Space Aliens - CircuitPython Game

Mr. Coxall

Jan 23, 2020

Contents

1 Install CircuitPython **5**

2 Your IDE **7**
2.1 Hello, World! 8

3 Image Banks **11**

4 Game **13**
4.1 Background 13
4.2 Plane Selection 15
4.3 Show Airplane 16
4.4 Move Airplane 18
4.5 Enemy 18
4.6 Shoot the Missile 19

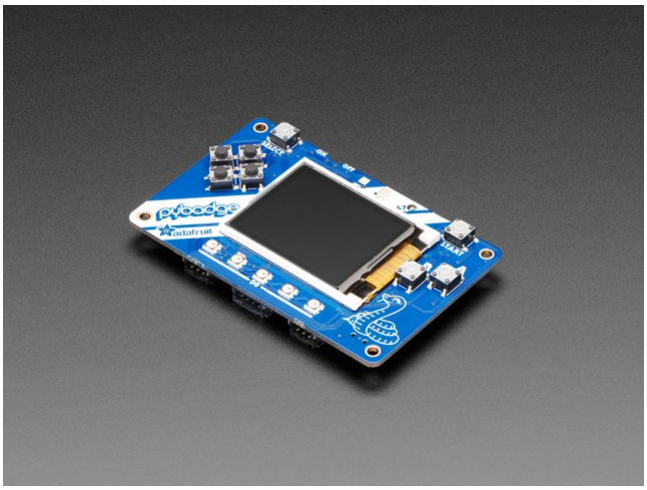
5 Menu System **21**
5.1 Splash Scene 21
5.2 Start Scene 23
5.3 Help Scene 24
5.4 Game Over Scene 25

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.

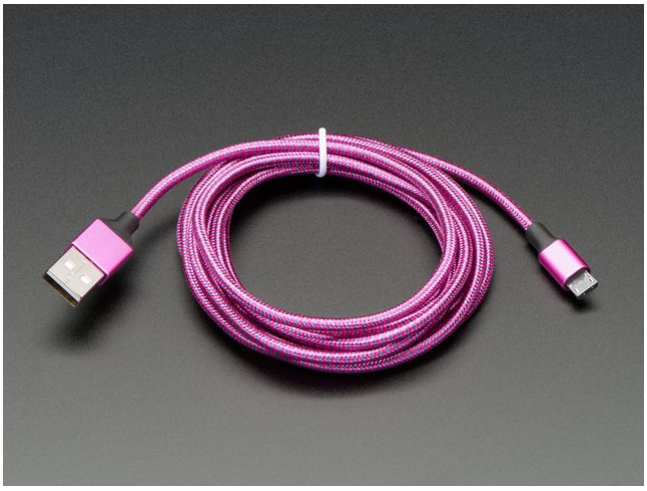
Parts

You will need the following items:



Adafruit PyBadge for MakeCode Arcade, CircuitPython or Arduino

PRODUCT ID: 4200

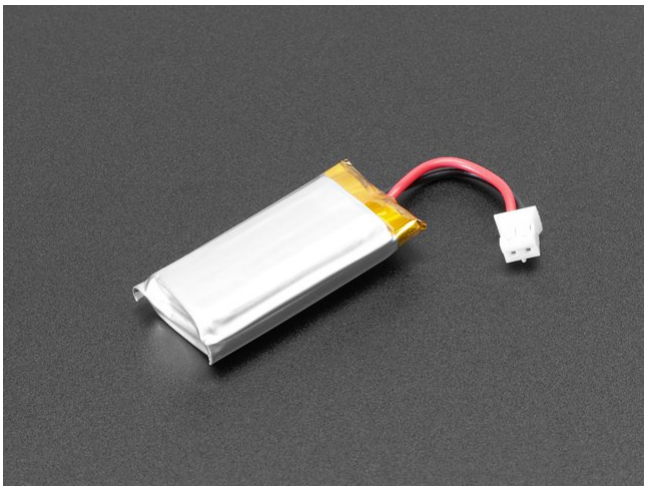


Pink and Purple Braided USB A to Micro B Cable - 2 meter long

PRODUCT ID: 4148

So you can move your CircuitPython code onto the PyBadge.

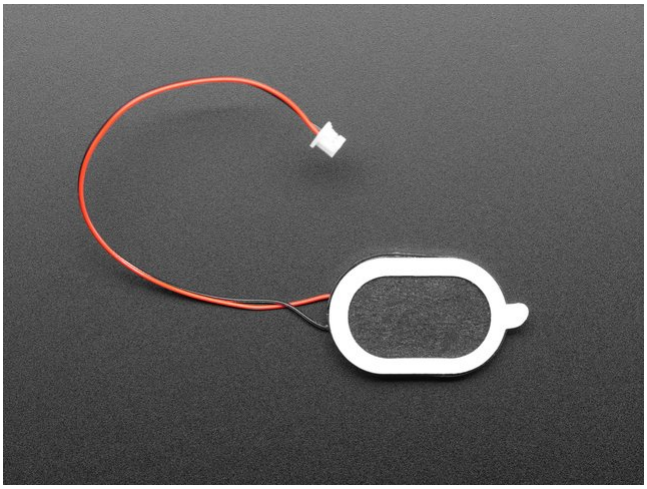
You might also want:



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

PRODUCT ID: 3898

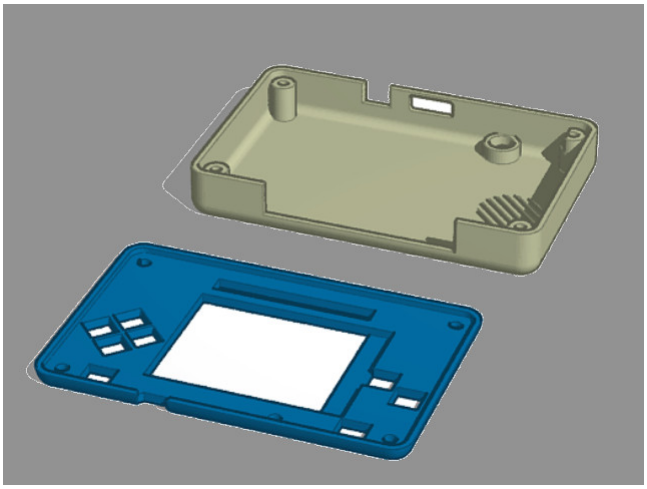
So that you can play the game without having it attached to a computer with a USB cable.



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.

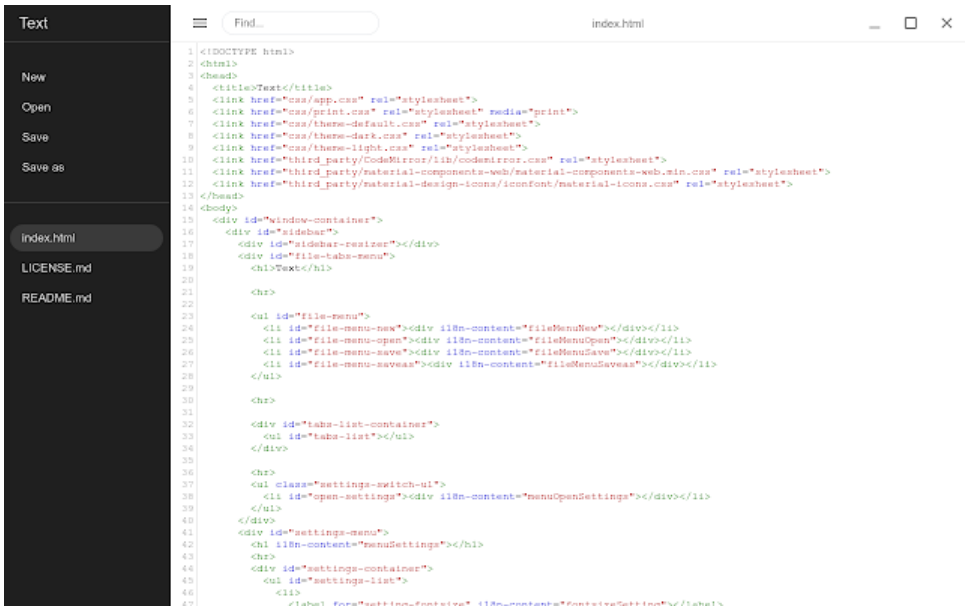


Fig. 1: Chromebook Text app

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 : Jay Lee
# 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 Avoid or Shoot

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:

- avoid or shoot image bank
- coin sound
- missile sound
- boom sound
- bird sound
- constants.py

This is constants file to be used in this game.

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

This section describes things realted to gameplay. Go through the post below.

4.1 Background

Avoid or Shoot needs a background. This code is puts the first image in the background. The first image is 16 x 16 px image at the top. in this case the first image of image bank is white colour image. Of course when you save the file, save it as `code.py` file:

```
image_bank_1 = stage.Bank.from_bmp16("avoid_or_shoot.bmp")

background = stage.Grid(image_bank_1, 10, 8)
```

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

This code will not work. The code above has a lot to do. Here is a better version that shows the background. You can see that you called the `main()` function. This is common in python code but usually not visible in CircuitPython. I am including it because by breaking the code into different functions to match different scenes, eventually will be really helpful.

```
for y_location in range(8):
    for x_location in range(16):
        if y_location == 1:
            tile_picked = random.randint(1, 3)
        else:
            tile_picked = random.randint(1, 1)
        background.tile(x_location, y_location, tile_picked)
```

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

This codes replace the background of pybadge with another 16x16 image. pybadge contains 16 images on the x-axis and 8 image on the y-axis in the image bank. It fills that background with a for loop. When the second image is filled on the y-axis, it randomly fills the second to fourth images in the image bank.

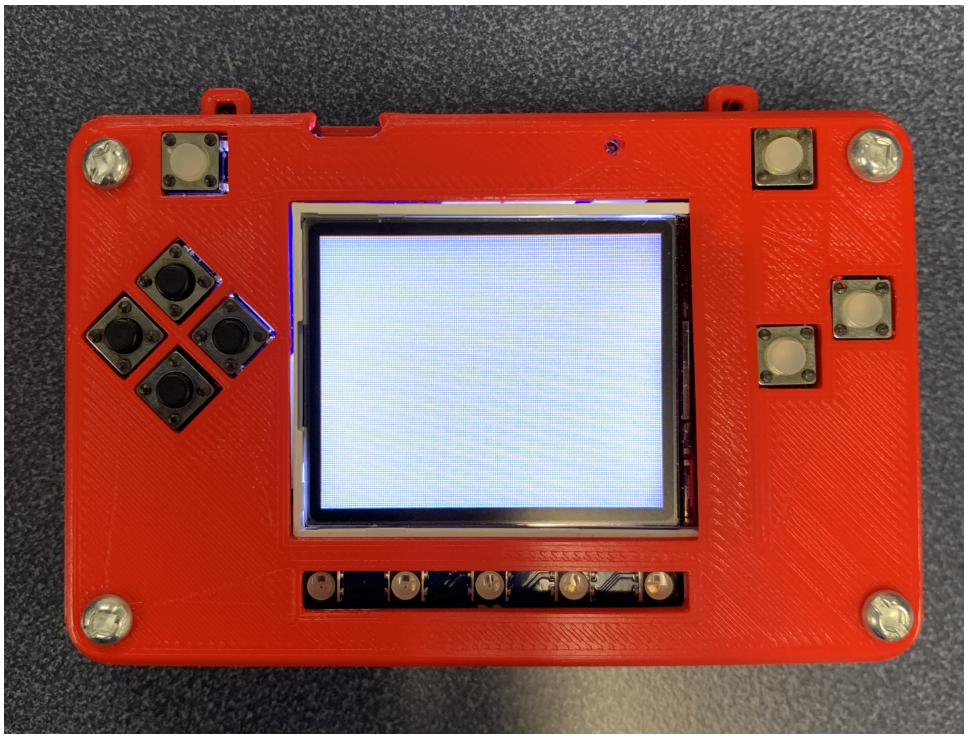


Fig. 1: White background on PyBadg

Fig. 2: Game Background on PyBadge

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

# Created by : Jay Lee
# Created on : Jan 2020
# This program display background of pybadge

import ugame
import stage
import random


def main():
    # this function display background of pybadge
    image_bank_1 = stage.Bank.from_bmp16("avoid_or_shoot.bmp")
    background = stage.Grid(image_bank_1, 10, 8)

    for y_location in range(8):
        for x_location in range(16):
            if y_location == 1:
                tile_picked = random.randint(1, 3)
            else:
                tile_picked = random.randint(1, 1)
            background.tile(x_location, y_location, tile_picked)

    game = stage.Stage(ugame.display, 60)
    game.layers = [background]
    game.render_block()

    while True:
        pass


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

This is the basis of the background function.

4.2 Plane Selection

I created the selection scene that can choose the plane by user. Planes are placed in quadrant on the screen. And the select box has to move exactly on the plane's position.

```
while True:
    keys = ugame.buttons.get_pressed()

    if keys & ugame.K_UP != 0:
        if select_box1.y != constants.SCREEN_Y / 4:
            select_box1.move(int(select_box1.x),
                             int(constants.SCREEN_Y / 4))
        else:
            pass
        pass
    if keys & ugame.K_DOWN != 0:
        if select_box1.y != (constants.SCREEN_Y * 3 / 4 -
                             constants.SPRITE_SIZE):
```

(continues on next page)

(continued from previous page)

```
            select_box1.move(int(select_box1.x),
                             int(constants.SCREEN_Y * 3 / 4 -
                                 constants.SPRITE_SIZE))
        else:
            pass
        pass
    if keys & ugame.K_LEFT != 0:
        if select_box1.x != constants.SCREEN_X / 4:
            select_box1.move(int(constants.SCREEN_X / 4),
                             int(select_box1.y))
        else:
            pass
        pass
    if keys & ugame.K_RIGHT != 0:
        if select_box1.x != (constants.SCREEN_X * 3 / 4 -
                             constants.SPRITE_SIZE):
            select_box1.move(int(constants.SCREEN_X * 3 / 4 -
                                 constants.SPRITE_SIZE),
                             int(select_box1.y))
        else:
            pass
        pass
```

There is full codes of selection scene. => selection_scene.py <=

In the codes, I send airplane information to the game.

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

Fig. 3: Plane Selection

Now, you can choose plane on your PyBadge.

4.3 Show Airplane

sprites can be placed in front of the background. Create a list called sprites and put the ninth image in the image bank.

```
sprites = []

plane = stage.Sprite(image_bank_1, 8, 72, 56)
sprites.append(plane)
```

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

This code will not work. There is a lot more to this code. Below is the full version of the code above. I imported stage and ugame. Also I add the list of sprites in layers and in the game rendering of while loop function.

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

# Created by : Jay Lee
# Created on : Jan 2020
# This program display sprite

import ugame
```

(continues on next page)

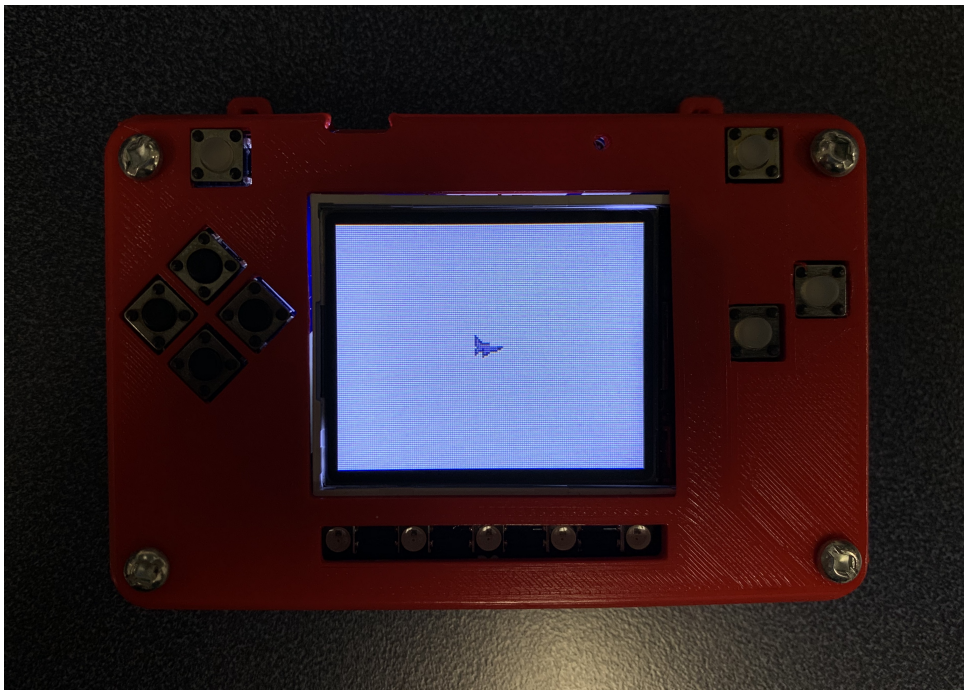


Fig. 4: Plane on PyBadg

(continued from previous page)

```
import stage


def main():
    # this function display background of pybadge
    image_bank_1 = stage.Bank.from_bmp16("avoid_or_shoot.bmp")

    background = stage.Grid(image_bank_1, 10, 8)

    sprites = []

    plane = stage.Sprite(image_bank_1, 8, 72, 56)
    sprites.append(plane)

    # 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
    game.layers = sprites + [background]
    # render the background and inital location of sprite list
    # most likely you will only render background once per scene
    game.render_block()

    # repeat forever, game loop
    while True:
        game.render_sprites(sprites)
        game.tick()


```

(continues on next page)

(continued from previous page)

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

Now, you can get a sprite to show up in front of your background on your PyBadge.

4.4 Move Airplane

sprites can move on the screen. you make some if statements in the while loop. It makes the plane to move. In pybadge, the y value increases as it go down, so you need to subtract the y value from the Up button.

```
While True:
    keys = ugame.buttons.get_pressed()

    if keys & ugame.K_RIGHT:
        plane.move(plane.x + 1, plane.y)
        pass
    if keys & ugame.K_LEFT:
        plane.move(plane.x - 1, plane.y)
        pass
    if keys & ugame.K_UP:
        plane.move(plane.x, plane.y - 1)
        pass
    if keys & ugame.K_DOWN:
        plane.move(plane.x, plane.y + 1)
        pass

    game.render_sprites(sprites)
    game.tick()
```

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

Fig. 5: Moving sprite on background

Now, you can move a plane on your PyBadge.

4.5 Enemy

The game requires the appearance of enemies and the loading missiles.

```
def show_flying():
    enemy_picked = random.randint(0, 1)
    if enemy_picked == 0:
        if bird.y < 0:
            bird.move(200, random.randint(0 + constants.SPRITE_SIZE,
                                          constants.SCREEN_Y -
                                          constants.SPRITE_SIZE))
    else:
        if enemy.y < 0:
            enemy.move(200, random.randint(0 + constants.SPRITE_SIZE,
                                           constants.SCREEN_Y -
                                           constants.SPRITE_SIZE))
```

Create a function to show enemy. This function represents randomly set y values for the enemy. Also create a function that represents a missile like this.

```
if bird.y > 0:
    bird.move(bird.x - 2, bird.y)
    if bird.x < constants.OFF_SCREEN_X:
        bird.move(constants.OFF_SCREEN_X, constants.OFF_SCREEN_Y)
        show_flying()
elif enemy.y > 0:
    enemy.move(enemy.x - 2, enemy.y)
    if enemy.x < constants.OFF_SCREEN_X:
        enemy.move(constants.OFF_SCREEN_X, constants.OFF_SCREEN_Y)
        show_flying()
```

Make this code is while loop. Then you can see the enemy moving. Also make the code that moves a missile like that.

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

Fig. 6: Moving enemy on Pybadge

4.6 Shoot the Missile

Create multiple missiles and put them in the list because missiles can appear on single screen.

```
missiles = []

for missile_number in range(constants.TOTAL_NUMBER_OF_MISSILES):
    missile = stage.Sprite(image_bank_3, 12, constants.OFF_SCREEN_X,
                           constants.OFF_SCREEN_Y)
    missiles.append(missile)
```

So I used for loop to create multiple missiles.

```
if keys & ugame.K_X != 0:
    if a_button == constants.button_state["button_up"]:
        a_button = constants.button_state["button_just_pressed"]
    elif a_button == constants.button_state["button_just_pressed"]:
        a_button = constants.button_state["button_still_pressed"]
else:
    if a_button == constants.button_state["button_still_pressed"]:
        a_button = constants.button_state["button_released"]
    else:
        a_button = constants.button_state["button_up"]
```

The above distinguishes when pressed, when still pressed, and when not pressed.

```
if a_button == constants.button_state["button_just_pressed"]:
    if number_of_missiles > 0:
        for missile_number in range(len(missiles)):
            if missiles[missile_number].y < 0:
                missiles[missile_number].move(plane.x, plane.y)
```

When the A button is pressed, the missile move to plane location.

```
for missile_number in range(len(missiles)):
    if missiles[missile_number].y > 0:
        missiles[missile_number].move(missiles[missile_number].x +
                                      constants.MISSLE_SPEED,
                                      missiles[missile_number].y)
        if missiles[missile_number].x > constants.SCREEN_X:
            missiles[missile_number].move(constants.OFF_SCREEN_X,
                                          constants.OFF_SCREEN_Y)
```

Then, missile move from plane to right side. And if the missile go off the screen, it moves to the location where I created it.

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

Fig. 7: Shooting missile

Now, you can shoot the missiles on your PyBadge.

CHAPTER 5

Menu System

This section describes things except gameplay. Go through the post below.

5.1 Splash Scene

Avoid or Shoot needs a background. This code is puts the first image in the background. The first image is 16 x 16 px image at the top. in this case the first image of image bank is white colour image. Of course when you save the file, save it as `code.py` file:

We use another image bank.

Fig. 1: MT Studio Image Bank

```
image_bank_2 = stage.Bank.from_bmp16("mt_game_studio.bmp")

# sets the background to image 0 in the bank
background = stage.Grid(image_bank_2, 10, 8)

background.tile(2, 2, 0)  # blank white
background.tile(3, 2, 1)
background.tile(4, 2, 2)
background.tile(5, 2, 3)
background.tile(6, 2, 4)
background.tile(7, 2, 0)  # blank white

background.tile(2, 3, 0)  # blank white
background.tile(3, 3, 5)
background.tile(4, 3, 6)
background.tile(5, 3, 7)
background.tile(6, 3, 8)
background.tile(7, 3, 0)  # blank white

```

(continues on next page)

(continued from previous page)

```
background.tile(2, 4, 0)  # blank white
background.tile(3, 4, 9)
background.tile(4, 4, 10)
background.tile(5, 4, 11)
background.tile(6, 4, 12)
background.tile(7, 4, 0)  # blank white

background.tile(2, 5, 0)  # blank white
background.tile(3, 5, 0)
background.tile(4, 5, 13)
background.tile(5, 5, 14)
background.tile(6, 5, 0)
background.tile(7, 5, 0)  # blank white

text = []

text1 = stage.Text(width=29, height=14, font=None,
                   palette=constants.MT_GAME_STUDIO_PALETTE, buffer=None)
text1.move(20, 10)
text1.text("MT Game Studios")
text.append(text1)
```

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

Fig. 2: MT Studio Splash on PyBadg

```
import time

while True:
    time.sleep(1.0)
    main_menu_scene()
```

That makes to switch scene to main menu scene after 1 second with timer.

5.2 Start Scene

Avoid or Shoot needs a logo. So I make th logo of this game. This image is the same image shown on the left.

We use another image bank.

Fig. 3: Game Title Image Bank

```
image_bank_3 = stage.Bank.from_bmp16("avoid_title.bmp")

# sets the background to image 0 in the bank
background = stage.Grid(image_bank_3, 10, 8)

background.tile(2, 2, 0)
background.tile(3, 2, 1)
background.tile(4, 2, 2)
background.tile(5, 2, 3)
background.tile(6, 2, 0)

background.tile(2, 3, 0)
background.tile(3, 3, 4)
background.tile(4, 3, 5)
background.tile(5, 3, 6)
background.tile(6, 3, 0)

background.tile(2, 4, 0)
background.tile(3, 4, 7)
background.tile(4, 4, 8)
background.tile(5, 4, 9)
background.tile(6, 4, 0)

text = []

text1 = stage.Text(width=29, height=14, font=None,
                   palette=constants.MT_GAME_STUDIO_PALETTE, buffer=None)
text1.move(35, 100)
text1.text("START:START")
text.append(text1)

text2 = stage.Text(width=29, height=14, font=None,
                   palette=constants.MT_GAME_STUDIO_PALETTE, buffer=None)
text2.move(35, 110)
text2.text("HELP:SELECT")
text.append(text2)
```

This codes is used to create a large image by concatenatiing the images in the image bank one by one. It was previously covered in Splash Scene.

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

Also we need to go to other scene with buttons.

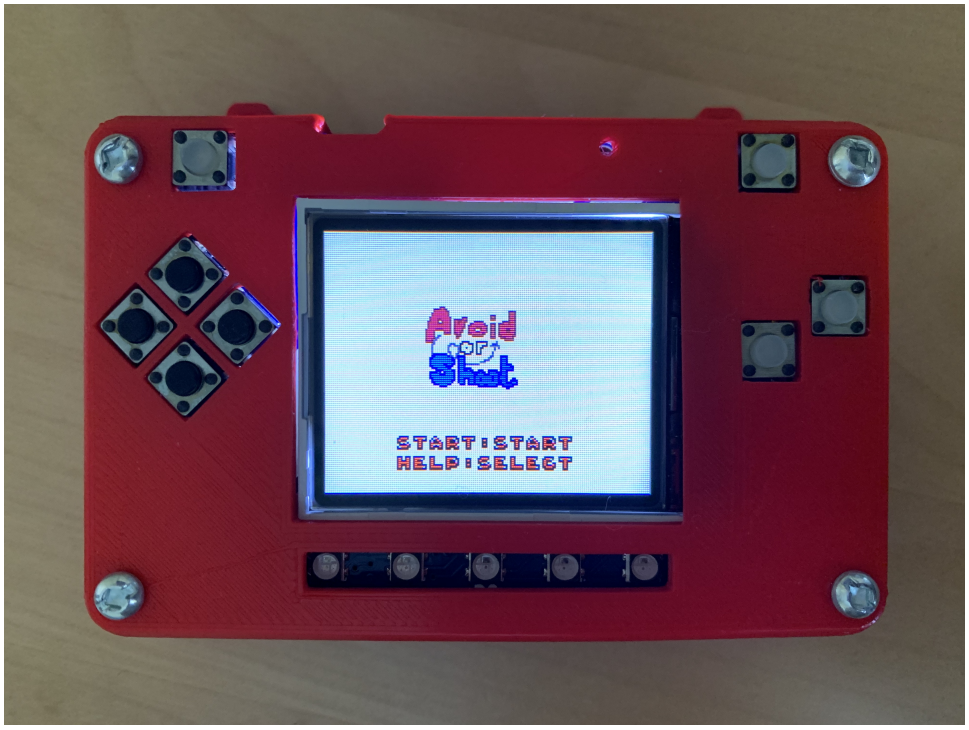


Fig. 4: Game Title on PyBadg

```
while True:
    keys = ugame.buttons.get_pressed()

    if keys & ugame.K_START != 0:
        selection_scene()
    elif keys & ugame.K_SELECT != 0:
        help_scene()

    game.tick()
```

That makes to switch scenes to selection and help scene with start and select button.

5.3 Help Scene

Add some information about this game. Add text with this help_text.py. Then create a box to control the sound.

```
b_button = constants.button_state["button_up"]
start_button = constants.button_state["button_up"]
select_button = constants.button_state["button_up"]

box = []

check_box = stage.Sprite(image_bank_3, 4, 130, 60)
box.append(check_box)

border = []
```

(continues on next page)

(continued from previous page)

```

box_border = stage.Sprite(image_bank_3, 13, 130, 60)
border.append(box_border)
```

The above code will display a sound control box on the screen.

```
while True:
    keys = ugame.buttons.get_pressed()

    if keys & ugame.K_O != 0:
        if b_button == constants.button_state["button_up"]:
            b_button = constants.button_state["button_just_pressed"]
        elif b_button == constants.button_state["button_just_pressed"]:
            b_button = constants.button_state["button_still_pressed"]
    else:
        if b_button == constants.button_state["button_still_pressed"]:
            b_button = constants.button_state["button_released"]
        else:
            b_button = constants.button_state["button_up"]

    if b_button == constants.button_state["button_just_pressed"]:
        if keys & ugame.K_O != 0:
            volume += 1
    if volume % 2 == 1:
        check_box.move(constants.OFF_SCREEN_X, constants.OFF_SCREEN_Y)
    else:
        check_box.move(130, 60)
```

I made it only work when the button was pressed and the variable called volume is added by 1 and divided into odd and even numbers. So if the volume is odd number, it is mute.

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

Fig. 5: Mute Button

Now, you can mute the sound to press B button.

5.4 Game Over Scene

When the game is over, the screen switches to here. This scene shows some information.

```
def game_over_scene(final_score, cause):
```

You get the score of game and cause from the game.

```
text2 = stage.Text(width=29, height=14, font=None,
                   palette=constants.MT_GAME_STUDIO_PALETTE, buffer=None)
text2.move(50, 50)
text2.text("Score:{0}".format(final_score))
text.append(text2)

if cause == 0:
    text3 = stage.Text(width=29, height=14, font=None,
```

(continues on next page)

(continued from previous page)

```
                        palette=constants.MT_GAME_STUDIO_PALETTE,
                        buffer=None)
    text3.move(12, 80)
    text3.text("YOU HIT THE BIRD!")
    text.append(text3)
else:
    text3 = stage.Text(width=29, height=14, font=None,
                        palette=constants.MT_GAME_STUDIO_PALETTE,
                        buffer=None)
    text3.move(10, 80)
    text3.text("YOU HIT THE PLANE!")
    text.append(text3)
```

The score put on the screen. And use if statement to indicate the cause. game_over.py This is the full codes of game over scene.

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

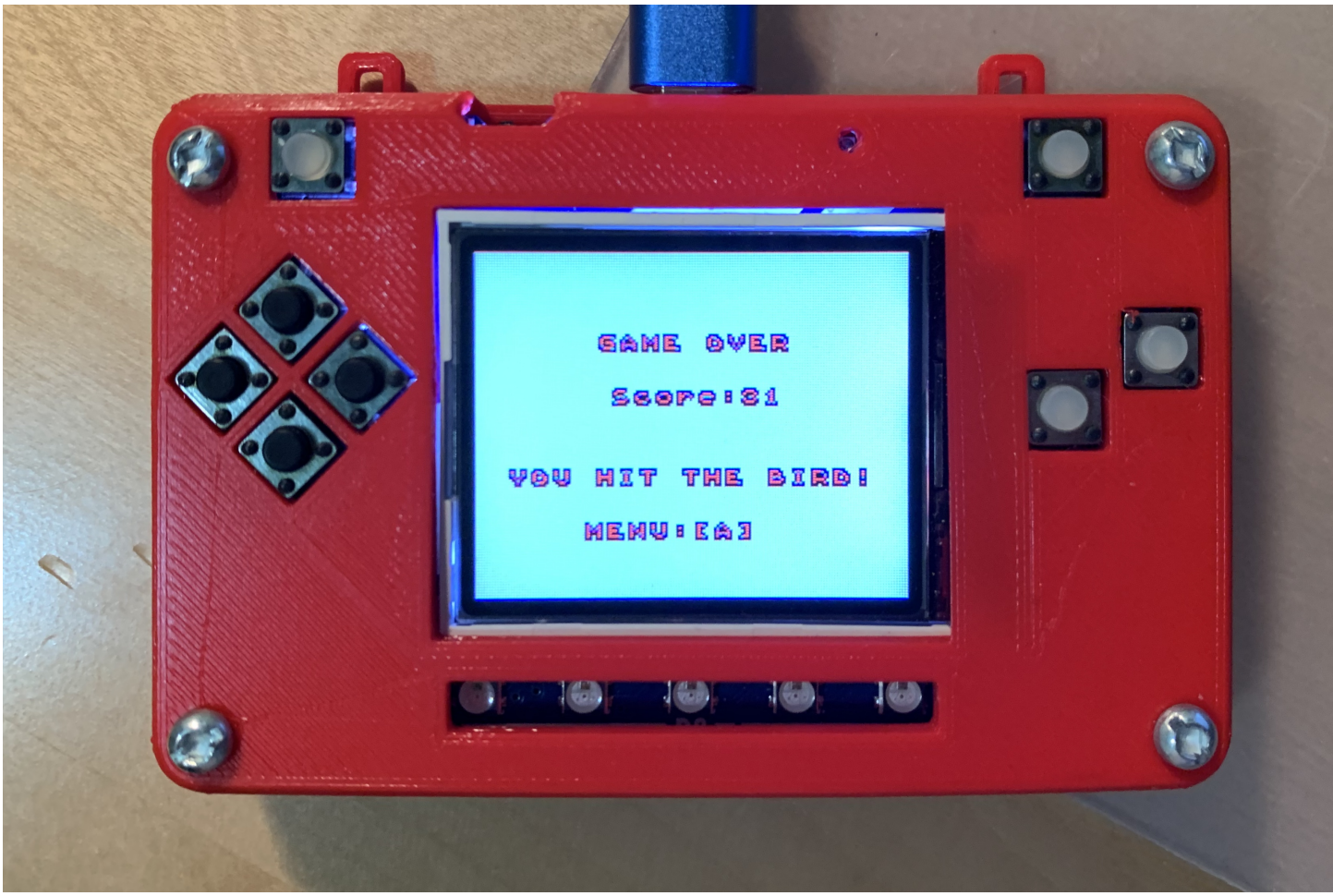


Fig. 6: Game Over Scene