```
import time
import pygame
import warnings
import torch
import os
import random
from ultralytics import YOLO
import math
import cv2
import logging
from ultralytics.utils import LOGGER
class SuppressDetectionLine(logging.Filter):
    def filter(self, record):
        # Suppress lines starting with frame index and resolution (e.g., "0:
480x640")
        return not record.getMessage().startswith("0: 480x640")
LOGGER.addFilter(SuppressDetectionLine())

#fix screen issues
#import ctypes
#ctypes.windll.user32.SetProcessDPIAware()

#yolo model setup
model = YOLO("C:/Users/andre/OneDrive/Documents/Visual Studio
Code/YoloModels/v4.pt")
model.conf = 0.7
model.to('cuda') #this uses gpu instead of cpu, as learned in the
optimization.py it works faster
cap = cv2.VideoCapture(0)
# Define the function of detecting boxes
def detect_boxes():
    #print("detecting boxes")
    ret, frame = cap.read()
    if not ret:
        return []

    results = model.predict(source=frame, conf=0.7, verbose=False)
    boxes = []

    for box in results[0].boxes:
        x1, y1, x2, y2 = map(int, box.xyxy[0].cpu().numpy())
        print("foundBox")
        boxes.append([x1, y1, x2, y2])
        #for (x1, y1, x2, y2) in boxes:
            #cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            #cv2.imshow("YOLO Detection", frame)
    #if not boxes:
        #print("no box found")
    return boxes


#pygame setup
os.environ['SDL_VIDEO_WINDOW_POS'] = "%d,%d" % (1920, 100)  # Adjust if needed
pygame.init()
pygame.display.set_caption("CamGame")
screen = pygame.display.set_mode((1920, 1080), pygame.FULLSCREEN)
clock = pygame.time.Clock()
running = True
dt = 0

#warnings
if not pygame.image.get_extended():
    warnings.warn("CAN'T LOAD IMAGES")
```

```
#calimage loader - used for calibration
calImage = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual Studio
Code/All Code Files/Handv2/sprites/calibrator.png")
calImageScaled = pygame.transform.scale(calImage, (128, 128))
ogBack = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual Studio
Code/All Code Files/Handv2/sprites/backarrow.png")
backArrow = pygame.transform.scale(ogBack, (60, 60))

#get the corners of the screen
def calibrate():
    corners = []

    # Top-left
    print("Step 1: Top-left")
    screen.fill("purple")
    screen.blit(calImageScaled, (0, 0))
    pygame.display.flip()
    pygame.event.pump()
    time.sleep(2)
    boxes = detect_boxes()
    while not boxes:
        print("Retrying top-left...")
        boxes = detect_boxes()
    corners.append([boxes[0][0], boxes[0][1]])

    # Top-right fakeout
    #print("Step 2: Top-right")
    screen.fill((255, 0, 150))  # PINK background
    screen.blit(calImageScaled, ((screen.get_width()/2),
(screen.get_height()/2)))  # move inward
    pygame.display.flip()
    pygame.event.pump()
    time.sleep(2)
    boxes = detect_boxes()
    while not boxes:
        #print("Retrying top-right...")
        boxes = detect_boxes()
    #corners.append([boxes[0][2], boxes[0][1]])

    # Top-right
    print("Step 2: Top-right")
    screen.fill((255, 0, 150))  # PINK background
    screen.blit(pygame.transform.scale(calImage, (128,128)), (screen.get_width()
- 128, 0))
    pygame.display.flip()
    pygame.event.pump()
    time.sleep(2)
    boxes = detect_boxes()
    while not boxes:
        print("Retrying top-right...")
        boxes = detect_boxes()
    corners.append([boxes[0][2], boxes[0][1]])

    # Bottom-left
    print("Step 3: Bottom-left")
    screen.fill((255, 0, 0))
    screen.blit(calImageScaled, (0, (screen.get_height() - 128)))
    pygame.display.flip()
    pygame.event.pump()
    time.sleep(2)
    boxes = detect_boxes()
    while not boxes:
        print("Retrying bottom-left...")
```

```
        boxes = detect_boxes()
    corners.append([boxes[0][0], boxes[0][3]])

    # Bottom-right
    print("Step 4: Bottom-right")
    screen.fill((255, 150, 150))
    screen.blit(calImageScaled, ((screen.get_width() - 128),
(screen.get_height() - 128)))
    pygame.display.flip()
    pygame.event.pump()
    time.sleep(2)
    boxes = detect_boxes()
    while not boxes:
        print("Retrying bottom-right...")
        boxes = detect_boxes()
    corners.append([boxes[0][2], boxes[0][3]])

    # Calculate bounding box
    bounds = []
    bounds.append([(corners[0][0]+corners[2][0])/2, (corners[1][0]+corners[3]
[0])/2])
    bounds.append([(corners[0][1]+corners[1][1])/2, (corners[2][1]+corners[3]
[1])/2])

    print("Calibration complete.")
    print(str(bounds[0][0]) + " , " + str(bounds[0][1]) + " , " + str(bounds[1]
[0]) + " , " + str(bounds[1][1]) + " , ")
    return bounds


def handLoc():
    ret, frame = cap.read()
    if not ret:
        return []


    results = model.track(source=frame, conf=0.7, stream=True, persist=True)
#model.predict actually detects frame by frame (cpu intensive) whereas
model.track 'keeps the engines running' so it doesnt have to startup every time
(and is more intensive on GPU which is not as impactful on the preformance)
    hands = []

    for r in results:
        for hand in r.boxes:
            xc, yc, w, h = map(int, hand.xywh[0].cpu().numpy())
            #print("handFound")
            class_id = int(hand.cls[0])
            t = model.names[class_id]
            hands.append([xc, yc, t, w, h])
            for (xc, yc, t, w, h) in hands:
                x1 = int(xc - w / 2)
                y1 = int(yc - h / 2)
                x2 = int(xc + w / 2)
                y2 = int(yc + h / 2)
                #cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                #cv2.imshow("YOLO Detection", frame)

    return hands

def map_value(x, in_min, in_max, out_min, out_max): #This is just a map()
function. I am used to arduino and it has this integrated, so this is a basic
script so i have it here too lol
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
```

```
#ret, frame = cap.read()
#cv2.rectangle(frame, (int(screenbounds[0][0]), int(screenbounds[1][0])),
(int(screenbounds[0][1]), int(screenbounds[1][1])), (0, 0, 255), 2)
#cv2.imshow("YOLO Detection", frame)


#this is the hand tracking program - Scene 1
player_pos = pygame.Vector2(screen.get_width() / 2, screen.get_height() / 2)
def handTrackProgram():
    global scene
    # fill the screen with a color to wipe away anything from last frame
    screen.fill("purple")

    pygame.draw.circle(screen, "red", player_pos, 40)

    #hands = [x center, y center, t ype, width, height]
    hands = handLoc()
    min_distance = float('inf')
    closest_hand_pos = None

    for hand in hands:
        if (screenbounds[0][0] <= hand[0] <= screenbounds[0][1] and
            screenbounds[1][0] <= hand[1] <= screenbounds[1][1]):

            clamped_x = max(min(hand[0], screenbounds[0][1]), screenbounds[0]
[0])
            clamped_y = max(min(hand[1], screenbounds[1][1]), screenbounds[1]
[0])

            handx = map_value(clamped_x, screenbounds[0][0], screenbounds[0][1],
0, screen.get_width())
            handy = map_value(clamped_y, screenbounds[1][0], screenbounds[1][1],
0, screen.get_height())

            if hand[2] == "openHand":
                dist = ((handx - player_pos.x) ** 2 + (handy - player_pos.y) **
2) ** 0.5
                if dist < min_distance:
                    min_distance = dist
                    closest_hand_pos = (handx, handy)
            if handx < 70 and handy < 70 and hand[2] == "closedHand":
                scene = 4

    if closest_hand_pos:
        player_pos.x += (closest_hand_pos[0]-player_pos.x) * 0.5
        player_pos.y += (closest_hand_pos[1]-player_pos.y) * 0.5
        #print(f"{player_pos.x}, {player_pos.y}")

    return

#load images
def TicTacImgLoader():
    global board, colorO, colorX, greyO, greyX
    ogBoard = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual Studio
Code/All Code Files/Handv2/sprites/TicTacs/TicTacBoard.png")
    board = pygame.transform.scale(ogBoard, (512, 512))
    ogColorO = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual
Studio Code/All Code Files/Handv2/sprites/TicTacs/colorO.png")
    colorO = pygame.transform.scale(ogColorO, (132, 132))
    ogColorX = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual
Studio Code/All Code Files/Handv2/sprites/TicTacs/colorX.png")
    colorX = pygame.transform.scale(ogColorX, (132, 132))
```

```
    ogGreyO = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual Studio
Code/All Code Files/Handv2/sprites/TicTacs/greyO.png")
    greyO = pygame.transform.scale(ogGreyO, (132, 132))
    ogGreyX = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual Studio
Code/All Code Files/Handv2/sprites/TicTacs/greyX.png")
    greyX = pygame.transform.scale(ogGreyX, (132, 132))
    return


def TicTacSetup():
    global move_locked, current_player, boardPlacement
    TicTacImgLoader()
    move_locked = False
    current_player = 1
    boardPlacement = [0]*9 #0 is none, 1 is x, 2 is o
    move_locked = False  # lock input while hand is closed
    current_player = 1  # 1 = X, 2 = O
    #screen.blit(board, ((screen.get_width()/2) - 256, (screen.get_height()/2) -
256))  # boardSpawn

    return 3


def check_winner(board):
    win_combinations = [
        [0,1,2], #top row
        [3,4,5], #middle row
        [6,7,8], #bottom row
        [0,3,6], #left column
        [1,4,7], #middle column
        [2,5,8], #right column
        [0,4,8], #diagonol 1
        [2,4,6] # diagonol 2 (i cant spell lol)
    ]
    for combo in win_combinations:
        a, b, c = combo
        if board[a] == board[b] == board[c] and board[a] != 0:
            return board[a] #return the winning symbol
    return None #no winnter


def TicTacRun():
    global move_locked, current_player, boardPlacement, scene
    # Check winner
    winner = check_winner(boardPlacement)
    if winner:
        print(f"Player {winner} wins!")
        time.sleep(2)
        TicTacSetup()  # Reset
        return
    elif 0 not in boardPlacement:
        print("Draw!")
        time.sleep(2)
        TicTacSetup()  # Reset
        return

    screen.fill("purple")
    #screen.fill((97, 132, 216))  # the background
    screen.blit(board, ((screen.get_width()/2) - 256, (screen.get_height()/2) -
256))  # boardSpawn
    hands = handLoc()
        # Board position
    board_left = (screen.get_width() / 2) - 256
    board_top = (screen.get_height() / 2) - 256
```

```
    cell_size = 170  # Since 512 / 3 = ~170

    for hand in hands:
        hand_x, hand_y, hand_type, w, h = hand #BTW, i like this way better.
instead of doing the [] thing with hands, i just dont wanna go back and change
it lol
        # Map hand coordinates to screen
        if screenbounds[0][0] <= hand_x <= screenbounds[0][1] and
screenbounds[1][0] <= hand_y <= screenbounds[1][1]:
            clamped_x = max(min(hand_x, screenbounds[0][1]), screenbounds[0][0])
            clamped_y = max(min(hand_y, screenbounds[1][1]), screenbounds[1][0])
            mapped_x = map_value(clamped_x, screenbounds[0][0], screenbounds[0]
[1], 0, screen.get_width())
            mapped_y = map_value(clamped_y, screenbounds[1][0], screenbounds[1]
[1], 0, screen.get_height())

            if mapped_x < 70 and mapped_y < 70 and hand_type == "closedHand":
                scene = 4

            # Determine which cell hand is over
            col = int((mapped_x - board_left) // cell_size)
            row = int((mapped_y - board_top) // cell_size)

            if 0 <= row < 3 and 0 <= col < 3:
                idx = row * 3 + col
                center_x = board_left + col * cell_size + cell_size // 2 - 66  #
Adjusted for icon size
                center_y = board_top + row * cell_size + cell_size // 2 - 66
                if boardPlacement[idx] == 0:
                    if hand_type == "openHand":
                        # Show preview
                        preview = greyX if current_player == 1 else greyO
                        screen.blit(preview, (center_x, center_y))
                        move_locked = False
                    elif hand_type == "closedHand" and not move_locked:
                        # Place move
                        move_locked = True
                        boardPlacement[idx] = current_player
                        placed = colorX if current_player == 1 else colorO
                        screen.blit(placed, (center_x, center_y))



                        # Switch player
                        current_player = 2 if current_player == 1 else 1

    for i in range(9):
        row = i // 3
        col = i % 3
        center_x = board_left + col * cell_size + cell_size // 2 - 66
        center_y = board_top + row * cell_size + cell_size // 2 - 66

        if boardPlacement[i] == 1:
            screen.blit(colorX, (center_x, center_y))
        elif boardPlacement[i] == 2:
            screen.blit(colorO, (center_x, center_y))

    return

def IconImgLoader():
    global TicTacIcon, PongIcon, HandTrackerIcon, AngryIcon
    ogTicTacIcon = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual
Studio Code/All Code Files/Handv2/sprites/Icons/tictacicon.png")
    TicTacIcon = pygame.transform.scale(ogTicTacIcon, (240, 240))
```

```
    ogPongIcon = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual
Studio Code/All Code Files/Handv2/sprites/Icons/pongicon.png")
    PongIcon = pygame.transform.scale(ogPongIcon, (240, 240))
    ogHandTrackerIcon =
pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual Studio Code/All Code
Files/Handv2/sprites/Icons/handtrackericon.png")
    HandTrackerIcon = pygame.transform.scale(ogHandTrackerIcon, (240, 240))
    ogangryIcon = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual
Studio Code/All Code Files/Handv2/sprites/Icons/angrybirdsicon.png")
    AngryIcon = pygame.transform.scale(ogangryIcon, (240, 240))
    return

def IconScreen():
    global scene
    screen.fill((97, 132, 216))  # the background
    screen.blit(TicTacIcon, ((screen.get_width()/2) - 40-240-120,
(screen.get_height()/2) - 120))  # leftmost iconSpawn
    screen.blit(PongIcon, ((screen.get_width()/2) - 120, (screen.get_height()/2)
- 120))  # midLeft iconSpawn
    screen.blit(HandTrackerIcon, ((screen.get_width()/2) + 40+120,
(screen.get_height()/2) - 120))  # midRight iconSpawn
    #screen.blit(AngryIcon, ((screen.get_width()/2) + 40+240+80,
(screen.get_height()/2) - 120))  # rightmost iconSpawn

    hands = handLoc()
    cell_size = 280

    for hand in hands:
        hand_x, hand_y, hand_type, w, h = hand
        left_box = (screen.get_width()/2)-420
        if screenbounds[0][0] <= hand_x <= screenbounds[0][1] and
screenbounds[1][0] <= hand_y <= screenbounds[1][1]:
            clamped_x = max(min(hand_x, screenbounds[0][1]), screenbounds[0][0])
            clamped_y = max(min(hand_y, screenbounds[1][1]), screenbounds[1][0])
            mapped_x = map_value(clamped_x, screenbounds[0][0], screenbounds[0]
[1], 0, screen.get_width())
            mapped_y = map_value(clamped_y, screenbounds[1][0], screenbounds[1]
[1], 0, screen.get_height())

            if ((screen.get_height()/2) - 120) <= mapped_y <=
((screen.get_height()/2) + 120) and ((screen.get_width()/2) - 40-240-120-20) <=
mapped_x <= ((screen.get_width()/2) + 40+240+20+120):
                # Determine which icon hand is over
                icon = int((mapped_x - left_box) // cell_size)
                print(icon)
                print(mapped_x)
                if hand_type == "closedHand":
                    if icon == 0:
                        scene = 2 #tictac
                    elif icon == 1:
                        scene = 5
                        print("pong")
                    elif icon == 2:
                        scene = 1 #handtracking
                    elif icon == 4:
                        #scene = whatever is angrybird
                        print("angrybird")
                    else:
                        print("error")


    return

def Pong_Setup():
```

```
    global paddle, left_player_pos, right_player_pos, ball_pos, ball_vel
    ogpaddle = pygame.image.load("C:/Users/andre/OneDrive/Documents/Visual
Studio Code/All Code Files/Handv2/sprites/pong/paddle.png")
    paddle = pygame.transform.scale(ogpaddle, (21, 93)) #x3

    print("width")
    print(screen.get_width())
    print("height")
    print(screen.get_height())

    left_player_pos = pygame.Vector2(70,(screen.get_height()/2)-(310/2))
    right_player_pos = pygame.Vector2(screen.get_width() - 140,
(screen.get_height()/2)-(310/2))
    angle = random.randint(-45,45)
    speed = 20
    ball_vel = pygame.Vector2(speed*math.cos(math.radians(angle)),
speed*math.sin(math.radians(angle)))
    ball_pos = pygame.Vector2(screen.get_width()/2,screen.get_height()/2)


def reflect_ball(ball_pos, paddle_pos, is_left):
    # Center of paddle
    paddle_center_y = paddle_pos.y + 155 / 2
    # Difference from center (normalized from -1 to 1)
    relative_intersect = (ball_pos.y - paddle_center_y) / (155 / 2)
    relative_intersect = max(min(relative_intersect, 1), -1)  # clamp

    # Max angle from horizontal (like a curve)
    max_bounce_angle = math.radians(60)  # adjust for more or less curvature

    bounce_angle = relative_intersect * max_bounce_angle

    speed = ball_vel.length()
    new_dir_x = math.cos(bounce_angle)
    new_dir_y = math.sin(bounce_angle)

    # Flip x direction depending on which paddle
    if not is_left:
        new_dir_x *= -1

    return pygame.Vector2(new_dir_x * speed, new_dir_y * speed)


def Pong_Run():
    global scene, paddle, left_player_pos, right_player_pos, ball_pos, ball_vel
    screen.fill((108,110,160))
    screen.blit(paddle, left_player_pos)
    screen.blit(paddle, right_player_pos)
    keys = pygame.key.get_pressed()
    if keys[pygame.K_w]:
        left_player_pos.y -= 600 * dt
    if keys[pygame.K_s]:
        left_player_pos.y += 600 * dt
    if keys[pygame.K_DOWN]:
        right_player_pos.y += 600 * dt
    if keys[pygame.K_UP]:
        right_player_pos.y -= 600 * dt

    hands = handLoc()

    for hand in hands:
        hand_x, hand_y, hand_type, w, h = hand

        if screenbounds[0][0] <= hand_x <= screenbounds[0][1] and
```

```
screenbounds[1][0] <= hand_y <= screenbounds[1][1]:
            clamped_x = max(min(hand_x, screenbounds[0][1]), screenbounds[0][0])
            clamped_y = max(min(hand_y, screenbounds[1][1]), screenbounds[1][0])
            mapped_x = map_value(clamped_x, screenbounds[0][0], screenbounds[0]
[1], 0, screen.get_width())
            mapped_y = map_value(clamped_y, screenbounds[1][0], screenbounds[1]
[1], 0, screen.get_height())
            if mapped_x < 70 and mapped_y < 70 and hand_type == "closedHand":
                scene = 4

            if mapped_x < 280:
                left_player_pos.y = mapped_y - 45
            if mapped_x > screen.get_width() - 280:
                right_player_pos.y = mapped_y - 45


    pygame.draw.circle(screen, (255, 190, 11), ball_pos, 20)
    ball_pos = pygame.Vector2(ball_pos.x + ball_vel.x,ball_pos.y + ball_vel.y)

    if ball_pos.y < 22 or ball_pos.y > screen.get_height() - 22:
        ball_vel.y *= -1
    if ball_pos.x < 22 or ball_pos.x > screen.get_width() - 22:
        scene = 5

    ball_rect = pygame.Rect(ball_pos.x - 20, ball_pos.y -20, 40, 40) # ball
radius = 20
    left_rect = pygame.Rect(left_player_pos.x, left_player_pos.y, 21, 93)
    right_rect = pygame.Rect(right_player_pos.x, right_player_pos.y, 21, 93)

    if ball_rect.colliderect(left_rect):
        ball_vel = reflect_ball(ball_pos, left_player_pos, True)

    if ball_rect.colliderect(right_rect):
        ball_vel = reflect_ball(ball_pos, right_player_pos, False)

running = True
scene = 5 #4 is the icon screen
screenbounds = []
screenbounds = calibrate()
IconImgLoader()

while running:
    # poll for events
    # pygame.QUIT event means the user clicked X to close your window
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False


    if scene == 1:
        handTrackProgram()
    elif scene == 2:
        scene = 3
        TicTacSetup()
        print('TicTac Setup')
    elif scene == 3:
        TicTacRun()
        #print('TicTac Run')
    elif scene == 4:
        IconScreen()
        #print('IconMenu')
    elif scene == 5:
        scene = 6
        Pong_Setup()
```

```
        print('Pong Setup')
    elif scene == 6:
        Pong_Run()
        #print('pong run')
    else:
        print("Error")


    if scene != 4:
        screen.blit(backArrow, (4, 0))  #back arrow
    # flip() the display to put your work on screen
    pygame.display.flip()

    # limits FPS to 60
    # dt is delta time in seconds since last frame, used for framerate-
    # independent physics.
    dt = clock.tick(60) / 1000

# Cleanup
cap.release()
cv2.destroyAllWindows()
pygame.quit()
```