```
#include <Arduino.h>

#include <WiFi.h>

#include <WiFiClientSecure.h>

#include <HTTPClient.h>

#include <ArduinoJson.h>

#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>
```

---

1. #include <Arduino.h>

- **What it is:** This is the **core Arduino library** that gives you access to Arduino functions like pinMode(), digitalWrite(), delay(), etc.
- **Why needed:** ESP32 sketches rely on this to use standard Arduino functions. Without it, basic setup/loop won't work.

---

2. #include <WiFi.h>

- **What it is:** ESP32's official **Wi-Fi library**.
- **What it does:** Lets your ESP32 connect to Wi-Fi networks as a client or access point.
- **Why needed:** Your bot connects to Wi-Fi to fetch motivational quotes/messages from an online API.

---

3. #include <WiFiClientSecure.h>

- **What it is:** A library that extends WiFiClient to support **SSL/TLS encrypted connections** (HTTPS).
- **What it does:** Enables ESP32 to talk securely to web servers (like https://api.something.com).
- **Why needed:** Most APIs today require **HTTPS**, not plain HTTP. This library ensures data is encrypted.

---

4. #include <HTTPClient.h>

- **What it is:** High-level library to simplify making **HTTP/HTTPS requests** (GET, POST, etc.).
- **What it does:** Instead of writing raw network code, you can just do http.begin(url) → http.GET() → http.getString().
- **Why needed:** You'll use this to **fetch motivational quotes/messages** from the API.

5. #include <ArduinoJson.h>

- **What it is:** A popular Arduino library to handle **JSON data**.
- **What it does:** Lets you parse JSON responses from web APIs and extract values (like "quote": "Stay positive").
- **Why needed:** API responses are usually in JSON format, so this library is key for reading them.

---

6. #include <Wire.h>

- **What it is:** The **I^{2}C communication library**.
- **What it does:** Lets ESP32 communicate with devices using I^{2}C protocol (like sensors, OLED screens, etc.).
- **Why needed:** Your **OLED display** communicates with ESP32 via I^{2}C (SDA/SCL pins).

---

7. #include <Adafruit_GFX.h>

- **What it is:** Adafruit's **graphics core library**.
- **What it does:** Provides drawing functions (lines, shapes, fonts, text).
- **Why needed:** The OLED screen needs this for text/graphics rendering. Without it, you can't draw text or shapes.

---

8. #include <Adafruit_SSD1306.h>

- **What it is:** A driver library specifically for **SSD1306 OLED displays**.
- **What it does:** Works with Adafruit_GFX to control OLED screens (initialize, clear, print text, draw).
- **Why needed:** Your **0.91" 128x64 OLED** is SSD1306-based, so this is the main display driver.

✅ In short:

- **Arduino basics:** Arduino.h
- **Wi-Fi + HTTPS:** WiFi.h, WiFiClientSecure.h, HTTPClient.h
- **API parsing:** ArduinoJson.h
- **Display handling:** Wire.h, Adafruit_GFX.h, Adafruit_SSD1306.h

---

```
const char* WIFI_SSID     = "Tenda_2A6998";   // Your Wi-Fi SSID
const char* WIFI_PASSWORD = "9496744880";    // Your Wi-Fi password
// ======================== Perplexity API setup =====================
const char* PPLX_ENDPOINT = "https://api.perplexity.ai/chat/completions"; // REST endpoint
const char* PPLX_MODEL    = "sonar"; // Model name (try "sonar-chat" if 400)
const char* PPLX_API_KEY  = "pplx-AXg6o1w0QDDV"; //
```

```
const char* PPLX_ENDPOINT = "https://api.perplexity.ai/chat/completions";
```

What it is: A constant C-string (const char*) storing the API endpoint URL.

What it does: This is the server address where your ESP32 will send HTTP requests.

Why needed: When the ESP32 uses HTTPClient, it needs to know the exact URL to send requests to.

```
const char* PPLX_MODEL    = "sonar"; // try "sonar-chat" if 400 persists
```

What it is: A string that sets the model name for the API.

What it does: The Perplexity API lets you choose which AI model to query (like “sonar” or “sonar-chat”).

Why needed: When you make a request, you must tell the API which model should generate the response.

Note: If you get a 400 error (bad request), it may mean the model name is invalid, so you can switch to "sonar-chat".

```
const char* PPLX_API_KEY  = "pplx-AXg6o1w0QDDVW4ni1A";
```

What it is: Your Perplexity API key, stored as a string.

What it does: This key authenticates your ESP32 with Perplexity’s servers — basically proving you’re allowed to use their service.

Why needed: Without it, the API will reject your request with an unauthorized (401) error.

✅ **Block Summary:**

These three constants (PPLX_ENDPOINT, PPLX_MODEL, PPLX_API_KEY) tell your ESP32 where to send API requests, which model to use, and how to authenticate with Perplexity AI

```
// = OLED (SSD1306 128x64 I2C) =

#define SCREEN_WIDTH 128

#define SCREEN_HEIGHT 64

#define OLED_I2C_ADDR 0x3C

#define I2C_SDA_PIN   4

#define I2C_SCL_PIN   5

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
```

```
#define SCREEN_WIDTH 128

#define SCREEN_HEIGHT 64
```

- **What it is:** Preprocessor constants (macros).
- **Why:** Defines the **pixel resolution** of your OLED:
    - 128 pixels wide (columns)
    - 64 pixels tall (rows)
- Used by the Adafruit library to correctly allocate display memory.

```
#define OLED_I2C_ADDR 0x3C
```

- **What it is:** The **I^{2}C address** of the OLED module.
- **Why:** Every I^{2}C device has an address so the ESP32 knows which one it's talking to.
- **0x3C** is the most common address for 128×64 SSD1306 OLEDs.
  (Sometimes it's 0x3D, depending on the module.)

```
#define I2C_SDA_PIN   4

#define I2C_SCL_PIN   5
```

- **What it is:** Tells ESP32 which pins to use for **I^2C communication**:
    - SDA (data line) → GPIO 4
    - SCL (clock line) → GPIO 5
- These pins must match how your OLED is wired.
- You can change them if you connect to different pins.

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

- **What it is:** Creates an **Adafruit_SSD1306 display object**.
- **Parameters explained:**
    1. SCREEN_WIDTH → 128
    2. SCREEN_HEIGHT → 64
    3. &Wire → tells it to use the **Wire (I^2C) library** for communication
    4. -1 → no reset pin (many OLED modules don't have a dedicated reset pin, so -1 is used).

Now you can use commands like display.begin(), display.clearDisplay(), display.print(), etc., to draw text/graphics on the screen.

✅ **Block Summary:**

This block **defines the display size, I^2C address, and pins**, then creates an Adafruit_SSD1306 object. It's the setup needed so your ESP32 can talk to the OLED screen and print text/graphics.

```
// ======================== IR gesture pins ==========================

const int IR_LEFT_PIN  = 34;          // LEFT sensor GPIO (happy)

const int IR_RIGHT_PIN = 35;          // RIGHT sensor GPIO (tired)


// ======================== UX timings (ms) ==========================

const unsigned long DEBOUNCE_MS   = 80;    // IR debounce

const unsigned long COOLDOWN_MS   = 1200;  // Cooldown between triggers
const unsigned long SHOW_MS       = 9000;  // Quote display duration
const unsigned long HAPPY_SPIN_MS = 900;   // Spin time (~360°)
const unsigned long FWD_MS        = 1000;  // Forward move time
const unsigned long STOP_MS       = 1000;  // Pause stop time
unsigned long lastTrig = 0;               // Last trigger timestamp
```

✅ **Block Summary:**

- Two IR sensors on pins **34 (left → happy)** and **35 (right → tired)**.
- Timers ensure smooth actions:
  - DEBOUNCE_MS = ignore tiny false signals.
  - COOLDOWN_MS = wait between valid triggers.
  - SHOW_MS = how long to show the AI reply.
  - HAPPY_SPIN_MS, FWD_MS, STOP_MS = motor movement durations.
- lastTrig remembers the last time an IR event happened.

```
// ============== Idle big-eyes + blink animation state ==============
unsigned long idleTimer = 0;            // Pace idle breathing
unsigned long blinkTimer = 0;           // Blink timing
bool    blinkActive = false;          // Check mid-blink
uint8_t  blinkPhase  = 0;              // Blink phase

// Eye geometry helper (centers two eyes on screen)
struct EyeGeom {
  int w = 40;          // Eye width
  int h = 36;          // Eye height
  int r = 8;           // Corner radius
  int gap = 10;      // Gap between eyes
  int leftX, leftY, rightX, rightY;                  // Computed positions
  void computeCentered() {
    int totalW = (w * 2) + gap;                      // Total width of both eyes + gap
    leftX  = (SCREEN_WIDTH  - totalW) / 2;           // Left eye X
    rightX = leftX + w + gap;                        // Right eye X
    leftY  = (SCREEN_HEIGHT - h) / 2;                // Both eyes Y
    rightY = leftY;
  }
} eyes;                                              // Global instance

// Draw a single eye rectangle; supports partial height for blinking
void drawBigEye(int x, int y, int w, int h, int r, float blinkFrac, int offsetX, int offsetY){
  int hh = h;                                        // Effective height
  if (blinkFrac > 0) {                               // If blinking, reduce height
    int closePix = (int)(h * blinkFrac);             // Pixels closed
    hh = max(1, h - closePix);                       // Keep at least 1px
    y += closePix / 2;                               // Center the slit vertically
  }
  x += offsetX; y += offsetY;                        // Apply gaze offsets
  if (x < -w || y < -hh || x > SCREEN_WIDTH || y > SCREEN_HEIGHT) return; // Off-screen guard
  display.fillRoundRect(x, y, w, hh, r, SSD1306_WHITE); // Draw filled rounded eye
}

// Draw top eyelids (black triangles) for "tired" look
void drawTiredLids(int x, int y, int w, int h, int r, int lidH){
  lidH = constrain(lidH, 0, h/2);                    // Clamp lid height
  if (lidH <= 0) return;                             // Nothing to draw
  display.fillTriangle(x,   y-1, x+w, y-1, x,   y+lidH-1, SSD1306_BLACK); // Left edge
  display.fillTriangle(x,   y-1, x+w, y-1, x+w, y+lidH-1, SSD1306_BLACK); // Right edge
}

// Draw bottom eyelid mask for "happy" smiley eyes
void drawHappyBottomLid(int x, int y, int w, int h, int r, int lift){
  lift = constrain(lift, 0, h);                  // Clamp
  if (lift <= 0) return;                         // Nothing to draw
  display.fillRoundRect(x-1, (y + h) - lift + 1, w+2, h, r, SSD1306_BLACK); // Mask bottom
}

// Compose a face: mood 0=neutral, 1=tired, 2=happy; blink & gaze offsets
void drawFaceMood(uint8_t mood, float blinkFrac, int gazeX, int gazeY){
  display.clearDisplay();                        // Clear frame
  // Base eyes (with blink & gaze)
  drawBigEye(eyes.leftX,  eyes.leftY,  eyes.w, eyes.h, eyes.r, blinkFrac, -gazeX, gazeY);
  drawBigEye(eyes.rightX, eyes.rightY, eyes.w, eyes.h, eyes.r, blinkFrac,  gazeX,  gazeY);
  // Mood overlays
  if (mood == 1) {                               // Tired: add droopy lids
    int lidH = eyes.h/3;                         // Lid height
    drawTiredLids(eyes.leftX,  eyes.leftY,  eyes.w, eyes.h, eyes.r, lidH);
    drawTiredLids(eyes.rightX, eyes.rightY, eyes.w, eyes.h, eyes.r, lidH);
  } else if (mood == 2) {                        // Happy: lifted bottom lids
    int lift = eyes.h/3;
    drawHappyBottomLid(eyes.leftX,  eyes.leftY,  eyes.w, eyes.h, eyes.r, lift);
    drawHappyBottomLid(eyes.rightX, eyes.rightY, eyes.w, eyes.h, eyes.r, lift);
  }
  display.display();                    // Push to OLED
}

// Idle animation: breathing (vertical), slow gaze pan, random blinking
void updateIdleAnimation(){
  static int breath = 0, bdir = 1;              // Breathing offset & direction
  if (millis() - idleTimer >= 40) {             // Update ~25 FPS
    idleTimer = millis();
    breath += bdir;                             // Move offset
    if (breath > 3)  bdir = -1;                 // Reverse at bounds
    if (breath < -3) bdir = 1;
  }

  static int gx = 0, gdir = 1;                  // Gaze X offset & direction
  static unsigned long gazeT = 0;               // Gaze timer
  if (millis() - gazeT >= 100) {                // Update every 100 ms
    gazeT = millis();
    gx += gdir;                                 // Pan horizontally
    if (gx > 4)  gdir = -1;                     // Bounce at edges
    if (gx < -4) gdir = 1;
  }

  // Trigger random blinks every 3–7s
  if (!blinkActive && millis() - blinkTimer > 3000UL + (unsigned long)random(0, 4000)) {
    blinkActive = true;                         // Start blink
    blinkPhase  = 0;                            // Reset phase
    blinkTimer  = millis();                     // Timestamp
  }
  float blinkFrac = 0.0f;                    // Default: open
  if (blinkActive) {                         // If blinking, set fraction 0..1..0
    if (blinkPhase <= 8) blinkFrac = blinkPhase / 8.0f;          // Closing
    else            blinkFrac = max(0.0f, 1.0f - (blinkPhase-8)/8.0f); // Opening
    if (millis() - blinkTimer >= 20) {          // Advance every 20 ms
      blinkTimer = millis();
      blinkPhase++;
      if (blinkPhase > 16) {                 // Done
        blinkActive = false;                 // Stop blink
        blinkFrac = 0.0f;                    // Eyes open
      }
    }
  }

  drawFaceMood(0, blinkFrac, gx, breath);          // Render neutral face
}

// ========================= Small status text =========================
void showSmall(const String& a, const String& b=""){
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);             // White text
  display.setTextSize(1);                          // Small font
  display.setCursor(0,0);                          // Top-left
  display.println(a);                              // First line
  if (b.length()) display.println(b);              // Optional second line
  display.display();                               // Show
}
```

```
// =============== Quote rendering (word-wrap helper) ==================
bool renderWrapped(uint8_t sz, int topY, const String& text){
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(sz);                       // 1 or 2 (double-size)
  int lineH = 8*sz + (sz==1 ? 2 : 4);            // Line height with padding
  int maxC  = (sz==2 ? 10 : 20);                 // Max chars per line (rough)
  int y = topY, start = 0;                       // Cursor & text index
  while (start < (int)text.length() && y <= SCREEN_HEIGHT - lineH){
    int len = min(maxC, (int)text.length() - start);  // Tentative span
    int br = -1;                                 // Breakpoint (space)
    for (int i = len; i > 0; --i){               // Scan backward for space
      if (text.charAt(start + i - 1) == ' ') { br = i - 1; break; }
    }
    if (br < 0) br = len;                        // If none, hard break
    String line = text.substring(start, start + br);  // Extract line
    int16_t bx, by; uint16_t bw, bh;             // Measure line width
    display.getTextBounds(line, 0, 0, &bx, &by, &bw, &bh);
    int x = (SCREEN_WIDTH - bw) / 2;             // Center horizontally
    if (x < 0) x = 0;                            // Clamp
    display.setCursor(x, y);                     // Move cursor
    display.print(line);                         // Draw line
    y += lineH;                                  // Next line
    start += br;                                 // Advance index
    while (start < (int)text.length() && text.charAt(start) == ' ') start++; // Skip spaces
  }
  display.display();
  return start >= (int)text.length();               // True if all text printed
}

// Render a quote big; fallback to small if it doesn’t fit
void showQuoteReadable(const String& t){
  if (!t.length()) { showSmall("No text"); return; } // Guard empty
  display.clearDisplay();
  if (!renderWrapped(2, 4, t))                     // Try size 2 at y=4
    renderWrapped(1, 0, t);                        // Else size 1 full height
}

// ======================== IR helper (active-LOW) ====================
bool triggeredLow(int pin){
  if (digitalRead(pin) == LOW) {                   // First detect LOW
    delay(DEBOUNCE_MS);                            // Debounce wait
    return digitalRead(pin) == LOW;                // Confirm still LOW
  }
  return false;                                    // Not triggered
}

// ======================== Prompt templates =========================
const char* happyPrompts[] = {                     // Variations for happy
  "Fresh upbeat micro-quote, simple words, 6-9 words.",
  "Short cheerful line, plain words, 6-9 words.",
  "Quick uplifting boost, simple words, 6-9 words."
};
const char* tiredPrompts[] = {                     // Variations for tired
  "Supportive micro-quote for tired mood, simple words, 6-9 words.",
  "Gentle encouragement for fatigue, plain words, 6-9 words.",
  "Kind, calm nudge for rest, simple words, 6-9 words."
};

// =========== Robust extractor for the Perplexity response ============
String extractFirstText(JsonVariantConst root){
  if (!root.containsKey("choices")) return String(); // Must have choices
  JsonVariantConst choices = root["choices"];        // Get choices
  if (!choices.is<JsonArrayConst>()) return String(); // Expect array
  JsonArrayConst arr = choices.as<JsonArrayConst>();  // Cast array
  if (arr.size() == 0) return String();              // Empty guard
  JsonVariantConst c0 = arr[0];                       // First choice

  // Common fields: message.content (string) or text
  if (c0["message"]["content"].is<const char*>())
    return String(c0["message"]["content"].as<const char*>());
  if (c0["text"].is<const char*>())
    return String(c0["text"].as<const char*>());

  // Sometimes content is an array of blocks; search for text fields
  if (c0["message"]["content"].is<JsonArrayConst>()) {
    JsonArrayConst blocks = c0["message"]["content"].as<JsonArrayConst>();
    for (JsonVariantConst b : blocks) {
      if (b["text"].is<const char*>())               // Direct text field
        return String(b["text"].as<const char*>());
      if (b["type"].is<const char*>() &&              // Typed block with data.text
          String(b["type"].as<const char*>()) == "text" &&
          b["data"]["text"].is<const char*>()) {
```

```
        return String(b["data"]["text"].as<const char*>());
      }
    }
  }
  return String();                              // Not found
}

// ======================== Networking (HTTPS) ========================
String fetchQuoteOnline(const String& topic, int& httpCodeOut){
  httpCodeOut = -1;                                // Default code
  if (WiFi.status() != WL_CONNECTED) return String(); // Need Wi-Fi

  WiFiClientSecure client;                          // TLS socket
  client.setInsecure();                            // Skip cert validation (dev mode)
  HTTPClient http;                                 // HTTP wrapper
  http.setReuse(false);                            // No keep-alive reuse
  http.setTimeout(15000);                            // 15s timeout
  if (!http.begin(client, PPLX_ENDPOINT)) return String(); // Init URL

  http.addHeader("Content-Type", "application/json"); // JSON body
  http.addHeader("Authorization", String("Bearer ") + PPLX_API_KEY); // Auth header

  String nonce = String((uint32_t)random(0xFFFFFFFF), HEX); // Random nonce to vary
prompts

  const int HN = sizeof(happyPrompts) / sizeof(happyPrompts[0]); // Count happy prompts
  const int TN = sizeof(tiredPrompts) / sizeof(tiredPrompts);    // Count tired prompts
  const char* prompt = (topic == "happy")
```

```
    ? happyPrompts[random(0, HN)]                 // Pick random happy prompt
    : tiredPrompts[random(0, TN)];                // Or tired prompt

  StaticJsonDocument<640> req;                    // Build request JSON
  req["model"]       = PPLX_MODEL;                // Model name
  req["temperature"] = 0.9;                       // More creative
  req["max_tokens"]  = 24;                        // Short reply
  JsonArray msgs = req.createNestedArray("messages"); // Chat messages array

  JsonObject m1 = msgs.createNestedObject();         // System message
  m1["role"]    = "system";                       // Role
  m1["content"] = "Reply ONLY with one very short line (<=10 words). No author, no quotes."; // Style guard

  JsonObject m2 = msgs.createNestedObject();         // User message
  m2["role"]    = "user";                         // Role
  m2["content"] = String(prompt) + " nonce=" + nonce; // Prompt + nonce

  String payload; serializeJson(req, payload);       // Serialize JSON to string
  delay(5);                                       // Tiny yield
  int code = http.POST(payload);                  // POST request
  httpCodeOut = code;                             // Output status

  String out;                                     // Response text
  if (code == 200) {                              // Success
    String body = http.getString();               // Read body
    StaticJsonDocument<3072> doc;                 // Parse buffer
    DeserializationError err = deserializeJson(doc, body); // Parse JSON
    if (!err) {
      out = extractFirstText(doc.as<JsonVariantConst>());  // Extract text
    }
  }
  http.end();                                  // Close connection

  out.trim();                                  // Clean whitespace
  if (out == "null" || out == "(null)") out = "";   // Sanitize nulls
  if (out.startsWith("\"") && out.endsWith("\"") && out.length() >= 2)
    out = out.substring(1, out.length() - 1);        // Strip quotes
  while (out.endsWith(".") || out.endsWith("!") || out.endsWith("?"))
    out.remove(out.length() - 1);                  // Remove trailing punctuation
  out.trim();
  return out;                                  // Final text (may be empty)
}

// ======================== MOTOR CONTROL ==============================
// Driver wiring:
//  Motor A (left):  IN1=27, IN2=26, ENA=14
//  Motor B (right): IN3=25, IN4=33, ENB=32
const int IN1 = 27, IN2 = 26, ENA = 14;             // Left motor pins
const int IN3 = 25, IN4 = 33, ENB = 32;             // Right motor pins
const int CH_A = 0, CH_B = 1;                       // PWM channels
const int FREQ = 20000;                             // 20 kHz PWM (quiet)
const int RES  = 8;                                 // 8-bit duty (0..255)

// Speed presets (0..255)
uint8_t DUTY_FWD  = 200;                            // Forward speed
```

```
uint8_t DUTY_REV  = 200;                    // Reverse speed
uint8_t DUTY_SPIN = 210;                    // Spin speed

// Stop left motor
void motorA_stop(){
  digitalWrite(IN1, LOW);                 // Disable H-bridge input 1
  digitalWrite(IN2, LOW);                 // Disable input 2 (coast)
  ledcWrite(CH_A, 0);                    // Zero PWM
}

// Stop right motor
void motorB_stop(){
  digitalWrite(IN3, LOW);                 // Disable input 3
  digitalWrite(IN4, LOW);                 // Disable input 4
  ledcWrite(CH_B, 0);                    // Zero PWM
}

// Left forward with dead-time before enabling PWM
void motorA_forward(uint8_t d){
  ledcWrite(CH_A, 0);                    // Ensure PWM off
  digitalWrite(IN1, HIGH);                // Set direction forward
  digitalWrite(IN2, LOW);
  delayMicroseconds(200);                   // Dead-time (reduce shoot-through)
  ledcWrite(CH_A, d);                    // Apply duty
}

// Left reverse
void motorA_reverse(uint8_t d){
  ledcWrite(CH_A, 0);
  digitalWrite(IN1, LOW);                    // Reverse direction
  digitalWrite(IN2, HIGH);
  delayMicroseconds(200);
  ledcWrite(CH_A, d);
}

// Right forward
void motorB_forward(uint8_t d){
  ledcWrite(CH_B, 0);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  delayMicroseconds(200);
  ledcWrite(CH_B, d);
}

// Right reverse
void motorB_reverse(uint8_t d){
  ledcWrite(CH_B, 0);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  delayMicroseconds(200);
  ledcWrite(CH_B, d);
}

// Initialize motor pins & PWM
void motorsSetup(){
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);      // Left H-bridge inputs
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);       // Right inputs


  ledcSetup(CH_A, FREQ, RES);                  // Configure PWM A
  ledcSetup(CH_B, FREQ, RES);                  // Configure PWM B
  ledcAttachPin(ENA, CH_A);                    // ENA -> CH_A
  ledcAttachPin(ENB, CH_B);                    // ENB -> CH_B


  motorA_stop();                               // Start stopped
  motorB_stop();
}


// Convenience wrappers
inline void motorsStop(){      motorA_stop(); motorB_stop(); }
inline void motorsForward(){   motorA_forward(DUTY_FWD);
motorB_forward(DUTY_FWD); }
inline void motorsBackward(){  motorA_reverse(DUTY_REV);  motorB_reverse(DUTY_REV); }
inline void motorsSpinRight(){ motorA_forward(DUTY_SPIN); motorB_reverse(DUTY_SPIN);
}
inline void motorsSpinLeft(){  motorA_reverse(DUTY_SPIN); motorB_forward(DUTY_SPIN); }


// ============================== setup() ================================
void setup(){
  randomSeed(analogRead(34) ^ millis());          // Seed RNG (ADC noise ^ time)


  pinMode(IR_LEFT_PIN,  INPUT_PULLUP);            // IR inputs (active-LOW)
  pinMode(IR_RIGHT_PIN, INPUT_PULLUP);


  Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);           // I²C start with custom pins
  Wire.setClock(400000);                    // 400 kHz fast-mode
  display.begin(SSD1306_SWITCHCAPVCC, OLED_I2C_ADDR); // Init OLED

  eyes.computeCentered();                    // Position eyes
  updateIdleAnimation();                     // Draw first frame

  motorsSetup();                             // Init motors

  WiFi.mode(WIFI_STA);                       // Station mode
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);           // Connect to AP
  unsigned long t0 = millis();               // Start timer
  while (WiFi.status() != WL_CONNECTED && millis() - t0 < 15000) { // Wait up to 15s
    delay(200);                              // Poll interval
  }
}

// ============================== loop() ===============================
void loop(){
  updateIdleAnimation();                     // Keep eyes alive

  if (millis() - lastTrig < COOLDOWN_MS) {           // Enforce cooldown
    delay(20);                               // Short idle
    return;                                  // Skip triggers
  }

  // ---------------- LEFT sensor => Happy routine ----------------
  if (triggeredLow(IR_LEFT_PIN)){
    lastTrig = millis();                     // Stamp last trigger
    unsigned long tFace = millis();                  // Show happy face briefly
    while (millis() - tFace < 600) {                 // ~600 ms
      drawFaceMood(2, 0.0f, 2, -1);                  // mood=2 (happy)
      delay(16);                                     // ~60 FPS
    }

    motorsSpinRight();                               // Celebrate spin
    delay(HAPPY_SPIN_MS);                            // Spin duration
    motorsStop();                                    // Stop
    delay(150);                                      // Settle

    int code = 0;                                    // HTTP code holder
    String q = fetchQuoteOnline("happy", code);      // Get upbeat quote
    if (q.length()) {                                // If success
      display.clearDisplay();
      showQuoteReadable(q);                          // Show nicely wrapped
      delay(SHOW_MS);                                // Keep on screen
    } else {
      showSmall(String("HTTP ") + code, "No quote"); // Show error
      delay(1200);
    }

    updateIdleAnimation();                           // Refresh idle face
    return;                                          // Avoid checking RIGHT this loop
  }
```

```
  // --------------- RIGHT sensor => Tired routine ----------------
  if (triggeredLow(IR_RIGHT_PIN)){
    lastTrig = millis();                          // Stamp last trigger


    unsigned long tFace = millis();               // Show tired face briefly
    while (millis() - tFace < 600) {
      drawFaceMood(1, 0.0f, -2, 1);               // mood=1 (tired)
      delay(16);
    }
    motorsForward(); delay(FWD_MS);               // Forward
    motorsStop();    delay(STOP_MS);              // Pause
    motorsForward(); delay(FWD_MS);               // Forward again
    motorsStop();    delay(150);                  // Settle
    int code = 0;                                 // HTTP code holder
    String q = fetchQuoteOnline("tired", code);   // Get supportive quote
    if (q.length()) {
      display.clearDisplay();
      showQuoteReadable(q);                       // Show nicely wrapped
      delay(SHOW_MS);
    } else {
      showSmall(String("HTTP ") + code, "No quote"); // Show error
      delay(1200);
    }
    updateIdleAnimation();                        // Refresh idle face
    return;                                       // Avoid extra work
  }
  delay(15);                                      // Small idle delay
}
```