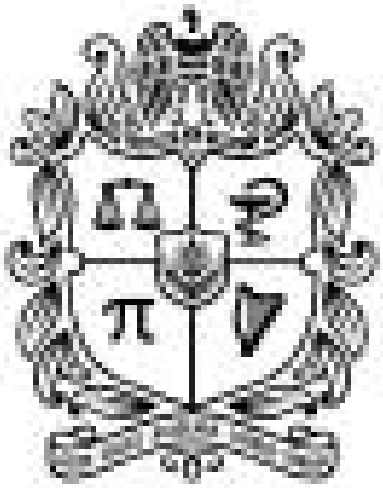


UNIVERSIDAD
NACIONAL
DE COLOMBIA

PYNG-P0NG

Joan Alejandro Muñoz Bejarano

Ingeniería Electrónica

Programación de computadores

Universidad Nacional de Colombia

Sede Bogotá

2024-2S

Tabla de contenido

Resumen

PYNG-P0NG es un videojuego inspirado en el clásico Pong, desarrollado en Python y combinando el uso de Tkinter, Pygame y Firebase para ofrecer una experiencia de juego completa y conectada. El juego incorpora un sistema de registro e inicio de sesión, permitiendo a los usuarios crear cuentas y almacenar sus resultados en la nube a través de Firebase Realtime Database. Una vez autenticados, los jugadores pueden elegir entre diferentes modos de juego: enfrentarse contra la CPU o contra otro jugador (1vs1). Además, se ofrece la posibilidad de personalizar el aspecto visual del juego mediante la selección de temas de colores, así como ajustar la dificultad de la partida, lo cual afecta la velocidad de la bola y de los movimientos tanto del jugador como de la CPU.

La interfaz de usuario está diseñada con Tkinter, proporcionando ventanas intuitivas para el inicio de sesión, registro de usuarios, selección de configuraciones y el menú principal, en el que se muestran instrucciones claras y opciones de navegación. Durante la partida, Pygame gestiona la dinámica del juego, incluyendo la detección de colisiones, el movimiento de la bola y las paletas, y el registro de puntos. Al finalizar una partida —ya sea al alcanzar un límite de tiempo o de puntos—, el marcador se guarda automáticamente en la base de datos y se ofrece la opción de consultar los últimos resultados.

Este sistema integrado combina diversas tecnologías para ofrecer una experiencia de juego interactiva, permitiendo a los usuarios disfrutar de un juego clásico con mejoras modernas en conectividad y personalización, logrando que la experiencia sea tanto nostálgica como innovadora.

Requerimientos funcionales

- Registro e Inicio de Sesión de Usuarios:

  El sistema debe permitir que los usuarios se registren ingresando un nombre de usuario y contraseña. Durante el registro, se valida que los campos no estén vacíos y que el nombre de usuario no se encuentre ya en uso. En el inicio de sesión, se comprueba que las credenciales ingresadas coincidan con las almacenadas en Firebase, proporcionando mensajes de error claros en caso de fallos y confirmando el acceso correcto mediante notificaciones.

- Menú Principal y Navegación:

  Una vez autenticado, el usuario accede a un menú principal donde puede elegir entre distintos modos de juego (1vs1 o contra la CPU), consultar el marcador o salir. Este menú actúa como centro de navegación, facilitando el acceso a las distintas funcionalidades del sistema mediante botones e instrucciones visuales, y proporcionando una experiencia de usuario intuitiva.

- Personalización del Juego:

  Antes de iniciar una partida, se ofrece una ventana de configuración donde el usuario selecciona el tema de colores (que afecta el fondo, la bola y la paleta) y la dificultad del juego. La selección de dificultad modifica parámetros clave como la velocidad de la bola, la velocidad del jugador y la velocidad de la CPU, permitiendo ajustar la experiencia de juego según las preferencias del usuario.

- Gestión y Almacenamiento de Resultados:

  Al finalizar una partida —ya sea por alcanzar un límite de tiempo o un número determinado de puntos—, el marcador se guarda automáticamente en Firebase. Además, se implementa

una función que permite al usuario consultar los últimos cinco marcadores registrados, ofreciendo un historial rápido de sus partidas previas.

Requerimientos no funcionales

- Interfaz Gráfica Intuitiva y Atractiva:

  La aplicación utiliza Tkinter para construir una interfaz de usuario amigable, con elementos visuales consistentes como etiquetas, botones y campos de entrada. El uso de colores contrastantes y fuentes legibles, junto con una disposición lógica de los componentes, asegura que los usuarios puedan navegar y utilizar el sistema de forma sencilla y agradable.

- Experiencia de Juego Fluida y Dinámica:

  Pygame se encarga de la simulación del juego, asegurando que los movimientos de la bola y de las paletas sean suaves y consistentes. Se implementa un bucle de juego que opera a 60 fotogramas por segundo, garantizando una respuesta inmediata a las acciones del jugador y una experiencia de juego sin interrupciones.

- Integración Confiable con Firebase:

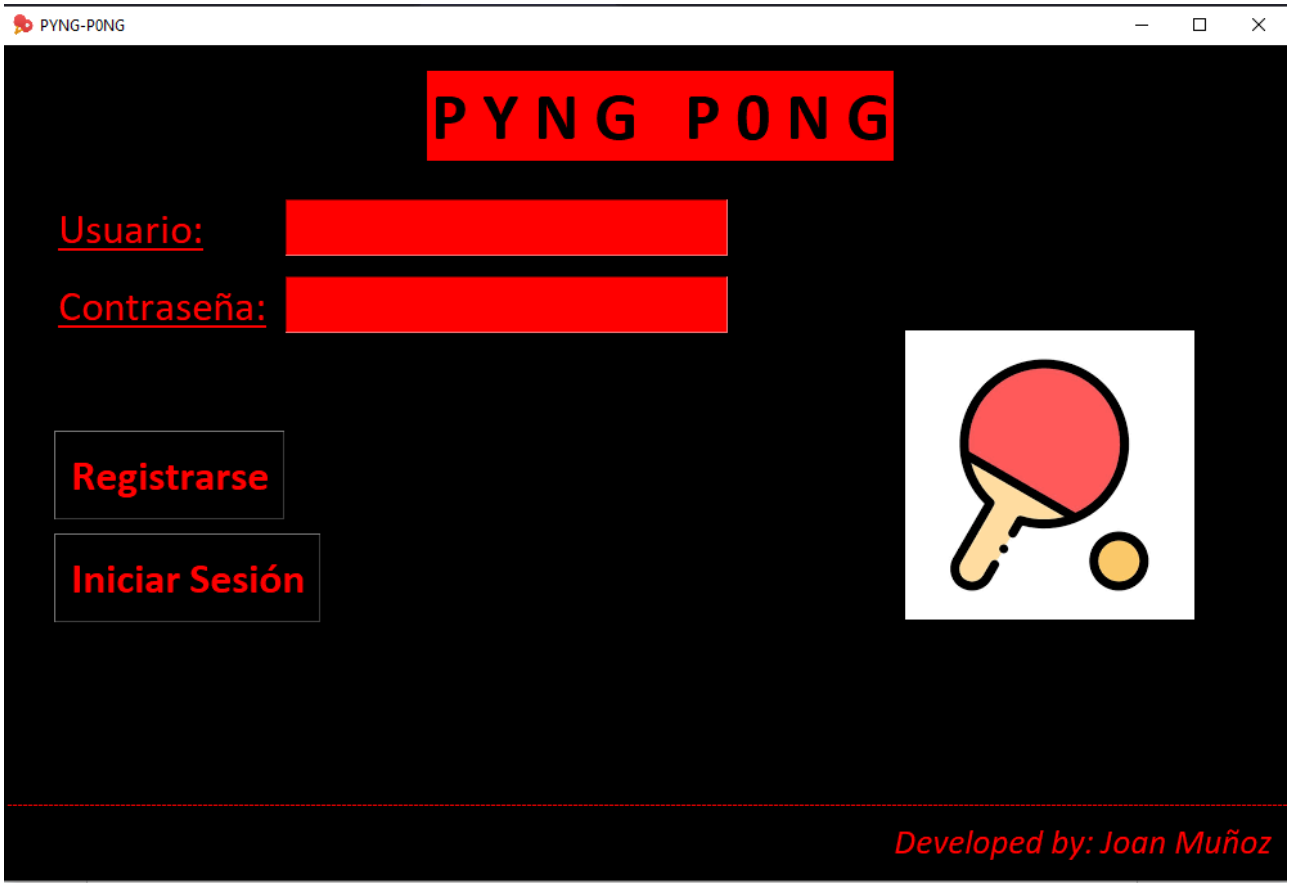
  La integración con Firebase Realtime Database permite la persistencia de datos en la nube, ofreciendo un mecanismo seguro y escalable para almacenar información de usuarios y resultados de partidas. Se utiliza un certificado de autenticación para establecer la conexión, lo que refuerza la seguridad en la transmisión y almacenamiento de datos.

- Rendimiento y Estabilidad del Sistema:

  El código está estructurado en funciones modulares que separan claramente la lógica del juego, la gestión de la interfaz y las operaciones de base de datos. Esta modularidad, junto con un manejo adecuado de eventos y ciclos de juego, contribuye a un rendimiento óptimo y a la estabilidad general del sistema, minimizando posibles errores o caídas durante la ejecución.

Ejecución

1. El programa inicia en el menú de logueo e inicio de sesión

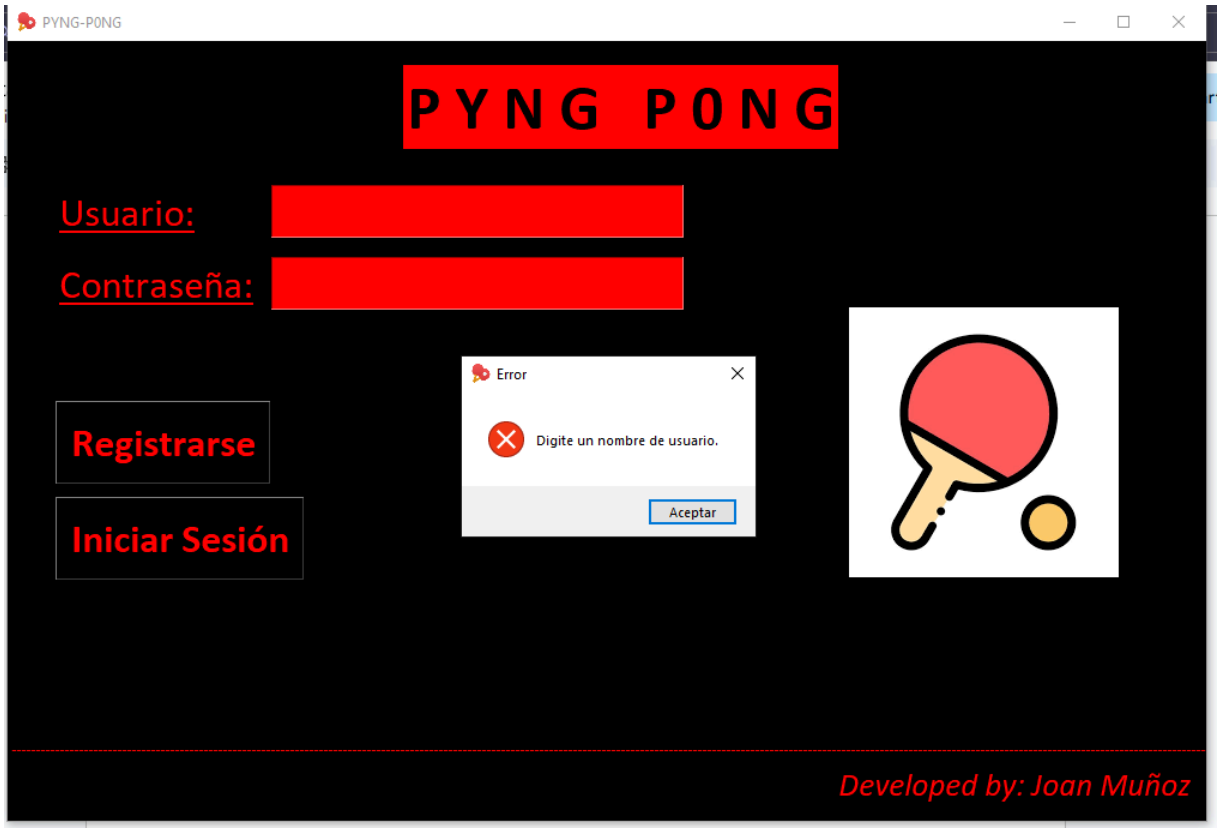


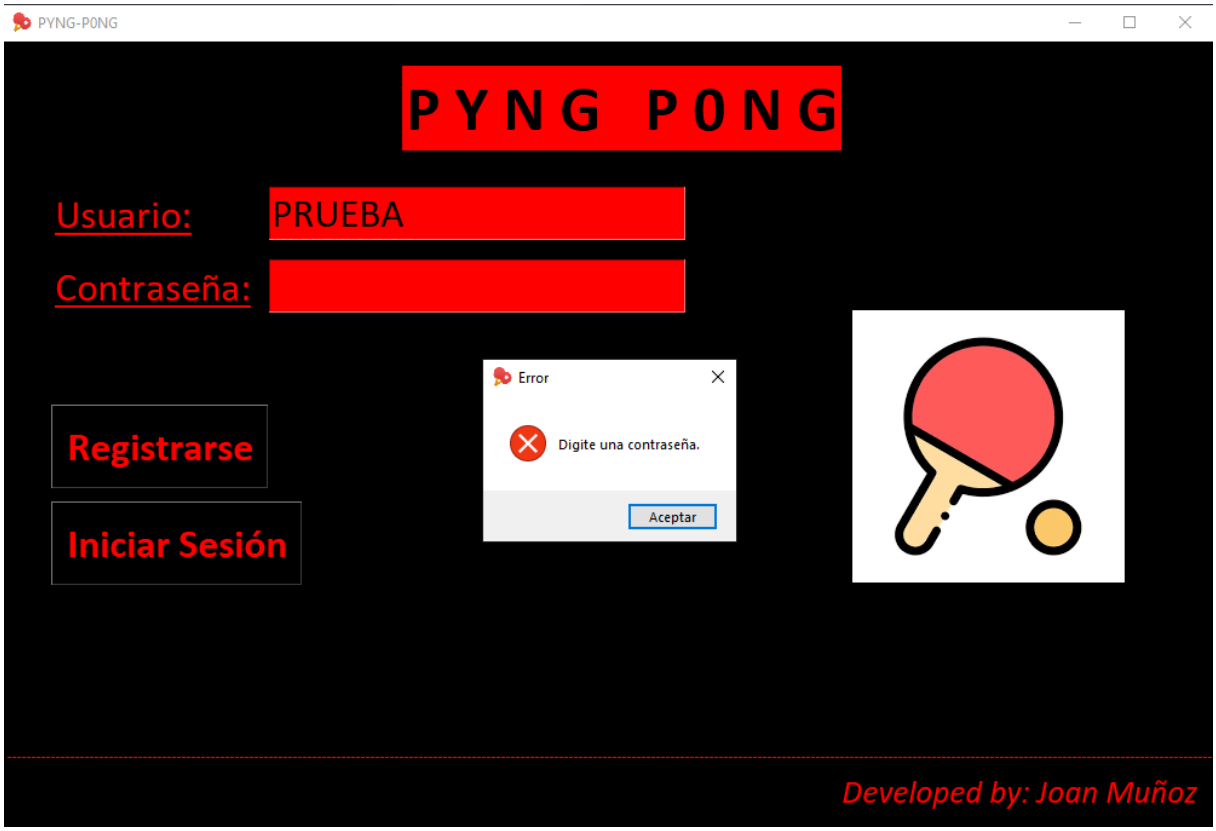
Este se compone de:

- ➔ Titulo (label)
- ➔ Etiqueta de Usuario (Label)
- ➔ Etiqueta de Contraseña (Label)
- ➔ Pie de página (Label
- ➔ Cuadros de texto - Usuario/Contraseña (Entry)
- ➔ Imagen de referencia (Png)
- ➔ Botones de logueo y registro (Button)
- ➔ Ventanas emergentes (Messagebox)

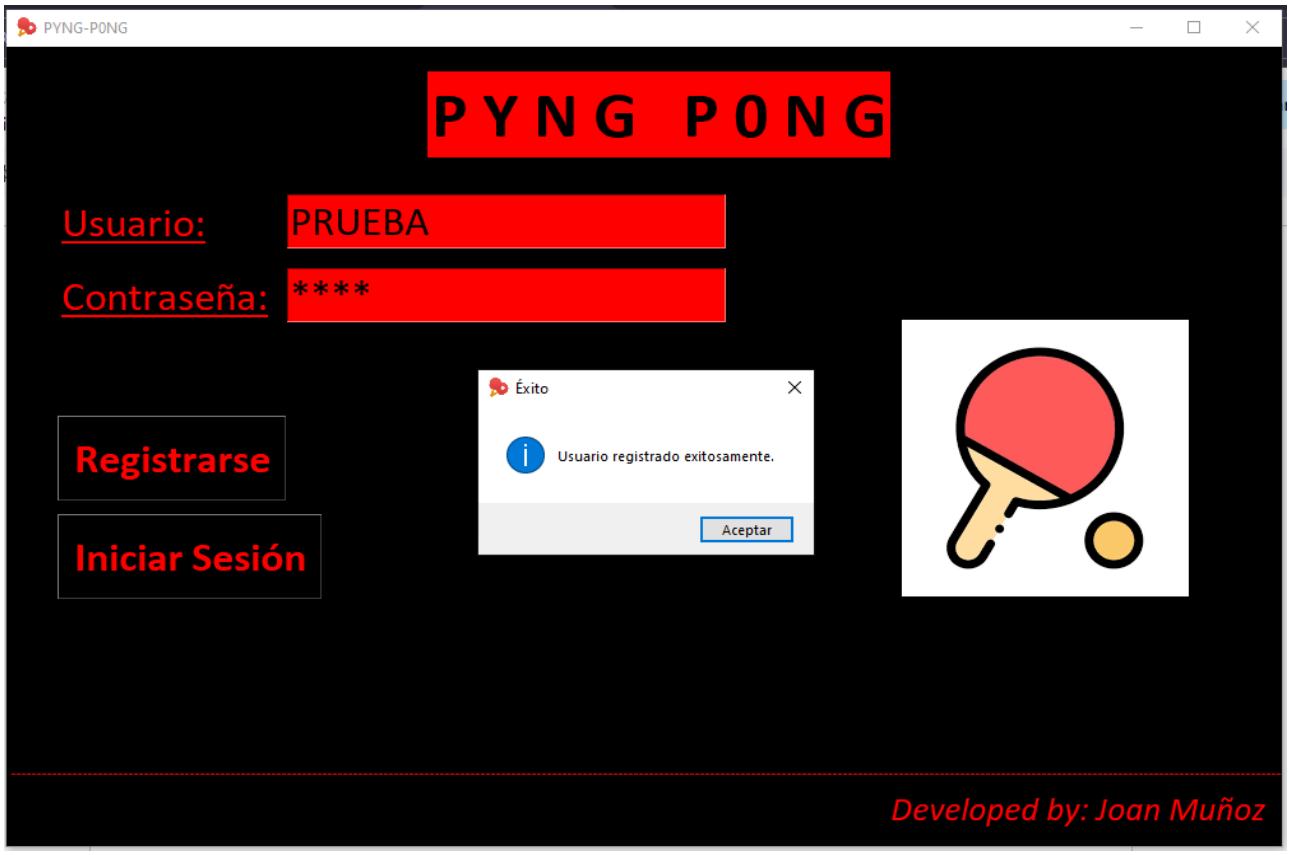
Sus Interacciones son:

➔ Loguearse o registrarse sin llenar los cuadros




PYNG-P0NG
PYNG P0NG
Usuario:
PRUEBA
Contraseña:
Registrarse
Iniciar Sesión
Error
Digite una contraseña.
Aceptar
Developed by: Joan Muñoz

➔ Registrar un usuario nuevo

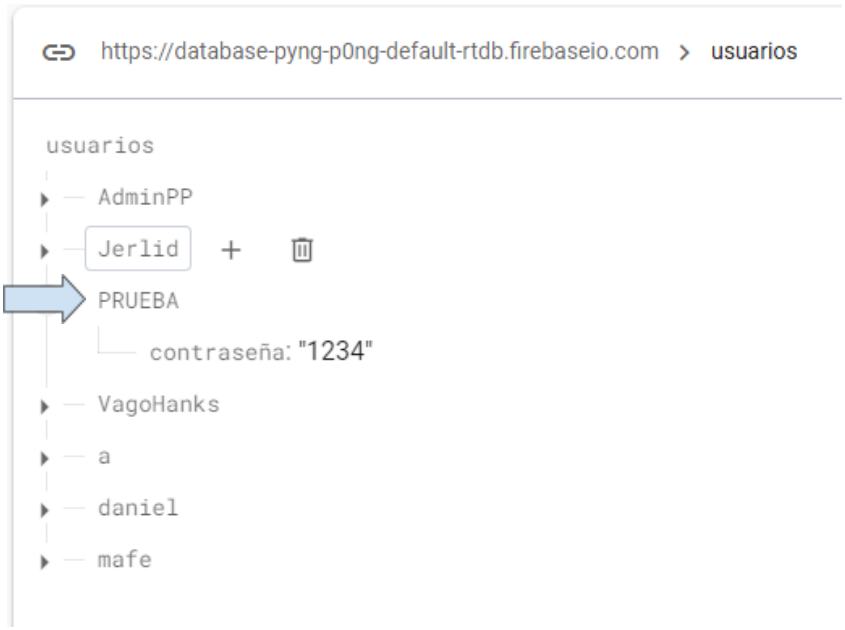


➔ Registrar un usuario ya existente

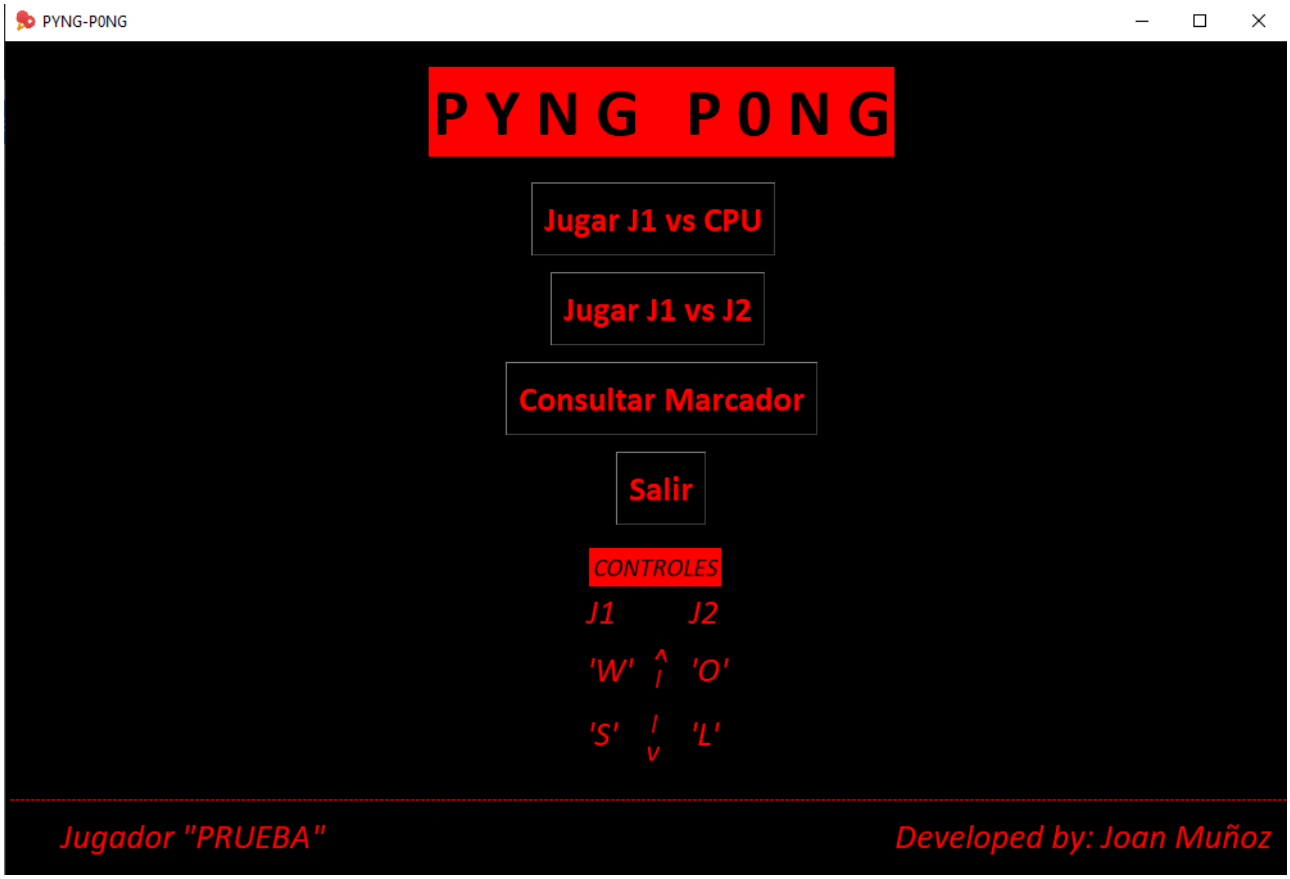
➔ Loguearse con los datos incorrectos

➔ Loguearse correctamente

Evidencia de la creación del usuario en la base de datos de Firebase:



2. Una vez logueado correctamente pasamos al menú inicial del juego



Este se compone de:

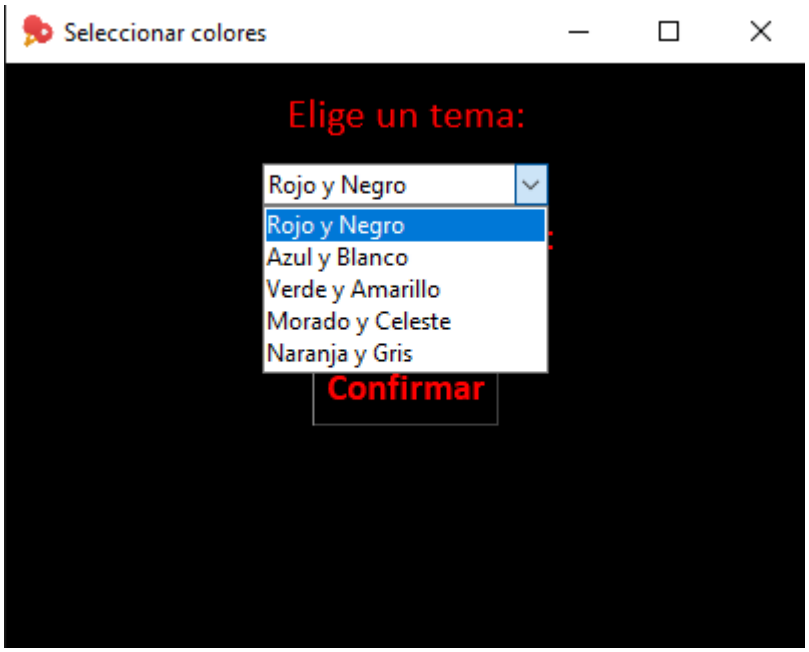
- Titulo (Label)
- Pie de página (Label)
- Jugador Activo (Label)
- Guia de controles (Labels)
- Botón jugar J1 vs Cpu (Button)
- Botón jugar J1 vs J2 (Button)
- Botón consultar marcador (Button)
- Botón salir (Button)

Interacciones:

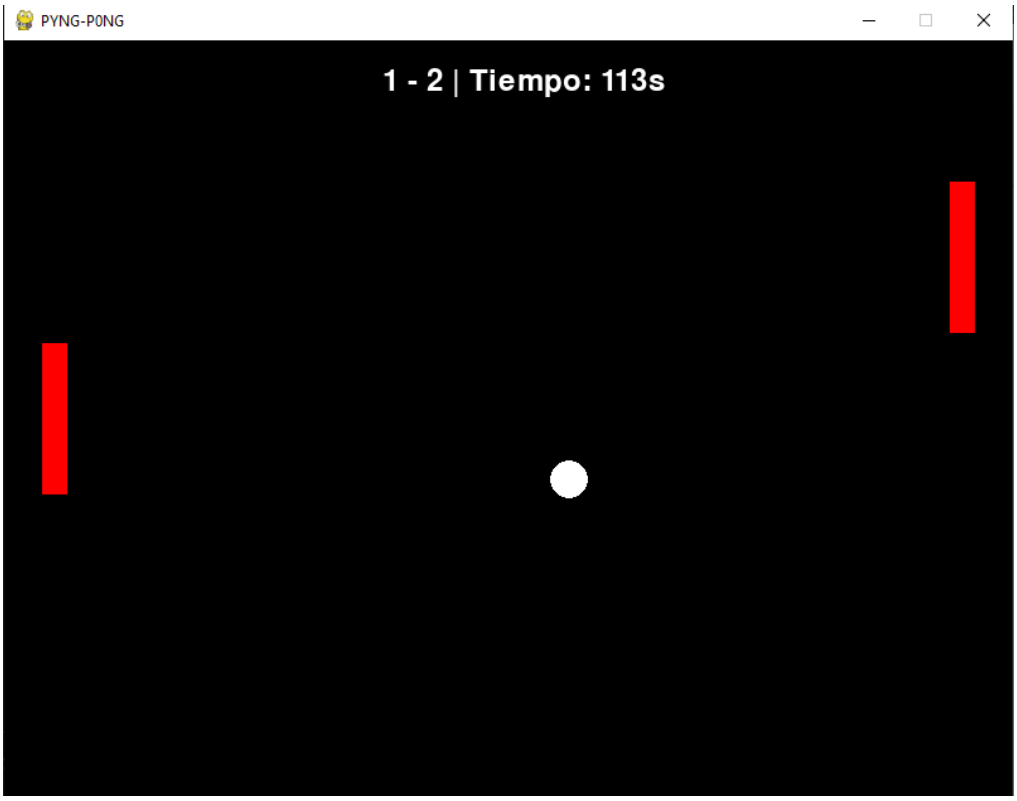
- Jugar (aplica para ambos modos J1vsCPU J1vsJ2)

Se ejecuta una nueva ventana para seleccionar tema de colores y dificultad de la partida.

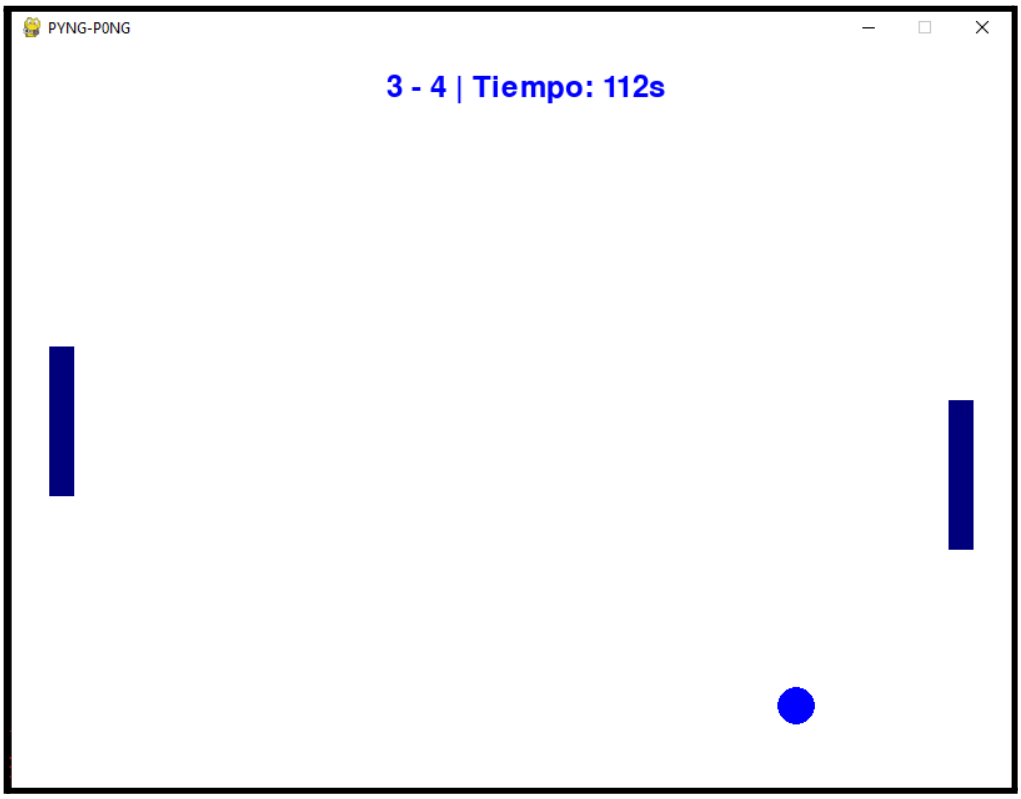
Opciones de color:



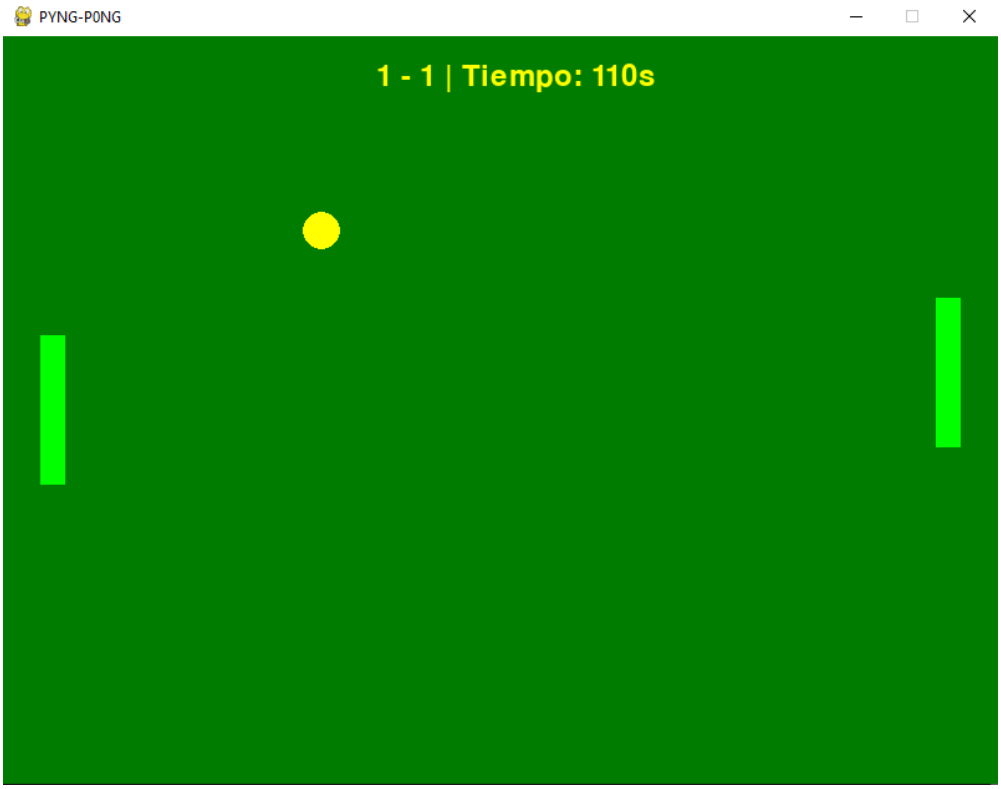
- Rojo y Negro:



- Azul y Blanco:

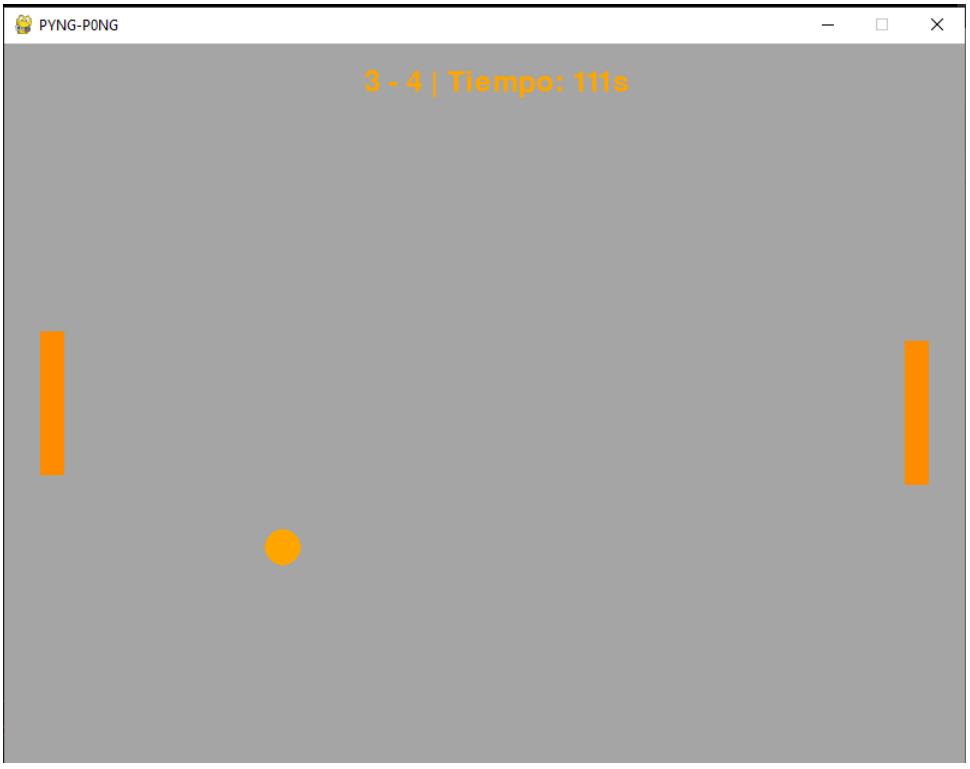


- Verde y Amarillo:

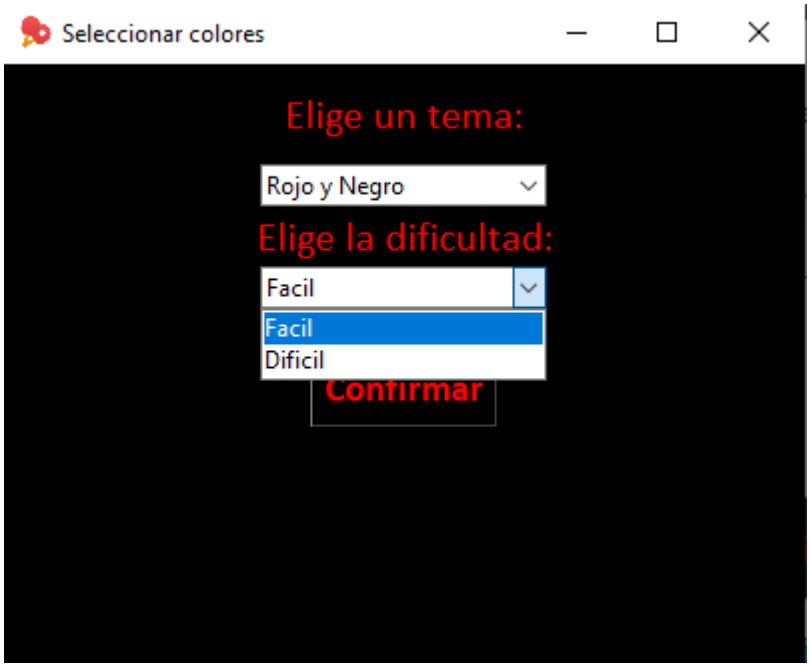


- Morado y Celeste:

- Naranja y Gris:



Opciones de dificultad:



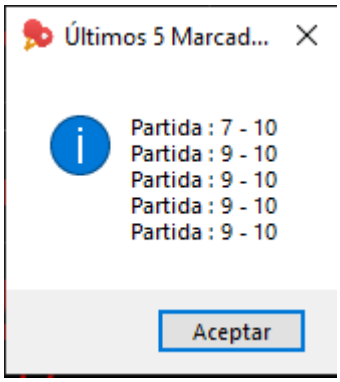
Fácil:

- Velocidad de la bola = 5
- Velocidad de paleta = 4

Difícil:

- Velocidad de la bola = 12
- Velocidad de la paleta = 8

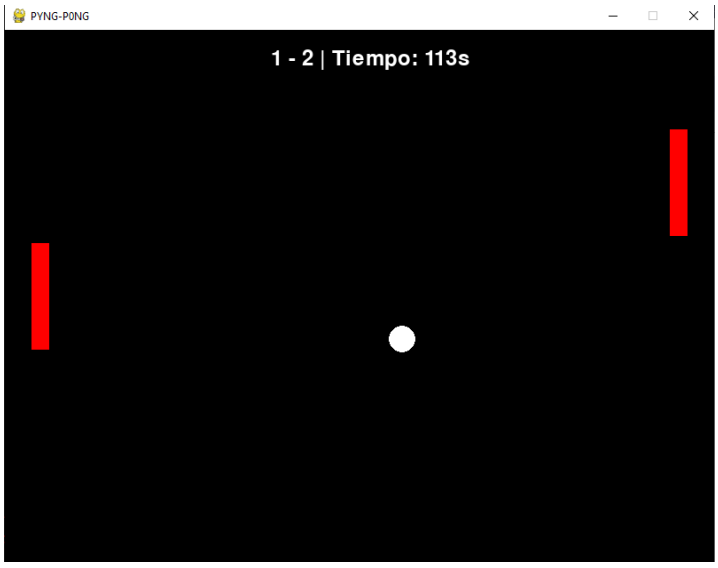
Consultar marcador



Muestra el resultado de las últimas 5 partidas de ese jugador, evidencia en la base de datos:

- Botón de salir, destruye la ventana y abre la del logueo

3. Interfaz de juego



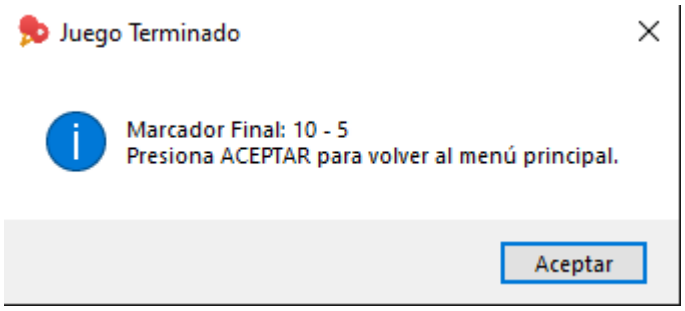
Este se compone de:

- ★ Paletas
- ★ Pelota
- ★ Marcador
- ★ Tiempo

Los controles de las paletas son:

- ★ J1: ‘W’ y ‘S’ arriba y abajo respectivamente
- ★ J2: ‘O’ y ‘L’ arriba y abajo respectivamente
- ★ CPU: Se mueve siguiendo la pelota pero con una velocidad menor

Las partidas finalizan cuando el tiempo (2 minutos) llega a cero o cuando alguno de los jugadores obtiene 10 puntos. Una vez finalizada la partida sale una ventana emergente con el marcador y final y la opción de volver al menú.



Código

Las librerías utilizadas en el código son:

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
from tkinter import PhotoImage
import firebase_admin
from firebase_admin import credentials, db
import pygame
import sys

```

Este segmento de código importa las bibliotecas necesarias para el funcionamiento del proyecto PYNG-P0NG.

- Tkinter se usa para crear la interfaz gráfica del usuario (GUI), incluyendo botones, cuadros de texto y mensajes emergentes. También se importa ttk para widgets con un diseño más moderno y PhotoImage para la gestión de imágenes en la interfaz.
- Firebase Admin permite la conexión con Firebase Realtime Database, utilizando credentials para autenticar la aplicación y db para interactuar con la base de datos (guardar y consultar datos de los usuarios y marcadores).
- Pygame se encarga de la lógica del juego, manejando gráficos, eventos del teclado y colisiones.
- Sys proporciona funciones del sistema, como la capacidad de cerrar el programa correctamente cuando sea necesario.

Todo el código está segmentado por funciones, las únicas sentencias fuera de las funciones son:

```
# Inicializar Firebase
cred = credentials.Certificate("database-pyng-p0ng-firebase-adminsdk-fbsvc-07abffe6fa.json")
firebase_admin.initialize_app(cred, {
    "databaseURL": "https://database-pyng-p0ng-default-rtdb.firebaseio.com/"
})

# Iniciar pygame
pygame.init()
# Configuración de la ventana de Pygame
ANCHO, ALTO = 800, 600
```

Este bloque de código establece la conexión con Firebase para gestionar los datos de los jugadores y sus marcadores. También prepara Pygame para la ejecución del juego y define el tamaño de la ventana de las partidas.

```
# Crear ventana principal
root = tk.Tk()
root.title("PYNG-P0NG")
root.geometry("1000x650")
root.configure(bg="black")
root.iconbitmap("icono.ico")

# interfaz del menu de logueo
label_linea = tk.Label(root, text="------------------------------------------------------------------------------------------
label_linea.place(x=0, y=580)
label_Titulo = tk.Label(root, text="P Y N G   P 0 N G", bg="red", font=("Calibri", 40, "bold"))
label_Titulo.place(x=330, y=20)
label_pie = tk.Label(root, text="Developed by: Joan Muñoz", fg="red", bg="black", font=("Calibri", 20, "italic"))
label_pie.place(x=690, y=600)
label_usuario = tk.Label(root, text="Usuario:", fg="red", bg="black", font=("Calibri", 25, "underline"))
label_usuario.place(x=40, y=120)
entry_usuario = tk.Entry(root, fg="black", bg="red", font=("Calibri", 25))
entry_usuario.place(x=220, y=120)
label_contraseña = tk.Label(root, text="Contraseña:", fg="red", bg="black", font=("Calibri", 25, "underline"))
label_contraseña.place(x=40, y=180)
entry_contraseña = tk.Entry(root, show="*", fg="black", bg="red", font=("Calibri", 25))
entry_contraseña.place(x=220, y=180)
img = PhotoImage(file="IMG1.png")
label_img = tk.Label(root, image=img, bg="black")
label_img.place(x=700, y=220)
btn_registrar = tk.Button(root, text="Registrarse", command=registrar_usuario, fg="red", bg="black", font=("Calibri", 25, "bold"))
btn_registrar.place(x=40, y=300)
btn_logueo = tk.Button(root, text="Iniciar Sesión", command=logueo, fg="red", bg="black", font=("Calibri", 25, "bold"))
btn_logueo.place(x=40, y=380)

root.mainloop()
```


El otro bloque de código fuera de las funciones es el anterior donde se crea la ventana principal del programa (root) y se define sus atributos como geometría, icono, fondo, tamaño.

Seguido de esto los elementos de la ventana también se definen y se utilizan diferentes métodos para configurar su visualización.

Ejemplo

- .place: para ubicarlos en la venta
- bg=:color del fondo
- fg=:color de letra
- font: tamaño, fuente y tipado (negrilla, cursiva, subrayado)

A los botones se les asigna el comando de ejecutar la función correspondiente una vez sean

accionados.

Seguidamente el código está modulado por diferentes funciones.

```
# funcion de logueo
def logueo():
    usuario = entry_usuario.get()
    contraseña = entry_contraseña.get()
    if usuario == "":
        messagebox.showerror("Error", "Digite el nombre de usuario.")
        return
    if contraseña == "":
        messagebox.showerror("Error", "Digite la contraseña.")
        return
    ref = db.reference(f"usuarios/{usuario}")
    dato_usuario = ref.get()
    if dato_usuario and dato_usuario.get("contraseña") == contraseña:
        messagebox.showinfo("Bienvenido", f"¡Bienvenido al juego, {usuario}!")
        abrir_menu_principal(usuario)
    else:
        messagebox.showerror("Error", "Usuario o contraseña incorrectos.")
```

logueo() se encarga de autenticar a los usuarios verificando sus credenciales en la base de datos Firebase.

1. Se recuperan el nombre de usuario y la contraseña ingresados en los campos de entrada de la interfaz gráfica (entry_usuario y entry_contraseña).
2. Si el usuario no ingresó un nombre o una contraseña, se muestra un mensaje de error mediante messagebox.showerror() y la función se detiene (return).
3. Se genera una referencia a la ruta "usuarios/{usuario}" dentro de Firebase.

   Se obtiene la información del usuario almacenada en la base de datos y se guarda en dato_usuario.

4. Se verifica que dato_usuario contenga información válida (es decir, que el usuario exista en la base de datos).

   Se compara la contraseña ingresada con la contraseña almacenada en Firebase.

   Si coinciden, se muestra un mensaje de bienvenida y se llama a

abrir_menu_principal(usuario), que redirige al usuario al menú del juego.

Si las credenciales son incorrectas, se muestra un mensaje de error.

```python
#funcion registrar usuario
def registrar_usuario():
    usuario = entry_usuario.get()
    contraseña = entry_contraseña.get()
    if usuario == "":
        messagebox.showerror("Error", "Digite un nombre de usuario.")
        return
    if contraseña == "":
        messagebox.showerror("Error", "Digite una contraseña.")
        return
    ref = db.reference(f"usuarios/{usuario}")
    if ref.get():
        messagebox.showerror("Error", "El nombre de usuario ya está en uso.")
    else:
        ref.set({"contraseña": contraseña})
        messagebox.showinfo("Éxito", "Usuario registrado exitosamente.")
```

La función registrar_usuario() se encarga de registrar nuevos usuarios en la base de datos Firebase

1. Se obtiene el nombre de usuario y la contraseña que el usuario ha escrito en la interfaz gráfica mediante entry_usuario.get() y entry_contraseña.get().
2. Se verifica si el usuario dejó en blanco el campo de nombre o contraseña.

   Si algún campo está vacío, se muestra un mensaje de error y se interrumpe la función con return.

3. Se crea una referencia a la base de datos en la ruta "usuarios/{usuario}", donde se almacenan los datos del usuario.

   Se usa ref.get() para verificar si ya existe información en esa referencia, lo que indica que el usuario ya está registrado.

   Si el usuario ya existe, se muestra un mensaje de error y la función finaliza.

4. Si el usuario no existe en la base de datos, se guarda la contraseña en Firebase con

ref.set({"contraseña": contraseña}).

Se muestra un mensaje de confirmación de registro exitoso.

```
# funcion para abrir el menu principal
def abrir_menu_principal(usuario):
    root.withdraw()
    menu = tk.Tk()
    menu.title("PYNG-P0NG")
    menu.geometry("1000x650")
    menu.configure(bg="black")
    menu.iconbitmap("icono.ico")
#configuracion de interfaz tkinter
    label_Titulo = tk.Label(menu, text="P Y N G   P 0 N G", bg="red", font=("Calibri", 40, "bold"))
    label_Titulo.place(x=330, y=20)
    label_pie = tk.Label(menu, text="Developed by: Joan Muñoz", fg="red", bg="black", font=("Calibri", 20, "italic"))
    label_pie.place(x=690, y=600)
    label_pie = tk.Label(menu, text=f"Jugador \"{usuario}\"", fg="red", bg="black", font=("Calibri", 20, "italic"))
    label_pie.place(x=40, y=600)
    label_j1 = tk.Label(menu, text="J1", fg="red", bg="black", font=("Calibri", 20, "italic"))
    label_j1.place(x=450, y=425)
    label_j1 = tk.Label(menu, text="\'W\'", fg="red", bg="black", font=("Calibri", 20, "italic"))
    label_j1.place(x=450, y=470)
    label_j1 = tk.Label(menu, text="\'S\'", fg="red", bg="black", font=("Calibri", 20, "italic"))
    label_j1.place(x=450, y=520)
    label_up = tk.Label(menu, text="^", fg="red", bg="black", font=("Calibri", 20, "italic"))
    label_up.place(x=500, y=462)
    label_up = tk.Label(menu, text="|", fg="red", bg="black", font=("Calibri", 15, "italic"))
    label_up.place(x=503, y=482)
    label_do = tk.Label(menu, text="v", fg="red", bg="black", font=("Calibri", 18, "italic"))
    label_do.place(x=496, y=535)
    label_do = tk.Label(menu, text="|", fg="red", bg="black", font=("Calibri", 15, "italic"))
    label_do.place(x=500, y=517)
    label_j2 = tk.Label(menu, text="J2", fg="red", bg="black", font=("Calibri", 20, "italic"))
    label_j2.place(x=530, y=425)
```



```
label_j2 = tk.Label(menu, text="\'O\'", fg="red", bg="black", font=("Calibri", 20, "italic"))
label_j2.place(x=530, y=470)
label_j2 = tk.Label(menu, text="\'L\'", fg="red", bg="black", font=("Calibri", 20, "italic"))
label_j2.place(x=530, y=520)
label_control = tk.Label(menu, text="CONTROLES", fg="black", bg="red", font=("Calibri", 15, "italic"))
label_control.place(x=455, y=395)
label_linea = tk.Label(menu, text="----------------------------------------------------------------------------------------------------
label_linea.place(x=0, y=580)
btn_jugar = tk.Button(menu, text="Jugar J1 vs CPU", command=lambda: seleccionar_color("cpu", usuario, puntos_jugador1=0, puntos_ju
btn_jugar.place(x=410, y=110)
btn_jugar = tk.Button(menu, text="Jugar J1 vs J2", command=lambda: seleccionar_color("1v1", usuario, puntos_jugador1=0, puntos_jug
btn_jugar.place(x=425, y=180)
btn_marcadr = tk.Button(menu, text="Consultar Marcador",command=lambda: consultar_marcador(usuario), fg="red", bg="black", font=("
btn_marcadr.place(x=390, y=250)
btn_salir = tk.Button(menu, text="Salir", command=lambda: cerrar_menu(menu), fg="red", bg="black", font=("Calibri", 20, "bold"))
btn_salir.place(x=476, y=320)
```

La función abrir_menu_principal_() se encarga de ejecutar la ventana de menu principal.

- Oculta la principal ventana root.withdraw()
- Define las principales características de la ventana y el formato de sus elementos
- llama a las funciones como comandos de los botones principales

```python
# Funcion de cerrar el menu principal y volver al logueo
def cerrar_menu(menu):
    menu.destroy()

    entry_usuario.delete(0, tk.END)
    entry_contraseña.delete(0, tk.END)

    root.deiconify()
```

cerrar_menu() es la función que se ejecuta al presionar el botón salir del menú

- destruye la ventana del menú principal menu.destroy()
- inicializa los cuadros de texto en ceros
- vuelve a mostrar la ventana oculta root.deiconify

```python
# Función para seleccionar color y dificultad antes de partida
def seleccionar_color(modo, usuario, puntos_jugador1=0, puntos_jugador2=0):
    ventana_color = tk.Tk()
    ventana_color.title("Seleccionar colores")
    ventana_color.geometry("400x300")
    ventana_color.iconbitmap("icono.ico")
    ventana_color.configure(bg="black")

    tk.Label(ventana_color, text="Elige un tema:", fg="red", bg="black", font=("Calibri", 15)).pack(pady=10)

    lista=ttk.Combobox(ventana_color, values=["Rojo y Negro","Azul y Blanco","Verde y Amarillo","Morado y Celeste","Naranja y Gris"])
    lista.pack()
    tk.Label(ventana_color, text="Elige la dificultad:", fg="red", bg="black", font=("Calibri", 15)).pack()
    lista1=ttk.Combobox(ventana_color, values=["Facil","Dificil"])
    lista1.pack()
    #Función para confirmar las opciones elegidas dentro de la funcion
    def confirmar_opcion():
        opcion=lista.get()
        dificultad=lista1.get()
        if opcion=="Rojo y Negro":
            fondo=(0, 0, 0)
            pelota=(255, 255, 255)
            paleta=(255, 0, 0)
        elif opcion=="Azul y Blanco":
            fondo=(255, 255, 255)
            pelota=(0, 0, 255)
            paleta=(0, 0, 128)
        elif opcion=="Verde y Amarillo":
            fondo=(0, 128, 0)
            pelota=(255, 255, 0)
            paleta=(0, 255, 0)
```

```
69    elif opcion=="Morado y Celeste":
70        fondo=(128, 0, 128)
71        pelota=(0, 255, 255)
72        paleta=(75, 0, 130)
73    elif opcion=="Naranja y Gris":
74        fondo=(169, 169, 169)
75        pelota=(255, 165, 0)
76        paleta=(255, 140, 0)
77    if dificultad == "Facil":
78        velocidad=5
79        velocidad_jug=4
80        velocidad_cpu=2.5
81    elif dificultad == "Dificil":
82        velocidad=12
83        velocidad_jug=8
84        velocidad_cpu=6.5
85    ventana_color.destroy()
86    iniciar_juego(modo, fondo, pelota, paleta, velocidad, velocidad_jug, velocidad_cpu, usuario, puntos_jugador1=0, puntos_jugador2=0)
87
88    _confirmar = tk.Button(ventana_color, text="Confirmar",command=confirmar_opcion, fg="red", bg="black", font=("Calibri", 14, "bold"))
89    _confirmar.pack(pady=20)
```

La función seleccionar_color() permite al usuario elegir un esquema de colores y un nivel de dificultad antes de comenzar una partida de Pyng-P0ng.

- Se crea una nueva ventana de Tkinter con el título "Seleccionar colores", dimensiones de 400x300 píxeles y fondo negro.
- Se configura el ícono (icono.ico), que esta en el mismo directorio que el script.
- Se crea una etiqueta para indicar la selección del color.
- Se agrega un ttk.Combobox, un menú desplegable con cinco combinaciones de colores para la partida.
- Se añade otra etiqueta para indicar la selección de la dificultad.
- Se incluye otro ttk.Combobox para elegir entre "Fácil" y "Difícil"

  Función interna (confirmar_opcion()):

  - Esta función se ejecuta cuando el usuario presiona el botón "Confirmar".
  - Se obtienen las selecciones de color (opcion) y dificultad (dificultad).
  - Se asignan colores a los elementos del juego:
    - fondo (color de la pantalla de juego).
    - pelota (color de la pelota).
    - paleta (color de las paletas de los jugadores).
  - Cada esquema de color tiene su combinación específica
  - Fácil: Movimiento más lento para la pelota y los jugadores.
  - Difícil: Aumenta la velocidad de la pelota y de los jugadores.
  - Se cierra la ventana de selección (ventana_color.destroy()).

- Se llama a la función iniciar_juego(), pasando las configuraciones seleccionadas de color y dificultad

Se crea un botón "Confirmar", que ejecuta la función confirmar_opcion() cuando se presiona.

```python
#Funcion de interfaz de juego
def iniciar_juego(modo, COLOR_FONDO, COLOR_BOLA, COLOR_PALETA, VELOCIDAD_BOLA,
                  VELOCIDAD_PALETA, VELOCIDAD_CPU, usuario, puntos_jugador1=0, puntos_jugador2=0):
    pygame.init()
    ventana = pygame.display.set_mode((ANCHO, ALTO))
    pygame.display.set_caption("PYNG-P0NG")

    jugador1 = pygame.Rect(30, ALTO // 2 - 60, 20, 120)
    jugador2 = pygame.Rect(ANCHO - 50, ALTO // 2 - 60, 20, 120)
    bola = pygame.Rect(ANCHO // 2 - 15, ALTO // 2 - 15, 30, 30)
    bola_dx, bola_dy = VELOCIDAD_BOLA, VELOCIDAD_BOLA

    puntos_jugador1, puntos_jugador2 = 0, 0
    inicio_tiempo = pygame.time.get_ticks()
    reloj = pygame.time.Clock()

    while True:
        tiempo_actual = pygame.time.get_ticks()
        tiempo_transcurrido = (tiempo_actual - inicio_tiempo) // 1000  # En segundos

        if tiempo_transcurrido >= 120 or puntos_jugador1 >= 10 or puntos_jugador2 >= 10:
            pygame.quit()
            guardar_marcador(usuario, puntos_jugador1, puntos_jugador2)
            messagebox.showinfo("Juego Terminado",
                                f"Marcador Final: {puntos_jugador1} - {puntos_jugador2}
                                "\nPresiona ACEPTAR para volver al menú principal.")
            return
```

```
for evento in pygame.event.get():
    if evento.type == pygame.QUIT:
        pygame.quit()
        sys.exit()

teclas = pygame.key.get_pressed()
if teclas[pygame.K_w] and jugador1.top > 0:
    jugador1.y -= VELOCIDAD_PALETA
if teclas[pygame.K_s] and jugador1.bottom < ALTO:
    jugador1.y += VELOCIDAD_PALETA
if teclas[pygame.K_o] and jugador2.top > 0:
    jugador2.y -= VELOCIDAD_PALETA
if teclas[pygame.K_l] and jugador2.bottom < ALTO:
    jugador2.y += VELOCIDAD_PALETA

if modo == "cpu":
    if jugador2.centery < bola.centery:
        jugador2.y += VELOCIDAD_CPU
    elif jugador2.centery > bola.centery:
        jugador2.y -= VELOCIDAD_CPU

bola.x += bola_dx
bola.y += bola_dy

if bola.top <= 0 or bola.bottom >= ALTO:
    bola_dy = -bola_dy

if bola.colliderect(jugador1):
    bola_dx = abs(bola_dx)
elif bola.colliderect(jugador2):
    bola_dx = -abs(bola_dx)
```

```
if bola.left <= 0:
    puntos_jugador2 += 1
    bola.x, bola.y = ANCHO // 2 - 15, ALTO // 2 - 15
    bola_dx = VELOCIDAD_BOLA
elif bola.right >= ANCHO:
    puntos_jugador1 += 1
    bola.x, bola.y = ANCHO // 2 - 15, ALTO // 2 - 15
    bola_dx = -VELOCIDAD_BOLA

ventana.fill(COLOR_FONDO)
pygame.draw.rect(ventana, COLOR_PALETA, jugador1)
pygame.draw.rect(ventana, COLOR_PALETA, jugador2)
pygame.draw.ellipse(ventana, COLOR_BOLA, bola)

fuente = pygame.font.Font(None, 36)
texto = fuente.render(f"{puntos_jugador1} - {puntos_jugador2} | Tiempo: {120 - tiempo_transcurrido}s", True, COLOR_BOLA)
ventana.blit(texto, (ANCHO // 2 - 100, 20))

pygame.display.flip()
reloj.tick(60)
```

iniciar_juego() es el núcleo del juego Pyng-P0ng

1. Inicialización y Configuración de Pygame

- Se inicializa Pygame y se configura la ventana del juego con un tamaño (ANCHO, ALTO).
- Se le asigna un título "PYNG-P0NG".

2. Creación de Elementos del Juego

- Se crean los rectángulos de los jugadores y la pelota.
- Se definen las posiciones iniciales.

3. Variables del Juego

- Se establecen las velocidades de la pelota en x y y.
- Se inicializan los puntajes.
- Se obtiene el tiempo de inicio para controlar la duración del juego.
- Se usa pygame.time.Clock() para controlar los FPS.

4. Control del Tiempo y Fin del Juego

- Se calcula el tiempo transcurrido en segundos.
- Condición de fin de juego:
    - 120 segundos transcurridos.
    - Un jugador alcanza 10 puntos.
- Se cierra Pygame y se guarda el marcador en Firebase.

5. Controles del Jugador

- Se usa pygame.key.get_pressed() para leer las teclas presionadas.

6. CPU

- Si el modo es "cpu", el jugador 2 se mueve automáticamente
- Sigue la posición de la pelota con una velocidad VELOCIDAD_CPU

7. Movimiento de la Pelota

- La pelota se mueve sumando su velocidad a las coordenadas x e y.

8. Rebote contra Bordes Superiores/Inferiores

- Si la pelota toca el borde superior o inferior, se invierte su dirección (bola_dy).

9. Rebote contra los Jugadores

- Si la pelota colisiona con un jugador, cambia su dirección.

10. Puntos y Reinicio de la Pelota

- Si la pelota sale por la izquierda, el jugador 2 gana un punto.
- Si la pelota sale por la derecha, el jugador 1 gana un punto.
- La pelota se reinicia al centro.

11. Actualizar la Pantalla y Control de FPS

- Se actualiza la pantalla con pygame.display.flip().
- Se limita la velocidad del juego a 60 FPS con reloj.tick(60).

```python
# Función para guardar marcador en Firebase
def guardar_marcador(usuario, puntos_jugador1, puntos_jugador2):
    ref = db.reference(f"usuarios/{usuario}/marcadores")
    marcador = {"puntos_jugador1": puntos_jugador1, "puntos_jugador2": puntos_jugador2}
    ref.push(marcador)
```

- guardar_marcador(usuario, puntos_jugador1, puntos_jugador2) almacena correctamente los marcadores de un usuario en Firebase bajo la ruta: usuarios/{usuario}/marcadores
- Cada marcador se guarda como un nuevo nodo con un identificador único generado por push().

```python
# Función para consultar marcadores
def consultar_marcador(usuario):
    ref = db.reference(f"usuarios/{usuario}/marcadores")
    marcadores = ref.get()
    if not marcadores:
        messagebox.showinfo("Marcadores", "No hay marcadores guardados.")
        return
    ultimos_5 = list(marcadores.values())[-5:]
    texto_marcadores = "\n".join([f"Partida : {m['puntos_jugador1']} - {m['puntos_jugador2']}" for i, m in enumerate(ultimos_5)])
    messagebox.showinfo("Últimos 5 Marcadores", texto_marcadores)
```

Esta función consulta los últimos 5 marcadores guardados en Firebase para un usuario específico y los muestra en un messagebox.

Conclusión

Conclusión General

El proyecto PYNG-P0NG integra Python, Pygame, Tkinter y Firebase para ofrecer una experiencia de juego interactiva con registro de marcadores en la nube. Los jugadores pueden competir contra otro jugador o la CPU, mientras que los resultados se almacenan y pueden consultarse fácilmente. La combinación de Firebase permite un almacenamiento eficiente de datos, y la interfaz gráfica brinda una navegación intuitiva.