Sistemas Expertos

Laboratorio 2

Milena Rojas Leiva B76703

Documentación Laboratorio 2

En esta documentación primero veremos una descripción del código, luego se mencionarán las decisiones de diseño y por último se ilustrarán los resultados obtenidos.

```
7   import tkinter as tk
8   from tkinter import filedialog
9   import pandas as pd
```

Tkinter lo importamos para hacer la interfaz gráfica de usuario (GUI).

Filedialog nos proporciona las funciones necesarias para abrir y guardad archivos a través de la interfaz de usuario, en este caso se usa para el documento de Excel.

Pandas es una biblioteca de Python que usamos para analizar los datos.

```
11  #Definimos las reglas para dar un diagnostico
12  reglas_diagnostico = {
13      "Antracnosis":{
14          "sintomas_presentes":["manchas_oscuras_hojas_frutos", "pudricion"],
15          "sintomas_ausentes":["manchas_negras_alargadas", "reduccion_foliar",
16                               "marchitez", "necrosis_cogollo", "hojas_jovenes_sin_abrir","pustulas_naranjas",
17                               "manchas_negras_circulares_frutos_hojas", "caida_prematura_hojas", "decoloracion_tallos_raices",
18                               "perdida_vigor","manchas_oscuras_base_fruto", "pudricion_acuosa","manchas_amarillas_marrones_hojas",
19                               "formacion_lesiones","polvo_blanco_hojas", "tallo_deforme", "patron_mosaico_hoja", "deformacion_hoja_fruto"],
20          "diagnostico": "Antracnosis",
21          "explicacion": "Presencia de manchas oscuras en hojas y frutos, en algunos casos se presenta pudrición."
22      },
```

De la línea 12 hasta la línea 126 definimos las reglas de diagnostico donde definimos el nombre de la enfermedad con los síntomas que presenta dicha enfermedad y los síntomas que no tiene. Se da un diagnóstico con el nombre de la enfermedad y una explicación, estos datos son los que se muestran como salida en la interfaz de usuario.

```
128 def evaluar_reglas(sintomas, reglas):
129     diagnosticos= []
130     for enfermedad, regla in reglas.items():
131         if set(regla["sintomas_presentes"]).issubset(sintomas) and not set(regla["sintomas_ausentes"]).intersection(sintomas):
132             diagnosticos.append((enfermedad, regla["diagnostico"], regla["explicacion"]))
133     return diagnosticos
134
```

Definimos una función evaluar_reglas con los parámetros síntomas y reglas.

Creamos un bucle for que itera a través de los elementos del diccionario reglas, cada elemento contiene las reglas asociadas a una enfermedad.

Creamos un if que evalúa si los síntomas presentes y los síntomas ausentes de la regla actual coinciden con los síntomas proporcionados

Si las condiciones del if se cumplen, se agrega el diagnóstico correspondiente a la lista de diagnósticos.

Por último, retornamos la lista de diagnósticos encontrados.

Sistemas Expertos

Laboratorio 2

Milena Rojas Leiva B76703

```python
def cargar_datos_excel():
    filename = filedialog.askopenfilename(title="Seleccionar archivo Excel", filetypes=(("Archivos Excel", "*.xlsx"),("Todos los archivos", "*.*")))

    # Verificar si se seleccionó un archivo
    if filename:
        # Cargar datos desde el archivo Excel
        df = pd.read_excel(filename)

        # Obtener síntomas y valores de las plantas desde el DataFrame
        síntomas = df.columns.tolist()
        valores = df.values.tolist()

        # Evaluar las reglas de diagnóstico para cada planta
        resultados = {}
        for i, planta in enumerate(valores):
            síntomas_planta = [síntoma for j, síntoma in enumerate(síntomas) if planta[j] == "yes"]
            diagnosticos = evaluar_reglas(síntomas_planta, reglas_diagnostico)
            resultados[f"Planta {i+1}"] = diagnosticos

        # Mostrar los diagnósticos en la GUI
        mostrar_diagnosticos(resultados)
```

Definimos una función cargar_datos_excel.

Con “filename” logramos que mediante un cuadro de dialogo se pueda seleccionar el Excel donde están los datos a analizar.

Si el usuario seleccionó un archivo Excel cargamos los datos a un DataFrame, que organiza los datos en filas y columnas para facilitar la interpretación de los mismos.

En el for iteramos sobre las filas de valores del DataFrame, cada fila de valor representa los síntomas presentes en una planta. Se recopilan los síntomas que están presentes, con la función anterior “evaluar reglas” guardamos en un diccionario el número de planta y su diagnóstico.

Por último, llamamos una función mostrar_diagnosticos y le pasamos el diccionario de resultados.

```python
def mostrar_diagnosticos(resultados):
    for widget in root.winfo_children():
        widget.destroy()

    # Crear un frame para contener los diagnósticos
    frame = tk.Frame(root)
    frame.pack(fill="both", expand=True)



    # Crear un widget de texto para mostrar los diagnósticos
    text_widget = tk.Text(frame, wrap="word", font=("Arial", 12))
    text_widget.pack(side="left", fill="both", expand=True)

    # Agregar una barra de desplazamiento vertical
    scrollbar = tk.Scrollbar(frame, orient="vertical", command=text_widget.yview)
    scrollbar.pack(side="right", fill="y")

    # Configurar el widget de texto para que sea desplazable
    text_widget.config(yscrollcommand=scrollbar.set)

    text_widget.insert( index: "end",  chars: "Diagnósticos\n\n",  *args: "center bold")
    for planta, diagnosticos in resultados.items():
        text_widget.insert( index: "end",  chars: f"Planta: {planta}\n",  *args: "bold")
        for diagnostico in diagnosticos:
            text_widget.insert( index: "end",  chars: f"Enfermedad: {diagnostico[1]}\n")
            text_widget.insert( index: "end",  chars: f"Explicación: {diagnostico[2]}\n")
        text_widget.insert( index: "end",  chars: "\n")  # Espacio entre cada conjunto de diagnósticos
```

Sistemas Expertos

Laboratorio 2

Milena Rojas Leiva B76703

Este código proporciona una manera de mostrar los diagnósticos de manera estructurada y, si el contenido excede el tamaño de la ventana, permite desplazarse verticalmente para ver todo el contenido.

```
    # Establecer el estilo para el texto centrado y en negrita
    text_widget.tag_configure("center", justify="center")
    text_widget.tag_configure("bold", font=("Arial", 12, "bold"))



    # Ajustar la altura de la ventana para mostrar todos los resultados
    total_height = min(text_widget.winfo_height() + 20, 500)  # Altura máxima de la ventana
    total_height = max(total_height, 200)  # Altura mínima de la ventana

    # Establecer la geometría del frame y de la ventana
    frame.update_idletasks()  # Actualizar el frame para calcular correctamente su tamaño
    frame.config(width=600, height=total_height)  # Establecer el tamaño del frame

# Crear la ventana principal de la aplicación
root = tk.Tk()
root.title("Diagnóstico de Enfermedades en Plantas")

# Botón para cargar datos desde Excel
cargar_datos_button = tk.Button(root, text="Seleccione el archivo Excel", command=cargar_datos_excel, borderwidth=2, bg="#
cargar_datos_button.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

# Establecer la geometría de la ventana en el centro de la pantalla
ancho_ventana = 800  # Ancho mínimo de la ventana
alto_ventana = 600  # Alto de la ventana
```

En este fragmento se configura el estilo del texto en el widget de texto para que esté en negrita. Luego ajusta dinámicamente la altura de la ventana para mostrar todos los resultados sin necesidad de una barra de desplazamiento, asegurándose de que la ventana tenga un tamaño mínimo y máximo. Finalmente, crea un botón más grande y centrado en la ventana principal para cargar datos desde un archivo Excel.

```
# Obtener dimensiones de la pantalla
ancho_pantalla = root.winfo_screenwidth()
alto_pantalla = root.winfo_screenheight()

# Calcular las coordenadas para centrar la ventana
x = (ancho_pantalla - ancho_ventana) // 2
y = (alto_pantalla - alto_ventana) // 2  # Posicionar la ventana en la parte superior de la pantalla

# Establecer la geometría de la ventana y centrarla
geometry_string = f"{ancho_ventana}x{alto_ventana}+{x}+{y}"
root.geometry(geometry_string)

# Ejecutar el bucle principal de la aplicación
root.mainloop()
```

En este fragmento se calculan las dimensiones de la pantalla y luego determinan las coordenadas para posicionar la ventana en el centro de la pantalla, pero en la parte superior. Finalmente, establecen la geometría de la ventana y ejecutan el bucle principal de la aplicación para que la ventana sea visible y pueda interactuar con ella.

Sistemas Expertos

Laboratorio 2

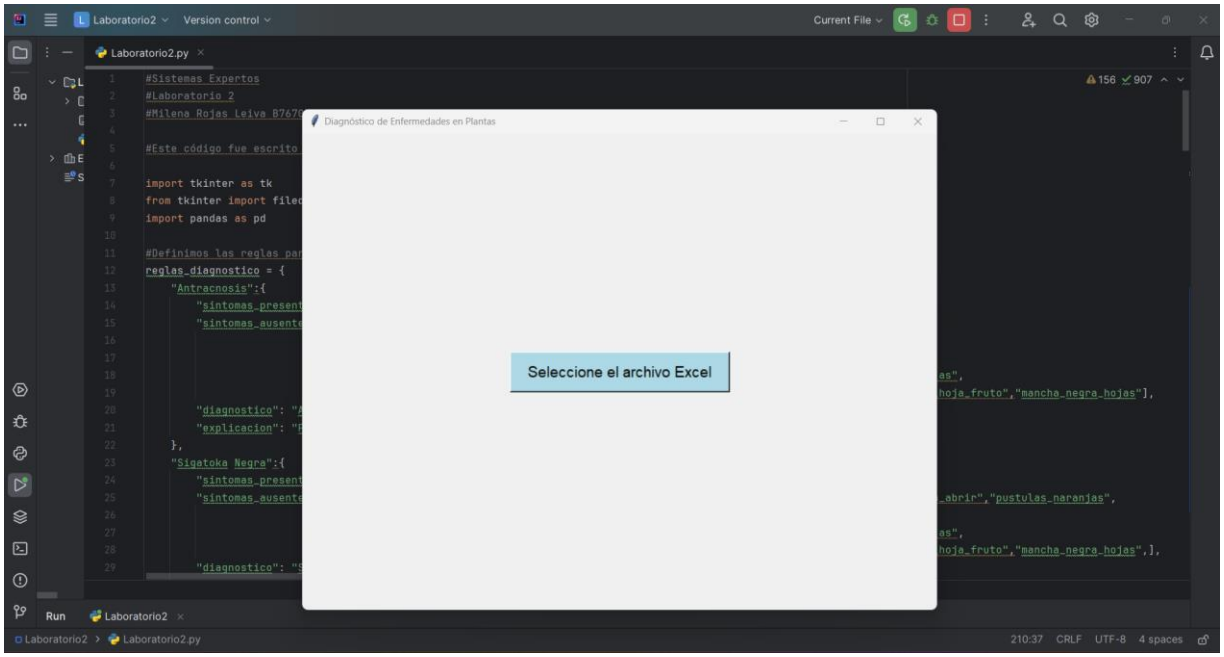
Milena Rojas Leiva B76703

Decisiones de Diseño de la GUI

Cuando se ejecuta el programa en la ventana principal podemos ver un botón de forma distintiva, que nos lleva a escoger el documento Excel donde tenemos los datos. Luego de seleccionar el archivo en la misma ventana se muestran los resultados donde se enumeran cada fila como “Planta 1” y sucesivamente cambia el número, el diagnóstico con el nombre de la enfermedad y su debida explicación. Se decidió poner una barra lateral por si el programa se llega a ejecutar con un documento Excel que tenga muchos datos.

El documento Excel tiene en la primera fila los síntomas de las enfermedades, en las filas siguientes se pone “yes” o “no” dependiendo la enfermedad que se esta evaluando. Para el Excel que se llama “Datos Plantas” el orden de las enfermedades debe ser el siguiente: Royal Café, Mancha Angular, Antracnosis, Oídio, Virosis, Mancha Negra, Pudrición Cogollo, Sigatoka Negra, Mancha de Asfalto, Marchitez por Fusarium, Podredumbre Negra.

Resultados



Sistemas Expertos

Laboratorio 2

Milena Rojas Leiva B76703