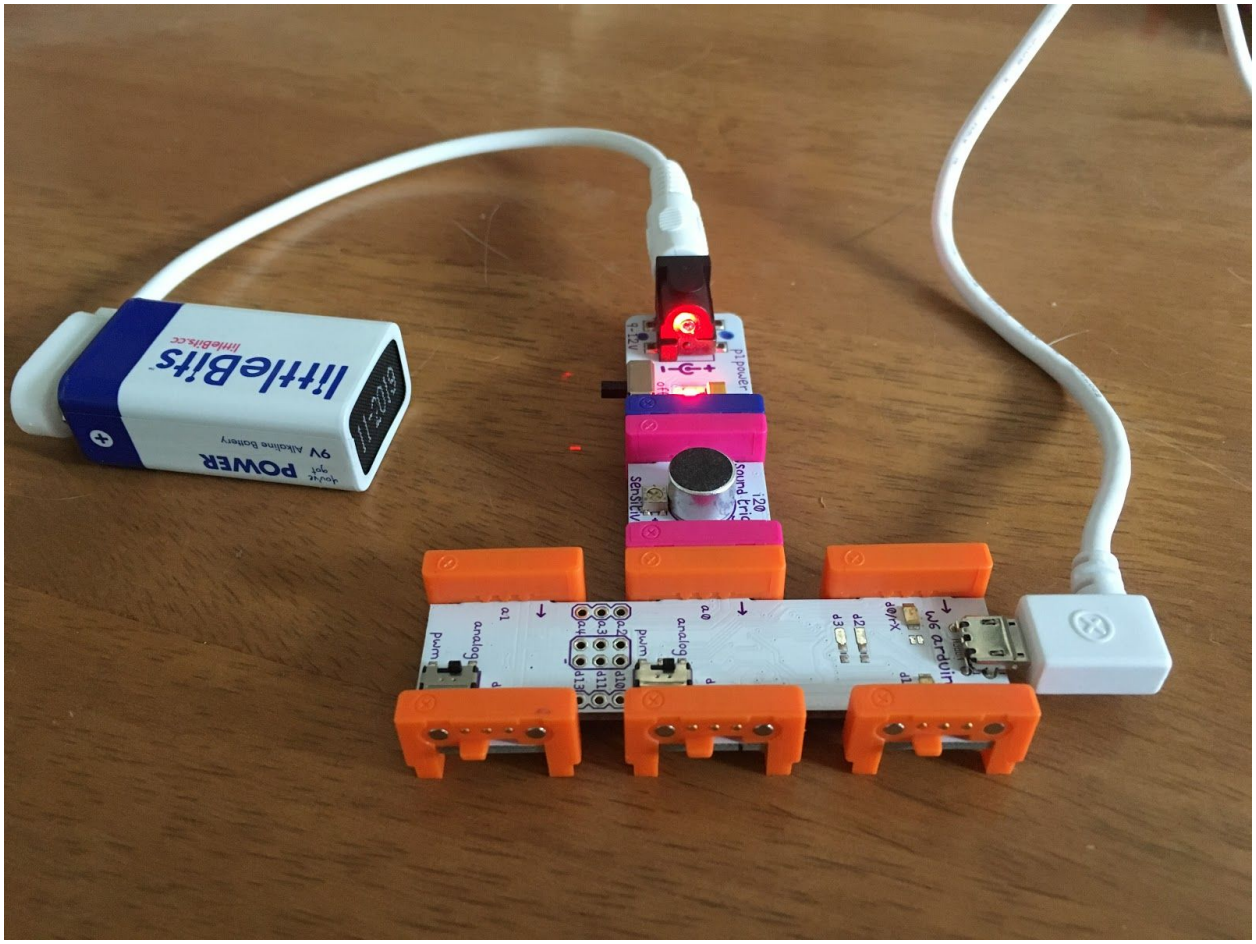


Project Case Study: Noise Alarm!

Year level band: 9-10

Description:

This project creates a visual noise detector, using littleBits combined with Arduino and Processing to build skill in the use of digital systems, Object-Oriented programming design and implementation, and data representation.

This project can be undertaken over a number of sessions, depending on the depth of discussion associated with the design stages. Students can work independently or in teams, working collaboratively on design and implementation stages. Students will learn about Object-Oriented programming using existing Classes and Objects, as well in designing and creating a single new Class, making this a good first project.

Resources:

- littleBits Arduino Kit (specifically: power, sound trigger, arduino, micro USB cable)

- Mac or PC with Arduino IDE and Processing IDE installed
- Spare 9V batteries

Prior Student Learning:

A basic understanding of circuits is useful.

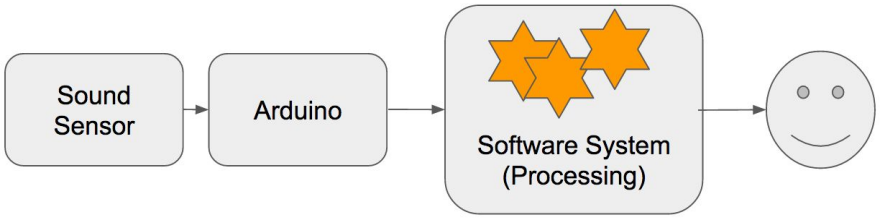
An understanding of general and Object-Oriented programming concepts.

Digital Technologies Summary	This activity introduces students to: ● Complex digital systems, including the use of littleBits circuitry, data representation and transmission in networked environments. ● Object-Oriented design and implementation, including the analysis, tracing and understanding and existing Object-Oriented software components and the design and creation of new Object-Oriented software components. ● The project introduces the students to the idea of systems thinking, focusing on how components are connected to each other and communicate using defined protocols.
Band	**Content Descriptors**
9-10	● Investigate the role of hardware and software in managing, controlling and securing the movement of and access to data in networked digital systems (ACTDIK034) ● Analyse and visualise data to create information and address complex problems, and model processes, entities and their relationships using structured data (ACTDIP037) ● Design the user experience of a digital system by evaluating alternative designs against criteria including functionality, accessibility, usability, and aesthetics (ACTDIP039) ● Design algorithms represented diagrammatically and in structured English and validate algorithms and programs through tracing and test cases (ACTDIP040) ● Implement modular programs, applying selected algorithms and data structures including using an object-oriented programming language (ACTDIP041) ● Plan and manage projects using an iterative and collaborative approach, identifying risks and considering safety and sustainability (ACTDIP044)

Achievement Standards	• Students plan and manage digital projects using an iterative approach. • They define and decompose complex problems in terms of functional and non-functional requirements. • Students design and evaluate user experiences and algorithms. • They design and implement modular programs, including an object-oriented program, using algorithms and data structures involving modular functions that reflect the relationships of real-world data and data entities. • They share and collaborate online, establishing protocols for the use, transmission and maintenance of data and projects.

Project Outline

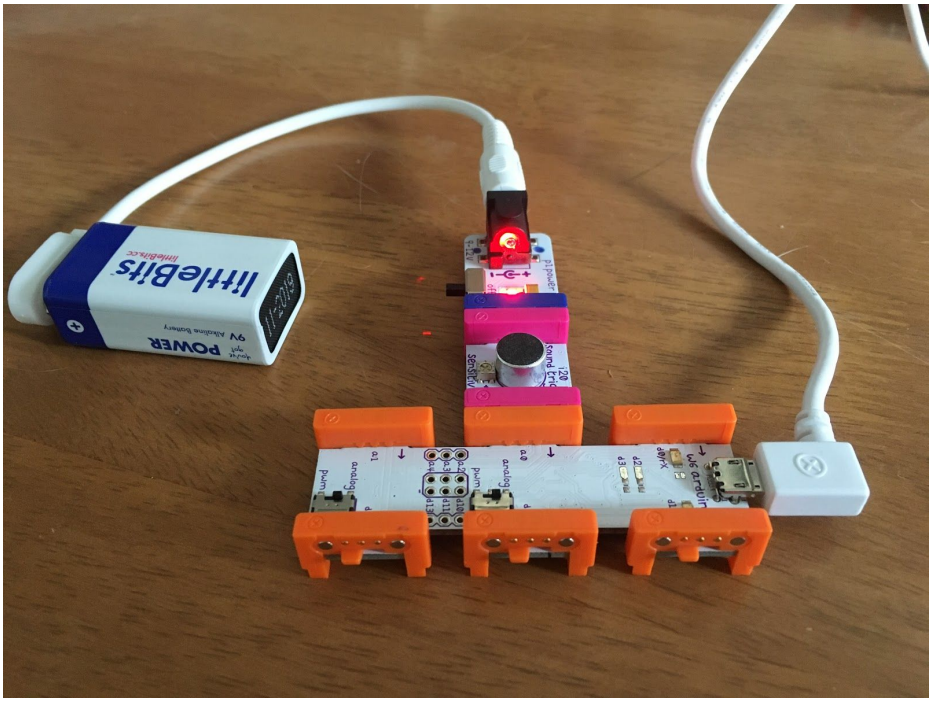
In this project, students will work through the problem solving and project design process to create a visual noise detector, using LittleBits sensors to relay noise information via an Arduino board through to our Processing-based software system that will visually indicate when noise has been detected.

Defining the Problem	To create a **visual alarm** for when noise levels are too high in a classroom. In this problem, we want to create some kind of visual symbol or alert that will go off when noise levels reach too high a level. If we decompose this problem we can see that we will need: • some way of detecting the noise, • a way for our software system to be notified when this happens (using our Arduino to connect to our sensor), • and a way for our software system to alert us when this happens (the visual alarm!).  Our system is going to involve working with both hardware and software, and so we will need to understand what we have available in hardware that can assist us. Our LittleBits set includes a Sound Trigger. This bit is an affecting bit - it affects modules connected after it, and passes on information. In this case, it passes on an ‘ON’ signal to the bit connected after it.

If we want our software system to know when this occurs, we can use our Arduino bit to interface between our Little Bits system and our software system. We can connect our Arduino bit like so, connecting it after the Sound Trigger bit, and using our Micro USB cable to connect the Arduino to our computer.

Our Arduino bit requires power to be seen by our computer, so we will need to connect a Battery and a Power bit to the start of our circuit. Now, when sound is detected, a signal can be sent from the Sound Trigger bit through to our computer!

This done by sending signals through a Serial port connected to our computer. This represents a great opportunity to explore Serial port connections, and how data can be transmitted using this method. A good class project could be to prepare a poster or presentation on how the Serial port is used in this case.



Now that understand how our hardware will work in the design of our system, let's move on to starting the design of our software system.

There are all sorts of different alarms that we could create using our software system, but when we think about this problem, a clear visual alarm will be best - an audio alarm will not work very well in a high noise environment!

We want something that will be easily seen and will continue to attract attention while the noise levels are high.

In this case, we have chosen to create quite a simple alarm that is a single colour display, that will change colour each time noise is detected that exceeds the required level. If noise continues to exceed the level, and our

sensor continues to tell us that it has been exceeded, then we will continue to change colour, creating a dynamic, colourful, visual display.

We are using hardware in our system (LittleBits), arduino as a hardware/software interface to our computer, and Processing as the software environment for building our animation using Object-Oriented programming. Each component or language has been chosen because it serves a very specific purpose, and is good at achieving that purpose.

Our Arduino bit and software system is designed specifically for interfacing in this way. The Processing programming environment is designed to easily create graphics, and so is well suited to creating visual games or animations. Why do we have such special purpose languages? This can be a good opportunity for students to discuss why we have different kinds of programming languages, and what benefits it brings to us when designing new software systems.

We know that every Processing program has a similar structure, similar in many ways to an Arduino sketch. Each Processing program will contain the following two functions:

- setup()
- draw()

The setup() function is used to set up our drawing canvas, and is called once when our program starts. The draw() function is called repeatedly, based upon the *framerate* at which our image is updated. As we want to repeatedly wait for information from our sensor, this function might be a good place to contain this behaviour.

But what about the rest of our program? We want to model an alarm in our system - this seems like a likely Class in our design. What does our Alarm have to do? It needs to draw a colour square in our canvas when a noise is detected, and it needs to change that colour each time a noise is detected.

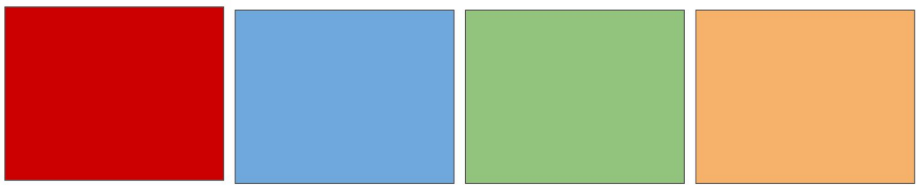
This kind of Alarm might be useful for different things - perhaps not just for when a noise is detected, but also at a specific time of day, or when a specific event occurs. So let's not include the trigger for the Alarm in our Class - it could vary depending on the use of our Alarm Objects, and is an external decision.

This gives us a very high level design of one Class, that contains functional behaviour to draw a square whose colour changes each time it is drawn.

Class: Alarm	**Variables**	**Functions**
	?	Display

How will the Alarm Class know how large a square to draw? We will need to include data that tells how how large our canvas is as well.

<table>
<tr>
<td></td>
<td>
<table>
<tr><td rowspan="2">Class:
Alarm</td><td>Variables</td><td>Functions</td></tr>
<tr><td>Canvas Size</td><td>Display</td></tr>
</table>


</td>
</tr>
<tr>
<td>Understanding our Data</td>
<td>
We have to deal with data of many different types in this problem, including:

Signals from our hardware noise sensors
Data needed to define the colour and size of our alarm.

Our LittleBits Sound Trigger bit passes an ‘ON’ signal as its output. Our Arduino bit is able to receive this signal, and can pass on another signal to our Processing software system. We could represent this data in many ways, but we have chosen to use binary representation, where a 1 is used to indicate an ‘ON’ signal to our Processing software system.
Our Processing programming environment is designed for creation of images of animations, using a fixed size canvas, whose size can be defined in the setup() functions described above.

The data that our Alarm class needs to create its visual alarm is the:

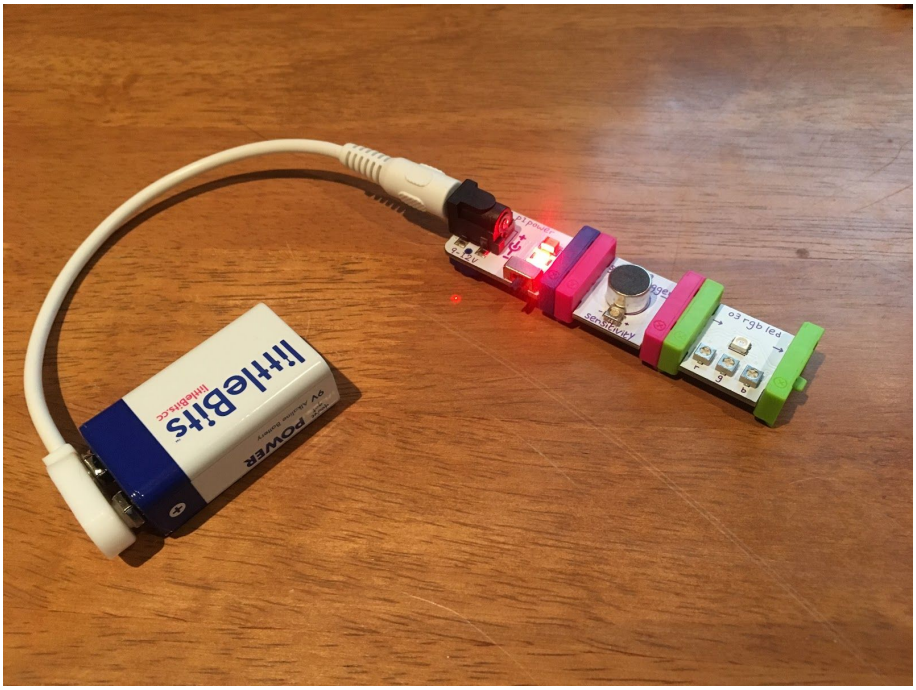
size of the canvas, to direct how large a square to draw, and
the colour to fill the square with.

The size of the canvas is represented in pixels, and can be represented using one variable (of type integer) as we are drawing a square canvas. This variable can provided to an Alarm Object when it is constructed, by passing it as a parameter in its constructor. This identifies a key information relationship between our setup() function which will create our Alarm Object and the construction of the Alarm Object.

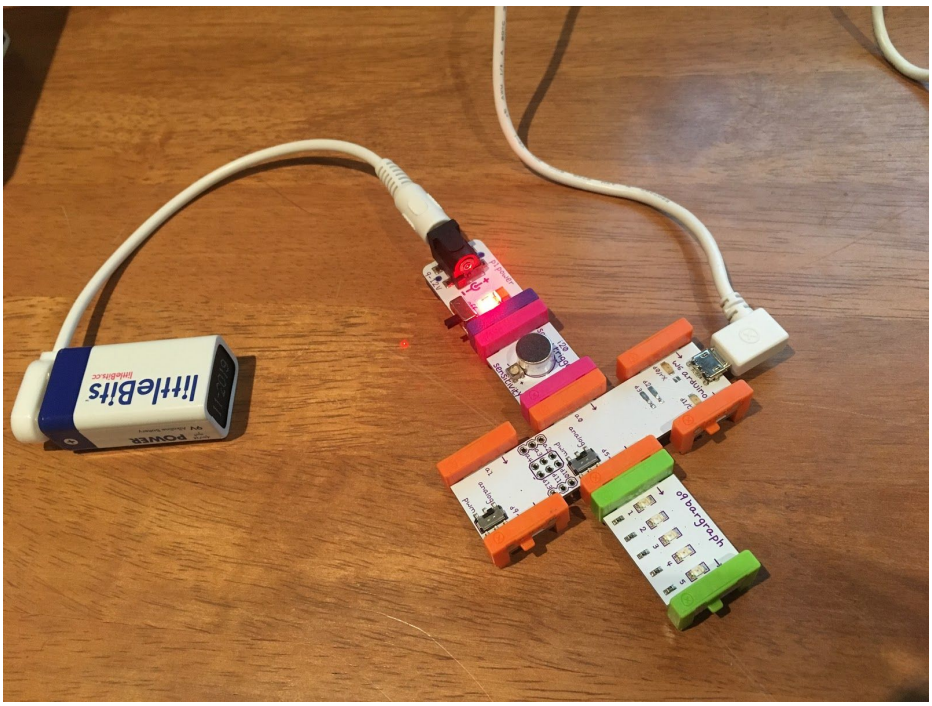
The colour that we want to fill the square with should change each time we draw the square (for each time that our Sound Trigger bit is activated).

A good way to continuously change our colour, is to use a random number generator to select our colour values. Colour is represented in Processing using RGB, and so we can use a random number generator to identify a random number selection between 0 and 255 for each of the Red, Green and Blue components.

Why between 0 and 255? That is because we are using an 8-bit integer representation for each colour component. This represents a valuable opportunity to discuss data representation, and to explore different ways that this data could be represented. There is a great resource, using Processing, that explains RGB and how colour is represented provided by the
</td>
</tr>
</table>

	Processing.org community: https://processing.org/tutorials/color/
Prototyping and User Interface Design	There are multiple opportunities for prototyping and experimenting with user interface design in this project. Students can work together to design their own visual alarm. While we have chosen to use a randomly changing colour square as our alarm, there are many different ways that the alarm could be represented, which students could prepare prototypes for using paper prototyping. We can work through a paper prototype for our own alarm - this would be very simple, perhaps exploring sequences of colour that would make a good alarm! A simple approach could be used here - maybe using 'postit' notes or paint swatches! To become comfortable with the hardware being used in this project, we can design a simpler hardware-based alarm to build a quick prototype of this part in the project. We can use a combination of our Sound Trigger bit and a Buzzer but or an LED-based bit to show how LittleBits work, and to demonstrate and explore signal progression. 
Implementation	We can use our stepwise development approach here to structure our implementation, first, looking at connecting our Sound Trigger bit as a sensor to our Arduino system, and then secondly, getting our Arduino system to pass on our Sound Trigger signal to our Processing software system, and finally, to implement our Alarm Class.

	We will talk through some ideas for overall testing at the end of this case study, however we will also talk about testing each stage as we complete it.
	
	Stage 1: Connecting our Arduino
	
	We can connect our Arduino bit to our Sound Trigger bit as a way of capturing our 'ON' signal. If you have not done this before, there is a great tutorial provided by LittleBits: http://littlebits.cc/tips-tricks/arduino-setup-tutorial
	
	We then want to implement an Arduino software component using an Arduino sketch that is able to receive the 'ON' signal from our Sound Trigger bit as our first requirement. Our first sketch (below) has two parts:
	● the setup() function which initialises the soundTriggerPin connected at A0 on our Arduino bit as an input source and starts a Serial connection to our Arduino bit, and
	● the loop() function, which attempts to read from our Sound Trigger bit.
	
	`const int soundTriggerPin = A0;`
	`int soundTriggerState = 0;`
	
	`void setup() {`
	`  pinMode(soundTriggerPin, INPUT);`
	`  Serial.begin(9600);`
	`}`
	
	`void loop() {`
	`  soundTriggerState = digitalRead(soundTriggerPin);`
	`  delay(10);`
	`}`
	
	But how do we know if we have received a signal and this is working correctly? We will need to test this - a great way to do this without needing to add more complicated code, is to connect our Arduino bit to an output but, such as a LED bit, and get it to activate that LED bit if it receives a signal from our Sound Trigger bit.



At the moment, we are reading our signal every 10 milliseconds - if we have constant noise for a period of time, then we will receive a sequence of ‘ON’ signals from our Sound Trigger. To test this with our LED bit, let’s change the logic somewhat so that we turn our LED on when a sound is detected the first time, or in other words when the signal we have received has changed from the last signal. Once the noise goes away, our LED should turn back off again.

We can do this like so, adding two variables that will help us keep track of whether the signal we have received has changed or not. New code is shown in **bold**, to make it easier to identify.

```
const int soundTriggerPin = A0;
const int ledPin = 5;
int soundTriggerState = 0;
int counter = 0;
int lastState = 0;

void setup() {
  pinMode(soundTriggerPin, INPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  digitalWrite(ledPin, LOW);
}

void loop() {
   soundTriggerState = digitalRead(soundTriggerPin);
   delay(10);
   if (soundTriggerState != lastState) {
      counter++;
```

```
      if (counter % 2 == 0) {
        digitalWrite(ledPin, LOW);
      }
      else {
        digitalWrite(ledPin, HIGH);
      }
    }
    lastState = soundTriggerState;
}
```

We have modified our algorithm so that we are now identifying when a signal has changed, either from on to off, or from off to on.

This is a good test that our passing of information through our Arduino bit is working well, and is also a good opportunity to discuss digital and analog signals, and how signals are passed between these components.

Stage 2: Passing our Signal on

Now that we have tested our Arduino code and our LittleBits system, we can return to our next stage of passing on the signal from our Arduino software component to our Processing software component. To do this, we take our original Arduino sketch, and we add code to pass on our signal again via the Serial port to our Processing system. Here, we check whether the Sound Trigger is detecting noise, and if it is, we send on our signal.

In our approach we will continue to send a signal while the Sound Trigger bit is detecting noise. This is a design choice, as we can continue to trigger our alarm while the noise levels are high. Depending on your design of your alarm, you may want to do this as well, or you may want to only pass on the signal when the noise has changed, as we implemented in our testing example above.

```
const int soundTriggerPin = A0;
int soundTriggerState = 0;

void setup() {
    pinMode(soundTriggerPin, INPUT);
    Serial.begin(9600);
     while(!Serial);
}

void loop() {
  soundTriggerState = digitalRead(soundTriggerPin);
  delay(10);
  if (soundTriggerState == 1) {
     Serial.println(soundTriggerState);
  }
}
```

Stage 3: Building our Alarm Class

Recall our Alarm class:

Class: Alarm	Variables	Functions
	Canvas Size	Display

In this class, we are supporting a single functional behaviour, which is drawing a randomly selected colour square over our canvas each time our Display() function is called.

Here is an outline of our Alarm Class in Processing, with a variable to contain the canvas size, a constructor and a display() function. At the moment, neither of our functions does anything:

```
class Alarm {

 int canvasSize;

 Alarm (int _canvasSize) {
 }

 void display() {
 }
}
```

Our constructor function is responsible for setting up everything a new Alarm Object needs when it is created. In this case, we should store the value passed to the constructor in the parameter _canvasSize in its Object variable, canvasSize. We can also do other useful things here, such as setting visual attribute preferences, if we wanted to.

Our display() function should draw a square the size of our canvas, with a new fill colour each time. We can do this by setting the fill colour using the inbuilt random() function to generate RGB values, each within the range 0 and 255.

Here is what our code looks like now:

```
class Alarm {

 int canvasSize;

 Alarm (int _canvasSize) {
    canvasSize = _canvasSize;
 }

 void display() {
   fill(random(255),random(255),random(255));
   rect(0,0,canvasSize,canvasSize);
 }
}
```

Stage 4: Putting it all together!

	Now we can work on building the connection between our Arduino and our Processing code from the Processing end - this is the final step. To connect our Processing system to our Arduino system, we need to connect our Processing code to the Serial port.
	
	We start in our setup() function by creating our canvas, and creating a new Serial Object using an existing Processing class that we have imported from the processing.serial library. We then create a new Alarm Object, passing the canvas size as a parameter.
	
	After this, in our draw() function, we continue to read values from our Serial port, while it is available to read from. If we have a value, we call our display() function in our Alarm Object and then reset our buffer!
	
	`import processing.serial.*;`
	
	`Serial myPort;`
	`String buff = "";`
	`int NEWLINE = 10;`
	`Alarm myAlarm;`
	
	`void setup()`
	`{`
	`  size(512, 512);`
	`  myPort = new Serial(this, Serial.list()[2], 9600);`
	`  myPort.bufferUntil('\n');`
	`  background(87, 36, 124);`
	`  myAlarm = new Alarm(512);`
	`}`
	
	`void draw() {`
	`  while (myPort.available () > 0) {`
	`    int value = myPort.read();`
	`    if (value != NEWLINE) {`
	`      buff += char(value);`
	`    }`
	`    else {`
	`      myAlarm.display();`
	`      buff = "";`
	`    }`
	`  }`
	`}`
	
	So that is all of our implementation complete!
	
	This is an interesting project, as it helps explore many aspects of Digital Technologies, including digital systems, data representation, algorithm design and Object-Oriented programming.

Testing and Evaluation	We have tested our implementation throughout our implementation stage, however now that it is complete, we can test it all together!

	The easiest way for us to test this is by undertaking a series of test cases with different levels and periods of noise, waiting to see what happens in our Alarm. We should be able to see a progression of random colour squares appear each time there is considerable noise. What if we don't? Or what if we our Alarm goes off all of the time? There are a couple of things we can do here to test and debug our program. One thing to check is the sensitivity of our hardware. Our Sound Trigger bit has an adjustable sensitivity, which can be adjusted using a little screw driver. This is the first thing you might want to try to see if there is a change in behaviour. The other thing you can do, is to trace through your code by hand, to see if it makes sense and works as you expect. A core review might be useful here as well, as working as a team, it is often easier to locate problems in implementation. You might also want to consider including trace statements at different stages in your code, printing out values of variables or perhaps identifying when branches in your if statements have been selected.
Project Extensions	There are many ways that you can open this project up for extensions. A couple that we have thought of include: ● Opening up the design for the Alarm Class. ● Recording the noise level data so that a graph of noise levels can be produced, perhaps including statistical analysis of common periods of noise in your classroom. Of course you might also want to see what you can do with different input sensors, and try to create a whole different project using the Sound Trigger, perhaps a visual door bell for someone who is unable to hear very well!

Project Reflection

In this project, students will undertake an extensive design and implementation process, combining elements of hardware and software. The project is well suited to be taught over a number of sessions, and using group work. While we have not included explicit assessment information here, we have provided a discussion of how these sessions might be structured, and how students might be able to demonstrate their learning.

For a more extensive discussion of assessment possibilities within Digital Technologies at this level, please join us in our online community and course: www.csermoocs.adelaide.edu.au

Learning construction	Students work independently or in teams to construct the LittleBits circuit, and designing and implementing the Arduino and Processing components. This project is well suited to teamwork, and can be used to build collaboration and communication skills.

	The activity is about experimenting, trying new solutions, and debugging. Once students have successfully built the circuit, ask them to explore what other systems could be designed and connected. The project can be extended in multiple ways through different Class designs, or the use of different sensors. Encourage students to help each other - and look for help on the internet. Ask a friend. Ask Google. Then ask the teacher.
Learning demo	While students are working in groups, ask questions to give them the opportunity to demonstrate their thinking and understanding: What challenges have you faced in building this circuit? What other bits could you add to your circuit and how would you use them? What can they tell you about how the Arduino system works to connect the LittleBits hardware and the Processing components? Why do we have these different parts in our software system? Why do we use different programming languages for the different parts? Students can undertake a code review of their implementation, using this as an opportunity to reflect on how their design works. Can they think of different ways to design or implement their system?
Learning reflection	Remind students that LittleBits is a prototyping platform. Now that they have successfully designed and built a system that combines both hardware and software using Arduino and Processing, perhaps share the project extension ideas we have described above, and ask the students to think about what alternative systems they could design and create. Students can be asked to reflect upon their design decisions, and any changes that they had to make to their design as their project progressed. Why did they have to make changes? Was it because of changes to their understanding of the requirements? Or the way that the hardware or software operated? Students can be asked to reflect on whether their final outcome matched their expectations. Were they able to produce the software system that they wanted to? Why or why not? Would they change anything about their design now that it is complete?

Teacher/Student Instructions:

If using the mounting board:

- Snap your circuit together before pressing into the mounting board
- Press down Bitsnaps (the coloured edges) rather than the white circuit board

CSER Professional Learning:

This Case Study supports learning from the CSER 9&10 Digital Technologies: Explore! MOOC. You can join us here to learn more about Digital Technologies in Years 9 & 10: www.csermoocs.adelaide.edu.au

Further Resources:

1. Information about the littleBits system and some example projects: http://littlebits.cc/projects
2. Information on Getting Arduino started with Little Bits: http://littlebits.cc/tips-tricks/arduino-setup-tutorial
3. Examples on using Processing with Arduino and Little Bits: http://littlebits.cc/projects/diy-etch-a-sketch