**Can you complete these scenarios?**

**Scenario 1:**

**You want to use a dial or something similar to control lights.**

**Scenario 2:**

**You want an easy way to tell the temperature from across the room, or in different rooms of the thermostat.**

**Scenario 3:**

**You want a remote way to control the lights in your room. So you don’t have to get up to turn them on/off**

**Scenario 4:**

**You want to sync up lights to sound.**

**Scenario 5:**

**You want your robot to wave goodbye when people leave**

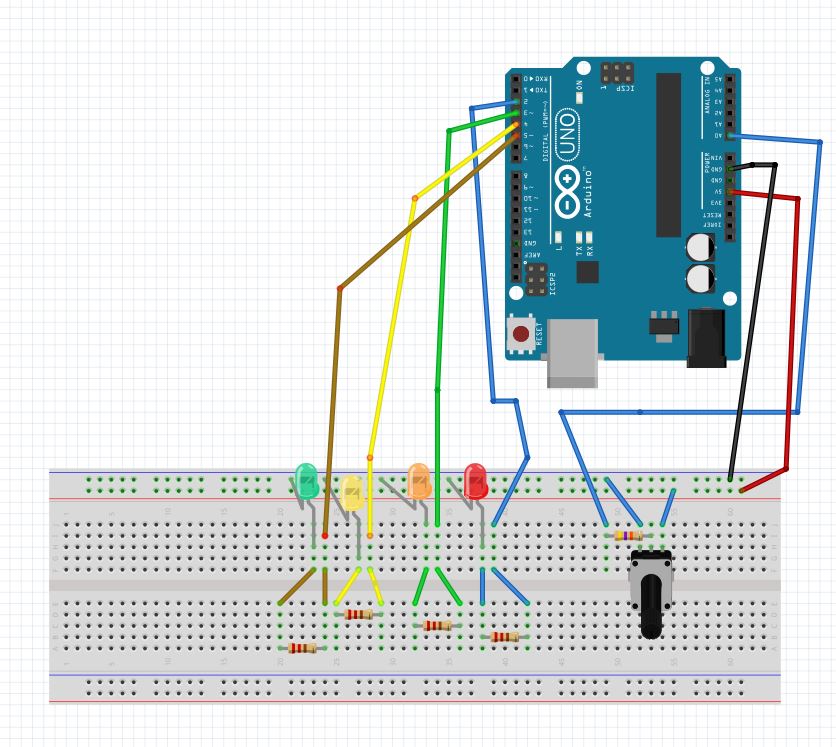
**Scenario 6: You want to be alerted when someone enters your room**

**Come up with your own scenarios!!!**

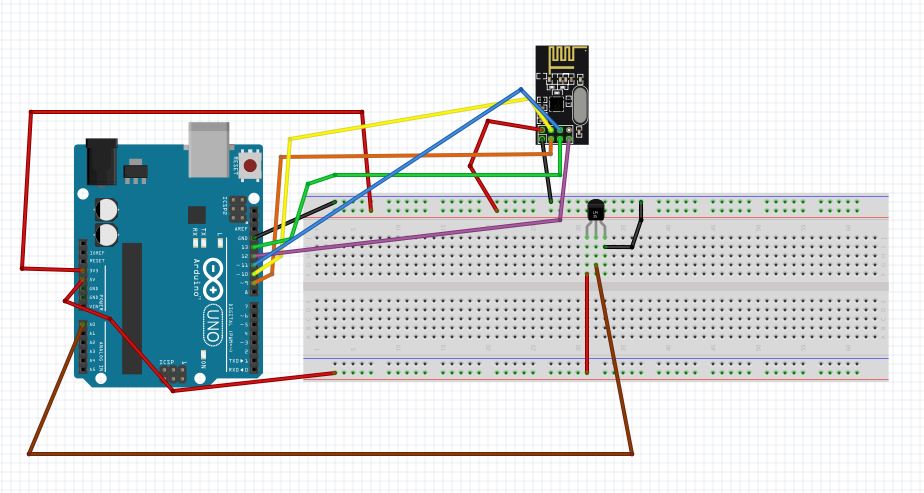
**And hacks!**

**Wiring diagrams**

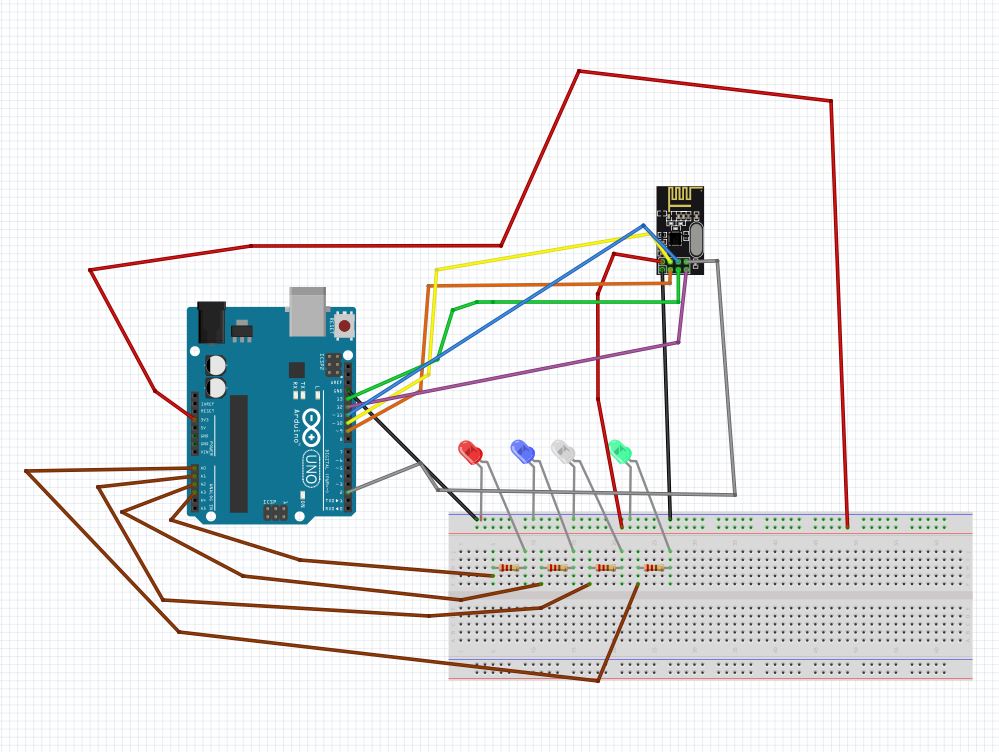
**Diagram for Scenario 1**

****

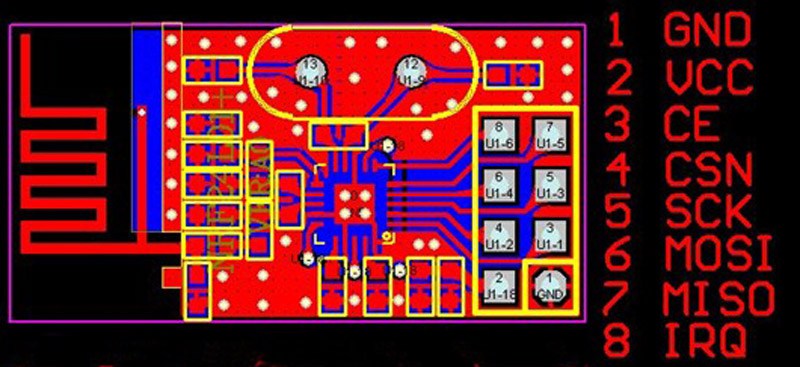
**Diagram for scenario2a (transmitter)**

****

**Diagram for scenario2b(receiver)**

****

**Pin map for the RF transceiver**

****

**For all the pins used in the transceiver the nrf24 library will handle control of the pins and what they should be doing. DO NOT write code that controls the pins that the nrfTransciever is connected too. (Unless it is through the nrf24 library)**

**1 GND**- ground

**2 VCC**- positive connection for power. This transceiver needs to be connected to 3.3v

**3 CE**- chip enable. The signal sent to this pin tells the transceiver that it should be looking for a radio frequency message or sending a radio frequency message.

**4 CSN**- chip select not. This pin tells the chip if it is communicating with the Arduino’s serial peripheral Interface or not.

**5 SCK**- connects the chip to the Arduino’s clock so they can communicate with each other.

**6 MOSI**- Master Out Slave In. The master is the Arduino, slave is the chip. This is where the Arduino sends data to the chip.

**7 MISO**- Master In Slave Out. This is where the chip sends data to the Arduino.

**8 IRQ**- Interrupt. This pins sends a signal to the Arduino that something interesting has happened. We will only be using this pin on the receiver end. More on how interrupts work below.

**Interrupts**- typically Arduinos only check on the status of a pin or change something to when you ask them to in the code. This means they could easily miss a brief change in sensor data (a quick button push, or movement detected by an IR sensor) if the code is telling them to do something else like a delay(). This is solved by interrupts. To better explain interrupts I will go through the syntax of the attachInterrupt function: attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);

digitalPinToInterrupt(pin) – a function that tells the Interrupt what pin will trigger it. You can choose either the Arduino’s pin# 2 or 3 as the pin parameter for this function. This function cannot write to a pin or the Serial Monitor.

ISR- this is a function that is defined in the Arduino sketch you use an interrupt in. This function can NOT take any parameters in and returns nothing. Inside the function you will change a variable, probably based on the reading of a pin or nRF data.

mode- this defines when the interrupt is triggered. This can be LOW, CHANGE, RISING, or FALLING.

It’s perfectly fine if this is confusing, it will get more clear with more practice and research. The Skelton codes I have written should allow you to avoid having issue with interrupts.