Curso de Computación: Codificando con JavaScript 2ª parte

Tariq Rashid
Traducción y adaptación de Mario del Solar Moraga[1] Autores

[1] *Profesor del Colegio Homeduca de Colina.*

21 de abril de 2024

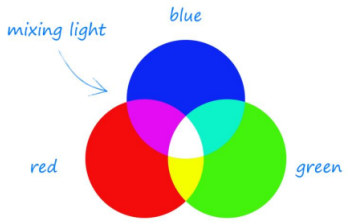

Figura 1: *Los colores básicos son verde, azul y rojo.*

Esta es la segunda parte del curso. Recuerda abrir el editor de código desde el sitio web `www.codeguppy.com` .

1. Mezclando Colores

Qué haremos
En este proyecto vamos a:

- aprender a mezclar nuestros propios colores
- ver cómo podemos calcular los colores

Ya hemos elegido colores usando nombres como rosa y naranja. Ahora aprenderemos a mezclar nuestros propios colores. Esto es como mezclar pintura para conseguir el color que queremos. Mezclando luz roja, verde y azul es como vemos los colores en las pantallas de nuestros televisores, portátiles y teléfonos inteligentes.
Puedes adquirir algo de experiencia creando colores en el siguiente sitio web: `https://www.w3schools.com/colors/colors_rgb.asp`

2. Comenzamos con un círculo

Entra a codeguppy, ve al botón verde que dice CODE NOW y escribe:

```
fill('yellow');
circle(400, 300, 200);
```

Escojamos ese mismo amarillo usando los niveles rojo, verde y azul.

```
fill(255, 255, 0);
circle(400, 300, 200);
```

Intenta mezclar diferentes niveles de rojo, verde y azul para cambiar el color del círculo. Recuerde mantener los niveles entre 0 y 255.

2.1. Mezclamos colores aleatoriamente

Los niveles RGB para cualquier color son números del 0 al 255. ¿Y si elegimos esos números al azar? Eso significaría mezclar cantidades aleatorias de luz roja, verde y azul para crear un color.

```
var r = randomNumber(0, 255);
var g = randomNumber(0, 255);
var b = randomNumber(0, 255);
fill(r, g, b);
circle(400, 300, 200);
```

Puede ver que estamos configurando las variables r, g y b para que sean números aleatorios entre 0 y 255. Estamos usando esos números en la instrucción fill() para establecer un color. Ese color lo usaremos para rellenar el círculo que dibujamos. No sabemos de qué color será el círculo porque estará mezclado niveles RGB aleatorios.
¿Qué color obtienes? Ejecute su código nuevamente para obtener un color diferente.

2.2. Muchísimos colores

Escribamos una función para dibujar un círculo de radio 50 en un lugar aleatorio del lienzo. Usaremos nuestro código anterior para darle un color aleatorio. También usaremos la instrucción repetir() para llamar a esta función 50 veces y dibujar 50 círculos. Aquí hay un código que hace esto.

```
repeat(50, balloon);
function balloon()
{
var r = randomNumber(0, 255);
var g = randomNumber(0, 255);
var b = randomNumber(0, 255);
var x = randomNumber(100, 700);
var y = randomNumber(100, 500);
fill(r, g, b);
circle(x, y, 50);
}
```

Llamé a la función *balloon*. Puedes ver que elegimos números aleatorios entre 0 y 255 para la luz roja, verde y azul.

2.3. Cambiando el rango

Hemos estado creando colores mezclando niveles aleatorios de rojo, verde y azul. Esos niveles eran un número entre 0 y 255. ¿Y si elegimos los niveles RGB de un rango diferente?

```
var r = randomNumber(100, 255);
var g = randomNumber(100, 255);
var b = randomNumber(0, 10);
```

Los niveles rojo y verde se eligen entre 100 y 255. El nivel azul se elige entre 0 y 10.

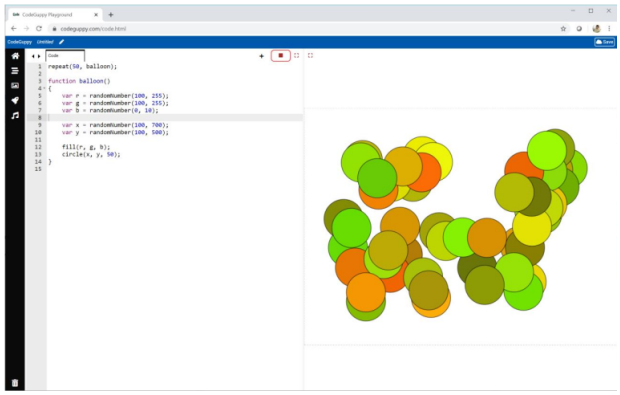

Figura 2: *Caption*

Probemos algunos rangos diferentes:

```
var r = 0;
var g = randomNumber(100, 255);
var b = randomNumber(100, 255);
```

Esta vez el nivel rojo está configurado para que siempre sea 0. Los niveles verde y azul pueden estar entre 100 y 255. El resultado es un bonito conjunto de verdes y azules que me recuerdan al mar. Ejecútalo y muestra el resultado al profesor.

2.4. Desafío

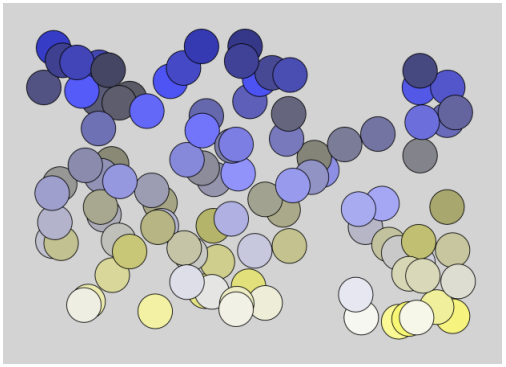

Figura 3: *¿Còmo hacer este cambio de colores según altura en la tela?*

Las coordenadas (x, y) de cada círculo se utilizan para calcular los niveles rojo, verde y azul del color de ese círculo.

- Los niveles rojo y verde son la coordenada *y* dividida por 2.
- El nivel azul es un número aleatorio entre 100 y 255. Escribe tu propio código para hacer esto. Pruebe también sus propios cálculos de color.

3. Más repeticiones o bucles

En este proyecto vamos a:

- obtener información sobre los contadores de bucles
- ver cómo los contadores de bucle pueden resultar útiles como parámetros de función

3.1. Muchísimos globos

Echa un vistazo a la figura 4 con la imagen de cuatro globos pequeños. El código que hizo este dibujo es simple.

```
circle(100, 300, 25);
circle(150, 300, 25);
circle(200, 300, 25);
circle(250, 300, 25);
```

¿Puedes ver qué cambia entre cada instrucción circular? La coordenada x es 100 para el primer círculo. Para el segundo círculo es 150. Sigue creciendo en 50 hasta llegar a 250 para el cuarto círculo.

¿Y si quisiéramos dibujar 13 globos? Podríamos escribir 13 instrucciones de *circle*. Sería mucho escribir y muy aburrido. ¡Debe haber una mejor manera! ¿Podría la repetición de instrucciones ayudarnos a evitar escribir mucho?

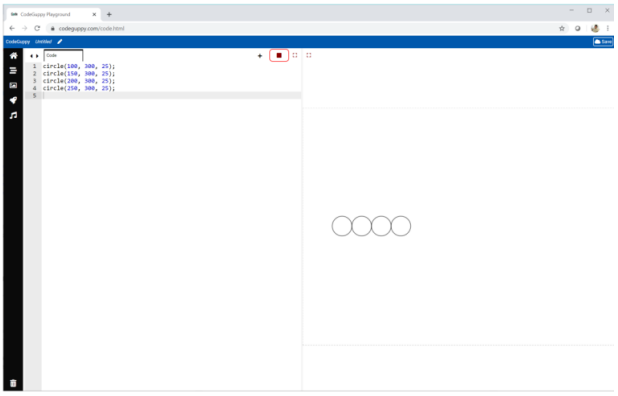

Figura 4: *Cuatro círculos en línea.*

repeat(13, balloon); Es una buena idea, pero este código no funcionará porque la función *balloon* no sabrá dónde dibujar cada círculo. Si repetir pudiera pasar información a *balloon*, eso realmente ayudaría: podría indicarle a la función *balloon* dónde dibujar cada círculo.

3.2. Repetir con extrapoderes!

Eche un vistazo a este código que muestra una nueva forma de utilizar la repetición:
repeat(100, 700, 50, balloon);
Aquí, la instrucción de repetición mantiene un contador. Este contador de bucle comienza en 100 y llega hasta 700, aumentando en 50 cada vez. A la función *balloon* se le pasa el contador como parámetro. Entonces, la instrucción de repetición llama a *balloon*(100), luego a *balloon*(150), luego a *balloon*(200), a *balloon*(250),... hasta llegar a *balloon*(700). Esto es realmente útil porque la función *balloon* puede usar el número como coordenada x del círculo.
La siguiente figura ilustra lo anterior:

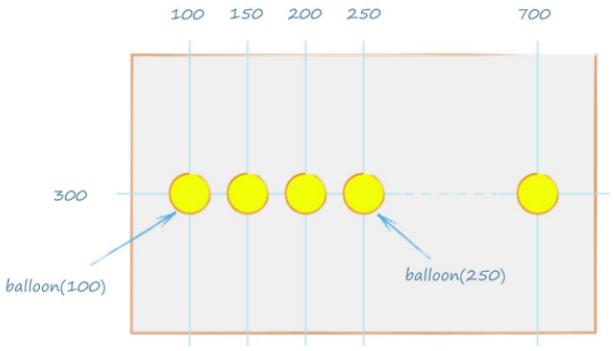

Figura 5: *Usamos la función balloon para ubicar los círculos en la tela.*

Necesitamos escribir nuestra función *balloon* para aceptar un solo parámetro.
function balloon(x) {
circle(x, 300, 25);
}

Puede ver que la función *balloon* toma un parámetro, que hemos llamado x. La función dibuja un círculo en las coordenadas (x, 300), con radio 25. Así es como se ve mi código.
repeat(100, 700, 50, balloon);
function balloon(x) {
circle(x, 300, 25);
}
Puedes ver que la función balloon(x) es realmente simple. Solo contiene una instrucción para *circle*. Fuera de la función, solo hay una instrucción repetida. He aquí los resultados:

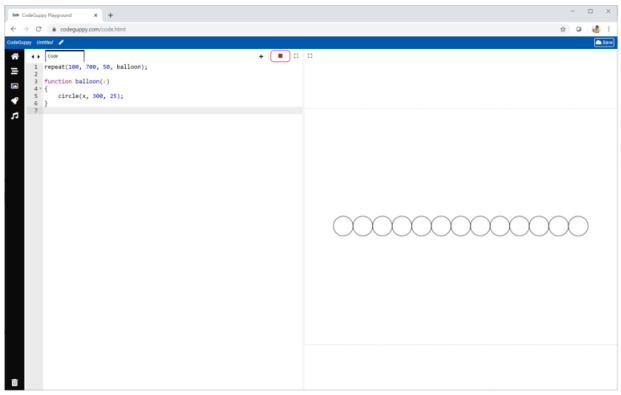

Figura 6: *Muchos círculos repetidos usando una función.*

¡Eso funciono! Esa fue una pequeña cantidad de código para crear estos 13 globos. Pasar información a código repetido como este es una idea realmente poderosa. Se utiliza en casi todas partes: en la creación de juegos, arte digital, música electrónica, control de robots y también aplicaciones para tu teléfono inteligente. ¡Es bueno aprender y practicar!

4. Inténtalo por ti mismo

Intentemos usar el parámetro x de la función balloon(x) para decidir el tamaño del círculo. Aquí está mi propio experimento:
circle(x, 300, x/20);
Escribe el código, ejecuta el comando y muestra al profesor tus resultados.

4.1. Cambiando el parámetro color

Esto es lo que probé:
function balloon(x) {
fill(x/2, x/4, 128);
circle(x, 300, x/20);
}
El color se mezcla con un nivel de rojo de x/2, un nivel de verde de x/4 y un nivel de azul establecido en 128. ¡Los resultados son bastante interesantes!

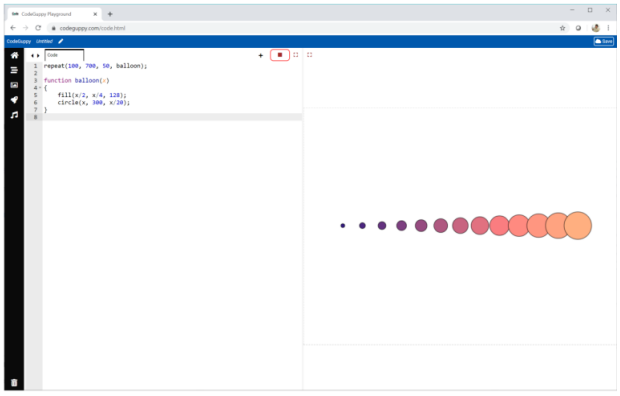

Figura 7: *Diferentes colores y tamaños en variación regular.*

5. Matemática Artística

En este proyecto vamos a:

- aprender sobre una función matemática simple: la onda sinusoidal
- aprender sobre una función matemática simple: la onda sinusoidal

5.1. Dibujando funciones matemáticas

En la escuela a veces trabajamos con funciones matemáticas como esta: $\mathbf{y = x + 3}$. Si $\mathbf{x = 1}$ entonces $\mathbf{y = 4}$ y si $\mathbf{x = 2}$ entonces $\mathbf{y = 3}$. Esta tabla muestra y cuando x va de 0 a 5: Si dibujamos puntos en todos estos

x	y
0	3
1	4
2	5
3	6
4	7
5	8

Cuadro 1: *Tabla de valores de* x *e* y

(x,y) podríamos ver un patrón. Eche un vistazo a este código. Nada de esto es nuevo, pero hablaremos de ello a continuación.

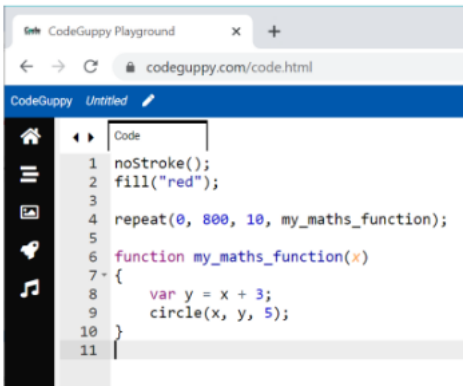

Figura 8: *Caption*

Veamos primero my_maths_function.

```
function my_maths_function(x) {
var y = x + 3;
circle(x, y, 5);
}
```

Toma un parámetro x y lo usa para calcular $x+3$. La respuesta se pone en una variable y. Luego se dibuja un pequeño círculo en (x, y). Entonces my_maths_function está haciendo lo que está haciendo $y = x + 3$. Queremos pasar a my_maths_function diferentes valores de x.

Podemos usar la instrucción de repetición para iniciar un contador en 0 y seguir incrementándolo en 10, hasta llegar a 800. Estos pueden ser todos los valores de x pasados a my_maths_function.

```
repeat(0, 800, 10, my_maths_function);
```

Ejecute el código para ver qué patrón forman esos... círculos?.

¿Qué ocurrió? A medida que x se hace más grande y se mueve hacia la derecha, y también se hace más grande y se mueve hacia abajo en el lienzo. Por eso hay una línea de puntos que se mueve hacia la derecha y hacia abajo. Este patrón es algo aburrido. Probemos con una función matemática diferente para ver si forma un patrón más interesante: $y = (x/20)^2$

Esta vez x se divide por 20 y la respuesta se eleva al cuadrado. Solo es necesario cambiar una línea de código en my_maths_function.

```
var y = sq( x / 20 );
```

Inténtalo y muestra tus resultados al profesor.

El camino parece una pelota lanzada desde una pared. Comienza a caer lentamente y luego se acelera. Probemos una función matemática más: $y = sen(x)$

Ese sen es la abreviatura de seno. Quizás lo hayas visto en tu clase de matemáticas. No te preocupes si no lo has visto antes, ¡solo lo usaremos para divertirnos, no para trabajar! Cambia tu my_maths_function de esta manera:

```
var y = sin( x );
```

Ejecute su código para ver qué patrón crea esta función sinusoidal.

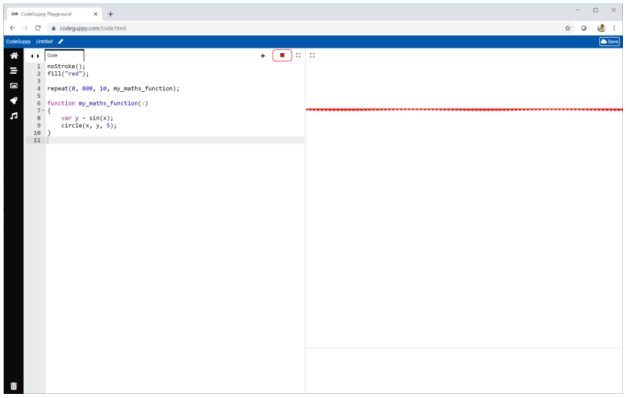

Figura 9: *Resultado del código con función seno.*

¿Qué pasó? Si miras de cerca, puedes ver que algo está

sucediendo en la parte superior del lienzo. ¿Quizás el patrón es muy corto? Podemos hacer el patrón más alto multiplicando sen(x) por 50.

```
var y = 50 * sin( x );
```

También cambiemos el patrón hacia abajo para que cuando y sea 0, los puntos se dibujen en el medio del lienzo. Esto es fácil de hacer. Simplemente sumamos 300 a la coordenada y cuando dibujamos los círculos.

```
circle(x, 300 + y, 5);
```

Su código y salida deberían verse así:

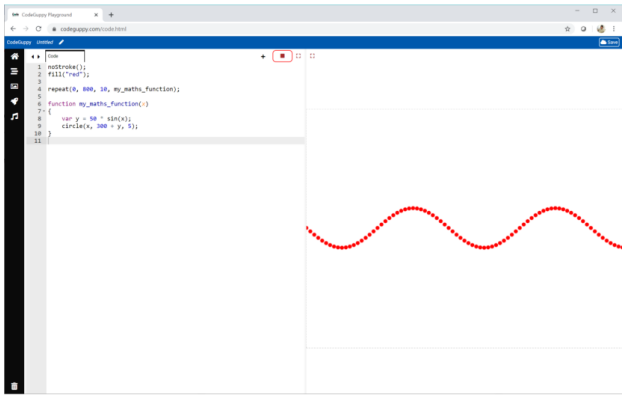

Figura 10: *Resultado del código con función seno.*

5.2. Inténtalo por ti mismo

Eche un vistazo al código my_maths_function nuevamente.

```
var y = 50 * sen( x );
```

Intente cambiar el número 50, lo que hace que la función seno sea más alta. También intente multiplicar x por un número diferente para hacerlo más grande.

5.3. Ondas sinusoidales para tamaños de formas

Usemos ondas sinusoidales para decidir el tamaño de las formas. Eche un vistazo a este código my_maths_function.

```
var size = 100 * sin(x);
rectangle(x, 300, 5, size);
```

Sabemos que sen(x) sube y baja como una ola. Multiplicarlo por 100 hace que la ola sea más alta. Hemos puesto este número en una variable llamada tamaño. Luego dibujamos un rectángulo en $(x, 300)$, que se encuentra en el centro del lienzo. Tiene un ancho de 5 y una altura de talla. Su código y salida deberían verse como en la figura 11:

¡Ese es un patrón genial! ¿Por qué los rectángulos van por encima y por debajo de la línea media? Es porque las ondas sinusoidales van por encima y por debajo de cero. Estos valores positivos y negativos significan que las alturas del rectángulo también se vuelven positivas y negativas. Puedes ver una onda

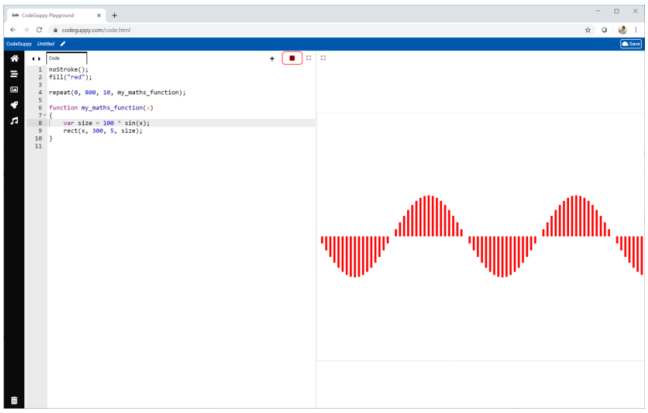

Figura 11: *Ondas senoidales dibujadas por rectángulos delgados.*

sinusoidal que sube y baja entre -1 y +1 aquí: `https://www.desmos.com/calculator/hyimynd5yr`

Prueba diferentes funciones sinusoidales para decidir el tamaño de los rectángulos. Quizás quieras intentar usar dos funciones sinusoidales juntas. Podrías sumarlos. Incluso podrías intentar multiplicarlos. Aquí está mi propio experimento:

```
var size = 100 * sin(x * 3) * sin(x * 0.5);
```

Ejecuta el código y muestra el resultado al profesor.

5.4. Funcione senoidales pueden mezclar colores

Usemos ondas sinusoidales para mezclar colores. Eche un vistazo a este código:

```
var r = 255 * sq(sin(x));
var g = 0;
var b = 128;
fill(r, g, b);
```

La parte verde se establece en 0 y la parte azul se establece en 128. La parte roja depende de qué es x. Sabemos que sen(x) sube y baja como una ola. El código eleva ese número al cuadrado y luego multiplica la respuesta por 255. Esa se convierte en la parte roja del color que estamos mezclando. ¿Por qué hemos elevado al cuadrado la función seno? Esta imagen explica por qué. La onda sinusoidal sube y baja

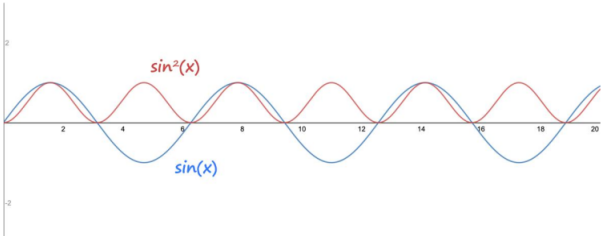

Figura 12: *Ondas senoidales que coinciden en parte.*

entre -1 y +1. Si elevamos los valores al cuadrado, van de 0 a +1. Esto hace que sea más fácil escalar los valores a niveles RGB. Simplemente los multiplicamos

por 255. Usemos ese color para dibujar un rectángulo delgado a lo largo de todo el lienzo.
`rect(x, 0, 10, 600);`
Su código y salida deberían verse así:

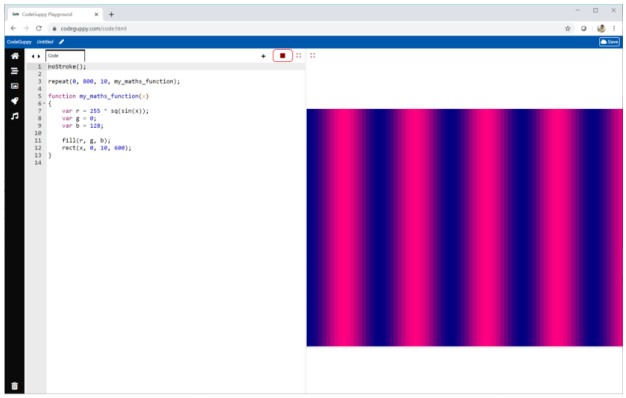

Figura 13: *Bellas imágenes usando la función seno.*

La función seno ondulado ha mezclado una cantidad ondulada de rojo con el color de los rectángulos delgados. Cuando el valor del seno es alto, el nivel de rojo es alto y obtenemos ese color rosa brillante. Cuando el valor del seno es bajo, el nivel de rojo es bajo y obtenemos ese color púrpura más oscuro. He aquí mi propio experimento:

```
var r = 255 * sq(sin(x));
var g = 255 * sq(sin(x * 0.1));
var b = 128;
```

La parte roja cambia más rápido que la parte verde. Modifica el código anterior y muestra tus resultados.

5.5. Desafío

¿Puedes escribir código para crear esta imagen realmente genial?

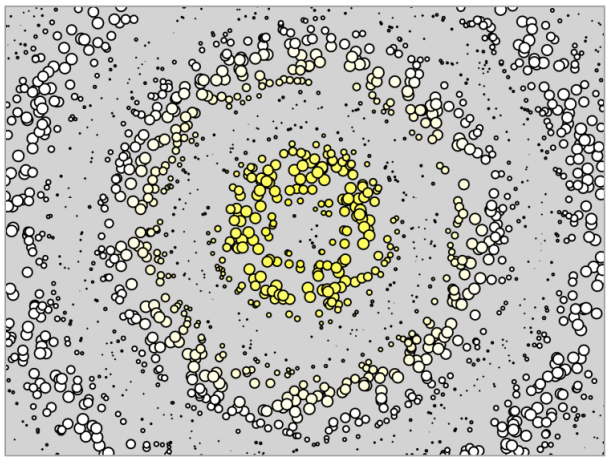

Figura 14: *Caption*

- Hay 1800 puntos colocados en lugares aleatorios del lienzo. Puedes usar un bucle para dibujarlos.
- El tamaño de los puntos crece y se reduce con la distancia desde el centro. Eso significa usar el seno de la distancia.
- El color se mezcla utilizando la distancia desde el centro.

6. Más colores!

En este proyecto vamos a:

- aprender a mezclar colores usando el modelo de color HSB
- ver cómo puede ser más útil que el modelo RGB

6.1. Pensar colores en RGB no es nada fácil...

Elegir un color mezclando luz roja, verde y azul es muy común. Pero no siempre es la forma más fácil. ¿Puedes calcular mentalmente los valores RGB del amarillo? La respuesta es (255, 255, 0). Usando estos números, ¿puedes calcular los valores RGB para un amarillo claro o un amarillo oscuro? No es fácil. Por suerte, se inventaron diferentes formas de elegir y mezclar colores para hacerlo más fácil. Veremos uno a continuación.

6.1.1. Una forma diferente de elegir colores

Eche un vistazo a la siguiente imagen de una rueda de colores. Alrededor de la rueda podemos ver colores

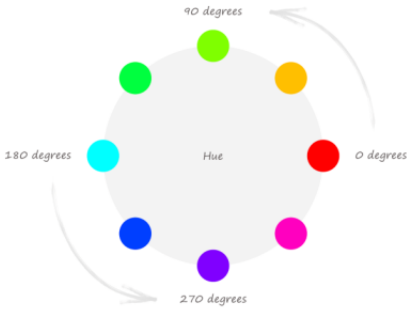

Figura 15: *Rueda de colores.*

como rojo, verde y azul. Para elegir un color decimos qué tan lejos está la rueda. Una buena forma de hacerlo es utilizar el ángulo. Aquí hay unos ejemplos:

- El rojo está a 0 grados
- El naranja está a 45 grados
- el verde está a 120 grados
- El azul está a 240 grados

Otra palabra para los colores alrededor de la rueda es tono (hue). ¿Cuál es el ángulo del tono morado?

(Fin de la segunda parte)