""" paper air force

top-down flying game

fly your paper airplane with mouse

bomb or shoot targets

avoid rain

"""

Use the multiline comment at the top of your code to identify the name and other useful information about the file.

import pygame, sys

pygame.init()

The first line will import the *pygame* and *sys* modules*. Pygame* is used to enable the 2d api and call on a variety of sprite, drawing and event functions*. Sys* is used to call upon a variety of system parameters and functions, including *exit().* The second line will initialize the *pygame* module

screen = pygame.display.set\_mode((640, 480))

The *screen* variable listed above is the same as in previous assignments, and defines the display window that will be used by the background surface later in the game function.

class Plane(pygame.sprite.Sprite):

def \_\_init\_\_(self):

pygame.sprite.Sprite.\_\_init\_\_(self)

A *class* is an object. More importantly, it is the blueprint of an object. It describes how copies (instances) of this object will behave in your program. Small changes to the arguments listed for each method will allow you to make instances of the object with some variation. The arguments for this specific method are listed below.

self.image = pygame.image.load("images/pa\_fold0009.jpg")

self.image = self.image.convert()

tranColor = self.image.get\_at((1, 1))

self.image.set\_colorkey(tranColor)

self.rect = self.image.get\_rect()

self.rect.center = (320, 240)

Use the *\_\_init\_\_(self)* method as a ‘constructor’ for the class. Any object based on this class will have assigned to it the attributes listed in this method. Notice that the arguments listed within the *\_\_init\_\_* method start with *self*. Classes need a way to refer to themselves, and a *self* argument in the \_\_init\_\_ method is the way to handle that.

def update(self):

mousex, mousey = pygame.mouse.get\_pos()

self.rect.center = (mousex, mousey)

An *update()* method in this class will alter *self.image’s* position by assigning the rectangle’s *centerx* and *centery* values to those of the mouse cursor on the screen. The position will be updated with every frame. The update method is told to run within the *while* loop listed in the *main()* function.

class Ocean(pygame.sprite.Sprite):

def \_\_init\_\_(self):

pygame.sprite.Sprite.\_\_init\_\_(self)

self.image = pygame.image.load("images/ocean.gif")

self.image = self.image.convert()

self.rect = self.image.get\_rect()

The *Ocean* class will define only one object in the game, the ocean background.

def game():

pygame.display.set\_caption("Paper Air Force")

background = pygame.Surface(screen.get\_size())

background.fill((0, 0, 0))

screen.blit(background, (0, 0))

The *background* entity, like the *screen* entity mentioned earlier, was used in previous examples and assignments. Both exist outside of our classes for now; the *background* is defined in the *game()* function.

plane = Plane()

ocean = Ocean()

Here is where we create the instance objects, otherwise known as sprites. All sprite instances need to point to the Class from which they are derived. For naming convention purposes, sprite objects are lower-case, and Class objects are upper-case.

oceanSprites = pygame.sprite.Group(ocean)

planeSprites = pygame.sprite.Group(plane)

Sprite groups are created to help organize sprites, and to assist with drawing and updating sprites within the game. They are also there to help with various events, such as collisions. Populate sprite groups by passing instances in the argument. In this example, an *oceanSprites* group is created to handle the ocean sprite used as the background and the *planeSprites* group is there to handle the plane sprite.

clock = pygame.time.Clock()

keepGoing = True

The *clock* argument will call on pygame’s *time* method to setup the timing for the game. The *keepGoing* argument is a sentry variable that allows us to leave the *while* loop (below) once the variable has been reset to *False*.

while keepGoing:

clock.tick(30)

pygame.mouse.set\_visible(False)

for event in pygame.event.get():

if event.type == pygame.QUIT:

keepGoing = False

pygame.display.quit()

pygame.quit()

sys.exit()

Here is where you can find the *while* loop. Most of it contains a *for* loop that checks for events. In this case, if the event is a *QUIT* event, both *pygame.display* and *pygame* will quit, and the *sys exit()* function will be called. Most importantly, the sentry variable *keepGoing* will be set to *False*, allowing us to break out of the while loop.

oceanSprites.update()

planeSprites.update()

oceanSprites.draw(screen)

planeSprites.draw(screen)

pygame.display.flip()

Sprites need to be cleared, updated, and redrawn every frame. It is much easier to manage this with sprite groups than with individual sprites. Meanwhile, *pygame.display.flip()* will update the entire contents of the screen display.

#return mouse cursor

pygame.mouse.set\_visible(True)

if \_\_name\_\_ == "\_\_main\_\_":

game()

----------------------------------

""" paper air force

top-down flying game

fly your paper airplane with mouse

bomb targets

avoid rain

"""

import pygame, random, sys

pygame.init()

#direction constants

PA\_FOLD = 0

PA\_PITCH\_DOWN = 1

PA\_PITCH\_UP = 2

PA\_ROLL\_LEFT = 3

PA\_ROLL\_RIGHT = 4

PA\_FLY = 5

screen = pygame.display.set\_mode((640, 480))

class Bullet(pygame.sprite.Sprite):

def \_\_init\_\_(self):

pygame.sprite.Sprite.\_\_init\_\_(self)

self.image = pygame.image.load("images/bullet\_sprite.png")

self.image = self.image.convert()

tranColor = self.image.get\_at((1,1))

self.image.set\_colorkey(tranColor)

self.rect = self.image.get\_rect()

self.dy = 20

#self.rect.center = (320,240)

self.image = pygame.transform.scale(self.image,(20,40))

def update(self):

self.rect.centery -= self.dy

def checkbounds(self):

if self.rect.centery <= 0:

self.reset()

def reset(self):

pygame.sprite.Sprite.remove

class Plane(pygame.sprite.Sprite):

def \_\_init\_\_(self):

pygame.sprite.Sprite.\_\_init\_\_(self)

self.PA\_FOLD = 0

self.PA\_ROLL\_LEFT = 1

self.PA\_ROLL\_RIGHT = 2

self.PA\_FLY\_STRAIGHT = 3

#self.imgList = []

#self.loadPics()

self.dir = self.PA\_FLY\_STRAIGHT

#self.frame = 0

self.image = pygame.image.load("images/pa\_fold0009.jpg")

self.image = self.image.convert()

tranColor = self.image.get\_at((1, 1))

self.image.set\_colorkey(tranColor)

self.rect = self.image.get\_rect()

self.rect.center = (320, 240)

self.delay = 3

self.pause = self.delay

self.speed = 10

self.dx = 0

self.dy = 0

def update(self):

mousex, mousey = pygame.mouse.get\_pos()

self.rect.center = (mousex, mousey)

self.input()

def input(self):

rel\_x, rel\_y = pygame.mouse.get\_rel()

if rel\_x < 0:

#print ("<0")

self.image = pygame.image.load("images/pa\_roll\_left0009.jpg")

self.image = self.image.convert()

tranColor = self.image.get\_at((1, 1))

self.image.set\_colorkey(tranColor)

elif rel\_x > 0:

self.image = pygame.image.load("images/pa\_roll\_right0009.jpg")

self.image = self.image.convert()

tranColor = self.image.get\_at((1, 1))

self.image.set\_colorkey(tranColor)

elif rel\_y > 0:

self.image = pygame.image.load("images/pa\_pitch\_up0009.jpg")

self.image = self.image.convert()

tranColor = self.image.get\_at((1, 1))

self.image.set\_colorkey(tranColor)

elif rel\_y < 0:

self.image = pygame.image.load("images/pa\_pitch\_down0004.jpg")

self.image = self.image.convert()

tranColor = self.image.get\_at((1, 1))

self.image.set\_colorkey(tranColor)

elif rel\_x == 0 and rel\_y == 0:

self.image = pygame.image.load("images/pa\_pitch\_down0000.jpg")

self.image = self.image.convert()

tranColor = self.image.get\_at((1, 1))

self.image.set\_colorkey(tranColor)

class Opp(pygame.sprite.Sprite):

def \_\_init\_\_(self):

pygame.sprite.Sprite.\_\_init\_\_(self)

#self.dir = self.PA\_FLY\_STRAIGHT

self.image = pygame.image.load("images/pa\_fold0009.jpg")

self.image = self.image.convert()

tranColor = self.image.get\_at((1, 1))

self.image.set\_colorkey(tranColor)

self.rect = self.image.get\_rect()

self.rect.center = (320, 50)

self.delay = 3

self.pause = self.delay

self.speed = 10

self.dx = 0

self.dy = 0

self.image = pygame.transform.flip(self.image, 0, 1)

def update(self):

pass

class Ocean(pygame.sprite.Sprite):

def \_\_init\_\_(self):

pygame.sprite.Sprite.\_\_init\_\_(self)

self.image = pygame.image.load("images/ocean.gif")

self.rect = self.image.get\_rect()

self.dy = 5

self.reset()

def update(self):

self.rect.bottom += self.dy

if self.rect.bottom >= 1440:

self.reset()

def reset(self):

self.rect.top = -960

def game():

pygame.display.set\_caption("Mail Pilot!")

background = pygame.Surface(screen.get\_size())

background.fill((0, 0, 0))

screen.blit(background, (0, 0))

plane = Plane()

ocean = Ocean()

bullet = Bullet()

opp = Opp()

pygame.key.set\_repeat(1,1)

oceanSprites = pygame.sprite.Group(ocean)

planeSprites = pygame.sprite.Group(plane)

bulletSprites = pygame.sprite.Group(bullet)

oppSprites = pygame.sprite.Group(opp)

#pygame.transform.scale(bullet.image, (20,20))

#print (bullet.image.size)

clock = pygame.time.Clock()

keepGoing = True

while keepGoing:

clock.tick(30)

pygame.mouse.set\_visible(False)

for event in pygame.event.get():

if event.type == pygame.QUIT:

keepGoing = False

pygame.display.quit()

pygame.quit()

sys.exit()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE:

bullet.rect.center = plane.rect.center

print(bulletSprites)

if pygame.sprite.spritecollide(plane, oppSprites, 0):

print("boom")

oceanSprites.update()

planeSprites.update()

bulletSprites.update()

oppSprites.update()

oceanSprites.draw(screen)

planeSprites.draw(screen)

bulletSprites.draw(screen)

oppSprites.draw(screen)

pygame.display.flip()

#plane.sndEngine.stop()

#return mouse cursor

#pygame.mouse.set\_visible(True)

#return scoreboard.score

def main():

donePlaying = False

score = 0

while not donePlaying:

#donePlaying = instructions(score)

if not donePlaying:

score = game()

if \_\_name\_\_ == "\_\_main\_\_":

main()