

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 разработки 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.



Рисунок 1 – Вид светлой темы для приложения

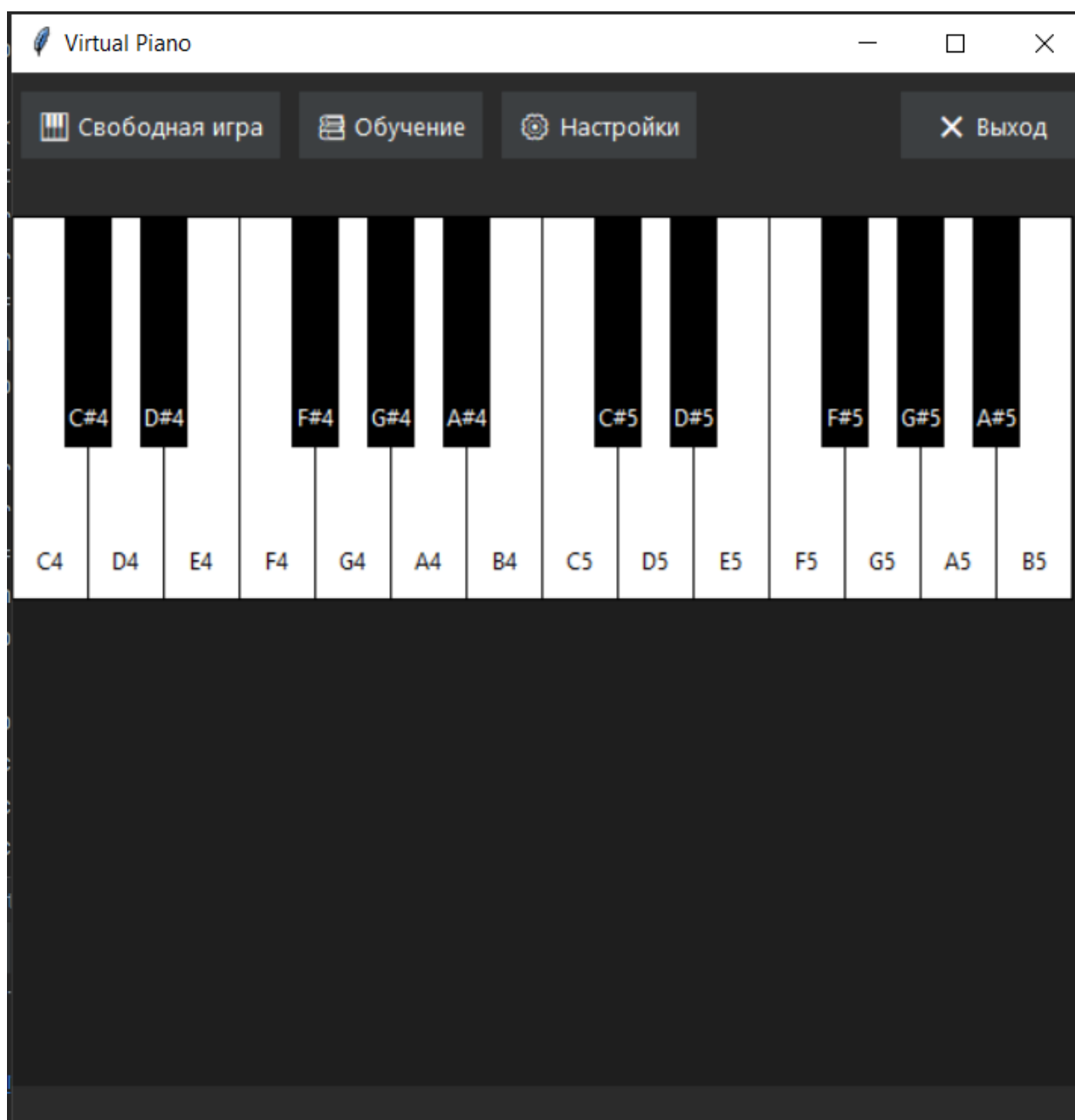


Рисунок 2 – Вид темной темы для приложения

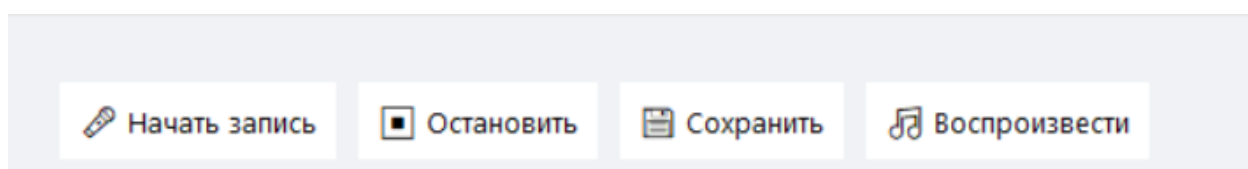


Рисунок 3 – Вид кнопок приложения при светлой теме в режиме свободной игры

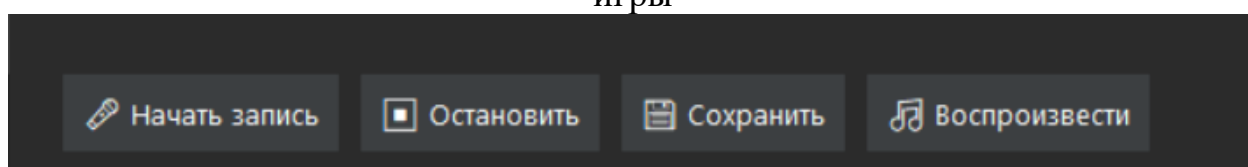


Рисунок 4 – Вид кнопок приложения при темной теме в режиме свободной игры

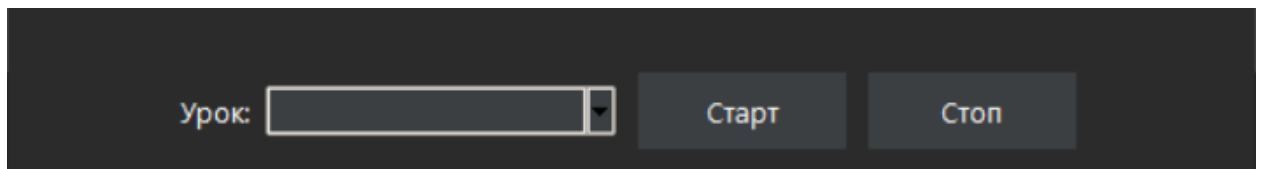


Рисунок 5 – Вид кнопок приложения при темной теме в режиме обучения

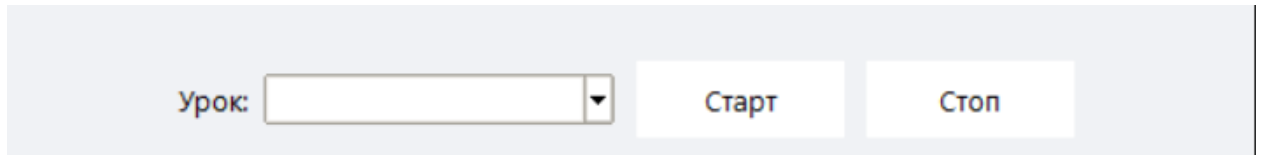


Рисунок 6 – Вид кнопок приложения при светлой теме в режиме обучения

Проект в среде разработки

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()
```