CS: Objects in Python

Advanced Topics

Lab - Advanced Topics

Advanced Topics Lab 1

Lab 1 - Setup

We are going to create a clone of the game Flappy Bird that makes use of the topics covered in this module. The tutorial above uses a collection of Pygame objects to make the game. We are going to create a composite class that has Pygame components, a list of objects, and the lab will use two Python files.

The first thing to do is set up a generic game. Import the `pygame` and `sys` modules. Once the Pygame module has been initialized, we can create a screen, which is the game window. The clock controls the frame rate of the game.

```
# Flappy Bird clone

import pygame
import sys

pygame.init()
screen = pygame.display.set_mode((400, 720))
clock = pygame.time.Clock()
```

If you were to run the code above, the game window would appear for a split second and then disappear. We need to add some more code to make the screen persistent. The infinite while loop means the game will run until the user clicks on the icon to close the window. Pygame is continually monitoring game events (key presses, mouse clicks, etc.). Iterate through the list of events and check to see if the user is quitting Pygame (clicking the `X` icon) and then stop the game. We also need to update the display (needed to see animations) and set the frame rate of the game.

```
while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit()

  pygame.display.update()
  clock.tick(120)
```

Run the code now. You should see a black window appear with a width of 400 pixels and a height of 720 pixels. It should remain visible until you close the window.

▼ **Code**

Your code should look like this:

```python
# Flappy Bird clone

import pygame
import sys

pygame.init()
screen = pygame.display.set_mode((400, 720))
clock = pygame.time.Clock()

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit() # shutdown game completely

  pygame.display.update()
  clock.tick(120)
```

Source: Flappy Bird tutorial

Advanced Topics Lab 2

Lab 2 - Game Class

We first need to outline the `Game` class that will control this project. Be sure that you are working in the `game.py` file. Start by importing the `pygame` and `random` modules.

```
# File for Game class

import pygame
import random
```

The constructor for this class will be quite long as there are many aspects of the game. This is not the definite set of attributes, more will be added over time. The constructor expects the user to pass it file paths for images to be used for the bird, pipe, background, and the floor. `self.bird_rect` represents the rectangle surrounding the bird image. These rectangles will be used for collision detection.

```
class Game:
  def __init__(self, bird_img, pipe_img, background_img, ground_img):

    self.bird = pygame.image.load(bird_img).convert_alpha()
    self.bird_rect = self.bird.get_rect(center = (70, 180))
    self.pipe = pygame.image.load(pipe_img).convert_alpha()
    self.background = pygame.image.load(background_img).convert_alpha(

    self.ground = pygame.image.load(ground_img).convert()
```

Next, we are going to create some methods. The images for the game need to be specific sizes. The `resize_images` method will make sure that each image is the correct size. The numbers in between parentheses are called tuples, and they represent the new width and height of each image.

```
def resize_images(self):
    self.bird = pygame.transform.scale(self.bird, (51, 34))
    self.pipe = pygame.transform.scale(self.pipe, (80, 438))
    self.ground = pygame.transform.scale(self.ground, (470, 160))
    self.background = pygame.transform.scale(self.background, (400, 72
```

Next, we are going to draw the background image to the game window. Pygame uses the "surface" metaphor to describe how it works. The game window is a surface. Pygame allows you to place surfaces on top of other surfaces. When the constructor says `pygame.image.load`, it is creating a surface out of an image file. The `show_background` method takes `screen` which is the game window and adds the background image to it. Pygame uses `blit` (the documentation does not describe what this means) as a way of saying "draw". So `screen.blit(self.background, (0, 0))` means to draw the background image on the game window in the top-left corner.

```
def show_background(self, screen):
    screen.blit(self.background, (0,0))
```

We want to implement the class and methods created above. Be sure that you are in the `lab2.py` file. Import the `Game` class. Then instantiate `game` as an object of the `Game` class. Then resize the images.

```
# Flappy Bird clone

import pygame
import sys
from game import Game

pygame.init()
screen = pygame.display.set_mode((400, 720))
clock = pygame.time.Clock()

game = Game('student_folder/flappy_bird/bird.png', 'student_folder/fla

game.resize_images()
```

When it comes to adding images to the game window, this should be done inside the `while True` loop. This should be done after iterating through the game events and before the display is updated. Do not forget that `show_background` needs the `screen` parameter.

```
while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit() # shutdown game completely

  game.show_background(screen)

  pygame.display.update()
  clock.tick(120)
```

▼ **Code**

Your code in the `game.py` file should look like this:

```
# File for Game class

import pygame
import random

class Game:
  def __init__(self, bird_img, pipe_img, background_img, ground_img)

    self.bird = pygame.image.load(bird_img).convert_alpha()
    self.bird_rect = self.bird.get_rect(center = (70, 180))
    self.pipe = pygame.image.load(pipe_img).convert_alpha()
    self.background = pygame.image.load(background_img).convert_alph

    self.ground = pygame.image.load(ground_img).convert()

  def resize_images(self):
    self.bird = pygame.transform.scale(self.bird, (51, 34))
    self.pipe = pygame.transform.scale(self.pipe, (80, 438))
    self.ground = pygame.transform.scale(self.ground, (470, 160))
    self.background = pygame.transform.scale(self.background, (400,


  def show_background(self, screen):
    screen.blit(self.background, (0,0))
```

Your code in the `lab2.py` file should look like this:

```python
# Flappy Bird clone

import pygame
import sys
from game import Game

pygame.init()
screen = pygame.display.set_mode((400, 720))
clock = pygame.time.Clock()

game = Game('student_folder/flappy_bird/bird.png', 'student_folder/f

game.resize_images()

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit() # shutdown game completely

  game.show_background(screen)

  pygame.display.update()
  clock.tick(120)
```

Advanced Topics Lab 3

Lab 3 - The Ground

Drawing the Ground

Notice that the background image does not show the ground. This is because we want to give the appearance of the bird flying to the right. In reality, we are going to animate the ground (and later the pipes) to move to the left. So the ground needs to be separate from the background image.

Animating the ground means there needs to be an attribute to represent the ground's position in the window. Add the `ground_position` attribute and set its value to 0.

```
class Game:
  def __init__(self, bird_img, pipe_img, background_img, ground_img):

    self.bird = pygame.image.load(bird_img).convert_alpha()
    self.bird_rect = self.bird.get_rect(center = (70, 180))
    self.pipe = pygame.image.load(pipe_img).convert_alpha()
    self.background = pygame.image.load(background_img).convert_alpha(

    self.ground= pygame.image.load(ground_img).convert()
    self.ground_position = 0
```

To draw the ground, we are going to create the `show_ground` method for the `Game` class. The x-position of the ground should be `self.ground_position`. Because a surface is being drawn on the game window, this method needs the `screen` parameter.

```
  def show_ground(self, screen):
    screen.blit(self.ground, (self.ground_position, 650))
```

Verify that you are in the `lab3.py` file. The ground should appear on top of the background image, so make sure to put the `show_ground` method after the `show_background` method.

```
game.show_background(screen)


game.show_ground(screen)


pygame.display.update()
clock.tick(120)
```

Animating the Ground

We want the ground to move to the left, so create the method `move_ground` in the `game.py` file. Since the ground needs to move to the left, the `ground_position` attribute needs to decrease. This shifts the position of the ground image one pixel to the left every frame.

```
def move_ground(self):
  self.ground_position -= 1
```

In the `lab3.py` file, call this method after you show the ground.

```
game.show_ground(screen)
game.move_ground()
```

Improving the Animation

This animation is not very good because the ground disappears and never comes back. To fix this, we are going to change how it is drawn and then make it repeat. In the `game.py` file, change the `show_ground` method so that it draws the ground image twice. The first image will appear as normal, while the second image will be 470 pixels to the right, out of the game window.

```
def show_ground(self, screen):
  screen.blit(self.ground, (self.ground_position, 650))
  screen.blit(self.ground, (self.ground_position + 470, 650))
```

Then, update the `move_ground` method so that when `ground_position` is less than or equal to -400 its value is reset to zero. This causes the animation to repeat.

```
def move_ground(self):
  self.ground_position -= 1
  if self.ground_position <= -400:
    self.ground_position = 0
```

▼ **Code**

Your code in the `game.py` file should look like this:

```
# File for Game class

import pygame
import random

class Game:
  def __init__(self, bird_img, pipe_img, background_img, ground_img)

    self.bird = pygame.image.load(bird_img).convert_alpha()
    self.bird_rect = self.bird.get_rect(center = (70, 180))
    self.pipe = pygame.image.load(pipe_img).convert_alpha()
    self.background = pygame.image.load(background_img).convert_alph

    self.ground = pygame.image.load(ground_img).convert()
    self.ground_position = 0

  def resize_images(self):
    self.bird = pygame.transform.scale(self.bird, (51, 34))
    self.pipe = pygame.transform.scale(self.pipe, (80, 438))
    self.ground = pygame.transform.scale(self.ground, (470, 160))
    self.background = pygame.transform.scale(self.background, (400,


  def show_background(self, screen):
    screen.blit(self.background, (0,0))

  def show_ground(self, screen):
    screen.blit(self.ground, (self.ground_position, 650))
    screen.blit(self.ground, (self.ground_position + 470, 650))

  def move_ground(self):
    self.ground_position -= 1
    if self.ground_position <= -400:
      self.ground_position = 0
```

Your code in the `lab3.py` file should look like this:

```python
# Flappy Bird clone

import pygame
import sys
from game import Game

pygame.init()
screen = pygame.display.set_mode((400, 720))
clock = pygame.time.Clock()

game = Game('student_folder/flappy_bird/bird.png', 'student_folder/f

game.resize_images()

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit() # shutdown game completely

  game.show_background(screen)

  game.show_ground(screen)
  game.move_ground()

  pygame.display.update()
  clock.tick(120)
```

Advanced Topics Lab 4

Lab 4 - The Bird

Drawing the Bird

Before we can add the bird, we need to distinguish between an active game and a game that is over. Add the attribute `self.active` to the constructor.

```
class Game:
  def __init__(self, bird_img, pipe_img, background_img, ground_img):

    self.bird = pygame.image.load(bird_img).convert_alpha()
    self.bird_rect = self.bird.get_rect(center = (70, 180))
    self.pipe = pygame.image.load(pipe_img).convert_alpha()
    self.background = pygame.image.load(background_img).convert_alpha(

    self.ground = pygame.image.load(ground_img).convert()
    self.ground_position = 0
    self.active = True
```

Adding the bird to the game window is fairly similar to how the background image was added. Verify that you are in the `game.py` file and add the `show_bird` method. This method requires a parameter of `screen`, which is the game window. Remember, the bird has a rectangle (`self.bird_rect`) around it. The bird's position is the same as the rectangle's.

```
def show_bird(self, screen):
    screen.blit(self.bird, self.bird_rect)
```

The next step is to call this method in the `lab4.py` file. However, the bird should not always appear in the game. If it hits a pipe or the ground the game is over. This is why we created the `self.active` attribute. When this is `True`, then the game is being played and the bird should appear.

```
while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit() # shutdown game completely

  game.show_background(screen)

  if game.active:
    game.show_bird(screen)

  game.show_ground(screen)
  game.move_ground()

  pygame.display.update()
  clock.tick(120)
```

Adding Gravity

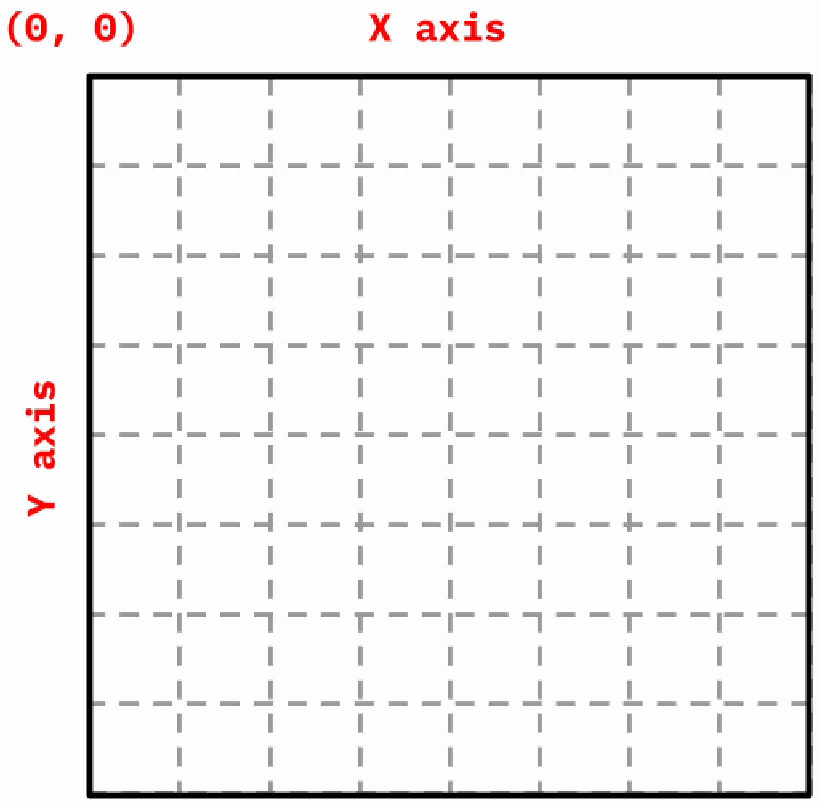
The next step is to add some gravity to the bird. We also want the bird to point down as it falls and point up when the user presses the space bar. In the `Game` class, add a gravity attribute. The larger the number, the faster the bird will fall. The `bird_movement` attribute is used to calculate the amount the bird moves. Finally, `rotated_bird` is a Pygame surface of the bird rotated to point in the correct direction.

```
    self.ground = pygame.image.load(ground_img).convert()
    self.ground_position = 0
    self.active = True
    self.gravity = 0.05
    self.bird_movement = 0
    self.rotated_bird = pygame.Surface((0, 0))
```

We need two methods to update and rotate the bird. In each frame, `gravity` is added to `bird_movement`, which is added to the position of the `bird_rect`. Remember, the position of the bird is determined by the position of `bird_rect`. If you continually rotate the same surface, Pygame will decrease the image quality. Instead, we are going to create a new bird image (`self.rotated_bird`) from the original file (`self.bird`). The `rotate_bird` method is a helper method that returns a rotated image. If you want the bird to rotate more, multiply by a number larger than 3.

▼ **Window Refresher**

Remember, in Pygame the y-coordinate increases as you move down the game window. So gravity should be a positive number.



The Window

```
def update_bird(self):
    self.bird_movement += self.gravity
    self.rotated_bird = self.rotate_bird()
    self.bird_rect.centery += self.bird_movement

def rotate_bird(self):
  new_bird = pygame.transform.rotozoom(self.bird,-self.bird_movement *

  return new_bird
```

The `show_bird` method needs to be updated so that it draws `self.rotated_bird` on the game window instead of `self.bird` which is not rotated.

```
def show_bird(self, screen):
  screen.blit(self.rotated_bird, self.bird_rect)
```

Verify that you are in the `lab4.py` file and call the `update_bird` method in the game loop. The bird should fall out of the game window while slowly rotating downward.

```
game.show_background(screen)

if game.active:
  game.show_bird(screen)
  game.update_bird()

game.show_ground(screen)
game.move_ground()
```

“Flapping” the Bird’s Wings

The user is going to press the space bar to make the bird flap its wings. The bird should move up the game window, which means bird’s position needs to get smaller. The `bird_movement` controls the the position. Making the bird “flap” its wings is a two-step process. First, set `bird_movement` to zero. This keeps the bird from falling any further. Then subtract 2.5 from `bird_movement`.

```
def flap(self):
  self.bird_movement = 0
  self.bird_movement -= 2.5
```

Pressing the space bar is an event in Pygame. So calling this method should be a part of the for loop that iterates over the game events. Write one conditional to check if the event is a key press. Write a second conditional to see if the key pressed is the space bar. If so, call the `flap` method. **Important**, you need to click on the game window so that the space bar presses are a Pygame event. If not, the key presses will be ignored by Pygame.

```
while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit() # shutdown game completely

    if event.type == pygame.KEYDOWN:
      if event.key == pygame.K_SPACE and game.active:
        game.flap()
```

▼ **Code**

Your code in the `game.py` file should look like this:

```
  # File for Game class

  import pygame
  import random

  class Game:
    def __init__(self, bird_img, pipe_img, background_img, ground_img)

      self.bird = pygame.image.load(bird_img).convert_alpha()
      self.bird_rect = self.bird.get_rect(center = (70, 180))
      self.pipe = pygame.image.load(pipe_img).convert_alpha()
      self.background = pygame.image.load(background_img).convert_alph

      self.ground= pygame.image.load(ground_img).convert()
      self.ground_position = 0
      self.active = True
      self.gravity = 0.05
      self.bird_movement = 0
      self.rotated_bird = pygame.Surface((0, 0))

    def resize_images(self):
      self.bird = pygame.transform.scale(self.bird, (51, 34))
      self.pipe = pygame.transform.scale(self.pipe, (80, 438))
      self.ground = pygame.transform.scale(self.ground, (470, 160))
      self.background = pygame.transform.scale(self.background, (400,


    def show_background(self, screen):
      screen.blit(self.background, (0,0))

    def show_ground(self, screen):
      screen.blit(self.ground, (self.ground_position, 650))
      screen.blit(self.ground, (self.ground_position + 470, 650))

    def move_ground(self):
      self.ground_position -= 1
      if self.ground_position <= -400:
        self.ground_position = 0

    def show_bird(self, screen):
      screen.blit(self.rotated_bird, self.bird_rect)

    def update_bird(self):
      self.bird_movement += self.gravity
      self.rotated_bird = self.rotate_bird()
      self.bird_rect.centery += self.bird_movement

    def rotate_bird(self):
      new_bird = pygame.transform.rotozoom(self.bird,-self.bird_moveme

      return new_bird

    def flap(self):
      self.bird_movement = 0
      self.bird_movement -= 2.5
```

Your code in the `lab4.py` file should look like this:

```
# Flappy Bird clone

import pygame
import sys
from game import Game

pygame.init()
screen = pygame.display.set_mode((400, 720))
clock = pygame.time.Clock()

game = Game('student_folder/flappy_bird/bird.png', 'student_folder/f

game.resize_images()

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit()

    if event.type == pygame.KEYDOWN:
      if event.key == pygame.K_SPACE and game.active:
        game.flap()

  game.show_background(screen)

  if game.active:
    game.show_bird(screen)
    game.update_bird()

  game.show_ground(screen)
  game.move_ground()

  pygame.display.update()
  clock.tick(120)
```

Advanced Topics Lab 5

Lab 5 - The Pipes

Adding the Pipes

Since there are several pipes that appear during the game, a list will be used to keep track of all of these objects. Pipes should appear at a specific interval. To do this, we are going to create a custom Pygame event. Start by creating the `SPAWNPIPE` event in the `lab5.py` file. Then have the event fire every 1.8 seconds (Pygame measures time in milliseconds).

```
SPAWNPIPE = pygame.USEREVENT
pygame.time.set_timer(SPAWNPIPE, 1800)
```

In the loop that iterates though all of the Pygame events, check to see if the `SPAWNPIPE` event has been activated. If so, call the `add_pipe()` to add a set of pipes to the game window.

```
if event.type == SPAWNPIPE:
  game.add_pipe()
```

In the previous step, we called the `add_pipe` method that has yet to be defined. Before we can define that method, we need to add some attributes to the `Game` class. Go to the `game.py` file and declare `pipes` and `pipe_height`. Both of these attributes are lists; `pipes` is a list of pipe surfaces that appear in the game. This list should be empty to start the game. There will be three different sizes of pipes in the game; `pipe_height` is a list of these heights.

```
class Game:
  def __init__(self, bird_img, pipe_img, background_img, ground_img):

    self.bird = pygame.image.load(bird_img).convert_alpha()
    self.bird_rect = self.bird.get_rect(center = (70, 180))
    self.pipe = pygame.image.load(pipe_img).convert_alpha()
    self.background = pygame.image.load(background_img).convert_alpha(

    self.ground= pygame.image.load(ground_img).convert()
    self.ground_position = 0
    self.active = True
    self.gravity = 0.05
    self.bird_movement = 0
    self.rotated_bird = pygame.Surface((0, 0))
    self.pipes = []
    self.pipe_height = [280, 425, 562]
```

Now we can define the `add_pipe` method. It first randomly selects a number from the `pipe_height` list. This number represents the vertical position of the pipes. We want there to be a top and bottom pipe, so two surfaces are created. The begin at position 600, which is outside the game window. The `- 211` represents the gap between the two pipes between which the bird will fly. Finally, these pipes are added to the list `pipes`.

```
def add_pipe(self):
  random_pipe_pos = random.choice(self.pipe_height)
  bottom_pipe = self.pipe.get_rect(midtop = (600, random_pipe_pos))
  top_pipe = self.pipe.get_rect(midbottom = (600, random_pipe_pos - 21

  self.pipes.append(bottom_pipe)
  self.pipes.append(top_pipe)
```

There is no `TRY IT` button because you could not see the pipes as they have not been drawn to the game window.

Moving the Pipes

We are going to update the position of the pipes before drawing to the game window. Iterate through the list of each pipe and change the x-position so the pipe moves to the left (the `-=` operator). The number `1.75` represents the speed of the pipes. You can adjust it to your liking. Once the pipes have moved past the left side of the game window (the x-position is less than or equal to -40), then remove the pipe from the list.

```
def move_pipes(self):
  for pipe in self.pipes:
    pipe.centerx -= 1.75
    if pipe.centerx <= -40:
      self.pipes.remove(pipe)
```

Now go to the `lab5.py` file and call the `move_pipes` method after updating the bird.

```
if game.active:
  game.show_bird(screen)
  game.update_bird()
  game.move_pipes()
```

Again, there is no `TRY IT` button because the pipes as they have not been drawn to the game window.

Drawing the Pipes

We are now ready to draw the pipes to the game window. The default pipe image has the opening facing up. Half of the pipes (the bottom pipes) can use the default image. The other half (the top pipes) need to flip the image vertically so the opening is facing down. We can determine if a pipe is on the top or bottom by the y-coordinate of the bottom of the pipe. If the bottom of the pipe is greater than or equal to the bottom edge of the game window (700), then this pipe does not need to be flipped. Draw it to the game window as normal. If the bottom of the pipe is inside the game window, this image needs to be flipped. Notice that `pygame.transform.flip` takes two Boolean values. The first one represents if the image is flipped horizontally, while the second represents if the image is flipped vertically.

```
def show_pipes(self, screen):
  for pipe in self.pipes:
    if pipe.bottom >= 700:
      screen.blit(self.pipe, pipe)
    else:
      flip_pipe = pygame.transform.flip(self.pipe, False, True)
      screen.blit(flip_pipe, pipe)
```

Go to the `lab5.py` file and call the `show_pipes` method the the pipes have been updated. Since `show_pipes` is drawing to the game window, it needs `screen` as a parameter.

```
if game.active:
  game.show_bird(screen)
  game.update_bird()
  game.move_pipes()
  game.show_pipes(screen)
```

Running the game now, you should see pairs of pipes moving from right to left across the game window.

Collision Detection

Right now, the bird can hit pipes or the ground and not affect game play. We are going to implement collision detection and stop the game when the bird hits a pipe or the ground. Iterate through the list of pipes and ask if the bird is colliding with the pipe. This is why we use the rectangles around the bird and pipes. Pygame will do the calculations to see if the two objects are touching. If the bird is touching a pipe, set `self.active` to `False`. We also need to check to see if the bird is exiting the top of the game window (the top of the bird is less than or equal to -100) or if bird is touching the ground (the bottom of the bird is greater than or equal to 650). In either case, set `self.active` to `False`.

```
def check_collision(self):
  for pipe in self.pipes:
    if self.bird_rect.colliderect(pipe):
      self.active = False

  if self.bird_rect.top <= -100 or self.bird_rect.bottom >= 650:
    self.active = False
```

Go back to `lab5.py` and call `check_collision` after the pipes have been moved.

```
if game.active:
  game.show_bird(screen)
  game.update_bird()
  game.move_pipes()
  game.show_pipes(screen)
  game.check_collision()
```

The bird and pipes should disappear once the bird collides with an object.

▼ **Code**

Your code in the `game.py` file should look like this:

```
  # File for Game class

  import pygame
  import random

  class Game:
    def __init__(self, bird_img, pipe_img, background_img, ground_img)

      self.bird = pygame.image.load(bird_img).convert_alpha()
      self.bird_rect = self.bird.get_rect(center = (70, 180))
      self.pipe = pygame.image.load(pipe_img).convert_alpha()
      self.background = pygame.image.load(background_img).convert_alph

      self.ground= pygame.image.load(ground_img).convert()
      self.ground_position = 0
      self.active = True
      self.active = True
      self.gravity = 0.05
      self.bird_movement = 0
      self.rotated_bird = pygame.Surface((0, 0))
      self.pipes = []
      self.pipe_height = [280, 425, 562]

    def resize_images(self):
      self.bird = pygame.transform.scale(self.bird, (51, 34))
      self.pipe = pygame.transform.scale(self.pipe, (80, 438))
      self.ground = pygame.transform.scale(self.ground, (470, 160))
      self.background = pygame.transform.scale(self.background, (400,


    def show_background(self, screen):
      screen.blit(self.background, (0,0))

    def show_ground(self, screen):
      screen.blit(self.ground, (self.ground_position, 650))
      screen.blit(self.ground, (self.ground_position + 470, 650))

    def move_ground(self):
      self.ground_position -= 1
      if self.ground_position <= -400:
        self.ground_position = 0

    def show_bird(self, screen):
      screen.blit(self.rotated_bird, self.bird_rect)

    def update_bird(self):
      self.bird_movement += self.gravity
      self.rotated_bird = self.rotate_bird()
      self.bird_rect.centery += self.bird_movement

    def rotate_bird(self):
      new_bird = pygame.transform.rotozoom(self.bird,-self.bird_moveme

      return new_bird

    def flap(self):
      self.bird_movement = 0
      self.bird_movement -= 2.5
    def add_pipe(self):
      random_pipe_pos = random.choice(self.pipe_height)
      bottom_pipe = self.pipe.get_rect(midtop = (600, random_pipe_pos)

      top_pipe = self.pipe.get_rect(midbottom = (600, random_pipe_pos

      self.pipes.append(bottom_pipe)
      self.pipes.append(top_pipe)

    def move_pipes(self):
      for pipe in self.pipes:
        pipe.centerx -= 1.75
        if pipe.centerx <= -40:
          self.pipes.remove(pipe)

    def show_pipes(self, screen):
      for pipe in self.pipes:
        if pipe.bottom >= 700:
          screen.blit(self.pipe, pipe)
        else:
          flip_pipe = pygame.transform.flip(self.pipe, False, True)
          screen.blit(flip_pipe, pipe)

    def check_collision(self):
      for pipe in self.pipes:
        if self.bird_rect.colliderect(pipe):
          self.active = False

      if self.bird_rect.top <= -100 or self.bird_rect.bottom >= 650:

        self.active = False
```

Your code in the `lab5.py` file should look like this:

```
# Flappy Bird clone

import pygame
import sys
from game import Game
from game import Game

pygame.init()
screen = pygame.display.set_mode((400, 720))
clock = pygame.time.Clock()

SPAWNPIPE = pygame.USEREVENT
pygame.time.set_timer(SPAWNPIPE, 1800)

game = Game('student_folder/flappy_bird/bird.png', 'student_folder/f

game.resize_images()

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit()

    if event.type == pygame.KEYDOWN:
      if event.key == pygame.K_SPACE and game.active:
        game.flap()

    if event.type == SPAWNPIPE:
      game.add_pipe()

  game.show_background(screen)

  if game.active:
    game.show_bird(screen)
    game.update_bird()
    game.move_pipes()
    game.show_pipes(screen)
    game.check_collision()

  game.show_ground(screen)
  game.move_ground()

  pygame.display.update()
  clock.tick(120)
```

Advanced Topics Lab 6

Lab 6 - Wrapping Up

Adding a Score

We need a few attributes so we can draw the score to the game window. The attribute `score` keeps track of the player’s score. The `font` attribute loads the default system font with the size of 48.

```
class Game:
  def __init__(self, bird_img, pipe_img, background_img, ground_img):

    self.bird = pygame.image.load(bird_img).convert_alpha()
    self.bird_rect = self.bird.get_rect(center = (70, 180))
    self.pipe = pygame.image.load(pipe_img).convert_alpha()
    self.background = pygame.image.load(background_img).convert_alpha(

    self.ground= pygame.image.load(ground_img).convert()
    self.ground_position = 0
    self.active = True
    self.gravity = 0.05
    self.bird_movement = 0
    self.rotated_bird = pygame.Surface((0, 0))
    self.pipes = []
    self.pipe_height = [280, 425, 562]
    self.score = 0
    self.font = pygame.font.SysFont(None, 48)
```

The next step is to increment the player’s score. The score does not reflect the number of pipes passed. Instead, players get a fraction (0.01) of a point for every frame. You can type cast the score as an integer so the game only displays whole numbers. So the longer the bird stays in flight, the higher the score. So the score is dependent upon flying between pipes, but it is not directly related to it.

```
def update_score(self):
  self.score += 0.01
```

The `show_score` method draws the score to the game window. This depends on if the game is actively played or not (we will had a high score once the game ends). `show_score` takes parameters for the state of the game (playing or game over), the screen, and the color of the text. The score itself needs to be converted to an integer and then to a string. It will be at the top of the window in the middle.

```
def show_score(self, game_state, screen, color):
  if game_state == 'playing':
    score_surface = self.font.render(str(int(self.score)), True, color

    score_rect = score_surface.get_rect(center=(202, 75))
    screen.blit(score_surface, score_rect)
```

Now go to the `lab6.py` file and call these two methods. The tuple `(255, 255, 255)` represents the color white. Computers mix red, green, and blue (called RGB) to come up with different colors. You can use online tools to generate RGB values if you want to use a different color.

```
if game.active:
  game.show_bird(screen)
  game.update_bird()
  game.move_pipes()
  game.show_pipes(screen)
  game.check_collision()
  game.update_score()
  game.show_score('playing', screen, (255, 255, 255))
```

Game Over

When the bird collides with a pipe or the ground, we want to show the high score and give the user a message on how to restart the game. In the `lab6.py` file, add an `else` statement to the `if game.active:` conditional. Call the game_over method and pass it the screen and a color for the text. This method will draw the high score and the instructions to the game window.

```
if game.active:
  game.show_bird(screen)
  game.update_bird()
  game.move_pipes()
  game.show_pipes(screen)
  game.check_collision()
  game.update_score()
  game.show_score('playing', screen, (255, 255, 255))
else:
  game.game_over(screen, (255, 255, 255))
```

```
def game_over(self, screen, color):
  self.update_high_score()
  self.show_score('game_over', screen, color)
```

The game_over method calls two helper methods, update_high_score and show_score. This time, however, show_score takes the game_over parameter, which indicates that the high score and instructions need to be drawn to the game window. Start by adding the high_score attribute and setting its initial value to 0.

```
class Game:
  def __init__(self, bird_img, pipe_img, background_img, ground_img):

    self.bird = pygame.image.load(bird_img).convert_alpha()
    self.bird_rect = self.bird.get_rect(center = (70, 180))
    self.pipe = pygame.image.load(pipe_img).convert_alpha()
    self.background = pygame.image.load(background_img).convert_alpha(

    self.ground= pygame.image.load(ground_img).convert()
    self.ground_position = 0
    self.active = True
    self.gravity = 0.05
    self.bird_movement = 0
    self.rotated_bird = pygame.Surface((0, 0))
    self.pipes = []
    self.pipe_height = [280, 425, 562]
    self.score = 0
    self.font = pygame.font.SysFont(None, 48)
    self.high_score = 0
```

Then create the `update_high_score` method. This method asks if the current score is greater than the high score. If so, change `high_score` to `score`.

```
def update_high_score(self):
  if self.score > self.high_score:
    self.high_score = self.score
```

Next, modify the `show_score` method. The score should be visible if the game is being played or if it is over. So remove the conditional about `playing` and just draw the score. However, we are going to add the text `Score:` before the number. Check to see if the game is over with a conditional. If true, we want to add the instructions `Press the Space Bar to Play Again`. However, this string is too long to put on a single line. Pygame does not recognize the newline character `\n`, so these instructions need to use two different surfaces. Finally, add the high score at the bottom of the game window.

```
def show_score(self, game_state, screen, color):
  score_surface = self.font.render('Score: {:d}'.format(int(self.score

  score_rect = score_surface.get_rect(center=(200, 75))
  screen.blit(score_surface, score_rect)

  if game_state == 'game_over':
    restart_text1 = self.font.render('Press Space Bar', True, color)

    restart_rect1 = restart_text1.get_rect(center=(200, 280))
    screen.blit(restart_text1, restart_rect1)

    restart_text2 = self.font.render('to Play Again', True, color)
    restart_rect2 = restart_text2.get_rect(center=(200, 340))
    screen.blit(restart_text2, restart_rect2)

    high_score_surface = self.font.render('High Score: {:d}'.format(in

    high_score_rect = high_score_surface.get_rect(center=(200, 610))

    screen.blit(high_score_surface, high_score_rect)
```

You should now see the instructions for restarting the game as well as the high score on the game window when the game ends. The space bar will not, however, cause the game to restart.

Restarting the Game

The final step is to have the game restart when the user presses the space bar. Since this is a Pygame event, we are going to add another conditional to `lab6.py`. The conditional should be “inside” the conditional that checks if a key is down. The game is over when `game.active` is False. So if the space bar is pressed while `game.active` is False, then call the `restart` method.

```
if event.type == pygame.KEYDOWN:
  if event.key == pygame.K_SPACE and game.active:
    game.flap()

  if event.key == pygame.K_SPACE and game.active == False:
    game.restart()
```

In the `game.py` file, add the `restart` method. Set the `active` attribute back to True, delete all of the pipe objects from the `pipes` list, but the bird back to its starting position, set `bird_movement` back to 0, and set the score back to 0. The game should revert back to its original state, and you can continue to play.

```
def restart(self):
  self.active = True
  del self.pipes[:]
  self.bird_rect.center = (70, 180)
  self.bird_movement = 0
  self.score = 0
```

▼ **Code**

Your code in the `game.py` file should look like this:

```
  # File for Game class

  import pygame
  import random

  class Game:
    def __init__(self, bird_img, pipe_img, background_img, ground_img)

      self.bird = pygame.image.load(bird_img).convert_alpha()
      self.bird_rect = self.bird.get_rect(center = (70, 180))
      self.pipe = pygame.image.load(pipe_img).convert_alpha()
      self.background = pygame.image.load(background_img).convert_alph

      self.ground= pygame.image.load(ground_img).convert()
      self.ground_position = 0
```

```
        self.active = True
        self.gravity = 0.05
        self.bird_movement = 0
        self.rotated_bird = pygame.Surface((0, 0))
        self.pipes = []
        self.pipe_height = [280, 425, 562]
        self.score = 0
        self.font = pygame.font.SysFont(None, 48)
        self.high_score = 0

    def resize_images(self):
        self.bird = pygame.transform.scale(self.bird, (51, 34))
        self.pipe = pygame.transform.scale(self.pipe, (80, 438))
        self.ground = pygame.transform.scale(self.ground, (470, 160))
        self.background = pygame.transform.scale(self.background, (400,


    def show_background(self, screen):
        screen.blit(self.background, (0,0))

    def show_ground(self, screen):
        screen.blit(self.ground, (self.ground_position, 650))
        screen.blit(self.ground, (self.ground_position + 470, 650))

    def move_ground(self):
        self.ground_position -= 1
        if self.ground_position <= -400:
          self.ground_position = 0

    def show_bird(self, screen):
        screen.blit(self.rotated_bird, self.bird_rect)

    def update_bird(self):
        self.bird_movement += self.gravity
        self.rotated_bird = self.rotate_bird()
        self.bird_rect.centery += self.bird_movement

    def rotate_bird(self):
        new_bird = pygame.transform.rotozoom(self.bird,-self.bird_moveme

        return new_bird
```

```
  def flap(self):
    self.bird_movement = 0
    self.bird_movement -= 2.5

  def add_pipe(self):
    random_pipe_pos = random.choice(self.pipe_height)
    bottom_pipe = self.pipe.get_rect(midtop = (600, random_pipe_pos)

    top_pipe = self.pipe.get_rect(midbottom = (600, random_pipe_pos

    self.pipes.append(bottom_pipe)
    self.pipes.append(top_pipe)

  def move_pipes(self):
    for pipe in self.pipes:
      pipe.centerx -= 1.75
      if pipe.centerx <= -40:
        self.pipes.remove(pipe)

  def show_pipes(self, screen):
    for pipe in self.pipes:
      if pipe.bottom >= 700:
        screen.blit(self.pipe, pipe)
      else:
        flip_pipe = pygame.transform.flip(self.pipe, False, True)
        screen.blit(flip_pipe, pipe)

  def check_collision(self):
    for pipe in self.pipes:
      if self.bird_rect.colliderect(pipe):
        self.active = False

    if self.bird_rect.top <= -100 or self.bird_rect.bottom >= 650:

      self.active = False

  def update_score(self):
    self.score += 0.01

  def show_score(self, game_state, screen, color):
    score_surface = self.font.render('Score: {:d}'.format(int(self.s
    score_surface = self.font.render('Score: {:d}'.format(int(self.s

    score_rect = score_surface.get_rect(center=(200, 75))
    screen.blit(score_surface, score_rect)

    if game_state == 'game_over':
      restart_text1 = self.font.render('Press Space Bar', True, colo

      restart_rect1 = restart_text1.get_rect(center=(200, 280))
      screen.blit(restart_text1, restart_rect1)

      restart_text2 = self.font.render('to Play Again', True, color)

      restart_rect2 = restart_text2.get_rect(center=(200, 340))
      screen.blit(restart_text2, restart_rect2)

      high_score_surface = self.font.render('High Score: {:d}'.forma

      high_score_rect = high_score_surface.get_rect(center=(200, 610

      screen.blit(high_score_surface, high_score_rect)

  def game_over(self, screen, color):
    self.update_high_score()
    self.show_score('game_over', screen, color)

  def update_high_score(self):
    if self.score > self.high_score:
      self.high_score = self.score

  def restart(self):
    self.active = True
    del self.pipes[:]
    self.bird_rect.center = (70, 180)
    self.bird_movement = 0
    self.score = 0
```

Your code in the `lab6.py` file should look like this:

```
# Flappy Bird clone
```

```
import pygame
import sys
from game import Game

pygame.init()
screen = pygame.display.set_mode((400, 720))
clock = pygame.time.Clock()

SPAWNPIPE = pygame.USEREVENT
pygame.time.set_timer(SPAWNPIPE, 1800)

game = Game('student_folder/flappy_bird/bird.png', 'student_folder/f

game.resize_images()

while True:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      pygame.quit()
      sys.exit()

    if event.type == pygame.KEYDOWN:
      if event.key == pygame.K_SPACE and game.active:
        game.flap()

      if event.key == pygame.K_SPACE and game.active == False:
        game.restart()

    if event.type == SPAWNPIPE:
      game.add_pipe()

  game.show_background(screen)

  if game.active:
    game.show_bird(screen)
    game.update_bird()
    game.move_pipes()
    game.show_pipes(screen)
    game.check_collision()
    game.update_score()
```

```
      game.show_score('playing', screen, (255, 255, 255))
    else:
      game.game_over(screen, (255, 255, 255))

    game.show_ground(screen)
    game.move_ground()

    pygame.display.update()
    clock.tick(120)
```

Advanced Topics Lab Challenge

Problem

There exists a user-defined class named `Stats` in the Python module `stats`. **Note**, there are no parameters when you instantiate an object of the `Stats` class. It has the methods, `mean`, `median`, and `mode`. Write a script that uses each of the methods with the following list:

```
[8, 7, 3, 9, 1, 4, 3]
```

Expected Output

When you print each method call with the list above, you should see the output below.

Method Call	Output
`print(my_stats.mean([8, 7, 3, 9, 1, 4, 3]))`	mean: 5
`print(my_stats.median([8, 7, 3, 9, 1, 4, 3]))`	median: 4
`print(my_stats.mode([8, 7, 3, 9, 1, 4, 3]))`	mode: 3