МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«КУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет физики, математики, информатики

Кафедра программного обеспечения и администрирования информационных систем

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

Интерфейсы программирования приложений

на тему: *Проектирование пользовательского интерфейса десктопного приложения*

Обучающегося 2 курса очной формы обучения

направления подготовки

09.03.01 Информатика и вычислительная техника

Направленность (профиль)

Прикладной искусственный интеллект

Никитина Андрея Андреевича

Руководитель:

старший преподаватель кафедры ПОАИС

Ураева Елена Евгеньевна

Курск, 2025

**Индивидуальное задание**

1. Разработать пользовательский интерфейс для десктопного клиента, на основе макета, описанного в лабораторной работе 1.При необходимости актуализировать карту навигации и черновой прототип с учетом вида интерфейса.
2. На основании одной из стилистик разработать дизайн концепцию приложения.
3. Создать проект в среде разработке согласно разработанному интерфейсу. Обеспечить переходы между экранами приложения согласно навигационной карте при выполнении действий пользователя по описанным сценариям. Допустимо использовать в проекте статичные данные для элементов вывода.
4. Разработать систему оповещений для альтернативных веток сценария.

**Дизайн концепция приложения**

На рисунках 1-6 представлены виды приложения в различных темах.

Изображение выглядит как текст, пианино, Музыкальная клавиатура, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1 – Вид светлой темы для приложения

Изображение выглядит как пианино, Музыкальная клавиатура, музыкальный инструмент, Электропианино

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2 – Вид темной темы для приложения

Изображение выглядит как текст, снимок экрана, Шрифт, белый

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3 – Вид кнопок приложения при светлой теме в режиме свободной игры

Изображение выглядит как снимок экрана, текст, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4 – Вид кнопок приложения при темной теме в режиме свободной игры

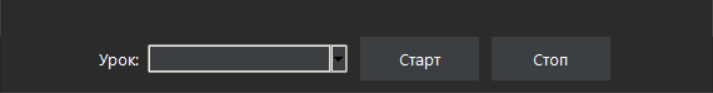


Рисунок 5 – Вид кнопок приложения при темной теме в режиме обучения



Рисунок 6 – Вид кнопок приложения при светлой теме в режиме обучения

**Проект в среде разработке**

Ниже представлен код программы в среде разработки Python :

import tkinter as tk  
from tkinter import ttk, filedialog, messagebox  
import pygame  
import os  
import json  
from datetime import datetime  
  
pygame.mixer.init()  
pygame.mixer.set\_num\_channels(32)  
  
class PianoApp:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Virtual Piano")  
 self.root.geometry("570x550")  
 self.root.configure(bg="#f0f2f5")  
 self.root.option\_add("\*Font", ("Segoe UI", 10))  
  
 self.style = ttk.Style()  
 self.style.theme\_use('clam')  
  
 self.style.configure("TButton",  
 background="#ffffff",  
 foreground="#333333",  
 padding=6,  
 relief="flat",  
 font=("Segoe UI", 10))  
 self.style.map("TButton",  
 background=[('active', '#e6e6e6')],  
 foreground=[('active', '#000000')])  
  
 self.style.configure("Title.TLabel",  
 font=("Segoe UI", 12, "bold"),  
 background="#f0f2f5",  
 foreground="#222222")  
  
 self.style.configure("TFrame", background="#f0f2f5")  
  
 self.is\_recording = False  
 self.record\_start\_time = None  
 self.recorded\_notes = []  
 self.current\_lesson = None  
 self.show\_settings = False  
 self.settings = {  
 'volume': 0.7,  
 'show\_note\_names': True,  
 'highlight\_color': '#FFD700',  
 'keyboard\_octaves': 2,  
 'sound\_type': 'piano',  
 'effect': 'none',  
 'midi\_enabled': False,  
 'show\_dark\_theme': False  
 }  
  
 self.sounds = self.load\_sounds()  
  
 self.style = ttk.Style()  
 self.create\_widgets()  
 self.create\_piano\_keys()  
 self.update\_volume()  
 self.load\_settings()  
 self.apply\_theme()  
  
 def load\_settings(self):  
 try:  
 with open('piano\_settings.json', 'r') as f:  
 self.settings.update(json.load(f))  
 except FileNotFoundError:  
 pass  
  
 def save\_settings(self):  
 with open('piano\_settings.json', 'w') as f:  
 json.dump(self.settings, f)  
  
 def create\_widgets(self):  
 top\_frame = ttk.Frame(self.root)  
 top\_frame.pack(pady=10, fill=tk.X)  
  
 ttk.Button(top\_frame, text="🎹 Свободная игра", command=self.show\_free\_play).pack(side=tk.LEFT, padx=5)  
 ttk.Button(top\_frame, text="📚 Обучение", command=self.show\_learning\_mode).pack(side=tk.LEFT, padx=5)  
 ttk.Button(top\_frame, text="⚙️ Настройки", command=self.toggle\_settings).pack(side=tk.LEFT, padx=5)  
 ttk.Button(top\_frame, text="❌ Выход", command=self.on\_exit).pack(side=tk.RIGHT, padx=5)  
  
 self.canvas = tk.Canvas(self.root, bg="white", height=220)  
 self.canvas.pack(pady=20, fill=tk.BOTH, expand=True)  
  
 self.recording\_frame = ttk.Frame(self.root)  
 self.init\_recording\_ui()  
  
 self.learning\_frame = ttk.Frame(self.root)  
 self.init\_learning\_ui()  
  
 self.settings\_frame = ttk.Frame(self.root)  
 self.init\_settings\_ui()  
  
 def init\_recording\_ui(self):  
 ttk.Button(self.recording\_frame, text="🎤 Начать запись", command=self.start\_recording).pack(side=tk.LEFT, padx=5)  
 ttk.Button(self.recording\_frame, text="⏹ Остановить", command=self.stop\_recording).pack(side=tk.LEFT, padx=5)  
 ttk.Button(self.recording\_frame, text="💾 Сохранить", command=self.save\_recording).pack(side=tk.LEFT, padx=5)  
 ttk.Button(self.recording\_frame, text="🎵 Воспроизвести", command=self.load\_and\_play\_recording).pack(side=tk.LEFT, padx=5)  
  
 self.recording\_label = ttk.Label(self.recording\_frame, text="")  
 self.recording\_label.pack(side=tk.LEFT, padx=10)  
  
 def init\_learning\_ui(self):  
 ttk.Label(self.learning\_frame, text="Урок:").pack(side=tk.LEFT)  
 self.lesson\_combo = ttk.Combobox(self.learning\_frame,  
 values=["Гамма C-мажор", "Тренажер нот", "Детская песенка"])  
 self.lesson\_combo.pack(side=tk.LEFT, padx=5)  
 ttk.Button(self.learning\_frame, text="Старт", command=self.start\_lesson).pack(side=tk.LEFT, padx=5)  
 ttk.Button(self.learning\_frame, text="Стоп", command=self.stop\_lesson).pack(side=tk.LEFT, padx=5)  
  
 def init\_settings\_ui(self):  
 self.settings\_frame.config(padding=10, relief="groove", borderwidth=2)  
 ttk.Label(self.settings\_frame, text="Громкость:").grid(row=0, column=0, sticky='w')  
 self.volume\_scale = ttk.Scale(self.settings\_frame, from\_=0, to=1, command=lambda v: self.update\_volume())  
 self.volume\_scale.set(self.settings['volume'])  
 self.volume\_scale.grid(row=0, column=1, sticky='ew')  
  
 self.show\_notes\_var = tk.BooleanVar(value=self.settings['show\_note\_names'])  
 ttk.Checkbutton(self.settings\_frame, text="Показывать названия нот",  
 variable=self.show\_notes\_var, command=self.toggle\_note\_names).grid(row=1, columnspan=2, sticky='w')  
  
 ttk.Label(self.settings\_frame, text="Цвет подсветки:").grid(row=2, column=0, sticky='w')  
 self.color\_entry = ttk.Entry(self.settings\_frame)  
 self.color\_entry.insert(0, self.settings['highlight\_color'])  
 self.color\_entry.grid(row=2, column=1, sticky='ew')  
  
 ttk.Label(self.settings\_frame, text="Инструмент:").grid(row=3, column=0, sticky='w')  
 self.sound\_type\_var = tk.StringVar(value=self.settings['sound\_type'])  
 ttk.Combobox(self.settings\_frame, textvariable=self.sound\_type\_var,  
 values=['piano', 'organ', 'synth']).grid(row=3, column=1, sticky='ew')  
  
 ttk.Label(self.settings\_frame, text="Эффект:").grid(row=4, column=0, sticky='w')  
 self.effect\_var = tk.StringVar(value=self.settings['effect'])  
 ttk.Combobox(self.settings\_frame, textvariable=self.effect\_var,  
 values=['none', 'reverb', 'echo']).grid(row=4, column=1, sticky='ew')  
  
 self.midi\_var = tk.BooleanVar(value=self.settings['midi\_enabled'])  
 ttk.Checkbutton(self.settings\_frame, text="Включить MIDI-клавиатуру",  
 variable=self.midi\_var).grid(row=5, columnspan=2, sticky='w')  
  
 self.dark\_theme\_var = tk.BooleanVar(value=self.settings['show\_dark\_theme'])  
 ttk.Checkbutton(self.settings\_frame, text="🌗 Тёмная тема",  
 variable=self.dark\_theme\_var).grid(row=6, columnspan=2, sticky='w')  
  
 ttk.Button(self.settings\_frame, text="✅ Применить", command=self.apply\_settings).grid(row=10, columnspan=2, pady=10)  
  
  
 def create\_piano\_keys(self):  
 self.canvas.delete("all")  
 white\_width = 40  
 black\_width = 24  
 key\_height = 200  
 octaves = self.settings['keyboard\_octaves']  
 total\_keys = 7 \* octaves  
  
 white\_notes\_order = ['C', 'D', 'E', 'F', 'G', 'A', 'B']  
 black\_notes\_map = {'C': 'C#', 'D': 'D#', 'F': 'F#', 'G': 'G#', 'A': 'A#'}  
  
 white\_key\_positions = []  
 key\_index = 0  
  
 for i in range(total\_keys):  
 note\_name = white\_notes\_order[i % 7]  
 octave = 4 + i // 7  
 full\_note = f"{note\_name}{octave}"  
  
 x0 = key\_index \* white\_width  
 white\_key\_positions.append((note\_name, x0, octave))  
 rect = self.canvas.create\_rectangle(  
 x0, 0, x0 + white\_width, key\_height,  
 fill="white", outline="black", tags=("white\_key", full\_note)  
 )  
 if self.settings['show\_note\_names']:  
 self.canvas.create\_text(x0 + white\_width / 2, key\_height - 20,  
 text=full\_note, tags=("label", full\_note))  
 key\_index += 1  
  
 for i in range(len(white\_key\_positions) - 1):  
 note\_name, x0, octave = white\_key\_positions[i]  
 if note\_name in black\_notes\_map:  
 black\_note = black\_notes\_map[note\_name] + str(octave)  
 bx = x0 + white\_width - black\_width // 2  
 self.canvas.create\_rectangle(  
 bx, 0, bx + black\_width, key\_height \* 0.6,  
 fill="black", outline="black", tags=("black\_key", black\_note)  
 )  
 if self.settings['show\_note\_names']:  
 self.canvas.create\_text(bx + black\_width / 2, key\_height \* 0.6 - 15,  
 text=black\_note, fill="white", tags=("label", black\_note))  
  
 self.canvas.config(bg="#e6e6e6", highlightthickness=0)  
 self.canvas.config(width=total\_keys \* white\_width)  
 self.canvas.tag\_bind("white\_key", "<Button-1>", self.on\_key\_press)  
 self.canvas.tag\_bind("black\_key", "<Button-1>", self.on\_key\_press)  
  
 def on\_key\_press(self, event):  
 item = self.canvas.find\_closest(event.x, event.y)[0]  
 tags = self.canvas.gettags(item)  
 if len(tags) > 1:  
 note = tags[1]  
 self.play\_sound(note)  
 self.animate\_key\_press(item)  
 if self.is\_recording:  
 self.record\_note(note)  
  
 def play\_sound(self, note):  
 if note in self.sounds:  
 sound = self.sounds[note]  
 channel = pygame.mixer.find\_channel()  
 if channel:  
 channel.play(sound)  
  
 if self.settings['effect'] == 'echo':  
 self.root.after(200, lambda: channel.play(sound))  
 elif self.settings['effect'] == 'reverb':  
 sound.set\_volume(self.settings['volume'] \* 0.6)  
  
 def animate\_key\_press(self, item):  
 original\_color = "white" if "white\_key" in self.canvas.gettags(item) else "black"  
 self.canvas.itemconfig(item, fill=self.settings['highlight\_color'])  
 self.root.after(100, lambda: self.canvas.itemconfig(item, fill=original\_color))  
  
 def start\_recording(self):  
 self.is\_recording = True  
 self.recorded\_notes = []  
 self.record\_start\_time = datetime.now()  
 self.recording\_label.config(text="Запись...")  
  
 def stop\_recording(self):  
 self.is\_recording = False  
 self.recording\_label.config(text=f"Записано нот: {len(self.recorded\_notes)}")  
  
 def record\_note(self, note):  
 timestamp = (datetime.now() - self.record\_start\_time).total\_seconds()  
 self.recorded\_notes.append({'note': note, 'timestamp': round(timestamp, 2)})  
  
 def save\_recording(self):  
 if not self.recorded\_notes:  
 messagebox.showwarning("Ошибка", "Нет записанных данных")  
 return  
  
 filename = filedialog.asksaveasfilename(defaultextension=".json",  
 filetypes=[("JSON files", "\*.json")])  
 if filename:  
 with open(filename, 'w') as f:  
 json.dump(self.recorded\_notes, f)  
 messagebox.showinfo("Сохранено", "Запись успешно сохранена")  
  
 def load\_and\_play\_recording(self):  
 filename = filedialog.askopenfilename(filetypes=[("JSON files", "\*.json")],  
 title="Выберите файл записи")  
 if filename:  
 try:  
 with open(filename, 'r') as f:  
 notes = json.load(f)  
 self.play\_recorded\_notes(notes)  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось загрузить файл: {e}")  
  
 def play\_recorded\_notes(self, notes):  
 if not notes:  
 messagebox.showwarning("Пусто", "Запись пуста")  
 return  
  
 for note\_data in notes:  
 delay = int(note\_data['timestamp'] \* 1000)  
 self.root.after(delay, lambda note=note\_data['note']: (  
 self.play\_sound(note),  
 self.highlight\_key(note)  
 ))  
  
 def highlight\_key(self, note):  
 items = self.canvas.find\_withtag(note)  
 for item in items:  
 original\_color = "white" if "white\_key" in self.canvas.gettags(item) else "black"  
 self.canvas.itemconfig(item, fill=self.settings['highlight\_color'])  
 self.root.after(1000, lambda: self.canvas.itemconfig(item, fill=original\_color))  
  
 def show\_free\_play(self):  
 self.learning\_frame.pack\_forget()  
 self.settings\_frame.pack\_forget()  
 self.recording\_frame.pack(pady=10)  
  
 def show\_learning\_mode(self):  
 self.recording\_frame.pack\_forget()  
 self.settings\_frame.pack\_forget()  
 self.learning\_frame.pack(pady=10)  
  
 def toggle\_settings(self):  
 if self.show\_settings:  
 self.settings\_frame.pack\_forget()  
 self.show\_settings = False  
 else:  
 self.settings\_frame.pack(pady=10, fill=tk.X)  
 self.show\_settings = True  
  
 def start\_lesson(self):  
 lesson = self.lesson\_combo.get()  
 self.current\_lesson = self.generate\_lesson(lesson)  
 self.play\_next\_note()  
  
 def stop\_lesson(self):  
 self.current\_lesson = None  
  
 def generate\_lesson(self, lesson\_name):  
 lessons = {  
 "Гамма C-мажор": ['C4', 'D4', 'E4', 'F4', 'G4', 'A4', 'B4', 'C5'],  
 "Тренажер нот": ['C4', 'D4', 'E4', 'F4', 'G4', 'A4', 'B4', 'C5'] \* 2,  
 "Детская песенка": ['C4', 'C4', 'G4', 'G4', 'A4', 'A4', 'G4',  
 'F4', 'F4', 'E4', 'E4', 'D4', 'D4', 'C4']  
 }  
 return lessons.get(lesson\_name, []).copy()  
  
 def play\_next\_note(self):  
 if self.current\_lesson:  
 if not self.current\_lesson:  
 messagebox.showinfo("Урок завершен", "Вы успешно завершили урок!")  
 return  
  
 note = self.current\_lesson.pop(0)  
 self.highlight\_key(note)  
 self.root.after(1500, self.play\_next\_note)  
  
 def update\_volume(self):  
 self.settings['volume'] = self.volume\_scale.get()  
 for sound in self.sounds.values():  
 sound.set\_volume(self.settings['volume'])  
  
 def toggle\_note\_names(self):  
 self.settings['show\_note\_names'] = self.show\_notes\_var.get()  
 self.create\_piano\_keys()  
  
 def apply\_settings(self):  
 try:  
 self.settings['highlight\_color'] = self.color\_entry.get()  
 self.settings['sound\_type'] = self.sound\_type\_var.get()  
 self.settings['effect'] = self.effect\_var.get()  
 self.settings['midi\_enabled'] = self.midi\_var.get()  
 self.settings['show\_dark\_theme'] = self.dark\_theme\_var.get()  
  
 self.apply\_theme()  
 self.create\_piano\_keys()  
 self.save\_settings()  
 except:  
 messagebox.showerror("Ошибка", "Некорректные параметры")  
  
 def apply\_theme(self):  
 if self.settings['show\_dark\_theme']:  
 bg\_color = "#2b2b2b"  
 fg\_color = "#ffffff"  
 btn\_bg = "#3c3f41"  
 highlight = "#ffaa00"  
 canvas\_bg = "#1e1e1e"  
 else:  
 bg\_color = "#f0f2f5"  
 fg\_color = "#000000"  
 btn\_bg = "#ffffff"  
 highlight = "#FFD700"  
 canvas\_bg = "#e6e6e6"  
  
 self.root.configure(bg=bg\_color)  
 self.style.configure("TFrame", background=bg\_color)  
 self.style.configure("TLabel", background=bg\_color, foreground=fg\_color)  
 self.style.configure("TCheckbutton", background=bg\_color, foreground=fg\_color)  
 self.style.configure("TButton", background=btn\_bg, foreground=fg\_color)  
 self.style.configure("TCombobox", fieldbackground=btn\_bg, background=btn\_bg)  
  
 self.canvas.config(bg=canvas\_bg)  
 self.settings['highlight\_color'] = highlight  
  
 def load\_sounds(self):  
 sounds = {}  
 notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']  
  
 if not os.path.exists("sounds"):  
 os.makedirs("sounds")  
 messagebox.showwarning("Внимание", "Папка sounds создана. Добавьте звуковые файлы!")  
  
 for note in notes:  
 for octave in [4, 5]:  
 filename = f"sounds/{note}{octave}.wav"  
 if os.path.exists(filename):  
 sounds[f"{note}{octave}"] = pygame.mixer.Sound(filename)  
 return sounds  
  
 def on\_exit(self):  
 if messagebox.askokcancel("Выход", "Вы уверены, что хотите выйти?"):  
 self.root.destroy()  
  
 def check\_midi\_input(self):  
 if not self.settings.get('midi\_enabled'):  
 return  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = PianoApp(root)  
 root.mainloop()