import pygame  
import math  
import random  
  
pygame.init()  
  
  
  
pygame.mixer.init()  
pygame.mixer.music.load(r"C:\Users\memos\Downloads\i-love-my-8-bit-game-console-301272.mp3")  
pygame.mixer.music.set\_volume(0.5) # optional: volume from 0.0 to 1.0  
pygame.mixer.music.play(-1) # -1 makes the music loop forever  
  
# Constants  
WIDTH, HEIGHT = 1500, 1000  
WIN = pygame.display.set\_mode((WIDTH, HEIGHT))  
pygame.display.set\_caption("AIR HOCKEY")  
  
FPS = 60  
  
BOX\_LEFT = 50  
BOX\_RIGHT = WIDTH - 50  
BOX\_TOP = 50  
BOX\_BOTTOM = HEIGHT - 50  
  
  
# Colors  
POWDER\_BLUE = (102,155,188)  
LINEN = (253,240,213)  
WHITE = (255, 255, 255)  
BLACK = (0, 0, 0)  
DARK\_RED = (218, 69, 39)  
BACKGROUND = POWDER\_BLUE# Light Blue  
RED = (255, 69, 58)  
DARK\_BLUE = (0, 48, 73)  
BLUE = (100, 149, 237)  
ORANGE = (234,103,45)  
GOAL\_COLOUR = DARK\_BLUE  
PADDLE1 = (120,0,0)  
PADDLE2 = (193,18,31)  
  
PADDLE\_WIDTH, PADDLE\_HEIGHT = 20, 150  
BALL\_RADIUS = 15  
GOAL\_RADIUS = 200  
GOAL\_THICKNESS = 30  
  
SCORE\_FONT = pygame.font.SysFont("Roboto", 50)  
WINNING\_SCORE = 10  
  
# Classes  
class Ball:  
 MAX\_VEL = 5  
 COLOUR = ORANGE  
  
 def \_\_init\_\_(self, x, y, radius):  
 self.x = self.original\_x = x  
 self.y = self.original\_y = y  
 self.radius = radius  
 self.x\_vel = self.MAX\_VEL  
 self.y\_vel = 0  
  
 def draw(self, win):  
 pygame.draw.circle(win, self.COLOUR, (self.x, self.y), self.radius)  
  
 def move(self):  
 self.x += self.x\_vel  
 self.y += self.y\_vel  
  
 def reset(self):  
 self.x = self.original\_x  
 self.y = self.original\_y  
 self.y\_vel = random.randint(1, 5)  
 self.x\_vel = -self.x\_vel if self.x\_vel != 0 else self.MAX\_VEL  
  
  
class Paddle:  
 VEL = 4  
  
 def \_\_init\_\_(self, x, y, width, height, colour):  
 self.x = self.original\_x = x  
 self.y = self.original\_y = y  
 self.width = width  
 self.height = height  
 self.COLOUR = colour  
  
 def draw(self, win):  
 pygame.draw.rect(win, self.COLOUR, (self.x, self.y, self.width, self.height), border\_radius=10)  
  
 def move\_y(self, up=True):  
 if up:  
 self.y -= self.VEL  
 else:  
 self.y += self.VEL  
  
 def move\_x\_left(self, move):  
 if move:  
 self.x -= self.VEL  
  
 def move\_x\_right(self, move):  
 if move:  
 self.x += self.VEL  
  
 def reset(self):  
 self.x = self.original\_x  
 self.y = self.original\_y  
  
  
class GoalArc:  
 def \_\_init\_\_(self, center, radius, start\_angle, stop\_angle,colour):  
 self.center = center  
 self.radius = radius  
 self.start\_angle = start\_angle  
 self.stop\_angle = stop\_angle  
 self.colour = colour  
  
 def draw(self, win):  
 rect = pygame.Rect(  
 self.center[0] - self.radius,  
 self.center[1] - self.radius,  
 self.radius \* 2,  
 self.radius \* 2  
 )  
 pygame.draw.arc(win, self.colour, rect, self.start\_angle, self.stop\_angle, GOAL\_THICKNESS)  
  
  
from collections import deque  
  
class AIPaddle(Paddle):  
 def \_\_init\_\_(self, x, y, width, height, colour, ball):  
 super().\_\_init\_\_(x, y, width, height, colour)  
 self.ball = ball  
  
 def bfs\_track\_puck(self):  
 start\_y = self.y  
 goal\_y = self.ball.y - self.height // 2  
 visited = set()  
 queue = deque()  
 queue.append((start\_y, []))  
  
 while queue:  
 current\_y, path = queue.popleft()  
  
 if abs((current\_y + self.height // 2) - self.ball.y) <= 5:  
 if path:  
 move = path[0]  
 if move == "up":  
 self.move\_y(up=True)  
 elif move == "down":  
 self.move\_y(up=False)  
 return  
  
 for direction in ["up", "down"]:  
 new\_y = current\_y - self.VEL if direction == "up" else current\_y + self.VEL  
 if 0 <= new\_y <= BOX\_BOTTOM - self.height and new\_y not in visited:  
 visited.add(new\_y)  
 queue.append((new\_y, path + [direction]))  
  
 def dfs\_track\_x(self):  
 stack = [(self.x, [])]  
 visited = set()  
  
 while stack:  
 current\_x, path = stack.pop()  
  
 if abs((current\_x + self.width // 2) - self.ball.x) <= 5:  
 if path:  
 move = path[0]  
 if move == "left":  
 self.move\_x\_left(True)  
 elif move == "right":  
 self.move\_x\_right(True)  
 return  
  
 for direction in ["left", "right"]:  
 new\_x = current\_x - 3 if direction == "left" else current\_x + 3  
  
 if WIDTH // 2 <= new\_x <= BOX\_RIGHT - self.width and new\_x not in visited:  
  
 visited.add(new\_x)  
 stack.append((new\_x, path + [direction]))  
  
  
# Draw Function  
def draw(win, paddles, ball, left\_score, right\_score, goals):  
 win.fill(BACKGROUND)  
  
  
  
 # Game Box  
 inner\_rect = pygame.Rect(50, 50, WIDTH - 100, HEIGHT - 100)  
 pygame.draw.rect(win, LINEN, inner\_rect, border\_radius=20)  
 pygame.draw.rect(win, DARK\_BLUE, inner\_rect, width=10, border\_radius=20)  
  
 # Score Display  
 pygame.draw.rect(win, WHITE, (WIDTH//2 - 150, 10, 300, 60), border\_radius=15)  
 left\_score\_text = SCORE\_FONT.render(f"{left\_score}", True, PADDLE1)  
 right\_score\_text = SCORE\_FONT.render(f"{right\_score}", True, PADDLE2)  
 win.blit(left\_score\_text, (WIDTH // 2 - 80, 20))  
 win.blit(right\_score\_text, (WIDTH // 2 + 40, 20))  
  
 for goal in goals:  
 goal.draw(win)  
  
 for paddle in paddles:  
 paddle.draw(win)  
  
 pygame.draw.rect(win, DARK\_BLUE, (WIDTH // 2 - 2, 50, 10, HEIGHT - 100))  
  
 def draw\_dashed\_line(surface, color, x, y\_start, y\_end, dash\_length=30, space\_length=10, width=10):  
 y = y\_start  
 while y < y\_end:  
 end\_y = min(y + dash\_length, y\_end)  
 pygame.draw.line(surface, color, (x, y), (x, end\_y), width)  
 y += dash\_length + space\_length  
  
 # Call it inside your draw() function  
 draw\_dashed\_line(win, DARK\_BLUE, WIDTH // 4 - PADDLE\_WIDTH // 2 + 5, 50, HEIGHT - 50)  
 draw\_dashed\_line(win, DARK\_BLUE, 3\*WIDTH // 4 - PADDLE\_WIDTH // 2 + 5, 50, HEIGHT - 50)  
  
 pygame.draw.circle(win, DARK\_BLUE, (WIDTH // 2, HEIGHT // 2), 100, 90)  
 pygame.draw.circle(win, LINEN, (WIDTH // 2, HEIGHT // 2), 85)  
  
 for goal in goals:  
 goal.draw(win)  
  
 for paddle in paddles:  
 paddle.draw(win)  
  
 ball.draw(win)  
 pygame.display.update()  
  
def handle\_collision(ball, left\_paddle, right\_paddle):  
 # Wall collisions (top & bottom)  
 if ball.y + ball.radius >= BOX\_BOTTOM or ball.y - ball.radius <= BOX\_TOP:  
 ball.y\_vel \*= -1  
  
 # Create Rects for paddles and ball  
 ball\_rect = pygame.Rect(ball.x - ball.radius, ball.y - ball.radius, ball.radius \* 2, ball.radius \* 2)  
 left\_rect = pygame.Rect(left\_paddle.x, left\_paddle.y, left\_paddle.width, left\_paddle.height)  
 right\_rect = pygame.Rect(right\_paddle.x, right\_paddle.y, right\_paddle.width, right\_paddle.height)  
  
 #checks for collision of the two rectangles everywhere  
 if ball\_rect.colliderect(left\_rect) :  
 #move ball just outside of the paddle  
 ball.x = left\_paddle.x + left\_paddle.width + ball.radius # move ball outside the paddle  
 ball.x\_vel \*= -1  
  
 #handles the angling of collision closer it is to the paddle center more angled collision  
 middle\_y = left\_paddle.y + left\_paddle.height / 2  
 diffence\_in\_y = middle\_y - ball.y  
 reduction\_factor = (left\_paddle.height / 2) / ball.MAX\_VEL  
 y\_vel = diffence\_in\_y / reduction\_factor  
 ball.y\_vel = -y\_vel  
  
 elif ball\_rect.colliderect(right\_rect):  
 ball.x = right\_paddle.x - ball.radius # move ball outside the paddle  
 ball.x\_vel \*= -1  
  
 middle\_y = right\_paddle.y + right\_paddle.height / 2  
 diffence\_in\_y = middle\_y - ball.y  
 reduction\_factor = (right\_paddle.height / 2) / ball.MAX\_VEL  
 y\_vel = diffence\_in\_y / reduction\_factor  
 ball.y\_vel = -y\_vel  
  
 #left\_up\_edge = pygame.rect(0,HEIGHT//2-GOAL\_RADIUS,0)  
  
  
  
  
def handle\_paddle\_movement(keys, left\_paddle, right\_paddle):  
 if keys[pygame.K\_w] and left\_paddle.y - left\_paddle.VEL >= BOX\_TOP:  
 left\_paddle.move\_y(up=True)  
 if keys[pygame.K\_s] and left\_paddle.y + left\_paddle.VEL + left\_paddle.height <= BOX\_BOTTOM:  
 left\_paddle.move\_y(up=False)  
 if keys[pygame.K\_d] and left\_paddle.x + left\_paddle.VEL + left\_paddle.width <= WIDTH // 2:  
 left\_paddle.move\_x\_right(move=True)  
 if keys[pygame.K\_a] and left\_paddle.x - left\_paddle.VEL >= BOX\_LEFT:  
 left\_paddle.move\_x\_left(move=True)  
  
 if keys[pygame.K\_UP] and right\_paddle.y - right\_paddle.VEL >= BOX\_TOP:  
 right\_paddle.move\_y(up=True)  
 if keys[pygame.K\_DOWN] and right\_paddle.y + right\_paddle.VEL + right\_paddle.height <= BOX\_BOTTOM:  
 right\_paddle.move\_y(up=False)  
 if keys[pygame.K\_RIGHT] and right\_paddle.x + right\_paddle.width + right\_paddle.VEL <= BOX\_RIGHT:  
 right\_paddle.move\_x\_right(move=True)  
 if keys[pygame.K\_LEFT] and right\_paddle.x - right\_paddle.VEL >= WIDTH // 2:  
 right\_paddle.move\_x\_left(move=True)  
  
  
  
def choose\_game\_mode():  
 *"""Display options for choosing the game mode."""* font = pygame.font.SysFont("Times New Roman", 40, bold=True)  
  
 header\_text = font.render("Choose Game Mode", True, WHITE)  
  
 option\_font = pygame.font.SysFont("Times New Roman", 30)  
 pvp\_text = option\_font.render("1. Player vs Player", True, WHITE)  
 pve\_text = option\_font.render("2. Player vs AI", True, WHITE)  
  
 WIN.fill(BACKGROUND)  
 WIN.blit(header\_text, (WIDTH//2 - header\_text.get\_width()//2, HEIGHT//3 - header\_text.get\_height()//2))  
 WIN.blit(pvp\_text, (WIDTH//2 - pvp\_text.get\_width()//2, HEIGHT//2 - pvp\_text.get\_height()//2))  
 WIN.blit(pve\_text, (WIDTH//2 - pve\_text.get\_width()//2, HEIGHT//2 + 40))  
  
 pygame.display.update()  
  
 waiting = True  
 while waiting:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 exit()  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_1:  
 return "PVP"  
 elif event.key == pygame.K\_2:  
 return "PVE"  
  
  
  
  
  
def show\_winner\_popup(left\_score, right\_score, winner\_text):  
 font = pygame.font.SysFont("Times New Roman", 60)  
  
 # 1) Render the final scores side by side  
 score\_text = f"Final Score – P1: {left\_score} | P2: {right\_score}"  
 score\_surf = font.render(score\_text, True, WHITE)  
 sx = WIDTH // 2 - score\_surf.get\_width() // 2  
 sy = HEIGHT // 3 - score\_surf.get\_height() // 2  
 WIN.fill(BACKGROUND)  
 WIN.blit(score\_surf, (sx, sy))  
 pygame.display.update()  
  
 # Pause so player can read the final score  
 pygame.time.delay(2000)  
  
 # 2) Render the winner message  
 winner\_surf = font.render(winner\_text, True, WHITE)  
 wx = WIDTH // 2 - winner\_surf.get\_width() // 2  
 wy = HEIGHT // 2 - winner\_surf.get\_height() // 2  
 WIN.fill(BACKGROUND)  
 WIN.blit(winner\_surf, (wx, wy))  
  
 # 3) Render the “Play Again?” prompt  
 prompt\_surf = font.render("Play Again? (Y/N)", True, WHITE)  
 px = WIDTH // 2 - prompt\_surf.get\_width() // 2  
 py = wy + winner\_surf.get\_height() + 40  
 WIN.blit(prompt\_surf, (px, py))  
  
 pygame.display.update()  
  
 # 4) Wait for Y/N  
 waiting = True  
 while waiting:  
 for e in pygame.event.get():  
 if e.type == pygame.QUIT:  
 pygame.quit()  
 exit()  
 if e.type == pygame.KEYDOWN:  
 if e.key == pygame.K\_y:  
 return True  
 if e.key == pygame.K\_n:  
 pygame.quit()  
 exit()  
 return False  
def show\_welcome\_screen():  
 font = pygame.font.SysFont("Times New Roman", 60, bold=True)  
 text = font.render("Welcome to AIR HOCKEY!", True, WHITE)  
 prompt = pygame.font.SysFont("Times New Roman", 30).render("Press any key to continue...", True, WHITE)  
  
 WIN.fill(BACKGROUND)  
 WIN.blit(text, (WIDTH // 2 - text.get\_width() // 2, HEIGHT // 3))  
 WIN.blit(prompt, (WIDTH // 2 - prompt.get\_width() // 2, HEIGHT // 2))  
 pygame.display.update()  
  
 waiting = True  
 while waiting:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 exit()  
 elif event.type == pygame.KEYDOWN:  
 waiting = False  
  
  
  
# Place this at the bottom of your script  
  
def show\_instructions():  
 WIN.fill(BACKGROUND)  
  
 title\_font = pygame.font.SysFont("Roboto", 48, bold=True)  
 text\_font = pygame.font.SysFont("Roboto", 28)  
  
 # Title  
 title\_text = title\_font.render("How To Play", True, WHITE)  
 WIN.blit(title\_text, (  
 WIN.get\_width() // 2 - title\_text.get\_width() // 2,  
 40  
 ))  
  
 instructions = [  
 "Player 1:",  
 "Use W and S to move up and down.",  
 "Use A and D to move back and front.",  
 "",  
 "Player 2:",  
 "Use Up and Down arrows to move up and down.",  
 "Use Left and Right arrows to move back and front.",  
 "",  
 "Press P to Pause, R to Restart, Q to Quit (during Pause).",  
 "Press Enter or Spacebar to start the game."  
 ]  
  
 # Calculate vertical start for centering  
 line\_height = 40  
 total\_text\_height = len(instructions) \* line\_height  
 start\_y = WIN.get\_height() // 2 - total\_text\_height // 2 + 30 # +30 for slight downward shift  
  
 # Render instructions  
 for i, line in enumerate(instructions):  
 text = text\_font.render(line, True, WHITE)  
 WIN.blit(text, (  
 WIN.get\_width() // 2 - text.get\_width() // 2,  
 start\_y + i \* line\_height  
 ))  
  
 pygame.display.flip()  
  
 # Wait for any key press  
 waiting = True  
 while waiting:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 exit()  
 elif event.type == pygame.KEYDOWN:  
 waiting = False  
  
# Main Game Loop  
def main():  
 while True:  
 clock = pygame.time.Clock()  
 show\_welcome\_screen()  
 mode = choose\_game\_mode()  
  
 # Show instructions before starting the game  
 show\_instructions() # Display the instructions popup  
  
 ball = Ball(WIDTH // 2, HEIGHT // 2, BALL\_RADIUS)  
 # Initialize paddles and ball after instructions  
 left\_paddle = Paddle(WIDTH // 4, HEIGHT // 2 - PADDLE\_HEIGHT // 2, PADDLE\_WIDTH, PADDLE\_HEIGHT, PADDLE1)  
  
 if mode == "PVE":  
  
 right\_paddle = AIPaddle(WIDTH \* 3 / 4, HEIGHT // 2 - PADDLE\_HEIGHT // 2, PADDLE\_WIDTH, PADDLE\_HEIGHT,  
 PADDLE2, ball)  
 else:  
 right\_paddle = Paddle(WIDTH \* 3 // 4, HEIGHT // 2 - PADDLE\_HEIGHT // 2, PADDLE\_WIDTH, PADDLE\_HEIGHT, PADDLE2)  
  
 left\_goal = GoalArc((BOX\_LEFT, HEIGHT // 2), GOAL\_RADIUS, math.radians(270), math.radians(90),DARK\_BLUE)  
 left\_goal\_border = GoalArc((BOX\_LEFT, HEIGHT // 2),GOAL\_RADIUS-25, math.radians(270), math.radians(90),BACKGROUND)  
  
 right\_goal = GoalArc((BOX\_RIGHT, HEIGHT // 2), GOAL\_RADIUS, math.radians(90), math.radians(270),DARK\_BLUE)  
 right\_goal\_border = GoalArc((BOX\_RIGHT, HEIGHT // 2), GOAL\_RADIUS-25,math.radians(90), math.radians(270),BACKGROUND)  
  
 goals = [left\_goal, right\_goal,left\_goal\_border,right\_goal\_border]  
  
 left\_score, right\_score = 0, 0  
 run = True  
 paused = False # Flag for pause state  
  
 while run:  
 clock.tick(FPS)  
  
 # Handle events including pause key and quit/restart menu  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 exit()  
 elif event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_p: # Toggle pause  
 paused = not paused  
 elif event.key == pygame.K\_r and paused: # Restart the game  
 left\_score, right\_score = 0, 0  
 ball.reset()  
 left\_paddle.reset()  
 right\_paddle.reset()  
 paused = False  
 elif event.key == pygame.K\_q and paused: # Exit the game  
 pygame.quit()  
 exit()  
  
 # Game logic runs only if not paused  
 if not paused:  
 keys = pygame.key.get\_pressed()  
 handle\_paddle\_movement(keys, left\_paddle, right\_paddle)  
  
 if mode == "PVE" and isinstance(right\_paddle, AIPaddle):  
 right\_paddle.bfs\_track\_puck()  
 right\_paddle.dfs\_track\_x()  
  
 ball.move()  
 handle\_collision(ball, left\_paddle, right\_paddle)  
  
 # Goal detection  
 dist\_to\_left = math.hypot(ball.x - 0, ball.y - HEIGHT // 2)  
 dist\_to\_right = math.hypot(ball.x - WIDTH, ball.y - HEIGHT // 2)  
  
 if dist\_to\_left <= GOAL\_RADIUS - ball.radius and ball.x < WIDTH // 2:  
 right\_score += 1  
 if right\_score == WINNING\_SCORE:  
 if not show\_winner\_popup(left\_score, right\_score, "Player 2 Wins!"):  
 return  
 run = False  
 break  
 ball.reset()  
 left\_paddle.reset()  
 right\_paddle.reset()  
 pygame.time.delay(500)  
  
 elif dist\_to\_right <= GOAL\_RADIUS - ball.radius and ball.x > WIDTH // 2:  
 left\_score += 1  
 if left\_score == WINNING\_SCORE:  
 if not show\_winner\_popup(left\_score, right\_score, "Player 1 Wins!"):  
 return  
 run = False  
 break  
 ball.reset()  
 left\_paddle.reset()  
 right\_paddle.reset()  
 pygame.time.delay(500)  
  
  
 elif ball.x - ball.radius <= BOX\_LEFT and dist\_to\_left > GOAL\_RADIUS:  
  
 ball.x = BOX\_LEFT + ball.radius + 1 # Prevent sticking  
  
 ball.x\_vel \*= -1  
  
  
 elif ball.x + ball.radius >= BOX\_RIGHT and dist\_to\_right > GOAL\_RADIUS:  
  
 ball.x = BOX\_RIGHT - ball.radius - 1  
  
 ball.x\_vel \*= -1  
  
 # Always draw, but overlay pause message if needed  
 draw(WIN, [left\_paddle, right\_paddle], ball, left\_score, right\_score, goals)  
  
 keys = pygame.key.get\_pressed()  
 if keys[pygame.K\_p]:  
 action = show\_pause\_menu()  
 if action == "restart":  
 main() # Restart the game  
 return  
 elif action == "resume":  
 continue  
  
 if not show\_play\_again():  
 break  
  
 pygame.quit()  
  
  
def show\_pause\_menu():  
 font = pygame.font.SysFont("Robotto", 50, bold=True)  
 small\_font = pygame.font.SysFont("Robotto", 30)  
  
 overlay = pygame.Surface((WIDTH, HEIGHT))  
 overlay.set\_alpha(200)  
 overlay.fill((0, 0, 0)) # Semi-transparent black  
  
 WIN.blit(overlay, (0, 0))  
  
 # Title  
 title\_text = font.render("Game Paused", True, WHITE)  
 WIN.blit(title\_text, (WIDTH // 2 - title\_text.get\_width() // 2, HEIGHT // 3))  
  
 # Instructions  
 instructions = [  
 "Press Enter to Resume",  
 "Press R to Restart the Game",  
 "Press Q to Quit the Game"  
 ]  
  
 for i, line in enumerate(instructions):  
 line\_surf = small\_font.render(line, True, WHITE)  
 WIN.blit(line\_surf, (WIDTH // 2 - line\_surf.get\_width() // 2, HEIGHT // 2 + i \* 40))  
  
 pygame.display.update()  
  
 # Wait for input  
 while True:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 exit()  
 elif event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_RETURN:  
 return "resume"  
 elif event.key == pygame.K\_r:  
 return "restart"  
 elif event.key == pygame.K\_q:  
 pygame.quit()  
 exit()  
  
  
def show\_play\_again():  
 font = pygame.font.SysFont("Times New Roman", 60)  
 play\_again\_text = font.render("Are you sure? (Y/N)", True, WHITE)  
 text\_rect = play\_again\_text.get\_rect(center=(WIDTH // 2, HEIGHT // 2))  
  
 # Clear the screen and display the question  
 WIN.fill(BACKGROUND)  
 WIN.blit(play\_again\_text, text\_rect)  
 pygame.display.update()  
  
 waiting = True  
 while waiting:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 exit()  
 if event.type == pygame.KEYDOWN:  
 if event.key == pygame.K\_y:  
 return True  
 elif event.key == pygame.K\_n:  
 return False  
  
 # Adding a delay to ensure that the display isn't updated too many times.  
 pygame.time.wait(100) # Wait for 100 milliseconds (you can adjust if needed)  
  
 return False # Default return value, if somehow no key is pressed  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()