Emergency Routing System Documentation

1. Project Overview

The Emergency Routing System is designed to provide optimized routing for emergency services in Bangalore by considering real-time traffic conditions. This system utilizes the OpenStreetMap (OSM) data, NetworkX for graph analysis, and matplotlib for visualization. The goal is to enable quicker response times for emergency services by computing the shortest path between an origin and a destination while accounting for traffic delays.

Key Features:

- Download and visualize road networks for Bangalore.
- Compute the shortest path based on traffic conditions.
- Simulate real-time traffic delays on routes.
- Visualize the adjusted route for easy comprehension.

2. Requirements

2.1 Software Requirements

- Python 3.x: Ensure you have Python installed on your machine.
- Libraries: Install the following Python libraries:
  - osmnx
  - networkx
  - matplotlib
  - geopandas

2.2 Installation Command

pip install osmnx networkx matplotlib geopandas

3. Setup Instructions

3.1 Clone the Repository

If you are using a version control system, clone the repository containing the project files.

```
git clone https://github.com/your_username/emergency_routing_system.git
cd emergency_routing_system
```

3.2 Import Libraries

At the beginning of your Python script, import the necessary libraries:

```
import osmnx as ox
import networkx as nx
import matplotlib.pyplot as plt
import random
```

4. Project Workflow

4.1 Step 1: Downloading the Road Network

Use OSMnx to download the driving network for Bangalore:

```
place_name = 'Bangalore, India'
ox.config(log_console=True)
G = ox.graph_from_place(place_name, network_type='drive')
G = ox.simplify_graph(G)
```

4.2 Step 2: Define Origin and Destination Points

Geocode the origin and destination addresses to find their respective coordinates:

```
origin_address = 'Kempegowda International Airport, Bangalore, India'
destination_address = 'Bangalore Palace, Bangalore, India'
origin_point = ox.geocode(origin_address)
destination_point = ox.geocode(destination_address)
```

4.3 Step 3: Find Nearest Nodes

Identify the nearest nodes in the graph for the defined origin and destination:

```
origin_node = ox.distance.nearest_nodes(G, origin_point[1], origin_point[0])
destination_node = ox.distance.nearest_nodes(G, destination_point[1], destination_point[0])
```

4.4 Step 4: Simulate Traffic Conditions

Add a traffic delay to each edge and adjust the weights accordingly:

```
for u, v, k, data in G.edges(keys=True, data=True):
    data['traffic_delay'] = random.uniform(1.0, 1.5)  # Random delay between 0% to 50%
    data['adjusted_weight'] = data['length'] * data['traffic_delay']
```

4.5 Step 5: Compute the Shortest Path

Calculate the shortest path based on the adjusted weights:

```
route_with_traffic = nx.shortest_path(G, origin_node, destination_node, weight='adjusted_weight')
route_length_with_traffic = nx.shortest_path_length(G, origin_node, destination_node, weight='adjusted_weight')
print(f'Adjusted route length with traffic: {route_length_with_traffic / 1000:.2f} km')
```

4.6 Step 6: Visualize the Route

Finally, visualize the computed route on the map:

```
fig, ax = ox.plot_graph_route(G, route_with_traffic, node_size=0, bgcolor='white',
                              route_color='blue', route_linewidth=3, show=False, close=False)
plt.title('Adjusted Route Considering Simulated Traffic')
plt.show()
```

5. Usage Instructions

1. Run the script: Execute the Python script to compute and visualize the emergency route.

2. View output: The console will display the adjusted route length, and a plot will show the route on the map.

Example Console Output:

Adjusted route length with traffic: X.XX km

Example Visualization:

[Example Route Visualization](link_to_visualization_image)

6. Future Improvements

- Real-time Traffic Data Integration: Integrate with APIs to fetch live traffic data instead of using simulated delays.
- User Interface: Develop a web-based user interface for easier interaction.
- Incident Reporting System: Allow users to report incidents that can affect routing.
- Performance Optimization: Implement caching strategies for frequently used routes.
- Mobile Application: Extend functionality to a mobile platform for on-the-go accessibility.

7. Conclusion

The Emergency Routing System aims to enhance the responsiveness of emergency services in Bangalore by optimizing routing based on real-time conditions. With further development, this project has the potential to significantly improve emergency response times and service efficiency.