Manual del Programador

Proyecto: Traductor Bidireccional Baile-español

Versión 1

Equipo: KDR CONSULTECH

Fecha: 5 de junio de 2024

MANUAL DEL PROGRAMADOR

Tabla de contenido

Descripción del proyecto

Este programa de traducción de Braille permite convertir texto en español en caracteres Braille y genera archivos PDF e imágenes en modo espejo con la respectiva traducción en Braille. Además, el proyecto utiliza diccionarios para mapear los caracteres alfabéticos y numéricos junto a sus correspondientes símbolos en braille.

Estructura del código

El código se organiza en varios archivos y módulos que trabajan en conjunto para la traducción y generación de archivos PDF y JPG.

- T2B_code.py: aquí se maneja la lógica de traducción de texto a Braille y se define un diccionario llamado 'mapeo_braille' para mapear caracteres individuales y números a su representación en braille.
- about_form_Design.py: este módulo define una ventana de información "about" para mostrar la versión del software y el autor.
- T2B_form_design.py: contiene métodos para crear la interfaz de usuario y gestionar la traducción de texto a Braille y la generación de archivos e imágenes.
- Main_form_design.py: define la Ventana principal de la aplicación, incluyendo la barra superior, barra lateral y el panel de contenido principal, además, de permitir la navegación entre los diferentes formularios de la aplicación.
- Building_form_design.py: se define un formulario en construcción simple que muestra cuando una funcionalidad de la aplicación esta o se encuentra en desarrollo.
- convertTo.py: este módulo proporciona la funcionalidad para convertir texto en Braille en diferentes formatos: PDF utilizando Tkinter para la interfaz gráfica y ReportLab para la generación de PDF.
- BrailleApp_KDR.py: es la entrada principal del programa, inicializa la aplicación y ejecuta el ciclo principal de eventos de la interfaz gráfica.

Requisitos del entorno

- Python en la versión 3.11 o superior
- Tkinter, biblioteca necesaria para la creación de interfaces graficas de usuario
- Reportlab: biblioteca necesaria para crear documentos PDFs.
- Pillow(PIL): biblioteca para abrir, guardar y manipular imágenes en variedad de formato, PNG,JPGE, JPG, entre otros.

Instalaciones necesarias

- Clonar el repositorio del proyecto desde gitHub
- Instalar las dependencias necesarias utilizando pip:
    - pip install reportlab
    - pip install pillow
    - pip install customtkinter

- Ejecuta la aplicación:

    BrailleApp_KDR.py

Detalle de métodos

obtener_texto():
Solicita al usuario que ingrese un texto, lo divide en palabras y clasifica cada palabra como numérica o alfabética.

Returns:Una lista de tuplas donde cada tupla contiene una palabra y un indicador de tipo (0 para números, 1 para caracteres).

```python
def obtener_texto():
    texto = input("Ingresa un texto: ")

    # Dividir la cadena en palabras
    palabras = texto.split()

    # Imprimir la lista de palabras
    print("Lista de palabras:")
    print(palabras)

    resultados = [] # 0 para numeros, 1 para caracteres
    for palabra in palabras:
        if palabra.isdigit():
            resultados.append((palabra, 0))
        else:
            resultados.append((palabra, 1))

    print(resultados)
    return resultados
```

texto_a_braille(resultados):
Convierte una lista de palabras clasificadas en su equivalente en braille.

Args: Lista de tuplas donde cada tupla contiene una palabra y un indicador.
Returns: Una cadena de texto convertida a braille.

```
def texto_a_braille(resultados):
  braille = ''
  for palabra, tipo in resultados:
    if tipo == 0:
      braille += ' ⠄'


    for caracter in palabra:
      if caracter.lower() in mapeo_braille:
        if caracter.isupper():
          braille += ' ⠠'
        braille += mapeo_braille[caracter.lower()]
      else:
        braille += ' '

  return braille
```

generar_pdf(texto_braille, nombre_archivo):
Genera un archivo PDF con el texto en braille.

Args:
Texto convertido a braille.
Nombre del archivo PDF a generar.

```
def generar_pdf(texto_braille, nombre_archivo):
    ruta_fuente = "Devs/Fonts/ONCE_CBE_6.ttf"
    pdfmetrics.registerFont(TTFont('Braille',
ruta_fuente))
    ancho, alto = letter

    # Inicializar el lienzo (canvas)
    c = canvas.Canvas("Devs/Results/" + nombre_archivo,
pagesize=letter)

    # Configurar la fuente personalizada
    c.setFont("Braille", 20)  # Reemplaza 12 con el tamaño
de fuente deseado

    # Dividir el texto en líneas para que quepa en la
página
    lineas = [texto_braille[i:i+35] for i in range(0,
len(texto_braille), 35)]
```

```
    # Definir la posición inicial para escribir el texto
    x_inicial = 30
    y_inicial = alto - 50

    # Escribir el texto en el lienzo
    for linea in lineas:
      c.drawString(x_inicial, y_inicial, linea)
      y_inicial -= 20 * 2   # Ajustar la posición vertical
para la siguiente línea

    # Guardar el PDF
    c.save()
```

generar_pdf_espejo(texto_braille, nombre_archivo):
Genera un archivo PDF con el texto en braille en modo espejo.

Args:
Texto convertido a braille.
Nombre del archivo PDF a generar.

```
def generar_pdf_espejo(texto_braille, nombre_archivo):
    # Cargar la fuente personalizada
    ruta_fuente = "Devs/Fonts/ONCE_CBE_6.ttf"
    pdfmetrics.registerFont(TTFont('Braille', ruta_fuente))

    # Definir el tamaño de la página
    ancho, alto = letter

    # Inicializar el lienzo (canvas)
    c = canvas.Canvas("Devs/Results/" + nombre_archivo,
pagesize=letter)

    # Configurar la fuente personalizada
    c.setFont("Braille", 20)

    # Configurar la impresión en modo espejo horizontalmente
    c.scale(-1, 1)  # Invertir horizontalmente
```

```
    # Dividir el texto en líneas para que quepa en la página
    lineas = [texto_braille[i:i+35] for i in range(0,
len(texto_braille), 35)]

    # Definir la posición inicial para escribir el texto
    x_inicial = -ancho + 50
    y_inicial = alto - 50

    # Escribir el texto en el lienzo
    for linea in lineas:
      c.drawString(x_inicial, y_inicial, linea)
      y_inicial -= 20 * 2   # Ajustar la posición vertical
para la siguiente línea

    # Guardar el PDF
    c.save()
```

main():
Función principal que maneja la obtención de texto, conversión a braille y generación de los PDFs.

```
def main():
  # Leer el archivo de texto
  resultados = obtener_texto()

  # Texto a braile
  texto_braille = texto_a_braille(resultados)
  print(texto_braille)

  # Generar PDF
  generar_pdf(texto_braille, "texto_braille.pdf")

  # Generar PDF Espejo
  generar_pdf_espejo(texto_braille,
"texto_braille_espejo.pdf")

if __name__ == "__main__":
   main()
```

Clase ConvertTo
Esta clase maneja la conversión del texto en Braille a un archivo PDF y ofrece opciones para guardarlo en modo normal o espejo.

__init__(self)

Inicializa la clase, obtiene el texto Braille final y la ruta a la fuente Braille, y registra la fuente con ReportLab.

```
def __init__(self):
        print("convert")
        self.raw_braille = T2BCode.get_final_braille()
        self.ruta_fuente = UtilPath().get_font_path()
        self.set_font()
```

generar_pdf_espejo(self)
Genera un archivo PDF con el texto en Braille en modo espejo.

Salida: Archivo PDF guardado en la ruta especificada por el usuario.

```
def generar_pdf_espejo(self):

        archivo = self.get_save_name()

        # Crear un objeto canvas para el documento PDF
        if archivo:
            nombre = archivo

            # Definir el tamaño de la página
            ancho, alto = letter

            # Inicializar el lienzo (canvas)
            c = canvas.Canvas(nombre, pagesize=letter)

            # Configurar la fuente personalizada
            c.setFont("Braille", 20)

            # Configurar la impresión en modo espejo horizontalmente
            c.scale(-1, 1)  # Invertir horizontalmente

            # Dividir el texto en líneas para que quepa en la página
            lineas = [self.raw_braille[i:i+35] for i in range(0, len(self.raw_braille), 35)]

            # Definir la posición inicial para escribir el texto
            x_inicial = -ancho + 50
            y_inicial = alto - 50

            # Escribir el texto en el lienzo
            for linea in lineas:
                c.drawString(x_inicial, y_inicial, linea)
                y_inicial -= 20 * 2   # Ajustar la posición vertical para la siguiente línea

            # Guardar el PDF
            c.save()
```

```
        if c:
            self.succesful_save()
```

convert_2_image(self)
Convierte el texto en braille a una imagen y la almacena en el sistema de archivos

Salida: Archivo de imagen almacena en una ruta especificada por el usuario.

```
    def convert_2_image(self):
        BACKGROUND_COLOR = "white"
        text_color = "black"

        # Filediallog para preguntar ruta para guardar la imagen
        archivo = self.get_save_name('1')

        if archivo:
            nombre = archivo
            ancho, alto = letter

            # Crear nueva imagen con el tama;o especificado y el color de fondo
            imagen = Image.new("RGB", (int(ancho), int(alto)), BACKGROUND_COLOR)

            # Crear un objeto ImageDraw para dibujar en la imagen
            draw = ImageDraw.Draw(imagen)

            # Definir la fuente y el tamaño del texto
            if self.ruta_fuente:
                fuente = ImageFont.truetype(self.ruta_fuente, 20)

                # Dividir el texto en líneas para que quepa en la imagen
                lineas = [self.raw_braille[i:i+35] for i in range(0, len(self.raw_braille), 35)]

                # Definir la posición inicial para escribir el texto
                x_inicial = 30
                y_inicial = 50

                # Escribir el texto en la imagen
                for linea in lineas:
                    draw.text((x_inicial, y_inicial), linea, fill = text_color, font = fuente)
                    y_inicial += 20 * 2

                # Guardar imagen
                imagen.save(nombre)

                if imagen:
                    self.succesful_save()
```

set_font(self)
Registra una fuente TrueType con la librería ReportLab para usarla en los documentos PDF.

```
def set_font(self):
    # Registrar un TrueType font con la libreria ReportLab de pdfmetrics
    pdfmetrics.registerFont(TTFont('Braille', self.ruta_fuente))
```

get_save_name(self)
Abre un diálogo para que el usuario elija la ubicación y el nombre del archivo donde se guardará el PDF.

Retorno: Cadena con la ruta y el nombre del archivo seleccionado por el usuario.

```
  def get_save_name(self):
    return filedialog.asksaveasfilename(
        defaultextension = ".*", title = "Save File", filetypes = (("PDF Files", "*.pdf"), ("Text Files", "*.txt"))
    )
```

succesful_save(self)
Muestra un mensaje de éxito cuando el archivo PDF se guarda correctamente.

```
  def succesful_save(self):
    messagebox.showinfo("Success", "Archivo guardado con exito")
```

T2B_code.py

__init__(self, text)
Inicializa la clase, recibe el texto que será convertido y llama al método texto_a_braile para iniciar la conversión.

```
def __init__(self, text):
    self.new_text = text
    self.texto_a_braile(self.new_text)
```

obtener_texto(self, texto)
Divide el texto en palabras y clasifica cada palabra como numérica o alfabética.

Entrada: texto o cadena de texto que será procesada.
Salida: Una lista de tuplas, donde cada tupla contiene una palabra y un identificador de tipo (0 para números, 1 para caracteres alfabéticos).

```
  def obtener_texto(self, texto):
    # Dividir la cadena en palabras
    palabras = texto.split()
```

```
    # Imprimir la lista de palabras
    print("Lista de palabras:")
    print(palabras)

    resultados = [] # 0 para numeros, 1 para caracteres
    for palabra in palabras:
        if palabra.isdigit():
            resultados.append((palabra, 0))
        else:
            resultados.append((palabra, 1))

    print(resultados)
    return resultados
```

texto_a_braile(self, texto)
Convierte el texto a su representación en Braille utilizando un diccionario de mapeo.

Entrada: texto o una cadena de texto que será convertida a Braille.

Aquí se divide el texto en palabras, para poder identificar el tipo de cada palabra, se mapea cada carácter de la palabra a su representación en Braille, se agregan el símbolo de mayúscula si el carácter corresponde a una letra mayúscula, se añade el espacio entre palabras en la representación en Braille.

```
def texto_a_braile(self, texto):
    palabras_array = self.obtener_texto(texto)

    global braille
    braille = '  '

    for palabra, tipo in palabras_array:
        if tipo == 0:
            braille += ' ⠼'

        for caracter in palabra:
            if caracter.lower() in mapeo_braille:  # Si el caracter está en el diccionario, añadir su representación en braille
                if caracter.isupper():  # Si es mayúscula, añadir el símbolo de mayúscula en braille
                    braille += ' ⠠'
                braille += mapeo_braille[caracter.lower()]  # Convertimos el caracter a braille y lo añadimos al resultado
            else:
                braille += ' '  # Si el caracter no está en el diccionario, lo reemplazamos con un espacio en blanco
        braille += '  ' # Añadir espacio entre palabras en braille
        print(braille)
```

get_final_braille()
Devuelve el texto final en Braille después de la conversión.

Salida: Cadena de texto en Braille.

```
def get_final_braille():
    return braille
```

about_form_Design.py

__init__(self) -> None
Inicializa la ventana llamando a los métodos config_window y create_widgets para configurar la ventana y crear los widgets necesarios.

```
def __init__(self) -> None:
    super().__init__()
    self.config_window()
    self.create_widgets()
```

config_window(self)
Configura la ventana principal, estableciendo el título y centrando la ventana en la pantalla.
Se establece el título de la ventana a "About", define el tamaño de la ventana (w, h), y posteriormente, llama a la función centrar_ventana del módulo util_vent para centrar la ventana.

```
def config_window(self):
    # configuracion incial de la ventana
    self.title("About")
    w, h = 400, 100
    util_vent.centrar_ventana(self, w, h)
```

create_widgets(self)
Crea y configura los widgets dentro de la ventana.

Primero, crea un Label para mostrar la versión del software, configura las propiedades del Label (color de texto, fuente, padding, ancho) y empaqueta el Label en la ventana. Después, crea otro Label para mostrar el autor del software y configura las propiedades del Label (color de texto, fuente, padding, ancho) para volver a empaqueta el Label en la ventana.

```
def create_widgets(self):
    # configuracion de los widgets
    self.labelVersion = tk.Label(
```

```
            self, text = "Version : 1.0"
        )
        self.labelVersion.config(
            fg = "BLACK", font = (
                "Roboto", 15
            ),
            pady = 10, width = 20
        )
        self.labelVersion.pack()

        self.labelAutor = tk.Label(
            self, text = "Autor : KDR CONSULTECH"
        )
        self.labelAutor.config(
            fg = "#000000", font = (
                "Roboto", 15
            ),
            pady = 10, width = 30
        )
        self.labelAutor.pack()
```

T2B_form_design.py

__init__(self, main_panel)
Inicializa la interfaz gráfica llamando a los métodos create_frames, create_top_widgets, create_center_widgets y create_bottom_widgets para configurar la ventana y crear los widgets necesarios.

```
def __init__(self, main_panel):
        self.create_frames(main_panel)
        self.create_top_widgets()
        self.create_center_widgets("")
        self.create_bottom_widgets()
```

create_frames(self, main_panel)
Crea y configura los marcos (frames) principales de la ventana.

Parámetros:
main_panel: El panel principal en el que se colocarán los marcos.

```
def create_frames(self, main_panel):
        self.top_frame = tk.Frame(main_panel, bg=COLOR_FONDO, height = 50)
        self.top_frame.pack(side = tk.TOP, fill = 'both', expand = True)

        self.center_frame = tk.Frame(main_panel, bg=COLOR_FONDO, height = 50)
        self.center_frame.pack(side = tk.TOP, fill = 'both', expand = True)

        self.bottom_frame = tk.Frame(main_panel, bg=COLOR_FONDO, height = 30)
        self.bottom_frame.pack(side = tk.TOP, fill = 'both', expand = False)
```

create_top_widgets(self)
Crea y configura los widgets en el marco superior, incluyendo el cuadro de texto y los botones de traducción y limpieza.

```
def create_top_widgets(self):
        ancho = 20
        alto = 1
        font_awesome = font.Font(family = 'FontAwesome', size = 12)

        # Crear Entry widget
        self.textbox = ctk.CTkTextbox(
            self.top_frame, font=("Arial", 13), fg_color="GREY", text_color="BLACK",
height = 100
        )
        self.textbox.pack(padx=100, pady=20, side = tk.TOP, fill='both', expand=True)

        # Crear botones
        self.button_traslate = tk.Button(self.top_frame)
        self.button_clear_box = tk.Button(self.top_frame)

        buttons_info =[
            ("TRADUCIR", self.button_traslate, "\uf04b", self.trad_2_braille),
            ("LIMPIAR", self.button_clear_box, "\uf00d", self.clear_textbox)
        ]

        for text, button, icon, cm in buttons_info:
            self.top_buttons_config(button, text, icon, font_awesome, ancho, alto, cm)
```

top_buttons_config(self, button, text, icon, font_awesome, ancho, alto, cm)
Configura los botones superiores con texto, icono, fuente, tamaño y comando especificados.

```
  def top_buttons_config(self, button, text, icon, font_awesome, ancho, alto, cm):
        button.config(
            text = f"  {icon}\t{text}", anchor = "w", font = font_awesome, bd = 0, bg =
COLOR_BOTONES, fg = "WHITE", width = ancho, height = alto, command = cm
        )
        button.pack(padx = 10, pady = 5, side = tk.RIGHT)
        self.bind_hover_events(button)
```

create_center_widgets(self, text)
Crea y configura los widgets en el marco central, incluyendo una etiqueta para mostrar el texto traducido.

Parámetros:
text: El texto a mostrar en la etiqueta central.

```
def create_center_widgets(self, text):
        self.text_label = ctk.CTkLabel(self.center_frame, font=("Arial",
13),fg_color="GREY", text = text, text_color = "BLACK", corner_radius = 7, height =
100, anchor = 'w', wraplength = 550)
        self.text_label.pack(padx=100, pady=20, side = tk.TOP, fill='both', expand=True)
```

create_bottom_widgets(self)
Crea y configura los widgets en el marco inferior, incluyendo los botones para generar archivos PDF.

```
def create_bottom_widgets(self):
        # Crear botones
        self.button_normal = tk.Button(self.bottom_frame)
        self.button_espejo = tk.Button(self.bottom_frame)

        buttons_info =[
            ("PDF - NORMAL", self.button_normal, "\uf04b", self.to_pdf_normal),
            ("PDF - ESPEJO", self.button_espejo, "\uf00d", self.to_pdf_espejo)
        ]

        for text, button, icon, cm in buttons_info:
            ancho = 20
            alto = 1
            font_awesome = font.Font(family = 'FontAwesome', size = 11)

            self.top_buttons_config(button, text, icon, font_awesome, ancho, alto, cm)
```

Funcionalidades Principales

trad_2_braille(self)
Obtiene el texto del cuadro de texto, lo traduce a braille y actualiza la etiqueta central con el texto traducido.

```
def trad_2_braille(self):
        print("Traducir")

        new_text = self.get_text()
        print(new_text)

        T2BCode(new_text)
        final_text = T2BCode.get_final_braille()

        self.clear_panel(self.center_frame)
        self.create_center_widgets(final_text)
```

clear_textbox(self)
Limpia el cuadro de texto y la etiqueta central.

```
def clear_textbox(self):
        print("Limpiar")
        self.textbox.delete(0.0, 'end')
        self.clear_panel(self.center_frame)
        self.create_center_widgets("")
```

get_text(self)
Obtiene el texto del cuadro de texto.

clear_panel(self, panel)
Elimina todos los widgets de un panel especificado.

```
def get_text(self):
        return self.textbox.get(0.0, 'end')

    def get_text_braille(self, text):
        return self.trad_2_braille()

    def clear_panel(self, panel):
        for widget in panel.winfo_children():
            widget.destroy()
```

to_pdf_espejo(self)
Genera un archivo PDF en modo espejo con el texto traducido a braille.

```
    def to_pdf_espejo(self):
        print("PDF ESPEJO")
        Convert2PDF().generar_pdf_espejo()
```

Main_from_design.py

__init__(self)
Inicializa la ventana principal llamando a los métodos config_window, paneles, barra_sup_ctrl y menu_lat_ctrl para configurar la ventana y crear los componentes necesarios.

```
def __init__(self):
        super().__init__()
        self.config_window()
        self.paneles()
        self.barra_sup_ctrl()
        self.menu_lat_ctrl()
```

config_window(self)

Configura la ventana principal con un título y dimensiones específicas.

```
def config_window(self):
        #Configuracion inicial de la ventana
        self.title('Braille Traslate')
        w, h = 1024, 600
        util_vent.centrar_ventana(self, w, h)
```

paneles(self)
Crea y configura los frames principales de la ventana, incluyendo la barra superior, el menú lateral y el panel de contenido.

```
def paneles(self):
        self.barra_sup = tk.Frame(
            self, bg = COLOR_BARRA_SUP, height = 50
        )
        self.barra_sup.pack(side = tk.TOP, fill = 'both')

        self.menu_lat = tk.Frame(
            self, bg = COLOR_MENU_LAT, width = 150
        )
        self.menu_lat.pack(side = tk.LEFT, fill = 'both', expand = False)

        self.fondo = tk.Frame(
            self, bg = COLOR_FONDO
        )
        self.fondo.pack(side = tk.RIGHT, fill = 'both', expand = True)
```

barra_sup_ctrl(self)
Configura la barra superior con un título, un botón para mostrar/ocultar el menú lateral y una etiqueta de información.

```
def barra_sup_ctrl(self):
        # Configuracion de la barra superior
        font_awesome = font.Font(family = 'FontAwesome', size = 12)

        # Etiqueta de titulo
        self.labelTitulo = tk.Label(
            self.barra_sup, text = "Braille Traslate"
        )
        self.labelTitulo.config(fg = "#fff", font = (
            "Roboto", 15),
            bg = COLOR_BARRA_SUP, pady = 10, width = 20
        )
        self.labelTitulo.pack(side = tk.LEFT)

        # Boton del menu principal
        self.menuButton = tk.Button(
            self.barra_sup, text = "\uf0c9", font = font_awesome,
```

```
            command = self.toggle_panel, bd = 0, bg = COLOR_BARRA_SUP,
            fg = "WHITE"
        )
        self.menuButton.pack(side = tk.LEFT)

        # Etiqueta de informacion
        self.labelTitulo = tk.Label(
            self.barra_sup, text = "VERSION DE PRUEBA"
        )
        self.labelTitulo.config(fg = "#fff", font = (
            "Roboto", 10),
            bg = COLOR_BARRA_SUP, padx = 10, width = 20
        )
        self.labelTitulo.pack(side = tk.RIGHT)
```

menu_lat_ctrl(self)
Configura el menú lateral con botones para acceder a diferentes formularios de la aplicación.

```
def menu_lat_ctrl(self):
        # Configuracion de la barra lateral

        ancho_menu = 20
        alto_menu = 2
        font_awesome = font.Font(family = "FontAwesome", size = 15)

        # Botones del menu lateral

        self.T2BButton = tk.Button(self.menu_lat)
        self.B2TButton = tk.Button(self.menu_lat)
        self.aboutButton = tk.Button(self.menu_lat)

        buttons_info = [
            ("T2B", self.T2BButton, "\uf2a1", self.open_T2B),
            ("B2T", self.B2TButton, "\uf031", self.open_build),
            ("About", self.aboutButton, "\uf05a", self.open_about)
        ]

        for text, button, icon, cm in buttons_info:
            self.menu_button_conf(button, text, icon, font_awesome, ancho_menu,
alto_menu, cm)
```

menu_button_conf(self, button, text, icon, font_awesome, ancho, alto, cm)
Configura los botones del menú lateral con texto, icono, fuente, tamaño y comando especificados.

```
def menu_button_conf(self, button, text, icon, font_awesome, ancho, alto, cm):
        button.config(
```

```
            text = f"  {icon}\t{text}", anchor = "w", font = font_awesome,
            bd = 0, bg = COLOR_MENU_LAT, fg = "WHITE", width = ancho, height =
alto,
            command = cm
        )
        button.pack(side = tk.TOP)
        self.bind_hover_events(button)
```

toggle_panel(self)
Muestra u oculta el menú lateral.

```
def toggle_panel(self):
        # Cambiar el estado del panel lateral
        if self.menu_lat.winfo_ismapped():
            self.menu_lat.pack_forget()
        else:
            self.menu_lat.pack(side = tk.LEFT, fill = tk.Y)
```

Funcionalidades Principales

open_about(self)
Abre el formulario de información "About".

open_build(self)
Limpia el panel de contenido y carga el formulario "Building".

open_T2B(self)
Limpia el panel de contenido y carga el formulario "T2B".

clear_panel(self, panel)
Elimina todos los widgets de un panel especificado.

```
def open_about(self):
        AboutFormDesign()

    def open_build(self):
        self.clear_panel(self.fondo)
        BuildingFormDesign(self.fondo)

    def open_T2B(self):
        self.clear_panel(self.fondo)
        T2BFormDesign(self.fondo)

    def clear_panel(self, panel):
        for widget in panel.winfo_children():
            widget.destroy()
```

Dependencias

Util.util_ventana: Para centrar la ventana en la pantalla.
*Util.config_color*s: Para los colores de la interfaz.
Forms.about_form_design: Para el formulario de información "About".
Forms.building_form_design: Para el formulario "Building".
Forms.T2B_form_design: Para el formulario de traducción de texto a braille "T2B".

Define los colores utilizados en la interfaz gráfica con Util.config_colors:

```
# Configuración de colores de los elementos de la GUI
COLOR_BARRA_SUP = "DARKBLUE"
COLOR_MENU_LAT = "LIGHTBLUE"
COLOR_FONDO = "LIGHTGRAY"
COLOR_CURSOR_MENU = "GRAY"
COLOR_BOTONES = "DARKBLUE"
```

Building_from_design.py

__init__(self, main_panel)
Inicializa el formulario de construcción creando los marcos (frames) superior e inferior, y configurando una etiqueta con el mensaje de "EN CONSTRUCCIÓN".

```
def __init__(self, main_panel):
        self.top_bar = tk.Frame(main_panel)
        self.top_bar.pack(side=tk.TOP, fill=tk.X, expand = False)

        self.bottom_bar = tk.Frame(main_panel)
        self.bottom_bar.pack(side=tk.BOTTOM, fill='both', expand = True)

        self.labelTitulo = tk.Label(
            self.top_bar, text = "EN CONSTRUCCION"
        )
        self.labelTitulo.config(
            fg = "DARKRED", font = (
                "Roboto", 30
            ),
            bg = COLOR_FONDO
        )
        self.labelTitulo.pack(side=tk.TOP, fill='both', expand = True)
```

Dependencias

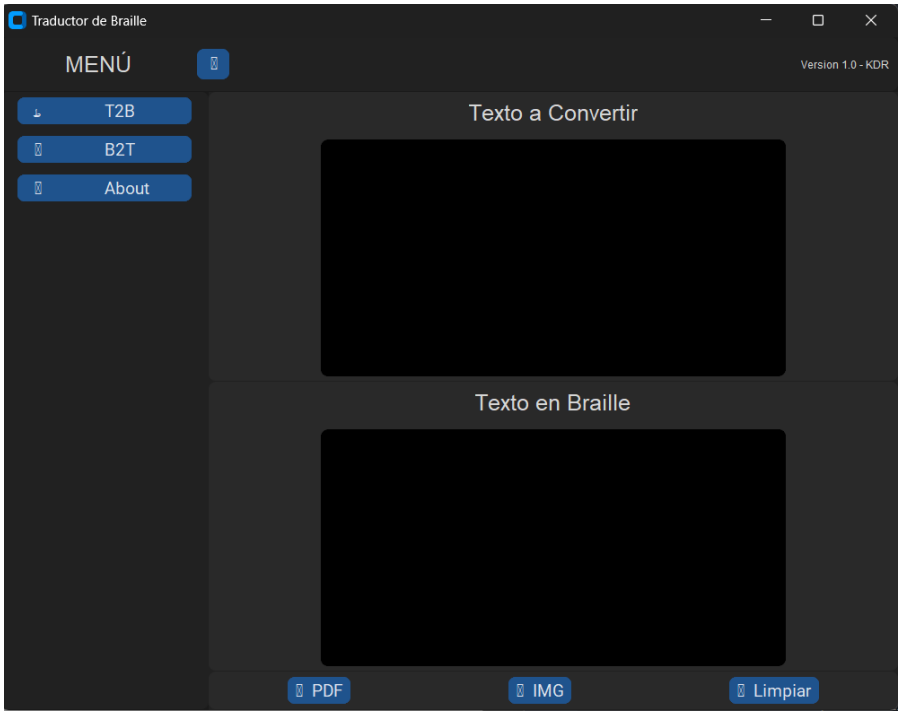
tkinter: Para la creación de la interfaz gráfica de usuario.
Util.config_colors: Para los colores de la interfaz, en este caso COLOR_FONDO para el fondo de la etiqueta.

Ejemplo de ejecución

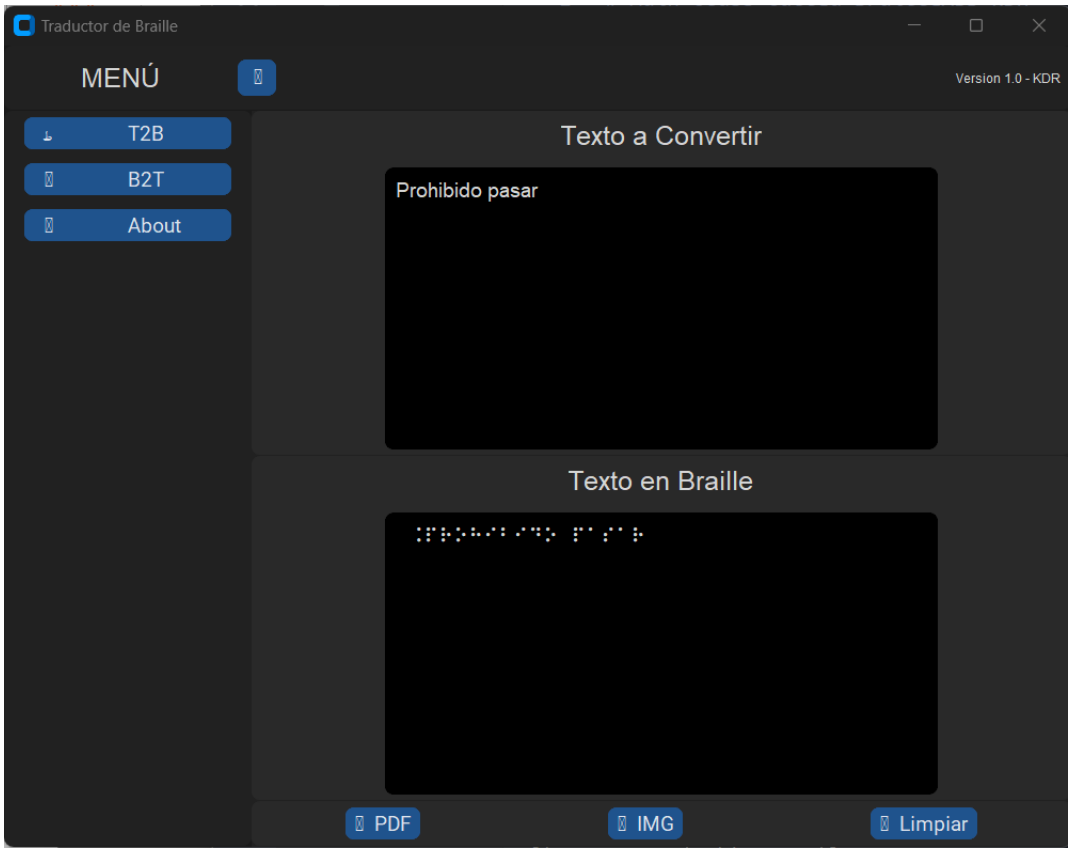
1. Iniciar el programa

BrailleApp_KDR.py

2. Se desplegará la siguiente ventana



3. Ingrese el texto a traducir en el campo correspondiente, la traducción a braille se generará de manera automática, la cual podrá observase en el recuadro inferior.



4. Una vez realizada la traducción, se podrá generar los documentos correspondientes en el formato deseado.