import tkinter as tk

from tkinter import messagebox, filedialog

from tkinter import PhotoImage

from tkinter import ttk

from tkinterdnd2 import DND\_FILES, TkinterDnD

def update\_line\_numbers(text\_area, line\_numbers):

    """Обновление номеров строк."""

    line\_numbers.config(state=tk.NORMAL)

    line\_numbers.delete("1.0", tk.END)

    line\_count = text\_area.index(tk.END).split(".")[0]

    line\_numbers.insert(tk.END, "\n".join(str(i) for i in range(1, int(line\_count))))

    line\_numbers.config(state=tk.DISABLED)

def on\_text\_change(event=None, text\_area=None, line\_numbers=None):

    """Обработчик изменения текста."""

    update\_line\_numbers(text\_area, line\_numbers)

def create\_document():

    """Создание нового документа."""

    new\_tab = ttk.Frame(notebook)

    notebook.add(new\_tab, text="Новый документ")

    frame = tk.Frame(new\_tab)

    frame.pack(fill=tk.BOTH, expand=True)

    line\_numbers = tk.Text(frame, width=4, padx=3, takefocus=0, border=0, background="lightgray", state=tk.DISABLED)

    line\_numbers.pack(side=tk.LEFT, fill=tk.Y)

    text\_area = tk.Text(frame, undo=True, font=("Arial", selected\_size.get()))

    text\_area.pack(side=tk.RIGHT, expand=True, fill=tk.BOTH)

    text\_area.bind("<KeyRelease>", lambda event: on\_text\_change(event, text\_area, line\_numbers))

    new\_tab.text\_area = text\_area

    new\_tab.file\_path = None

    update\_line\_numbers(text\_area, line\_numbers)

def open\_document():

    """Открытие документа."""

    file\_path = filedialog.askopenfilename(filetypes=[("Text Files", "\*.txt"), ("All Files", "\*.\*")])

    if file\_path:

        new\_tab = ttk.Frame(notebook)

        notebook.add(new\_tab, text=file\_path.split("/")[-1])

        frame = tk.Frame(new\_tab)

        frame.pack(fill=tk.BOTH, expand=True)

        line\_numbers = tk.Text(frame, width=4, padx=3, takefocus=0, border=0, background="lightgray", state=tk.DISABLED)

        line\_numbers.pack(side=tk.LEFT, fill=tk.Y)

        text\_area = tk.Text(frame, undo=True, font=("Arial", selected\_size.get()))

        text\_area.pack(side=tk.RIGHT, expand=True, fill=tk.BOTH)

        text\_area.bind("<KeyRelease>", lambda event: on\_text\_change(event, text\_area, line\_numbers))

        with open(file\_path, "r", encoding="utf-8") as file:

            text\_area.insert(tk.END, file.read())

        new\_tab.text\_area = text\_area

        new\_tab.file\_path = file\_path

        update\_line\_numbers(text\_area, line\_numbers)

def save\_document():

    """Сохранить документ."""

    current\_tab = notebook.nametowidget(notebook.select())

    if current\_tab and hasattr(current\_tab, "text\_area"):

        text\_area = current\_tab.text\_area

        file\_path = current\_tab.file\_path

        if file\_path:

            with open(file\_path, "w", encoding="utf-8") as file:

                file.write(text\_area.get("1.0", tk.END))

        else:

            save\_document\_as()

def save\_document\_as():

    """Сохранить документ с новым именем."""

    current\_tab = notebook.nametowidget(notebook.select())

    if current\_tab and hasattr(current\_tab, "text\_area"):

        text\_area = current\_tab.text\_area

        file\_path = filedialog.asksaveasfilename(defaultextension=".txt",

                                                 filetypes=[("Text Files", "\*.txt"), ("All Files", "\*.\*")])

        if file\_path:

            with open(file\_path, "w", encoding="utf-8") as file:

                file.write(text\_area.get("1.0", tk.END))

            notebook.tab(current\_tab, text=file\_path.split("/")[-1])

            current\_tab.file\_path = file\_path

def update\_font\_size(\*args):

    """Обновить размер шрифта в активной вкладке, не меняя размер окна."""

    size = selected\_size.get()

    current\_tab = notebook.nametowidget(notebook.select())

    if current\_tab and hasattr(current\_tab, "text\_area"):

        current\_tab.text\_area.config(font=("Arial", size))

def get\_active\_text\_area():

    """Получить текстовое поле активной вкладки."""

    current\_tab = notebook.nametowidget(notebook.select())

    return current\_tab.text\_area if current\_tab and hasattr(current\_tab, "text\_area") else None

def setup\_hotkeys(root):

    root.bind("<Control-n>", lambda event: create\_document())

    root.bind("<Control-o>", lambda event: open\_document())

    root.bind("<Control-s>", lambda event: save\_document())

    root.bind("<Control-Shift-S>", lambda event: save\_document\_as())

    root.bind("<Control-q>", lambda event: exit\_program())

    root.bind("<Control-z>", lambda event: undo(get\_active\_text\_area()))

    root.bind("<Control-y>", lambda event: redo(get\_active\_text\_area()))

    root.bind("<Control-x>", lambda event: cut\_text(get\_active\_text\_area()))

    root.bind("<Control-c>", lambda event: copy\_text(get\_active\_text\_area()))

    root.bind("<Control-v>", lambda event: paste\_text(get\_active\_text\_area()))

    root.bind("<Control-a>", lambda event: select\_all(get\_active\_text\_area()))

def undo(text\_area):

    """Отменить действие."""

    if text\_area:

        text\_area.event\_generate("<<Undo>>")

def redo(text\_area):

    """Повторить действие."""

    if text\_area:

        text\_area.event\_generate("<<Redo>>")

def cut\_text(text\_area):

    """Вырезать текст."""

    if text\_area:

        text\_area.event\_generate("<<Cut>>")

def copy\_text(text\_area):

    """Копировать текст."""

    if text\_area:

        text\_area.event\_generate("<<Copy>>")

def paste\_text(text\_area):

    """Вставить текст."""

    if text\_area:

        text\_area.event\_generate("<<Paste>>")

def select\_all(text\_area):

    """Выделить весь текст."""

    if text\_area:

        text\_area.tag\_add("sel", "1.0", tk.END)

def exit\_program():

    """Выход из программы."""

    root.quit()

def delete\_text(text\_area):

    """Удалить текст."""

    text\_area.delete("1.0", tk.END)

def show\_help():

    """Показать справку."""

    messagebox.showinfo("Справка", "Это руководство пользователя.")

def about():

    """Информация о программе."""

    messagebox.showinfo("О программе", "Информация о программе.")

def syntax\_analysis():

    """Синтаксический анализ."""

    messagebox.showinfo("Пуск", "Анализ запущен.")

def task\_statement():

    """Постановка задачи."""

    messagebox.showinfo("Текст", "Постановка задачи.")

def grammar():

    """Грамматика."""

    messagebox.showinfo("Текст", "Грамматика.")

def grammar\_classification():

    """Классификация грамматики."""

    messagebox.showinfo("Текст", "Классификация грамматики.")

def analysis\_method():

    """Метод анализа."""

    messagebox.showinfo("Текст", "Метод анализа.")

def error\_handling():

    """Диагностика ошибок."""

    messagebox.showinfo("Текст", "Диагностика и нейтрализация ошибок.")

def test\_example():

    """Тестовый пример."""

    messagebox.showinfo("Текст", "Тестовый пример.")

def bibliography():

    """Список литературы."""

    messagebox.showinfo("Текст", "Список литературы.")

def source\_code():

    """Исходный код."""

    messagebox.showinfo("Текст", "Исходный код программы.")

# Создание окна

root = tk.Tk()

root.title("Редактор")

# Устанавливаем начальные размеры окна

root.geometry("800x600")  # Устанавливаем начальные размеры окна (ширина x высота)

# Главное меню

menu\_bar = tk.Menu(root)

# Файл

file\_menu = tk.Menu(menu\_bar, tearoff=0)

file\_menu.add\_command(label="Создать", command=create\_document)

file\_menu.add\_command(label="Открыть", command=open\_document)

file\_menu.add\_command(label="Сохранить как", command=save\_document\_as)

file\_menu.add\_separator()

file\_menu.add\_command(label="Выход", command=exit\_program)

menu\_bar.add\_cascade(label="Файл", menu=file\_menu)

# Правка

edit\_menu = tk.Menu(menu\_bar, tearoff=0)

edit\_menu.add\_command(label="Отменить", command=undo)

edit\_menu.add\_command(label="Повторить", command=redo)

edit\_menu.add\_separator()

edit\_menu.add\_command(label="Вырезать", command=cut\_text)

edit\_menu.add\_command(label="Копировать", command=copy\_text)

edit\_menu.add\_command(label="Вставить", command=paste\_text)

edit\_menu.add\_separator()

edit\_menu.add\_command(label="Удалить", command=delete\_text)

edit\_menu.add\_command(label="Выделить все", command=select\_all)

menu\_bar.add\_cascade(label="Правка", menu=edit\_menu)

# Пуск

run\_menu = tk.Menu(menu\_bar, tearoff=0)

run\_menu.add\_command(label="Синтаксический анализ", command=syntax\_analysis)

menu\_bar.add\_cascade(label="Пуск", menu=run\_menu)

# Справка

help\_menu = tk.Menu(menu\_bar, tearoff=0)

help\_menu.add\_command(label="Вызов справки", command=show\_help)

help\_menu.add\_separator()

help\_menu.add\_command(label="О программе", command=about)

menu\_bar.add\_cascade(label="Справка", menu=help\_menu)

root.config(menu=menu\_bar)

# Панель инструментов

toolbar = tk.Frame(root)

#"C:\Users\Никита\Desktop\КР\icons\copy.png"

# Загрузка изображений

icons = {

    "Создать": PhotoImage(file="C:/Users/Никита/Desktop/КР/icons/new.png"),

    "Открыть": PhotoImage(file="C:/Users/Никита/Desktop/КР/icons/open.png"),

    "Сохранить": PhotoImage(file="C:/Users/Никита/Desktop/КР/icons/save.png"),

    "Сохранить как": PhotoImage(file="C:/Users/Никита/Desktop/КР/icons/save.png"),

    "Отменить": PhotoImage(file="C:/Users/Никита/Desktop/КР/icons/undo.png"),

    "Повторить": PhotoImage(file="C:/Users/Никита/Desktop/КР/icons/redo.png"),

    "Копировать": PhotoImage(file="C:/Users/Никита/Desktop/КР/icons/copy.png"),

    "Вырезать": PhotoImage(file="C:/Users/Никита/Desktop/КР/icons/cut.png"),

    "Вставить": PhotoImage(file="C:/Users/Никита/Desktop/КР/icons/paste.png"),

}

buttons = [

    ("Создать", create\_document),

    ("Открыть", open\_document),

    ("Сохранить как", save\_document),

    ("Отменить", lambda: undo(get\_active\_text\_area())),

    ("Повторить", lambda: redo(get\_active\_text\_area())),

    ("Копировать", lambda: copy\_text(get\_active\_text\_area())),

    ("Вырезать", lambda: cut\_text(get\_active\_text\_area())),

    ("Вставить", lambda: paste\_text(get\_active\_text\_area())),

]

for text, command in buttons:

    btn = tk.Button(toolbar, image=icons.get(text), command=command)

    btn.image = icons.get(text)  # Чтобы изображение не удалялось сборщиком мусора

    btn.pack(side=tk.LEFT, padx=2, pady=2)

toolbar.pack(fill=tk.X)

# Размер шрифта

font\_sizes = [8, 10, 12, 14, 16, 18, 20, 24, 28, 32, 36]

selected\_size = tk.IntVar(value=12)

font\_size\_menu = tk.OptionMenu(toolbar, selected\_size, \*font\_sizes, command=update\_font\_size)

font\_size\_menu.pack(side=tk.LEFT, padx=5, pady=2)

# Функция для обновления размера шрифта

def update\_font\_size(event=None):

    size = selected\_size.get()

    # Обновляем размер шрифта на текущей вкладке

    for text\_widget in notebook.winfo\_children():

        for widget in text\_widget.winfo\_children():

            if isinstance(widget, tk.Text):

                widget.config(font=("Arial", size))

from tkinter import ttk

# Словарь языков

LANGUAGES = {

    "Русский": {

        "file": "Файл",

        "edit": "Правка",

        "run": "Пуск",

        "help": "Справка",

        "create": "Создать",

        "open": "Открыть",

        "save\_as": "Сохранить как",

        "exit": "Выход",

        "undo": "Отменить",

        "redo": "Повторить",

        "cut": "Вырезать",

        "copy": "Копировать",

        "paste": "Вставить",

        "delete": "Удалить",

        "select\_all": "Выделить все",

        "syntax\_analysis": "Синтаксический анализ",

        "about": "О программе",

        "help\_call": "Вызов справки",

    },

    "English": {

        "file": "File",

        "edit": "Edit",

        "run": "Run",

        "help": "Help",

        "create": "New",

        "open": "Open",

        "save\_as": "Save As",

        "exit": "Exit",

        "undo": "Undo",

        "redo": "Redo",

        "cut": "Cut",

        "copy": "Copy",

        "paste": "Paste",

        "delete": "Delete",

        "select\_all": "Select All",

        "syntax\_analysis": "Syntax Analysis",

        "about": "About",

        "help\_call": "Help",

    },

}

# Переменная для хранения текущего языка

selected\_language = tk.StringVar(value="Русский")

def change\_language(event=None):

    """Изменяет язык интерфейса."""

    lang = selected\_language.get()

    translation = LANGUAGES[lang]

    # Обновление текста меню

    menu\_bar.entryconfig(1, label=translation["file"])

    menu\_bar.entryconfig(2, label=translation["edit"])

    menu\_bar.entryconfig(3, label=translation["run"])

    menu\_bar.entryconfig(4, label=translation["help"])

    file\_menu.entryconfig(0, label=translation["create"])

    file\_menu.entryconfig(1, label=translation["open"])

    file\_menu.entryconfig(2, label=translation["save\_as"])

    file\_menu.entryconfig(4, label=translation["exit"])

    edit\_menu.entryconfig(0, label=translation["undo"])

    edit\_menu.entryconfig(1, label=translation["redo"])

    edit\_menu.entryconfig(3, label=translation["cut"])

    edit\_menu.entryconfig(4, label=translation["copy"])

    edit\_menu.entryconfig(5, label=translation["paste"])

    edit\_menu.entryconfig(7, label=translation["delete"])

    edit\_menu.entryconfig(8, label=translation["select\_all"])

    run\_menu.entryconfig(0, label=translation["syntax\_analysis"])

    help\_menu.entryconfig(0, label=translation["help\_call"])

    help\_menu.entryconfig(2, label=translation["about"])

# Добавляем выпадающий список выбора языка

language\_menu = ttk.Combobox(toolbar, textvariable=selected\_language, values=list(LANGUAGES.keys()), state="readonly")

language\_menu.pack(side=tk.RIGHT, padx=10)

language\_menu.bind("<<ComboboxSelected>>", change\_language)

# Строка состояния

status\_var = tk.StringVar()

status\_label = tk.Label(root, textvariable=status\_var, relief=tk.SUNKEN, anchor="w")

status\_label.pack(side=tk.BOTTOM, fill=tk.X)

# Создание вкладок

notebook = ttk.Notebook(root)

notebook.pack(fill=tk.BOTH, expand=True)

# Окно для вывода результатов

output\_frame = tk.Frame(root)

output\_frame.pack(fill=tk.BOTH, expand=False)

# Создание таблицы

columns = ("path", "line", "error")

output\_table = ttk.Treeview(output\_frame, columns=columns, show="headings")

# Определяем заголовки столбцов

output\_table.heading("path", text="Путь к файлу")

output\_table.heading("line", text="Строка")

output\_table.heading("error", text="Ошибка")

# Определяем ширину колонок

output\_table.column("path", width=300)

output\_table.column("line", width=40, anchor="center")

output\_table.column("error", width=400)

# Добавляем скроллбар

scrollbar = ttk.Scrollbar(output\_frame, orient="vertical", command=output\_table.yview)

output\_table.configure(yscroll=scrollbar.set)

# Располагаем элементы

output\_table.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

# Функция для добавления ошибки в таблицу

def add\_error\_to\_table(file\_path, line, error\_msg):

    """Добавить ошибку в таблицу."""

    output\_table.insert("", tk.END, values=(file\_path, line, error\_msg))

setup\_hotkeys(root)

root.mainloop()