**CIS 316 Quarterly Accumulative Project**

**Process**

Originally when I submitted my proposal for this project at the beginning of the quarter, I wanted to do an outdoor motion sensor that would be mounted to one of the trees lining the trails out of our yard. This was supposed to have an Arduino Wi-Fi hat on it along with the PIR sensor, and the idea was that it would communicate to another Arduino with Wi-Fi hat inside our door which would in turn alert us when motion was detected via an alarm and printout to the LCD. As I started looking into this, I realized that there were going to be issues with mounting and waterproofing that I couldn’t address without spending additional money. My original plans would have also required me to set up some kind of server on at least one of the two unos and develop a way for this server to receive and use information, considering this I also felt that my original plans may have been a bit too ambitious for a quarterly project. I ultimately decided to do something similar, but something that did not require the additional money on my part and was a little simpler to code. The project that I decided on was a motion sensor that lets us know that our fridge door is open by printing “Door Open” to an external LCD display and playing the imperial march using a piezo buzzer. This system uses the same PIR sensor, but all the real time computing is done on one board and no networking is required.

**Challenges**

True to my nature, I started working on this project just a few nights ago 😊. I had previously wired up the LCD and PIR sensor for different labs that we have done throughout the quarter, so that part was not too hard, but I needed to do some additional research on the piezo buzzer and figure out a way to continually run a loop that would play the imperial march when motion was detected and play nothing when there was no motion + print out that the door had been closed.

I struggled for quite a while getting the setup to do anything with consistency. It would play notes in a very distorted way and would never register when there was no movement. After spending a couple of hours trying to figure this out by plugging in variables in different places and rewriting if/else statements multiple times, I learned that the PIR sensor always goes through a setup phase when it is first detected and any results within the first 60 seconds should be disregarded. Armed with this knowledge I made a few more changes to my code and finally got the sensor to properly detect motion when it was there and flip its bits to LOW when no infrared motion was detected.

At this point the notes that were playing from my piezo buzzer were still very much distorted, so I borrowed this nifty loop from a gist here(<https://gist.github.com/tagliati/1804108>), which allowed me to make sure the notes were playing for long enough and that they were properly spaced out. I also took a note from this gist and simply defined each note along with its corresponding frequency. Using this method was much more efficient than trying to use Arduino’s built-in tone() method and then dealing with the mess of sorting out frequencies, tone-length, and delays.

After adding this method, I also went ahead and installed the corresponding LED that this project used as a sort of a visualizer for the tune as it plays. My final code and a couple pictures of the project are below. You will find more information and a demo in the accompanying video:

#include <LiquidCrystal.h>

#define c 261

#define d 294

#define e 329

#define f 349

#define g 391

#define gS 415

#define a 440

#define aS 466

#define b 494

#define cH 523

#define cSH 554

#define dH 587

#define dSH 622

#define eH 659

#define fH 698

#define fSH 740

#define gH 784

#define gSH 830

#define aH 880

// Defining each note along with its corresponding frequency in hz

int pirPin = 9;

// pin for pir sensor

int buzzPin = 10;

// pin for piezo buzzer

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int state = LOW;

int val = 0;

void setup() {

  lcd.begin(16, 2);

  pinMode(pirPin, INPUT);

  pinMode(LED\_BUILTIN, OUTPUT);

  pinMode(buzzPin, OUTPUT);

  tone(buzzPin, 1000, 500);

}

void loop() {

  val = digitalRead(pirPin);

  // read current value of pirPin

  //

  if (val == HIGH) {

    if (state == LOW) {

      state = HIGH;

      lcd.setCursor(0, 0);

      lcd.print("Door Open");

      imperialMarch();

      noTone(buzzPin);

    }

  } else {

    val = LOW;

    if (state == HIGH) {

      state = LOW;

      lcd.clear();

      lcd.setCursor(0, 0);

      lcd.print("Door Closed");

      noTone(buzzPin);

      state = LOW;

    }

  }

}

void note (unsigned char buzzPin, int frequencyInHertz, long timeInMilliseconds)

{

    digitalWrite(LED\_BUILTIN, HIGH);

    // set LED to high at beginning of note loop to show note is playing

    int i;

    // counter data member

    long delayAmount = (long)(1000000/frequencyInHertz);

    // set amount of time to play note by delaying between high and low

    long loopTime = (long)((timeInMilliseconds\*1000)/(delayAmount\*2));

    // set length of loop for each note

    // for loop that sets the pin value to HIGH for each note

    // delays for delayAmount time before setting pin value back to LOW(off) and delays again

    // before playing next note

    for (i=0;i<loopTime;i++)

    {

        digitalWrite(buzzPin,HIGH);

        delayMicroseconds(delayAmount);

        digitalWrite(buzzPin,LOW);

        delayMicroseconds(delayAmount);

    }

    digitalWrite(LED\_BUILTIN, LOW);

    // set LED to low to show note has ended

    delay(20);

    //a little delay to make all notes sound separate

}

// notes of imperial march with their corresponding frequencies and time in milliseconds

void imperialMarch()

{

    note(buzzPin, a, 500);

    note(buzzPin, a, 500);

    note(buzzPin, a, 500);

    note(buzzPin, f, 350);

    note(buzzPin, cH, 150);

    note(buzzPin, a, 500);

    note(buzzPin, f, 350);

    note(buzzPin, cH, 150);

    note(buzzPin, a, 1000);

    note(buzzPin, eH, 500);

    note(buzzPin, eH, 500);

    note(buzzPin, eH, 500);

    note(buzzPin, fH, 350);

    note(buzzPin, cH, 150);

    note(buzzPin, gS, 500);

    note(buzzPin, f, 350);

    note(buzzPin, cH, 150);

    note(buzzPin, a, 1000);

    note(buzzPin, aH, 500);

    note(buzzPin, a, 350);

    note(buzzPin, a, 150);

    note(buzzPin, aH, 500);

    note(buzzPin, gSH, 250);

    note(buzzPin, gH, 250);

    note(buzzPin, fSH, 125);

    note(buzzPin, fH, 125);

    note(buzzPin, fSH, 250);

    delay(250);

    note(buzzPin, aS, 250);

    note(buzzPin, dSH, 500);

    note(buzzPin, dH, 250);

    note(buzzPin, cSH, 250);

    note(buzzPin, cH, 125);

    note(buzzPin, b, 125);

    note(buzzPin, cH, 250);

    delay(250);

    note(buzzPin, f, 125);

    note(buzzPin, gS, 500);

    note(buzzPin, f, 375);

    note(buzzPin, a, 125);

    note(buzzPin, cH, 500);

    note(buzzPin, a, 375);

    note(buzzPin, cH, 125);

    note(buzzPin, eH, 1000);

    note(buzzPin, aH, 500);

    note(buzzPin, a, 350);

    note(buzzPin, a, 150);

    note(buzzPin, aH, 500);

    note(buzzPin, gSH, 250);

    note(buzzPin, gH, 250);

    note(buzzPin, fSH, 125);

    note(buzzPin, fH, 125);

    note(buzzPin, fSH, 250);

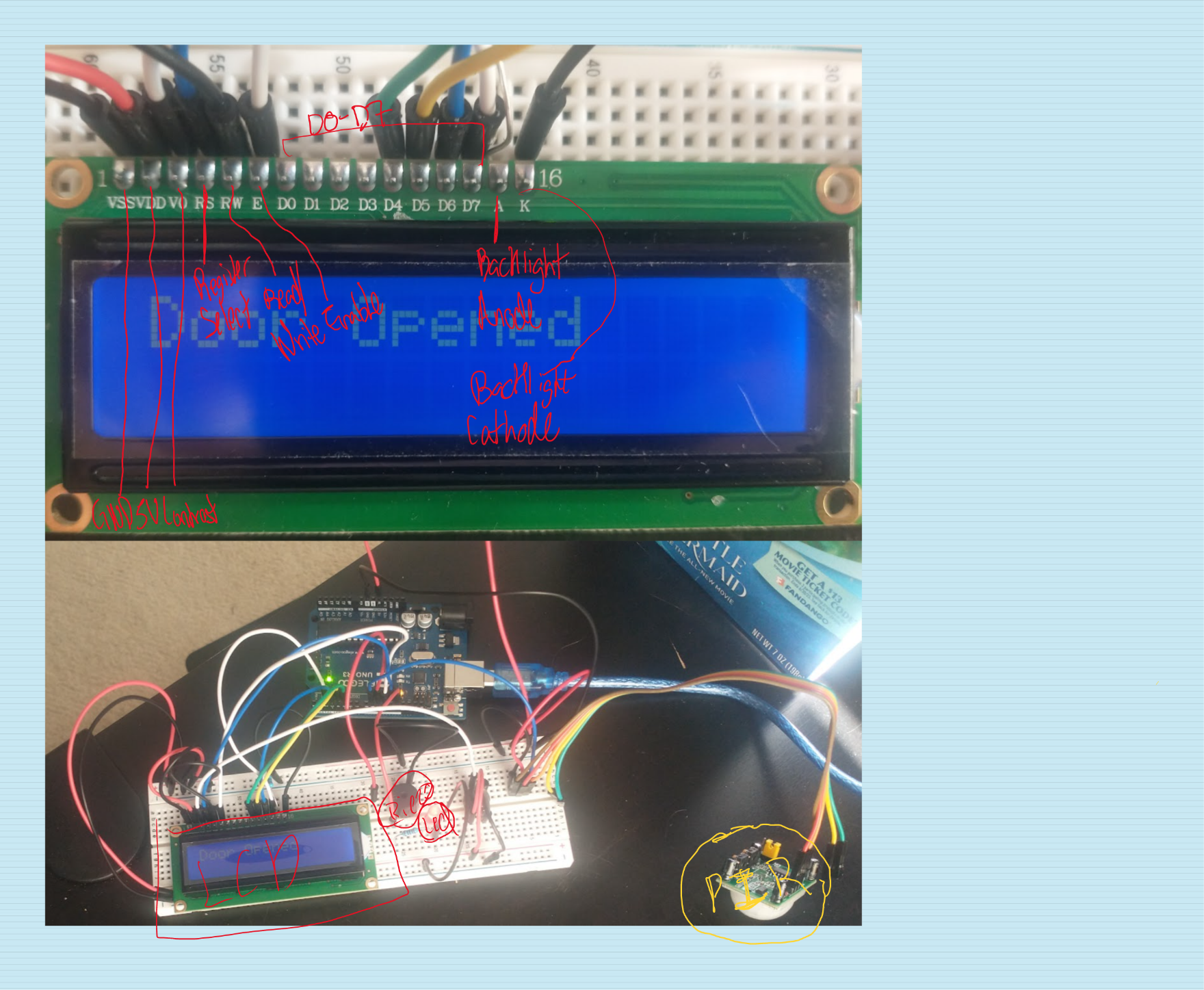
    delay(250);

    note(buzzPin, aS, 250);

    note(buzzPin, dSH, 500);

    note(buzzPin, dH, 250);

    note(buzzPin, cSH, 250);



    note(buzzPin, cH, 125);

    note(buzzPin, b, 125);

    note(buzzPin, cH, 250);

    delay(250);

    note(buzzPin, f, 250);

    note(buzzPin, gS, 500);

    note(buzzPin, f, 375);

    note(buzzPin, cH, 125);

    note(buzzPin, a, 500);

    note(buzzPin, f, 375);

    note(buzzPin, c, 125);

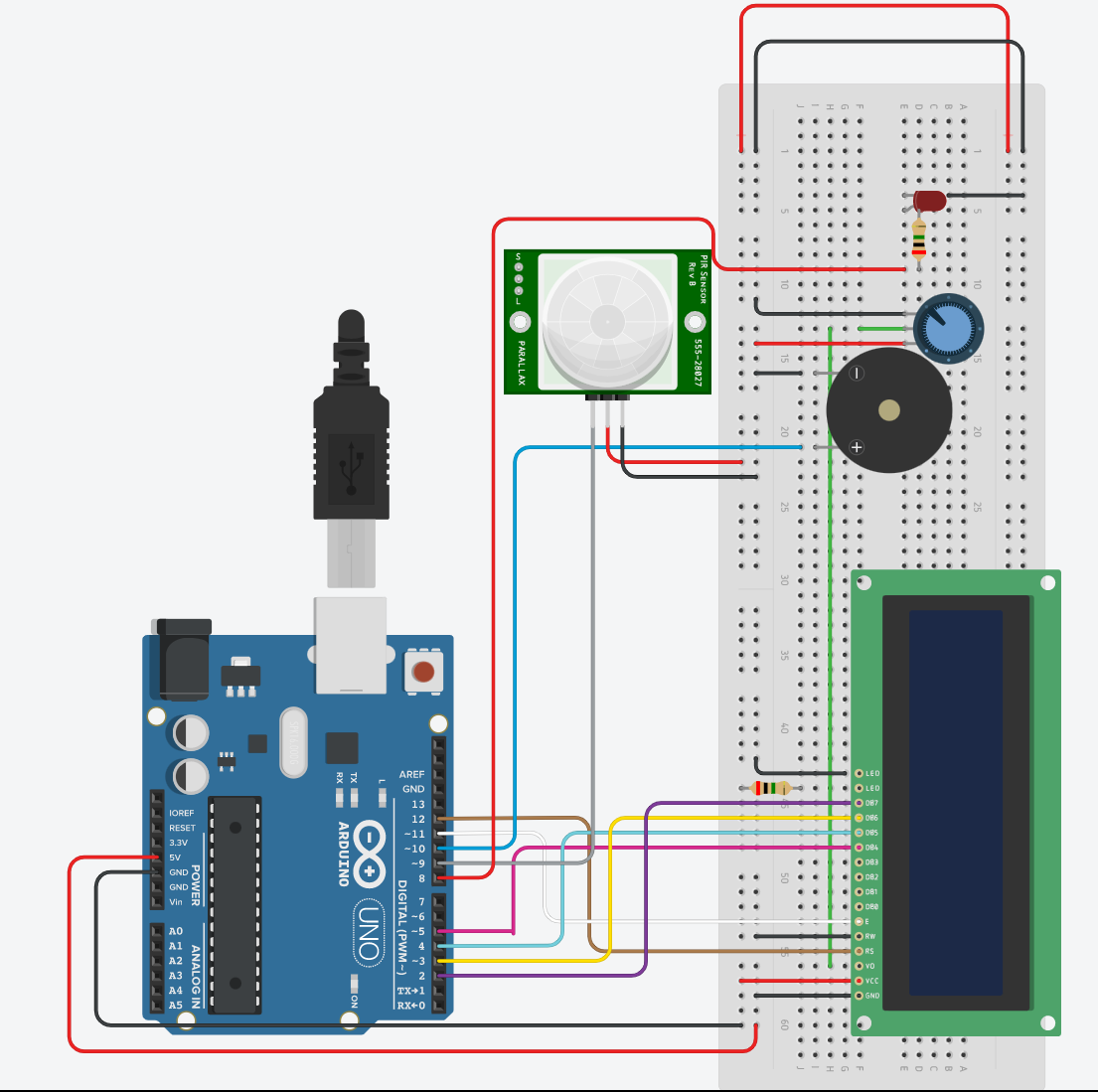
    note(buzzPin, a, 1000);

}

|  |  |  |
| --- | --- | --- |
| **Name** | **Quantity** | **Component** |
| UUNO | 1 | Arduino Uno R3 |
| PIRPIR | 1 | PIR Sensor |
| RResistor R1 | 2 | 2000 kΩ Resistor |
| PIEZObuzzer | 1 | Piezo |
| ULCD | 1 | LCD 16 x 2 |
| Rpotpot | 1 | 10000 kΩ Potentiometer |
| D1 | 1 | Red LED |

**Parts List**

**Circuit**

****