import pygame,math  
from pygame.locals import \*  
from math import \*  
from random import randint  
  
pygame.mixer.pre\_init(44100, -16, 2, 2048)  
pygame.init()  
pygame.joystick.init()  
  
black, white, light\_gray, gray, dark\_gray = (0,0,0), (255,255,255), (170,170,170), (128,128,128), (85,85,85)  
red, green, blue = (255,0,0), (0,255,0), (0,0,255)  
yellow, magenta, cyan = (255,255,0), (255,0,255), (0,255,255)  
N,NE,E,SE,S,SW,W,NW,C = 0,1,2,3,4,5,6,7,8  
  
class Mouse(object):  
 def \_\_init\_\_(self):  
 self.x, self.y, self.visible, self.width, self.height, self.name = 0, 0, True, 0, 0, "mouse"  
 self.LeftClick, self.LeftClickable, self.LeftButton = False, True, None  
 self.RightClick, self.RightClickable, self.RightButton = False, True, None  
  
class KeyBoard(object):  
 def \_\_init\_\_(self):  
 self.Down, self.Up, self.Pressed = None, None, pygame.key.get\_pressed()  
  
class Joystick(object):  
 def \_\_init\_\_(self):  
 self.pad = [0 for x in range(9)]  
 self.button = [0 for x in range(12)]  
 if pygame.joystick.get\_count() == 0:  
 self.pad[8] = True  
 self.connected = False  
 print("No Joystick Connected")  
 else:  
 self.connected = True  
 self.joystick = pygame.joystick.Joystick(0)  
 self.joystick.init()  
 self.pad = [0 for x in range(9)]  
 self.button = [0 for x in range(self.joystick.get\_numbuttons())]  
 def stick(self,s="left",axis="x"):  
 if self.connected:  
 pole = 0  
 if s == "right":  
 pole += 2  
 if axis == "y":  
 pole += 1  
 return self.joystick.get\_axis(pole)  
  
  
keys = KeyBoard()  
mouse = Mouse()  
joy = Joystick()  
  
class Font(object):  
 def \_\_init\_\_(self, color = white, size = 24, shadowColor = None, family = None):  
 self.color, self.size, self.shadowColor = color, size, shadowColor  
  
 if family != None and family[-3:] == "ttf":  
 self.family = family   
 elif family != None:  
 self.findFont(family)  
 else:  
 self.family = None  
   
 def findFont(self,newFont):  
 self.family = pygame.font.match\_font(newFont)  
  
class Sound(object):  
 def \_\_init\_\_(self,path,chan):  
 self.chan = chan  
 self.file = pygame.mixer.Sound(path)  
  
 def play(self,time=0):  
 #time is measured in ms  
 c = pygame.mixer.Channel(self.chan)  
 if not c.get\_busy():  
 c.play(self.file,maxtime=time)  
  
class Game(object):  
 def \_\_init\_\_(self,w,h,title,time=0):  
 self.width,self.height = w,h  
 pygame.display.set\_caption(title)  
 self.screen = pygame.display.set\_mode([w,h])  
 self.background,self.backgroundXY, self.backgroundXYSet = None, [],False  
 self.fps, self.time, self.clock = 20, time + 1, pygame.time.Clock()  
 self.left, self.top, self.right, self.bottom = 0,0,w,h  
 self.over = False  
 self.score = 0  
 self.font = Font()  
  
 def clearBackground(self,color=(0,0,0)):  
 self.screen.fill(color)  
   
 def setBackground(self,bkGraphics):  
 self.background = bkGraphics  
 self.backgroundXY = []  
 self.backgroundXY.append({"x":bkGraphics.x,"y":bkGraphics.y})  
 self.backgroundXY.append({"x":bkGraphics.x,"y":bkGraphics.y})  
  
 def drawBackground(self):  
 self.background.draw()  
  
 def scrollBackground(self,direction,amt):   
 if not self.backgroundXYSet:  
 if direction == "left" or direction == "right":  
 self.backgroundXY[1]["x"] = self.backgroundXY[0]["x"] + self.background.width   
 elif direction == "up" or direction == "down":  
 self.backgroundXY[1]["y"] = self.backgroundXY[0]["y"] + self.background.height   
 self.backgroundXYSet = True  
   
 for i in range(len(self.backgroundXY)):  
 if direction == "left":  
 self.backgroundXY[i]["x"] -= amt  
 if self.backgroundXY[0]["x"] + self.background.width / 2 <= 0:  
 self.backgroundXY[0]["x"] = self.backgroundXY[1]["x"] + self.background.width   
 if self.backgroundXY[1]["x"] + self.background.width / 2 <= 0:  
 self.backgroundXY[1]["x"] = self.backgroundXY[0]["x"] + self.background.width  
   
 elif direction == "right":  
 self.backgroundXY[i]["x"] += amt  
 if self.backgroundXY[0]["x"] - self.background.width / 2 >= self.width:  
 self.backgroundXY[0]["x"] = self.backgroundXY[1]["x"] - self.background.width   
 if self.backgroundXY[1]["x"] - self.background.width / 2 >= self.width:  
 self.backgroundXY[1]["x"] = self.backgroundXY[0]["x"] - self.background.width   
   
 elif direction == "up":  
 self.backgroundXY[i]["y"] -= amt  
 if self.backgroundXY[0]["y"] + self.background.height / 2 <= 0:  
 self.backgroundXY[0]["y"] = self.backgroundXY[1]["y"] + self.background.height   
 if self.backgroundXY[1]["y"] + self.background.height / 2 <= 0:  
 self.backgroundXY[1]["y"] = self.backgroundXY[0]["y"] + self.background.height  
   
 elif direction == "down":  
 self.backgroundXY[i]["y"] += amt  
 if self.backgroundXY[0]["y"] - self.background.height / 2 >= self.height:  
 self.backgroundXY[0]["y"] = self.backgroundXY[1]["y"] - self.background.height   
 if self.backgroundXY[1]["y"] - self.background.height / 2 >= self.height:  
 self.backgroundXY[1]["y"] = self.backgroundXY[0]["y"] - self.background.height  
   
 self.background.moveTo(self.backgroundXY[i]["x"],self.backgroundXY[i]["y"])  
  
 def drawText(self,msg,x,y,newFont = None):  
 if newFont == None:  
 newFont = self.font  
 try:  
 textfont = pygame.font.Font(newFont.family,newFont.size)  
 except:  
 textfont = pygame.font.Font(pygame.font.match\_font("arial"),24)  
   
 if newFont.shadowColor != None:  
 text = textfont.render(str(msg),True,newFont.shadowColor)  
 self.screen.blit(text,[x+1,y+1])   
 text = textfont.render(str(msg),True,newFont.color)  
 self.screen.blit(text,[x,y])  
  
 def displayScore(self,x=5,y=5,newFont = None):  
 self.drawText("Score: " + str(self.score),x,y, newFont)  
  
 def displayTime(self,x=0,y=0,newFont = None):  
 self.drawText("Time: " + str(int(self.time)),x,y,newFont)  
  
 def setMusic(self,path):  
 pygame.mixer.music.load(path)  
  
 def playMusic(self):  
 pygame.mixer.music.play(-1,0.0)  
  
 def stopMusic(self):  
 pygame.mixer.music.stop()  
  
 def update(self,fps=1):  
 self.fps = fps  
 if self.time > 0:  
 self.time -= 1/fps  
 mouse.LeftClick = False  
 mouse.RightClick = False  
 pygame.display.flip()  
 self.clock.tick(fps)  
  
 def viewMouse(self,visibility):  
 mouse.visible = visibility  
 pygame.mouse.set\_visible(visibility)  
  
 def processInput(self):  
 self.keysPressed = pygame.key.get\_pressed()  
 keys.Pressed = pygame.key.get\_pressed()  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 self.over = True  
 if event.type == pygame.KEYDOWN:  
 keys.Down = event.key  
 else:  
 keys.Down = None  
  
 if event.type == pygame.KEYUP:  
 keys.Up = event.key  
 else:  
 keys.Up = None  
  
 pos = pygame.mouse.get\_pos()  
 mouse.x, mouse.y = pos  
 button = pygame.mouse.get\_pressed()  
 mouse.LeftButton = button[0]  
 mouse.RightButton = button[2]  
   
 if mouse.LeftClickable and mouse.LeftButton:  
 mouse.LeftClick = True  
 mouse.LeftClickable = False  
 if not mouse.LeftClickable and not mouse.LeftButton:  
 mouse.LeftClickable = True  
   
 if mouse.RightClickable and mouse.RightButton:  
 mouse.RightClick = True  
 mouse.RightClickable = False  
 if not mouse.RightClickable and not mouse.RightButton:  
 mouse.RightClickable = True  
   
 pygame.mouse.set\_visible(mouse.visible)  
 if joy.connected:  
 padx,pady = joy.joystick.get\_hat(0)  
 joy.pad = [0 for x in range(9)]  
 if padx == 0 and pady == 1: joy.pad[N] = True  
 if padx == -1 and pady == 0: joy.pad[E] = True  
 if padx == 0 and pady == -1: joy.pad[S] = True  
 if padx == 1 and pady == 0: joy.pad[W] = True  
 if padx == 0 and pady == 0: joy.pad[C] = True  
  
 for x in range(len(joy.button)):  
 joy.button[x] = joy.joystick.get\_button(x)  
  
 def wait(self,key):  
 while True:  
 self.processInput()  
 if self.keysPressed[key]:  
 return  
  
 def quit(self):  
 pygame.quit()  
  
class Image(object):  
 def \_\_init\_\_(self,path,game,use\_alpha=True):  
 self.game = game  
 if not isinstance(path, str):  
 self.image = path  
 else:  
 if use\_alpha:  
 self.image = pygame.image.load(path).convert\_alpha()  
 else:  
 self.image = pygame.image.load(path).convert()  
 trans\_color = self.image.get\_at((0,0))  
 self.image.set\_colorkey(trans\_color)  
   
 self.width,self.original\_width,self.oldwidth = self.image.get\_width(),self.image.get\_width(),self.image.get\_width()  
 self.height, self.original\_height,self.oldheight = self.image.get\_height(), self.image.get\_height(), self.image.get\_height()  
 self.rect = None  
 self.original, self.src = self.image, self.image  
 self.angle, self.da = 0,0  
 self.x, self.y, self.dx, self.dy, self.dxsign, self.dysign = self.game.width/2,self.game.height/2,0,0,1,1  
 self.left, self.top, self.right, self.bottom = 0,0,0,0  
 self.bounce = False  
 self.rotate,self.rotate\_angle, self.rda = "still",0,0  
 self.speed = 0  
 self.visible = True  
 self.health = 100  
 self.damage = 0  
  
 def setImage(self,image):  
 self.image = image  
 self.original = image  
  
 def collidedWith(self,obj,shape="circle"):  
 if (obj.visible or isinstance(obj,Mouse)) and self.visible:   
 if shape =="circle":  
 dx = self.x - obj.x  
 dy = self.y - obj.y  
 d = sqrt(pow(dx,2) + pow(dy,2))  
 if d < self.width/4 + obj.width/4:  
 return True   
 elif shape == "rectangle" and self.rect.colliderect(obj.rect):  
 return True   
 return False  
  
 def draw(self):  
 if self.width != self.oldwidth or self.height != self.oldheight:  
 self.resizeTo(self.width,self.height)  
 if self.rotate == "left" or self.rotate == "right" or self.rotate == "to":  
 self.image = self.original  
 self.image = pygame.transform.rotate(self.image,self.rotate\_angle \* 180 / math.pi)  
 self.width,self.height = self.image.get\_width(),self.image.get\_height()  
 self.oldwidth,self.oldheight = self.width,self.height  
 if self.visible:  
 self.game.screen.blit(self.image, [self.x - self.width/2,self.y - self.height/2])  
 self.left, self.top, self.right, self.bottom = self.x-self.width/2,self.y-self.height/2, self.x + self.width/2, self.y + self.height/2  
 self.rect = pygame.Rect(self.left,self.top,self.width,self.height)  
  
 def move(self, bounce = False):  
 if bounce:  
 if self.left < self.game.left or self.right > self.game.right:  
 self.changeXSpeed()  
 if self.top < self.game.top or self.bottom > self.game.bottom:  
 self.changeYSpeed()  
 self.calculateSpeedDeltas()  
 self.x += self.dx \* self.dxsign  
 self.y += self.dy \* self.dysign  
 self.draw()  
  
 def changeXSpeed(self,dx = -999):  
 if dx == -999:  
 self.dxsign = -self.dxsign  
 else:  
 self.dx = dx  
 def changeYSpeed(self,dy = -999):  
 if dy == -999:  
 self.dysign = -self.dysign  
 else:  
 self.dy = dy  
  
 def forward(self,speed):  
 self.speed = speed  
 self.calculateSpeedDeltas()  
 #self.move()  
 def rotateBy(self,angle=0,direction="right"):  
 rad = angle \* math.pi / 180  
 self.rotate = direction  
 if direction == "right":  
 rad = -rad  
 self.rotate\_angle = self.rotate\_angle + rad  
 self.angle = self.angle + rad  
 #self.move()  
 def moveTo(self,x,y):  
 self.x,self.y = x,y  
 self.draw()  
 def moveTowards(self,obj,speed):  
 self.setSpeed(speed,self.angleTo(obj))  
 self.move()  
 def rotateTowards(self,obj):  
 self.rotate\_angle = (self.angleTo(obj) + 90) \* math.pi / 180  
 self.rotate = "to"  
 #self.draw()  
 def rotateTo(self,angle):  
 self.rotate = "left"  
 self.rotate\_angle = angle \* math.pi / 180  
 self.angle = angle \* math.pi / 180  
 def angleTo(self,obj):  
 dx = obj.x - self.x  
 dy = obj.y - self.y  
 if dy == 0: #avoid division by zero  
 dy = 0.00001  
 angle = math.atan(dx/dy) \* 180 / math.pi  
 if dy > 0:  
 angle += 180  
 return angle  
 def setSpeed(self,speed,angle=-999):  
 if angle == -999:  
 angle = math.degrees(self.angle)   
 self.angle, self.speed = math.radians(angle), speed  
 self.calculateSpeedDeltas()  
 def calculateSpeedDeltas(self):  
 self.dx = self.speed \* math.sin(self.angle - math.pi)  
 self.dy = self.speed \* math.cos(self.angle - math.pi)  
 def makeVisible(self,visibility = -999):  
 if visibility == -999:  
 visibility = not self.visible  
 self.visible = visibility  
  
 def moveX(self,a):  
 self.x = self.x + a  
 def moveY(self,a):  
 self.y = self.y + a  
 def moveXY(self,a,b):  
 self.x = self.y + a  
 self.y = self.y + b  
  
 def resizeTo(self,w,h):  
 self.original = pygame.transform.scale(self.src,(int(w),int(h)))  
 self.image = self.original  
 self.width,self.height = self.image.get\_width(),self.image.get\_height()  
 self.oldwidth,self.oldheight = self.width,self.height  
 def resizeBy(self,pct):  
 factor = 1 + pct/100.0  
 self.resizeTo(int(self.width \* factor), int(self.height \* factor))  
  
 def isOffScreen(self,side="all"):  
 offscreen = False  
 if side=="all":  
 offscreen = self.right < self.game.left or self.left > self.game.right or self.top > self.game.bottom or self.bottom < self.game.top  
 elif side == "bottom":  
 offscreen = self.top > self.game.bottom  
 elif side == "top":  
 offscreen = self.bottom < self.game.top  
 elif side == "left":  
 offscreen = self.right < self.game.left  
 elif side == "right":  
 offscreen = self.left > self.game.right   
 return offscreen  
  
 def getAngle(self,angle="deg"):  
 if angle == "deg":  
 return math.degrees(self.angle)  
 return self.angle  
  
class Animation(Image):  
 def \_\_init\_\_(self,path,sequence,game, width = 0, height = 0,frate = 1,use\_alpha=True):  
 self.f, self.frate, self.ftick, self.loop, self.once = 0,frate,0,True,True  
 self.game = game  
 self.playAnim = True  
 self.images = []  
 self.source = []  
 if width == 0 and height == 0:  
 Image.\_\_init\_\_(self,path + "1.gif",game)  
 for i in range(sequence):  
 self.images.append(pygame.image.load(path + str(i+1) + ".gif").convert\_alpha())  
 self.source.append(self.images[i])  
 else:  
 if use\_alpha:  
 self.sheet = pygame.image.load(path).convert\_alpha()  
 else:  
 self.sheet = pygame.image.load(path).convert()  
 trans\_color = self.sheet.get\_at((0,0))  
 self.sheet.set\_colorkey(trans\_color)  
   
 tmp = self.sheet.subsurface((0,0,width,height))  
 Image.\_\_init\_\_(self,tmp,game)  
 self.frame\_width, self.frame\_height = width, height  
 self.frame\_rect = 0,0,width,height  
 try:  
 self.columns = self.sheet.get\_width() / width  
 except:  
 print("Wrong size sheet")  
 for i in range(sequence):  
 frame\_x = (i % self.columns) \* self.frame\_width  
 frame\_y = (i // self.columns) \* self.frame\_height  
 rect = ( frame\_x, frame\_y, self.frame\_width, self.frame\_height )  
 frame\_image = self.sheet.subsurface(rect)  
 self.images.append(frame\_image)  
 self.source.append(frame\_image)  
 def play(self):  
 self.playAnim = True  
 def stop(self):  
 self.playAnim = False  
 def nextFrame(self):  
 self.ftick += 1  
 if self.ftick % self.frate == 0:  
 self.f += 1  
 self.ftick = 0  
 if self.f > len(self.images)-1:  
 self.f = 0  
 self.draw()  
 def prevFrame(self):  
 self.ftick += 1  
 if self.ftick % self.frate == 0:  
 self.f -= 1  
 self.ftick = 0  
 if self.f < 0:  
 self.f = len(self.images)-1  
 self.draw()  
 def draw(self, loop = True):  
 if self.visible:  
 Image.setImage(self, self.images[self.f])  
 Image.draw(self)  
 self.ftick += 1  
 if self.ftick % self.frate == 0 and self.playAnim:  
 self.f += 1  
 self.ftick = 0  
 if not loop and self.f == len(self.images)-1:  
 self.visible = False  
 self.f = 0  
 if self.f > len(self.images)-1:  
 self.f = 0  
 self.ftick = 0  
   
 def rotateBy(self,angle=0,direction="right"):  
 Image.rotateBy(self,angle,direction)  
 return  
 def resizeTo(self,w,h):  
 self.width, self.height = w, h  
 for i in range(len(self.images)):  
 self.images[i] = pygame.transform.scale(self.source[i],(int(self.width),int(self.height)))  
 def resizeBy(self,pct):  
 factor = 1 + pct / 100.0  
 self.resizeTo(self.width \* factor, self.height \* factor)