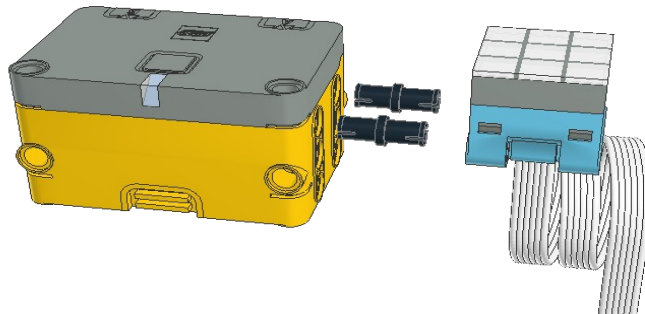# Bubble Level Using Word Blocks
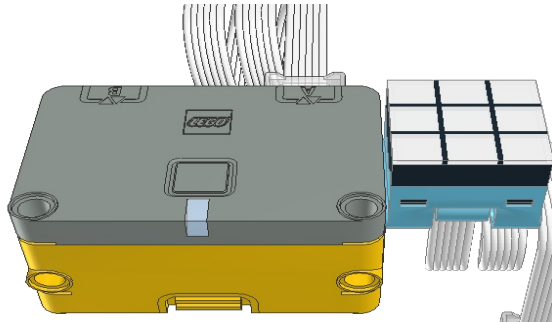
Exercise 1. Build a simple bubble level

This exercise is to build a couple of levels using just a hub and an LED matrix.
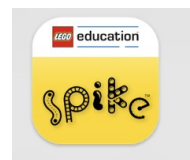
Let's do the build.

Attach the LED matrix to the right side of the hub using two small black connectors.

Connect the LED matrix to port A of the hub.

The LED matrix may be installed in one of two orientations. To test that it is correct, we need to turn on a single LED of the matrix. Turn on the host (Mac, iPad, PC), find the Spike App and start it.

When the app starts, select Spike Essentials on the left side of the window.

From the center portion the window, select the new project box.

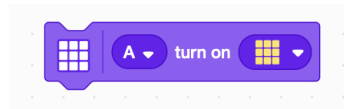Give the project a name, say "LED Test".

Select word blocks.

WORD BLOCKS

Click on the create button to open the programming window.

CREATE

The program is already started for you by supplying a start block when program starts

In the side menu find the block that turns on the LED matrix.

Click and drag that block below to the start block. When the shadow appears below the start block, drop the block and it will attach to the start block. This is like adding pieces of a puzzle.
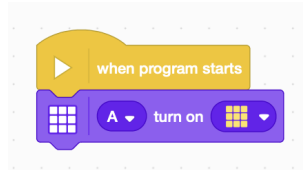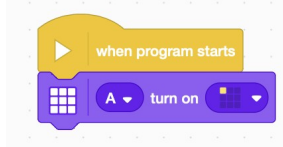
Open the drop down menu of the LED matrix block by clicking on its down arrow.
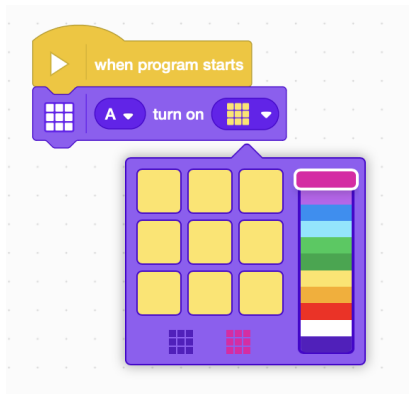
Select the colors of the LEDS so that only the upper left LED is lit. When completed it should look something like:
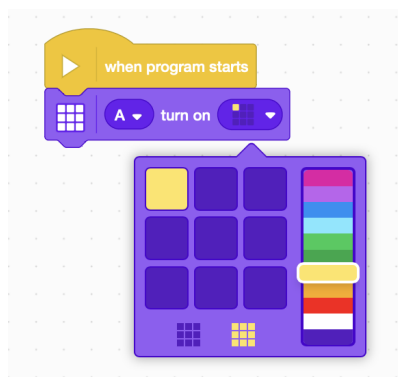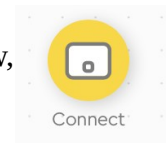
Close the drop down control menu.

Now we are ready to connect the app to the hub.

Find and click on the connect icon in the upper left corner of the app window, power up the hub by pressing the button. The LED in front of the button should flash blue.

The app window should change to show at least one hub. Select the hub you want to connect to. The LED on the hub should now light solid blue.
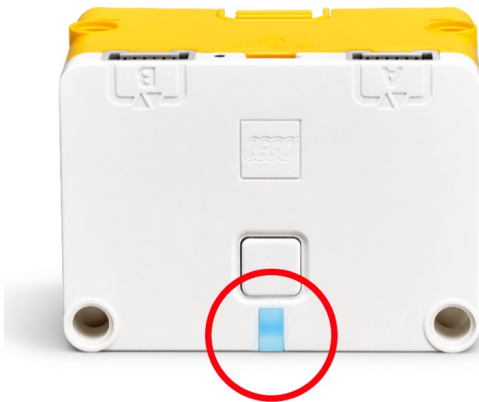
What do the Hub's color-coded error messages mean?

If you're having issues starting or connecting your Hub, your might see the Light display a color-coded error message. Here's an overview of the error message and how to fix them.

The Light is flashing orange.
    The Hub's battery is running low. Connect the Hub to your device via the USB cable to begin charging the battery.
The Light is flashing red.
    There's an extended load on the Hub. Turn off the Hub and let it cool down for 15 minutes.
The Light is blue.
    The Hub is connected to your device using BLE (Bluetooth Low Energy).
The Light is white.
    The Hub is connected to your device using a USB cable.
The Light is flashing white.
    The Hub is waiting to be connected.
The Light is flashing violet/green/blue.
    The Hub OS has been updated, and it's restarting.

Return to the project window.

Press the start button in the lower right corner of the app window.

Verify that the upper left hand LED of the LED matrix is lit. If it isn't, change the orientation of the LED display and test again.

Exercise 2. Program a simple bubble level

> A note about this tutorial. Some directions and commentary are provided on a page and then it asks you to write the code to implement the concept BEFORE turning the page. The next page has an example of what was being requested. If you want to learn the concept, the best way to do that is to actually use the concept. If you turn the page and copy the code, the only thing you will learn is how to copy the code. We could train a parrot to do that. The choice is yours: practice and learn or just copy. Don't be a parrot. (Of course the code does give you a way to check what you did against another way to solve the problem. Both may be right!)

The hub is a small computer that is easy to program. There is one thing to always remember about computers:

A computer is STUPID. It does exactly what you tell it to do. It knows only what you tell it. It cannot figure out what you meant to do. So tell it step by step exactly what you want it to do.

The next task is to write a program that indicates when the hub is not level and when it is.

1) Light up the LEDs on the side of the LED matrix that is the high side of the hub.

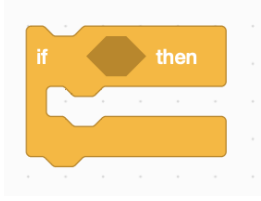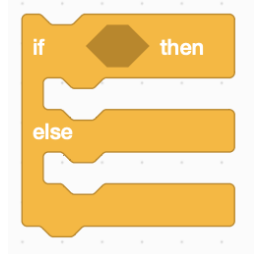2) When the hub is level, light up on the center LED.

A start to the program can look like the following:
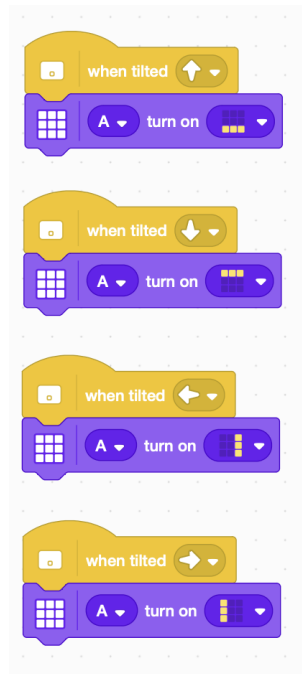
Ask yourself the same questions an engineer would ask: Does it meet all of the requirements? What is missing? How can this be fixed? Can this be improved?

The program satisfies the first requirement of reporting various tilts, but it does not report when the hub is level by lighting the center LED of the array. To fix this one must use a conditional statement. LEGO has two conditional statements: an if...then block and an if...then...else statement. Either can be used.

A conditional blocks in LEGO block can be either:

An "if...then…" block	an "if...then...else…" block
A "condition" is required which evaluates to a true or false value. These are light blue hexagonal shaped blocks.	A "condition" is required which evaluates to a true or false value. These are light blue hexagonal shaped blocks.
When the condition is true, the blocks under the "then" arm are executed and control passes to the end of the block.	When the condition is true, the blocks under the "then" arm are executed and control passes to the end of the block.
When the condition is false, control passes immediately to the end of the block.	When the condition is false, the blocks under the "else" arm are executed and control passes to the end of the block.

Conditional blocks are used when code is executed only some of the time or more specifically only when the condition is met.

In this exercise we want to change the LED matrix based on the conditions of the direction of the tilt of the hub or no tilt at all. The available conditions are (the direction is selectable by pressing on the oval with the downward pointing triangle:

To do the measurements over and over, a repeat block can be use. Spike had two kinds of repeat blocks. One repeats for a specified number of times and the other repeats forever. Well, at least until the program is stopped.
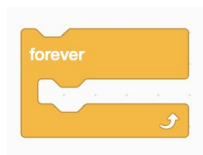
Can you write a program that satisfies the rerequirements. Try to program yourself before turning the page.
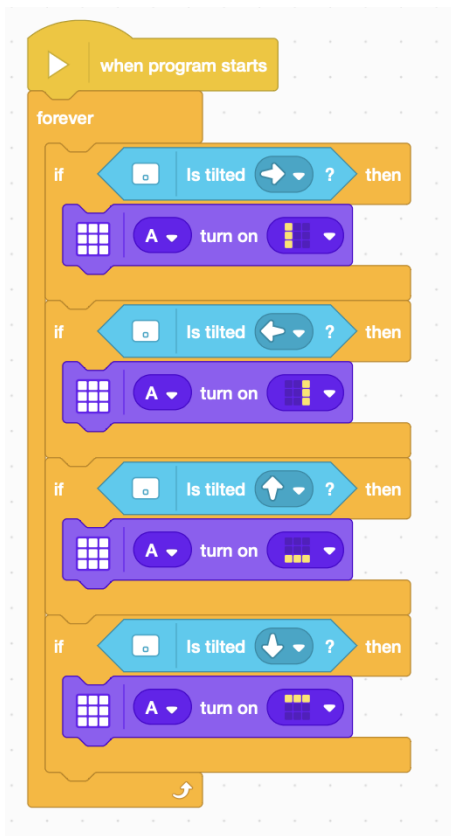
The tilt directions are covered, but what about no tilt at all?

The beauty of programming is there is always more than one way to solve the problem. The "correctness" of a particular solution can be dependent on a lot of things, but usually the simplest answer is the best.

So how does one test for the false outcome of a test? We can use the "else" part of a conditional, but we'll hold off on that for now to keep the program structure the same. There is a block to change a "false" condition to "true" and a "true" condition to "false". It is the "not" operator.

So we can find out when the hub is not tilted to the right. This is different that knowing when the hub is tilted to the left, because it includes when the hub is level. We need to know then no tilt is applied. We can easily test when any tilt is applied by using the "or" operator. This is true when either of its operands are true.

We can nest the conditions to test more than two operands.

The following is equivalent. Can you see the difference?

Try to improve the program with what you have learned and test your code before turning the page.
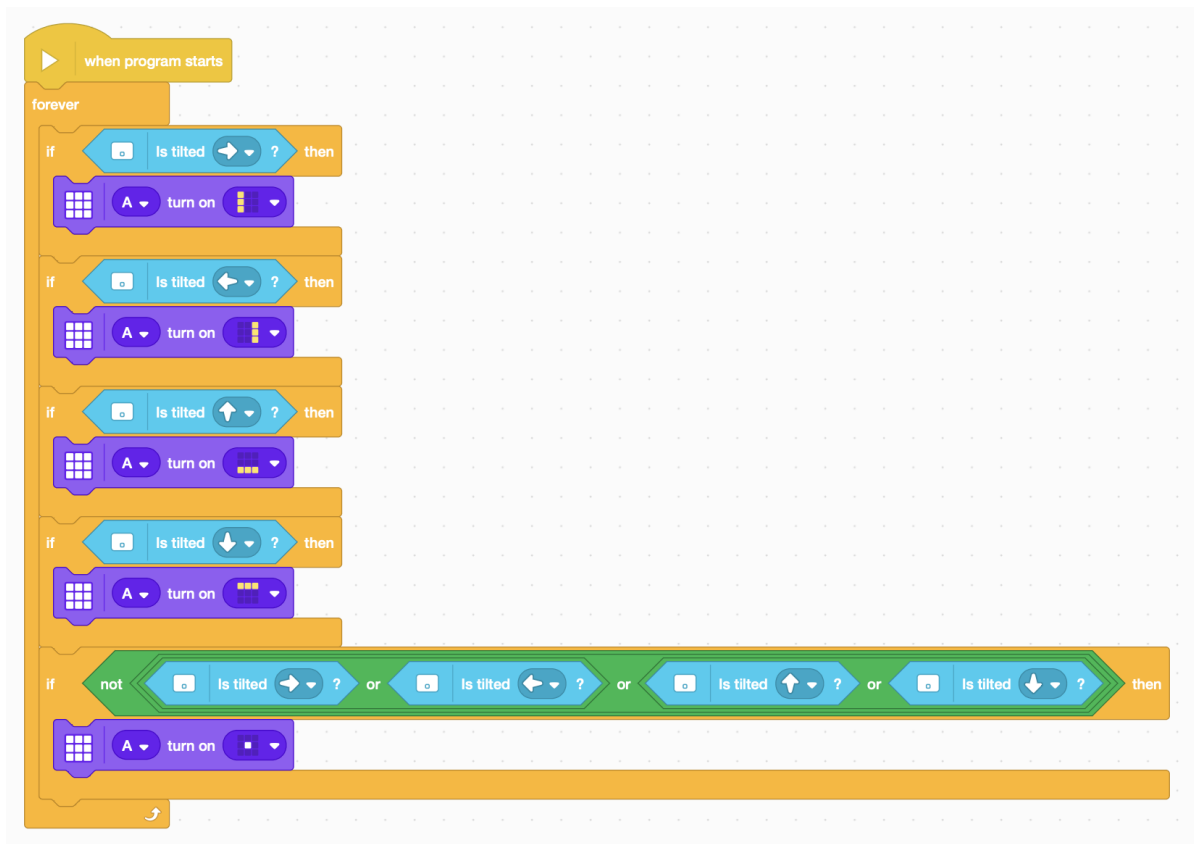
Another way to code the same functionality is to use conditionals with the "else" clause to pickup up on what to do when a condition is false.

Try to program the same functionality with "if...then...else…" blocks and test your code against the requirements before turning the page.
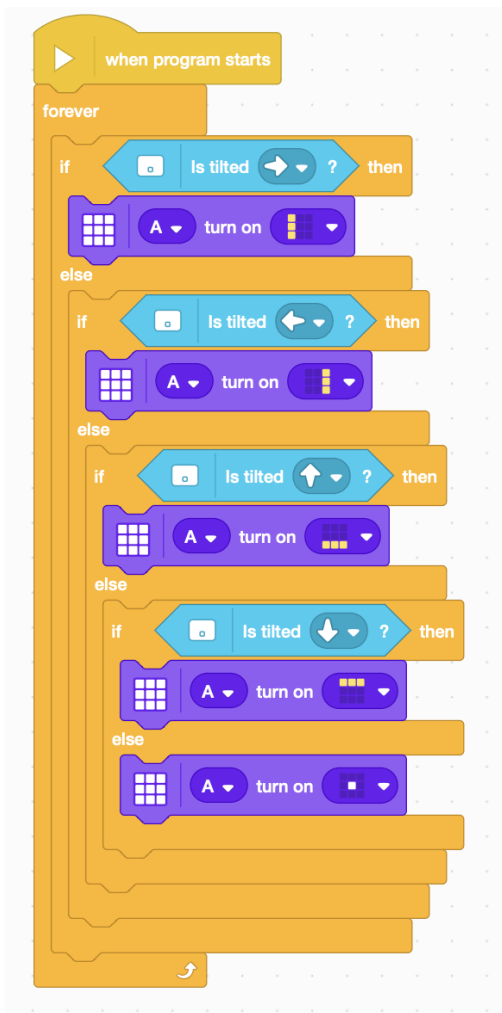
The next question an engineer asks is "Can this design be improved?"

This level works as long as you only tilt in one of four basic directions. It gets a little confused if you tilt two directions at once, say right and forward. We can build a sort of level called a bubble level that can handle being tilting in eight directions instead of the original four.

Try to write the program to work in eight directions and test your code to ensure that it works before turning the page to see an example.
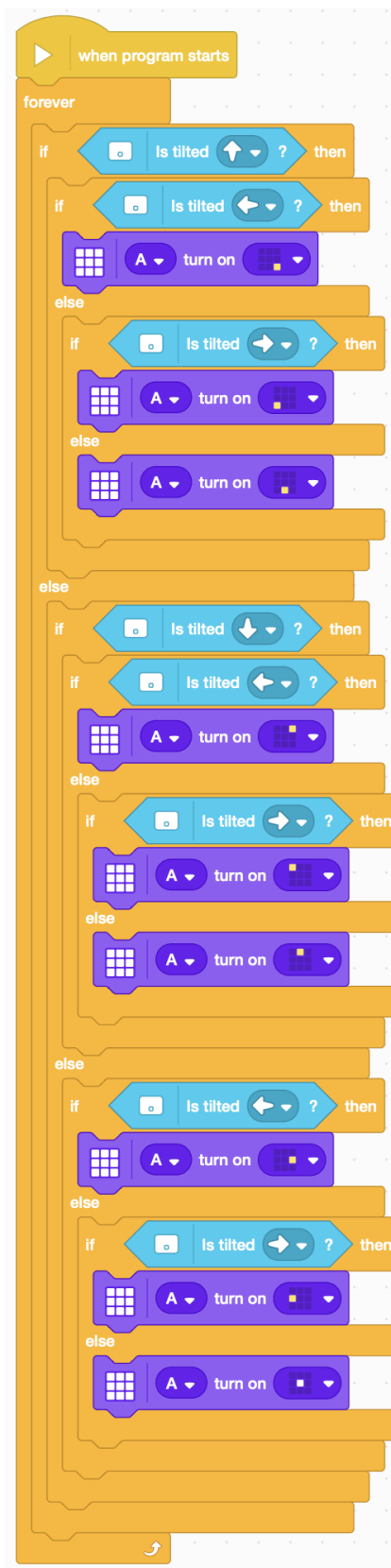
This program uses four pairs of "if...then...else…" conditionals to sort forward-backward-middle and then separately right-left-center.

Test this program to ensure that it acts as you expect. Are all branches of the code taken? (This is a fairly standard test of software quality.)

The bottom line is that the code really does not work. The "is tilted" blocks are coupled together, so that only one fires at a time. Luckily there is a way to fix this and that is to replace the "is tilted" blocks with a block that reads the tilt angle and then compares that angle to determine what direction the hub is tilted. (Pitch is the angle forward and backward off vertical and roll is the side to side angle off vertical. You can see these angles by clicking on the connect icon in the upper left corner of the window, while the hub is connected to the host computer and app. Tilt and turn the hub to see values change.

A bit of the problem is to know what value to use in place of the "100" value above. One could put in a number, but then whenever you wanted to change that value, you would have to change that value everywhere that it occurred.

Most programming languages, including LEGO, allow you to create a named variable, use the variable in place of a number, and even change the value of the variable during the running of the program. In this case the value stays fixed, or constant, as the value stays the same throughout the running of the program.

Create the variable by clicking on the icon: **Make a Variable**

Give the variable a meaningful name, say "Threshold". (If you name the variable something non-sensical like "frog" than you have to remember that "frog" really is an angle threshold. It is easier to remember that "Threshold" is an angle threshold.
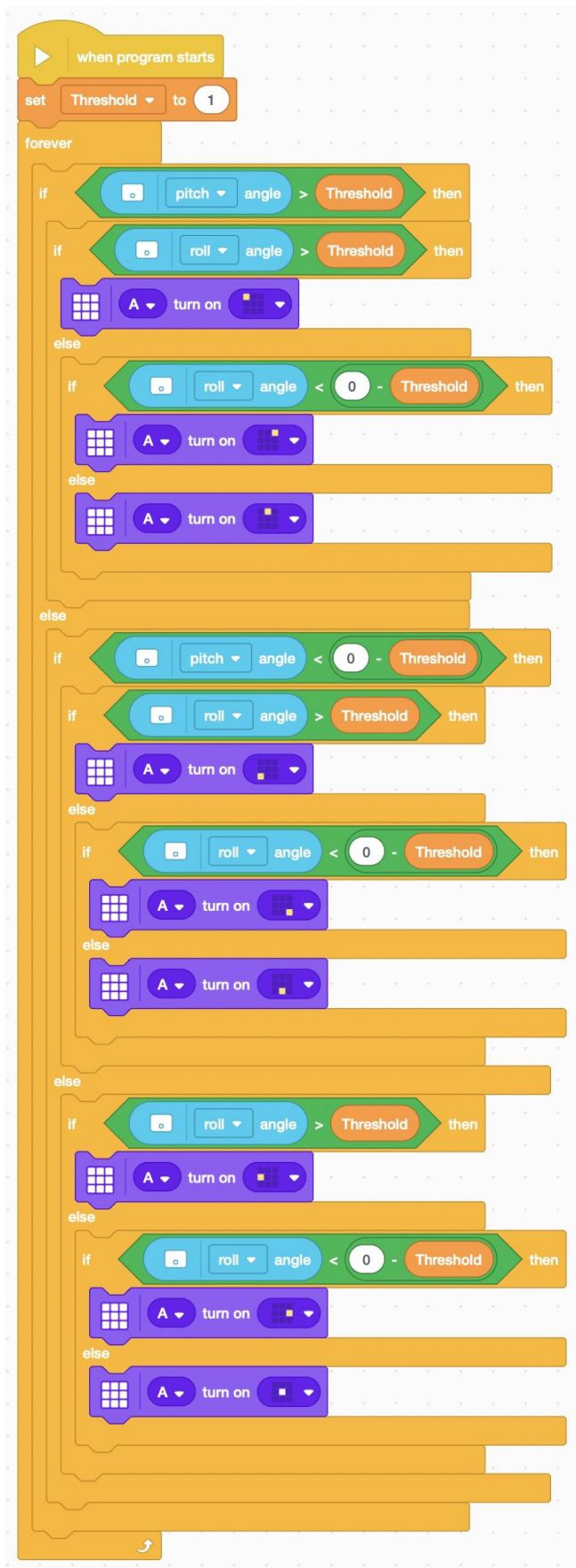
New Variable ✕
Threshold
CANCEL   OK
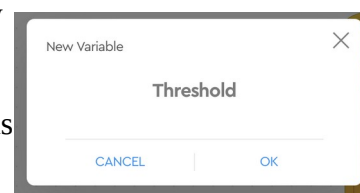
You can set the value of the new variable: set Threshold to 1

You can use the value of the new variable in your program with the value block: Threshold

Try to modify your program with the things on this page and test the program to see if it works as expected, before turning the page.

This code fixes the problems of the previous example. Note the changes. See how the comparisons are made with when the threshold is on the negative side of zero. Do you understand every block and what it is asking the computer to do? Also think about the how this coding is better or worse than the last program.

Again the pesky engineer asks "Can this be improved?" See if you can think of improvements. Write code to implement your ideas and test it to make sure it is operating before you turn the page.

This program provides the Threshold variable, but it is not clear what value it should be set at. Try higher values and see how the program behaves. Can you thing of applications where this characteristic would be desirable? Also try lower values and see how the program behaves. Does the program still work? Is this an improvement or not? What do you think the value should be? Many programs require a bit of tinkering to find the correct control values.
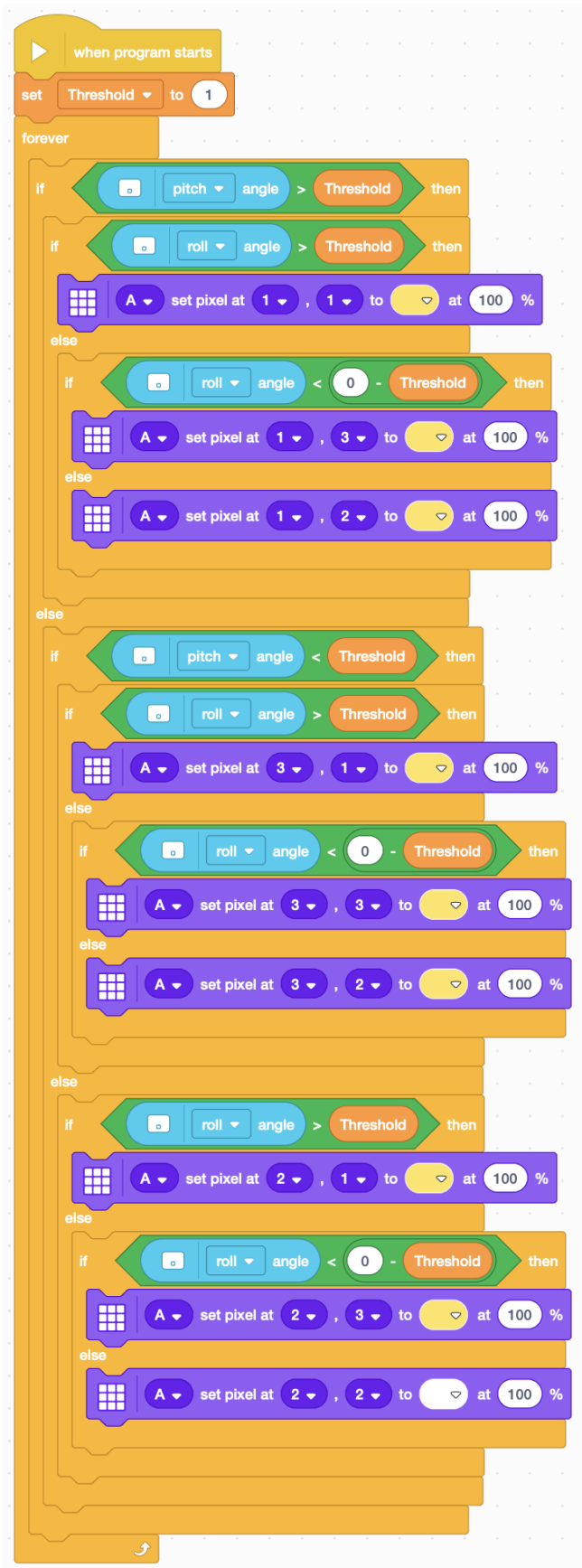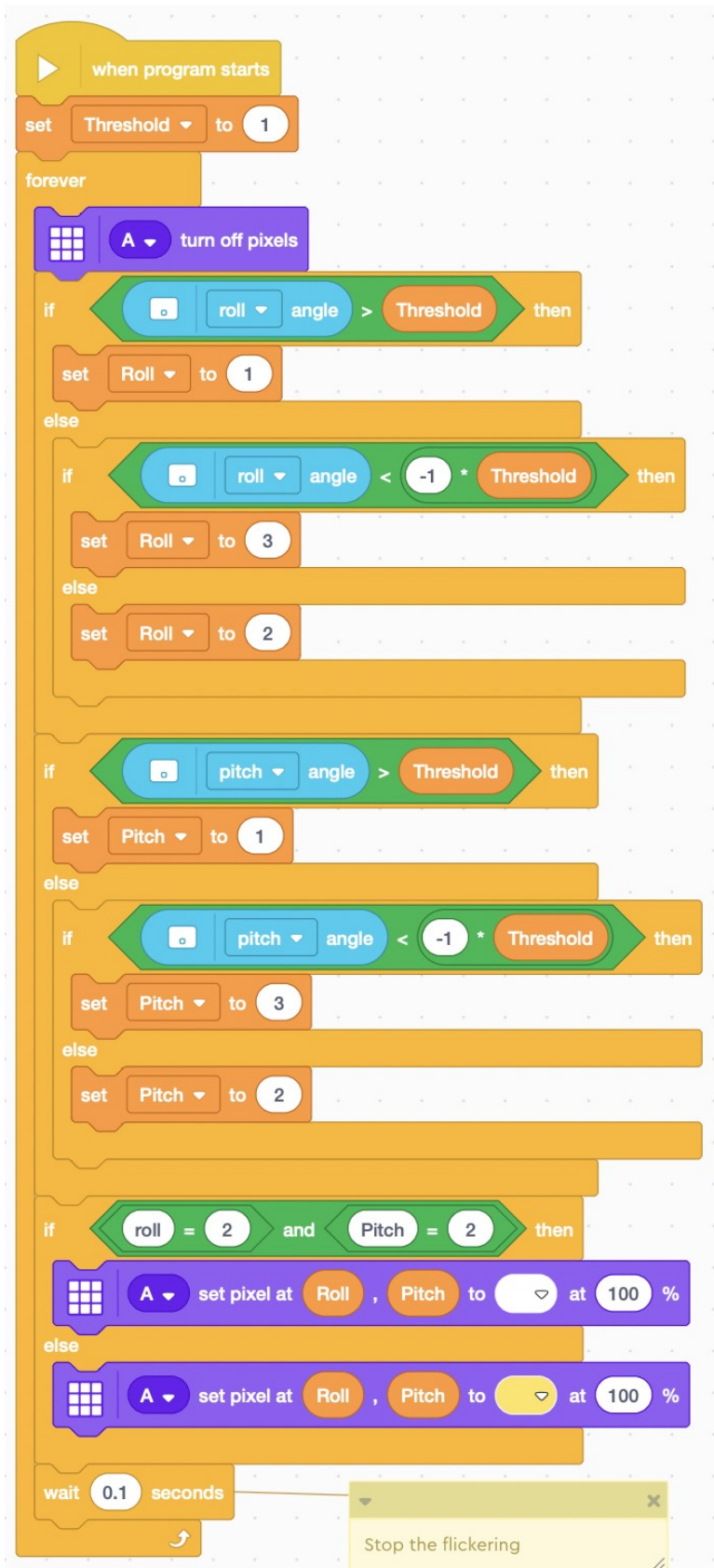
Look at the code to the left and find the differences between this code and the previous example. Do both programs do the same thing? If you are in doubt, you can implement this code and check it out. Is one program "better" than the other? Is one easier to read and understand?

Reading code like this is sometimes called a "code review." Reviewing other people's code makes you think about the problem to be solved in a different way. You try to make sure that the problem is being solved, not necessarily that the problem is being solved in the way that you think it should be solved. "Constructive" comments can be useful to improve the program and the skills of the programmer. Just putting a program down without offering real improvements does not help the problem being solved or the programmer. There are always more than one way of doing something and there can be multiple reasons why either one is better or either one is worse. There are always tradeoffs.

Guess what the engineer again asks? Can this nearly perfect project be improved yet again?

This time the engineer thinks that maybe the structure of the code can be improved to use less conditional statements and less repeated blocks of code and hence less chance for an error to creep in. The process of changing the structure of code is called re-factoring. Can you come up with a way to re-factor the code to make it less redundant.

Try out your ideas and test the code to make sure that it works before turning the page.

```
when program starts
set Threshold to 1
forever
    A  turn off pixels
    if  roll angle > Threshold  then
        set Roll to 1
    else
        if  roll angle < -1 * Threshold  then
            set Roll to 3
        else
            set Roll to 2

    if  pitch angle > Threshold  then
        set Pitch to 1
    else
        if  pitch angle < -1 * Threshold  then
            set Pitch to 3
        else
            set Pitch to 2

    if  roll = 2  and  Pitch = 2  then
        A  set pixel at Roll , Pitch to ▽ at 100 %
    else
        A  set pixel at Roll , Pitch to ▽ at 100 %

    wait 0.1 seconds
    Stop the flickering
```

How is this code different than the previous example. Do both programs do the same thing? Implement this code to be sure. Which is easier to write? Which is easier to read? Are the variables "Pitch" and "Roll" set to the same value all the time or are they set to different values depending on the particular leg of the program taken. What do these values mean? (Or another way of asking is why were these particular values chosen?)