Algorithmic Art Workbook #6 Particles and Perlin

Table of Contents

MIT Licence

Workbook #6 Quick Content Summary

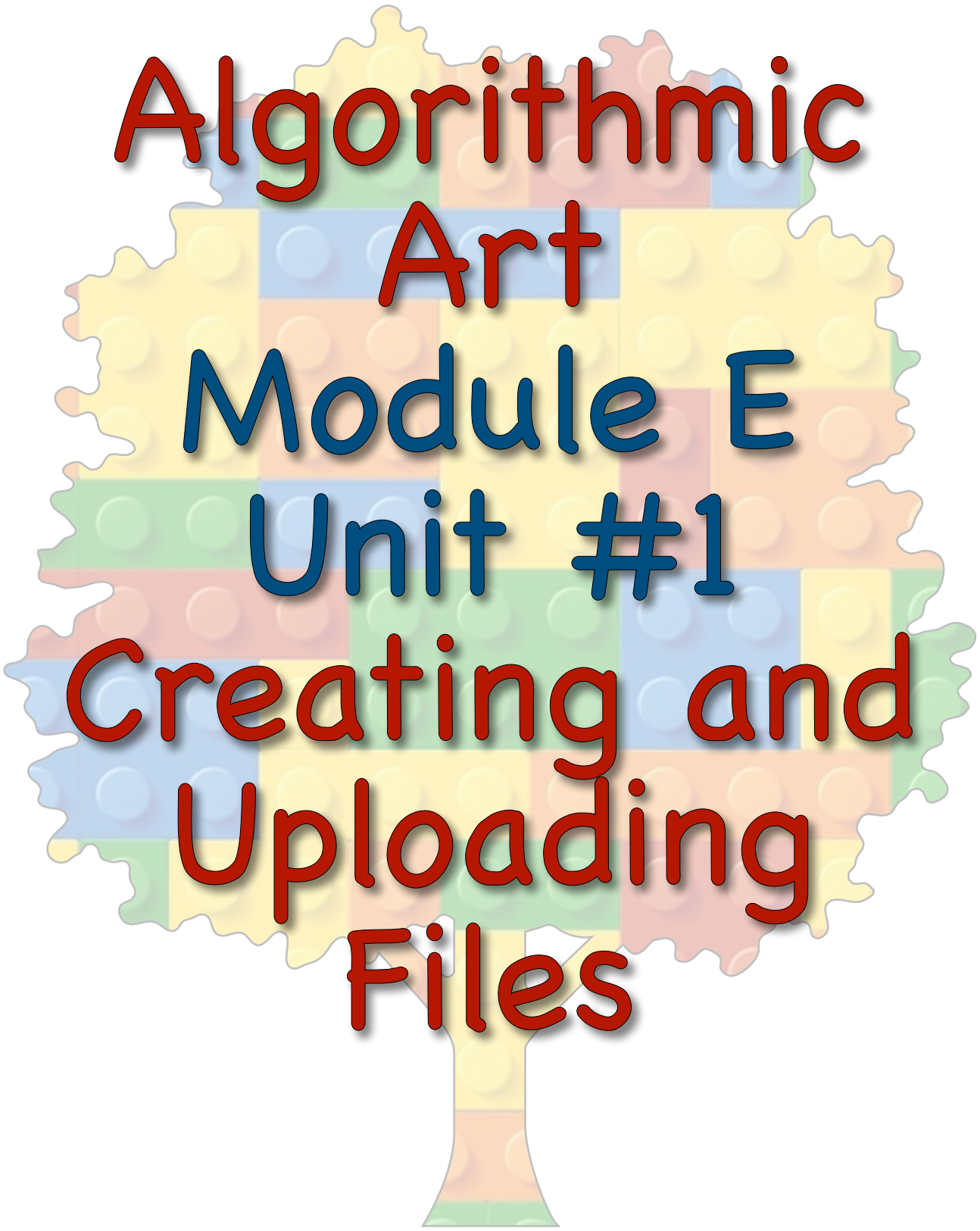
One of the ways you can develop very interesting effects is to use particles which have a particular property. There are various forms of random, and Perlin Noise is one that can be used to great effect.

With p5.js, you can upload files whether they are images, music, videos, fonts, or even 3D objects. You can also break down your code into more manageable chunks; for instance, if you create a class, then you might put that in a separate file (think of it as a folder or tab). This keeps your code very tidy rather than one long column of lines of code. It does also mean you can find the important bits quickly.

The only slight downside is that you have to come out of one file and jump into another. I have gone out of my way to keep it to a minimum but where we have moved from one file to another I have made it very clear. It is just one of those trade offs. If you don't want to do that then you are welcome to keep everything in the main sketch, it will still work. It will also keep this workbook a lot shorter. Swings and roundabouts I guess.

The code is in the yellow boxes, any new code is highlighted in blue, so you don't have to type out the whole thing again each time. Set up an account (highly recommended) so you can save your work as you go along rather than starting again each time. It will also remind you where you were up to.

Most browsers work but Chrome is best as there is some functionality missing in some browsers. I suspect it has been tested on Chrome to guarantee it works in Chrome browser.



Algorithmic Art
Module E
Unit #1
Creating and Uploading Files

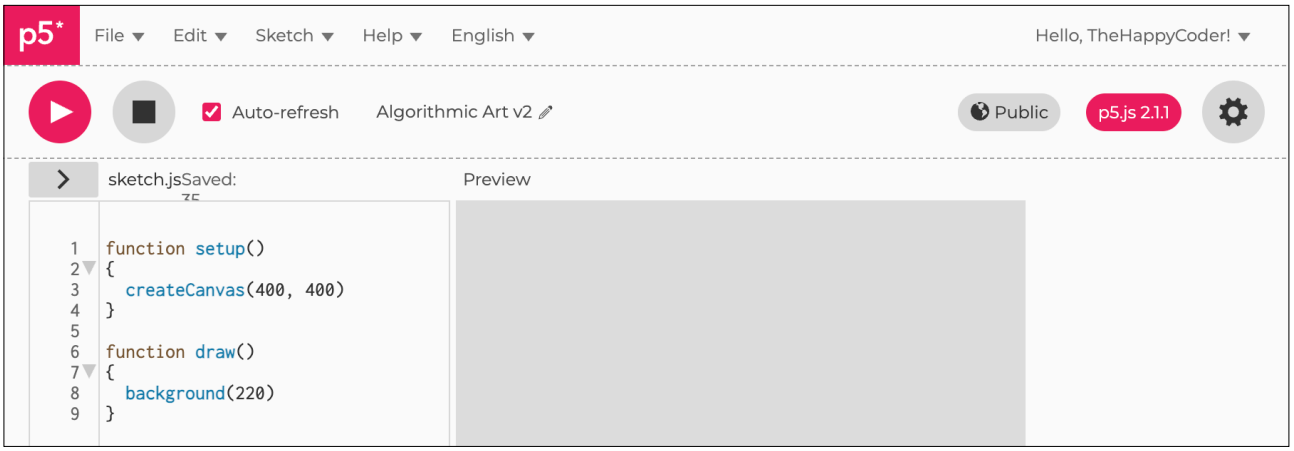
Module E Unit #1: Creating and Uploading Files

There aren't any sketches in this unit, just an explanation concerning the file system and the console. It is quite intuitive and straightforward but also needs practise. So if it goes horribly wrong, try, try again.

We will be mainly concerned with creating files. They help to keep the amount of one code in one place to a minimum by distributing across a few. This is particularly useful when using Object Orientated Programming with classes. You will see, otherwise you can have one very long sketch.

! This information is relevant to p5.js version 1.11.11 (latest default) but some time in 2026 (mid to late) a version 2.x will become the default. You can already use the new version.

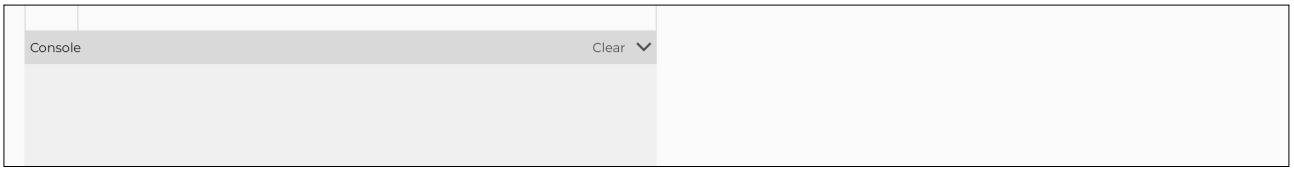
Version 2.2.2 (as of time of writing)
- just click on the button to select the version you want



The purpose of the console

The `console` is the grey boxed section underneath where you enter your code. This is a very useful function for a number of reasons. It is where you will get error messages and where you can send information. It's main purpose is for debugging which is another word for problem solving.

Figure 1: the console



Error messages

If you editor picks up on some glaring and obvious errors in your code which can be anything from a missed comma or semicolon to an unknown variable appearing. Mostly you will get something useful and informative although sometimes it just flags a problem and leaves you to search for it.

Sometimes the error code may be highlighted in red and it may even give you the line number to identify where the problem is (or the following line number where it does become a problem).

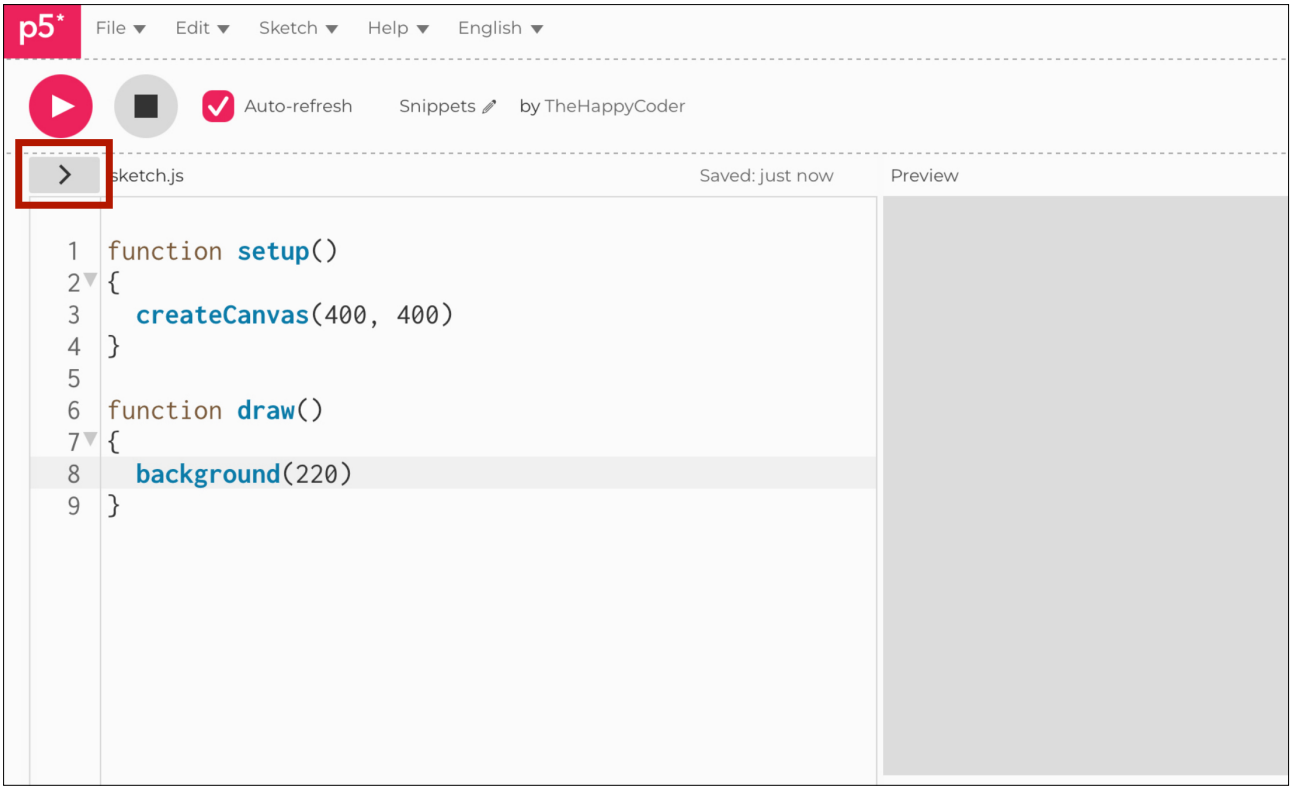
Console log

We can put a line of code in called `console.log()`. This means you are going to log something in the console. Sometimes it can be a piece of text such as `console.log('finished training')`, or it can be the value of a variable such as `console.log(counter)` which will give you the value of the variable at that time in the code. Another really helpful use is debugging a problem where you are not sure what is happening, for example, to an array.

Accessing the sketch files

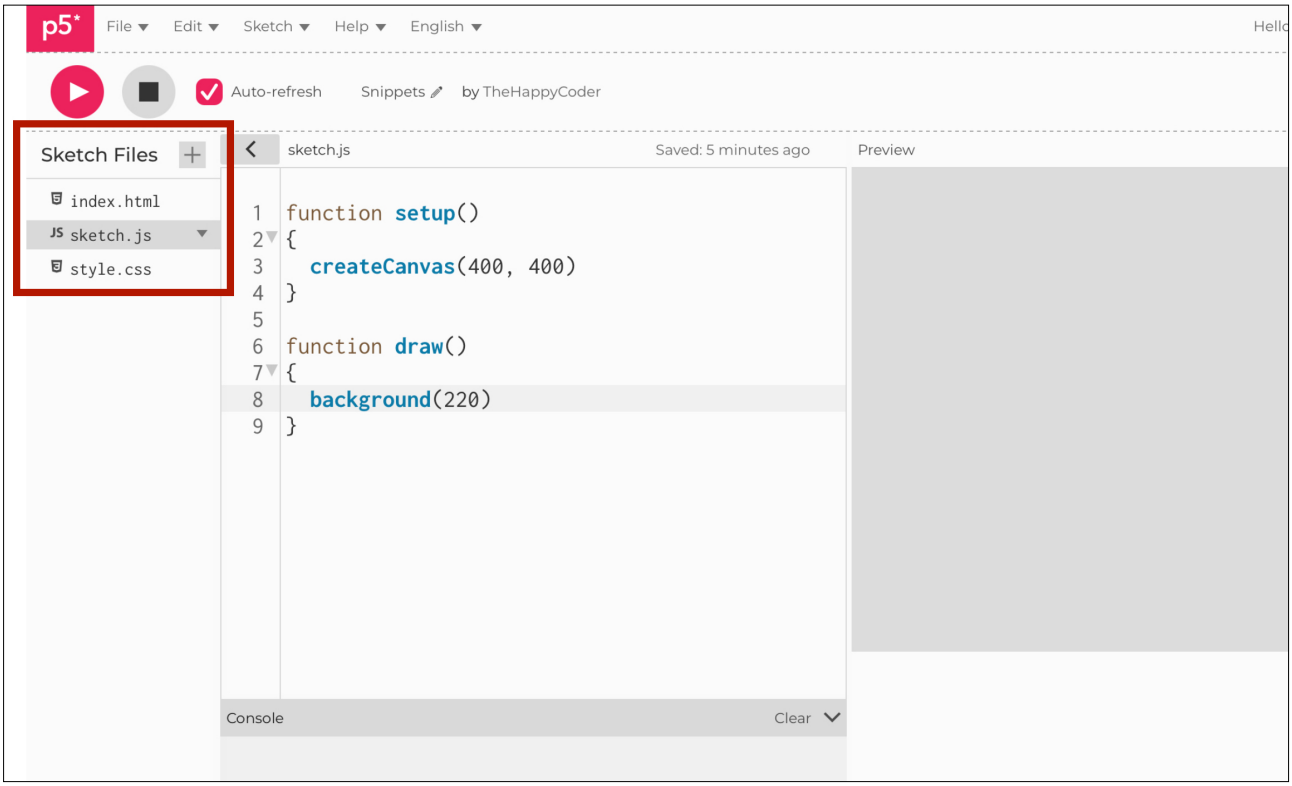
It is going to be critical to understand where the sketch files are and what files are used in creating a sketch. To the left of where you type your code you will find a grey box with a grey arrow. I have highlighted in red, click on it.

Figure 2: sketch files



When you click on it you will get a menu of `Sketch Files` associated with the editor. Those listed here are `index.html`, `sketch.js` and `style.css`. The main ones you will be interested in are the `sketch.js` and `index.html`. The style.css can be ignored (for now).

Figure 3: files menu

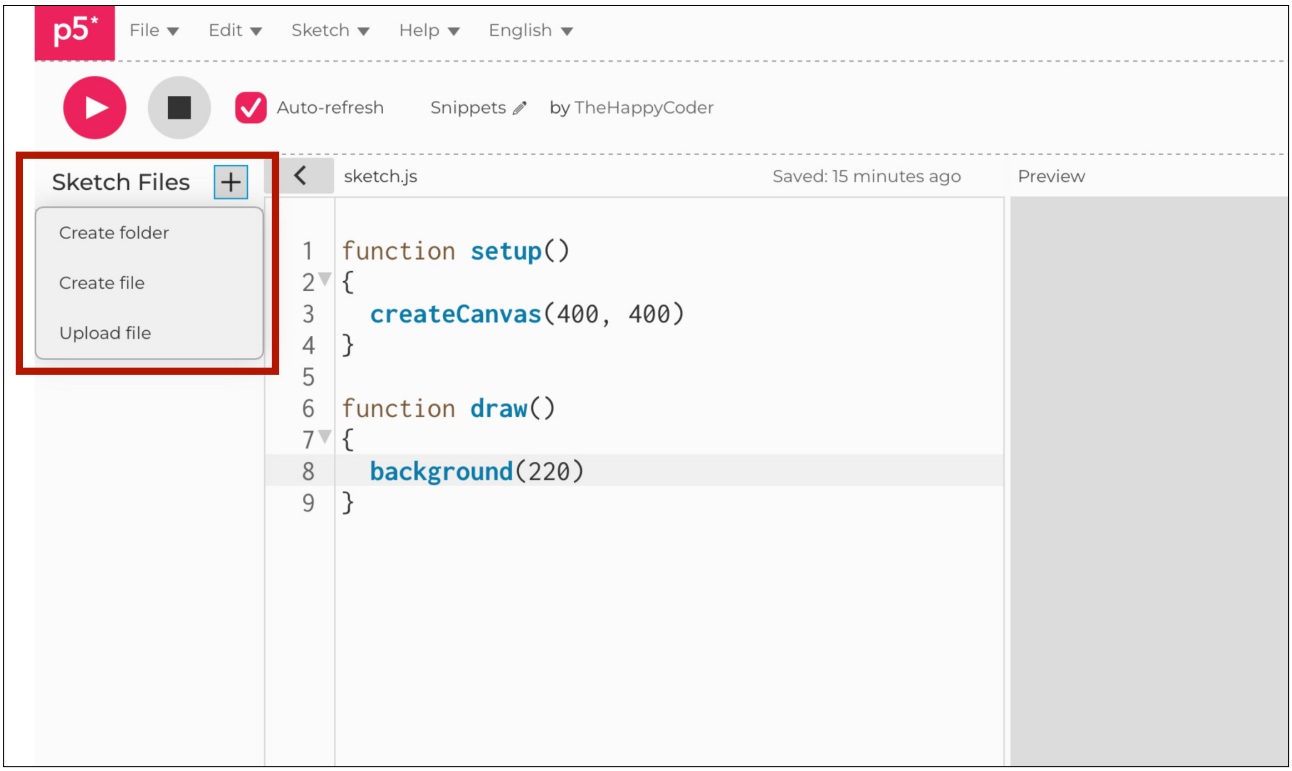


Adding Files

Although I have avoided adding extra files in the majority of the coding there are times when adding them makes things neater and, on the whole, easier to manage. You have the `sketch.js` by default as your main coding file but you may want to add more. You will need to give it a name and file extension `.js`, for example `particles.js`.

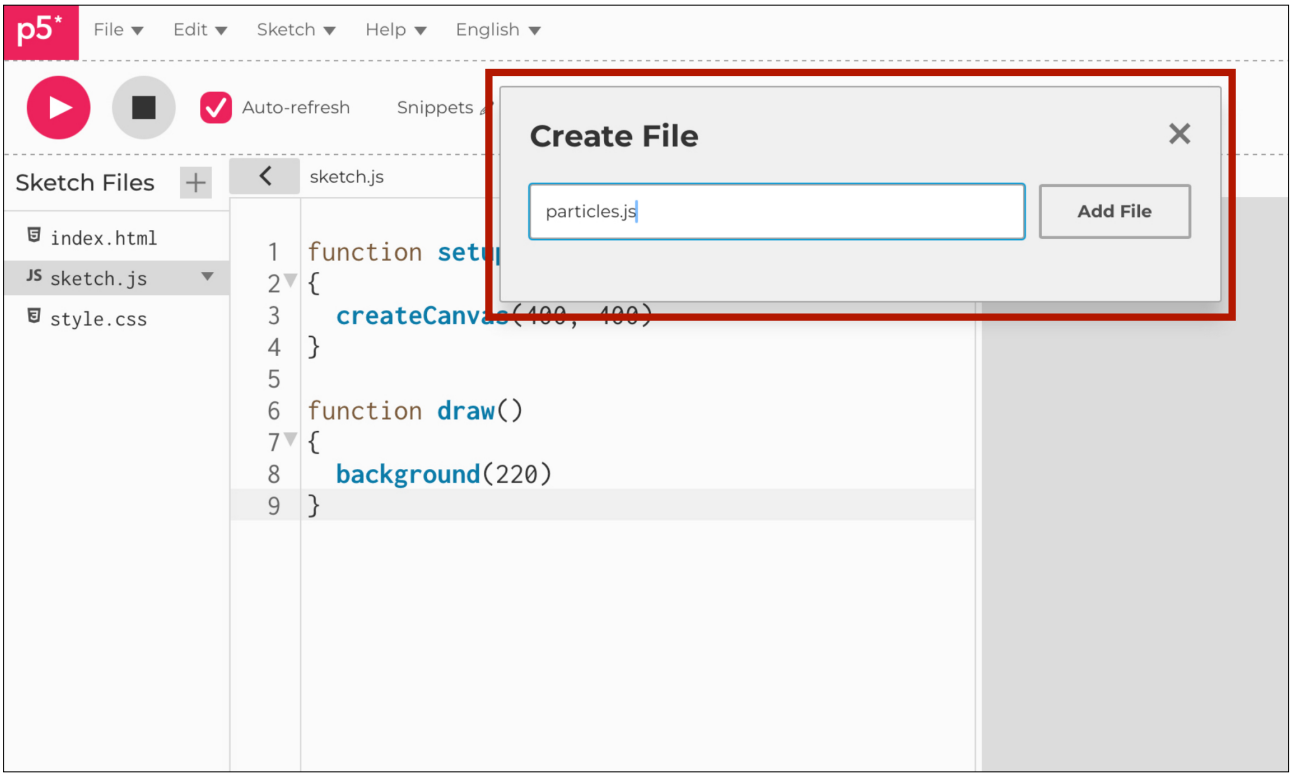
A) You add them by clicking on the grey `+` sign next to `Sketch File`.

Figure 4: adding files



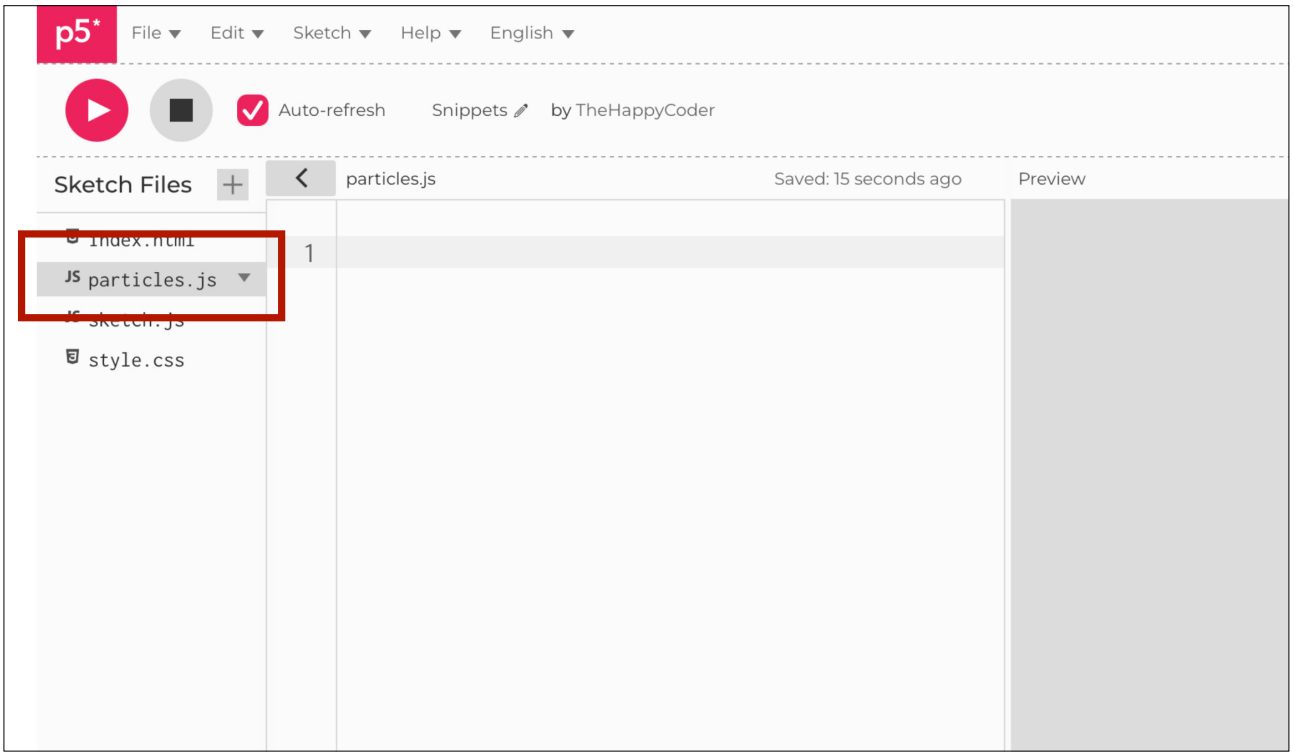
B) Now you click on `Create file`. You will get a box appearing waiting for you to put the name of the file in (important that it is case sensitive and must have the `.js` file extension at the end).

Figure 5: creating and naming files



C) Click on `Add File` and the file will appear in the `Sketch files`. Notice that the file is empty of any code. However, we aren't done yet, to use this new file we have to make reference to it in the `index.html` file.

Figure 6: adding the file

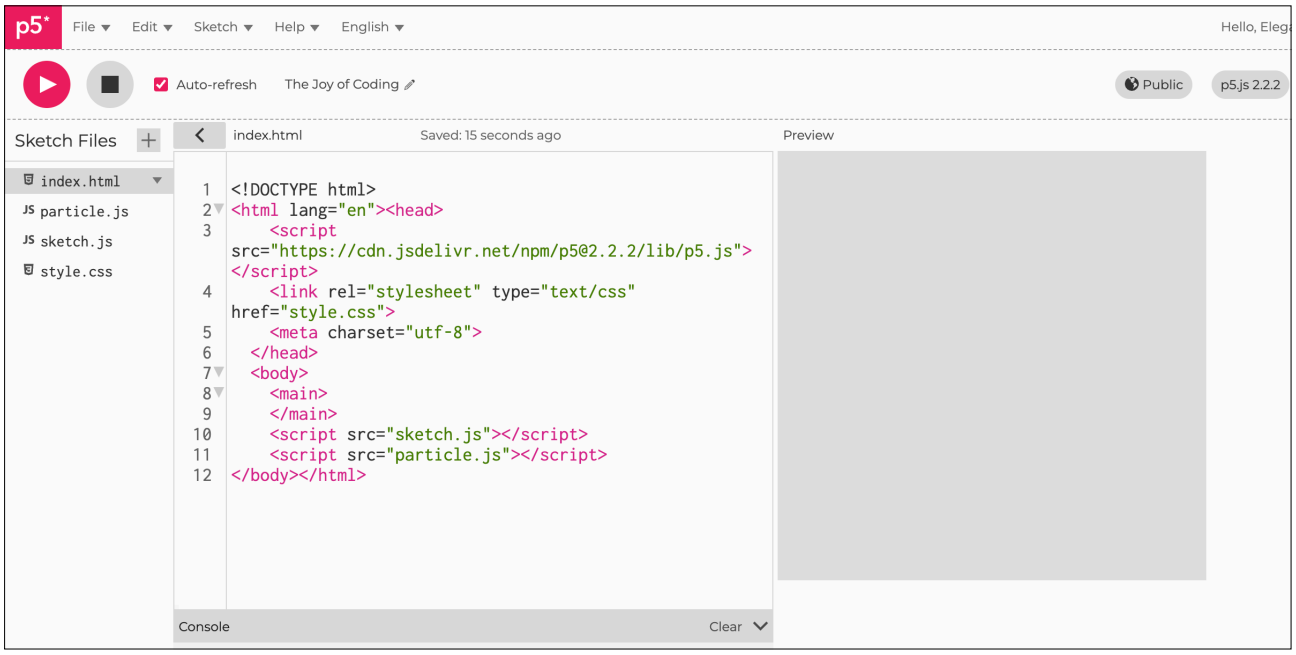


The index.html File

Although we are just adding another file we have created you will be adding other script tags which we use for links to `ml5.js` and `matter.js` etc, but for now I will show you how to add the file we have just created. This bit is often easy to forget to do and then you wonder why the code is throwing a wobbler at you.

D) This time we click on the `index.html` file to open a bewildering amount of code. Don't be put off if you are not familiar with html tags.

Figure 7: the index.html (showing version 2.x)



This is the code within the index.html file. The index file is the core of the editor because it is where the editor interacts with the web browser. HTML is a tag based coding language and is often used to create web sites. All this is in the web editor by default. We don't need to install the p5.js code instead we access the library through a link. Here is a brief explanation:

```
<script src="https://cdn.jsdelivr.net/npm/p5@1.11.13/lib/p5.js"></
script>
```

The default version is 1.11.13 (as of writing) but when you use version 2.x, this line will automatically update. You will also have a sound library (which will not be necessary in version 2.x)

```
<script src="https://cdn.jsdelivr.net/npm/p5@1.11.13/lib/addons/
p5.sound.min.js">
```

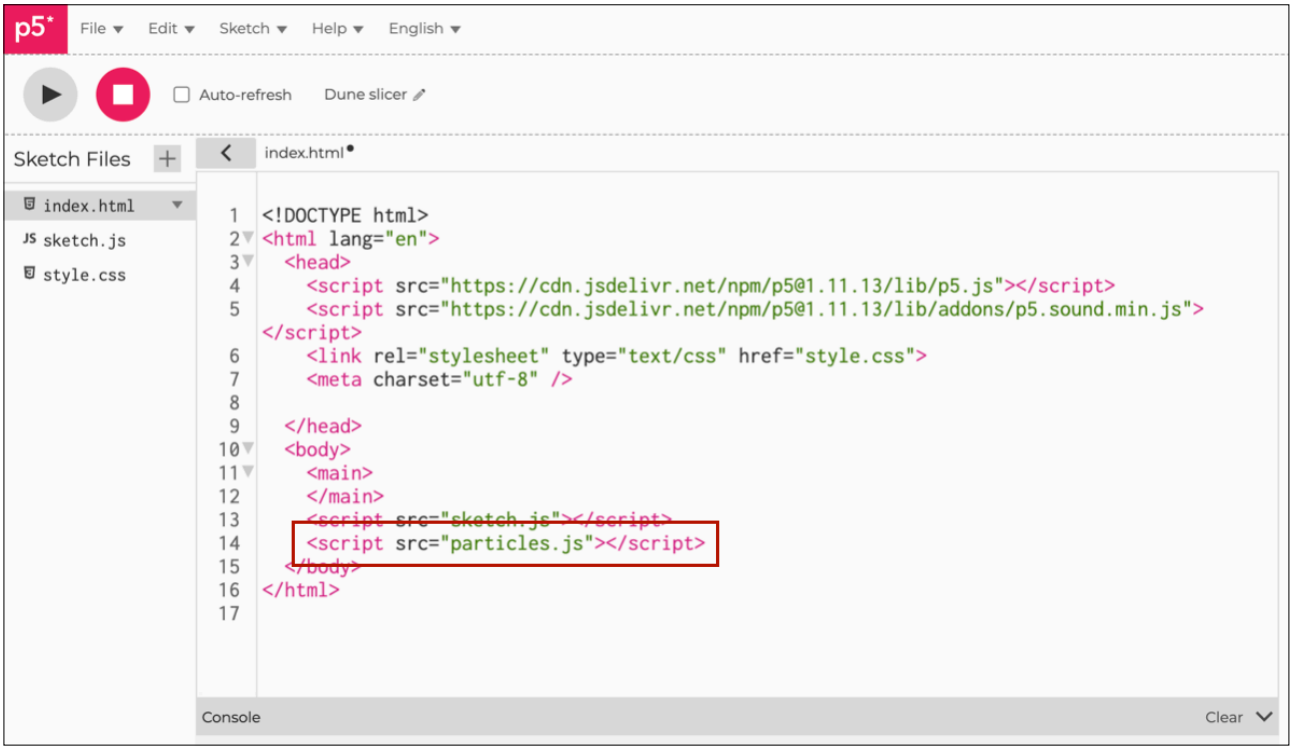
Notice that these lines have <.../> symbols, these are called tags. There is a lot I could share about JavaScript, CSS and HTML, how they work together to create websites, how they give text style and functionality, but that is a topic all on its own and not essential for this tutorial, although might include one at some time in the future.

! Please note that in version 2.x there is no p5.sound.min as it is incorporated (or added separately) in version 2.

```
<script src="https://cdn.jsdelivr.net/npm/p5@2.2.2/lib/p5.js"></
script>
```

E) The key point here is the line of code that shows the `sketch.js` tags, what I do is copy and paste it underneath and change the second `sketch.js` to the new name of the file, in this case `particles.js`. It has to match the name exactly.

Figure 8: index.html new file (showing version 1.x)



Uploading Files

Going back to the `Add files` bit we can also `Upload files`. These tend to be files that contain images, data, videos, models, fonts etc. the process is quite simple but you do need to have the files ready to either access through browsing or to drag and drop from your desktop.

Figure 9: click on the add files button (+)

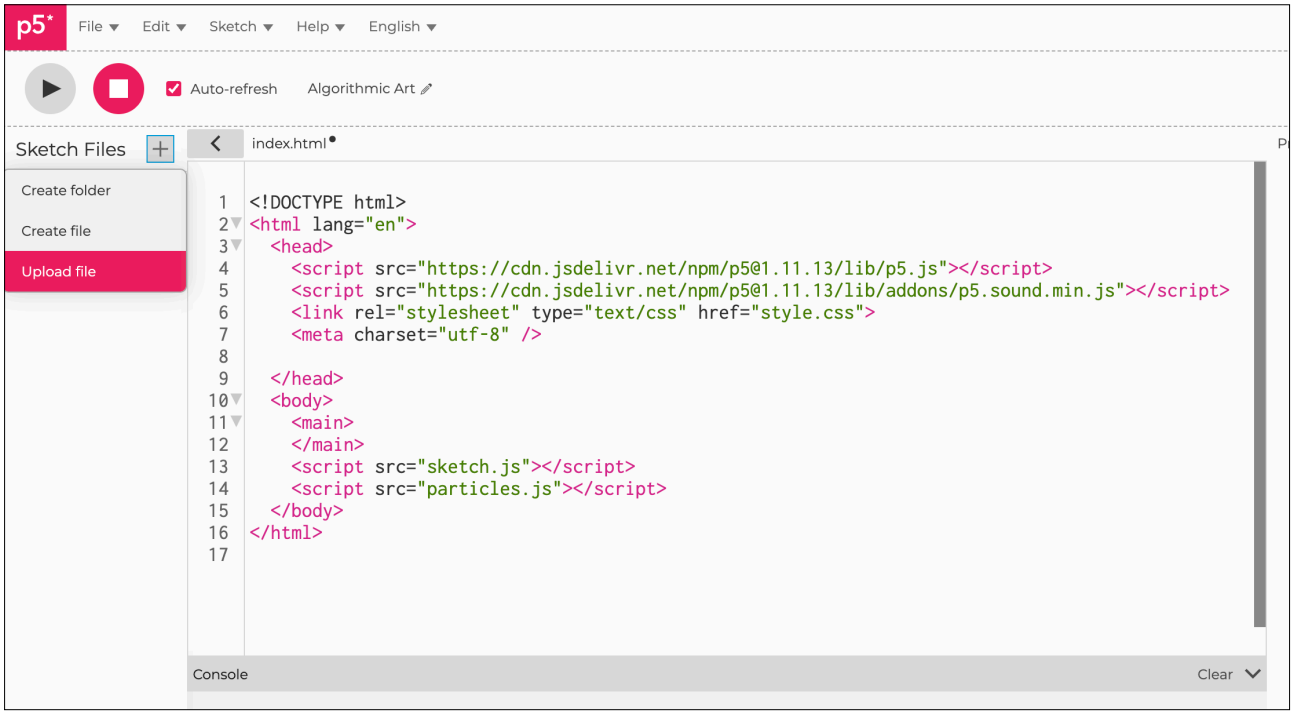
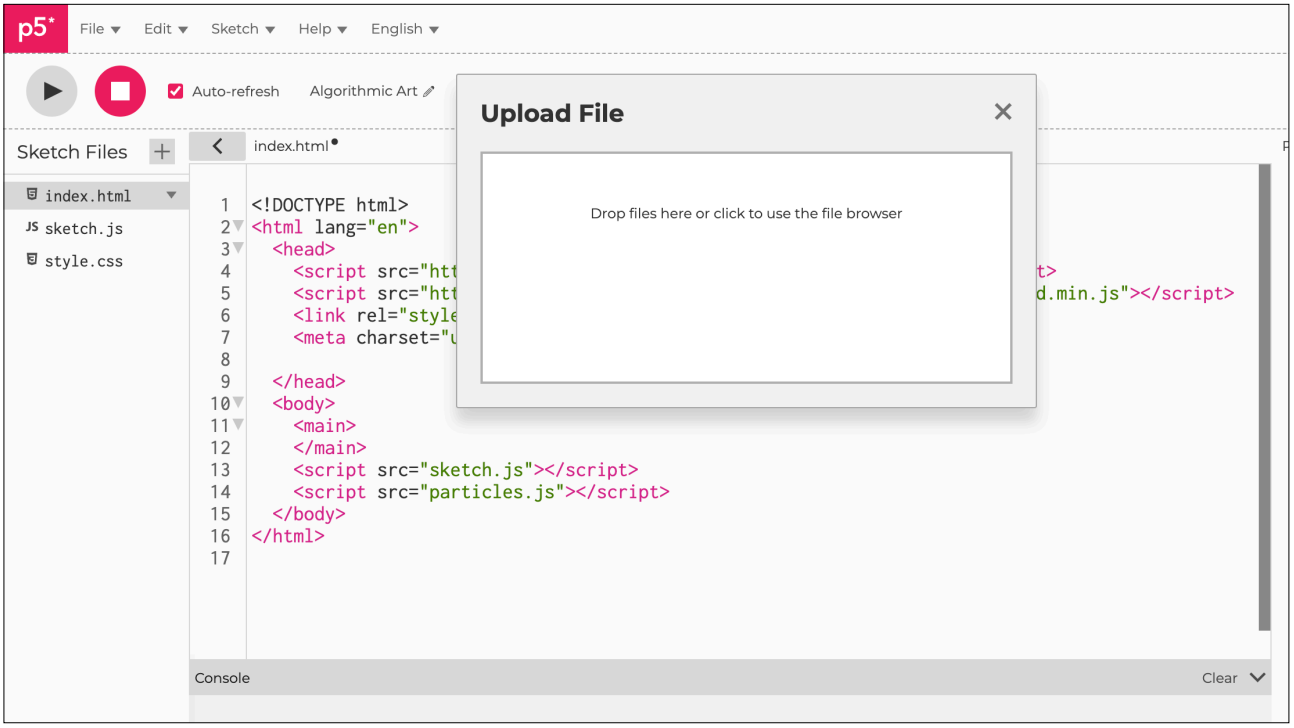
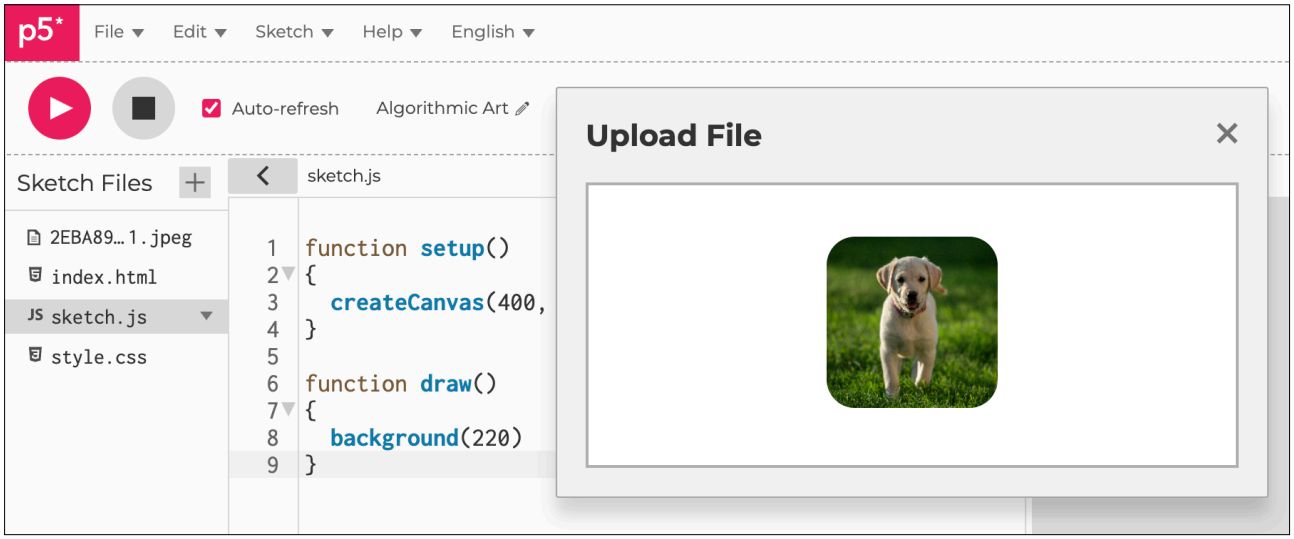


Figure 10: select Upload File



In this instance I uploaded an image of a dog I used for AI from unsplash. You will notice that the image has a file name on the right in the Sketch files. We can change this to something more memorable. Just note that there is a limit to the size of file and the first one I chose was just too big, the limit is 5MB, so make sure you choose a file that will meet those limitations.

Figure 11: uploading image



Let's change the name to dog.jpeg. It is important to keep the same file extension but we can change the name.

Figure 12: renaming the file/image

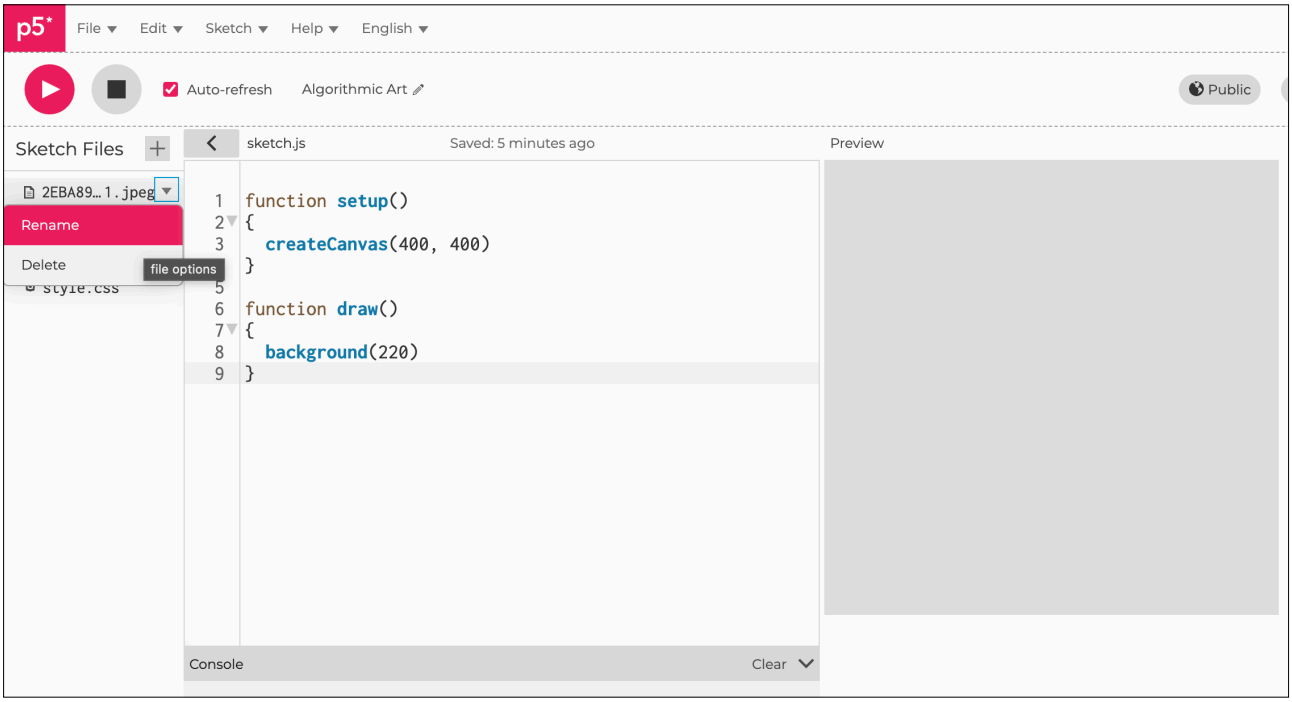


Figure 13: the renamed file

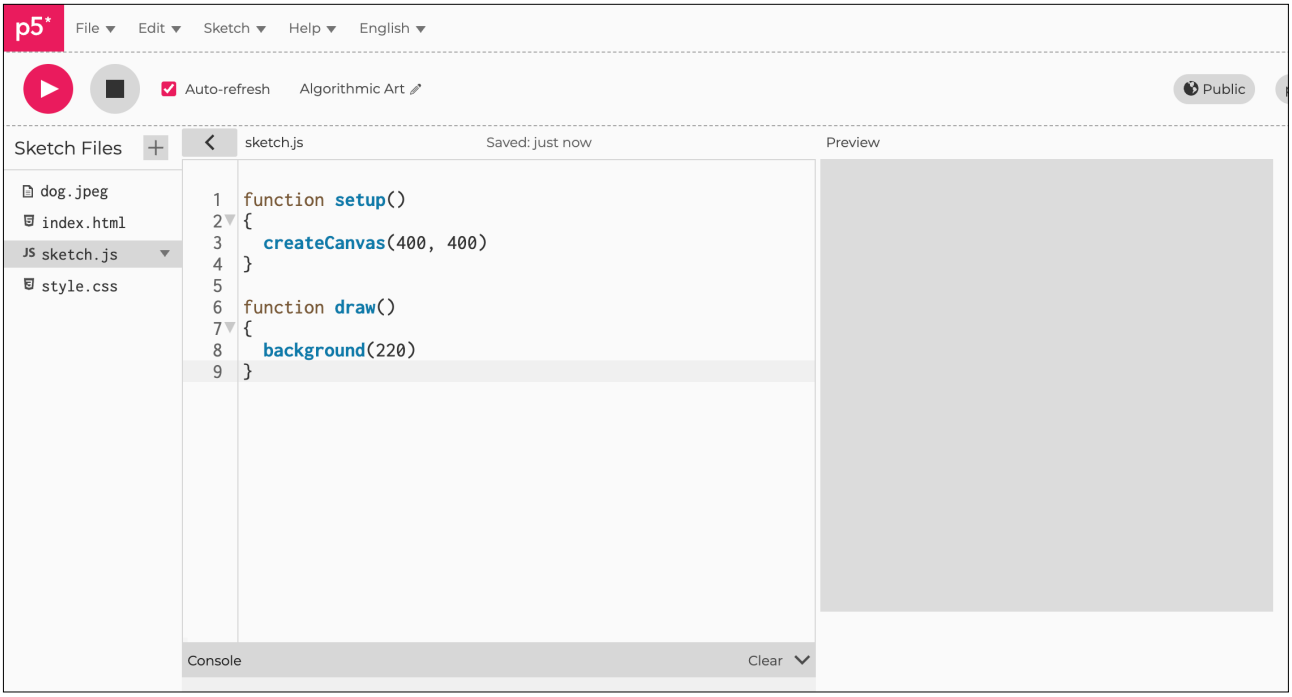
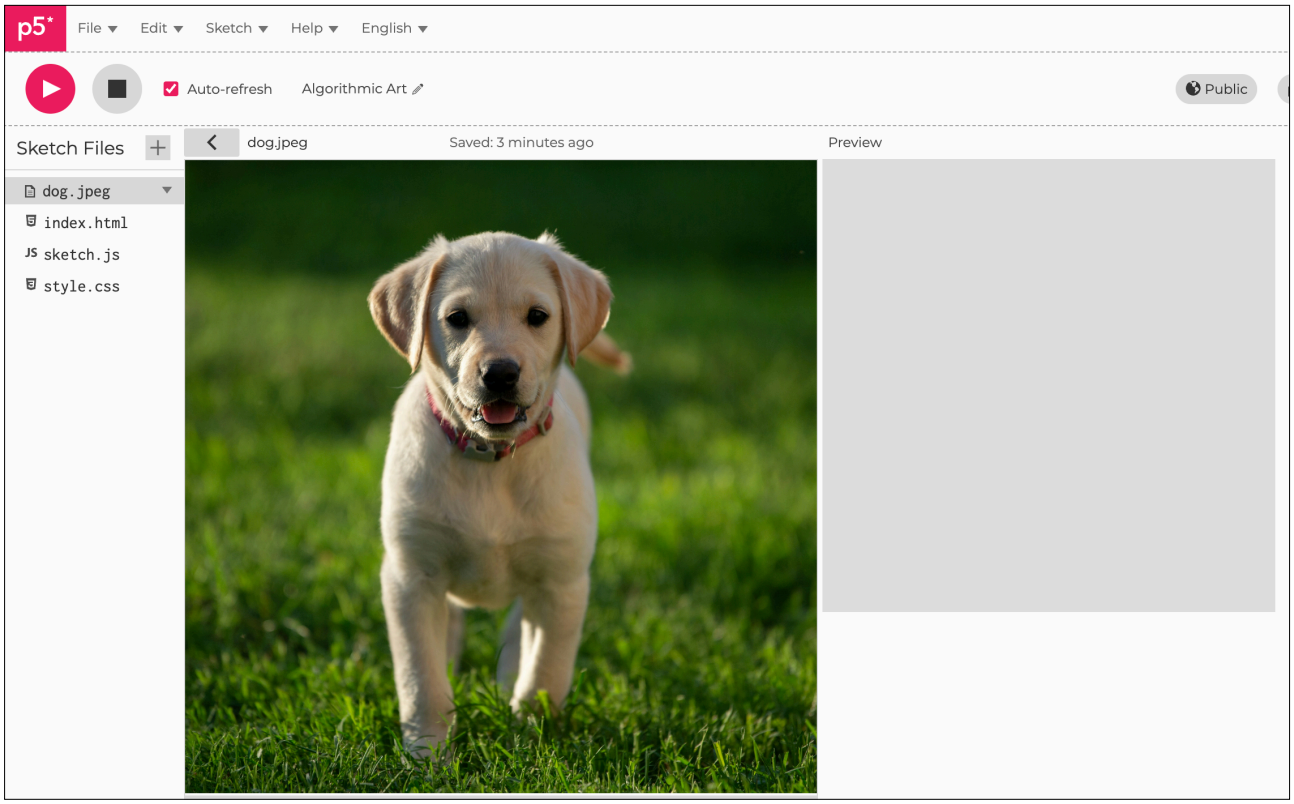


Figure 14: click on the file to see the image



Algorithmic Art
Module E
Unit #2
Perlin noise

Module E Unit #2: perlin noise

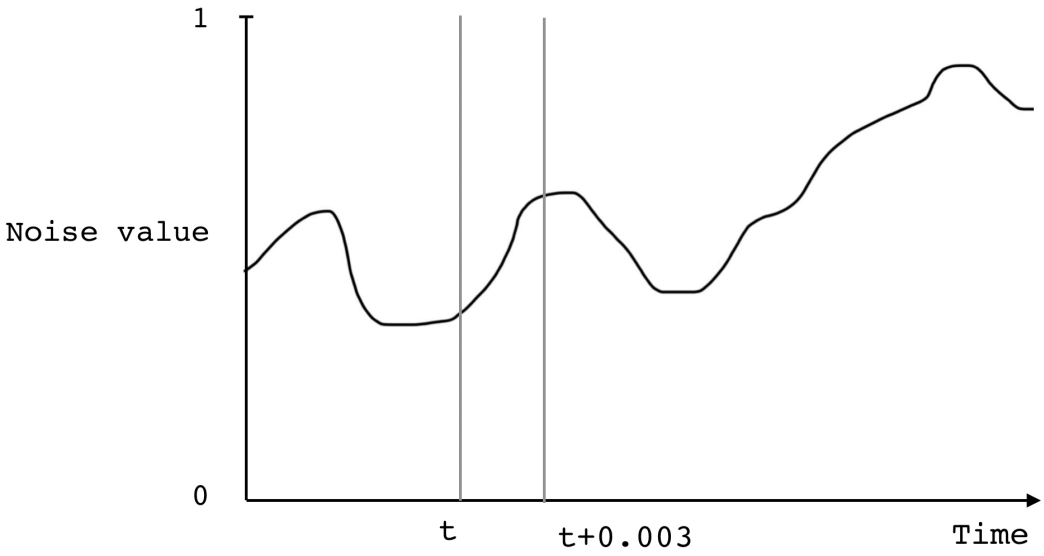
Perlin `noise()` returns a random value between `0.0` and `1.0` at a specific point in time. You specify the start time. It is then incremented along this smooth random time line in steps; the smaller the steps (for instance, `0.005`) means there are very small steps but the changes are therefore smoother, whereas larger steps (`0.03`) obviously create a much greater degree of randomness and possibly less smooth changes. It seems to work best between `0.005` and `0.03`.

If this seems strange at first, it is because it is less intuitive than just plucking a random number out of the air. If you have two variables that you want to have different random noise outcomes, you simply use the `noise()` function but start at different times (points along the line), for instance, one variable could start at `3` and the other starts at `100`.

The beauty of this is that it is random but it bears some relationship with the previous random number at a particular point in time. So if you move it on a small increment, you get a slight adjustment to the random value.

If you want to understand how it works, there are a number of articles around; the `Nature of Code` is a good reference point.

Figure 15: perlin noise graph

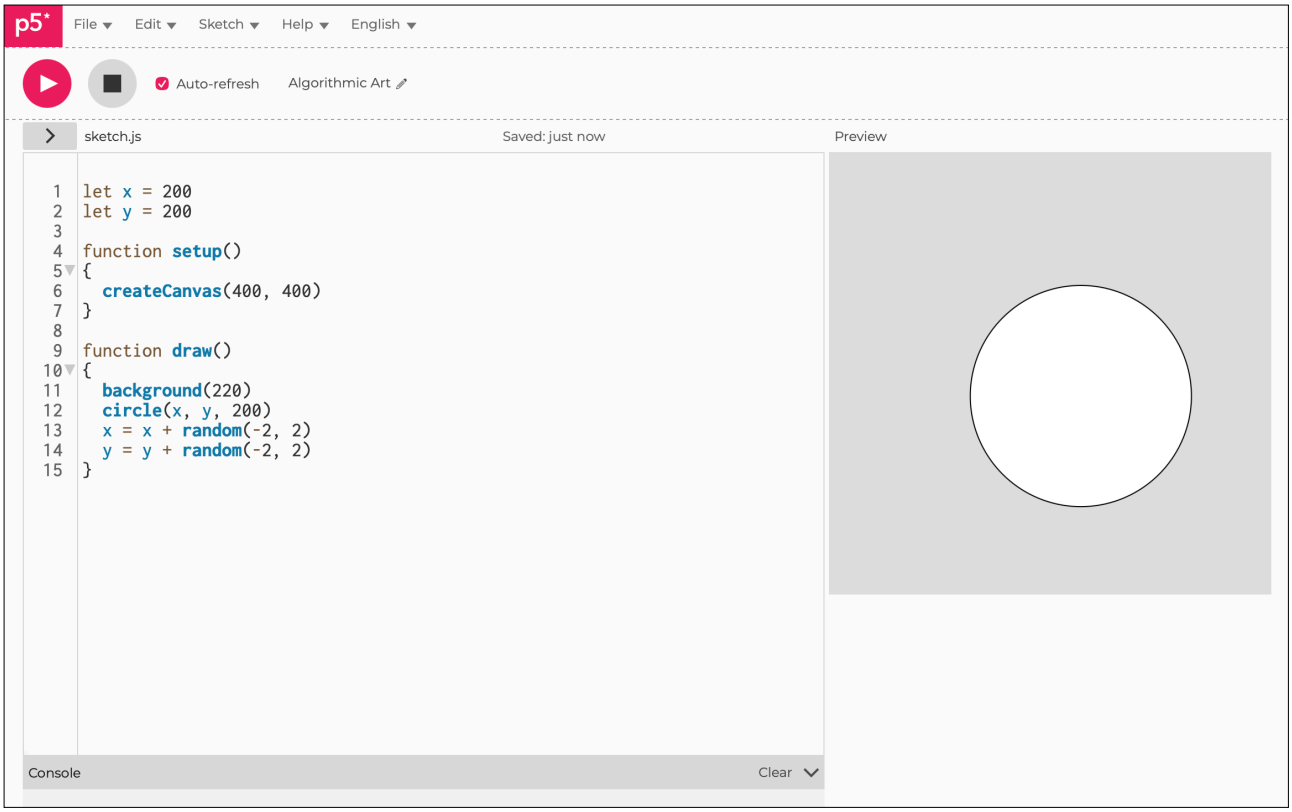


Sketch E2.1 starting with a standard sketch

Starting with our normal basic sketch.

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
}
```

Figure E2.2



Sketch E2.2 randomly moving circle

We start the circle in the centre of the canvas and randomly move it. We are just going to use the `random()` function to move the circle around the canvas. You will notice that it is not very smooth or fluid; `noise` will give you something more natural.

```
let x = 200
let y = 200

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  circle(x, y, 200)
  x = x + random(-2, 2)
  y = y + random(-2, 2)
}
```

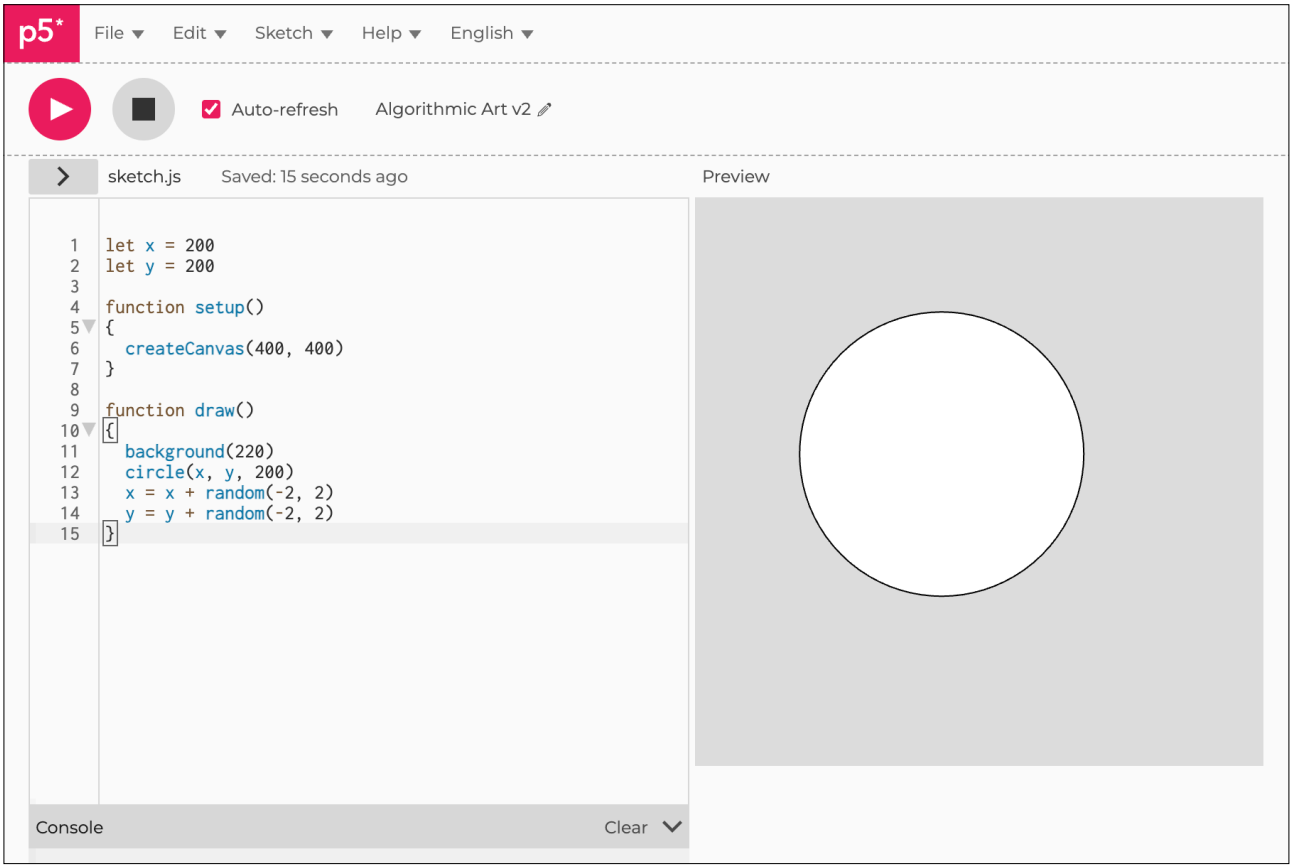
Challenges

1. Try random values of (5, -5).
2. Try different values for x and for y.

Code Explanation

`x = x + random(-2, 2)`	Add a random value between -2 and 2 to the x value on each iteration.
`y = y + random(-2, 2)`	Add a random value between -2 and 2 to the y value on each iteration.

Figure E2.2



Sketch E2.3 smooth random movement

We have replaced the `random()` function with the `noise()` function. Notice that the jerkiness has gone, replaced by a much smoother movement, almost as if floating in the air. Also, it looks a lot more complicated. We have two start times (`3` and `10`). Because `noise` returns values between `0` and `1`, we use the `map()` function to scale the movement up to the `width` and `height` of the canvas. Then, on each iteration, we move along the `noise` timeline by `0.005` increments.

```
let timeX = 3
let timeY = 10

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  let x = map(noise(timeX), 0, 1, 0, width)
  let y = map(noise(timeY), 0, 1, 0, height)
  circle(x, y, 200)
  timeX = timeX + 0.005
  timeY = timeY + 0.005
}
```

Notes

To see how noise works and why it is so much better than just `random()`, this short programme illustrates the smoothness of the movement.

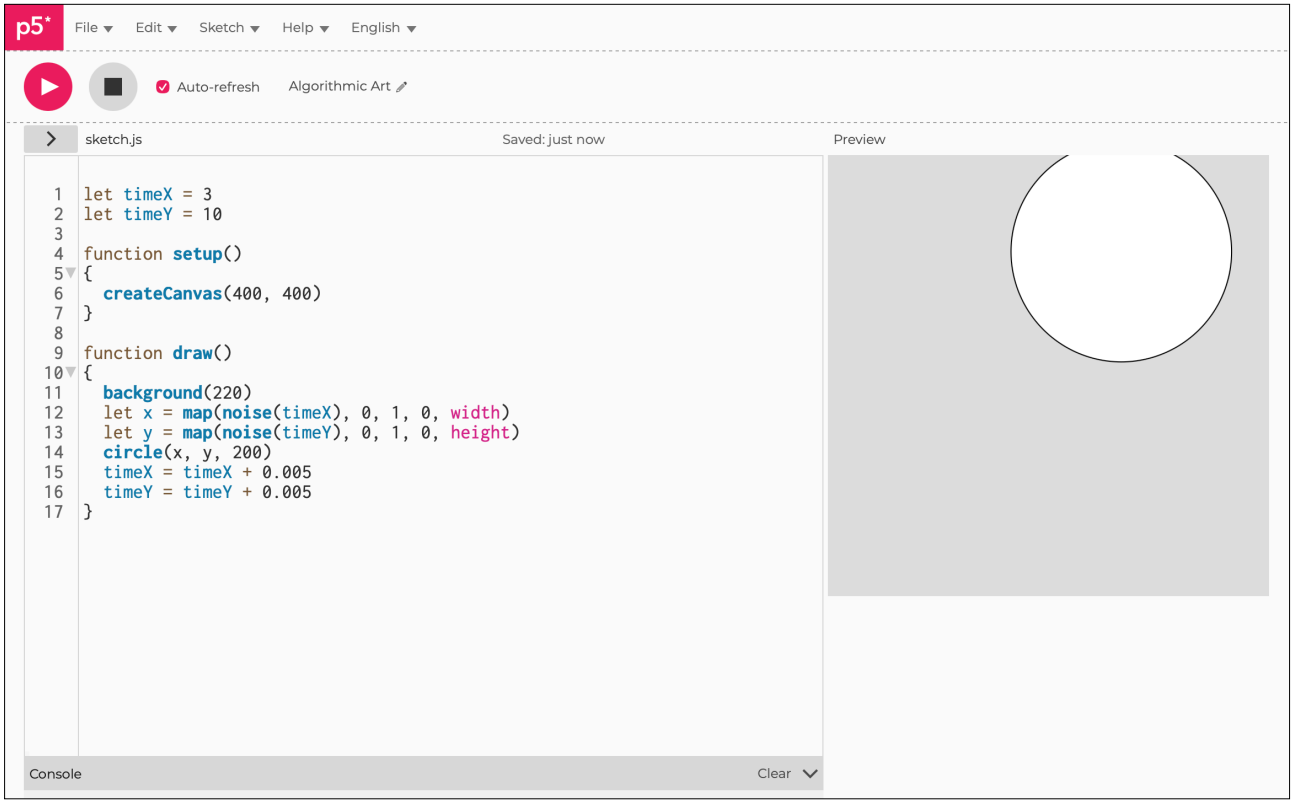
Challenges

1. Try different increments for timeX and timeY.
2. What happens if you give them both the same start on the timeline?
3. Try: `timeX = timeX + 0.05`.
4. Replace it with `timeX += 0.005`.

Code Explanation

`let timeX = 3`	The starting value for the x timeline
`let timeY = 10`	The starting value for the y timeline
`let x = map(noise(timeX), 0, 1, 0, width)`	Maps the timeX value to the width of the canvas
`let y = map(noise(timeY), 0, 1, 0, height)`	Maps the timeY value to the height of the canvas
`timeX = timeX + 0.005`	Adds an increment to the timeX timeline
`timeY = timeY + 0.005`	Adds an increment to the timeY timeline

Figure E2.3


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
let timeX = 3
let timeY = 10
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
let x = map(noise(timeX), 0, 1, 0, width)
let y = map(noise(timeY), 0, 1, 0, height)
circle(x, y, 200)
timeX = timeX + 0.005
timeY = timeY + 0.005
}
Console
Clear

Sketch E2.4 random colour B/W

! Start a new sketch.

Adding a colour element to change the grayscale.

```
let timeCol = 3
let col = 0

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  fill(col)
  col = map(noise(timeCol), 0, 1, 0, 255)
  circle(width/2, height/2, 200)
  timeCol = timeCol + 0.005
}
```

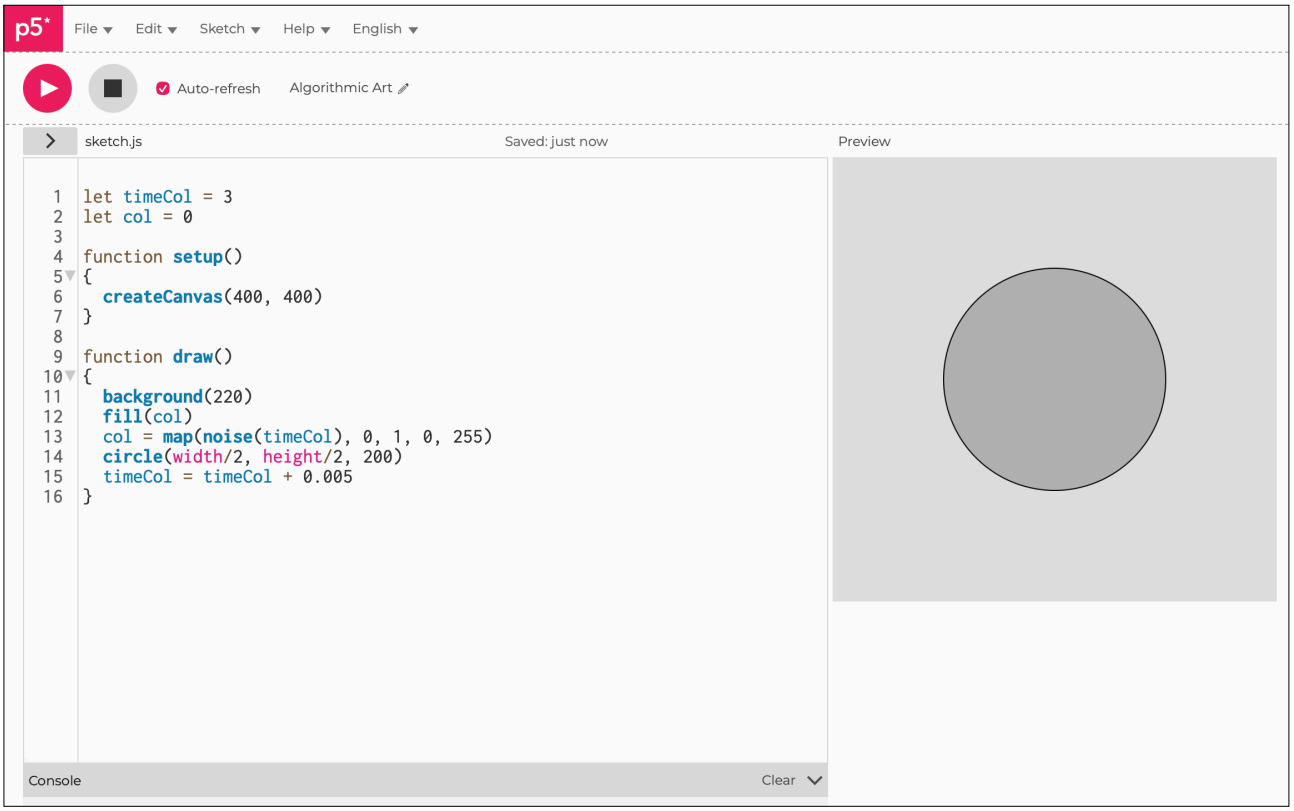
Notes

The colour of the circle changes gradually through the greyscale values between `0` and `255`. The transition from one level of grey to another is also smoother and much more pleasing to the eye.

Challenge

Have the background change as well using `noise()`.

Figure E2.4



Sketch E2.5 random colour RGB

! Remove: the non-relevant code, we are replacing a single colour with the RGB ones instead.

Changing all three to create a flow of colour changes.

```
let timeRed = 30
let timeGreen = 300
let timeBlue = 3000
let colRed = 0
let colGreen = 0
let colBlue = 0

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  fill(colRed, colGreen, colBlue)
  colRed = map(noise(timeRed), 0, 1, 0, 255)
  colGreen = map(noise(timeGreen), 0, 1, 0, 255)
  colBlue = map(noise(timeBlue), 0, 1, 0, 255)
  circle(width/2, height/2, 200)
  timeRed += 0.003
  timeGreen += 0.003
  timeBlue += 0.003
}
```

Notes

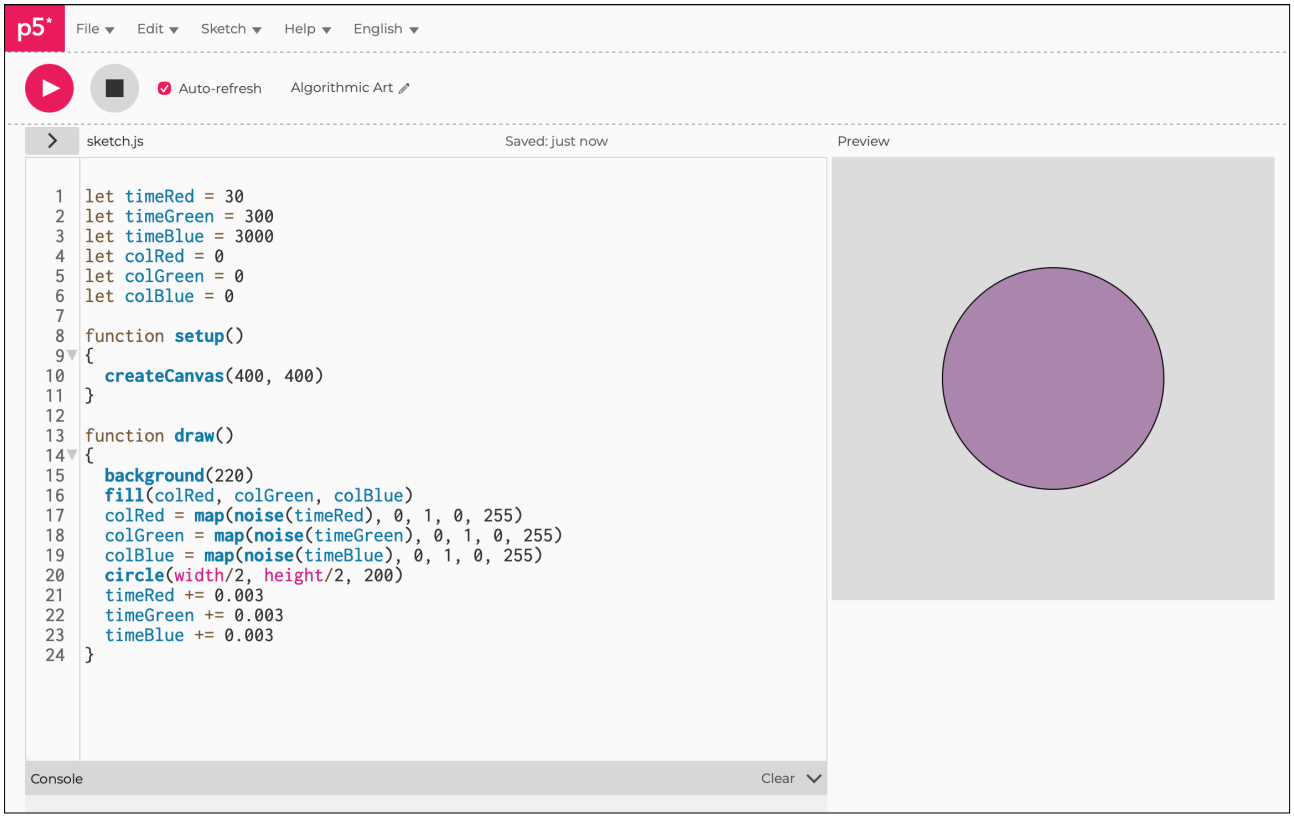
Like the previous sketch, we get a smoother and more subtle transition from one colour to the next.

This can take a little while to get your head around. With random, you give it a number to randomise up to and from, but with noise, you are picking a point on a random timeline. Play with this and persevere; it will become a bit more intuitive.

Challenges

1. Change the background colour, easy hint: just swap the colour variables round, for instance: `background(colGreen, colBlue, colRed)`
2. Movement, colour, and size all at the same time.

Figure E2.5



Algorithmic Art
Module E
Unit #3
2D Perlin Noise

Module E Unit #3: 2D Perlin Noise

Although we touched on Perlin noise in the first unit, we will continue by revisiting it from another angle, which may help you in understanding Perlin noise intuitively before jumping into 2D Perlin noise.

Sketch E3.1 starting sketch

! Our starting sketch

This is a little bit of a recap of 1D Perlin noise; we will build on this very quickly.

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
}
```

Sketch E3.2 row of pixels

We create a `for()` loop and a point every pixel across the width of the canvas.

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  for (x = 0; x < width; x++)
  {
    point(x, 200)
  }
}
```

Figure E3.2


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
for (x = 0; x < width; x++)
{
point(x, 200)
}
}
Console
Clear

Sketch E3.3 vertex replaces point

Replace the point with a vertex and join them all up. You should get a line as before.

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  beginShape()
  for (x = 0; x < width; x++)
  {
    vertex(x, 200)
  }
  endShape()
}
```

Notes

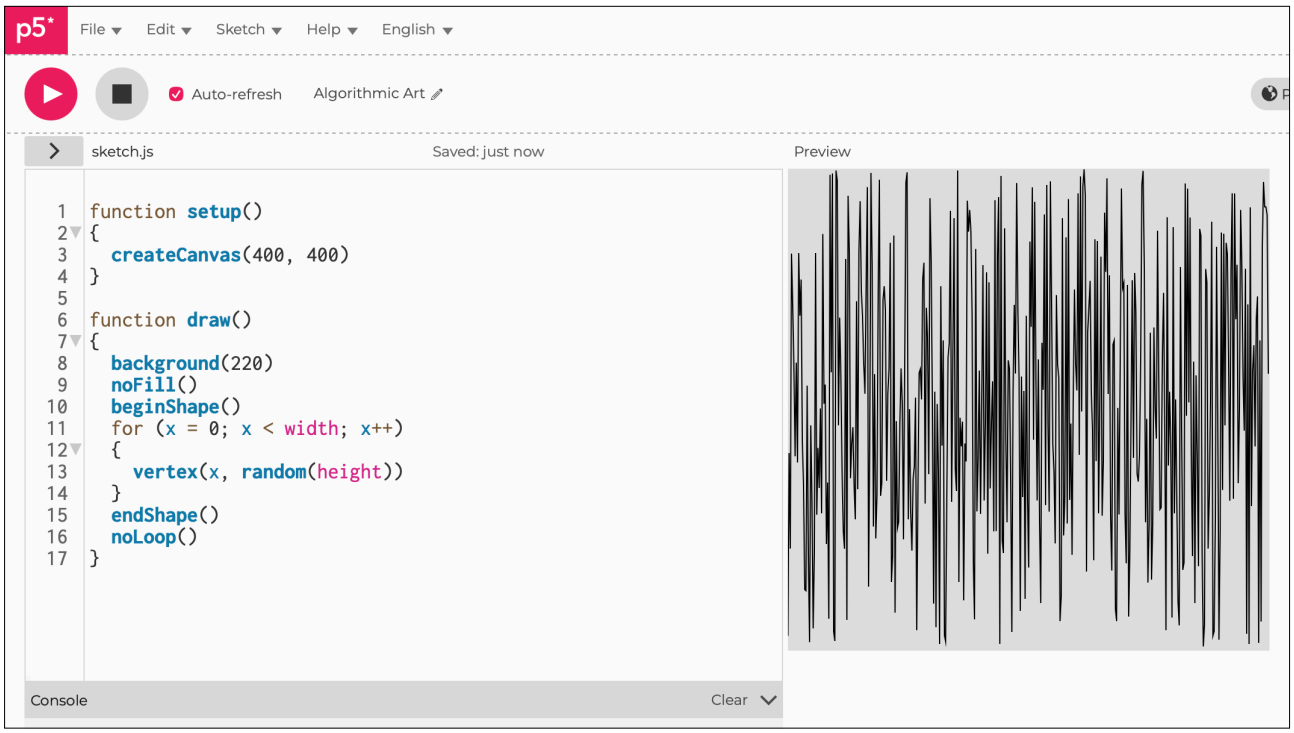
You should have what looks like the same as before. Instead, we have line joined the points (pixels) but too small to see.

Sketch E3.4 random y

Instead of a `y` value of `200`, let's give it a random value between the top and the bottom of the canvas. We also include a `noLoop()` function so that we have a static image; if you comment it out, you will see the random effect animated.

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  noFill()
  beginShape()
  for (x = 0; x < width; x++)
  {
    vertex(x, random(height))
  }
  endShape()
  noLoop()
}
```

Figure E3.4

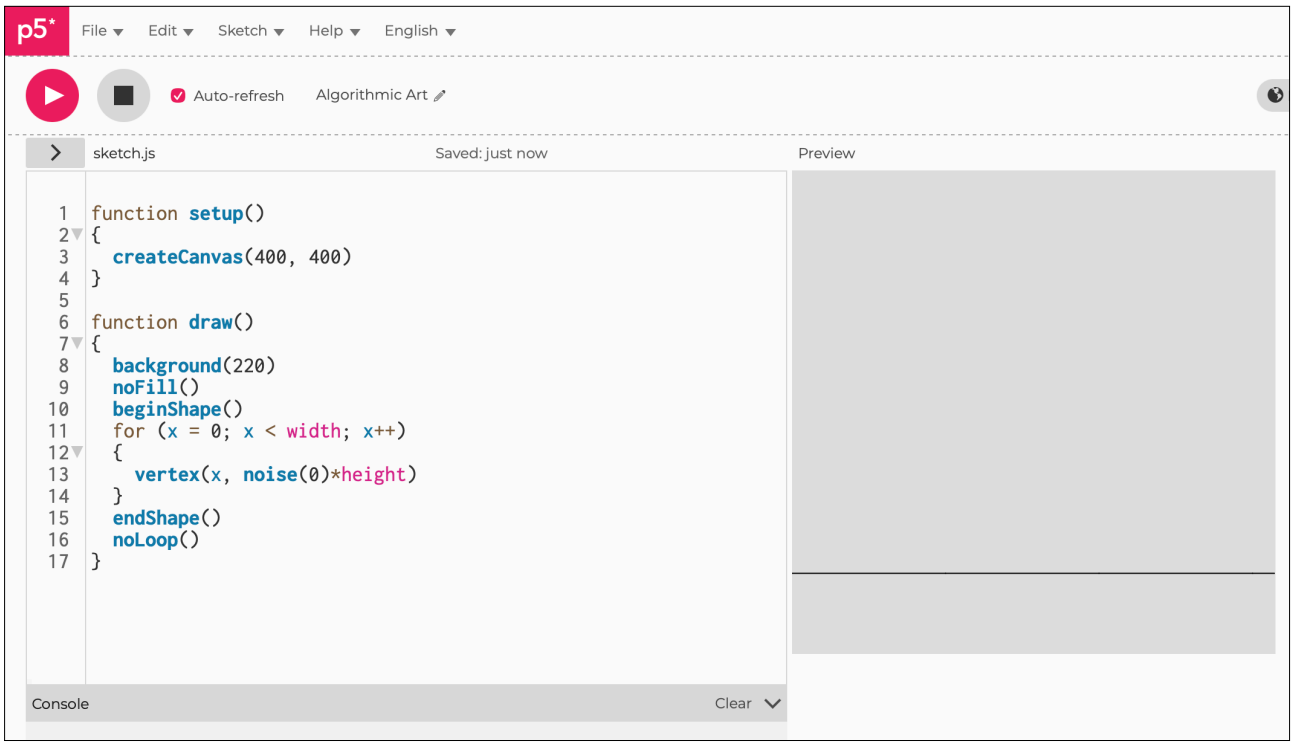

p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
noFill()
beginShape()
for (x = 0; x < width; x++)
{
vertex(x, random(height))
}
endShape()
noLoop()
}
Console
Clear

Sketch E3.5 introducing the perlin noise

We can introduce Perlin `noise()` rather than `random()`. We will get the `y` value at `0` and also multiply it by the `height` because noise returns a value between `0` and `1`. We will get a straight line somewhere between the top and the bottom of the canvas. Refresh and you will get a different value returned.

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  noFill()
  beginShape()
  for (x = 0; x < width; x++)
  {
    vertex(x, noise(0)*height)
  }
  endShape()
  noLoop()
}
```

Figure E3.5

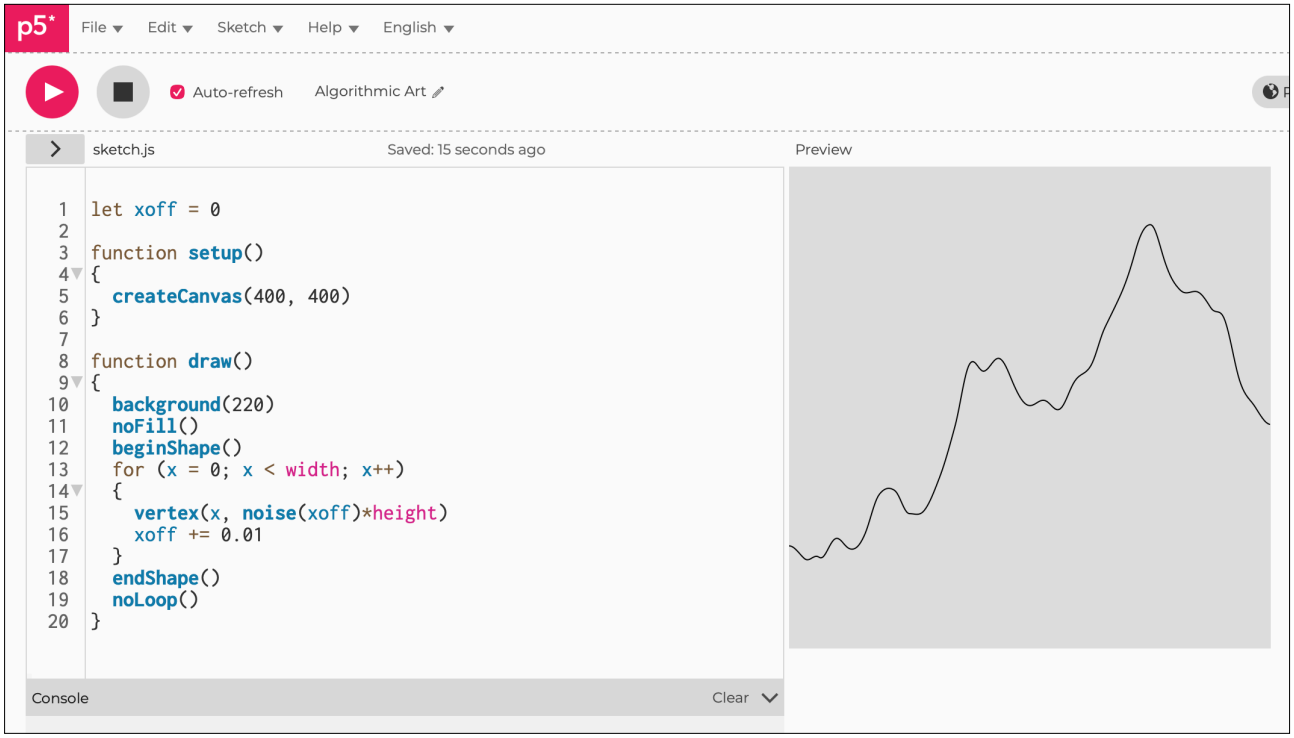

p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
noFill()
beginShape()
for (x = 0; x < width; x++)
{
vertex(x, noise(0)*height)
}
endShape()
noLoop()
}
Console
Clear

Sketch E3.6 xoff

But what we have is a static value of Perlin noise. We want to move through the values, so we need another variable called `xoff` (short for x offset). Previously, we just used the variable time. This is what Perlin `noise()` random looks like compared to pure `random()` as we had before. Nice and smooth.

```
let xoff = 0

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  noFill()
  beginShape()
  for (x = 0; x < width; x++)
  {
    vertex(x, noise(xoff)*height)
    xoff += 0.01
  }
  endShape()
  noLoop()
}
```

Figure E3.6


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
let xoff = 0
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
noFill()
beginShape()
for (x = 0; x < width; x++)
{
vertex(x, noise(xoff)*height)
xoff += 0.01
}
endShape()
noLoop()
}
Console
Clear

Sketch E3.7 a bit static

This is still static, and we want to animate it so that you can see it flowing. We will replace the initial value with a variable called `inc` (short for increment). Nothing changes yet.

```
let xoff = 0
let inc = 0.01

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  noFill()
  beginShape()
  for (x = 0; x < width; x++)
  {
    vertex(x, noise(xoff)*height)
    xoff += inc
  }
  endShape()
  noLoop()
}
```

Sketch E3.8 scrolling graph

Let's add another variable so that it doesn't begin at zero. We also comment out the noLoop() so we can animate. What you should get is a scrolling graph as Perlin `noise()` changes. We are effectively moving the `start` forward.

```
let xoff = 0
let inc = 0.01
let start = 0

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  noFill()
  xoff = start
  beginShape()
  for (x = 0; x < width; x++)
  {
    vertex(x, noise(xoff)*height)
    xoff += inc
  }
  endShape()
  start += inc
  // noLoop()
}
```

Notes

You should have the Perlin `noise()` line scrolling slowly across the canvas.

Challenge

Try different `inc` values.

Sketch E3.9 noise detail

We can control the Perlin `noise()` with `noiseDetail()`. It takes two arguments, the default is `4` and `0.5`. The first one is the `octave` and the second one is the `fall-off`. Think detail rather than worry about what they mean.

```
let xoff = 0
let inc = 0.01
let start = 0

function setup()
{
  createCanvas(400, 400)
  noiseDetail(4, 0.5)
}

function draw()
{
  background(220)
  noFill()
  xoff = start
  beginShape()
  for (x = 0; x < width; x++)
  {
    vertex(x, noise(xoff)*height)
    xoff += inc
  }
  endShape()
  start += inc
  // noLoop()
}
```

Notes

There is no appreciable difference immediately.

Challenge

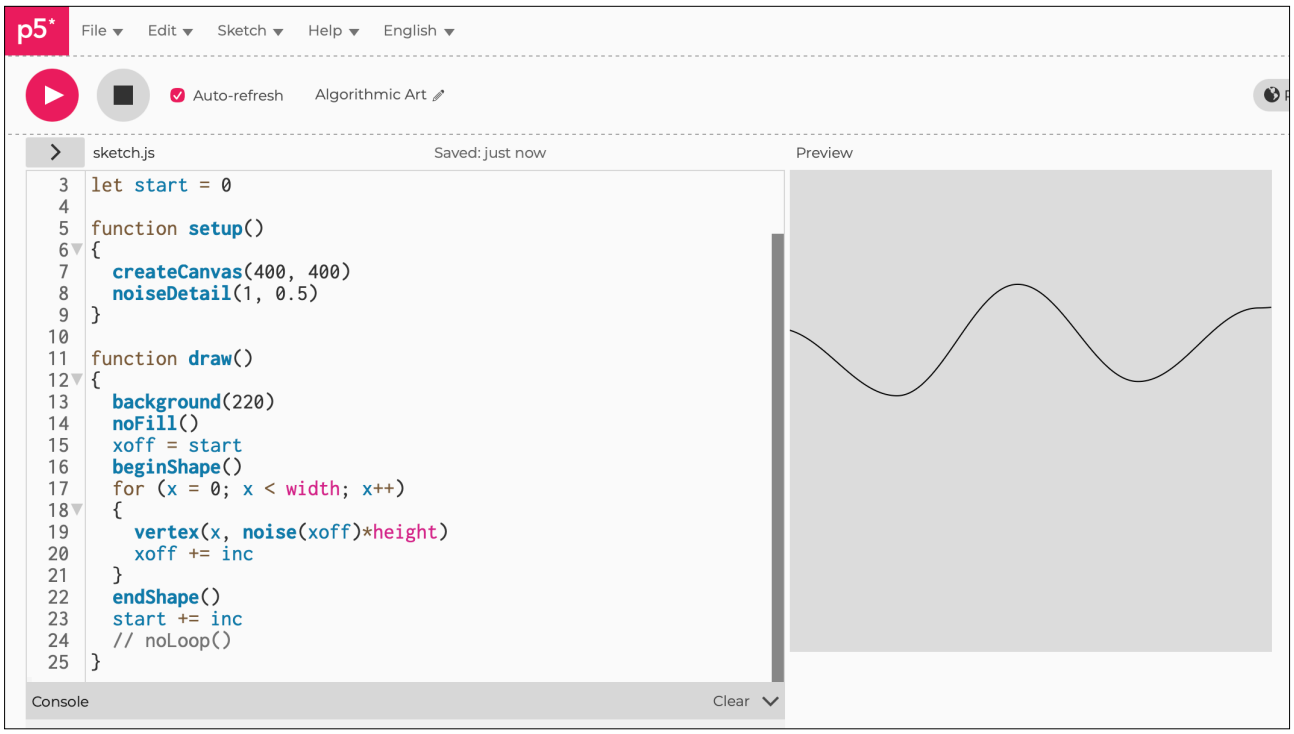
Play with the values. I recommend between `1` and `24` for the first argument and `0` to `1` for the second.

Sketch E3.10 smoothing

Creating a much smoother graph.

```
let xoff = 0
let inc = 0.01
let start = 0

function setup()
{
  createCanvas(400, 400)
  noiseDetail(1, 0.5)
}

function draw()
{
  background(220)
  noFill()
  xoff = start
  beginShape()
  for (x = 0; x < width; x++)
  {
    vertex(x, noise(xoff)*height)
    xoff += inc
  }
  endShape()
  start += inc
  // noLoop()
}
```

Figure E3.10

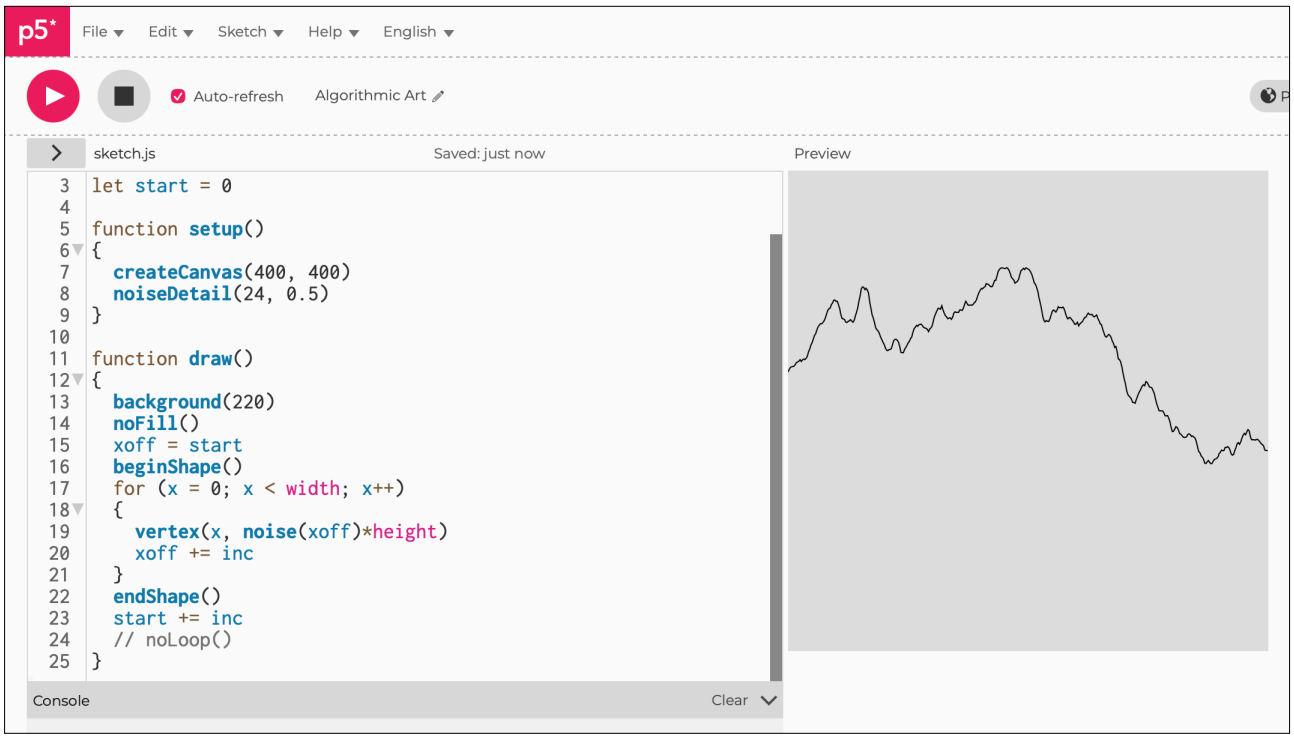

p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
3 let start = 0
4
5 function setup()
6 {
7 createCanvas(400, 400)
8 noiseDetail(1, 0.5)
9 }
10
11 function draw()
12 {
13 background(220)
14 noFill()
15 xoff = start
16 beginShape()
17 for (x = 0; x < width; x++)
18 {
19 vertex(x, noise(xoff)*height)
20 xoff += inc
21 }
22 endShape()
23 start += inc
24 // noLoop()
25 }
Console
Clear

Sketch E3.11 not so smooth

Less smooth.

```
let xoff = 0
let inc = 0.01
let start = 0

function setup()
{
  createCanvas(400, 400)
  noiseDetail(24, 0.5)
}

function draw()
{
  background(220)
  noFill()
  xoff = start
  beginShape()
  for (x = 0; x < width; x++)
  {
    vertex(x, noise(xoff)*height)
    xoff += inc
  }
  endShape()
  start += inc
  // noLoop()
}
```

Figure E3.11


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
3 let start = 0
4
5 function setup()
6 {
7 createCanvas(400, 400)
8 noiseDetail(24, 0.5)
9 }
10
11 function draw()
12 {
13 background(220)
14 noFill()
15 xoff = start
16 beginShape()
17 for (x = 0; x < width; x++)
18 {
19 vertex(x, noise(xoff)*height)
20 xoff += inc
21 }
22 endShape()
23 start += inc
24 // noLoop()
25 }
Console
Clear

2D perlin noise

So far, we have only explored 1-dimensional (1D) noise(), which is the value of y as x moves along the x-axis (time). For 2D Perlin noise(), we need to consider all the surrounding values for any pixel on the canvas.

We are going to use the basis of the above sketch for the next section, which will generate a more interesting pattern for you to explore. This will form the basis for FlowFields later.

Sketch E3.12 time for 2D

Remove all the highlighted in blue (and commented out).

```
let xoff = 0
let inc = 0.01
// let start = 0

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  // background(220)
  // noFill()
  // xoff = start
  // beginShape()
  for (x = 0; x < width; x++)
  {
    // vertex(x, noise(xoff)*height)
    // xoff += inc
  }
  // endShape()
  // start += inc
  // noLoop()
}
```

Sketch E3.13 are we ready?

You should have something like this.

```
let xoff = 0
let inc = 0.01

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  for (x = 0; x < width; x++)
  {

  }
}
```

Notes

We have no canvas.

Sketch E3.14 loading the pixels

We are going to use a function called `loadPixels()` to examine whichever pixels we want to look at (in this case, all of them) and then `updatePixels()` after we have done something to them. We cycle through every pixel, hence the `y for()` loop.

```
let xoff = 0
let inc = 0.01

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  loadPixels()
  for (x = 0; x < width; x++)
  {
    for (y = 0; y < height; y++)
    {


    }
  }
  updatePixels()
}
```

Notes

Again, no canvas; be patient.

Sketch E3.15 four channels

There are four channels: red, green, blue, and alpha for each pixel. So we are going to set the red to full (255) with no green or blue and full alpha (255). We have to set the pixelDensity() to 1 because of my display (Mac Retina); you may not need it depending on your monitor/screen.

```
let xoff = 0
let inc = 0.01
let index

function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

function draw()
{
  loadPixels()
  for (x = 0; x < width; x++)
  {
    for (y = 0; y < height; y++)
    {
      index = (x + y * width) * 4
      pixels[index + 0] = 255
      pixels[index + 1] = 0
      pixels[index + 2] = 0
      pixels[index + 3] = 255
    }
  }
  updatePixels()
}
```

Notes

What you will see is a red canvas. Play with the other values and see how it works.

Challenge

Create other colours.

Figure E3.15

Sketch E3.16 random static

If you randomise the values for RGB, you get a random image (add noLoop() to stop it flickering). However, each pixel has no correlation with any of the surrounding pixels; every pixel is random in isolation.

```
let xoff = 0
let inc = 0.01
let index
let val

function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

function draw()
{
  loadPixels()
  for (x = 0; x < width; x++)
  {
    for (y = 0; y < height; y++)
    {
      index = (x + y * width) * 4
      val = random(255)
      pixels[index + 0] = val
      pixels[index + 1] = val
      pixels[index + 2] = val
      pixels[index + 3] = 255
    }
  }
  updatePixels()
  noLoop()
}
```

Notes

What do you think would happen if we gave each pixel its separate random value?

Figure E3.16

Sketch E3.17 make it noisy

Next step is to introduce `noise()` so that we can have the situation where each pixel is related to the surrounding pixels. We first replace `random()` with `noise()` and multiply by `255` because `noise()` returns a value between `0` and `1`, what we want are values between `0` and `255`. Every time you refresh the sketch, you get a different shade of grey.

```
let xoff = 0
let inc = 0.01
let index
let val

function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

function draw()
{
  loadPixels()
  for (x = 0; x < width; x++)
  {
    for (y = 0; y < height; y++)
    {
      index = (x + y * width) * 4
      val = noise(xoff) * 255
      pixels[index + 0] = val
      pixels[index + 1] = val
      pixels[index + 2] = val
      pixels[index + 3] = 255
    }
  }
  updatePixels()
  noLoop()
}
```

Notes

You get a single colour of some grey value.

Figure E3.17

Sketch E3.18 a wee amount

If we change each pixel `xoff` value by a small amount, we get the following effect (which is still not it, by the way).

```
let xoff = 0
let inc = 0.01
let index
let val

function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

function draw()
{
  loadPixels()
  for (x = 0; x < width; x++)
  {
    for (y = 0; y < height; y++)
    {
      index = (x + y * width) * 4
      val = noise(xoff) * 255
      pixels[index + 0] = val
      pixels[index + 1] = val
      pixels[index + 2] = val
      pixels[index + 3] = 255
      xoff += inc
    }
  }
  updatePixels()
  noLoop()
}
```

Figure E3.18


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
8 createCanvas(400, 400)
9 pixelDensity(1)
10 }
11
12 function draw()
13 {
14 loadPixels()
15 for (x = 0; x < width; x++)
16 {
17 for (y = 0; y < height; y++)
18 {
19 index = (x + y * width) * 4
20 val = noise(xoff) * 255
21 pixels[index + 0] = val
22 pixels[index + 1] = val
23 pixels[index + 2] = val
24 pixels[index + 3] = 255
25 xoff += inc
26 }
27 }
28 updatePixels()
29 noLoop()
30 }
Console
Clear

Sketch E3.19 reversing the loops

Presently, we have the `y loops()` inside the `x`, but they need to be the other way round. We do a `row` first, so `y` needs to stay static as we move through from left to right.

```
let xoff = 0
let inc = 0.01
let index
let val

function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

function draw()
{
  loadPixels()
  for (y = 0; y < height; y++)
  {
    for (x = 0; x < width; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff) * 255
      pixels[index + 0] = val
      pixels[index + 1] = val
      pixels[index + 2] = val
      pixels[index + 3] = 255
      xoff += inc
    }
  }
  updatePixels()
  noLoop()
}
```

Notes

What is happening is that the `Perlin noise()` value is changing as it goes across the canvas and then continues at the start of the next row of pixels. So what you are seeing is `Perlin noise()`, but it has no relationship with the value above or below.

Figure E3.19

Sketch E3.20 yoff

We want a yoff to work downwards as well, but when it moves down a row, we want the xoff to reset, not carry on with the Perlin noise() increments.

```
let xoff = 0
let yoff = 0
let inc = 0.01
let index
let val

function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

function draw()
{
  loadPixels()
  for (y = 0; y < height; y++)
  {
    xoff = 0
    for (x = 0; x < width; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      pixels[index + 0] = val
      pixels[index + 1] = val
      pixels[index + 2] = val
      pixels[index + 3] = 255
      xoff += inc
    }
    yoff += inc
  }
  updatePixels()
  noLoop()
}
```

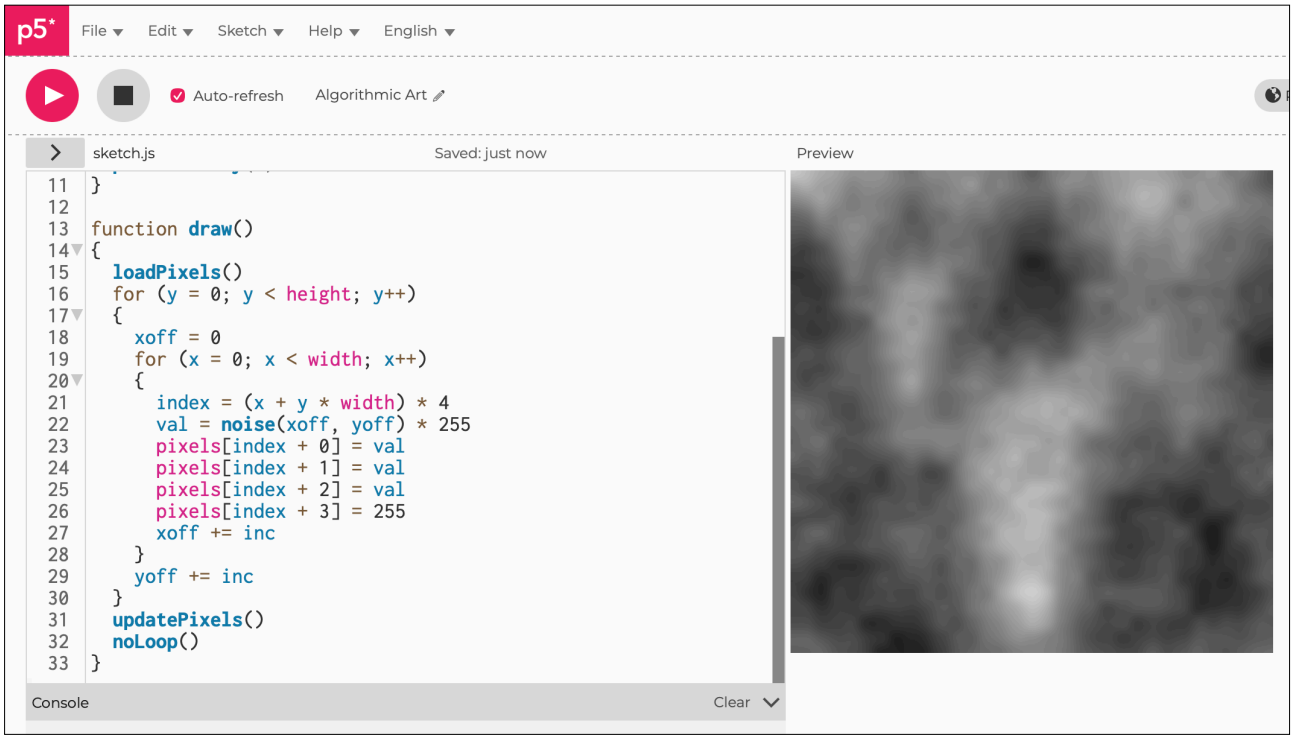
Notes

What you get is the image where each pixel has now some relationship with its neighbouring ones. If the lines of code above aren't obvious, this is where you take the time to work through slowly what is happening to each pixel as you move across the canvas and then onto the next row of pixels. This is where the logic of coding comes in.

Challenges

Change the `inc` value to see the effect.
Add noise values for each colour (RGB).
Use `noiseDetail()` to see what difference you can make to the image.

Figure E3.20



Algorithmic Art Module E Unit #4 Smoke and Fire

Module E Unit #4: smoke and fire

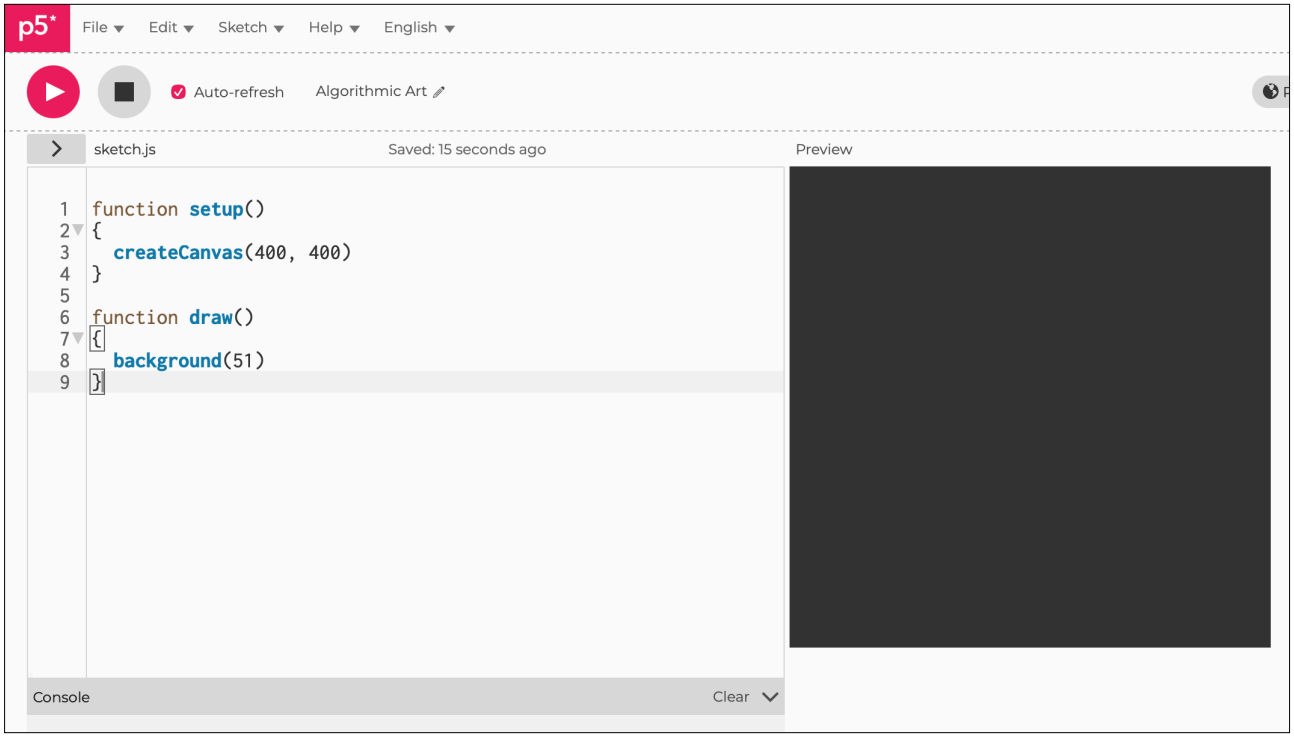
Creating a simple smoke or fire effect using a simple particle system with p5.js.

Sketch E4.1 starting sketch

This is our basic sketch; this gives us a nice dark background for a change.

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
}
```

Figure E4.1


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(51)
}
Console
Clear

Sketch E4.2 adding a particle class

We are going to add a particle class with a constructor function. We want the particles to start near the bottom and in the middle, then travel up and fade.

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
}
```

```
class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
  }
}
```

Notes

Nothing has changed.

Sketch E4.3 adding move and show

We want a `show()` function and a `move()` function.

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
}

class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
  }

  show()
  {

  }

  move()
  {

  }
}
```

Sketch E4.4 draw a circle

We draw a circle at these x and y co-ordinates filling it with white (255).

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
}

class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
  }

  show()
  {
    stroke(255)
    fill(255, 10)
    circle(this.x, this.y, 16)
  }

  move()
  {

  }
}
```

Sketch E4.5 particle p

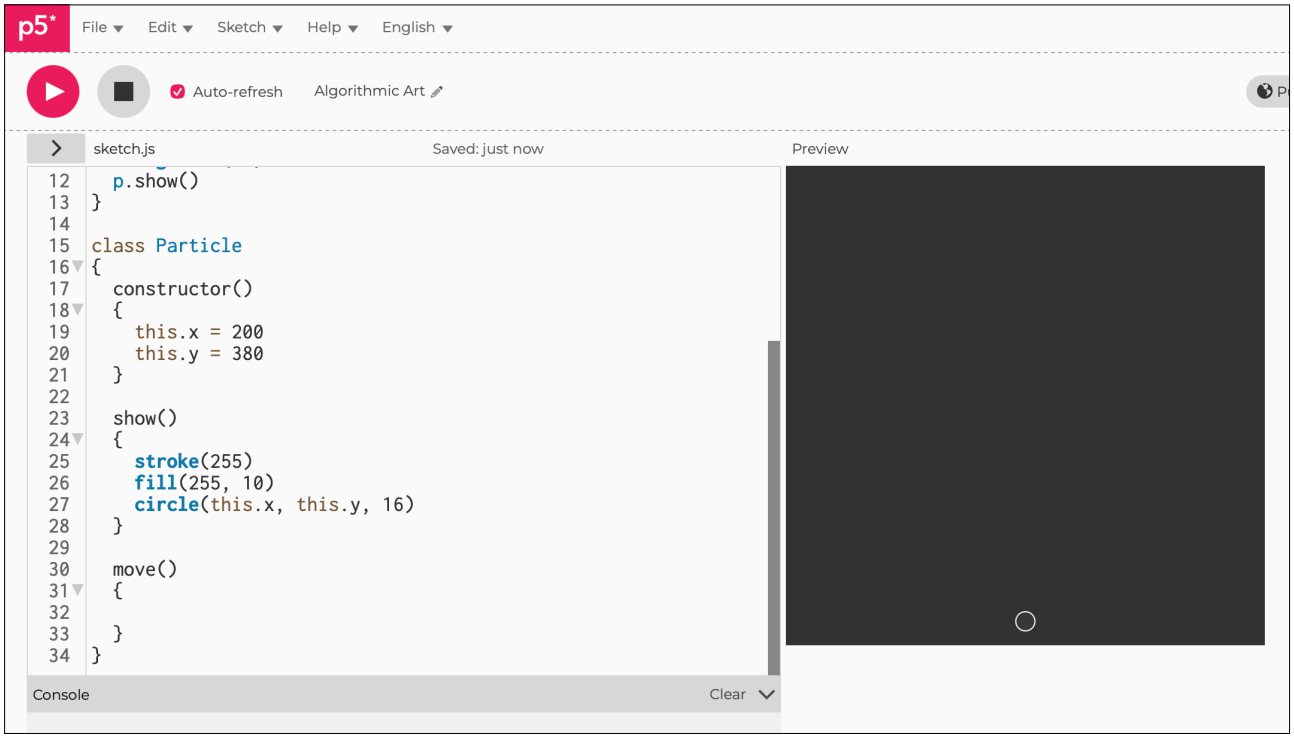
Let's create one particle called p so we can see it.

```
let p

function setup()
{
  createCanvas(400, 400)
  p = new Particle()
}

function draw()
{
  background(51)
  p.show()
}

class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
  }

  show()
  {
    stroke(255)
    fill(255, 10)
    circle(this.x, this.y, 16)
  }

  move()
  {

  }
}
```

Notes

At last, we have something to see!

Figure E4.5



Sketch E4.6 an array of particles

We want to create many particles, so we create an array and `push()` them into the array using a `for()` loop. We will start with just one particle.

```
let p
let particles = []

function setup()
{
  createCanvas(400, 400)
  p = new Particle()
  particles.push(p)
}

function draw()
{
  background(51)
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].show()
  }
}

class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
  }

  show()
  {
    stroke(255)
    fill(255, 10)
    circle(this.x, this.y, 16)
  }
```

```
  move()
  {

  }
}
```

Notes

You should have exactly the same as before.

Sketch E4.7 move the particle

Let's give it some movement, velocity. We create two variables called velX and velY.

```
let p
let particles = []

function setup()
{
  createCanvas(400, 400)
  p = new Particle()
  particles.push(p)
}

function draw()
{
  background(51)
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].move()
    particles[i].show()
  }
}

class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
    this.velX = random(-1, 1)
    this.velY = random(-5, -1)
  }

  show()
  {
    stroke(255)
    fill(255, 10)
    circle(this.x, this.y, 16)
```

```
  }

  move()
  {
    this.x += this.velX
    this.y += this.velY
  }
}
```

Notes

The particle (circle) will drift upwards. Every time you replay the sketch, the particle will have a different (random) velocity.

The += symbol is shorthand for:

```
this.x = this.x + this.velX
this.y = this.y + this.velY
```

Sketch E4.8 many particles

We want more than one particle. To do this quickly, we cut and paste the following code from `setup()` into `draw()`.

! You delete those lines of code in `setup()`.

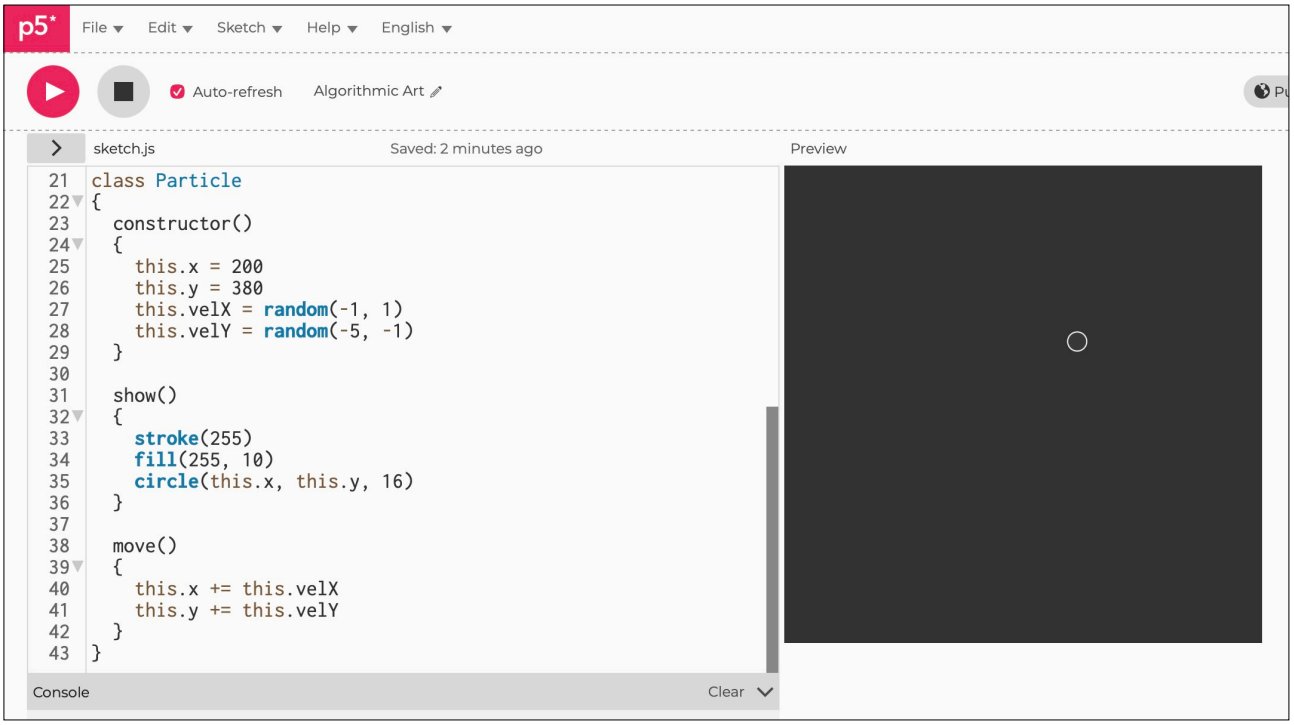
```
let p
let particles = []

function setup()
{
  createCanvas(400, 400)
  // p = new Particle()
  // particles.push(p)
}

function draw()
{
  background(51)
  p = new Particle()
  particles.push(p)
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].movement()
    particles[i].show()
  }
}

class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
    this.velX = random(-1, 1)
    this.velY = random(-5, -1)
  }
```

```
  show()
  {
    stroke(255)
    fill(255, 10)
    circle(this.x, this.y, 16)
  }

  move()
  {
    this.x += this.velX
    this.y += this.velY
  }
}
```

Notes

You get a stream of particles.

Figure E4.8



Sketch E4.9 no smoke without fire

This is a pleasing effect but still not like smoke or fire. So we will introduce some alpha. This is so we can fade it and remove it altogether when it has disappeared; otherwise, the programme will run slower and slower as we create more and more articles.

! We replace stroke(255) with noStroke().

```
let p
let particles = []

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
  p = new Particle()
  particles.push(p)
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].movement()
    particles[i].show()
  }
}

class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
    this.velX = random(-1, 1)
    this.velY = random(-5, -1)
    this.alpha = 255
  }
```

```
  show()
  {
    noStroke()
    fill(255, this.alpha)
    circle(this.x, this.y, 16)
  }

  move()
  {
    this.x += this.velX
    this.y += this.velY
    this.alpha -= 5
  }
}
```

Notes

The line of code: `this.alpha -= 5` means `this.alpha = this.alpha - 5`.

Figure E4.9

Sketch E4.10 too many particles

We need to address the issue. If we include a line to see how many particles we have created and still exist, we can use `console.log()` in `draw()`. We can give it the parameter of `particles.length`, and as you let it run for a short while, the number just keeps on going up.

```
let p
let particles = []

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
  p = new Particle()
  particles.push(p)
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].move()
    particles[i].show()
  }
  console.log(particles.length)
}

class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
    this.velX = random(-1, 1)
    this.velY = random(-5, -1)
    this.alpha = 255
  }
```

```
  show()
  {
    noStroke()
    fill(255, this.alpha)
    circle(this.x, this.y, 16)
  }

  move()
  {
    this.x += this.velX
    this.y += this.velY
    this.alpha -= 5
  }
}
```

Notes

After running this for a few minutes, you will get the idea. We will keep it there for the moment and delete it later after we have removed the particles that have faded to zero. You will notice that it is fine for a few minutes and then starts to run very slowly.

Figure E4.10

Sketch E4.11 remove dead particles

To remove the particles that are less than zero (fade), we use the splice() function in a new function that checks to see if it is zero or less. We will call this function finished().

```
let p
let particles = []

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
  p = new Particle()
  particles.push(p)
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].move()
    particles[i].show()
    if (particles[i].finished())
    {
      particles.splice(i, 1)
    }
  }
  console.log(particles.length)
}

class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
    this.velX = random(-1, 1)
    this.velY = random(-5, -1)
```

```
    this.alpha = 255
  }

  show()
  {
    noStroke()
    fill(255, this.alpha)
    circle(this.x, this.y, 16)
  }

  move()
  {
    this.x += this.velX
    this.y += this.velY
    this.alpha -= 5
  }

  finished()
  {
    return this.alpha < 0
  }
}
```

Notes

The code: `return this.alpha < 0` means the function `finished()` will return `true` when the alpha is less than zero. Think boolean (true/false). In my case, the counter stops at 52.

Figure E4.11

Sketch E4.12 backwards array

Although this looks fine, there is a problem. When you are using `splice()`, we are doing something a bit odd when we delete something from an array counting from the front. This can mean that you jump past the next one after you have deleted a particle.

The way around it is to start from the end and check through the array backwards. We start with the last element in the array, which is at `particles.length - 1` because the numbering system starts from `0` (not 1), count backwards (`i--`) while `i` is greater than zero (`> 0`). It may take a bit of thinking through, but the logic is very sound. It is a good idea to explore the nature of arrays and how they are spliced.

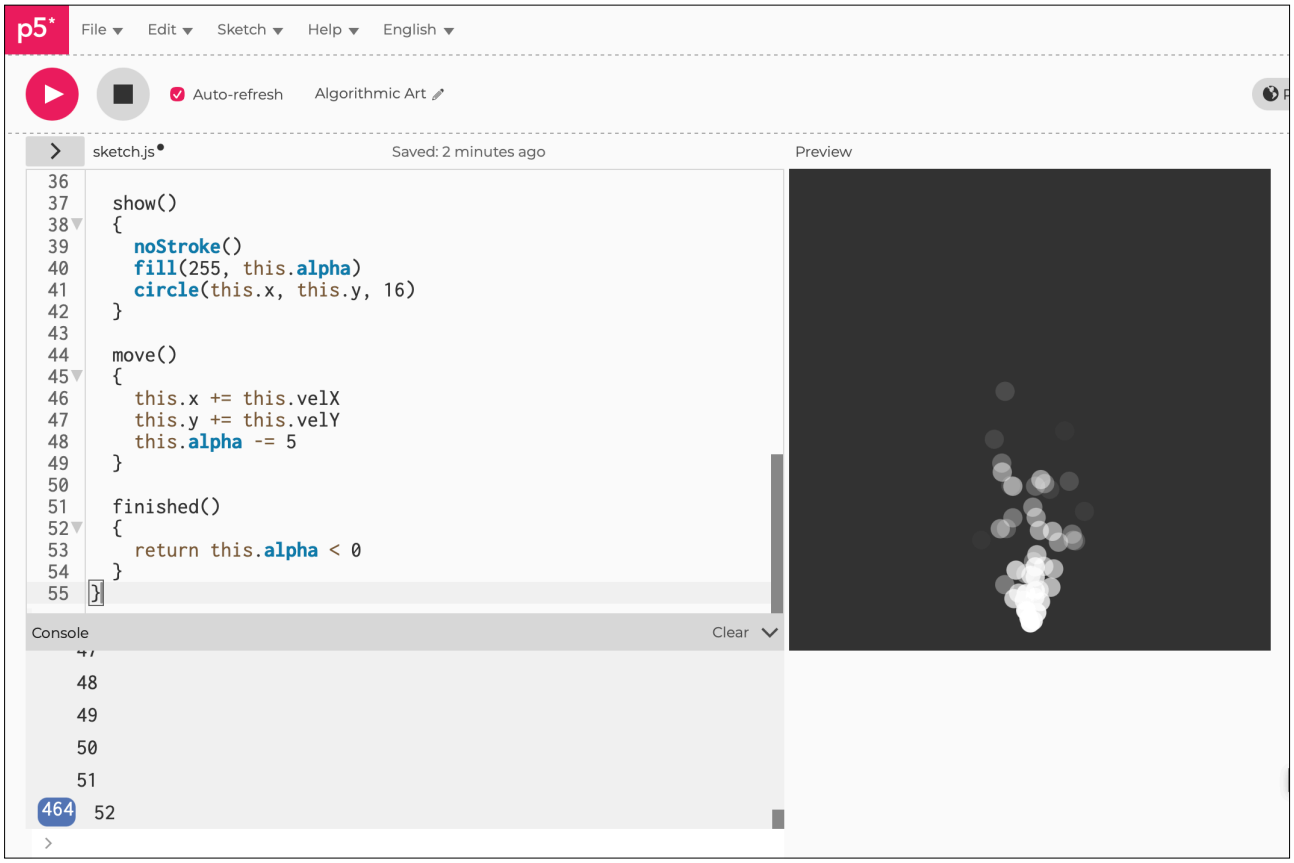
```
let p
let particles = []

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
  p = new Particle()
  particles.push(p)
  for (let i = particles.length - 1; i > 0; i--)
  {
    particles[i].move()
    particles[i].show()
    if (particles[i].finished())
    {
      particles.splice(i, 1)
    }
  }
  console.log(particles.length)
}

class Particle
{
  constructor()
```

```
  {
    this.x = 200
    this.y = 380
    this.velX = random(-1, 1)
    this.velY = random(-5, -1)
    this.alpha = 255
  }

  show()
  {
    noStroke()
    fill(255, this.alpha)
    circle(this.x, this.y, 16)
  }

  move()
  {
    this.x += this.velX
    this.y += this.velY
    this.alpha -= 5
  }

  finished()
  {
    return this.alpha < 0
  }
}
```

Notes

When you run this with the `console.log()`, you will see that it has the same effect; you may see a slight improvement. It didn't stick at 51, much cleaner and better coding.

Figure E4.12



Sketch E4.13 adding more particles

We only added one particle per frame in draw(), now let's add, say, five particles per frame.

! Remove console.log().

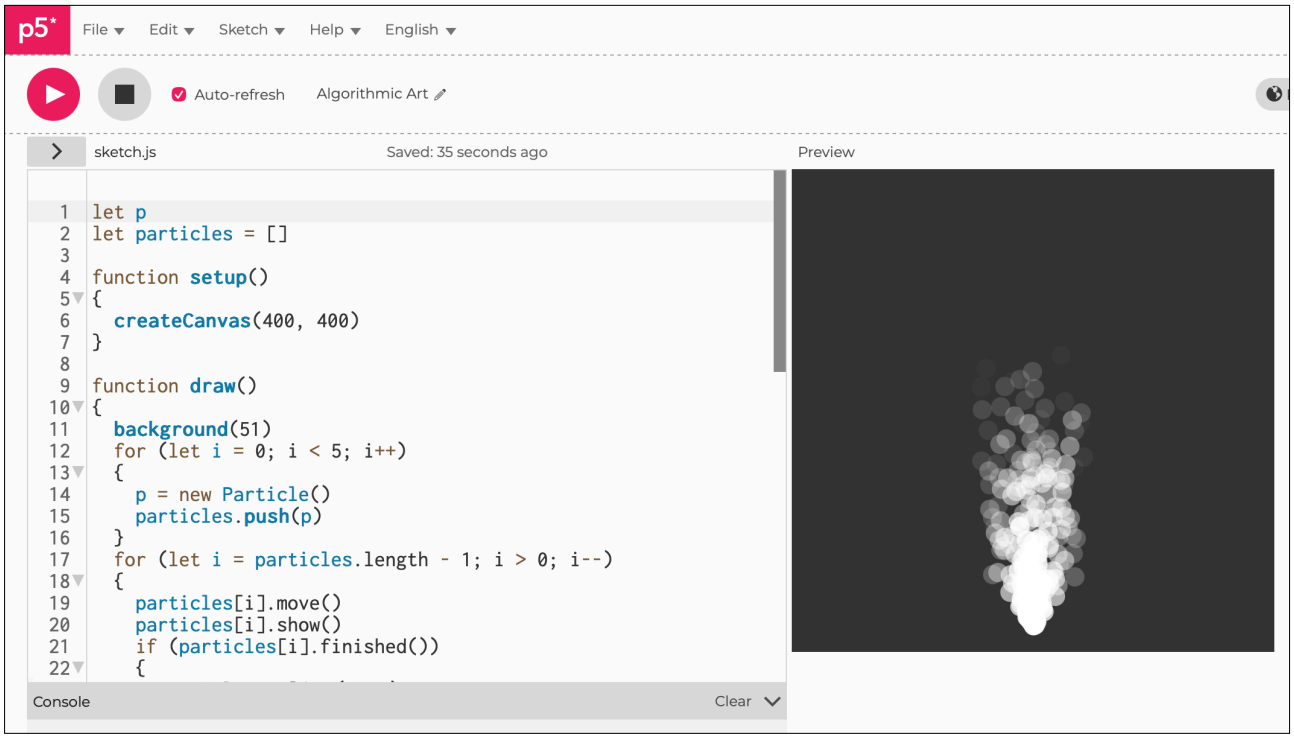
```
let p
let particles = []

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
  for (let i = 0; i < 5; i++)
  {
    p = new Particle()
    particles.push(p)
  }
  for (let i = particles.length - 1; i > 0; i--)
  {
    particles[i].move()
    particles[i].show()
    if (particles[i].finished())
    {
      particles.splice(i, 1)
    }
  }
  // console.log(particles.length)
}

class Particle
{
  constructor()
  {
```

```
    this.x = 200
    this.y = 380
    this.velX = random(-1, 1)
    this.velY = random(-5, -1)
    this.alpha = 255
  }

  show()
  {
    noStroke()
    fill(255, this.alpha)
    circle(this.x, this.y, 16)
  }

  move()
  {
    this.x += this.velX
    this.y += this.velY
    this.alpha -= 5
  }

  finished()
  {
    return this.alpha < 0
  }
}
```

Notes

It looks so much better.

Figure E4.13



Sketch E4.14 some adaptations

Just some little adjustments.

```
let p
let particles = []

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
  for (let i = 0; i < 5; i++)
  {
    p = new Particle()
    particles.push(p)
  }
  for (let i = particles.length - 1; i > 0; i--)
  {
    particles[i].move()
    particles[i].show()
    if (particles[i].finished())
    {
      particles.splice(i, 1)
    }
  }
}

class Particle
{
  constructor()
  {
    this.x = 200
    this.y = 380
    this.velX = random(-1, 1)
    this.velY = random(-6, -2)
```

```
    this.alpha = 255
    this.radius = 20
  }

  show()
  {
    noStroke()
    fill(255, this.alpha)
    circle(this.x, this.y, this.radius)
  }

  move()
  {
    this.x += this.velX
    this.y += this.velY
    this.alpha -= 5
    this.radius -= 0.001
  }

  finished()
  {
    return this.alpha < 0
  }
}
```

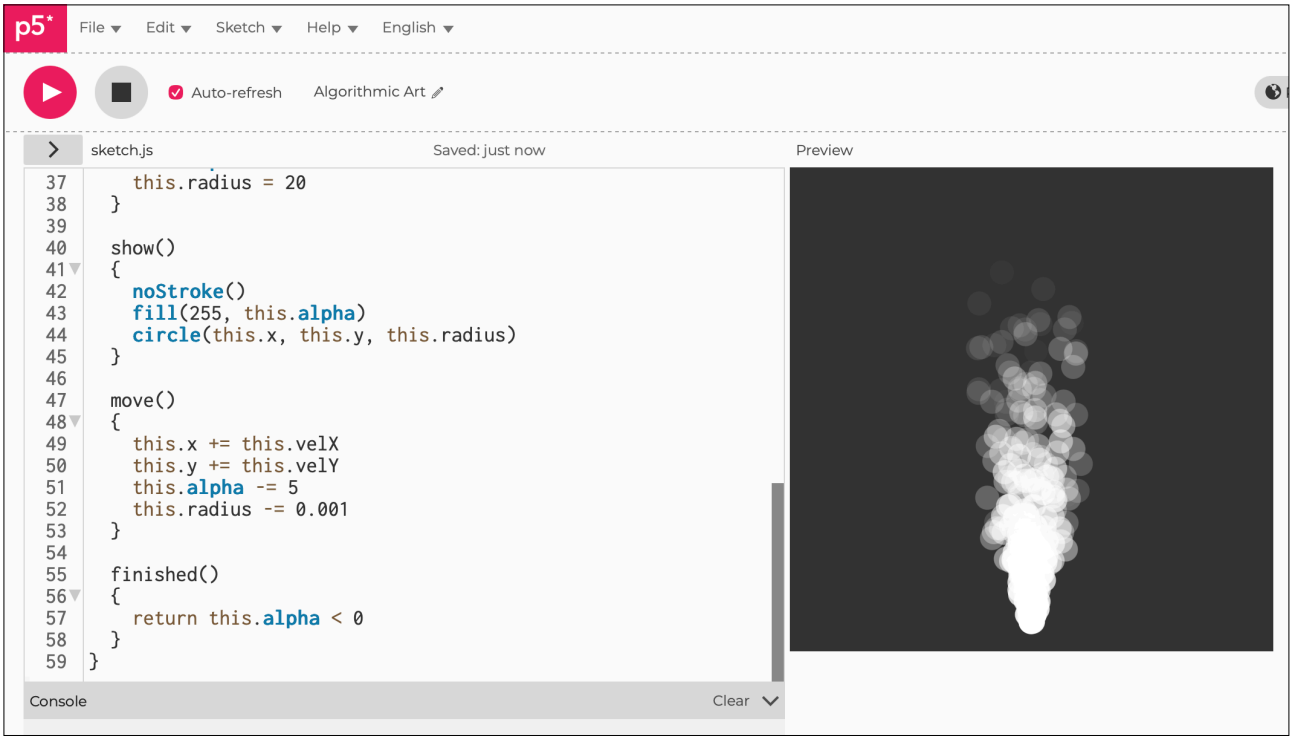
Notes

Even better.

Challenges

1. Change the diameter randomly.
2. Add some colour and then fade.

Figure E4.14




Algorithmic
Art
Module E
Unit #5
Fireworks
Display

Module E Unit #5: fireworks display

Creating a firework display using particles. This builds on the previous unit covering simple particle systems. This is a bit all over the place in terms of simple tutorials; that doesn't mean that it is no good, rather it will need refactoring in the future. The final result is brilliant, but we do go round the houses a little bit.

We will be adding files and working from those separate files for some of the classes. To make it clear, I will add a heading to each sketch with the file it belongs to, and I will make clear references where we switch from one file to another. If you have never done this before, don't worry; you will very quickly get the hang of it.

We will be adding two more files: `particle.js` and `firework.js`. see below.

Sketch E5.1 starting sketch

! Now in sketch.js

Our starting sketch in the normal place.

sketch.js

```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(100)
}
```

Adding particle.js File

Add this file using the drop down menu (see relevant unit for more detail).

Figure 16: Add file called particle.js

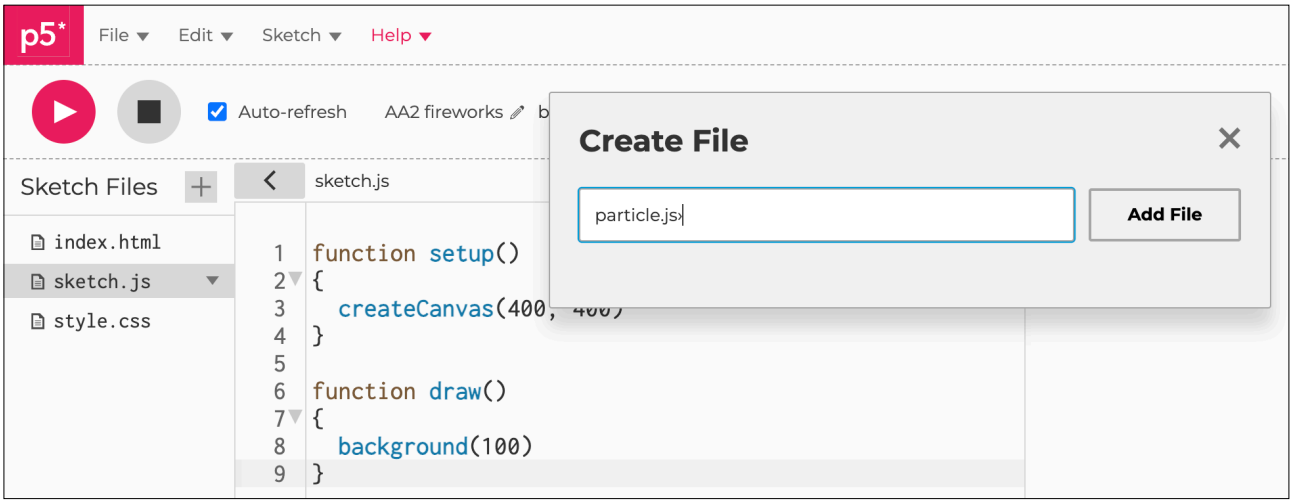
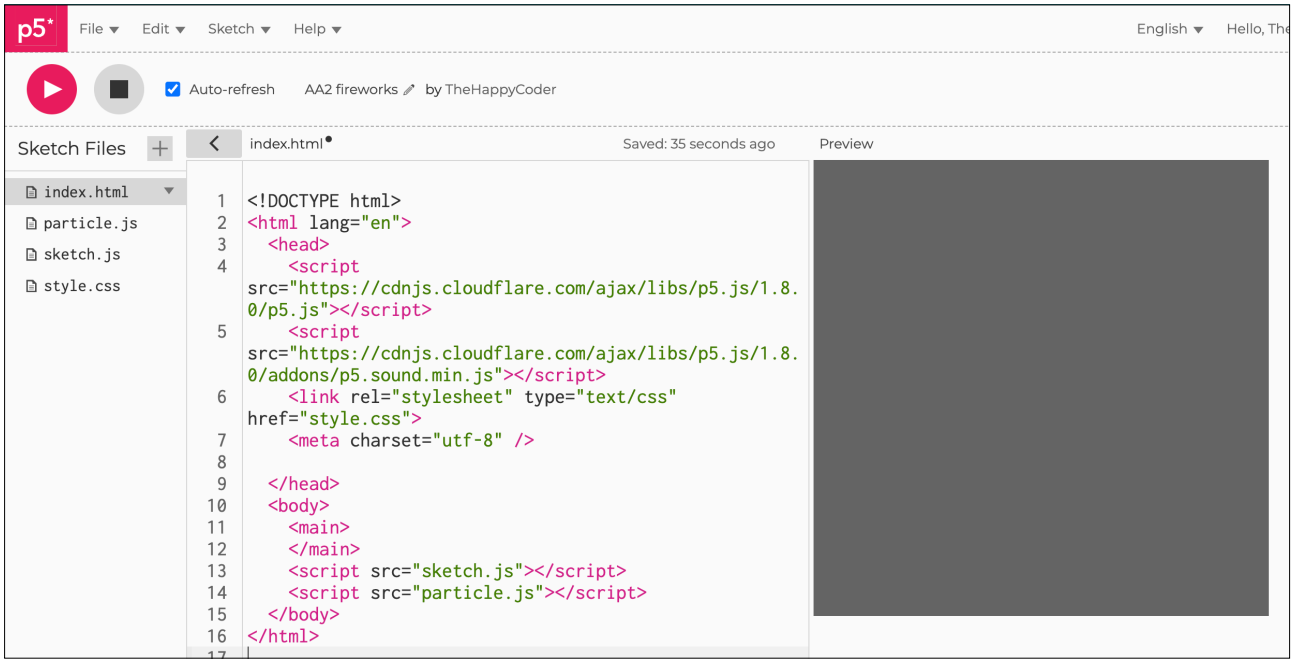


Figure 17: Add line of code into index.html



Sketch E5.2 index.html

! The index.html file

In the index.html file, we need to reference the particle.js file. Notice we are suing the 2.x version of p5.js (recommended)

index.html

```
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.2.2/lib/p5.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
    <script src="particle.js"></script>
</body></html>
```

Sketch E5.3 create vectors

! The particle.js file

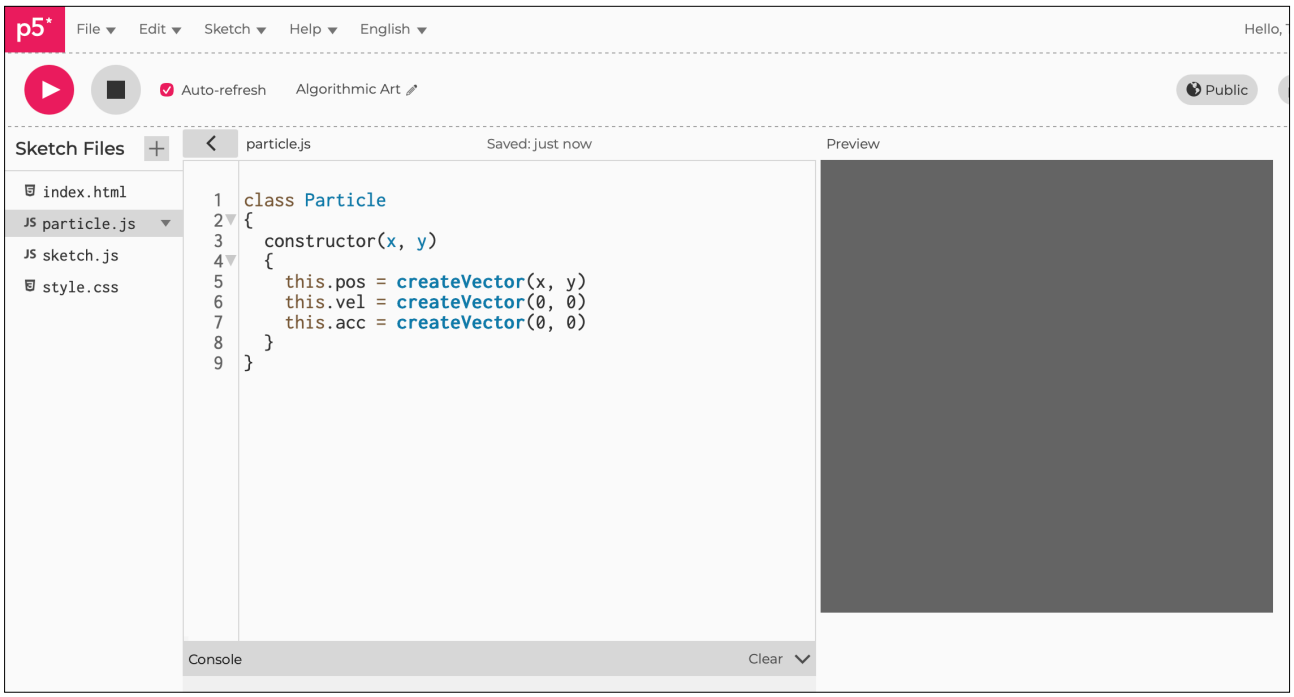
In a previous module, you were introduced to classes, so we create a particle class with a constructor that takes two variables (`x, y`) with three vectors:

A) position (`pos`)
B) velocity (`vel`)
C) acceleration (`acc`)

particle.js

```
class Particle
{
  constructor(x, y)
  {
    this.pos = createVector(x, y)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }
}
```

Figure E5.3


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
Public
Sketch Files
index.html
particle.js
sketch.js
style.css
particle.js
Saved: just now
Preview
class Particle
{
constructor(x, y)
{
this.pos = createVector(x, y)
this.vel = createVector(0, 0)
this.acc = createVector(0, 0)
}
}
Console
Clear

Sketch E5.4 move() function

We create a `move()` function which adds the velocity to the position of the particle.

particle.js

```
class Particle
{
  constructor(x, y)
  {
    this.pos = createVector(x, y)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.pos.add(this.vel)
  }
}
```

Sketch E5.5 creating a force

To get the acceleration, we create a force. It will be fired upwards and fall under gravity; both are forces. We create a variable called force within which we will apply it, hence the function `applyForce()`. This is so the acceleration can accumulate.

From the basic equation `force = mass x acceleration`, but we can ignore mass, so `force = acceleration`.

particle.js

```
class Particle
{
  constructor(x, y)
  {
    this.pos = createVector(x, y)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    this.pos.add(this.vel)
  }
}
```

Sketch E5.6 adding the force

Adding this to the `move()` function. We multiply the acceleration (`acc`) by zero so that on each frame the acceleration is constant. (You can comment out this line to see what it does later.)

particle.js

```
class Particle
{
  constructor(x, y)
  {
    this.pos = createVector(x, y)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }
}
```

Sketch E5.7 draw the particle

We now need to draw the particle, and so we create a `show()` function.

particle.js

```
class Particle
{
  constructor(x, y)
  {
    this.pos = createVector(x, y)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
  }
}
```

Sketch E5.8 size and colour

! The sketch.js file

You will notice that nothing happens when you run the sketch. So we need to move back to the main sketch.js. Here we give the particle a white colour (255) and a size (4). To test that this works, we add a variable called `firework` and use the `Particle class` to draw it.

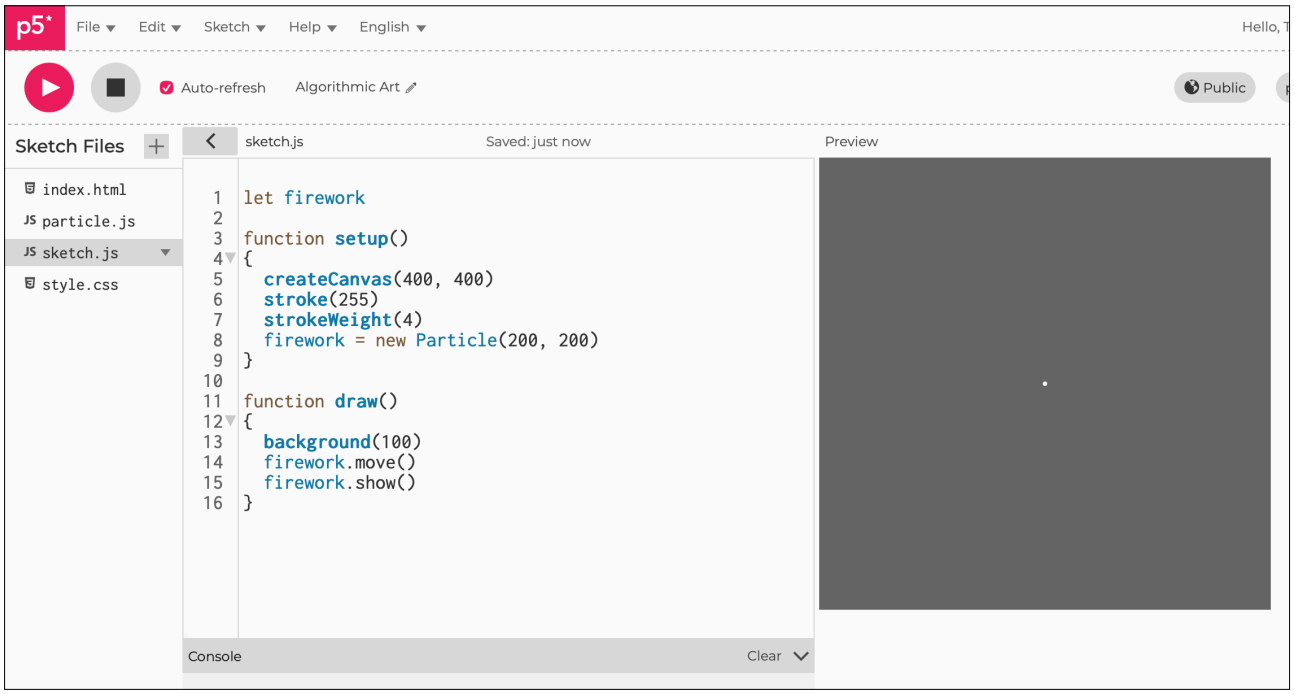
sketch.js

```
let firework

function setup()
{
  createCanvas(400, 400)
  stroke(255)
  strokeWeight(4)
  firework = new Particle(200, 200)
}

function draw()
{
  background(100)
  firework.move()
  firework.show()
}
```

Notes

A new particle is drawn in the centre, something to see at last.

Figure E5.8



Sketch E5.9 moving upwards

Let's get it moving upwards from a new starting position on the ground at random.

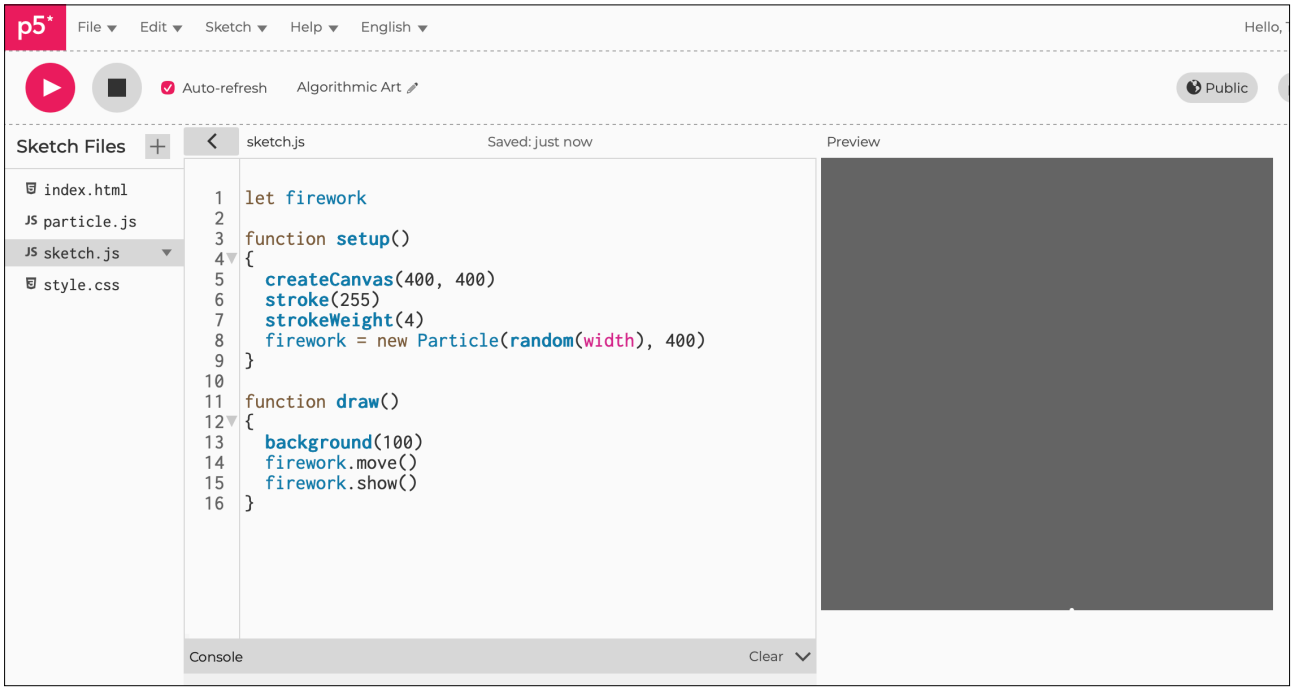
sketch.js

```
let firework

function setup()
{
  createCanvas(400, 400)
  stroke(255)
  strokeWeight(4)
  firework = new Particle(random(width), 400)
}

function draw()
{
  background(100)
  firework.move()
  firework.show()
}
```

Notes

You can just see it sitting at the bottom.

Figure E5.9



Sketch E5.10 upwards velocity

! The particle.js file

In the Particle class, we need to give an upwards velocity of -5.

particle.js

```
class Particle
{
  constructor(x, y)
  {
    this.pos = createVector(x, y)
    this.vel = createVector(0, -5)
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
  }
}
```

Notes

What you should see is the particle start at the bottom and move upwards and off the canvas.

Figure E5.10

Sketch E5.11 gravity

! In sketch.js

Now let us add `gravity`, which is a force, and it points down (`0.1`).

sketch.js

```
let firework
let gravity

function setup()
{
  createCanvas(400, 400)
  gravity = createVector(0, 0.1)
  stroke(255)
  strokeWeight(4)
  firework = new Particle(random(width), 400)
}

function draw()
{
  background(100)
  firework.applyForce(gravity)
  firework.move()
  firework.show()
}
```

Notes

You notice it doesn't go very high, so let's increase the vertical velocity to 8.

Sketch E5.12 higher

! Back to the particle.js file

Making it go a little higher.

particle.js

```
class Particle
{
  constructor(x, y)
  {
    this.pos = createVector(x, y)
    this.vel = createVector(0, -8)
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
  }
}
```

Sketch E5.13 lots of fireworks

! The sketch.js file

At the moment, we have one firework shooting up, but we want lots of them (and then explode later on), so we create an array of fireworks.

sketch.js

```
let fireworks = []
let gravity

function setup()
{
  createCanvas(400, 400)
  gravity = createVector(0, 0.1)
  stroke(255)
  strokeWeight(4)
  firework = new Particle(random(width), 400)
}

function draw()
{
  background(100)
  firework.applyForce(gravity)
  firework.move()
  firework.show()
}
```

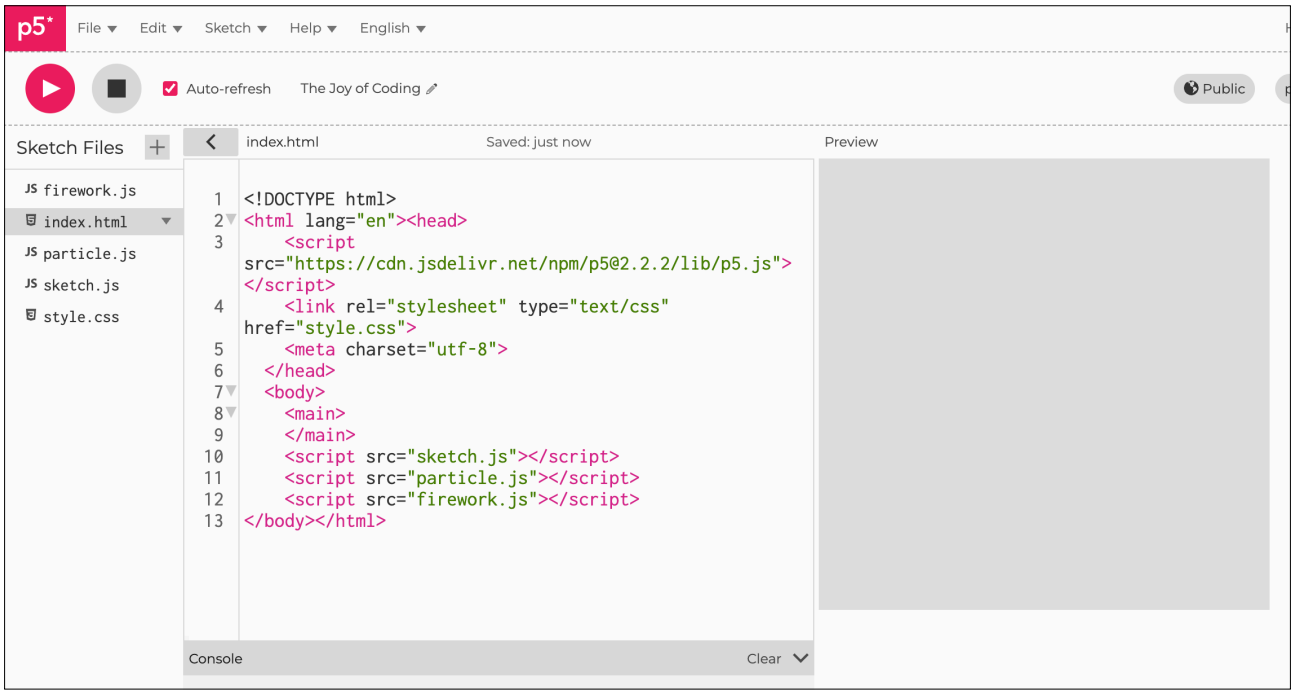
Notes

We still have the one particle going and returning to the ground.

Adding another File, firework.js

To facilitate this, we will create another file called `fireworks` and a corresponding class.

Figure 3: add another file called firework.js



Sketch E5.14 explode

! The firework.js file

We create a firework that is going to explode.

firework.js

```
class Firework
{
  constructor()
  {
    this.firework = new Particle(random(width), height)
  }

  move()
  {
    this.firework.applyForce(gravity)
    this.firework.move()
  }

  show()
  {
    this.firework.show()
  }
}
```

Sketch E5.15 remove redundant code

We need to reference these in the main sketch. No changes will be seen just yet.

! Remove the redundant lines of code (commented //).

sketch.js

```
let fireworks = []
let gravity

function setup()
{
  createCanvas(400, 400)
  gravity = createVector(0, 0.1)
  fireworks.push(new Firework())
  stroke(255)
  strokeWeight(4)
  // firework = new Particle(random(width), 400)
}

function draw()
{
  background(100)
  for (let i = 0; i < fireworks.length; i++)
  {
    fireworks[i].move()
    fireworks[i].show()
  }
  // firework.applyForce(gravity)
  // firework.move()
  // firework.show()
}
```

Sketch E5.16 cleaned up a bit

Which should look like this now.

sketch.js

```
let fireworks = []
let gravity

function setup()
{
  createCanvas(400, 400)
  gravity = createVector(0, 0.1)
  fireworks.push(new Firework())
  stroke(255)
  strokeWeight(4)
}

function draw()
{
  background(100)
  for (let i = 0; i < fireworks.length; i++)
  {
    fireworks[i].move()
    fireworks[i].show()
  }
}
```

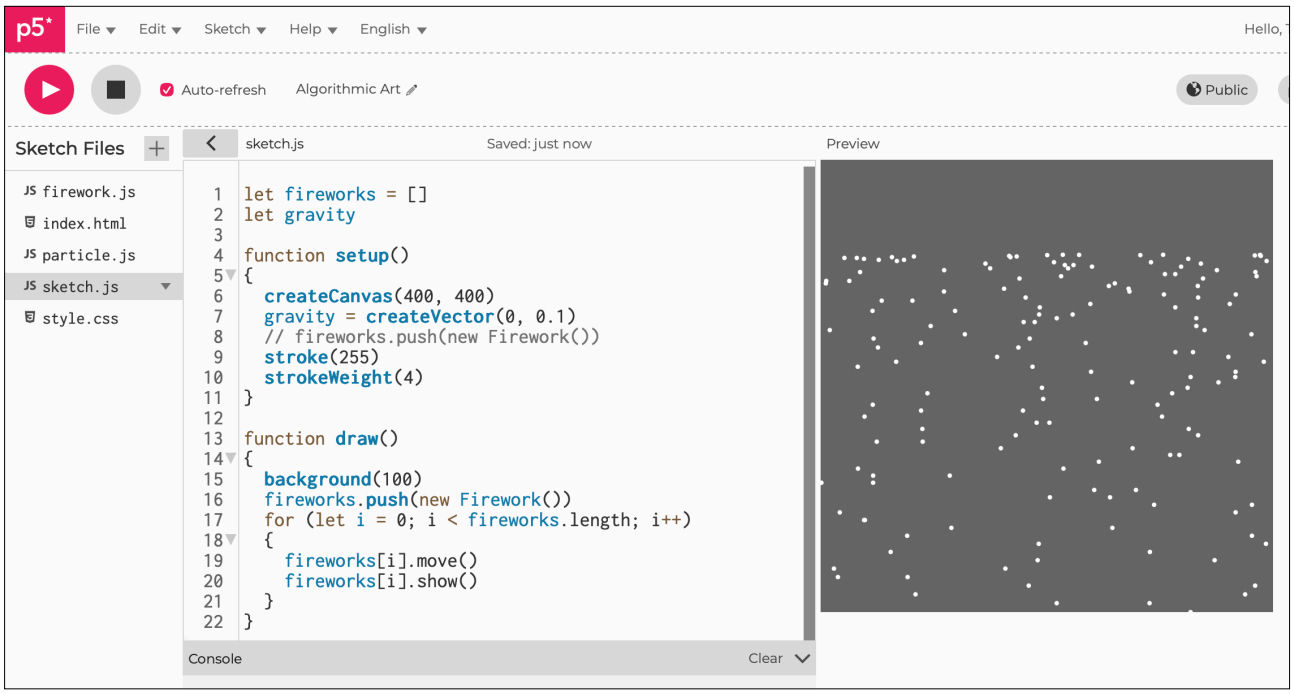
Sketch E5.17 from setup() to draw()

If we put the line of code creating an array of fireworks from `setup()` into `draw()`, we get lots of fireworks.

sketch.js

```
let fireworks = []
let gravity

function setup()
{
  createCanvas(400, 400)
  gravity = createVector(0, 0.1)
  // fireworks.push(new Firework())
  stroke(255)
  strokeWeight(4)
}

function draw()
{
  background(100)
  fireworks.push(new Firework())
  for (let i = 0; i < fireworks.length; i++)
  {
    fireworks[i].move()
    fireworks[i].show()
  }
}
```

Figure E5.17


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
Public
Sketch Files
firework.js
index.html
particle.js
sketch.js
style.css
sketch.js
Saved: just now
Preview
let fireworks = []
let gravity
function setup()
{
createCanvas(400, 400)
gravity = createVector(0, 0.1)
// fireworks.push(new Firework())
stroke(255)
strokeWeight(4)
}
function draw()
{
background(100)
fireworks.push(new Firework())
for (let i = 0; i < fireworks.length; i++)
{
fireworks[i].move()
fireworks[i].show()
}
}
Console
Clear

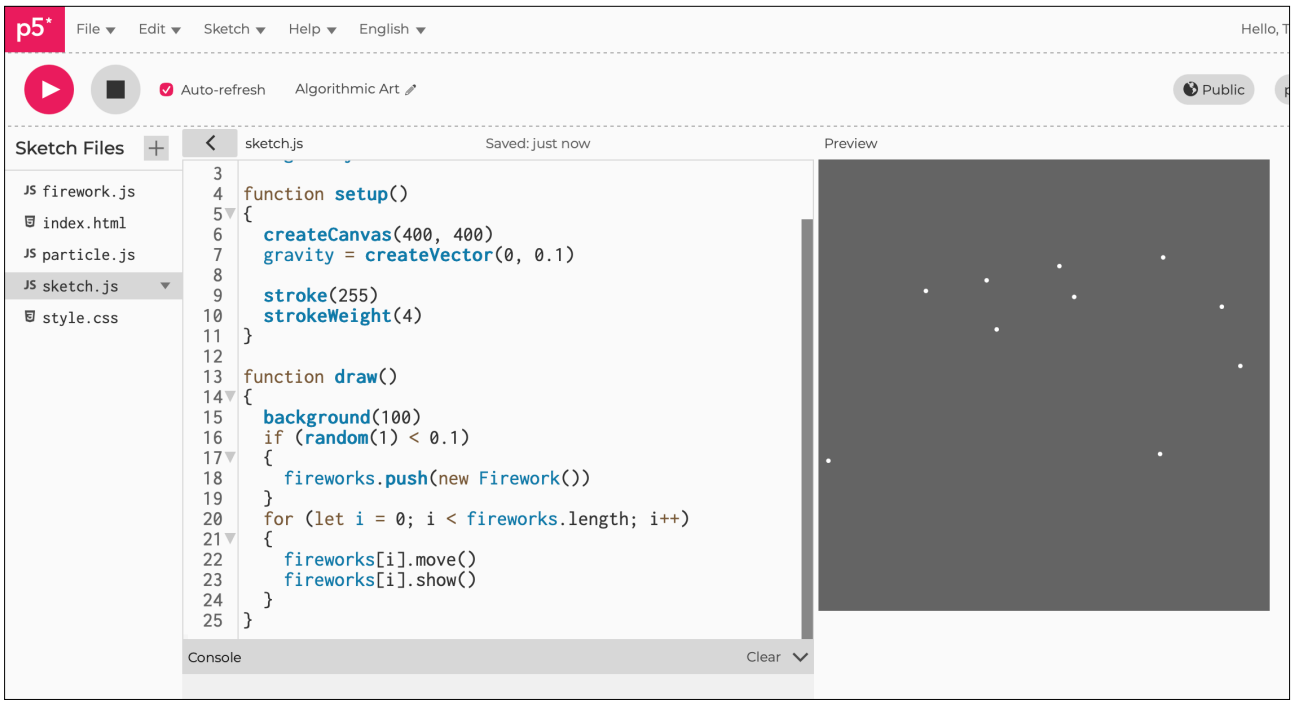
Sketch E5.18 not so many

We want more than one but not hundreds, so if we put the creation of a firework in the `draw()` function, we can limit how many it draws by using a random selection. In this case, every frame there is a `10%` chance of creating a firework.

sketch.js

```
let fireworks = []
let gravity

function setup()
{
  createCanvas(400, 400)
  gravity = createVector(0, 0.1)

  stroke(255)
  strokeWeight(4)
}

function draw()
{
  background(100)
  if (random(1) < 0.1)
  {
    fireworks.push(new Firework())
  }
  for (let i = 0; i < fireworks.length; i++)
  {
    fireworks[i].move()
    fireworks[i].show()
  }
}
```

Figure E5.18


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
Public
Sketch Files
firework.js
index.html
particle.js
sketch.js
style.css
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400)
gravity = createVector(0, 0.1)
stroke(255)
strokeWeight(4)
}
function draw()
{
background(100)
if (random(1) < 0.1)
{
fireworks.push(new Firework())
}
for (let i = 0; i < fireworks.length; i++)
{
fireworks[i].move()
fireworks[i].show()
}
}
Console
Clear

Sketch E5.19 random height

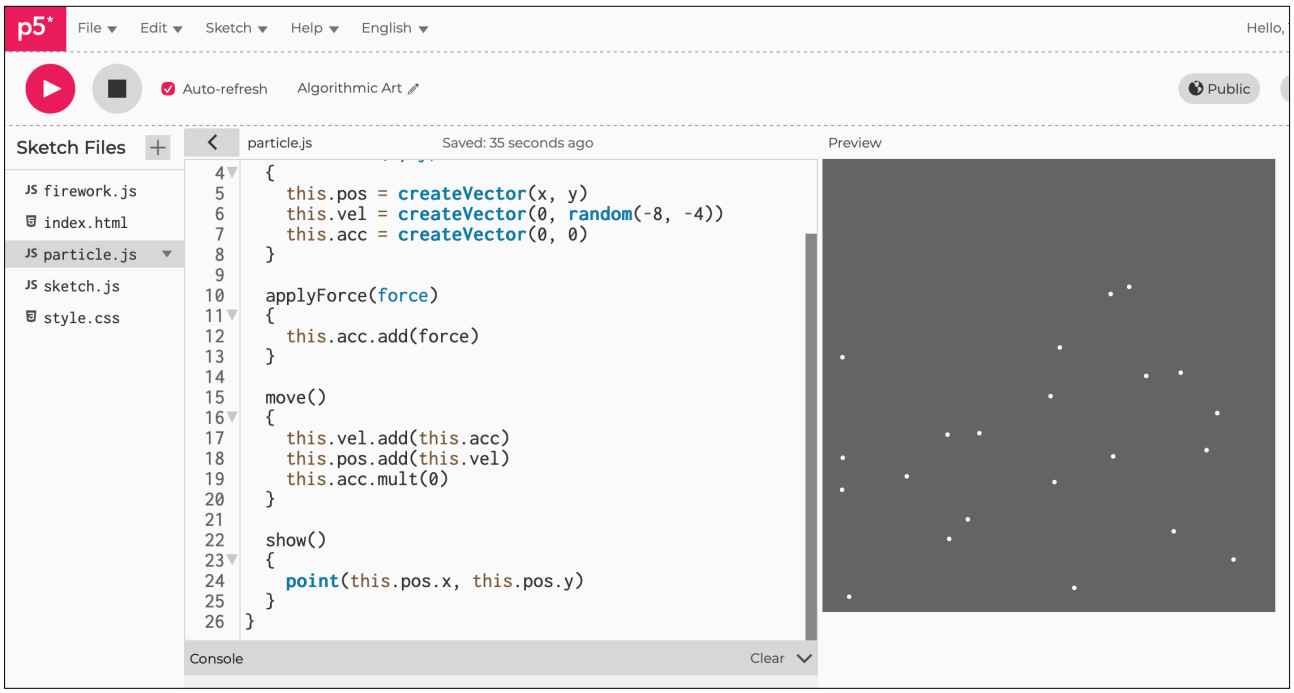
! The particle.js file

Let's randomise the height as well, so in particle.js we do the following. We can play with these later.

particle.js

```
class Particle
{
  constructor(x, y)
  {
    this.pos = createVector(x, y)
    this.vel = createVector(0, random(-8, -4))
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
  }
}
```

Figure E5.19


p5*
File
Edit
Sketch
Help
English
Hello,
Auto-refresh
Algorithmic Art
Public
Sketch Files
firework.js
index.html
particle.js
sketch.js
style.css
particle.js
Saved: 35 seconds ago
Preview
{
this.pos = createVector(x, y)
this.vel = createVector(0, random(-8, -4))
this.acc = createVector(0, 0)
}
applyForce(force)
{
this.acc.add(force)
}
move()
{
this.vel.add(this.acc)
this.pos.add(this.vel)
this.acc.mult(0)
}
show()
{
point(this.pos.x, this.pos.y)
}
}
Console
Clear

Sketch E5.20 top of travel

! The firework.js file

We want to explode the firework at the point it reaches the top of its travel. This is when the velocity is zero. As it goes up, it is negative, becomes zero, and then positive on the way down. First, we only draw the firework if it exists by using the if statement (`this.firework`) in both `move()` and `show()`.

firework.js

```
class Firework
{
  constructor()
  {
    this.firework = new Particle(random(width), height)
  }

  move()
  {
    if (this.firework)
    {
      this.firework.applyForce(gravity)
      this.firework.move()
    }
  }

  show()
  {
    if (this.firework)
    {
      this.firework.show()
    }
  }
}
```

Sketch E5.21 null

Now, to make it null, which will make it disappear. This isn't permanent because we really want it to explode, but we will get to that in a moment.

firework.js

```
class Firework
{
  constructor()
  {
    this.firework = new Particle(random(width), height)
  }

  move()
  {
    if (this.firework)
    {
      this.firework.applyForce(gravity)
      this.firework.move()
      if (this.firework.vel.y >= 0)
      {
        this.firework = null
      }
    }
  }

  show()
  {
    if (this.firework)
    {
      this.firework.show()
      if (this.firework.vel.y >= 0)
      {
        this.firework = null
      }
    }
  }
}
```

Notes

As it reaches its highest point, it then disappears.

Sketch E5.22 boolean

You get the idea, but we don't want to just remove them; we want to explode them. So we will change what happens at the top of the to a boolean expression. It will have the same effect. We will call the boolean variable `exploded`.

firework.js

```
class Firework
{
  constructor()
  {
    this.firework = new Particle(random(width), height)
    this.exploded = false
  }

  move()
  {
    if (!this.exploded)
    {
      this.firework.applyForce(gravity)
      this.firework.move()
      if (this.firework.vel.y >= 0)
      {
        this.exploded = true
      }
    }
  }

  show()
  {
    if (!this.exploded)
    {
      this.firework.show()
      if (this.firework.vel.y >= 0)
      {
        this.exploded = true
      }
    }
  }
```

```
}
```

Notes

You could start with it being true, and then you would have to reverse all the logic in the rest of the sketch. It's a preference thing.

Sketch E5.23 explode() function

As we want to have the firework explode, we will create a function called `explode()`. We want it to explode when it reaches the very top (when it is null or true). In this function, we are going to make `100` particles.

! Removed the following lines of code:

```
if (this.firework.vel.y >= 0)
{
  this.exploded = true
}
```

firework.js

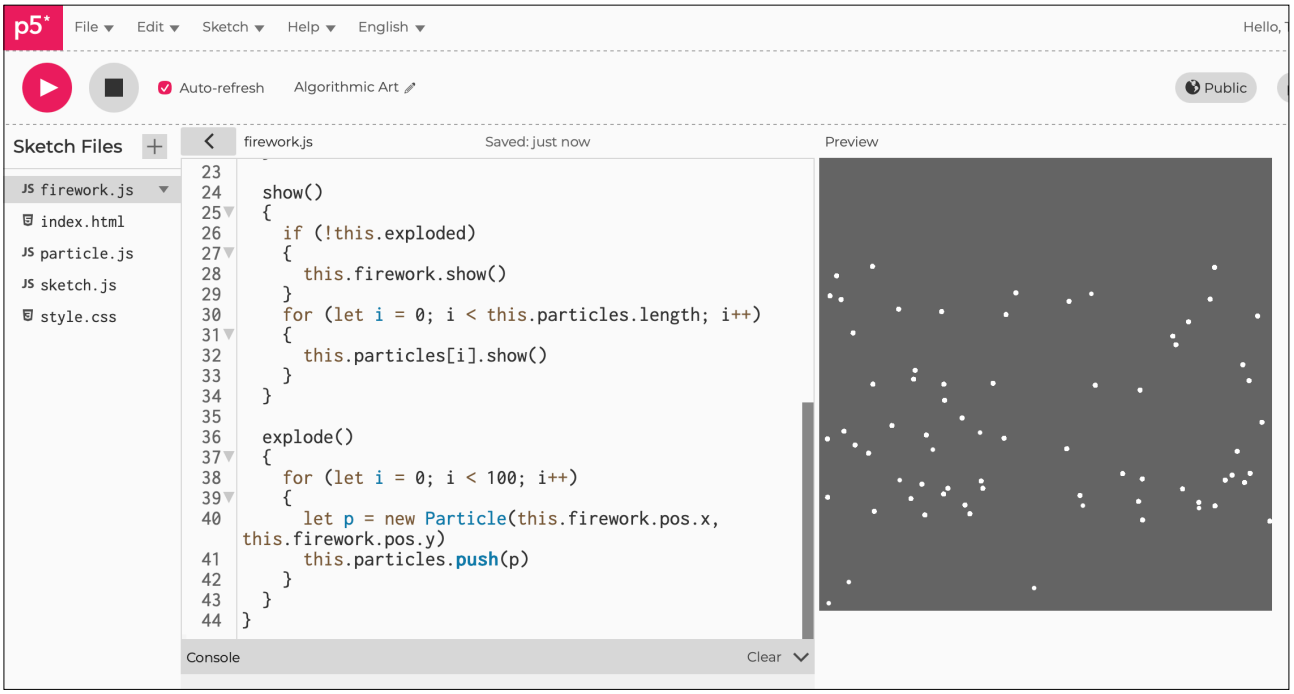
```
class Firework
{
  constructor()
  {
    this.firework = new Particle(random(width), height)
    this.exploded = false
    this.particles = []
  }

  move()
  {
    if (!this.exploded)
    {
      this.firework.applyForce(gravity)
      this.firework.move()
      if (this.firework.vel.y >= 0)
      {
        this.exploded = true
        this.explode()
      }
    }
  }

  show()
  {
    if (!this.exploded)
    {
      this.firework.show()
```

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

  explode()
  {
    for (let i = 0; i < 100; i++)
    {
      let p = new Particle(this.firework.pos.x, this.firework.pos.y)
      this.particles.push(p)
    }
  }
}
```

Notes

What you see is the particles going up, and then the 100 particles just stay put. We need to apply a force to those particles and move them.

Figure E5.23



Sketch E5.24 adding gravity

Applying gravity to the 100 particles.

firework.js

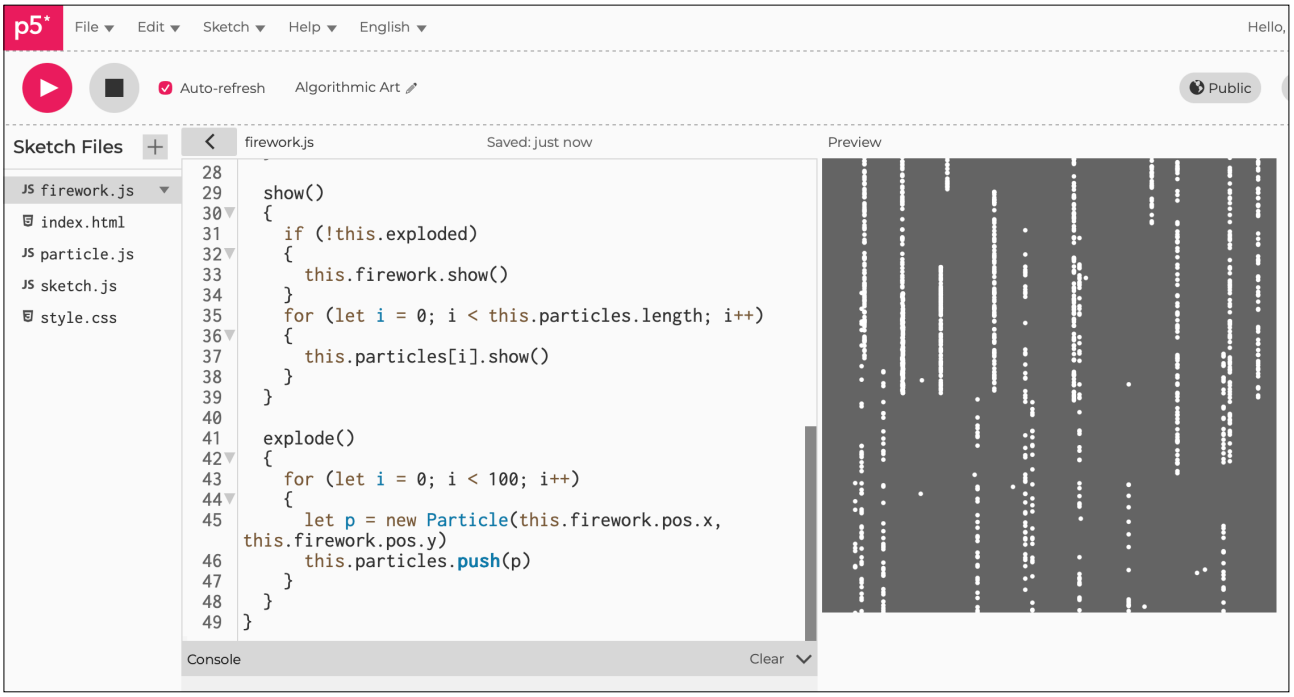
```
class Firework
{
  constructor()
  {
    this.firework = new Particle(random(width), height)
    this.exploded = false
    this.particles = []
  }

  move()
  {
    if (!this.exploded)
    {
      this.firework.applyForce(gravity)
      this.firework.move()
      if (this.firework.vel.y >= 0)
      {
        this.exploded = true
        this.explode()
      }
    }
    for (let i = 0; i < this.particles.length; i++)
    {
      this.particles[i].applyForce(gravity)
      this.particles[i].move()
    }
  }

  show()
  {
    if (!this.exploded)
    {
      this.firework.show()
    }
```

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

  explode()
  {
    for (let i = 0; i < 100; i++)
    {
      let p = new Particle(this.firework.pos.x, this.firework.pos.y)
      this.particles.push(p)
    }
  }
}
```

Notes

They are exploding but only moving vertically.

Figure E5.24



Sketch E5.25 two types of particle

! The particle.js file

In particle.js, we add another argument. This is because we have two types of particle: one for the going-up bit (rocket firework) and the explode bit. This allows us to give different vector values to the two types of particle. `firework` becomes a boolean variable.

We use `p5.Vector.random2D()` so that we have a random vector in both the horizontal and the vertical.

particle.js

```
class Particle
{
  constructor(x, y, firework)
  {
    this.pos = createVector(x, y)
    if (firework)
    {
      this.vel = createVector(0, random(-8, -4))
    }
    else
    {
      this.vel = p5.Vector.random2D()
    }
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }
```

```
  show()
  {
    point(this.pos.x, this.pos.y)
  }
}
```

Notes

You get them bubbling at the bottom of the canvas. We will raise them into the air.

Figure E5.25

Sketch E5.26 true

! The firework.js

In the firework.js sketch, we need to add the boolean to be true for the particle.

firework.js

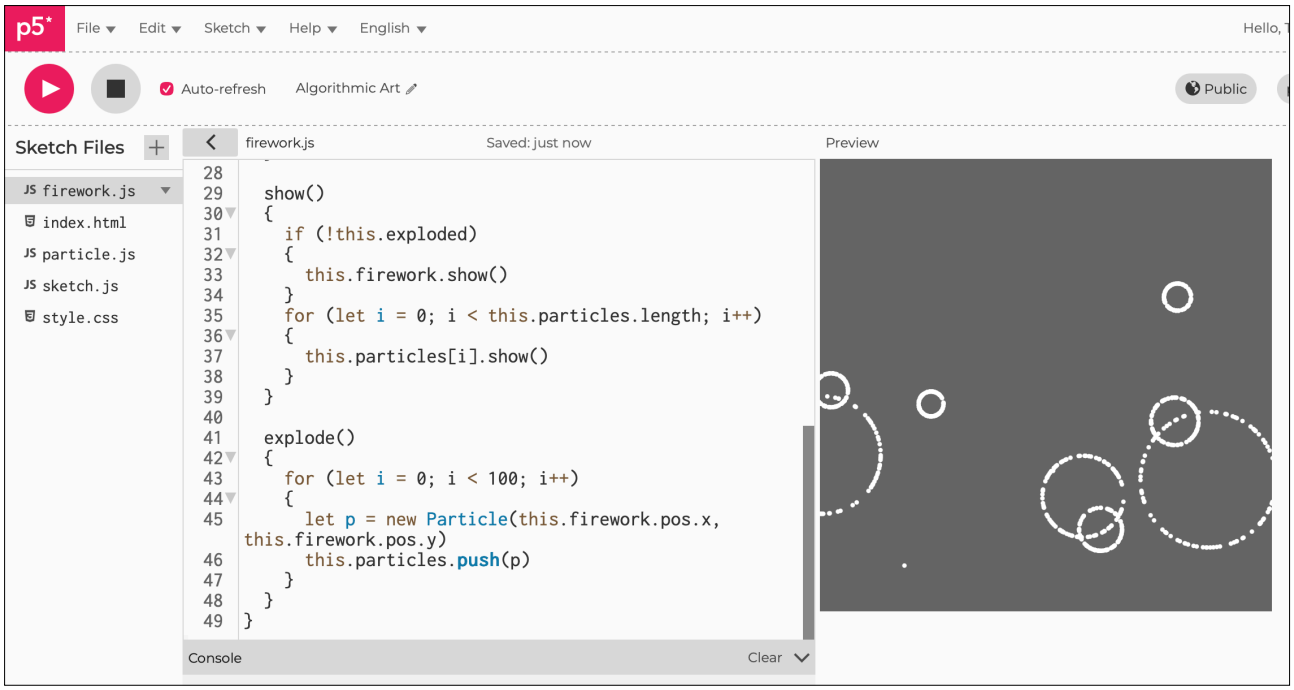
```
class Firework
{
  constructor()
  {
    this.firework = new Particle(random(width), height, true)
    this.exploded = false
    this.particles = []
  }

  move()
  {
    if (!this.exploded)
    {
      this.firework.applyForce(gravity)
      this.firework.move()
      if (this.firework.vel.y >= 0)
      {
        this.exploded = true
        this.explode()
      }
    }
    for (let i = 0; i < this.particles.length; i++)
    {
      this.particles[i].applyForce(gravity)
      this.particles[i].move()
    }
  }

  show()
  {
    if (!this.exploded)
    {
```

```
      this.firework.show()
    }
    for (let i = 0; i < this.particles.length; i++)
    {
      this.particles[i].show()
    }
  }

  explode()
  {
    for (let i = 0; i < 100; i++)
    {
      let p = new Particle(this.firework.pos.x, this.firework.pos.y)
      this.particles.push(p)
    }
  }
}
```

Notes

When you run it, you get something like this. Also, it starts running slower and slower as you create more and more particles. It looks rather nice as it is.

Figure E5.26



Sketch E5.27 random velocity

! The particle.js

We have nice circles, but we want not just random direction but also random velocity.

particle.js

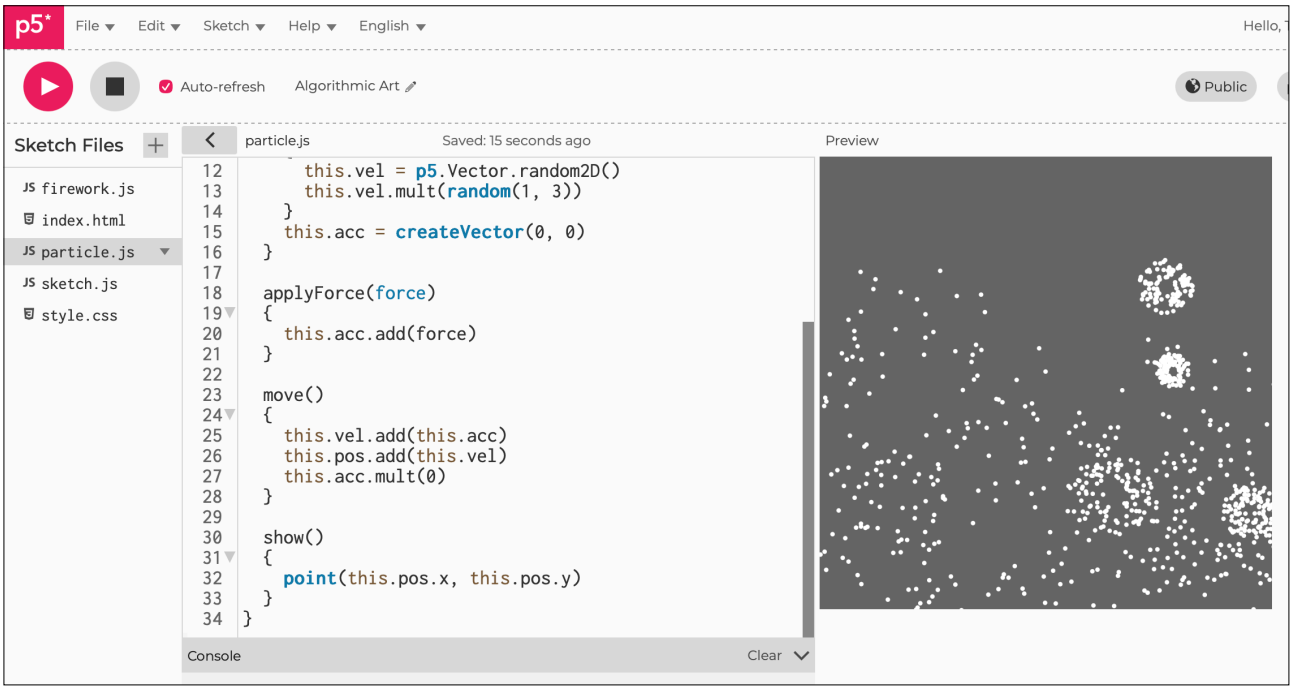
```
class Particle
{
  constructor(x, y, firework)
  {
    this.pos = createVector(x, y)
    if (firework)
    {
      this.vel = createVector(0, random(-8, -4))
    }
    else
    {
      this.vel = p5.Vector.random2D()
      this.vel.mult(random(1, 3))
    }
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
```

```
  }
}
```

Notes

They now explode a little bit more realistically.

Figure E5.27



Sketch E5.28 slow down

We want to keep track of those firework particles and cause them to not just fly away, also to slow down.

particle.js

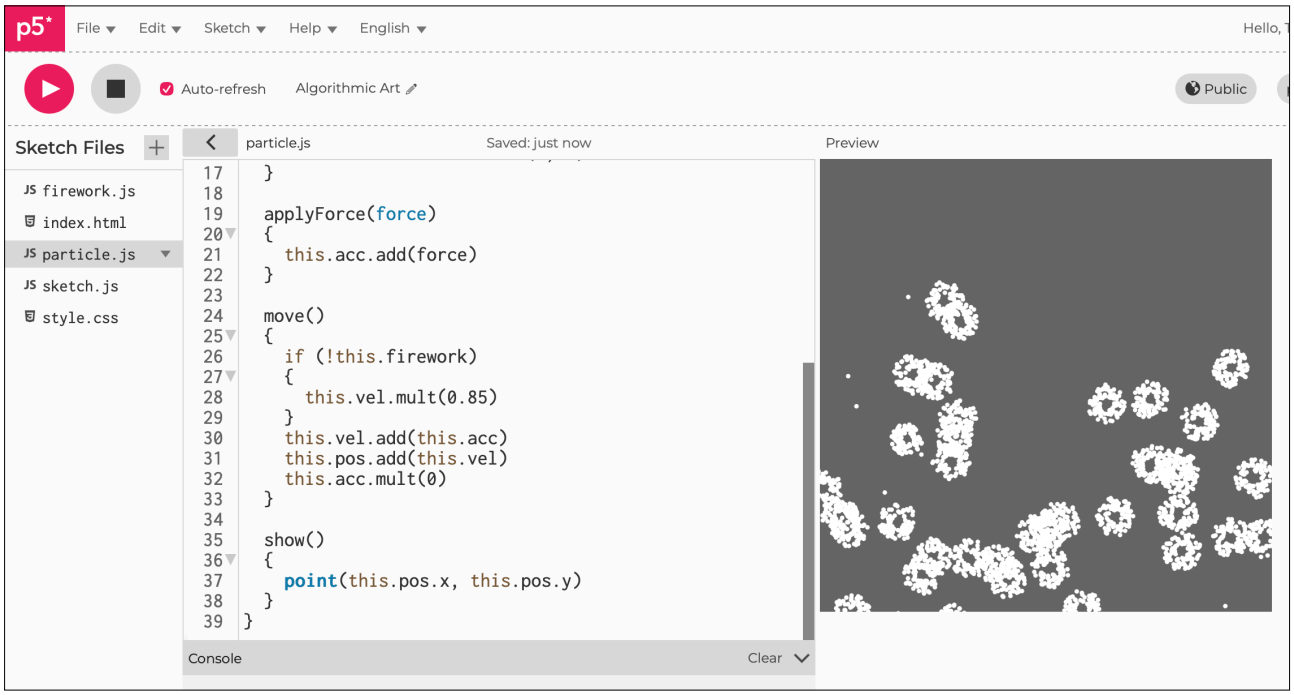
```
class Particle
{
  constructor(x, y, firework)
  {
    this.pos = createVector(x, y)
    this.firework = firework
    if (this.firework)
    {
      this.vel = createVector(0, random(-8, -4))
    }
    else
    {
      this.vel = p5.Vector.random2D()
      this.vel.mult(random(1, 3))
    }
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    if (!this.firework)
    {
      this.vel.mult(0.85)
    }
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }
```

```
  show()
  {
    point(this.pos.x, this.pos.y)
  }
}
```

Notes

This has something of a bizarre effect, because the particles (or sparks) would fade quite soon after the initial explosion.

Figure E5.28



Sketch E5.29 fade

To get the fade effect, we create a new variable called `lifespan`. It will decrease by `4` on each frame and it will be drawn as a `stroke()` value.

particle.js

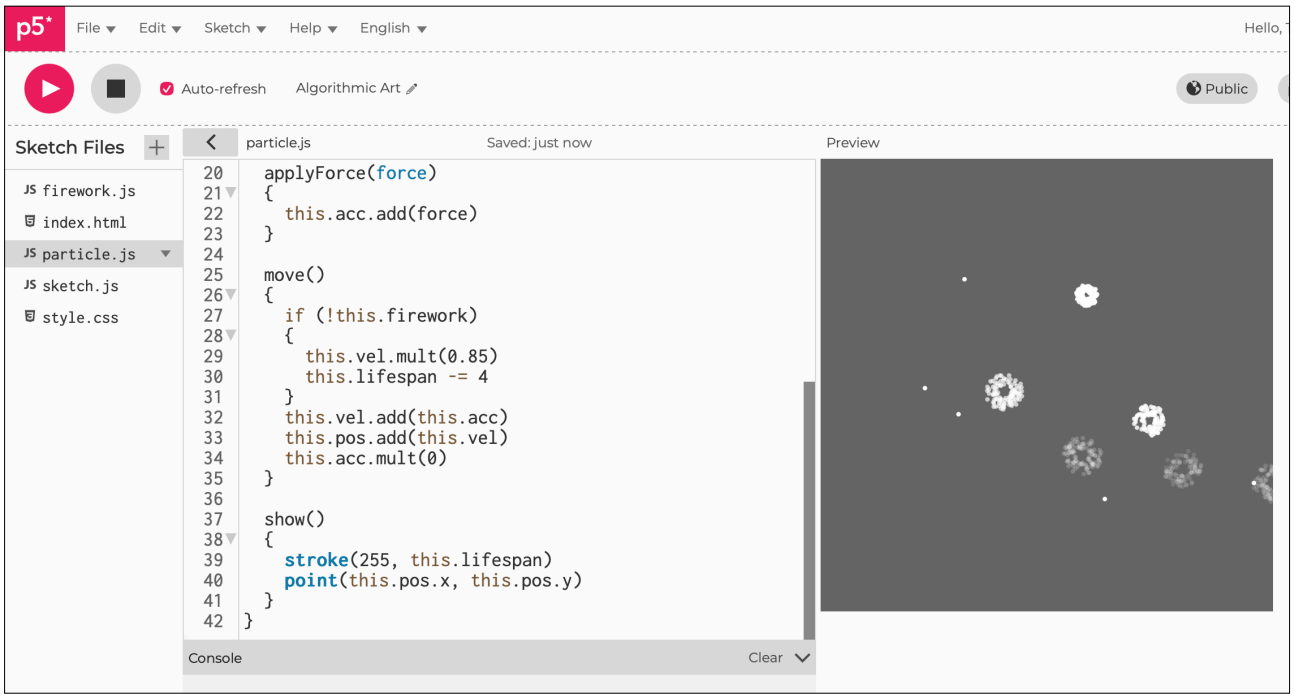
```
class Particle
{
  constructor(x, y, firework)
  {
    this.pos = createVector(x, y)
    this.firework = firework
    this.lifespan = 255
    if (this.firework)
    {
      this.vel = createVector(0, random(-8, -4))
    }
    else
    {
      this.vel = p5.Vector.random2D()
      this.vel.mult(random(1, 3))
    }
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    if (!this.firework)
    {
      this.vel.mult(0.85)
      this.lifespan -= 4
    }
    this.vel.add(this.acc)
    this.pos.add(this.vel)
```

```
    this.acc.mult(0)
  }

  show()
  {
    stroke(255, this.lifespan)
    point(this.pos.x, this.pos.y)
  }
}
```

Notes

The `lifespan` variable becomes the `alpha` value and causes them to fade.

Figure E5.29



Sketch E5.30 improve fade adjustment

We want to have only the sparks fireworks fade, so a little bit of adjustment to a few parameters for velocity as well to give a more pleasing effect. You can play with these yourself.

particle.js

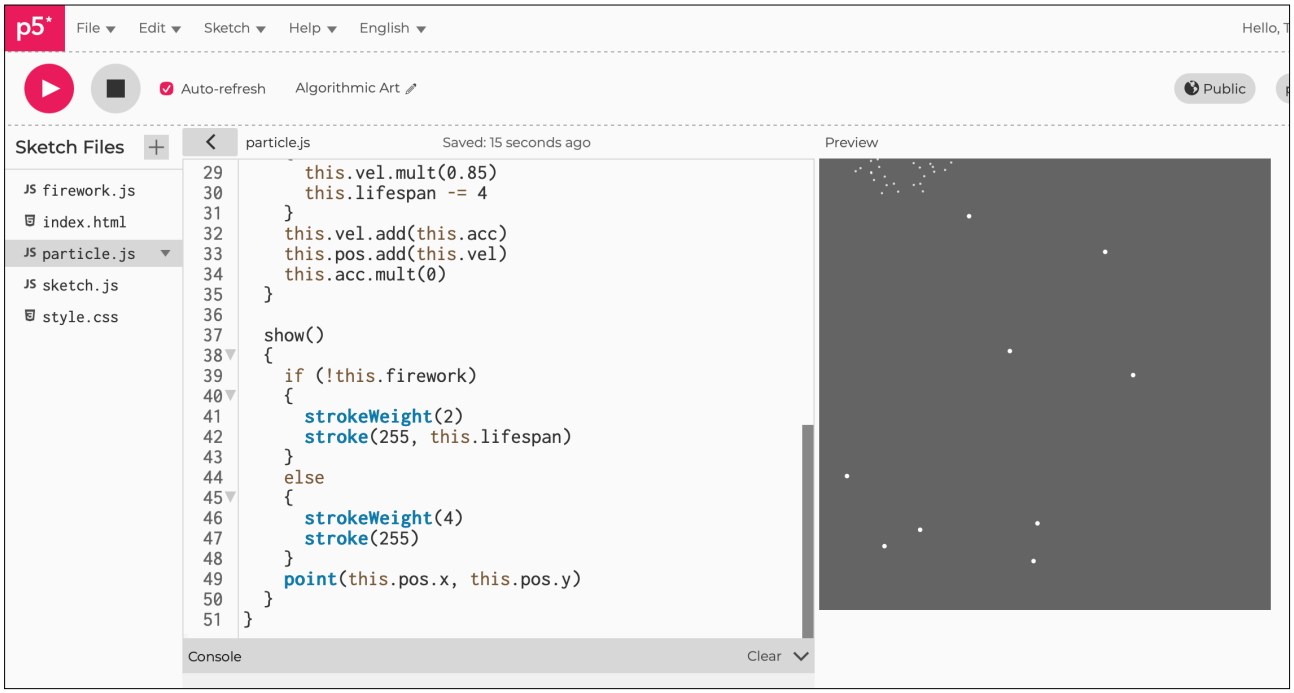
```
class Particle
{
  constructor(x, y, firework)
  {
    this.pos = createVector(x, y)
    this.firework = firework
    this.lifespan = 255
    if (this.firework)
    {
      this.vel = createVector(0, random(-12, -8))
    }
    else
    {
      this.vel = p5.Vector.random2D()
      this.vel.mult(random(2, 10))
    }
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    if (!this.firework)
    {
      this.vel.mult(0.85)
      this.lifespan -= 4
    }
    this.vel.add(this.acc)
```

```
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  show()
  {
    if (!this.firework)
    {
      strokeWeight(2)
      stroke(255, this.lifespan)
    }
    else
    {
      strokeWeight(4)
      stroke(255)
    }
    point(this.pos.x, this.pos.y)
  }
}
```

Notes

Definitely more like a firework display, but more to do.

Figure E5.30



Sketch E5.31 increase gravity reduce particles

! The sketch.js

We need to increase gravity a little more in sketch.js, and also reduce the number of particles being generated.

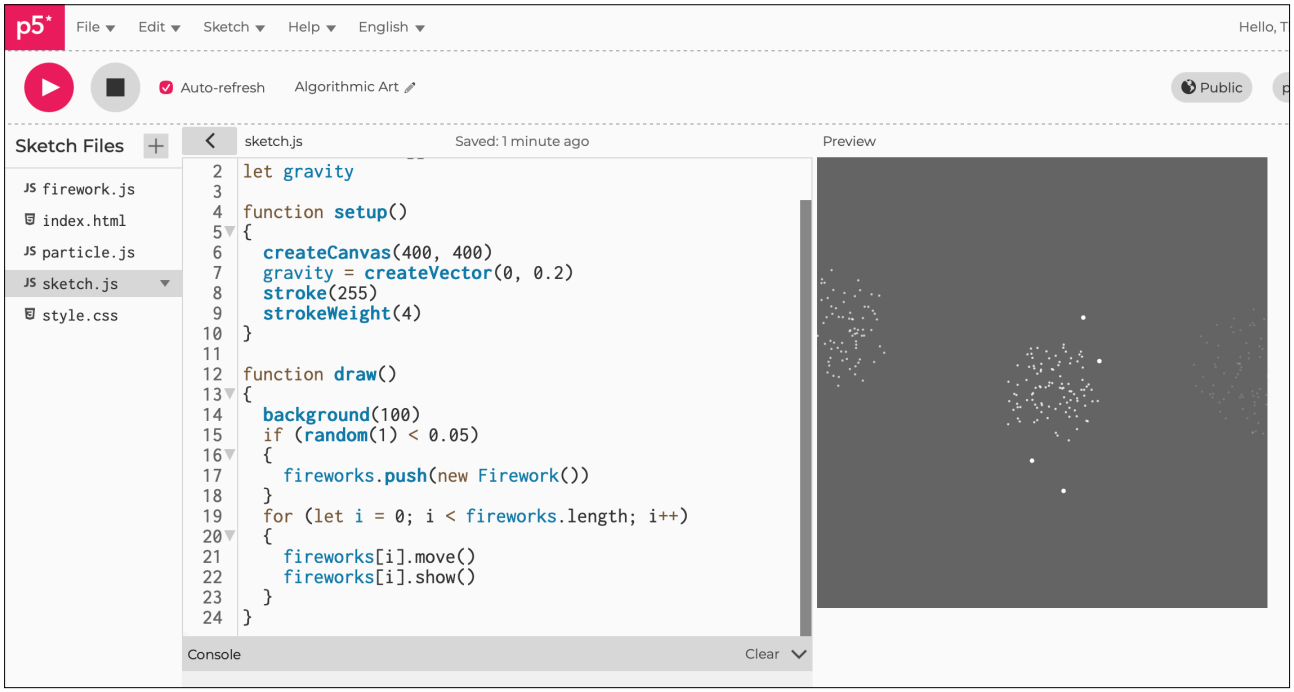
sketch.js

```
let fireworks = []
let gravity

function setup()
{
  createCanvas(400, 400)
  gravity = createVector(0, 0.2)
  stroke(255)
  strokeWeight(4)
}

function draw()
{
  background(100)
  if (random(1) < 0.05)
  {
    fireworks.push(new Firework())
  }
  for (let i = 0; i < fireworks.length; i++)
  {
    fireworks[i].move()
    fireworks[i].show()
  }
}
```

Notes

Still improving the effect.

Figure E5.31



Sketch E5.32 removing spent particles

! The firework.js file

We create lots of particles, but we never remove them from the array. This is why it starts to run very slowly after a short while. We have to delete those particles which have faded.

firework.js

```
class Firework
{
  constructor()
  {
    this.firework = new Particle(random(width), height, true)
    this.exploded = false
    this.particles = []
  }

  move()
  {
    if (!this.exploded)
    {
      this.firework.applyForce(gravity)
      this.firework.move()
      if (this.firework.vel.y >= 0)
      {
        this.exploded = true
        this.explode()
      }
    }
    for (let i = this.particles.length - 1; i >= 0; i--)
    {
      this.particles[i].applyForce(gravity)
      this.particles[i].move()
    }
  }

  show()
  {
    if (!this.exploded)
```

```
    {
      this.firework.show()
    }
    for (let i = 0; i < this.particles.length; i++)
    {
      this.particles[i].show()
    }
  }

  explode()
  {
    for (let i = 0; i < 100; i++)
    {
      let p = new Particle(this.firework.pos.x, this.firework.pos.y)
      this.particles.push(p)
    }
  }
}
```

Notes

This should make no difference to the running of the sketch. If it does, then just check though that line of code.

Sketch E5.33 splice the faded

We need to splice, remove in this case the particle that is done (faded), done will be another function which we create in particle.js very shortly.

firework.js

```
class Firework
{
  constructor()
  {
    this.firework = new Particle(random(width), height, true)
    this.exploded = false
    this.particles = []
  }

  move()
  {
    if (!this.exploded)
    {
      this.firework.applyForce(gravity)
      this.firework.move()
      if (this.firework.vel.y >= 0)
      {
        this.exploded = true
        this.explode()
      }
    }
    for (let i = this.particles.length - 1; i >= 0; i--)
    {
      this.particles[i].applyForce(gravity)
      this.particles[i].move()
      if (this.particles[i].done())
      {
        this.particles.splice(i, 1)
      }
    }
  }

  show()
```

```
  {
    if (!this.exploded)
    {
      this.firework.show()
    }
    for (let i = 0; i < this.particles.length; i++)
    {
      this.particles[i].show()
    }
  }

  explode()
  {
    for (let i = 0; i < 100; i++)
    {
      let p = new Particle(this.firework.pos.x, this.firework.pos.y)
      this.particles.push(p)
    }
  }
}
```

Notes

You will get an error message if you run the code.

Sketch E5.34 it is done()

! The particle.js file

Adding the `done()` function in particle.js, we use the boolean return as `true`, which means that it will splice (remove) that particle when it is done (`lifespan` is zero or less).

particle.js

```
class Particle
{
  constructor(x, y, firework)
  {
    this.pos = createVector(x, y)
    this.firework = firework
    this.lifespan = 255
    if (this.firework)
    {
      this.vel = createVector(0, random(-12, -8))
    }
    else
    {
      this.vel = p5.Vector.random2D()
      this.vel.mult(random(2, 10))
    }
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    if (!this.firework)
    {
      this.vel.mult(0.85)
      this.lifespan -= 4
    }
```

```
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  done()
  {
    if (this.lifespan < 0)
    {
      return true
    }
    else
    {
      return false
    }
  }

  show()
  {
    if (!this.firework)
    {
      strokeWeight(2)
      stroke(255, this.lifespan)
    }
    else
    {
      strokeWeight(4)
      stroke(255)
    }
    point(this.pos.x, this.pos.y)
  }
}
```

Notes

It should work fine now.

Sketch E5.35 sparks and rockets

! The firework.js file

We need to remove the rocket part of the firework now (we did the sparks). We want to delete the rocket firework when it has exploded and there are no more particles. First, we create a done() function for the firework.

firework.js

```
class Firework
{
  constructor()
  {
    this.firework = new Particle(random(width), height, true)
    this.exploded = false
    this.particles = []
  }

  done()
  {
    if (this.exploded && this.particles.length === 0)
    {
      return true
    }
    else
    {
      return false
    }
  }

  move()
  {
    if (!this.exploded)
    {
      this.firework.applyForce(gravity)
      this.firework.move()
      if (this.firework.vel.y >= 0)
      {
        this.exploded = true
        this.explode()
```

```
      }
    }
    for (let i = this.particles.length - 1; i >= 0; i--)
    {
      this.particles[i].applyForce(gravity)
      this.particles[i].move()
      if (this.particles[i].done())
      {
        this.particles.splice(i, 1)
      }
    }
  }

  show()
  {
    if (!this.exploded)
    {
      this.firework.show()
    }
    for (let i = 0; i < this.particles.length; i++)
    {
      this.particles[i].show()
    }
  }

  explode()
  {
    for (let i = 0; i < 100; i++)
    {
      let p = new Particle(this.firework.pos.x, this.firework.pos.y)
      this.particles.push(p)
    }
  }
}
```

Notes

Looking good.

Sketch E5.36 backwards array

! The sketch.js file

Now we need to go to the sketch.js and firstly go through the array backwards and delete it (splice it). This may seem odd, but there is logic to this, just not immediately intuitive.

sketch.js

```
let fireworks = []
let gravity

function setup()
{
  createCanvas(400, 400)
  gravity = createVector(0, 0.2)
  stroke(255)
  strokeWeight(4)
}

function draw()
{
  background(100)
  if (random(1) < 0.05)
  {
    fireworks.push(new Firework())
  }
  for (let i = fireworks.length - 1; i >= 0; i--)
  {
    fireworks[i].move()
    fireworks[i].show()
    if (fireworks[i].done())
    {
      fireworks.splice(i, 1)
    }
  }
}
```

Notes

You should see the same as before, but it should not slow down. You can check by adding this line of code at the end of `draw()`:

```
console.log(fireworks.length)
```

...and remove it once you are satisfied!

Sketch E5.37 trails

Let's add some trails to the fireworks.

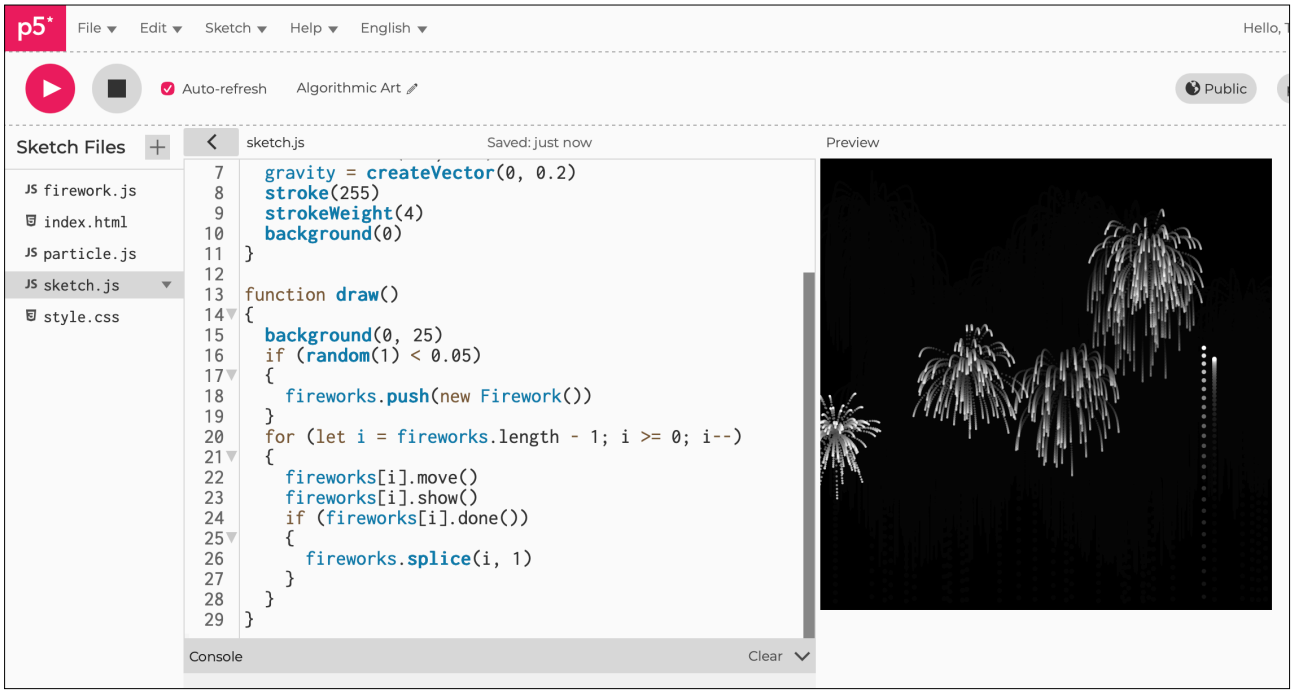
sketch.js

```
let fireworks = []
let gravity

function setup()
{
  createCanvas(400, 400)
  gravity = createVector(0, 0.2)
  stroke(255)
  strokeWeight(4)
  background(0)
}

function draw()
{
  background(0, 25)
  if (random(1) < 0.05)
  {
    fireworks.push(new Firework())
  }
  for (let i = fireworks.length - 1; i >= 0; i--)
  {
    fireworks[i].move()
    fireworks[i].show()
    if (fireworks[i].done())
    {
      fireworks.splice(i, 1)
    }
  }
}
```

Notes

We now have something resembling a fireworks display.

Figure E5.37



Sketch E5.38 colour

! The firework.js file

We are going to add colour with HSB rather than RGB (just a bit nicer). In the firework.js sketch, we introduce a variable called hu, which is the first argument for the colourMode(HSB). It is going to generate a random colour.

firework.js

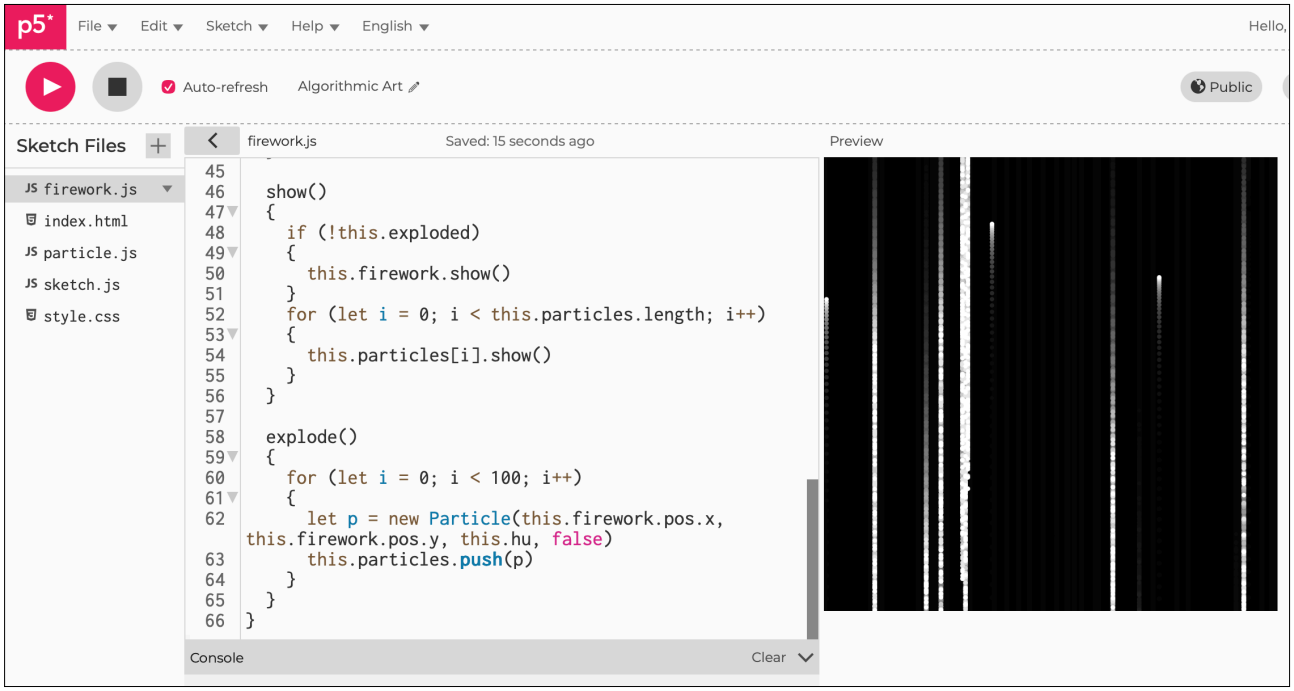
```
class Firework
{
  constructor()
  {
    this.hu = random(255)
    this.firework = new Particle(random(width), height, this.hu, true)
    this.exploded = false
    this.particles = []
  }

  done()
  {
    if (this.exploded && this.particles.length === 0)
    {
      return true
    }
    else
    {
      return false
    }
  }

  move()
  {
    if (!this.exploded)
    {
      this.firework.applyForce(gravity)
      this.firework.move()
      if (this.firework.vel.y >= 0)
      {
```

```
        this.exploded = true
        this.explode()
      }
    }
    for (let i = this.particles.length - 1; i >= 0; i--)
    {
      this.particles[i].applyForce(gravity)
      this.particles[i].move()
      if (this.particles[i].done())
      {
        this.particles.splice(i, 1)
      }
    }
  }

  show()
  {
    if (!this.exploded)
    {
      this.firework.show()
    }
    for (let i = 0; i < this.particles.length; i++)
    {
      this.particles[i].show()
    }
  }

  explode()
  {
    for (let i = 0; i < 100; i++)
    {
      let p = new Particle(this.firework.pos.x, this.firework.pos.y, this.hu,
false)
      this.particles.push(p)
    }
  }
}
```

Notes

This will break it. Don't panic, we will resolve it shortly.

Figure E5.38



Sketch E5.39 adding the colour in

! The particle.js file

In the particle.js sketch, we add it in this way.

particle.js

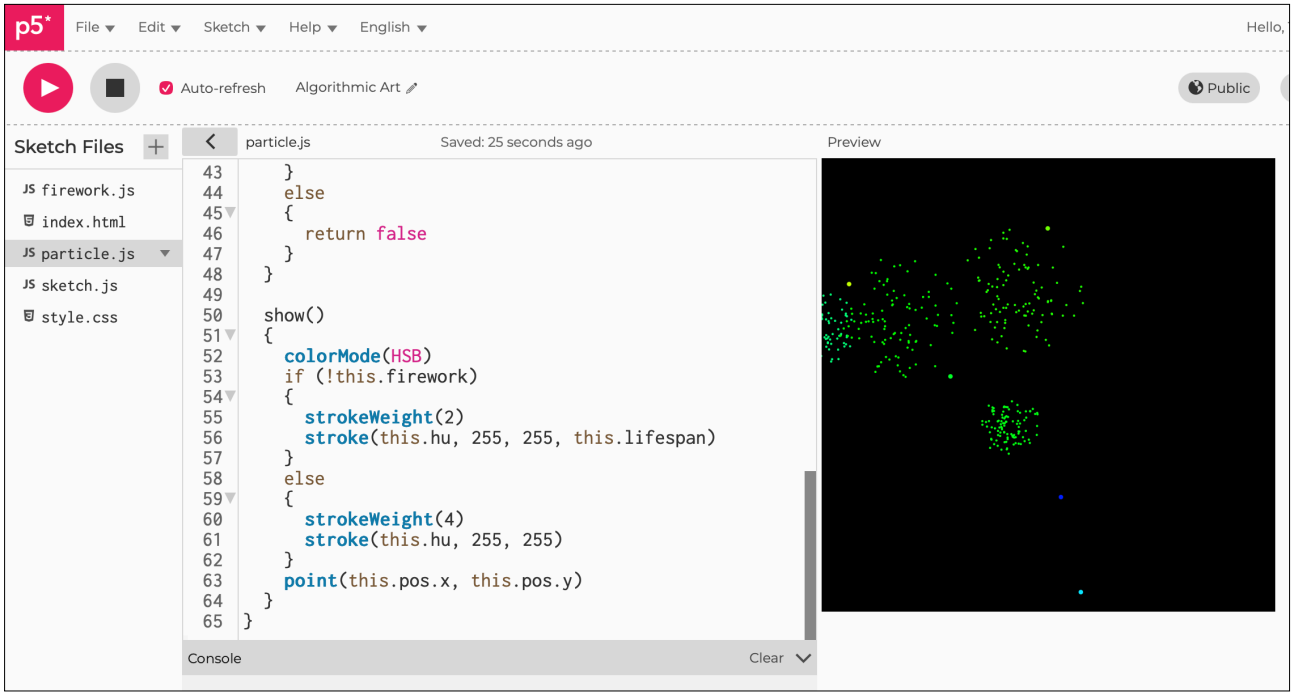
```
class Particle
{
  constructor(x, y, hu, firework)
  {
    this.pos = createVector(x, y)
    this.firework = firework
    this.lifespan = 255
    this.hu = hu
    if (this.firework)
    {
      this.vel = createVector(0, random(-12, -8))
    }
    else
    {
      this.vel = p5.Vector.random2D()
      this.vel.mult(random(2, 10))
    }
    this.acc = createVector(0, 0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  move()
  {
    if (!this.firework)
    {
      this.vel.mult(0.85)
      this.lifespan -= 4
    }
```

```
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  done()
  {
    if (this.lifespan < 0)
    {
      return true
    }
    else
    {
      return false
    }
  }

  show()
  {
    colorMode(HSB)
    if (!this.firework)
    {
      strokeWeight(2)
      stroke(this.hu, 255, 255, this.lifespan)
    }
    else
    {
      strokeWeight(4)
      stroke(this.hu, 255, 255)
    }
    point(this.pos.x, this.pos.y)
  }
}
```

Notes

And we are back again, this time in colour.

Figure E5.39



Sketch E5.40 colorMode(RGB)

! The sketch.js file

But we have lost the trails, so we need to change the `colorMode()` to `RGB` in sketch.js

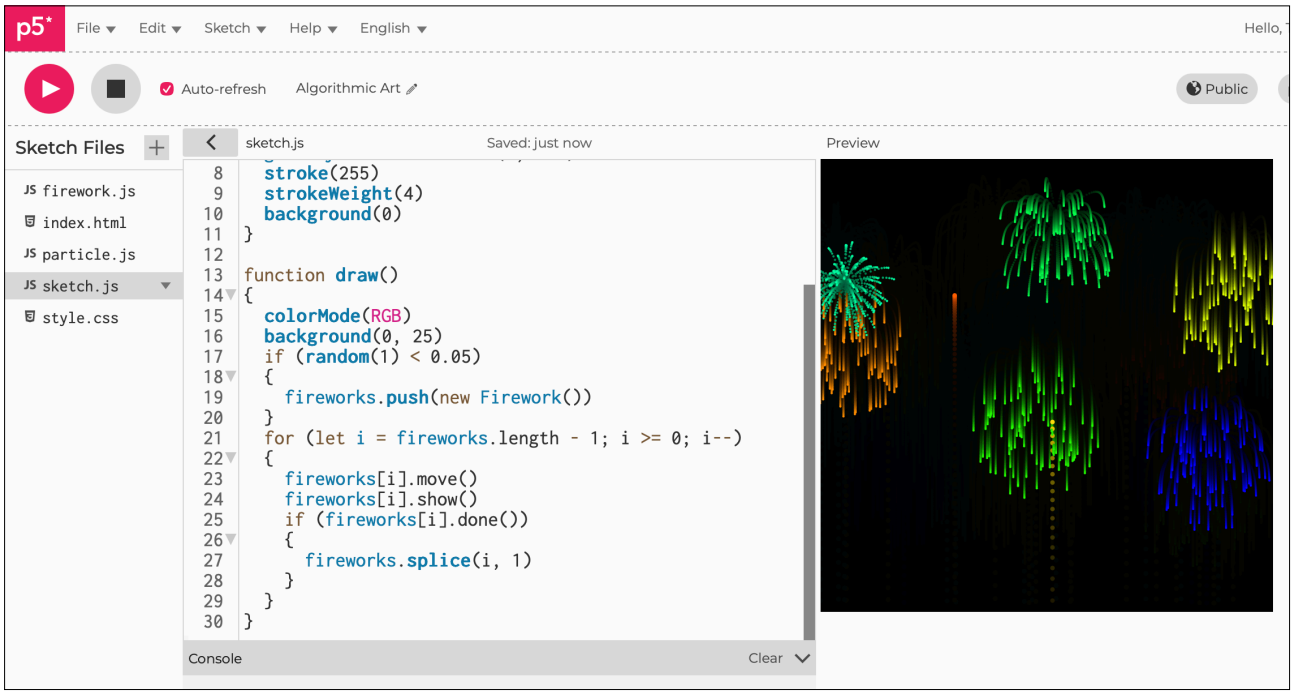
sketch.js

```
let fireworks = []
let gravity

function setup()
{
  createCanvas(400, 400)
  gravity = createVector(0, 0.2)
  stroke(255)
  strokeWeight(4)
  background(0)
}

function draw()
{
  colorMode(RGB)
  background(0, 25)
  if (random(1) < 0.05)
  {
    fireworks.push(new Firework())
  }
  for (let i = fireworks.length - 1; i >= 0; i--)
  {
    fireworks[i].move()
    fireworks[i].show()
    if (fireworks[i].done())
    {
      fireworks.splice(i, 1)
    }
  }
}
```

Notes

Now that looks even better. We are done. You have your fireworks display.

Figure E5.40



Algorithmic Art
Module E
Unit #6
Perlin Flowfield

Module E Unit #6: perlin flowfield

We are going to replace pixels with arrows with Perlin noise to show a flowfield. We start where we left off with 2D Perlin noise and instead of a grayscale value we will have a vector which will point in a direction according to Perlin noise.

Keep the fireworks display sketch from the previous unit, suggest duplicating it. We can use the `particle.js` file for this. We don't need the ~~`firework.js`~~ file. Delete all the code in those files.

Sketch E6.1 starting point

! Our initial sketch in sketch.js

We will start with familiar code from module A unit #2, Perlin 2D. Saves a bit of time, assuming you have completed that unit.

sketch.js

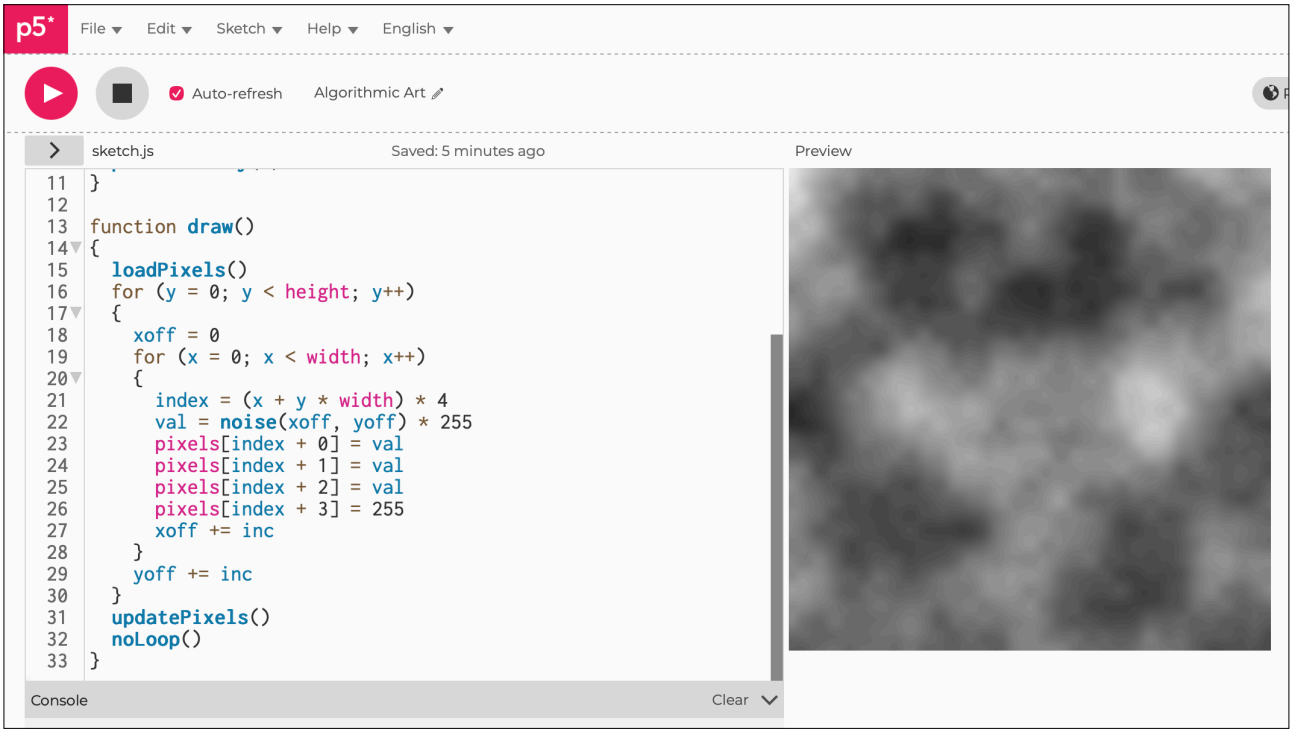
```
let xoff = 0
let yoff = 0
let inc = 0.01
let index
let val

function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

function draw()
{
  loadPixels()
  for (y = 0; y < height; y++)
  {
    xoff = 0
    for (x = 0; x < width; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      pixels[index + 0] = val
      pixels[index + 1] = val
      pixels[index + 2] = val
      pixels[index + 3] = 255
      xoff += inc
    }
    yoff += inc
  }
  updatePixels()
```

```
  noLoop()
}
```

Notes

This is what we had before.

Figure E6.1



Sketch E6.2 vectors not pixels

But we want vectors, not pixels. So we also don't want a vector for every pixel. So we will space them out with a variable, which we will call `scl` (short for scale). We will need columns (`cols`) and rows (`rows`). To get the right number of rows and columns, we divide the width and height by `scl` and also use `floor` so we have an integer number.

We can also get rid of all the references to `pixels` (`//` commented out and highlighted in blue).

sketch.js

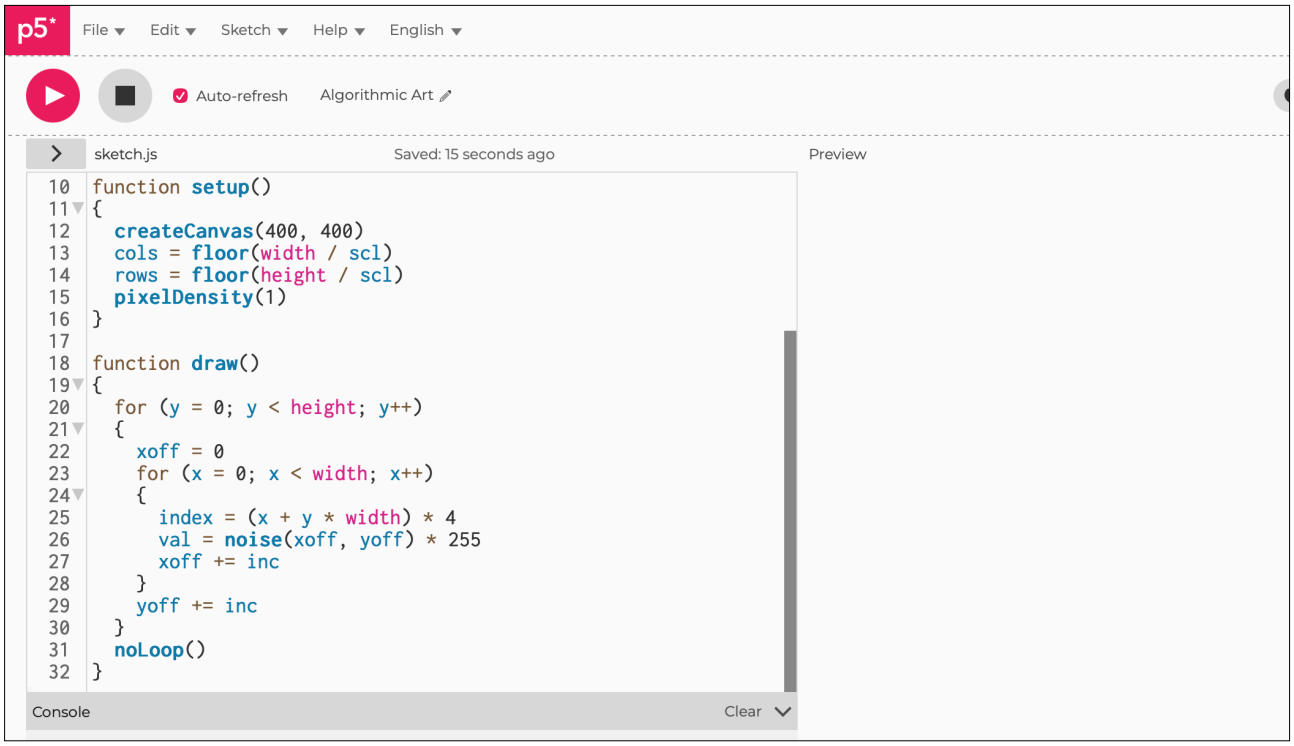
```
let xoff = 0
let yoff = 0
let inc = 0.01
let index
let val
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  // loadPixels()
  for (y = 0; y < height; y++)
  {
    xoff = 0
    for (x = 0; x < width; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      // pixels[index + 0] = val
```

```
      // pixels[index + 1] = val
      // pixels[index + 2] = val
      // pixels[index + 3] = 255
      xoff += inc
    }
    yoff += inc
  }
  // updatePixels()
  noLoop()
}
```

Notes

Nothing to show for it.

Figure E6.2



Sketch E6.3 random squares

To make sure everything is working ok we will use those variables we have just created to draw a square and fill it randomly.

sketch.js

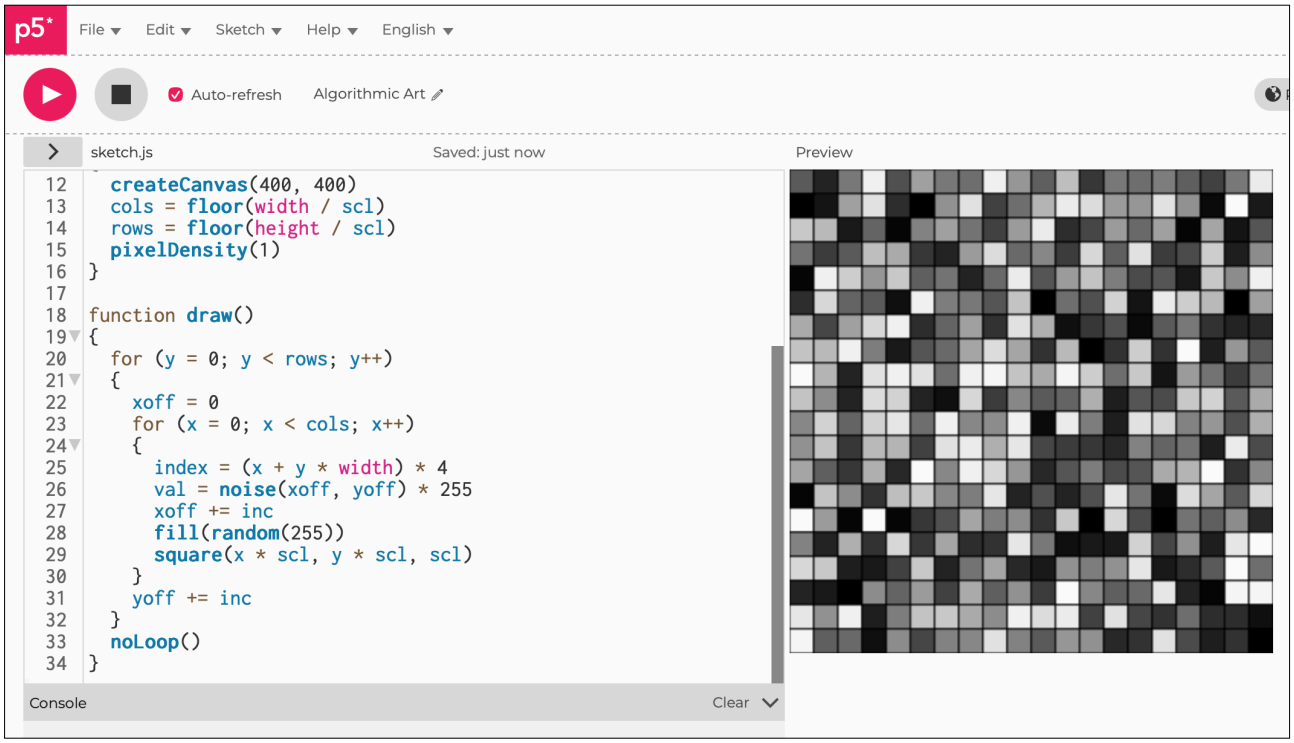
```
let xoff = 0
let yoff = 0
let inc = 0.01
let index
let val
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      xoff += inc
      fill(random(255))
      square(x * scl, y * scl, scl)
    }
    yoff += inc
  }
  noLoop()
```

```
}
```

Notes

Just randomly filled squares.

Figure E6.3



Sketch E6.4 perlin fill

Instead of randomly filling the squares with greyscale, let's change it to Perlin. We already have the Perlin value as val and make the inc (increment) a little bigger.

sketch.js

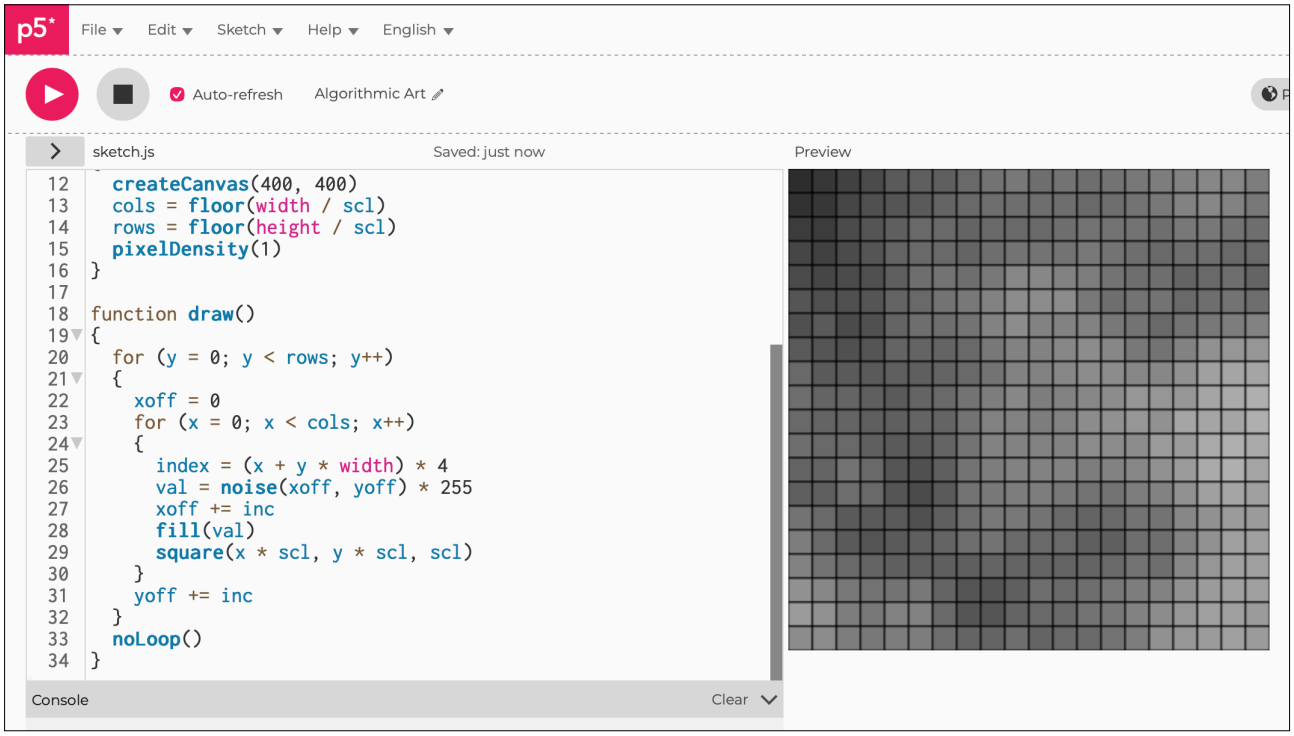
```
let xoff = 0
let yoff = 0
let inc = 0.1
let index
let val
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      xoff += inc
      fill(val)
      square(x * scl, y * scl, scl)
    }
    yoff += inc
  }
  noLoop()
```

```
}
```

Notes

The grey scaling is linked to the `Perlin noise()` value.

Figure E6.4



Sketch E6.5 vector lines

Instead of colouring in a square, we are going to draw a vector as a line, removing the `fill()` and `square()`.

sketch.js

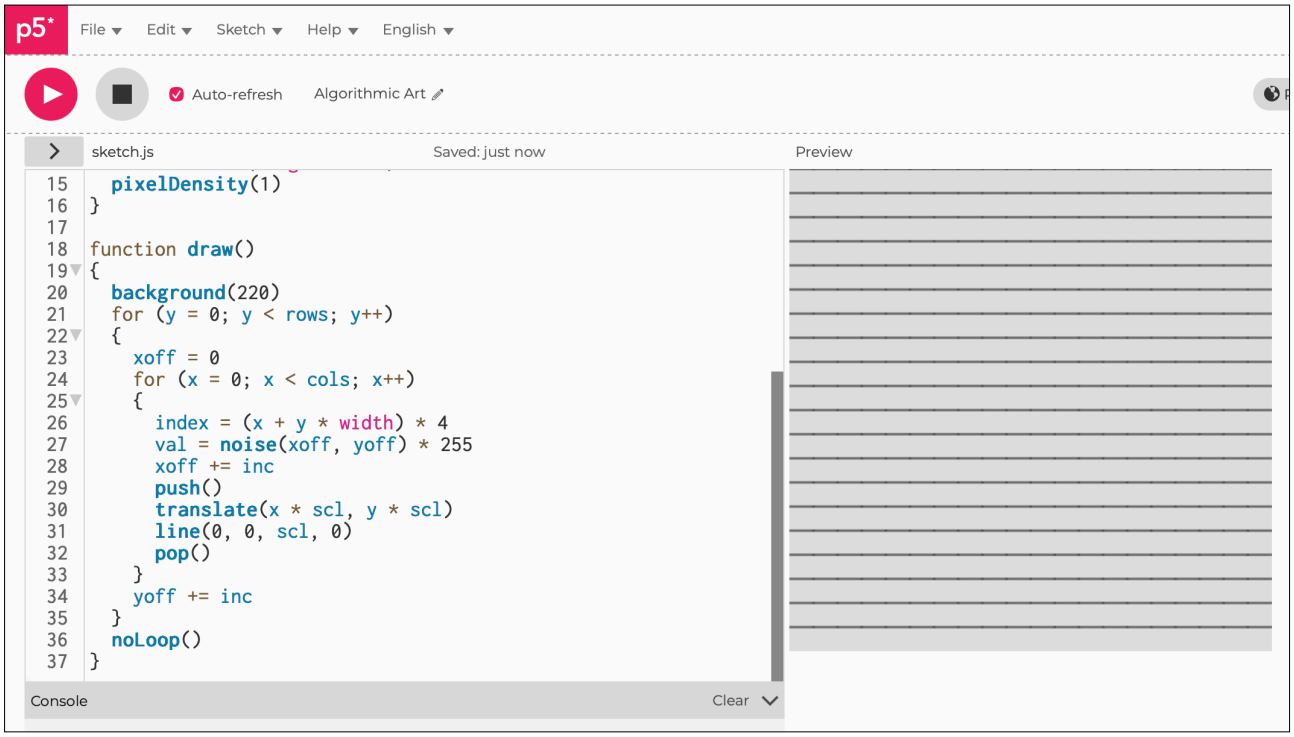
```
let xoff = 0
let yoff = 0
let inc = 0.1
let index
let val
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      xoff += inc
      push()
      translate(x * scl, y * scl)
      line(0, 0, scl, 0)
      pop()
      // fill(val)
```

```
      // square(x * scl, y * scl, scl)
    }
    yoff += inc
  }
  noLoop()
}
```

Notes

We aren't making use of the `noise()` yet.

Figure E6.5



Sketch E6.6 rotate

We are going to rotate a vector from an angle using a p5 function called: `p5.Vector.fromAngle()` and using the `heading()` function. We will rotate by `0.5 radians`, which is `30° degrees`.

sketch.js

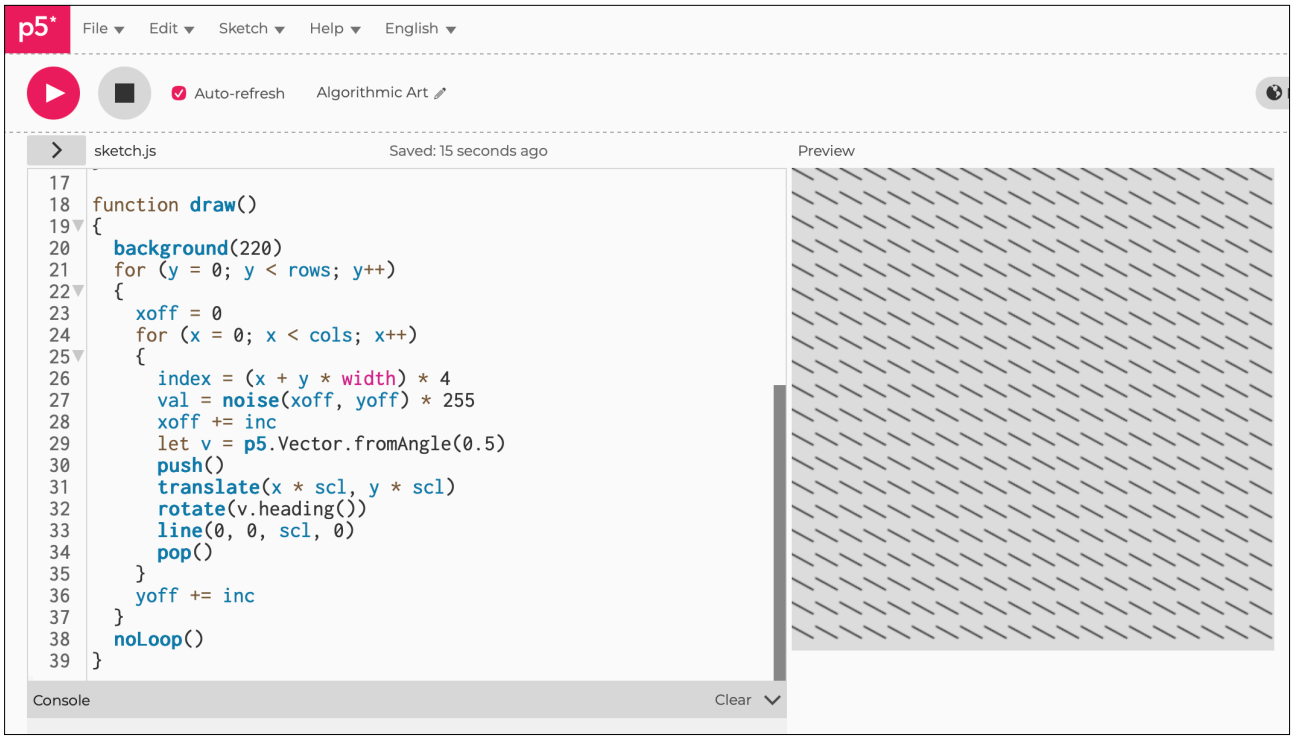
```
let xoff = 0
let yoff = 0
let inc = 0.1
let index
let val
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      xoff += inc
      let v = p5.Vector.fromAngle(0.5)
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
```

```
      line(0, 0, scl, 0)
      pop()
    }
    yoff += inc
  }
  noLoop()
}
```

Notes

All angled nicely.

Figure E6.6



Sketch E6.7 random angle

Instead of a fixed angle, let us use random() from 0 to 2π.

sketch.js

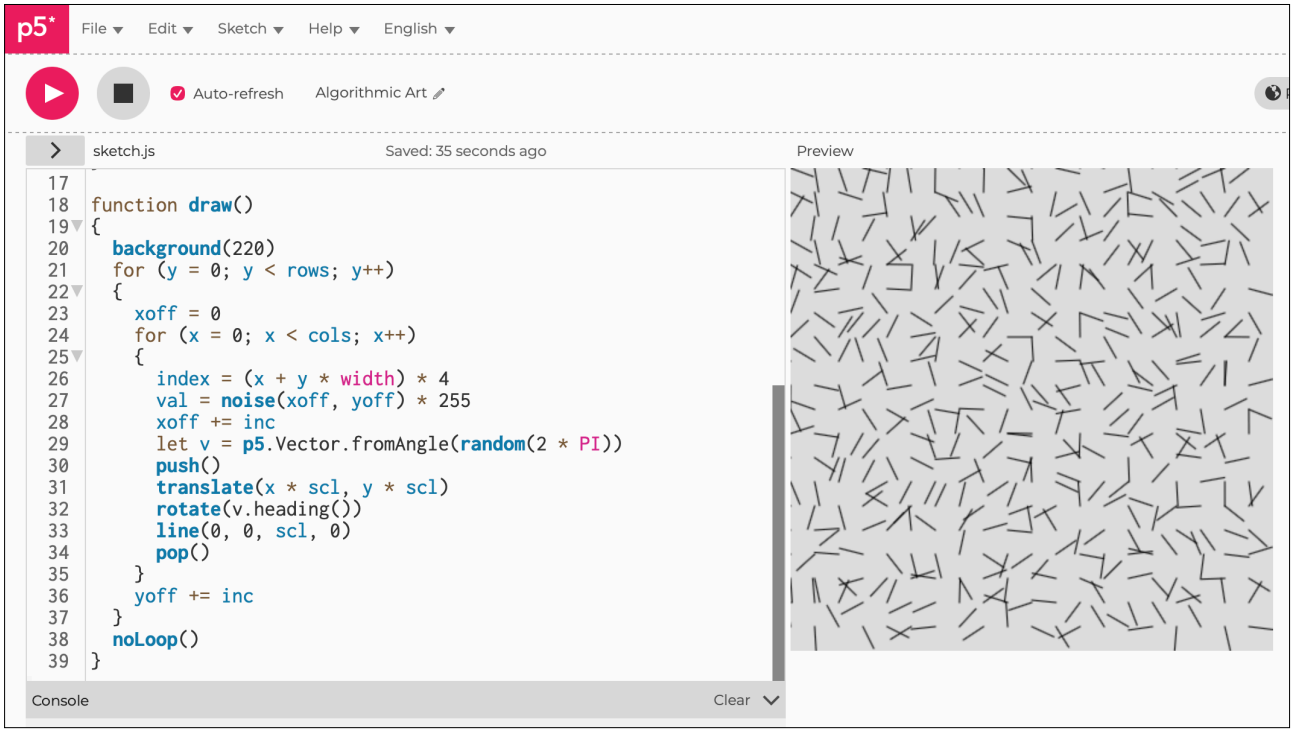
```
let xoff = 0
let yoff = 0
let inc = 0.1
let index
let val
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      xoff += inc
      let v = p5.Vector.fromAngle(random(2 * PI))
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      line(0, 0, scl, 0)
      pop()
```

```
    }
    yoff += inc
  }
  noLoop()
}
```

Notes

They are randomly arranged.

Figure E6.7



Sketch E6.8 perlin angle

Rather than pure random, we incorporate Perlin noise. We change the variable name from `val` to something more relevant (`angle`). We multiply by 2π (360°) and then draw the vector to that angle. Every time you refresh it, it changes, but you can see the Perlin noise effect.

sketch.js

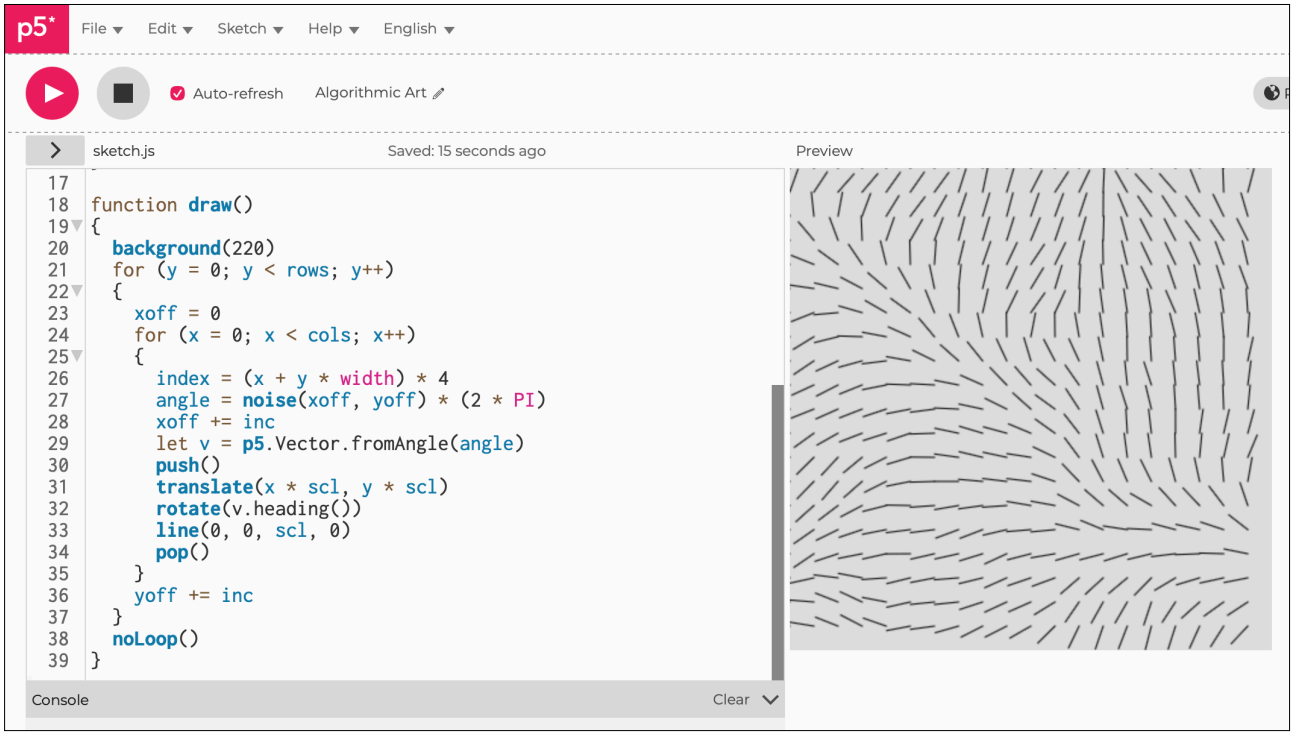
```
let xoff = 0
let yoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
      translate(x * scl, y * scl)
```

```
      rotate(v.heading())
      line(0, 0, scl, 0)
      pop()
    }
    yoff += inc
  }
  noLoop()
}
```

Notes

The angles now have something in common with each other.

Figure E6.8



Sketch E6.9 third dimension

Let's create a third dimension for time and call it zoff. We will change the time in small increments. This will animate as if we have slices of time. Each slice is connected to the previous one through Perlin noise. We need to make a few alterations to have a slow animated Perlin noise effect. You can play with the parameters.

sketch.js

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
```

```
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      line(0, 0, scl, 0)
      pop()
    }
    yoff += inc
    zoff += 0.0005
  }
  // noLoop()
}
```

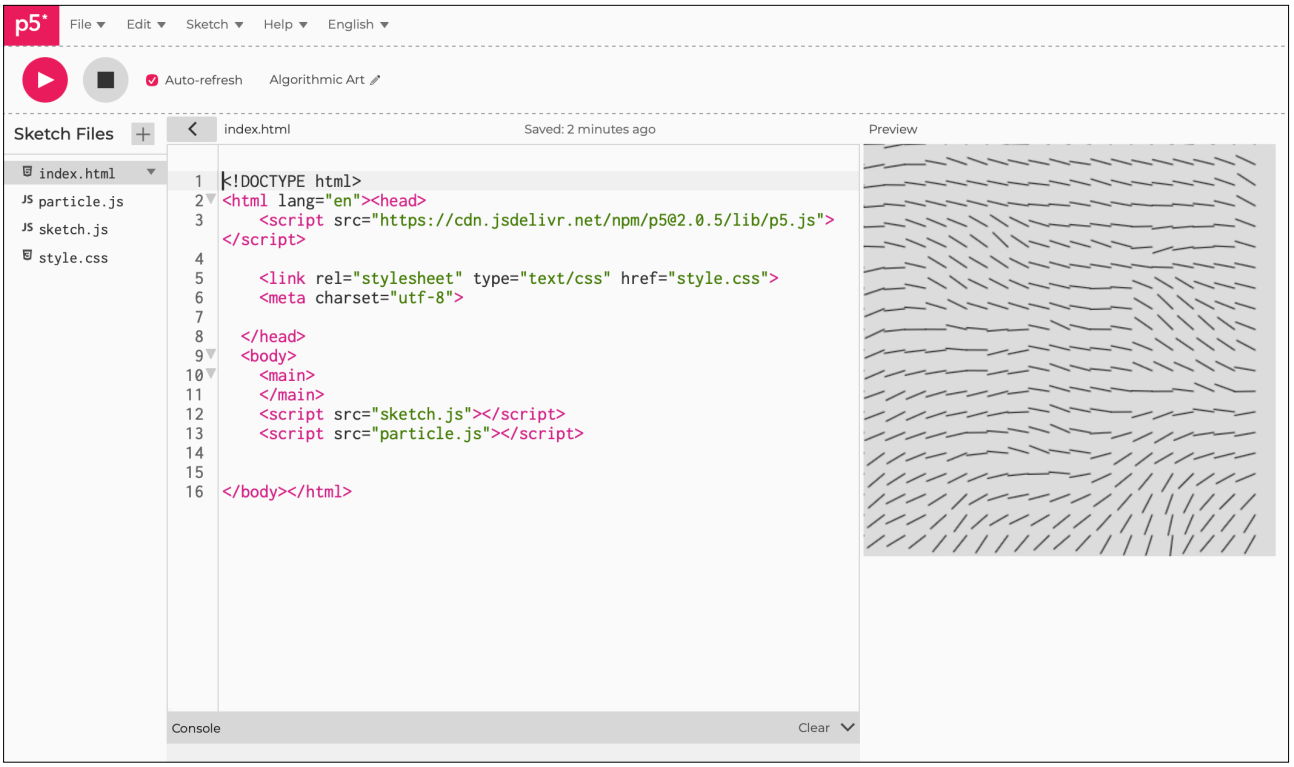
Notes

You now get a very pleasing, slowly moving, choreographed set of vectors.

A Particle File

If you don't already have the file `particle.js`, then please add it now as you did with the previous unit on fireworks. Remember to add it to the `index.html` file.

Adding the particle.js file



Sketch E6.10 particle class

! Our first particle.js sketch

We have added a constructor() function to the Particle class.

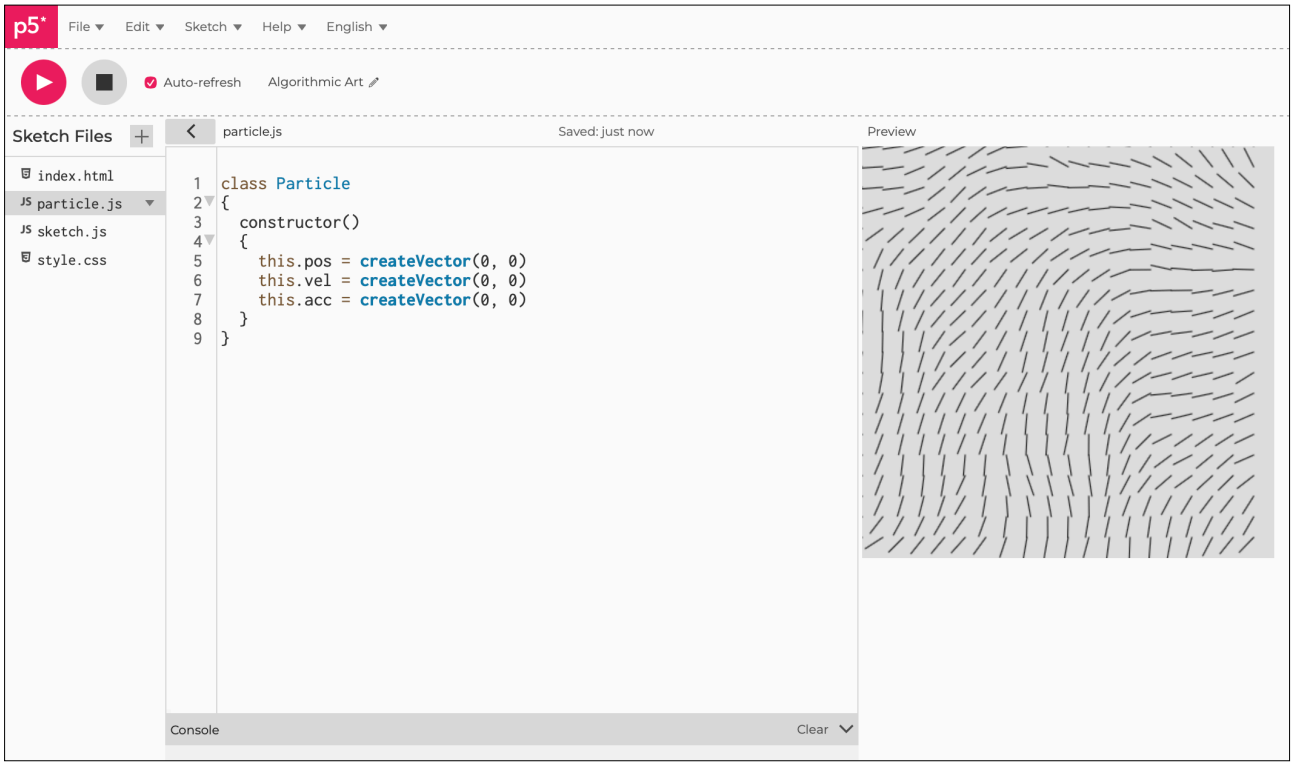
particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(0, 0)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }
}
```

Notes

We have created three vectors for the position, the velocity, and the acceleration.

Figure E6.10



Sketch E6.11 position, velocity and acceleration

We will add in the position, velocity, and acceleration to the move() function and reset the acceleration to zero.

particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(0, 0)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }
}
```

Notes

Nothing is changing just yet.

Sketch E6.12 the force

A force is applied to an acceleration

particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(0, 0)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }
}
```

Sketch E6.13 particle

Now, to draw the particle.

particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(0, 0)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
  }
}
```

Notes

Still nothing to see here. We need to go to the main sketch and reference it there.

Sketch E6.14 it is there honestly!

! The sketch.js file

Let us draw the particle in the main sketch. You may not be able to see it very well, but it is there, honest. We create a particle array and put one particle in it at index [0].

sketch.js

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
  particles[0] = new Particle()
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
```

```
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      line(0, 0, scl, 0)
      pop()
    }
    yoff += inc
    zoff += 0.0005
  }
  particles[0].show()
  particles[0].move()
}
```

Sketch E6.15 many particles

Let's make 100 particles with a for() loop.

sketch.js

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
```

```
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      line(0, 0, scl, 0)
      pop()
    }
    yoff += inc
    zoff += 0.0005
  }
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].show()
    particles[i].move()
  }
}
```

Sketch E6.16 random particles

! The particle.js file

We can check if this is working in particle.js and give them a random velocity.

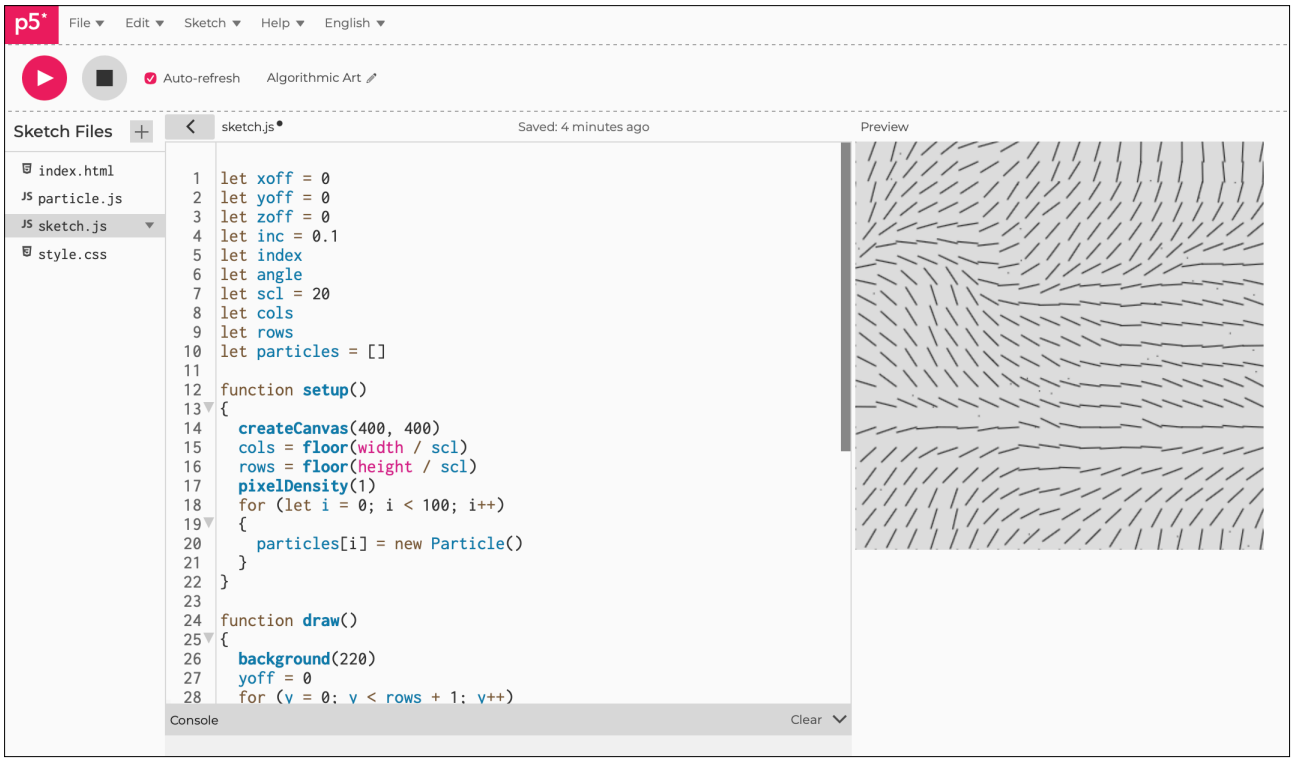
particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
  }
}
```

Notes

You should see them if you look closely.

Figure E6.16



Sketch E6.17 falling off the edge

To stop them disappearing off the edge of the canvas, we will create an `edge()` function boundary in particle.js. In the `setup()` part of the sketch.js, I have commented out the `pixelDensity()` and introduced a `strokeWeight(3)` for clarity (both of these are temporary and optional).

particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {

    point(this.pos.x, this.pos.y)
  }

  edges()
  {
    if (this.pos.x > width)
    {
```

```
      this.pos.x = 0
    }
    if (this.pos.x < 0)
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }
}
```

Sketch E6.18 hide the lines

! The sketch.js file

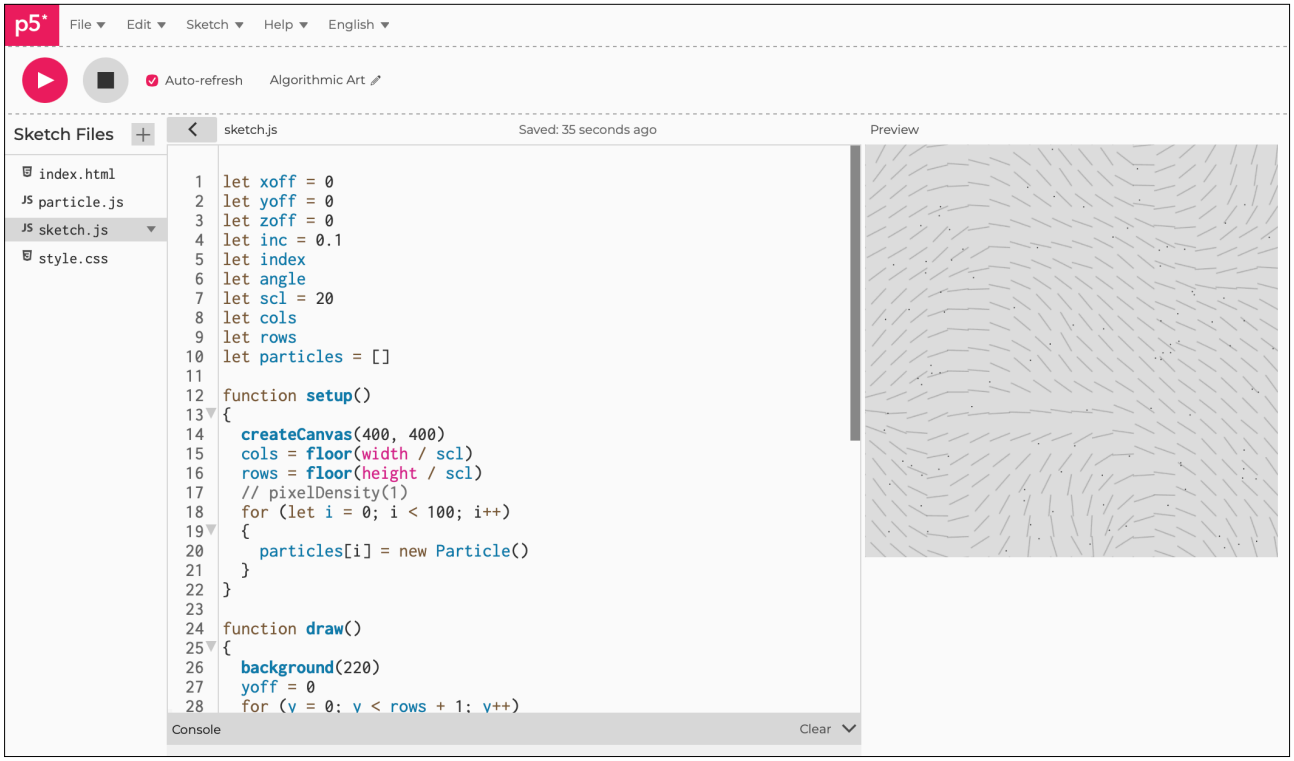
To hide the lines a bit, make the following adjustments (as well as adding the edges() function).

sketch.js

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  // pixelDensity(1)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
```

```
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      strokeWeight(1)
      stroke(0, 50)
      line(0, 0, scl, 0)
      pop()
    }
    yoff += inc
    zoff += 0.0005
  }
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].show()
    particles[i].move()
    particles[i].edges()
  }
}
```

Figure E6.18


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
Sketch Files
index.html
particle.js
sketch.js
style.css
sketch.js
Saved: 35 seconds ago
Preview
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
function setup()
{
createCanvas(400, 400)
cols = floor(width / scl)
rows = floor(height / scl)
// pixelDensity(1)
for (let i = 0; i < 100; i++)
{
particles[i] = new Particle()
}
}
function draw()
{
background(220)
yoff = 0
for (y = 0; y < rows + 1; y++)
Console
Clear

Sketch E6.19 better particles

! The particle.js file

In particle.js, I want to see the particles clearly.

particle.js

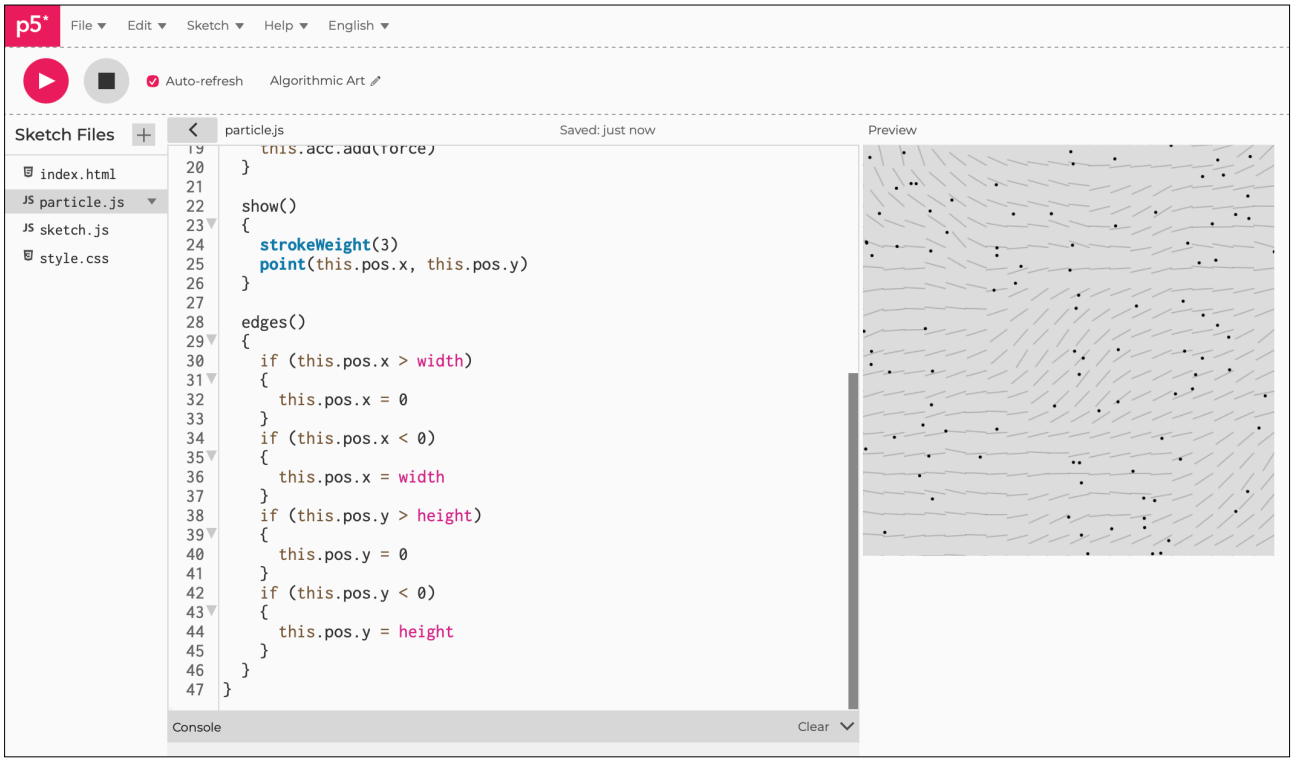
```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  edges()
  {
    if (this.pos.x > width)
    {
      this.pos.x = 0
```

```
    }
    if (this.pos.x < 0)
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }
}
```

Notes

The points aren't moving according to `perlin noise()`.

Figure E6.19



Sketch E6.20 array of vectors

! The sketch.js file

We want to store the values of the vectors in an array and then attribute them to the particles nearest to them. We create an empty array called flowfield[]. We use the formula for calculating the index and change the width to cols (and remove the times 4). Now we can fill the array with those vectors v.

sketch.js

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  // pixelDensity(1)
  flowfield = new Array(cols * rows)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
```

```
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      flowfield[index] = v
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      strokeWeight(1)
      stroke(0, 50)
      line(0, 0, scl, 0)
      pop()
    }
    yoff += inc
    zoff += 0.0005
  }
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].show()
    particles[i].move()
    particles[i].edges()
  }
}
```

Notes

Nothing new happening yet.

Sketch E6.21 following the vectors

! The particle.js file

We need to write a function called follow() in the particle class that helps us identify which particle is near which vector. It will have an argument called vectors. We have to find the co-ordinates of the particle according to how the vector lines are laid out.

particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  edges()
  {
    if (this.pos.x > width)
```

```
    {
      this.pos.x = 0
    }
    if (this.pos.x < 0)
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }

  follow(vectors)
  {
    let x = floor(this.pos.x/scl)
    let y = floor(this.pos.y/scl)
  }
}
```

Sketch E6.22 the index value

From that, we calculate the index value from x and y using the formula.

particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  edges()
  {
    if (this.pos.x > width)
    {
      this.pos.x = 0
    }
    if (this.pos.x < 0)
```

```
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }

  follow(vectors)
  {
    let x = floor(this.pos.x/scl)
    let y = floor(this.pos.y/scl)
    let index = x + (y * cols)
  }
}
```

Sketch E6.23 index force

We calculate the force of the vector at that index.

particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  edges()
  {
    if (this.pos.x > width)
    {
      this.pos.x = 0
    }
    if (this.pos.x < 0)
```

```
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }

  follow(vectors)
  {
    let x = floor(this.pos.x/scl)
    let y = floor(this.pos.y/scl)
    let index = x + (y * cols)
    let force = vectors[index]
  }
}
```

Sketch E6.24 may the force be with you

We then apply the force.

particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  edges()
  {
    if (this.pos.x > width)
    {
      this.pos.x = 0
    }
    if (this.pos.x < 0)
```

```
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }

  follow(vectors)
  {
    let x = floor(this.pos.x/scl)
    let y = floor(this.pos.y/scl)
    let index = x + (y * cols)
    let force = vectors[index]
    this.applyForce(force)
  }
}
```

Sketch E6.25 mad dash

Let's make the `velocity` zero in particle.js, then set a maximum speed `this.maxSpeed = 4`, and then limit the speed in the `move()` function.

particle.js

```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
  }

  move()
  {
    this.vel.add(this.acc)
    this.vel.limit(this.maxSpeed)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  edges()
  {
    if (this.pos.x > width)
    {
```

```
      this.pos.x = 0
    }
    if (this.pos.x < 0)
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }

  follow(vectors)
  {
    let x = floor(this.pos.x/scl)
    let y = floor(this.pos.y/scl)
    let index = x + (y * cols)
    let force = vectors[index]
    this.applyForce(force)
  }
}
```

Notes

We have come to a complete standstill. Be patient, just wait.

Sketch E6.26 the magnitude of it all

! The sketch.js file

In the sketch.js, we can set the magnitude of the force. You will notice that it goes a bit mad if you run it, but it is the beginning of something.

sketch.js

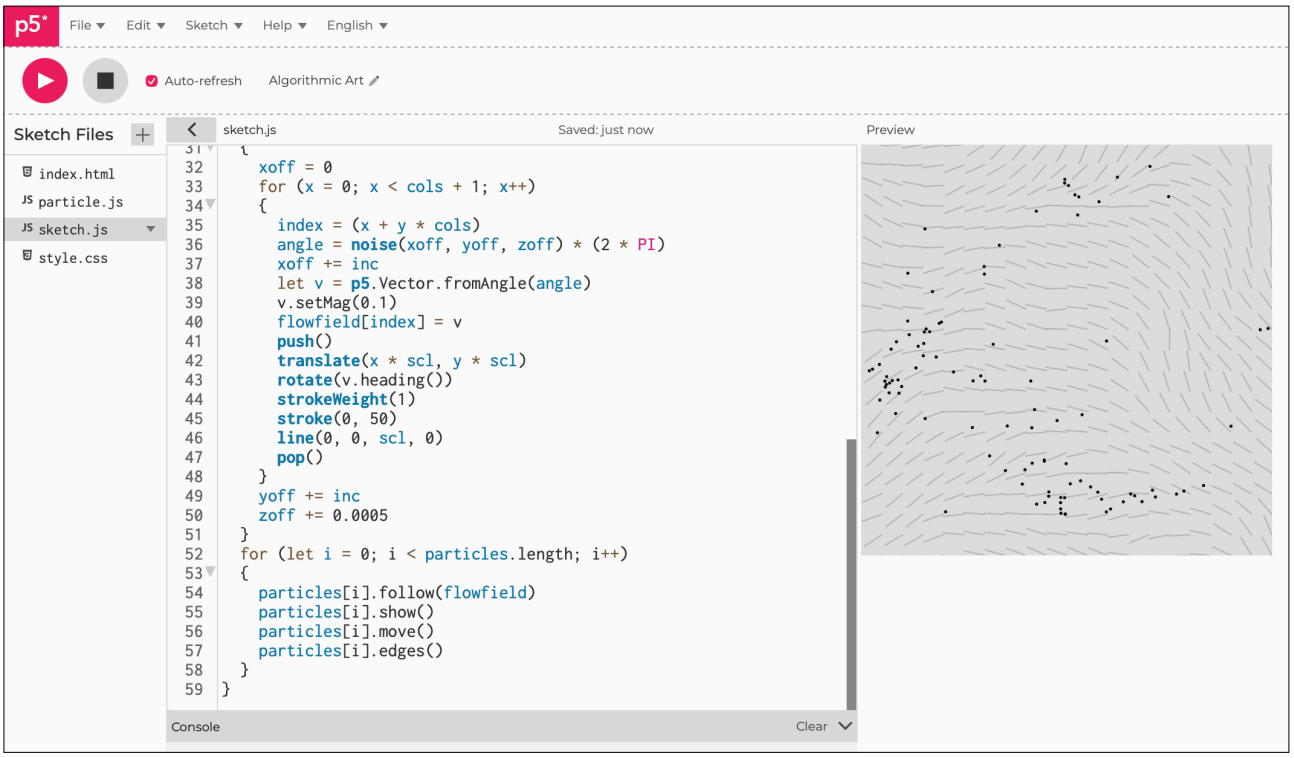
```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  // pixelDensity(1)
  // flowfield = new Array(cols * rows)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
```

```
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      v.setMag(0.1)
      flowfield[index] = v
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      strokeWeight(1)
      stroke(0, 50)
      line(0, 0, scl, 0)
      pop()
    }
    yoff += inc
    zoff += 0.0005
  }
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].follow(flowfield)
    particles[i].show()
    particles[i].move()
    particles[i].edges()
  }
}
```

Notes

You should see them start to follow the flowfield.

Figure E6.26



Sketch E6.27 zoff off

Now set the magnitude to 5 and turn off the zoff and you should see it very clearly.

sketch.js

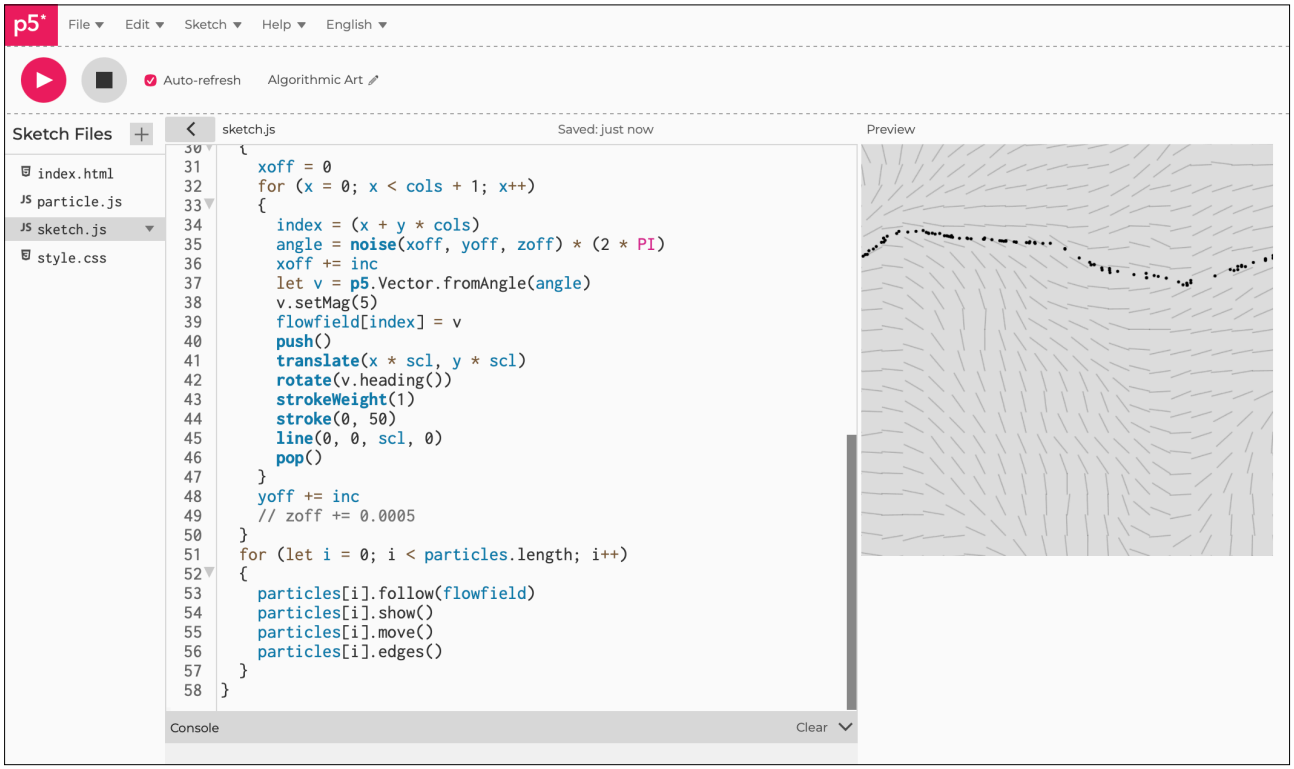
```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  // pixelDensity(1)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
```

```
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      v.setMag(5)
      flowfield[index] = v
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      strokeWeight(1)
      stroke(0, 50)
      line(0, 0, scl, 0)
      pop()
    }
    yoff += inc
    // zoff += 0.0005
  }
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].follow(flowfield)
    particles[i].show()
    particles[i].move()
    particles[i].edges()
  }
}
```

Notes

You can see it following an element of the flowfield.

Figure E6.27



Sketch E6.28 no lines

Removing the line vectors and adding more particles gives us a better feel, as well as reducing the force to 1 and also making the angle of rotation larger.

sketch.js

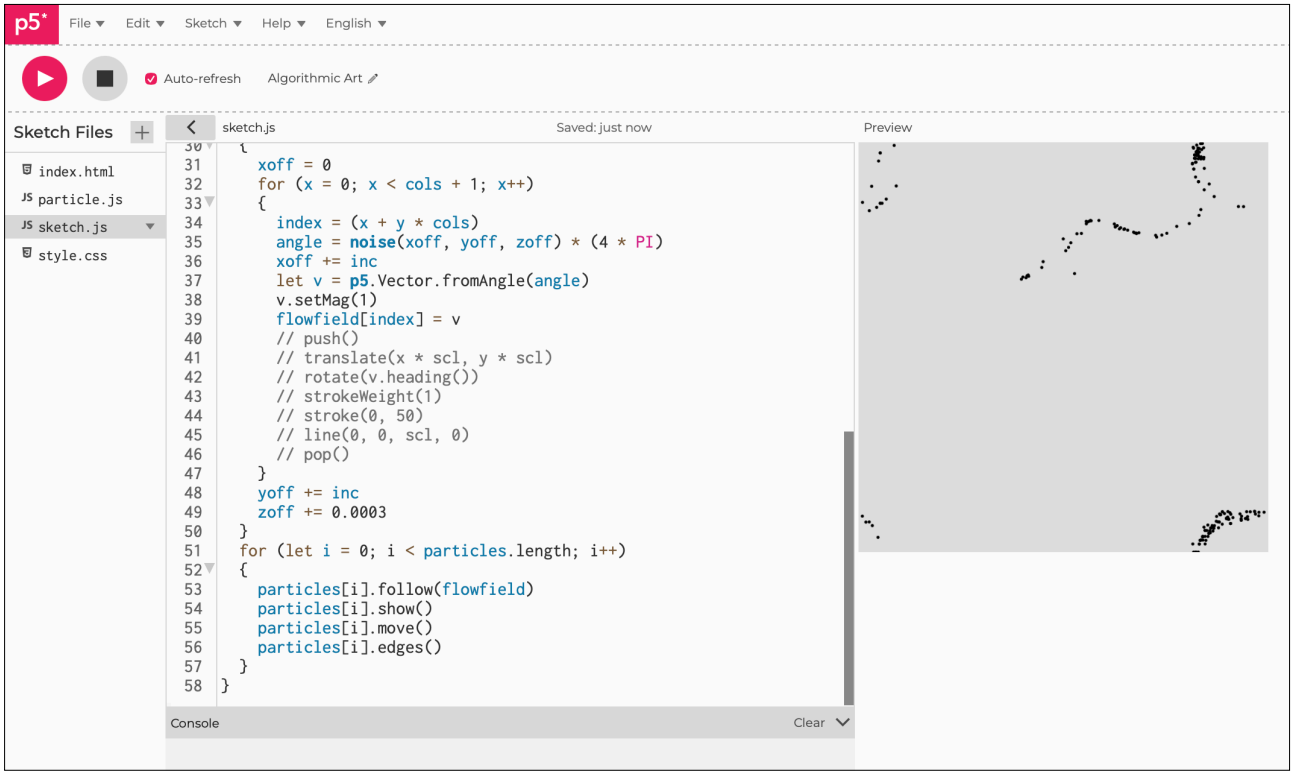
```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  // pixelDensity(1)
  for (let i = 0; i < 200; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
```

```
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (4 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      v.setMag(1)
      flowfield[index] = v
      // push()
      // translate(x * scl, y * scl)
      // rotate(v.heading())
      // strokeWeight(1)
      // stroke(0, 50)
      // line(0, 0, scl, 0)
      // pop()
    }
    yoff += inc
    zoff += 0.0003
  }
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].follow(flowfield)
    particles[i].show()
    particles[i].move()
    particles[i].edges()
  }
}
```

Notes

You do see some strange movement.

Figure E6.28



Sketch E6.29 nice visual

Making more amendments to get a nice visual. Move the `background()` into the `setup()` function so that it draws it only once, and increase the number of particles to `500`.

sketch.js

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  // pixelDensity(1)
  for (let i = 0; i < 500; i++)
  {
    particles[i] = new Particle()
  }
  background(220)
}

function draw()
{
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
```

```
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (4 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      v.setMag(1)
      flowfield[index] = v
    }
    yoff += inc
    zoff += 0.0003
  }
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].follow(flowfield)
    particles[i].show()
    particles[i].move()
    particles[i].edges()
  }
}
```

Notes

Interesting, but we can do even better.

Figure E6.29

Sketch E6.30 alpha stroke

! The particle.js file

In particle.js, we change the drawing of the point. Making it a strokeWeight() of 1 and a stroke() with some alpha.

particle.js

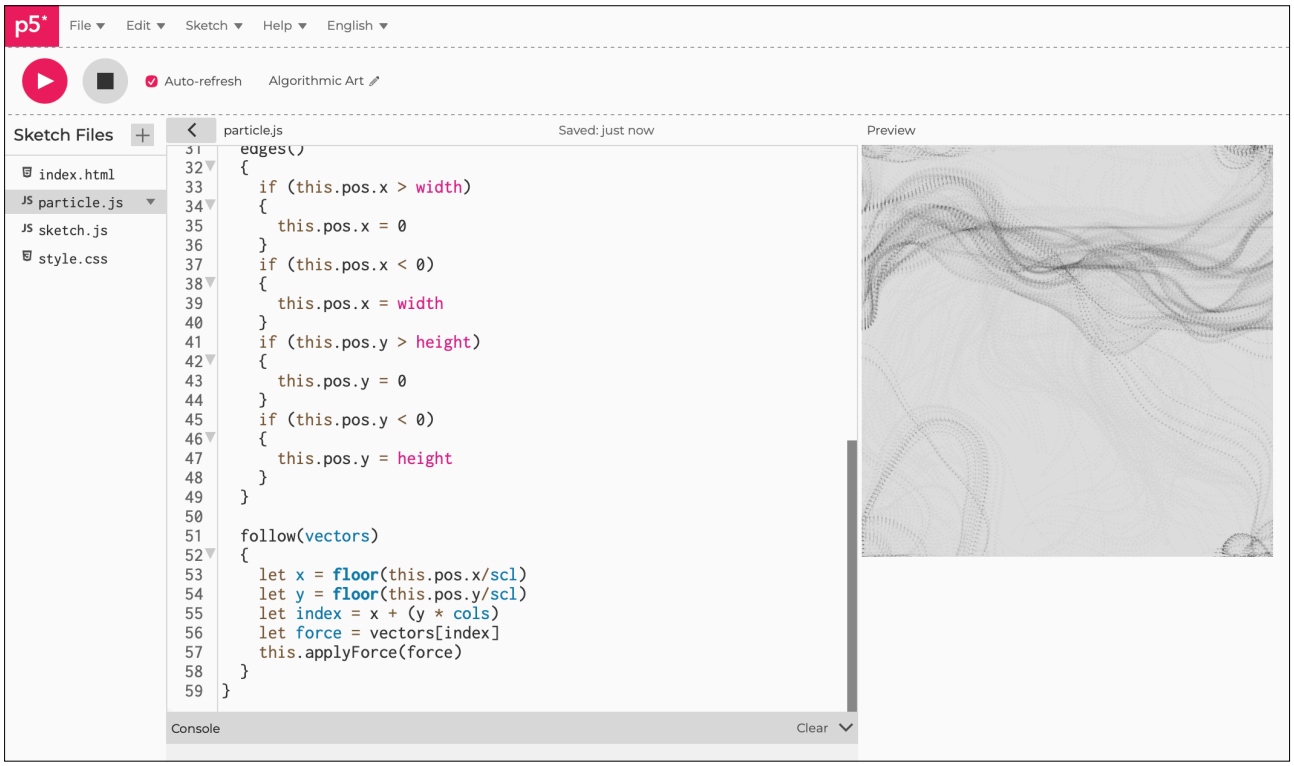
```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
  }

  move()
  {
    this.vel.add(this.acc)
    this.vel.limit(this.maxSpeed)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    strokeWeight(1)
    stroke(0, 5)
    point(this.pos.x, this.pos.y)
  }

  edges()
```

```
  {
    if (this.pos.x > width)
    {
      this.pos.x = 0
    }
    if (this.pos.x < 0)
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }

  follow(vectors)
  {
    let x = floor(this.pos.x/scl)
    let y = floor(this.pos.y/scl)
    let index = x + (y * cols)
    let force = vectors[index]
    this.applyForce(force)
  }
}
```

Notes

You will notice that it is a bit pixelated because the point is moving faster than the frame rate. We will address this in the next part.

Figure E6.30



Sketch E6.31 the previous position

We copy the position of the pixel before it moves and call `this.prevPos` using the `copy()` function. Then, when the pixel moves, it draws a line between its position and its previous position.

particle.js

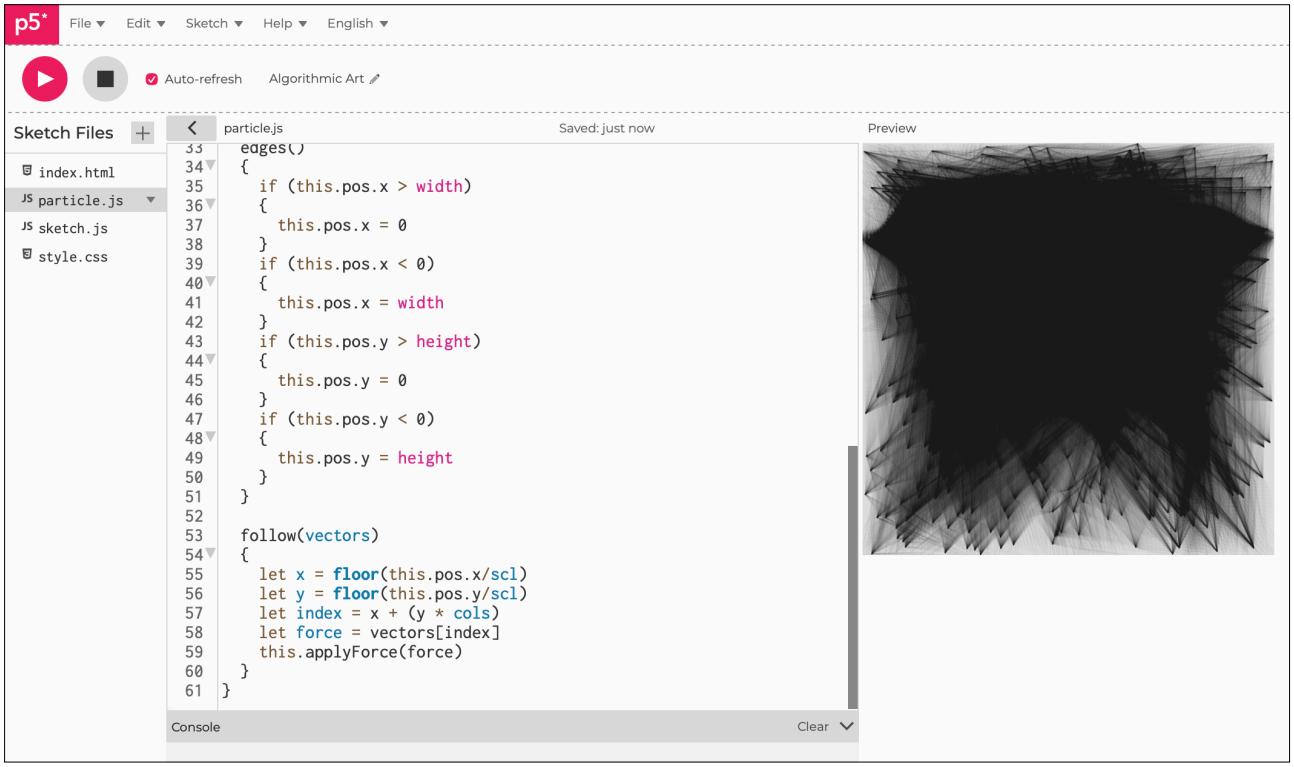
```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
    this.prevPos = this.pos.copy()
  }

  move()
  {
    this.vel.add(this.acc)
    this.vel.limit(this.maxSpeed)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    strokeWeight(1)
    stroke(0, 5)
    line(this.pos.x, this.pos.y, this.prevPos.x, this.prevPos.y)
    // point(this.pos.x, this.pos.y)
  }
```

```
  edges()
  {
    if (this.pos.x > width)
    {
      this.pos.x = 0
    }
    if (this.pos.x < 0)
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }

  follow(vectors)
  {
    let x = floor(this.pos.x/scl)
    let y = floor(this.pos.y/scl)
    let index = x + (y * cols)
    let force = vectors[index]
    this.applyForce(force)
  }
}
```

Notes

Hmmm, very quickly we have a problem because of the edges. However, we can rectify this.

Figure E6.31



Sketch E6.32 updating everything

We create another function to update the previous position.

particle.js

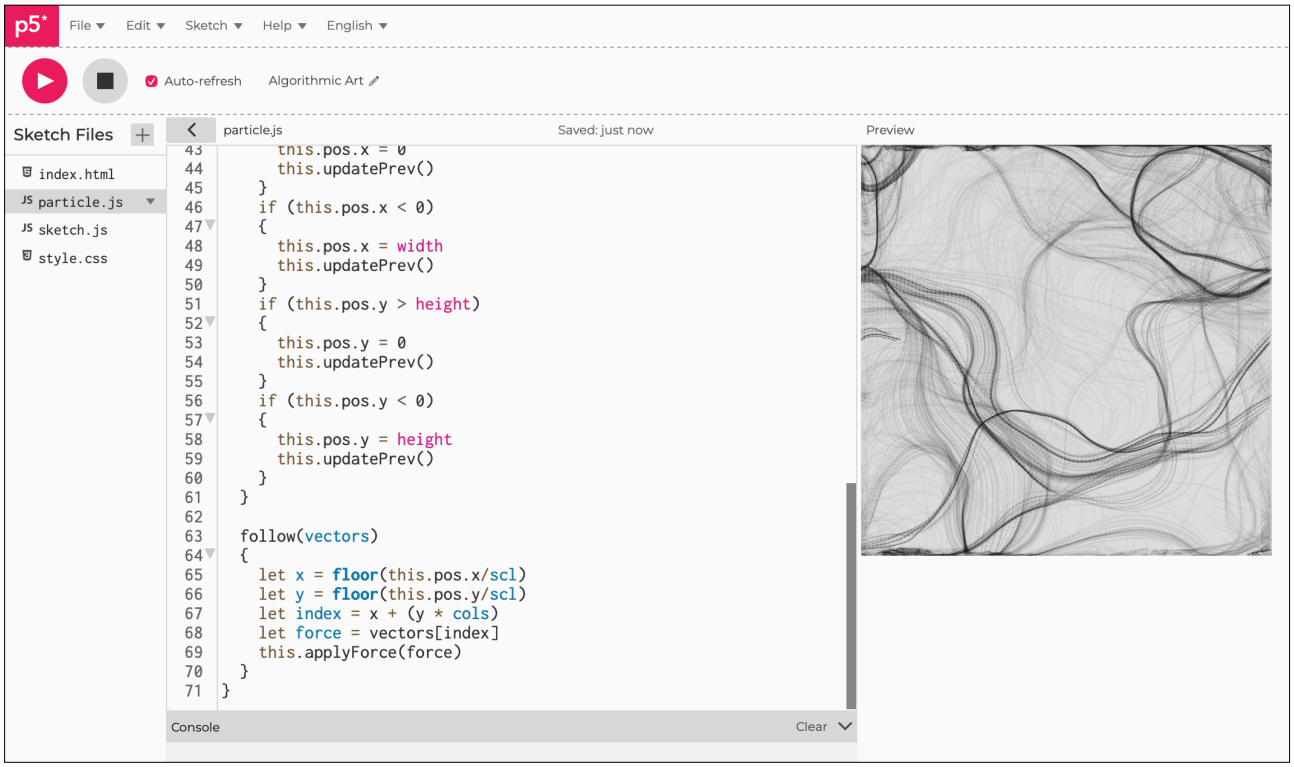
```
class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
    this.prevPos = this.pos.copy()
  }

  move()
  {
    this.vel.add(this.acc)
    this.vel.limit(this.maxSpeed)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  show()
  {
    strokeWeight(1)
    stroke(0, 5)
    line(this.pos.x, this.pos.y, this.prevPos.x, this.prevPos.y)
    this.updatePrev()
  }

  updatePrev()
  {
```

```
    this.prevPos.x = this.pos.x
    this.prevPos.y = this.pos.y
  }

  edges()
  {
    if (this.pos.x > width)
    {
      this.pos.x = 0
      this.updatePrev()
    }
    if (this.pos.x < 0)
    {
      this.pos.x = width
      this.updatePrev()
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
      this.updatePrev()
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
      this.updatePrev()
    }
  }

  follow(vectors)
  {
    let x = floor(this.pos.x/scl)
    let y = floor(this.pos.y/scl)
    let index = x + (y * cols)
    let force = vectors[index]
    this.applyForce(force)
  }
}
```

Notes

A much improved response.

Figure E6.32



Sketch E6.33 edges before show

! The sketch.js file

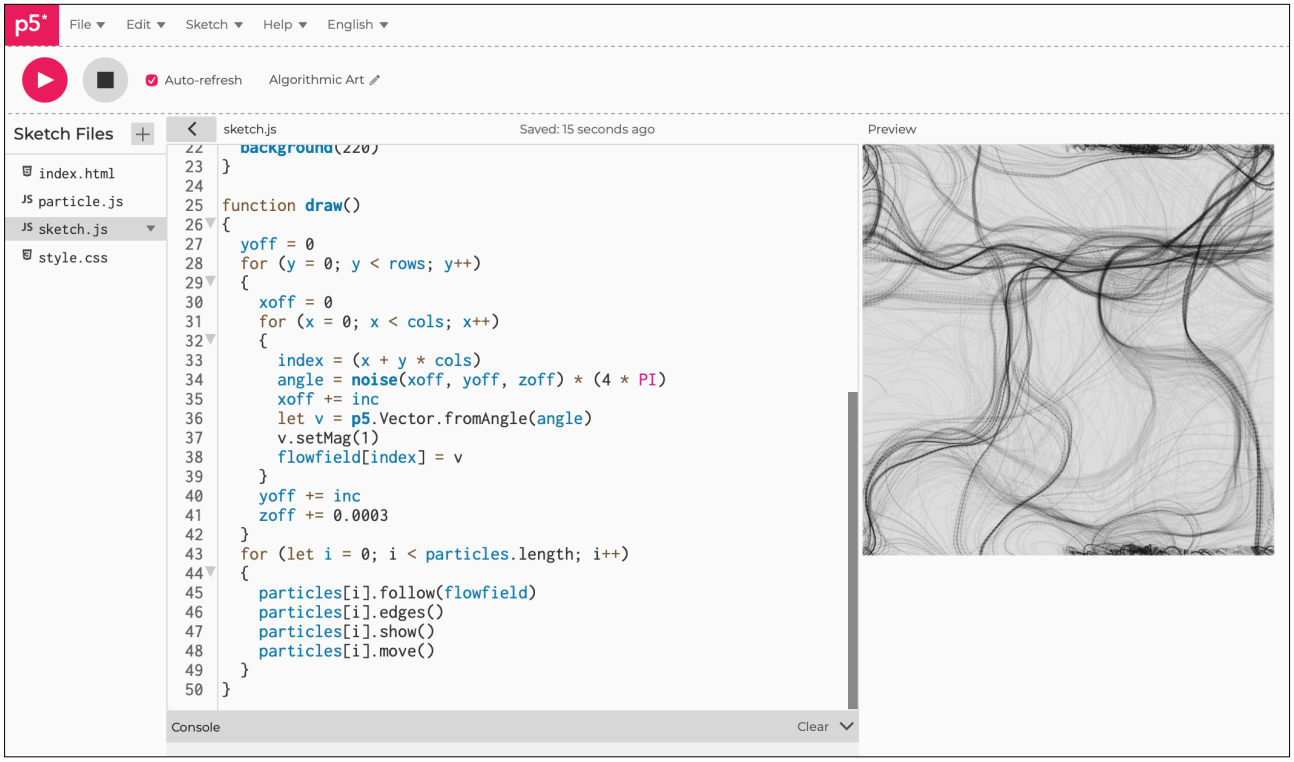
Move the edges() before show() so they are calculated first before drawing the particles/lines. Also removed the pixelDensity() and the drawing of lines (all commented out anyway //).

sketch.js

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  for (let i = 0; i < 300; i++)
  {
    particles[i] = new Particle()
  }
  background(220)
}

function draw()
{
  yoff = 0
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
```

```
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (4 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      v.setMag(1)
      flowfield[index] = v
    }
    yoff += inc
    zoff += 0.0003
  }
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].follow(flowfield)
    particles[i].edges()
    particles[i].show()
    particles[i].move()
  }
}
```

Figure E6.33


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
Sketch Files
index.html
particle.js
sketch.js
style.css
sketch.js
Saved: 15 seconds ago
Preview
background(220)
}
function draw()
{
yoff = 0
for (y = 0; y < rows; y++)
{
xoff = 0
for (x = 0; x < cols; x++)
{
index = (x + y * cols)
angle = noise(xoff, yoff, zoff) * (4 * PI)
xoff += inc
let v = p5.Vector.fromAngle(angle)
v.setMag(1)
flowfield[index] = v
}
yoff += inc
zoff += 0.0003
}
for (let i = 0; i < particles.length; i++)
{
particles[i].follow(flowfield)
particles[i].edges()
particles[i].show()
particles[i].move()
}
}
Console
Clear

Challenges

To improve the effect, I suggest altering the following:

1. Change the max force in `setMag()` value.
2. Alter `xoff`.
3. Change the amount of `alpha`.
4. Change the `strokeWeight()` to `0.5`.
5. Reduce the `maxSpeed`.
6. Change the number of particles.
7. Background to black and the lines to white/grey.
8. Introduce coloured lines.

All the above are tweaks to the parameters; an example below of what I mean.