MikeS11 Proton Pack Code - How to prepare the software environment and upload the firmware to your Arduino Nano hardware.

Warning:

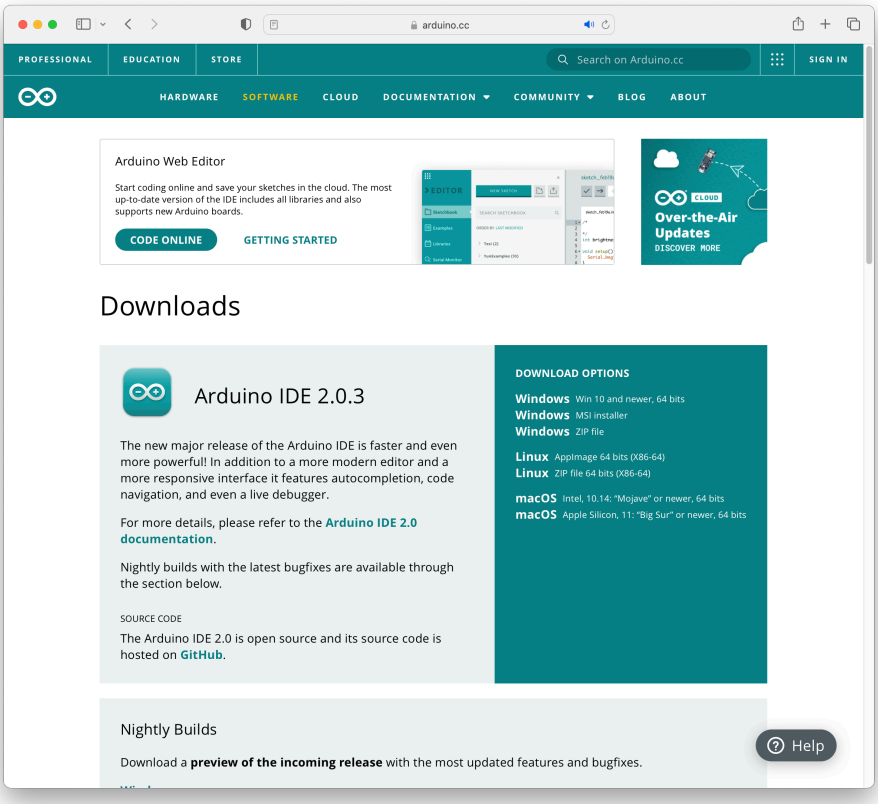
When loading code on to the Nano it is best to pull it out of the pub so there is nothing connected to the Nanos GPIO pins. As soon as the Nano is plugged in via USB it will be drawing power from your PC. If your battery/power source is plugged in to the nano this could easily cause damage to your PC and everything in-between. In turn, if the battery is disconnected the attached LEDs etc will try to pull power from your PC through the Nano and again cause damage to the PC/Nano. The components will be trying to pull a higher amperage from the USB connection than your usb port or Nano is spec'ed to handle.

The Process - The Arduino IDE Application

For simplicity Im going to describe using the Arduino IDE software to transfer the code from your Mac to the Nano but there are alternatives like Visual Studio Code.

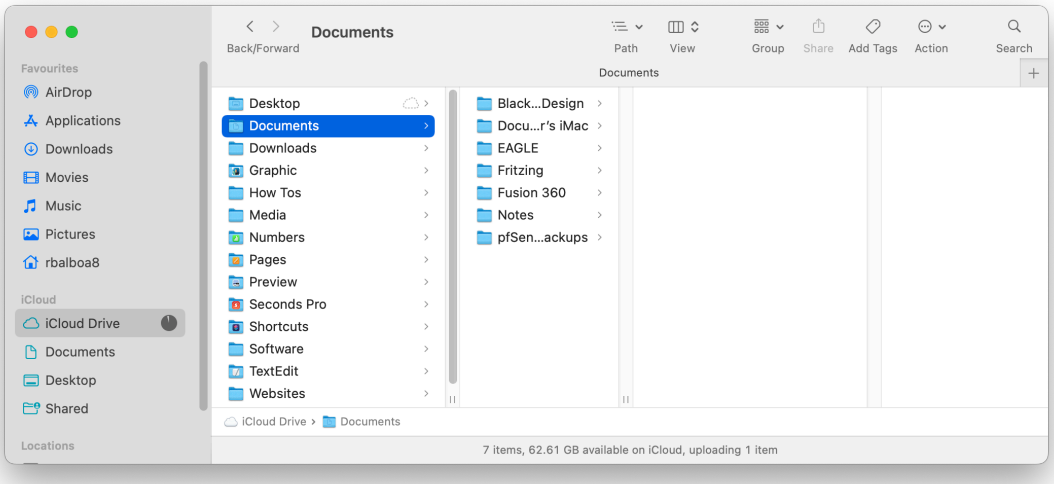
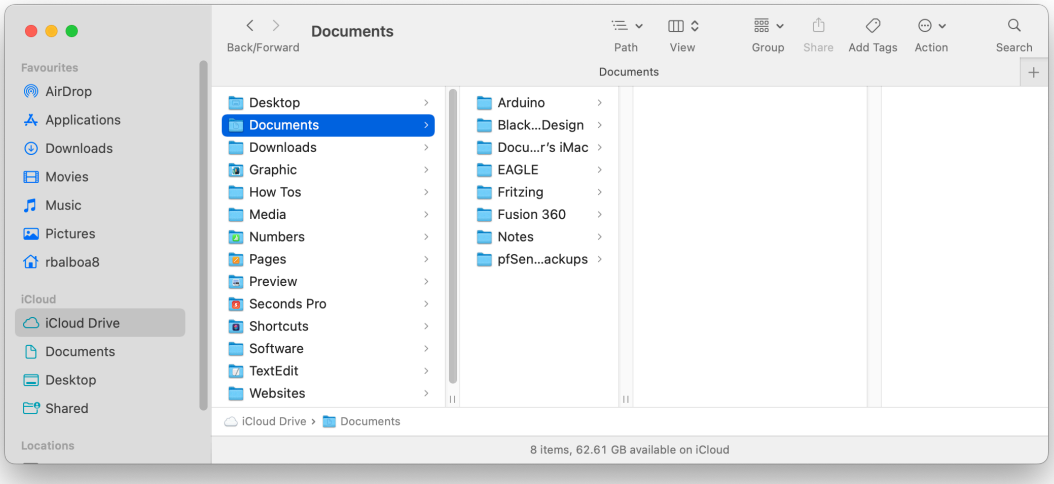
Download the applicable version of the Arduino software for your system from

https://www.arduino.cc/en/software



The current version at the point of writing this document is version: 2.0.3.

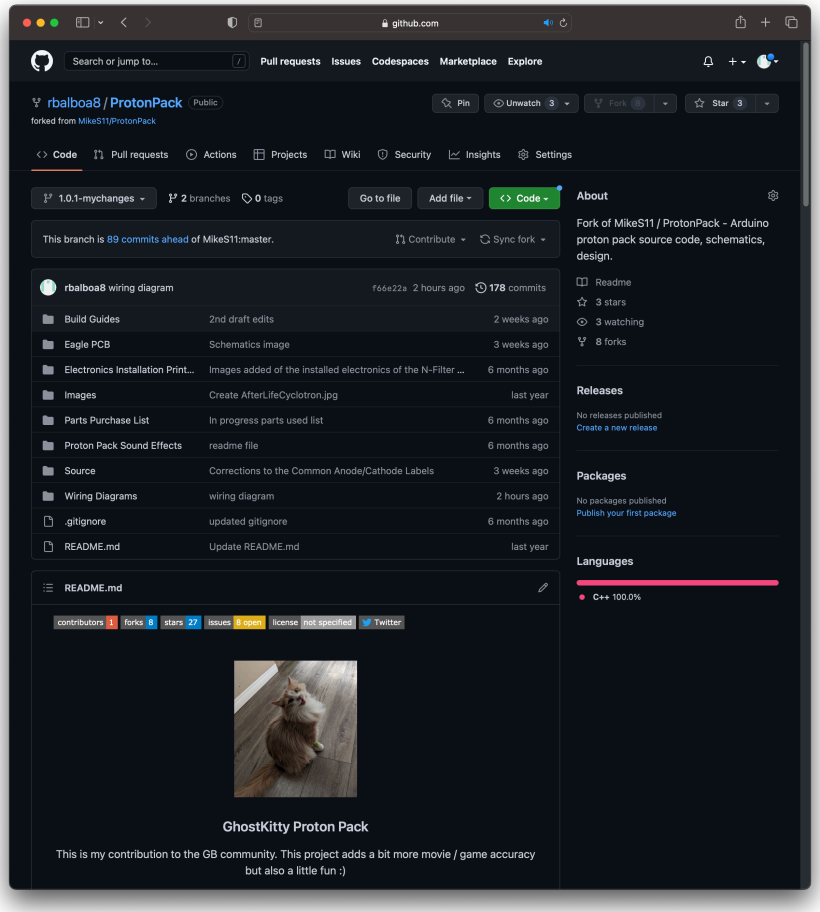
Once downloaded install the application and open it for the first time. When you open the application for the first time it will add a directory at the root of your current users "Home" directory called "Arduino". We will use this shortly, for now close the Arduino IDE application.

Before	
After	

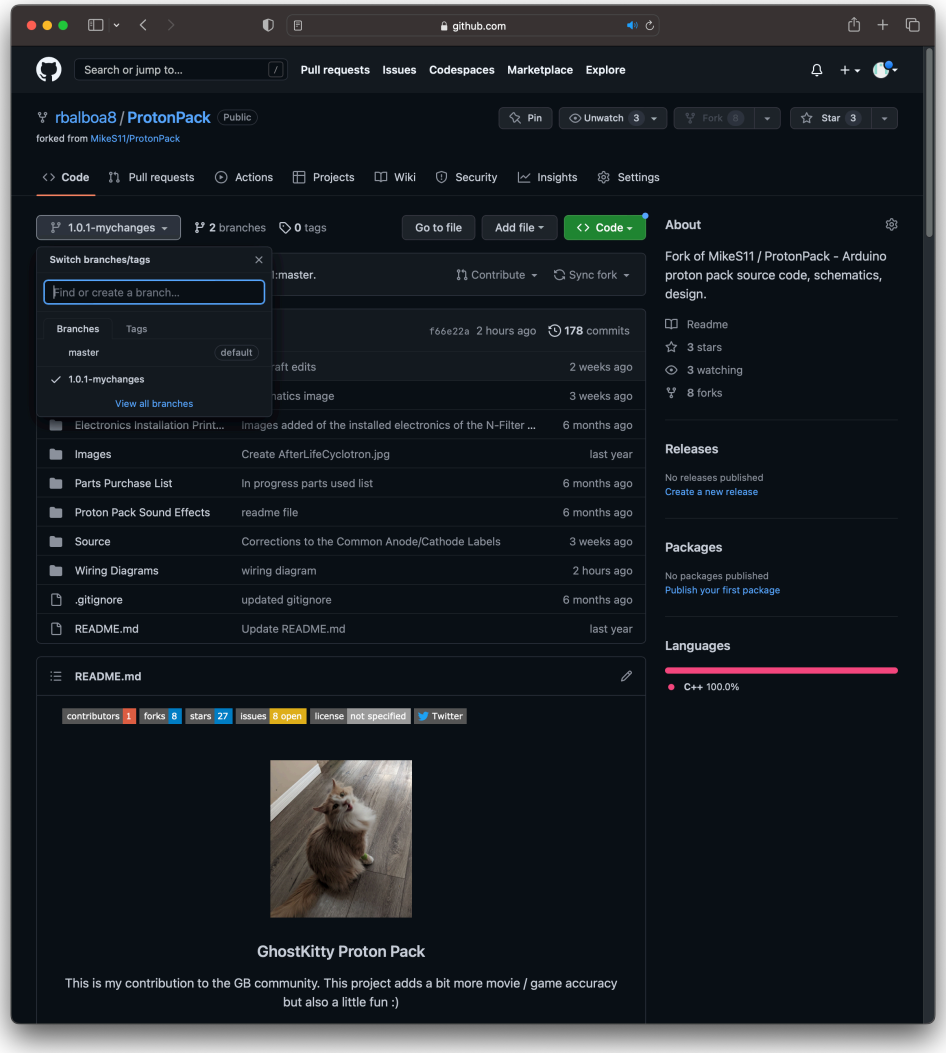
Getting the code

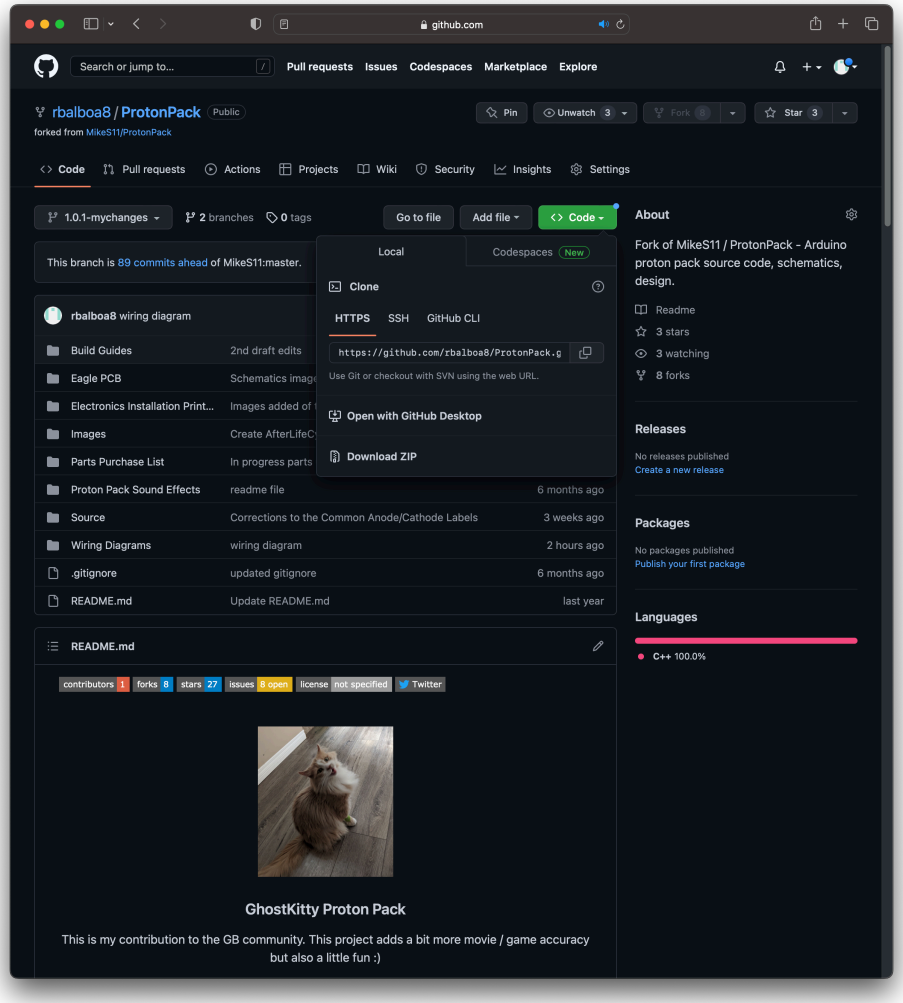
You are going to need the project code or sketch, as its know in the Arduino world. For those new to Arduino, this code will be used by the Arduino IDE application and converted in to machine code, that will then uploaded and be stored on the Nano microcontroller. The brain of this electronics project. Once this is uploaded and the Nano reboots you should have all the functionality of a replica Ghostbusters Proton Pack as specified on the main project web page. This process of converting the Arduino sketch/code in to something the Arduino Nano understands is called compiling.

The original project code can be found on GitHub ,which is an online hosting platform used primarily for software sharing/distribution. When you follow the link below you will find the main page and it features a large green button labelled “<> code” (ii). Select this button and then the “Download ZIP” option that should have now appeared.

(i) My GitHub Fork of the MikeS11 Project	

(ii) Choosing the correct branch



(iii) Downloading the files	

NOTE: There are a couple of issues with the original code that you would want to fix. This could be done manually but you can also download the “copy” of the code that I have put on GitHub with these changes already in place. One of the benefits of GitHub is that users can make a “Fork”, or copy, of the original project on to their own GitHub account. They can then make their own duplicate of the project to allow them make changes safely without affecting the original version of the code. This would be called a “Branch” (ii). This is part of the way projects are managed and how new features and possible bugs are worked out without affecting the original working code. This is exactly what I have done and you can find the links to both the original and “My changes” branch of the code at the below links.

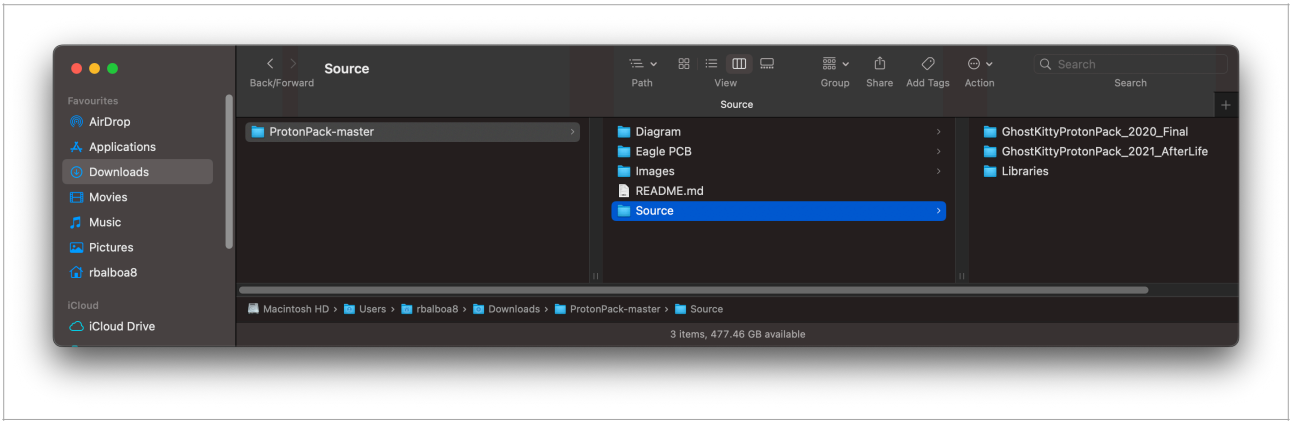
Original Code:
https://github.com/rbalboa8/ProtonPack

My fork of the code with the changes I have made (i):
https://github.com/rbalboa8/ProtonPack/tree/1.0.1-mychanges

What I have changed:

####ToDo###

Once you have downloaded the code unzip the download, if your OS does not do so automatically, and open the main directory. You should see a folder labelled “Source” and you should open this to find three further folders.

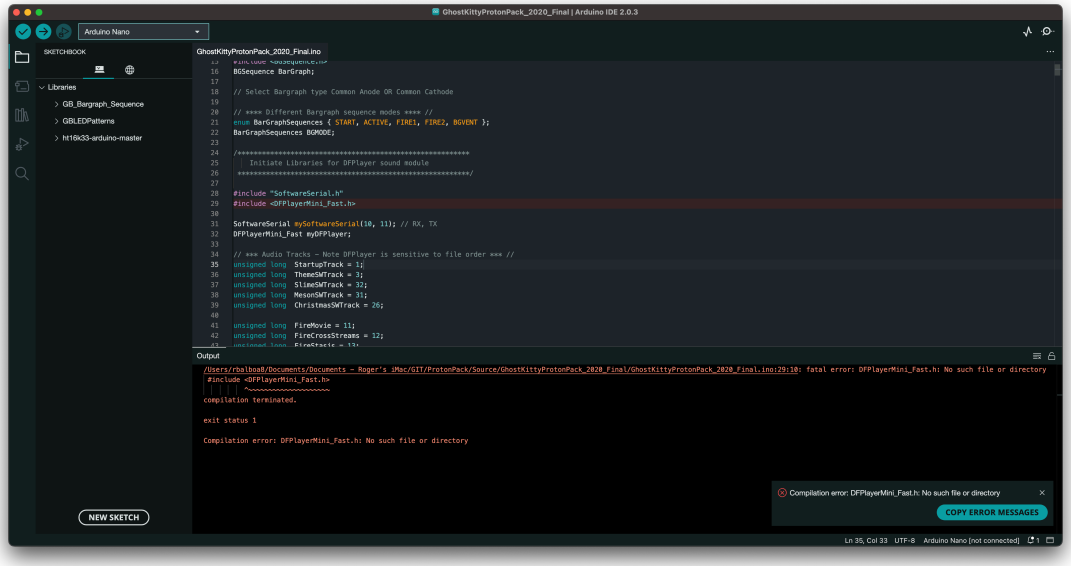


The first is labelled “GhostKittyProtonPack_2020_Final” and is the project code folder you will want later if you are building a Ghostbusters 84/89 pack.

The “GhostKittyProtonPack_2021_AfterLife” folder is for those building the Phoebe/Spengler pack from Ghostbusters Afterlife. You will only be using one of these directories dependent on your needs, the other can be ignored.

The final folder “Libraries” is needed by the Arduino IDE and we will put this in the right place on your PC now. These are special libraries that have been specially written for this project. For this reason we are manually installing them with a “Drag & Drop” method later. We will also be using some mainstream libraries that can be installed automatically via a menu in the Arduino IDE application.

Arduino Libraries make our lives easier. Someone has taken the time to write all the instructions for how the microcontroller, like the Nano we are using in this project, to know HOW to interface with other hardware or how to perform repeatable effects like LED sequence routines. The main code can therefore just reference these functions within in these libraries via simple names rather than repeated large blocks of code. Again, the reason to explain this is so you can understand if the Arduino IDE does not have all the required libraries for this particular code it will give you an error. If you see these errors you should have a little understanding what you might have missed.

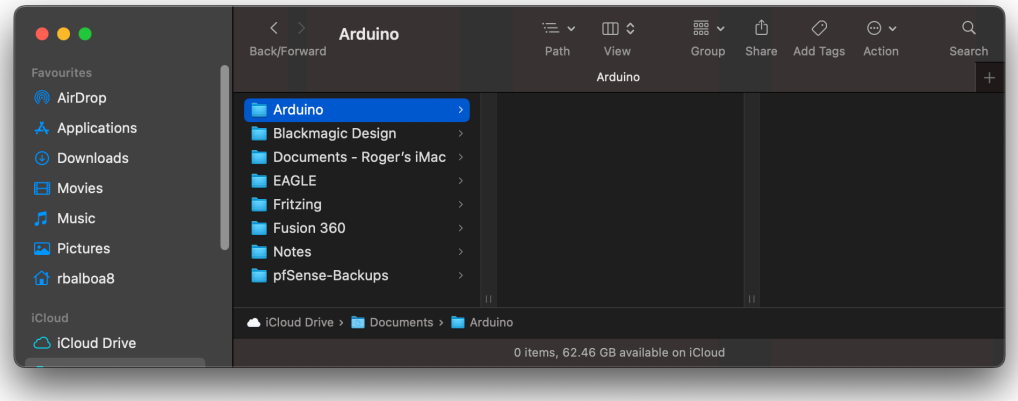
Example of a missing Library error	

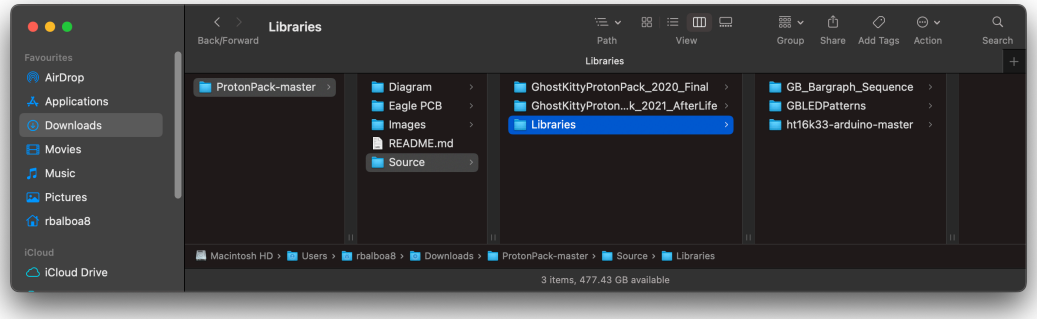
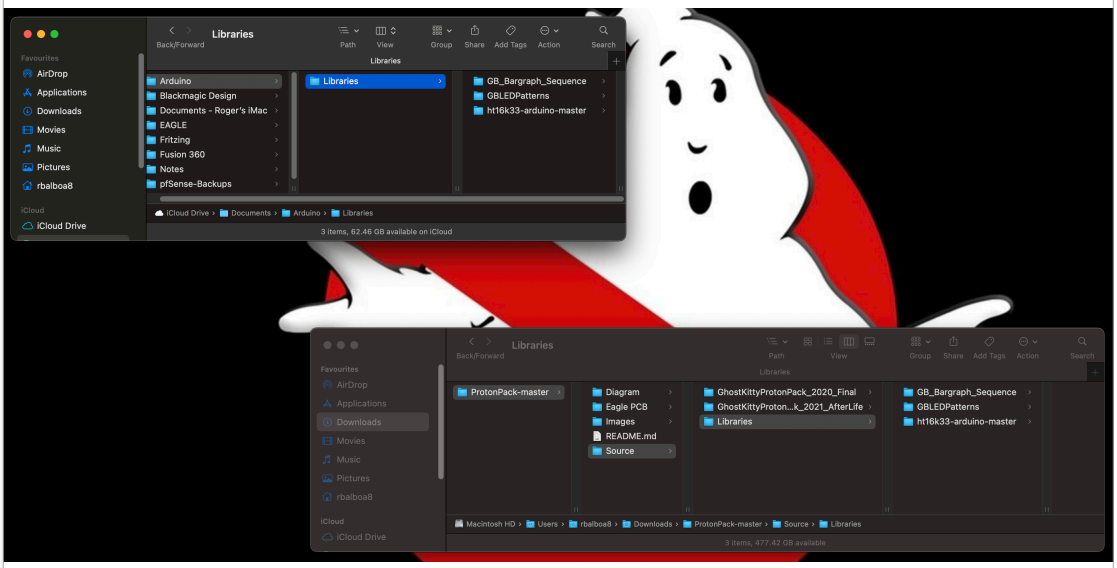
So let's install the libraries. NOTE: Again I have made changes within some of the libraries code to correct issues, so you may want to consider using these over the original code. The choice is yours.

Setting up the Arduino IDE Environment

Installing the downloaded Arduino Libraries:

From your GitHub project download drag/copy the “Libraries” folder in to the Arduino directory that was created when we opened the software for the first time earlier.

The Arduino User/ Documents folder	

The downloaded GitHub directory	
Before manually installing the Arduino libraries.	
After manually installing the Arduino libraries.	

Once this is in place you can plug in your Nano to the PC THEN open the Arduino IDE once more. NOTE: If you want to check that your Mac can “see” your Arduino board and doesn’t need additional drivers you can perform the additional instructions below.

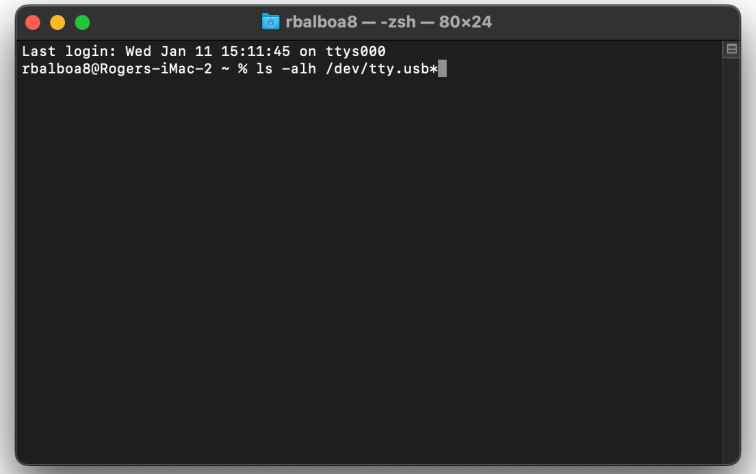
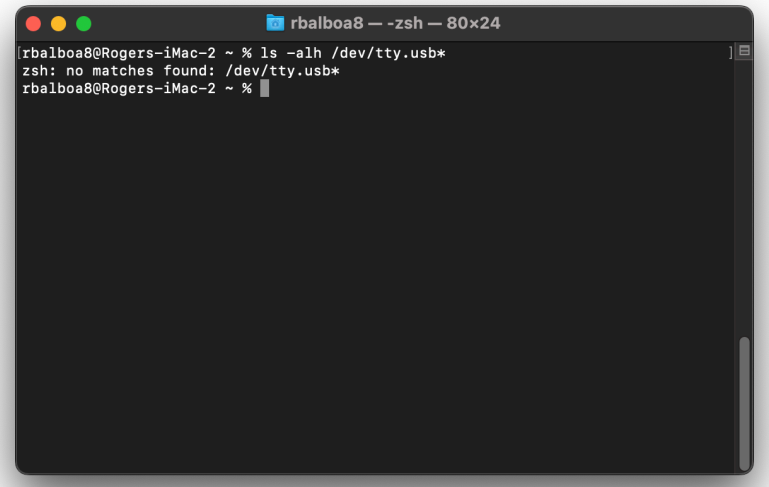
MacOS specific: Before you plug in your Nano via the USB cable, open a Terminal. To do this you can use the MacOS Spotlight search feature by pressing “Command + Space” and searching ‘terminal’. Then select the Terminal.app once it appears.

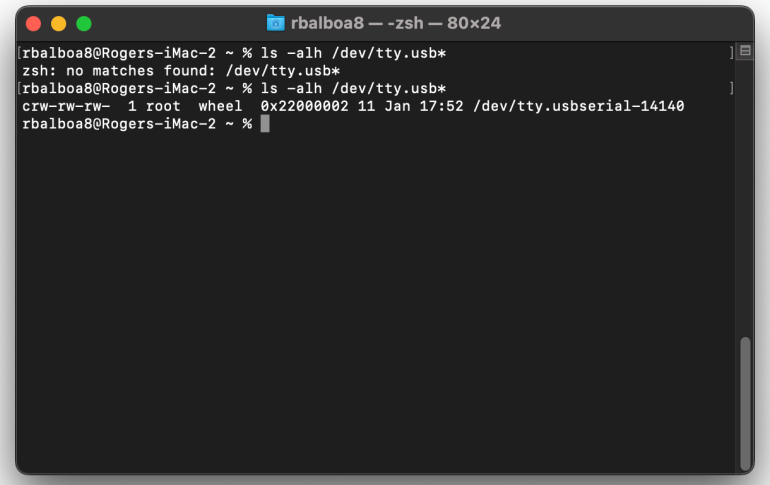
In the terminal, copy and paste the following command:

```
ls -alh /dev/tty.usb*
```

This command will give you a short list, if any, of usb serial devices it currently sees connected. Now plug in the Arduino, you should see the red led light up on the board telling you the device has power, and run the same terminal again. This time you should see a listing like the following:

```
~ % ls -alh /dev/tty.usb*
crw-rw-rw-  1 root  wheel  0x22000002 29 Dec 22:55 /dev/
tty.usbserial-141320
```

Type the command in to the terminal	
Terminal message before plugging in the Arduino Nano	

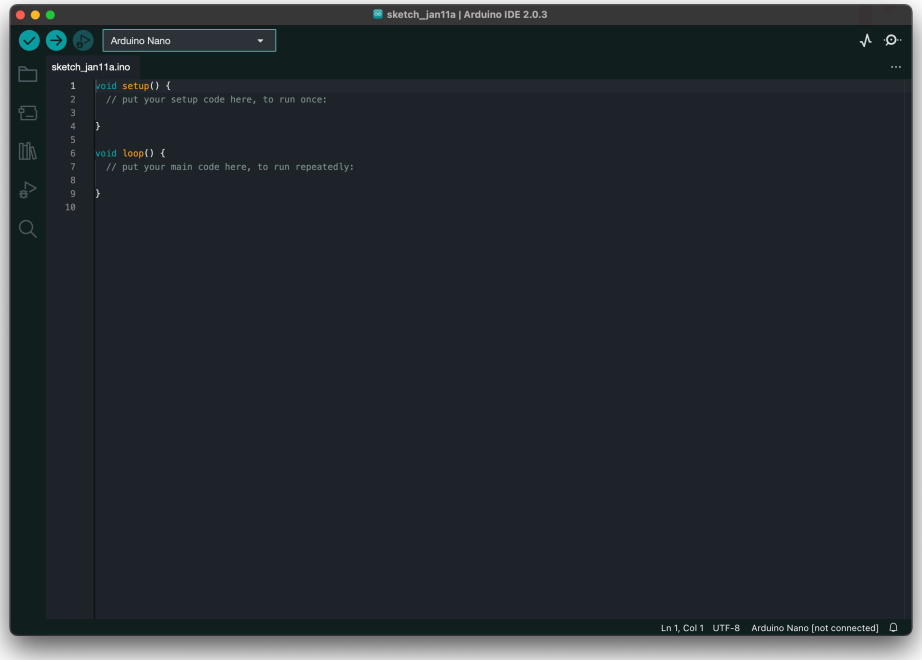
Terminal message after plugging in the Arduino Nano	

This method allows you to see the output change once the Nano is plugged in but also identify the exact device name of your Nano just incase you have other serial USB devices attached and can't distinguish which is your Nano.

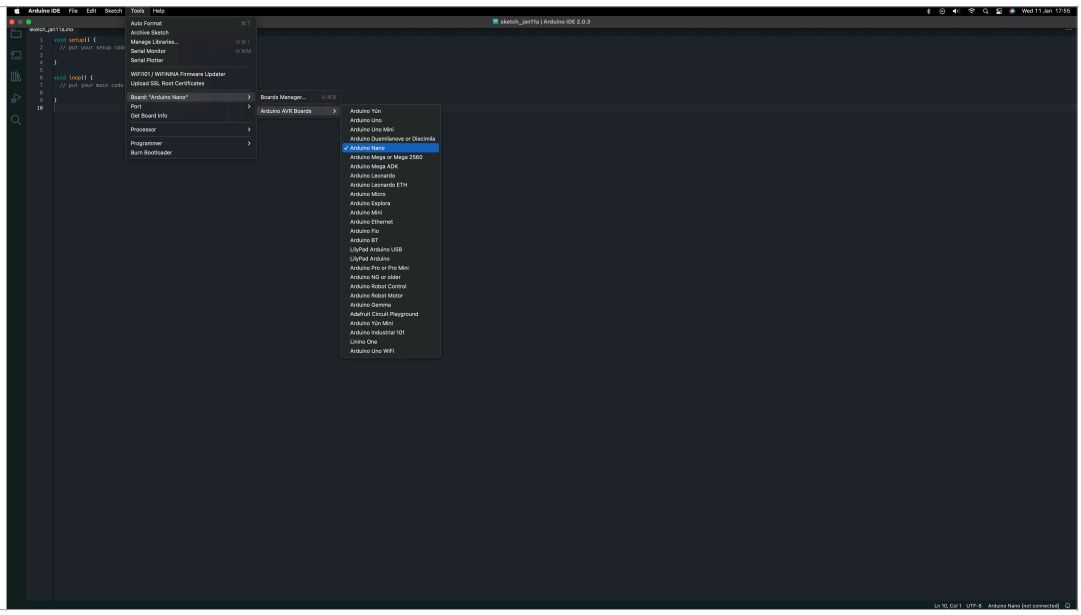
###todo troubleshooting explaination####

The Arduino IDE

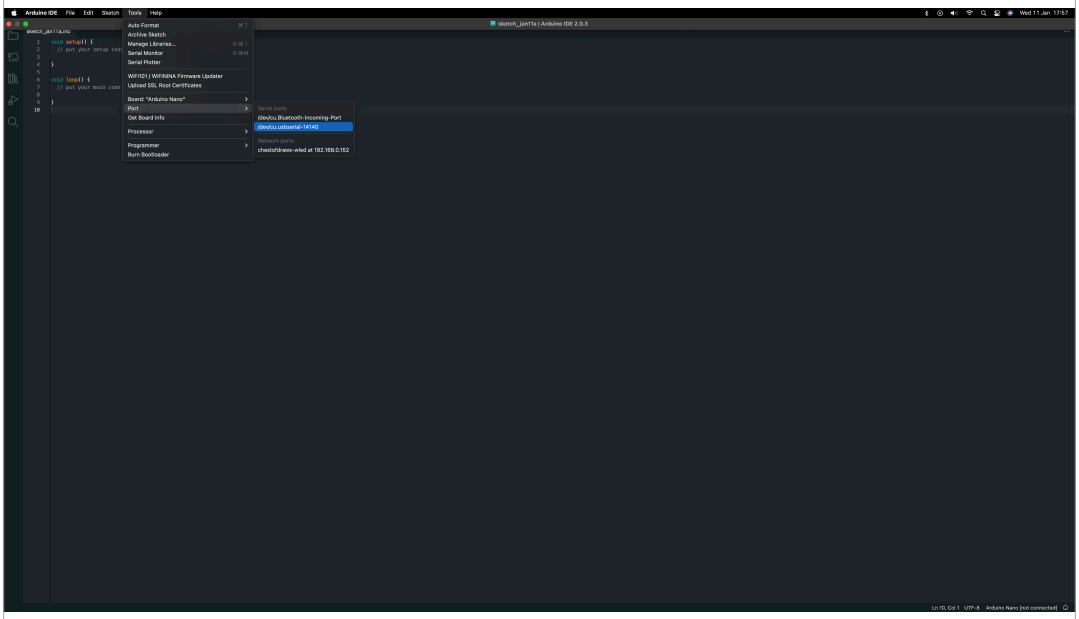
Setting up the Hardware serial communication:

Opening the Arduino IDE	

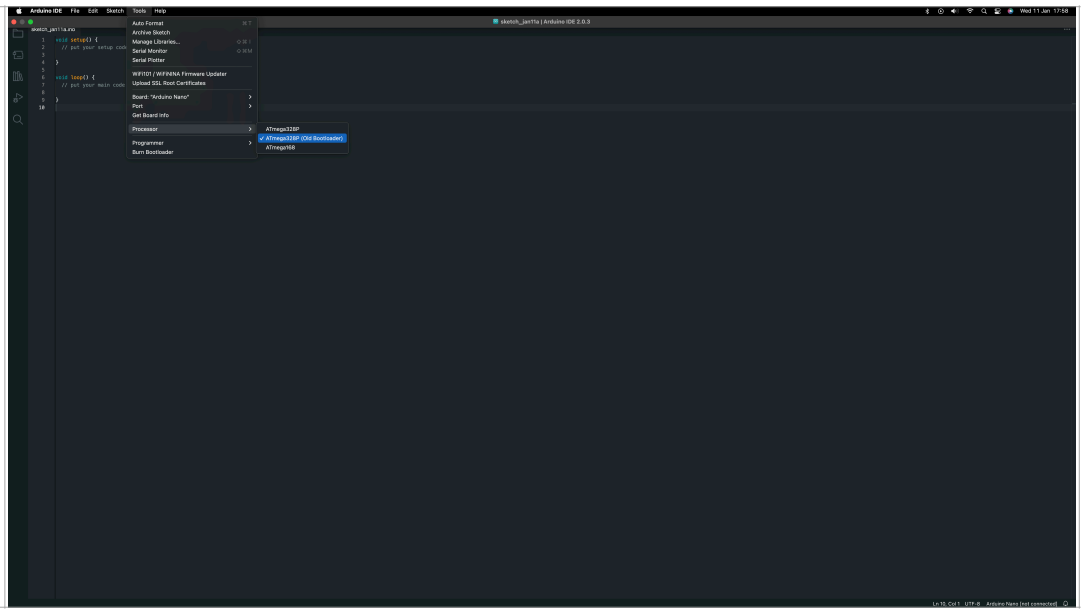
Within the Arduino application, find the Top bar and select the "Tools" menu. We are going to revisit this menu a number of times, firstly select the "Board" option that has appeared. From the next menus choose "Arduino AVR Boards" then "Arduino Nano".

Choosing the Arduino Nano as the microcontroller in for this project	

Next, return to the “Tools” menu, choose “Port” and then choose your boards device name. On a Mac you will most likely see a device labelled something like “/dev/tty.usb*”.

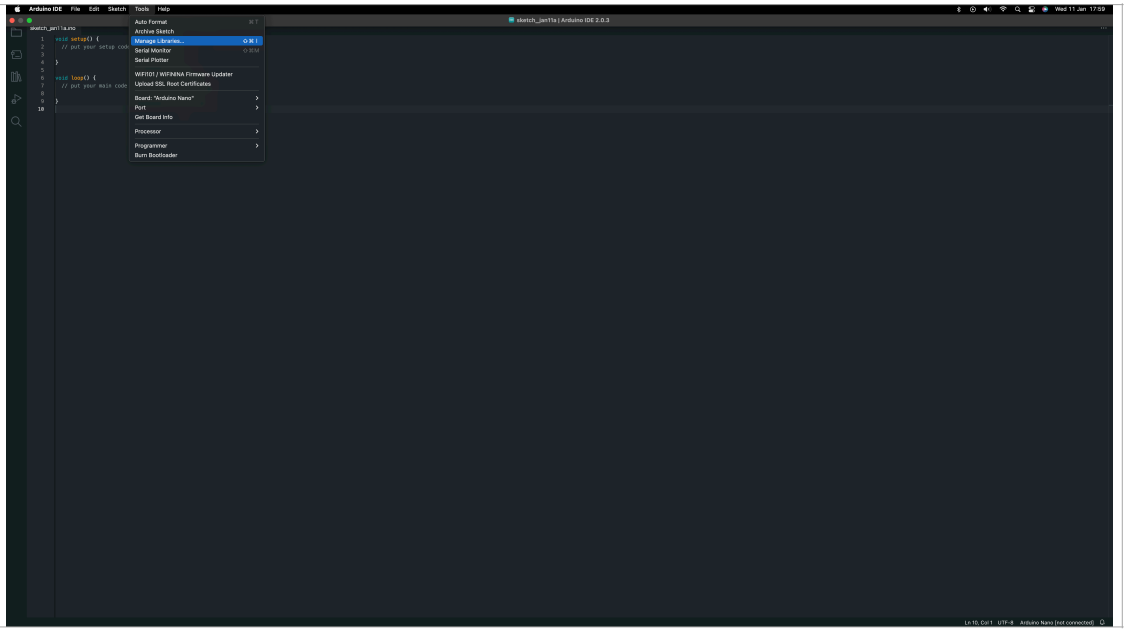
Choosing the correct communications port for the Arduino Nano plugged in	

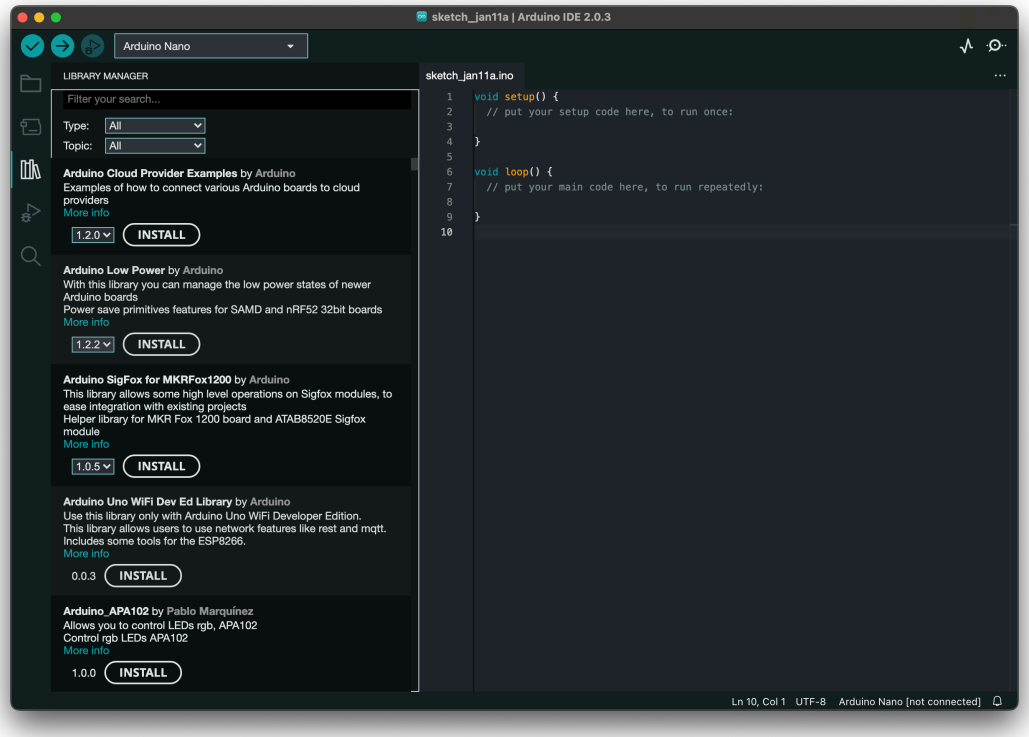
Return to the “Tools” menu once more and this time choose the “Processor” option. What you choose next depends on wether you purchased a genuine Arduino Nano Board or a clone. If your board is genuine then choose the “ATmega328P” option, for a clone choose “ATmega328P (Old Bootloader)”. It won’t harm if you choose the wrong option, you will just see multiple connection errors then an upload failure. If this happens you can change to the other option and try again.

Choosing the correct processor for the Arduino Nano	

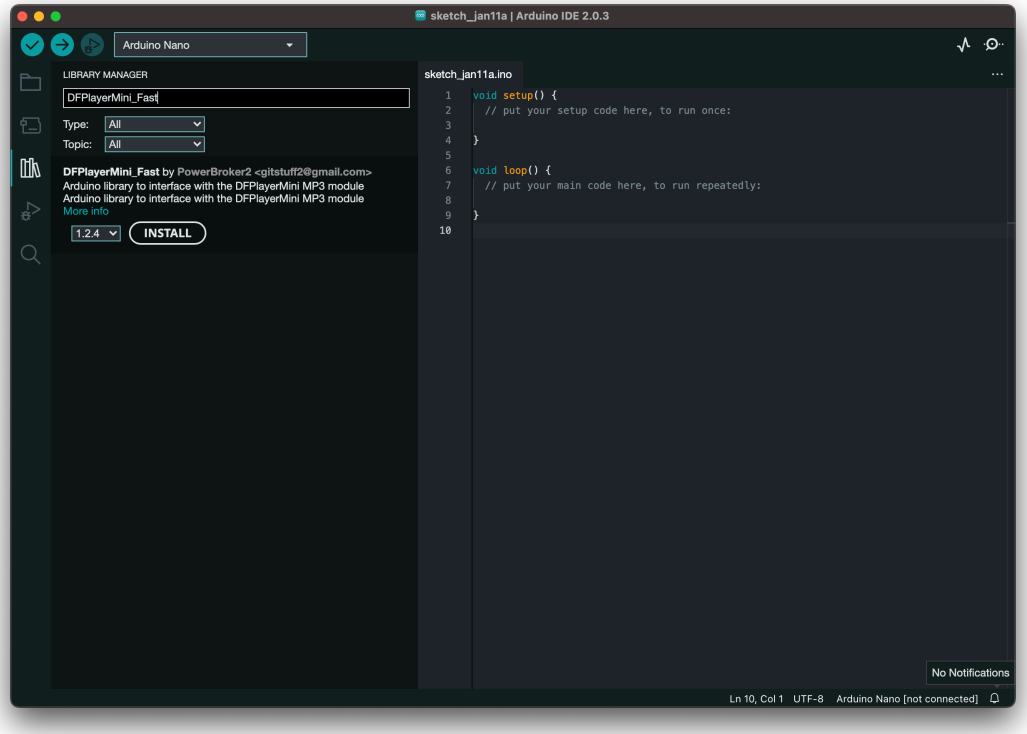
Installing additional required Libraries:

At this point your Arduino software should be setup to talk to the Nano hardware but we just need to install the last libraries required for this project. From the Top Bar return to the “Tools” menu and select the “Manage Libraries” option. You should now see the Libraries Manager open in the main windows along the left side.

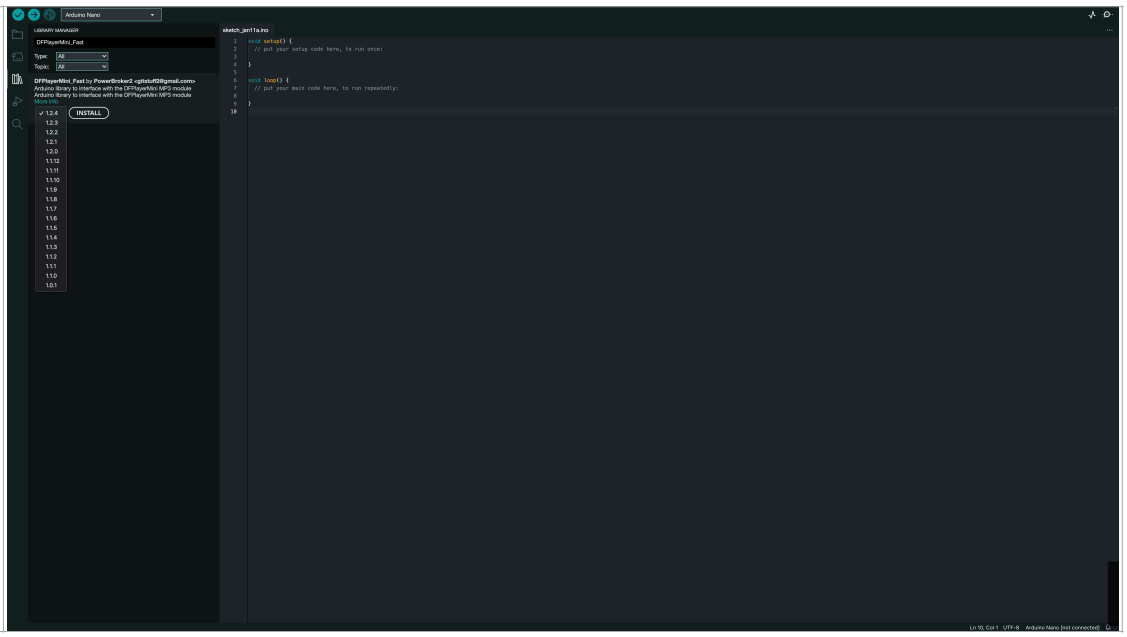
Opening the Libraries Manager from the Tools Menu	

The Libraries Manager Window	

At the top there is a search option that we will use to find the missing libraries, first search for “DFPlayerMini_Fast”. You should see one search result matching DFPlayerMini_Fast with an install button.

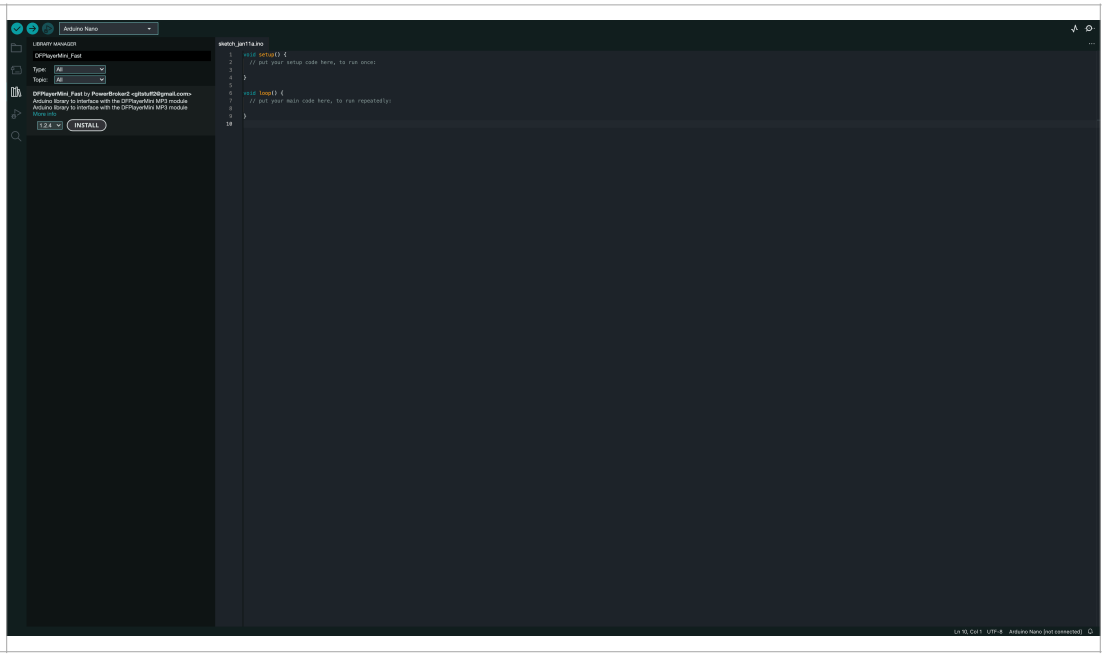
Search for libraries via the Libraries Manager	

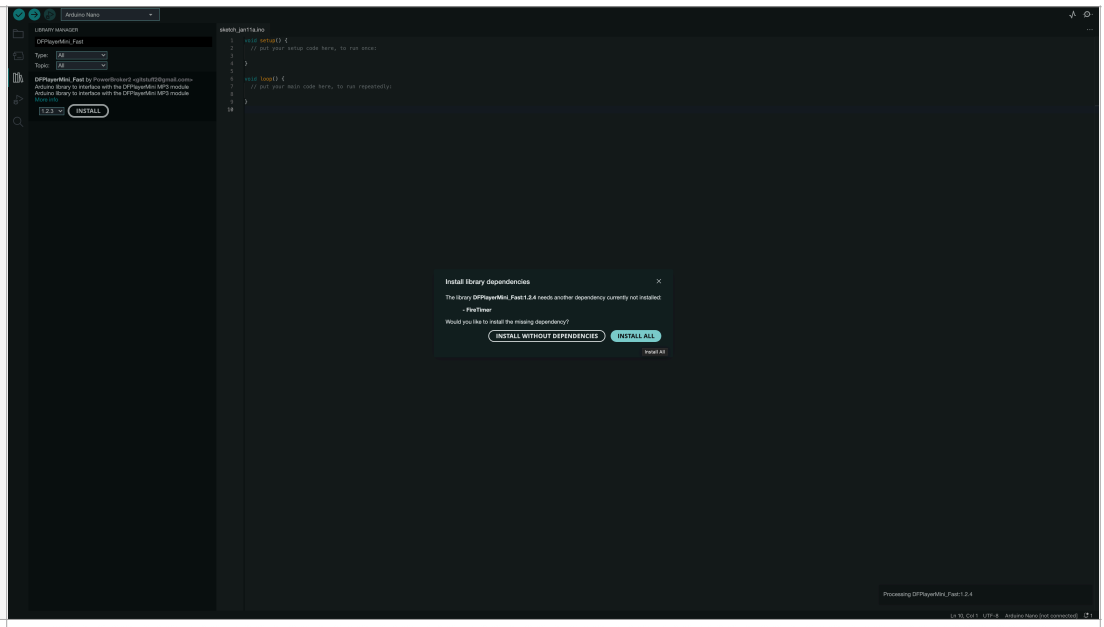
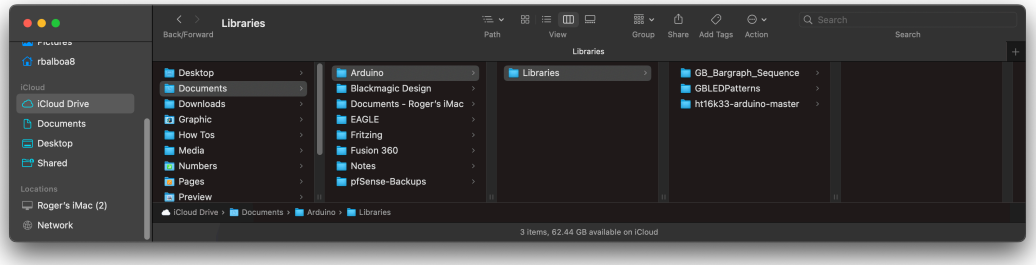
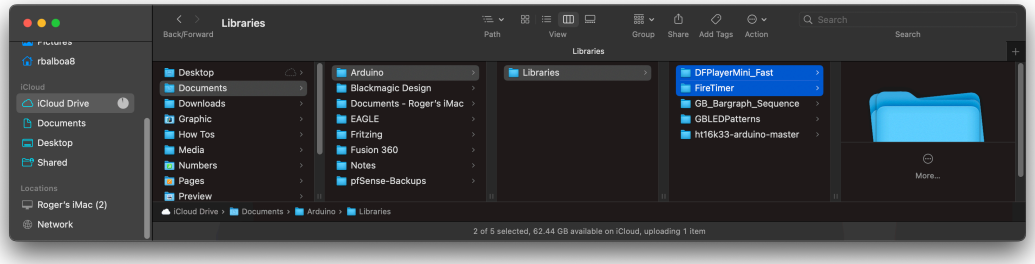
NOTE: next to the Install button is a drop down menu where you can choose the specific version of the library but in this case you can just install the latest version.

Selecting a specific version of a library	

Some projects may specify the version and will not work with some others. If you ever have problems this can be useful to know and worth checking if you see errors.

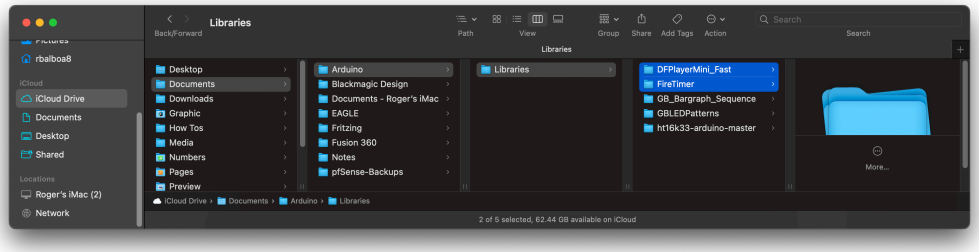
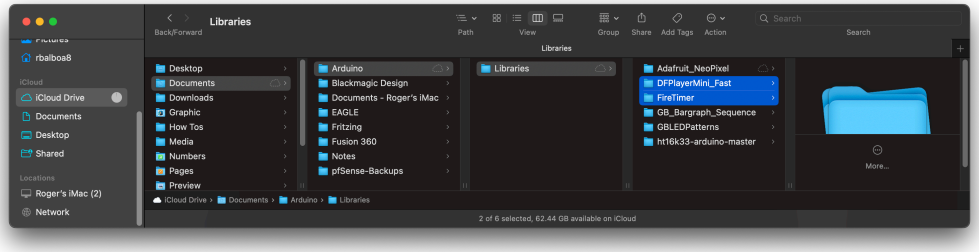
The “DFPlayerMini_Fast” library depends on the “FireTimer” library also being installed so make sure, when prompted, you choose the “Install All” option and NOT “INSTALL WITHOUT DEPENDANCIES”.

Installing the DFPlayerMini_Fast library	

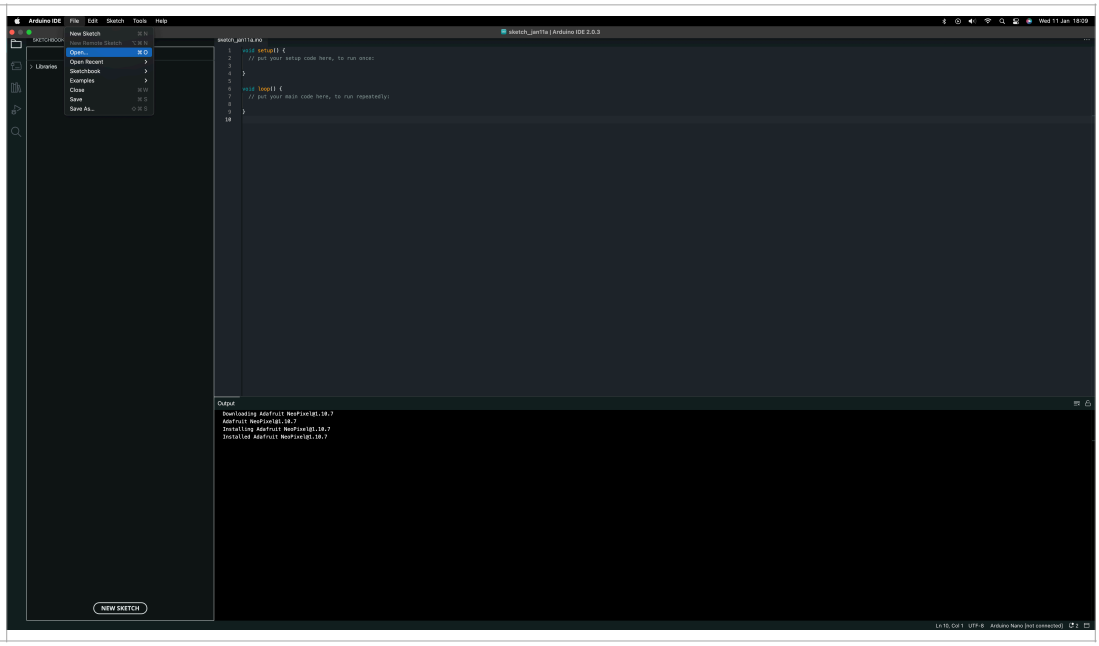
Installing dependencies for the DFPlayerMini _Fast library	
The User/ Documents/ Arduion/ Libraries folder BEFORE installing the DFPlayerMini _Fast library	
The User/ Documents/ Arduion/ Libraries folder AFTER installing the DFPlayerMini _Fast library	

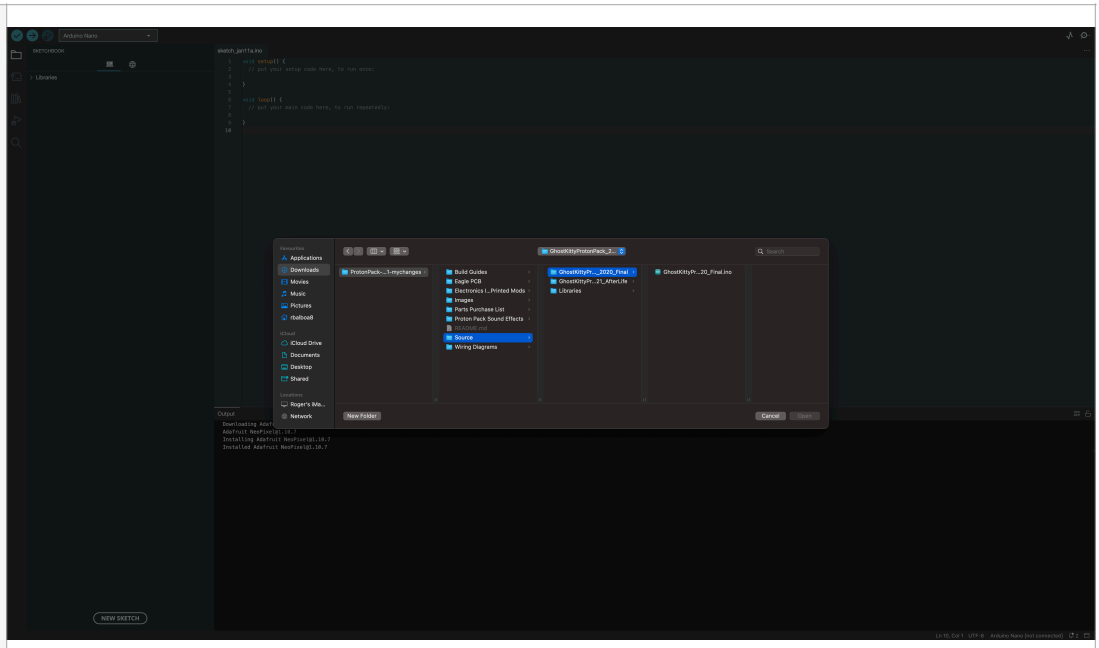
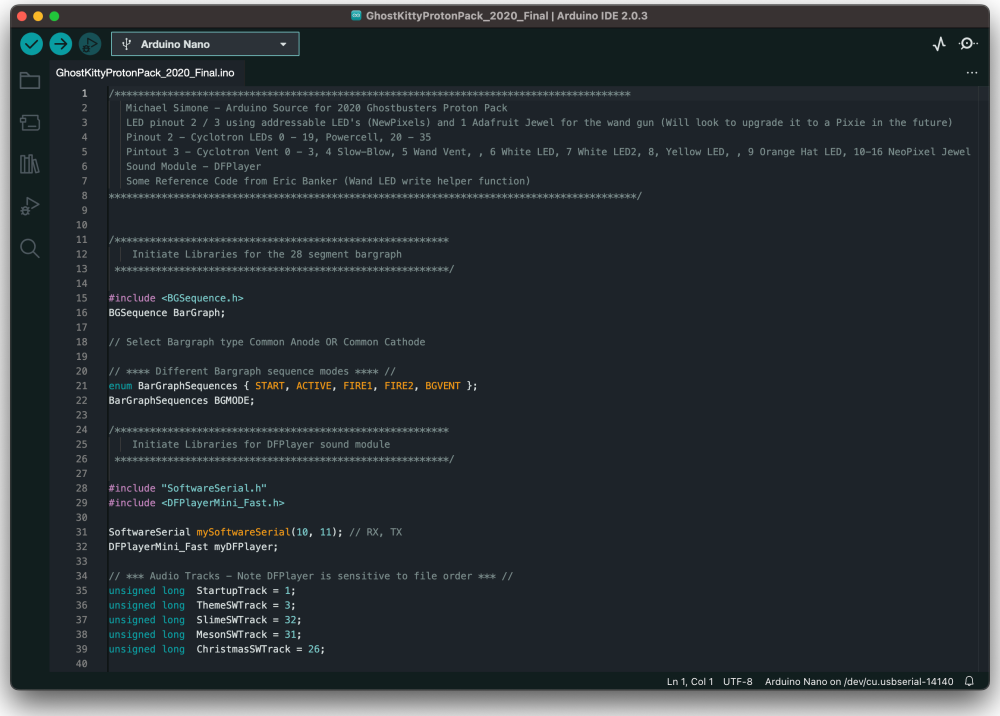
Now we just repeat the process for the “Adafruit_NeoPixel” Library from Adafruit. That should be it now for the software setup.

Installing the Adafruit_NeoPixel library	

The User/ Documents/ Arduion/Libraries folder BEFORE installing the Adafruit_NeoPixel library	
The User/ Documents/ Arduion/Libraries folder BEFORE installing the Adafruit_NeoPixel library	

Next we need to open the project code, navigate to the top bar and choose “File”, “Open” and navigate to the project folder we downloaded earlier. From there, remembering which directory we need for the version of the Proton Pack we are making, enter the directory and select the file ending “*.ino”. This is the Arduino sketch extension. In the Arduino application you should see the code appear in the main window.

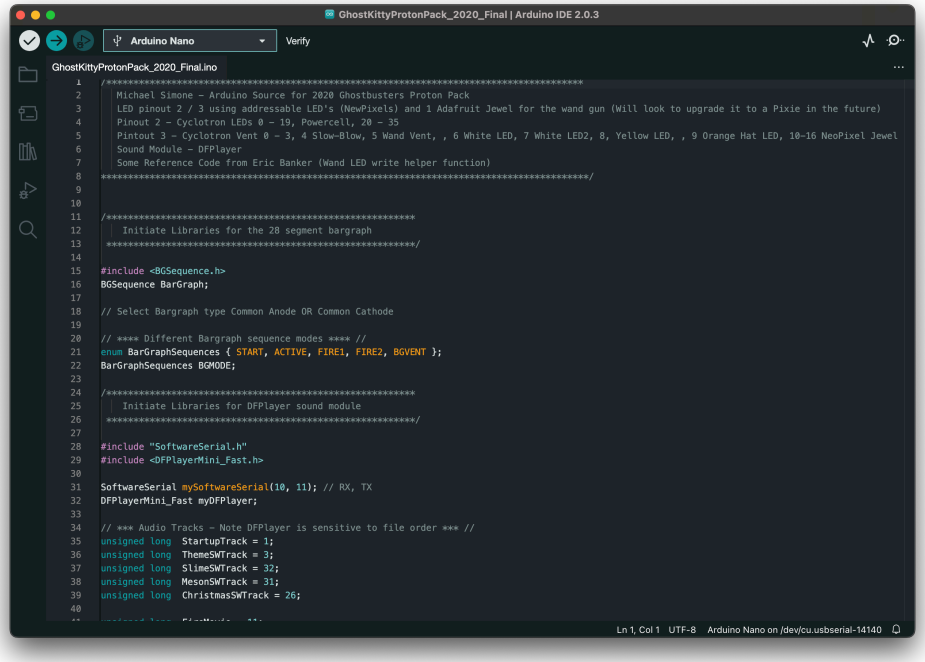
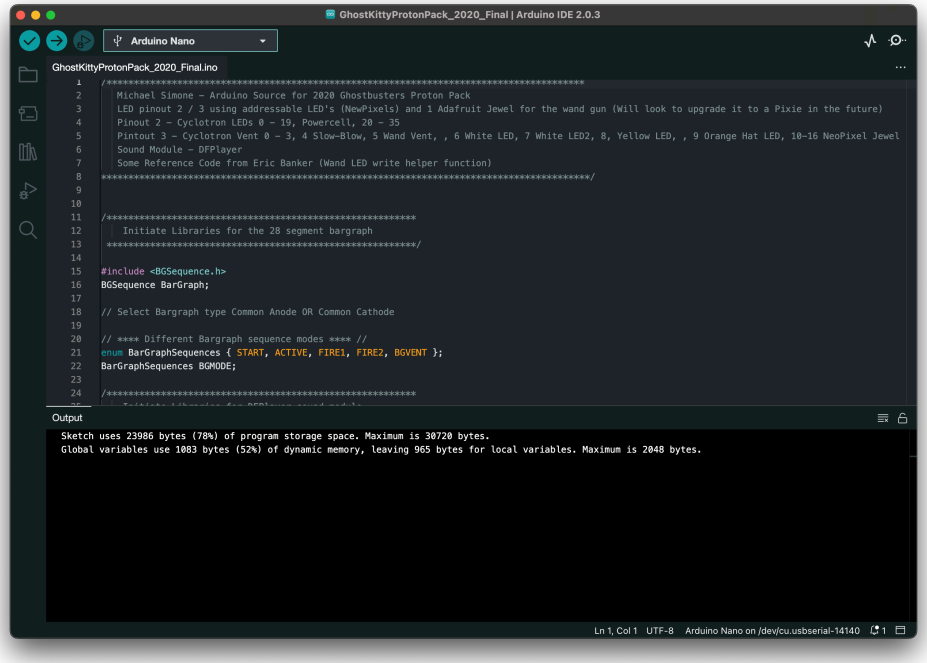
Open the project code via the File menu	

Navigate to the .ino file for the project	
The project code open in the Arduino IDE	

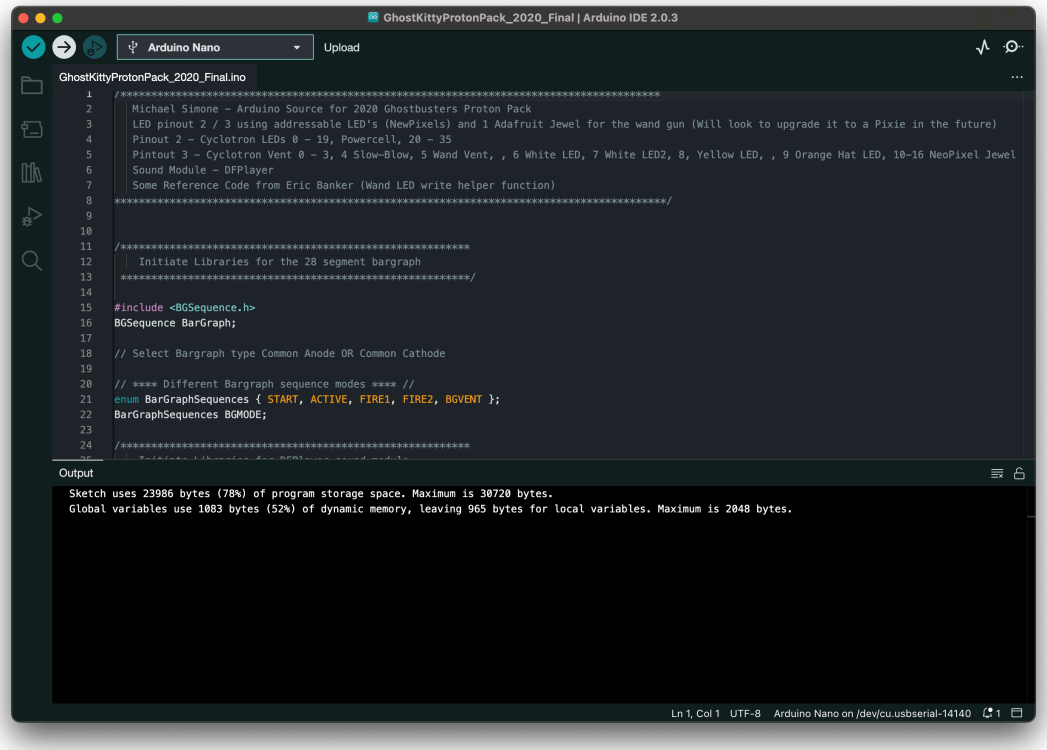
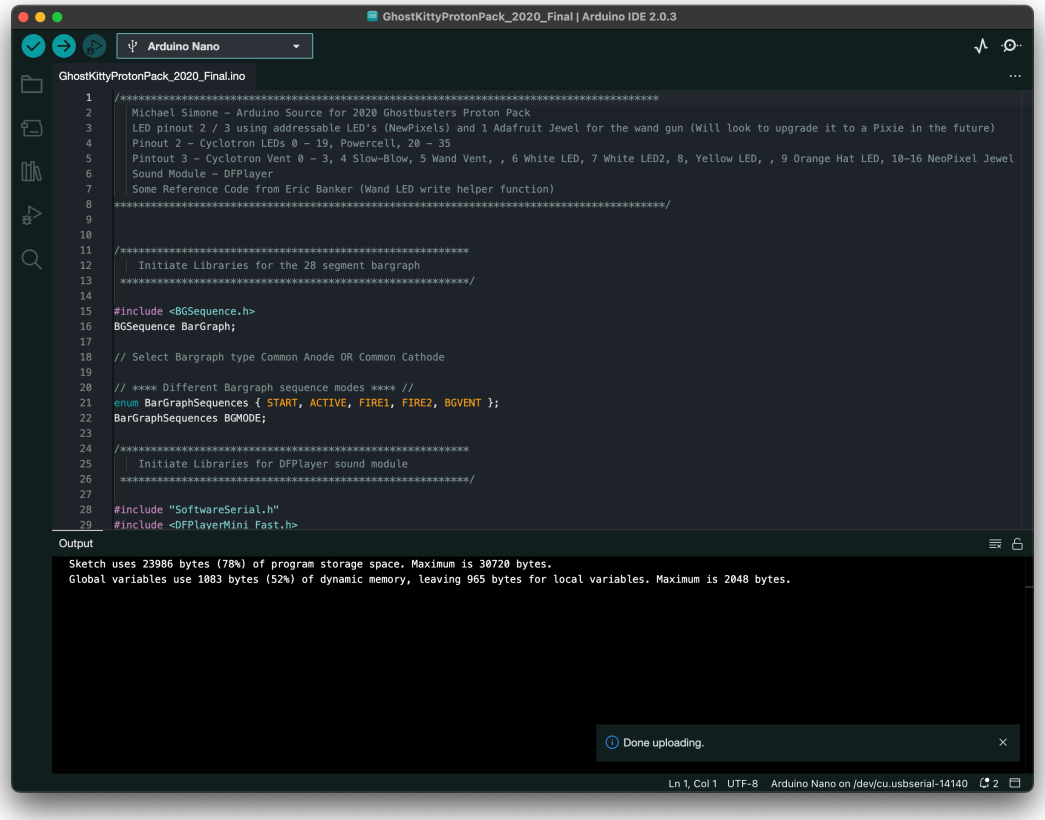

```
/*****************************************************************************************
   Michael Simone - Arduino Source for 2020 Ghostbusters Proton Pack
   LED pinout 2 / 3 using addressable LED's (NewPixels) and 1 Adafruit Jewel for the wand gun (Will look to upgrade it to a Pixie in the future)
   Pinout 2 - Cyclotron LEDs 0 - 19, Powercell, 20 - 35
   Pintout 3 - Cyclotron Vent 0 - 3, 4 Slow-Blow, 5 Wand Vent, , 6 White LED, 7 White LED2, 8, Yellow LED, , 9 Orange Hat LED, 10-16 NeoPixel Jewel
   Sound Module - DFPlayer
   Some Reference Code from Eric Banker (Wand LED write helper function)
*****************************************************************************************/


/*********************************************************
    Initiate Libraries for the 28 segment bargraph
 *********************************************************/

#include <BGSequence.h>
BGSequence BarGraph;

// Select Bargraph type Common Anode OR Common Cathode

// **** Different Bargraph sequence modes **** //
enum BarGraphSequences { START, ACTIVE, FIRE1, FIRE2, BGVENT };
BarGraphSequences BGMODE;

/*********************************************************
    Initiate Libraries for DFPlayer sound module
 *********************************************************/

#include "SoftwareSerial.h"
#include <DFPlayerMini_Fast.h>

SoftwareSerial mySoftwareSerial(10, 11); // RX, TX
DFPlayerMini_Fast myDFPlayer;

// *** Audio Tracks - Note DFPlayer is sensitive to file order *** //
unsigned long  StartupTrack = 1;
unsigned long  ThemeSWTrack = 3;
unsigned long  SlimeSWTrack = 32;
unsigned long  MesonSWTrack = 31;
unsigned long  ChristmasSWTrack = 26;
```

To be sure everything is ready we are going to compile the software without sending it to the Nano. To do this we hit the verify button identified as a “Tick” symbol over in the top left area of the Arduino application window. You will see a new panel open below the code window with a title of OUTPUT. This is where you will receive information about any errors or, if everything is good, you should see some similar output to:

Sketch uses 924 bytes (3%) of program storage space. Maximum is 30720 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.

The Verify button, used to run and check the code	
The Verify output message	

This tells you the code compiled without issue and can now be uploaded to the nano. We do this by hitting the “Upload” button, defined by a right arrow symbol next to the previously used verify/ tick button.

The Upload button used to Upload/ Install the code to the Arduino Nano	
A successful Upload/ Install message	

NOTE: if you have not previously used the “Verify” button to check the code compiles okay the “Upload” button will also run a compile before it uploads as it can’t upload anything until it is compiled. If you have already compiled, the computation part is done and the Arduino IDE will just get on with the upload. The reason for both buttons might not make immediate sense but with experience you will workout when both have their uses.

At this point your nano should be ready to use. There is no way to truly tell until the peripherals are plugged in so unplug the Nano from the PC and, keeping the nano unpowered, plug in some of the peripherals. The code is written in such a way that if none of the switches are plugged in the proton pack “On” sequence will run and the Proton Pack will then sit in its normal running state. So for testing purposes you could just plugin a few of the LEDS and your battery/power source.