```
import tkinter as tk
from tkinter import ttk
import tkintermapview
from shapely.geometry import Polygon, Point ,LineString
import math
import pygame
from PIL import Image, ImageTk
import pandas as pd
import pyperclip  # To handle clipboard operations
from datetime import datetime
import os
import sys
pygame.mixer.init()

# Load audio files
sound_safe_zone = pygame.mixer.Sound('safe_zone.mp3')
sound_danger_zone = pygame.mixer.Sound('danger_zone.mp3')
# sound_destination = pygame.mixer.Sound('Destination.mp3')

# Define global variables
path_coords = []
selected_speed = 500
is_moving = False
update_interval = 1000
paused = False
factor = 0
step = 0
red_path = []
last_status = None
status_popup = None
last_direction = None
direction_popup = None
last_status = None
last_zones = set()

# Function to load danger zones from Excel file with validation
def load_danger_zones(file):
    danger_zones = []
    df = pd.read_excel(file)

    for index, row in df.iterrows():
        nation = row['Nation']
        coords = []
        i = 1

        while True:
            lat_key = f'Lat{i}'
            lon_key = f'Lon{i}'
            if lat_key in row and lon_key in row:
                lat = row[lat_key]
                lon = row[lon_key]

                if pd.isna(lat) or pd.isna(lon):  # Stop if either lat or lon is empty
                    break
                coords.append((lat, lon))
                i += 1
            else:
                break

        # Check if the number of coordinates is valid
        if len(coords) < 3 or len(coords) > 16:
            # Show pop-up message
            error_popup = tk.Tk()
            error_popup.title("Error")
            tk.Label(error_popup, text="Error: Count of Lat lon pair must be min 3 and
max 16!", font=("Arial", 14),fg='red').pack(pady=20, padx=20)
            tk.Button(error_popup, text="Exit", command=sys.exit).pack(pady=10)
            error_popup.mainloop()
            return []

        danger_zones.append((nation, coords))

    return danger_zones

def write_date(filename="latlon.txt"):
    global today_date
    today_date = datetime.now().strftime("%Y-%m-%d")
    desktop_path = os.path.join("C:\\Users\\rsneh\\Desktop", filename)
    with open(desktop_path, 'a') as file:
        file.write("Date : " +today_date)
        file.write("\n__________________________\n")


def write_safelatlon_to_file(coords, filename='latlon.txt'):
    desktop_path = os.path.join("C:\\Users\\rsneh\\Desktop", filename)
    with open(desktop_path, 'a') as file:
        for coord in coords:
            file.write(f"\n Safe zone : {coord[0]} {coord[1]}")


def write_dangerlatlon_to_file(coords, filename='latlon.txt'):
    desktop_path = os.path.join("C:\\Users\\rsneh\\Desktop", filename)
    with open(desktop_path, 'a') as file:
        for coord in coords:
            file.write(f"\n Danger zone : {coord[0]} {coord[1]}")


# Create the initial input window
def show_input_window():
    def close_input_window():
        input_window.destroy()
        show_map_window()

    input_window = tk.Tk()
    # Heading
    heading = tk.Label(input_window, text="Aerosafe Navigator", font=("Book
Antiqua", 25, "bold"), fg="midnight blue")
    heading1 = tk.Label(input_window, text="An interactive Geospatial Path
Simulation with Danger Zone Detection", font=("Book Antiqua", 15), fg="midnight
blue")
    heading.pack(pady=15)
    heading1.pack(pady=15)

    img = Image.open("plane.png")
    img = img.resize((800, 600))
    photo = ImageTk.PhotoImage(img)
    image_label = tk.Label(input_window, image=photo)
    image_label.photo = photo
    image_label.pack(pady=10)

    input_window.title("Version 1.0")

    # Close the input window after 4 seconds and show the map window
    input_window.after(4000, close_input_window)
    input_window.mainloop()

# Show the main map window
def show_map_window():
    global path_coords, selected_speed, is_moving, update_interval, paused,
direction_popup
    global factor, step, red_path, last_status, status_popup
    write_date()
    danger_zones = load_danger_zones('zone.xlsx')

    # Create tkinter window
    root_tk = tk.Tk()
    root_tk.geometry(f"{1000}x{700}")
    root_tk.title("Version 1.0")

   # Load the airplane image (adjust the path to your image file)
    airplane_image = tk.PhotoImage(file="plane.png")
    airplane_marker = None

    # Create map widget
    map_widget = tkintermapview.TkinterMapView(root_tk, width=1000, height=700,
corner_radius=0)
    map_widget.pack(fill="both", expand=True)

    def polygon_click(polygon):
        print(f"polygon clicked - text: {polygon.name}")

    # Set marker for Karnataka
    map_widget.set_zoom(7)

    # Define and draw danger zones
    for nation, coords in danger_zones:
        map_widget.set_polygon(coords, outline_color="red", fill_color="red")

    # Convert danger zones to polygons for Shapely
    danger_polygons = [Polygon([(lon, lat) for lat, lon in coords]) for _, coords in
danger_zones]

    # Function to check if point is inside any danger zone or impact zone
    def check_location(lat, lon):
        point = Point(lon, lat)
        is_inside_danger = any(point.within(poly) for poly in danger_polygons)
        return is_inside_danger

    # Function to create a path with multiple points
    def create_path(coords):
        return coords



    # Function to draw path on the map
    def draw_path(path_coords):
        map_widget.set_path(path_coords, color="blue")

    # Function to handle map clicks
    def on_map_click(lat, lon):
        global path_coords
        path_coords.append((lat, lon))
        map_widget.set_marker(lat, lon, text=f"Point {len(path_coords)}")
        draw_path(path_coords)


    danger_polygons = [(nation, Polygon([(lon, lat) for lat, lon in coords])) for nation,
coords in danger_zones]

    def check_overlapping_zones(lat, lon):
        point = Point(lon, lat)
        overlapping_zones = [nation for nation, poly in danger_polygons if
point.within(poly)]
        return overlapping_zones


    def move_line():
```

```
    global path_coords, selected_speed, is_moving, update_interval, paused
    global factor, step, red_path, last_status, status_popup, last_direction,
direction_popup, airplane_marker

    filename = 'latlon.txt'
    desktop_path = os.path.join("C:\\Users\\rsneh\\Desktop", filename)

    if not path_coords or len(path_coords) < 2:
        print("Not enough path coordinates.")
        return

    def interpolate(start, end, factor):
        return start + factor * (end - start)

    def interpolate_point(p1, p2, factor):
        return (interpolate(p1[0], p2[0], factor), interpolate(p1[1], p2[1], factor))

    def update_line():
        global factor, step, red_path, last_status, status_popup, last_direction,
direction_popup, airplane_marker ,last_zones

        start_point = path_coords[step]
        end_point = path_coords[step + 1]
        intermediate_point = interpolate_point(start_point, end_point, factor)
        red_path.append(intermediate_point)

        airplane_img = Image.open("jet.png")
        resized_airplane_img = airplane_img.resize((60, 60), Image.LANCZOS)
        airplane_photo = ImageTk.PhotoImage(resized_airplane_img)

        # Draw the red path
        if len(red_path) > 1:
            try:
                map_widget.set_path(red_path, color="red")
            except Exception as e:
                print(f"Error drawing path: {e}")

        if len(red_path) == 1:  # Create the airplane marker at the start
            airplane_marker = map_widget.set_marker(intermediate_point[0],
intermediate_point[1], icon=airplane_photo)
        else:  # Update the airplane marker position
            airplane_marker.set_position(intermediate_point[0], intermediate_point[1])

        # Check zone status
        overlapping_zones = check_overlapping_zones(intermediate_point[0],
intermediate_point[1])

        if overlapping_zones:
                if last_status != "Impact Zone" or set(overlapping_zones) != last_zones:
                        status = f"You have entered the danger zones of {',
'.join(overlapping_zones)}!"
                        if status_popup:
                            status_popup.destroy()
                        status_popup = tk.Toplevel(root_tk)
                        status_popup.title("Zone Status")
                        tk.Label(status_popup, text=status, font=("Arial", 14)).pack(pady=20,
padx=20)
                        sound_danger_zone.play()
                        last_status = "Impact Zone"
                        last_zones = set(overlapping_zones)  # Update last zones
                    write_dangerlatlon_to_file([intermediate_point])
            else:
                status = "You are Safe now"
                if last_status != "Safe Zone":
                    if status_popup:
                        status_popup.destroy()
                    status_popup = tk.Toplevel(root_tk)
                    status_popup.title("Zone Status")
                    tk.Label(status_popup, text=status, font=("Arial", 14)).pack(pady=20,
padx=20)
                    sound_safe_zone.play()
                    last_status = "Safe Zone"
                    last_zones = set()
                write_safelatlon_to_file([intermediate_point])

            # Calculate direction
            # direction_angle = math.degrees(math.atan2(end_point[0] - start_point[0],
end_point[1] - start_point[1]))
            # if last_direction != direction_angle:
            #     if direction_popup:
            #         direction_popup.destroy()
            #     direction_popup = tk.Toplevel(root_tk)
            #     direction_popup.title("Flight Direction")
            #     tk.Label(direction_popup, text=f"Current Direction:
{direction_angle:.2f}°", font=("Arial", 14)).pack(pady=20, padx=20)
            #     last_direction = direction_angle


            factor += 0.01
            if factor >= 1:
                factor = 0
                step += 1

            if step < len(path_coords) - 1 and not paused:
                root_tk.after(update_interval, update_line)
            elif step >= len(path_coords) - 1:
                # sound_destination.play()
                with open(desktop_path,'a') as file:
```

```
                file.write(f"\n--------------------------------------------------------------------
---------------------------------------------------------------------------------------------\n")

        # Reset and start simulation
        if not is_moving:
            is_moving = True
            factor = factor if not paused else 0
            red_path = red_path if not paused else []
            print("Starting simulation...")

            # Adjust update interval based on speed
            update_interval = int(50000 / selected_speed)
            update_line()
        else:
            print("Simulation is already running.")


    def clear_path():
        global path_coords, is_moving, factor, step, red_path, last_status,
airplane_marker
        airplane_marker=None
        # Clear the list of path coordinates
        path_coords.clear()

        # Reset simulation control variables
        is_moving = False
        factor = 0
        step = 0
        red_path = []
        last_status = None

        # Remove the airplane marker if it exists
        if airplane_marker:
            map_widget.delete(airplane_marker)
            airplane_marker = None

        # Clear the map
        map_widget.delete_all_marker()
        map_widget.delete_all_path()

    def undo_path():
        global path_coords
        if len(path_coords) > 0:  #this sees if there is atleast 1 path to remove
            path_coords.pop()  # Remove the last coordinate from the path
            map_widget.delete_all_marker()  # Remove all markers
            map_widget.delete_all_path()  # Remove all paths
            # Redraw remaining markers and path
            for idx, (lat, lon) in enumerate(path_coords):
                map_widget.set_marker(lat, lon, text=f"Point {idx + 1}")  #after each path
is removed the marker points are adjusted accordingly
```

```
            if len(path_coords) > 1:
                draw_path(path_coords)  # Redraw the blue path with remaining
coordinates
        else:
            print("No more markers to undo.")

    def on_load_lat_lon_button_click():
        global path_coords

        # Use clipboard content for coordinates
        clipboard_content = pyperclip.paste().strip()  # Clean any extra spaces or new
lines  and pasteis to copy the content of clipboard
        try:
            # Expecting coordinates in the format "lat lon"
            lat, lon = map(float, clipboard_content.split()) # clipboard content is split
based on space or new line and it is stored inside lat lon
            path_coords.append((lat, lon))
            map_widget.set_marker(lat, lon, text=f"Marker {len(path_coords)}")
            draw_path(path_coords)

            # Clear the clipboard content after loading
            pyperclip.copy("")
        except ValueError:
            print("Clipboard content is not in the correct format.")

    def start_simulation():
        global paused
        if not is_moving and path_coords:
            paused = False
            print("Starting simulation...")
            move_line()
        else:
            print(is_moving, path_coords)
            print("Cannot start simulation: Already running or no path coordinates.")

    def stop_simulation():
        global is_moving, paused
        if is_moving:
            print("Stopping simulation...")
            paused = True  # Ensure paused is reset
            is_moving = False
            print("Simulation stopped.")

    # Buttons
    button_frame = tk.Frame(root_tk)
    button_frame.pack(fill='x', side='bottom')

    move_button = tk.Button(button_frame, text="        Start        ",
command=start_simulation,font=15,bg='lightblue',fg='black')
    move_button.pack(side='left', padx=5, pady=5)
    stop_button = tk.Button(button_frame, text="       Stop       ",
command=stop_simulation,font=15,bg='lightblue',fg='black')
    stop_button.pack(side='left', padx=5, pady=5)

    clear_button = tk.Button(button_frame, text="      Clear Path      ",
command=clear_path,font=15,bg='lightblue',fg='black')
    clear_button.pack(side='left', padx=5, pady=5)

    undo_button = tk.Button(button_frame, text="      Undo Path      ",
command=undo_path,font=15,bg='lightblue',fg='black')
    undo_button.pack(side='left', padx=5, pady=5)

    load_lat_lon_button = tk.Button(button_frame, text="      Load Lat Lon       ",
command=on_load_lat_lon_button_click,font=15,bg='lightblue',fg='black')
    load_lat_lon_button.pack(side='left', padx=5, pady=5)

     # Speed selection dropdown
    speed_label = tk.Label(root_tk, text="Select Speed:",font=10)
    speed_label.pack(side='left', padx=5, pady=5)

    speed_options =
[50,100,150,200,250,300,350,400,450,500,550,600,650,700,750,800,850,900,950,100
0,1050,1100,1150,1200,1250,1300,1350,1400,1450,1500,1550,1600,1650,1700,1750,
1800,1850,1900,1950,2000]
    speed_dropdown = ttk.Combobox(root_tk, values=speed_options)
    speed_dropdown.current(3)  # Set default value
    speed_dropdown.pack(side='left', padx=5, pady=5)
    def on_speed_select(event):
        global selected_speed
        selected_speed = int(speed_dropdown.get())

    speed_dropdown.bind("<<ComboboxSelected>>", on_speed_select)
    # Bind map click event
    map_widget.bind("<Button-1>", lambda e: on_map_click(map_widget.get_lat(e.x,
e.y), map_widget.get_lon(e.x, e.y)))

    root_tk.mainloop()

# Start the application with input window
show_input_window()
```