```
// Variables to define the number of columns and rows in the grid
int cols, rows;
// Variable to set the size of each grid cell
int scale = 20;
// Width of the entire flow field (virtual grid)
int w = 1400;
// Height of the entire flow field (virtual grid)
int h = 800;
// Array to store the flow field vectors
PVector[] flowfield;
// ArrayList to store particles that will move around
ArrayList<Particle> particles = new ArrayList<Particle>();

```

1- 13:

Aqui estamos indicando diferentes variables para cuando las llamemos a acción más tarde en el sketch las reconozca. Con "**int cols,rows;**" le indicamos que abran columnas y filas. Con **int scale = 20** le decimos que la scale sera de 20 pixeles. Con **w** le indicamos que el width sera de 1400, y con **h** le indicamos que el Height sera de 800.

Por último, creamos un Array donde guardaremos los vectores (PVector que se llamara **flowfield**. y indicamos un **ArrayList** donde guardaremos las particulas que se moveran por el canvas.

```
// The setup function runs once when the program starts
void setup() {
  // Set the size of the window where the artwork will be displayed
  size(1400, 800);
  // Calculate the number of columns and rows based on the width, height, and scale
  cols = w / scale;
  rows = h / scale;
  // Initialize the flow field array with a size based on the number of grid cells
  flowfield = new PVector[cols * rows];

  // Add 1000 particles to the particles list
  for (int i = 0; i < 9000; i++) {
    particles.add(new Particle());  // Create a new particle and add it to the list
  }
}
```

14-28:

Aqui vamos a crear el setup fuction que empieza cuando el programa empieza. Empezamos por especificar el tamaño del canvas (**size**). Y seguimos calculando el numero de columnas y filas. Como hemos indicado que queremos que las columans y las filas tengan una escala de 20, lo que haremos es indicar **cols = w / scale**; (es lo mismo que decir la columna = 1400/20 y así sabremos el numero de columnas) y hacemos lo mismo con las filas **rows = h / scale** (las filas = 800/20 = al numero de filas).

Después iniciamos el **flowfield** en base al numero de casillas que tiene la malla. y añadimos 9000 particulas a la **ArrayList** que le hemos llamado **Particles**.

```
31  void draw() {
32    // Set the background color to black
33    background(0);
34
35    // Variable to track the vertical noise offset
36    float yoff = 0;
37
38    // Loop through all rows in the grid
39    for (int y = 0; y < rows; y++) {
40      // Variable to track the horizontal noise offset
41      float xoff = 0;
42      // Loop through all columns in the grid
43      for (int x = 0; x < cols; x++) {
44        // Generate a random angle using Perlin noise
45        float angle = noise(xoff, yoff) * TWO_PI * 4;
46        // Calculate the index for the flowfield array
47        int index = x + y * cols;
48        // Create a vector pointing in the direction of the angle
49        PVector v = PVector.fromAngle(angle);
50        // Store this vector in the flowfield array
51        flowfield[index] = v;
52        // Increment the horizontal noise offset
53        xoff += 0.1;
54      }
55      // Increment the vertical noise offset
56      yoff += 0.1;
57    }
```

31-57:

Ahora empezamos la función del void draw. Empezamos por especificar el color del background que va a ser negro (0).

Y indicamos la variable para traquear el vertical noise offset que empezará en el numero 0. Esto lo haremos para crear todos los vectores que se almacenaran dentro del Array. Para esto crearemos un for.
for (int y = 0; y < rows; y++) **int y = 0;** indica que la variable y tiene un valor de 0. **y < rows;** indica que cuando el valor de y sea inferior al de las filas (es decir 40) se aplicara la siguiente función **y++.**

Dentro de este **for** tenemos más funciónes que se inician cuando este sucede. primero indicamos la variable **xoff** que es igual a 0. **for (int x = 0; x < cols; x++)** en este caso hacemos lo mismo que con la y. Ahora lo que hacemos es indicar que significa **angle**, y para saber que es angle generamos un angulo random usando el **Perlin noise** (*(Perlin noise is a random sequence generator producing a more natural, harmonic succession of numbers than that of the standard random() function.))* Lo que hacemos con este noise es indicarle si trabajara en 1d, 2d o 3d, en este caso en 2d usando las variables **xoff** y **yoff**, esto lo multiplicamos por **TWO_PI** y luego *** 4** (numero que hemos elegido nosotros). A continuación indicamos que es **index**, este es el calculo del indice para el flowfield array. **int index = x + y * cols.** Ahora creamos un vector en la dirección del angulo que hemos creado anres. Para hacer esto, indicamos que **v = PVector.fromAngle (Angle).** Ahora le indicamos que este vector quede guardado en el **flowfield** (que es el array que hemos creado antes de vectores). Por último antes de cerrar este **for** indicamos que cada vez que aumente será de **+0.1** haciendo referencia a **xoff**. Y después aun nos falta cerrar el **for** de la **y**, y incrementamos también en **+0.1 la yoff**.

```
59    // Loop through all particles and update them
60    for (Particle p : particles) {
61      // Make the particle follow the flow field
62      p.follow(flowfield);
63      // Update the particle's position
64      p.update();
65      // Draw the particle on the screen
66      p.show();
67      // Check if the particle has reached the edges of the screen
68      p.edges();
69    }
70  }
```

59-70:

En esras lineas de codigo se indica el dibujo que se hace en los visuales, pero esta haciendo un llamamiento a otras funciones que se han especificado después. Así que para entender las siguientes lineas primero analizaremos las otras funciones.

```
72  // Particle class defines the properties and behaviors of each particle
73  class Particle {
74    // Variables for position, velocity, and acceleration
75    PVector pos, vel, acc;
76    // Variable to set the maximum speed of the particle
77    float maxspeed = 2;
78
```

72-78:

class Particle {} define las propiedades y comportameinto de cada particula. Y empezamos indicando las **variables** de posición (**pos**), de velocidad (**vel**) y de aceleración (**acc**). Y añadimos un **float** (que significa que el numero puede ser decimal) que indicara un **máximo de velocidad**, en este caso **2**. **PVector** es un termino que significa que contiene dos valores uno para la (**x,y**).

```
79   // Constructor that initializes the particle with a random position
80   Particle() {
81     // Set the initial position to a random point on the screen
82     pos = new PVector(random(width), random(height));
83     // Set the initial velocity to zero
84     vel = new PVector(0, 0);
85     // Set the initial acceleration to zero
86     acc = new PVector(0, 0);
87   }
```

79-87:

Particle (){} es un constructor (es la parte de la receta que empieza a hacer cosas). En este caso es donde el ordenador crea nuevas particulas y decide donde empezarán de la pantalla. Cuando creas una particula este codigo empieza autonmáticamente.

pos = new PVector(random(width), random(height)); esta linea indica el punto de inicio de la particula. **random**() es una función que escoge un numero random, en este caso lo hace eligiendo una posicón random del **width** y del **height**.

vel = new PVector(0, 0); Esta linea indica la velocidad inicial de la particula, en este caso en 0.
acc = new PVector(0, 0);En esta linea se especifica la aceleración inicial, en este caso en 0, es decir que no aumentará o reducirá la velocidad cuando empiece.

Para resumir estas lineas de codigo:

- Este codigo crea unas particulas con una posición random de punto de inicio.
- Al inicio esta particula no se mueve ni cambia la velocidad.
- Esta particula tiene un limite de velocidad que no puede superar, en este caso 2.

```
89   // Function to make the particle follow the flow field
90   void follow(PVector[] vectors) {
91     // Calculate the column and row based on the particle's position
92     int x = int(pos.x / scale);
93     int y = int(pos.y / scale);
94
```

89-94:

Este código es una función que le dice a una "partícula" cómo debe moverse siguiendo un "campo de flujo."

void follow(PVector[ ] vectors) {} **follow** es el nombre de la función que indica seguir, se encarga de hacer que las particulas sigan las direcciones del campo de flujo. **PVector[ ] vectors** es una lista de vectores (direcciones) que forman un **campo de flujo**. Un **vector**, es simplemente una **dirección con una magnitud** (longitud), indicando hacía dónde y con qué fuerza debe moverse la particula.

int x = int(pos.x / scale); , int y = int(pos.y / scale); pos.x y **pos.y** son las coordenadas de la partícula. **scale** es el tamaño de cada casilla en esta cuadricula. Estas lineas de código **calculan en qué columna** (**x**) **y en que fila** (**y**) de la cuadrícula de encuentra la partícula.

```
95      // Ensure the column and row values stay within the bounds of the
96      x = constrain(x, 0, cols - 1);
97      y = constrain(y, 0, rows - 1);
98
99      // Calculate the index for the flow field array
100     int index = x + y * cols;
101     // Get the vector from the flow field at the calculated index
102     PVector force = vectors[index];
103     // Apply the vector as a force to the particle
104     applyForce(force);
105   }
```

95-105:

x = constrain(x, 0, cols - 1);, y = constrain(y, 0, rows - 1); Constrain significa **limitar** o restringir. Con estas lineas nos aseguramos de que la **columna (x) y (y)** esten **dentro de los límites del campo de flujo**. Y para hacer eso usamos los cols y rows.

int index = x + y * cols; indice es un número de referencia para ubicar algo en una lista. Esta línea **convierte** las coordenadas de la cuadrícula (columna y fila) en un numero que puede usar para **encontrar la dirección correcta en la lista de vectores** del campo de flujo.

PVector force = vectors[index]; Ahora que tenemos el **índice**, usamos ese número para obtener el **vector** (la dirección y magnitud) del campo de flujo en esa **posición específica. force** es el nombre que le damos a este vector que la particula va a seguir.

applyForce(force); aplica esa fuerza haciendo que se mueva en la dirección que indica el vector.

Para resumir estas líneas de código:

- Toma la posición de la partícula.
- Encuentra en que parte del campo de flujo se encuentra.
- Y le indica que dirección tiene que seguir, esto hace que parezca que la particula esta fluyendo por la pantalla.

```
107   // Function to apply a force to the particle
108   void applyForce(PVector force) {
109     // Add the force to the acceleration of the particle
110     acc.add(force);
111   }
```

107-111:

Este código le dice a la particula cómo debe responder cuando se le aplica una fuerza. **force** es algo que hace que las partículas se muevan o cambie su movimiento.

void applyForce(PVector force) {} **applyForce** es el nombre de la función. esta función se encarga de recibir una fuerza y aplicársela a la partícula para que su movimiento cambie. **PVector force** es la fuerza que se va a aplicar.

acc.add(force); **acc** es la aceleración de la particula, controla cómo su velocidad cambia en el tiempo. **add(force)** es sumar/ añadir la aceleración.

Para resumir estas líneas de código:

- Esta es la función que aplica una fuerza a una particula.
- La fuerza se suma a la aceleración.
- Cada vez que se llama a esta función, la particula recibe un "empujón" en la dirección y con la intensidad que le diga el vector de fuerza.

```
113   // Function to update the particle's position
114   void update() {
115     // Add the acceleration to the velocity
116     vel.add(acc);
117     // Limit the velocity to the maximum speed
118     vel.limit(maxspeed);
```

113-123:

Este código es fundamental por que controla cómo la partíicula se mueve en la pantalla. Es el "motor" que le indica que le dice a la partícula qué hacer en cada momento. Este código actualiza la posición de la partícula en cada cuadro de animación. Y prepara la partícula para el próximo movimiento.

void update() {} esta función se encarga de calcular el nuevo movimiento de la partícula.

vel.add(acc); **vel** es la velocidad de la partícula. **acc** es la aceleración que controla cómo cambia la velocidad.

vel.limit(maxspeed); esta línea se asegura de que la partícula no se mueva muy rápido. **maxspeed** es el límite de velocidad máxima. **limit** es como poner un tope de velocidad, si la partícula intenta ir más rapido que **maxspeed**, esta línea la frena.

```
119      // Add the velocity to the position (move the particle)
120      pos.add(vel);
121      // Reset the acceleration to zero for the next frame
122      acc.mult(0);
123    }
```

113-123:

pos.add(vel); Aquí es donde la partícula realmente se mueve. **pos** es la posición actual de la partícula. Al sumar la **vel** a la posición, la partícula cambia de lugar según qué tan rápido se esté moviendo.

acc.mult(0); Después de mover la partícula es necesario restablecer la aceleración para el próximo cuaro de animación. mult(0) lo que hace es que al multiplicarla por 0 la elimina o reinicia a 0. Sino la aceleración se acumularia y aceleraría sin control.

Para resumir estas líneas de código:

- Esta función actualiza la posición de la partícula en la pantalla.
- Primero, ajusta la velocidad de la partícula según la aceleración.
- Segundo, limita esa velocidad para que no vaya demasiado rápido.
- Tercero, usa la velocidad para mover la partícula a su nueva posición.
- Cuarto, reinicia la aceleración a = para prepararse para el próximo cuadro de la animación.

```
125    // Function to display the particle on the screen
126    void show() {
127      // Set the color of the particle (white with some transparency)
128      stroke(255, 50);
129      // Set the thickness of the particle's stroke
130      strokeWeight(2);
131      // Draw a point at the particle's current position
132      point(pos.x, pos.y);
133    }
```

125-133:

Este código se encarga de mostrar la partícula.
void show() {} **show** es el nombre de la función, hace que aparezca en la pantlla para que podamos verla.
stroke(255, 50); **stroke** se refiere al color. **255** indica el color y **50** la opacidad.
strokeWeight(2); **strokeWeight** establece el grosor del contorno de la partícula. **2** es el valor.
point(pos.x, pos.y);point es la función que dibuja el punto en la pantalla. pos.x y pos.y son las coordenadas actuales de la partícula.

Para resumir estas líneas de código:

- Esta función se encarga de hacer visible la partícula.
- Primero, establece el color de la partícula y opacidad.
- Segundo, define el grosor del contorno de la partícula.
- Finalmente, dibuja el punto en su posición.

```
135   // Function to wrap the particle around the edges of the screen
136   void edges() {
137     // If the particle goes off the right edge, move it to the left edge
138     if (pos.x > width) pos.x = 0;
139     // If the particle goes off the left edge, move it to the right edge
140     if (pos.x < 0) pos.x = width;
141     // If the particle goes off the bottom edge, move it to the top edge
142     if (pos.y > height) pos.y = 0;
143     // If the particle goes off the top edge, move it to the bottom edge
144     if (pos.y < 0) pos.y = height;
145   }
146 }
```

135-146:

Este código es una función que se encarga de envolver a la partícula alrededor de los bordes de la pantalla. Es decir, cuando la partícula sale de un lado de la pantalla, automáticamente reaparece en el lado opuesto (como si creara un efecto de teletransportación).

void edges() {} **edges** es el nombre de la función. Se encarga de controlar lo que sucede cuando la partícula llega a los bordes de la pantalla.

if (pos.x > width) pos.x = 0; **if** es una condición que verifica si algo es verdadero. En este caso verifica si la partícula ha salido del borde derecho. **pos.x > width**: **pos.x** es la posición horizontal de la partícula **width** es el ancho de la pantalla. Si **pos.x** es mayor que **width**, significa que la partícula ha salido del borde derecho de la pantalla. **pos.x = 0**; Si la partícula se sale por el borde derecho, esta línea la mueve al borde iquierdo estableciendo en posición 0.

if (pos.x < 0) pos.x = width; Esta linea verifica si la partícula ha salido por el borde izquierdo de la pantalla.

pos.x < 0 si la posición horizontal de la partícula es menor que 0, significa que ha salido por el borde izquierdo. pos.x = width; En este caso la partícula se mueve al borde derecho de la pantalla (a la posición de width).

if (pos.y > height) pos.y = 0; Aqui se verifica si la partícula ha salido por el borde inferior de la pantalla. pos.y < height entonces pos.y es la posición vertical de la partícula. height es la altura de la pantalla. Si pos.y es mayor que height significa que la partícula ha salido por el borde inferior. pos.y = 0: en este caso la partícula se mueve al borde superior de la pantalla, estableciendo su posicion en 0.

if (pos.y < 0) pos.y = height; (al revés con esta)

Para resumir estas líneas de código:

- Esta función asegura que la partícula nunca salga de la pantalla, y que reaparezca al lado inverso.