        ctx.font = "20px monospace";
        ctx.textAlign = "center";
        ctx.textBaseline = "middle";
```

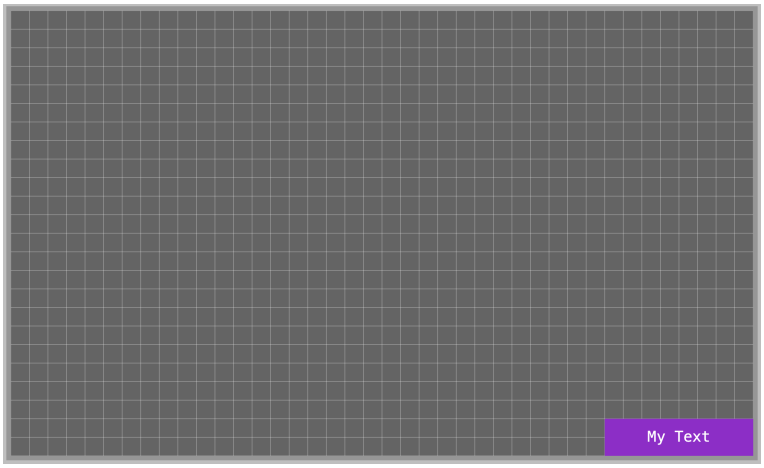
```
        ctx.fillStyle = this.txtColour;
        ctx.fillText(this.txt, this.x+this.w/2, this.y+this.h/2);
    }
}
```

Listing 14: objects

```
<!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>Objects Start</title>
</head>
<body>
<canvas id='myCanvas'></canvas>
<script type="text/javascript" src="init.js"> </script>
<script type="text/javascript" src="objects.js"> </script>
<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])
  G.update();
  // note that the text is set through the update function call
  T.update("My Text");

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

Listing 15: index (note text being set using update function call)

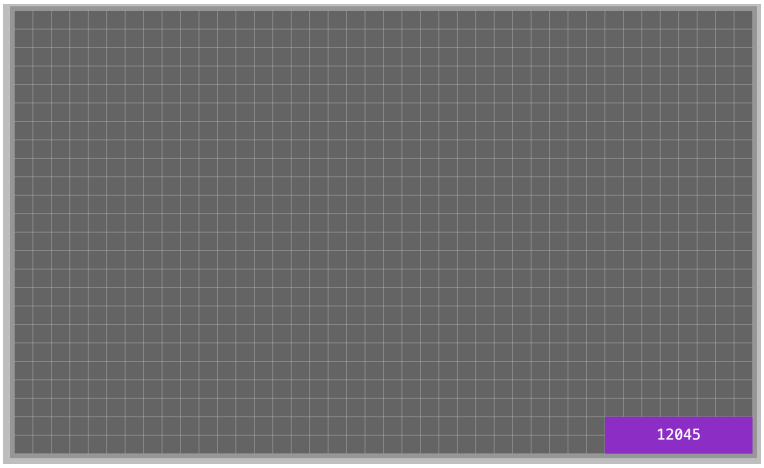


12.2 Introduce the animation frame

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

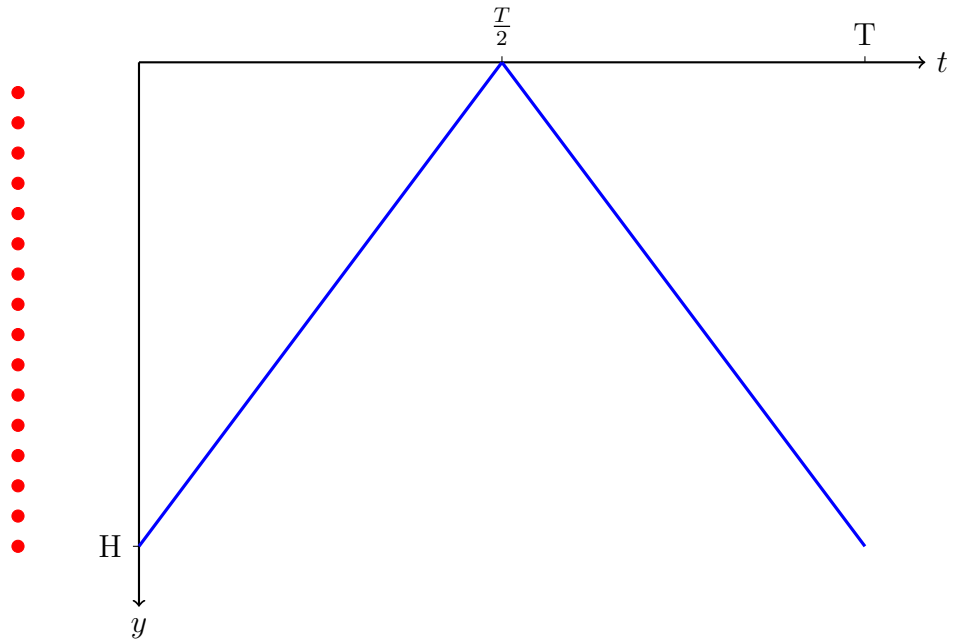


13 Animation

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



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

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.

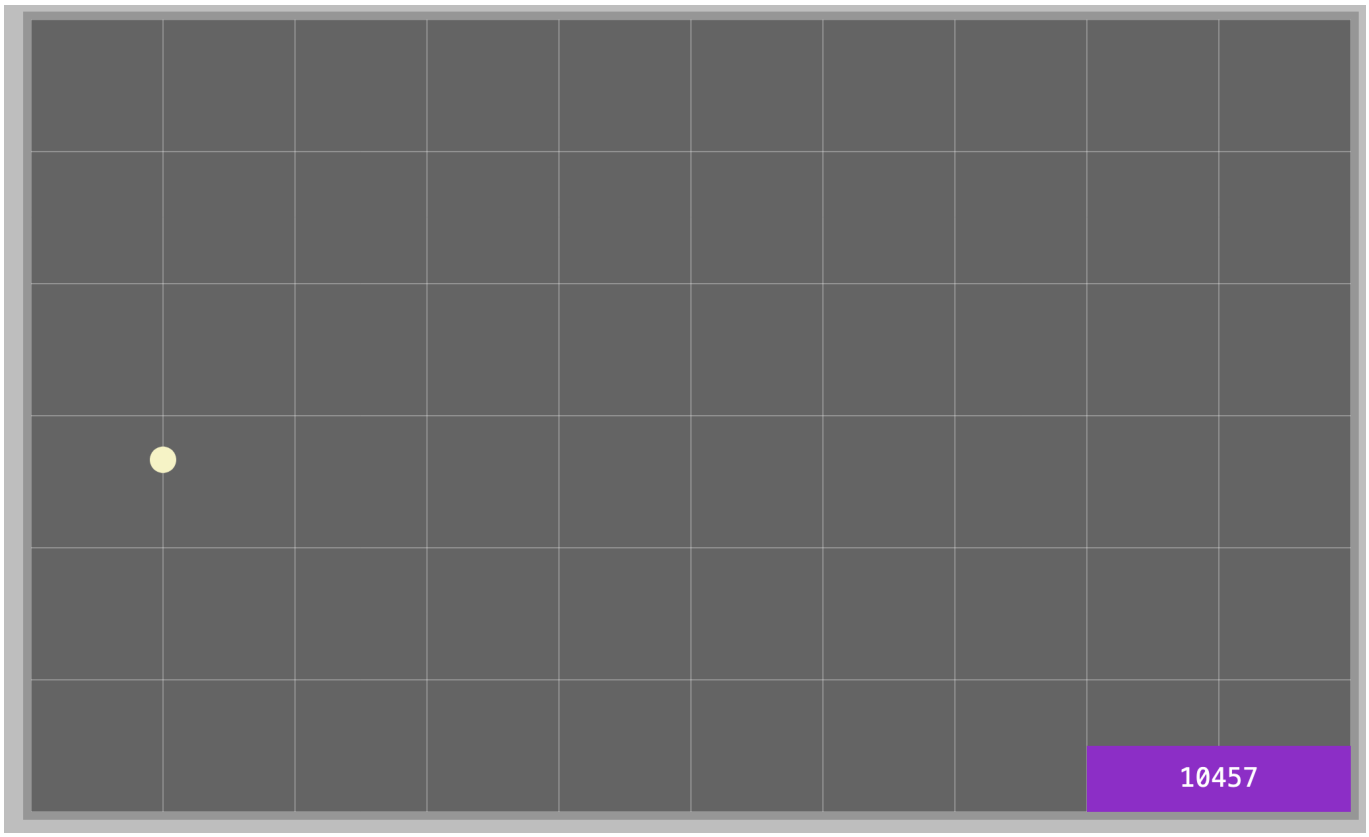
Moving Ball Object

```
/**
 * Ball that moves up down
 * @param {number} x_b base x position
 * @param {number} y_b base y position
 * @param {number} radius radius
 * @param {string} fillColour fill colour
 * @param {number} T total Tick interval (50 ticks = about 1 second)
 * @param {number} H total Height covered by up/down motion
 */
class MovingBall{
    constructor(x_b,y_b,r, fillcolour, T, H){
        this.x_b = x_b;
        this.y_b = y_b;
        this.r = r;
        this.fillColour = fillcolour;
        // animation variables
        this.t = 0;
        this.T = T;
        this.H = H;
    }
    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)
        this.drawCircle(this.x_b,y+this.y_b, this.r)
    }

    linearinterpolate(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
        let y;
        if(t<T/2){
            y = (-2*H*t)/(T) + H
        }else{
            y = (2*H*t)/(T) - H
        }
        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      }
```

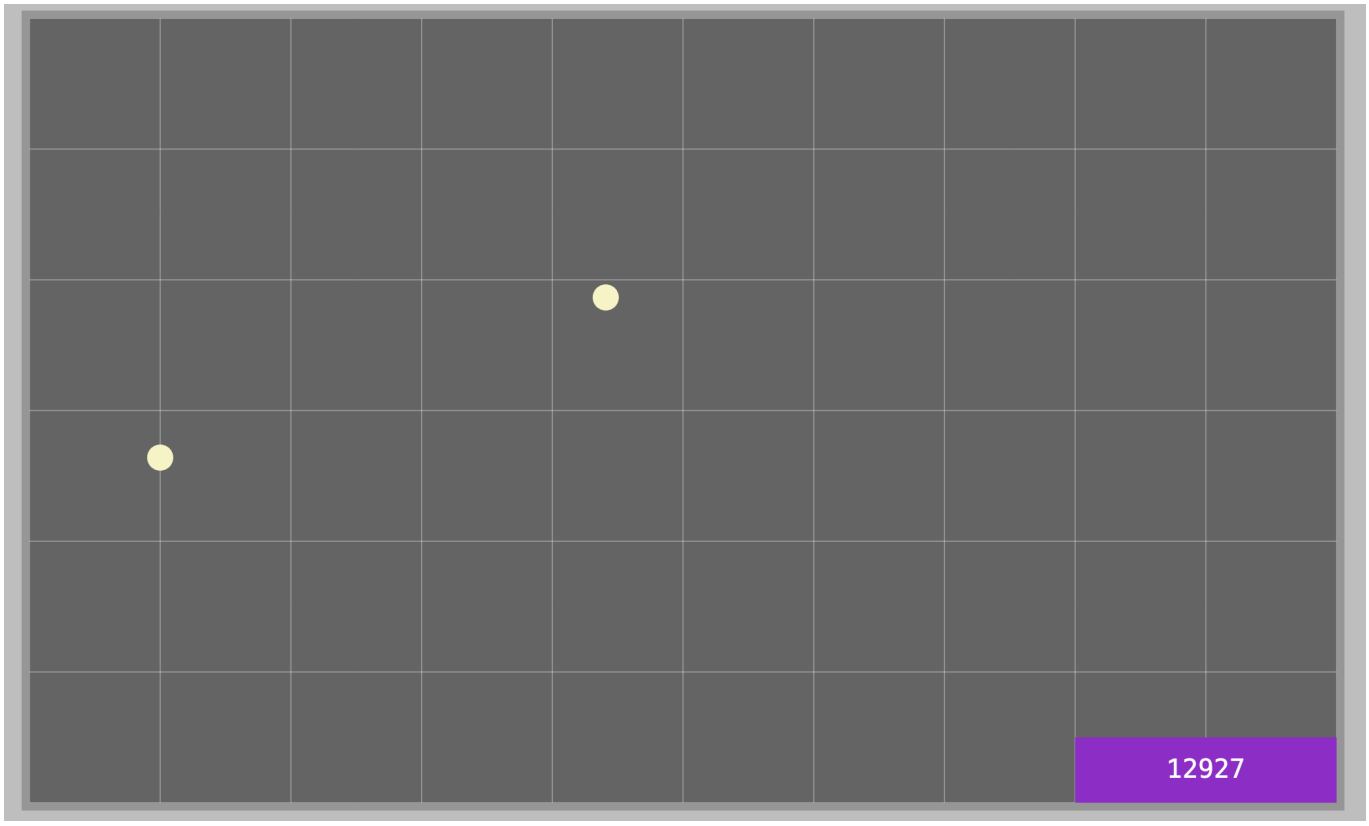
Listing 17: Moving Ball



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

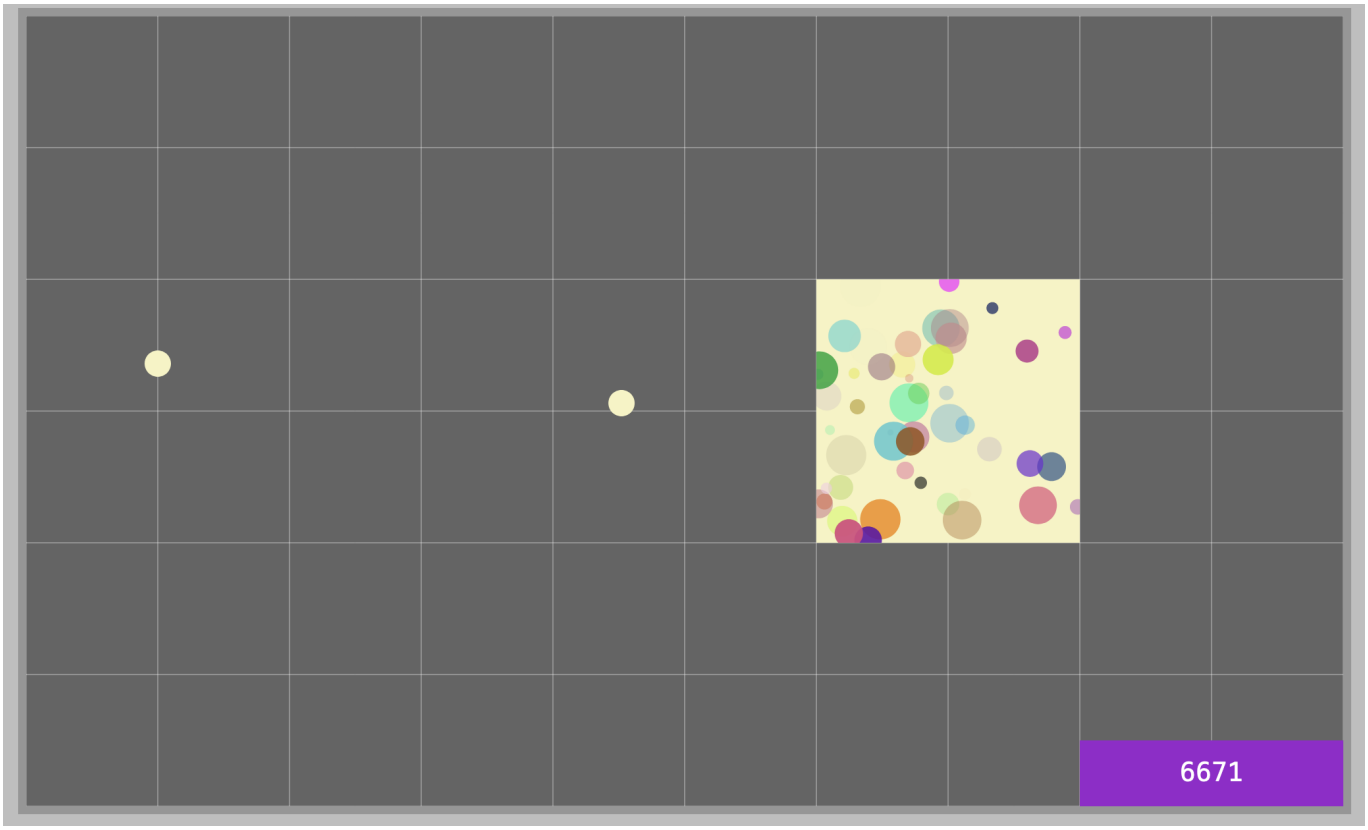

12927

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

```
/**
 * Block of moving balls
 * @param {number} x_b base x position
 * @param {number} y_b base y position
 * @param {number} radius radius
 * @param {string} fillColour fill colour
 * @param {number} T total Tick interval (50 ticks = about 1 second)
 * @param {number} H total Height covered by up/down motion
 */
class BallGroup{
    constructor(x_b,y_b,r, fillColour, T, H){
        this.x_b = x_b
        this.y_b = y_b
        this.H = H
        this.fillColour = fillColour
        // create a list that is going to hold a whole set of ball objects
        this.BSet = []
        //run a loop (in this case x50)
        for(let i=0; i<50; i++){
            // randomly set the amount of x interval shift
            let xIS = 1+5*Math.random()
            // randomly adjust the interval
            let randT = T + 2*T*Math.random()
            // create random red, green, blue
            let red = 255*Math.random();
            let green = 255*Math.random();
            let blue = 255*Math.random();
            // create random transparency
            let alpha = Math.random();
            // concatenate to make a rgb string
            let randColour = "rgba("+ red + ","+ green + "," + blue + "," + alpha
    +")"
            // randomly adjust the radius size
            let radius = r*7*Math.random()+r
            // create a moving ball using these values
            let temp = new MovingBallBoth(x_b,y_b,radius, randColour, randT, H,
   xIS)
            // push it into the BSet list
            this.BSet.push(temp)
        }
    }
    update(){
        // this is an extra bit
        // save the canvas
```

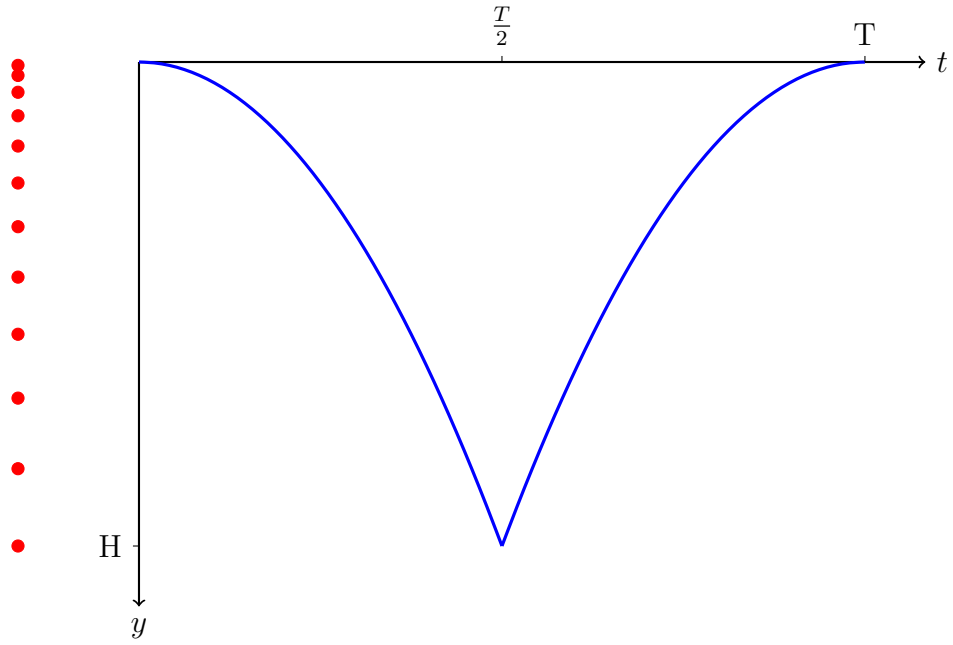
```
        ctx.save()
        // draw and fill a background rectangle
        this.drawRect(this.x_b, this.y_b, this.H, this.H)
        // the clip method will "clip out" anything outside the rectangle
        ctx.clip()
        // run a loop through the BSet
        for(let i=0; i<this.BSet.length ; i++){
            // call update on each moving ball
            this.BSet[i].update()
        }
        // restore the context (this removes the clip)
        ctx.restore()
```

Listing 19: Field of moving balls



13.2 Quadratic Interpolation

We can follow the same idea using parabolas:



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

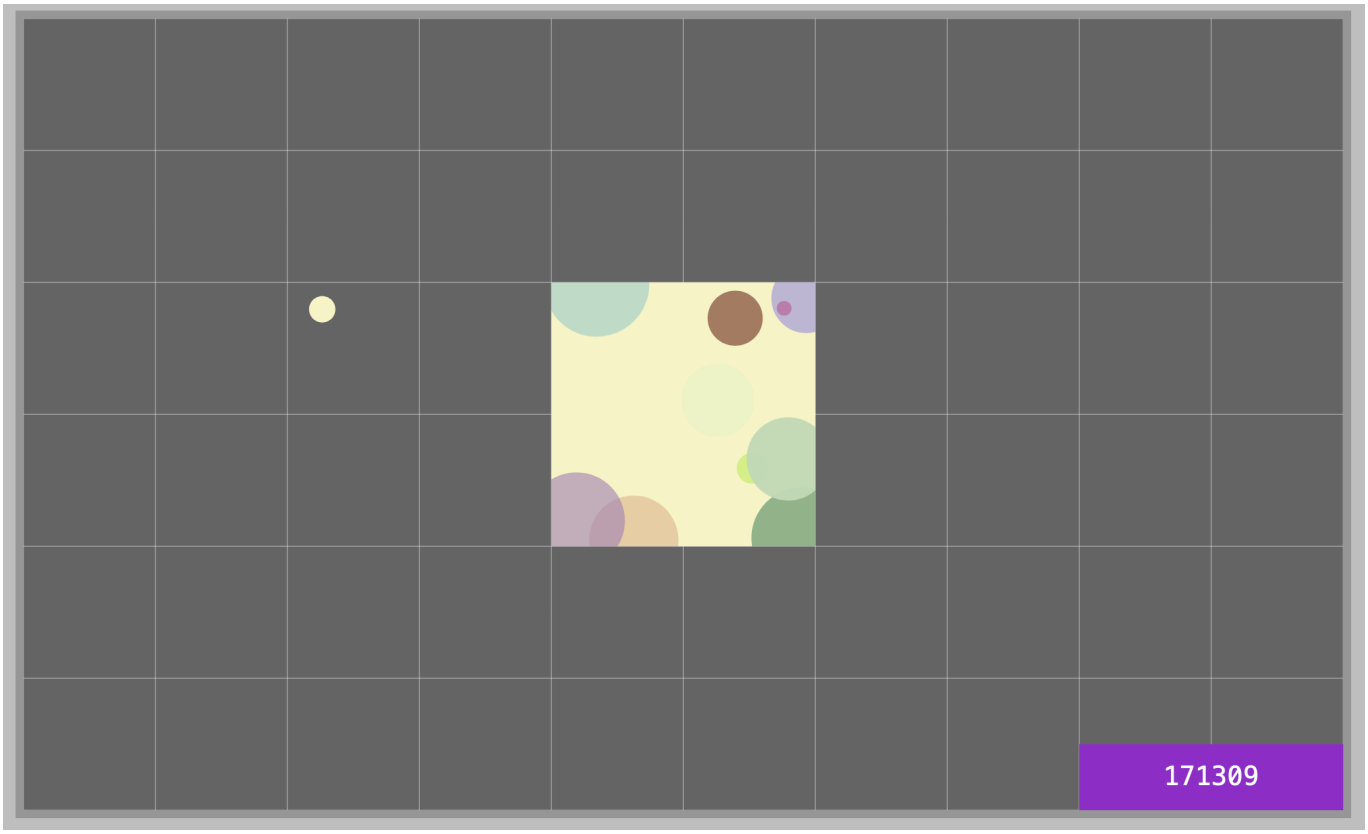
In this case, we would have the up down motion as quadratic, and the the left right as linear.

Below is the code for a Quadratic Ball Class.
Notice how short it is. What's going on here?

```
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
```

```
        let y;
        if(t<T/2){
            y = ( 4*H*Math.pow(t,2) )/( Math.pow(T,2) )
        }else{
            y = ( 4*H*Math.pow(t-T,2) )/( Math.pow(T,2) )
        }
        return y
    }
}
```

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.

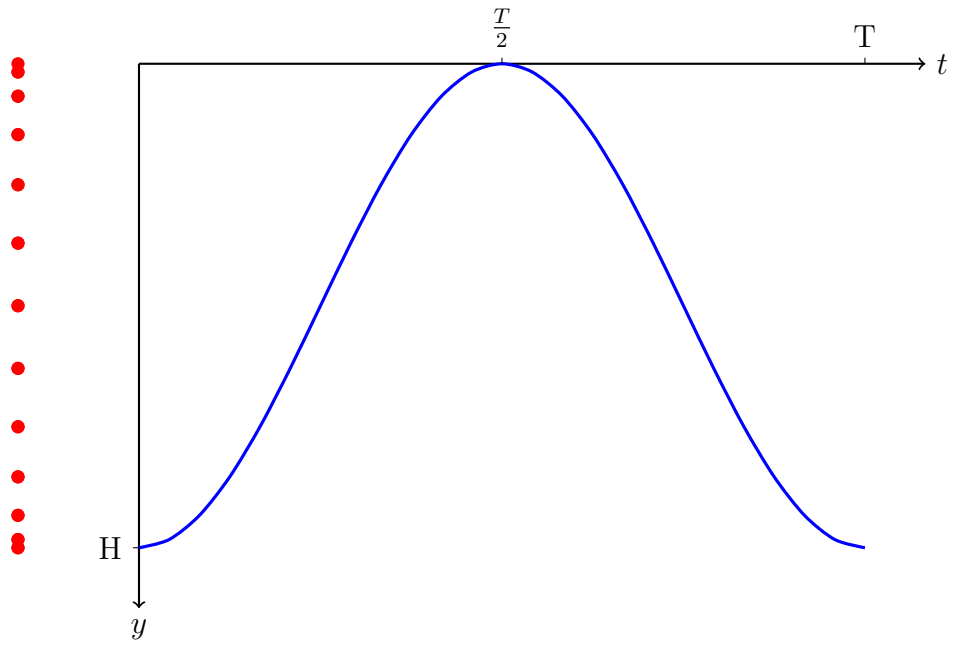
```
            let radius = r*10*Math.random()+r
            // create a moving ball using these values
            let temp = this.getObject(x_b,y_b,radius, randColour, randT, H, xIS)
            // push it into the BSet list
            this.BSet.push(temp)
        }
    }
    getObject(x_b,y_b,radius, randColour, randT, H, xIS){
        return  new QuadraticBall(x_b,y_b,radius, randColour, randT, H, xIS)
    }
```

```
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).

```
/**
 * Ball that moves trigonometrically
 * @param {number} x_b base x position
 * @param {number} y_b base y position
 * @param {number} radius radius
 * @param {string} fillColour fill colour
 * @param {number} T total Tick interval (50 ticks = about 1 second)
 * @param {number} H total Height covered by up/down motion
 * @param {number} xIS x interval shift (to make a different interval for left
   right movement
 */
class TrigBall extends MovingBallBoth{
    draw(){
        // get y value from the piecewise function
        let y = this.trigInterpolate(this.t, this.T, this.H)
        // the interval is multiplied by the x interval shift
        // see what happens with this commented out code
        //let x = this.trigInterpolate(this.t + this.T/4, this.T, this.H)
        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)
    }
    trigInterpolate(t,T,H){
        let y = (H/2) * Math.cos( (2*Math.PI/T) *t) + H/2
        return y
    }
}
/**
 *Group of trigonometrically moving balls
 * @param {number} x_b base x position
 * @param {number} y_b base y position
 * @param {number} radius radius
 * @param {string} fillColour fill colour
 * @param {number} T total Tick interval (50 ticks = about 1 second)
 * @param {number} H total Height covered by up/down motion
 */
class TrigBallGroup extends QuadraticBallGroup{
    getObject(x_b,y_b,radius, randColour, randT, H, xIS){
        return new TrigBall(x_b,y_b,radius, randColour, randT, H, xIS)
    }
}
```

Listing 22: Quadratic Ball Class