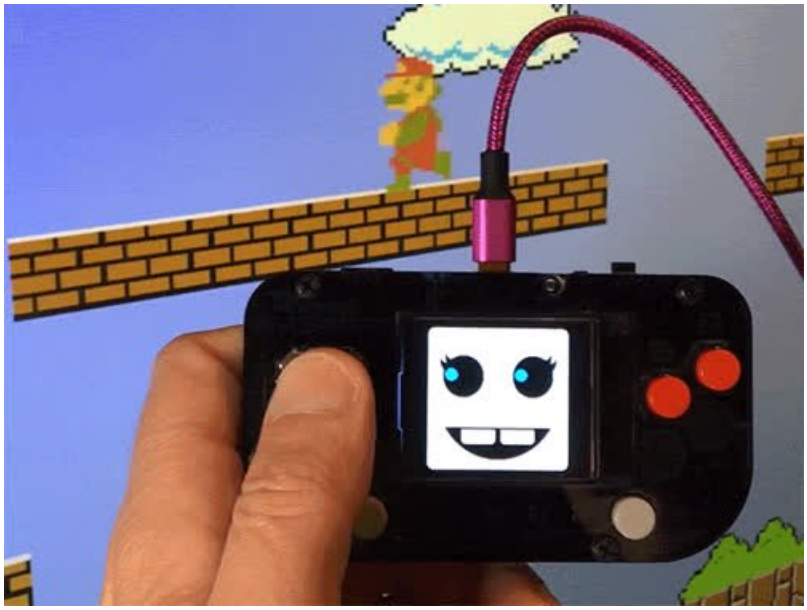
adafruit learning system

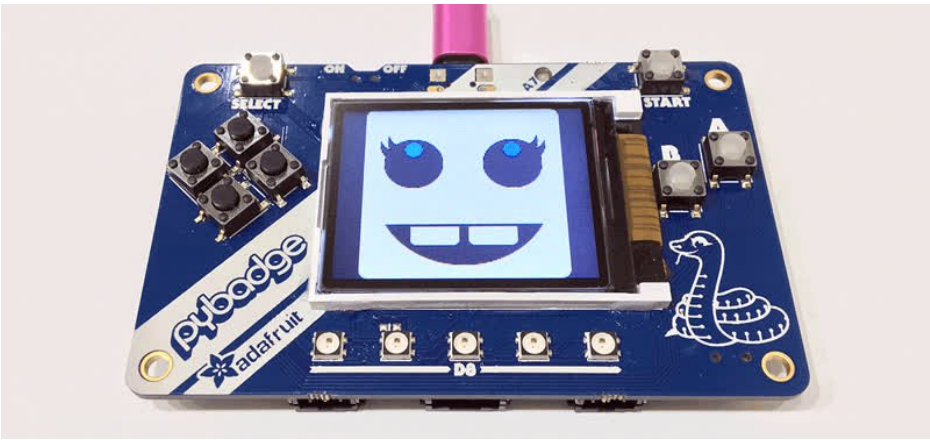
JOY of Arcada — USB Game Pad for Adafruit PyGamer and PyBadge

Created by Phillip Burgess



Last updated on 2019-06-17 03:20:13 PM UTC

Overview



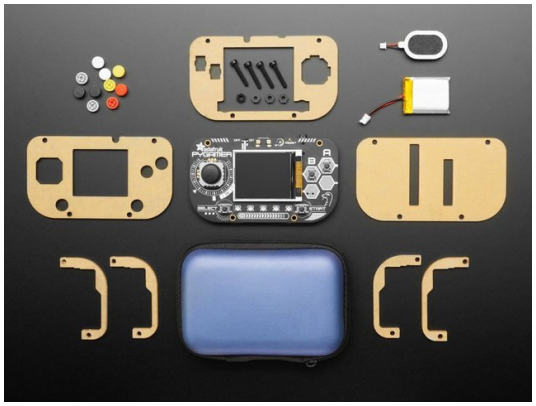
Surely you've played some little **games** right on your **PyGamer** or **PyBadge** — perhaps using **CircuitPython** or Microsoft **MakeCode**. But did you know...you can also use PyGamer or PyBadge as a **USB game controller** with your regular computer or with popular emulators on Raspberry Pi! Not just *any* game controller though. With that little screen, why not give it some *personality?* So we made a little animated friend and named her **JOY**. Joy's eyes blink and follow the joystick, and she makes an occasional encouraging "pew pew!" as buttons are pressed.

This is a new take on an earlier project — JOY Controller for Adafruit Feather (https://adafru.it/F2P) — which required soldering and 3D printing. Now that we have these all-in-one devices like PyGamer, it's much easier to get something going!

JOY will watch and cheer you on as you play, but doesn't *have to be* a game controller. She can control all manner of software or media such as YouTube, Photoshop, Premiere, Ableton Live, etc. Anything that a USB keyboard can do, JOY can operate as well.

Hardware

You can use this project on a number of Adafruit products including the PyBadge and PyGamer lines of products.



Adafruit PyGamer Starter Kit

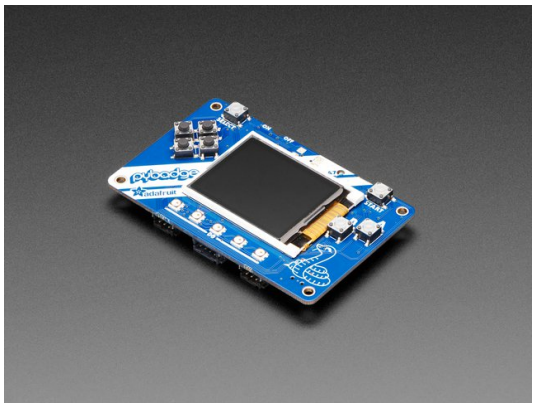
OUT OF STOCK

OUT OF STOCK

Adafruit PyGamer for MakeCode Arcade, CircuitPython or Arduino

OUT OF STOCK

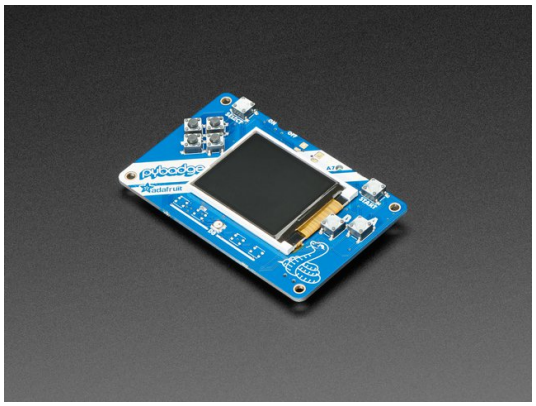
OUT OF STOCK



Adafruit PyBadge for MakeCode Arcade, CircuitPython or Arduino

OUT OF STOCK

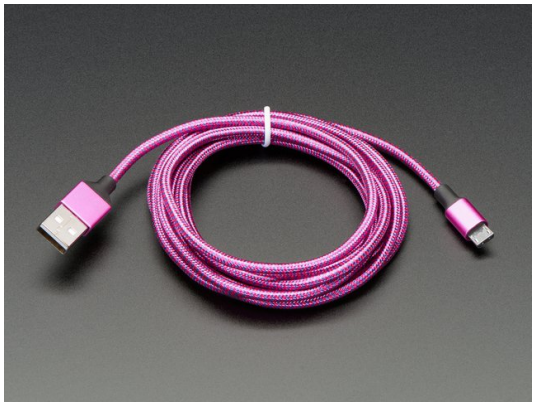
OUT OF STOCK



Adafruit PyBadge LC - MakeCode Arcade, CircuitPython or Arduino

OUT OF STOCK

OUT OF STOCK



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

$3.95
IN STOCK

ADD TO CART

Software

Ready-Made Software

Plug **PyGamer** or **PyBadge** into your computer with a **USB** cable. Make sure the **power switch** is set to the "**on**" position, then **double-click the RESET button** on the top or back of the board.

After a moment, a small flash drive called **PYGAMERBOOT** or **PYBADGEBOOT** should appear on your system. Drag-and-drop one of the **.UF2 files** (downloadable below) on to this flash drive. There will be a few seconds of LED flashing, then the drive will be ejected.

Here's the .UF2 file specifically for **PyGamer** boards:

https://adafru.it/F2Q

https://adafru.it/F2Q

And a version for **PyBadge** (regular or LC):

https://adafru.it/F2R

https://adafru.it/F2R

You can also build JOY from source code (it's an Arduino project) but it's quite involved. This is explained on the "How it Works" page.

Customizing JOY for Different Key Setups

The default button-to-key assignments on JOY won't be ideal for everyone's needs, but are easily **customized** without having to edit and recompile the code.

When connected to USB, a PyGamer or PyBadge appears on your computer as a small flash drive called **CIRCUITPY** (if it does not, you'll need to go through the one-time CircuitPython installation (https://adafru.it/Em8) for the board, then reload one of the JOY .UF2 files above).

In the root level of this drive (not inside any folder), create a text file called `joy.cfg` using any plain-text editor you like. Here's an example you can copy-and-paste, then edit to your liking:

```
{
  "a":      "Z",
  "b":      "X",
  "start":  "1",
  "select": "5",
  "up":     "UP_ARROW",
  "down":   "DOWN_ARROW",
  "left":   "LEFT_ARROW",
  "right":  "RIGHT_ARROW"
}
```

The file uses "JSON" syntax...which can be fairly picky, but it's an established standard and we can rely on well-tested code to read it.

Each line consists of a *keyword* (corresponding to one of the buttons on Joy) and a *value* (corresponding to keys on a keyboard). Both in quotes, with a colon (:) between them and a comma at the end of the line (except for the last item). The entire set is then contained inside a set of { curly braces }. Yes, JSON is that specific.

Each **keyword** is one of eight *specific* strings, and *must* be **lower-case**: "a", "b", "start", "select", "up", "down", "left" and "right". *No exceptions.*

Each **value** is one key name from the following table (these can be upper or lower case):

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 1 2 3 4 5 6 7 8 9 0	RETURN ESCAPE BACKSPACE TAB SPACE MINUS EQUAL LEFT_BRACKET RIGHT_BRACKET BACKSLASH EUROPE_1 SEMICOLON APOSTROPHE GRAVE COMMA PERIOD SLASH CAPS_LOCK F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 PRINT_SCREEN SCROLL_LOCK PAUSE INSERT HOME PAGE_UP DELETE END	PAGE_DOWN RIGHT_ARROW LEFT_ARROW DOWN_ARROW UP_ARROW NUM_LOCK KEYPAD_DIVIDE KEYPAD_MULTIPLY KEYPAD_SUBTRACT KEYPAD_ADD KEYPAD_ENTER KEYPAD_1 KEYPAD_2 KEYPAD_3 KEYPAD_4 KEYPAD_5 KEYPAD_6 KEYPAD_7 KEYPAD_8 KEYPAD_9 KEYPAD_0 KEYPAD_DECIMAL EUROPE_2 APPLICATION POWER KEYPAD_EQUAL F13 F14 F15 LEFT_CONTROL LEFT_SHIFT LEFT_ALT LEFT_GUI RIGHT_CONTROL RIGHT_SHIFT RIGHT_ALT RIGHT_GUI

JOY reads this file on startup. You'll get an **alert message** if the file is **missing** or the **syntax** is broken (make sure all the quotes and commas are in the right places).

If some buttons work but others do not, it's most likely a key name that's misspelled or not in the above list (it won't report an error — the JSON syntax is valid, just the word is wrong).

This file is read on startup *only.* Changes are *not* detected live. After editing, write your changes to the file, then tap

the **reset** button on PyGamer/PyBadge to reload.

How it Works

Joy is written as an **Arduino** sketch. For something that basically does one task...a USB game pad, albeit one embellished with a lot of graphical flair...it's an awfully *big and hairy* Arduino sketch.

It was written this way for performance reasons, to keep everything animated and responsive. But to be honest, between ever-faster microcontrollers and ongoing improvements in CircuitPython speed and features, there will probably be no *need* to program projects like this in such a tedious manner in the future! But if you're curious, here's a link to the code:

Joy_of_Arcada source code on Github (https://adafru.it/F2V)

There are four files: **Joy_of_Arcada.ino** is the main Arduino sketch, which is accompanied by three header files (.h) containing tables of graphics, sound and keyboard codes.

Joy_of_Arcada is so named because it uses our *Adafruit_Arcada* library, which encapsulates a lot of graphics, sound and control-related functions common to several Adafruit boards. The Arcada library, in turn, depends on a *whole bunch* of other libraries to provide the lower-level functionality. So many libraries, in fact, that rather than list them all here it's best to link to this other guide explaining all the prerequisites (https://adafru.it/EU5).

You should also have the latest Adafruit SAMD boards package for Arduino (version 1.5 or later, we suggest the latest version in the Arduino library manager). If you've used other Adafruit SAMD boards in the past (M0, M4, HalloWing, etc.), it's worth checking for any recent updates (Tools→Board→Boards Manager...). In addition, before compiling this code, make sure to select Tools→USB Stack→TinyUSB (this lets our code access files on the PyGamer/PyBadge flash filesystem).

Animating Joy

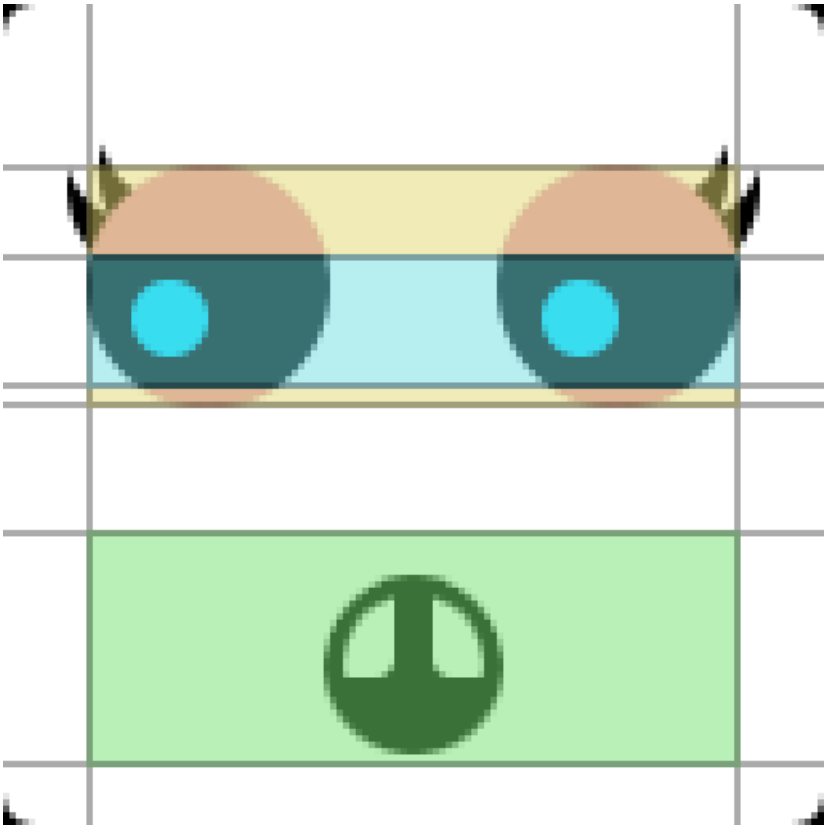
To ensure button and joystick input is processed expediently, the code pulls shenanigans to draw the face very quickly.

First, the entire screen is not drawn for every frame of animation. There's really only a rectangular section in the middle where all the motion occurs — the bounds of the eyes and mouth. So after clearing the screen and drawing a full-face bitmap just once, all subsequent updates refresh only this middle area.

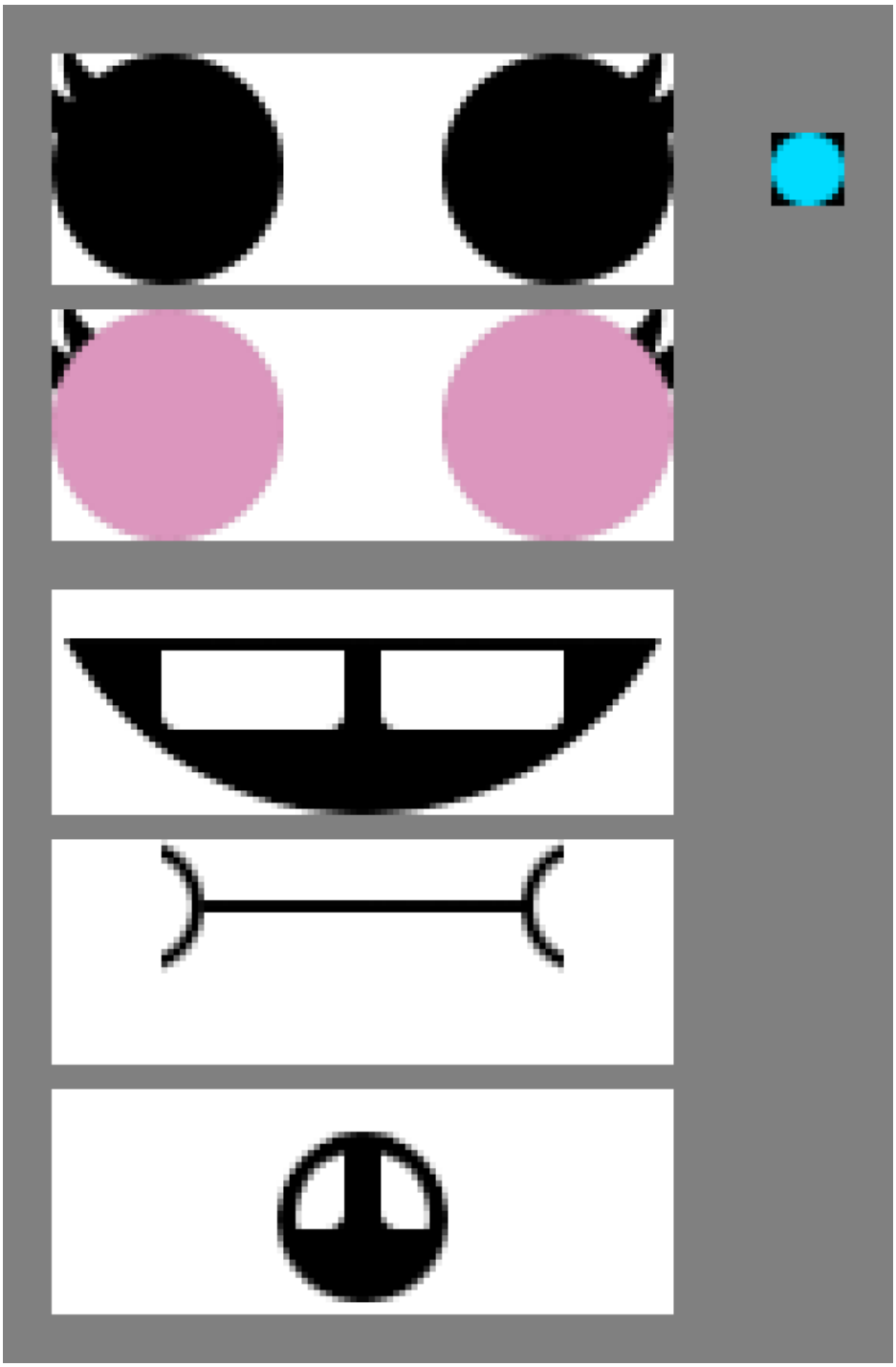
An offscreen buffer for just this area is maintained in RAM (called a *framebuffer* in Arcada library parlance, or a *GFXcanvas16* object in Adafruit_GFX terms). We periodically modify sections of the buffer in RAM and copy it to the screen.

The offscreen buffer is processed in regions, of varying height but all the same width. Even though that wastes a little memory (the mouth is not as wide as the eyes, for example), making everything the same width allows us to use a single `memcpy()` call to draw each region, because the scanlines are contiguous in memory (moving data between different-width images would require copying each scanline separately). It's a one-dimensional operation rather than 2-D.

The pupils are a special case. Those *are* drawn more conventionally (using the `drawRGBBitmap()` function from the Adafruit_GFX library, because they're round and we need that code's masking capability). Then the eyelids are drawn on top of this when needed, using `memcpy()` as previously described.



So drawing the face then is mostly a matter of copying one of several fixed-sized bitmap images (encoded in the graphics.h file — more on that in a moment) to a corresponding area in the offscreen buffer.



Copying each completed frame of face animation to the screen is done using *direct memory access* (DMA), which lets the data transfer occur "in the background," not using any instruction cycles...allowing us to move on with handling more joystick and button input while the screen redraws.

Preparing the Graphics

As alluded to above, the graphics are encoded as tables in a header file (part of program memory), not as image files in the flash filesystem. They're just **huge arrays of 16-bit values**, one value per pixel.

It's a frequent misconception that we have some kind of finely-crafted tool for converting images into header files

like this, but that's not true. Typically I'll use some throwaway **Python** code and the **Python Imaging Library** (PIL or its offspring Pillow) for such conversions. This often starts with an existing image conversion script (such as the one from the Uncanny Eyes project — tablegen.py in this repository (https://adafru.it/F2X)), tweaking it for the task at hand...but it's *extremely rare* that anything like this is held onto. **Every project's needs are different**, and it's better for your mental health to think of these little one-off scripts as disposable tissues, not precious gems to be hoarded. It's very informal stuff and that's actually a good thing. Python makes it so quick.

If you really need something ready-made though, this online tool (https://adafru.it/E-q) can handle quite a number of situations.