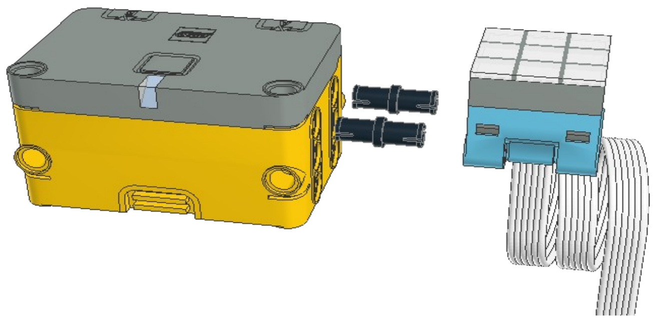
Bubble Level Using Word Blocks

Exercise 1. Build a simple bubble level

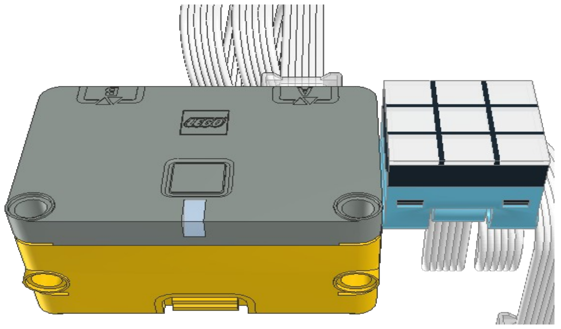
This exercise is to build a couple of levels using just a hub and an LED matrix.

Let's do the build.

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



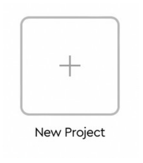
Connect the LED matrix to port A of the hub.



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.

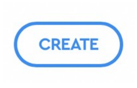
From the center portion the window, select the new project box.



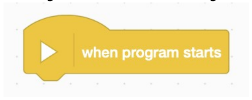
Give the project a name, say "LED Test".

Select word blocks.

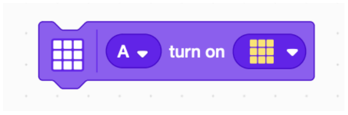
Click on the create button to open the programming window.



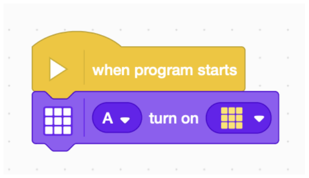
The program is already started for you by supplying a start block:when program starts



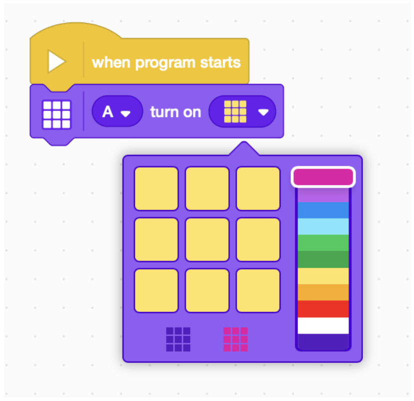
In the side menu find the block that turns on the LED matrix.



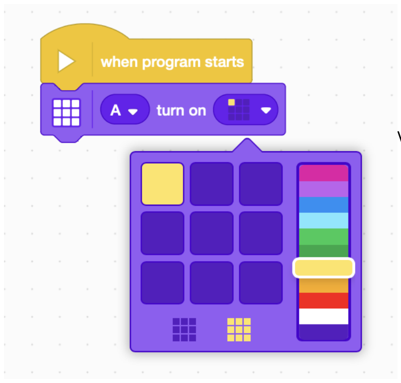
Click and drag that block below to the start block. When the shadow appears below the start block, drop the block and it will attach to the start block.



Open the drop down menu of the LED matrix block by clicking on its down arrow.

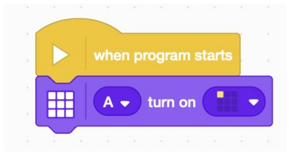


Select the colors of the LEDS so that only the upper left LED is lit. When completed it should look something like:



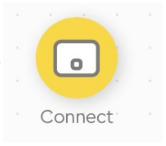
\

Close the drop down control menu.

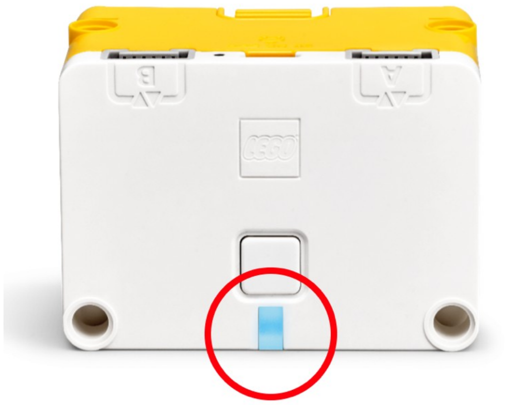


Now we are ready to connect the app to the hub.

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

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

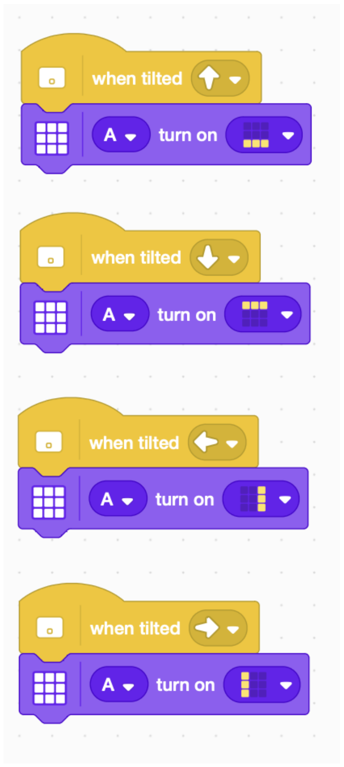
A computer is STUPID. It does exactly what you tell it to do. It knows only what you tell it. It cannot figure out what you meant to do. So tell it step by step exactly what you want it to do.

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:

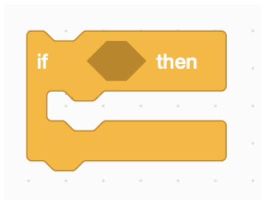
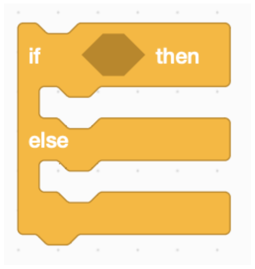


```
when tilted ↑
A turn on

when tilted ↓
A turn on

when tilted ←
A turn on

when tilted →
A turn on
```

Does it meet all of the requirements? What is missing? How can this be fixed?

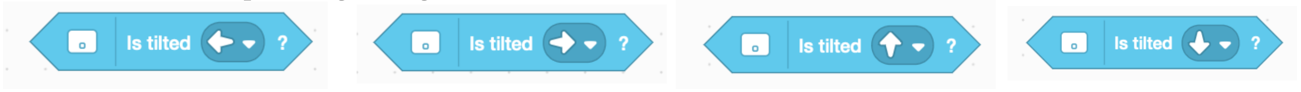
The program satisfies the first requirement of reporting various tilts, but it does not report when the hub is level by lighting the center LED of the array.To fix this one must use a conditional statement. LEGO has two conditional statements: an if...then block and an if...then...else statement. Either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

A conditional blocks in LEGO block can be either:

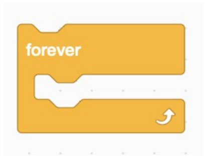
An "if...then…" block 	an "if...then...else…" block 
A "condition" is required which evaluates to a true or false value. These are light blue hexagonal shaped blocks.	A "condition" is required which evaluates to a true or false value. These are light blue hexagonal shaped blocks.
When the condition is 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.

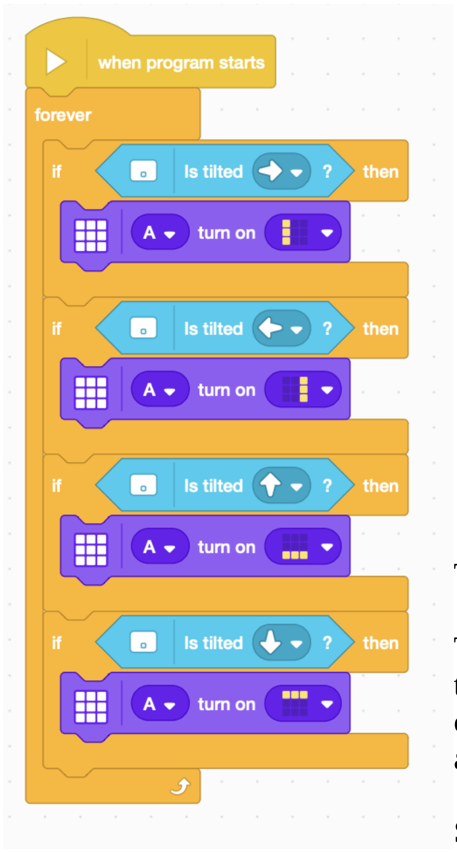
In this exercise we want to change the LED matrix based on the conditions of the direction of the tilt of the hub or no tilt at all. The available conditions are (the direction is selectable by pressing on the oval with the downward pointing triangle:



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 requirements. Try to program yourself before turning the page.



```
when program starts
forever
    if Is tilted → ? then
        A turn on
    if Is tilted ← ? then
        A turn on
    if Is tilted ↑ ? then
        A turn on
    if Is tilted ↓ ? then
        A turn on
```

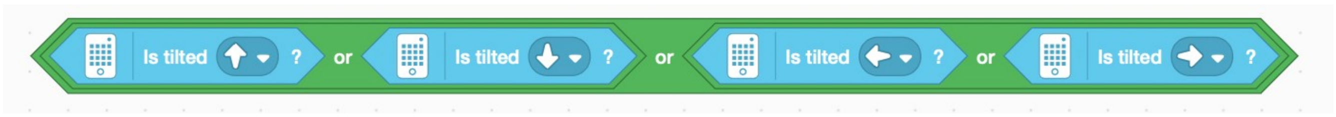
The tilt directions are covered, but what about no tilt at all?

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.

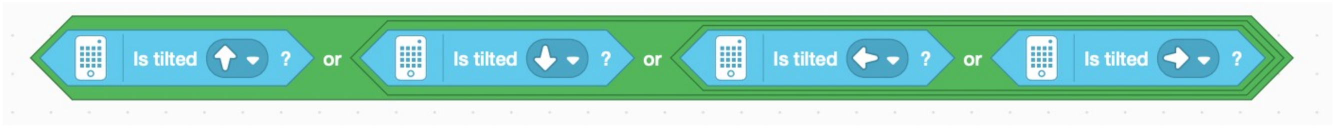
So we can find out when the hub is not tilted to the right. This is different that knowing when the hub is tilted to the left, because it includes when the hub is level. We need to know then no tilt is applied. We can easily test when any tilt is applied by using the “or” operator. This is true when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

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



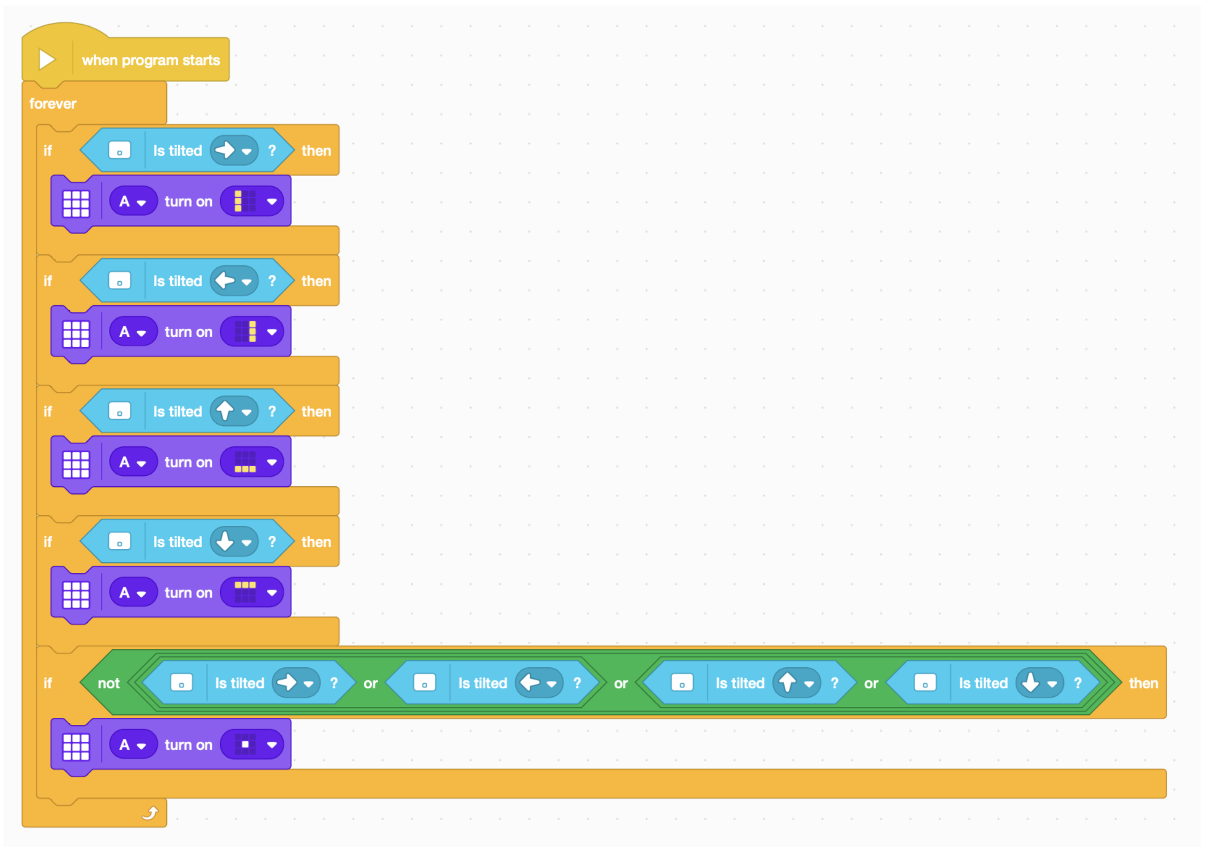
```
Is tilted ↑ ? or Is tilted ↓ ? or Is tilted ← ? or Is tilted → ?
```

The following is equivalent. Can you see the difference?



```
Is tilted ↑ ? or Is tilted ↓ ? or Is tilted ← ? or Is tilted → ?
```

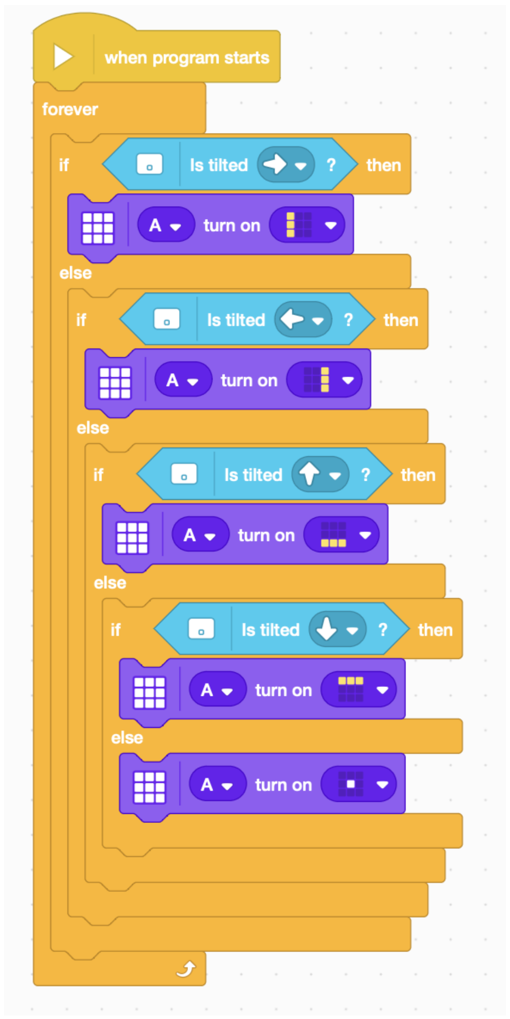
Try to improve the program with what you have learned and test your code before turning the page.



```
when program starts
forever
    if Is tilted → ? then
        A turn on
    if Is tilted ← ? then
        A turn on
    if Is tilted ↑ ? then
        A turn on
    if Is tilted ↓ ? then
        A turn on
    if not Is tilted → ? or Is tilted ← ? or Is tilted ↑ ? or Is tilted ↓ ? then
        A turn on
```

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.

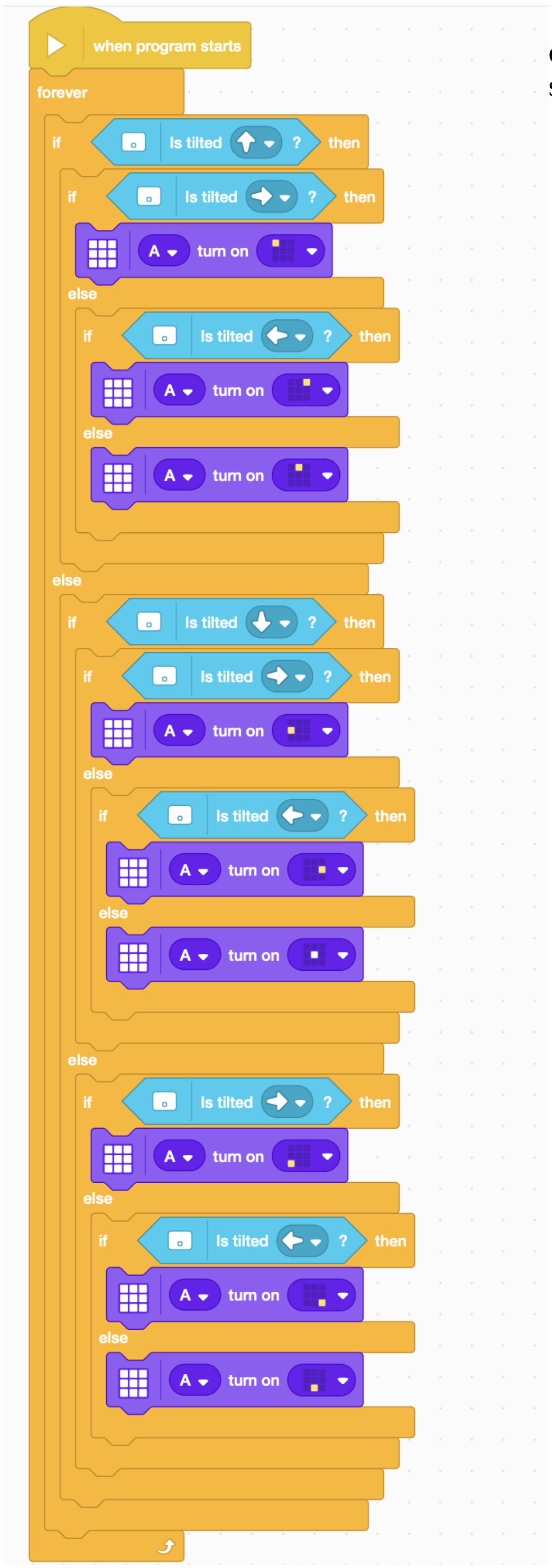


```
when program starts
forever
  if Is tilted → ? then
    A turn on
  else
    if Is tilted ← ? then
      A turn on
    else
      if Is tilted ↑ ? then
        A turn on
      else
        if Is tilted ↓ ? then
          A turn on
        else
          A turn on
```

The next question an engineer asks is “Can this design be improved?”

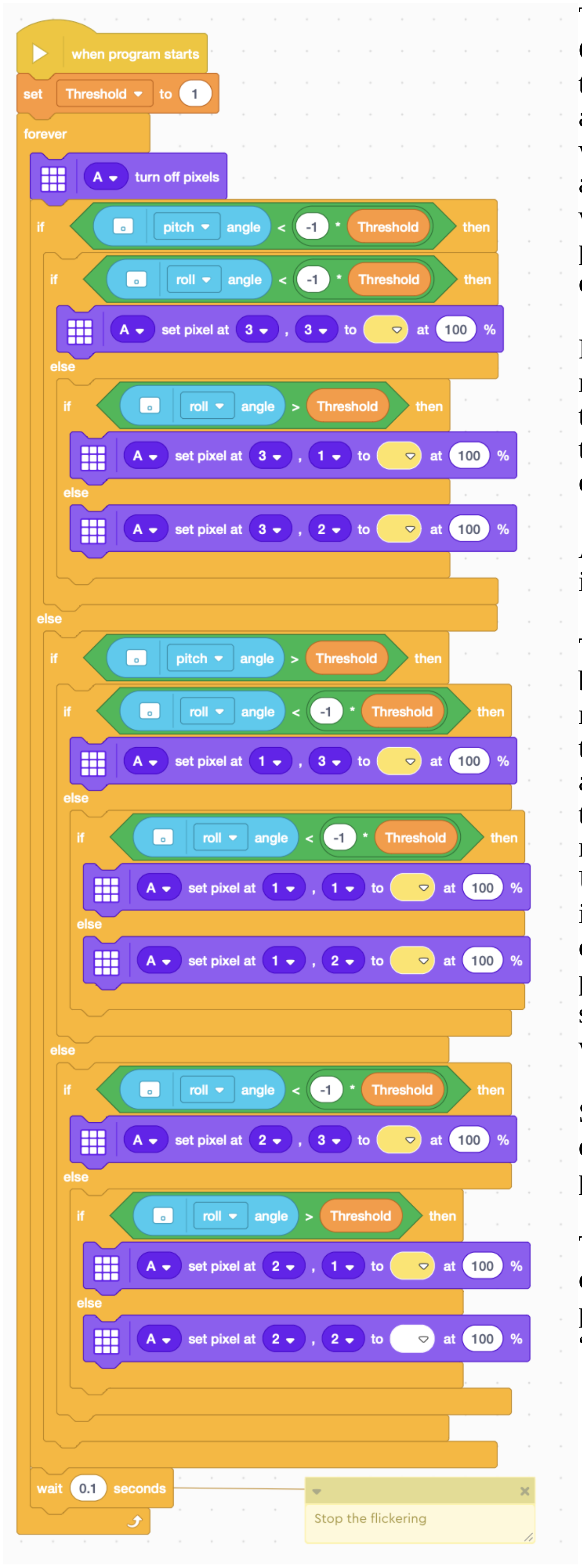
This level works as long as you only tilt in one of four basic directions. It gets a little confused if you tilt two directions at once, say right and forward. We can build a sort of level called a bubble level that can handle being tilting in eight directions instead of the original four.

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.



```
when program starts
forever
    if Is tilted ↑ ? then
        if Is tilted → ? then
            A turn on
        else
            if Is tilted ← ? then
                A turn on
            else
                A turn on
    else
        if Is tilted ↓ ? then
            if Is tilted → ? then
                A turn on
            else
                if Is tilted ← ? then
                    A turn on
                else
                    A turn on
        else
            if Is tilted → ? then
                A turn on
            else
                if Is tilted ← ? then
                    A turn on
                else
                    A turn on
```

This program uses four pairs of "if...then...else…" conditionals to sort forward-backward-middle and then separately right-left-center.



```
when program starts
set Threshold to 1
forever
    A turn off pixels
    if pitch angle < -1 * Threshold then
        if roll angle < -1 * Threshold then
            A set pixel at 3 , 3 to at 100 %
        else
            if roll angle > Threshold then
                A set pixel at 3 , 1 to at 100 %
            else
                A set pixel at 3 , 2 to at 100 %
    else
        if pitch angle > Threshold then
            if roll angle < -1 * Threshold then
                A set pixel at 1 , 3 to at 100 %
            else
                if roll angle < -1 * Threshold then
                    A set pixel at 1 , 1 to at 100 %
                else
                    A set pixel at 1 , 2 to at 100 %
        else
            if roll angle < -1 * Threshold then
                A set pixel at 2 , 3 to at 100 %
            else
                if roll angle > Threshold then
                    A set pixel at 2 , 1 to at 100 %
                else
                    A set pixel at 2 , 2 to at 100 %
    wait 0.1 seconds
```

Stop the flickering

This code is very similar to the last in functionality. Can you spot the differences and rationalize whether the change does the same thing or something different as the last program? Do you think the code will work? Do you understand every block and what it is asking the computer to do. Also think about the whether this coding is better or worse than the last program. You may want to implement the code to try out the different style.

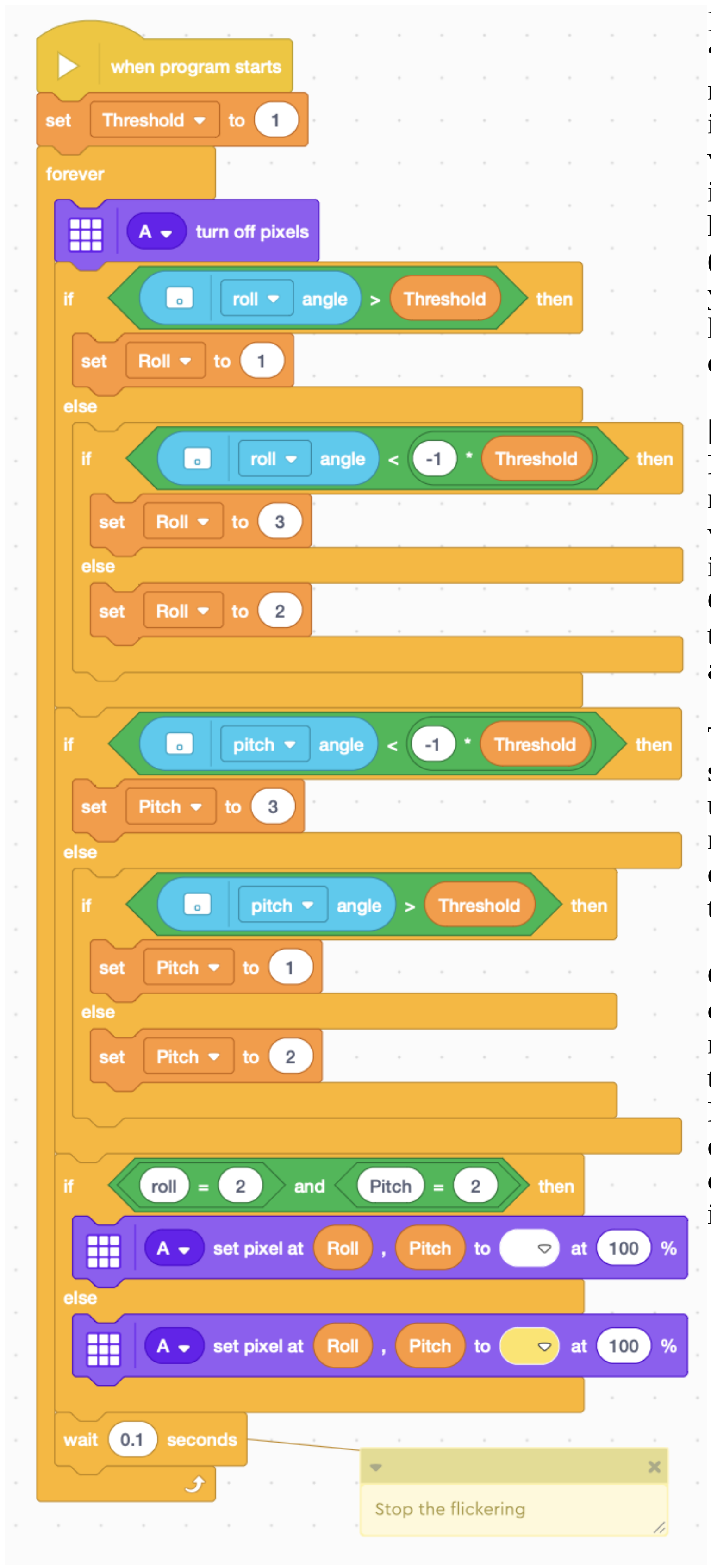
Reading code like this is sometimes called a “code review.” Reviewing other people’s code makes you think about things differently, and should not be about trying to get others to conform to your ways of expressing the code.

Again the pesky engineer asks “Can this be improved?”

The engineer thinks that a variable sensitivity would be nice. The hub provides a way to measure pitch and roll. Pitch is the angle forward and backward. Roll is the angle from side to side. (A third angle, yaw, is the angle of direction. We don’t need to use yaw at this time.) These angles can be viewed in real time. First make sure the hub is connected via Bluetooth (or USB) with the Spike App. Then press the “Connect” icon in the upper left of the Spike App window. This opens a window that shows the Hub with a selector to pick various angles. You can see that roll works for side to side tilts. Likewise you can see that pitch works for forward and backward tilts.

So he introduces a “variable” which can be set to a desired threshold for the number of degrees of roll or pitch for a condition to be met.

The task this time is to modify the existing code to change the “tilt condition” to a check of the roll and pitch angles against a variable threshold using a “greater than” and “less than” conditional operators.



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

If the engineer was honest he would ask, “Was this an improvement?” The answer is maybe. Since the stated objective was to improve sensitivity and with the threshold variable set to 1 or less, there is no improvement. If the objective was to allow higher angles before triggering the level (less sensitivity, then the answer would be yes. Can you think of applications where less sensitivity and less accuracy is desirable?

[What about zero? Does it work? Remember that the angle values are rounded before the test, so there should be values greater than 0, less than 0 and zero itself.]

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

The engineer thinks that maybe the structure of the code can be improved to use less condition statements and less repeated code and hence less chance for an error to creep in. The process of changing the structure of code is called re-factoring.

Change the code so that the pitch angle is checked to set a variable that selects the row of LEDs and that the roll angle is check to set a variable that selects the column of LEDs. Use those two variable to select the one LED to turn on. For extra measure change the color of the LED when the hub is level.