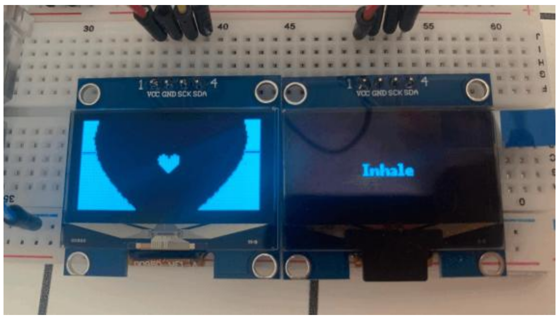
Overview

The goal of making this simple breath visualiser is to help you stay grounded, calm and present in times of panic, anxiety or stress.

Admittedly, it is probably easier to find and use one of these online, but I figured I'd challenge myself to make something physical and learn a bit about embedded systems while I'm at it.



Equipment I used

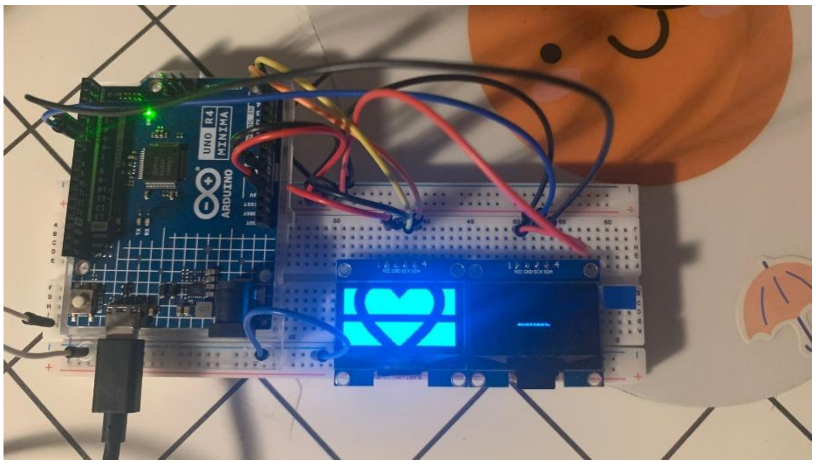
- Arduino Uno R4 Minima
- Jumper wires (male-male)
- x2 SH1106 OLED screens
- Breadboard
- heart GIF

Process

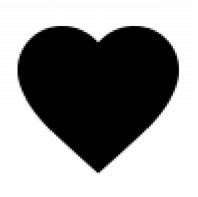
1. **<u>Pin wiring for 2 OLED screens</u>**

OLED #1	VCC	3.3V
	GND	GND
	SCL/SCK (clock)	A5
	SDA	A4

OLED #2	VCC	3.3V
	GND	GND
	SCL/SCK (clock)	Digital 6
	SDA	Digital 5



2. **Optimise GIF**
   Originally, the expanding and contracting heart GIF was 96x96 pixels.
   I cropped it (using this site: https://ezgif.com/) to 128x64 so it would be able to fit the size of my OLED screen. Although the full image couldn't be in it, it proved to be an easier approach when you convert each frame to byte arrays since everything needs to be very specific and to scale, if not, you'll end up with some "corrupted looking" pixels in the animation.



Then, I split the frames of the GIF (also using https://ezgif.com/). I got around ~17 frames for the entire animation. Save it as PNG.

(Note: when it comes to the software, you can get away with just downloading and converting the first ~6-ish frames or half of your total frames and then have the loop run the frames backwards for the contraction of the heart.)

Using this site: https://javl.github.io/image2cpp/ you can convert PNGs to byte arrays and vice versa. I converted most of my frames of the animation into byte arrays. Depending on how the images show up on the OLED, you may have to select the "flip image: horizontal" option in image settings on the site.
For the output, you can select "Arduino code" or "plain bytes".
The "draw mode" should be set to "Horizontal – 1 bit per pixel".

Reference the Arduino sketch and copy in your byte arrays for each frame and place them in the “heart_bits[][]” array.

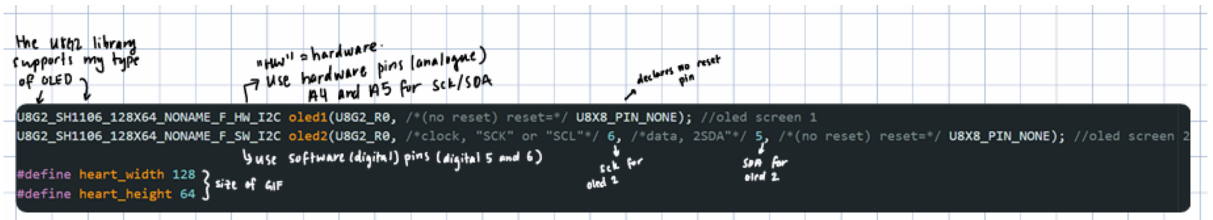
3. **<u>Coding the software</u>**
   I installed the u8g2 library since it’s the one that works for my SH1106 OLED screens. There should be exactly 1024 bytes in each byte array since 128×64=8192/8=1024. If you have less, e.g 1008 bytes when you copied the byte arrays from the website, you can add in 16 (or however many you need) “0x00”’s to the end of each array to get it to 1024. I have also declared in the heart_bits[] array to have [1040] bytes since I got an error when I declared it as [1024]…but it works! So don’t touch it.

   The drawAnimation() function and heart GIF was taken from https://github.com/tigrisli/oled-animation/.

4. Compile (and pray that you get no errors…) and then upload onto the Arduino !

///

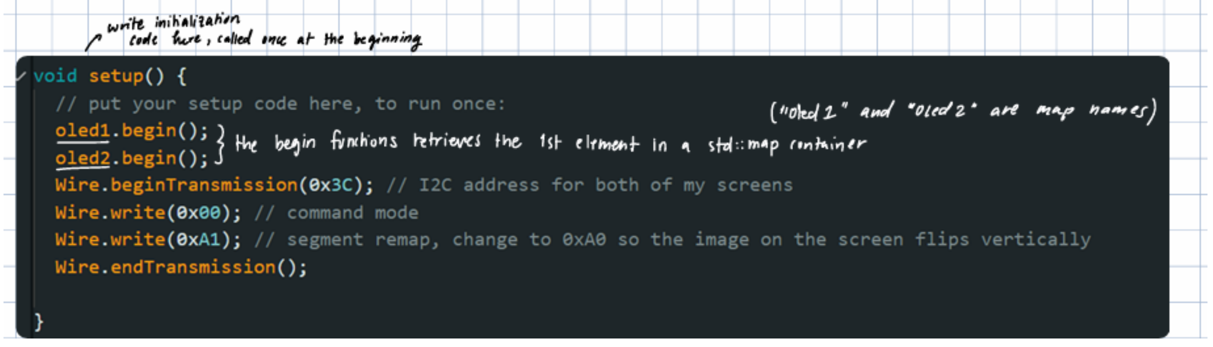
I also wrote some extra notes about the code so I could better understand it.



```
U8G2_SH1106_128X64_NONAME_F_HW_I2C oled1(U8G2_R0, /*(no reset) reset=*/ U8X8_PIN_NONE); //oled screen 1
U8G2_SH1106_128X64_NONAME_F_SW_I2C oled2(U8G2_R0, /*clock, "SCK" or "SCL"*/ 6, /*data, 2SDA"*/ 5, /*(no reset) reset=*/ U8X8_PIN_NONE); //oled screen 2

#define heart_width 128
#define heart_height 64
```

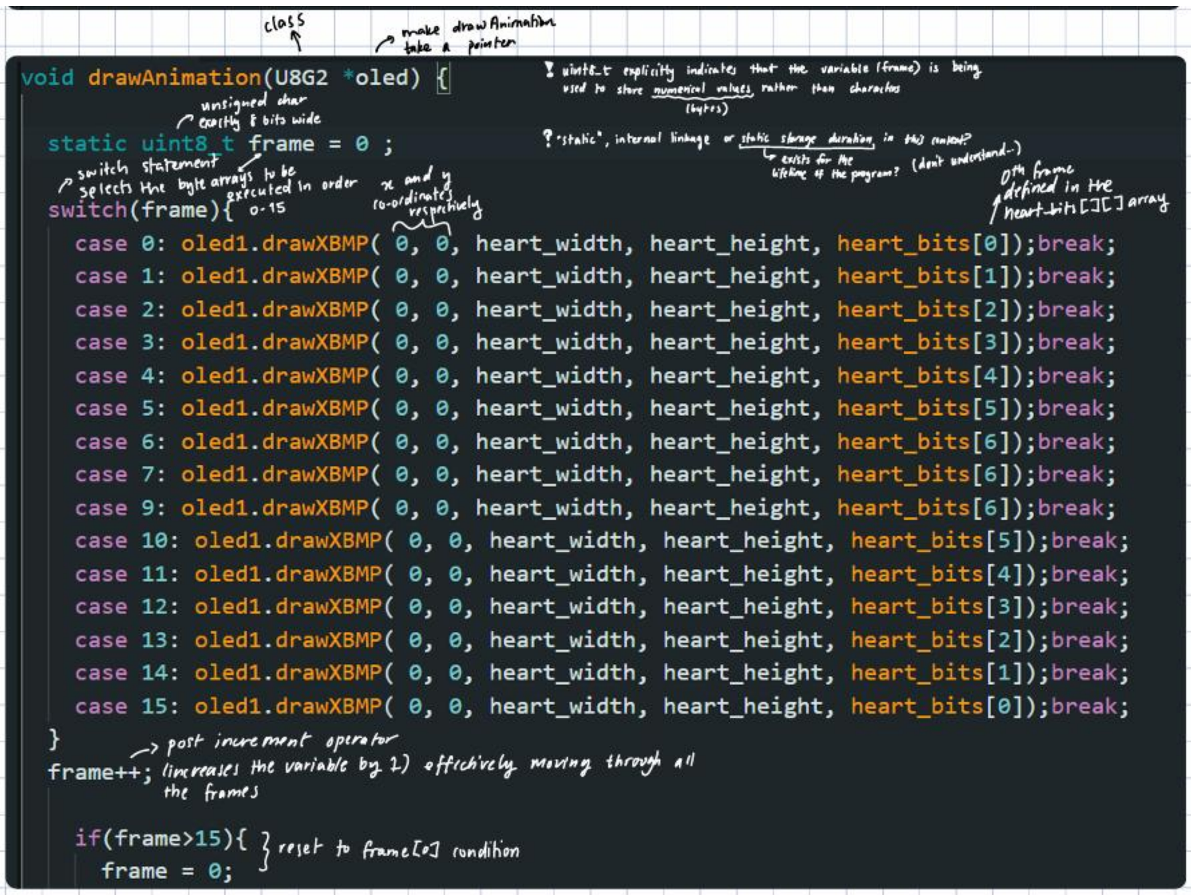
the u8g2 library supports my type of OLED
"HW" = hardware. use hardware pins (analogue) A4 and A5 for SCK/SDA
declares no reset pin
use software (digital) pins (digital 5 and 6)
SCK for oled 2
SDA for oled 2
size of GIF



```
void setup() {
  // put your setup code here, to run once:
  oled1.begin();
  oled2.begin();
  Wire.beginTransmission(0x3C); // I2C address for both of my screens
  Wire.write(0x00); // command mode
  Wire.write(0xA1); // segment remap, change to 0xA0 so the image on the screen flips vertically
  Wire.endTransmission();

}
```

write initialization code here, called once at the beginning
("oled1" and "oled2" are map names)
the begin functions retrieves the 1st element in a std::map container



class

make drawAnimation take a pointer

```
void drawAnimation(U8G2 *oled) {

  static uint8_t frame = 0 ;

  switch(frame){
    case 0: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[0]);break;
    case 1: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[1]);break;
    case 2: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[2]);break;
    case 3: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[3]);break;
    case 4: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[4]);break;
    case 5: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[5]);break;
    case 6: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[6]);break;
    case 7: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[6]);break;
    case 9: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[6]);break;
    case 10: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[5]);break;
    case 11: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[4]);break;
    case 12: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[3]);break;
    case 13: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[2]);break;
    case 14: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[1]);break;
    case 15: oled1.drawXBMP( 0, 0, heart_width, heart_height, heart_bits[0]);break;
  }
  frame++;

    if(frame>15){
      frame = 0;
```

! uint8_t explicitly indicates that the variable (frame) is being used to store numerical values rather than characters (bytes)

unsigned char exactly 8 bits wide

? "static", internal linkage or static storage duration in this context? ↳ exists for the lifetime of the program? (don't understand -.)

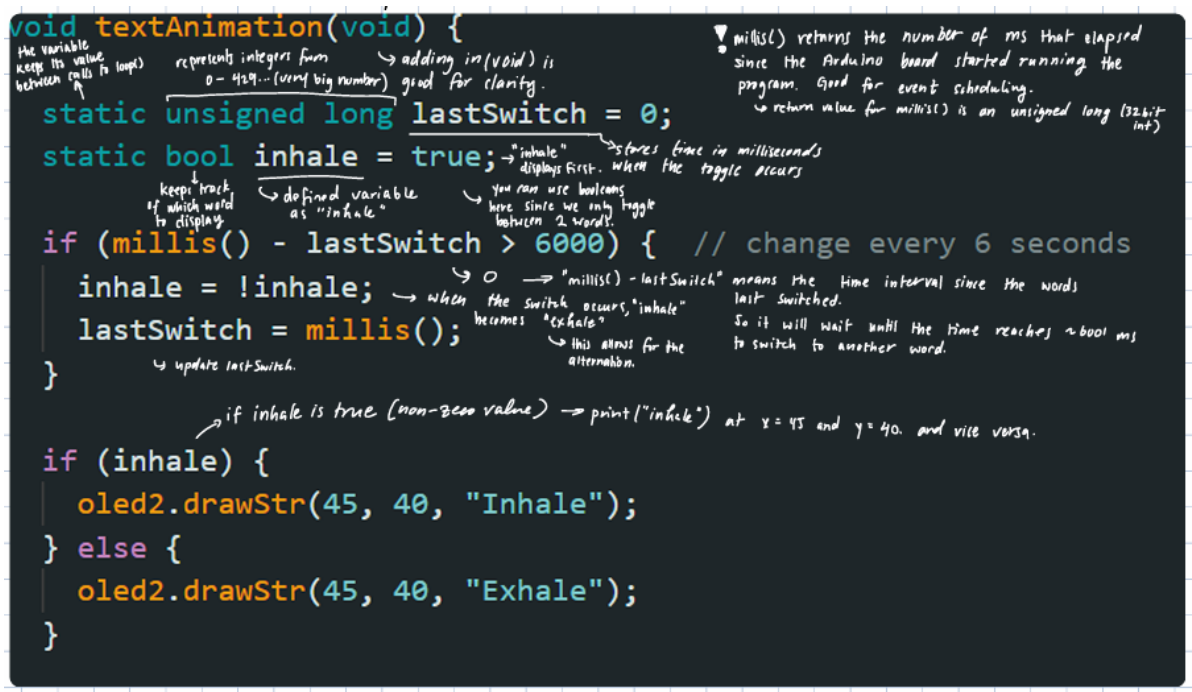
switch statement selects the byte arrays to be executed in order 0-15

x and y co-ordinates respectively

0th frame defined in the heart-bits[ ][ ] array

→ post increment operator (increases the variable by 1) effectively moving through all the frames

} reset to frame[0] condition



```
void textAnimation(void) {
  static unsigned long lastSwitch = 0;
  static bool inhale = true;

  if (millis() - lastSwitch > 6000) {  // change every 6 seconds
    inhale = !inhale;
    lastSwitch = millis();
  }

  if (inhale) {
    oled2.drawStr(45, 40, "Inhale");
  } else {
    oled2.drawStr(45, 40, "Exhale");
  }
```

the variable keeps its value between calls to loop()

represents integers from 0 - 429... (very big number)

↳ adding in (void) is good for clarity.

! millis() returns the number of ms that elapsed since the Arduino board started running the program. Good for event scheduling.
↳ return value for millis() is an unsigned long (32bit int)

→ "inhale" displays first.

↳ stores time in milliseconds when the toggle occurs

keeps track of which word to display

↳ defined variable as "inhale"

↳ you can use booleans here since we only toggle between 2 words.

↳ 0 → "millis() - lastSwitch" means the time interval since the words last switched.
So it will wait until the time reaches ~6001 ms to switch to another word.

→ when the switch occurs, "inhale" becomes "exhale"
↳ this allows for the alternation.

↳ update lastSwitch.

if inhale is true (non-zero value) → print("inhale") at x = 45 and y = 40. and vice versa.

To improve this, you could add a case where the OLED instructs you to hold your breath for 3-5 seconds in between inhaling and exhaling.