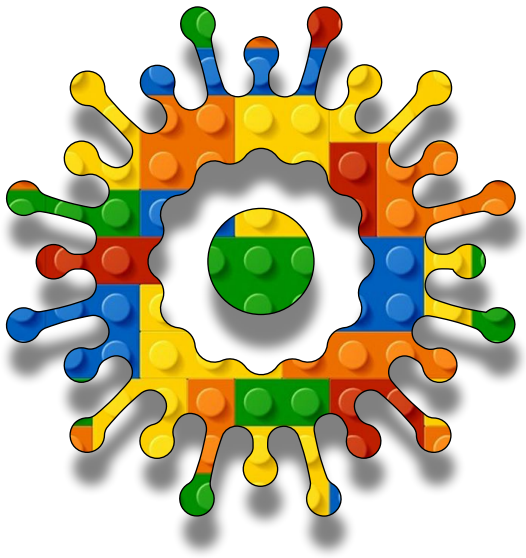
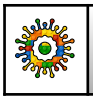
Artificial Intelligence Module C Unit #3 smart cars

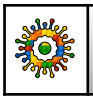




Content

Module C Unit #3 smart cars

Introduction to Unit #3 smart cars
The index.html file
Adding target.js and vehicle.js files
Adding them to the index.html file
Sketch C3.1 starting sketch
Sketch C3.2 target circle
Sketch C3.3 target class
Sketch C3.4 perlin noise
Sketch C3.5 moving target
Sketch C3.6 a vehicle
Sketch C3.7 random position
Sketch C3.8 moving towards the target
Sketch C3.9 the seek function
Sketch C3.10 seek and ye shall find
Sketch C3.11 more vehicles
A neuroevolution brain
Sketch C3.12 give the vehicle has a brain
Sketch C3.13 steering the vehicles
Sketch C3.14 the outputs
Sketch C3.15 they move
Sketch C3.16 fitness scores
Sketch C3.17 we ned a radius
Sketch C3.18 overlap
Weighted selection
Sketch C3.19 lifespan
Reproduction
Sketch C3.20 the next generation
Sketch C3.21 birth of a child
Sketch C3.22 lifeCounter
What next?
Sketch C3.23 adding a slider



Introduction to smart cars neuroevolution

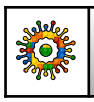
We are going to give these cars a smart brain (ml5.js). They have to learn to chase after a moving target. All they do is get a reward according to how well they do. This uses a Genetic Algorithm to select those who perform best. We aren't going to train the algorithm but select the ones that perform better, for every car will have its own neural network based on random weights.

Over time, we select the best ones (that get closest to the target) and breed them, think natural selection, evolution. After many populations have come and gone, we are left with ones that have succeeded in breeding the right attributes (weights) to follow a moving target.

We are going to create two classes, target and vehicle. For these, we put them in two new files and add them to the index.html file. This was covered in the coding snippets part 4, so you will be familiar with the process of adding files and the purposes of classes. However, I will still emphasise their purpose and function so that you become more familiar with them.

It goes without saying that you will also need to have the line of code for ml5.js in the index.html file. You will also note that we will be switching from one file to another and back again. This means that you might want to keep the side panel open as you code. This way, you can switch quickly from one to the other.

If you don't want the fuss of going from file to file, then you can add the classes in the sketch.js file and just have one long line of code.



The index.html file

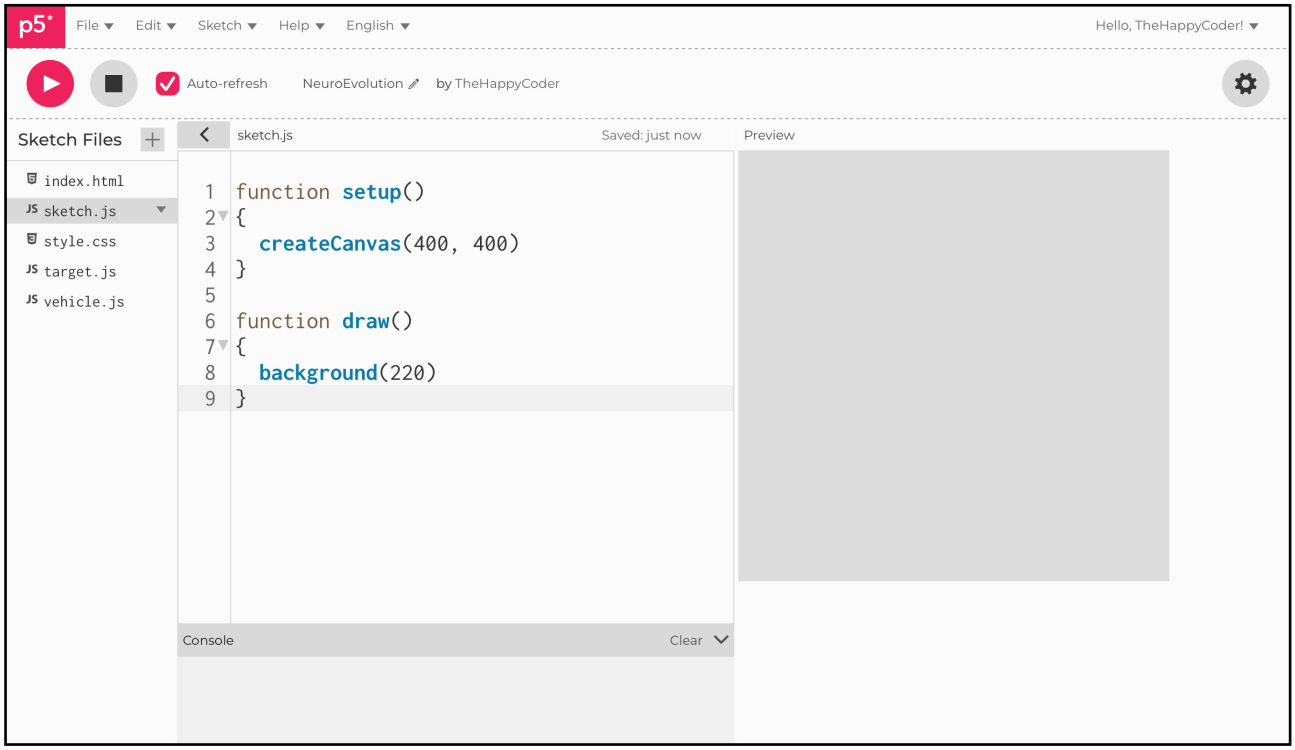
Adding ml5.js as per usual.

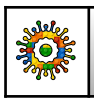
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.0/p5.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.0/addons/p5.sound.min.js"></script>
    <script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8" />

  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
  </body>
</html>
```

Adding target.js and vehicle.js files

Now we are going to add the next two files, target.js and vehicle.js, to our list of files. See figure 1.

Figure 1: adding the files

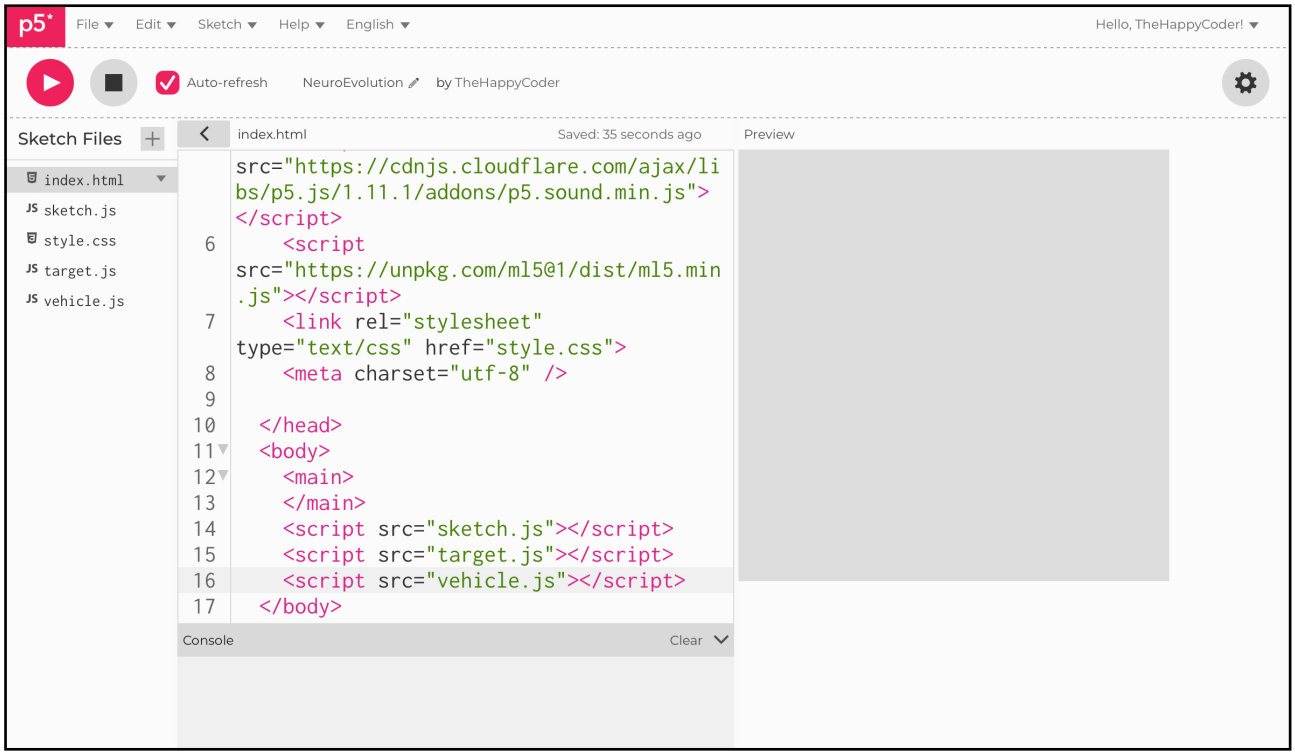


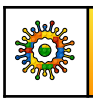


Adding them to the index.html file

This step is easy to forget. Just copy and paste sketch.js and change the names to `target` and `vehicle`. To access the files, just click on them, but these two new ones should be completely empty. See `figure 2`.

Figure 2: adding files in index.html





Sketch C3.1 starting sketch

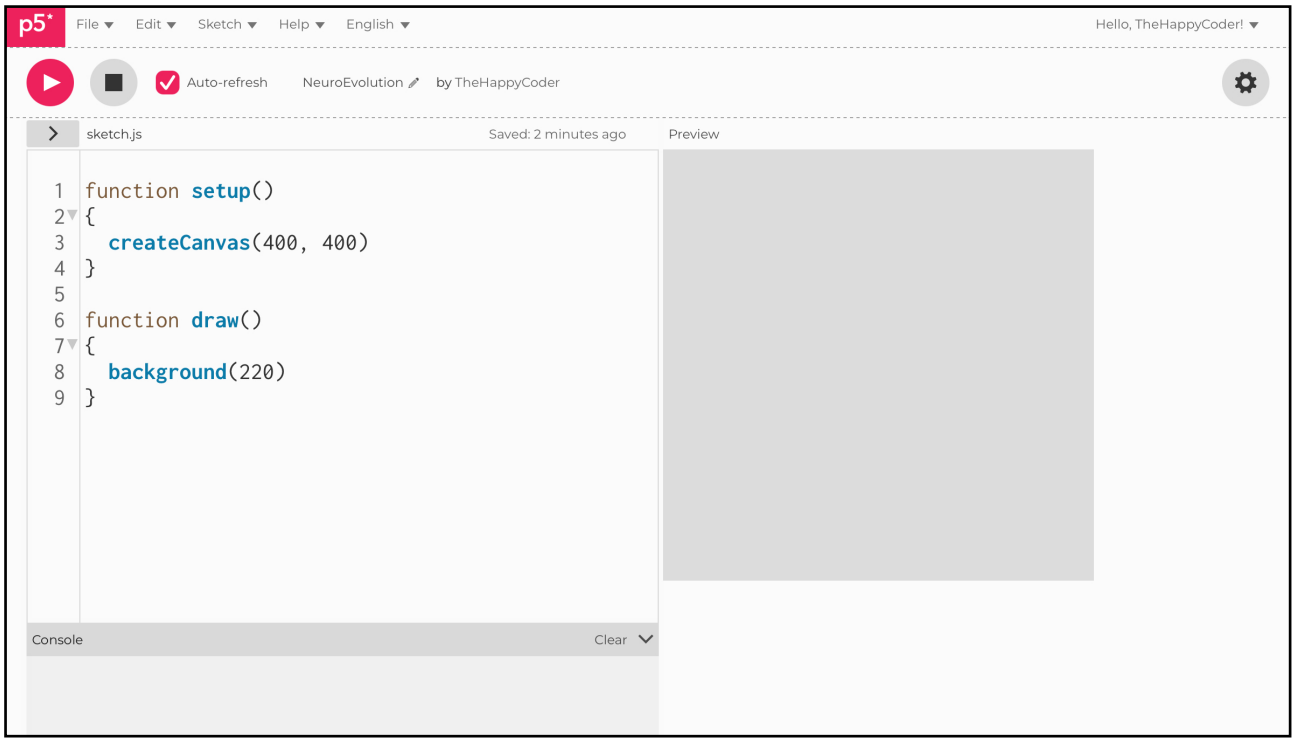
! Starting sketch in `sketch.js`

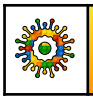
Our starting sketch, note that this is for the `sketch.js` file (see the heading).

sketch.js

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

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

Figure C3.1


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
NeuroEvolution
by TheHappyCoder
sketch.js
Saved: 2 minutes ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 }
Console
Clear



Sketch C3.2 target circle

We are going to create a `target` which will just be a circle. This is what the `vehicles` (cars) will chase around the canvas. The `target` becomes a vector object. From that vector, we pull out the `x` and `y` values, which we will initialise to `width/2` and `height/2`.

sketch.js

```
let target

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

function draw()
{
  background(220)
  target = createVector(width/2, height/2)
  circle(target.x, target.y, 30)
}
```

🗒️ Notes

You get a static `target` (circle).

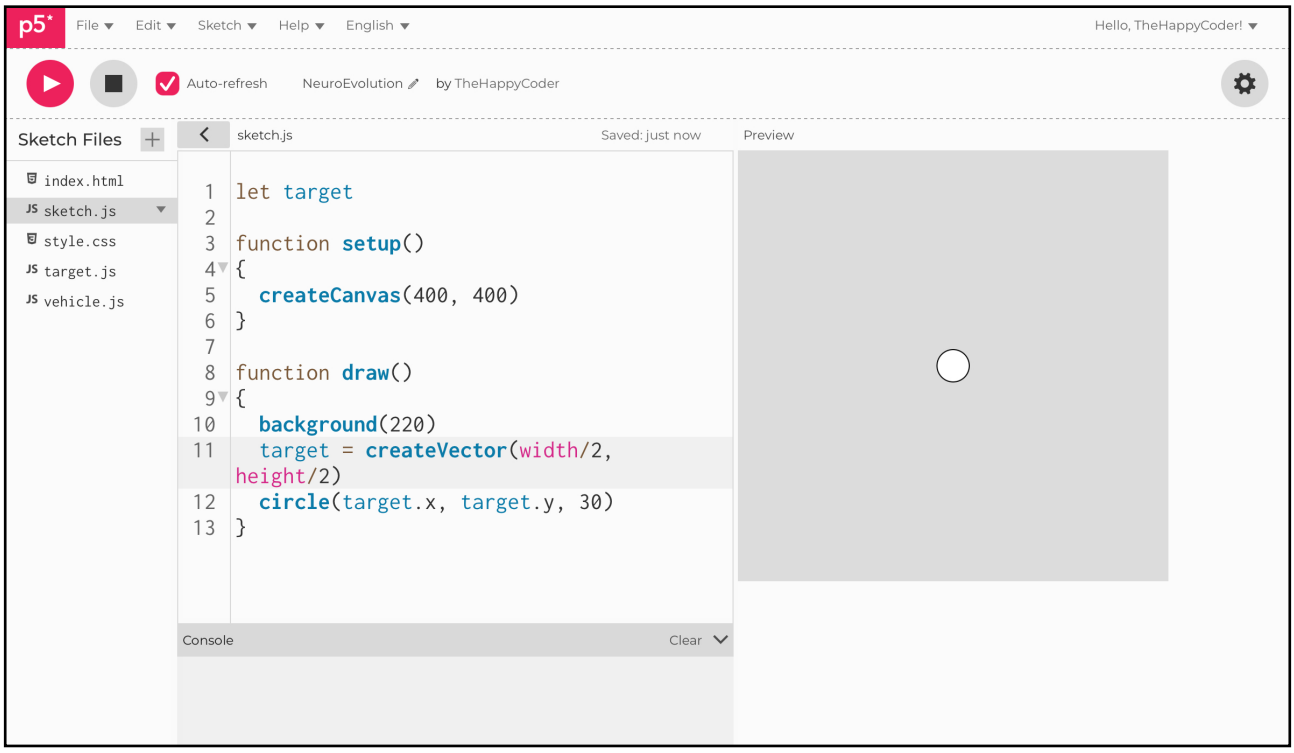
🌻 Challenge

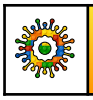
Add some colour or even change the shape.

🛠️ Code Explanation

`target = createVector(width/2, height/2)`	The target is a vector, its position is in the centre of the canvas
`circle(target.x, target.y, 30)`	The circle is drawn to the coordinates of the target (target.x and target.y)

Figure C3.2


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
NeuroEvolution
by TheHappyCoder
Sketch Files
index.html
sketch.js
style.css
target.js
vehicle.js
sketch.js
Saved: just now
Preview
1 let target
2
3 function setup()
4 {
5 createCanvas(400, 400)
6 }
7
8 function draw()
9 {
10 background(220)
11 target = createVector(width/2,
height/2)
12 circle(target.x, target.y, 30)
13 }
Console
Clear



Sketch C3.3 target class

❗ Now we move over to the `target.js` file.

We don't want a stationary `target`; we want to give it movement, and in this instance, a random motion called Perlin `noise`. For this, we need to create a class called `Target`.

target.js

```
class Target
{
  constructor()
  {
    this.xoff = 0
    this.yoff = 1000
    this.position = createVector()
  }
}
```

Notes

If you haven't completed the unit `coding snippets 4`, then here is a brief explanation of `Perlin Noise`. Think of it as a continuous random number where the next random value is dependent on the previous one. What this means is that there isn't a wild discrepancy from one value to the next. This creates the illusion of a smooth, gentle wandering of the circle.

Imagine a wavy line and the starting point on this wavy line is at `0` (`xoff`), which will return a value between `0` and `+1`. If you move along the line to position `1000` (`yoff`), you get another value between `0` and `+1`. We then inch along this line and we get a random value just a bit more or a bit less than the previous value. We do this for the `x offset` and the `y offset`; otherwise, we would have the same value for both if they both started at `0` or `1000`, for example.

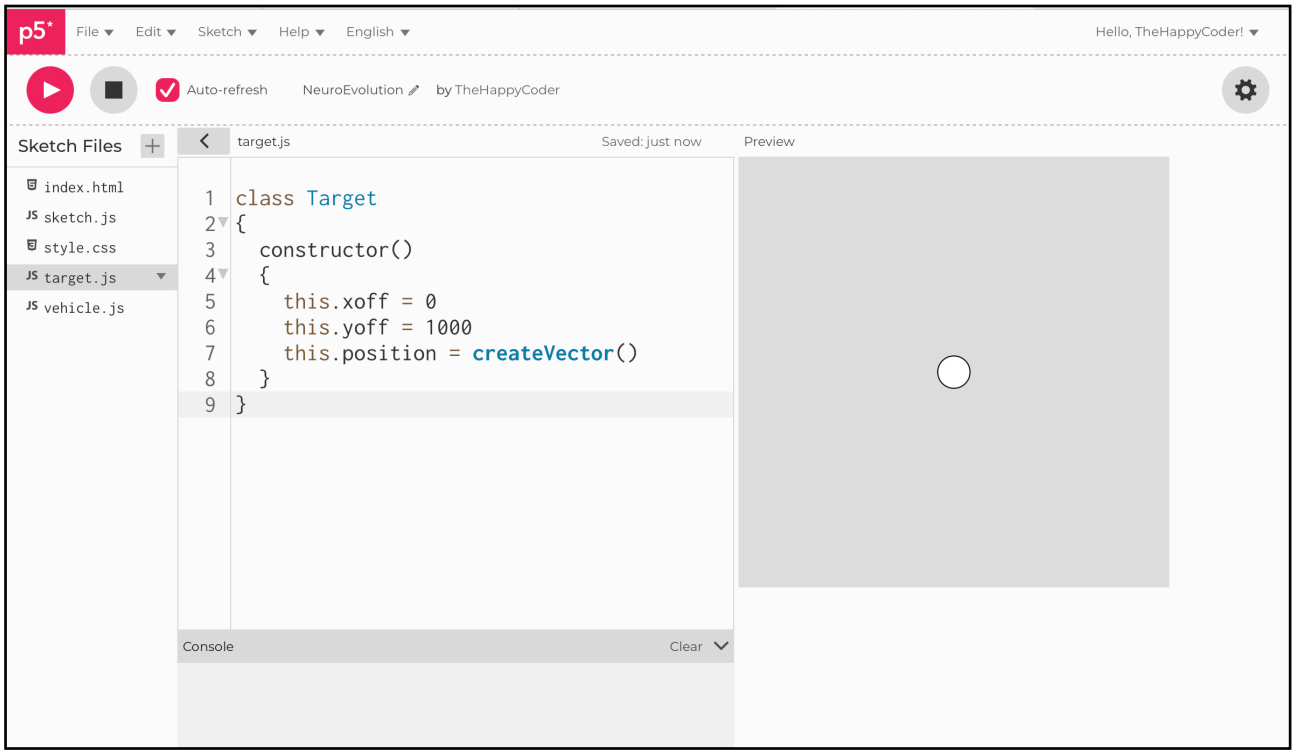
🌻 Challenges

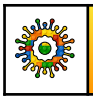
1. Have a different starting position on the Perlin `noise` timeline for the `x` and `y` components.
2. What do you think would happen if they were the same?

🛠️ Code Explanation

`this.xoff = 0`	The initial x value on the perlin noise time line
`this.yoff = 1000`	The initial y value on the perlin noise time line
`this.position = createVector()`	Create a vector for the position of the target

Figure C3.3


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
NeuroEvolution
by TheHappyCoder
Sketch Files
index.html
sketch.js
style.css
target.js
vehicle.js
target.js
Saved: just now
Preview
1 class Target
2 {
3 constructor()
4 {
5 this.xoff = 0
6 this.yoff = 1000
7 this.position = createVector()
8 }
9 }
Console
Clear



Sketch C3.4 perlin noise

We are going to use this randomness to move the `target` (circle) around the canvas. So we need to connect the position of the target to the `noise` (Perlin) value and then increment along that random wavy line by `0.01` for `x` and `y`. We add our two favourite functions, `show()` and `move()`.

target.js

```
class Target
{
  constructor()
  {
    this.xoff = 0
    this.yoff = 1000
    this.position = createVector()
  }

  move()
  {
    this.position.x = noise(this.xoff) * width
    this.position.y = noise(this.yoff) * height
    this.xoff += 0.01
    this.yoff += 0.01
  }

  show()
  {
    circle(this.position.x, this.position.y, 30)
  }
}
```

🗒️ Notes

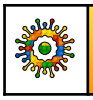
We multiply by `width` and `height` because the `noise()` function returns values between `0` and `1`, and we want the `target` to move between `0` and the `width` and `height` of the canvas.

🌻 Challenge

How would you use the `map()` function?

🛠️ Code Explanation

`this.position.x = noise(this.xoff) * width`	We get our target x position from the perlin noise() function (times the width)
`this.position.y = noise(this.yoff) * height`	We get our target y position from the perlin noise() function (times the height)
`this.xoff += 0.01`	Increment along the noise time line for x
`this.yoff += 0.01`	Increment along the noise time line for y



Sketch C3.5 moving target

❗ Back to `sketch.js`. Comment out and remove the two lines of code for the circle.
We are going to use the `Target` class to create a target object and then run the `update()` and `show()` functions from that object. Now you have a `target` (circle) wandering around the canvas.

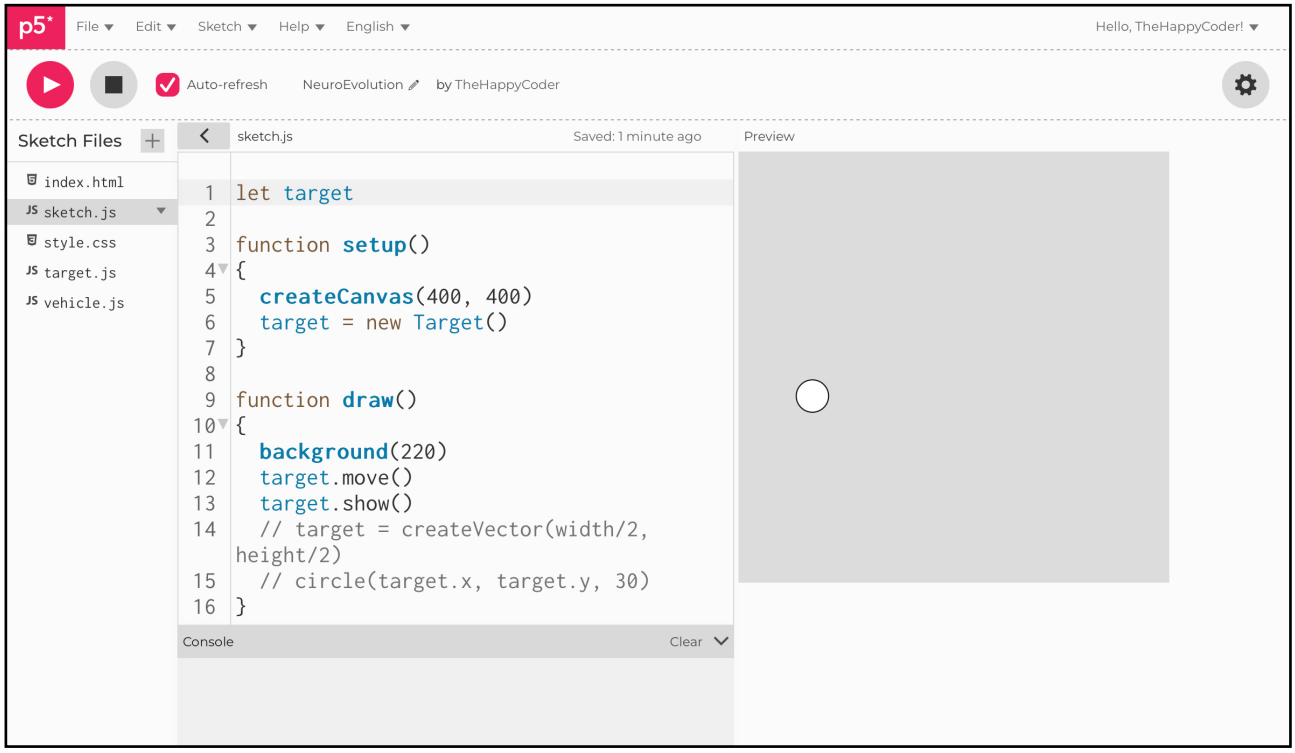
sketch.js

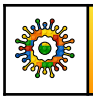
```
let target


function setup()
{
  createCanvas(400, 400)
  target = new Target()
}


function draw()
{
  background(220)
  target.move()
  target.show()
  // target = createVector(width/2, height/2)
  // circle(target.x, target.y, 30)
}
```

Notes

The `target` should be moving around the canvas in a gentle swooping way.

Figure C3.5


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
NeuroEvolution
by TheHappyCoder
Sketch Files
index.html
sketch.js
style.css
target.js
vehicle.js
sketch.js
Saved: 1 minute ago
Preview
let target
function setup()
{
createCanvas(400, 400)
target = new Target()
}
function draw()
{
background(220)
target.move()
target.show()
// target = createVector(width/2, height/2)
// circle(target.x, target.y, 30)
}
Console
Clear



Sketch C3.6 a vehicle

❗ Hop over to vehicle.js

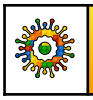
Now we need a vehicle. We will create a Vehicle class and in its constructor function we will have position, velocity, and acceleration. This should be fairly familiar to you by now.

vehicle.js

```
class Vehicle
{
  constructor(x, y)
  {
    this.position = createVector(x, y)
    this.velocity = createVector(0, 0)
    this.acceleration = createVector(0, 0)
  }

  show()
  {
    push()
    translate(this.position.x, this.position.y)
    rect(0, 0, 10, 5)
    pop()
  }
}
```

Notes

When we create a vehicle, the constructor() function will receive two arguments, the (x, y) coordinates. Nothing will appear just yet as we haven't actually created a vehicle. We previously called them pos, vel, and acc, just nice to be different.



Sketch C3.7 random position

❗ Back to sketch.js

Let's create one vehicle for now at some random position.

sketch.js

```
let target
let vehicle

function setup()
{
  createCanvas(400, 400)
  target = new Target()
  vehicle = new Vehicle(random(width), random(height))
}

function draw()
{
  background(220)
  target.move()
  target.show()
  vehicle.show()
}
```

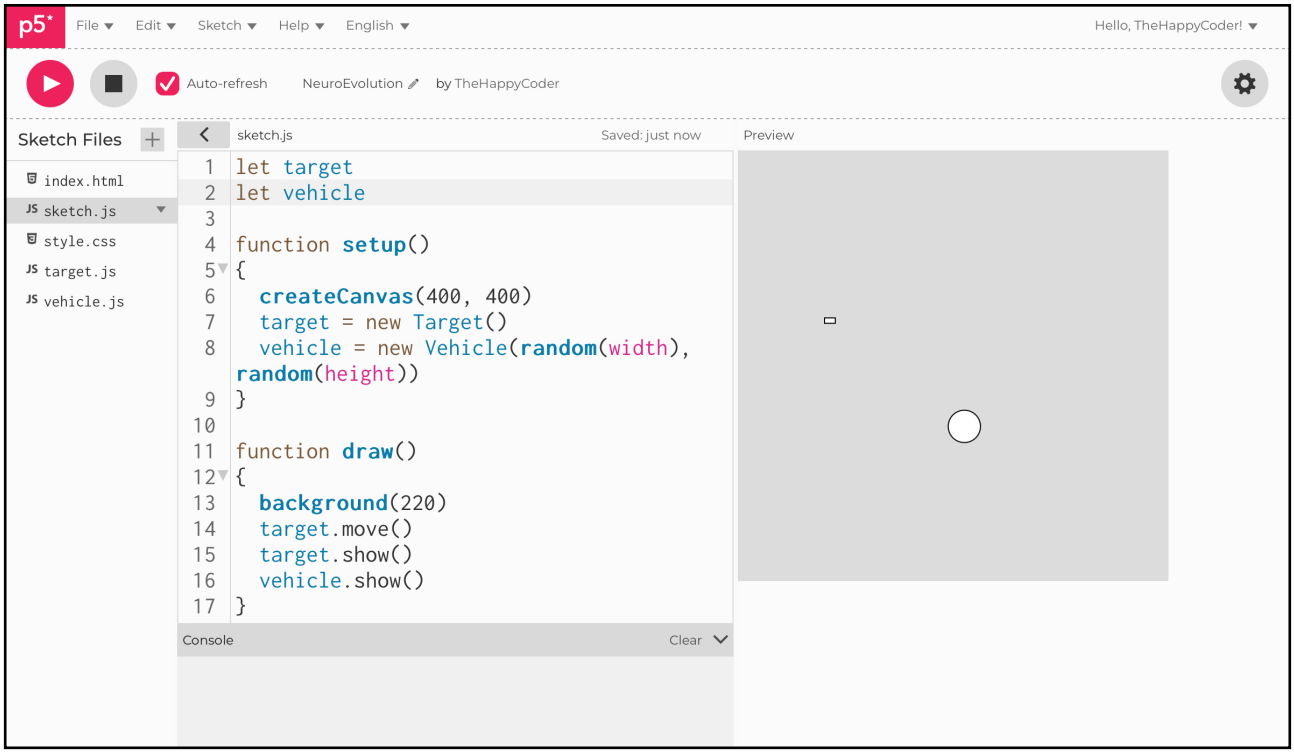
🗒️ Notes

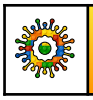
Every time you run the sketch, you will get another random vehicle somewhere on the canvas.

🌻 Challenge

Give the vehicle some colour.

Figure C3.7


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
NeuroEvolution
by TheHappyCoder
Sketch Files
index.html
sketch.js
style.css
target.js
vehicle.js
sketch.js
Saved: just now
Preview
let target
let vehicle
function setup()
{
createCanvas(400, 400)
target = new Target()
vehicle = new Vehicle(random(width),
random(height))
}
function draw()
{
background(220)
target.move()
target.show()
vehicle.show()
}
Console
Clear



Sketch C3.8 moving towards the target

❗ Moving back to `vehicle.js`

We will now get the `vehicle` moving using the usual formula of adding the components together for movement.

vehicle.js

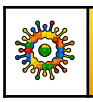
```
class Vehicle
{
  constructor(x, y)
  {
    this.position = createVector(x, y)
    this.velocity = createVector(0, 0)
    this.acceleration = createVector(0, 0)
    this.maxSpeed = 4
  }

  move()
  {
    this.velocity.add(this.acceleration)
    this.velocity.limit(this.maxSpeed)
    this.position.add(this.velocity)
    this.acceleration.mult(0)
  }

  show()
  {
    push()
    translate(this.position.x, this.position.y)
    rect(0, 0, 10, 5)
    pop()
  }
}
```

🗒️ Notes

We won't see anything happen yet (except the `target` wandering) because we haven't the `move()` function in `sketch.js`.

🛠️ Code Explanation

`this.acceleration.mult(0)`	The acceleration is zeroed so that the vehicle doesn't run away with itself



Sketch C3.9 the seek function

! Because we have covered most of this in the coding snippets 4 unit with the seek() function, I won't labour this part.
We will want to get it to move towards the target and also to rotate. This was covered briefly in the coding snippets 4 unit using the heading() function. Also, we do need to have a maximum force. The vehicle will be steered towards the target by a force of some magnitude; we turn the vehicle in that direction. This is where we add the applyForce() function and the seek() function.

vehicle.js

```
class Vehicle
{
  constructor(x, y)
  {
    this.position = createVector(x, y)
    this.velocity = createVector(0, 0)
    this.acceleration = createVector(0, 0)
    this.maxSpeed = 4
    this.maxForce = 0.2
  }

  move()
  {
    this.velocity.add(this.acceleration)
    this.velocity.limit(this.maxSpeed)
    this.position.add(this.velocity)
    this.acceleration.mult(0)
  }

  applyForce(force)
  {
    this.acceleration.add(force)
```

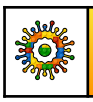
```
  }

  seek(target)
  {
    let desired = p5.Vector.sub(target, this.position)
    desired.setMag(this.maxSpeed)
    let steer = p5.Vector.sub(desired, this.velocity)
    steer.limit(this.maxForce)
    this.applyForce(steer)
  }

  show()
  {
    let angle = this.velocity.heading()
    push()
    translate(this.position.x, this.position.y)
    rotate(angle)
    rect(0, 0, 10, 5)
    pop()
  }
}
```

🗒️ Notes

In the show function, we find the `angle` from the `heading()` function and then rotate the vehicle according to that `angle`. The `seek()` and `applyForce()` functions are the same as we explored in `coding snippets 4`.

🛠️ Code Explanation

`let angle = this.velocity.heading()`	We get the angle from the direction of the velocity vector
`rotate(angle)`	We rotate the vehicle by that angle



Sketch C3.10 seek and ye shall find

❗ Return to `sketch.js`

We create a vector for the `position` of the `target` and then seek that vector.

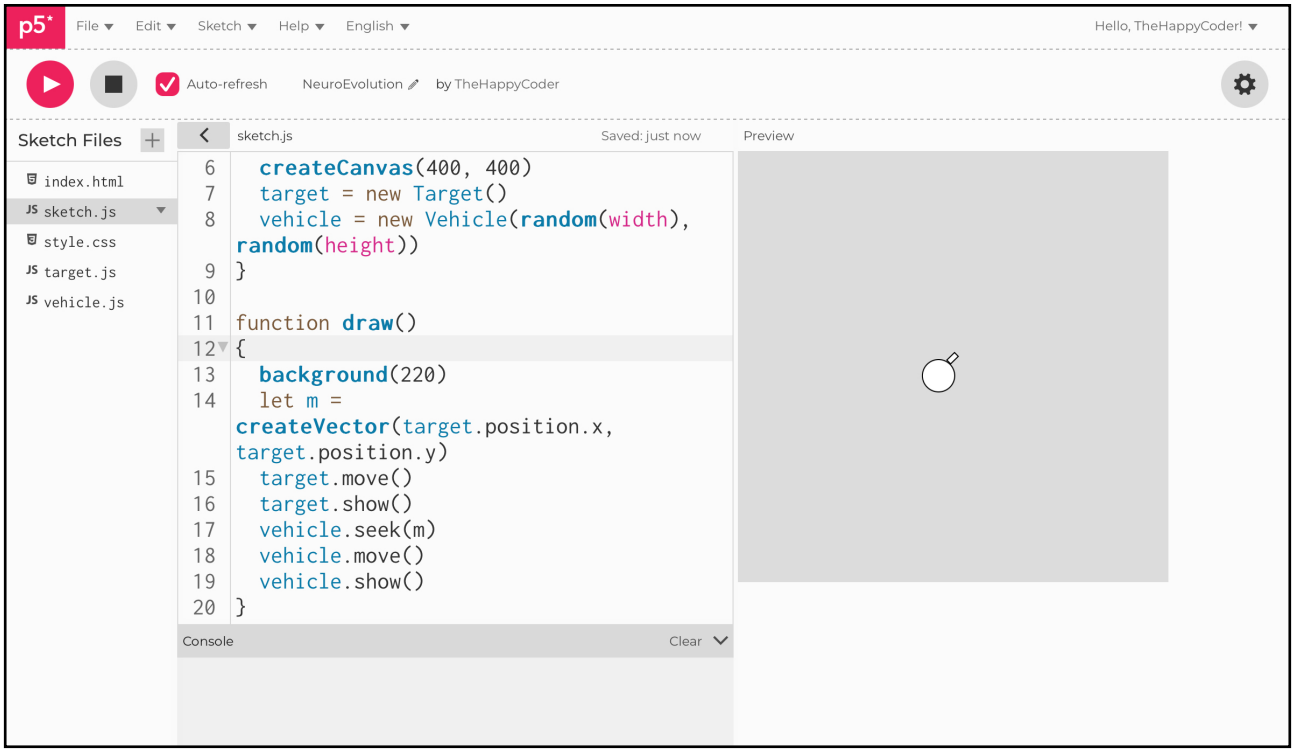
sketch.js

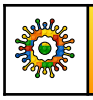
```
let target
let vehicle

function setup()
{
  createCanvas(400, 400)
  target = new Target()
  vehicle = new Vehicle(random(width), random(height))
}

function draw()
{
  background(220)
  let m = createVector(target.position.x, target.position.y)
  target.move()
  target.show()
  vehicle.seek(m)
  vehicle.move()
  vehicle.show()
}
```

Notes

The `vehicle` now follows the `target`, job done! It is quite mesmerising to watch.

Figure C3.10


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
NeuroEvolution
by TheHappyCoder
Sketch Files
index.html
sketch.js
style.css
target.js
vehicle.js
sketch.js
Saved: just now
Preview
6 createCanvas(400, 400)
7 target = new Target()
8 vehicle = new Vehicle(random(width), random(height))
9 }
10
11 function draw()
12 {
13 background(220)
14 let m = createVector(target.position.x, target.position.y)
15 target.move()
16 target.show()
17 vehicle.seek(m)
18 vehicle.move()
19 vehicle.show()
20 }
Console
Clear



Sketch C3.11 more vehicles

We are now going to create 50 of these vehicles. We will replace a number of lines of code to move from the single vehicle to an array of vehicles.

sketch.js

```
let target
let vehicles = []

function setup()
{
  createCanvas(400, 400)
  target = new Target()
  for (let i = 0; i < 50; i++)
  {
    vehicles[i] = new Vehicle(random(width), random(height))
  }
}

function draw()
{
  background(220)
  let m = createVector(target.position.x, target.position.y)
  target.move()
  target.show()
  for (let vehicle of vehicles)
  {
    vehicle.seek(m)
    vehicle.move()
    vehicle.show()
  }
}
```

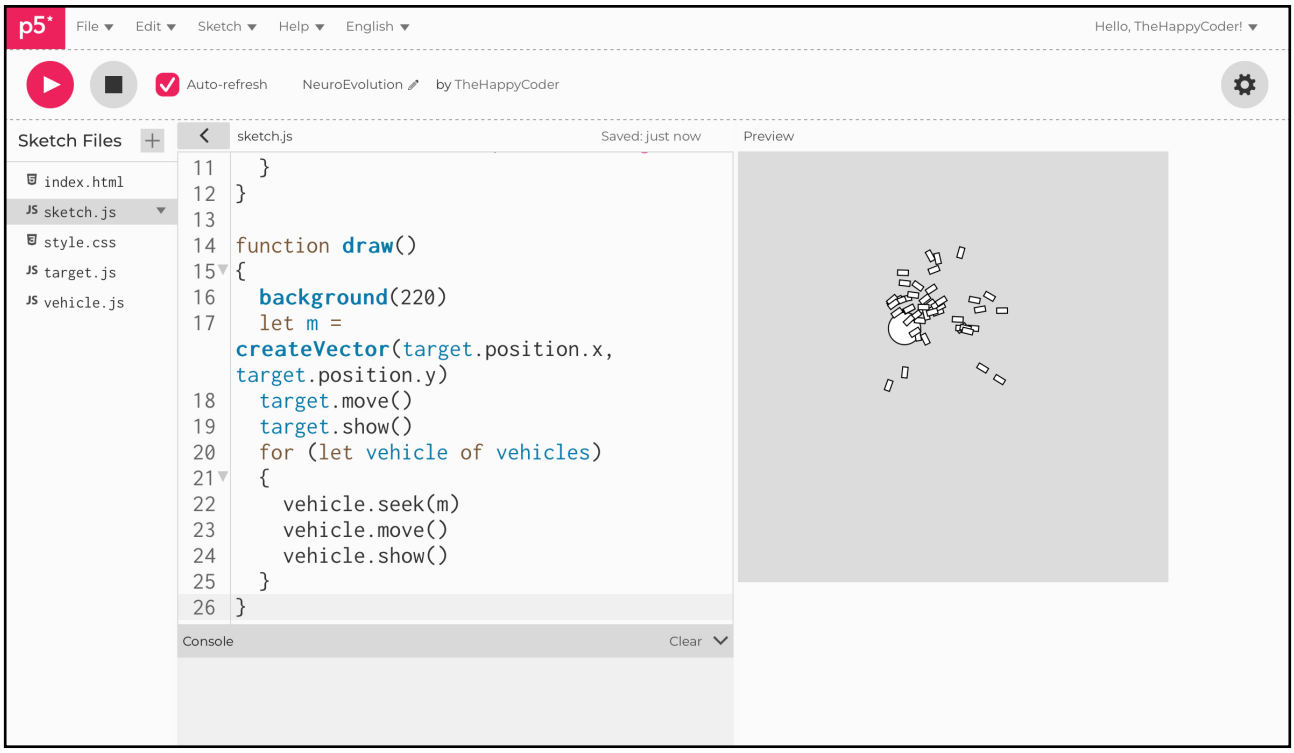
🗒️ Notes

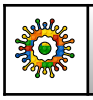
You should have a whole bunch of `vehicles` following the target, as shown in `figure C3.11`. Each `vehicle` will start from a different random position but eventually coalesce into one vehicle.

🌻 Challenge

Try different numbers of vehicles

Figure C3.11


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
NeuroEvolution
by TheHappyCoder
Sketch Files
index.html
sketch.js
style.css
target.js
vehicle.js
sketch.js
Saved: just now
Preview
11 }
12 }
13
14 function draw()
15 {
16 background(220)
17 let m =
createVector(target.position.x,
target.position.y)
18 target.move()
19 target.show()
20 for (let vehicle of vehicles)
21 {
22 vehicle.seek(m)
23 vehicle.move()
24 vehicle.show()
25 }
26 }
Console
Clear



A neuroevolution brain

Although much of the `neural network` stuff we have already covered is used here in this module and unit, there is a subtle difference. We can see it in the lines of code in the next sketch as we add the `neural network` brain to the `vehicles`. First of all, we give the brain the inputs and outputs:

The five inputs are:

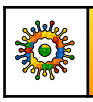
1. The x component of the vector between the vehicle and the target
2. The y component of the vector between the vehicle and the target
3. The distance between the target and the vehicle
4. The x component of the velocity of the vehicle
5. The y component of the velocity of the vehicle

The outputs are:

1. The angle the vehicle must turn to move towards the target
2. The magnitude of the power to move it to the target

Then we are going to give it the task: `regression`. However, here is the big difference: we will inform the neural network that this is a neuroevolution network with the `neuroEvolution: true` line of code. Then, finally, we inform the neural network that we are not doing any training with the `noTraining: true` line of code.

Remember that this is a `Reinforcement Learning` challenge; there is no data for the neural network to learn from. We are simply going to select the best brains (neural networks) that appear to solve the challenge without hardcoding the vehicles with the `seek()` function, which really is the simple solution to this challenge.



Sketch C3.12 giving the vehicle a brain

! Bob over to vehicle.js

In the Vehicle class, we give the vehicle a brain. The brain is the neural network which has five inputs and two outputs. There will be 50 brains driving their own vehicles.

vehicle.js

```
class Vehicle
{
  constructor(x, y)
  {
    this.position = createVector(x, y)
    this.velocity = createVector(0, 0)
    this.acceleration = createVector(0, 0)
    this.maxSpeed = 4
    this.maxForce = 0.2
    ml5.setBackend("cpu")
    this.brain = ml5.neuralNetwork({
      inputs: 5,
      outputs: 2,
      task: "regression",
      neuroEvolution: true,
      noTraining: true
    })
  }

  move()
  {
    this.velocity.add(this.acceleration)
    this.velocity.limit(this.maxSpeed)
    this.position.add(this.velocity)
    this.acceleration.mult(0)
```

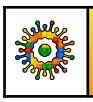
```
  }

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

  seek(target)
  {
    let desired = p5.Vector.sub(target, this.position)
    desired.setMag(this.maxSpeed)
    let steer = p5.Vector.sub(desired, this.velocity)
    steer.limit(this.maxForce)
    this.applyForce(steer)
  }

  show()
  {
    let angle = this.velocity.heading()
    push()
    translate(this.position.x, this.position.y)
    rotate(angle)
    rect(0, 0, 10, 5)
    pop()
  }
}
```

🗒️ Notes

Notice that this is a regression task; it is also not a normal neural network. This means we can access the `mutate()` and `crossover()` functions later as part of the neuroevolution network we will implement later.

🛠️ Code Explanation

`neuroEvolution: true,`	This is a neuroevolution challenge
`noTraining: true`	We are not doing any training



Sketch C3.13 steering the vehicles

From the inputs, we need to know how much to steer the `vehicles`. For that, we need to know their `velocity`, their `position` relative to the target, and the desired velocity. This is all about magnitude and direction at the end of the day. We do need to do a bit of normalising so that things don't get out of control, hence dividing things by the `width` or the `maxSpeed`.

We are going to rewrite the entire `seek(target)` function in the `Vehicle` class. Nothing will happen (except an error message) because we have done nothing about the outputs, which will determine the force to move the vehicle.

There is a lot to unpack in the `seek()` function. These are our five inputs for the neural network, hence the refactoring.

vehicle.js

```
class Vehicle
{
  constructor(x, y)
  {
    this.position = createVector(x, y)
    this.velocity = createVector(0, 0)
    this.acceleration = createVector(0, 0)
    this.maxSpeed = 4
    this.maxForce = 0.2
    ml5.setBackend("cpu")
    this.brain = ml5.neuralNetwork({
      inputs: 5,
      outputs: 2,
      task: "regression",
      neuroEvolution: true,
      noTraining: true
    })
```

```
  }

  move()
  {
    this.velocity.add(this.acceleration)
    this.velocity.limit(this.maxSpeed)
    this.position.add(this.velocity)
    this.acceleration.mult(0)
  }

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

  seek(target)
  {
    let v = p5.Vector.sub(target.position, this.position)
    let distance = v.mag() / width
    v.normalize()
    let inputs = [
      v.x,
      v.y,
      distance,
      this.velocity.x / this.maxSpeed,
      this.velocity.y / this.maxSpeed
    ]
  }

  show()
  {
    let angle = this.velocity.heading()
    push()
```

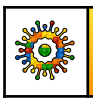
```
    translate(this.position.x, this.position.y)
    rotate(angle)
    rect(0, 0, 10, 5)
    pop()
  }
}
```

🗒️ Notes

You will get an error if you try to run it now, so don't try. The variable `v` is a vector of the difference between any one vehicle's position and the target position. We then get the `distance` from the magnitude of that difference (`mag()`). We then normalise the vector `v` before creating the five inputs from all that data.

🛠️ Code Explanation

`v = p5.Vector.sub(target.position, this.position)`	Subtracting two vectors to get a third vector v
`distance = v.mag() / width`	Distance is the magnitude of the vector v, reducing the magnitude by the width
`v.normalize()`	Normalising v
`v.x`	The x component of the vector v
`v.y`	The y component of the vector v
`this.velocity.x / this.maxSpeed`	Reducing the x component of the velocity by the maxSpeed
`this.velocity.y / this.maxSpeed`	Reducing the y component of the velocity by the maxSpeed



Sketch C3.14 the outputs

❗ Back to `sketch.js`

Now for the outputs. We have two outputs, the angle and the magnitude. The predicted values are random. They take in the values from the inputs, then generate outputs that are meaningless. Also, you will notice that in the `brain` there is an argument for `noTraining`. This is because we are not using any data to train the neural network; we are just choosing to select the best performing ones. So when you run this, they just disappear off into the distance.

❗ We will remove:

```
let m = createVector(target.position.x, target.position.y)
```

and replace with:

```
vehicle.seek(target)
```

sketch.js

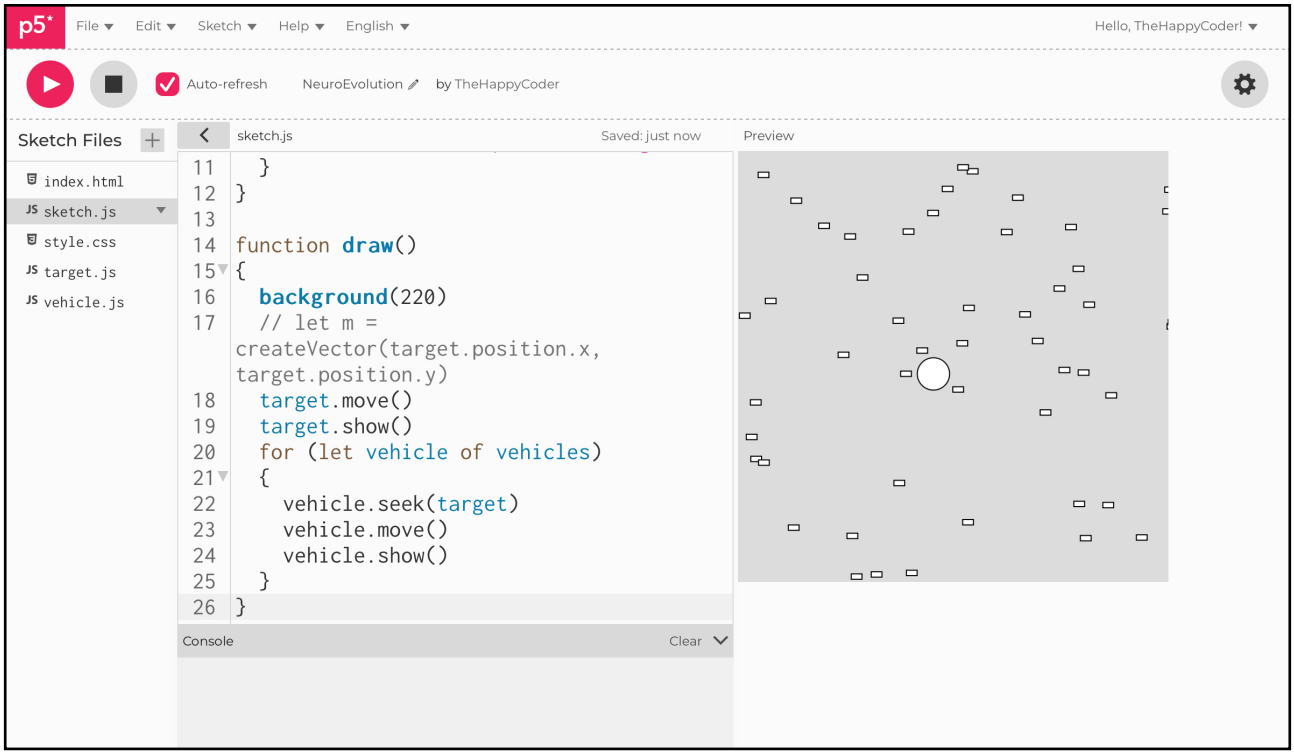
```
let target
let vehicles = []

function setup()
{
  createCanvas(400, 400)
  target = new Target()
  for (let i = 0; i < 50; i++)
  {
    vehicles[i] = new Vehicle(random(width), random(height))
  }
}

function draw()
{
  background(220)
  // let m = createVector(target.position.x, target.position.y)
  target.move()
  target.show()
```

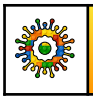
```
  for (let vehicle of vehicles)
  {
    vehicle.seek(target)
    vehicle.move()
    vehicle.show()
  }
}
```

Notes

You will get static vehicles.

Figure C3.14


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
NeuroEvolution
by TheHappyCoder
Sketch Files
index.html
sketch.js
style.css
target.js
vehicle.js
sketch.js
Saved: just now
Preview
11 }
12 }
13
14 function draw()
15 {
16 background(220)
17 // let m =
createVector(target.position.x,
target.position.y)
18 target.move()
19 target.show()
20 for (let vehicle of vehicles)
21 {
22 vehicle.seek(target)
23 vehicle.move()
24 vehicle.show()
25 }
26 }
Console
Clear



Sketch C3.15 they move

❗ Skip on over to vehicle.js

Now let's get them moving.

vehicle.js

```
class Vehicle
{
  constructor(x, y)
  {
    this.position = createVector(x, y)
    this.velocity = createVector(0, 0)
    this.acceleration = createVector(0, 0)
    this.maxSpeed = 4
    this.maxForce = 0.2
    ml5.setBackend("cpu")
    this.brain = ml5.neuralNetwork({
      inputs: 5,
      outputs: 2,
      task: "regression",
      neuroEvolution: true,
      noTraining: true
    })
  }

  move(target)
  {
    this.velocity.add(this.acceleration)
    this.velocity.limit(this.maxSpeed)
    this.position.add(this.velocity)
    this.acceleration.mult(0)
  }
```

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

  seek(target)
  {
    let v = p5.Vector.sub(target.position, this.position)
    let distance = v.mag() / width
    v.normalize()
    let inputs = [
      v.x,
      v.y,
      distance,
      this.velocity.x / this.maxSpeed,
      this.velocity.y / this.maxSpeed
    ]
    let outputs = this.brain.predictSync(inputs)
    let angle = outputs[0].value * TWO_PI
    let magnitude = outputs[1].value
    let force = p5.Vector.fromAngle(angle)
    force.setMag(magnitude)
    this.applyForce(force)
  }

  show()
  {
    let angle = this.velocity.heading()
    push()
    translate(this.position.x, this.position.y)
    rotate(angle)
    rect(0, 0, 10, 5)
    pop()
```

```
  }
}
```

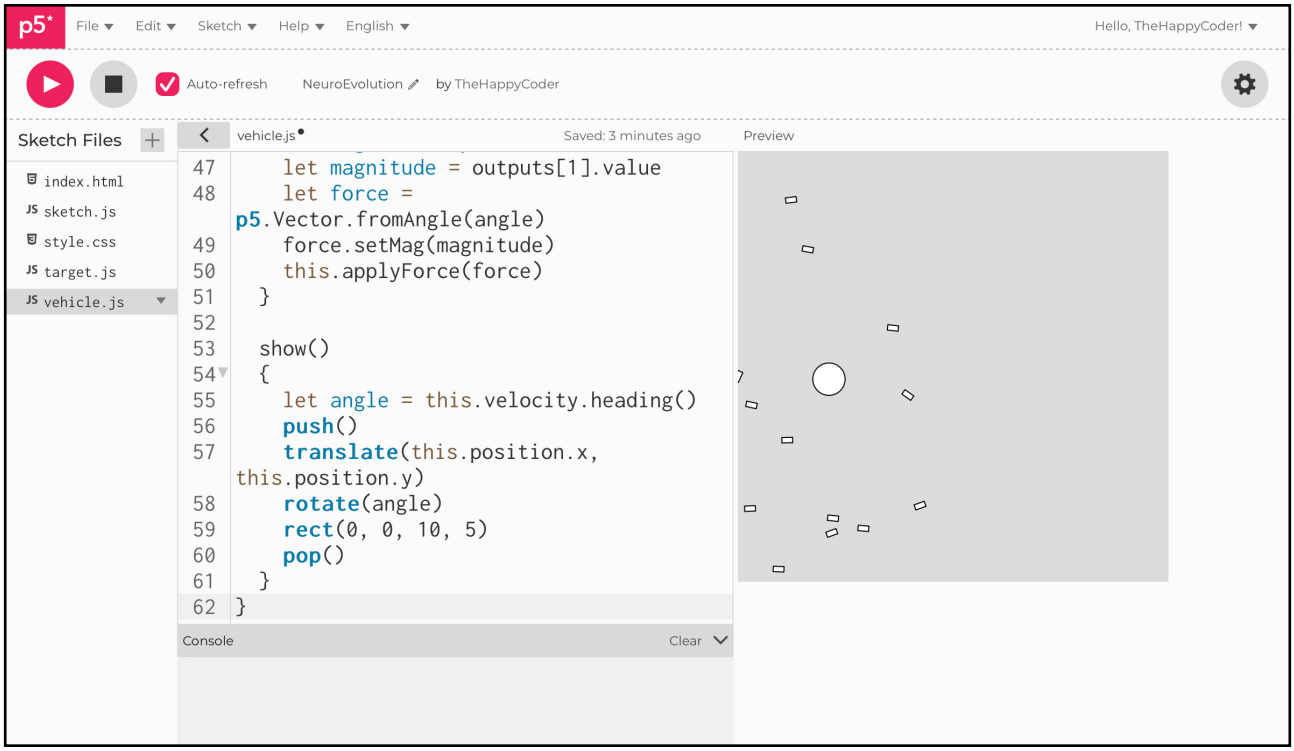
🗒️ Notes

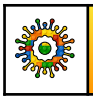
They move, but they just wander off into the sunset! This is because they have no reason to do otherwise.

🛠️ Code Explanation

`outputs = this.brain.predictSync(inputs)`	predictSync() means it waits for the inputs before predicting
`angle = outputs[0].value * TWO_PI`	The outputs index [0] is the angle which we multiply by 2π (to scale the output value
`magnitude = outputs[1].value`	The outputs index [1] is the magnitude
`force = p5.Vector.fromAngle(angle)`	The function fromAngle() creates a vector from an angle in radians

Figure C3.15


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
NeuroEvolution
by TheHappyCoder
Sketch Files
index.html
sketch.js
style.css
target.js
vehicle.js
vehicle.js
Saved: 3 minutes ago
Preview
47 let magnitude = outputs[1].value
48 let force = p5.Vector.fromAngle(angle)
49 force.setMag(magnitude)
50 this.applyForce(force)
51 }
52
53 show()
54 {
55 let angle = this.velocity.heading()
56 push()
57 translate(this.position.x, this.position.y)
58 rotate(angle)
59 rect(0, 0, 10, 5)
60 pop()
61 }
62 }
Console
Clear



Sketch C3.16 fitness scores

For this to evolve, we need to attribute a fitness score to each vehicle. We need to know a number of things; one of the main indicators is how close the vehicle gets to the target, so we move an imaginary circle, which will help us know when the target and the vehicle overlap. We call the vehicle radius `l`, and for the target, we will call this `r`.

vehicle.js

```
class Vehicle
{
  constructor(x, y)
  {
    this.position = createVector(x, y)
    this.velocity = createVector(0, 0)
    this.acceleration = createVector(0, 0)
    this.l = 5
    this.fitness = 0
    this.maxSpeed = 4
    this.maxForce = 0.2
    ml5.setBackend("cpu")
    this.brain = ml5.neuralNetwork({
      inputs: 5,
      outputs: 2,
      task: "regression",
      neuroEvolution: true,
      noTraining: true
    })
  }

  move()
  {
    this.velocity.add(this.acceleration)
```

```
    this.velocity.limit(this.maxSpeed)
    this.position.add(this.velocity)
    this.acceleration.mult(0)
    let d = p5.Vector.dist(this.position, target.position)
    if (d < this.l + target.r)
    {
      this.fitness++
    }
  }

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

  seek(target)
  {
    let v = p5.Vector.sub(target.position, this.position)
    let distance = v.mag() / width
    v.normalize()
    let inputs = [
      v.x,
      v.y,
      distance,
      this.velocity.x / this.maxSpeed,
      this.velocity.y / this.maxSpeed
    ]
    let outputs = this.brain.predictSync(inputs)
    let angle = outputs[0].value * TWO_PI
    let magnitude = outputs[1].value
    let force = p5.Vector.fromAngle(angle)
    force.setMag(magnitude)
    this.applyForce(force)
```

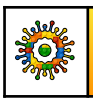
```
  }

  show()
  {
    let angle = this.velocity.heading()
    push()
    translate(this.position.x, this.position.y)
    rotate(angle)
    rect(0, 0, this.l*2, this.l)
    pop()
  }
}
```

🗒️ Notes

We haven't actually defined `r` yet (see next sketch).

🛠️ Code Explanation

`d = p5.Vector.dist(this.position, target.position)`	We calculate the distance between the position of each vehicle and the target
`if (d < this.l + target.r)`	If the distance is less than the l value (5) and the r value (15 in next sketch) combined then…
`this.fitness++`	Increase the fitness score of that particular vehicle



Sketch C3.17 we ned a radius

! Mosey on over to target.js

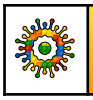
We need a variable for the radius r of the circle (target) to measure the fitness.

target.js

```
class Target
{
  constructor()
  {
    this.xoff = 0
    this.yoff = 1000
    this.position = createVector()
    this.r = 15
  }

  move()
  {
    this.position.x = noise(this.xoff) * width
    this.position.y = noise(this.yoff) * height
    this.xoff += 0.01
    this.yoff += 0.01
  }

  show()
  {
    circle(this.position.x, this.position.y, this.r * 2)
  }
}
```

🗒️ Notes

We now have a variable so we can compare the distance between the vehicle and the target.



Sketch C3.18 overlap

! Back we go to sketch.js

So we have added a fitness to each vehicle, and those that pass within a distance of r + l (they overlap) their fitness will increase. This is all prep for later on as we add in more features. Yet we do need to normalise the fitness of the vehicle; otherwise, you will get huge differences. We do this very simply by adding up all the fitnesses (sum) and dividing an individual vehicle fitness by that total value (sum) of all the fitnesses together. This means that they all add up to 1, which is important later on.

sketch.js

```
let target
let vehicles = []

function setup()
{
  createCanvas(400, 400)
  target = new Target()
  for (let i = 0; i < 50; i++)
  {
    vehicles[i] = new Vehicle(random(width), random(height))
  }
}

function draw()
{
  background(220)
  target.move()
  target.show()
  for (let vehicle of vehicles)
  {
    vehicle.seek(target)
    vehicle.move()
```

```
    vehicle.show()
  }
}
```

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

🗒️ Notes

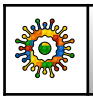
The `vehicle of vehicles` goes through the array of `vehicles`, one by one. We have created this new function, but nothing will happen until we call it.

🌻 Challenge

You could change the name to (thing of vehicles), and so what else would you have to change?

🛠️ Code Explanation

`sum = 0`	The total (sum) starts with zero
`sum += vehicle.fitness`	Go through all the vehicles adding their fitness scores as we go along
`vehicle.fitness = vehicle.fitness / sum`	Change the fitness score of each vehicle by dividing by the total (sum)



Weighted selection

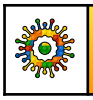
The next step is to select the highest-scoring vehicles, the ones with the best fitness. However, this isn't very natural, even if it seems very logical. This is an approach you might consider whereby you walk through the population array, picking out the highest pairs of values and mating them, and removing them from the array before looking for the next highest pair.

Instead, we will use what is called `weighted selection`, which still gives every vehicle a chance to be selected but will favour those who have the highest fitness. The way that `Dan Shiffman` explained it in his brilliant book `Nature of Code` is summarised thus:

Imagine that you are in a relay race; the race starts, and you hand over the baton to the next runner, and so on, until your team reaches the end of the race. But instead of each runner running the same distance on each leg of the relay, the fittest runs a longer leg based on their fitness, and the least fit runs a shorter leg, and this is where the analogy breaks down a bit; the winner is the runner that crosses the line first.

This means that it is possible for any runner (depending on the distance of the race) to potentially win, although the fittest runner has a higher probability of winning because it is more likely that they will be there at the end when they cross the finishing line.

If you read the code below, you can see how that works, even if it isn't very intuitive. I will include a link to the website that has the book online for you to read.



Sketch C3.19 the relay race

We want to reproduce with the better vehicles based on their fitness. But first, we need to decide which ones are the best and select their brains. For this, we use a technique called `weighted selection`. This is a more realistic way of natural selection. The best ones will still have the best chance, so it isn't purely random, but it does give other lesser vehicles to reproduce as well.

It uses a probability, `random(1)`, which is the length of the race between `0` and `1`. It then goes through the array, reducing that distance (called `start`) until it reaches the end (`0`). Whichever vehicle crosses the line at the end is the winner. The clock stops (the array stops, hence `index--`), then continues again with the array from where we left off, with a new random number (race distance) and sees who the winner is on this occasion until we have gone through the whole array of vehicles (`population`).

Remember that the `fitness score` of all the vehicles adds up to `1`.

sketch.js

```
let target
let vehicles = []

function setup()
{
  createCanvas(400, 400)
  target = new Target()
  for (let i = 0; i < 50; i++)
  {
    vehicles[i] = new Vehicle(random(width), random(height))
  }
}

function draw()
{
```

```
  background(220)
  target.move()
  target.show()
  for (let vehicle of vehicles)
  {
    vehicle.seek(target)
    vehicle.move()
    vehicle.show()
  }
}

function normaliseFitness()
{
  let sum = 0
  for (let vehicle of vehicles)
  {
    sum += vehicle.fitness
  }
  for (let vehicle of vehicles)
  {
    vehicle.fitness = vehicle.fitness / sum
  }
}

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

```
  index--
  return vehicles[index].brain
}
```

🗒️ Notes

This is a bit challenging to get your head around at first. This approach is memory efficient but computationally more challenging; this is only a problem if there is a very large population where another approach might be better.

🌻 Challenge

Can you think of a better way?

🛠️ Code Explanation

`index = 0`	We begin with the first in the array of population, this is our first runner
`start = random(1)`	We create a random number between 0 and 1, this is the length of the race
`while (start > 0)`	We keep going until we reach the end of the race
`start = start - vehicles[index].fitness`	We remove the fitness score for that vehicle (runner) reducing the start random value (race distance)
`index++`	Go to the next vehicle (runner)
`index--`	When finished stop the race
`return vehicles[index].brain`	Get the detail of the best vehicle (runner)

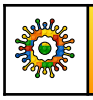
Reproduction

We want a `lifespan` for the current vehicles, say `250` frames, so that just before they die off, they reproduce a better version of themselves. They don't all mate; only those who have a reasonably high fitness.

From the `weighted selection`, we chose two parents and created a child through crossover, where the child will carry some of the genes from each parent. This child will have a mutation rate. This is important; otherwise, the gene pool becomes stagnant. There needs to be some element of randomness. This is often seen in nature. If those mutations are useful, they can be incorporated; otherwise, they die off.

The ml5.js machine learning library has a built-in function for `crossover()` and `mutation()` which is a relatively new addition but ever so helpful if you are developing simulations or games.

The `crossover()` function simply takes some of the weights of one successful parent's neural network with the weights of another successful parent's neural network and, in some way, combines them to produce a new neural network with these combined weights. There are various ways this can be done, but that is for another discussion. For now, let us just make use of it.



Sketch C3.20 the next generation

To create the next generation, we need to create a new function and we will call `reproduction()`. Here, we will create the next generation of children, which will be the same length as our first generation. This is a holding array called `nextVehicles`, and once we have filled it, we will rename it `vehicles`.

sketch.js

```
let target
let vehicles = []

function setup()
{
  createCanvas(400, 400)
  target = new Target()
  for (let i = 0; i < 50; i++)
  {
    vehicles[i] = new Vehicle(random(width), random(height))
  }
}

function draw()
{
  background(220)
  target.move()
  target.show()
  for (let vehicle of vehicles)
  {
    vehicle.seek(target)
    vehicle.move()
    vehicle.show()
  }
```

```
}

function normaliseFitness()
{
  let sum = 0
  for (let vehicle of vehicles)
  {
    sum += vehicle.fitness
  }
  for (let vehicle of vehicles)
  {
    vehicle.fitness = vehicle.fitness / sum
  }
}

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

function reproduction()
{
  let nextVehicles = []
  for (let i = 0; i < vehicles.length; i++)
  {
```

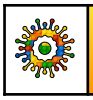
```
    let parentA = weightedSelection()
    let parentB = weightedSelection()
    let child = parentA.crossover(parentB)
    child.mutate(0.01)
    nextVehicles[i] = new Vehicle(random(width), random(height),
child)
  }
  vehicles = nextVehicles
}
```

🗒️ Notes

Notice that we have an extra argument for creating the next generation of vehicle, the `child`. We will need to add it to the `constructor()` function in the `Vehicle` class.

🛠️ Code Explanation

`nextVehicles = []`	We create an empty array for the new vehicles.
`parentA = weightedSelection()`	Select the first parent, call the weightedSelection() function
`parentB = weightedSelection()`	Select the second parent, call the weightedSelection() function
`child.mutate(0.01)`	When the child is created apply a tiny mutation rate of 0.01
`nextVehicles[i] = new Vehicle(random(width), random(height), child)`	Create a new vehicle from the child
`vehicles = nextVehicles`	Rename all the new vehicles: vehicles



Sketch C3.21 birth of a child

❗ Go over to `vehicle.js`

You may have noticed that when we create a `new Vehicle()`, we have included a third argument called `child`. This is effectively the `brain`, but when we originally created a vehicle, we only used two arguments, the `x` and the `y` position. So we need to factor that in when creating our second generation and onwards. We need to check if there is a `brain` in the first place, and if we have, then we update it with the new one.

We get around this by checking to see if there is already a brain and replacing it. If there isn't a brain, we give it one. We alter the `constructor()` function of the `Vehicle` Class as appropriate.

❗ We create an if...else() statement to check whether the vehicle already has a brain and if not, then create one. We have moved the neural network (brain) inside the else() statement, hence it is highlighted.

vehicle.js

```
class Vehicle
{
  constructor(x, y, brain)
  {
    this.position = createVector(x, y)
    this.velocity = createVector(0, 0)
    this.acceleration = createVector(0, 0)
    this.l = 5
    this.fitness = 0
    this.maxSpeed = 4
    this.maxForce = 0.2
    if (brain)
    {
      this.brain = brain
    }
    else
```

```
    {
      ml5.setBackend("cpu")
      this.brain = ml5.neuralNetwork({
        inputs: 5,
        outputs: 2,
        task: "regression",
        neuroEvolution: true,
        noTraining: true
      })
    }
  }

  move()
  {
    this.velocity.add(this.acceleration)
    this.velocity.limit(this.maxSpeed)
    this.position.add(this.velocity)
    this.acceleration.mult(0)
    let d = p5.Vector.dist(this.position, target.position)
    if (d < this.l + target.r)
    {
      this.fitness++
    }
  }

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

  seek(target)
  {
    let v = p5.Vector.sub(target.position, this.position)
```

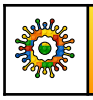
```
    let distance = v.mag() / width
    v.normalize()
    let inputs = [
      v.x,
      v.y,
      distance,
      this.velocity.x / this.maxSpeed,
      this.velocity.y / this.maxSpeed
    ]
    let outputs = this.brain.predictSync(inputs)
    let angle = outputs[0].value * TWO_PI
    let magnitude = outputs[1].value
    let force = p5.Vector.fromAngle(angle)
    force.setMag(magnitude)
    this.applyForce(force)
  }

  show()
  {
    let angle = this.velocity.heading()
    push()
    translate(this.position.x, this.position.y)
    rotate(angle)
    rect(0, 0, this.l*2, this.l)
    pop()
  }
}
```

🗒️ Notes

If you run it, you shouldn't get any errors, but neither will you see the next generation just yet. We need to have a lifespan so that the previous generation dies out and the next is birthed.

🛠️ Code Explanation

`constructor(x, y, brain)`	Added the brain (child) component argument to the constructor function
`if (brain)`	Check to see if the vehicle already has a brain
`this.brain = brain`	If it does then replace it with the new brain (child)



Sketch C3.22 lifeCounter

❗ Return to `sketch.js`

All we have got left to do is give the vehicle a `lifespan` and then `reproduce` it. We need to count the number of `iterations` rather than actual frames (which is another way of doing it); this way, we could have a countdown. We are also going to keep track of how many generations there are and display this later on.

sketch.js

```
let target
let vehicles = []
let lifeSpan = 250
let lifeCounter = 0
let generations = 0

function setup()
{
  createCanvas(400, 400)
  target = new Target()
  for (let i = 0; i < 50; i++)
  {
    vehicles[i] = new Vehicle(random(width), random(height))
  }
}

function draw()
{
  background(220)
  target.move()
  target.show()
  for (let vehicle of vehicles)
  {
```

```
    vehicle.seek(target)
    vehicle.move()
    vehicle.show()
  }
  lifeCounter++
  if (lifeCounter > lifeSpan)
  {
    normaliseFitness()
    reproduction()
    lifeCounter = 0
    generations++
  }
}

function normaliseFitness()
{
  let sum = 0
  for (let vehicle of vehicles)
  {
    sum += vehicle.fitness
  }
  for (let vehicle of vehicles)
  {
    vehicle.fitness = vehicle.fitness / sum
  }
}

function weightedSelection()
{
  let index = 0
  let start = random(1)
  while (start > 0)
  {
```

```
    start = start - vehicles[index].fitness
    index++
  }
  index--
  return vehicles[index].brain
}

function reproduction()
{
  let nextVehicles = []
  for (let i = 0; i < vehicles.length; i++)
  {
    let parentA = weightedSelection()
    let parentB = weightedSelection()
    let child = parentA.crossover(parentB)
    child.mutate(0.01)
    nextVehicles[i] = new Vehicle(random(width), random(height),
child)
  }
  vehicles = nextVehicles
}
```

🗒️ Notes

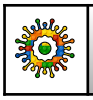
You should now see the generations appearing. They should also start to follow the target after a little while (you have to be patient). If you are impatient, then look at the next sketch.

🌻 Challenges

1. Add colour or images (instead of rectangles) for the scene.
2. How would you go about creating a 3D version (not easy but not impossible)?

🛠️ Code Explanation

`lifeSpan = 250`	The life of a vehicle
`lifeCounter = 0`	Keeps track of how many iterations there have been
`generations = 0`	Counts the number of generations
`lifeCounter++`	Counts the number of iterations, adding 1 each time
`if (lifeCounter > lifeSpan)`	Chen is to see if the vehicles have reached the end of their lifespan
`normaliseFitness()`	The fitness is normalised at the end of each generation
`reproduction()`	The reproduction happens at the end of the generation
`lifeCounter = 0`	Reset the counter
`generations++`	Add another generation to the tally

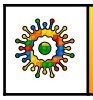


What next?

In a sense, you have done it. If you watch long enough, you will see the changes take place. The `vehicles` will start to follow the `target` as they evolve through the improved fitness levels of those who are successful. This, however, is a slow process, and there is a way of speeding it up and also having a bit more information on the number of `generations` you are at.

We can use a `slider` to speed up the process. Remember that the computer can do this much faster than we are watching it. This is so we can see it happening in real time, as it were.

You can be as creative as you want. I created some bees following a honey pot.



Sketch C3.23 adding a slider

Adding a slider, what we are doing is increasing the rate of iterations as another for() loop within draw(). We can also see the number of generations; this helps get an idea of how many generations have passed. The create slider function has three arguments: the minimum value (1), the maximum value (20), and the increment value (1).

sketch.js

```
let target
let vehicles = []
let lifeSpan = 250
let lifeCounter = 0
let generations = 0
let timeSlider

function setup()
{
  createCanvas(400, 400)
  target = new Target()
  for (let i = 0; i < 50; i++)
  {
    vehicles[i] = new Vehicle(random(width), random(height))
  }
  timeSlider = createSlider(1, 20, 1)
  timeSlider.position(10, 420)
}

function draw()
{
  background(220)
  target.show()
  for (let vehicle of vehicles)
```

```
  {
    vehicle.show()
  }

  for (let i = 0; i < timeSlider.value(); i++)
  {
    for (let vehicle of vehicles)
    {
      vehicle.seek(target)
      vehicle.move()
    }
    target.move()
    lifeCounter++
  }

  if (lifeCounter > lifeSpan)
  {
    normaliseFitness()
    reproduction()
    lifeCounter = 0
    generations++
  }
  textSize(40)
  text(generations, 10, 380)
}

function normaliseFitness()
{
  let sum = 0
  for (let vehicle of vehicles)
  {
    sum += vehicle.fitness
  }
```

```
  for (let vehicle of vehicles)
  {
    vehicle.fitness = vehicle.fitness / sum
  }
}

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

function reproduction()
{
  let nextVehicles = []
  for (let i = 0; i < vehicles.length; i++)
  {
    let parentA = weightedSelection()
    let parentB = weightedSelection()
    let child = parentA.crossover(parentB)
    child.mutate(0.01)
    nextVehicles[i] = new Vehicle(random(width), random(height),
child)
  }
  vehicles = nextVehicles
}
```

🗒️ Notes

Moving the `slider` along until around `150` iterations shows a huge improvement in their attraction to the target.

🌻 Challenge

Try other values

Figure C3.23


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
NeuroEvolution
by TheHappyCoder
Sketch Files
index.html
sketch.js
style.css
target.js
vehicle.js
sketch.js
Saved: just now
Preview
77
78 function reproduction()
79 {
80 let nextVehicles = []
81 for (let i = 0; i < vehicles.length; i++)
82 {
83 let parentA = weightedSelection()
84 let parentB = weightedSelection()
85 let child = parentA.crossover(parentB)
86 child.mutate(0.01)
87 nextVehicles[i] = new Vehicle(random(width), random(height), child)
88 }
89 vehicles = nextVehicles
90 }
Console
Clear
164