import pygame # 匯入Pygame模組，用於開發遊戲

import sys # 匯入sys模組，用於退出遊戲

import random # 匯入random模組，用於生成隨機數值

# 初始化Pygame

pygame.init()

# 設定窗口大小

WINDOW\_WIDTH, WINDOW\_HEIGHT = 800, 600 # 定義遊戲窗口的寬度和高度

WINDOW = pygame.display.set\_mode((WINDOW\_WIDTH, WINDOW\_HEIGHT)) # 創建遊戲窗口

pygame.display.set\_caption("賽車遊戲") # 設定窗口的標題

# 定義顏色（RGB格式）

WHITE = (255, 255, 255) # 白色

BLACK = (0, 0, 0) # 黑色

RED = (255, 0, 0) # 紅色

GREEN = (0, 255, 0) # 綠色

BLUE = (0, 0, 255) # 藍色

YELLOW = (255, 255, 0) # 黃色

PURPLE = (128, 0, 128) # 紫色

ORANGE = (255, 165, 0) # 橙色

CYAN = (0, 255, 255) # 青色

# 設定賽車

CAR\_WIDTH, CAR\_HEIGHT = 40, 80 # 定義賽車的寬度和高度

car = pygame.Surface((CAR\_WIDTH, CAR\_HEIGHT)) # 創建賽車的Surface對象

car.fill(RED) # 填充賽車顏色為紅色

car\_x, car\_y = (WINDOW\_WIDTH // 2 - CAR\_WIDTH // 2, WINDOW\_HEIGHT - CAR\_HEIGHT - 10) # 設定賽車初始位置（在窗口底部中間偏左）

car\_speed = 15 # 設定賽車的左右移動速度

# 設定障礙物

OBSTACLE\_WIDTH, OBSTACLE\_HEIGHT = 50, 50 # 定義障礙物的寬度和高度

obstacles = [] # 障礙物列表，用於儲存所有生成的障礙物

obstacle\_speed = 20 # 設定障礙物向下移動的速度

obstacle\_spawn\_rate = 10 # 設定障礙物生成的頻率（每幀生成一次）

# 設定計時器

font = pygame.font.Font(None, 36) # 創建字體對象，字體大小為36

start\_ticks = pygame.time.get\_ticks() # 記錄遊戲開始的時間（毫秒）

def create\_obstacle():

"""

創建隨機形狀、顏色和位置的障礙物

"""

shape\_type = random.choice(['rect', 'circle', 'ellipse']) # 隨機選擇障礙物的形狀類型

color = random.choice([GREEN, BLUE, YELLOW, PURPLE, ORANGE, CYAN]) # 隨機選擇障礙物的顏色

x = random.randint(0, WINDOW\_WIDTH - OBSTACLE\_WIDTH) # 隨機生成障礙物的橫向位置

y = -OBSTACLE\_HEIGHT # 設定障礙物的縱向位置，使其從窗口上方外生成

if shape\_type == 'rect': # 如果形狀類型是矩形

width = random.randint(40, 60) # 隨機生成矩形的寬度

height = random.randint(40, 60) # 隨機生成矩形的高度

return {'type': 'rect', 'color': color, 'rect': pygame.Rect(x, y, width, height)} # 返回包含矩形信息的字典

elif shape\_type == 'circle': # 如果形狀類型是圓形

radius = random.randint(20, 30) # 隨機生成圓形的半徑

return {'type': 'circle', 'color': color, 'pos': (x + radius, y + radius), 'radius': radius} # 返回包含圓形信息的字典

elif shape\_type == 'ellipse': # 如果形狀類型是橢圓形

width = random.randint(30, 60) # 隨機生成橢圓的寬度

height = random.randint(30, 60) # 隨機生成橢圓的高度

return {'type': 'ellipse', 'color': color, 'rect': pygame.Rect(x, y, width, height)} # 返回包含橢圓信息的字典

def draw\_obstacles(obstacles):

"""

繪製所有障礙物

"""

for obstacle in obstacles: # 遍歷每個障礙物

if obstacle['type'] == 'rect': # 如果是矩形障礙物

pygame.draw.rect(WINDOW, obstacle['color'], obstacle['rect']) # 繪製矩形

elif obstacle['type'] == 'circle': # 如果是圓形障礙物

pygame.draw.circle(WINDOW, obstacle['color'], obstacle['pos'], obstacle['radius']) # 繪製圓形

elif obstacle['type'] == 'ellipse': # 如果是橢圓形障礙物

pygame.draw.ellipse(WINDOW, obstacle['color'], obstacle['rect']) # 繪製橢圓

def move\_obstacles(obstacles):

"""

移動所有障礙物

"""

for obstacle in obstacles: # 遍歷每個障礙物

if obstacle['type'] in ['rect', 'ellipse']: # 如果是矩形或橢圓形障礙物

obstacle['rect'].y += obstacle\_speed # 向下移動障礙物

elif obstacle['type'] == 'circle': # 如果是圓形障礙物

obstacle['pos'] = (obstacle['pos'][0], obstacle['pos'][1] + obstacle\_speed) # 向下移動障礙物的位置

def remove\_offscreen\_obstacles(obstacles):

"""

移除超出窗口的障礙物

"""

return [obstacle for obstacle in obstacles if obstacle['type'] in ['rect', 'ellipse'] and obstacle['rect'].y < WINDOW\_HEIGHT or

obstacle['type'] == 'circle' and obstacle['pos'][1] - obstacle['radius'] < WINDOW\_HEIGHT] # 只保留在窗口內的障礙物

def check\_collision(car\_rect, obstacles):

"""

檢查賽車是否與任何障礙物發生碰撞

"""

for obstacle in obstacles: # 遍歷每個障礙物

if obstacle['type'] == 'rect': # 如果是矩形障礙物

if car\_rect.colliderect(obstacle['rect']): # 檢查是否發生碰撞

return True # 發生碰撞，返回True

elif obstacle['type'] == 'circle': # 如果是圓形障礙物

circle\_rect = pygame.Rect(obstacle['pos'][0] - obstacle['radius'], obstacle['pos'][1] - obstacle['radius'],

obstacle['radius'] \* 2, obstacle['radius'] \* 2) # 創建包圍圓形的矩形

if car\_rect.colliderect(circle\_rect): # 檢查是否發生碰撞

distance = ((car\_rect.centerx - obstacle['pos'][0]) \*\* 2 + (car\_rect.centery - obstacle['pos'][1]) \*\* 2) \*\* 0.5 # 計算賽車中心與圓心之間的距離

if distance < obstacle['radius']: # 如果距離小於圓形的半徑，則發生碰撞

return True # 發生碰撞，返回True

elif obstacle['type'] == 'ellipse': # 如果是橢圓形障礙物

ellipse\_rect = pygame.Rect(obstacle['rect'].left, obstacle['rect'].top, obstacle['rect'].width, obstacle['rect'].height) # 創建橢圓的矩形邊框

if car\_rect.colliderect(ellipse\_rect): # 檢查是否發生碰撞

# 簡單的碰撞檢測，實際上應用更複雜的碰撞檢測算法

return True # 發生碰撞，返回True

return False # 未發生碰撞，返回False

def draw\_timer():

"""

顯示遊戲時間的計時器

"""

seconds = (pygame.time.get\_ticks() - start\_ticks) // 1000 # 計算遊戲開始以來的秒數

timer\_text = font.render(f"時間: {seconds}s", True, BLACK) # 渲染計時器文字

WINDOW.blit(timer\_text, (10, 10)) # 將計時器顯示在窗口的左上角

# 主遊戲循環

clock = pygame.time.Clock() # 創建一個時鐘對象，用於控制幀率

running = True # 遊戲運行狀態標誌

obstacle\_timer = 0 # 障礙物計時器，用於控制障礙物生成頻率

obstacles = [] # 清空障礙物列表

while running: # 遊戲主循環

for event in pygame.event.get(): # 遍歷所有事件

if event.type == pygame.QUIT: # 如果事件類型是退出

pygame.quit() # 退出Pygame

sys.exit() # 退出程式

# 取得鍵盤狀態

keys = pygame.key.get\_pressed() # 獲取當前按下的鍵

if keys[pygame.K\_LEFT]: # 如果按下左方向鍵

car\_x -= car\_speed # 向左移動賽車

if keys[pygame.K\_RIGHT]: # 如果按下右方向鍵

car\_x += car\_speed # 向右移動賽車

# 確保賽車不會移出窗口

car\_x = max(0, min(car\_x, WINDOW\_WIDTH - CAR\_WIDTH)) # 限制賽車的位置，使其不會超出窗口邊界

# 更新障礙物

obstacle\_timer += 1 # 增加障礙物計時器

if obstacle\_timer > obstacle\_spawn\_rate: # 如果計時器超過生成頻率

obstacle\_timer = 0 # 重置計時器

obstacles.append(create\_obstacle()) # 創建新障礙物並添加到列表中

move\_obstacles(obstacles) # 移動所有障礙物

obstacles = remove\_offscreen\_obstacles(obstacles) # 移除超出窗口的障礙物

# 檢查碰撞

car\_rect = pygame.Rect(car\_x, car\_y, CAR\_WIDTH, CAR\_HEIGHT) # 創建賽車的矩形邊框

if check\_collision(car\_rect, obstacles): # 如果發生碰撞

print("碰撞！遊戲結束。") # 在控制台輸出碰撞信息

pygame.quit() # 退出Pygame

sys.exit() # 退出程式

# 更新窗口

WINDOW.fill(WHITE) # 填充窗口背景為白色

WINDOW.blit(car, (car\_x, car\_y)) # 將賽車繪製到窗口上

draw\_obstacles(obstacles) # 繪製所有障礙物

draw\_timer() # 顯示計時器

pygame.display.update() # 更新窗口顯示

# 設定幀率

clock.tick(30) # 設定遊戲的幀率為30幀每秒