Space Aliens - CircuitPython Game

Mr. Coxall

Jan 23, 2020

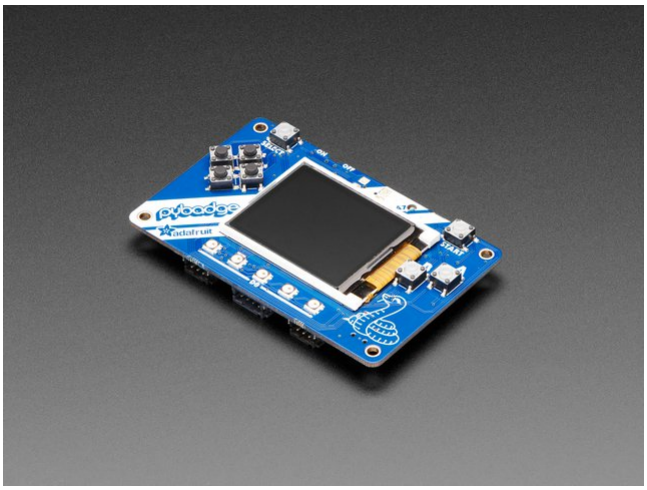
Contents

In this project we will be making an old school style video game for the Adafruit PyBadge. We will be using CircuitPython and the stage library to create a Space Invaders like game. The stage library makes it easy to make classic video games, with helper libraries for sound, sprites and collision detection. The game will also work on other variants of PyBadge hardware, like the PyGamer and the EdgeBadge. The full completed game code with all the assets can be found here.

The guide assumes that you have prior coding experience, hopefully in Python. It is designed to use just introductory concepts. No Object Oriented Programming (OOP) are used so that students in particular that have completed their first course in coding and know just variables, if statements, loops and functions will be able to follow along.

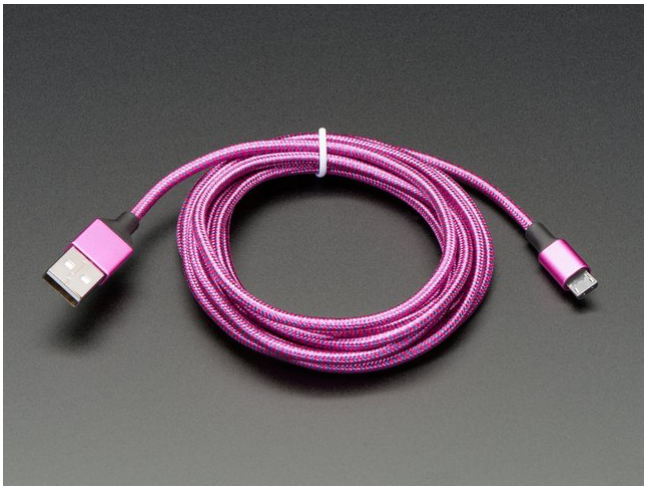
Parts

You will need the following items:



Adafruit PyBadge for MakeCode Arcade, CircuitPython or Arduino

PRODUCT ID: 4200

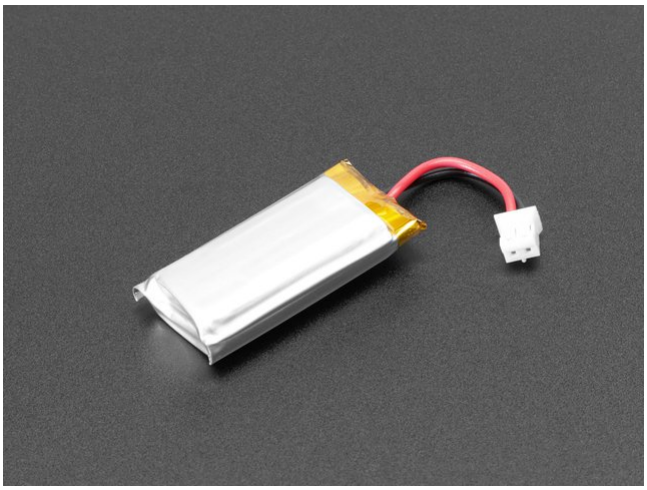


Pink and Purple Braided USB A to Micro B Cable - 2 meter long

PRODUCT ID: 4148

So you can move your CircuitPython code onto the PyBadge.

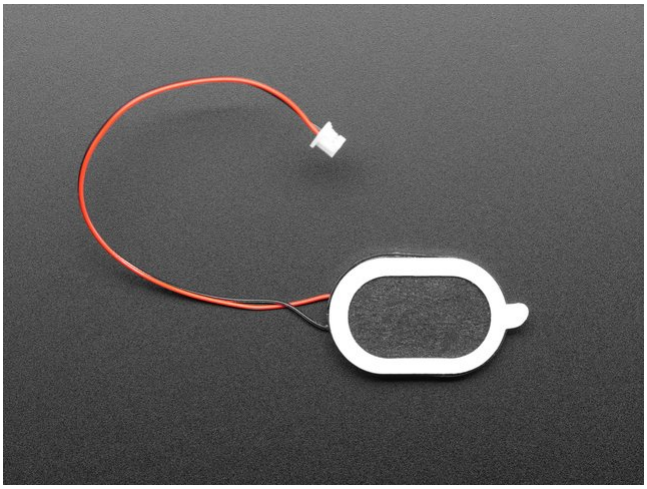
You might also want:



Lithium Ion Polymer Battery Ideal For Feathers - 3.7V 400mAh

PRODUCT ID: 3898

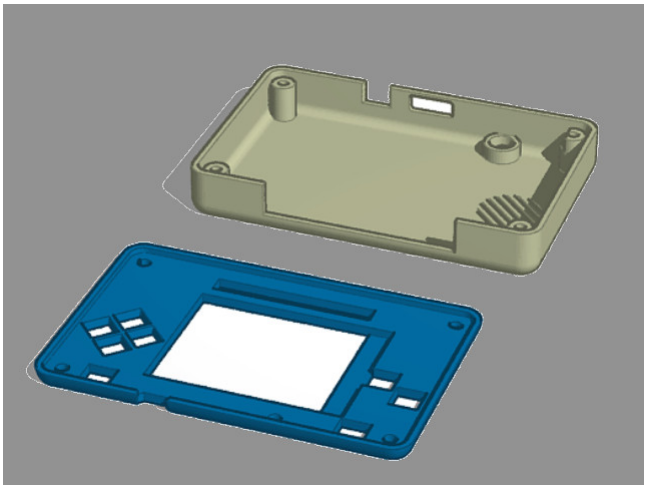
So that you can play the game without having it attached to a computer with a USB cable.



Mini Oval Speaker - 8 Ohm 1 Watt

PRODUCT ID: 3923

If you want lots of sound. Be warned, the built in speaker is actually pretty loud.



3D Printed Case

I did not create this case. I altered Adafruit's design. One of the screw posts was hitting the built in speaker and the

case was not closing properly. I also added a piece of plastic over the display ribbon cable, to keep it better protected. You will need 4 x 3M screws to hold the case together.

CHAPTER 1

Install CircuitPython

Fig. 1: Clearing the PyBadge and loading the CircuitPython UF2 file

Before doing anything else, you should delete everything already on your PyBadge and install the latest version of CircuitPython onto it. This ensures you have a clean build with all the latest updates and no leftover files floating around. Adafruit has an excellent quick start guide here to step you through the process of getting the latest build of CircuitPython onto your PyBadge. Adafruit also has a more detailed comprehensive version of all the steps with complete explanations here you can use, if this is your first time loading CircuitPython onto your PyBadge.

Just a reminder, if you are having any problems loading CircuitPython onto your PyBadge, ensure that you are using a USB cable that not only provides power, but also provides a data link. Many USB cables you buy are only for charging, not transfering data as well. Once the CircuitPython is all loaded, come on back to continue the tutorial.

CHAPTER 2

Your IDE

One of the great things about CircuitPython hardware is that it just automatically shows up as a USB drive when you attach it to your computer. This means that you can access and save your code using any text editor. This is particularly helpful in schools, where computers are likely to be locked down so students can not load anything. Also students might be using Chromebooks, where only "authorized" Chrome extensions can be loaded.

If you are working on a Chromebook, the easiest way to start coding is to just use the built in Text app. As soon as you open or save a file with a `*.py` extension, it will know it is Python code and automatically start syntax highlighting.

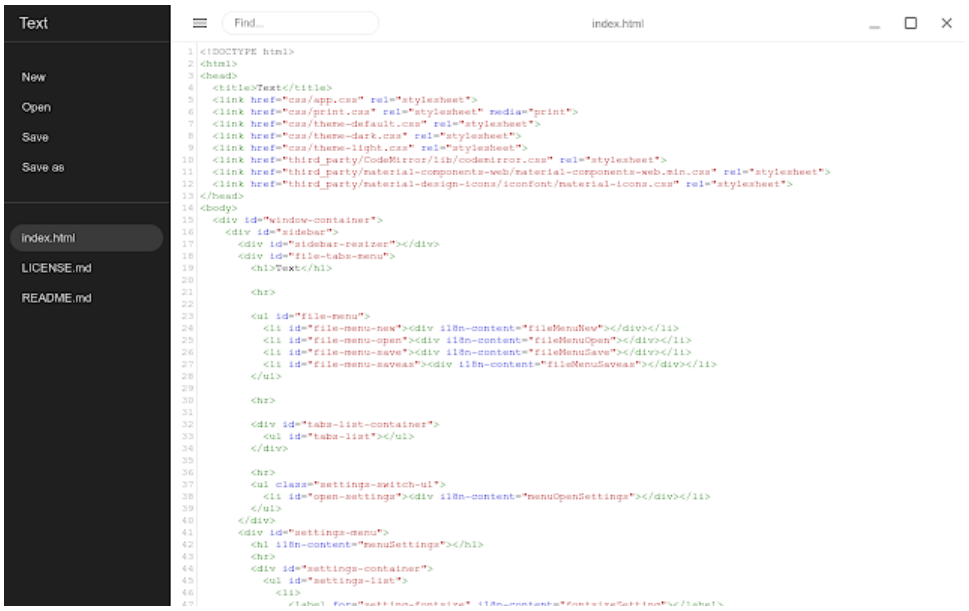


Fig. 1: Chromebook Text app

If you are using a non-Chromebook computer, your best beat for an IDE is Mu. You can get it for Windows, Mac, Raspberry Pi and Linux. It works seamlessly with CircuitPython and the serial console will give you much needed debugging information. You can download Mu here.

Fig. 2: Mu IDE

Since with CircuitPython devices you are just writing Python files to a USB drive, you are more than welcome to use any IDE that you are familiar using.

2.1 Hello, World!

Yes, you know that first program you should always run when starting a new coding adventure, just to ensure everything is running correctly! Once you have access to your IDE and you have CircuitPython loaded, you should make sure everything is working before you move on. To do this we will do the traditional "Hello, World!" program. By default CircuitPython looks for a file called `code.py` in the root directory of the PyBadge to start up. You will place the following code in the `code.py` file:

```
print("Hello, World!")
```

As soon as you save the file onto the PyBadge, the screen should flash and you should see something like:

Although this code does work just as is, it is always nice to ensure we are following proper coding conventions, including style and comments. Here is a better version of Hello, World! You will notice that I have a call to a `main()` function. This is common in Python code but not normally seen in CircuitPython. I am including it because by breaking the code into different functions to match different scenes, eventually will be really helpful.

```
#!/usr/bin/env python3

# Created by : Mr. Coxall
# Created on : January 2020
# This program prints out Hello, World! onto the PyBadge


```

(continues on next page)

Fig. 3: Hello, World! program on PyBadge

(continued from previous page)

```
8  def main():
9      # this function prints out Hello, World! onto the PyBadge
10     print("Hello, World!")
11
12
13 if __name__ == "__main__":
14     main()
```

Congratulations, we are ready to start.

CHAPTER 3

Image Banks

Before we can start coding a video game, we need to have the artwork and other assets. The stage library from CircuitPython we will be using is designed to import an "image bank". These image banks are 16 sprites staked on top of each other, each with a resolution of 16x16 pixels. This means the resulting image bank is 16x256 pixels in size. Also the image bank **must** be saved as a 16-color BMP file, with a pallet of 16 colors. To get a sprite image to show up on the screen, we will load an image bank into memory, select the image from the bank we want to use and then tell CircuitPython where we would like it placed on the screen.

Fig. 1: Image Bank for Shooter Shootout

For sound, the stage library can play back `*.wav` files in PCM 16-bit Mono Wave files at 22KHz sample rate. Adafruit has a great learning guide on how to save your sound files to the correct format here.

If you do not want to get into creating your own assets, other people have already made assets available to use. All the assets for this guide can be found in the GitHub repo here:

- shooter shootout image bank
- 1st menu image bank
- 2nd menu image bank
- 3rd menu image bank
- coin sound
- pew sound
- boom sound
- crash sound

Please download the assets and place them on the PyBadge, in the root directory. Your previoud "Hello, World!" program should restart and run again each time you load a new file onto the PyBadge, hopefully with no errors once more.

Assets from other people can be found here.

CHAPTER 4

Game

X

4.1 Background

The background for this game is pretty simple, all that was used is the first image of the image bank and place it all around the scene

4.2 Tank

In this project we used a series of 16 pixels by 16 pixels picture to act as our characters, enemies and such. All we did was make a 16x256 file and inserted the sprites along the picture and then uploaded to the pybadge.

Next we continued to use the sprite and find a way to move it around, and this was how we did it.

```
if keys & ugame.K_RIGHT != 0:
    # if tank moves off right screen, move it back
    if tank.x > constants.SCREEN_X - constants.SPRITE_SIZE:
        tank.x = constants.SCREEN_X - constants.SPRITE_SIZE
    # else move tank right
    else:
        tank.move(tank.x + constants.BALL_SPEED, tank.y)
        tank.set_frame(frame=None,rotation=1)
        tank_direction = "right"

# if left D-Pad is pressed
if keys & ugame.K_LEFT != 0:
    # if tank moves off left screen, move it back
```

(continues on next page)

(continued from previous page)

```
    if tank.x < 0:
        tank.x = 0
    # else move tank left
    else:
        tank.move(tank.x - constants.BALL_SPEED, tank.y)
        tank.set_frame(frame=None,rotation=3)
        tank_direction = "left"

if keys & ugame.K_UP != 0:
    # if tank moves off up screen, move it back
    if tank.y > constants.SCREEN_Y - constants.SPRITE_SIZE:
        tank.y = constants.SCREEN_Y - constants.SPRITE_SIZE
    # else move tank up
    else:
        tank.move(tank.x, tank.y - 1)
        tank.set_frame(frame=None,rotation=0)
        tank_direction = "up"

# if left D-Pad is pressed
if keys & ugame.K_DOWN != 0:
    # if tank moves off down screen, move it back
    if tank.y < 0:
        tank.y = 0
    # else move tank down
    else:
        tank.move(tank.x, tank.y + 1)
        tank.set_frame(frame=None,rotation=2)
        tank_direction = "down"
```

Next we had to update the direction of the ship which goes with the directins the lasers are shooting, we did so by assigning a direction with a variable to change direction of ship by rotating and shooting direction.

```
if a_button == constants.button_state["button_just_pressed"]:
    # fire a laser, if we have enough power (meaning we have not used up all the␣
↪lasers)
    for laser_number in range(len(lasers)):
        if lasers[laser_number].x < 0:
            lasers[laser_number].move(tank.x, tank.y)
            lasers_direction[laser_number] = tank_direction
            sound.stop()
            sound.play(pew_sound)
            break

# each frame move the lasers, that have been fired, up

# first make all the neopixels yellow, then make them green if it is moving up
lasers_moving_counter = -1
for pixel_number in range(0, 5):
    pixels[pixel_number] = (0, 10, 0)

for laser_number in range(len(lasers)):
    if lasers[laser_number].x > 0 :
        laser_move_direction = None
        if lasers_direction[laser_number] == "down":
            lasers[laser_number].move(lasers[laser_number].x, lasers[laser_number].y␣
↪+ constants.ATTACK_SPEED)
```

(continues on next page)

(continued from previous page)

```
        elif lasers_direction[laser_number] == "up":
            lasers[laser_number].move(lasers[laser_number].x, lasers[laser_number].y -
↪ constants.ATTACK_SPEED)
        elif lasers_direction[laser_number] == "left":
            lasers[laser_number].move(lasers[laser_number].x - constants.ATTACK_SPEED,
↪ lasers[laser_number].y)
        elif lasers_direction[laser_number] == "right":
            lasers[laser_number].move(lasers[laser_number].x + constants.ATTACK_SPEED,
↪ lasers[laser_number].y)


        lasers_moving_counter = lasers_moving_counter + 1
        pixels[lasers_moving_counter] = (10, 10 - (2 * lasers_moving_counter + 2), 0)
        if lasers[laser_number].y < constants.OFF_TOP_SCREEN:
            lasers[laser_number].move(constants.OFF_SCREEN_X, constants.OFF_SCREEN_Y)
        if lasers[laser_number].y > 128:
            lasers[laser_number].move(constants.OFF_SCREEN_X, constants.OFF_SCREEN_Y)
        if lasers[laser_number].x > 160:
            lasers[laser_number].move(constants.OFF_SCREEN_X, constants.OFF_SCREEN_Y)
        if lasers[laser_number].y < 1:
            lasers[laser_number].move(constants.OFF_SCREEN_X, constants.OFF_SCREEN_Y)

if lasers_moving_counter == 4:
    for pixel_number in range(0, 5):
        pixels[pixel_number] = (10, 0, 0)
pixels.show()
```

X

CHAPTER 5

Menu System

X

5.1 Start Scene

For the start scene there are two parts, the picture and the button loop. The picture is just like the splash screens in which we put sprites together on a screen. And for the loop all we did was a while True loop and put it so that when they press start it detects and goes to game scene.

And this is the code we used:

5.2 Splash Scene

For our splash screens, we have to have two of them: MT Game Studios and our company studios. For a splash screen to work all we had to do was take a 160x128 picture and create our design, then we cut it up into 16x16 pictures and put it in a image bank. Once it's in an image bank we put together like a puzzle on the pybadge, like so:

Here is the MT Game Studio splash screen:

And here is the PP Company splash screen:

5.3 Game Over Scene

Finally, we have the game over scene which is exactly like the menu screen: