Desk Assistant Documentation

Project Description:

Create a desk assistant that can control aspects of your room and assist with tasks on your desk while you are working. Use Arduino hardware and 3D print all designs

Version 2.0:

Notes:

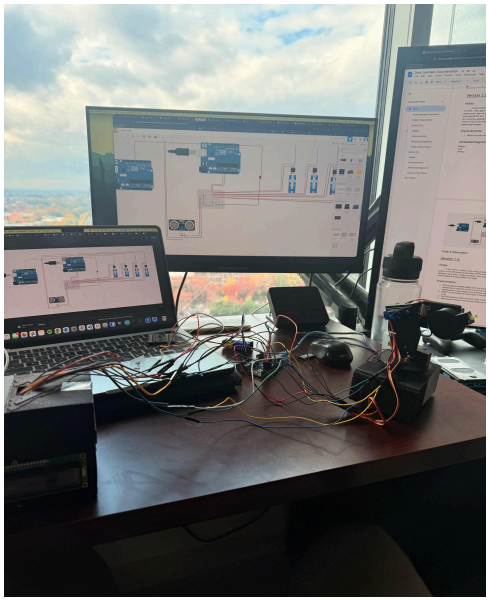
In this version I add a wire management base to put my phone on and organize wires on my desk. I also add a servo robotic arm to the top of the original design. It has an ultrasonic sensor and a light on the end of the arm. When the sensor detects a wave it turns on the light and extends the arm. When it detects another wave it retracts the arm and turns off the light. I am using two separate arduino units to make this design work. While designing this version I fixed my 3D printer setup so that it no longer needs constant supervision to ensure filament doesn't get tangled.

Improvements:

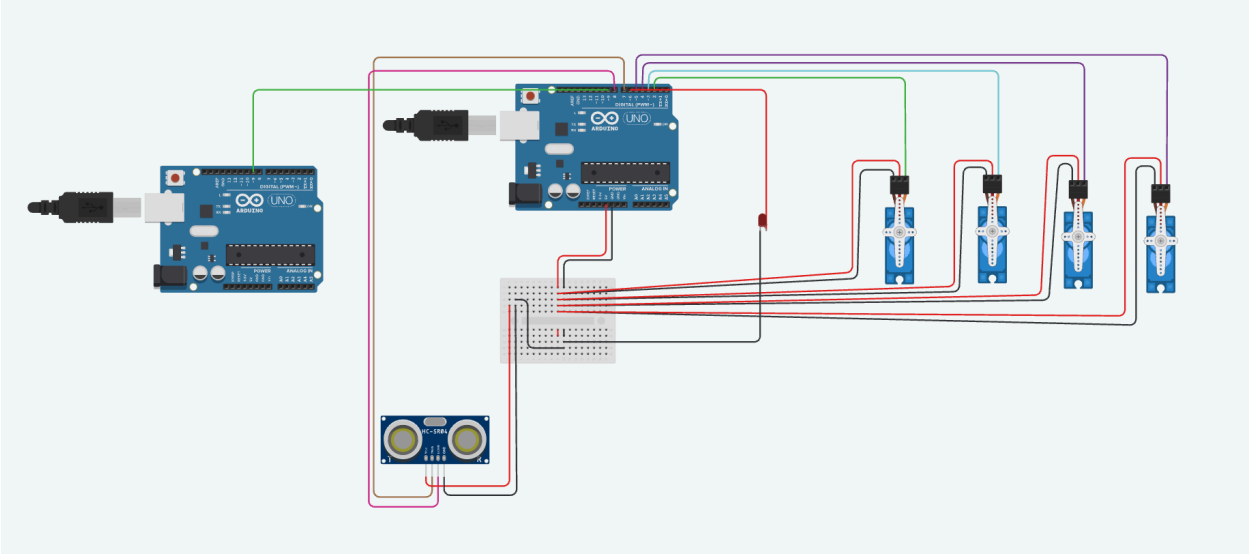
1. Make it so that the entire system runs off of only one arduino.

Attributes/Diagrams:

Design:



Wiring:



Code & Description:

```
#include <Servo.h>

// pins
const uint8_t PIN_BASE    = 2;
const uint8_t PIN_BOTTOM  = 3;
const uint8_t PIN_MIDDLE  = 4;
const uint8_t PIN_TOP     = 5;
const uint8_t PIN_LED     = 6;
const uint8_t PIN_TRIG    = 7;
const uint8_t PIN_ECHO    = 8;
const uint8_t PIN_SIGNAL  = 9;

// objects
Servo baseServo, bottomServo, middleServo, topServo;

// positions (tune)
const int baseIdle = 90;
const int bottomIdle = 140;
const int middleIdle = 70;
const int topIdle = 90;

const int baseExtend  = 110;
const int bottomExtend = 30;
const int middleExtend = 0;
const int topExtend    = 50;

// motion tuning
const int STEP_DEG = 1;
const unsigned long STEP_INTERVAL_MS = 18; // adjust: lower = faster

// ultrasonic detection tuning
const int PRESENCE_DIST_CM = 5;      // close presence threshold
const int WAVE_DELTA_CM = 6;          // single-delta threshold
const unsigned long WAVE_WINDOW_MS = 700; // time window to count wave deltas
const unsigned long WAVE_DEBOUNCE_MS = 900; // min time between toggles

// extra ignore after motion completes (ms)
const unsigned long IGNORE_AFTER_MOTION_MS = 400;
```

```
// state
bool systemOn = false;
bool extended = false;
bool moving = false; // true during motion
int curBase, curBottom, curMiddle, curTop;

// timing
unsigned long lastStepMillis = 0;
unsigned long lastToggleMillis = 0;
unsigned long lastUltrasonicMillis = 0;
const unsigned long ULTRA_PERIOD_MS = 60; // sample period

// wave counting
unsigned long lastWaveDeltaTime = 0;
int waveCount = 0;

// ignore window while moving/settling
unsigned long ignoreUntil = 0;

// ultrasonic smoothing
int lastValidDist = -1;

void setup() {
  Serial.begin(9600);

  baseServo.attach(PIN_BASE);
  bottomServo.attach(PIN_BOTTOM);
  middleServo.attach(PIN_MIDDLE);
  topServo.attach(PIN_TOP);

  pinMode(PIN_SIGNAL, INPUT);
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);

  curBase = baseIdle;
  curBottom = bottomIdle;
```

```
curMiddle = middleIdle;
curTop = topIdle;
writeAll();

digitalWrite(PIN_LED, LOW);
lastStepMillis = millis();
lastToggleMillis = 0;
lastUltrasonicMillis = 0;


oid loop() {
int signal = digitalRead(PIN_SIGNAL);

if (signal == HIGH) {
  systemOn = true;
} else {
  systemOn = false;
}

if (systemOn) {
  unsigned long now = millis();

  // 1) ultrasonic sampling (periodic)
  if (now - lastUltrasonicMillis >= ULTRA_PERIOD_MS) {
    lastUltrasonicMillis = now;
    int dist = readUltrasonicCm();
    if (dist > 0) handleUltrasonic(dist, now);
  }

  // 2) step servos
  if (now - lastStepMillis >= STEP_INTERVAL_MS) {
    lastStepMillis = now;
    updateMotionStep();
  }
} else {
  digitalWrite(PIN_LED, LOW);
}
```

```
// read HC-SR04 (returns cm, -1 on timeout)
int readUltrasonicCm() {
  digitalWrite(PIN_TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);
  unsigned long dur = pulseIn(PIN_ECHO, HIGH, 30000UL); // 30 ms timeout
  if (dur == 0) return -1;
  return (int)(dur / 58UL);
}

/* Improved handler:
   - If we're in an ignore window, do nothing.
   - Presence triggers immediate candidate.
   - Wave requires two large deltas within WAVE_WINDOW_MS to count as a wave.
   - Toggle only if enough time passed since last toggle and not currently moving.
   - Start motion sets ignoreUntil = now + motionDuration + IGNORE_AFTER_MOTION_MS.
*/
void handleUltrasonic(int dist, unsigned long now) {
  // ignore while moving or in ignore period
  if (now < ignoreUntil) {
    // Serial.println("Ignoring ultrasonic (in ignoreUntil).");
    lastValidDist = dist; // update anyway to keep delta logic sane
    return;
  }

  // compute delta
  if (lastValidDist > 0) {
    int delta = abs(dist - lastValidDist);
    if (delta >= WAVE_DELTA_CM) {
      // record wave delta time and increment count
      if (now - lastWaveDeltaTime <= WAVE_WINDOW_MS) {
        waveCount++;
      } else {
        waveCount = 1;
      }
```

```
      waveCount = 1;
    }
    lastWaveDeltaTime = now;
    // Serial.print("delta "); Serial.print(delta); Serial.print(" waveCount "); Serial.println(waveCount);
  }
}
lastValidDist = dist;

bool presence = (dist > 0 && dist <= PRESENCE_DIST_CM);
bool waveDetected = (waveCount >= 1); // require 2 deltas within window

if ((presence && waveDetected) && !moving && (now - lastToggleMillis >= WAVE_DEBOUNCE_MS)) {
  // toggle
  extended = !extended;
  lastToggleMillis = now;
  digitalWrite(PIN_LED, moving ? HIGH : LOW);
  moving = true;
  Serial.print("Toggling extended="); Serial.print(extended); Serial.print(" dist="); Serial.println(dist);

  // estimate motion duration (rough): steps * interval
  int maxDelta = 0;
  maxDelta = max(maxDelta, abs(curBottom - (extended?bottomExtend:bottomIdle)));
  maxDelta = max(maxDelta, abs(curMiddle - (extended?middleExtend:middleIdle)));
  maxDelta = max(maxDelta, abs(curTop - (extended?topExtend:topIdle)));
  // add a margin
  unsigned long estimatedMotionMs = (unsigned long)maxDelta * STEP_INTERVAL_MS + 80;
  ignoreUntil = now + estimatedMotionMs + IGNORE_AFTER_MOTION_MS;

  // reset waveCount so subsequent motion doesn't use stale deltas
  waveCount = 0;
  lastWaveDeltaTime = 0;
}
```

```
/ step servos toward targets (1 deg per step)
oid updateMotionStep() {
 int targetBase   = extended ? baseExtend   : baseIdle;
 int targetBottom = extended ? bottomExtend : bottomIdle;
 int targetMiddle = extended ? middleExtend : middleIdle;
 int targetTop    = extended ? topExtend    : topIdle;

 bool anyMoving = false;
 if (stepToward(curBase, targetBase)) anyMoving = true;
 if (stepToward(curBottom, targetBottom)) anyMoving = true;
 if (stepToward(curMiddle, targetMiddle)) anyMoving = true;
 if (stepToward(curTop, targetTop)) anyMoving = true;

 writeAll();

 if (!anyMoving) {
   if (moving) {
     moving = false; // finished motion
     Serial.println(extended ? "Extend complete" : "Retract complete");
     // note: ignoreUntil was already set when motion started
   }
 }


/ helper: move by up to STEP_DEG; return true if still moving after step
ool stepToward(int &cur, int target) {
 if (cur < target) {
   int delta = min(STEP_DEG, target - cur);
   cur += delta;
   return cur != target;
 } else if (cur > target) {
   int delta = min(STEP_DEG, cur - target);
   cur -= delta;
   return cur != target;
 }
```

```
  return false;


oid writeAll() {
  baseServo.write(curBase);
  bottomServo.write(curBottom);
  middleServo.write(curMiddle);
  topServo.write(curTop);
```

Version 1.0:

Notes:

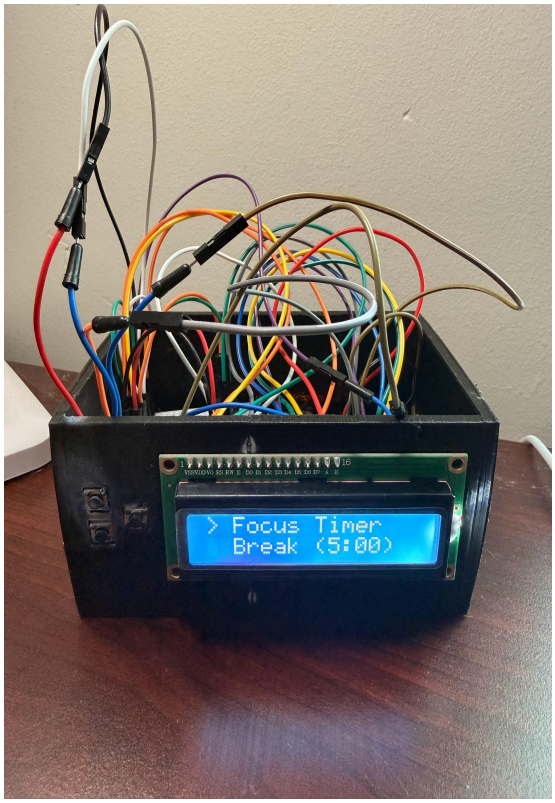
In this version I create a menu using an arduino and lcd screen and three buttons. The buttons on the left control up and down and the button on the right is selected. So far the menu includes a work timer, a break countdown, and motivational quotes. It is all housed within a 3d printed casing that was printed in two parts.

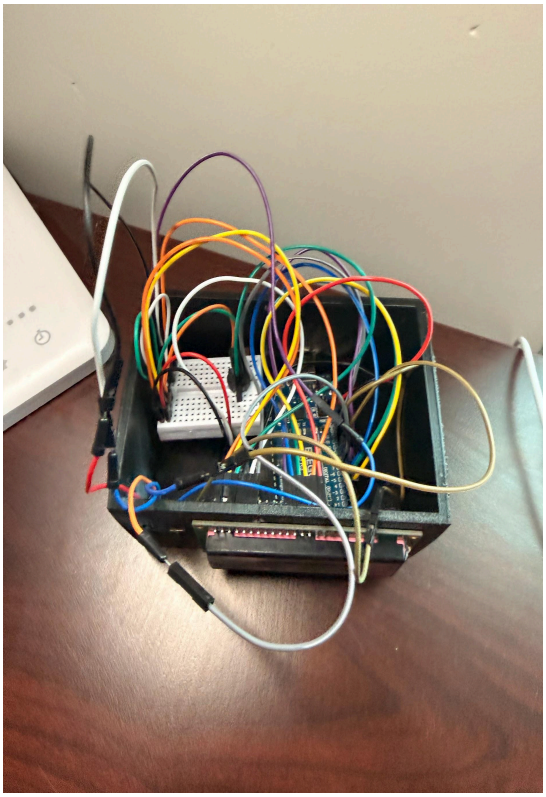
Improvements:

1. Sink all arduino models slightly more into the 3D print. This model did not account for pins/bumps on the back so both uno and lcd are slightly too raised. This will be addressed for a later version since the prints still work with adjustments.
2. Make it so that buttons fit more snugly in the frame. Goal should be to not use superglue at all
3. Get PCB screws to attach all boards firmly to the frame
4. Adjust printer setting so raft removes easier and cleaner

Attributes/Diagrams:

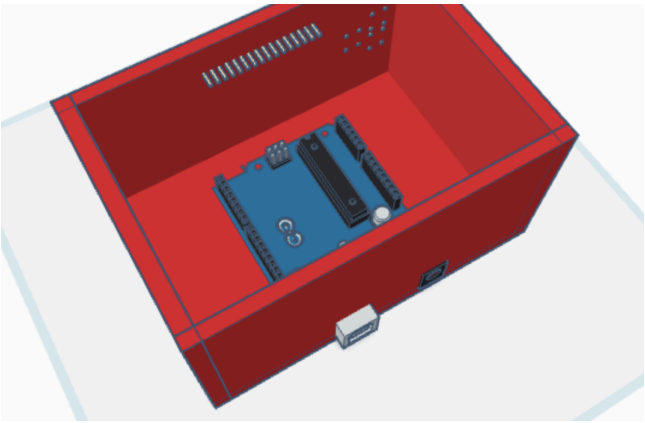
Design:



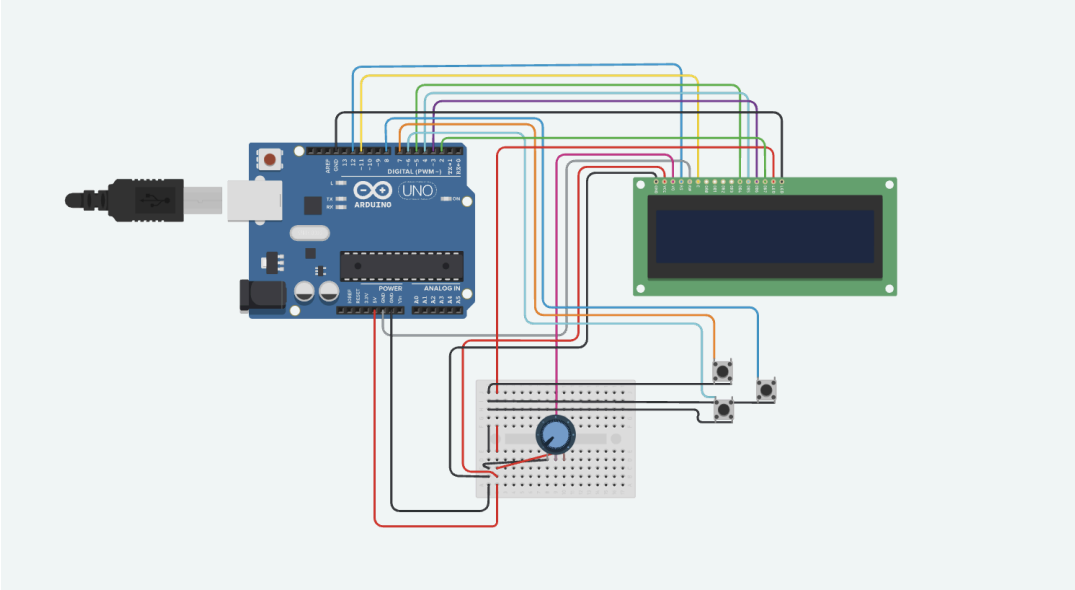


Front of Desk assistant:

Top view of Desk Assistant:



Wiring Diagram:



Code & Description:

```
#include <LiquidCrystal.h>
#include <EEPROM.h>
#include <avr/pgmspace.h>


////////////// LCD PINS /////////////////
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // RS,E,D4,D5,D6,D7

////////////// Buttons /////////////////
const byte BTN_UP = 6;
const byte BTN_DOWN = 7;
const byte BTN_SEL = 8;

struct Btn {
 byte pin;
 bool state; // true = High/not pressed
 bool lastState;
 uint32_t lastChange;
};

Btn bUp {BTN_UP, true, true, 0};
Btn bDown {BTN_DOWN, true, true, 0};
Btn bSel {BTN_SEL, true, true, 0};

const uint16_t DEBOUNCE_MS = 25;

bool readBtnEdge(Btn &b, bool wantFallingEdge = true) {
 bool raw = digitalRead(b.pin);
 uint32_t now = millis();
 if (raw != b.lastState) {
   b.lastChange = now;
```

```
  b.lastState = raw;
 }
 if ((now-b.lastChange) > DEBOUNCE_MS) {
  if (raw != b.state) {
    b.state = raw;
    if (wantFallingEdge && b.state == LOW) return true; // pressed
    if (!wantFallingEdge && b.state == HIGH) return true; // released
  }
 }
 return false;
}

//////////////////// Menu ////////////////////
enum Screen {SCR_MENU, SCR_FOCUS, SCR_BREAK, SCR_QUOTES};
Screen current = SCR_MENU;

const char item0[] PROGMEM = "Focus Timer";
const char item1[] PROGMEM = "Break (5:00)";
const char item2[] PROGMEM = "Quotes";

const char* const MENU_ITEMS[] PROGMEM = {item0, item1, item2};
const uint8_t MENU_LEN = sizeof(MENU_ITEMS) / sizeof(MENU_ITEMS[0]);
uint8_t menuIndex = 0;

//////////////////// Quotes ////////////////////
const char q0[] PROGMEM = "Small steps win.";
const char q1[] PROGMEM = "Focus beats mood.";
const char q2[] PROGMEM = "5 mins > 0 mins.";
const char q3[] PROGMEM = "One thing at a time.";
const char q4[] PROGMEM = "Breathe. Begin.";
const char* const QUOTES[] PROGMEM = { q0, q1, q2, q3, q4 };
const uint8_t QUOTE_COUNT = sizeof(QUOTES) / sizeof(QUOTES[0]);
uint8_t quoteIndex = 0;

//////////////////// Focus Timer (count up) ////////////////////
bool focusRunning = false;
uint32_t focusStart = 0;
uint32_t focusElapsed = 0;

//////////////////// Break Timer (count down) ////////////////////
bool breakRunning = false;
const uint32_t BREAK_START_MS = 5UL * 60UL * 1000UL;
uint32_t breakRemaining = BREAK_START_MS;
uint32_t breakLastTick = 0;

//////////////////// UTILS ////////////////////
void printLine16(const char* s, uint8_t row) {
 lcd.setCursor(0, row);
 for (uint8_t i = 0; i < 16; i++) {
  char c = s[i];
  if (c == '\0') {
    for (; i < 16; i++) lcd.print(' ');
    return;
  }
```

```
    lcd.print(c);
  }
}

void printLine16Str(const String &s, uint8_t row) {
  lcd.setCursor(0, row);
  for (uint8_t i = 0; i < 16; i++) {
    char c = (i < (int)s.length()) ? s[i] : ' ';
    lcd.print(c);
  }
}

String formatMMSS(uint32_t ms) {
  uint32_t t = ms / 1000UL;
  uint16_t mm = t / 60UL;
  uint8_t ss = t % 60UL;
  char buf[9];
  snprintf(buf, sizeof(buf), "%02u:%02u", mm, ss);
  return String(buf);
}

void getProgmemString(const char* const* table, uint8_t idx, char* out, size_t outSize) {
  const char* p = (const char*) pgm_read_ptr(&table[idx]);
  strncpy_P(out, p, outSize);
  out[outSize - 1] = '\0';
}

////////////////// Draw Screens //////////////////
void drawMenu() {
  char line0[17], line1[17];

  // Itendify current line with '>'
  getProgmemString(MENU_ITEMS, menuIndex, line0, sizeof(line0));
  char top[17];
  snprintf(top, sizeof(top), "> %-14.14s", line0);
  printLine16(top, 0);

  // wrap items
  uint8_t next = (menuIndex + 1) % MENU_LEN;
  getProgmemString(MENU_ITEMS, next, line1, sizeof(line1));
  char bot[17];
  snprintf(bot, sizeof(bot), " %-14.14s", line1);
  printLine16(bot, 1);
}

void drawFocus() {
  String line0 = "Focus Timer";
  String line1 = formatMMSS(focusRunning ? (focusElapsed + (millis() - focusStart)) : focusElapsed);
  printLine16Str(line0, 0);
  printLine16Str(line1, 1);
}

void drawBreak() {
  String line0 = "Break Timer";
```

```
 String line1 = formatMMSS(breakRemaining);
 printLine16Str(line0, 0);
 printLine16Str(line1, 1);
 if (breakRemaining == 0) {
   printLine16Str("Break over!", 1);
 }
}

void drawQuotes() {
 char q[33];
 getProgmemString(QUOTES, quoteIndex, q, sizeof(q));

 char top[17], bot[17];
 memset(top, ' ', 16); top[16] = '\0';
 memset(bot, ' ', 16); bot[16] = '\0';

 for (int i = 0; i < 16 && q[i]; i++) top[i] = q[i];
 for (int i = 0; i < 16 && q[i + 16]; i++) bot[i] = q[i + 16];
 printLine16(top, 0);
 printLine16(bot, 1);
}

////////////////// State Helpers ////////////////////////
void enterMenu() {
 current = SCR_MENU;
 lcd.clear();
 drawMenu();
}

void enterFocus() {
 current = SCR_FOCUS;
 lcd.clear();
 focusRunning = false;
 drawFocus();
}

void enterBreak() {
 current = SCR_BREAK;
 lcd.clear();
 breakRunning = false;
 breakRemaining = BREAK_START_MS;
 breakLastTick = millis();
 drawBreak();
}

void enterQuotes() {
 current = SCR_QUOTES;
 lcd.clear();
 drawQuotes();
}

////////////////// Setup ////////////////////
void setup() {
 pinMode(BTN_UP, INPUT_PULLUP);
```

```
 pinMode(BTN_DOWN, INPUT_PULLUP);
 pinMode(BTN_SEL, INPUT_PULLUP);

 lcd.begin(16, 2);
 printLine16("Desk Assistant", 0);
 printLine16("Loading...", 1);
 delay(600);
 enterMenu();
}

////////////////////// Loop //////////////////////////
void loop() {
 bool up = readBtnEdge(bUp);
 bool down = readBtnEdge(bDown);
 bool sel = readBtnEdge(bSel);

 switch(current) {
   case SCR_MENU: {
     if (up) {
       menuIndex = (menuIndex + MENU_LEN - 1) % MENU_LEN;
       drawMenu();
     }
     if (down) {
       menuIndex = (menuIndex + 1) % MENU_LEN;
       drawMenu();
     }
     if (sel) {
       if (menuIndex == 0) enterFocus();
       else if (menuIndex == 1) enterBreak();
       else enterQuotes();
     }
   } break;

   case SCR_FOCUS: {
     // Controls: Select = start/pause, Down = reset, Up = back
     if (sel) {
       if (!focusRunning) {
         focusRunning = true;
         focusStart = millis();
       } else {
         focusRunning = false;
         focusElapsed += (millis() - focusStart);
       }
     }
     if (down) {
       focusRunning = false;
       focusElapsed = 0;
     }
     if (up) {
       enterMenu();
       break;
     }

     //Update time
```

```
    static uint32_t lastDraw = 0;
    uint32_t now = millis();
    if (now - lastDraw >= 250) {
      drawFocus();
      lastDraw = now;
    }
  } break;

  case SCR_BREAK: {
    // Controls: Select = start/pause, Down = reset, Up = back
    if (sel) {
      breakRunning = !breakRunning;
      if (breakRunning) {
        breakLastTick = millis();
      }
    }
    if (down) {
      breakRunning = false;
      breakRemaining = BREAK_START_MS;
    }
    if (up) {
      enterMenu();
      break;
    }

    uint32_t now = millis();
    if (breakRunning && breakRemaining > 0) {
      if (now - breakLastTick >= 1000) {
        uint32_t secs = (now - breakLastTick) / 1000;
        if (secs > breakRemaining / 1000) {
          secs = breakRemaining / 1000;
        }
        breakRemaining -= secs * 1000UL;
        breakLastTick += secs * 1000UL;
      }
    }
    static uint32_t lastDraw = 0;
    if (now - lastDraw >= 250) {
      drawBreak();
      lastDraw = now;
    }

  } break;

  case SCR_QUOTES: {
    // Controls: UP/Down = prev/next quote, Select = back
    if (up) {
      quoteIndex = (quoteIndex + QUOTE_COUNT - 1) % QUOTE_COUNT;
      drawQuotes();
    }
    if (down) {
      quoteIndex = (quoteIndex + 1) % QUOTE_COUNT;
      drawQuotes();
    }
```

```
    if (sel) {
      enterMenu();
    }
  } break;
 }
}
```

Next Steps:

- [ ] Add desk wire management system beneath it
- [ ] Figure out batteries and pin scalability
- [ ] In next box above add a lighting feature
- [ ] voice goal additions
- [ ] phone connecting
- [ ] servo motion sensor
- [ ] Be able to play music through it
- [ ] Make Robotic Arm