Intro to Processing / Tweaking Simon

SparkFun Electronics Summer Semester

About Processing

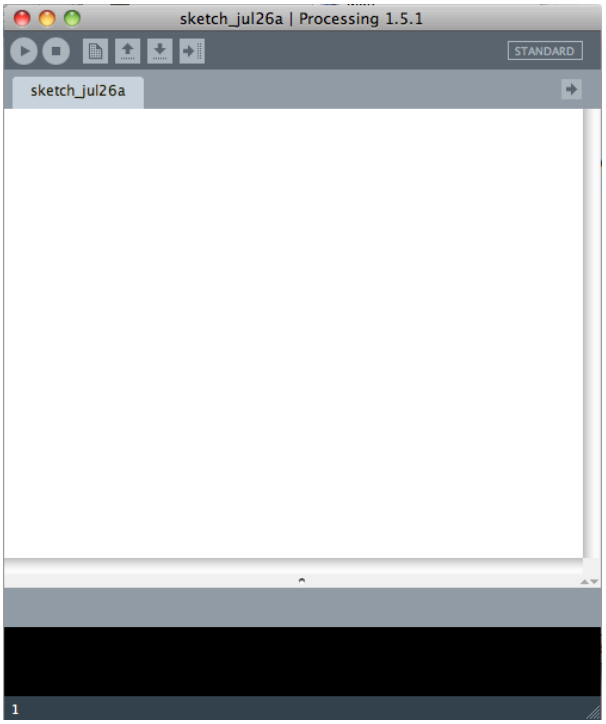
Processing is a free, open source, cross-platform programming language and environment for people who want to create images, animations, and interactions. It's easy to get started, and more importantly, it can serve as an easy way to add a visual component or interaction with your hardware project (visualizing sensor data, playing a game, displaying video, etc.).

Setup

• Download and install the latest version of Processing from http://processing.org (that's it!)

Getting Started

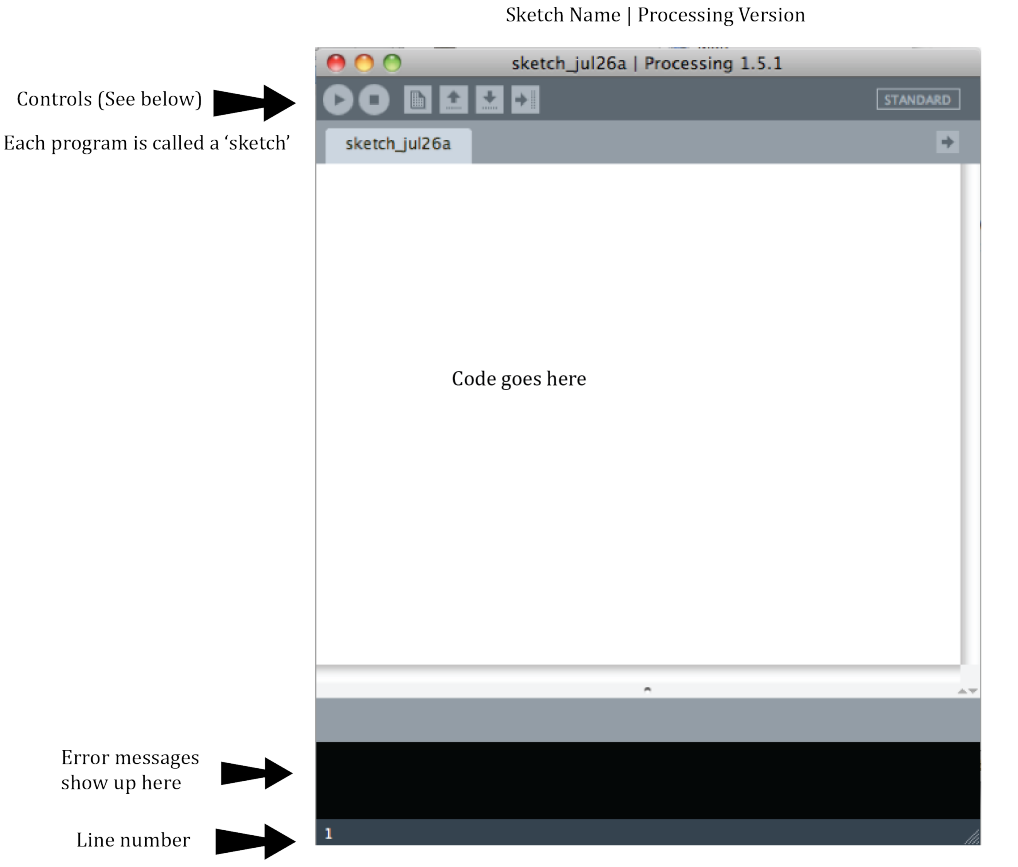
• Open up Processing. You should see something like the window below:

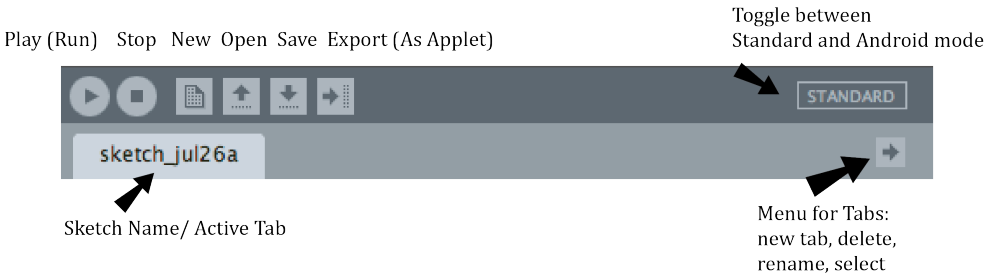


The IDE (Integrated Development Environment, i.e. what all the buttons do)

Intro to Processing / Tweaking Simon

SparkFun Electronics Summer Semester





Your First Sketch

• In the programming area, type:

```
print("hello world!");
```

• Press the 'run' button. Voila! You should see the message 'hello world!' in the output area of the IDE. You will also notice that a small grey box popped up – that's because Processing is primarily used as a visual tool, and so it comes built in with a canvas ready to draw on.

Drawing

• Drawing simple shapes in Processing is very straightforward. For example, typing:

```
ellipse(40, 50, 100, 110);
```

Gives us an ellipse with an x coordinate of 40, a y coordinate of 50, a width of 100, and a height of 110. Notice how the size goes off the canvas – we'll get to that in a sec.

Processing has a ton of built-in methods and functions - it would take us way too long to cover them all – but the full reference can be found at:
http://processing.org/reference/

Let's get serious

While very simple Processing sketches don't require much in the way of format, for anything dynamic (including motion, repetition, reading data, etc.) you'll want to set up your processing sketch with a certain structure.

To help with this, Processing has a few reserved functions (much like Arduino): setup() and draw()

As you might imagine, we do our setup within the setup() function and our drawing within the draw() function. For practical purposes this means that setup() only gets

called once at the beginning of the sketch, and that draw() is called repeatedly, much like the loop() function in Arduino.

In addition, things like import statements for libraries and global variables typically go before the setup function. Custom functions, class declarations, and other reserved methods can go after the draw function or in a separate tab, depending on your preference. (We'll talk more about this later).

Bouncy Ball

In this example, we're going to see the basic structure of a Processing program set up to display a bouncing ball.

• In Processing, navigate to the top menu under File -> Examples. A pop-up window (or drop-down list) should appear. Go to Topics -> Motion -> Bounce and open the sketch.

• Go ahead and run it. You should see a white ball 'bouncing' off the edges of the screen. Here's the code if you can't find it:

```
int size = 60;       // Width of the shape
float xpos, ypos;    // Starting position of shape

float xspeed = 2.8;  // Speed of the shape
float yspeed = 2.2;  // Speed of the shape

int xdirection = 1;  // Left or Right
int ydirection = 1;  // Top to Bottom

void setup()
{
  size(640, 200);
  noStroke();
  frameRate(30);
  smooth();
  // Set the starting position of the shape
  xpos = width/2;
  ypos = height/2;
}

void draw()
{
  background(102);
```

Intro to Processing / Tweaking Simon

SparkFun Electronics Summer Semester

```
  // Update the position of the shape
  xpos = xpos + ( xspeed * xdirection );
  ypos = ypos + ( yspeed * ydirection );

  // Test to see if the shape exceeds the boundaries of the screen
  // If it does, reverse its direction by multiplying by -1
  if (xpos > width-size || xpos < 0) {
    xdirection *= -1;
  }
  if (ypos > height-size || ypos < 0) {
    ydirection *= -1;
  }

  // Draw the shape
  ellipse(xpos+size/2, ypos+size/2, size, size);
}
```

A few things worth noticing / doing:

• Notice how setup() and draw() are used and where they're placed, look at how global variables are used

• Look at some of the common built-in functions: size(), background(), width, height, frameRate(), smooth()

• Play with the size, speed, and direction variables and see what happens

• Move the background() command to the setup() function. What happens? Why?

• Change colors using stroke(), fill(), and background()

• Remove smooth() – what happens?

Intro to Processing / Tweaking Simon

SparkFun Electronics Summer Semester

Hacking Simon!

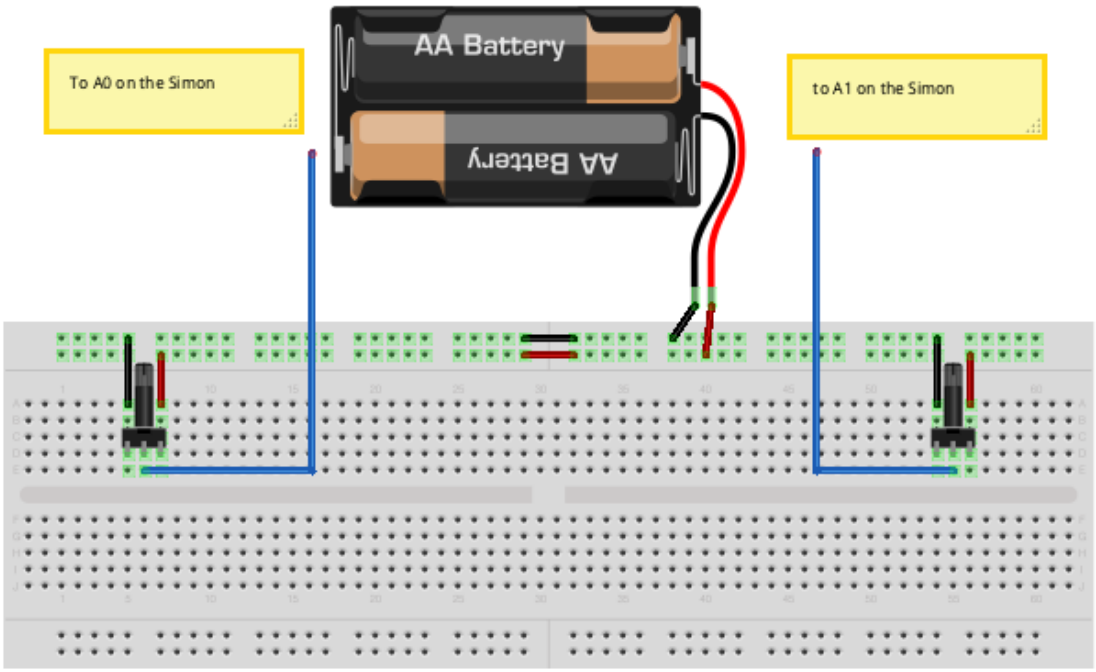
Part 1: Hardware Setup

• If you haven't already, solder in some female headers to the breakout pins on your Simon.

• Next, get your breadboard and run power and ground from the VCC and GND pins on the Simon to the breadboard.

• On your breadboard, set up 2 trimpots, somewhat far apart, and run them to pins A0 and A1 on the Simon. Make sure you run power and ground to your trimpots as well.

A little diagram: Just think of the batteries as your power and ground from the Simon board. You'll want to space out the two potentiometers as much as possible, so that it's easier to turn both at once.



Part 2: Arduino

Now that you're hardware is setup, open a new sketch in Arduino, and paste in the SimonSketch code:

```
/* Simon Sketch - A Simon Tweak from SparkFun */

//define pins for led's and buttons
#define blueLed 13
#define yellowLed 3
#define redLed 5
#define greenLed 10

#define blueButton 12
#define yellowButton 2
#define redButton 6
#define greenButton 9

int leftPot = A0;
int rightPot = A1;


int buttonState;  //variable to detect button press
int numButtons = 4;  //number of buttons
int buttons[] = {  //put our buttons in an array
  blueButton, yellowButton, redButton, greenButton};
int leds[] = { //put our led's in an array
  blueLed, yellowLed, redLed, greenLed};

void setup() {

  //init our pins - input for buttons, output for led's
  for(int i = 0; i < numButtons; i++) {
    pinMode(buttons[i], INPUT);
    pinMode(leds[i], OUTPUT);
    digitalWrite(buttons[i], HIGH);  //init internal pull-up on button pins
  }

  Serial.begin(9600);  //begin serial communication

  //UnComment this line after you configure your button pins
  //establishContact();
}
```

Intro to Processing / Tweaking Simon

SparkFun Electronics Summer Semester

```
void loop() {

//  UnComment these lines after you configure your button pins
//  if (Serial.available() > 0) {
//
//    int inByte = Serial.read();

    //send trimpot values
    int leftPotVal = analogRead(leftPot);
    Serial.print(leftPotVal, DEC);
    Serial.print(";");
    int rightPotVal = analogRead(rightPot);
    Serial.print(rightPotVal, DEC);

    //read buttons
    for(int i = 0; i < numButtons; i ++) {

      //is there a press?
      buttonState = digitalRead(buttons[i]);

      //if so, light up the corresponding led, and send the value to
      // processing via serial
      if (buttonState == 0) {
        digitalWrite(leds[i], HIGH);
        Serial.print(";");
        Serial.print(buttons[i]);
        delay(100);
        digitalWrite(leds[i], LOW);
      }
    }
    Serial.print('\n');

  }
//UnComment this line after you configure your button pins
//}

void establishContact() {
  while (Serial.available() <= 0) {
    Serial.println("hello");   // send a starting message
    delay(300);
  }
}
```

Intro to Processing / Tweaking Simon

SparkFun Electronics Summer Semester

• Next, get your FTDI breakout connected to your computer and to the Simon Board. You may need some male or female headers to do this.

• When plugging in the FTDI breakout to your Simon board, make sure the BLK and GRN markings on the FTDI breakout match up with those on the Simon.

• In Arduino, select 'LilyPad Arduino w/ATmega 328' as your board type.

• Upload the code!

• **IMPORTANT**: You will have to configure your button and LED pins at the top of the sketch to match the colors they are assigned to. (e.g., blueLed should be with blue button and actually make the color blue when pressed). You can do this by opening the Serial Monitor from Arduino and looking at the number that shows up when you press each button. This is the button pin number that should be associated with the color of that LED.

• Note the formatting we do around printing out the sensor values – this is going to help us separate them back out in Processing.

• After you've configured your button pins, Un-comment the lines of code that say 'uncomment this line after you've configured your buttons'. There are 3 places where you have to do this (around line 38, lines 44-46, and line 74).

• Re-upload your code to the Simon.

• If you re-uploaded the Arduino code successfully, open up the Serial Monitor – what do you see? Why?

• If you see 'hello' over and over again, you're done! Congrats!

Intro to Processing / Tweaking Simon

SparkFun Electronics Summer Semester

Part 3: Interlude – Getting serial data into Processing

We're going to walk through the basics of setting up Serial communication in Processing. You're going to want to do most of these steps any time you want to communicate with Processing using the serial port, especially for things like reading in sensor values from an Arduino.

• First, open Processing. (Duh.) Start a new sketch. Call it 'SerialBasic', or whatever you prefer.

• Under the 'Sketch' menu in Processing, go to 'Import Library' and select 'Serial I/O'. You should see something like: import processing.serial.*; in your sketch now. Good job! You just imported your first library. This allows us to make use of some Serial communication commands that will make our job much easier.

• Continue by copying the rest of the program below – make sure to read the comments so you understand what each line is for.

```
/* SerialBasic w/ handshake -
 * Prints out a set of sensor readings from Arduino using the Serial Library
 */

//import the Serial library – should be there already
import processing.serial.*;

Serial myPort;  //the Serial port object

// since we're doing serial handshaking,
// we need to check if we've heard from the microcontroller
boolean firstContact = false;


void setup() {

  //  initialize your serial port:
  //  this code picks the first port in the array of available ports,
  //   and sets the baud rate to 9600
  myPort = new Serial(this, Serial.list()[0], 9600);
  myPort.bufferUntil('\n'); //buffer until we get a carriage return
}
```

Intro to Processing / Tweaking Simon

SparkFun Electronics Summer Semester

```
void draw() {
  //we can leave the draw method empty,
  //because all our programming happens in the SerialEvent (see below)
}

//the serialEvent method is called every time we get new stuff in on the
// serial port
//in this case we're constantly getting info, so it acts as our draw loop
void serialEvent( Serial myPort) {

  //put the incoming data into a String -
  //the '\n' is our end delimiter indicating the end of a complete packet
  String myString = myPort.readStringUntil('\n');

  //make sure our data isn't empty before continuing
  if (myString != null) {

    //trim whitespace and formatting characters (like carriage return)
    myString = trim(myString);
    //println(myString);

    //look for our 'hello' string to start the handshake
    //if it's there, clear the buffer, and send a request for data
    if (firstContact == false) {
      if (myString.equals("hello")) {
        myPort.clear();
        firstContact = true;
        myPort.write('A');
        println("contact");
      }
    }
    else { //if we've already got contact, keep getting and parsing data

      //split the string of data back into separate values, using the
      //semicolon we printed in Arduino
      int sensors[] = int(split(myString, ';'));
      //run through the sensor values and print them out
      for (int sensorNum = 0; sensorNum < sensors.length; sensorNum++) {
        println("Sensor " + sensorNum + ": " + sensors[sensorNum]);
      }
      // when you've parsed the data you have, ask for more:
      myPort.write("A");
    }
  }
}
```

• Plug in your Simon and run your Processing sketch, if there are no errors, you should see the sensor values from your Simon being printed out in the Processing terminal. Sweet!

• If not, check for syntax errors, make sure you imported the serial library, hooked up your Simon properly, and don't have any other serial monitors open.

• Setting up Serial communication in this way in called a 'handshake' – Arduino waits for an incoming byte from Processing before it starts sending data, and waits to send more data until another byte from Processing arrives (signaling that Processing is done with the earlier data). This helps control the flow of data between the two programs.

Part 4: SimonSketch!

At this point, we've got our Simon hooked up to some sensors, and we can read the values of those sensors in Processing. Now it's time to use these values in Processing to create a little thing we call the SimonSketch (think Simon meets Etch-A-Sketch).

1. Copy and Paste your SerialBasic code into a new sketch, called 'SimonSketch'.

2. What does an etch-a-sketch do? It draws, and it moves using two dials, right? So we need to keep track of our position, using info from our two dials (our trim pots). Let's make global variables for our pots and the x and y coordinate of where we're currently drawing.

Add these lines below your 'Serial myPort;' declaration:

```
int leftPot;
int rightPot;

float x = width/2;
float y = height/2;
```

This gives us a place to store the values from the pots, as well as an x and y for the position of our cursor.

3. At the beginning of your Setup() function, we're going to put a few methods to 'setup' our drawing environment.

```
size(1000, 750); //defines the size of our canvas (width, height)
background(255); //defines the starting background color (255 is white)
stroke(255); //defines the starting stroke color
smooth(); //the smooth() function 'smoothes out' motion and curves
```

4. Although we will be drawing on the canvas, we can still leave the draw() method empty and put all the actual drawing in the serialEvent method.

5. Skip down to the section in the serialEvent method after we print out the sensor values (around line 70, shown below). We're going to assign the sensor values from Arduino to our local leftPot and rightPot variables:

```
for (int sensorNum = 0; sensorNum < sensors.length; sensorNum++) {
        println("Sensor " + sensorNum + ": " + sensors[sensorNum]);
        println(sensors.length);
      }

      leftPot = sensors[0];  // <- new stuff
      rightPot = sensors[1];
```

6. Now that we've got the sensor values in their own variables, we've got to turn them into numbers we want to use. In this case it means casting the integers as floats so that we can 'map' them to the width and height of the canvas – this is just an easy way of taking the normal range of sensor values (0 to 1023) and mapping them in a way that makes sure they cover the whole canvas evenly (since our canvas might be bigger or smaller than 1023 pixels squared).

Add the following directly below the lines you just wrote:

```
y = (float)leftPot; //casting values from int to float
x = (float)rightPot;

y = map(y, 0, 1023, 0, height); //map to height and width of canvas
x = map(x, 0, 1023, 0, width);

//print out the new values for good measure
println("X: " + x + " " + "Y: " + y);
```

7. Great. We have sensor values that map to the height and width of our canvas, so now we just have to draw a point (or a small circle) wherever our sensors tell us to go:

```
ellipse(x, y, 5, 5);
```

Yup. Simple as that.

8. Next, we're going to add in some functionality for changing colors based on the button pushes from Simon, as well as an 'erase' mode if we push two buttons at once. To do this, we can just check the size of the sensor value array – if we get 3 values, the third corresponds to the button we should change colors to. If we can 4 values, then 2 buttons are being pushed at once, which is our cue to go into 'erase' mode.

```
//if there's a third value, a button has been pressed, so change the color

if (sensors.length > 2) {
  int colorC = sensors[2];
  changeColor(colorC);
//this is passing the value to our 'changeColor' function (see below)
}

if (sensors.length > 3) {  //if you press 2 buttons, set to erase mode
        fill(255);
        stroke(255);
}
```

Before this code will work – we need to write our changeColor method that we use above. Put this after your serialEvent method at the end of your sketch:

```
void changeColor(int colorC) {
  //match the color to the button press
  switch (colorC) {
  case 2: //yellow
    stroke(255, 255, 0);
    fill(255, 255, 0);
    break;
  case 6: //red
    stroke(255, 0, 0);
    fill(255, 0, 0);
```

```
    break;
  case 9: //green
    stroke(0, 255, 0);
    fill(0, 255, 0);
    break;
  case 12: //blue
    stroke(0, 0, 255);
    fill(0, 0, 255);
    break;
  default:
    stroke(0, 0, 255);
  }
}
```

That's it! You've hacked your Simon! Try running your sketch. You'll have to push a button to get drawing, then control the drawing with the two potentiometers, just like an Etch-A-Sketch. Try changing colors using the buttons, and make sure your erase mode works. Get painting!

Here's the full sketch in case you get stumped:

```
/* SimonSketch - A Simon Tweak from Sparkfun
 * Use the buttons on your simon to pick colors,
 * the trimpots to move your cursor
 * 2 = yellow
 * 6 = red
 * 9 = green
 * 12 = blue
 * incoming string = leftPot;rightPot;buttonPress
 * if no button press, then just leftPot;rightPot
 */

import processing.serial.*;

Serial myPort;
int leftPot;
int rightPot;

float x = width/2;
float y = height/2;

// Whether we've heard from the microcontroller
boolean firstContact = false;
```

Intro to Processing / Tweaking Simon

SparkFun Electronics Summer Semester

```
void setup() {

  size(1000, 750);
  background(255);
  stroke(255);
  smooth();
  println(Serial.list()); //list serial ports
  //init serial object (picks 1st port available)
  myPort = new Serial(this, Serial.list()[0], 9600);
  myPort.clear();
}

void draw() {
}

void serialEvent(Serial myPort) {

  String myString = myPort.readStringUntil('\n');
  // if you got any bytes other than the linefeed:
  if (myString != null) {

    myString = trim(myString);
    //println(myString);

    if (firstContact == false) {
      if (myString.equals("hello")) {
        myPort.clear();
        firstContact = true;
        myPort.write('A');
        println("contact");
      }
    }
    else {
      int sensors[] = int(split(myString, ';'));
      // print out the values you got:
      for (int sensorNum = 0; sensorNum < sensors.length; sensorNum++){
        println("Sensor " + sensorNum + ": " + sensors[sensorNum]);
        println(sensors.length);
      }

      leftPot = sensors[0];
      rightPot = sensors[1];

      y = (float)leftPot;
      x = (float)rightPot;

      y = map(y, 0, 1023, 0, height);
      x = map(x, 0, 1023, 0, width);
```

Intro to Processing / Tweaking Simon

SparkFun Electronics Summer Semester

```
      println("X: " + x + " " + "Y: " + y);

      ellipse(x, y, 5, 5);
//if there's a third value, a button has been pressed, change the color
      if (sensors.length > 2) {
        int colorC = sensors[2];
        changeColor(colorC);
      }

      if (sensors.length > 3) { //got 2 buttons, set to erase mode
        fill(255);
        stroke(255);
      }
      // when you've parsed the data you have, ask for more:
      myPort.write("A");
    }
  }
}

void changeColor(int colorC) {
  //match the color to the button press
  switch (colorC) {
  case 2: //yellow
    stroke(255, 255, 0);
    fill(255, 255, 0);
    break;
  case 6: //red
    stroke(255, 0, 0);
    fill(255, 0, 0);
    break;
  case 9: //green
    stroke(0, 255, 0);
    fill(0, 255, 0);
    break;
  case 12: //blue
    stroke(0, 0, 255);
    fill(0, 0, 255);
    break;
  default:
    stroke(0, 0, 255);
  }
}
```

Going Further

The Simon has 6 Analog Pins broken out for us to use, so why stop after only using 2? Let's push it a little further. In addition, we're going to write a class called *SimonCursor* to control how our cursor behaves while we draw.

The following is just one example, but feel free to improve and improvise using other components.

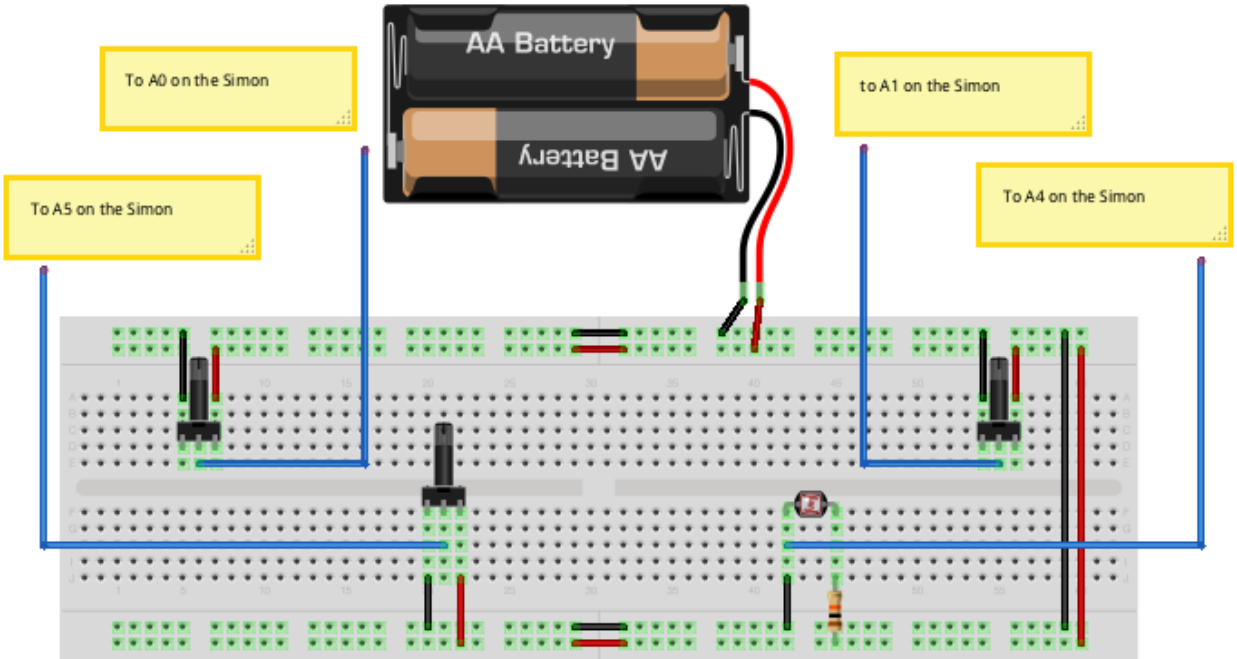
SimonSketchPro with Size and Transparency Control

Step 1: Hardware

• A another Trim Pot to your board, and connect it to Pin A5 on the Simon

• Add a photo-resistor to your board (don't forget the 10K resistor), and connect it to Pin A4 on the Simon

It should look something like this when you're done:



Step 2: Arduino

Since we added more sensors, we're going to have to change our Arduino code to read those sensors and send them out to Processing.

1. Copy and Paste your 'SimonSketch' code into a new sketch called 'SimonSketchPro' (or whatever you like).

2. Add these lines at the beginning of your sketch for Pin declarations (new code is **Bold**):

```
int leftPot = A0;
int rightPot = A1;
int lightSensor = A4;
int sizePot = A5;
```

3. In our loop() method, we'll have to read, format, and print the new sensor values:

```
int leftPotVal = analogRead(leftPot);
Serial.print(leftPotVal, DEC);
Serial.print(";");
int rightPotVal = analogRead(rightPot);
Serial.print(rightPotVal, DEC);

Serial.print(";");   // <- new
int sizePotVal = analogRead(sizePot);
Serial.print(sizePotVal, DEC);
Serial.print(";");
int lightSensorVal = analogRead(lightSensor);
Serial.print(lightSensorVal, DEC);
```

4. Before you upload, comment out the lines for the handshake and check your serial monitor to make sure the sensor values are being sent correctly.

IMPORTANT: Every photocell is a little different and gets different values depending on the conditions (sunny vs. cloudy, indoor vs. outdoor, etc.). As you look at the serial monitor, make a note of the high and low values you're getting – we'll need this later in the Processing sketch.

Make sure to uncomment the handshake code and re-upload your code before moving on.

That's it for the Arduino part! On to our Processing code!

Step 3: Processing

Now we have to change a few things in our Processing code. We'll have the extra trim pot control the size of our stylus, and the photocell will control the transparency.

1. Copy and Paste your 'SimonSketch' Processing code into a new sketch called 'SimonSketchPro' (or whatever you like).

2. Let's start writing our class. You can start a class at the bottom of your current code, but you may find it cleaner to create a new tab (under the tab menu) and name it after your class, in this case SimonCursor.

3. In this case, our class is going to take care of the values from our four sensors to control the X position, Y position, size of the cursor, and its transparency. So let's declare our class and its attributes:

```
class SimonCursor {

 float x;
 float y;
 float siz;
 float light;

}
```

4. After the 'float light;' line, we need to add a *constructor*. This is a statement that prepares the object for use upon its creation, by setting up the parameters that align with the variables we just declared. Basically it's a method with the same name as your class, and a dummy variable for each attribute your object possesses. It may look a little strange, but it's necessary:

```
SimonCursor (float _x, float _y, float _siz, float _light) {
  x = _x;
  y = _y;
```

```
  siz = _siz;
  light = _light;
 }
```

Notice the underscores – these differentiate the variables in our constructor method from those in our class (even though we're assigning them to each other).

5. Next we're going to crate a render() method to draw our cursor with the position and size that we've passed in:

```
  void render() {
    ellipse(x,y,siz,siz);
  }
```

6. Finally, we're going to move our entire changeColor method into our new class – just copy and paste the entire method in, after the render() method. Don't forget the last curly bracket (}) to close out our class.

7. We're done with writing our class – now we need to go back into the main Processing sketch and change our code to make use of our new class and to read in the new values from our two new sensors.

8. Back in our SimonSketchPro tab,we need to declare our SimonCursor object from our new class and create a few more global variables for our new sensor values coming in (new code in **bold**):

```
SimonCursor myCursor;
int leftPot;
int rightPot;
float sizePot = 0;
int lightSensor = 0;
```

9. We also need to define variables to keep track of the new dimensions we want to control: size and light (new code in **bold**):

```
float x = width/2;
float y = height/2;
float siz; // (size is a protected term in processing, hence 'siz')
float light;
```

10. Now's the time to put in the high and low values you recorded from your photocell:

```
// our high and low photocell values
int high = 550;
int low = 150;
```

11. Next, skip down to your serialEvent() method. We'll need to find a place for the extra sensor values coming in (new code in **bold**):

```
leftPot = sensors[0];
rightPot = sensors[1];
sizePot = sensors[2];
lightSensor = sensors[3];
```

12. As before, we'll have to convert the sensor values to floats so we can map them to the range of values we want (new code in **bold**):

```
y = (float)leftPot;
x = (float)rightPot;
siz = (float)sizePot;
light =(float)lightSensor;
y = map(y, 0, 1023, 0, height);
x = map(x, 0, 1023, 0, width);
siz = map(siz, 0, 1023, 2, 50);
light = map(light, low, high, 0, 255); //note the high and low values
println("X: " + x + " " + "Y: " + y + " " + "Size: " + siz + " " + "Light: "
+ light);
```

13. Now we get to make use of our new cursor object – we'll create a new SimonCursor object called 'myCursor', and pass it in all the variables it expects (x,y,size, light):

```
myCursor = new SimonCursor(x,y,siz,light);
```

14. We've also got to tell it to draw itself now that it has all the values it needs:

```
myCursor.render();
```

15. Since our normal sensor array size is going to be bigger (always reading 4 sensors instead of 2), we have to adjust our code for determining when to change colors and when to enter erase mode – and since we moved our changeColor method into our new class we've got to call that method as part of our myCursor object:

```
if (sensors.length > 4) {
      int colorC = sensors[4];
      myCursor.changeColor(colorC);
}

if (sensors.length > 5) {  //if you press 2 buttons, set to erase mode
      fill(255);
      stroke(255);
}
```

9. The last change is incorporating the photocell value into our changeColor() method back in our Class tab, to control the alpha value (transparency) of our colors (new code in **Bold**):

```
void changeColor(int colorC) {
  //match the color to the button press
  switch (colorC) {
  case 2: //yellow
    stroke(255, 255, 0, light);
    fill(255, 255, 0, light);
    break;
  case 6: //red
    stroke(255, 0, 0, light);
    fill(255, 0, 0, light);
    break;
  case 9: //green
    stroke(0, 255, 0, light);
    fill(0, 255, 0, light);
    break;
  case 12: //blue
    stroke(0, 0, 255, light);
    fill(0, 0, 255, light);
    break;
  default:
    stroke(0, 0, 255, light);
  }
}
```

You're finished! Try running your new Simon Sketch code – note that the more you cover the photocell when changing colors (that's when we send the photocell value to the changeColor method), the more transparent your drawing will be.