Space Aliens - CircuitPython Game

Mr. Coxall

Jan 22, 2020

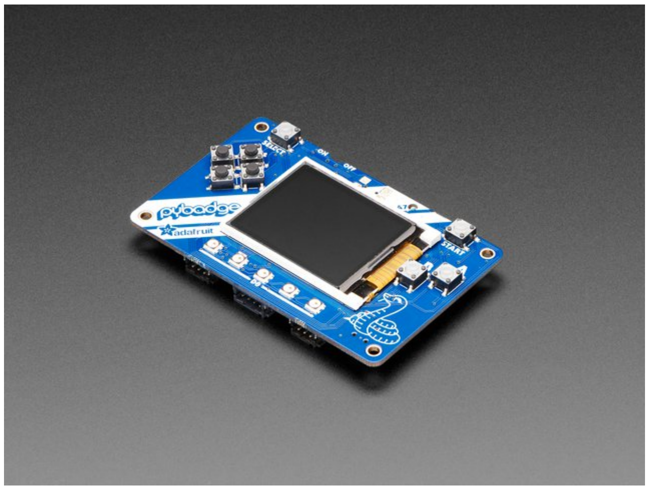
Contents

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 Space Invaders and 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.

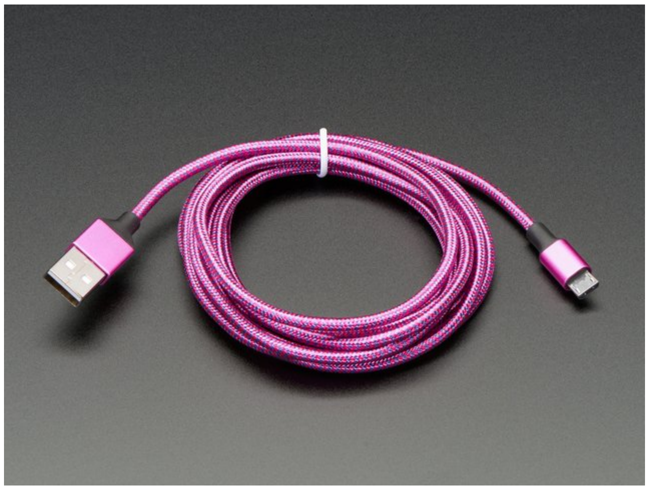
Parts

You will need the following items:



Adafruit PyBadge for MakeCode Arcade, CircuitPython or Arduino

PRODUCT ID: 4200

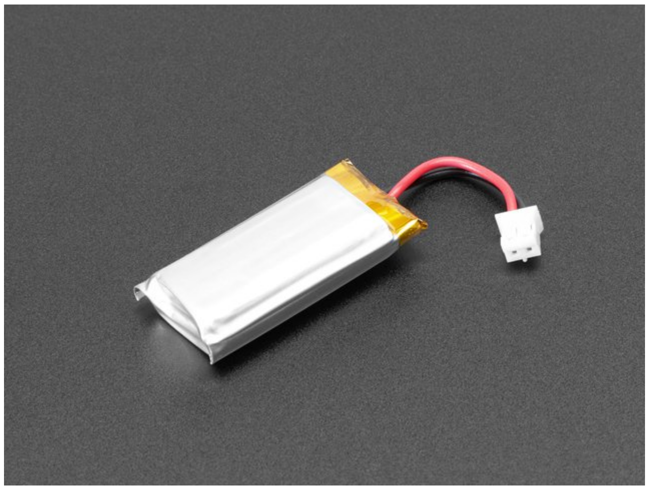


Pink and Purple Braided USB A to Micro B Cable - 2 meter long

PRODUCT ID: 4148

So you can move your CircuitPython code onto the PyBadge.

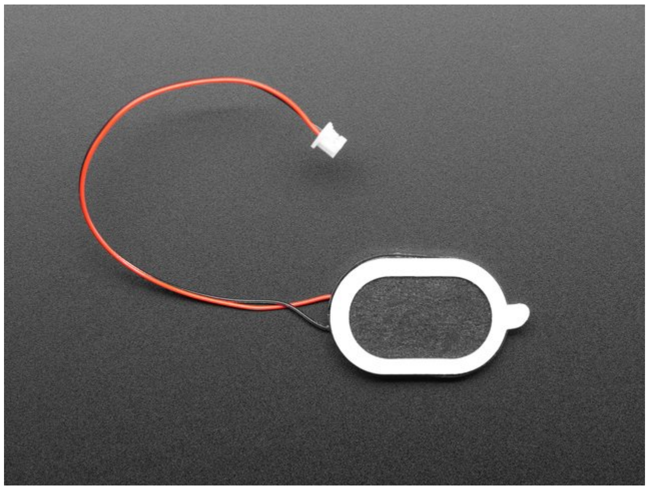
You might also want:



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

PRODUCT ID: 3898

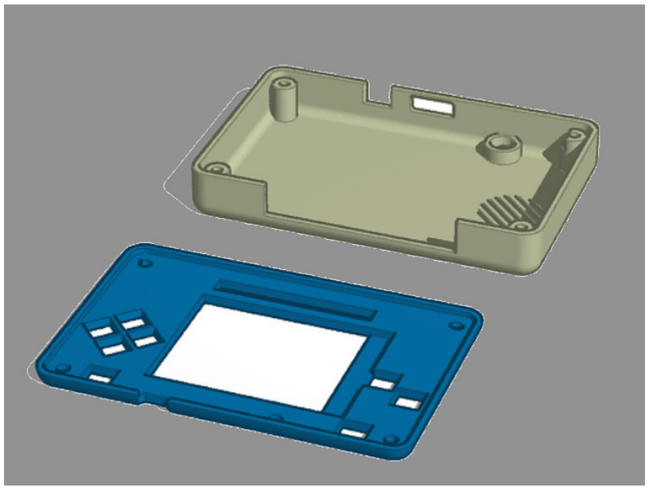
So that you can play the game without having it attached to a computer with a USB cable.



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.

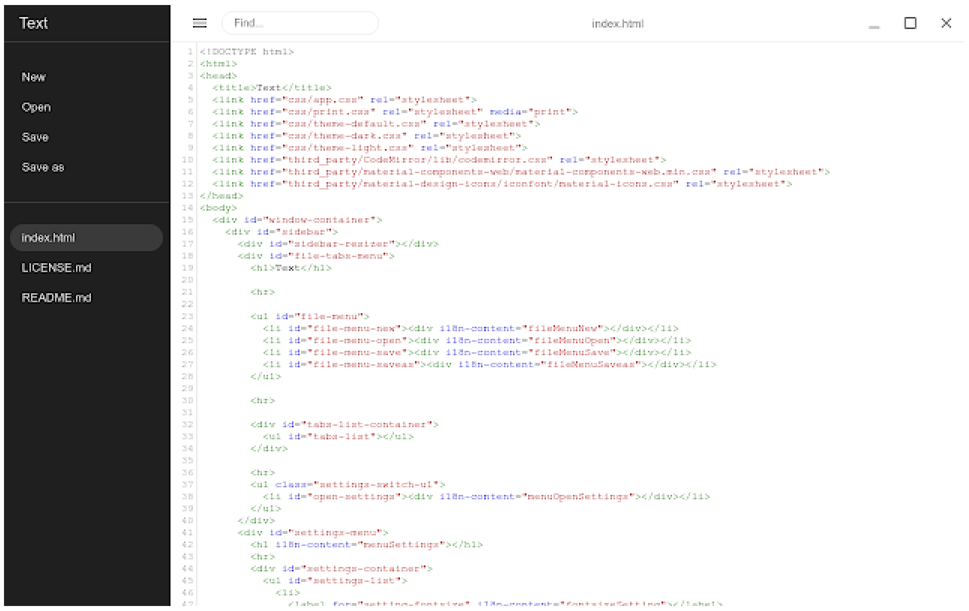
Just a reminder, if you are having any problems loading CircuitPython onto your PyBadge, ensure that you are using a USB cable that not only provides power, but also provides a data link. Many USB cables you buy are only for charging, not transfering data as well. Once the CircuitPython is all loaded,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

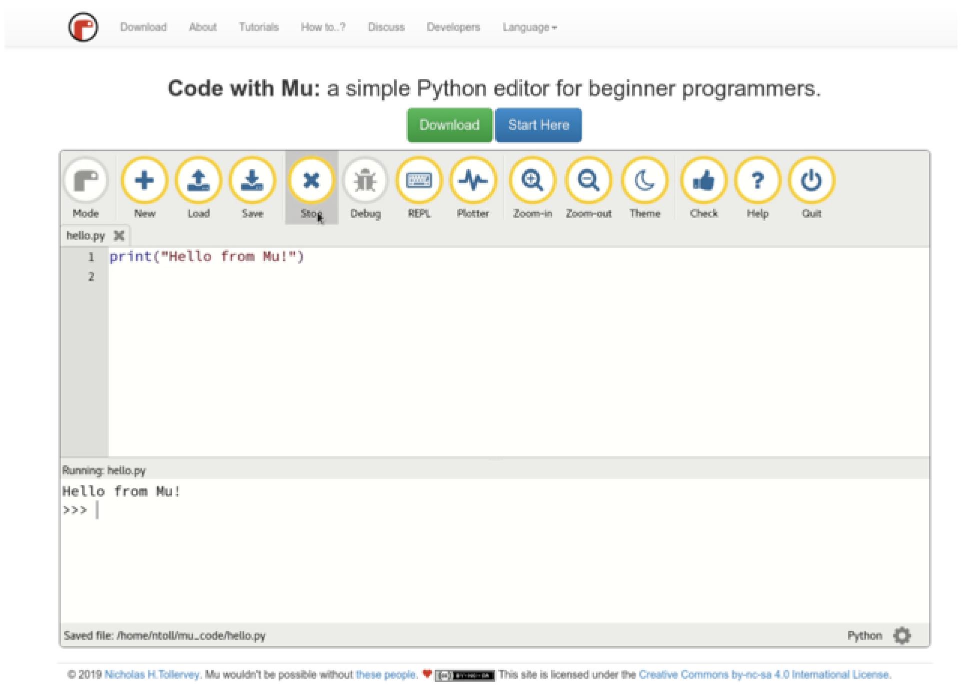
If you are working on a Chromebook, the easiest way to start coding is to just use the built in Text app. As soon as you open or save a file with a `*.py` extension, it will know it is Python code and automatically start syntax highlighting.



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

        <hr>

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

        <hr>

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

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

Fig. 1: Chromebook Text app

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



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

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

Running: hello.py

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

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

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

Fig. 2: Mu IDE

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

2.1 Hello, World!

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

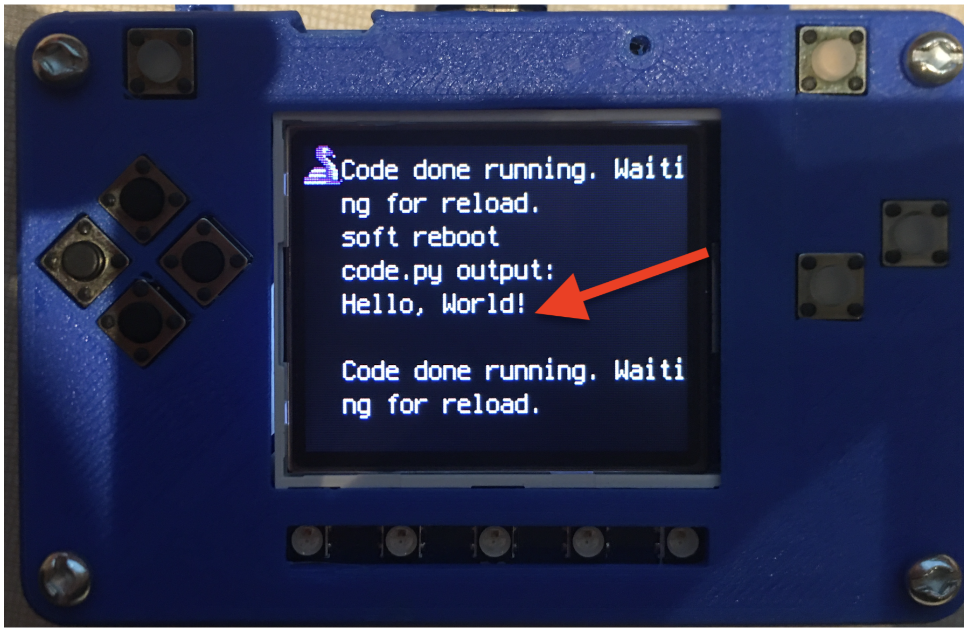
```python
print("Hello, World!")
```

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.

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

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


```

(continues on next page)



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

Code done running. Waiting for reload.
```

Fig. 3: Hello, World! program on PyBadge

(continued from previous page)

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


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

Congratulations, we are ready to start.

CHAPTER 3

Image Banks and Sounds

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 Asteroid Dodger

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:

- MT Studios image bank
- Snakob Studios image bank
- Game and menu image bank
- Snakob hiss sound
- Coin sound
- Ammo spawn sound
- Ammo collection sound
- Firing laser sound
- Asteroid destruction sound
- Asteroid collision 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 following steps are how to create the actual game scene itself. Here is the rundown of what you will be programming: you are controlling a space ship and must dodge on coming asteroids that travel across the screen. The asteroids spawn at random positions off screen. Throughout the game ammo packs will spawn throughout the map. If the ship runs over the ammo pack, the player picks it up and get one type of laser shot according to the colour of the pack they picked up. If the pack is red, the player gets one shot that travels straight out from the ship. If the pack was yellow, the user gets a spread shot with one laser travelling straight out and two others flanking it travelling diagonally. If the pack is blue, eight lasers travel out from the ship in different directions. The user can fire these lasers by pressing the "A" button. The lasers will travel in the direction of the last button pressed on the D-Pad. When a laser collides with an asteroid, both are removed from the scene. When a player hits an asteroid the game ends. The steps below will help you program your game scene according to what has been described here. I would advise you to follow the steps in order. You can find a video of the working game *here <https://www.youtube.com/watch?v=TlK0fTQpyVU>*

4.1 Constants

Before you start programming, you will need to create a file with constants in it. This file will be unchangeable from your code.py file to avoid inconsistencies in your code. Here are a few of the constants you will need:

- The size of the screen along the X axis
- The size of the screen along the Y axis
- The number of grid spaces along the X axis
- The number of grid spaces along the Y axis
- The size of your sprites
- The movement speed of the spaceship
- Coordinates for off the top of the screen
- Coordinates for off the left of the screen
- Coordinates for off the right of the screen

- Coordinates for off the bottom of the screen
- A cap for the amount of lasers you want
- A cap for the amount of asteroids you have per side of screen
- The X coordinate for sprites being stored off screen
- The Y coordinate for sprites being stored off screen
- The speed of the asteroids
- The speed of the lasers
- Another laser speed to feel consistent with lasers moving upwards during the 360 degree spread shot

Remember when creating your constants that they should all be written in capitals to distinguish them from other variables in your program.

```
SCREEN_X = 160
SCREEN_Y = 128
SCREEN_GRID_X = 16
SCREEN_GRID_Y = 8
SPRITE_SIZE = 16
SHIP_MOVEMENT_SPEED = 1
OFF_TOP_SCREEN = -1 - SPRITE_SIZE
OFF_LEFT_SCREEN = -1 - SPRITE_SIZE
OFF_RIGHT_SCREEN = SCREEN_X + SPRITE_SIZE
OFF_BOTTOM_SCREEN = SCREEN_Y + SPRITE_SIZE
LASER_CREATION_TOTAL = 8
ASTEROID_CREATION_TOTAL = 3
OFF_SCREEN_X = -500
OFF_SCREEN_Y = -100
ASTEROID_SPEED = 1
LASER_SPEED = 2
EXTRA_LASER_SPEED = 3
```

You will also want colour palettes for your text. Two will be provided here: one for the MT Studios splash scene, while the other is a more generic score palette.

```
MT_GAME_STUDIO_PALETTE = (b'\xf8\x1f\x00\x00\xcey\x00\xff\xf8\x1f\xff\x19\xfc\xe0\xfd\xe0'
   b'\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff')

SCORE_PALETTE = (b'\xf8\x1f\x00\x00\xcey\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff'
   b'\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff')
```

The last thing you will want in your constants file should be a list to keep track of the four button states: up (not pressed), just pressed, still pressed, and released.

```
# Using for button state
button_state = {
"button_up": "up",
"button_just_pressed": "just pressed",
"button_still_pressed": "still pressed",
"button_released": "released",
}
```

Once you have your constants file you should be able to start programming your actual game.

4.2 Background

The first thing you will want to do is get the background working on your game scene. You must first set the image bank to be the gamesprite image bank. Use a for loop to place a random background tile (chosen from either tile with or without a star on it) throughout the screen. Be sure this is set under your game scene function.

```
# The image bank for the game
image_bank_1 = stage.Bank.from_bmp16("gamesprite.bmp")

# sets the background to image 1 in the bank
background = stage.Grid(image_bank_1, 10, 8)
for x_location in range(constants.SCREEN_GRID_X):
    for y_location in range(constants.SCREEN_GRID_X):
        selected_tile = random.randint(0, 1)
        background.tile(x_location, y_location, selected_tile)
```

The next thing to do is to make sure the background is rendered on your game. You will need to set your framerate to 60, then set your game layers. You may not have any other layers than your background right now, however when you do add more be sure that the background layer is always the last one referenced. The final thing you need to do is render your game. These few lines of code should be placed just above the game function's game loop.

```
# 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()
```

You should now have a functioning background. Your initial game scene should look something like this:

```
game_scene():
    # This is the game scene for Asteroid Dodger

    # The image bank for the game
    image_bank_1 = stage.Bank.from_bmp16("gamesprite.bmp")

    # sets the background to image 1 in the bank
    background = stage.Grid(image_bank_1, 10, 8)
    for x_location in range(constants.SCREEN_GRID_X):
        for y_location in range(constants.SCREEN_GRID_X):
            background.tile(x_location, y_location, 0)

    # 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()

    # Game loop
    while True:
        # Get user input

```

(continues on next page)

(continued from previous page)

```
        # Update game logic

        # Redraw sprite list
        pass
```

4.3 Space Ship

The next step is to get your spaceship working. You have to start by creating a list that will hold your primary sprites (namely your spaceship). You will then need to create your spaceship sprite and append it to the list in the 0th position. Remember, this is done outside your game loop.

```
# This list contains the primary sprites
sprites = []

# Creating spaceship sprite
ship = stage.Sprite(image_bank_1, 14, 75, 56)
sprites.insert(0, ship)
```

You will need to paint your sprite onto the screen. You can do this by adding your sprite list in front of your background.

```
# 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()
```

The next thing you will need to do to ensure it continues to show up is to have it rendered on screen. You do this at the bottom of the inside of your game loop.

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

You will need to paint and render any new sprite list you add using the same methods. Finally, you will want to make sure your spaceship will be able to move, and keep it from moving off screen. This is all done in the game loop. The first thing you have to do is to set your keys to be watching if a button is being pressed. Next, since your user will be using the D-Pad to move around, you will want to add an if statement to detect if a specific button on the D-Pad. Depending on which button is pressed, your spaceship will move in a different direction. You will also want to make sure your spaceship can't move off screen. You can do this by putting an if statement inside your previous if statement. If the ship's X or Y coordinates goes off the screen limits indicated in your constants file, it will move the ship back on screen. There is also a variable that is changed each time a button on the D-Pad is pressed. This variable will be used later to determine the directions the lasers fire.

```
    # get user input
    keys = ugame.buttons.get_pressed()

    # Move ship right
    if keys & ugame.K_RIGHT:
        state_of_button = 2
        if ship.x > constants.SCREEN_X - constants.SPRITE_SIZE:
            ship.move(constants.SCREEN_X - constants.SPRITE_SIZE, ship.y)
```

(continues on next page)

(continued from previous page)

```
        else:
            ship.move(ship.x + constants.SHIP_MOVEMENT_SPEED, ship.y)
        pass

    # Move ship left
    if keys & ugame.K_LEFT:
        state_of_button = 4
        if ship.x < 0:
            ship.move(0, ship.y)
        else:
            ship.move(ship.x - constants.SHIP_MOVEMENT_SPEED, ship.y)
        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)
        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)
        else:
            ship.move(ship.x, ship.y + constants.SHIP_MOVEMENT_SPEED)
        pass
```

Your spaceship should now be able to move properly without going off screen.

4.4 Asteroids

Once you have your spaceship working you can now add functionality for the asteroids. To have asteroids at each side of the screen, you will need four different lists (one for each side). Use a for loop to create asteroids up to the set amount from your constants file, and append the asteroids to their lists. This is to be done outside your gameloop. Be sure your asteroids have been painted and rendered on screen.

```
# Creating asteroids
# Asteroids staring from the left
left_asteroids = []
for left_asteroid_number in range(constants.ASTEROID_CREATION_TOTAL):
    single_left_asteroid = stage.Sprite(image_bank_1, 4,
                                        constants.OFF_SCREEN_X,
                                        constants.OFF_SCREEN_Y)
    left_asteroids.append(single_left_asteroid)
reset_left_asteroid()

# Asteroids staring from the top
top_asteroids = []
for top_asteroid_number in range(constants.ASTEROID_CREATION_TOTAL):
    single_up_asteroid = stage.Sprite(image_bank_1, 5,
```

(continues on next page)

(continued from previous page)

```
                                            constants.OFF_SCREEN_X,
                                            constants.OFF_SCREEN_Y)
    top_asteroids.append(single_up_asteroid)
reset_top_asteroid()

# Asteroids starting from the right
right_asteroids = []
for right_asteroid_number in range(constants.ASTEROID_CREATION_TOTAL):
    single_right_asteroid = stage.Sprite(image_bank_1, 6,
                                            constants.OFF_SCREEN_X,
                                            constants.OFF_SCREEN_Y)
    right_asteroids.append(single_right_asteroid)
reset_right_asteroid()

# Asteroids staring from the bottom
bottom_asteroids = []
for down_asteroid_number in range(constants.ASTEROID_CREATION_TOTAL):
    single_down_asteroid = stage.Sprite(image_bank_1, 7,
                                            constants.OFF_SCREEN_X,
                                            constants.OFF_SCREEN_Y)
    bottom_asteroids.append(single_down_asteroid)
reset_bottom_asteroid()
```

When you have your asteroids ready, you will need to be able to have them move across the screen. Once again, each side of their screen will need a way to scroll asteroids. For this, create a for loop in your game loop that itterates through all the asteroids in a list. An if statement within will deterimine if the asteroid isn't in purgatory off screen, and will scroll across the screen in the desired direction. Within said if statement should be another if statement determining if the asteroid has reached the other side of the screen. If the asteroid has, it will be moved back into purgatory off screen and wait to be sent out again.

```
        # Scroll asteroids from left of screen
        for left_asteroid_number in range(len(left_asteroids)):
            if left_asteroids[left_asteroid_number].x < constants.OFF_RIGHT_SCREEN:
                left_asteroids[left_asteroid_number].move(
                left_asteroids[left_asteroid_number].x + constants.ASTEROID_SPEED,
                left_asteroids[left_asteroid_number].y)
                if left_asteroids[left_asteroid_number].x > constants.SCREEN_X:
                    left_asteroids[left_asteroid_number].move(constants.OFF_SCREEN_X,
                                                              constants.OFF_SCREEN_Y)
                    reset_left_asteroid()

        # Scroll asteroids from top of screen
        for top_asteroid_number in range(len(top_asteroids)):
            if top_asteroids[top_asteroid_number].y < constants.OFF_BOTTOM_SCREEN:
                top_asteroids[top_asteroid_number].move(
                top_asteroids[top_asteroid_number].x,
                top_asteroids[top_asteroid_number].y + constants.ASTEROID_SPEED)
                if top_asteroids[top_asteroid_number].y > constants.SCREEN_Y:
                    top_asteroids[top_asteroid_number].move(constants.OFF_SCREEN_X,
                                                            constants.OFF_SCREEN_Y)
                    reset_top_asteroid()

        # Scroll asteroids from right of screen left
        for right_asteroid_number in range(len(right_asteroids)):
            if right_asteroids[right_asteroid_number].x > constants.OFF_LEFT_SCREEN:
                right_asteroids[right_asteroid_number].move(
```

(continues on next page)

(continued from previous page)

```
                right_asteroids[right_asteroid_number].x - constants.ASTEROID_SPEED,
                right_asteroids[right_asteroid_number].y)
                if right_asteroids[right_asteroid_number].x < 0 - constants.SPRITE_SIZE:
                    right_asteroids[right_asteroid_number].move(constants.OFF_SCREEN_X,
                                                                constants.OFF_SCREEN_Y)
                    reset_right_asteroid()

        # Scroll asteroids from bottom of screen
        for down_asteroid_number in range(len(bottom_asteroids)):
            if bottom_asteroids[down_asteroid_number].y > constants.OFF_TOP_SCREEN:
                bottom_asteroids[down_asteroid_number].move(
                bottom_asteroids[down_asteroid_number].x,
                bottom_asteroids[down_asteroid_number].y - constants.ASTEROID_SPEED)
                if bottom_asteroids[down_asteroid_number].y < 0 - constants.SPRITE_SIZE:
                    bottom_asteroids[down_asteroid_number].move(constants.OFF_SCREEN_X,
                                                                constants.OFF_SCREEN_Y)
                    reset_bottom_asteroid()
```

The final thing you will need is a way to set and reset the asteroids off screen at a random location and distance so they remain unpredictable to the player. You will need to create four seperate functions outside your game loop, one for each asteroid list. These functions will be called immediately after the creation of your asteroids and when they reach the other side of the screen from where they started (they must be called after all the processes from above). The proper placement of the function calls is displayed in the sample code above. When the function is called, it will have a for loop with an if statement that checks each asteroid of the particular list to see if it is on screen or not. This is similar to what you have done above. If the asteroid is read as in purgatory off screen, it will be moved to a random X and Y coordinate just off the screen and begin its way across the screen. This way, each time the function is called the asteroid will reset itself without interfering with the other asteroids.

```
# These functions set and reset the start coordinates of asteroids
def reset_left_asteroid():
    # Sets and resets the start coordinates of asteroids starting on the left
    for left_asteroid_number in range(len(left_asteroids)):
        if left_asteroids[left_asteroid_number].x < 0:
            left_asteroids[left_asteroid_number].move(random.randint
                                                      (-100, 0 -
                                                       constants.SPRITE_SIZE),
                                                      random.randint
                                                      (0, constants.SCREEN_Y))
            break

def reset_top_asteroid():
    # Sets and resets the start coordinates of asteroids starting on the top
    for top_asteroid_number in range(len(top_asteroids)):
        if top_asteroids[top_asteroid_number].y < 0:
            top_asteroids[top_asteroid_number].move(random.randint
                                                    (0, constants.SCREEN_X),
                                                    random.randint
                                                    (-100, 0 -
                                                     constants.SPRITE_SIZE))
            break

def reset_right_asteroid():
    # Sets and resets the start coordinates of asteroids starting on the right
    for right_asteroid_number in range(len(right_asteroids)):
        if right_asteroids[right_asteroid_number].x < 0:
            right_asteroids[right_asteroid_number].move(random.randint
```

(continues on next page)

(continued from previous page)

```
                                                                  (constants.SCREEN_X, 228),
                                                                  random.randint
                                                                  (0, constants.SCREEN_Y))
               break

    def reset_bottom_asteroid():
        # Sets and resets the start coordinates of asteroids starting on the bottom
        for down_asteroid_number in range(len(bottom_asteroids)):
            if bottom_asteroids[down_asteroid_number].y < 0:
                bottom_asteroids[down_asteroid_number].move(random.randint
                                                        (0, constants.SCREEN_X),
                                                        random.randint
                                                        (160 + constants.SPRITE_SIZE,
                                                         260))
                break
```

You should now have asteroids that scroll across the screen from all four directions and are able to reset themselves.

4.5 Ammo Packs

The space ship is not able to fire lasers immediately. For this, it needs ammo packs. I will cover mechanics on the different lasers you can fire later and how to do so, but for now this will show you how the ammo packs work. The first thing you will need is to create a new list for your ammo pack sprites. There are three different ammo packs you will need to generate and append to the list. Assure that they originate in purgatory off screen. Also be sure that you have added them to be painted and rendered on the screen.

```
# This list contains the ammo packs
ammo = []

# Creating ammo pack sprites
single_shot = stage.Sprite(image_bank_1, 15,
                           constants.OFF_SCREEN_X,
                           constants.OFF_SCREEN_Y)
ammo.append(single_shot)
spread_shot = stage.Sprite(image_bank_1, 2,
                           constants.OFF_SCREEN_X,
                           constants.SCREEN_GRID_Y)
ammo.append(spread_shot)
around_shot = stage.Sprite(image_bank_1, 3, constants.OFF_SCREEN_X,
                           constants.OFF_SCREEN_Y)
ammo.append(around_shot)
```

Next you will need a function that calls on a random ammo pack and places it somewhere random on the screen. The first thing you will need to do for this is set some initial values. These values include a variable for the timer, and a random number generator that determines when the timer stops and resets. Some of these values will be used later when determining what lasers fire and how, but they will be used in the next step.

```
# Setting the ammo generation timer and values
timer = 0
generation_time = random.randint(300, 1000)
ammo_type = 0
firing_type = 0
state_of_button = 0
```

You now need to make the function itself. The first three lines in the function should make sure the ammo packs are all in purgatory before a new one is placed. This way, there won't be two ammo packs on screen at once. Next you need a random number generator set between 1 and 100 that will determine what type of ammo pack spawns. When the function is called, an ammo pack will be moved on screen depending on which number was chosen. You should have a 50% chance of getting a singular projectile pack, 30% chance of getting a three projectile spread, and 20% chance of getting an eight projectile all around shot. Also depending on which pack was chosen, a variable has to be updated to indicate what kind of laser to fire when the player decides to use the ammo.

```python
# This function randomly generates ammo packs
def spawn_ammo():
    single_shot.move(constants.OFF_SCREEN_X, constants.OFF_SCREEN_Y)
    spread_shot.move(constants.OFF_SCREEN_X, constants.OFF_SCREEN_Y)
    around_shot.move(constants.OFF_SCREEN_X, constants.OFF_SCREEN_Y)
    type_of_ammo = random.randint(1, 100)
    ammo_variant = 0
    if type_of_ammo <= 50:
        ammo_variant = 1
        single_shot.move(random.randint(0 + constants.SPRITE_SIZE,
                                        constants.SCREEN_X -
                                        constants.SPRITE_SIZE),
                         random.randint(0 + constants.SPRITE_SIZE,
                                        constants.SCREEN_Y -
                                        constants.SPRITE_SIZE))
    elif type_of_ammo >= 51 and type_of_ammo <= 80:
        ammo_variant = 2
        spread_shot.move(random.randint(0 + constants.SPRITE_SIZE,
                                        constants.SCREEN_X -
                                        constants.SPRITE_SIZE),
                         random.randint(0 + constants.SPRITE_SIZE,
                                        constants.SCREEN_Y -
                                        constants.SPRITE_SIZE))
    elif type_of_ammo >= 81:
        ammo_variant = 3
        around_shot.move(random.randint(0 + constants.SPRITE_SIZE,
                                        constants.SCREEN_X -
                                        constants.SPRITE_SIZE),
                         random.randint(0 + constants.SPRITE_SIZE,
                                        constants.SCREEN_Y -
                                        constants.SPRITE_SIZE))
```

Something important you will need is a timer that calls your spawn ammo function after a short period of time. In the game loop, create a for loop that itterates through a small set of numbers. Each time the itteration is complete, add 1 to your pre established timer variable. When your timer equals your generation time variable, your ammo function is called. The timer resets to zero, a new random number is chosen for the generation time, and the process repeats itself.

```python
        # Ammo spawn timer
        for counter in range(1, 61):
            if counter == 60:
                timer = timer + 1
                if timer == generation_time:
                    ammo_type = spawn_ammo()
                    timer = 0
                    generation_time = random.randint(300, 1000)
                else:
                    continue
```

The last thing you will need is something to detect if the spaceship has collided with (picked up) the ammo pack. To do this you will need a for loop watching if a series of coordinates (hitbox) on each have intersected. If this happens, the ammo pack is removed from the screen. At this point the player should be able to fire a specific laser pattern depending on which ammo pack they picked up.

```
    # This detects if the ship has hit and collected an ammo pack
    for ammo_number in range(len(ammo)):
        if ammo[ammo_number].x > 0:
            for sprite_number in range(len(sprites)):
                if sprites[sprite_number].x > 0:
                    if stage.collide(ammo[ammo_number].x + 6,
                                     ammo[ammo_number].y + 3,
                                     ammo[ammo_number].x + 10,
                                     ammo[ammo_number].y + 13,
                                     sprites[sprite_number].x + 1,
                                     sprites[sprite_number].y + 1,
                                     sprites[sprite_number].x + 14,
                                     sprites[sprite_number].y + 14):
                        ammo[ammo_number].move(constants.OFF_SCREEN_X,
                                               constants.OFF_SCREEN_Y)
                        sound.stop()
                        sound.play(load_sound)
```

Your ammo packs should now be fully functional.

4.6 Sounds

Before I continue on with the lasers, lets talk a bit about adding sound to the game. For Asteroid Dodger, you need 5 distinct sounds:

- Loading sound that plays when a player gets an ammo pack
- Laser sound that plays when a laser is fired
- An ammo spawning sound that plays when an ammo pack appears on the screen
- An impact sound that indicates when an asteroid has been destroyed by a laser
- A crash sound to indicate that an asteroid has struck the player's spaceship

To load the sounds, you need to open up the sounds in your game function. You will then need to define your sound variable, and make sure that the sounds are not muted.

```
# Getting sounds ready
laser_sound = open("laser.wav", 'rb')
crash_sound = open("crash.WAV", 'rb')
ammo_sound = open("ammo.wav", 'rb')
load_sound = open("load.wav", 'rb')
impact_sound = open("impact.wav", 'rb')
sound = ugame.audio
sound.stop()
sound.mute(False)
```

You then need to add them where needed. For example, when you pick up an ammo pack, the load sound should play. Or, when the spawn ammo function is called, the ammo sound plays. You can do this by using the sound.play(*your desired sound*) function and passing it in the sound you want to play. Always make sure a sound.stop() is included before you play the new sound as to not distract from a sound that may currently be playing.

```
        # This detects if the ship has hit and collected an ammo pack
        for ammo_number in range(len(ammo)):
            if ammo[ammo_number].x > 0:
                for sprite_number in range(len(sprites)):
                    if sprites[sprite_number].x > 0:
                        if stage.collide(ammo[ammo_number].x + 6,
                                         ammo[ammo_number].y + 3,
                                         ammo[ammo_number].x + 10,
                                         ammo[ammo_number].y + 13,
                                         sprites[sprite_number].x + 1,
                                         sprites[sprite_number].y + 1,
                                         sprites[sprite_number].x + 14,
                                         sprites[sprite_number].y + 14):
                            ammo[ammo_number].move(constants.OFF_SCREEN_X,
                                                   constants.OFF_SCREEN_Y)
                            sound.stop()
                            sound.play(load_sound)
```

The above example is that when the spaceship picks up an ammo pack the loading sound plays. As you are programming, you may add sounds that are applicable to the instances they may be needed in.

4.7 Lasers

The next thing to get working is actually firing the lasers themselves. Like all the other sprites, they must first be created and added to a list. Make sure the list is both painted and rendered on screen. Similar to the asteroids, the lasers will be created and appended to a list with a for loop and will appear in purgatory off screen.

```
    # This list contains the laser sprites
    lasers = []

    # Generating laser sprites
    for laser_number in range(constants.LASER_CREATION_TOTAL):
        single_laser = stage.Sprite(image_bank_1, 10,
                                    constants.OFF_SCREEN_X,
                                    constants.OFF_SCREEN_Y)
        lasers.append(single_laser)
```

The first think we will want to do before we program the lasers is to have a way to count the asteroids that have been hit by the lasers. To do this, initialize a variable equal to 0 to keep score outside your game loop. Every time an asteroid is hit with a laser, the score will increase by one.

```
    # Score counter for the asteroids
    asteroid_counter = 0
```

Lasers are fundamentaly the hardest and most tedious part of this program. Before we start working on the lasers, we need a method of firing them. The player must use the A button to fire their lasers after they pick up an ammo pack. We first need to set up a way of detecting the state of the A button as we want the lasers to fire when the button has just been pressed. This should be in the game loop.

```
        # A button to fire
        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"]:
```

(continues on next page)

(continued from previous page)

```
                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"]
```

Next we need to program multiple ways for the ammo to fire once the A button has been pressed, as well as the different types of ammo. I am going to do this through a large if statement with smaller if statements in between to figure out what direction to fire and how many lasers to fire. I will start with detecting what kind of ammo the user has in the game loop. The variable that determines this is the firing type. If the user hits the A button and they have not picked up an ammo pack, nothing will happen. If the user has picked up a red ammo pack, one laser will be moved to the ship's coordinates. If the user has picked up a yellow ammo pack, three lasers will be moved to the ship's coordinates. If the user has picked up a blue ammo pack, all eight laser will be moved to the ship's coordinates. On the end of each there is four if statements that detect which button was last pressed on the D-Pad. For the singular projectile and three projectile spread, they will always travel in the direction of the last button pressed on the D-Pad. In any instance that lasers are summoned to the ship's coordinates, the laser sound will play indicating the player has fired a laser.

```
        # Firing ammo using the a button
        if a_button == constants.button_state["button_just_pressed"]:
            for laser_number in range(len(lasers)):
                # No ammo
                if ammo_type == 0:
                    break
                # Single shot
                elif ammo_type == 1:
                    if lasers[1].x < 0:
                        lasers[1].move(ship.x, ship.y)
                        sound.stop()
                        sound.play(laser_sound)
                        firing_type = 1
                        ammo_type = 0
                        if state_of_button == 1:
                            firing_direction = 1
                        elif state_of_button == 2:
                            firing_direction = 2
                        elif state_of_button == 3:
                            firing_direction = 3
                        elif state_of_button == 4:
                            firing_direction = 4
                    break
                # Spread shot
                elif ammo_type == 2:
                    if lasers[1].x < 0:
                        lasers[1].move(ship.x, ship.y)
                    if lasers[2].x < 0:
                        lasers[2].move(ship.x, ship.y)
                    if lasers[3].x < 0:
                        lasers[3].move(ship.x, ship.y)
                        sound.stop()
                        sound.play(laser_sound)
                        firing_type = 2
                        ammo_type = 0
                        if state_of_button == 1:
                            firing_direction = 1
```

(continues on next page)

(continued from previous page)

```
                        elif state_of_button == 2:
                            firing_direction = 2
                        elif state_of_button == 3:
                            firing_direction = 3
                        elif state_of_button == 4:
                            firing_direction = 4
                        break
                    # Around shot
                elif ammo_type == 3:
                    if lasers[0].x < 0:
                        lasers[0].move(ship.x, ship.y)
                    if lasers[1].x < 0:
                        lasers[1].move(ship.x, ship.y)
                    if lasers[2].x < 0:
                        lasers[2].move(ship.x, ship.y)
                    if lasers[3].x < 0:
                        lasers[3].move(ship.x, ship.y)
                    if lasers[4].x < 0:
                        lasers[4].move(ship.x, ship.y)
                    if lasers[5].x < 0:
                        lasers[5].move(ship.x, ship.y)
                    if lasers[6].x < 0:
                        lasers[6].move(ship.x, ship.y)
                    if lasers[7].x < 0:
                        lasers[7].move(ship.x, ship.y)
                        sound.stop()
                        sound.play(laser_sound)
                        firing_type = 3
                        ammo_type = 0
                        break
                    else:
                        continue
```

Now we need a way to move the lasers across the screen in the desired direction. To do this we will have a for loop in the game loop that continuosly itterate through all eight lasers.

```
        # Firing lasers
        for laser_number in range(len(lasers)):
```

There are three large chunks inside this for loop. The first is the singular projectile shot. If the laser isn't off screen in purgatory (because the player has pressed the A button), the laser will scroll in a specific direction according to the last button pressed on the D-Pad. Once the laser reaches off screen, it is moved back to purgatory to await its next use. The type of ammo and firing direction is then reupdated to zero as to not cause problems with future ammo packs collected and lasers fired.

```
            # Single shot
            if firing_type == 1:
                # Upwards shot
                if lasers[1].x > 0 and firing_direction == 1:
                    lasers[1].move(lasers[1].x, lasers[1].y -
                                   constants.LASER_SPEED)
                    if lasers[1].y < constants.OFF_TOP_SCREEN:
                        lasers[1].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                        firing_type = 0
                        firing_direction = 0
```

(continues on next page)

(continued from previous page)

```
                # Right shot
                elif lasers[1].y > 0 and firing_direction == 2:
                    lasers[1].move(lasers[1].x + constants.LASER_SPEED,
                                   lasers[1].y)
                    if lasers[1].x >= constants.OFF_RIGHT_SCREEN:
                        lasers[1].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                        firing_type = 0
                        firing_direction = 0
                # Downwards shot
                elif lasers[1].x > 0 and firing_direction == 3:
                    lasers[1].move(lasers[1].x, lasers[1].y +
                                   constants.LASER_SPEED)
                    if lasers[1].y >= constants.OFF_BOTTOM_SCREEN:
                        lasers[1].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                        firing_type = 0
                        firing_direction = 0
                # Left shot
                elif lasers[1].y > 0 and firing_direction == 4:
                    lasers[1].move(lasers[1].x - constants.LASER_SPEED,
                                   lasers[1].y)
                    if lasers[1].x < constants.OFF_LEFT_SCREEN:
                        lasers[1].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                        firing_type = 0
                        firing_direction = 0
```

The second chunk is the three projectile spread shot. It functions similarly to the singular projectile. If the laser is not in purgatory, one of the three lasers travels straight across the screen according to the firing direction. The two other lasers travel diagonally in the same relative direction as the first one on a perpendicular angle from one another. All of them travel at the same speed. All three lasers must leave the screen before they are sent back to their off screen purgatory. This way if a laser is removed early for striking an asteroid the other two don't dissappear for what seems like no reason. After being moved back to purgatory, just like the singular shot, the type of ammo and firing direction is then reupdated to zero as to not cause problems with future ammo packs collected and lasers fired.

```
            if firing_type == 2:
                # Up shot
                if lasers[laser_number].y > -17 and firing_direction == 1:
                    lasers[1].move(lasers[1].x, lasers[1].y
                                   - constants.LASER_SPEED)
                    lasers[2].move(lasers[2].x + constants.LASER_SPEED,
                                   lasers[2].y - constants.LASER_SPEED)
                    lasers[3].move(lasers[3].x - constants.LASER_SPEED,
                                   lasers[3].y - constants.LASER_SPEED)
                    if lasers[laser_number].y < constants.OFF_TOP_SCREEN:
                        lasers[1].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                    if lasers[2].y < constants.OFF_TOP_SCREEN:
                        lasers[2].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                    if lasers[3].y < constants.OFF_TOP_SCREEN:
                        lasers[3].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                    if lasers[1].x == constants.OFF_SCREEN_X and \
                       lasers[2].x == constants.OFF_SCREEN_X and \
```

(continues on next page)

(continued from previous page)

```
                     lasers[3].x == constants.OFF_SCREEN_X:
                      firing_type = 0
                      firing_direction = 0
              # Right shot
              elif lasers[laser_number].x < 176 and firing_direction == 2:
                  lasers[1].move(lasers[1].x + constants.LASER_SPEED,
                                 lasers[1].y)
                  lasers[2].move(lasers[2].x + constants.LASER_SPEED,
                                 lasers[2].y - constants.LASER_SPEED)
                  lasers[3].move(lasers[3].x + constants.LASER_SPEED,
                                 lasers[3].y + constants.LASER_SPEED)
                  if lasers[1].x >= constants.OFF_RIGHT_SCREEN:
                      lasers[1].move(constants.OFF_SCREEN_X,
                                     constants.OFF_SCREEN_Y)
                  if lasers[2].x >= constants.OFF_RIGHT_SCREEN:
                      lasers[2].move(constants.OFF_SCREEN_X,
                                     constants.OFF_SCREEN_Y)
                  if lasers[3].x >= constants.OFF_RIGHT_SCREEN:
                      lasers[3].move(constants.OFF_SCREEN_X,
                                     constants.OFF_SCREEN_Y)
                  if lasers[1].x == constants.OFF_SCREEN_X and \
                     lasers[2].x == constants.OFF_SCREEN_X and \
                     lasers[3].x == constants.OFF_SCREEN_X:
                      firing_type = 0
                      firing_direction = 0
              # Downwards shot
              elif lasers[laser_number].y > 0 and firing_direction == 3:
                  lasers[1].move(lasers[1].x, lasers[1].y +
                                 constants.LASER_SPEED)
                  lasers[2].move(lasers[2].x - constants.LASER_SPEED,
                                 lasers[2].y + constants.LASER_SPEED)
                  lasers[3].move(lasers[3].x + constants.LASER_SPEED,
                                 lasers[3].y + constants.LASER_SPEED)
                  if lasers[1].y >= constants.OFF_BOTTOM_SCREEN:
                      lasers[1].move(constants.OFF_SCREEN_X,
                                     constants.OFF_SCREEN_Y)
                  if lasers[2].y >= constants.OFF_BOTTOM_SCREEN:
                      lasers[2].move(constants.OFF_SCREEN_X,
                                     constants.OFF_SCREEN_Y)
                  if lasers[3].y >= constants.OFF_BOTTOM_SCREEN:
                      lasers[3].move(constants.OFF_SCREEN_X,
                                     constants.OFF_SCREEN_Y)
                  if lasers[1].x == constants.OFF_SCREEN_X and \
                     lasers[2].x == constants.OFF_SCREEN_X and \
                     lasers[3].x == constants.OFF_SCREEN_X:
                      firing_type = 0
                      firing_direction = 0
              # Left shot
              elif lasers[laser_number].x > -17 and firing_direction == 4:
                  lasers[1].move(lasers[1].x - constants.LASER_SPEED,
                                 lasers[1].y)
                  lasers[2].move(lasers[2].x - constants.LASER_SPEED,
                                 lasers[2].y + constants.LASER_SPEED)
                  lasers[3].move(lasers[3].x - constants.LASER_SPEED,
                                 lasers[3].y - constants.LASER_SPEED)
                  if lasers[1].x < constants.OFF_LEFT_SCREEN:
                      lasers[1].move(constants.OFF_SCREEN_X,
```

(continues on next page)

(continued from previous page)

```
                                                constants.OFF_SCREEN_Y)
                    if lasers[2].x < constants.OFF_LEFT_SCREEN:
                        lasers[2].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                    if lasers[3].x < constants.OFF_LEFT_SCREEN:
                        lasers[3].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                    if lasers[1].x == constants.OFF_SCREEN_X and \
                       lasers[2].x == constants.OFF_SCREEN_X and \
                       lasers[3].x == constants.OFF_SCREEN_X:
                        firing_type = 0
                        firing_direction = 0
```

The third and final chunk of the for loop is the all around shot. This shot sends all eight traveling in different directions from one another. Four are heading straight across the screen, either vertically or horizontally parallel to its edge. The other four are traveling diagonally across the screen relatively towards each of the corners, similar to the two flanking projectiles from the spread shot. The two projectiles traveling parallel to the horizontal edge of the screen are traveling at a heightened speed from the rest of the projectiles. The reason for this is that during early play tests it felt and looked clumsy to have those two lasers leave the screen last because the screen is a rectangle with the horizontal edges longer than the vertical ones. All eight projectiles must be off screen before the lasers are moved back to purgatory. Just like the singular and spread shots, after the lasers have been moved back to purgatory, the type of ammo and firing direction is then reupdated to zero as to not cause problems with future ammo packs collected and lasers fired.

```
            # Around shot
            if firing_type == 3:
                if lasers[laser_number].x > -17:
                    lasers[0].move(lasers[0].x, lasers[0].y -
                                   constants.LASER_SPEED)
                    lasers[1].move(lasers[1].x + constants.LASER_SPEED,
                                   lasers[1].y - constants.LASER_SPEED)
                    lasers[2].move(lasers[2].x + constants.EXTRA_LASER_SPEED,
                                   lasers[2].y)
                    lasers[3].move(lasers[3].x + constants.LASER_SPEED,
                                   lasers[3].y + constants.LASER_SPEED)
                    lasers[4].move(lasers[4].x, lasers[4].y +
                                   constants.LASER_SPEED)
                    lasers[5].move(lasers[5].x - constants.LASER_SPEED,
                                   lasers[5].y + constants.LASER_SPEED)
                    lasers[6].move(lasers[6].x - constants.EXTRA_LASER_SPEED,
                                   lasers[6].y)
                    lasers[7].move(lasers[7].x - constants.LASER_SPEED,
                                   lasers[7].y - constants.LASER_SPEED)
                    if lasers[0].y < constants.OFF_TOP_SCREEN:
                        lasers[0].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                    if lasers[1].y < constants.OFF_TOP_SCREEN:
                        lasers[1].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                    if lasers[2].x >= 176:
                        lasers[2].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                    if lasers[3].y > constants.OFF_BOTTOM_SCREEN:
                        lasers[3].move(constants.OFF_SCREEN_X,
                                       constants.OFF_SCREEN_Y)
                    if lasers[4].y > constants.OFF_BOTTOM_SCREEN:
                        lasers[4].move(constants.OFF_SCREEN_X,
```

(continues on next page)

(continued from previous page)

```
                                     constants.OFF_SCREEN_Y)
                  if lasers[5].y > constants.OFF_BOTTOM_SCREEN:
                      lasers[5].move(constants.OFF_SCREEN_X,
                                     constants.OFF_SCREEN_Y)
                  if lasers[6].x <= -17:
                      lasers[6].move(constants.OFF_SCREEN_X,
                                     constants.OFF_SCREEN_Y)
                  if lasers[7].y < constants.OFF_TOP_SCREEN:
                      lasers[7].move(constants.OFF_SCREEN_X,
                                     constants.OFF_SCREEN_Y)
                  if lasers[0].x == constants.OFF_SCREEN_X and \
                     lasers[1].x == constants.OFF_SCREEN_X and \
                     lasers[2].x == constants.OFF_SCREEN_X and \
                     lasers[3].x == constants.OFF_SCREEN_X and \
                     lasers[4].x == constants.OFF_SCREEN_X and \
                     lasers[5].x == constants.OFF_SCREEN_X and \
                     lasers[6].x == constants.OFF_SCREEN_X and \
                     lasers[7].x == constants.OFF_SCREEN_X:
                      firing_type = 0
                      firing_direction = 0
```

The final thing you will need is a way to detect if there has been a collision between the lasers and asteroids. This will be done in a way similar to how a player picks up ammo packs. A for loop will itterate through both the asteroids and the lasers to determine if either of their hit boxes ever overlap. Like the ammo-spaceship collision detect, this is to be done in the game loop. If there is an overlap, the hit asteroid will be taken off screen and the proper reset asteroid function will be called. The laser that hit the asteroid will be moved back to purgatory. When any asteroid is hit, the impact sound plays to indicate an asteroid has been destroyed. The score variable also increases by one every time an asteroid is hit with a laser. As there are four different asteroid lists, there has to be four different for loops, one that detects collisions between a laser and an asteroid of its respective list.

```
        # This detects if any lasers hit asteroids heading right
        for laser_number in range(len(lasers)):
            if lasers[laser_number].x > 0:
                for asteroid_number in range(len(left_asteroids)):
                    if left_asteroids[asteroid_number].x > 0:
                        if stage.collide(left_asteroids[asteroid_number].x + 1,
                                         left_asteroids[asteroid_number].y + 1,
                                         left_asteroids[asteroid_number].x + 15,
                                         left_asteroids[asteroid_number].y + 15,
                                         lasers[laser_number].x + 3,
                                         lasers[laser_number].y + 3,
                                         lasers[laser_number].x + 13,
                                         lasers[laser_number].y + 13):
                            left_asteroids[asteroid_number].move(constants.OFF_SCREEN_X,
                                                                 constants.OFF_SCREEN_Y)
                            lasers[laser_number].move(constants.OFF_SCREEN_X,
                                                      constants.OFF_SCREEN_Y)
                            sound.stop()
                            sound.play(impact_sound)
                            reset_left_asteroid()
                            asteroid_counter = asteroid_counter + 1

        # This detects if any lasers hit asteroids heading down
        for laser_number in range(len(lasers)):
            if lasers[laser_number].x > 0:
                for asteroid_number in range(len(top_asteroids)):
```

(continues on next page)

(continued from previous page)

```
                    if top_asteroids[asteroid_number].x > 0:
                        if stage.collide(top_asteroids[asteroid_number].x + 1,
                                         top_asteroids[asteroid_number].y + 1,
                                         top_asteroids[asteroid_number].x + 15,
                                         top_asteroids[asteroid_number].y + 15,
                                         lasers[laser_number].x + 3,
                                         lasers[laser_number].y + 3,
                                         lasers[laser_number].x + 13,
                                         lasers[laser_number].y + 13):
                            top_asteroids[asteroid_number].move(constants.OFF_SCREEN_X,
                                                                constants.OFF_SCREEN_Y)
                            lasers[laser_number].move(constants.OFF_SCREEN_X,
                                                      constants.OFF_SCREEN_Y)
                            sound.stop()
                            sound.play(impact_sound)
                            reset_top_asteroid()
                            asteroid_counter = asteroid_counter + 1

        # This detects if any lasers hit asteroids heading left
        for laser_number in range(len(lasers)):
            if lasers[laser_number].x > 0:
                for asteroid_number in range(len(right_asteroids)):
                    if right_asteroids[asteroid_number].x > 0:
                        if stage.collide(right_asteroids[asteroid_number].x + 1,
                                         right_asteroids[asteroid_number].y + 1,
                                         right_asteroids[asteroid_number].x + 15,
                                         right_asteroids[asteroid_number].y + 15,
                                         lasers[laser_number].x + 3,
                                         lasers[laser_number].y + 3,
                                         lasers[laser_number].x + 13,
                                         lasers[laser_number].y + 13):
                            right_asteroids[asteroid_number].move(constants.OFF_SCREEN_
↪X,
                                                                  constants.OFF_SCREEN_
↪Y)
                            lasers[laser_number].move(constants.OFF_SCREEN_X,
                                                      constants.OFF_SCREEN_Y)
                            sound.stop()
                            sound.play(impact_sound)
                            reset_right_asteroid()
                            asteroid_counter = asteroid_counter + 1

        # This detects if any lasers hit asteroids heading up
        for laser_number in range(len(lasers)):
            if lasers[laser_number].x > 0:
                for asteroid_number in range(len(bottom_asteroids)):
                    if bottom_asteroids[asteroid_number].x > 0:
                        if stage.collide(bottom_asteroids[asteroid_number].x + 1,
                                         bottom_asteroids[asteroid_number].y + 1,
                                         bottom_asteroids[asteroid_number].x + 15,
                                         bottom_asteroids[asteroid_number].y + 15,
                                         lasers[laser_number].x + 3,
                                         lasers[laser_number].y + 3,
                                         lasers[laser_number].x + 13,
                                         lasers[laser_number].y + 13):
                            bottom_asteroids[asteroid_number].move(constants.OFF_SCREEN_
↪X,
```

(continues on next page)

(continued from previous page)

```
                                                                    constants.OFF_SCREEN_
↪Y)
                        lasers[laser_number].move(constants.OFF_SCREEN_X,
                                                  constants.OFF_SCREEN_Y)
                        sound.stop()
                        sound.play(impact_sound)
                        reset_bottom_asteroid()
                        asteroid_counter = asteroid_counter + 1
```

If you did everything correct you should now be able to fire three different types of lasers and have them destroy asteroids.

4.8 Asteroid Collisions

Now that the most important elements of the game are working, we now need to program how the game will end. The game is supposed to end when an asteroid hit box collides with the spaceship hit box. There are however two things we need to worry about first. There are two main objectives to the game: to destroy as many lasers as you can, and to survive for as long as you can. In the lasers section, I showed you how to keep track of the number of asteroids destroyed. To calculate how long the player survived, all we need is the start time. Initialize a variable outside the game loop for the start time. Have this variable use the time.time(*number of seconds to sleep*) function from the time module. This variable will then equal the exact epoch time the player entered the game scene at. In case you do not understand what epoch time is, I explained it on the game over page of the menus section.

```
# This variable records the time the game scene launched
start_time = time.time()
```

Now we can begin working on how the game ends. The game ends when an asteroid touches the player's spaceship, so we will need more collision detection just like I have done for the ammo packs and lasers. Create four for loops (one for each asteroid list respectively) in the game loop, and have them itterate through their respective asteroid list to see if the asteroid hit box has overlapped with the spaceship hit box. If an asteroid hits the spaceship, play the crash sound. You must then use the time.sleep() function to freeze the game in place for four seconds. This will aid in a seemless transition from your game scene to your game over scene. After the pause, be sure to put a sound.stop() to stop any sounds still playing before transitioning to the game over scene. Lastly, call the game over scene, and pass it your asteroids destroyed variable and your start time variable.

```
    # This detects a collision between the ship and asteroids going right
    for asteroid_number in range(len(left_asteroids)):
        if left_asteroids[asteroid_number].x > 0:
            if stage.collide(left_asteroids[asteroid_number].x + 1,
                             left_asteroids[asteroid_number].y + 1,
                             left_asteroids[asteroid_number].x + 15,
                             left_asteroids[asteroid_number].y + 15,
                             ship.x + 3, ship.y + 3, ship.x + 12, ship.y + 12):
                sound.stop()
                sound.play(crash_sound)
                time.sleep(4.0)
                sound.stop()
                game_over_scene(asteroid_counter, start_time)

    # This detects a collision between the ship and asteroids going down
    for asteroid_number in range(len(top_asteroids)):
        if top_asteroids[asteroid_number].x > 0:
            if stage.collide(top_asteroids[asteroid_number].x + 1,
```

(continues on next page)

(continued from previous page)

```
                                top_asteroids[asteroid_number].y + 1,
                                top_asteroids[asteroid_number].x + 15,
                                top_asteroids[asteroid_number].y + 15,
                                ship.x + 3, ship.y + 3, ship.x + 12, ship.y + 12):
                    sound.stop()
                    sound.play(crash_sound)
                    time.sleep(4.0)
                    sound.stop()
                    game_over_scene(asteroid_counter, start_time)

        # This detects a collision between the ship and asteroids going left
        for asteroid_number in range(len(right_asteroids)):
            if right_asteroids[asteroid_number].x > 0:
                if stage.collide(right_asteroids[asteroid_number].x + 1,
                                right_asteroids[asteroid_number].y + 1,
                                right_asteroids[asteroid_number].x + 15,
                                right_asteroids[asteroid_number].y + 15,
                                ship.x + 3, ship.y + 3, ship.x + 12, ship.y + 12):
                    sound.stop()
                    sound.play(crash_sound)
                    time.sleep(4.0)
                    sound.stop()
                    game_over_scene(asteroid_counter, start_time)

        # This detects a collision between the ship and asteroids going up
        for asteroid_number in range(len(bottom_asteroids)):
            if bottom_asteroids[asteroid_number].x > 0:
                if stage.collide(bottom_asteroids[asteroid_number].x + 1,
                                bottom_asteroids[asteroid_number].y + 1,
                                bottom_asteroids[asteroid_number].x + 15,
                                bottom_asteroids[asteroid_number].y + 15,
                                ship.x + 3, ship.y + 3, ship.x + 12, ship.y + 12):
                    sound.stop()
                    sound.play(crash_sound)
                    time.sleep(4.0)
                    sound.stop()
                    game_over_scene(asteroid_counter, start_time)
```

Assuming you have followed all the steps correctly, you should now have a fully functional game scene.

CHAPTER 5

Menu System

The following steps are how to get the different scenes apart from the game scene working. See the game section to see how to get the game scene working.

5.1 MT Splash Scene

Asteroid Dodger has two studios to its name: it was made by Snakob Studios, and distributed by MT Game Studios. Each has a splash scene before the actual menu shows up. The MT Game Studios splash scene shows up first. The first thing you must do is establish an image bank for this scene. You must use the mt game studios image bank. Then set the background of the screen to be the first image in the image bank. In this case, the background will be plain white.

```python
# 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)
```

You must then make sure the MT Game Studios logo displays. You can do this to set individual tiles from your image bank to different spots on the screen. If the pieces are fitted together properly, the image on screen will be the MT Game Studios logo.

```python
# used this program to split the image 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
```

(continues on next page)

(continued from previous page)

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

You must also display the title of the studio on screen. You must first create a list to store all your text. Then create the text by establishing its font, size, palette (mt game studios palette in your constants file), and its location. Append your text to the text list.

```
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)
```

The next thing you will want to do is to get the sound ready. Open the coin sound file in the MT splash scene, then define your sound variable. Be sure that your sound is not muted before you play the sound file.

```
# get sound ready
# Use 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)
```

For any of your text or background tiles to appear they must be painted on the screen. You can do this by modifying your initial game layer. be sure your text is in front of your background. You will also need to set the frame rate to 60, and render your text and background on screen.

```
# 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
```

(continues on next page)

(continued from previous page)

```
game.render_block()
```

In your game loop, have a time.sleep() for three seconds to keep the MT Game Studios scene up for three seconds. After three seconds, call the next splash scene to swap scenes.

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

    # update game logic

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

    # redraw sprite list
```

You should now have a fully functional MT Game Studios splash scene.

5.2 Snakob Splash Scene

The second company to make Asteroid Dodger is Snakob Studios. They are the main creators of the game, and will be the second splash scene to appear. Like the other splash scene, you must first set an image bank. The image bank this time will be the splash scene image bank. Then set the background of the screen to be the first image in the image bank. In this case, the background will be plain black.

```
# Splash screen image bank
image_bank_3 = stage.Bank.from_bmp16("splash_scene.bmp")

# sets the background to image 1 in the bank
background = stage.Grid(image_bank_3, constants.SCREEN_GRID_X, \
                        constants.SCREEN_GRID_Y)
```

You must also display the Snakob Studios title text. Similar to the MT splash scene, create a list to hold all your text. Create the text with proper dimensions, palette, and coordinates, then append it to the list.

```
# This list holds the text
text = []

# The Snakob Studios text
text1 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text1.move(25, 20)
text1.text("Snakob Studios")
text.append(text1)
```

For the Snakob Studios logo, instead of setting individual background tiles, they will be interlinked sprites to form in an image. Create a list to hold all the sprites. Then append each piece of the Snakob logo to the list. Be sure it is in its proper coordinates to interlock correctly with the other sprites.

```
# This list holds the sprites for snakob
sprites =[]

# These sprites connect to create snakob
```

(continues on next page)

(continued from previous page)

```python
bottom_left = stage.Sprite(image_bank_3, 13, 65, 100)
sprites.append(bottom_left)

mid_tail = stage.Sprite(image_bank_3, 14, 81, 100)
sprites.append(mid_tail)

mid_left_neck = stage.Sprite(image_bank_3, 9, 65, 84)
sprites.append(mid_left_neck)

mid_right_neck = stage.Sprite(image_bank_3, 10, 81, 84)
sprites.append(mid_right_neck)

mid_left_side_face = stage.Sprite(image_bank_3, 5, 65, 68)
sprites.append(mid_left_side_face)

right_eye = stage.Sprite(image_bank_3, 6, 81, 68)
sprites.append(right_eye)

left_side_face = stage.Sprite(image_bank_3, 4, 49, 68)
sprites.append(left_side_face)

end_of_tongue = stage.Sprite(image_bank_3, 7, 97, 68)
sprites.append(end_of_tongue)

top_of_left_eye = stage.Sprite(image_bank_3, 1, 65, 52)
sprites.append(top_of_left_eye)

end_of_tail = stage.Sprite(image_bank_3, 3, 97, 52)
sprites.append(end_of_tail)

left_eyebrow = stage.Sprite(image_bank_3, 2, 81, 52)
sprites.append(left_eyebrow)

bulky_part_tail = stage.Sprite(image_bank_3, 8, 49, 84)
sprites.append(bulky_part_tail)

lower_tail = stage.Sprite(image_bank_3, 12, 49, 100)
sprites.append(lower_tail)

extra_pixels = stage.Sprite(image_bank_3, 15, 96, 100)
sprites.append(extra_pixels)
```

You will also need to get Snakob's hissing sound working. To do this, open the hiss sound file in the snakob splash scene, then define your sound variable. Be sure that your sound is not muted before you play the sound file.

```python
# Get sounds ready
hiss_sound = open("hiss.wav", 'rb')
sound = ugame.audio
sound.stop()
sound.mute(False)
sound.play(hiss_sound)
```

The next thing to do is to make sure all your sprites, text, and background tiles are set properly. Similar to what you did in the MT splash scene, paint them on the proper layers, set the frame rate to 60, and render the initial position of the sprites and text.

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

The final thing to do is make a timer to swap out of the Snakob splash scene. You can do this by adding a time.sleep() for three seconds in your game loop, then call the menu scene. You will also need to make sure your sprites remain rendered on screen.

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

    # update game logic
    time.sleep(3.0)
    menu_scene()

    # redraw sprite list
    game.render_sprites(sprites)
    game.tick()
```

You should now have a working Snakob Studios splash scene.

5.3 Menu Scene

The menu scene is how you get into the main game or look at the rules. Like all scenes previous, the first thing that needs to be done is to set the image bank, which in this case is the gamesprites image bank. You must then use a for loop to itterate through each tile and place a plain blank tile as the background.

```
# The image bank for the game
image_bank_1 = stage.Bank.from_bmp16("gamesprite.bmp")

# sets the background to image 1 in the bank
background = stage.Grid(image_bank_1, 10, 8)
for x_location in range(constants.SCREEN_GRID_X):
    for y_location in range(constants.SCREEN_GRID_X):
        background.tile(x_location, y_location, 1)
```

You will also need to set up button detection for the start and select buttons, as they will be how you swap scenes later on.

```
# Reupdating keys
keys = 0

# Buttons to keep state information on
start_button = constants.button_state["button_up"]
select_button = constants.button_state["button_up"]
```

There are also three instances of text: the title, and two instructional texts, one reading 'press start to begin' and the other saying 'press select for rules'. Like previous scenes, you must create a list to contain the text. Then create the text with the appropriate dimensions, palette (mt game studios palette), and coordinates. Append each text to the list.

```python
# The list that holds all the text
text = []

text1 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text1.move(20, 20)
text1.text("Asteroid Dodger")
text.append(text1)

text2 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text2.move(15, 80)
text2.text("'Start' to begin")
text.append(text2)

text3 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text3.move(8, 100)
text3.text("'Select' for rules")
text.append(text3)
```

You will then need to make a larger model of the game scene spaceship show up on this scene. To do this, create a list to hold your sprites, then append the left and right sides of the spaceship to the list. Line up the coordinates so that the two sprites connect to create a larger scale spaceship.

```python
# This list keeps all the sprites
sprites = []

# Adding the left and right of the ship to the sprite list
ship_left = stage.Sprite(image_bank_1, 11, 64, 45)
sprites.append(ship_left)
ship_right = stage.Sprite(image_bank_1, 12, 80, 45)
sprites.append(ship_right)
```

As it is for every other scene, you must once again set your frame rate to 60, paint the sprite, text and background layers and render the initial scene screen.

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

Next you must get the start and select buttons working. In your game loop, set keys to detect if either of the two buttons are pressed. Use an if statement to detect whether the start button is pressed or not. If the start button is pressed, swap to the game scene by calling the game scene function.

```python
    # get user input
    keys = ugame.buttons.get_pressed()

    # update game logic
    if keys & ugame.K_START != 0:  # Start button
        keys = 0
        ugame.K_START = 0
```

(continues on next page)

(continued from previous page)

```
            game_scene()
            break
```

Now use another if statement to detect if the select button has been pressed. If the select button is pressed, swap to the rules scene by calling the rules scene function.

```
        if keys & ugame.K_SELECT != 0:  # Select button
            keys = 0
            ugame.K_SELECT = 0
            rules_scene()
            break
```

Also insert a render into your game loop to ensure that the sprites on screen stay rendered.

```
        # redraw sprite list
        game.render_sprites(sprites)
        game.tick()
```

You should now have a properly functioning menu scene.

5.4 Rules Scene

In this scene you can see the rules on how to play Asteroid Dodger. The first thing you will need to do is set the image bank to be the gamesprites image bank. To set your background, use a for loop to itterate through all the on screen tiles and place a black background.

```
    # The image bank for the game
    image_bank_1 = stage.Bank.from_bmp16("gamesprite.bmp")

    # sets the background to image 1 in the bank
    background = stage.Grid(image_bank_1, 10, 8)
    for x_location in range(constants.SCREEN_GRID_X):
        for y_location in range(constants.SCREEN_GRID_X):
            background.tile(x_location, y_location, 1)
```

You will need to display all your rules text and a title for your scene. You will need to create a list to hold the text you create. Next create the text with the desired dimensions, palette (mt game studios palette), and coordinates. Append your text to the list. If you run out of room in a particular line to display it on screen, be sure that the next line is closer to the previous one to make it look like a continuation rather than a new rule.

```
    # The list that holds all the text
    text = []

    # The following text displays all the game rules
    text1 = stage.Text(width=45, height=30, font=None,
                       palette=constants.SCORE_PALETTE, buffer=None)
    text1.move(50, 5)
    text1.text("Rules")
    text.append(text1)

    text2 = stage.Text(width=45, height=30, font=None,
                       palette=constants.SCORE_PALETTE, buffer=None)
    text2.move(0, 20)
    text2.text("Use D-Pad to move")
```

(continues on next page)

(continued from previous page)

```
text.append(text2)

text3 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text3.move(0, 35)
text3.text("Touch ammo to pickup")
text.append(text3)

text4 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text4.move(0, 50)
text4.text("Release 'A' to shoot")
text.append(text4)

text5 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text5.move(0, 65)
text5.text("Game ends when")
text.append(text5)

text6 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text6.move(0, 75)
text6.text("hit by asteroid")
text.append(text6)

text7 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text7.move(15, 105)
text7.text("'Start' to begin")
text.append(text7)
```

The next thing you will need to do is to be sure that your frame rate is 60, and that your text and background have been properly layered and rendered on screen.

```
# 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()
```

The last thing you will need to do is to make sure you can get to your game scene. In your game loop, set keys to detect whether or not a button has been pressed. Use an if statement to detect if the start button has been pressed. If the start button has been pressed, swap to the game scene by calling the game scene function.

```
    # get user input
    keys = ugame.buttons.get_pressed()

    # update game logic
    if keys & ugame.K_START != 0:  # Start button
        keys = 0
        ugame.K_START = 0
        game_scene()
        pass
```

If these instructions were followed correctly, your rules scene should now work.

5.5 Game Over Scene

The game over scene is the scene that the game scene will swap to once the player's spaceship has collided with an asteroid. The first thing you will need to do is set the image bank to be the gamesprites image bank. To set your background, use a for loop to itterate through all the on screen tiles and place a black background.

```
# The image bank for the game
image_bank_1 = stage.Bank.from_bmp16("gamesprite.bmp")

# sets the background to image 1 in the bank
background = stage.Grid(image_bank_1, 10, 8)
for x_location in range(constants.SCREEN_GRID_X):
    for y_location in range(constants.SCREEN_GRID_X):
        background.tile(x_location, y_location, 1)
```

Before I continue explaining how to place more items on screen, we should stop and make sure that we have all the information we need to display. When the game scene swapped to the game over scene, it passed two variables: one with the amount of asteroids the player destroyed and the other being the exact epoch time that the game scene was activated. Epoch time is the exat amount of time in seconds that has passed since January 1st, 1970. We can use this to calculate the amount of time the player survived by subtracting the game start epoch time from the epoch time the game over scene activated, subtracting an additional 4 from the amount of seconds the game scene stopped after the asteroid collided witht the spaceship. Your answer will be how long the player survived in seconds.

```
# Converting epoch time to seconds
seconds_survived = time.time() - time_start - 4
```

The next thing to display on screen is the text: the game over text, the number of asteroids destroyed text, and the time survived text. You will need to create a list to hold the text you create. Next create the text with the desired dimensions, palette (mt game studios palette), and coordinates. Append your text to the list. For the amount of asteroids destroyed text, display the asteroids destroyed variable passed into the game over scene function by the game scene function. To display the time the player survived in seconds, display the time survived variable that you performed the caluclations with in the step above.

```
# The list that holds all the text
text = []

# The game over text
text1 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text1.move(50, 5)
text1.text("Game Over")
text.append(text1)

# This text displays how many asteroids the user destroyed
text2 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
text2.move(5, 40)
text2.text("Asteroids Shot: {0}".format(asteroids_destroyed))
text.append(text2)

# This text displays how many asteroids the user destroyed
text3 = stage.Text(width=45, height=30, font=None,
                   palette=constants.SCORE_PALETTE, buffer=None)
```

(continues on next page)

(continued from previous page)

```
text3.move(5, 80)
text3.text("Alive: {0} seconds".format(seconds_survived))
text.append(text3)
```

The last thing to do is to be sure that your frame rate is 60, and that your text and background have been properly layered and rendered on screen.

```
# 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()
```

Your game over scene should now be working.