import tkinter as tk

from tkinter import ttk, messagebox

import random

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk import pos\_tag

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

class MCQGeneratorApp:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("MCQ Generator - NLP Based")

        self.root.geometry("850x700")

        self.root.config(bg="#fdf6e3")

        self.questions = []

        self.user\_answers = []

        self.current\_question = 0

        self.score = 0

        self.create\_widgets()

    def create\_widgets(self):

        title = tk.Label(self.root, text="MCQ Generator (NLP-Based)", font=("Helvetica", 20, "bold"),

                         bg="#fdf6e3", fg="#8B4513")

        title.pack(pady=20)

        input\_frame = tk.Frame(self.root, bg="#fffaf0", bd=2, relief=tk.RIDGE)

        input\_frame.pack(padx=20, pady=10, fill="x")

        tk.Label(input\_frame, text="Paste Text Below:", font=("Arial", 12), bg="#fffaf0", fg="#8B4513").pack(anchor="w", padx=10, pady=5)

        self.input\_text = tk.Text(input\_frame, height=10, font=("Arial", 12), wrap=tk.WORD)

        self.input\_text.pack(padx=10, pady=5, fill="x")

        control\_frame = tk.Frame(input\_frame, bg="#fffaf0")

        control\_frame.pack(pady=10)

        tk.Label(control\_frame, text="Number of MCQs: ", bg="#fffaf0", fg="#8B4513").pack(side=tk.LEFT)

        self.num\_mcqs = ttk.Combobox(control\_frame, values=[5, 10, 15, 20, 25, 30], width=5)

        self.num\_mcqs.set(5)

        self.num\_mcqs.pack(side=tk.LEFT, padx=10)

        ttk.Button(control\_frame, text="Generate MCQs", command=self.generate\_mcqs).pack(side=tk.LEFT, padx=10)

        # Add Scrollable Question Frame

        container = tk.Frame(self.root)

        container.pack(fill="both", expand=True, padx=20, pady=10)

        canvas = tk.Canvas(container, bg="#f5f5dc")

        scrollbar = ttk.Scrollbar(container, orient="vertical", command=canvas.yview)

        self.q\_frame = tk.Frame(canvas, bg="#f5f5dc")

        self.q\_frame.bind("<Configure>", lambda e: canvas.configure(scrollregion=canvas.bbox("all")))

        canvas.create\_window((0, 0), window=self.q\_frame, anchor="nw")

        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)

        scrollbar.pack(side="right", fill="y")

        self.canvas = canvas  # Save reference to scroll programmatically if needed

    def generate\_mcqs(self):

        text = self.input\_text.get("1.0", tk.END).strip()

        try:

            num = int(self.num\_mcqs.get())

        except:

            messagebox.showerror("Error", "Invalid number of MCQs selected.")

            return

        if not text or len(text.split()) < 10:

            messagebox.showerror("Error", "Please enter a longer text.")

            return

        sentences = sent\_tokenize(text)

        random.shuffle(sentences)

        noun\_pool = []

        for sentence in sentences:

            tagged = pos\_tag(word\_tokenize(sentence))

            for word, tag in tagged:

                if tag.startswith('NN') and word.isalpha():

                    noun\_pool.append(word)

        self.questions.clear()

        self.user\_answers.clear()

        self.score = 0

        self.current\_question = 0

        for sentence in sentences:

            words = word\_tokenize(sentence)

            tagged = pos\_tag(words)

            blanks = [word for word, tag in tagged if tag.startswith('NN') and word.isalpha()]

            if not blanks:

                continue

            correct\_answer = random.choice(blanks)

            blanked\_sentence = sentence.replace(correct\_answer, "\_\_\_\_\_")

            distractors = list(set([w for w in noun\_pool if w != correct\_answer]))

            if len(distractors) < 2:

                continue

            options = random.sample(distractors, 2) + [correct\_answer]

            random.shuffle(options)

            correct\_index = options.index(correct\_answer)

            self.questions.append({

                "question": blanked\_sentence,

                "options": options,

                "answer\_index": correct\_index

            })

            if len(self.questions) >= num:

                break

        if not self.questions:

            messagebox.showinfo("No Questions", "Could not generate MCQs from the given text.")

            return

        self.show\_question()

    def show\_question(self):

        for widget in self.q\_frame.winfo\_children():

            widget.destroy()

        if self.current\_question >= len(self.questions):

            self.show\_results()

            return

        q = self.questions[self.current\_question]

        tk.Label(self.q\_frame, text=f"Q{self.current\_question + 1}: {q['question']}",

                 font=("Arial", 14, "bold"), bg="#f5f5dc", fg="#654321", wraplength=750, justify="left").pack(pady=10)

        self.selected\_option = tk.IntVar()

        for idx, option in enumerate(q["options"]):

            rb = tk.Radiobutton(self.q\_frame, text=option, variable=self.selected\_option, value=idx,

                                font=("Arial", 12), bg="#f5f5dc", fg="#4b3832", anchor="w", wraplength=700, justify="left")

            rb.pack(anchor="w", padx=20, pady=5)

        tk.Button(self.q\_frame, text="Next", command=self.next\_question,

                  bg="#8B4513", fg="white", font=("Arial", 12, "bold")).pack(pady=10)

    def next\_question(self):

        selected = self.selected\_option.get()

        q = self.questions[self.current\_question]

        self.user\_answers.append(selected)

        if selected == q["answer\_index"]:

            self.score += 1

        self.current\_question += 1

        self.show\_question()

    def show\_results(self):

        for widget in self.q\_frame.winfo\_children():

            widget.destroy()

        tk.Label(self.q\_frame, text=f"🎯 Your Score: {self.score} out of {len(self.questions)}",

                 font=("Arial", 16, "bold"), bg="#f5f5dc", fg="green").pack(pady=10)

        for i, q in enumerate(self.questions):

            user\_ans = self.user\_answers[i]

            correct = q["answer\_index"]

            color = "#e0ffe0" if user\_ans == correct else "#ffe0e0"

            summary = tk.Frame(self.q\_frame, bg=color, bd=1, relief=tk.SOLID)

            summary.pack(fill="x", pady=5, padx=5)

            tk.Label(summary, text=f"Q{i + 1}: {q['question']}", font=("Arial", 12, "bold"),

                     bg=summary["bg"], wraplength=750, justify="left").pack(anchor="w", padx=10, pady=3)

            for idx, opt in enumerate(q["options"]):

                tag = ""

                if idx == correct:

                    tag = "✅ Correct"

                elif idx == user\_ans:

                    tag = "❌ Your choice"

                display\_text = f"{chr(65 + idx)}. {opt} {tag}"

                tk.Label(summary, text=display\_text, font=("Arial", 11), bg=summary["bg"],

                         wraplength=730, justify="left").pack(anchor="w", padx=20)

        tk.Button(self.q\_frame, text="Restart", command=self.restart, bg="#8B4513",

                  fg="white", font=("Arial", 12, "bold")).pack(pady=10)

    def restart(self):

        self.input\_text.delete("1.0", tk.END)

        for widget in self.q\_frame.winfo\_children():

            widget.destroy()

        self.score = 0

        self.questions = []

        self.user\_answers = []

        self.current\_question = 0

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    app = MCQGeneratorApp(root)

    root.mainloop()