



LABORATORIO 3 PROGRAMACION

CARLOS MANUEL DIAZ, ALVARO ANDRES GUZMAN

Resumen— Esta práctica de laboratorio implementa los métodos aprendidos durante todo el curso especialmente la programación orientada a objetos que será el método principal para llevar a cabo la práctica, se continua con la creación de una interfaz de usuario que tiene como función iniciar sesión, registrar, obtener preguntas de manera aleatoria de una categoría, obtener puntaje e información de usuario, se utiliza la librería principal llamada tkinter que permite la creación de la clase trivia que será la encargada de mantener el código de la interfaz, se utilizan funciones para hacer legible y consumir menos memoria en el código, se crean objetos dentro de la clase como por ejemplo etiquetas de texto, botones, barra de carga y carga de imágenes, las categorías de preguntas y usuarios estarán almacenados en archivos de texto lo cual hará que sea más fácil acceder a ellos.

Palabras clave—TKINTER, ARCHIVOS, FUNCIONES, INTERFAZ, USUARIO, OBJETOS, PROGRAMACION, LIBRERÍA, METODO.

I. INTRODUCCIÓN

La programación orientada a objetos es un modelo de programación que brinda las guías para trabajar con él se basa en la creación de objetos y clases en la actualidad se usa especialmente en la creación de aplicaciones, interfaces también permite una mejor estructura de código y evitar la reutilización de funciones en el programa [1]

En la práctica de laboratorio se propone el desarrollo de una interfaz gráfica para observar la funcionalidad de las funciones del código users.py mediante la interfaz que permitirá interactuar al usuario de manera dinámica , respondiendo preguntas, obteniendo puntaje , abrir , cerrar sesión y registrar otro usuario , se utilizan métodos en la programación orientada

a objetos como lo es la librería tkinter que es la que permitirá la visualización de todos los objetos que hacen la interfaz , pillow que es la librería para importar una imagen desde la dirección en donde se encuentre guardada , se importa trivia_client el cual devuelve las funciones de users mediante un servidor local.

• OBJETIVOS GENERAL

Desarrollar un programa que refleje la funcionalidad del juego de preguntas desarrollado en el laboratorio 2, esto mediante una interfaz gráfica que le permitirá al usuario interactuar con el programa.

ESPECÍFICOS

- Hacer uso correcto de las funciones que contiene la programación orientada a objetos para realizar la interfaz.
- Crear una clase, definir objetos y atributos correctamente.
- Diseñar y desarrollar una interfaz gráfica interactiva para un juego en Python, utilizando bibliotecas como tkinter

II. INSTRUMENTOS

- II. Computador
- II. Entornos de desarrollo como visual studio
- II. PYTHON
- II. Lenguaje C++

III. MÉTODOS.

- Desarrollar una interfaz gráfica mediante la biblioteca tkinter que permita al usuario interactuar con la funcionalidad del programa.
- Hacer uso de funciones para cada una de las funcionalidades del juego de preguntas.



PROGRAMA DE INGENIERÍA ELÉCTRICA

Acreditado de Alta Calidad
Fes, MEX. 11/05 – 17/Julio 2010 (3 años)

- Creación de la clase triviapp que será encargada de contener el código para la interfaz y la clase Question para el almacenamiento de preguntas.
- Carga de una imagen para ser utilizada como fondo en la ventana de la interfaz.
- Creación barra de carga que simulará el proceso de carga del juego.
- Creación de etiquetas de texto, botones que permitirán el acceso a las funciones del programa.
- Hacer uso de excepciones en el código para detectar errores.

Código desarrollado para la interfaz de juego:

```
import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk
import time
import trivi_client

class Question:
    def __init__(self, text, options, correct_answer):
        self.text = text
        self.options = options
        self.correct_answer = correct_answer

class TriviaApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Trivia")
        self.root.geometry("800x800")
        self.server_url = "http://localhost:80"
        self.current_user = {"name": "", "password": "", "score": 0}
        self.questions = [] # Lista de preguntas
        self.question_number = 0
        self.correct_answers = 0 # Contador de respuestas correctas
        self.incorrect_answers = 0 # Contador de respuestas incorrectas

        # Configurar la interfaz
        self.set_bg()
        self.fill_progress_bar()

    def set_bg(self):
        img = Image.open(r'C:\Users\PERSONAL\Downloads\preguntas.jpg.jpg')
        img = img.resize((800, 800))
        self.bg_image = ImageTk.PhotoImage(img)
        self.label_bg = tk.Label(self.root, image=self.bg_image)
        self.label_bg.place(relwidth=1, relheight=1)

    def resize_image(self, image, width, height):
        img = Image.open(image)
        img = img.resize((width, height), Image.Resampling.LANCZOS)
        return img

    def fill_progress_bar(self):
        self.clear_window()
        tk.Label(self.root, text="Cargando...", font=("Arial", 12), bg="lightblue").pack(pady=20)

        progress_bar = ttk.Progressbar(self.root, orient="horizontal", length=400, mode="determinate")
        progress_bar.pack(pady=20)

        for i in range(100):
            progress_bar["value"] = i
            self.root.update()
            time.sleep(0.03)

        self.show_main_menu()

    def clear_window(self):
        for widget in self.root.winfo_children():
            if widget != self.label_bg:
                widget.destroy()

    def update_info(self, message):
        self.clear_window()
        tk.Label(self.root, text=message, font=("Arial", 14), bg="lightblue").pack(pady=20)

    def show_main_menu(self):
        self.clear_window()
        tk.Label(self.root, text="Menú Principal", font=("Arial", 18, "bold"), bg="lightblue").pack(pady=20)
        tk.Button(self.root, text="Iniciar Sesión", font=("Arial", 14), command=self.show_login_form).pack(pady=10)
        tk.Button(self.root, text="Registrarse", font=("Arial", 14), command=self.show_register_form).pack(pady=10)

    def show_login_form(self):
        self.login_action()
        name = entry_name.get()

    def login_action():
        password = entry_password.get()
        response = trivi_client.openSession(self.server_url, name, password)

        if response.lower().strip() == "sesión iniciada":
            self.current_user["name"] = self.current_user["password"] = name, password
            self.update_info("Sesión iniciada exitosamente.")
            self.show_user_menu()
        else:
            self.update_info("Error al iniciar sesión. Verifique su nombre o contraseña.")

        self.clear_window()
        tk.Label(self.root, text="Iniciar Sesión", font=("Arial", 12, "bold"), bg="lightblue").pack(pady=20)
        tk.Label(self.root, text="Nombre:", font=("Arial", 12), bg="lightblue").pack(pady=5)
        entry_name = tk.Entry(self.root, font=("Arial", 12))
        entry_name.pack(pady=5)

        tk.Label(self.root, text="Contraseña:", font=("Arial", 12), bg="lightblue").pack(pady=5)
        entry_password = tk.Entry(self.root, font=("Arial", 12), show="*")
        entry_password.pack(pady=5)

        tk.Button(self.root, text="Iniciar Sesión", font=("Arial", 12), command=login_action).pack(pady=10)
        tk.Button(self.root, text="Volver", font=("Arial", 12), command=self.show_main_menu).pack(pady=10)

    def show_register_form(self):
        self.register_action()
        name = entry_name.get()
        password = entry_password.get()
        response = trivi_client.registerUser(self.server_url, name, password)
        self.update_info("Registro: " + response)
        self.show_main_menu()

    def register_action():
        name = entry_name.get()
        password = entry_password.get()
        response = trivi_client.registerUser(self.server_url, name, password)
        self.update_info("Registro: " + response)
        self.show_main_menu()

    def show_user_menu(self):
        self.clear_window()
        tk.Label(self.root, text="Bienvenido", font=(self.current_user["name"])), font=("Arial", 18, "bold"), bg="lightblue").pack(pady=20)
        tk.Label(self.root, text="Nombre:", font=("Arial", 12), bg="lightblue").pack(pady=5)
        entry_name = tk.Entry(self.root, font=("Arial", 12))
        entry_name.pack(pady=5)

        tk.Label(self.root, text="Contraseña:", font=("Arial", 12), bg="lightblue").pack(pady=5)
        entry_password = tk.Entry(self.root, font=("Arial", 12), show="*")
        entry_password.pack(pady=5)

        tk.Button(self.root, text="Registrar", font=("Arial", 12), command=register_action).pack(pady=10)
        tk.Button(self.root, text="Volver", font=("Arial", 12), command=self.show_main_menu).pack(pady=10)

    def show_score(self):
        self.update_info()
        new_score = entry_score.get()
        if new_score.isdigit():
            trivi_client.updateScore(self.server_url, self.current_user["name"], self.current_user["password"], int(new_score))
            score_label.config(text="Puntaje actualizado a: " + new_score)
        else:
            error_label.config(text="Por favor, ingrese un puntaje válido.")

        self.clear_window()
        tk.Label(self.root, text="Nuevo puntaje:", font=("Arial", 12, "bold"), bg="lightblue").pack(pady=20)
        current_score = trivi_client.getScore(self.server_url, self.current_user["name"], self.current_user["password"])
        score_label = tk.Label(self.root, text="Puntaje actual: (" + current_score + ", font=("Arial", 14, "bold"), bg="lightblue")
        score_label.pack(pady=20)

        tk.Label(self.root, text="Nuevo puntaje:", font=("Arial", 12, "bold"), bg="lightblue").pack(pady=5)
        entry_score = tk.Entry(self.root, font=("Arial", 12))
        entry_score.pack(pady=5)

        tk.Button(self.root, text="Actualizar Puntaje", font=("Arial", 12), command=update_score).pack(pady=10)
        error_label = tk.Label(self.root, text="Error", font="red", bg="lightblue")
        error_label.pack(pady=5)

    def update_score():
        new_score = entry_score.get()
        if new_score.isdigit():
            trivi_client.updateScore(self.server_url, self.current_user["name"], self.current_user["password"], int(new_score))
            score_label.config(text="Puntaje actualizado a: " + new_score)
        else:
            error_label.config(text="Por favor, ingrese un puntaje válido.")

    def choose_category(self):
        self.clear_window()
        tk.Label(self.root, text="Elegir una categoría:", font=("Arial", 16, "bold"), bg="lightblue").pack(pady=20)
        tk.Button(self.root, text="Categoría 1", font=("Arial", 14), command=lambda: self.start_quiz(1)).pack(pady=10)
        tk.Button(self.root, text="Categoría 2", font=("Arial", 14), command=lambda: self.start_quiz(2)).pack(pady=10)

    def start_quiz(self, category):
        self.questions = self.load_questions(category)
        self.question_number = 0
        self.correct_answers = 0
        self.incorrect_answers = 0

        if not self.questions:
            self.update_info("No se encontraron preguntas en la categoría seleccionada.")
        else:
            self.ask_question()

    def load_questions(self, cat):
        file_map = {1: "preguntas_categoria_1.txt", 2: "preguntas_categoria_2.txt"}
        selected_file = file_map.get(cat)

        if not selected_file:
            return []

        questions = []
        try:
            with open(selected_file, "r", encoding="utf-8") as file:
                question_buffer = []
                for line in file:
                    line = line.strip()
                    if not line:
                        question_buffer.append(line)
                    if len(question_buffer) == 6: # Una pregunta completa
                        text, options, correct = question_buffer
                        questions.append(Question(text, options, correct))
                        question_buffer.clear()
        except FileNotFoundError:
            return []

        return questions

    def ask_question(self):
        if self.question_number >= len(self.questions):
            self.finish_quiz()
            return

        current_question = self.questions[self.question_number]
        self.clear_window()

        tk.Label(self.root, text="Pregunta (" + str(self.question_number + 1) + ", font=("Arial", 16, "bold"), bg="lightblue").pack(pady=20)
        tk.Label(self.root, text=current_question.text, font=("Arial", 14), wraplength=600, bg="lightblue").pack(pady=10)
```

```
password = entry_password.get()
response = trivi_client.openSession(self.server_url, name, password)

if response.lower().strip() == "sesión iniciada":
    self.current_user["name"] = self.current_user["password"] = name, password
    self.update_info("Sesión iniciada exitosamente.")
    self.show_user_menu()
else:
    self.update_info("Error al iniciar sesión. Verifique su nombre o contraseña.")

self.clear_window()
tk.Label(self.root, text="Iniciar Sesión", font=("Arial", 12, "bold"), bg="lightblue").pack(pady=20)
tk.Label(self.root, text="Nombre:", font=("Arial", 12), bg="lightblue").pack(pady=5)
entry_name = tk.Entry(self.root, font=("Arial", 12))
entry_name.pack(pady=5)

tk.Label(self.root, text="Contraseña:", font=("Arial", 12), bg="lightblue").pack(pady=5)
entry_password = tk.Entry(self.root, font=("Arial", 12), show="*")
entry_password.pack(pady=5)

tk.Button(self.root, text="Iniciar Sesión", font=("Arial", 12), command=login_action).pack(pady=10)
tk.Button(self.root, text="Volver", font=("Arial", 12), command=self.show_main_menu).pack(pady=10)

def show_register_form(self):
    self.register_action()
    name = entry_name.get()
    password = entry_password.get()
    response = trivi_client.registerUser(self.server_url, name, password)
    self.update_info("Registro: " + response)
    self.show_main_menu()

    def register_action():
        name = entry_name.get()
        password = entry_password.get()
        response = trivi_client.registerUser(self.server_url, name, password)
        self.update_info("Registro: " + response)
        self.show_main_menu()

    def show_user_menu(self):
        self.clear_window()
        tk.Label(self.root, text="Bienvenido", font=(self.current_user["name"])), font=("Arial", 18, "bold"), bg="lightblue").pack(pady=20)
        tk.Label(self.root, text="Nombre:", font=("Arial", 12), bg="lightblue").pack(pady=5)
        entry_name = tk.Entry(self.root, font=("Arial", 12))
        entry_name.pack(pady=5)

        tk.Label(self.root, text="Contraseña:", font=("Arial", 12), bg="lightblue").pack(pady=5)
        entry_password = tk.Entry(self.root, font=("Arial", 12), show="*")
        entry_password.pack(pady=5)

        tk.Button(self.root, text="Registrar", font=("Arial", 12), command=register_action).pack(pady=10)
        tk.Button(self.root, text="Volver", font=("Arial", 12), command=self.show_main_menu).pack(pady=10)

    def show_score(self):
        self.update_info()
        new_score = entry_score.get()
        if new_score.isdigit():
            trivi_client.updateScore(self.server_url, self.current_user["name"], self.current_user["password"], int(new_score))
            score_label.config(text="Puntaje actualizado a: " + new_score)
        else:
            error_label.config(text="Por favor, ingrese un puntaje válido.")

        self.clear_window()
        tk.Label(self.root, text="Nuevo puntaje:", font=("Arial", 12, "bold"), bg="lightblue").pack(pady=20)
        current_score = trivi_client.getScore(self.server_url, self.current_user["name"], self.current_user["password"])
        score_label = tk.Label(self.root, text="Puntaje actual: (" + current_score + ", font=("Arial", 14, "bold"), bg="lightblue")
        score_label.pack(pady=20)

        tk.Label(self.root, text="Nuevo puntaje:", font=("Arial", 12, "bold"), bg="lightblue").pack(pady=5)
        entry_score = tk.Entry(self.root, font=("Arial", 12))
        entry_score.pack(pady=5)

        tk.Button(self.root, text="Actualizar Puntaje", font=("Arial", 12), command=update_score).pack(pady=10)
        error_label = tk.Label(self.root, text="Error", font="red", bg="lightblue")
        error_label.pack(pady=5)

    def update_score():
        new_score = entry_score.get()
        if new_score.isdigit():
            trivi_client.updateScore(self.server_url, self.current_user["name"], self.current_user["password"], int(new_score))
            score_label.config(text="Puntaje actualizado a: " + new_score)
        else:
            error_label.config(text="Por favor, ingrese un puntaje válido.")

    def choose_category(self):
        self.clear_window()
        tk.Label(self.root, text="Elegir una categoría:", font=("Arial", 16, "bold"), bg="lightblue").pack(pady=20)
        tk.Button(self.root, text="Categoría 1", font=("Arial", 14), command=lambda: self.start_quiz(1)).pack(pady=10)
        tk.Button(self.root, text="Categoría 2", font=("Arial", 14), command=lambda: self.start_quiz(2)).pack(pady=10)

    def start_quiz(self, category):
        self.questions = self.load_questions(category)
        self.question_number = 0
        self.correct_answers = 0
        self.incorrect_answers = 0

        if not self.questions:
            self.update_info("No se encontraron preguntas en la categoría seleccionada.")
        else:
            self.ask_question()

    def load_questions(self, cat):
        file_map = {1: "preguntas_categoria_1.txt", 2: "preguntas_categoria_2.txt"}
        selected_file = file_map.get(cat)

        if not selected_file:
            return []

        questions = []
        try:
            with open(selected_file, "r", encoding="utf-8") as file:
                question_buffer = []
                for line in file:
                    line = line.strip()
                    if not line:
                        question_buffer.append(line)
                    if len(question_buffer) == 6: # Una pregunta completa
                        text, options, correct = question_buffer
                        questions.append(Question(text, options, correct))
                        question_buffer.clear()
        except FileNotFoundError:
            return []

        return questions

    def ask_question(self):
        if self.question_number >= len(self.questions):
            self.finish_quiz()
            return

        current_question = self.questions[self.question_number]
        self.clear_window()

        tk.Label(self.root, text="Pregunta (" + str(self.question_number + 1) + ", font=("Arial", 16, "bold"), bg="lightblue").pack(pady=20)
        tk.Label(self.root, text=current_question.text, font=("Arial", 14), wraplength=600, bg="lightblue").pack(pady=10)
```



```

def answer_action(selected_option):
    if selected_option == current_question.correct_answer:
        self.correct_answers += 1
    else:
        self.incorrect_answers += 1

    self.question_number += 1
    self.ask_question()

    for option in current_question.options:
        tk.Button(self.root, text=option, font=("Arial", 12), wraplength=300,
                  command=lambda opt=option: answer_action(opt)).pack(pady=5)

def finish_quiz(self):
    self.clear_window()
    tk.Label(self.root, text="Juego terminado", font=("Arial", 16, "bold"), bg="lightblue").pack(pady=20)
    tk.Label(self.root, text="Respuestas correctas: " + str(self.correct_answers), font=("Arial", 14), bg="lightblue").pack(pady=10)
    tk.Label(self.root, text="Respuestas incorrectas: " + str(self.incorrect_answers), font=("Arial", 14), bg="lightblue").pack(pady=10)
    tk.Button(self.root, text="Volver al Menú", font=("Arial", 14), command=self.show_user_menu).pack(pady=10)

def get_connected_users(self):
    self.clear_window()
    tk.Label(self.root, text="Usuarios conectados", font=("Arial", 16, "bold"), bg="lightblue").pack(pady=20)
    users = trivia_client.getConnectedUsers(self.server_url)
    for user in users:
        tk.Label(self.root, text=user, font=("Arial", 14), bg="lightblue").pack(pady=5)
    tk.Button(self.root, text="Volver al Menú", font=("Arial", 14), command=self.show_user_menu).pack(pady=10)

def logout(self):
    response = trivia_client.closeSession(self.server_url, self.current_user["name"], self.current_user["password"])
    self.update_info(response)
    self.show_main_menu()

if __name__ == "__main__":
    root = tk.Tk()
    app = TriviaApp(root)
    root.mainloop()

```

Funcionalidad del código:

Se importa las librerías necesarias para la creación de la interfaz y también se importa el código `trivia_client` que es el encargado de devolver la funcionalidad de las funciones definidas en `users.py` esto mediante un servidor de red local, se crea la clase `Question` que va a contener las preguntas y respuestas dentro de la interfaz.

Clase TriviaApp

Es la clase en donde estará contenido el programa encargado de la creación de la interfaz dentro de ella los atributos definidos son:

- `Self.root`: Ventana en donde se vera reflejada la funcionalidad de las funciones del juego.
- `Self.server_url`: Url que es la dirección del servidor que maneja todas las peticiones que se hace mediante `trivia_client` para que devuelva las funciones de `users` que tienen la dinámica del juego.
- `self.current_user`: Un diccionario en donde se almacenará la información del usuario actual como nombre, contraseña, puntaje
- `self.question`: Es una lista, que almacenara las preguntas cargadas para el juego.
- `Self.question_number`: contador del número actual de la pregunta que se esta llevando
- `Self_correct_answer`: contador de respuestas correctas

- `Self_incorrect_answer`: contador de respuestas incorrectas dadas por el usuario
- `self.bg_image`: Es la imagen de fondo cargada para la interfaz gráfica.

Objetos creados en el programa

Los objetos creados en el programa son creados mediante la librería `tkinter` que contiene todos los widgets para la creación de interfaces graficas.

- `root`: Es la ventana definida dentro de la clase madre que tiene como función reflejar las demás funciones para el juego.
- `Self.bg_image`: imagen cargada como fondo de la ventana que se mantendrá durante toda la ejecución del programa.
- `Widgets`: Se crean etiquetas de texto y para mostrar imágenes en el caso del programa el texto en donde se reflejarán las preguntas y las demás instrucciones, se crean botones que al presionarlos van a retornar las funciones definidas en el código, se crea una barra de progreso que va a simular un proceso de carga de 0 a 100% con una ejecución cada 0.03 segundos para poder observar la simulación.

Funciones implementadas en el código:

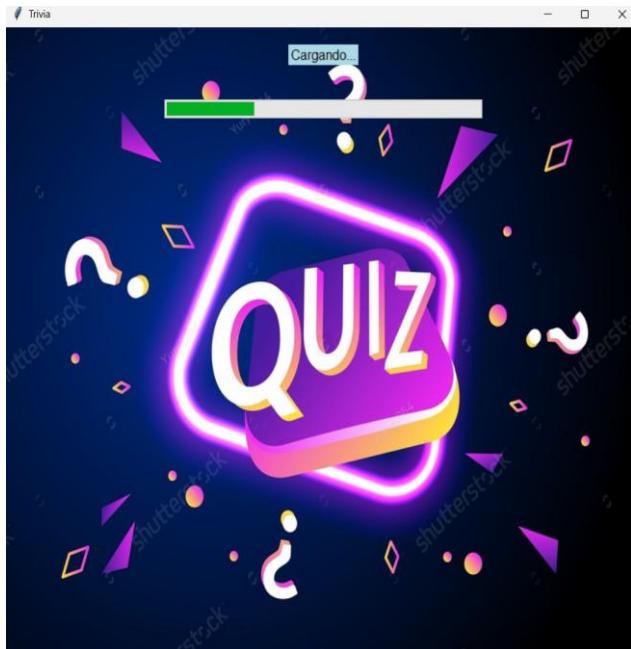
- `show_main_menu(self)`: La función se encarga de mostrar el menú principal del programa. Aquí se colocan botones como "Iniciar Sesión", "Registrar Usuario" y "Salir", permitiendo al usuario seleccionar una opción. Esta función organiza la interfaz para facilitar la navegación por las diferentes secciones del juego
- `fill_progress_bar(self)`: Esta función simula una barra de progreso que se llena gradualmente de 0 a 100. Es parte de una animación que se muestra antes de que el juego comience. Una vez que la barra se completa, la función llama a la función del menú para iniciar el juego, lo que da al usuario una sensación de espera antes de empezar.



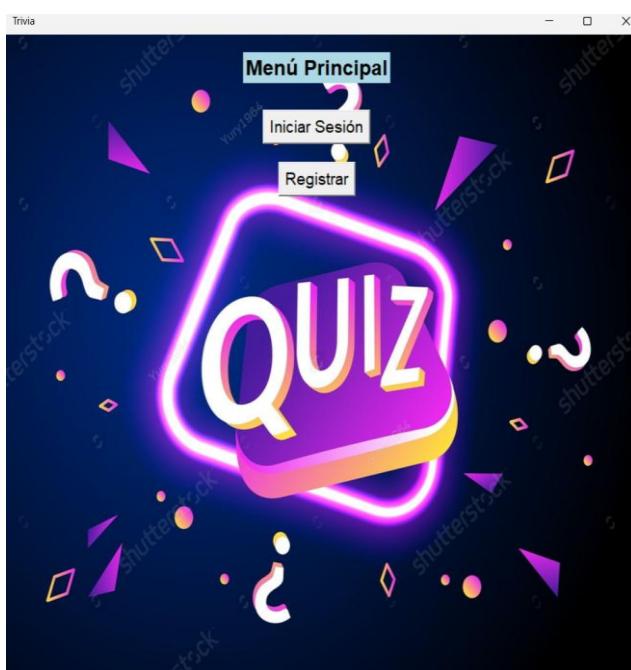
- **show_question(self):** La función se encarga de mostrar una pregunta del juego en la interfaz. Muestra la pregunta junto con las opciones de respuesta, las cuales son botones que permiten al usuario seleccionar su respuesta. Esta función es fundamental para interactuar con el usuario durante el juego.
- **load_questions(self, filename):** La función load_questions() se encarga de cargar las preguntas desde un archivo. Este archivo contiene tanto las preguntas como las opciones de respuesta. La función lee el archivo y extrae las preguntas y sus opciones, almacenándolas en la lista self.questions, lo que facilita su uso posterior en el juego también la función maneja una excepción la cual es validar que existe una pregunta dentro del archivo leído, de lo contrario retornará la excepción que será error.
- **start_quiz(self):** Esta función inicia el juego. Al ser llamada, reinicia todos los contadores necesarios (como el de respuestas correctas) y luego muestra la primera pregunta llamando a show_question(). Su función principal es dar inicio al juego y gestionar el flujo de preguntas y respuestas.
- **check_answer(self, selected_option):** verifica si la respuesta seleccionada por el usuario es correcta. Compara la opción elegida con la respuesta correcta y actualiza los contadores de respuestas correctas e incorrectas. Al final, muestra la siguiente pregunta llamando a show_question().
- **Finish_quiz(self):** La función es responsable de finalizar el quiz y mostrar los resultados. Al ser llamada, muestra el puntaje final del usuario, es decir, cuántas preguntas respondió correctamente y cuántas incorrectamente y muestra el botón para volver al menú principal.
- **show_register_form:** Muestra el formulario de registro donde el usuario puede ingresar su nombre y contraseña. Al completar el formulario, la función interna register_action recoge los datos y los envía al servidor mediante la función registerUser para registrar al usuario. Luego, muestra la respuesta del servidor y lo devuelve al menú inicial para iniciar sesión.
- **show_login_form:** muestra el formulario de inicio de sesión donde el usuario puede ingresar su nombre y contraseña. Al hacer clic en el botón de inicio de sesión, la función interna login_action recoge los datos y los envía al servidor mediante la función openSession para verificar las credenciales. Si la sesión se inicia correctamente, se guarda el nombre y la contraseña del usuario en el diccionario current_user, y se muestra un mensaje de éxito. Luego, se muestra el menú del usuario. Si las credenciales son incorrectas, se muestra un mensaje de error solicitando al usuario que verifique su nombre o contraseña.
- **Logout:** cierra la sesión del usuario actual. Envía una solicitud al servidor utilizando la función closeSession, pasando el nombre de usuario y la contraseña almacenados en el diccionario current_user. Luego, muestra la respuesta del servidor (que podría ser un mensaje de éxito o error) mediante la función update_info. Finalmente, se muestra el menú principal, regresando al punto inicial de la aplicación.



Pruebas de escritorio interfaz:

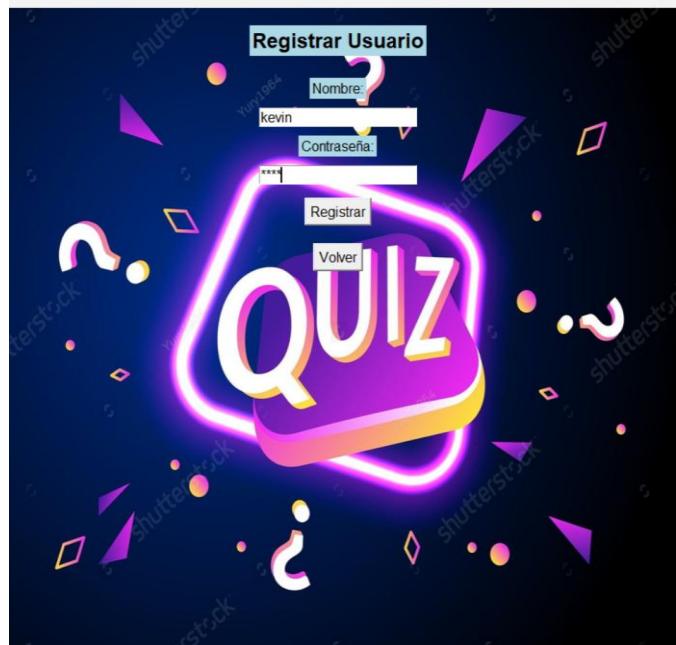


Al iniciar el programa de la interfaz se puede observar la creación de la ventana que permitirá interactuar al usuario, una etiqueta de texto mostrando un mensaje y la creación de la barra de progreso llamando a la función `fill_progress_bar(self)` que es la encargada de simular la carga de 0 a 100%.

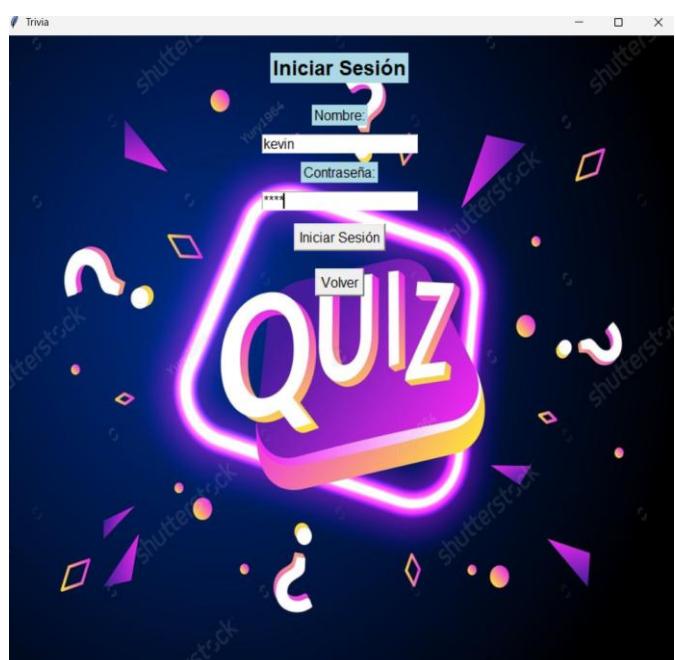


Luego de llenarse la barra de carga la ventana va a llamar a `self_clear_window` para que limpie la ventana es decir borrar el contenido y poder imprimir el contenido de otra función cual se encarga de mostrar el menú principal junto con

dos botones para iniciar sesión y para registrar usuario.



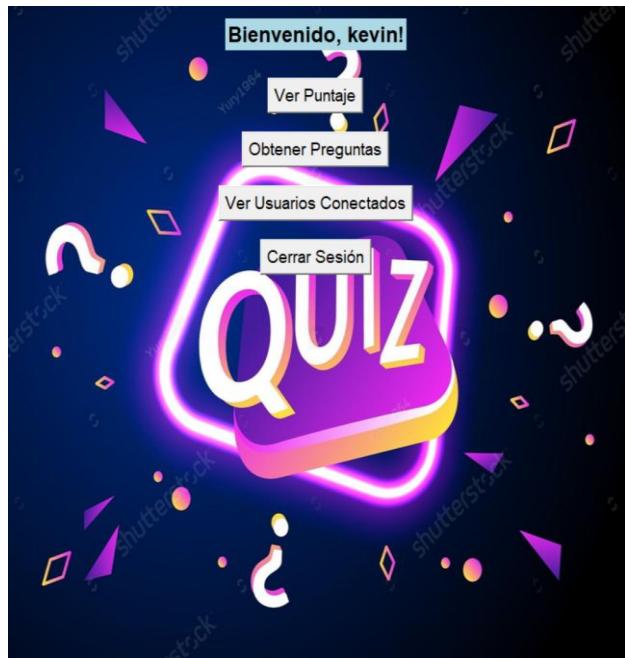
Al seleccionar el botón de registrar usuario llama a la función `show_register_form(self)` la cual es la encargada de mandar la solicitud al servidor para el registro del usuario para que pueda iniciar sesión de manera satisfactoria y si se retorna correctamente devolverá al inicio para iniciar sesión y poder jugar.



Al presionar el botón de inicio de sesión despliega la opción para que el usuario ingrese su nombre y contraseña para iniciar sesión llamando a la función `show_login_form(self)` que mandara la



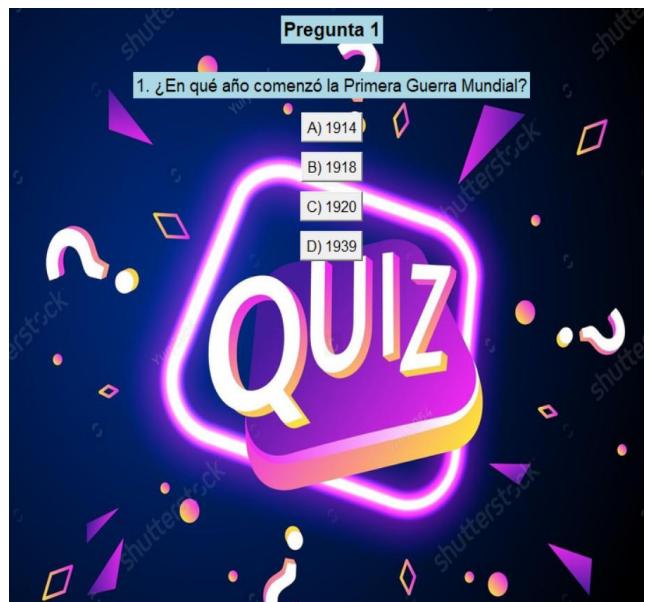
solicitud al servidor para poder ingresar, va a tener en cuenta parámetros como lo son nombre y contraseña, si el usuario se registro con una contraseña e ingresa otra no va a permitir acceder ya que las credenciales serian incorrectas.



Al ser los parámetros correctos (nombre, contraseña) la función va a devolver sesión iniciada posteriormente se va a implementar la función `show_user_menu(self)` la cual es la encargada de imprimir el menú principal de usuario se puede observar etiqueta de texto con mensaje de bienvenida y nombre del usuario y hay varios botones para elegir la opción requerida.



Al seleccionar la primera opción en el menú principal que es “ver puntaje” podemos mirar el puntaje del usuario y actualizarlo para realizar este paso es necesario implementar la función `view_score(self)` en ella este contenido el paso para mostrar el puntaje actual del usuario mediante la solicitud al servidor de la función `getScore` y la función `updateScore` se encargara de actualizar el nuevo puntaje y guardarlo.



Al seleccionar la segunda opción la cual es obtener preguntas y nos permite elegir si categoría 1 o 2 según la opción elegida la función `load_question` se encargará de leer el archivo de preguntas según la categoría seleccionada y mostrarla en la ventana de la interfaz.





Al seleccionar la última opción que es cerrar sesión se llama a la función logout la cual hace una solicitud al servidor para cerrar sesión teniendo en cuenta los mismos parámetros (nombre, contraseña) los cuales al ser correctos retornara la función un cierre se sesión exitosa.

VI. CONCLUSIONES

- La implementación de funciones para gestionar archivos y sesiones permitió comprender de manera práctica el flujo de información entre cliente y servidor, reforzando los conceptos teóricos vistos en clase.
- Se aplican los métodos vistos en clase para la creación de la interfaz de forma dinámica y eficiente.
- Se entiende sobre la creación de clases, atributos y objetos y de cómo interactúan para la creación de la interfaz.
- Manejo de excepciones correctamente para validar un error en una función de forma más sencilla.
- El manejo de entradas no válidas permite una interacción sin interrupciones, guiando al usuario hasta que ingrese una opción correcta.
- El uso del servidor es una herramienta de gran ayuda para realizar solicitudes como inicio y cierre de sesión para que se hagan los pasos correctamente.

REFERENCIAS

[1] M. M. Canelo, “¿Qué es la Programación Orientada a Objetos?”, Profile Software Services, 02-nov-2020. [En línea]. Disponible en: <https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/>. [Consultado: 20-nov-2024].

[2] yacklyon, “CURSO de PYTHON 2020 @ QUE ES POO”. [En línea]. Disponible en: https://www.youtube.com/watch?v=71S1WywB4cY&list=PLg9145ptuAigw5pV_DRznXdOsX19dorDs. [Consultado: 18-nov-2024].

[3] “Interfaces gráficas de usuario con Tk”, Python documentation. [En línea]. Disponible en: <https://docs.python.org/es/3/library/tk.html>. [Consultado: 20-nov-2024].

[4] pildorasinformaticas, “Curso python. Interfaces gráficas I. vídeo 42”. [En línea]. Disponible en: <https://www.youtube.com/watch?v=hTUJC8HsC2I>. [Consultado: 19-nov-2024].

Enlace github:

<https://github.com/AG-Program/Informe-lab-3-.git>



PROGRAMA DE

INGENIERÍA ELÉCTRICA

Acreditado de Alta Calidad

Fec. MEX. 11/05 - 17. Julio 2010 (3 años)