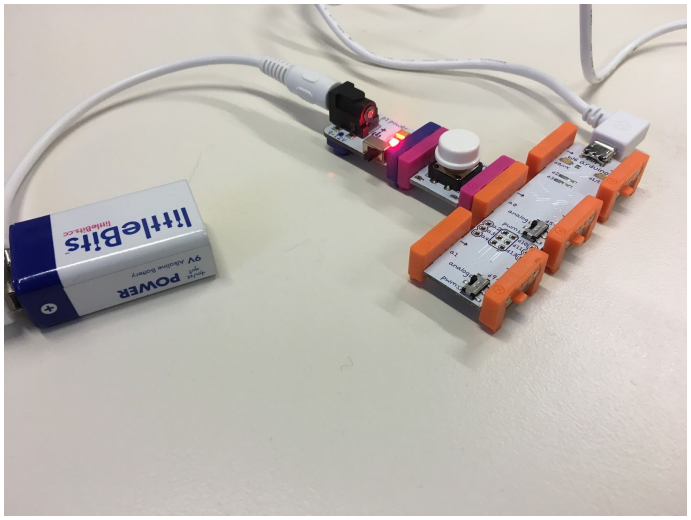


Project Case Study: Teddy Dunk!

Year level band: 9-10

Description:

This project creates a remix of a ‘dunk tank’ game, 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 multiple new Classes, making this a good intermediate project.

Resources:

- littleBits Arduino Kit (specifically: power, button, arduino, micro USB cable)
- Mac or PC with Arduino IDE and Processing IDE installed
- Spare 9V batteries

Prior Student Learning:

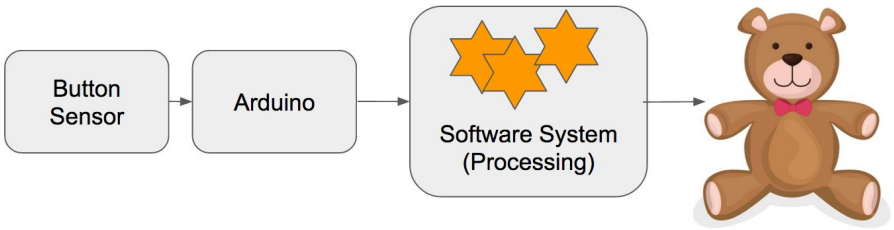
A basic understanding of circuits is useful.

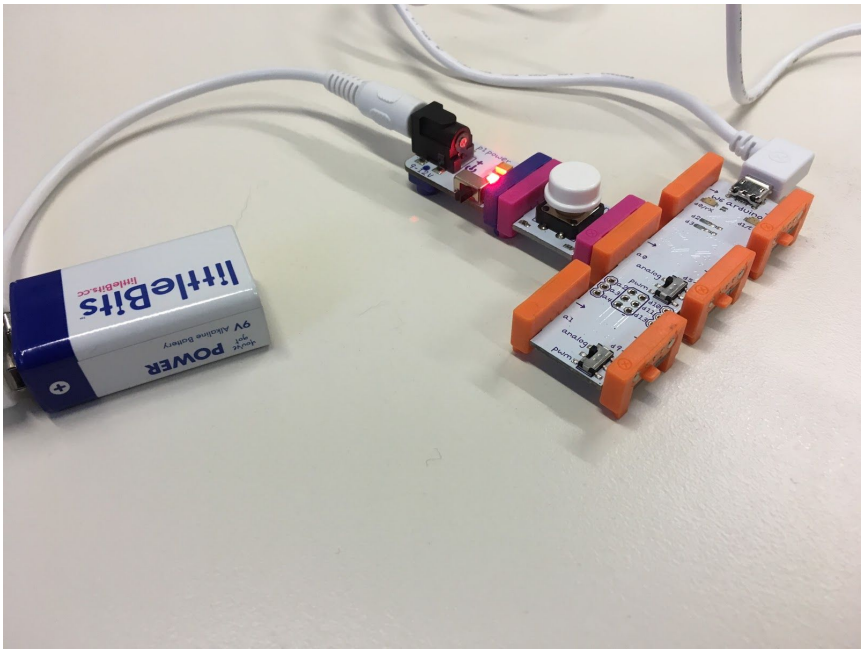
An understanding of general and Object-Oriented programming concepts.

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 Processing game, using LittleBits sensors to interact with the game via an Arduino board.

Defining the Problem	To create a **virtual dunk tank game** where a teddy bear character is dropped into a tank when a physical button is hit by a ball. In this problem, we are creating a game that blends virtual and physical interactivity. Our game will be a virtual dunk tank, where a character will drop into a container, or tank, based on a physical button being pressed. If we decompose this problem we can see that we will need: ● some way of detecting whether a button has been pressed, ● a way for our software system to be notified when this happens (using our Arduino to connect to our sensor), ● and the game that dunks teddy when this happens!  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 Button bit. 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, when the button is pressed, and then the ‘OFF’ signal when the button is released. Each time a button is pressed, two signals are sent. 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 Button 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 our Button is pressed, a signal can be sent from the Button 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 games that we could create that can be operated by a button - perhaps a quiz game, or one where a button is used to control the activity of a character, such as jumping over obstacles, or other movement. We have decided to create a simply game, where if the button is pressed, our Teddy Bear character falls off the screen of our display. What makes this a little more challenging is that the button needs to be pressed by being hit by a ball!

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 game 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 a Teddy Bear in our system - this seems like a likely Class in our design. What does our Teddy Bear have to do? It needs to draw itself in the middle of our canvas initially, waiting for the Button to be pressed. Once the button is pressed, we need to draw our Teddy Bear falling off the screen.

This gives us a very high level design of one Class, that contains functional behaviour to draw our Teddy Bear perched on his seat, to let it know to start falling, and to continue to fall down the screen.

	Variables	**Functions**
Class: Teddy		Seated StartFalling Falling

How will the Teddy Object know where it is to be drawn? We will need to add Height and Width values for the size of our canvas, so that we can work out how to place our teddy in the centre of our canvas.

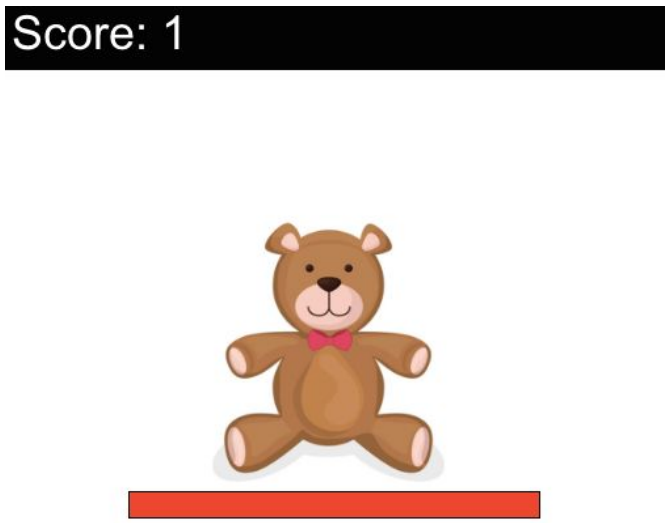
	Variables	**Functions**
Class: Teddy	Height, Width	Seated StartFalling Falling

If we are playing this as a game, then we will want to know how many times our Teddy Bear has been dunked! Let's add a simple scoreboard where we can count the number of times since the start of the game, that our Teddy has fallen.

If we were to do this, then we might want to add a Scoreboard Class, which is able to keep track of the number of times that Falling has been called, and can display the current score. Our Scoreboard might also need to know the size of our canvas, so that it can draw itself appropriately. In this case, we will draw our Scoreboard starting in the left hand corner, so we only need to know how wide our canvas is.

	<table><tr><td rowspan="2">Class: Scoreboard</td><td>Variables</td><td>Functions</td></tr><tr><td>Count, Width</td><td>Display Increment</td></tr></table>

Understanding ourData	We have to deal with data of many different types in this problem, including:
	● Signals from our hardware Button sensor.
	● Data needed to define where we draw our Teddy, both at the start of the game, and as it is falling.
	● Data needed to define where we draw our Scoreboard.
	● The count of the number of times our Teddy has been dropped.
	
	Our LittleBits Button bit passes a ‘ON’ and ‘OFF’ signals as its output: ‘ON’ when the Button is pressed, and ‘OFF’ when it is released. Our Arduino bit is able to receive this signal, and can pass on another signal to our Processing software system.
	
	We only need to pass on the fact that our Button has been pressed to the Processing game, so we have chosen to pass on the value of 1 when our Button has been pressed.
	
	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 in pixels. We will need to know the location of our image file that represents our Image, as well a data associated with its location. To identify where we need to place our Teddy Bear image on the canvas, we need to know the width and height of the canvas, as well as the width and height of the image, teddyImg.
	
	To place our image in the centre of our canvas, we can work out our intended coordinates as:
	`xCoordinate = (canvasWidth-teddyImg.width)/2;`
	`yCoordinate = (canvasHeight-teddyImg.height)/2;`
	
	We will need to be able to tell whether our image should be shown in the middle of the screen, stationary, or whether it should be drawn falling down the screen. We will need a variable that records this value.
	
	As our image falls down the screen, we need to redraw it at different coordinates. To slowly move the image down, we can redraw the image 20 pixels further down in each frame.
	
	For our Scoreboard, we will place a rectangle at the top of our canvas like so, to display our Score:

<table>
<tr>
<td></td>
<td>


We will need to keep track of the score itself, as well as the x and y coordinates, and width for the scoreboard.

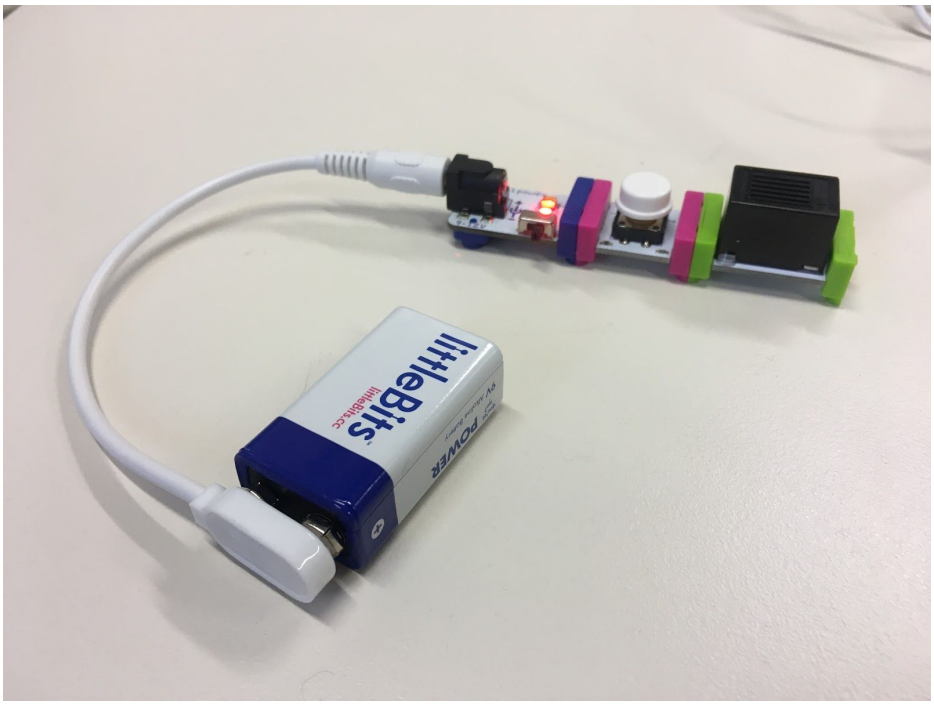
Redefining our Classes, based on this additional data we have:

<table>
<tr><td rowspan="2">Class:
Teddy</td><td>Variables</td><td>Functions</td></tr>
<tr><td>width (integer)
height (integer)
xCoordinate (integer)
yCoordinate (integer)
teddyImg (PImage)
fall (boolean)</td><td>Seated
StartFalling
Falling</td></tr>
</table>

<table>
<tr><td rowspan="2">Class:
Scoreboard</td><td>Variables</td><td>Functions</td></tr>
<tr><td>Count (Integer)
Width (Integer)</td><td>Display
Increment</td></tr>
</table>

There are several different data types used here - this could be a good opportunity to discuss why we have so many different data types, why that is helpful, and why we have made the decisions that we have to use these ones in particular. Would other data types be useful here?
</td>
</tr>
<tr>
<td>Prototyping and User Interface Design</td>
<td>There are multiple opportunities for prototyping and experimenting with user interface design in this project.

Students can work together to design their own game using a Button as input. While we have chosen to use a dunk tank game, many different games could be represented using the same sensor input, which students could prepare</td>
</tr>
</table>

	prototypes for using paper prototyping. We can work through a paper prototype for our own game - showing the starting point with our Teddy Bear image shown in the middle of our screen and our scoreboard, and then sequences of images showing the image falling down the screen, and the updating of our scoreboard. In particular, it could be useful here to work through the logic here of how we work with the X,Y coordinate system used in Processing. For example, working on paper, how would we work out the starting X,Y coordinates to draw our Teddy so that it is shown in the middle of the canvas? What would we need to change as we move the image down? How many pixels do you think you should move it as the canvas is redrawn? What about our scoreboard? Where should it be placed? What should happen to the score each time our Teddy falls? To become comfortable with the hardware being used in this project, we can design a simpler hardware-based system to build a quick prototype of this part in the project. We can use a combination of our Button 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 Button bit as a sensor to our Arduino system, and then secondly, getting our Arduino system to pass on our Button signal to our Processing software system, and finally, to implement our Teddy and Scoreboard Classes.

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 Button bit as a way of capturing our 'ON' and 'OFF' signals. If you have not done this before, there is a great tutorial provided by LittleBits: http://littlebits.cc/tips-tricks/arduino-setup-tutorial

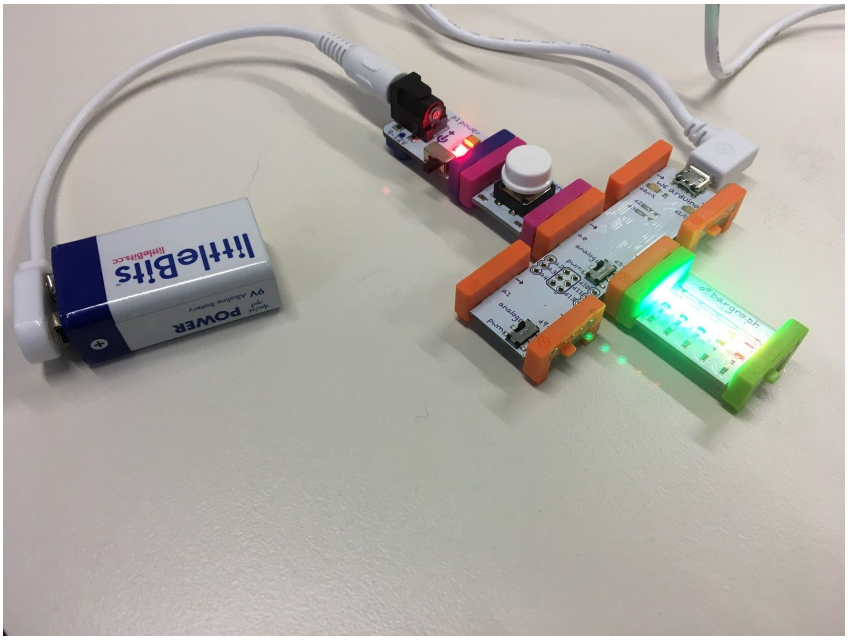
We then want to implement an Arduino software component using an Arduino sketch that is able to receive the sequence of 'ON' and 'OFF' signals from our Button bit as our first requirement. Our first sketch (below) has two parts:

- the setup() function which initialises the buttonPin 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 Button bit.

```
const int buttonPin = A0;
int counter = 0;
int buttonState = 0;
int lastButtonState = 0;

void setup() {
  pinMode(buttonPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  buttonState = digitalRead(buttonPin);
  delay(10);
  if (buttonState != lastButtonState) {
     counter++;
     if (counter %2 == 0) {
          // have pressed and released our button
     }
   }
   lastButtonState = buttonState;
}
```

Because we are receiving both 'ON' and 'OFF' signals, our logic is a little more complicated. This would be a good opportunity to work through the logic of this code using a flowchart.

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 Button bit.



In this code, we are going to alternate between turning our light on and off, each time our Button is pushed. As each Button push has two signals, this means that we will only turn our light on every four signals. Again, this might be worth working through as a flowchart.

The code we have added is shown in **bold**. We have added code to register our output connection, initially set out light to on, and also code that tests for every fourth signal, then turning on our light!

```
const int buttonPin = A0;
const int ledPin = 5;
int counter = 0;
int buttonState = 0;
int lastButtonState = 0;

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

void loop() {
  buttonState = digitalRead(buttonPin);
  delay(10);
  if (buttonState != lastButtonState) {
     counter++;
     if (counter %2 == 0) {
        // have pressed and released our button
     }
```

```
    if (counter % 4 == 0) {
        digitalWrite(ledPin, HIGH);
    }
    else {
        digitalWrite(ledPin, LOW);
    }
  }
  lastButtonState = buttonState;
}
```

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 Button has been pushed, and if it has, we send on our signal.

```
const int buttonPin = A0;
int counter = 0;
int buttonState = 0;
int lastButtonState = 0;

void setup() {
  pinMode(buttonPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  buttonState = digitalRead(buttonPin);
  delay(10);
  if (buttonState != lastButtonState) {
    counter++;
    if (counter %2 == 0) {
        // have pressed and released our button
        Serial.println(counter);
    }
  }
  lastButtonState = buttonState;
}
```

Stage 3: Building our Scoreboard Class

Recall our Scoreboard Class:

Class: Scoreboard	Variables	Functions
	Count (Integer) Width (Integer)	Display Increment

The following Processing code shows an outline of our Scoreboard Class, with the variables and functions we have defined:

```
class Scoreboard {
  int count;
  int canvasWidth;

  Scoreboard (int _width) {
  }

  void increment() {
  }

  void display() {
  }
}
```

Our constructor is responsible for setting up any Scoreboard Objects that are created, and here should simply set our count to zero, and record the intended width of the scoreboard. Our increment() function is responsible for increasing our score, and our display() function is responsible for displaying an updated scoreboard. Here is what our code looks like now with this behaviour added:

```
class Scoreboard {
  int count;
  int canvasWidth;

  Scoreboard (int _width) {
    count = 0;
    canvasWidth = _width;
  }

  void increment() {
    count++;
  }

  void display() {
    fill(0,0,0);
    rect(0,0,canvasWidth,50);
    fill(255);
    textFont(f,36);
    text("Score: " + count,10,35);
  }
}
```

Our scoreboard display() function uses several Processing functions to draw itself. This is an opportunity to discuss programming APIs, and how students can discover what existing functions a language provides, perhaps reviewing the Processing.org site, and reviewing ways that students can problem solve. What would make a function likely to already exist? Is it a commonly used task? Does it fit the context that the language has been designed for?

Stage 4: Building our Teddy Class

Recall our Teddy Class:

Class: Teddy	Variables	Functions
	width (integer) height (integer) xCoordinate (integer) yCoordinate (integer) teddyImg (PImage) fall (boolean)	Seated StartFalling Falling

In this class, we are supporting a relatively simple functionality: we need to display an image of our Teddy Bear in the middle of the screen at the start, and then once we have been told that we should start our Teddy falling, i.e. when we have received a Button push, we then progressively move our Teddy down the canvas until he falls off the edge.

Remember that Processing has two built-in functions, similarly to Arduino: setup() and draw(). Our draw() function is called each time that our canvas is redrawn, and so, when our Teddy is falling, we can use this by having our draw() function call the Falling() function in our Teddy Object. Each time the canvas is redrawn, our Teddy will fall further down the screen.

Here is an outline of our Teddy Class in Processing, with variables to represent the canvas size, the location of our image, the filename where we will load our image from, and a boolean value that describes whether we are currently falling or not.

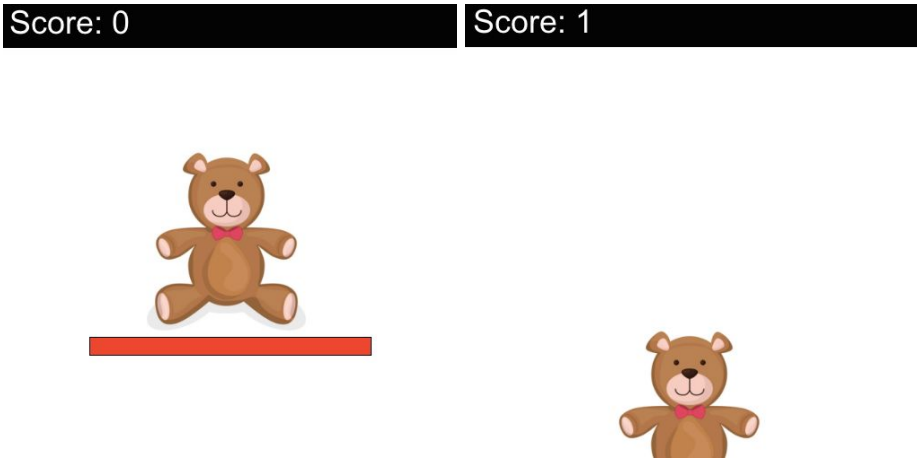
```
class Teddy {
 int canvasWidth;
 int canvasHeight;
 int xCoordinate;
 int yCoordinate;
 PImage teddyImg;
 boolean fall;

 Teddy (int _width, int _height) {
 }

 void seated() {
 }

 void startFalling() {
```

```
 }

 boolean falling() {
 }
}
```

Our constructor function is responsible for setting up everything that a new Teddy Object needs when it is created. In this case, we should store the values passed to us as parameters, and load our Teddy image from its file.

Our seated() function is responsible for drawing our Teddy in his stationary position. Our falling() and startFalling() functions are closely related. startFalling() is called by our draw() function when we have received a signal from our Button, via our Arduino system. This function simply records that our Teddy is now falling by setting our boolean fall variable to true.

Our falling() function then is responsible for actually making our Teddy fall, moving the Teddy image further down the canvas each time it is called, like so:



Here is what our code looks like now. Work through each function, tracing the code as a class, perhaps adopting a code review style to ensure that everyone understands what is happening in each line. Revisit the questions from our prototyping stage to reflect on our choices regarding the location of our Teddy image.

```
class Teddy {
 int canvasWidth;
 int canvasHeight;
 int xCoordinate;
 int yCoordinate;
 PImage teddyImg;
 boolean fall;

 Teddy (int _width, int _height) {
  canvasWidth = _width;
  canvasHeight = _height;
```

```
  teddyImg = loadImage("/Users/katrina/Desktop/TeddyImage.png");
  fall = false;
 }

 void seated() {
  xCoordinate = (canvasWidth-teddyImg.width)/2;
  yCoordinate = (canvasHeight-teddyImg.height)/2;
  image(teddyImg,xCoordinate,yCoordinate);
  fill(255,0,0);

rect(xCoordinate-50,yCoordinate+teddyImg.height,xCoordinate+teddyImg.
width-50,20);
 }

 void startFalling() {
  fall = true;
 }

 boolean falling() {
  if (fall) {
   if (yCoordinate < canvasHeight) {
    yCoordinate = yCoordinate+5;
    image(teddyImg,xCoordinate,yCoordinate);
    return true;
   }
   else fall = false;
  }
  return false;
 }
}
```

Stage 5: 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 Teddy Object and a new Scoreboard Object, passing the necessary parameters.

After this, in our draw() function, we continue to read values from our Serial port, while it is available to read from. Our draw() function has two key parts. The first is to receive the signal from our Arduino system, and if we have received a signal, we set a variable startFalling to be true.

In our second part, if startFalling is true, then we work with our Objects to update our game - incrementing our score, telling our Teddy to start falling, clearing our background to remove our seat, and displaying our updated scoreboard. Now that we've told our Teddy to start falling, we can then set startFalling back to false.

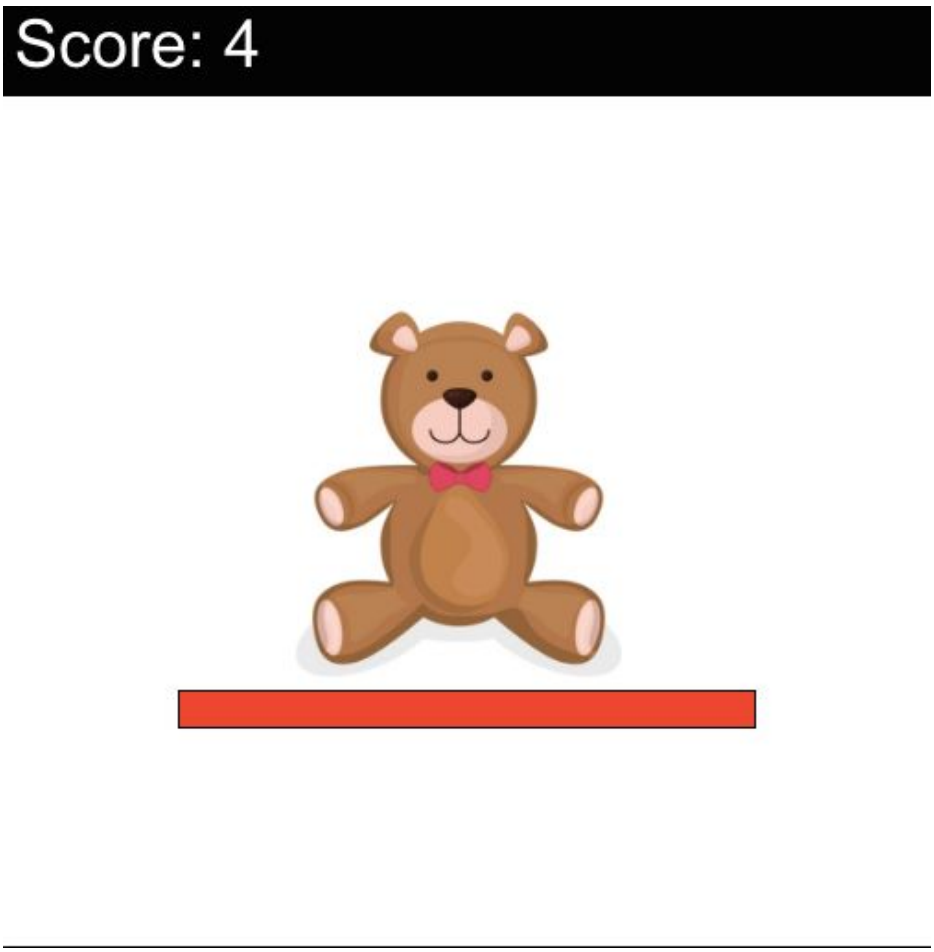
Then, while our Teddy is still falling, we continue to use our falling() function in our Teddy Object to move our Teddy down our canvas. However, if our Teddy is no longer falling, we change back to showing our Teddy seated, and ready to wait for the Button to be pushed!

The logic here is a little complex, and it can be an appropriate point to work through the order of function calls that we have described, working through a specific use case example of exactly what happens when that Button is pushed, and in what order. Do things happen as you expect?

```
import processing.serial.*;

Serial myPort;
String buff = "";
int NEWLINE = 10;
boolean startFalling = false;
PFont f;
Teddy t;
Scoreboard s;

void setup()
{
 size(512, 512);
 myPort = new Serial(this, Serial.list()[2], 9600);
 myPort.bufferUntil('\n');
 background(255, 255, 255);
 t = new Teddy(512,512);
 s = new Scoreboard(512);
 t.seated();
 f = createFont("Arial",16,true);
 s.display();
}

void draw() {
   while (myPort.available () > 0) {
     int value = myPort.read();
     if (value != NEWLINE) {
       buff += char(value);
       startFalling = true;
     }
     else {
       print(buff);
       buff = "";
     }
 }
 if (startFalling) {
    s.increment();
    t.startFalling();
    clear();
    background(255, 255, 255);
    s.display();
    startFalling = false;
```

	}
	if (t.falling()) {}
	else t.seated();
	}
	
	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. There are a few challenging stages in this project involving opportunities for students to build their algorithm design and debugging skills, as well as exploring relationships between Objects in an Object-Oriented software system.

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 where we push our button at different rates and times, waiting to see what happens in our game, and observing our Scoreboard updating and our Teddy falling. What if we don't? Or what if we our Scoreboard shows the wrong value? There are a few things we can do here to test and debug our program. One thing to check our hardware. In our earlier stages of implementation, we had tested our system to ensure that the hardware was working as expected, and to test the integration of our Arduino bit. This is the first thing you might want to test again 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: ● Designing a multi-player game, combining multiple hardware sensors and redesigning the game and the scoreboard. ● Designing a new game that incorporates a Button as a the key interactive point. What about combining both virtual and physical Buttons? ● Design a Quiz game instead of a Dunk Tank game. ● Explore the use of different input sensors, and how they might be used in a game.

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