Space Aliens - CircuitPython Game

Mr. Coxall

Jan 09, 2020

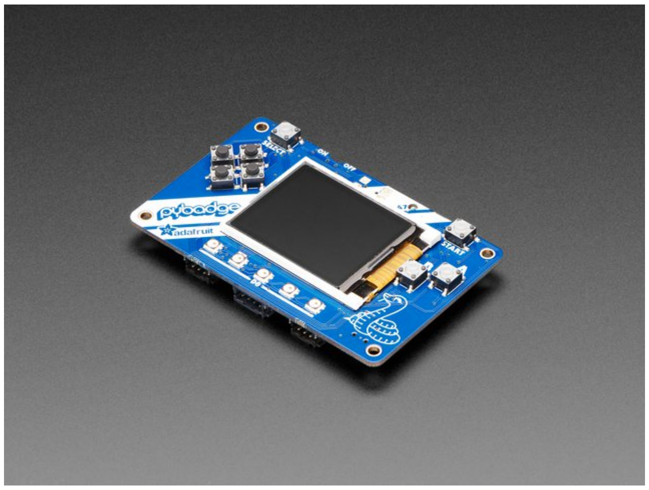
Contents

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.

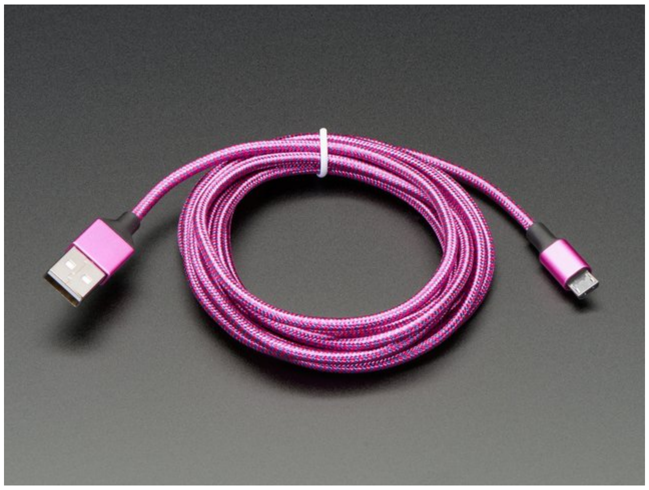
Parts

You will need the following items:



Adafruit PyBadge for MakeCode Arcade, CircuitPython or Arduino

PRODUCT ID: 4200

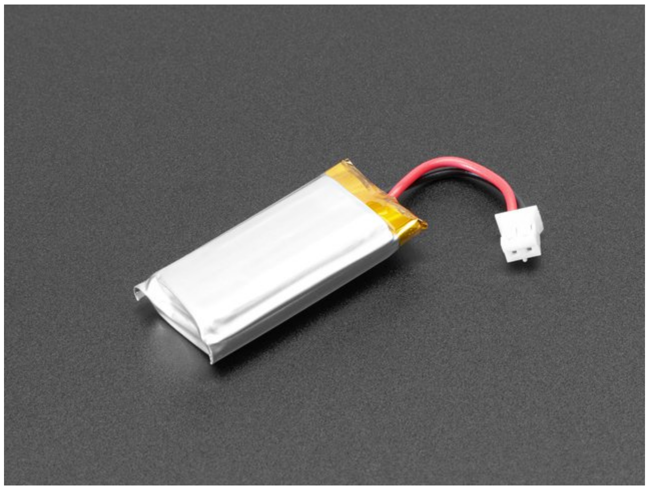


Pink and Purple Braided USB A to Micro B Cable - 2 meter long

PRODUCT ID: 4148

So you can move your CircuitPython code onto the PyBadge.

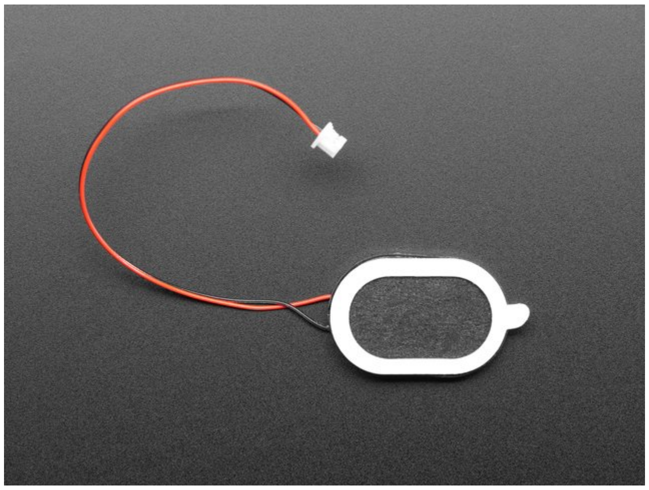
You might also want:



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

PRODUCT ID: 3898

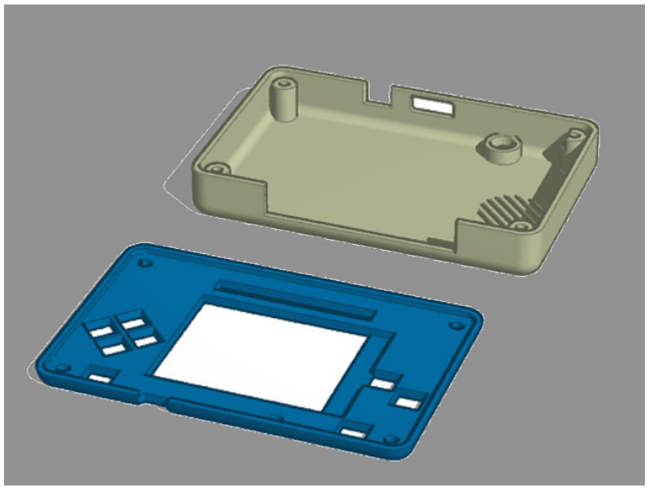
So that you can play the game without having it attached to a computer with a USB cable.



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.

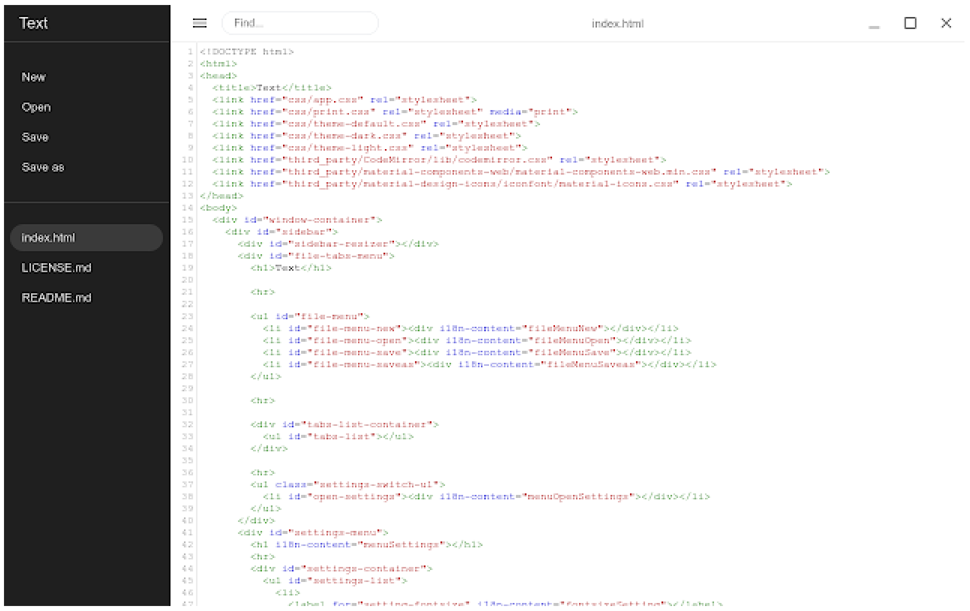
Just a reminder, if you are having any problems loading CircuitPython onto your PyBadge, ensure that you are using a USB cable that not only provides power, but also provides a data link. Many USB cables you buy are only for charging, not transfering data as well. Once the CircuitPython is all loaded,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

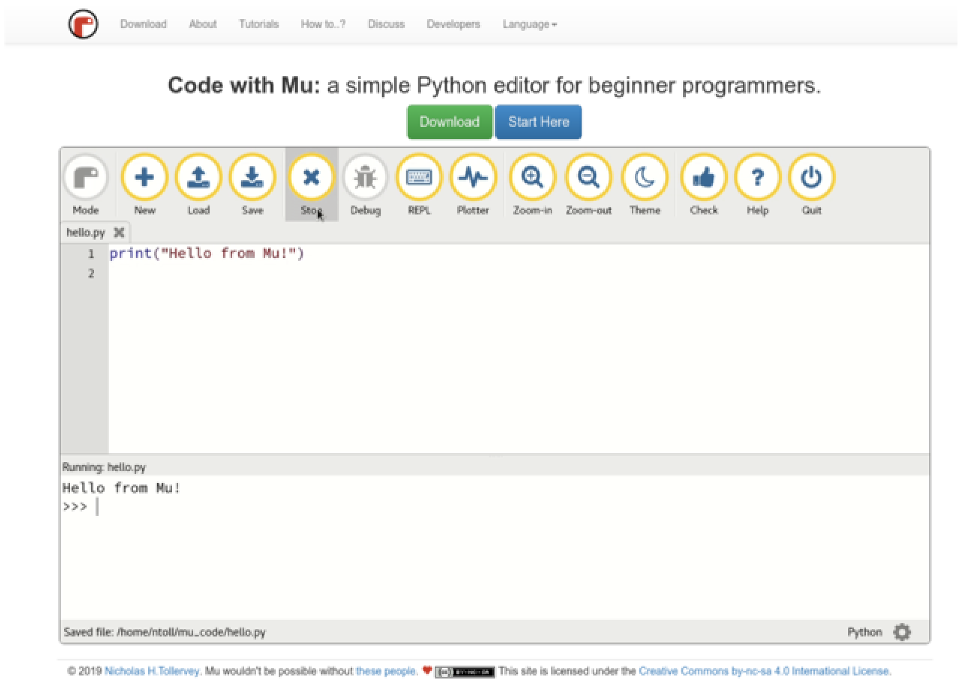
If you are working on a Chromebook, the easiest way to start coding is to just use the built in Text app. As soon as you open or save a file with a `*.py` extension, it will know it is Python code and automatically start syntax highlighting.



```
<!DOCTYPE html>
<html>
<head>
  <title>Text</title>
  <link href="css/app.css" rel="stylesheet">
  <link href="css/print.css" rel="stylesheet" media="print">
  <link href="css/theme-default.css" rel="stylesheet">
  <link href="css/theme-dark.css" rel="stylesheet">
  <link href="css/theme-light.css" rel="stylesheet">
  <link href="third_party/CodeMirror/lib/codemirror.css" rel="stylesheet">
  <link href="third_party/material-components-web/material-components-web.min.css" rel="stylesheet">
  <link href="third_party/material-design-icons/iconfont/material-icons.css" rel="stylesheet">
</head>
<body>
  <div id="window-container">
    <div id="sidebar">
      <div id="sidebar-resizer"></div>
      <div id="file-tabs-menu">
        <h1>Text</h1>

        <hr>

        <ul id="file-menu">
          <li id="file-menu-new"><div i18n-content="fileMenuNew"></div></li>
          <li id="file-menu-open"><div i18n-content="fileMenuOpen"></div></li>
          <li id="file-menu-save"><div i18n-content="fileMenuSave"></div></li>
          <li id="file-menu-saveas"><div i18n-content="fileMenuSaveas"></div></li>
        </ul>

        <hr>

        <div id="tabs-list-container">
          <ul id="tabs-list"></ul>
        </div>

        <hr>
        <ul class="settings-switch-ul">
          <li id="open-settings"><div i18n-content="menuOpenSettings"></div></li>
        </ul>
      </div>
      <div id="settings-menu">
        <h1 i18n-content="menuSettings"></h1>
        <hr>
        <div id="settings-container">
          <ul id="settings-list">
            <li>
              <label for="setting-fontsize" i18n-content="fontsizeSetting"></label>
```

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.



Code with Mu: a simple Python editor for beginner programmers.

```
print("Hello from Mu!")
```

Running: hello.py

```
Hello from Mu!
>>>
```

Saved file: /home/ntoll/mu_code/hello.py

© 2019 Nicholas H.Tollervey. Mu wouldn't be possible without these people. ♥ This site is licensed under the Creative Commons by-nc-sa 4.0 International License.

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!")
```

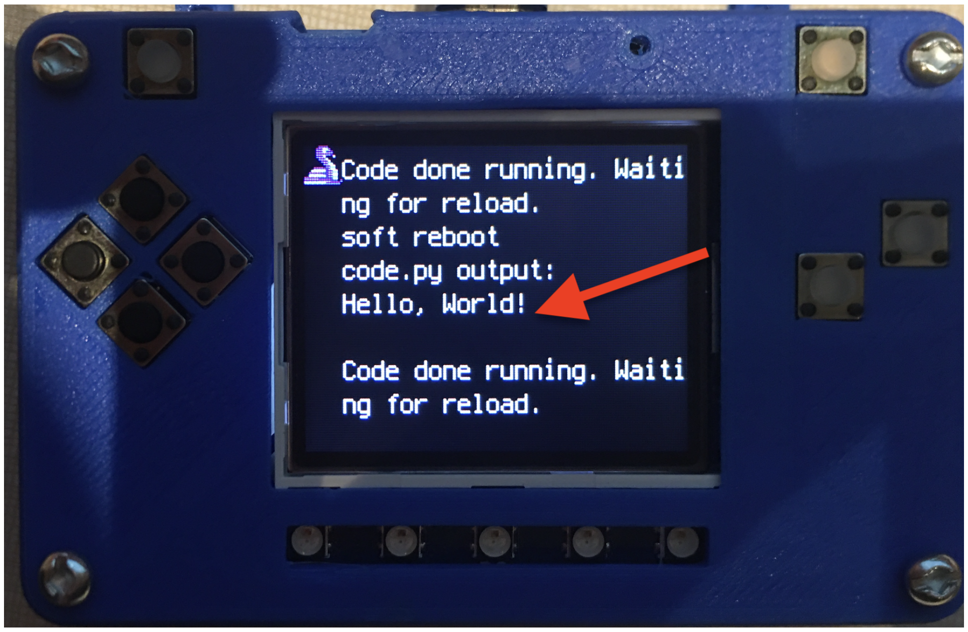
As soon as you save the file onto the PyBadge, the screen should flash and you should see something like:

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.

Listing 1: code.py

```
#!/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)



```
Code done running. Waiti
ng for reload.
soft reboot
code.py output:
Hello, World!

Code done running. Waiti
ng for reload.
```

Fig. 3: Hello, World! program on PyBadge

(continued from previous page)

```python


def game_scene():
    # this function prints out Hello, World! onto the PyBadge
    print("Hello, World!")


if __name__ == "__main__":
    game_scene()
```

Congratulations, we are ready to start.

Note: Full code and assets that can be copied onto PyBadge for this step can be found here.

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. 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 Space Aliens

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:

- space aliens 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.

Note: Full code and assets that can be copied onto PyBadge for this step can be found here.

CHAPTER 4

Background

Now that we have our image bank the next step is to actually get something showing up on the screen. The first thing we will do is fill the background with the first (or zeroith, if you are a computer scientist) image in our image bank. We will take the image and just place it over and over again, tiling it all over the screen. Since the image is just a plain white square, this will just make the entire background white for us.

Luckaly CircuitPython has some built in libraries that will make this much easier for us. We will be using 2, `ugame` and `stage`. We will import these 2 libraries at the top of our code and then we will write the following code to tile the background with the zeroith image:

Listing 1: code.py

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

# Created by: Mr. Coxall
# Created on: Sep 2019
# This file is the "Space Aliens" game
#   for CircuitPython

import ugame
import stage


def game_scene():
    # this function is the game scene

    # an image bank for CircuitPython
    image_bank_1 = stage.Bank.from_bmp16("./space_aliens.bmp")

    # sets the background to image 0 in the bank
    background = stage.Grid(image_bank_1, 160, 120)

    # create a stage for the background to show up on
    #   and set the frame rate to 60fps
    game = stage.Stage(ugame.display, 60)
```

(continues on next page)

(continued from previous page)

```python
    # set the layers, items show up in order in this list
    game.layers = [background]
    # render the background and inital location of sprite list
    # most likely you will only render background once per scene
    game.render_block()


if __name__ == "__main__":
    game_scene()
```

Note: Full code and assets that can be copied onto PyBadge for this step can be found here.

CHAPTER 5

Sprites

Now that we have the background, lets place a single sprite on the screen. Sprites are handled differently than the background, they can be placed anywhere we want on the sceen, not just tiled. We will create a varaible that holds a specific image from the image bank. We will also create a **list** that will hold all the sprites we want to paint onto the screen. We will take our ship variable and add it to this list.

Painting the entire screen each frame is not practical. The refresh rate just is not fast enough. Instead when the scene loads we will paint the background and then all the sprites on top of it. After that, each frame we will just redraw the sprites in the sprites list. This will let us have a much faster refesh rate. To accomplish this after we first paint the entire screen, we will create a loop, usually known as a gaming loop, that will keep running the code to update our sprites.

Listing 1: code.py

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

# Created by: Mr. Coxall
# Created on: Sep 2019
# This file is the "Space Aliens" game
#   for CircuitPython

import ugame
import stage


def game_scene():
    # this function is the game scene

    # create a list fo rall the sprites to be held in
    sprites = []

    # an image bank for CircuitPython
    image_bank_1 = stage.Bank.from_bmp16("./space_aliens.bmp")

    # sets the background to image 0 in the bank
```

(continues on next page)

(continued from previous page)

```
    background = stage.Grid(image_bank_1, 160, 120)

    # the ship sprite(the 5th image, starting to count at 0)
    #   will show up at location (72,56), which places it in middle of screen
    ship = stage.Sprite(image_bank_1, 5, 72, 56)
    # add the ship sprite to the sprite list
    sprites.append(ship)

    # create a stage for the background to show up on
    #   and set the frame rate to 60fps
    game = stage.Stage(ugame.display, 60)
    # set the layers, items show up in order, sprites and background behind
    game.layers = sprites + [background]
    # render the background and inital location of sprites list
    # most likely you will only render background once per scene
    # it is slow to draw
    game.render_block()

    # repeat forever, game loop
    while True:
        # get user input

        # update game logic

        # redraw sprites list
        game.render_sprites(sprites)
        game.tick() # wait until refresh rate finishes


if __name__ == "__main__":
    game_scene()
```

Note: Full code and assets that can be copied onto PyBadge for this step can be found here.