SISTEMAS OSCILATORIOS
MANUAL DE CÓDIGO

Contenido:

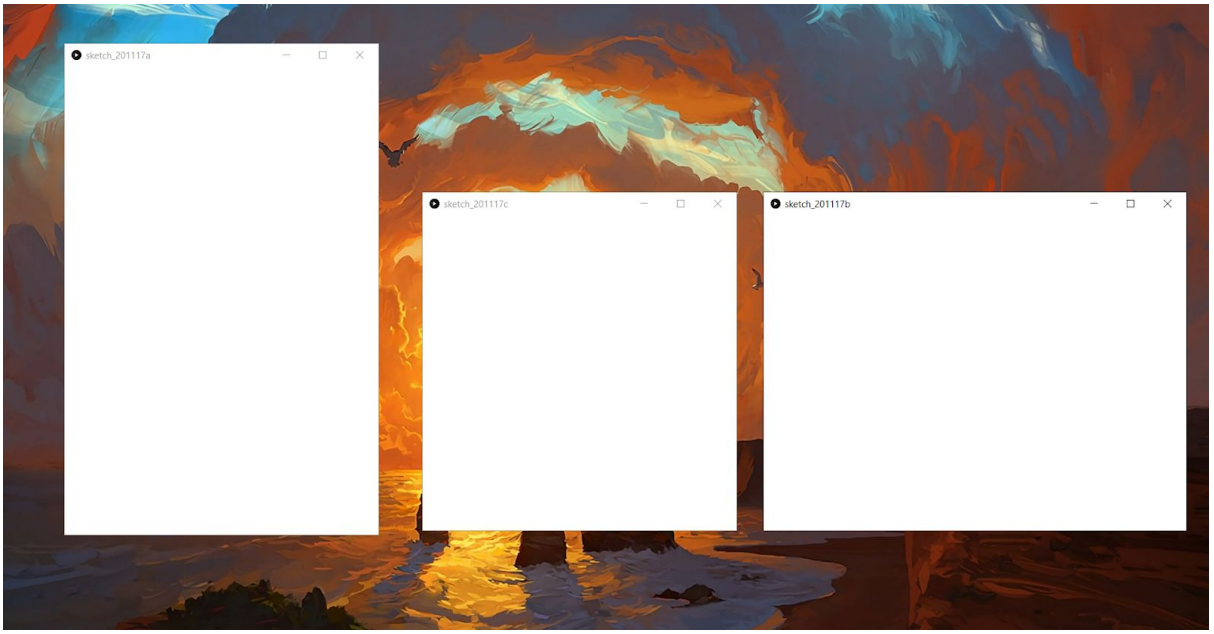
1 Introducción

En este manual explicaremos las partes más complejas de nuestro código ya que es demasiado extenso preferimos explicarlos con ejemplos utilizando el código que hemos ido desarrollando.

2 Cambiar El Tamaño

En esta sección veremos ejemplos sobre cómo conseguimos que nuestro programa permite cambiar el tamaño de ventana y que todo lo que se está dibujando cambie de tamaño con ella.

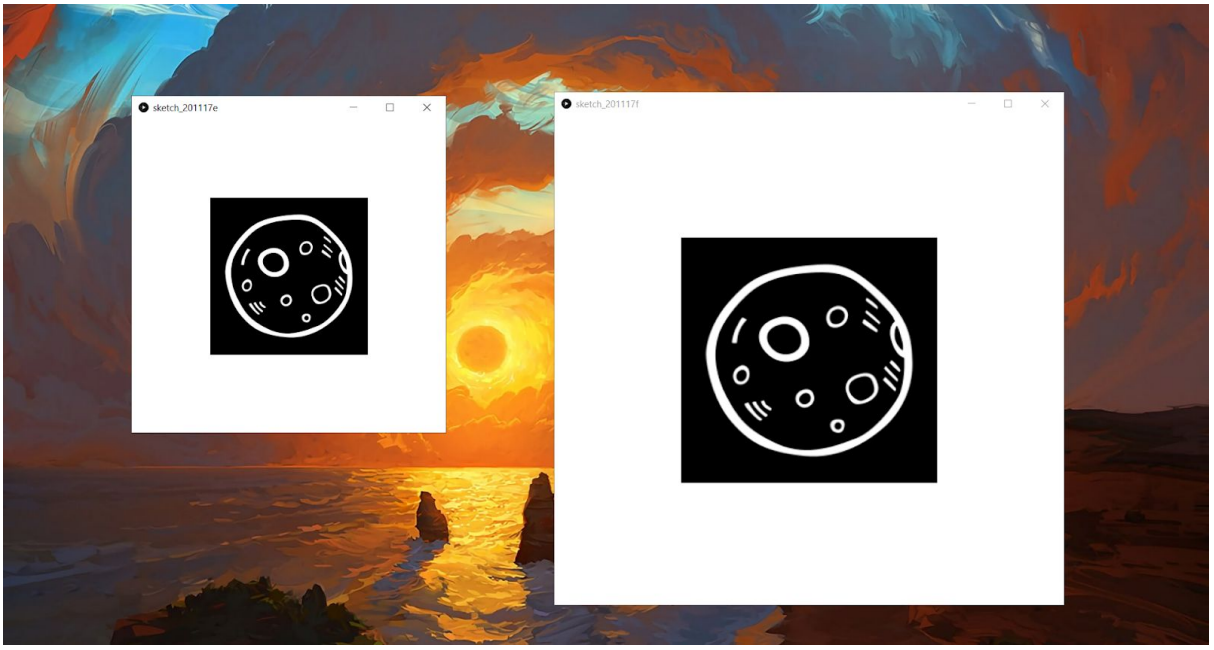
1.1 Tamaño de ventana




```
void setup(){
        size(500,500);
        surface.setResizable(true);
}
void draw(){
        background(255);
}
```

Por defecto cambiar el tamaño de ventana está deshabilitado, solo se necesita cambiarlo a verdadero para que podamos cambiar el tamaño de ventana, esto se logra con añadir “*surface.setResizable(true);*” en su setup.

1.2 Tamaño de Imágenes y Objetos

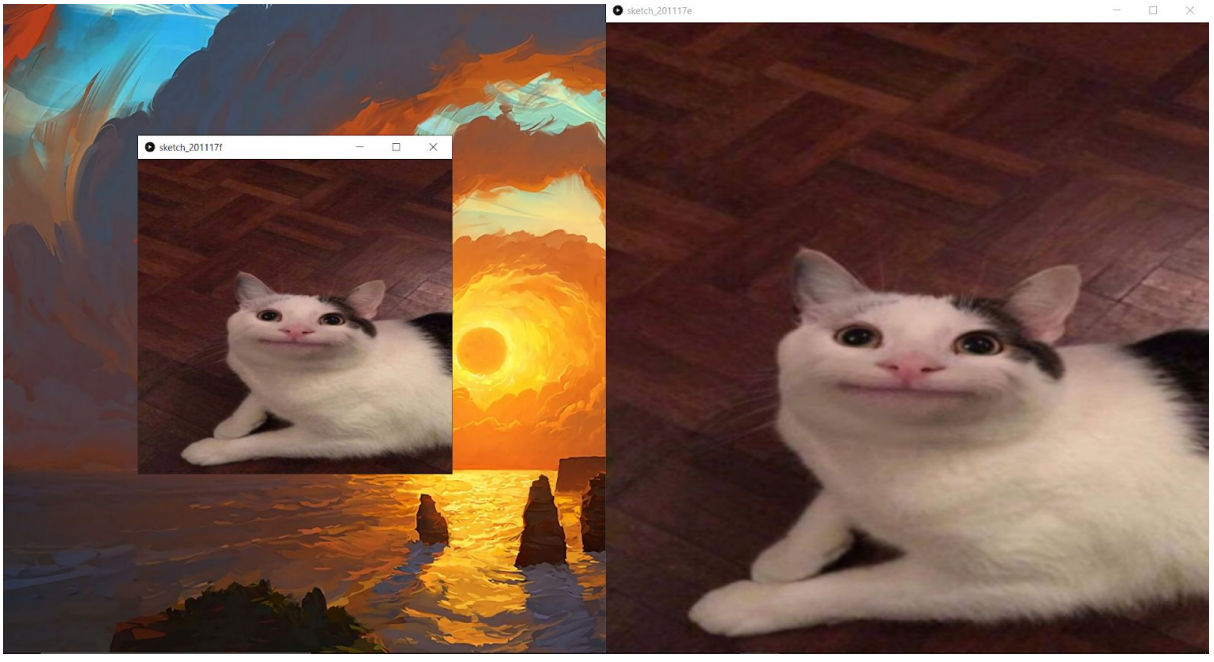



```
float W=width;
float H=height;
PImage moon;
void setup(){
        size(500,500);
        surface.setResizable(true);
        moon=loadImage("little moon.png");
        moon.resize(250,250);
}
void draw(){
          background(255);
         if(W!=width | H!=height){
                W=width;
                H=height;
         }
```

```
  scale(W/500,H/500);
  rectMode(CENTER);
  imageMode(CENTER);
  fill(0);
  rect(250,250,250,250);
  image(moon,250,250);
}
```

Para cambiar el tamaño de imágenes pequeñas y objetos que se dibujan en la ventana como rectángulos, líneas y eclipses. Se puede utilizar función *scale(x,y,z)* de processing, la cual transforma el tamaño de lo que venga después de su declaración. En el ejemplo se puede ver como utilizamos el *scale()* para que la imagen “little moon” y el rectángulo mantengan su tamaño en función del tamaño de la ventana (por alguna razón no sirve colocar la *width* y la *height* directamente, por eso utilizamos las variables *W* y *H* y las actualizamos cuando se cambia de tamaño la ventana).

1.3 Tamaño de Imágenes grandes



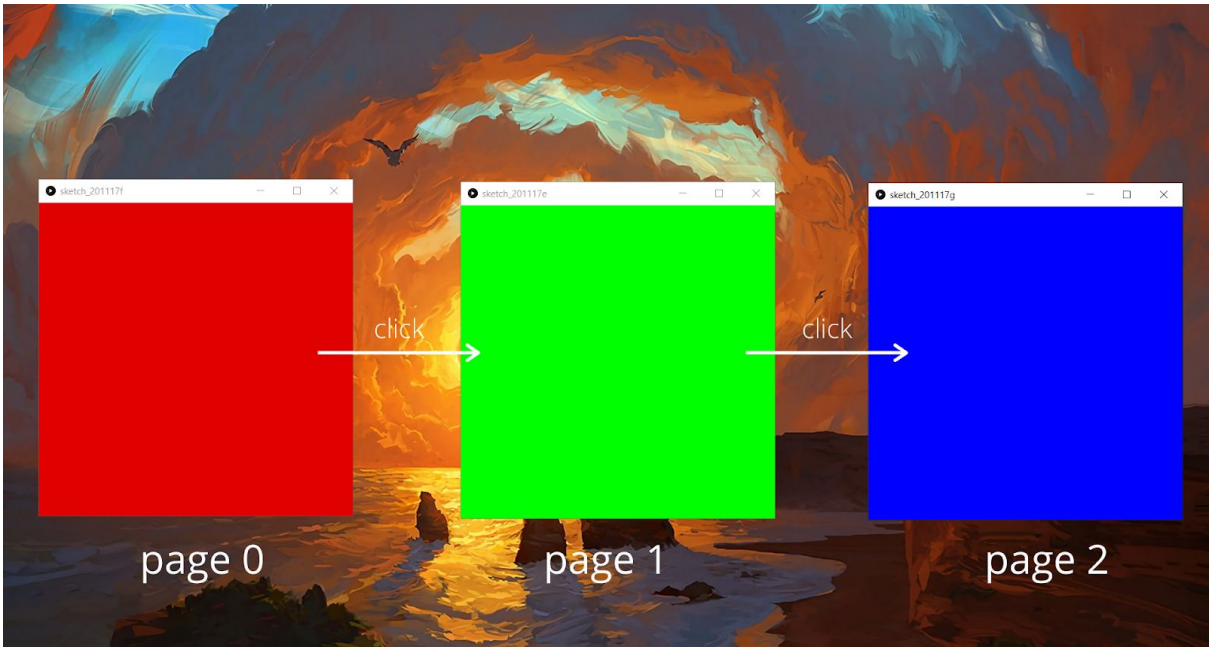

```
float W=width;
float H=height;
PImage cat, cat2;
void setup(){
    size(500,500);
```

```
        surface.setResizable(true);
        cat=loadImage("Polite Cat.jpg");
        cat.resize(width,height);
        cat2=cat.get();
}
void draw(){
         background(255);
        if(W!=width | H!=height){
                cat2=cat.get();
                cat2.resize(width,height);
                W=width;
                H=height;
        }
        image(cat2,0,0);
}
```

Cuando lidiamos con imágenes grandes (como las de fondo) y ventanas que permiten cambiar de tamaño las cosas se ponen complicadas, si utilizas *scale()* o *image(img, a, b, c, d)* en tu loop, la calidad se mantiene pero los framerates se mueren porque la imagen original se guarda en memoria y se tiene que transformar cada vez que se reinicia la loop. Mientras tanto, si utilizas *img.resize(x,y)* de una imagen img en tu loop, los framerates se mantienen pero la calidad muere esto es debido a que *img.resize()* cambia el tamaño de la imagen en memoria. Aquí es donde entra nuestra combinación que se puede ver en el código, básicamente la imagen original se guarda en un PImage *"cat" y se copia dentro de otra* PImage *"cat2".* Ahora cuando se realiza un cambio de tamaño se vuelve a copiar la imagen original y se cambia al tamaño deseado. De esta manera se mantienen la calidad de imagen y los framerates.

2 Cambiar de Página

En esta sección veremos como conseguimos cambiar de páginas en el programa



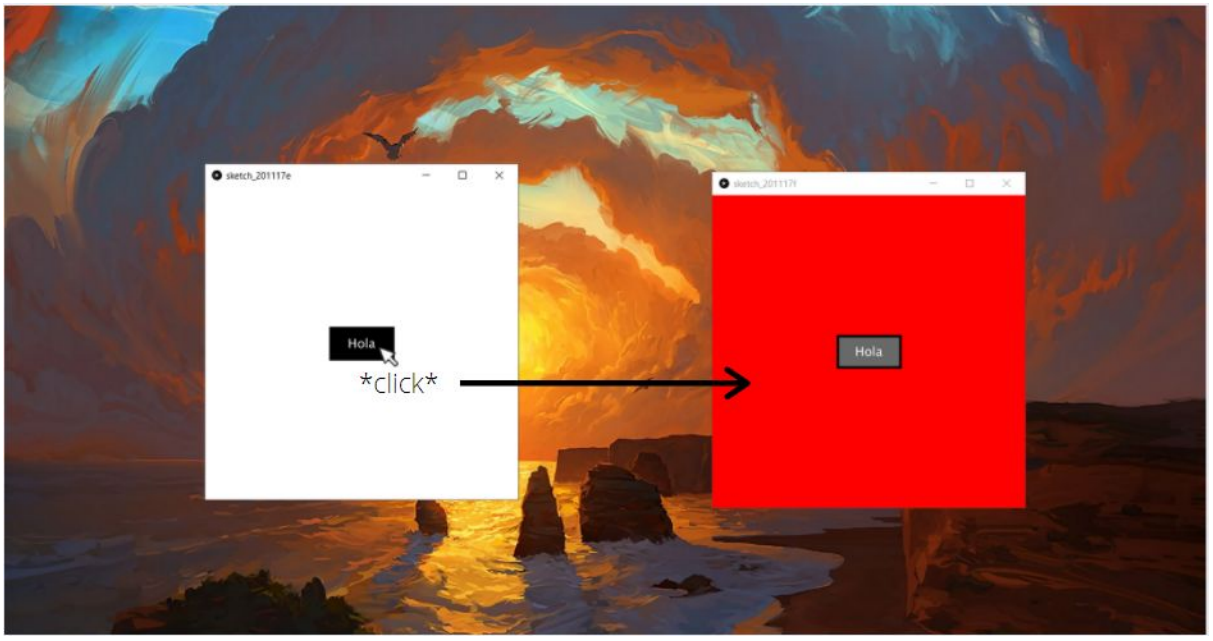

```
int page=0;
void setup(){
	size(500,500);
	surface.setResizable(true);
}
void draw(){
	switch(page){
		case 0:
		page0();
		break;
		case 1:
		page1();
		break;
		case 2:
		page2();
		break;
	}
```

```
}
void page0(){
	background(225,0,0);
}
void page1(){
	background(0,255,0);
}
void page2(){
	background(0,0,255);
}
void mouseClicked(){
	page+=1;
}
```

La manera más fácil y sencilla de dividir nuestro programa en diferentes páginas se basa en utilizar una variable entera global *“page”* en conjunto con un *switch(page){}* para que la variable *“page”* dicte en cual pagina nos encontramos. Cuando *“page”* es 0 se activa el caso 0 y se hace el llamado a una función *void* (no devuelve ningún valor) *“page0”* donde se sitúa el código que se ejecuta para la página 0. En este caso utilizamos el click del mouse para cambiar de pagina pero usted puede hacerlo de la manera que desee con solo cambiar el valor de la variable *“page”.*

3 Botones

En esta sección se explicará con detalle la clase *“Button”* que utilizamos para crear los botones




```
Button Hola;
color colour= color(255,255,255);
void setup(){
size(500,500);
surface.setResizable(true);
Hola= new Button("Hola", 20, width/2, height/2, 100, 50);
}
void draw(){
  background(colour);
  Hola.display();
}

void mouseMoved(){
  Hola.isInside();
}
```

```
void mouseClicked(){
  if(Hola.isHovering){
    if(colour==color(255,255,255)){
      colour= color(255,0,0);
    }else{
      colour= color(255,255,255);
    }
  }
}

class Button{
  boolean isHovering;
  String txt;
  float txtsize, x, y, w, h;

  //Recibe las variables de cada botón
  Button(String label, float lblsize, float posX, float posY, float Width, float Height){
    txt= label;
    txtsize= lblsize;
    x= posX;
    y= posY;
    w= Width;
    h= Height;
  }

  void display(){
    //Dibuja el botón
    rectMode(CENTER);
    fill(isHovering? color(0,0,0) : color(105,105,105));
    stroke(0);
```

```
    strokeWeight(4);
    rect(x,y,w,h);

    //Dibuja el texto del botón
    textAlign(CENTER);
    textSize(txtsize);
    fill(255);
    if(height>=720){
      text(txt, x, y+7/((0+height)/720));
    }else{
      text(txt, x, y+7);
    }
  }
  //Verifica si el cursor esta dentro del boton
  boolean isInside() {
    return isHovering = mouseX > (x-w/2)*width/500 & mouseX < (x+w/2)*width/500 &
mouseY > (y-h/2)*height/500 & mouseY < (y+h/2)*height/500;
  }
}
```

3.1 Declaración del botón

Empezaremos explicando la clase *“Button”* que creamos para el programa.

Se comienza con la declaración del objeto, el boton *“Hola”* antes del setup

```
Button Hola;
```

En el setup se declaran los valores *Button(String label, float lblsize, float posX, float posY, float Width, float Height)* del botón *“Hola”*

```
Hola= new Button("Hola", 20, width/2, height/2, 100, 50);
```

En ese entonces dentro de la clase *“Button”* se inicia el method *“Button(String label, float lblsize, float posX, float posY, float Width, float Height)”* donde las variables introducidas son guardadas variables locales de la clase para que después sean usadas en otras funciones.

3.2 Dibujo del botón

En la loop draw, se inicia la función *display()* de la clase, esta función como su nombre alude, dibuja el botón en la ventana.

```
Hola.display();
```

Se hace el llamado y la función arranca. Con los valores guardados del botón *“Hola”* se dibuja el botón.

```
 void display(){
    //Dibuja el boton
    rectMode(CENTER);
    fill(isHovering? color(0,0,0) : color(105,105,105));
    stroke(0);
    strokeWeight(4);
    rect(x,y,w,h);

    //Dibuja el texto del boton
    textAlign(CENTER);
    textSize(txtsize);
    fill(255);
    if(height>=720){
      text(txt, x, y+7/((0+height)/720));
    }else{
      text(txt, x, y+7);
    }
  }
```

3.3 Mouse dentro del botón

Cuando el mouse esta dentro del boton, el boton cambia de color. Para realizar eso se tiene que verificar la posicion del boton en cada momento

```
void mouseMoved(){
  Hola.isInside();
}
```

La función de *mouseMoved()* de processing se activa cada vez que se mueve el mouse, después usamos la función *isInside()* para revisar si el mouse está dentro

```
boolean isInside() {
   return isHovering = mouseX > (x-w/2)*width/500 & mouseX < (x+w/2)*width/500 & mouseY > (y-h/2)*height/500 & mouseY < (y+h/2)*height/500;
 }
```

Esta función nos devuelve si el mouse está dentro del botón con un boolean *"isHovering"*, el cual se utiliza en la función *mouseClicked()* de processing para realizar el cambio de colores cuando el botón es clickeado

```
void mouseClicked(){
  if(Hola.isHovering){
    if(colour==color(255,255,255)){
      colour= color(255,0,0);
    }else{
      colour= color(255,255,255);
    }
  }
}
```

Cuando *isHovering* regresa verdadero el boton se pinta de color negro

```
fill(isHovering? color(0,0,0) : color(105,105,105));
```

y se permite el efecto del botón si es clickeado.

4 Plano cartesiano

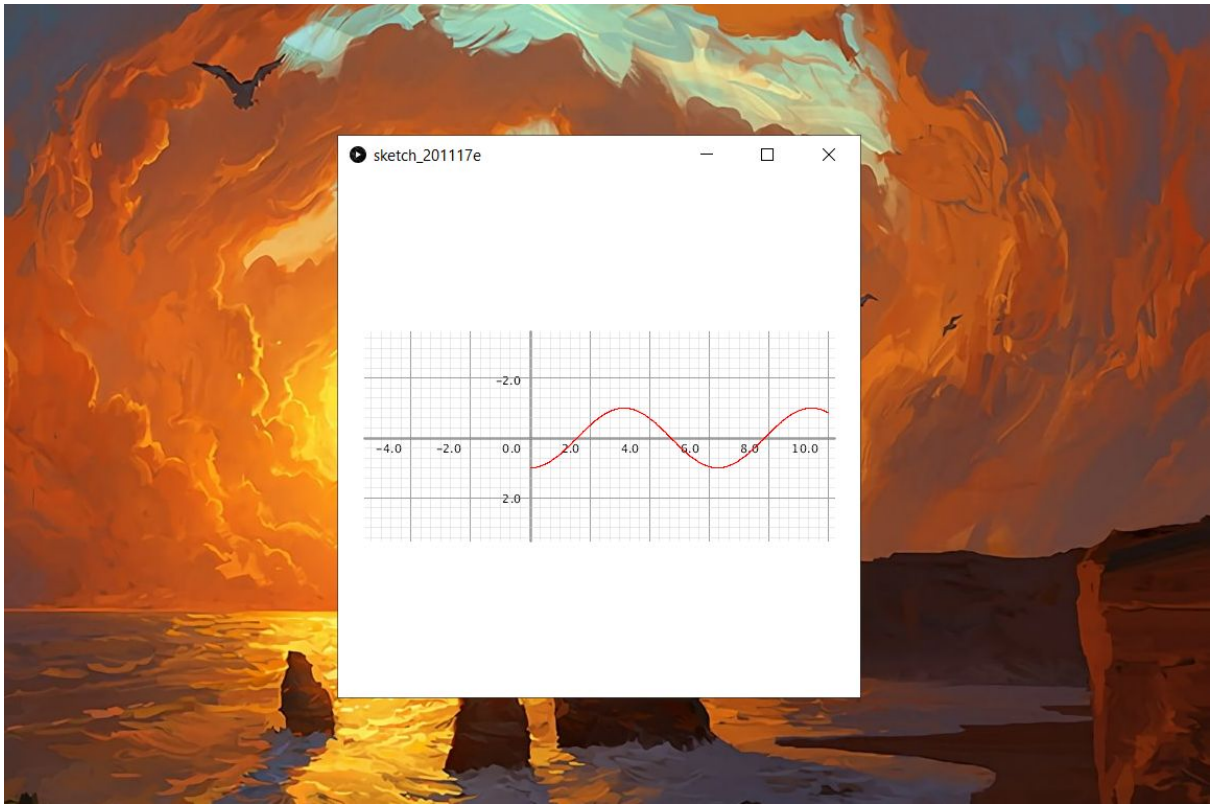
En esta sección explicaremos el rompecabezas de la función dibujar. En nuestro programa esta función dibuja un plano cartesiano y el objeto a dibujar dentro de ese plano independiente de lo demás que es dibujado en la pantalla e independiente de otros planos cartesianos con simplemente utilizar un vector float en posición de una variable float normal. Para simplificar la explicación reduciremos la función a que dibuje 1 solo plano

se va de

```
void dibujar(int k){
scaleP[k]
}
```

a

```
void dibujar{
scaleP
}
```




```
float mat[]= new float[1];
float scaleP=1;
float scaleG=1;
```

```
float r=1;
float x=1;
float xPan=250, yPan=250;
int nX=0, nY=0;

void setup(){
size(500,500);
surface.setResizable(true);
}
void draw(){
  float j=0;
  for(int i=0; i<mat.length; i++){
    mat[i]=cos(j);
    j+=0.016666666666666666666666666666667;
  }
  background(255,255,255);
  dibujar();

  float[] tmpA= new float[mat.length+1];
      for(int i=0; i<mat.length; i++){
        tmpA[i]= mat[i];
        tmpA[i]= mat[i];
        tmpA[i]= mat[i];
        mat= tmpA;
      }

}

void dibujar(){
  if(scaleP>2){
```

```
    scaleG= r*2;
    r=scaleG;
    scaleP=1;
    x/=0.5;
  }
  if(scaleP<0.5){
    scaleG= r/2;
    r= scaleG;
    scaleP=1;
    x*=0.5;
  }
  pushMatrix();
    translate(0,0);
    textAlign(RIGHT);
    translate(xPan, yPan);
    scale(scaleP);
    for (int i=-(int)(225/scaleP)-(int)(nX/scaleP); i<(int)(225/scaleP)-(int)(nX/scaleP);
i++) {
      if (i%60==0) {
        if (i==0) {
          strokeWeight(2);
        } else {
          strokeWeight(1);
        }
        stroke(105, 105, 105);
        line(i, -100/scaleP-nY/scaleP, i, 100/scaleP-nY/scaleP);
        fill(0);
        if(i>-(int)(225/scaleP)-(int)(nX/scaleP) && i<(int)(225/scaleP)-(int)(nX/scaleP)){
          if(15/scaleP>-(int)(100/scaleP)-(int)(nY/scaleP) &&
15/scaleP<(int)(100/scaleP)-(int)(nY/scaleP)){
            text(String.valueOf((float)(i/60)/x), i-10, 15);
```

```
        }
      }
    }
    if (i%10==0) {
      strokeWeight(0.5);
      stroke(211, 211, 211);
      line(i, -100/scaleP-nY/scaleP, i, 100/scaleP-nY/scaleP);
    }
  }

  for (int i=-(int)(100/scaleP)-(int)(nY/scaleP); i<(int)(100/scaleP)-(int)(nY/scaleP);
i++) {
    if (i%60==0) {
      if (i==0) {
        strokeWeight(2);
      } else {
        strokeWeight(1);
      }
      stroke(105, 105, 105);
      line(-225/scaleP-nX/scaleP, i, 225/scaleP-nX/scaleP, i);

      if (i!=0) {
        fill(0);
        if(i>-(int)(100/scaleP)-(int)(nY/scaleP) && i<(int)(100/scaleP)-(int)(nY/scaleP)){
          if(-10/scaleP>-(int)(225/scaleP)-(int)(nX/scaleP) &&
-10/scaleP<(int)(225/scaleP)-(int)(nX/scaleP)){
            text(String.valueOf((float)(i/60)/x), -10, i+5);
          }
        }
      }
    }
```

```
      if (i%10==0) {
        strokeWeight(0.5);
        stroke(211, 211, 211);
        line(-225/scaleP-nX/scaleP, i, 225/scaleP-nX/scaleP, i);
      }
    }
    popMatrix();
    pushMatrix();
    translate(0,0);
    translate(xPan, yPan);
    scale(scaleG);
    stroke(255,0,0);
    strokeWeight(1/scaleG);
    for(int i=1; i<mat.length; i++){
      if(((int)(i-1)*r)>((int)(-225/scaleP)-(int)(nX/scaleP)) &&
(int)((i)*r)<((int)(225/scaleP)-(int)(nX/scaleP))){
        if((int)((60*mat[i])*r)>(-(int)(100/scaleP)-(int)(nY/scaleP)) &&
(int)((60*mat[i-1])*r)<((int)(100/scaleP)-(int)(nY/scaleP))){
          line(i-1, 60*mat[i-1], i, 60*mat[i]);
        }
      }
    }
    popMatrix();
}

void mouseWheel(MouseEvent event){
    int e = event.getCount();
  if(mouseX>100*width/1280 & mouseX<1000*width/1280){
    if(mouseY>250*height/720 & mouseY<500*height/720){
    if(e>0){
```

```
      scaleG/=1.05;
      scaleP/=1.05;
    }else{
      scaleG*=1.05;
      scaleP*=1.05;
    }
  }
  }
}

void mouseDragged(MouseEvent event) {
  if(mouseX>100*width/1280 & mouseX<1000*width/1280){
    if(mouseY>250*height/720 & mouseY<500*height/720){
      xPan = xPan+(mouseX - pmouseX);
      yPan = yPan+(mouseY - pmouseY);
      nX= nX+(mouseX - pmouseX);
      nY= nY+(mouseY - pmouseY);
    }
  }
}
```

4.1 Inicialización de variables

Primero se inicializan las variables que se utilizaran para dibujar el plano cartesiano

```
float mat[]= new float[600];
```

En el vector *“mat”* se guardan los valores de la gráfica cos(x) (se mantuvo a un máximo de 600 para simplificar las cosas)

```
float scaleP=1;
float scaleG=1;
float r=1;
float x=1;
```

```
float xPan=250, yPan=250;
```

xPan es donde se encuentra el centro del plano en X y yPan es donde se encuentra el centro del plano en Y

```
int nX=0, nY=0;
```

4.2 Cálculo de gráfica y crecimiento

En la loop draw se calcula los valores de la matriz

```
for(int i=0; i<mat.length; i++){
   mat[i]=cos(j);
   j+=0.016666666666666666666666666666667;
 }
```

Se suma 0.016666666666666666666666666666667 cada iteración lo que llega a ser 1 segundo cada 60 píxeles lo que termina igualando la velocidad de dibujo a la velocidad de oscilación real en segundos.

```
 dibujar();
```

Se llama a la función *dibujar* para crear el plano y la gráfica dentro del plano

```
float[] tmpA= new float[mat.length+1];
     for(int i=0; i<mat.length; i++){
       tmpA[i]= mat[i];
       tmpA[i]= mat[i];
       tmpA[i]= mat[i];
       mat= tmpA;
     }
```

Después se realiza el crecimiento del vector copiandolo en un vector más grande *“tmpA”* y devolviendolo con el nombre *“mat”*

4.3 Función dibujar

```
void dibujar(){
  if(scaleP>2){
    scaleG= r*2;
    r=scaleG;
```

```
    scaleP=1;
    x/=0.5;
  }
  if(scaleP<0.5){
    scaleG= r/2;
    r= scaleG;
    scaleP=1;
    x*=0.5;
  }
```

ScaleP viene siendo el valor de escala del plano cartesiano, y scaleG viene siendo el valor de escala de la gráfica.

Cuando scaleP>2 o scaleP<0.5, scaleP vuelve al valor de 1 y x dependiendo del caso se multiplica o divide por 0.5. Esto hace la elusión de los cuadrados se dividen o se multiplican en el plano. El valor de x es el divisor de los números que aparecen en el plano.

scaleG se cambia al valor de r/2 o r*2 dependiendo del caso para que scaleG se quede sincronizado con scaleP para que la gráfica se quede en la posición correcta.

Mas adelante se utiliza un push y pop Matrix para que solo las transformaciones deseadas afecten al plano cartesiano.

```
  pushMatrix();
    translate(0,0);
    textAlign(RIGHT);
    translate(xPan, yPan);
```

Este translate nos permite movernos en el plano con los valores de xPan y yPan.

```
    scale(scaleP);
```

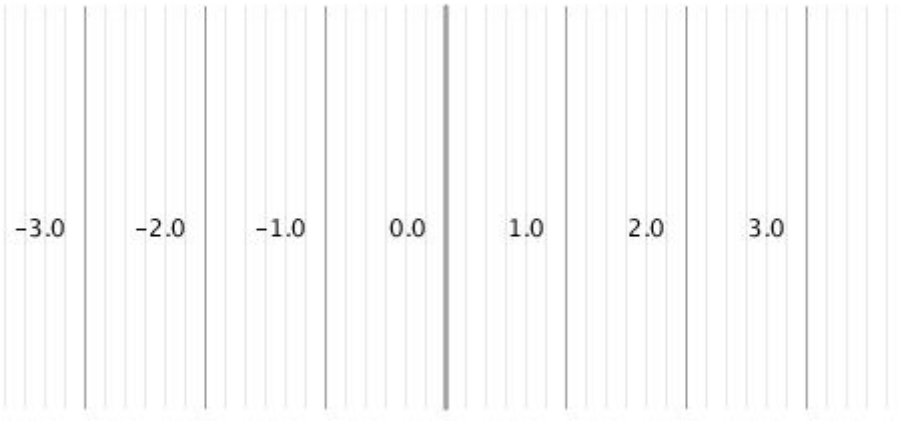
En este for loop se necesita dividir con el scaleP para que el tamaño general del plano se quede consistente, mientras que se realice scale(scaleP) para que se vea el zoom en las líneas visibles dentro del cuadro.

También se restan los valores de nX y nY para que el plano cartesiano no se mueva de su lugar en la ventana

En este for loop se dibujan las líneas verticales y sus numeros

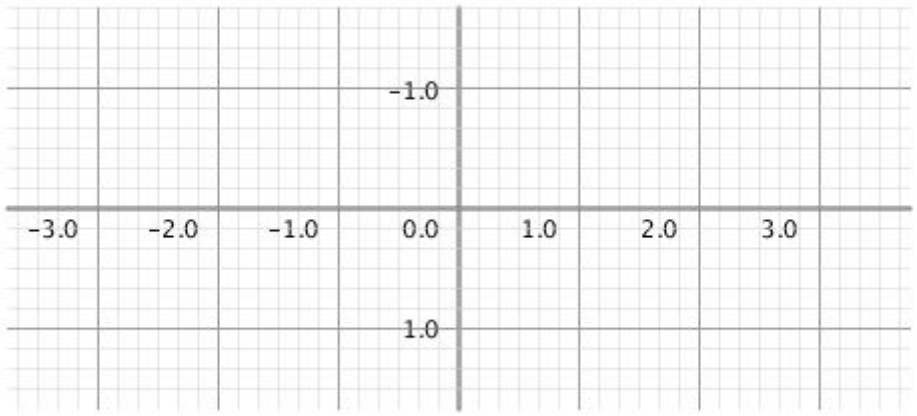
```
    for (int i=-(int)(225/scaleP)-(int)(nX/scaleP); i<(int)(225/scaleP)-(int)(nX/scaleP);
i++) {
      if (i%60==0) {
```

```
      if (i==0) {
        strokeWeight(2);
      } else {
        strokeWeight(1);
      }
      stroke(105, 105, 105);
      line(i, -100/scaleP-nY/scaleP, i, 100/scaleP-nY/scaleP);
      fill(0);
      if(i>-(int)(225/scaleP)-(int)(nX/scaleP) && i<(int)(225/scaleP)-(int)(nX/scaleP)){
        if(15/scaleP>-(int)(100/scaleP)-(int)(nY/scaleP) &&
15/scaleP<(int)(100/scaleP)-(int)(nY/scaleP)){
          text(String.valueOf((float)(i/60)/x), i-10, 15);
        }
      }
    }
    if (i%10==0) {
      strokeWeight(0.5);
      stroke(211, 211, 211);
      line(i, -100/scaleP-nY/scaleP, i, 100/scaleP-nY/scaleP);
    }
  }
```



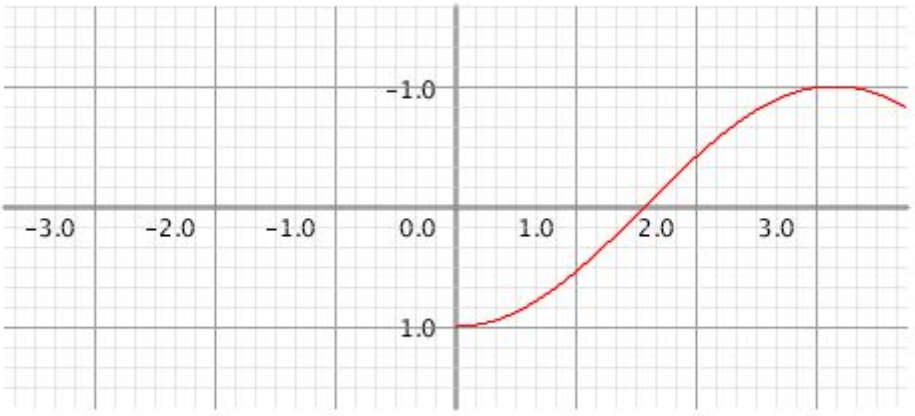
En este for loop se dibujan las líneas horizontales y sus numeros

```
 for (int i=-(int)(100/scaleP)-(int)(nY/scaleP); i<(int)(100/scaleP)-(int)(nY/scaleP); i++)
{
    if (i%60==0) {
      if (i==0) {
        strokeWeight(2);
      } else {
        strokeWeight(1);
      }
      stroke(105, 105, 105);
      line(-225/scaleP-nX/scaleP, i, 225/scaleP-nX/scaleP, i);

      if (i!=0) {
        fill(0);
        if(i>-(int)(100/scaleP)-(int)(nY/scaleP) && i<(int)(100/scaleP)-(int)(nY/scaleP)){
          if(-10/scaleP>-(int)(225/scaleP)-(int)(nX/scaleP) &&
-10/scaleP<(int)(225/scaleP)-(int)(nX/scaleP)){
            text(String.valueOf((float)(i/60)/x), -10, i+5);
          }
        }
      }
    }

    if (i%10==0) {
      strokeWeight(0.5);
      stroke(211, 211, 211);
      line(-225/scaleP-nX/scaleP, i, 225/scaleP-nX/scaleP, i);
    }
  }
  popMatrix();
```



Aqui se dibuja la grafica con la escala scaleG

```
    pushMatrix();
    translate(0,0);
    translate(xPan, yPan);
    scale(scaleG);
    stroke(255,0,0);
    strokeWeight(1/scaleG);
    for(int i=1; i<mat.length; i++){
      if(((int)(i-1)*r)>((int)(-225/scaleP)-(int)(nX/scaleP)) &&
(int)((i)*r)<((int)(225/scaleP)-(int)(nX/scaleP))){
        if((int)((60*mat[i])*r)>(-(int)(100/scaleP)-(int)(nY/scaleP)) &&
(int)((60*mat[i-1])*r)<((int)(100/scaleP)-(int)(nY/scaleP))){
          line(i-1, 60*mat[i-1], i, 60*mat[i]);
        }
      }
    }
    popMatrix();
}
```



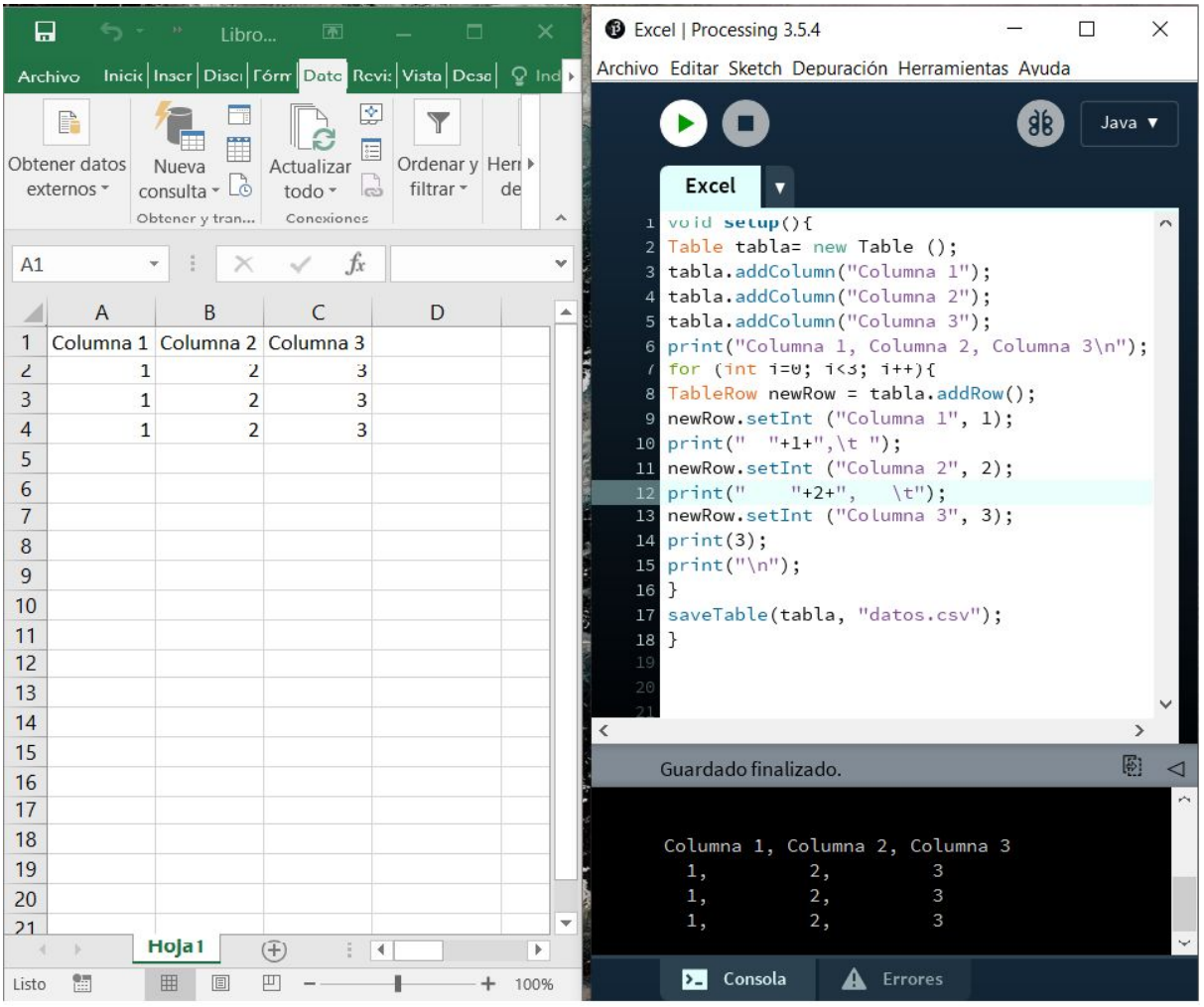
La función mouseWheel tira un valor negativo cuando se mueve la rueda hacia adelante y un valor positivo cuando se mueve la rueda hacia atrás. Esto lo podemos utilizar para crear el zoom con las 2 variables de escala scaleP y scaleG multiplicandolas y dividiéndolas según el caso con un número (cuando más grande el número más zoom y un zoom hay).

```
void mouseWheel(MouseEvent event){
   int e = event.getCount();
 if(mouseX>100*width/1280 & mouseX<1000*width/1280){
   if(mouseY>250*height/720 & mouseY<500*height/720){
   if(e>0){
      scaleG/=1.05;
      scaleP/=1.05;
    }else{
      scaleG*=1.05;
      scaleP*=1.05;
    }
 }
 }
}
```

Con la función mouseDragged() de processing podemos calcular la distancia en la que se arrastra el mouse. restamos la posición actual con la anterior y lo sumamos a las variables xPan, yPan, nX, nY para mantener el plano cartesiano en orden cuando lo movemos.

```
void mouseDragged(MouseEvent event) {
  if(mouseX>100*width/1280 & mouseX<1000*width/1280){
    if(mouseY>250*height/720 & mouseY<500*height/720){
      xPan = xPan+(mouseX - pmouseX);
      yPan = yPan+(mouseY - pmouseY);
      nX= nX+(mouseX - pmouseX);
      nY= nY+(mouseY - pmouseY);
    }
  }
}
```

5 Archivos

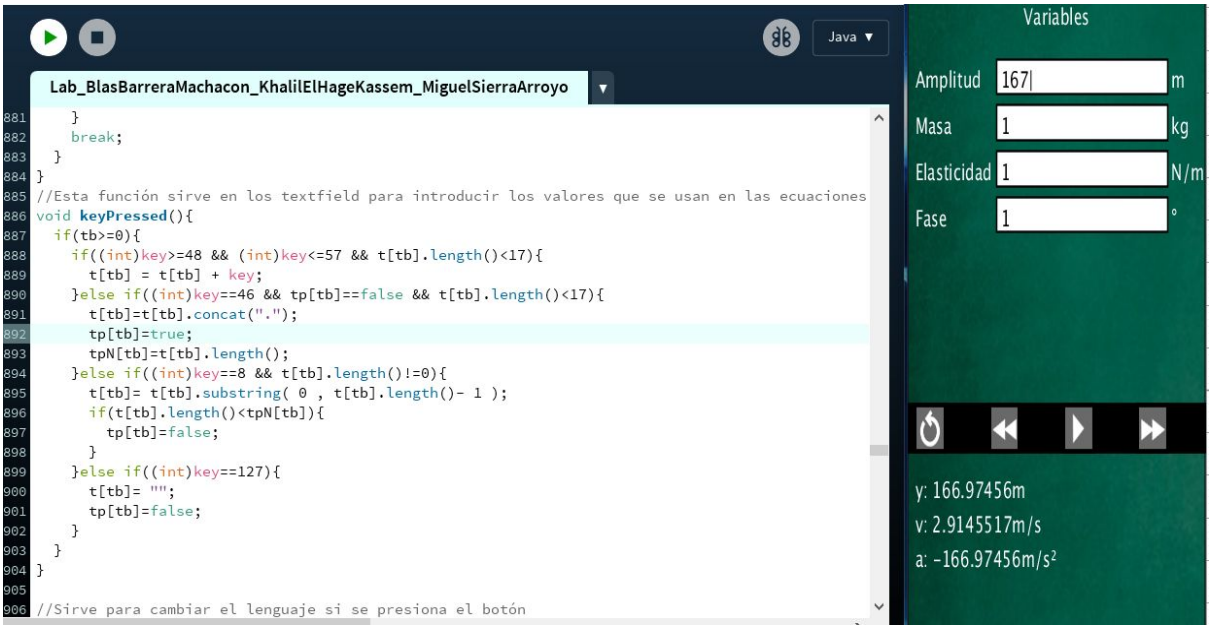


En esta sección se mostrará cómo se implementa la creación una tabla en un archivo .CSV (Comma Separated Value) que usamos en nuestro programa para guardar los datos de la posición, velocidad y aceleración.

```
Table tabla= new Table ();
tabla.addColumn(“Columna 1”);
tabla.addColumn(“Columna 2”);
tabla.addColumn(“Columna 3”);
print(“Columna 1, Columna 2, Columna 3\n”);
for (int i=0; i<3; i++){
        TableRow newRow = tabla.addRow();
        newRow.setInt (“Columna 1”, 1);
        print(1+”,\t “);
        newRow.setInt (“Columna 2”, 2);
        print(2+”, \t“);
        newRow.setInt (“Columna 3”, 3);
        print(3);
```

```
        print("\n");
}
saveTable(tabla, "datos.csv");
```

6 Campos de texto



Ejemplo:

```
int tpN;
String t="";
boolean tp=false;
void setup(){
  size(500,500);
}
void draw(){
  background(0);
textMode(CENTER);
text(t,250,250);
}
void keyPressed(){

if((int)key>=48 && (int)key<=57 && t.length()<17){
t= t + key;
       }else if((int)key==46 && tp==false && t.length()<17){
          t=t.concat(".");
          tp=true;
```

```
            tpN=t.length();
        }else if((int)key==8 && t.length()!=0){
            t= t.substring(0, t.length()- 1);
            if(t.length()<tpN){
                tp=false;
            }
        }else if((int)key==127){
            t= "";
            tp=false;
        }
    }
```

En los campos de texto, permitimos que el usuario edite el valor de las variables que inciden en el movimiento del sistema. En nuestro caso, no permitimos que se ingresen caracteres distintos a números, o números negativos. Todo esto se hace en base a la nomenclatura de las teclas que son presionadas por el usuario al momento de digitar el valor deseado.
El primer condicional verifica si las teclas presionadas corresponden a las del teclado numérico, en cuyo caso concatenar el valor digitado con el ya escrito en el campo de texto.
El segundo condicional verifica si el usuario presiona la tecla punto para separar los números en decimal, en cuyo caso concatena el punto al campo de texto ya modificado
El tercer y cuarto condicional sirven para eliminar números del Campo de texto mediante la tecla Suprimir y Supr respectivamente, la primera elimina un sólo valor mientras que la otra elimina todo el número del Campo de texto.

7 Música

En nuestro programa, hemos decidido incluir música, pues queremos que el usuario esté a gusto y cómodo a medida que experimenta usando nuestro programa, que relacione la física que se le presenta con la relajación de una buena música que ambiente toda la simulación. A su vez, entendemos que a más de un usuario puede que no le guste, o que no se sienta con ganas en ese momento de escuchar música, por lo que tiene la opción de pausar o reproducir la música a su antojo.

[good jazz music]

```
import ddf.minim.*;
Minim music;
AudioPlayer jazz;
void setup(){

  music= new Minim(this);
  jazz= music.loadFile("relaxing-jazz-hiphop.mp3");
  jazz.loop();
}

void draw(){

}
```