**FLAPPY BIRD**

***“Pájaro Flojo”***

**INTEGRANTES**

**GALLINA, JUAN IGNACIO**

**ALCIBAR, FACUNDO**

**FERNANDEZ, FRANCESCO**

**CASAMAYU, JUANSE**

**VALLEJO, BAUTISTA**

**ASIGNATURA Y FORMATO**

**TECNOLOGÍA DE CONTROL**

**PYTHON**

**CURSO Y AÑO LECTIVO**

**7A 2025**



**ÍNDICE**

**1. INTRODUCCIÓN………………………………………………………………...2**

**2. DESARROLLO…………………………………………………………………..3**

**3. ASPECTO DEL JUEGO………… ……………………………………………..7**

**4. CODIGO……………………………………………………..……………………8**

**5. CONCLUSIÓN………………………………………………………………….16**

**INTRODUCCIÓN**

En el marco del segundo cuatrimestre del presente año, se nos presentó el desafío de encabezar un proyecto codificado en Python. Entre todos los proyectos que se nos vinieron a la cabeza, fruto de mucha investigación e interés en el tema, y algunas propuestas que nos cedió el docente, dimos con la idea y el interés de crear un videojuego al estilo del Flappy Bird, con una interfaz gráfica utilizando pygame. Este fue llamado “Pájaro Flojo”, en honor y parodia a la traducción del nombre del juego original en inglés. En este informe iremos detallando el proceso de escritura del código, las fuentes que utilizamos, entre otras cuestiones. Y se ira demostrando el proceso de creación. Asimismo, se adjunta un link que deriva a un repositorio en GitHub, el cual almacenara todo el proceso y los códigos correspondientes.

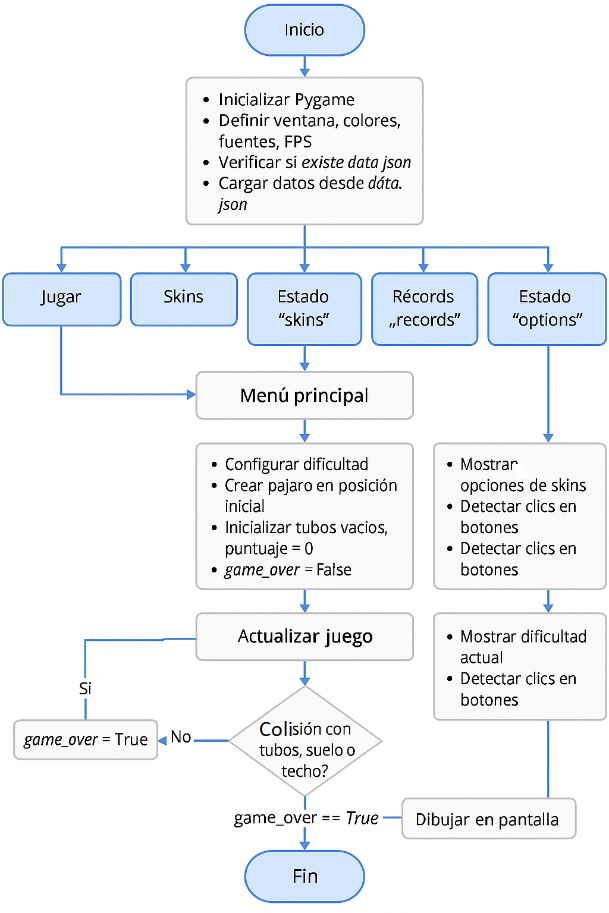
Link de repositorio: <https://github.com/juangallina/FLAPPY-BIRD-pajaro-flojo-.git>

**DESARROLLO**

Desde el primer momento en que nos planteamos hacer este proyecto, nuestra meta fue crear un juego que no únicamente mueva a un pájaro saltando entre tubos, sino que además tenga un menú de pausa, donde se puedan elegir skins, ver récords, cambiar la dificultad; una estética arcade y atractiva; y sobre todas las cosas, sea cómodo y agradable de jugar.

Semana a semana fuimos realizando avances hasta dar con el resultado final. Cabe aclarar que para que el juego se pueda ejecutar y utilizar de manera correcta, tuvimos que contar con la biblioteca pygame, la cual utilizando la biblioteca SDL (Simple DirectMedia Layer) nos proporcionó herramientas para manejar gráficos, sonidos, eventos del usuario (teclado, mouse, joystick) y detectar colisiones.

La idea fue que el gameplay se base únicamente en una acción tan simple como tocar la barra espaciadora o hacer click izquierdo para hacer al pájaro saltar. Y que, en función de esto, a medida que se vayan pasando los tubos, se cuente un puntaje. Este puntaje es posible verlo en un apartado del menú, así como el cambio de dificultades, el cambio de skins, y la posibilidad de salir del juego haciendo un click. El diagrama de flujo del código es el siguiente:



La explicación del flujo del juego es la siguiente:

1. Inicialización

Se importan las librerías necesarias: pygame, sys, os, json, random, math y deque. Se inicializa Pygame con pygame.init(). Se define la ventana del juego con dimensiones ANCHO = 480 y ALTO = 720. Se establecen fuentes y colores para la interfaz. Se crea o carga un archivo data.json para guardar:

* Puntaje más alto (highscore)
* Últimos puntajes (last\_scores)
* Skin seleccionada
* Dificultad del juego

Funciones importantes:

* load\_data(): carga los datos desde data.json.
* save\_data(d): guarda los datos en data.json.

2. Definición de clases principales

a) Botón (Button): Representa un botón en la interfaz.

Propiedades:

* Posición, tamaño, color normal y color al pasar el mouse.
* Acción que ejecuta al hacer clic.

Métodos:

* draw(surf): dibuja el botón en la superficie surf.
* click(pos): ejecuta la acción si se hace clic sobre el botón.

b) Pájaro (Bird): Representa al personaje principal.

Propiedades:

* Posición (x, y), velocidad vertical vy, radio y skin.
* Estado alive para indicar si sigue vivo.

Métodos:

* flap(): realiza un salto hacia arriba modificando la velocidad.
* update(dt): aplica gravedad y actualiza la posición.
* draw(surf): dibuja el pájaro en la ventana.
* rect: retorna el rectángulo de colisión del pájaro.

c) Tubo (Tube): Representa los obstáculos del juego.

Propiedades:

* Posición horizontal x, posición y altura del hueco gap\_y, gap\_h.
* passed: indica si el pájaro ya pasó el tubo.

Métodos:

* update(speed): mueve el tubo hacia la izquierda.
* draw(surf): dibuja el tubo superior e inferior.
* collides(bird): verifica colisión con el pájaro.

3. Lógica del juego (play\_game): Se define la dificultad según la configuración (Easy, Normal, Hard) ajustando:

* Tamaño del hueco entre tubos (gap)
* Velocidad de los tubos (speed)
* Intervalo de aparición de tubos (spawn\_interval)

Se inicializa el pájaro, la lista de tubos, temporizador de aparición y puntaje.

Bucle principal de juego:

* Procesa eventos de teclado y mouse:
* Espacio o clic → pájaro aletea.
* Escape → salir del juego.
* Actualiza posición del pájaro y genera nuevos tubos según el temporizador.
* Verifica colisiones con tubos y suelo.
* Actualiza puntaje si pasa un tubo.

Dibuja todos los elementos en pantalla: Fondo celeste, nubes, tubos, suelo, pájaro, puntaje y mensaje de “PERDISTE” si es necesario.

4. Menús y estados del juego:

El juego utiliza un sistema de estados (state) para alternar entre: Menu principal (state="menu"):

Botones: Jugar, Skins, Récords, Opciones, Salir.

Pantalla de Skins (state = "skins"): Permite elegir el color del pájaro.

Pantalla de Récords (state = "records"): Muestra el puntaje más alto y los últimos 5 puntajes.

Pantalla de Opciones (state = "options"): Permite cambiar la dificultad entre Easy, Normal y Hard.

Funciones clave:

* choose\_skin(name): cambia la skin y vuelve al menú.
* toggle\_difficulty(): alterna la dificultad y guarda los cambios.

5. Bucle principal (main). Mientras el juego esté corriendo:

* Procesa eventos de Pygame.
* Dibuja la pantalla según el estado actual (menu, play, skins, records, options).
* Llama a play\_game() cuando el jugador inicia una partida.
* Actualiza los récords y guarda datos al finalizar cada partida.
* Renderiza botones, títulos, puntajes y elementos gráficos según el estado.

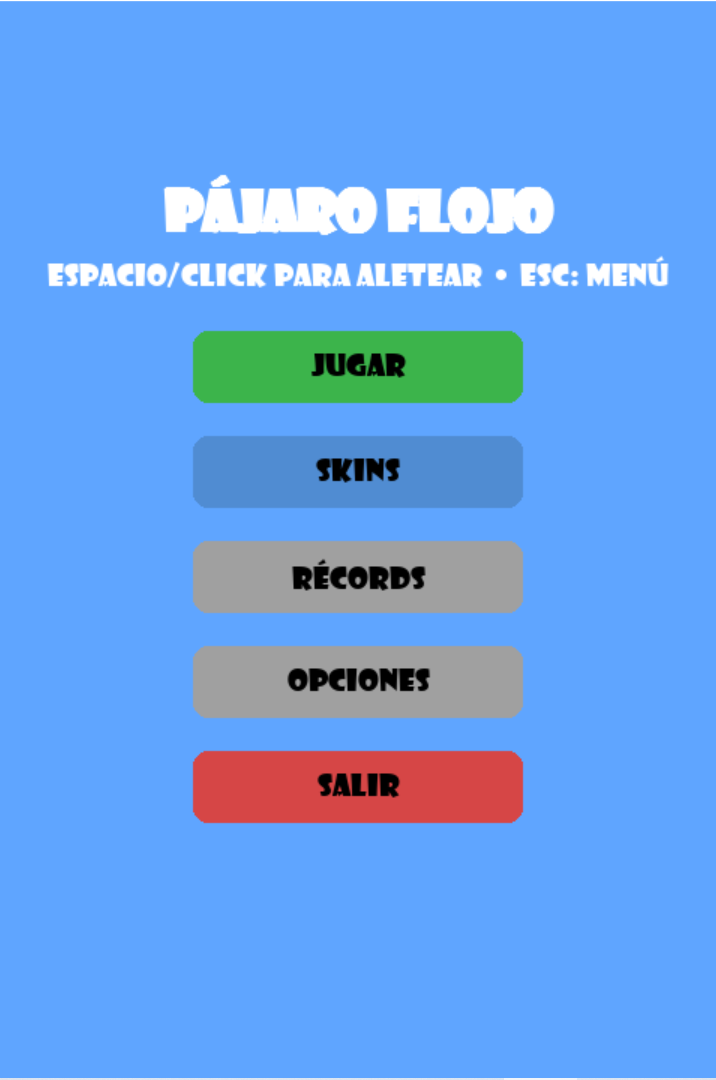
6. Resumen del flujo

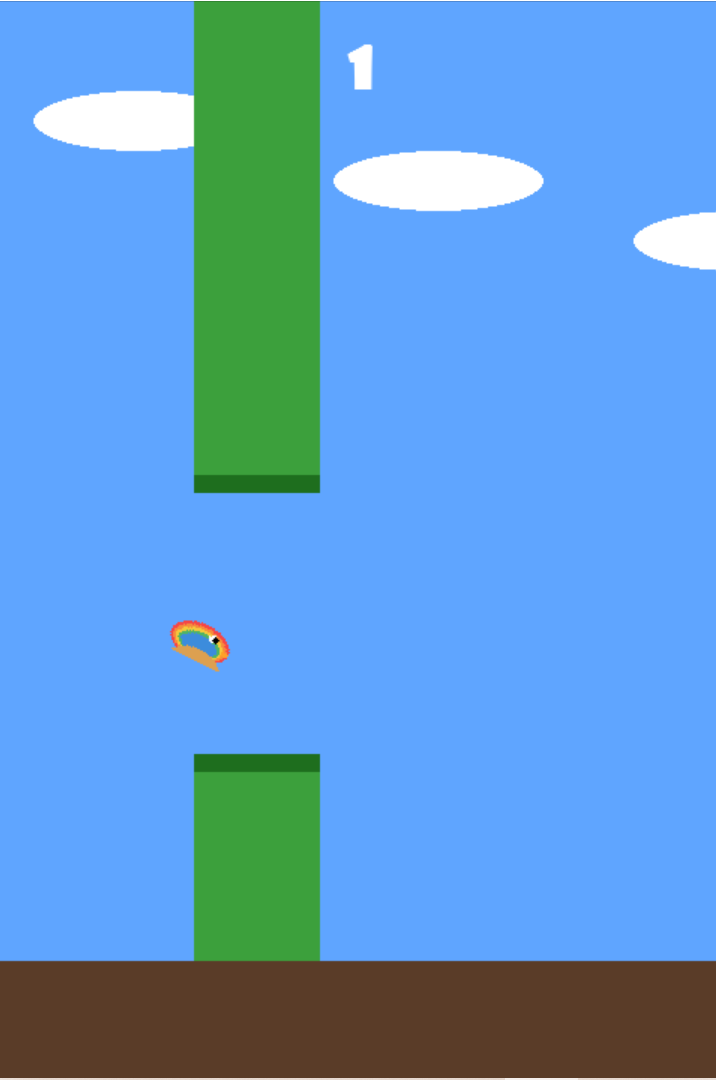
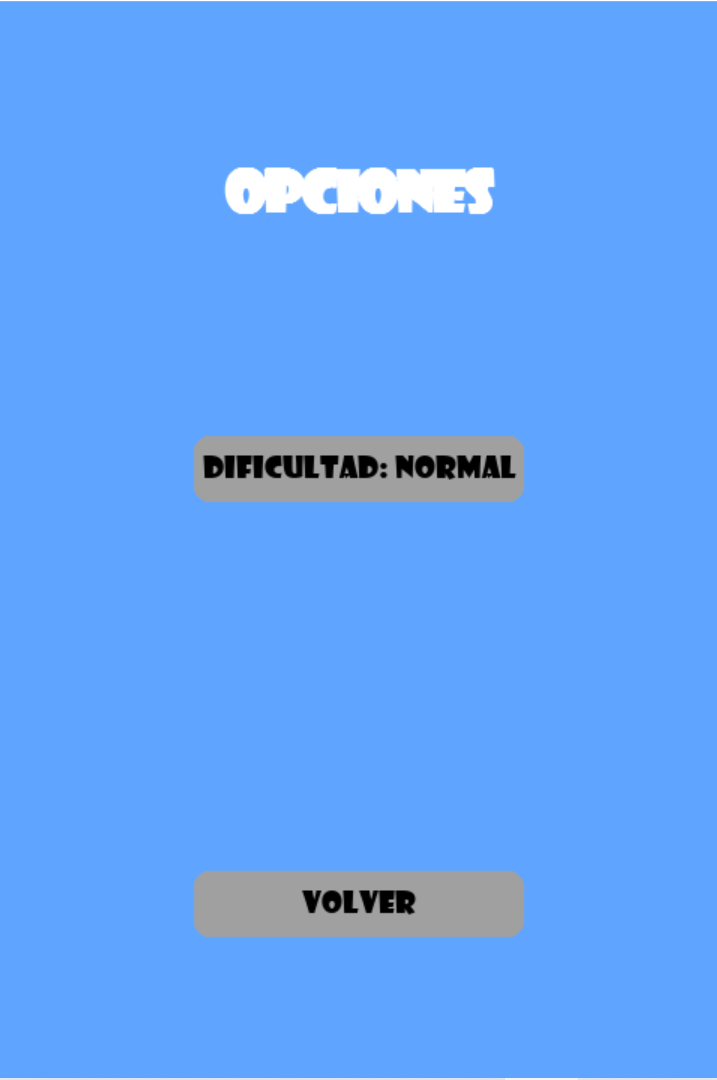
* Inicio → carga recursos y datos.
* Menu principal → elige acción:
* Jugar → inicia play\_game.
* Skins → cambia apariencia del pájaro.
* Récords → muestra puntajes guardados.
* Opciones → cambia dificultad.

Durante la partida:

* Pájaro se mueve con gravedad y aleteos.
* Tubos se generan y se desplazan.
* Se detectan colisiones y puntaje.
* Fin de partida → actualiza récords y vuelve al menú.

**ASPECTO DEL JUEGO**





**CÓDIGO**

import pygame, sys, os, json, random, math

from collections import deque

pygame.init()

ANCHO, ALTO = 480, 720

VENTANA = pygame.display.set\_mode((ANCHO, ALTO))

pygame.display.set\_caption("Pajaro Flojo - Menú Integrado")

FPS = 60

# Fuentes

FUENTE = pygame.font.SysFont("Showcard Gothic", 20)

FUENTE\_GRANDE = pygame.font.SysFont("Showcard Gothic", 38, bold=True)

# Colores

BLANCO = (255,255,255)

NEGRO = (0,0,0)

CELESTE = (95,165,255)

VERDE = (60,180,75)

ROJO = (214,70,70)

AMARILLO = (245,200,60)

AZUL = (80,140,210)

GRIS = (160,160,160)

DATA\_FILE = "data.json"

if not os.path.exists(DATA\_FILE):

    with open(DATA\_FILE, "w", encoding="utf-8") as f:

        json.dump({"highscore":0, "last\_scores":[], "skin":"amarillo", "difficulty":"Normal"}, f)

def load\_data():

    with open(DATA\_FILE, "r", encoding="utf-8") as f:

        return json.load(f)

def save\_data(d):

    with open(DATA\_FILE, "w", encoding="utf-8") as f:

        json.dump(d, f, indent=2)

data = load\_data()

# --------------------

# UI Boton

# --------------------

class Button:

    def \_\_init\_\_(self, text, x, y, w, h, col, col\_hover, action=None):

        self.text = text

        self.rect = pygame.Rect(x,y,w,h)

        self.col = col

        self.col\_hover = col\_hover

        self.action = action

    def draw(self, surf):

        mouse = pygame.mouse.get\_pos()

        hover = self.rect.collidepoint(mouse)

        color = self.col\_hover if hover else self.col

        pygame.draw.rect(surf, color, self.rect, border\_radius=10)

        txt = FUENTE.render(self.text, True, NEGRO)

        surf.blit(txt, txt.get\_rect(center=self.rect.center))

    def click(self, pos):

        if self.rect.collidepoint(pos) and self.action:

            self.action()

# --------------------

# Pajaro

# --------------------

class Bird:

    def \_\_init\_\_(self, x, y, skin="amarillo"):

        self.x = x

        self.y = y

        self.vy = 0.0

        self.radius = 18

        self.time = 0.0

        self.skin = skin

        self.alive = True

    def flap(self):

        self.vy = -9.5

    def update(self, dt):

        self.time += dt

        g = 0.6

        self.vy += g

        self.y += self.vy

    def draw(self, surf):

        colors = {

            "amarillo": (245,200,60),

            "rojo": (214,70,70),

            "azul": (80,140,210),

            "verde": (60,180,75),

            "rainbow": None

        }

        body\_col = colors.get(self.skin, (245,200,60))

        body\_rect = pygame.Rect(0,0, self.radius\*2+6, int(self.radius\*1.4))

        surf\_body = pygame.Surface(body\_rect.size, pygame.SRCALPHA)

        if self.skin == "rainbow":

            bands = [(245,70,70),(245,140,60),(245,200,60),(80,180,100),(60,140,220)]

            for i,c in enumerate(bands):

                r = int(body\_rect.height\*(1 - i\*0.12))

                pygame.draw.ellipse(surf\_body, c, pygame.Rect(0+i\*2, i\*2, body\_rect.width-i\*4, r))

        else:

            pygame.draw.ellipse(surf\_body, body\_col, pygame.Rect(0,0,body\_rect.width,body\_rect.height))

        pygame.draw.circle(surf\_body, BLANCO, (int(body\_rect.width\*0.68), int(body\_rect.height\*0.3)), max(2, self.radius//6))

        pygame.draw.circle(surf\_body, NEGRO, (int(body\_rect.width\*0.73), int(body\_rect.height\*0.3)), max(1, self.radius//9))

        pygame.draw.polygon(surf\_body, (255,160,40), [(int(body\_rect.width\*0.9), int(body\_rect.height\*0.45)), (int(body\_rect.width\*1.12), int(body\_rect.height\*0.5)), (int(body\_rect.width\*0.9), int(body\_rect.height\*0.6))])

        wing\_offset = math.sin(self.time\*12) \* 6

        pygame.draw.ellipse(surf\_body, (220,160,80), pygame.Rect(int(body\_rect.width\*0.12), int(body\_rect.height\*0.5+wing\_offset), int(body\_rect.width\*0.9), int(body\_rect.height\*0.9)))

        rotated = pygame.transform.rotate(surf\_body, int(self.vy\*3))

        surf.blit(rotated, rotated.get\_rect(center=(int(self.x), int(self.y))))

    @property

    def rect(self):

        return pygame.Rect(int(self.x-self.radius), int(self.y-self.radius), self.radius\*2, self.radius\*2)

# --------------------

# Tubo

# --------------------

class Tube:

    WIDTH = 84

    def \_\_init\_\_(self, x, gap\_y, gap\_h):

        self.x = x

        self.gap\_y = gap\_y

        self.gap\_h = gap\_h

        self.passed = False

    def update(self, speed):

        self.x -= speed

    def draw(self, surf):

        top\_h = int(self.gap\_y - self.gap\_h//2)

        bot\_y = int(self.gap\_y + self.gap\_h//2)

        pygame.draw.rect(surf, (60,160,60), (int(self.x), 0, self.WIDTH, top\_h))

        pygame.draw.rect(surf, (60,160,60), (int(self.x), bot\_y, self.WIDTH, ALTO - bot\_y))

        pygame.draw.rect(surf, (30,110,30), (int(self.x), max(0,top\_h-12), self.WIDTH, 12))

        pygame.draw.rect(surf, (30,110,30), (int(self.x), bot\_y, self.WIDTH, 12))

    def collides(self, bird: Bird):

        br = bird.rect

        top\_rect = pygame.Rect(int(self.x), 0, self.WIDTH, int(self.gap\_y - self.gap\_h/2))

        bot\_rect = pygame.Rect(int(self.x), int(self.gap\_y + self.gap\_h/2), self.WIDTH, ALTO - int(self.gap\_y + self.gap\_h/2))

        return br.colliderect(top\_rect) or br.colliderect(bot\_rect)

# --------------------

# Logica del juego

# --------------------

def play\_game():

    global data

    clock = pygame.time.Clock()

    diff = data.get("difficulty","Normal")

    if diff == "Easy":

        gap = 200; speed = 3.6; spawn\_interval = 1.8

    elif diff == "Hard":

        gap = 150; speed = 5.0; spawn\_interval = 1.1

    else:

        gap = 175; speed = 4.2; spawn\_interval = 1.4

    bird = Bird(ANCHO\*0.28, ALTO\*0.5, skin=data.get("skin","amarillo"))

    tubes = []

    spawn\_timer = 0.0

    score = 0

    running = True

    game\_over = False

    ground\_y = ALTO - 80

    while running:

        dt = clock.tick(FPS) / 1000.0

        spawn\_timer += dt

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit(); sys.exit()

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_SPACE:

                    if not game\_over: bird.flap()

                    else: return score

                if event.key == pygame.K\_ESCAPE:

                    return score

            if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

                if not game\_over: bird.flap()

        if not game\_over:

            bird.update(dt)

            if spawn\_timer >= spawn\_interval:

                spawn\_timer = 0.0

                margin = 120

                gap\_y = random.randint(margin + gap//2, ALTO - margin - gap//2 - 80)

                tubes.append(Tube(ANCHO + 20, gap\_y, gap))

            for t in tubes:

                t.update(speed)

                if (not t.passed) and (bird.x > t.x + Tube.WIDTH):

                    t.passed = True

                    score += 1

            tubes = [t for t in tubes if t.x + Tube.WIDTH > -50]

            for t in tubes:

                if t.collides(bird):

                    game\_over = True

                    bird.alive = False

            if bird.y - bird.radius < 0:

                bird.y = bird.radius

                bird.vy = 0

            if bird.y + bird.radius > ground\_y:

                game\_over = True

                bird.alive = False

        # DRAW

        VENTANA.fill(CELESTE)

        for i in range(3):

            cx = (i\*200 + pygame.time.get\_ticks()\*0.02) % (ANCHO+120) - 60

            pygame.draw.ellipse(VENTANA, (255,255,255,50), (cx, 60 + i\*40, 140, 40))

        for t in tubes:

            t.draw(VENTANA)

        pygame.draw.rect(VENTANA, (90,60,40), (0, ground\_y, ANCHO, ALTO-ground\_y))

        bird.draw(VENTANA)

        srf = FUENTE\_GRANDE.render(str(score), True, BLANCO)

        VENTANA.blit(srf, (ANCHO//2 - srf.get\_width()//2, 28))

        if game\_over:

            over = FUENTE\_GRANDE.render("¡PERDISTE!", True, ROJO)

            VENTANA.blit(over, (ANCHO//2 - over.get\_width()//2, ALTO//2 - 20))

            tip = FUENTE.render("Pulsa ESPACIO para reiniciar o ESC para salir", True, BLANCO)

            VENTANA.blit(tip, (ANCHO//2 - tip.get\_width()//2, ALTO//2 + 36))

        pygame.display.flip()

# --------------------

# Menus y estados

# --------------------

state = "menu"

clock = pygame.time.Clock()

def set\_state(s): global state; state = s

def main\_menu\_buttons():

    return [

        Button("Jugar", ANCHO//2-110, 220, 220, 48, VERDE, AMARILLO, lambda: set\_state("play")),

        Button("Skins", ANCHO//2-110, 290, 220, 48, AZUL, AMARILLO, lambda: set\_state("skins")),

        Button("Récords", ANCHO//2-110, 360, 220, 48, GRIS, AMARILLO, lambda: set\_state("records")),

        Button("Opciones", ANCHO//2-110, 430, 220, 48, GRIS, AMARILLO, lambda: set\_state("options")),

        Button("Salir", ANCHO//2-110, 500, 220, 48, ROJO, AMARILLO, lambda: pygame.event.post(pygame.event.Event(pygame.QUIT)))

    ]

def skins\_buttons():

    x = ANCHO//2 - 110

    return [

        Button("Amarillo", x, 220, 220, 44, AMARILLO, BLANCO, lambda: choose\_skin("amarillo")),

        Button("Rojo", x, 280, 220, 44, ROJO, BLANCO, lambda: choose\_skin("rojo")),

        Button("Azul", x, 340, 220, 44, AZUL, BLANCO, lambda: choose\_skin("azul")),

        Button("Verde", x, 400, 220, 44, VERDE, BLANCO, lambda: choose\_skin("verde")),

        Button("Rainbow", x, 460, 220, 44, (255,130,130), BLANCO, lambda: choose\_skin("rainbow")),

        Button("Volver", x, 520, 220, 44, GRIS, AMARILLO, lambda: set\_state("menu"))

    ]

def records\_buttons():

    return [Button("Volver", ANCHO//2-110, ALTO-140, 220, 44, GRIS, AMARILLO, lambda: set\_state("menu"))]

def options\_buttons():

    return [

        Button(f"Dificultad: {data.get('difficulty','Normal')}", ANCHO//2-110, 290, 220, 44, GRIS, AMARILLO, toggle\_difficulty),

        Button("Volver", ANCHO//2-110, ALTO-140, 220, 44, GRIS, AMARILLO, lambda:set\_state("menu"))

    ]

def choose\_skin(name):

    global data

    data["skin"] = name

    save\_data(data)

    set\_state("menu")

def toggle\_difficulty():

    cur = data.get("difficulty","Normal")

    order = ["Easy","Normal","Hard"]

    i = (order.index(cur)+1) % len(order)

    data["difficulty"] = order[i]

    save\_data(data)

    set\_state("options")

# --------------------

# Loop principal

# --------------------

def main():

    global state, data

    botones = main\_menu\_buttons()

    while True:

        clock.tick(FPS)

        VENTANA.fill(CELESTE)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit(); sys.exit()

            if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

                pos = event.pos

                for b in botones:

                    b.click(pos)

            if event.type == pygame.KEYDOWN and event.key == pygame.K\_ESCAPE:

                if state != "menu":

                    set\_state("menu")

        if state == "menu":

            t = (pygame.time.get\_ticks() \* 0.002) % (2\*math.pi)

            title\_y = 110 + math.sin(t)\*8

            txt = FUENTE\_GRANDE.render("PÁJARO FLOJO", True, BLANCO)

            VENTANA.blit(txt, (ANCHO//2 - txt.get\_width()//2, int(title\_y)))

            subtitle = FUENTE.render("Espacio/Click para aletear  •  ESC: menú", True, BLANCO)

            VENTANA.blit(subtitle, (ANCHO//2 - subtitle.get\_width()//2, 170))

            botones = main\_menu\_buttons()

            for b in botones:

                b.draw(VENTANA)

        elif state == "play":

            sc = play\_game()

            data = load\_data()

            if sc > data.get("highscore",0):

                data["highscore"] = sc

            ls = deque(data.get("last\_scores",[]), maxlen=10)

            ls.appendleft(sc)

            data["last\_scores"] = list(ls)[:10]

            save\_data(data)

            state = "menu"

        elif state == "skins":

            txt = FUENTE\_GRANDE.render("Skins", True, BLANCO)

            VENTANA.blit(txt, (ANCHO//2 - txt.get\_width()//2, 110))

            botones = skins\_buttons()

            for b in botones:

                b.draw(VENTANA)

        elif state == "records":

            data = load\_data()

            txt = FUENTE\_GRANDE.render("Récords", True, BLANCO)

            VENTANA.blit(txt, (ANCHO//2 - txt.get\_width()//2, 40))

            hs = FUENTE.render(f"Highscore: {data.get('highscore',0)}", True, BLANCO)

            VENTANA.blit(hs, (ANCHO//2 - hs.get\_width()//2, 120))

            last = data.get("last\_scores",[])

            for i, s in enumerate(last[:5]):

                sc\_txt = FUENTE.render(f"{i+1}. {s}", True, BLANCO)

                VENTANA.blit(sc\_txt, (ANCHO//2 - sc\_txt.get\_width()//2, 160 + i\*30))

            botones = records\_buttons()

            for b in botones:

                b.draw(VENTANA)

        elif state == "options":

            txt = FUENTE\_GRANDE.render("Opciones", True, BLANCO)

            VENTANA.blit(txt, (ANCHO//2 - txt.get\_width()//2, 110))

            botones = options\_buttons()

            for b in botones:

                b.draw(VENTANA)

        pygame.display.flip()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**CONCLUSIÓN**

El desarrollo del proyecto “Pájaro Flojo” permitió integrar conocimientos teóricos y prácticos de programación, aplicando principios de la programación orientada a objetos, el manejo de eventos gráficos y la implementación de interfaces interactivas mediante la librería Pygame.

Durante el proceso se diseñó un sistema completo que incluye menús, selección de skins, gestión de récords, opciones de dificultad y una jugabilidad fluida basada en la detección de colisiones y la física del movimiento. Esto demostró la importancia de una estructura modular del código, donde cada clase y función cumple un rol definido y contribuye al funcionamiento general del juego.

Asimismo, el proyecto fortaleció competencias clave en diseño lógico, resolución de problemas, depuración y optimización de código, además de fomentar la creatividad al desarrollar un entorno visual y dinámico.

En conclusión, “Pájaro Flojo” representa un trabajo sólido tanto desde el punto de vista técnico como formativo, evidenciando la capacidad de combinar programación, diseño y lógica computacional para crear una aplicación funcional, entretenida y con potencial de expansión futura.