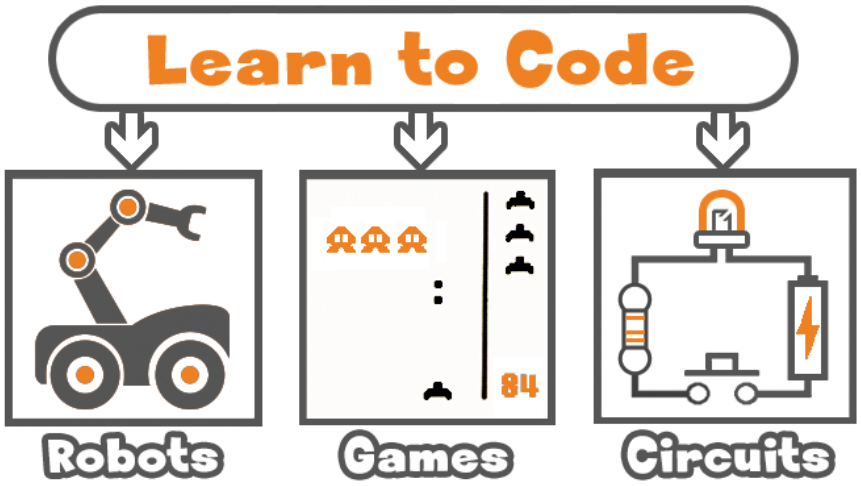
BrainPad®

PLAY | CODE | EXCEL

Microsoft MakeCode Guide

for coding

Robotics, Games, and Circuits



www.brainpad.com

CONTENTS

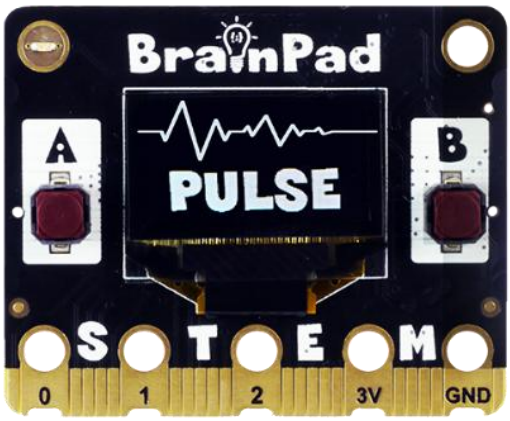
INTRODUCTION

This guide provides insight into the unlimited potential of using the BrainPad Pulse for coding robotics, games, and circuits. BrainPad works for classrooms, after-school clubs, or summer camps. The information contained within will help beginners to those already well-versed in MakeCode. It starts with the basics in a user-friendly way, allowing you to progress at your own pace and try new things as you advance your coding knowledge.

BRAINPAD PULSE

BrainPad Pulse is a Coding Micro-Computer, a powerful educational STEM tool that can be used to teach Robotics, Games, and Circuits. It works with everyone, from kids to college students and even professionals, thanks to its multiple coding options.

The BrainPad Pulse is the “brains” of your project. It can be used on its own, with its display, accelerometer, buzzer, and buttons.



And it can connect to other accessories to give you more options, like the BrainBot.

There is a range of available accessories as well, https://www.brainpad.com/accessories/.

GLOBAL THINKING

We work hard on making the BrainPad available to every culture and language. If you feel up to the challenge and want to translate this book into your language, please let us know!

If you are translating this book, please add a section about you.

LICENSE

STEM

"STEM" stands for Science, Technology, Engineering, and Mathematics. The STEM acronym was introduced in 2001 by the U.S. National Science Foundation. STEM is now one of the most talked about topics in education. STEM education, however, is more than just these four fields of study. The STEM educational approach is aimed at connecting classroom learning to the real world by emphasizing communication, collaboration, critical thinking, and creativity while teaching the engineering design process. The term "STEAM" is the same as STEM with additional emphasis on the arts.

THE PHILOSOPHY

STEM education requires an evolving platform, not a toy. The BrainPad evolves to match your skill level. Many toys are marketed as STEM tools, but most lack the versatility to keep students engaged for any length of time. The BrainPad ecosystem consolidates programming (from beginner to professional), electronics, engineering, robotics, and more into an inexpensive platform that can be used from grade school through college. It is backed by our 15 years of professional engineering experience. The same tools commercial customers use to program our industrial controllers can also be used to program the BrainPad Pulse. No other STEM platform can lead students from drag and drop block programming to professional programming and engineering like the BrainPad Pulse.

MAKECODE

The BrainPad Pulse provides multiple coding options, including Python and .NET C#. It provides Python through Thonny IDE and provides .NET C# through Microsoft Visual Studio. Those coding options are for advanced users and require a PC Windows machine. This guide will not cover these topics, but there are plenty of lessons available at https://www.brainpad.com .

The focus in this guide will be on MakeCode, an online coding editor.

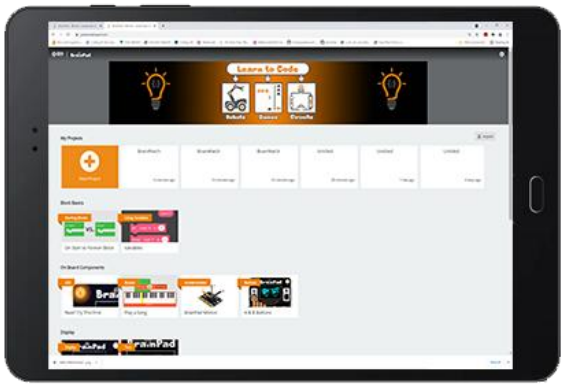
NOTHING TO INSTALL

When using MakeCode, there is nothing to install -- everything works through an Internet browser. All that is needed is an Internet connection.

WORKS ON EVERYTHING

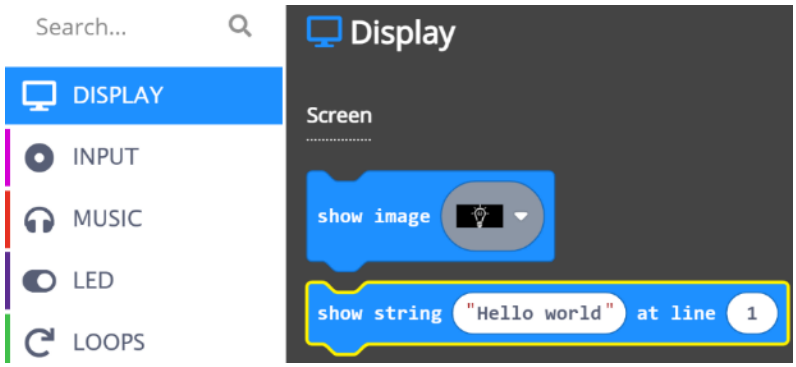
Any system with a modern browser can use MakeCode. This includes Windows, Mac and Chromebook machines.

MakeCode will even work on phones and tablets, however this is not recommended.



BLOCKS TO CODE

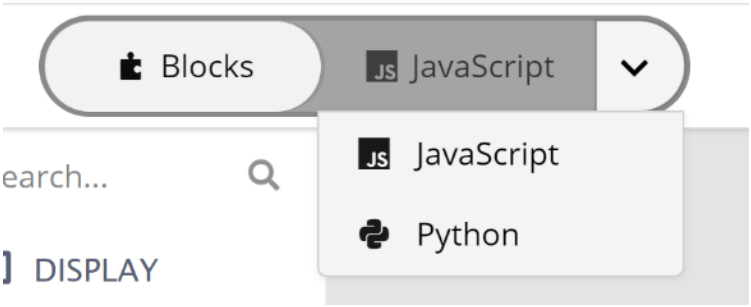
Users can code the BrainPad by arranging blocks. This is user friendly and works with younger learners.



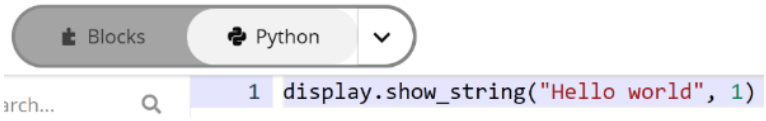
The Display menu, for example, has a “show string” block.

Dragging the “show image” block to the “on start” block creates a program that shows an image on the screen.

Coders have the option to type the code by switching to JavaScript or Python.



This changes the coding surface from blocks to typed coding. The earlier blocks that showed the string will look like this in Python.



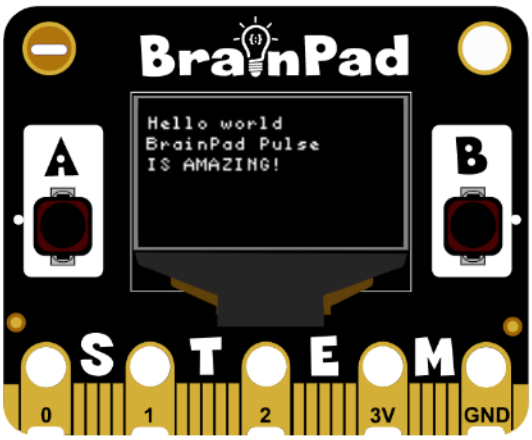
We can now type a few lines of code…

```
1 display.show_string("Hello world", 1)
2 display.show_string("BrainPad Pulse", 2)
3 display.show_string("IS AMAZING!", 3)
```

… and we would be coding in Python!

THE SIMULATOR

The MakeCode simulator is basically a virtual BrainPad Pulse. The simulator allows you to try programs right in your browser without loading the program onto the BrainPad Pulse. This is also an excellent try-before-you-buy option if you haven't received your BrainPad Pulse yet.



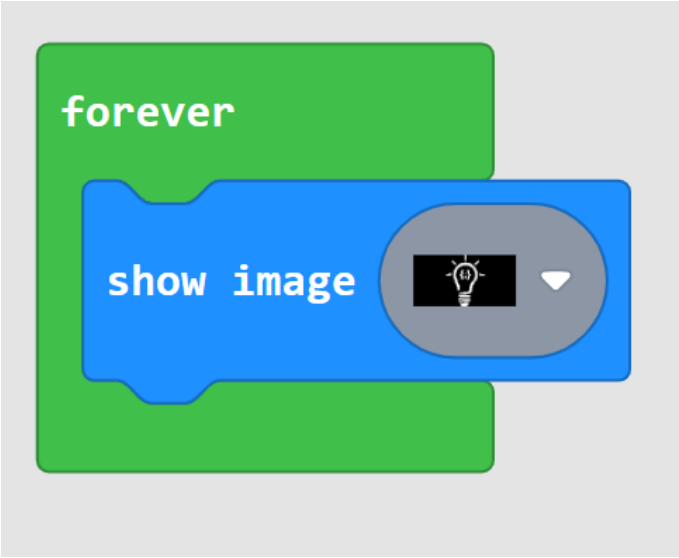
The simulator is great for getting started and for quick tests of code snippets; however, it does not replace the need for hardware, especially when starting to connect the BrainPad Pulse to other circuits.

GETTING STARTED

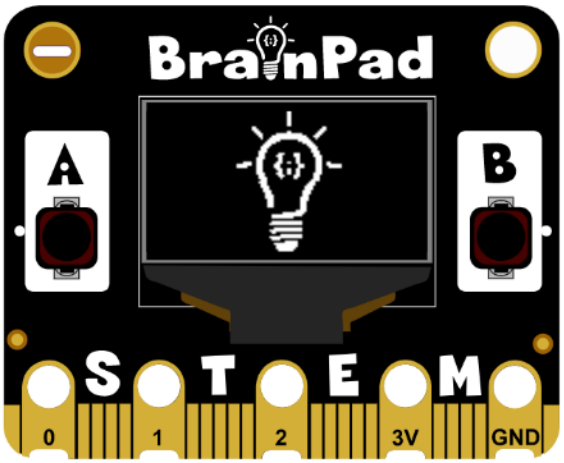
Enough talking – let's start having fun and making some projects! We will be using Microsoft MakeCode, so head on to https://makecode.brainpad.com/. The website includes a lot of material and projects, but we will just dig right into starting our own project. Go ahead and click the New Project button.

FOREVER LOOP

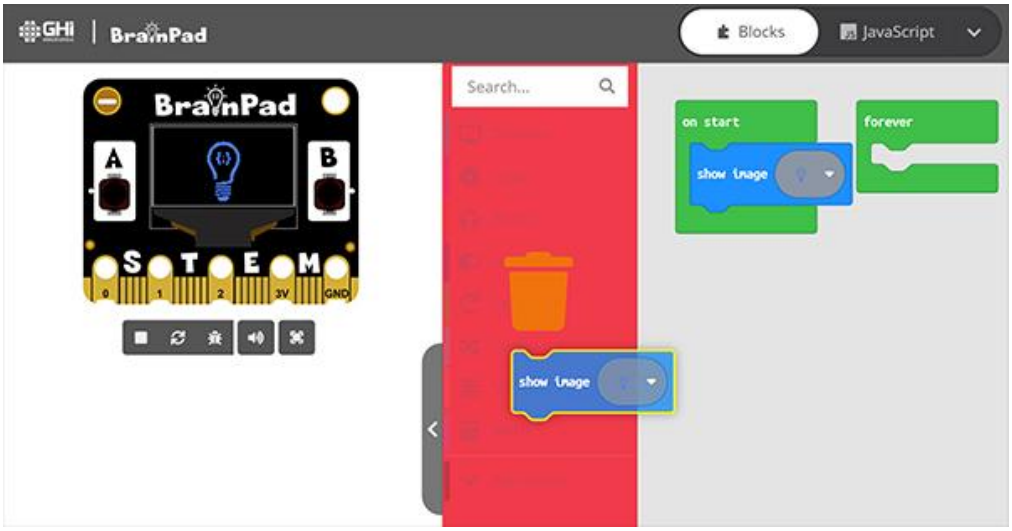
Now, from under the Display option in the menu, drag the block “show image” and place it into the “forever” block. The forever block is used for any code that you want running all the time, forever! The instructions within the forever block will always execute while the BrainPad has power. Once all the instructions have executed, the BrainPad runs the code again starting with the first instruction in the forever block. This is known as an infinite loop and is used often in computer programming.



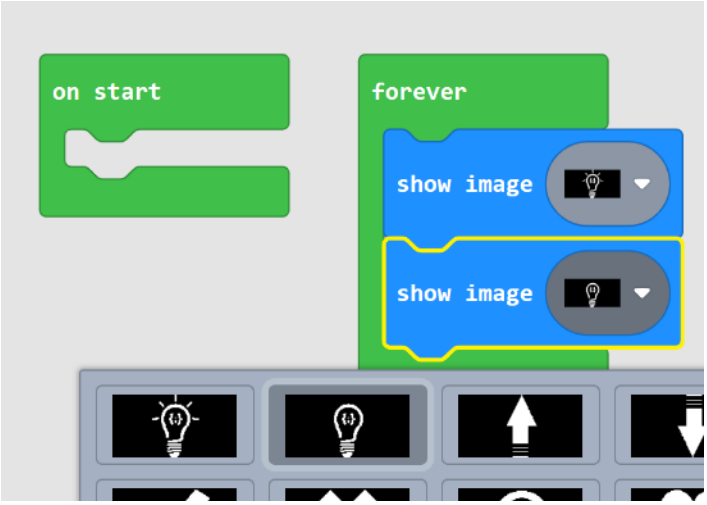
The simulator will now show the lightbulb!



If you dragged a block that you do not need, you can trash it by dragging it back into the block menu.



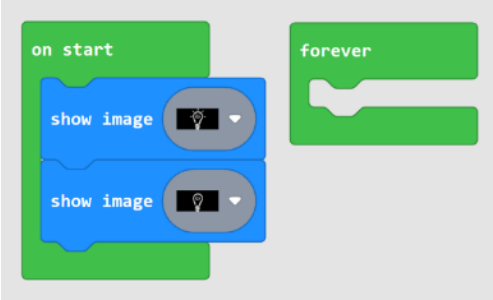
Repeat the same step, but in the second block click on the image and change it to something else, like the light bulb without the lines.



Do you see a “flashing” lightbulb on the display?

ON START

The forever loop keeps on repeating forever, but what if we need something to run one time? This is where the “on start” block comes in handy. Whatever is in it will happen only once. Move the earlier blocks from “forever” to “on start”. By the way if you drag the top block, you will automatically drag all blocks under it.

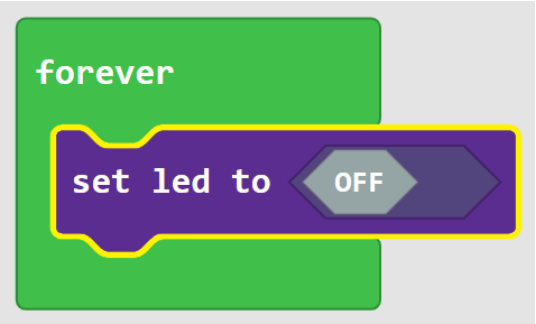


The Lightbulb will now show with lines very quickly and then the lines will go away, but the light bulb will still be on the display unchanged… forever!

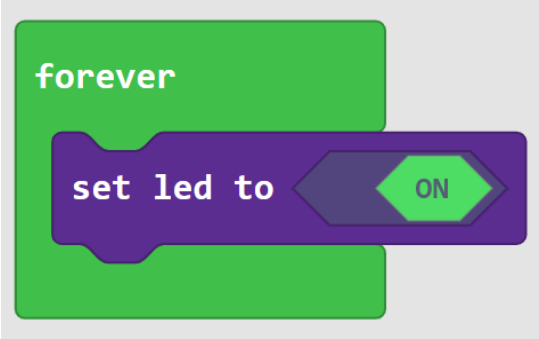
DEBUGGING

Debugging is the process of looking for a bug in the code. A “bug”? When writing a program that the computer understands and runs, the outcome may not be what we expected. This is a called a bug. Debugging is the process of analyzing the code to find and correct the unexpected outcome. For example, computers run fast and a way to debug is in stepping through code to see what is happening. Computers are fast. How fast you ask? A lot faster than you think! Let us show you how fast.

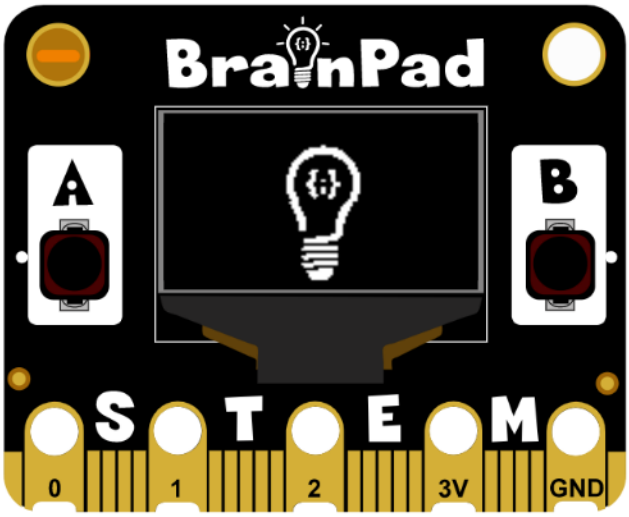
From under LED, drag “set led to” to the “forever” loop.



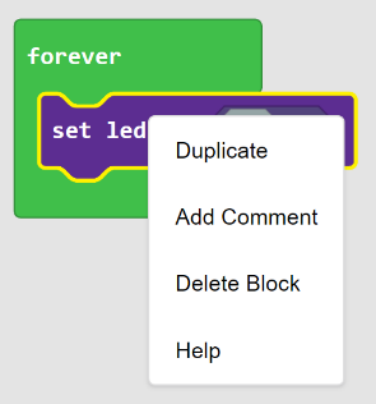
Click on the “off” switch to change it to “on”.



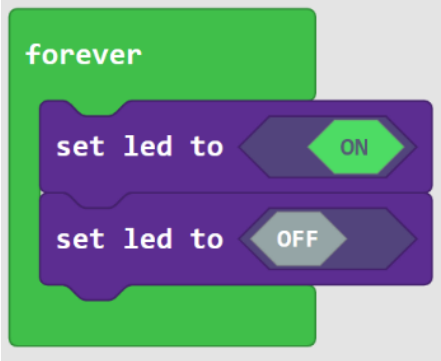
The simulator now shows a lit LED. It is the orange line in the top left corner of BrainPad Pulse.



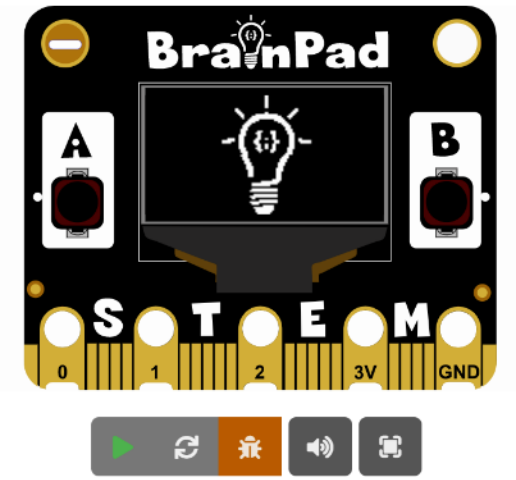
Now add a second “set led to”, but instead of dragging it in, we will right-click the existing one and then “duplicate”.



Keep the second block to “off”.

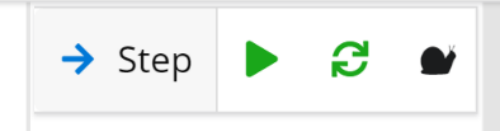


This forever loop will now turn the LED on and off forever, blinking the LED. But the LED on the simulator is not blinking, is it? Well, I promise you it is blinking, but you are just not seeing it. It may flicker but will not blink properly. This is because the loop is running just too fast for our eyes to see it. This will be true if we were to run this loop on the circuit. This is an example of where debugging comes in handy.

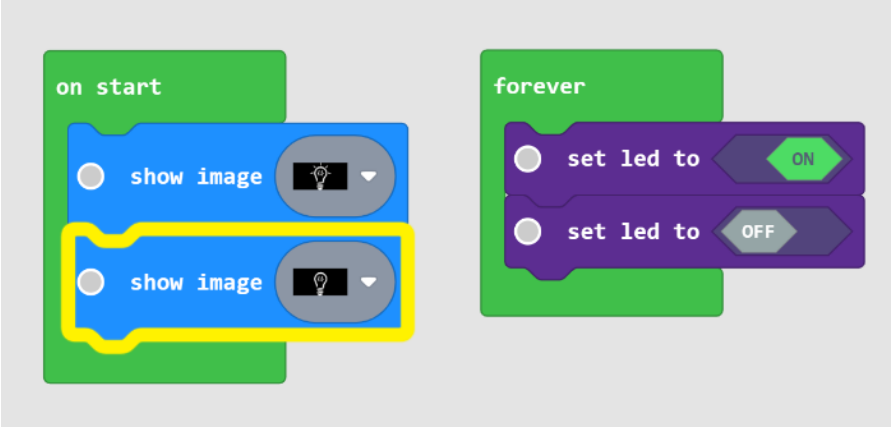


Click on the little bug in the simulator.

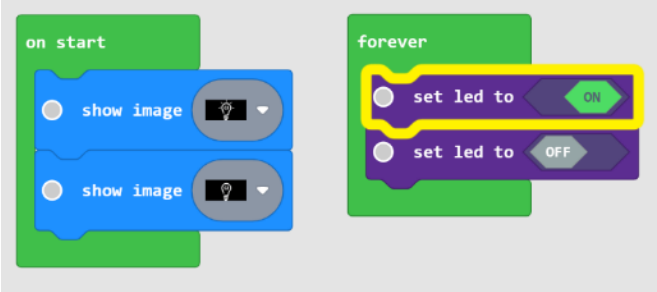
The block menu will change to the debugging menu.



The debugger will now run the first block and a yellow line will highlight the next block to run. At this point, the display will show the lightbulb with lines.



Click on the Step button and a yellow line will highlight the next block. The display will now have a lightbulb without lines. Keep clicking the step to see the yellow line going back and forth in the forever loop that turns the LED on and off.

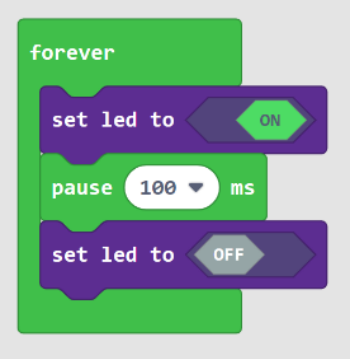


The LED on the simulator will work just fine, matching the blocks. Have we just found the bug? The problem in this case is simply speed. The LED is blinking, but it is too fast for our eyes to see.

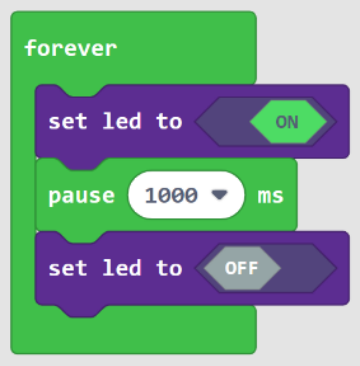
Click the “bug” icon again to exit the debugger and the LED will not blink, just like before. Don’t you wish there is a way to “pause” the program for some time?

PAUSE

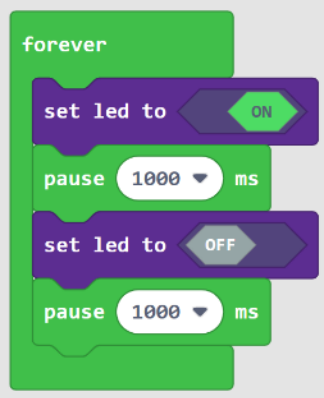
In some cases, it is desired to pause the system for some time, just like in the previous blinking LED example. From under loops, drag in the “pause” block. The value given to the block is in milliseconds. In case you did not know, 100ms is tenth of a second.



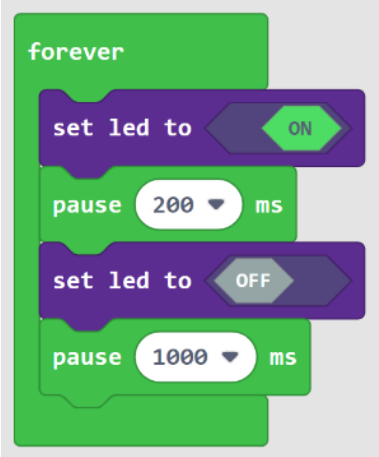
Change the delay to 1 second, which is 1000ms.



The LED is starting to flicker on the simulator, but it is still not blinking. Let’s analyze the forever loop. An LED is turned on, a pause stops the program for 1sec, so the LED will be lit for a whole second. Then a block turns the LED off. But immediately after that, the forever loop will rerun the LED on block. The LED was, in fact, off but for a very short time. Let us add a second pause.



The LED is now coming on for one second and off for one second. Change the delays to different values and observe the simulator. I will set mine on for 200ms and off for 1sec.



FLASHING THE BRAINPAD

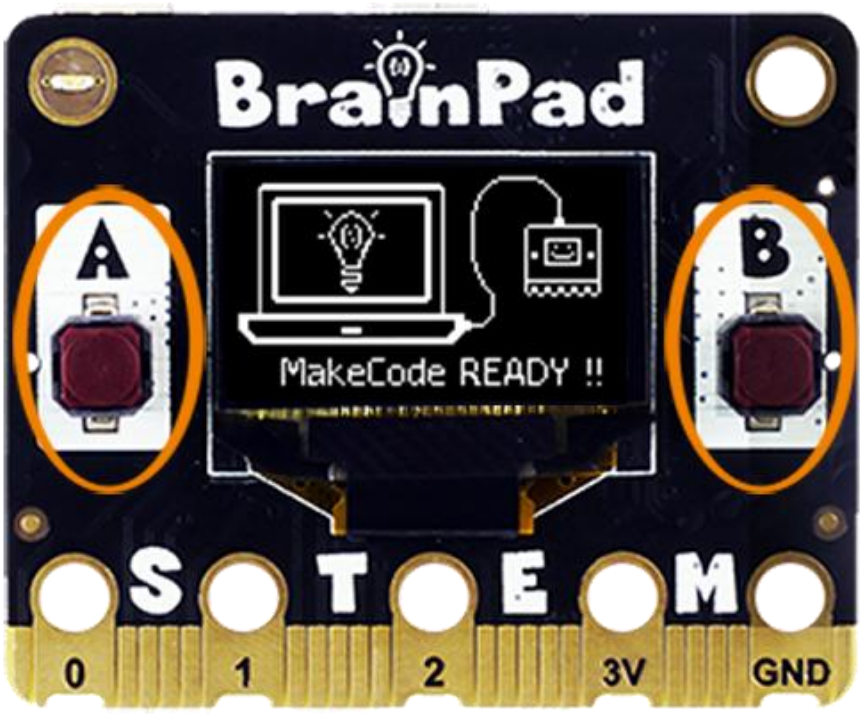
The process of copying a program onto the BrainPad Pulse is easy but can be tricky the first time you do it. If this section is still not clear, there is a video on the MakeCode intro lesson to show you how to flash your board. https://www.brainpad.com/lessons/makecode-intro/

First, you must connect the BrainPad Pulse to your computer using a Micro-USB cable. By the way, you can use Windows, a Chromebook, a Mac, or almost anything with a modern web browser and a USB port.

A Micro-USB cable should have been included with your BrainPad Pulse. You might also have an extra one at home. If you don't have one, they are readily available. Micro-USB cables are even sold at most gas stations!

To load a new MakeCode program onto the BrainPad Pulse, it needs to be put in the "**MakeCode READY!!**" mode.

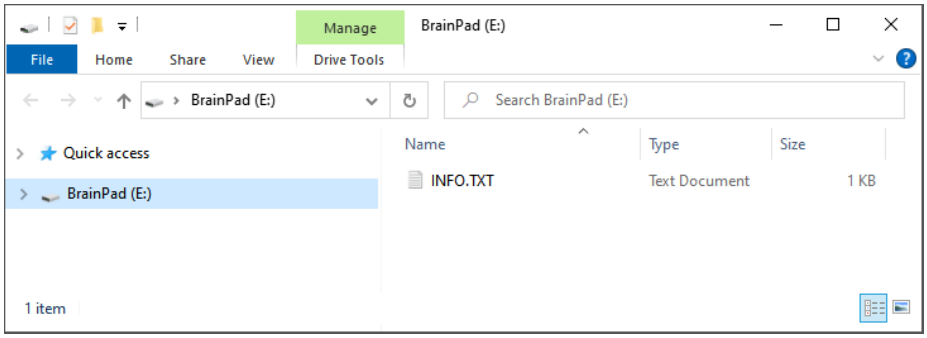
To put the BrainPad Pulse in the "**MakeCode READY!!**" mode, press and hold both A and B buttons for about 3 seconds, until you see the "**MakeCode READY!!**" image.



A & B Buttons

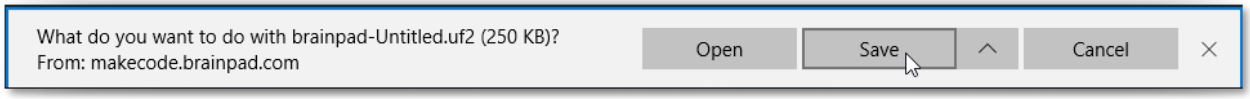
If you hold A and B buttons for 5 seconds and screen does not show the "**MakeCode READY!!**" image, then your board is loaded/prepped to do other languages, like Python or .NET C#. Fear not, you can change it back to MakeCode as shown on the MakeCode Setup lesson https://www.brainpad.com/lessons/makecode-setup/ .

When the BrainPad is ready to accept MakeCode programs, the connected PC will detect a storage device called BrainPad, similar to a storage device when you plug in a USB memory stick.

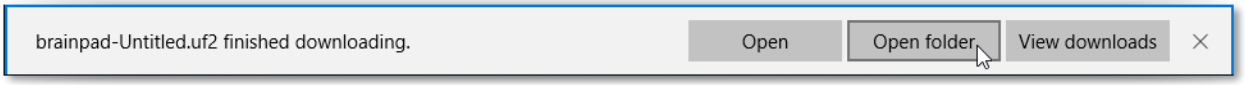


Go back to the browser with the program we made earlier. Click the download button and save the file on your computer.

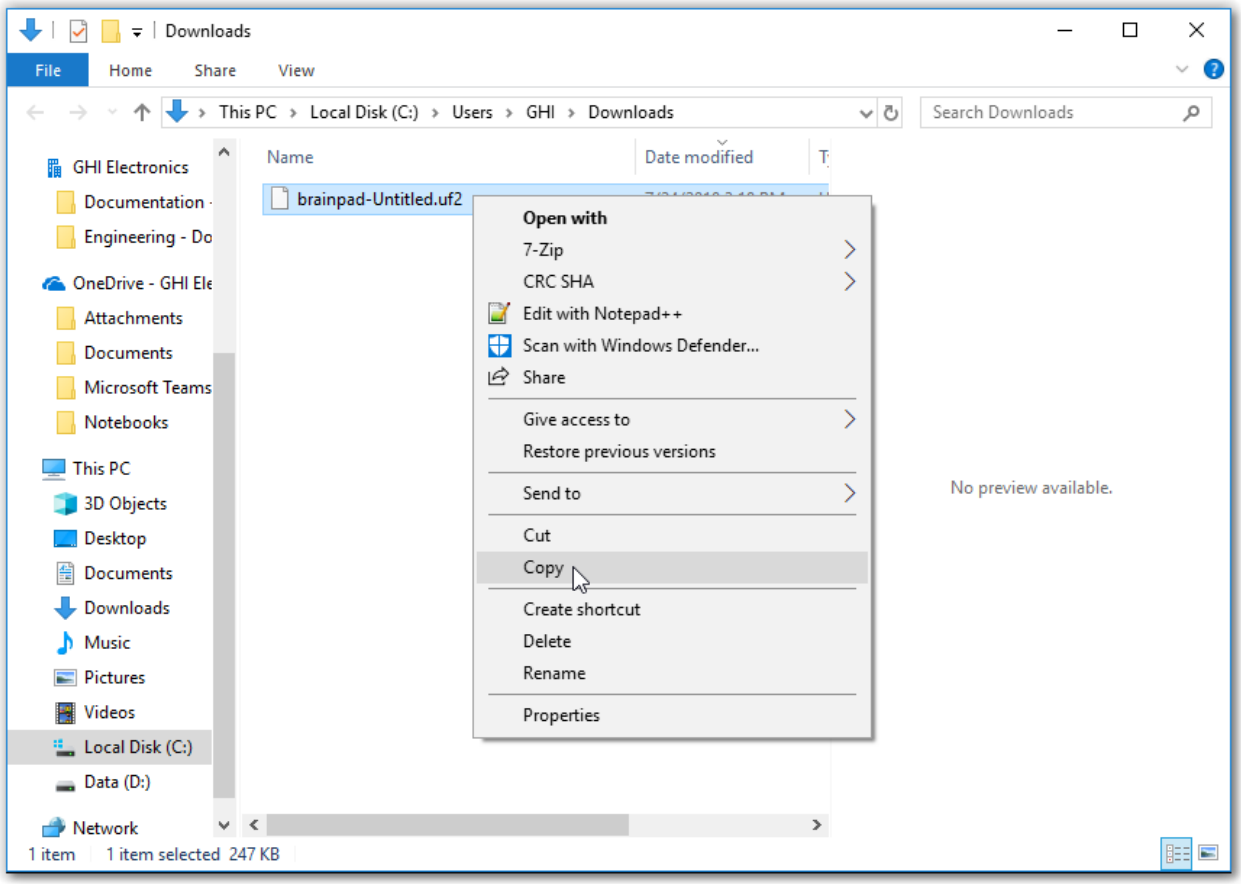
If you are using Microsoft Edge, you should see a dialog box like the one shown below:



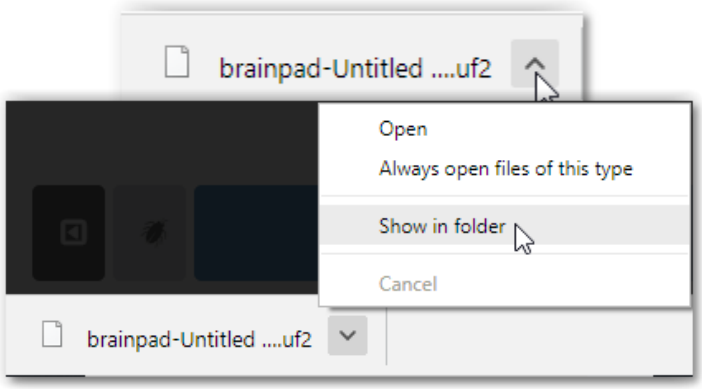
Click the Save button and then the “Open folder” button in the next dialog box to show the downloaded file.



The file will be highlighted. You can right click on the file to copy it.



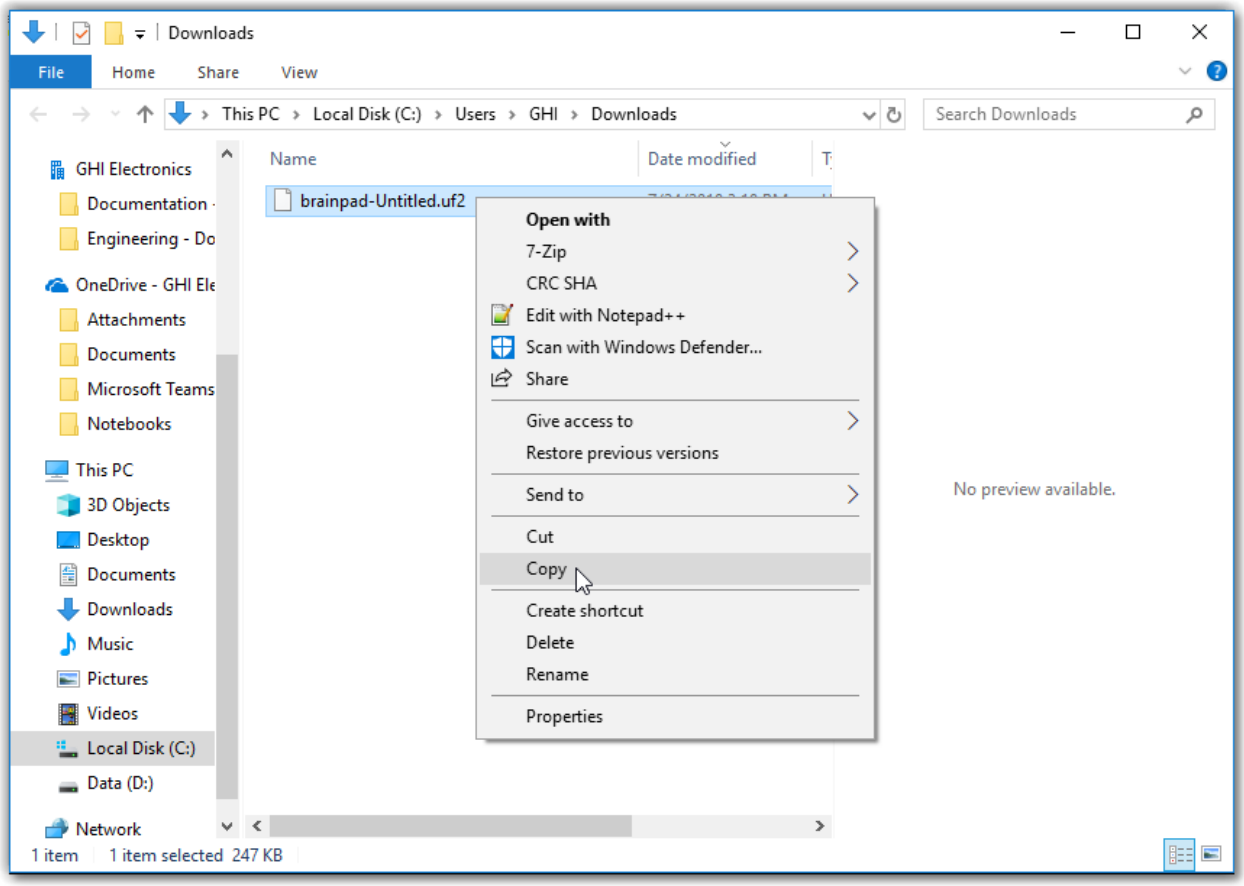
If you are using the Chrome browser, the file will be shown at the lower left corner of the screen:



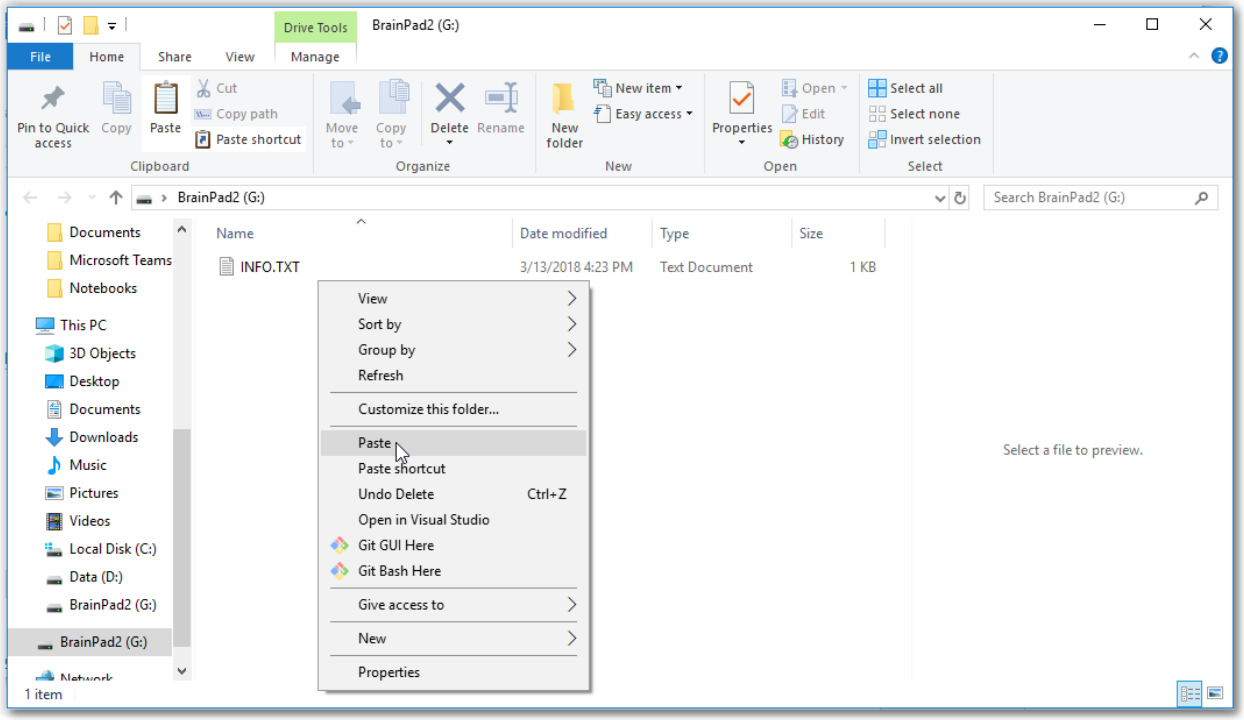
Click on the small up arrow and then select “Show in folder.”

The downloaded file will be highlighted. You can right click on the file to copy it.

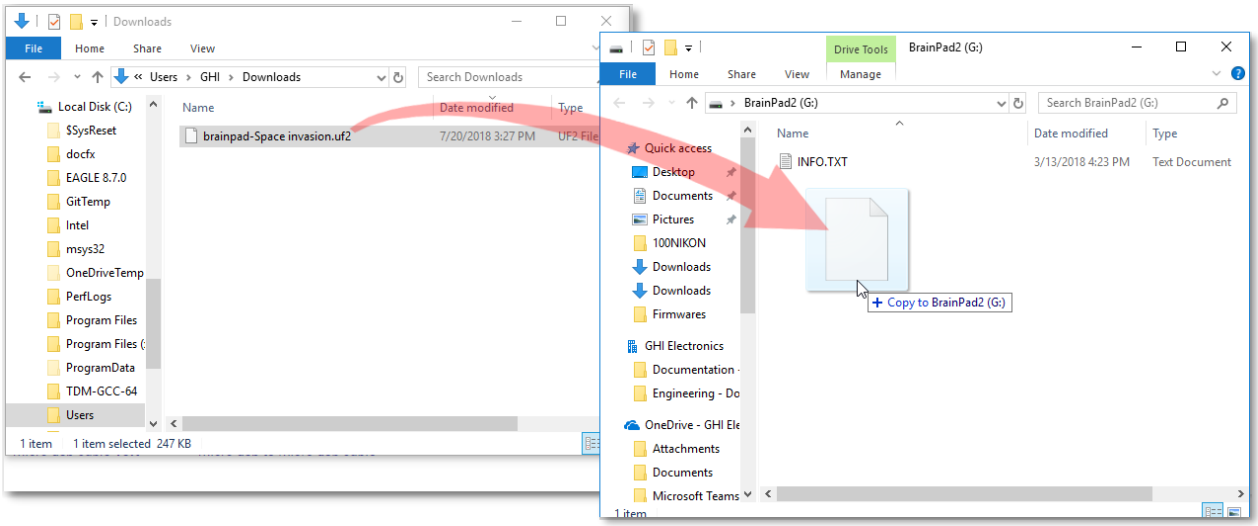
Note that browsers will not delete the old files when you download again. Instead, a number will be added to the filename (for example “brainpad-Untitled (1).uf2”). Make sure you copy the latest file. The latest file will be the highlighted file.

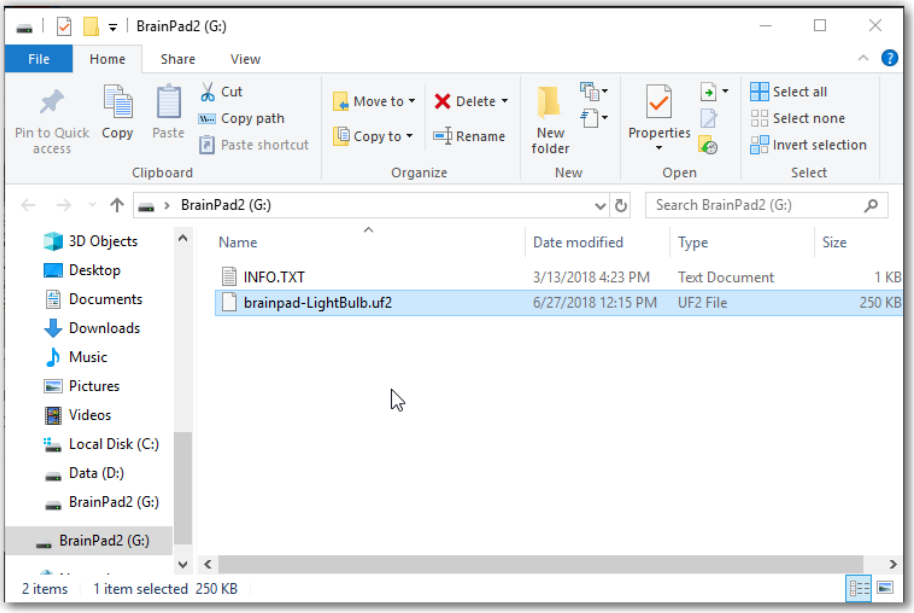


We can now paste the file we have downloaded into the BrainPad.

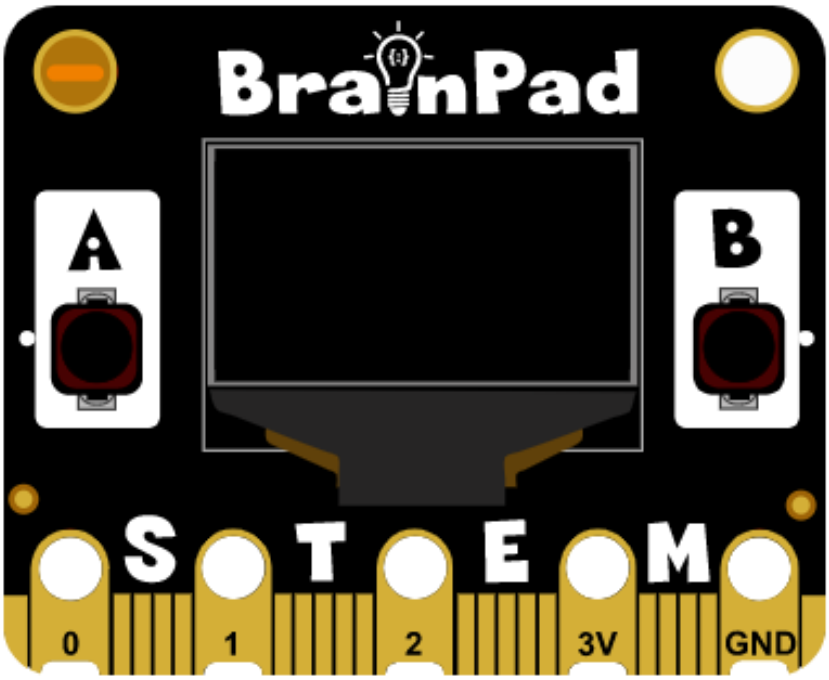


You can also drag and drop the downloaded file into the BrainPad window.





Once the file is done copying to the BrainPad Pulse, it will reset and execute the file you have just loaded. This will be our program from earlier that shows the light bulb on the screen “on start” and then blinks the LED “forever”.



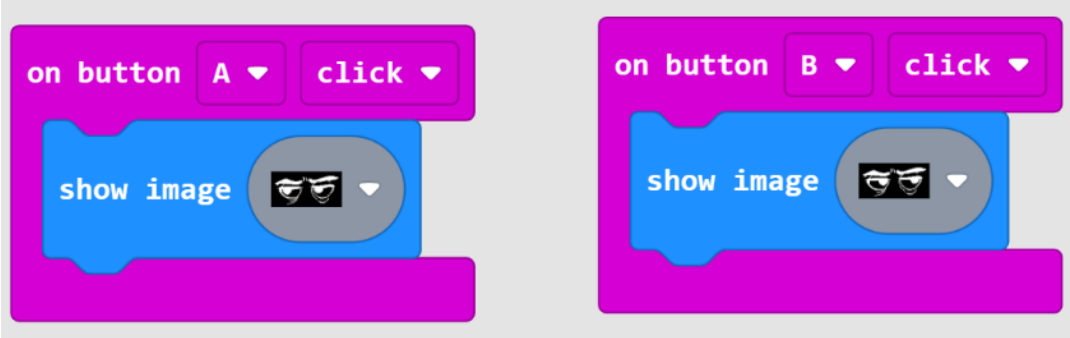
BLOCK BASICS

This chapter will highlight the BrainPad onboard components and at the same time show different block types to add variables and control the program's flow.

EVENTS WITH BUTTONS

There are two ways to check if a button is pressed. The first way is to sit in a loop and keep on checking the current status of the button. The second way is done by letting the system internally keep checking for us and let us know when the button is pressed. The system "lets us know" through an event. Think of the "on start" block as an event block that gets called when the system starts, meaning the event is triggered on start. Similarly, a block can be triggered "on button". However, the "on button" block needs to know what button and when to trigger, like when the button is clicked. Those blocks that get triggered are referred to as events.

From Inputs, add the "on button" block. Duplicate the block and modify one to be button A and one to be button B. Then add a "show image" to each block. Use any 2 images you like. We will use the eyes that are looking left with the A button and the eyes that are looking right with the B button.

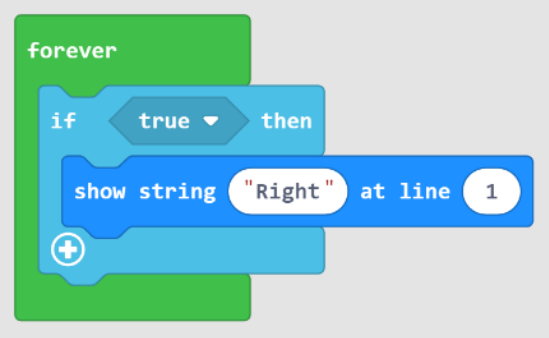


At first, nothing will show on the screen as we have not added anything to on start block. Now try the A and B buttons. Pressing each one of the buttons will show its image on the screen.

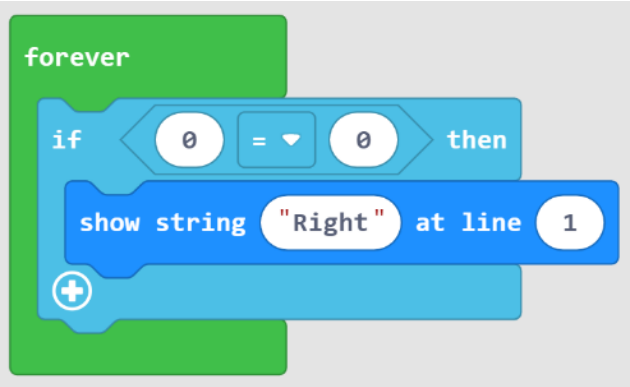
IF AND MOVEMENT

This section will cover the if statement to check for a condition. Or to compare values. We will use the accelerometer to measure tilt, which is measuring the earth's gravity. If the value is more than a certain number, the screen will show right and if it is less than a value, it will show left. Somewhere in the middle, it will show flat.

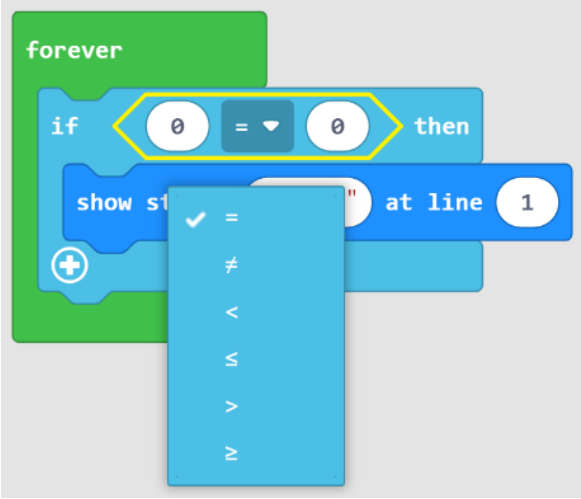
We are not using an event here like we did with the buttons. We will need to continuously check the accelerometer value. We will do so using the if statement, found under logic. We then add "show string" from the display section and change the string to say "Right".



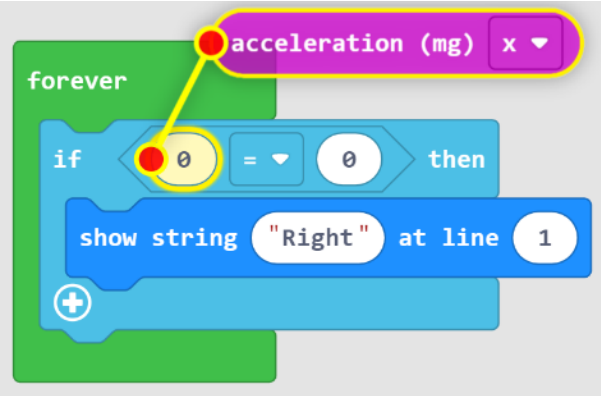
The display will now show Right, but why? The if statement checks for some condition to be true or false. In this case, the if statement is defaulted to true. We will need to change it to a condition that compares two values. The comparison blocks are also found under logic. Add the comparison block to replace the default true.



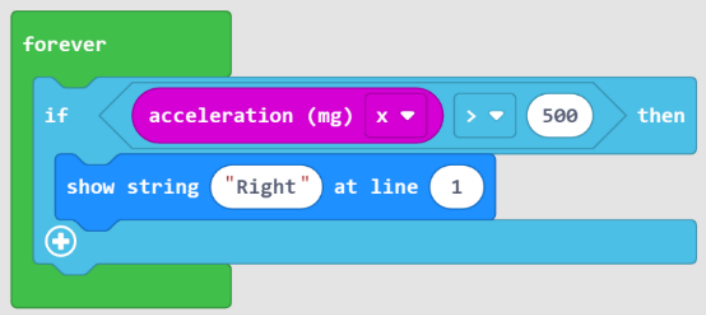
This block has the option to compare less than, larger than, equal to and more.



Under inputs, we can bring in the acceleration and add to the comparison.

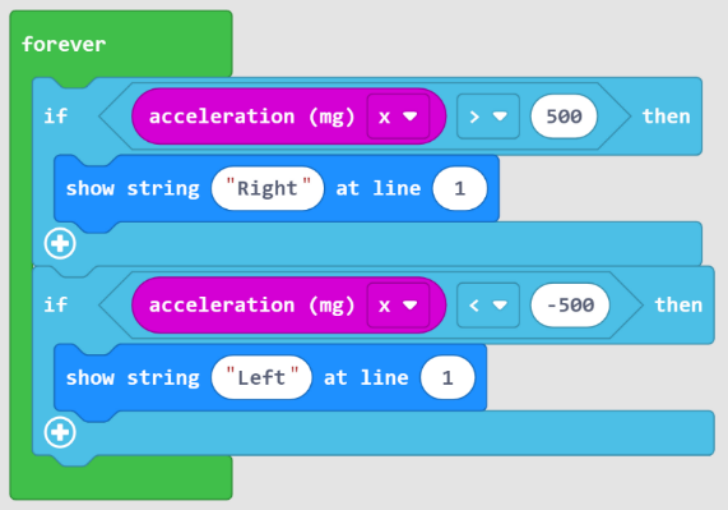


MakeCode will help by showing a line to an appropriate spot for the block. The acceleration outputs a value that is maxed at 1023 in one direction and -1023 on the other direction. The output will be zero right in the middle. Let's compare the value to be more than 500.



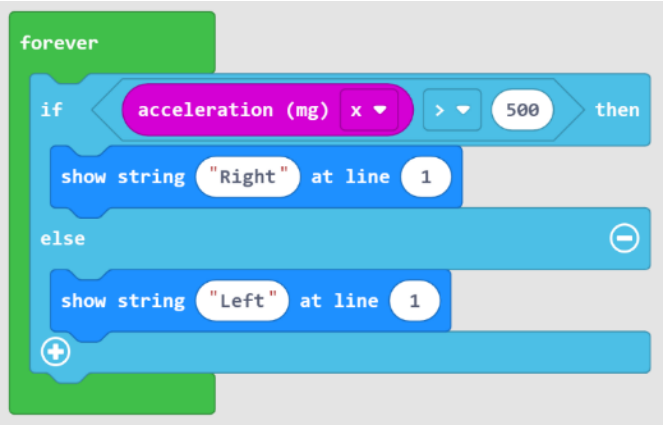
The display should not show anything by default, but then tilting it to the right will show "Right" on the screen.

Duplicate the same block and change it to show left when the value is less than -500.

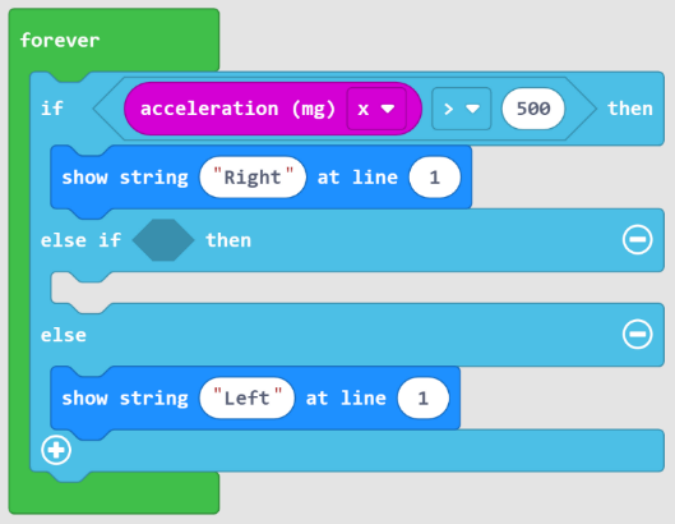


That is two completely separate if statements (if blocks). Like this example, it is best to use the else statement that goes hand in hand with the if statement. It simply means, if something is true do this, or else do that.

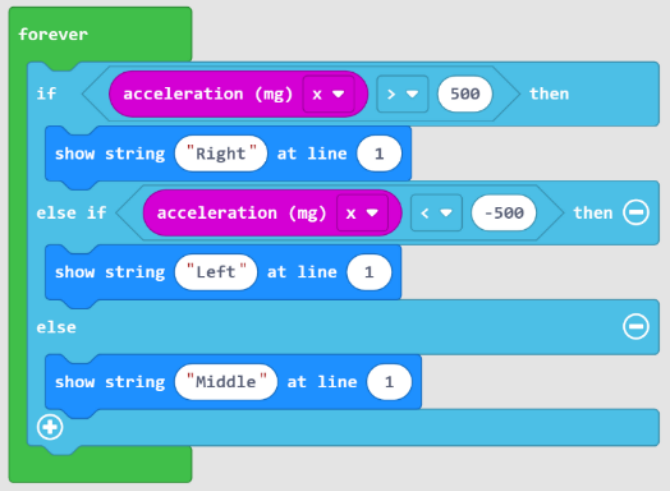
Remove one of the if statements and only keep one of them. Now click the plus sign at the bottom of the if statement. We can now add “show string” under “else” to print Left when the value is less than 500 and Right when the value is more than 500. Not exactly what we need just yet, but let’s try it.



What we need is to show left when the value is less than -500. And we also want to show “middle” when the value is between 500 and -500. We can click the plus sign again, the one at the bottom of the block. And now we have room for another “else if” option.



We now can finish the block to do exactly what we have set to accomplish.



VARIABLE SOUND

How high of a frequency can you hear? Do you know the older we get the lower our top hearing frequency will be? We will make a device to check out top hearing frequency. We will use a variable to keep track of the frequency.

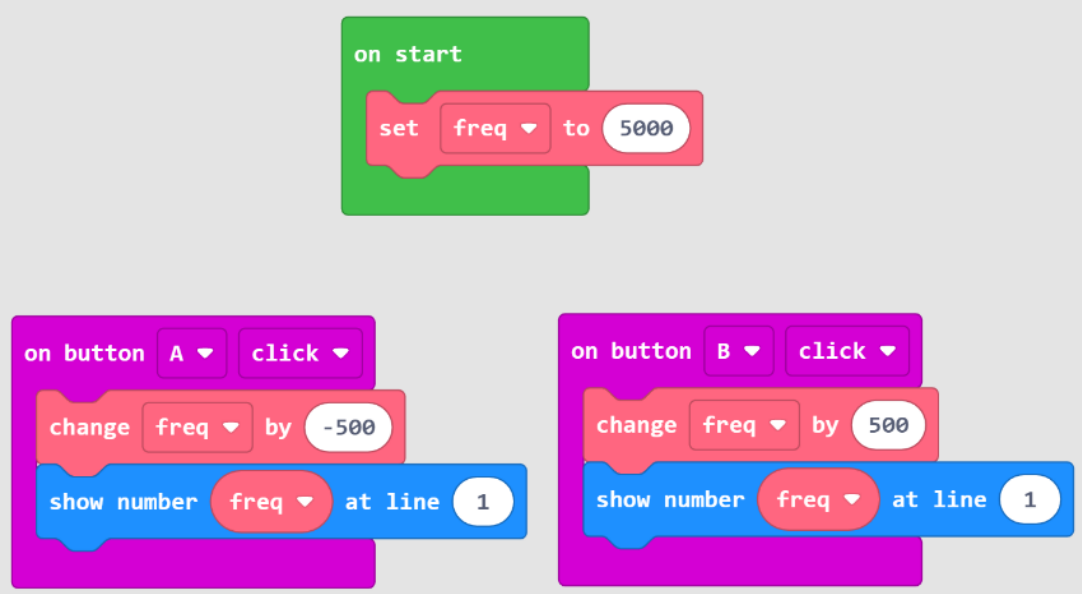
A variable is used to hold a value that will be used later. We can use the variable in our code. For example, we can print that variable or use it to set the frequency of the sound. We can also change the value of the variable. We can set it to something completely different or increase/decrease the variable.

This example will use the “on button” event we have used earlier. Pressing A will decrease the frequency variable and pressing B will increase the frequency variable.

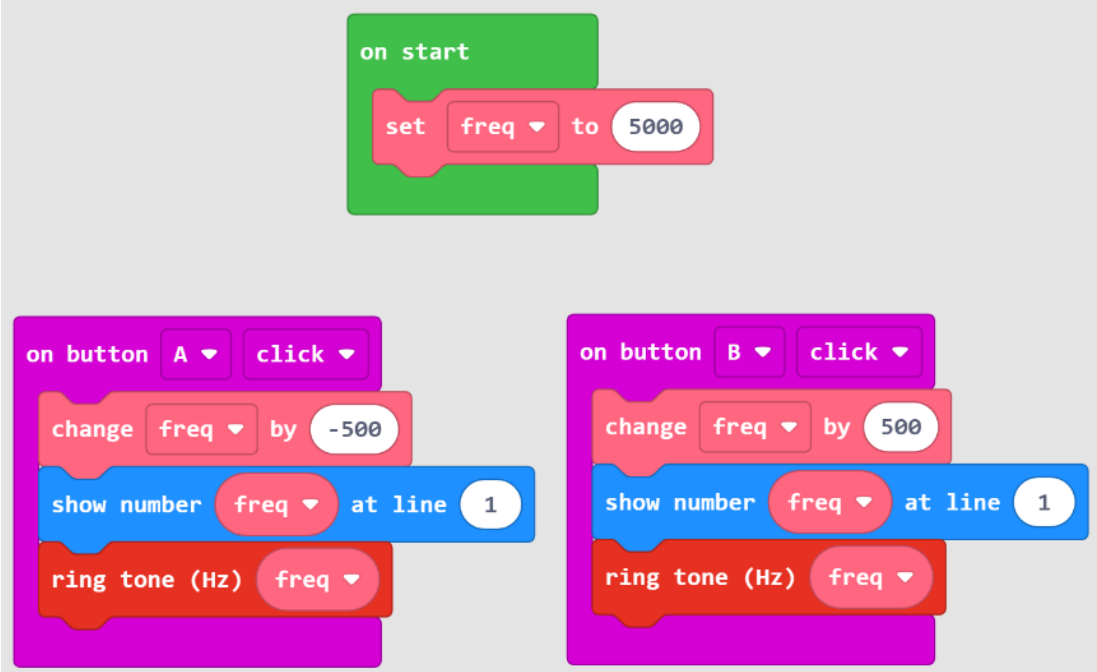
First, make a variable called freq. From under the variables group, click **Make Variable…**

Name the variable freq, and you will have some new options under variables.

We can now set the initial value of the variable to 5000, since we know everyone can hear 5000hz usually. Then we need to add the “on button” events. Where one will increase the variable by 500 and the other will decrease by 500. To decrease the value, we simply use a negative number. Finally, we want to show the value on the screen and so we “show number” and the number is the variable.



Pressing the buttons will now change the variable and show it on the screen. Test it out and make sure all is working as expected. We now can add the “ring tone” block to set the speaker to the desired frequency.



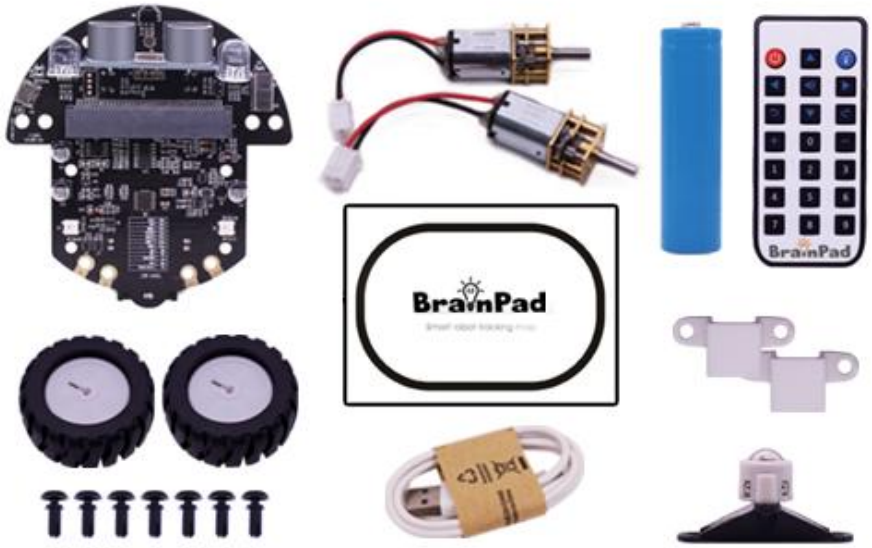
So, what frequency can you hear?

ROBOTICS

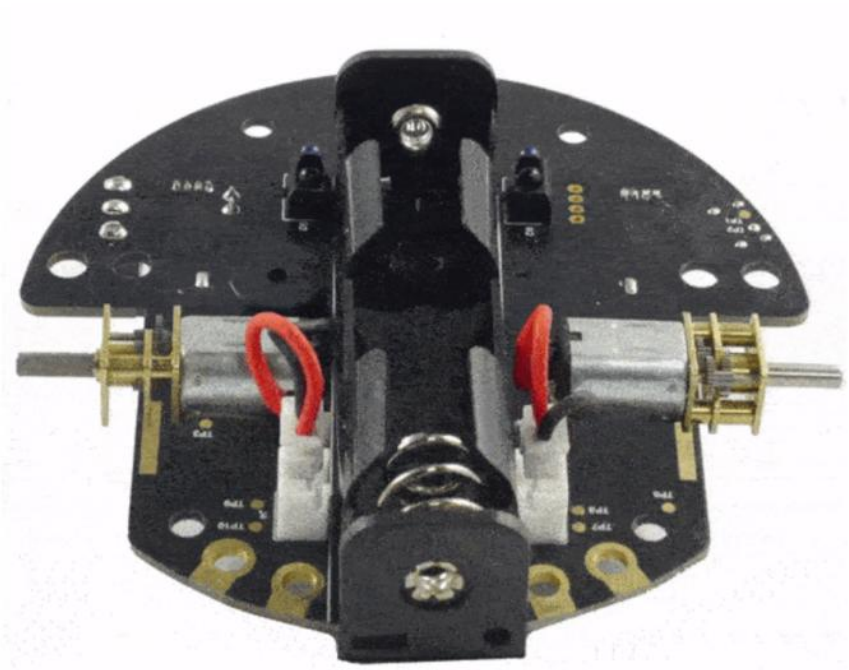
The simulator has served us well so far, but it is time to start controlling a robot. BrainBot pairs up beautifully with BrainPad Pulse. It is plug and play with the provided extension, which we will use now.

ASSEMBLY

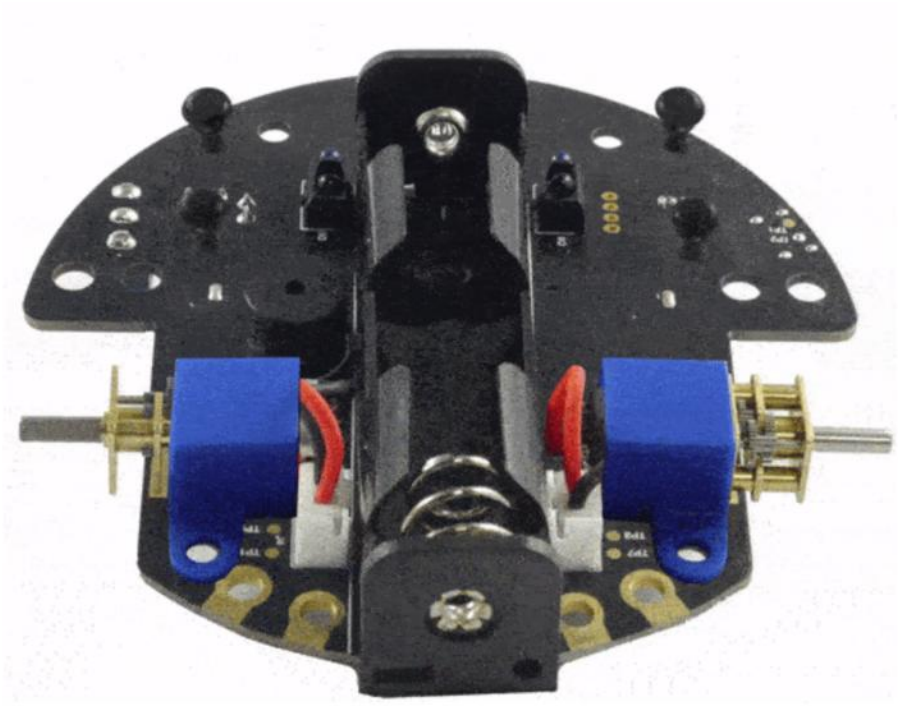
Out of the box, the robot requires minor assembly. We will keep the explanation brief here, but the website has more details https://www.brainpad.com/lessons/brainbot-assembly/ .



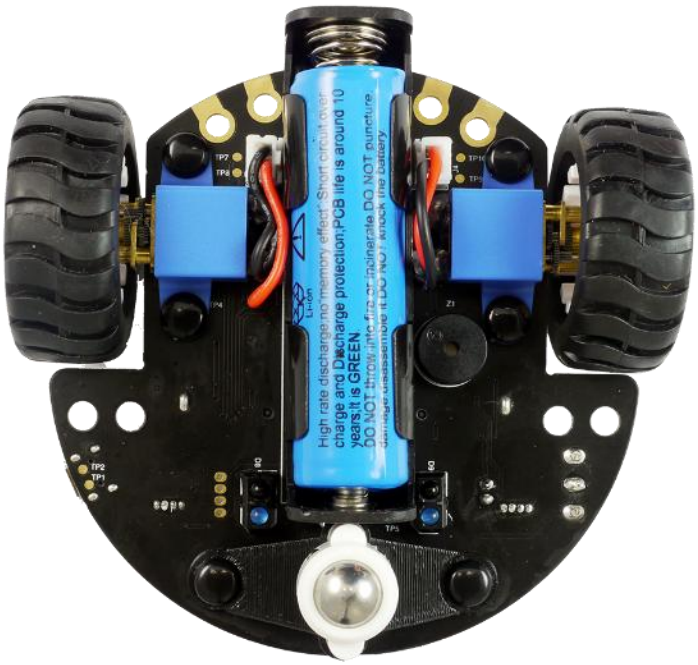
The first step is to add the motors, which come with the connectors to plug right in.



They are supported by brackets and plastic rivets.



The front caster wheel also needs to be mounted using the included rivets. And do not forget about the battery!

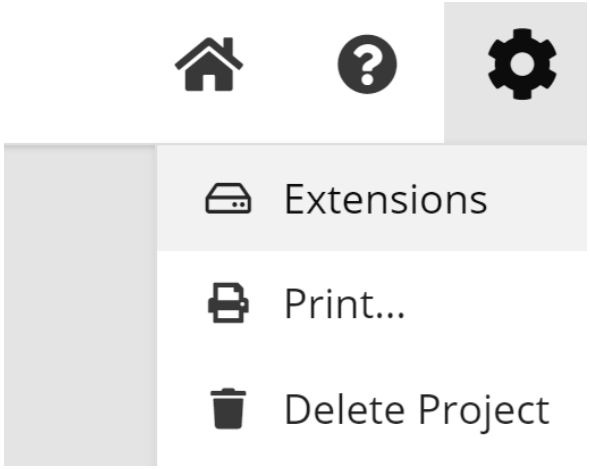


By the way, this is a high-density battery, and some safety precautions are necessary, https://www.brainpad.com/safety/ .

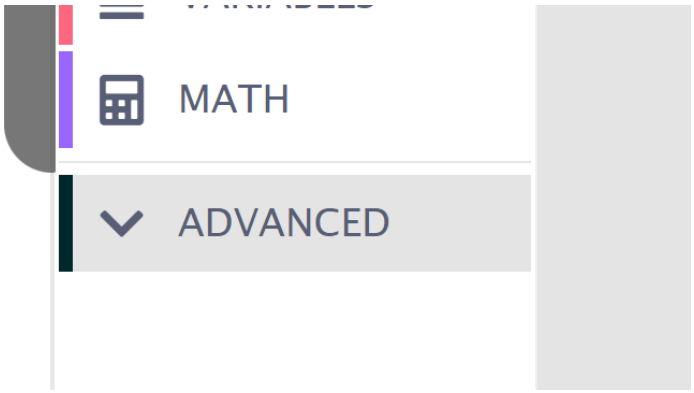
EXTENSIONS

MakeCode loads many default blocks that are common for all projects. Those blocks can be extended to add new additional features/blocks to the system.

Adding an extension can be done in two different ways. The first way is through using the settings gear in the top right corner of the MakeCode interface.

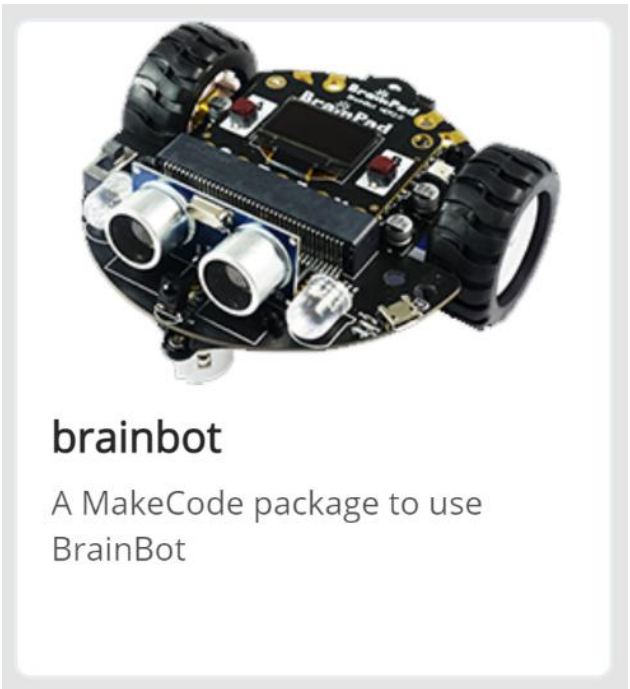


The second option is found under the advanced tab at the bottom of the block menu.

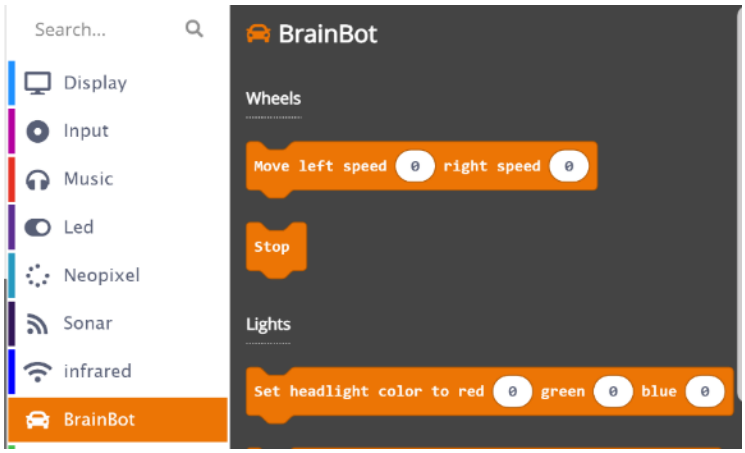


Extensions is found at the very bottom of the list.

From the available extensions, select and add the BrainBot extension.

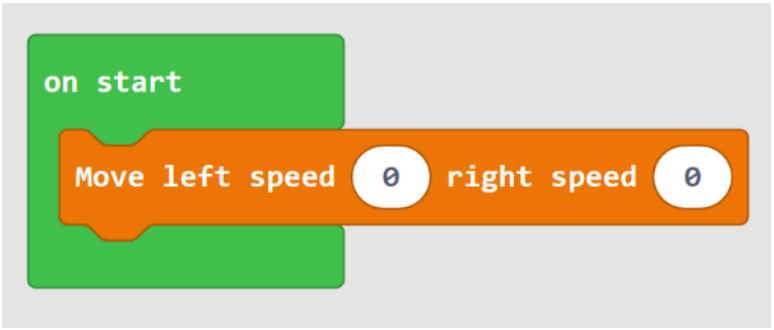


And now MakeCode has an array of new blocks to control the robot.

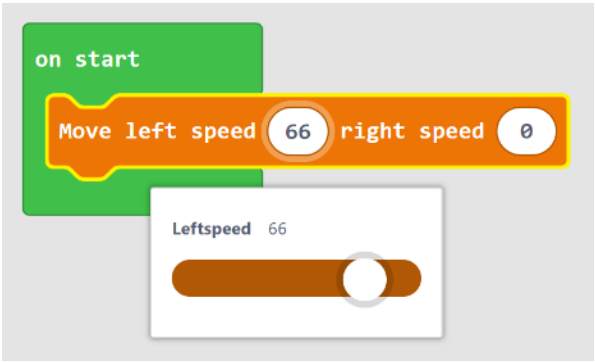


MOVING

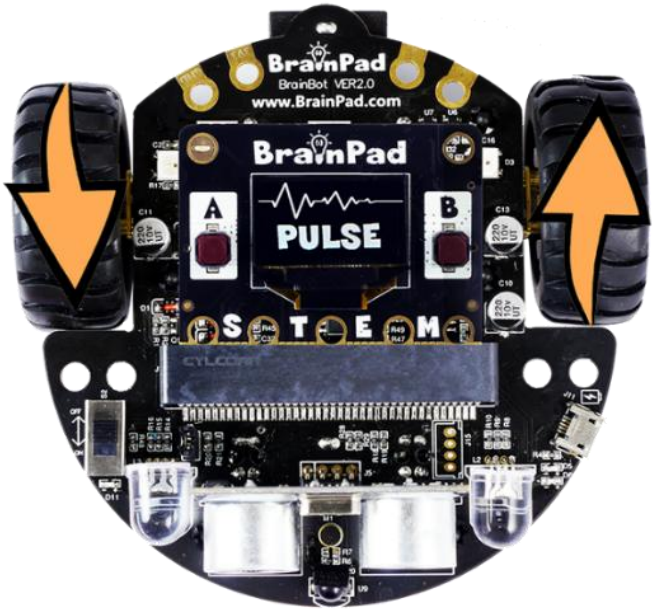
Moving the robot is done by setting the left and right wheel speeds independently, using the move block.



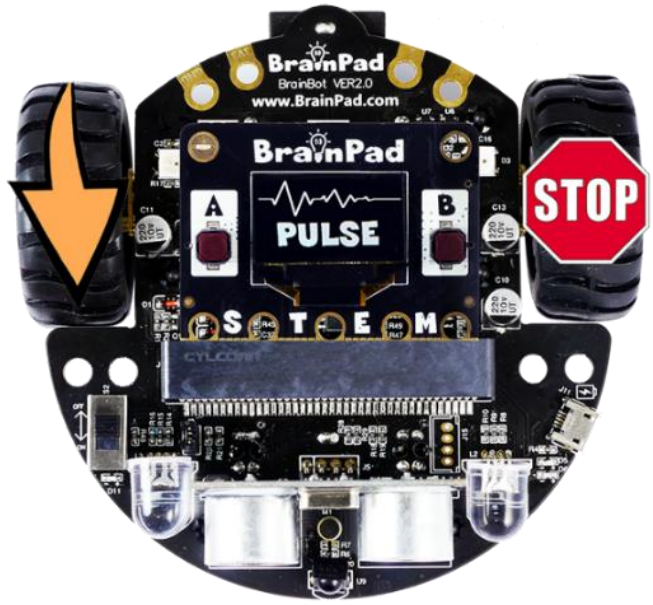
The speed can be set to a positive number, for going forward, and a negative number for going backwards.



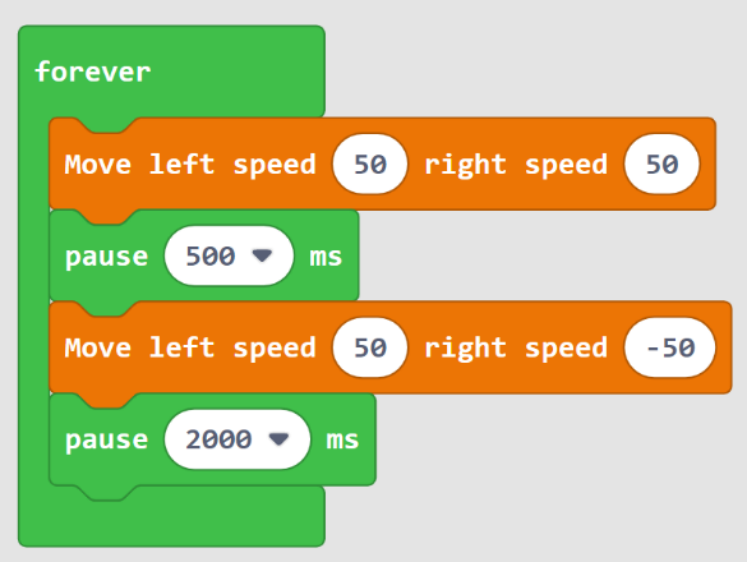
Set both motors to 50 and the robot will move forward at half speed. Then changing the number to negative numbers will make the robot go in reverse. There are two ways to run the robot. Turning in place can be done by going forward on one wheel and backwards on the other.



The other option is to stop one of the wheels and move the other. This will make the robot turn in a larger radius.



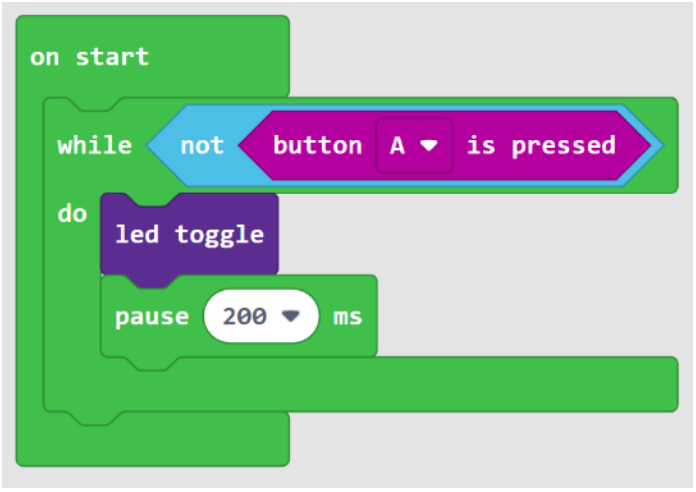
Here is an example to make the robot move forward for 500ms (half second) and then turn in place for 2sec. This, of course, needs to happen in a forever loop.



Can you change the blocks so the robot is moving as it is going around a square? The blocks are the same, you just need to change the timing and the speeds.

RUN OFF

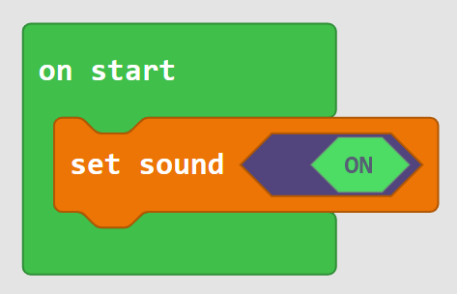
When using the robot, it is a good idea to have it wait for a button press before it starts performing whatever it is coded to do. This is simply done by adding a while loop that keeps looping as long as the button is not pressed. Pay attention to “not” in the code. And while the button is not pressed, we will toggle/blink the LED.



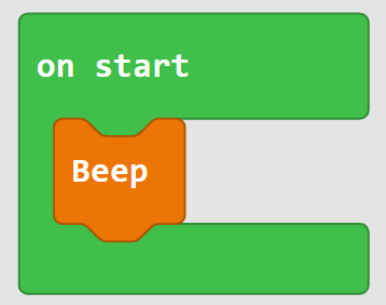
BUZZER

BrainBot includes a buzzer. By the way, the BrainPad Pulse onboard buzzer/speaker is another option and has more options as well.

The buzzer on the robot is set to a specific frequency. It can be activated/deactivated using the set sound block.

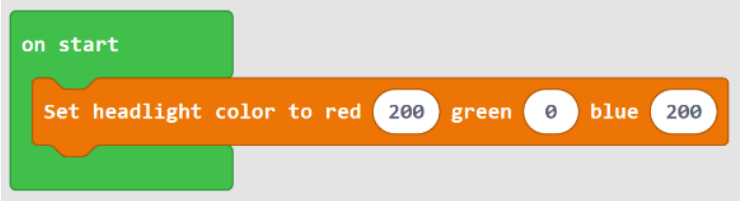


Or a beep sound can be generated.

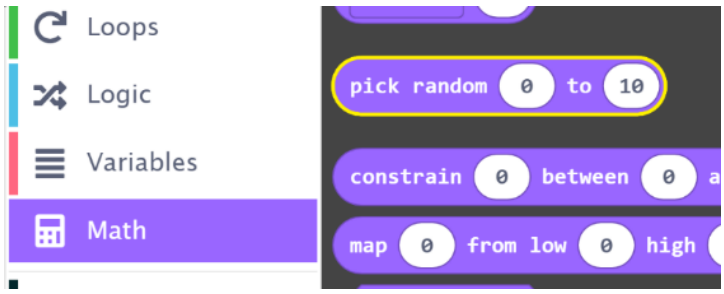


LIGHTS

There are headlights and taillights on BrainBot. The headlights can be set to any color, but both lights get set to the same color. The combination of Red, Green, and Blue (RGB) is what makes up the final color. For example, blue and red make purple.

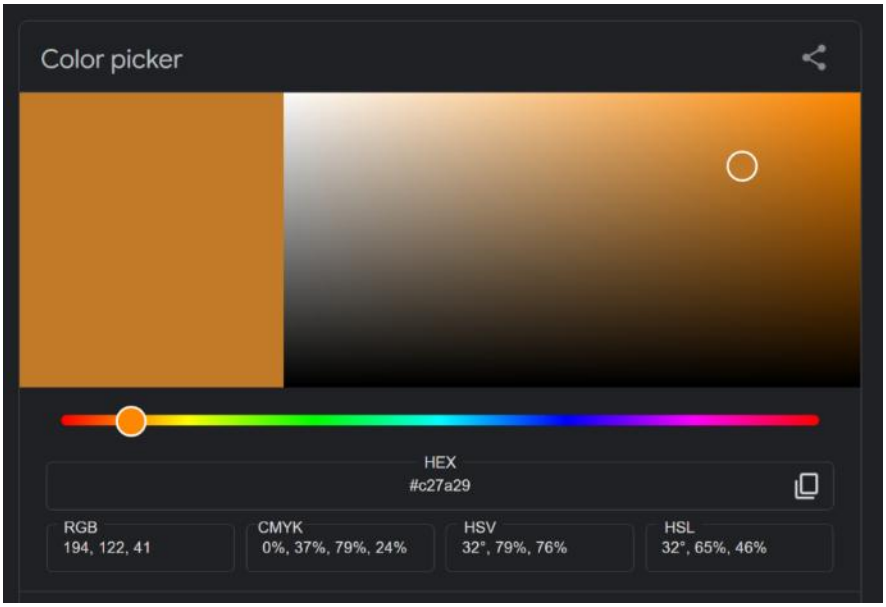


What about a program that shows different random colors? From Math, use the “pick random” block.



The range of color is 0 to 255, and so we set the range for that. We will randomize all three colors and put in a loop to change the color every 100ms. Note how this is a forever loop.

There are many online tools to help. In fact, just googling “color picker” will show a tool for colors. The picker presents the selected color in different formats. The one we need is RGB.

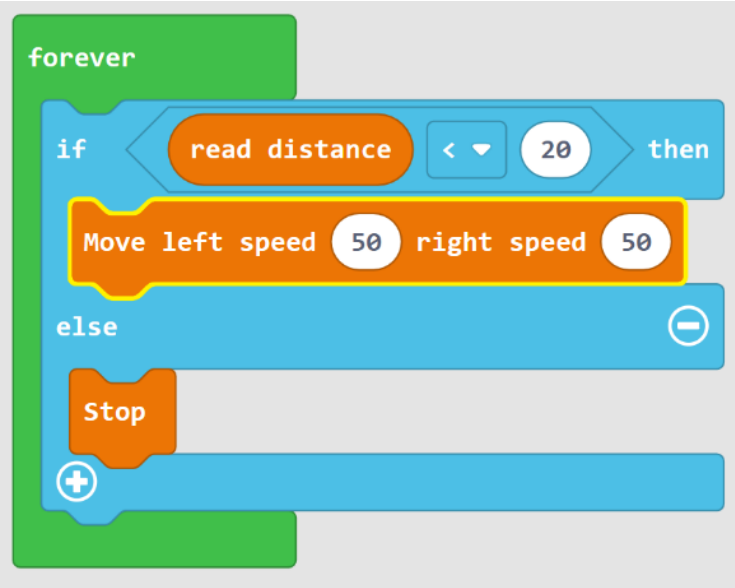


The two taillights can each be set to a different color. Let’s alternate the two lights red and blue. Something like a police car!

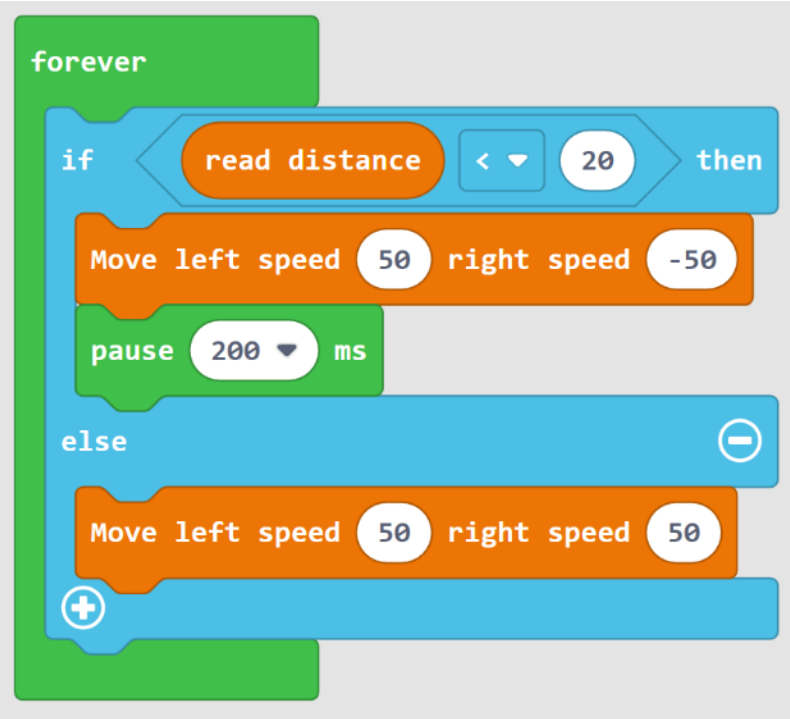
DISTANCE SENSOR

The front of BrainBot has a distance sensor. This sensor uses ultrasound to send a pulse, and then it measures the time needed for the sound to bounce back.

An example can be in a robot that “attacks” your hand! The robot will sit still until it detects an object in front of it, and then it will move forward up to it.



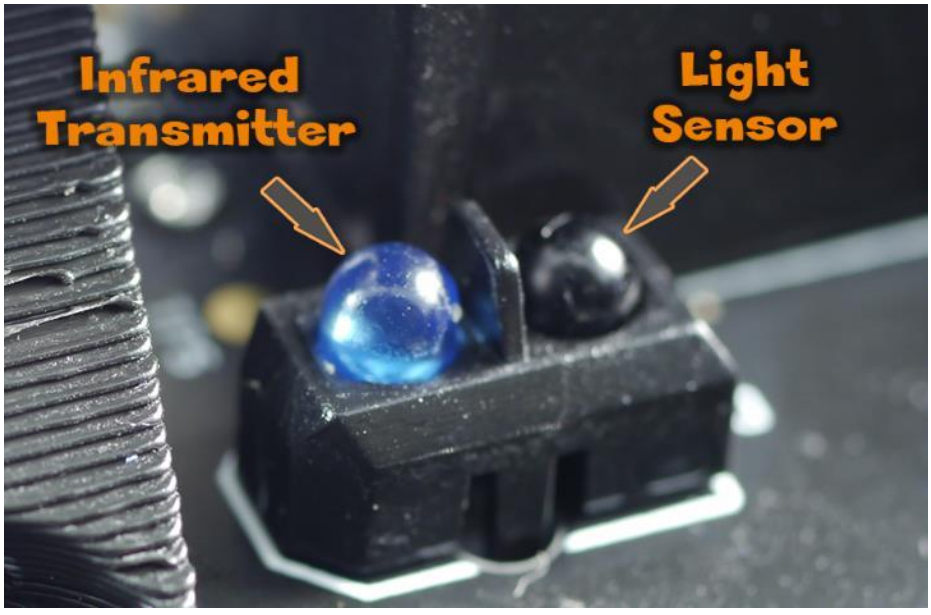
The robot can also use distance measurement to avoid obstacles. The used algorithm is very simple. If the robot detects a distance less than 20cm, it will turn for 200ms and then try to go forward again.



LINE FOLLOWER

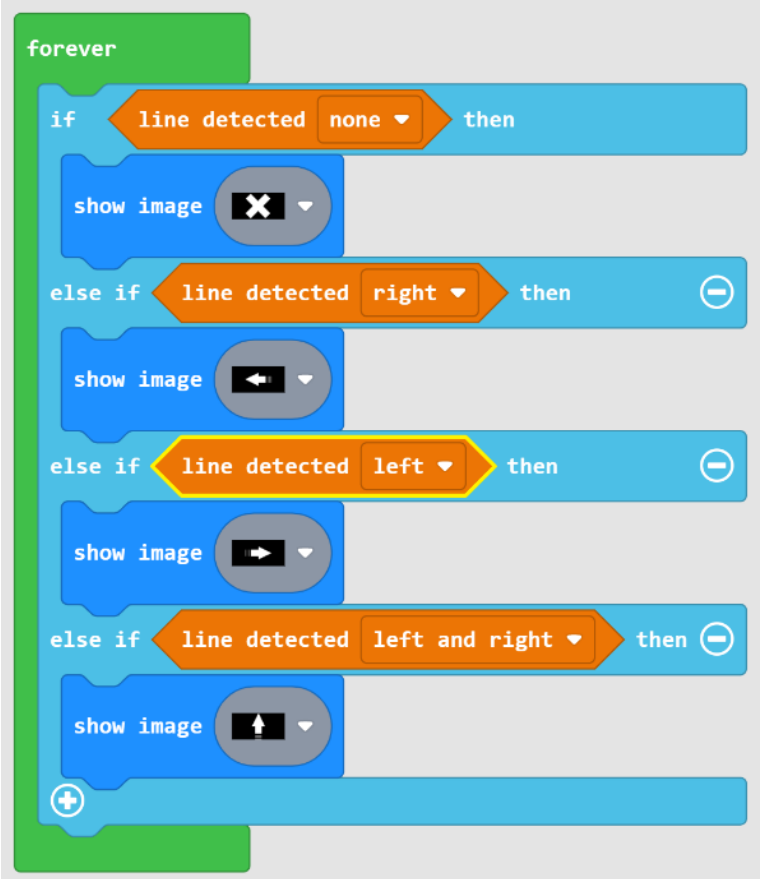
There are two ground reflector sensors.

The ground sensors use the ground to reflect an infrared beam back to a light sensor located next to them. If you look closely at them, you'll see a separator between both. This is so it only gets the value from the reflection off the ground and not spill over from the beam itself. The infrared light sensor may look black to our eyes, but it is designed to filter out all other light except the infrared.



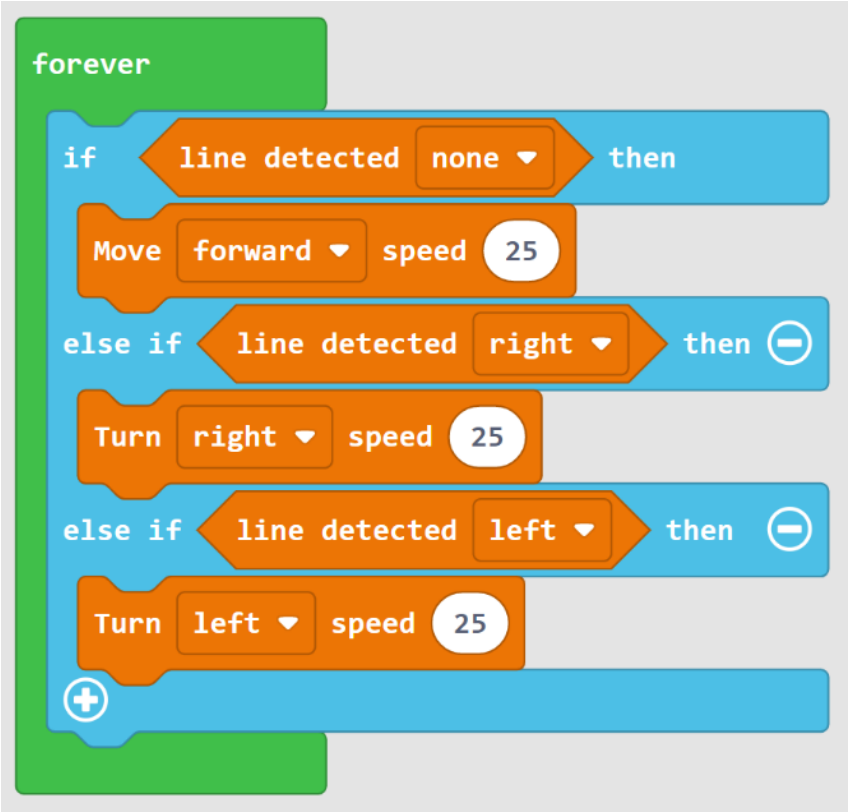
BrianBot Kit includes a line follower sheet, that can be used to get us started.

Before moving the robot, let's show the sensor readings on the screen to get a better understanding. Note how the left and right arrows are pointing the opposite direction since the BrainPad Pulse is mounted backwards from the "driver's" perspective.



The black line on the sheet fits right in between the two sensors, resulting in detection to being none. We are showing the X image to indicate that no lone is being detected. Place the robot on the sheet, over the line. Rotate the robot slightly right and left and observe the screen.

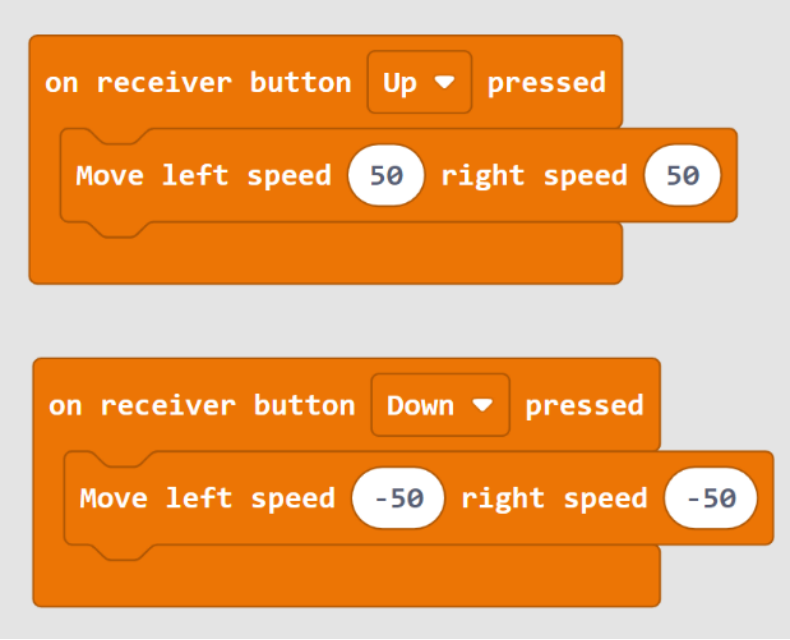
Let us give the robot some simple AI. We will move forward when no line is detected, assuming the robot started on the line. When the robot moves off the line, too far to the right, we would detect line on the left side. In this case, we need to rotate the robot to the left to bring it back. If the robot goes the other way, the action is the same but opposite.



Note that as the light sensor rely and measure the light levels, they do not work properly in a very bright environment, like under direct sunlight.

REMOTE CONTROL

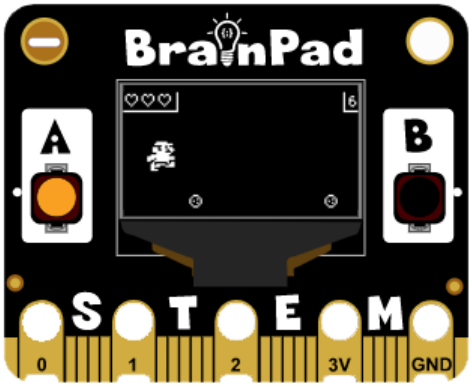
BrainBot also includes an infrared remote control. A simple use can be to control the movement of the robot. Like push up to move forward and push down to move backwards.



A better method would be to have a smarter AI robot but use the remote to tweak variables. Do not forget to take advantage of the screen to show information.

GAMES

Video games are a good fun way to experiment with coding. It requires no additional hardware and excites both children and adults.

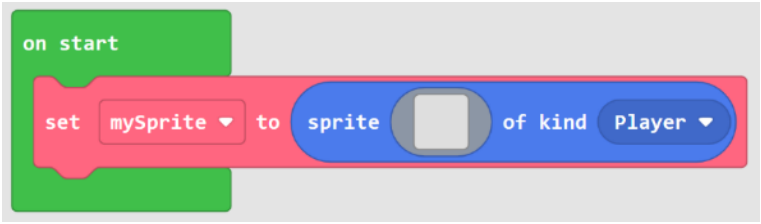


The MakeCode engine for Pulse includes a game engine. Just note that the game engine functions take over the screen control. The blocks under “Display” category should not be used once we use the game engine functions.

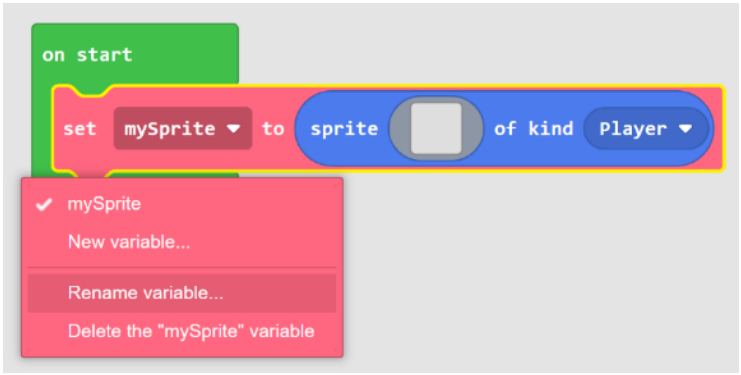
SPRITES

Sprites are the 2D images shown on the screen. They can be stationary, can move, and can be animated. A sprite can be the hero being controlled by the buttons. Then the enemies will try to attack the hero. Both the hero and the enemy are sprites, but the way they are handled in a game determine their behavior.

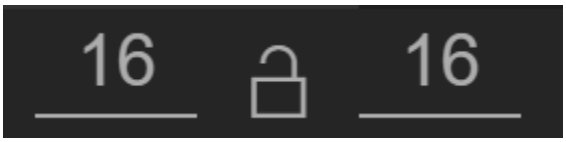
The Sprites category is found under advanced. From there, we can add a new sprite to our game.



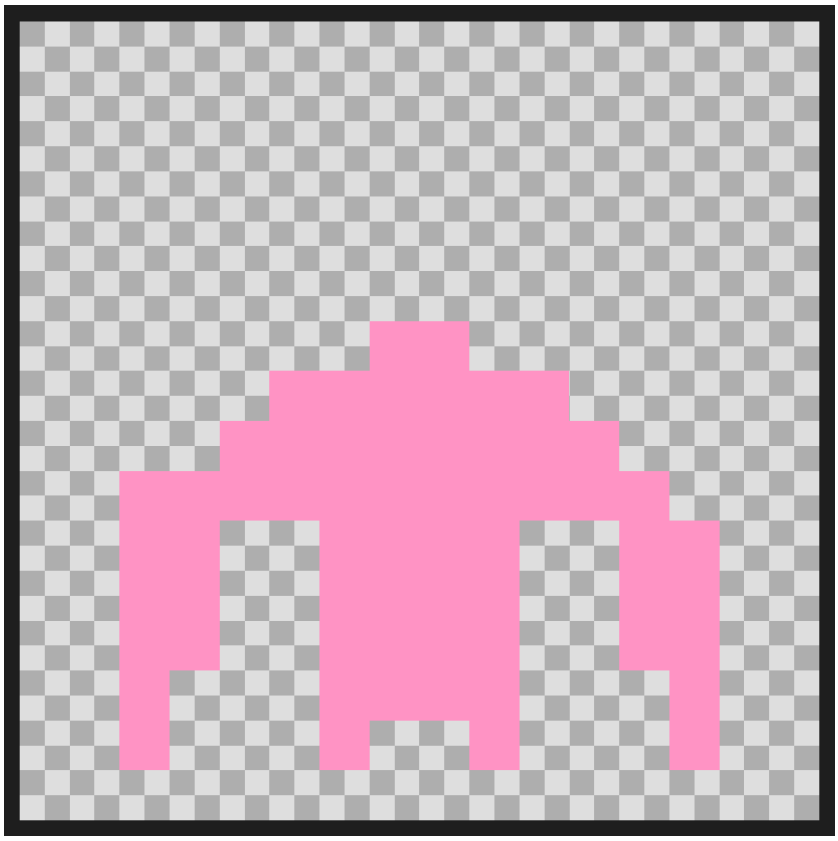
We should start by giving our sprite a better name than “mySprite”. This will help later when you have many sprites on the screen. Let us name it spaceship.



Click on the name and then “Rename variable…”. We can now give our sprite a new name. Next, we need to draw the sprite. There is a gallery of available sprites/images that can be used, or we can draw our own. We will make our own. First, make sure the sprite size is set to 16x16.

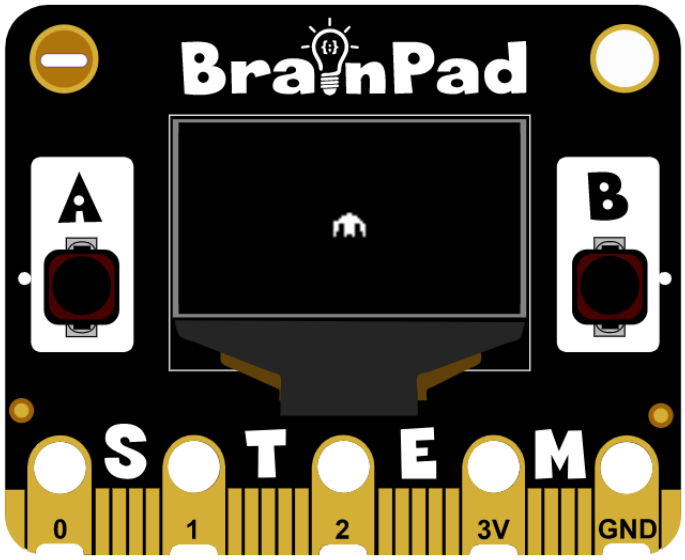


Then draw a spaceship!

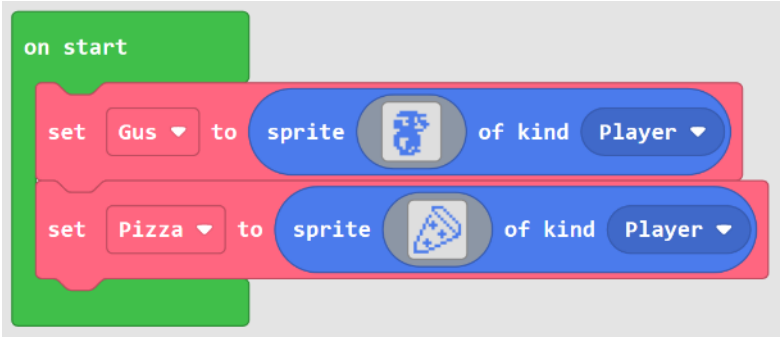


By the way, BrainPad Pulse is only black and white. Any color used will end up showing white on the screen.

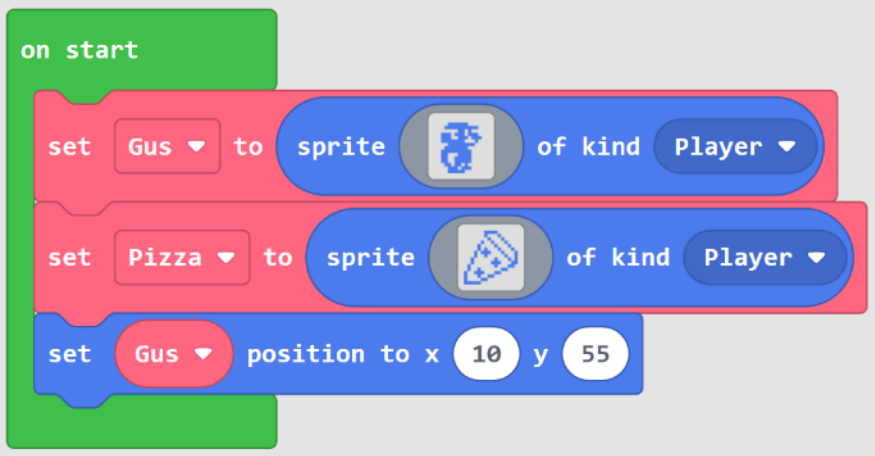
The emulator (and the device) will now show the sprite in the middle of the screen.



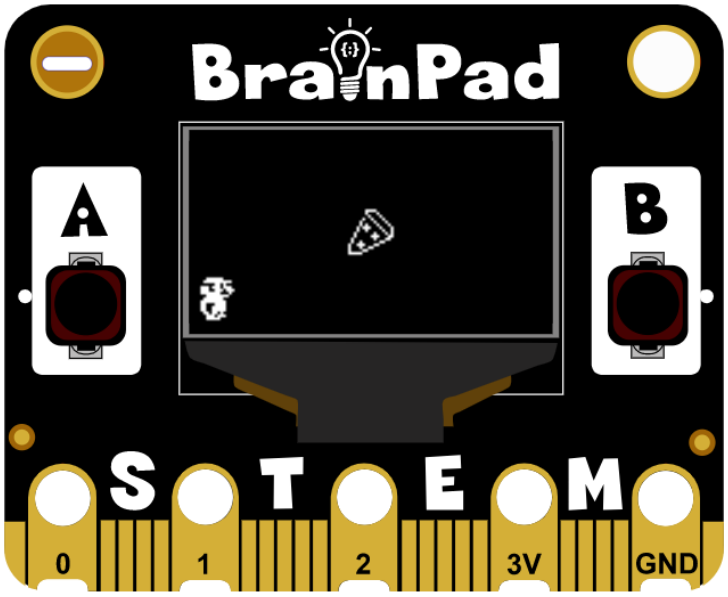
Let us start building a simple game. The created game will be “Hungry Gus”, where the player (Gus) will be going after pizza. So, we will start by adding “Gus” to the screen. Use the built-in gallery this time to select the guy. And, add another sprite, one for the pizza. Do not forget to rename the sprites appropriately.



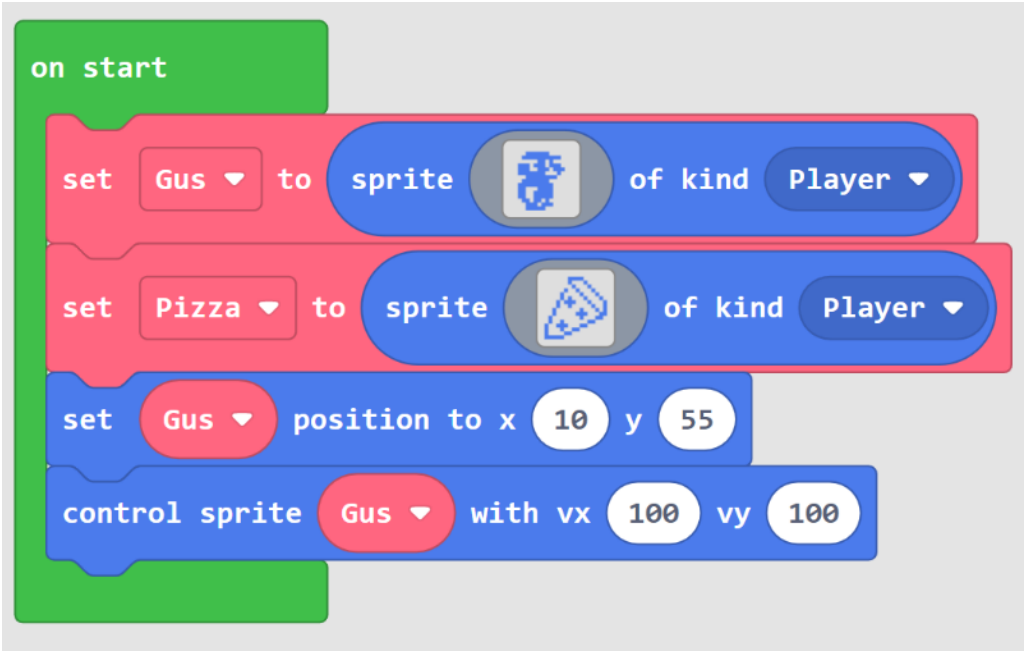
Both sprites will show overlapping on the display. We want “Gus” at the bottom of the screen.



Sprites will now show on the screen clearly.

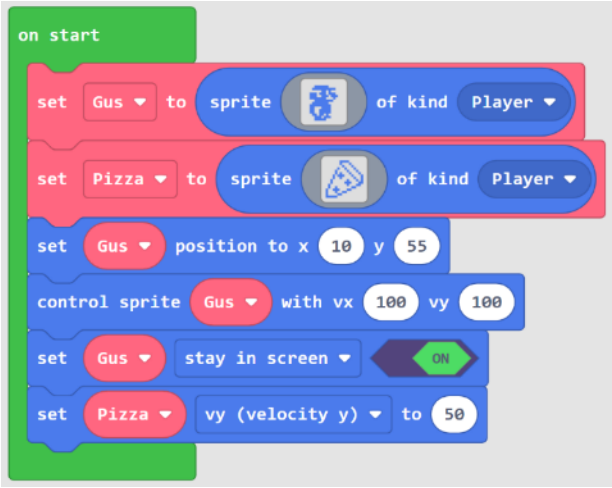


We now have the option to move the sprite manually, but in most cases, we can simply give control over the sprite to the engine using **control sprite** block. This block will automatically move the sprite left and right using the A and B buttons.



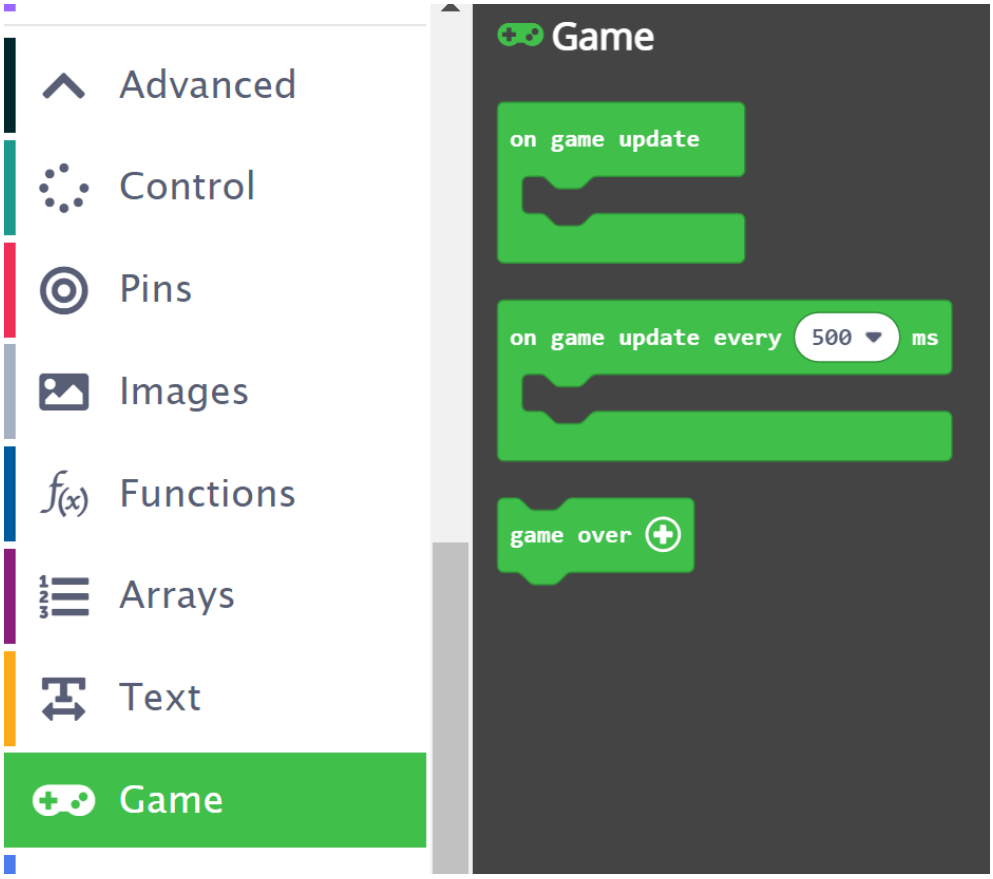
When moving the sprite, we can see that it can leave the screen if we continue to push one of the buttons. Once, again, we can manually check the sprite position and keep it on the screen or let the engine handle that for us by simply adding a single block to enable **stay on screen**.

The game engine can also move sprites by giving them acceleration and velocity. We will move the pizza by giving it velocity of 50 in the y axis.

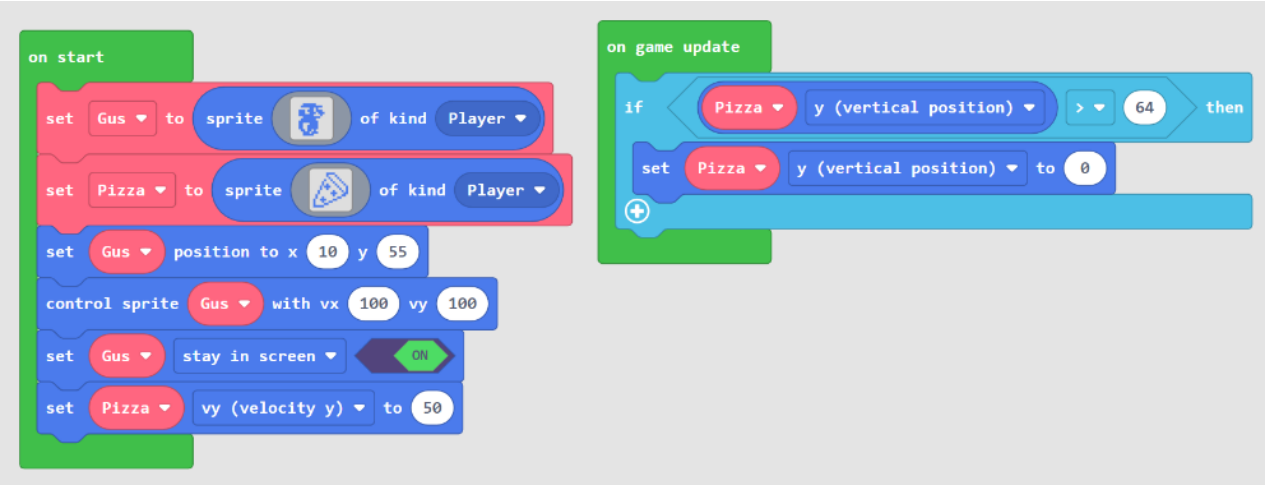


The pizza will now fall down and continue to exit the screen. We want the pizza to continue to fall down, and we have two options to accomplish that. We can “destroy” the old pizza when it leaves the screen and then make a new one, or we can simply change its position back to the top. We will go with the second option and move the pizza to the top.

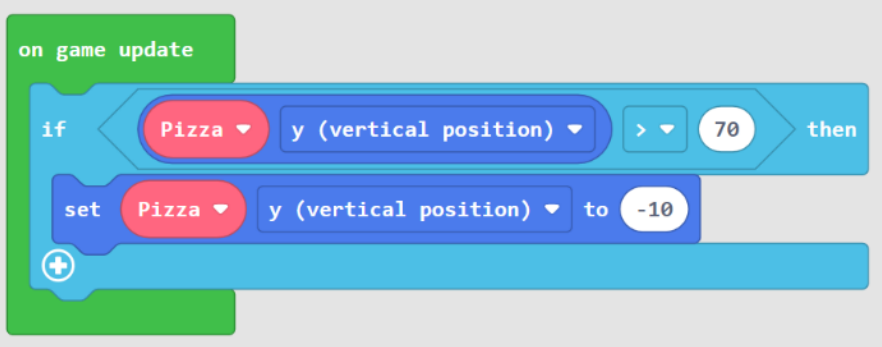
We will check the sprites position inside the game loop. This is similar to the forever loop, but the game loop runs once for every game frame instead of repeating as fast as possible. Find and add the **on game update** block.



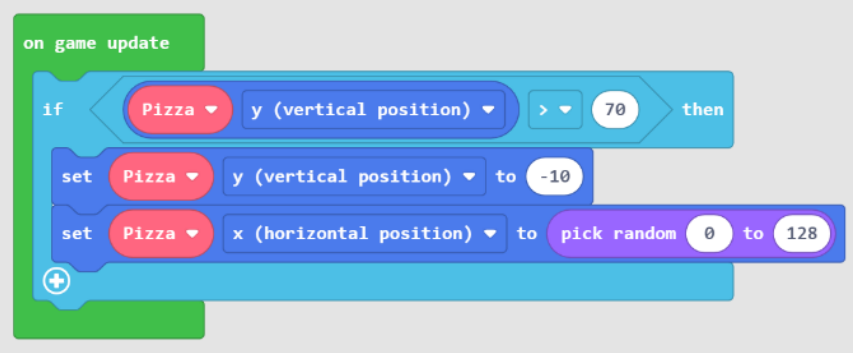
In the loop, we can check the position of the pizza. If the pizza position is more than the screen height, then move the pizza back to the top.



The pizza will now start falling endlessly! A little possible improvement is to start the pizza above and outside the screen and then let it continue to fall outside the screen at the bottom.



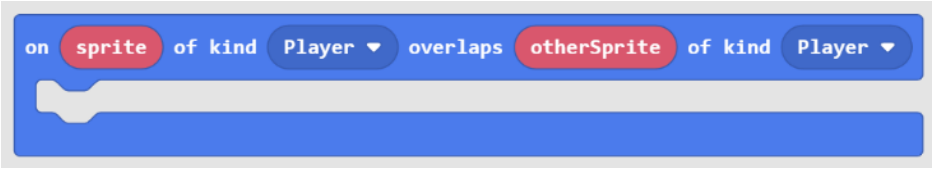
We next need to make the pizza fall from a different X location every time it starts to fall. For that, the **pick random** block from under **Math** category comes in handy. The random number we need here is anywhere from 0 (very left) to 128 (very right).



Let's summarize what is happening. We have a pizza sprite that starts with velocity of 50 on the Y direction. This will make the pizza move down. Then in the game loop, we check the current position of the pizza. Once the pizza's current Y position is more than the screen height, we will move the pizza back to the top. And we also move the pizza left and right with a random number, so the pizza comes down from different X locations.

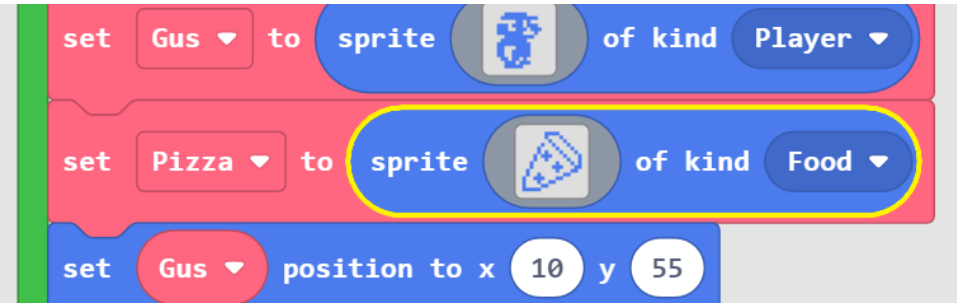
COLLISION

Collision is a feature in the game engine that detects that a sprite is overlapping another sprite.

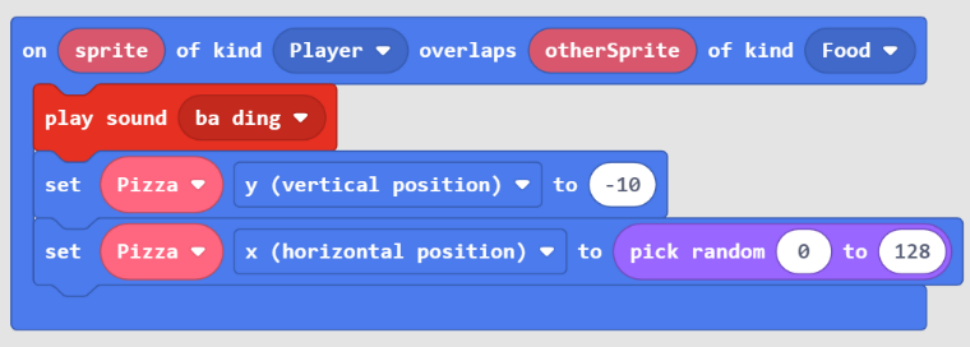


The game can have sprites of different kinds, like player, enemy, food, health, reward…etc. The kind of sprite comes in handy here because there will be a different outcome depending on what sprite kinds overlap. For example, a player overlapping enemy will kill the player. But an enemy overlapping another enemy is safe from the enemy and the player.

Continuing with the “Hungry Gus” game, we can detect if there is an overlap between Gus and the pizza, that is between Player and Food. We will start by changing the pizza sprite kind from Player to Food. By the way, this is not necessary in this simple game, but it is a good habit.

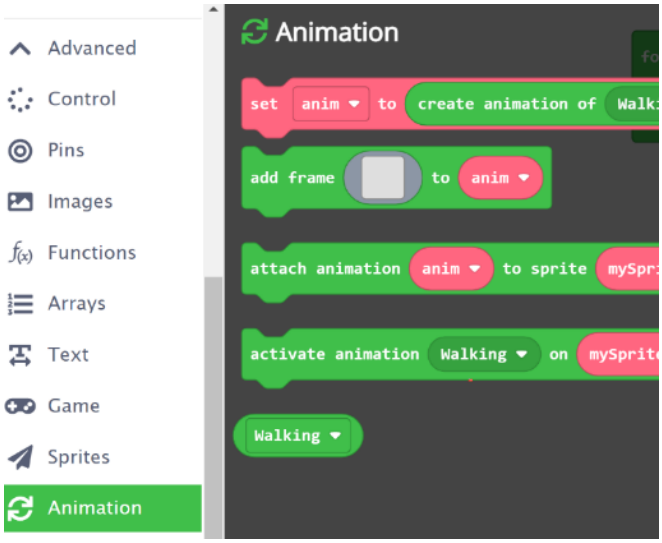


Now, if an overlap is detected, a sound is generated, and the pizza is moved to the top.



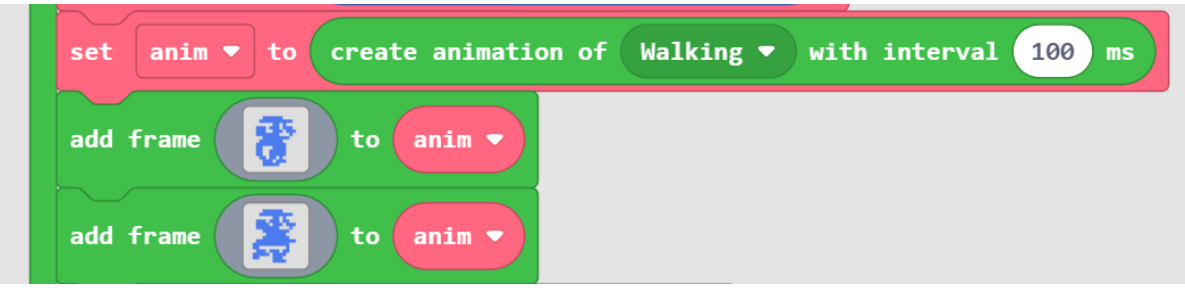
ANIMATIONS

We have created a complete game without needing to animate sprites; however, adding animations would make the game more interesting. There is a whole category dedicated for animations.

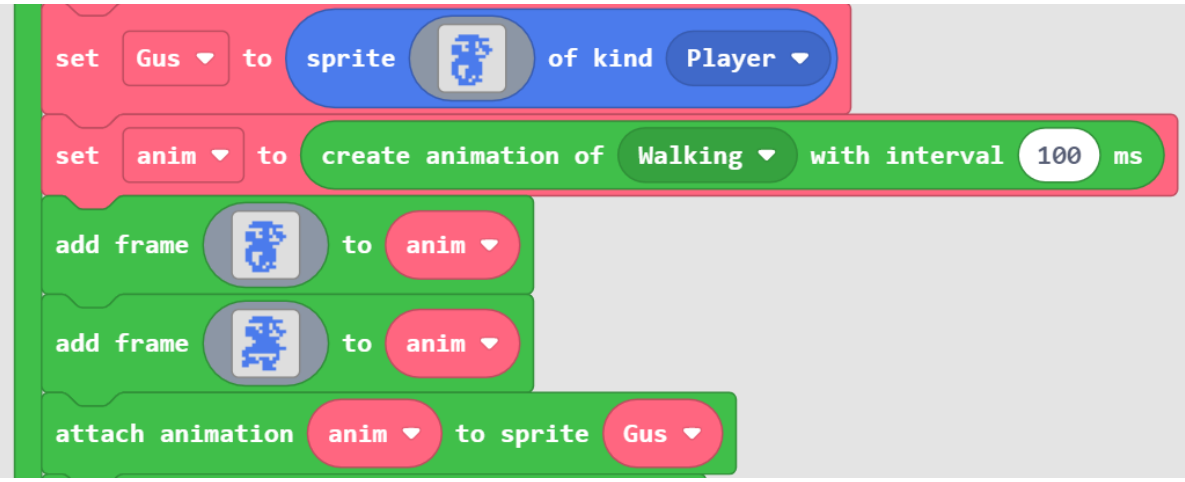


An animation is sequence sprites that the engine will automatically repeat. Those sprites override and replace what the sprite has for the non-animated image we have been using. We can continue with “Hungry Gus” game and make Gus walk!

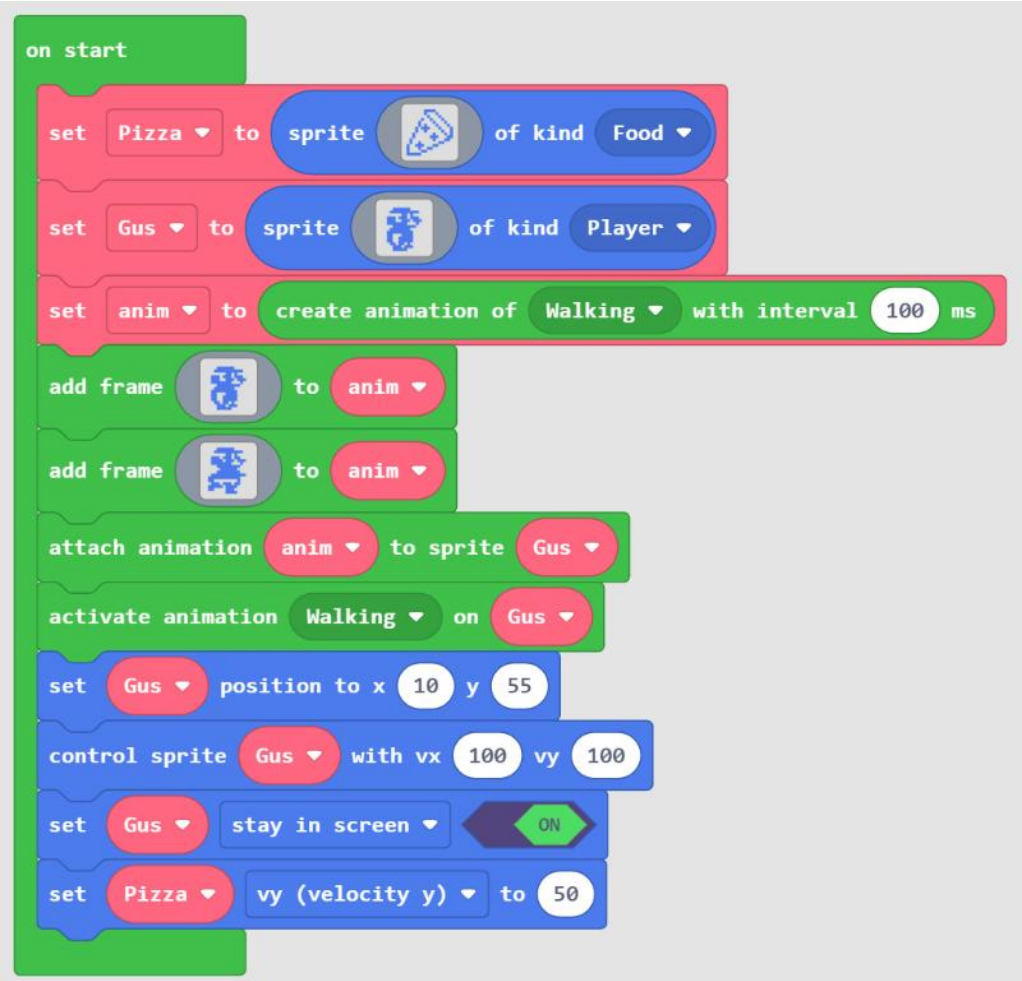
First, we need an animation. We will set the interval to 100ms and add two frames to that same animation. It is possible to add multiple animations to a sprite, like one for walking left and one for walking right… maybe also for jumping or falling. In this example, we will stick with **Walking**.



The animation can now be attached to any of the sprites. In this example, it is attached to “Gus” sprite.



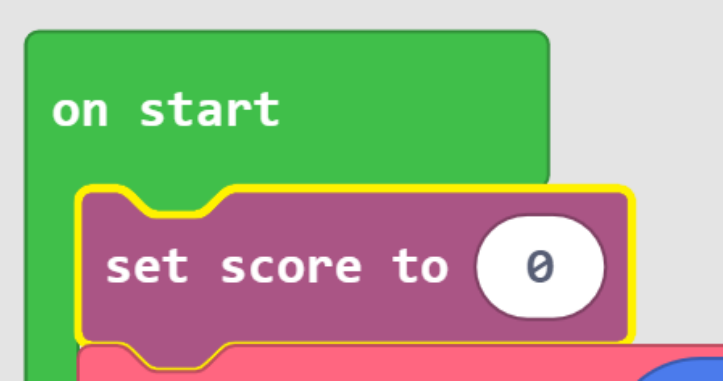
We now have the option to activate the animation anytime we desire. In this case, we will activate it right away and keep it active. Here is the complete **on start** block listing.



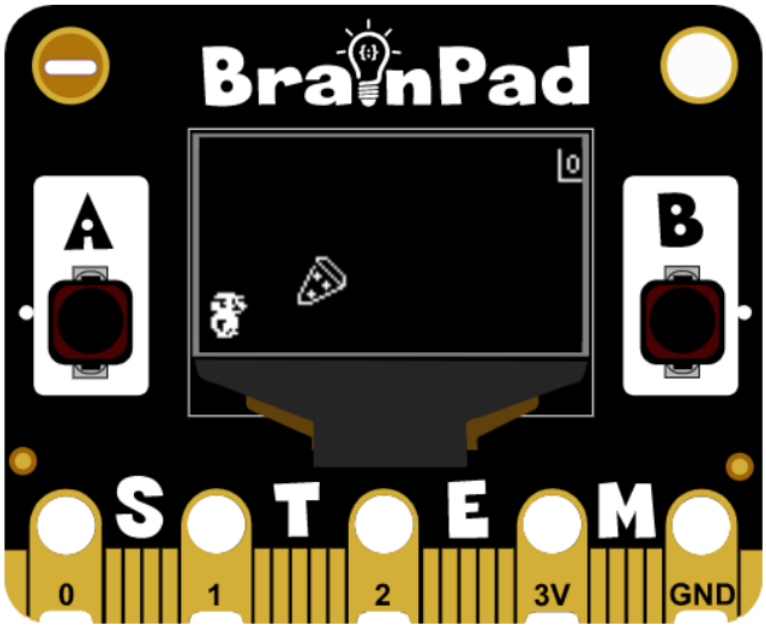
SCORING & LIVES

The built-in engine includes blocks to help in keeping track of lives and scores. Those blocks are found under **Info** category.

To get a score for catching a pizza, we start by adding the **set score** block anywhere at the beginning, inside the **on start** block.

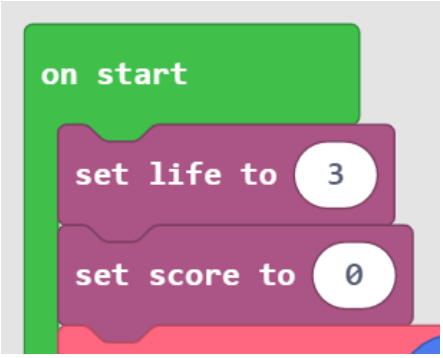


This will automatically add score at the top right corner of the screen.



The score can then be increased when desired, like when Player overlaps Food.

Similarly, the player can have lives. Maybe start with three lives.



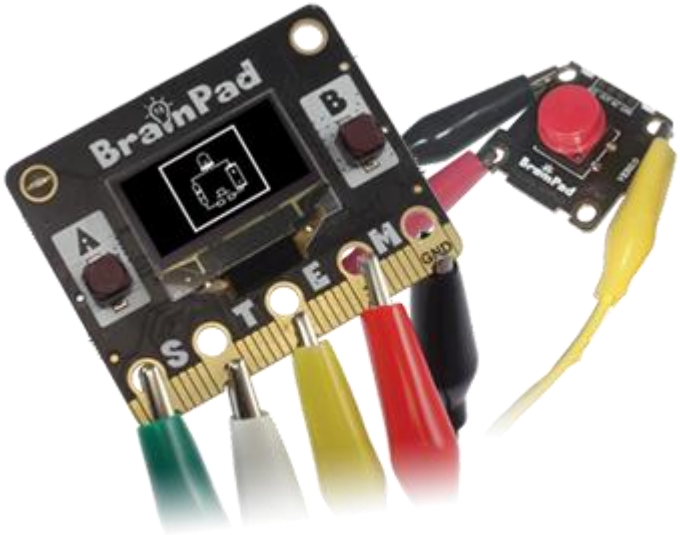
Perhaps, we decrease lives when Gus misses a pizza slice. This happens when the pizza reaches the bottom without an overlap.

The game engine will automatically terminate the game and show the score when there are no more lives.

CIRCUITS

There are thousands of options when it comes to possible modules, sensors and motors that can be used with BrainPad Pulse. This section will utilize the BrainClip accessory kit to demonstrate how circuits can be used. The kit consists of several modular circuits with hole pads on the corners. It also includes alligator-clip wires.

The modules can be connected to BrainPad Pulse using those clip wires.



DIGITAL

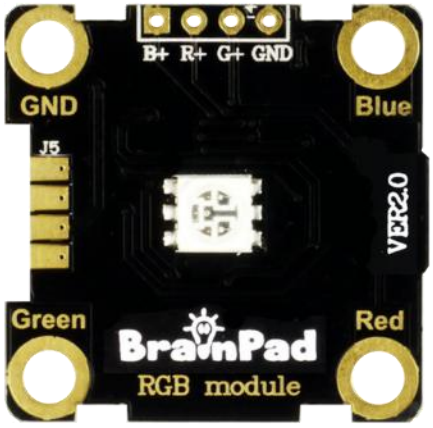
Processors usually have many “digital” pins that can be used as inputs or outputs. When saying “digital pins” we mean that the pin can be set to “one” or “zero”, nothing else. A “one” means the pin is active, it is on, it is high, it has voltage on it. When the pin is “zero”, it is inactive, it is off, it has no voltage on it.

DIGITAL OUTPUTS

We know that a digital output pin can be set to zero or one. But note that one doesn't mean it is 1 volt, but it means that the pin is supplying voltage. If the processor is powered off 3.3V, then the state 1 on a pin means that there is 3.3V on the output pin. By the way, it is not going to be exactly 3.3V but very close. When the pin is set to zero, then its voltage is very close to zero volts.

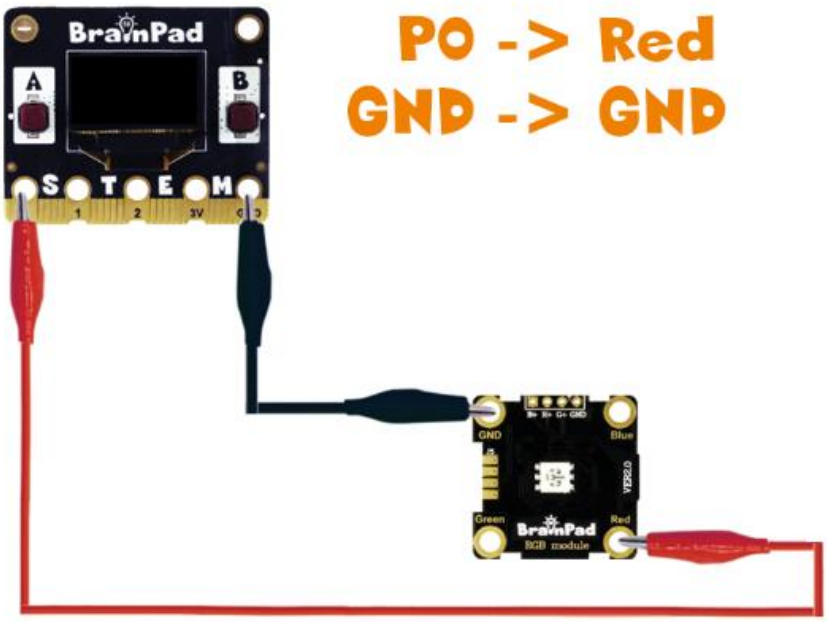
Those digital pins are very weak! They can't be used to drive devices that require a lot of power. For example, a motor may run on 3.3V, but you CANNOT connect it directly to the processor's digital pin. That is because the processor output is 3.3V but with very little power. It is meant to be a “signal” that drives the circuit that, in turn, drives the motor. You can still drive a small LED directly from a pin.

How about we control the BrainClip RGB LED?

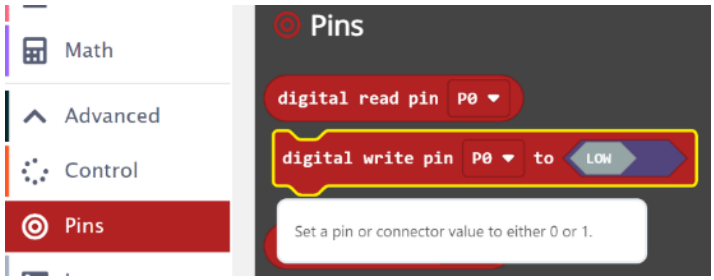


This is an RGB LED, meaning it has three mini light elements inside that are capable of producing Red, Green, and Blue colors. These are the primary colors used to make up any other colors. More on that in a minute.

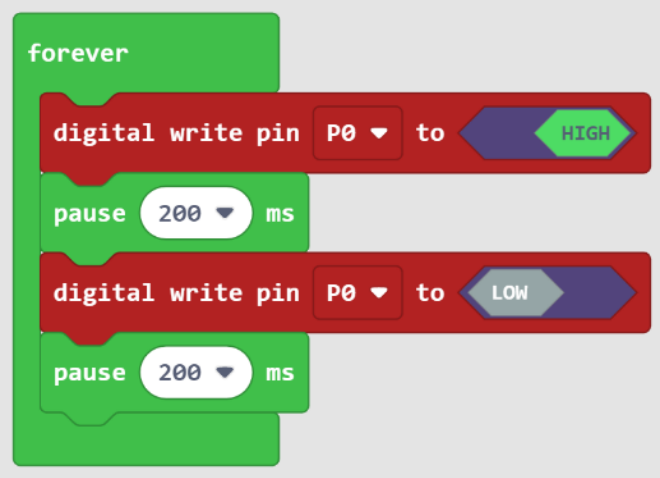
Use the alligator clip wires to connect the pad labeled Red on the RGB module to the first pad P0 on BrainPad Pulse. Also, connect another alligator clip wire from the GND pad on the BrainPad Pulse to the GND pad on the RGB module.



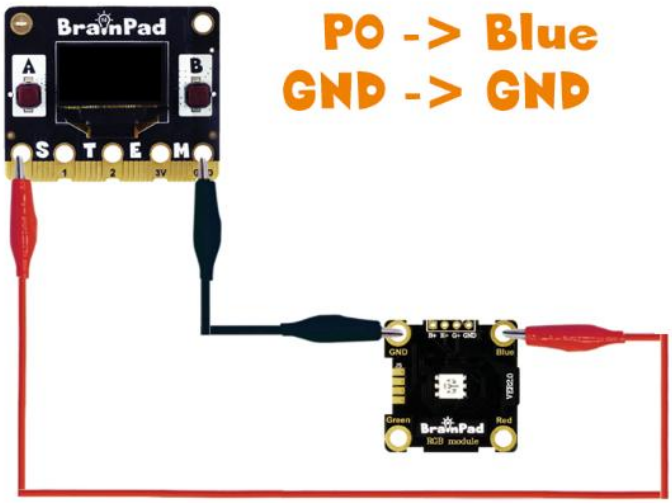
For blocks, we need the “Pins” block category, which is found under “Advanced”. We are specifically interested in “digital write pin”.



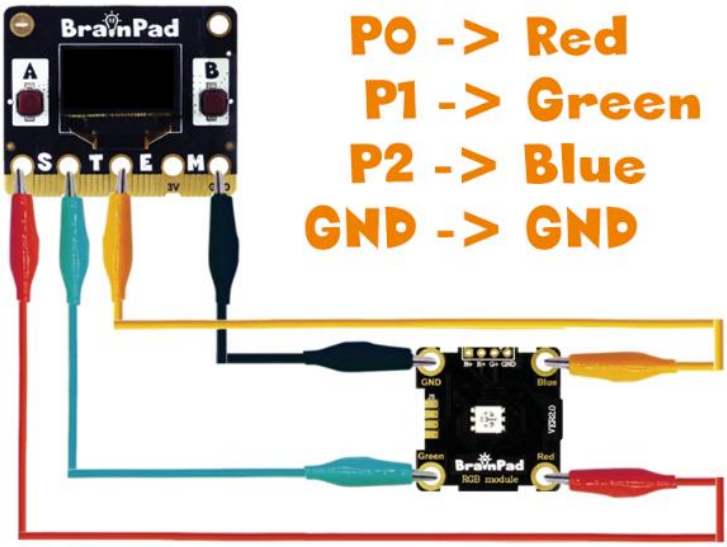
We will use the forever loop to set the pin to high and low. Do not forget about the delays, computers are very fast.



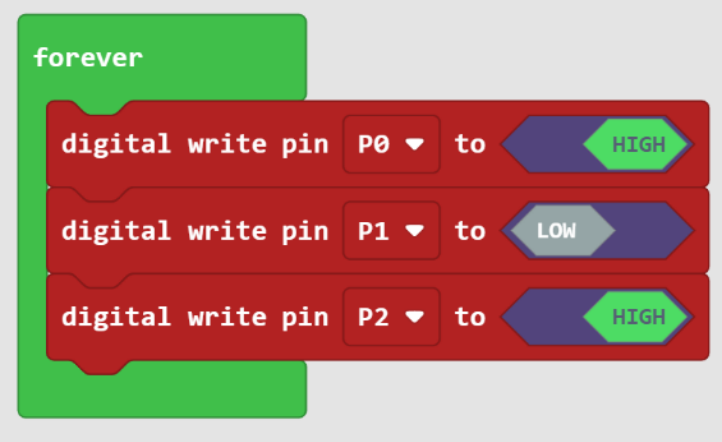
Load the program onto BrainPad Pulse and the RGB module should blink Red, awesome! Move the alligator clip from the Red pad to the Blue pad on the RGB module and the LED will now blink Blue.



We now want to control all three primary colors. Connect 4 alligator clip wires as shown in this diagram.



You can now set the LED to any of the 8 combinations. Why 8? 2 to the power of 3 is 8! That is 7 colors plus off, which is “Black”. For example, turning Red and Blue on will make Purple.

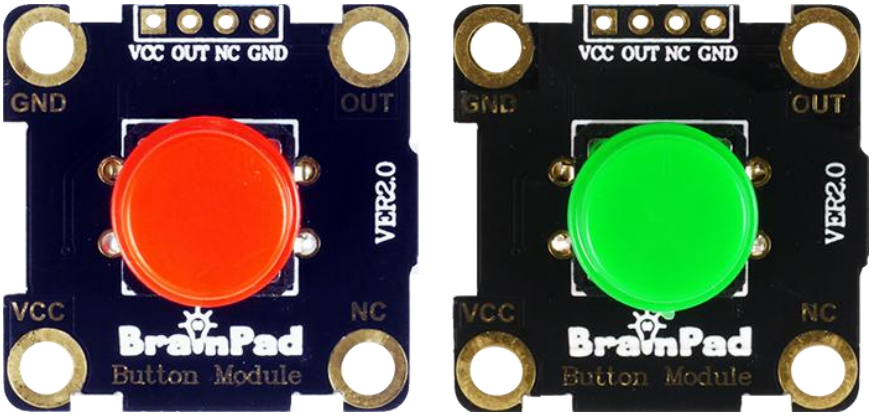


It is possible to get more colors by using analog output. More on this later.

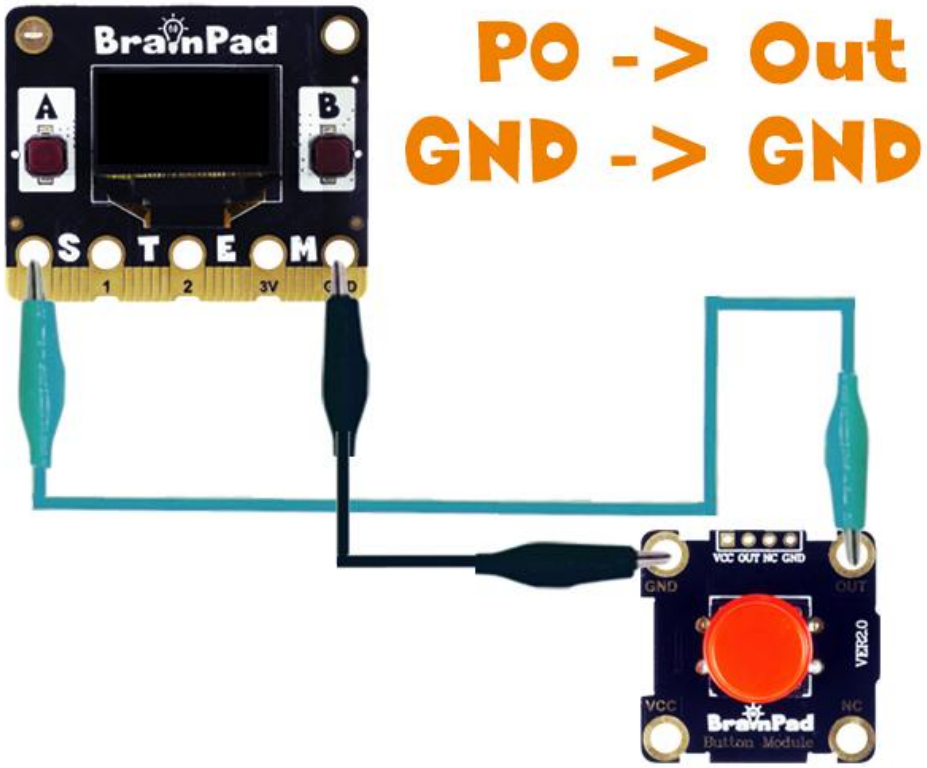
DIGITAL INPUTS

Digital inputs sense if the state on its pin is high or low. There are limitations on those input pins. For example, the minimum voltage on the pin is 0 volts. A negative voltage may damage the pin or the processor. Also, the maximum you can supply to the pin must be less than the processor power source voltage, which is 3.3V on BrainPad Pulse. However, the pins on the BrainPad Pulse are 5V-tolerant. This means that even though the processor runs on 3.3V, the input pins can tolerate up to 5V. But why 5V? Older digital circuits ran on 5V. And many digital circuits today are 5V. Being 5V tolerant allows us to use any of those digital circuits with the BrainPad Pulse.

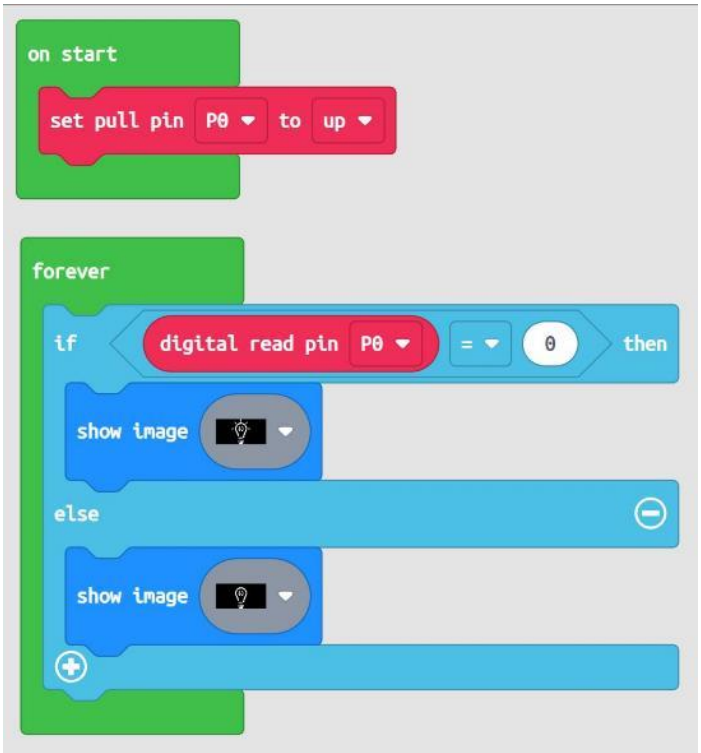
Important note: 5V-tolerant doesn't mean the processor can be powered off 5V. Always power it with 3.3V. Only the input pins can tolerate 5V on them. We will start with the button to try out digital inputs.



Connect the button as shown.



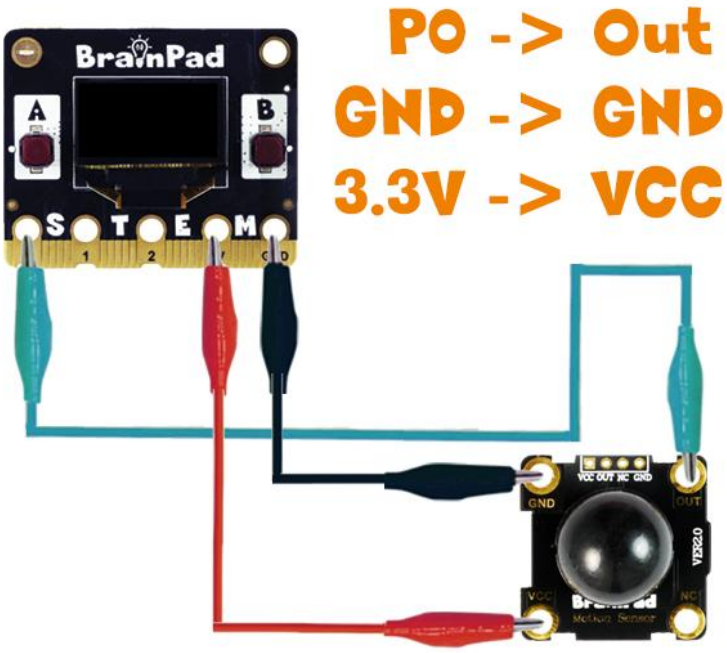
The code will use the button to control the lightbulb on the screen.



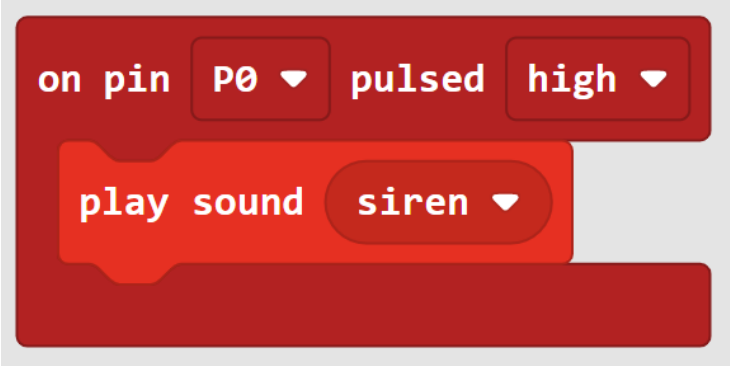
Have you noticed how we check for a “0” state on the pin to turn the light on? This is because we are using a block to pull the P0 high/up. Measuring the pin on the button module will result in “1”, which is high/up. Pressing the button will make a connection between the pin and GND, making the pin “0”. This maybe seems backwards to you, but this is normal practice when connecting a button.

Another possible module in the BrainClip kit is the Motion Sensor module.

We will use this sensor to create an alarm. The OUT pad goes to P0, with power and ground connecting as usual.



Instead of using a loop to check the pin repeatedly, we will use an event that is fired when the pin changes. The play sound block is found under the music category.



The event has the option to fire when the pin goes from low to high (pulse high) or from high to low (pulse low).

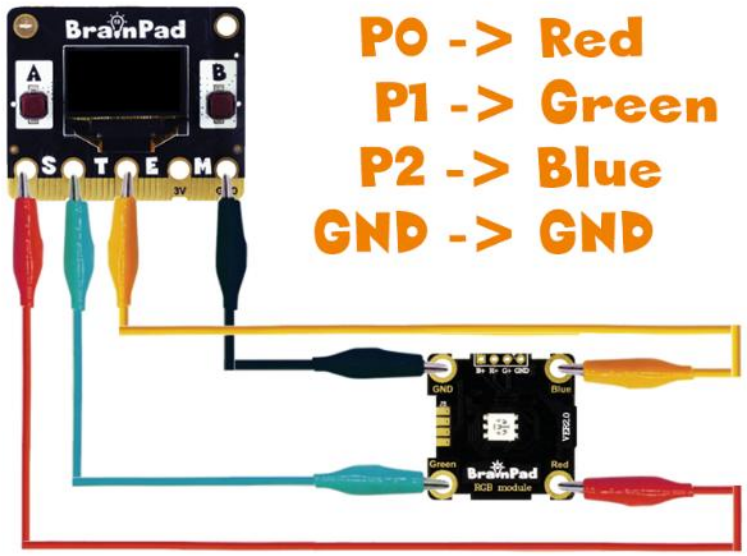
ANALOG

An analog value has level instead of just on and off in digital pins. An analog output is achieved by using something called PWM. An analog input requires a special circuit that is called ADC.

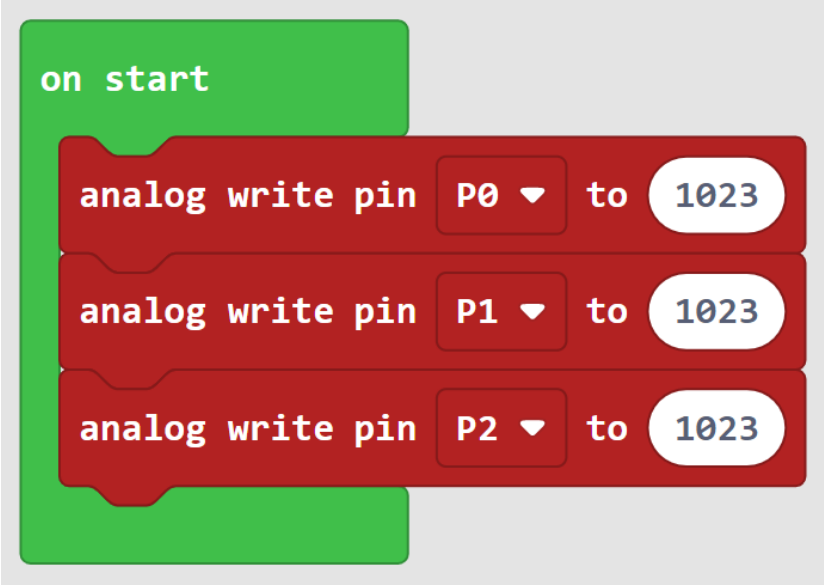
ANALOG OUTPUT

We have used the RGB LED module before and only managed to get 7 colors out of it. Using Analog control of a pin, we can control the level of energy going to pin; therefore, controlling the intensity of each LED element.

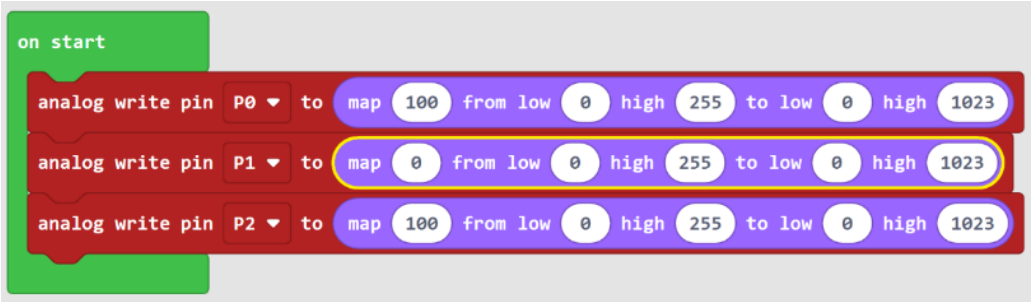
The connections are just like before.



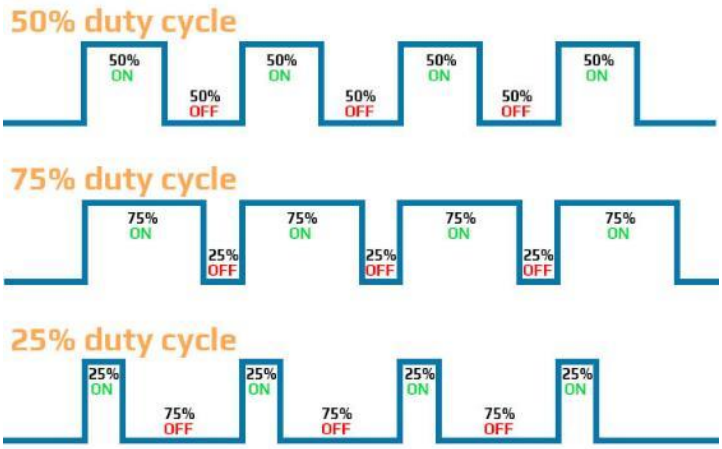
The analog write can be set to any value 0 to 1023.



The analog value allowed for analog write is 10bit in resolution, which is where the 1023 comes from. Two to the power of 10 is 1024, allowing for value 0 to 1023. If we are using a color picker tool, we need to scale/map the values from 255 max to 1023 max.



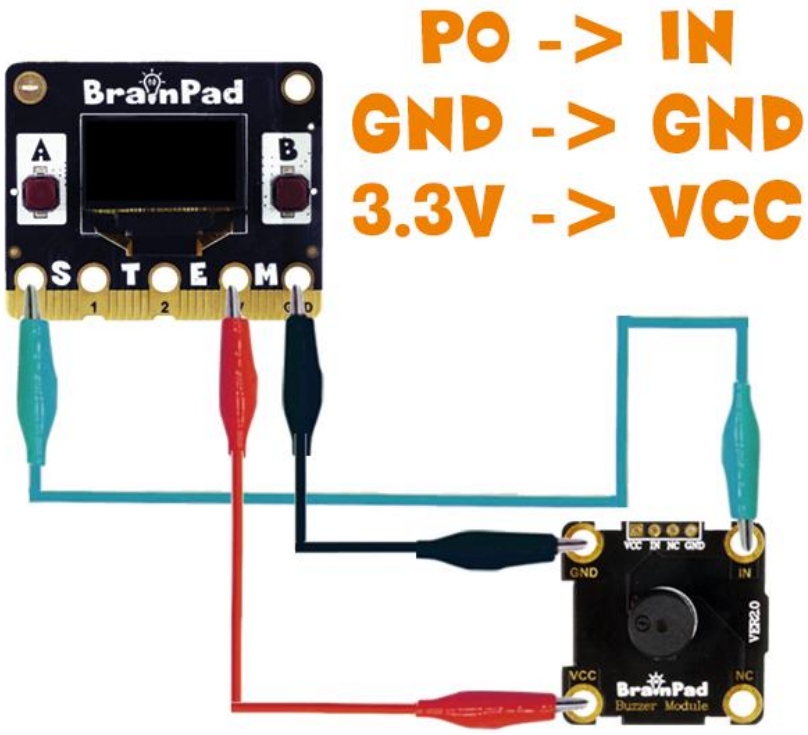
Generating sounds is also possible with Analog Output as this is nothing but a PWM signal. Here is what a PWM signal would look like.



Controlling the level of energy on a pin (intensity on a light) is done by turning a pin on and off very rapidly. Now if we keep the duty cycle at 50% (that is half 1023), then the pin will generate a basic square wave. We then can change the PWM frequency to generate different tones.

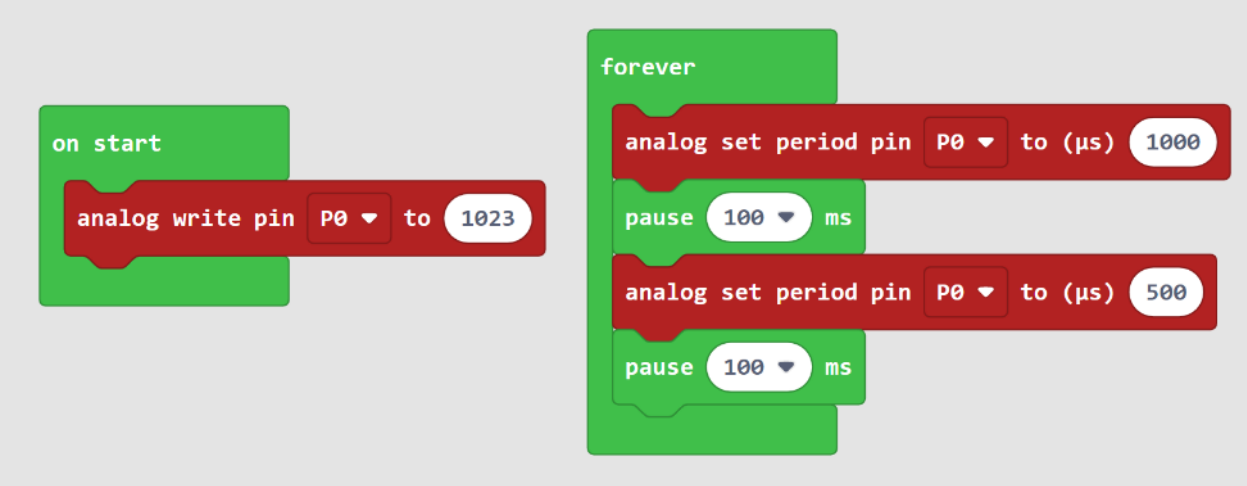
Let's use the buzzer module to demonstrate.

The connections are the usual, with IN going to P0.



You can also connect headphones or speakers using the 3.5mm connector. Connect ground to the outer ring, and P0 to the tip of the connector.

The code will set the duty cycle to half 1023 and then change the period. In case you didn't know, frequency = 1/period. Meaning 1000us (1ms) period equals 1Khz.

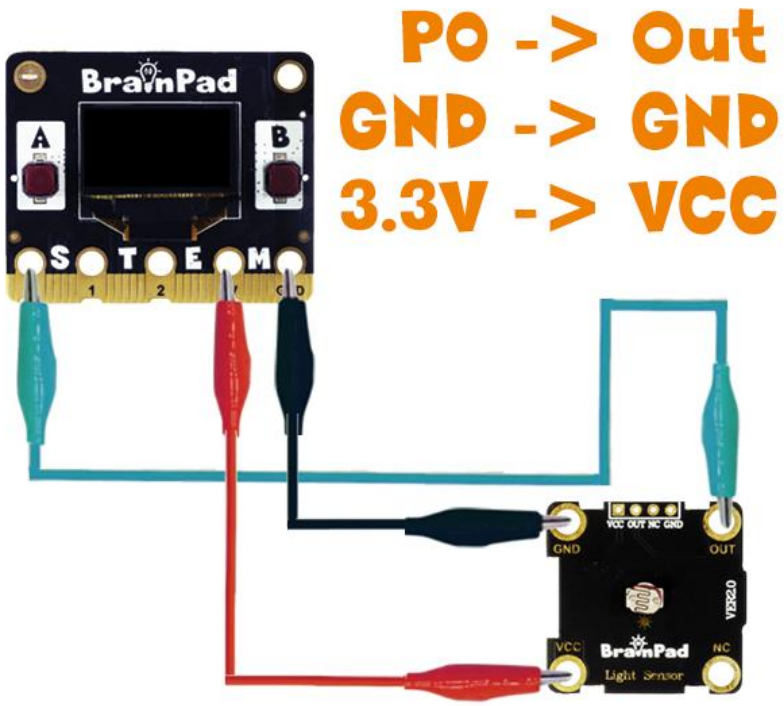


ANALOG INPUT

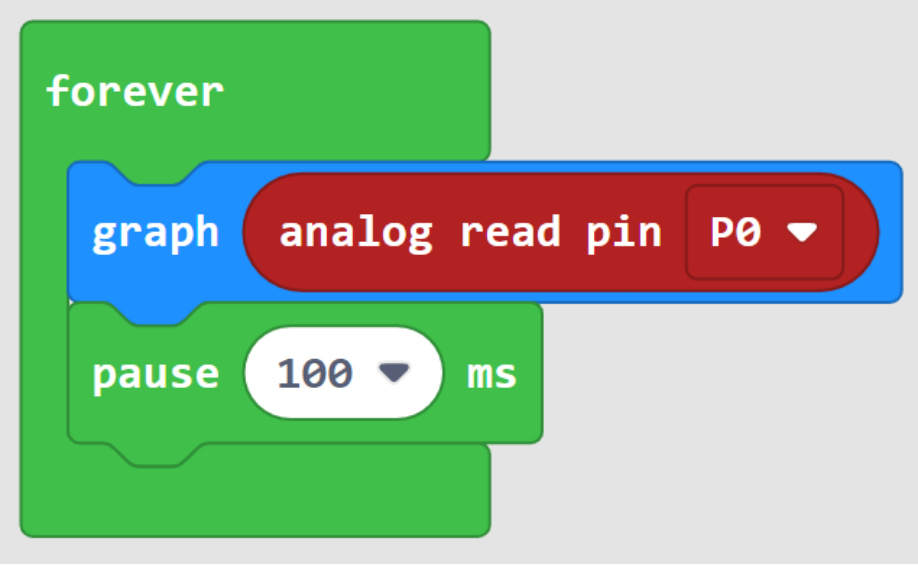
Digital input pins can only read high and low (one or zero), but analog input pins can read the voltage level. Analog input pins are called ADC for Analog to Digital Converter.

We will use the light sensor module with our first analog input example. This module outputs an analog value reflecting the ambient light level.

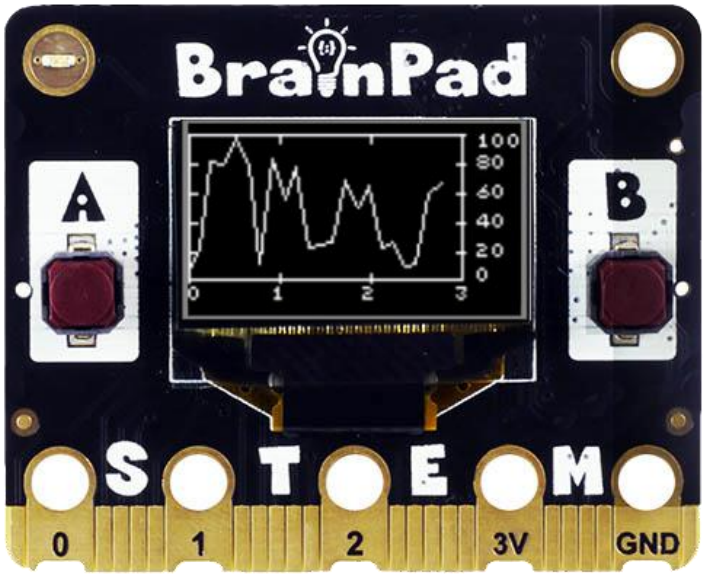
The connections are the usual, with OUT going to P0.



We can use the light level to detect it is nighttime and, therefore, turn the porch light on, for example. This example, however, will graph the light level.

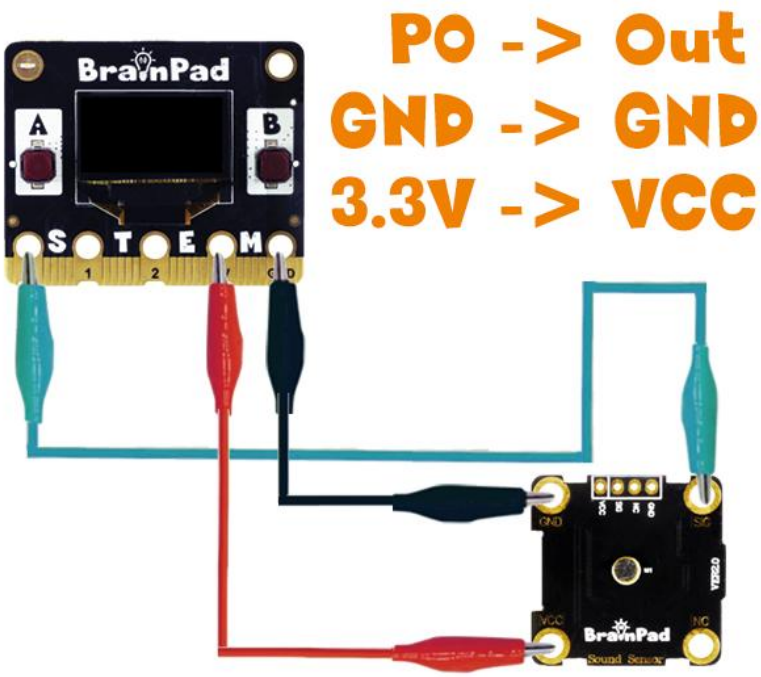


Change the light level around the sensor and observe the screen.

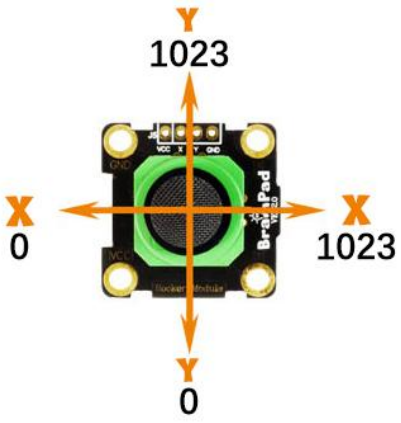


Keep the same program running but change the light sensor with sounds sensor.

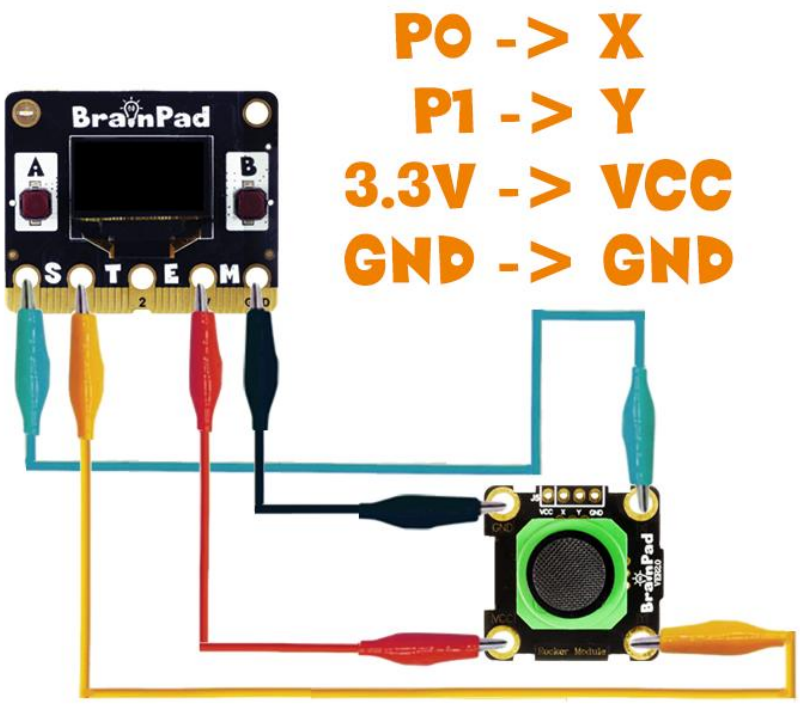
The connections are the same. Go ahead and start screaming!



The Rocker module has 2 analog outputs, for X and Y.

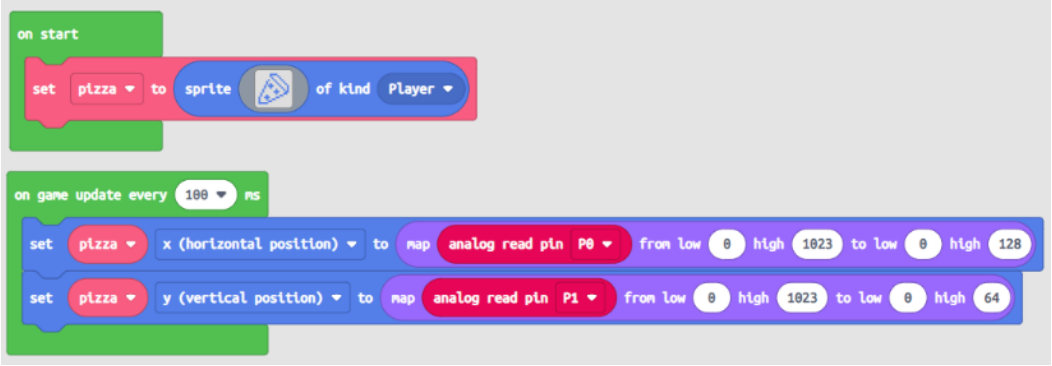


We will connect X to P0 and Y to P1.

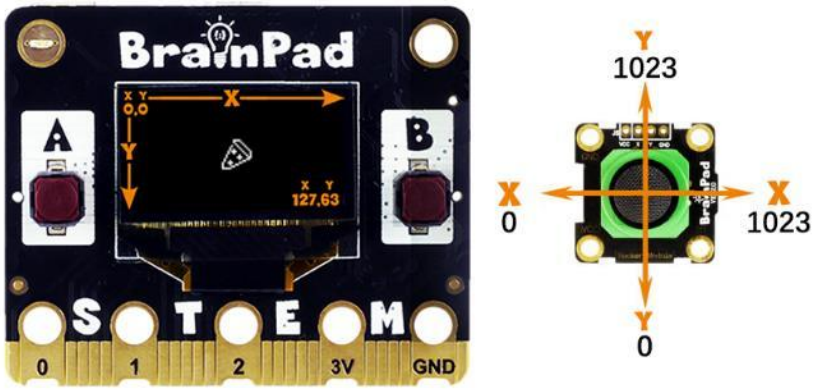


We can show both values on the display.

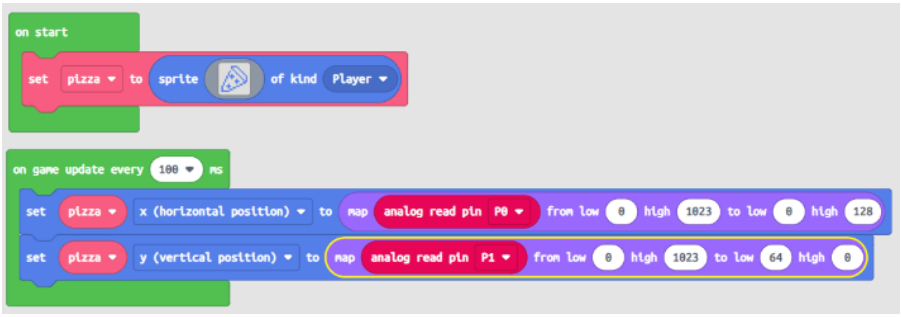
To use the Rocker to move a sprite, we use the map block to scale the values from the rocker to the size of the screen.



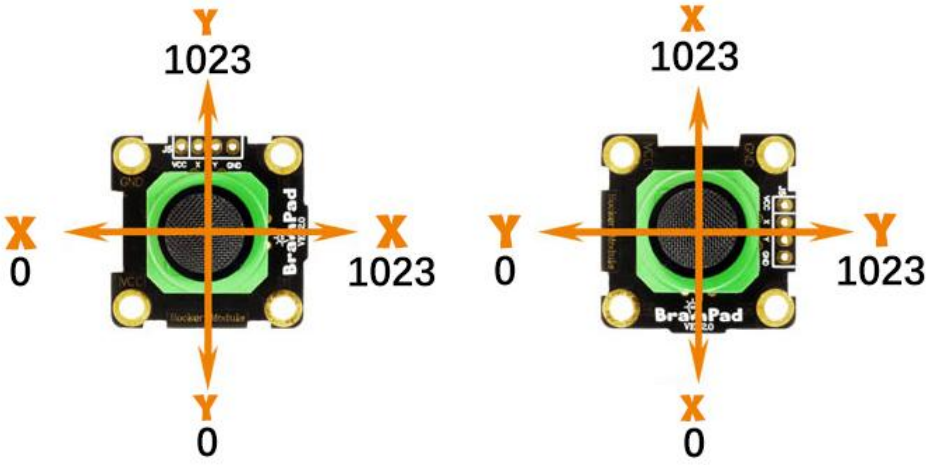
You will notice that the sprite moves left and right properly, but it is inverted in the Y (up and down) direction. This is because the display starts with zero location at the very top left, where the rocker starts with zero at the bottom.



To fix this you have 2 options. The first option is by changing the map to invert the Y value.



The second option is to just rotate the rocker and then swapping X and Y.

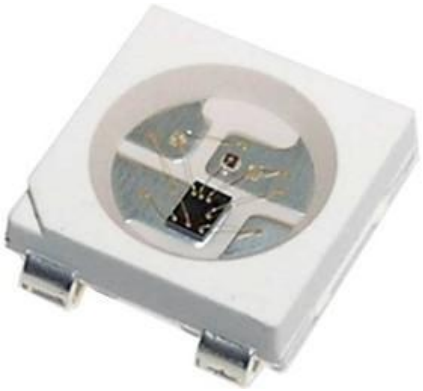


The code is now the same as original, but we swap axis by swapping P0 and P1.

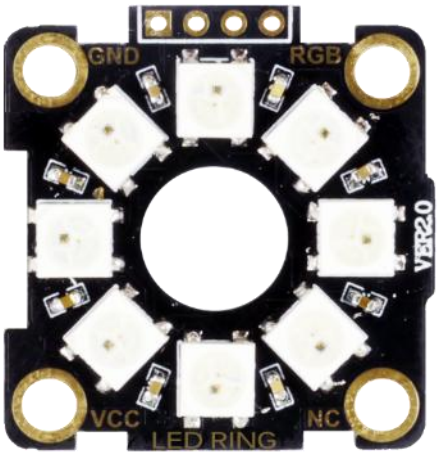
SMART LEDS

Consider this, every LED needs a pin to control it. Not only this, that pin needs to support analog put (PWM) to control the intensity of the LED. Color LEDs have three elements, so that is three pins. Having 10 color LEDs will require 30 pins! What if we need 100 LEDs? This is difficult and an impractical use.

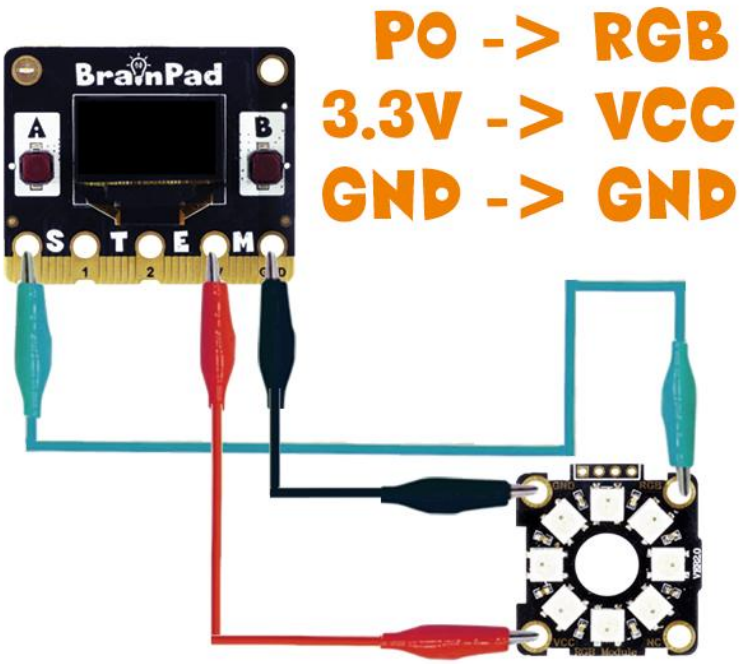
Smart LEDs have a built-in micro that controls the three LED elements. It uses a single pin to read a signal from a control circuit. Look closer and you will see the micro!



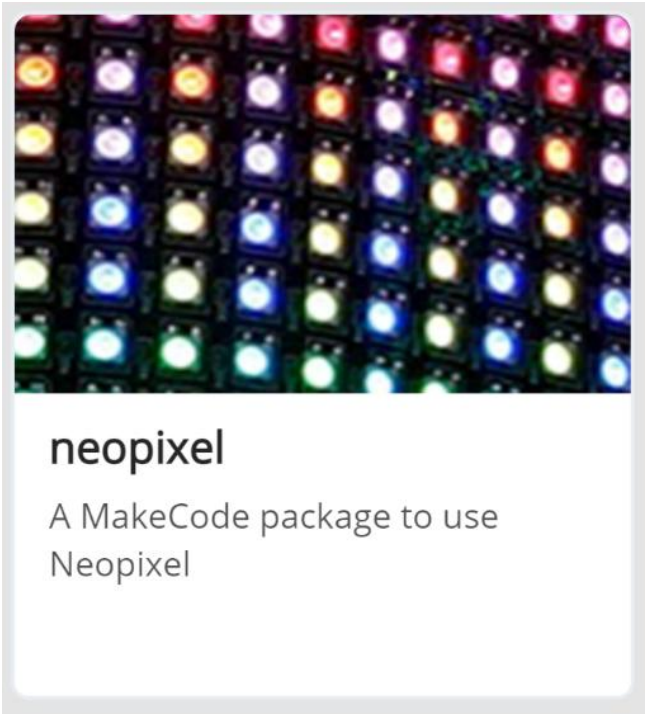
Those smart LEDs are chained and, therefore, they all can be controlled from a single pin on the controller. The “controller” in this case is BrainPad Pulse. The smart LED in the BrainClip kit is the LED Ring module.



The RGB pad is where the signal comes in to the 8 smart RGB LEDs.



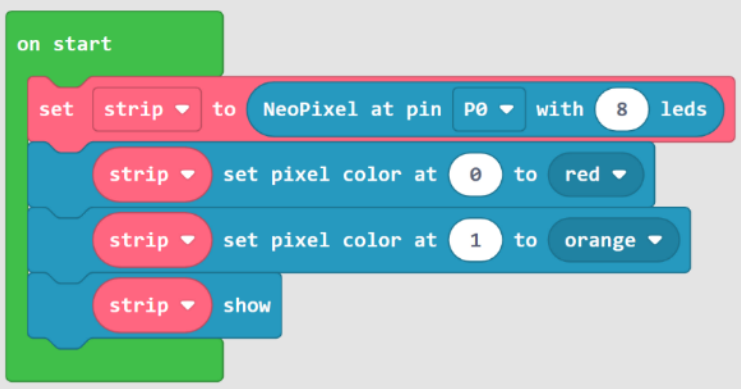
The special signal that needs to be generated to control those smart LEDs is found in the neopixel extension. Go to Advanced options under the block menu and click on Extensions. Now add the neopixel extension.



A new set of options will be presented.

To initialize the driver, add to on start and set the length of the LED strip. We have 8 on the LED Ring.

We can set any of the 8 LEDs to any desired color! This example will set the first LED to red and second to orange.



Note how the show block is required. Set pixel color block only tells the driver what we want, but it does not send the signal to the LEDs yet. Only when the show block is used the signal will be sent.

Can we use random numbers to create a lightshow? We have 8 lights, from 0 to 7, and then we need a random 0 to 255 for the three primary colors. This is going to be a very long block!

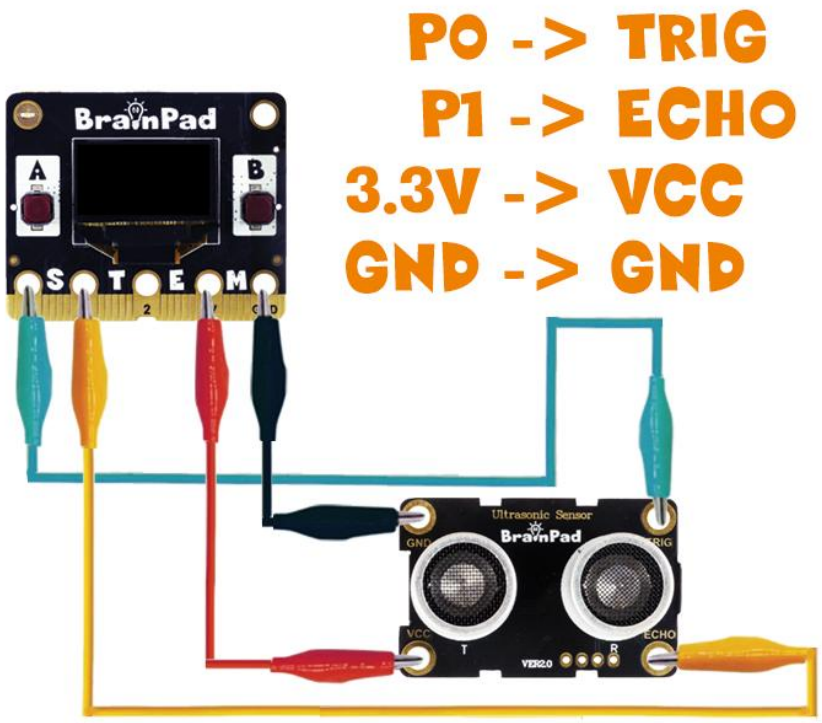
This is a good place to point out that blocks are simple but not always practical in real-life coding applications.

DISTANSE SENSOR

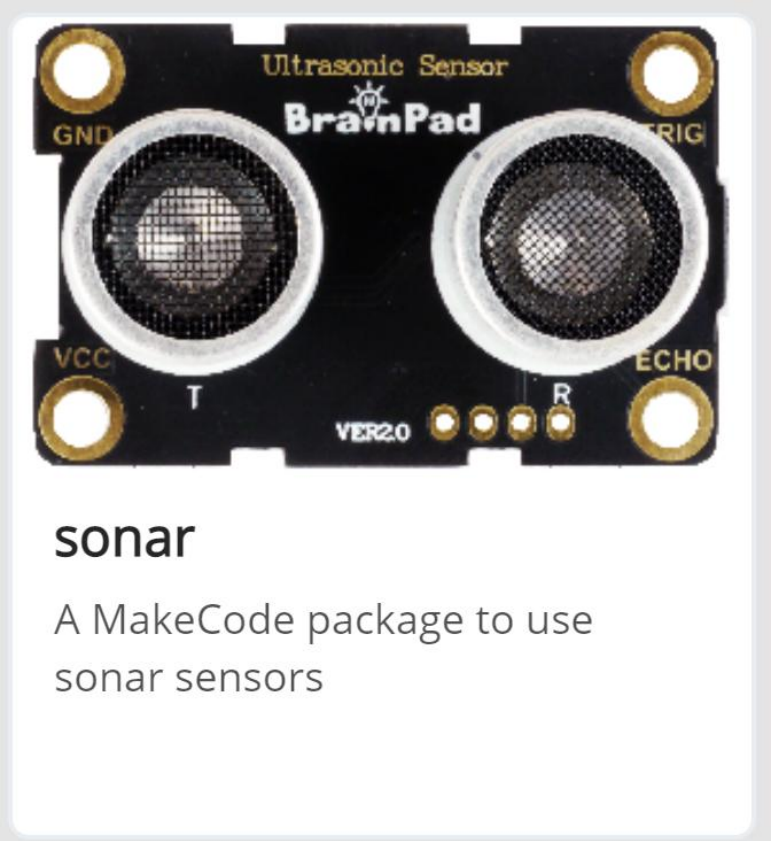
There are many ways to measure distance, but the most common sensor uses ultrasound to measure distance by sending a pulse and measuring the time needed for the response to come back. The pulse is of a higher frequency than we can hear and so the sensor seems quiet, but it is not!

BrainClip kit includes an Ultrasonic Sensor module.

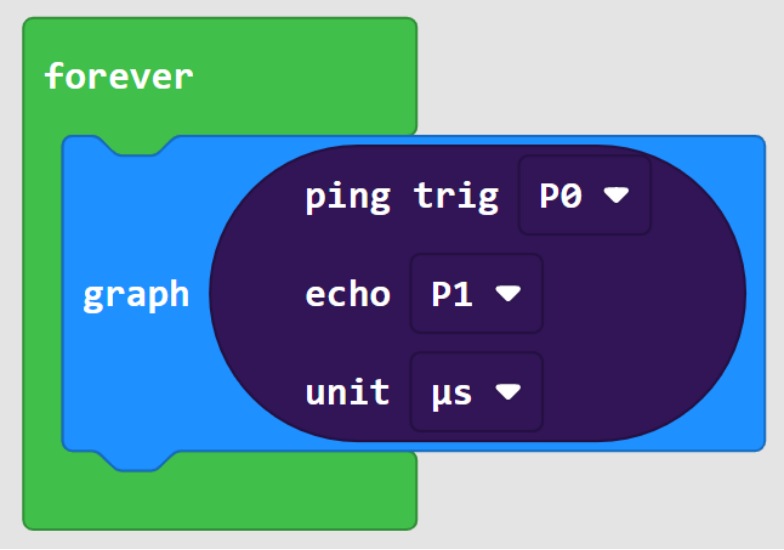
Sending a pulse starts with TRIG pin, the trigger. The response with the distance comes back on the ECHO pin. We will use TRIG on P0 and ECHO on pin P1.



The driver for the ultrasonic sensor is in an extension. Add the sonar extension like we did with other extensions before.



We can now graph the value on the display.

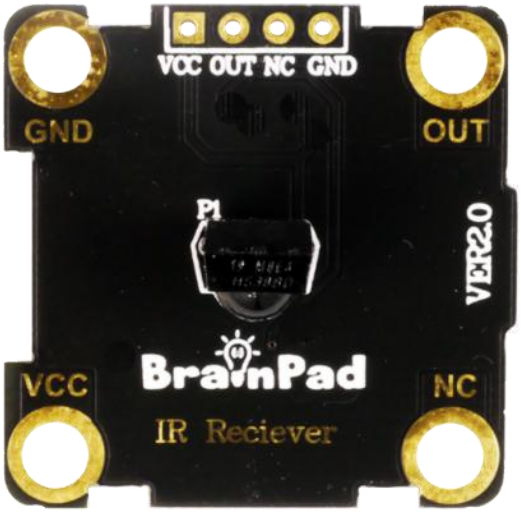


Keep in mind the sensor uses the sound reflection to measure distance. It will, therefore, work best with solid and flat surfaces.

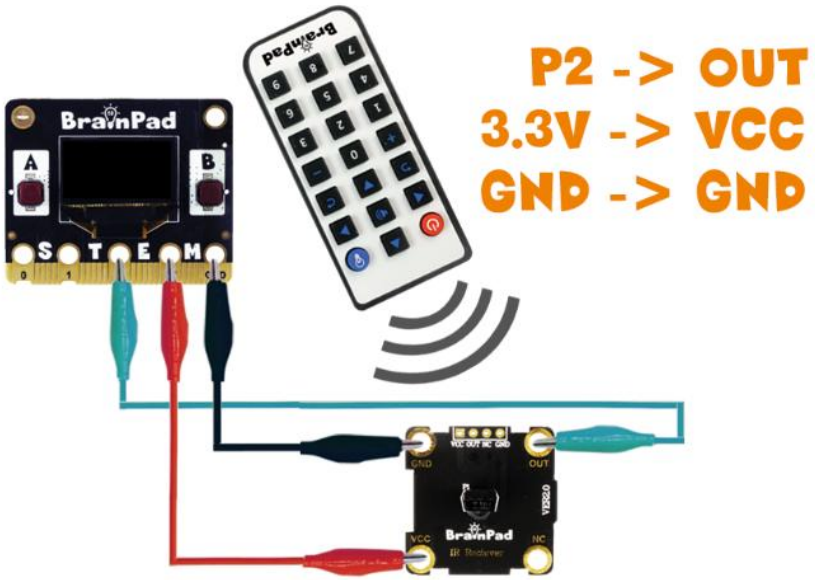
INFRARED REMOTE

InfraRed (IR) Remote controls are old technology that are still in use today due to their simplicity. The remote sends a pulse of IR light that we do not see, then an IR receiver detects those pulses and uses them to decode a meaningful signal. TinyCLR includes an NCR standard signals decoder. This works with the remote found with BrainClip and the BrainBot kits.

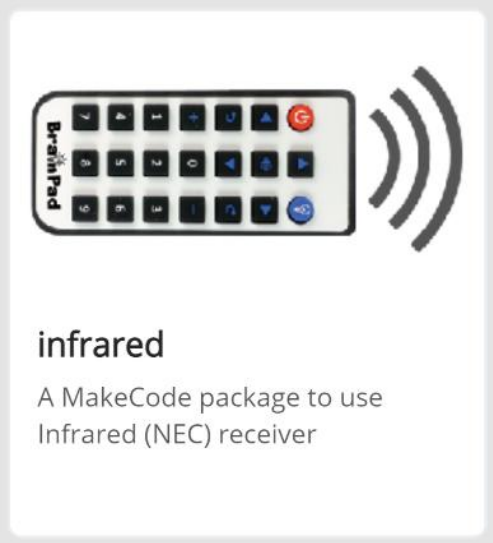
To read/see the IR signal, we need the IR Receiver module.



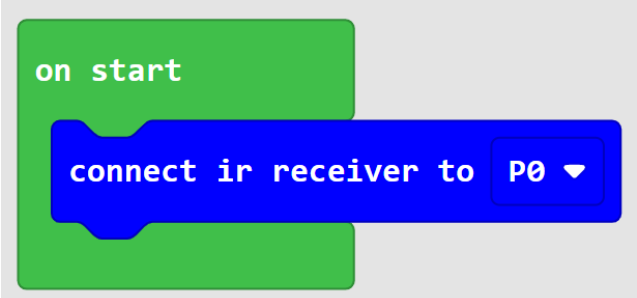
The OUT pin is the signal that needs to be decoded.



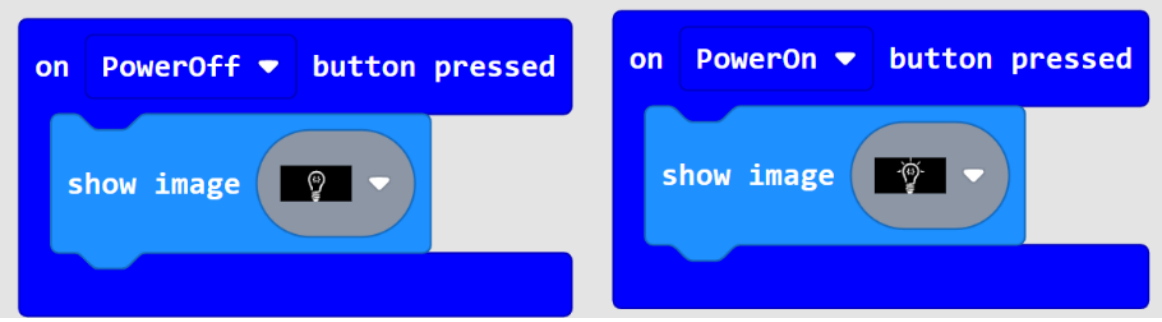
To decode IR signals, we need the infrared extension.



The driver needs to know where the receiver is connected.



An event is raised when a button is detected on the remote. We can use the power on and power off buttons to control the lightbulb on the screen.

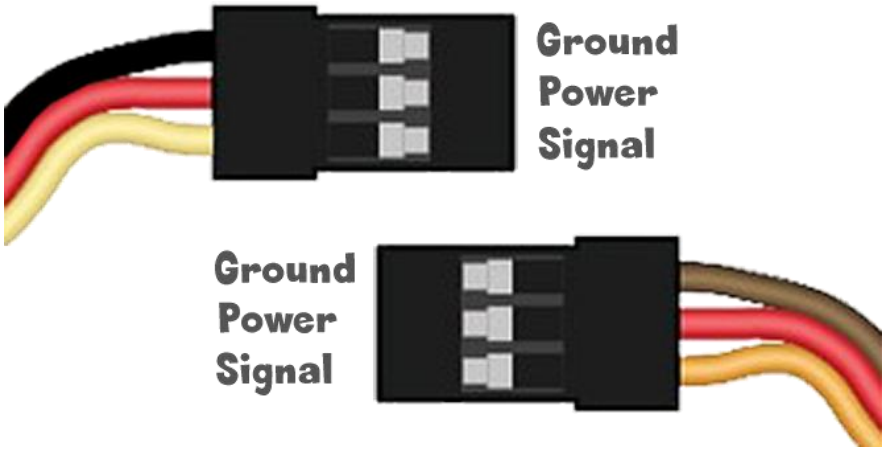


SERVO MOTORS

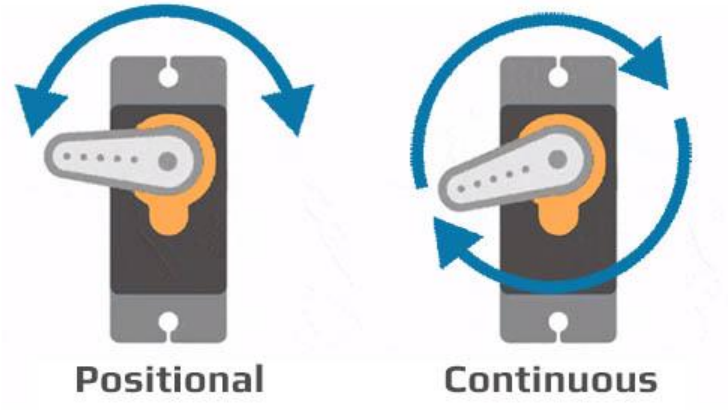
BrainClip kit does not include a servo motor, but servo motors are very common.

Traditional motors convert electrical energy to mechanical rotation. When it comes to speed, the higher the voltage, the faster the motor. Servo motors are a bit smarter! They take a voltage (power) for energy but also take a signal that comes into the servo to tell it what we want it to do. This page covers the RC Servo motor, originally made for RC (radio controlled) hobby vehicles. On RC Servos, the signal coming in is a pulse that repeats 50 times per second, that is every 20 milliseconds. The incoming pulse size can vary between 1 and 2 milliseconds typically, and then repeats every 20 milliseconds. The smart circuit inside the servo sees the pulse and then controls the internal motor based on the pulse size.

The three wires on a servo are usually Black, Red, and Yellow. The Red and Black ones are for power, where Red is positive (power source) and Black is negative (ground). The Yellow wire is the signal. Some other servos have Orange, Red, and Brown. Where Red and Brown are the power and Orange is the signal. If in doubt, always remember that the middle pin is always the power source, and then the negative pin (the ground) is the darker color, like Brown or Black.

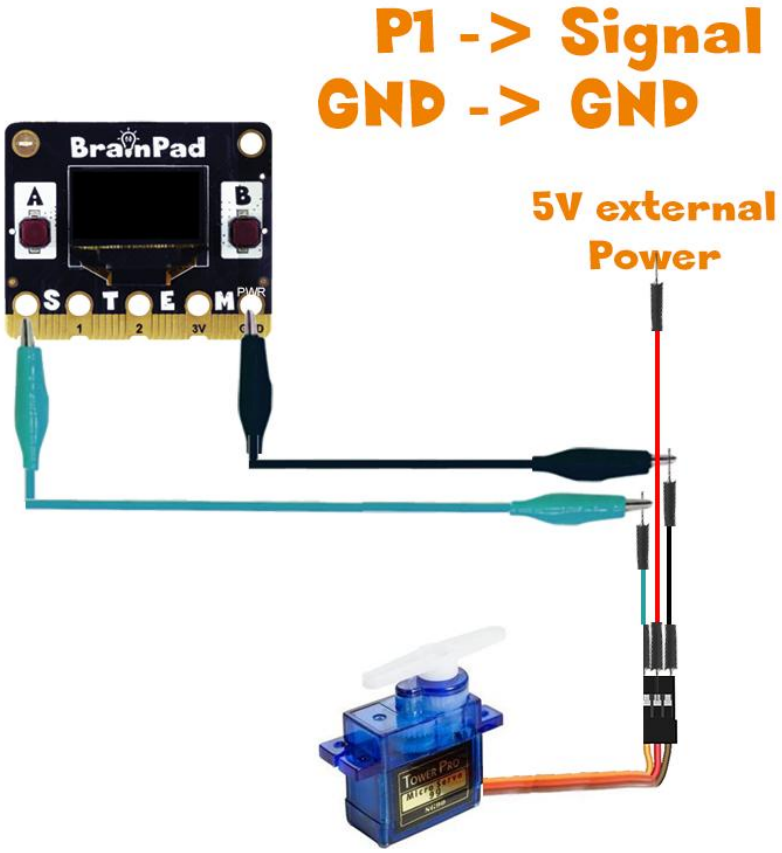


There are two types of servo motors, positional and continuous.



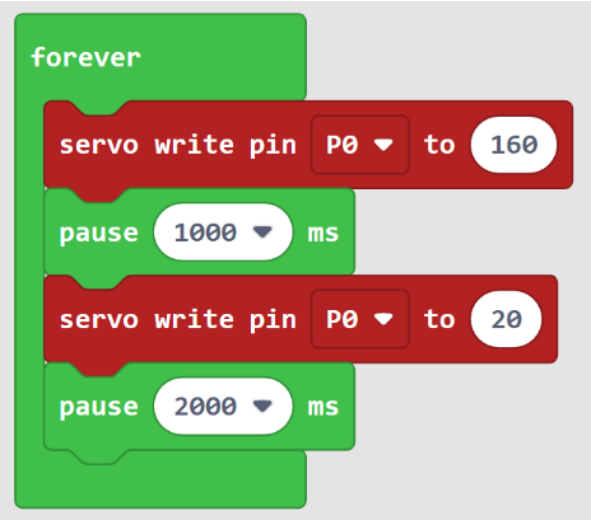
As name suggests, continuous servos keep on rotating, and the positional servos move to a specific position. For example, a positional servo is used to steer the wheels on a car robot or to turn the camera to a specific direction.

Servo motors generally need 5V to operate. You can use the VIN pad on the BrainPad Pulse, in the top right corner, to draw 5V directly from the USB cable. This is good for powering 1 servo motor. If you need more motors for a robot or something more complex, a dedicated power source is necessary. To wire a servo to the BrainPad Pulse we'll need to use pin to pin wires to use inside the servo's plug. Wire as shown in the diagram.

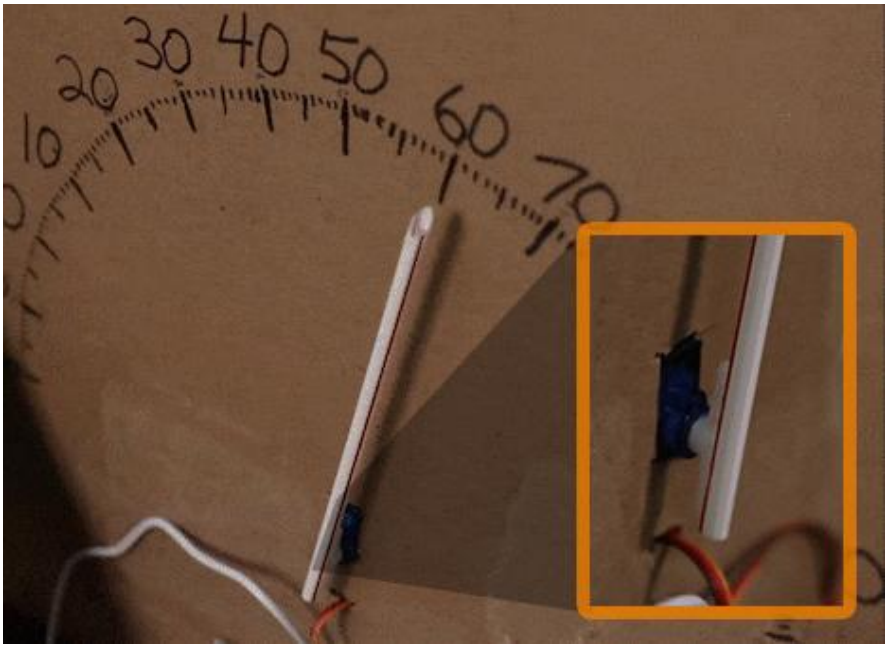


The "servo write" block is found under pins, under advanced. The block accepts 0 to 180 degrees, which is the angle of where the servo will be pointing.

A loop can send the servo back and forth indefinitely.



A good example and an easy use of a servo is in creating a very large gauge that shows a value. Use a straw or a pin that is mounted on the servo as a dial that points to a value printed on a cardboard.



WHAT'S NEXT?

Microsoft MakeCode and the BrainPad Pulse make a great combo to learn programming. The simulator means you can easily test code without even having a BrainPad. Then you can load the program on the BrainPad Pulse by simply copying the downloaded program file to the BrainPad window.

Start using what you have learned to build your own projects. If working with students, you can use the examples in this guide to create your own lesson plans.

For those interested in going beyond MakeCode, there are professional coding options available for the BrainPad. Visit the website to learn about the possibilities, www.brainpad.com .

www.BrainPad.com