THE ClimaColour Code

Colour as Environmental Intelligence

Abstract

Environmental conditions are more often than not invisible and something that is felt rather than visually perceivable. The ClimaColour Code provides our spatial surroundings with their **own language** that enables them to better communicate with us as humans.

By translating climatic conditions, such as **temperature** and **humidity**, into light, the device communicates its climatic '**mood**' with a calm, pulsing glow that mirrors a heartbeat. Thus, expressing bio-climatic changes in a subtle, human way for people to understand; enabling us to connect with our spaces on a more **empathic** level.

Iaac

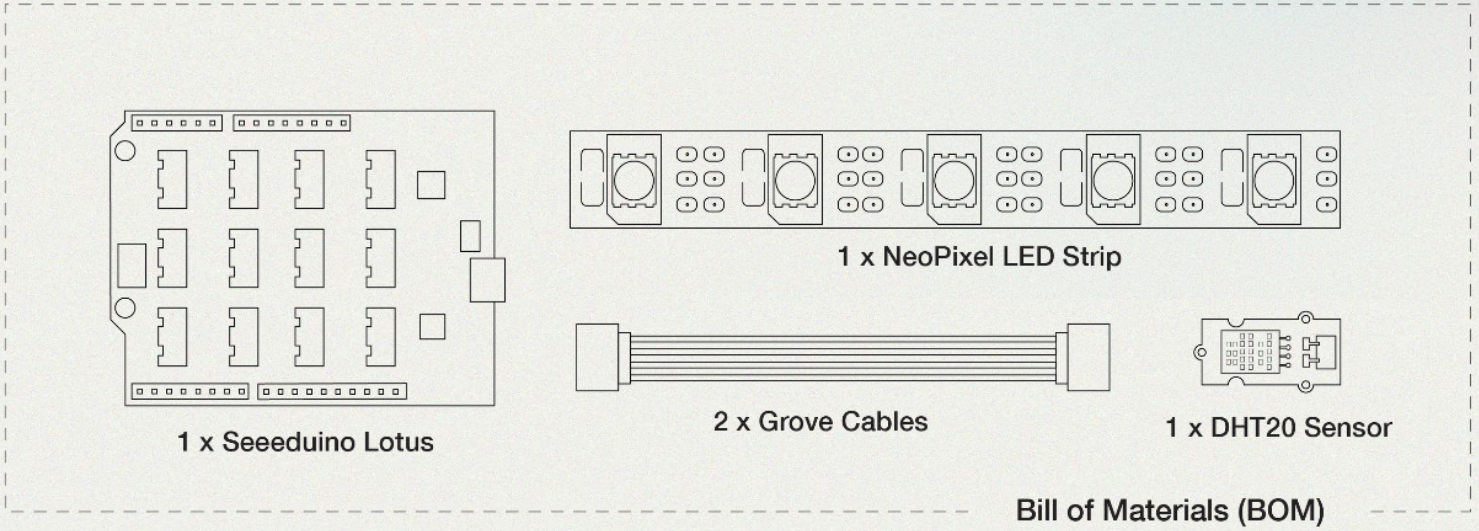
Concept

When a sensor reads temperature and humidity in real time, the ClimaColour Code converts that data into colour and a pulse behaviour in light.

The system logic follows human intuition that creates outputs based on visual cues we associate with certain conditions to show its ‘mood’.

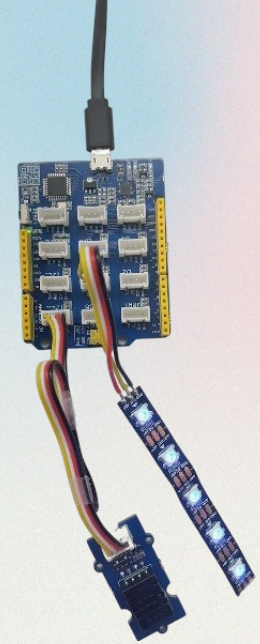
Bill of Materials (BOM)

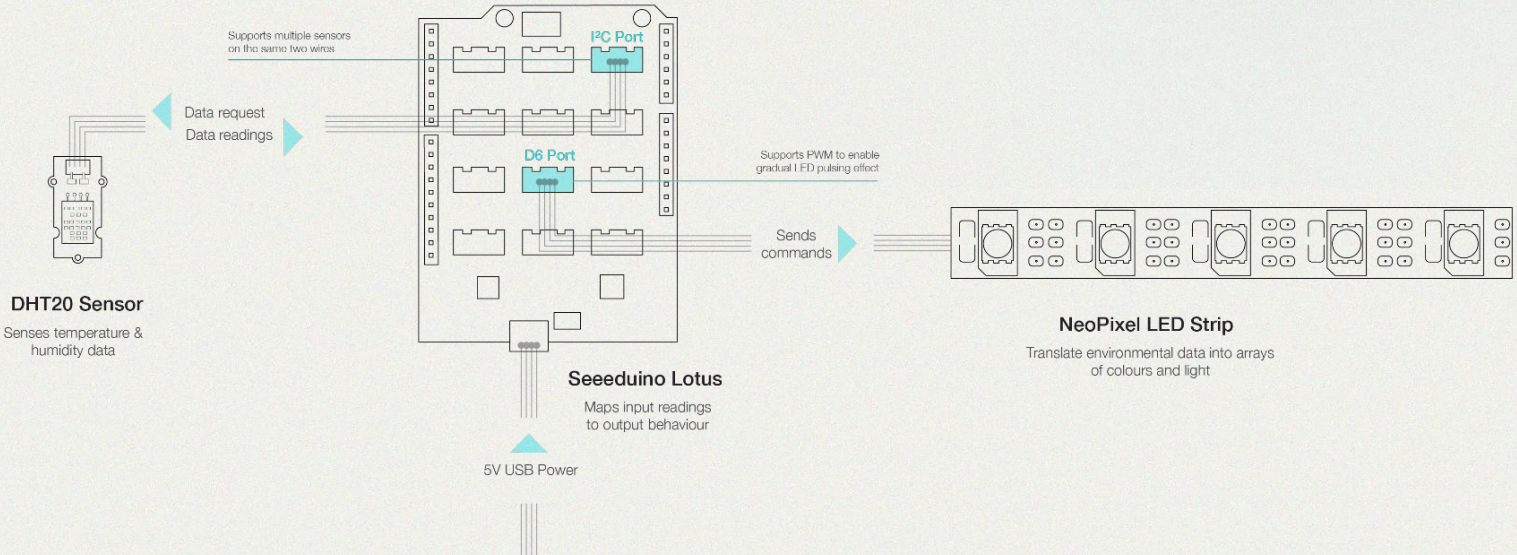
The Input–Process–Output Components



Schematic

The Input–Process–Output System

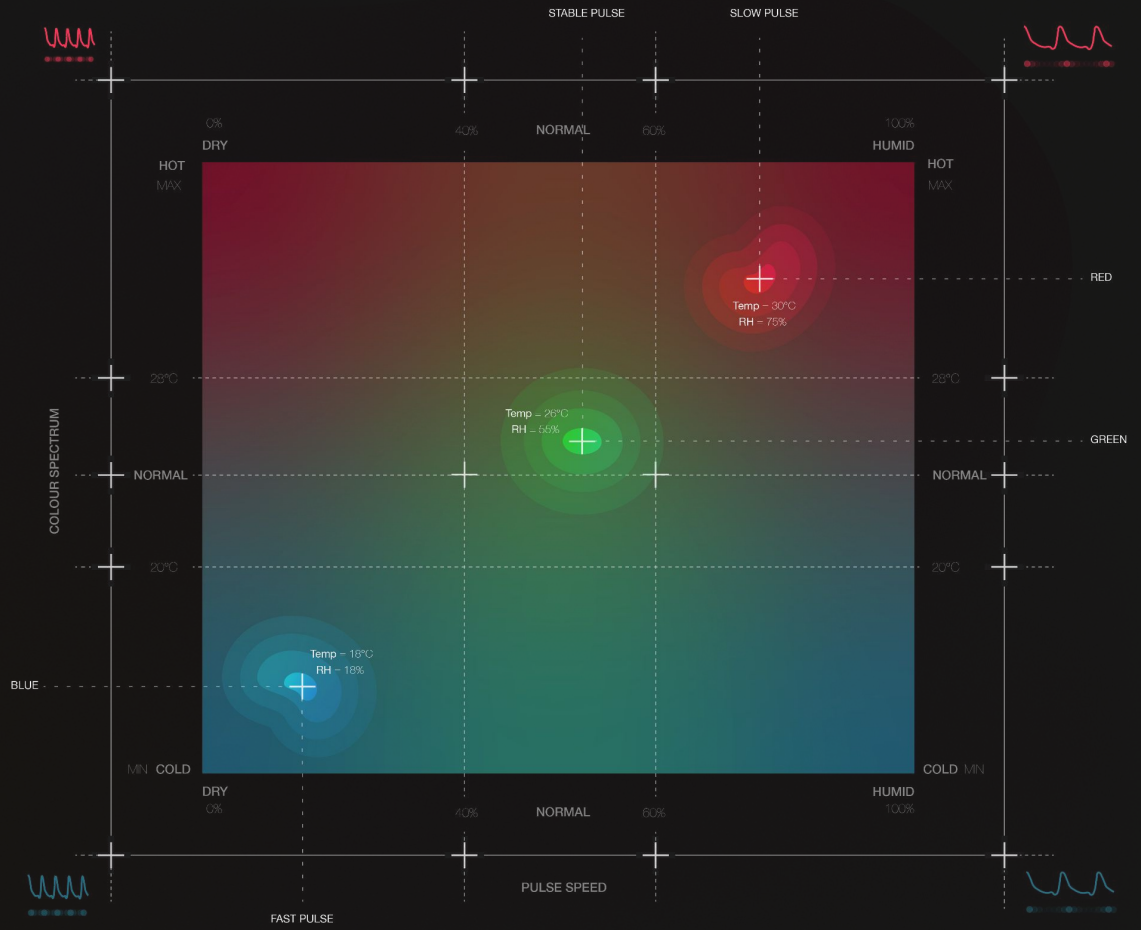


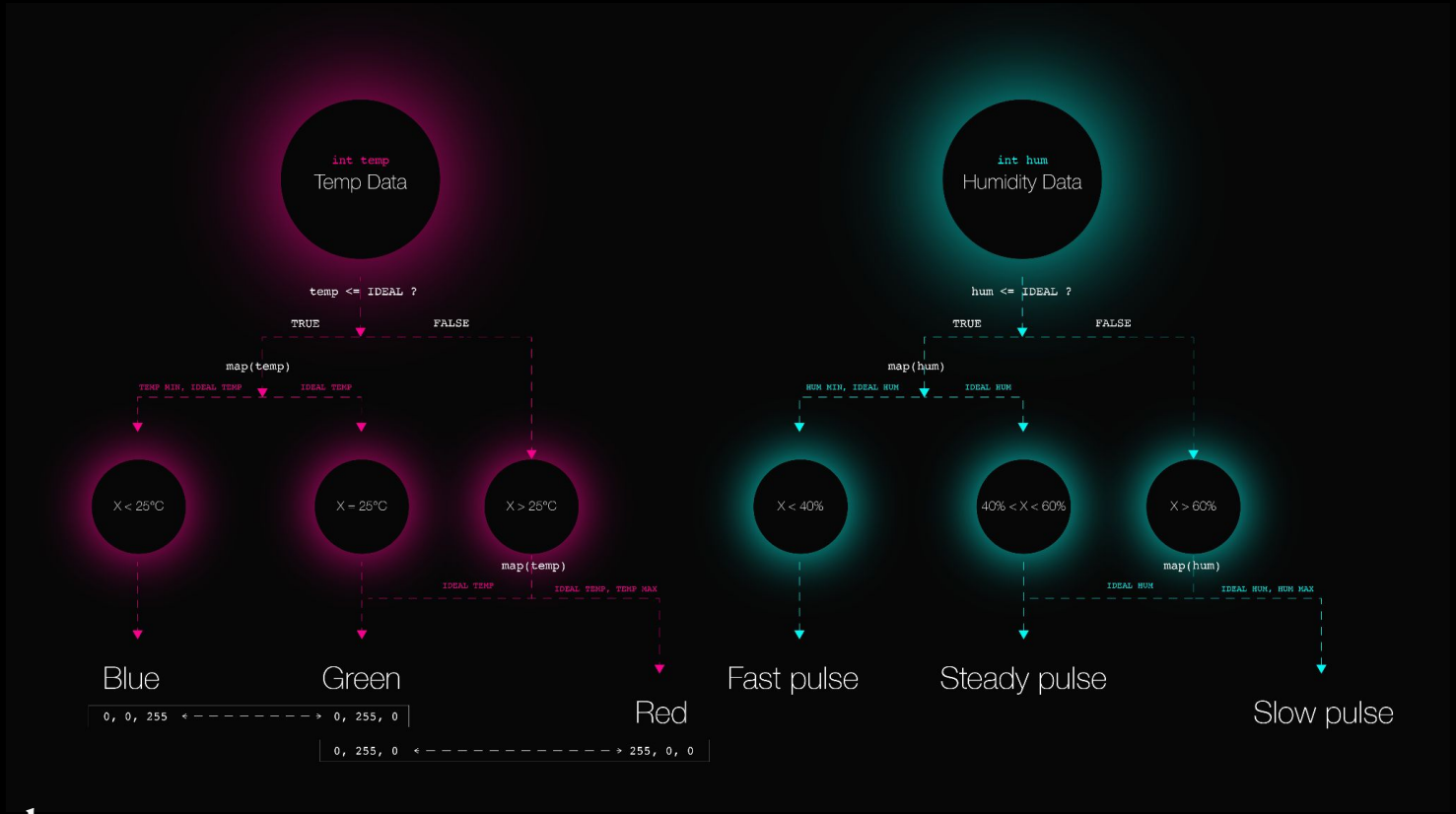


Iaac

Behaviour Matrix

Condition	Cold & Dry	Cold & Humid	Hot & Dry	Hot & Humid	Ideal (25° C)
Colour	Blue	Blue	Red	Red	Green
Bright Level	Low	High	Low	High	Normal
Pulse Speed	Slow	Slow	Fast	Fast	Steady
Mood	Fragile calm	Heavy calm	Stressed thirsty	Overwhelmed suffocating	Balanced and relaxed

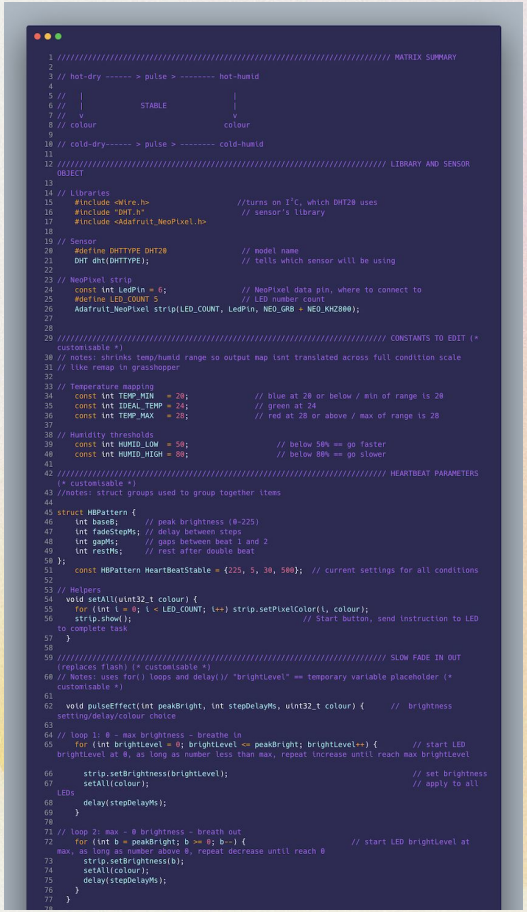




Script Flowchart

Thought Process Logic

Isaac



```
1  ////////////////////////////////////////////////////////////////////////////// MATRIX SUMMARY
2
3  // hot-dry ------ > pulse > -------- hot-humid
4
5  //   |                          |
6  //   |          STABLE          |
7  //   v                          v
8  // colour                     colour
9
10 // cold-dry------ > pulse > -------- cold-humid
11
12 ////////////////////////////////////////////////////////////////////////////// LIBRARY AND SENSOR OBJECT
13
14 // Libraries
15    #include <Wire.h>                  //turns on I²C, which DHT20 uses
16    #include "DHT.h"                   // sensor's library
17    #include <Adafruit_NeoPixel.h>
18
19 // Sensor
20    #define DHTTYPE DHT20              // model name
21    DHT dht(DHTTYPE);                  // tells which sensor will be using
22
23 // NeoPixel strip
24    const int LedPin = 6;              // NeoPixel data pin, where to connect to
25    #define LED_COUNT 5                // LED number count
26    Adafruit_NeoPixel strip(LED_COUNT, LedPin, NEO_GRB + NEO_KHZ800);
27
28
29 ////////////////////////////////////////////////////////////////////////////// CONSTANTS TO EDIT (* customisable *)
30 // notes: shrinks temp/humid range so output map isnt translated across full condition scale
31 // like remap in grasshopper
32
33 // Temperature mapping
34    const int TEMP_MIN   = 20;         // blue at 20 or below / min of range is 20
35    const int IDEAL_TEMP = 24;         // green at 24
36    const int TEMP_MAX   = 28;         // red at 28 or above / max of range is 28
37
38 // Humidity thresholds
39    const int HUMID_LOW  = 50;                 // below 50% == go faster
40    const int HUMID_HIGH = 80;                 // below 80% == go slower
41
42 ////////////////////////////////////////////////////////////////////////////// HEARTBEAT PARAMETERS (* customisable *)
43 //notes: struct groups used to group together items
44
45 struct HBPattern {
46    int baseB;     // peak brightness (0-225)
47    int fadeStepMs; // delay between steps
48    int gapMs;      // gaps between beat 1 and 2
49    int restMs;     // rest after double beat
50 };
51    const HBPattern HeartBeatStable = {225, 5, 30, 500};  // current settings for all conditions
52
53 // Helpers
54  void setAll(uint32_t colour) {
55    for (int i = 0; i < LED_COUNT; i++) strip.setPixelColor(i, colour);
56    strip.show();                            // Start button, send instruction to LED to complete task
57  }
58
59 ////////////////////////////////////////////////////////////////////////////// SLOW FADE IN OUT (replaces flash) (* customisable *)
60 // Notes: uses for() loops and delay()/ "brightLevel" == temporary variable placeholder (* customisable *)
61
62  void pulseEffect(int peakBright, int stepDelayMs, uint32_t colour) {      //  brightness setting/delay/colour choice
63
64 // loop 1: 0 - max brightness - breathe in
65    for (int brightLevel = 0; brightLevel <= peakBright; brightLevel++) {        // start LED brightLevel at 0, as long as number less than max, repeat increase until reach max brightLevel
66
67      strip.setBrightness(brightLevel);                                         // set brightness
68      setAll(colour);                                                           // apply to all LEDs
69      delay(stepDelayMs);
70    }
71
72 // loop 2: max - 0 brightness - breath out
73    for (int b = peakBright; b >= 0; b--) {                          // start LED brightLevel at max, as long as number above 0, repeat decrease until reach 0
74      strip.setBrightness(b);
75      setAll(colour);
76      delay(stepDelayMs);
77    }
78  }
```

```
79 ////////////////////////////////////////////////////////////////////////////// CREATE HEARTBEAT PATTERN (* customisable *)
80 //notes: uses struct group again
81
82  void heartbeat(const HBPattern& HBSettings, uint32_t colour) {                //  heartbeat(//"HeartBeatKit"/"HBSettings" == custom names of function - instructs: brightness, speed, rest time
83    pulseEffect(HBSettings.baseB, HBSettings.fadeStepMs, colour);                   // pulseEffect recalls to previous slow fade in fade out double loops
84    delay(HBSettings.gapMs);                                                        // Beat #1 + gap
85    pulseEffect(HBSettings.baseB, HBSettings.fadeStepMs, colour);                   // Beat #2 + rest
86    delay(HBSettings.restMs);
87  }
88
89 ////////////////////////////////////////////////////////////////////////////// START UP INSTRUCTION
90
91  void setup() {                                  // inittialisation
92    pinMode(LedPin, OUTPUT);                      // specifies this the pin is going to send signals
93    strip.begin();                                // wakes up LED strip connection
94    strip.clear();                                // starting point 0 for LED
95    strip.setBrightness(0);                       // starting point 0 for LED
96    strip.show();                                 // tells to do LED strip connection/here: communications says LEDs OFF
97
98    Serial.begin(115200);                         // Arduino and your laptop connection
99    Wire.begin();                                 // Start listening on the I²C wire/allow DHT sensor to send data to the Arduino
100   dht.begin();                                  // starts temperature and humidity sensor
101 }
102
103 ////////////////////////////////////////////////////////////////////////////// REPEAT CYCLE / CLAMP VALUES
104 // LOOP == sensor read > decide temp colour > decide humid pulse ==> create heartbeat
105
106 void loop() {
107   float dataReadings[2] = {0};                                  // array list with two slots: make two empty boxes
108   int err = dht.readTempAndHumidity(dataReadings);              // read DHT sensor, place values in box
109
110   if (err == 0) {
111     float humidity    = dataReadings[0];                        // float gives decimals
112     float temperature = dataReadings[1];
113
114     // Clamp
115     if (temperature < TEMP_MIN) temperature = TEMP_MIN;         // clamping, prevent glitch numbers creating weird results, safety cushion
116     if (temperature > TEMP_MAX) temperature = TEMP_MAX;
117     if (humidity < 0)  humidity = 0;
118     if (humidity > 100) humidity = 100;
119
120
121 ////////////////////////////////////////////////////////////////////////////// COLOUR MAPPING
122 // notes: conditional if/else and map() used
123 // map() == converts a number from temp range to colour range
124
125    int red = 0, green = 0, blue = 0;                                                // starting empty colour presets - all set brightness to 0
126
127    if (temperature <= IDEAL_TEMP) {                                                 // if under 25C, do the following:
128      blue  = map((int)temperature, TEMP_MIN,   IDEAL_TEMP, 255, 0);                 // blue will happen more if closer to min, and less closer to 25C
129      green = map((int)temperature, TEMP_MIN,   IDEAL_TEMP,   0, 255);               // green will happen more if closer to 25C, and less closer to min
130      red   = 0;                                                                     // red never happens in min - 25 range
131    } else {                                                                         // if above 25C, do the following:
132      green = map((int)temperature, IDEAL_TEMP, TEMP_MAX,   255, 0);                 // green will happen more closer to 25C, less closer to max
133      red   = map((int)temperature, IDEAL_TEMP, TEMP_MAX,     0, 255);               // red will happen more if closer to max, and less closer to 25C
134      blue  = 0;                                                                     // blue never happens in 25C - max range
135    }
136
137   uint32_t colourCode = strip.Color(red, green, blue);                             // ColourCode
138
139 ////////////////////////////////////////////////////////////////////////////// PULSE MAPPING
```

```
139 ////////////////////////////////////////////////////////////////////////////// PULSE MAPPING
140
141 // STABLE HEARTBEAT SETTINGS : copy values from struct into local variables to adjust
142    int peakBright = HeartBeatStable.baseB;
143    int stepDelay  = HeartBeatStable.fadeStepMs;
144    int restMs     = HeartBeatStable.restMs;
145    int gapMs      = HeartBeatStable.gapMs;
146
147
148  // DRY == High Pulse
149    const int DRY_PEAK =  50;                                  // how bright when dry (50 out of 255) - tiny breaths
150
151 if (humidity < HUMID_LOW) {
152    int dimPeak = map((int)humidity, 0, HUMID_LOW, 0, DRY_PEAK);
153    peakBright  = dimPeak;
154
155    int fasterStep  = map((int)humidity, 0, HUMID_LOW, 0, 0);
156    int shorterRest = map((int)humidity, 0, HUMID_LOW, 0, 0);
157    int extraGap    = map((int)humidity, 0, HUMID_LOW, 0, 0);
158
159    stepDelay = max(1, stepDelay - fasterStep);
160    restMs    = max(0, restMs - shorterRest);
161    gapMs     = gapMs + extraGap;
162  } else {
163    // normal humid → normal peak
164    peakBright = HeartBeatStable.baseB;
165  }
166
167
168  // HUMID == Low Pulse
169    if (humidity > HUMID_HIGH) {
170      int extraStep = map((int)humidity, HUMID_HIGH, 100, 0, 15);   // slower fade (0→+15ms)
171      int shorterGap = map((int)humidity, HUMID_HIGH, 100, 0, 15);  // reduce gap by up to 15ms
172      int extraRest = map((int)humidity, HUMID_HIGH, 100, 0, 400);  // still rest longer overall
173
174      stepDelay += extraStep;          // slower fade → more glow time
175      gapMs = max(0, gapMs - shorterGap);   // shorter "off" period between beats
176      restMs += extraRest;             // longer rest before next heartbeat
177    }
178
179
180 ////////////////////////////////////////////////////////////////////////////// SERIAL TERMINAL OUTPUT
181 String mood;
182 if (temperature < IDEAL_TEMP && humidity < HUMID_LOW) mood = "Dry & Cold 🥶";
183 else if (temperature < IDEAL_TEMP && humidity > HUMID_HIGH) mood = "Cold & Humid 🌧️";
184 else if (temperature > IDEAL_TEMP && humidity < HUMID_LOW) mood = "Hot & Dry 🔥";
185 else if (temperature > IDEAL_TEMP && humidity > HUMID_HIGH) mood = "Hot & Humid 🥵";
186 else mood = "Stable & Calm 💚";
187
188 Serial.print(" Temperature: "); Serial.print(temperature, 1); Serial.print(" °C  ");
189 Serial.print(" Humidity: "); Serial.print(humidity, 1); Serial.print(" % ");
190 Serial.println("→  Mood: "); Serial.println(mood);
191
192    HBPattern h = {peakBright, stepDelay, gapMs, restMs};
193    heartbeat(h, colourCode);
194
195  } else {
196    Serial.println("DHT20 read error");
197    strip.clear();
198    strip.setBrightness(0);
199    strip.show();
200    delay(200);
201  }
202 }
203
```

Key Themes

Environmental Intelligence

We care more about what we can see.
By visualising comfort and discomfort, our empathy will encourage us to make better choices regarding our energy use.

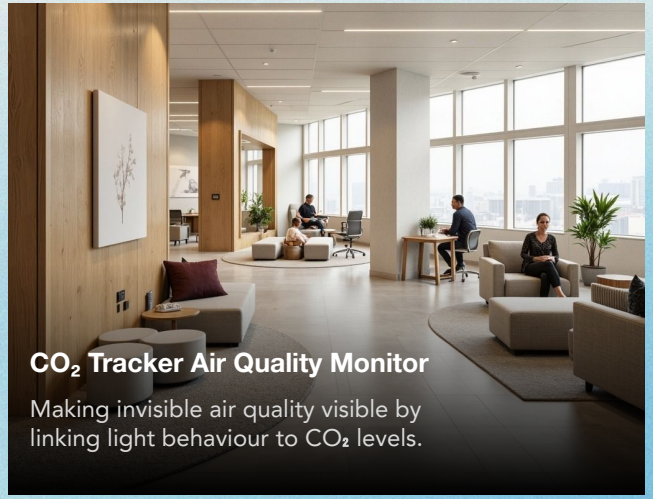
Joyful Experience

The pulsing light and varying colours allows our daily environments to take on different forms.

Emotional Connection

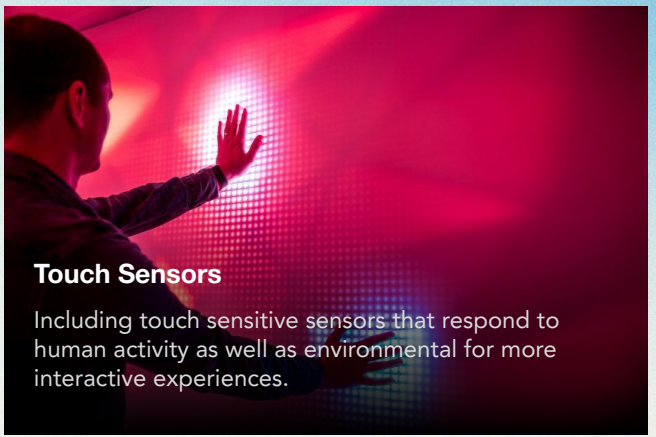
By giving spaces a mood and heartbeat, the system builds a relationship between us and the places we inhabit.

Potential Next Steps



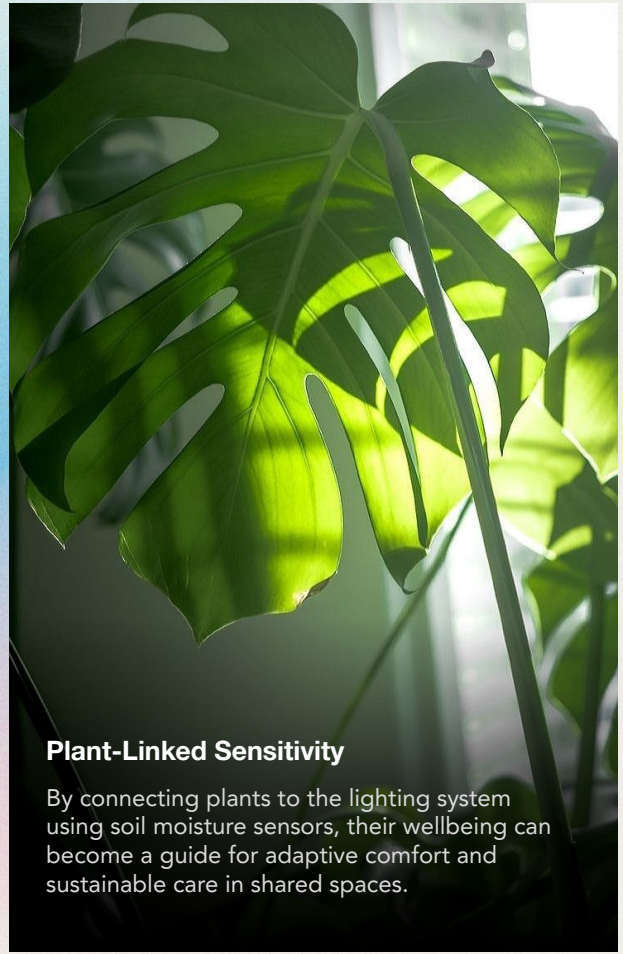
CO_2 Tracker Air Quality Monitor

Making invisible air quality visible by linking light behaviour to CO_2 levels.



Touch Sensors

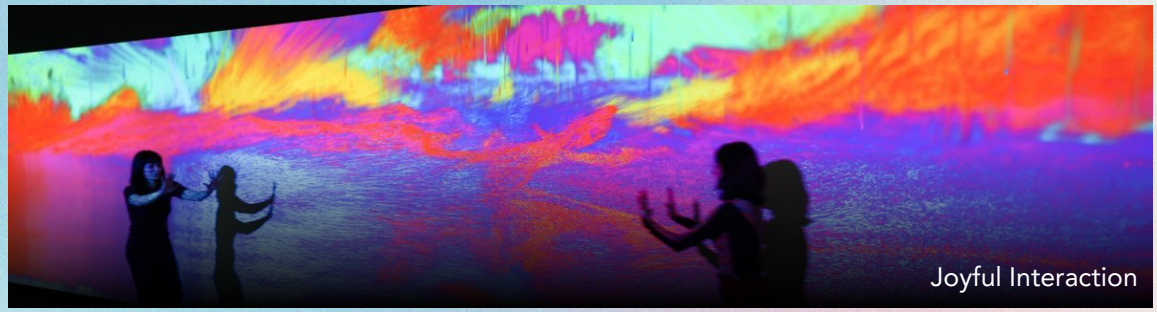
Including touch sensitive sensors that respond to human activity as well as environmental for more interactive experiences.

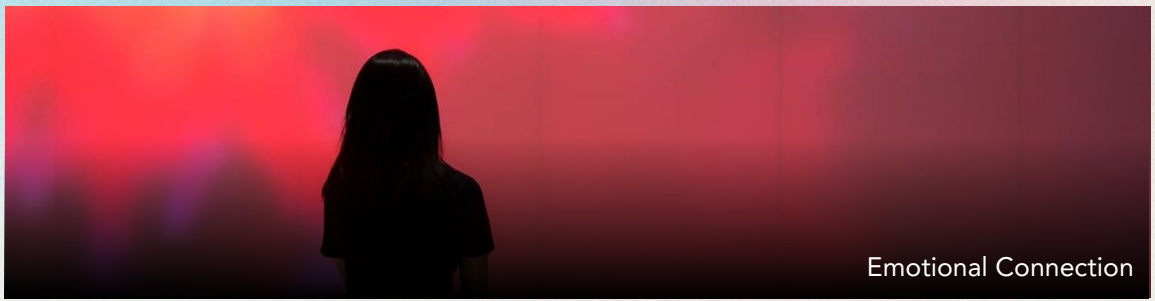


Plant-Linked Sensitivity

By connecting plants to the lighting system using soil moisture sensors, their wellbeing can become a guide for adaptive comfort and sustainable care in shared spaces.

Iaac





By exploiting human empathy using biological logic, the ClimaColour Code has the potential to make us more proactive in addressing how our environments are changing through increased awareness.

By taking the concept of interactive installations beyond purely entertainment value and making it more accessible in daily life, it allows us to experience more novelty and joy from our everyday spaces.

The Climacolour Code

By Yasmine MacCallum

A project developed as part of IAAC, Institute for Advanced Architecture of Catalonia

Faculty:
Daniel Mateos
Adai Suriñach
Antoine Jaunard