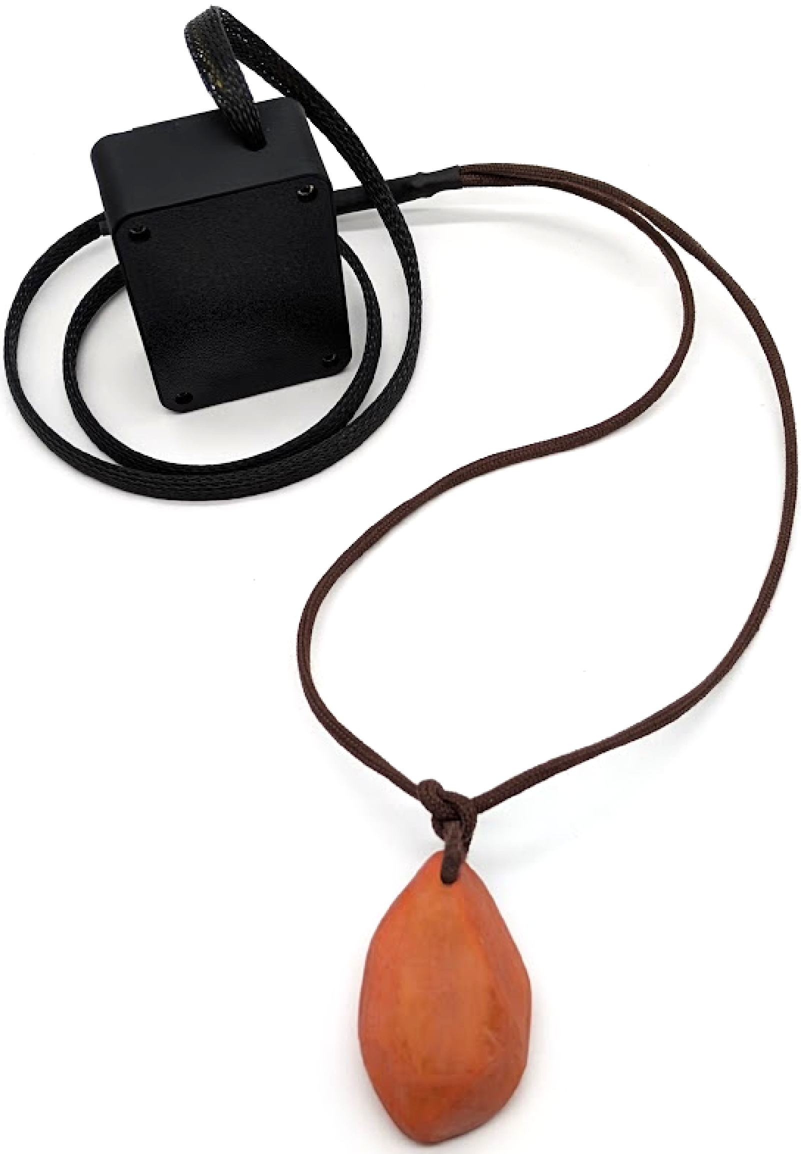
Glowing Necklace

User guide



Les
FRENCH TWINS
TONY & JORDAN

1. Device description

The necklace consists of a hollow resin gemstone housing an array of LEDs, a push button, and a magnet to attach a string for levitation. The power wires run through the necklace ropes, connecting to a control box that can be clipped onto a belt.

On the side of the control box, there is a power switch with the label "ON" indicating the powered position. When powering on the device, the green LED, located to the left of the USB-C port **blinks 3 times**.

2. Battery management

Inside, a rechargeable battery can be charged via the USB-C port. **To charge the battery, the device must be powered ON**. The green LED provides status feedback:

Slow blinking → Battery is charging.

Fast blinking → Charging is not possible (device is powered OFF).

Solid light → Battery is fully charged.

The device should be charged after 5 uses.

3. Safety considerations

The LED array inside the stone can get really hot if powered at 100% for more than a minute.

4. Programming

To program the device, an Arduino code is provided. The only file intended for modification is **"sequence.h"**, where you can customize the behavior as needed.

Quick Guide to Modify Animation Sequence in sequence.h

The animation sequence is defined as follows:

```
const AnimationStep sequence[] = {
   {20, 2.0},   // Go to 20% in 2 seconds
   {0, 2.0},    // Back to 0% in 2 seconds
   {50, 2.0},   // Go to 50% in 2 seconds
   {0, 2.0},    // Back to 0% in 2 seconds
   {100, 3.0},  // Go to 100% in 3 seconds
   {100, 3.0},  // Stay at 100% for 3 seconds
   {20, 2.0},   // Go to 20% in 2 seconds
   {100, 2.0},  // Go to 100% in 2 seconds
   {0, 2.0}     // Back to 0% in 2 seconds
};
```

How to Modify:

- **Ramp time**: The second value defines the time to reach the target percentage. For example, `{50, 2.0}` ramps up to 50% over 2 seconds.
- **To stay at a value**: To stay at a value (e.g., 100%) for 2 seconds, use `{100, 2.0}`. The ramp time means it will reach 100% and stay there for 2 seconds before moving to the next step.
- **Add/remove steps**: Add or remove `{targetPercent, durationSec}` pairs as needed.

Other Settings:

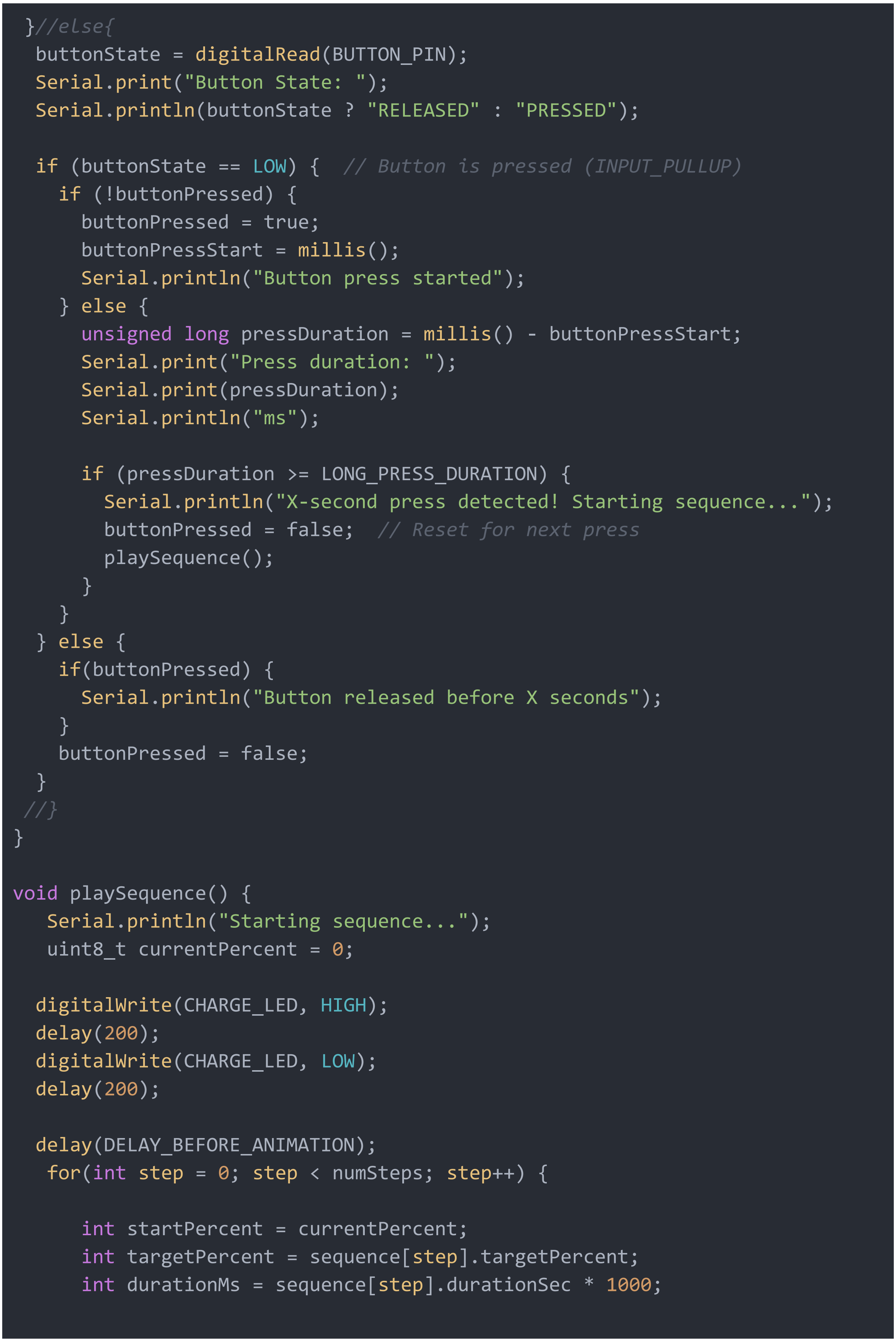
```
const unsigned long LONG_PRESS_DURATION = 2000; //Duration of button press
to trigger the sequence
const unsigned long DELAY_BEFORE_ANIMATION = 5000; //Delay before sequence
```

- **Button press duration:** Change LONG_PRESS_DURATION (default 2000 ms).
- **Delay before animation:** Change DELAY_BEFORE_ANIMATION (default 5000 ms).

Here’s the full code :

SEQUENCER.ino

```
// PROPERTY OF Rémi Lacombe 2025
#include "sequence.h"
// Pin Definitions
const int BATTERY_LEVEL = A0;
const int USB_PIN = D8;
const int CHARGE_LED = D10;
const int PWM_OUT = D4;
const int BUTTON_PIN = D5;

// Battery Configuration
const float BATTERY_MIN_CHARGE_VOLTAGE = 3.6;
const float BATTERY_MAX_CHARGE_VOLTAGE = 4.0;

// Define your sequence here - easy to edit

const int numSteps = sizeof(sequence) / sizeof(sequence[0]);

// PWM Configuration
const uint32_t LEDC_FREQ = 8000;
const uint8_t LEDC_RESOLUTION = 8;
const uint16_t FADE_STEP_DELAY_MS = 2;
const uint8_t MIN_BRIGHTNESS = 0;
const uint8_t MAX_BRIGHTNESS = 255;

unsigned long buttonPressStart = 0;
bool buttonPressed = false;

int buttonState = 0;
bool isUSBPlugged = false;

void setup() {
 Serial.begin(115200);
 Serial.println("\nStarting up...");
 pinMode(BATTERY_LEVEL, INPUT_PULLDOWN);
 pinMode(USB_PIN, INPUT_PULLDOWN);
 pinMode(CHARGE_LED, OUTPUT);
 pinMode(BUTTON_PIN, INPUT_PULLUP);

 ledcAttach(PWM_OUT, LEDC_FREQ, LEDC_RESOLUTION);
 ledcWrite(PWM_OUT, MIN_BRIGHTNESS);  // Start with LED off

 for(int i = 1; i <= 3; i++){
   digitalWrite(CHARGE_LED, HIGH);
   delay(200);
   digitalWrite(CHARGE_LED, LOW);
   delay(200);
 }

 Serial.println("Setup complete!");
}

void loop() {
 isUSBPlugged = digitalRead(USB_PIN);
 float battery_level = getVbatt();

 Serial.println("-------------------");
 Serial.print("Battery Level: ");
 Serial.print(battery_level, 3);
 Serial.println("V");
 Serial.println(isUSBPlugged);

 if(isUSBPlugged){
   if(battery_level >= BATTERY_MIN_CHARGE_VOLTAGE && battery_level <=
BATTERY_MAX_CHARGE_VOLTAGE){
     digitalWrite(CHARGE_LED, HIGH);
     Serial.println("Charge LED: ON (Battery charged)");
   } else if (battery_level > BATTERY_MAX_CHARGE_VOLTAGE){
     digitalWrite(CHARGE_LED, HIGH);
     delay(200);
     digitalWrite(CHARGE_LED, LOW);
     delay(200);
     Serial.println("Charge LED: FAST BLINK (Battery disconnected)");
   } else {
     digitalWrite(CHARGE_LED, HIGH);
     delay(1000);
     digitalWrite(CHARGE_LED, LOW);
     delay(1000);
     Serial.println("Charge LED: SLOW BLINK (Battery outside charge
range)");
   }
```



```
 }//else{
  buttonState = digitalRead(BUTTON_PIN);
  Serial.print("Button State: ");
  Serial.println(buttonState ? "RELEASED" : "PRESSED");

  if (buttonState == LOW) {  // Button is pressed (INPUT_PULLUP)
    if (!buttonPressed) {
      buttonPressed = true;
      buttonPressStart = millis();
      Serial.println("Button press started");
    } else {
      unsigned long pressDuration = millis() - buttonPressStart;
      Serial.print("Press duration: ");
      Serial.print(pressDuration);
      Serial.println("ms");

      if (pressDuration >= LONG_PRESS_DURATION) {
        Serial.println("X-second press detected! Starting sequence...");
        buttonPressed = false;  // Reset for next press
        playSequence();
      }
    }
  } else {
    if(buttonPressed) {
      Serial.println("Button released before X seconds");
    }
    buttonPressed = false;
  }
 //}
}

void playSequence() {
   Serial.println("Starting sequence...");
   uint8_t currentPercent = 0;

  digitalWrite(CHARGE_LED, HIGH);
  delay(200);
  digitalWrite(CHARGE_LED, LOW);
  delay(200);

  delay(DELAY_BEFORE_ANIMATION);
   for(int step = 0; step < numSteps; step++) {

      int startPercent = currentPercent;
      int targetPercent = sequence[step].targetPercent;
      int durationMs = sequence[step].durationSec * 1000;
      // Calculate number of steps for smooth transition
      int numSteps = 1000;  // Update 100 times during the transition
      int stepDelay = durationMs / numSteps;
      for(int i = 0; i <= numSteps; i++) {
          currentPercent = map(i, 0, numSteps, startPercent, targetPercent);
          // Convert percent to PWM value
          uint8_t pwmValue = map(currentPercent, 0, 100, MIN_BRIGHTNESS,
MAX_BRIGHTNESS);
          ledcWrite(PWM_OUT, pwmValue);
          delay(stepDelay);
      }
  }
}

float getVbatt() {
 uint32_t Vbatt = 0;
 for(int i = 0; i < 16; i++) {
   Vbatt = Vbatt + analogReadMilliVolts(BATTERY_LEVEL);
 }
 float Vbattf = 2 * Vbatt / 16 / 1000.0;
 return Vbattf;
}
```

sequence.h :

```
// PROPERTY OF Rémi Lacombe 2025
// Animation sequence definition
struct AnimationStep {
   uint8_t targetPercent;  // 0-100%
   float durationSec;      // Duration in seconds
};

const AnimationStep sequence[] = {
   {20, 2.0},   // Go to 20% in 2 seconds
   {0, 2.0},    // Back to 0% in 2 seconds
   {50, 2.0},
   {0, 2.0},
   {100, 3.0},
   {100, 3.0}, //Stay at 100% for 3 seconds
   {20, 2.0},
   {100, 2.0},
   {0, 2.0}
};

const unsigned long LONG_PRESS_DURATION = 2000; //Duration of button press
to trigger the sequence
```

```
const unsigned long DELAY_BEFORE_ANIMATION = 5000; //Delay before sequence
```