Space Aliens - CircuitPython Game

Mr. Coxall

Jan 23, 2020

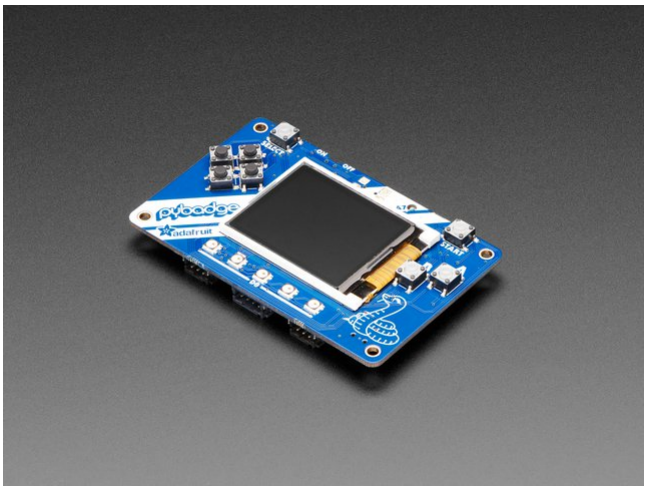
Contents

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 Frogger 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.

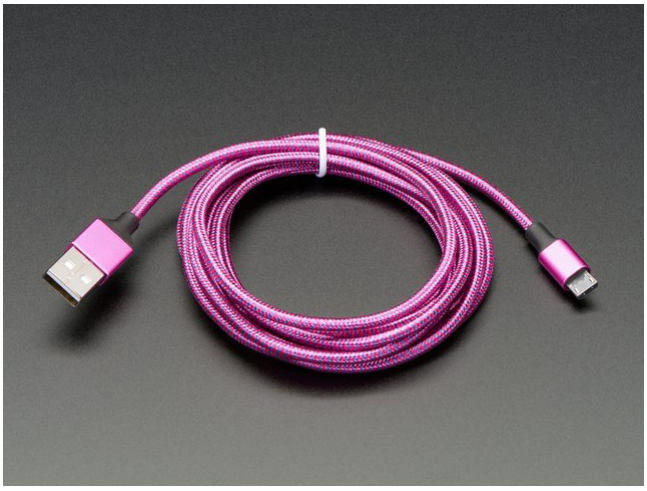
Parts

You will need the following items:



Adafruit PyBadge for MakeCode Arcade, CircuitPython or Arduino

PRODUCT ID: 4200

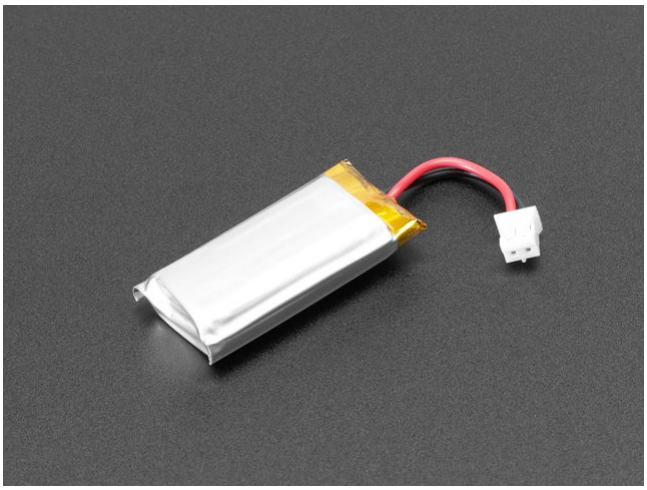


Pink and Purple Braided USB A to Micro B Cable - 2 meter long

PRODUCT ID: 4148

So you can move your CircuitPython code onto the PyBadge.

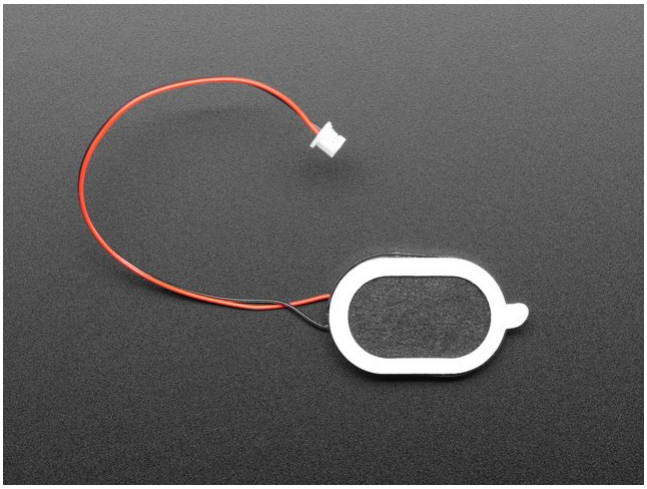
You might also want:



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

PRODUCT ID: 3898

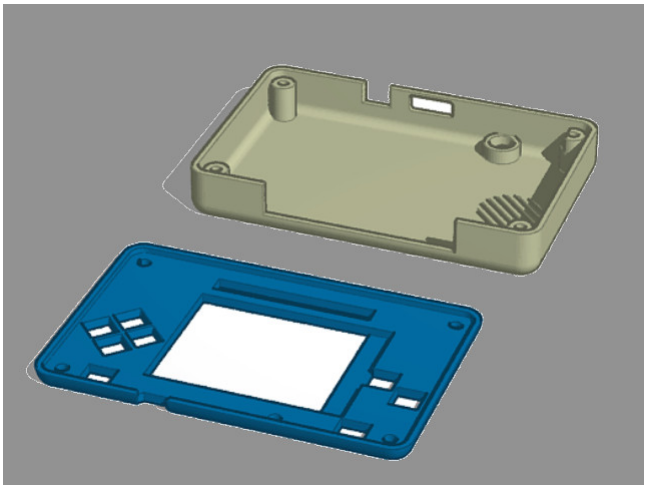
So that you can play the game without having it attached to a computer with a USB cable.



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.

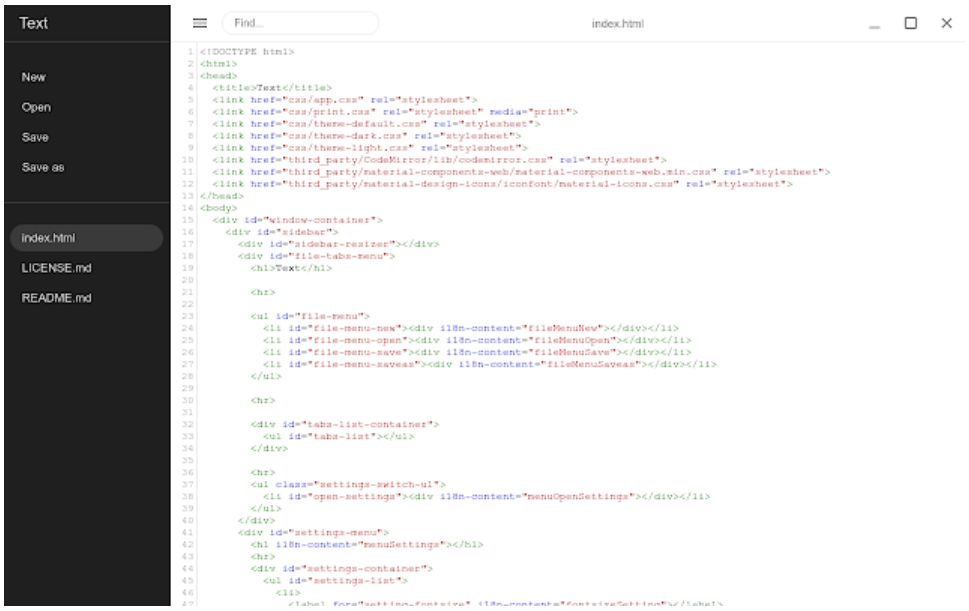


Fig. 1: Chromebook Text app

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 Clown Town

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:

- clown town image bank
- coin sound
- horn sound
- splat 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

This is where we started to make the actual game itself we first started by creating our background and placing our sprites, where we than procceded to make our sprite move accross the screen, as well as setting a border for it to not to escape the screen. We than procceded to spawn objects and make them fall down from the top of the screen. We later than used for loops and if statements to make objects respawn randomly at the top of the screen, and finally we added a scoring system that increases by one every time you complete a wave.

```
  def game_scene():
    # this function is the game scene
    score = 0

    text = []

    score_text = stage.Text(width=29, height=14, font=None, palette=constants.SCORE_
↪PALETTE, buffer=None)
    score_text.clear()
    score_text.cursor(0, 0)
    score_text.move(1, 1)
    score_text.text("Score: {0}".format(score))
    text.append(score_text)

    def show_tomato():
        # make an tomato show up on screen on the x-axis
        for tomato_number in range(len(tomatos)):
            if tomatos[tomato_number].x < 0:
                tomatos[tomato_number].move(random.randint
                                            (0 + constants.SPRITE_SIZE,
                                             constants.SCREEN_X -
                                             constants.SPRITE_SIZE),
                                            constants.OFF_TOP_SCREEN)
                break

    def show_pie():
        for pie_number in range(len(pies)):
            if pies[pie_number].x < 0:
```

(continues on next page)

(continued from previous page)

```
                pies[pie_number].move(random.randint
                                      (0 + constants.SPRITE_SIZE,
                                       constants.SCREEN_X -
                                       constants.SPRITE_SIZE),
                                      constants.OFF_TOP_SCREEN)
                break

    def show_balloon():
        for balloon_number in range(len(balloons)):
            if balloons[balloon_number].x < 0:
                balloons[balloon_number].move(random.randint
                                      (0 + constants.SPRITE_SIZE,
                                       constants.SCREEN_X -
                                       constants.SPRITE_SIZE),
                                      constants.OFF_TOP_SCREEN)
                break

    # an image bank for CircuitPython
    image_bank_2 = stage.Bank.from_bmp16("sprites.bmp")

    splat_sound = open("splat.wav", "rb")
    sound = ugame.audio
    sound.stop()
    sound.mute(False)

    tomatos = []
    pies = []
    balloons = []

    # drops tomatos
    for tomato_number in range(constants.TOTAL_NUMBER_OF_TOMATOS):
        a_single_tomato = stage.Sprite(image_bank_2, 3,
                                       constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
        tomatos.append(a_single_tomato)

    show_tomato()

    # drops pie
    for pie_number in range(constants.TOTAL_NUMBER_OF_PIES):
        a_single_pie = stage.Sprite(image_bank_2, 4,
                                    constants.OFF_SCREEN_X,
                                    constants.OFF_SCREEN_Y)
        pies.append(a_single_pie)

    show_pie()


    # drops balloon
    for balloon_number in range(constants.TOTAL_NUMBER_OF_BALLOONS):
        a_single_balloon = stage.Sprite(image_bank_2, 5,
                                        constants.OFF_SCREEN_X,
                                        constants.OFF_SCREEN_Y)
        balloons.append(a_single_balloon)

    show_balloon()

```

(continues on next page)

(continued from previous page)

```
    # sets the background to image 0 in the bank
    background = stage.Grid(image_bank_2, constants.SCREEN_GRID_X, constants.SCREEN_
↪GRID_Y)

    clown = stage.Sprite(image_bank_2, 2, 74, 56)
    sprites.insert(0, clown)  # insert at the top of sprite list

    # create a stage for the background to show up
    # setting the frame rate to 60fps
    game = stage.Stage(ugame.display, 60)
    # setting the layers to show them in order
    game.layers = text + sprites + pies + tomatos + balloons + [background]
    # rendering the background and the locations of the sprites
    game.render_block()

    # repeat forever game loop
    while True:
        # get user input
        keys = ugame.buttons.get_pressed()

        if keys & ugame.K_RIGHT != 0:
            if clown.x > constants.SCREEN_X - constants.SPRITE_SIZE:
                clown.move(constants.SCREEN_X - constants.SPRITE_SIZE, clown.y)
            else:
                clown.move(clown.x + constants.CLOWN_SPEED, clown.y)
        if keys & ugame.K_LEFT != 0:
            if clown.x < 0:
                clown.move(0, clown.y)
            else:
                clown.move(clown.x - constants.CLOWN_SPEED, clown.y)
        if keys & ugame.K_UP != 0:
            if clown.y < 0:
                clown.move(clown.x, 0)
            else:
                clown.move(clown.x, clown.y - constants.CLOWN_SPEED)
        if keys & ugame.K_DOWN != 0:
            if clown.y > constants.SCREEN_Y - constants.SPRITE_SIZE:
                clown.move(clown.x, constants.SCREEN_Y - constants.SPRITE_SIZE)
            else:
                clown.move(clown.x, clown.y + constants.CLOWN_SPEED)

        # resets tomatos and adds score
        for tomato_number in range(len(tomatos)):
            if tomatos[tomato_number].x > 0:
                tomatos[tomato_number].move(tomatos[tomato_number].x,
                                            tomatos[tomato_number].y +
                                            constants.TOMATO_SPEED)
                if tomatos[tomato_number].y > constants.SCREEN_Y:
                    tomatos[tomato_number].move(constants.OFF_SCREEN_X,
                                                constants.OFF_SCREEN_Y)
                    score += 1
                    score_text.clear()
                    score_text.cursor(0, 0)
                    score_text.move(1, 1)
                    score_text.text("Score: {0}".format(score))
                    game.render_block()
                    show_tomato()
```

(continues on next page)

(continued from previous page)

```
        # resets pies
        for pie_number in range(len(pies)):
            if pies[pie_number].x > 0:
                pies[pie_number].move(pies[pie_number].x,
                                      pies[pie_number].y +
                                      constants.PIE_SPEED)
                if pies[pie_number].y > constants.SCREEN_Y:
                    pies[pie_number].move(constants.OFF_SCREEN_X,
                                          constants.OFF_SCREEN_Y)
                    show_pie()

        # resets balloons
        for balloon_number in range(len(balloons)):
            if balloons[balloon_number].x > 0:
                balloons[balloon_number].move(balloons[balloon_number].x,
                                      balloons[balloon_number].y +
                                      constants.BALLOON_SPEED)
                if balloons[balloon_number].y > constants.SCREEN_Y:
                    balloons[balloon_number].move(constants.OFF_SCREEN_X,
                                          constants.OFF_SCREEN_Y)
                    show_balloon()

        # collision with tomato
        for tomato_number in range(len(tomatos)):
            if tomatos[tomato_number].x > 0:
                if stage.collide(tomatos[tomato_number].x + 1,
                                 tomatos[tomato_number].y,
                                 tomatos[tomato_number].x + 15,
                                 tomatos[tomato_number].y + 15,
                                 clown.x, clown.y, clown.x + 15, clown.y + 15):
                    sound.stop()
                    sound.play(splat_sound)
                    time.sleep(2.0)
                    sound.stop()
                    sprites.remove(clown)
                    game_over_scene(score)

        # collision with pie
        for pie_number in range(len(pies)):
            if pies[pie_number].x > 0:
                if stage.collide(pies[pie_number].x + 1,
                                 pies[pie_number].y,
                                 pies[pie_number].x + 15,
                                 pies[pie_number].y + 15,
                                 clown.x, clown.y, clown.x + 15, clown.y + 15):
                    sound.stop()
                    sound.play(splat_sound)
                    time.sleep(2.0)
                    sound.stop()
                    sprites.remove(clown)
                    game_over_scene(score)

        # collision with balloon
        for balloon_number in range(len(balloons)):
            if balloons[balloon_number].x > 0:
                if stage.collide(balloons[balloon_number].x + 1,
```

(continues on next page)

(continued from previous page)

```
                                balloons[balloon_number].y,
                                balloons[balloon_number].x + 15,
                                balloons[balloon_number].y + 15,
                                clown.x, clown.y, clown.x + 15, clown.y + 15):
                    sound.stop()
                    sound.play(splat_sound)
                    time.sleep(2.0)
                    sound.stop()
                    sprites.remove(clown)
                    game_over_scene(score)

        # update game logic

        # redraw sprite list
        game.render_sprites(sprites + pies + tomatos + balloons)
        game.tick()  # wait until refresh rate finishes
```

4.1 Background

For our background we used a simple statement to fill our background with the first box of our sprites. Our background is a simple circus themed red and black stripes running down the screen. We used it for our splash, game, and game over screen.

```
 background = stage.Grid(image_bank_2, constants.SCREEN_GRID_X, constants.SCREEN_
↪GRID_Y)
```

4.2 Clown

For our main character we created a simple 16x16 clown sprite which we than just placed at the center of the screen when the user presses start from the menu screen. We spawned our sprite using a simple statement and placing its coordinates.

```
# an image bank for CircuitPython
image_bank_2 = stage.Bank.from_bmp16("sprites.bmp")

clown = stage.Sprite(image_bank_2, 2, 74, 56)
sprites.insert(0, clown)  # insert at the top of sprite list
```

CHAPTER 5

Menu System

In this game we have 3 scenes which consist of the the Start Scene, Splash Scene, and Game Over Scene.

5.1 Start Scene

After the splash scenes we transison to the menu scene where we see a clown along with its title and a text saying "PRESS START".

```
  def main_menu_scene():
    # this function is the MT splash scene

    # an image bank for CircuitPython
    image_bank_2 = stage.Bank.from_bmp16("clown.bmp")
    image_bank_3 = stage.Bank.from_bmp16("sprites.bmp")

    # sets the background to image 0 in the bank
    background = stage.Grid(image_bank_3, constants.SCREEN_GRID_X, constants.SCREEN_
↪GRID_Y)

    text = []

    text1 = stage.Text(width=29, height=15, font=None, palette=constants.MT_GAME_
↪STUDIO_PALETTE, buffer=None)
    text1.move(40, 10)
    text1.text("Clown Town")
    text.append(text1)

    clown1 = stage.Sprite(image_bank_2, 1, 70, 56)
    sprites.append(clown1)

    clown2 = stage.Sprite(image_bank_2, 2, 70, 72)
    sprites.append(clown2)

```

(continues on next page)

(continued from previous page)

```
    clown3 = stage.Sprite(image_bank_2, 3, 54, 56)
    sprites.append(clown3)

    clown4 = stage.Sprite(image_bank_2, 4, 86, 56)
    sprites.append(clown4)

    clown5 = stage.Sprite(image_bank_2, 5, 54, 72)
    sprites.append(clown5)

    clown6 = stage.Sprite(image_bank_2, 6, 86, 72)
    sprites.append(clown6)

    clown7 = stage.Sprite(image_bank_2, 8, 70, 40)
    sprites.append(clown7)

    clown8 = stage.Sprite(image_bank_2, 0, 54, 40)
    sprites.append(clown8)

    clown9 = stage.Sprite(image_bank_2, 7, 86, 40)
    sprites.append(clown9)

    text2 = stage.Text(width=29, height=14, font=None, palette=constants.MT_GAME_
↪STUDIO_PALETTE, buffer=None)
    text2.move(35, 110)
    text2.text("PRESS START")
    text.append(text2)

    horn_sound = open("horn.wav", 'rb')
    sound = ugame.audio
    sound.stop()
    sound.mute(False)
    sound.play(horn_sound)

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

    # removes menu clown
    sprites.remove(clown1)
    sprites.remove(clown2)
    sprites.remove(clown3)
    sprites.remove(clown4)
    sprites.remove(clown5)
    sprites.remove(clown6)
    sprites.remove(clown7)
    sprites.remove(clown8)
    sprites.remove(clown9)

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

```

(continues on next page)

(continued from previous page)

```
        # update game logic

        # Wait for 3 seconds
        keys = ugame.buttons.get_pressed()

        if keys & ugame.K_START != 0:  # Start button
            game_scene()

        # redraw sprite list
```

5.2 Splash Scene

In the splash scene we start with a plain white splsh scene and transion to thethe MT Game Studio splash scene after 0.5 seconds which transciosns to the TJ Game splash scene after 3 seconds.

```
def blank_white_reset_scene():
    # this function is the splash scene game loop

    # do house keeping to ensure everythng is setup

    # set up the NeoPixels
    pixels = neopixel.NeoPixel(board.NEOPIXEL, 5, auto_write=False)
    pixels.deinit() # and turn them all off

    # reset sound to be off
    sound = ugame.audio
    sound.stop()
    sound.mute(False)

    # an image bank for CircuitPython
    image_bank_1 = stage.Bank.from_bmp16("mt_game_studio.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
```

(continues on next page)

(continued from previous page)

```
    game = stage.Stage(ugame.display, 60)
    # set the layers, items show up in order
    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()

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

        # update game logic

        # Wait for 1/2 seconds
        time.sleep(0.5)
        mt_splash_scene()

        # redraw sprite list
```

```
def mt_splash_scene():
    # this function is the MT splash scene

    # an image bank for CircuitPython
    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, constants.SCREEN_GRID_X, constants.SCREEN_
↪GRID_Y)

    # used this program to split the iamge into tile: https://ezgif.com/sprite-
↪cutter/ezgif-5-818cdbcc3f66.png
    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

    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)
```

(continues on next page)

(continued from previous page)

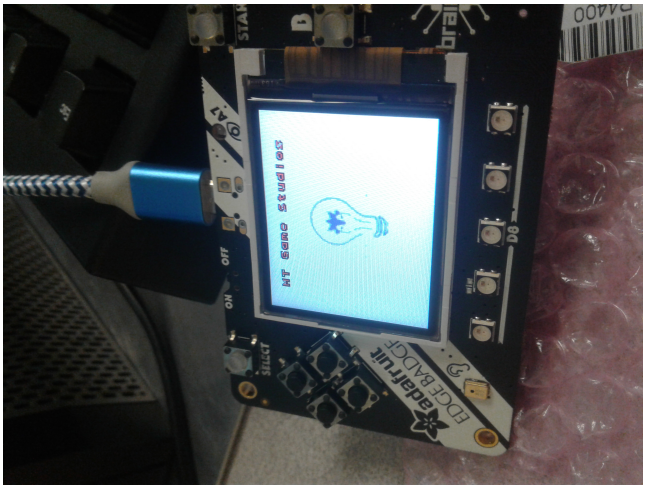
```
    background.tile(6, 5, 0)
    background.tile(7, 5, 0)  # blank white

    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)

    # get sound ready
    # follow this guide to convert your other sounds to something that will work
    #    https://learn.adafruit.com/microcontroller-compatible-audio-file-conversion
    coin_sound = open("coin.wav", 'rb')
    sound = ugame.audio
    sound.stop()
    sound.mute(False)
    sound.play(coin_sound)

    # 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 = text + [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:
        # get user input

        # update game logic

        # Wait for 1 seconds
        time.sleep(3.0)
        game_splash_scene()

        # redraw sprite list
```

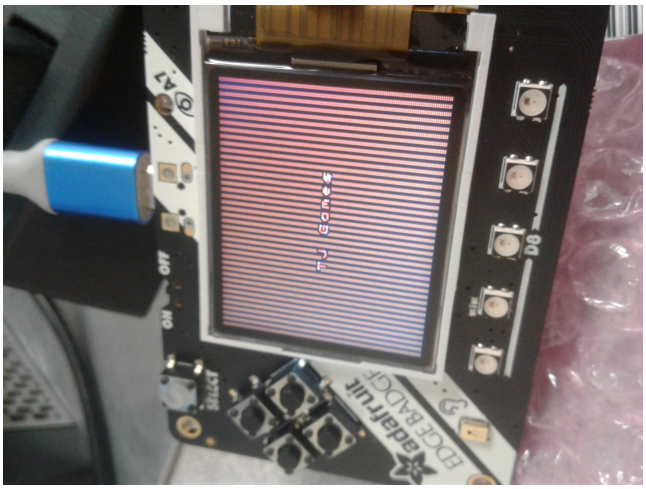
```
def game_splash_scene():
    # this function is the MT splash scene

    # an image bank for CircuitPython
    image_bank_2 = stage.Bank.from_bmp16("sprites.bmp")

    # sets the background to image 0 in the bank
    background = stage.Grid(image_bank_2, constants.SCREEN_GRID_X, constants.SCREEN_
↪GRID_Y)

    text = []

    text1 = stage.Text(width=29, height=15, font=None, palette=constants.MT_GAME_
↪STUDIO_PALETTE, buffer=None)
    text1.move(50, 60)
```

(continues on next page)

(continued from previous page)

```
    text1.text("TJ Games")
    text.append(text1)

    # 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 = text + [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:
        # get user input

        # update game logic

        # Wait for 3 seconds
        time.sleep(3.0)
        main_menu_scene()

        # redraw sprite list
```





5.3 Game Over Scene

If you get hit by any of the objects the game makes a splat sound and takes you to the game over scene, where it gives you you final score and to press start if you wish to play again.

```
  def game_over_scene(final_score):
    # this function is the game over scene

    # an image bank for CircuitPython
    image_bank_2 = stage.Bank.from_bmp16("sprites.bmp")

    # sets the background to image 0 in the bank
    background = stage.Grid(image_bank_2, constants.SCREEN_GRID_X, constants.SCREEN_
↪GRID_Y)

    text = []

    text1 = stage.Text(width=29, height=14, font=None,
                       palette=constants.MT_GAME_STUDIO_PALETTE, buffer=None)
    text1.move(22, 20)
    text1.text("Final Score: {:0>2d}".format(final_score))
    text.append(text1)

    text2 = stage.Text(width=29, height=14, font=None,
                       palette=constants.MT_GAME_STUDIO_PALETTE, buffer=None)
    text2.move(43, 60)
    text2.text("GAME OVER")
    text.append(text2)

    text3 = stage.Text(width=29, height=14, font=None,
                       palette=constants.MT_GAME_STUDIO_PALETTE, buffer=None)
    text3.move(32, 110)
    text3.text("PRESS SELECT")
    text.append(text3)

    # 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)

```
    # set the background layer
    game.layers = sprites + text + [background]
    # render the background
    # most likely you will only render background once per scene
    game.render_block()

    # Game loop
    while True:
        # Update game logic
        keys = ugame.buttons.get_pressed()

        if keys & ugame.K_SELECT != 0:
            keys = 0
            main_menu_scene()
```