Mohamed bin Zayed University of Artificial Intelligence

Individual Assignment 1 - POI Management System

AI1030 - Python Programming

1 Introduction

A **Point of Interest (POI)** is a specific location or feature that someone may find useful, interesting, or worth noting. POIs are often used in maps, navigation systems, and travel guides. Examples include natural landmarks, cultural sites, services and facilities, and many more. In short, POIs help people identify and navigate places of significance, whether for travel, daily needs, or exploration.

In this assignment, you will develop a **POI Management System** that can store information about POIs and also information about visitors. A user may interact with your application using a **simple text-based menu** that you need to implement.

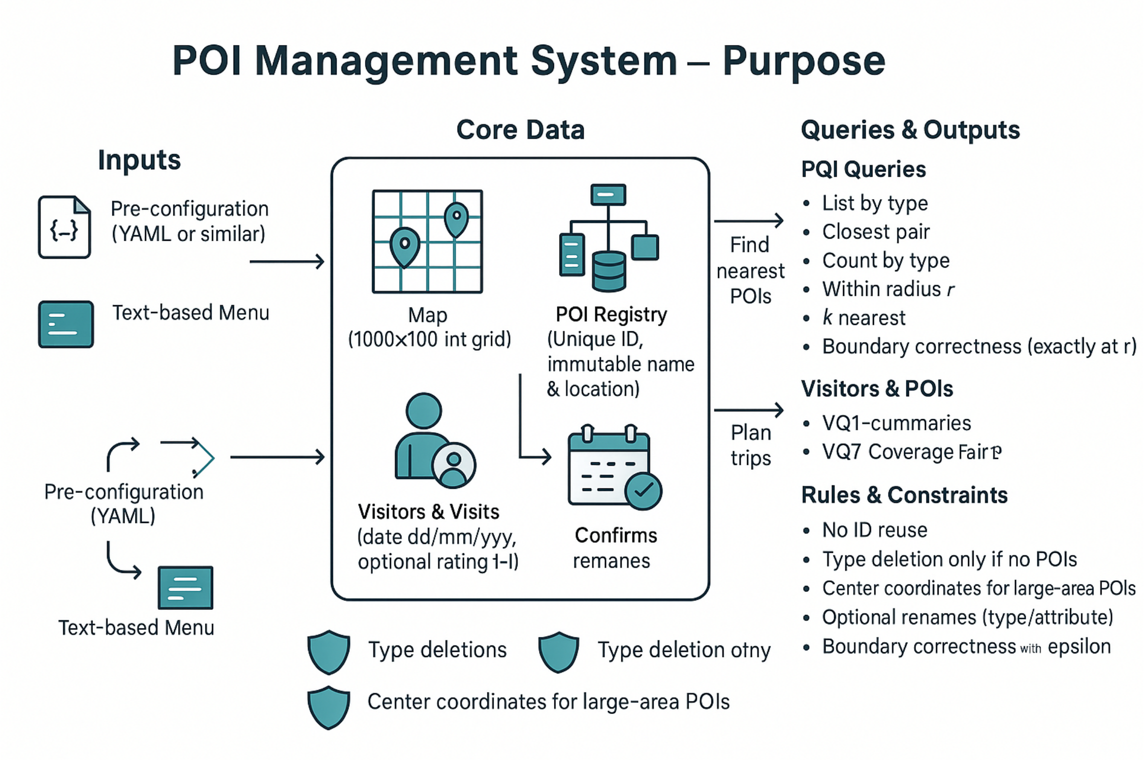
POI Management System – Purpose
Inputs
Pre-configuration (YAML or similar)
Text-based Menu
Pre-configuration (YAML)
Text-based Menu
Core Data
Map (1000×100 int grid)
POI Registry (Unique ID, immutable name & location)
Visitors & Visits (date dd/mm/yyy, optional rating 1-I)
Confirms remanes
Type deletions
Type deletion otny
Center coordinates for large-area POIs
Find nearest POIs
Plan trips
Queries & Outputs
PQI Queries
List by type
Closest pair
Count by type
Within radius r
k nearest
Boundary correctness (exactly at r)
Visitors & POIs
VQ1-cummaries
VQ7 Coverage Fair
Rules & Constraints
No ID reuse
Type deletion only if no POIs
Center coordinates for large-area POIs
Optional renames (type/attribute)
Boundary correctness with epsilon

Figure 1: POI Management System — Purpose. The system ingests pre-configuration and menu input, maintains map/POI/visitor data with key invariants (no ID reuse, type deletion constraints, center coordinates for large-area POIs), and answers POI/visitor queries including boundary correctness and coverage fairness.

2 Initialization & Pre-configuration (config file)

- On startup, the application must **optionally load an initial configuration** (e.g., POI types, attributes, POIs, visitors) **from a human-readable configuration file (YAML or a similar structured text format)**.
- The exact file name/location/structure is **intentionally unspecified**; design your loader to be robust to missing/extra fields and to **validate inputs** against the rules below (report and handle any issues).

3 Data

- **Map:** fixed space of a 1000×1000 grid, all coordinates are integers.
- **POI (point of interest):** a location on the map with a specific name (e.g., Zayed International Airport) and a collection of features (attributes) that characterize the POI. Different POI types may define different attributes. Each POI is assigned exactly one POI type. Example types include (but are not limited to): archaeological site, museum, park, restaurant, hotel, beach, forest, lake, aquarium, educational institution, medical center.
  Clarification (large-area POIs): if a POI covers a large area (e.g., a forest or beach), its **location refers to the center** (as a single location on the grid).
- **POI types & attributes:** The system must allow adding and deleting POI types, and adding or deleting attributes for a POI type.
  Constraint: a POI type can be deleted **only if no POIs** of that type exist in the system.
- **POI identifiers:** Each POI has a **unique identifier** that is **never reused**. Deleting POI id $= X$ does not permit any future POI to take id $= X$. Identifiers, names, and locations are **immutable** (not updatable).
- **Visitors:** Each visitor has a unique id, name, and nationality. Store, per visitor, the POIs they have visited. A rating (integer 1–10) is optional for each visit. Record the visit date in `dd/mm/yyyy`. (No need to support deletion of visitors.)

Clarification: your solution should handle insertions of new POIs, deletions of existing POIs and insertions of new Visitors and visits to POIs. Deletions of Visitors and visits are not required.

Extensions (optional): rename operations

- **Renaming Attributes:** Support of renaming an attribute name within a POI type. If implemented, all existing POIs of that type should be migrated accordingly (unmatched attributes may be set to `None` or dropped—state the policy).
- **Renaming a POI Type:** Support of renaming a POI type. If implemented, existing POIs should reflect the new type name consistently.
  If you implement these, document the policies and any migration steps taken.

4 Queries for POIs

Let the Euclidean distance between two coordinates (x_1, y_1) and (x_2, y_2) be

$$d\big((x_1, y_1), (x_2, y_2)\big) \;=\; \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\,.$$

1. For a specific POI type, list the POIs with all attribute values.
2. Find the pair of POIs (id, name, coordinates) with the **minimum pair-wise Euclidean distance**.
3. Print the **number of POIs per POI type**.
4. Given a map location $c_0 = (x_0, y_0)$ and a radius r (float number), list all POIs within distance **at most** r from c_0 (id, name, coordinates, type and distance from c_0).
5. Given $c_0 = (x_0, y_0)$ and an integer k, list the k **POIs closest** to c_0 in increasing distance order (id, name, coordinates, type, and distance).
6. **Boundary correctness.**
   (a) For a given c_0 and radius r, **list POIs exactly at distance** r from c_0 (id, name, coordinates, type, and computed distance).
   (b) Describe and implement a **robust comparison** strategy for floating-point calculations so that POIs at the boundary are neither wrongly excluded nor duplicated (e.g., an epsilon-based equality check with a clearly stated $\varepsilon > 0$).
   (c) Use the **center coordinate** (or any other fixed reference location) for large-area POIs when computing the distance.

5 Queries for Visitors and POIs

1. For a specific visitor, list all the visited POIs (id, name, date).
2. Print the number of visitors per POI (POI id, number of visitors).
3. Print the number of POIs per visitor (visitor id, number of POIs).
4. List the k visitors (id, name) who have visited the largest number of POIs.
5. List the k POIs (id, name) that have the largest number of visitors.

7. **Coverage fairness.** Given thresholds m and t, list the visitors who have visited at least m POIs across at least t distinct POI types. Output: (visitor id, name, nationality, total number of POIs visited, number of distinct POI types).

Counting Conventions (apply unless otherwise stated)

- **Distinct by default.** All phrases of the form "number of X per Y" are computed over *unique X*. Repeat visits by the same visitor to the same POI count at most once for these counts.
- **Visit event vs. coverage.** If a query explicitly asks for *total number of visits*, it counts every recorded visit event (multiplicity), including multiple visits to the same POI by the same visitor.
- **Visitor of a POI.** A visitor is considered a visitor of a POI if there is at least one recorded visit to that POI (rating optional).
- **Deterministic ordering.** When sorting by counts, break the ties by ascending id, then by name.

6 AI Usage Log

Include a single table with **only** the following two columns:

- **Prompt** — the exact text you sent to any AI tool.
- **Verification (method & result)** — how you verified the AI's output (e.g., unit tests, manual calculation, cross-check) **and** the outcome (accepted / modified / rejected, with a brief note if modified).

Unreported use of AI tools may result in mark deductions if you cannot demonstrate a clear understanding of your submission.

Template:

Prompt	**Verification (method & result)**
(paste exact prompt text)	*(e.g., Wrote unit tests for PQ5; 2/5 failed → fixed tie-break; re-ran: all pass → **modified**)*

7 Deliverables

7.1 Code (40%)

- A working **CLI application** in Python implementing all required operations and queries.
- **Config loader** (YAML or similar) with validation and clear error reporting.
- **Test suite**: at least 6 tests covering distance edge-cases (including boundary ε), type/attribute updates, visitor stats, and config validation.
- Code readability: modular structure, docstrings, and comments explaining key invariants (ID non-reuse, type deletion constraint).

7.2 Report (40%)

- **Design & Data Model**: entities, invariants and (if used) persistence format.
- **Edge Policies**: boundary ε, tie-break rules, counting rules (distinct vs. repeated visits), coordinate bounds.
- **Usage Guide**: sample menu session + sample configuration.
- **Reflection** (300–500 words): trade-offs and constraints that shaped the design.

7.3 Recorded Demo (20%)

- Duration: 8–10 minutes.
- Show the CLI, load a configuration, demonstrate core queries (PQ2, PQ4/5, VQ4/5) and edge-cases (boundary correctness, coverage fairness).

Bonus (up to +5%)

- Optional extensions (e.g., attribute/type rename with migration), plus 3 targeted tests.

8 Grading Rubric

Component	**Criteria**	**Points**
Code (40%)	- Functionality - Correctness - Code readability and comments - Includes 6 test cases	40
Report (40%)	- Description of the data model and the overall design - Explain how you handled edge policies - How to use the application (menus, etc.) - Reflection (300–500 words) on trade-offs and constraints - Professional structure and clarity	40
Demo (20%)	- 8–10 minutes, well-structured - Testing the CLI, load a configuration, demonstrate core queries (PQ2, PQ4/5, VQ4/5) and edge-cases (boundary correctness, coverage fairness) - Clear articulation of lessons learned	20
Extension (5%)	- Renaming of attributes - Renaming of POI types - Handling migration - Three targeted test cases	5
Total		**105**

9 Grade Bands

Grade	GPA	Score Range (%)	Performance Description
A+	4.0	95–100	Exceptional work, complete and polished in all aspects; interpreter, report, and presentation exceed expectations.
A	4.0	90–94	Excellent work, very strong across all components with only minor improvements possible.
A-	3.7	85–89	Very good work, complete with small gaps in detail, clarity, or testing.
B+	3.3	80–84	Good work, most requirements met with some issues in implementation or explanation.
B	3.0	75–79	Satisfactory work, functional but with gaps in coverage or clarity.
B-	2.7	70–74	Adequate work, significant issues in one component (e.g., interpreter or report).
C+	2.3	65–69	Passable work, limited testing or reflection, some incomplete elements.
C	2.0	60–64	Barely sufficient, multiple weaknesses across code, report, or presentation.
C-	1.7	55–59	Weak performance, incomplete work with major flaws but some evidence of effort.
D+	1.3	50–54	Very limited work, serious issues across most components, minimal demonstration of understanding.
D	1.0	40–49	Poor work, interpreter largely non-functional or report missing key sections.
F	0.0	0–39	Unsatisfactory, major requirements missing or not attempted.