Adventure Kit 2 (AI Apocalypse)

Guidebook

Evan Robinson

2023

[Prologue 3](#_Toc147075233)

[To Do List 4](#_Toc147075234)

[Chapter 01: Moving In 6](#_Toc147075235)

[Section 01: Do these antiques still work? 6](#_Toc147075236)

[Section 02: Bad Wiring Systems 7](#_Toc147075237)

[Code: 8](#_Toc147075238)

[Commentary: 9](#_Toc147075239)

[Section 03: Easy Light Toggles 11](#_Toc147075240)

[Code: 11](#_Toc147075241)

[Commentary: 13](#_Toc147075242)

[Interlude: Refactoring 14](#_Toc147075243)

[Classes: 14](#_Toc147075244)

[Section 04: Solar Simulation Shenanigans 20](#_Toc147075245)

[Code: 20](#_Toc147075246)

[Commentary 20](#_Toc147075247)

[Section 05: 404 Error: Alarms not found 21](#_Toc147075248)

[Code: 21](#_Toc147075249)

[Commentary 21](#_Toc147075250)

[Section 06: Dim the Lights! 22](#_Toc147075251)

[Code: 22](#_Toc147075252)

[Commentary 22](#_Toc147075253)

[Chapter 02: Base Security 101 23](#_Toc147075254)

# Prologue

How did I get here?

I joined a couple of Facebook groups, in particular [Adventure Kit 2: Day 0](https://www.facebook.com/groups/adventurekit2/). That’s how. I thought I might be able to gently mentor some people in the writing of code for Arduino systems. And in turn be mentored in the creation of circuitry to for which to write code.

So, when I started the inventr.io Adventure Kit 2 (AI Apocalypse) (hereinafter “Kit 2”), I thought I’d post my progress periodically.

When nobody objected, I continued. Right now, I’m working on Chapter 01, section[[1]](#footnote-1) 4: Solar Simulation Shenanigans[[2]](#footnote-2).

And I’m starting to collect the posts in this document, which I will (probably) eventually post to the Facebook group. I seriously doubt it’s going to be worth actually publishing it.  
  
To a large extent, the contents of this document will be the same as the posts I have put in the Facebook group

# To Do List

Chapter 01: Moving In

1. Do these antiques still work:
   * Install System
   * Run Blink successfully
2. Bad Wiring Systems (Fixing the Lights)
   * Control an LED with the HERO (on/off)
3. Easy Light Toggles (adding button inputs)
   * Set light state using momentary on/off button (down on/up off)
   * Use button to signal Hero to turn light off/on – press changes light from on to off to on
4. Solar Simulation Shenanigans (power grid issues)
   * Use photoresistor to simulate solar charging (more light, more charge)
   * Maintain “battery level”
     + Increase with solar input
     + Decrease with time light is on
     + Only charge if level is below a certain level
     + Turn light off when battery power level drops below a different level
     + *Extend to Add Green LED indicating 95%+ charge*
     + *Extend to Add Red LED indicating 25%- charge*
     + *Extend to Add always-on display to show current battery level and solar power*
5. 404 Error: Alarms not found (buzzer)
   * Use buzzer to announce battery low power level
   * Beep periodically when below the low power level
6. Dim the Lights (potentiometer & PWM)
   * Use a potentiometer to set power level
   * Set light intensity (via PWM) according to current desired light power level

Chapter 02: Base Security 101

1. Motion Sensor Security System
   * Hook up PIR Motion Sensor
   * Connect PIR Motion Sensor to White LED (simulated floodlight)
   * Button to Turn Floodlight off
   * Connect PIR Motion Sensor to Red Alert LED
   * After delay, Flash LED
   * After delay, Audible Alarm
   * Button to Turn Alert Off
2. Keypad Door Lock
   * Connect KeyPad for Input
   * Connect LCD display for Text Output
   * Add Brightness Pot to control LCD display brightness
   * Set PIN in code
   * Detect PIN input by keypad
   * Identify correct PIN and display success on LCD Display
   * Identify incorrect PIN and display failure on LCD Display
   * Add Audible tones for success and failure
   * Add Red/Green LED for success failure indicator
   * After 3 failed attempts, do not take additional input for set time
3. NFC Badges
   * 3.3V!!!
   * Add RFID reader
   * Distinguish between multiple users
     + Add multiple fobs
     + Different levels of access
     + Different responses (name?)
   * Audio access/access denied tone
   * Red/Green LED access denied/access permitted
4. RTTTL Alarm
   * Switch Alarms to use tone “music”
     + Low power
     + Critical power
     + Access permitted
     + Access denied
     + Fob vs keypad?
   * Get “super secret message” for later

Chapter 03: Greenhouse

1. Forgetting to water the garden again (Dry Plant Warning System)
   * Add Greenhouse object
     + Status Light & Audio
     + Water Level Measure
     + Status Light color AND brightness (PWM)
2. Heat Management Pt. 1 – Fan Ventilation System Simulation
   * Greenhouse ventilation
   * Temperature sensor
   * Fan speed (PWM)
   * *Extend to Display temperature and humidity*
3. Heat Management Pt. 2 – Automatic Fan System Failure (Power draw too high – Relays)
   * Add Relay
   * Add external power supply (OLED?)

Chapter 04: Daily Life Essentials

1. Accurate Alarm Clock
   * Add RTC
     + 7 segment display
     + Audible alert
     + Clock board
   * Display current time
   * Display/set alarm time
   * *Extend to add Multiple Alarms*
   * *Extend to Time Zones? UI to select*
   * *Extend to Brightness Level?*
2. Infrared Smart Lights
   * IR Receiver
   * IR Remote
   * Add light flicker indicator for receiving remote signal?
   * Display code received
   * Add RGB LED
   * Attach R,G,B on/off to remote buttons
   * *Add Remote buttons:*
     + *turn interior lights on/off*
     + *turn floodlight (exterior light) on/off*
     + *turn greenhouse fan on/off*
     + *silence audible alarms*
     + *Status Display: battery, solar, motion sensor status, most recent entry, current time, greenhouse water status, greenhouse fan status, next alarm time,*
3. Clap Lights
   * Add audible sensor
   * Detect clap
   * Interior Lights on/off with clap
   * Detect multiple claps
   * Interior/exterior lights on/off with different claps
     + 1 clap: interior Lights
     + 2 claps: exterior lights

Chapter 05: Phoenix Restoration (Resistance Group for Humanity)

1. There are other survivors. Getting Started T-Display (and discovering others exist!)
   * Hook up and test T Display
2. The other survivors share their knowledge – Time to fight back! (Advanced T-Display Networking/Communication)
   * Set up WiFi Access Point: BanzaiInstitute
   * Display IP address for TDisplay
   * Connect device to Wifi: BanzaiInstitute
   * Access IP address from above
   * Turn Light on/off
   * *Extend to Red/Green LED: on/off/color*
   * *Extend to include Password for WiFi: BBQS@uce*

Chapter 06: Base Security++ (Radar System)

1. Automatic 180 Degree Sweep Radar Upgrade
   * Re-wire other systems to accommodate Touch Screen Board
   * Wire up Touch Screen & Ultrasonic Sensor
   * Add Servo
   * Set Servo to scan 180°
   * Display detected objects
   * *Color code changes*
   * *Add UI to override servo position*
     + *Single point*
     + *Scan endpoints*
   * *Add Alarm indicator for detected item of concern*
2. False Signals – RGB Turret w/LCD Touch Screen
   * 9V battery for power
   * Joystick wired for X only
   * Detect joystick button
   * Stepper Motor with RGB LED “gun”
   * Random color
   * *Test separate button for “Fire”*
3. False Signals 2 – RGB Turret w/T-Display
   * Duplicate 20 above using T-Display
     + 9V battery for power
     + Joystick wired for X only
     + Detect joystick button
     + Stepper Motor with RGB LED “gun”
     + Random color
     + *Test separate button for “Fire”*

Chapter 07: Showdown Against The AI

1. Official Victory Signal Flare (Finale!)
   * SSID
   * SSID\_PASSWORD
   * USER\_EMAIL
   * API\_KEY
   * Make the animation happen
   * Check on the map

# Chapter 01: Moving In

## Section 01: Do these antiques still work?

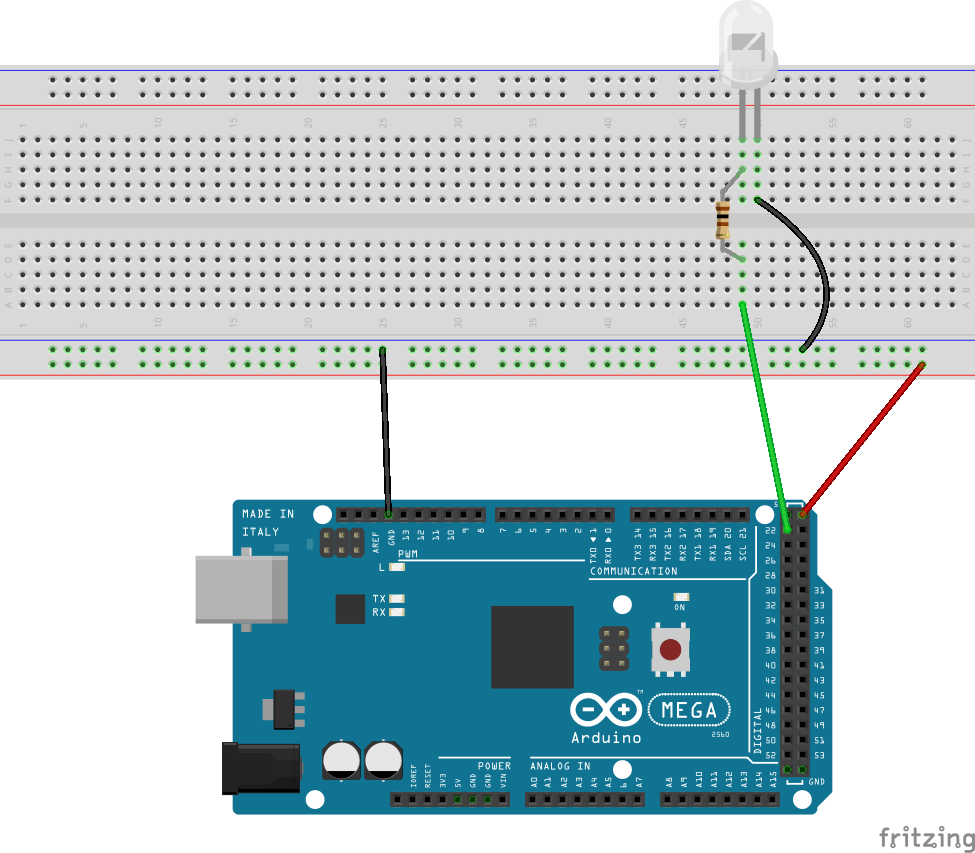
I’m leaving this one for the inventr.io people to manage. It is how you get all the software downloaded and connect up your HERO board and make sure you know how to compile code, upload it to the board, and make the built in LED light blink to your command.

## Section 02: Bad Wiring Systems

ADDED: I forgot to say that I'm posting these because I hope they will be helpful to people with less experience than I have coding. It's entirely possible I'm wrong, in which case somebody tell me and I'll stop.

I'm also posting these because I'm almost a complete noob at the electronics, and I'm trying to work them from scratch here (still using searches to find things out, but I'm not just copying any circuit that is provided by inventr). So I imagine that at some point somebody will say (roughly) "WTF are you doing? Do it this way!" -- and I'm very happy with that idea.

Feel free to disagree with me. To quote Bull Durham, "I believe what I believe", and I'm not going to insist that y'all agree.

My spec: Control an LED with the HERO (on and off).

Circuit 1: Section 2: Bad Wiring Systems

|  |
| --- |
| *White LED* |
| *Resistor 100 Ω* |

### Code:

#include <Arduino.h>

const uint8\_t whiteLEDControlPin = 22;

const int ledOnIndefintely = 0;

// put function forward declarations here:

void turnLEDOnFor(int millis = 1000, uint8\_t pin = whiteLEDControlPin);

void turnLEDOff(uint8\_t pin = whiteLEDControlPin);

void setup() {

pinMode(whiteLEDControlPin, OUTPUT);

}

#define THROW\_AWAY\_CODE[[3]](#footnote-3)

void loop() {

#ifdef THROW\_AWAY\_CODE

digitalWrite(whiteLEDControlPin, HIGH);

delay(1000);

digitalWrite(whiteLEDControlPin, LOW);

delay(1000);

#else

turnLEDOnFor(300);

delay(300);

#endif

}

// put function definitions here:

void turnLEDOnFor(int millis, uint8\_t pin) {

digitalWrite(pin, HIGH);

if (millis != ledOnIndefintely) {

delay(millis);

turnLEDOff();

}

}

void turnLEDOff(uint8\_t pin) {

digitalWrite(pin, LOW);

}

Code Block 1: Section 2: Bad Wiring Systems

### Commentary:

Use of ‘const’ instead of #define

I prefer using ‘const int constantName = XX’ to ‘#define CONSTANT\_NAME XX primarily because I’ve been programming for long enough that I’ve seen other people make a lot of mistakes, I’ve made a lot of mistakes, and I’ve used enough tool sets that had bugs in them that I dislike #define.

Why?

Because #define is a textual substitution, not a syntactic or semantic substitution. Those are big words that means #define is like doing a search/replace on your code. If you’ve ever tried to, say, replace each use of ‘end’ in a file with ‘END’, you will have discovered that you have oddities like ‘sEND’ and ‘trEND’ which you did not intEND[[4]](#footnote-4).

Furthermore, #define can be used to create more complex substitutions, such as ‘#define MAX(x, y) x > y x : y’ which replaces ‘MAX(x,y) with a complex expression (the c++ ‘ternary operator’) which almost certainly needs to be fully parenthesized[[5]](#footnote-5) to deliver what the author intended.

Because such substitutions happen in the scanning and tokenizing section of the compiler (I know, way more than you want to hear), the code as compiled is not easily available to the programmer, which means the bug is **hard to find**.

Not Good.

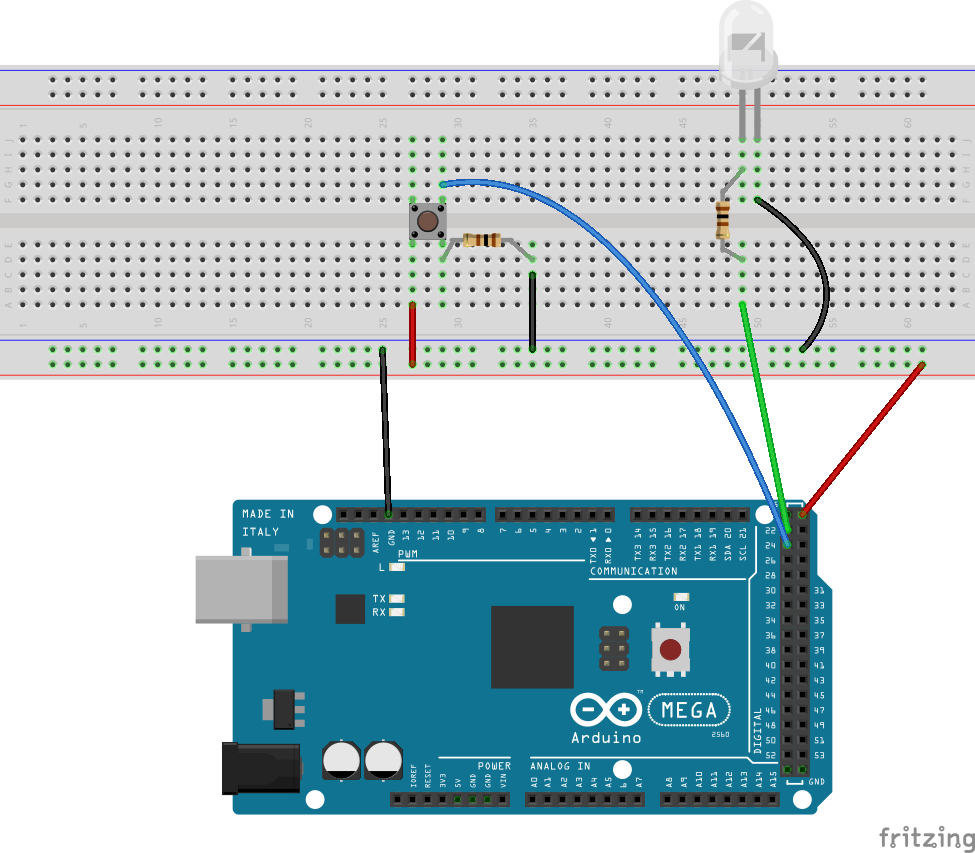
Use of default parameters

The ‘int millis = 1000, uint8\_t pin = whiteLEDControlPin’ portion of the forward declarations uses something called ‘default parameters’. Essentially, if I call the function turnLEDOnFor() without any parameters, it uses these. If I call it with a single parameter, it is as though I gave it only the first parameter (millis). Thus by passing whiteLEDControlPin as a default parameter, I preserve my option to control any LED with turnLEDOnFor() and turnLEDOff(), but I don’t need to provide the pin parameter every time I call them.

## Section 03: Easy Light Toggles

My spec:

1. toggle light state using momentary on/off button (directly);
2. use button to signal Hero to toggle light from off to on and back



Circuit 2: Section 3: Easy Light Toggles

|  |
| --- |
| *White LED* |
| *Resistor 100* *Ω (on LED)* |
| Resistor 10K Ω (on Switch*)* |
| Momentary On Switch |

### Code:

#include <Arduino.h>

const uint8\_t whiteLEDControlPin = 22;

const uint8\_t buttonInputPin = 24;

const int ledOnIndefintely = 0;

const long debounceInterval = 50L;

const int buttonPressed = 1;

// put function forward declarations here:

void turnLEDOnFor(int millis = 1000, uint8\_t pin = whiteLEDControlPin);

void turnLEDOff(uint8\_t pin = whiteLEDControlPin);

void toggleInteriorLights(void);

void setup() {

pinMode(whiteLEDControlPin, OUTPUT);

turnLEDOff();

pinMode(buttonInputPin, INPUT);

Serial.begin(9600);

Serial.println("setup complete");

}

void loop() {

static int previousPinValue = 0;

static long previousDebounceTime = millis();

int pinValue = digitalRead(buttonInputPin);

if ((millis() - previousDebounceTime) < debounceInterval) {

// debouncing: if the transition was less than debounceInterval ms ago, ignore it

return;

}

if (previousPinValue != pinValue) {

previousPinValue = pinValue;

if (pinValue == buttonPressed) {

toggleInteriorLights();

}

}

}

// put function definitions here:

void toggleInteriorLights(void) {

static bool lightIsOn = false;

if (lightIsOn) {

turnLEDOff();

lightIsOn = false;

}

else {

turnLEDOnFor(ledOnIndefintely);

lightIsOn = true;

}

}

void turnLEDOnFor(int millis, uint8\_t pin) {

digitalWrite(pin, HIGH);

if (millis != ledOnIndefintely) {

delay(millis);

turnLEDOff();

}

}

void turnLEDOff(uint8\_t pin) {

digitalWrite(pin, LOW);

}

Code Block 2: Section 3: Easy Light Toggles

### Commentary:

1. ‘toggleInteriorLights()’

This is the first indication that the code is going to become more Literate. Everything else in the code here and that we’ve seen so far is very hardware specific. Very much in the Arduino/electronics domain, if you will. The code that runs the lighting refers to LED and lots of variable names reference ‘pin’ of one sort or another.

What I mean by “in the Arduino … domain” is that the code is not talking about the fantasy inventr.io is providing us. Instead, its talking about the microcontroller and the things on the breadboard.

The next Section, Solar Simulation Shenanigans, will change this in a big way by refactoring the code into the problem domain.

1. Debouncing

“Debouncing” a key or button refers to managing the moments where it is so close to closing or opening that the value presented by the electronics is indeterminate or changing back and forth so quickly that it may be 0 one millisecond, 1 the next, and 0 again the one after that.

Humans do not operate at that speed.

So we introduce code to make sure that the state of the switch is stable for some period of time that humans can operate at. In this code, the simplest possible debouncing: ignoring any state change in the switch return value for at least 50ms. It’s not perfect, but for the learning experience it is good enough.

## Interlude: Refactoring

[Refactoring](https://refactoring.com/) is restructuring your code to make it better.

What does “better” mean?

For my purposes, it means “easier to understand”. Because that means “easier to change”, “easier to reuse”, “easier to fix”, and “easier to build upon”.

You will find a variety of definitions online for the word “[refactoring](https://en.wikipedia.org/wiki/Code_refactoring)”. Common among them is the idea that this change does not change the underlying function, and that the changes are made systematically – sometimes to the extent of following a bunch of rules.

I use the word somewhat loosely.

I was introduced to the word “refactoring” in 1999, when I worked at Adobe Systems in the core technology group. [Jon Reid](https://qualitycoding.org/about/), one of my team, got very interested in the [book](https://www.amazon.com/dp/B07LCM8RG2/) when it came out. As I was no longer primarily a programmer, I got the gist but haven’t ever done the formal process.

In this case, there are going to be two major things happening to the code: I’m going to introduce some classes (real Object Oriented Programming stuff!); and I’m going to rename and re-organize the code from the hardware/microcontroller oriented names and organization into what I call the “problem domain*[[6]](#footnote-6)*” – instead of building circuits and writing code to make electronic components do certain things, I’m going to start creating a “post-apocalyptic dwelling”.

So no more referring to the lights as “LEDLights” or anything like that. These lights are the interior lights of our house. The button isn’t constantly referred to using the buttonInputPin, now it’s called the interiorLightsButton.

### Classes:

led.h / led.cpp:

The LED class encapsulates the basic functioning of an LED on the breadboard, with a pulldown resistor (as opposed to a pullup resistor – made evident by the use of HIGH as on and LOW as off). It provides a constructor (used to create the object, bind it to the relevant pin, and set the necessary pinMode), methods (functions) to turn the LED on and off, and a method to determine whether the LED is currently on.

Classes in c++ (opinions differ as to whether the Arduino language is a full implementation of c++ or not. I have no opinion) are generally declared (described from an external point of view) in a .h file and defined (described sufficiently to create executable code) in a .cpp file.

Classes are the basic unit of inheritance (the re-use of code by creating subclasses or derivative classes which extend their superclass or ancestor class). Effectively that means that I can create a subclass of LED which would have the same public interface but run different code where necessary (if creating a class to control an RGB LED, or a BiColor LED). The public interface can be extended to include additional necessary functionality (as to set the color of an RGB LED).

/\*

led.h

Evan Robinson, 2023-09-30

Class to manage LED on arduino

Presumes HIGH is on, LOW is off

\*/

#ifndef led\_h

#define led\_h

#include <Arduino.h>

class LED {

public:

LED(uint8\_t pin);

void turnOn();

void turnOff();

bool isOn();

private:

uint8\_t \_pin;

bool \_isOn;

};

#endif

Code Block 3: Interlude: led.h (the declaration of the class LED)

/\*

led.cpp

Evan Robinson, 2023-09-30

Class to manage LED on arduino

Presumes HIGH is on, LOW is off

\*/

#include "led.h"

#include <Arduino.h>

LED::LED(uint8\_t pin) {

\_pin = pin;

pinMode(\_pin, OUTPUT);

turnOff();

}

void LED::turnOn() {

digitalWrite(\_pin, HIGH);

\_isOn = true;

}

void LED::turnOff() {

digitalWrite(\_pin, LOW);

\_isOn = false;

}

bool LED::isOn() {

return \_isOn;

}

Code Block 4: Interlude: led.cpp (the definition of the class LED)

button.h / button.cpp:

The Button class encapsulates the basic function of a momentary on push button (or switch), again with a pulldown resistor (so HIGH is pressed, LOW is not pressed).

/\*

button.h

Evan Robinson, 2023-09-30

Class to manage momentary button on arduino

Presumes HIGH is pressed, LOW is not pressed

Press occurs on down press of button

\*/

#ifndef button\_h

#define button\_h

#include <Arduino.h>

class Button {

public:

Button(uint8\_t pin);

bool isPressed();

bool hasChanged();

int value();

protected:

private:

uint8\_t \_pin;

int \_value;

int \_valueHasChanged;

};

#endif

Code Block 5: Interlude: button.h (the declaration of the class Button)

Note that there is only one method where the value of the push button is read from the hardware. This guarantees that any necessary housekeeping will be done no matter how the user programmer accesses the value.

/\*

Button.cpp

Evan Robinson, 2023-09-30

Class to manage momentary button on arduino

Presumes HIGH is pressed, LOW is not pressed

Press occurs on down press of button

\*/

#include "button.h"

#include <Arduino.h>

Button::Button(uint8\_t pin) {

\_pin = pin;

pinMode(\_pin, INPUT);

\_value = value();

\_valueHasChanged = false;

}

bool Button::isPressed() {

\_value = value();

return (\_value == HIGH);

}

int Button::value() {

// TBD: Debounce

int currentValue = digitalRead(\_pin);

\_valueHasChanged = (\_value != currentValue);

// if (\_valueHasChanged) {

// Serial.println(currentValue);

// }

return digitalRead(\_pin);

}

bool Button::hasChanged() {

return \_valueHasChanged;

}

Code Block 6: Interlude: button.cpp

/\*

Simulated post-apocalyptic dwelling

Supports simulated interior lighting (interiorLights)

controlled by a momentary switch (interiorLightsButton)

which toggles lights on and off

\*/

#include <Arduino.h>

#include "button.h"

#include "led.h"

// Hardware values

const uint8\_t whiteLEDControlPin = 22;

const uint8\_t buttonInputPin = 24;

// Dwelling Contents

Button interiorLightsButton = Button(buttonInputPin);

LED interiorLights = LED(whiteLEDControlPin);

// Forward Declarations

void interiorLighting(void);

// Arduino Setup

void setup() {

}

// Arduino Loop

void loop() {

// Loop every 1/10th seconds

delay(100);

interiorLighting();

}

// Local Functions

void interiorLighting() {

// Turn interiorLights on and off

if (interiorLightsButton.isPressed()) {

if (interiorLightsButton.hasChanged()) {

if (interiorLights.isOn()) {

Serial.println("Lights Off");

interiorLights.turnOff();

}

else {

Serial.println("Lights On");

interiorLights.turnOn();

}

}

}

}

Code Block 7: Interlude: main.cpp

Although the file is still called main.cpp, it bears little resemblance to its predecessor. The pin definitions are the same, but no setup happens in setup() anymore because it’s all handled by the constructors under the // Dwelling Contents comment.

The interior lights are now fully managed by a single function call in loop(): interiorLighting();

interiorLighting() asks interiorLightsButton if it ‘isPressed’. If it is, and it ‘hasChanged’, then it’s time for us to toggle the interior lights from on to off or off to on.

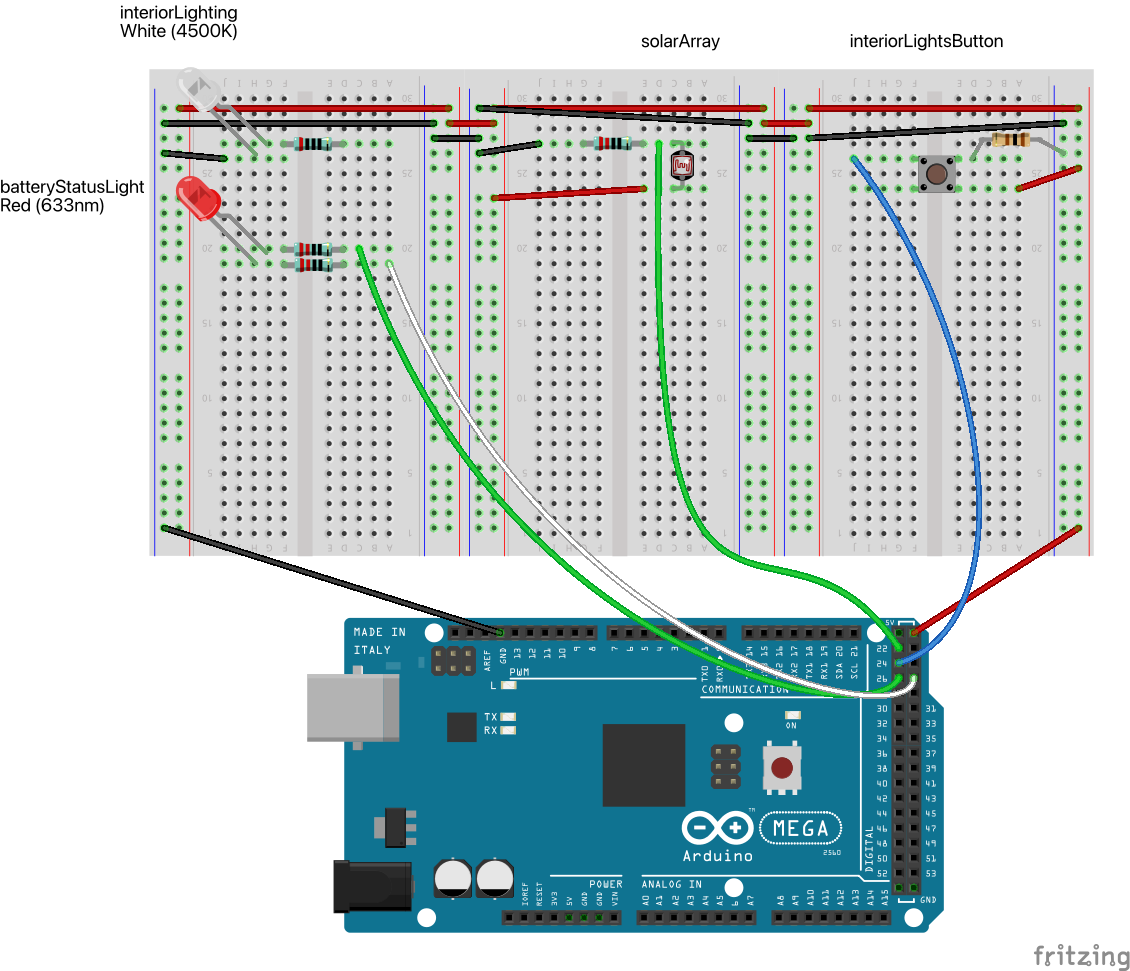
We are now in the problem domain, doing things to our simulated dwelling instead of our real physical circuitry.

Now it’s time to add the simulated battery and solar charging systems.

## Section 04: Solar Simulation Shenanigans

The Spec:

* + Use photoresistor to simulate solar charging (more light, more charge)
  + Maintain “battery level”
    - Increase with solar input
    - Decrease with time light is on
    - Only charge if level is below a certain level
    - Turn light off when battery power level drops below a different level
    - Add Green LED indicating 90%+ charge
    - Blink LED from 80%-90%
    - Add Red LED indicating 25%- charge
    - Blink LED from 10%-25%



Circuit 3: Section 4: Solar Simulation Shenanigans

### Code:

/\*

Simulated post-apocalyptic dwelling

Supports simulated interior lighting (interiorLights)

controlled by a momentary switch which toggles lights

on and off

Simulated electrical storage (electricalStorage) and

solar charger (power level supplied by photoresistor)

Lighting now simulates a power drop when on

\*/

#include <Arduino.h>

#include <math.h>

#include "button.h"

#include "led.h"

#include "power.h"

#include "photoresistor.h"

// Hardware values

const uint8\_t whiteLEDControlPin = 22;

const uint8\_t buttonInputPin = 24;

const uint8\_t photoResistorInputPin = 0;

// Timing constants

const unsigned long oneTenthOfASecond = 100L; // one 'tick'

const int ticksPerLighting = 1; // lighting input happens every tick

const int ticksPerCharging = 10; // charging happens every second

// Dwelling Contents

Button interiorLightsButton = Button(buttonInputPin);

LED interiorLights = LED(whiteLEDControlPin);

Power electricalStorage = Power(photoResistorInputPin);

const double interiorLightsPowerUsage = 3.0;

// Forward Declarations

void interiorLighting(void);

void batteryChargingAndUsage(void);

// Arduino Setup

void setup() {

Serial.begin(9600);

Serial.println("setup complete");

}

// Arduino Loop

// Instead of using delay(), millis() is used to enforce a timing 'tick' of

// 1/10 of a second (oneTenthOfASecond).

void loop() {

static int tickCount = 0;

static unsigned long previousMillis = 0L;

unsigned long currentMillis = millis();

if ((currentMillis - previousMillis) < oneTenthOfASecond) {

return;

}

previousMillis = currentMillis;

tickCount++;

electricalStorage.tick();

if ((tickCount % ticksPerLighting) == 0) {

interiorLighting();

}

if ((tickCount % ticksPerCharging) == 0) {

batteryChargingAndUsage();

}

}

// Local Functions

void interiorLighting() {

// Turn interiorLights on and off using button

if (interiorLightsButton.isPressed()) {

if (interiorLightsButton.hasChanged()) {

if (interiorLights.isOn()) {

Serial.println("Lights Off");

interiorLights.turnOff();

}

else if (!electricalStorage.isCritical()) {

if (electricalStorage.isLow()) {

Serial.println("Lights On But DIMMED");

// TBD: turn them on dim if batteryLevel

// is between critical and warning levels

interiorLights.turnOn();

}

else {

Serial.println("Lights On");

interiorLights.turnOn();

}

}

}

}

// manage lights depending upon current power levels

if (interiorLights.isOn()) {

if (electricalStorage.isCritical()) {

interiorLights.turnOff();

Serial.println("Power Critical, Lights Off");

}

else if (electricalStorage.isLow()) {

// TBD: dim interiorLights

Serial.println("Lights are on, but dimmed");

}

}

}

void batteryChargingAndUsage() {

static double previousBatteryLevel = electricalStorage.batteryLevel();

electricalStorage.chargeBattery();

// account for interiorLights power usage

if (interiorLights.isOn()) {

electricalStorage.usePower(interiorLightsPowerUsage);

}

// send battery power level to serial

// TBD: Remove when this info is sent to hardware display instead of Serial

if (previousBatteryLevel != electricalStorage.batteryLevel()) {

electricalStorage.showStatus();

}

previousBatteryLevel = electricalStorage.batteryLevel();

}

### Commentary:

I’ve given up providing a parts list (comments may change that) in favor of trying to make sure that the resistors are properly shown in fritzing and using labels to indicate which piece of hardware goes with which piece of code.

1. In this image, I used a RED LED to indicate a BiColor Red/Green LED (with two leads) because fritzing does not have a BiColor Red/Green LED in the standard parts bin. Such LEDs are cheaply available (without a data sheet) from Amazon.[[7]](#footnote-7)
2. I’ve gone a little far afield of the instructions here. There are a LOT of TBDs (To Be Dones) in comments (you can search for them if you have a helpful IDE), which mostly have to do with adding hardware I haven’t included here: a buzzer for audible alerts; a supplementary display to give you continuous information on your electrical use and solar collection; dimming the simulated interior lights by using PWM (Pulse Width Modulation) on the white LED. Not only did I add a completely new electrical system status light, I bought extra hardware to make it use only one LED – but it wouldn’t be hard to wire it up using two: a red and a green – it wouldn’t even use any more resistors.
3. In line with the Interlude of Refactoring, I now have objects for hardware like LEDs (both single and bicolor), the button[[8]](#footnote-8), and the photoresistor, but *also* for objects within the simulation like the Power subsystem, the photoresistor which appears in the code as the \_solarArray within the Power subsystem, and the LED which is now the interiorLights object.[[9]](#footnote-9) More and more things are appearing in the problem domain, which is, IMO, good for understanding the system.
4. I have switched my development IDE from Arduino to PlatformIO (using Visual Studio Code). I’m pretty happy with it. Immediately after writing this section (or perhaps during it), I will be moving the code from separate directories on my desktop for each section into git and probably into GitHub, since I move around a certain amount and it will be inconvenient to try and keep multiple systems synced without using tools to do so.

## Section 05: 404 Error: Alarms not found

### Code:

### Commentary

## Section 06: Dim the Lights!

### Code:

### Commentary

# Chapter 02: Base Security 101

1. Unlike Adventure Kit 1: 30 Days Lost in Space, each project in Kit 2 isn’t conveniently identifiable by a Globally Unique Identifier (GUID), so I have referred to each individual post in the Course Content as a “section”. I hope that’s clear enough. [↑](#footnote-ref-1)
2. Yes, in the Course Content it’s spelled Shinanigans. They’re wrong. [↑](#footnote-ref-2)
3. An explanation of the #ifdef THROW\_AWAY\_CODE ... #else ... [#endif](https://www.facebook.com/hashtag/endif?__eep__=6&__cft__%5B0%5D=AZUcF_stNFqjnbb-3jxzxtOQTCUhHgCAOXloOz-pSDxSzykiiaJdSIdeMHkJ7GBH_xiD0QLa6NYX3R2J1L0wxCBze0gYdNSDu-WSTHrtp029GY8_lvsI6xjoIfKXsRWmXvnSf8klo4IuOlnGgpYBCMHo&__tn__=R%5D-R):

   The four lines of code between [#ifdef](https://www.facebook.com/hashtag/ifdef?__eep__=6&__cft__%5B0%5D=AZUcF_stNFqjnbb-3jxzxtOQTCUhHgCAOXloOz-pSDxSzykiiaJdSIdeMHkJ7GBH_xiD0QLa6NYX3R2J1L0wxCBze0gYdNSDu-WSTHrtp029GY8_lvsI6xjoIfKXsRWmXvnSf8klo4IuOlnGgpYBCMHo&__tn__=R%5D-R) THROW\_AWAY\_CODE and [#else](https://www.facebook.com/hashtag/else?__eep__=6&__cft__%5B0%5D=AZUcF_stNFqjnbb-3jxzxtOQTCUhHgCAOXloOz-pSDxSzykiiaJdSIdeMHkJ7GBH_xiD0QLa6NYX3R2J1L0wxCBze0gYdNSDu-WSTHrtp029GY8_lvsI6xjoIfKXsRWmXvnSf8klo4IuOlnGgpYBCMHo&__tn__=R%5D-R) are only compiled if the macro THROW\_AWAY\_CODE has been defined earlier in the source file.

   As the code is, THROW\_AWAY\_CODE has been defined, and those four lines will be executed. So far as I'm concerned, that is the simplest solution to this particular spec.

   HOWEVER, as I plan to use this code as the basis for future code, I have already refactored the code into what I consider a more obvious, understandable, usable, and extendable form.

   I may be wrong about any one or more of those adjectives: I may find in the future that this code (the code outside the THROW\_AWAY\_CODE macro region) is less obvious, less understandable, less usable, and/or less extendable. But it's my current judgement that I'm doing future me a favor here. [↑](#footnote-ref-3)
4. This is a common enough error that Word repeatedly replaced ‘intEND’ with ‘intend’ while I was writing this. [↑](#footnote-ref-4)
5. #define MAX(x, y) (((x) > (y)) (x) : (y)): which is hardly the most readable thing in the world – although I’ve seen worse.

   And if either ‘x’ or ‘y’ are not simple identifiers or numeric constants but instead are statements, it gets worse fast. Consider, just for a moment, what happens in this MAX macro if x is ‘counter++’, which may be textually substituted in twice, so you might get two increments instead of one. But that’s not the real problem. The real problem is that you didn’t intend for the values ‘passed’ to MAX(x, y) to be effected at all, much less twice. [↑](#footnote-ref-5)
6. The first time I remember using this specific term, I was a Technical Director at Electronic Arts providing advice to the team that was designing and building Madden Football for the 3DO console. We were discussing how the developers should handle kickoffs, and the designer was talking about force applied to the football and vectors and directions.  
   I stopped him and said (approximately) “you want the code to mirror the problem domain. How does a kicker think about kicking the football? They don’t say to themselves ‘I’m gonna hit this ball with 72 Newtons of force in a horizontal vector of 22° to the right of center at an upward angle of 42.5°’. They say to themselves ‘kick away to the right side’, ‘onside straight ahead looking for a pooch up’, or ‘I want the ball to bounce straight up near the right sideline and goal line”.  
   These last three descriptions exist in the world of football, not the world of physics. Yes, they will eventually get translated to the world of physics, but that’s not how they start. [↑](#footnote-ref-6)
7. I bought [these](https://www.amazon.com/dp/B07WTCB6H6/) for about 8¢ (US) apiece. I sacrificed two to figure out how they worked and what size resistor would work (220Ω, IIRC – I tried to make the fritzing image correct). The trick is that you apply HIGH to one pin and LOW to the other to get one color, then swap the current flow to get the alternate color. In this context RED means bad, GREEN means good, and blinking means not quite so bad/good as solid. [↑](#footnote-ref-7)
8. Oh, there are stories about the button hardware. I spent about 20 minutes on Amazon, Adafruit, and several other sites and have ordered perhaps 20 different kinds of button hardware in hopes of finding several which are easier to wire and control. I will spare you the details, bu the best way I’ve found to use the included tactile switch button is to use pliers to straighten each leg and then push it *really, really hard* into the breadboard – and then it *might* stop providing floating signals which turn the lights on and off like gremlins (I had to try this with several of them before I got one that was reasonably stable). [↑](#footnote-ref-8)
9. Having the \_solarArray (photoresistor) as a *subobject* of the Power object requires a special technique for constructing the Power object: using a [*constructor initializer list*](https://www.fi.muni.cz/usr/jkucera/tic/tic0145.html) because the \_solarArray object can’t use a default constructor – it requires a pin number input from the Power object constructor. [↑](#footnote-ref-9)