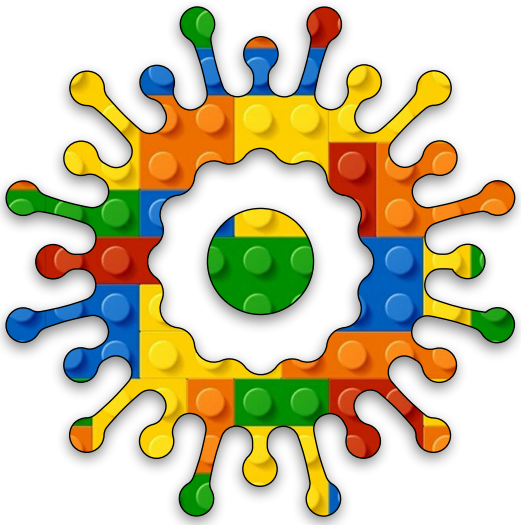
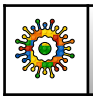
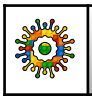
Making Games Module A Unit #6 maze game





Content

Module A Unit #6 the maze



Introduction to the maze

This is in two parts:

Section 1: Creating the Maze

The first part takes you step by step through creating a maze where every cell is visited and then retraces its steps back to the start.

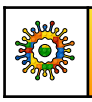
Section 2: Moving around the Maze

The second section is to use that created maze and make a navigable game out of it. You could have a Pac-Man type game where there are resources to collect and enemies seeking you out.

Section 1: Creating the Maze

This uses an algorithm to create a maze where it has visited every part at least once. It then backtracks, checking that there are no cells that have been missed. This is a solution to a challenging problem, and now you will be able to create any maze, any size, repeatedly.

You will need to add another file called `cell.js`.



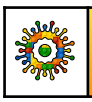
Sketch A6.1 adding the file

❗ index.html

Create a file called cell.js and add it to the index.html file.

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.10/p5.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.10/addons/p5.sound.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="cell.js"></script>
  </body>
</html>
```



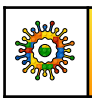
Sketch A6.2 adding a grid

❗ sketch.js

Using depth-first search. Create a grid of cells (rows and cols) numbered from 0 to whatever. The width of a cell is w in sketch.js.

sketch.js

```
let cols
let rows
let w = 40

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

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



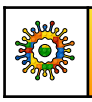
Sketch A6.3 creating a cell class

❗ cell.js

And in the `cell.js` we create the `Cell class`, where `i` is the number of `cols` and `j` is the number of `rows`.

cell.js

```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
  }
}
```



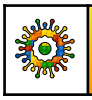
Sketch A6.4 rows and columns

❗ sketch.js

In the `sketch.js`, we will have `10` cols and `10` rows. The `floor()` function returns whole numbers.

sketch.js

```
let cols
let rows
let w = 40

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

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

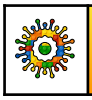


Sketch A6.5 a grid array

Create a nested loop to create 10x10 cells and an array (grid) to put them in. You can `console.log(grid.length)` to check that you have 100 cells.

sketch.js

```
let cols
let rows
let w = 40
let grid = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width/w)
  rows = floor(height/w)
  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
}

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



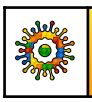
Sketch A6.6 display grid array

We can loop through all of them in `draw()` and display them with a `show()` function in the Cell class.

sketch.js

```
let cols
let rows
let w = 40
let grid = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width/w)
  rows = floor(height/w)
  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < grid.length; i++)
  {
    grid[i].show()
  }
```

```
}
```



Sketch A6.7 drawing the grid

❗ cell.js

The `show()` function in the Cell class.

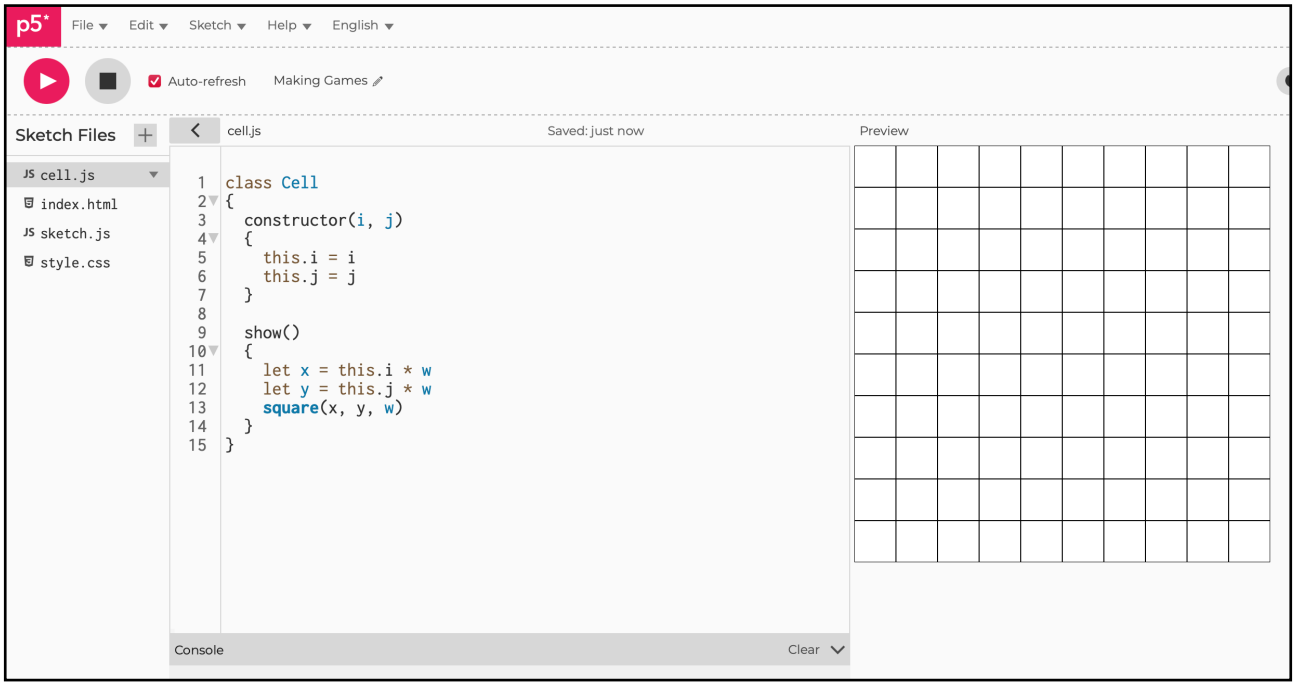
cell.js

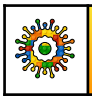
```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
  }

  show()
  {
    let x = this.i * w
    let y = this.j * w
    square(x, y, w)
  }
}
```

Notes

You should get something like this below.

Figure A6.7


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Making Games
Sketch Files
cell.js
index.html
sketch.js
style.css
cell.js
Saved: just now
Preview
class Cell
{
constructor(i, j)
{
this.i = i
this.j = j
}
show()
{
let x = this.i * w
let y = this.j * w
square(x, y, w)
}
}
Console
Clear



Sketch A6.8 lines not squares

A good way to check is to draw it out on a piece of scrap paper. Here is the cell class with the necessary amendments. Remove the square.

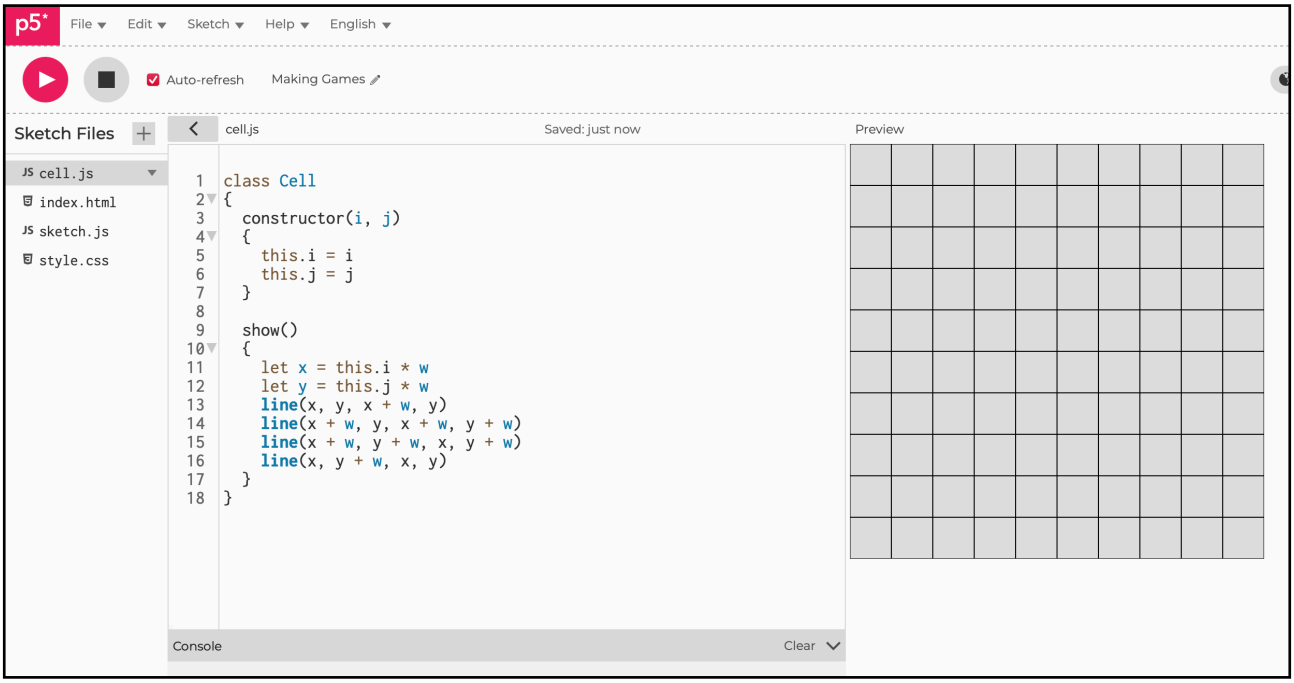
cell.js

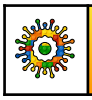
```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
  }

  show()
  {
    let x = this.i * w
    let y = this.j * w
    line(x, y, x + w, y)
    line(x + w, y, x + w, y + w)
    line(x + w, y + w, x, y + w)
    line(x, y + w, x, y)
  }
}
```

Notes

It is now grey because these are lines, not squares, and the background is showing through.

Figure A6.8


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Making Games
Sketch Files
cell.js
index.html
sketch.js
style.css
cell.js
Saved: just now
Preview
class Cell
{
constructor(i, j)
{
this.i = i
this.j = j
}
show()
{
let x = this.i * w
let y = this.j * w
line(x, y, x + w, y)
line(x + w, y, x + w, y + w)
line(x + w, y + w, x, y + w)
line(x, y + w, x, y)
}
}
Console
Clear

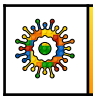


Sketch A6.9 checking for walls

Using a boolean to check if there are walls present, clockwise, top, right, bottom, left. Create a boolean array for each wall in the constructor function. The order of the walls is important as it cycles through the array in that order.

cell.js

```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
    this.walls = [true, true, true, true]
  }

  show()
  {
    let x = this.i * w
    let y = this.j * w
    line(x, y, x + w, y)
    line(x + w, y, x + w, y + w)
    line(x + w, y + w, x, y + w)
    line(x, y + w, x, y)
  }
}
```



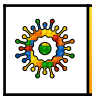
Sketch A6.10 walls and lines

Now to organise the walls corresponding to the lines, we are replacing the lines with walls.

cell.js

```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
    this.walls = [true, true, true, true]
  }

  show()
  {
    let x = this.i * w
    let y = this.j * w
    if (this.walls[0])
    {
      line(x, y, x + w, y)
    }
    if (this.walls[1])
    {
      line(x + w, y, x + w, y + w)
    }
    if (this.walls[2])
    {
      line(x + w, y + w, x, y + w)
    }
    if (this.walls[3])
    {
```

```
      line(x, y + w, x, y)
    }
  }
}
```



Sketch A6.11 keeping track

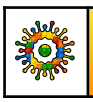
❗ sketch.js

We need to keep track of what has been visited and what hasn't.

sketch.js

```
let cols
let rows
let w = 40
let grid = []
let current

function setup()
{
  createCanvas(400, 400)
  cols = floor(width/w)
  rows = floor(height/w)

  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
  current = grid[0]
}

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

```
  for (let i = 0; i < grid.length; i++)
  {
    grid[i].show()
  }
  current.visited = true
}
```



Sketch A6.12 a false visit

❗ cell.js

Putting it into the `cell.js`, start off with a false visit (i.e. not visited).

cell.js

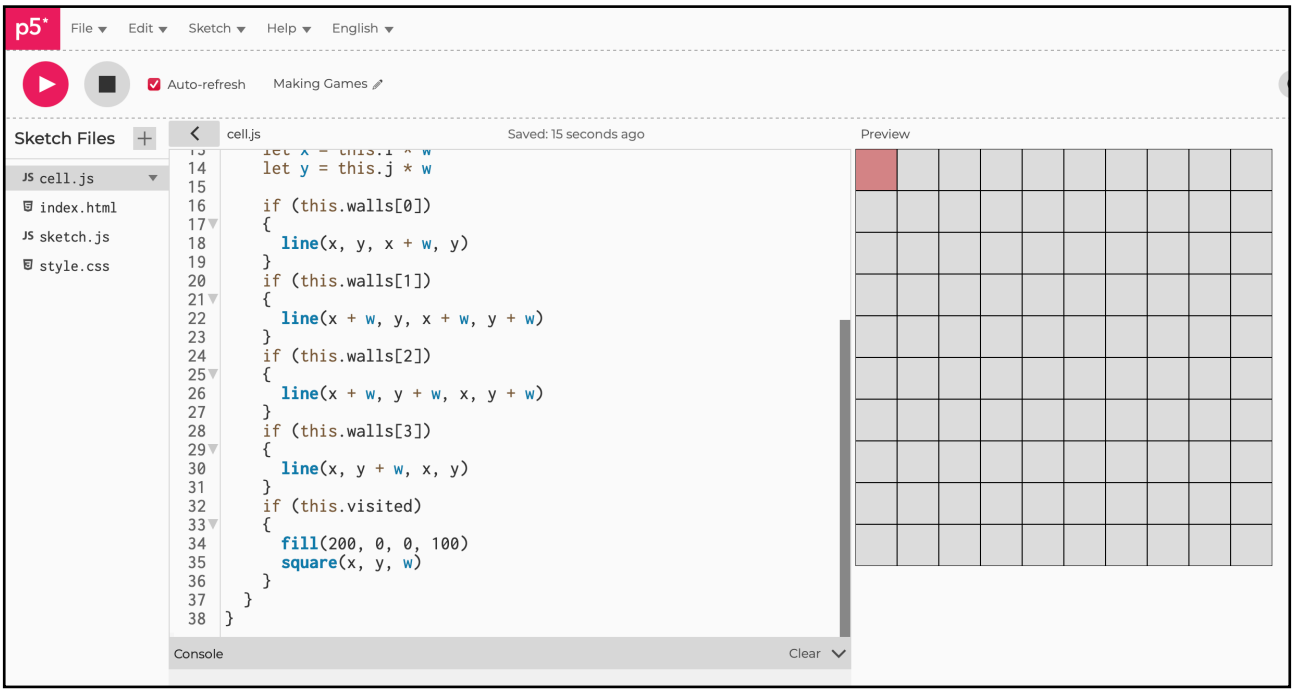
```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
    this.walls = [true, true, true, true]
    this.visited = false
  }

  show()
  {
    let x = this.i * w
    let y = this.j * w

    if (this.walls[0])
    {
      line(x, y, x + w, y)
    }
    if (this.walls[1])
    {
      line(x + w, y, x + w, y + w)
    }
    if (this.walls[2])
    {
      line(x + w, y + w, x, y + w)
```

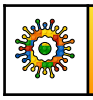
```
    }
    if (this.walls[3])
    {
      line(x, y + w, x, y)
    }
    if (this.visited)
    {
      fill(200, 0, 0, 100)
      square(x, y, w)
    }
  }
}
```

Notes

You should have something like this below.

Figure A6.12


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Making Games
Sketch Files
cell.js
index.html
sketch.js
style.css
cell.js
Saved: 15 seconds ago
Preview
14 let y = this.j * w
16 if (this.walls[0])
17 {
18 line(x, y, x + w, y)
19 }
20 if (this.walls[1])
21 {
22 line(x + w, y, x + w, y + w)
23 }
24 if (this.walls[2])
25 {
26 line(x + w, y + w, x, y + w)
27 }
28 if (this.walls[3])
29 {
30 line(x, y + w, x, y)
31 }
32 if (this.visited)
33 {
34 fill(200, 0, 0, 100)
35 square(x, y, w)
36 }
37 }
38 }
Console
Clear



Sketch A6.13 checking the neighbours

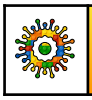
In cell.js, checking the neighbours (there is a lot of code to add, but it is mostly repetitive). It goes round in order: TOP -> RIGHT -> BOTTOM -> LEFT.

cell.js

```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
    this.walls = [true, true, true, true]
    this.visited = false
  }

  checkNeighbours()
  {
    let neighbours = []
    let top = grid[index(this.i, this.j - 1)]
    let right = grid[index(this.i + 1, this.j)]
    let bottom = grid[index(this.i, this.j + 1)]
    let left = grid[index(this.i - 1, this.j)]
    if (top && !top.visited)
    {
      neighbours.push(top)
    }
    if (right && !right.visited)
    {
      neighbours.push(right)
    }
    if (bottom && !bottom.visited)
```

```
    {
      neighbours.push(bottom)
    }
    if (left && !left.visited)
    {
      neighbours.push(left)
    }
    if (neighbours.length > 0)
    {
      let r = floor(random(0, neighbours.length))
      return neighbours[r]
    }
    else
    {
      return undefined
    }
  }

  show()
  {
    let x = this.i * w
    let y = this.j * w

    if (this.walls[0])
    {
      line(x, y, x + w, y)
    }
    if (this.walls[1])
    {
      line(x + w, y, x + w, y + w)
    }
    if (this.walls[2])
    {
```

```
      line(x + w, y + w, x, y + w)
    }
    if (this.walls[3])
    {
      line(x, y + w, x, y)
    }
    if (this.visited)
    {
      fill(200, 0, 0, 100)
      square(x, y, w)
    }
  }
}
```



Sketch A6.14 checking for edges

❗ sketch.js

In `sketch.js`, also need to check if it is on the edge because there are no neighbours on that side.

sketch.js

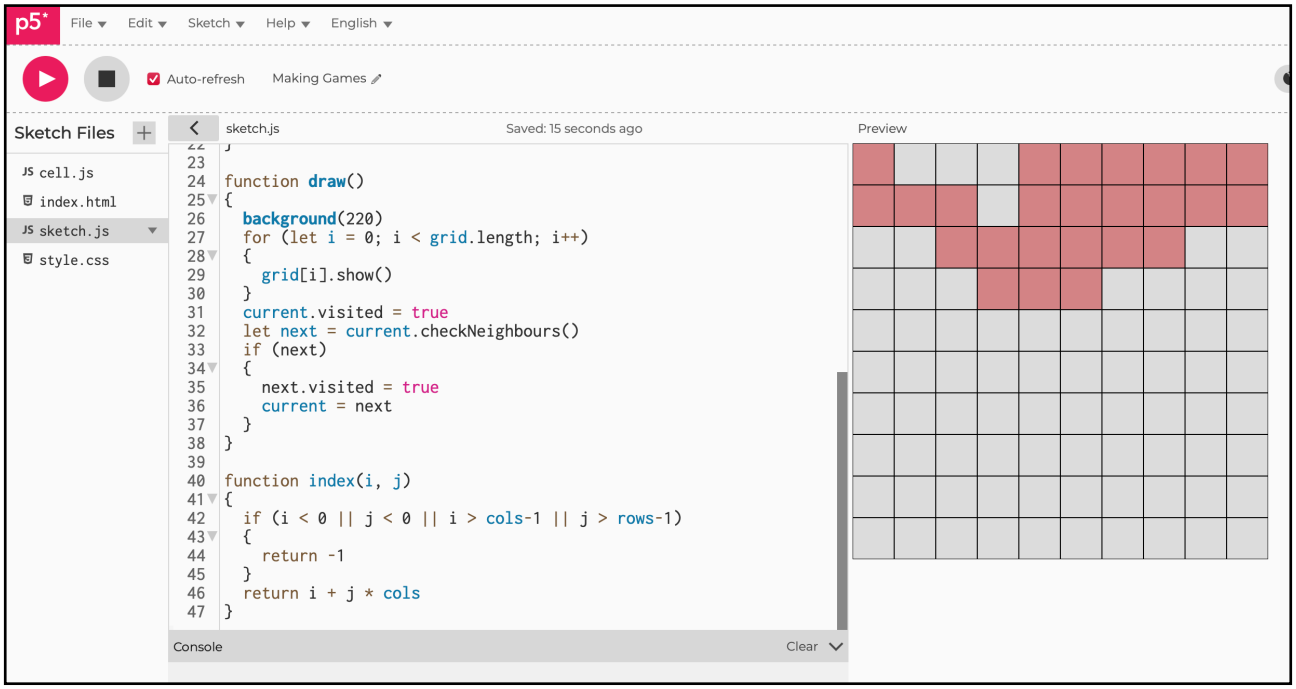
```
let cols
let rows
let w = 40
let grid = []
let current

function setup()
{
  createCanvas(400, 400)
  cols = floor(width/w)
  rows = floor(height/w)

  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
  current = grid[0]
}

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

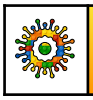
```
  for (let i = 0; i < grid.length; i++)
  {
    grid[i].show()
  }
  current.visited = true
  let next = current.checkNeighbours()
  if (next)
  {
    next.visited = true
    current = next
  }
}

function index(i, j)
{
  if (i < 0 || j < 0 || i > cols-1 || j > rows-1)
  {
    return -1
  }
  return i + j * cols
}
```

Notes

Finally, at the end of this section, you should get a randomly generated image similar to (but not exactly like) this one.

Figure A6.14


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Making Games
Sketch Files
cell.js
index.html
sketch.js
style.css
sketch.js
Saved: 15 seconds ago
Preview
function draw()
{
background(220)
for (let i = 0; i < grid.length; i++)
{
grid[i].show()
}
current.visited = true
let next = current.checkNeighbours()
if (next)
{
next.visited = true
current = next
}
}
function index(i, j)
{
if (i < 0 || j < 0 || i > cols-1 || j > rows-1)
{
return -1
}
return i + j * cols
}
Console
Clear



Sketch A6.15 removing the walls

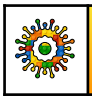
Create a `removeWalls()` function in `draw()`.

sketch.js

```
let cols
let rows
let w = 40
let grid = []
let current

function setup()
{
  createCanvas(400, 400)
  cols = floor(width/w)
  rows = floor(height/w)

  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
  current = grid[0]
}

function draw()
{
  background(220)
  for (let i = 0; i < grid.length; i++)
```

```
  {
    grid[i].show()
  }
  current.visited = true
  let next = current.checkNeighbours()
  if (next)
  {
    next.visited = true
    removeWalls(current, next)
    current = next
  }
}

function index(i, j)
{
  if (i < 0 || j < 0 || i > cols-1 || j > rows-1)
  {
    return -1
  }
  return i + j * cols
}

function removeWalls(a, b)
{
  let x = a.i - b.i
  if (x == 1)
  {
    a.walls[3] = false
    b.walls[1] = false
  }
  else if (x == -1)
  {
    a.walls[1] = false
```

```
    b.walls[3] = false
  }

  let y = a.j - b.j
  if (y == 1)
  {
    a.walls[0] = false
    b.walls[2] = false
  }
  else if (y == -1)
  {
    a.walls[2] = false
    b.walls[0] = false
  }
}
```



Sketch A6.16 draw and delete lines

❗ cell.js

Deleting the lines where appropriate.

cell.js

```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
    this.walls = [true, true, true, true]
    this.visited = false
  }

  checkNeighbours()
  {
    let neighbours = []
    let top = grid[index(this.i, this.j - 1)]
    let right = grid[index(this.i + 1, this.j)]
    let bottom = grid[index(this.i, this.j + 1)]
    let left = grid[index(this.i - 1, this.j)]

    if (top && !top.visited)
    {
      neighbours.push(top)
    }
    if (right && !right.visited)
    {
      neighbours.push(right)
    }
```

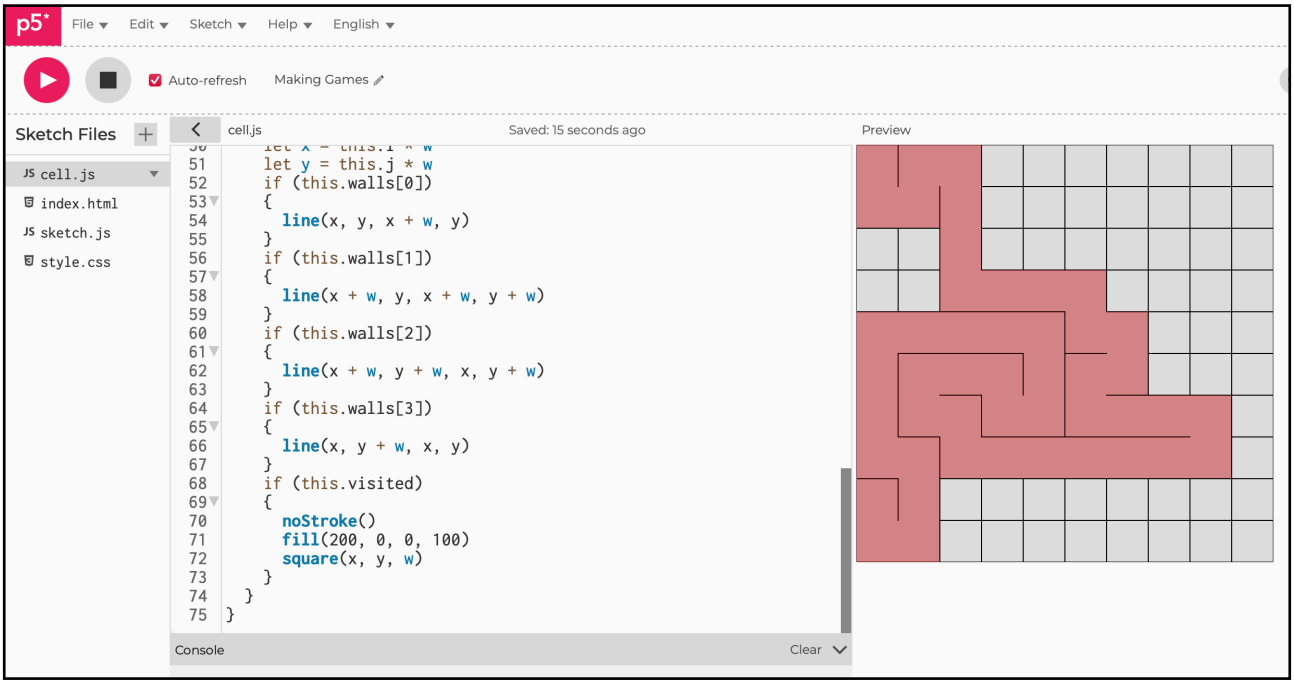
```
    if (bottom && !bottom.visited)
    {
      neighbours.push(bottom)
    }
    if (left && !left.visited)
    {
      neighbours.push(left)
    }

    if (neighbours.length > 0)
    {
      let r = floor(random(0, neighbours.length))
      return neighbours[r]
    }
    else
    {
      return undefined
    }
  }

  show()
  {
    stroke(0)
    let x = this.i * w
    let y = this.j * w
    if (this.walls[0])
    {
      line(x, y, x + w, y)
    }
    if (this.walls[1])
    {
      line(x + w, y, x + w, y + w)
    }
```

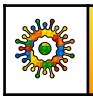
```
    if (this.walls[2])
    {
      line(x + w, y + w, x, y + w)
    }
    if (this.walls[3])
    {
      line(x, y + w, x, y)
    }
    if (this.visited)
    {
      noStroke()
      fill(200, 0, 0, 100)
      square(x, y, w)
    }
  }
}
```

Notes

You should be getting something like this, not perfect but getting closer.

Figure A6.16


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Making Games
Sketch Files
cell.js
index.html
sketch.js
style.css
cell.js
Saved: 15 seconds ago
Preview
let y = this.j * w
if (this.walls[0])
{
line(x, y, x + w, y)
}
if (this.walls[1])
{
line(x + w, y, x + w, y + w)
}
if (this.walls[2])
{
line(x + w, y + w, x, y + w)
}
if (this.walls[3])
{
line(x, y + w, x, y)
}
if (this.visited)
{
noStroke()
fill(200, 0, 0, 100)
square(x, y, w)
}
}
}
Console
Clear



Sketch A6.17 current cell colour

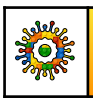
Going to draw the current cell a different colour.

cell.js

```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
    this.walls = [true, true, true, true]
    this.visited = false
  }

  checkNeighbours()
  {
    let neighbours = []
    let top = grid[index(this.i, this.j - 1)]
    let right = grid[index(this.i + 1, this.j)]
    let bottom = grid[index(this.i, this.j + 1)]
    let left = grid[index(this.i - 1, this.j)]

    if (top && !top.visited)
    {
      neighbours.push(top)
    }
    if (right && !right.visited)
    {
      neighbours.push(right)
    }
    if (bottom && !bottom.visited)
```

```
    {
      neighbours.push(bottom)
    }
    if (left && !left.visited)
    {
      neighbours.push(left)
    }

    if (neighbours.length > 0)
    {
      let r = floor(random(0, neighbours.length))
      return neighbours[r]
    }
    else
    {
      return undefined
    }
  }

  highlight()
  {
    let x = this.i * w
    let y = this.j * w
    noStroke()
    fill(0, 255, 0, 100)
    square(x, y, w)
  }

  show()
  {
    stroke(0)
    let x = this.i * w
    let y = this.j * w
```

```
    if (this.walls[0])
    {
      line(x, y, x + w, y)
    }
    if (this.walls[1])
    {
      line(x + w, y, x + w, y + w)
    }
    if (this.walls[2])
    {
      line(x + w, y + w, x, y + w)
    }
    if (this.walls[3])
    {
      line(x, y + w, x, y)
    }

    if (this.visited)
    {
      noStroke()
      fill(200, 0, 0, 100)
      square(x, y, w)
    }
  }
}
```



Sketch A6.18 displaying it

❗ sketch.js
And we add it to sketch.js.

sketch.js

```
let cols
let rows
let w = 40
let grid = []
let current

function setup()
{
  createCanvas(400, 400)
  cols = floor(width/w)
  rows = floor(height/w)
  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
  current = grid[0]
}

function draw()
{
  background(220)
  for (let i = 0; i < grid.length; i++)
```

```
  {
    grid[i].show()
  }
  current.visited = true
  current.highlight()
  let next = current.checkNeighbours()
  if (next)
  {
    next.visited = true
    removeWalls(current, next)
    current = next
  }
}

function index(i, j)
{
  if (i < 0 || j < 0 || i > cols-1 || j > rows-1)
  {
    return -1
  }
  return i + j * cols
}

function removeWalls(a, b)
{
  let x = a.i - b.i
  if (x == 1)
  {
    a.walls[3] = false
    b.walls[1] = false
  }
  else if (x == -1)
  {
```

```
    a.walls[1] = false
    b.walls[3] = false
  }

  let y = a.j - b.j
  if (y == 1)
  {
    a.walls[0] = false
    b.walls[2] = false
  }
  else if (y == -1)
  {
    a.walls[2] = false
    b.walls[0] = false
  }
}
```

🗒️ Notes

You should get something like this below.

🌻 Challenge

Decrease `W` (i.e. increase the number of cells in the grid) and see how long before it cuts in on itself and stops.

Figure A6.18a

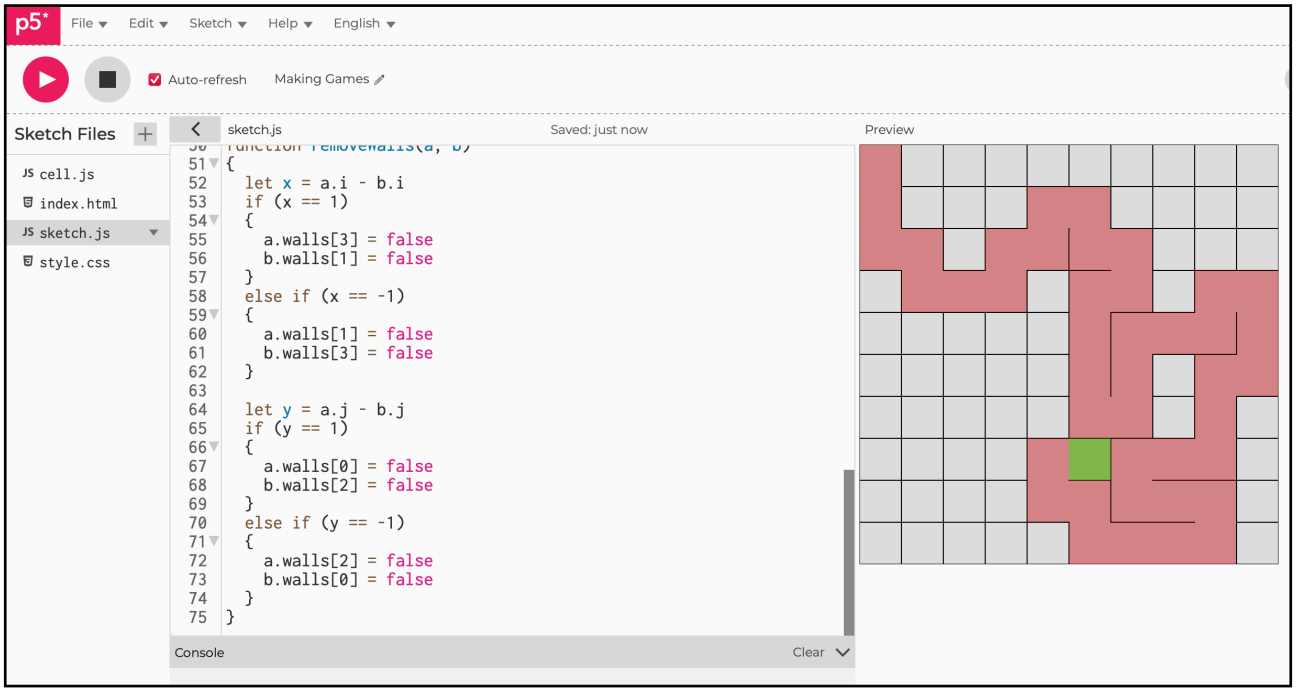
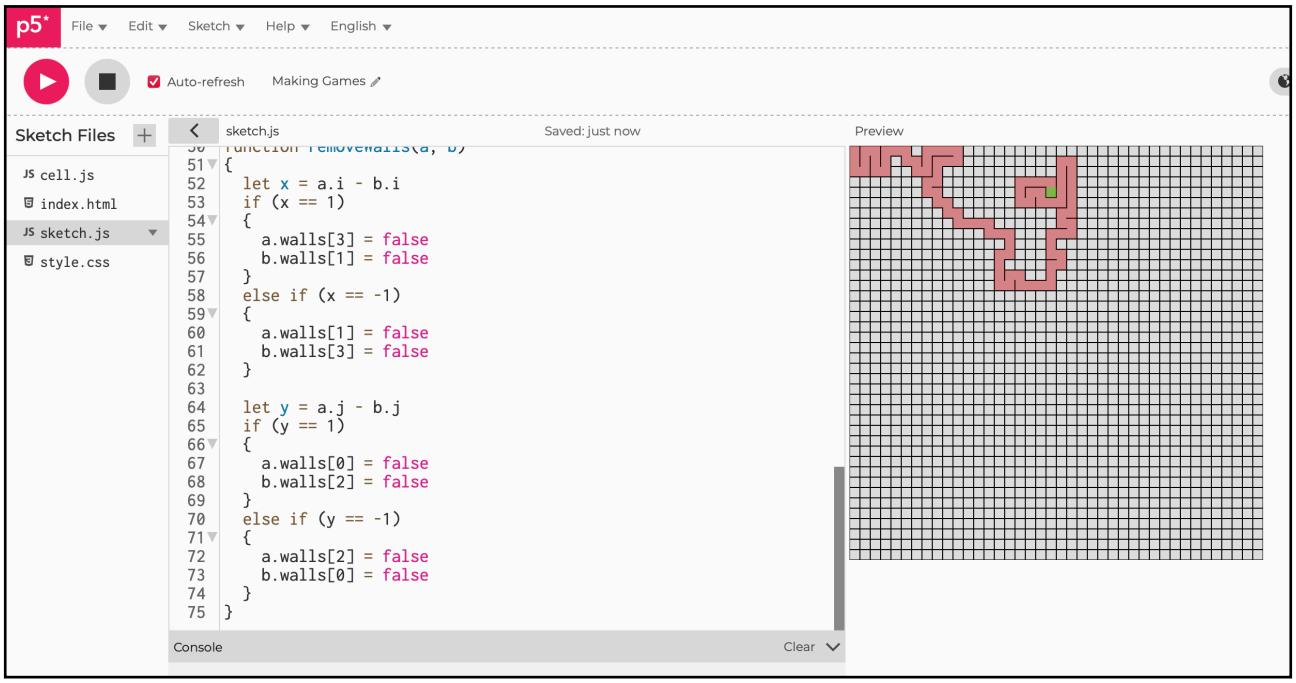
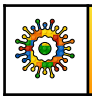


Figure A6.18b





Sketch A6.19 backtracking

Now for the backtracking bit! Introducing a stack. It is a type of data structure. Last one in, first one out (imagine a stack of paper). A queue is the opposite, where the first person in is the first one out. Simple.

sketch.js

```
let cols
let rows
let w = 40
let grid = []
let current
let stack = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width/w)
  rows = floor(height/w)
  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
  current = grid[0]
}

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

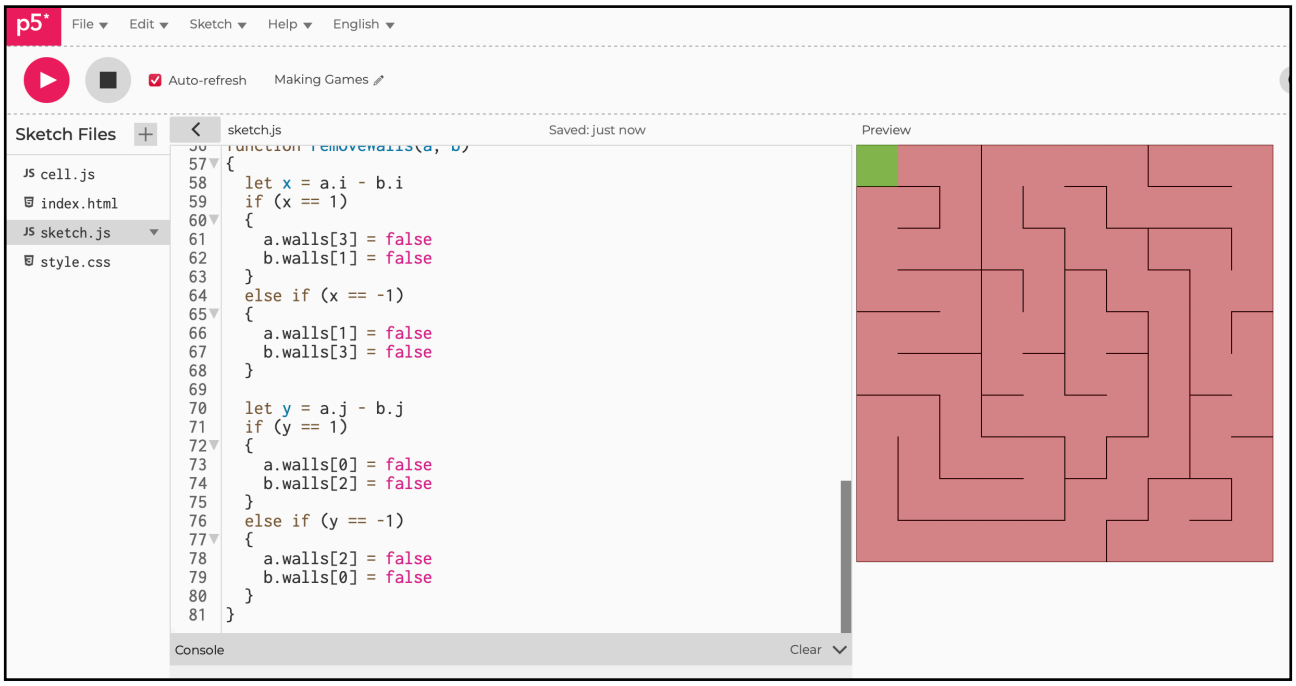
```
  for (let i = 0; i < grid.length; i++)
  {
    grid[i].show()
  }
  current.visited = true
  current.highlight()
  let next = current.checkNeighbours()
  if (next)
  {
    next.visited = true
    stack.push(current)
    removeWalls(current, next)
    current = next
  }
  else if (stack.length > 0)
  {
    current = stack.pop()
  }
}

function index(i, j)
{
  if (i < 0 || j < 0 || i > cols-1 || j > rows-1)
  {
    return -1
  }
  return i + j * cols
}

function removeWalls(a, b)
{
  let x = a.i - b.i
  if (x == 1)
```

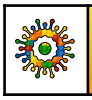
```
  {
    a.walls[3] = false
    b.walls[1] = false
  }
  else if (x == -1)
  {
    a.walls[1] = false
    b.walls[3] = false
  }

  let y = a.j - b.j
  if (y == 1)
  {
    a.walls[0] = false
    b.walls[2] = false
  }
  else if (y == -1)
  {
    a.walls[2] = false
    b.walls[0] = false
  }
}
```

🗒️ Notes

It should go to every cell.

Figure A6.19


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Making Games
Sketch Files
cell.js
index.html
sketch.js
style.css
sketch.js
Saved: just now
Preview
function removeWalls(a, b)
{
let x = a.i - b.i
if (x == 1)
{
a.walls[3] = false
b.walls[1] = false
}
else if (x == -1)
{
a.walls[1] = false
b.walls[3] = false
}
let y = a.j - b.j
if (y == 1)
{
a.walls[0] = false
b.walls[2] = false
}
else if (y == -1)
{
a.walls[2] = false
b.walls[0] = false
}
}
Console
Clear



Sketch A6.20 printing the maze

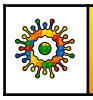
Having a go at printing the maze.

sketch.js

```
let cols
let rows
let w = 40
let grid = []
let current
let stack = []
let canvas

function setup()
{
  canvas = createCanvas(400, 400)
  cols = floor(width/w)
  rows = floor(height/w)
  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
  current = grid[0]
}

function draw()
{
  background(255)
```

```
  for (let i = 0; i < grid.length; i++)
  {
    grid[i].show()
  }
  current.visited = true
  current.highlight()
  let next = current.checkNeighbours()
  if (next)
  {
    next.visited = true
    stack.push(current)
    removeWalls(current, next)
    current = next
  }
  else if (stack.length > 0)
  {
    current = stack.pop()
  }
}

function index(i, j)
{
  if (i < 0 || j < 0 || i > cols-1 || j > rows-1)
  {
    return -1
  }
  return i + j * cols
}

function removeWalls(a, b)
{
  let x = a.i - b.i
  if (x == 1)
```

```
  {
    a.walls[3] = false
    b.walls[1] = false
  }
  else if (x == -1)
  {
    a.walls[1] = false
    b.walls[3] = false
  }

  let y = a.j - b.j
  if (y == 1)
  {
    a.walls[0] = false
    b.walls[2] = false
  }
  else if (y == -1)
  {
    a.walls[2] = false
    b.walls[0] = false
  }
}

function mouseClicked()
{
  saveCanvas(canvas, 'maze', 'png')
}
```



Sketch A6.21 now ready to print

❗ cell.js

In `cell.js`, just a few adjustments.

cell.js

```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
    this.walls = [true, true, true, true]
    this.visited = false
  }

  checkNeighbours()
  {
    let neighbours = []
    let top = grid[index(this.i, this.j - 1)]
    let right = grid[index(this.i + 1, this.j)]
    let bottom = grid[index(this.i, this.j + 1)]
    let left = grid[index(this.i - 1, this.j)]

    if (top && !top.visited)
    {
      neighbours.push(top)
    }
    if (right && !right.visited)
    {
      neighbours.push(right)
    }
```

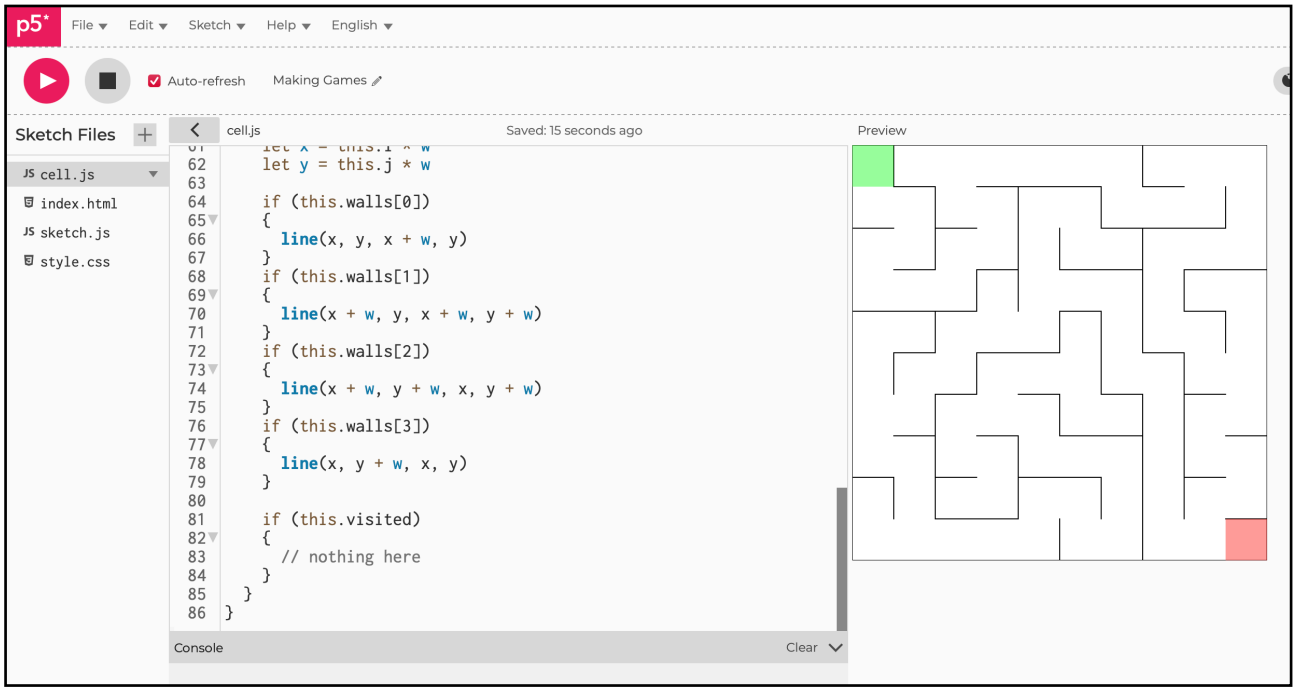
```
    if (bottom && !bottom.visited)
    {
      neighbours.push(bottom)
    }
    if (left && !left.visited)
    {
      neighbours.push(left)
    }

    if (neighbours.length > 0)
    {
      let r = floor(random(0, neighbours.length))
      return neighbours[r]
    }
    else
    {
      return undefined
    }
  }

  highlight()
  {
    let x = this.i * w
    let y = this.j * w
    noStroke()
    fill(0, 255, 0, 100)
    square(x, y, w)
    fill(255, 0, 0, 100)
    square(width - w, height - w, w)
  }

  show()
  {
```

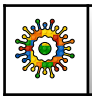
```
    stroke(0)
    let x = this.i * w
    let y = this.j * w

    if (this.walls[0])
    {
      line(x, y, x + w, y)
    }
    if (this.walls[1])
    {
      line(x + w, y, x + w, y + w)
    }
    if (this.walls[2])
    {
      line(x + w, y + w, x, y + w)
    }
    if (this.walls[3])
    {
      line(x, y + w, x, y)
    }

    if (this.visited)
    {
      // nothing here
    }
  }
}
```

📝 Notes

This gives us a nice, clean maze. You might want to add the finishing square (in red, perhaps).

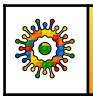
Figure A6.21


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Making Games
Sketch Files
cell.js
index.html
sketch.js
style.css
cell.js
Saved: 15 seconds ago
Preview
let y = this.j * w
if (this.walls[0])
{
line(x, y, x + w, y)
}
if (this.walls[1])
{
line(x + w, y, x + w, y + w)
}
if (this.walls[2])
{
line(x + w, y + w, x, y + w)
}
if (this.walls[3])
{
line(x, y + w, x, y)
}
if (this.visited)
{
// nothing here
}
}
}
Console
Clear



Section 2: Navigating the Maze

Here we make a playable version of the maze and start to create a Pac-Man style game. This would be a good game to develop as is but could be the basis for developing an AI to travel through the maze, learning on each iteration... something for the future.



Sketch A6.22 adding control keys

❗ sketch.js

Adding a player controlled with the arrow keys. `indexP` is the index of where the player is in the grid. The grid starts at `0` and goes incrementally across and then drops down a level and continues. So in a `5x5` grid, each grid is numbered `0` to `24`. The m is calculated for when you move down and up a level on the grid depending on its size. Note, cell.js remains unaltered.

❗ Remove the `mouseClicked()` function for this, otherwise you will keep generating PNGs!

sketch.js

```
let cols
let rows
let w = 40
let grid = []
let current
let stack = []
let x = 0
let y = 0
let indexP = 0
let m

function setup()
{
  canvas = createCanvas(400, 400)
  background(255)
  cols = floor(width / w)
  rows = floor(height / w)
  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
```

```
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
  current = grid[0]
  m = width/w
}

function draw()
{
  background(220)
  for (let i = 0; i < grid.length; i++)
  {
    grid[i].show()
  }
  current.visited = true
  current.highlight()
  let next = current.checkNeighbours()
  if (next)
  {
    next.visited = true
    stack.push(current)
    removeWalls(current, next)
    current = next
  }
  else if (stack.length > 0)
  {
    current = stack.pop()
  }
  noStroke()
  fill(0, 255, 0, 100)
  square(x, y, w)
```

```
  fill(255, 0, 0, 100)
  square(width - w, height - w, w)
}

function index(i, j)
{
  if (i < 0 || j < 0 || i > cols - 1 || j > rows - 1)
  {
    return -1
  }
  return i + j * cols
}

function removeWalls(a, b)
{
  let x = a.i - b.i
  if (x == 1)
  {
    a.walls[3] = false
    b.walls[1] = false
  }
  else if (x == -1)
  {
    a.walls[1] = false
    b.walls[3] = false
  }

  let y = a.j - b.j
  if (y == 1)
  {
    a.walls[0] = false
    b.walls[2] = false
  }
```

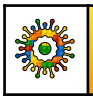
```
  else if (y == -1)
  {
    a.walls[2] = false
    b.walls[0] = false
  }
}
```

```
function keyPressed()
{
  if (keyCode == RIGHT_ARROW)
  {
    if (grid[indexP].walls[1] == false)
    {
      indexP = indexP + 1
      x = x + w
    }
  }
  if (keyCode == LEFT_ARROW)
  {
    if (grid[indexP].walls[3] == false)
    {
      indexP = indexP - 1
      x = x - w
    }
  }
  if (keyCode == UP_ARROW)
  {
    if (grid[indexP].walls[0] == false)
    {
      indexP = indexP - m
      y = y - w
    }
  }
```

```
  if (keyCode == DOWN_ARROW)
  {
    if (grid[indexP].walls[2] == false)
    {
      indexP = indexP + m
      y = y + w
    }
  }
}
```

🗒️ Notes

The data for each cell wall is given as:
`console.log(grid[0].walls[0])`.



Sketch A6.23 continuous movement

Making the player travel continuously while the key is pressed. Changing the name of the `keyPressed()` function. Adding a `frameRate()` of `10` to slow it down, otherwise it is impossible to stop midway between walls. Note, `cell.js` remains unaltered.

sketch.js

```
let cols
let rows
let w = 40
let grid = []
let current
let stack = []
let x = 0
let y = 0
let indexP = 0
let m

function setup()
{
  canvas = createCanvas(400, 400)
  background(255)
  cols = floor(width / w)
  rows = floor(height / w)
  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
```

```
  current = grid[0]
  m = width/w
}

function draw()
{
  background(255)
  for (let i = 0; i < grid.length; i++)
  {
    grid[i].show()
  }
  current.visited = true
  current.highlight()
  let next = current.checkNeighbours()
  if (next)
  {
    next.visited = true
    stack.push(current)
    removeWalls(current, next)
    current = next
  }
  else if (stack.length > 0)
  {
    current = stack.pop()
  }
  moved()
  noStroke()
  fill(0, 255, 0, 100)
  square(x, y, w)
  fill(255, 0, 0, 100)
  square(width - w, height - w, w)
}
```

```
function index(i, j)
{
  if (i < 0 || j < 0 || i > cols - 1 || j > rows - 1)
  {
    return -1
  }
  return i + j * cols
}

function removeWalls(a, b)
{
  let x = a.i - b.i
  if (x == 1)
  {
    a.walls[3] = false
    b.walls[1] = false
  }
  else if (x == -1)
  {
    a.walls[1] = false
    b.walls[3] = false
  }

  let y = a.j - b.j
  if (y == 1)
  {
    a.walls[0] = false
    b.walls[2] = false
  }
  else if (y == -1)
  {
    a.walls[2] = false
    b.walls[0] = false
```

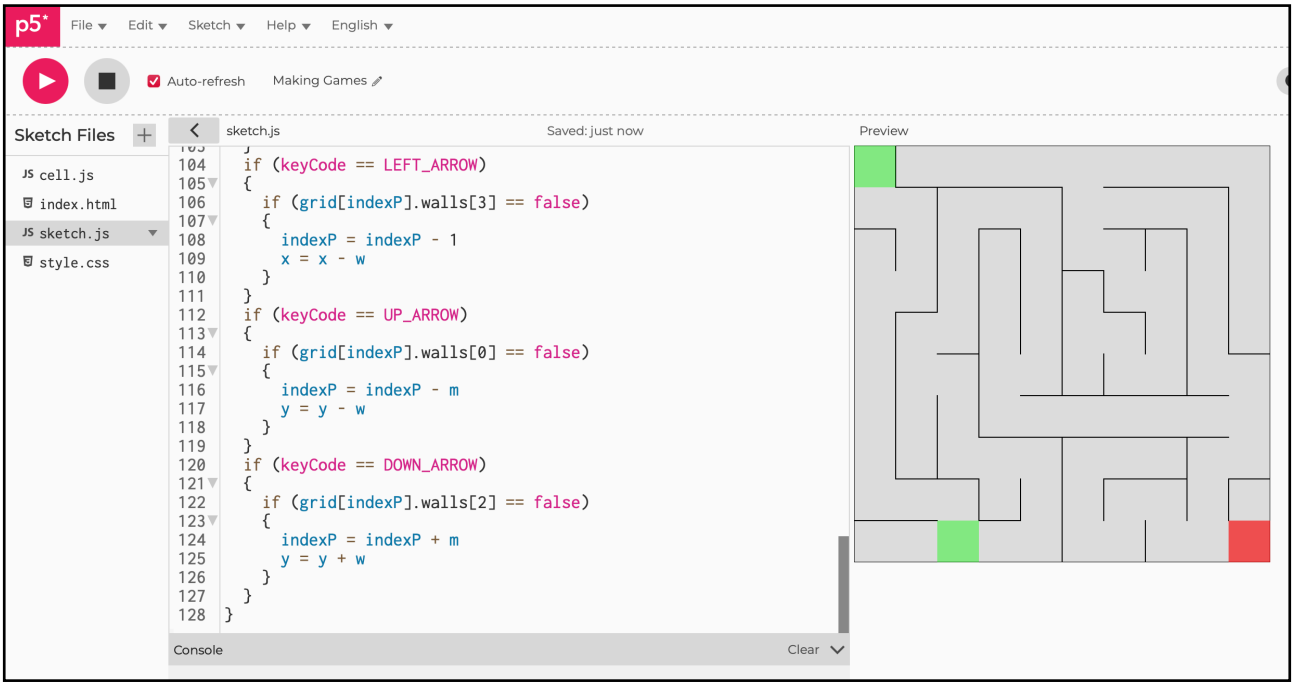
```
  }
}

function moved()
{
  if (keyIsPressed)
  {
    frameRate(10)
    if (keyCode == RIGHT_ARROW)
    {
      if (grid[indexP].walls[1] == false)
      {
        indexP = indexP + 1
        x = x + w
      }
    }
    if (keyCode == LEFT_ARROW)
    {
      if (grid[indexP].walls[3] == false)
      {
        indexP = indexP - 1
        x = x - w
      }
    }
    if (keyCode == UP_ARROW)
    {
      if (grid[indexP].walls[0] == false)
      {
        indexP = indexP - m
        y = y - w
      }
    }
    if (keyCode == DOWN_ARROW)
```

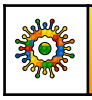
```
    {
      if (grid[indexP].walls[2] == false)
      {
        indexP = indexP + m
        y = y + w
      }
    }
  }
}
```

🗒️ Notes

Click on the canvas and use the arrow keys to navigate the maze, one click at a time. We will smooth this later.

Figure A6.23


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Making Games
Sketch Files
cell.js
index.html
sketch.js
style.css
sketch.js
Saved: just now
Preview
104 if (keyCode == LEFT_ARROW)
105 {
106 if (grid[indexP].walls[3] == false)
107 {
108 indexP = indexP - 1
109 x = x - w
110 }
111 }
112 if (keyCode == UP_ARROW)
113 {
114 if (grid[indexP].walls[0] == false)
115 {
116 indexP = indexP - m
117 y = y - w
118 }
119 }
120 if (keyCode == DOWN_ARROW)
121 {
122 if (grid[indexP].walls[2] == false)
123 {
124 indexP = indexP + m
125 y = y + w
126 }
127 }
128 }
Console
Clear



Sketch A6.24 mover and finisher

Adding two circles, one will be you and the other the finishing point.

sketch.js

```
let cols
let rows
let w = 40
let grid = []
let current
let stack = []
let x = 0
let y = 0
let indexP = 0
let m

function setup()
{
  canvas = createCanvas(400, 400)
  background(255)
  cols = floor(width / w)
  rows = floor(height / w)
  for (let j = 0; j < rows; j++)
  {
    for (let i = 0; i < cols; i++)
    {
      let cell = new Cell(i, j)
      grid.push(cell)
    }
  }
  current = grid[0]
  m = width/w
```

```
}

function draw()
{
  background(220)
  for (let i = 0; i < grid.length; i++)
  {
    grid[i].show()
  }
  current.visited = true
  current.highlight()
  let next = current.checkNeighbours()
  if (next)
  {
    next.visited = true
    stack.push(current)
    removeWalls(current, next)
    current = next
  }
  else if (stack.length > 0)
  {
    current = stack.pop()
  }
  if (stack.length == 0)
  {
    move()
    fill(0, 128, 128)
    stroke(0)
    circle(x + w/2, y + w/2, w/2)
    fill(150, 0, 0)
    circle(width - w/2, height - w/2, w/2)
  }
}
```

```
function index(i, j)
{
  if (i < 0 || j < 0 || i > cols - 1 || j > rows - 1)
  {
    return -1
  }
  return i + j * cols
}

function removeWalls(a, b)
{
  let x = a.i - b.i
  if (x == 1)
  {
    a.walls[3] = false
    b.walls[1] = false
  }
  else if (x == -1)
  {
    a.walls[1] = false
    b.walls[3] = false
  }

  let y = a.j - b.j
  if (y == 1)
  {
    a.walls[0] = false
    b.walls[2] = false
  }
  else if (y == -1)
  {
    a.walls[2] = false
```

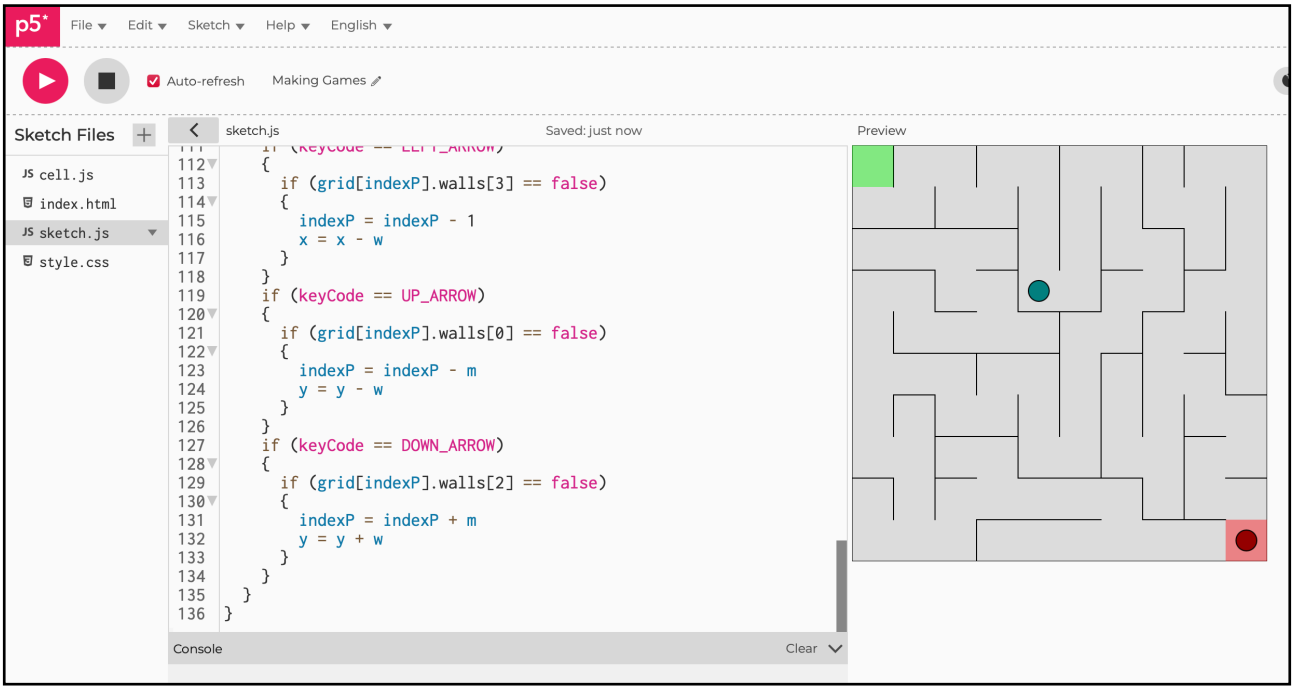
```
    b.walls[0] = false
  }
}

function move()
{
  if (keyIsPressed)
  {
    frameRate(10)
    if (keyCode == RIGHT_ARROW)
    {
      if (grid[indexP].walls[1] == false)
      {
        indexP = indexP + 1
        x = x + w
      }
    }
    if (keyCode == LEFT_ARROW)
    {
      if (grid[indexP].walls[3] == false)
      {
        indexP = indexP - 1
        x = x - w
      }
    }
    if (keyCode == UP_ARROW)
    {
      if (grid[indexP].walls[0] == false)
      {
        indexP = indexP - m
        y = y - w
      }
    }
```

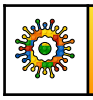
```
    if (keyCode == DOWN_ARROW)
    {
      if (grid[indexP].walls[2] == false)
      {
        indexP = indexP + m
        y = y + w
      }
    }
  }
}
```

🗒️ Notes

The squares are still there.

Figure A6.24


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Making Games
Sketch Files
cell.js
index.html
sketch.js
style.css
sketch.js
Saved: just now
Preview
112 {
113 if (grid[indexP].walls[3] == false)
114 {
115 indexP = indexP - 1
116 x = x - w
117 }
118 }
119 if (keyCode == UP_ARROW)
120 {
121 if (grid[indexP].walls[0] == false)
122 {
123 indexP = indexP - m
124 y = y - w
125 }
126 }
127 if (keyCode == DOWN_ARROW)
128 {
129 if (grid[indexP].walls[2] == false)
130 {
131 indexP = indexP + m
132 y = y + w
133 }
134 }
135 }
136 }
Console
Clear



Sketch A6.25 a tweak

❗ cell.js

Removing the squares.

cell.js

```
class Cell
{
  constructor(i, j)
  {
    this.i = i
    this.j = j
    this.walls = [true, true, true, true]
    this.visited = false
  }

  checkNeighbours()
  {
    let neighbours = []
    let top = grid[index(this.i, this.j - 1)]
    let right = grid[index(this.i + 1, this.j)]
    let bottom = grid[index(this.i, this.j + 1)]
    let left = grid[index(this.i - 1, this.j)]

    if (top && !top.visited)
    {
      neighbours.push(top)
    }
    if (right && !right.visited)
    {
      neighbours.push(right)
    }
```

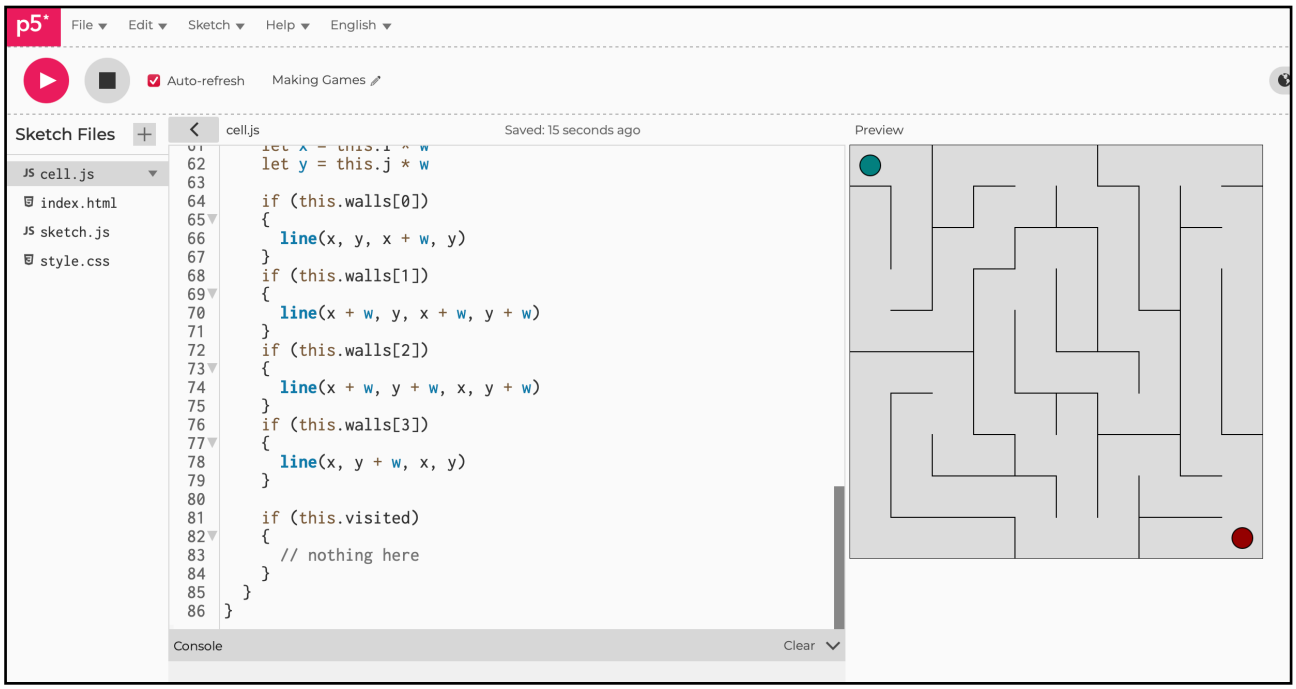
```
    if (bottom && !bottom.visited)
    {
      neighbours.push(bottom)
    }
    if (left && !left.visited)
    {
      neighbours.push(left)
    }

    if (neighbours.length > 0)
    {
      let r = floor(random(0, neighbours.length))
      return neighbours[r]
    }
    else
    {
      return undefined
    }
  }

  highlight()
  {
    let x = this.i * w
    let y = this.j * w
    // noStroke()
    // fill(0, 255, 0, 100)
    // square(x, y, w)
    // fill(255, 0, 0, 100)
    // square(width - w, height - w, w)
  }

  show()
  {
```

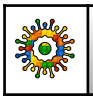
```
    stroke(0)
    let x = this.i * w
    let y = this.j * w

    if (this.walls[0])
    {
      line(x, y, x + w, y)
    }
    if (this.walls[1])
    {
      line(x + w, y, x + w, y + w)
    }
    if (this.walls[2])
    {
      line(x + w, y + w, x, y + w)
    }
    if (this.walls[3])
    {
      line(x, y + w, x, y)
    }

    if (this.visited)
    {
      // nothing here
    }
  }
}
```

📝 Notes

Click on the canvas first, then use the arrow keys to navigate.

Figure A6.25


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Making Games
Sketch Files
cell.js
index.html
sketch.js
style.css
cell.js
Saved: 15 seconds ago
Preview
let y = this.j * w
if (this.walls[0])
{
line(x, y, x + w, y)
}
if (this.walls[1])
{
line(x + w, y, x + w, y + w)
}
if (this.walls[2])
{
line(x + w, y + w, x, y + w)
}
if (this.walls[3])
{
line(x, y + w, x, y)
}
if (this.visited)
{
// nothing here
}
}
}
Console
Clear



Next?

Suggestions:

1. Make more of a game out of it.
2. Count the number of steps.
3. Measure the time taken.
4. Automatically load a new maze once it is completed.
5. Increase the size and complexity of the maze.

These mazes are not that complicated or difficult, but one reason for including this is for the possibility of using it to develop an AI to navigate the maze efficiently.