Mouse Painter: Emulate Mice with MakeCode

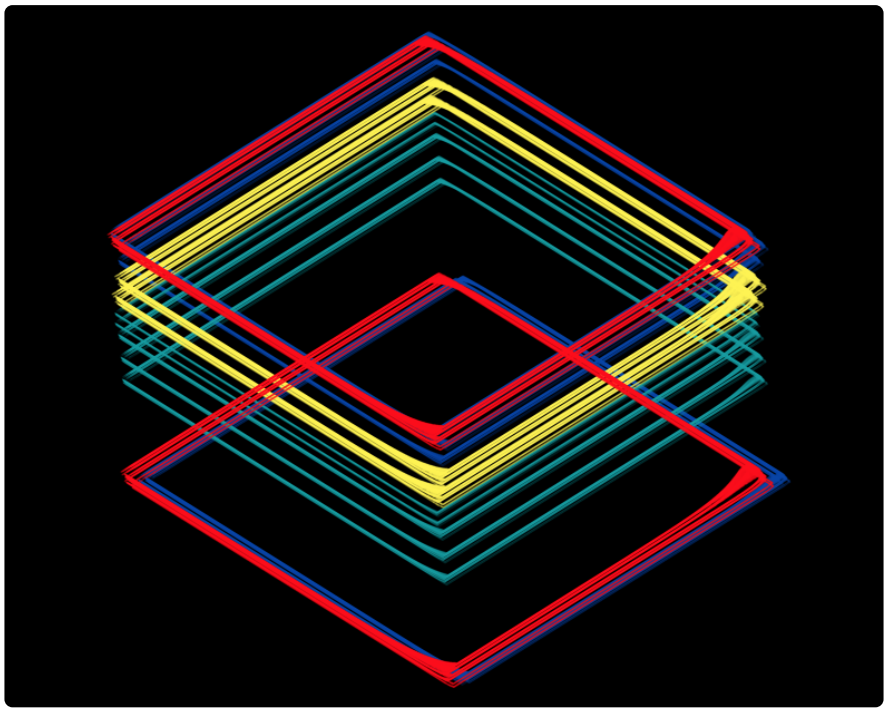
Created by John Park

https://learn.adafruit.com/mouse-painter-emulate-mice-with-makecode

Last updated on 2024-06-03 02:52:33 PM EDT

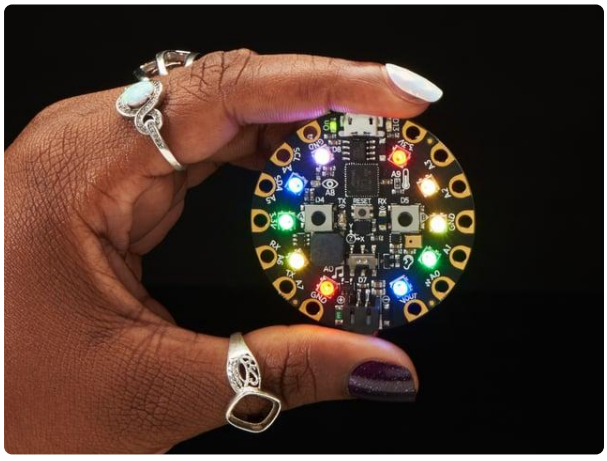
Table of Contents

Overview

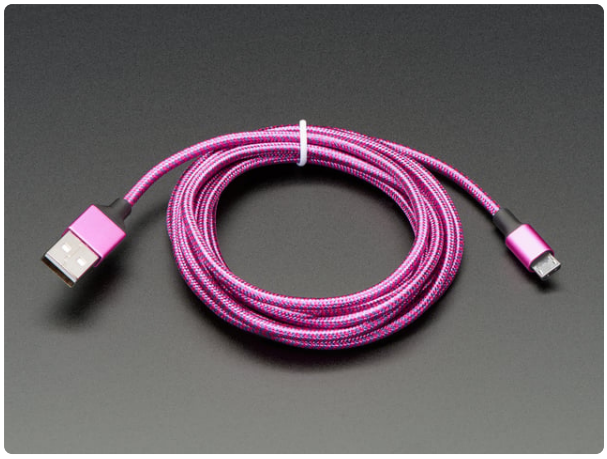


Drawing beautiful curves and perfect circles with a mouse is hard! It's been likened to trying to paint with a bar of soap. You can enlist your Circuit Playground Express coded with MakeCode to be your Mouse Painter assistant. Loaded with pre-programmed shapes, the CPX can "click" your mouse buttons and move your cursor for you to create gorgeous shape in your favorite digital paint program.

This guide will show you how to code it in MakeCode, and then you can expand on your palette of available shapes to make your own masterpiece.



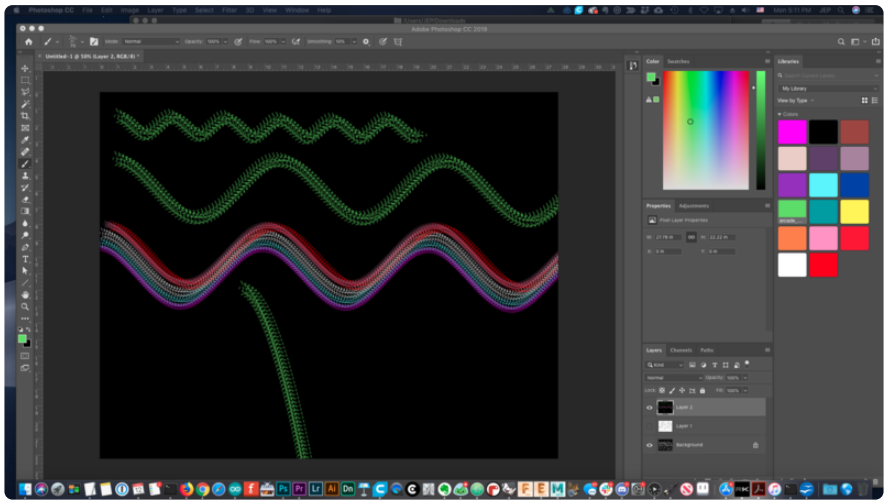
Circuit Playground Express
Circuit Playground Express is the next step towards a perfect introduction to electronics and programming. We've taken the original Circuit Playground Classic and...
https://www.adafruit.com/product/3333



Pink and Purple Braided USB A to Micro B Cable - 2 meter long

This cable is super-fashionable with a woven pink and purple Blinka-like pattern! First let's talk about the cover and over-molding. We got these in custom colors,...
https://www.adafruit.com/product/4148

Code the Mouse Painter in MakeCode



Getting Started with MakeCode

If you're new to MakeCode, head to this guide to get started (https://adafru.it/wWd). Once you're familiar with MakeCode on your Circuit Playground Express, return here.

Load the Code

Let's start by downloading the code so we can take a look at it in the MakeCode editor, as well as load it on the Circuit Playground Express to use it.

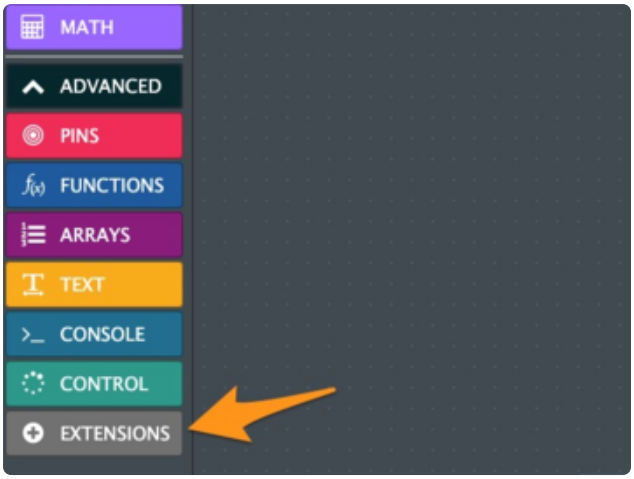
Load the code by heading to this share link (https://adafru.it/FzK). Then, click the **Edit** button to open it into the MakeCode editor.

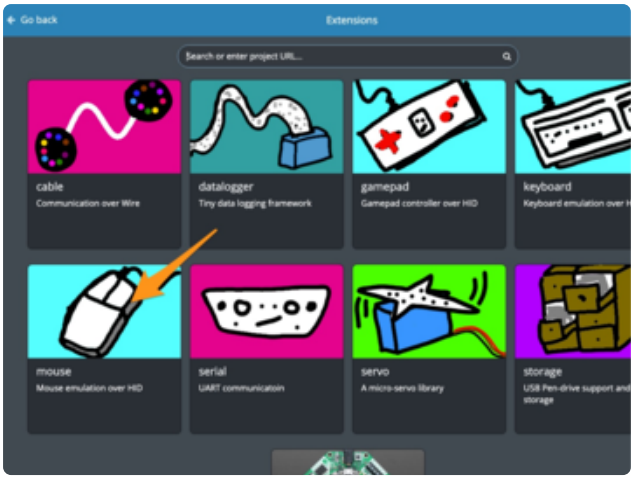
Alternately, you can download the .uf2 file linked below and import it into MakeCode by dragging it onto the MakeCode browser window.

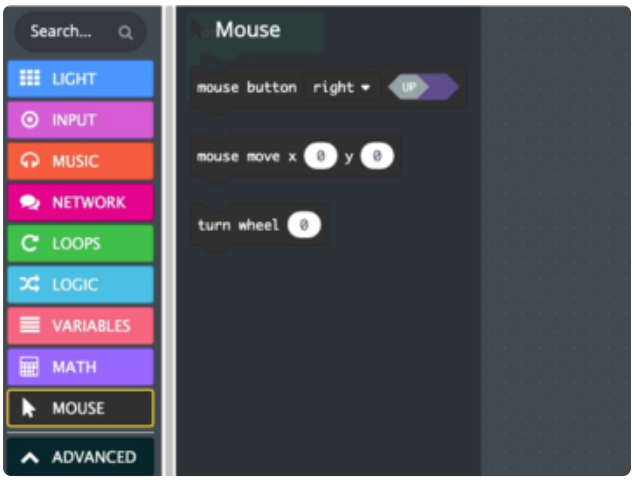
circuitplayground-Mouse_Painter.uf2

https://adafru.it/FzL

Let's have a look now at how it works.







Add Extension

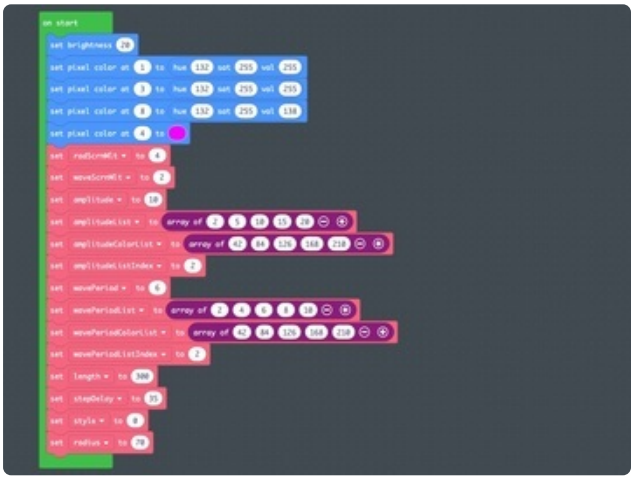
For the CPX to act as a mouse emulator, we need to add the HID Mouse (Human Interface Device) extension to MakeCode. First, click on the **Advanced** button and then click **Extensions.**

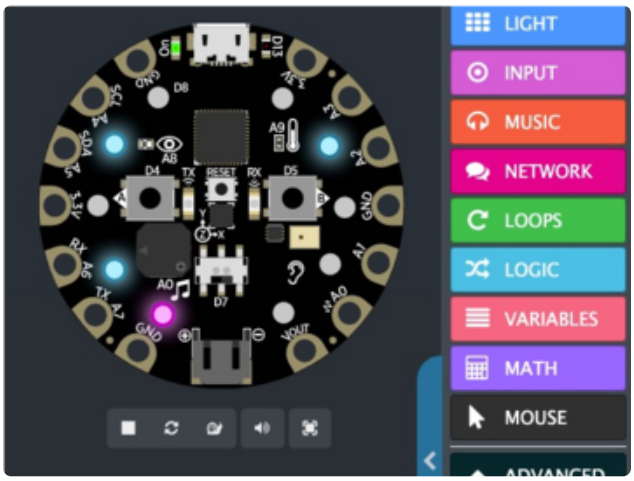
Then, click on the **mouse** extension and it will be added to your project.

You'll now see the **Mouse** category and associated blocks available.

You can now emulate three different things -- clicking the mouse buttons, moving the mouse horizontally and vertically, and turning the mouse wheel.

Mouse emulation can be used for more... nefarious uses, too! Check out the Phantom Mouse Jiggler (https://adafru.it/AWv) project to see how to prank someone with a mysterious cursor that has a mind of its own.





Setup

NeoPixels

In the **on start** block we'll turn on a few NeoPixels as indicators. These are the ones nearest the cap touch pads we'll be using to adjust parameters later, such as radius and amplitude of our shapes.

Note how we use the HSV (hue, saturation, value) color model rather than the default RGB (red, green, blue) for the three pixels that will be changing color to indicate parameter values. The HSV model is simpler to deal with when changing colors, as only a single number -- the hue -- needs to change in order to cycle through the color wheel.

The magenta pixel at the bottom will be flipped between NeoPixel 4 and 5 to indicate the position of the slide switch, and thus, which shape set we're accessing with the A & B buttons.

Variables

We also set up a large number of different variables that are used throughout the rest of the program. This is a convenient place to do so, allowing you to tweak values in a single place as you experiment with what shapes you like most!

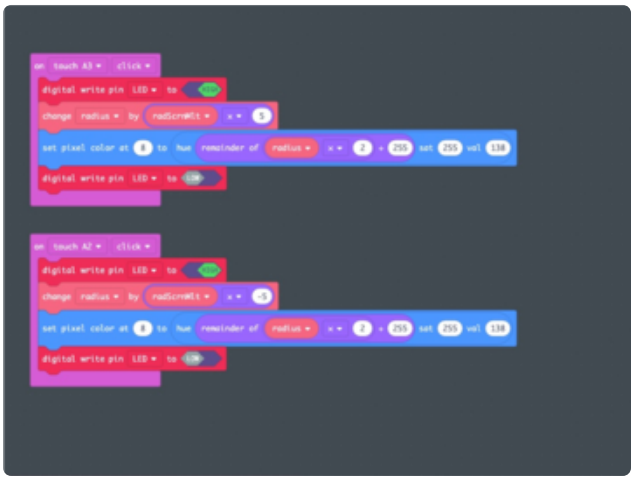
Switch and Buttons

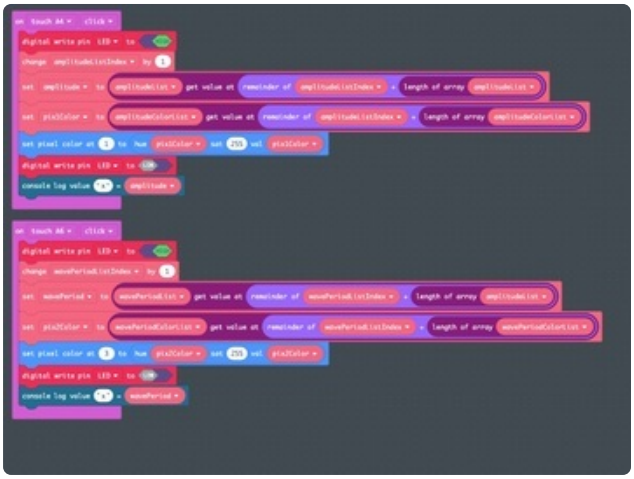
While **Switch & Buttons** would make a great title for an animated buddy cop film about an unlikely crime fighting duo of a fast talking lizard (Switch) and a highly educated cat (Buttons), that's not what's going on here. No, we are literally talking about the slide switch and two momentary buttons on the CPX.

We'll use the switch to flip between two different sets of shapes that can be drawn by toggling the **style** variable between **0** and **1.** We'll also flip the magenta NeoPixel to the corresponding position of the physical switch.

When button A is pressed, we call either the **sineWave** function or the **cosineWave** function, depending on the switch position.

Button B calls either the **circle** function or the **diamond** function.





Cap Touch

We can use the capacitive touch pads as inputs to adjust the parameters of our drawn shapes. The **A3** and **A2** pads are used to increase or decrease the radius variable that we'll use in a moment when drawing the circle and diamond shapes.

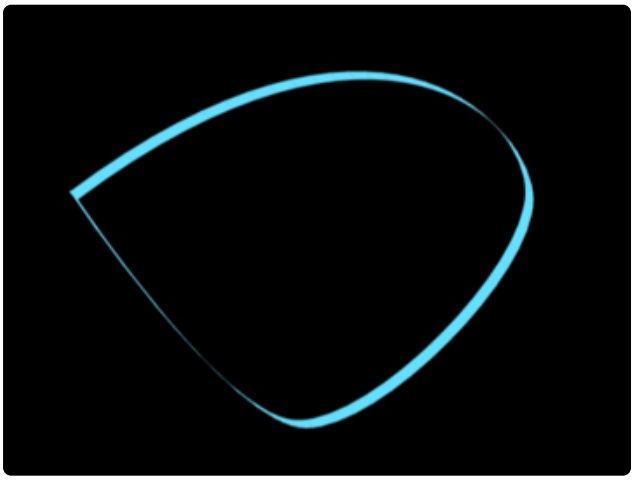
When the **on touch A3 click** block runs, first we light the on board red LED, then we increment the **radius** variable by 5 * the **radScrnMlt** variable. That variable was on we set in the **on start** loop, and it gives us a convenient place to multiply our radius values broadly in case we use the mouse painter on a system with a different screen resolution. I found that moving between computers all of my shapes seemed too small on the larger resolution monitor, so a quick tweak to the **radScrnMlt** is the answer.

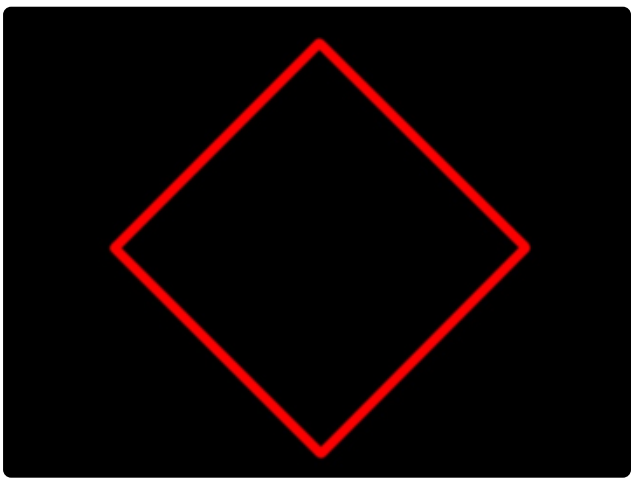
Next, we adjust the hue of the nearby NeoPixel 8. This changes along with the change in the radius variable, and by using the **remainder of __ / 255** block, we can constrain all values to within the 0-255 range of the hue parameter. In other words, the colors will cycle as we increase the radius.

Then we turn off the red LED. This all happens very quickly, so the LED appears to blink rapidly with each touch of the pad, a nice indicator that the input has been received.

The **A2** pad is used in the same way as **A3**, but to decrease the radius instead.

The **A4** and **A6** pads will be used in much the same way, but instead of incrementing values, they each cycle through an array of numbers we created in the **on start** loop. This gives five different amplitudes and five different wave periods to choose from when creating sine waves and cosine waves.





Diamond Drawing

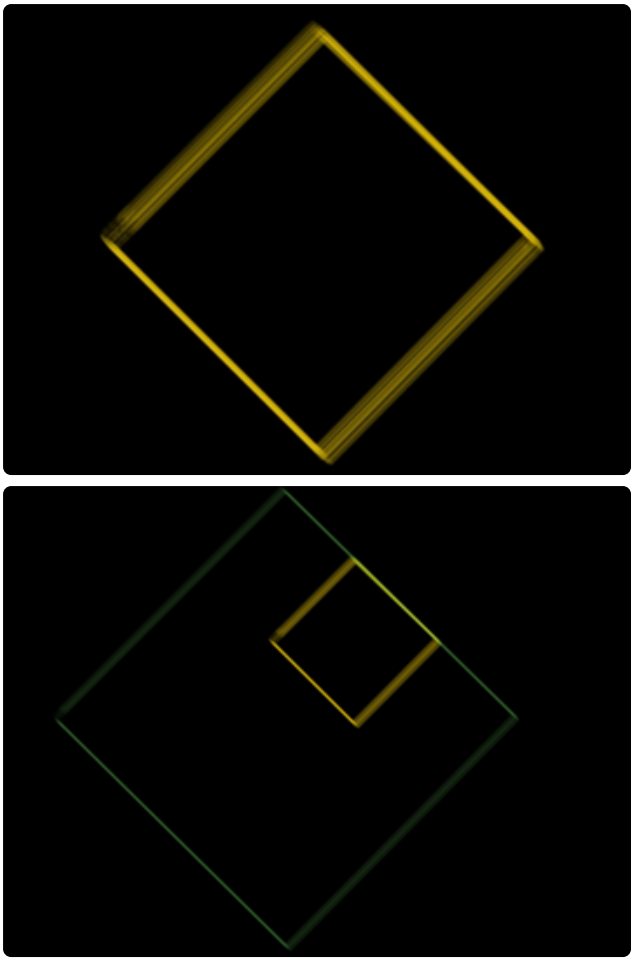
This one is the simplest one -- when the **diamond** function is called, first, the mouse button is pressed. NOTE: there is a bug currently that requires us to call the **mouse button right** when we want to click the left mouse button.

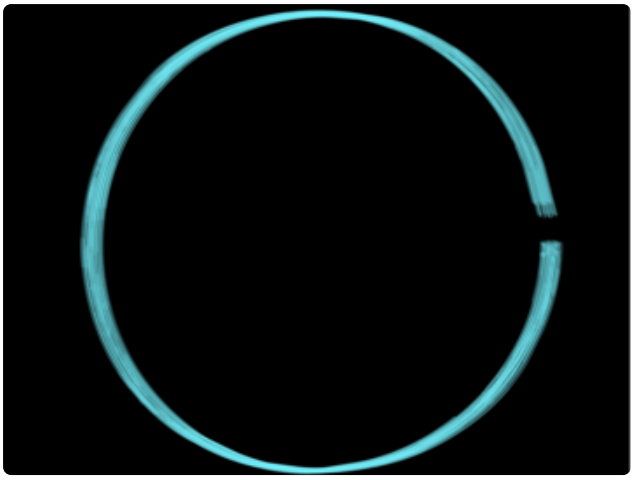
Next, we move the mouse to the right on **x** and up on **y.** Rather than hard-code a value, we use the **radius** variable (which we can change by tapping the **A3** pad) multiplied by the **radScrnMlt** number, multiplied by positive 4 on **x** and negative 4 on **y.**

We then pause half a second to allow the brush to catch up, and then repeat the process until we have drawn the diamond pattern.

Depending on brush tool settings, in some software, such as the excellent, free Sketch.io Sketchpad (https://adafru.it/FzM), this will result in a rounded diamond, while others, such as Photoshop, will create a straight sided one.

Also note how different brush tip profiles will create interesting angled/calligraphic effects. We can tap the **A3** button to change the radius for some variety as well.





Circles

Drawing a smooth circle can be very tricky by hand, particularly when using a mouse instead of a pencil! Thankfully, we can use a formula to draw one for us. Most drawing programs can create a circle, sure, and you can then pick a color to outline it. But the advantage to drawing it in realtime is that we can use angled/calligraphy brushes and other stamping/tip effects that only work when the stroke is painted in real-time.

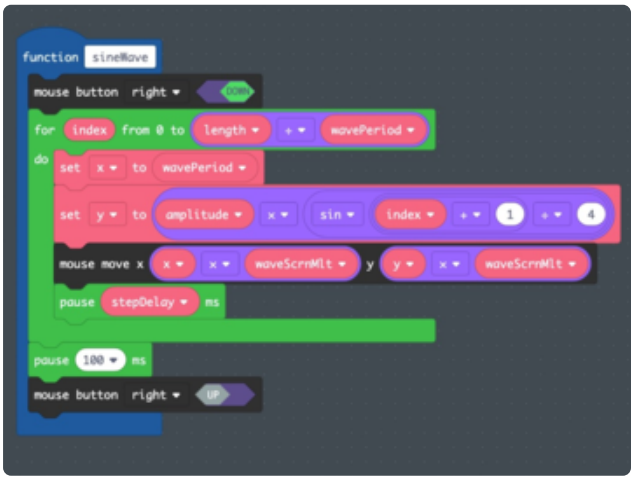
Many circle calculation formulas are for plotting the points in an absolute set. We, however, need to calculate the relative offset of the mouse position from step to step of the plotting process.

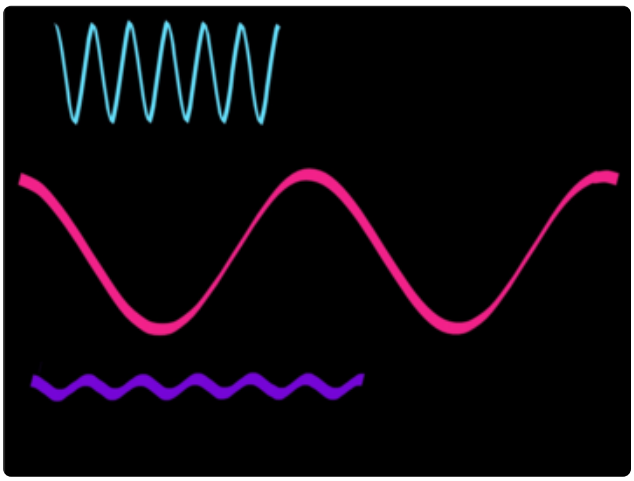
After looking at some online reference (https://adafru.it/FzN) and consulting with the much more mathematically competent Jan Goolsbey (https://adafru.it/Ckk), here's the formula I used in pseudocode, you can see the block implementation in MakeCode in the image to the left:

```
theta = 0 // starting angle (in radians) and the variable that increases with each
loop
lastX = (cos(theta) * radius) // a temporary variable that stores the previous X
value; used to calculate the difference between the current position and the
previous
lastY = (sin(theta) * radius)  // a temporary variable that stores the previous Y
value; used to calculate the difference between the current position and the
previous
step = 0.1  // amount in radians added to theta angle variable for each loop;
"walks" around the circle in a clockwise direction
repeat until theta is less than or equal to ((2 * pi) + step) { // loops from the
starting value of theta until it reaches a full circle plus one step (to close the
circle completely)
    set x to (radius * (cos)theta)) // the cosine function produces a value between
-1 and 1; multiplying by the radius scales the circle to the selected size
    set y to (radius * (sin)theta)) // the sine function...
    move mouse x to (x - lastX) // calculate the difference between the new
position's absolute x value and the previous; move the mouse incrementally from the
old position to the new position
    move mouse y to (y - lastY) // ... absolute y value ...
    pause (stepDelay / 2)ms // wait a while for the drawing application to respond
to the mouse movement
    change theta by step // get ready to plot the next point on the circle by
incrementing the theta angle value
    set lastX to x // record the new position's x value so it can be used as the
```

```
previous value during the next loop cycle
    set lastY to y // ..... y value ...
```

Note: in order to use pi we can either approximate it as 3.14159 in the Blocks mode, or head to JavaScript mode and use the built-in constant 'Math.PI' See the reference for more info: https://makecode.adafruit.com/blocks/math





Waves

A beautiful, undulating sine or cosine wave is another wonderful, yet difficult-to-draw shape.

Our formula here is a bit more simple than the circle. We loop through a certain number of times defined by the length variable divided by the wavePeriod (width) variable, and do the following:

```
set x to wavePeriod // plots the first x position
set y to (amplitude * sin((index + 1) / 4))
move mouse.x to x
move mouse.y to y
pause the stepDelay amount // allow the brush to catch up to the cursor
```

By tapping the **A4** and **A6** pads on the CPX, the amplitude and wave period may be adjusted respectively, thus creating different sine wave patterns.

Currently, the style 1 mode with the slide switch in the right position creates a cosine wave when the A button is pressed. It is very similar to the sine wave -- for extra credit, what other shape could you create instead?

Enjoy making your CPX Mouse Painter assisted digital art!