DOCUMENTACION **FlappyBird_&_HandRecognition**

Este proyecto es una referencia al típico juego llamado Flappy Bird, pero se controla por medio de el reconocimiento de manos, más específicamente con el pulgar e índice, mediante OpenCV y Mediapipe, además de que el juego se realizó mediante Pygame.

Inicializamos ciertas variables de manera global, pero

SCREEN_WIDTH -> Es el tamaño (ancho) de la pantalla

SCREEN_HEIGHT -> Tamaño (alto) de la pantalla

SPEED -> Velocidad de salto para el pájaro

Nota: cuando hace el bump es el -SPEED (salta)

GRAVITY -> Gravedad

GAME_SPEED -> Esta es la velocidad del juego en general (suelo y tuberías)

GROUND_HEIGHT -> Altura del suelo

PIPE_WIDTH -> Ancho de la tuberia

PIPE_HEIGHT -> Alto de la tubería

PIPE_GAP -> Espacio entre tuberías (se tuvo que modificar para que el juego sea amigable)

Variables de configuración para pygame

screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT)) -> Se inicializa con los tamaños de las variables globales

pygame.display.set_caption(‘’) -> Nombre asignado a la ventana

BLUE=(77, 192, 202, 255) -> Asignación de el color (tomando como referencia el fondo de la imagen)

WHITE=(255, 255, 255) -> color del score

background = pygame.image.load('fondoCielo.png') -> carga la imagen en este caso el fondo

background = pygame.transform.scale(background, (SCREEN_WIDTH, SCREEN_HEIGHT)) -> tiene las escalas de toda la pantalla.

bird = pygame.image.load('pajaroTieso.png') -> carga del pajaro

bird = pygame.transform.scale(bird, (34, 24)) -> lo carga con x size

pipe_img = pygame.image.load('tuberia.png') -> carga la imagen de la tuberia

pipe_img = pygame.transform.scale(pipe_img, (PIPE_WIDTH, PIPE_HEIGHT)) -> se adapta a los valores de las variables globales enfocadas a la tuberia

pipe_inverted_img = pygame.transform.flip(pipe_img, False, True) -> giramos la imagen para la tubería invertida

grounds = pygame.image.load('base.png') -> carga el suelo

grounds = pygame.transform.scale(grounds, (SCREEN_WIDTH, GROUND_HEIGHT)) -> con los tamaños designados

#rect[0] posicion de X, rect[1] posicion en y, rect[2] ancho y rect[3] para altura

Clase Bird:

self.image = pygame.Surface((34, 24), pygame.SRCALPHA) ->creamos superficie transparente del pájaro

self.image.blit(bird, (0, 0)) ->dibujamos la imagen del pajaro DENTRO del espacio designado (desde el punto 0,0 (la esquina))

self.speed = 0 -> velocidad inicial

self.rect = self.image.get_rect() -> crea un rectángulo que envuelve al pajaro con el fin de detectar colisiones (con el suelo o con las tuberías)

self.rect[0] = SCREEN_WIDTH / 6 -> posición horizontal del pájaro

self.rect[1] = SCREEN_HEIGHT / 2 ->posición vertical del pájaro

self.image.convert_alpha() ->convertimos la imagen a formato alpha para que sea transparente

Métodos de Bird

Update: Aumenta la velocidad y mueve el pájaro hacia abajo

Bump: Cuando salta, invierte el valor (con el que salta)

#rect[0] posicion de X, rect[1] posicion en y, rect[2] ancho y rect[3] para altura

Clase Pipe:

self.image = pygame.Surface((PIPE_WIDTH, PIPE_HEIGHT), pygame.SRCALPHA) -> Carga el fondo (superficie)

if inverted:

self.image.blit(pipe_inverted_img,(0,0)) -> llama a la imagen invertida declarada arriba

else:

self.image.blit(pipe_img, (0, 0)) -> si no manda la versión normal (original)

self.rect = self.image.get_rect() -> crea el espacio que envuelve, en donde se detecta si hay colisiones, siendo esta la más importante

self.rect[0] = xpos -> esto es la posicion del de la tubería den x (más adelante se ve como se generan las tuberias de manera aleatoria)

self.passed = False -> se inicializa en Falso, es un variable para indicar si ya paso o no

if inverted:

self.rect[1] =-(self.rect[3] -ysize) -> Es el tamaño, pero invertido (en negativo) tomando en cuenta la posición en cuanto a la altura (para dar espacio para que el pájaro pase=

else:

self.rect[1] = SCREEN_HEIGHT-ysize -> el tamaño total – el tamaño en y para saber el tamaño original de la tuberia

Método de Pipe:

Update -> el único update como tal es el movimiento en dirección contraria (izquierda a derecha)

#rect[0] posicion de X, rect[1] posicion en y, rect[2] ancho y rect[3] para altura

Clase Ground

self.image = pygame.Surface((SCREEN_WIDTH, GROUND_HEIGHT), pygame.SRCALPHA) -> se carga la superficie

self.image.blit(grounds, (0,0)) ->Se le pega la imagen a dicha superficie

self.rect = self.image.get_rect() -> es un un espacio que cubre (con el cual se pude generar inclusive colisiones[que es lo que busco])

self.rect[0] =xpos ->posicion horizontal(aun que directamente pudo ser el size de la pantalla)

self.rect[1] = SCREEN_HEIGHT - GROUND_HEIGHT -> para asignarle el tamaño (en y)

Método de Ground:

Update: Lo que se creo (el suelo) se mueve a la derecha

Extras:

OffScreen: Es para saber cuándo el ancho completo de la tubería sale de la pantalla (retorna true o false dependiendo)

RandomSize: Genera un numero aleatorio (en este caso de 100 y 300) que representa el tamaño de la tubería de abajo

pipe = Pipe(False, xpos, size) -> se le asigna al pipe el tamaño, recordando que es True en el primer término si la tubería esta normal (no invertida)

pipe_inverted = Pipe(True, xpos, SCREEN_HEIGHT - size - PIPE_GAP) -> cuando está invertida (True), la posición es el tamaño total – el tamaño(recién sacado para la no invertida) – el tamaño (predefinido como variable global) del espacio en “blanco” en donde pasara el pajaro

Clase Score:

self.score = 0 -> Claramente debe inicar en 1

self.font = pygame.font.Font(None, 25) -> fuente de las letras (de tamaño 25) y el none es la default.

self.image = pygame.Surface((100, 50), pygame.SRCALPHA) -> Crea una superficie de x,y tamaño

self.rect = self.image.get_rect() -> posiciona el sprite en el juego

self.rect.topleft = (5, 5) -> esto es donde estará ubicado (esquina superior izquierda [cerca del 0,0])

Método de score:

Update:

self.text = self.font.render("SCORE: "+str(int(self.score)), 1, WHITE) -> se va renderizando, el titulo (SCORE) más el score como tal, el 1 es son para bordes suaves y el color blanco que declare arriba.

self.image = self.text -> esto va actualizando el texto

Clase DetectarManos (este se debe desglosar muy a detalle):

def __init__(self, mode=False, max_hands=1, detection_confidence=0.5, tracking_confidence=0.5):

mode: Trata la imagen como algo en movimiento

max_hands cantidad máxima de manos a detectar (no se suele usar 2 manos así que por eficiencia lo deje así)

detection_confidence es el mínimo para encontrar la mano, poner menos puede afectar

trancking_confidence confianza mínima para el seguimiento de manos

self.mp_hands = mp.solutions.hands -> importa la libreria de mediapipe para detectar manos

self.tip = [4, 8] -> los landmarks, hace referencia a el pulgar (thumb_tip) y al indicie (index_finger_tip)

self.lista_landmarks = []

Métodos de DetectarManos:

Detectar_manos -> Recibe un frame

frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) -> se trata el frame con formato RGB, usa BGR por default pero mediapipie trata con RGB.

self.resultados = self.hands.process(frame_rgb) -> Es para detectar en el frame si hay una mano

```
if resultados.multi_hand_landmarks:
    for hand_landmarks in resultados.multi_hand_landmarks:
        for id in self.tip: #para cada mano y para cada ID
            lm = hand_landmarks.landmark[id] #obtiene las posiciones lm.x y y
            h, w, _ = frame.shape #saca los valores de alto y ancho del frame
            cx, cy = int(lm.x * w), int(lm.y * h) # y los inormaliza (que esto es como convertirlos a pixeles)
            self.lista_landmarks.append([id, cx, cy]) #retorna la lista con el id y las posiciones
```

Calcular_distancia -> calcula la distancia entre los puntos de referencia

if len(self.lista_landmarks) <2: -> si no se detectaron MINIMO 2 puntos

return float('inf') -> retorna infinito, esto es adrede para que no se pueda medir

x1, y1 = self.lista_landmarks[0][1:] -> en caso contrario, extrae las coordenadas de los puntos de interés (en este caso para 4(pulgar))

x2, y2 = self.lista_landmarks[1][1:] -> para 8 (índice)

return math.hypot(x2 - x1, y2 -y1) -> distancia x,y entre los dos dedos, con hypot, de math, el cual es la distancia euclidiana (meramente un Pitágoras), y es como una media solo para saber la distancia

MAIN:

pygame.sprite.Group() -> crea un grupo vacio de sprites, esto se repite para cada clase relacionada con el juego

bird = Bird() ->instancia de la clase Bird

bird_group.add(bird) -> Se agrega al grupo el bird

for i in range(2): #para ground

ground = Ground(SCREEN_WIDTH *i) ->dos suelos, uno detras del otro (para que parezca que el suelo tiene movimiento)

ground_group.add(ground) -> se agrega el suelo al grupo

for i in range(2): #para pipes

pipes = randomSize(SCREEN_WIDTH *i+800) -> generamos tubos (es 800 para que inice fuera de la vista del jugador)

pipe_group.add(pipes[0])

pipe_group.add(pipes[1])

score_group = pygame.sprite.Group()

score_group.add(score)

clock = pygame.time.Clock() -> es un controlador de FPS

game_started = False -> Evita que el juego empiece (ya que se tarda en cargar el juego por la detección)

cap = cv2.VideoCapture(0) -> abrimos la cámara

detector =DetectarManos(detection_confidence=0.5, tracking_confidence=0.5) -> llamamos a DetectarManos

while True:

success, frame = cap.read() #leemos frame de la camara

detector.detectar_manos(frame) -> manda a llamar detectar_manos madandole el frame

distance = detector.calcular_distancia() # medimos distancia entre dedos

if distance < 30: #el resultado debe ser menor a 30

game_started =True #empieza el juego

bird.bump() #cuando la distancia es menor a 30 el pájaro salta

for event in pygame.event.get():

if event.type == QUIT: #si se cierra la ventana se termina el juego

pygame.quit()

cap.release()

cv2.destroyAllWindows()

sys.exit() #se destruye todo

screen.fill(BLUE) -> asignación de color azul

screen.blit(background, (0, 0)) -> dibujamos fondo

if game_started: -> si el juego inició, se mandan a llamar los update de todas las clases

bird_group.update()

ground_group.update()

pipe_group.update()

score_group.update()

if offScreen(ground_group.sprites()[0]): ->si un suelo se va, lo reemplazamos

ground_group.remove(ground_group.sprites()[0]) -> se eliminia ese suelo

new_ground = Ground(SCREEN_WIDTH - 20) -> se crea otro

ground_group.add(new_ground) -> se agrega

if offScreen(pipe_group.sprites()[0]): ->si un tubo se va, generamos nuevos

pipe_group.remove(pipe_group.sprites()[0])

pipe_group.remove(pipe_group.sprites()[0])

pipes = randomSize(SCREEN_WIDTH * 2)

pipe_group.add(pipes[0])

pipe_group.add(pipes[1])

for pipe in pipe_group:

if pipe.rect[0] + PIPE_WIDTH < bird.rect[0] and not pipe.passed:

pipe.passed = True -> ya se paso

score.score +=0.5 ->hice la novatada de incrementar 0.5

if pygame.sprite.groupcollide(bird_group, pipe_group, False, False) or pygame.sprite.groupcollide(bird_group, ground_group, False, False): -> si el Pajaro choca contra el pipe o con el ground

return -> se sale del main y termina

#se dibujan los elementos en la pantalla y se actualizan los nuevos dibujos

bird_group.draw(screen)

pipe_group.draw(screen)

ground_group.draw(screen)

score_group.draw(screen)

pygame.display.update()

clock.tick(60) -> esto es un limitante de frames, en este caso 60

#al salir se elimina todo