```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
import fitz  # PyMuPDF
import json
import os


class PDFViewerApp:
  def __init__(self, master):
      self.master = master
      self.master.title("PDF Analisador v0.2")
      self.zoom = 1.0


      # === NOVA ESTRUTURA DE LAYOUT ===

      # Frame superior para botões
      self.button_frame = tk.Frame(master)
      self.button_frame.pack(side="top", fill="x", pady=5)

      # Botões de controle
      tk.Button(self.button_frame, text="+ Zoom", command=self.zoom_in).pack(side="left", padx=5)
      tk.Button(self.button_frame, text="- Zoom", command=self.zoom_out).pack(side="left", padx=5)
      tk.Button(self.button_frame, text="Salvar seleção", command=self.save_selection).pack(side="left", padx=5)
      tk.Button(self.button_frame, text="Aplicar censura", command=self.apply_redaction).pack(side="left", padx=5)
      tk.Button(self.button_frame, text="Censurar ocorrências de...", command=self.censor_all_occurrences_dialog).pack(side="left", padx=5)


      # Frame central que conterá o canvas e scrollbar
      self.central_frame = tk.Frame(master)
      self.central_frame.pack(fill="both", expand=True)
```

```
# Canvas principal - agora preenche todo o espaço disponível
self.canvas = tk.Canvas(self.central_frame, bg="gray85")  # Cor de fundo para visualizar a área
self.canvas.pack(side="left", fill="both", expand=True)

# Scrollbar vertical
self.v_scrollbar = tk.Scrollbar(self.central_frame, orient="vertical", command=self.canvas.yview)
self.v_scrollbar.pack(side="right", fill="y")

# Scrollbar horizontal (útil para PDFs largos)
self.h_scrollbar = tk.Scrollbar(master, orient="horizontal", command=self.canvas.xview)
self.h_scrollbar.pack(side="bottom", fill="x")

# Configurar canvas com ambas scrollbars
self.canvas.configure(
    yscrollcommand=self.v_scrollbar.set,
    xscrollcommand=self.h_scrollbar.set
)


# Frame que conterá as páginas do PDF
self.pdf_container = tk.Frame(self.canvas, bg="gray85")

# Lista de seleção (abaixo da scrollbar horizontal)
self.selection_frame = tk.Frame(master)
self.selection_frame.pack(side="bottom", fill="x", before=self.h_scrollbar)
self.selection_label = tk.Label(self.selection_frame, text="Textos Selecionados:")
self.selection_label.pack(anchor="w")
self.selection_listbox = tk.Listbox(self.selection_frame, height=5)
self.selection_listbox.pack(fill="x", padx=5, pady=5)


# Variáveis de estado
```

```
    self.pdf_doc = None
    self.pdf_path = None
    self.page_images = []
    self.bbox_list = []
    self.bbox_rects = []
    self.page_canvases = []
    self.selected_bboxes = []


    # Configurar redimensionamento
    self.canvas.bind("<Configure>", self.on_canvas_resize)

    # NOVO: Configurar navegação com scroll e teclado
    self.configure_smooth_scrolling()

    # NOVO: Dar foco ao canvas para receber eventos de teclado
    self.canvas.focus_set()

    self.setup_menu()


def on_canvas_resize(self, event):
    """Reposiciona o PDF quando a janela é redimensionada"""
    if hasattr(self, 'canvas_window_id'):
        self.center_pdf_container()


def center_pdf_container(self):
    """Centraliza o container do PDF no canvas"""
    # Atualiza a região de scroll
    self.canvas.configure(scrollregion=self.canvas.bbox("all"))

    # Obtém as dimensões
```

```
    canvas_width = self.canvas.winfo_width()
    canvas_height = self.canvas.winfo_height()
    container_width = self.pdf_container.winfo_reqwidth()
    container_height = self.pdf_container.winfo_reqheight()

    # Calcula a posição centralizada
    # Se o conteúdo for menor que o canvas, centraliza
    # Se for maior, alinha no topo/esquerda (0,0)
    x = max(0, (canvas_width - container_width) // 2)
    y = 0  # Sempre começar do topo para facilitar a leitura

    # Move o container para a posição centralizada
    self.canvas.coords(self.canvas_window_id, x, y)


def setup_scrolling(self):
    """
    Configura o scroll do mouse wheel para funcionar em qualquer lugar do PDF.

    Esta função resolve o problema do Tkinter onde eventos de scroll não são
    propagados automaticamente entre widgets. Vinculamos o mesmo handler a
    todos os widgets relevantes.
    """

    def on_mousewheel(event):
        """
        Processa eventos de scroll do mouse.

        O comportamento varia entre sistemas operacionais:
        - Windows: usa event.delta (múltiplos de 120)
        - Linux/Mac: usa event.num (4 para cima, 5 para baixo)
        """
        if event.delta:
```

```
            # Windows: delta negativo = scroll para baixo
            # Dividimos por 120 porque o Windows envia deltas em múltiplos de 120
            scroll_amount = -1 * (event.delta / 120)
        else:
            # Linux/Mac: num 4 = scroll para cima, num 5 = scroll para baixo
            scroll_amount = -1 if event.num == 4 else 1

        # Move o canvas verticalmente
        # O valor 2 controla a velocidade do scroll (ajuste conforme necessário)
        # "units" significa que estamos scrollando em unidades definidas pelo widget
        self.canvas.yview_scroll(int(scroll_amount * 2), "units")

        # Retorna "break" para impedir que o evento se propague para outros widgets
        # Isso evita comportamento errático de scroll duplo
        return "break"

    # Vincula o evento de scroll ao canvas principal
    self.canvas.bind("<MouseWheel>", on_mousewheel)  # Windows
    self.canvas.bind("<Button-4>", on_mousewheel)    # Linux/Mac scroll up
    self.canvas.bind("<Button-5>", on_mousewheel)    # Linux/Mac scroll down

    # IMPORTANTE: Vincula também ao container do PDF
    # Sem isso, o scroll só funcionaria quando o mouse está sobre áreas vazias
    self.pdf_container.bind("<MouseWheel>", on_mousewheel)
    self.pdf_container.bind("<Button-4>", on_mousewheel)
    self.pdf_container.bind("<Button-5>", on_mousewheel)


def bind_scroll_to_page(self, widget):
    """
    Vincula eventos de scroll a um widget específico (página ou canvas).

    Esta função é crucial porque cada página do PDF é um widget separado,
```

```
    e precisamos garantir que o scroll funcione quando o mouse está sobre
    qualquer uma delas.
    """

    def on_mousewheel(event):
        # Repassa o evento para o canvas principal
        # Isso cria uma "ponte" entre o widget local e o canvas de scroll
        if event.delta:
            scroll_amount = -1 * (event.delta / 120)
        else:
            scroll_amount = -1 if event.num == 4 else 1

        self.canvas.yview_scroll(int(scroll_amount * 2), "units")
        return "break"

    # Vincula para todos os sistemas operacionais
    widget.bind("<MouseWheel>", on_mousewheel)  # Windows
    widget.bind("<Button-4>", on_mousewheel)    # Linux/Mac
    widget.bind("<Button-5>", on_mousewheel)    # Linux/Mac


def configure_smooth_scrolling(self):
    """
    Configura scroll suave e adiciona atalhos de teclado para navegação.

    Esta função melhora significativamente a experiência do usuário ao
    adicionar várias formas de navegar pelo documento.
    """

    # Atalhos de teclado para navegação vertical
    # Setas: movimento fino (1 unidade por vez)
    self.master.bind("<Up>", lambda e: self.canvas.yview_scroll(-1, "units"))
    self.master.bind("<Down>", lambda e: self.canvas.yview_scroll(1, "units"))
```

```
# Page Up/Down: movimento de página (10 unidades por vez)
self.master.bind("<Prior>", lambda e: self.canvas.yview_scroll(-10, "units"))  # Page Up
self.master.bind("<Next>", lambda e: self.canvas.yview_scroll(10, "units"))   # Page Down

# Home/End: ir para início/fim do documento
# moveto aceita valores de 0.0 (topo) a 1.0 (fim)
self.master.bind("<Home>", lambda e: self.canvas.yview_moveto(0))
self.master.bind("<End>", lambda e: self.canvas.yview_moveto(1))

# Atalhos para navegação horizontal (útil para PDFs largos)
self.master.bind("<Left>", lambda e: self.canvas.xview_scroll(-1, "units"))
self.master.bind("<Right>", lambda e: self.canvas.xview_scroll(1, "units"))

# Scroll horizontal com Shift + Mouse Wheel
def on_shift_mousewheel(event):
    """
    Permite scroll horizontal quando Shift está pressionado.

    Isso é especialmente útil para PDFs que são mais largos que a tela,
    como planilhas ou diagramas horizontais.
    """
    # Verifica se Shift está pressionado
    # 0x0001 é a máscara de bits para a tecla Shift
    if event.state & 0x0001:
        if event.delta:
            scroll_amount = -1 * (event.delta / 120)
        else:
            scroll_amount = -1 if event.num == 4 else 1

        # Move horizontalmente em vez de verticalmente
        self.canvas.xview_scroll(int(scroll_amount * 2), "units")
        return "break"
```

```
    # Vincula o scroll horizontal com Shift
    self.canvas.bind("<Shift-MouseWheel>", on_shift_mousewheel)
    self.canvas.bind("<Shift-Button-4>", on_shift_mousewheel)
    self.canvas.bind("<Shift-Button-5>", on_shift_mousewheel)


def setup_menu(self):
    menubar = tk.Menu(self.master)
    filemenu = tk.Menu(menubar, tearoff=0)
    filemenu.add_command(label="Abrir PDF", command=self.open_pdf_dialog)
    menubar.add_cascade(label="Arquivo", menu=filemenu)
    self.master.config(menu=menubar)


def open_pdf_dialog(self):
    file_path = filedialog.askopenfilename(filetypes=[("PDF Files", "*.pdf")])
    if file_path:
        self.pdf_path = file_path
        self.load_pdf(file_path)


def load_pdf(self, path):
    """Carrega o PDF mantendo as coordenadas alinhadas"""
    try:
        # Limpa o estado anterior
        if hasattr(self, 'canvas_window_id'):
            self.canvas.delete(self.canvas_window_id)

        # Limpa listas
        self.page_images.clear()
        self.bbox_list.clear()
        self.bbox_rects.clear()
```

```
self.page_canvases.clear()
self.selected_bboxes.clear()
self.selection_listbox.delete(0, tk.END)


# Limpa widgets antigos
for widget in self.pdf_container.winfo_children():
    widget.destroy()


# Abre o PDF
self.pdf_doc = fitz.open(path)

# Processa cada página
for page_num, page in enumerate(self.pdf_doc):
    # Renderiza a página com zoom
    mat = fitz.Matrix(self.zoom, self.zoom)
    pix = page.get_pixmap(matrix=mat)
    img = Image.frombytes("RGB", [pix.width, pix.height], pix.samples)
    photo = ImageTk.PhotoImage(img)
    self.page_images.append(photo)


    # Frame para conter cada página (com borda para visualização)
    page_frame = tk.Frame(self.pdf_container, bg="white", relief="solid", borderwidth=1)
    page_frame.pack(pady=10)


    # Canvas para cada página
    page_canvas = tk.Canvas(
        page_frame,
        width=photo.width(),
        height=photo.height(),
```

```
    bg="white",
    highlightthickness=0
)
page_canvas.pack()

# Adiciona a imagem da página
page_canvas.create_image(0, 0, anchor="nw", image=photo)


# Processa e desenha as bboxes
page_bbox_list = []
page_bbox_rects = []


blocks = page.get_text("dict")["blocks"]
for block in blocks:
    if "lines" in block:
        for line in block["lines"]:
            for span in line["spans"]:
                text = span["text"].strip()
                if not text:  # Ignora texto vazio
                    continue

                bbox = span["bbox"]
                page_bbox_list.append((bbox, text))

                # Converte coordenadas do PDF para coordenadas do canvas (com zoom)
                x0, y0, x1, y1 = [coord * self.zoom for coord in bbox]

                # Cria retângulo visível
                rect_id = page_canvas.create_rectangle(
                    x0, y0, x1, y1,
                    outline="red",
```

```
                    width=1,
                    tags=f"bbox_page{page_num}"  # Tag para facilitar identificação
                )

                # Armazena informações do retângulo
                page_bbox_rects.append((rect_id, bbox, text))


    # Configura evento de clique para a página
    # Usa closure para capturar as variáveis corretas
    def make_click_handler(canvas, rects, page_idx):
        return lambda event: self.on_canvas_click(event, canvas, rects, page_idx)

    page_canvas.bind("<Button-1>", make_click_handler(page_canvas, page_bbox_rects, page_num))

    # Armazena referências
    self.bbox_list.append(page_bbox_list)
    self.bbox_rects.append(page_bbox_rects)
    self.page_canvases.append(page_canvas)


# Cria a janela do canvas e centraliza
self.canvas_window_id = self.canvas.create_window(0, 0, window=self.pdf_container, anchor="nw")

# NOVO: Configura o scroll após carregar o PDF
self.setup_scrolling()

# NOVO: Para cada página criada, vincula o scroll
# Isso garante que o scroll funcione em qualquer lugar do documento
for page_frame in self.pdf_container.winfo_children():
    self.bind_scroll_to_page(page_frame)
    # E também para o canvas dentro de cada frame
    for widget in page_frame.winfo_children():
```

```
            if isinstance(widget, tk.Canvas):
                self.bind_scroll_to_page(widget)

        # Aguarda o container ser renderizado e então centraliza
        self.master.after(100, self.center_pdf_container)

    except Exception as e:
        messagebox.showerror("Erro", f"Erro ao carregar PDF: {str(e)}")


def on_canvas_click(self, event, canvas, rects, page_index):
    """Processa clique nas bboxes com coordenadas corretas"""
    # Coordenadas do clique relativas ao canvas da página
    x_click, y_click = event.x, event.y

    # Verifica cada bbox
    for rect_id, bbox, text in rects:
        # Converte bbox para coordenadas do canvas (com zoom)
        x0, y0, x1, y1 = [coord * self.zoom for coord in bbox]

        # Verifica se o clique foi dentro desta bbox
        if x0 <= x_click <= x1 and y0 <= y_click <= y1:
            # Verifica se já está selecionada
            found = None
            for i, (p, r, b, t) in enumerate(self.selected_bboxes):
                if p == page_index and r == rect_id:
                    found = i
                    break

            if found is not None:
                # Desmarca a seleção
                canvas.itemconfig(rect_id, outline="red", width=1)
                self.selected_bboxes.pop(found)
```

```
                self.remove_text_from_listbox(text)
            else:
                # Marca como selecionada
                canvas.itemconfig(rect_id, outline="blue", width=2)
                self.selected_bboxes.append((page_index, rect_id, bbox, text))
                self.selection_listbox.insert(tk.END, f"Pág {page_index+1}: {text}")

            # Para após processar o primeiro match
            break


def remove_text_from_listbox(self, text):
    """Remove texto da lista de seleção"""
    items = self.selection_listbox.get(0, tk.END)
    for index, item in enumerate(items):
        # Compara apenas a parte do texto (ignora o prefixo da página)
        if text in item:
            self.selection_listbox.delete(index)
            break


def save_selection(self):
    """Salva as seleções em JSON"""
    if not self.selected_bboxes:
        messagebox.showinfo("Info", "Nenhuma seleção para salvar")
        return

    output = [
        {"page": p, "bbox": list(b), "text": t}
        for (p, _, b, t) in self.selected_bboxes
    ]

    filename = filedialog.asksaveasfilename(
```

```
            defaultextension=".json",
            filetypes=[("JSON files", "*.json")]
        )

        if filename:
            with open(filename, "w", encoding="utf-8") as f:
                json.dump(output, f, indent=2, ensure_ascii=False)
            messagebox.showinfo("Sucesso", f"Seleção salva em {filename}")


def apply_redaction(self):
    """Aplica censura nas áreas selecionadas"""
    # Validação inicial: verifica se há seleções
    if not self.selected_bboxes:
        messagebox.showinfo("Info", "Nenhuma área selecionada para censurar")
        return

    # Validação: verifica se o PDF original ainda existe e é acessível
    if not self.pdf_path or not os.path.exists(self.pdf_path):
        messagebox.showerror("Erro", "O arquivo PDF original não foi encontrado. Por favor, reabra o arquivo.")
        return

    # Obtém o diretório padrão (mesmo do arquivo original)
    initial_dir = os.path.dirname(self.pdf_path)
    initial_file = os.path.splitext(os.path.basename(self.pdf_path))[0] + "_censurado.pdf"

    # Diálogo para escolher onde salvar
    output_path = filedialog.asksaveasfilename(
        defaultextension=".pdf",
        filetypes=[("PDF files", "*.pdf")],
        initialfile=initial_file,
        initialdir=initial_dir  # Define o diretório inicial
    )
```

```
# Se o usuário cancelou o diálogo
if not output_path:
    return

# Validação: verifica se o diretório de destino existe
output_dir = os.path.dirname(output_path)
if not os.path.exists(output_dir):
    messagebox.showerror("Erro", f"O diretório {output_dir} não existe.")
    return

# Validação: verifica permissões de escrita
if os.path.exists(output_dir) and not os.access(output_dir, os.W_OK):
    messagebox.showerror("Erro", f"Sem permissão para escrever no diretório {output_dir}")
    return

try:
    # Abre o documento PDF original
    doc = fitz.open(self.pdf_path)

    # Conta quantas redações serão aplicadas
    redaction_count = len(self.selected_bboxes)

    # Aplica redação em cada área selecionada
    for (page_index, _, bbox, _) in self.selected_bboxes:
        # Validação: verifica se o índice da página é válido
        if page_index < len(doc):
            page = doc[page_index]
            page.add_redact_annot(bbox, fill=(0, 0, 0))
        else:
            print(f"Aviso: Índice de página {page_index} inválido")

    # Aplica todas as redações de uma vez
```

```
            for page in doc:
                page.apply_redactions()

            # Salva o documento censurado
            doc.save(output_path)
            doc.close()

            # Mensagem de sucesso com informações úteis
            messagebox.showinfo(
                "Sucesso",
                f"PDF censurado salvo com sucesso!\n\n"
                f"Arquivo: {os.path.basename(output_path)}\n"
                f"Local: {output_dir}\n"
                f"Áreas censuradas: {redaction_count}"
            )

        except PermissionError:
            messagebox.showerror("Erro", f"Sem permissão para salvar o arquivo em:\n{output_path}")
        except Exception as e:
            # Log detalhado do erro para debug
            error_msg = f"Erro ao aplicar censura:\n\n{type(e).__name__}: {str(e)}"
            if hasattr(e, '__traceback__'):
                import traceback
                error_details = ''.join(traceback.format_tb(e.__traceback__))
                print(f"Detalhes do erro:\n{error_details}")
            messagebox.showerror("Erro", error_msg)


    def censor_all_occurrences_dialog(self):
        """Diálogo para censurar todas as ocorrências de um texto"""
        dialog = tk.Toplevel(self.master)
        dialog.title("Censurar todas as ocorrências")
        dialog.geometry("400x200")
```

```
    tk.Label(dialog, text="Escolha uma opção:", font=("Arial", 12)).pack(pady=10)

    # Opção 1: Usar textos selecionados
    if self.selected_bboxes:
        tk.Button(
            dialog,
            text="Censurar todas as ocorrências dos textos selecionados",
            command=lambda: [self.censor_texts_from_selection(), dialog.destroy()],
            width=40
        ).pack(pady=5)

    # Opção 2: Digite o texto
    tk.Label(dialog, text="Ou digite o texto para censurar:").pack(pady=10)

    entry = tk.Entry(dialog, width=40)
    entry.pack(pady=5)
    entry.focus()

    tk.Button(
        dialog,
        text="Censurar este texto",
        command=lambda: [self.censor_text(entry.get()), dialog.destroy()],
        width=20
    ).pack(pady=5)


def censor_text(self, target_text):
    """Censura todas as ocorrências de um texto específico"""
    # Validação: texto não pode estar vazio
    if not target_text.strip():
        messagebox.showwarning("Aviso", "Por favor, digite um texto para censurar")
        return
```

```
# Validação: verifica se o PDF original ainda existe
if not self.pdf_path or not os.path.exists(self.pdf_path):
    messagebox.showerror("Erro", "O arquivo PDF original não foi encontrado. Por favor, reabra o arquivo.")
    return

# Prepara o diálogo de salvamento
initial_dir = os.path.dirname(self.pdf_path)
initial_file = os.path.splitext(os.path.basename(self.pdf_path))[0] + "_censurado_texto.pdf"


output_path = filedialog.asksaveasfilename(
    defaultextension=".pdf",
    filetypes=[("PDF files", "*.pdf")],
    initialfile=initial_file,
    initialdir=initial_dir
)

if not output_path:
    return

# Validação do diretório de destino
output_dir = os.path.dirname(output_path)
if not os.path.exists(output_dir):
    messagebox.showerror("Erro", f"O diretório {output_dir} não existe.")
    return

try:
    # Abre o documento
    doc = fitz.open(self.pdf_path)
    count = 0
    pages_affected = set()  # Para rastrear páginas que foram modificadas
```

```
# Procura e censura o texto em todas as páginas
for page_index, page in enumerate(doc):
    page_had_redactions = False
    blocks = page.get_text("dict")["blocks"]

    for block in blocks:
        if "lines" in block:
            for line in block["lines"]:
                for span in line["spans"]:
                    text = span["text"].strip()
                    # Comparação exata (case-sensitive)
                    if text == target_text.strip():
                        bbox = span["bbox"]
                        page.add_redact_annot(bbox, fill=(0, 0, 0))
                        count += 1
                        page_had_redactions = True

    if page_had_redactions:
        pages_affected.add(page_index + 1)  # +1 para número da página (1-based)

# Se não encontrou nenhuma ocorrência
if count == 0:
    doc.close()
    messagebox.showinfo("Info", f"Nenhuma ocorrência de '{target_text}' foi encontrada no documento.")
    return

# Aplica as redações
for page in doc:
    page.apply_redactions()

# Salva o documento
doc.save(output_path)
doc.close()
```

```
            # Mensagem detalhada de sucesso
            pages_list = sorted(list(pages_affected))
            pages_str = ", ".join(map(str, pages_list))

            messagebox.showinfo(
                "Sucesso",
                f"Censura aplicada com sucesso!\n\n"
                f"Texto censurado: '{target_text}'\n"
                f"Ocorrências: {count}\n"
                f"Páginas afetadas: {pages_str}\n\n"
                f"Arquivo salvo em:\n{output_path}"
            )

        except PermissionError:
            messagebox.showerror("Erro", f"Sem permissão para salvar o arquivo em:\n{output_path}")
        except Exception as e:
            error_msg = f"Erro ao censurar texto:\n\n{type(e).__name__}: {str(e)}"
            messagebox.showerror("Erro", error_msg)


    def censor_texts_from_selection(self):
        """Censura todas as ocorrências dos textos selecionados"""
        # Validação inicial
        if not self.selected_bboxes:
            messagebox.showinfo("Info", "Nenhum texto selecionado para censurar")
            return

        # Validação: verifica se o PDF original ainda existe
        if not self.pdf_path or not os.path.exists(self.pdf_path):
            messagebox.showerror("Erro", "O arquivo PDF original não foi encontrado. Por favor, reabra o arquivo.")
            return
```

```
# Coleta textos únicos das seleções
textos_alvo = set(t.strip() for (_, _, _, t) in self.selected_bboxes)

# Remove textos vazios do conjunto
textos_alvo.discard("")  # Remove string vazia se existir

if not textos_alvo:
    messagebox.showinfo("Info", "Nenhum texto válido encontrado nas seleções")
    return

# Prepara o diálogo de salvamento
initial_dir = os.path.dirname(self.pdf_path)
initial_file = os.path.splitext(os.path.basename(self.pdf_path))[0] + "_censurado_selecao.pdf"

output_path = filedialog.asksaveasfilename(
    defaultextension=".pdf",
    filetypes=[("PDF files", "*.pdf")],
    initialfile=initial_file,
    initialdir=initial_dir
)

if not output_path:
    return

# Validação do diretório de destino
output_dir = os.path.dirname(output_path)
if not os.path.exists(output_dir):
    messagebox.showerror("Erro", f"O diretório {output_dir} não existe.")
    return

try:
    # Abre o documento
    doc = fitz.open(self.pdf_path)
```

```
count = 0
text_count = {}  # Contador para cada texto censurado
pages_affected = set()

# Inicializa contador para cada texto
for texto in textos_alvo:
    text_count[texto] = 0

# Procura e censura todos os textos selecionados
for page_index, page in enumerate(doc):
    page_had_redactions = False
    blocks = page.get_text("dict")["blocks"]

    for block in blocks:
        if "lines" in block:
            for line in block["lines"]:
                for span in line["spans"]:
                    text = span["text"].strip()
                    if text in textos_alvo:
                        bbox = span["bbox"]
                        page.add_redact_annot(bbox, fill=(0, 0, 0))
                        count += 1
                        text_count[text] += 1
                        page_had_redactions = True

    if page_had_redactions:
        pages_affected.add(page_index + 1)

# Se não encontrou nenhuma ocorrência adicional
if count == 0:
    doc.close()
    messagebox.showinfo(
        "Info",
```

```
            "Nenhuma ocorrência adicional dos textos selecionados foi encontrada.\n"
            "(As ocorrências já selecionadas serão mantidas para censura individual)"
        )
        return

    # Aplica as redações
    for page in doc:
        page.apply_redactions()

    # Salva o documento
    doc.save(output_path)
    doc.close()

    # Prepara relatório detalhado
    report_lines = ["Censura aplicada com sucesso!\n"]
    report_lines.append(f"Total de ocorrências censuradas: {count}")
    report_lines.append(f"Textos diferentes censurados: {len(textos_alvo)}")
    report_lines.append(f"Páginas afetadas: {', '.join(map(str, sorted(pages_affected)))}\n")

    # Adiciona detalhes por texto (apenas os mais frequentes)
    report_lines.append("Detalhamento:")
    sorted_texts = sorted(text_count.items(), key=lambda x: x[1], reverse=True)
    for i, (texto, qtd) in enumerate(sorted_texts[:5]):  # Mostra até 5 textos
        if qtd > 0:
            # Trunca textos muito longos para a exibição
            display_text = texto if len(texto) <= 30 else texto[:27] + "..."
            report_lines.append(f"  • '{display_text}': {qtd} ocorrências")

    if len(sorted_texts) > 5:
        report_lines.append(f"  • ... e {len(sorted_texts) - 5} outros textos")

    report_lines.append(f"\nArquivo salvo em:\n{output_path}")
```

```
            messagebox.showinfo("Sucesso", "\n".join(report_lines))

        except PermissionError:
            messagebox.showerror("Erro", f"Sem permissão para salvar o arquivo em:\n{output_path}")
        except Exception as e:
            error_msg = f"Erro ao censurar seleção:\n\n{type(e).__name__}: {str(e)}"
            messagebox.showerror("Erro", error_msg)


    def zoom_in(self):
        """Aumenta o zoom em 25%"""
        self.zoom *= 1.25
        if self.pdf_path:
            self.load_pdf(self.pdf_path)


    def zoom_out(self):
        """Diminui o zoom em 25%"""
        self.zoom /= 1.25
        if self.pdf_path:
            self.load_pdf(self.pdf_path)


if __name__ == "__main__":
    root = tk.Tk()
    root.geometry("1200x800")
    app = PDFViewerApp(root)
    root.mainloop()
```