```
import tkinter as tk
import winsound
from gtts import gTTS
import pygame
class Senior_Switchman:
    def __init__(self):
        self.switch_state = [0] * 16
        self.scheduled_turn_on = {}
        self.scheduled_turn_off = {}
        self.executed_commands = []
    def turn_on(self, switch_number):
        self.switch_state[switch_number - 1] = 1

    def turn_off(self, switch_number):
        self.switch_state[switch_number - 1] = 0

    def sense_state(self, switch_number):
        return self.switch_state[switch_number - 1]

    def execute_commands(self, commands, echo=True):
        for command in commands:
            command = command.strip().split()
            if echo:
                print(" ".join(command))

            if "TIME" in command:
                time_index = command.index("TIME") + 1
                current_time = int(command[time_index])
            elif "ON" in command:
                switch_indexes = [int(x) for x in command if x.isdigit()]
                for switch_index in switch_indexes:
                    self.turn_on(switch_index)
            elif "OFF" in command:
                switch_indexes = [int(x) for x in command if x.isdigit()]
                for switch_index in switch_indexes:
                    self.turn_off(switch_index)
            elif "?" in command:
                if "ALL" in command:
                    self.print_all_switch_states()
                else:
                    switch_indexes = [int(x) for x in command if x.isdigit()]
                    for switch_index in switch_indexes:
                        state = "ON" if self.sense_state(switch_index) else
                    "OFF"
                            print(f"Switch {switch_index} state: {state}")
            elif "ALL" in command:
                if "ON" in command:
                    self.turn_on_all()
```

```
                elif "OFF" in command:
                    self.turn_off_all()
            elif "AT" in command:
                time_index = command.index("AT") + 1
                command_time = int(command[time_index])
                switch_indexes = [int(x) for x in command if x.isdigit()]
                for switch_index in switch_indexes:
                    self.schedule_turn_on_at_time(switch_index, command_time)
            elif "STOP" in command:
                print("SIMULATION END.")
                break
    def turn_on_all(self):
        self.switch_state = [1] * 16
    def turn_off_all(self):
        self.switch_state = [0] * 16
    def schedule_turn_on_at_time(self, switch_number, command_time):
        if command_time not in self.scheduled_turn_on:
            self.scheduled_turn_on[command_time] = []
        self.scheduled_turn_on[command_time].append(switch_number)
    def simulate_time(self):
        time_elapsed = 0
        for command_time in sorted(self.scheduled_turn_on.keys()):
            if command_time < time_elapsed:
                time_elapsed = 0
            time_elapsed += (command_time - time_elapsed) % 2400
            switches_to_turn_on = self.scheduled_turn_on[command_time]
            for switch_number in switches_to_turn_on:
                self.turn_on(switch_number)

    def print_all_switch_states(self):
        for switch_index, state in enumerate(self.switch_state, start=1):
            state_str = "ON" if state else "OFF"
            print(f"Switch {switch_index} state: {state_str}")
class ControlWindow(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Command Execution Window")
        self.geometry("1500x1300")

        self.switch_state = [0] * 16
        self.scheduled_turn_on = {}
        self.scheduled_turn_off = {}
        self.executed_commands = []
        try:
            self.background_img = tk.PhotoImage(file=r"C:\Users\Lenovo\Pictures
              \k.png")
            self.background_label = tk.Label(self, image=self.background_img)
            self.background_label.place(x=0, y=0, relwidth=1, relheight=1)
        except tk.TclError as e:
```

```
            print("Error loading background image:", e)

        self.label = tk.Label(self, text="Welcome to the Control Window!",
          font=("Elephant", 20))
        self.label.pack()
        self.command_label = tk.Label(self, text="Enter commands (e.g. ON 1,
          OFF 2, ? 1, ? ALL):")
        self.command_label.pack()
        self.command_entry = tk.Entry(self, width=50)
        self.command_entry.pack()

        self.execute_button = tk.Button(self, text="Execute Commands",
          command=self.execute_commands)
        self.execute_button.pack()

        self.switchman_gui_window = None
        self.switch_on_sound_path = r"C:\Users\Lenovo\Downloads\light-
          switch-156813.wav"
        self.switch_off_sound_path = r"C:\Users\Lenovo\Downloads\light-
          switch-156813.wav"
    def turn_on(self, switch_number):
        self.switch_state[switch_number - 1] = 1
        self.play_sound_effect(self.switch_on_sound_path)
    def turn_off(self, switch_number):
        self.switch_state[switch_number - 1] = 0
        self.play_sound_effect(self.switch_off_sound_path)
    def play_sound_effect(self, sound_path):
        try:
            # Play the custom audio file using winsound
            winsound.PlaySound(sound_path, winsound.SND_FILENAME)

        except Exception as e:
            print("Error playing sound:", e)
    def sense_state(self, switch_number):
        return self.switch_state[switch_number - 1]
    def execute_commands(self):
        commands = self.command_entry.get().strip().split(', ')
        self.command_entry.delete(0, tk.END)
        try:
            while commands:
                command = commands.pop(0)
                command_parts = command.split()
                if command_parts[0] == "STOP":
                    break
                if command_parts[0] == "?":
                    if len(command_parts) == 2 and command_parts[1] == "ALL":
                        self.print_all_switch_states()
                    else:
                        switch_indexes = [int(x) for x in command_parts if
```

```
                    x.isdigit()]
                        for switch_index in switch_indexes:
                            state = "ON" if self.sense_state(switch_index) else
                     "OFF"
                            print(f"Switch {switch_index} state: {state}")
                else:
                    self.execute_command(command)
                self.executed_commands.append(command)
            self.simulate_time()  # Execute scheduled commands
        except Exception as e:
            tk.MessageBox.showerror("Error", str(e))
        if not commands and not command_parts[0] == "?":
            if not self.switchman_gui_window:
                self.switchman_gui_window = SeniorSwitchmanGUI(self)
            else:
                self.switchman_gui_window.update_switch_images()
    def execute_command(self, command):
        command_parts = command.split()
        if "TIME" in command_parts:
            time_index = command_parts.index("TIME") + 1
            current_time = int(command_parts[time_index])
        elif "ON" in command_parts:
            switch_indexes = [int(x) for x in command_parts if x.isdigit()]
            for switch_index in switch_indexes:
                self.turn_on(switch_index)
        elif "OFF" in command_parts:
            switch_indexes = [int(x) for x in command_parts if x.isdigit()]
            for switch_index in switch_indexes:
                self.turn_off(switch_index)
        elif "?" in command_parts:
            if "ALL" in command_parts:
                self.print_all_switch_states()
            else:
                switch_indexes = [int(x) for x in command_parts if x.isdigit()]
                for switch_index in switch_indexes:
                    state = "ON" if self.sense_state(switch_index) else "OFF"
                    print(f"Switch {switch_index} state: {state}")
        elif "ALL" in command_parts:
            if "ON" in command_parts:
                self.turn_on_all()
            elif "OFF" in command_parts:
                self.turn_off_all()
        elif "AT" in command_parts:
            time_index = command_parts.index("AT") + 1
            command_time = int(command_parts[time_index])
            switch_indexes = [int(x) for x in command_parts if x.isdigit()]
            for switch_index in switch_indexes:
                self.schedule_turn_on_at_time(switch_index, command_time)
```

```
    def turn_on_all(self):
        self.switch_state = [1] * 16

    def turn_off_all(self):
        self.switch_state = [0] * 16

    def schedule_turn_on_at_time(self, switch_number, command_time):
        if command_time not in self.scheduled_turn_on:
            self.scheduled_turn_on[command_time] = []
        self.scheduled_turn_on[command_time].append(switch_number)

    def simulate_time(self):
        time_elapsed = 0
        for command_time in sorted(self.scheduled_turn_on.keys()):
            if command_time < time_elapsed:
                time_elapsed = 0
            time_elapsed += (command_time - time_elapsed) % 2400
            switches_to_turn_on = self.scheduled_turn_on[command_time]
            for switch_number in switches_to_turn_on:
                if self.sense_state(switch_number) == 0:
                    self.turn_on(switch_number)

    def print_all_switch_states(self):
        for switch_index, state in enumerate(self.switch_state, start=1):
            state_str = "ON" if state else "OFF"
            print(f"Switch {switch_index} state: {state_str}")

class SwitchStateWindow(tk.Toplevel):
    def __init__(self, switch_states):
        super().__init__()
        self.title("Switch States")
        self.geometry("1200x1200")

        self.switch_states = switch_states

        self.label = tk.Label(self, text="Switch States:")
        self.label.pack()

        self.text_area = tk.Text(self, height=10, width=30)
        self.text_area.pack()

        self.update_switch_states()

    def update_switch_states(self):
        self.text_area.delete("1.0", tk.END)
        for switch_number, state in self.switch_states.items():
            state_str = "ON" if state else "OFF"
            self.text_area.insert(tk.END, f"Switch {switch_number}: {state_str}↵
              \n")
```

```
class SeniorSwitchmanGUI(tk.Toplevel):
    def __init__(self, control_window):
        super().__init__(control_window)
        self.title("Senior Switchman GUI")
        self.geometry("800x800")
        self.control_window = control_window
        self.canvas = tk.Canvas(self, width=700, height=700)
        self.canvas.pack()
        try:
            self.house_img = tk.PhotoImage(file=r"C:\Users\Lenovo\Pictures
              \h.pn")
            self.house_image_id = self.canvas.create_image(350, 350,
              image=self.house_img)
        except tk.TclError as e:
            print("Error loading house image:", e)

        try:
            self.switch_on_img = tk.PhotoImage(file=r"C:\Users\Lenovo\Pictures
              \MAIN SWITCH ON.png").subsample(7)
            self.switch_off_img = tk.PhotoImage(file=r"C:\Users\Lenovo\Pictures
              \MAIN SWITCH OFF.png").subsample(7)
        except tk.TclError as e:
            print("Error loading switch images:", e)

        self.switch_buttons = []
        for i in range(16):
            switch_button = tk.Button(self, image=self.switch_off_img,
              command=lambda idx=i: self.toggle_switch(idx))
            self.switch_buttons.append(switch_button)

        self.back_button = tk.Button(self, text="Back",
          command=self.back_to_control_window)
        self.back_button.place(x=10, y=10)
        self.place_switches()
        # Update switch state when the GUI is created
        self.update_switch_images()
class SeniorSwitchmanGUI(tk.Toplevel):
    def __init__(self, control_window):
        super().__init__(control_window)
        self.title("Senior Switchman GUI")
        self.geometry("800x800")
        self.control_window = control_window
        self.canvas = tk.Canvas(self, width=700, height=700)
        self.canvas.pack()
        try:
            self.house_img = tk.PhotoImage(file=r"C:\Users\Lenovo\Pictures
              \CHANGE 1.png")
            self.house_image_id = self.canvas.create_image(350, 350,
```

```
            image=self.house_img)
        except tk.TclError as e:
            print("Error loading house image:", e)
        try:
            self.switch_on_img = tk.PhotoImage(file=r"C:\Users\Lenovo\Pictures
              \MAIN SWITCH ON.png").subsample(7)
            self.switch_off_img = tk.PhotoImage(file=r"C:\Users\Lenovo\Pictures
              \MAIN SWITCH OFF.png").subsample(7)
        except tk.TclError as e:
            print("Error loading switch images:", e)
        self.switch_buttons = []
        for i in range(16):
            switch_button = tk.Button(self, image=self.switch_off_img,
              command=lambda idx=i: self.toggle_switch(idx))
            self.switch_buttons.append(switch_button)
        self.back_button = tk.Button(self, text="Back",
          command=self.back_to_control_window)
        self.back_button.place(x=10, y=10)
        self.place_switches()
        # Update switch state when the GUI is created
        self.update_switch_images()
    def place_switches(self):
        switch_size = 70
        house_center_x = (350 + 250) / 2
        house_center_y = (350 + 250) / 2

        switch_positions = [
            (house_center_x - switch_size * 3, house_center_y - switch_size *
              3),
            (house_center_x - switch_size, house_center_y - switch_size * 3),
            (house_center_x + switch_size, house_center_y - switch_size * 3),
            (house_center_x + switch_size * 3, house_center_y - switch_size *
              3),

            (house_center_x - switch_size * 3, house_center_y - switch_size),
            (house_center_x - switch_size, house_center_y - switch_size),
            (house_center_x + switch_size, house_center_y - switch_size),
            (house_center_x + switch_size * 3, house_center_y - switch_size),

            (house_center_x - switch_size * 3, house_center_y + switch_size),
            (house_center_x - switch_size, house_center_y + switch_size),
            (house_center_x + switch_size, house_center_y + switch_size),
            (house_center_x + switch_size * 3, house_center_y + switch_size),

            (house_center_x - switch_size * 3, house_center_y + switch_size *
              3),
            (house_center_x - switch_size, house_center_y + switch_size * 3),
            (house_center_x + switch_size, house_center_y + switch_size * 3),
            (house_center_x + switch_size * 3, house_center_y + switch_size *
```

```
                3),
        ]

        for i, (x, y) in enumerate(switch_positions):
            self.switch_buttons[i].place(x=x, y=y)

    def toggle_switch(self, switch_number):
        self.control_window.execute_commands([f"ON {switch_number + 1}"])
        self.update_switch_images()

    def update_switch_images(self):
        # Update the switch images based on the current state of the switches
        for i, button in enumerate(self.switch_buttons):
            if self.control_window.sense_state(i + 1):
                button.config(image=self.switch_on_img)
            else:
                button.config(image=self.switch_off_img)

    def back_to_control_window(self):
        self.control_window.switchman_gui_window = None
        self.destroy()
class SwitchStateWindow(tk.Toplevel):
    def __init__(self, switch_states):
        super().__init__()
        self.title("Switch States")
        self.geometry("1200x1200")

        self.switch_states = switch_states

        self.label = tk.Label(self, text="Switch States:")
        self.label.pack()

        self.text_area = tk.Text(self, height=10, width=30)
        self.text_area.pack()

        self.update_switch_states()

    def update_switch_states(self):
        self.text_area.delete("1.0", tk.END)
        for switch_number, state in self.switch_states.items():
            state_str = "ON" if state else "OFF"
            self.text_area.insert(tk.END, f"Switch {switch_number}: {state_str}
              \n")
class StartWindow(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Senior Switchman Start Window")
        self.geometry("1300x1000")
```

```
        try:
            self.start_bg_img = tk.PhotoImage(file=r"")
            self.start_bg_label = tk.Label(self, image=self.start_bg_img)
            self.start_bg_label.place(x=0, y=0, relwidth=1, relheight=1)
        except tk.TclError as e:
            print("Error loading start background image:", e)

        self.label = tk.Label(self, text="Welcome to Senior Switchman", font=
          ("Elephant", 30))
        self.label.pack(pady=50)

        self.play_welcome_message()

        self.start_button = tk.Button(self, text="Start",
          command=self.open_control_window)
        self.start_button.pack()
    def play_welcome_message(self):
        welcome_message = "Welcome to Senior Switchman"
        tts = gTTS(welcome_message)
        tts.save("welcome_message.mp3")
        pygame.mixer.init()
        pygame.mixer.music.load("welcome_message.mp3")
        pygame.mixer.music.play()
    def open_control_window(self):
        self.destroy()
        control_window = ControlWindow()
        control_window.mainloop()
if __name__ == "__main__":
    start_window = StartWindow()
    start_window.mainloop()
```