**onMain()**

**{**

Create Our Map Variable.Greeting Message, And display the menu.

The menu will have several options. Wohi process of making a main while loop until our. Choice variable does not come out as a certain value. The other options will be to update. Our map variable. Will be updated via this while loop. You add / remove roads , find shortest path ,update traffic , add emergency routes , visualize our graph , and exit. These are our options. Each one leading to a function.

**}**

**GraphCreation()**

**{**

Create an empty graph variable to represent the city as nodes (intersections) and edges (roads).

Load the road network data from a file, which includes intersections and travel times.

For each entry in the file, add an edge to the graph with its weight (travel time or congestion level).

This function will handle both the initial graph creation and dynamic updates later.

**}**

**LoadVehicles()**

**{**

Read the "vehicles.csv" file to load details about each vehicle (ID, start point, end point).

Store this data in a list or array, where each element represents a vehicle object.

This function ensures that vehicle data is ready for operations like route finding.

**}**

**LoadTrafficSignals()**

**{**

Open the "traffic\_signal\_timings.csv" file to get initial green time durations for traffic signals.

Store each signal with its intersection as the key in a hash table for quick access.

This function allows traffic signal conditions to be dynamically updated later.

**}**

**LoadEmergencyVehicles()**

**{**

Read the "emergency\_vehicles.csv" file to fetch details about emergency vehicles, including priority levels.

Store this information in a separate list, as these vehicles will be managed differently.

Emergency vehicles will use a specialized routing algorithm (like A\*) to ensure priority handling.

**}**

**LoadEmergencyVehicles()**

**{**

Read the "emergency\_vehicles.csv" file to fetch details about emergency vehicles, including priority levels.

Store this information in a separate list, as these vehicles will be managed differently.

Emergency vehicles will use a specialized routing algorithm (like A\*) to ensure priority handling.

**}**

**AddRoad()**

**{**

Prompt the user to enter start and end intersections for the road.

Ask for the travel time or congestion level to assign as the edge weight.

Update the graph by adding the new road as an edge between the specified intersections.

**}**

**RemoveRoad()**

**{**

Ask the user for the start and end intersections of the road to be removed.

Check if the road exists in the graph.

If it does, delete the edge from the graph structure and confirm the removal.

**}**

**FindShortestPath()**

**{**

Ask the user to specify a vehicle ID to determine its start and end points.

Use the Dijkstra algorithm to compute the shortest path between these points in the graph.

Display the path and the total travel time back to the user.

**}**

**UpdateTrafficSignals()**

**{**

Prompt the user to enter the intersection where the signal timing needs updating.

Ask for the new green light duration for that intersection.

Update the value in the traffic signals hash table and confirm the changes to the user.

**}**

**HandleEmergencyVehicles()**

**{**

Ask the user to specify an emergency vehicle ID.

Retrieve the start and end points for this vehicle.

Use the A\* algorithm to calculate the fastest route while considering a heuristic for priority.

Override normal traffic signals along the route to clear the path for the emergency vehicle.

**}**

**VisualizeGraph()**

**{**

Display the current state of the graph, including all intersections and roads, along with their weights.

For a text-based visualization, list all nodes and edges.

For a graphical visualization, show the network visually if supported.

**}**