Lab 19: PyGame (cont.)

Apr 10, 2018

Main Event

1. LibPng

In some implementations, Pygame relies on libpng to display PNG images. If you are experiencing problems trying to blit PNG's to your screen, it may be because you don't have this library installed.

- Mac users should install via Homebrew:

```
$> brew install libpng
```

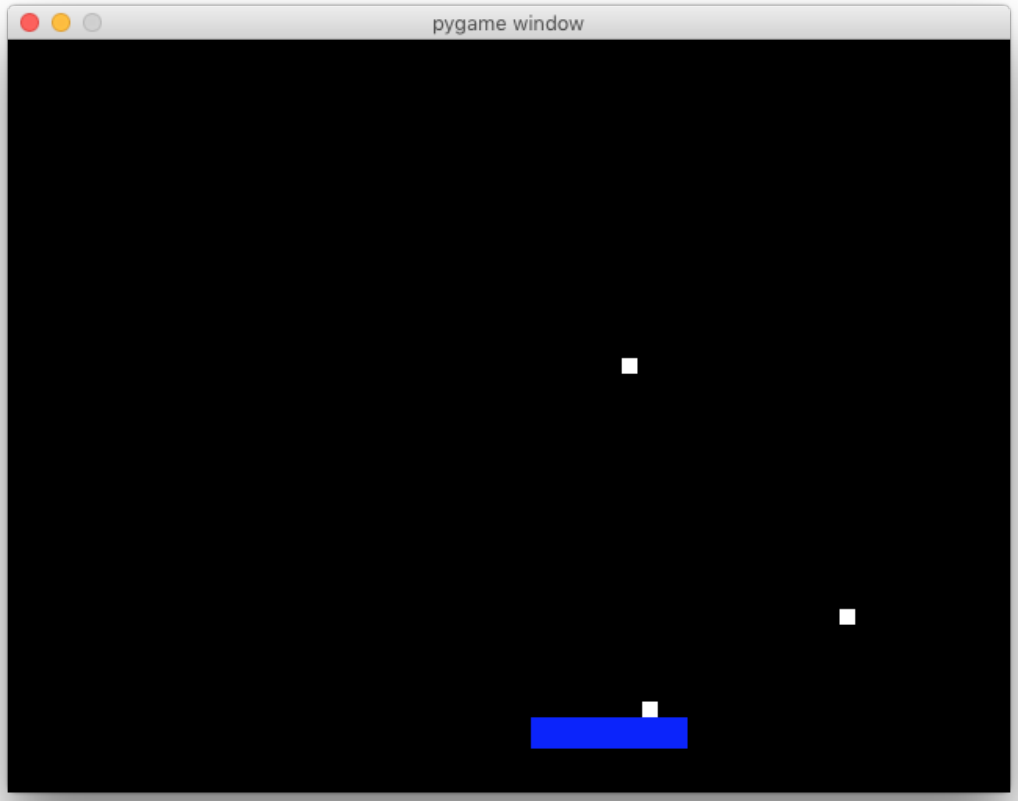
- Linux users can install via apt:

```
$> sudo apt-get install libpng-dev
```

- Windows users should ask a course staff member.

2. Paddle ball

Make the paddle hit the ball:



- Update your particle class to operate with rectangles instead of circles. Your particles should move around the screen, “bouncing” off the walls. For now, it is good enough to have a single particle on your surface.
- Create another rectangle that is able to “slide” from left-to-right depending on the arrow key being pressed.
- If the ball collides with the sliding rectangle, have it “bounce” away in another direction. How this bouncing is accomplished is up to you. Pygame provides a method for calculating the physics, if you’re up for figuring it out.
- Maintain a counter in the upper-right corner of the screen that increments each time the particle bounces off the slider.
- If the ball moves past the slider—its coordinates are below the bottom of the screen—the game should end.
- Finally, if a user clicks somewhere on the surface, add a particle to the game based on the position of the mouse click.

Solution:

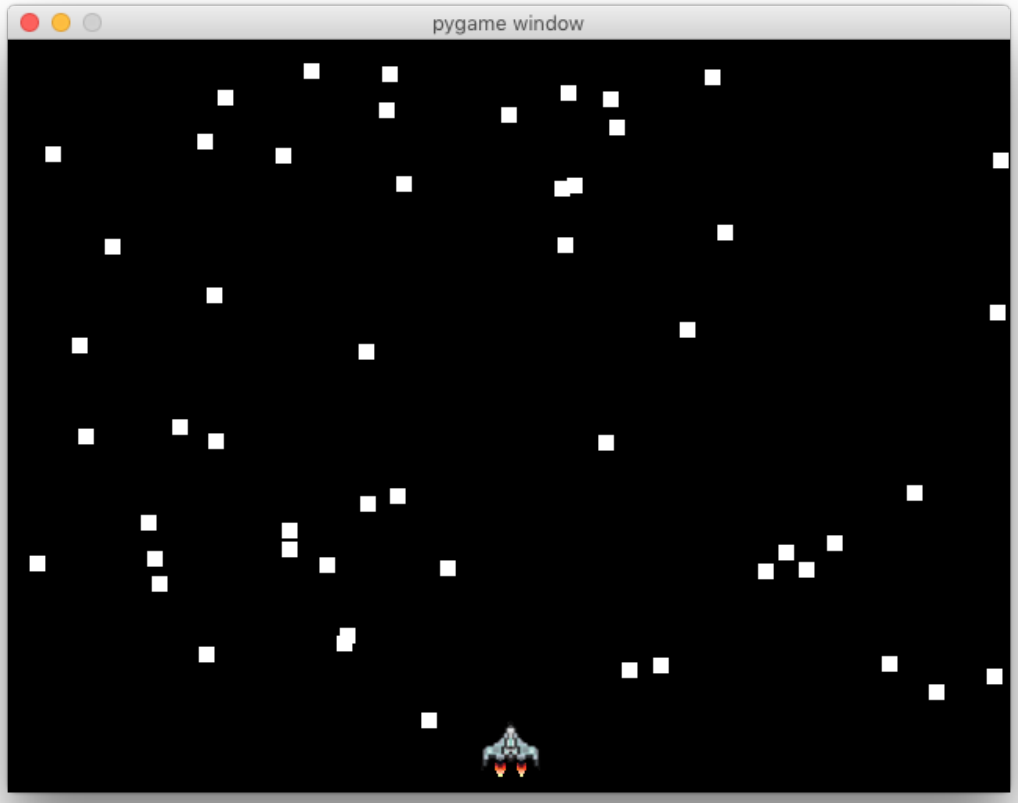
```
import sys
import pygame

class Pair:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def astuple(self):
        return (self.x, self.y)

class Ball:
    def __init__(self, pos):
        self.sprite = pygame.Rect(pos, (10, 10))
        self.velocity = Pair(1, 1)

    def move(self):
        self.sprite.move_ip(self.velocity.astuple())

    def display(self, surface):
        pygame.draw.rect(surface, (255, 255, 255), self.sprite)

class ScoreBoard:
    def __init__(self, padding):
        self.font = pygame.font.Font(None, 36)
        self.score = 0
        self.padding = padding

    def update(self):
        self.score += 1

    def display(self, surface):
        text = self.font.render(str(self.score), True, (255, 0, 0))
        pair = Pair(surface.get_width() - text.get_width() - self.padd
                    text.get_height() + self.padding)
        surface.blit(text, pair.astuple())

pygame.init()
surface = pygame.display.set_mode((640, 480))

width = 100
```

```
left = round(surface.get_width() / 2 - width)
top = round(surface.get_height() * 0.9)
bar = pygame.Rect(left, top, width, 20)

score = ScoreBoard(10)

balls = []
playing = True

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            newball = Ball(pygame.mouse.get_pos())
            balls.append(newball)

    # Don't draw a new screen if the game is over
    if not playing:
        continue

    surface.fill((0, 0, 0))

    # Render the score board
    score.display(surface)

    # Move the paddle based on key movement
    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT] and bar.left - 1 > 0:
        bar.move_ip(-1, 0)
    if keys[pygame.K_RIGHT] and bar.right + 1 < surface.get_width():
        bar.move_ip(1, 0)
    pygame.draw.rect(surface, (0, 0, 255), bar)

    # Manipulate the balls
    for b in balls:
        b.move()
        b.display(surface)
        if b.sprite.colliderect(bar) or b.sprite.top < 0:
            # If the ball was heading downward, it hit the paddle
            # (otherwise it hit the ceiling)
            if b.velocity.y > 0:
                score.update()
```

```
            b.velocity.y *= -1
        if b.sprite.left < 0 or b.sprite.right >= surface.get_width():
            b.velocity.x *= -1
        if b.sprite.bottom > surface.get_height():
            playing = False

    pygame.display.update()
pygame.display.quit()
```

3. Invaders

The objective of this game is to have the spaceship dodge particles:



An image of the spaceship can be downloaded here. Its dimensions are 40-pixels by 40-pixels.

- Particles will now be created dynamically and must decide where to be placed on their own. Thus, modify your particle class from the previous exercise such that the constructor takes a single argument representing the width of the surface. The constructor should then create a rectangle and decide where it

should go; that is between position zero and the width. Since it starts at the top, the horizontal component can always be zero.

The velocity of the particle should be such that it falls straight down.

- Create a class that contains your ship. The class should actually do two things: maintain the ship image—so actually load the image in the constructor—and maintain a rectangle that surrounds the image. The rectangle should not be displayed, but its location should *always* correspond to the ship image. To do this, create three methods:
  - Display the object

    ```
    def display(self)
    ```

    "Blits" the image to the surface using the coordinates of the rectangle. A rectangle object has several attributes that will allow you get access to its location; `topleft` is probably the most useful.
  - Move left

    ```
    def left(self)
    ```

    Moves the rectangle to the left, much like what was done to the bar in the previous exercise.
  - Move right

    ```
    def right(self)
    ```

    Analogous, but moves the rectangle to the right.
- In the event loop, move the ship based on the key board input; again, much like what was done in the previous exercise.
- Decide whether to create a new particle. Using the random number generator can help.
- Move and display each particle, along with the ship. If a particle collides with a ship, the game should be over.

Solution:

```
import random

import pygame

class Ball:
    def __init__(self, width):
        where = random.randint(10, width - 10)
        self.sprite = pygame.Rect(where, 0, 10, 10)
        self.velocity = (0, 1)

    def move(self):
        self.sprite.move_ip(self.velocity)

class Ship:
    def __init__(self, surface):
        self.surface = surface
        self.img = pygame.image.load('space-ship.png').convert_alpha()
        width = self.img.get_width()
        self.box = pygame.Rect(round(self.surface.get_width() / 2 - wi
                               round(self.surface.get_height() * 0.9),
                               width,
                               self.img.get_height())

    def display(self):
        self.surface.blit(self.img, self.box.topleft)

    def left(self):
        if self.box.left - 1 > 0:
            self.box.move_ip(-1, 0)

    def right(self):
        if self.box.right + 1 < self.surface.get_width():
            self.box.move_ip(1, 0)

pygame.init()
surface = pygame.display.set_mode((640, 480))

ship = Ship(surface)
balls = []

play = True
```

```
while play:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            play = False
            break

    if random.randrange(10) > 8:
        b = Ball(surface.get_width())
        balls.append(b)

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        ship.left()
    if keys[pygame.K_RIGHT]:
        ship.right()

    surface.fill((0, 0, 0))

    for i in balls:
        i.move()
        pygame.draw.rect(surface, (255, 255, 255), i.sprite)
        if i.sprite.colliderect(ship.box):
            play = False
            break

    ship.display()

    pygame.display.update()
pygame.display.quit()
```

Additional Practice

1. Removing particles

Managing all those random particles in the previous exercise required some sort of collection-like data structure. If you were able to play the game for a while, you may notice it getting slow. This is likely because that collection structure has grown into a behemoth of particles. However, do you actually need all of those particles?

Once a particle makes its way off of the screen—it has been dodged by the ship—there is really no need to keep it in the collection any longer. There are several strategies for removing elements from a collection. Think through how you might

do this on paper before trying an implementation. Keep in mind that removing items from a collection while also iterating over that collection is generally not a good idea.

2. Shooting invaders

Allow your ship to go on the offensive:

- If the up-arrow is pressed, a “bullet” (another rectangle, really) should be release from the nose of the ship. It should travel upwards, until it hits a particle. Such a collision should erase both the bullet and the particle from the screen.
- Keep score! For each particle that’s hit, increment some counter.
- Make the end-of-game nicer than merely having Pygame quit. Some screen overlay announcing the finish could be fun.

3. Physics simulation

Physics is an important part of getting video games right. Portions of this lab have touched on how to do that, but have done so in a crude fashion, and with very little depth. Work your way through Peter Collingridge’s Pygame physics simulation tutorial to become a real master.

Introduction to Computer Science

Introduction to Computer Science
jerome.white@nyu.edu

jerome-white

Learning computer science concepts through practice.