Space Aliens - CircuitPython Game

Mr. Coxall

Jan 16, 2020

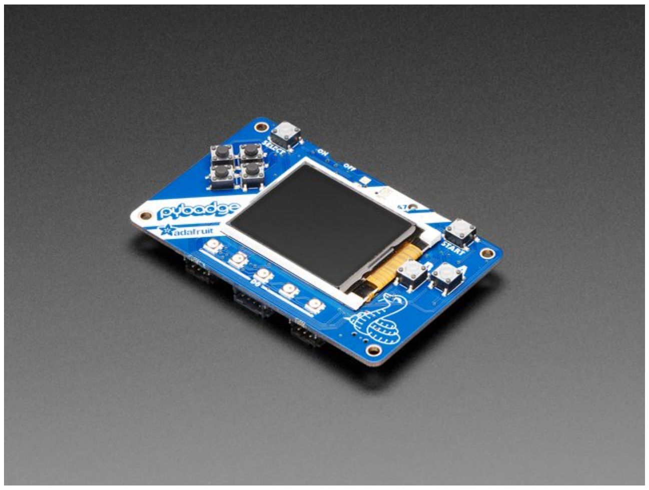
Contents

In this project I made an old school style video game for the Adafruit PyBadge. To recreate it you will need to use an CircuitPython and the stage library to create the game which I called Egg Collector. 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 anyone with a basic grade 11 knowledge of python programming can recreate it.

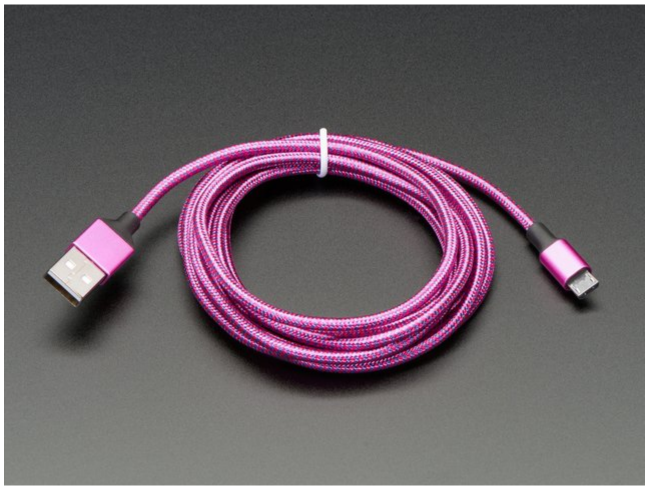
Parts

You will need the following items:



Adafruit PyBadge for MakeCode Arcade, CircuitPython or Arduino

PRODUCT ID: 4200

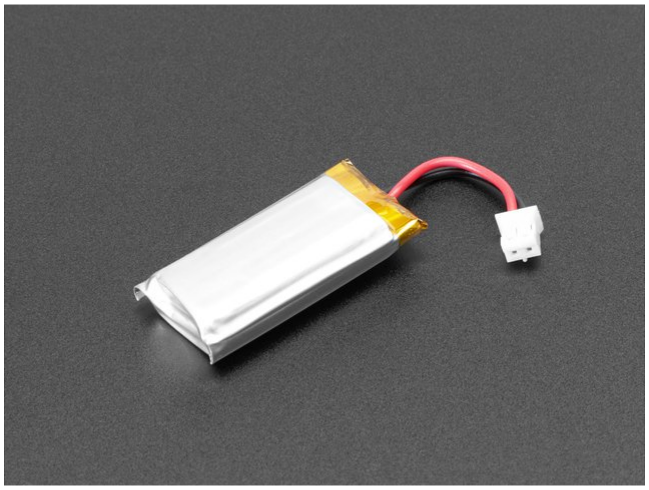


Pink and Purple Braided USB A to Micro B Cable - 2 meter long

PRODUCT ID: 4148

So you can move your CircuitPython code onto the PyBadge.

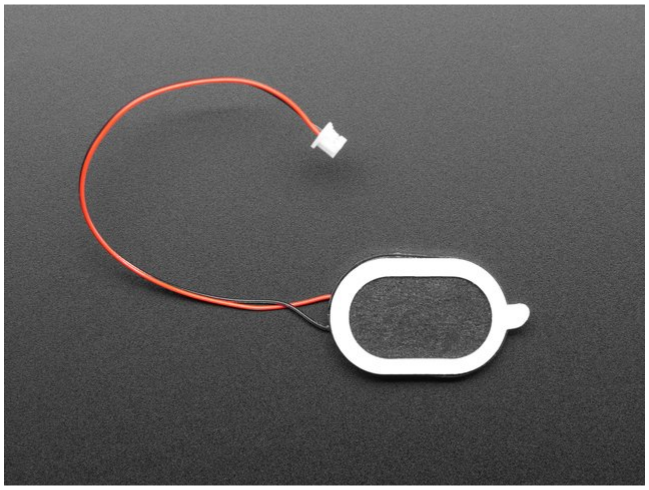
You might also want:



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

PRODUCT ID: 3898

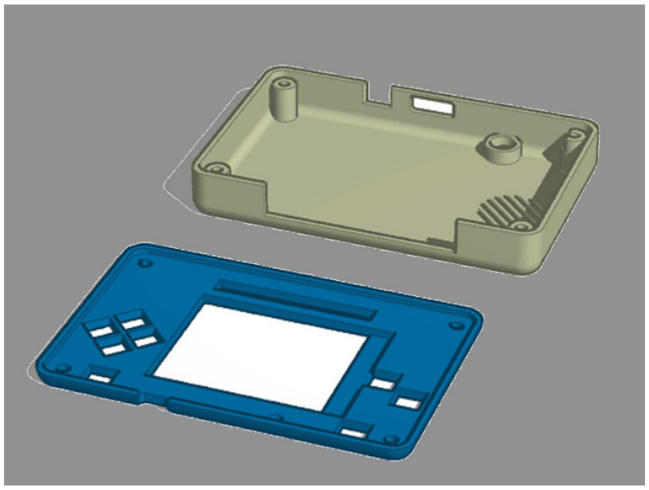
So that you can play the game without having it attached to a computer with a USB cable.



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 Used Mr Coxall's design <https://learn.adafruit.com/pybadge-case/>‘_.

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.

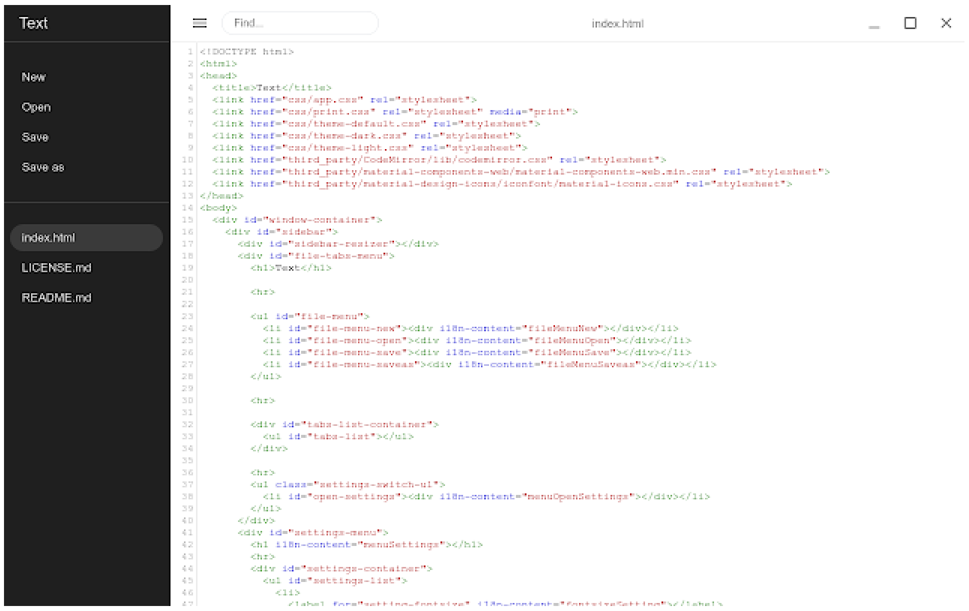
Just a reminder, if you are having any problems loading CircuitPython onto your PyBadge, ensure that you are using a USB cable that not only provides power, but also provides a data link. Many USB cables you buy are only for charging, not transfering data as well. Once the CircuitPython is all loaded,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

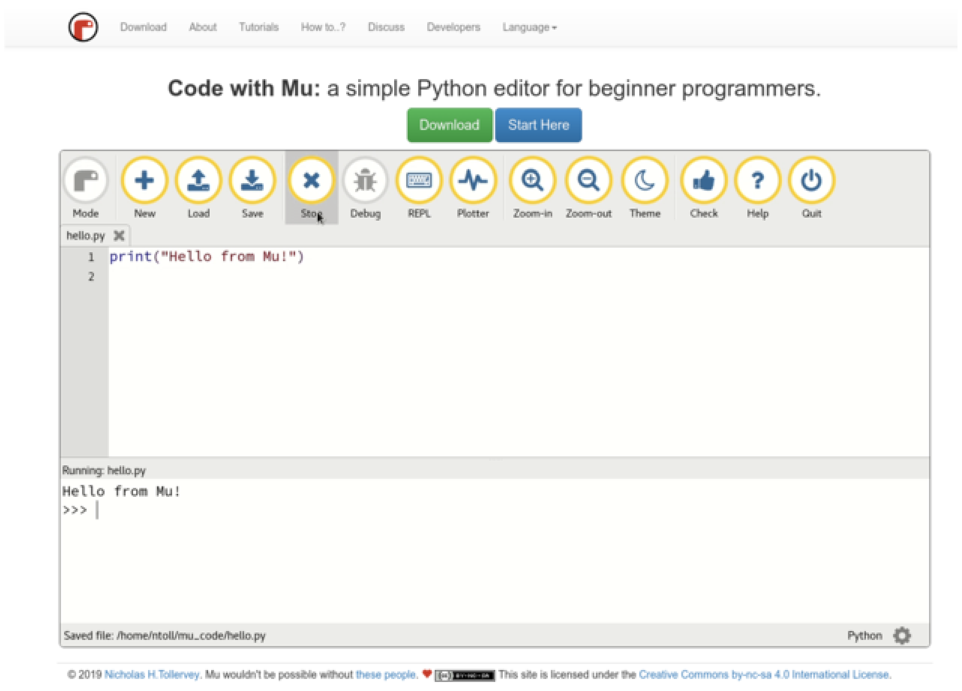
If you are working on a Chromebook, the easiest way to start coding is to just use the built in Text app. As soon as you open or save a file with a `*.py` extension, it will know it is Python code and automatically start syntax highlighting.



```
<!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.


Download
About
Tutorials
How to..?
Discuss
Developers
Language
Code with Mu: a simple Python editor for beginner programmers.
Download
Start Here
Mode
New
Load
Save
Stop
Debug
REPL
Plotter
Zoom-in
Zoom-out
Theme
Check
Help
Quit
hello.py
Running: hello.py
Saved file: /home/ntoll/mu_code/hello.py
Python
© 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.

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

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

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:

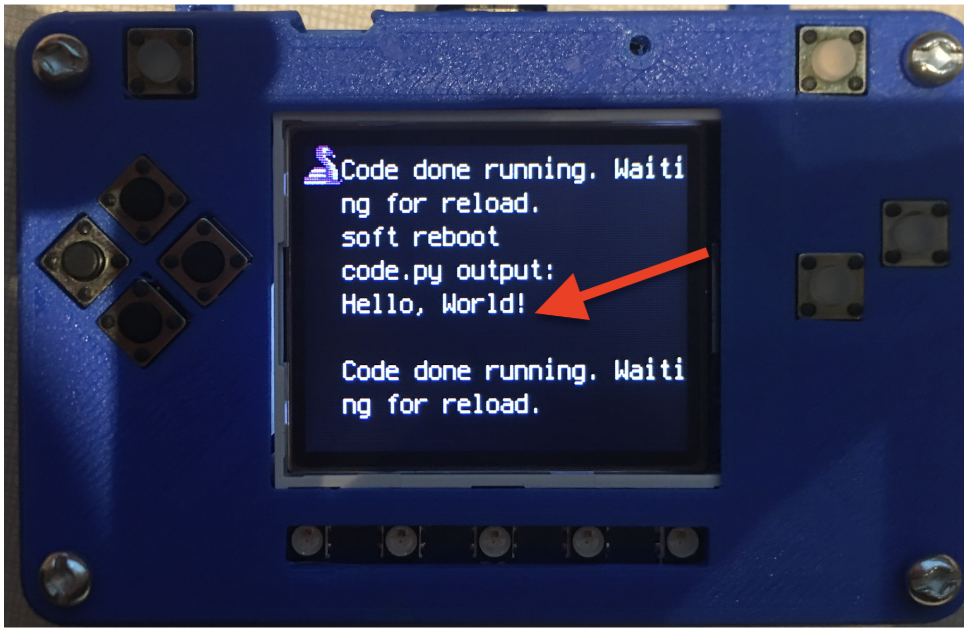
```python
print("Hello, World!")
```

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. Waiti
ng for reload.
soft reboot
code.py output:
Hello, World!

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

Fig. 3: Hello, World! program on PyBadge

(continued from previous page)

```python
def 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

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 Egg Collector

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:

- Egg Collector Assets Folder: https://github.com/Douglass-Jeffrey/ICS3U-2019-Group22

Please download the assets and place them on the PyBadge, in the root directory. Your previous "Hello, World!" program should restart and run again each time you load a new file onto the PyBadge, hopefully with no errors once more.

Assets from other people can be found here.

CHAPTER 4

Game

This section contains the logic you will need to create your version of the Egg Collector Game. Other functions that the game requires will be explained in their respective sections.

Chicken(s)

In Egg Collector the main playable character is a chicken who moves to collect the egg and avoid the bombs descending the screen. The sprite list I made contains a left facing chicken and a right one, and if you wish to make your chicken sprite turn one must swap the chickens every movement and run checks on both of them while they are on screen to determine if a bomb or egg touches either. So in order to make the chickens we must first generate sprites sprite in the game scene outside of the game loop. Always remember to render the layers and set which appear above the others in your game scene with your background always at the back. I set my chicken sprites as the foremost layer at the end of the scene outside of the game loop and created it like this :

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

# Created by: Douglass Jeffrey
# Created on: Dec 2019
# This file is an example of how to create the chicken sprites


def game_scene():

# list to hold chicken sprites
    chickens = []

    # create right chicken sprite
    chickenR = stage.Sprite(image_bank_2, 1, 80, 128 - constants.SPRITE_SIZE)
    chickens.insert(0, chickenR)  # insert at top of sprite list

    # create left chicken sprite
    chickenL = stage.Sprite(image_bank_2, 2, constants.OFF_SCREEN_X,
                            constants.OFF_SCREEN_Y)
    chickens.append(chickenL)

```

(continues on next page)

(continued from previous page)

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

I made sure to append it to a list and refresh it as well as the bomb and egg sprites 60 times per second inside of the game loop.

Because the chicken has to save the falling eggs, I allow it to move left and right based on user input. I also chose to allow the chicken to move past one side of the screen and appear at the other but having something like that in your game is up to you. To move the chicken I had the user press the d-pad pertaining to the direction they wish to travel in. To do this I set up an if statement using the button states that were declared in our constants file. The if statement first checks if the chicken is touching an edge of the screen and moves it if it is not. If you dont want to include this piece of code, moving the chicken can be as simple as: if X button pressed: chickenR.move(chickenR.x + chicken_speed, chickenR.y). An example of my code for moving the chicken can be found here (I also added a speed button) :

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

# Created by: Douglass Jeffrey
# Created on: Dec 2019
# This file is an example of how to create the chicken sprites


def game_scene():

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

        # sets button states
        if keys & ugame.K_X != 0:  # A button
            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 right D-Pad is pressed
        if keys & ugame.K_RIGHT != 0:
            # if chicken moves off right screen, move it back
            if chickenR.x > constants.SCREEN_X - constants.SPRITE_SIZE:
                chickenR.x = 0
            # else move chicken right
            else:
                # if chickenL is onscreen and right d-pad is pressed
                    # replace chickenL with chickenR
                if chickenL.x > 0:
                    chickenR.move(chickenL.x, chickenL.y)
                    chickenL.move(constants.OFF_SCREEN_X,
                                  constants.OFF_SCREEN_Y)
                    # once chicken is faced in direction of pressed d-pad
                    #    move chicken that way
```

(continues on next page)

(continued from previous page)

```
                        chickenR.move(chickenR.x + chicken_speed, chickenR.y)
                    else:
                        # if chickenL isnt onscreen and right d-pad is
                        #    pressed move chickenR
                        chickenR.move(chickenR.x + chicken_speed, chickenR.y)

            # if left D-Pad is pressed
            if keys & ugame.K_LEFT != 0:
                # if chicken moves off left screen, move it back
                if chickenL.x < 0 and chickenL.y != constants.OFF_SCREEN_Y:
                    chickenL.x = constants.SCREEN_X
                # else move chicken left
                else:
                    # if chickenR is onscreen and left d-pad is pressed replace
                    #    chickenL with chickenL
                    if chickenR.x > 0:
                        chickenL.move(chickenR.x, chickenR.y)
                        chickenR.move(constants.OFF_SCREEN_X,
                                      constants.OFF_SCREEN_Y)
                        # once chicken is faced in direction of pressed d-pad
                        #    move chicken that way
                        chickenL.move(chickenL.x - chicken_speed, chickenL.y)
                    else:
                        # if chickenR isnt onscreen and left d-pad is
                        #    pressed move chickenL
                        chickenL.move(chickenL.x - chicken_speed, chickenL.y)


            # if A Button (speed) is pressed
            if a_button == constants.button_state["button_still_pressed"]:
                chicken_speed += 1
                # increase speed at which chicken moves
                if chicken_speed > 3:
                    chicken_speed = 3

            # if A Button (speed) is not pressed
            if a_button == constants.button_state["button_up"]:
                chicken_speed = 2


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

Eggs and Bombs

In Egg Collector, both eggs and bombs rain down from the sky as the chicken (your player character) attempts to catch them by moving along the ground and positioning itself underneath them. In my version of the game, catching an egg awards the player with one point, and missing one deducts two. Missing a bomb awards no points but catching one ends the game. First and foremost, in order to make the eggs rain down from the sky, an extra function is required for each to reposition them above the screen once they finish moving across the screen. These functions are found here:

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

# Created by: Douglass Jeffrey
# Created on: Dec 2019
# This file is the contains the egg and bomb moving functions for Egg collector

```

(continues on next page)

(continued from previous page)

```python

def game_scene():
    # Function to make eggs reappear at the top of the screen
    def show_egg():
        for egg_number in range(len(eggs)):
            if eggs[egg_number].x < 0:  # meaning it is off the screen,
                eggs[egg_number].move(random.randint(0 + constants.SPRITE_SIZE,
                                                     constants.SCREEN_X -
                                                     constants.SPRITE_SIZE),
                                      constants.OFF_TOP_SCREEN)
                break

    # Function to make bombs reappear at the top of the screen
    def show_bomb():
        for bomb_number in range(len(bombs)):
            if bombs[bomb_number].x < 0:  # meaning it is off the screen
                bombs[bomb_number].move(random.randint
                                        (0 + constants.SPRITE_SIZE,
                                        constants.SCREEN_X -
                                        constants.SPRITE_SIZE),
                                        constants.OFF_TOP_SCREEN
                                        - random.randint(0, 50))
                break


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

should be placed inside of the game scene but before the game loops.

Next we need to generate the eggs and bombs and place them in their respective lists by using for loops like this

these loops simply append the amount of eggs you choose to be in your game into a list and places them off screen.

In order to determine if the eggs are touching the bottom of the screen, one must make a loop in the game loop to check whether or not they are in contact with the screen Y value (bottom of screen). This loop can be found here:

I added many other things like increase in bomb and egg speed and some sounds to signify when they touch the bottom of the screen but that design choice is entirely up to the creator. From here making the eggs and bombs rain down is simple enough, I added an else to the if statement which determines if the bombs are touching the ground to allow them to continue moving at a specific speed if they are not touching the ground.

The final piece of logic determines if the eggs and bombs touch the chicken. What happens when they touch is a decision the creator must make but the main part of the logic remains the same nonetheless. To determine when the eggs and bombs touch the chicken, we will be defining the area of each sprite onscreen then using an if statement and stage.collide to determine if any of the 16X16 sprites overlap eachother at any given moment. Here is an example from my version of the game :

Score

The score system in egg collector relies upon catching the eggs in my version of the game. This part of the code is honestly your choice whether or not you wish to include it or how you wish to include it. Firstly I set the score variable to 0 at the top of my function. Then set up where the score text would appear in my game, chose the pallette its text would use and formatted it. I didnt forget to set its layer above the background to allow it to actually show up, and I remembered to render it along with the chicken eggs and bombs in the game loop. I set the score up so that whenever an egg is caught it would increase by one and whenever an egg was lost it would decrease by 2. This is done in the collision detection loops. Here is how I set up score in my version of the game:

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

# Created by: Douglass Jeffrey
# Created on: Dec 2019
# This file is an example of score in egg collector


def game_scene():

    # game score
    score = 0

    # add text at top of screen for score
    score_text = stage.Text(width=29, height=14, font=None,
                            palette=constants.SCORE_PALETTE, buffer=None)
    score_text.clear()
    score_text.cursor(0, 0)
    score_text.move(1, 1)
    score_text.text("Score: {0}".format(score))


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

If you have made it this far then good job! there is only a bit of work left to do.

4.1 Background

The background for Egg Collector is quite complex and includes many sprites. When I refer to the background I refer to everything that happens behind the actual game (including the large tree). Although the complexity varies, one thing remains true in all scenes except the MT games splash screen; they all utilise the first image in the egg collector bank as a canvas for the sprites to be drawn on. To use the Egg Collector image bank, I made sure to set it to a variable in each scene so I could pick out whichever 16X16 sprite I wanted from it. Once I got this out of the way, I made sure to set my background to image 0 in the bank to allow the blue sky to appear. After this I created lists for the different parts of the tree and started appending sprites to them to draw my large tree on screen. I created 3 arrays each for different parts of the background and had them show up in a specific order by setting the layers in my preferred way.

Grass

To make the grass on the ground I created a for loop to create grass sprites at intervals of 16 from 0 (the leftmost point on the screen) to 160 (the rightmost point)

Tree Trunk

To create the tree trunk, I created the right and left bases facing eachother in the center of the screen. I then created the middle of the trunk on top and in the middle of the two base sprites before adding two branch sprites facing each other above the middle of the trunk sprite and at the saem x values as he two base sprites.

Foliage

To create the leaves on the trees I experimented with many different sprites until i found something i thought looked decent. The trick I used was to not make the leaf sprites completely symmetrical.

If you want to use my background instead of experimenting yourself, it can be found here:

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

```

(continues on next page)

(continued from previous page)

```python
# Created by: Douglass Jeffrey
# Created on: Dec 2019
# This file is an example of how to create the chicken sprites


def game_scene():

    image_bank_2 = stage.Bank.from_bmp16("egg_collector_image_bank_test.bmp")

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


    # list to create plants at the bottom of the screen
    plants = []
    # procedurally generating grass
    for grass_number in range(0, 10):
        a_single_grass = stage.Sprite(image_bank_2, 5, constants.GRASS_POINT
                                      + increaser, 128 - 16)
        plants.append(a_single_grass)
        increaser += 16

    # list to hold all trunk sprites
    trunk = []

    trunkL = stage.Sprite(image_bank_2, 6, constants.SPRITE_SIZE * 4, 112)
    trunk.append(trunkL)

    trunkR = stage.Sprite(image_bank_2, 7, constants.SPRITE_SIZE * 5, 112)
    trunk.append(trunkR)

    trunkM = stage.Sprite(image_bank_2, 8, 72, 96)
    trunk.append(trunkM)

    trunkM2 = stage.Sprite(image_bank_2, 8, 72, 80)
    trunk.append(trunkM2)

    trunk_branchL = stage.Sprite(image_bank_2, 9,
                                 constants.SPRITE_SIZE * 4, 64)
    trunk.append(trunk_branchL)

    trunk_branchR = stage.Sprite(image_bank_2, 10,
                                 constants.SPRITE_SIZE * 5, 64)
    trunk.append(trunk_branchR)

    # list to hold all leaf/foliage sprites
    foliage = []

    foliageLBB = stage.Sprite(image_bank_2, 12, 60, 59)
    foliage.append(foliageLBB)

    foliageRBB = stage.Sprite(image_bank_2, 11, 84, 59)
    foliage.append(foliageRBB)

    foliageLMB = stage.Sprite(image_bank_2, 13, constants.SPRITE_SIZE * 4, 50)
    foliage.append(foliageLMB)
```

(continues on next page)

(continued from previous page)

```

    foliageRMB = stage.Sprite(image_bank_2, 13, constants.SPRITE_SIZE * 5, 50)
    foliage.append(foliageRMB)

    foliageLLB = stage.Sprite(image_bank_2, 12, constants.SPRITE_SIZE * 3, 50)
    foliage.append(foliageLLB)

    foliageRRB = stage.Sprite(image_bank_2, 11, constants.SPRITE_SIZE * 6, 50)
    foliage.append(foliageRRB)

    foliageLMM = stage.Sprite(image_bank_2, 13, constants.SPRITE_SIZE * 4, 40)
    foliage.append(foliageLMM)

    foliageRMM = stage.Sprite(image_bank_2, 13, constants.SPRITE_SIZE * 5, 40)
    foliage.append(foliageRMM)

    foliageLLM = stage.Sprite(image_bank_2, 13, constants.SPRITE_SIZE * 3, 40)
    foliage.append(foliageLLM)

    foliageRRM = stage.Sprite(image_bank_2, 13, constants.SPRITE_SIZE * 6, 40)
    foliage.append(foliageRRM)

    foliageLLLM = stage.Sprite(image_bank_2, 12, 38, 40)
    foliage.append(foliageLLLM)

    foliageRRRM = stage.Sprite(image_bank_2, 11, 104, 40)
    foliage.append(foliageRRRM)

    foliageLLLT = stage.Sprite(image_bank_2, 15, 38, 26)
    foliage.append(foliageLLLT)

    foliageRRRT = stage.Sprite(image_bank_2, 14, 104, 26)
    foliage.append(foliageRRRT)

    foliageLLMT = stage.Sprite(image_bank_2, 13, 54, 26)
    foliage.append(foliageLLMT)

    foliageRRMT = stage.Sprite(image_bank_2, 13, 88, 26)
    foliage.append(foliageRRMT)

    foliageLMMT = stage.Sprite(image_bank_2, 13, 70, 26)
    foliage.append(foliageLMMT)

    foliageRMMT = stage.Sprite(image_bank_2, 13, 72, 26)
    foliage.append(foliageRMMT)

    foliageLLTT = stage.Sprite(image_bank_2, 15, 50, 14)
    foliage.append(foliageLLTT)

    foliageRRTT = stage.Sprite(image_bank_2, 14, 91, 14)
    foliage.append(foliageRRTT)

    foliageLMTT = stage.Sprite(image_bank_2, 13, 65, 14)
    foliage.append(foliageLMTT)

    foliageRMTT = stage.Sprite(image_bank_2, 13, 75, 14)
    foliage.append(foliageRMTT)
```

(continues on next page)

(continued from previous page)

```python

    foliage_deco_1 = stage.Sprite(image_bank_2, 12, 60, 20)
    foliage.insert(0, foliage_deco_1)

    foliage_deco_2 = stage.Sprite(image_bank_2, 11, 80, 30)
    foliage.insert(1, foliage_deco_2)

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

CHAPTER 5

Menu Scene

In Egg Collector the menu scene acts as a preview to the actual game. We see the tree in the background as well as eggs falling just like the actual game. Making this scene was actually quite easy as i just used assets i had previously created in the game scene like the tree and the egg falling loop. Other than reusing those assets I added in text displaying the games name and prompting the user to start and added an if statement in the game loop which called the game scene function if the start button is pressed on the pybadge.

5.1 Splash Scene

The Egg Collector spash screen is very simple and easy to make. All I did was create text on screen displaying "Produced by" and then my name with a blue sky background.

5.2 Game Over Scene

To create Egg Collector's final Game Over scene, I displayed the final score of the game at the top of the screen by formatting the score variable into text which I then appended to a list of text which also holds the emboldened words "GAME OVER" and "PRESS SELECT". I then bound the select key of the pybadge to an if statement which returns the user to the main menu scene. For this I used the Score pallette as I prefer it with the blue background of Egg Collector.