Space Aliens - CircuitPython Game

Mr. Coxall

Jan 22, 2020

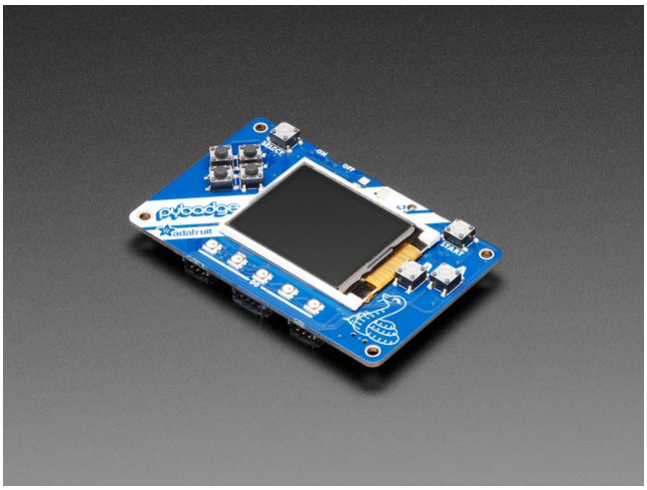
Contents

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 Asteroids 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.

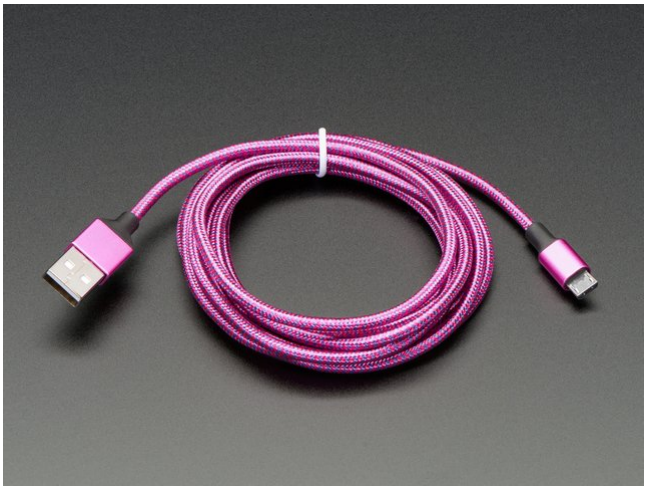
Parts

You will need the following items:



Adafruit PyBadge for MakeCode Arcade, CircuitPython or Arduino

PRODUCT ID: 4200

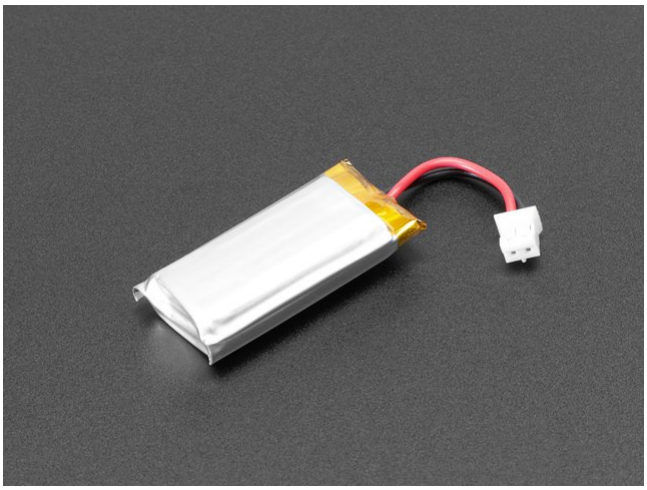


Pink and Purple Braided USB A to Micro B Cable - 2 meter long

PRODUCT ID: 4148

So you can move your CircuitPython code onto the PyBadge.

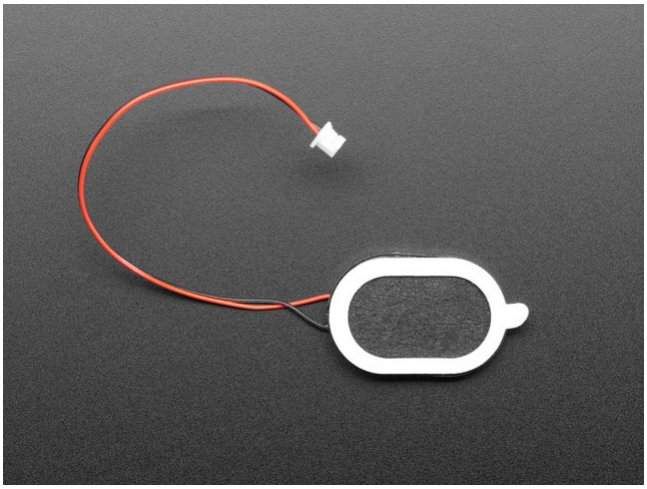
You might also want:



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

PRODUCT ID: 3898

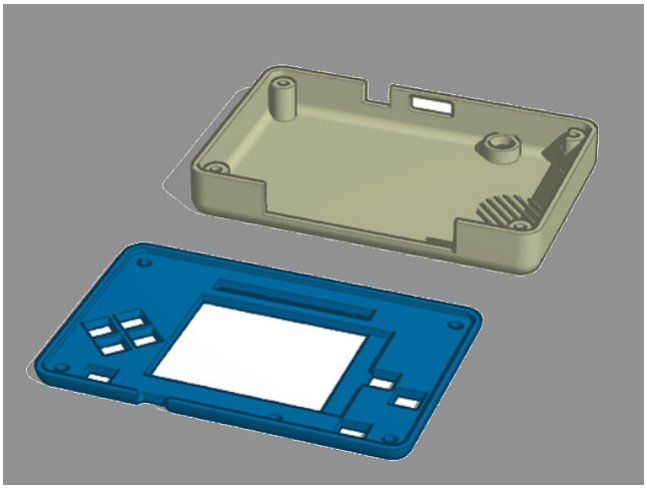
So that you can play the game without having it attached to a computer with a USB cable.



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.

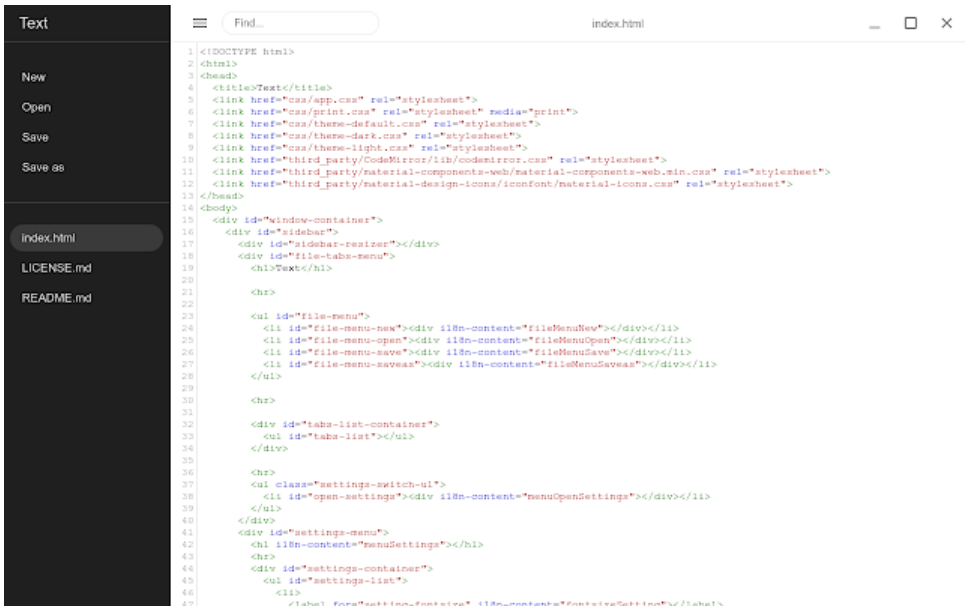


Fig. 1: Chromebook Text app

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 Ship and Lasers for the game

Fig. 2: Image Bank for Asteroids and Enemies for the game

Fig. 3: Image Bank for Background for the game

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:

- ship and lasers image bank
- asteroids and enemies image bank
- background 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

The game scene starts out with the player/ship spawning in the middle of the screen. The player can move around using the d-pad and shoot lasers using the a button. Once the game starts, Asteroids will come down from the top of the screen, while two types of enemies come from the left side of the screen moving right. The aim of the game is to kill/avoid all enemies/asteroids before losing your three lives. On the main menu, there is an option to play on easy or hard mode (a for easy, b for hard(code shown in menu part of this documentation)). Destroying asteroids nets you five points on easy(fifteen points for hard), while enemies net you ten points on easy(thirty points for hard). The game gets progressively harder as you kill more enemies and asteroids. Once all three of your lives are gone, GAME OVER!

Here is the code for the main game:

```
def game_scene(diff_mul):
  # this function is the game scene
  # background image bank ready
  background_bank = stage.Bank.from_bmp16("background.bmp")
  image_bank_0 = stage.Bank.from_bmp16("meteor.bmp")
  image_bank_1 = stage.Bank.from_bmp16("ship-and-lasers.bmp")
  background = stage.Grid(background_bank, constants.SCREEN_GRID_X, constants.SCREEN_
↪GRID_Y)
  for x_location in range(constants.SCREEN_GRID_X):
      for y_location in range(constants.SCREEN_GRID_Y):
          tile_picked = random.randint(0, 15)
          background.tile(x_location, y_location, tile_picked)
  a_button = constants.button_state["button_up"]
  shoot_sound = open("pew.wav", 'rb')
  boom_sound = open("boom.wav", 'rb')
  sound = ugame.audio
  sound.stop()
  sound.mute(False)

  # Buttons that you want to keep state information on
  a_button = constants.button_state["button_up"]
  start_button = constants.button_state["button_up"]
  select_button = constants.button_state["button_up"]

```

(continues on next page)

(continued from previous page)

```
  sprites = []
  ship = stage.Sprite(image_bank_1, 0, 80, 64)
  sprites.insert(0, ship)  # insert at the top of sprite list

  score = 0
  scoretext = []
  score_text = stage.Text(width=29, height=14, font=None,
                          palette=constants.SCORE_PALETTE, buffer=None)
  score_text.cursor(0, 0)
  score_text.move(1, 118)
  score_text.text("Points: {0}".format(score))
  scoretext.append(score_text)

  lives = 3
  livestext = []
  lives_text = stage.Text(width=29, height=14, font=None,
                          palette=constants.LIVES_PALETTE, buffer=None)
  lives_text.cursor(0, 0)
  lives_text.move(1, 1)
  lives_text.text("Lives: {0}".format(lives))
  livestext.append(lives_text)

  asteroids = []
  for asteroids_number in range(constants.TOTAL_ASTEROIDS * diff_mul):
      single_asteroid = stage.Sprite(image_bank_0, 0, constants.OFF_TOP_SCREEN,␣
↪constants.OFF_TOP_SCREEN)
      asteroids.append(single_asteroid)

  enemy_1 = []
  for enemy_number_1 in range(constants.TOTAL_ENEMY_1 * diff_mul):
      single_1 = stage.Sprite(image_bank_0, 1, constants.OFF_TOP_SCREEN, constants.
↪OFF_TOP_SCREEN)
      enemy_1.append(single_1)

  enemy_2 = []
  for enemy_number_2 in range(constants.TOTAL_ENEMY_2 * diff_mul):
      single_2 = stage.Sprite(image_bank_0, 2, constants.OFF_TOP_SCREEN, constants.
↪OFF_TOP_SCREEN)
      enemy_2.append(single_2)

  lasers = []
  for laser_number in range(constants.TOTAL_NUMBER_OF_LASERS):
      single_laser = stage.Sprite(image_bank_1, 8, constants.OFF_TOP_SCREEN,␣
↪constants.OFF_TOP_SCREEN)
      lasers.append(single_laser)

  enemy_count = 1
  show_enemy(asteroids)
  show_enemy_2(enemy_1)
  show_enemy_3(enemy_2)
  death_mul = 1
  # set frame rate to 60fps
  game = stage.Stage(ugame.display, 60)
  # set layers, items show up in order
  game.layers = sprites + enemy_1 + enemy_2 + asteroids + lasers + scoretext +␣
↪livestext + [background]
  # render background and sprite list
```

(continues on next page)

(continued from previous page)

```
    game.render_block()
    # repeat forever, game loop
    while True:
        # get user input
        keys = ugame.buttons.get_pressed()
        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"]

        if a_button == constants.button_state["button_just_pressed"]:
            for laser_number in range(len(lasers)):
                if lasers[laser_number].x < 0:
                    lasers[laser_number] .move(ship.x, ship.y)
                    sound.stop()
                    sound.play(shoot_sound)
                    break
        for laser_number in range(len(lasers)):
                if lasers[laser_number].x > 0:
                    if ship.rotation == 0:
                        lasers[laser_number].set_frame(rotation=0)
                        lasers[laser_number].move(lasers[laser_number].x, lasers[laser_
↪number].y - constants.LASER_SPEED)
                        if lasers[laser_number].y < constants.OFF_SCREEN_Y:
                            lasers[laser_number].move(constants.OFF_SCREEN_X, constants.
↪OFF_SCREEN_Y)
                    elif ship.rotation == 1:
                        lasers[laser_number].set_frame(rotation=1)
                        lasers[laser_number].move(lasers[laser_number].x + constants.
↪LASER_SPEED, lasers[laser_number].y)
                        if lasers[laser_number].x > 160:
                            lasers[laser_number].move(constants.OFF_SCREEN_X, constants.
↪OFF_SCREEN_Y)
                    elif ship.rotation == 2:
                        lasers[laser_number].set_frame(rotation=0)
                        lasers[laser_number].move(lasers[laser_number].x, lasers[laser_
↪number].y + constants.LASER_SPEED)
                        if lasers[laser_number].y > 128:
                            lasers[laser_number].move(constants.OFF_SCREEN_X, constants.
↪OFF_SCREEN_Y)
                    elif ship.rotation == 3:
                        lasers[laser_number].set_frame(rotation=1)
                        lasers[laser_number].move(lasers[laser_number].x - constants.
↪LASER_SPEED, lasers[laser_number].y)
                        if lasers[laser_number].x < 5:
                            lasers[laser_number].move(constants.OFF_SCREEN_X, constants.
↪OFF_SCREEN_Y)
        # Move ship right
        if keys & ugame.K_RIGHT:
            state_of_button = 2
            if ship.x > constants.SCREEN_X - constants.SPRITE_SIZE:
```

(continues on next page)

(continued from previous page)

```
                ship.move(constants.SCREEN_X - constants.SPRITE_SIZE, ship.y)
            else:
                ship.move(ship.x + constants.SHIP_MOVEMENT_SPEED * death_mul * diff_mul,
↪ ship.y)
                ship.set_frame(rotation=1)
            pass

        # Move ship left
        if keys & ugame.K_LEFT:
            state_of_button = 4
            if ship.x < 5:
                ship.move(5, ship.y)
            else:
                ship.move(ship.x - constants.SHIP_MOVEMENT_SPEED * death_mul * diff_mul,
↪ ship.y)
                ship.set_frame(rotation=3)
            pass

        # Move ship up
        if keys & ugame.K_UP:
            state_of_button = 1
            if ship.y < 0:
                ship.move(ship.x, 0)
            else:
                ship.move(ship.x, ship.y - constants.SHIP_MOVEMENT_SPEED * death_mul *␣
↪diff_mul)
                ship.set_frame(rotation=0)
            pass

        # Move ship down
        if keys & ugame.K_DOWN:
            state_of_button = 3
            if ship.y > constants.SCREEN_Y - constants.SPRITE_SIZE:
                ship.move(ship.x, constants.SCREEN_Y - constants.SPRITE_SIZE * death_
↪mul * diff_mul)
            else:
                ship.move(ship.x, ship.y + constants.SHIP_MOVEMENT_SPEED * death_mul *␣
↪diff_mul)
                ship.set_frame(rotation=2)
            pass

        # update game logic
        for asteroid_number in range(len(asteroids)):
            if asteroids[asteroid_number].x > 0:
                asteroids[asteroid_number].move(asteroids[asteroid_number].x,␣
↪asteroids[asteroid_number].y + constants.ENEMY_SPEED  * diff_mul * death_mul)
                if asteroids[asteroid_number].y > constants.SCREEN_Y:
                    asteroids[asteroid_number].move(constants.OFF_SCREEN_X, constants.
↪OFF_SCREEN_Y)
                    show_enemy(asteroids)
        for enemy_number_1 in range(len(enemy_1)):
            if enemy_1[enemy_number_1].y > 0:
                enemy_1[enemy_number_1].move(enemy_1[enemy_number_1].x + constants.
↪ENEMY_SPEED * death_mul * diff_mul, enemy_1[enemy_number_1].y)
                if enemy_1[enemy_number_1].x > constants.SCREEN_X:
                    enemy_1[enemy_number_1].move(constants.OFF_SCREEN_X, constants.OFF_
↪SCREEN_Y)
```

(continues on next page)

(continued from previous page)

```
                    show_enemy_2(enemy_1)
        for enemy_number_2 in range(len(enemy_2)):
            if enemy_2[enemy_number_2].y > 0:
                enemy_2[enemy_number_2].move(enemy_2[enemy_number_2].x + constants.
↪ENEMY_SPEED * death_mul * diff_mul, enemy_2[enemy_number_2].y)
                if enemy_2[enemy_number_2].x > constants.SCREEN_X:
                    enemy_2[enemy_number_2].move(constants.OFF_SCREEN_X, constants.OFF_
↪SCREEN_Y)
                    show_enemy_3(enemy_2)
        for laser_number in range(len(lasers)):
            if lasers[laser_number].x > 0:
                for enemy_number_1 in range(len(enemy_1)):
                    if enemy_1[enemy_number_1].x > 0:
                        if stage.collide(lasers[laser_number].x, lasers[laser_number].y,
                                         lasers[laser_number].x + 16, lasers[laser_
↪number].y + 16,
                                         enemy_1[enemy_number_1].x, enemy_1[enemy_
↪number_1].y,
                                         enemy_1[enemy_number_1].x + 16, enemy_1[enemy_
↪number_1].y + 16):
                            enemy_1[enemy_number_1].move(constants.OFF_SCREEN_X,␣
↪constants.OFF_SCREEN_Y)
                            lasers[laser_number].move(constants.OFF_SCREEN_X, constants.
↪OFF_SCREEN_Y)
                            score += 10*diff_mul
                            score_text.clear()
                            score_text.cursor(0,0)
                            score_text.move(1, 118)
                            score_text.text("Points: {0}".format(score))
                            sound.stop()
                            sound.play(boom_sound)
                            show_enemy_2(enemy_1)
                            show_enemy_2(enemy_1)
                            death_mul += (1/30)
                for enemy_number_2 in range(len(enemy_2)):
                    if enemy_2[enemy_number_2].x > 0:
                        if stage.collide(lasers[laser_number].x, lasers[laser_number].y,
                                         lasers[laser_number].x + 16, lasers[laser_
↪number].y + 16,
                                         enemy_2[enemy_number_2].x, enemy_2[enemy_
↪number_2].y,
                                         enemy_2[enemy_number_2].x + 16, enemy_2[enemy_
↪number_2].y + 16):
                            enemy_2[enemy_number_2].move(constants.OFF_SCREEN_X,␣
↪constants.OFF_SCREEN_Y)
                            lasers[laser_number].move(constants.OFF_SCREEN_X, constants.
↪OFF_SCREEN_Y)
                            score += 10*diff_mul
                            score_text.clear()
                            score_text.cursor(0,0)
                            score_text.move(1, 118)
                            score_text.text("Points: {0}".format(score))
                            sound.stop()
                            sound.play(boom_sound)
                            show_enemy_3(enemy_2)
                            show_enemy_3(enemy_2)
                            death_mul += (1/30)
```

(continues on next page)

(continued from previous page)

```
            for asteroid_number in range(len(asteroids)):
                if asteroids[asteroid_number].x > 0:
                    if stage.collide(lasers[laser_number].x, lasers[laser_number].y,
                                     lasers[laser_number].x + 16, lasers[laser_
↪number].y + 16,
                                     asteroids[asteroid_number].x,␣
↪asteroids[asteroid_number].y,
                                     asteroids[asteroid_number].x + 16,␣
↪asteroids[asteroid_number].y + 16):
                        asteroids[asteroid_number].move(constants.OFF_SCREEN_X,␣
↪constants.OFF_SCREEN_Y)
                        lasers[laser_number].move(constants.OFF_SCREEN_X, constants.
↪OFF_SCREEN_Y)
                        score += 5*diff_mul
                        score_text.clear()
                        score_text.cursor(0,0)
                        score_text.move(1, 118)
                        score_text.text("Points: {0}".format(score))
                        sound.stop()
                        sound.play(boom_sound)
                        show_enemy(asteroids)
                        show_enemy(asteroids)
                        death_mul += (1/30)
        for enemy_number_1 in range(len(enemy_1)):
            if enemy_1[enemy_number_1].x > 0:
                if stage.collide(enemy_1[enemy_number_1].x, enemy_1[enemy_number_1].y,
                                 enemy_1[enemy_number_1].x + 16, enemy_1[enemy_number_
↪1].y + 16,
                                 ship.x, ship.y,
                                 ship.x + 16, ship.y + 16):
                    lives -= 1
                    ship.move(-100, -100)
                    sound.stop()
                    sound.play(boom_sound)
                    time.sleep(1)
                    if lives == 0:
                        game_over_scene(score)
                    else:
                        lives_text.clear()
                        lives_text.cursor(0,0)
                        lives_text.move(1, 1)
                        lives_text.text("Lives: {0}".format(lives))
                        ship.move (random.randint(16, 146), random.randint(16, 106))
        for enemy_number_2 in range(len(enemy_2)):
            if enemy_2[enemy_number_2].x > 0:
                if stage.collide(enemy_2[enemy_number_2].x, enemy_2[enemy_number_2].y,
                                 enemy_2[enemy_number_2].x + 16, enemy_2[enemy_number_
↪2].y + 16,
                                 ship.x, ship.y,
                                 ship.x + 16, ship.y + 16):
                    lives -= 1
                    ship.move(-100, -100)
                    sound.stop()
                    sound.play(boom_sound)
                    time.sleep(1)
                    if lives == 0:
                        game_over_scene(score)
```

(continues on next page)

(continued from previous page)

```
                else:
                    lives_text.clear()
                    lives_text.cursor(0,0)
                    lives_text.move(1, 1)
                    lives_text.text("Lives: {0}".format(lives))
                    ship.move (random.randint(16, 146), random.randint(16, 106))
    for asteroid_number in range(len(asteroids)):
        if asteroids[asteroid_number].x > 0:
            if stage.collide(asteroids[asteroid_number].x, asteroids[asteroid_
↪number].y,
                             asteroids[asteroid_number].x + 16, asteroids[asteroid_
↪number].y + 16,
                             ship.x, ship.y,
                             ship.x + 16, ship.y + 16):
                lives -= 1
                ship.move(-100, -100)
                sound.stop()
                sound.play(boom_sound)
                time.sleep(1)
                if lives == 0:
                    game_over_scene(score)
                else:
                    lives_text.clear()
                    lives_text.cursor(0,0)
                    lives_text.move(1, 1)
                    lives_text.text("Lives: {0}".format(lives))
                    ship.move (random.randint(16, 146), random.randint(16, 106))
    # redraw sprite list
    game.render_sprites(sprites + asteroids + enemy_1 + enemy_2 + lasers)
    game.tick()
```

The full game code is here for full use for anybody wishing to make this game :)

4.1 Background

For the game, the background is space themed, just like the original arcade game(see home page link for example). To make the background show up for this game, we take the background image bank which is already created and take random images from the bank to paste it across the PyBadge screen. First, we set up our background image bank:

```
# background image bank ready
background_bank = stage.Bank.from_bmp16("background.bmp")
```

Then, we create a loop where we take a single image from the background bank and paste it on the screen. This loop occurs until the background fills up the whole PyBadge screen:

```
background = stage.Grid(background_bank, constants.SCREEN_GRID_X, constants.SCREEN_
↪GRID_Y)
for x_location in range(constants.SCREEN_GRID_X):
    for y_location in range(constants.SCREEN_GRID_Y):
        tile_picked = random.randint(0, 15)
        background.tile(x_location, y_location, tile_picked)
```

The background now shows up on the screen!

4.2 Ship/Player

As stated in the main game page, the player controls a ship using the d-pad. When a player presses a direction, the ship rotates to the direction in which the player is pressing(ex. player presses left, ship rotates and moves left). The player shoots lasers by pressing a, and the lasers also go in the direction that the ship is facing in. The controls for the game are very simple, thus a simple and fun game! Here is the code to get the ship showing up in the game:

```
sprites = []
  ship = stage.Sprite(image_bank_1, 0, 80, 64)
  sprites.insert(0, ship)  # insert at the top of sprite list
```

The code for moving the ship is in the main game page.

CHAPTER 5

Menu System

To make the game look more professional, there are three extra scenes in our game: The start scene(main menu), the splash scene, and Game Over Scene.

5.1 Menu Scene

The main menu scene includes the title of the game(Asteroids), the background(which is talked about in this documentation), and an option to play easy mode or hard mard by pressing a or b respectively. Here is the code for the main menu scene:

```
def main_menu_scene():
  # this function is the menu scene
  # this code is only temporary so that I can work on game scene

  # an image bank for CircuitPython
  image_bank_0 = stage.Bank.from_bmp16("background.bmp")
  image_bank_1 = stage.Bank.from_bmp16("meteor.bmp")

  # difficulty multipliers that will be passed over to the game scene
  easy_mode = 1
  hard_mode = 3


  # sets the background
  background = stage.Grid(image_bank_0, constants.SCREEN_GRID_X,
                          constants.SCREEN_GRID_Y)
  for x_location in range(constants.SCREEN_GRID_X):
      for y_location in range(constants.SCREEN_GRID_Y):
          tile_picked = random.randint(0, 15)
          background.tile(x_location, y_location, tile_picked)
  sprites = []
  text = []
  text1 = stage.Text(width=29, height=14, font=None, palette=constants.MT_GAME_STUDIO_
↪PALETTE, buffer=None)
```

(continues on next page)

(continued from previous page)

```
  text1.move(10, 20)
  text1.text("Asteroid Breaker")
  text.append(text1)
  text2 = stage.Text(width=29, height=14, font=None, palette=constants.MT_GAME_STUDIO_
↪PALETTE, buffer=None)
  text2.move(5, 100)
  text2.text("A = Easy  B = Hard")
  text.append(text2)

  title_meteor = stage.Sprite(image_bank_1, 0, 80, 64)
  sprites.append(title_meteor)


  # 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()
  # repeat forever, game loop
  while True:
      keys = ugame.buttons.get_pressed()
      # get user input
      if keys & ugame.K_X != 0:
          game_scene(easy_mode)
      if keys & ugame.K_O != 0:
          game_scene(hard_mode)


      # update game logic
      # redraw sprite list
      pass # just a placeholder until you write the code
```

The main menu scene leads into the main game.

5.2 Splash Scene

The splash scene includes 3 short splash scenes before reaching the main menu scene. the entire game begins with a short white screen which shows up for half a second:

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

  # do house keeping to ensure everythng is setup
  # reset sound to be off
  sound = ugame.audio
  sound.stop()
  sound.mute(False)

  # an image bank for CircuitPython
```

(continues on next page)

(continued from previous page)

```
  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
  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
```

Then, a splash screen which displays MT Studios along with a lightbulb image appears for a second:

```
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)
```

(continues on next page)

(continued from previous page)

```
  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)

  # 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(1.0)
      game_splash_scene()

      # redraw sprite list
```

Finally, a splash screen which displays our "company" name(Rousseau & Watson Corporations) is shown for a second, along with a coin sound:

```
def game_splash_scene():
  # this function is the game 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)

  text = []

  text1 = stage.Text(width=29, height=14, font=None, palette=constants.MT_GAME_STUDIO_
↪PALETTE, buffer=None)
```

(continues on next page)

(continued from previous page)

```
  text1.move(19, 50)
  text1.text("Rousseau & Watson")
  text.append(text1)

  text2 = stage.Text(width=29, height=14, font=None, palette=constants.MT_GAME_STUDIO_
↪PALETTE, buffer=None)
  text2.move(35, 60)
  text2.text("Corporations")
  text.append(text2)

  # 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
      time.sleep(1.0)
      main_menu_scene()
```

After, the game moves to the main menu scene.

5.3 Game Over Scene

The game moves to the game over scene when the player loses all three of their lives. The game over scene displays score that they received when they died, as well as an option to go back to the menu scene by pressing any button. When the player presses a button, the game returns to the main menu scene, where they can 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("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)

  text = []

```

(continues on next page)

(continued from previous page)

```
    text1 = stage.Text(width=29, height=14, font=None,
                       palette=constants.MT_GAME_STUDIO_PALETTE, buffer=None)
    text1.move(30, 66)
    text1.text("Your 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(35, 20)
    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(27, 110)
    text3.text("PRESS ANY BUTTON!")
    text.append(text3)

    text4 = stage.Text(width=29, height=14, font=None,
                       palette=constants.MT_GAME_STUDIO_PALETTE, buffer=None)
    text4.move(27, 120)
    text4.text("TO MAIN MENU")
    text.append(text4)

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

    # repeat forever, game loop
    while True:
        # get user input
        keys = ugame.buttons.get_pressed()
        if keys & ugame.K_SELECT != 0:
            keys = 0
            main_menu_scene()
        elif keys & ugame.K_START != 0:
            keys = 1
            main_menu_scene()
        elif keys & ugame.K_X != 0:
            keys = 2
            main_menu_scene()
        elif keys & ugame.K_O != 0:
            keys = 3
            main_menu_scene()
        elif keys & ugame.K_UP != 0:
            keys = 4
            main_menu_scene()
        elif keys & ugame.K_DOWN != 0:
            keys = 5
            main_menu_scene()
        elif keys & ugame.K_LEFT != 0:
            keys = 6
```

(continues on next page)

(continued from previous page)

```
68              main_menu_scene()
69          elif keys & ugame.K_RIGHT != 0:
70              keys = 7
71              main_menu_scene()
72          # redraw sprite list
73          pass # just a placeholder until you write the code
```