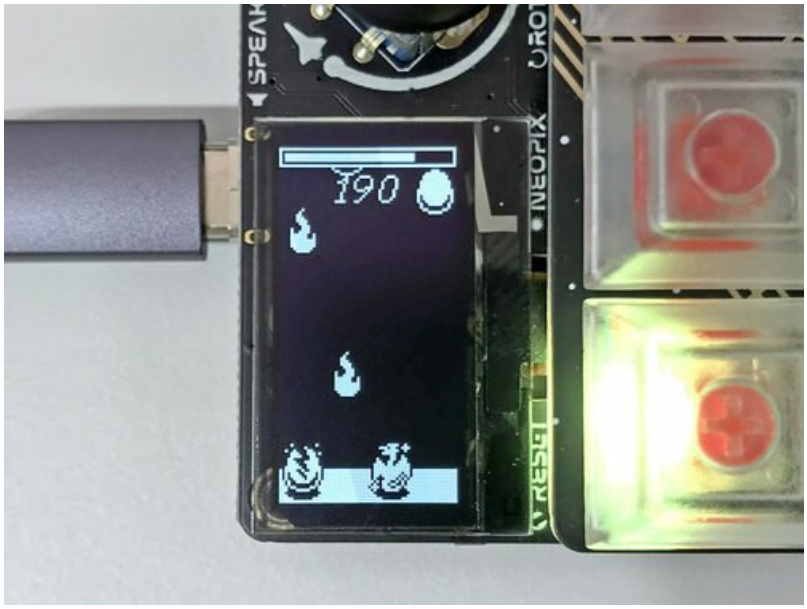
adafruit learning system

Dragon Drop: a CircuitPython Game for MacroPad

Created by Phillip Burgess



Last updated on 2021-08-11 10:55:37 PM EDT

Guide Contents

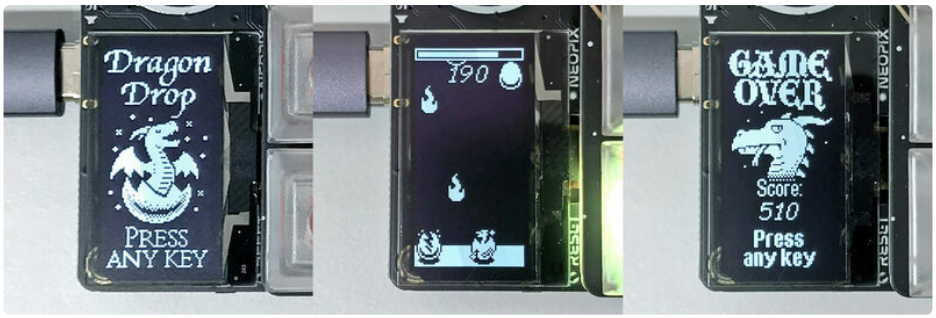
Overview

Adafruit's *MacroPad* **(https://adafru.it/TJD)** was designed as the ultimate keyboard accessory — whether for application hotkeys (https://adafru.it/TJE) or making music (https://adafru.it/TWc) or other computer-like duties.

One thing that really sets MacroPad apart from similar devices is that it's *not* simply a small keyboard...it's a whole *development board* running CircuitPython (https://adafru.it/cpy-welcome). Very handy for computer-connected tasks, certainly, but nothing actually *demands* that. What else could one do with this? How about a trinary handheld calculator? Attach a USB battery and make it portable. Make a *Quantum Leap* Handlink cosplay accessory. Or program some little games.

Let's try the latter and make a game! Just a simplistic one, but hopefully it plants some good ideas...

- MacroPad isn't *just* a keyboard, it's *fully programmable in CircuitPython.*
- It can do more than keyboard macros.
- It can be made portable with a USB battery.
- Consider using Macropad sideways or upside-down if your project calls for it. *There is no "up" in space!*



Parts

Your browser does not support the video tag.

Adafruit MacroPad RP2040 Starter Kit - 3x4 Keys + Encoder + OLED

Strap yourself in, we're launching in T-minus 10 seconds...Destination? A new Class M planet called MACROPAD! M here stands for Microcontroller because this 3x4 keyboard...

Out of Stock

Out of Stock

- or -

Adafruit MACROPAD RP2040 Bare Bones - 3x4 Keys + Encoder + OLED

Strap yourself in, we're launching in T-minus 10 seconds...Destination? A new Class M planet called MACROPAD! M here, stands for Microcontroller because this 3x4 keyboard...

Out of Stock

Out of Stock

Adafruit MacroPad RP2040 Enclosure + Hardware Add-on Pack

Dress up your Adafruit Macropad with PaintYourDragon's fabulous decorative silkscreen enclosure and hardware kit. You get the two custom PCBs that are cut to act as a protective...

$4.95

In Stock

Add to Cart

Kailh Mechanical Key Switches - Linear Red - 12 Pack

For crafting your very own custom keyboard, these Kailh Red Linear mechanical key switches are deeee-luxe! With smooth actuation and Cherry MX compatibility,...

$7.95

In Stock

Add to Cart

Clear Keycaps for MX Compatible Switches - 12-pack

Here is a 12 pack of Clear DSA keycaps for your next mechanical keyboard or

$6.95

In Stock

Add to Cart

CircuitPython

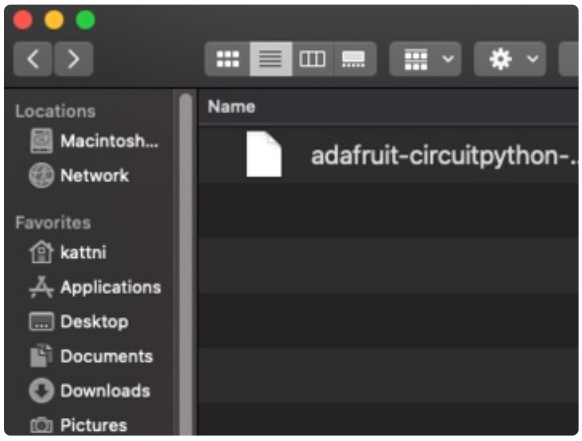
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

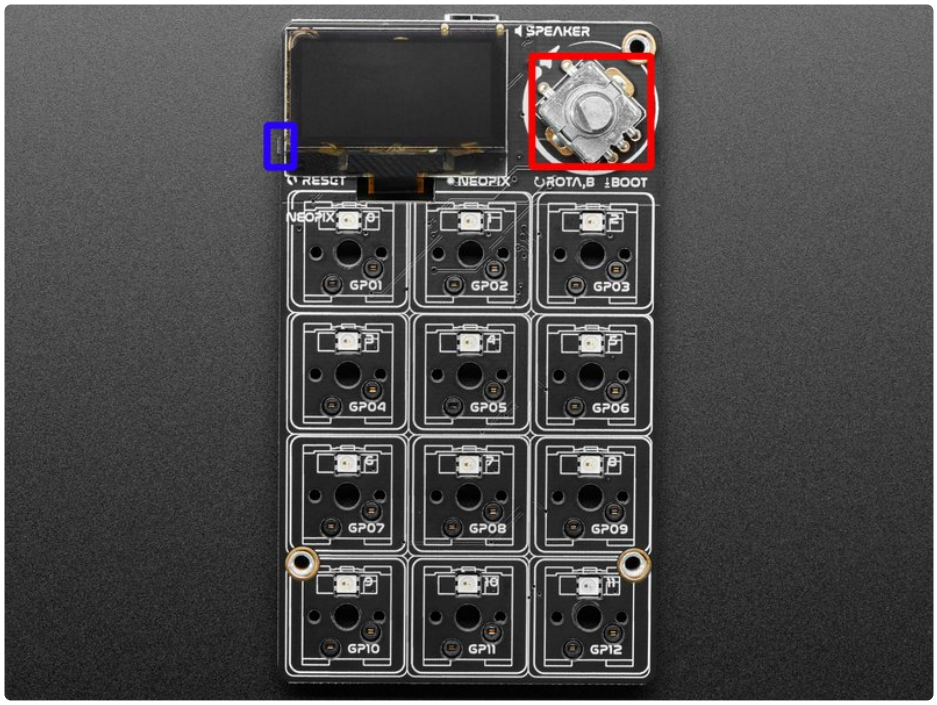
https://adafru.it/TB9

https://adafru.it/TB9



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.



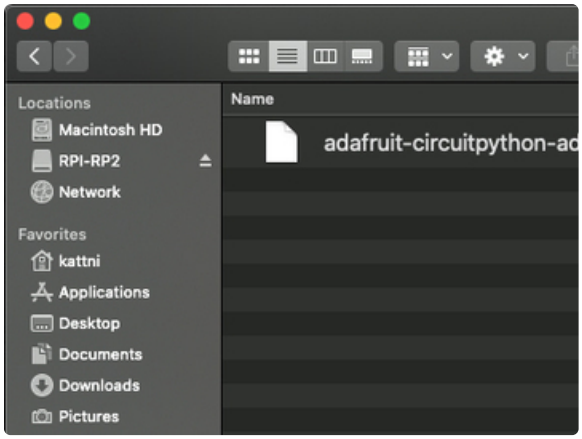
The BOOT button is the button switch in the rotary encoder! To engage the BOOT button, simply press down on the rotary encoder.

To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

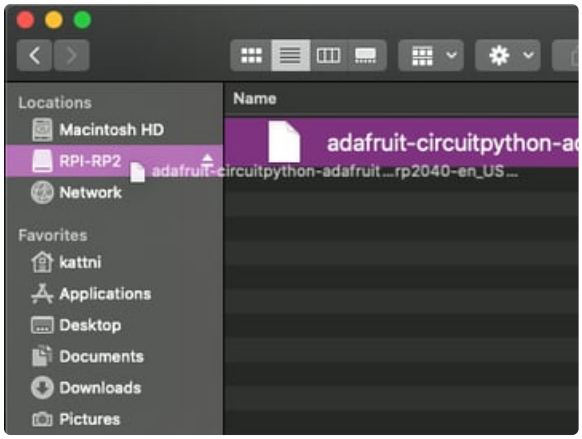
If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.

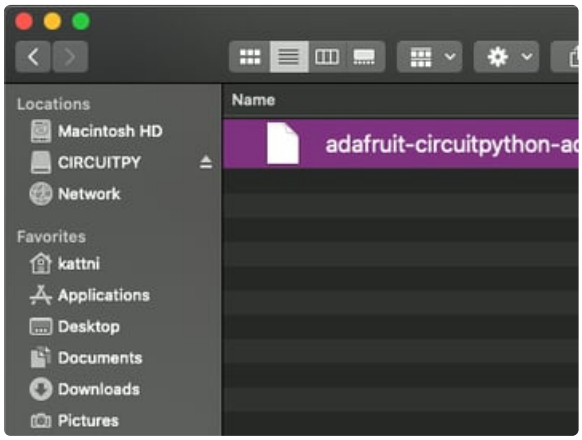
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **RPI-RP2**.



Drag the **adafruit_circuitpython_etc.uf2** file to **RPI-RP2.**



The **RPI-RP2** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it, you're done! :)

Safe Mode

You want to edit your **code.py** or modify the files on your **CIRCUITPY** drive, but find that you can't. Perhaps your board has gotten into a state where **CIRCUITPY** is read-only. You may have turned off the **CIRCUITPY** drive altogether. Whatever the reason, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode *bypasses any code in* **boot.py** (where you can set **CIRCUITPY** read-only or turn it off completely). Second, *it does not run the code in* **code.py**. And finally, *it does not automatically soft-reload when data is written to the* **CIRCUITPY** *drive*.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the **CIRCUITPY** drive.

Entering Safe Mode in CircuitPython 6.x

This section explains entering safe mode on CircuitPython 6.x.

To enter safe mode when using CircuitPython 6.x, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 700ms. On some boards, the onboard status LED (highlighted in green above) will turn solid yellow during this time. If you press reset during that 700ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

Entering Safe Mode in CircuitPython 7.x

This section explains entering safe mode on CircuitPython 7.x.

To enter safe mode when using CircuitPython 7.x, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

Once you've entered safe mode successfully in CircuitPython 6.x, the LED will pulse yellow.

If you successfully enter safe mode on CircuitPython 7.x, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot. Press again to
exit safe mode.

Press any key to enter the REPL. Use CTRL-D to reload.
```

You can now edit the contents of the **CIRCUITPY** drive. Remember, *your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.*

Flash Resetting UF2

If your board ever gets into a really *weird* state and doesn't even show up as a disk drive when installing CircuitPython, try loading this 'nuke' UF2 which will do a 'deep clean' on your Flash Memory. **You will lose all the files on the board**, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

https://adafru.it/RLE

https://adafru.it/RLE

Project Code

As games go, Dragon Drop is rudimentary...it's more about presenting some starter code for your own project ideas, in a medium-size program. I was surprised how well CircuitPython could handle game animation on the little OLED screen.

CircuitPython 7.0.0 alpha 6 fixes some "scratchy" audio issues. If using an earlier version, it's worth upgrading!

Text Editor

If you just want to try out the game, skip ahead to the "Download the Project Bundle" section below.

If you plan to get into the source code to customize gameplay or learn more about it, Adafruit recommends using the **Mu** editor for editing your CircuitPython code. You can get more info in this guide (https://adafru.it/ANO).

Alternatively, you can use any text editor that saves simple text files.

Download the Project Bundle

Your project will use a specific set of CircuitPython libraries and the **code.py** file, along with a folder of graphics and sound files. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Make backups of any files on the board you want to keep, then drag the contents of the uncompressed bundle directory onto your MACROPAD board's **CIRCUITPY** drive, replacing any existing files or directories with the same names.

```
"""
Dragon Drop: a simple game for Adafruit MACROPAD. Uses OLED display in
portrait (vertical) orientation. Tap one of four keys across a row to
catch falling eggs before they hit the ground. Avoid fireballs.
"""

# pylint: disable=import-error, unused-import
import gc
import random
import time
import displayio
import adafruit_imageload
from adafruit_macropad import MacroPad
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text import label
from adafruit_progressbar.progressbar import HorizontalProgressBar
import board      # These three can be removed
```

```
import audiocore  # if/when MacroPad library
import audiopwmio # adds background audio


# CONFIGURABLES -----------------------

MAX_EGGS = 7          # Max count of all projectiles; some are fireballs
PATH = '/dragondrop/' # Location of graphics, fonts, WAVs, etc.


# UTILITY FUNCTIONS AND CLASSES --------

def background_sound(filename):
    """ Start a WAV file playing in the background (non-blocking). This
        func can be removed if/when MacroPad lib gets background audio. """
    # pylint: disable=protected-access
    macropad._speaker_enable.value = True
    audio.play(audiocore.WaveFile(open(PATH + filename, 'rb')))

def show_screen(group):
    """ Activate a given displayio group, pause until keypress. """
    macropad.display.show(group)
    macropad.display.refresh()
    # Purge any queued up key events...
    while macropad.keys.events.get():
        pass
    while True: # ...then wait for first new key press event
        key_event = macropad.keys.events.get()
        if key_event and key_event.pressed:
            return

# pylint: disable=too-few-public-methods
class Sprite:
    """ Class holds sprite (eggs, fireballs) state information. """
    def __init__(self, col, start_time):
        self.column = col                       # 0-3
        self.is_fire = (random.random() < 0.25) # 1/4 chance of fireballs
        self.start_time = start_time            # For drop physics
        self.paused = False


# ONE-TIME INITIALIZATION --------------

macropad = MacroPad(rotation=90)
macropad.display.auto_refresh = False
macropad.pixels.auto_write = False
macropad.pixels.brightness = 0.5
audio = audiopwmio.PWMAudioOut(board.SPEAKER) # For background audio

font = bitmap_font.load_font(PATH + 'cursive-smart.pcf')

# Create 3 displayio groups -- one each for the title, play and end screens.

title_group = displayio.Group()
title_bitmap, title_palette = adafruit_imageload.load(PATH + 'title.bmp',
                                                      bitmap=displayio.Bitmap,
                                                      palette=displayio.Palette)
```

```
title_group.append(displayio.TileGrid(title_bitmap, pixel_shader=title_palette,
                                      width=1, height=1,
                                      tile_width=title_bitmap.width,
                                      tile_height=title_bitmap.height))

# Bitmap containing eggs, hatchling and fireballs
sprite_bitmap, sprite_palette = adafruit_imageload.load(
    PATH + 'sprites.bmp', bitmap=displayio.Bitmap, palette=displayio.Palette)
sprite_palette.make_transparent(0)

play_group = displayio.Group()
# Bitmap containing five shadow tiles ('no shadow' through 'max shadow')
shadow_bitmap, shadow_palette = adafruit_imageload.load(
    PATH + 'shadow.bmp', bitmap=displayio.Bitmap, palette=displayio.Palette)
# Tilegrid with four shadow tiles; one per column
shadow = displayio.TileGrid(shadow_bitmap, pixel_shader=shadow_palette,
                            width=4, height=1, tile_width=16,
                            tile_height=shadow_bitmap.height, x=0,
                            y=macropad.display.height - shadow_bitmap.height)
play_group.append(shadow)
shadow_scale = 5 / (macropad.display.height - 20) # For picking shadow sprite
life_bar = HorizontalProgressBar((0, 0), (macropad.display.width, 7),
                                 value=100, min_value=0, max_value=100,
                                 bar_color=0xFFFFFF, outline_color=0xFFFFFF,
                                 fill_color=0, margin_size=1)
play_group.append(life_bar)
# Score is last object in play_group, can be indexed as -1
play_group.append(label.Label(font, text='0', color=0xFFFFFF,
                              anchor_point=(0.5, 0.0),
                              anchored_position=(macropad.display.width // 2,
                                                 10)))

end_group = displayio.Group()
end_bitmap, end_palette = adafruit_imageload.load(
    PATH + 'gameover.bmp', bitmap=displayio.Bitmap, palette=displayio.Palette)
end_group.append(displayio.TileGrid(end_bitmap, pixel_shader=end_palette,
                                    width=1, height=1,
                                    tile_width=end_bitmap.width,
                                    tile_height=end_bitmap.height))
end_group.append(label.Label(font, text='0', color=0xFFFFFF,
                             anchor_point=(0.5, 0.0),
                             anchored_position=(macropad.display.width // 2,
                                                90)))


# MAIN LOOP -- alternates play and end-game screens --------

show_screen(title_group) # Just do this once on startup

while True:

    # NEW GAME -------------------------

    sprites = []
    score = 0
    play_group[-1].text = '0' # Score text
    life_bar.value = 100
```

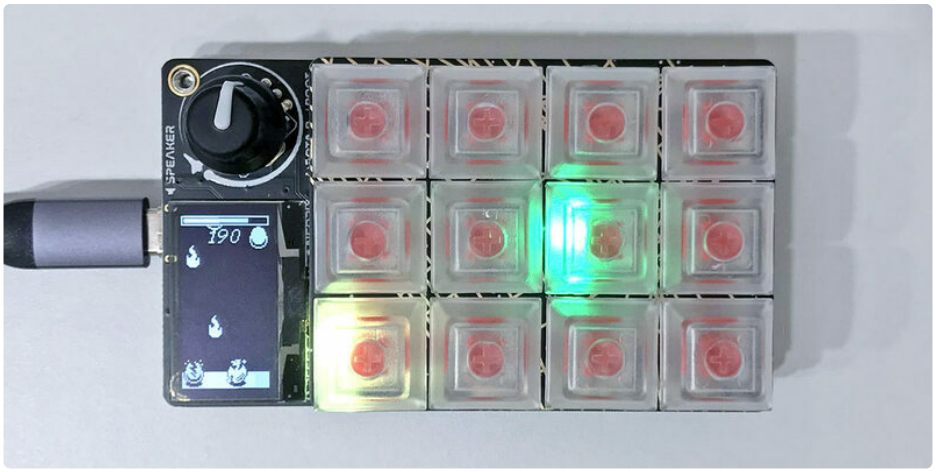
```
    life_bar.value = 100
    audio.stop()
    macropad.display.show(play_group)
    macropad.display.refresh()
    start = time.monotonic()

    # PLAY UNTIL LIFE BAR DEPLETED -----

    while life_bar.value > 0:
        now = time.monotonic()
        speed = 10 + (now - start) / 30   # Gradually speed up
        fire_sprite = 3 + int((now * 6) % 2.0) # For animating fire

        # Coalese any/all queued-up keypress events per column
        column_pressed = [False] * 4
        while True:
            event = macropad.keys.events.get()
            if not event:
                break
            if event.pressed:
                column_pressed[event.key_number % 4] = True

        # For determining upper/lower extents of active egg sprites per column
        column_min = [macropad.display.height] * 4
        column_max = [0] * 4

        # Traverse sprite list backwards so we can pop() without index problems
        for i in range(len(sprites) - 1, -1, -1):
            sprite = sprites[i]
            tile = play_group[i + 1] # Corresponding 1x1 TileGrid for sprite
            column = sprite.column
            elapsed = now - sprite.start_time # Time since add or pause event

            if sprite.is_fire:
                tile[0] = fire_sprite # Animate all flame sprites

            if sprite.paused:               # Sprite at bottom of screen
                if elapsed > 0.75:          # Hold position for 3/4 second,
                    for x in range(0, 9, 4): # then LEDs off,
                        macropad.pixels[x + sprite.column] = (0, 0, 0)
                    sprites.pop(i)          # and delete Sprite object and
                    play_group.pop(i + 1)   # element from displayio group
                    continue
                if not sprite.is_fire:
                    column_max[column] = max(column_max[column],
                                             macropad.display.height - 22)
            else: # Sprite in motion
                y = speed * elapsed * elapsed - 16
                # Track top of all sprites, bottom of eggs only
                column_min[column] = min(column_min[column], y)
                if not sprite.is_fire:
                    column_max[column] = max(column_max[column], y)
                tile.y = int(y) # Sprite's vertical pos. in play_group

                # Handle various catch or off-bottom actions...
                if sprite.is_fire:
                    if y >= macropad.display.height: # Off bottom of screen,
```

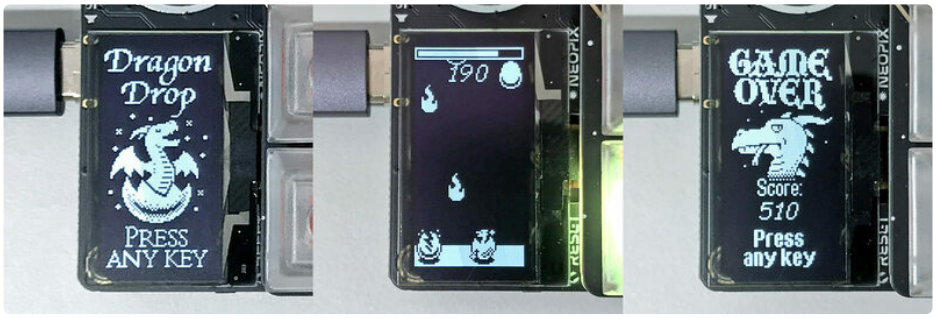
```
                    sprites.pop(i)              # remove fireball sprite
                    play_group.pop(i + 1)
                    continue
                if y >= macropad.display.height - 40:
                    if column_pressed[column]:
                        # Fireball caught, ouch!
                        background_sound('sizzle.wav') # I smell bacon
                        sprite.paused = True
                        sprite.start_time = now
                        tile.y = macropad.display.height - 20
                        life_bar.value = max(0, life_bar.value - 5)
                        for x in range(0, 9, 4):
                            macropad.pixels[x + sprite.column] = (255, 0, 0)
            else: # Is egg...
                if y >= macropad.display.height - 22:
                    # Egg hit ground
                    background_sound('splat.wav')
                    sprite.paused = True
                    sprite.start_time = now
                    tile.y = macropad.display.height - 22
                    tile[0] = 1 # Change sprite to broken egg
                    life_bar.value = max(0, life_bar.value - 5)
                    macropad.pixels[8 + sprite.column] = (255, 255, 0)
                elif column_pressed[column]:
                    if y >= macropad.display.height - 40:
                        # Egg caught at right time
                        background_sound('rawr.wav')
                        sprite.paused = True
                        sprite.start_time = now
                        tile.y = macropad.display.height - 22
                        tile[0] = 2 # Hatchling
                        macropad.pixels[4 + sprite.column] = (0, 255, 0)
                        score += 10
                        play_group[-1].text = str(score)
                    elif y >= macropad.display.height - 58:
                        # Egg caught too early
                        background_sound('splat.wav')
                        sprite.paused = True
                        sprite.start_time = now
                        tile.y = macropad.display.height - 40
                        tile[0] = 1 # Broken egg
                        life_bar.value = max(0, life_bar.value - 5)
                        macropad.pixels[sprite.column] = (255, 255, 0)

    # Select shadow bitmaps based on each column's lowest egg
    for i in range(4):
        shadow[i] = min(4, int(column_max[i] * shadow_scale))

    # Time to introduce a new sprite? 1/20 chance each frame, if space
    if (len(sprites) < MAX_EGGS and random.random() < 0.05 and
            max(column_min) > 16):
        # Pick a column randomly...if it's occupied, keep trying...
        while True:
            column = random.randint(0, 3)
            if column_min[column] > 16:
                # Found a clear spot. Add sprite and break loop
                sprites.append(Sprite(column, now))
```

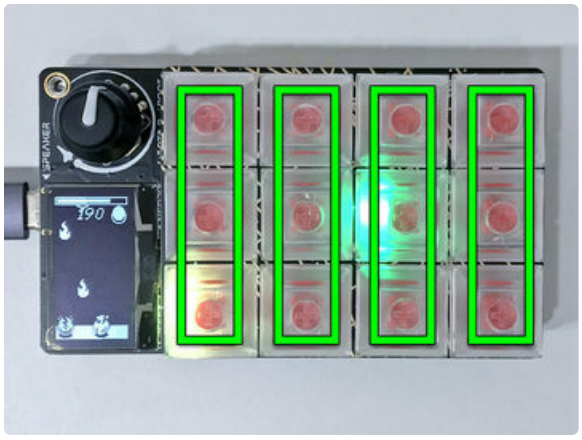
```
                play_group.insert(-2, displayio.TileGrid(sprite_bitmap,
                                                         pixel_shader=sprite_palette,
                                                         width=1, height=1,
                                                         tile_width=16,
                                                         tile_height=sprite_bitmap.height,
                                                         x=column * 16,
                                                         y=-16))
                break

    macropad.display.refresh()
    macropad.pixels.show()
    if not audio.playing:
        # pylint: disable=protected-access
        macropad._speaker_enable.value = False
    gc.collect()

    # Encoder button pauses/resumes game.
    macropad.encoder_switch_debounced.update()
    if macropad.encoder_switch_debounced.pressed:
        for n in (True, False, True): # Press, release, press
            while n == macropad.encoder_switch_debounced.pressed:
                macropad.encoder_switch_debounced.update()
        # Sprite start times must be offset by pause duration
        # because time.monotonic() is used for drop physics.
        now = time.monotonic() - now # Pause duration
        for sprite in sprites:
            sprite.start_time += now

# GAME OVER ------------------------

time.sleep(1.5) # Pause display for a moment
macropad.pixels.fill(0)
macropad.pixels.show()
# pylint: disable=protected-access
macropad._speaker_enable.value = False
# Pop any sprites from play_group (other elements remain, and sprites[]
# list is cleared at start of next game).
for _ in sprites:
    play_group.pop(1)
end_group[-1].text = str(score)
show_screen(end_group)
```

Gameplay

Turn MacroPad **sideways**, with the display and USB port on the **left**. This game uses the screen in a tall "portrait" orientation — we wanted to show that's possible.







From the title screen, tap any key to start.

During gameplay, there's just **four keys** to worry about...or four columns, any key within a column will work.

Dragon stuff — eggs and fireballs — fall from the sky! These are arranged in **four columns**, corresponding to the **four keys**.

"Catch" an egg by tapping the corresponding key just before it hits the ground. *But not too soon either.*

Don't catch the fireballs!

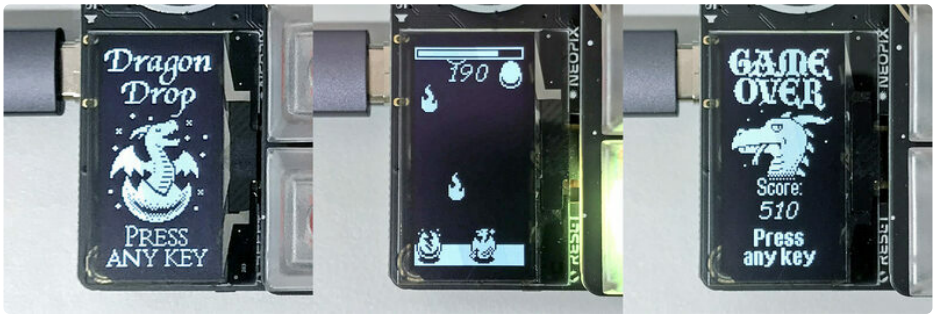
You'll get a bit of additional visual feedback from the NeoPixels under MacroPad's keys.

Earn 10 points (and a little "rawr!" dragon hatchling) for every good catch.

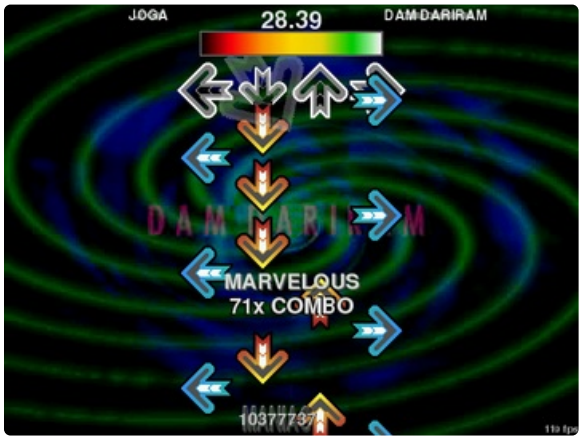
Your "energy" (shown as a bar across top of screen) is depleted slightly for broken eggs or burned hands. Game continues until energy runs out. The pace gradually increases.

A closing screen shows your final score, and you can cap a key to try again.

Reflection



So I wanted to write a **small but reasonably complete** game to show an "unexpected" (not-keyboard-macros) use of MacroPad.



Lately I've been enjoying VR rhythm games like *Beat Saber* and *Synth Riders,* and my initial idea was to capture just a tiny slice of that, albeit in two dimensions. Go back a couple generations, and prior rhythm games like *Guitar Hero* and *Dance Dance Revolution* were essentially that, and would suit MacroPad's keys.

(screenshot: pydance (https://adafru.it/UbW))

But even this would be **Too Much Project** ... building the "maps" to synchronize music and actions would be *A Development Ordeal*. I wanted something that could be written in a couple days, just a few bitmaps and a sensible amount of code. Not a serious game, just enough to show that these things can be done if you want.

Peeling back layers was like a reverse chronological walk through game history. *Tetris* might be doable but a bit more code than desired. Full-on game music would be complex. And so forth, until reaching the irreducible "catch things falling from above" play mechanic, with some "avoid these other things" to spice it up. It's rudimentary, but maybe I could add appeal with some **dragons**. *Everyone* loves dragons!

The resulting flavor is reminiscent of some early “catch” games like Activision’s *Kaboom* for the Atari 2600 (catch bombs) or the arcade game that inspired it, Atari’s *Avalanche* (catch rocks). Amusingly, I learned afterward that the original concept for *Avalanche* had falling eggs, but play testers didn’t like that. Oops.

That got me thinking how early video games always had elaborate box art and a whole *back story,* even with simple games. Partly of course because the graphics were so crude...but also, video games were such a foreign concept, it was necessary to create some emotional “hook” to an otherwise abstract thing. Early players would be lost without it.

Nobody asks what the cubes are in *Beat Saber* or why it’s so vital to chop them up. The concept is

entirely abstract, but needs no back story...it's intuitive to play and the action is viscerally appealing, what more do you need? "This is a rhythm game. Have fun!"

Nearly half a century in, video games evolved a stable of tropes — the maze game, the shooting game, puzzle game, RPG and so forth — much as movies and media tap into established tropes to avoid lengthy exposition...it simplified the "Gameplay" page significantly. You don't always notice it when looking at a single specimen, but walk backward through a game's or film's predecessors and you can see a progression as things congeal.