Data Structures and Algorithms

Project: Ambulance Management System

1. Overview

This project aims to simulate the operations of an **Ambulance Management System** serving multiple hospitals.
The system automates the allocation of ambulance cars to patients based on defined priorities and dynamic events that occur during the simulation.
The project focuses on the design of an event-driven simulation using efficient data structures to model real-world entities and interactions.

2. Simulation Concept

The simulation runs over discrete **timesteps**, where each timestep represents a unit of time (e.g., one minute).
At each timestep, several actions may occur:

- New patient requests.
- Ambulance car assignments.
- Pickups and hospital returns.
- Request cancellations or reassignments.

After all actions for the current timestep are processed, the simulation proceeds to the next one.

3. System Entities

3.1 Patients

Each patient in the system has the following information:

- **Request Time (QT):** Time the patient requests the service.
- **Pickup Time (PT):** Time the patient is picked up by a car.

- **Nearest Hospital ID:** The hospital closest to the patient.
- **Distance to Nearest Hospital:** In meters.
- **Patient Type:**
    - **EP** — Emergency Patient (highest priority).
    - **SP** — Special Patient (requires special equipment).
    - **NP** — Normal Patient.

3.2 Hospitals

- The system includes **H hospitals**, each identified by an ID (1..H).
- A **distance matrix** defines the distance between every pair of hospitals.
- Each hospital owns a set of ambulance cars.

The distance matrix is loaded from the input file at system startup and defines inter-hospital routing.

3.3 Ambulance Cars

Each car has:

- **Type:**
    - **SC** — Special Car (equipped for SP patients).
    - **NC** — Normal Car.
- **Speed:** Number of meters per timestep.
- **Status:**
    - **Ready** — waiting at hospital.
    - **Assigned** — moving to pick up a patient.
    - **Loaded** — carrying a patient back to hospital.

Each car can carry **only one patient** at a time.

4. Key Time Definitions

- **QT (Request Time)** – when the patient requests service.
- **AT (Assignment Time)** – when a car is assigned.

- **PT (Pickup Time)** – when the patient is picked up.
- **FT (Finish Time)** – when the patient reaches the hospital.
- **WT (Waiting Time)** = PT − QT
- **Car Busy Time:** Total time the car is not free.

5. Car Assignment Algorithm

Each hospital manages a queue of patient requests according to the following rules:

1. **Emergency Patients (EP):**
    - Always served first.
    - Stored in a priority queue based on case severity.
    - Assign to **NC** cars first; if none are available, assign to **SC** cars.
    - If no cars are available, the hospital reports to the central system, which redirects the patient to another hospital with:
        - The **shortest EP waiting list**, and
        - If tied, the **nearest hospital**.
2. **Special Patients (SP):**
    - Served using **SC** cars only.
    - Managed on a **First Come First Serve (FCFS)** basis.
    - Wait until an SC car is available.
3. **Normal Patients (NP):**
    - Served using **NC** cars only.
    - FCFS scheduling.
    - May **cancel** their request anytime before pickup (as specified in the input file).

If a patient request cannot be served at the current timestep, it remains in the waiting list for the next timestep.

6. Input and Output File Formats

6.1 Input File

```
H
SC_Speed NC_Speed
<Distance Matrix (H×H)>
<SCnum NCnum for each hospital>
R
<Request lines>
C
<Cancellation lines>
```

Request Line Format:

```
TYPE QT PID HID DIST [SEVERITY]
```

Examples:

```
NP 3 1 2 159
SP 3 2 1 588
EP 12 3 4 433 5
```

Cancellation Line Format:

```
CT PID HID
```

6.2 Output File

Each completed patient record is written as:

```
FT PID QT WT
```

At the end of the file, the following statistics must appear:

- Total number of patients (and counts per type)
- Total number of hospitals
- Total number of cars (and counts per type)
- Average waiting time

- Percentage of EPs not served by their home hospital
- Average car busy time
- Average utilization = (Average busy time / Total simulation time) × 100%

7. Program Modes

Interactive Mode

The system displays simulation details at each timestep:

- Current timestep
- Status of each hospital
- Waiting and in-service patients
- Outgoing and returning cars

The user presses any key to advance to the next timestep.

Silent Mode

No intermediate output; only the final output file is produced.

8. Suggested Class Design

- **Hospital** — Manages cars and patient queues.
- **Car** — Represents ambulance car attributes and states.
- **Patient** — Stores patient data.
- **SystemManager / Simulator** — Controls global time, manages events, and logs statistics.
- **IOManager** — Handles file input/output.

9. Recommended Data Structures

Entity	Recommended DS	Notes
Emergency patients	Priority Queue	Sorted by severity
Special patients	Queue	FCFS
Normal patients	Queue	FCFS with possible cancellation
Cars	Queue / List	Track available, assigned, and loaded
Hospitals	Array / Vector	Fixed number
Distance Matrix	2D Array	Loaded at startup
Completed Patients	List	For output ordering by FT

10. Implementation Notes

- Use discrete-event simulation logic.
- Implement modular classes with clear responsibility.
- Avoid global variables; pass references or pointers.
- Handle patient cancellation carefully to prevent dangling references.
- Always maintain sorted order for output by Finish Time (FT).

11. Sample Scenario

At timestep 12:

- Hospital 2 receives one **EP** and one **NP** request.
- One **NC** car is free → assigned to the EP immediately.
- The NP request is queued until another NC car becomes available.
- Once the EP patient is delivered, the car's status changes from *Loaded* → *Ready*.

12. Learning Outcomes

Through this project, students will:

- Understand the design of **event-driven simulations**.
- Apply **priority queues** for managing critical requests.
- Use **multiple queues** for different patient types.
- Implement and manage **state transitions** for dynamic entities.
- Handle **file input/output** with structured formats.
- Analyze **system performance** using computed statistics.
- Develop modular, reusable code using **object-oriented principles**.