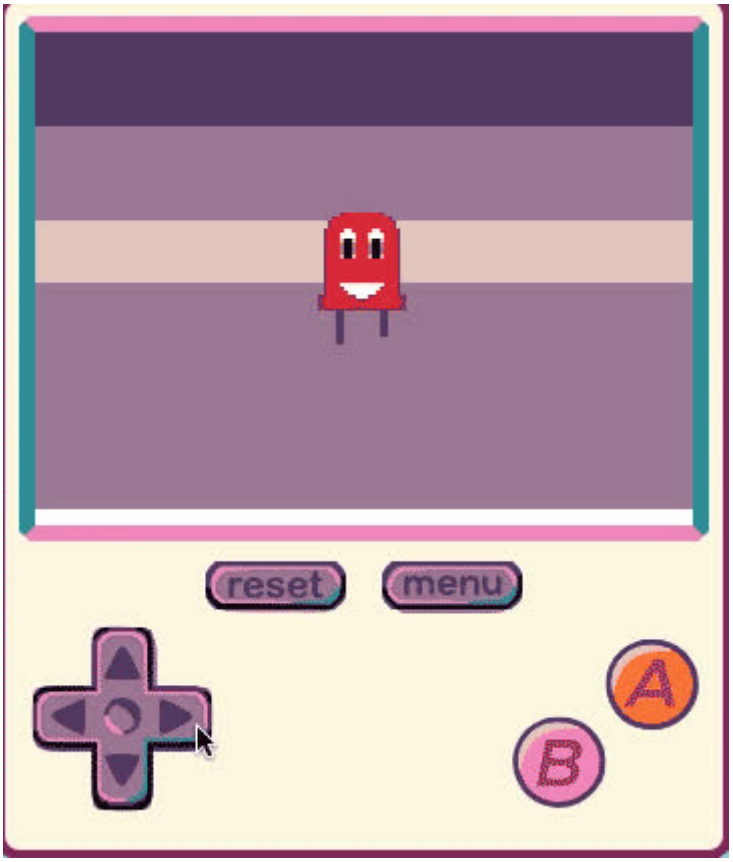
MakeCode Arcade Pixel Animation

Created by John Park

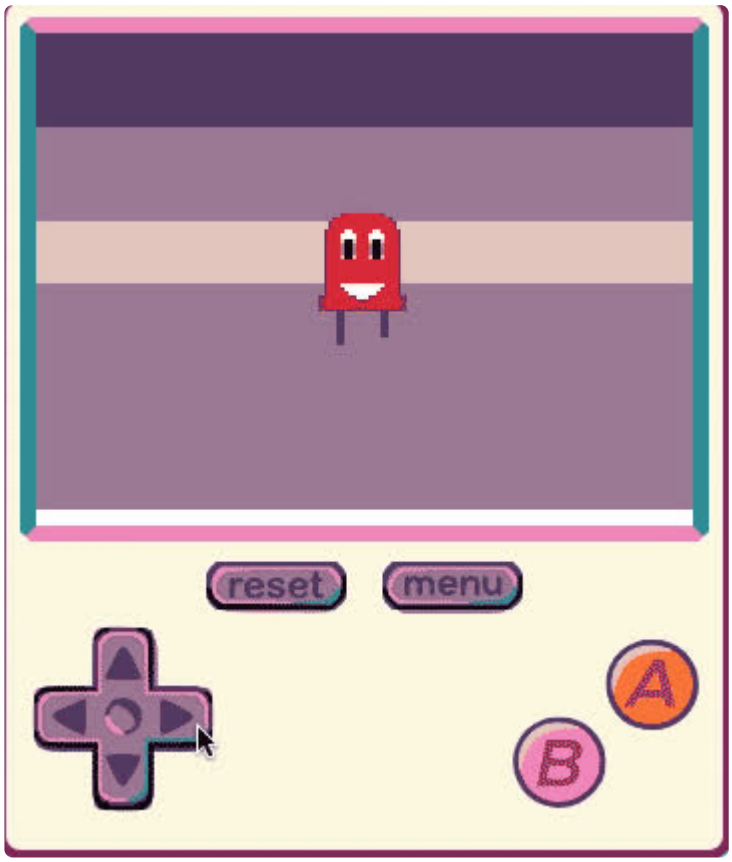


https://learn.adafruit.com/makecode-arcade-pixel-animation

Last updated on 2024-03-08 03:26:43 PM EST

Table of Contents

Overview

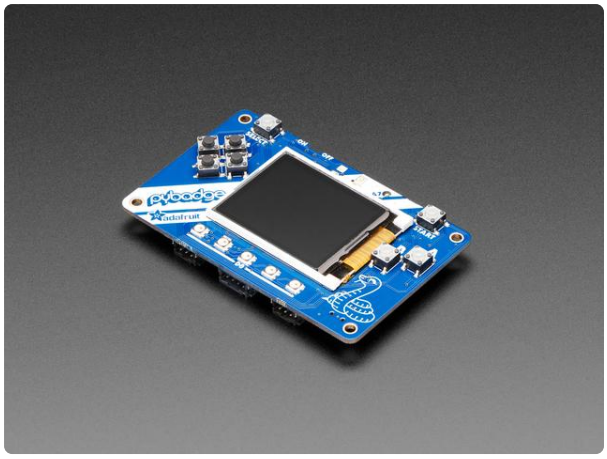


You can bring your MakeCode Arcade characters to life through the magic of animation! With just a few simple frames of pixel artwork and the right MakeCode blocks, you can make your player sprite react to actions in fun ways!

This guide will show you how to create frames of sprite animation, and how to set them up to play back depending on the game conditions.

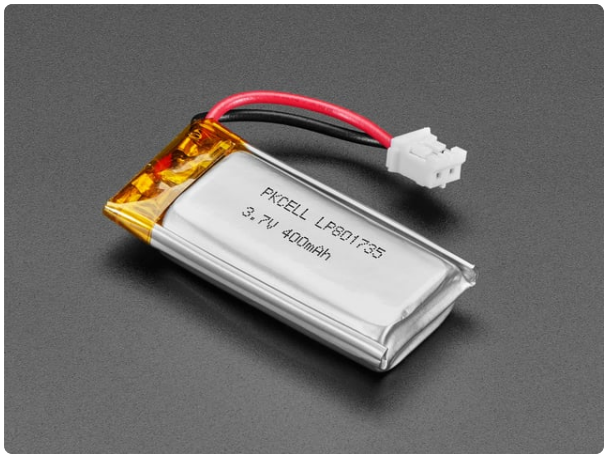
Parts

You can create this project entirely in the browser and test it using the MakeCode simulator, but it's even more fun to see your work come alive on hardware, such as the PyBadge!



[Adafruit PyBadge for MakeCode Arcade, CircuitPython, or Arduino]
What's the size of a credit card and can run CircuitPython, MakeCode Arcade or Arduino? That's right, its the Adafruit PyBadge! We wanted to see how much we...
https://www.adafruit.com/product/4200

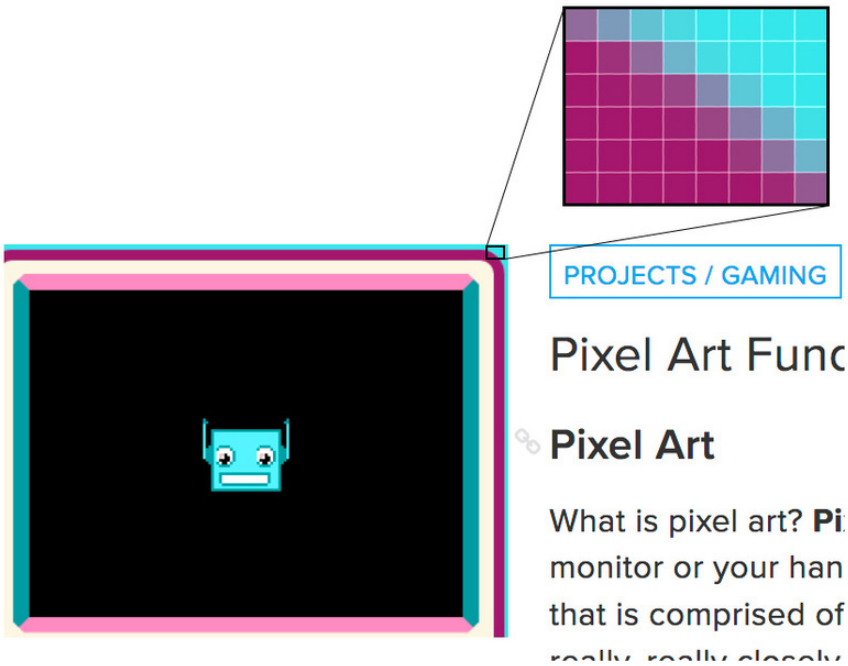
[Pink and Purple Braided USB A to Micro B Cable - 2 meter long]
This cable is super-fashionable with a woven pink and purple Blinka-like pattern! First let's talk about the cover and over-molding. We got these in custom colors,...
https://www.adafruit.com/product/4148



[Lithium Ion Polymer Battery Ideal For Feathers - 3.7V 400mAh]
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/3898

Sprite Animation Fundamentals

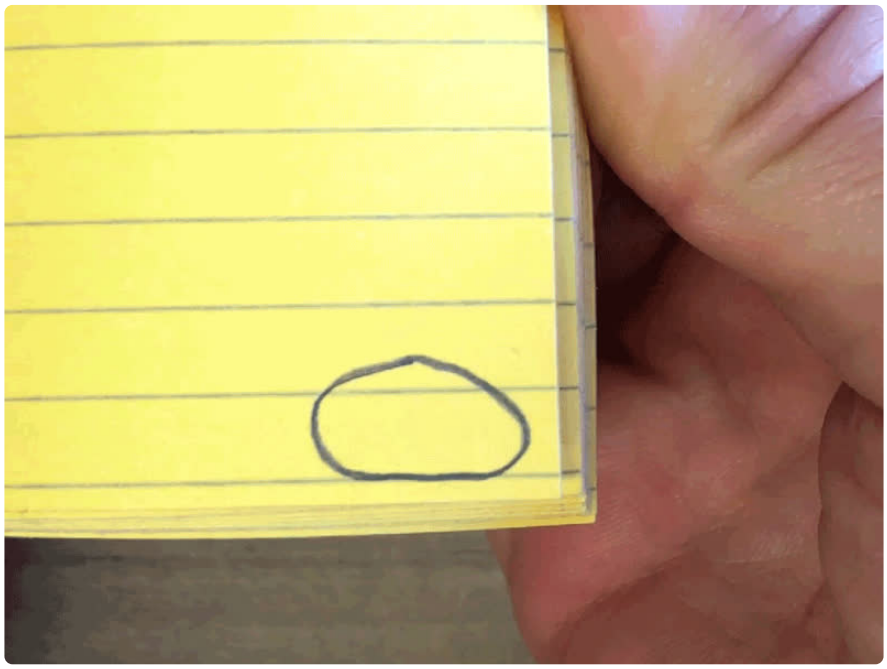
Before we get started with sprite animation, it's important to have a solid foundation in static pixel art creation. If needed, check out this page on Pixel Art Fundamentals (https://adafru.it/EI8) first.



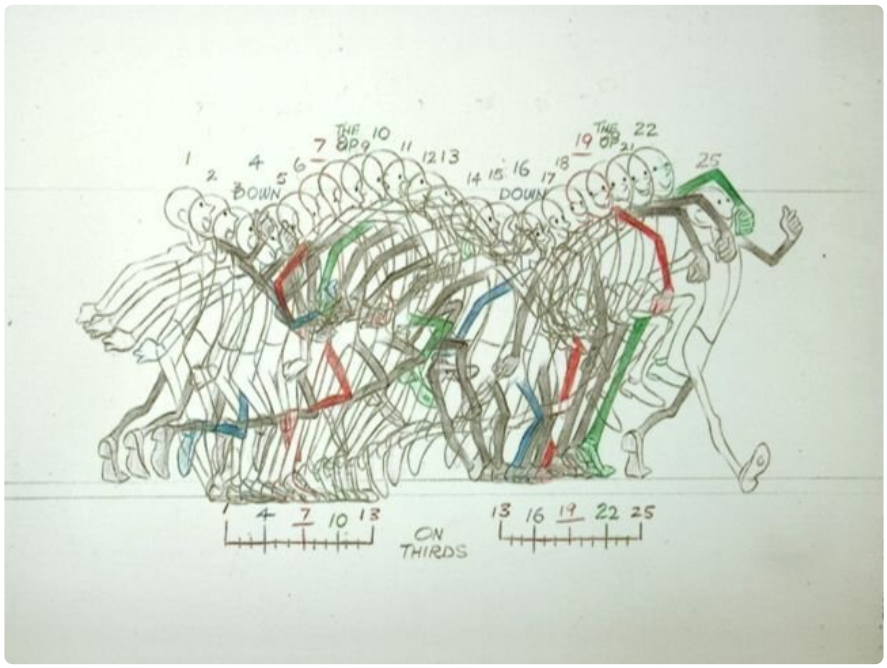
Animation Basics

Animation is a HUGE topic! It can be overwhelming, in fact, to consider all there is that goes into creating beautiful animation. The good thing is that we can keep it very simple and still learn a lot and get going fast.

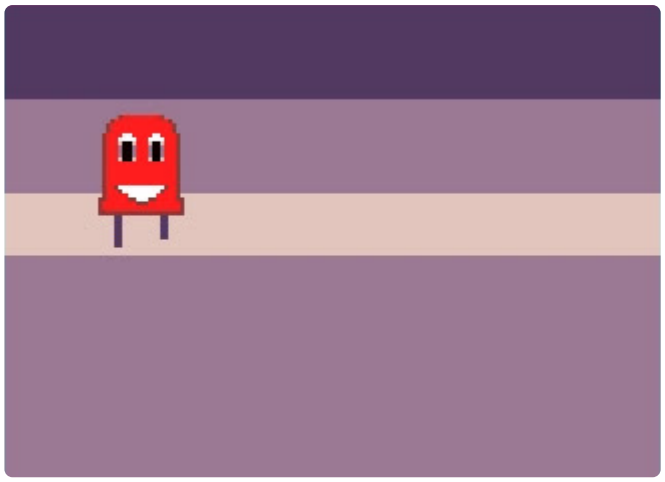
At its most basic, animation is the illusion of motion our brains trick us into perceiving when we view multiple still frames of artwork played back in quick succession. A flipbook you can doodle in the corner of your notebook is a great example of how this works and how simple it can be.



Typical film animation is created with somewhere between 12 and 24 unique frames per second, which can give it an incredibly fluid look, but is a heck of a lot of work, too!



We can get away with using just two unique frames of animation and still perceive the effect of animation, particularly when we're also using the game engine to move the sprites across the screen as they are animating!



Game Animation Frames

Pixel sprite animation used in games is normally created with many fewer frames of animation. This means we can typically get away with creating just a handful of unique sprites to convey the action.

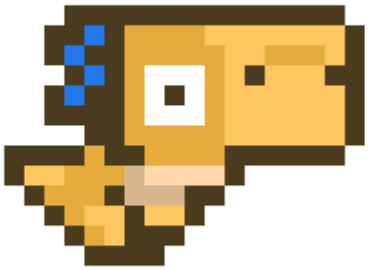
If a one second action of a character swinging a sword is comprised of 24 frames in a feature film animation, the same action may contain just three frames/sprites in a very simple game style, and up to nine or so frames for a more fluid style.

Here's a beautiful example from Pedro Medeiros's MiniBoss Studios tutorial set available here (https://adafru.it/EI9).

Cycles

Unlike a feature animation, nearly everything a character does in a game (excluding cinematic cut scenes) can be broken down into repeating cycles. Walking, running, jumping, standing idle and breathing, swinging a sword, grabbing a prize, and on and on. These are all actions that can be considered looping cycles.

Here are some wonderful, short cycles created by Arks (https://adafru.it/EIa) for his Dino characters. (You can purchase the sprites here if you like (https://adafru.it/EIb) and use them in your own games, or even just to study them.)



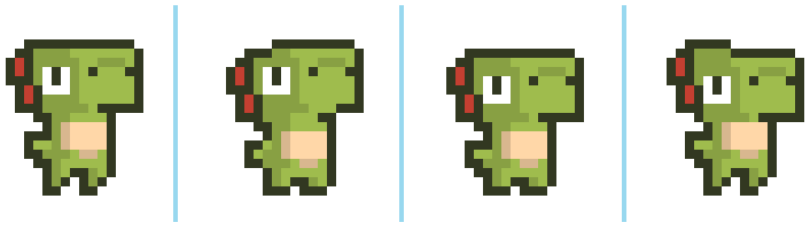
This means we can focus on creating individual cycles, and then worry later about setting up the game engine to call on the appropriate cycle depending on what's going on in the game.

Player is moving the d-pad to the right? Play back the walking cycle. Player presses the 'A' button? Play the jump cycle. And so on.

Let's look at a typical, simple cycle, the "idle animation".

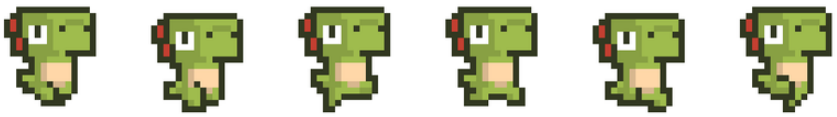
Idle Cycle

An idle cycle is an animation loop that plays back when the character is standing still. Rather than just holding on a single sprite, which can be a bit boring, an idle animation can be as simple as a few frames of breathing or blinking, and can get as complex as yawning, looking around, head scratching, etc.



Walk Frames

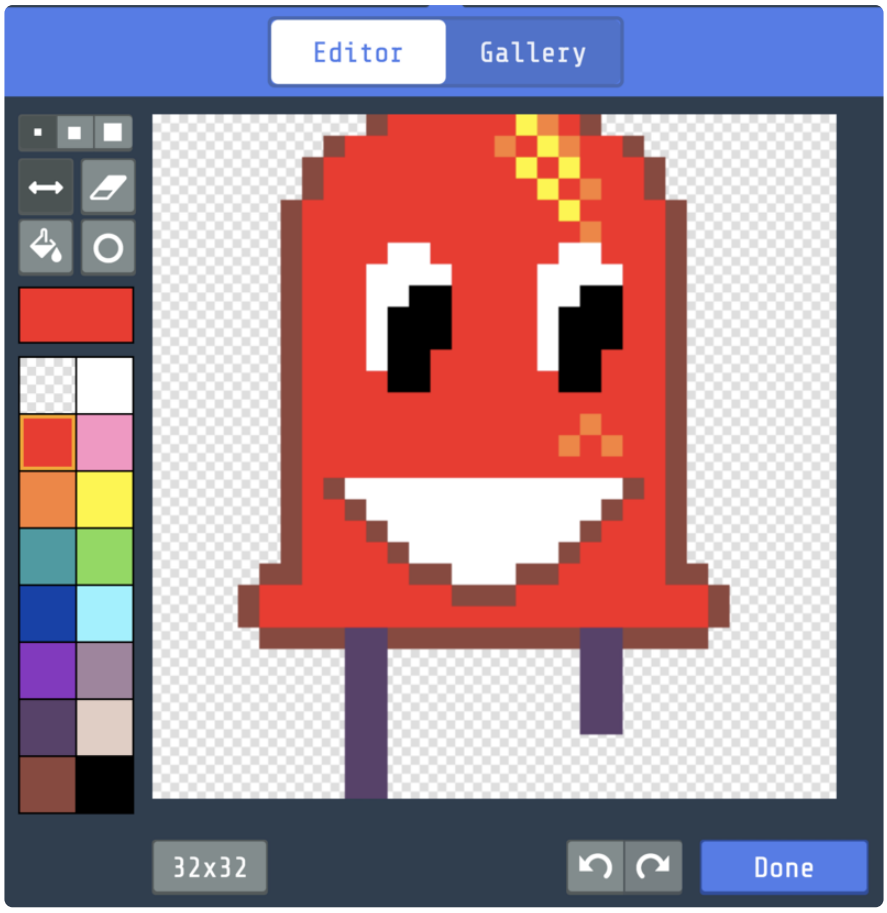
Here's a walk cycle made of six frames:



Now, let's create our own animation cycles for Ruby, the red LED!

Create Sprite Animation in MakeCode

If you're not familiar with creating static sprites inside of MakeCode already, have a look at this guide page (https://adafru.it/Elc) and then come back here to create animation.



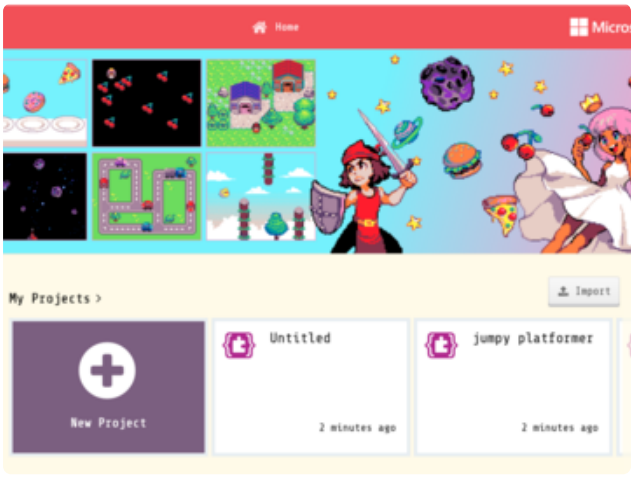
First, go to MakeCode Arcade and create a new project.

Next, click on **Advanced** > **Extensions**

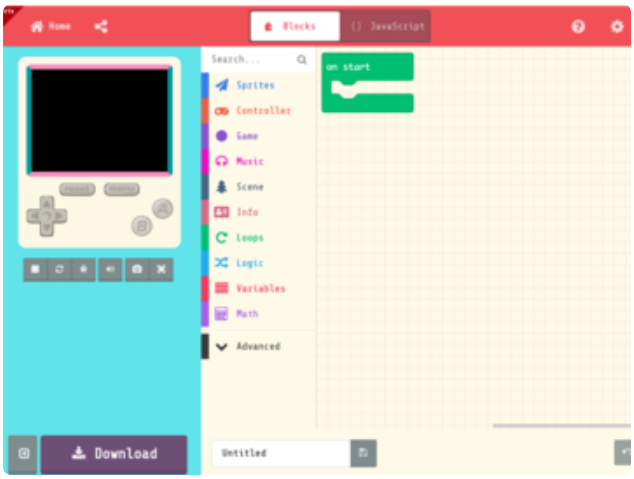
click the **Animation** extension to add this library.

MakeCode Arcade

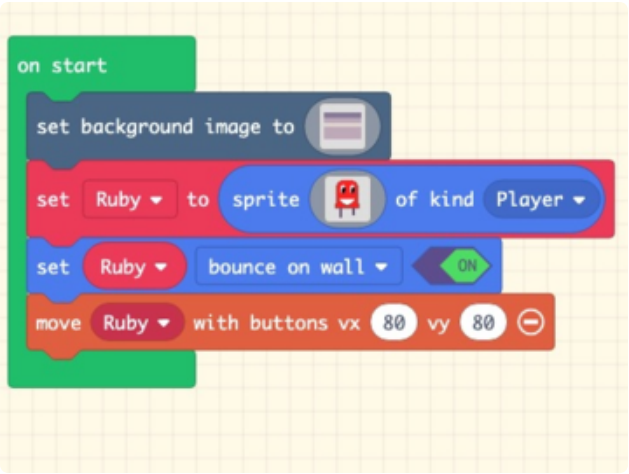
First, open a new Chrome browser window (Chrome works best) and go to MakeCode Arcade beta (https://adafru.it/EQg).



Next, click on the **New Project** button.

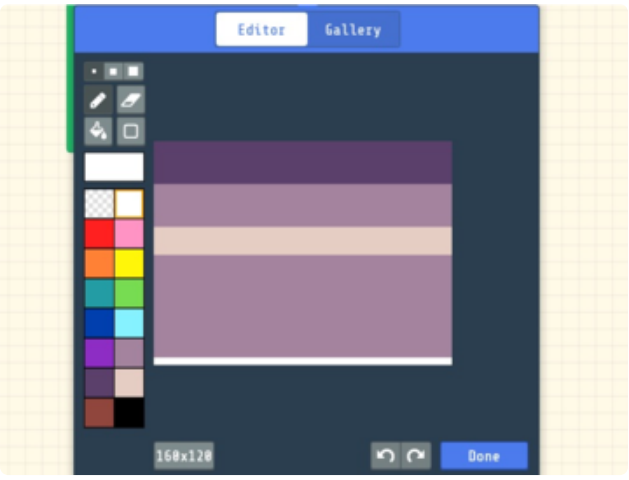


This will take you to the MakeCode Arcade Editor.



Scene Setup

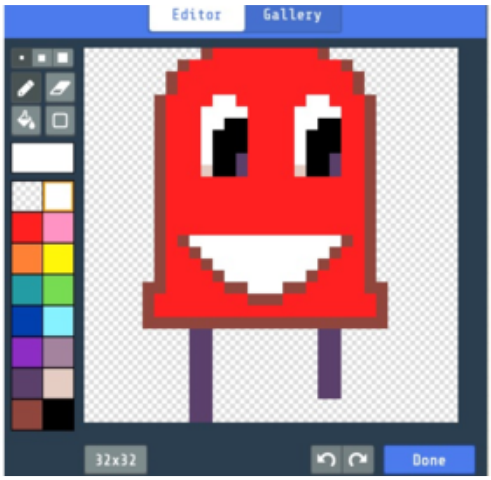
Create a basic scene as we did in the pixel art tutorial including:



Background image such as the one shown here

Player sprite named **Ruby**

Ruby set to **bounce on wall**

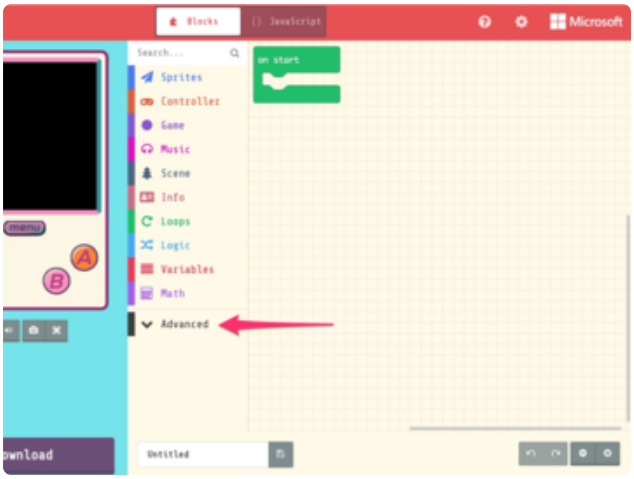


Ruby controlled with buttons and a velocity of **80** on **vx** and **vy**

If you would like help getting the scene set up in MakeCode to this point, you can head to this link to check it out (https://adafru.it/Eld).

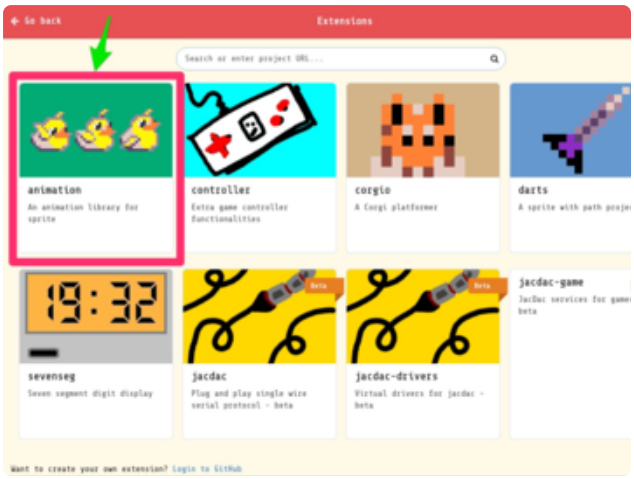
Animation Extension

In order to create animation for our player sprite, we'll need to add the **Animation** extension to MakeCode Arcade. You can think of this as a set of helpers or libraries that add animation related blocks to the editor.

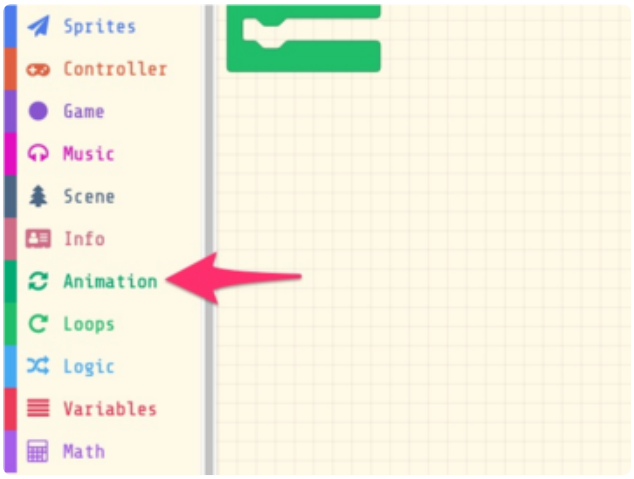


In order to load the Animation extension, first click on the **Advanced** category to expand the category list.

Next, click on **Extensions.** This will bring up the Extensions selection window.



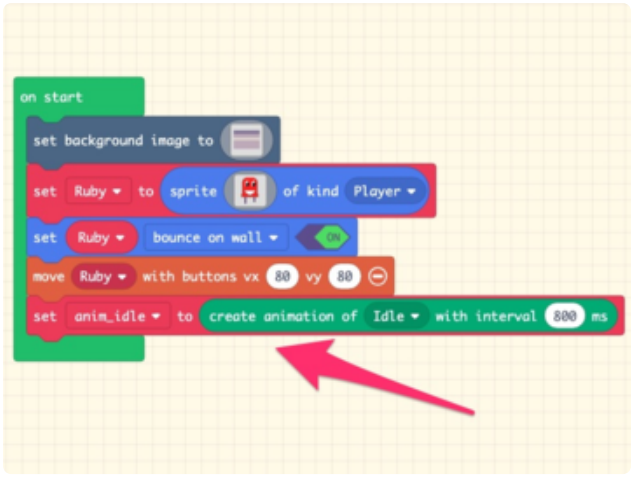
Click on the **Animation** extension item and it will load the animation blocks into the editor session.



Animation Blocks

Now, you can click on the **Animation** category.

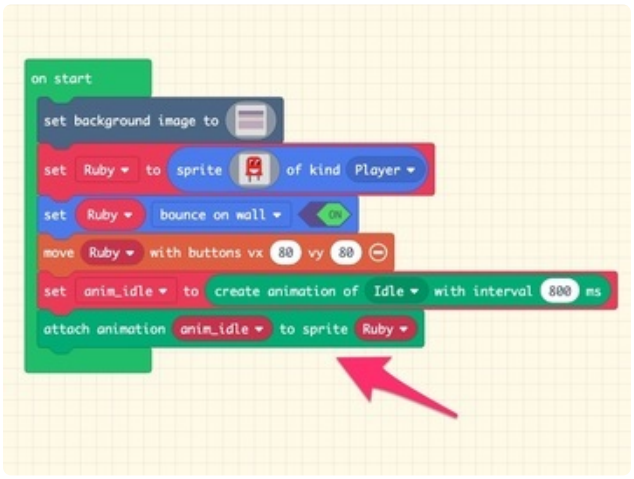
In it you'll see the first block we'll need to add, the **set anim to create animation of Walking with interval** block. Add it to the existing **on start** block.



The first animation we'll add is an idle animation, which is what Ruby will do when there is no controller input. Rename the **anim** variable in the block you just added to **anim_idle,** and also rename the **Walking** variable in the **create** section of the block to **Idle.**

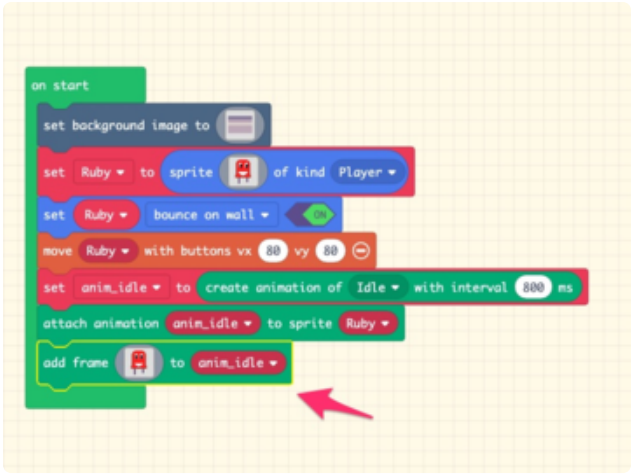
We can also set the interval to run each frame of the animation a bit quicker than one per second. You can always fine tune this later to your tastes, once you have frames to test. I ended up setting mine to **800ms**.

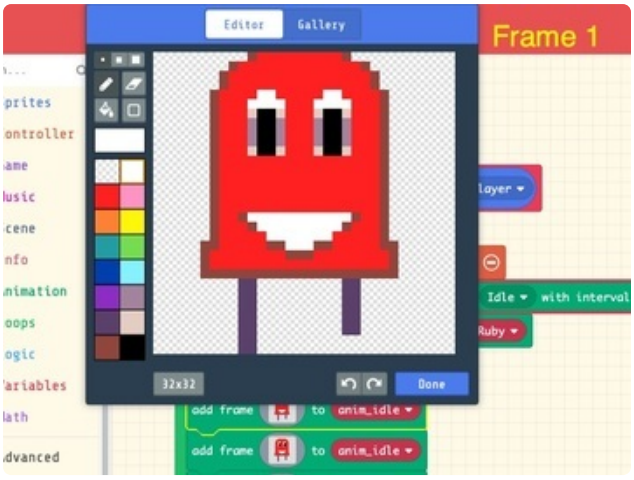
Attach Animation



The animation block we created needs to be attached to a sprite. You may have many sprites in your game eventually, so this step ensures that the game engine knows which animation loops are associated with which sprite characters/ enemies/food, etc.

Add an **attach animation** block to the **on start** loop. From the dropdown lists choose **anim_idle** as the animation variable name and **Ruby** as the sprite.





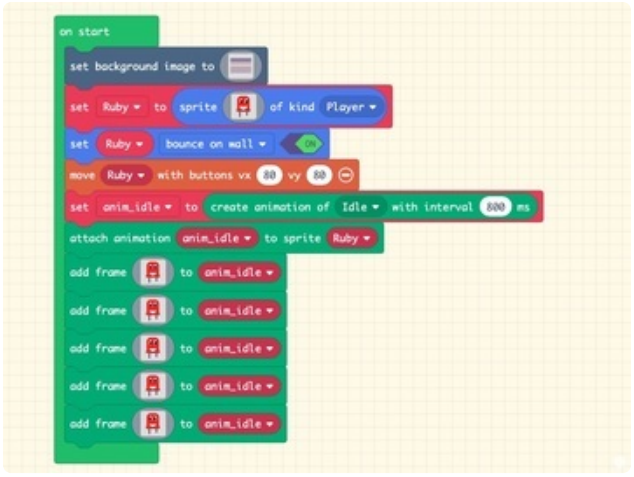
Add Animation Frames

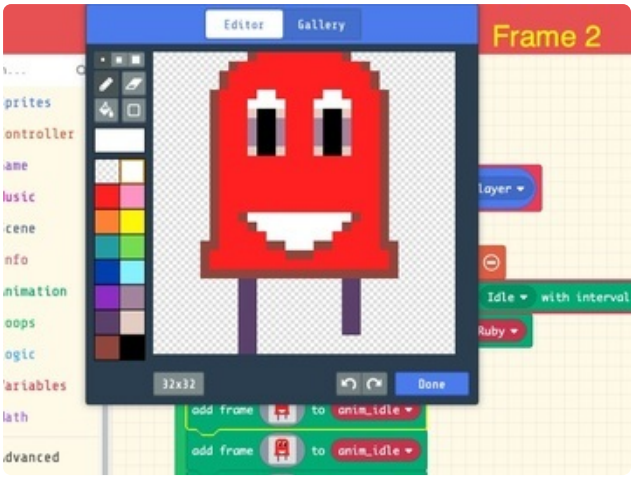
Now we'll get to the really fun part! Adding frames of animation!

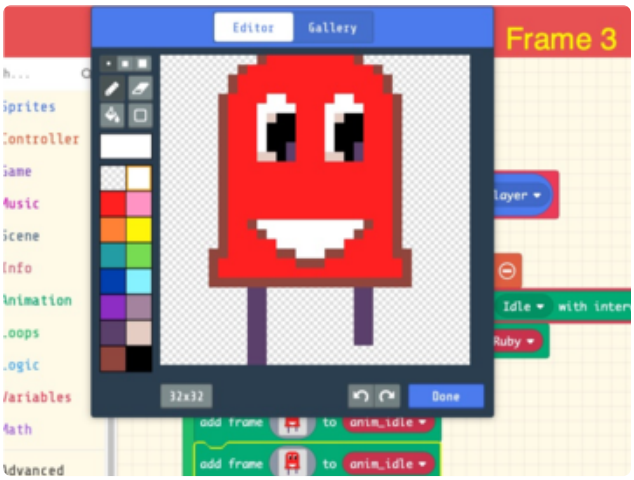
First, drag in an **add frame** block as shown. In it, you'll use the sprite editor to create your first frame of the idle animation.

I chose to re-create Ruby but with her eyes facing forward. (This is the only one you'll create from scratch, the remaining frames will be duplicates of this one with minor changes as the eyes look left and right.)

Next, chose the **anim_idle** from the dropdown menu. This way, we're indicating that this frame of animation belongs to that cycle.







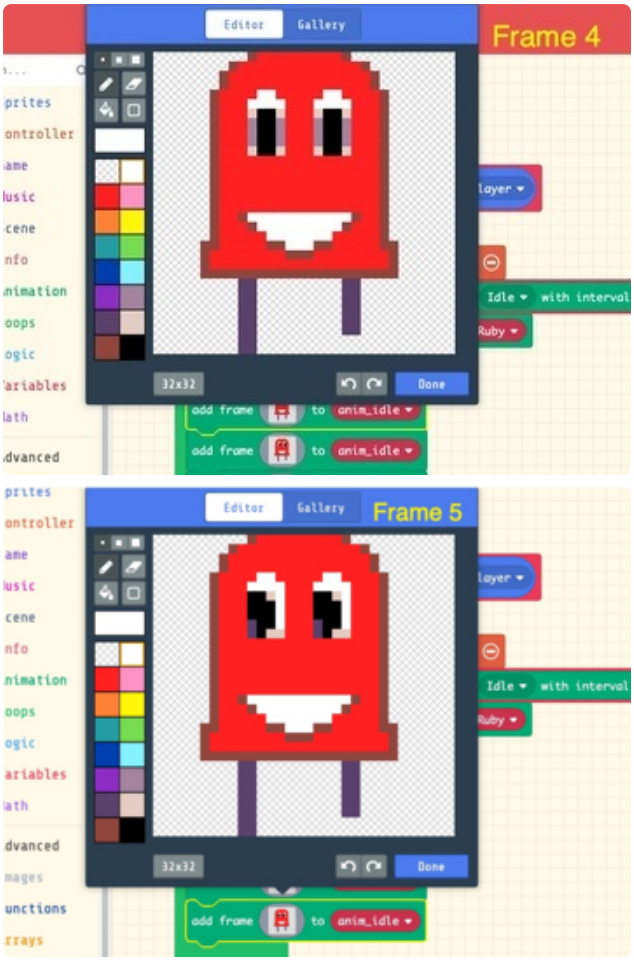
We'll make a total of five frames for the idle animation. Duplicate the first frame by right clicking on the **add frame** block and choosing **Duplicate.** Add this and three more duplicates to the block as shown, for a total of five frames.

Frame 2: Leave the second frame an exact copy of the first -- this will create a pause at the top of the cycle. This uneven timing will give the animation a more organic feel as opposed to perfectly even intervals between gaze shifts.

Frame 3: For the third frame, re-draw the eyes to look in the screen right direction.

Frame 4: Leave frame four as is looking straight ahead.

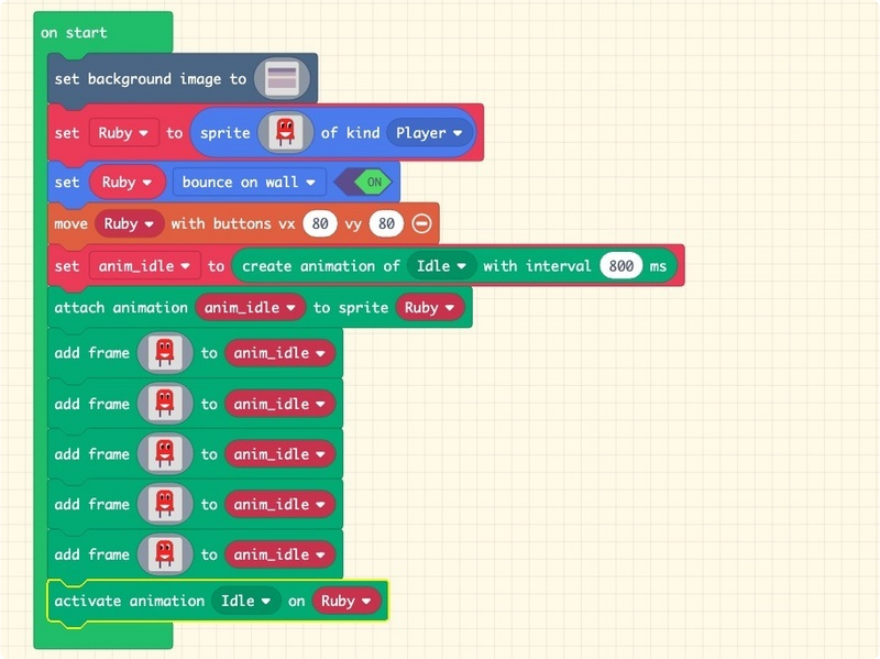
Frame 5: Finally, redraw frame five looking left.



Activate Animation

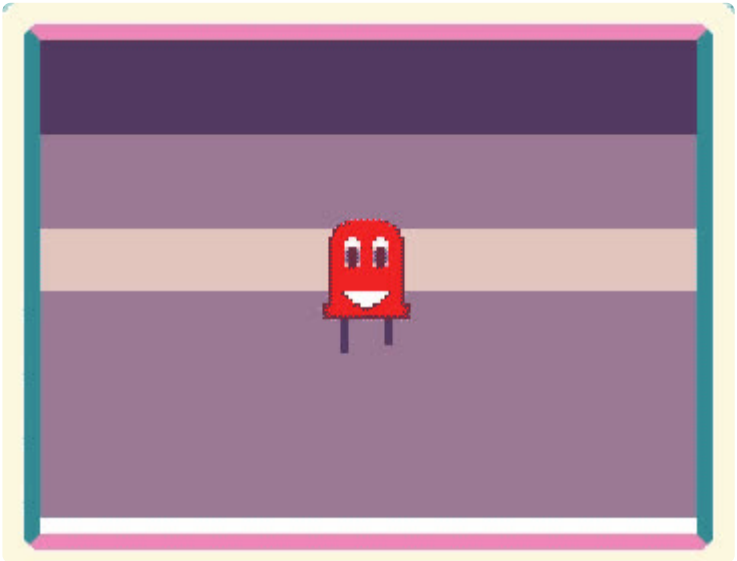
We've now created an animation set, attached it to a character, and added frames of animation. The final thing to do is tell the game to activate the animation under some condition. We can, for example have a jump animation only activate when a button is pressed, or a happy dance animation activate when a certain score is reached.

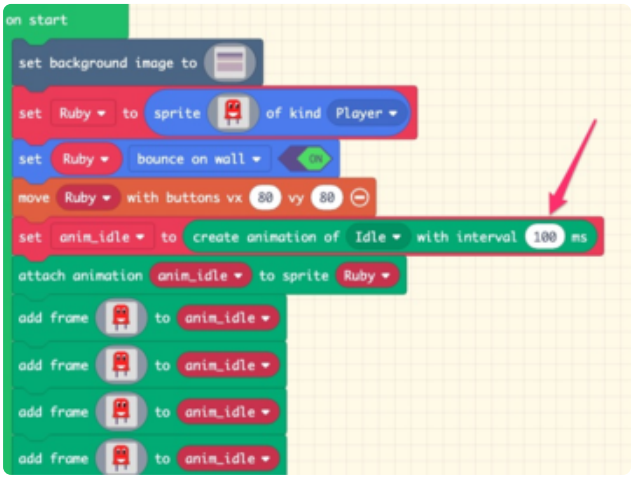
Right now, let's have the idle animation play when the game begins. Add an **activate animation** block as shown here, and then set the dropdown menus to **Idle** and **Ruby.** This specifies which cycle to run on which character sprite.



Here's what the idle animation looks like.

And here it is playing in the MakeCode Arcade simulator.

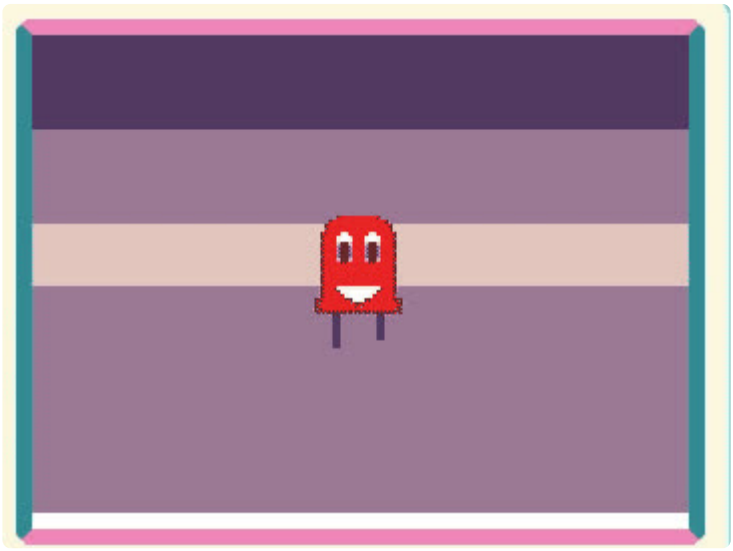




Interval Speed

One interesting trick you can use in games to make the most use out of your work is to adjust timing of animation cycles rather than create entirely new ones. For example, a if we need Ruby to look like she's reading something, we could speed up the Idle animation cycle by adjusting the interval speed temporarily!

Here it is with an interval of **100ms** instead of **800ms**.



Walk Cycle Animation

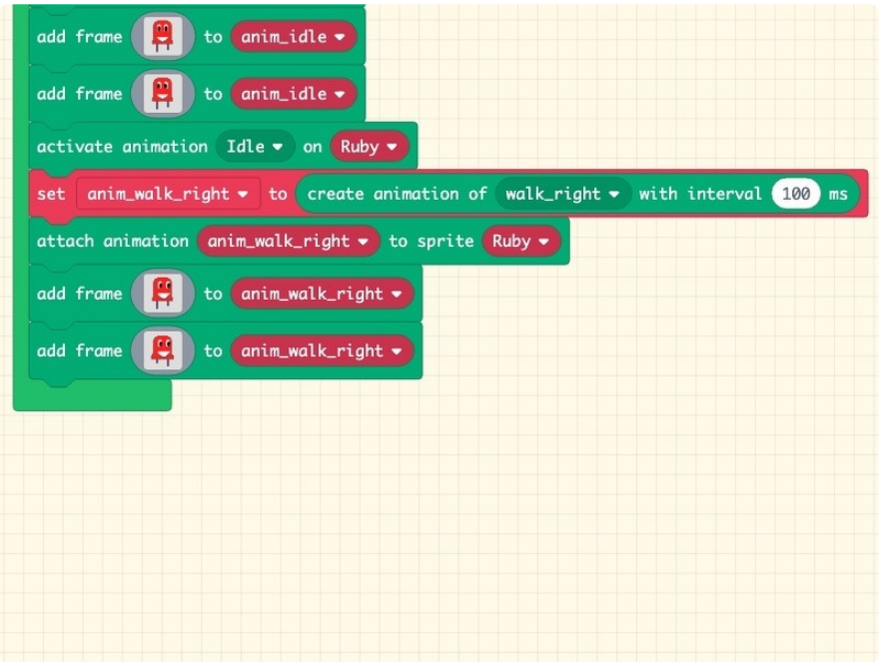
Now that we have our Idle animation built, we'll create a walk cycle. The idea here is that we'll make four different walk cycles, one for each direction Ruby can move on the d-pad, so walk right, walk left, walk up, and walk down.

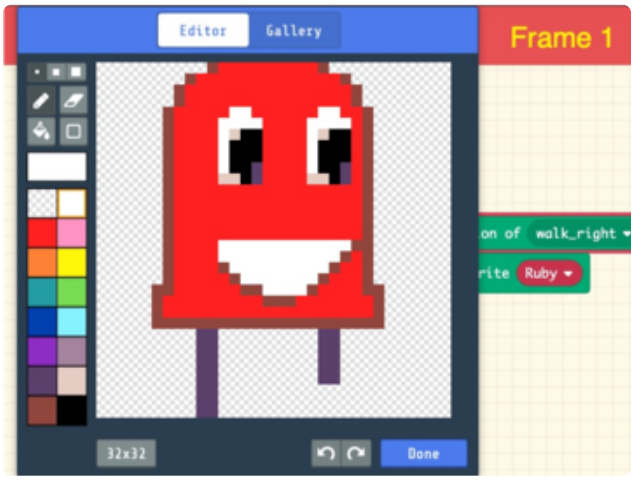
We'll create the first walk cycle to the screen right, then we'll be able to duplicate the block set for the other three walk cycles.

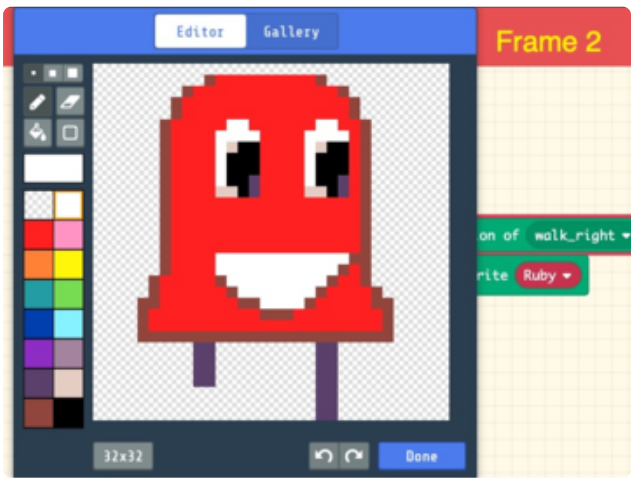
The setup is the same blocks you used for the idle animation, but this time we'll only use two frames of animation, and set the interval to a quick 100ms.

You can duplicate one of the **add frame** blocks from the idle animation so that you don't need to re-draw Ruby from scratch.

Name the animation variable and action as shown here to **anim_walk_right** and **walk_right**.







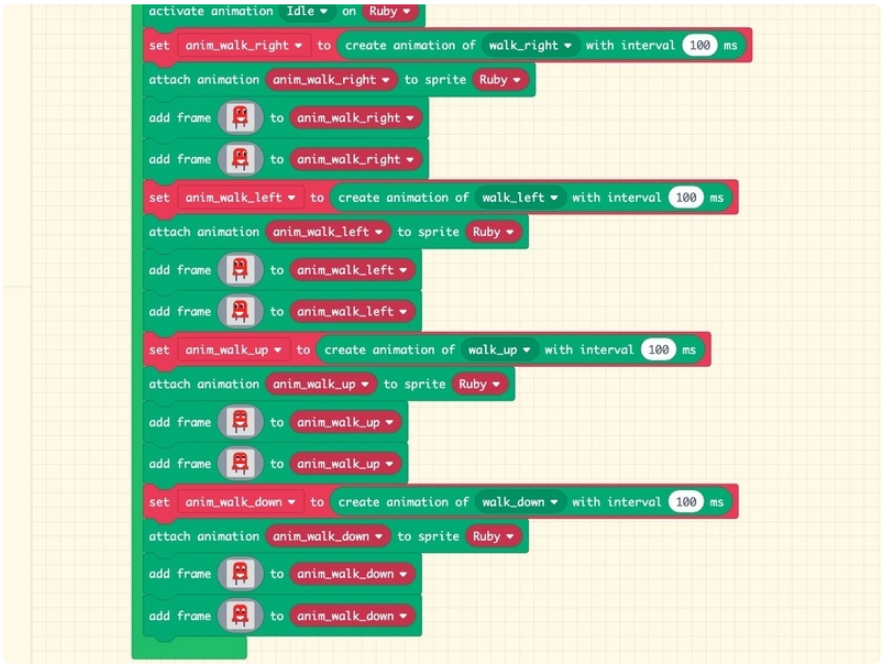
Walk Right

This will be a simple, quick walk with lots of excitement built in, as well as a shift in the character to slightly face the direction she's going.

Edit the first frame of the walk cycle so that the eyes are shifted to the right and the screen right leg is lifted up higher than the default.

Edit the second frame so that Ruby's body has shifted down a bit, the screen left leg is lifted, and you can even add some secondary motion or squash to the silhouette of the character by widening the base of the hips as shown here.

Now, duplicate the walk cycle three more times for the walk_left, walk_up, and walk_down animations.



Walk Left

This can be a simple mirroring of the walk_right frames.

Walk Up

Here I adjusted the eyes to look upward broadened the smile a bit for visual interest.

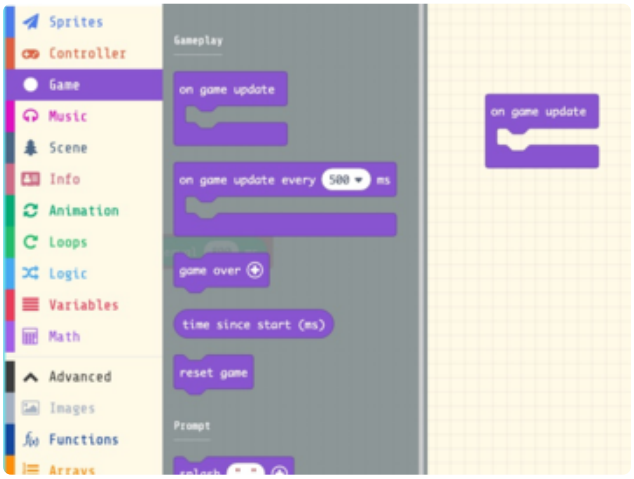
Walk Down

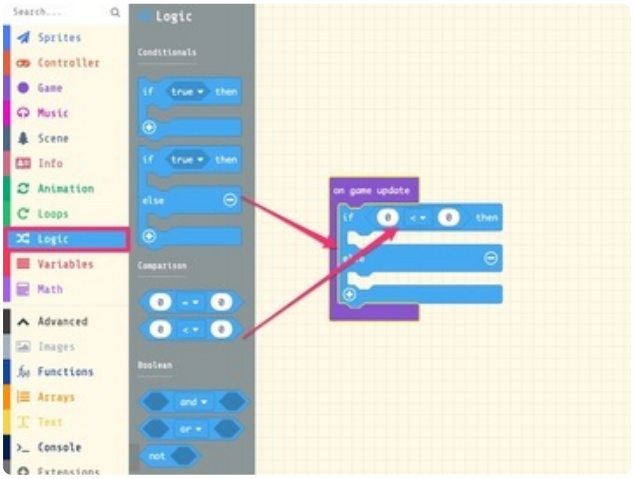
Finally, for the down walk adjust the eyes to look downward and you can adjust the mouth a bit again for a slight change that helps the cycle read differently.

Trigger the Animation Cycles

OK, now we have these animation cycles, how do we actually use them? Other than the Idle cycle, none of them will play back at the moment. We need to create conditions to trigger them depending on the user input to the controller.

We want to check the game state to test if the character is moving or not, and if so, in which direction. Depending on the direction of motion, measured with a velocity (direction and speed) check, we can activate the proper animation cycle.



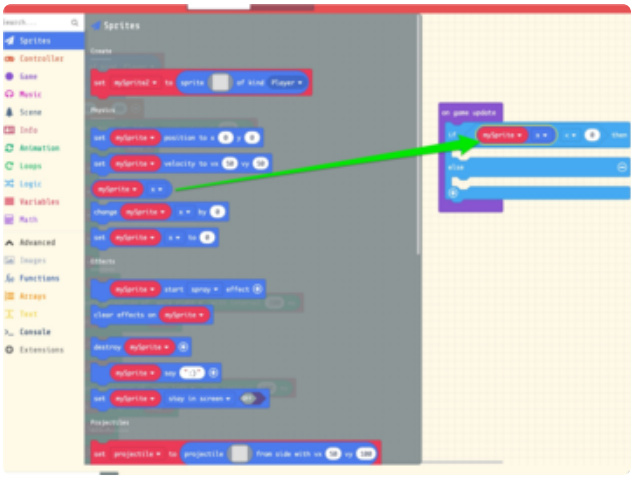


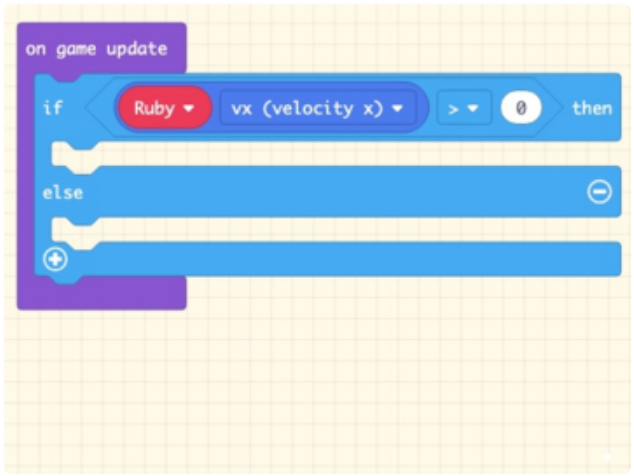
Update and Conditional Comparison

Add an **on game update** block from the **Game** category. We can then place blocks in here that will run on every clock tick of the game code to check what's happened since the last check. This is a great way to do things like monitor a character's velocity.

Into this update block, we'll drag an **if_else** block from the **Logic** category. We're going to use this to test the conditions of Ruby's velocity.

Now, add a Boolean comparison operator **0 < 0** to the first **if** statement. We'll adjust this in a moment to compare Ruby's velocity on each axis to 0, but the basic operation of this block is to return a `True` or `False` answer depending on the result of the comparison.



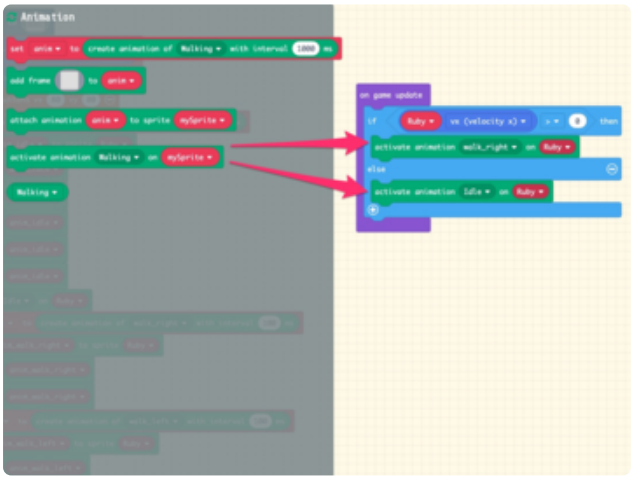


Velocity Check

In order to check Ruby's velocity, first drag a **mySprite x** block from the Sprites category into the comparison operator as shown.

Then, change the dropdown to pick **Ruby** and the attribute to **velocity x.**

To test first if Ruby is moving to the right, we need to see if her velocity on x is greater than 0. Set the comparison operator symbol to > greater than.



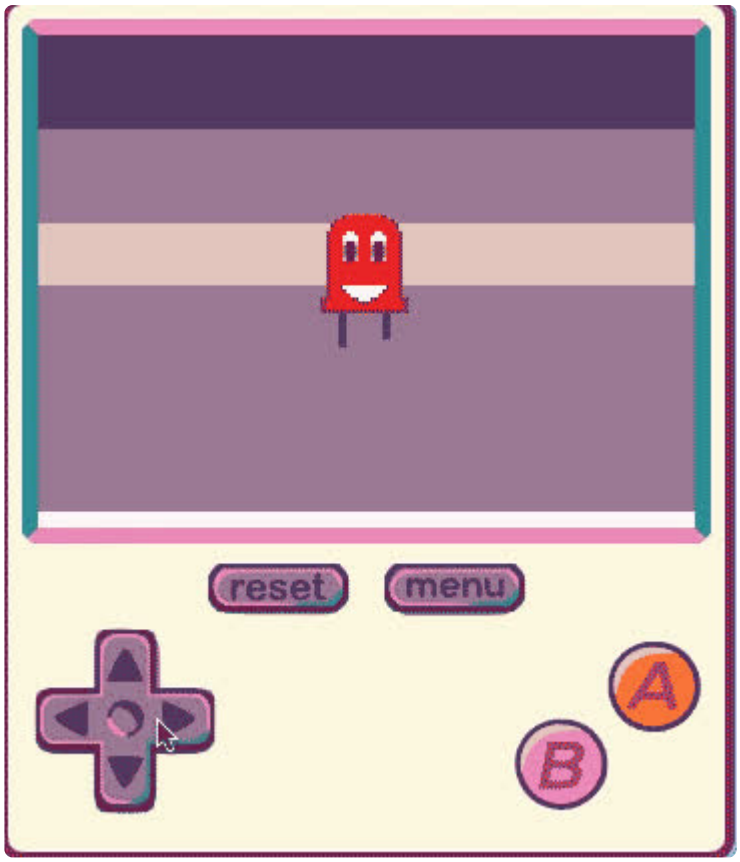
Activate!

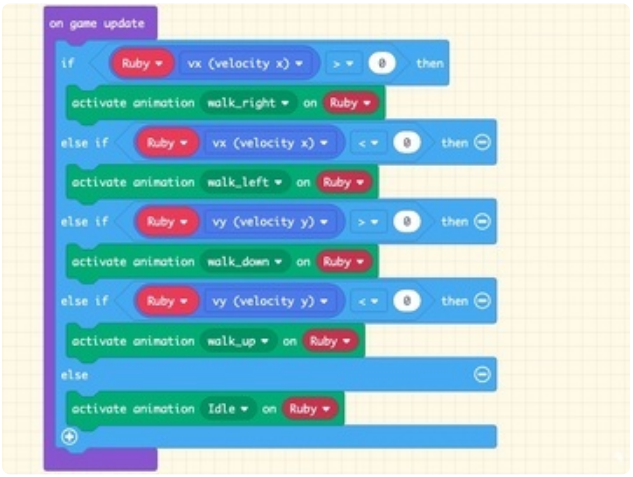
Now we can activate the walk right animation cycle when Ruby's velocity on x is greater than 0.

We'll also tell the conditional statement to activate the idle animation when Ruby isn't moving.

From the **Animation** category, drag in two of the activate animation blocks as shown, and set the dropdowns to **walk_right** on **Ruby** and **Idle** on **Ruby** respectively.

Test this out in the simulator now. You should see Ruby walking with the proper walk cycle when moving screen right, but sliding in other directions.





All Cycles

Now, we can add the remaining four cycles. First, click the + plus sign three times in the **if then** block to add three **if else** sections.

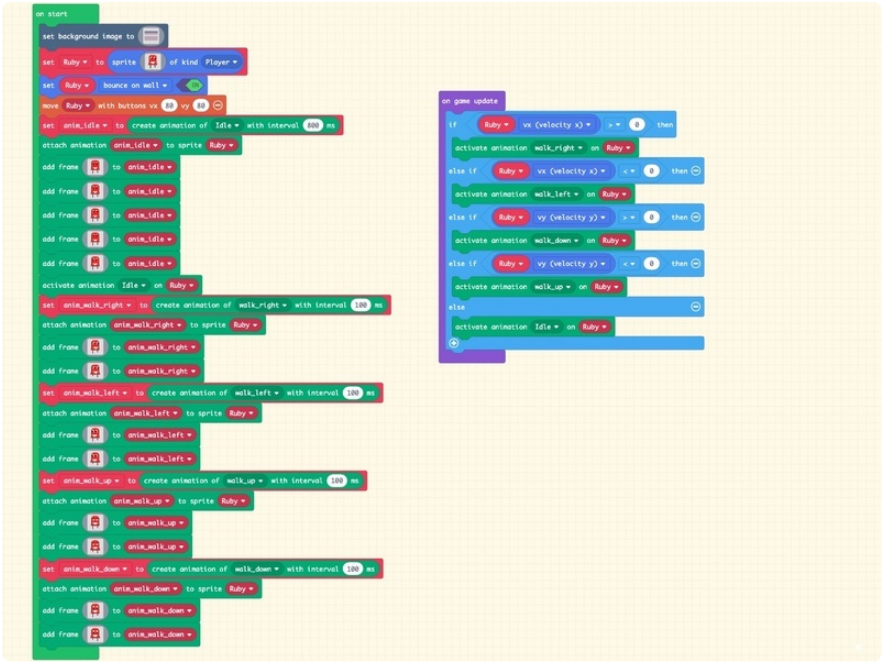
Into these, add the following comparisons and activations:

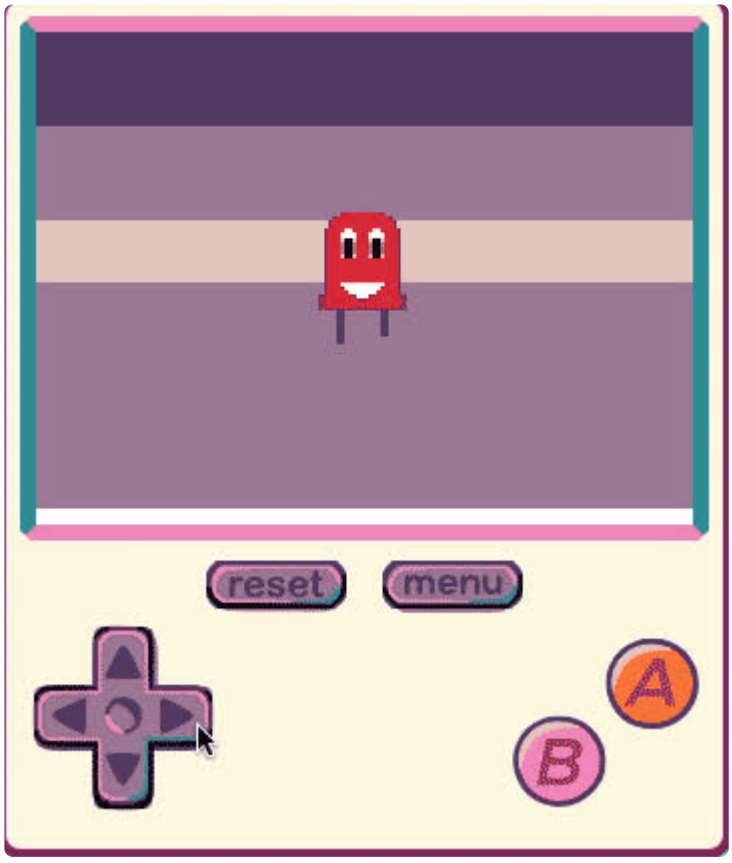
If else **Ruby vx < 0**, then **activate walk_left**

If else **Ruby vy > 0,** then **activate walk_up**

If else **Ruby vy < 0,** then activate **walk_down**

Here is the final project, click this link to check it out (https://adafru.it/Ele) in MakeCode.





Head to the next page to see how you can load your MakeCode Arcade game onto a Pybadge!

Update the PyBadge/PyGamer Bootloader

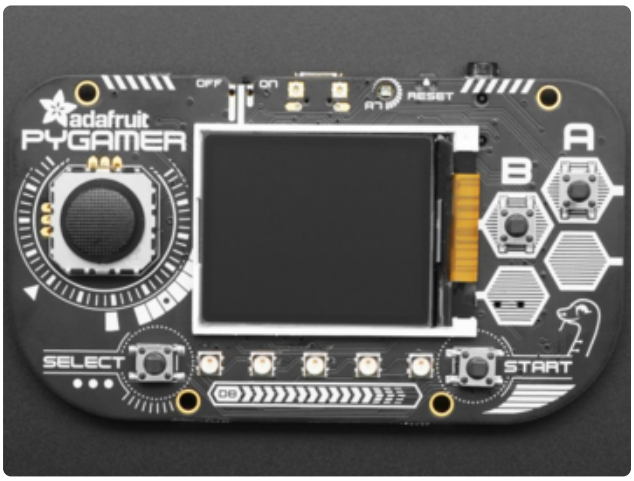
You are at the bleeding edge of handheld, open source, game playing hardware and software, what with your PyBadge/PyBadge LC or PyGamer! Congratulations! It's fun

and exciting! It is also changing and improving all the time, so please update your bootloaders before proceeding to put your MakeCode Arcade games on the board!!

Among lots of other reasons, update the bootloader to prevent a problem with MacOS 10.14.4, to fix button problems, and get the thumbstick to work!

PyBadge/PyBadge LC Bootloader

If you have a **PyBadge** or **PyBadge LC**, please go to this page for instructions on updating the bootloader. (https://adafru.it/EWI)



PyGamer Bootloader

If you have a **PyGamer**, please go to this page for instructions on updating the bootloader. (https://adafru.it/EWJ)

A HUUUUUUGE number of people have problems because they pick a 'charge only' USB cable rather than a "Data/Sync" cable. Make 100% sure you have a good quality syncing cable. Srsly, I can't even express how many times people have nearly given up due to a flakey USB cable! Enter Alert Text...

Hardware Checks

If, after updating your board's bootloader, you still think you may have a hardware problem, here's a great way to test out all of the functions. From buttons, to the light sensor, thumbstick (PyGamer only), accelerometer (PyGamer and PyBadge only, not the LC), and more, we've got a super nifty set of hardware test .UF2 files you can use.

Click on the link for your board below for more info and a link to the appropriate UF2 file.

PyBadge/PyBadge LC Hardware Check

https://adafru.it/EWK

PyGamer Hardware Check

https://adafru.it/EWL

Another way to do a hardware check is with the handy, dandy MakeCode Arcade Basic Hardware Test. This was created with MakeCode Arcade and you can use it to check that your d-pad buttons or thumb joystick can move the yellow face around the screen, and that the A and B buttons work to play a sound (just make sure you have a speaker plugged in to the PyGamer first).

You can open this link (https://adafru.it/EWP) to get to it, or download the UF2 file below and drag it onto your board's USB drive in bootloader mode.

arcade-Basic-Hardware-Test.UF2

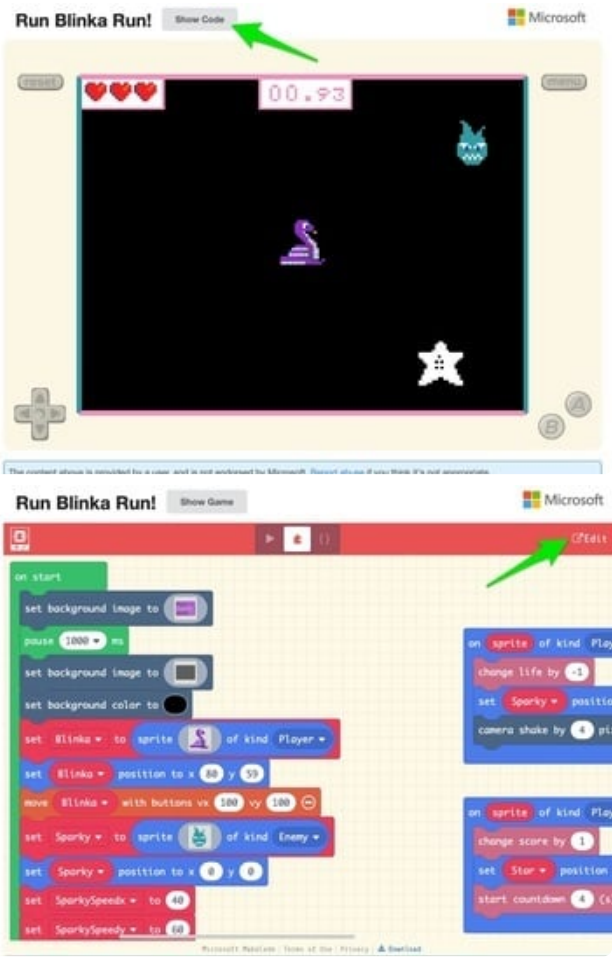
https://adafru.it/EWQ

Load a MakeCode Game on PyGamer/ PyBadge

Let's load a game! For example, here's a link to **Run, Blinka, Run!** To open the game in the MakeCode Arcade editor, first, click the share link below. This will allow you to play the game in the browser right away.

Makecode Arcade Game: Run, Blinka, Run!

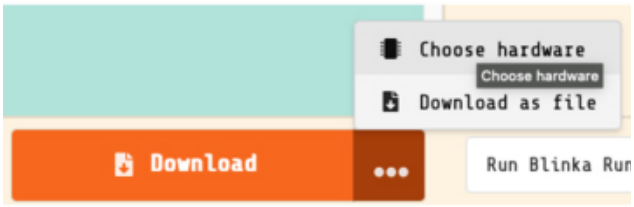
https://adafru.it/Fqf



Then, click on the Show Code button in the upper left corner. The shows the code for the game, and by clicking the Edit button in the upper right corner, it'll open into the editor where you can upload it to your PyGamer/PyBadge.

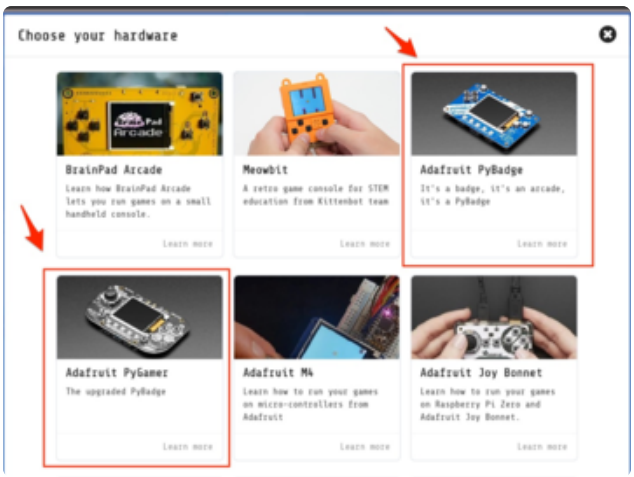
Once you have a game working on the MakeCode Arcade web editor, it's time to download it and flash it onto your board.

Please only use the Google Chrome browser with MakeCode! It has WebUSB support and seems to work best



Board Definition

In order to load a game made in MakeCode Arcade onto the PyBadge, first choose the proper board definition inside of MakeCode. Click the ellipsis (...) next to DOWNLOAD and then the **Choose Hardware** item.

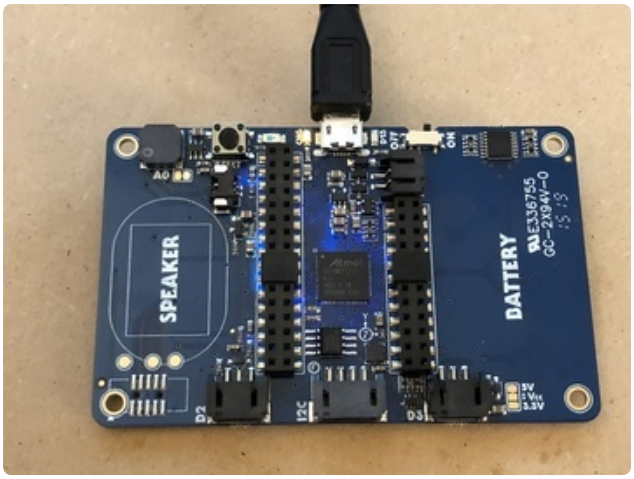


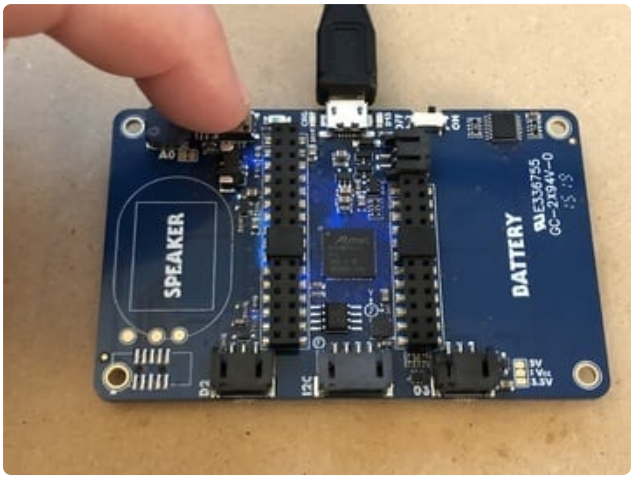
Change Board screen

Click on the image of your board, either the PyBadge/PyBadge LC or the PyGamer

This will cause the game .uf2 file for your particular board to be saved to your hard drive. You only need to do this the first time you use a new board. Thereafter you can simply click the **Download** button on the MakeCode Arcade editor page.

A HUUUUUUGE number of people have problems because they pick a 'charge only' USB cable rather than a "Data/Sync" cable. Make 100% sure you have a good quality syncing cable. Srsly, I can't even express how many times people have nearly given up due to a flakey USB cable!



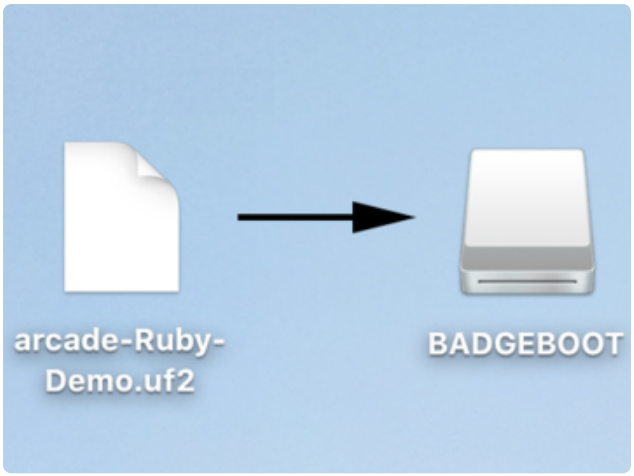


Bootloader Mode

Now, we'll put the board into bootloader mode so we can drag on the saved .uf2 file. On the back side of the board you'll see a reset button at the top. Make sure the board is plugged into your computer via USB with a USB micro B to A data cable. Also, be sure the board is turned on.

Then, press the reset button. This will initiate bootloader mode.

When the board is in bootloader mode you'll see a screen similar to this one show up.

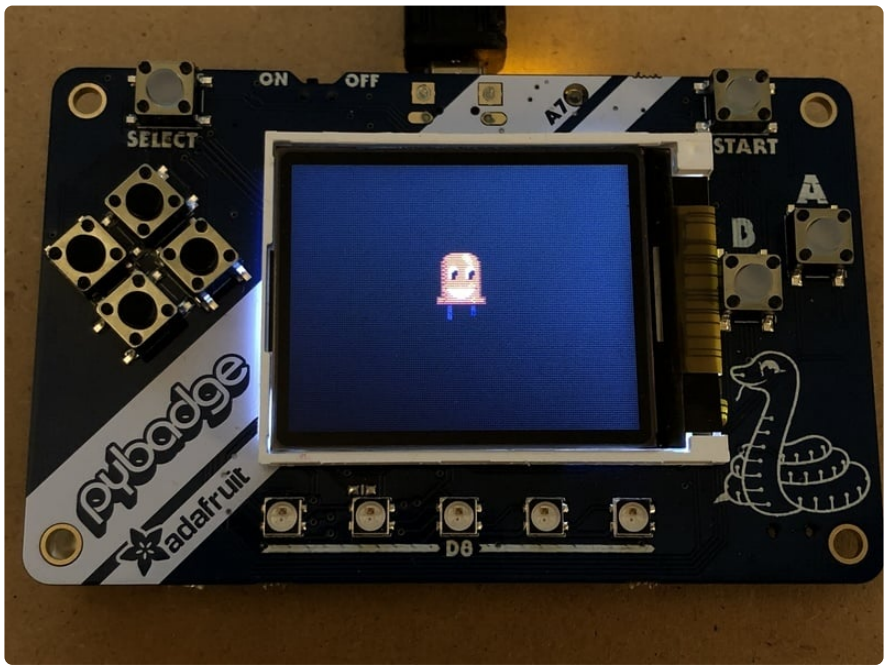


Drag and Drop

Now that the board is in bootloader mode, you should see a **BADGEBOOT** drive show up on your computer as a USB flash drive. Simply drag the arcade game .uf2 file onto the drive.

Play!

That's all there is to it! Once the file is copied over the board will restart and launch the game!



Keep an eye on Adafruit.com for additional game related content.

Troubleshooting MakeCode Arcade

If you run into trouble with MakeCode Arcade, here are some resources for getting help:

- Microsoft MakeCode Arcade Forum (https://adafru.it/EXI)
- Adafruit MakeCode Forum (https://adafru.it/EXJ)

- Microsoft MakeCode Arcade Discord (https://adafru.it/EXK) -- look for the #arcade channel
- Adafruit MakeCode Discord () -- look for the #makecode channel

Only use the Google Chrome browser with MakeCode!