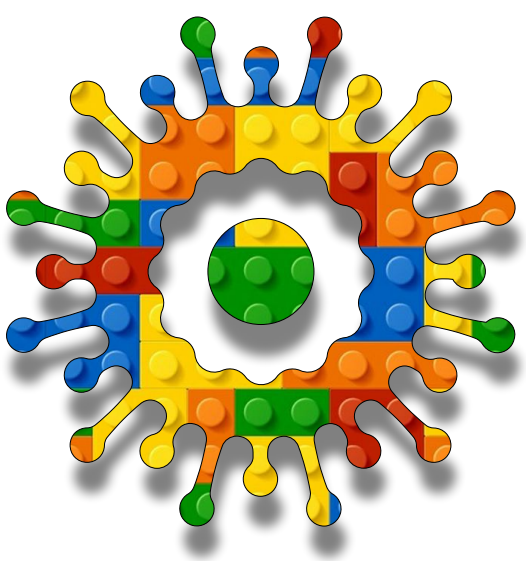
Artificial Intelligence Module C Unit #4 flappy bird



Module C Unit #4 flappy bird

Sketch C4.31 eliminating silly birds

Introduction to flappy bird neuroevolution

Flappy Bird is a relatively simple and yet very addictive game. It is a game that many emulate in various coding languages, and p5.js is no exception. In this module, we will start from scratch but not make a full working game. The basic game has a bird that flaps its wings and goes up when you tap the space bar (or click the mouse) and falls towards the ground when you stop. It has to fly through a series of gaps created by random pipes. The idea is to make it through all the pipes without hitting one for as long as possible.

We are going to give the bird a brain, a neural network, and see if we can train it to play the game through using a genetic algorithm approach, which is a form of reinforcement learning. This is another neuroevolution solution similar to the smart cars example previously. That one was a regression task; this one is a classification task.

In the same process as before, those birds that last the longest will have the best fitness scores and will pass on their genes (weights) to the next generation (population).

Each bird will have a brain that has a random set of weights. We want to use. The following data in our neural network as inputs:

1. The y position of the bird
2. The bird's velocity
3. The position of the top (or bottom) pipe
4. The x position of the pipes

The reason that we can have four and not five inputs is that the distance between the position of the top and bottom pipe is a constant (gap). This is a classification problem because the output is either to jump up or not to jump; our outputs are:

1. jump
2. not jump

To summarise our bird brain, we could have a hidden layer of, say, eight nodes, so the feedforward neural network will have the following:

- 4 inputs
- 8 hidden nodes
- 2 outputs
- plus 2 biases

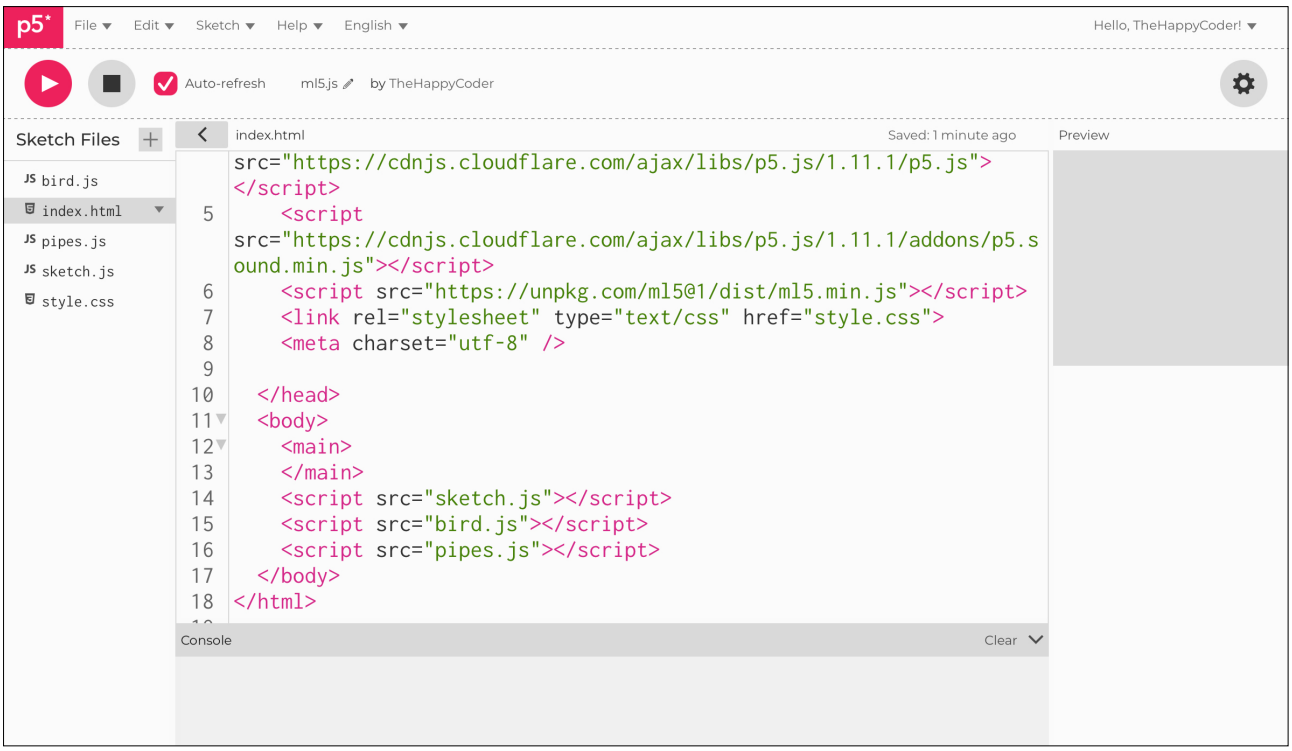
The basic game

First, we need to code the basic game before we do anything clever with neural networks and genetic algorithms. We need two files on top of the sketch.js main file. We also need to add the ml5.js line of code to the index.html file. The two named files are:

- bird.js
- pipes.js

You will have done this in the previous unit, so I won't repeat the process here in any detail. What you should have is shown in figure 1 below.

Figure 1



The index.html file (don't forget)

Adding the ml5.js line of code, as well as the files.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.1/p5.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.1/addons/p5.sound.min.js"></script>
    <script src="https://unpkg.com/ml5@1/dist/ml5.min.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="bird.js"></script>
    <script src="pipes.js"></script>
  </body>
</html>
```

Sketch C4.1 starting sketch

Our main starting sketch in `sketch.js`, with a wider and thinner canvas of `600` by `200`.

sketch.js

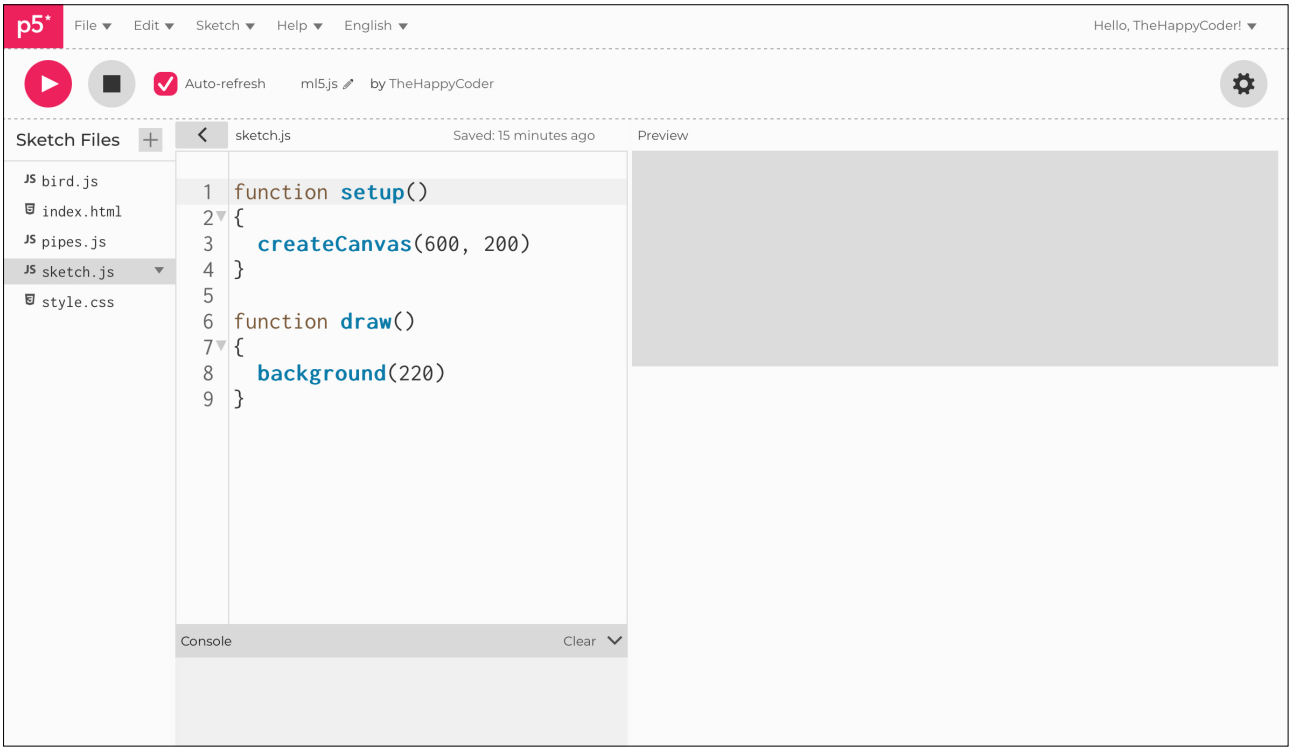
```
function setup()
{
  createCanvas(600, 200)
}

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

Notes

We have a long, thin canvas because of the nature of the game; you can do it with `400` by `400` if you wish.

Figure C4.1


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
Sketch Files
bird.js
index.html
pipes.js
sketch.js
style.css
sketch.js
Saved: 15 minutes ago
Preview
1 function setup()
2 {
3 createCanvas(600, 200)
4 }
5
6 function draw()
7 {
8 background(220)
9 }
Console
Clear

Sketch C4.2 Bird class

❗ Move to `bird.js`.

In `bird.js`, we are going to create a `Bird` class with a `constructor()` function. It will have an `x` horizontal value, which will stay constant as the pipes effectively come towards it rather than the other way round. A `y` vertical value will vary when the mouse is clicked. The `velocity` will be zero initially. There will be some `gravity` so that it falls to the ground when the mouse is not pressed, and a `force` that pushes it upwards against `gravity` when the mouse is pressed.

bird.js

```
class Bird
{
  constructor()
  {
    this.x = 50
    this.y = 120
    this.velocity = 0
    this.gravity = 0.5
    this.force = -10
  }
}
```

🗒️ Notes

Defining and initialising all the variables we need.

🛠️ Code Explanation

`this.x = 50`	Fixed distance from the lefthand edge
`this.y = 120`	Starting vertical position
`this.velocity = 0`	Starting velocity
`this.gravity = 0.5`	Arbitrary gravity value acting downwards
`this.force = -10`	Flap force acting upwards when mouse clicked, hence negative value

Sketch C4.3 gravity

When the bird flaps its wings (metaphorically speaking), the `force` (a form of acceleration) is added to the vertical `velocity`. `Gravity` is acting on the bird all the time (another acceleration), and the resultant `velocity` is added to the `y` component of its position. And to give some realism(!), we add some damping (`0.95`). Also, when it lands on the floor, it just stays there. The bird is represented by a circle. We will now have our usual functions `move()` and `show()` plus another one called `flap()` which is the equivalent to the applyForce() previously.

bird.js

```
class Bird
{
  constructor()
  {
    this.x = 50
    this.y = 120
    this.velocity = 0
    this.gravity = 0.5
    this.force = -10
  }

  flap()
  {
    this.velocity += this.force
  }

  move()
  {
    this.velocity += this.gravity
    this.y += this.velocity
    this.velocity *= 0.95
```

```
    if (this.y > height)
    {
      this.y = height
      this.velocity = 0
    }
  }

  show()
  {
    stroke(0)
    noFill()
    circle(this.x, this.y, 20)
  }
}
```

🗒️ Notes

At this point, we are just creating the game and doing it quickly. We want to focus on the main event, which is the neuroevolution part. For now, just take the time to understand the code without there being too much explanation at this point from me.

🛠️ Code Explanation

`this.velocity += this.force`	This is the upward force (negative 10) when the bird flaps its wings
`this.velocity += this.gravity`	Adding gravity to the velocity
`this.y += this.velocity`	Adding the velocity to the vertical position
`this.velocity *= 0.95`	Adding a bit of simulation
`if (this.y > height)`	Checks to see if it has hit the floor
`this.y = height`	Stays on the floor
`this.velocity = 0`	Reduce the velocity to zero on the floor until it flaps

Sketch C4.4 pipes

❗ Go to the pipes.js file.

Now, to add the `pipes`, we have two pipes, one at the top and one at the bottom. We have a `constructor()` function for the pipes, which will have a vertical space between the pipes (`100`), a random top position for a pipe, and a corresponding bottom position for a pipe. Each pipe will be `20` wide, and the velocity of the pipes has the value of `2`.

pipes.js

```
class Pipe
{
  constructor()
  {
    this.spacing = 100
    this.top = random(height - this.spacing)
    this.bottom = this.top + this.spacing
    this.x = width
    this.w = 20
    this.velocity = 2
  }
}
```

🗒️ Notes

This just sets the variables for the simple pipes.

🛠️ Code Explanation

`this.spacing = 100`	This is the gap between the top and bottom pipe and is what flappy bird is going to fly through
`this.top = random(height - this.spacing)`	The position of the top pipe, at some random position
`this.bottom = this.top + this.spacing`	The bottom pipe is relevant to the top pipe plus the gap (spacing)
`this.x = width`	The pipe starts at the far righthand edge
`this.w = 20`	The width of the pipe
`this.velocity = 2`	The velocity of the pipe moving

Sketch C4.5 drawing the pipes

We will draw the pipes as simple rectangles. Remember, we aren't trying to recreate all the full features of the game, just the minimum. The `velocity` for the pipes is not the same as the velocity for the bird going up and down. We add the `show()` and `move()` functions.

pipes.js

```
class Pipe
{
  constructor()
  {
    this.spacing = 100
    this.top = random(height - this.spacing)
    this.bottom = this.top + this.spacing
    this.x = width
    this.w = 20
    this.velocity = 2
  }

  show()
  {
    fill(51)
    noStroke()
    rect(this.x, 0, this.w, this.top)
    rect(this.x, this.bottom, this.w, height - this.bottom)
  }

  move()
  {
    this.x -= this.velocity
  }
}
```

🗒️ Notes

We won't see the bird or the pipes just yet; we will need to add these functions to the main sketch.

🛠️ Code Explanation

`rect(this.x, 0, this.w, this.top)`	Top pipe rectangle
`rect(this.x, this.bottom, this.w, height - this.bottom)`	Bottom pipe rectangle
`this.x -= this.velocity`	Moves the pipes from right to left hence the negative increment

Sketch C4.6 collision

We need to know when the bird collides with the pipes. We will put this collision in the `pipe.js` class. `vColl` means vertical collision and logically `hColl` means horizontal collision. If you think through where the bird (as a single point `(x, y)`) is in relation to the pipes, you can understand the collision part. The argument (`bird`) will be the position of that one particular bird, as there will be many of them at the start.

pipes.js

```
class Pipe
{
  constructor()
  {
    this.spacing = 100
    this.top = random(height - this.spacing)
    this.bottom = this.top + this.spacing
    this.x = width
    this.w = 20
    this.velocity = 2
  }

  show()
  {
    fill(51)
    noStroke()
    rect(this.x, this.w, this.top)
    rect(this.x, this.bottom, this.w, height - this.bottom)
  }

  move()
  {
    this.x -= this.velocity
```

```
  }


  collides(bird)
  {
    let vColl = bird.y < this.top || bird.y > this.bottom
    let hColl = bird.x > this.x && bird.x < this.x + this.w
    return vColl && hColl
  }
}
```

🗒️ Notes

You will realise that nothing happens when you try to run this because we haven't put anything in `draw()` just yet. So, let's get something happening on the canvas, let's make a playable, if rudimentary, Flappy Bird.

🛠️ Code Explanation

`vColl = bird.y < this.top || bird.y > this.bottom`	Checks to see if either one of these two condition as are true
`hColl = bird.x > this.x && bird.x < this.x + this.w`	Checks to see if both conditions are true at the same time
`return vColl && hColl`	Returns true or false

Sketch C4.7 drawing the bird!

❗ Return to `sketch.js`.

We need to create the bird, which will be a simple circle, and lots of pipes moving across the canvas. Each bird will be a separate bird, whereas we will have an array to keep track of the pipes.

sketch.js

```
let bird
let pipes = []

function setup()
{
  createCanvas(600, 200)
  bird = new Bird()
  pipes.push(new Pipe())
}

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

🗒️ Notes

Nowt (northern for nothing) to see yet.

🛠️ Code Explanation

`let bird`	Our bird variable
`let pipes = []`	An empty array of pipes
`bird = new Bird()`	Create a new bird
`pipes.push(new Pipe())`	Adding new pipes to the array

Sketch C4.8 mouse pressed

To make the bird fly, we click the mouse (usually, we press the space bar).

sketch.js

```
let bird
let pipes = []

function setup()
{
  createCanvas(600, 200)
  bird = new Bird()
  pipes.push(new Pipe())
}

function mousePressed()
{
  bird.flap()
}

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

🗒️ Notes

Still nothing to see yet.

🛠️ Code Explanation

`function mousePressed()`	Checks to see if mouse is clicked (pressed)
`bird.flap()`	Calls the flap() function in Bird class

Sketch C4.9 oops

Now, to draw the pipes and the bird, we will have to say `oops` when we hit the pipe and remember to press the mouse to fly/jump. We draw a `pipe` every `100` frames using modulo (`%`) and by counting the number of frames.

sketch.js

```
let bird
let pipes = []

function setup()
{
  createCanvas(600, 200)
  bird = new Bird()
  pipes.push(new Pipe())
}

function mousePressed()
{
  bird.flap()
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
    pipes[i].show()
    pipes[i].move()
    if (pipes[i].collides(bird))
    {
      text("OOPS", 50, height/2)
```

```
    }
  }
  bird.move()
  bird.show()
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}
```

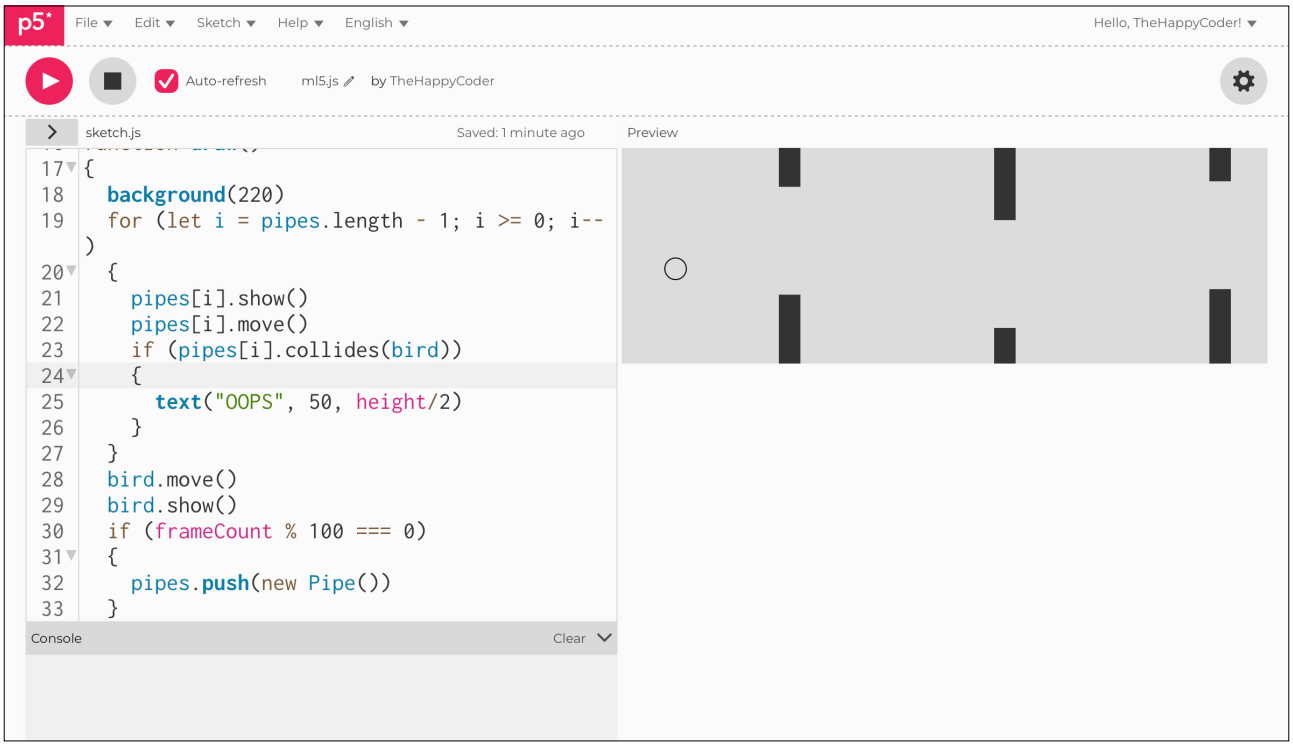
🗒️ Notes

If you click the mouse (or mousepad), the bird (circle) should move upwards, and when it hits a pipe, you get an oops on the canvas. The for() loop for the pipes works backwards so that it encounters the next pipe in the array, which is being added to constantly. The array will just keep on growing, something we will address later on. Th frame rate is how often the screen refreshes; it is a continuous number.

🛠️ Code Explanation

`for (let i = pipes.length - 1; i >= 0; i--)`	This has the effect of always being then next pipe approaching the bird
`pipes[i].show()`	Draws each set of pipes created
`pipes[i].move()`	Moves that pipe
`if (pipes[i].collides(bird))`	Checks to see if the collision conditions have been met
`text("OOPS", 50, height/2)`	If it has then text oops on canvas
`if (frameCount % 100 === 0)`	The % gives you the remainder, so if it is exactly divisible by a hundred then. . .
`pipes.push(new Pipe())`	. . .make a new pipe

Figure C4.9


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 1 minute ago
Preview
{
background(220)
for (let i = pipes.length - 1; i >= 0; i--
)
{
pipes[i].show()
pipes[i].move()
if (pipes[i].collides(bird))
{
text("OOPS", 50, height/2)
}
}
bird.move()
bird.show()
if (frameCount % 100 === 0)
{
pipes.push(new Pipe())
}
Console
Clear

Sketch C4.10 culling the array

We have a slight problem. When the pipes go off the edge of the canvas, they need to be removed; otherwise, the array will just get bigger and bigger. Eventually, over time, the game will slow down. We use `splice()` to remove them from the array.

sketch.js

```
let bird
let pipes = []

function setup()
{
  createCanvas(600, 200)
  bird = new Bird()
  pipes.push(new Pipe())
}

function mousePressed()
{
  bird.flap()
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
    pipes[i].show()
    pipes[i].move()
    if (pipes[i].collides(bird))
    {
      text("OOPS", 50, height/2)
```

```
    }
    if (pipes[i].offscreen())
    {
      pipes.splice(i, 1)
    }
  }
  bird.move()
  bird.show()
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}
```

🗒️ Notes

You will get an error because we haven't created the `offscreen()` function in `pipe.js` yet.

🛠️ Code Explanation

`for (let i = pipes.length - 1; i >= 0; i--)`	Cycles backwards through the array of pipes
`if (pipes[i].offscreen())`	Checks to see if the pipe has gone off the canvas
`pipes.splice(i, 1)`	It it has remove that pipe

Sketch C4.11 splice the pipe

❗ Wander over to `pipes.js`.

The pipe is removed from the array if the `x` value of that pipe (`this.x`) is less than `-20` (the width of the pipe); in other words, it has fully gone off the edge of the canvas. Here we add the `offscreen()` function.

pipes.js

```
class Pipe
{
  constructor()
  {
    this.spacing = 100
    this.top = random(height - this.spacing)
    this.bottom = this.top + this.spacing
    this.x = width
    this.w = 20
    this.velocity = 2
  }

  show()
  {
    fill(51)
    noStroke()
    rect(this.x, 0, this.w, this.top)
    rect(this.x, this.bottom, this.w, height - this.bottom)
  }

  move()
  {
    this.x -= this.velocity
  }
```

```
  collides(bird)
  {
    let vColl = bird.y < this.top || bird.y > this.bottom
    let hColl = bird.x > this.x && bird.x < this.x + this.w
    return vColl && hColl
  }

  offscreen()
  {
    return this.x < -this.w
  }
}
```

🗒️ Notes

It should work perfectly now. This is all to do with managing its memory. Your browser only has so much memory, and clogging it up like this is not great and may crash or grind to a halt eventually.

🛠️ Code Explanation

`return this.x < -this.w`	Checks to see if this condition is true

Adding the brain

All we have done so far is make a simplified functional version of the game. We are now going to develop the idea of a genetic algorithmic approach to getting the bird to fly through the pipes perfectly without hardcoding it, but instead using a neuroevolution technique.

This is very similar to the smart cars, so it isn't completely new for you (assuming you completed that unit). Just make sure you have the ml5.js installed in the index.html file. I will still walk you through it just in case you haven't or have forgotten.

Sketch C4.12 simple bird brain

❗ Go to `bird.js`.

We are going to give the bird a `brain` as an ml5.js neural network. We place this brain in the `constructor()` part of the class. When you run this, you should still get the same, but check for any error messages in the console to make sure you have typed everything in OK.

bird.js

```
class Bird
{
  constructor()
  {
    ml5.setBackend("webgl")
    this.brain = ml5.neuralNetwork(
    {
      inputs: 4,
      outputs: ["flap", "no flap"],
      task: "classification",
      noTraining: true,
      neuroEvolution: true
    })
    this.x = 50
    this.y = 120
    this.velocity = 0
    this.gravity = 0.5
    this.force = -10
  }

  flap()
  {
    this.velocity += this.force
```

```
  }

  move()
  {
    this.velocity += this.gravity
    this.y += this.velocity
    this.velocity *= 0.95
    if (this.y > height)
    {
      this.y = height
      this.velocity = 0
    }
  }
  show()
  {
    stroke(0)
    noFill()
    circle(this.x, this.y, 20)
  }
}
```

🗒️ Notes

We haven't connected the bird and brain together yet; the bird has no brain!

🛠️ Code Explanation

`inputs: 4`	There are four inputs
`outputs: ["flap", "no flap"]`	Our two outputs
`task: "classification"`	Task declared
`noTraining: true`	Not doing any training
`neuroEvolution: true`	Declaring it is a neuro evolution network

Sketch C4.13 next pipe please

This is where we need to think a bit about what is happening. Once the pipe has gone past the bird, it is no longer relevant. We already delete it when it goes off the canvas. But we need a way of checking the distance to the next one in front of the bird, not just the one at the front of the array. So in `bird.js`, we need a function to measure that. The break command simply terminates that checking loop. The new function `think(pipes)` has the argument from the array of pipes.

bird.js

```
class Bird
{
  constructor()
  {
    ml5.setBackend("webgl")
    this.brain = ml5.neuralNetwork(
    {
      inputs: 4,
      outputs: ["flap", "no flap"],
      task: "classification",
      noTraining: true,
      neuroEvolution: true
    })
    this.x = 50
    this.y = 120
    this.velocity = 0
    this.gravity = 0.5
    this.force = -10
  }

  think(pipes)
  {
```

```
    let nextPipe = null
    for (let pipe of pipes)
    {
      if (pipe.x + pipe.w > this.x)
      {
        nextPipe = pipe
        break
      }
    }
  }

  flap()
  {
    this.velocity += this.force
  }

  move()
  {
    this.velocity += this.gravity
    this.y += this.velocity
    this.velocity *= 0.95
    if (this.y > height)
    {
      this.y = height
      this.velocity = 0
    }
  }
  show()
  {
    stroke(0)
    noFill()
    circle(this.x, this.y, 20)
```

```
  }
}
```

🗒️ Notes

Just check each pipe in turn.

🛠️ Code Explanation

`nextPipe = null`	The term null means empty or no value
`for (let pipe of pipes)`	Goes through the array of pipes pulling out a pipe at a time
`nextPipe = pipe`	The nextPipe now has a value
`break`	Stops the loop

Sketch C4.14 input data

Now we have the next pipe, we can add it to the data inputs for our neural network. These are our four inputs.

bird.js

```
class Bird
{
  constructor()
  {
    ml5.setBackend("webgl")
    this.brain = ml5.neuralNetwork(
    {
      inputs: 4,
      outputs: ["flap", "no flap"],
      task: "classification",
      noTraining: true,
      neuroEvolution: true
    })
    this.x = 50
    this.y = 120
    this.velocity = 0
    this.gravity = 0.5
    this.force = -10
  }

  think(pipes)
  {
    let nextPipe = null
    for (let pipe of pipes)
    {
      if (pipe.x + pipe.w > this.x)
```

```
    {
      nextPipe = pipe
      break
    }
  }
  let inputs = [
    this.y,
    this.velocity,
    nextPipe.top,
    nextPipe.x - this.x
  ]
}

flap()
{
  this.velocity += this.force
}

move()
{
  this.velocity += this.gravity
  this.y += this.velocity
  this.velocity *= 0.95
  if (this.y > height)
  {
    this.y = height
    this.velocity = 0
  }
}

show()
{
```

```
    stroke(0)
    noFill()
    circle(this.x, this.y, 20)
  }
}
```

🗒️ Notes

We now have our four `inputs`, and the `nextPipe` will be the pipe it sees.

🛠️ Code Explanation

`this.y`	Input 1: the vertical position
`this.velocity`	Input 2: the current velocity of the bird
`nextPipe.top`	Input 3: the vertical position of the top pipe
`nextPipe.x - this.x`	Input 4: how far away is the next pipe

Sketch C4.15 normalise

To `normalise` all the `inputs`, we can divide any vertical measurements by the height, which are the first three `inputs`. The fourth `input` is the horizontal distance, so one is divided by the width. This gives us values between `0` and `1`.

bird.js

```
class Bird
{
  constructor()
  {
    ml5.setBackend("webgl")
    this.brain = ml5.neuralNetwork(
    {
      inputs: 4,
      outputs: ["flap", "no flap"],
      task: "classification",
      noTraining: true,
      neuroEvolution: true
    })
    this.x = 50
    this.y = 120
    this.velocity = 0
    this.gravity = 0.5
    this.force = -10
  }

  think(pipes)
  {
    let nextPipe = null
    for (let pipe of pipes)
    {
```

```
    if (pipe.x + pipe.w > this.x)
    {
      nextPipe = pipe
      break
    }
  }
  let inputs = [
    this.y / height,
    this.velocity / height,
    nextPipe.top / height,
    (nextPipe.x - this.x) / width
  ]
}

flap()
{
  this.velocity += this.force
}

move()
{
  this.velocity += this.gravity
  this.y += this.velocity
  this.velocity *= 0.95
  if (this.y > height)
  {
    this.y = height
    this.velocity = 0
  }
}

show()
```

```
  {
    stroke(0)
    noFill()
    circle(this.x, this.y, 20)
  }
}
```

🗒️ Notes

Everything is now normalised.

Sketch C4.16 synchronicity

We want the code to run synchronously so that the model waits for the inputs before carrying on. We use something a bit useful in ml5.js called classifySync(), which is the same as classify but runs synchronously. If the result is classified as flap, then the function flap() is run; otherwise, nothing will happen.

bird.js

```
class Bird
{
  constructor()
  {
    ml5.setBackend("webgl")
    this.brain = ml5.neuralNetwork(
    {
      inputs: 4,
      outputs: ["flap", "no flap"],
      task: "classification",
      noTraining: true,
      neuroEvolution: true
    })
    this.x = 50
    this.y = 120
    this.velocity = 0
    this.gravity = 0.5
    this.force = -10
  }

  think(pipes)
  {
    let nextPipe = null
    for (let pipe of pipes)
```

```
    {
      if (pipe.x + pipe.w > this.x)
      {
        nextPipe = pipe
        break
      }
    }
    let inputs = [
      this.y / height,
      this.velocity / height,
      nextPipe.top / height,
      (nextPipe.x - this.x) / width
    ]
    let results = this.brain.classifySync(inputs)
    if (results[0].label === "flap")
    {
      this.flap()
    }
  }

  flap()
  {
    this.velocity += this.force
  }

  move()
  {
    this.velocity += this.gravity
    this.y += this.velocity
    this.velocity *= 0.95
    if (this.y > height)
    {
```

```
      this.y = height
      this.velocity = 0
    }
  }

  show()
  {
    stroke(0)
    noFill()
    circle(this.x, this.y, 20)
  }
}
```

🗒️ Notes

Do remember that there is no training; each bird has a different set of random weights. The outputs will be effectively random despite having the data inputs; it has no idea it has to jump or even why it should jump; it does not even know the rules of the game.

🛠️ Code Explanation

`results = this.brain.classifySync(inputs)`	From the input data we classify the output (results)
`if (results[0].label === "flap")`	If the first result is flap then. . .
`this.flap()`	. . .flap like a bird

Sketch C4.17 population of birds

❗ Return to `sketch.js`.

We only currently have one bird. We need a population of birds all trying to be the best bird they can be. We will make an array of `200` new birds. We need an array of birds, and so we use a `for()` loop to create this population of birds.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
}

function mousePressed()
{
  bird.flap()
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
```

```
  {
    pipes[i].show()
    pipes[i].move()
    if (pipes[i].collides(bird))
    {
      text("OOPS", 50, height/2)
    }
    if (pipes[i].offscreen())
    {
      pipes.splice(i, 1)
    }
  }
  bird.move()
  bird.show()
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}
```

🗒️ Notes

No point in running it yet; we have a few more amendments to make.

🛠️ Code Explanation

`let birds = []`	An empty array of birds replaces singular bird
`for (let i = 0; i < population; i++)`	Creating a population of birds
`birds[i] = new Bird()`	Adding the birds to the array

Sketch C4.18 to flap or not to flap

We are now going to get the bird to think whether it should flap or not, update its decision, and then show us what it can do. However, it is still too early to run this just yet; we are still laying the foundation. Put `bird.move()` and `bird.show()` inside a new `for-of()` loop and call the `think(pipes)` function.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
}

function mousePressed()
{
  bird.flap()
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
```

```
    pipes[i].show()
    pipes[i].move()
    if (pipes[i].collides(bird))
    {
      text("OOPS", 50, height/2)
    }
    if (pipes[i].offscreen())
    {
      pipes.splice(i, 1)
    }
  }
  for (let bird of birds)
  {
    bird.think(pipes)
    bird.move()
    bird.show()
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}
```

🗒️ Notes

We cycle through all the birds one at a time.

Sketch C4.19 GPU v CPU

We can use the GPU or the CPU. I won't go into the difference here except to say that we want to use the CPU for better performance. The GPU is better when there is a heavy demand for graphics in some games, for instance. The `.tf` reference is for TensorFlow, which underpins ml5.js. TensorFlow is used with Python for machine learning, but there is a JavaScript version called TensorFlow.js, which is compatible with ml5.js and p5.js.

❗ If it is still running slow or very jerky, then suggest reducing the number of birds, or try `webgl`.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
  ml5.tf.setBackend("cpu")
}

function mousePressed()
{
  bird.flap()
}
```

```
function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
    pipes[i].show()
    pipes[i].move()
    if (pipes[i].collides(bird))
    {
      text("OOPS", 50, height/2)
    }
    if (pipes[i].offscreen())
    {
      pipes.splice(i, 1)
    }
  }
  for (let bird of birds)
  {
    bird.think(pipes)
    bird.move()
    bird.show()
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}
```

🗒️ Notes

I found that this did make quite a difference depending on what device/machine I used. The other alternative might be webgl. Play around and see what works best for you.

Sketch C4.20 bird fitness

❗ Hop over to `bird.js`.

Next job is to define and find the `fitness` of each bird. We need to add two more features, `fitness` and `alive`, in the `constructor()` function. The bird's `fitness` obviously increases the longer it is `alive`; hence, `fitness` is a number, but `alive` is a boolean; it is either alive or not, think of `oops` being a bit more terminal. We increment the `fitness` in the `move()` function.

bird.js

```
class Bird
{
  constructor()
  {
    ml5.setBackend("webgl")
    this.brain = ml5.neuralNetwork(
    {
      inputs: 4,
      outputs: ["flap", "no flap"],
      task: "classification",
      noTraining: true,
      neuroEvolution: true
    })
    this.x = 50
    this.y = 120
    this.velocity = 0
    this.gravity = 0.5
    this.force = -10
    this.fitness = 0
    this.alive = true
  }
```

```
think(pipes)
{
  let nextPipe = null
  for (let pipe of pipes)
  {
    if (pipe.x + pipe.w > this.x)
    {
      nextPipe = pipe
      break
    }
  }
  let inputs = [
    this.y / height,
    this.velocity / height,
    nextPipe.top / height,
    (nextPipe.x - this.x) / width
  ]
  let results = this.brain.classifySync(inputs)
  if (results[0].label === "flap")
  {
    this.flap()
  }
}

flap()
{
  this.velocity += this.force
}

move()
{
  this.velocity += this.gravity
```

```
    this.y += this.velocity
    this.velocity *= 0.95
    if (this.y > height)
    {
      this.y = height
      this.velocity = 0
    }
    this.fitness++
  }

  show()
  {
    stroke(0)
    noFill()
    circle(this.x, this.y, 20)
  }
}
```

🗒️ Notes

When the bird hits the pipe, we will want to eliminate it, remove it (sounds less brutal).

🛠️ Code Explanation

`this.fitness = 0`	Each bird has a fitness of zero
`this.alive = true`	Each bird is initialised as being alive at the start
`this.fitness++`	Increment the bird's fitness on each iteration.

Sketch C4.21 the bird is dead

❗ Return to sketch.js.

However, we need a dead bird (sorry) when it hits a pipe. So, we have to do a bit of refactoring to incorporate this here and also in the next sketch. We have a nested loop to achieve this.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
  ml5.tf.setBackend("cpu")
}

function mousePressed()
{
  bird.flap()
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
```

```
  {
    pipes[i].show()
    pipes[i].move()
    if (pipes[i].collides(bird))
    {
      text("OOPS", 50, height/2)
    }
    if (pipes[i].offscreen())
    {
      pipes.splice(i, 1)
    }
  }
  for (let bird of birds)
  {
    if (bird.alive)
    {
      bird.think(pipes)
      bird.move()
      bird.show()
    }
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}
```

🗒️ Notes

The only birds left are those which are alive (true), obviously.

🛠️ Code Explanation

`if (bird.alive)`	Checks to see if this bird is alive (true) or dead (false)

Sketch C4.22 collision

Now, when the bird collides with the pipe. We do a bit more reconfiguring. Also, we remove the lines of code which are commented out (//). We are no longer operating the birds with the mouse. We eliminate the birds if they collide with the pipe.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
  ml5.tf.setBackend("cpu")
}

// function mousePressed()
// {
//   bird.flap()
// }

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
```

```
    pipes[i].show()
    pipes[i].move()
    // if (pipes[i].collides(bird))
    // {
    //   text("OOPS", 50, height/2)
    // }
    if (pipes[i].offscreen())
    {
      pipes.splice(i, 1)
    }
  }
  for (let bird of birds)
  {
    if (bird.alive)
    {
      for (let pipe of pipes)
      {
        if (pipe.collides(bird))
        {
          bird.alive = false
        }
      }
      bird.think(pipes)
      bird.move()
      bird.show()
    }
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}
```

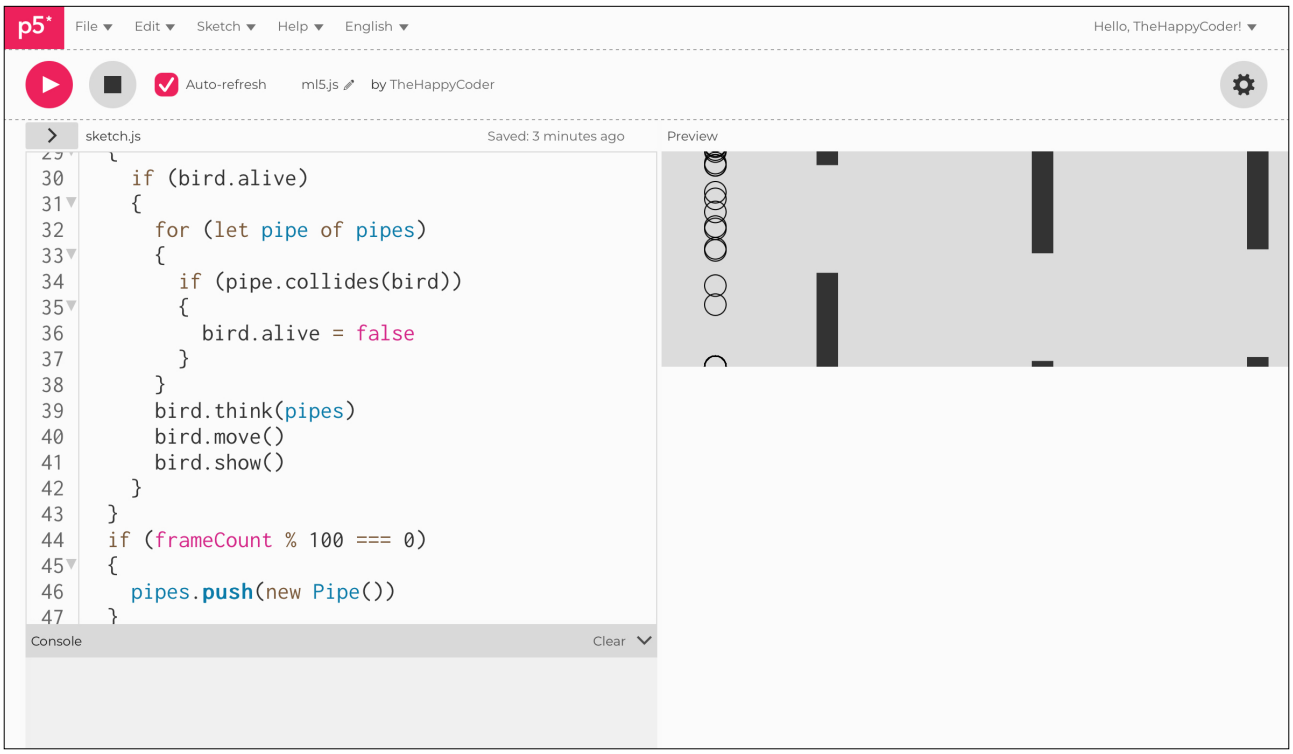
🗒️ Notes

We now have a working population of birds trying to get through the pipes; however, they only last for one population, and once they have all died, there are none more. We want to reproduce the next generation.

🛠️ Code Explanation

`for (let pipe of pipes)`	Goes through each pipe in turn
`if (pipe.collides(bird))`	Checks for any collisions
`bird.alive = false`	If a collision occurs that bird is now dead (false)

Figure C4.22


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 3 minutes ago
Preview
if (bird.alive)
{
for (let pipe of pipes)
{
if (pipe.collides(bird))
{
bird.alive = false
}
}
bird.think(pipes)
bird.move()
bird.show()
}
}
if (frameCount % 100 === 0)
{
pipes.push(new Pipe())
}
Console
Clear

Sketch C4.23 is anyone there

There is no predetermined lifespan. So they live as long as they don't hit a pipe, and when they have all died out, then we select the best ones. We create a function to check if there are any birds left; if there are still some birds flapping, it returns false and carries on checking.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
  ml5.tf.setBackend("cpu")
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
    pipes[i].show()
    pipes[i].move()
    if (pipes[i].offscreen())
    {
      pipes.splice(i, 1)
```

```
    }
  }
  for (let bird of birds)
  {
    if (bird.alive)
    {
      for (let pipe of pipes)
      {
        if (pipe.collides(bird))
        {
          bird.alive = false
        }
      }
      bird.think(pipes)
      bird.move()
      bird.show()
    }
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}
```

```
function allBirdsDead()
{
  for (let bird of birds)
  {
    if (bird.alive)
    {
      return false
    }
```

```
  }
  return true
}
```

🗒️ Notes

Not an imaginative name for a function, but it does what it says on the tin!

🛠️ Code Explanation

`for (let bird of birds)`	Check through all the birds
`if (bird.alive)`	If a there is an alive bird then. . .
`return false`	. . .the function allBirdsDead() is false, until. . .
`return true`	. . .there are no alive birds

Sketch C4.24 mating the best ones

What we want to do now is select the fittest birds and mate them to make even better birds for the next round. This is an evolutionary approach to selection. We will use the `weightedSelection()` algorithm that we used in smart cars. To remind you, the fittest have the advantage, but the others always have a chance, so it is a bit more like how natural selection occurs.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
  ml5.tf.setBackend("cpu")
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
    pipes[i].show()
    pipes[i].move()
    if (pipes[i].offscreen())
```

```
    {
      pipes.splice(i, 1)
    }
  }
  for (let bird of birds)
  {
    if (bird.alive)
    {
      for (let pipe of pipes)
      {
        if (pipe.collides(bird))
        {
          bird.alive = false
        }
      }
      bird.think(pipes)
      bird.move()
      bird.show()
    }
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}

function allBirdsDead()
{
  for (let bird of birds)
  {
    if (bird.alive)
    {
```

```
      return false
    }
  }
  return true
}
```

```
function weightedSelection()
{
  let index = 0
  let start = random(1)
  while (start > 0)
  {
    start = start - birds[index].fitness
    index++
  }
  index--
  return birds[index].brain
}
```

Notes

We continue to put the pieces together; we still have to normalise the fitness values. We went through the code for weighted selection in some detail in smart cars.

Sketch C4.25 normalising fitness

Now, to normalise the fitness, we use the `for-of()` loop to add up all the fitness scores of all the birds (`sum`) and then go through each one and divide by said `sum`. This means that we have fitness scores between `0` and `1`, and they all add up to `1`.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
  ml5.tf.setBackend("cpu")
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
    pipes[i].show()
    pipes[i].move()
    if (pipes[i].offscreen())
    {
      pipes.splice(i, 1)
```

```
    }
  }
  for (let bird of birds)
  {
    if (bird.alive)
    {
      for (let pipe of pipes)
      {
        if (pipe.collides(bird))
        {
          bird.alive = false
        }
      }
      bird.think(pipes)
      bird.move()
      bird.show()
    }
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}

function allBirdsDead()
{
  for (let bird of birds)
  {
    if (bird.alive)
    {
      return false
    }
```

```
  }
  return true
}

function weightedSelection()
{
  let index = 0
  let start = random(1)
  while (start > 0)
  {
    start = start - birds[index].fitness
    index++
  }
  index--
  return birds[index].brain
}
```

```
function normaliseFitness()
{
  let sum = 0
  for (let bird of birds)
  {
    sum += bird.fitness
  }
  for (let bird of birds)
  {
    bird.fitness = bird.fitness / sum
  }
}
```

🗒️ Notes

This should be fairly straightforward. Another piece of the jigsaw. The code was covered in smart cars.

Sketch C4.26 crossover

Now we tackle `crossover`, where we combine the DNA or genes of two parents that are successful or have a high `fitness` level. This could be very problematic because all you are doing is choosing the weights of one neural network or the other. What ml5.js has is a built-in function for this called `crossover()`, and we create a `reproduction()` function to calculate all this.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
  ml5.tf.setBackend("cpu")
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
    pipes[i].move()
    pipes[i].show()
    // pipes[i].move()
```

```
    if (pipes[i].offscreen())
    {
      pipes.splice(i, 1)
    }
  }
  for (let bird of birds)
  {
    if (bird.alive)
    {
      for (let pipe of pipes)
      {
        if (pipe.collides(bird))
        {
          bird.alive = false
        }
      }
      bird.think(pipes)
      bird.move()
      bird.show()
    }
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}

function allBirdsDead()
{
  for (let bird of birds)
  {
    if (bird.alive)
```

```
    {
      return false
    }
  }
  return true
}

function weightedSelection()
{
  let index = 0
  let start = random(1)
  while (start > 0)
  {
    start = start - birds[index].fitness
    index++
  }
  index--
  return birds[index].brain
}

function normaliseFitness()
{
  let sum = 0
  for (let bird of birds)
  {
    sum += bird.fitness
  }
  for (let bird of birds)
  {
    bird.fitness = bird.fitness / sum
  }
}
```

```
function reproduction()
{
  let parentA = weightedSelection()
  let parentB = weightedSelection()
  let child = parentA.crossover(parentB)
}
```

🗒️ Notes

You can see how the `crossover()` function works.

Sketch C4.27 mutation

To `mutate`, we can use another new function included in ml5.js called (guess what) `mutate()`. Mutation is important because it creates some variety to prevent a poor initial selection from causing the birds to die out prematurely. The number indicates how often a weight is altered. `0.01` is `1%` of the population.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
  ml5.tf.setBackend("cpu")
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
    pipes[i].move()
    pipes[i].show()
    // pipes[i].move()
    if (pipes[i].offscreen())
```

```
    {
      pipes.splice(i, 1)
    }
  }
  for (let bird of birds)
  {
    if (bird.alive)
    {
      for (let pipe of pipes)
      {
        if (pipe.collides(bird))
        {
          bird.alive = false
        }
      }
      bird.think(pipes)
      bird.move()
      bird.show()
    }
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}

function allBirdsDead()
{
  for (let bird of birds)
  {
    if (bird.alive)
    {
```

```
      return false
    }
  }
  return true
}

function weightedSelection()
{
  let index = 0
  let start = random(1)
  while (start > 0)
  {
    start = start - birds[index].fitness
    index++
  }
  index--
  return birds[index].brain
}

function normaliseFitness()
{
  let sum = 0
  for (let bird of birds)
  {
    sum += bird.fitness
  }
  for (let bird of birds)
  {
    bird.fitness = bird.fitness / sum
  }
}
```

```
function reproduction()
{
  let parentA = weightedSelection()
  let parentB = weightedSelection()
  let child = parentA.crossover(parentB)
  child.mutate(0.01)
}
```

🗒️ Notes

We do need some mutation.

Sketch C4.28 next generation

Once we have `crossover` and `mutation`, we need a new population. So we need a new (empty) array of birds, and we will call this `nextBirds = []`. This is part of the `reproduction()` function. After we have gone through the population and created the next birds from the children, we then call this new generation of birds the current population. This means we are rinsing and repeating with each successive population replacing the previous one, which we hope is better.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
  ml5.tf.setBackend("cpu")
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
    pipes[i].move()
    pipes[i].show()
```

```
    // pipes[i].move()
    if (pipes[i].offscreen())
    {
      pipes.splice(i, 1)
    }
  }
  for (let bird of birds)
  {
    if (bird.alive)
    {
      for (let pipe of pipes)
      {
        if (pipe.collides(bird))
        {
          bird.alive = false
        }
      }
      bird.think(pipes)
      bird.move()
      bird.show()
    }
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
}

function allBirdsDead()
{
  for (let bird of birds)
  {
```

```
    if (bird.alive)
    {
      return false
    }
  }
  return true
}

function weightedSelection()
{
  let index = 0
  let start = random(1)
  while (start > 0)
  {
    start = start - birds[index].fitness
    index++
  }
  index--
  return birds[index].brain
}

function normaliseFitness()
{
  let sum = 0
  for (let bird of birds)
  {
    sum += bird.fitness
  }
  for (let bird of birds)
  {
    bird.fitness = bird.fitness / sum
  }
```

```
}

function reproduction()
{
  let nextBirds = []
  for (let i = 0; i < population; i++)
  {
    let parentA = weightedSelection()
    let parentB = weightedSelection()
    let child = parentA.crossover(parentB)
    child.mutate(0.01)
    nextBirds[i] = new Bird(child)
  }
  birds = nextBirds
}
```

🗒️ Notes

A bit of refactoring, but a similar outcome to smart cars in the previous unit.

Sketch C4.29 a new brain

❗ Fly over to bird.js.

The new birds still have no brains, so we need to alter the Bird class to correct this omission. If the bird has no brain, then create one; if it does (because it is the next generation), then use it. The constructor() function now receives an argument (brain), which is the new child brain from the crossover.

bird.js

```
class Bird
{
  constructor(brain)
  {
    if (brain)
    {
      this.brain = brain
    }
    else
    {
      ml5.setBackend("webgl")
      this.brain = ml5.neuralNetwork(
      {
        inputs: 4,
        outputs: ["flap", "no flap"],
        task: "classification",
        noTraining: true,
        neuroEvolution: true
      })
    }
    this.x = 50
    this.y = 120
    this.velocity = 0
```

```
  this.gravity = 0.5
  this.force = -10
  this.fitness = 0
  this.alive = true
}

think(pipes)
{
  let nextPipe = null
  for (let pipe of pipes)
  {
    if (pipe.x + pipe.w > this.x)
    {
      nextPipe = pipe
      break
    }
  }
  let inputs = [
    this.y / height,
    this.velocity / height,
    nextPipe.top / height,
    (nextPipe.x - this.x) / width
  ]
  let results = this.brain.classifySync(inputs)
  if (results[0].label === "flap")
  {
    this.flap()
  }
}

flap()
{
```

```
    this.velocity += this.force
  }

  move()
  {
    this.velocity += this.gravity
    this.y += this.velocity
    this.velocity *= 0.95
    if (this.y > height)
    {
      this.y = height
      this.velocity = 0
    }
    this.fitness++
  }

  show()
  {
    stroke(0)
    noFill()
    circle(this.x, this.y, 20)
  }
}
```

🗒️ Notes

We have a brain for every generation.

Sketch C4.30 reset and off we go again

❗ Return to `sketch.js`.

Now, to complete the code to make this work. If all the birds are dead, then we need to reset the pipes. To do that, we create a new function called `resetPipes()`.

sketch.js

```
let birds = []
let pipes = []
let population = 200

function setup()
{
  createCanvas(600, 200)
  for (let i = 0; i < population; i++)
  {
    birds[i] = new Bird()
  }
  pipes.push(new Pipe())
  ml5.tf.setBackend("cpu")
}

function draw()
{
  background(220)
  for (let i = pipes.length - 1; i >= 0; i--)
  {
    pipes[i].show()
    pipes[i].move()
    if (pipes[i].offscreen())
    {
```

```
      pipes.splice(i, 1)
    }
  }
  for (let bird of birds)
  {
    if (bird.alive)
    {
      for (let pipe of pipes)
      {
        if (pipe.collides(bird))
        {
          bird.alive = false
        }
      }
      bird.think(pipes)
      bird.move()
      bird.show()
    }
  }
  if (frameCount % 100 === 0)
  {
    pipes.push(new Pipe())
  }
  if (allBirdsDead())
  {
    normaliseFitness()
    reproduction()
    resetPipes()
  }
}

function resetPipes()
```

```
{
  pipes.splice(0, pipes.length - 1)
}

function allBirdsDead()
{
  for (let bird of birds)
  {
    if (bird.alive)
    {
      return false
    }
  }
  return true
}

function weightedSelection()
{
  let index = 0
  let start = random(1)
  while (start > 0)
  {
    start = start - birds[index].fitness
    index++
  }
  index--
  return birds[index].brain
}

function normaliseFitness()
{
  let sum = 0
```

```
  for (let bird of birds)
  {
    sum += bird.fitness
  }
  for (let bird of birds)
  {
    bird.fitness = bird.fitness / sum
  }
}

function reproduction()
{
  let nextBirds = []
  for (let i = 0; i < population; i++)
  {
    let parentA = weightedSelection()
    let parentB = weightedSelection()
    let child = parentA.crossover(parentB)
    child.mutate(0.01)
    nextBirds[i] = new Bird(child)
  }
  birds = nextBirds
}
```

🗒️ Notes

The reason for deleting the pipes except the last one (`pipes.length - 1`) is because if we don't, it will try to find the distance to the next pipe, but there isn't one briefly. This stops us from getting an error message.

🛠️ Code Explanation

`pipes.splice(0, pipes.length - 1)`	Deletes all the pipes except for the last one

Sketch C4.31 eliminating silly birds

❗ Here in bird.js.

One final tweak: we can also eliminate any birds that fly off the screen or hit the ground in bird.js.

bird.js

```
class Bird
{
  constructor(brain)
  {
    if (brain)
    {
      this.brain = brain
    }
    else
    {
      ml5.setBackend("webgl")
      this.brain = ml5.neuralNetwork(
      {
        inputs: 4,
        outputs: ["flap", "no flap"],
        task: "classification",
        noTraining: true,
        neuroEvolution: true
      })
    }
    this.x = 50
    this.y = 120
    this.velocity = 0
    this.gravity = 0.5
    this.force = -10
```

```
  this.fitness = 0
  this.alive = true
}

think(pipes)
{
  let nextPipe = null
  for (let pipe of pipes)
  {
    if (pipe.x + pipe.w > this.x)
    {
      nextPipe = pipe
      break
    }
  }
  let inputs = [
    this.y / height,
    this.velocity / height,
    nextPipe.top / height,
    (nextPipe.x - this.x) / width
  ]
  let results = this.brain.classifySync(inputs)
  if (results[0].label == "flap")
  {
    this.flap()
  }
}

flap()
{
  this.velocity += this.force
}
```

```
  move()
  {
    this.velocity += this.gravity
    this.y += this.velocity
    this.velocity *= 0.95
    if (this.y > height || this.y < 0)
    {
      this.alive = false
    }
    this.fitness++
  }

  show()
  {
    stroke(0)
    noFill()
    circle(this.x, this.y, 20)
  }
}
```

🗒️ Notes

If you leave it running for a while, you should see the results as a single bird (although technically not necessarily) working its way through the maze of pipes successfully.

🌻 Challenge

There are a lot of things you could do to improve or change this flappy bird neuroevolution programme.

1. Build a slider to speed it up, as we did with the smart cars.
2. Add an image of the flappy bird (see Games in the resources tab).

🛠️ Code Explanation

`if (this.y > height || this.y < 0)`	If the bird has gone below the bottom edge or gone above the top edge then. . .
`this.alive = false`	. . .the bird is dead